SPARK-V: Small Power-Efficient Architecture for RISC-V

1st Justin Lebeau
Electrical and Computer Engineering
Rice University
Houston, Texas, United States
jhl6@rice.edu

***Abstract*—Wearable biomedical devices require processors that balance performance with strict energy constraints, as battery life directly impacts usability and reliability in real world health monitoring. Current solutions often rely on proprietary low power processors, which can be expensive and lack flexibility for customization. Our project proposes an open-source alternative: developing and validating a low power RISC-V processor tailored for biomedical applications. We begin by validating an existing RISC-V core on an FPGA platform to demonstrate full functionality and suitability for wearable workloads. Building on this foundation, we target a 30% reduction in system power through architectural and implementation-level optimizations. Ultimately, this project aims to produce a validated, energy efficient processor ready for tapeout, enabling more accessible and customizable solutions for biomedical wearables.**

***Index Terms*—RISC-V, Low-Power Design**

I. Introduction

The design goal of this project is to create a processor for use in wearable biomedical devices. Since these devices are battery-powered, they require an architecture that is as low-energy as possible to maximize device usability and reliability.

While low-power processors currently exist, many of these are proprietary, which often makes them restrictive, expensive, or challenging to customize. To address this, the SPARK-V project proposes an open-source alternative using the RISC-V instruction set. The project begins by utilizing WALLY, an existing open-source RISC-V core from Harvey Mudd College, as a baseline [1].

The primary goal is to achieve significant power efficiency, specifically targeting a 30% reduction in system power. This will be accomplished through systematic architectural and implementation-level power optimizations, culminating in a validated, energy-efficient processor.

II. Methods

This project will be broken across two semesters, and thus I designed a set of goals for each semester. This semester, the goals are to set up a Virtual Machine with all the needed tools for design and verification, build the base WALLY design on an FPGA and run benchmark tests, and create a set of design specifications for the final design. This will allow me to work on more physical design aspects next semester.

For this semester, the work will be done using a Virtual Machine on the Rice ORION cluster. This will include validating the existing design, and modifying the source RTL to use the proper extensions. This will also require discussion into the desired extension set. I will also use Questa and Verilator as simulation tools to test each set of extensions on the design.

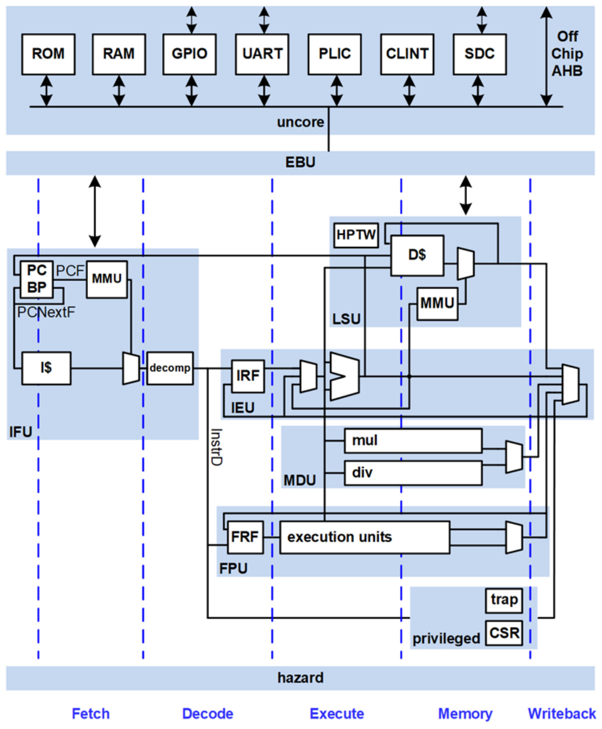


Fig. 1. Block Diagram of the WALLY system [1]

III. Results

A. Development Environment Setup

The initial work in developing WALLY requires a reliable development environment. For this purpose, I set up a Virtual Machine (VM) on the ORION system. This setup is complete, and I am able to see the source files and run verification tests. I created an instructional document detailing the steps required to set up the WALLY project for development for future users. This process is currently undergoing testing by a senior capstone team, who is also working with WALLY. The WALLY core itself is available through an open-source GitHub repository [1]. In addition to the GitHub repository, the VM contains open-source and proprietary chip design software. This includes open-source RISC-V compilers, and Verilator, an open-source SystemVerilog simulator. It also includes Questa, Design Compiler, and Vivado, commercial chip design tools.

