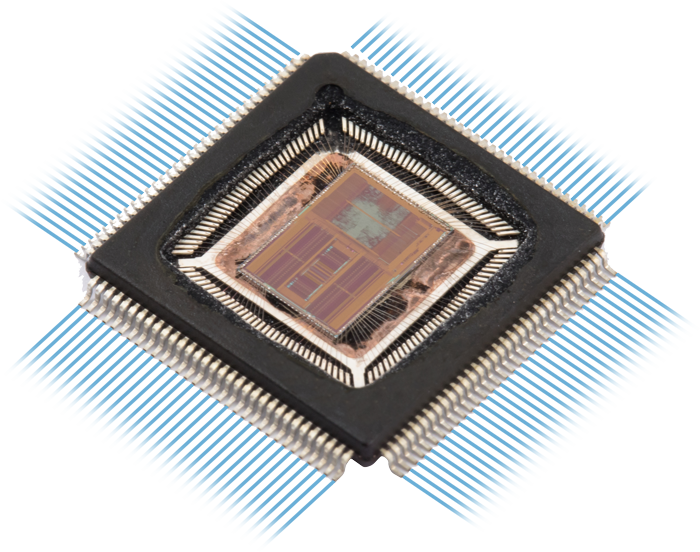
# **Chương 1: Bán dẫn và thiết kế vi mạch**

* 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). Tiếp theo, chúng ta sẽ có 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.



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.

Bên cạnh đó, VLSI còn mang lại hiệu quả cao về mặt chi phí. Nhờ việc tích hợp nhiều linh kiện lên một chip duy nhất, số lượng linh kiện rời rạc trong mạch được giảm đáng kể, từ đó hạ thấp chi phí sản xuất. Ngoài ra, công nghệ này còn giúp cải thiện hiệu năng tổng thể, đồng thời tối ưu hóa kích thước và hiệu suất của hệ thống, góp phần quan trọng vào sự phát triển của ngành công nghiệp bán dẫn.

* 1. quy trình thiết kế ASIC
     1. Giới thiệu chung

Quy trình cơ bản để thiết kế một vi mạch số thường bao gồm các giai đoạn chính như Định nghĩa yêu cầu thiết kế (Design Specification), Mô tả bằng RTL (RTL description), Kiểm tra chức năng (Functional Verification), Thiết kế logic (Logic Design), Thiết kế vật lý (Physical Design), Mô phỏng sau khi hoàn tất bố cục (Post Layout Simulation) và cuối cùng là Sản xuất (Production).

A diagram of a process

AI-generated content may be incorrect.

* + 1. 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 Cấp Thấp (Low Block Level Design).

A diagram of a diagram

AI-generated content may be incorrect.

* + 1. RTL Description

Mô tả RTL (RTL Description), hay còn gọi là viết mã RTL, 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, v.v.

A diagram of a data flow

Description automatically generated

Sau khi hoàn thành mã RTL, người 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.

A screenshot of a computer

Description automatically generated

* + 1. Functional verification

Sau khi hoàn thành việc viết mã, thiết kế sẽ trải qua giai đoạn xác minh (Verification) 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 (Functional Verification) chỉ hoàn tất khi tất cả lỗi đã được khắc phục và thiết kế đáp ứng đầy đủ yêu cầu đề ra.

* + 1. Logic Design

Thiết kế logic (Logic Design) là giai đoạn tổng hợp (synthesis), 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.

A diagram of a diagram

Description automatically generated

* + 1. Physical Design

Thiết kế vật lý (Physical Design) 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, được gửi đến nhà máy để sản xuất vi mạch thực tế.

A black text and a white background

Description automatically generated

* + 1. Post Layout Simulation

Mô phỏng sau bố cục (Post Layout Simulation) là 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 (functional simulation), mô phỏng thời gian (timing simulation), phân tích công suất (power analysis) và phân tích tính toàn vẹn tín hiệu (signal integrity analysis).

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. 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.

A computer and a diagram of a color spectrum

Description automatically generated with medium confidence

* 1. **Quy trình thiết kế ASIC Front-End.**
     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 (RTL Description) 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.

