Rogues Gallery - Reconfigurable Computing

Tarun Maddali, Kyle Suter, Jonathan Wu, Aadi Kapur (Spring 2020)

Project Goals:

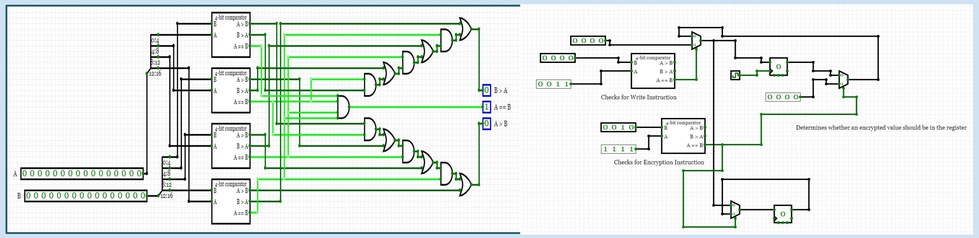
The Reconfigurable Computing team enhances pipelined-CPU runtimes by brainstorming and implementing new and non-conventional hardware schemas. The idea is to think of something brand new (ridiculous or not), and challenge the industry standards.

Project Overview and Methods:

- ➔ SRBP (Smart Register Branch Predictor):
  - ◆ serves as register file and branch predictor.
  - ◆ Sorts registers in real time.
  - ◆ Pros:
    - 100% accuracy.
  - ◆ Cons:
    - Higher energy consumption.
    - write cycle time extended.
- ➔ Encryption:
  - ◆ post-quantum safe.
  - ◆ Algorithm
    - AES-128 (per instructional encryption).
    - Data sent to specified encryption register.
    - Decryption instruction reverses changes.
- ➔ Caching:
  - ◆ Faster way to access some memory bytes
  - ◆ Focusing on reducing cache-miss rate
  - ◆ Brainstorm replacement algorithms so queried data is in cache most of the time

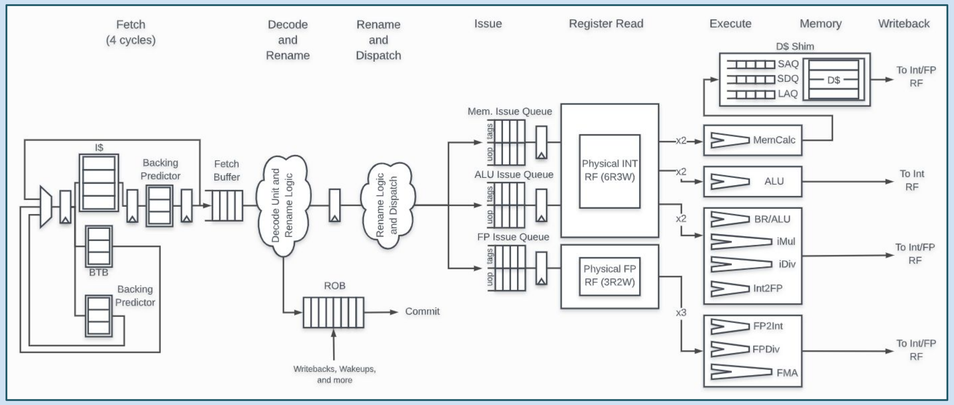
Design Methodology

Initial Schematics:



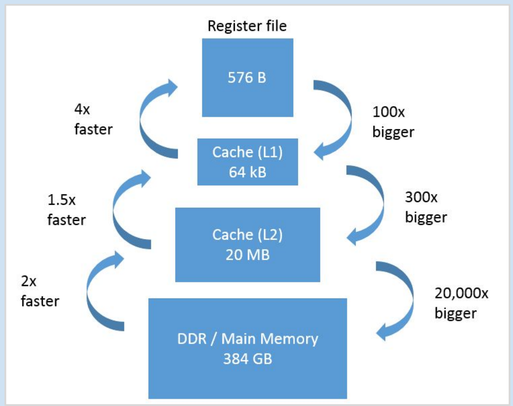
Above Left: a custom 32-bit ripple comparator.
Above Right: A hardware encryption circuit that outlines the algorithm

BOOM: A RISC-V Implementation:



Above: BOOM is a popular implementation of the RISC-V ISA. Our plan is to append our accelerations into the BOOM architecture for performance increases.

