ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

ĐỒ ÁN TỐT NGHIỆP
CAPSTONE PROJECT

NGÀNH: KỸ THUẬT ĐIỀU KHIỂN VÀ TỰ ĐỘNG HÓA

ĐỀ TÀI:

THIẾT KẾ VÀ XÁC MINH LÕI IP MỀM ASIC (ASIC SOFT IP) CHUẨN MÃ HÓA AES-128

Người hướng dẫn: **TS. NGUYỄN THỊ THANH QUỲNH**
KS. NGUYỄN CAO THÀNH

Sinh viên thực hiện:
1. NGUYỄN HỮU VINH – MSSV: 105200439 – LỚP: 20TDHCLC2

Đà Nẵng, 6/2025

TÓM TẮT

Tên đề tài: Thiết Kế và Xác Minh Lõi IP Mềm ASIC (ASIC Soft IP) Chuẩn Mã Hóa AES - 128
Sinh viên thực hiện: Nguyễn Hữu Vinh
Số thẻ SV: 105200439 Lớp: 20TDHCLC2
Trong bối cảnh bảo mật dữ liệu trở thành yêu tố then chốt trong các hệ thống điện tử và cấu trúc IoT, việc thiết kế lõi IP mềm tích hợp trong cấu trúc vi điều khiển để thực hiện chức năng bảo mật dữ liệu trở thành một vấn đề cấp thiết. Đề tài này tập trung vào thiết kế và xác minh lõi IP mềm thực hiện chuẩn mã hóa AES-128. Chuẩn AES-128 là một thuật toán mã hóa đối xứng, sử dụng khóa 128 bit và hoạt động trên khối dữ liệu 128 bit qua 10 vòng lặp biến đổi. Đề tài thực hiện mô hình hóa thuật toán AES-128 ở mức Register Transfer Level sử dụng ngôn ngữ Verilog và SystemVerilog, nhằm đảm bảo khả năng tổng hợp và tích hợp vào các hệ thống phần cứng thực tế. Phần xác minh sử dụng phương pháp kiểm thử theo tầng lớp với SystemVerilog và Universal Verification Methodology, các thành phần chính sử dụng trong môi trường xác minh như driver, monitor, scoreboard và interface. Mô hình lõi IP AES sẽ sử dụng công cụ kiểm thử Xcelium của Cadence để kiểm tra các trường hợp có thể xảy ra để thiết kế chính xác.

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ tên sinh viên: Nguyễn Hữu Vinh Số thẻ sinh viên: 105200439

Lớp: 20TDHCLC2 Khoa: Điện Ngành: Kỹ thuật điều khiển và Tự động hóa

1. *Tên đề tài đồ án:*

Thiết Kế và Xác Minh Lõi IP Mềm ASIC (ASIC Soft IP) Chuẩn Mã Hóa AES - 128

2. *Đề tài thuộc diện: ☐Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện*

3. *Các số liệu và dữ liệu ban đầu:*

Nội dung các phần thuyết minh và tính toán:

Chương 1: Giới thiệu về bán dẫn và thiệt kế vi mạch

Chương 2: Giới thiệu về AES-128 về nguyên lý hoạt động và ứng dụng

Chương 3: Thiết kế lõi IP mềm, thiết kế khối mức cao và mức thấp, mô tả RTL cho toàn bộ thiết kế

Chương 4: Thực hiện xây dựng môi trường xác minh chức năng, tạo các trường hợp kiểm thử chức năng hoạt động của lõi IP

Chương 5: Thực hiện mô phỏng thiết kế đặc tả ở chương 3 và xác minh chức năng ở chương 4.

4. *Các bản vẽ, đồ thị (ghi rõ các loại và kích thước bản vẽ):*

Họ tên người hướng dẫn: TS. Nguyễn Thị Thanh Quỳnh
KS. Nguyễn Cao Thành

5. *Ngày giao nhiệm vụ đồ án: 17/02/2025*

6. *Ngày hoàn thành đồ án: 17/06/2025*

Đà Nẵng, ngày 17 tháng 06 năm 2025

Trưởng Bộ môn Tự động hóa	**Người hướng dẫn**
TS. Giáp Quang Huy	TS. Nguyễn Thị Thanh Quỳnh

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

PHIẾU KIỂM SOÁT TIẾN ĐỘ LÀM ĐỒ ÁN TỐT NGHIỆP

(Phiếu dành cho người hướng dẫn/sinh viên)

Họ tên sinh viên: Nguyễn Hữu Vinh Số thẻ SV: 105200439

Tên đề tài ĐATN: Thiết Kế và Xác Minh Lõi IP Mềm ASIC (ASIC Soft IP) Chuẩn Mã Hóa AES - 128

Họ tên người HD: TS. Nguyễn Thị Thanh Quỳnh Đơn vị: Khoa Điện

Tuần	Ngày	Khối lượng		GVHD ký tên
		đã thực hiện (%)	tiếp tục thực hiện (%)	
1	17/2	Lựa chọn tên đề tài đồ án tốt nghiệp (100%)	Nghiên cứu sơ bộ về đề tài đồ án tốt nghiệp	
2	24/2	Nghiên cứu sơ bộ về đề tài đồ án tốt nghiệp, lắng nghe góp ý của kỹ sư và giáo viên hướng dẫn (100%)	Tìm hiểu về quy trình thiết kế vi mạch, đọc tài liệu về thuật toán mã hóa AES	
3	3/3	Duyệt lần 1: Đánh giá khối lượng hoàn thành %: Được tiếp tục làm ĐATN ☐ Không tiếp tục thực hiện ĐATN ☐		
4	10/3	Nghiên cứu thuật toán mã hóa AES (100%)	Thiết kế kiến trúc của toàn bộ lõi, giao tiếp giữa các khối, các tín hiệu vào ra.	
5	17/3			
6	24/3	Xây dựng kiến trúc tổng thể khối mức cao của lõi (100%)	Thiết kế chi tiết bên trong khối mức cao hoạt động 2 chức năng mã hóa và giải mã	
7	31/3	Thiết kế khối mức thấp của chức năng mã hóa (100%)	Nghiên cứu về chức năng tạo khóa vòng của mã hóa	
8	07/4	Thiết kế khối mức thấp của chức năng tạo khóa vòng mã hóa (100%)	Nghiên cứu chức năng giải mã và tạo khóa vòng ngược	
9	14/4	Thiết kế khối mức thấp của chức năng giải mã và tạo khóa vòng ngược (100%)	Nghiên cứu dùng ngôn ngữ mô tả phần cứng Verilog để mô tả thiết kế ở khối mức thấp	
10	21/4	Viết mô tả thiết kế bằng ngôn ngữ Verilog cho toàn bộ khối mức thấp, và thực hiện mô phỏng để kiểm tra (100%)	Nghiên cứu xây dựng môi trường xác minh chức năng cho thiết kế	
11	28/4	Xây dựng môi trường xác minh chức năng của thiết kế (100%)	Nghiên cứu các trường hợp để kiểm tra thiết kế, đảm bảo tất cả các trường hợp thực tế có thể xảy ra đều được kiểm thử	

12	05/5	Xây dựng các trường hợp kiểm tra, gồm 8 trường hợp kiểm tra cho 2 chức năng mã hóa và giải mã (100%)	Chạy mô phỏng để kiểm tra tính đúng đắn của thiết kế	
13	12/5	Xác minh chức năng của thiết kế hoạt động đúng như mong đợi, thiết kế thỏa mãn để chuyển qua bước tiếp theo của quy trình thiết kế vi mạch	Nghiên cứu viết thuyết minh đồ án, báo cáo tiến độ.	
14	19/5	Duyệt lần 2: Đánh giá khối lượng hoàn thành %: Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		
15	26/5	Viết báo cáo, hoàn thành các phần nhỏ (100%)	Làm slide và chuyển bị cho ngày bảo vệ đồ án	
16	02/6			
17	09/6			
18	16/6	Duyệt lần 3: Đánh giá khối lượng hoàn thành %: Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		

LỜI NÓI ĐẦU VÀ CẢM ƠN

Em xin chân thành gửi lời cảm ơn sâu sắc đến Cô Nguyễn Thị Thanh Quỳnh – giảng viên khoa Điện, Trường Đại học Bách Khoa Đà Nẵng, người đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình thực hiện đồ án tốt nghiệp với đề tài "Thiết Kế và Xác Minh Lõi IP Mềm ASIC (ASIC Soft IP) Chuẩn Mã Hóa AES - 128". Nhờ vào sự định hướng chuyên môn, những góp ý quý báu và sự động viên của Cô, em đã có thể hoàn thành tốt đồ án này.

Em cũng xin trân trọng cảm ơn anh Nguyễn Cao Thành – kỹ sư tại FPT Semiconductor, người đã hỗ trợ em rất nhiều về kiến thức, kỹ thuật, cho em sử dụng các phần mềm chuyên dụng và chia sẻ kinh nghiệm thực tế quý giá trong lĩnh vực thiết kế vi mạch, góp phần quan trọng vào việc hoàn thiện nội dung và định hướng triển khai đề tài.

Bên cạnh đó, em xin gửi lời cảm ơn đến quý thầy cô trong khoa Điện đã trang bị cho em kiến thức chuyên môn vững vàng trong suốt quá trình học tập.

Mặc dù đã cố gắng hết sức, nhưng đồ án không tránh khỏi những thiếu sót. Em rất mong nhận được sự góp ý của quý thầy cô và các anh chị để hoàn thiện hơn nữa.

Trân trọng cảm ơn!

LỜI CAM ĐOAN LIÊM CHÍNH HỌC THUẬT

Em xin cam đoan rằng đồ án tốt nghiệp với đề tài "Thiết Kế và Xác Minh Lõi IP Mềm ASIC (ASIC Soft IP) Chuẩn Mã Hóa AES - 128" là nghiên cứu độc lập của em với sự hỗ trợ từ giảng viên hướng dẫn TS. Nguyễn Thị Thanh Quỳnh và công ty FPT-Semiconductor.

Em xin cam đoan toàn bộ số liệu được cung cấp từ báo cáo đều là của công ty và đây là kết quả nghiên cứu hoàn toàn trung thực, không sao chép từ bất kì một công trình nghiên cứu khác nào. Những tài liệu trích dẫn đều đã được ghi rõ nguồn gốc.

Em xin chịu hoàn toàn trách nhiệm nếu có bất kỳ sự sao chép, gian dối kết quả nào trong sản phẩm đồ án này.

Sinh viên thực hiện

Nguyễn Hữu Vinh

MỤC LỤC

DANH SÁCH CÁC BẢNG, HÌNH VẼ

DANH SÁCH CÁC KÝ HIỆU, CHỮ VIẾT TẮT

KÝ HIỆU	CHỮ VIẾT TẮT, CHÚ GIẢI
VLSI	Very Large Scale Integration
RTL	Register Transfer Level
MUX	Multiplexer
UVM	Universal Verification Methodology
HDL	Hardware Description Language
VHDL	VHSIC Hardware Description Language
IP	Intellectual Property
SoC	System on Chip
FF	Flip Flop
IP	Intellectual Property
UVM	Universal Verification Methodology
DUT	Design Under Test
MSB	Most Significant Bit
LSB	Least Significant Bit
Phép nhân 2 byte	“.”
Phép XOR 2 byte	“+”

MỞ ĐẦU

Mục đích và mục tiêu thực hiện đề tài:

Trong bối cảnh Internet of Things (IoT) ngày càng phát triển mạnh mẽ và được ứng dụng rộng rãi trong nhiều lĩnh vực như công nghiệp, y tế, giao thông, nhà thông minh,... việc bảo vệ dữ liệu truyền tải trong các hệ thống nhúng trở nên vô cùng cấp thiết. Các thiết bị IoT thường có giới hạn về tài nguyên phần cứng, nhưng lại yêu cầu tính bảo mật cao để ngăn chặn việc rò rỉ, giả mạo hoặc đánh cắp thông tin. Do đó, việc tích hợp các giải pháp mã hóa dữ liệu hiệu quả và tối ưu vào vi điều khiển hoặc vi mạch nhúng là một xu hướng tất yếu. Trong các chuẩn mã hóa hiện nay, AES (Advanced Encryption Standard) là một trong những thuật toán phổ biến, mạnh mẽ và đã được chuẩn hóa bởi NIST, đặc biệt phiên bản AES-128 rất phù hợp với các hệ thống nhúng.
Vì lý do đó, em chọn đề tài " Thiết Kế và Xác Minh Lõi IP Mềm ASIC (ASIC Soft IP) Chuẩn Mã Hóa AES - 128" với mục tiêu như thiết kế một lõi mã hóa AES-128 có khả năng tích hợp vào kiến trúc vi điều khiển hoặc hệ thống SoC, phục vụ nhu cầu mã hóa dữ liệu trực tiếp trên phần cứng, đảm bảo lõi IP có khả năng tái sử dụng, tổng hợp được trên ASIC và tối ưu về tài nguyên phần cứng và hướng đến việc triển khai thực tế trong các hệ thống IoT có yêu cầu bảo mật cao.

Việc xây dựng lõi IP này không chỉ mang tính ứng dụng cao mà còn giúp em tiếp cận thực tế quy trình thiết kế và xác minh phần cứng số, chuẩn bị hành trang tốt cho công việc trong lĩnh vực thiết kế vi mạch.

Phạm vi nghiên cứu:

Đề tài tập trung vào chức năng mã hóa (Encryption) và chức năng giải mã (Decryption). Thiết kế lõi IP được thực hiện dưới dạng IP mềm, mô tả bằng ngôn ngữ mô tả phần cứng Verilog, kiểm nghiệm thiết kế sử dụng phần mềm ModelSim, chưa triển khai thiết kế vật lý.

Việc xác minh chức năng được thực hiện mô phỏng trên công cụ Xcelium của Cadence, xây dựng môi trường xác minh sử dụng ngôn ngữ SystemVerilog.

Đối tượng nghiên cứu

Thuật toán mã hóa AES-128: Cấu trúc, hoạt động nội tại, và các bước xử lý như SubBytes, ShiftRows, MixColumns, AddRoundKey.

Thiết kế lõi IP mềm: Mô tả thuật toán AES-128 bằng ngôn ngữ phần cứng Verilog, đảm bảo khả năng tổng hợp để tích hợp vào các kiến trúc vi điều khiển hoặc hệ thống SoC.

Quy trình xác minh phần cứng: Áp dụng phương pháp kiểm thử theo tầng lớp và sử dụng mô hình UVM (Universal Verification Methodology) để kiểm tra chức năng và độ đúng đắn của lõi IP.

Tích hợp vào hệ thống vi mạch: Định hướng sử dụng lõi IP trong các ứng dụng yêu cầu bảo mật như thiết bị IoT, vi điều khiển, hoặc nền tảng ASIC.

Phương pháp nghiên cứu:

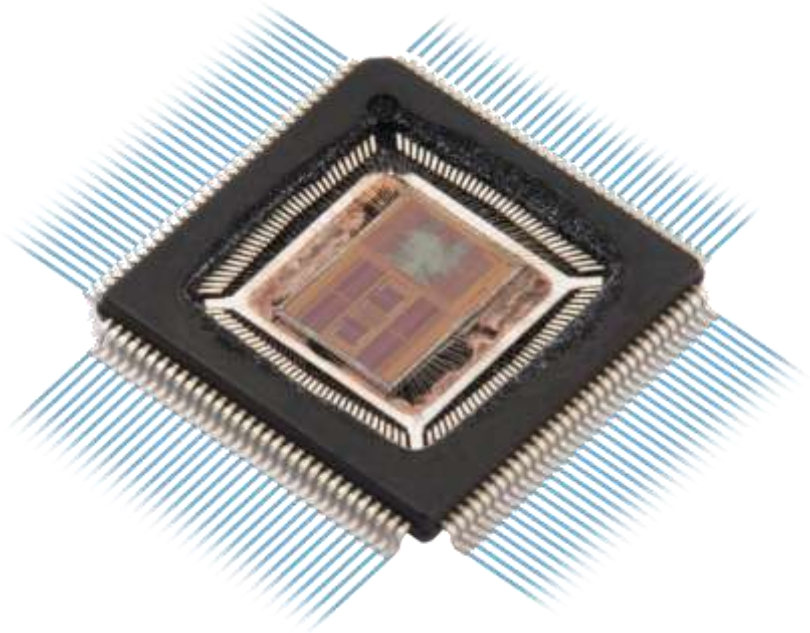
Tìm hiểu chi tiết về thuật toán mã hóa AES-128 thông qua các tài liệu chuẩn của NIST. Phân tích kiến trúc phần cứng phù hợp để triển khai thuật toán AES-128 ở mức RTL. Áp dụng ngôn ngữ Verilog để mô tả thuật toán AES-128. Thiết kế các khối chức năng chính như: KeyExpansion, RoundFunction, MixColumns, SubBytes,... dưới dạng module độc lập dể dễ kiểm thử và sửa lỗi. Sử dụng phần mềm mô phỏng như ModelSim để chạy và kiểm thử chức năng RTL. Tìm hiểu các kỹ thuật xác minh chức năng, đặc biệt là phương pháp viết testbench theo mô hình tầng lớp và mô hình UVM. Viết testbench kiểm thử 2 chức năng chính là mã hóa và giải mã trong AES.

Cấu trúc của đồ án tốt nghiệp:

Chương 1: BÁN DẪN VÀ THIẾT KẾ VI MẠCH

1.1. Giới thiệu chung

Trong chương này sẽ tìm hiểu về khái niệm VLSI, một công nghệ quan trọng trong thiết kế vi mạch, cho phép tích hợp số lượng lớn linh kiện điện tử, đặc biệt là transistor, vào một vi mạch tích hợp (IC). Phần tiếp theo, sẽ trình bày cái nhìn tổng quan về quy trình thiết kế ASIC trong thiết kế vi mạch, trước khi đi sâu vào từng bước của thiết kế ASIC Front-End.



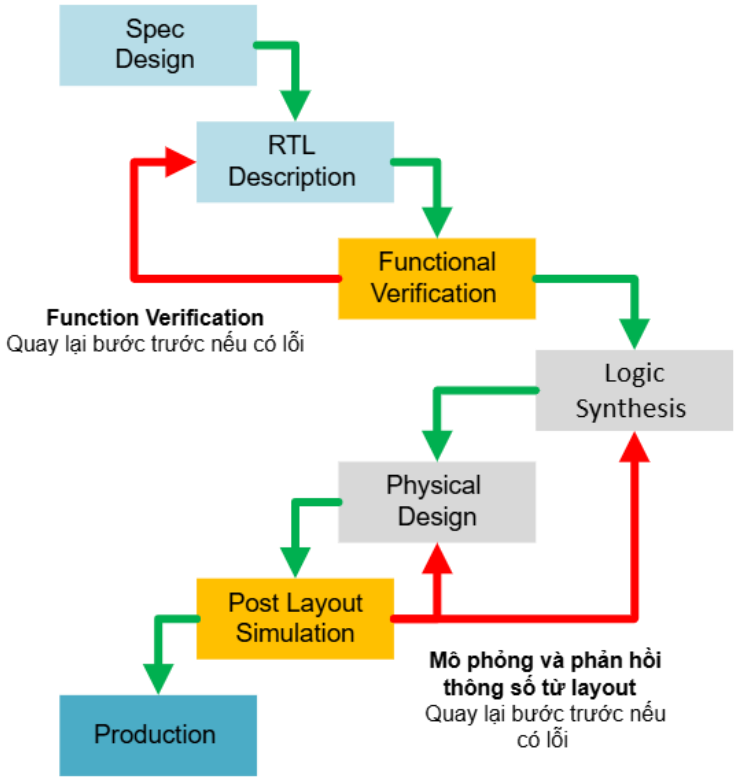
Hình 1.1: Hình ảnh về một vi mạch

VLSI, viết tắt của Very Large Scale Integration, là công nghệ được sử dụng để thiết kế và chế tạo các mạch tích hợp có mật độ linh kiện cao. Nhờ vào công nghệ này, một số lượng lớn transistor cùng các thành phần điện tử khác có thể được tích hợp trên cùng một con chip. Điều này giúp tạo ra các IC có độ phức tạp cao nhưng vẫn giữ được kích thước nhỏ gọn, phục vụ nhiều lĩnh vực khác nhau như thiết bị điện tử tiêu dùng, máy tính, hệ thống truyền thông và thiết bị y tế.

Một trong những ưu điểm nổi bật của VLSI là khả năng tích hợp một lượng lớn transistor vào một chip duy nhất, cho phép các IC thực hiện từ các tác vụ logic đơn giản đến những phép toán phức tạp. Công nghệ này không chỉ giúp tối ưu hiệu suất, tiết kiệm năng lượng mà còn nâng cao độ tin cậy của hệ thống, yếu tố quan trọng đối với nhiều ứng dụng thực tế, góp phần quan trọng vào sự phát triển của ngành công nghiệp bán dẫn.

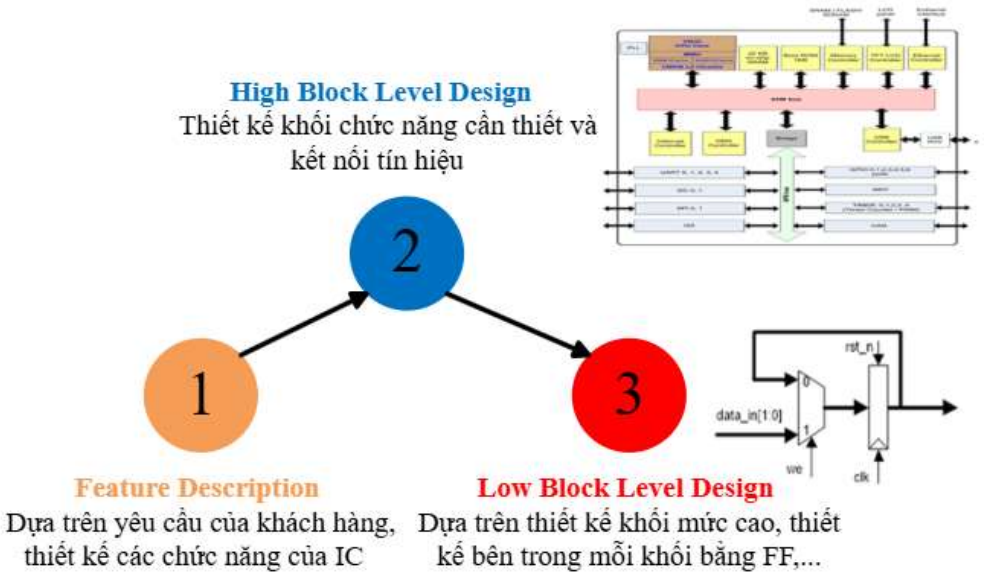
1.2. Quy trình thiết kế ASIC

Quy trình cơ bản để thiết kế một vi mạch số thường bao gồm các giai đoạn chính như đặc tả thiết kế (Design Specification), mô tả bằng RTL (Register Transfer Level Description), kiểm tra chức năng (Functional Verification), tổng hợp logic (Logic Synthesis), thiết kế vật lý (Physical Design), mô phỏng sau khi hoàn tất bố cục (Post Layout Simulation) và cuối cùng là sản xuất (Production).



1.2.1. Đặc tả thiết kế (Design Specification)

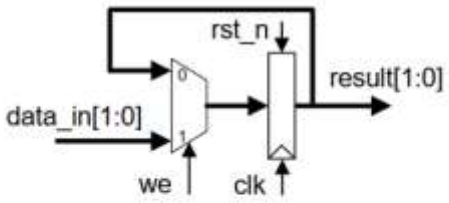
Dựa trên yêu cầu từ khách hàng, các kỹ sư tiến hành phân tích và xây dựng cấu trúc tổng thể, xác định giao tiếp và tín hiệu của Chip/IC thông qua giai đoạn Thiết kế Khối Cấp Cao (High Block Level Design). Sau đó, từng khối trong thiết kế này sẽ được chi tiết hóa bằng cách sử dụng các mạch logic cơ bản như AND, OR, XOR, NOT cùng với các phần tử như bộ chọn dữ liệu (MUX) và chốt D-flip flop trong giai đoạn thiết kế khối mức thấp (Low Block Level Design).



Hình 1.1: Các bước trong Design Specification

1.2.2. Mô tả RTL (RTL Description)

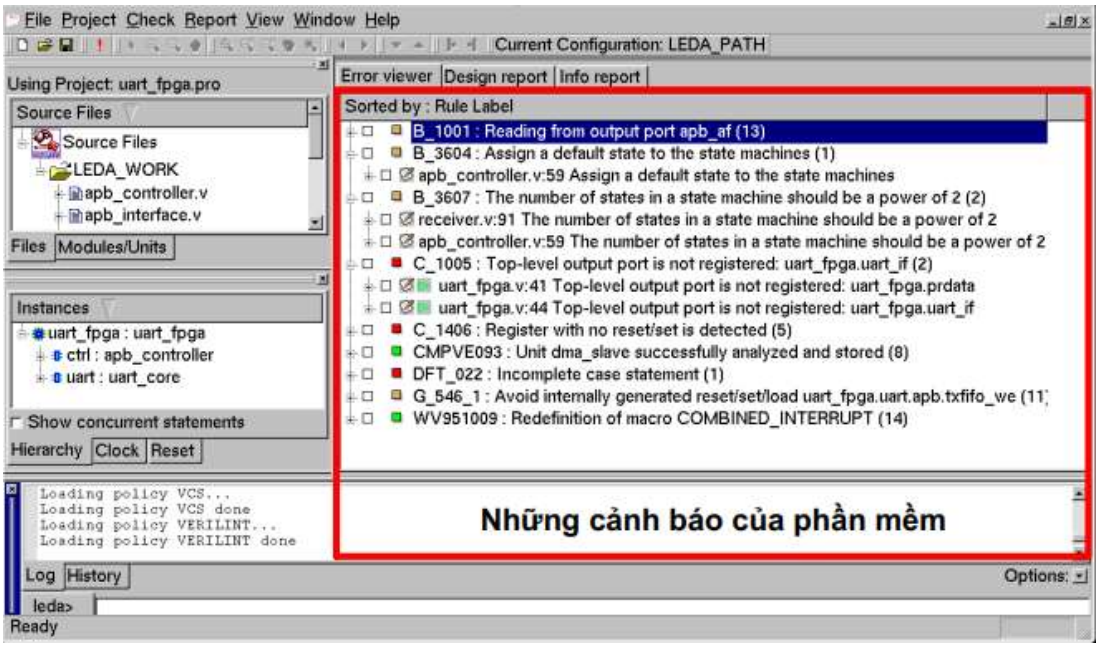
Mô tả RTL, hay còn gọi là RTL code, là quá trình sử dụng ngôn ngữ mô tả phần cứng (HDL) để biểu diễn thiết kế dựa trên các phân tích cấu trúc đã thực hiện trước đó. Hai ngôn ngữ phổ biến nhất trong lĩnh vực này là Verilog và VHDL. Việc viết mã RTL có thể thực hiện trên các trình soạn thảo tích hợp trong phần mềm thiết kế hoặc sử dụng các công cụ chỉnh sửa văn bản như Notepad++, EmEditor, VIM, Xemacs, conTEXT.




```
always @(posedge clk) begin
  if (!rst_n)  result[1:0] <= 2'b00;
  else if (we) result[1:0] <= data_in[1:0];
end
```

Hình 1.2: Một đoạn RTL code mô tả mạch nguyên lý

Sau khi hoàn thành mã RTL, thiết kế cần kiểm tra tính chính xác của mã theo các quy tắc về cú pháp, nguyên tắc thiết kế vi mạch số cũng như các quy định riêng của từng công cụ thiết kế. Việc kiểm tra này giúp phát hiện và khắc phục sớm các lỗi trước khi chuyển sang các giai đoạn tiếp theo. Một số phần mềm phổ biến có thể sử dụng gồm LEDA (Synopsys) và NC-Verilog (Cadence) dành cho thiết kế ASIC, hoặc Quartus II (Altera) và Vivado (Xilinx) dành cho thiết kế trên FPGA.



Hình 1.3: Minh họa những cảnh báo khi thực hiện kiểm tra RTL code dùng phần mềm LEDA (Synopsys)

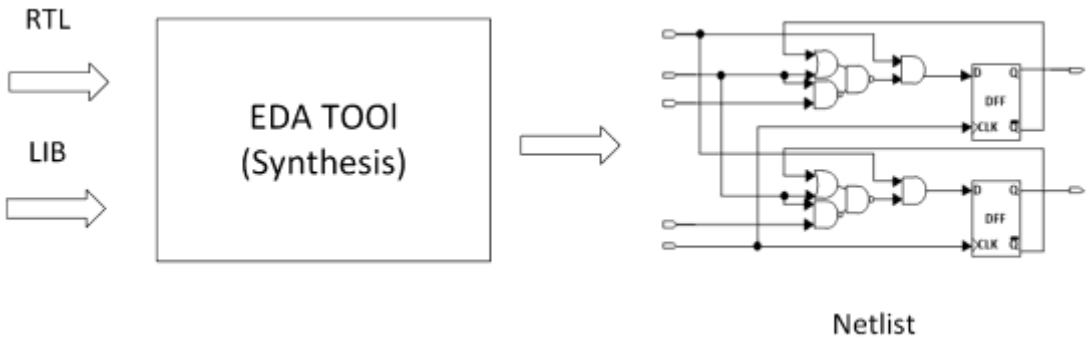
1.2.3. Xác minh chức năng (Functional Verification)

Sau khi hoàn thành việc viết mã, thiết kế sẽ trải qua giai đoạn xác minh để đảm bảo các chức năng hoạt động chính xác. Ở bước này, một môi trường kiểm thử sẽ được thiết lập dựa trên phương pháp luận tiêu chuẩn, sử dụng ngôn ngữ lập trình chuyên dụng như SystemVerilog. Các kịch bản và trường hợp kiểm thử sẽ được lập trình và thực thi trong môi trường đã xây dựng. Tất cả các bài kiểm thử được thiết kế theo quy chuẩn, đảm bảo tính tổng quát cao và bao phủ đầy đủ các chức năng của thiết kế. Nếu trong quá trình kiểm thử phát hiện lỗi hành vi, thiết kế sẽ quay lại giai đoạn mô tả RTL để sửa chữa. Quá trình xác minh chức năng chỉ hoàn tất khi tất cả lỗi đã được khắc phục và thiết kế đáp ứng đầy đủ yêu cầu đề ra.

1.2.4. Tổng hợp logic (Logic Synthesis)

Sau khi quá trình xác minh đã hoàn thành, thiết kế sẽ được chuyển qua bước tiếp theo đó là tổng hợp logic, trong đó netlist cấp cổng được tự động tạo ra từ mã RTL viết bằng các ngôn ngữ mô tả phần cứng như Verilog hoặc VHDL. Netlist là danh sách chứa các thành phần mạch, bao gồm các cổng logic như AND, OR, XOR, NOT, flip-flop và các kết nối giữa chúng, thể hiện cách các phần tử trong mạch hoạt động và tương tác với nhau. Netlist giúp mô tả hành vi của mạch mà không quan tâm đến cách bố trí vật lý.

Quá trình tổng hợp sử dụng các công cụ EDA chuyên dụng như Synopsys Design Compiler hoặc Cadence Genus. Các công cụ này nhận đầu vào là mã RTL cùng với thư viện LIB tương ứng, sau đó tạo ra netlist để phục vụ cho các bước thiết kế tiếp theo.

