Workload-Driven Optimization of RISC-V Core Configurations for Embedded Applications

1st Ryan Frost
Bradley Dept. of ECE
Virginia Tech
Blacksburg, USA
rfrost26@vt.edu

2rd Wangzhi Zhan
Bradley Dept. of ECE
Virginia Tech
Blacksburg, USA
wzhan24@vt.edu

***Abstract*—General purpose proecssors often take up too much space and too much power for specific embedded applications that require only a fraction of the provided resources. This project utilizes a new design pipeline to systematically test the RISC-V Berkeley Out-of-Order Machine (BOOM) on specific embedded workloads. The BOOM core's parameters are changed on a granular level, manipulating frontend widths, scalar widths, reorder buffer sizes, cache size and associativity, translation lookaside buffer, and branch predictors. The four workloads tested are Dijkstra's Algorithm, Advanced Encryption Standard (AES), Fast Fourier Transform (FFT), and Huffman Encoding.**

The key metric analyzed is Instructions per Cycle (IPC) over Gate Count, which represents performance per chip area that is relevant for embedded applications. IPC is calculated using Chipyard and Verilator, and Gate Count is calculated using Yosys synthesis. Our results indicate that larger chip sizes have diminishing return, and that a smaller, more specific configuration has the best performance per area. This shows that targeted hardware design can lower chip area while maintaining similar performance.

***Index Terms*—embedded, BOOM, Yosys, IPC, Gate Count**

I. Introduction

Embedded devices are becoming ever more prevelent in the modern age. These devices often have unique workloads, and the size and power efficiency of the chips for these devices is often just as important as the actual performance. General purpose processors typically focus less on power and space, opting for more advanced branch predictors, larger issue widths, and larger caches. However, these processors can result in extra resources that are not fully utilized by more simple embedded workloads. A processor with more silicon area typically correlates to greater power consumption, which can be detrimental to a constrained embedded environment.

The modularity of the RISC-V BOOM core presents an opportunity to analyze the space efficiency of specific chip configurations. Designing the most opitmal design is challenging because of the interconnected dependencies of the core parameters. Increasing one aspect of the chip has cascading effects on how well the other resources are used.

In this project, we evaluate the trade-offs between different chip configurations and their impact on the area efficiency. We use UC Berkeley's BOOM core and Yosys to systematically evaluate performance using IPC and are using Gate Count. This metric informs our decision on the most area efficient configuration for embedded workloads.

A. Workloads

- **Dijkstra's Algorithm**: Used in robotics and autonomous vehicles for pathfinding and decision making.
- **Advanced Encryption Standard (AES)**: Used in security and cryptography for secure data transmission.
- **Fast Fourier Transform (FFT)**: Used in digital signal processing for sensor data analysis.
- **Huffman Encoding**: Used to compress data before transmission between devices

B. Contributions

- **Systematic Design Analysis**: Seven different core parameters are manipulated and the resulting change in performance and area is analyzed.
- **Pipeline Automation**: Candidate design configurations are easily created and supported by a modular scala file and Yosys synth file.
- **Workload-Driven Optimization**: Specific workloads are used to validate performance for different embedded uses.
- **Github**: https://github.com/rfrost096/CompArchProject

II. Related Work

This project build on current research into hardware design automation and embedded device optimization. The related research focused on here is focused on the development of the BOOM core, the existing design space exploration frameworks, and workload characterization.

A. BOOM Core

The Berkeley Out-of-Order Machine (BOOM) was introduced by Celio et al. in 2015 [1]. Defined as a "synthesizable, parameterized, superscalar out-of-order RISC-V core," its development has allowed for iterative design of processor cores with a host of additional tools as part of the UC Berkeley chipyard. The hardware is manipulated by changing Chisel files that map out the parameters of the core. Several predesigned aspects of the core can be implemented without having to modify Chisel files on a granular level.