Right: A cache hierarchy. We plan to integrate a caching system into the BOOM pipeline to decrease the time it takes to access memory.



Lessons Learned

- ➔ CIrcuit Verse (basic designs).
- ➔ PyRTL (initial implementation).
  - ◆ Intuitive Python library but lacks documentation.
- ➔ Chisel (final implementation).
  - ◆ Scala-based embedded language with clearer correlation to the hardware it was simulating.
- ➔ ISAs
- ➔ Encryption
- ➔ Branch Prediction
- ➔ Caching

Next Steps and Future Challenges

- ➔ Near-complete Chisel implementations, though not heavily tested and not optimized; plan to consolidate these implementations.
- ➔ With 1-cycle branch-prediction, a branch target buffer to forgo computing target destination each time, only 1 cycle is needed for the entire branching process.
- ➔ Integrate completed designs with the RISC-V BOOM architecture, benchmark performance changes.

Rogues Gallery - Reconfigurable Computing

Tarun Maddali, Kyle Suter, Aadi Kapur, Yihan Jiang, Roye Eshed (Fall 2020)

Project Goals:

The Reconfigurable Computing team enhances pipelined-CPU runtimes by building upon Open Source frameworks and implementing innovative hardware schemas. Our goal is to think of hardware in a non-conventional fashion and challenge the industry standard.

Project Overview and Methods:

➔ *SRBP (Smart Register Branch Predictor):*
- ◆ Serves as register file and branch predictor.
- ◆ Sorts registers per register-write, in an insertion-sort fashion.
- ◆ Tradeoffs: 100% accuracy, but higher energy consumption and write cycle time extended.

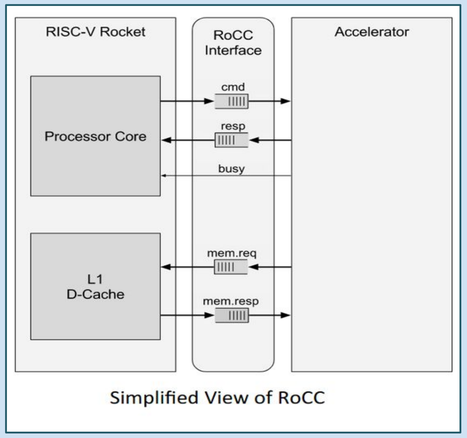
➔ *Encryption: Post-quantum safe*
- ◆ AES-128 (per instructional encryption).
- ◆ Data sent to specified encryption register.
- ◆ Decryption instruction reverses changes.

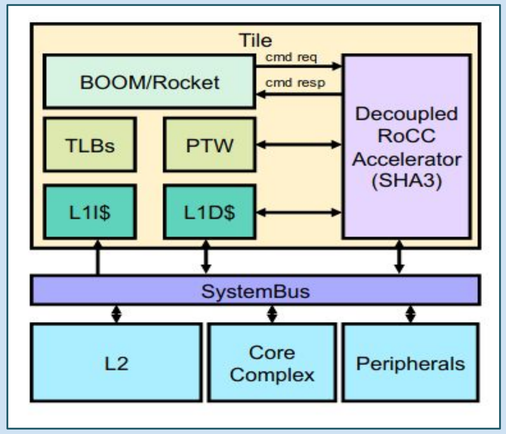
➔ *Hardware I/O and Interrupt Support:*
- ◆ Act as an interface for BOOM to incorporate other devices.
- ◆ Implement Interrupt Support for I/O + Traps.

➔ *RoCC Accelerator Integration:*
- ◆ Began development on a assembly instruction mapping.
- ◆ Purpose was to eventually map the Chisel modules to BOOM and memory using RoCC.

