NTNU – Trondheim
Norwegian University of
Science and Technology

Prototyping a RISC-V based SHMAC with Chisel

Sondre Lefsaker

December 2014

PROJECT
Department of Computer and Information Science
Norwegian University of Science and Technology

Supervisor: Magnus Jahre

Co-supervisor: Yaman Umuroglu

Problem Description

The Single-ISA Heterogeneous MAny-core Computer (SHMAC) project aims to develop an infrastructure for instantiating diverse heterogeneous architectures on FPGAs. A prototype has already been developed and a variety of co-processor tiles are being investigated. The SHMAC is currently using the proprietary ARM instruction set as the single ISA. However, hardware implementations of the ARM ISA introduce extra challenges in research and development, since there are no open implementations that support modern variants of the ARM ISA. To address these issues, alternative ISAs are being considered. RISC-V is a recently proposed open ISA that aims to facilitate computer architecture research for both academia and industry. It is designed to support different microarchitectural styles and extensions for customized accelerators.

The goal of this project is to evaluate the feasibility of the RISC-V ISA for SHMAC in a simulated environment. The simulator should support instantiating tiled layouts connected by a network-on-chip. A minimum of two tile types (RISC-V tile and memory tile) should be supported. If time permits, additional heterogeneous tile types may be developed and evaluated.

This thesis will be co-supervised by Yaman Umuroglu.

Abstract

Energy efficiency is now the main performance constraint in modern processors. There is an extensive amount of research going into new architectures with a focus on energy efficiency, both to increase performance and to reduce the amount of power usage. The Dark Silicon effect has lead to a paradigm shift in the design of computing systems, leading architects to design heterogeneous computers with specialized processors for different computing tasks. The SHMAC has already been developed, and is an infrastructure for creating heterogeneous systems of different processing and memory elements connected together with an on-chip interconnect network.

In this project, the RISC-V ISA is investigated and proposed as a replacement to the current ISA in which the SHMAC is currently based upon. Then, a new prototype of the SHMAC with Sodor processor cores based on RISC-V is developed with the Chisel hardware description language. Additionally, the SHMAC's network-on-chip is ported from Verilog to Chisel and a new memory tile is added to the system. In the end, a cycle-accurate simulator is used to evaluate and test two different instances of the new prototype. The prototype demonstrates that it is feasible to base further development of the SHMAC on the RISC-V ISA; this achieves the advantages of being able to utilize the ISA's extensibility and its open hardware implementations.

Contents

List of Tables

List of Figures

List of Abbreviations

CPU Central Processing Unit. 4, 6

CSR Control Status Register. 15

EECS Energy Efficient Computing Systems. 1

FPGA Field-Programmable Gate-Array. 1, 5

FPU Floating Point Unit. 11, 14

HDL Hardware Description Language. 3, 8, 22

HMP Heterogeneous Many-core Processor. 2

HTIF Host-target Interface. 15

IoT Internet of Things. 12

ISA Instruction Set Architecture. 1–4, 10–12, 20, 23, 24

MMU Memory Management Unit. 11

NoC Network on Chip. 3, 5, 13, 22

NUMA Non-Uniform Memory Access. 4, 5, 14, 22

PE Processing Element. 3, 4

SHMAC Single-ISA Heterogeneous MAny-core Computer. 1–5, 8–13, 16–24

SoC System on a Chip. 13

TLB Translation Lookaside Buffer. 11

UCB University of California, Berkeley. 8, 11, 12, 14

1 Introduction

Moores Law states that the number of transistors in a dense integrated circuit doubles approximately every second year. This law has been more or less fulfilled, as industry has been able to push the limits of transistor and processor design. Dennard Scaling states that while the size of the transistors gets smaller, their power density stays constant. This introduces a scaling problem that leads to what we know as the Dark Silicon Effect [8]. Dark Silicon refers to the parts of the processor that has to be shut down in order for it not to exceed the power budget.

1.1 The SHMAC Project

The Single-ISA Heterogeneous MAny-core Computer (SHMAC) is a research project initiated by the Energy Efficient Computing Systems (EECS) research group at the Faculty for Information Technology, Mathematics and Electrical Engineering at NTNU. EECS' research targets energy efficient computing systems across all abstraction layers.

The Dark Silicon Effect makes heterogeneous computing systems very likely to come, and SHMAC attempts to investigate some of the challenges that may be imposed by such a system. SHMAC is a tile-based architecture with a 2D-mesh interconnect, with each tile containing either a processor core, an accelerator, a memory module, or a combination of the different elements. The system is highly configurable, and the main goal of the project is to find different configurations that will maximize performance for an application with a fixed power budget. The architecture is described in greater detail in Section 2.2.3.

The SHMAC is currently using the Amber processor core, which is based on the ARMv4t Instruction Set Architecture (ISA). The ISA is protected under licenses that makes it ill-suited for an academic research project like SHMAC. This problem has led the project to investigate a different ISA for the SHMAC, namely RISC-V. In Section 2.4 it is argued for why RISC-V is a suitable replacement ISA.

1.2 The SHMAC Simulation Infrastructure

SHMACsim refers to a cycle-accurate simulation infrastructure previously developed for the SHMAC. As the development of SHMAC has steadily continued forward, the implementation of SHMACsim has remained the same. The two projects have diverged over time, and it is once again need for a simulation infrastructure for the SHMAC.

Being able to simulate hardware designs while they are being developed is important. Even though simulation is slower than hardware emulation on a Field-

Programmable Gate-Array (FPGA), it still allows for faster prototyping and testing of different hardware designs. It can prove to be very valuable for a project like SHMAC, where the purpose is to research new designs and architectures for Heterogeneous Many-core Processor (HMP) systems.

1.3 Requirements

The goal of this project is to evaluate the feasibility of using the RISC-V ISA for the SHMAC. Several requirements regarding a new prototype of the SHMAC can be derived from the problem described in the beginning of this report. Table 1.1 summarizes some of the more prominent ones that will be examined throughout the rest of the report.

Requirement	Description
R1	The new prototype of the SHMAC will need a network on chip
R2	The prototype has to support instantiation of memory tiles
R3	The prototype has to support instantiation of processor tiles based on the RISC-V ISA
R4	The RISC-V processor must support uncached and non-uniform memory access
R5	It must be possible to instantiate and run the prototype of the SHMAC within a simulated environment
R6	The simulation infrastructure must support instantiation of different layouts

Table 1.1: Requirements for the project, derived from the problem description.

In Section 2, the reader is introduced to the necessary background work for this project. Implementation and testing of the new prototype of the SHMAC are described in Section 3, and the results from simulating two different layouts of the prototype are provided and discussed with respect to the requirements in Section 4. Finally, Section 5 provides the conclusion for the report and discusses some of the work that did not make it into the project.

