RISC-V Hackathon Summary Report

Student Names & IDs
[**Alaa Shaaban**
Marsel Nasr]

Introduction

As part of the RISC-V Hackathon event at the Hebrew University, we developed a custom hardware accelerator to enhance the performance of SHA256 algorithm. SHA256 plays a critical role in cryptographic systems, and its computational intensity makes it a good candidate for hardware acceleration. Our objective was to offload the core transformation step of the algorithm to an FPGA connected to a RISC-V processor, reducing computation time and CPU load.

Background

SHA256 processes input data by dividing it into 512-bit blocks and applying 64 rounds of logical and arithmetic operations to each block. Although the algorithm is efficient by design, executing all these rounds purely in software can become a bottleneck, particularly on resource-constrained or embedded systems.
To address this, we implemented a hardware accelerator for SHA256 on an FPGA. This allows us to offload the compute-intensive hashing operations from the CPU, significantly improving performance while maintaining compatibility with existing software. Communication between the RISC-V processor and the hardware SHA256 core is handled via memory-mapped I/O, enabling straightforward integration and ensuring that the software interface remains unchanged.

Goals

1. To design and implement a cryptographic accelerator for the SHA256 algorithm.
2. To achieve a significant reduction in execution time (clock cycles) while ensuring correctness.
3. To minimize the use of FPGA hardware resources such as LUTs, MUXes, DSPs, and memory blocks.

The Path and Tools Used to Achieve the Goal

To achieve our goals, we took a hands-on engineering approach that combined both software and hardware development. Our process started by getting familiar with the existing C implementation, then pinpointing where the code was slowing things down. From there, we focused on designing an interface that allowed the software to communicate with the hardware on the FPGA.

Development Process:

1. We started by thoroughly reviewing and understanding the provided C implementation of SHA256.
2. Identified SHA256Transform() as the most compute-intensive function.
3. Reimplemented this function in hardware (accelerator.sv).
4. Modified a register interface module (accelerator_regs.sv) for communication.
5. Modified the C code (sha256.c) to interact with the accelerator via memory-mapped I/O
6. Measured performance gains

Solution Description

Our solution accelerates the SHA256 algorithm by offloading its computationally intensive core transformation function, SHA256Transform(), from the RISC-V CPU to a custom hardware accelerator implemented on an FPGA. The key aspects of this include both hardware and software components working together:
Hardware Accelerator (SystemVerilog Implementation)

- **Accelerator(accelerator.sv)**:
  The hardware accelerator implements the core SHA256 rounds, performing 64 iterations of logical and arithmetic operations in parallel. This design exploits FPGA parallelism to significantly speed up computation. The functionality is as follows:

    - **Input Stage**: The accelerator receives a 512-bit message block divided into sixteen 32-bit words and the SHA256 internal state comprising eight 32-bit words.
    - **Message Scheduler**: Generates 48 additional 32-bit words from the initial sixteen-word input using SHA256-specific functions (SIG0 and SIG1), producing a total of 64 words required for the computation.
    - **Compression Logic**: Executes the SHA256 compression rounds in parallel, applying rotations (EP0, EP1), conditional operations (CH, MAJ), additions, and XOR operations.
    - **State Registers**: Stores intermediate results and the final SHA256 state, ready for retrieval by the CPU upon completion.
    - **Control Mechanism**: Uses a state machine controlled via memory-mapped registers (REG_CONTROL) for synchronization with the CPU.

- **Memory-mapped Registers**: The accelerator interacts with the processor using memory-mapped I/O registers. This involves defining specific addresses for message input (REG_MSG_BASE), SHA256 state input (REG_STATE_IN_BASE), SHA256 state output (REG_STATE_OUT_BASE), and a control register (REG_CONTROL) that initiates the operation and signals its completion.
- **Control Logic**: The FPGA accelerator uses a simple control logic mechanism (CTRL_GO and CTRL_DONE) to synchronize operations. The CPU sets the CTRL_GO bit to initiate hashing, and the FPGA sets the CTRL_DONE bit upon completion, signalling to the processor that the results are ready to be read.

