MỤC LỤC

MỤC LỤC **1**
BẢNG VIẾT TẮT **3**
DANH SÁCH BẢNG **4**
DANH SÁCH HÌNH **5**
TÓM TẮT LUẬN VĂN **6**
MỞ ĐẦU **7**
CHƯƠNG 1 MỘT SỐ PHƯƠNG PHÁP TẤN CÔNG VÀ BẢO VỆ THIẾT KẾ FPGA **9**
1.1 Một số phương pháp tấn công **9**
1.1.1 Nhân bản, dán nhãn sai và sản xuất vượt số lượng 9
1.1.2 Kỹ thuật đảo ngược 10
1.1.3 Kỹ thuật tấn công đọc lại 11
1.2 Một số phương pháp bảo vệ **12**
1.2.1 Mã hóa bitstream 13
1.2.2 Xác thực bitstream 16
1.2.3 Sử dụng IC xác thực 17
1.3 Kết luận chương **18**
CHƯƠNG 2 GIẢI PHÁP BẢO VỆ THIẾT KẾ FPGA BẰNG IC XÁC THỰC **19**
2.1 Lựa chọn thành phần **19**
2.2 Thiết kế giải pháp **20**
2.3 Thiết kế cần bảo vệ Bộ đếm 8-bit **22**
2.4 Thiết kế lõi xác thực **22**
2.4.1 Bộ tạo số ngẫu nhiên 22
2.4.2 Thuật toán SHA1 23
2.4.3 Cấu trúc bản tin sử dụng để xác thực. 24
2.4.4 Giao thức 1-wire 25
2.4.5 Lưu đồ thuật toán chương trình chính. 26
2.5 Kết quả **26**
2.6 Kết luận **27**
CHƯƠNG 3 GIẢI PHÁP MÃ HÓA THIẾT KẾ FPGA **28**
3.1 Giải pháp thực hiện **28**
3.2 Thuật toán GOST 28147-89 ..... **28**
3.2.1 Ký hiệu 28
3.2.2 Phép biến đổi Sbox 30

3.2.3 Các biến đổi 30
3.2.4 Thuật toán lược đồ khóa 31
3.2.5 Thuật toán mã hóa cơ bản 31
3.3 Xây dựng phần mềm mã hóa.....32
3.4 Thiết kế bộ giải mã 33
3.5 Thiết kế lõi mật mã GOST 28147-89 34
3.6 Mô phỏng và thử nghiệm.....35
3.7 Kết luận 37
KẾT LUẬN 38
TÀI LIỆU THAM KHẢO.....39

BẢNG VIẾT TẮT

Ký hiệu	Ý nghĩa Tiếng Anh	Ý nghĩa Tiếng Việt
AE	Authenticated Encryption	Mã hóa chứng thực
AES	Advanced Encryption Standard	Chuẩn mã hóa tiên tiến
ASIC	Application Specific Integrated Circuit	Mạch tích hợp chuyên dụng
AXI	Advanced eXtensible Interface	Giao tiếp mở rộng tiên tiến
BRAM	Block Random Access Memory	Khối bộ nhớ truy cập ngẫu nhiên
CMAC	Cipher-based MAC	Mã xác thực dựa trên mã hóa
CPLD	Complex Programmable Logic Device	Linh kiện logic lập trình
CRC	Cyclic Redundancy Check	mã kiểm tra dư thừa tuần hoàn
EEPROM	Electrically Erasable Programmable Read-Only Memory	Bộ nhớ chỉ đọc có thể xóa bằng điện
FIPS	Federal Information Processing Standards	Tiêu chuẩn xử lý thông tin liên bang
FPGA	Field Programmable Gate Array	Mảng các cổng lập trình
HDL	Hardware Description Language	Ngôn ngữ mô tả phần cứng
IC	Integrated Circuit	Mạch tích hợp
ICAP	Internal Configuration Access Port	Cổng cấu hình trong
IO	Input Output	Cổng vào ra
LFSR	Linear Feedback Shift Register	Thanh ghi dịch phản hồi tuyến tính
LUT	Look-Up Table	Bảng tra
MAC	Message Authentication Code	Mã bản tin xác thực
PCB	Printed Circuit Board	Bảng mạch in
RAM	Random Access Memory	Bộ nhớ truy cập ngẫu nhiên
RNG	Random Number Generator	Bộ tạo số ngẫu nhiên

DANH SÁCH BẢNG

Bảng 2-1: Các hàm f được sử dụng trong thuật toán SHA-1 25

Bảng 2-2: Khối dữ liệu được sử dụng để tính giá trị MAC 26

Bảng 2-3: Thông số tài nguyên sử dụng sau tổng hợp 29

DANH SÁCH HÌNH

Hình 1.1: Quy trình từ HDL đến bitstream. 10
Hình 1.2: Kết hợp quá trình mã hoá và xác thực bitstream 16
Hình 1.3: Mô hình phương thức sử dụng IC xác thực [10]. 18
Hình 2.1: Thiết kế lõi xác thực và ứng dụng. 20
Hình 2.2 Quá trình hoạt động của lõi xác thực. 21
Hình 2.3: Thiết kế chính bộ đếm 8bit. 22
Hình 2.4: Thiết kế bộ tạo số ngẫu nhiên. 22
Hình 2.5: Giản đồ sóng của giao thức 1-Wire 25
Hình 2.6: Lưu đồ thuật toán luồng điều khiển chính. 26
Hình 3.3 Phần mềm mã hóa tệp cấu hình FPGA. 33
Hình 3.4 Giải thuật thực hiện phần mềm. 33
Hình 3.5 Cấu trúc bộ giải mã tệp cấu hình. 34
Hình 3.6 Giải thuật thực hiện trên Microblaze. 34
Hình 3.7 Cấu trúc lõi mật mã GOST 28147-89. 35
Hình 3.8 Đóng gói IP của lõi mật mã GOST 28147-89 sử dụng Xilinx Vivado. 35
Hình 3.9 Mô phỏng thử nghiệm lõi mật mã GOST 28147-89. 36
Hình 3.10 Thử nghiệm sau khi nạp thiết kế bộ cộng. 37

TÓM TẮT LUẬN VĂN

Luận văn mô tả một số phương pháp tấn công chính nhằm sao chép và phân tích trái phép thiết kế FPGA (Field-Programmable Gate Array) và phương pháp bảo vệ tích cực nhằm bảo vệ thiết kế FPGA. Hai giải pháp mã hóa bitstream và giải pháp sử dụng vi mạch xác thực được lựa chọn để thực nghiệm. Giải pháp sử dụng vi mạch xác thực được hướng đến cho các vi mạch FPGA giá thấp không có khả năng tái cấu hình động, không có các tính năng bảo vệ của nhà sản xuất FPGA. Giải pháp được xây dựng dựa trên vi xử lý mềm Picoblaze và vi mạch xác thực DS28E01. Vi xử lý Picoblaze tiến hành quá trình xác thực với DS28E01, từ đó quyết định cho phép thiết kế cần bảo vệ được phép hoạt động. Kết quả thực nghiệm chỉ ra rằng giải pháp thiết kế đáp ứng được yêu cầu đặt ra: chiếm ít tài nguyên của linh kiện FPGA, sử dụng ít chân linh kiện, giao thức đơn giản và giá thành mua linh kiện rẻ. Giải pháp thứ hai là mã hóa bitstream, thiết kế FPGA được tổng hợp, ánh xạ thiết kế vào các tài nguyên trên FPGA, sau đó tạo ra tệp dữ liệu cấu hình và được mã hóa bằng phần mềm trên máy tính. Sau khi được truyền xuống bo mạch, nó được giải mã bằng lõi thuật toán GOST 28147-89 và cấu hình vào vùng tài nguyên FPGA đã định trước. Để thực hiện giải pháp, một bộ giải mã bitstream FPGA được xây dựng dựa trên bộ vi xử lý mềm Microblaze và lõi IP GOST 28147-89. Thực nghiệm cho thấy, giải pháp đề xuất đáp ứng được mục tiêu bảo vệ thiết kế FPGA mà không sử dụng giải pháp bảo vệ của nhà sản xuất vi mạch FPGA.

MỞ ĐẦU

Lý do lựa chọn đề tài

FPGA là linh kiện bán dẫn đa dụng, có thể lập trình và tái lập trình được, thường được sử dụng để mô tả các hàm logic của người thiết kế. Từ năm 2000, các nhà sản xuất FPGA đưa thêm các khối chức năng, trước kia là các ngoại vi bên ngoài, vào trong FPGA. Hiện nay, một linh kiện FPGA có thể chứa một bộ xử lý nhúng, khối xử lý tín hiệu số, các bộ thu truyền dữ liệu gigabit, khối quản lý xung đồng hồ và các bộ chuyển đổi tương tự số (ADC)…

Các thành phần sẵn có và khả năng tái lập trình được làm cho các FPGA có thể sánh ngang với các vi mạch tích hợp chuyên dụng (ASIC: Application Specific Integrated Circuit) khi so sánh về hiệu năng, chi phí và thời gian phát triển. Phần lớn các FPGA thế hệ mới đều được sản xuất dựa trên những công nghệ mới nhất để cạnh tranh so với ASIC về hiệu suất, điện năng và chi phí. Tính năng cấu hình lại, khả năng có thể thực hiện bất kỳ chức năng nào và khả năng tính toán song song làm cho FPGA có ưu thế vượt trội so với các bộ vi xử lý tuần tự.

Sự phát triển trong khả năng và không gian ứng dụng của FPGA tạo nên hai vấn đề về bảo mật. Thứ nhất là các thiết kế FPGA ngày nay tiêu tốn rất nhiều nguồn lực đầu tư, cần phải được bảo vệ. Thứ hai, sự tăng nhanh sử dụng FPGA trong các ứng dụng yêu cầu tính năng bảo mật, do vậy tính năng bảo mật các thiết kế FPGA được quan tâm trong lĩnh vực quân đội, điều khiển-tự động, công nghiệp tiêu dùng, nghiên cứu cộng đồng, mặc dù mỗi lĩnh vực có yêu cầu và quan điểm bảo mật riêng.

Để giải quyết các vấn đề bảo mật, các hãng sản xuất FPGA cũng đã tích hợp các giải pháp bảo mật vào thiết bị của họ. Tuy nhiên, việc thực hiện các giải pháp này thường được triển khai trên các dòng sản phẩm đắt tiền và vẫn có thể tồn tại lỗ hổng làm lộ khóa. Do đó, luận văn đi vào nghiên cứu các biện pháp tấn công và bảo vệ thiết kế cho FPGA dựa trên công nghệ SRAM từ đó thử nghiệm giải pháp cho phép thay thế biện pháp bảo vệ của nhà sản xuất FPGA. Một thuận lợi của học viên là đề tài luận văn cũng là một phần nhiệm vụ nghiên cứu của học viên tại cơ quan công tác. Do đó, học viên có điều kiện tiếp xúc, khai thác các trang thiết bị sẵn có tại đơn vị.

Mục tiêu đề tài

- Nghiên cứu, tìm hiểu các phương pháp tấn công và bảo vệ thiết kế

FPGA.

-Thực nghiệm triển khai hai giải pháp bảo vệ thiết kế FPGA là giải pháp sử dụng cho thiết bị giá thấp sử dụng vi mạch xác thực và giải pháp sử dụng thuật toán mật mã để mã hóa bitstream của thiết kế FPGA. Giải pháp sử dụng vi mạch xác thực cần nhỏ gọn chiếm ít tài nguyên của linh kiện FPGA, giao tiếp với vi mạch xác thực ngoài cần sử dụng ít chân linh kiện. Giải pháp mã hóa bitstream được triển khai dựa hệ nhúng với vi xử lý Microblaze, bộ giải mã thuật toán GOST 28147-89 được mô hình hóa ở mức truyền thanh ghi RTL.

Phương pháp nghiên cứu

Để thực hiện mục tiêu trên, phương pháp nghiên cứu được sử dụng gồm:

- *Phương pháp nghiên cứu lý thuyết*: Nghiên cứu tìm hiểu các phương pháp tấn công và bảo vệ thiết kế FPGA, các thuật toán hàm băm SHA-1, mã khối GOST 28147-89, kỹ thuật lập trình cho hệ vi xử lý, kỹ thuật mô hình hóa một chức năng phần cứng ở mức RTL bằng ngôn ngữ mô tả phần cứng Verilog từ đó nghiên cứu triển khai các giải pháp bảo vệ thiết kế FPGA.

- *Phương pháp thiết kế*: Phát triển và xây dựng các giải pháp dựa trên hệ vi xử lý và lõi IP ở mức RTL. Tổng hợp phần cứng với công nghệ FPGA;

- *Phương pháp mô phỏng và kiểm chứng*: Mô phỏng thiết kế lõi IP mật mã trên phần mềm ISE Simulator và Vivado Simulator nhằm kiểm tra chức năng và đánh giá hiệu năng của lõi mật mã.

- *Phương pháp kiểm thực*: Kiểm nghiệm thiết kế trên các bo mạch phát triển.

Kết cấu luận văn

Cấu trúc của luận văn được bố cục thành 3 chương: Chương 1 đề cập đến một số phương pháp tấn công và bảo vệ thiết kế FPGA. Chương 2 đề cập đến giải pháp bảo vệ thiết kế FPGA thông qua vi mạch xác thực. Giải pháp mã hóa thiết kế FPGA sẽ được trình bày trong Chương 3. Cuối cùng là một số kết luận và hướng phát triển tiếp theo.

Chương 1 MỘT SỐ PHƯƠNG PHÁP TẤN CÔNG VÀ BẢO VỆ THIẾT KẾ FPGA

1.1 Một số phương pháp tấn công

1.1.1 Nhân bản, dán nhãn sai và sản xuất vượt số lượng

FPGA là chip đa năng, có nghĩa rằng dữ liệu cấu hình (bitstream) cung cấp cho một thiết bị có thể được sử dụng cấu hình cho bất kỳ linh kiện FPGA cùng họ và có kích thước tương đồng. Như vậy, kẻ tấn công có thể làm bản sao bằng việc ghi lại bitstream khi chúng được truyền tải đến FPGA và sử dụng trong hệ thống và sản phẩm khác, việc làm nhái sẽ rẻ hơn nhiều so với bản gốc. Việc nhân bản không yêu cầu quá nhiều tài nguyên so với việc phân tích logic và không yêu cầu kỹ sư cao cấp. Điều này có thể được coi là lỗ hổng bảo mật của FPGA SRAM. Kẻ tấn công, người không cần hiểu chi tiết về thiết kế, có thể coi thiết kế như là một hộp đen và chỉ cần đầu tư vào việc sao chép các bảng mạch mà FPGA được gắn trên đó, nên tiết kiệm được khoản chi phí phát triển.