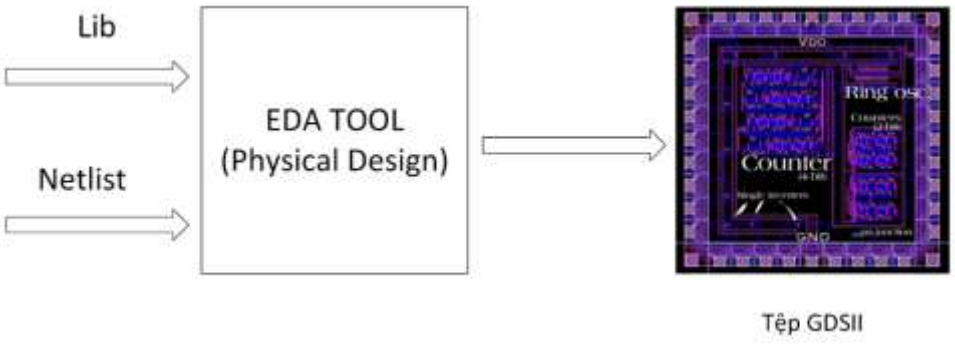


Hình 1.4: Quá trình tổng hợp tạo ra netlist

1.2.5. Thiết kế vật lý (Physical Design)

Thiết kế vật lý là giai đoạn chuyển đổi netlist (sơ đồ mạch logic) thành layout (bố cục vật lý), chuẩn bị cho quá trình chế tạo vi mạch. Netlist, bao gồm các cổng logic và kết nối, cùng với thư viện tiêu chuẩn (LIB) chứa thông tin về các thành phần, sẽ được nhập vào các công cụ thiết kế tự động hóa (EDA) như Cadence Innovus, Synopsys IC Compiler II hoặc Mentor Graphics IC Station.

Trong quá trình này, các bước quan trọng như lập kế hoạch sắp xếp (floorplanning), bố trí linh kiện (placement), tổng hợp cây xung nhịp (clock tree synthesis), và định tuyến kết nối (routing) sẽ được thực hiện để tạo ra một bố cục hoàn chỉnh. Sau đó, các kiểm tra như DRC (Design Rule Check) và LVS (Layout Versus Schematic) sẽ được tiến hành để đảm bảo layout tuân thủ các quy tắc thiết kế và khớp với netlist ban đầu. Kết quả cuối cùng của quá trình này là tệp GDSII, chứa bản vẽ chi tiết của chip.



Hình 1.5: Quá trình thiết kế vật lý

1.2.6. Mô phỏng sau bố cục (Post Layout Simulation)

Một bước quan trọng trong quá trình thiết kế vi mạch, được thực hiện sau khi hoàn thành giai đoạn thiết kế vật lý. Mục đích của giai đoạn này là kiểm tra và xác nhận rằng thiết kế vẫn hoạt động chính xác khi có sự tác động của các yếu tố vật lý thực tế như độ trễ dây dẫn và hiệu ứng ký sinh.

Quá trình này giúp phát hiện các lỗi thời gian (timing violations), đảm bảo tính

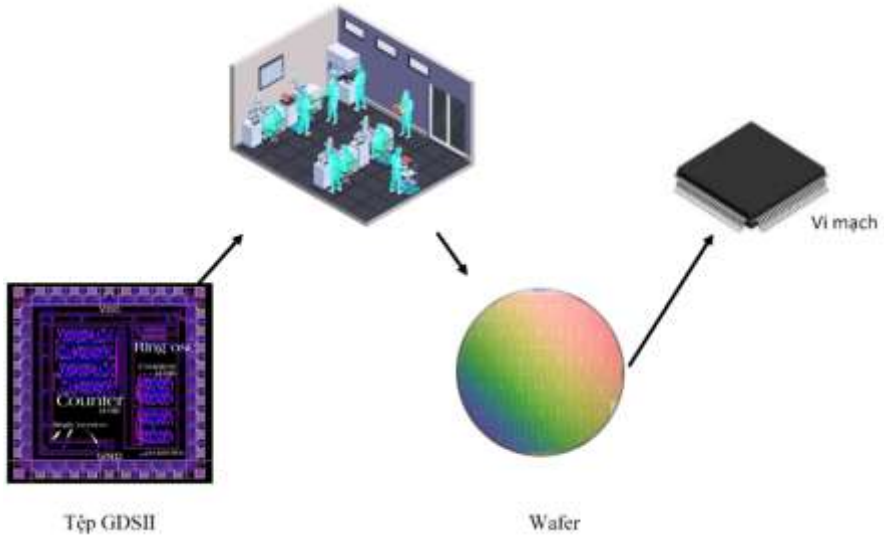
toàn vẹn tín hiệu và xác minh rằng thiết kế có thể hoạt động ổn định trong các điều kiện biên. Một trong những công đoạn quan trọng là trích xuất ký sinh (parasitic extraction) để tạo ra các tệp SPEF hoặc DSPF, sau đó sử dụng các tệp này để mô phỏng netlist có đầy đủ các thành phần điện trở và tụ điện ký sinh. Các loại mô phỏng phổ biến trong giai đoạn này bao gồm mô phỏng chức năng, mô phỏng thời gian, phân tích công suất và phân tích tính toàn vẹn tín hiệu.

Mặc dù post-layout simulation đòi hỏi nhiều thời gian, tài nguyên tính toán và xử lý dữ liệu phức tạp, nhưng đây là một bước không thể bỏ qua nhằm đảm bảo vi mạch hoạt động chính xác và ổn định trước khi đưa vào sản xuất.

1.2.7. Sản xuất (Production)

Quá trình sản xuất vi mạch bắt đầu từ việc sử dụng tệp GDSII, kết quả của quá trình thiết kế bố cục mạch. Tệp GDSII chứa thông tin chi tiết về các lớp và cấu trúc vật lý của chip, được dùng để tạo ra mask phục vụ cho quá trình quang khắc (photolithography).

Tiếp theo, wafer silicon được chuẩn bị và trải qua một loạt các bước xử lý như oxy hóa, quang khắc, khắc (etching), lắng đọng vật liệu và cấy ion để hình thành các lớp chức năng của vi mạch. Sau khi hoàn tất, wafer sẽ được cắt thành từng chip riêng lẻ, đóng gói và kiểm tra chất lượng nhằm đảm bảo sản phẩm đạt tiêu chuẩn trước khi tiến hành sản xuất hàng loạt.



Hình 1.6: Quá trình sản xuất vi mạch

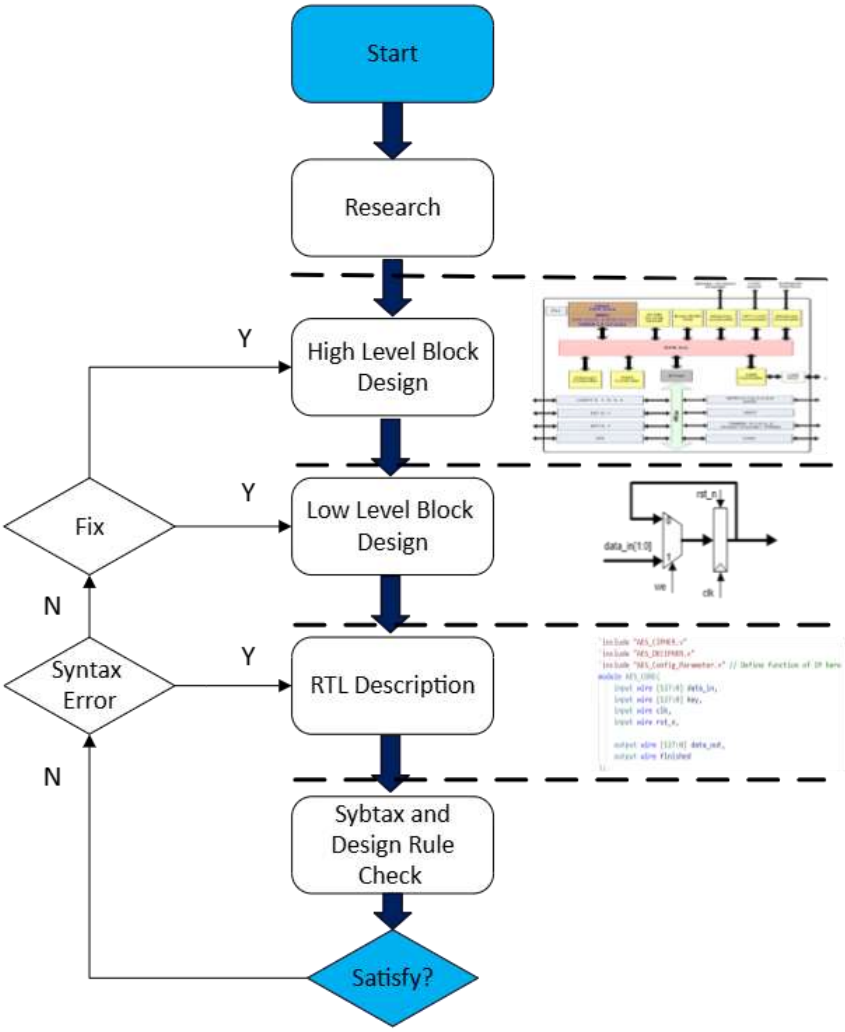
1.3. Quy trình thiết kế ASIC Front-End

1.3.1. Thiết kế đặc tả (Design Specification)

Giai đoạn thiết kế front-end ban đầu bao gồm các bước: Specification Design, RTL

Description và Functional Verification. Trong thực tế, kỹ sư đảm nhiệm phần Specification Design thường trực tiếp viết mã RTL, vì vậy có thể gộp hai bước này lại thành một phần gọi là Design Specification.

Mô tả mã nguồn là quá trình sử dụng ngôn ngữ mô tả phần cứng để diễn tả thiết kế dựa trên các phân tích cấu trúc đã thực hiện trước đó. Ngôn ngữ phổ biến cho việc này bao gồm Verilog và VHDL, cho phép biểu diễn hành vi và cấu trúc của mạch số theo cách có thể tổng hợp được thành phần cứng thực tế.



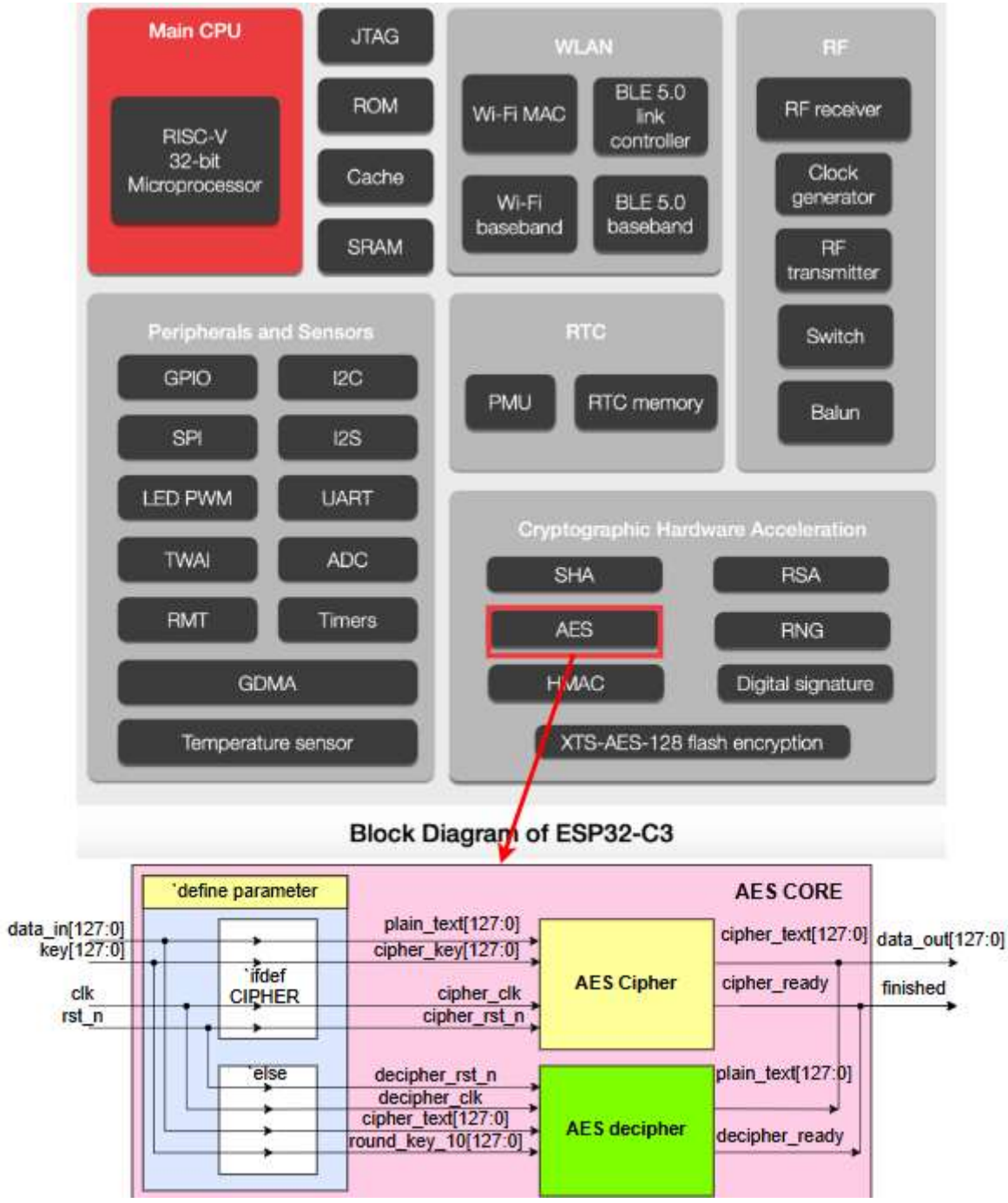
Hình 1.7: Các bước phân tích thiết kế vi mạch số

Nghiên cứu thị trường (Research): Từ những yêu cầu thiết kế của khách hàng, kỹ sư thiết kế thực hiện tìm hiểu và nghiên cứu đến các yêu cầu của khách hàng. Các vấn đề này có thể là ứng dụng của thiết kế, nguyên lý hoạt động, năng lượng tiêu thụ…, thông qua Internet, bài báo, sách vở. Bước này cho phép kỹ sư nắm rõ các vấn đề liên quan đến thiết kế để đưa ra các đánh giá, so sánh và đưa ra các hướng thiết kế tốt nhất đối với yêu cầu đặt ra.

Thiết kế khối mức cao (High Level Block Design): Là xây dựng các vấn đề và

các cấu trúc ban đầu của thiết kế. Ở bước này, kỹ sư phải xác định được các yếu tố sau đây: mô tả chức năng là xác định được các tính năng mà thiết kế sẽ hỗ trợ từ những yêu cầu thiết kế; giao tiếp là xác định các tín hiệu giao tiếp với các thiết bị ngoại vi khác hoặc CPU, các tín hiệu này có thể là ngỏ vào, ngỏ ra hoặc tín hiệu hai chiều; mô tả các tín hiệu là kĩ sư thiết kế sẽ cung cấp thông tin về chức năng cụ thể của từng tín hiệu giao tiếp, độ rộng, tính đồng bộ hay bất đồng bộ của tín hiệu,…; thiết kế là tại bước này, kĩ sư sẽ thiết kế các khối chức năng bên trong của thiết kế sao cho phù hợp với các tính năng đã đề ra ở mục mô tả chức năng; dạng sóng là vẽ giản đồ định thời các tín hiệu giao tiếp, ở đây là các xung là lý tưởng có thể chưa giống với dạng sóng sau khi thiết kế hoàn chỉnh nhưng dạng sóng này đúng với mong muốn của người thiết kế; các thanh ghi là các thanh ghi cấu hình hoạt động, các thanh ghi thể hiện trạng thái hoạt động, thanh ghi dữ liệu,… của thiết kế mà người sử dụng có thể truy cập được.

Bước thiết kế khối mức cao có thể áp dụng cho nhiều tầng thiết kế. Trong một chip vi điều khiển sẽ có nhiều khối chức năng, bước này được áp dụng cho toàn bộ chip. Như trong STM32L4 + LoRa thì dữ liệu cảm biến gửi qua mạng LoRa được mã hóa AES-128 để tránh bị nghe lén. Hay trong ESP32-C3 là một vi điều khiển có kết nối wifi để bảo vệ phần cứng, thường giao tiếp với các Server và cloud, AES giúp mã hóa và giải mã an toàn thông tin nhận được. Đề thiết kế được lõi IP mềm cho mã hóa AES tích hợp bên trong vi điều khiển, người thiết kế sẽ thiết kế các khối chức năng bên trong lõi. Kết thúc thiết kế khối mức cao, người thiết kế có thể hình dung tổng thể toàn bộ thiết kế và là cơ sở để thực hiện bước tiếp theo là thiết kế khối mức thấp (Low level block design).



Hình 1.8: Ứng dụng AES-128 trong vi điều khiển ESP32-C3

Thiết kế khối mức thấp: trong giai đoạn này, thiết kế cần mô tả cụ thể cách từng khối chức năng hoạt động dựa trên mô hình đã được xác định trước đó. Điều này giúp đảm bảo sự chính xác về mặt logic và cấu trúc trước khi chuyển sang bước viết mã RTL. Sử dụng các cổng AND, OR, XOR, MUX,... để xây dựng cấu trúc mạch. Cung cấp thông tin chi tiết về chức năng và yêu cầu vận hành của từng khối. Bước này giúp định hình rõ ràng kiến trúc thiết kế, tạo tiền đề cho quá trình hiện thực hóa bằng ngôn ngữ mô tả phần cứng.

Mô tả RTL : Giai đoạn này yêu cầu sử dụng ngôn ngữ mô tả phần cứng như Verilog

hoặc VHDL để biểu diễn thiết kế ở mức logic cổng. Các công cụ hỗ trợ viết và kiểm tra mã RTL bao gồm Synopsys Leda và Modelsim.

Kiểm tra cú pháp và quy tắc thiết kế (Syntax and Design Rule Check): Sau khi hoàn thành, mã RTL cần được kiểm tra bằng các công cụ chuyên dụng như Modelsim hoặc Synopsys Leda để phát hiện lỗi cú pháp và vi phạm quy tắc thiết kế. Nếu xuất hiện cảnh báo hoặc lỗi, có thể chỉnh sửa mã RTL hoặc quay lại điều chỉnh thiết kế ở mức khối thấp hoặc cao hơn để đảm bảo hệ thống hoạt động đúng theo yêu cầu.

1.3.2. Xác minh chức năng (Functional Verification)

Xác minh chức năng là một giai đoạn quan trọng trong quá trình thiết kế vi mạch số ASIC, nhằm đảm bảo rằng thiết kế đáp ứng đầy đủ các yêu cầu được đặt ra trong tài liệu đặc tả. Mục tiêu chính của bước này là phát hiện và khắc phục lỗi trước khi sản xuất, qua đó nâng cao độ tin cậy của sản phẩm, hướng đến chip hoạt động chính xác ngay lần chế tạo đầu tiên. Ngoài ra, việc thực hiện xác minh hiệu quả còn giúp tối ưu thời gian phát triển và giảm số lần chỉnh sửa thiết kế, góp phần rút ngắn chu kỳ phát triển tổng thể.

Quy trình này bao gồm nhiều giai đoạn, mỗi giai đoạn đóng vai trò quan trọng trong việc xác nhận tính chính xác của thiết kế. Dưới đây là các bước cụ thể dựa trên sơ đồ quy trình.:

Nghiên cứu tài liệu đặc tả (Research Specification): Bước đầu tiên trong quy trình là phân tích và nghiên cứu tài liệu đặc tả kỹ thuật của thiết kế. Đây là nền tảng giúp nhóm xác minh hiểu rõ các yêu cầu chức năng, giao thức, cũng như điều kiện vận hành của hệ thống. Đặc tả kỹ thuật thường bao gồm chi tiết về giao diện (interfaces), trạng thái hệ thống, và hành vi mong đợi. Việc nắm vững các yêu cầu này giúp xây dựng các trường hợp kiểm thử phù hợp, đảm bảo rằng mọi khía cạnh của thiết kế đều được đánh giá đúng. Nếu hiểu sai hoặc bỏ sót bất kỳ điểm nào trong đặc tả, quá trình xác minh có thể đi sai hướng, dẫn đến việc bỏ lỡ lỗi hoặc phát hiện lỗi không chính xác.

Lập kế hoạch xác minh (Verification Planning): Sau khi nghiên cứu kỹ đặc tả kỹ thuật, nhóm xác minh tiến hành xây dựng kế hoạch kiểm thử chi tiết. Kế hoạch này xác định rõ các chức năng cần kiểm tra, phạm vi bao phủ mong muốn và các điều kiện xác nhận tính chính xác của thiết kế. Các tính năng cần kiểm tra, xác định danh sách các chức năng cốt lõi của thiết kế, bao gồm giao diện truyền thông, xử lý dữ liệu và các luồng dữ liệu quan trọng. Mục tiêu bao phủ, đảm bảo kiểm tra toàn diện thông qua các tiêu chí như độ bao phủ mã (code coverage), độ bao phủ chức năng (functional coverage) và độ bao phủ trạng thái. Điều này giúp phát hiện các kịch bản hoạt động chưa được kiểm tra.

Xác định khả năng sử dụng VIP (Verification IP): Ở bước này, kiểm tra xem xét khả năng sử dụng các gói Verification IP có sẵn hoặc cần phát triển môi trường kiểm thử mới. Nếu có VIP, các gói VIP giúp tiết kiệm thời gian nhờ tích hợp sẵn các giao thức tiêu chuẩn, cung cấp driver, monitor và scoreboard để hỗ trợ kiểm tra tự động. Nếu không có VIP, nhóm sẽ phải tự xây dựng môi trường xác minh, bao gồm tạo driver để mô phỏng dữ liệu đầu vào, thiết lập monitor để thu thập dữ liệu đầu ra, và phát triển scoreboard để so sánh kết quả với mô hình tham chiếu.

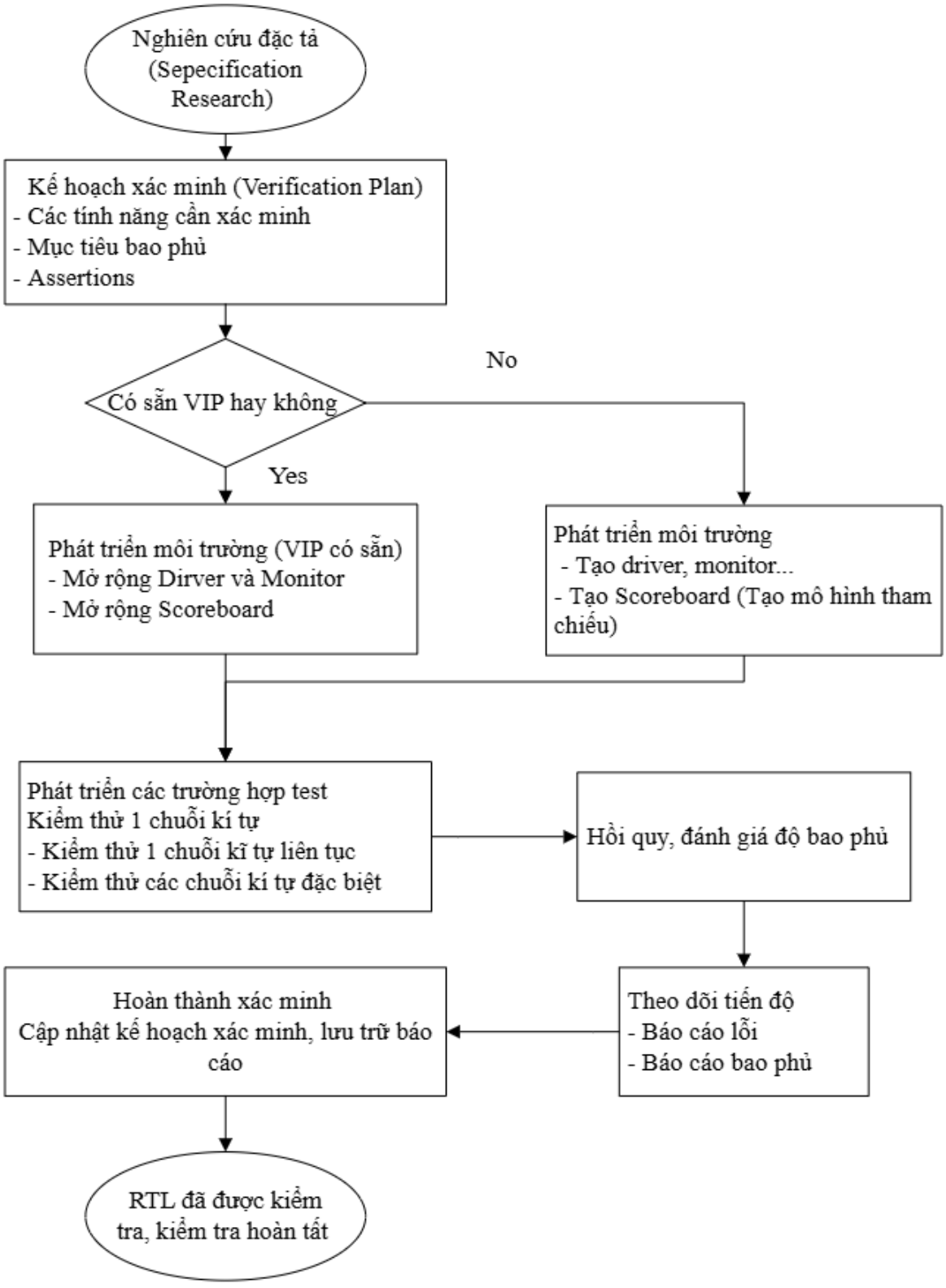
Chạy kiểm thử và hồi quy (Regression Testing): Sau khi xây dựng đầy đủ môi trường kiểm thử, các bài test sẽ được chạy trên thiết kế để kiểm tra tính chính xác. Kiểm thử ban đầu, đánh giá các chức năng cơ bản để phát hiện lỗi sớm. Lặp lại các bài kiểm tra theo chu kỳ để đảm bảo rằng những thay đổi trong thiết kế không gây ảnh hưởng đến các tính năng đã được xác nhận trước đó.

Theo dõi tiến độ và phân tích kết quả: Trong suốt quá trình kiểm thử, đội ngũ sẽ giám sát và phân tích kết quả để phát hiện lỗi và cải tiến thiết kế. Báo cáo lỗi, mức độ bao phủ và các điểm cần tối ưu sẽ được ghi nhận để điều chỉnh chiến lược kiểm thử.

Hoàn thiện quy trình kiểm thử (Verification Closure): Khi tất cả các lỗi đã được khắc phục và mục tiêu bao phủ đã đạt yêu cầu, quy trình kiểm thử được hoàn tất. Kế hoạch xác minh và kết quả kiểm thử sẽ được lưu trữ để làm tài liệu tham khảo.

Xác minh RTL hoàn chỉnh (Verified RTL): Thiết kế RTL sau khi được kiểm chứng đầy đủ sẽ sẵn sàng cho các bước tiếp theo như tổng hợp (synthesis) và kiểm tra vật lý.

Quy trình Functional Verification là một chuỗi công việc chặt chẽ, từ việc nghiên cứu tài liệu, lập kế hoạch, phát triển môi trường kiểm thử, thực hiện hồi quy, theo dõi tiến độ đến hoàn tất xác minh. Dưới đây là lưu đồ xác minh chức năng:



Hình 1.9: Lưu đồ xác minh chức năng

1.4. Kết luận

Kết thúc chương này sẽ nắm được định nghĩa về VLSI, một quy trình được sử dụng để tích hợp một số lượng lớn bóng bán dẫn (transistor) trên một vi mạch đơn, tìm hiểu

về quy trình thiết kế vi mạch, từ việc mô tả kiến trúc cấp cao đến việc xác minh chức năng thông qua các ngôn ngữ phần cứng như Verilog và VHDL.

Các bước thiết kế logic và thiết kế vật lý cũng đã được mô tả, giúp định hình chip trước khi đưa vào sản xuất. Chương này kết thúc với những kiến thức nền tảng và chi tiết về thiết kế VLSI và ASIC, đóng vai trò quan trọng trong việc phát triển các vi mạch tích hợp cho các ứng dụng thực tế.

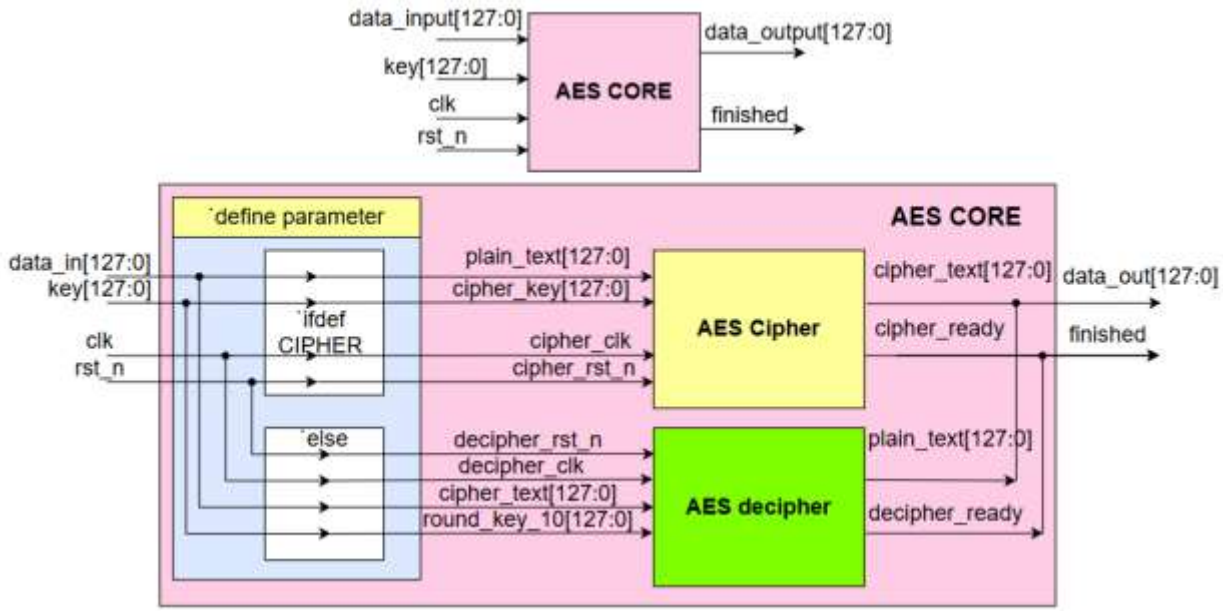
Chương 2: GIỚI THIỆU VỀ AES-128

2.1. Giới thiệu chương

AES (Advanced Encryption Standard) là một thuật toán mã hóa khối đối xứng được chuẩn hóa bởi Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST) vào năm 2001. AES thay thế thuật toán DES (Data Encryption Standard) trước đó do khả năng bảo mật cao hơn. AES có các biến thể khác nhau dựa trên độ dài khóa: AES-128, AES-192, AES-256, trong đó AES-128 là phiên bản được sử dụng rộng rãi nhất nhờ vào hiệu suất cao và độ bảo mật phù hợp cho nhiều ứng dụng.

AES-128 là một thuật toán mã hóa khối, hoạt động trên các khối dữ liệu có kích thước 128-bit và sử dụng khóa 128-bit. Quá trình mã hóa và giải mã của AES-128 bao gồm nhiều vòng lặp biến đổi dữ liệu dựa trên phép toán đại số và logic.

Đồ án sẽ thiết kế lõi IP mềm thực hiện 2 chức năng mã hóa hoặc giải mã, lõi IP (IP core) sẽ mặc định là thực hiện chức năng mã hóa, thì chức năng giải mã sẽ bị vô hiệu hóa. Khi cần thực hiện chức năng giải mã thì sẽ cấu hình cho lõi IP thực hiện chức năng giải mã, chức năng mã hóa sẽ bị vô hiệu hóa.