2 Background

This section describes the necessary background information about the SHMAC architecture and its implementation. Further, the reader is introduced to the Chisel Hardware Description Language (HDL) and, the RISC-V ISA and some of its hardware implementations.

2.1 The Network on Chip

A Network on Chip (NoC) is the only scalable form of communication between cores on many-core computers and has emerged as the solution for on-chip communication [7]. Network theory is applied together with on-chip communication, forming different topologies with higher efficiency than the conventional bus-based and crossbar interconnection topologies. Examples of three common NoC topologies are shown in Figure 2.1.

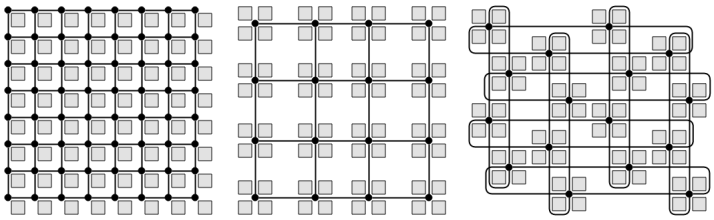

Figure 2.1: Three common NoC topologies showing a 2D Mesh, a Concentrated Mesh and a Concentrated Torus, respectively.

2.2 The SHMAC Architecture

The SHMAC [10] is a tile based architecture, where Processing Elements (PEs) and memory modules are connected together with their closest neighbors in a (n,m) grid network consisting of n rows and m columns. All the different tiles communicate with each other through a common interface, which is defined by the NoC connecting them together. Different versions of the SHMAC are then realized by configuring the different tiles and their location in the network.

The SHMAC requires that all PEs implement and execute the whole ISA, but not necessarily with the same performance. This makes it possible to easily evaluate the performance of different processors and accelerators, all executing the same bytecode. This requirement makes the choice of ISA very important. A high-level architecture of the SHMAC is shown in Figure 2.2.

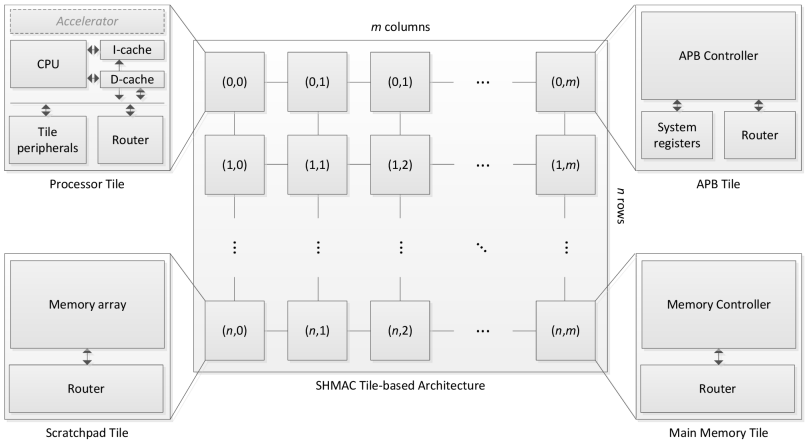

Figure 2.2: The SHMAC tile based architecture. Figure retrieved from the SHMAC project plan [10].

2.2.1 The SHMAC Processor Tile

Due to SHMAC's architecture of separate processor and memory tiles, it is a requirement that all PEs support both Non-Uniform Memory Access (NUMA) and uncached memory access. The processor tiles have to consist of at least one PE that implements the whole ISA and a router that is connected to the network. The design also supports additional tile peripherals like timers, interrupt controllers, and registers, where local information about the processors can be stored, e.g. information about its position and its id. This information can, among other things, be used by the router in order to send data packets to the right destination tile. The Central Processing Unit (CPU) can also be equipped with different accelerators specialized for different computing tasks, but it is not required for the accelerators to implement the whole ISA.

Currently, SHMAC implements the Amber [14] processor core from the OpenCores project, which is based on the ARMv4t ISA. The Amber processor tile consists of a CPU core, separate instruction and data caches, a router, and tile peripherals. All peripherals and caches are connected together with the router using a Wishbone bus. The architecture is shown in Figure 2.3.

As previously mentioned in Section 1.1, the current ISA and its implementation have imposed a lot of problems for the SHMAC project. The ISA is protected by copyright licenses and suffers from lack of extensibility, which makes it problematic to evaluate certain kinds of programs (e.g. programs optimized with vector instructions). The Amber core also suffers from being badly documented, which makes it difficult to update and maintain.

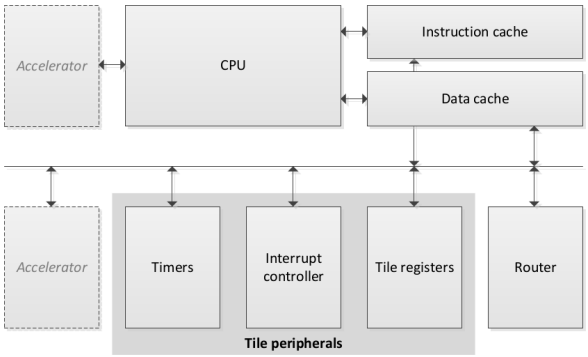

Figure 2.3: The Amber processor tile. Figure retrieved from the SHMAC project plan [10].

2.2.2 The SHMAC Memory Map

The memory space is divided into five different sections. Starting from the high end, the memory space consists of system registers, tile registers, scratchpad tile memory, main memory and an exception table. The scratchpad tile memory is divided into k different sections, one for each tile. They have all been given 128 MB of address space each, and the number for scratchpad tile memories are dependent on factors such as available RAM on the FPGA, among others. The memory space is shown in Figure 2.4.

It is important to note that SHMAC uses NUMA, which means that the memory access time depends on the memory location relative to the processor. As a result of this, it is not possible to know when to expect a response after a memory request goes off-tile. Thus, accessing the local scratchpad memory will always be faster than accessing any other off-tile memory.

2.2.3 The SHMAC Router and Interconnect Architecture

The different tiles on the SHMAC communicates with each other via a NoC for on-chip and off-chip communication. The routers have five ports it can send packets to, which are North, South, East, West and Local. Since each tile is identified by its x and y coordinates in the network, it is natural that the tile's input and output wires defines the network itself. It is also important to note that the network is absent from data-races, deadlocks and livelocks due to the XY routing algorithm [13] used to forward the packets.