Nhà thiết kế và phát triển hệ thống thường có hai mối lo ngại chính liên quan đến nhân bản. Thứ nhất, các hệ thống nhái sẽ gây tổn hại về doanh thu sau khi đã đầu tư lớn cho quá trình nghiên cứu phát triển sản phẩm. Thứ hai, các sản phẩm nhái luôn có chất lượng kém hơn nhiều so với sản phẩm gốc, nên nếu hệ thống giả giống như hệ thống gốc, sẽ làm mất danh tiếng và tăng chi phí hỗ trợ khách hàng. Do vậy, phương pháp hiệu quả nhất để chống lại kẻ tấn công đó là tăng giá thành cho việc tấn công nhân bản thành công, với phương pháp này, lợi nhuận thu được từ việc nhân bản các thiết kế tiến đến không.

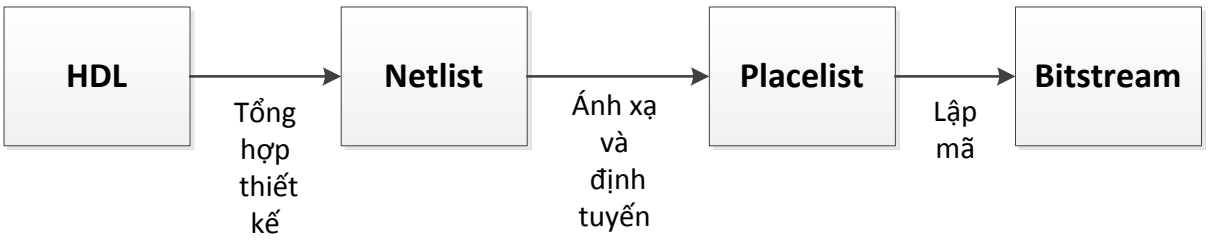
Ngành công nghiệp điện tử đã đối mặt với việc làm giả phần cứng trong nhiều thập kỷ qua, hầu hết đến từ khu vực Châu Á. Bên cạnh việc sao chép trộm thiết kế, sản xuất vượt định mức cũng là mối quan tâm lớn cho nhiều công ty. Khi sản phẩm được sản xuất bởi bên thứ ba, quá trình sản xuất, lắp ráp và kiểm tra phần cứng trước khi đưa đến người tiêu dùng được thực hiện bởi bên thứ ba, do đó sản phẩm có thể sản xuất nhiều hơn số lượng đặt hàng và bán vượt quá mà không phải chịu chi phí phát triển. Thậm chí, các mẫu thiết kế có thể bị bán cho đối thủ cạnh tranh (PCB layout, bitstream). Để tránh điều này, các công ty sản xuất phải đủ điều kiện cơ sở vật chất, đáng tin cậy và cần "giám sát" họ, nhưng điều này là khó có thể thực hiện ở nhiều quốc gia và không khả thi cho nhiều công ty nhỏ.

Dán nhãn sai FPGA cũng là vấn đề lớn đối với các nhà sản xuất FPGA và phát triển hệ thống. Sửa đổi hoặc xóa đánh dấu trên một gói IC là điều bình

thường và các nhà thiết kế hệ thống đã làm như vậy trong nhiều năm, để khiến cho các kỹ thuật đảo ngược hệ thống thêm khó khăn. Tuy nhiên, khi FPGA không được mua thông qua các nhà phân phối được ủy quyền, người dùng khó có thể chắc chắn được các thành phần bên trong được đóng gói theo đúng như bao bì. Nếu là một thiết bị hoàn toàn khác nhau, hoặc một thành viên trong họ FPGA nhưng nhỏ hơn, điều đó là đơn giản để kiểm tra, mặc dù sau khi mua. Mức tốc độ (speed grade) của thiết bị cho phép xác định tần số tối đa mà thiết bị có thể hỗ trợ, tuy nhiên đây là thông số rất khó có thể đo kiểm, nên một FPGA có mức tốc độ thấp hơn có thể được ghi nhãn sai ở mức cao hơn và bán với giá cao hơn so với giá trị thật. Không có cách nào cho người mua hoặc người bán thực sự biết những gì được đóng gói bên trong chip, ngoài việc cấu hình chúng và quan sát kết quả. Đối với các công ty thương mại, phương thức an toàn nhất là mua các thiết bị từ nhà cung cấp hoặc từ nhà phân phối chứ không phải mua trên mạng, tuy nhiên đối với một số người chỉ cần số lượng nhỏ với giá rẻ, sẽ có nguy cơ phải đối diện với sự gian dối này.

1.1.2 Kỹ thuật đảo ngược

Có thể định nghĩa kỹ thuật đảo ngược (reverse engineering) bitstream là thực hiện biến đổi thông tin trong bitstream thành mô tả chức năng của thiết kế ban đầu hoặc kỹ thuật đảo ngược là quá trình xử lý biến đổi bitstream quay trở lại ngôn ngữ mô tả phần cứng (Hardware decription language – HDL) hoặc netlist.



Hình 1.1: Quy trình từ HDL đến bitstream.

Từ dữ liệu đảo ngược có thể xác định dữ liệu quan trọng từ bitstream, như các khóa, nội dung BRAM/LUT hoặc trạng thái các cell bộ nhớ, mà không cần khôi phục đầy đủ chức năng. Kỹ thuật đảo ngược là hợp pháp ở nhiều nước với một số hạn chế để thực hiện tương tác hoặc phát hiện hành vi xâm phạm bằng sáng chế hoặc các quyền khác. Đảo ngược toàn bộ bitstream sẽ biết toàn bộ thiết kế và các dữ liệu có thể được sử dụng để sản xuất bitstream hoàn toàn khác so với ban đầu, như vậy sẽ khó chứng minh được hành vi xâm phạm bản quyền. Ngoài ra, dữ liệu bí mật ẩn trong bitstream sẽ bị phát hiện bởi kẻ tấn công.

Lập mã (encode) bitstream thường không được mô tả và không rõ ràng, nhà cung cấp FPGA giữ mã hóa này như một bí mật. Sự khó hiểu, độ phức tạp và kích thước bitstream làm cho quá trình xử lý ngược trở nên khó khăn và tốn thời gian, mặc dù về mặt lý thuyết thì có thể thực hiện được. Hiện nay, chưa có báo cáo nào về sự đảo ngược thành công các bitstream FPGA như định nghĩa ở trên hoặc đánh giá tính khả thi về thời gian dựa trên dữ liệu và phân tích thực nghiệm. Nguy cơ này chắc chắn bị ngăn chặn một cách có hiệu quả bằng các cơ chế pháp lý trong môi trường khoa học và trong môi trường thương mại, mặc dù một số tổ chức và một số quốc gia sẽ ít quan tâm về vấn đề này. Tuy nhiên, việc sử dụng FPGA càng ngày trở nên phổ biến và giá trị của các thiết kế trên FPGA càng tăng, thì các tấn công đảo ngược sẽ càng nhanh chóng phát triển.

Tách dữ liệu từ RAM và LUT từ bitstream không phải là mới, tuy nhiên, phần khó nhất của quá trình xử lý là quá trình chuyển đổi tự động các cấu trúc triển khai (placelist) trở lại thành netlist, từ đó những chức năng thiết kế ban đầu có thể được lấy ra. Một số dự án và công ty đã công bố thực hiện thành công qua trình này nhưng rất khó để kiểm chứng.

1.1.3 Kỹ thuật tấn công đọc lại

Kỹ thuật tấn công đọc lại là một quá trình xử lý lấy lại một bản chụp của trạng thái hiện tại FPGA trong khi đang hoạt động. Theo yêu cầu, FPGA sẽ gửi bản chụp bao gồm cấu hình, LUT và nội dung bộ nhớ đến máy chủ hoặc thiết bị khác, thông qua cổng cấu hình. Ảnh này khác so với bitstream gốc bởi nó không có header, footer, lệnh khởi tạo và no-ops; dữ liệu động trong LUT và BRAM khác với trạng thái khởi tạo. Đọc ngược là một công cụ có đặc tính mạnh mẽ trong việc xác minh và kiểm tra sản phẩm FPGA của nhà sản xuất và nhà phát triển, cho phép nhà phát triển hệ thống xác minh tính đúng đắn của thiết kế như đang hoạt động trên chính FPGA.

Tuy nhiên, nếu kẻ tấn công có thể đọc lại thiết kế, thêm header và footer sẽ tạo thành một phiên bản chỉnh sửa và sử dụng trong một thiết bị khác, ngoài ra, nó cũng có thể thực hiện kỹ thuật phân tích đảo ngược. Đồng thời, với phương thức "readback difference attack", kẻ tấn công có thể quan sát tín hiệu thay đổi trên từng chu kỳ xung đồng hồ, từ đó có thể bỏ qua cơ chế bảo vệ. Xem xét trường hợp một lõi chức năng đợi một tín hiệu cho phép từ quá trình xác thực. Nếu người tấn công có được kiểm soát đầu vào clock, họ có thể chụp ảnh trước khi tín hiệu được thiết lập, cấp tín hiệu clock và sau đó sẽ chụp những ảnh khác. Thông qua một quá trình lặp đi lặp lại tương đối dễ, so sánh các ảnh, kẻ

tấn công có thể xác định bit thay đổi trạng thái của tín hiệu. Sau đó, bitstream gốc có thể bị chỉnh sửa để có tín hiệu cho phép mãi mãi. Tuy nhiên, phương pháp đọc ngược cũng có thể được sử dụng như một cơ chế bảo vệ bằng việc cung cấp các dấu hiệu giả mạo.

Xilinx cung cấp một bit trong bitstream cho việc loại bỏ khả năng đọc ngược, nhưng lại dễ dàng tìm thấy qua các mô tả của hãng. Tuy nhiên, khi mã hóa bitstream được sử dụng nó sẽ vô hiệu hóa một số thanh ghi trong FPGA để ngăn chặn quá trình đọc ngược. Về lý thuyết, bit vô hiệu hóa này có thể xác định thông qua tấn công không xâm lấn, nhưng không có cơ sở chứng minh phương thức có thể thành công khi thực hiện. Thiết bị Altera không có khả năng readback và do đó không thể bị tấn công theo phương pháp này.

1.2 Một số phương pháp bảo vệ

Tính hiệu quả của một cơ chế bảo vệ được đánh giá bằng chi phí, kỹ năng, công cụ và thời gian cần thiết để phá vỡ các bảo vệ đó. Trước khi nghiên cứu các kỹ thuật bảo vệ, có thể phân loại các phương pháp bảo vệ như sau:

- Phương pháp răn đe xã hội (Social deterrents) cung cấp bởi hệ thống pháp luật và dựa vào vấn đề hành chính, vấn đề truy tố và tạm giam. Thiết kế có thể được bảo vệ về nhãn hiệu, bản quyền, bí mật thương mại, bằng sáng chế, hợp đồng, thoả thuận cấp phép hay gọi chung là " sở hữu trí tuệ". Tuy nhiên, vấn đề răn đe xã hội chỉ hiệu quả tại nơi có luật pháp được tôn thủ chặt chẽ và được thi hành nghiêm túc. Thái độ với quyền sở hữu thiết kế ở các quốc gia rất khác nhau, điều này không hoàn toàn ngăn chặn được vấn đề này. Ở hầu hết các nước hàng giả được sản xuất, quyền sở hữu có xu hướng không được thi hành.
- Phương pháp bảo vệ tích cực (Active deterrents) là cơ chế mật mã và vật lý nhằm ngăn chặn hành vi trộm cắp và sử dụng trái phép thiết kế. Phương pháp bảo vệ tích cực có tính hiệu quả cao nếu thực hiện chính xác. Kết hợp với việc răn đe xã hội, phương pháp bảo vệ tích cực có thể giúp thuyết phục toà án rằng các nhà thiết kế đưa ra những biện pháp thích hợp nhằm bảo vệ thiết kế và thủ phạm đã chủ động phá vỡ chúng.
- Phương pháp bảo vệ phản ứng (Reactive deterrents) cung cấp cho việc phát hiện hoặc ghi nhận bằng chứng về sự xâm nhập hoặc gian lận. Phương pháp này dựa trên một vài cơ chế, như đóng dấu thuỷ phân (watermark), dấu vết cá nhân (fingerprinting) và ẩn mã (steganography); phương pháp này hữu ích cho quá trình điều tra hoặc cải thiện an ninh hệ thống sau khi bị tấn công hoặc xâm nhập.

Trong luận văn này, học viên chủ yếu giới thiệu phương pháp bảo vệ chủ động.