Hình 2.1: Lõi IP thực hiện 2 chức năng mã hóa và giải mã

Lõi IP khi thực hiện một trong hai chức năng thì các tín hiệu sẽ được cấu hình để kết nối như bên bảng bên dưới:

Bảng 2.1: Nguyên lý cầu hình chân dữ liệu đầu vào ra của lõi

Chức năng	Cấu hình tham số
Mã hóa	data_in[127:0] = plain_text[127:0]
	key[127:0] = cipher_key[127:0]
	clk = cipher_clk
	rst_n = cipher_rst_n
	data_out[127:0] = cipher_text[127:0]
	finished = cipher_ready
Giải mã	data_in[127:0] = cipher_text[127:0]
	key[127:0] = round_key_10[127:0]
	clk = decipher_clk
	rst_n = decipher_rst_n
	data_out[127:0] = plain_text[127:0]
	finished = decipher_ready

2.2. Chức năng chính của lõi AES

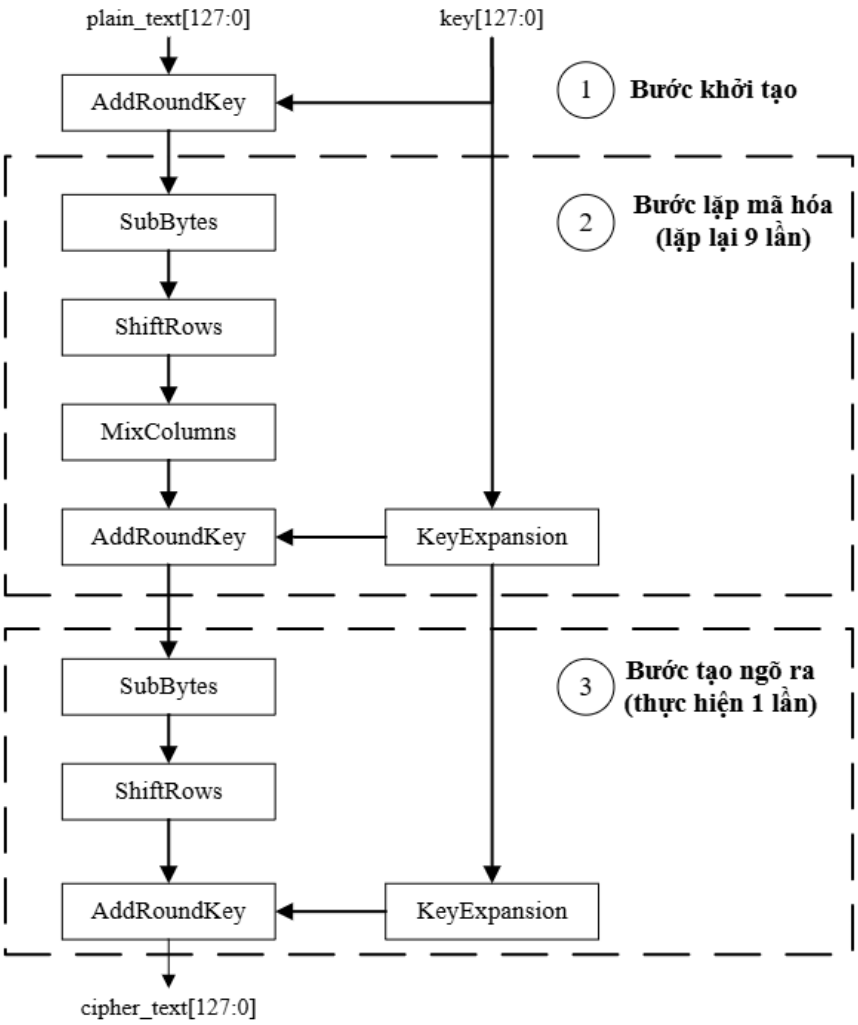
2.2.1. Chức năng mã hóa

Mã hóa AES được thực hiện thông qua 5 chức năng chính là AddRoundKey, SubBytes, ShiftRows, MixColumns và KeyExpansion. Năm chức năng này được sắp xếp để thực hiện ba bước cơ bản.

Bước 1. Bước khởi tạo: dữ liệu cần được mã hóa plain_text[127:0] kết hợp với key[127:0] bằng chức năng AddRoundKey

Bước 2. Bước lặp mã hóa: kết quả bước 1 được sử dụng để thực hiện tuần tự các chức năng SubBytes, ShiftRows, MixColumns và AddRoundKey. Bước này được lặp lại 9 lần. Chú ý, KeyExpansion thực hiện song song với bước AddRoundKey để tạo khóa vòng cho chức năng này. Chức năng SubBytes áp dụng một bảng thay thế (S-Box) để thay thế từng byte dữ liệu nhằm tăng tính phi tuyến tính. Chức năng ShiftRows sẽ dịch vòng các hàng của ma trận dữ liệu để tạo sự phân tán. Sau đó chức năng MixColumns sẽ kết hợp các byte trong từng cột bằng phép nhân ma trận trong trường hữu hạn GF(2^8). Và cuối cùng là AddRoundKey sẽ thực hiện phép XOR giữa dữ liệu và khóa con (derived key) của từng vòng.

Bước 3. Bước tạo ngõ ra: Sau 9 lần lặp ở bước 2, kết quả được sử dụng để thực hiện tuần tự các chức năng SubBytes, ShiftRows và AddRoundKey để tạo ngõ ra cipher_text[127:0].



Hình 2.2: Quá trình mã hóa AES-128

2.2.2. Chức năng giải mã

Mã hóa chuyển một "chuỗi dữ liệu gốc" (plaintext) thành một "chuỗi dữ liệu được mã hóa" (ciphertext) thông qua một khóa mã (key) giúp che dấu thông tin gốc ban đầu. Giải mã là quá trình nghịch đảo (Inverse cipher) của quá trình mã hóa. Nó giúp khôi phục lại plaintext từ một ciphertext.

Trong quá trình giải mã, ma trận ciphertext sẽ bị biến đổi bởi các chức năng để tạo ra các dữ liệu trung gian gọi là ma trận trạng thái. Ma trận khóa mã sẽ bị biến đổi bởi chức năng KeyExpansion như trong quá trình mã hóa. Tuy nhiên, thứ tự sử dụng các khóa vòng trong quá trình giải mã ngược với quá trình mã hóa, nghĩa là khóa vòng số 10 sẽ được sử dụng đầu tiên. Tiếp theo đó là khóa vòng số 9, số 8, ..., cuối cùng là khóa mã gốc.

Quá trình giải mã được thực hiện qua 5 chức năng là AddRoundKey, InvSubBytes, InvShiftRows, InvMixColumns và InvKeyExpansion. Chú ý, InvKeyExpansion không phải là một chức năng được mô tả trong chuẩn mà là một tên gọi được thêm vào để chỉ

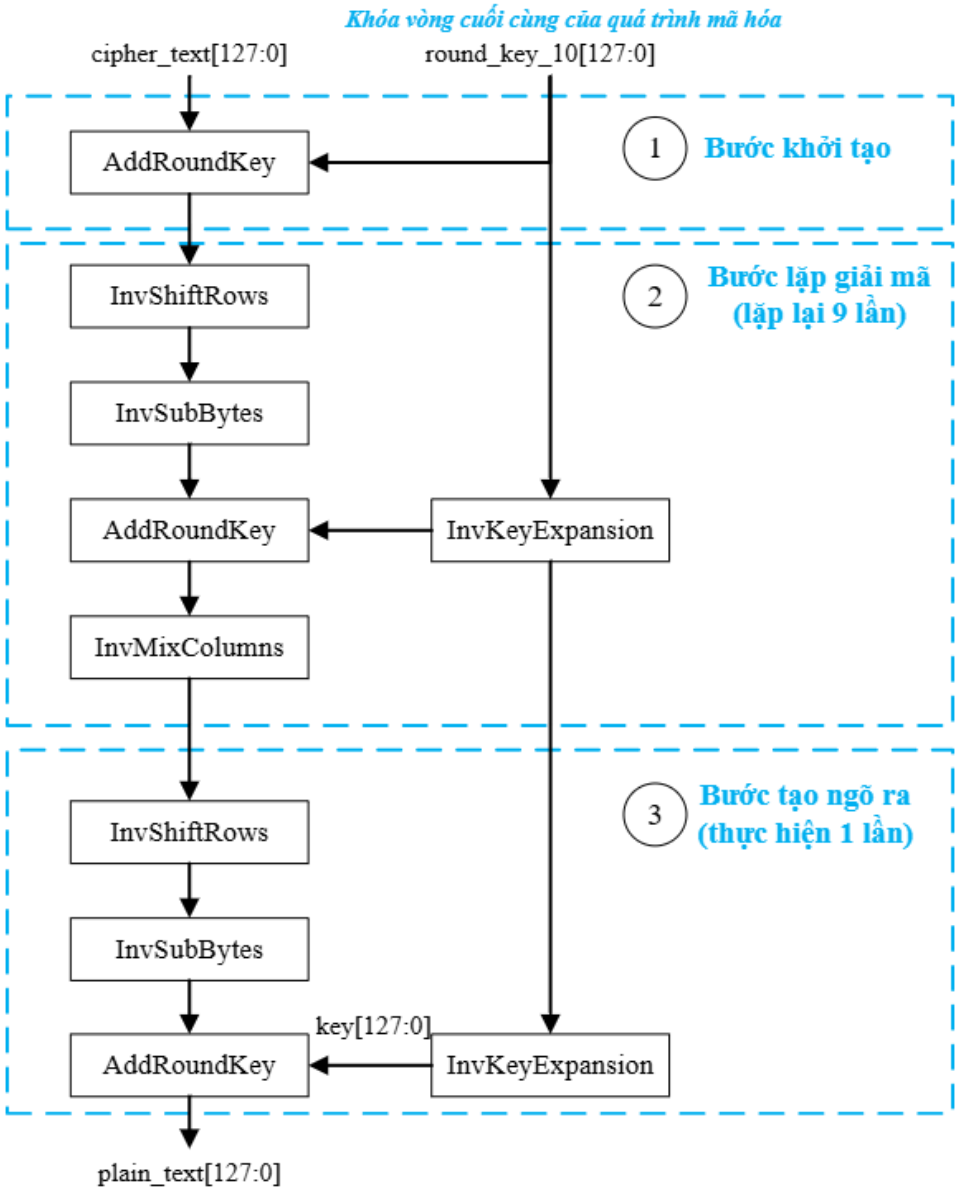
quá trình biến đổi ngược giá trị khóa vòng từ khóa vòng số 10 đến khóa mã gốc.

Giống với mã hóa, chức năng giải mã cũng có 3 bước:

Bước 1. Bước khởi tạo: Dữ liệu cần được giải mã cipher_text[127:0] kết hợp với khóa vòng thứ 10, round_key_10[127:0], bằng chức năng AddRoundKey

Bước 2. Bước lặp giải mã: kết quả bước 1 được sử dụng để thực hiện tuần tự các chức năng InvShiftRows, InvSubBytes, AddRoundKey và InvMixColumns. Bước này được lặp lại 9 lần. Chức năng InvKeyExpansion sẽ thực hiện song song với bước AddRoundKey để tạo khóa vòng cho chức năng này.

Bước 3. Bước tạo ngõ ra: Sau 9 lần lặp ở bước 2, kết quả được sử dụng để thực hiện tuần tự các chức năng InvShiftRows, InvSubBytes và AddRoundKey với khóa mã ban đầu để tạo ra plain_text[127:0].



Hình 2.3: Quá trình giải mã

2.3. Ứng dụng của AES-128

AES-128 được sử dụng rộng rãi trong nhiều lĩnh vực do tính bảo mật cao và hiệu suất tốt. Các ứng dụng phổ biến như : bảo mật dữ liệu trong mạng, sử dụng trong giao thức TLS/SSL (Transport Layer Security/ Secure Sockets Layer) để bảo vệ thông tin trên Internet; mã hóa ổ đĩa và thiết bị lưu trữ, được sử dụng trong BitLocker, VeraCrypt và các hệ thống mã hóa khác; giao tiếp không dây an toàn, dùng trong WPA2 để bảo mật Wi-Fi và ứng dụng nhúng và IoT, bảo mật dữ liệu trong các thiết bị thông minh.

2.4. Kết luận

AES-128 là một thuật toán mã hóa mạnh mẽ, đáng tin cậy và đã được chuẩn hóa trên toàn cầu. Với kiến trúc chặt chẽ và khả năng bảo mật cao, nó trở thành một lựa chọn hàng đầu trong bảo mật dữ liệu. Trong các hệ thống SoC (System-on-Chip), AES-128 thường được tích hợp như một IP core để tăng tốc quá trình mã hóa và giải mã trong phần cứng.

Chương 3: THIẾT KẾ LÕI IP MỀM CHUẨN AES-128

3.1. Giới thiệu chương

Thiết kế lõi IP mềm AES-128 cần đảm bảo đáp ứng các yêu cầu về hiệu suất, độ trễ, tài nguyên phần cứng và khả năng tích hợp vào hệ thống SoC. Chương này tập trung vào việc xây dựng đặc tả thiết kế của lõi IP, bao gồm các yêu cầu kỹ thuật, kiến trúc hệ thống, giao diện phần cứng, và cơ chế hoạt động của từng khối chức năng.

Các nội dung chính được đề cập trong chương bao gồm:

1. Xác định các thông số kỹ thuật của lõi AES-128.
2. Phân tích kiến trúc phần cứng, thiết kế khối mức cao, thiết kế khối mức thấp
3. Thiết kế giao diện kết nối giữa các khối bên trong lõi

Cấu trúc của các mô-đun chính, bao gồm bộ tạo khóa vòng cho chức năng mã hóa (Key Expansion), bộ tạo khóa con cho chức năng giải mã (Inv Key Expansion), khối mã hóa (Encryption Core) và khối giải mã (Decryption Core)

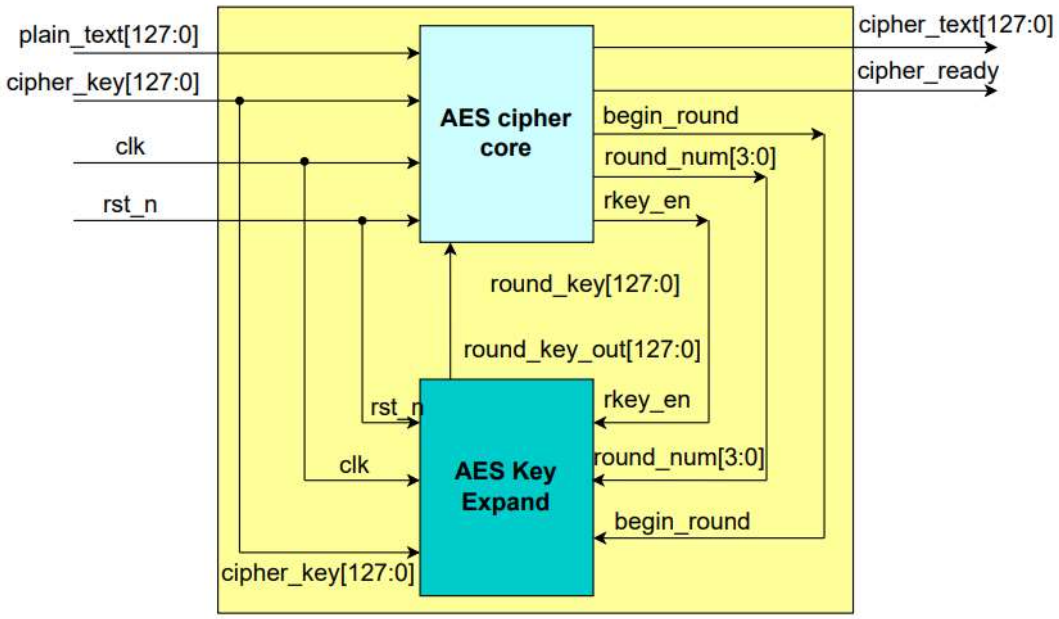
Những đặc tả thiết kế này là cơ sở để triển khai mô hình phần cứng AES-128 bằng ngôn ngữ mô tả phần cứng (HDL) và kiểm thử tính đúng đắn của lõi IP trong môi trường giả lập.

3.2. Thiết kế lõi IP mềm phần mã hóa

Sau khi nghiên cứu kỹ về thông số kỹ thuật, hiểu rõ về thuật toán, thì chúng ra sẽ bắt đầu triển khai tới phần tiếp theo của quy trình. Hình 1.3 là lõi IP mềm thực hiện chức năng mã hóa. Dữ liệu cần được mã hóa plain_text và khóa cipher_key là đầu vào, để tạo các khóa vòng cho dữ liệu, khối tạo khóa sẽ hoạt động song song với hoạt động của khối mã hóa. Sau khi biến đổi qua các vòng thì sẽ đưa dữ liệu đầu ra cipher_text là dữ liệu đã được mã hóa. Để hiểu rõ hơn về từng chi tiết bên trong của lõi, các phần tiếp theo sẽ trình bày rõ hơn về thiết kế logic mức cao và mức thấp.

3.2.1. Thiết kế khối mức cao (High Level Block Design)

Lõi IP mềm AES chức năng mã hóa, AES Cipher được thiết kế gồm 2 khối: Khối mã hóa dữ liệu AES cipher core và khối tính toán tạo khóa vòng AES Key Expand.



Hình 3.1: Lõi bên trong của AES Cipher

Các tín hiệu vào ra và kết nối nội bên trong của AES Cipher:

Bảng 3.1: Các tín hiệu vào ra của AES Cipher

Tín hiệu	**Mô tả**
clk	clock đồng bộ
rst_n	reset tích cực mức thấp
cipher_key[127:0]	khóa mã
plain_text[127:0]	dữ liệu cần mã hóa (plaintext)
cipher_ready	báo trạng thái bộ mã hóa sẵn sàng hoạt động. Tín hiệu này chỉ bằng 0 khi bộ mã hóa đang trong quá trình mã hóa dữ liệu
cipher_text[127:0]	dữ liệu sau khi được mã hóa (ciphertext)

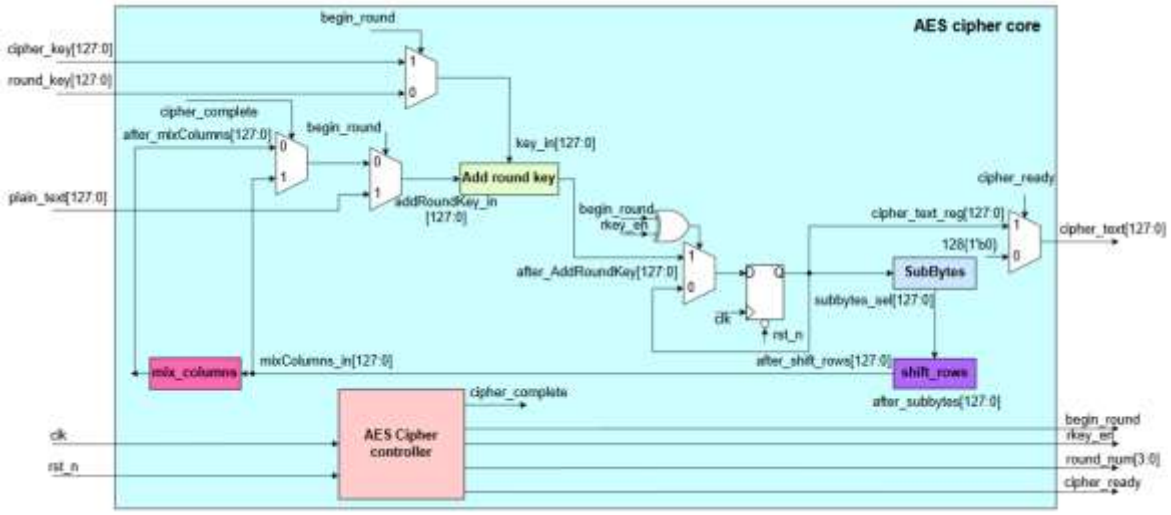
Bảng 3.2: Các tiến hiệu giao tiếp giữa AES cipher core và AES Key Expand

Tín hiệu	**Mô tả**
rkey_en	tín hiệu cho phép khối AES Key Expand hoạt động
round_num[3:0]	tín hiệu báo vòng lặp mã hóa hiện tại, round_num[3:0]=0 là bước khởi tạo đầu tiên.
round_key_out[127:0]	là giá trị khóa vòng từ khối AES Key Expand gửi cho khối AES cipher core
begin_round	tín hiệu bắt đầu mã hóa, begin_round=1 khi round_num[3:0]=0

3.2.2. Thiết kế khối mức thấp (Low Level Block Design)

Sau khi hoàn thành thiết kế khối mức cao, lõi thiết kế hiện rõ cấu trúc tổng thể, các thành phần chính và cách chúng tương tác. Phần này sẽ đi sâu vào bên trong của các khối mức cao để tiếp tục phân tích và thiết kế các thành phần bên trong nó, điều này giúp chi tiết hóa hệ thống và chuẩn bị cho việc viết mã mô tả.

Khối AES cipher core sẽ được thiết kế chi tiết hơn, từ những chức năng của thuật toán, việc thực thi sẽ được thực hiện theo thứ tự các chức năng, đường đi của dữ liệu và thiết kế tổng thể của lõi mã hóa như hình ảnh bên dưới, sau đó sẽ đi vào chi tiết cụ thể để mô tả từng chức năng.

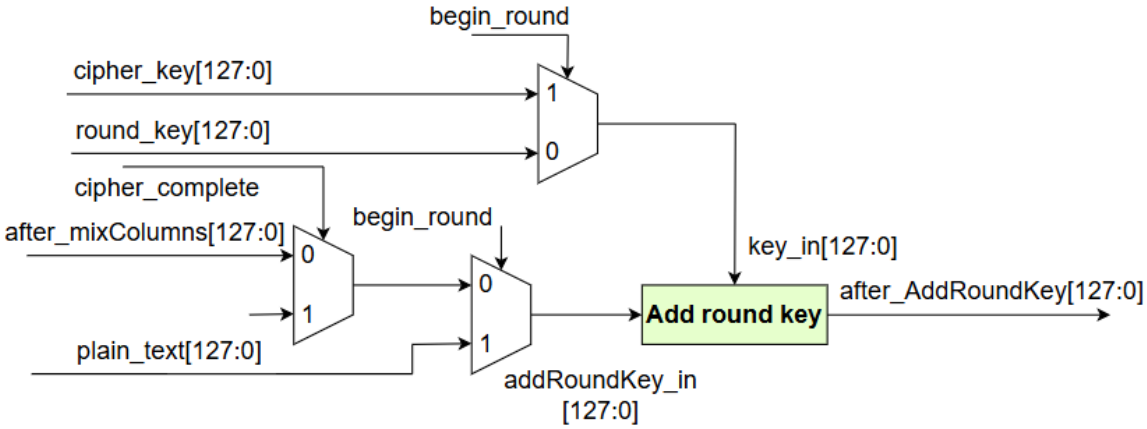


Hình 3.2: Thiết kế mức thấp lõi mã hóa AES

Thiết kế khối mức thấp cho lõi mã hóa:

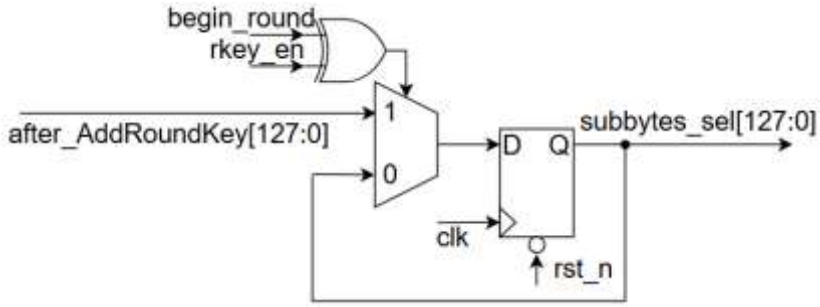
Sau khi đưa dữ liệu cần mã hóa và khóa vào lõi IP, do bộ mã hóa chỉ lấy dữ liệu đưa vào làm đầu vào ở vòng khởi tạo đầu tiên, tiếp theo nó sẽ lấy dữ liệu của kết quả biến đổi vòng trước đó làm đầu vào, nên phải dùng bộ MUX để chọn dữ liệu đưa vào bộ AddRoundkey.

Ban đầu, khi đưa dữ liệu và khóa vào, tín hiệu begin_round sẽ bằng 1, bộ MUX sẽ bắt và đưa plain_text và cipher_key làm dữ liệu đầu vào cho vòng khởi tạo, đưa vào AddRoundKey. Tới vòng tiếp theo, tín hiệu begin_round sẽ bằng 0, dữ liệu đầu vào của vòng đầu tiên sẽ là kết quả đầu ra của vòng khởi tạo, tương tự như thế, kết quả đầu ra của vòng này sẽ là đầu vào của vòng kế tiếp, nó sẽ biến đổi qua 10 vòng để tạo kể quả đầu ra.



Hình 3.3: Khối bắt dữ liệu đầu vào cho lõi IP mã hóa

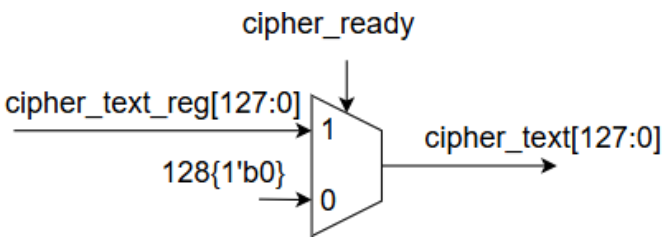
Dữ liệu sau khi AddRoundKey thì sẽ được đưa vào SubBytes. Sau mỗi vòng biến đổi tương ứng với 1 chu kì xung clk, do đó sẽ sử dụng Flip flop để chốt dữ liệu cho tới chu kì tiếp theo mới thay đổi, đảm bảo lõi mã hóa hoạt động đúng, bắt dữ liệu chính xác từng thời điểm. Bên cạnh đó, có thêm tín hiệu rkey_en, tín hiệu này cho phép bộ mã hóa hoạt động ở chu kì xung clk đầu tiên.



Hình 3.4: Dùng FF để chốt dữ liệu sau mỗi vòng

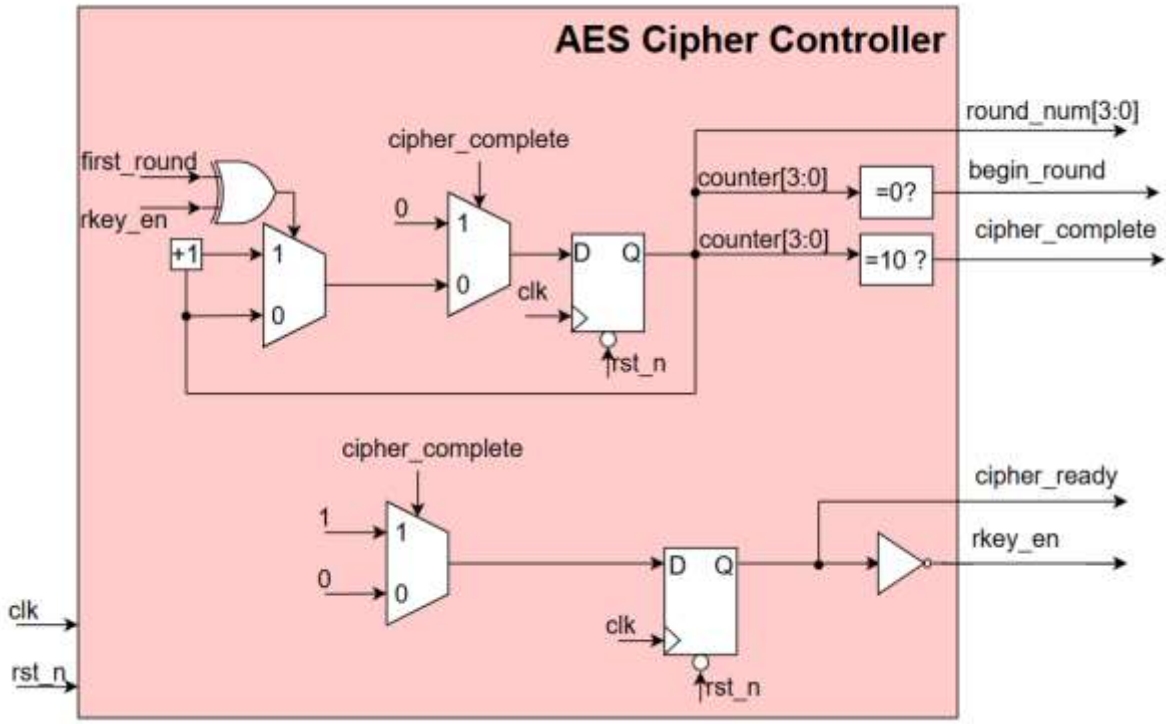
Dữ liệu đầu ra sau Add round key sẽ đưa vào bộ SubBytes, tiếp theo đầu ra sẽ đưa và shift_rows và mix_columns, rồi kết quả sẽ đưa qua trở lại Add round key vòng tiếp theo, nó sẽ lặp lại 9 vòng. Tới vòng thứ 10 thì tín hiệu cipher_complete bằng 1, ở vòng 10 sẽ không có bước mix_columns.

Sau khi dữ liệu đã đủ 10 vòng biến đổi, tín hiệu cipher_ready sẽ bằng 1, lúc này bộ MUX sẽ lấy dữ liệu của cipher_text_reg là dữ liệu đầu ra, dữ liệu sau khi mã hóa sẽ là cipher_text.



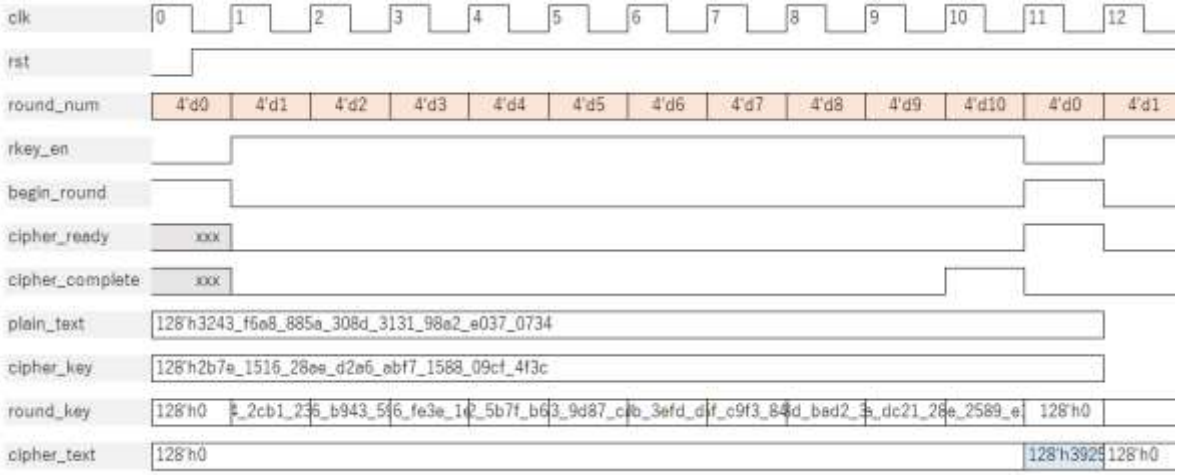
Hình 3.5: Đưa kết quả đầu ra sau khi hoàn thành mã hóa

Để có được các tín hiệu điều khiển các bộ MUX, thì cần có một khối AES cipher controller.



