**Multi-Core Processor with RISC V ISA Implemented on FPGA**

Graduate Project Proposal

Benjamin Kueffler

November 19, 2018

[**1. Table of Acronyms** 3](#_Toc531278623)

[**2. Introduction** 4](#_Toc531278624)

[**3. Objective** 4](#_Toc531278625)

[**4. Application** 4](#_Toc531278626)

[**5. Proposed Work** 5](#_Toc531278627)

[**5.1. System Design** 6](#_Toc531278628)

[**5.2. Processor Features** 6](#_Toc531278629)

[**5.2.1 Five Stage Pipeline** 6](#_Toc531278630)

[**5.2.2 Hazard Unit** 6](#_Toc531278631)

[**5.2.3 Branch Predictor (Stretch Goal)** 6](#_Toc531278632)

[**5.2.4 Cache Coherency (Stretch Goal)** 6](#_Toc531278633)

[**5.2. System Block Diagram** 7](#_Toc531278634)

[**6. Methodology** 8](#_Toc531278635)

[**6.1. Processor Design** 8](#_Toc531278636)

[**6.2. Synthesis & Test** 8](#_Toc531278637)

[**7. Schedule** 8](#_Toc531278638)

[**8. Required Resources** 9](#_Toc531278639)

[**9. References** 10](#_Toc531278640)

# **1. Table of Acronyms**

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| **AXI** | Advanced eXtensible Interface |
| **CPU** | Central Processing Unit |
| **DUT** | Design Under Test |
| **FPGA** | Field-Programmable Gate Array |
| **HDL** | Hardware Description Language |
| **ISA** | Instruction Set Architecture |
| **RISC** | Reduced Instruction Set Computer |
| **RV32I** | Base Integer Instruction Set, 32-bit (RISC V) |
| **VHDL** | Very High Speed Integrated Circuit Hardware Description Language |

# **2. Introduction**

Multicore processors have expanded dramatically in the last two decades. Their proliferation was based out of a need for faster computation without increasing CPU frequencies. Pushing higher frequencies in CPU designs increases the power and resulting thermals. This created a desire for multicore processors due to their higher performance per watt [1]. This need has continued into the 2010s from the rise of embedded devices, such as smartphones, which have an even greater need for power savings and multitasking.

Alongside any processor is the underlying instruction set that it is built around. Instruction sets are often classified by complexity of the instructions. An instruction set that relies on many different complex instructions is called a complex instruction set computer (CISC), whereas one with a smaller base instruction set is called a reduced instruction set computer (RISC). RISC ISAs allow for a simplified pipeline within the CPU, due to each instruction being relatively less complex than with a CISC ISA [2]. The RISC V ISA was chosen for this project due to its simple base instruction set, its ubiquity as an open source ISA, and its ability to expand the instruction set beyond the base instruction set (RV32I).

While the performance and tradeoffs of multicore and ISAs can be debated, we can realize a hardware implementation of both of these concepts using Field-Programmable Gate Arrays (FPGA). Using this reprogrammable hardware allows for quick modifications, as well as analysis of the design tradeoffs. The number of cores can be easily modified via parameters at design time in order to synthesize and test the performance of various levels of multicore CPUs. FPGAs also allow for a cheap way to test a prototype CPU without requiring fabrication or rely on simulation results for performance estimates.

# **3. Objective**

The objective of this project is to construct a RISC V based multicore central processor unit within an FPGA and demonstrate performance upon changing the number of generated cores.

# **4. Application**

Modern embedded applications have been implementing multicore processors for stronger performance and versatile power utilization. For example, the Apple A12 Bionic mobile SOC contains six CPU cores on its own in addition to GPU and neural processor [3]. These kind of mobile applications, while not doing the same level of multi-threaded workloads as workstations, still utilize multiple cores in order to have a subset of cores to do high performance, while the rest of the cores act as energy efficient (low clock speed) CPUs This allows these embedded devices to utilize fast cores when necessary, while saving battery life by using energy efficient cores when only small computations are needed.

For the purpose of targeting the embedded application, the FPGA design will be tested for this in of application by running an image processing algorithm and comparing the performance and energy consumption with one core against a multicore approach.