B. *WALLY on an FPGA*

In order to implement WALLY on an FPGA, I first had to decide on a specific board to use. The WALLY GitHub provides three options for FPGA implementations, but the most affordable option is the Arty A7 100T FPGA Board from Digilent. This board contains an AMD Artix-7 FPGA, as well as the needed connections for testing, including JTAG and UART support, as well as PMOD connectors that can be used to access an SD card [2].

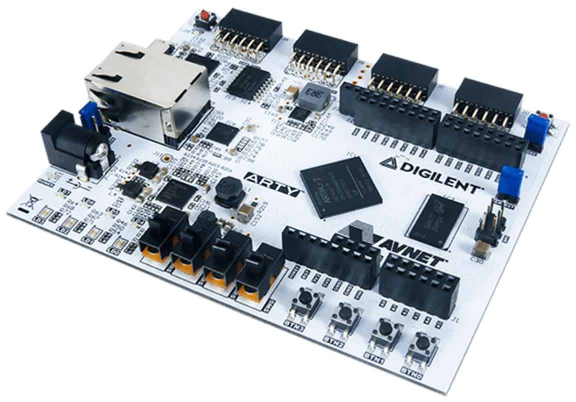
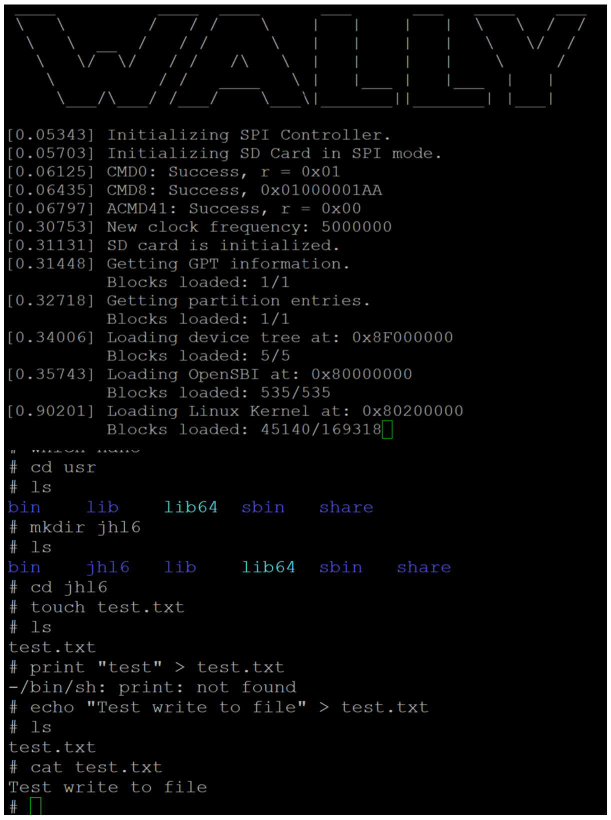


Fig. 2. The Arty A7 100-T FPGA Board [2]

With this board chosen, I was able to use Vivado to generate a bitfile to program the FPGA with. The WALLY GitHub provided a script to flash an SD card with a Linux Kernel image, but since I was using a VM I could not directly flash it. So, I modified the script to write the output to a file, which I was then able to download and flash from my laptop. I also copied the Vivado project to my computer to program the FPGA.

I was able boot Linux on the FPGA, and running basic commands to demonstrate that it was responsive and functional. Screenshots showing the FPGA board booting Linux and responding to commands can be seen below.

The next steps for this portion of the project involve deciding on a testing plan. I need to be able to test several designs and compare power usage for each, which may be done on the FPGA or in simulation.



```
[0.05343] Initializing SPI Controller.
[0.05703] Initializing SD Card in SPI mode.
[0.06125] CMD0: Success, r = 0x01
[0.06435] CMD8: Success, 0x01000001AA
[0.06797] ACMD41: Success, r = 0x00
[0.30753] New clock frequency: 5000000
[0.31131] SD card is initialized.
[0.31448] Getting GPT information.
          Blocks loaded: 1/1
[0.32718] Getting partition entries.
          Blocks loaded: 1/1
[0.34006] Loading device tree at: 0x8F000000
          Blocks loaded: 5/5
[0.35743] Loading OpenSBI at: 0x80000000
          Blocks loaded: 535/535
[0.90201] Loading Linux Kernel at: 0x80200000
          Blocks loaded: 45140/169318
# cd usr
# ls
bin     lib     lib64  sbin   share
# mkdir jhl6
# ls
bin     jhl6   lib     lib64  sbin   share
# cd jhl6
# touch test.txt
# ls
test.txt
# print "test" > test.txt
-/bin/sh: print: not found
# echo "Test write to file" > test.txt
# ls
test.txt
# cat test.txt
Test write to file
#
```

