


Low-latency I/O RISC-V CPU core in FPGA fabric - Atharva Kashalkar


BeagleBoard.org
May 16, 2024

Table of contents

Chapter 1

Introduction

Implementation of PRU subsystem on BeagleV-Fire's FPGA fabric, resulting in a real-time microcontroller system working alongside the main CPU, providing low-latency access to I/O .

1.1 Summary links

- **Contributor:** Atharva Kashalkar
- **Mentors:** Jason Kridner, Cyril Jean
- **Code Repository:** TBD

1.2 Status

This project is currently just a proposal.

1.3 Proposal

Accounts created across OpenBeagle, Discord and Beagle Forum
PR was created for the cross-compilation task.

1.4 About

- **Resume:** Find my resume here
- **Forum:** u/roger18 (Atharva Kashalkar)
- **OpenBeagle:** Roger18 (Atharva Kashalkar)
- **Github:** RapidRoger18 (Atharva Kashalkar)
- **School:** Veermata Jijabai Technological Institute (VJTI)
- **Country:** India
- **Primary language:** English Hindi
- **Typical work hours:** 9 AM-11 PM Indian Standard Time
- **Previous GSoC participation:** First Time Applicant

Chapter 2

Project

Project name: Low-latency I/O RISC-V CPU core in FPGA fabric.

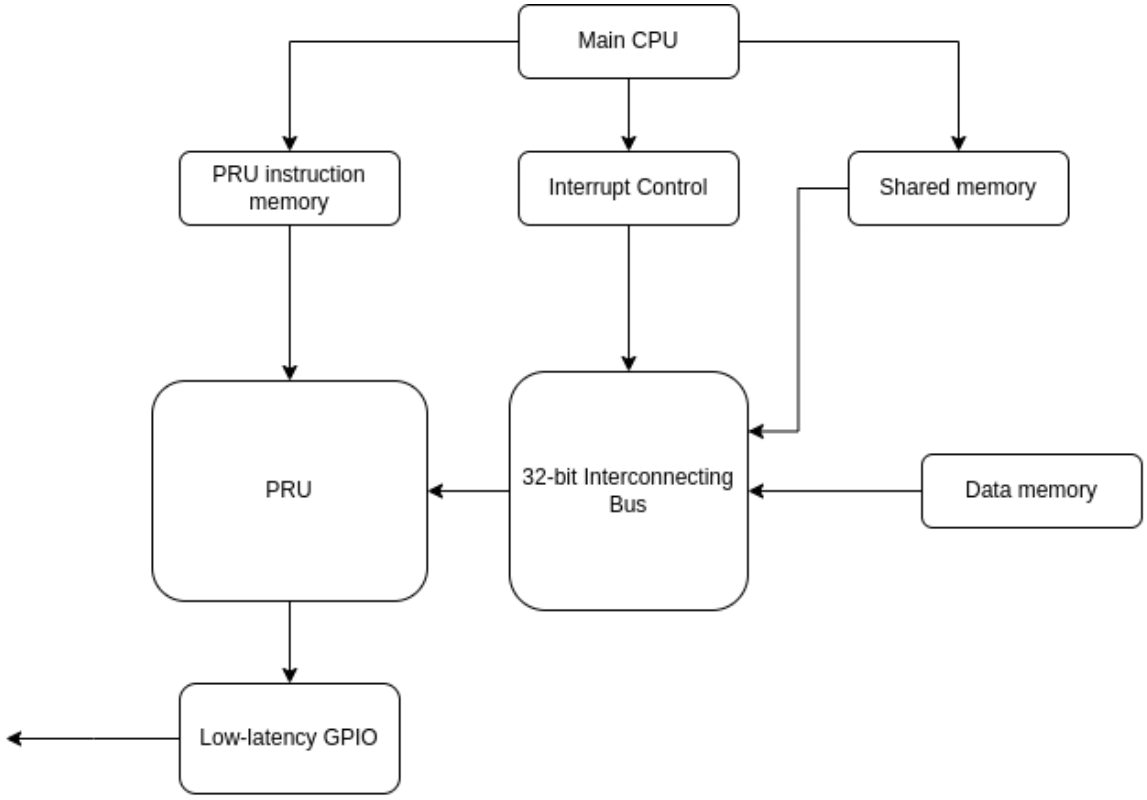
2.1 Description

To provide the capability of a Programmable Real-time Unit Industrial Control SubSystem (PRU-ICSS), which is present on several BeagleBone boards, I propose to deploy an existing RISC-V 32IM core with a customized Instruction Set Architecture on FPGA Fabric present on BeagleV-Fire. The goal of this deployment is to provide high bandwidth between the CPU and I/O, resulting in a on-board microcontroller.