1.2.1 Mã hóa bitstream

Mã hóa là chức năng cung cấp tính tin cậy cho dữ liệu và phụ thuộc vào khóa bí mật, ngay cả khi thuật toán được công khai. Mã hóa bitstream tại thời điểm cuối của quá thiết kế và giải mã trong FPGA có thể chống lại sự nhân bản, kỹ thuật đảo ngược và một vài trường hợp khác, nó cung cấp tính năng bảo vệ chống giả mạo. Mã hóa dữ liệu cấu hình cho thiết bị có thể lập trình được đề nghị vào năm 1992 trong một bằng sáng chế của Austin [8], được thực hiện lần đầu trong các thiết bị Actel's 60RS, tuy nhiên nó là một ví dụ tồi trong việc phân phối khóa. Sau khi bitstream được sản xuất, phần mềm yêu cầu khóa từ người dùng và mã hoá dữ liệu cấu hình của bitstream. Người dùng sau đó "lập trình" khóa đã sử dụng trên FPGA. Thông tin cấu hình trong bitstream đã được mã, được truyền đến các bộ nhớ thông qua bộ giải mã hoá trước đó. Kẻ tấn công có bitstream mã hoá không sử dụng được vì không có khóa; do đó không thể đảo ngược hoặc sử dụng trong một thiết bị khác (giả sử khoá khác nhau được tải vào mỗi FPGA). FPGA Altera Stratix II và III cho phép người thiết kế chọn bitstream bắt buộc phải qua bộ giải mã, không cho phép bitstream không mã hoá. Điều này ngăn cản việc thực thi bất kỳ bitstream không được mã với khóa đúng, nhưng không ngăn chặn tấn công từ chối dịch vụ cố gắng liên tục đưa đến một bitstream không hợp lệ.

a. Lưu trữ khóa

Khoá cần được cung cấp bên trong thiết bị lúc giải mã bitstream, có hai công nghệ lưu trữ khóa được sử dụng hiện nay: bay hơi (volatile) và không bay hơi (non-volatile). Sử dụng kiểu lưu trữ bay hơi, khóa được giữ trong các bộ nhớ SRAM và được cung cấp nguồn bởi pin gắn ngoài khi nguồn nuôi FPGA không được cung cấp. Ưu điểm của kiểu lưu trữ khoá này là cho phép xoá khoá (cơ khí hoặc điện) để chống lại tấn công vật lý khi thiết bị không cấp nguồn, và buộc những kẻ tấn công cấp điện áp cho thiết bị trong suốt quá trình tấn công bán xâm lấn và tấn công xâm lấn. Những thuộc tính này chủ yếu thu hút các nhà thiết kế bảo mật phù hợp với chuẩn FIPS 140-2. Nhược điểm chính là pin chiếm không gian PCB (đặc biệt với nơi giữ pin), hỏng hóc và có thể cần được thay thế. Do đó pin là điều đáng quan tâm cho các nhà thiết kế xem xét chi phí hợp lý, khi không nhất thiết phải cần công nghệ bảo mật cao nhất.

Sử dụng lưu trữ không bay hơi, khóa được nhúng cố định vào trong các thiết bị sử dụng cầu trì, lập trình laser, Flash hoặc EEPROM. Tuy nhiên, tích hợp bộ nhớ không bay hơi với công nghệ CMOS là thách thức khi thêm một bước sản xuất phi chuẩn có ảnh hưởng đến chi phí, năng suất và độ tin cậy. Đó là lý do tại sao nhà sản xuất FPGA mới bắt đầu quan tâm đến kiểu lưu trữ khóa không bay hơi. Nhúng khóa có ưu điểm là không yêu cầu thiết bị ngoài và chi phí thấp hơn so với giải pháp dùng pin. Nhúng khóa có thể chống lại việc gian lận với các khóa có thể lập trình vào các thiết bị tại cơ sở đáng tin cậy trước khi đưa đến bên thứ ba lắp ráp và thử nghiệm hệ thống.

Xilinx cung cấp kiểu lưu trữ khóa bay hơi với mã hoá Triple-DES cho VirtexII/PRO và AES-256 cho Virtex-4/5 và mới hơn. Nếu mã hoá được sử dụng, cơ chế readback và cấu hình từng phần sẽ bị vô hiệu hoá. Tuy nhiên các cổng truy cập cấu hình nội bộ (ICAP) vẫn được kích hoạt, cho phép đọc nội dung cấu hình và gửi ra thông qua cổng IO thông thường. Do đó khi mã hoá được sử dụng, các nhà thiết kế nên lưu ý đến cuộc tấn công Trojan horse có thể được chèn vào bởi một nhân viên hoặc nhà cung cấp lõi. Lattice cung cấp lưu trữ khoá không bay hơi và sử dụng thuật toán mã hoá AES-128 trong ECPM2/M. Altera Stratix II FPGA có lưu trữ khóa không bay hơi cho mã hoá AES-128, nhưng yêu cầu các thành phần bên ngoài trên mạch in để tạo các điện áp cần thiết cho lập trình khoá. Với Stratix III, Altera cung cấp bộ nhớ bay hơi lập trình qua JTAG và bộ nhớ khoá không bay hơi cho việc mã hoá bitstream sử dụng thuật toán AES-256, với sơ đồ yêu cầu hai khóa, một chìa khoá được sử dụng để mã hoá một 'khóa thực', cái được sử dụng cho việc mã hoá bitstream. Trong FPGA, khóa thực có thể được xáo trộn, sau đó được lưu trữ trong bộ nhớ phân tán làm cho kẻ tấn công xâm lấn và bán xâm lấn trở nên khó khăn trong việc tìm vị trí khoá. Khi bitstream đã mã hoá gửi đến FPGA, một chức năng xáo trộn nghịch đảo được sử dụng để sản xuất khóa thực cho giải mã.

b. Quản lý khoá

Chuẩn NITS FIPS 800-57 nhận xét rằng "quản lý khóa thường là một bước cuối trong quá trình phát triển mật mã. Kết quả là, hệ thống mật mã thường không hỗ trợ khả năng và giao thức quản lý khóa, nhưng lại là yếu tố cần thiết để cung cấp việc bảo mật đầy đủ…Vấn đề quản lý khóa cần được đặt ra trong giai đoạn mô hình/ phát triển ban đầu của vòng đời phát triển mật mã". Mặc dù đây là một trong những khía cạnh quan trọng nhất của vấn đề bảo mật, và cũng được xem như một phần của chiến lược phòng thủ để bảo vệ bản quyền và sự phân phối thiết kế.

Quản lý khóa là quá trình xử lý việc tạo khóa và phân phối khóa. Như vậy bảo mật hệ thống dựa trên khóa bí mật, thì cơ sở hạ tầng quản lý khóa là quan trọng trong vấn đề lựa chọn thuật toán, giao thức và thực hiện chúng. Quy trình giữ khóa bí mật trong khi lưu trữ, vận chuyển và cập nhật, cùng với vấn đề kiểm soát truy cập cần đưa vào tổng chi phí của chiến lược bảo vệ, thủ tục quản lý khoá nên được quy định chặt chẽ.

Ta có thể xem xét một ví dụ đơn giản về bài toán quản lý khoá cho mã hoá bitstream. Bước đầu tiên là thiết lập giá trị khóa. Nếu một khóa sử dụng cho việc mã hoá toàn bộ sản phẩm, chi phí sẽ thấp nhưng kết quả sẽ là một thảm hoạ lớn, khi một thiết bị bị lộ khóa sẽ làm suy yếu mức độ bảo mật của toàn bộ các hệ thống. Mặt khác, nếu mỗi bitstream được mã hoá với một khóa duy nhất, chi phí cần cao hơn, nhưng sự phá hoại sẽ chỉ xảy ra ở một hệ thống cục bộ. Giải pháp có thể nằm giữa hai cực và quyết định dựa trên mức độ tin tưởng cho hệ thống phát triển và những nguy hiểm do chung khóa gây nên. Bước tiếp theo là xác định các khóa chính và mã hoá bitstream. Hệ thống phát triển cần xác định tập hợp người đáng tin cậy cho việc xử lý khóa và quyết định nếu tin tưởng phần mềm các nhà cung cấp không để rò rỉ các khóa hoặc viết phần mềm mã riêng. Và sau đó sử dụng máy tính đáng tin cậy trong môi trường an toàn cho việc mã hoá bitstream và nạp khóa vào trong FPGA. Dữ liệu cơ sở của khóa cần được bảo vệ vật lý và kiểm soát, được giới hạn cho người có quyền hạn thích hợp.

c. Một số vấn đề với phương pháp mã hoá

Mặc dù mã hoá có thể đảm bảo tính bảo mật dữ liệu cấu hình, những kẻ tấn công vẫn có thể xây dựng một bitstream không giống với bản gốc nhưng vẫn hợp lệ. Mã khối có thể hoạt động trong một số chế độ, trong các chế độ đó, các bản tin có độ dài lớn chia thành khối n bit để xử lý. Mỗi một chế độ có ưu điểm và nhược điểm tuỳ thuộc vào ứng dụng. Ví dụ, chế độ ECB sẽ mã hoá một khối độc lập với khối khác, kết quả là các khối dữ liệu giống nhau sẽ cho các bản mã giống nhau. Đây được xem là điểm yếu cho bản tin với các khối lặp đi lặp lại – như bitstream FPGA vì kẻ tấn công có thể quan sát được khối như nhau và có thể hoán đổi chúng. Chế độ CBC được sử dụng ngăn chặn sự lặp đi lặp lại của chế độ ECB bằng việc bắt đầu với việc mã hoá một vector khởi tạo ngẫu nhiên (Initialization Vector) và bản mã mỗi khối sau phụ thuộc vào kết quả bản mã của khối trước. Mã hoá không phát hiện sự thay đổi của bản mã, kẻ thù có thể chuyển đổi m bits trong một khối của một bản mã dòng bit của thiết kế khi không biết khoá. Đương nhiên, những khối mà một trong những bits được sửa lại có thể không chứa dữ liệu có liên quan đến phần tấn công thiết kế. Tấn công

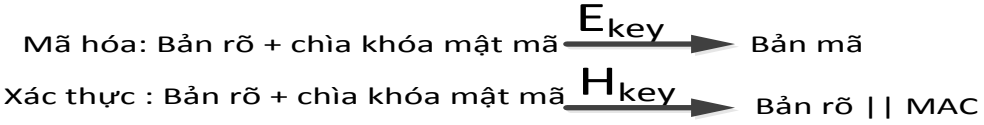
này có thể sử dụng với mục đích sẽ tìm được tập hợp bit khi thay đổi có thể vô hiệu hoá cơ chế bảo vệ. Chế độ mã hoá cần phải được lựa chọn cẩn thận theo các yêu cầu bí mật của ứng dụng. Họ Virtex của Xilinx sử dụng chế độ CBC trong khi Altera và Lattice không chỉ rõ chế độ hoạt động mã khối.

1.2.2 Xác thực bitstream

Xác thực bitstream được xem như là một giải pháp cho một số vấn đề liên quan đến bảo mật FPGA. Xác thực cung cấp hai điều: thứ nhất là xác định thực thể, cho phép người nhận tin nhắn chắc chắn về người gửi tin nhắn; thứ hai là đảm bảo toàn vẹn thông điệp, thông điệp không bị thay đổi trong khi truyền. Xác thực đôi khi được coi trọng hơn mã hóa, khi các thiệt hại của một kẻ tấn công có thể gây ra bằng phương thức mạo danh có thể lớn hơn nhiều so với việc người đó có khả năng đọc được thông tin liên lạc bí mật. Mã hóa bitstream là một bước tiến tốt để bảo vệ thiết kế, mặc dù chúng ta đã thấy rằng nó không có nghĩa là để đảm bảo tính toàn vẹn của dữ liệu, nó chỉ có tính bảo mật. Do đó, mã hóa bảo vệ các bitstream chống nhân bản và kỹ thuật đảo ngược trong môi trường độc lập của FPGA trong khi xác thực bảo đảm chính xác và hoạt động của bitstream khi nó chạy trên FPGA.

Các nhà cung cấp FPGA thường sử dụng kiểm tra tuyến tính, chẳng hạn như mã kiểm tra dư thừa tuần hoàn (CRC), để bảo vệ FPGA từ bitstreams bị hỏng do lỗi truyền dẫn trên một kênh bị nhiễu. Kết quả của một bitstream bị hỏng sẽ làm sai chức năng và chập mạch có thể làm hỏng thiết bị nếu cường độ dòng điện đủ lớn và kéo dài trong một thời gian dài. Mã tuyến tính tốt cho việc phát hiện lỗi bit không chủ ý, nhưng các tấn công xử lý chèn bit có thể thực hiện. Tuy vậy, kiểm tra tuyến tính cũng thiếu một yêu cầu quan trọng của xác thực: xác định thực thể.

Xác thực thường được thực hiện bằng cách tính toán một mã xác thực (MAC). MAC được tạo ra bởi hàm một chiều và không va chạm, được tạo ra từ một bản tin có độ dài tùy ý và kết quả là một chuỗi bit có độ dài cố định. Như vậy, một kẻ tấn công khó có thể tính toán để tìm thấy những bản tin ban đầu, hoặc tìm ra hai thông điệp khác nhau mà kết quả là một chuỗi. Việc xác định thực thể đạt được bằng cách kết hợp một khóa chung vào quá trình này.



Hình 1.2: Kết hợp quá trình mã hoá và xác thực bitstream

Chuỗi MAC được gắn vào các tin nhắn, có thể hoặc không được mã hóa, và được xác minh ở người nhận bằng cách tính toán lại MAC của tin nhắn một lần nữa. Nếu những giá trị MAC sau tính toán giống nhau, người nhận biết rằng thông điệp là xác thực và rằng bất cứ ai tạo ra nó là người sở hữu khóa chia sẻ trước. MAC yêu cầu các khóa đối xứng được thiết lập trước khi gửi thông báo nhưng xác thực dựa trên chữ ký cũng có thể được thực hiện bằng cách sử dụng mã hóa khóa công khai trong đó mỗi người tham gia có một khóa riêng và khoá công khai. Bằng việc ký với khóa riêng, một chữ ký được tạo ra có thể được xác minh với các khóa công khai. Mật mã chìa khoá công khai có tất cả các thuộc tính cần thiết cho việc giải quyết bài toán bảo mật khoá bitstream, ngoại trừ việc triển khai hiện nay là quá tốn kém.