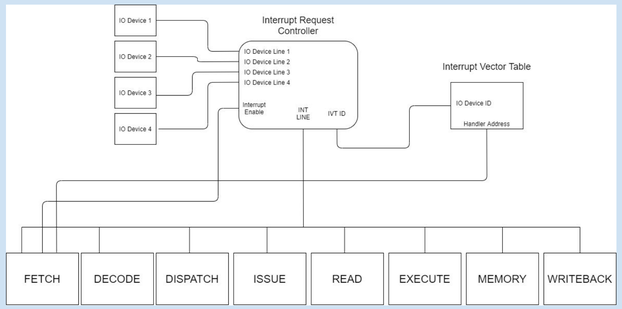
Design Methodology

Rocket Chip Coprocessor (RoCC) Interface:



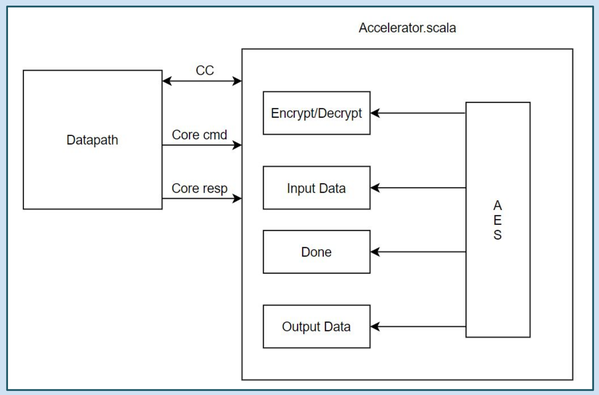


Above Left: Interface between a custom accelerator, the Rocket Chip processor and memory.
Above Right: RoCC interface from the perspective of a top-level design file.



Above: Display of the use of the Interrupt Request Controller to handle Interrupts from IO Devices and execute the Interrupt Handler for devices..

Right: A block diagram that shows the integration of the AES and the accelerator by the Rocc interface.



Lessons Learned

- ➔ *Chisel*: Scala-embedded HDL
- ➔ *ISAs*: Risc-V and custom instructions
- ➔ *Encryption*: AES-128
- ➔ *Branch Prediction*: Accuracy tradeoffs
- ➔ *RoCC Interface* (BOOM-Core integrations)
- ➔ *Hardware I/O and Interrupt Support*
- ➔ *Assembly*: new instr BOOM-integration
- ➔ *Chipyard Custom Accelerator Configuration*
- ➔ *Coding Practices* (GitHub, Code Review)

Next Steps and Future Challenges

- ➔ Further test completed modules, and look for edge case bugs.
- ➔ Implement a Branch Target Buffer, so that the Branch *Decision* and Branch *Target* are both known in 1-cycle.
- ➔ Ensure that Encryption and Decryption work on a wide range of values.
- ➔ Integrate completed designs with the RISC-V BOOM architecture, benchmark performance changes.
- ➔ Create new assembly instructions that account for our modules within the BOOM processor.

Rogues Gallery - Reconfigurable Computing

Roye Eshed, Varun Valada (Fall 2021)

Project Goals:

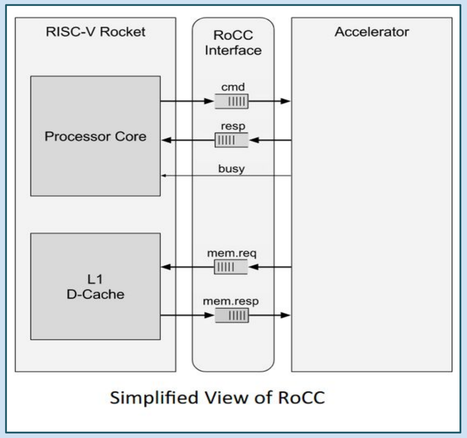
The Reconfigurable Computing team enhances pipelined-CPU runtimes by building upon Open Source frameworks and implementing innovative hardware schemas. Our goal is to think of hardware in a non-conventional fashion and challenge the industry standard and implementing those schemas together to create innovative hardware microarchitectures.

