

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ

-----o0o-----



LUẬN VĂN TỐT NGHIỆP

**THIẾT KẾ PHẦN CỨNG XỬ LÝ HÀM BẮM
KECCAK CHO GIẢI THUẬT SHA3**

GVHD: TS. Trần Hoàng Linh

SVTH: Trần Công Tiến

MSSV: 1810580

TP. HỒ CHÍ MINH, THÁNG 4 NĂM 2022

-----☆-----

-----☆-----

Số: _____/BKĐT
Khoa: **Điện – Điện tử**
Bộ Môn: **Điện Tử**

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

1. HỌ VÀ TÊN: Trần Công Tiến MSSV: 180580
2. NGÀNH: **ĐIỆN TỬ - VIỄN THÔNG** LỚP : DD18DV08
 1. Đề tài: Thiết kế phần cứng xử lý hàm băm Keccak cho giải thuật SHA3.
 2. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu):
.....
.....
.....
.....
.....
.....
.....
3. Ngày giao nhiệm vụ luận văn:
4. Ngày hoàn thành nhiệm vụ:
5. Họ và tên người hướng dẫn: Phần hướng dẫn
.....
.....

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Tp.HCM, ngày..... tháng..... năm 20
CHỦ NHIỆM BỘ MÔN

NGƯỜI HƯỚNG DẪN CHÍNH

PHẦN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ):.....
Đơn vị:.....
Ngày bảo vệ :
Điểm tổng kết:
Nơi lưu trữ luận văn:

LỜI CẢM ƠN

Trong thời gian làm luận văn và đề cương luận văn, em đã nhận được nhiều sự giúp đỡ đóng góp ý kiến từ thầy cô, gia đình và bạn bè.

Em xin gửi lời cảm ơn chân thành đến TS. Trần Hoàng Linh, trưởng bộ môn Điện Tử - Trường Đại học Bách Khoa thành phố Hồ Chí Minh đã tận tình hướng dẫn, chỉ bảo em trong suốt thời gian làm luận văn và đề cương luận văn, anh Vương Tuấn Hùng – người trực tiếp theo sát em trong quá trình hoàn thành đề tài.

Em cũng xin gửi lời cảm ơn chân thành đến các thầy cô giảng viên trường Đại học Bách Khoa thành phố Hồ Chí Minh đã giảng dạy, truyền đạt những kiến thức từ đại cương cho đến chuyên ngành.

Em cũng xin gửi lời cảm ơn sâu sắc đến gia đình, bạn bè đã luôn ủng hộ và hỗ trợ em rất nhiều trong quá trình theo học tại đại học.

Em xin gửi lời chúc sức khỏe đến thầy cô, gia đình và bạn bè!

Tp. Hồ Chí Minh, ngày 23 tháng 12 năm 2021 .

Sinh viên

MỤC LỤC

1. GIỚI THIỆU	1
1.1 Tổng quan	1
1.2 Tình hình nghiên cứu trong và ngoài nước	1
1.3 Mục tiêu đề tài	1
2. NỘI DUNG LÝ THUYẾT ĐỀ TÀI	2
2.1 Hàm băm và một số ứng dụng của hàm băm	2
2.1.1 Hàm băm	2
2.1.2 Ứng dụng của hàm băm	3
2.2 Sponge Function	4
2.3 Bộ đệm Pad10*1	7
2.4 Hàm hoán vị Keccak	8
2.4.1 Biến đổi θ	11
2.4.2 Biến đổi ρ	12
2.4.3 Biến đổi π	13
2.4.4 Biến đổi χ	14
2.4.5 Biến đổi ι	14
2.4.6 Hàm hoán vị Keccak-p[b, n_r]	16
2.4.7 Hàm hoán vị Keccak-f[b]	16
2.4.8 Hàm Keccak[c](N,d)	16
2.5 Thiết kế SHA-3 bằng thuật toán Keccak	17
3. GIẢI PHÁP THỰC HIỆN	17
3.1 Module Padding	19
3.2 Module Step-Mapping	20
3.3 Module Control	22
4. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG	23

4.1 Yêu cầu thiết kế	23
4.2 Phân tích thiết kế	23
4.3 Sơ đồ khối tổng quát	24
4.4 Sơ đồ khối chi tiết và nhiệm vụ, chức năng từng khối	24
4.4.1 Bộ đệm ngõ vào Buffer_in	24
4.4.2 Bộ VSX	26
4.4.3 Bộ STA	28
4.4.4 Bộ ATS	28
4.4.5 Bộ biến đổi Transformation Round	29
4.4.6 Bộ Trunc	31
4.4.7 Khối điều khiển tín hiệu Control	33
5. THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM	36
5.1 Create_Random_Keccak.py	38
5.2 Hash_Function.py	40
5.3 Compare.py	40
6. KẾT QUẢ THỰC HIỆN	41
6.1 Cách thức đo đạc, thử nghiệm	41
6.2 Số liệu đo đạc	41
7. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	43
8. TÀI LIỆU THAM KHẢO	43

DANH SÁCH HÌNH

Hình 2 - 1 Cấu trúc bọt biển	5
Hình 2 - 2 Giải thuật bọt biển	6
Hình 2 - 3 Hàm absorbing function	6
Hình 2 - 4 Hàm Squeezing function	7
Hình 2 - 5 Mảng trạng thái 3 chiều	9
Hình 2 - 6 Các thành phần của mảng trạng thái	10
Hình 2 - 7 Ký hiệu chiều x, y, z của mảng trạng thái	11
Hình 2 - 8 Biến đổi Theta	11
Hình 2 - 9 Biến đổi Rho	12
Hình 2 - 10 Biến đổi Pi	13
Hình 2 - 11 Biến đổi Chi	14
Hình 3 - 1 Sơ đồ tổng quát hàm băm SHA-3	18
Hình 3 - 2 Sơ đồ khối StepMapping trong thiết kế [1]	21
Hình 3 - 3 Sơ đồ khối tổng quát trong thiết kế [3]	22
Hình 4 - 1 SHA3-core	23
Hình 4 - 2 Sơ đồ khối tổng quát của thiết kế	24
Hình 4 - 3 Sơ đồ khối khối Buffer_in	24
Hình 4 - 4 Sơ đồ khối VSX	26
Hình 4 - 5 Sơ đồ khối chi tiết khối VSX	27
Hình 4 - 6 Sơ đồ khối STA	28
Hình 4 - 7 Sơ đồ khối ATS	29
Hình 4 - 8 Sơ đồ khối Transformation_round	30
Hình 4 - 9 Sơ đồ chi tiết khối Transformation_round	30
Hình 4 - 10 Bảng ROM lưu giữ giá trị round_constant dùng trong Iota	31
Hình 4 - 11 Sơ đồ khối Trunc	32

Hình 4 - 12 Máy trạng thái khối Control	33
Hình 5 - 1 Kết quả khi test chuỗi ngõ vào trên bằng Python	37
Hình 5 - 2 File Create_Random_Keccak.py	38
Hình 5 - 3 Kết quả khi chạy file Create_Random_Keccak.py	40
Hình 5 - 4 File Hash_Function.py	40
Hình 5 - 5 File Compare.py	41
Hình 6 - 1 Kết quả tính toán hàm băm bằng phần cứng và phần mềm	42
Hình 6 - 2 Kết quả khi so sánh hai kết quả	42
Hình 6 - 3 Tần số của hệ thống	42
Hình 6 - 4 Kết quả tổng hợp phần cứng	43

DANH SÁCH BẢNG

Bảng 2 - 1 Các thông số rotate trong Rho	12
Bảng 2 - 2 Các thông số rotate trong Rho khi $w = 64$	13
Bảng 2 - 3 Các thông số trong từng giải thuật SHA-3	17
Bảng 3 - 1 So sánh giữa các thiết kế	19
Bảng 3 - 2 Thông số hậu tố trong từng giải thuật băm SHA-3	20
Bảng 4 - 1 Chân kết nối trong khối Buffer_in	25
Bảng 4 - 2 Bảng số Byte tương ứng mỗi MODE	26
Bảng 4 - 3 Chân kết nối trong khối VSX	27
Bảng 4 - 4 Chân kết nối trong khối STA	28
Bảng 4 - 5 Chân kết nối trong khối ATS	29
Bảng 4 - 6 Chân kết nối trong khối Transformation_round	31
Bảng 4 - 7 Chân kết nối trong khối Trunc	32
Bảng 4 - 8 Chân kết nối trong khối Control	34
Bảng 4 - 9 Các trạng thái trong khối Control	35
Bảng 4 - 10 Các tín hiệu ngõ ra tương ứng trong khối Control	36
Bảng 4 - 11 Các thông số trong hàm Keccak	37

1. GIỚI THIỆU

1.1 Tổng quan

Trong lĩnh vực nghiên cứu về mã hóa (crypto), hàm băm là một trong những cốt lõi của lĩnh vực này. Hàm băm là một giải thuật nhằm sinh ra giá trị băm tương ứng với mỗi khối dữ liệu, ngõ ra của hàm băm là một chuỗi ký tự có độ dài cố định và chuỗi ký tự này là duy nhất gọi là “hash”. Vì những tính chất đặc biệt, hàm băm được ứng dụng nhiều, nhất là ở lĩnh vực mã hóa: chữ ký số, mã hóa lượng tử, ... Một ví dụ về ứng dụng nổi tiếng của hàm băm hiện nay là SHA2-256 sử dụng trong lĩnh vực Blockchain.

Một số hàm băm đã xuất hiện trước đây có thể kể đến như là MD2, MD4, MD5, SHA1, SHA2, ... Trong đó 2 hàm băm khá thông dụng là SHA1. Tuy nhiên vào năm 2005, SHA1 đã bị tìm ra lỗi bảo mật và đến nay không còn đảm bảo an toàn nữa. Hiện nay vẫn chưa có cuộc tấn công mã hóa nào có thể phá được tính bảo mật của SHA2 nên hiện nay hàm băm này được sử dụng phổ biến.

SHA-3 được NIST phát hành vào ngày 5 tháng 8 năm 2015. Đây có lẽ là tiêu chuẩn hàm băm mới nhất cho đến hiện nay. SHA-3 là một tập con của họ nguyên thủy mật mã rộng hơn là Keccak. Thuật toán Keccak được đưa ra bởi Guido Bertoni, Joan Daemen, Michael Peeters và Gilles Van Assche. Keccak dựa trên cấu trúc bọt biển (sponge). Cấu trúc này cũng có thể được sử dụng để xây dựng các nguyên thủy mã hóa khác như các hệ mật mã dòng. SHA-3 cũng có các kích cỡ đầu ra tương tự như SHA-2 bao gồm: 224, 256, 384 và 512 bit.

1.2 Tình hình nghiên cứu trong và ngoài nước

Dù là một tiêu chuẩn hàm băm mới xuất hiện và hiện tại việc triển khai SHA3 thay thế cho SHA2 là vẫn chưa được rộng rãi, các thiết kế phần cứng của SHA3 đã được triển khai từ lâu. Bởi vì tầm quan trọng và tiện ích của việc xử lý hàm băm trên phần cứng, nhiều thiết kế hàm băm này đã được đề xuất và thường thì có hai hướng chính trong đó một là tối ưu diện tích thiết kế phần cứng và hướng thứ hai là tăng tốc độ xử lý.

Trong thiết kế [4], nhóm tác giả đã đề xuất kiến trúc pipeline cho Keccak nhằm tăng tốc độ xử lý. Thiết kế được mô phỏng bằng Modelsim và được thực hiện trên Virtex -5.

Trong thiết kế [5], nhóm tác giả đã đề xuất kiến trúc cho Keccak giúp tiết kiệm được phần cứng đến 16% các tài nguyên công logic. Thiết kế hoạt động với tần số là 301.02MHz đồng thời throughput là 30.1Mbps/slice. Tuy nhiên thiết kế chỉ thực hiện với SHA3-512.

1.3 Mục tiêu đề tài

Nội dung 1: Tìm hiểu lý thuyết về hàm băm và một số ứng dụng của hàm băm.

Nội dung 2: Tìm hiểu lý thuyết về hàm bọt biển “Sponge Function”.

Nội dung 3: Tìm hiểu lý thuyết về hàm hoán vị Keccak.

Nội dung 4: Thiết kế hàm băm SHA3 bằng thuật toán Keccak.

* Hỗ trợ 4 đầu ra phổ biến là 224bit, 256bit, 384bit, 512bit theo chuẩn FIPS 202.

* Hỗ trợ hai giải thuật SHAKE-256 và SHAKE-512 với ngõ ra không cố định.

2. NỘI DUNG LÝ THUYẾT ĐỀ TÀI

2.1 Hàm băm và một số ứng dụng của hàm băm

2.1.1 Hàm băm

Hàm băm là giải thuật nhằm sinh ra các giá trị băm tương ứng với mỗi khối dữ liệu (có thể là một chuỗi ký tự, một đối tượng trong lập trình hướng đối tượng, ...). Mỗi giá trị tạo ra là duy nhất nhưng về lý thuyết vẫn có thể xảy ra hiện tượng trùng khóa hay còn gọi là đụng độ. Người ta vẫn chấp nhận nó và cố gắng cải thiện giải thuật để giảm thiểu sự đụng độ đó. Hàm băm thường được dùng để xây dựng bảng băm nhằm giảm chi phí tính toán khi tìm một khối dữ liệu trong một tập hợp nhờ việc so sánh các giá trị băm nhanh hơn việc so sánh những khối dữ liệu có kích thước lớn.