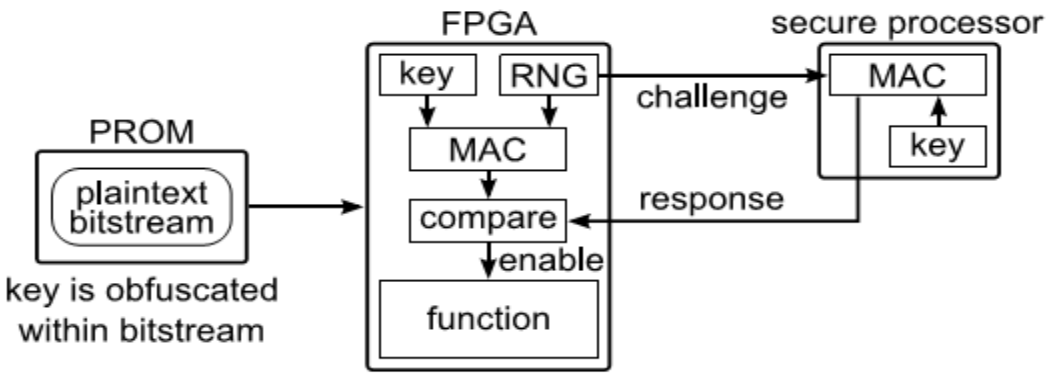
Đối với việc tính toán MAC, nó có thể được tạo ra bằng hàm băm hoặc mã khối ở chế độ nhất định. Mã khối chế độ CBC-MAC có thể được sử dụng để tính toán một MAC đơn giản bằng cách lấy kết quả khối mã hoá cuối cùng là giá trị MAC. CBC-MAC là không an toàn cho bản tin có độ dài tuỳ biến, và NIST hiện nay đã được tiêu chuẩn hóa một chế độ được gọi là CMAC để khắc phục hạn chế này. Trong trường hợp này, không nên sử dụng một chìa khoá cho cả mã hóa và tính toán MAC với một bản rõ. Việc sử dụng chung chìa khoá cho cả hai hoạt động này là một mô hình mới, một chế độ mới và thuật toán mã hóa đã được đề xuất để làm điều đó một cách an toàn; được gọi chung là mã hóa chứng thực (AE).

Để xác thực bitstream, Parelkar đề xuất và đánh giá việc sử dụng các thuật toán AE khác nhau và hàm băm. Parelkar kết luận CCM là tốt nhất, phù hợp khi xét về hiệu suất và kích thước. Parelkar và Gaj đề xuất việc sử dụng chế độ EAX. Tuy nhiên, một lý do quan trọng tại sao mã hóa chứng thực có thể không lý tưởng là bởi vì nó sử dụng một khóa duy nhất cho cả hai hoạt động, đặc điểm này là một bất lợi khi các hoạt động cần phải được tách do cân nhắc về quản lý chìa khoá, kiểm soát truy cập, xác nhận nhiều đối tượng.

1.2.3 Sử dụng IC xác thực

Ngoài các phương pháp trên, các nhà phát triển ứng dụng không phải mã hóa an toàn nhưng có thể làm tăng chi phí quá trình nhân bản với chi phí hợp lý. Những giải pháp này phù hợp cho các thiết bị cấp thấp không có khả năng mã hóa bitstream. Năm 2000, Kessner [7] đề xuất một biện pháp chống lại hành vi trộm cắp bitstream FPGA với một CPLD gửi một dòng khoá LFSR cho FPGA để so sánh với chuỗi khoá tương tự được tạo ra bên trong để xác minh rằng nó được gắn trên các bo mạch chính hãng. Gần đây hơn, cả hai nhà sản xuất FPGA

là Altera và Xilinx đã đề xuất mô hình chống lại quá trình nhân bản, được phác thảo trong Hình 1.3. FPGA chia sẻ một khóa với một bộ xử lý xác thực đặt bên cạnh nó. Các FPGA sẽ gửi một số ngẫu nhiên được tạo bởi một bộ tạo số ngẫu nhiên và cả hai thực hiện một tính toán chính dựa trên nó. Các thiết bị xử lý xác thực gửi kết quả lại cho FPGA. FPGA so sánh với kết quả nhận tự tính; nếu hai kết quả giống nhau, một tín hiệu cho phép được gửi đến các phần còn lại trên FPGA. Kết quả là các thiết kế đã xác nhận rằng FPGA nó đang hoạt động trên các bảng mạch in mong đợi. Trong khi thực hiện giải pháp này, bộ tạo số ngẫu nhiên phải không dễ dàng bị ảnh hưởng bởi nhiệt độ và điện áp thay đổi, cũng như một số tấn công khác.



Hình 1.3: Mô hình phương thức sử dụng IC xác thực [10].

Từ họ sản phẩm Spartan-3A, Xilinx cung cấp một "DNA", là một bộ nhớ không bay hơi, được thiết lập bởi nhà máy sản xuất, người dùng có thể truy cập. Xilinx đề nghị sử dụng số này như là một "khóa" cho các chương trình thẩm tra thiết kế sử dụng một thiết bị bên ngoài như cách đã mô tả ở trên. Tuy nhiên, dãy số trên không phải là một bí mật - nó có thể được đọc bởi bất cứ ai, do đó khó có thể đảm bảo an toàn.

1.3 Kết luận chương

Chương này trình bày một số phương pháp tấn công và phương pháp bảo vệ các thiết kế FPGA. Bên cạnh phương pháp tấn công đã được nêu trong chương này, các thiết kế FPGA hoàn toàn có thể bị khám phá về chức năng bởi các phương pháp tấn công phần cứng như tấn công kênh kề (side-channel attack), tấn công vật lý xâm lấn và bán xâm lấn. Để thực hiện bảo vệ chống sử dụng trái phép thiết kế FPGA, luận văn lựa chọn phương pháp thứ ba và giải pháp mã hóa như đã mô tả ở phần trên để thực nghiệm và trình bày ở các chương sau.

Chương 2 GIẢI PHÁP BẢO VỆ THIẾT KẾ FPGA BẰNG IC XÁC THỰC

Trong chương trước, chúng ta đã tìm hiểu một số phương pháp tấn công các thiết kế trên FPGA và một vài giải pháp bảo vệ thông qua mã hóa bistream, xác thực bistream và sử dụng IC xác thực. Chương này sẽ đề cập đến giải pháp cụ thể để bảo vệ thiết kế FPGA thông qua vi mạch xác thực.

2.1 Lựa chọn thành phần

a. Yêu cầu thiết kế

Vì thiết kế hướng đến sử dụng kèm theo trên các linh kiện FPGA giá rẻ không có hỗ trợ mã hoá bitstream và xác thực bitstream, nên lõi xác thực cần đạt các yêu cầu sau:

- Thiết kế lõi cần nhỏ gọn chiếm ít tài nguyên của linh kiện.
- Giao diện với IC xác thực ngoài cần sử dụng ít chân linh kiện.
- Giao thức giao tiếp giữa hai linh kiện đơn giản.
- Chi phí mua IC xác thực ngoài thấp.
- Thuật toán xác thực đáp ứng được yêu cầu bảo mật.

b. Lựa chọn thành phần

Hiện tại, các thiết kế và dự án được triển khai trên FPGA tại đơn vị công tác của học viên đều chủ yếu thực hiện trên linh kiện FPGA của Xilinx, nên lõi thiết kế trong luận văn này cũng hướng tới triển khai trên các sản phẩm của Xilinx. Với những ưu điểm của bộ vi xử lý Picoblaze tiết kiệm tài nguyên, đơn giản và sự ưu tiên sử dụng linh kiện Xilinx, nên Picoblaze sẽ là trái tim của toàn bộ thiết kế.

Thuật toán xác thực SHA-1 được lựa chọn để xác thực thiết kế. SHA-1 được sử dụng bởi khả năng mật mã tốt hơn so với SHA-0 và nhỏ gọn hơn so với SHA-256, nên SHA-1 thích hợp với việc thực hiện trên Picoblaze có bộ nhớ chương trình giới hạn. Ngoài ra, SHA-1 đã ra đời được khoảng thời gian khá dài, nên các lựa chọn cho IC xác thực bên ngoài phong phú hơn và giá thành linh kiện rẻ hơn.

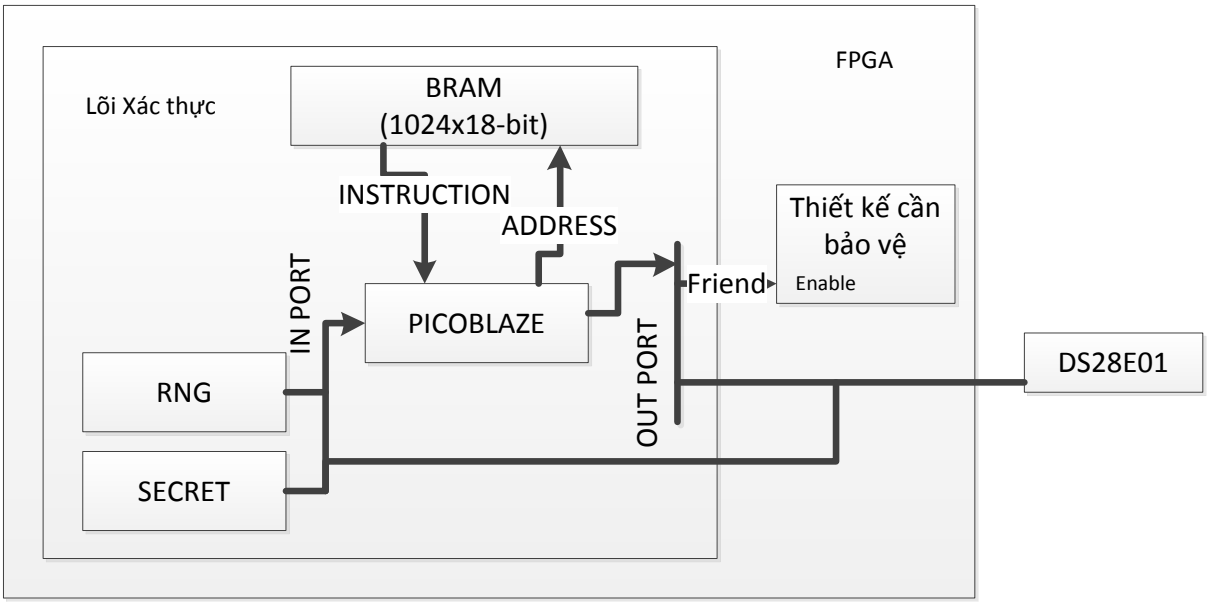
IC xác thực có hai hãng chính mà luận văn hướng đến lựa chọn là Maxim Integrated và Atmel. Cả hai hãng điện tử đều có các sản phẩm phù hợp với thuật toán lựa chọn, ngoài ra giá linh kiện thấp phù hợp với yêu cầu đã đặt ra.Tuy nhiên, tôi lựa chọn linh kiện của Maxim Integrated, và cụ thể là DS28E01-100.

Đầu tiên, tôi sử dụng linh kiện của Maxim Integrated bởi linh kiện sử dụng giao thức 1-Wire đơn giản, tốn ít tài nguyên khi lập trình trên Picoblaze (ATMEL sử dụng giao thức I2C phức tạp hơn). Thứ hai, tôi có thể tiếp cận với tài kiệu kĩ thuật và linh kiện mẫu từ Maxim dễ dàng hơn.

c. Giới thiệu chungDS28E01-100

DS28E01-100 kết hợp bộ nhớ EEPROM 1024 bits với xác thực challenge-and-response thực hiện thuật toán hàm băm SHA1 theo tiêu chuẩn ISO/IEC10.118-3. Thiết bị này có thể xử lý SHA-1 đầu vào gồm khối bí mật 64 bits kết hợp với một 40-bit ngẫu nhiên và dữ liệu bên trong linh kiện để cung cấp một mức độ bảo mật xác thực giữa một hệ thống chủ và linh kiện phụ trợ. Các mảng EEPROM 1024-bit được cấu hình thành bốn trang 256 bits với một bộ nhớ đệm 64-bit để thực hiện các hoạt động ghi. Tất cả các trang bộ nhớ có thể được đặt ở chế độ chống ghi, và một trang có thể được đặt ở chế độ EPROM-emulation. Mỗi DS28E01-100 có số định danh 64-bit riêng đảm bảo tính duy nhất của ROM được nhà máy "khắc laze" vào chip. Các DS28E01-100 giao tiếp qua bus 1-Wire và tuân thủ theo chuẩn giao thức 1-Wire. Để có thêm thông tin về đặc tính, tham số điện người đọc có thể tham khảo thêm tài liệu kỹ thuật của hãng.

2.2 Thiết kế giải pháp

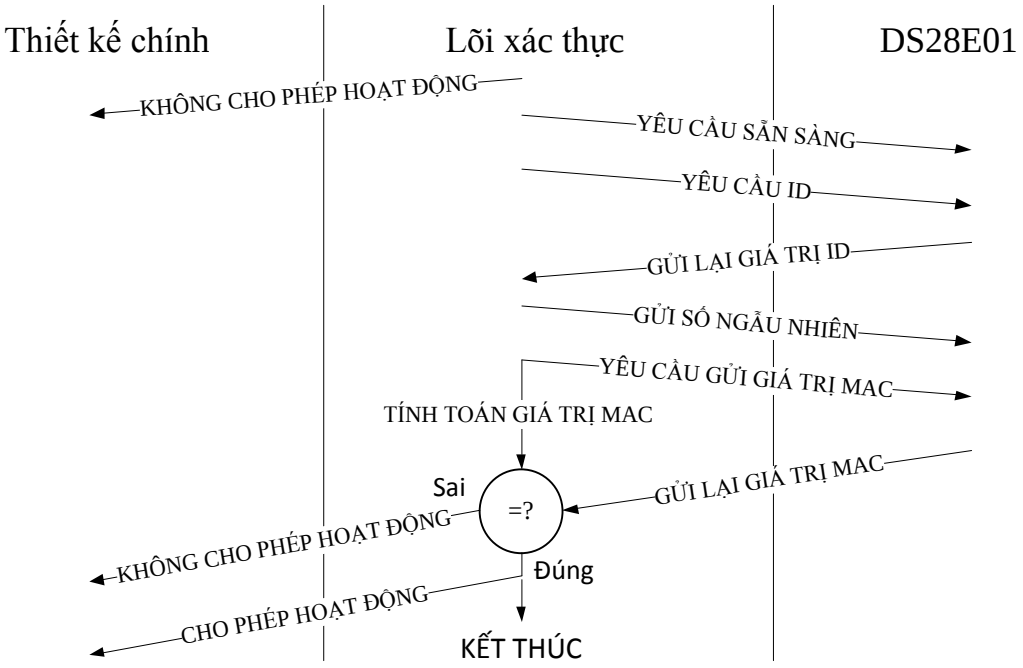


Hình 2.1: Thiết kế lõi xác thực và ứng dụng.

