Design Space Exploration and Optimization of the BOOM Processor

Brandon Black
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ, USA
brandonblack@email.arizona.edu

Kaitlyn Oura
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ, USA
koura@email.arizona.edu

Dustin Wright
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ, USA
dwright7@email.arizona.edu

***Abstract*—The Berkeley-Out-of-Order-Machine (BOOM) is a state-of-the-art, open source RISC-V processor designed to be easily modified and flexible for designers, whether the goal is use as a functional general-use processor or a domain-specific adapted design. For this reason, there is motivation to optimize the performance of the processor as much as possible out of the box, as these optimizations will more readily flow forward into future use cases.**

With this in mind, we undertake a series of parametric sweeps on three distinct pipeline components, as well as architectural adjustments to the memory unit and the branch predictor. Results are analyzed, and adjusting for the better configuration, a further parametric sweep is performed to find a further optimized result. We present a configuration for the BOOM small core that exceeds the average instructions per cycle (IPC) by 3.08% against their own RISC-V benchmark suite.

I. Motivation

BOOM was initially designed by Berkeley to fulfill the need for a baseline out-of-order processor that implements the RISC-V instruction set architecture (ISA). BOOM was built upon Rocket Chip, another Berkeley Architecture Research product that exists as an open-source System-on-Chip (SoC) generation tool that outputs synthesizable RTL [2]. Rocket Chip is implemented in a custom Hardware Construction Language (HCL) called Chisel. Chisel is flexible and allows the designer to utilize techniques such as object orientation to describe design features. Chisel, embedded in Scala, is also able to generate C++ based software simulation files and Verilog code for functional verification and testing purposes [2].

The result of this pioneering work by Berkeley is a user-friendly development platform directly supporting the RISC-V ISA ecosystem. The BOOM core is actively developed, with a followup release of BOOM v2 in 2017, and future plans to release a BOOM v3 [3]. This establishes that the BOOM architecture, while already powerful, has room for growth and design improvement, especially when considered for application-specific functionality.

II. Related work

BOOM is novel in its unique position as the first general implementation of a RISC-V out-of-order processor. There is currently no academic competitor to the BOOM design. Initial implementation branches for industry use are beginning to see adaptation, the most notable of which is ET-Maxion by Esperanto Technologies. According to the SPEC2k6Int benchmark, ET-Maxion is twice as fast as BOOM v2 [4]. However, the project exists behind closed-door development and therefore is not suitable for analysis.

BOOM takes much of its design features and inspiration from its in-order predecessor, Rocket. Rocket is a 5-stage scalar core that implements the RV32G and RV64G ISAs [2]. However, Rocket is also compatible with RISC-V. Rocket also serves as a library of modules. These modules consist of several types of caches and various other functional units. BOOM has drawn from this development library to optimize its own module implementations and to benefit from the continued development of the Rocket environment [1].

BOOM compares itself in terms of supported features with various other heterogeneous out-of-order processors (HMPs) [1]. Not all HMPs need to be out-of-order in their issue policy, however they can support the capability. One example of this is a 3D HMP. Single-ISA heterogeneous multi-core processors are comprised of multiple core types that are functionally equivalent but micro-architecturally diverse. This paradigm has gained a lot of attention in the literature as a way to optimize performance and energy [5].

III. Methodology

Our approach to BOOM architectural contribution is twofold. First, we perform parametric sweeps of three significant parts of the hardware configuration. We observe the results of these sweeps and draw conclusions on best configurations. We explore the impact of adding an n-way data cache with respect to the resulting Instructions Per Cycle (IPC). Second, we perform analysis on the BOOM branch predictor options and test a new predictor configuration against established benchmarks and analyze the results. We make available our starting state [8], results and resources [7].

A. Parametric Sweeps

Three distinct configurations of the BOOM pipeline were targeted by for parameter sweeps. These were chosen for their roles as highly utilized components of the main pipeline. All configuration adjustments targeted the small BOOM core configuration with the intent of keeping compilation times manageable (approximately fifteen minutes instead of hours).

The first component targeted was the Instruction Fetch Buffer (IFB). This component is used to decouple the instruction fetch front end of the processor from the rest of the pipeline, allowing for more rapid instruction consumption. The impact of this type of adjustment is largely dependent on the type of instruction pattern being observed, but ideally allows for greater saturation of resources. The IFB length was swept in incremental powers of two, from 2 buffers up to 128 buffers.

