American University of Beirut

EECE 502: Final Year Project

Final Report

Arrow: A Configurable RISC-V Vector Accelerator for Machine Learning Applications

Authors:
Imad Al Assir
Mohamad El Iskandarani
Hadi Rayan El Sandid

Supervisor:
Dr. Mazen Saghir

Spring 2021

1 Executive Summary

Recently, the world has witnessed a massive increase in applications requiring Machine Learning and Deep Learning. These applications span multiple areas including healthcare, security, automotive industry, etc. A lot of research effort is currently going into building optimized hardware platforms for these applications and some of these solutions are algorithm-specific accelerators, optimizing applications on GPGPUs, TPUs, systolic arrays, neuromorphic computing and others. While these solutions successfully accelerate their assigned tasks, they are either too specific and thus quickly obsolete, require too much power or are fairly complex and are still in their early phases. One other solution is to use vector processors since these are designed to work on long vectors of data, making them ideal for ML and DL applications. Therefore, our solution is to build a vector accelerator for Machine Learning that is compliant with the recent RISC-V Vector Extension ISA.

The tasks needed to implement this solution are numerous and include: a VHDL description of the accelerator compliant with the RISC-V Vector ISA, an FPGA prototype of the accelerator integrated into a RISC-V core, a gem5 model of the accelerator, modifications to the LLVM/-Clang cross-compiler to optimize code for our hardware.

The team was able to achieve satisfactory progress by testing, simulating, and implementing the vector coprocessor on an FPGA, and benchmarking in software to check for potential application speedups.

Contents

List of Figures

2 Introduction

Recently, artificial intelligence (AI) and machine learning (ML) have become the key component of many applications. From Netflix's recommender system to medical diagnosis tools, ML impacted every field in one way or another. However, ML's effectiveness relies heavily on the presence of huge amounts of data which are becoming increasingly difficult for scalar-based computers to process. It is predicted that the amount of data managed by datacenters in the upcoming decade to increase 50-fold as opposed to the computing resources' increase of 10-fold [1]. Thus, other designs and approaches should be investigated to manage this data explosion.

2.1 Motivation

We are interested in designing custom hardware catered to accelerating ML applications, so we deviate from scalar-based designs to explore a vector-based approach. The foundation of our design is based on the premise that machine learning is largely composed of parallelizable operations, such as matrix multiplication and other linear algebra operations. Exploiting this parallelism through a vector processor design could reap promising performance benefits.

2.2 Overview of Related Work

Other approaches have been to execute ML and DL tasks on tensor processing units (TPUs) or using neuromorphic computing. These solutions are promising but are still in their early stages and are fairly complex. Another approach has been to design accelerators for specific algorithms. While these algorithm-specific accelerators are highly effective, the algorithms they accelerate change very quickly, thus making the accelerator obsolete quickly.

Building a vector accelerator is not novel; these are currently being used in both industry (e.g. Intel AVX-512 vector extensions [2], ARM Scalable Vector Extension (SVE) [3], Andes RISC-V Vector Processor NX27V [4], etc.) and academia (e.g. University of Southampton's AI Vector Accelerator (AVA) [5], ETH Zurich's Ara [6], etc.). However, the industry projects are not open-source/customizable, thus rendering academic research impossible, and the academic projects are either not implemented in hardware (AVA) and/or are meant for High-Performance Computing (HPC) not edge devices (Ara).

2.3 Our solution: Arrow

We present Arrow: a design-time configurable vector accelerator for Machine Learning applications based on the RISC-V Vector Extension. Our design is simulated using gem5, described using VHDL, then implemented on a Xilinx Field Programmable Gate Array (FPGA). It is scalable, configurable at design-time, and achieves 100+ MHz frequency, making it suitable for edge devices.

2.4 Report Structure

In this report, we will first present the background necessary to understand the motivation behind our project as well as its technical details in section 3 and talk about related work in section 4. We will then present our methodology in section 5, design in section 6 and implementation in section 7, before presenting our experiments and results in section 8 and discussing them as well as future work in section 9.

3 Background

Supercomputers have long been synonymous to vector processors. As Valero notes in [7], vector processors have "reigned supreme" from 1975 to 1995. Their success can be attributed to their innovative idea of operating on complete vectors, exploiting data-level parallelism, as well as their ability to hide memory latency and overall energy efficiency –since they have less instructions to fetch and decode. Although superscalar processors took over due to their economic feasibility and increased research effort, current processors are increasingly starting to resemble classical vector architectures. In fact, many processors now support Single Instruction Multiple Data (SIMD) extensions (e.g. Intel AVX-512 [2]) with large vector-like execution units. Even Graphics Processing Units (GPUs), especially AMD's, closely resemble vector processors with their internal SIMD-vector units and their reliance on Single Instruction Multiple Threads (SIMT) – a combination of simultaneous multithreading and SIMD. Moreover, just like current architectures borrowed ideas from vector architectures, the opposite is now true as there have been several recent attempts to revive vector processors by using current techniques such as Out-Of-Order execution. Finally, Valero et al. believe that vector processors could make a comeback in applications that work on long vectors such as Deep Learning and Gene Sequencing. However, they mention that for them to be successful, an ISA should support long vector instructions, and runtime systems and programming models should adapt to this new approach.

The open RISC-V ISA specification aims to fill this gap Valero mentioned by leading an effort towards vector processing through its vector extension [8]. This extension is in active development, and is at version 0.10 (i.e. post-0.9, not 0.1) at the time of this writing. Due to the availability of open-source RISC-V scalar cores and the liberal nature of the ISA, we chose to design our vector accelerator based on this extension.

Multiple tools offer support for the RISC-V ISA and its extensions. This includes the LLVM/-Clang toolchain (current release 12.0) [9], which is a set of open-source compiler tools available under the LLVM project. More particularly, a modified version of the LLVM/Clang toolchain which offers vector intrinsic functions has been developed in the context of the EPI project (European Processor Initiative) [10]. These intrinsic functions are built-in directly into the compiler, and enable us to invoke vector instructions without necessarily having to write assembly routines in our code.

For simulation of RISC-V binaries, there is the gem5 simulator (current release 21.0), which is a modular platform used to simulate computer systems for academic research. It offers parameterized models for many hardware components (i.e. DRAM, CPUs, Caches...), and an API to model custom hardware for use in the simulator. Configuration files are used to specify the structure of our simulated computer-system (i.e. which hardware components to use, how they are connected), and also to specify the general characteristics of the computer-system we would like to simulate (i.e. CPU frequency, memory size, etc.).

The SPIKE simulator [11] allows to run RISC-V binaries, albeit it is more oriented towards testing and debugging than assessing performance. It is possible to attach the GDB/GNU debugger to SPIKE to assist in the debugging process of RISC-V binaries.

The MLPerf benchmark suite [12] contains a set of standardized workloads which can be used to assess the performance of machine learning hardware. One of its variations is the TinyMLPerf [13], which offers a set of benchmarks which can be used to assess the inference performance of edge devices.

4 Related Work

The need for faster customized hardware is a result of the exponentially increasing demand on machine learning applications. In [14], Jouppi et al. note that the increasing usage of voice recognition by consumers required a double in computation power by the servers! Adding more cores and servers wouldn't suffice to keep up with the demand, which made a case for using ASICs with the purpose of accelerating specific computations and algorithms. A recent prime example of such hardware is the tensor processing unit (TPU). The TPU was designed to run as a coprocessor to accelerate the inference stage of neural networks, where it is basically a multi pipeline stage matrix unit that relies on data coming from different directions instead of just one buffer. It yielded incredible performance against CPUs and GPUs in running neural network inference applications and models reaching 40x GPU performance and 80x CPU performance. On a large scale, cost-performance overshadows raw performance, and the TPU manages to achieve 14-34x better performance per watt versus the traditional CPU and GPU servers.

Another noteworthy novel approach to accelerate ML algorithms is Neuromorphic Computing, specifically Intel's Loihi: a Manycore Processor with On-Chip Learning [15]. The design is inspired by how the brain functions and, although some biological mechanisms are hard to replicate, they were able to achieve a preliminary design using simplified abstractions. Loihi is specifically designed for Spiking Neural Networks (SNN), which unlike Artificial Neural Networks "incorporate time as an explicit dependency in their computations". Conventional architectures do not support SNN models well. However, these models could be underappreciated, just like ANNs were before improvements to CPUs and GPUs. Results obtained show that although an ultra-low voltage processor (Intel Atom) outperformed Loihi for a small number of unknowns in a least absolute shrinkage and selection operator (LASSO) operation –which is a method used to enhance the prediction accuracy and interpretability of a statistical model [16]–, Loihi significantly shined for a large number of unknowns with almost 50x less energy, 120x less delay and 5000x less EDP. These preliminary results show the potential for SNNs and neuromorphic computing.

The usage of FPGAs in algorithm acceleration is not novel by any means. As seen in [17], Bai et al. managed to run a 48 node cluster of Zynq FPGAs to accelerate cryptographic algorithms which yielded impressive results against a traditional processor and a many-core server. A key benefit of utilizing FPGAs is exploiting parallelism and energy efficiency, for the cluster achieved 3.6x better energy efficiency than the server using the performance per watt metric.

Several projects similar to Arrow can be found in the literature. One such project is ETH's Ara coprocessor,a tightly coupled 64-bit application core processor that runs on a rather outdated RISC-V version of 0.5[6]. Ara's logic is considered a bit heavy for embedded system applications which increases resource utilization and power consumption. Another noteworthy ongoing project is University of Southampton's Minimal RISC-V Vector Processor for Embedded Systems [18]. It is the only processor with a hardware implementation, and it utilizes a subset of the RISC-V instructions. The processor was implemented at a frequency of 50 MHz on an Intel Cyclone V FPGA, and certain operations such as matrix multiply and filtering were used for testing. Comparing their results to ours would give us a valid idea of Arrow compared to the current literature.

5 Methodology

In this section, we present the methodology we used to approach the design of the vector accelerator.

5.1 Hardware

Our first step was to get a firm grasp of the basics before we could dive into the design phase. We refreshed our memory on vector computing and studied comparable architectures in our literature review. Next, we studied the RISC-V vector instruction set architecture specifications sheet to start planning our design and pointing out any constraints we should abide by.
The design phase was ready to launch. We began coding each component in isolation and testing it on its own. Once we ensured that it was working correctly, we would combine it with another component in another entity. For example, once we made sure each of the vector register file and arithmetic and logic unit (ALU) were functional, we coupled them and tested the newly obtained entity. This bottom up procedure was our approach throughout the design phase. If the testing did not yield the desired results, we would revisit the components and debug to find the root cause of the problem.
Once the VHDL description of the components was complete, the next step was to setup an Advanced eXtensible Interface (AXI) interface [19] whose purpose was to communicate with our processor, since the accelerator was designed to run as a coprocessor with MicroBlaze [20] –a soft-core microprocessor, i.e. that is implemented entirely in the general-purpose memory and logic fabric of Xilinx FPGAs– on FPGA. Once the AXI interface was ready, we tested it in software using the Xilinx Software Development Kit (SDK) [21] and use the Integrated Logic Analyzer (ILA) to debug internal signals.

5.2 Software

First, we have looked into compiler toolchain to be able to cross-compile C/C++ code to RISC-V binaries. We have decided to use a modified version of the LLVM/Clang compiler toolchain, as it also offered support for RISC-V vector instructions. Then, we have used the SPIKE simulator and GDB debugger to verify that our cross-compiled code did not contain any bugs or errors. Once we had verified our code to be bug-free, we used the gem5 simulator to run our cross-compiled binaries on a simulated RISC-V system containing a vector model approximating our accelerator design.

We have repeated this cross-compilation and simulation process multiple time during the development of vectorized routines and benchmarks.

6 Design

In this section, we present our design requirements, specifications and architecture, describing each component on its own before presenting the architecture as a whole.

6.1 Requirements

The project requirements are as follows:

- Accelerator should comply with the latest version of the RISC-V Vector Extension ISA (v0.10 at the time of writing) [8].
- We require the accelerator to run at a clock frequency of at least 100 MHz as a loose lower bound.
- The gem5 model of our accelerator should at least approximate the performance of its hardware counterpart. When used in simulations, it should run at a similar clock frequency (100+ MHz), and should deliver a speedup in most vectorized applications when compared to their serial counterparts.

As for specific constraints such as timing, area, power consumption, they have not been discussed yet as our main focus currently is correctness.

6.2 Deliverables

The expected deliverables at the end of the project are:

- A VHDL implementation of the vector accelerator hardware. This includes a controller, a vector register file, a vector Arithmetic and Logic Unit (ALU) and a memory unit. The datapath must consist of at least two lanes. The processor should also support chaining.
- Support for a subset of instructions specified in ISA. Details can be found in Appendix A.
- Support for special data types for Machine Learning: bfloat16 [22], posits [23].
- Field Programmable Gate Array (FGPA) prototype of the vector accelerator. The accelerator should be integrated within a RISC-V core.
- Implementation of a gem5 model that models our accelerator approximately, and which can be used in gem5 simulations to assess the speedup incurred when our model is introduced in a system to enable vectorized operations.
- Customization of the LLVM/Clang cross-compiler [9] to optimize C code execution on the accelerator, and to offer support for additional data types used in ML kernels (bfloat16, posits), and custom instructions.
- Execution of ML programs on the platform with minor to no modifications.

6.3 Design Elements and Architecture

The vector accelerator design consists of various components. It consists of 5 stages (for simplicity): Decode (split instruction into appropriate fields), Controller (Control signals generation), banked Vector Register File, Execute (ALU) and Write-Back. The processor supports chaining[1] between ALU lanes themselves, and between memory and ALU. This technique aims to increase computational speed by taking the result directly from the generating unit, thus skipping additional register file bank accesses and avoiding Read-After-Write (RAW) hazards. We describe below the design of each component, before showing the full architecture.