Hàm băm được ví như là trái tim của công nghệ mã hóa. Tầm quan trọng của hàm băm lần đầu tiên được nhận ra với việc phát minh ra mật mã khóa công khai (PKC) của Diffie và Hellman [Whitfield Diffie and Martin Hellman. New Directions in Cryptography] vào năm 1976 và trở thành một phần không thể thiếu của PKC từ đó. Thật không may, những tiến bộ gần đây trong phân tích mật mã đã bộc lộ những yếu điểm cố hữu trong hầu hết các hàm băm dẫn đến một cuộc nghiên cứu vào sâu hơn trong lĩnh vực này. Có hai phương pháp tiếp cận chính đó là cải tiến từ các công trình cũ hoặc là tiến hành xây dựng các giải thuật hàm băm khác [Cryptographic Hash Functions: Recent Design Trends and Security Notions]. Các hàm băm về cơ bản là hàm một chiều, chúng ánh xạ các đầu vào có giá trị tùy ý thành một chuỗi có độ dài cố định và đầu vào thường lớn hơn so với đầu ra do đó dẫn đến hiện tượng xung đột tức là hai giá trị ngõ vào cùng cho ra một giá trị hàm băm mà đã được nhắc đến ở trên. Một hàm băm tốt cần phải đảm bảo thỏa mãn các điều kiện sau:

- Tính toán nhanh với bất cứ thông tin ngõ vào nào.
- Không thể đảo ngược quá trình tạo ra giá trị băm đã cho.

Ví dụ: mã hash sử dụng giải thuật SHA-256 của đoạn mã vào “Hello” có giá trị là 0x185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969, tuy nhiên việc có được giá trị hàm băm là chuỗi mã hex trên thì không thể suy ra được đoạn mã ngõ vào là “Hello”.

- Không thể tìm thấy được hai ngõ vào cùng cho ra một giá trị băm.

Ví dụ: Cũng như ví dụ trên, với cùng một chuỗi mã hex là 0x185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969 không thể tìm được chuỗi thứ hai nào khác cho cùng giá trị hash trên ngoài đoạn mã “Hello”.

- Chỉ một thay đổi nhỏ ở ngõ vào cũng sẽ tạo ra một giá trị băm hoàn toàn không liên quan đến giá trị băm cũ.

Ví dụ: hai đoạn mã sau “Hello” và “Helo” sẽ cho ra hai giá trị băm (sử dụng giải thuật SHA-256) lần lượt như sau:

0x185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969 và
0x375738319e86099fe081fabee238c40d6f038959da383c99ca3fe146e5cc8b7e.

Hai mã hash trên hoàn toàn không liên quan gì với nhau mặc dù chỉ có một thay đổi nhỏ ở ngõ vào.

Từ đoạn mã “Hello” ban đầu sau quá trình băm thì trở thành một chuỗi số mà chúng ta nhìn vào sẽ không thể hiểu được. Điều này có vẻ giống với việc mã hóa (encrypt) nhưng khác biệt ở chỗ, ở mã hóa ta có thể giải mã được từ chuỗi số khó hiểu đó còn ở hàm băm thì không như tính chất đã trình bày ở trên. Điều này giúp cho hàm băm có ứng dụng đặc biệt trong công nghệ mã hóa và trở thành một phần không thể thiếu của công nghệ này.

2.1.2 Ứng dụng của hàm băm

Các hàm băm được ứng dụng trong nhiều lĩnh vực, chúng thường được thiết kế để phù hợp với từng ứng dụng. Ngoài ra, chúng cũng có thể được sử dụng như các hàm băm thông thường, để lập chỉ mục dữ liệu trong bảng băm, lấy đặc trưng của dữ liệu, phát hiện dữ liệu trùng lặp hoặc làm tổng kiểm tra để phát hiện lỗi các dữ liệu ngẫu nhiên.

- Trong khoa học máy tính, bảng băm là một cấu trúc dữ liệu sử dụng hàm băm để ánh xạ từ giá trị xác định, được gọi là khóa đến giá trị tương ứng. Do đó bảng băm là một bảng kết hợp. Hàm băm được sử dụng để chuyển đổi từ khóa thành chỉ số (giá trị băm) trong mảng lưu trữ các giá trị tìm kiếm [wiki]. Bảng băm là một ứng dụng quan trọng của hàm băm.

- Hàm băm đảm bảo tính toàn vẹn và xác thực của thông tin. Việc so sánh các giá trị băm (digest) sau khi truyền đi có thể xác định được xem có bất kỳ thay đổi nào được thực

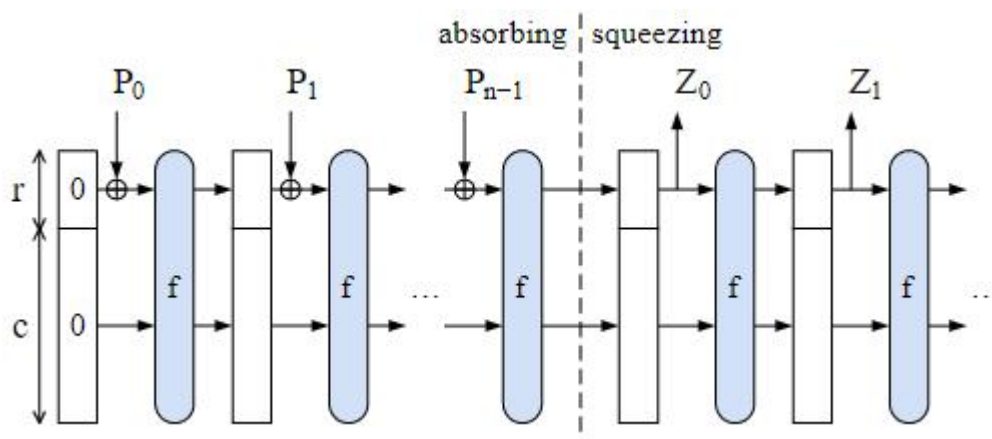
hiện đối với dữ liệu ban đầu hay không. Một khi kiểm tra thấy được có sự thay đổi ở giá trị băm có nghĩa là thông tin đã có sự thay đổi (dựa vào tính chất của hàm băm) và ta sẽ biết được. Các hàm băm MD5, SHA-1 hoặc SHA-2 đôi khi được sử dụng trên các trang web hoặc diễn đàn để cho phép xác minh tính toàn vẹn cho các tệp được tải xuống. Phương pháp này thiết lập một chuỗi tin cậy miễn là các hàm băm được đăng trên một trang web đáng tin cậy - được xác thực bởi HTTPS. Sử dụng hàm băm mật mã và chuỗi tin cậy sẽ phát hiện các thay đổi độc hại đối với tệp. Các mã phát hiện lỗi không liên quan đến mật mã như kiểm tra dự phòng theo chu kỳ chỉ ngăn chặn các thay đổi không độc hại của tệp, vì một hành vi giả mạo có chủ đích có thể dễ dàng được tạo ra để có giá trị mã va chạm.

- Tạo và xác minh chữ ký số. Hầu như tất cả các lược đồ chữ ký số đều yêu cầu tính toán bản tóm lược của thông điệp bằng các hàm băm mật mã. Điều này cho phép việc tính toán và tạo chữ ký được thực hiện trên một khối dữ liệu có kích thước tương đối nhỏ và cố định thay vì trên toàn bộ văn bản dài. Tính chất toàn vẹn thông điệp của hàm băm mật mã được sử dụng để tạo các lược đồ chữ ký số an toàn và hiệu quả.

- Việc sử dụng hàm băm để xác minh mật khẩu là hiệu quả hơn nhiều so với việc sử dụng dạng văn bản (plain text) hay là dạng mã hóa (encryption). Nếu ta sử dụng plain text để lưu dữ liệu mật khẩu, rõ ràng nếu như một hacker tìm được plain text đó, họ có thể dễ dàng biết được mật khẩu vì plain text là ở dạng văn bản có thể đọc được. Nếu sử dụng encryption, thì như đã trình bày ở trên, nếu hacker tìm ra được khóa để có thể giải mã ngược lại thông điệp đó thì cũng sẽ biết được mật khẩu dẫn đến ta bị mất thông tin. Còn sử dụng hàm băm thì khác, nếu hacker có được giá trị băm thì cũng không thể nào truy ngược lại giá trị văn bản ban đầu bởi vì tính chất của hàm băm do đó việc bảo mật mật khẩu sẽ an toàn hơn.

2.2 Sponge Function

Sponge Function hay Sponge Construction là hàm bọt biển hay cấu trúc bọt biển, là một loại thuật toán mà có số trạng thái hữu hạn ở bên trong, nhận đầu vào có độ dài bất kỳ và tạo ra đầu ra có độ dài mà ta mong muốn. Hàm bọt biển khởi tạo cấu trúc bọt biển, là cấu trúc lặp đi lặp lại đơn giản, xây dựng một hàm tạo ra đầu ra có độ dài cố định dựa trên việc hoán vị một chuỗi độ dài cố định. Với giao diện này, một hàm bọt biển cũng có thể được sử dụng như một mật mã dòng (stream cypher) hoặc như là một hàm băm, do đó hàm bọt biển bao gồm một loạt các chức năng với các hàm băm và mật mã dòng. Cấu trúc bọt biển đóng vai trò quan trọng trong việc thiết kế thuật toán Keccak.



Hình 2 - 1 Cấu trúc bọt biển

Cấu trúc bọt biển là một cấu trúc lặp đi lặp lại liên tục để xây dựng một hàm F với đầu vào tùy ý thành đầu ra có độ dài cố định dựa trên phép hoán vị độ dài cố định f với độ dài cố định là b . Ở đây b được gọi là chiều rộng (width). Cấu trúc bọt biển hoạt động trên trạng thái $b = r + c$ bit. Giá trị r được gọi là tốc độ bit (bitrate) và giá trị c là dung lượng (capacity). Các bit c cuối cùng của trạng thái không bao giờ bị ảnh hưởng trực tiếp bởi các khối đầu vào và không bao giờ được xuất ra trong giai đoạn ép.

Một hàm bọt biển hay cấu trúc bọt biển được xây dựng từ 3 thành phần chính:

- * Một mảng trạng thái bao gồm b bit trong đó gồm 2 phần là r (bitrate) và c (capacity) đã trình bày ở trên.

- * Phép hoán vị $f: \{0,1\}^b \rightarrow \{0,1\}^b$ để biến đổi mảng trạng thái này thành mảng trạng thái khác.

- * Hàm đệm P để thêm các bit vào đầu vào để hình thành một mảng trạng thái có độ dài là bội số của b thì mới có thể đưa vào hàm bọt biển.

Algorithm 1 The sponge construction $\text{SPONGE}[f, \text{pad}, r]$

Require: $r < b$

Interface: $Z = \text{sponge}(M, \ell)$ with $M \in \mathbb{Z}_2^*$, integer $\ell > 0$ and $Z \in \mathbb{Z}_2^\ell$

$P = M || \text{pad}[r](|M|)$

$s = 0^b$

for $i = 0$ to $|P|_r - 1$ **do**

$s = s \oplus (P_i || 0^{b-r})$

$s = f(s)$

end for

$Z = \lfloor s \rfloor_r$

while $|Z|_{r,r} < \ell$ **do**

$s = f(s)$

$Z = Z || \lfloor s \rfloor_r$

end while

return $\lfloor Z \rfloor_\ell$

Hình 2 - 2 Giải thuật bọt biển

Đầu tiên, tất cả các bit của trạng thái được khởi tạo bằng không. Thông điệp đầu vào được đệm và cắt thành các khối r bit. Sau đó, quá trình xây dựng bọt biển được tiến hành theo hai giai đoạn: giai đoạn hấp thụ (absorbing) tiếp theo là giai đoạn ép (squeezing).

* Trong giai đoạn hấp thụ, những khối r -bit đầu vào được XOR với khối r -bit trong trạng thái, xen kẽ với hàm hoán vị f . Khi tất cả đầu vào đều được xử lý, cấu trúc bọt biển sẽ chuyển sang giai đoạn ép. Figure 3 chỉ ra giải thuật cho giai đoạn hấp thụ.

Algorithm 3 The absorbing function $\text{ABSORB}[f, r]$

Require: $r < b$

Interface: $s = \text{absorb}(P)$ with $P \in \mathbb{Z}_2^*$, and $s \in \mathbb{Z}_2^b$

$s = 0^b$

for $i = 0$ to $|P|_r - 1$ **do**

$s = s \oplus (P_i || 0^{b-r})$

$s = f(s)$

end for

return s

Hình 2 - 3 Hàm absorbing function

* Trong giai đoạn ép, r bit đầu tiên của trạng thái được trả về dưới dạng các khối đầu ra, xen kẽ với hàm hoán vị f , nếu như số lượng khối đầu ra lớn hơn r - do người dùng tùy ý chọn thì hoán vị f sẽ được sử dụng để tạo ra các bit tiếp theo. Figure 4 chỉ ra giải thuật cho giai đoạn ép.

Algorithm 4 The squeezing function $\text{SQUEEZE}[f, r]$ **Require:** $r < b$

Interface: $Z = \text{squeeze}(s, \ell)$ with $s \in \mathbb{Z}_2^b$, integer $\ell > 0$ and $Z \in \mathbb{Z}_2^\ell$
 $Z = \lfloor s \rfloor_r$
while $|Z|_r < \ell$ **do**
 $s = f(s)$
 $Z = Z || \lfloor s \rfloor_r$
end while
return $\lfloor Z \rfloor_\ell$

Hình 2 - 4 Hàm Squeezing function

Với bộ ba thành phần: phép hoán vị f , phép đệm Pad, và bitrate r như đã trình bày ở trên, ta có thể xây dựng được một cấu trúc bọt biển.

Thuật toán bọt biển SPONGE[f , Pad, r](N, d) được trình bày như sau:

Input: Chuỗi bit đầu vào N và số nguyên không âm d là chiều dài của output.

Output: Chuỗi bit Z trong đó $\text{len}(Z) = d$.

Các bước thực hiện như sau:

1. Đặt $P = N || \text{Pad}(r, \text{len}(N))$.
2. Đặt $n = \text{len}(P) / r$.
3. Đặt $c = b - r$.
4. Đặt $P_0, P_1, \dots, P_{n-2}, P_{n-1}$ được tách ra từ $P = P_0 || P_1 || \dots || P_{n-2} || P_{n-1}$.
5. Đặt $S = 0^b$.
6. Cho i chạy từ 0 đến $n-1$, đặt $S = f(S \oplus (P_i || 0^c))$.
7. Đặt Z là một chuỗi rỗng.
8. Đặt $Z = Z || \text{Trunc}_r(S)$: Trong đó $\text{Trunc}_r(S)$ là cắt r bit trong chuỗi S .
9. Nếu $d \leq |Z|$, ta sẽ trả về kết quả là $\text{Trunc}_d(Z)$, nếu không thì sẽ tiếp tục bước 10.
10. Đặt $S = f(S)$, quay lại bước 8.

d quyết định chiều dài của ngõ ra chứ không ảnh hưởng đến giá trị và thứ tự các bit trong chuỗi của chúng. Thực tế thì d có thể là một số vô hạn và vòng lặp chỉ kết thúc khi đạt đến độ dài ngõ ra mong muốn.