Toàn bộ hệ thống gồm có hai thành phần chính là thiết kế chính và lõi xác thực. Nội dung của luận văn là tập trung vào thiết kế lõi xác thực, nên tôi không xây dựng thiết kế chính lớn. Luận văn chỉ thực hiện phần thiết kế chính là một bộ đếm 8-bit. Bộ đếm 8-bit này chỉ hoạt động khi mà việc xác thực hoàn tất và xác thực đúng là bo mạch hợp lệ.

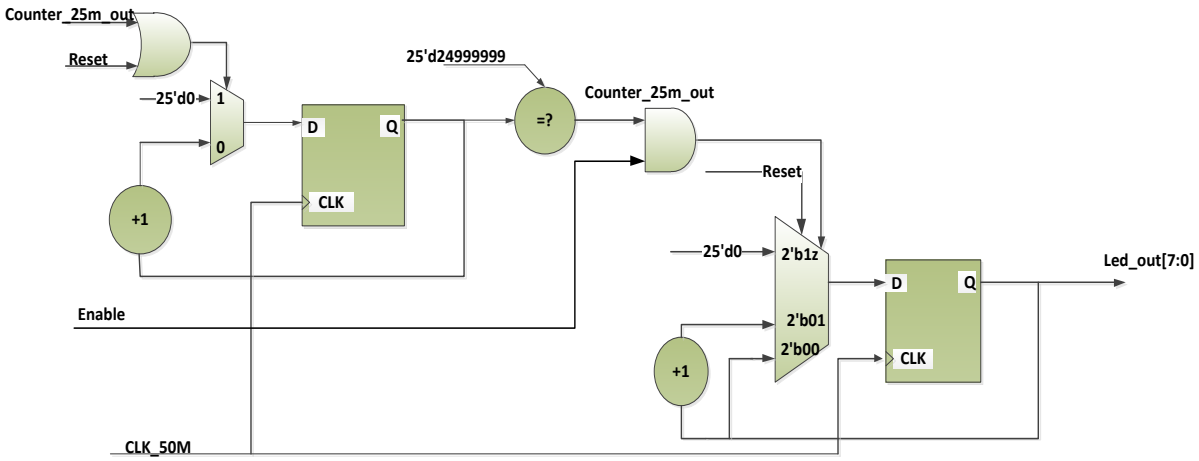
Thiết kế lõi xác thực gồm 3 thành phần chính là bộ xử lý Picoblaze và bộ nhớ chương trình, bộ tạo số ngẫu nhiên (Random Number Generator) và bộ nhớ khoá bí mật (Secret). Đầu tiên, bộ nhớ secret chứa khoá bí mật tạo thành bản tin để tính toán MAC. 64 bit bí mật được dùng để lập trình trước vào IC xác thực. Thứ hai, bộ tạo số ngẫu nhiên nhằm tạo ra dữ liệu thay đổi cho mỗi lần tiến hành xác thực. Bộ tạo số ngẫu nhiên xây dựng với nguồn ngẫu nhiên là jitter của vòng dao động bên trong FPGA. Trong luận văn này, bộ tạo số ngẫu nhiên được tiến hành thiết kế mô phỏng theo nguyên lý được đề xuất bởi Sunnar, được trình bày ở phần sau (mục 2.4). Tuy nhiên, thiết kế của bộ tạo ngẫu nhiên trong luận văn này chưa kiểm tra được chất lượng của bộ tạo số ngẫu nhiên này. Cuối cùng, bộ xử lý Picoblaze là trung tâm xử lý chính của lõi, nó thực hiện việc tính giá trị MAC phục vụ cho xác thực, điều khiển giao tiếp vào ra với IC xác thực DS28E01, và thông báo cho thiết kế chính biết kết xác thực.

Quy trình tiến hành xác thực:



Hình 2.2 Quá trình hoạt động của lõi xác thực.

2.3 Thiết kế cần bảo vệ Bộ đếm 8-bit

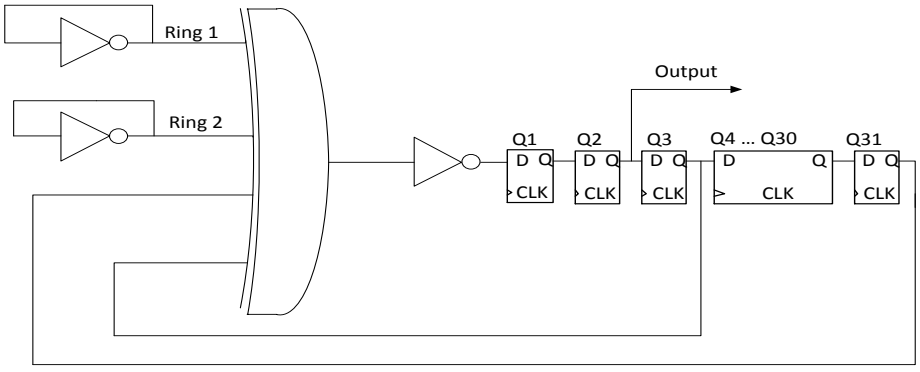


Hình 2.3: Thiết kế chính bộ đếm 8bit.

Với mục tiêu tập trung triển khai giải pháp thực hiện bảo vệ thiết kế chính, học viện lựa chọn thiết kế chính là một bộ đếm thời gian có cấu trúc được mô tả như trên Hình 2.3. Sau khi xác thực thành công, lõi xác thực sẽ cung cấp tín hiệu enable cho phép bộ đếm hoạt động và bộ đếm sẽ tăng giá trị thêm một sau mỗi giây.

2.4 Thiết kế lõi xác thực

2.4.1 Bộ tạo số ngẫu nhiên



Hình 2.4: Thiết kế bộ tạo số ngẫu nhiên.

Bộ tạo số ngẫu nhiên được thiết kế dựa mô phỏng theo đề xuất của Sunnar [3], gồm hai thành phần chính là các vòng dao động và mạch sau xử lý. Đầu tiên, hai vòng dao động được thiết kế dựa trên một cổng NOT cho phép tạo dao động tần số cao >300 MHz. Phần thứ hai là mạch LFRS dựa trên đa thức x^31+x^3+1, mạch này cho phép tăng đặc tính phân bố của chuỗi và tạo tốc độ đầu ra ổn định. Cấu trúc của nó được mô tả như Hình 2.4.

2.4.2 Thuật toán SHA1

Đầu vào: thông điệp với độ dài tối đa 2^64 bits
Đầu ra: giá băm (message digest) có độ dài 160 bits
Giải thuật gồm 5 bước thao tác trên các khối 512 bits

Bước 1. Nhồi dữ liệu

Thông điệp đầu vào sẽ được đệm thêm dữ liệu sao cho độ dài thông điệp L = n*512 + 448 bit. Số bit thêm sẽ nằm trong khoảng [1-512]. Dữ liệu được thêm vào cuối thông điệp gồm 1 bit 1 và theo sau là các bit 0.

Bước 2. Thêm độ dài

Kích thước ban đầu của thông điệp sẽ được biểu diễn dưới dạng số nhị phân 64 bit và thêm vào cuối chuỗi nhị phân ta thu được từ bước trên. Lúc này ta có một thông điệp có độ dài là L'= (n+1)*512 bit.

Bước 3. Khởi tạo bộ đệm MD

Hai bộ đệm 160-bit (sử dụng 5 thanh ghi 32 bit) được dùng lưu trữ các giá trị băm trung gian và kết quả. Năm thanh ghi của bộ đệm đầu tiên được đánh đặt tên là A, B, C, D, E và tương tự cho bộ đệm thứ 2 là H0, H1, H2, H3, H4.

Bước 4. Xử lý các khối dữ liệu 512 bit

Giải thuật thực hiện tất cả 80 bước, chia thành 4 vòng lặp có cấu trúc như nhau, chỉ khác nhau ở các hàm logic f_t.

Bảng 2-1: Các hàm f được sử dụng trong thuật toán SHA-1

Bước	Hàm	Giá trị
$(0 \le t \le 19)$	$f_t = f(B,C,D)$	(B AND C) OR ((NOT B) AND D)
$(20 \le t \le 39)$	$f_t = f(B,C,D)$	B XOR C XOR D
$(40 \le t \le 59)$	$f_t = f(B,C,D)$	(B AND C) OR (B AND D) OR (C AND D)
$(60 \le t \le 79)$	$f_t = f(B,C,D)$	B XOR C XOR D

Mỗi vòng có đầu vào gồm khối 512-bit hiện thời và một bộ đệm 160-bit. Chia khối dữ liệu thành 16 nhóm (mỗi nhóm gồm 32 bit) và đặt theo thứ tự là: W0, W1 ...W15. Mở rộng từ 16 nhóm 32 bit lên 80 nhóm 32 bit bằng vòng lặp:

Từ vòng t = 16 đến 79 thực hiện

$$W_t = (W_{t-3} \text{ XOR } W_{t-8} \text{XOR } W_{t-14} \text{ XOR } W_{t-16}) \ll 1 \qquad (2.1)$$

Gán A = H0, B = H1, C = H2, D = H3, E = H4.

Mỗi vòng lập sử dụng theo công thức chung với một hằng số Kt $(0 \le t \le 79)$ như sau:

Từ vòng t = 0 tới 79 thực hiện

$$TEMP = (A \ll 5) + f(B, C, D) + E + W_t + K_t \quad (2.2)$$

E = D; D = C; C = (B<<30); B = A; A = TEMP;

Với:

Kt = 5A827999 ($0 \leq t \leq 19$)

Kt = 6ED9EBA1 ($20 \leq t \leq 39$)

Kt = 8F1BBCDC ($40 \leq t \leq 59$)

Kt = CA62C1D6 ($60 \leq t \leq 79$)

Đầu ra của 4 vòng (bước 80) được cộng với giá trị của bộ đệm để tạo ra 1 chuỗi kết quả dài 160 bit.

H0 = H0 + A

H1 = H1 + B

H2 = H2 + C

H3 = H3 + D

H4 = H4 + E.

Bước 5. Xuất kết quả

Sau khi thao tác trên toàn bộ N khối dữ liệu (blocks). Kết quả là chuỗi băm 160-bit có dạng.

H={H0 H1 H2 H3 H4}

2.4.3 Cấu trúc bản tin sử dụng để xác thực.

Bảng 2-2: Khối dữ liệu được sử dụng để tính giá trị MAC

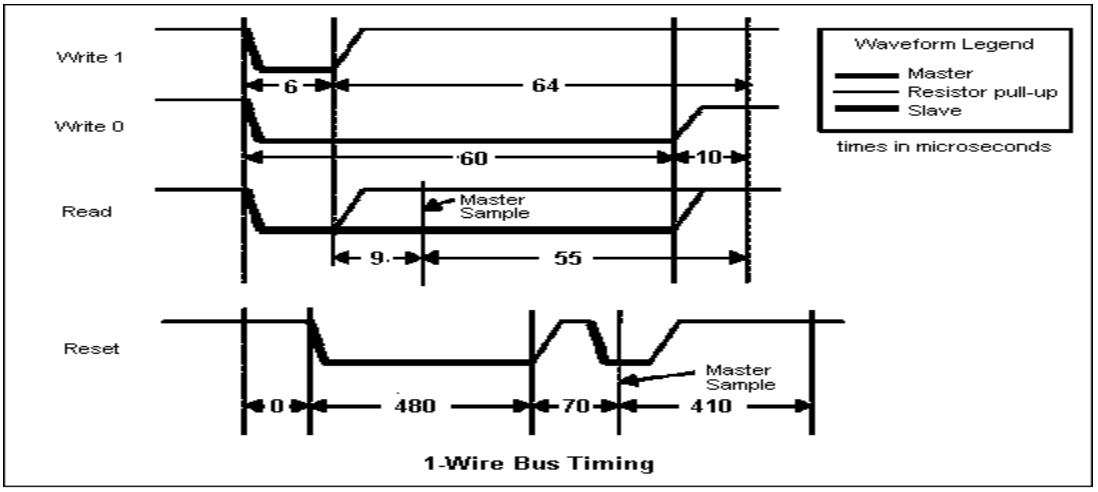
M0	secret1	secret1	secret2	secret3
M1	page_data0	page_data1	page_data2	page_data3
M2	page_data4	page_data5	page_data6	page_data7
M3	page_data8	page_data9	page_data10	page_data11
M4	page_data12	page_data13	page_data14	page_data15
M5	page_data16	page_data17	page_data18	page_data19
M6	page_data20	page_data21	page_data22	page_data23
M7	page_data24	page_data25	page_data26	page_data27

M8	page_data28	page_data29	page_data30	page_data31
M9	scratchpad2	scratchpad3	0xFF	0xFF
M10	0x40	0x33	serial_num0	serial_num1
M11	serial_num2	serial_num3	serial_num4	serial_num5
M12	secret4	secret5	secret6	secret7
M13	scratchpad4	scratchpad5	scratchpad6	0x80
M14	0x00	0x00	0x00	0x00
M15	0x00	0x00	0x01	0xB8

Để tiến hành tính toán được giá trị MAC phục vụ cho xác thực, các dữ liệu cần thiết được ghép với nhau thành một khối dữ liệu 440 bit, sau đó được đệm thêm một byte để đủ 448 bit, thêm 64 bit kích thước khối để tạo thành khối 512 bit. Dữ liệu được tổ chức như trong bảng 2.2.