[1] Chaining is not yet supported but will be later on.

6.3.1 Decoder

The decoder is a purely combinational circuit that takes the 32-bit instruction received from the scalar processor and decomposes it into the appropriate fields required by the other components.

6.3.2 Controller

The controller is the brain of the vector processor. It receives the decoded instruction and generates the appropriate control signals for each vector lane.

6.3.3 Vector Register File

The vector register file (VRF) holds the 32 architectural vector registers, as well as the offset generators that generate the read/write byte offsets within 1 vector register and appropriate write enable for each byte. The 32 vector registers are currently divided into 2 banks with 2 read ports and 1 write port each, but this distribution differs based on the number of lanes. The max length of the vector register, denoted by VLEN, can be set at design time. Figure 1 illustrates the VRF architecture.

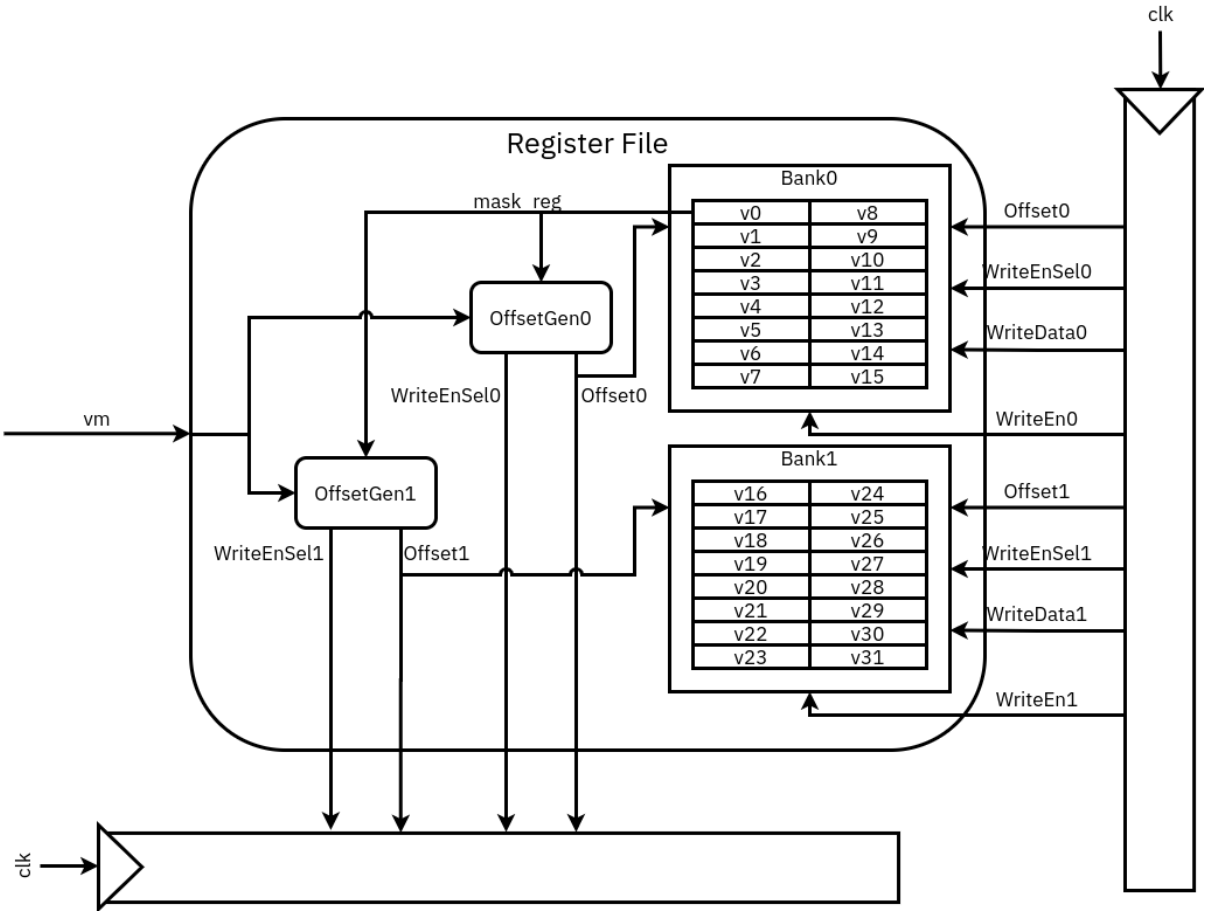

Figure 1: Vector Register File Design

6.3.4 ALU

The ALU was originally designed to compute one element per cycle,and the operands were ELEN bits wide, where ELEN is the maximum size of an element (1024 bits). Thus, if the incoming operands were 8 bits wide for instance, the unit would be heavily underutilized. Increasing computation parallelism would greatly improve performance and throughput.

First, we had to agree on the transfer size per cycle, and the consensus was on 64 bits, meaning 64 bits of data (per operand) will be transferred from the register file bank to the ALU

per cycle regardless of the element size. This entails that a 64 bit chunk could hold 8 contiguously stored 8 8-bit elements, or 4 16-bit elements, or 2 32-bit elements, or 1 64-bit element, or a chunk of higher sized elements. This allows room for parallelism, where depending on the element size, the ALU operates on different bits and achieves up to 8x throughput. Figure 2 illustrates how the same 64-bit chunk is viewed and computed differently based on the element size.

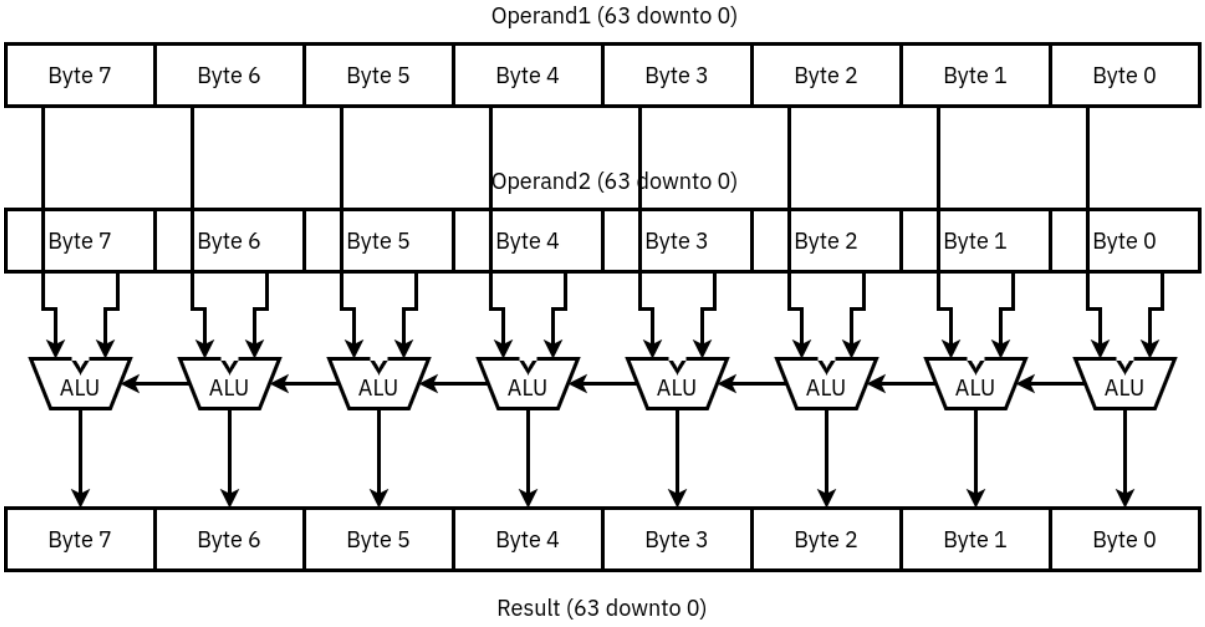

Figure 2: Sample SIMD ALU operation

6.3.5 Memory

Memory operations are divided into several steps:

- **Address generation on the Arrow side**: performed by the memory generator unit. In this step, this unit is responsible for generating the base memory address for a memory burst transaction, appropriate store data, and appropriate byte enable signals (for non-unit stride accesses). More on this in section 7.
- **Issuing memory transactions on the AXI side**: performed by an AXI master interface. In short, this component is responsible for translating the memory request received from Arrow to an AXI transaction.
- **Translating AXI transaction to memory transaction**: performed by the Memory Interface Generator (MIG). After travelling through the AXI interconnect, the AXI transaction is received by MIG. This component consits of basically a memory controller for the on-FPGA DDR3 memory, and an AXI slave to interface with the system (through AXI). In layman terms, its role is to translate the AXI transaction to an operation (or set of operations) the DDR3 understands.
- **Receiving data**: in case of loads, the data travels through the components mentioned above in the opposite direction before reaching Arrow. When it does, the memory generator unit is responsible for taking the data received and saving them in the appropriate registers, at appropriate element offsets.

6.3.6 Scalability

As mentioned, Arrow is designed to be scalable with configurable number of operating lanes. The ISA states that there are 32 vector registers, so in our current 2 lane design, each bank is

composed of 16 vector registers. Thus, each bank would hold 32/NB_LANES vector registers, so if we were to utilize 4 lanes, each bank would hold 8 vector registers.

6.3.7 Overall Architecture

The overall architecture can be seen in Figure 3 on a separate page.

6.3.8 Software Component Workflow

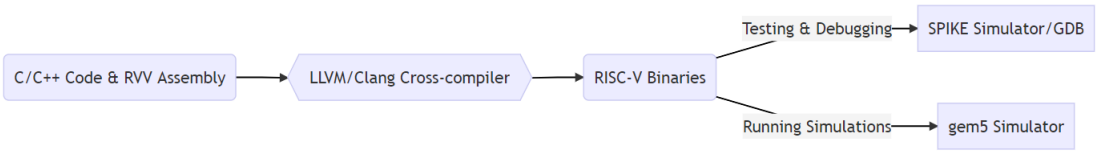

Figure 4: Software Component Workflow

There are three main stages in the software component of the project, as represented in the workload in Figure 4 : The development of C/C++ code, which calls RISC-V vector subroutines, the use of the LLVM/Clang cross-compiler to compile the code into RISC-V binaries, and the testing and simulation process using these binaries and additional benchmarks with SPIKE/GDB and the gem5 simulator.

6.3.9 Benchmarks Development & LLVM/Clang Compiler Toolchain

Since we cannot compile natively for RISC-V on our machines, we have used the LLVM/Clang toolchain to cross-compile our C/C++ code to RISC-V binaries.

To assess the performance of a system with our accelerator, we have designed and cross-compiled a set of vectorized routines which were mostly based on the works released publicly by the University of Southampton [5]. We have also cross-compiled benchmarks released by the Barcelona Supercomputing Center [24], as they supported RISC-V and its vector instructions.

6.3.10 gem5 Simulator Model

We have used the gem5 simulator to design a complete system containing a scalar-core and memory components, as well as a vector accelerator, which would approximately model our design. The simulation runs on this system allowed us to assess the performance of vectorized applications in the presence of our vector model.

6.4 Design Decisions

In this section, we group some decisions we have made and why we have made them. They are presented in a list form, for the convenience of the reader. Note that most of them are temporary and will be improved upon in the future.

- **Instruction scheduling**: should be done manually or supported by the compiler because there is no instruction queue or any scheduling mechanism in Arrow itself. We chose to do so to be able to focus on the correctness of these instructions' execution instead of optimizations.
- **Support for integers only**: although one of our requirements was to support special datatypes useful for ML acceleration (e.g. bfloat16, posits, etc.), we chose to avoid them for now, to be able to focus on the general architecture.