The next component to be targeted was the Reorder Buffer (ROB). The ROB establishes the BOOM's status as an out-of-order core. It silently manages the flow of operation execution, giving the impression of in-order execution while providing the benefits of out-of-order flow. The length of the ROB becomes a natural target for a parameter sweep. An ideal buffer length permits efficient pipeline saturation and component utilization, however a buffer sufficiently larger than ideal length wastes resources. The ROB length was swept in incremental powers of two, from 1 buffer to 128 buffers.

The last component run against parameter sweeps was the Load/Store Unit (LSU). This unit facilitates communication between main memory and processor data caches. Due to the complex nature of an out-of-order processor, it becomes beneficial to be able to prioritize the order of execution of load and store operations. For example, the unit is able to send store addresses to memory ahead of data, allowing for relaxed store operations when the data is not needed in the near future for a load operation. Similar to the prior components, a minimal buffer size is beneficial to facilitate normal operation parameters, but provides limited benefit beyond the threshold. The length of the LSU was swept in powers of two from 1 buffer to 128 buffers.

Finally, a more substantial hardware modification was performed in an effort to gauge potential performance improvements. A data cache was introduced into the small BOOM core configuration. One of the advantages of the BOOM ecosystem is its compatibility with and usage of the Rocket Chip processor building block components, making a solution like adding a data cache manageable instead of monumental. The cache configuration allowed for easy adjustment of the associativity of the cache. The unit serves as an intermediary unit for the processor and main memory, and supports modern cache features such as automatic miss return data suppression. The associativity of the data cache was swept as well, from 1-way through 32-way, increasing as powers of two.

B. Branch Predictor

BOOM uses a two level branch predictor: one is a faster ”next-line predictor” (NLP) and the other a ”backing predictor” (BPD) which is slower and more complex [1]. For BOOM, the NLP is a Branch Target Buffer (BTB) branch predictor, while the backing predictor utilizes the well-known Gshare predictor. Within their repository, Berkeley also has options for utilization of a bimodal table for prediction as well as a random branch predictor as well as a ”TAgged GEometric length” (TAGE) predictor [1]. The bimodal table is a simple kind of branch predictor that uses a table of 2-bit counters, while the TAGE predictor is much more complex. There are also hybrid branch predictors that leverage use of more than one kind of predictor. This is the foundation for the chosen approach toward a new branch predictor scheme that we propose in this project.

The proposed hybrid branch predictor takes advantage of the structure of a saturating counter and a perceptron-based branch predictor. The state machine for the predictor is much like the 2-bit saturating counter. One state is to predict not taken, one for taken, and the another state is used to utilize a perceptron-based branch predictor. A state machine for this branch predictor is shown in Figure 1.

The n-bit perceptron is only used after mispredictions. The global branch history for this part of the predictor is only updated when the perceptron is used for prediction. This branch predictor uses resources equivalent to a normal 2-bit saturating counter and a single perceptron due to the hybrid behavior. However, it is worth noting that this approach has a memory footprint much smaller than commonly utilized predictor tables such as the bimodal table or the TAGE predictors. This model of the hybrid branch predictor will be implemented and verified in python for validity. This will be done by using the GCC and MCF benchmarks to determine its performance in terms of accuracy. A parameter sweep for the depth of the perceptron in the hybrid predictor was also conducted. The results will be shown and discussed in the next

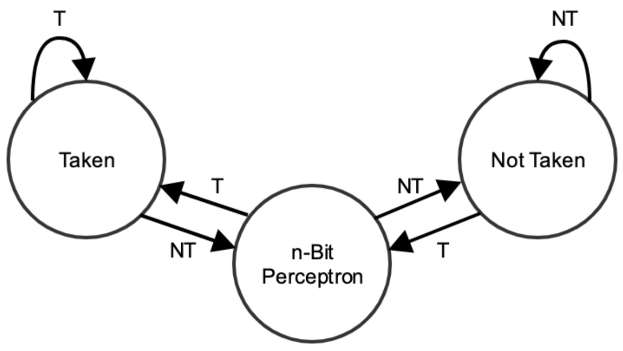


Fig. 1. Proposed predictor design as visualized state machine. The perceptron predictor only updates and predicts when the active state is the middle state.

section. Three different configurations of this hybrid predictor were tested: one where the perceptron and branch history is only updated when used for prediction, one where the branch history is always updating regardless of whether the perceptron was used for prediction and one where the branch history and the perceptron are always updating. Once completed, the best configuration will be implemented into the BOOM processor.