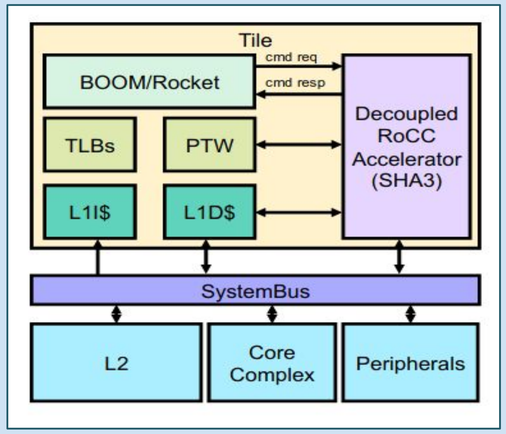
Project Overview and Methods:

- ➔ *AES: Post-quantum safe*
  - ◆ AES-128 (per instructional encryption).
  - ◆ Data sent to specified encryption register.
  - ◆ Decryption instruction reverses changes.
- ➔ *Hardware I/O and Interrupt Support:*
  - ◆ Act as an interface for BOOM to incorporate other devices by having an interrupt controller that stall the dispatch queue until the interrupt completes
  - ◆ Implement Interrupt Support for I/O + Traps that allows for native hardware interrupts rather than merely software ones.
- ➔ *Chipyard:*
  - ◆ Finalized the VM configuration to compile BOOM and Chipyard configurations.
  - ◆ Purpose was to eventually map the Chisel modules to BOOM and memory.
  - ◆ A RoCC interface helps the processor communicate with our custom Chisel modules.

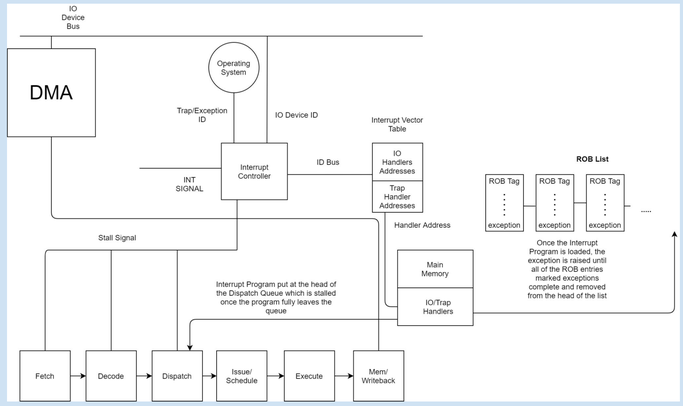
Design Methodology

Rocket Chip Coprocessor (RoCC) Interface:





Above Left: Interface between a custom accelerator, the Rocket Chip processor and memory. Above Right: RoCC interface from the perspective of a top-level design file for a similar SHA3 encryption module.



Left: Showing the Interrupt Controller receiving an interrupt request, putting it into the head of the dispatch queue and then updating the rob entries with an exception flag raised until the interrupt program completes and then sending the date through DMA if needed.

Right:The waveform for the default sample chipyard.

Lessons Learned

- ➔ *Chisel*: Scala-embedded HDL
- ➔ *ISAs*: Risc-V and custom instructions
- ➔ *Encryption*: AES-128
- ➔ *RoCC Interface* (BOOM-Core integrations)
- ➔ *Hardware I/O and Interrupt Support: Challenges in Superscalar Pipelines*
- ➔ *Assembly*: new instr BOOM-integration
- ➔ *Chipyard Custom Accelerator Configuration*
- ➔ *Coding Practices* (GitHub, Code Review)
- ➔ Environment Setup

Next Steps and Future Challenges

- ➔ Further test completed modules, and look for edge case bugs.
- ➔ Readapt the work done in Interrupt Support for basic pipelines this semester into pipeline support for Superscalar Processors
- ➔ Ensure that Encryption and Decryption work on a wide range of values.
- ➔ Finish integrating completed designs with the RISC-V BOOM architecture, benchmark performance changes.
- ➔ Create a way for the BOOM processor to interface with our integrated AES/SRBP accelerator.

Rogues Gallery - Reconfigurable Computing

Roye Eshed, Varun Valada (Fall 2021)

Project Goals:

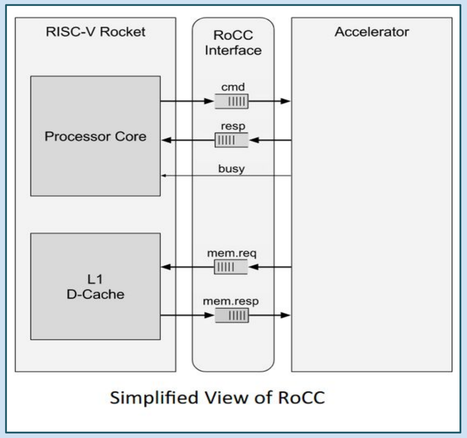
The Reconfigurable Computing team enhances pipelined-CPU runtimes and execution by building upon Open Source frameworks and implementing innovative hardware schemas. Our goal is to think of hardware in a non-conventional fashion and challenge the industry standard and implementing those schemas together to create innovative hardware microarchitectures.