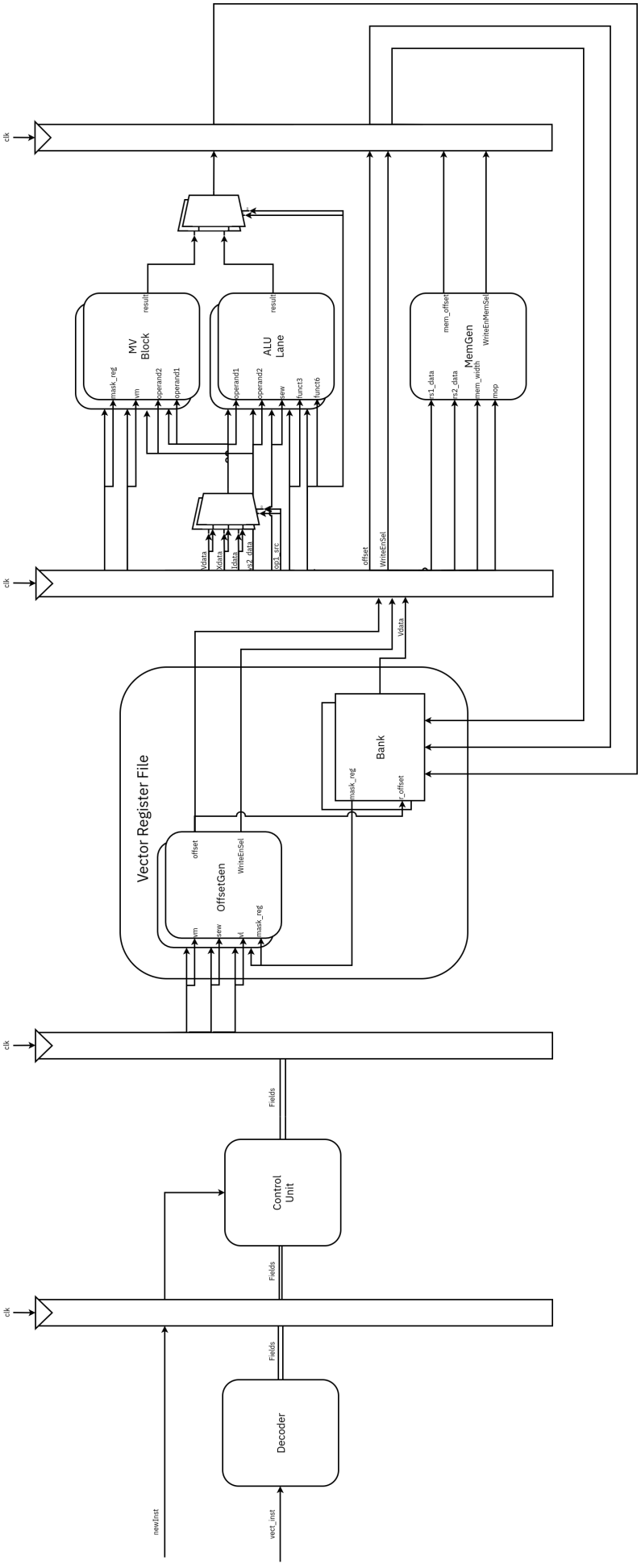

Figure 3: Datapath without Memory

- **Working on subset of ISA**: we chose to implement only a subset of the RVV spec, also to be able to focus on the general architecture.
- **Support for custom instructions**: in its nature, the RISC-V ISA supports custom instructions. While some of these can help us achieve better performance on ML applications, they were also deemed not a priority for the scope of the FYP.
- **Support for vector length (VLEN) up to 256 bits**: supporting larger VLEN only requires adding cases in the register file banks.
- **Dedicated banks per lane**: Each lane can only read/write to a specific bank of the VRF. This is not ideal, and can be improved by sharing banks across lanes. However, this requires arbitration logic which was not our priority.
- **Memory operations**: Memory operations can be a bottleneck in every architecture, so it is crucial to implement them properly. Unfortunately, we did not have the time to implement them efficiently, as will be discussed in section 7.1.5. Unit-stride and strided operations are supported, while indexed operations are yet to be implemented.
- **Using the LLVM/Clang Toolchain** : We have decided to use the LLVM/Clang toolchain as a cross-compiler, due to its support for RISC-V vector instruction and vector intrinsics.
- **Implementing custom instructions in LLVM/Clang** : The flexibility of the LLVM/-Clang codebase allows us to implement custom instructions with relatively low effort. In future stages of the project, we plan to implement support for additional data types related to ML applications (bfloat16, posits).
- **Using the U. of Southampton Benchmarks** : Rather than start from scratch when developing our benchmarks, we have decided to use the work released publicly by the U. of Southampton, and extend on it.
- **Writing vectorized benchmarks in Assembly** : While the LLVM/Clang toolchain offers vector intrinsics that could be used when developing benchmarks, we have decided to rather write them ourselves in Assembly code, as to avoid overhead in our benchmarks.
- **Using gem5 for Simulations** : The gem5 simulator has support for the RISC-V ISA and offers relatively accurate results. It can be configured to model a variety of popular computer-architecture systems.
- **Using existing Vector Model** : Due to the complexity of the gem5 simulator, implementing a vector architecture from scratch would have been very time-consuming. We have decided to use a gem5 vector model developed by the Barcelona Supercomputing Center (BSC) to approximate our own vector architecture.
- **Using BSC Benchmarks for Simulations** : Due to compatibility issues with the gem5 vector model developed by the Barcelona Supercomputing Center, we have decided to use a set of vectorized benchmarks they have released in our simulations. While these benchmarks do not assess the performance of our model in ML-related application, they at least display the potential speedup that can be gained when vectorized general purpose applications.

7 Implementation

After talking about the overall design and describing how components should function, we now move towards implementation details. These are slightly more technical and are more detailed than the design section.

7.1 Implementation details

In this section, we go over the implementation of the components designed in Section 6.

7.1.1 Controller

The controller is composed of two components:

- Control unit: receives the fields from the decoder and sets control signals. Note that the control unit is blind to the notion of lanes in our design; it functions in a scalar fashion. The multiplexing between lanes is done by the following component.
- Lane selection: in order to determine what lane to propagate the control signals to, we have to rely on the vector register numbers read and written by the instruction. If the vector registers read and written by the instruction belong to the [0,15] bracket, the first lane is selected. If the vector registers read and written by the instruction belong to the [16,31] bracket, the second lane is selected and the control signals are set in their appropriate bit numbers of the Controller's output ports.

7.1.2 Vector Register File

Note that we chose a banked design with 2 read ports and 1 write port each, to be able to service both lanes efficiently. The 32 registers required by the ISA were split into 2, with each bank containing respectively the lower and upper 16 vector registers. There could have been other ways to split the banks including an odd/even distribution or other more complex methods, but this method was chosen for its simplicity. The implication here is that the compiler has to choose registers that map to banks in an alternating manner. In other words, if a vector add instruction is to be executed, the compiler maps the operands to registers from the lower bank. Next, if a vector subtract instruction is to be executed, the compiler should map the operands to registers from the other bank to avoid structural hazards. Note that there could have been a method to arbitrate that in hardware on-the-fly, but it requires more complex hardware that is currently not our priority.

The upgrade in the offset generator ripples changes to other parts of the design, including the register file banks. Since WriteEnSel's bits indicate if the corresponding byte is to be written or not, the bank's dependency on the element width becomes useless, which allows us to greatly reduce the logic needed for the bank implementation.

7.1.3 SIMD ALU

We expect our ALU to be synthesized as a cluster of smaller 8-bit ALUs, where these collaborate (i.e. propagate carry across units) in case SEW ¿ 8 bits. Note that this is what we are expecting the synthesis tool to infer, but it might infer a less efficient implementation: 8 8-bit, 4 16-bit, 2 32-bit, and 1 64-bit ALU without collaboration between them. Also note that this could be guaranteed by specifying flags for the synthesis tools to ensure that the components are inferred as expected, but due to time limitations, we were unable to do that, so we relied blindly on the tools. It is, however, on our to-do list.

7.1.4 Offset Generator

Previously, our offset generator could output one element bit mask per cycle. This approach is no longer feasible, for we can now operate on up to 8 elements per cycle due to our 64-bit chunk. Because of this change, we introduced a new signal, WriteEnSel, whose bits indicate if the corresponding byte of WriteData is to be written or not. Figure 5 demonstrates this mapping through a sample write.

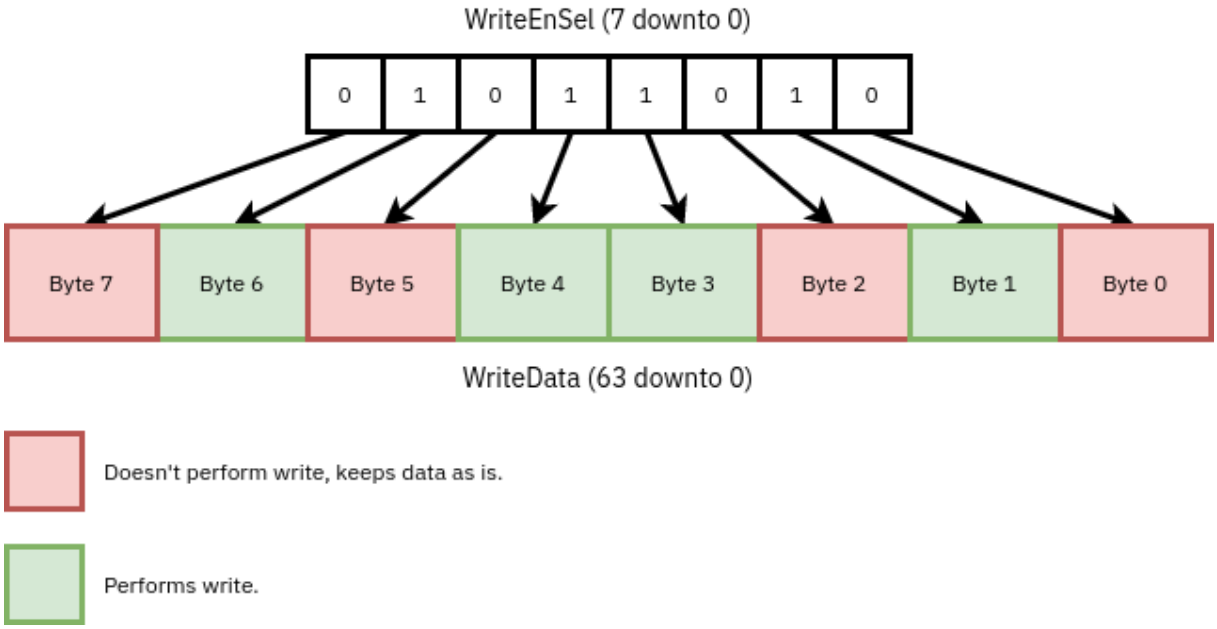

Figure 5: Sample write to VRF bank with WriteEnSel

An additional crucial improvement over the previous implementation is that the done signal is now set with the last offset, not the cycle after it, as it was previously. This detail is important for software testing and more efficient performance, since the done signal was set one cycle late in the previous design, so an unnecessary cycle delay has been omitted.

7.1.5 Memory

There are several addressing modes for memory operations: unit-stride, strided (fixed stride) and indexed (stride read from a vector register).

Regarding the memory generator on the Arrow side, each addressing mode was dealt with differently:

- Unit-stride: the logic behind the unit stride is identical to that of the offset generator. Since accesses are contiguous, the memory generator sets the write enable bits depending on element size and mask bits.
- Strided: the logic behind strided operations is different, as each element is separated by an offset as opposed to contiguous elements. The logic is made even more complex by the fact that the number of elements per transaction and stride change the number of useful bytes per cycle. So, on every cycle, the memoy generator calculates how many elements (or fragments of elements) are found in this current cycle, and it updates global counters that keep count of how many elements have been processed so far and how many elements are left.
- Vector indexed: is yet to be fully supported and tested.

Regarding the memory interface, the following are the implementation details of each addressing mode:

- Unit-stride: The unit-stride is easy to implement and basically requires reading a specific number of bytes located contiguously in memory. In our case, since the AXI data width and memory bus on the Arrow side have the same size (64 bits), we basically have to keep shipping 8 bytes at a time until we reach the required amount of bytes.

- Strided and vector indexed: for the strided and indexed addressing modes, the full 8 bytes might not be needed, as the addresses might not be contiguous in memory. For loads, this will lead to reduced throughput as we might not need all the data loaded from memory. However, this is especially a problem for stores as writing the full 8 bytes might lead to overwriting potentially useful data. This happens when the data to be stored is less than 8 bytes and the store addresses are not contiguous in memory. To deal with this, AXI supports narrow transactions, which basically allow accesses that are less than the full AXI data width. These narrow transactions solve the problem of overwriting potentially useful data, as it enables writing only the valid part of the bus (containing useful data) instead of the full bus. However, due to time limitations, we were not able to support narrow transactions, and thus we have to deal with the assumption that non-contiguous stores might overwrite data around the store address.

7.1.6 FPGA Implementation

The FPGA implementation is as follows. The Arrow datapath is enclosed within an AXI IP to allow it to communicate with the scalar processor (MicroBlaze for now) and memory. Note that we chose to implement a separate IP for the component that translates memory requests to AXI transactions in order to make the Arrow IP as generic as possible. Also note that the DDR3 (interfaced through MIG) is shared across MicroBlaze and Arrow. A diagram of the FPGA design can be seen in Figure 6.

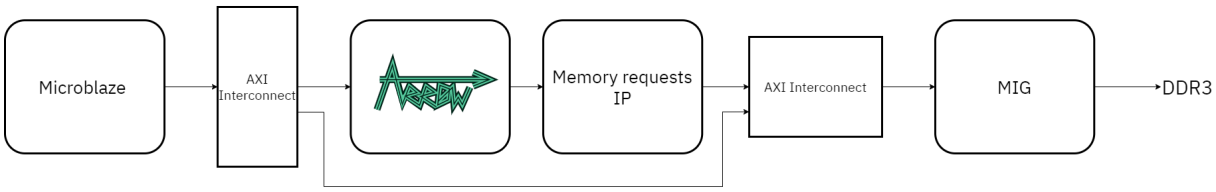

Figure 6: FPGA design block diagram

7.1.7 Benchmarks Development

When we first started writing vectorized code in C/C++, we either wrote in-line assembly or separate assembly files to invoke subroutines using RISC-V vector instructions. Writing these subroutines by hand was time-consuming, so we decide to look for any previous work which might have already completed the implementation.

We have found an implementation of vectorized functions realized by a group of students at the university of Southampton, which they have released publicly. We have based ourselves on their work to develop our vectorized subroutines, making minor modifications to update this code to both fit our hardware and the updated version of RVV we were using (from RVV v0.8 to RVV v0.9). There are still some modifications that need to be made before we can use these vectorized subroutines in larger Machine-Learning related programs (i.e. matrix arithmetic, convolution operations) to benchmark the performance of our vector accelerator model. The bulk of these assembly routines is shown in Figure 7.