IV. Results

A. Parametric Sweeps

All parameter sweeps were benchmarked using the RISC-V micro-benchmark suite, consisting of 11 micro-benchmarks tooled towards testing the average instructions per cycle (IPC) for the individual benchmark. These benchmarks are: VVAdd, Towers, SPMV, RSort, QSort, Multiply, MT-VVAdd, MT-MatMul, MM, Median, and Dhrystone. All benchmarks are treated with equal weight, so results are reported as the average of the resulting IPCs reported by each benchmark.

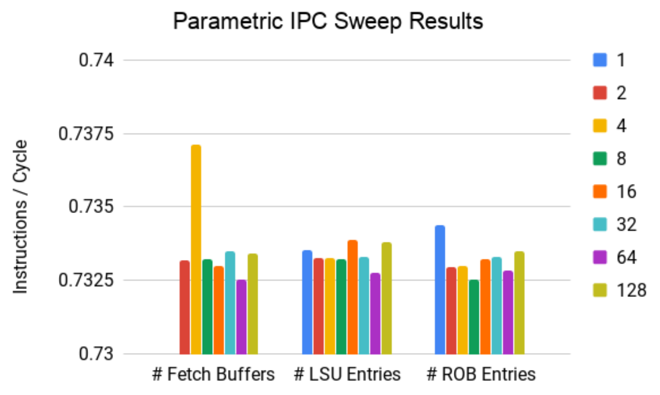


Fig. 2. Parametric sweeps for multiple configurations. Note the y-axis scale. Values are taken from averages across all benchmarks, with even weighting per benchmark.

Fig. 2 introduces the parametric sweep results for average resulting IPC for each of the parameterized metrics of the BOOM small core. Note that the base configuration for the core is 16 ROB entries, 8 IFBs, and 8 LSU entries. Therefore for the parameter sweeps, these are the configurations used for the other parameters when they are not currently being swept. Likewise, results reported for those same parameter settings during their respective sweeps are identical. It can be observed that optimal parameters occurred when 4 IFBs are instanced along with 16 ROB entries and 8 LSU entries. These parameters were repeated to confirm results, which proved to be consistent across multiple runs.

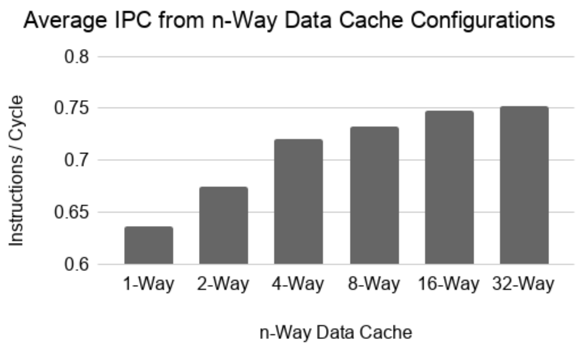


Fig. 3. Changing the associativity of the data cache has a much more significant impact on IPC than any one other parameter sweep. Improvement tapers off after 16-way and 32-way associativity.

Fig. 3 showcases the results from the addition of the data cache to the BOOM small configuration. The resulting average IPC shows a positive correlation with increasing associativity of the cache. There is an observable logarithmic relationship between the n-way split, tapering off after reaching roughly 16-way to 32-way associativity. Interestingly, the default BOOM small core IPC is approximately equivalent with the inclusion of a 4-way cache, and better than smaller associativity splits. This is an unintuitive result, but may provide insight into the operation types and structures dominating the benchmarks.

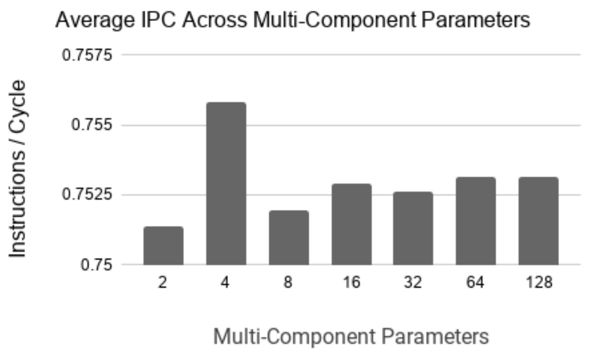


Fig. 4. Sweep of multi-component parameters with 32-way cache. Note the performance spike at set parameters of 4, consistent with previous parametric IPC sweep results.

