MỤC LỤC

DANH MỤC BẢNG

DANH MỤC HÌNH VẼ

PHẦN MỞ ĐẦU

Trong thời đại công nghệ số hiện nay, nhu cầu bảo vệ thông tin cá nhân, dữ liệu nhạy cảm và truyền thông an toàn ngày càng trở nên cấp thiết. Đặc biệt, với sự bùng nổ của các thiết bị kết nối trong mạng lưới Internet vạn vật (IoT), các hệ thống nhúng, cũng như điện toán đám mây, việc đảm bảo an toàn dữ liệu trong môi trường có khả năng bị tấn công mạng là vô cùng quan trọng. Trong bối cảnh đó, các thuật toán mã hóa đóng vai trò then chốt trong việc đảm bảo tính bảo mật, toàn vẹn và riêng tư của dữ liệu.

Một trong những thuật toán mã hóa được sử dụng phổ biến và đáng tin cậy nhất hiện nay là Advanced Encryption Standard (AES) – hay còn gọi là Chuẩn mã hóa tiên tiến. AES là một thuật toán mã hóa đối xứng do Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST) ban hành và được xem là tiêu chuẩn chính thức để bảo mật thông tin điện tử trong nhiều lĩnh vực từ quốc phòng, tài chính, đến thương mại điện tử.

AES hoạt động với nhiều độ dài khóa khác nhau: 128-bit, 192-bit và 256-bit, và hỗ trợ nhiều chế độ mã hóa như ECB, CBC, CTR, GCM,... giúp linh hoạt trong các ứng dụng cụ thể. Tuy nhiên, do tính chất phức tạp của thuật toán AES – bao gồm các phép toán tuyến tính và phi tuyến trong không gian trường hữu hạn – việc triển khai AES sao cho đạt hiệu suất cao, đặc biệt là trong môi trường có tài nguyên hạn chế như thiết bị nhúng và IoT, là một thách thức không nhỏ. Tuy nhiên, các bộ tăng tốc phần cứng AES hiện tại vẫn gặp phải nhiều rào cản như:

- Khó tối ưu hóa cho nhiều chế độ và độ dài khóa khác nhau, độ linh hoạt thấp.
- Chiếm nhiều tài nguyên phần cứng, ảnh hưởng đến việc tích hợp vào hệ thống nhúng nhỏ gọn;
- Hiệu suất không ổn định khi xử lý dữ liệu thời gian thực có tính biến đổi cao.
- Để giải quyết các hạn chế trên, đề tài này đề xuất một kiến trúc tăng tốc phần cứng AES mới tích hợp vào bộ xử lý RISC-V – gọi là AES-RV. Kiến trúc này nhằm:
- Tối ưu hóa băng thông và tốc độ xử lý dữ liệu đầu vào/đầu ra bằng cách sử dụng bộ đệm nội cao băng thông;
- Giảm độ trễ trong thực thi các phép toán AES thông qua tập lệnh tùy chỉnh riêng
- Tăng tốc quá trình xử lý dữ liệu tuần tự nhờ kỹ thuật pipeline và truyền dữ liệu kiểu ping-pong.

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 Tổng quan về RISC-V

1.1.1 Giới thiệu chung về RISC-V

RISC-V là một kiến trúc tập lệnh (Instruction Set Architecture – ISA) dạng mở và miễn phí bản quyền, được phát triển bởi nhóm nghiên cứu tại Đại học California, Berkeley, từ năm 2010. Khác với các ISA thương mại như x86 (Intel, AMD) hay ARM (ARM Holdings), RISC-V được thiết kế với mục tiêu đơn giản, linh hoạt, dễ triển khai, và đặc biệt là hoàn toàn mở mã nguồn – cho phép bất kỳ cá nhân hoặc tổ chức nào cũng có thể sử dụng, sửa đổi, và triển khai mà không tốn chi phí bản quyền.

ISA là tập hợp các lệnh cơ bản mà một bộ xử lý có thể hiểu và thực hiện. Một ISA hiệu quả sẽ đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất, tiêu thụ năng lượng và khả năng mở rộng của hệ thống.

RISC-V thuộc họ kiến trúc RISC (Reduced Instruction Set Computing) – tức là kiến trúc máy tính sử dụng tập lệnh đơn giản, giới hạn về số lượng nhưng có tốc độ thực thi cao. RISC-V kế thừa những ưu điểm của các kiến trúc RISC trước đó và đồng thời được thiết kế để phù hợp với nhu cầu hiện đại như hệ thống nhúng, thiết bị IoT, máy chủ, thiết bị di động, và cả các hệ thống hiệu năng cao.

1.1.2. Các đặc điểm nổi bật của RISC-V

Về mặt kiến trúc, RISC-V được thiết kế theo hướng đơn giản hóa và mô-đun hóa, cho phép dễ dàng cấu hình và mở rộng tùy theo yêu cầu ứng dụng. Cụ thể, RISC-V bao gồm một tập lệnh cơ sở (Base ISA) tối giản, cùng với nhiều tập lệnh mở rộng (Extension ISA) có thể lựa chọn theo nhu cầu như:

- **M (Multiply/Divide):** hỗ trợ các phép nhân chia số nguyên
- **A (Atomic):** cung cấp thao tác nguyên tử để đồng bộ dữ liệu trong hệ đa xử lý
- **F và D (Floating-point):** hỗ trợ số thực dấu chấm động đơn và kép theo chuẩn IEEE-754
- **C (Compressed):** cung cấp mã lệnh rút gọn giúp giảm dung lượng bộ nhớ chương trình
- **V (Vector):** hỗ trợ xử lý vector, đặc biệt hữu ích trong các ứng dụng trí tuệ nhân tạo và tính toán song song hiệu năng cao.

Nhờ đó, RISC-V có thể được tùy biến để đáp ứng nhiều mục tiêu thiết kế, từ các vi điều khiển công suất siêu thấp trong thiết bị IoT cho đến các bộ vi xử lý phức tạp phục vụ hệ thống máy chủ hoặc xử lý tín hiệu số chuyên sâu.

Ngoài ra, RISC-V sở hữu khả năng mở rộng cao, cho phép cấu hình với độ dài thanh ghi 32-bit, 64-bit, hoặc 128-bit, phù hợp với đa dạng các lớp thiết bị – từ nhúng công suất thấp đến hệ thống hiệu năng cao.

Với tập lệnh đơn giản, nhất quán và được tối ưu hóa về hiệu suất, kiến trúc RISC-V có thể đạt hiệu suất xử lý cao trong khi vẫn duy trì mức tiêu thụ năng lượng thấp, một yếu tố then chốt đối với các hệ thống nhúng, thiết bị di động và các ứng dụng IoT.

1.1.3. Cấu trúc mã lệnh (Instruction Encoding) của RISC-V

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Bảng 1. *Bảng định dạng lệnh (instruction formats) trong kiến trúc RISC-V*

Mỗi lệnh trong RISC-V đều dài 32 bit. Các bit được đánh số từ 31 (trái) đến 0 (phải). Tùy theo loại lệnh, các bit này được chia thành các trường (field) khác nhau:

- **Cấu trúc tổng quát của một câu lệnh (instruction)**

opcode	7-bit (luôn nằm ở bit 0–6) – xác định loại lệnh.
rd	5-bit – thanh ghi đích (destination register).
rs1, rs2	thanh ghi nguồn (source registers).
funct3, funct7	xác định chức năng chi tiết hơn của lệnh (phân biệt các lệnh trong cùng một nhóm opcode).
imm	giá trị hằng số tức thời (immediate).

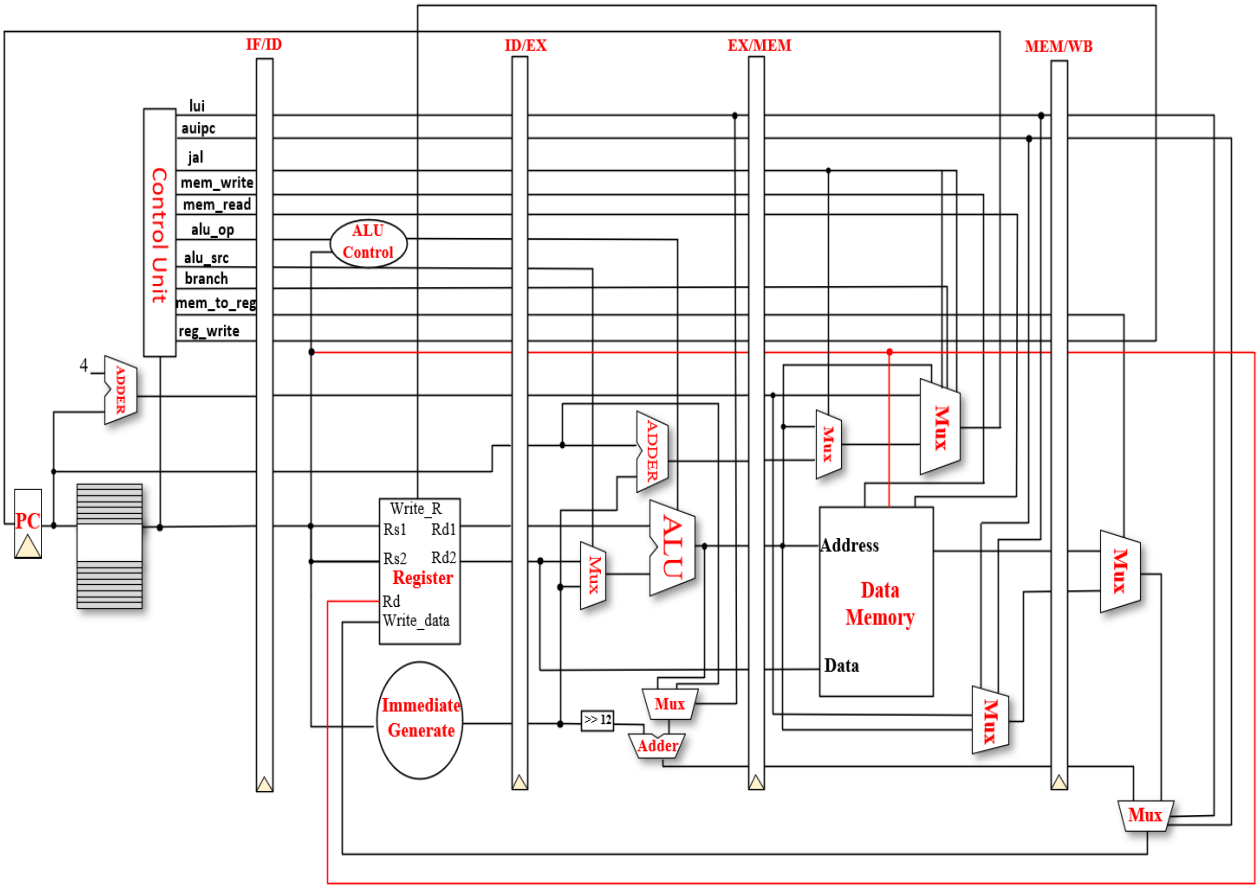
Bảng 2: *Cấu trúc và chức năng các trường trong mã lệnh RISC-V*

- **Các định dạng tệp lệnh RISC-V**

Loại	Ý nghĩa
R-type	Các lệnh tính toán giữa 2 thanh ghi (vd: **add**, **sub**, **and**, **or**)
I-type	Lệnh tính toán với hằng số, load, jump (**addi**, **lw**, **jalr**, **ecall**)
S-type	Lệnh lưu vào bộ nhớ (store: **sw**, **sb**, **sh**)
B-type	Lệnh rẽ nhánh có điều kiện (**beq**, **bne**, **blt**, **bge**)
U-type	Lệnh nạp địa chỉ hằng (lớn) (**lui**, **auipc**)
J-type	Lệnh nhảy không điều kiện (**jal**)

Bảng 3. *Phân loại định dạng và chức năng các lệnh trong tập lệnh RISC-V*

1.1.4 Sơ đồ kiến trúc RISC_V



Hình 9*. Sơ đồ khối mô tả kiến trúc cơ bản RISCV_32I*

Trong sơ đồ kiến trúc tổng quát RV_32I bao gồm các khối chính có chức năng cơ bản như sau:

Tên module	Chức năng
PC	Lưu trữ địa chỉ của lệnh hiện tại cần thực thi. Sau mỗi chu kỳ thực thi, PC sẽ tăng lên để trở đến lệnh tiếp theo.
Instruction Memory	Lưu trữ các lệnh chương trình đang thực thi.
Control Unit	Điều khiển hoạt động của các khối khác, giải mã lệnh và phát tín hiệu điều khiển.
ALU Control	Giải mã tín hiệu điều khiển từ Control Unit để quyết định phép toán mà ALU thực hiện.
Register File	Lưu trữ các giá trị dữ liệu tạm thời trong 32 thanh ghi 32-bit.
Immediate Generate	Tạo ra các giá trị immediate từ các lệnh dạng I, S, B, U, J để sử dụng trong ALU hoặc điều khiển nhánh.
ALU	Thực hiện các phép toán số học (cộng, trừ, nhân) và logic (AND, OR …) Các lệnh xử lý rẽ nhánh và nhảy thay đổi giá trị PC đã được tích hợp trong khối ALU kết hợp với các Mux điều khiển
Data Memory	Lưu trữ dữ liệu mà chương trình cần thao tác.

Bảng 4*. Bảng tóm tắt chức năng các khối chính của RV_32I*

Trong kiến trúc RV32I, các khối chức năng được tổ chức và phối hợp một cách hợp lý để thực hiện lệnh theo mô hình pipeline 5 tầng (5-stage pipeline), nhằm tối ưu hóa hiệu suất xử lý và đảm bảo thực thi liên tục theo từng chu kỳ xung nhịp. Năm tầng chính trong pipeline bao gồm: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), và Write Back (WB). Giữa các tầng là các thanh ghi pipeline trung gian, cho phép truyền dữ liệu một cách tuần tự và chính xác từ tầng này sang tầng kế tiếp.