7.1.8 LLVM/Clang Compiler Toolchain

To optimize vector code for our hardware, we have made use of in-line RISC-V assembly calls in our C/C++ code to have more control over the flow of operations. This modification of

Function Name	Description
vect_add	Vector Addition
vect_mult	Vector Multiplication
vect_addReduction	Vector Reduction (Type #1)
vect_maxReduction	Vector Reduction (Type #2)
vect_dotProduct	Vector Dot Product
vect_ReLu	Vector Rectified Linear Unit

Figure 7: Set of Vectorized benchmarks released by U. of Southampton

LLVM/Clang also offered a wide variety of intrinsics which could be used to call vector instructions without being required to write RISC-V vector instructions in in-line assembly every time. However, this came at the cost of optimization and overhead in the code as this approach did not give us as much control over the flow of operations. These intrinsics are documented on the repository hosting the code to the modified LLVM/Clang fork, which was quite useful during the programming process. Figure 8 shows an implementation of vector multiplication using vector intrinsics, and Figure 9 shows an implementation of the same vector multiplication algorithm using assembly.

```
void vec_mul(long N, int *c, int *a, int *b) {
  long i;
  for (i = 0; i < N;) {
    long gvl = __builtin_epi_vsetvl(N - i, __epi_e32, __epi_m1);
    __epi_2xi32 va = __builtin_epi_vload_2xi32(&a[i], gvl);
    __epi_2xi32 vb = __builtin_epi_vload_2xi32(&b[i], gvl);
    __epi_2xi32 vc = __builtin_epi_vmul_2xi32(va, vb, gvl);
    __builtin_epi_vstore_2xi32(&c[i], vc, gvl);
    i += gvl;
  }
}
```

Figure 8: Vectorized Matrix Multiplication using LLVM/Clang Intrinsics

```
vect_mult:
        .globl   vect_mult
        vsetvli t0, a0, e8, m1, d1 # Set vector length
        vle.v v1, (a1) # Get first vector
        add a1, a1, t0 # Bump pointer
        vle.v v2, (a2) # Get second vector
        add a2, a2, t0 # Bump pointer
        vmul.vv v3, v1, v2 # Multiply vectors
        sub a0, a0, t0 # Decrement number done
        vse.v v3, (a3) # Store result
        add a3, a3, t0 # Bump pointer
        bnez a0, vect_mult # Loop back
        ret # Finished
```

Figure 9: Vectorized Matrix Multiplication using assembly code

Since we have started using the U. of Southampton project's work as a base for our vectorized functions, we have definitely discarded the use of intrinsic functions in our code, as they introduce a certain level of overhead which might be problematic in later stages when we try to design performant vectorized benchmarks related to Machine-Learning.

The flexibility of the LLVM/Clang codebase has also allowed us to test support for additional custom instructions. We have only implemented dummy instructions to make sure we understand

the process, but we plan on adding support later for additional instructions related to data types like bfloat16 and posits which are becoming more common-use in ML kernel implementations.

7.1.9 gem5 Simulation Model

We have opted to run our gem5 simulations in system-call emulation mode (i.e. All system calls are emulated, there is no OS running) rather than full-system emulation (i.e. True simulation, where an OS must be provided for the simulated system to function). While system-call emulation delivers less accurate results than full-system emulation, its advantage is that there is no need for an OS to run in real time on the system as all system-calls are emulated. Figure 10 shows a sample gem5 simulation in system-call emulation mode.

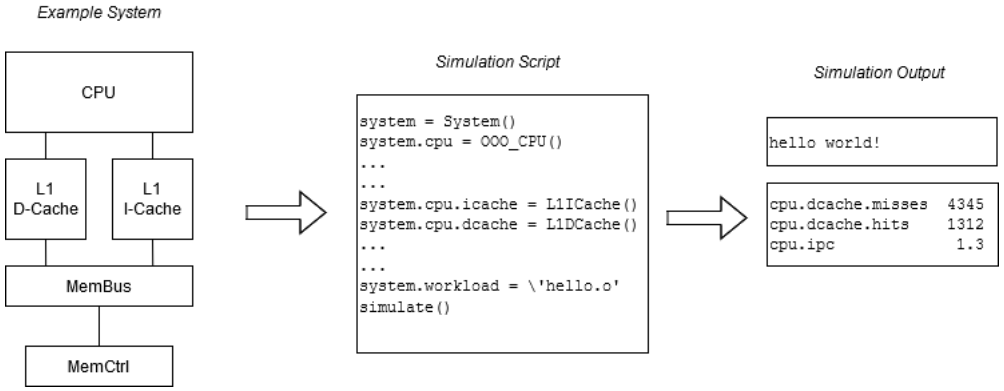

Figure 10: Sample system-call emulation simulation in gem5

To model our vector accelerator unit in gem5, we have used an existing design by the Barcelona Supercomputing Center which could be configured to approximate our own vector unit. This model has been designed with HPC applications in mind, but it can be re-configured to approximate our own vector unit design. It is possible to modify parameters in gem5's configuration files to set the number of vector registers, the number of vector lanes, the clock of the vector accelerator, the maximum vector length, and much more. Figure 11 shows a diagram of our gem5 simulation system with the vector model.

The model has some notable restrictions we had to work around, as the supported values for SEW are 32/64-bits, the supported value of LMUL is 1 only, and 4-byte words are the smallest addressable units possible. Masked memory operations, as well as widening and narrowing vector instructions are also not currently supported. Major architectural differences are also present, like the implementation of a re-order buffer, and a ring lane interconnection topology. Some of these options were disabled to better approximate our vector unit. Figure 12 shows a diagram of the BSC vector engine model, and its architectural features.

Also, the RVV extension version currently supported by this vector model is RVV v0.7.1, as opposed to our targeted RVV v0.9. This does not necessarily translate to a difference in performance, but it does have an effect on the benchmarks we can run on this model, since some instructions differ (especially those related to Control-Status registers). Therefore, we have decided to use a vectorized benchmark suite provided by the Barcelona Supercomputing Center alongside this model to perform simulations in gem5 and assess the potential speedup of serial vs vectorized applications on a model configured to approximate our vector unit.

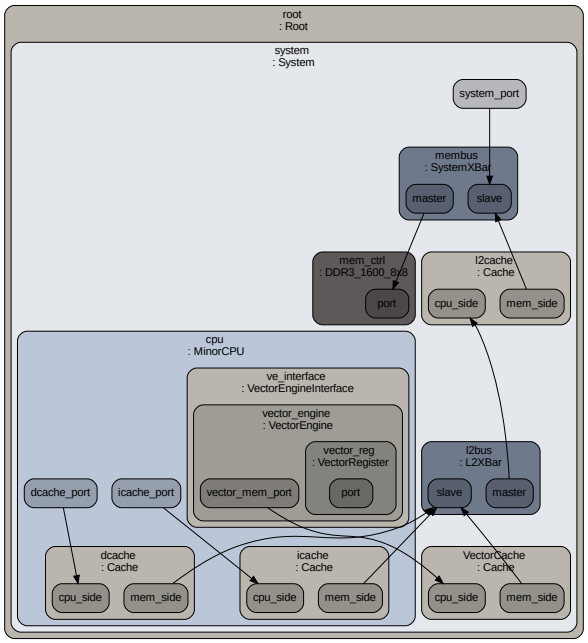

Figure 11: Diagram of our gem5 Simulation System with the vector model

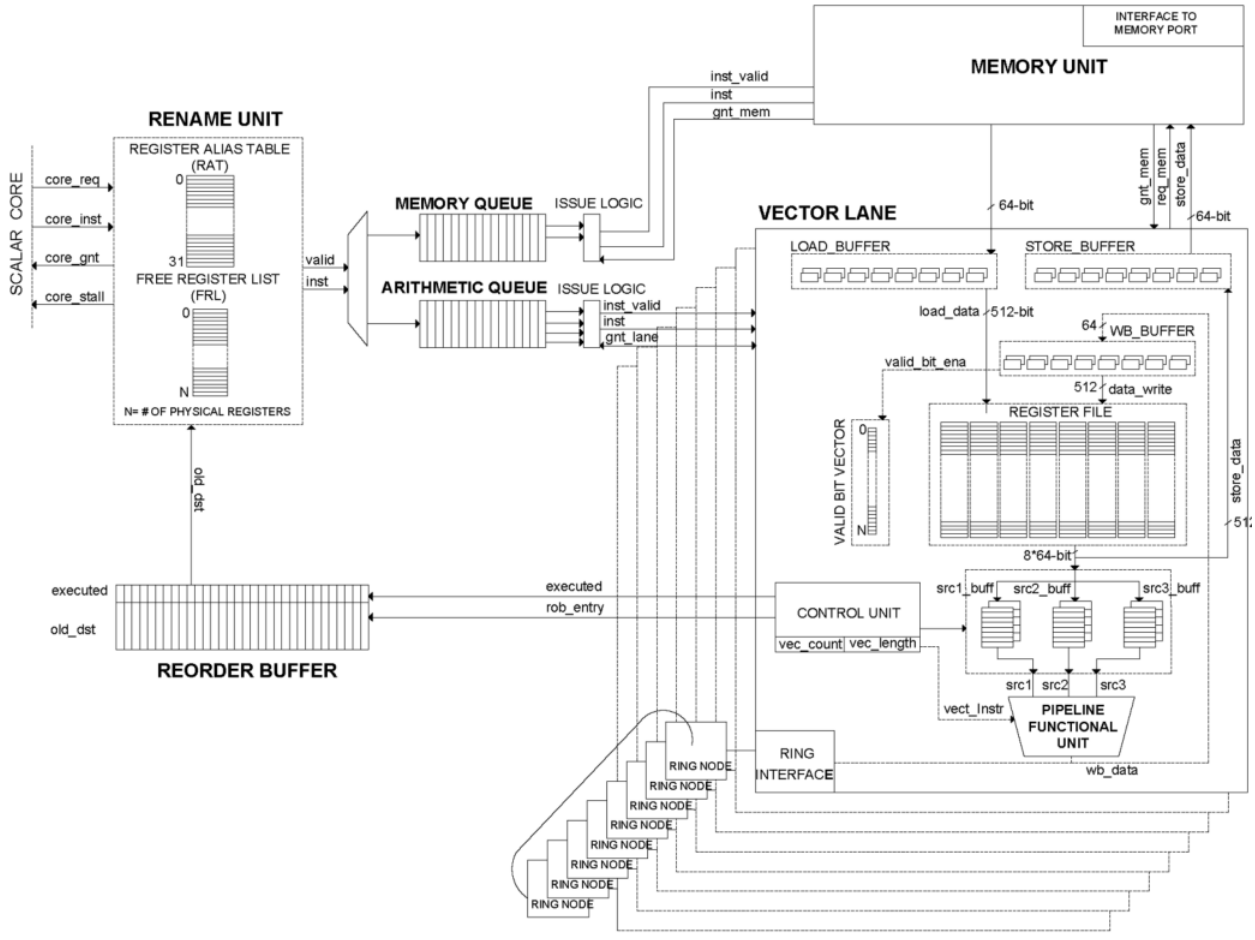

Figure 12: BSC vector engine Diagram (taken from the BSC paper on the model)

8 Experiments and Results

In this section, we present the experiments performed and the results obtained.

8.1 Hardware

Prior to implementation, extensive debugging and simulation were performed to ensure the timing and correctness of each component and their combinations. It would be redundant to show simulation waveforms, as our arrival at the implementation stage below is proof of the correctness of our previous testing.

In order to find Arrow's utilization and power consumption, we compared MicroBlaze alone against Arrow coupled with MicroBlaze. Both setups were implemented at 100 MHz. The results can be seen in Figures 13 and 14, respectively.

8.1.1 MicroBlaze

- **Resource utilization**:

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	2241	133800	1.67
LUTRAM	204	46200	0.44
FF	1495	267600	0.56
BRAM	32	365	8.77
IO	4	285	1.40
BUFG	3	32	9.38
MMCM	1	10	10.00

Figure 13: MicroBlaze Post-Implementation Utilization Summary

- **Power consumption**: 0.27W

8.1.2 Arrow with MicroBlaze

- **Resource utilization**:

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	2715	133800	2.03
LUTRAM	204	46200	0.44
FF	2268	267600	0.85
BRAM	32	365	8.77
IO	4	285	1.40
BUFG	3	32	9.38
MMCM	1	10	10.00

Figure 14: Arrow with MicroBlaze Post-Implementation Utilization Summary

- **Power consumption**: 0.297W

By comparing the values obtained, it is safe to say that Arrow's consumption is rather insignificant, where the lookup tables (LUT) consumption witnessed an increase of 21% and flipflop (FF) consumption increased by 51%. As for power consumption, it is very reasonable.

8.2 Software

To assess the performance of our hardware unit, we have ran a set of gem5 simulations on a simulated system containing the BSC vector accelerator unit, configured to approximate the performance of our Arrow unit, as shown in Figure 15. A key point was to verify that we would obtain a speedup in execution if the scalar core and our accelerator unit ran at the same expected frequency as our hardware implementation (i.e. 110MHz).

gem5 System Parameters	**Value**
Scalar Core - CPU Type	In-Order
Scalar Core - Clock Cycle	110 MHz
System - DRAM Size	2048 MB
System - Caches used	Yes (L1/L2 for Scalar core)
Vector Unit - Number of Lanes	4
Vector Unit - Clock	110 MHz
Vector Unit - Max Vector Length	128 Bytes
Vector Unit - ReOrder Buffer	Disabled
Vector Unit - Number of Physical Vector Registers	40

Figure 15: Parameters of our gem5 system

We have used the benchmarks offered by the BSC alongside their gem5 vector unit model to assess its performance with these parameters (Figure 16. Each benchmark is available in a serial format (i.e. non-vectorized), as well as in a vectorized format making use of RISC-V vector instructions. While none of these benchmarks are particularly catering to ML-related

applications, they can at least allow us to observe any speedup in serial vs vectorized applications with the introduction of our vector design. These benchmarks contain input size preset : *Tiny - Small - Medium - Large - Native.*

Application	**Algorithmic Model**	**Benchmark Suite**
Blackscholes	Dense Linear Algebra	PARSEC/PARVEC
Canneal	Unstructured Grid	PARSEC/PARVEC
ParticleFilter	Structured Grid	Rodinia
PathFinder	Dynamic Programming	Rodinia
StreamCluster	Dense Linear Algebra	PARSEC/PARVEC
Swaptions	MapReduce	PARSEC/PARVEC

Figure 16: Benchmarks used in simulations

We have decided to test our benchmarks only with the *Tiny* and *Small* presets, due to the exponentially long simulation times taken by others input size presets (i.e. upwards of 40 hours). We assess the runtime of each program in its entirety, meaning the initialization code and the kernel/operations themselves. Figure 17 and 18 show the simulation times we have obtained, and Figure 19 displays the speedup ratio when comparing the serial and vectorized versions of each simulated applications.