BOOM has already been iterated on into SonicBOOM (labeled as v3 in chipyard), as shown by Zhao et al. [2]. Notable

additions in SonicBOOM are TAGE branch prediction, support for the Rocket Custom Coprocessor interface (RoCC), and short-forwards branch omptimzations. This paper focuses on scaling up the BOOM core to match industry leading large superscalar cores, while we focus on scaling BOOM down to optimize for embedded workloads.

B. Deisng Space Exploration (DSE)

DSE is the process of systematically iterating through parameters in a design space to approach the optimal solution. One approach to this with the BOOM core is presented by Bai et al. [3] as BOOM-Explorer. The authors' solution to the increasing design challenge of mangaging thousands of lines of architecture parameters was to invent a "active learning algorithm" called MicroAL. They focused on "exploring Pareto optimality", where two metrics have a trade-off relationship and the goal is to optimize for both. This is similar to the trade-off we optimize for between performance and area.

Paletti et al. [4] instead focused on DSE for FPGA-based designs. They introduced Dovado, which is a CAD tool developed to support hardware developers when dealing with highly paramaterized RTL designs. Dovado takes parameter ranges and finds the "Pareto set" according to user defined metrics (example is achieveable frequency and LUTs usage). Dovando focuses on FPGA mapping using Vivado, while this project focuses on gate count for estimating the area efficiency.

Sausserea et al. [5] implemented their own "flexible pipeline" that focused on researching the impact of pipeline stages and pipeline depth in an in-order RISC-V core. They looked at impacts to frequency and resource utilization on both FPGAs and ASICs. This project focuses on the pipeline width and resources, contrary to this paper's focus on pipeline depth.

C. Workload Characterization

Workload characterization is understanding how a specific application will use/demand the resources of the processor it is run on. For embedded applications, Anuradha et al. [6] show how to characterize heterogeneoous processor workloads. They inspired the selection of two of our workloads, Dijkstra's Algorithm and Fast Fourier Transform. The authors analyzed these two workloads along with others on FPGAs and DSPs. This work validates our diverse selection of workloads that we use to show how a generic BOOM core is inefficient in specific embedded applications.

III. Proposed Method

We propose a workload-driven design space exploration pipeleine to approach the optimal BOOM core configuration for embedded applications evaluated by performance over area.

A. Baseline Architecture

The baseline architecture for this project is the SonicBOOM (BOOMv3) core as implemented in UC Berkeley's chipyard repository. BOOMv3 is a superscalar out-of-order, synthesizable RISC-V core. This core was selected because of its modularity and existing research in its design space.

The default BOOM core utilizes TAGE branch prediction, decodes RISC-V instructions into micro-ops, dispatches micro-ops into issue quues, executes operations, utilizes a load store queue for memory management, and retires instruciton in order, supporting precise exceptions. For this project, the specific BOOM configuration we have assigned as default is the MediumBoomConfig. The small, medium, and large configurations are options in seperate parameters for optimization of the pipeline.

B. Parameters

There are seven paramters to be changed in the pipeline, and they are listed below:

TABLE I
Pipeline Parameters

Name	Options	Description
Branch Predictor	TAGE, Boom2BPD	TAGE is larger but generally more accurate
Cache Line Size	32, 64	Changes how much data is loaded per cache load
Cache Associativity	2, 4	Increases cache complexity and reduces conflict misses
TLB Ways	4, 8	Impacts virtual memory performance
Frontend Width	small, medium, large	Defines fetch width, fetch buffer, fetch target queue
Scalar Width	small, medium, large	Defines decode width and issue width
Window Width	small, medium, large	Defines ROB entries, LSQ entries, max branch count

C. Automated Pipeline

We also propose a robust pipeline for testing user defined configurations. The pipeline itself has 4 stages: verifying config parameters, vefifying Scala compile, running simulation in Verilator, and calculating number of gates using Yosys.