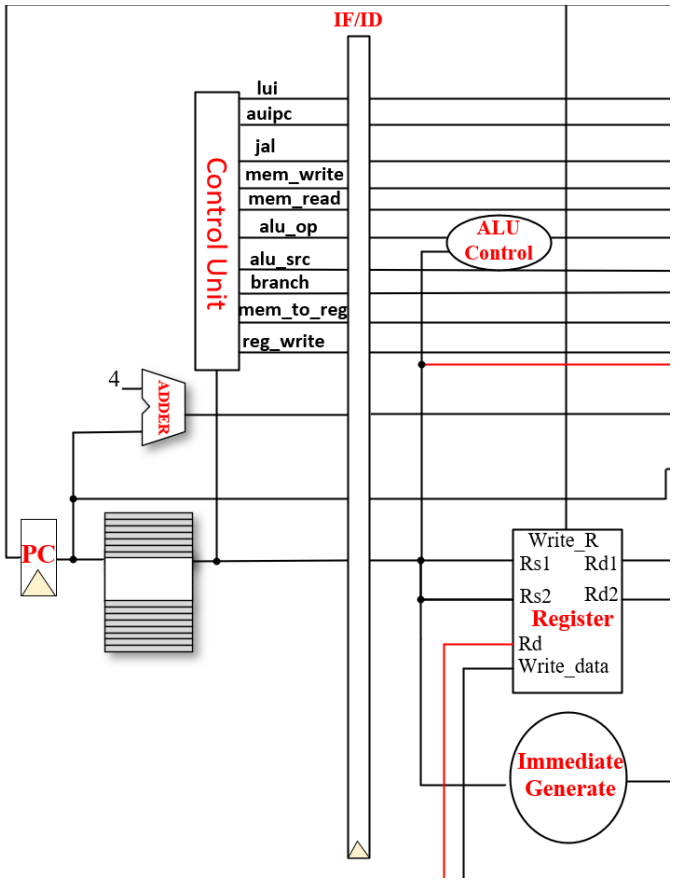
- **Tầng IF – Instruction Fetch:** Tầng này chịu trách nhiệm nạp lệnh từ bộ nhớ chương trình (Instruction Memory) dựa trên giá trị hiện tại của Program Counter (PC). Lệnh nạp vào cùng với địa chỉ PC được lưu trong thanh ghi pipeline IF/ID, để chuyển tiếp đến tầng giải mã.

- **Tầng ID – Instruction Decode**: Tại đây, lệnh được giải mã để xác định loại thao tác (tính toán số học, logic, load/store, hoặc rẽ nhánh). Đồng thời, các thanh ghi nguồn (rs1, rs2) được đọc từ Register File, và giá trị immediate (nếu có) được trích xuất. Thông tin giải mã được lưu trong thanh ghi ID/EX để chuyển đến tầng thực thi.
- **Tầng EX – Execute**: Trong tầng này, đơn vị tính toán (ALU) thực hiện các phép toán số học, logic, tính toán địa chỉ bộ nhớ hoặc kiểm tra điều kiện nhảy. Kết quả từ ALU, địa chỉ truy cập bộ nhớ (nếu cần), và dữ liệu cần ghi (trong lệnh store) được ghi vào thanh ghi EX/MEM để truyền đến tầng kế tiếp.
- **Tầng MEM – Memory Access**: Tầng này thực hiện thao tác truy xuất bộ nhớ. Nếu là lệnh load, dữ liệu sẽ được đọc từ Data Memory; nếu là lệnh store, dữ liệu được ghi vào bộ nhớ. Đối với các lệnh không liên quan đến bộ nhớ, tầng này được bỏ qua về mặt chức năng, nhưng vẫn duy trì tính tuần tự trong pipeline. Dữ liệu đầu ra hoặc kết quả tính toán từ ALU được chuyển vào thanh ghi MEM/WB**.**
- **Tầng WB – Write Back**: Đây là tầng cuối cùng, nơi kết quả của lệnh (từ ALU hoặc dữ liệu đọc từ bộ nhớ) được ghi ngược trở lại vào Register File. Việc ghi kết quả này đảm bảo dữ liệu sẵn sàng cho các lệnh kế tiếp sử dụng.

Kiến trúc pipeline 5 tầng của RV32I cho phép các lệnh được xử lý song song ở các giai đoạn khác nhau trong cùng một thời điểm. Mỗi chu kỳ đồng hồ, một lệnh mới được nạp vào pipeline, trong khi các lệnh trước đó tiếp tục tiến hành ở các tầng kế tiếp. Điều này giúp tăng hiệu suất thực thi, rút ngắn độ trễ trung bình trên mỗi lệnh và đảm bảo khả năng xử lý lệnh liên tục, thay vì phải chờ từng lệnh hoàn tất trước khi bắt đầu lệnh kế tiếp. Sau đây sẽ trình bày chi tiết cấu trúc của từng tầng thực thi:

a) **Tầng IF và ID**

Tầng IF/ID là giai đoạn mở đầu trong pipeline 5 tầng của kiến trúc RV32I, chịu trách nhiệm nạp lệnh (Instruction Fetch) từ bộ nhớ chương trình và giải mã lệnh (Instruction Decode) để trích xuất thông tin điều khiển và các toán hạng cần thiết phục vụ cho các tầng xử lý kế tiếp.



***Hình 10**. Sơ đồ mô tả kiến trúc các khối trong tầng IF và ID của RISC-V*

Việc tổ chức tầng này thành các khối chức năng rõ ràng cùng hệ thống thanh ghi pipeline trung gian cho phép đảm bảo luồng dữ liệu nhất quán và tối ưu hóa hiệu suất thực thi theo hướng song song.

Các khối chức năng chính trong tầng IF/ID bao gồm:

1. **PC (Program Counter)**

PC giữ địa chỉ của lệnh hiện tại đang được thực thi. Sau mỗi chu kỳ xung nhịp, địa chỉ này cập nhật bằng cách cộng thêm 4 (tương ứng với độ dài mỗi lệnh 32-bit) thông qua khối **Adder**, nhằm trỏ đến địa chỉ của lệnh kế tiếp trong **Instruction Memory**.

2. **Instruction Memory**

Đây là khối bộ nhớ chỉ đọc (ROM) dùng để lưu trữ toàn bộ chương trình. Instruction Memory nhận địa chỉ từ PC và trả về lệnh tương ứng để đưa vào pipeline. Lệnh này sau đó sẽ được truyền đến Control Unit để giải mã.

3. **Control Unit**

Sau khi lệnh được nạp, Control Unit sẽ giải mã opcode để xác định loại lệnh (load, store, toán học, nhánh...). Dựa trên kết quả giải mã, Control Unit tạo ra các tín hiệu điều khiển như alu_op, mem_read, mem_write, reg_write, alu_src, branch,... nhằm điều phối hoạt động của các khối trong pipeline.

4. **Adder**

Khối Adder tính toán địa chỉ tiếp theo cho PC bằng cách thực hiện phép cộng PC + 4. Đây là bước cần thiết để CPU tuần tự nạp lệnh mới trong chương trình.

5. **Register File**

Đây là tập hợp các thanh ghi tổng quát, dùng để lưu trữ các toán hạng và kết quả trung gian. Trong giai đoạn giải mã, hai thanh ghi nguồn rs1 và rs2 sẽ được đọc từ Register File dựa trên trường chỉ định trong lệnh.

6. **Immediate Generator**

Đối với các lệnh có chứa giá trị hằng số (immediate) như load, store, hay branch, khối Immediate Generator sẽ trích xuất và tính toán giá trị này từ các bit tương ứng trong lệnh đã giải mã.

7. **ALU Control**

ALU Control là khối trung gian giữa Control Unit và ALU, nhận tín hiệu điều khiển và thông tin từ các trường funct của lệnh để xác định **phép toán cụ thể** mà ALU cần thực hiện (như cộng, trừ, so sánh, dịch bit...).

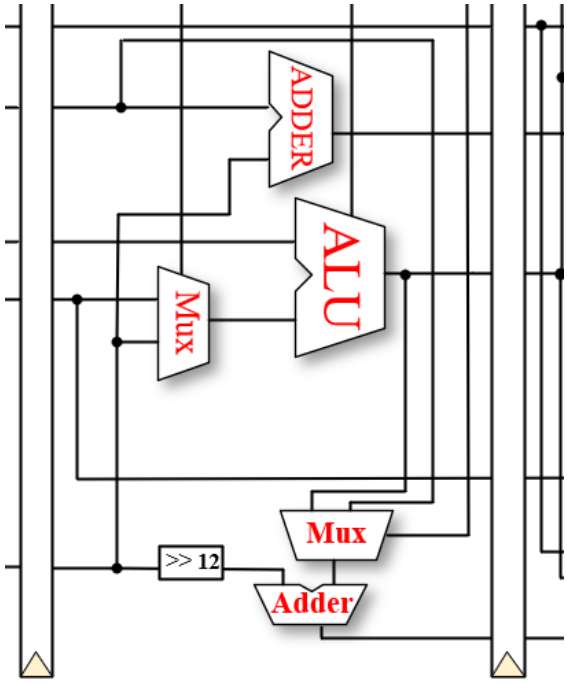
Ngoài các khối chức năng trên, tầng IF/ID còn bao gồm hai thanh ghi pipeline quan trọng:

- **IF/ID Register**: Lưu trữ lệnh hiện tại được nạp từ Instruction Memory cùng với giá trị PC tương ứng. Điều này đảm bảo thông tin được giữ ổn định khi truyền sang tầng ID.
- **ID/EX Register**: Lưu giữ các giá trị toán hạng (rs1, rs2), giá trị immediate, thông tin điều khiển từ Control Unit và các trường quan trọng của lệnh để chuẩn bị cho giai đoạn **thực thi (Execute)** ở tầng EX.

Tổ chức logic của tầng IF/ID đóng vai trò nền tảng cho hoạt động của pipeline, không chỉ đảm bảo **luồng lệnh chính xác và ổn định** mà còn cho phép hệ thống **xử lý nhiều lệnh song song** trong các giai đoạn khác nhau. Điều này góp phần quan

trọng vào việc **nâng cao hiệu suất tổng thể của CPU**, đặc biệt trong các kiến trúc RISC đơn giản và hiệu quả như RV32I.

b) Tầng thực thi EX



***Hình 11**. Sơ đồ mô tả kiến trúc các khối trong tầng EX của RISC-V*

Tầng **EX (Execute)** đóng vai trò là **giai đoạn thực thi trung tâm trong pipeline 5 tầng của kiến trúc RV32I**, nơi diễn ra các phép toán số học, logic và tính toán địa chỉ. Sau khi lệnh được giải mã và chuẩn bị ở tầng ID, các thông tin cần thiết như toán hạng, giá trị immediate, và tín hiệu điều khiển được chuyển đến tầng EX thông qua thanh ghi pipeline **ID/EX**. Tại đây, các phép toán cốt lõi sẽ được thực hiện để tạo ra kết quả đầu ra của lệnh, phục vụ cho các giai đoạn tiếp theo như truy cập bộ nhớ hoặc ghi kết quả vào thanh ghi.

Tầng EX bao gồm các thành phần chính như sau:

- **ALU (Arithmetic Logic Unit)**: Đây là khối chức năng trung tâm của tầng EX, chịu trách nhiệm thực hiện các phép toán số học (cộng, trừ, nhân, dịch bit...) và logic (AND, OR, XOR...). ALU nhận hai đầu vào – có thể là hai thanh ghi hoặc một thanh ghi kết hợp với một giá trị immediate – tùy theo loại lệnh. Các tín hiệu điều khiển từ khối ALU Control sẽ xác định phép toán cụ thể cần thực hiện. Kết

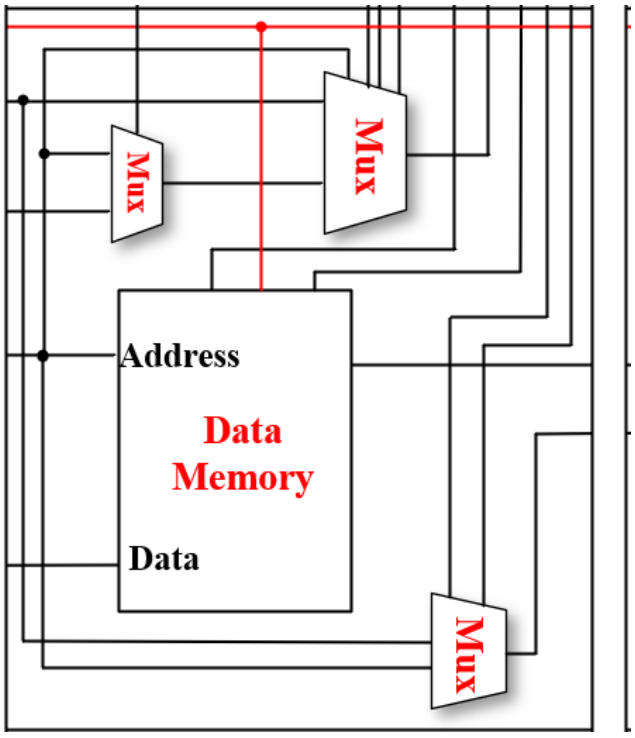
quả tính toán của ALU sẽ được truyền đến tầng tiếp theo thông qua thanh ghi pipeline EX/MEM.

- **MUX (Multiplexer)**: Các bộ chọn (MUX) được sử dụng để xác định nguồn dữ liệu đầu vào cho ALU. Dữ liệu đầu vào thứ nhất thường được lấy từ thanh ghi rs1, trong khi đầu vào thứ hai có thể được chọn giữa giá trị từ rs2 hoặc immediate, tùy thuộc vào tín hiệu điều khiển ALUSrc từ Control Unit. Sử dụng MUX giúp tăng tính linh hoạt, xử lý nhiều loại lệnh khác nhau.
- **Adder**: Trong trường hợp lệnh nhánh, một khối cộng riêng được sử dụng để tính toán địa chỉ nhánh mục tiêu, bằng cách cộng giá trị của PC với giá trị offset (thường là immediate được dịch trái). Kết quả từ Adder này sẽ được sử dụng ở tầng sau để quyết định liệu nhánh có xảy ra hay không.
- **ID/EX Register**: Đây là thanh ghi pipeline trung gian giữa tầng ID và EX. Nó lưu trữ toàn bộ thông tin cần thiết đã được giải mã từ lệnh, bao gồm giá trị toán hạng từ thanh ghi rd1, rd2, giá trị immediate, các tín hiệu điều khiển và thông tin opcode/funct. Điều này đảm bảo dữ liệu được truyền đồng bộ vào ALU trong chu kỳ xử lý hiện tại.
- **EX/MEM Register**: Sau khi ALU hoàn tất tính toán, kết quả được lưu vào thanh ghi pipeline EX/MEM, cùng với các thông tin điều khiển liên quan đến thao tác bộ nhớ (như MemRead, MemWrite, RegWrite) và địa chỉ thanh ghi đích (rd). Đây là bước chuẩn bị để truyền thông tin đến tầng truy cập bộ nhớ (MEM).

Về chức năng tổng thể, tầng EX chịu trách nhiệm thực hiện **các phép toán cốt lõi của hệ thống**, đồng thời tính toán địa chỉ truy cập bộ nhớ cho lệnh load/store và địa chỉ nhánh trong các lệnh điều khiển luồng. Kết quả của các phép toán trong tầng này sẽ quyết định nội dung xử lý tại tầng MEM và WB, do đó EX được xem là **tầng quan trọng nhất trong pipeline**, quyết định trực tiếp đến độ chính xác và hiệu suất của quá trình thực thi lệnh.