The interconnect protocol is based on the simple, but powerful, Decoupled-interface, also known as a ready-valid protocol [15]. In this interface, each *data-bus* has two wires associated with them, *ready* and *valid*, respectively. These

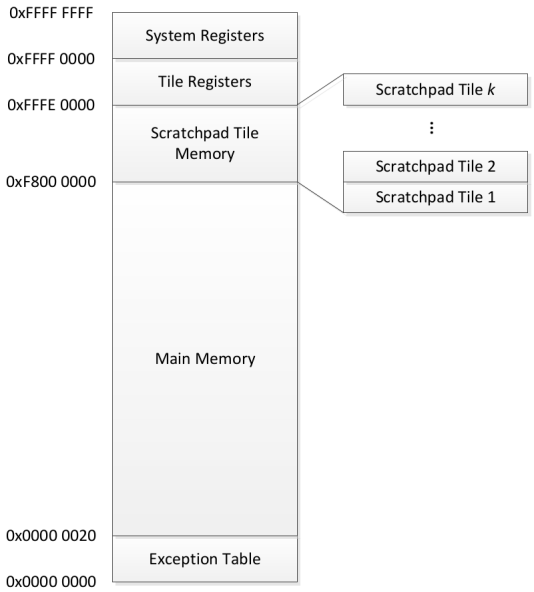

Figure 2.4: The SHMAC memory map. Figure retrieved from the SHMAC project plan [10].

three signals allow for correct communication between producers and consumers and if desired, across multiple clock-domains. The data between them is transferred when the valid-signal is set by the producer and the ready-signal is set by the consumer. An illustration of the protocol is shown in Figure 2.5.

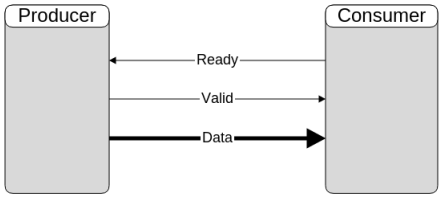

Figure 2.5: The Decoupled-interface between a producer and a consumer. Transfer of data occurs when both ready and valid are set.

The tiles are all connected together with a *router*, which is local for each tile. The router has five ports containing FIFO buffers to send and receive packets to and from its neighboring tiles. The last port of the router is connected to internal modules on the tile, which can be memories, CPUs or accelerators. A high level overview of such a network is shown in Figure 2.6.

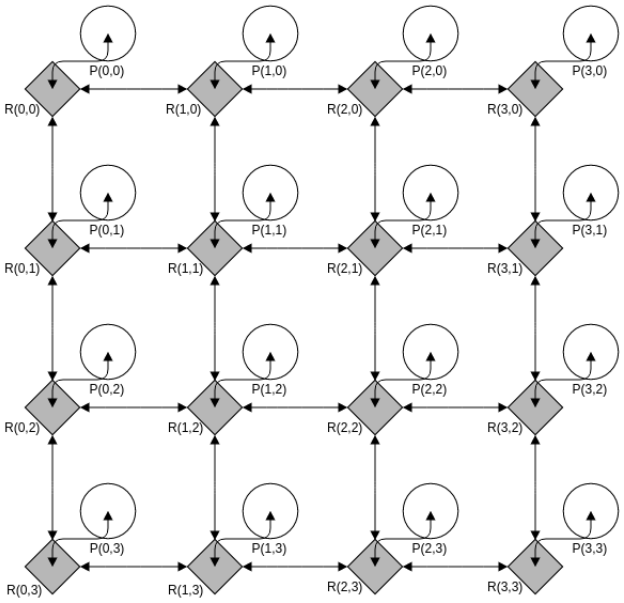

Figure 2.6: The SHMAC interconnect architecture. Each tile is connected with at most four neighbors via the router's input and output ports.

The router consists of several different modules, and its architecture is show in Figure 2.7. A *Crossbar* is at the center of the router, connecting the *Input ports* and *Output ports* together. The data path is driven by the ready/valid signals from these ports. The crossbar transmits packets only when the data is valid and the ports are ready. The *select*-signal to the crossbar is generated by an XY routing function and five Round Robin arbiters, one for each of the output ports.

It takes two clock cycles for a packet to traverse the router. The computation of the direction of the packets is done on the first cycle, while the switching and forwarding of the packets using the arbitration units is done on the second cycle. At most, five packets can traverse the router at any time, and this can only happen if the five input ports sends a packet in five different directions.

The packet is a central part of the interconnection protocol. It is 196 bits wide, where 53 bits are reserved for the header and 16 bits are reserved for the source and destination tile of the packet. The remaining 128 bits are reserved for the payload. The full packet format is shown in Figure 2.8.

The destination of the packet is sent to the XY routing function, which in turn is used to tell the crossbar in which direction to forward the respective packet. The destination of the packet is either the packet's *dest* field or the packet's

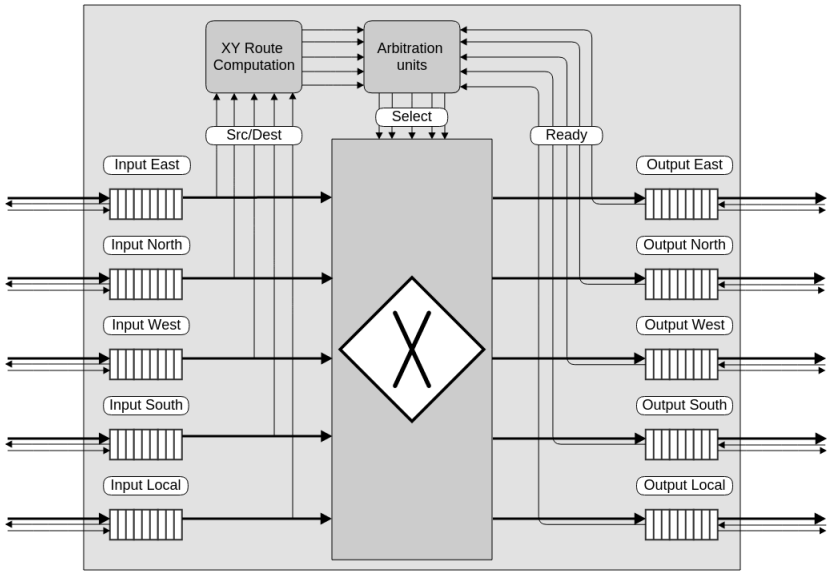

Figure 2.7: The SHMAC router architecture.

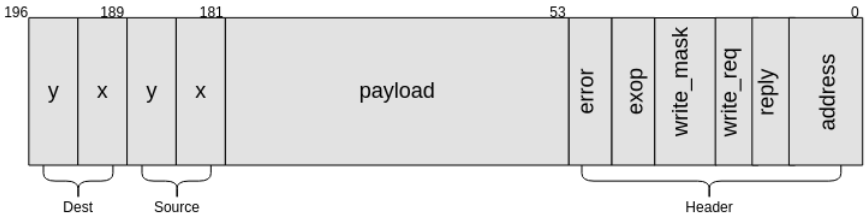

Figure 2.8: The SHMAC packet format.

source field, depending on whether the *reply* field in the header is set or not.