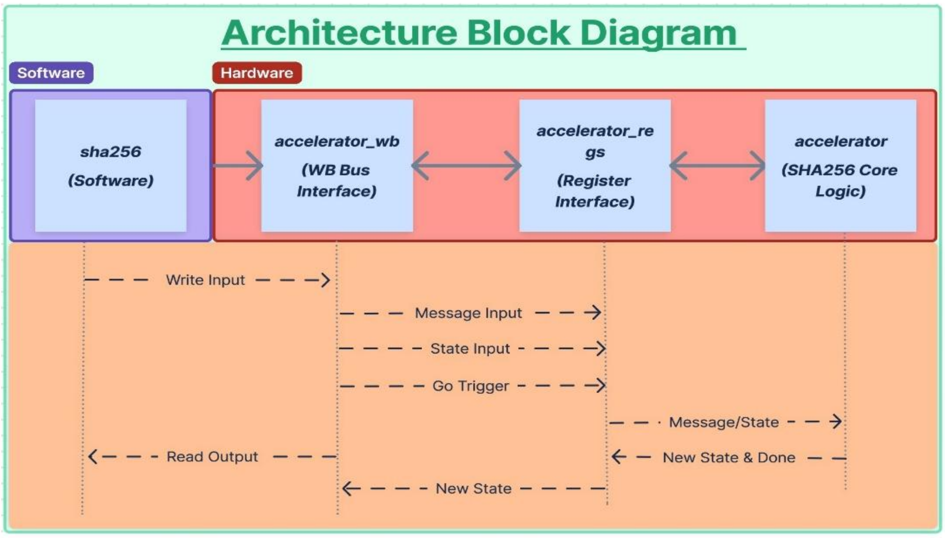
Software Integration (Modified C Implementation)

- The SHA256Transform() function, originally executed fully in software, was modified to transfer its computation to the FPGA. Instead of computing hash rounds in software, this function now:
    - Writes the current 512-bit message block into FPGA memory-mapped registers.
    - Writes the current SHA256 internal state into the FPGA.
    - Signals the FPGA to begin computation.
    - Waits for completion signalled by the FPGA hardware.
    - Reads the updated SHA256 state back from the FPGA registers.
- Despite hardware acceleration, the software interface (SHA256Init, SHA256Update, SHA256Final) remained the same as the original implementation.

This comprehensive integration allowed us to harness the parallel computing capabilities of FPGAs, greatly accelerating SHA256 computation.

Hardware vs. Software Partitioning:

Component	Location	Explanation
64-round SHA256 compression loop	Hardware (accelerator.sv)	Implemented entirely in RTL to reduce cycle count
Message block input/output	Hardware + Software	Written/read via registers
Initial/final state registers	Hardware + Software	Used to pass and update hash state
Control logic (start/done)	Hardware + Software	FSM and status flags in hardware, polled by software
Padding, message splitting	Software	These operations remain in SHA256Update() and SHA256Final()
Final hash formatting (hex)	Software	Final state is converted into printable output in C

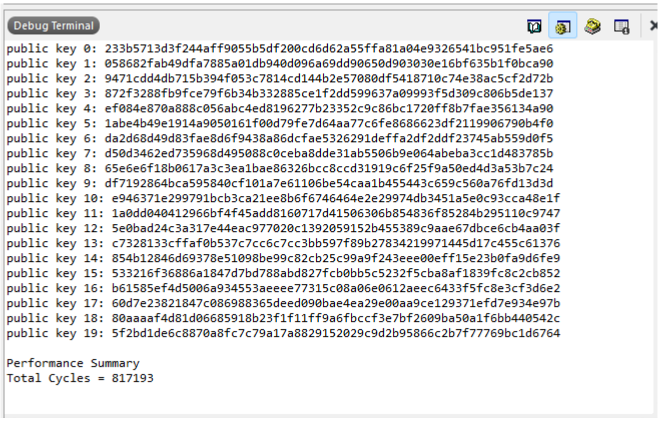

Architecture Block Diagram
Software
Hardware
sha256 (Software)
accelerator_wb (WB Bus Interface)
accelerator_regs (Register Interface)
accelerator (SHA256 Core Logic)
Write Input
Message Input
State Input
Go Trigger
Message/State
New State & Done
Read Output
New State

Development Environment and Project Results

Development Technologies:

- **Languages**: C (Software), SystemVerilog (Hardware)
- **EDA Tools**: Vivado 2023.2 for synthesis, simulation, and FPGA bitstream generation
- **Software Tools**: SEGGER Embedded Studio for writing and deploying the RISC-V sha256.c program
- **Target Hardware**: Digilent Nexys A7-100T development board featuring a Xilinx Artix-7 FPGA