Việc tổ chức tầng EX thành các khối chức năng riêng biệt và sử dụng thanh ghi pipeline để lưu trữ trung gian cho phép các lệnh được **thực thi song song** tại các tầng khác nhau trong pipeline. Điều này góp phần tối ưu hóa hiệu suất xử lý, đồng thời đảm bảo tính nhất quán và độ ổn định trong hệ thống xử lý lệnh của kiến trúc RISC-V.

c) Tầng MEM



***Hình 12**. Sơ đồ mô tả kiến trúc các khối trong tầng MEM của RISC-V*

Tầng MEM (Memory Access) là giai đoạn thứ tư trong pipeline 5 tầng của kiến trúc RV32I, đảm nhiệm vai trò xử lý các thao tác truy xuất bộ nhớ. Cụ thể, tầng này thực hiện đọc dữ liệu từ bộ nhớ đối với các lệnh load, hoặc ghi dữ liệu vào bộ nhớ trong các lệnh store. Đây là tầng trung gian, kết nối trực tiếp giữa giai đoạn tính toán tại tầng EX và giai đoạn ghi kết quả tại tầng WB.

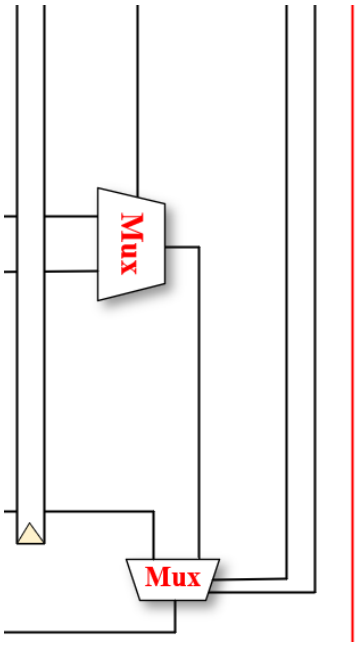
Tầng MEM bao gồm các thành phần chính sau:

- **Data Memory**: Là khối bộ nhớ dữ liệu chính của hệ thống, nơi lưu trữ và cung cấp dữ liệu phục vụ cho các lệnh truy xuất bộ nhớ. Khi lệnh là load, Data Memory sẽ đọc dữ liệu tại địa chỉ do ALU tính toán và gửi giá trị về hệ thống. Ngược lại, nếu là lệnh store, dữ liệu từ thanh ghi nguồn sẽ được ghi vào địa chỉ chỉ định trong bộ nhớ.
- **MUX (Multiplexer)**: Là bộ chọn tín hiệu, quyết định dữ liệu nào sẽ được đưa đến thanh ghi đích trong tầng WB. MUX lựa chọn giữa kết quả từ ALU (đối với các lệnh tính toán) hoặc dữ liệu đọc từ bộ nhớ (đối với lệnh load), tùy theo tín hiệu điều khiển MemToReg được truyền từ tầng trước.

- **EX/MEM Register**: Là thanh ghi pipeline trung gian giữa tầng EX và MEM, giữ ổn định toàn bộ thông tin cần thiết cho truy xuất bộ nhớ như địa chỉ, dữ liệu cần ghi (nếu có), kết quả từ ALU và các tín hiệu điều khiển (MemRead, MemWrite). Nhờ đó, thông tin có thể được truyền đúng và kịp thời đến Data Memory hoặc tầng tiếp theo.
- **MEM/WB Register**: Là thanh ghi pipeline nằm giữa tầng MEM và WB, lưu trữ kết quả từ bộ nhớ (nếu là load) hoặc kết quả tính toán từ ALU, chuẩn bị để ghi vào **Register File** tại tầng WB.

Về tổng thể, tầng MEM đảm bảo các thao tác truy xuất dữ liệu được thực hiện chính xác và kịp thời, đặc biệt với các lệnh lw, sw, lb, sh,... Đối với các lệnh không liên quan đến bộ nhớ (như add, sub, and,...), tầng này đóng vai trò truyền tiếp dữ liệu mà không thực hiện thao tác truy xuất, giúp giữ cho pipeline hoạt động trơn tru. Sự kết hợp giữa các khối chức năng và các thanh ghi pipeline như EX/MEM và MEM/WB không chỉ giúp giữ ổn định dữ liệu trong quá trình thực thi, mà còn đảm bảo khả năng xử lý song song và liên tục giữa các lệnh, từ đó tăng thông lượng và hiệu năng tổng thể của hệ thống vi xử lý dựa trên kiến trúc RISC-V.

d) Tầng WB



Hình 13. *Sơ đồ mô tả kiến trúc các khối trong tầng WB của RISC-V*

Tầng WB (Write Back) là giai đoạn cuối cùng trong pipeline 5 tầng của kiến trúc RV32I, đảm nhiệm vai trò ghi kết quả xử lý trở lại vào Register File, nơi lưu

trữ các giá trị toán hạng phục vụ cho các lệnh kế tiếp trong chương trình. Kết quả ghi vào có thể đến từ hai nguồn chính: bộ nhớ dữ liệu (Data Memory) trong các lệnh load, hoặc kết quả tính toán từ ALU trong các lệnh số học và logic.

Tầng WB bao gồm các thành phần chính sau:

- **MUX (Multiplexer)**: Là bộ chọn dữ liệu đầu vào cho quá trình ghi vào thanh ghi đích. MUX quyết định nguồn dữ liệu được ghi – chọn giữa kết quả từ Data Memory (đối với lệnh load) hoặc từ ALU (đối với lệnh tính toán). Tín hiệu điều khiển MemToReg từ các tầng trước quyết định việc lựa chọn này. Dữ liệu được chọn sẽ được chuyển đến Register File.
- **MEM/WB Register**: Là thanh ghi pipeline trung gian giữa tầng MEM và WB, giữ ổn định dữ liệu đầu ra từ tầng MEM trong suốt chu kỳ xử lý. Dữ liệu này bao gồm giá trị đọc từ bộ nhớ hoặc kết quả từ ALU, cùng với các tín hiệu điều khiển liên quan. Thanh ghi này đảm bảo dữ liệu được truyền chính xác đến tầng WB mà không bị sai lệch.
- **Register File**: Đây là tập hợp các thanh ghi chung (general-purpose registers) của bộ xử lý, nơi lưu giữ các kết quả tính toán hoặc dữ liệu được load từ bộ nhớ. Trong giai đoạn WB, dữ liệu từ MUX sẽ được ghi vào thanh ghi đích được xác định bởi trường rd trong lệnh. Điều này đánh dấu sự hoàn tất vòng đời thực thi của một lệnh, và kết quả sẽ sẵn sàng cho các lệnh tiếp theo sử dụng.

Về mặt chức năng tổng thể, tầng WB có vai trò kết thúc chu trình thực thi lệnh một cách chính xác. Trong các lệnh tính toán như ADD, SUB, AND,... dữ liệu ghi lại là kết quả từ ALU. Trong khi đó, đối với các lệnh truy xuất bộ nhớ như LW, dữ liệu được ghi là giá trị được đọc từ bộ nhớ tại tầng MEM. Mặc dù là giai đoạn cuối, tầng WB đóng vai trò rất quan trọng trong việc đảm bảo tính đúng đắn của toàn bộ hệ thống pipeline – nếu kết quả không được ghi chính xác, các lệnh tiếp theo sẽ nhận dữ liệu sai và dẫn đến sai sót logic toàn cục.

Việc tổ chức tầng WB với sự kết hợp giữa MUX linh hoạt, thanh ghi pipeline MEM/WB ổn định, và Register File hiệu quả giúp tăng cường độ chính xác, tính linh hoạt và hiệu năng cho toàn bộ pipeline. Tầng này hoàn thiện chu trình xử lý lệnh, đồng thời góp phần đảm bảo khả năng thực thi liên tục và tối ưu của CPU theo kiến trúc RISC-V.

e) Khối điều khiển logic (Control Logic)

Trong kiến trúc vi xử lý RISC-V**,** Control Unit (bộ điều khiển) đóng vai trò trung tâm trong việc điều phối hoạt động của toàn bộ hệ thống pipeline. Nhiệm vụ chính của khối này là giải mã lệnh được nạp từ Instruction Memory **và** phát sinh các tín hiệu điều khiển phù hợp, nhằm dẫn dắt các khối chức năng như ALU, Register File, Data Memory, MUX,... hoạt động đồng bộ và đúng với yêu cầu của từng loại lệnh.

Dựa vào trường opcode và các trường phụ như funct3, funct7, Control Unit xác định định dạng lệnh (R-type, I-type, S-type, B-type, U-type, J-type), từ đó sinh ra các tín hiệu điều khiển tương ứng. Các tín hiệu này sẽ quy định luồng dữ liệu**,** nguồn toán hạng**,** loại phép toán, và nơi ghi kết quả, đảm bảo mỗi lệnh được thực thi chính xác qua các tầng pipeline.

Dưới đây là mô tả các tín hiệu điều khiển chính và vai trò của chúng trong hoạt động của hệ thống:

- **alu_op**

- *Chức năng*: Xác định loại phép toán mà ALU cần thực hiện, bao gồm các phép toán số học, logic, dịch bit hoặc so sánh.
- *Hoạt động*: Được suy ra từ opcode và chuyển đến khối ALU Control để lựa chọn phép toán cụ thể.
- *Ví dụ*: Với lệnh add, alu_op sẽ điều khiển ALU thực hiện phép cộng; với sub, điều khiển phép trừ.

- **mem_read / mem_write**

- *Chức năng*: Điều khiển thao tác truy xuất dữ liệu từ bộ nhớ.
- *Hoạt động*: mem_read được kích hoạt đối với lệnh load (như lw), còn mem_write được bật đối với lệnh store (như sw).
- *Ví dụ*: Lệnh lw x2, 0(x1) sẽ bật mem_read để đọc dữ liệu từ địa chỉ x1 + 0.

- **reg_write**

- *Chức năng*: Cho phép ghi kết quả vào thanh ghi đích trong Register File.
- *Hoạt động*: Khi bật, kết quả từ ALU hoặc từ bộ nhớ (nếu là lệnh load) sẽ được ghi vào thanh ghi chỉ định bởi trường rd.
- *Ví dụ*: Với lệnh add x3, x1, x2, kết quả x1 + x2 sẽ được ghi vào thanh ghi x3.

- **branch**
  - *Chức năng*: Điều khiển các lệnh nhảy có điều kiện như beq, bne.
  - *Hoạt động*: Dựa trên tín hiệu so sánh đầu ra từ ALU để xác định có thay đổi giá trị PC hay không.
  - *Ví dụ*: Với lệnh beq x1, x2, label, nếu x1 == x2, tín hiệu branch được kích hoạt để nhảy đến label.
- **alu_src**
  - *Chức năng*: Chọn nguồn toán hạng thứ hai đầu vào cho ALU.
  - *Hoạt động*: Nếu là lệnh I-type, ALU sử dụng immediate; nếu là R-type, sử dụng hai thanh ghi nguồn.
  - *Ví dụ*: Lệnh addi x3, x1, 10 sẽ bật alu_src để chọn immediate 10 làm toán hạng.
- **mem_to_reg**
  - *Chức năng*: Lựa chọn nguồn dữ liệu ghi vào thanh ghi đích.
  - *Hoạt động*: Bật nếu kết quả đến từ Data Memory (lệnh load); tắt nếu kết quả từ ALU.
  - *Ví dụ*: Lệnh lw x5, 0(x1) sẽ bật mem_to_reg để ghi dữ liệu đọc từ bộ nhớ vào x5.
- **jalr / auipc**
  - *Chức năng*: Điều khiển các lệnh nhảy không điều kiện (jalr) và các lệnh thao tác với thanh ghi PC như auipc.
  - *Hoạt động*: Cập nhật giá trị mới cho PC theo kết quả tính toán từ immediate hoặc từ thanh ghi nguồn.
  - *Ví dụ*: Với lệnh jalr x1, 0(x2), PC sẽ được cập nhật bằng x2 + 0.
- **lui**
  - *Chức năng*: Điều khiển lệnh LUI (Load Upper Immediate).
  - *Hoạt động*: Ghi giá trị immediate vào 20 bit cao của thanh ghi đích.
  - *Ví dụ*: lui x5, 0x1000 sẽ ghi 0x10000000 vào x5.

Tổng thể, Control Unit là bộ não của pipeline, giúp điều khiển các tín hiệu cho phép thực hiện, điều phối toàn bộ hoạt động thực thi lệnh bằng cách phát ra các tín hiệu điều khiển chính xác và kịp thời.

1.1.5 So sánh RISC-V với các ISA khác

Tiêu chí	RISC-V	ARM	x86
Bản quyền	Mở, miễn phí	Có bản quyền	Có bản quyền
Độ phức tạp	Đơn giản, dễ triển khai	Trung bình	Phức tạp
Tính mở rộng	Cao (mô-đun hóa tốt)	Tốt	Thấp
Hệ sinh thái	Đang phát triển mạnh	Rộng lớn	Rộng lớn
Phù hợp với nhúng/IoT	Rất phù hợp	Phù hợp	Không phù hợp

Bảng 5. *So sánh đặc điểm chung của RISC-V và các ISA khác*

1.1.6 Ứng dụng của RISC-V trong thực tiễn

Với đặc điểm kiến trúc mở, linh hoạt và dễ tùy biến, RISC-V ngày càng được ứng dụng rộng rãi trong nhiều lĩnh vực công nghệ, từ thiết bị nhúng công suất thấp đến các hệ thống tính toán hiệu năng cao. Một số lĩnh vực ứng dụng tiêu biểu của RISC-V trong thực tiễn có thể kể đến như sau:

- Thiết bị nhúng (Embedded Systems):

RISC-V đang được tích hợp trong các bộ vi điều khiển (MCU) sử dụng cho hệ thống cảm biến, thiết bị đo đạc, điều khiển tự động trong công nghiệp, robot, thiết bị gia dụng thông minh,... Nhờ cấu trúc đơn giản và hiệu quả, RISC-V giúp giảm thiểu chi phí phần cứng mà vẫn đảm bảo hiệu suất và khả năng mở rộng.

- Thiết bị IoT (Internet of Things):

Với khả năng tiêu thụ năng lượng thấp và kiến trúc mô-đun có thể tùy chỉnh theo yêu cầu, RISC-V trở thành nền tảng lý tưởng cho các thiết bị kết nối như thiết bị đeo tay thông minh, cảm biến không dây, thiết bị y tế cá nhân, thiết bị theo dõi môi trường,... Nhiều vi xử lý RISC-V hiện đã được tối ưu hóa cho các ứng dụng IoT thời gian thực với khả năng ngủ sâu và phản hồi nhanh.

- Xử lý tín hiệu và bảo mật (DSP & Cryptography):

RISC-V hỗ trợ mở rộng tập lệnh tùy chỉnh, cho phép tích hợp các phần tử tăng tốc phần cứng như AES (Advanced Encryption Standard), SHA (Secure Hash Algorithm), hoặc các tập lệnh xử lý vector. Điều này giúp tăng tốc đáng kể các tác

vụ liên quan đến mã hóa, giải mã, xử lý tín hiệu số và trí tuệ nhân tạo – đặc biệt phù hợp trong các ứng dụng yêu cầu cao về bảo mật và hiệu suất tính toán.