2.3 The Chisel Hardware Description Language

Chisel is an open-source HDL developed at the University of California, Berkeley (UCB), that supports advanced hardware design using highly parameterized generator layered domain-specific hardware languages [1, 4]. It is a Scala embedded programming language, which means that many of the features present in Scala are also available in Chisel. This allows hardware designers to work at a higher level of abstraction than with HDLs like VHDL and Verilog. Some features of Chisel that make it attractive for further development of the SHMAC

are described below.

Object-Orientation
: Some features of the Chisel Standard Library are *basic data types* and collections of such types called *bundles*. These bundles are used to interface with Chisel *modules* (similar to a Verilog module). Reuse of such bundles and modules allow for very efficient wiring when describing hardware.

Type-inference
: A powerful feature present in Scala is type-inference, where types of values and variables do not explicitly have to be specified by the designer, but rather inferred by the compiler. This increases productivity where changes can be made to component interfaces, without having to make changes to every other component integrating with this interface.

Generation of simulator and hardware
: One of the biggest motivations behind using Chisel is that the code can be compiled into both a cycle-accurate C++ simulator and synthesizable Verilog code. A problem on the SHMAC project is that the SHMAC implementation and its simulator have diverged and become two separate projects. Chisel makes it possible to have one code base for both.

The software architecture of Chisel is shown in Figure 2.9. The different target-sources are automatically generated by the Chisel compiler. Currently, Chisel only has support for a C++ and a Verilog back-end, but it can be extended to support other back-ends like e.g. a VHDL and a SystemC back-end.

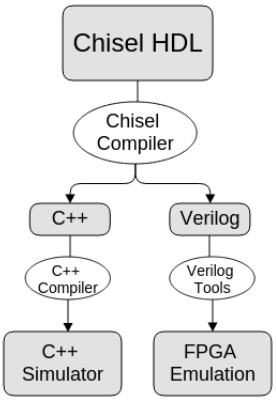

Figure 2.9: Chisel has a Scala front-end and support for multiple back-ends.

2.4 The RISC-V ISA

The RISC-V (pronounced "RISC 5") ISA [17] was originally designed to be used for computer architecture research and education. It is an open ISA that is freely available to use by both academia and industry. The ISA allows for efficient implementation of many different processor designs. Some of RISC-V's features that make it especially attractive for the SHMAC project are mentioned below.

It is open and free
: The Amber cores used on SHMAC are based on the ARMv4t ISA, and this has become a problem. The project aims to investigate and evaluate different architectures, so a proprietary ISA limits the extensibility of the SHMAC. Introducing an open ISA like RISC-V to the project can prove to be a wise decision, making it easier to explore a variety of different architectures.

It is easy to extend
: RISC-V consists of a small core of *integer instructions* that programs can depend on, but it is easily expandable with both *standard instructions*, like e.g. floating point instructions, and for extensions with *entirely new opcodes* that could be useful to new kinds of accelerators. This suits the SHMAC architecture's design, where each processing tile has to support the whole ISA, i.e. the base ISA, and instead be extended with accelerators that implement the optional extensions to the ISA.

Hardware implementations already exist
: An important choice for a new ISA is the availability of hardware implementations. There is a number of open-source cores available, the ones that are interesting for this project are discussed in greater detail in Section 2.4.1.

It has a rich software toolchain
: A number of software tools have been developed alongside RISC-V, including a GCC cross-compiler, an LLVM cross-compiler and a software ISA simulator and verification suite. A Linux port for RISC-V is also available. Software tools like these help increase the productivity during development. The reader is encouraged to visit the RISC-V website for the full list of available tools [16].

Other design choices and goals of RISC-V and argumentation for why a new and open ISA is needed is explained in greater detail by Asanović and Patterson [2].

2.4.1 RISC-V Hardware Implementations

The availability of open hardware implementations is essential in the choosing of a new ISA for the SHMAC. For the case of RISC-V, several open-source processor cores are already available. The ones that are interesting for this project are described below.

The Sodor Processor Collection [5] is a collection of five simple processor cores that implement the 32-bit Base Integer ISA. They have been developed at UCB to demonstrate a wide range of different integer pipeline implementations of RISC-V. They have been written in Chisel and are used in UCB's Computer Design classes. The cores and their differences are summarized in Table 2.1.

Type	**Summary**
1-stage	Essentially an ISA Simulator. It also implements the RV32IS supervisor mode of RISC-V, which allows the execution of the RISC-V proxy kernel [12].
2-stage	Demonstrates pipelining in Chisel.
3-stage Z-Scale	Uses sequential memory. It also implements the RV32IS supervisor mode of RISC-V, which allows the execution of the RISC-V proxy kernel.
5-stage	Can be compiled with either a fully bypassed pipeline or a fully interlocked pipeline implementation. The fully interlocked pipeline stalls until all pipeline hazards have been resolved, whereas the fully bypassed version only stalls during use-load hazards.
"bus"-based	Features a bus-based micro-coded implementation, where the different components of the machine communicates through a common bus.

Table 2.1: Summary of the Sodor Processor Collection [3, 5].

All the cores, except for the 3-stage one, are connected to a simple asynchronous scratchpad memory for shared instructions and data. The cores start fetching instructions from memory address `0x2000` and upwards when their *reset*-signal is unset.

The Rocket Core [11] is a new high-performance processor, featuring a 6-stage single-issue in-order pipeline, which executes the 64-bit scalar RISC-V ISA. It is tightly bundled with on-chip instruction and data cache, and implements both a Translation Lookaside Buffer (TLB) and a Memory Management Unit (MMU), with support for booting modern operating systems like Linux. It can also be extended with an optional IEEE 754-2008-compliant Floating Point

Unit (FPU), which can execute both single- and double-precision floating-point operations.

The Rocket core is used in architecture research at UCB and shows promising results when it is compared against the ARM Cortex-A5, with respect to both performance and energy efficiency [11].

2.5 Alternatives to RISC-V

There are also other alternatives for an open and free ISA, mainly OpenRISC by the GNU OpenCores community, and the SPARC V8 by Sun Microsystems. A comparison between the three ISAs and their features were performed by Asanović and Patterson, which is shown in Table 2.2. They designed the ISA to be both extensible and equipped for a wide range of computers, with a goal of working in both small-scale embedded devices fit for the Internet of Things (IoT) and in large-scale warehouse computers that require addressing more memory than can be accessed by a 64-bit memory address.

	Base+ext.	Compact Code	Quad FP	32-bit	64-bit	128-bit	GCC	LLVM	Linux	QEMU
SPARC V8			✓	✓			✓	✓	✓	✓
OpenRISC				✓	✓		✓	✓	✓	✓
RISC-V	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.2: Comparison between SPARC V8, OpenRISC and RISC-V.