2.3 Bộ đệm Pad₁₀*1

Ta đã biết được nguyên lý hoạt động của hàm bọt biển, nhận đầu vào có độ dài bất kỳ và cho kết quả ngõ ra có độ dài cố định. Để làm được điều đó, ta cần phải chia đầu vào thành

các khối để xử lý. Trong khi ngõ vào có độ dài không cố định, ta cần phải có thêm một bộ đệm để thêm vào những bit bổ sung để tạo thành một khối có đủ độ dài để xử lý. Bộ đệm sử dụng trong Keccak là Multi-rate padding hay còn được ký hiệu là pad10^*1 được thực hiện khá đơn giản đó là thêm bit 1 sau chuỗi ngõ vào, sau đó là một chuỗi các bit 0 và kết thúc bằng một bit 1. Số bit 0 sẽ tùy thuộc vào độ dài của chuỗi đầu vào ban đầu mà tạo thành một chuỗi đầu vào sau khi qua bộ đệm có độ dài là bội số của giá trị bitrate r . Pad10^*1 thêm vào tối thiểu là 2 bit và tối đa là tối đa là $b + 1$ bit. Thuật toán $\text{Pad10}^*1(x, m)$ được biểu diễn như sau:

Input: số nguyên dương x và số nguyên không âm m .

Output: chuỗi P mà $m + \text{len}(P)$ là bội số của x .

Các bước thực hiện như sau:

1. Đặt $j = (-m - 2) \bmod x$.
2. Trả lại kết quả $P = 1 \parallel 0^j \parallel 1$.

Số bit 0 không cố định sẽ tạo ra được chiều dài P cần thiết để thực hiện tính toán trong thuật toán bọt biển.

2.4 Hàm hoán vị Keccak

Hàm băm thiết kế theo thuật toán Keccak sử dụng cấu trúc bọt biển như đã trình bày ở 2.2, cấu trúc này sử dụng các hoán vị lặp dựa trên nguyên lý lặp đi lặp lại các biến đổi tuyến tính và phi tuyến. Theo đó biến đổi phi tuyến cung cấp tính xáo trộn cho các bit được xử lý qua hàm vòng, còn biến đổi tuyến tính sẽ đảm đương nhiệm vụ khuếch tán rộng hơn tính xáo trộn này. Việc sử dụng đơn lẻ hai tính chất này sẽ không mang lại hiệu quả trong các thiết kế mật mã. Chúng chỉ mang lại hiệu quả khi được kết hợp với nhau. Keccak là hàm băm đã chiến thắng trong cuộc thi tuyển chọn hàm băm SHA-3 do NIST - tổ chức an ninh mật mã Hoa Kỳ tổ chức. Nguyên lý thiết kế của nó cũng dựa trên nguyên tắc trên. Hàm vòng của nó có dạng theo tiêu chuẩn FIPS 202 như sau:

$$\text{Rnd}(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$$

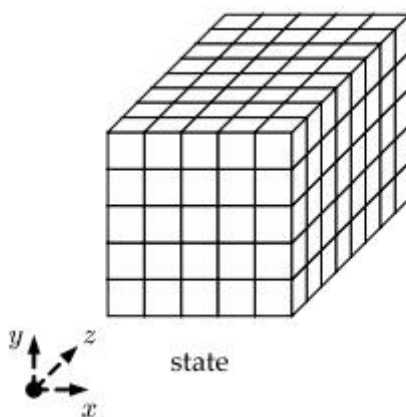
Trong đó, tầng tuyến tính của nó là kết hợp bởi một số thành phần tuyến tính như phép biến đổi Theta (phép θ), biến đổi Pi (phép π), biến đổi Rho (phép ρ) và phép biến đổi Iota (phép ι). Còn biến đổi phi tuyến được đảm bảo bởi phép biến đổi Chi (phép χ). Mỗi phép biến đổi trên được gọi là một phép ánh xạ (step mapping). Chi tiết sẽ được trình bày ở 2.4.1 - 2.4.5.

Trạng thái của một mảng các bit được cập nhật liên tục trong quá trình xử lý. Đối với phép hoán vị Keccak-p, trạng thái được biểu diễn bằng một chuỗi (S) hoặc là một mảng 3 chiều (A). Phép hoán vị Keccak-p gồm hai thông số đó là b là chiều rộng (width) đã trình bày ở trên và n_r gọi là số lần lặp của phép hoán vị, ký hiệu là Keccak-p[b, n_r]. Trong đó b có thể là một trong số các số sau {25, 50, 100, 200, 400, 800, 1600} và n_r là bất kỳ số nguyên dương nào. Bản đặc tả thông số kỹ thuật trong bộ tiêu chuẩn SHA3 bao gồm thêm hai đại lượng khác liên quan đến b là $b/25$ và $\log_2(b/25)$ lần lượt ký hiệu là w và l trong đó $w = 2^l$, l thuộc {0, 1, 2, 3, 4, 5, 6}.

* Nếu S ký hiệu là một chuỗi biểu diễn trạng thái thì các bit của nó được đánh số từ 1 đến b-1, do đó

$$S = S[0]||S[1]||S[2]||\dots||S[b-1]$$

Nếu biểu diễn trạng thái đó dưới dạng một mảng A 3 chiều có kích thước 5 x 5 x w bit khi đó các chỉ số thỏa mãn $0 \leq x \leq 4$, $0 \leq y \leq 4$, $0 \leq z \leq (w - 1)$. Trong đó, tích $5 \times 5 \times w = b$ chính là số bit của mảng A. Figure 5 biểu diễn mảng trạng thái A trong trường hợp b = 200 do đó w = 8.



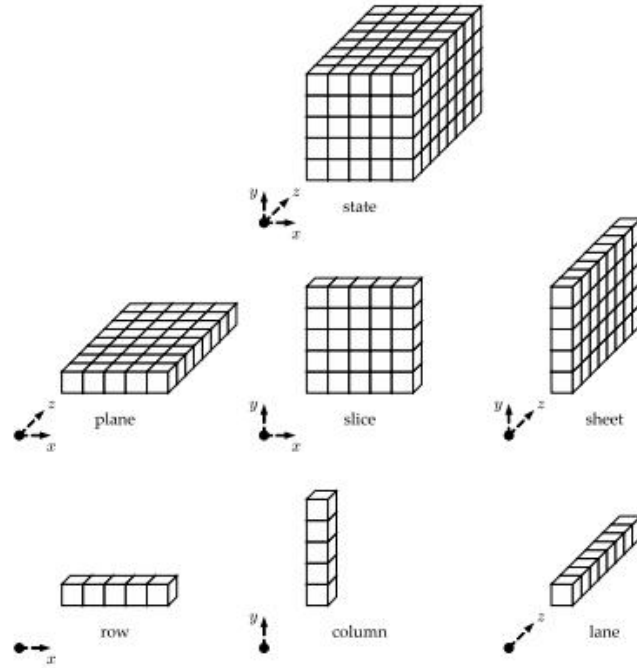
Hình 2 - 5 Mảng trạng thái 3 chiều

Mảng trạng thái A là một mảng 3 chiều trong đó các mảng con hai chiều được tạo thành từ A gồm các mảng *sheet*, *plane* và *slice*; các mảng con một chiều được tạo thành từ A gồm các mảng *column*, *row* và *lane*, trong đó:

- * *sheet*: là mảng con hai chiều gồm $b/5$ bit theo trục tọa độ x cố định.
- * *plane*: là mảng con hai chiều gồm $b/5$ bit theo trục tọa độ y cố định.
- * *slice*: là mảng con hai chiều gồm 25 bit theo trục tọa độ z cố định.
- * *column*: là mảng con một chiều gồm 5 bit theo trục tọa độ x và z cố định.

* *row*: là mảng con một chiều gồm 5 bit theo trục tọa độ y và z cố định.

* *lane*: là mảng con một chiều gồm b/25 bit theo trục tọa độ x và y cố định.



Hình 2 - 6 Các thành phần của mảng trạng thái

Để biểu diễn một chuỗi trạng thái từ mảng trạng thái 3 chiều, ta định nghĩa như sau:

Đối với mỗi cặp số nguyên (i, j) sao cho $0 \leq i \leq 4$, $0 \leq j \leq 4$, xác định chuỗi $lane[i, j]$:

$$lane[i, j] = A[i, j, 0] || A[i, j, 1] || A[i, j, 2] || \dots || A[i, j, w-1].$$

Đối với mỗi số nguyên i sao cho $0 \leq i \leq 4$, xác định $plane[i]$:

$$plane[i] = lane[0, i] || lane[1, i] || lane[2, i] || lane[3, i] || lane[4, i].$$

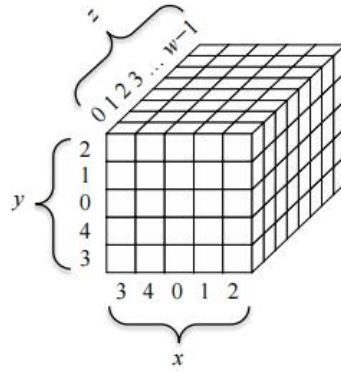
Do đó,

$$S = plane[0] || plane[1] || plane[2] || plane[3] || plane[4].$$

Để biểu diễn một mảng trạng thái từ chuỗi trạng thái ta định nghĩa như sau: đối với mọi bộ ba (x, y, z) sao cho $0 \leq x \leq 4$, $0 \leq y \leq 4$, $0 \leq z \leq (w - 1)$, ta có:

$$A[x, y, z] = S[w(5y+x) + z]$$

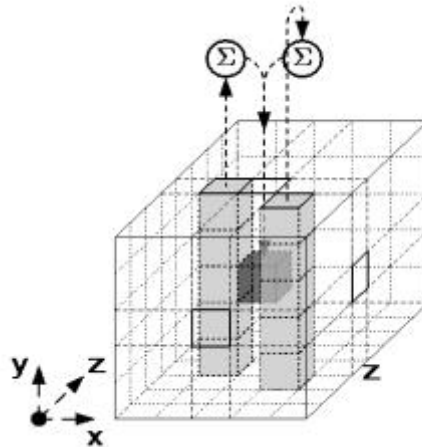
Trong đó quy ước tọa độ $(x, y) = (0, 0)$ nằm ở trung tâm của *slice* như sau:



Hình 2 - 7 Ký hiệu chiều x, y, z của mảng trạng thái

Như đã trình bày ở phần trên, hàm vòng trong Keccak được biểu diễn bằng 5 phép ánh xạ trong đó ngoại trừ phép ι có thêm một ngõ vào thứ hai là i_r , các phép còn lại có ngõ vào là một mảng trạng thái ba chiều và ngõ ra cũng là một mảng trạng thái ba chiều đã được biến đổi.

2.4.1 Biến đổi Θ



Hình 2 - 8 Biến đổi Theta

Input: Mảng trạng thái A

Output: Mảng trạng thái A'

Các bước biến đổi như sau:

1. Với tất cả các cặp (x, z) với $0 \leq x \leq 4$ và $0 \leq z \leq (w-1)$

$$C[x, z] = A[x, 0, z] \oplus A[x, 1, z] \oplus A[x, 2, z] \oplus A[x, 3, z] \oplus A[x, 4, z]$$

2. Với tất cả các cặp (x, z) với $0 \leq x \leq 4$ và $0 \leq z \leq (w-1)$

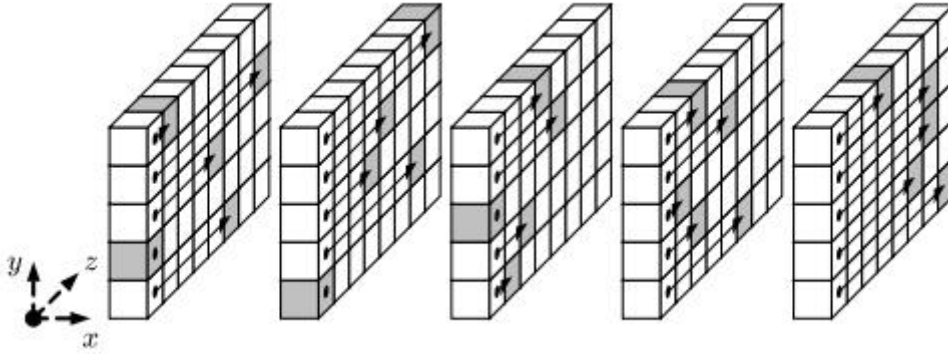
$$D[x, z] = C[(x - 1) \bmod 5, z] \oplus C[(x + 1) \bmod 5, (z - 1) \bmod w]$$

3. Với tất cả các bộ ba (x, y, z) với $0 \leq x \leq 4, 0 \leq y \leq 4, 0 \leq z \leq (w-1)$

$$A'[x, y, z] = A[x, y, z] \oplus D[x, z]$$

Minh họa phép biến đổi θ như trong Figure 8.

2.4.2 Biến đổi ρ



Hình 2 - 9 Biến đổi Rho

Input: Mảng trạng thái A

Output: Mảng trạng thái A'

Các bước biến đổi như sau:

1. Với tất cả z với $0 \leq z \leq (w-1)$, ta đặt $A'[0, 0, z] = A[0, 0, z]$.
2. Đặt $(x, y) = (1, 0)$.
3. Cho t chạy từ 0 đến 23:
 - a. Với tất cả z thỏa mãn $0 \leq z \leq (w-1)$ ta đặt

$$A'[x, y, z] = A[x, y, (z-(t+1)(t+2)/2) \bmod w]$$
 - b. Đặt $[x, y] = [y, (2x + 3y) \bmod 5]$.
4. Trả lại kết quả A'.