A diagram of a computer program

Description automatically generated

Các bước phân tích thiết kế vi mạch số

* Nghiện cứu thị trường:

Từ những yêu cầu thiết kế của khách hàng, người 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 người thiết kế nắm rõ các vấn đề liên quan đến thiết kế để đưa ra các đánh giá, so sánh và đưa ra các hương 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, người thiết kế phải xác định được các yếu tố sau đây:

Mô tả chức năng: xác định được các tính năng mà thiết kế sẽ hỗ trợ từ những yêu cầu thiết kế.

Giao tiếp: 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 hiều hai chiều.

Mô tả các tín hiệu: người 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: tại bước này, người thiết kế 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: 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. Ví dụng trong một chip vi điều khiển, bước này được áp dụng cho toàn bộ chip vi điều khiển, vì nó chưa nhiều khối chức năng. Như trong STM32L4 + LoRa thì dữ liệu cảm biến gửi qua mạng LoRa được mã hóa AES-128 để tránh bị nghe lén. 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 trọng lõi.

A diagram of a data processing process

AI-generated content may be incorrect.

Thiết kế khối mức cao cho lõi IP mềm Mã hóa AES 128

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)

* Thiết kế khối mức thấp.

Trong giai đoạn này, người 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.

A diagram of a computer

AI-generated content may be incorrect.

Ví dụ sơ đồ mạch nguyên lý mức cổng logic

* Mô tả RTL ( RTL Description)

Giai đoạn này yêu cầu sử dụng ngôn ngữ mô tả phần cứng (HDL) 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.

Một tệp RTL code chứa một cặp từ khóa “**module/endmodule**” điển hình bao gồm:

+ Phần ghi chú đầu: Cung cấp thông tin như tên công ty, dự án, chức năng của mã RTL và tác giả. Các ghi chú này được viết sau dấu “//”hoặc giữa “/\* \*/”.

+ Khai báo trước khi tổng hợp: Bao gồm `define, `timescale, hoặc `include để đưa vào các tệp chứa định nghĩa và thông tin thời gian.

+ Định nghĩa module: Sử dụng cặp từ khóa `module và `endmodule để mô tả từng khối.

+ Khai báo hằng số: Định nghĩa bằng `parameter, `localparam, hoặc qua các tệp chứa giá trị cố định dùng trong thiết kế.

+ Tín hiệu giao tiếp: Bao gồm đầu vào, đầu ra hoặc tín hiệu hai chiều kết nối với các thành phần bên ngoài.

+ Biến và tín hiệu nội bộ: Các biến chỉ sử dụng trong module, có thể là dây kết nối, thanh ghi hoặc biến lặp.

+ Mô tả chức năng module: Viết bằng HDL để hiện thực hóa thiết kế đã phân tích ở bước trước.

* 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, người thiết kế 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. ***Xác minh chức năng (Functional Verification)***

Xác minh chức năng (Functional Verification) 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 (Specification). 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 First Silicon Success (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.

A screenshot of a black background

AI-generated content may be incorrect.

Quá trình kiểm tra chức năng (Functional Verification) là một phần thiết yếu trong phát triển thiết kế phần cứng số, nhằm đảm bảo rằng mạch số (DUT – Design Under Test) hoạt động đúng theo yêu cầu kỹ thuật đã đề ra. 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.
* Assertions: Xác định và sử dụng các điều kiện logic để kiểm tra tín hiệu và hành vi của thiết kế trong suốt quá trình xác minh.

*Xác định khả năng sử dụng VIP (Verification IP):*Ở bước này, đội ngũ 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.
* Hồi quy (Regression Testing): 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.

Nhìn chung, 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. Mục tiêu chính là đảm bảo rằng thiết kế hoạt động chính xác theo đặc tả, đạt độ tin cậy cao, giảm thiểu rủi ro và tăng khả năng First Silicon Success.

* 1. **Kết luận**

Kết thúc chương này, chúng ta 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. Chúng ta đã 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 chi tiết, 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.

## **2.2 Nguyên lý hoạt động của AES-128**

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.

Chúng ta sẽ thiết kế lõi IP mềm thực hiện 2 chức năng mã hóa và giải mã, ban đầu lõi IP sẽ mặc định là thực hiện chức năng mã hóa, thì chức năng giải mã sẽ bị vô hiệu hóa. Khi chúng ta 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 ảnh về nguyên lý thiết kế được trình bày bên dưới.