Fig. 3. Linux Booting and Running on WALLY

C. *Design Specifications*

Our goal is to create a fully functional RISC-V processor with ultra low-power usage. In order to minimize power usage, we will need to cut down as many extensions as possible while still keeping the core functionality of the chip. To identify the most power-efficient configuration, this semester's work will focus on comparing the power usage of several design components using simulation and FPGA testing. Components under investigation include the cache configuration, the division unit, and the branch prediction logic. The results of this analysis will be used to decide on the best architectural configuration. The current design plan specifies the following features:

- A **single-core, single-threaded processor**.
- An **integer-only** instruction set.
- Inclusion of the **base instruction set**.
- Inclusion of the **compressed (C)** instruction set to reduce code memory usage.
- Inclusion of the **minimal multiplication (Zmmul)** extension. This implements multiplication but not division, allowing for increased performance while avoiding the full area requirement of division circuitry.

The design specification for the SPARK-V core maintains a high degree of flexibility, prioritizing power efficiency while allowing for targeted performance enhancements. Additional components, such as a dedicated division unit or an instruction/data cache, may be incorporated if testing determines their benefits outweigh the marginal power cost. The decision to include a division unit, for instance, will depend on whether its performance gain in critical biomedical signal processing routines justifies the increased complexity and area overhead, especially since multiplication is already included. Similarly, the incorporation of a cache will be evaluated by comparing its power consumption against the energy savings achieved by reducing external memory accesses. This iterative process, which involves comparing the power usage of several different designs, ensures that the final architecture remains energy-conscious. Therefore, the core will only adopt new extensions or subsystems that offer a substantial, application-specific return on investment in terms of power-to-performance ratio. This also means that future testing will require some examples of programs and routines that are common in biomedical sensors, so that functional unit usage can be measured.

In addition to these decisions that affect the core of WALLY, I also need to decide how to manage the peripheral connections WALLY contains. As per the GitHub project, these components are referred to as the ”uncore” module. These include ROM, RAM, timers, GPIO, and communication over UART. While the design of the core has been specified above, additonal work will be needed to decide which uncore components will be used, and what settings will be applied to them.

IV. Discussion

At this point in the semester, I have been able to set up the needed development environment on ORION and validate the base WALLY design by using it to boot Linux on an FPGA. I also defined an initial architectural specification with the goal of maximizing power-efficiency for wearable biomedical applications. This plan focuses on a minimalist design through the reduction of extensions and add-ons. The current specification is a single-core, single-threaded, integer-only processor utilizing the base instruction set, the compressed (C) extension to reduce code size, and the multiplication (Zmmul) extension (excluding division) for performance. The next focus is to test and finalize the best architectural configuration based on the initial design specification. This will involve using simulation and FPGA testing to compare the power usage of key design elements, such as caches, division unit implementation, and branch prediction.

Looking ahead to next semester, the project will move into the physical design phase. The primary goal is to start from the optimized architectural configuration and apply implementation-level decisions to achieve the overall objective of a 30% reduction in system power. Ultimately, the aim is to produce a validated, energy-efficient processor ready for tapeout. I am also looking into additional power-reduction possibilities, including a paper that proposes an entirely battery-less RISC-V processor, which would be an extremely interesting implementation if possible with this design [3]. This could allow for a truly mobile processor for medical applications, but may introduce new challenges.

Acknowledgment

I would like to thank Dr. Ray Simar for his support on this project, as well as Dr. James Stine and Jacob Pease from Oklahoma State University for their continued guidance. I would also like to thank David Harris and Sarah Harris for granting pre-publication access to their textbook covering the WALLY design which has helped in understanding the processor. Additionally, thanks to the Rice Electrical and Computer Engineering Department for funding this project.

CRediT Author Statement

Justin Lebeau: Conceptualization, Investigation, Methodology, Software, Validation, Writing

References

[1] https://github.com/openhwgroup/cvw
[2] https://digilent.com/shop/arty-a7-100t-artix-7-fpga-development-board/
[3] D. S. Truesdell, J. Breiholz, S. Kamineni, N. Liu, A. Magyar and B. H. Calhoun, ”A 6–140-nW 11 Hz–8.2-kHz DVFS RISC-V Microprocessor Using Scalable Dynamic Leakage-Suppression Logic,” in IEEE Solid-State Circuits Letters, vol. 2, no. 8, pp. 57-60, Aug. 2019, doi: 10.1109/LSSC.2019.2938897.