Hình 3.6: Bộ điều khiển các tín hiệu cần thiết cho bộ mã hóa

Khối này sẽ hoạt động dựa vào chu kỳ xung clk, cứ sau mỗi chu kì xung clk, thì các giá trị đầu ra sẽ thay đổi như đồ thị được minh hoạt bên dưới:



Hình 3.7: Minh họa tín hiệu điều khiển thay đổi sau mỗi chu kì xung clk

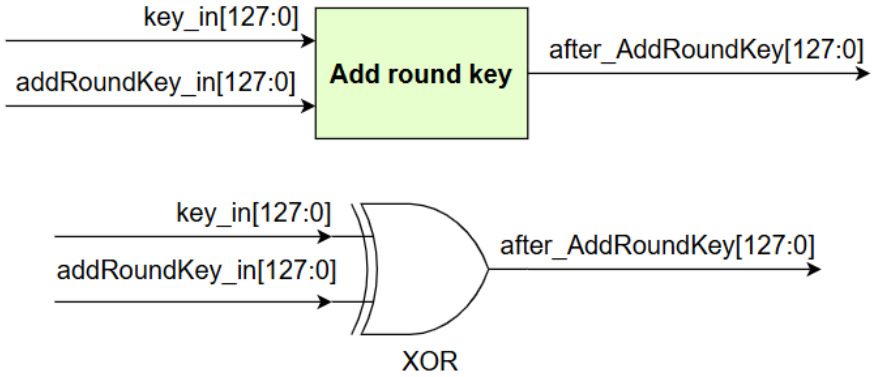
Thiết kế chi tiết cho từng chức năng trong lõi:

Theo như đã phân tích trước đó, quy trình mã hóa sẽ thực hiện qua 10 vòng, 9 vòng đầu gồm 4 chức năng SubBytes, ShiftRows, MixColumns và AddRoundKey, vòng cuối cùng không có MixColumns. Dữ liệu đầu vào là plain_text và cipher_key cùng độ rộng 128 bit (16 byte).

Chức năng AddRoundKey:

Bước khởi tạo, thực hiện XOR khóa mã (cipher_key) với ma trận dữ liệu (plain_text). Bước lặp mã hóa và bước tạo ngõ ra thì XOR khóa vòng (round_key) với ma trận trạng thái sau bước mix_coloumns (after_mixColumns)

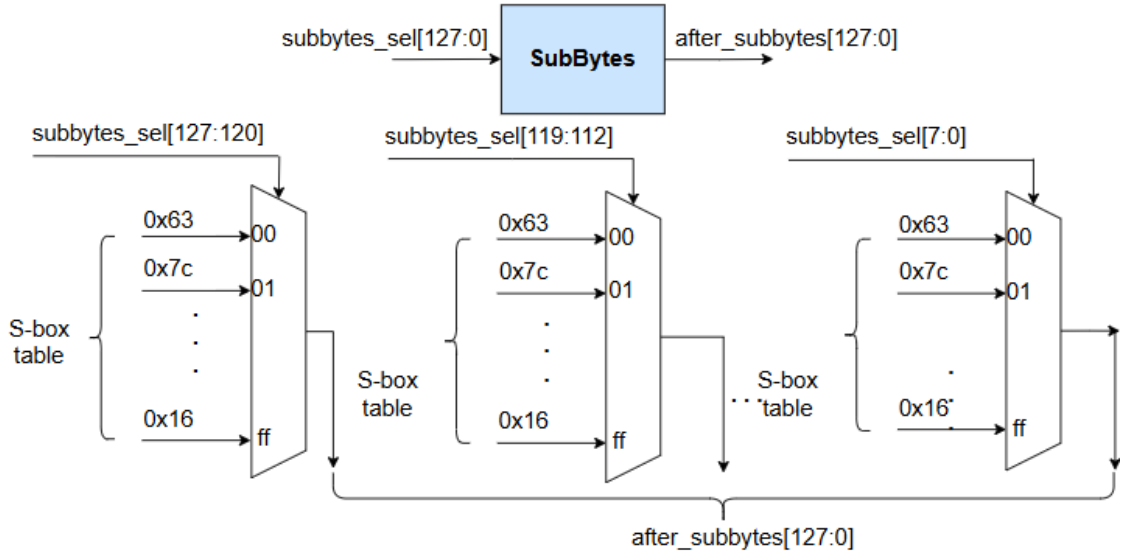
Ở chức năng này, nó sẽ XOR lần lượt từng byte tương ứng cùng vị trí của hai ma trận với nhau và kết quả sẽ tạo ra một ma trận mới đưa vào chức năng tiếp theo.



Hình 3.8: Khối chức năng AddRoundKey

Chức năng SubBytes:

Chức năng này thực hiện thay thế từng byte của ma trận trạng thái, ngõ ra của AddRoundKey bằng một giá trị đã quy định trong chuẩn AES. Bảng quy định này có giá trị thay thế gọi là S-box.

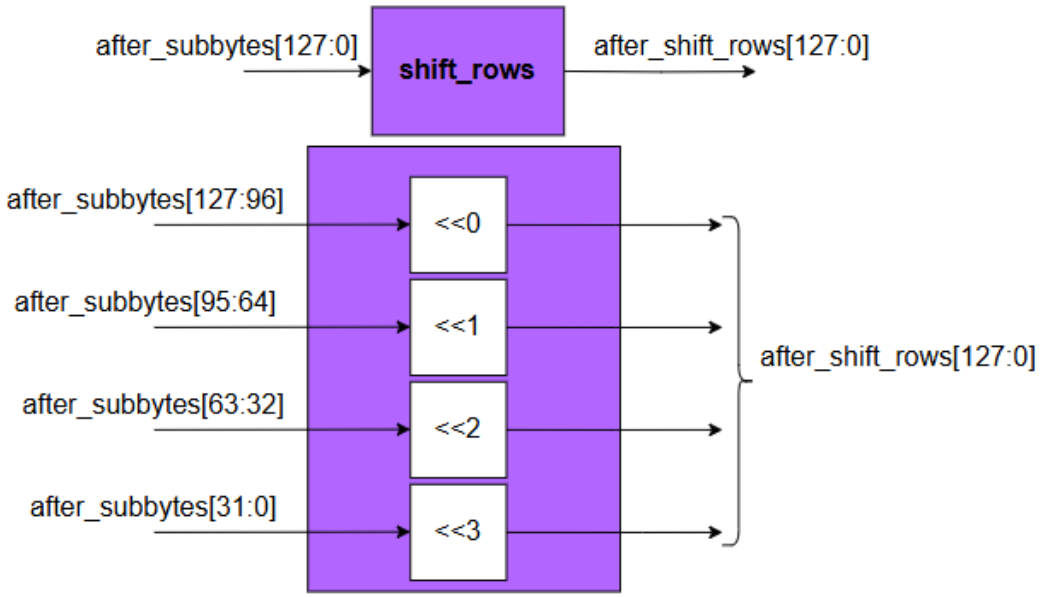


Hình 3.9: Khối chức năng SubBytes

Chức năng ShiftRows:

Chức năng ShiftRows thực hiện quay trái từng hàng của ma trận trạng thái, ngõ ra của SubBytes, theo byte với hệ số quay tăng dần từ 0 đến 3. Hàng đầu tiên có hệ số quay là 0 thì các byte được giữ nguyên vị trí. Hàng thứ hai có hệ số quay là 1 thì các byte

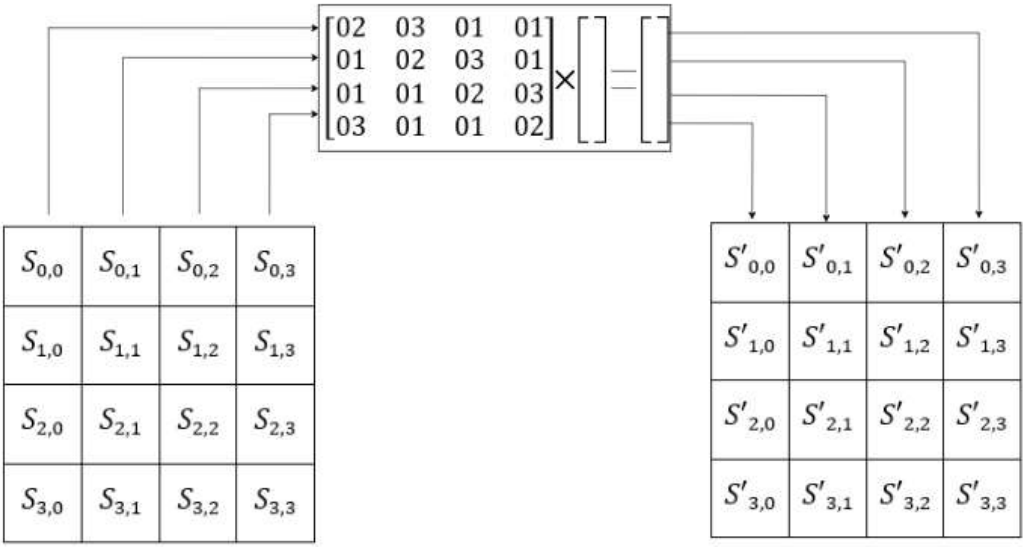
được quay một byte. Hàng thứ ba quay hai byte và hàng thứ tự quay ba byte.



Hình 3.10: Chức năng ShiftRows

Chức năng MixColumns:

Chức năng MixColumns thực hiện nhân từng cột của ma trận trạng thái, ngõ ra của ShiftRows, với một ma trận chuyển đổi quy định bởi chuẩn AES.



Hình 3.11: Nguyên lý biến đổi của chức năng

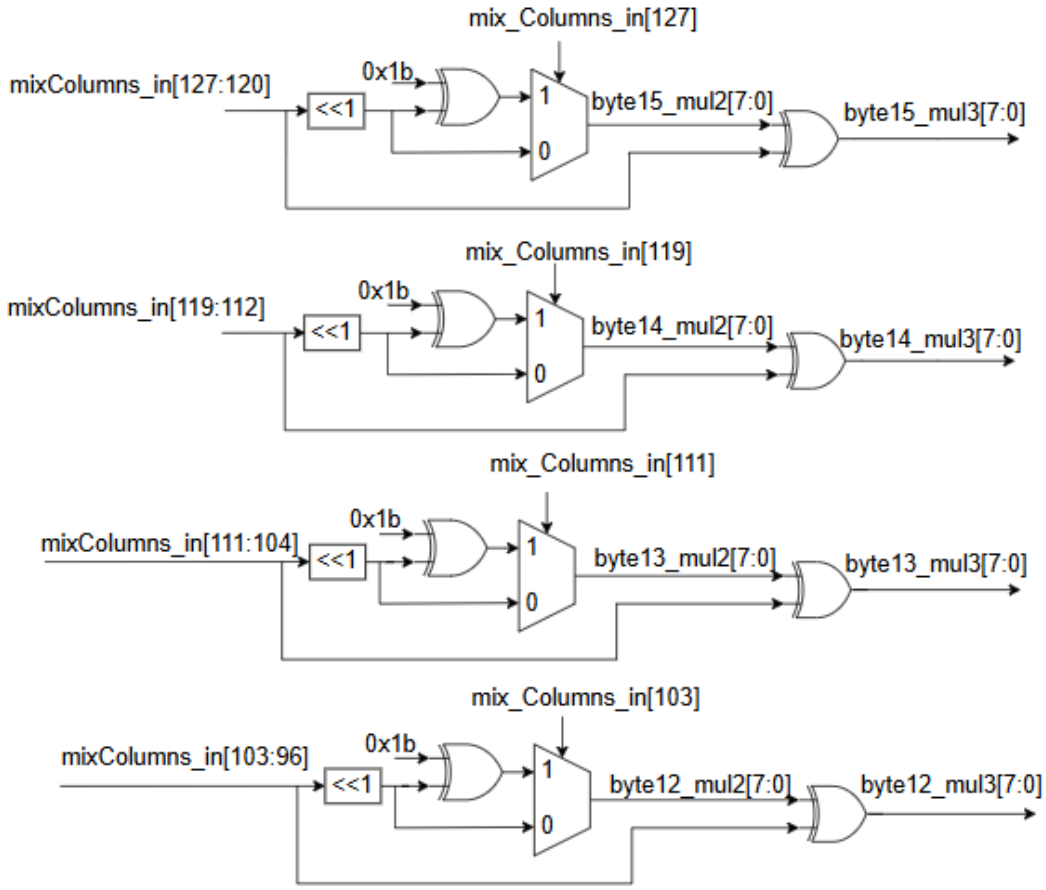
Việc biến đổi một cột của ma trận trạng thái được thực hiện bởi hai phép toán là nhân (.) và XOR (+).

Mỗi ô là 1 byte, tất cả đều ở dạng Hex, biểu thức sau tạo ra phần từ S'00 ở hàng 1 cột 1 "chức năng MixColumns":

S'00 = S00.0x02 + S10.0x03 + S20 + S30 = S00.0x02 + (S10.0x02 + S10.0x01) + S20 + S30

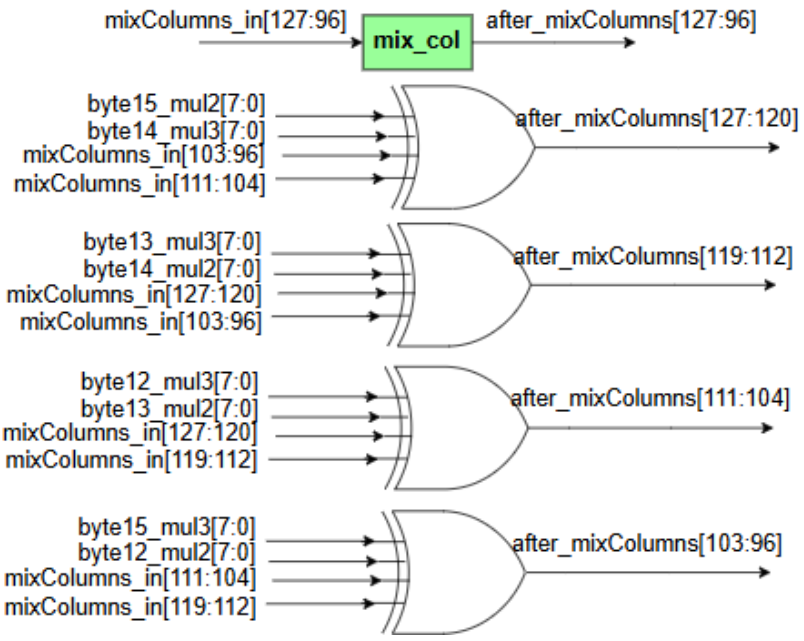
Phép nhân với 0x01 thì giữ nguyên giá trị. Phép nhân với 0x02 tương đương với việc dịch trái một bit và XOR có điều kiện như sau: Nếu bit MSB (bít đầu tiên bên trái) của giá trị được dịch bằng 1 thì giá trị sau khi dịch được XOR với 0x1b còn nếu bit MSB của giá trị được dịch bằng 0 thì giữ giá trị sau khi dịch.

Thiết kế sơ đồ logic cho chức năng: Phần tử nào nhân với 0x02, kết quả sau đó XOR với chính nó thì sẽ ra kết quả của phần tử đó nhân với 0x03



Hình 3.12: Thiết kế logic tạo ra các phần từ nhân với H02 và H03

Kết quả sẽ XOR 4 phần từ lại để thu được byte kết quả, dưới đây là kết quả của cột đầu tiên trong ma trận trạng thái:



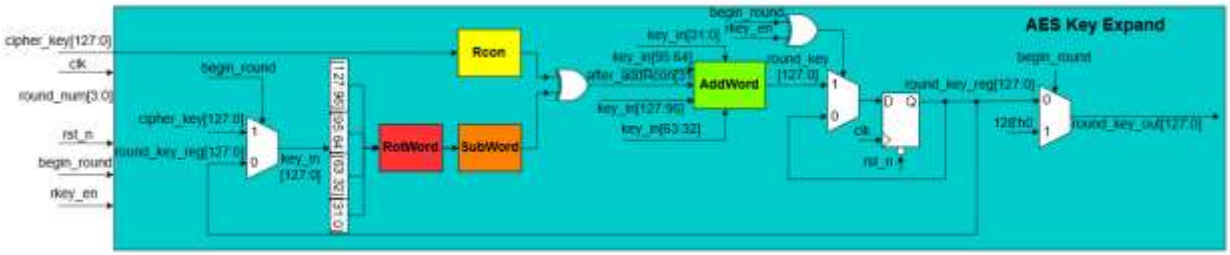
Hình 3.13: Dữ liệu đầu ra cột đầu tiên của ma trận trạng thái

Tương tự như cách tính ở cột 1, các cột còn lại của ma trận trạng thái cũng sẽ được tính dựa trên sơ đồ logic đó.

3.2.3. Thiết kế lõi AES Key Expand

Chức năng KeyExpansion thực hiện tính toán khóa vòng cho bước lặp mã hóa và bước tạo ngõ ra. Kết quả của một lần thực thi KeyExpansion là một khóa vòng sử dụng cho chức năng AddRoundKey. Với mã hóa AES-128, số khóa vòng là 10 tương ứng với 9 lần AddRoundKey ở bước lặp mã hóa và 1 lần AddRoundKey ở bước tạo ngõ ra. Chức năng này được thực hiện thông qua 4 chức năng nhỏ bên trong là RotWord, SubWord, AddRcon và AddW.

Sơ đồ logic thiết kế bên trong khối AES Key Expand:



Hình 3.14: Sơ đồ logic của lõi AES Key Expand

Sơ đồ logic triển khai tạo khóa cho các vòng sẽ thực theo trình từ trái qua phải, cứ mỗi chu kì xung clk thì nó sẽ thực hiện biến đổi 1 vòng tạo khóa, tại thời điểm cạnh lên của xung clk, FlipFlop sẽ bắt dữ liệu ở chân D qua chân Q, cập nhật giá trị mới cho khóa vòng tiếp theo, sau khi đủ 10 vòng tạo khóa thì bộ MUX phía sau D-FF sẽ bắt dữ liệu round_key_reg.

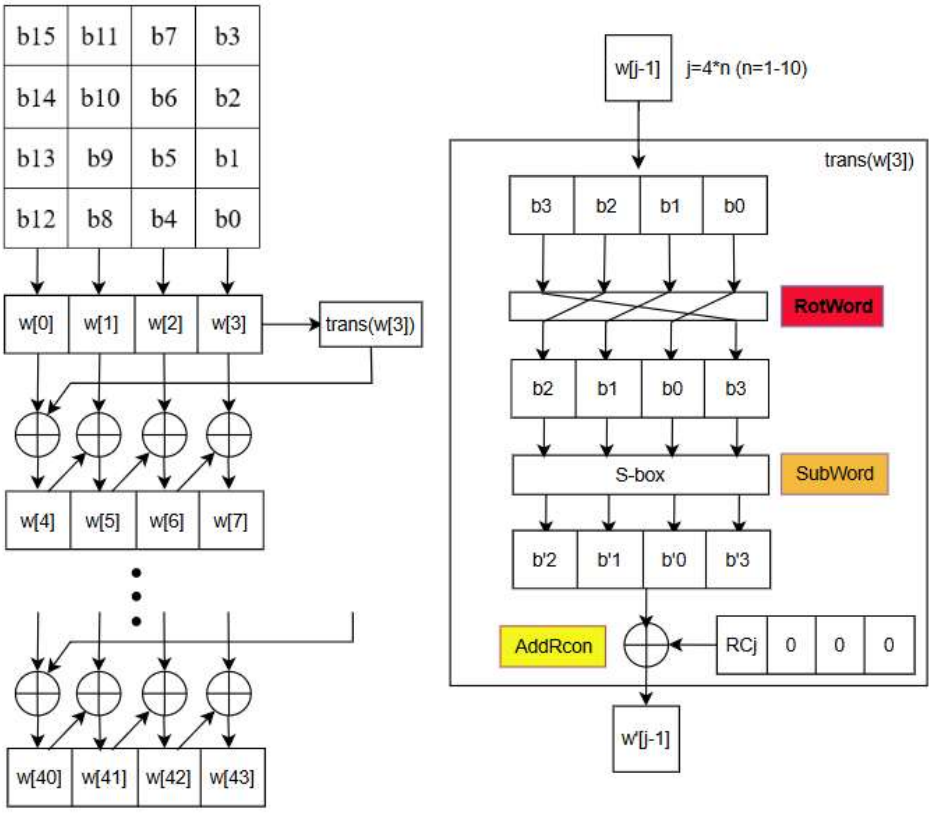
Mỗi khóa vòng có 128 bit, được chia làm 4 word, mỗi word là 4 byte và ký hiệu là w[j] với j là số nguyên. Mã hóa AES-128 có 1 khóa mã và 10 khóa vòng nên tổng số word là 44 từ w[0] tới w[33]. Khóa mã ban đầu đưa vào có 4 word là w[0], w[1], w[2] và w[3]. Khóa vòng 1 có 4 word là w[4], w[5], w[6] và w[7]. Tương tự khóa vòng 2 tới khóa vòng 9. Khóa vòng 10 có 4 word là w[40], w[41], w[42] và w[43].

Từ w[j] tính theo công thức sau, với 3<j<44

W[j] = AddW[j-4] = w[j-1] XOR w[i-4]

w[j=4*n] = AddW[j-4] = trans(w[j-1]) XOR w[j-4]

Khi tính từ các word ở vị trí j là bội số của 4, là w[4], w[8],… và w[40] thì w[j-1] phải được biến đổi qua 3 chức năng RotWord, SubWord và AddRcon, gọi là trans(w[j-1]), trước khi XOR với w[j-4]. Ví dụ w[4] = trans(w[3]) XOR w[0].

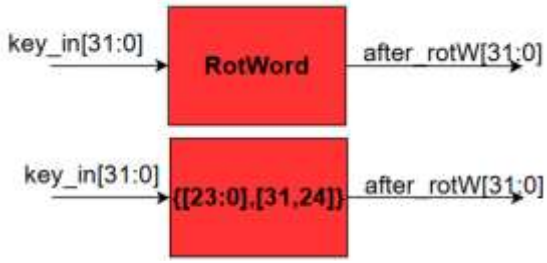


Hình 3.15: Các bước biến đổi để tạo khóa vòng

Các chức năng RotWord, SubWord và AddRcon sẽ được mô tả chi tiết hơn ở các phần sau.

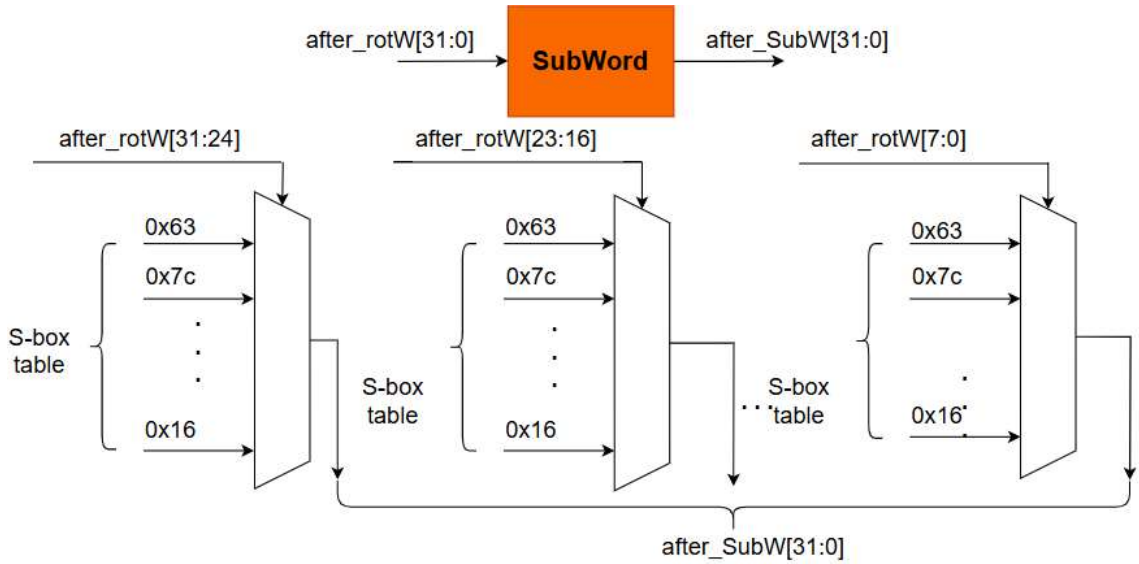
Chức năng RotWord:

Chức năng này thực hiện quay trái từ w[j] một byte



Hình 3.16: Minh họa triển khai chức năng RotWord

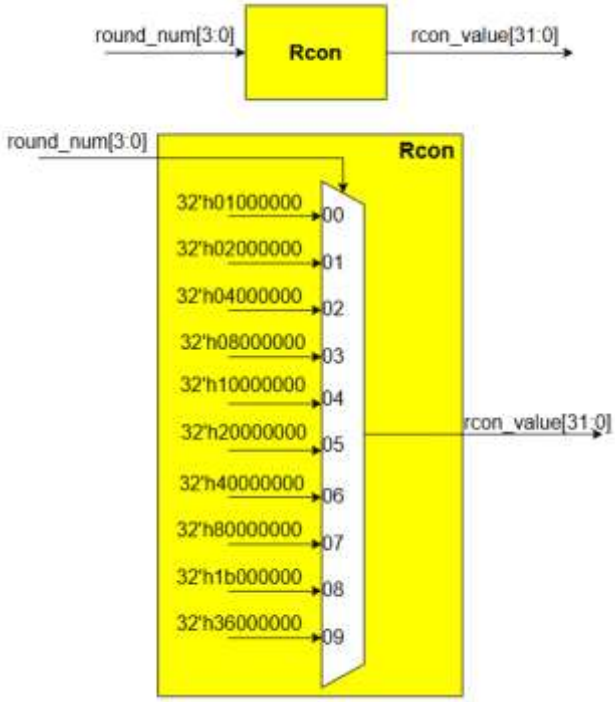
Chức năng SubWord: Chức năng này sẽ thay thế từng byte của kết quả RotWord theo bảng S-box như chức năng SubBytes.



Hình 3.17: Triển khai thiết kế chức năng SubWord

Chức năng AddRcon:

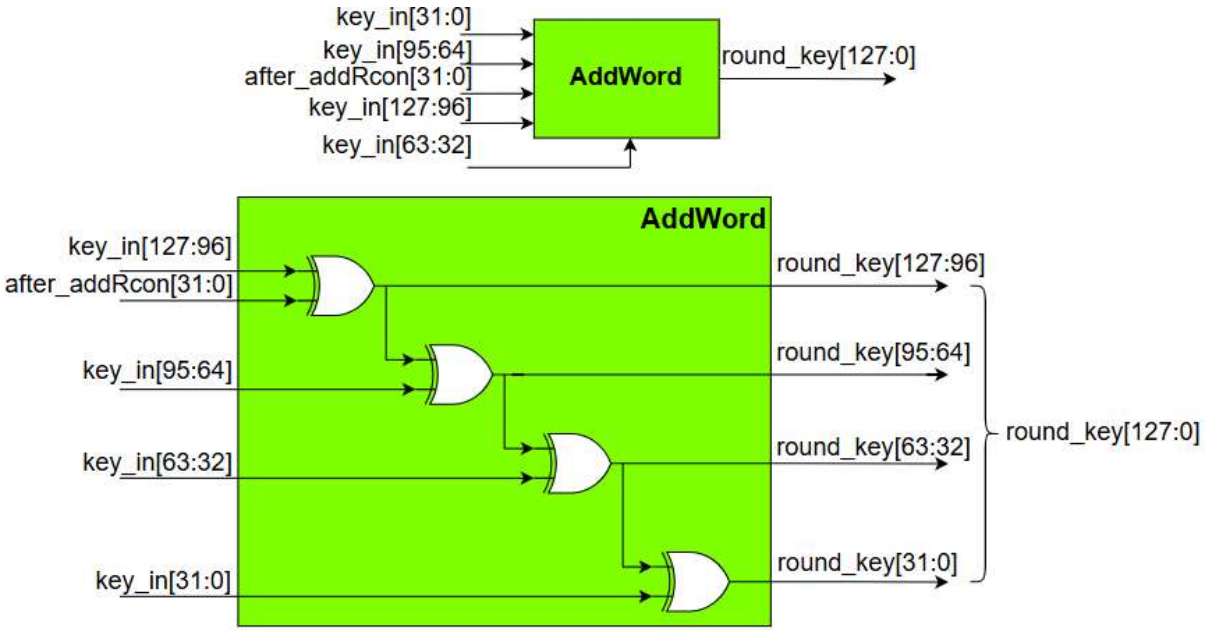
Chức năng AddRcon thực hiện XOR kết quả SubWord và giá trị Rcon[j/4] với j là bội số của 4. Số lượng giá trị Rcon[j/4] là 10 tương ứng với 10 lần tính khóa vòng. Chức năng AddRcon sẽ tạo ra kết quả cuối cùng của biến đổi trans(w[j-1]).



Hình 3.18: Thiết kế chức năng AddRcon

Chức năng AddW:

Thực hiện XOR w[j-4] với w[j-1] hoặc trans(w[j-1]) để tạo ra khóa vòng.

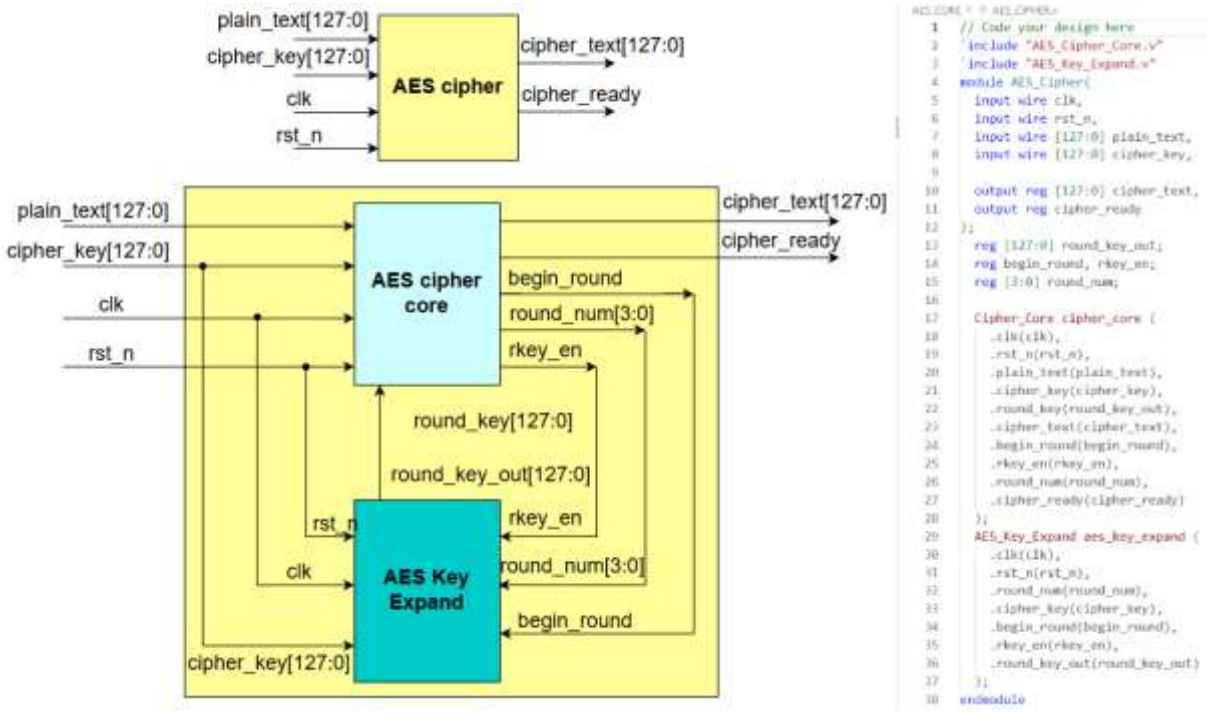


Hình 3.19: Sơ đồ logic triển khai chức năng AddW

3.2.4. Mô tả RTL.

Từ thiết kế logic ở mức thấp, sẽ sử dụng ngôn ngữ mô tả phần cứng (HDL) là Verilog để mô tả toàn bộ thiết kế.