A computer screen shot of a diagram

AI-generated content may be incorrect.

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

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 |

### ***2.2.1 Quá trình 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.

+ SubBytes: Áp dụng một bảng thay thế (S-Box) để thay thế từng byte dữ liệu nhằm tăng tính phi tuyến tính.

+ ShiftRows: Dịch vòng các hàng của ma trận dữ liệu để tạo sự phân tán.

+ MixColumns: Kết hợp các byte trong từng cột bằng phép nhân ma trận trong trường hữu hạn GF(2^8).

+ AddRoundKey: 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].

A diagram of a computer

AI-generated content may be incorrect.

### ***2.2.2 Quá trình giải mã***

Mã hóa chuyển một "bản rõ" (plaintext) thành một "bản mã" (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 bản rõ từ một bản mã.

Trong quá trình giải mã, ma trận ciphertext sẽ bị biến đổi bởi các chức năng AddRoundKey, InvSubBytes, InvShiftRows hoặc InvMixColumns 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 cơ bản là AddRoundKey, InvSubBytes, InvShiftRows, InvMixColumns và InvKeyExpansion. Chú ý, InvKeyExpansion không phải là một chức năng được mô tả trong chuẩn mà chỉ 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.

* Bước 1. Bước khởi tạo: Dữ liệu cần được mã hóa 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. Chú ý, InvKeyExpansion 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 khôi phục lại plain\_text[127:0]

A diagram of a flowchart

AI-generated content may be incorrect.

* InvSubBytes: Thay thế byte bằng bảng S-Box nghịch đảo.
* InvShiftRows: Dịch vòng ngược các hàng
* InvMixColumns: Áp dụng phép biến đổi nghịch đảo trên các cột.
* AddRoundKey: XOR với khóa con tương ứng.

## **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:

* Bảo mật dữ liệu trong mạng: Sử dụng trong giao thức TLS/SSL 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.
* 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:

* Xác định các thông số kỹ thuật của lõi AES-128.
* Phân tích kiến trúc phần cứng, thiết kế khối mức cao, thiết kế khối mức thấp
* Thiết kế giao diện kết nối giữa các khối bên trong lõi
* Cấu trúc của các mô-đun chính, bao gồm bộ sinh khóa con cho chức năng mã hóa (Key Expansion), bộ sinh 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 kĩ 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. 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, ta sẽ trình bày rõ ở các phần tiếp theo.

A screenshot of a computer

Description automatically generated

Hình, Khối mức cao của bộ mã hóa

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

Bộ mã hóa AES-128, aes128\_cipher\_top, được thiết kế gồm 2 khối:

* Khối mã hóa dữ liệu AES cipher core
* Khối tính toán tạo khóa vòng AES Key Expand

Bộ mạ hóa gồm các tín hiệu giao tiếp sau:

* 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, bản rõ (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, bản mã (ciphertext).

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

* 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

A screenshot of a computer

AI-generated content may be incorrect.

Hình, Lõi bên trong của mã hóa AES

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

Sau khi thiết kế ở khối mức cao, tiếp theo chúng ta 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, để các kỹ sư mô tả RTL có thể đọc vào và hiểu được, sau đó sẽ dùng ngôn ngữ lập trình mô tả để mô tả lại đặc tả thiết kế của chúng ta.

Khối AES cipher core sẽ được thiết kế chi tiết hơn, với các yêu cầu của thuật toán mã hóa, chúng ta đã thiết kế được phần bên trong của nó như hình bên dưới:

A diagram of a computer

AI-generated content may be incorrect.

Để hiểu thêm thì chúng ta đi vào mô tả chi tiết nguyên lý hoạt động của AES cipher core.

* + - 1. Tổng quát về thành phần trong thiết kế mức thấp

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 ta 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 Add round key. 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.

A diagram of a computer

AI-generated content may be incorrect.

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

Dữ liệu sau khi Add round key thì sẽ được đưa vào SubBytes, sau mỗi vòng biến đổi tương ứng với 1 chu kì clk, vì thế chúng ta sẽ sử dụng Flip flop để chốt dữ liệu cho tới chu kì tiếp theo mới thay đổi, nhằm bộ mã hóa hoạt động đúng, bắt dữ liệu chính xác đúng thời điểm. Bên cạnh đó, có thêm tín hiệu rkey\_en, tín hiệu này cho phép bộ mã hóa hoạt động.