The results achieved in terms of acceleration:



```
public key 0: 233b5713d3f244aff9055b5df200cd6d62a55ffa81a04e9326541bc951fe5ae6
public key 1: 058682fab49dfa7885a01db940d096a69dd90650d903030e16bf635b1f0bca90
public key 2: 9471cdd4db715b394f053c7814cd144b2e57080df5418710c74e38ac5cf2d72b
public key 3: 872f3288fb9fce79f6b34b332885ce1f2dd599637a09993f5d309c806b5de137
public key 4: ef084e870a888c056abc4ed8196277b23352c9c86bc1720ff8b7fae356134a90
public key 5: 1abe4b49e1914a9050161f00d79fe7d64aa77c6fe8686623df2119906790b4f0
public key 6: da2d68d49d83fae8d6f9438a86dcfae5326291deffa2df2ddf23745ab559d0f5
public key 7: d50d3462ed735968d495088c0ceba8dde31ab5506b9e064abeba3cc1d483785b
public key 8: 65e6e6f18b0617a3c3ea1bae86326bcc8ccd31919c6f25f9a50ed4d3a53b7c24
public key 9: df7192864bca595840cf101a7e61106be54caa1b455443c659c560a76fd13d3d
public key 10: e946371e299791bcb3ca21ee8b6f6746464e2e29974db3451a5e0c93cca48e1f
public key 11: 1a0dd040412966bf4f45add8160717d41506306b854836f85284b295110c9747
public key 12: 5e0bad24c3a317e44eac977020c1392059152b455389c9aae67dbce6cb4aa03f
public key 13: c7328133cffaf0b537c7cc6c7cc3bb597f89b27834219971445d17c455c61376
public key 14: 854b12846d69378e51098be99c82cb25c99a9f243eee00eff15e23b0fa9d6fe9
public key 15: 533216f36886a1847d7bd788abd827fcb0bb5c5232f5cba8af1839fc8c2cb852
public key 16: b61585ef4d5006a934553aeeee77315c08a06e0612aeec6433f5fc8e3cf3d6e2
public key 17: 60d7e23821847c086988365deed090bae4ea29e00aa9ce129371efd7e934e97b
public key 18: 80aaaaf4d81d06685918b23f1f11ff9a6fbccf3e7bf2609ba50a1f6bb440542c
public key 19: 5f2bd1de6c8870a8fc7c79a17a8829152029c9d2b95866c2b7f77769bc1d6764

Performance Summary
Total Cycles = 817193
```

Acceleration Achieved: Offloading the SHA256Transform() function to custom FPGA hardware yielded substantial performance gains over the original implementation. Key results include:

- **Total Execution Time: 817,193** clock cycles to hash 20 input messages
- **Average per Hash:** ~40,860 cycles
- **Performance Improvement:** Approximately **2.6× faster** than the baseline C implementation

Testing and Validation:

Correctness: Compared hash outputs from hardware and software for 20 test strings – all matched.

Performance: Measured cycle counts using RISC-V performance counters before and after acceleration.

Results: The hardware accelerator significantly reduced execution time. We also verified integration of two accelerators, though final tests used only one.

Attempt at Dual Accelerator Integration

To explore parallelization and further reduce execution time, we attempted to instantiate and use two hardware accelerators. While we successfully implemented the dual-accelerator in hardware, we encountered several challenges when attempting to use both simultaneously. Ultimately, only one accelerator was used in practice. The main issues faced during dual integration included:

- **Output Synchronization Failure**: Difficulty in reliably detecting when both accelerators completed processing.
- **Timing-Related Race Conditions**: Unpredictable timing led to non-deterministic behaviour.
- **Input Race Hazards**: Overlapping writes caused inconsistent input handling.
- **Shared Resource Conflicts**: Competition for shared signals or buses created errors.
- **Missing Edge Detection**: GO signal transitions were missed due to lack of edge-triggered logic.
- **Register Map Conflicts**: Confusing or overlapping addresses between accelerators.

While the dual-hardware path was promising, additional work is required to properly isolate and sync them.

Conclusion

Contribution of the Acceleration:

Through this project, we demonstrated that critical cryptographic functions like SHA256 can be accelerated on FPGA with direct CPU integration. Our custom hardware accelerator replaced the most time-consuming portion of SHA256 while keeping the software structure intact. Our hardware accelerator significantly improved performance, reducing execution time to **817,193 cycles** for 20 SHA256 hashes—about **2.6× faster** than the software-only version. The design is compact, correct, and scalable for future extensions.

Critical Reflection

- **What Worked Well**:
  - Correct and stable results
  - Clear Interface: memory-mapped registers simplified CPU-FPGA integration.
  - Quick Bottleneck Identification: Rapidly targeted SHA256Transform() for efficient acceleration.
  - Acceleration Efficiency: Hardware implementation reduced execution time by ~2.6×.
- **What Could Be Improved**:
  - Future designs could pipeline multiple blocks or hash batches for increased throughput
  - Dual-accelerator support
  - Exploring pipelining or loop unrolling could enhance hardware efficiency.