Application	**Runtime - Serial version**	**Runtime - Vectorized version**
Blackscholes	549.6 milliseconds	41.5 milliseconds
Canneal	7.4 milliseconds	6.2 milliseconds
ParticleFilter	200.5 milliseconds	199.9 milliseconds
PathFinder	31.5 milliseconds	3.5 milliseconds
StreamCluster	2109 milliseconds	357.8 milliseconds
Swaptions	773.0 milliseconds	194.9 milliseconds

Figure 17: Simulation Results for preset *Tiny*

Application	**Runtime - Serial version**	**Runtime - Vectorized version**
Blackscholes	4438.7 milliseconds	358.4 milliseconds
Canneal	-	-
ParticleFilter	1533.8 milliseconds	1394.9 milliseconds
PathFinder	4651.6 milliseconds	514.2 milliseconds
StreamCluster	-	-
Swaptions	6171.3 milliseconds	1552.0 milliseconds

Figure 18: Simulation Results for preset *Small*

As observed, the speedup ratio for different vectorized applications and input sets range from **1.2x** to **13x** faster execution. The speedup ratio for different vectorized applications and input sets range from **1.2x** to **13x** faster execution. Since we have observed a relative speedup in most of these general-purpose applications, we expect to see a speedup in later stages of the project when we perform test using ML-related benchmarks.

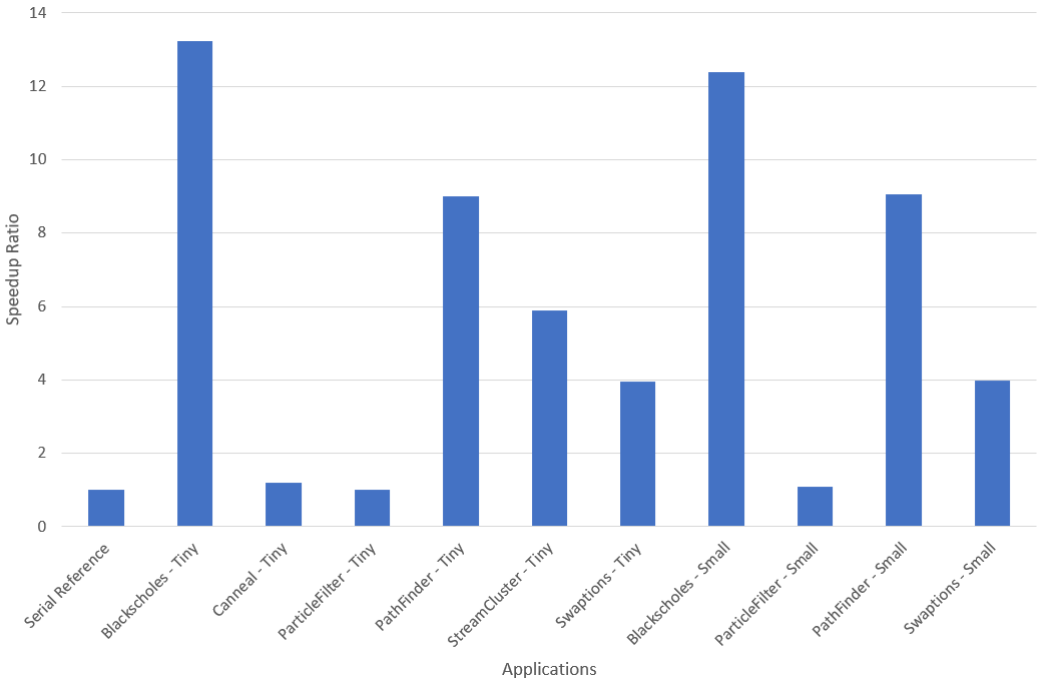

Figure 19: Speedup ratio when comparing serial vs vectorized for each simulated application

9 Discussion and Future Work

Comparing our results to that of USouthampton's processor, Arrow's clock frequency of 100 MHz is double that of USouthampton's[18]. In addition, with Arrow's fixed 64 bit inputs, it can simultaneously run twice as many operations as USouthampton due to its 32-bit fixed input.

The future is bright for Arrow; we merely set the base with our FYP, and there is still a lot to be done. Some future tasks will be including a posit/bfloat16/Floating Point Unit, adding support for custom instructions useful for ML, implementing instruction scheduling mechanisms, improving the memory system by adding a store buffer for example and supporting narrow transactions, writing a script to automatically generate an instance of the vector architecture given set specifications (e.g. number of lanes, max element width, max vector length, etc.), integrating the design into a RISC-V core, benchmarking using ML applications on the hardware itself, and many more.

10 Conclusion

In summary, we were able to design a vector coprocessor both in hardware and as a software simulation model. The results were quite promising compared to the literature, where Arrow achieved a clock frequency of 100 MHz and significant benchmark speedups ranging between **1.2x** and **13x** compared to scalar processors. Arrow is still at its start, and its promising power will start to emerge once further work in ML testbenching is performed.

11 Constraints and Applicable Standards

In this section, we present the technical and non-technical constraints, as well as standards that we abide by.

11.1 Technical Constraints

Technical constraints include:

- Compliance with the RISC-V Vector ISA specifications.
- Programmability of the accelerator in C using appropriate intrinsics.
- Operation of the hardware prototype at a clock rate of 100 MHz or better.
- Support for special data types for Machine Learning: bfloat16, posits.
- Utilization of Xilinx tools and prototyping on a Nexys Video FPGA [25].
- Achievement of better performance and less energy consumption in gem5 simulations of vectorized versions of ML kernels and benchmarks on Arrow compared to the scalar version of these kernels.
- Ability of the hardware design to scale to support more lanes and/or execution units.

11.2 Non-Technical Constraints

Non-technical constraints include:

- Availability of FPGA and software programs in ECE labs.

11.3 Applicable Standards

The standards that we abide by include:

- **ARM AMBA AXI4** [19]
- 1076-2019 - **IEEE Standard for VHDL** Language Reference Manual [26]
- **RISC-V Vector Extension ISA**

References

[1] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys*, vol. 48, pp. 1–33, May 2016.

[2] Intel, *Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Overview.*

[3] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, and et al., "The arm scalable vector extension," *IEEE Micro*, vol. 37, p. 26–39, Mar 2017.

[4] A. T. Corporation, "Andes RISC-V Vector Processor NX27V Is Upgraded to RVV 1.0." https://www.globenewswire.com/news-release/2020/12/02/2138618/0/en/Andes-RISC-V-Vector-Processor-NX27V-Is-Upgraded-to-RVV-1-0.html, Dec. 2020.

[5] British Computer Society Open Source Specialists, "Expanding a RISC-V Processor with Vector Instructions for Accelerating Machine Learning," Feb. 2021.

[6] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, "Ara: A 1 GHz+ Scalable and Energy-Efficient RISC-V Vector Processor with Multi-Precision Floating Point Support in 22 nm FD-SOI," *arXiv:1906.00478 [cs]*, Oct. 2019.

[7] J. Dongarra, V. Getov, and K. Walsh, "The 30th Anniversary of the Supercomputing Conference: Bringing the Future Closer—Supercomputing History and the Immortality of Now," *Computer*, vol. 51, pp. 74–85, Oct. 2018.

[8] RISC-V, *Working Draft of the Proposed RISC-V V Vector Extension.*

[9] *The LLVM Compiler Infrastructure.* 2020.

[10] "Roger Ferrer / llvm-epi." https://repo.hca.bsc.es/gitlab/rferrer/llvm-epi.

[11] "Riscv/riscv-isa-sim." RISC-V, Apr. 2021.

[12] *MLperf.* Nov. 2020.

[13] "Mlcommons/tiny." MLCommons, Apr. 2021.

[14] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ACM, June 2017.

[15] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, pp. 82–99, Jan. 2018.

[16] *Lasso (Statistics).* Wikimedia Foundation, Oct. 2020.

[17] X. Bai, J. Yang, Q. Dai, and Z. Chen, "A hybrid ARM-FPGA cluster for cryptographic algorithm acceleration," *Concurrency and Computation: Practice and Experience*, vol. 31, Aug. 2019.

[18] M. Johns and T. J. Kazmierski, "A minimal risc-v vector processor for embedded systems," in *2020 Forum for Specification and Design Languages (FDL)*, pp. 1–4, 2020.

[19] ARM, *AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite*.

[20] Xilinx, *Xilinx MicroBlaze Processor Reference Guide*. 2020.

[21] Xilinx, *Xilinx Software Development Kit (SDK)*. 2020.

[22] *BFloat16: The Secret to High Performance on Cloud TPUs — Google Cloud Blog*. Google.

[23] J. Gustafson and I. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, 2017.

[24] C. Ramírez, C. A. Hernández, O. Palomar, O. Unsal, M. A. Ramírez, and A. Cristal, "A risc-v simulator and benchmark suite for designing and evaluating vector architectures," *ACM Trans. Archit. Code Optim.*, vol. 17, Nov. 2020.

[25] Digilent, *Nexys Video Reference Manual*.

[26] "IEEE Standard for VHDL Language Reference Manual," *IEEE Std 1076-2019*, pp. 1–673, Dec. 2019.

[27] Xilinx, *Xilinx Vivado Design Suite*. 2020.

[28] J. Lowe-Power, *Main Page: Gem5*.

[29] "GDB: The GNU Project Debugger." https://www.gnu.org/software/gdb/.

[30] "The PARSEC Benchmark Suite." https://parsec.cs.princeton.edu/.

[31] J. M. Cebrian, M. Jahre, and L. Natvig, "ParVec: Vectorizing the PARSEC benchmark suite," *Computing*, vol. 97, pp. 1077–1100, Nov. 2015.

[32] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54, Oct. 2009.

[33] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *IEEE International Symposium on Workload Characterization (IISWC'10)*, pp. 1–11, Dec. 2010.

A Tools and Resources Used

The tools and resources used include:

- **Xilinx Vivado Design Suite**: used to write and simulate VHDL code. Imad and Mohammad were familiar with this tool. [27]
- **Xilinx Software Development Kit (SDK)**: used to write C/C++ code and execute them on FPGA. [21]
- **Nexys Video FPGA**: used for hardware prototyping. The FPGA was acquired from the ECE labs.
- **gem5 Simulator**: used for simulating the vector accelerator in a system, benchmarking it and comparing it to other designs. The gem5 simulator is open-source, and its codebase is publicly available. [28]
- **SPIKE Simulator** : used to test binaries cross-compiled to our RISC-V target, and assess the functionality of our code. The SPIKE simulator is open source, and its codebase is publicly available. [11]
- **GDB Debugger**: used as a portable debugger alongside SPIKE. The GDB Debugger is open source, and its codebase is publicly available. [29]
- **LLVM/Clang Toolchain**: The LLVM/Clang Toolchain is open-source, and its codebase is publicly available. [9]
- **MLPerf Benchmark Suite**: used to benchmark the performance of our hardware with ML-related workloads. The MLPerf benchmark suite is open-source, and its set of repositories are publicly available. [12]
- **PARSEC/PARVEC Benchmark Suites**: The Princeton Application Repository for Shared-Memory Computers (PARSEC) is a benchmark suite composed of multithreaded programs. The suite focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors [30]. PARVEC is a vectorized version of this benchmark suite. [31]
- **Rodinia Benchmark Suite**: There are many suites for parallel computing on general-purpose CPU architectures, but accelerators fall into a gap that is not covered by previous benchmark development. Rodinia is released to address this concern. [32] [33]
- **Github**: used for version control and for storing/sharing code.

B Design Alternatives that were not implemented

B.1 Hardware

- Array-based Implementation: recall that our controller deals with multiple lanes by multiplexing through the bits of the ports. For example, the WriteEn port is NB_LANES bits wide. If the lane to operate on is the first, we look at the 0th bit of WriteEn. While fully functional, this method is cumbersome for the programmer to understand, so it was of interest to attempt to utilize arrays to facilitate the readability of the code. The difference between both approaches is depicted in Figure 20.