Image processing was chosen as the algorithm to test due to its varying performance among single core, multicore, and GPU workloads. Image Processing is also a common application in mobile embedded systems, particularly with the large amounts of image preprocessing being required for machine learning [4]. Image preprocessing involves taking a series of images from the dataset and performing some alterations before utilizing the altered image. This kind of processing is often done in machine learning applications in order to train the model using uniquely altered images. Multicore processors are often preferred for this function in order to easily handoff images to the GPU while it’s training the model.

In order to test the performance and energy utilization of the processor cores, a small amount of code will be produced and loaded into the memory alongside a dataset. The code will take some image and perform a variety of preprocessing instructions on it. In order to match a realistic use embedded multicore processors, many of the cores will have varying clock speeds in order to decrease energy use while still providing higher performance when required.

# **5. Proposed Work**

The work will involve implementing, testing, and synthesizing the CPU (DUT) using VHDL and Xilinx Vivado ISE. The primary work will be designing the CPU, however, an FPGA system will need to be created around the CPU in order to provide the environment that the CPU can be tested under. The components in this FPGA system will be reused from Xilinx IP libraries and from other open source resources.

## **5.1. System Design**

The design shall feature an AXI bus in order to establish the CPU core(s) as a master, and the higher memory as the slave. The highest level memory established in this system is intended to be an SD card, communicating over SPI. This allows large amount of memory to be accessed and loaded into the CPU’s cache. The memory will contain a pre-compiled program via a RISC V RS32I compiler.

## **5.2. Processor Features**

In the interest of reducing the complexity of the project, only critical aspects of the CPU will be included in this design, with others being left as a stretch goal.

### **5.2.1 Five Stage Pipeline**

The processor will include a five stage pipeline. This is a common pipeline used in many RISC designs, the pipelining of the stages allows for a higher clock speed to be achieved in return for longer latency.

Table 1) Five Stages of Pipeline

|  |  |
| --- | --- |
| **Stage** | **Description** |
| **Fetch** | Grabs the instruction from the cache, handles program counter. |
| **Decode** | Reads the instruction from the opcode in order to determine action. Branch/Jump commands are processed. |
| **Execute** | Contains the arithmetic logic unit in order to carry out a transaction. |
| **Memory Access** | Access to the data memory occurs. |
| **Writeback** | Write back the results of the execution and memory access into the appropriate register. |

### **5.2.2 Hazard Unit**

The hazard unit of the processor shall handle data and control hazards as a result of the pipelined nature of the device. Data hazards will come into play when transactions that occur later in the pipeline rely on data that has not been computed further along down the pipeline. This can often be handled by forwarding the relevant data later along in the pipe. Efforts will be taken to reduce stalling along the pipeline. Control hazards will appear in the pipeline when the branch predicted does not match the actual branch calculated. The hazard unit will attempt to remedy this by incorporating stalls when the prediction is incorrect. This will result in a large performance hit.

### **5.2.3 Branch Predictor (Stretch Goal)**

The decode stage of the pipeline will contain some simple branch prediction scheme. For example, static branch prediction. As a stretch goal, a simple dynamic branch prediction scheme may be implemented in order to cut down on missed predictions.

### **5.2.4 Cache Coherency (Stretch Goal)**

In order to cut down on the complexity of the design, software is intended to handle any cache coherency issues that may arise (for example, by loading in four completely memory isolated programs in a CPU with four cores generated). As a stretch goal, this can be expanded to allow the hardware to handle cache coherency issues. This would mean that the hardware would need to implement a snooping mechanism in order to alert itself when another core has modified its shared data.

## **5.2. System Block Diagram**

The system was built with the AXI4 protocol in mind, this protocol is commonly used in FPGA designs in order to achieve a high performance, memory-mapped system [5]. Xilinx also has numerous configurable IP that can take advantage of such a system.



Figure 1) System Block Diagram

Table 2) System Block Diagram Descriptions

|  |  |
| --- | --- |
| **Component** | **Description** |
| **CPU Core** | The design under test, the CPU core(s) acting as an AXI master to the rest of the system |
| **CPU Start/Status** | AXI slave registers that allow for the UART master to command the DUT to start or read status for debugging |
| **USB Controller** | Xilinx IP that accepts USB I/O and converts the USB to AXI transactions so the PC may check status and start the CPU cores. Also allows loading of memory. |
| **SPI Controller** | Allows the CPU Core to access external memory found on an SD card or other memory SPI peripheral. |
| **AXI Bus Interconnect** | Xilinx IP that allows for each AXI master to talk to a memory addressed AXI slave. |

# **6. Methodology**

## **6.1. Processor Design**

The processor will be designed in a modular fashion, each individual stage will be broken into its own component and individually tested through HDL verification. A final test bench will be created to verify that each command that is sent through the processor can be handled, along with strings of commands that cause hazards. The individual verification of each submodule ensures that the length of time for each test bench is relatively small, so that small changes can be verified in a relatively short length of time. Tests will be assertion based, so that a fail state will be returned to the programmer when an unexpected test result appears.