- Hệ thống máy tính hiệu năng cao (HPC & General Computing):

Nhiều doanh nghiệp và tổ chức nghiên cứu lớn đang đầu tư phát triển bộ vi xử lý RISC-V cho máy tính cá nhân, máy chủ, và siêu máy tính (High - Performance Computing). Với khả năng mở rộng từ 32-bit đến 64-bit và 128-bit, cùng hỗ trợ đa lõi, RISC-V hứa hẹn sẽ là nền tảng cạnh tranh trong các hệ thống tính toán hiệu năng cao trong tương lai.

- Giáo dục và nghiên cứu:

Nhờ vào tính chất mở, minh bạch, không ràng buộc bản quyền, RISC-V đã trở thành lựa chọn hàng đầu trong giảng dạy và nghiên cứu về kiến trúc máy tính tại các trường đại học, viện nghiên cứu trên toàn cầu. Việc tiếp cận dễ dàng với mã nguồn và công cụ phần mềm đi kèm giúp sinh viên và nhà nghiên cứu hiểu sâu về cấu trúc và nguyên lý hoạt động của vi xử lý, đồng thời thử nghiệm các cải tiến kiến trúc một cách dễ dàng.

1.2 Tổng quan về mã mật AES

1.2.1. Giới thiệu về AES

Thuật toán AES là một thuật toán mã hóa khối đối xứng, tức là cùng một khóa được sử dụng cho cả quá trình mã hóa và giải mã, và dữ liệu được chia thành từng khối cố định có độ dài 128 bit để xử lý. Về mặt kỹ thuật, AES có thể hoạt động với ba độ dài khóa khác nhau: 128 bit, 192 bit và 256 bit, tương ứng với ba phiên bản là AES-128, AES-192 và AES-256. Số vòng (round) xử lý của thuật toán phụ thuộc vào độ dài khóa: lần lượt là 10, 12 và 14 vòng cho các khóa 128, 192 và 256 bit.

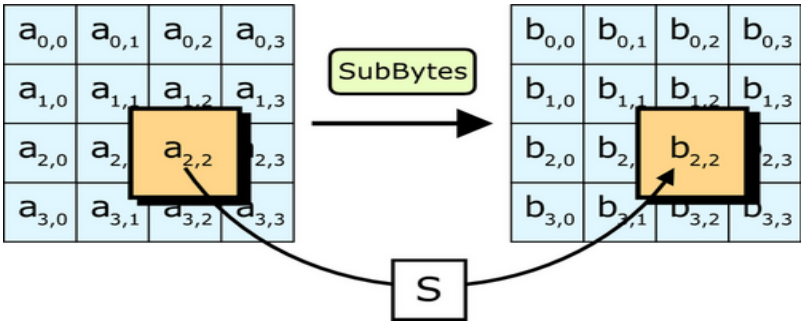
Điểm nổi bật trong kiến trúc của AES là việc nó xử lý dữ liệu trong một ma trận 4x4 byte, còn gọi là State, nơi mỗi byte được xem là một phần tử của trường hữu hạn GF(2^8). Tất cả các phép toán trong AES, bao gồm phép cộng, nhân, hoán vị… đều được thực hiện trong không gian đại số này, nhằm đảm bảo tính an toàn và hiệu quả.

1.2.2. Cấu trúc thuật toán AES

Về mặt cấu trúc, thuật toán AES gồm ba thành phần chính: key expansion (mở rộng khóa), vòng lặp mã hóa chính (main rounds) và vòng khởi đầu (initial round). Mỗi vòng mã hóa (trừ vòng đầu và cuối) gồm bốn bước chính, cụ thể như sau:

1.2.2.1 SubBytes (Thay thế byte)

Các byte được thế thông qua bảng tra S-box. Đây chính là quá trình phi tuyến của thuật toán. Hộp S-box này được tạo ra từ một phép biến đổi khả nghịch trong trường hữu hạn GF (2^8) có tính chất phi tuyến. Để chống lại các tấn công dựa trên các đặc tính đại số, hộp S-box này được tạo nên bằng cách kết hợp phép nghịch đảo với một phép biến đổi affine khả nghịch. Hộp S-box này cũng được chọn để tránh các điểm bất động (fixed point).



Hình 14. *Minh họa bước SubBytes trong thuật toán AES*

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

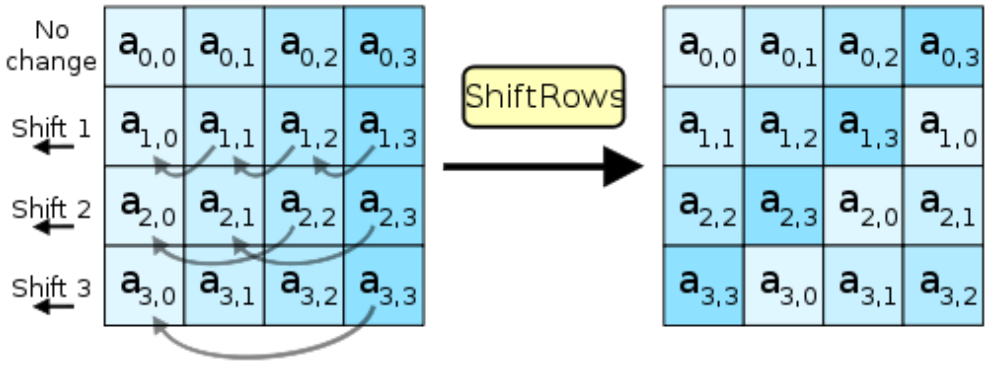
Bảng 6*. Bảng ánh xạ Sbox (dùng cho mã hóa)*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	82	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	89	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	84	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	87	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	38	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Bảng 7. *Bảng ánh xạ ngược Sbox của AES (dùng trong giải mã)*

1.2.2.2. ShiftRows (Dịch hàng)

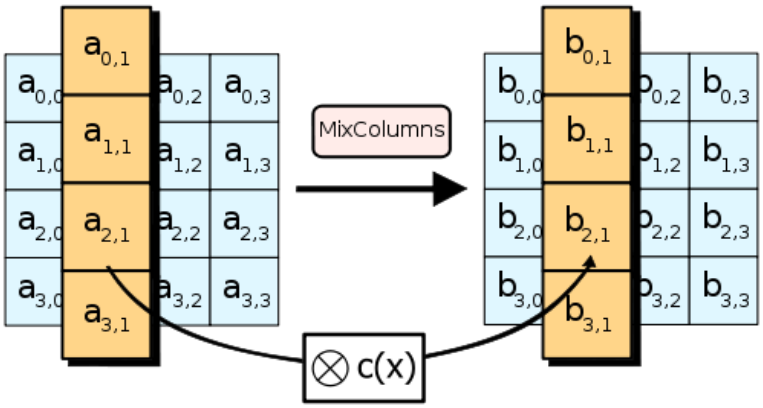
Bước này thực hiện việc dịch vòng các byte trong từng hàng của ma trận trạng thái theo một độ dịch xác định. Cụ thể, hàng đầu tiên không dịch, hàng thứ hai dịch trái 1 byte, hàng thứ ba dịch trái 2 byte và hàng thứ tư dịch trái 3 byte. Mục đích là để phá vỡ tính cục bộ trong dữ liệu.



Hình 15. *Minh họa bước ShiftRows trong thuật toán AES*

1.2.2.3 MixColumns (Trộn cột)

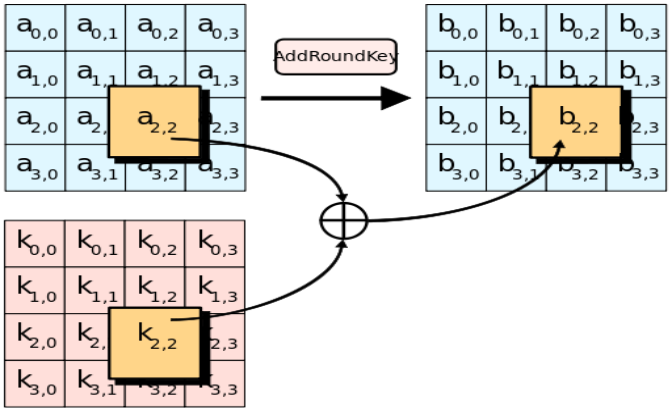
Mỗi cột của ma trận được coi là một đa thức bậc 3 trong GF(2^8) và nhân với một ma trận cố định để tạo ra sự khuếch tán (diffusion). Toán học trong bước này đảm bảo rằng một thay đổi nhỏ trong input sẽ ảnh hưởng đến toàn bộ block output



Hình 16. *Minh họa bước MixColumns trong thuật toán AES*

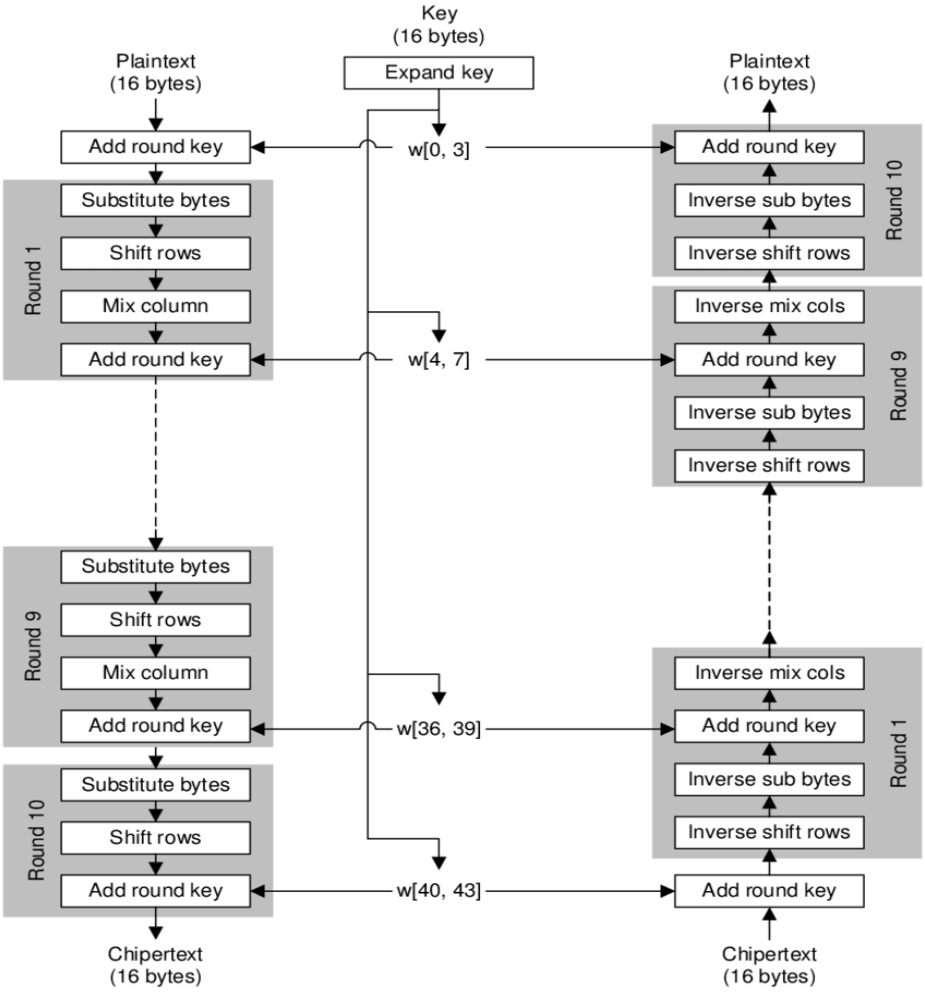
1.2.2.4 AddRoundKey (Cộng khóa vòng)

Trong bước này, khóa con (round key) được XOR với ma trận trạng thái hiện tại. Các khóa con được sinh ra từ quá trình mở rộng khóa ban đầu bằng một thuật toán gọi là **Key Expansion**. Ở vòng mã hóa đầu tiên chỉ có bước AddRoundKey, trong khi vòng cuối cùng bỏ qua bước MixColumns.



Hình 17. *Minh họa bước AddRoundKey trong thuật toán AES*

Tổng thể cấu trúc này tạo thành một mạng lưới khuếch tán và thay thế mạnh mẽ giúp AES đạt được độ an toàn rất cao, ngay cả khi triển khai trên phần cứng có tài nguyên hạn chế như các hệ thống IoT.



Hình 18. *Sơ đồ các bước mã hóa AES với độ dài khóa 128 bit*

1.2.3 Mô hình toán học và đại số trong AES:

Tất cả các phép toán trong AES được thực hiện trên trường hữu hạn GF(2^8) – một môi trường nhị phân gồm 256 phần từ. Trường này được xây dựng dựa trên đa thức bất khả quy $x^8 + x^4 + x^3 + x^1$,và các byte (8 bit) được coi như các phần tử trong trường. Phép nhân và cộng trong trường GF(2^8) không giống như số học thường. Mà sử dụng Modulo của đa thức đã nói trên. Cụ thể:

- Phép cộng trong GF(2^8) tương đương phép xor giữa hai byte

$$a \oplus b = a \text{ xor } b$$

- Phép nhân được thực hiện bằng cách nhân hai đa thức đại diện và sau đó chia lấy dư theo đã thức bất khả quy:

$$a \cdot b \bmod \left(x^8 + x^4 + x^3 + x^1\right)$$

Ma trận MixColumns sử dụng GF(2^8) với một ma trận cố định:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

Trong đó, các hệ số như ***02***, ***03*** cũng là những phần tử trong GF(2^8). Việc sử dụng đại số trừu tượng trong thiết kế thuật toán AES là một trong nhưng yếu tô giúp nó có tính bảo mật cao và khải năng triển khai trên phần cứng tốt.

1.3 Cơ sở lý thuyết và thực tiễn khi chọn đề tài

Một số nghiên cứu đã tập trung vào việc tăng tốc thuật toán AES [27], điều này đặc biệt quan trọng nhằm đảm bảo an toàn cho các thiết bị IoT, điện toán biên và thiết bị cá nhân, đồng thời vẫn duy trì hiệu quả và tính linh hoạt [19]. Trong [28], một bộ tăng tốc AES-256 dựa trên lõi RISC-V RV32IMFC 5 tầng đã cải thiện tốc độ xử lý từ 82–84% so với giải pháp phần mềm. Tuy nhiên, thiết kế này không tích hợp phần mở rộng tập lệnh AES chuyên biệt, dẫn đến hạn chế trong khả năng tích hợp hệ thống. Tương tự, nghiên cứu [29] đề xuất một tập lệnh tùy chỉnh cho AES trên lõi IBEX, đạt hiệu quả năng lượng cao hơn tới 662 lần so với TinyAES. Tuy vậy, thiết kế này không cho phép thực thi song song với bộ xử lý, làm giảm thông lượng trong các ứng dụng IoT đa nhiệm.