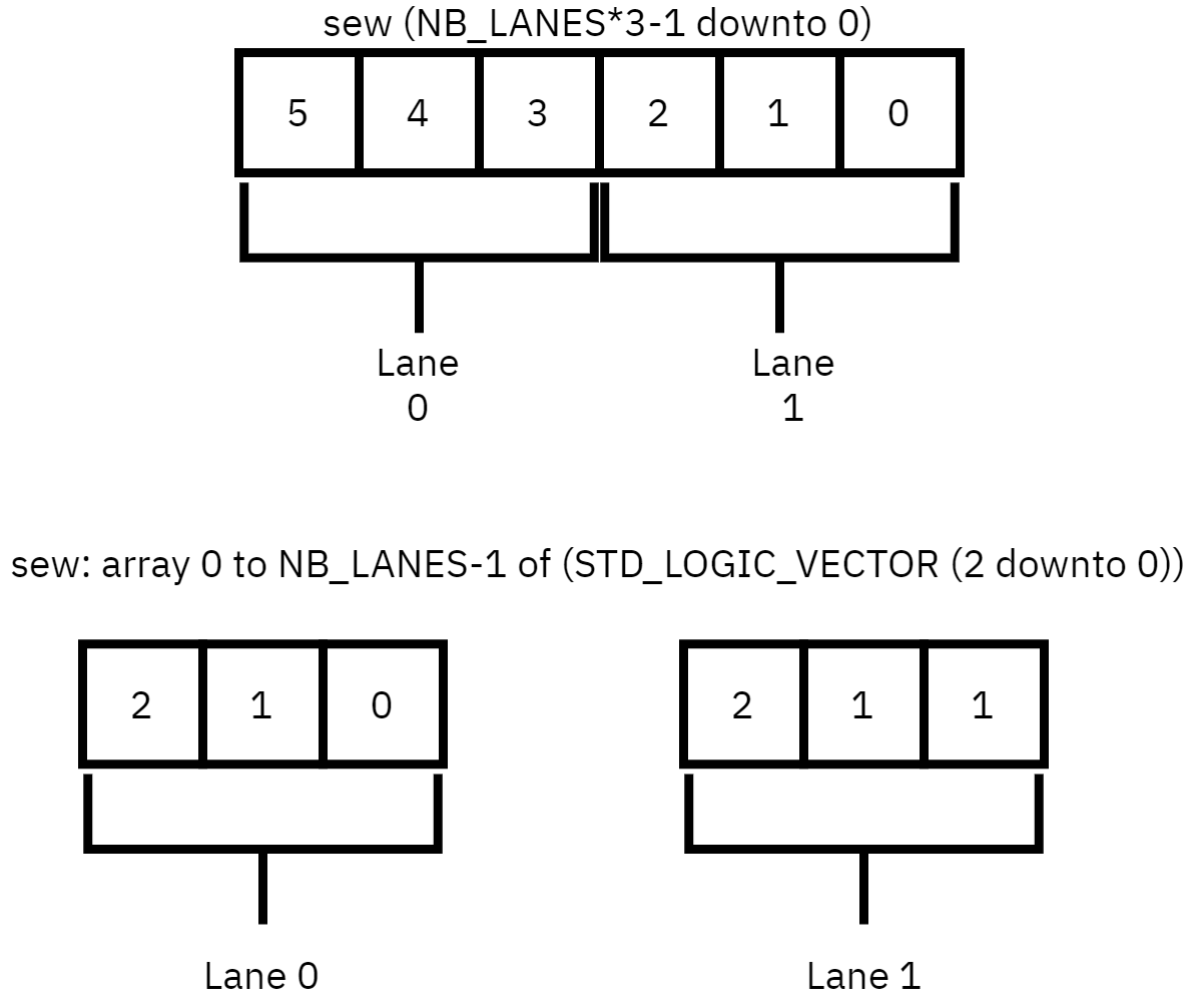

Figure 20: Arrow-Based Implementation

This renovation should have no impact on functionality. While simulations were successful and readability was indeed improved in writing and simulating test-benches, the results achieved during implementation were inadequate, where worst negative slack was **-86.710** ns at a clock frequency of 40 MHz. While this doesn't completely negate the approach, perhaps certain optimization methods are necessary to make the array-based approach more implementable. Therefore, we had to revert back to the flat-vector implementation.

- Another noteworthy attempt at enhancing readability and ease of handling the code was to use a package file that acts as a library for all the generics used throughout the project. This way, entity declarations and port mappings are simpler, and the centralization of the user-defined generics prevents any incoherence of values between the files, since all generics refer to the same one defined in the package file.

B.2 Software

B.2.1 GNU/GCC Cross-compiler

In earlier stages of the project, we have been planning to use the GNU/GCC toolchain to cross-compile C/C++ code to RISC-V binaries with vector instructions support. We have moved on to use the LLVM/Clang toolchain instead, due to its modularity and how it would allow us to make changes further ahead to accommodate additional custom RISC-V vector instructions.

B.2.2 gem5 Accelerator Modeling Frameworks

We have briefly considered working with two accelerator modeling frameworks designed for use with gem5 : gem5 SALAM, which is an LLVM-based framework used for modeling and simulating custom hardware accelerators, and gem5 ALADDIN, which is a tool used to study the complex behaviors and interactions between CPUs and hardware accelerators. The issue with both of these projects is that they offered little to no support for the RISC-V ISA, not allowing us to use them to their full-extent in our works.

C Project Plan and Task Distribution

The tasks were distributed across the team: Imad and Mohammad took care of the hardware side, while Hadi worked on the software side.

C.1 Hardware

During the Fall semester, Imad and Mohammad reviewed all the work they had performed in Spring 2019-2020 as EECE 499. They then debugged their work and implemented the components up to the combination of VRF, ALU, Controller.

During the Spring semester, the project went through a major overhaul due to the implemenation of the SIMD ALU; nearly every component was revisited and improved. This lead to an increase of the FPGA clock frequency from 40 MHz to 100+ MHz. Imad also worked on the memory implementation

C.2 Software

During the Fall semester, Hadi got used to the different software tools (e.g. gem5, LLVM/Clang, SPIKE) and looked into the MLPerf benchmark suite.

During Spring, he mainly worked on the implementation of the gem5 simulation model and benchmarks.

D Project Description and Agreement Form

American University of Beirut

Department of Electrical and Computer Engineering

Project Description and Agreement Form

Final Year Project

Faculty Supervisor	**Dr. Mazen Saghir**
Co-Supervisor [optional]	N/A
Sponsor [optional] ***Is there industry support or funding the project?***	No
Project Title ***Descriptive title not necessarily the final title that will be adopted by the team***	A Configurable RISC-V Vector Accelerator for Machine Learning Applications
Project Description and Design Aspects ***What is the main motivation for the project?*** ***Specify the desired needs that the final product is expected to meet.***	With increased reliance on artificial intelligence and machine learning there is a growing need for efficient computing platforms to support emerging applications at suitable levels of performance and energy efficiency. Vector processing is a computing style that exploits high levels of data parallelism, making it well suited for efficiently accelerating machine learning algorithms used in video, image, and audio processing applications. The aim of this project is to design, implement, and evaluate a configurable RISC-V vector accelerator using a Xilinx FPGA device. The accelerator should communicate with a host processor using a suitable AXI4 interface. It should also be capable of transferring data between a vector register file and a DDR memory chip. Finally, the vector accelerator should be programmable in C using appropriate intrinsics. The latency and throughput of vectorized machine learning kernels will be measured using appropriate Xilinx Vivado tools and compared against non-vectorized software implementations.
Expected Deliverables ***Required deliverable(s) from the team at the conclusion of the design project***	1- A design-time configurable VHDL model of the vector accelerator. 2- Support for a subset of instructions specified in ISA. Details can be found in appendix 3- A customized LLVM cross-compiler that optimizes code for the accelerator

	4- A functional hardware prototype implemented on a Nexys Video FPGA and capable of executing vectorized machine learning kernels with minor to no modifications. 5- A gem5 simulation model of the accelerator and comparison to existing solutions. Also benchmark design using MLperf.
Technical Constraints ***A preliminary list of multiple realistic technical constraints, e.g. power, accuracy, real-time operation , … The technical constraints included should be detailed and specific to the design project not generic.***	1- The accelerator should be compliant with the RISC-V Vector ISA. 2- The accelerator should be programmable in C using appropriate intrinsics. 3- The hardware prototype should operate at a clock rate of 100 MHz or better.
Non-Technical Constraints ***A preliminary list of multiple realistic non-technical constraints, e.g. cost, environmental friendliness, social acceptance , political, ethical, health and safety, etc… The non-technical constraints included should be detailed and specific to the project not generic.***	1- Availability of target FPGA and software programs in ECE labs. 2- Working from home due to COVID-19.
Contemporary Issues ***Cite one or more recent articles pertaining to the project or project area: news articles, blog discussions, academic articles, conference topics, etc.***	Machine learning hardware accelerators; vector architectures; RISC-V.
Resources and Engineering Tools ***Identify resources and engineering tools needed and whether they are available or need to be acquired (if known), e.g. software licenses, instruments, facilities, components, …***	1-Xilinx Vivado and SDK 2-RISC-V toolchain 3-gem5 simulator 4-Nexys Video FPGA
Possible Applicable Standards ***List potential standards directly or indirectly used or involved in the project***	1- RISC-V Vector Extension ISA 2- ARM AXI4 3- 1076-2019 - IEEE Standard for VHDL
List of Disciplines	-Hardware, Computer Architecture, and Digital Systems -Software Engineering

Identify at least THREE engineering disciplines (within or outside ECE)	-Machine Learning
Number of Students ***Please consider the number of disciplines checked above***	3 students
Required Courses [Optional] ***List the courses that are essential for the successful execution of the project (especially advanced courses)***	1- EECE 420 2- EECE 421 3- EECE 423 4- EECE 430
Date Submitted	September 4 2020

E Meeting Minutes

Final Year Project Meeting Minutes

Meeting #1		
Date: 9.30.2020	Time: from 12:00 to 13:00	Location: Zoom
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Description of FYP	
Discussion	**Scope of FYP**
Dr. Saghir went over the scope and the end-goal of the FYP.	
Conclusions	

- Design a RISC-V vector processor as per the RISC-V Vector Extension ISA.
- Integrate the processor into a RISC-V core (sweRV core), but first into MicroBlaze.
- Build hardware on Nexys Video FPGA
- Create a model on the GEM5 simulator to simulate the complete system with cache and memory in software. Can help debug hardware issues and compare to other existing solutions
- Ideally, would be able to run ML algorithms with none to few changes.
- Benchmark using MLPerf
- Low priority: Configure accelerators and generate VHDL code through a Python script to facilitate setup.

Action Items	**Person Responsible**	**Deadline**
Finalize and test hardware of RISC-V vector processor	Imad Al Assir, Mohammad El Iskandarani	End of Fall
Support evaluation and validation of the hardware by developing APIs for MicroBlaze in software	Hadi El Sandid, Imad Al Assir	End of Fall
Migrate from MicroBlaze to RISC-V sweRV core	Hadi El Sandid, Imad Al Assir	End of Spring
Model the vector processor in GEM5	Hadi El Sandid	End of Fall
Study how to run ML algorithms on vector processor	Mohammad El Iskandarani	End of Spring
Benchmark using MLPerf	Mohammad El Iskandarani	End of Spring
Low priority: Write a Python script to configure accelerators and generate VHDL code automatically.	Hadi El Sandid	End of Spring

Agenda Item: Choice of Processor	
Discussion	
Discussed which processor to use: MicroBlaze softcore processor vs RISC-V sweRV core	
Conclusions	

Final Year Project Meeting Minutes

- MicroBlaze:
  - Pros: accessible, have prior experience with it.
  - Cons: vector processor does not integrate very well with it. It will be a co-processor, not autonomous (e.g. able to access memory on its own) as it should be.
- RISC-V sweRV:
  - Pros: compatibility with RISC-V Vector Processor, better integration, can use native RISC-V compiler to vectorize code.
  - Cons: no prior experience, harder to get started

Action Items	**Person Responsible**	**Deadline**
Implement platform with MicroBlaze softcore processor	Hadi El Sandid, Imad Al Assir	End of Fall
Migrate to RISC-V sweRV	Hadi El Sandid, Imad Al Assir	End of Spring

Agenda Item: Tasks for the near future

Discussion	

Discussed where we should start, what to do in the near future

Conclusions	

- Hadi should familiarize himself with MicroBlaze and development for FPGA using Xilinx Vivado, by building a simple application. Imad can help with that since he has significant experience.
- Hadi should start looking into GEM5 and get it running on his machine.
- Since Imad and Mohammad had previously started the project in EECE 499, need to revisit previously faced problems and document them. Dr. Saghir will then help solve them.
- Imad, Mohammad and Dr. Saghir should meet again to review design choices made previously (e.g. number of lanes, scheduling mechanism, ...)

Action Items	**Person Responsible**	**Deadline**
Build simple MicroBlaze application on FPGA	Hadi El Sandid	By next meeting
Setup GEM5 on personal machine and get up to speed	Hadi El Sandid	In 2 weeks
Review and report problems faced previously	Imad Al Assir, Mohammad El Iskandarani	By next meeting
Review previous design choices	Imad Al Assir, Mohammad El Iskandarani	By next meeting

Final Year Project Meeting Minutes

Meeting #3		
Date: 9.30.2020	Time: from 12:00 to 13:00	Location: Zoom
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Description of FYP	
Discussion	**Scope of FYP**
Dr. Saghir went over the scope and the end-goal of the FYP.	
Conclusions	

- Design a RISC-V vector processor as per the RISC-V Vector Extension ISA.
- Integrate the processor into a RISC-V core (sweRV core), but first into MicroBlaze.
- Build hardware on Nexys Video FPGA
- Create a model on the GEM5 simulator to simulate the complete system with cache and memory in software. Can help debug hardware issues and compare to other existing solutions
- Ideally, would be able to run ML algorithms with none to few changes.
- Benchmark using MLPerf
- Low priority: Configure accelerators and generate VHDL code through a Python script to facilitate setup.

