Implementing RISC-V Scalar Cryptography Extension Based on Chisel

Hongren Zheng
Instructor: Mingyu Gao

2022-06-06

Section 1

Background

Hardware crypto acceleration

- Crypto operations is pervasive for modern apps
  - HTTPS: when you are browsing websites
  - Bitlocker: encrypt you disk
- Imagine you are watching a 4K video from encrypted network/disk
  - Software-only can not handle this load
  - need specialized hardware!

RISC-V: the ISA

- To drive specialized hardware, we need Instruction Set Architecture (ISA)
  - ISA is the "bridge" between hardware and software
- RISC-V is an open standard ISA
  - First developed in Berkeley around 2010
  - Unlike proprietary/private standards like x86/ARM
- RISC-V recently ratified the scalar cryptography extension
  - In Autumn 2021
  - Covers block cipher/hash function: AES/SHA/SM4/SM3

Chisel: the HDL

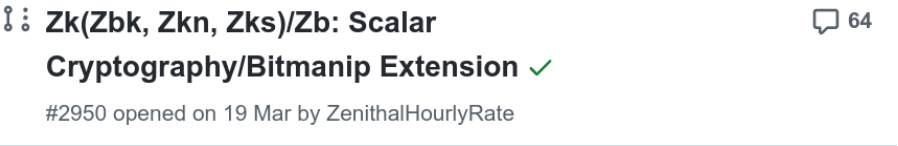
- To implement hardware, we need a hardware description language (HDL)
- Chisel is an HDL based on Scala
- Commonly used by RISC-V community
- Its syntax is more flexible compared with Verilog
  - Parameterized modules (code once, instantiate everywhere)
  - higher-order function like `map`/`reduce`/`filter`

Section 2

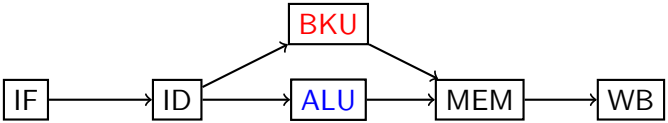
Designs

Circuit Design based on Rocket Chip

- To implement an extension, we must have a core
- Rocket Chip as the core
  - Open source RISC-V core, 2.2k stars
  - Taped out by Sifive
- Upstreamed my design to rocket chip
  - Got feedback from Andrew Waterman, the designer of RISC-V
  - Heated discussion on this (64 comments as shown below)



Five stage pipeline

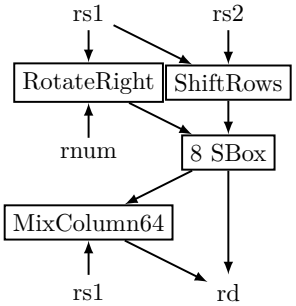


- Classical five stages: IF, ID, EXE, MEM, WB
- EXE means Execution, it often contains ALU (Arithmetic Logic Unit)
- My work: in EXE stage
  - Add BKU (Bitmanip Crypto Unit)
  - Replace ALU with ABLU (Arithmetic Bitmanip Logic Unit)

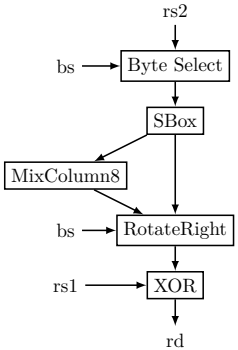
BKU and ABLU

- BKU: Implement bitmanip/scalar crypto extensions
  - Zb: Bit manipulation like rotation, crossbar
  - Zkn: NIST cipher suite, AES and SHA256/SHA512
  - Zks: ShangMi cipher suite, SM4 and SM3
- ABLU: merge common logic of bitmanip into ALU
  - ANDN: `a & ~b`
  - Can just be implemented along side AND: `a & b`
  - Result: `a & Mux(~b, b)`
  - Reusing 64 and gates
- My design can be used by both 32bit and 64bit architecture (thanks to Chisel)
- My design has small hardware cost
  - Merged many common logics
  - Area of an multiplier/divider

Merged logics



(a) AES for RV64

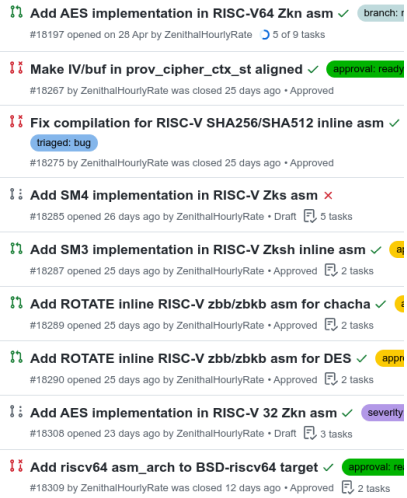


(b) AES for RV32

- Merged several instructions into one datapath
- Reuse common module between architecture (thanks to Chisel)

Software Design based on OpenSSL

- To evaluate my circuit design, I programmed corresponding software
- In **Assembly** language
  - performant crypto needs hand-written asm
- Exploit crypto ISA extension
  - Details and examples in my thesis
- Based on OpenSSL
  - widely-used crypto software lib
- Upstreamed my design to OpenSSL
  - 9 PRs, 3 merged

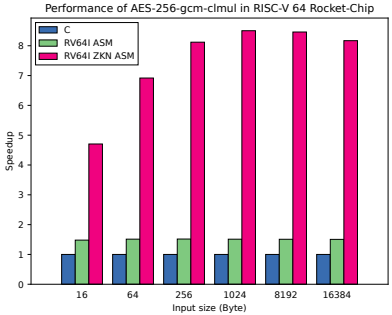


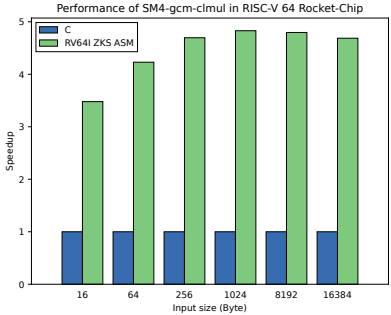
Section 3

Evaluation

Evaluation

- Running in xc7k325tffg900-2 FPGA, 100 MHz
- Baseline: software-only OpenSSL
- Target: Hardware accelerated OpenSSL
- For RV64, up to **10X** for AES, 5X for SM4
- For RV32, up to 4X for AES, 3.7X for SM4





Section 4

Q&A