The SPARC V8 ISA only supports 32-bit addresses, potentially limiting the amount of different heterogeneous processor cores that can be investigated. Thus, the most interesting alternative to RISC-V would be the OpenRISC ISA. The OpenCores project aims to develop a series of general purpose processors using the OpenRISC ISA. The ISA supports both 32-bit and 64-bit processors and optional floating point and vector processing support, much like RISC-V, but it is not so expandable. It is also an older project, as it started in 2000, and is not so much prepared for what it is believed that the future with the IoT and large-scale warehouse computers are going to look like [2].

Two features that make RISC-V especially suited for the SHMAC is its extensibility and its support for different address spaces, which support a wide range of different architectures. RISC-V has also shown good potential with several different open-source hardware implementations throughout the last few years.

3 Methodology

This section covers the development of the new prototype of the SHMAC. The router is ported from the existing version of the SHMAC, the memory tile is created from scratch, and the a Sodor processor core is modified in order to fit SHMAC's architectural requirements. At the end of the section, the process of developing and testing hardware with Chisel is described.

3.1 Choosing a NoC for simulation

One of the core features of heterogeneous computers like the SHMAC is the data-bus or the NoC that connects all the tiles together, creating a System on a Chip (SoC). Ultimately, we want an efficient network that connects all tiles together, which allow them to communicate with each other via packets that are automatically routed across the network.

This project will need a new NoC that can be easily simulated. There is a couple of different options available which are discussed below.

Porting the existing one
: While it is relatively easy to rewrite the existing SHMAC router from the Verilog source code to Chisel, it is not the most desirable option due to the extra amount of time it will take to implement and test. The existing NoC is not a very efficient one, but it is a minimal path network free of deadlocks and livelocks.

Creating a new one
: Creating a new and better NoC for SHMAC is not on top of the priority list of the project. A new design will consume a lot of time for implementation and testing, but it could prove to be a valuable choice if the network turns out to be the bottleneck of performance.

Using an existing one
: There is a number of available alternatives for a generated NoC, some more suitable than others. Requirements for such a solution would be that it is both open and freely available and that it is possible to run it in a simulated environment. The most promising alternative is the OpenSoC Fabric - a parameterizable and open source NoC generator written in Chisel, capable of generating a wide range of different network topologies [9].

It would have been ideal to use OpenSoC Fabric to generate the NoC, but it had not yet been released at the time of this writing. It was decided to port the existing one to Chisel, with the possibility of later moving over to OpenSoC Fabric if performance turns out to be a problem.

3.2 Deciding on a Processor Core

All the RISC-V hardware implementations discussed in Section 2.4.1 communicates with some kind of cache or scratchpad memory. Thus, no matter the choice of processor core, it will require some modification in order for it to support NUMA. Out of the different implementations, the Rocket core is the most interesting one. It is an advanced core that is already used as basis for UCB's architecture research, it provides support for running operating systems like Linux, and it can easily be extended with an FPU. However, it is tightly bundled with the on-chip data and instruction cache and it will require more work with integration than this project allows.

Out of the different available cores in the Sodor Processor Collection, the 3-stage one was chosen as the best option. It differs from the other Sodor processors in that it also implements the minimal version of the RISC-V supervisor mode (similar to the 1-stage, but with better performance).

The pipeline of the 3-stage Sodor processor, also known as the RV32IS Z-Scale [6], is split into a *Front-end* and a *Back-end*. The three stages are *Fetch*, *Execute* and *Writeback*, respectively. The fetch stage belongs to the front-end and the execute and writeback stages belongs to the back-end. This modular design allows for experimentation with different implementations of the front-end, potentially increasing the performance of the instruction fetch stage. A detailed overview over the processor core is provided in Appendix A.

3.3 Porting the Router and Interconnect Network

The implemented router is almost directly ported from the existing Verilog code, with the main difference being that the Chisel code is a lot more compact and readable. This is partly due to the fact that hardware-modules like *Queues* and *Arbiters* are already part of the Chisel Standard Library.

The architecture of the router is the same as the one shown in Figure 2.7. A difference between the implementations is that this packet is smaller, with only 4 bytes reserved for the payload, instead of the 16 bytes that was described in Section 2.2.3. It made first iteration of implementation easier, but it can later be changed in order to improve overall performance. The updated packet format is shown in Figure 3.1.

3.4 Implementation of the Memory Tile

The memory tile consists of two separate modules; a *Router* and a *Memory* that are connected to the router's Local input and output ports. The memory module is provided as part of the Chisel Standard Library, which makes it easy to instantiate and use. The architecture is shown in Figure 3.2.

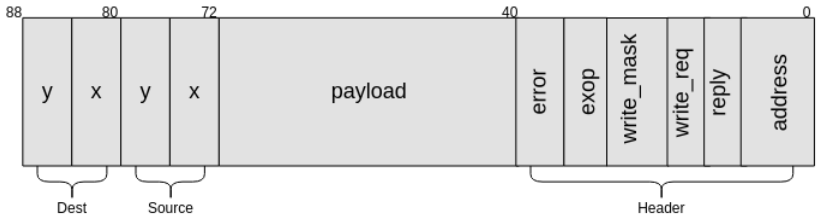

Figure 3.1: The packet format for the SHMAC prototype.

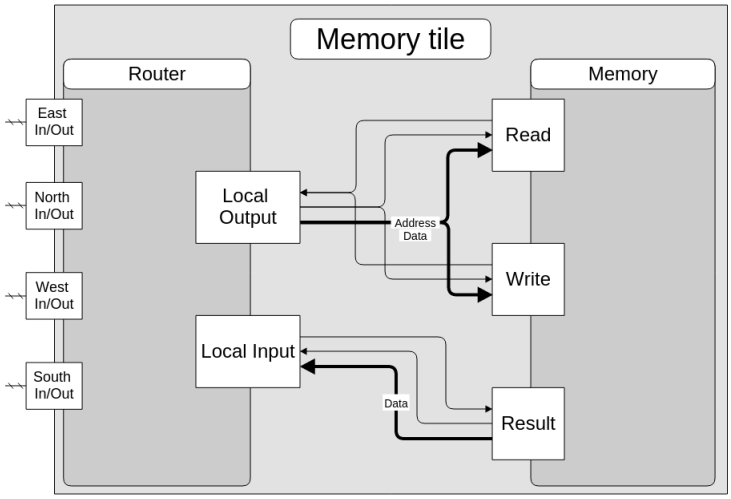

Figure 3.2: The Memory Tile.