2.4.4 Giao thức 1-wire

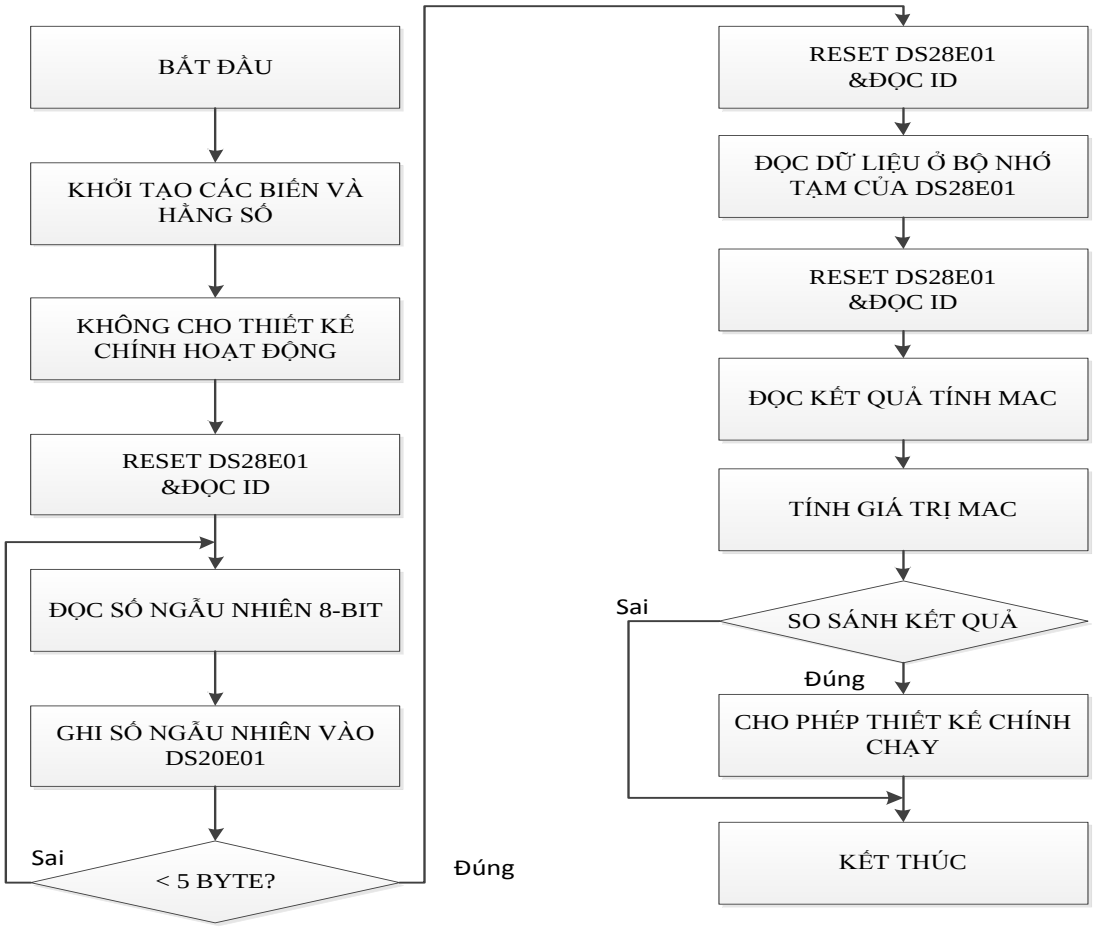
1-Wire là chuẩn bus giao tiếp với thiết bị ngoại vi được thiết kế bởi Dallas Semiconductor Corp. Chuẩn này cung cấp việc truyền dữ liệu tốc độ thấp và nguồn nuôi qua cùng một chân tín hiệu đơn. Thiết bị chủ sẽ thực hiện khởi tạo và điều khiển quá trình trao đổi thông tin với một hoặc nhiều thiết bị và giao thức điều khiển được mô tả như trong Hình 2.5.



Hình 2.5: Giản đồ sóng của giao thức 1-Wire
(https://mysite.du.edu/~etuttle/electron/elect74.htm).

2.4.5 Lưu đồ thuật toán chương trình chính.

Bộ vi xử lý mềm Picoblaze là trung tâm của lõi xác thực, thực hiện điều khiển trao đổi thông tin với DS28E01 qua bus 1-Wire, tính toán giá trị MAC để thực hiện xác thực. Chương trình để Picoblaze được lập trình theo giải thuật có lưu đồ được miêu tả trong Hình 2.6.



Hình 2.6: Lưu đồ thuật toán luồng điều khiển chính.

2.5 Kết quả

Thiết kế lõi xác thực dựa trên picoblaze được tổng hợp trên công nghệ FPGA Spartan 6 (XC6SLX9) bằng bộ công cụ thiết kế ISE của hãng Xilinx. Tài nguyên Flip-Flop và LUT phần cứng của chip FPGA được sử dụng cho thực hiện lõi xác thực được chỉ ra trong Bảng 2-3, với tỷ lệ sử dụng tương ứng là 1% và 3%. Với lượng tài nguyên sử dụng thấp như vậy, nó là khả thi trong các thiết kế sử dụng vi mạch FPGA có ít tài nguyên. Sau khi tổng hợp, thiết kế được ánh xạ, triển khai và nạp vào bo mạch thử nghiệm và thiết kế đã chạy ổn định. Khi mô-đun xác thực được gắn trên bo mạch, sau khi tiến hành xác thực, bộ đếm đã

tiến hành đếm và hiển thị kết quả trên dãy đèn led. Khi mô-đun được tháo ra hay khoá trong thiết kế bị thay đổi, bộ đếm đã không hoạt động.

Bảng 2-3 Thông số tài nguyên sử dụng sau tổng hợp

Loại tài nguyên	**Tài nguyên sử dụng**	**Tài nguyên có sẵn**	**Tỷ lệ sử dụng**
Flip-Flop	121	11440	1%
LUT	199	5720	3%

2.6 Kết luận

Trong chương này, luận văn trình bày các chi tiết thiết kế lõi xác thực thiết kế. Thiết kế đã đạt một số yêu cầu như: giá thành rẻ và lõi xác thực sử dụng rất ít tài nguyên của linh kiện. Tuy nhiên, thiết kế này vẫn còn một số hạn chế như:

- Thiết kế không có khả năng chống lại tấn công phân tích ngược do không có cơ chế mã hoá luồng dữ liệu cấu hình.
- Thiết kế đơn giản nên không có khả năng phát hiện được mã nguồn chương trình đã bị thay đổi hay chưa.
- Bộ tạo số ngẫu nhiên chưa được kiểm tra đánh giá về độ ngẫu nhiên.

Chương 3 GIẢI PHÁP MÃ HÓA THIẾT KẾ FPGA

3.1 Giải pháp thực hiện

Toàn bộ hệ thống thiết kế được chia làm hai phần: là thiết kế cần bảo vệ và lõi giải mã. Như đã nêu ở chương trước, thiết kế chính cần bảo vệ được sử dụng chủ yếu phục vụ mục tiêu thử nghiệm cho giải pháp bảo vệ, nên tác giả chọn một bộ cộng để làm thiết kế chính giả định. Dữ liệu cấu hình thiết kế chính sẽ được thực hiện mã hóa bởi thuật toán GOST 28147-89. Thiết kế cần bảo vệ sẽ được cấu hình sau khi quá trình giải mã thành công.

Bộ giải mã thực hiện nhiệm vụ giải mã thiết kế thiết kế nhận được. Sau đó, nó sẽ cấu hình thiết bị FPGA và phần tài nguyên đã định trước trên thiết bị. Thiết kế lõi giải mã được xây dựng dựa trên bộ vi xử lý Microblaze và lõi thuật toán GOST 28147-89.

Quy trình thực hiện:

Bước 1: Xây dựng thiết kế lập trình, tổng hợp thiết kế, triển khai thiết kế trên FPGA và tạo tệp cấu hình của thiết kế cần bảo vệ.

Bước 2: Sử dụng phần mềm mã hóa tệp cấu hình thiết kế cần bảo vệ, và lưu vào thiết bị nhớ flash, thẻ sd hay NVRAM.

Bước 3: Khởi tạo thiết bị với lõi giải mã và xác thực thiết kế.

Bước 4: Tải thiết kế đã được mã hóa, giải mã và xác thực tệp cấu hình. Sau đó tiến hành cấu hình FPGA nếu vượt qua được quá trình xác thực.

3.2 Thuật toán GOST 28147-89

Thuật toán tựa GOST là thuật toán mã hóa dựa trên chuẩn thuật toán GOST 28147-89 (nay được chuyển thành một chuẩn mã hóa trong chuẩn GOST 34.12-2015 [14]). Thuật toán được thiết kế dựa trên cấu trúc Feistel với kích thước khối 64 bit, độ dài khóa là 256 bits và có số vòng là 32 vòng.

3.2.1 Ký hiệu

Trong chuẩn này các ký hiệu sau được sử dụng

V^*	Tập hợp tất cả các chuỗi nhị phân độ dài hữu hạn, gồm cả chuỗi trống
V_s	Tập tất cả các chuỗi nhị phân độ dài s, trong đó s là số nguyên không âm; việc đánh số các chuỗi con và thành phần của chuỗi được thực hiện từ phải qua trái bắt đầu từ 0.

$U \times W$	Tích thẳng (đề-các) của tập U và tập W;
$\|A\|$	Số thành phần (độ dài) của chuỗi $A \in V^*$ (nếu A – chuỗi trống, thì $\|A\| = 0$);
$A\|\|B$	Nối của hai chuỗi $A, B \in V^*$, tức là một chuỗi từ $V_{\|A\| + \|B\|}$, trong đó chuỗi con cùng các thành phần có chỉ số lớn từ $V_{\|A\|}$ trùng với chuỗi A và chuỗi con cùng với các thành phần có chỉ số nhỏ từ $V_{\|B\|}$ trùng với với chuỗi B;
$A <<<_{11}$	Dịch vòng của chuỗi $A \in V_{32}$ đi 11 thành phần về phía các thành phần có chỉ số lớn;
$\oplus$	Phép cộng modulo 2 theo từng thành phần của hai chuỗi nhị phân có độ dài như nhau;
$\mathbf{Z}_{2^s}$	Vành của các đồng dư theo modulo 2^s;
(	Phép toán cộng trong vành $\mathbf{Z}_{2^{32}}$;
□	Trường hữu hạn $GF(2)[x]/p(x)$, trong đó $p(x) = x^8 + x^7 + x^6 + x + 1 \in GF(2)[x]$; các phần tử của trường □ được biểu diễn bằng các số nguyên, trong đó phần tử $z_0 + z_1 \cdot \theta + \ldots + z_7 \cdot \theta^7 \in$ □ tương ứng với số $z_0 + 2 \cdot z_1 + \ldots + 2^7 \cdot z_7$, trong đó $z_i \in \{0, 1\}$, $i = 0, 1, \ldots, 7$ và θ ký hiệu lớp đồng dư theo modulo $p(x)$ mà chứa x;
$\mathrm{Vec}_s: \mathbf{Z}_{2^s} \to V_s$	Ánh xạ song ánh đặt tương ứng một phần tử của vành $\mathbf{Z}_{2^s}$ với biểu diễn nhị phân của nó, tức là đối với phần tử bất kỳ $z \in \mathbf{Z}_{2^s}$, mà được biểu diễn ở dạng $z = z_0 + 2 \cdot z_1 + \ldots + 2^{s-1} \cdot z_{s-1}$, trong đó $z_i \in \{0, 1\}$, $i = 0, 1, .., s - 1$, đẳng thức sau được thực hiện $\mathrm{Vec}_s(z) = z_{s-1}\|\ldots.\|z_1\|z_0$;
$\mathrm{Int}_s: V_s \to \mathbf{Z}_{2^s}$	Ánh xạ ngược với ánh xạ Vec_s, tức là $\mathrm{Int}_s = \mathrm{Vec}_s^{-1}$;
$\Delta: V_8 \to$ □	Ánh xạ song ánh, đặt tương ứng một chuỗi nhị phân từ V_8 với một phần tử của trường □ như sau: chuỗi $z_7\|\|\ldots\|\|z_1\|\|z_0$, $z_i \in \{0, 1\}$, $i = 0, 1, .., 7$, tương ứng với phần tử $z_0 + z_1 \cdot \theta + \ldots + z_7 \cdot \theta^7 \in$ □;
$\nabla:$ □ $\to V_8$	Ánh xạ ngược với ánh xạ Δ, tức là $\nabla = \Delta^{-1}$;
$\Phi\Psi$	Tổ hợp của hai ánh xạ, trong đó ánh xạ Ψ được thực hiện trước;
Φ^s	Tổ hợp của ánh xạ Φ^{s-1} với Φ, còn $\Phi^1 = \Phi$.

3.2.2 **Phép biến đổi Sbox**

Các phép thế $\pi_i = \mathrm{Vec}_4\pi_i\text{'}\mathrm{Int}_4: V_4 \to V_4$, trong đó $\pi_i\text{'}: \mathbf{Z}_{2^4} \to \mathbf{Z}_{2^4}$, $i = 0, 1, \ldots, 7$ được sử dụng là các biến đổi song ánh phi tuyến. Các giá trị của các phép thế π_i' được mô tả dưới đây ở dạng các mảng $\pi_i\text{'} = (\pi_i\text{'}(0), \pi_i\text{'}(1), \ldots, \pi_i\text{'}(15))$, $i = 0, 1, .., 7$:

Bộ S-hộp của thuật toán tựa GOST có thể được sử dụng khác nhau. Nói cách khác, các sbox không được chuẩn hóa và coi là một thành phần bí mật góp phần bảo mật dữ liệu. Sau đây là một sbox ví dụ có trong chuẩn GOST 34.12-2015:

π_0' = (12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1);

π_1' = (6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15);

π_2' = (11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0);

π_3' = (12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11);

π_4' = (7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12);

π_5' = (5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0);

π_6' = (8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7);

π_7' = (1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2).