A diagram of a computer

AI-generated content may be incorrect.

hình, 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.

A diagram of a computer

AI-generated content may be incorrect.

* 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.

A diagram of a computer

AI-generated content may be incorrect.

* Để có được các tín hiệu điểu khiển các bộ mux, thì chúng ta cần có một khối AES cipher controller, khối này sẽ hoạt động dựa vào chu kì clk, cứ sau mỗi chu kì clk, thì các giá trị đầu ra sẽ thay đổi như sơ đồ bên dưới:

A diagram of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

* + - 1. Chi tiết trong từng khối biến đổi

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), để bạn đọc cảm thấy dễ hiểu hơn về thuật toán, tôi sẽ chia 16 byte này thành 1 ma trận gồm 4 hàng 4 cột cho cả plain\_ text và cipher\_key.

Sau đây ta đi vào chi tiết của từng bước biến đổi.

* Chức năng AddRoundKey:

Hay xem hình bên dưới, minh họa cách hoạt động của chức năng này:

A diagram of a rectangular object with a circle and a plus and a cross

AI-generated content may be incorrect.

Dưới đây là thiết kế của chức năng:

A black background with a white circle and a green sign

Description automatically generated

Chức năng này thực hiện:

* Bước khởi tạo, nó sẽ thực hiện XOR khóa mã với ma trận dữ liệu
* Bước lặp mã hóa và bước tạo ngõ ra: XOR khóa vòng với ma trận trạng thái.

Ở chức năng này, nó sẽ XOR lần lượt từng byte tương ứng cùng vị trí của hai ma trận khóa mã và dữ liệu với nhau.

* 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.

A diagram of a graph

AI-generated content may be incorrect.

Dưới đây là thiết kế của chức năng:

A screenshot of a computer

Description automatically generated

* 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 tư quay ba byte.

A diagram of a graph

AI-generated content may be incorrect.

Dưới đây là thiết kế của chức năng:

A screenshot of a computer

Description automatically generated

* 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

A diagram of a number system

AI-generated content may be incorrect.

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 = S00.02 + S10.03 + S20 + S30 = S00.02 + (S10.02 + S10.01) + S20 + S30

A number grid with numbers

AI-generated content may be incorrect.A group of math equations

AI-generated content may be incorrect.

Phép nhân với 01 thì giữ nguyên giá trị. Phép nhân với 02 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 tri được dịch bằng 1 thì giá trị sau khi dịch được XOR với H1b
* 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 từ nào với 02, kết quả sau đó XOR với chính nó thì sẽ ra kết quả của phần tử đó nhân với 03:

A screenshot of a computer

Description automatically generated

Hình 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 tiền trong ma trận trạng thái:

A diagram of a computer code

AI-generated content may be incorrect.

- Tương tự như cách triển khai của cột 1, các cột còn lại cũng sẽ được thiết kế tương tự.

### 3.3.2.3. 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 dựng 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 là RotWord, SubWord, AddRcon và AddW.

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

A diagram of a computer

AI-generated content may be incorrect.

Hình 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].

Các chức năng RotWord, SubWord và AddRcon được mô tả và thiết kế bên dưới:

* Chứ năng RotWord:

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

A screenshot of a computer screen

Description automatically generated

Hình Sơ đồ logic triển khai chức năng RutWord

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

A yellow rectangular object with numbers and arrows

Description automatically generated

Hình Sơ đồ logic triển khai chức năng AddRcon

* Chức năng AddW:

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

A screenshot of a computer screen

Description automatically generated

Hình Sơ đồ logic triển khai chức năng AddW.

* + 1. ***Mô tả RTL***

Từ thiết kế logic ở mưc thấp, chúng ta sẽ sử dụng ngôn ngữ mô tả phần cứng (HDL) là Verilog để mô tả thiết kế. Ví dụ mô tả RTL code chức năng shift\_rows:

**A computer screen shot of a diagram

AI-generated content may be incorrect.**

Hình Mô tả RTL code chức năng shift\_rows

Các chức năng trong quá trình mã hóa đều được mô tả RTL code. Kết thúc bước mô tả các cổng logic hay đoạn văn bằng RTL code, ta đến bước tiếp theo để kiểm tra lỗi và luật thiết kế. Sau bước này là kết thúc quá trình mô tả RTL code, chuyển qua xác minh.