3.5 Implementation of the Sodor Tile

The Sodor Tile's I/O-interface consists of its routers input and output ports and control signals for the Sodor core itself. All memory requests and responses are wrapped in a packet and sent over the on-chip interconnect network. Thus, even though the core itself has two memory ports dedicated for *instructions* and *data* respectively, they are both wired up to the same data bus going off-tile. This is achieved with a custom memory arbiter that arbitrates between the different memory accesses and connects to the router.

The core is controlled by a Host-target Interface (HTIF), originally used by the Sodor cores to load programs and data to its scratchpad memory and read its Control Status Registers (CSRs). The only signal from the HTIF that is still in use in this implementation is the core's *reset*-signal. The architecture of the processor tile is shown in Figure 3.3.

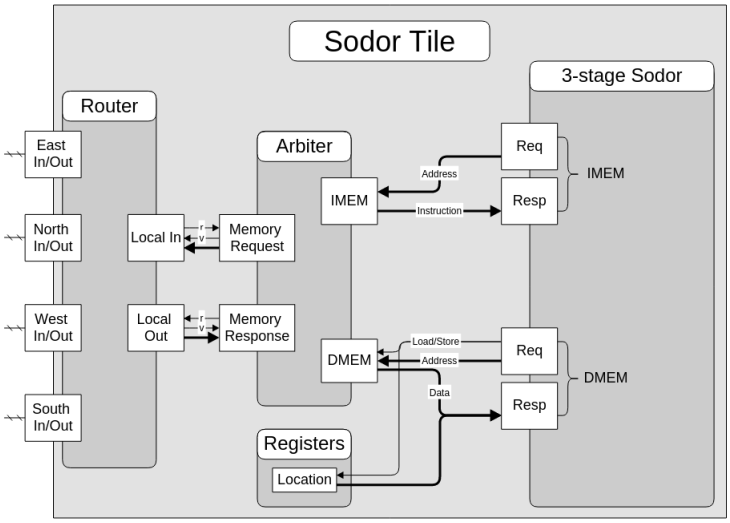

Figure 3.3: The Sodor Processor Tile.

The Sodor core needed some modification in order to fit SHMAC's architecture and its requirement for uncached and variable length memory accesses. The scratchpad memory was removed and the memory arbiter was modified to stall both the data- and the control-path of the processor, when it was waiting for either instructions or data. These modifications were done by another member of the SHMAC team.

The Sodor Tile is also equipped with one additional *tile location register*. This register contains the *id* of the tile, based on its location in the network, and it can be obtained by reading from memory address `0x1`. This register makes it easy to write a single program that all cores can start to execute, but behave differently by branching to different parts of the program based on the value of the regsiter.

Since there is no on-tile cache associated with the core, it will, in the best case scenario (the case where it neighbors its instruction memory), start executing a new instruction every 9th clock cycle. The 8 first cycles are dedicated to requesting and receiving a packet wrapping the instruction, and the last cycle to start executing it and requesting the next one.

3.6 Working with Chisel

There is a big difference between programming a standard microprocessor chip and programming a programmable circuit. The former involves the specification of a sequence of actions, or instructions, while the latter involves a configuration

of the machine itself.

Chisel's high level of abstraction makes hardware description a more productive process. The code is also easier to read, partly due to Chisel's way of splitting a wide data-bus into smaller, parameterizable fields (called a bundle), instead of directly accessing it with different indexes. This is of great help because it makes it possible to abstract away the wide bus that is otherwise associated with the SHMAC packet, and instead work with a collection of seem-to-be separate data fields.

3.6.1 Testing and Simulation

Every module that has been implemented has also been tested. Testing hardware is just as important, if not even more important, as testing software. It is very expensive to fix bugs and errors in hardware after it is in production, making for a good motivation to test and simulate hardware designs during every stage of development.

Testing and simulation of Chisel modules are done in the same Scala-embedded environment as the modules are described in. The modules have been tested differently depending on its core feature. The different tests can be classified as being either one, or a combination, of the following;

Unit tests
: Have been implemented for the simplest and most basic modules. These tests have covered most or all of the different scenarios of I/O for the modules, in order to verify a solid implementation. An example of a module that is covered by such a test is the *crossbar* found in the Router.

Integration tests
: Have been important when testing and verifying that the integration between the different modules are working as supposed to. They have played the biggest part in verifying overall correctness of the system, especially when testing modules like the *MemoryTile* and the *SodorTile*.

Simulation tests
: Although it can be argued that all tests are simulations, they differ slightly in the kind of functionality that is being tested. The simulations have more setup, and they typically only check that the system is behaving correctly and produces the correct results without checking for correctness at every clock cycle. Simulation tests were performed on the modified version of the 3-stage Sodor core and for the different instances of the SHMAC prototype.

Table 3.1 summarizes the different hardware components that make up the router and what kind of premise they have been tested on. Table 3.2 summarizes the different tests that have been performed on the memory component and the modified version of the 3-stage Sodor core. The same tests have also been performed on the respective tiles, and they are summarized in Table 3.3.

These programs also form the basis when simulating the different instances of the SHMAC prototype. The results from these simulations are presented in Section 4.

	Module	Unit Test	Integration	Simulation
Router	XY Route Computation	✓		
	Input Port	✓	✓	
	Output Port	✓	✓	
	Crossbar	✓		
	Direction Arbiter	✓		
	Direction Router	✓	✓	
	Packet Format	✓		
	Router	✓	✓	

Table 3.1: Test coverage of the modules in the Router.

	Test Description	Unit Test	Integration	Simulation
Mem	Simple Read/Write	✓		
	Sum of memory	✓		
Sodor	Store Immediate	✓		✓
	Add Immediate	✓		✓
	Simple Load/Store	✓		✓
	Load, Add, Store	✓		✓

Table 3.2: Test and simulation coverage of the Memory module and the Sodor core.

	Test Description	Unit Test	Integration	Simulation
Mem Tile	Simple Read/Write		✓	
	Sum of memory		✓	
Sodor Tile	Store Immediate		✓	✓
	Simple Load/Store		✓	✓
	Load, Add, Store		✓	✓
	Load Tile Location		✓	✓

Table 3.3: Test and simulation coverage of the Memory Tile and the Sodor Tile.

4 Results and Discussion

This project has been as much of an evaluation of Chisel as it has been an evaluation of RISC-V. This section contains a quick comparison of Chisel and Verilog, and presents the results from simulating two different layouts of the SHMAC prototype.

4.1 Comparing Chisel and Verilog

This has been an implementation heavy project, where most of the time has been spent writing, testing and simulating Chisel code. Since the router was almost directly ported from Verilog to Chisel, it is easy to compare the two code bases against each other in order to try to give a direct comparison between Chisel and Verilog.

Modules	**Verilog**	**Chisel**
FIFO	70	-
Arbiter	71	30
Input Port	35	18
Output Port	22	8
Routing Function	22	24
X-Bar	37	23
Router	311	79
Network Interface	122	69
Sum	**690**	**296**

Table 4.1: Comparing expressiveness of Verilog and Chisel based on the number of lines of code present in the Router module.