2.2 Goals and Objectives

The ultimate aim of this project is to have a functional Risc-V soft core on the BeagleV-Fire FPGA fabric, which will be functionally equivalent to a PRU subsystem on the BeagleBone Black. Together with ultra-low latency I/O operations, this core will be able to execute Risc-V instructions acting as a microcontroller. BeagleV-Fire will feature a functional PRU-comparable subsystem on its FPGA fabric by the project's conclusion.

The programmable nature of the PRU, along with its access to pins, events, and all SoC resources, provides flexibility in implementing fast real-time responses, specialized data handling operations, custom peripheral interfaces, and in offloading tasks from the other processor cores of the system-on-chip (SoC). This is a basic block diagram for PRU design:



Source:- Inspired from https://inst.eecs.berkeley.edu/~ee192/sp20/files/am335x-pru.pdf

Based on the block diagram above we can divide the project into two stages. Stage 1 will include deciding the most suitable pre-existing core based on its performance concerning its size, complexity, accessibility, etc. Customizing the core to meet project requirements and integrating it with BeagleV-Fire gateware, followed by extensive Verification of the modifications using preset verification techniques. This stage will conclude with having a stable communication protocol between the PRU and main CPU. This will ensure that the PRU's instruction memory can be written by main CPU and have an interrupt signal to control the PRU.

At Stage 1 Evaluation, a functional Risc-V PRU with CPU access support will be prepared for deployment.

The primary aim for Stage 2 will be establishing the necessary I/O functions, and stable communication between the PRU and these I/O. I/O and other peripherals can be mapped to particular special addresses to which the PRU can write, which will enable the user to configure the I/O devices to any parameters using a PRU C Library, which will try to abstract I/O usage for the user. Changes to be made to build scripts to add an option of including a PRU design to the gateware when flashing the Board. Updating Device Tree Overlay to enable PRU whenever necessary. Having a test program for users to get familiar with the use of PRU.

After Stage 2, BeagleV-Fire will host a fully functional PRU system that can be controlled by invoking a specific function within the main CPU.

2.3 Software

- Verilog HDL.
- Verilator.
- Libero SoC suite.
- Microchip Softconsole
- ModelSim ME.
- Linux.

- OpenBeagle CI.

2.4 Hardware

Ability to program BeagleV-Fire using serial port and set up JTAG for effective debugging.

Chapter 3

Timeline

3.1 Timeline summary

Date	Activity
April	Understand detailed use cases of existing cores and shortlist them based on requirements
May 1	Start bonding - Discussing implementation methods with Mentors
May 15	College Examinations
June 1	Start coding and introductory video
June 3	*Milestone #1, Introductory YouTube video (June 3rd)*
June 10	*Milestone #2, Modifiying RV core (June 10th)*
June 17	*Milestone #3, Verification of core (June 17th)*
June 24	*Milestone #4, Remote-proc setup (June 24th)*
July 1	*Milestone #5, Setup Device Tree (July 1st)*
July 8	Submit midterm evaluations
July 15	*Milestone #6, Mapping I/O (July 15th)*
July 22	*Milestone #7, Verification of mapped I/O (July 22nd)*
July 29	*Milestone #8, Add customizability (July 29th)*
August 5	*Milestone #9, Setup Scripts (Aug 5th)*
August 12	*Milestone #10, Documentation and Tutorial(Aug 12th)*
August 19	Submit final project video, submit final work to GSoC site, and complete final mentor evaluation

3.2 Timeline detailed

3.2.1 Community Bonding Period (May 1st - May 15th)

- Get to know mentors and discuss project implementation.
- read documentation, and get up to speed to begin working on the projects
- shortlisting pre-existing cores based on initial assessment, by reading available documentation.

3.2.2 Coding begins (May 27th)

3.2.3 Milestone #1, Introductory YouTube video (June 3rd)

- Make an introductory video
- Selecting the best-suited core by comparing their functionality, size, availability of extensions, etc.
- Setting up remote access on BeagleV-Fire and completing the LED-blink tutorial given in the documentation.

3.2.4 Milestone #2, Modifiying RV core (June 10th)

- Modification of the selected core to meet project requirements, like memory configurations, interrupt control, etc.
- Removing unnecessary extensions to reduce size and complexity, without changing its efficiency.

3.2.5 Milestone #3, Verification of core (June 17th)

- Performing Verification of PRU core using pre-determined verification methods by using Verilator or any other Verification software.
- This is to make sure the PRU functions as required after modifications.

3.2.6 Milestone #4, Remote-proc setup (June 24th)

- Integration of PRU core with BeagleV-Fire gateware. This is to ensure PRU deployment through BeagleV-Fire gateware.
- Set up Remote-proc Interfacing between PRU and main CPU. This will ensure CPU access to PRU's instruction memory.

3.2.7 Milestone #5, Setup Device Tree (July 1st)

- Coninuing to work on Remote-proc framework.
- Ensuring stable PRU workflow.
- Setting up Device Tree Overlay to include PRU when necessary.

3.2.8 Submit midterm evaluations (July 8th)

- Complete pending Stage 1 tasks, if any.