## **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.

A screenshot of a computer screen

AI-generated content may be incorrect.

### ***3.4.1 Thiết kế khối mức cao (High Level Block Design)***

Bộ giải mã AES-128, aes128\_decipher\_top, được thiết kế gồm 2 khối:

* Khối mã hóa dữ liệu AES decipher core
* Khối tính toán tạo khóa vòng AES Inv Key Expand

Bộ giải mã gồm các tín hiệu giao tiếp sau:

* clk\_sys: 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, bản mã
* decipher\_en: tín hiệu cho phép bộ giải mã hoạt động, tích cực 1 chu kỳ xung clock clk\_sys để yêu cầu bộ giải mã hoạt động
* 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, bản rõ.

Các tín hiệu giao tiếp giữa khối AES decipher core và AES Inv Key Expand:

* 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 giải mã, begin\_round=1 khi round\_num[3:0]=0

A screenshot of a computer screen

AI-generated content may be incorrect.

### ***3.4.2 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, chúng ta 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:

A diagram of a computer

AI-generated content may be incorrect.

Hình Sơ đồ logic bên trong AES decipher core

Để hiểu thêm thì tôi sẽ đi sâu vào mô tả chi tiết nguyên lý hoạt động của từng chức năng trong lõi.

3.4.2.1. Tổng quát về thành phần trong thiết kế mức thấp lõi giải mã

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, dữ liệu đưa vào khối InvShiftRows rồi tiếp theo qua khối InvSubBytes sau đó quay về AddRoundKey.

Tại bước AddRoundKey này, khóa đầu vào cũng sẽ lấy tương tự như ở bộ mã hóa.

Sau khi thực hiện AddRoundKey thì sẽ qua bước InvMixColumns, đầu ra của bước này thì sẽ kết thúc 1 vòng biến đổi. Tại sườn lên của xung clk thứ 2 thì sẽ qua vòng tiếp theo, lúc này tín hiệu first\_time\_en = 0, thì dữ liệu đưa vào InvShiftRows sẽ được lấy từ đầu ra của InvMixColumns.

A diagram of a computer

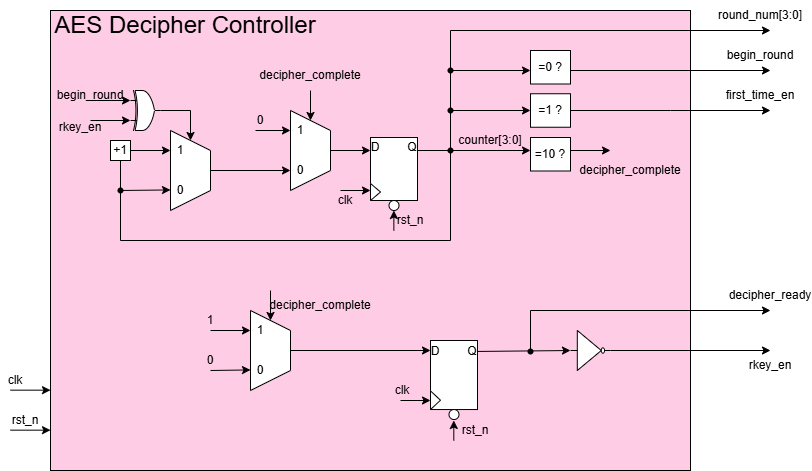
AI-generated content may be incorrect.

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 là tạo ngõ ra dữ liệu đầu ra của AddRoundKey sẽ có tại chân D của FF, khi cạnh lên clk của chu kì tiếp theo thì decipher\_ready = 1, sẽ bắt giá trị tại chân Q của FF gán cho plain\_text[127:0] và đây là kết quả giải mã.