We can see from the results, summarized in Table 4.1, that the Chisel code base is about half the size compared to that of Verilog. This is mostly due to the fact that,

1. There are already several modules present in Chisel, including both FIFO Queues and Round Robin Arbiters.
2. There is no use of *generate* statements in the Verilog code, which makes the Router module a lot larger than that of Chisel.

The expressiveness and high level of abstraction combined with Chisel's simulation infrastructure allows for a quick and iterative process of describing and testing hardware modules.

4.2 Simulations

Two different instances of the SHMAC prototype have been simulated, as described in Section 3.6.1. For simplicity, the two instances are referred to as *Simple SHMAC* and *Advanced SHMAC* throughout the following subsections. These two instances are interesting because they demonstrate a working prototype of the SHMAC based on the RISC-V ISA, with the Simple SHMAC acting as a proof-of-concept and the Advanced SHMAC demonstrating a many-core system.

4.2.1 Simulation of Simple SHMAC

The first layout demonstrates the simplest possible version of the SHMAC, consisting only of a processor tile and a memory tile. The system architecture is shown in Figure 4.1. The simulations demonstrate a fully working system, with the memory acting as both IMEM and DMEM. The different simulation tests are summarized in Table 4.2.

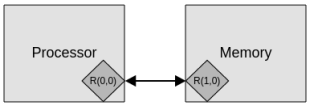

Figure 4.1: High-level architecture of Simple SHMAC.

Name	**Purpose**	**Result**
Store Immediate	Demonstrates the integration between the processor tile and the memory tile.	**Success**
Load, Add, Store	Demonstrates a fully working system where the processor loads two numbers from memory, sums them together and stores the result.	**Success**
Branch	Based on the previous simulation, but sums and stores different numbers based on whether the branch should be taken or not.	**Success**
Branch on Location	Essentially the same as the previous one, the difference being that it branches based on the value in its tile location register.	**Success**

Table 4.2: Simulation summary of Simple SHMAC.

4.2.2 Simulation of Advanced SHMAC

The second layout that has been simulated is shown in Figure 4.2. This layout is interesting because it demonstrates a system with multiple cores working sepa-

rately, both accessing the same memory and executing the same program.

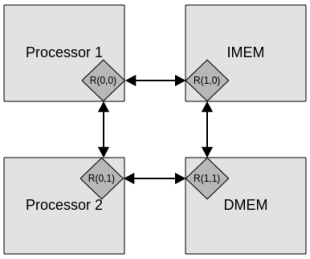

Figure 4.2: High-level architecture of Advanced SHMAC.

Address	Program	P1	P2
0x2000	LW r2, 0x1	|	|
0x2004	BEQ r0, r2, 0x18	|	|
0x2008	LW r3, 0x0		|
0x200c	LW r4, 0x4		|
0x2010	ADD r5, r3, r4		|
0x2014	SW r0, r5, 0x10		|
0x2018	JAL r0, 0x14		|
0x201c	LW r3, 0x8	|	
0x2020	LW r4, 0xc	|	
0x2024	ADD r5, r3, r4	|	
0x2028	SW r0, r5, 0x14	|	
0x202c	...	|	|

Table 4.3: The program used during simulation of Advanced SHMAC.

The program that was used during simulation is shown in Table 4.3. It starts by loading the value of the tile location register and branches if the value is equal to 0, with the result being that P1 executes the second part of the program and P2 executing the first part. P2 also executes a jump-instruction, so both processors will continue executing the same code at memory address `0x202c`.

Another thing to notice from this simulation is that P2 is further away from the IMEM, making every instruction execution four cycles longer than for P1. The result of this is that P2 finishes the program later in time. Similarly, P1 spends four extra cycles to load and store data from the DMEM.

4.3 Discussion of Requirements

The goal of the project was to evaluate the feasibility of RISC-V for SHMAC in a simulated environment. Section 1.3 introduced a number of requirements

for the new SHMAC prototype; how the problems were examined and solved is discussed below.

The new prototype of the SHMAC will need a network on chip
The router and the NoC were the first part of the implementation that was tackled. It was decided to port the router from the existing Verilog code, even though it would require more time to develop a new NoC than using an existing infrastructure to generate it. The implementation proved to be valuable in order to become familiar with Chisel and hardware description in general.

The prototype has to support instantiation of memory tiles
The Memory class and the predefined I/O-interfaces included in the Chisel Standard Library made for a quick implementation of the Memory Tile without much problem.

The prototype has to support instantiation of processor tiles based on the RISC-V ISA
It was decided to use the 3-stage Sodor processor for this project. It took more time than first anticipated to attain deterministic behavior from the processor. Deterministic behavior was achieved with simple test programs and manually manipulating the I/O-signals until it worked as expected.

The RISC-V processor must support uncached and non-uniform memory access
The Sodor core's scratchpad memory was removed and its internal functionality had to be modified in order for it to support NUMA. These modifications might have played a role with some of its hard-to-determine behavior and has made the process of integrating the core into the project harder.

It must be possible to instantiate and run the prototype of the SHMAC within a simulated environment
The SHMAC has been lacking a simulator that is up to date with the development status of the project, which give motivation to the use of Chisel throughout this project. Chisel has proved to be a valuable tool and HDL, making the process of describing hardware an easier and quicker task, as well as providing the simulator for the project. Chisel also generates synthesizable Verilog from the same code base, but it is not without its flaws. It is still under active development, and features like cleaner error messages and better error handling would have been appreciated. Especially in some situations where obscure null pointers and runtime errors with undescriptive error messages occur even for the simplest mistakes.

The simulation infrastructure must support instantiation of different layouts

Two different layouts of the new SHMAC prototype were simulated and presented, demonstrating both a simple proof-of-concept of the SHMAC and a system with multiple processors and variable memory access time. These two layouts have demonstrated a working machine based on the RISC-V ISA and have helped showing the feasibility of the RISC-V ISA for the SHMAC.

5 Conclusion

The proprietary ARM ISA that is currently in use on the SHMAC has introduced a lot of challenges during its research and development, and it is time to replace it with another one. Out of the different available alternatives, RISC-V was deemed to be the most suitable. Additionally, the project has been lacking a simulator that is up to date with the project, motivating the move of further development to the new platform defined by Chisel.

This project has been part of introducing a lot of new potential tools and technologies to the SHMAC project, including RISC-V and its toolchain, Chisel and Sodor. A new prototype of the SHMAC was realized on the new architecture, and two different instances of the prototype were generated and simulated. The two instances demonstrated a working system of separate tiles connected together with an on-tile interconnect network. The new prototype of the SHMAC showed that the move over to the RISC-V ISA is indeed a feasible option.