The final parameter sweep was an optimization pass of the original parameters with a 32-way cache in place. The intention of this run was to try to find a best configuration that could synergize with the cache, and try to take advantage of the relationship between the parameterized components. More specifically, the IFB, LSU, and ROB are expected to positively impact performance as their individual lengths increase, as long as a single one of the three is not serving as a bottleneck for the others. With this in mind, the IFB, LSU, and ROB were swept at once in increasing powers of two, starting from 2 up to 128. The impact on IPC from this sweep can be observed in Fig. 4. Similarly to initial sweeps, a performance spike was observed when parameters for IFB, LSU, and ROB were set to 4 with a 32-way cache. This optimal configuration corresponded with an average IPC of 0.75579, up from the baseline configuration average IPC of 0.73321. This corresponds with a 3.08% improvement in average IPC compared with the baseline BOOM small configuration.

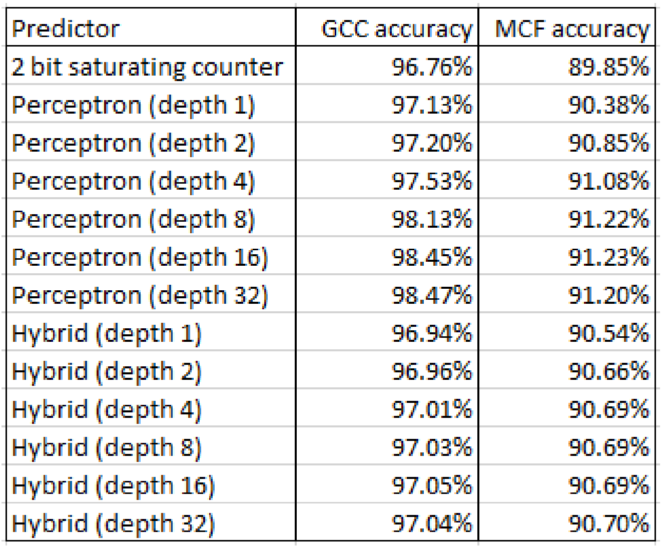
B. Branch Predictor

A Scala-written implementation of the proposed hybrid branch predictor was not successfully implemented in the BOOM processor. However, a Python-implemented prototype was tested using the GCC and MCF benchmarks. A parameter sweep of the depth of the perceptron was performed to determine the optimal configuration. Results from these parameter sweeps and benchmarks are tabulated in the next 3 figures and results are explained in the following paragraphs.

Fig. 5. shows the results of the parameter sweep for the base configuration of the proposed hybrid branch predictor. The depth indicator is length of the branch history and number of inputs into the perceptron. From the results we can see that the hybrid predictor performs better than the unmodified 2-bit saturating counter but worse than the perceptron with a depth of 1 for the GCC benchmark but performs better than both on the MCF benchmark for a depth of 1. For depths greater than one, the perceptron predictor always performs better than the hybrid since it is not reliant on taken and not taken states.

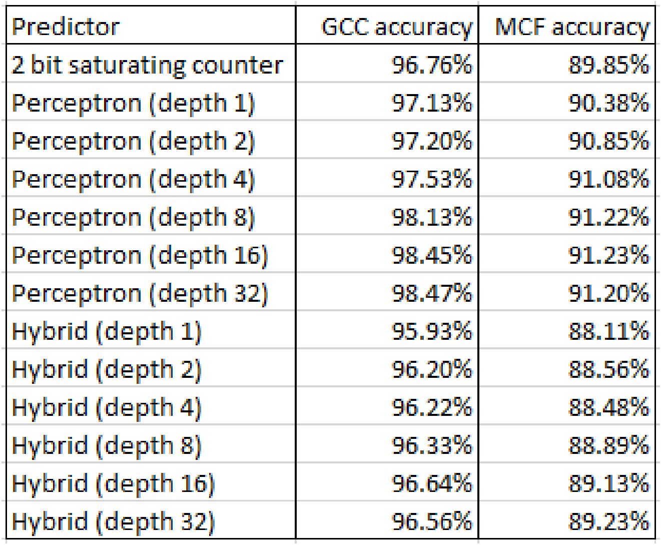
Fig. 6 shows the results of the parameter sweep for when the branch history of the predictor is always updating regardless of whether the perceptron was used or not. The intention was that with a better view of the branch history, the perceptron should be able to predict the branch more accurately than when it only received the branch history of the branches that it predicted. However, for both benchmarks the hybrid branch predictor performed worse in this situation than the previous configuration. Despite a better view of the recent branch history, a large majority of the history does not affect the weights of the perceptron, resulting in the branch history falsely influencing its output as it does not correspond with the perceptron's state.