3.2.3 **Các biến đổi**

Khi thực hiện các thuật toán mã hóa và giải mã, các biến đổi sau được sử dụng:

$t: V_{32} \to V_{32}$	$t(a) = t(a_7\|\|\ldots\|\|a_0) = \pi_7(a_7)\|\|\ldots\|\|\pi_0(a_0)$, trong đó $a = a_7\|\|\ldots\|\|a_0 \in V_{32}$, $a_i \in V_4$, $i = 0, 1, .., 7$;	(3.1)
$g[k]: V_{32} \to V_{32}$	$g[k](a) = (t(\mathrm{Vec}_{32}(\mathrm{Int}_{32}(a) \boxplus \mathrm{Int}_{32}(k))) \lll 11$, trong đó $k, a \in V_{32}$;	(3.2)
$G[k]: V_{32}\times V_{32} \to V_{32}\times V_{32}$	$G[k](a_1, a_0) = (a_0, g[k](a_0) \oplus a_1)$, trong đó $k, a_0, a_1 \in V_{32}$;	(3.3)
$G^*[k]: V_{32}\times V_{32} \to V_{64}$	$G^*[k](a_1, a_0) = (g[k](a_0) \oplus a_1)\|\|a_0$, trong đó $k, a_0, a_1 \in V_{32}$	(3.4)

3.2.4 **Thuật toán lược đồ khóa**

Các khóa vòng $K_i \in V_{32}$, $i = 1, 2, \ldots, 32$, được tạo ra trên cơ sở của khóa $K = k_{255}||\ldots||k_0 \in V_{256}$, $k_i \in V_1$, $i = 0, 1, \ldots, 255$ và được xác định bằng các đẳng thức:

$$K_{25} = K_{17} = K_9 = K_8 = k_{255}||\ldots||k_{224};$$
$$K_{26} = K_{18} = K_{10} = K_7 = k_{223}|| \ldots||k_{192};$$
$$K_{27} = K_{19} = K_{11} = K_6 = k_{191}||\ldots||k_{160};$$
$$K_{28} = K_{20} = K_{12} = K_5 = k_{159}|| \ldots||k_{128};$$
$$K_{29} = K_{21} = K_{13} = K_4 = k_{127}||\ldots||k_{96};$$
$$K_{30} = K_{22} = K_{14} = K_3 = k_{95}|| \ldots||k_{64};$$
$$K_{31} = K_{23} = K_{15} = K_2 = k_{63}||\ldots||k_{32};$$
$$K_{32} = K_{24} = K_{16} = K_1 = k_{31}|| \ldots||k_0;$$

3.2.5 **Thuật toán mã hóa cơ bản**

a. ***Thuật toán mã hóa***

Thuật toán mã hóa phụ thuộc vào các giá trị của các khóa vòng $K_i \in V_{32}$, $i = 1, 2, ..,32$, thực hiện phép thế $E_{K_1,\ldots,K_{32}}$, nó được cho trên tập V_{64} tương ứng với đẳng thức

$$E_{K_1,\ldots,K_{32}}(a) = G^*[K_{32}]G[K_{31}].... G[K_2]G[K_1](a_1, a_0), \quad (3.5)$$

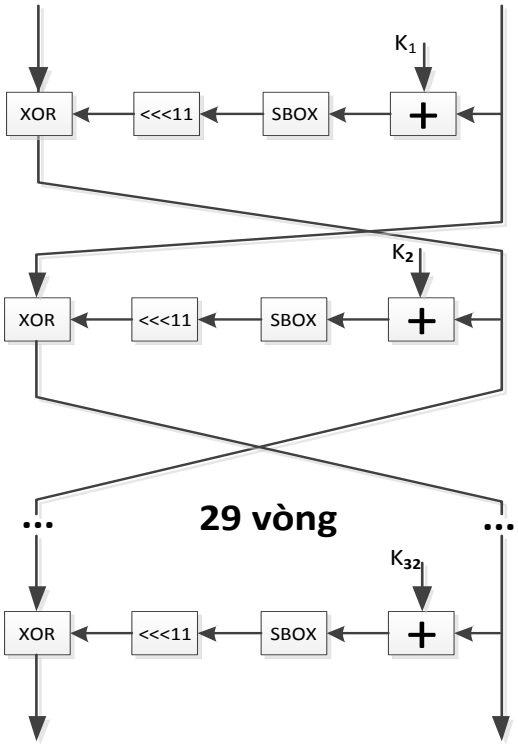
trong đó $a = a_1||a_0 \in V_{64}$, $a_0, a_1 \in V_{32}$.

b. ***Thuật toán giải mã***

Thuật toán giải mã mật phụ thuộc vào các giá trị của các khóa vòng $K_i \in V_{32}$, $i = 1,2,..,32$, thực hiện phép thế $D_{K_1,\ldots,K_{32}}$, nó được cho trên tập V_{64} tương ứng với đẳng thức

$$D_{K_1,\ldots,K_{32}}(a) = G^*[K_1]G[K_2].... G[K_{31}]G[K_{32}](a_1, a_0), \quad (3.6)$$

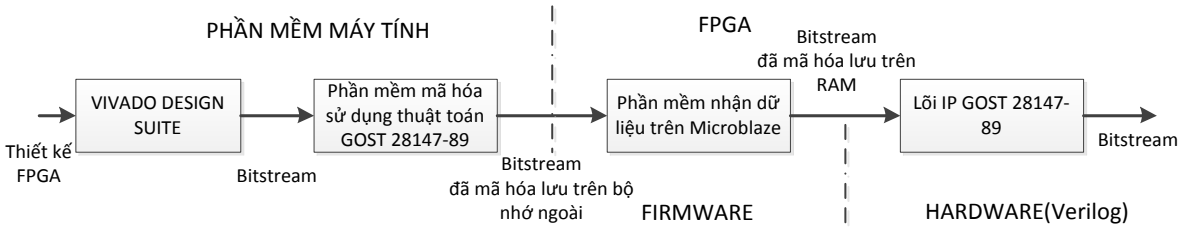
trong đó $a = a_1||a_0 \in V_{64}$, $a_0, a_1 \in V_{32}$.



Hình 3.1 Lưu đồ xử lý dữ liệu

3.3 Xây dựng phần mềm mã hóa

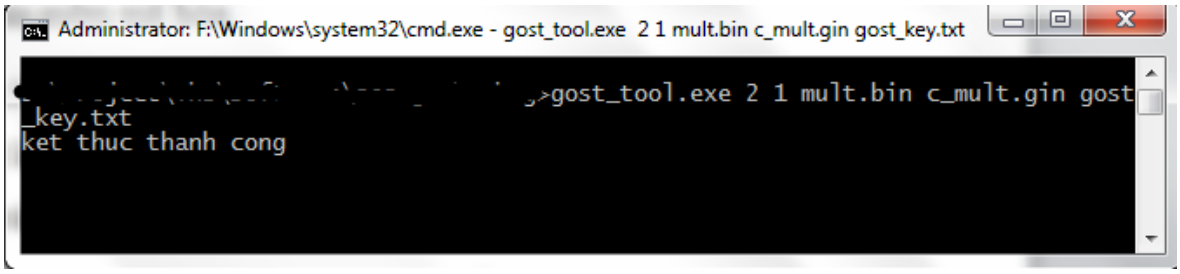
Giải pháp này được thực hiện kết hợp giữa phần mềm và thiết kế phần cứng. Trong đó bitstream của thiết kế FPGA sẽ được mã hóa bằng phần mềm trên máy tính và phần giải mã bitstream sẽ được thực hiện trên FPGA. Quá trình phân chia thực hiện giải pháp sẽ được mô tả chi tiết trong Hình 3.2.



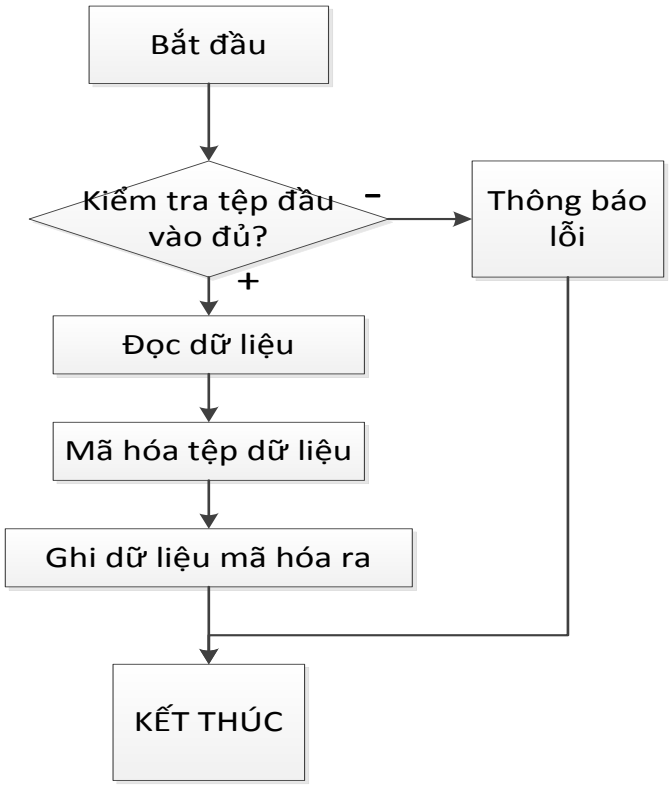
Hình 3.2 Quy trình thực hiện giải pháp

Trong luận văn này, học viên lựa chọn thuật toán GOST 28147-89 để thực hiện bảo vệ thực tệp cấu hình thiết kế FPGA. Phần mềm được sử dụng dựa cần tệp dữ liệu đầu vào là: tệp khóa và tệp tin cấu hình. Phần mềm sẽ tiến hành kiểm tra các tệp tin tham số đầu vào nếu không đầy đủ, chương trình sẽ thông báo lỗi và kết thúc. Khi các tham số đầu vào đầy đủ, chương trình phần mềm sẽ tiến hành mã hóa tệp tin cấu hình theo thuật toán GOST 28147-89 như đã trình bày ở Mục 3.2. Khi quá trình mã hóa kết thúc, tệp tin cấu hình sẽ được lưu lại vào ổ cứng cùng thư mục với tệp tin đầu vào. Tệp cấu hình sau mã hóa sẽ có hai thành

phần đó là kích thước thật của tệp cấu hình và dữ liệu được mã hóa. Lưu đồ chương trình của phần mềm được mô tả như trong Hình 3.4.



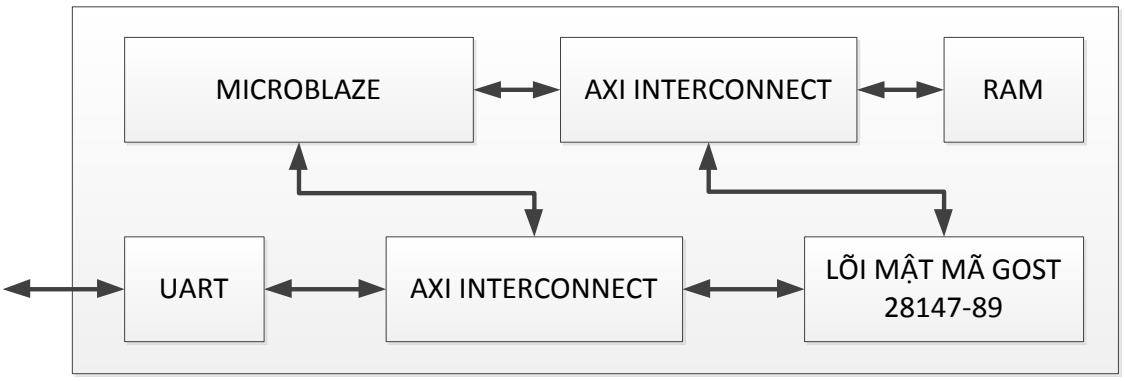
Hình 3.3 Phần mềm mã hóa tệp cấu hình FPGA.



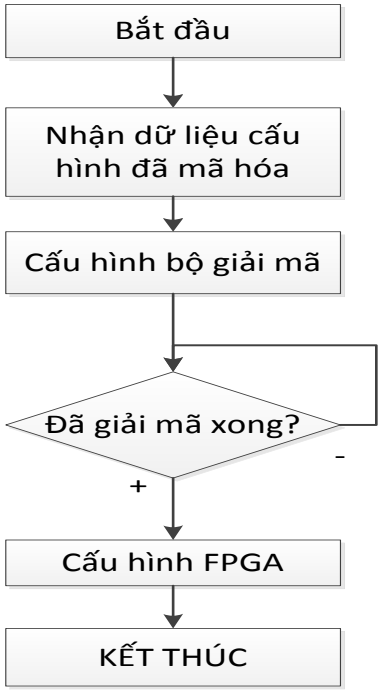
Hình 3.4 Giải thuật thực hiện phần mềm.

3.4 Thiết kế bộ giải mã

Bộ giải mã được xây dựng dựa trên bộ vi xử lý Microblaze và lõi thuật toán GOST28147-89, cấu trúc của bộ giải mã được mô tả trong Hình 3.4. Phần mềm chạy trên Microblaze sẽ thực hiện thu nhận dữ liệu cấu hình, nạp tham số khóa cho lõi thuật toán GOST28147-89 và địa chỉ vùng nhớ lưu dữ liệu cấu hình thiết kế. Lõi thuật toán GOST28147-89 thực hiện giải mã dữ liệu cấu hình với các tham số được nạp từ Microblaze. Sau khi giải mã xong, tệp cấu hình sẽ được Microblaze sử dụng để cấu hình vùng FPGA đã định trước. Giải thuật chương trình thực hiện trên Microblaze được mô tả trong Hình 3.6.



Hình 3.5 Cấu trúc bộ giải mã tệp cấu hình.



Hình 3.6 Giải thuật thực hiện trên Microblaze.

3.5 Thiết kế lõi mật mã GOST 28147-89

Lõi mật mã GOST 28147-89 được mô tả trong Hình 3.7 gồm các thành phần:

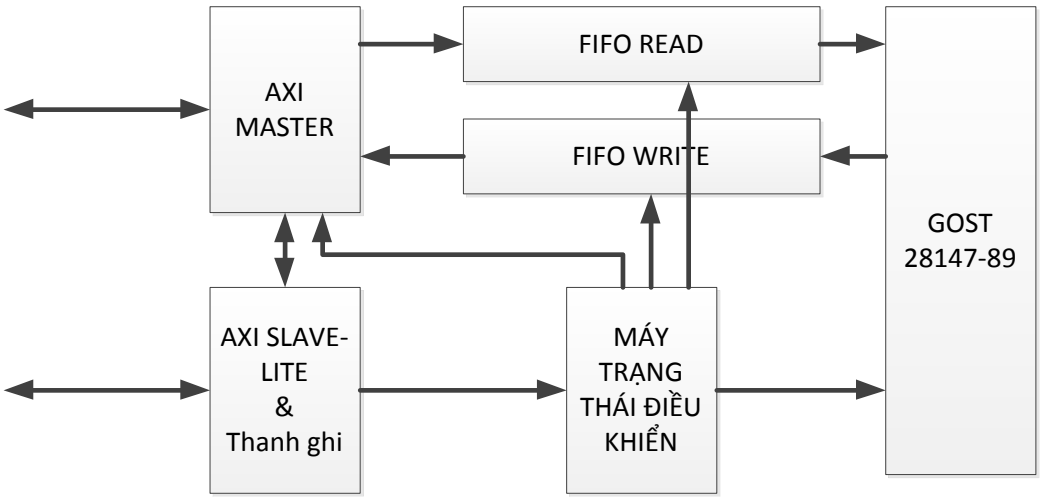
Máy trạng thái điều khiển bus AXI Master: thực hiện đọc và ghi dữ liệu từ vùng nhớ đã được cấu hình từ các thanh ghi cấu hình. Thiết kế cho phép đọc dữ liệu theo chùm (burst) hỗ trợ đến 16 từ 32 bits trong một lần đọc/ghi dữ liệu.