Trong khi đó, nghiên cứu [12] giới thiệu một bộ tăng tốc mật mã RISC-V với hai ALU 32-bit có thể nối ghép, cho phép thực hiện hai phép toán 32-bit song song hoặc một phép toán 64-bit kết hợp. Mặc dù đạt được mức tăng tốc xử lý từ 1.7 đến 3.0 lần, thiết kế này chỉ hỗ trợ AES-128 và không tối ưu hóa quá trình mở rộng khóa, làm giảm tính linh hoạt. Ngoài ra, nghiên cứu [27] cũng đề xuất một đơn vị mật mã linh hoạt có khả năng hỗ trợ nhiều thuật toán mật mã khác nhau. Tuy nhiên, đơn vị này không tối ưu hóa quá trình mở rộng khóa AES cũng như không hỗ trợ nhiều chế độ AES, dẫn đến giảm thông lượng và tính linh hoạt trong triển khai.

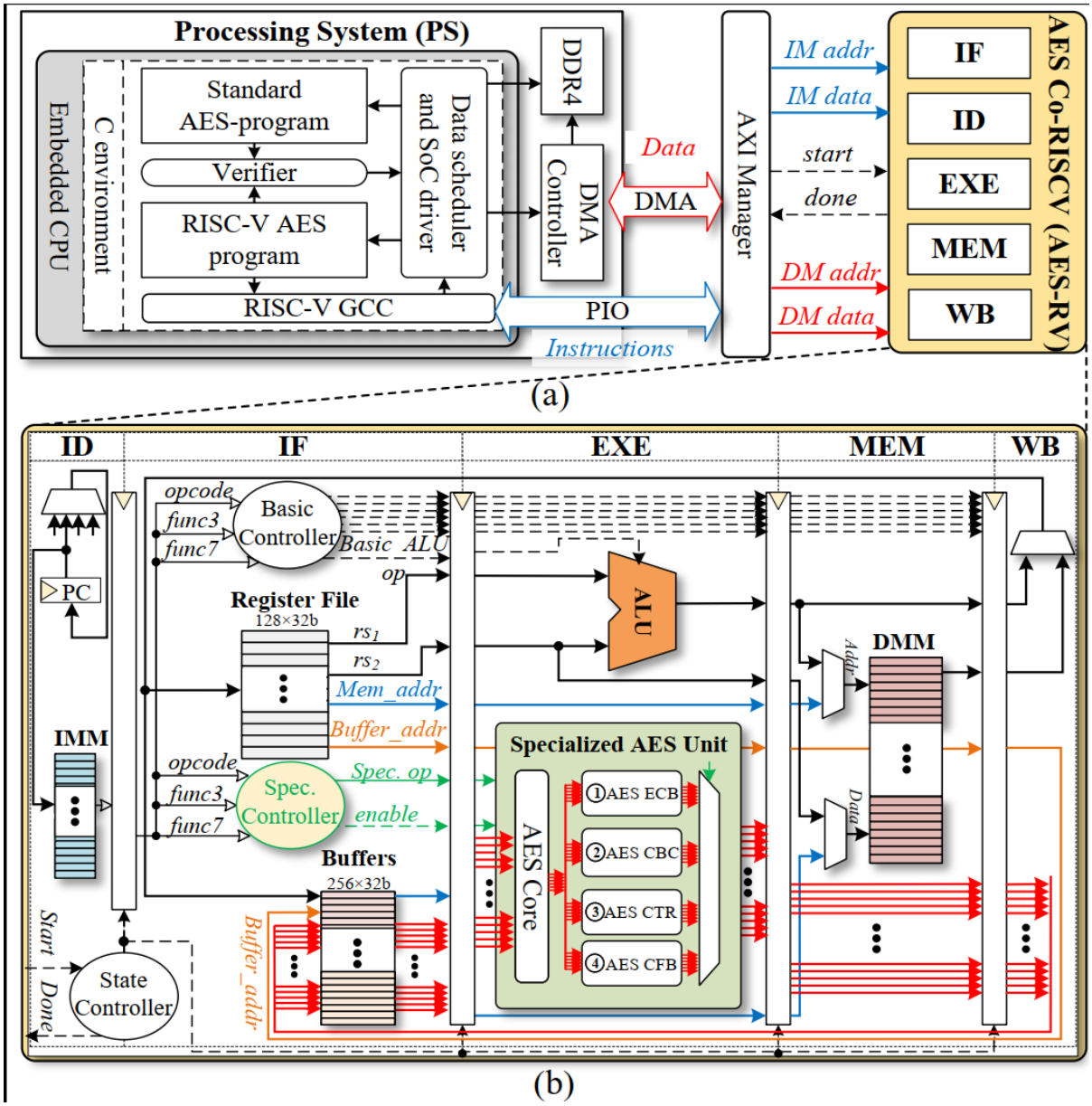
Tổng thể, các triển khai AES dựa trên kiến trúc RISC-V hiện nay vẫn gặp khó khăn trong việc cân bằng giữa tính linh hoạt và hiệu quả phần cứng, từ đó mở ra nhiều cơ hội để tiếp tục cải thiện về khả năng tích hợp, thực thi song song và hỗ trợ các chế độ hoạt động đa dạng của AES.

Để khắc phục những thách thức hiện tại, đề tài này đề xuất **AES-RV** – một bộ tăng tốc AES hiệu quả trên nền tảng RISC-V, được mở rộng với các lệnh AES độ trễ thấp nhằm nâng cao tính linh hoạt và hiệu quả năng lượng. Thiết kế AES-RV tích hợp ba cải tiến cốt lõi: (1) bộ đệm băng thông cao cho luồng dữ liệu liên tục, (2) đơn vị xử lý AES chuyên biệt với các lệnh độ trễ thấp, và (3) kiến trúc pipeline kết hợp với cơ chế bộ nhớ ping-pong nhằm tối ưu hóa luồng xử lý.

AES-RV được hiện thực hóa trên nền tảng SoC FPGA Xilinx ZCU102 và đã cho thấy hiệu năng vượt trội so với các nền tảng AES hiện có, nhờ vào khả năng hỗ trợ đầy đủ tất cả các chế độ và kích thước khóa của AES, đồng thời vẫn đáp ứng tốt các ràng buộc về tài nguyên và tiêu thụ năng lượng của thiết bị biên.

CHƯƠNG 2: KIẾN TRÚC ĐỀ XUẤT

2.1 Tổng quan kiến trúc hệ thống



Hình 1. (a) *Tổng quan kiến trúc hệ thống của AES-RV trong SoC.* ***(b)*** *Kiến trúc nội bộ của mô-đun AES-RV.*

Kiến trúc SoC tổng thể của bộ đồng xử lý AES trên nền RISC-V, như minh họa trong Hình 1(a), bao gồm hai thành phần chính: Hệ thống xử lý chính (Processing System – PS) và mô-đun AES-RV.

Hệ thống PS, được điều khiển bởi một bộ vi xử lý nhúng (embedded CPU), chịu trách nhiệm thực thi môi trường lập trình C để chạy cả chương trình AES tiêu

chuẩn và chương trình AES được xây dựng trên kiến trúc RISC-V. Chương trình AES-RISC-V sau khi được biên dịch sẽ được truyền vào mô-đun AES-RV để lưu trữ trong bộ nhớ lệnh (Instruction Memory – IM). Song song đó, dữ liệu tính toán được chuẩn bị thông qua bộ lập lịch dữ liệu (Data Scheduler) và trình điều khiển hệ thống (SoC Driver), giúp thiết lập và điều phối việc trao đổi dữ liệu với bộ nhớ dữ liệu (Data Memory – DM) của AES-RV.

Nhằm tối ưu hiệu suất truyền tải dữ liệu, dữ liệu lệnh được truyền thông qua cơ chế PIO (Programmed I/O), trong khi dữ liệu tính toán lớn được truyền bằng cơ chế DMA (Direct Memory Access) – vốn cho phép truyền dữ liệu tốc độ cao mà không cần can thiệp của CPU.

Mô-đun AES-RV bao gồm hai thành phần chính: AXI Manager và lõi AES-RV (AES-RV Core). Trong đó, AXI Manager chịu trách nhiệm giải mã và xử lý dữ liệu trao đổi giữa PS và mô-đun AES-RV. Cụ thể, dữ liệu tính toán được lưu vào bộ nhớ dữ liệu (DM), mã chương trình đã biên dịch (dưới dạng mã thập lục phân) được lưu vào bộ nhớ lệnh (IM), và các tín hiệu điều khiển như bắt đầu (start) và hoàn tất (done) được quản lý thông qua hai bộ điều khiển riêng biệt trong mô-đun AES-RV.

Tương tự như các kiến trúc RISC-V truyền thống, lõi xử lý AES-RV sử dụng một pipeline 5 tầng bao gồm: Instruction Fetch (IF) – nạp lệnh, Instruction Decode (ID) – giải mã lệnh, Execution (EXE) – thực thi, Memory Access (MEM) – truy cập bộ nhớ, và Writeback (WB) – ghi kết quả. Tuy nhiên, các lõi RISC-V thông thường chỉ sử dụng một ALU cơ bản với hai đầu vào và thực hiện một phép toán duy nhất trên mỗi chu kỳ, dẫn đến độ trễ cao khi thực hiện các phép toán AES phức tạp.

Để giải quyết vấn đề này, lõi AES-RV (Hình 1(b)) đã tích hợp một Đơn vị AES chuyên biệt (Specialized AES Unit – SAU) nhằm tăng tốc quá trình xử lý mật mã. SAU là một khối ALU linh hoạt cao, được thiết kế để hỗ trợ nhiều chế độ mã hóa AES như ECB, CBC, CTR và CFB, cùng với nhiều độ dài khóa khác nhau gồm 128, 192 và 256 bit. Đơn vị SAU được điều khiển thông qua một tập lệnh mở rộng tùy biến, cho phép thực thi các thao tác AES đa chế độ nhanh hơn đáng kể so với khi sử dụng các lệnh chuẩn của RISC-V.

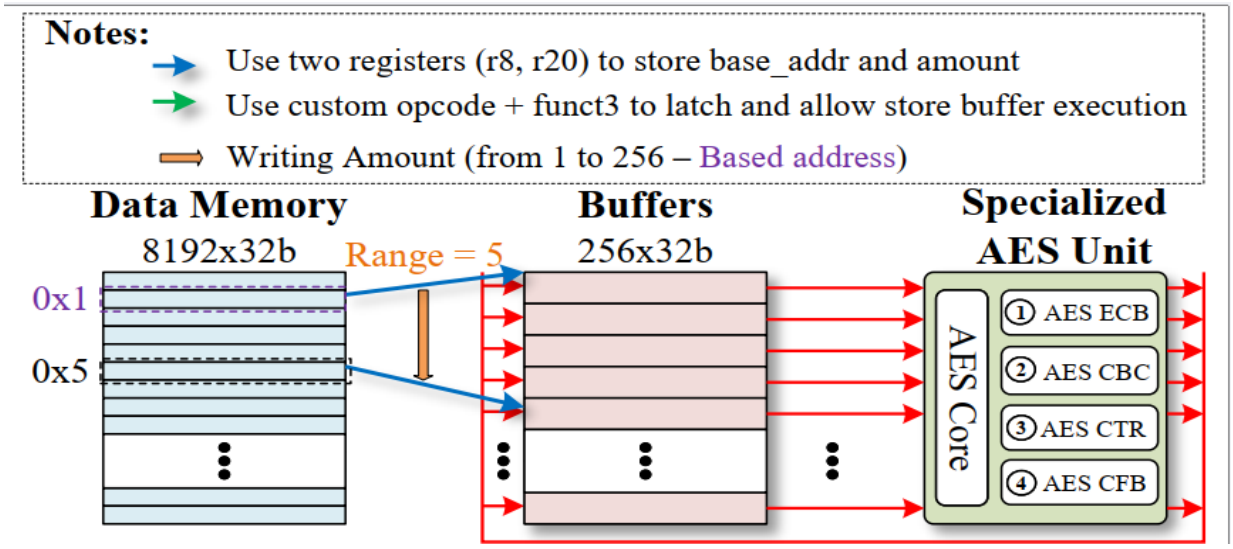
Để đảm bảo hiệu năng cao trong quá trình vận hành của SAU, hệ thống còn tích hợp một bộ đệm băng thông cao (high-bandwidth buffer system) nhằm hỗ trợ truyền dữ liệu trung gian hiệu quả giữa SAU và bộ nhớ dữ liệu DM.

2.2 Bộ đệm nội bộ băng thông cao phục vụ truy xuất dữ liệu liên tục tốc độ cao

Việc sử dụng đơn vị ALU trong lõi RISC-V tuân theo các quy tắc sử dụng tài nguyên truyền thống của kiến trúc RISC-V, đòi hỏi nhiều lệnh để tổ chức và sắp xếp dữ liệu phục vụ cho quá trình xử lý chính. Do đó, hiệu suất tính toán AES trên lõi RISC-V thông thường là rất thấp và không đáp ứng được các yêu cầu bảo mật hiện đại ngày nay.

Để khắc phục hạn chế này, kiến trúc **AES-RV** được cải tiến bằng cách tích hợp **Đơn vị AES chuyên biệt (Specialized AES Unit – SAU)**, bao gồm các thành phần được tối ưu hóa đặc biệt nhằm xử lý **AES đa chế độ** một cách hiệu quả. Do đặc thù yêu cầu lượng lớn dữ liệu đầu vào trong các vòng lặp tính toán liên tục, khối SAU cần được cấp phát dữ liệu với tốc độ cao và ổn định để đảm bảo hiệu năng.

Nhằm tránh việc gián đoạn trong luồng tính toán và duy trì hiệu suất xử lý cao, một hệ thống **256 bộ đệm (buffer) 32-bit** được bố trí phía trước khối SAU. Các bộ đệm này có khả năng lưu trữ lượng dữ liệu lớn như **khóa mã hóa (key)**, **vector khởi tạo (IV)**, **bản rõ (plain text)** và **bản mã (cipher text)**. Nhờ vào khả năng **nạp và lưu trữ toàn bộ dữ liệu chỉ trong một chu kỳ**, hệ thống đệm này cho phép SAU thực hiện các phép toán AES **liên tục và không bị gián đoạn do thiếu dữ liệu**, từ đó tối ưu hóa thông lượng xử lý và giảm đáng kể độ trễ toàn cục.



***Hình 2**. Bộ đệm nội bộ băng thông cao phục vụ truy xuất dữ liệu liên tục tốc độ cao*

Hình 2 minh họa chi tiết **cơ chế giao tiếp giữa bộ đệm băng thông cao và bộ nhớ dữ liệu (DM)**. Trong sơ đồ này, một lượng lớn dữ liệu có thể được truyền giữa **bộ nhớ dữ liệu (DM)** và **bộ đệm** thông qua việc quản lý các giá trị lưu trong hai

thanh ghi **r8** và **r20**. Cụ thể, thanh ghi **r8** lưu trữ địa chỉ trong DM dùng để đọc hoặc ghi dữ liệu, trong khi thanh ghi **r20** chứa số lượng phần tử 32-bit cần đọc hoặc ghi.