The Scala config file is generated using a new ModularBoomConfig.scala file that takes the seven parameters as environment variables and changes the BOOM config to make a compatible BOOM core with the specific configurations selected. Some values are changed slightly from their definitions in the small, medium, and large BOOM configurations defined in the repository to make them compatible with each other. The fundamental process is shown in Figure 1.

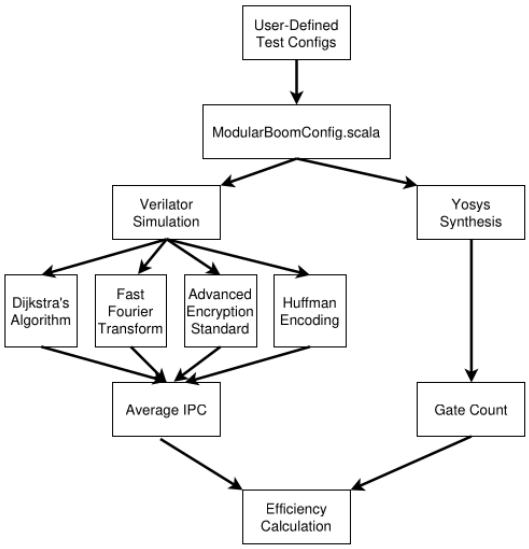

Fig. 1. Pipeline workflow to run analysis of user defined configurations.

IV. Evaluation Method

To assess BOOM core configurations against a theoretical embedded workload, we used 4 prominent embedded workloads and ran each using Verilator. The performance counters in the BOOM config enable the recording of total cycles and instructions retired for IPC calculation. Small, Medium, and Large BOOM configs are mixed together for evaluation.

A. Workload Analysis

TABLE II
Workload Characteristics

Workload	Application	Characteristics
Dijkstra's Algorithm	Robot Pathfinding	A lot of pointer chasing from the irregular memory access when access each node [7]. Branches are also data-dependent, testing the branch predictor.
AES-128 Encryption	IoT Security	A lot of integer bitwise processing focusing on ALU throughput and cache bandwidth.
FFT	Signal Processing	Floating point utilization and predictable memory accesses.
Huffman Encoding	Encoding for Transmission	Tree traversal with many branches tesing branch target buffer.

B. Evaluation Metrics

The efficiency of a configuration is determined by dividing Instructions per Cycle from the Verilator simulation by the total Gate Count from the Yosys synthesis.

$$\text{Efficiency} = \frac{\text{Instructions Per Cycle (IPC)}}{\text{Gate Count}} \quad (1)$$

The Verilator simulation provides cycle accurate metrics for the BOOM core configuration. Yosys provides a generic synthesis of the core. The Yosys synthesis typically relies on SRAM libraries to accurately determine the size of the cache. The available libraries are not extensive enough for the possible configurations and utilizing a tool such as OpenRAM to generate them proved to be difficult. Crucially, Yosys instead represents the SRAM as flip-flops, which has a major impact on the number of gates calculated for the cache.

TABLE III
Tested Configurations

CLS	Assoc.	TLB	Frontend	Scalar	Window	BPD
64	4	8	small	small	small	TAGE
64	4	8	small	medium	small	TAGE
64	4	8	small	medium	medium	TAGE
64	4	8	medium	medium	small	TAGE
64	4	8	medium	medium	medium	TAGE
64	4	8	medium	large	medium	TAGE
64	4	8	medium	large	large	TAGE
64	4	8	large	large	medium	TAGE
64	4	8	large	large	large	TAGE
64	4	4	medium	medium	medium	TAGE
64	4	8	small	small	small	Boom2
64	4	8	small	small	small	TAGE
64	2	8	small	small	small	TAGE
32	4	8	small	small	small	TAGE

[a]CLS: Cache Line Size
[a]Assoc: Cache Associativity
[a]TLB: Translation Lookaside Buffer Ways
[a]BPD: Branch Predictor

These configs test a myriad of mixes between small, medium, and large core parameters. The configurations focus on testing the scalar width with smaller and larger frontend and window width to try and get the most out of the more costly issue widths. The small configuration was used to test cache line size, cache associativity, TLB ways, and branch predictor to more accurately represent the kinds of small cores that typically run embedded workloads.