Tác động của biến đổi ρ làm xoay các bit của từng *lane* theo 1 chiều dài gọi là *offset*, với việc phụ thuộc vào các tọa độ cố định của x và y trong *lane*. Tương ứng với từng bit trong *lane* tọa độ z được sửa đổi bằng cách cộng modulo các *offset* theo kích thước *lane*.

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	153	231	3	10	171
$y=1$	55	276	36	300	6
$y=0$	28	91	0	1	190
$y=4$	120	78	210	66	253
$y=3$	21	136	105	45	15

Bảng 2 - 1 Các thông số rotate trong Rho

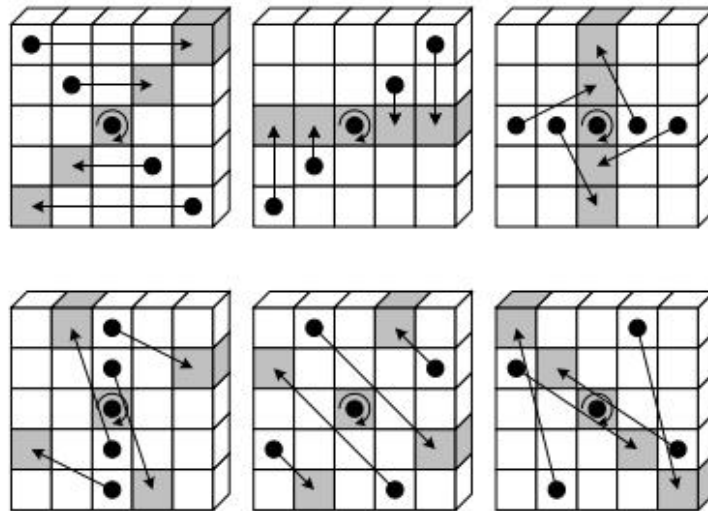
Trong hàm băm Keccak sử dụng để xây dựng SHA-3, $w = 64$ nên ta xây dựng được Bảng 2 - 2 chứa các thông số rotate sau:

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	25	39	3	10	43
$y=1$	55	20	36	44	6
$y=0$	28	27	0	1	62
$y=4$	56	14	18	2	61
$y=3$	21	8	41	45	15

Bảng 2 - 2 Các thông số rotate trong Rho khi $w = 64$

Minh họa phép biến đổi ρ với $w = 8$ được biểu diễn trong Figure 9 và các thông số *offset* được tính sẵn như trong Bảng 2 - 2.

2.4.3 Biến đổi π



Hình 2 - 10 Biến đổi Pi

Input: Mảng trạng thái A

Output: Mảng trạng thái A'

Các bước biến đổi như sau:

1. Với tất cả bộ ba (x, y, z) thỏa mãn điều kiện $0 \leq x \leq 4$, $0 \leq y \leq 4$, $0 \leq z \leq (w-1)$, đặt:

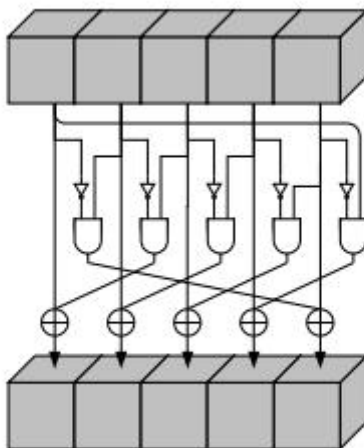
$$A'[x, y, z] = A[(x + 3y) \bmod 5, x, z]$$

2. Trả lại kết quả A'.

Biến đổi π thực chất là phép hoán vị các bit trên một *slice* của một trạng thái. Việc

hoán vị này là giống nhau cho toàn bộ w slice trong mảng trạng thái. Như vậy có thể ghép tất cả slice này và thực hiện hoán vị $lane$ trong khối trạng thái. Figure 10 mô tả tác động của biến đổi π lên các bit của mảng trạng thái.

2.4.4 Biến đổi χ



Hình 2 - 11 Biến đổi Chi

Input: Mảng trạng thái A

Output: Mảng trạng thái A'

Các bước biến đổi như sau:

1. Với tất cả những bộ 3 (x, y, z) thỏa mãn những điều kiện $0 \leq x \leq 4$, $0 \leq y \leq 4$, $0 \leq z \leq (w-1)$, đặt:

$$A'[x, y, z] = A[x, y, z] \oplus ((A[(x+1) \bmod 5, y, z] \oplus 1) \cdot A[(x+2) \bmod 5, y, z])$$

2. Trả lại kết quả A' .

Figure 11 biểu diễn tác động của biến đổi χ đến mảng trạng thái A .

2.4.5 Biến đổi ι

Biến đổi ι chỉ tác động lên $lane$ gốc, nghĩa là $lane$ có tọa độ $x = 0, y = 0$. Bản chất của nó là cộng vào $lane$ gốc các hằng số phụ thuộc vào chỉ số vòng của hoán vị. Phép biến đổi ι được tham số hóa bởi chỉ số vòng i_r . Trong phạm vi phép biến đổi ở thuật toán ι , tham số này xác định $l+1$ bit của giá trị $lane$ được gọi là hằng số vòng, và ký hiệu là RC . Mỗi bit của $l+1$ bit được tạo ra bởi một hàm mà hàm này dựa trên một thanh ghi dịch tuyến tính có phản hồi. Hàm này ký hiệu là rc và được định nghĩa ở thuật toán bên dưới.

Thuật toán $rc(t)$

Input: Số nguyên t

Output: bit $rc(t)$

Các bước của thuật toán:

1. Nếu $t \bmod 255 = 0$, trả lại kết quả 1
2. Đặt $R = 10000000$
3. Cho i chạy từ 1 tới $t \bmod 255$, đặt:

$$3.1. R = 0 \parallel R$$

$$3.2. R[0] = R[0] \oplus R[8]$$

$$3.3. R[4] = R[4] \oplus R[8]$$

$$3.4. R[5] = R[5] \oplus R[8]$$

$$3.5. R[6] = R[6] \oplus R[8]$$

$$3.6. R = Trunc_8[R]$$

4. Trả lại kết quả $R[0]$

Tác động của phép biến đổi ι là để biến đổi một vài bit của $lane[0,0]$ phụ thuộc vào chỉ số của i_r , còn lại 24 $lane$ khác đều không bị ảnh hưởng bởi phép biến đổi ι . Biến đổi ι bao gồm việc thêm các hằng số vòng và hướng tới phá vỡ tính đối xứng. Các bit của hằng số vòng là khác nhau từ vòng này đến vòng kia và được lấy là đầu ra của LFSR (Linear-feedback shift register) có độ dài lớn nhất. Các hằng số này chỉ được thêm trong một $lane$ của trạng thái. Do đó sự phá vỡ này sẽ được lan truyền thông qua θ và χ đối với tất cả $lane$ của trạng thái.

Thuật toán ι

Input: Mảng trạng thái A và chỉ số vòng i_r

Output: Mảng trạng thái A'

Các bước biến đổi như sau:

1. Với tất cả những bộ 3 (x, y, z) thỏa mãn những điều kiện $0 \leq x \leq 4, 0 \leq y \leq 4, 0 \leq z \leq (w-1)$, đặt:

$$A'[x, y, z] = A[x, y, z]$$

2. Đặt $RC = 0^w$

3. Cho j chạy từ 0 đến l ta đặt:

$$RC[2^j - 1] = rc(j + 7i_r)$$

4. Với tất cả z thỏa mãn $0 \leq z \leq (w-1)$, ta đặt:

$$A'[0, 0, z] = A'[0, 0, z] \oplus RC[z]$$

5. Trả lại kết quả A' .

2.4.6 Hàm hoán vị Keccak-p[b,n_r]

Keccak-p[b,n_r] là hàm hoán vị biến đổi một chuỗi bit ban đầu S có độ dài b bit thành một chuỗi S' có độ dài b bit gồm hai thông số: b - chiều dài của chuỗi bit và n_r là số vòng lặp trong phép biến đổi. Trong Keccak-p sử dụng hàm vòng Rnd như đã trình bày trên.

$$\text{Rnd}(A, i_r) = \iota(\chi(\pi(\rho(\theta(A))))), i_r)$$

Phép hoán vị Keccak-p được trình bày như sau:

Input: Chuỗi bit S có độ dài b và số vòng lặp n_r .

Output: Chuỗi bit S' có độ dài b .

Các bước biến đổi như sau:

1. Biến đổi chuỗi S thành ma trận trạng thái 3 chiều A như đã trình bày ở 2.4.
2. Cho i_r chạy từ $12 + 2l - n_r$ đến $12 + 2l - 1$, đặt $A = \text{Rnd}(A, i_r)$.
3. Chuyển đổi ma trận trạng thái 3 chiều A thành chuỗi S' có độ dài b .
4. Trả lại kết quả S' .

2.4.7 Hàm hoán vị Keccak-f[b]

Cũng là một họ của phép hoán vị Keccak. Keccak-f chính là Keccak-p[b,n_r] trong trường hợp $n_r = 12 + 2l$:

$$\text{Keccak-f}[b] = \text{Keccak-p}[b, 12 + 2l]$$

Keccak-f[1600,24] là cơ sở cho sáu hàm SHA-3 mà sẽ trình bày ở phần sau có thể viết ngắn gọn là Keccak-f[1600].

2.4.8 Hàm Keccak[c](N,d)

Như đã nhắc đến ở phần giới thiệu, Keccak được thiết kế dựa trên cấu trúc bọt biển do đó cần phải có những thành phần để xây dựng hàm bọt biển bao gồm mảng trạng thái, bộ đệm và phép hoán vị. Trong trường hợp $b = 1600$, bộ đệm sử dụng là $\text{Pad}10^*1$, và phép hoán vị là Keccak-f[1600] thì hàm bọt biển trở thành Keccak[c] chính là cấu trúc của thuật toán Keccak dùng để xây dựng các hàm SHA-3.

$$\text{Keccak}[c] = \text{SPONGE}[\text{Keccak-f}[1600], \text{Pad}10^*1, 1600 - c]$$

Keccak-c nhận đầu vào là N và d trong đó N là chuỗi bit đầu vào và d là độ dài của output.

$$\text{Keccak}[c](N,d) = \text{SPONGE}[\text{Keccak-f}[1600], \text{Pad}10^*1, 1600 - c](N,d)$$

2.5 Thiết kế SHA-3 bằng thuật toán Keccak

Như đã trình bày ở phần giới thiệu, Keccak là giải thuật đã giành chiến thắng trong cuộc thi thiết kế phần cứng cho giải thuật Secure Hash Algorithm -3 do NIST tổ chức. Cuộc thi này yêu cầu mỗi ứng viên phải đề xuất giải thuật hỗ trợ tối thiểu ngõ ra có 4 chiều dài khác nhau tùy thuộc vào chức năng của mỗi hệ thống. Họ SHA-3 gồm 4 hàm băm mã hóa chính : SHA3-224, SHA3-256, SHA3-384 và SHA3-512, và 2 hàm băm mở rộng: SHAKE128 và SHAKE256.

Với một đoạn thông điệp ký hiệu là M , bốn hàm băm mã hóa chính SHA-3 được định nghĩa bằng phép Keccak-[c] như đã trình bày ở 2.4.8 như sau:

$$\text{SHA3-224}(M) = \text{Keccak-[448]}(M||01, 224),$$

$$\text{SHA3-256}(M) = \text{Keccak-[512]}(M||01, 256),$$

$$\text{SHA3-384}(M) = \text{Keccak-[768]}(M||01, 384),$$

$$\text{SHA3-512}(M) = \text{Keccak-[1024]}(M||01, 512).$$

Trong mỗi trường hợp, capacity c bằng hai lần giá trị d - chiều dài của ngõ ra. Chuỗi ngõ vào N chính là thông điệp M chèn thêm hai bit suffix 01 phía sau để phân biệt với 2 thuật toán mở rộng là SHAKE128 và SHAKE256.

$$\text{SHAKE128}(M,d) = \text{Keccak-[256]}(M||1111, d),$$

$$\text{SHAKE256}(M,d) = \text{Keccak-[512]}(M||1111, d).$$

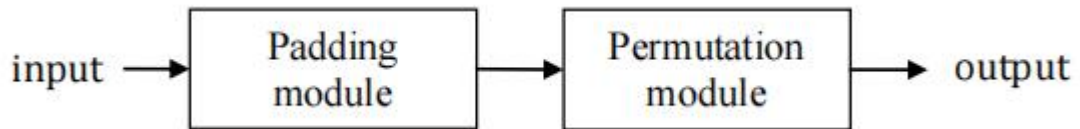
Khác với bốn hàm băm mã hóa chính của SHA-3 các hàm mở rộng này có độ dài output không cố định và phụ thuộc vào lựa chọn của người sử dụng. Chuỗi ngõ vào N chính là thông điệp M chèn thêm bốn bit 1111 phía sau.

Instance	Output Size d	Rate r -Block Size	Capacity c
SHA3-224(M)	224	1152	448
SHA3-256(M)	256	1088	512
SHA3-384(M)	384	832	768
SHA3-512(M)	512	576	1024

Bảng 2 - 3 Các thông số trong từng giải thuật SHA-3

3. GIẢI PHÁP THỰC HIỆN

Từ những nội dung lý thuyết đã trình bày ở phần 2, kiến trúc cơ bản của hàm băm Keccak cho giải thuật SHA-3 được trình bày đơn giản như sau



Hình 3 - 1 Sơ đồ tổng quát hàm băm SHA-3

Ngõ vào trước tiên được đi qua khối Padding để tạo thành một khối có độ dài là bội số của bitrate r . Padding Module được xây dựng dựa trên thuật toán Pad10*1 đã trình bày ở phần 2.3. Sau khi thực hiện Padding, ngõ ra của nó sẽ được đưa vào bộ hoán vị chính là module chính của giải thuật hàm băm Keccak chính là sự kết hợp giữa 5 hàm ánh xạ đã trình bày ở 2.4.1 - 2.4.5. Và sau đó kết quả được cắt ra theo đúng độ dài ngõ ra tương ứng và trả ra ngõ ra.