Buffer Accessing Instruction

funct7	rs2	rs1	funct3	rd	Opcode = 0101011
31 25	24 20	19 15	14 12	11 7	6 0

Opcode	funct3	Description
0101011	000	Latch the value of base_addr and amount
0101011	001	Set a flag to allow storing into the buffer

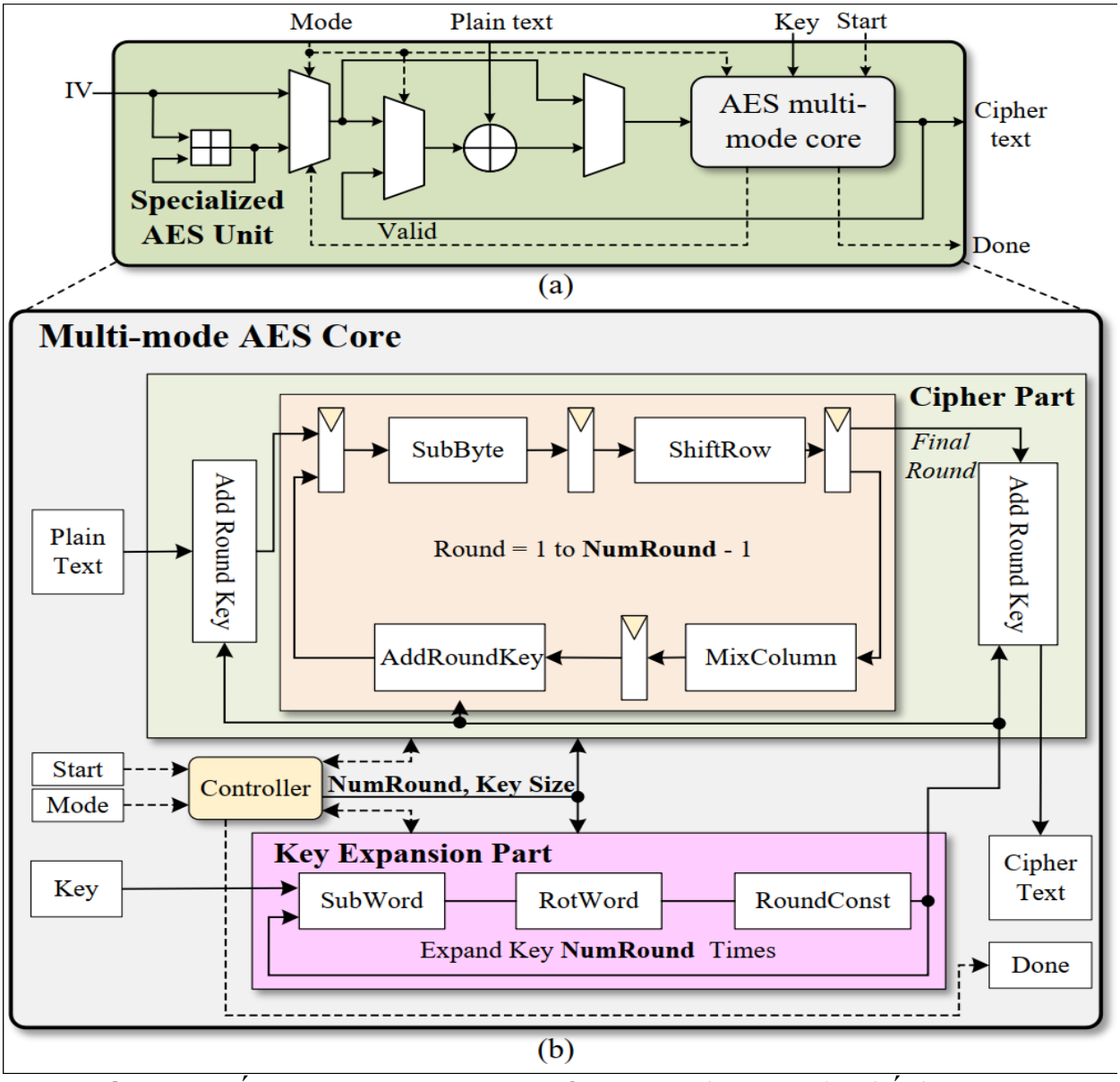
***Hình 3**. Cấu trúc các lệnh truy cập bộ đệm*

Quá trình truyền dữ liệu này được thực hiện bằng **tập lệnh truy cập bộ đệm**, như được mô tả trong Hình 3. Tập lệnh này bao gồm một lệnh dùng để chỉ định địa chỉ và số lượng dữ liệu, và một lệnh khác dùng để thiết lập cờ (flag) kích hoạt quá trình đọc/ghi. Khi toàn bộ bộ đệm đã được nạp đầy dữ liệu, **Đơn vị AES chuyên biệt (SAU)** có thể tải toàn bộ nội dung này và ngay lập tức tiến hành **mở rộng khóa AES (AES Key Expansion)** cũng như **tính toán các vòng chính (main rounds)**.

Trong khi đó, nếu sử dụng **tập lệnh RISC-V gốc**, sau mỗi vòng tính toán, hệ thống buộc phải thực thi một loạt lệnh để **tái sắp xếp dữ liệu** và lưu chúng vào bộ nhớ BRAM. Với việc tích hợp **bộ đệm băng thông cao**, toàn bộ quá trình tốn thời gian và tài nguyên này đã được loại bỏ hoàn toàn. Cần nhấn mạnh rằng, mỗi lần tái sắp xếp dữ liệu như vậy không chỉ cần nhiều lệnh mà còn mất nhiều thời gian **hơn cả thời gian thực hiện tính toán AES**, điều này cho thấy rõ lợi ích thiết thực của cơ chế đệm tốc độ cao trong việc tối ưu hóa hiệu năng tổng thể.

2.3 Đơn vị tính toán AES chuyên biệt (SAU) với tập lệnh mở rộng độ trễ thấp

Việc triển khai thuật toán AES thông qua tập lệnh RISC-V truyền thống thường dẫn đến độ trễ cao, do phải sử dụng một số lượng lớn lệnh để thực hiện toàn bộ quá trình mã hóa. Nhằm khắc phục điểm yếu này, đề tài đề xuất Đơn vị AES chuyên biệt (Specialized AES Unit – SAU), như minh họa trong Hình 4, để tăng tốc quá trình tính toán AES.Khối SAU đảm nhiệm việc xử lý toàn bộ các thành phần dữ liệu liên quan bao gồm bản rõ (plain text), khóa (key), vector khởi tạo (IV) và bản mã (cipher text). Quá trình điều khiển SAU được thực hiện thông qua hai tín hiệu điều khiển chính là start và done. Ngoài ra, thông tin cấu hình (configuration data) – bao gồm chế độ hoạt động – sẽ xác định số vòng lặp (rounds) và độ dài khóa tương ứng cho lõi AES đa chế độ.



Hình 4. *Kiến trúc Đơn vị AES chuyên biệt và lõi AES đa chế độ.*

Lõi xử lý AES đa chế độ (multi-mode AES core) trong SAU gồm ba thành phần chính: khối mã hóa (cipher part), khối mở rộng khóa (key expansion part) và bộ điều khiển (controller).

Trong đó:

- Khối mã hóa (cipher part) thực thi vòng lặp tính toán chính của thuật toán AES, gồm bốn bước: AddRoundKey, SubBytes, ShiftRows và MixColumns. Số vòng lặp và độ dài khóa được xác định dựa trên chế độ hoạt động đầu vào, do bộ điều khiển điều phối.
- Để giảm thiểu đường truyền tới hạn (critical path) và tăng tốc xử lý, phần mã hóa được triển khai theo kiến trúc pipeline 4 tầng.

- Khối mở rộng khóa (key expansion) chịu trách nhiệm tạo ra các khóa vòng (round keys) cần thiết cho từng vòng tính toán AES. Quá trình này sử dụng ba hàm chính:
  - SubWord (thay thế từng từ bằng S-box),
  - RotWord (xoay vòng các byte trong từ), và
  - RoundConst (cộng hằng số vòng).

Các hàm này giúp mở rộng khóa gốc thành chuỗi khóa phục vụ cho toàn bộ quá trình mã hóa.

Việc điều khiển hoạt động của SAU được thực hiện thông qua tập lệnh AES tùy biến (custom AES instruction set) như minh họa trong Hình 5. Bộ tập lệnh này được chia thành ba nhóm, tương ứng với các độ dài khóa: 128, 192 và 256 bit, xác định bởi mã opcode. Đồng thời, thông qua trường func3, khối SAU hỗ trợ tối đa bốn chế độ AES: ECB (Electronic Code Book), CFB (Cipher Feedback), CBC (Cipher Block Chaining) và CTR (Counter Mode).

AES Custom Instruction — Opcode = 0001011

funct7	rs2	rs1	funct3	rd
31 25	24 20	19 15	14 12	11 7

	Opcode	Func3	AES Mode
Custom 1	00 010 11	000	ECB 128
Custom 1	00 010 11	001	CFB 128
Custom 1	00 010 11	010	CBC 128
Custom 1	00 010 11	011	CTR 128
Custom 2	10 010 11	000	ECB 192
Custom 2	10 010 11	001	CFB 192
Custom 2	10 010 11	010	CBC 192
Custom 2	10 010 11	011	CTR 192
Custom 3	11 010 11	000	ECB 192
Custom 3	11 010 11	001	CFB 192
Custom 3	11 010 11	010	CBC 192
Custom 3	11 010 11	011	CTR 192

***Hình 5**. Tệp lệnh tùy biến để gọi đơn vị AES chuyên biệt*

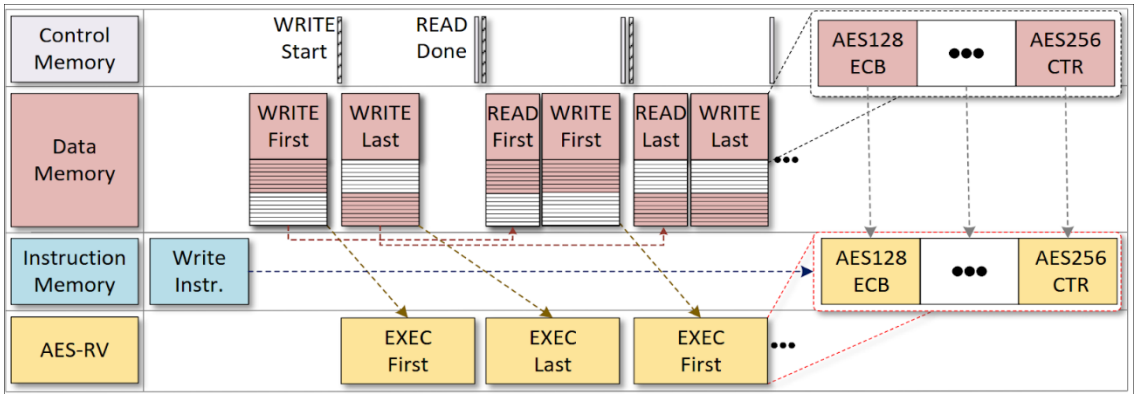
Tóm lại, việc tích hợp Đơn vị AES chuyên biệt (SAU) cùng với bộ đệm băng thông cao nội bộ đã mang lại sự cải thiện rõ rệt về hiệu năng cho quá trình tính toán AES đa chế độ. Thiết kế này không chỉ giảm độ trễ so với các giải pháp phần mềm truyền thống mà còn đảm bảo khả năng mở rộng, linh hoạt và hiệu quả năng lượng – đặc biệt phù hợp với các hệ thống nhúng hiện đại và thiết bị biên (edge devices).

2.4 Tổ chức đường ống (pipeline) hệ thống với cơ chế truyền dữ liệu bộ nhớ ping-pong

Trên thực tế, việc truyền một lượng lớn dữ liệu đến lõi AES-RV thông qua giao diện AXI có thể gây ra độ trễ đáng kể. Trong một hệ thống tổ chức theo cách tuần tự, thời gian chờ để đọc và ghi dữ liệu tính toán có thể vượt quá thời gian xử lý phần cứng, dẫn đến hiện tượng nghẽn cổ chai (bottleneck) cho toàn hệ thống. Nói cách khác, chỉ riêng việc tăng tốc lõi AES-RV không đủ để cải thiện đáng kể hiệu năng tổng thể nếu quá trình truyền dữ liệu vẫn còn chậm.

Để giải quyết vấn đề này, đề tài đề xuất một mô hình pipeline hệ thống kết hợp cơ chế truyền dữ liệu bộ nhớ kiểu ping-pong, nhằm giảm thiểu độ trễ truyền dữ liệu và tận dụng tối đa tài nguyên xử lý phần cứng.

Hình 6 minh họa lịch trình thời gian (timing schedule) của pipeline hệ thống đề xuất với cơ chế ping-pong. Trong mô hình này, bộ nhớ dữ liệu được chia làm hai phần bằng nhau: phần đầu (First) và phần cuối (Last). Ban đầu, chương trình lệnh và các thông tin cấu hình (bao gồm chế độ AES và độ dài khóa) được nạp vào lõi AES-RV. Sau đó, dữ liệu đầu vào được ghi vào nửa đầu của bộ nhớ dữ liệu (WRITE First), tiếp theo là tín hiệu điều khiển kích hoạt lõi AES-RV xử lý phần này (EXEC First).



Hình 6. *Lịch trình thời gian của pipeline hệ thống sử dụng cơ chế truyền dữ liệu bộ nhớ kiểu ping-pong.*

Trong khi EXEC First đang diễn ra, hệ thống song song ghi dữ liệu mới vào nửa sau của bộ nhớ dữ liệu (WRITE Last). Khi lõi AES-RV hoàn tất giai đoạn EXEC First, trong chu kỳ xử lý tiếp theo, hệ thống thực hiện đồng thời nhiều tác vụ:

- Đọc dữ liệu đã xử lý từ nửa đầu bộ nhớ (READ First)
- Ghi dữ liệu mới vào cùng vùng đó (WRITE First)
- Trong khi đó, lõi AES-RV xử lý dữ liệu ở nửa sau bộ nhớ (EXEC Last)

Mô hình này cho phép hệ thống xử lý thực hiện đồng thời ba nhiệm vụ: đọc kết quả đầu ra, ghi dữ liệu đầu vào mới, và tính toán vòng kế tiếp – tất cả đều diễn ra trong các vùng bộ nhớ xen kẽ. Nhờ vậy, lõi AES-RV có thể tiếp tục thực thi mà không phải chờ đợi quá trình truyền dữ liệu, giúp loại bỏ hoàn toàn thời gian rỗi do tắc nghẽn truyền dữ liệu trong toàn bộ hệ thống SoC.