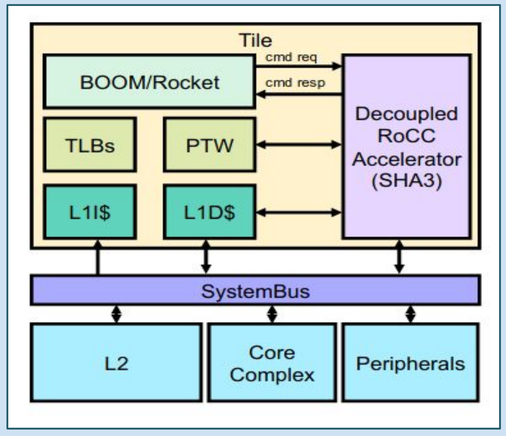
Project Overview and Methods:

- ➔ *AES: Post-quantum safe*
  - ◆ AES-128 (per instructional encryption).
  - ◆ Data sent to specified encryption register.
  - ◆ Decryption instruction reverses changes.
- ➔ *Hardware I/O:*
  - ◆ Implemented MMIO in Scala which can allow for devices to write and read back by accessing memory addresses above actual physical memory that correspond to physical devices.

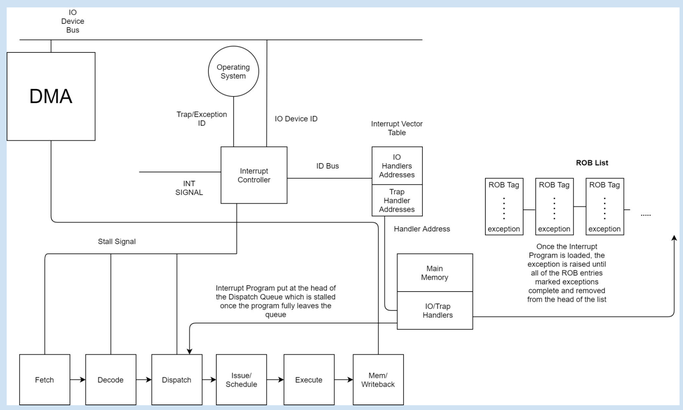
Design Methodology

Rocket Chip Coprocessor (RoCC) Interface:



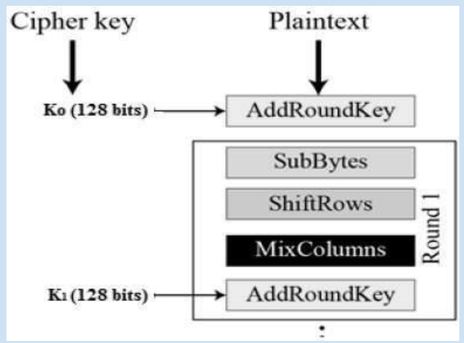


Above Left: Interface between a custom accelerator, the Rocket Chip processor and memory.
Above Right: RoCC interface from the perspective of a top-level design file for a similar SHA3 encryption module.



Left: Showing the Interrupt Controller and the MMIO system which uses the DMA to read and write data between the IO devices connected to the system.

Right: AES Encryption algorithm for one round



Lessons Learned

- ➔ *Chisel*: Scala-embedded HDL
- ➔ *Encryption*: AES-128
- ➔ *RoCC Interface* (BOOM-Core integrations)
- ➔ *Hardware I/O and Interrupt Support: Challenges in Superscalar Pipelines*

Next Steps and Future Challenges

- ➔ Further test completed modules, and look for edge case bugs.
- ➔ Implement the Interrupt System for the Superscalar Processor on top of the MMIO system that was implemented.
- ➔ Finish integrating completed designs with the RISC-V BOOM architecture, benchmark performance changes.
- ➔ Create a way for the BOOM processor to interface with our integrated AES/SRBP accelerator.
- ➔ Optimize hardware implementation of the encryption algorithm, eg. pipelining