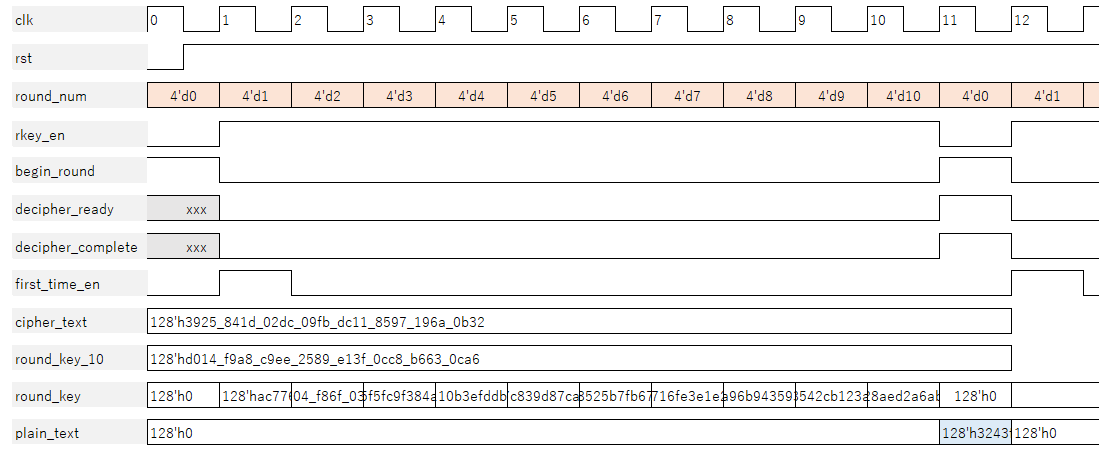
A diagram of a computer

AI-generated content may be incorrect.

* 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:



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 gia đoạn biến đổi chính từ khi đưa dữ liệu vào cho tới khi dữ liệu được mã hóa:



3.4.2.2. Chức năng của từng khối

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.

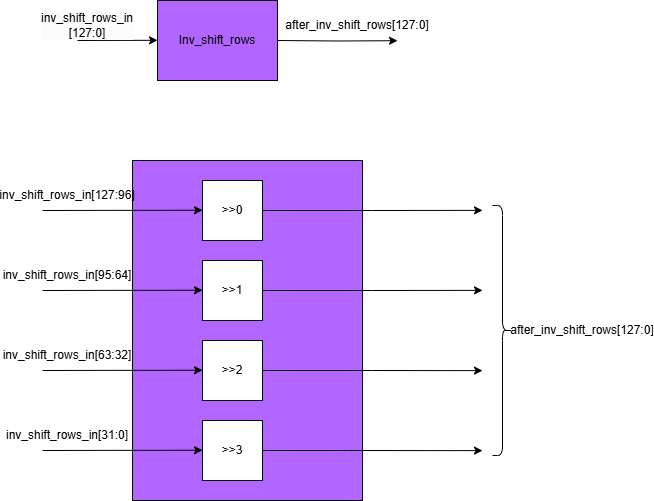
* Chức năng InvShiftRows:

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.

A diagram of a graph

AI-generated content may be incorrect.

Dưới đây là thiết kế của chức năng:



* 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).

A diagram of a diagram

AI-generated content may be incorrect.

Dưới đây là thiết kế của chức năng:

A screenshot of a computer

AI-generated content may be incorrect.

* 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.

A table with numbers and symbols

AI-generated content may be incorrect.A table with numbers and symbols

AI-generated content may be incorrect.

Nguyên tắc tính toán InvMixColumns là biến đổi các phép nhân với một số thành phép nhân với H01 và H02, thì cuối cùng nguyên tắc của nó sẽ tương tự như MixColumns.

- Phép nhân một byte A với H0e = B00001110 sẽ tương tự như sau:

A.H0e = A.H08 + A.H04 + A.H02 = A.H02.H02.H02 + A.H02.H02 + A.H02

- Phép nhân một byte A với H0b = B00001011 sẽ tương tự như sau:

A.H0b = A.H08 + A.H02 + A.H01 = A.H02.H02.H02 + A.H02 + A.H01

- Phép nhân một byte A với H0d = B00001101 sẽ tương tự như sau:

A.H0d = A.H08 + A.H04 + A.H01 = A.H02.H02.H02 + A.H02.H02 + A.H01

- Phép nhân một byte A với H09 = B00001001 sẽ tương tự như sau:

A.H0b = A.H08 + A.H01 = A.H02.H02.H02 + A.H01

Triển khai thành các sơ đồ logic:

A screenshot of a computer

Description automatically generated

Hình Tách phép nhân với một số thành các phép nhân cơ bản

Sau đó sẽ XOR các phần từ lại để tạo ra các phép nhân với H0e, H0d, H0b, và H09:

A screenshot of a computer

Description automatically generated

Tiếp tục XOR để tạo ra 4 byte ở cột 1:

A screenshot of a computer

Description automatically generated

Các cột còn lại thực hiện nguyên tắc thiết kế tương tự.