5.1 Future work

The work made in this project covers a lot of ground, which has left many potential tasks open to be done later. Some of the more prominent ones that did not make it into this project are mentioned below.

Experiment with OpenSoC Fabric
: OpenSoC Fabric is now available. It would be interesting to hook up different generated NoC designs to the router interface in order to evaluate their performance against each other. This also fits SHMAC's goal of evaluating different architectures for different problems. Since the NoC is the backbone connecting all the tiles together, it can quickly become the bottleneck in the system.

Optimize generation of routers
: One of Chisel's strong features is its high level of abstraction combined with the expressiveness of Scala. It is very efficient to use Chisel to generate different designs upon instantiation. Currently the routers are all instantiated with five ports, no matter where they are located in the network. It would be nice to be able to generate routers with only the required number of ports. In the case of the architecture in Section 4.2.2, 40% of the ports are left unused. This can turn out to be a problem in resource-constraint environments.

Create a better framework for tiled layouts and simulations
: There is a lot of potential for improvement by making the SHMAC module more generic. This will make it possible to create a more general simulator that supports different variations of the SHMAC, with better support for loading programs to memories and evaluating the simulation results. Such

an implementation is already in progress, but there was not enough time left of the project to cover it.

Improve performance with an optimized front-end

As already mentioned in Section 3.2, it is possible to compile the Sodor processor with different front-ends. Performance could be improved significantly by e.g. modifying it to either fetch more than just one instruction at a time, or issue several requests in bursts.

Improve performance with local cache

Performance was never a goal of this project. However, it could be beneficial to improve the performance of the Sodor processor by providing the core with an on-tile instruction and data-cache.

Introducing multiple clock-domains

There is a lot of potential for experimenting with different clock-domains on e.g. the interconnect architecture and the tiles. If the clock for the network operates at double speed of the processors, the performance of the packet communication will increase at the same rate - relative to the processor. It will be interesting to see if such experimentation can improve the overall performance of the system, with respect to energy efficiency.

6 Bibliography

[1] Chisel: Constructing Hardware in a Scala Embedded Language. Available: `https://chisel.eecs.berkeley.edu/`. [Accessed: 2014-09-30].

[2] K Asanović and DA Patterson. Instruction Sets Should Be Free: The Case For RISC-V. 2014.

[3] Krste Asanovic and Christopher Celio. CS152 Computer Architecture and Engineering - Laboratory. 2012.

[4] Jonathan Bachrach and Vincent Lee. Getting Started with Chisel. 2014.

[5] Christopher Celio. The Sodor Processor Collection. Available: `https://github.com/ucb-bar/riscv-sodor`. [Accessed: 2014-12-12].

[6] Christopher Celio. The Sodor Processor Collection Github Wiki. Available: `https://github.com/ucb-bar/riscv-sodor/wiki`. [Accessed: 2014-12-12].

[7] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 684–689, 2001.

[8] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *Proceeding of the 38th annual international symposium on Computer architecture - ISCA '11*, page 365, 2011.

[9] Farzad Fatollahi-Fard, David Donofrio, George Michelogiannakis, John Shalf, Ke Wen, John Bachan, and Daniel Burke. OpenSoC Fabric - Home. Available: `http://opensocfabric.lbl.gov/home.php`. [Accessed: 2014-09-14].

[10] Magnus Jahre. Single-ISA Heterogeneous MAny-core Computer (SHMAC) Project Plan. (March):1–21, 2014.

[11] Yunsup Lee, Andrew Waterman, Rimas Avizienis, Henry Cook, Chen Sun, Vladimir Stojanovic, and Krste Asanovic. A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators. *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*, pages 199–202, September 2014.

[12] Yunsup Lee, Andrew Waterman, and Christopher Celio. RISC-V Proxy Kernel. Available: `https://github.com/ucb-bar/riscv-pk`. [Accessed: 2014-12-12].

[13] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2), 1993.

[14] Conor Santifort. Amber Open Source Project Amber 2 Core Specification May 2013. (May), 2013.

[15] Montek Singh and Chapel Hill. Generalized Latency-Insensitive Systems for Single-Clock and Multi-Clock Architectures. 2004.

[16] Andrew Waterman, Yunsup Lee, and David Patterson. RISC-V - Home. Available: `http://riscv.org`. [Accessed: 2014-12-11].

[17] Andrew Waterman, Yunsup Lee, and David Patterson. The RISC-V Instruction Set Manual. I, 2014.

Appendices

A Z-Scale 3-Stage Sodor Processor Core

The detailed architecture of the 3-stage Sodor core is shown in Figure A.1.

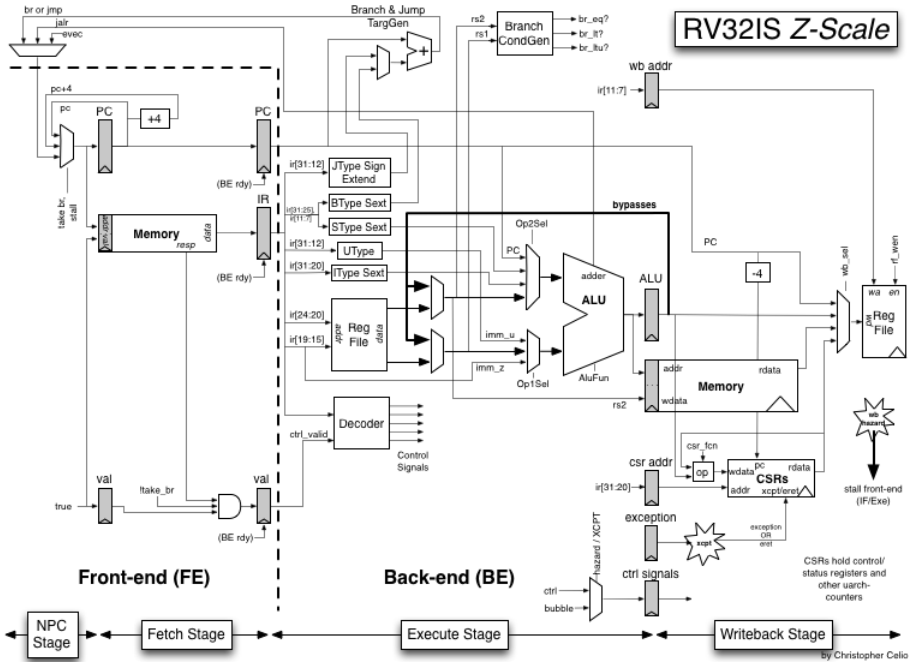

Figure A.1: The 3-stage Sodor processor core. Figure retrieved from the documentation of the Sodor Processor Collection [6]