# Implementation of Secured RISC-V Processor with Encryption Module

Karan Ravindra Misal
Student
Department of Electronics Engineering
(VLSI Design and Technology)
Shah And Anchor Kutchhi Engineering College
Mumbai, India
karan.17929@sakec.ac.in

Nirmol Munvar
Assistant Professor
Department of Electronics Engineering
(VLSI Design and Technology)
Shah And Anchor Kutchhi Engineering College
Mumbai, India
nirmol.munvar@sakec.ac.in

Dr. Subha Subramaniam
Professor
Department of Electronics Engineering
(VLSI Design and Technology)
Shah And Anchor Kutchhi Engineering College
Mumbai, India
subha.subramaniam@sakec.ac.in

Mr. Jagdish Sarode
Assistant Professor
Department of Electronics and Telecommunication
Shah And Anchor Kutchhi Engineering
College
Mumbai, India
jagdish.sarode@sakec.ac.in

***Abstract*—This paper presents the implementation of a secured RISC-V processor executing the RV32I base instruction set. The processor employs a load-store architecture and integrates an encryption unit between the memory and register bank to ensure data security for critical address regions.**

***Index Terms*—RISC-V, Load-Store Architecture, Encryption, Processor Security, RV32I**

## I. INTRODUCTION

When it comes to processor performance the most important thing is the processing speed and the power consumption. But our proposed processor follows a different approach by focusing on the security and encryption of the data in the processor. This paper describes the implementation of secured RISC V processor that execution RV32I base instruction set [1]. The processor follows a load store architecture where arithmetic and logical operations are performed only on the registers operands and the memory is accessed exclusively through load and store instructions.

The load store architecture also known as register-register architecture, offers several advantages including simplified instruction format, regular instruction timing and reduced complexity in control logic design. This architectural approach separates computation from memory access leading to more predictable performance characteristics.

Our proposed processor architecture adds an encryption unit between the memory and the register bank so that when ever data is passed between registers to memory the data will be encrypted and when the data is passed between memory to registers the data will be decrypted when the addresses are between the designated encryption addresses. This ensures the data security in the processor.

## II. RELATED WORK

Recent research integrates encryption accelerators into RISC-V processors to improve security and energy efficiency. Yang et al. [2] proposed a hybrid encryption-enabled RISC-V core optimized for IoT. Azad et al. [3] introduced RISE, a RISC-V SoC accelerating homomorphic encryption for edge computing. Zgheib et al. [4] extended RISC-V with AES hardware support for low-power IoT constraints.

## III. PROCESSOR ARCHITECTURE

### *A. Load-Store Architecture*

The processor adheres to a strict load-store model where:

- Arithmetic and logical operations are performed only on registers [5].
- Memory access is conducted exclusively through load and store instructions [5], [6].
- Data must be explicitly loaded from memory to registers before processing [5].
- Processed results must be explicitly stored back to memory [5].

This approach simplifies design by decoupling computation and memory access, leading to regular instruction formats and

simplified control logic [7]. It also guarantees consistent performance across instruction types [8].

### *B. RISC V Architecture*

The implemented processor follows a RISC V architecture a modern instruction set architecture (ISA) which is the 5th generation of the RISC architecture(Reduced Instruction Set Computer). [1]

Key Characteristics of RISC Architectures

- Simplified Instructions: RISC uses a limited set of simple instructions that can be executed very quickly.
- Single Clock Cycle Execution: Most instructions are designed to complete in one processor cycle, increasing overall speed.
- More Registers: RISC processors feature a larger number of general-purpose registers to reduce the need for memory access.
- Simple Decoding: Fewer, simpler instructions lead to simpler instruction decoding and control unit design.

### *C. Security Module (Encryption Module)*

An encryption module between the memory unit and register bank enhances data security during transfers:

- Data written from registers to memory is encrypted when the target address lies between 0 and 127 [9].
- Data read from memory to registers is decrypted when the source address lies between 0 and 127 [9].

Selective encryption based on address range balances performance and security, protecting sensitive memory regions from unauthorized access or tampering [8], [10].