## **6.2. Synthesis & Test**

The system design will be taken through synthesis using Xilinx Vivado ISE. After the synthesis stage, the user will interface through the design via the UART port. This UART port will allow the loading of memory with machine code, in order to carryout instructions. The UART port will also be responsible for starting and statusing each of the CPU cores, in order to engage the test. Live performance testing will occur with the machine code, and the memory will be read out by the UART port in order to verify that the correct data output has been written by the processor. The timing performance and instructions per second calculation will be calculated due to the presence of the RDTIME instruction at the start and end of our sample machine code. This instruction will return the arbitrary clock cycles that have passed since the first instruction [6].

# **7. Schedule**

Table 3) Schedule

|  |  |
| --- | --- |
| **Time Range** | **Description** |
| **Winter Break** | Create and synthesize all Xilinx IP intended to be used in the system design. Verify the functionality of these parts before starting design |
| **Week 1** | Create basic diagram of CPU design under test, including each stage and hazard unit/branch predictor/cache |
| **Week 2-3** | Implement basic instruction/data cache |
| **Week 3-5** | Test instruction/data cache |
| **Week 6-7** | Implement basic five stage pipeline (no hazard unit) |
| **Week 8-11** | Verify basic five stage pipeline (Do not test hazard edge cases) |
| **Week 12** | Implement Hazard Unit |
| **Week 13-14** | Verify Hazard Unit |
| **Week 15** | Finals |
| **Summer** | Synthesis, place, and route. Verify that single core processor can communicate with PC and carry out tests. Verify performance of single core processor. |
| **Week 16-17** | Implement configurable # of cores. |
| **Week 18-19** | Synthesize, place, and route with multiple cores. Verify that processors can run code which is *completely independent* of one another. Cache coherence not a concern for this test. |
| **Week 20** | Implement Branch Predictor |
| **Week 21** | Test Branch Predictor |
| **Week 22-24** | Implement Cache Coherence in hardware (Stretch goal) |
| **Week 25-27** | Verify cache coherence in simulation and final synthesis (Stretch goal) |
| **Week 28-30** | Document final performance results and present |

# **8. Required Resources**

The Xilinx Spartan7 Chip was chosen as the FPGA that would be used in implmenting the design. This is because it provides the most logical cells / cost for this purpose. The other FPGA that was considered was the Artex7, but for a die at the same cost, you would get less logic cells as a tradeoff for having GTP transceivers and PCI express support.

Table 4) Required Resources

|  |  |  |
| --- | --- | --- |
| **Resource** | **Description** | **Cost** |
| [**Spartan-7 50T Development Board**](https://store.digilentinc.com/arty-s7-spartan-7-fpga-board-for-hobbyists-and-makers/) | Development board containing the Spartan-7 50T chip along with programming port and SPI / DDR accessible memory for storing machine code. | $119 |
| **Vivado** | Development environment for generating Xilinx IP and synthesizing implementation. | Free |

# **9. References**

[1] “The Benefits of Quad Core CPUs in Mobile Devices.” NVIDIA, 2011. Tegra Whitepaper 0911a

[2] “RISC,” *Gartner IT Glossary*, 14-Nov-2012. [Online]. Available: https://www.gartner.com/it-glossary/risc-reduced-instruction-set-computer. [Accessed: 08-Oct-2018].

[3] Apple. (2018). *iPhone XS – A12 Bionic*. [online] Available at: https://www.apple.com/iphone-xs/a12-bionic/ [Accessed 19 Nov. 2018].

[4] TensorFlow. (2018). *Performance / TensorFlow.* [online] Available at: https://www.tensorflow.org/guide/performance/overview [Accessed 19 Nov. 2018].

[5] “AXI Reference Guide.” Xilinx, 15-Jul-2017.

[6] A. Waterman and K. Asanovi´c, “The RISC-V Instruction Set Manual.” CS Division, EECS Department, University of California, Berkeley, Berkeley, 07-May-2017.