Một trong những thiết kế đáng để tham khảo đó là thiết kế [1]. Ở thiết kế này, tác giả đã đạt được thông lượng (throughput) khá ấn tượng nhờ việc sử dụng kỹ thuật pipeline để giảm thời gian tính toán tới hạn. Một thiết kế khác là [3], tác giả giới thiệu một thiết kế giúp tiết kiệm tài nguyên phần cứng bằng cách kết hợp các phép ánh xạ ρ , π , χ lại từ đó giảm đi phần cứng xử lý, tần số hoạt động cũng khá cao tuy nhiên đánh đổi với thông lượng (throughput) khá thấp. Trong thiết kế [2], nhóm tác giả giới thiệu 5 kiến trúc khác nhau áp dụng kết hợp các kỹ thuật pipeline, subpipeline, unrolling đạt những kết quả ấn tượng trong đó tần số và thông lượng khá cao tuy nhiên tốn khá nhiều tài nguyên phần cứng so với [1]. Một số thiết kế khác đạt kết quả ấn tượng cũng được trình bày theo bảng dưới, các thông số sử dụng trong bảng là sử dụng thiết bị của Xilinx.

Thiết kế	Thiết bị	Tần số (MHz)	Diện tích (Slices)	Throughput (Gbps)
[1]	Virtex-5	389	1702	18.7
[2]	Virtex-6	344	1406	16.51
[3]	Virtex-5	301	240	7.224
[4]	Virtex-5	287.39	1388	11.5
[5]	Virtex-5	312.98	1304	7.511

Bảng 3 - 1 So sánh giữa các thiết kế

Để thực hiện cứng hóa cho giải thuật hàm băm Keccak, em xin được trình bày 3 module chính đóng góp từ việc nhận giá trị ngõ vào, xử lý ngõ vào và các tín hiệu điều khiển quá trình xử lý của hệ thống trên cho đến việc xuất ra giá trị ngõ ra với độ dài tương ứng và một số module liên quan khác.

Việc ngõ vào có thể là một số 64bit, 128bit hay 1024bit ... dẫn đến số lượng I/O port khi mô phỏng bằng các thiết bị FPGA sẽ không đáp ứng đủ. Trong thiết kế của em, thông điệp cần được băm là một chuỗi có độ dài là bội số của 128. Tại thời điểm cạnh tích cực của clock, một chuỗi 128 bit được đưa vào và theo lý thuyết là không giới hạn độ dài chuỗi ngõ vào. Tại chuỗi 128 cuối cùng, sẽ có một tín hiệu thông báo đây là khối cuối cùng để hệ thống tiến hành xử lý. Để thực hiện được điều đó, một bộ đệm được sử dụng để lưu giá trị từ chuỗi đầu tiên đến chuỗi cuối cùng. Khi tín hiệu ngõ vào đã được đưa vào hết, giá trị trong buffer sẽ được đi vào module Padding.

3.1 Module Padding

Padding là một phần không thể thiếu trong các thiết kế dựa trên cấu trúc bọt biển, nó thêm vào các bit bổ sung để tạo thành một khối hoặc là nhiều khối có độ dài cố định từ chuỗi ngõ vào. Module Padding sử dụng trong thiết kế là Pad10*1 có nguyên lý hoạt động như đã trình bày ở 2.3. Nó sẽ chèn một bit 1 vào sau chuỗi ngõ vào và tiếp theo là chuỗi bit 0 và kết thúc là một bit 1.

Trước khi thực hiện Pad10*1, thông điệp ngõ vào phải được chèn thêm các bit suffix như đã trình bày ở phần 2.5.

	r	c	Output length (bits)	Security level (bits)	Mbits	d
SHAKE128	1344	256	unlimited	128	1111	0x1F
SHAKE256	1088	512	unlimited	256	1111	0x1F
SHA3-224	1152	448	224	112	01	0x06
SHA3-256	1088	512	256	128	01	0x06
SHA3-384	832	768	384	192	01	0x06
SHA3-512	576	1024	512	256	01	0x06

Bảng 3 - 2 Thông số hậu tố trong từng giải thuật băm SHA-3

Bảng 3 - 1 trình bày các bit hậu tố suffix (Mbits) và hậu tố phân cách (d) được mã hóa từ các Mbits theo sau và độ dài của nó. Các Mbits và d được quyết định dựa trên loại thuật toán mà ta sử dụng. Vì vậy cần có một bộ mux để lựa chọn giá trị chèn vào thích hợp. Ở 6 giải thuật chính trong Keccak, ta cần phải dùng tín hiệu lựa chọn có độ dài là 3 bit. Mỗi tín hiệu lựa chọn quyết định một cặp số r và c tương ứng làm ngõ vào cho Module Padding. Trong các giải thuật SHA-3 thì SHAKE128 có độ dài bitrate r cao nhất nên em sẽ sử dụng một buffer có kích thước là 11×128 để lưu trữ. Tùy thuộc vào tín hiệu lựa chọn mà độ dài của bitrate là khác nhau và buffer sẽ gửi thông báo khi độ dài lưu trữ trong buffer bằng với bitrate của thuật toán mà ta lựa chọn.

Ví dụ, khi thuật toán ta lựa chọn là SHA3-226 có $r = 1152$, $c = 448$ thì khi chuỗi ngõ vào thứ 9 được ghi vào buffer ($9 = 1152/128$), tín hiệu thông báo buffer đã đầy sẽ được xuất ra và chuỗi 1152bit đó sẽ được kết hợp với 448 bit 0 tạo thành một khối 1600 bit đưa vào hàm băm xử lý.

Ví dụ trên là trong trường hợp xử lý đa khối, tức là khi buffer đầy, chuỗi ngõ vào vẫn chưa dừng lại. Chuỗi ngõ vào thứ nhất được đưa vào quá trình xử lý và chuỗi ngõ vào thứ hai lần lượt được đưa vào buffer cho đến khi có chuỗi cuối cùng. Trong trường hợp chuỗi cuối cùng đã xuất xong mà buffer vẫn chưa đầy, lúc này sẽ thực hiện chèn các bit hậu tố và Pad10*1 đến khi buffer đầy và tiến hành xử lý.

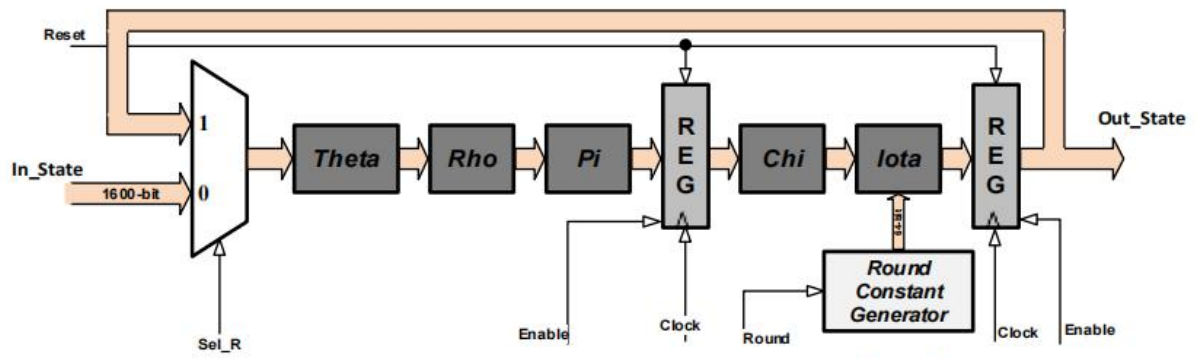
Ở cuối module Padding là một thanh ghi có vai trò làm giảm đường tới hạn, áp dụng kỹ thuật pipeline giúp xử lý các chuỗi khối tiết kiệm được thời gian hơn.

3.2 Module Step-Mapping

Đây là module chính thực hiện quá trình xử lý. Thông điệp sau khi đã đi qua k hồi Padding đã được chia thành nhiều khối, mỗi khối có độ dài là 1600bit bằng với số bit trong

ma trận trạng thái ba chiều mà ta sử dụng trong phép hoán vị Keccak đã được giới thiệu trong phần 2.4. Một hàm vòng là sự kết hợp giữa năm hàm ánh xạ và được lặp lại 24 lần trước khi đi vào bước Squeezing của cấu trúc bọt biển.

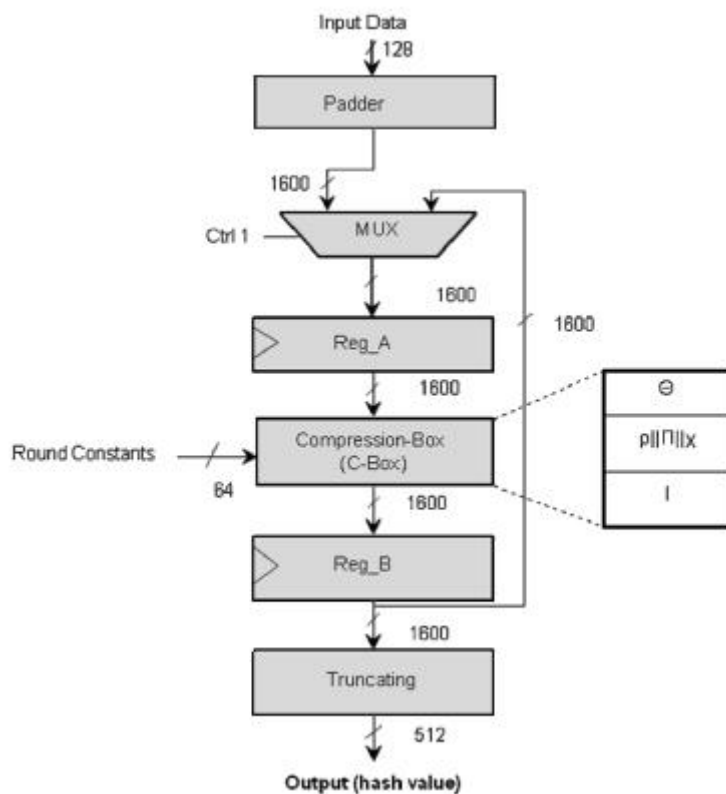
Một thiết kế đáng để tham khảo là thiết kế [1] khi tác giả đã áp dụng pipeline vào trong Module này và thu được kết quả là throughput rất lớn.



Hình 3 - 2 Sơ đồ khối StepMapping trong thiết kế [1]

Hai thanh ghi được chèn vào giữa bước *Pi*-*Chi* và cuối *Iota* để giảm đi thời gian tính toán tới hạn từ đó làm tăng tần số hoạt động của hệ thống.

Thiết kế [2] mang lại hiệu quả cao về diện tích vì tiêu thụ ít tài nguyên phần cứng có sơ đồ thiết kế cho module này như sau.



Hình 3 - 3 Sơ đồ khối tổng quát trong thiết kế [3]

Trong đề tài luận văn này, em sẽ cải tiến thiết kế giải thuật hàm băm Keccak bằng cách kết hợp pipeline như Hình 3 - 2 nhưng thanh ghi sẽ được đặt sau phép ánh xạ Theta và đồng thời gộp các phép ánh xạ Rho, Pi, Chi lại để tiết kiệm tài nguyên phần cứng hơn.

3.3 Module Control

Một module quan trọng khác nữa đó là module Control xử lý các tín hiệu điều khiển trong toàn hệ thống. Module Control sẽ cho ngõ ra là các tín hiệu điều khiển như: tín hiệu lựa chọn phép toán, tín hiệu thông báo đã bắt đầu quá trình băm, tín hiệu thông báo quá trình băm đã hoàn tất, tín hiệu đếm số vòng lặp trong phép hoán vị, ... Lựa chọn thiết kế của em đó là xây dựng module Control là một máy trạng thái (FSM).

4. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

4.1 Yêu cầu thiết kế

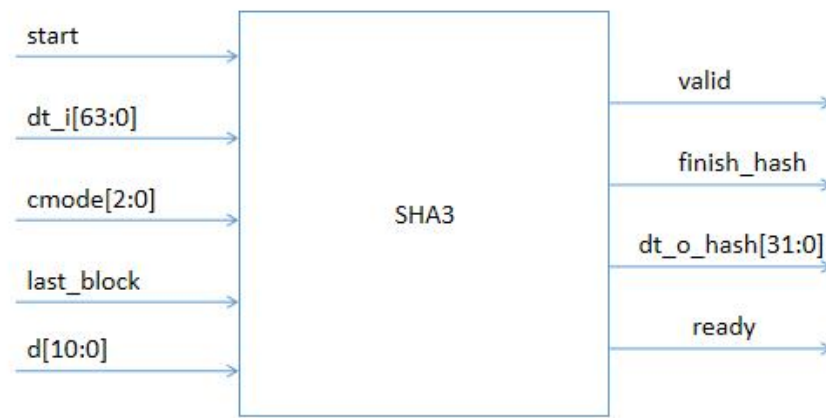
* Thiết kế giải thuật Keccak hỗ trợ cho các hàm băm SHA3-224, SHA3-256, SHA3-384, SHA3-512 và hai hàm mở rộng SHAKE-128, SHAKE-256 dưới dạng IP (Intellectual Property) core để sử dụng cho nhiều dự án khác nhau.

* Thiết kế đảm bảo tài nguyên tiết kiệm trên FPGA Altera Cyclone V.

* Thiết kế đạt tốc độ cao.