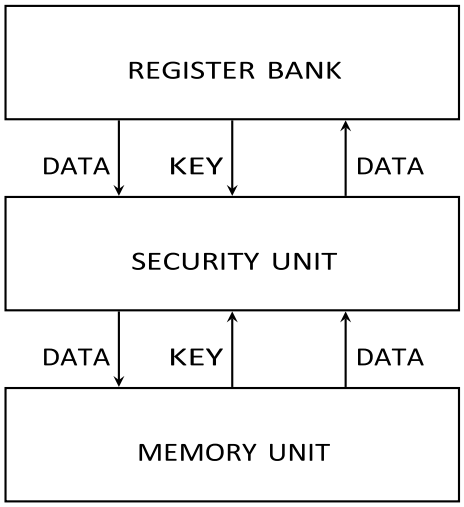


Fig. 1. SECURE DATA TRANSFER ARCHITECTUR

## IV. DETAILED MODULE DESCRIPTION

### *A. top main*

The top main module is the focal point of integration for a digital processing system, integrating the interaction between control logic, memory, registers, security, and arithmetic functions. It takes in a range of inputs, such as clock signals, data, opcodes, and address references, upon which it bases data flow and operation control among its submod- ules. Internally, it creates a control unit to translate opcodes and produce enable signals, a memory module for storing and retrieving data, a register file to handle operands, a security module to check access, and an arithmetic logic unit (ALU) to perform calculations. The module facilitates secure and synchronized data exchange between memory and registers, while also enabling conditional and arithmetic processing through the ALU. By encapsulating these components within a single top-level module, top main provides a scalable and modular framework suitable for implementing instruction-driven digital systems.

Fig. 2 shows the RTL diagram of the top main module:

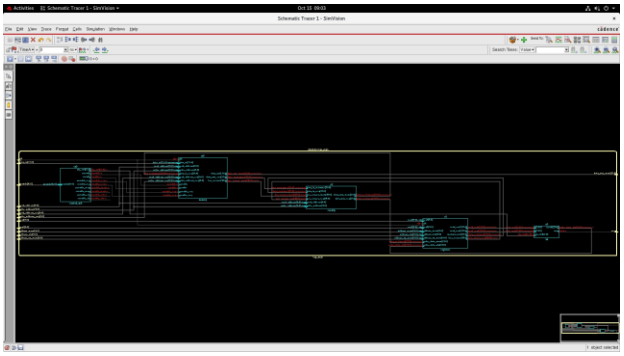

Fig. 2. top main RTL view

### *B. Control Unit*

The control unit serves as the processor's decision-making center, generating all necessary control signals based on the instruction's opcode. The module generates control signals including alu _ctrl, wenable, renable, wenable reg, renable _ reg, wenable mem, renable mem, wenable alu, renable alu. Currently the control unit generates the signals for the following states IDLE, LOAD STORE, MEMORY ACCESS, ADDITION, SUBTRACTION, AND, OR XOR, SLT, SLTU.

Fig. 3 shows the RTL view of the control unit:

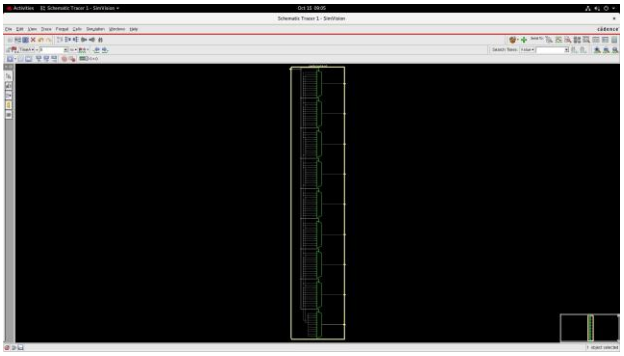

Fig. 3. Control Unit RTL view

*C. ALU Module*

The Arithmetic Logic Unit (ALU) performs all arithmetic and logical operations required by the instruction set [1], [5]. Supported operations include addition and subtraction, bitwise AND, OR, XOR operations, shift operations (left/right, logical/arithmetic), comparison operations (less than, equal),