Action Items	**Person Responsible**	**Deadline**
Finalize and test hardware of RISC-V vector processor	Imad Al Assir, Mohammad El Iskandarani	End of Fall
Support evaluation and validation of the hardware by developing APIs for MicroBlaze in software	Hadi El Sandid, Imad Al Assir	End of Fall
Migrate from MicroBlaze to RISC-V sweRV core	Hadi El Sandid, Imad Al Assir	End of Spring
Model the vector processor in GEM5	Hadi El Sandid	End of Fall
Study how to run ML algorithms on vector processor	Mohammad El Iskandarani	End of Spring
Benchmark using MLPerf	Mohammad El Iskandarani	End of Spring
Low priority: Write a Python script to configure accelerators and generate VHDL code automatically.	Hadi El Sandid	End of Spring

Agenda Item: Choice of Processor	
Discussion	
Discussed which processor to use: MicroBlaze softcore processor vs RISC-V sweRV core	
Conclusions	

Final Year Project Meeting Minutes

- MicroBlaze:
  - Pros: accessible, have prior experience with it.
  - Cons: vector processor does not integrate very well with it. It will be a co-processor, not autonomous (e.g. able to access memory on its own) as it should be.
- RISC-V sweRV:
  - Pros: compatibility with RISC-V Vector Processor, better integration, can use native RISC-V compiler to vectorize code.
  - Cons: no prior experience, harder to get started

Action Items	Person Responsible	Deadline
Implement platform with MicroBlaze softcore processor	Hadi El Sandid, Imad Al Assir	End of Fall
Migrate to RISC-V sweRV	Hadi El Sandid, Imad Al Assir	End of Spring

Agenda Item: Tasks for the near future

Discussion	

Discussed where we should start, what to do in the near future

Conclusions	

- Hadi should familiarize himself with MicroBlaze and development for FPGA using Xilinx Vivado, by building a simple application. Imad can help with that since he has significant experience.
- Hadi should start looking into GEM5 and get it running on his machine.
- Since Imad and Mohammad had previously started the project in EECE 499, need to revisit previously faced problems and document them. Dr. Saghir will then help solve them.
- Imad, Mohammad and Dr. Saghir should meet again to review design choices made previously (e.g. number of lanes, scheduling mechanism, ...)

Action Items	Person Responsible	Deadline
Build simple MicroBlaze application on FPGA	Hadi El Sandid	By next meeting
Setup GEM5 on personal machine and get up to speed	Hadi El Sandid	In 2 weeks
Review and report problems faced previously	Imad Al Assir, Mohammad El Iskandarani	By next meeting
Review previous design choices	Imad Al Assir, Mohammad El Iskandarani	By next meeting

Final Year Project Meeting Minutes

Meeting #4		
Date: 23/10/2020	Time: from 12:00 to 13:00	Location: Zoom
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Hardware updates

Discussion

Mohammad and Imad discussed their updates on the hardware design

Conclusions

- Mohammad and Imad explained their approach to the design scalability: making one large port whose size can change based on a generic. The port is then split into different signals and distributed to each component. This is necessary because, since it will be connected to the controller, it should match it. Dr. Saghir approved the design.
- Mohammad and Imad also mentioned that they will be working on the controller updates for next week. Dr. Saghir also approved.
- Dr. Saghir mentioned that he started working on bug fixes in the address generator masking logic but did not solve them completely yet. He said that he should finalize it by next week.

Action Items	**Person Responsible**	**Deadline**
Update controller design to enable scalability and control of multiple instructions in pipeline	Imad Al Assir, Mohammad El Iskandarani	Next week
Finalize address generator masking logic	Dr. Saghir	Next week

Agenda Item: Design and Simulation in GEM5

Discussion

Discussed the next steps we should take to simulate our RISC-V Vector Processor Unit in gem

Conclusions

- Hadi has modeled a basic system containing a host processor and an ALU co-processor, which communicate together through memory mapped I/O.
- There were some issues setting up the RISC-V cross-compiler to obtain proper RISC-V binaries. Hadi has agreed with Dr. Saghir to focus on getting a working installation of the RISC-V GNU Toolchain by next week.

Action Items	**Person Responsible**	**Deadline**
Setup the RISC-V GNU Toolchain, including the cross-compiler and simulator	Hadi El Sandid	Next week
Use the cross-compiler to obtain RISC-V binaries, and run them on a RISC-V compatible gem5 system	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Meeting #5		
Date: 29/10/2020	Time: from 12:30 to 13:30	Location: Zoom
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Hardware updates

Discussion

Mohammad and Imad discussed their updates on the hardware design

Conclusions

- Mohammad and Imad presented a problem related to the address generator and register file: delay of 1 cycle between provided offset and actual value read at that offset. Dr. Saghir said that problem might be: inferred latch (no else statement or default value set), else might need to pipeline address generation and regfile (shorter critical path than including address generation in regfile).
- Dr. Saghir proposed adding a register that counts the number of cycles taken by an instruction (to measure performance because we cannot insert probe into our hardware design). Useful for later comparing to GEM5 simulation results and validating hardware.
- Dr. Saghir also said that to mark it as a milestone and avoid future surprises, when we connect the controller to other components of the design (ALU, register file, etc...), we should synthesize and test it on FPGA.

Action Items	Person Responsible	Deadline
Debug regfile+address gen	Imad Al Assir, Mohammad El Iskandarani	Next week
Update controller design to enable scalability and control of multiple instructions in pipeline	Imad Al Assir, Mohammad El Iskandarani	Next week
Finalize address generator masking logic	Dr. Saghir	Next week
Prepare AXI interface for vector processor and test it on FPGA with MicroBlaze	Imad Al Assir	In 2 weeks

Agenda Item: Simulation and compilers

Discussion

Hadi discussed progress done in simulators (GEM5 and SPIKE) as well as running the GCC compiler on his machine.

Conclusions

Final Year Project Meeting Minutes

- Hadi successfully ran the Spike simulator, and RISC-V cross-compiler. Dr. Saghir suggested that we use the Spike simulator later on as a reference to validate our hardware. Hadi needs to look at possible statistics given by the Spike simulator.
- Hadi looked at how to add intrinsics (pragmas) to cross-compiler. Tried to develop some basic ones. Still not very successful. Dr. Saghir clarified that intrinsics are used to give hints to compiler e.g. use intrinsics to tell compiler to unroll loop 10 times => Give compiler info that is not available at compile-time. Info that does not necessarily translate into code.
- Dr. Saghir asked if we can do assembly inlining, Hadi responded we can. Saghir suggested creating an Assembly function. Hadi suggested: create header file and functions implemented using inline assembly then call these functions from C code.
- Dr. Saghir said that we need to do a comparison between LLVM and GNU compilers to view which is better. GNU might be adequate but need to do our homework. Specifically, check support for vector instructions.
- Hadi showed output of running a simple multiply co-processor in GEM5. Dr. Saghir noticed a high tick count for simple multiplication code. Need to look into that because it is important to have correct performance measurements, since this is the goal of our vector processor.

Action Items	**Person Responsible**	**Deadline**
Finish original co-processor code and verify tick count	Hadi El Sandid	Next week
Compare LLVM and GNU	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Meeting #6		
Date: 04/11/2020	Time: from 12:30 to 13:30	Location: Zoom
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Hardware updates

Discussion

Mohammad and Imad discussed their updates on the hardware design

Conclusions

- Mohammad and Imad first mentioned their solution to the address generator problem: flipping reading/writing edges in register file. Also they showed their simulation results to Dr. Saghir and he was satisfied.
- Dr. Saghir asked a few questions related to generics in the address generator design. Mohammad and Imad answered them and Dr. Saghir said that he should finish the address generator very soon.
- Mohammad and Imad mentioned that there were few undocumented changes in the v0.9 of the spec, so they updated the design to support them. Dr. Saghir requested that they go over the spec sheet again to make sure there are no other changes.
- Dr. Saghir reminded the team that they should synthesize the design and test it on FPGA to mark a milestone.

Action Items	**Person Responsible**	**Deadline**
Review spec v0.9 and make sure there are no unimplemented changes	Mohammad El Iskandarani, Imad Al Assir	Next week
Finalize address generator masking logic	Dr. Saghir	Next week
Prepare AXI interface for vector processor and test it on FPGA with MicroBlaze	Imad Al Assir	Next week

Agenda Item: Exploring different toolchains to generate RISC-V binaries

Discussion

Dr. Saghir had proposed we investigate other options that the GNU/GCC toolchain to generate RISC-V binaries. He recommended we investigate the LLVM toolchain.

Conclusions

Final Year Project Meeting Minutes

- Hadi presented the advantages of LLVM over GNU: LLVM uses modern ideas and is easier to use than GNU. However, to run Linux, he said that GNU is still required. Also, to generate RISC-V binaries, GNU (RISC-V toolchain) is required.
- Dr. Saghir mentioned that Hadi needs to look at the Mlperf benchmark suite, and research how to compile it using LLVM. He also requested that Hadi looks at APIs for LLVM and gets a running example for next week.
- Concerning compilation, Dr. Saghir mentioned that we do not necessarily need automatic code vectorization at this point; we just want the code to run. Therefore, we should use Assembly inlining to test our hardware.
- Dr. Saghir recommended we start looking at the support for RISC-V vector instructions in the current release of the LLVM compiler

Action Items	**Person Responsible**	**Deadline**
Become familiar with the LLVM/Clang compiler toolchain	Hadi El Sandid	Next week
Look into LLVM/Clang support for RISC-V vector instructions	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Meeting #1		
Date: 22/02/2021	Time: from 10:00 to 11:00	Location: Webex
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Hardware updates

Discussion

Mohammad and Imad discussed their updates on the hardware design improvements and memory interface

Conclusions

- Imad said that narrow transfers are too complex to implement, due to many variables and a difficulty to debug. They are a desired feature for later, but they will be skipped for now, for timing purposes. Dr. Saghir confirmed that this is not a problem because Arrow itself will support any memory operation, but the limitation is due to the platform we have and the interface available. Need to focus on getting contiguous accesses to work, and support sparse accesses, even if they are slow.
- Mohammad confirmed that the updated design of Arrow (saving cycles and more clean) works fine. Few bugs are present in the synthesis, but they should be easy to fix. Need to redesign done flag and implement some kind of dispatching/scheduling mechanism to interface properly with host device (MB or SweRV).

Action Items	**Person Responsible**	**Deadline**
Implement memory interface to support contiguous accesses	Imad Al Assir	Next week
Debug synthesis errors	Mohammad El Iskandarani, Imad Al Assir	Next week
Fix done flag and scheduling/dispatching mechanism	Mohammad El Iskandarani	In 2 weeks

Agenda Item: Design and Simulation in gem5

Discussion

Discussed the next steps we should take to simulate our RISC-V Vector Processor Unit in gem

Conclusions

- Hadi informed the group that gem5-SALAM will not work because it is mainly targeted for ARM devices. The build for RISC-V is broken, and would take significant time and effort to fix. => Back to vanilla gem5.
- Dr. Saghir asked about custom instructions and Hadi confirmed that these work fine.
- Hadi needs to implement a tightly-coupled vector unit and not a memory-mapped one because the latter uses a separate bus, which is significantly slower.
- Gem5 model: you only implement functionality (e.g.: take a C/C++ function and specify how many ticks/cycles it takes). Would still need to implement the Vector Register File though.

Action Items	**Person Responsible**	**Deadline**
Implement a functional vector addition, including a VRF	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Meeting #2		
Date: 01/03/2021	Time: from 10:00 to 11:00	Location: Webex
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Hardware updates

Discussion

Mohammad and Imad discussed their updates on the hardware design improvements and USouthampton project

Conclusions

- Mohammad mentioned that he fixed the synthesis errors but that timing constraints were not met. However, he confirmed that the design should be error-free, unlike before. Dr. Saghir recommended that we worry about timing later, and focus on functionality.
- Imad discussed USouthampton's vector accelerator design with Dr. Saghir: they only worked in simulation; single-cycle, no timing considerations. They do not have chaining and they use the minimum vector register size (32 registers of 32 bits). They also only implemented simple memory operations (no strided/indexed).
- Dr. Saghir recommended that we look at their SIMD ALU design, and evaluate the feasibility of implementing something similar.

Action Items	Person Responsible	Deadline
Implement memory interface to support contiguous accesses	Imad Al Assir	Next week
Study the feasibility of implementing smaller SIMD ALUs instead of 1 big ALU	Mohammad El Iskandarani, Imad Al Assir	Next week
Fix done flag and scheduling/dispatching mechanism	Mohammad El Iskandarani	Next week

Agenda Item: Design and Simulation in gem5

Discussion

Designing a tightly coupled vector accelerator in gem5 would require us to extend the simulator to support some necessary features.

Conclusions

- Hadi has explored different methods to (1) implement functional units with support for vector instructions, (2) tightly couple an accelerator to a CPU model, (3) design a Vector Register File that is compliant with RISC-V "V" spec
- Implementing the above features from scratch would be very-time consuming. It would be interesting to investigate some similar RISC-V vector accelerator implementation and their approach.
- Imad mentioned that Hadi should look at USouthampton's implemented benchmarks: TinyMLPerf and TFLMicro. Ported code is available on Github.

Action Items	Person Responsible	Deadline
Look into USouthampton's design for their RISC-V Vector accelerator, as well as their proposed benchmarks (Presentation + Repository)	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Evaluate the RISC-V accelerator design proposed by the Barcelona Supercomputing Center, to evaluate how parametrizable their model is, and how it performs against a system with no accelerator.	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Meeting #3		
Date: 10/03/2021	Time: from 10:00 to 11:00	Location: Webex
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Hardware updates

Discussion

Mohammad, Imad and Dr. Saghir discussed the feasibility of implementing SIMD ALUs

Conclusions

- Motivation for SIMD ALUs: current underutilization of big ALUs. Idea is to use smaller execution units depending on SEW. Downside: adding logic will lead to using more routing resources on the FPGA => possibly slower design.
- Question by Mohammad: how many chunks to ship? Dr. Saghir's answer: limit ELEN to 64 bits because accelerator targeted for embedded/edge devices.
- Imad: make the ALU design structurally and connect carries in a specific way. Dr. Saghir: no need, synthesis tools can infer this automatically.
- Take 1 week max to work on this design. Cannot spend more time on it.
- Need to modify: offset gen, ALU, VRF. For masked operations, need to perform gather/scatter first, or else the hardware would be doing useless work.