### ***3.4.3 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 mòng muốn không. Sau đây là đoạn code Verilog mô tả thiết kế của chức năng InvSubBytes.

Từ hình AA mô tả thiết kế của chức năng, triển khai qua code như bên hình dưới:

A screenshot of a computer

AI-generated content may be incorrect.

# **Chương 4. Xác minh lõi IP mềm chuẩn AES 128**

## **4.1 Giới thiệu chương**

Trong phần này tôi sẽ thảo luận chi tiết về xác minh lõi IP mền chuẩn AES 128 theo quy trình xác minh đã được trình bầy ở chương # bao gồm nghiên cứu đặc tả, lập kế hoạch xác minh, xây dựng môi trường và mô phỏng xác minh.

## **4.2. Nghiên cứu đặc tả (chức năng, ce, dec, chi tiết nhỏ)**

Quy trình xác minh và kiểm thử lõi IP mềm AES 128 sẽ được thực hiện dựa trên đặc tả kỹ thuật.

## **4.3. Lập kế hoạch xác minh**

Kế hoạch xác minh cho AES IP 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 AES IP 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 AES IP 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 AES IP 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.

A screenshot of a computer

AI-generated content may be incorrect.

Hình xx: kế hoạch xác minh cho AES IP

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, AES IP 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, AES IP và môi trường khiểm 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.

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 đich để kiểm tra xem AES IP có thực hiện đúng chức năng mã hóa hoặc giải mã khi được cấu hình hay không. AES IP 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.

Để đánh giá khả năng xử lý liên tục của AES IP, 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 AES IP 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 AES IP 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.

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 AES IP. 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 AES IP có thể xử lý mọi loại dữ liệu, kể cả các trường hợp đặc biệt mà thuật toán có thể gặp khó khăn.

## **4.4. Xây dựng môi trường xác minh**

việc xây dựng môi trường UVM trong DV 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 X là môi trường xác minh được xây dựng cho AES IP. 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.

A diagram of a software development process

AI-generated content may be incorrect.

Hình x: Môi trường xác minh cho AES IP

4.4.1. Interface

Trong quá trình xây dựng môi trường kiểm thử cho lõi mã hóa AES, một thành phần quan trọng không thể thiếu là giao tiếp (interface) giữa các khối chức năng trong testbench (như driver, monitor) và thiết kế cần kiểm thử (DUT – Device Under Test). Interface không chỉ đóng vai trò là cầu nối vật lý giữa testbench và DUT mà còn là công cụ giúp cô lập thiết kế khỏi môi trường kiểm thử, từ đó nâng cao khả năng tái sử dụng và mở rộng hệ thống.

Giao diện AES gồm các tín hiệu xung clock, reset, 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.

4.4.2. 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.

4.4.2. Sequence

Trong môi trường xác minh theo chuẩn UVM (Universal Verification Methodology), 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).

4.4.3. 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.

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình X

Hình x mô tả 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 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.

4.4.4. 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 AES IP 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).

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.

Quy trình được thực hiện như sau:

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 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.

A screenshot of a computer

AI-generated content may be incorrect.

Task check\_finish\_signal có 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 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.

A screenshot of a computer

AI-generated content may be incorrect.

4.4.5. 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.

A black and white diagram

AI-generated content may be incorrect.

Quy trình xử lý dữ liệu trong scoreboard được trình bày chi tiết qua sơ đồ luồng như sau:

Bắt đầu:

Scoreboard chờ nhận một transaction từ monitor thông qua analysis port. Transaction này bao gồm data\_input, key, data\_output.

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.

Lựa chọn mô hình tham chiếu:

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).

So sánh kết quả:

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.

## **4.5 Mô phỏng xác minh**

## **4.6 Kết luận**