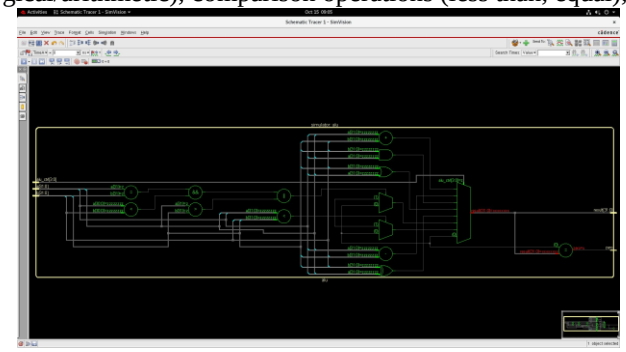

Fig. 4. ALU Module

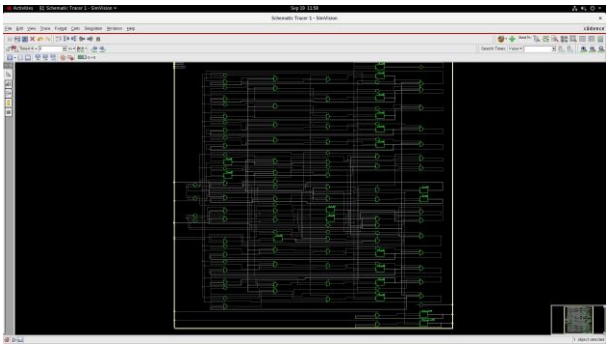

Fig. 5. Memory Module

and zero flag generation for conditional branches [1], [7]. The ALU design emphasizes simplicity and correctness over performance, with straightforward implementation of each operation type [5]. The zero flag output supports conditional instruction set architecture (ISA) branch instructions by indicating when the comparison result is zero [1], [7]. Fig. 4 shows the RTL view of the ALU module:

*D. Memory Module*

This module implements a dual access synchronous memory system with inbuilt access key mechanism [5], [10]. It supports a 1024-word memory array with 32-bit words and two separate read/write channels [5]. It facilitates simultaneous or separated access to the memory, making it more flexible and enhancing digital system throughput [6], [8]. Functional Overview

*Primary access path:* Controlled by data_in, read_address, write_address, wenable, and renable [7].

*Secondary access path:* Uses data_in_reg, read_address_reg, write_address_reg, wenable_reg, and renable_reg [7].

All operations are synchronized to the rising edge of the clock (clk), ensuring consistent timing and predictable behavior in digital systems [5], [7].

Fig. 5 shows the RTL view of the Memory Module:

Fig. 6. Registers module RTL view

*E. Registers module*

The registers module uses a synchronous memory access system that is built to sustain multiple read and write operations in one clock cycle [5], [7]. It has a 1024-word memory array, with each word consisting of 32 bits, and offers structured memory access to computation as well as data transfer operations [5]. Functional Overview

*Write operations:* There are two independent inputs (write data _mem and write data _alu) through which data is written into memory at locations address mem and address alu, respectively [7].

*Read operations:* Two registers (reg1 and reg2) are used to fetch data from memory and output it through read_reg1 and read_reg2 [7].

*Memory forwarding:* The value at address_to_mem is read and sent to memory_out, which can be used for chaining memory operations or interfacing with other modules [6], [8]. All operations are synchronized to the rising edge of the clock (clk), ensuring consistent timing and predictable behavior in digital systems [5], [7].

Fig. 6 shows the RTL view of the Registers Module:

*F. Security Unit*

The security module introduces a conditional data transformation mechanism based on access verification and address thresholds [9], [10]. It is designed to enhance data integrity and control within a memory system by applying mathematical operations only when specific access conditions are met [10]. Functional Overview

This module processes two input data streams—data_in_memory and data_in_reg—and generates two corresponding outputs—data_out_mem and data_out_reg [9]. The behavior of each output is governed by:

- A 16-bit access key (key_access_mem and key_access_reg) [10]
- A 10-bit address threshold (write_address and read_address) [9]

When the access key matches the predefined value (0x0032), the module performs conditional logic based on the address value:

- If write_address > 128, data_out_mem is computed using a quadratic transformation of data_in_reg [9].
- Otherwise, data_out_mem directly reflects data_in_reg [9].
- If read_address> 128, data_out_reg is derived from a nonlinear transformation of data_in_memory, followed by bitwise negation [9].
- Otherwise, data_out_reg simply mirrors data_in_memory [9].