Ví dụ mô tả RTL code của module top trong thiết kế chức năng mã hóa:



Hình 3.20: Mô tả RTL thiết kế mức cao lõi AES cipher

Trong đoạn code mô tả RTL, thực hiện khai báo cổng vào ra, các kết nối bên trong của lõi.

Các chức năng trong quá trình mã hóa đều được mô tả RTL code. Sau đó sẽ thực hiện gỡ lỗi và chạy mô phỏng kiểm thử, kiểm tra các trường hợp đưa dữ liệu đầu vào và đầu ra thu được các dạng sóng như mong muốn thì thiết kế mô tả RTL đã đúng. Còn nếu phát hiện bất kỳ lỗi nào, dạng sóng đầu ra thu được không như mong muốn thì phải quay lại bước trước đó để thực hiện thiết kế lại.

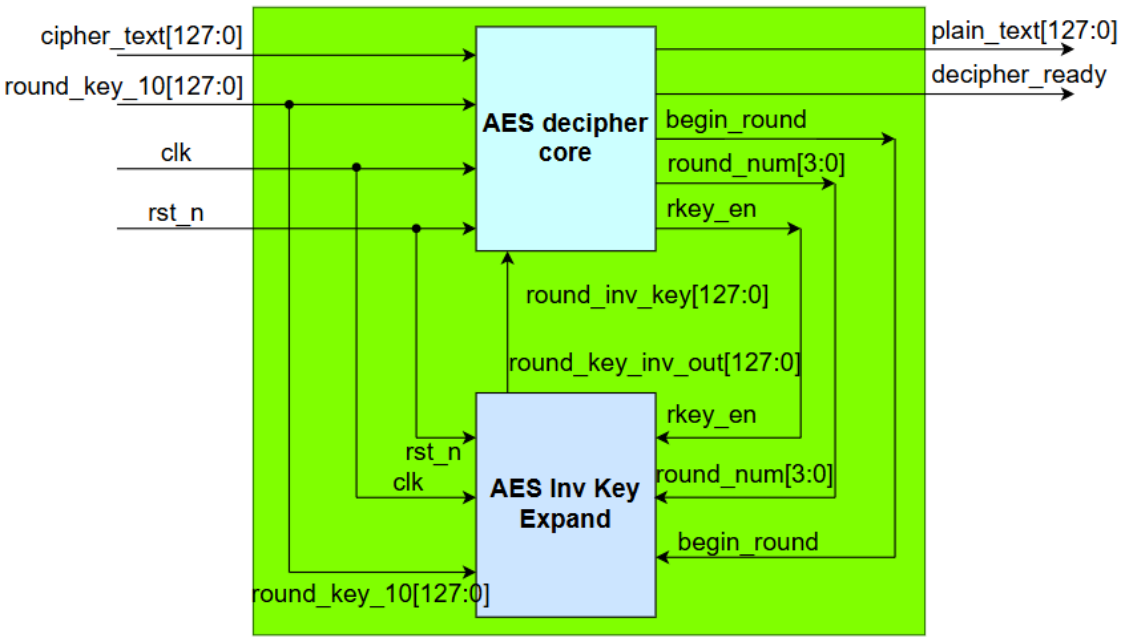
Sau bước này là kết thúc quá trình mô tả RTL code, chuyển qua xác minh.

3.3. Thiết kế lõi IP mềm phần giải mã

Quy trình thiết kế lõi IP mềm cho phần giải mã tương tự như mã hóa. Dữ liệu cần được giải mã cipher_text và khóa round_key_10 là đầu vào. Sau khi biến đổi qua các vòng thì sẽ đưa dữ liệu đầu ra plain_text là dữ liệu đã được giải mã.

3.3.1. Thiết kế khối mức cao (High Level Block Design)

Lõi IP mềm AES chức năng giải mã, AES decipher được thiết kế gồm 2 khối: Khối giải mã dữ liệu AES decipher core và khối tính toán tạo khóa vòng AES Inv Key Expand.



Hình 3.21: Lõi bên trong của AES decipher

Các tín hiệu vào ra và kết nối nội bên trong của AES decipher:

Bảng 3.3: Các tín hiệu vào ra của AES decipher

Tín hiệu	Mô tả
clk	clock đồng bộ
rst_n	reset tích cực mức thấp
round_key_10[127:0]	khóa vòng cho quá trình giải mã
cipher_text[127:0]	dữ liệu cần giải mã
decipher_ready	báo trạng thái bộ giải mã sẵn sàng hoạt động. Tín hiệu này chỉ bằng 0 khi bộ giải mã đang trong quá trình giải mã dữ liệu
plain_text[127:0]	dữ liệu sau khi được giải mã

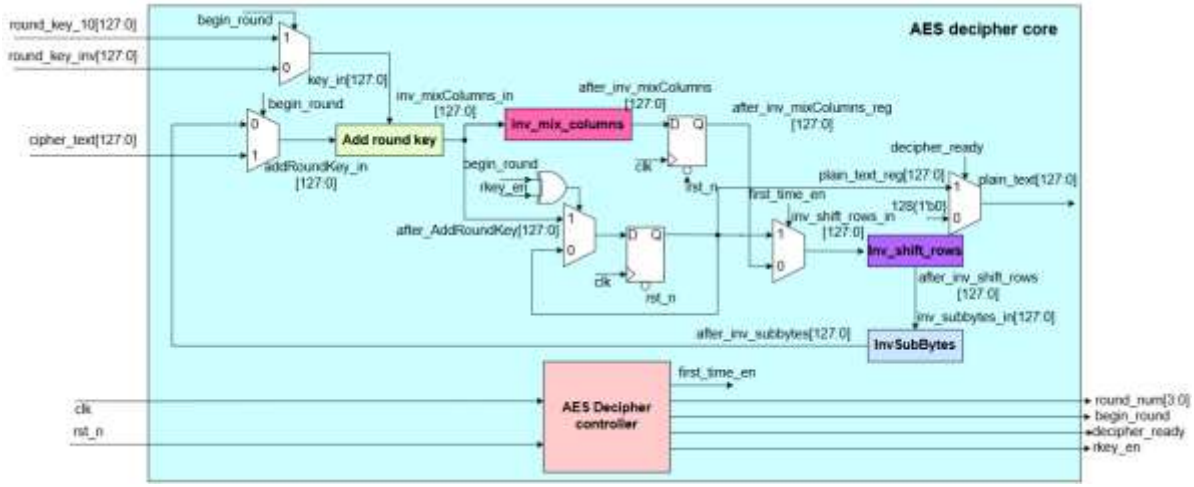
Bảng 3.4: Các tiến hiệu giao tiếp giữa AES decipher core và AES Inv Key Expand

Tín hiệu	Mô tả
rkey_en	tín hiệu cho phép khối AES Inv Key Expand hoạt động
round_num[3:0]	tín hiệu báo vòng lặp giải mã hiện tại, round_num[3:0]=0 là bước khởi tạo đầu tiên.
round_key_inv_out[127:0]	là giá trị khóa vòng từ khối AES Inv Key Expand gửi cho khối AES decipher core.
begin_round	tín hiệu bắt đầu mã hóa, begin_round=1 khi round_num[3:0]=0

3.3.2. Thiết kế khối mức thấp (Low Level Block Design)

Sau khi hoàn thành thiết kế khối mức cao cho lõi giải mã, tương tự như mã hóa, phần này sẽ đi sâu vào phân tích và thiết kế các thành phần bên trong khối mức cao đó.

Thiết kế khối mức thấp cho AES decipher core:



Hình 3.22: Sơ đồ logic bên trong AES decipher core

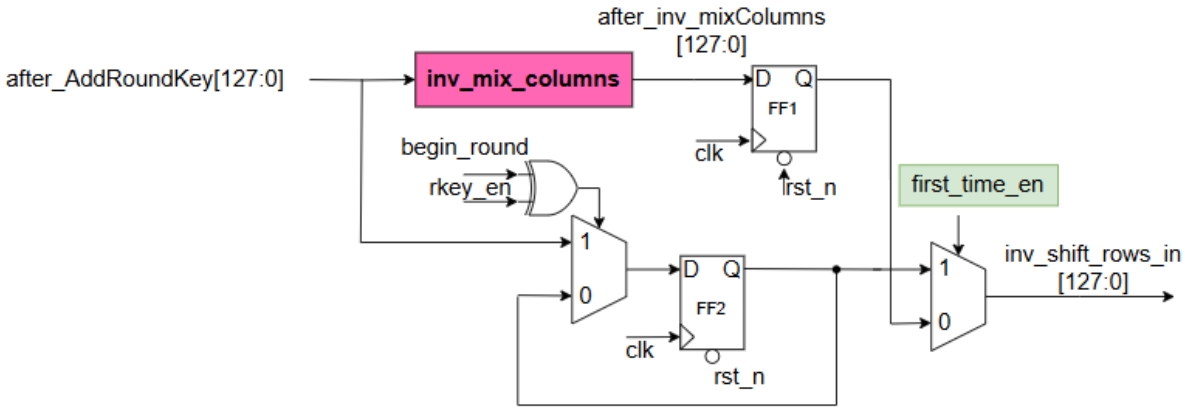
Thiết kế khối mức thấp cho lõi giải mã:

Các thành phần bên trong lõi giải mã cũng tương tự như mã hóa, các khối MUX, cổng XOR, FF cũng thực hiện các chức năng tương tự.

Dữ liệu đưa vào bộ giải mã ở bước khởi tạo tương tự như bộ mã hóa. Sau khi dữ liệu được đưa vào Add round key, thu được dữ liệu đầu ra, dữ liệu này sẽ được đưa vào làm đầu vào cho vòng đầu tiên.

Quá trình giải mã sẽ ngược lại so với quá trình mã hóa, tuy nhiên khóa của quá trình giải mã sẽ không dùng chung với khóa của quá trình mã hóa trước đó, nghĩa là sẽ sử dụng thuật toán biến đổi ngược để tạo khóa thay vì dùng thanh ghi để lưu mảng giá trị khóa của quá trình mã hóa trước đó, giúp tiết kiệm được bộ nhớ.

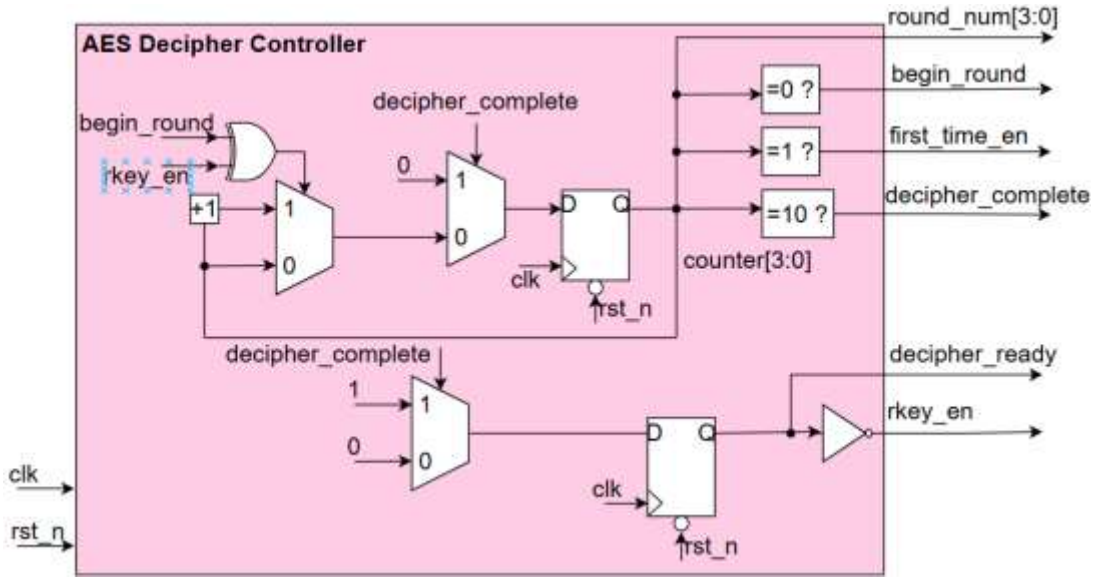
Sau bước khởi tạo, tới vòng biến đổi đầu tiên, dữ liệu đưa vào khối InvShiftRows rồi tiếp theo qua khối InvSubBytes sau đó quay về AddRoundKey và qua InvMixColumns thì sẽ kết thúc vòng biến đổi thứ 1. Để làm được như vậy, cần sử dụng một tín hiệu first_time_en để bắt dữ liệu tại sau mỗi cạnh lên của xung clk ở mỗi chu kì.



Hình 3.23: Sơ đồ logic bắt dữ liệu tại đầu mỗi vòng

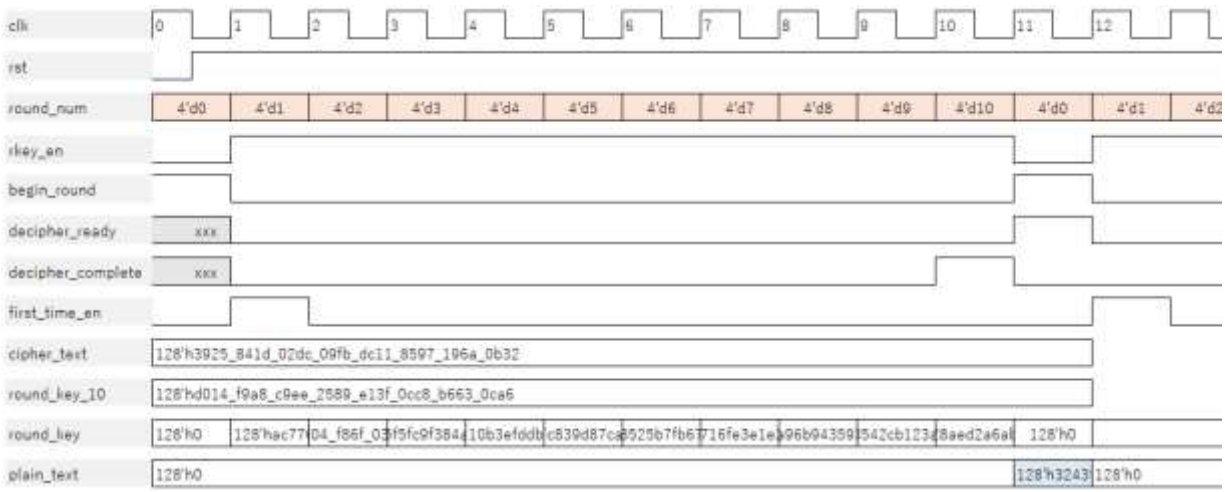
Nguyên tắc hoạt động của sơ đồ Hình 3.24 là tín hiện first_time_en =1 trong chu kì xung clk ở vòng thứ 1, đảm bảo tại sườn lên xung clk của chu kỳ đầu tiên, dữ liệu đưa vào chức năng InvShiftRows sẽ là dữ liệu tại chân Q của FF2. Kết thúc vòng biến đổi thứ 1, dữ liệu sau khi ra khỏi chức năng InvMixColumns thì sẽ có tại chân D của FF1. Tại sườn lên của chu kì xung clk thứ 2, tức qua vòng thứ 2, tín hiệu first_time_en = 0, dữ liệu tại chân Q của FF1 được bắt từ chân D, và sẽ đưa dữ liệu này vào chức năng tiếp theo. Tín hiện first_time_en = 0 trong các chu kỳ còn lại cho tới khi quá trình giải mã một chuỗi dữ liệu hoàn thành.

Để điều khiển các tín hiệu của bộ MUX thì sẽ cần một khối AES decipher controller tại ra các tín hiệu điều khiển, hình bên dưới là sơ đồ khối của bộ điều khiển tạo tín hiệu:



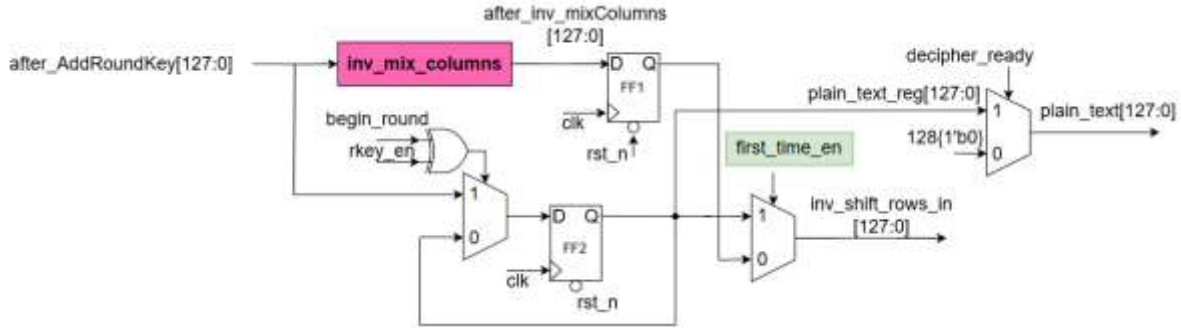
Hình 3.24: Bộ điều khiển các tín hiệu cần thiết cho bộ giải mã

Các tín hiệu sẽ biến đổi sau mỗi chu kì xung clk, và lặp lại sau mỗi lần hoàn thành quá trình giải mã dữ liệu. Hình dưới sẽ minh họa đầy đủ các giai đoạn biến đổi chính từ khi đưa dữ liệu vào cho tới khi dữ liệu được mã hóa.



Hình 3.25: Tín hiệu điều khiển lõi giải mã thay đổi sau mỗi chu kì xung clk

Quá trình giải mã và tạo khóa sẽ thực hiện song song lặp lại 9 lần tương tự như vòng 1. Vòng cuối cùng round_num[3:0] =10, decipher_complete = 1, dữ liệu sẽ thực hiện biến đổi qua 3 chức năng, không có chức năng InvMixColumns, kết thúc vòng tạo ngõ ra này, dữ liệu sẽ có tại chân D của FF2. Tại sườn lên của chu kì xung clk tiếp theo, tín hiệu decipher_ready = 1, bộ MUX sẽ bắt dữ liệu tại chân Q của FF2 gán cho plain_text[127:0] và đây là kết quả giải mã.



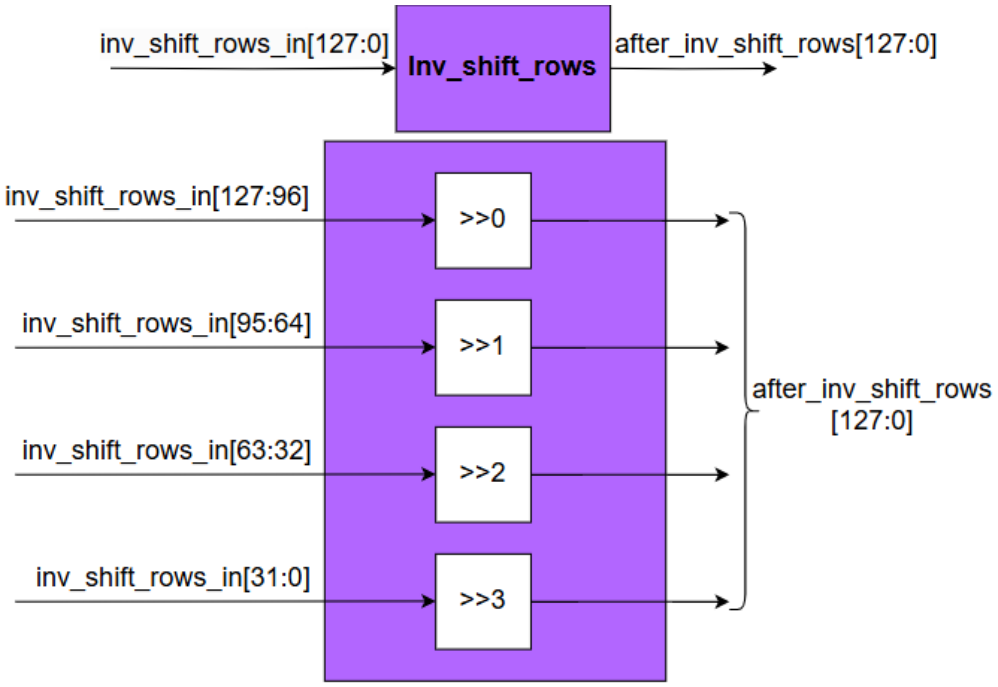
Hình 3.26: Khối logic tạo dữ liệu ngõ ra bộ giải mã

Thiết kế chi tiết cho từng chức năng trong lõi:

Tương tự với mã hóa, giải mã cũng trải qua 10 vòng, 9 vòng đầu đầy đủ 4 chức năng theo thứ tự InvShiftRows, InvSubBytes, AddRoundKey và InvMixColumns; vòng cuối cùng thì sẽ không có InvMixColumns.

Chức năng InvShiftRows:

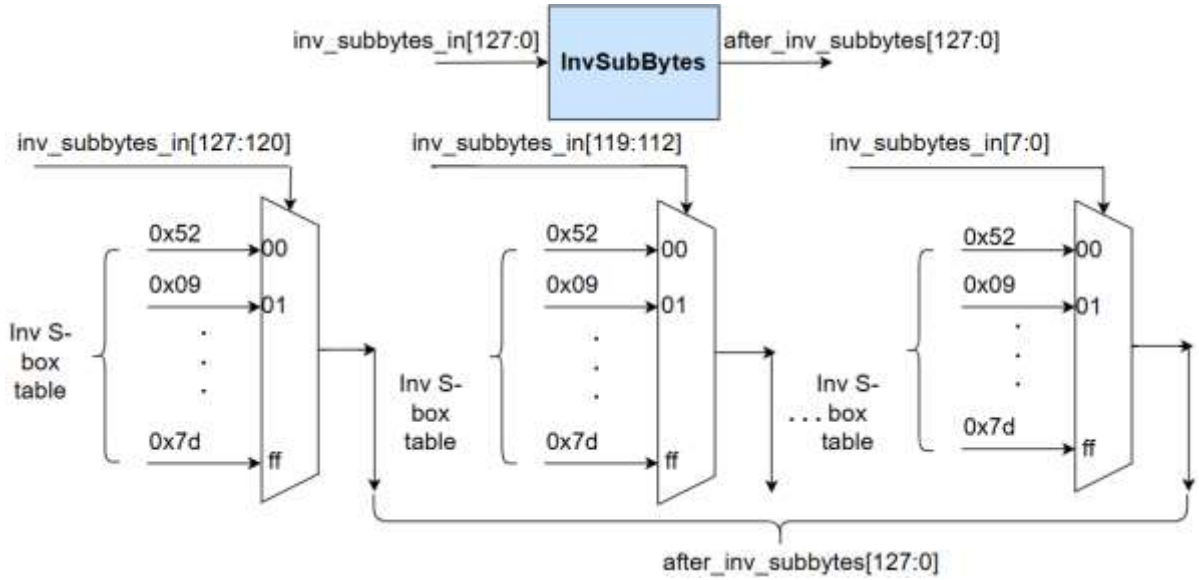
InvShiftRows là đảo của chức năng ShiftRows. InvShiftRows thực hiện quay phải từng hàng của ma trận trạng thái, sinh ra từ bước trước đó, theo byte với hệ số quay tăng dần từ 0 đến 3. Hàng đầu tiên có hệ số quay là 0 thì các byte được giữ nguyên vị trí. Hàng thứ hai có hệ số quay là 1 thì các được quay một byte. Hàng thứ ba quay hai byte và hàng thứ tư quay ba byte.



Hình 3.27: Chức năng InvShiftRows

Chức năng InvSubBytes:

Chức năng InvSubBytes là thực hiện thay thế từng byte của ma trận trạng thái, bằng một giá trị đã quy định trong chuẩn AES. Bảng quy định giá trị thay thế cho InvSubBytes gọi là S-box đảo (Inverse S-box).



Hình 3.28: Chức năng InvSubBytes

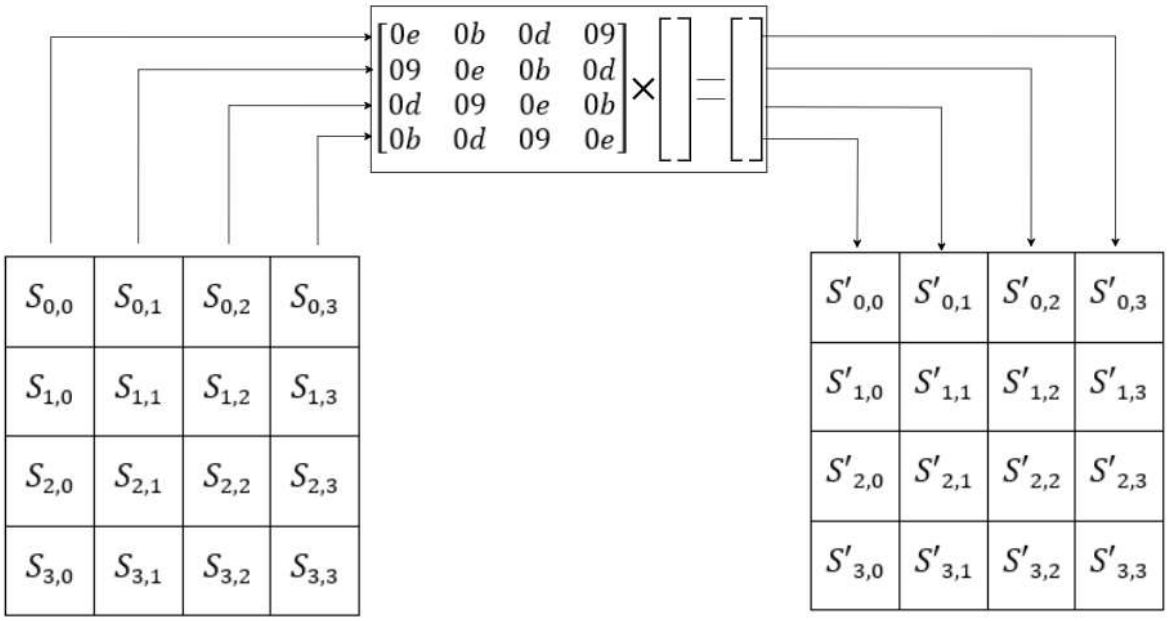
Chức năng AddRoundKey:

Chức năng AddRoundKey đảo trong quá trình giải mã cũng chính là chức năng AddRoundKey trong quá trình mã hóa nên gọi chung là AddRoundKey.

Chức năng InvMixColumns:

InvMixColumns của quá trình giả mã là đảo của MixColumns trong quá trình mã

hóa. Từng cột của ma trận trạng thái sẽ được nhân với ma trận chuyển đổi sau đây



Hình 3.29: Nhân ma trận trạng thái với ma trận chuyển đổi

Nguyên tắc tính toán InvMixColumns là biến đổi các phép nhân với một số thành phép nhân với 01_{16} và 02_{16}, thì cuối cùng nguyên tắc của nó sẽ tương tự như MixColumns.

Thuật toán trong chức này này như sau:

Phép nhân một byte A với *0x0e* ($0e_{16}$) = 00001110_2 sẽ tương tự như sau:

$A.0e_{16} = A.08_{16} + A.04_{16} + A.02_{16} = A.02_{16}.02_{16}.02_{16} + A.02_{16}.02_{16} + A.02_{16}$

Phép nhân một byte A với $0b_{16} = 00001011_2$ sẽ tương tự như sau:

$A.0b_{16} = A.08_{16} + A.02_{16} + A.01_{16} = A.02_{16}.02_{16}.02_{16} + A.02_{16} + A.01_{16}$

Phép nhân một byte A với $0d_{16} = 00001101_2$ sẽ tương tự như sau:

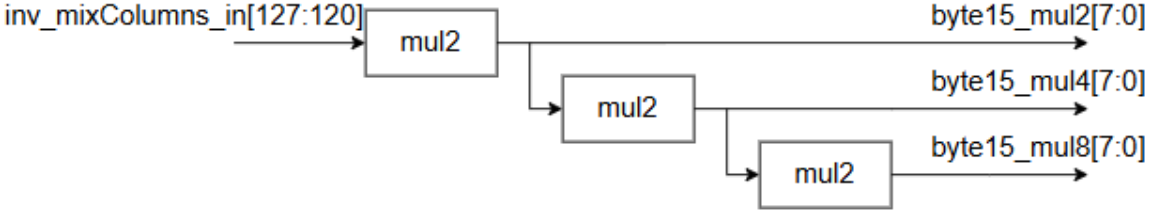
$A.0d_{16} = A.08_{16} + A.04_{16} + A.01_{16} = A.02_{16}.02_{16}.02_{16} + A.02_{16}.02_{16} + A.01_{16}$

Phép nhân một byte A với $09_{16} = 00001001_2$ sẽ tương tự như sau:

$A.09_{16} = A.08_{16} + A.01_{16} = A.02_{16}.02_{16}.02_{16} + A.01_{16}$

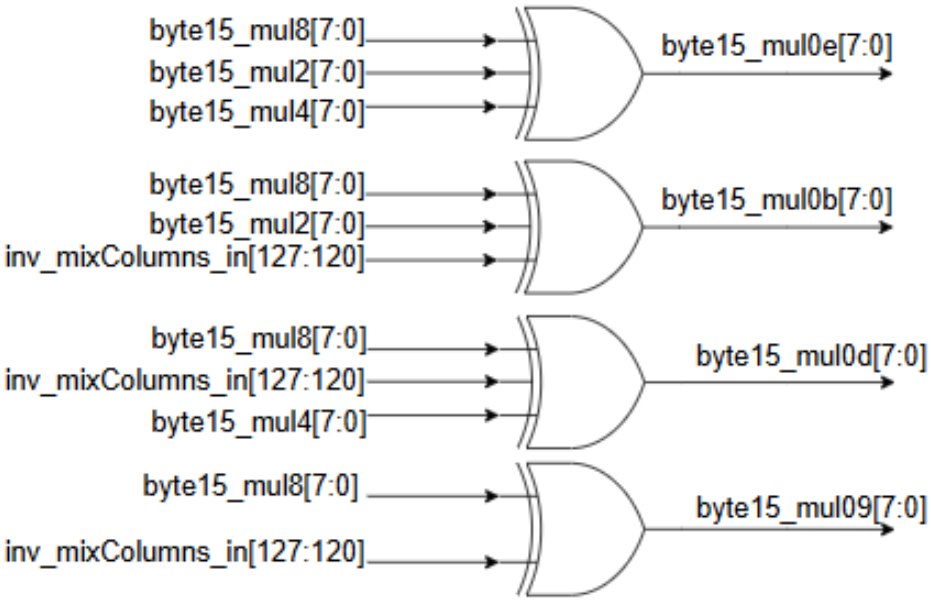
Triển khai thành các sơ đồ logic:

Phân tích một byte khi nhân với 02_{16}, 04_{16} và 08_{16} thì tách thành 3 cấp như hình dưới, các byte còn lại sẽ tương tự:



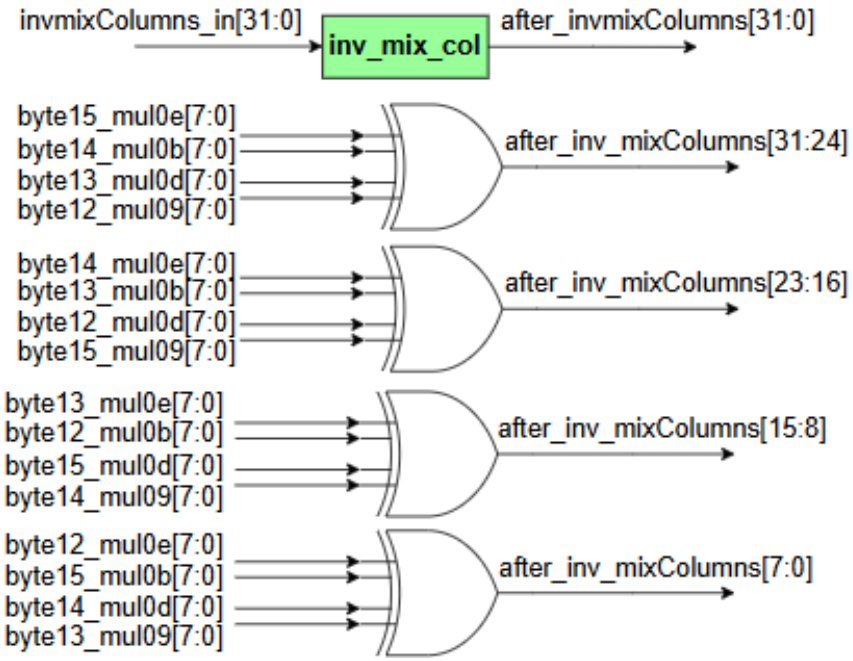
Hình 3.30: Tách một byte thành 3 cấp

Minh họa byte[15] nhân với $0e_{16}$, $0b_{16}$, $0d_{16}$ và 09_{16} lần lượt được triển khai, các byte còn lại sẽ tương tự:



Hình 3.31: Triển khai minh họa một byte nhân với $0e_{16}$, $0b_{16}$, $0d_{16}$ và 09_{16}

Tiếp theo, thực hiện phép XOR để thu được kết quả sau bước InvMixColumns của cột đầu tiên trong ma trận trạng thái, các cột còn lại sẽ được triển khai tương tự:



Hình 3.32: Kết quả của cột đầu tiên trong ma trận trạng thái

Triển khai thiết kế chức năng InvMixColumns phức tạp hơn MixColumns, yêu cầu phải chia nhỏ phép nhân và XOR để đưa về dạng nhân cơ bản, rồi sau đó sẽ XOR chúng lại với nhau. Hoàn thành bước thiết kế mức thấp này thì chuyển qua bước tiếp theo.