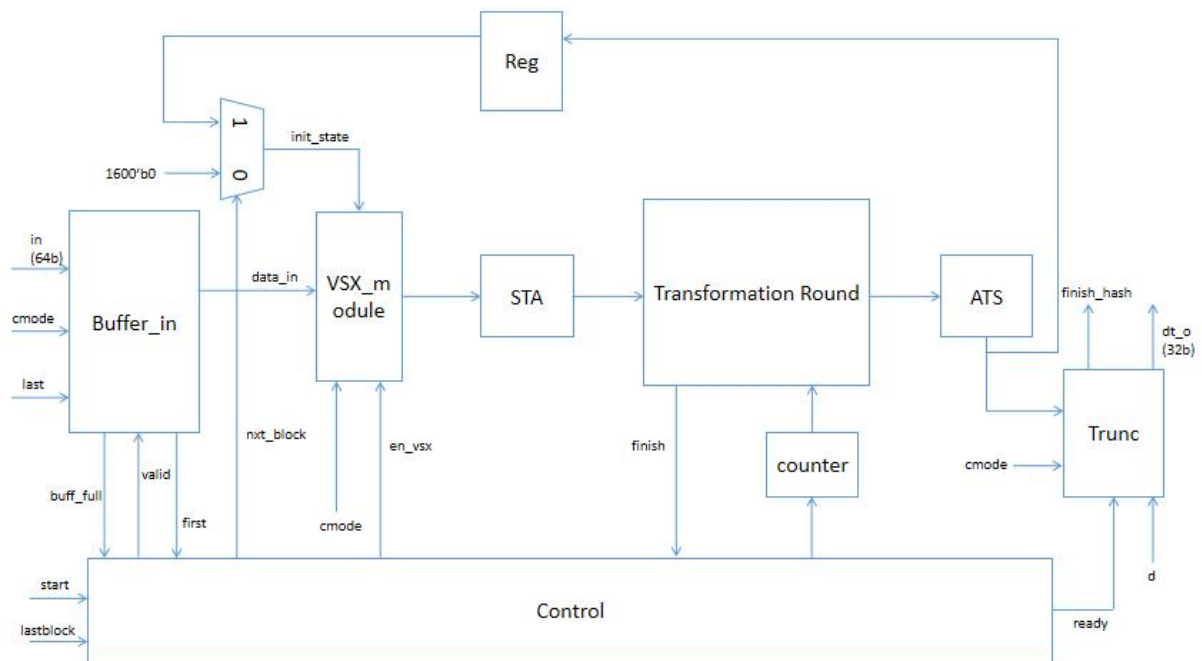
4.2 Phân tích thiết kế

Phương pháp thiết kế được thực hiện dưới dạng các mô đun phần cứng trực quan, dễ thay đổi và chỉnh sửa.



Hình 4 - 1 SHA3-core

4.3 Sơ đồ khối tổng quát

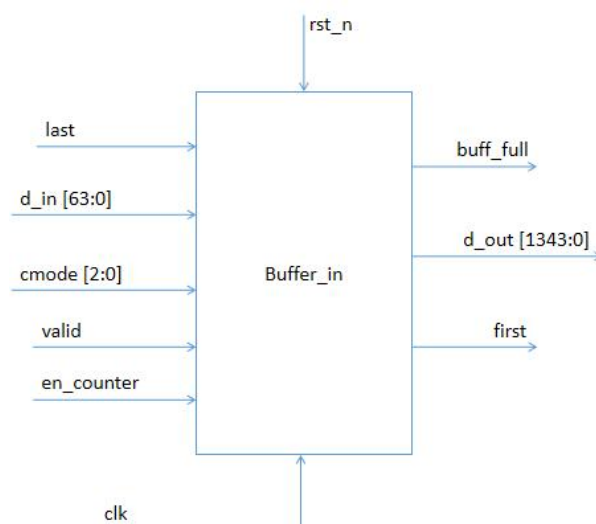


Hình 4 - 2 Sơ đồ khối tổng quát của thiết kế

4.4 Sơ đồ khối chi tiết và nhiệm vụ, chức năng từng khối

4.4.1 Bộ đệm ngõ vào Buffer_in

4.4.1.1 Mô hình chi tiết



Hình 4 - 3 Sơ đồ khối Buffer_in

4.4.1.2 Mô tả chân kết nối

Tín hiệu	Số bit	Ngõ vào	Ngõ ra	Mô tả
last	1	x		Tín hiệu thông báo chuỗi 64bit d_in đã là chuỗi cuối cùng thực hiện tính
d_in	64	x		Ngõ vào dữ liệu của hàm băm, gồm 64bit
cmode	3	x		Lựa chọn MODE của hàm băm
valid	1	x		Cho phép Buffer nhận giá trị từ d_in
en_counter	1	x		Tín hiệu cho phép khối Counter thực hiện, trong này dùng để xét điều
buff_full	1		x	Tín hiệu thông báo Buffer đã đầy
d_out	1344		x	Ngõ ra dữ liệu gồm 1344bit của Buffer, tín hiệu này sẽ kết hợp với tín hiệu từ Init_state tạo thành một khối 1600bit
first	1		x	Tín hiệu thông báo Block hiện tại là đầu tiên

Bảng 4 - 1 Chân kết nối trong khối Buffer_in

4.4.1.3 Mô tả chức năng

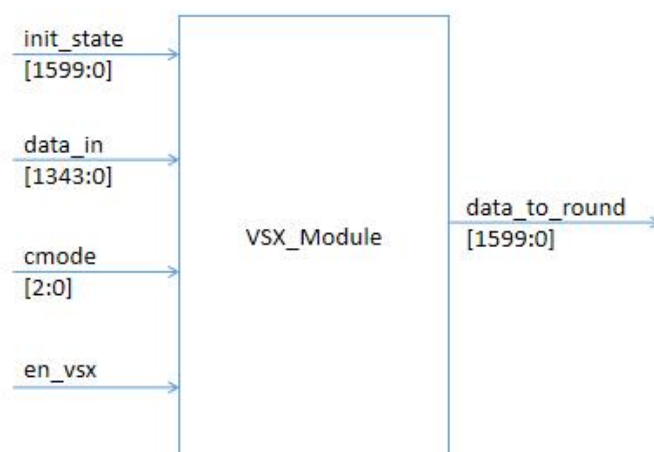
Buffer nhận giá trị từ ngõ vào đến lúc đầy thì sẽ bật tín hiệu buff_full hoặc khi có tín hiệu last thì sẽ dừng việc đọc lại. Trong trường hợp tín hiệu last ở mức cao mà buffer vẫn chưa đầy, thì buffer sẽ thực hiện việc padding với quy tắc pad10*1 cho đến khi buffer đầy và tín hiệu buff_full lúc đó sẽ được tích cực. Buffer chứa bao nhiêu bit thì đầy ? Tùy thuộc vào MODE mà ta sử dụng. MODE sử dụng tuân theo bản sau

cmode[2:0]	MODE	Byte of Buffer
0	SHA3-224	144
1	SHA3-256	136
2	SHA3-384	104
3	SHA3-512	72
4	SHAKE-128	168
5	SHAKE-256	136

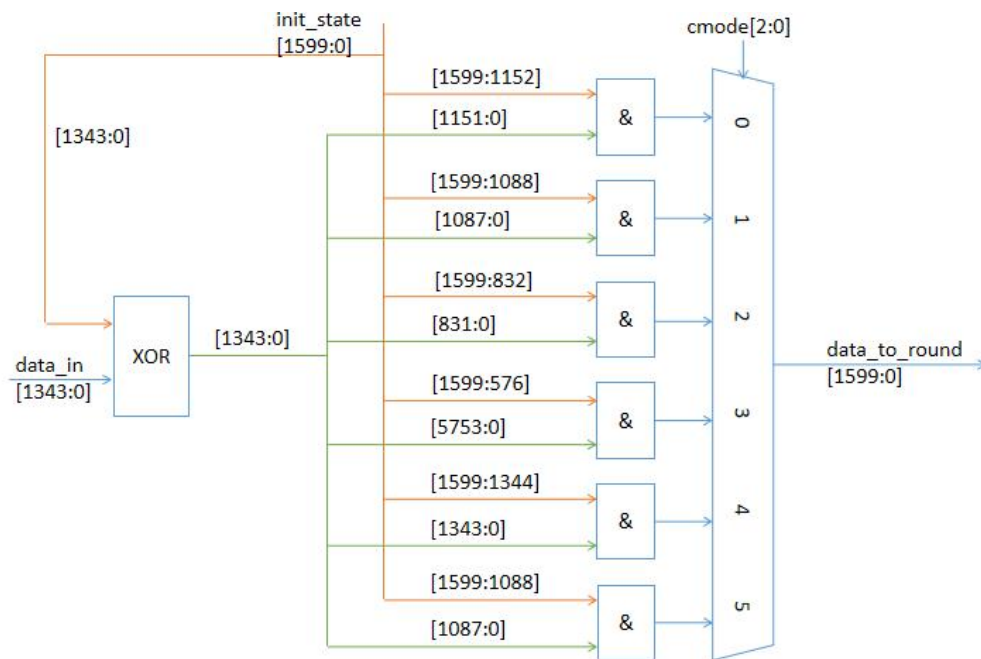
Bảng 4 - 2 Bảng số Byte tương ứng mỗi MODE

4.4.2 Bộ VSX

4.4.2.1 Mô hình chi tiết



Hình 4 - 4 Sơ đồ khối VSX



Hình 4 - 5 Sơ đồ khối chi tiết khối VSX

4.4.2.2 Mô tả chân kết nối

Tín hiệu	Số bit	Ngõ vào	Ngõ ra	Mô tả
init_state	1600	x		Trạng thái khởi tạo dùng để XOR với data_in
data_in	1344	x		Dữ liệu ngõ vào của khối VSX
cmode	3	x		Lựa chọn MODE
en_vsx	1	x		Cho phép VSX_Module hoạt động
data_to_round	1600		x	Chuỗi trạng thái ngõ ra 1600 bit

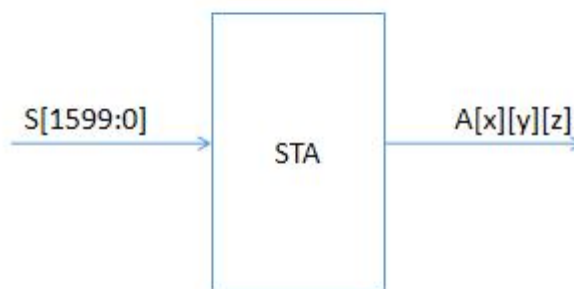
Bảng 4 - 3 Chân kết nối trong khối VSX

4.4.2.3 Mô tả chức năng

VSX_Module (Version Selection and XOR) là khối thực hiện nhiệm vụ lựa chọn MODE sau đó XOR data_in với chuỗi 1600bit init_state. Tùy vào MODE mà số bit XOR với init_state là khác nhau.

4.4.3 Bộ STA

4.4.3.1 Mô hình chi tiết



Hình 4 - 6 Sơ đồ khối STA

4.4.3.2 Mô tả chân kết nối

Tín hiệu	Số bit	Ngõ vào	Ngõ ra	Mô tả
S	1600	x		Chuỗi dữ liệu ngõ vào gồm chuỗi 1600bit
A	1600		x	Mảng trạng thái ngõ ra 3 chiều A[x][y][z] với x = 0..4 ; y = 0..4; z = 0..64

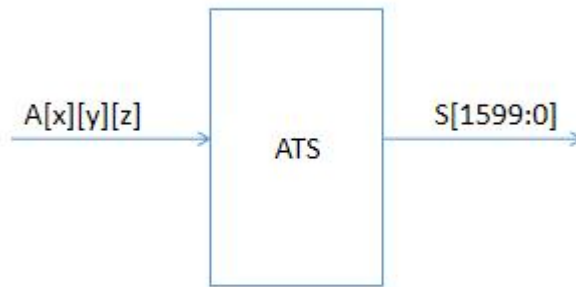
Bảng 4 - 4 Chân kết nối trong khối STA

4.4.3.3 Mô tả chức năng

Module STA dùng để chuyển đổi chuỗi dữ liệu ngõ vào gồm 1600bit thành một mảng trạng thái 3 chiều để thực hiện các phép tính toán hoán vị bên trong giải thuật Keccak.

4.4.4 Bộ ATS

4.4.4.1 Mô hình chi tiết



Hình 4 - 7 Sơ đồ khối ATS

4.4.4.2 Mô tả chân kết nối

Tín hiệu	Số bit	Ngõ vào	Ngõ ra	Mô tả
A	1600	x		Mảng trạng thái ngõ vào 3 chiều A[x][y][z] với $x = 0..4$; $y = 0..4$; $z = 0..64$
S	1600		x	Chuỗi dữ liệu ngõ ra gồm chuỗi 1600bit

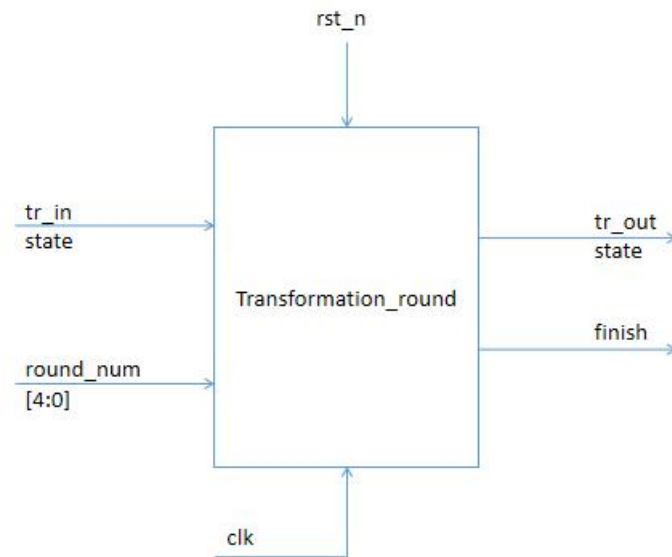
Bảng 4 - 5 Chân kết nối trong khối ATS

4.4.4.3 Mô tả chức năng

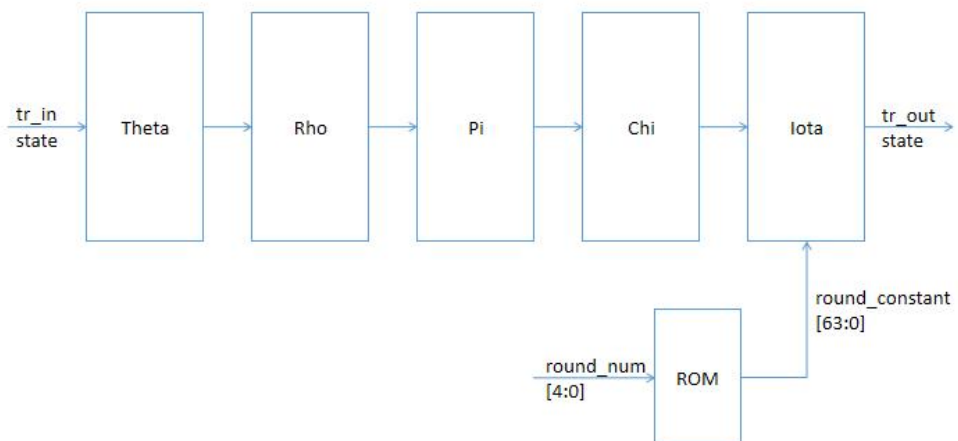
Module ATS dùng để chuyển đổi mảng trạng thái 3 chiều ngõ vào thành chuỗi dữ liệu 1600bit ngõ ra. Sau quá trình tính toán các hàm hoán vị, mảng trạng thái được chuyển về dạng chuỗi bit và xuất ra ngõ ra với số lượng bit phụ thuộc vào loại giải thuật SHA3 mà ta lựa chọn (cmode).