Fig. 7 shows the RTL view of the Security Module:

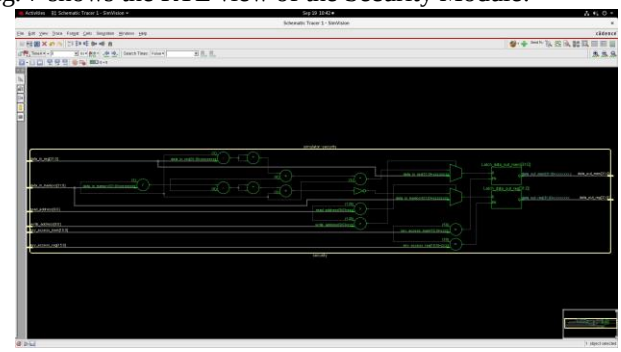

Fig. 7. Security Unit

## V. OPCODE TO OPERATION MAPPING

| opcode | operation |
|---|---|
| 5'b00000 and above 5'b01001 | IDLE |
| 5'b00001 | LOAD STORE |
| 5'b00010 | memory access |
| 5'b00011 | ADD |
| 5'b00100 | SUB |
| 5'b00101 | AND |
| 5'b00110 | OR |
| 5'b00111 | XOR |
| 5'b01000 | SLT |
| 5'b01001 | SLTU |

TABLE I
OPCODE TO OPERATION MAPPING

Table II specifies the mapping of the 5- bit instruction codes to their respective operations in the control logic of the processor. Each opcode is a unique identifier for a certain functional behavior like arithmetic (ADD, SUB), logical (AND, OR, XOR), comparison (SLT, SLTU), or memory operations (LOAD STORE, memory access). Opcodes outside of the specified operational range are reserved for the IDLE state to safely handle undefined or invalid instructions. This mapping makes it easier to de- coding of the control unit and facilitates effective operation execution on the basis of space-efficient binary encodings.

## VI. TEST PLAN

| Module | Feature | Verification Strategy |
|---|---|---|
| ALU | 1) Perform a selected logic on the given input | Check every single logic on the data and check its output in the scoreboard |
| | 2) Zero Flag | Check if zero flag goes up when result is zero |
| | 3) Default result | Check when Sel ≥ 4'b0111 the Result is zero |
| Security | Key access control | Check if the security block allows the data transfer when the keys are right |
| | Simultaneous encryption and decryption | Check if the data is encrypted and decrypted if addresses are less than 128 |
| Memory | Store data in memory | Check if the data is correctly stored in that address |
| | Simultaneous read and write to main memory and register file | Check if the data is stored and read simultaneously by main memory and register file |
| Register Bank | Store Data in Register Bank | Check if data is stored in the reg bank correctly at the correct address |
| | Simultaneous read and write to memory and ALU | Check if the data is stored and read simultaneously by memory and ALU |
| Control Unit | Giving the control signals for different opcodes | Check the output for every single opcode |
| Top Main | Perform simple RV32I instruction set | Check the output for every instruction |

TABLE II
VERIFICATION PLAN

This verification plan outlines how each hardware module in the design will be tested to ensure correct functionality.

- ALU: Check every single logic on the data and check its output and zero flag

- Security Module: Test correct key-based access control and ensure encryption/decryption work properly for valid address ranges.
- Memory: Check that data is stored correctly, and verify simultaneous read/write operations don't cause errors.
- Register Bank: Validate correct data storage and retrieval, and confirm proper simultaneous interactions with memory and the ALU.
- Control Unit: Test control signal generation for all opcodes.
- Top Main (Integration): Verify full instruction execution for the RV32I instruction set and confirm correct overall system behavior.