V. Evaluation Results

Evalution results are broken down by the scalar/issue width since it had the largest impact on gate count and by workload since ILP is a characteristic of the software.

A. Performance vs Area Analysis

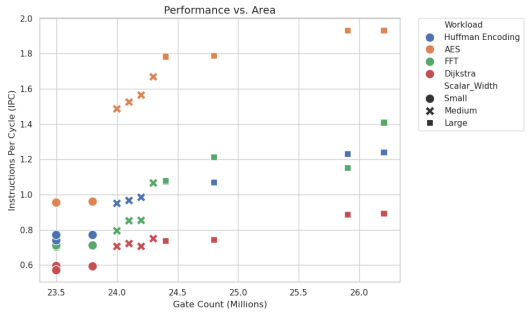

Fig. 2. Performance vs Area seperated by scalar width and workload.

There is a general positive correlation between instructions per cycle and gate count, which was expected from the start.

There is some variability between the workloads and how the added area impacts performance.

AES has the highest scalability, which could be anticipated since the added super scalar capability will most benefit the less time consuming bitwise operations.

Dijkstra is the least scalable, with the highest IPC being about 0.9. There seems to be neglible performance gains for this particular algorithm. This is likely due to the heavy data dependency of the branches and the pointer chasing as described before.

Overall, this graph appears to show how there is a significant jump between the small and medium scalar widths in terms of performance. The jump from medium to large scalar widths does not appear to have the same performance gains, although marginal gains are still present.

B. Impact of Scalar Width

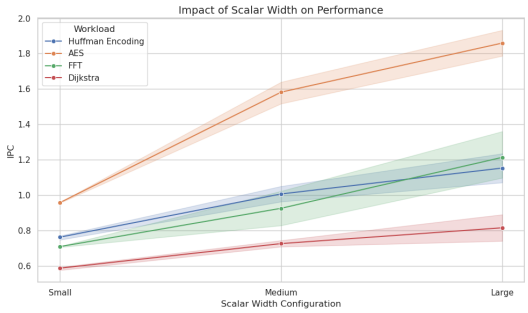

Fig. 3. Performance impact of scalar widths.

This figure shows the performance impact of the scalar widths. For Dijkstra, AES, and Huffman workloads, there is less performance gain between medium and large than there was between small and medium. Interestingly, FFT saw greater performance gains from medium to large. This is the only floating point workload, which may exploit more instruction level parallelism as more operations can be scheduled during the longer floating point operations. The FFT also has predictable memory accesses, so when the window width is increased along with the scalar width, the FFT workload can successfully execute more speculated operations than the other workloads.

C. Area Efficiency

The area effeciency appears to actually trend upward, with the large scalar width configurations having the highest area effeciency. While the increase in effeciency is less between medium and large as it is between small and medium, it still represents the best possible combination. One consideration is the impact of the bloated caches as a result of missing SRAM libraries (described in section IV.B). The larger cache sizes reduce the relative increase in area due to the scalar width. Cache optimization is also part of the analysis, and analysis on a per-parameter basis would add another level of granular analysis in future work.

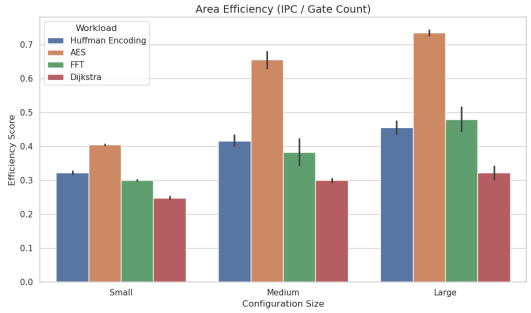

Fig. 4. Pipeline workflow to run analysis of user defined configurations.