3.3.3. Thiết kế lõi AES Inv Key Expand

Chức năng mã hóa sẽ dùng khóa vòng từ 1 tới 10, còn chức năng giải mã sẽ sử dụng khóa từ 10 về 1. Cách thiết kế thông thường là dùng mảng để lưu khóa vòng trong quá trình mã hóa, qua quá trình giải mã sẽ lấy đó và dùng lại, tuy nhiên cách này sẽ gây tốn tài nguyên bộ nhớ. Để giải quyết điều này, phần này sẽ thiết kế chức năng biến đổi khóa ngược theo thuật toán quy định.

Tuy nhiên, như đã đề cập ban đầu, hai chức năng mã hóa và giải mã sẽ hoạt động độc lập vì thế thứ tự khóa vòng sẽ tính từ vòng 1 tới 10 chứ không phải tính ngược lại từ 10 về 1. Nguyên tắc tính của Inv Key Expansion biến đổi khóa n+1 từ khóa thứ n như sau:

Mỗi khóa gồm 4 word với word[3] là MSB và word[0] là LSB

word[0] của khóa thứ n+1= word[0] của khóa thứ n XOR word[1] của khóa thứ n

word[1] của khóa thứ n+1 = word[1] của khóa thứ n XOR word[2] của khóa thứ n

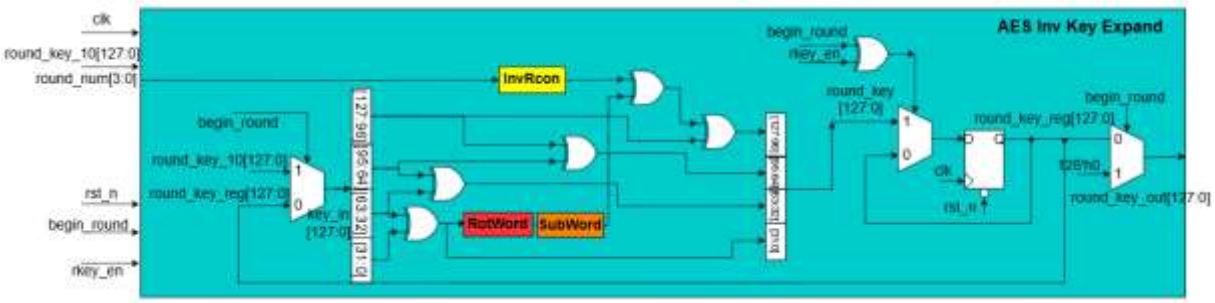
word[2] của khóa thứ n+1 = word[2] của khóa thứ n XOR word[3] của khóa thứ n

word[3] của khóa thứ n+1= word[3] của khóa thứ n XOR với giá trị sau khi biến đổi word[0] của khóa thứ n+1 (đã tính trên đây) qua các bước:

+ RotWord: đảo byte MSB xuống LSB

+ SubWord: ứng với function aes128_sbox()

+ InvAddRcon: lấy ngõ ra SubWord XOR với giá trị Rcon chọn từ function aes128_rcon_inv(). Function này chọn Rcon theo giá trị của round_num[3:0]. Thứ tự giá trị Rcon được dùng bị đảo so với quá trình mã hóa.



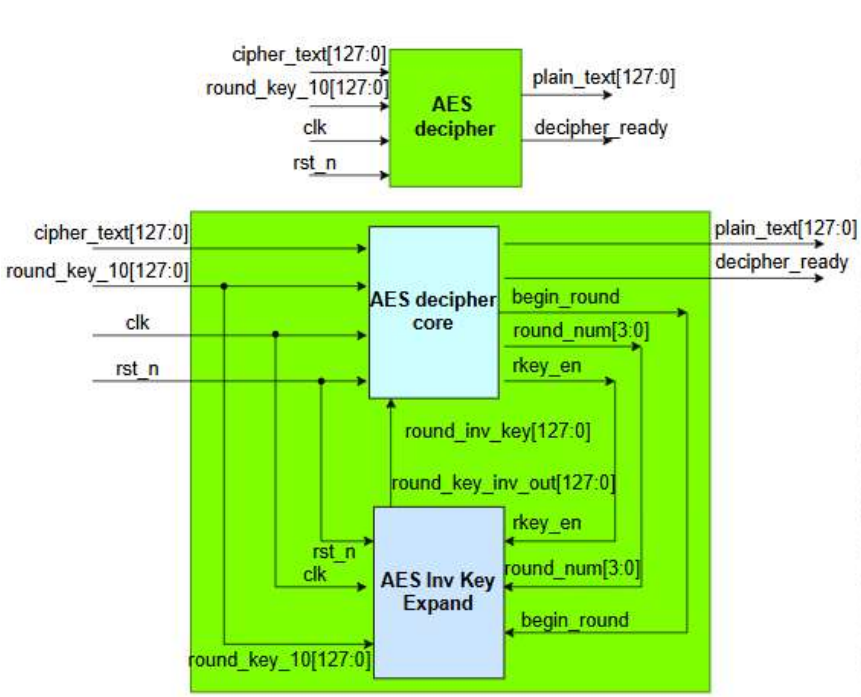
Hình 3.33: Sơ đồ logic của lõi AES Inv Key Expand

Khóa mã đầu tiên đưa vào là round_key_10[127:0] sẽ được gán cho word[3] là bit[127:96], word[2] là bit[95:64], word[1] là bit[63:32] và word[0] là bit[31:0] để tính khóa vòng tiếp theo sẽ thực hiện tương tự như mô tả ở trên. Sau khi có kết quả của w[0] của vòng tiếp theo thì sẽ trải qua các phép biến đổi RotWord, SubWord và InvRcon để tạo ra word[3] của vòng đó. Cuối cùng kết quả của khóa vòng tiếp theo được tạo ra gán vào round_key[127:0], sử dụng FF để chốt giá trị khóa sau mỗi vòng tạo khóa. Kết quả của đầu ra của khóa thứ n sẽ là đầu vào để tính khóa thứ n+1 nên có đường tín hiệu đi

ngược về là round_key_reg[127:0], cứ thực hiện liên tục vậy 10 vòng tạo khóa mã.

3.3.4. Mô tả RTL

Từ thiết kế ở mức thấp, dùng ngôn ngữ Verilog để viết mô tả thiết kế đó, sau đó chạy kiểm thử xem có lỗi gì không, hoạt động có đúng như mong muốn không. Sau đây là đoạn code Verilog mô tả thiết kế module top của lõi AES decipher:




```
`include "AES_Inv_Key_Expand.v"
`include "AES_Decipher_Core.v"
module AES_Decipher(
  input wire clk,
  input wire rst_n,
  input wire [127:0] cipher_text,
  input wire [127:0] round_key_10,

  output wire decipher_ready,
  output wire [127:0] plain_text
);
  wire [3:0] round_num;
  wire rkey_en,begin_round;
  wire [127:0] round_key_out;
  AES_decipher_core aes_decipher_core (
    .clk(clk),
    .rst_n(rst_n),
    .cipher_text(cipher_text),
    .round_key_10(round_key_10),
    .round_key_inv(round_key_out),

    .begin_round(begin_round),
    .plain_text(plain_text),
    .rkey_en(rkey_en),
    .round_num(round_num),
    .decipher_ready(decipher_ready)
  );
  AES_Inv_Key_Expand aes_inv_key_expand (
    .clk(clk),
    .rst_n(rst_n),
    .round_key_10(round_key_10),
    .round_num(round_num),
    .rkey_en(rkey_en),
    .begin_round(begin_round),
    .round_key_out(round_key_out)
  );
endmodule
```

Hình 3.34: Mô tả RTL module top của AES decipher

Sau khi hoàn thành bước viết mã RTL, tiến hành biên dịch và chạy kiểm tra và sửa lỗi, đảm bảo thiết kế chính xác như yêu cầu. Nếu phát hiện lỗi sai thì quay lại kiểu tra thiết kế khối mức thấp để thay đổi thiết kế cho phù hợp. Sau đó tiếp theo chuyển qua bước xác minh chức năng của thiết kế lõi IP mềm AES-128.

Chương 4: XÁC MINH LÕI IP MỀM CHUẨN AES-128

4.1. Giới thiệu chương

Trong chương này, sẽ trình bày chi tiết quy trình xác minh lõi IP mềm chuẩn AES-128, là một bước quan trọng trong việc đảm bảo tính đúng đắn và độ tin cậy của thiết kế phần cứng. Quy trình xác minh được thực hiện theo các bước đã được đề cập ở chương 1, bao gồm: nghiên cứu đặc tả, lập kế hoạch xác minh, xây dựng môi trường kiểm thử, phát triển các thành phần kiểm thử, và cuối cùng là thực hiện mô phỏng để đánh giá chất lượng thiết kế.

Tất cả các hoạt động trong quy trình xác minh đều được xây dựng dựa trên đặc tả kỹ thuật của lõi IP AES-128 đã được phân tích trong chương trước. Đặc tả này mô tả rõ các chức năng chính của lõi, bao gồm quá trình mã hóa (encryption), giải mã (decryption), cùng với các đặc điểm kỹ thuật cụ thể như độ dài khóa, số vòng lặp thuật toán, cách cấu hình.

Mục tiêu của chương này là mô tả cách đã tiếp cận và triển khai từng bước trong quá trình xác minh, từ khâu phân tích yêu cầu đến việc hiện thực hóa một môi trường xác minh hoàn chỉnh, có khả năng phát hiện lỗi và đánh giá độ bao phủ chức năng của lõi IP. Thông qua đó, sẽ minh họa cụ thể cách các thành phần trong môi trường xác minh UVM được sử dụng để kiểm tra hoạt động của lõi AES-128, đảm bảo rằng nó hoạt động chính xác theo đúng đặc tả kỹ thuật đã đề ra.

4.2. Lập kế hoạch xác minh

Kế hoạch xác minh cho lõi AES là một tài liệu quan trọng trong quá trình phát triển và kiểm thử thiết kế, đóng vai trò định hướng và quản lý toàn bộ hoạt động xác minh. Tài liệu này bao gồm việc xác định các chức năng cần kiểm tra, xây dựng các trường hợp kiểm thử cụ thể, thiết lập mục tiêu xác minh, cấu hình môi trường kiểm thử, xác định các kết quả mong đợi cũng như các tiêu chí đánh giá.

Mục đích chính của kế hoạch xác minh là đảm bảo quá trình kiểm thử diễn ra một cách đầy đủ, hiệu quả, nhất quán và có hệ thống. Một kế hoạch được xây dựng chặt chẽ sẽ giúp giảm thiểu rủi ro trong thiết kế, đảm bảo rằng lõi AES hoạt động chính xác theo yêu cầu đặt ra, đồng thời giúp tối ưu hóa quá trình phát hiện và sửa lỗi, từ đó rút ngắn thời gian phát triển sản phẩm.

Trong phạm vi kế hoạch này, việc xác minh lõi AES sẽ tập trung vào ba chức năng quan trọng: mã hóa (encryption), giải mã (decryption) và reset (khởi tạo lại hệ thống). Mỗi chức năng sẽ được kiểm tra thông qua nhiều tình huống khác nhau, đảm bảo rằng

lõi AES có thể hoạt động chính xác trong mọi trường hợp có thể xảy ra. Tổng cộng, sẽ có tám trường hợp kiểm tra (test cases) được xây dựng, bao quát từ các kiểm thử chức năng cơ bản đến các trường hợp đặc biệt và kiểm thử hiệu suất.

Đầu tiên, chức năng reset sẽ được kiểm tra thông qua hai trường hợp: reset trong quá trình mã hóa và reset trong quá trình giải mã. Mục tiêu chính của bài kiểm tra này là đảm bảo rằng khi tín hiệu reset được kích hoạt, lõi AES phải ngay lập tức giải phóng toàn bộ dữ liệu hiện có và đưa về trạng thái ban đầu. Điều này rất quan trọng trong các ứng dụng thực tế, nơi một quá trình mã hóa hoặc giải mã có thể gặp sự cố hoặc cần khởi động lại giữa chừng mà không làm ảnh hưởng đến tính bảo mật và tính nhất quán của dữ liệu. Để cấu hình cho 2 trường hợp này, lõi AES và môi trường kiểm thử sẽ được thiết lập ở chế độ mã hóa hoặc giải mã, sau đó dữ liệu đầu vào (data_in) sẽ được tạo một cách ngẫu nhiên để đánh giá khả năng xử lý với nhiều loại dữ liệu khác nhau. Tín hiệu reset sẽ được kích hoạt tại những thời điểm ngẫu nhiên trong quá trình hoạt động nhằm kiểm tra phản ứng của hệ thống khi bị gián đoạn ở các giai đoạn khác nhau. Để đảm bảo tính ổn định, bài kiểm thử sẽ được lặp lại 10 lần với các điều kiện khác nhau. Kết quả mong đợi sau khi reset là dữ liệu đầu ra (data_out) phải trở về 128'h 0 và tín hiệu finish phải bằng 1, xác nhận rằng quá trình xử lý đã dừng đúng cách và hệ thống sẵn sàng cho các tác vụ tiếp theo.

Bảng 4.1: Lập kế hoạch xác minh chức năng mã hóa trường hợp reset

Reset Test	**Mã hóa**	**AES_RESET_ENCRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra**: Đảm bảo rằng quá trình mã hóa giải phóng dữ liệu sau khi reset. **02. Thiết lập:** - Xác định CIPHER - Dữ liệu ngẫu nhiên - Thời gian ngẫu nhiên cho tín hiệu reset - Lặp lại 10 **03. Kỳ vọng:** data_out =128'h0, finish = 1
	Phương pháp kiểm tra	Kiểm tra dạng sóng
	Sự ưu tiên	Cao
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Thủ công
	Cài đặt	- define CIPHER - aes_multi_seq

		- set reset time - repeat(10)
	Tên trường hợp kiểm tra	aes_test_reset_enc
	Kết quả	PASS

Bảng 4.2: Lập kế hoạch xác minh chức năng giải mã trường hợp reset

Reset Test	**Giải mã**	**AES_RESET_DECRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra**: Đảm bảo rằng quá trình giải mã giải phóng dữ liệu sau khi reset. **02. Thiết lập:** - Xác định DECIPHER - Dữ liệu ngẫu nhiên - Thời gian ngẫu nhiên cho tín hiệu reset - Lặp lại 10 **03. Kỳ vọng:** data_out =128'h0, finish = 1
	Phương pháp kiểm tra	Kiểm tra dạng sóng
	Sự ưu tiên	Cao
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Thủ công
	Cài đặt	- define DECIPHER - aes_multi_seq - set reset time - repeat(10)
	Tên trường hợp kiểm tra	aes_test_reset_dec
	Kết quả	PASS

Tiếp theo là chức năng mã hóa và giải mã, với mỗi chức năng sẽ có 3 trường hợp kiểm thử là kiểm tra chế độ mã hóa/giải mã, kiểm tra mã hóa/giải mã nhiều khối dữ liệu, và kiểm tra với dữ liệu biên.

Trong đó trường hợp kiểm tra chế độ mã hóa/giải mã mục đích để kiểm tra xem lõi AES có thực hiện đúng chức năng mã hóa hoặc giải mã khi được cấu hình hay không. lõi AES và môi trường sẽ được cấu hình là mã hóa hoặc giải mã và thiết lập với khóa và dữ liệu đầu vào cố định, sau đó đầu ra sẽ được so sánh với giá trị đã định trước. Đây là một bước quan trọng nhằm xác minh rằng chế độ mã hóa hoặc giải mã có được kích

hoạt đúng cách và thuật toán hoạt động như mong đợi. Bài kiểm thử này sử dụng phương pháp waveform checking, quá trình xác minh sẽ được thực hiện thủ công.

Bảng 4.3: Lập kế hoạch kiểm tra chức năng mã hóa bằng kiểm tra dạng sóng

Encryption Test	**Lõi mã hóa**	**AES_MODE_DEFINETION_ENCRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra:** Xác minh rằng lõi AES hoạt động chính xác ở chế độ mã hóa khi được xác định **02. Thiết lập:** - Xác định CIPHER - Đặt data_input, giá trị khóa **03. Kỳ vọng:** Dữ liệu thực tế (từ DUT) = giá trị được xác định trước mong đợi.
	Phương pháp kiểm tra	Kiểm tra dạng sóng
	Sự ưu tiên	Cao
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Thủ công
	Cài đặt	- define CIPHER - data_input = 00112233445566778899aabbccddeeff - key = 000102030405060708090a0b0c0d0e0f
	Tên trường hợp kiểm tra	aes_test_definetion_enc
	Kết quả	PASS

Bảng 4.4: Lập kế hoạch kiểm tra chức năng giải mã bằng kiểm tra dạng sóng

Decryption Test	**Lõi giải mã**	**AES_MODE_DEFINETION_DECRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra:** Xác minh rằng lõi AES hoạt động chính xác ở chế độ mã hóa khi được xác định **02. Thiết lập:** - Xác định DECIPHER - Đặt data_input, giá trị khóa **03. Kỳ vọng:** Dữ liệu thực tế (từ DUT) = giá trị

		được xác định trước mong đợi.
	Phương pháp kiểm tra	Kiểm tra dạng sóng
	Sự ưu tiên	Cao
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Thủ công
	Cài đặt	- define DECIPHER - data_input = 69c4e0d86a7b0430d8cdb78070b4c55a - key = 13111d7fe3944a17f307a78b4d2b30c5
	Tên trường hợp kiểm tra	aes_test_definetion_dec
	Kết quả	PASS

Để đánh giá khả năng xử lý liên tục của lõi AES, trường hợp kiểm thử mã hóa/giải mã nhiều khối dữ liệu sẽ thực hiện mã hóa trên nhiều giá trị dữ liệu đầu vào được tạo ngẫu nhiên. Việc kiểm tra với dữ liệu đa dạng giúp phát hiện các lỗi tiềm ẩn trong thuật toán và đảm bảo lõi AES có thể vận hành ổn định trong điều kiện thực tế. Bài kiểm thử này được lặp lại 1000 lần với các giá trị dữ liệu khác nhau. Để đảm bảo độ chính xác, đầu ra của lõi AES sẽ được so sánh với golden data từ một mô hình tham chiếu. Phương pháp xác minh sử dụng scoreboard checking, giúp tự động so sánh và phát hiện sai sót, đảm bảo quá trình kiểm thử diễn ra hiệu quả.

Bảng 4.5: Lập kế hoạch kiểm tra chức năng mã hóa bằng kiểm tra scoreboard

Encryption Test	**Lõi mã hóa**	**AES_ENCRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra:** Xác minh chức năng mã hóa của lõi IP khi tạo ngẫu nhiên nhiều chuỗi dữ liệu và khóa. **02. Cài đặt:** - Xác định CIPHER - Ngẫu nhiên hóa dữ liệu - Lặp lại 10 **03. Kỳ vọng:** Dữ liệu thực tế (từ DUT) = dữ liệu tham chiếu (từ mô hình tham chiếu)

	Phương pháp kiểm tra	Kiểm tra scoreboard
	Sự ưu tiên	Cao
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Tự động
	Cài đặt	- define CIPHER - aes_multi_seq
	Tên trường hợp kiểm tra	aes_test_continuous_enc
	Kết quả	PASS

Bảng 4.6: Lập kế hoạch kiểm tra chức năng giải mã bằng kiểm tra scoreboard

Decryption Test	**Lõi mã hóa**	**AES_DECRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra:** Xác minh chức năng giải mã của lõi IP khi tạo ngẫu nhiên nhiều chuỗi dữ liệu và khóa. **02. Cài đặt:** - Xác định DECIPHER - Ngẫu nhiên hóa dữ liệu - Lặp lại 10 **03. Kỳ vọng:** Dữ liệu thực tế (từ DUT) = dữ liệu tham chiếu (từ mô hình tham chiếu)
	Phương pháp kiểm tra	Kiểm tra scoreboard
	Sự ưu tiên	Cao
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Tự động
	Cài đặt	- define DECIPHER - aes_multi_seq
	Tên trường hợp kiểm tra	aes_test_continuous_dec
	Kết quả	PASS

Cuối cùng, trường hợp kiểm thử với dữ liệu biên được thiết kế để đánh giá khả năng xử lý các giá trị đầu vào đặc biệt của lõi AES. Một số dạng dữ liệu có thể gây ra lỗi hoặc ảnh hưởng đến hiệu suất của thuật toán, do đó cần kiểm tra xem hệ thống có thể xử lý chính xác những trường hợp này hay không. Hệ thống sẽ được cung cấp các dữ liệu đầu vào có giá trị lớn nhất, giá trị nhỏ nhất, giá trị giữa, xen kẽ 1010 hoặc 0101. Việc kiểm tra này giúp đảm bảo lõi AES có thể xử lý mọi loại dữ liệu, kể cả các trường hợp đặc biệt mà thuật toán có thể gặp khó khăn.

Bảng 4.7: Lập kế hoạch xác minh chức năng mã hóa với các trường hợp đặc biệt

Encryption Test	**Lõi mã hóa**	**AES_BOUNDARY_DATA_ENCRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra:** Xác minh dữ liệu đầu ra chính xác như mong đợi với dữ liệu đầu vào **02. Thiết lập:** - Xác định CIPHER - Gán dữ liệu với các giá trị cụ thể **03. Kỳ vọng:** Dữ liệu thực tế (từ DUT) = dữ liệu tham chiếu (từ mô hình tham chiếu)
	Phương pháp kiểm tra	Kiểm tra scoreboard
	Sự ưu tiên	Thấp
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Thủ công
	Cài đặt	- define CIPHER - data_input[0], key[0] = 128'h00000000000000000000000000000000 - data_input[1], key[1] = 128'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF - data_input[2], key[2] = 128'hAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA - data_input[3], key[3] = 128'h55555555555555555555555555555555
	Tên trường hợp	aes_test_special_data_enc

	kiểm tra	
	Kết quả	PASS

Bảng 4.8: Lập kế hoạch xác minh chức năng giải mã với các trường hợp đặc biệt

Decryption Test	**Lõi giải mã**	**AES_BOUNDARY_DATA_DECRYPTION**
	Mô tả	**01. Mục tiêu kiểm tra:** Xác minh dữ liệu đầu ra chính xác như mong đợi với dữ liệu đầu vào **02. Thiết lập:** - Xác định DECIPHER - Gán dữ liệu với các giá trị cụ thể **03. Kỳ vọng:** Dữ liệu thực tế (từ DUT) = dữ liệu tham chiếu (từ mô hình tham chiếu)
	Phương pháp kiểm tra	Kiểm tra scoreboard
	Sự ưu tiên	Thấp
	Đánh giá bởi kỹ sư cấp cao	Có
	Được tạo	Thủ công
	Cài đặt	- define DECIPHER - data_input[0], key[0] = 128'h00000000000000000000000000000000 - data_input[1], key[1] = 128'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF - data_input[2], key[2] = 128'hAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA - data_input[3], key[3] = 128'h55555555555555555555555555555555
	Tên trường hợp kiểm tra	aes_test_special_data_dec
	Kết quả	PASS

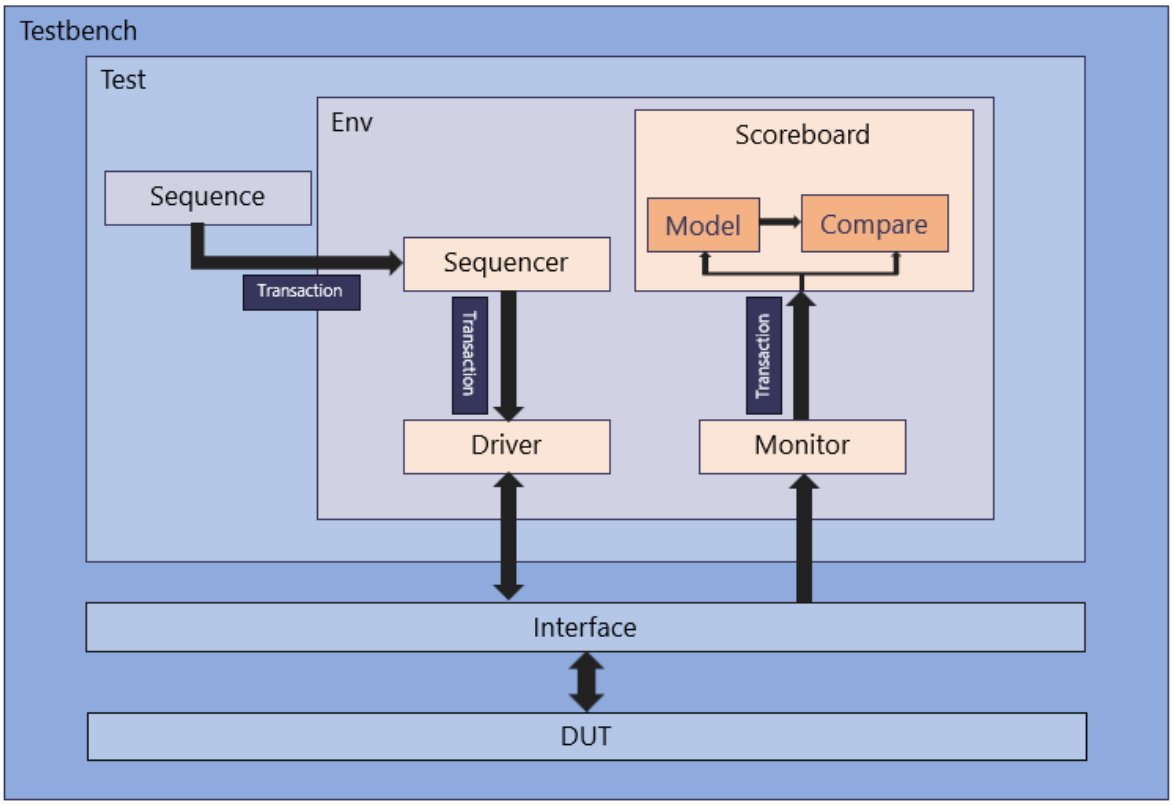
4.3. Xây dựng môi trường xác minh

Việc xây dựng môi trường UVM trong DV (Design Verification) là cần thiết để đảm bảo quá trình xác minh diễn ra hiệu quả, có hệ thống, dễ bảo trì và mở rộng, đồng

thời đáp ứng các yêu cầu chất lượng ngày càng cao trong thiết kế vi mạch hiện đại. Hình 4.1 là môi trường xác minh được xây dựng cho lõi AES. Trong đó, ở tầng cao nhất, test khởi tạo quá trình kiểm thử bằng cách gọi sequence, chuỗi các transaction được gửi đến sequencer. Sequencer điều phối transaction cho driver, nơi chuyển đổi chúng thành tín hiệu điều khiển DUT thông qua interface. Trong khi đó, monitor quan sát tín hiệu từ DUT, chuyển thành transaction gửi đến scoreboard. Scoreboard so sánh kết quả thực tế với dữ liệu mong đợi từ mô hình. Nếu có sai khác, hệ thống sẽ ghi nhận lỗi. Mô hình này giúp kiểm thử hiệu quả, dễ bảo trì và hỗ trợ tái sử dụng các thành phần trong testbench.

4.3.1. Giao diện (Interface)

Giao diện được viết bằng ngôn ngữ System Verilog, là một cấu trúc được sử dụng để nhóm các tín hiệu liên qua lại với nhau, giúp đơn giản hóa việc kết nối và quản lý các tín hiệu giữa các module khác nhau trong một thiết kế phần cứng. Ở môi trường này interface được thiết kế để hỗ trợ giao tiếp giữa các thành phần của môi trường và DUT (module AES).



Hình 4.1: Môi trường xác minh cho lõi AES

Giao diện AES được khai báo với hai tín hiệu là xung clk (clock) và rst_n (reset), được sử dụng để đồng bộ hóa và đặt lại giao diện, data_input là dữ liệu gốc cần mã hóa hoặc dữ liệu mã hóa cần giải mã, data_output là kết quả sau khi quá trình mã hóa hoặc

giải mã hoàn tất, key là khóa mã hóa dùng cho thuật toán AES và tín hiệu finished báo hiệu hoàn thành quá trình xử lý.

4.3.2. Giao dich (Transaction)

Transaction là lớp đại diện cho một đơn vị dữ liệu được truyền giữa các thành phần của testbench, thường được gọi là sequence item. Transaction đóng vai trò trung gian giữa sequence, driver, và monitor, đồng thời mang theo các thông tin cần thiết để kiểm thử DUT.

Transaction được khai báo với 3 trường là data_input, key, data_output. Trong đó data_input, key là hai trường được định nghĩa ngẫu nhiên (randomizable), chúng tương ứng với dữ liệu đầu vào và khóa mã hóa dùng trong quá trình kiểm thử AES, khi được tạo trong sequence, các giá trị này sẽ được gán tự động hoặc theo ràng buộc. Data_output là dữ liệu đầu ra từ DUT sau quá trình mã hóa/giải mã, trường này không cần tạo ngẫu nhiên vì nó được DUT tạo ra, và thường được điền bởi monitor.