Các thanh ghi cấu hình chứa các thông số khóa giải mã, địa chỉ lưu file cấu hình, kích thước tệp cấu hình của lõi mật mã và trạng thái của lõi. Các thông số này được đọc ghi thông qua Bus AXI Slave-lite.

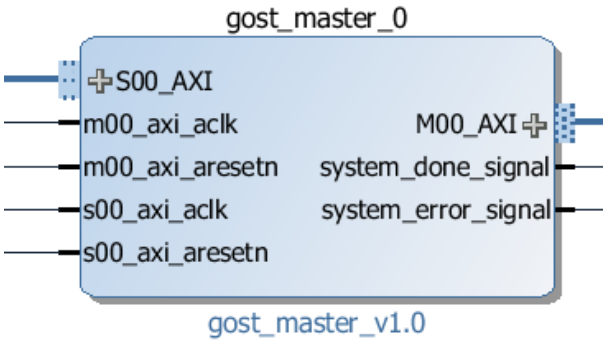
Bộ đệm dữ liệu vào và ra: là fifo chứa dữ liệu được đọc vào qua bus AXI Master và dữ liệu đã được giải mã trước khi trả lại bộ nhớ RAM.

Mô-đun thực hiện thuật toán GOST 28147-89: thực hiện giải mã dữ liệu. Mô đun được thiết kế theo cấu trúc dạng vòng lặp, tiêu tốn 32 chu kỳ xung nhịp đồng hồ để thực hiện mã hóa/giải mã một khối 64 bit.

Máy trạng thái điều khiển hoạt động: điều khiển hoạt động của toàn bộ lõi mật mã gồm có yêu cầu đọc dữ liệu, ghi dữ liệu từ bộ nhớ, điều khiển quá trình giải mã dữ liệu của toàn bộ dữ liệu.



Hình 3.7 Cấu trúc lõi mật mã GOST 28147-89.



Hình 3.8 Đóng gói IP của lõi mật mã GOST 28147-89 sử dụng Xilinx Vivado.

3.6 Mô phỏng và thử nghiệm

a. Mô phỏng thiết kế bộ lõi mật mã.

Mô hình mô phỏng gồm có 5 thành phần được kết nối như trong Hình 3.9. Thành phần tạo nên mô hình mô phỏng gồm có:

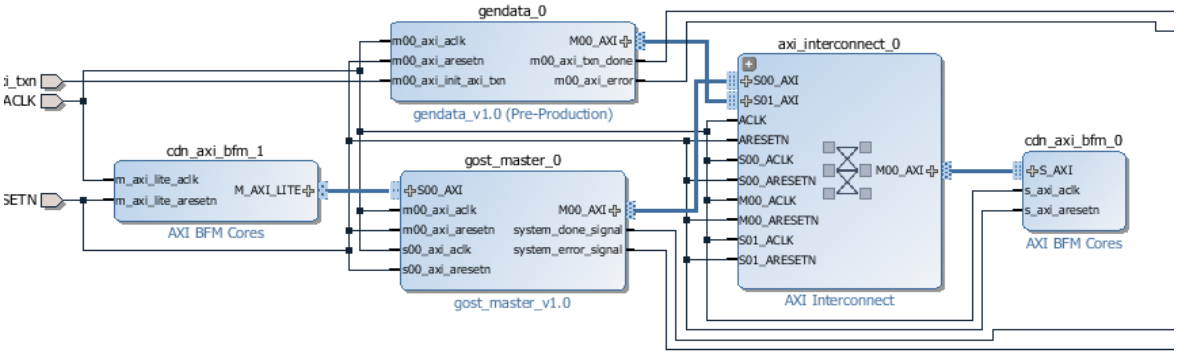
- Bộ mô phỏng chức năng Master bus AXI-lite (cdn_axi_bfm_1), được sử dụng để ghi dữ liệu điều khiển lõi mật mã GOST.

- Bộ mô phỏng bộ nhớ Slave AXI-full (cdn_axi_bfm_0), sử dụng để lưu trữ dữ liệu mô phỏng.

- Bộ tạo dữ liệu (gendata_0) sử dụng để tạo dữ liệu phục vụ mô phỏng.

- Bộ kết nối bus AXI (axi_interconnect_0) kết nối các thành phần.

- Thiết kế chính (gost_master_0) thiết kế cần mô phỏng.



Hình 3.9 Mô phỏng thử nghiệm lõi mật mã GOST 28147-89.

b. Thử nghiệm thiết kế

Để thử nghiệm thiết kế, học viên đã xây dựng một lõi IP thực hiện phép cộng. Thiết kế sau khi được giải mã sẽ được cấu hình lên phần tài nguyên FPGA đã được định trước. Bộ cộng sẽ thực hiện phép cộng toán hạng a và toán hạng b và trả lại kết quả qua c và d. Quá trình thực nghiệm giải pháp sẽ được thực hiện qua bước sau.

Bước 1: Thiết kế bộ cộng IP sẽ được tổng hợp, ánh xạ và tạo bitstream bằng phần mềm Vivado của Xilinx.

Bước 2: Bitstream sẽ được mã hóa bằng phần mềm đã được mô tả ở Mục 3.3.

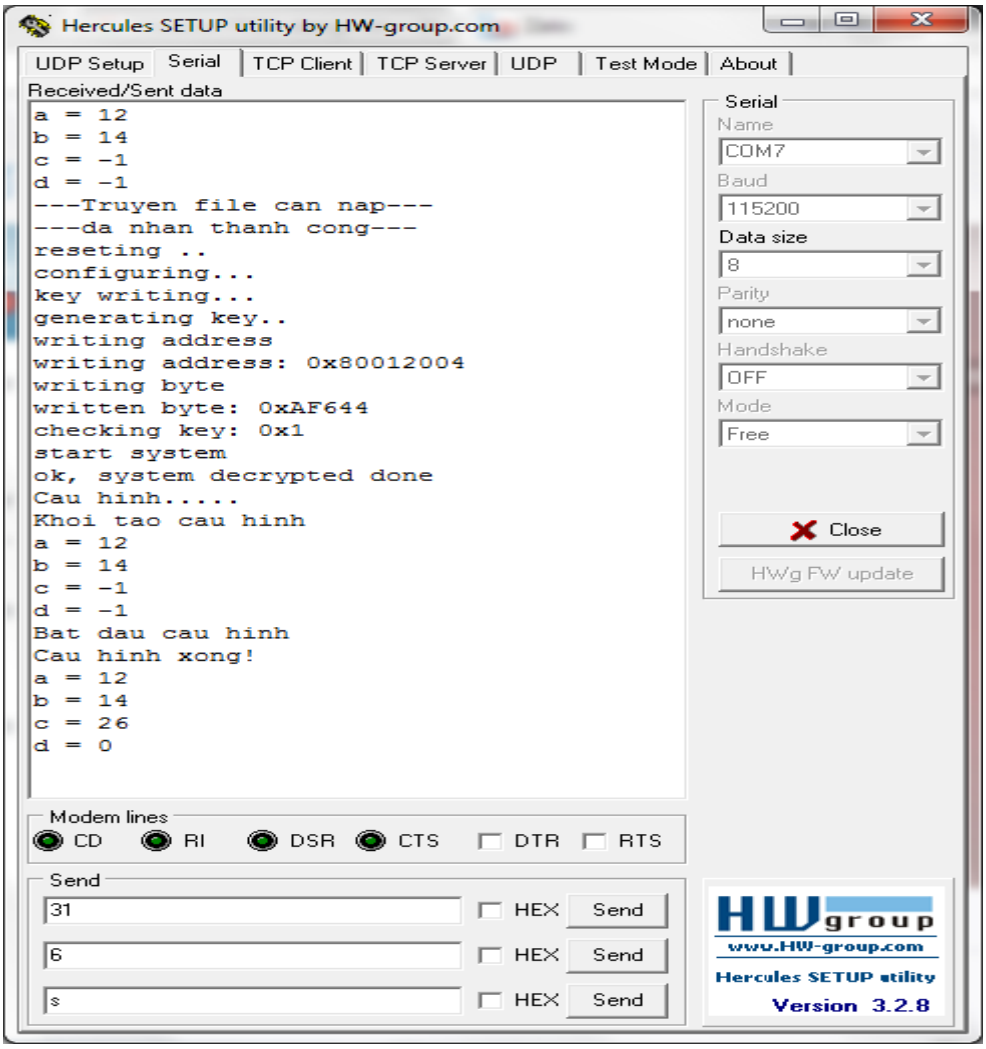
Bước 3: Bo mạch thử nghiệm với lõi giải mã (Mục 3.4) được cấu hình trước. Bộ giải mã đợi dữ liệu cấu hình từ máy tính.

Bước 4: Máy tính gửi dữ liệu xuống cho bo mạch qua cổng UART.

Bước 5: Sau khi nhận hết dữ liệu, bộ giải mã tiến hành giải mã thiết kế và nạp vào vùng tài nguyên FPGA đã định trước.

Hình 3.10 thể hiện quá trình hoạt động của bộ giải mã. Trước khi tiến hành cấu hình, kết quả thực hiện phép cộng là không chính xác do thiết kế chưa

được nạp. Sau khi cấu hình, kết quả thể hiện trên biến c và d thể hiện kết quả đúng như mong đợi.



Hình 3.10 Thử nghiệm sau khi nạp thiết kế bộ cộng.

3.7 Kết luận

Trong chương này luận văn đã đề xuất giải pháp sử dụng thuật toán mật mã GOST 28147-89 để thực hiện bảo vệ thiết kế FPGA thay cho giải pháp mặc định của nhà sản xuất FPGA. Lõi mật mã GOST 28147-89 trong chương này đã được thiết kế và thử nghiệm thành công. Lõi có giao tiếp chuẩn AXI – MASTER-FULL cho phép truy cập trực tiếp vào bộ nhớ, từ đó có thể tăng tốc xử lý dữ liệu. Tuy nhiên, lõi mã hóa/giải mã thiết kế theo cấu trúc vòng nên tốc độ chỉ đạt 120 Mbps tại tần số hoạt động 100 MHz.

KẾT LUẬN

Trong thời gian tìm hiểu và nghiên cứu dưới sự giúp đỡ tận tình của thầy hướng dẫn PGS-TS. Trần Xuân Tú, đến nay toàn bộ nội dung của luận văn đã được hoàn thành đáp ứng đầy đủ các yêu cầu đã đặt ra. Qua quá trình tìm hiểu thực hiện đề tài, học viên đã thu được những kết quả chính như sau:

- Nghiên cứu tìm hiểu các phương pháp tấn công lên bitstream nhằm sao chép và sử dụng trái phép thiết kế FPGA và các phương pháp nhằm bảo vệ thiết kế chống lại các tấn công này.
- Thử nghiệm thành công giải pháp bảo vệ thiết kế FPGA bằng phương pháp sử dụng vi mạch xác thực DS28E01, kết quả tổng hợp cho thấy giải pháp có kích thước nhỏ đáp ứng được yêu cầu cho các thiết bị FPGA có ít tài nguyên.
- Đề xuất thực giải pháp thực hiện giải pháp mã hóa bitstream thiết kế FPGA bằng thuật toán mật mã GOST 28147-89. Thiết kế thành công lõi xử lý mật mã GOST 28147-89 và mô hình hóa bằng ngôn ngữ Verilog.

Tuy nhiên, vẫn còn một số hạn chế như:

- Giải pháp bảo vệ sử dụng vi mạch xác thực ngoài không thể chống lại tấn công phân tích ngược và không có khả năng phát hiện mã nguồn chương trình đã bị thay đổi.
- Giải pháp mã hóa thiết kế bitstream không kiểm tra được tính toàn vẹn của bitstream. Tốc độ thực hiện giải mã còn chậm.

Để khắc phục các hạn chế trên và hoàn thiện giải pháp bảo vệ bitstream, học viên nhận thấy cần phải phát triển thêm thiết kế lõi kiểm tra tính toàn vẹn của bitstream của thiết kế FPGA.

TÀI LIỆU THAM KHẢO

[1] Altera Inc (2011), Stratix II Device Handbook.

[2] Altera Inc (2011), Stratix III Device Handbook.

[3] B. Sunar, W. J. Martin, and D. R. Stinson (2007), A provably secure true random number generator with built-in tolerance to active attacks,IEEE Transactions on Computers , vol. 56, no. 1, pp. 109–119.

[4] Bernhard Linke (2009), Application Note 4594: Protect Your FPGA Against Piracy: Cost-Effective Authentication Scheme Protects IP in SRAM-Based FPGA Designs, Maxim Integrated.

[5] Catalin Baetoniu (2010), Application Note 780: FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs, Xilinx.

[6] CJ Clark (2009), Business Consideration for Systems with RAM-based FPGA Configuration, Intellitech Corporation.

[7] D. Kessner(2000), Copy protection for SRAM based FPGA designs, http://web.archive.org/web/20031010002149/.

[8] K. Austin (1995), Data security arrangements for semiconductor programmable devices, United States Patent Office.

[9] Saar Drimer (2007), Authentication of FPGA Bitstreams: Why and How, Springer.

[10] Saar Drimer (2008), Volatile FPGA design security, University of Cambridge.

[11] Maxim Integrated Inc, Datasheet of DS28E01.

[12] Xilinx Inc, KCPSM6 User Guide.

[13] Xilinx Inc. User Guide 071, p123.

[14] GOST R 34.12-2015 Криптографическая защита информации. Блочные шифры. 2015.