Để đảm bảo đạt hiệu suất tối ưu, thời gian truyền dữ liệu phải ngắn hơn thời gian tính toán phần cứng, từ đó giúp hiệu năng toàn hệ thống đồng bộ với tốc độ xử lý phần cứng, bất kể độ trễ truyền tải tồn tại trên đường truyền.

CHƯƠNG 3: KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ HIỆU NĂNG

3.1 Kết quả triển khai trên SoC FPGA ZCU102

Để đánh giá tính đúng đắn và khả năng ứng dụng thực tiễn, kiến trúc AES-RV đã được triển khai trên nền tảng Zynq UltraScale+ MPSoC ZCU102, như minh họa trong Hình 1. Hệ thống xử lý (Processing System) được quản lý bởi bộ vi xử lý ARM Cortex-A53, chạy hệ điều hành Debian GNU/Linux 11 được cài đặt thông qua PetaLinux 2022.2. Phần logic khả trình (Programmable Logic) của SoC lưu trữ IP AES-RV, được tổng hợp bằng công cụ Vivado Design Suite 2022.2.

AES-RV đã được kiểm chứng bằng cách thực thi đầy đủ các hàm mã hóa AES với các độ dài khóa thuộc tập K = {128, 192, 256} và các chế độ hoạt động thuộc tập M = {ECB, CBC, CTR, CFB}

Thử nghiệm kiểm chứng thời gian thực trên SoC cho thấy AES-RV có thể xử lý 100.000 đầu vào bản rõ (plaintext) ngẫu nhiên cho mỗi chế độ hoạt động với độ chính xác 100%, hoạt động ở tần số 200 MHz.

Báo cáo sử dụng tài nguyên phần cứng cho thấy AES-RV tiêu tốn:

- 29.608 Flip-Flops (FFs)
- 32.483 Lookup Tables (LUTs)
- 12 khối RAM (Block RAM tiles), mỗi khối có dung lượng 36 KB

Phân tích tiêu thụ năng lượng cho thấy tổng công suất tiêu thụ của toàn hệ thống là 4.043 W, trong đó IP AES-RV chỉ chiếm 0.043 W công suất động (dynamic power).

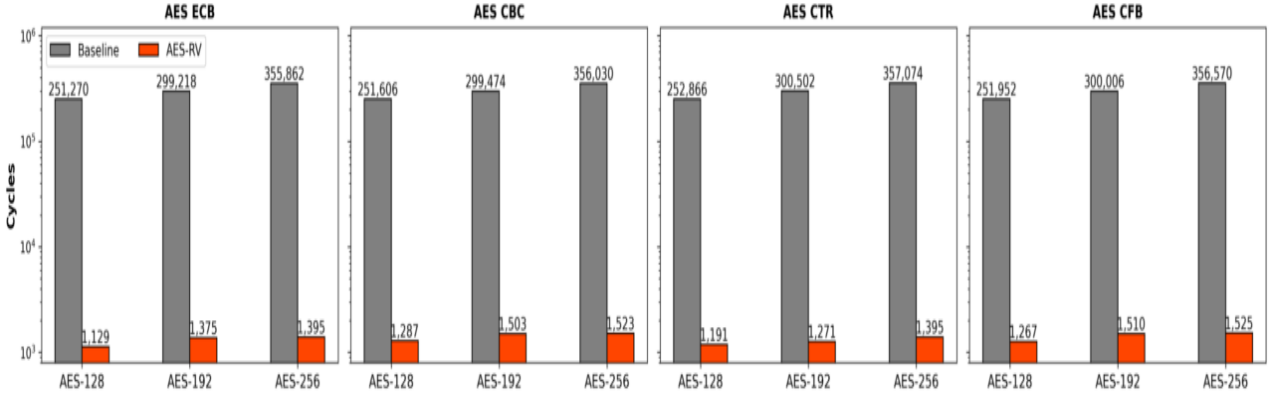
Nhìn chung, AES-RV thể hiện khả năng hoạt động ở tần số cao trên các hệ thống SoC thời gian thực trong khi vẫn tiêu thụ năng lượng ở mức rất thấp, điều này khiến nó trở thành một ứng viên lý tưởng để tích hợp vào các ứng dụng dựa trên SoC.

3.2 So sánh hiệu năng giữa AES-RV và triển khai RISC-V gốc

Trong phần này, số chu kỳ thực thi (execution cycles) của AES-RV khi mã hóa AES với tất cả các độ dài khóa KKK và chế độ MMM trên bốn khối dữ liệu liên tiếp được so sánh chi tiết với một phiên bản triển khai gốc của RISC-V. Kết quả so sánh được thu thập từ mô phỏng dạng sóng (waveform simulation) và được trình bày chi tiết trong Hình 7.

Đối với chế độ AES-ECB, AES-RV hoàn thành quá trình mã hóa trong khoảng 1.129 đến 1.395 chu kỳ, trong khi phiên bản RISC-V gốc cần từ 251.270

đến 355.862 chu kỳ, dẫn đến mức tăng tốc từ 217,61 lần đến 255,10 lần. Trong chế độ AES-CBC, AES-RV cần 1.287 đến 1.523 chu kỳ, so với 251.606 đến 356.030 chu kỳ ở bản gốc, đạt hiệu suất cao hơn 195,50 đến 233,77 lần. Với AES-CTR, thời gian thực thi của AES-RV dao động từ 1.191 đến 1.395 chu kỳ, so với 252.866 đến 357.074 chu kỳ của bản gốc, tương ứng mức tăng tốc từ 212,31 đến 255,97 lần. Cuối cùng, ở chế độ AES-CFB, AES-RV cần từ 1.267 đến 1.525 chu kỳ, trong khi phiên bản RISC-V gốc tiêu tốn 251.952 đến 356.570 chu kỳ, tương ứng hiệu suất cao hơn 198,68 đến 233,82 lần.



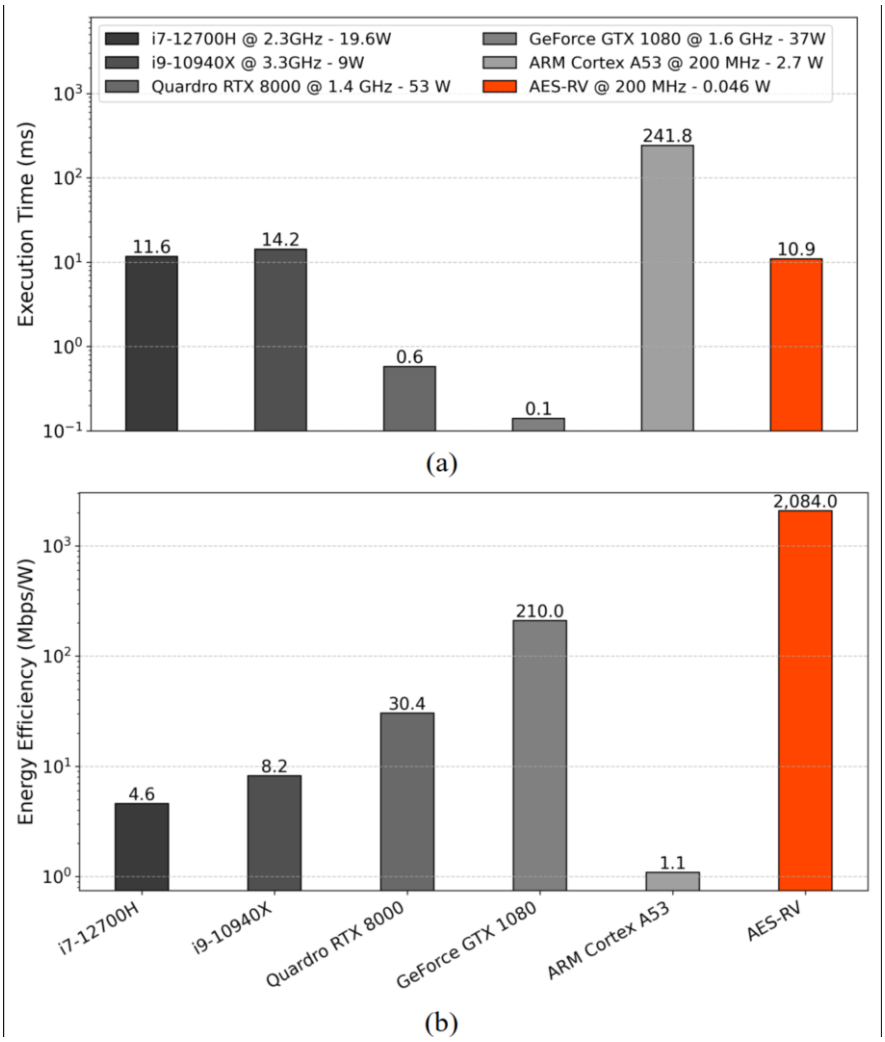
Hình 7. *So sánh hiệu năng giữa AES-RV và triển khai RISC-V cơ bản dựa trên số chu kỳ thực thi ở các chế độ AES và độ dài khóa khác nhau.*

Tổng kết lại, AES-RV vượt trội đáng kể so với kiến trúc RISC-V gốc nhờ vào tập lệnh AES tùy biến và các cơ chế tối ưu hóa bộ đệm, giúp giảm mạnh số chu kỳ xử lý, đồng thời nâng cao thông lượng và hiệu năng hệ thống tổng thể.

3.3 Đánh giá hiệu năng và hiệu quả năng lượng trên các nền tảng phần mềm thời gian thực

Để đánh giá hiệu năng và hiệu quả năng lượng, phần cứng đề xuất (AES-RV) đã được thử nghiệm với 8.192 bộ kiểm thử AES-CBC sử dụng khóa 128-bit, và so sánh với các nền tảng phần cứng hiệu suất cao bao gồm: CPU (Intel Core i7-12700H, Core i9-10940X), GPU (Quadro RTX 8000, GeForce GTX 1080 với khả năng xử lý 2.048 luồng), và CPU ARM Cortex-A53 trên nền tảng FPGA ZCU102 để đánh giá hiệu quả năng lượng. Kết quả chi tiết được trình bày trong Hình 8.Trong phần so sánh thời gian thực thi (Hình 8(a)), AES-RV đạt hiệu năng tương đương với Intel Core i7-12700H @ 2.3GHz, với tốc độ nhanh hơn 1,06 lần (11,6 ms so với 10,9 ms). Ngoài ra, AES-RV còn vượt qua Intel Core i9-10940X @ 3.3GHz với mức tăng tốc 1,30 lần (14,2 ms so với 10,9 ms). So với CPU ARM

Cortex-A53 @ 200 MHz, AES-RV thể hiện mức cải thiện đáng kể, nhanh hơn 22,18 lần (241,8 ms so với 10,9 ms). Đối với các GPU hiệu năng cao như Quadro RTX 8000 @ 1.4GHz và GeForce GTX 1080 @ 1.6GHz, AES-RV không vượt qua về mặt thời gian xử lý, do các GPU này hoàn thành tác vụ AES chỉ trong 0,6 ms và 0,1 ms tương ứng. Tuy nhiên, các CPU và GPU hiệu suất cao này có mức tiêu thụ điện năng rất lớn, khiến hiệu quả năng lượng của chúng thấp hơn đáng kể.



***Hình 8**. So sánh với các CPU/GPU hiệu năng cao về (a) thời gian thực thi và (b) hiệu quả năng lượng.*

Kết quả so sánh hiệu quả năng lượng (tính theo đơn vị Mbps/W) được trình bày trong Hình 8(b) cho thấy AES-RV vượt trội đáng kể so với cả CPU và GPU truyền thống. Cụ thể:

- AES-RV đạt hiệu quả năng lượng **2084 Mbps/W**, cao hơn:
  - **453,04 lần** so với **Intel Core i7-12700H @ 19,6W** (4,6 Mbps/W)

- **254,15 lần** so với **Intel Core i9-10940X @ 9W** (8,2 Mbps/W)**1894,55 lần** so với **ARM Cortex-A53 @ 2,7W** (1,1 Mbps/W)

- Đối với GPU:
  - AES-RV cao hơn **68,55 lần** so với **Quadro RTX 8000 @ 53W** (30,4 Mbps/W)
  - Và cao hơn **9,92 lần** so với **GeForce GTX 1080 @ 37W** (210 Mbps/W)

Tổng kết lại, AES-RV không chỉ đạt được hiệu năng xử lý tương đương với các CPU hiện đại, mà còn vượt trội rõ rệt về hiệu quả năng lượng so với cả CPU và GPU. Điều này làm cho AES-RV trở thành một giải pháp tối ưu cho các ứng dụng thời gian thực tiêu thụ điện năng thấp, đặc biệt trong các hệ thống nhúng và thiết bị biên.

3.4 Đánh giá toàn diện về hiệu năng và hiệu quả phần cứng trong các triển khai AES

Để chứng minh những cải tiến về hiệu quả phần cứng, các chỉ số đánh giá quan trọng như tần số hoạt động tối đa, mức sử dụng tài nguyên phần cứng, mức tiêu thụ điện năng, thông lượng (throughput), hiệu quả năng lượng (energy efficiency) và hiệu quả theo diện tích (area efficiency) đã được tổng hợp trong Bảng 8, nhằm so sánh với các nghiên cứu liên quan [27–30]. Đáng chú ý, chỉ có AES-RV và thiết kế trong [27] hỗ trợ thực thi toàn hệ thống trên SoC, trong khi các phương án còn lại chỉ tập trung vào mức lõi (core-level) mà không tích hợp đầy đủ vào hệ thống. Các công thức tính toán cho thông lượng, hiệu quả năng lượng, và hiệu quả diện tích lần lượt được trình bày trong Phương trình (1) – (3).

$$\text{Throughput} = \frac{F_{max} \times \text{Block Size}}{\text{Cycles per Block}} \ (\text{Mbps}) \qquad \textbf{(1)}$$

$$\text{Energy Efficiency} = \frac{\text{Throughput}}{\text{Power Comsumption}} \ (\text{Mbps/W}) \qquad \textbf{(2)}$$

$$\text{Efficiency} = \frac{\text{Throughput}}{\text{Total Slices}} \ (\text{Mbps/Slide}) \qquad \textbf{(3)}$$

Trong đó:

- $\mathbf{F_{max}}$**:** Tần số hoạt động tối đa của hệ thống phần cứng, đơn vị tính là Hertz (Hz). Tham số này thể hiện số chu kỳ đồng hồ mà hệ thống có thể thực hiện trong mỗi giây. Trong các hệ thống được triển khai trên FPGA hoặc SoC, giá trị này thường được xác định thông qua quá trình tổng hợp và phân tích thời gian (timing analysis).

- **Block Size**: Kích thước của một khối dữ liệu được mã hóa hoặc giải mã trong mỗi vòng xử lý, đơn vị là bit. Trong tiêu chuẩn AES, block size cố định là 128 bit (tương đương 16 byte).