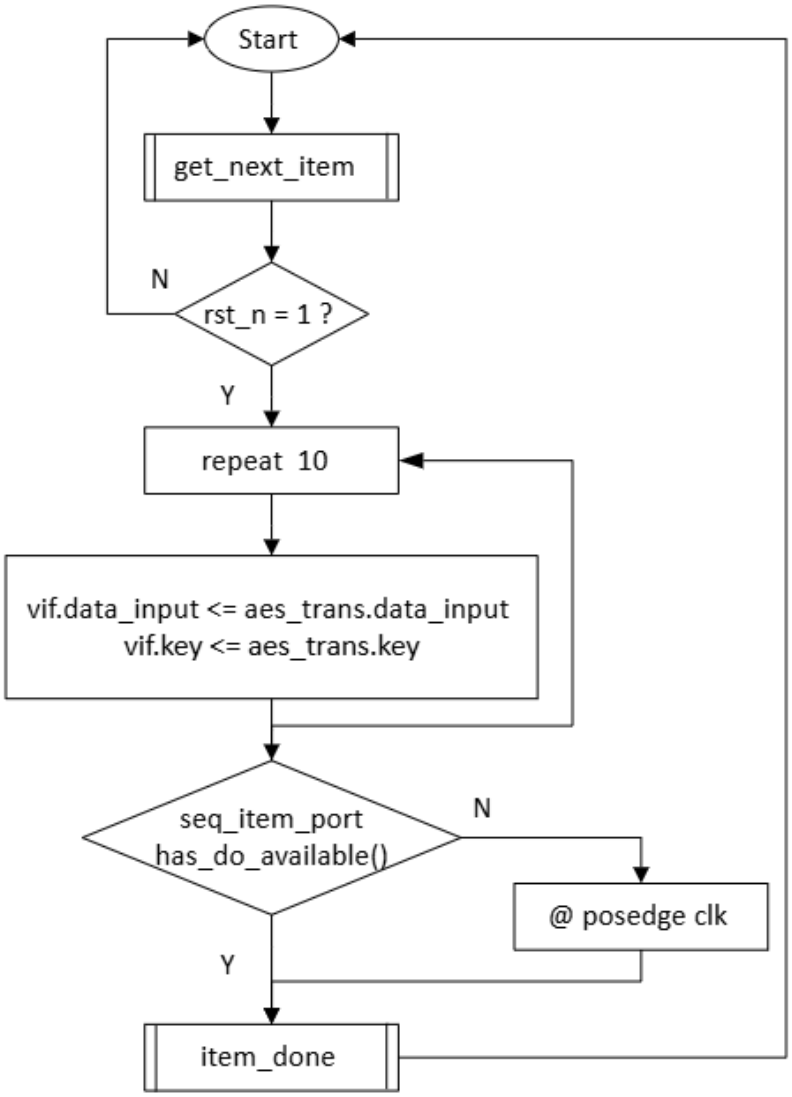
4.3.3. Chuỗi lệnh (Sequence)

Trong môi trường xác minh theo chuẩn UVM sequence là thành phần chịu trách nhiệm sinh ra các giao dịch (transaction) được gửi đến driver để kích thích thiết kế (DUT). Đối với IP mã hóa AES, các sequence sẽ đóng vai trò tạo ra các bộ dữ liệu đầu vào (plaintext) và khóa mã hóa (key), sau đó quan sát và so sánh kết quả đầu ra để xác minh tính đúng đắn của thuật toán.

Trong đồ án này, các sequence được thiết kế dựa trên một lớp cơ sở chung aes_base_sequence, sau đó mở rộng thành nhiều lớp con nhằm kiểm thử ở các cấp độ khác nhau, bao gồm: kiểm thử đơn lẻ (single test), kiểm thử ngẫu nhiên (random test), và kiểm thử trường hợp đặc biệt (special pattern test). Mỗi nhóm đều được thực hiện cho cả hai chức năng mã hóa (encryption) và giải mã (decryption).

4.3.4. Trình điều khiển (Driver)

Driver đóng vai trò trung gian giữa môi trường kiểm tra (testbench) và thiết kế (Design Under Test – DUT), đảm nhận nhiệm vụ chuyển đổi các giao dịch ở cấp độ trừu tượng (transaction-level) sang tín hiệu cụ thể (pin-level) điều khiển vào DUT. Trong hệ thống kiểm tra AES, driver được thiết kế để xử lý các giao dịch mã hóa và giải mã dựa trên thuật toán AES-128.



Hình 4.2: Sơ đồ luồng hoạt động của AES driver

Bắt đầu xử lý giao dịch: Driver sử dụng phương thức get_next_item() để nhận một transaction từ sequence. Transaction này chứa các trường dữ liệu như data_input, key và sau đó DUT sẽ xử lý để tạo ra data_output.

Chờ tín hiệu reset: Trước khi đưa dữ liệu vào DUT, driver sẽ đợi cho đến khi tín hiệu rst_n được nhả (logic 1), đảm bảo DUT đã thoát trạng thái khởi tạo và sẵn sàng hoạt động.

Cung cấp dữ liệu vào DUT: Ở cạnh lên tiếp theo của xung nhịp xung clk, driver gán giá trị data_input và key từ transaction vào các tín hiệu điều khiển tương ứng trong interface. Việc sử dụng @posedge clk đảm bảo tính đồng bộ khi truyền tín hiệu vào. Vì DUT thực hiện mã hóa hoặc giải mã trong 10 chu kỳ nên sẽ truyền dữ liệu vào DUT trong 10 chu kỳ đồ hồ bằng lệnh repeat(10).

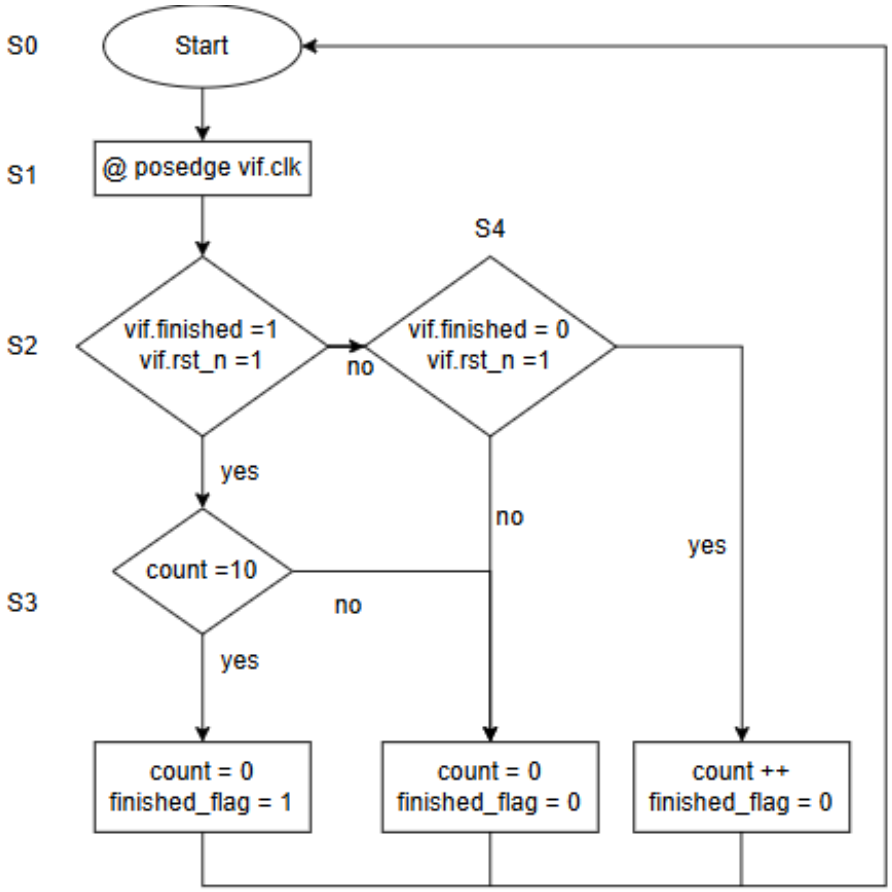
Kiểm tra sự sẵn sàng nhận giao dịch tiếp theo: Driver tiếp tục chờ đợi đến khi seq_item_port.has_do_available() trả về true, tức là sequence đã sẵn sàng giao thêm

transaction.

Kết thúc giao dịch: Khi quá trình truyền dữ liệu kết thúc, driver gọi item_done() để báo hiệu transaction đã hoàn tất xử lý, cho phép sequence gửi transaction tiếp theo.

4.3.5. Bộ giám sát (Monitor)

Trong kiến trúc UVM, monitor có nhiệm vụ quan sát tín hiệu từ DUT (Design Under Test) một cách bị động (không điều khiển), sau đó thu thập thông tin và truyền các transaction về scoreboard thông qua analysis port để phục vụ so sánh và đánh giá kết quả. Trong đồ án này, monitor của lõi AES được thiết kế gồm hai nhiệm vụ chính: kiểm tra tín hiệu hoàn tất (check_finish_signal) và thu thập, gửi dữ liệu (collect_send_data).



Hình 4.3: Sơ đồ luồng hoạt động của nhiệm vụ check_finish_signal

Sơ đồ Hình 4.3 thể hiện nhiệm vụ theo dõi tín hiệu finished do DUT xuất ra để xác định thời điểm quá trình mã hóa hoặc giải mã hoàn tất. Việc xác định chính xác thời điểm kết thúc giúp quá trình thu thập dữ liệu được đồng bộ và chính xác.

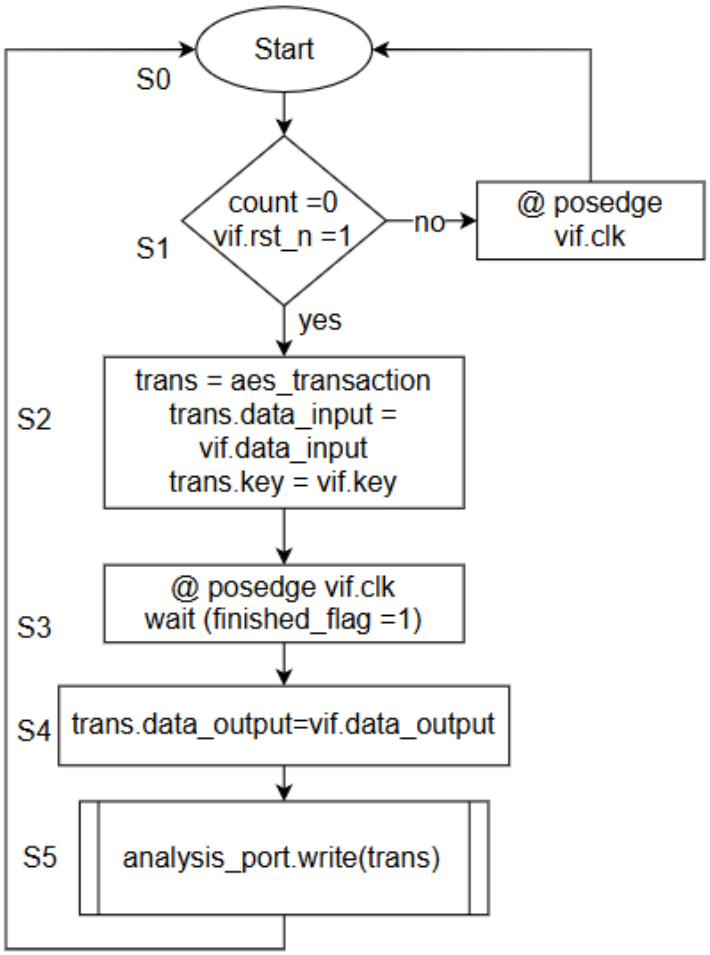
Quy trình hoạt động được mô tả qua sơ đồ trạng thái như sau:

Tại mỗi chu kỳ cạnh lên của xung nhịp xung clk, monitor kiểm tra xem hệ thống đã thoát trạng thái reset (rst_n == 1) hay chưa.

Nếu tín hiệu finished == 1 đồng thời rst_n == 1, biến đếm count sẽ được theo dõi. Nếu count đạt giá trị 10, cờ finished_flag được kích hoạt (finished_flag = 1) và count được đặt lại.

Nếu finished == 0 và hệ thống vẫn đang hoạt động (rst_n == 1), count sẽ tăng dần theo từng chu kỳ clock.

Nếu bất kỳ điều kiện nào không thỏa mãn, count sẽ được reset về 0 và finished_flag cũng sẽ về 0.



Hình 4.4: Sơ đồ luồng hoạt động của nhiệm vụ collect_send_data

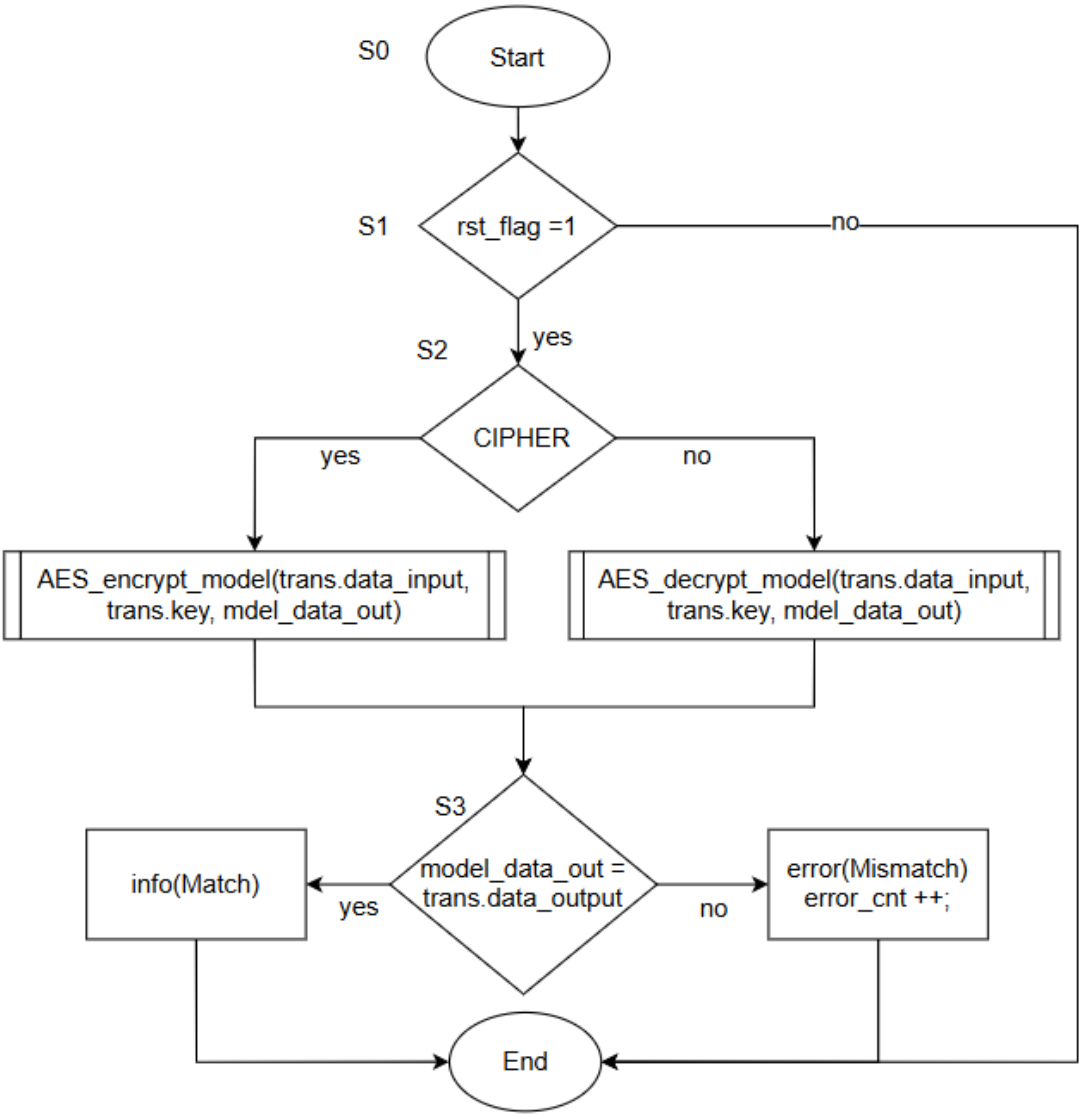
Sau khi phát hiện quá trình mã hóa hoặc giải mã đã hoàn tất, monitor sẽ tiến hành thu thập dữ liệu từ DUT và gửi về scoreboard để so sánh, luồng hoạt động của nhiệm vụ collect_send_data được thể hiện ở Hình 4.4.

Đầu tiên Monitor chờ điều kiện hệ thống ổn định (rst_n == 1) và biến count == 0 (bắt đầu phiên kiểm thử mới). Một transaction mới được khởi tạo từ lớp aes_transaction, các trường data_input và key được gán giá trị từ DUT thông qua interface. Monitor đợi cạnh lên kế tiếp của xung clk và chờ cho đến khi cờ finished_flag == 1, điều này bảo đảm dữ liệu đầu ra đã sẵn sàng. Trường data_output của transaction được gán từ output

thực tế của DUT. Cuối cùng, transaction được gửi tới scoreboard thông qua analysis_port.write(trans) để thực hiện so sánh.

4.3.6. Bảng đối chiếu (Scoreboard)

Scoreboard đóng vai trò là thành phần chính dùng để kiểm tra tính đúng đắn của dữ liệu đầu ra từ DUT. Scoreboard AES thực hiện so sánh giữa dữ liệu thực tế từ DUT và dữ liệu mong đợi được tạo ra từ mô hình tham chiếu (reference model). Nếu kết quả khớp, hệ thống ghi nhận một phép kiểm thành công; nếu sai khác, scoreboard sẽ cảnh báo và ghi nhận lỗi.



Hình 4.5: Luồng hoạt động của Scoreboard

Quy trình xử lý dữ liệu trong scoreboard được trình bày chi tiết qua sơ đồ luồng như Hình 4.5. Đầu tiên Scoreboard chờ nhận một transaction từ monitor thông qua analysis port. Transaction này bao gồm data_input, key, data_output. Sau đó kiểm tra tín hiệu reset (rst_flag = 1?), nếu hệ thống đang trong trạng thái reset (rst_flag == 1),

scoreboard sẽ không thực hiện kiểm tra để tránh sai lệch do dữ liệu chưa ổn định. Tiếp theo là kiểm tra chế độ mã hóa hay giải mã, nếu chế độ là mã hóa (CIPHER == 1), scoreboard gọi mô hình AES_encrypt_model (trans.data_input, trans.key, model_data_out), nếu chế độ là giải mã (CIPHER == 0), scoreboard gọi AES_decrypt_model (trans.data_input, trans.key, model_data_out). Sau khi thu được model_data_out từ mô hình, scoreboard thực hiện so sánh với trans.data_output (kết quả thực tế từ DUT). Nếu hai kết quả khớp nhau, scoreboard ghi log thông tin "Match" để báo thành công, nếu không khớp, scoreboard ghi log lỗi "Mismatch" và tăng biến đếm lỗi error_cnt.

Chương 5: KẾT QUẢ MÔ PHỎNG VÀ ĐÁNH GIÁ

5.1. Giới thiệu chương

Chương này thực hiện mô phỏng hoạt động của lõi IP ở các chức năng mã hóa và giải mã. Kiểm tra các trường hợp đặc biệt và quan sát kết quả dạng sóng để xác minh tính đúng đắn của lõi AES.

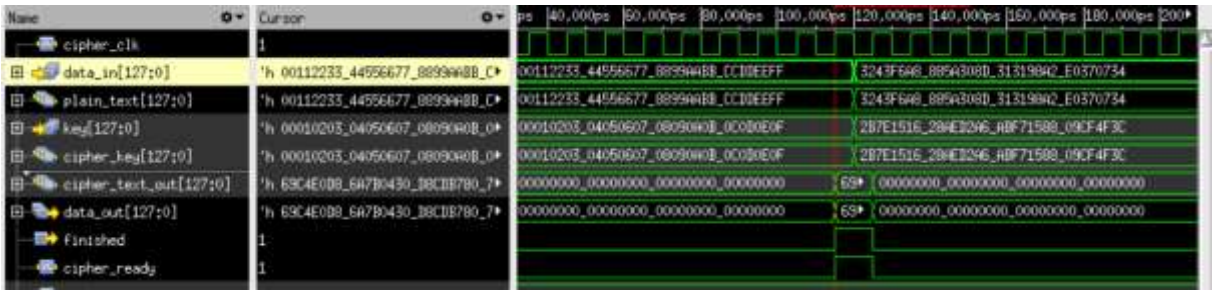
5.2. Mô phỏng và đánh giá thiết kế đặc tả

Phần này là kết quả mô phỏng kiểm tra thiết kế mô tả RTL ở chương 3, thực hiện kiểm tra từng chức năng mã hóa và giải mã.

5.2.1. Mô phỏng thiết kế RTL lõi AES cipher

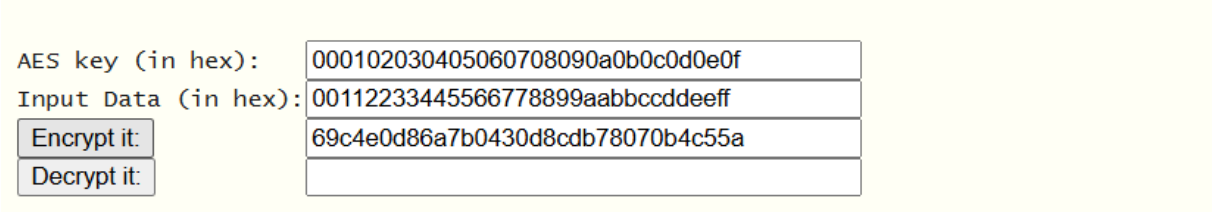
Sau khi hoàn thành mô tả RTL cho thiết kế, sẽ thực hiện mô phỏng và kiểm tra lỗi của thiết kế. Phần này sẽ trình bày kết quả của thiết kế mô tả RTL của khối mức cao lõi AES mã hóa.

Sử dụng công cụ Cadence Xcelium để chạy mô phỏng, ban đầu sẽ cấu hình lõi IP hoạt động chức năng mã hóa, sử dụng ngôn ngữ Verilog để tạo testbench cho lõi IP và tạo dữ liệu gán cho giá trị đầu vào là data_input và key. Kết quả đầu ra sẽ là giá trị data_output và tín hiệu finished báo hoàn thành quá trình mã hóa.



Hình 5.1: Kết quả khi lõi IP hoạt động ở chức năng mã hóa

Trong Hình 5.1, thực hiện tạo dữ liệu data_in và key, dữ liệu data_in sẽ được gán cho plain_text và key sẽ được gán cho cipher_key, kết quả đầu ra là dữ liệu được mã hóa cipher_text_out gán cho data_out, tín hiệu cipher_ready được gán cho finished. Với dữ liệu cần mã hóa và khóa như đã tạo, sử dụng ứng dụng kiểm tra thì thu được kết quả khớp với kết quả của lõi IP, điều này chứng tỏ lõi IP hoạt động chức năng mã hóa đúng như mong đợi, có thể triển khai bước tiếp theo trong quy trình thiết kế ASIC.

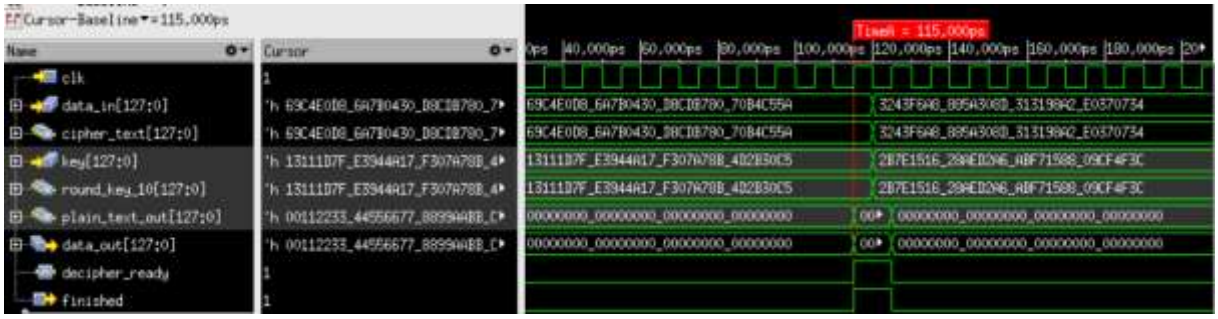


Hình 5.2: Dữ liệu mã hóa Encrypt sử dụng ứng dụng kiểm tra

5.2.2. Mô phỏng thiết kế RTL lõi AES decipher

Tiếp theo sẽ kiểm tra chức năng giải mã của lõi IP, cũng thực hiện tương tự như phần giải mã. Khi thực hiện chức năng giải mã thì chức năng còn lại không hoạt động. Dữ liệu đầu vào data_in lúc này sẽ được gán cho cipher_text và key sẽ được gán cho round_key_10, kết quả đầu ra là plain_text sẽ được gán cho data_out và decipher_ready sẽ được gán cho finished.

Tương tự cũng sử dụng công cụ Cadence để kiểm tra và thu được kết quả như hình bên dưới:



Hình 5.3: Kết quả khi lõi IP hoạt động ở chức năng giải mã

Sử dụng công cụ kiểm tra cũng thu được kết quả tương tự như lõi IP hoạt động chức năng giải mã. Do ứng dụng kiểm tra nhận key đầu vào của quá trình mã hóa sau đó biến đổi khóa nhiều vòng để tạo khóa vòng 10 làm đầu vào cho quá trình giải mã nên khi kiểm tra sẽ nhập key chung cho 2 chức năng. Chức năng giải mã hoạt động đúng, đảm bảo bước mô tả thiết kế đã đúng, có thể qua bước tiếp theo là xác minh thiết kế.

Hình 5.4: Dữ liệu giải mã Decrypt sử dụng ứng dụng kiểm tra

5.3. Mô phỏng xác minh chức năng

5.3.1. Giới thiệu về công cụ sử dụng trong phần xác minh thiết kế

Cadence Xcelium là một công cụ mô phỏng logic số hiệu suất cao được phát triển

bởi Cadence Design Systems, được sử dụng rộng rãi trong thiết kế và xác minh vi mạch số (digital design verification). Xcelium hỗ trợ nhiều ngôn ngữ mô tả phần cứng như Verilog, SystemVerilog, VHDL và hỗ trợ tích hợp với các testbench nâng cao như UVM. Công cụ này nổi bật với khả năng mô phỏng đa luồng (multi-threaded simulation), cho phép tận dụng sức mạnh xử lý của CPU hiện đại để tăng tốc quá trình mô phỏng. Ngoài ra, Xcelium cũng hỗ trợ xác minh formal và mô phỏng hỗn hợp giữa analog và digital trong các hệ thống SoC phức tạp. Đây là một giải pháp toàn diện dành cho các kỹ sư thiết kế và xác minh IC nhằm kiểm tra chức năng, hiệu năng và độ tin cậy của thiết kế trước khi chế tạo silicon.

5.3.2. Mô phỏng và kết quả xác minh chức năng mã hóa

Xác minh chức năng hoạt động của tín hiệu reset:

Tín hiệu reset không đồng bộ và hoạt động mức thấp, khi tín hiệu rst_n = 0 tác động thì dữ liệu hiện tại đang mã hóa sẽ bị xóa về 0. Sau khi rst_n = 1 thì sẽ ngưng tác động và quá trình mã hóa sẽ lấy dữ liệu ngay tại thời điểm sườn lên xung clk của chu kì xung clk tiếp theo đó.

Hình 5.5: Dữ liệu mã hóa khi có tác động của tín hiệu reset

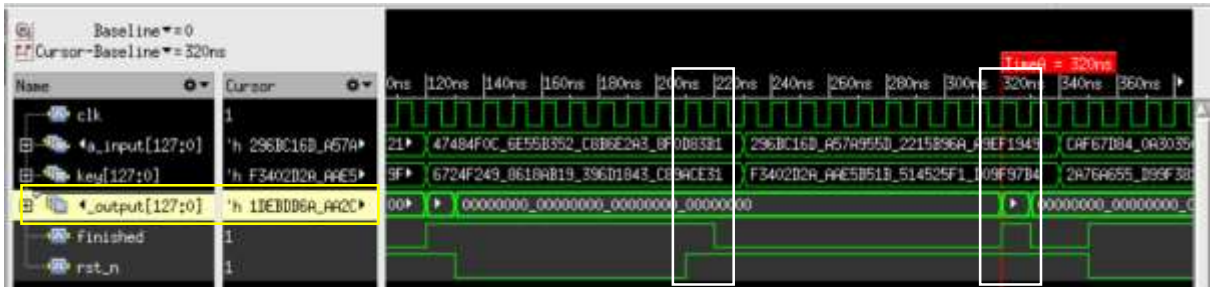
Nhìn vào dạng sóng ban đầu reset chưa tác động thì tín hiệu rst_n = 1, khi có tác động tại thời điểm 130ns thì tín hiệu rst_n = 0, lúc này chức năng mã hóa sẽ bị xóa về trạng thái ban đầu, dữ liệu đang mã hóa hiện tại sẽ bị xóa. Tại thời điểm 210ns, tín hiệu rst_n = 1, sau đó tại thời điểm 220ns tín hiệu finished = 0, bộ mã sẽ tiếp tục mã hóa, và lúc này, dữ liệu mã hóa sẽ bắt tại thời điểm 220ns với:

data_in[127:0] = 128'h 47484f0c6e55b352c8b6e2a38f0d83b1 và

key[127:0] = 128'h6724f2498618ab19396d1843c89ace31

Tại thời điểm 320ns sẽ tạo ra kết quả được mã hóa là:

data_out[127:0] = 128’h1debdd6aaa2c30fd0e5f237468349634.



Hình 5.6: Dữ liệu được mã hóa sau 10 chu kì xung clk

Từ kết quả kiểm tra trên thì thấy tín hiệu reset hoạt động đảm bảo đúng chức năng mong muốn khi thiết kế, dữ liệu bị xóa khi có tín hiệu tác động và hoạt động lại ngay sau khi ngừng tác động.

Xác minh đối với dữ liệu mã hóa là một gói dữ liệu:

Sử dụng tool Cadence Xcelium để thực hiện chạy mô phỏng, với dữ liệu cần mã hóa và khóa lần lượt là:

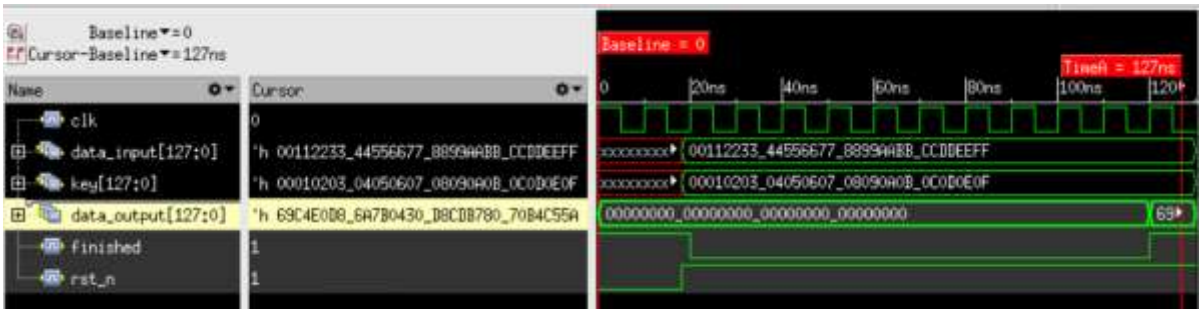
data_in[127:0] = 128'h00112233445566778899aabbccddeeff;

key[127:0] = 128'h000102030405060708090a0b0c0d0e0f;

Dùng công cụ của nhà cung cấp kiểm tra thì kết quả dữ liệu mã hóa mong đợi là:

data_out[127:0] = 128’h69c4e0d86a7b0430d8cdb78070b4c55a;

Thực hiện mô phỏng để lấy kết quả và xem dạng sóng của dữ liệu ra, thu được kết quả mô phỏng như hình bên dưới:



Hình 5.7: Kết quả trường hợp kiểm tra lõi mã hóa một gói dữ liệu

Kết quả cho thấy lõi IP hoạt động chức năng mã hóa đã chính xác, dữ liệu mã hóa giống như dữ liệu mong đợi.