Important: July 12 - 18:00 UTC: Midterm evaluation deadline (standard coding period)

3.2.9 Milestone #6, Mapping I/O (July 15th)

- Implementing required I/O functionalities.
- Mapping I/O to registers to enable interaction with CPU.

3.2.10 Milestone #7, Verification of mapped I/O (July 22nd)

- Verification of these I/O operations using simulation and generating common use cases.
- Deploying these modules on BeagleV-Fire FPGA and testing their latency and real-time application.

3.2.11 Milestone #8, Add customizability (July 29th)

- Adding customizability to CPU. This is to allow user to make small changes to CPU to observe the changes in output.
- This will allow user to learn internal working of a RISC processor.

3.2.12 Milestone #9, Setup Scripts (Aug 5th)

- Setting up gateware scripts wherever changes are needed. This will grant users easier access to CPU.
- Editing TCL scripts wherever necessary.

3.2.13 Milestone #10, Documentation and Tutorial(Aug 12th)

- Documenting the project and ways to access PRU on docs.beagleboard.org.
- Having an LED Blink tutorial for users to familiarize themselves with the PRU.

3.2.14 Final YouTube video (Aug 19th)

Submit final project video, submit final work to GSoC site, and complete final mentor evaluation

3.2.15 Final Submission (Aug 24th)

Important: August 19 - 26 - 18:00 UTC: Final week: GSoC contributors submit their final work product and their final mentor evaluation (standard coding period)

August 26 - September 2 - 18:00 UTC: Mentors submit final GSoC contributor evaluations (standard coding period)

3.2.16 Initial results (September 3)

Important: September 3 - November 4: GSoC contributors with extended timelines continue coding

November 4 - 18:00 UTC: Final date for all GSoC contributors to submit their final work product and final evaluation

November 11 - 18:00 UTC: Final date for mentors to submit evaluations for GSoC contributor projects with an extended deadline

---

Chapter 4

Experience

This project will require prior knowledge of Risc-V ISA, FPGA programming, Assembly, and Verilog.

I have previously worked on a project Risc-V 32IM core where I contributed towards the synthesis of a Risc-V core using Vivado Design suite and flashed it on UPduino FPGA using Yosys framework. This CPU was successfully able to run operations like the Fibonacci series and factorial of a number.

I was a Team Leader of the Semi-finalist Team in E-Yantra Robotics Competition (eYRC), IIT Bombay on Theme “AstroTinker Bot”, where we had to develop a single cycle Risc-V 32I core that was capable of implementing Dijktra Algorithm written in C code and cross-compiled to Risc-V binary instructions. This Competition exposed me to various debugging methods using JTAG connection through Quartus Prime Lite software. DE0-NANO development board with Cyclone IV FPGA was used to control the bot. The bot used PID-based Line Following and Electromagnet to pick and place blocks through an arena representing a Space Station. Verilog HDL codes for the project can be found here

I have prior experience with ESP32 microcontrollers for small projects.

Chapter 5

Approach

As a starting point, I have forked the BeagleV-Fire gateware Repository and completed the Blinky Tutorial as specified on BeagleBoard Docs

Continuing on this I have set up Libero SoC suite and Softconsole on my Local machine that will provide an easy debugging method using JTAG. I have tested this setup by implementing a simple UART receiver and simulating it in ModelSim ME using a simple testbench.

The concept of having a microprocessor and a microcontroller on a single SoC is amazing. I can see a variety of applications where a SoC like BeagleV-Fire would be ideal. This inspires me to take on this project and give my contribution to the open-source community. I have my full commitment to this project and am willing to put in complete efforts to complete the project within the allocated time.

I would also like to keep improving with the project after GSoC.

5.1 Contingency

If I get stuck on my project and my mentor isn't around, I will use the following resources:-

- BeagleV-Fire
- PRU-Documentation
- Picorv32 Docs
- VexRiscv Docs
- PRU cookbook
- TI PRU documentation

Moreover, the BeagleBoard community is very helpful in resolving doubts, I will use OpenBeagle forums to clarify any doubts left after referring to the above resources.

5.2 Benefit

This project will not only improve the use cases of BeagleV-Fire but also provide a very fast real-time subsystem that can be used as a microcontroller. It will provide the following functionalities: -

- Reduce memory and I/O resource consumption on the main CPU by performing basic computations in PRU itself thus providing more resources to perform complex tasks.
- Provide a real-time interface for fast, deterministic operations.

- Reprogammability of FPGA will allow to deployment of PRU only when necessary, thus providing additional logical elements when PRU is not being used.
- The configurable nature of FPGA will allow multiple levels of customizations and configurations on PRU, enabling the user to efficiently meet their requirements in the lowest possible resources.

The success of this project will result in an all-in-one SoC meeting all the requirements of BeagleBoard.org community.

5.3 Misc

PR request for cross-compilation task #182