4.4.5 Bộ biến đổi Transformation Round

4.4.5.1 Mô hình chi tiết



Hình 4 - 8 Sơ đồ khối Transformation_round



Hình 4 - 9 Sơ đồ chi tiết khối Transformation_round

4.4.5.2 Mô tả chân kết nối

Tín hiệu	Số bit	Ngõ vào	Ngõ ra	Mô tả
tr_in	1600	x		Ngõ vào dữ liệu của Transformation_round là mảng trạng thái ba chiều có 1600 phần tử (5x5x64)

round_num	5	x		Đếm số thứ tự vòng lặp và tra vào bảng ROM kết quả tương ứng sử dụng trong phép hoán vị Iota
tr_out	1600		x	Ngõ ra dữ liệu của Transformation_round là mảng trạng thái ba chiều có 1600 phần tử (5x5x64)
finish	1		x	Tín hiệu thông báo đã kết thúc tính toán

Bảng 4 - 6 Chân kết nối trong khối Transformation_round

4.4.5.3 Mô tả chức năng

Module Transformation_round thực hiện phép tính toán chính của Keccak, gồm 5 phép hoán vị như trên trong đó riêng phép hoán vị Iota (chỉ tác động lên lane gốc) cần thêm một hằng số vòng được cho sẵn trong bảng ROM và phụ thuộc vào thứ tự vòng hiện tại.

```

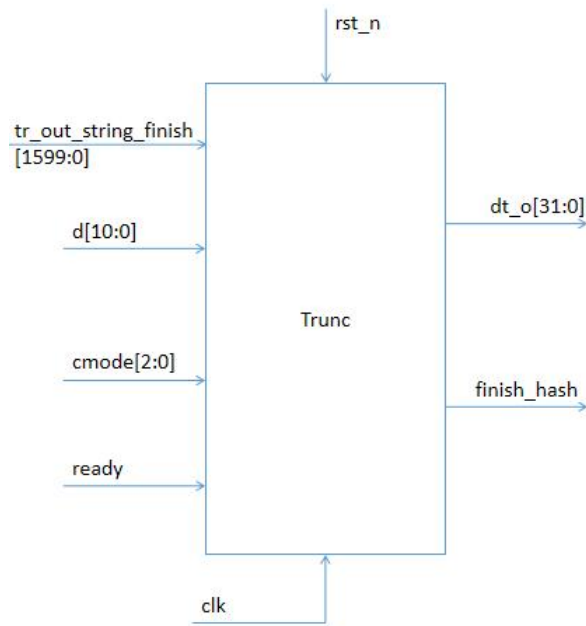
5'b00000 : round_constant = 64'h0000_0000_0000_0001;
5'b00001 : round_constant = 64'h0000_0000_0000_8082;
5'b00010 : round_constant = 64'h8000_0000_0000_808A;
5'b00011 : round_constant = 64'h8000_0000_8000_8000;
5'b00100 : round_constant = 64'h0000_0000_0000_808B;
5'b00101 : round_constant = 64'h0000_0000_8000_0001;
5'b00110 : round_constant = 64'h8000_0000_8000_8081;
5'b00111 : round_constant = 64'h8000_0000_0000_8009;
5'b01000 : round_constant = 64'h0000_0000_0000_008A;
5'b01001 : round_constant = 64'h0000_0000_0000_0088;
5'b01010 : round_constant = 64'h0000_0000_8000_8009;
5'b01011 : round_constant = 64'h0000_0000_8000_000A;
5'b01100 : round_constant = 64'h0000_0000_8000_808B;
5'b01101 : round_constant = 64'h8000_0000_0000_008B;
5'b01110 : round_constant = 64'h8000_0000_0000_8089;
5'b01111 : round_constant = 64'h8000_0000_0000_8003;
5'b10000 : round_constant = 64'h8000_0000_0000_8002;
5'b10001 : round_constant = 64'h8000_0000_0000_0080;
5'b10010 : round_constant = 64'h0000_0000_0000_800A;
5'b10011 : round_constant = 64'h8000_0000_8000_000A;
5'b10100 : round_constant = 64'h8000_0000_8000_8081;
5'b10101 : round_constant = 64'h8000_0000_0000_8080;
5'b10110 : round_constant = 64'h0000_0000_8000_0001;
5'b10111 : round_constant = 64'h8000_0000_8000_8008;

```

Hình 4 - 10 Bảng ROM lưu giữ giá trị round_constant dùng trong Iota

4.4.6 Bộ Trunc

4.4.6.1 Mô hình chi tiết



Hình 4 - 11 Sơ đồ khối Trunc

4.4.6.2 Mô tả chân kết nối

Tín hiệu	Số bit	Ngõ vào	Ngõ ra	Mô tả
Tr_out_string_finish	1600	x		Chuỗi dữ liệu ngõ vào của Trunc
d	11	x		Độ dài output mong muốn (sử dụng cho SHAKE) là bội số 32 có độ dài tối đa là 1024 bit
cmode	3	x		Tín hiệu lựa chọn MODE
ready	1	x		Tín hiệu thông báo đã sẵn sàng cho việc xuất dữ liệu ngõ ra
dt_o	32		x	Chuỗi dữ liệu ngõ ra
finish_hash	1		x	Tín hiệu thông báo kết thúc giải thuật băm

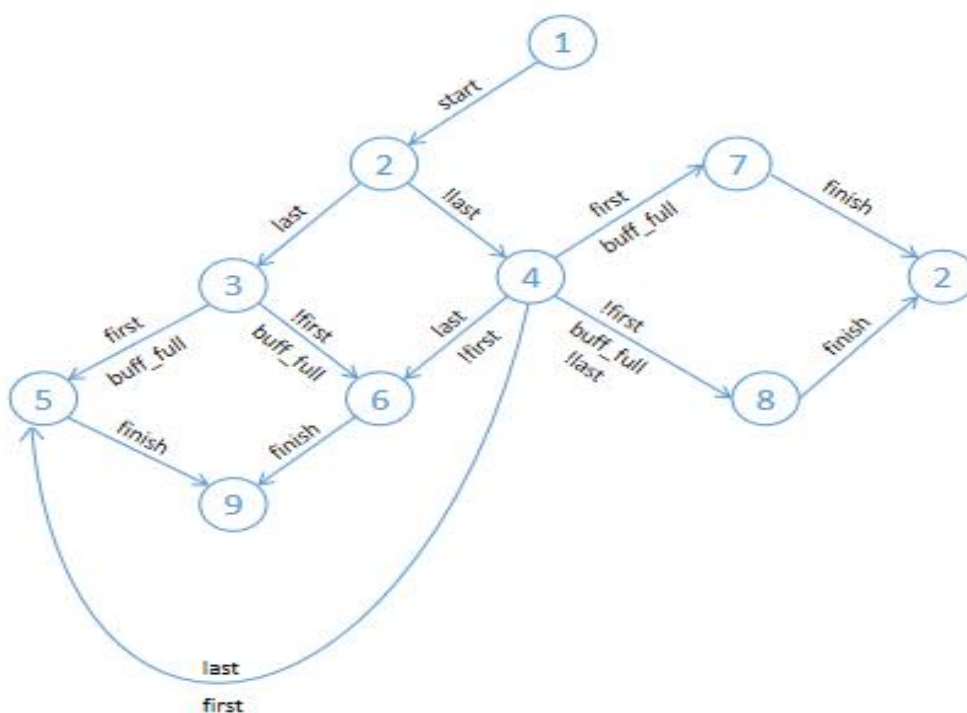
Bảng 4 - 7 Chân kết nối trong khối Trunc

4.4.6.3 Mô tả chức năng

Module Trunc dùng để tách chuỗi dữ liệu là ngõ ra của phép tính toán hàm vòng thành các chuỗi bit tương ứng với từng MODE của hàm băm và đưa giá trị ngõ ra thông qua dt_o gồm 32bit. Khi số bit được đưa ra đã tương ứng với độ dài output của hàm băm thì tín hiệu finish_hash sẽ được tích cực thông báo kết thúc quá trình tính toán hàm băm Keccak.

4.4.7 Khối điều khiển tín hiệu Control

4.4.7.1 Sơ đồ trạng thái của Module Control



Hình 4 - 12 Máy trạng thái khối Control

4.4.7.1 Mô tả chân kết nối

Tín hiệu	Số bit	Ngõ vào	Ngõ ra	Mô tả
start	1	x		Tín hiệu bắt đầu thực hiện tính toán hàm băm, dữ liệu ngõ vào bắt đầu đi vào
last	1	x		Tín hiệu thông báo chuỗi dữ liệu ngõ vào tính toán hàm băm là chuỗi cuối cùng

first	1	x		Tín hiệu thông báo khối tính toán hàm băm là khối đầu tiên
buff_full	1	x		Tín hiệu thông báo bộ đệm Buffer đã đầy
finish	1	x		Tín hiệu thông báo kết thúc tính toán hàm băm
valid	1		x	Tín hiệu cho phép buffer nhận giá trị
nxt_block	1		x	Tín hiệu điều khiển mux nhận giá trị của khối tiếp theo
en_vsx	1		x	Tín hiệu cho phép module VSX thực hiện
en_counter	1		x	Tín hiệu cho phép module counter thực hiện
ready	1		x	Tín hiệu thông báo hoàn thành các bước tính toán

Bảng 4 - 8 Chân kết nối trong khối Control

4.4.7.3 Mô tả các trạng thái của Module Control

THHT	Mô tả
S1	Trạng thái ban đầu
S2	Đã nhận tín hiệu start, tín hiệu valid = 1 để nhận giá trị từ input S7 và S8 khi nhận tín hiệu finish sẽ quay về S2
S3	Nếu đã là khối cuối cùng thì valid = 0 để không nhận giá trị nữa. Đồng thời en_vsx = 1 để thực hiện phép toán

S4	Nếu chưa phải khối cuối cùng thì valid = 1 để nhận tín hiệu đến khi buffer đầy, en_vsx vẫn tiếp tục bật
S5	Khi buffer_full = 1 và khối tính toán đang là khối đầu tiên first = 1 thì nxt_block = 0 (để nhận giá trị 1600'b0 thay vì ngõ ra trước đó)
S6	Khi buffer_full = 1 và khối tính toán ko là khối đầu tiên first = 0 thì nxt_block = 1 (để nhận giá trị là ngõ ra trước đó)
S7	Tương tự S5 nhưng trạng thái trước đó của nó là S4 (chưa là khối cuối)
S8	Tương tự S6 nhưng trạng thái trước đó của nó là S4 (chưa là khối cuối)
S9	Trạng thái kết thúc, khối cuối đã được tính toán xong, ready = 1

Bảng 4 - 9 Các trạng thái trong khối Control

TTHT	Các tín hiệu ngõ ra				
	Valid	Nxt_block	En_vsx	En_counter	ready
S1	0	0	0	0	0
S2	1	0	0	0	0
S3	0	0	1	0	0
S4	1	0	1	0	0
S5	0	0	1	1	0
S6	0	1	1	1	0
S7	0	0	1	1	0

S8	0	1	1	1	0
S9	0	0	0	0	1

Bảng 4 - 10 Các tín hiệu ngõ ra tương ứng trong khối Control

4.4.7.4 Mô tả chức năng

Module Control là module thực hiện nhiệm vụ điều khiển các tín hiệu điều khiển trong phần cứng của giải thuật Keccak. Ở trong thiết kế của em, sử dụng máy trạng thái kiểu Moore để thiết kế lên Module Control này.

5. THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM

Yêu cầu đặt ra cho phần mềm: Thể hiện được các giải thuật sẽ được thực hiện trong phần cứng, hỗ trợ trong việc tính toán, sửa lỗi và kiểm tra các test của phần cứng, tạo file testbench phục vụ cho việc kiểm tra. Phần mềm cũng sẽ hỗ trợ được việc tính toán của tất cả các hàm băm trong giải thuật SHA3 dựa trên thuật toán Keccak.

Em lựa chọn ngôn ngữ Python để thiết kế cho phần mềm để phần mềm đơn giản sử dụng. Giải thuật Keccak trên phần mềm được tham khảo từ nhóm kỹ sư Guido Bertoni, Joan Daemen, Michael Peeters, và Gilles Van Assche. Nhiều thông tin sẽ được đề cập ở website [Keccak Team](#).

Dưới đây là đoạn code ngắn để test cho giải thuật Keccak của nhóm kỹ sư trên. Đầu vào là chuỗi gồm có 32 kí tự mã Hex tương ứng với độ dài theo bit là 128 bit.