Xác minh đối với dữ liệu mã hóa là nhiều gói dữ liệu:

Tiếp tục với trường hợp kiểm tra thứ 2, khi dữ liệu cần mã hóa sẽ là một chuỗi nhiều gói dữ liệu ngẫu nhiên được đưa vào liên tục. Khi thực hiện kiểm tra trường hợp này, lõi IP cần phải thực hiện mã hóa liên tục, từng thời điểm yêu cầu các chức năng trong lõi thực hiện bắt đúng dữ liệu.

Dữ liệu cần mã hóa và khóa sẽ là 10 gói và một gói là 128 bit, mỗi một gói thì sẽ có một khóa khác nhau, dùng ngôn ngữ SystemVerilog để thực hiện.

Với dữ liệu vào, khóa được tạo ngẫu nhiên liên tục và dữ liệu ra mong muốn thu được lần lượt là:

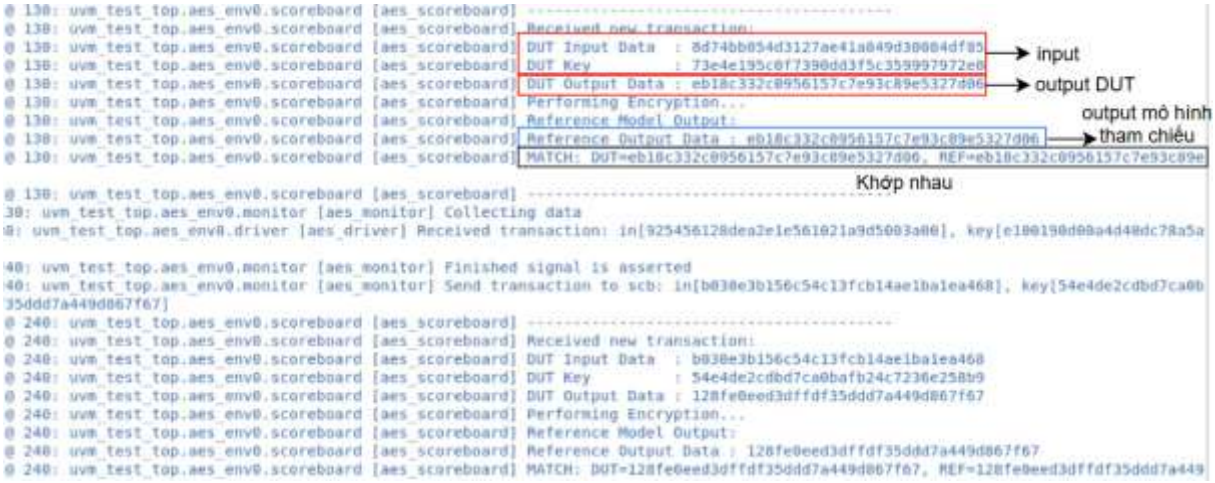
Bảng 5.1: Kiểm tra chức năng mã hóa lõi IP với các gói dữ liệu liên tục

Gói 1	data_in[127:0]	128’h8d74bb054d3127ae41a849d30084df85
	key[127:0]	128’h73e4e195c0f7390dd3f5c359997972e0
	data_out[127:0]	128’heb18c332c0956157c7e93c89e5327d06
Gói 2	data_in[127:0]	128’h b030e3b156c54c13fcb14ae1ba1ea468
	key[127:0]	128’h54e4de2cdbd7ca0bafb24c7236e258b9
	data_out[127:0]	128’h128fe0eed3dffdf35ddd7a449d867f67
Gói 3	data_in[127:0]	128’h925456128dea2e1e561021a9d5003a00
	key[127:0]	128’he100190d00a4d40dc78a5a203ee7223b
	data_out[127:0]	128’h08f60c71a624f2e2959fd83da036d86e
Gói 4	data_in[127:0]	128’h33df1229f29fcdcf4dc4ce2b5129a04d
	key[127:0]	128’h183692382f5e57121b7dec62b187cf68
	data_out[127:0]	128’h5c3ac1373e84e86947e151956116d1bf
Gói 5	data_in[127:0]	128’h 86f640ea8675d90e3064cf932e9ad74f
	key[127:0]	128’h9a7fd7f27963cd9401e7db3621371e4f
	data_out[127:0]	128’h8423de58795d9b2d42a6e4660e49be88
Gói 6	data_in[127:0]	128’h8b99e256496c4fdbfdf025e36d53df06
	key[127:0]	128’h67dbea3bdeb537937ac8269c8df50ef1
	data_out[127:0]	128’h53cc256768e932fd7d8a5519266dfd8e
Gói 7	data_in[127:0]	128’h4fa4cd7839f3844d69d151ec0d54b772
	key[127:0]	128’he0533acd4df31a962fc3f696654ee33d
	data_out[127:0]	128’hbf571df2162b527252a4e35d971caa94
Gói 8	data_in[127:0]	128’hd04094bbc1fabcf018efc70a0e9d6094
	key[127:0]	128’h1d6b79b3c909f3c5e0a34cfc35049428
	data_out[127:0]	128’ha6a52c80f4845b59cb8c8e7b8587ae28
Gói 9	data_in[127:0]	128’h0268cea979226121b2f9910f712dda6b
	key[127:0]	128’ha59685285f6cc07123f8fff401c8e6ce
	data_out[127:0]	128’hea88b82c9a6620aef48ddcc7158183b3
Gói 10	data_in[127:0]	128’hf3f8524c5ddac2f8eb5930ce350624f7
	key[127:0]	128’hd8dccee7ffbc0620a36a378039291d1d
	data_out[127:0]	128’h127cd1a07d68f16af55b6e51183793f6

Thực hiện mô phỏng để kiểm tra kết quả, sử dụng scoreboard để so sánh dữ liệu của mô hình và dữ liệu của thiết kế. Sau khi kiểm tra thì tất cả đều đúng, dữ liệu mã hóa của thiết kế trùng khớp với mô hình được xây dựng trước đó.

```
** Report counts by severity
UVM_INFO :  150
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[TEST_DONE]     1
[aes_driver]    10
[aes_monitor]    33
[aes_multi_en]     1
[aes_scoreboard]   102
[aes_test_continuous_enc]     2
Simulation complete via $finish(1) at time 1120 NS + 59
/mnt/CADENCE/xcelium_2209/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
xcelium> database -close waves
xcelium> exit
TOOL:   xrun    22.09-s004: Exiting on Apr 16, 2025 at 14:18:17 +07  (total: 00:00:03)
```

Hình 5.8: Kết quả trường hợp kiểm tra nhiều gói dữ liệu liên tục



Hình 5.9: Minh hoạ dữ liệu ra DUT của gói 1 so với mô hình tham chiếu

Hình 5.10: Kiểm tra dạng sóng của nhiều gói dữ liệu

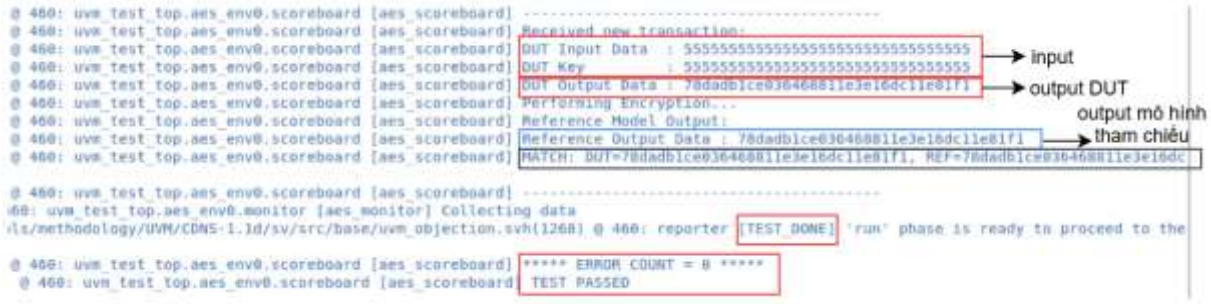
Khi truyền liên tục 10 gói dữ liệu, cứ sau 10 chu kì xung clk, thì tín hiệu finished = 1, sẽ xuất dữ liệu đầu ra, tiếp theo sẽ thực hiện mã hóa gói dữ liệu tiếp theo liên tục cho tới khi hoàn thành. Kết quả cho thấy lõi IP hoạt động chức năng mã hóa đã chính xác, dữ liệu mã hóa giống như dữ liệu mong đợi.

Xác minh đối với dữ liệu mã hóa là các gói kí tự đặc biệt:

Để đảm bảo bao phủ hết toàn bộ trường hợp kiểm tra của lõi, thực hiện đưa dữ liệu mã hóa và khóa với các giá trị đặc biệt. Dữ liệu mã hóa và khóa của trường hợp kiểm tra này như sau:

Gói 1	data_in[127:0]	128'h00000000000000000000000000000000
	key[127:0]	128'h00000000000000000000000000000000
	data_out[127:0]	128'h66e94bd4ef8a2c3b884cfa59ca342b2e
Gói 2	data_in[127:0]	128'hffffffffffffffffffffffffffffffff
	key[127:0]	128'hffffffffffffffffffffffffffffffff
	data_out[127:0]	128'hbcbf217cb280cf30b2517052193ab979
Gói 3	data_in[127:0]	128'haaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
	key[127:0]	128'haaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
	data_out[127:0]	128'h78498cde07d82a92b6a07efa970a854d
Gói 4	data_in[127:0]	128'h55555555555555555555555555555555
	key[127:0]	128'h55555555555555555555555555555555
	data_out[127:0]	128'h78dadb1ce036468811e3e16dc11e81f1

Thực hiện kiểm tra xác minh hoàn tất, tất cả đều PASSED, thì thu được các kết quả chính xác, dữ liệu ra của thiết kế trùng khớp với của mô hình.

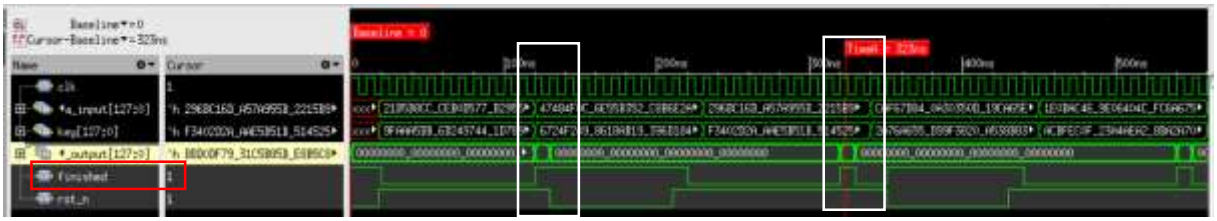


Hình 5.11: Kết quả kiểm tra các trường hợp đặc biệt

5.3.3. Mô phỏng và kết quả xác minh chức năng giải mã

Xác minh chức năng hoạt động của tín hiệu reset:

Tín hiệu reset tương tự như trong chức năng mã hóa, thực hiện chạy mô phỏng để kiểm tra chức năng của lõi IP.



Hình 5.12: Dữ liệu giải mã khi có tác động của tín hiệu reset

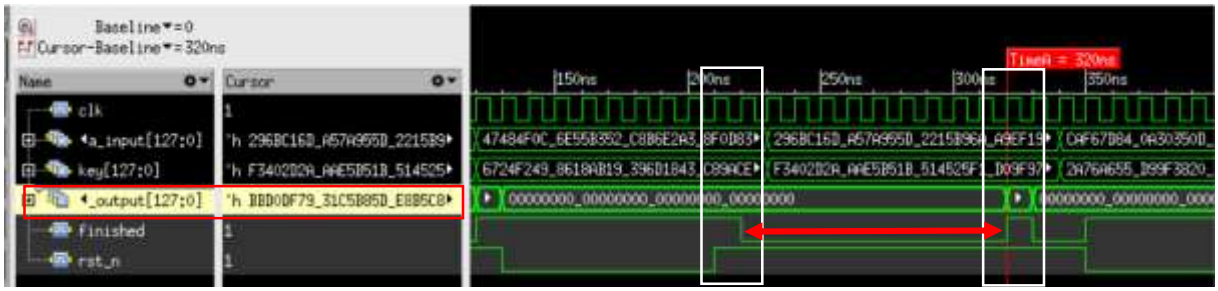
Tại thời điểm 130ns thì tín hiệu rst_n = 0, lúc này quá trình giải mã sẽ bị xóa về trạng thái ban đầu, dữ liệu đang giải mã hiện tại sẽ bị xóa. Tại thời điểm 210ns, tín hiệu rst_n = 1, sau đó tại thời điểm 220ns tín hiệu finished = 0, bộ giải mã sẽ tiếp tục, và lúc này, dữ liệu giải mã sẽ bắt tại thời điểm 220ns với:

data_in[127:0] = 128'h 47484f0c6e55b352c8b6e2a38f0d83b1 và

key[127:0] = 128'h6724f2498618ab19396d1843c89ace31

Tại thời điểm 320ns sẽ tạo ra kết quả được giải mã là:

data_out[127:0] = 128’hbbd0df7931c5b85de8b5c8a8b091e9ad.



Hình 5.13: Dữ liệu được giải mã sau 10 chu kì xung clk

Quá trình kiểm tra tương tự như chức năng mã hóa, chức năng giải mã của lõi IP hoạt động đúng khi có tín hiệu reset tác động, trường hợp kiểm tra này đạt yêu cầu, kiểm tra hoàn tất.

Xác minh đối với dữ liệu giải mã là một gói dữ liệu:

Trường hợp kiểm tra cho chức năng giả mã của lõi IP, thực hiện tương tự như chức năng giải mã, cần một gói dữ liệu mã hóa và một khóa đưa vào. Đối với chức năng này, sẽ dùng dữ liệu đầu ra của chức năng mã hóa đưa vào làm đầu vào cho chức năng giải mã. Còn khóa đầu vào của giải mã là khóa vòng cuối cùng của mã hóa.

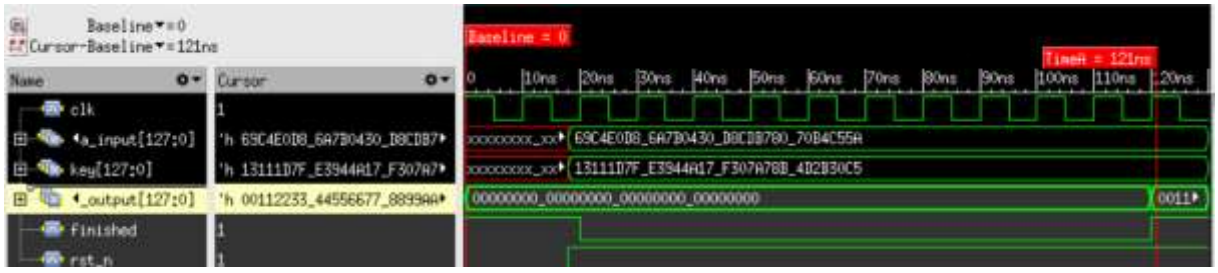
data_in[127:0] = 128'h 69c4e0d86a7b0430d8cdb78070b4c55a;

key[127:0] = 128'h13111d7fe3944a17f307a78b4d2b30c5;

Dùng công cụ của nhà cung cấp kiểm tra thì kết quả dữ liệu giải mã mong đợi là:

data_out[127:0] = 128’h00112233445566778899aabbccddeeff;

Thực hiện mô phỏng để lấy kết quả và xem dạng sóng của dữ liệu ra, thu được kết quả mô phỏng như hình bên dưới:



Hình 5.14: Kết quả trường hợp kiểm tra lõi giải mã một gói dữ liệu

Dữ liệu sau khi thực hiện giải mã thì thu được dữ liệu trùng khớp với dữ liệu đưa vào chức năng mã hóa ban đầu, điều này cho thấy thiết kế lõi IP chức năng giải mã đã hoạt động đúng đối với trường hợp một gói dữ liệu. Để đảm bảo bao phủ hết toàn bộ trường hợp kiểm thử, các phần tới sẽ kiểm tra như đã kiểm tra cho chức năng mã hóa.

Xác minh đối với dữ liệu giải mã là nhiều gói dữ liệu:

Tương tự như chức năng mã hóa, giải mã cũng sẽ cần dữ liệu giải mã và khóa sẽ là 10 gói. Với dữ liệu vào, khóa được tạo ngẫu nhiên liên tục và dữ liệu ra mong muốn thu được lần lượt là:

Bảng 5.2: Kiểm tra chức năng giải mã lõi IP với các gói dữ liệu liên tục

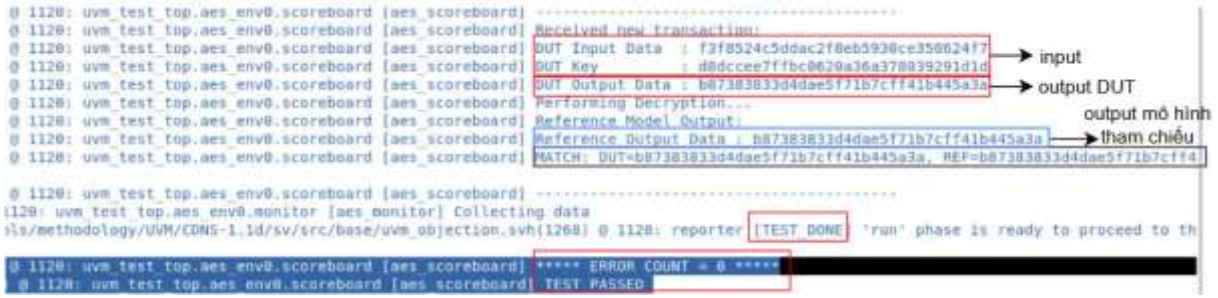
Gói 1	data_in[127:0]	128’h8d74bb054d3127ae41a849d30084df85
	key[127:0]	128’h73e4e195c0f7390dd3f5c359997972e0
	data_out[127:0]	128’hbbf3450f3e1cd3844f680b75bd376b6d
Gói 2	data_in[127:0]	128’h b030e3b156c54c13fcb14ae1ba1ea468
	key[127:0]	128’h54e4de2cdbd7ca0bafb24c7236e258b9
	data_out[127:0]	128’hb7de5dc14ec48f9a5af9fafbaa75da47
Gói 3	data_in[127:0]	128’h925456128dea2e1e561021a9d5003a00
	key[127:0]	128’he100190d00a4d40dc78a5a203ee7223b
	data_out[127:0]	128’h506a96c95cda7d9970862eff08508639
Gói 4	data_in[127:0]	128’h33df1229f29fcdcf4dc4ce2b5129a04d
	key[127:0]	128’h183692382f5e57121b7dec62b187cf68
	data_out[127:0]	128’h87533a2740d7e6efb05f5ef0041352c5
Gói 5	data_in[127:0]	128’h 86f640ea8675d90e3064cf932e9ad74f
	key[127:0]	128’h9a7fd7f27963cd9401e7db3621371e4f
	data_out[127:0]	128’h7f935c01db932f07d4f5be17ca44ffa2
Gói 6	data_in[127:0]	128’h8b99e256496c4fdbfdf025e36d53df06
	key[127:0]	128’h67dbea3bdeb537937ac8269c8df50ef1
	data_out[127:0]	128’h800adb19f3db49140765c89afe8fe3db
Gói 7	data_in[127:0]	128’h4fa4cd7839f3844d69d151ec0d54b772
	key[127:0]	128’he0533acd4df31a962fc3f696654ee33d
	data_out[127:0]	128’hceba3dff06024cce6e6610ce27428c80
Gói 8	data_in[127:0]	128’hd04094bbc1fabcf018efc70a0e9d6094
	key[127:0]	128’h1d6b79b3c909f3c5e0a34cfc35049428
	data_out[127:0]	128’h727748bb414aa580f10e08dcebb62d53
Gói 9	data_in[127:0]	128’h0268cea979226121b2f9910f712dda6b

	key[127:0]	128'ha59685285f6cc07123f8fff401c8e6ce
	data_out[127:0]	128'h138ef9a51b8aed65669f1c7b1bc18cf7
Gói 10	data_in[127:0]	128'hf3f8524c5ddac2f8eb5930ce350624f7
	key[127:0]	128'hd8dccee7ffbc0620a36a378039291d1d
	data_out[127:0]	128'hb87383833d4dae5f71b7cff41b445a3a

Thực hiện mô phỏng để kiểm tra kết quả, sử dụng scoreboard để so sánh dữ liệu của mô hình và dữ liệu của thiết kế. Sau khi kiểm tra thì tất cả đều đúng, dữ liệu giải mã của thiết kế trùng khớp với mô hình được xây dựng trước đó.

```
** Report counts by severity
UVM_INFO :  150
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[TEST_DONE]     1
[aes_driver]    10
[aes_monitor]    33
[aes_multi_de]     1
[aes_scoreboard]   102
[aes_test_continuous_dec]     2
Simulation complete via $finish(1) at time 1120 NS + 61
/mnt/CADENCE/xcelium_2209/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
xcelium> database -close waves
xcelium> exit
TOOL:   xrun    22.09-s004: Exiting on Apr 17, 2025 at 15:29:33 +07  (total: 00:00:04)
```

Hình 5.15: Kết quả trường hợp kiểm tra nhiều gói dữ liệu liên tục



Hình 5.16: Tất cả dữ liệu đều được giải mã đúng

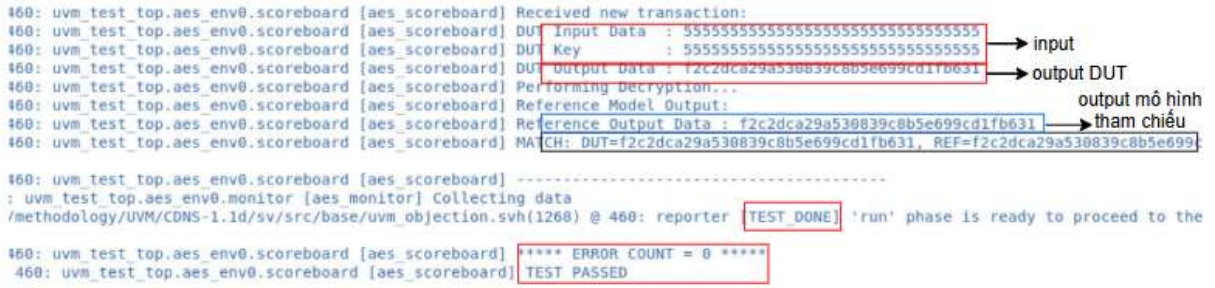
Hình 5.17: Kiểm tra dạng sóng của nhiều gói dữ liệu

Khi truyền liên tục 10 gói dữ liệu, cứ sau 10 chu kì xung clk, thì tín hiệu finished = 1, sẽ xuất dữ liệu đầu ra, tiếp theo sẽ thực hiện giải mã gói dữ liệu tiếp theo liên tục cho tới khi hoàn thành. Kết quả cho thấy lõi IP hoạt động chức năng giải mã liên tục đã chính xác, dữ liệu mã hóa giống như dữ liệu mong đợi.

Xác minh đối với dữ liệu giải mã là các gói kí tự đặc biệt:

Cũng tương tự chức năng giải mã, các trường hợp đặc biệt cũng áp dụng cho chức năng giải mã. Dữ liệu giải mã và khóa của trường hợp kiểm tra này cũng như của chức năng mã hóa.

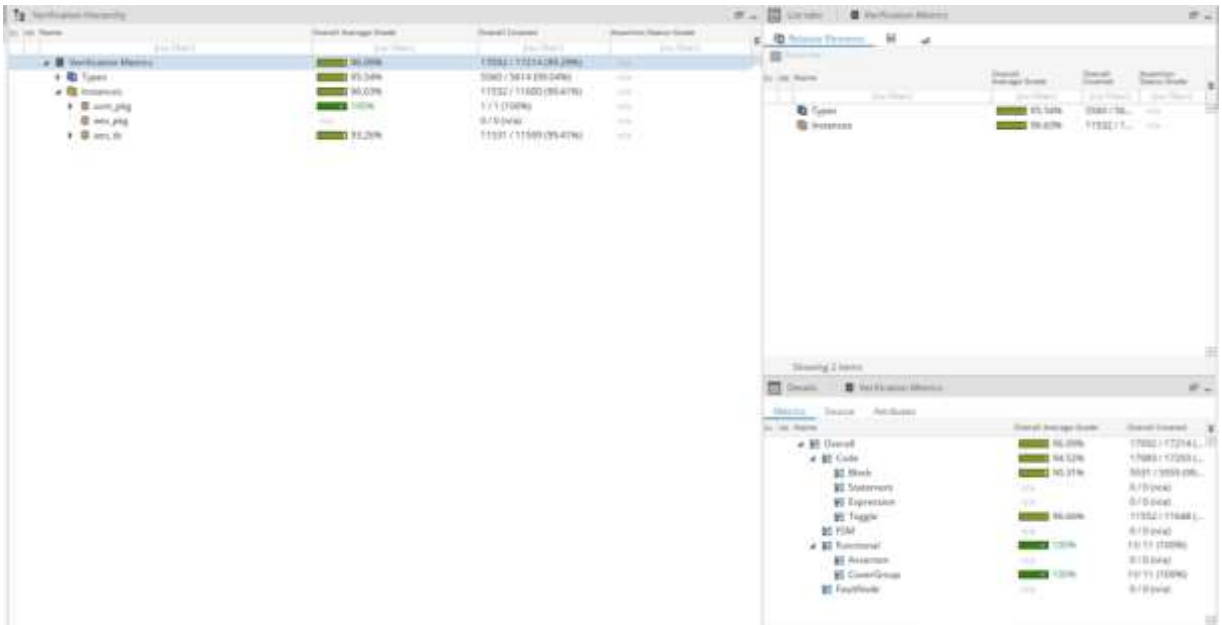
Thực hiện kiểm tra xác minh hoàn tất, tất cả đều PASSED, thì thu được các kết quả chính xác, dữ liệu ra của thiết kế trùng khớp với của mô hình.



Hình 5.18: Kết quả kiểm tra các trường hợp đặc biệt

5.3.4. Kết quả đánh giá độ bao phủ

Độ bao phủ giúp đánh giá mức độ mà quá trình mô phỏng đã kiểm tra được các phần khác nhau của thiết kế hay chưa. Trong Hình 5.19 là kết quả bao phủ của quá trình kiểm thử với độ bao phủ mã đạt 96% và độ bao phủ chức năng đạt 100%. Với mức bao phủ cao và xem lại các phần chưa được bao phủ không ảnh hưởng đến chức năng của lõi IP xác định rằng tất cả các trường hợp hầu như đã được kiểm tra.



Hình 5.19: Kết quả độ bao phủ

5.4. Kết luận và đánh giá chung

Sau quá trình thiết kế và mô phỏng, đã đưa ra một số nhận xét chung:

Quá trình thiết kế và mô tả được thực hiện ở quy trình Front-end. Quá trình làm

việc đều phải báo cáo tiến độ, lập kế hoạch cụ thể về thời gian bắt đầu vào hoàn thành từng quy trình của đồ án cho kỹ sư cấp cao trong công ty.

Mô phỏng kiểm tra mô tả và mô phỏng xác minh chức năng của thiết kế nhờ vào tool của công ty FPT Semiconductor.

Toàn bộ thiết kế được kiểm tra chính xác, mỗi lần hoạt động lõi AES chỉ thực hiện một chức năng mã hóa và giải mã. Điều này rất tiện cho khách hàng mong muốn thiết kế một lõi IP thực hiện một chức năng duy nhất, giúp tiết kiệm năng lượng, tăng tuổi thọ của lõi IP.

Khi thực hiện xác minh chức năng, toàn bộ các trường hợp được kiểm tra kỹ, quan sát dạng sóng, tất cả được giám sát bởi kỹ sư trong công ty FPT.

KẾT LUẬN

Cùng sự hỗ trợ của công ty FPT và giáo viên hướng dẫn, đồ án đã hoàn thành việc thiết kế và xác minh một lõi IP mềm, thực hiện chuẩn mã hóa AES-128 với chuẩn ASIC. Kết quả mô phỏng và kiểm tra chức năng cho thấy lõi IP hoạt động chính xác theo yêu cầu mô tả, đảm bảo tính đúng đắn và hiệu quả.

Đóng góp chính của đồ án là thiết kế thành công frontend của một lõi IP để có thể tích hợp bảo vệ các hệ thống nhúng và hệ thống SoC. Đồng thời, đồ án cũng ứng dụng phương pháp thiết kế và xác minh chức năng theo trình tự của một kỹ sư vi mạch thực tế, điều này là lợi thế cho sinh viên sau này khi đi làm.

Trong tương lai, hướng phát triển của đồ án bao gồm: tối ưu về thời gian mã hóa và tài nguyên phần cứng, cũng như tiếp tục hoàn thành các quy trình tiếp theo như tổng hợp, thiết kế vật lý để hoàn thành một lõi IP có thể đưa vào sản xuất và ứng dụng vào thực tế.

TÀI LIỆU THAM KHẢO

[1] Dworkin, Morris J., et al. "Advanced encryption standard (AES)." (2001).

[2] Marsaw, Nicholas J. "UVM Verification of a Floating Point Multiplier." (2019).

[3] Graphics, Mentor. "Universal Verification Methodology UVM Cookbook." *Online Methodology Documentation from the Verification Methodology Team* (2019).

[4] Palnitkar, Samir. *Verilog HDL: a guide to digital design and synthesis*. Vol. 1. Prentice Hall Professional, 2003.

[5] Spear, Chris. *SystemVerilog for verification: a guide to learning the testbench language features*. Springer Science & Business Media, 2008.

[6] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA, USA: Addison-Wesley, 2010.

[7] Zhang, Xinmiao, and Keshab K. Parhi. "High-speed VLSI architectures for the AES algorithm." *IEEE transactions on very large scale integration (VLSI) systems* 12.9 (2004): 957-967.

[8] Chen, Liang-Bi, et al. "The AES Design Space Exploration with a Soft IP Generator." *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2007)*. Vol. 2. IEEE, 2007.

[9] Saravanan, P., et al. "A high-throughput ASIC implementation of configurable advanced encryption standard (AES) processor." *IJCA Special Issue on Network Security and Cryptography, NSC* 3 (2011): 1-6.

[10] Kim, Ho Keun, and Myung Hoon Sunwoo. "Low power AES using 8-bit and 32-bit datapath optimization for small Internet-of-Things (IoT)." *Journal of Signal processing systems* 91 (2019): 1283-1289.

[11] Lin, Ming-Bo, and Jen-Hua Chuang. "The design of a high-throughput hardware architecture for the AES-GCM algorithm." IEEE Transactions on Consumer Electronics 70.1 (2023): 425-432.

[12] Zhu, Lin, et al. "A UVM-based AES IP Verification Platform with Automatic Testcases." *2016 International Conference on Engineering and Advanced Technology (ICEAT 2016)*. Atlantis Press, 2016.

[13] H. Foster, "2014 Wilson Research Group Functional Verification Study," Mentor Graphics, 2014.

PHỤ LỤC 1

Link github chứa mã toàn bộ thiết kế và môi trường xác minh để chạy mô phỏng trên phần mềm của Cadence Xcelium.
https://github.com/NguyenVinh1609/DATN_AES_IP.git