Fig. 7 shows the results of the parameter sweep for when the branch history and the perceptron are always updating. This configuration performed worse than the 2-bit saturating counter, the perceptron and the other two configurations of the hybrid branch predictor. This configuration and the previous configuration's lower performance makes sense in the scope of the predictor. Updating the branch history and the perceptron weights were detrimental to performance when the perceptron



Predictor	GCC accuracy	MCF accuracy
2 bit saturating counter	96.76%	89.85%
Perceptron (depth 1)	97.13%	90.38%
Perceptron (depth 2)	97.20%	90.85%
Perceptron (depth 4)	97.53%	91.08%
Perceptron (depth 8)	98.13%	91.22%
Perceptron (depth 16)	98.45%	91.23%
Perceptron (depth 32)	98.47%	91.20%
Hybrid (depth 1)	96.94%	90.54%
Hybrid (depth 2)	96.96%	90.66%
Hybrid (depth 4)	97.01%	90.69%
Hybrid (depth 8)	97.03%	90.69%
Hybrid (depth 16)	97.05%	90.69%
Hybrid (depth 32)	97.04%	90.70%

Fig. 5. Base configuration of the hybrid predictor's results on the GCC and MCF benchmarks. Accuracy is the number of correct branch predictions over the total number of branches.

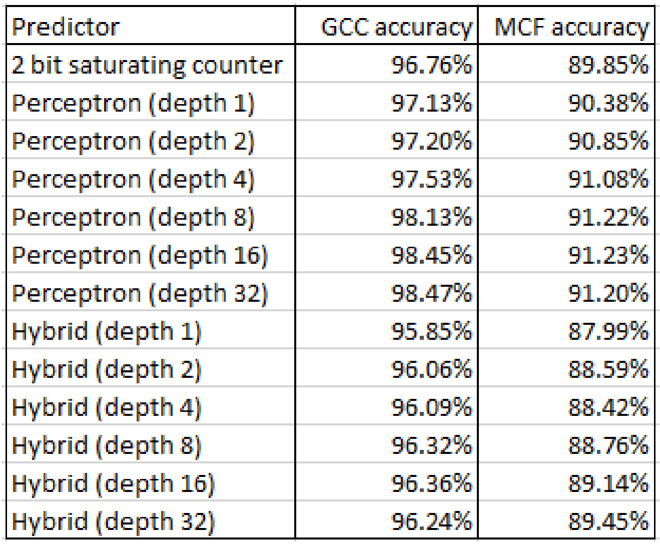


Predictor	GCC accuracy	MCF accuracy
2 bit saturating counter	96.76%	89.85%
Perceptron (depth 1)	97.13%	90.38%
Perceptron (depth 2)	97.20%	90.85%
Perceptron (depth 4)	97.53%	91.08%
Perceptron (depth 8)	98.13%	91.22%
Perceptron (depth 16)	98.45%	91.23%
Perceptron (depth 32)	98.47%	91.20%
Hybrid (depth 1)	95.93%	88.11%
Hybrid (depth 2)	96.20%	88.56%
Hybrid (depth 4)	96.22%	88.48%
Hybrid (depth 8)	96.33%	88.89%
Hybrid (depth 16)	96.64%	89.13%
Hybrid (depth 32)	96.56%	89.23%

Fig. 6. Results from when the branch history is always updating, while the perceptron weights are not.

was not the predictor being used in the state machine. This is most likely due to the fact that the weights are falsely updated based on the state machine, while the perceptron itself may have yielded the correct prediction. So constantly updating hurts that prediction.