Action Items	**Person Responsible**	**Deadline**
Implement memory interface to support contiguous accesses	Imad Al Assir	Next week
Implement simple standalone SIMD ALU, then modify Arrow datapath.	Mohammad El Iskandarani, Imad Al Assir	Next week

Agenda Item: Report on U. of Southampton's "AI-Vector-Accelerator"Project

Discussion

A team of students at U. of Southampton have worked on a RVV accelerator model, along with a set of tools to assess its performance.

Conclusions

- Hadi wrote a report on the team's work covering (1) their set of benchmarks utilizing RVV assembly, (2) and their implementation of RVV kernels in the TensorFlow and TinyMLPerf frameworks.
- After going over the report in our meeting, we decided it would be interesting to try and bump support for the RVV assembly benchmarks from RVV v0.8 to RVV v0.9, and to try and run the modified TinyMLPerf benchmarks using RVV assembly on the SPIKE simulator, or on the gem5 simulator.

Action Items	**Person Responsible**	**Deadline**
Look into the required changes that need to be made to bump the RVV assembly routines from v0.8 to v0.9	Hadi El Sandid	Next week
Run the TinyMLPerf Benchmark suite with RVV support on the SPIKE simulator, or/and the gem5 simulator.	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Agenda Item: Report on BSC's "RISC-V Vector Engine" Project	
Discussion	
A group of professors & PhD students at the Barcelona Supercomputing center have released a paper and a set of gem5-related tools describing a configurable RISC-V vector engine.	
Conclusions	
• Hadi wrote a report on the team's work covering (1) the available parameters and limitations of the vector engine model in gem5 (2) and running benchmarks on a system containing that model to assess its performance. • After going over the report in our meeting, we decided to do more testing on this model and look more into some details of the report which are still unclear.	

Action Items	**Person Responsible**	**Deadline**
Run more benchmarks, and do a sanity check on some of the parameters of the gem5 vector engine model (Reorder Buffer on/off..)	Hadi El Sandid	Next week
Clarify some aspects of the report by going through the paper/ code repository (Lane topologies available, what vector instructions are available in the current model..)	Hadi El Sandid	Next week

Final Year Project Meeting Minutes

Meeting #3		
Date: 17/03/2021	Time: from 10:00 to 11:00	Location: Webex
Meeting called by	Group	
Attendees	Imad Al Assir, Mohammad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Imad Al Assir	

Agenda Item: Hardware updates	
Discussion	

Mohammad, Imad and Dr. Saghir discussed the implementation of the SIMD ALU and the paper outline.

Conclusions	

- Mohammad notified Dr. Saghir that he completed the SIMD ALUs design, as well as the necessary modifications to the datapath. Everything is working as expected; we can work on up to 64 bits elements at at time, in each lane, both masked and unmasked.
- Dr. Saghir requested that we synthesize the new design to compare area, timing, etc. with the old design.
- Dr. Saghir then presented the outline of the paper that we will try to publish in the CARRV workshop, which is part of ISCA 2021, and distributed the tasks. He will take care of the intro and related works, Imad and Mohammad will take care of the hardware parts and Hadi the software parts. Deadline is May 14 (abstract is May 7).

Action Items	**Person Responsible**	**Deadline**
Finalize memory interface	Imad Al Assir	In 2 weeks
Synthesize new Arrow datapath	Mohammad El Iskandarani, Imad Al Assir	Next week

Final Year Project Meeting Minutes

Meeting #1		
Date: 24/03/2021	Time: from 10:00 to 11:00	Location: Webex
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohamad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Mohamad El Iskandarani	

Agenda Item: Hardware updates	
Discussion	
Mohamad and Imad discussed their updates on the hardware design improvements and memory interface.	
Conclusions	

- Mohamad began updating the memory unit according to the new SIMD approach, and discussed the issues with Imad and Dr. Saghir, such as vector stride and vector indexed operations.
- Based on Mohamad's mentioned points, Imad can now simplify the memory interface design and buffer structure.

Action Items	**Person Responsible**	**Deadline**
Finalize memory interface	Imad Al Assir	Next week
Implement reg_out writing process for 256 bits	Mohammad El Iskandarani, Imad Al Assir	Next week
Upgrade the memory unit and perform dissection processes on Arrow side	Mohammad El Iskandarani	Next week

Agenda Item: Update on the ARROW software components	
Discussion	
We talked about the required software deliverables for the remainder of our FYP & the CARRV workshop.	
Conclusions	

- Hadi should make the required changes to the RVV assembly routines by U. of Southampton to bump its RVV support from v0.8 to v0.9
- Hadi should prepare a presentation highlighting where each parameter of the BSC's gem5 model are defined in the source code of the model.

Action Items	**Person Responsible**	**Deadline**
Make the required changes that need to be made to bump the RVV assembly routines of U. of Southampton from v0.8 to v0.9	Hadi Sandid	By next meeting

Final Year Project Meeting Minutes

Prepare presentation on BSC's gem5 Vector Engine model, which details how different aspects of the model are handled in code.	Hadi Sandid	By next meeting

Final Year Project Meeting Minutes

Meeting		
Date: 2/04/2021	Time: from 10:00 to 11:00	Location: Webex
Meeting called by	Group	
Attendees	Imad Al Assir, Hadi El Sandid, Mohamad El Iskandarani, Dr. Mazen Saghir	
Minutes taker	Mohamad El Iskandarani	

Agenda Item: Hardware updates

Discussion

Mohamad and Imad discussed their updates on the hardware design improvements and memory interface.

Conclusions

- Mohamad is almost done updating the memory generator. Synthesis issues still remain.
- Imad needs to finalize memory interface.

Action Items	**Person Responsible**	**Deadline**
Finalize memory interface	Imad Al Assir	Next week
Implement reg_out writing process for 256 bits	Mohammad El Iskandarani, Imad Al Assir	Next week
Finish vector strided implementation.	Mohammad El Iskandarani	Next week

Agenda Item: Update on the ARROW software components

Discussion

We have talked about the required software deliverables for the remainder of our FYP & the CARRV workshop.

Conclusions

- Hadi has updated most of the RVV assembly routines provided by U. of Southampton. Some debug-ging is needed, in particular for the 'Dot Product' operation
- Hadi should look into possible memory scatter-gather operations which might be use to assess the performance of the Arrow vector unit.

Action Items	**Person Responsible**	**Deadline**
Make the required changes that need to be made to bump the RVV assembly routines of U. of Southampton from v0.8 to v0.9	Hadi Sandid	By next meeting

Final Year Project Meeting Minutes

Look into scatter-gather benchmarks which could be used with the Arrow vector unit.	Hadi Sandid	By next meeting

Final Year Project Meeting Minutes

Meeting		
Date: 14/04/2021	Time: from 10:00 to 11:00	Location: Webex

Meeting called by	Group
Attendees	Imad Al Assir, Hadi El Sandid, Mohamad El Iskandarani, Dr. Mazen Saghir
Minutes taker	Mohamad El Iskandarani

Agenda Item: Hardware updates

Discussion	

Mohamad and Imad discussed their updates on the hardware design improvements and memory interface.

Conclusions	

- Mohamad will work on the slides will Imad and Hadi will finalize the report.
- Mohamad and Imad will work on the Arrow to AXI memory interface connections.

Action Items	**Person Responsible**	**Deadline**
Write FYP report	Mohamad El Iskandarani, Imad Al Assir	Next week
Finish FYP slides	Mohamad El Iskandarani	Next week

Agenda Item: Update on the ARROW software components

Discussion	

We have talked about the required software deliverables for the remainder of our FYP & the CARRV workshop.

Conclusions	

- Hadi could not progress as much as he expected on last week's tasks due to deadlines and exams in other courses. He plans to resume working on them and finish them for next meeting.

Action Items	**Person Responsible**	**Deadline**
Work on memory Gather-Scatter operation in RVV v0.9 assembly (e.g. Multiplication of Two Sparse Matrices)	Hadi Sandid	By next meeting
Resume work on the BSC gem5 Vector engine model (e.g. bumping RVV support from v0.8 to v0.9)	Hadi Sandid	By next meeting

Final Year Project Meeting Minutes

Meeting		
Date: 4.22.2021	Time: from 2PM to 3PM	Location: Online Meeting
Meeting called by	Dr. Mazen Saghir	
Attendees	Dr. Mazen Saghir – Imad Assir – Mohamad Al Iskandarini – Hadi Sandid	
Minutes taker	Hadi Sandid	
Agenda Item: Discussing the Final Report and Presentation		
Discussion		
During this meeting, we have talked with Dr. Saghir about the different components we should include in our final report and presentation.		
Conclusions		
- Mohamad has shown Dr. Saghir an early version of the presentation and got his feedback. - Imad has raised concerns about the old report format, and how we are transitioning to the new format posted on Moodle for the Final report. - Hadi has asked questions about which benchmarks results to include in the final report.		

Action Items	**Person Responsible**	**Deadline**
Work on the Final Report & Presentation	Imad Assir – Mohamad Al Iskandarini – Hadi Sandid	April 25th

F Supported Instructions

Since the project is still in its early stages, only part of the RISC-V instruction set was implemented. These instructions are listed below, with a quick description. Please refer to the spec sheet for more details.
Note that .v stands for vector only instructions (e.g. memory instructions do not need multiple vector operands), .vv stands for vector-vector instructions, .vx vector-scalar instructions, and .vi vector-immediate instructions. Also note that the section number of each group of instructions refers to the section number in the RISC-V vector spec.

7.4 Vector Unit-Stride Instructions
vlw.v: vector load word signed
vlwu.v: vector load word unsigned
vle.v: vector load element
vsw.v: vector store word
vse.v: vector store element

7.5 Vector Strided Instructions
vlsw.v: vector load word strided signed
vlswu: vector load word strided unsigned
vlse.v: vector load element strided
vssw.v: vector store word strided
vsse.v: vector store element strided

7.6 Vector Indexed Instructions
vlxw.v: vector load word indexed signed
vlxwu.v: vector load word indexed unsigned
vlxe.v: vector load element indexed
vsxw.v: vector store word indexed
vsxe.v: vector store element indexed
vsuxw.v: vector store word unsigned indexed
vsuxe.v: vector store element unsigned indexed

12.1 Vector Single-Width Integer Add and Subtract
vadd.vv: element-wise vector addition
vadd.vx
vadd.vi
vsub.vv: element-wise vector subtraction (vs2[i]-vs1[i])
vsub.vx
vrsub.vx element-wise vector reverse subtraction (vs1[i]-vs2[i])
vrsub.vi

12.4 Vector Bitwise Logical Instructions
vand.vv: element-wise vector and
vand.vx
vand.vi
vor.vv: element-wise vector or
vor.vx
vor.vi
vxor.vv: element-wise vector xor
vxor.vx
vxor.vi

12.5 Vector Single-Width Bit Shift Instructions vsll.vv: shift left logical

vsll.vx
vsll.vi
vsrl.vv: shift right logical (zero-extended)
vsrl.vx
vsrl.vi
vsra.vv: shift right arithmetic (sign-extended)
vsra.vx
vsra.vi

12.7 Vector Integer Comparison Instructions

vmseq.vv: vd[i]= vs2[i]==vs1[i]
vmseq.vx:
vmseq.vi
vmsne.vv: vd[i]= vs2[i] != vs1[i]
vmsne.vx
vmsne.vi
vmsltu.vv: vd[i]= vs2[i] < vs1[i] unsigned
vmsltu.vx
vmslt.vv: vd[i]= vs2[i] < vs1[i] signed
vmslt.vx
vmsleu.vv: vd[i]= vs2[i] <= vs1[i] unsigned
vmsleu.vx
vmsleu.vi
vmsle.vv: vd[i]= vs2[i] <= vs1[i] signed
vmsle.vx
vmsle.vi
vmsgtu.vx: vd[i]= vs2[i] > vs1[i] unsigned
vmsgtu.vi
vmsgt.vx: vd[i]= vs2[i] > vs1[i] signed
vmsgt.vi

12.8 Vector Integer Min/Max Instructions

vminu.vv: element-wise minimum unsigned: vd[i]= min(vs2[i],vs1[i])
vminu.vx
vmin.vv: element-wise minimum signed: vd[i]= min(vs2[i],vs1[i])
vmin.vx
vmaxu.vv: element-wise maximum unsigned: vd[i]= max(vs2[i],vs1[i])
vmaxu.vx
vmax.vv: element-wise maximum signed: vd[i]= max(vs2[i],vs1[i])
vmax.vx

12.9 Vector Single-Width Integer Multiply Instructions

These instructions return the same number of bits as the operands (i.e. SEW)
vmul.vv: returns lower SEW bits of element-wise multiply signed
vmul.vx
vmulh.vv: returns higher SEW bits of element-wise multiply signed
vmulh.vx
vmulhu.vv: returns higher SEW bits of element-wise multiply unsigned
vmulhu.vx
vmulhsu.vv: returns higher SEW bits of element-wise multiply signed-unsigned
vmulhsu.vx

12.10 Vector Integer Divide Instructions

vdivu.vv: vd[i]= vs2[i]/vs1[i] unsigned
vdivu.vx
vdiv.vv: vd[i]= vs2[i]/vs1[i] signed
vdiv.vx
vremu.vv: vd[i]= vs2[i] rem vs1[i] unsigned
vremu.vx
vrem.vv: vd[i]= vs2[i] rem vs1[i] signed
vrem.vx

12.15 Vector Integer Merge Instructions

vmerge.vvm: vd[i] = v0[i].LSB ? vs1[i] : vs2[i]
vmerge.vxm
vmerge.vim

12.6 Vector Integer Move Instructions

vmv.v.v
vmv.v.x
vmv.v.i