- **Cycles per Block**: Số chu kỳ đồng hồ mà hệ thống cần để hoàn tất quá trình mã hóa hoặc giải mã một khối dữ liệu. Tham số này phụ thuộc vào cấu trúc pipeline, mức độ song song, độ sâu xử lý và các tối ưu hóa trong kiến trúc phần cứng AES.

- **Energy Efficiency:** (hiệu quả năng lượng) là chỉ số phản ánh lượng dữ liệu hệ thống có thể xử lý được trên mỗi đơn vị công suất tiêu thụ, đơn vị là megabit trên giây trên watt (Mbps/W)

- **Power Consumption** *(W)*: Là công suất điện tiêu thụ của hệ thống trong quá trình hoạt động, tính bằng Watt. Giá trị này bao gồm cả công suất động (dynamic) và công suất tĩnh (static).

- **Area Efficiency** (hiệu quả diện tích) là chỉ số thể hiện mức độ hiệu quả trong việc sử dụng tài nguyên phần cứng để đạt được thông lượng xử lý**,** đơn vị là megabit trên giây trên mỗi slice (Mbps/Slice)

- **Total Slices** *(Slice)*: Tổng số slices phần cứng được sử dụng trong quá trình tổng hợp thiết kế trên FPGA. Một slice là đơn vị cơ bản trong FPGA, bao gồm Look-Up Tables (LUTs), Flip-Flops**,** và các phần tử logic khác. Tổng số slices phản ánh mức độ chiếm dụng tài nguyên phần cứng.

References	Devices	F_{max} (MHz)	FFs	LUTs	BRAMs	#Slices[††]	Power (W)	Throughput (Mbps)	Energy Efficiency (Mbps/W)	Area Efficiency (Mbps/Slice)
ATC 2024 [27]	ZCU102 FPGA	210	7,584[*] 29,644[†]	7,562[*] 34,898[†]	16	2,531[*] 9,365[†]	0.136[*] 4.22[†]	4.81	3.54E+01[*] 1.14E+00[†]	1.90E-04[*] 5.14E-04[†]
ICECS 2024 [28]	22nm FDSOI ASIC	1,000	-	-	-	-	0.008	7.07	8.84E+02	-
PRIME 2024 [29]	Nexys Artix-7 FPGA	50	-	609	-	33,650	397	2.86	1.15E+02	8.50E-05
DDECS 2024 [30]	PYNQ Z2 FPGA	100	10,454	15,885	-	7,943	-	4.72	-	5.94E-04
AES-RV	ZCU102 FPGA	241	7,548[*] 29,608[†]	5,147[*] 32,483[†]	12	1,767[*] 8,601[†]	0.046[*] 4.043[†]	95.88	2.08E+03[*] 2.37E+01[†]	5.43E-02[*] 1.11E-02[†]

[*] Denotes results for the AES core only; [†] Denotes results for the entire SoC.
[††] PYNQ Z2: 1 Slice = 6 LUTs, 8 FFs; ZCU102: 1 Slice = 4 LUTs, 8 FFs, and 1 BRAM 36Kb = 40 slices.

Bảng 8. *So sánh các phương pháp triển khai AES trên nền tảng phần cứng về thông lượng và hiệu quả phần cứng.*

So với nghiên cứu [27] tại hội nghị ATC 2024, AES-RV hỗ trợ tần số hoạt động tối đa cao hơn 1,15 lần (241 MHz so với 210 MHz), đồng thời sử dụng ít tài nguyên phần cứng hơn cho cả lõi xử lý và hệ thống SoC với tỷ lệ lần lượt là

1,43/1,09 lần (1.767/8.601 slices so với 2.531/9.365 slices). Bên cạnh đó, AES-RV cũng tiêu thụ ít điện năng hơn 2,96/1,05 lần (0,046/4,043 W so với 0,136/4,22 W).

Về hiệu năng, AES-RV đạt thông lượng cao hơn 19,94 lần (95,88 Mbps so với 4,81 Mbps). Ngoài ra, AES-RV còn vượt trội về hiệu quả năng lượng, cao hơn 58,76/20,79 lần (2084,0/23,7 Mbps/W so với 35,4/1,14 Mbps/W) và hiệu quả theo diện tích cao hơn 285,26/21,6 lần (0,0543/0,0111 Mbps/Slice so với 0,00019/0,00051 Mbps/Slice). Tương tự, so với nghiên cứu [28] tại hội nghị ICECS 2024, AES-RV đạt thông lượng cao hơn 13,56 lần (95,88 Mbps so với 7,07 Mbps) và hiệu quả năng lượng cao hơn 2,36 lần (2084,0 Mbps/W so với 884 Mbps/W). Khi so sánh với [29] tại hội nghị PRIME 2024, AES-RV hoạt động ở tần số cao hơn 4,82 lần (241 MHz so với 50 MHz), sử dụng ít tài nguyên lõi hơn 19,05 lần (1.767 slices so với 33.650 slices) và tiêu thụ ít điện năng hơn 8.630 lần (0,046 W so với 397 W). Về hiệu năng, AES-RV đạt thông lượng cao hơn 33,52 lần (95,88 Mbps so với 2,86 Mbps), hiệu quả năng lượng cao hơn 18,08 lần (2084,0 Mbps/W so với 115,2 Mbps/W) và hiệu quả diện tích cao hơn 638,8 lần (0,0543 Mbps/Slice so với 0,000085 Mbps/Slice). Cuối cùng, so với nghiên cứu [30] tại hội nghị DDECS 2024, AES-RV hỗ trợ tần số tối đa cao hơn 2,41 lần (241 MHz so với 100 MHz) và yêu cầu ít tài nguyên lõi hơn 4,5 lần (1.767 slices so với 7.943 slices). Về hiệu năng, AES-RV đạt thông lượng cao hơn 20,32 lần (95,88 Mbps so với 4,72 Mbps) và hiệu quả diện tích cao hơn 91,42 lần (0,0543 Mbps/Slice so với 0,000594 Mbps/Slice).

Tổng kết lại, AES-RV thể hiện ưu thế vượt trội về thông lượng và hiệu quả phần cứng, nhờ vào tập lệnh được tối ưu hóa và kiến trúc phần cứng chuyên biệt, phù hợp với các hệ thống SoC yêu cầu xử lý thời gian thực với tiêu thụ năng lượng thấp.

KẾT LUẬN

Trong phạm vi nghiên cứu này, nhóm tác giả đã đề xuất và hiện thực hóa một bộ tăng tốc AES hiệu quả phần cứng trên nền tảng RISC-V, gọi là AES-RV, với mục tiêu cải thiện hiệu suất xử lý và hiệu quả năng lượng cho các ứng dụng mật mã trong hệ thống SoC nhúng và thiết bị biên thời gian thực. Hệ thống được thiết kế với ba cải tiến kiến trúc cốt lõi gồm: (1) bộ đệm nội bộ băng thông cao giúp truyền dữ liệu liên tục không gián đoạn, (2) đơn vị AES chuyên biệt (Specialized AES Unit – SAU) tích hợp tập lệnh mở rộng tùy biến hỗ trợ đầy đủ các chế độ AES (ECB, CBC, CTR, CFB) và các độ dài khóa (128, 192, 256 bit), và (3) cơ chế pipeline hệ thống kết hợp truyền dữ liệu bộ nhớ kiểu ping-pong nhằm tận dụng song song tối đa giữa xử lý và truyền tải. Các nội dung kỹ thuật được phân tích và hiện thực hóa trên nền tảng FPGA Zynq UltraScale+ MPSoC ZCU102 cho thấy AES-RV đạt được những kết quả nổi bật. Về hiệu suất, AES-RV cho thấy mức tăng tốc vượt trội từ 20 đến hơn 33 lần so với các thiết kế RISC-V cơ bản, và từ 13 đến hơn 19 lần so với các giải pháp phần cứng chuyên dụng được công bố gần đây. Đặc biệt, AES-RV chứng minh hiệu quả năng lượng vượt trội, cao hơn từ 18 đến gần 59 lần so với các CPU truyền thống như Intel Core i7/i9, ARM Cortex-A53 và thậm chí vượt cả GPU hiệu năng cao như Quadro RTX 8000 hay GTX 1080 với mức chênh lệch từ 9 đến hơn 68 lần. Đồng thời, kiến trúc đề xuất cũng cho thấy hiệu quả sử dụng tài nguyên phần cứng vượt trội, với hiệu quả theo diện tích cao hơn hàng trăm lần so với các thiết kế liên quan.

Như vậy, nghiên cứu này đã chứng minh rằng việc tích hợp các tối ưu hóa kiến trúc cùng tập lệnh chuyên biệt trên nền RISC-V có thể mang lại một giải pháp mật mã phần cứng hiệu quả, linh hoạt, có khả năng cạnh tranh với cả các nền tảng xử lý hiệu năng cao, đồng thời cung cấp một hướng tiếp cận mới trong việc hiện thực hóa các thuật toán bảo mật trên hệ thống nhúng thời gian thực. Trong tương lai, công trình này sẽ được mở rộng bằng cách tích hợp thêm tập lệnh hỗ trợ các thuật toán hậu lượng tử, điển hình như CRYSTALS-Kyber và CRYSTALS-Dilithium, nhằm tăng cường mức độ bảo mật và mở rộng phạm vi ứng dụng trong các hệ thống mật mã hiện đại.

TÀI LIỆU THAM KHẢO

[1] M. Sharma and et al., "A Survey of RISC-V CPU for IoT Applications," in *Proceedings of the International Conference on Innovative Computing & Communication (ICICC) 2022*, SSRN, February 2022.

[2] J. Park and et al., "Designing Low-Power RISC-V Multicore Processors With a Shared Lightweight Floating Point Unit for IoT Endnodes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, pp. 4106–4119, 2024.

[3] T.-T. Hoang and et al., "Low-power high-performance 32-bit RISCV microcontroller on 65-nm silicon-on-thin-BOX (SOTB)," *IEICE Electronics Express*, vol. 17, pp. 20200282–20200282, 2020.

[4] K.-D. Nguyen and et al., "A trigonometric hardware acceleration in 32-bit RISC-V microcontroller with custom instruction," *IEICE Electronics Express*, vol. 18, no. 16, pp. 20210266–20210266, 2021.

[5] M. Liu, "A co-design method of customized ISA design space exploration and fixed-point library construction for RISC-V dedicated processor," *IEICE Electronics Express*, vol. 19, no. 13, pp. 20220244– 20220244, 2022.

[6] Q. Yin and et al., "Design and implementation of RISC-V systemon-chip for SPWM generation based on FPGA," *IEICE Electronics Express*, vol. 21, no. 24, pp. 20240603–20240603, 2024.

[7] D. H. A. Le and et al., "High-Efficiency RISC-V-Based Cryptographic Coprocessor for Security Applications," in *International SoC Design Conference (ISOCC)*, pp. 103–104, 2024.

[8] Q. Zou and et al., "28nm asynchronous area-saving AES processor with high Common and Machine learning side-channel attack resistance," *IEICE Electron. Express*, vol. 18, p. 20210309, 2021.

[9] Leurent and et al., "New representations of the AES key schedule," *J. Cryptol.*, vol. 38, p. 1, 2025.

[10] W. K. Lee and et al., "Efficient Implementation of AES-CTR and AES-ECB on GPUs With Applications for High-Speed FrodoKEM and Exhaustive Key Search," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 69, p. 2962, 2022.

[11] Y. Zhang and et al., "A lightweight AES algorithm implementation for encrypting voice messages using field programmable gate arrays," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, p. 3878, 2022.

[12] N. H. Nguyen, , and et al., "LI-RV: A Fast and Efficient RISCV based Coprocessor for Lightweight Cryptography," in *2024 21st International SoC Design*

Conference (ISOCC), pp. 1–2, 2024.
[13] K. Stangherlin and M. Sachdev, "Design and Implementation of a Secure RISC-V Microprocessor," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 30, p. 1705, 2022.
[14] Pan and et al., "A Lightweight AES Coprocessor Based on RISC-V Custom Instructions," *Secur. Commun. Netw.*, p. 9355123, 2021.
[15] O. Simola and et al., "RISC-V Core with AES-256 Accelerator," in *31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, p. 1, 2024.
[16] Ignatius and et al., "Power Side-Channel Attacks on Crypto-core based on RISC-V ISA for High-security Applications," in *IEEE Access*, vol. 12, p. 150230, 2024.
[17] Cheng and et al., "A Hardware Security Evaluation Platform on RISCV SoC," in *IEEE International Test Conference in Asia (ITC-Asia)*, p. 1, 2024.
[18] O. Simola and et al., "RISC-V Core with AES-256 Accelerator," in *31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, p. 1, 2024.
[19] Salman and et al., "Lightweight Modifications in the Advanced Encryption Standard (AES) for IoT Applications: A Comparative Survey," in *2022 International Conference on Computer Science and Software Engineering (CSASE)*, pp. 325–330, 2022.
[20] S. Jeon and et al., "Cross-Layer Encryption of CFB-AES-TURBO for Advanced Satellite Data Transmission Security," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 58, p. 1, 2022.
[21] E. Choi and et al., "AESware: Developing AES-enabled low-power multicore processors leveraging open RISC-V cores with a shared lightweight AES accelerator," *Eng. Sci. Technol. Int. J.*, vol. 60, p. 1, 2024.
[22] C. Duran and E. Roa, "A 10pJ/bit 256b AES-SoC Exploiting Memory Access Acceleration," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 69, p. 1612, 2022.
[23] Y. M. Kuo and et al., "RISC-V Galois Field ISA Extension for Non-Binary Error-Correction Codes and Classical and Post-Quantum Cryptography," *IEEE Trans. Comput.*, vol. 72, p. 682, 2023.
[24] P. Nannipieri and et al., "VLSI Design of Advanced-Features AES Cryptoprocessor in the Framework of the European Processor Initiative," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 30, p. 177, 2022.
[25] W. Wang and et al., "An energy-efficient crypto-extension design for RISC-V," *Microelectron. J.*, vol. 115, p. 105165, 2021.
[26] D. Reis and et al., "IMCRYPTO: An In-Memory Computing Fabric for AES

Encryption and Decryption," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 30, p. 553, 2022.
[27] V. T. D. Le and et al., "RVCP: High-Efficiency RISC-V Co-Processor for Security Applications in IoT and Server Systems," in *2024 International Conference on Advanced Technologies for Communications (ATC)*, IEEE, 2024.
[28] Simola and et al., "RISC-V Core with AES-256 Accelerator," in *2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, 2024.
[29] Zgheib and et al., "Extending a RISC-V core with an AES hardware accelerator to meet IOT constraints," in *SMACD / PRIME 2021; International Conference on SMACD and 16th Conference on PRIME*, pp. 1–4, 2021.
[30] M. N. Rizi and et al., "Optimised AES with RISC-V Vector Extensions," in *2024 27th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 57–60, 2024