Situationally, it seems that the depth of 1 hybrid branch predictor performs better than the 2-bit saturating counter and the depth of 1 perceptron. The MCF benchmark results show that the hybrid branch predictor is marginally better at 90.54% accurate over 90.38% accurate. This is a rather very small improvement however, as a 0.2% performance increase probably will not be noticed in reality. It is unable to outperform the pure perceptron predictor at deeper depths which was to be



Predictor	GCC accuracy	MCF accuracy
2 bit saturating counter	96.76%	89.85%
Perceptron (depth 1)	97.13%	90.38%
Perceptron (depth 2)	97.20%	90.85%
Perceptron (depth 4)	97.53%	91.08%
Perceptron (depth 8)	98.13%	91.22%
Perceptron (depth 16)	98.45%	91.23%
Perceptron (depth 32)	98.47%	91.20%
Hybrid (depth 1)	95.85%	87.99%
Hybrid (depth 2)	96.06%	88.59%
Hybrid (depth 4)	96.09%	88.42%
Hybrid (depth 8)	96.32%	88.76%
Hybrid (depth 16)	96.36%	89.14%
Hybrid (depth 32)	96.24%	89.45%

Fig. 7. Results from when the branch history and perceptron weights are always updating.

expected. The goal of the hybrid branch predictor model was to be a good, low resource replacement for the 2-bit saturation counter but the performance improvement does not seem to be worth the extra hardware associated with it.

V. Conclusions

Results from the parameter sweeps indicated that there were not significant changes in benchmarked averaged IPC as a result of tuning any one parameter. We believe that this result is not surprising, as a bottleneck can occur when any one component does not have a maximum throughput on par with the other components. This parameter sweep cemented this understanding. Adding a data cache with larger associativity proved to be more fruitful, as a net gain in average IPC occurred when implementing a data cache with 8-way associativity or higher. The improvement with size fell off after 32-way associativity, and so an associativity greater than 32-way is not advised. Finally, after adding a 32-way cache, the first parameter sweeps were repeated across all swept components simultaneously, with the intention of finding an optimal configuration that prevented any one component buffer size from bottlenecking the others. It was found that setting the IFB, LSU, and ROB buffer lengths to 4 each resulted in a maximal average IPC with respect to every other chosen configuration in this study. With our optimal configuration, the average IPC increased by 3.08%, a notable improvement over the baseline configuration of the the BOOM small core.

The proposed hybrid branch predictor model was implemented and evaluated solely in a Python environment using the GCC and MCF benchmarks to get some accuracy measures. The hybrid model did perform better than the 2-bit saturating counter on both benchmarks. But with only a .2% improvement using the depth of 1 hybrid predictor, it does not seem like a great replacement for the 2-bit saturating counter. The extra hardware and complexity that would be required is not worth the small amount of performance increase. For future projects, a different model of branch predictor should be proposed. It could also be beneficial to get this predictor working within the BOOM ecosystem, but due to time constraints and the learning curve of the BOOM ecosystem and Scala and Chisel, this was not possible in the duration of this project. Successfully getting this design implemented in the BOOM processor could help solidify the extent of the extra hardware as there exists a method for converting the BOOM processor to Verilog code, which could then be used to map it to an FPGA and evaluate the resource usage that way. Since the depth of 1 hybrid branch predictor does perform slightly better, with resource usage statistics it is possible to find an application for which this hybrid predictor could be beneficial without sacrificing physical resources.

References

[1] Asanovic, K., Patterson, D. A., & Celio, C. (2015). The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor (No. UCB-EECS-2015-167). University of California at Berkeley Berkeley United States.

[2] Asanovic, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., ... & Karandikar, S. (2016). The rocket chip generator. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17.

[3] Celio, C., Chiu, P. F., Nikolic, B., Patterson, D. A., & Asanovi, K. (2017, September). Boom v2: an open-source out-of-order risc-v core. In First Workshop on Computer Architecture Research with RISC-V (CARRV)

[4] McLellan, P. (n.d.). RISC-V Cores: SweRV and ET-Maxion. Retrieved from https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/risc-v-cores-swerv-esperanto-sifive

[5] Rotenberg, E., Dwiel, B. H., Forbes, E., Zhang, Z., Widialaksono, R., Chowdhury, R. B. R., ... & Franzon, P. D. (2013, October). Rationale for a 3D heterogeneous multi-core processor. In Computer Design (ICCD), 2013 IEEE 31st International Conference on (pp. 154-168). IEEE.

[6] (2015) Industry-standard benchmarks for embedded systems. [Online]. Available: http://www.eembc.org/coremark

[7] Black, B., Oura, K., Wright, D. (2019, April). ECE 562 Project [Online]. Github Repository, https://github.com/koura911/ECE_562_project

[8] UC Berkeley Architecture Research. (2019). RISC-V Project Template. Github Repository, https://github.com/ucb-bar/project-template/tree/856f043999c7ee4fea05964a276f3b1f58553fa6