D. Discussion

The significant differences between workloads shows how understanding the target software can better inform hardware design decisions. Increasing the scalar width and other resources does not always benefit embedded workloads. In this project, the only workload that saw increasing benefit was FFT. A medium workload would best suit this particular set of workloads, as the additional complexity of the larger cores does not justify the diminishing performance returns.

Additional testing with branch prediction shows that TAGE is typically worth the added area cost over Boom2BPD. Increasing the cache line size also changed the size of the cache, resulting in better performance, but only marginally and not worth the added area. Cache associativity did not have a significant impact on performance. TLB ways had no impact on performance.

VI. Timeline

- ~~11/05 - 11/14 Get applications in a runnable state~~ Focused on area optimization rather than algorithm design
- ~~11/05 - 11/14~~ 11/24 - 11/28 Create pipeline for testing multiple configurations for multiple applications at once
- ~~11/14 - 11/19 Systematically categorize the applications based on parameter impact~~
- 11/28 - 12/1 Categorized performance based on scalar width
- ~~12/03 - 12/08~~ 12/01 - 12/03 Compare with standardized benchmarks for each category and document overall performance between final configurations (for presentation)
- 12/03 - 12/14 Conduct additional analysis on workloads with more configurations and analyze results
- 12/14 - 12/17 Finalize report

The initial proposal focused on testing new algorithms on different core configurations, but we pivoted from the algorithm-focused project to focus on area optimization. Pipeline development took about a week as expected once we started working on it. There was a bit of a rush to get preliminary results in for the presentation. There was some disconnect between the original plan and the new direction for the presentation, but they have been resolved for this report. Overall, the timeline was pretty accurate in terms of time taken, although there were definetly periods of cruch

time to meet deadlines. Specifically, getting the first Dijkstra's algorithm all the way through the pipeline in one go was challenging and required many bug fixes. The result was a more resilient pipeline for the other workloads to run smoothly through.

VII. Contributions

Ryan Frost Contributions:

- Pipeline Development
- Proposed Method
- Evaluation

Wangzhi Zhan Contributions:

- Workload Justification
- Related Work
- Pipeline Execution

References

[1] C. Celio, D. Patterson, and K. Asanović, "The Berkeley Out-of-Order Machine (BOOM): An Industry- Competitive, Synthesizable, Parameterized RISC-V Processor," 2015. Accessed: Dec. 18, 2025. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.pdf.

[2] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine," 2020. Available: https://carrv.github.io/2020/papers/CARRV2020_paper_15_Zhao.pdf

[3] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu and M. D. F. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework," 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), Munich, Germany, 2021, pp. 1-9, doi: 10.1109/ICCAD51958.2021.9643455.

[4] D. Paletti, D. Conficconi and M. D. Santambrogio, "Dovado: An Open-Source Design Space Exploration Framework," 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, 2021, pp. 128-135, doi: 10.1109/IPDPSW52791.2021.00027.

[5] J. Saussereau, C. Jego, C. Leroux and J. -B. Begueret, "Design and Implementation of a RISC-V core with a Flexible Pipeline for Design Space Exploration," 2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Istanbul, Turkiye, 2023, pp. 1-5, doi: 10.1109/ICECS58634.2023.10382774.

[6] P. Anuradha, H. Rallapalli, G. Narasimha and S. M. Ahmed, "Efficient workload characterization technique for heterogeneous processors," 2015 IEEE International Advance Computing Conference (IACC), Banglore, India, 2015, pp. 812-817, doi: 10.1109/IADCC.2015.7154819.

[7] J. . -S. Park, M. Penner and V. K. Prasanna, "Optimizing graph algorithms for improved cache performance," in IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 9, pp. 769-782, Sept. 2004, doi: 10.1109/TPDS.2004.44.