## VII. VERIFICATION METHODOLOGY

The verification of the RISC-V processor was done with a layered SystemVerilog testbench designed to test all functional blocks in a hierarchical mode. The testbench architecture includes the generator, driver, monitor, inter- face, and scoreboard components and offers a hierarchical and reusable verification environment. Every module of the processor — ALU, Security Unit, Memory, Register Bank, Control Unit, and Top Main — were individually verified based on the given Verification Plan (Table II). Random and directed test cases were utilized to verify correctness, functionality, and signal-level integrity. The generator generated random and bounded input stimulus for various types of instructions. The inputs were driven by the driver to the DUT via a virtual interface with proper synchronization with the system clock. The monitor observed and tracked signal activity and passed it on to the scoreboard, which checked the stored outputs against simulated ones for validation. The simulations were run under Cadence SimVision, with a 10-unit clock and with numerous randomized trans- actions. The verification phase considered first-order functional correctness and inter-module synchrony and produced a compre- hensive verification of the processor's secure data flow and behavior.

## VIII. VERIFICATION SIMULATION OUTPUT

The verification of the design was carried out using a layered testbench approach. The simulation results were carefully monitored, and all functional modules such as the ALU, memory, register bank, and control unit performed as expected according to the test vectors.

The output data from the simulation confirms that the design operates correctly under the specified conditions. Therefore, the verification process was successfully completed.

## IX. CONCLUSION

This paper presents the design and implementation of a secured RISC-V processor that executes the RV32I base instruction set. By adding a dedicated encryption module between the register bank and memory, the processor guarantees secure data transfer for critical memory areas. It maintains the efficiency and simplicity of a load-store architecture. The layered SystemVerilog verification technique verified that all functional units function as expected. These are the ALU, memory, register bank, control unit, and security unit. The findings indicate that selective encryption using address range attacks achieves an effective tradeoff between security and performance. In summary, the outlined architecture offers a scalable and modular platform for secure processor design. It can be extended to incorporate more advanced cryptographic functions or larger RISC-V instruction sets in future studies.

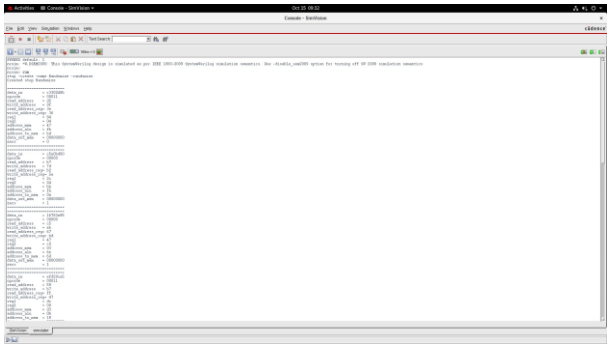

Fig. 8. Simulation output showing successful verification of the design.

## REFERENCES


[1] A. Waterman and K. Asanovic, "The RISC-V Instruction Set Manual´ Volume I: User-Level ISA," RISC-V Foundation, 2019.

[2] S. Yang, L. Shao, J. Huang, and W. Zou, "Design and Implementation of Low-Power IoT RISC-V Processor with Hybrid Encryption Accelerator," *Electronics*, 2023.

[3] Z. Azad, G. Yang, R. Agrawal, et al., "RISE: RISC-V SoC for En/Decryption Acceleration on the Edge for Homomorphic Encryption," arXiv preprint, 2023.

[4] A. Zgheib, O. Potin, J.-B. Rigaud, and J.-M. Dutertre, "Extending a RISC-V Core with an AES Hardware Accelerator to Meet IoT Constraints," SMACD/PRIME.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed., Morgan Kaufmann, 2019.

[6] J. E. Smith, "Decoupled Access/Execute Computer Architectures," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 289–308, 1982.

[7] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed., Morgan Kaufmann, 2013.

[8] G. Z. Chrysos and J. Emer, "Memory Dependence Prediction Using Store Sets," *Proc. 25th Ann. Int. Symp. Microarchitecture*, pp. 142–153, 1993.

[9] D. Kaplan, J. Powell, and T. Woller, "AMD Memory Encryption White Paper," AMD, 2016.

[10] V. Nagarajan, R. Gupta, and A. Krishnaswamy, “Compiler-Assisted Memory Encryption for Embedded Processors,” Proc. 2009 Embedded Systems Conference, 2009.