Các thông số đưa vào hàm Keccak bao gồm:

```
Keccak((in_length), in_string, r_length, c_length, suffix, out_length, verbose)
```

Trong đó

Thông số	Mô tả
in_length	Chiều dài tính theo bit của chuỗi dữ liệu cần hash

in_string	Chuỗi dữ liệu cần hash
r_length	Chiều dài bitrate (tùy thuộc vào MODE đã mô tả trong bảng 3 - 2)
c_length	Chiều dài capacity (tùy thuộc vào MODE đã mô tả trong bảng 3 - 2)
suffix	Giá trị hậu tố (đã mô tả trong bảng 3 - 2), giá trị mặc định là 0x01 biểu diễn cho giải thuật Keccak, 0x06 là SHA3 và 0x1F là SHAKE
out_length	Chiều dài tính theo bit của giá trị ngõ ra
verbose	Cho phép in chi tiết các bước tính toán ra terminal

Bảng 4 - 11 Các thông số trong hàm Keccak

Dưới đây là kết quả khi thử với chuỗi ngõ vào là :

“00112233445566778899AABBCCDDEEFF”, giải thuật Keccak trả về kết quả như bên dưới.

```

test_Keccak.py > ...
1 import Keccak
2 myKeccak = Keccak.Keccak()
3 #SHA3-224
4 myKeccak.Keccak((128, '00112233445566778899AABBCCDDEEFF'), 1152, 448, 0x01, 224, True)
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
```

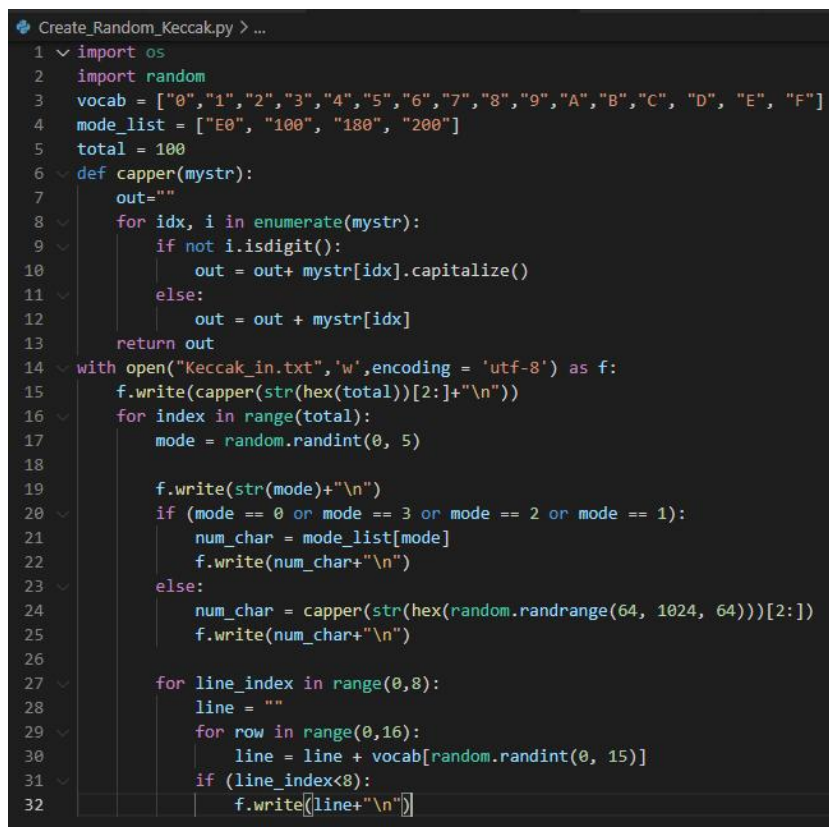
Kết quả này em đã kiểm định với nhiều website tính hàm băm khác ([Online Tools \(emn178.github.io\)](http://emn178.github.io), [New tab \(levantozturk.com\)](http://Newtab(levantozturk.com))) và cũng cho ra kết quả tương tự.

Để phục vụ cho việc tính toán và kiểm tra với số lượng testbench lớn hơn. Em đã tự thiết kế một vài hàm với các chức năng như sau:

- * Tạo file testbench ngẫu nhiên (Create_Random_Keccak.py)
- * Nhận ngõ vào là file Create_Random_Keccak.py, thực hiện tính toán hàm băm và trả kết quả ra file Keccak_out_check.txt.
- * So sánh kết quả khi thực hiện bằng phần mềm (python) và khi thực hiện bằng phần cứng (Modelsim) và in kết quả kiểm tra ra màn hình.

5.1 Create_Random_Keccak.py

File Create_Random_Keccak.py tạo ra ngẫu nhiên một file Keccak_in.txt là dữ liệu vào cho cả thiết kế phần cứng và phần mềm để đảm bảo việc kiểm tra được chính xác.



```

1 import os
2 import random
3 vocab = ["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"]
4 mode_list = ["E0", "100", "180", "200"]
5 total = 100
6 def capper(mystr):
7     out=""
8     for idx, i in enumerate(mystr):
9         if not i.isdigit():
10             out = out+ mystr[idx].capitalize()
11         else:
12             out = out + mystr[idx]
13     return out
14 with open("Keccak_in.txt",'w',encoding = 'utf-8') as f:
15     f.write(capper(str(hex(total))[2:]+\n"))
16     for index in range(total):
17         mode = random.randint(0, 5)
18
19         f.write(str(mode)+"\n")
20         if (mode == 0 or mode == 3 or mode == 2 or mode == 1):
21             num_char = mode_list[mode]
22             f.write(num_char+"\n")
23         else:
24             num_char = capper(str(hex(random.randrange(64, 1024, 64)))[2:])
25             f.write(num_char+"\n")
26
27         for line_index in range(0,8):
28             line = ""
29             for row in range(0,16):
30                 line = line + vocab[random.randint(0, 15)]
31             if (line_index<8):
32                 f.write(line+"\n")

```

Hình 5 - 2 File Create_Random_Keccak.py

Kết quả sau khi chạy file này sẽ xuất ra một file Keccak_in.txt như sau:

- * Dòng đầu tiên là số lượng test (dưới dạng mã HEX) mà ta dùng để tính toán hàm

băm SHA3.

* Dòng tiếp theo là cmode - MODE mà ta sử dụng: 0, 1, 2, 3, 4 và 5 lần lượt tương ứng với SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128 và SHAKE-256.

* Dòng tiếp theo là chiều dài của dữ liệu ngõ ra (tính theo BIT, hiển thị dưới dạng HEX). Ở những giải thuật SHA3-224, SHA3-256, SHA3-384 và SHA3-512 có thể không cần giá trị ngõ vào này tuy nhiên phần cứng của em hỗ trợ thêm hai giải thuật là SHAKE-128 và SHAKE-256 nên cần phải có ngõ vào là số bit ở ngõ ra.

* 8 dòng tiếp theo là dữ liệu cần được tính hàm băm.

Để việc đọc dữ liệu từ phần cứng dễ dàng hơn, em đã quy định cấu trúc file Keccak_in.txt như trên và các khối dữ liệu cần tính hàm băm đều có chiều dài là 512bit.

```
Keccak_in.txt
1 64
2 5
3 100
4 BE4EA607E57208D7
5 F9A73CE185C7A9F8
6 E9250C3AF090DFCC
7 9B621AE06B97E86F
8 F8A80741331C4A69
9 FB58AEF7F3EF87F4
10 1485FF353B9EF883
11 9E90B83ADC61F507
12 5
13 240
14 F9C8F4EB0A949FA5
15 FF374DF6425D7433
16 BFBE35D5540EBBE1
17 54ABAF1013358364
18 02F591D607B06AF2
19 8F4317C09BA8F086
20 6A32011E91523570
21 23C0BCF314DEA6D8
22 3
23 200
24 A077E0FBA21BCB56
25 52BA9C854744A2E6
26 B5FD2AF3B00B60A7
27 4DF6202EE2C5D688
28 8EB764F7C423FA6D
29 5117EAEDA1162DE8
30 940E446A8F6C3E2B
31 9EA9A8506F3D2647
```


Hình 5 - 3 Kết quả khi chạy file Create_Random_Keccak.py

Hiện tại, em thực hiện tạo ra 100 test trong một file Keccak_in.txt (giá trị “total” = 100). Giá trị trong file Keccak_in.txt được đọc vào từ file Hash_Function.py ở phần mềm và Module get_input.py ở phần cứng.

5.2 Hash_Function.py

Hash_Function.py là file chính thực hiện việc đọc dữ liệu từ file đầu vào là Keccak_in.txt, thực hiện việc tính toán dựa trên hàm Keccak có sẵn từ nhóm kỹ sư Keccak sau đó tính ra giá trị hàm băm và in kết quả ra file Keccak_out_check.txt.

```

from email import message
import Keccak
test_len = 512
num_test = 100
myKeccak = Keccak.Keccak()
file = open("Keccak_in.txt", 'r')
Lines = file.readlines()
Lines.pop(0)
with open("Keccak_out_check.txt", 'w', encoding = 'utf-8') as f:
    f.write("SHA3 value: \n")
    for index in range(0, num_test):
        mode = Lines[index*10].strip()
        len_out = int(Lines[index*10 + 1].strip(), 16)
        char = ""
        for sub in range(0, 8):
            char = char + Lines[index*10 + 1 + 1 + sub].strip()
        int_mode = int(mode)
        if (int_mode == 0):
            myKeccak.Keccak((test_len, char), 1152, 448, 0x06, 224, True)
        elif (int_mode == 1):
            myKeccak.Keccak((test_len, char), 1088, 512, 0x06, 256, True)
        elif (int_mode == 2):
            myKeccak.Keccak((test_len, char), 832, 768, 0x06, 384, True)
        elif (int_mode == 3):
            myKeccak.Keccak((test_len, char), 576, 1024, 0x06, 512, True)
        elif (int_mode == 4):
            myKeccak.Keccak((test_len, char), 1344, 256, 0x1F, int(len_out), True)
        elif (int_mode == 5):
            myKeccak.Keccak((test_len, char), 1088, 512, 0x1F, int(len_out), True)

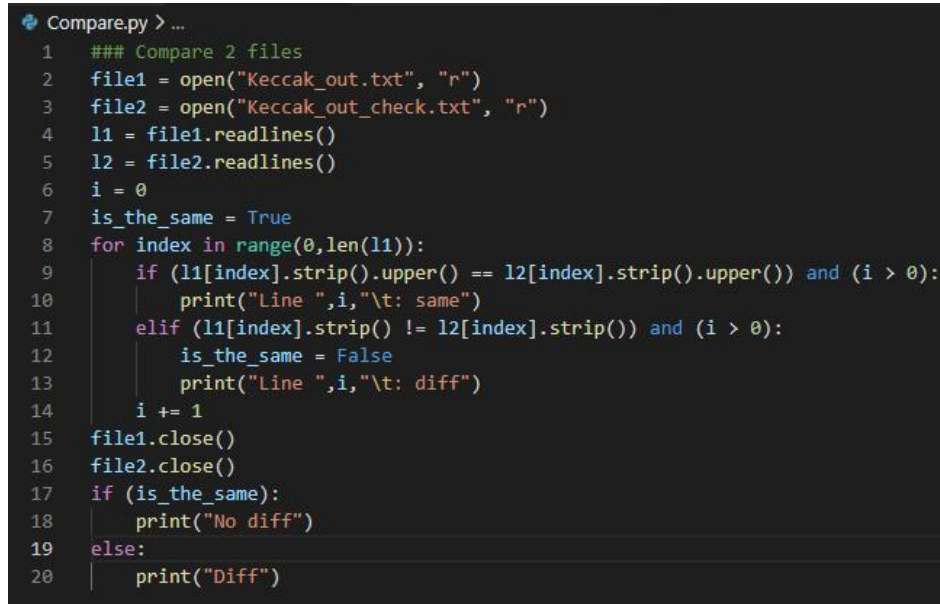
```

Hình 5 - 4 File Hash_Function.py

Dựa trên giá trị MODE đọc được từ file Keccak_in.txt mà ở file này, thực hiện tính toán với các thông số r, c, suffix, ... tương ứng.

5.3 Compare.py

File cuối cùng là Compare.py thực hiện chức năng so sánh giá trị ngõ ra trong file Keccak_out.txt (là ngõ ra của phần cứng) và file Keccak_out_check.txt (là ngõ ra của phần mềm). Kết quả hiện thị ra terminal.



```
Compare.py > ...
1  ### Compare 2 files
2  file1 = open("Keccak_out.txt", "r")
3  file2 = open("Keccak_out_check.txt", "r")
4  l1 = file1.readlines()
5  l2 = file2.readlines()
6  i = 0
7  is_the_same = True
8  for index in range(0, len(l1)):
9      if (l1[index].strip().upper() == l2[index].strip().upper()) and (i > 0):
10         print("Line ", i, "\t: same")
11     elif (l1[index].strip() != l2[index].strip()) and (i > 0):
12         is_the_same = False
13         print("Line ", i, "\t: diff")
14     i += 1
15 file1.close()
16 file2.close()
17 if (is_the_same):
18     print("No diff")
19 else:
20     print("Diff")
```

Hình 5 - 5 File Compare.py

6. KẾT QUẢ THỰC HIỆN

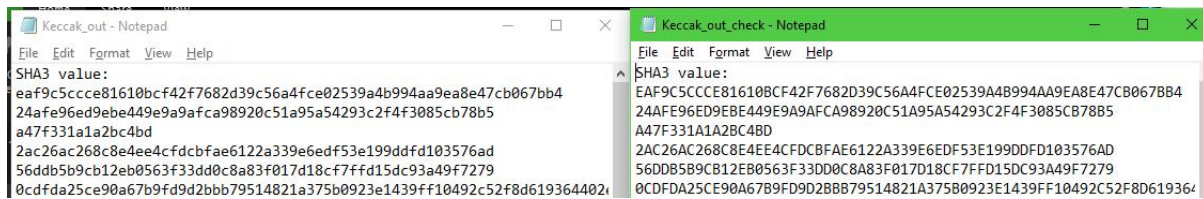
6.1 Cách thức đo đạc, thử nghiệm

Thiết kế phần cứng sau khi hoàn tất được thử nghiệm bằng chương trình mô phỏng phần cứng và phần mềm. Em sử dụng Modelsim để thiết kế và mô phỏng phần cứng, Python để thiết kế và mô phỏng phần mềm. Sau đó tổng hợp kết quả và đối chiếu để kiểm tra tính đúng đắn.

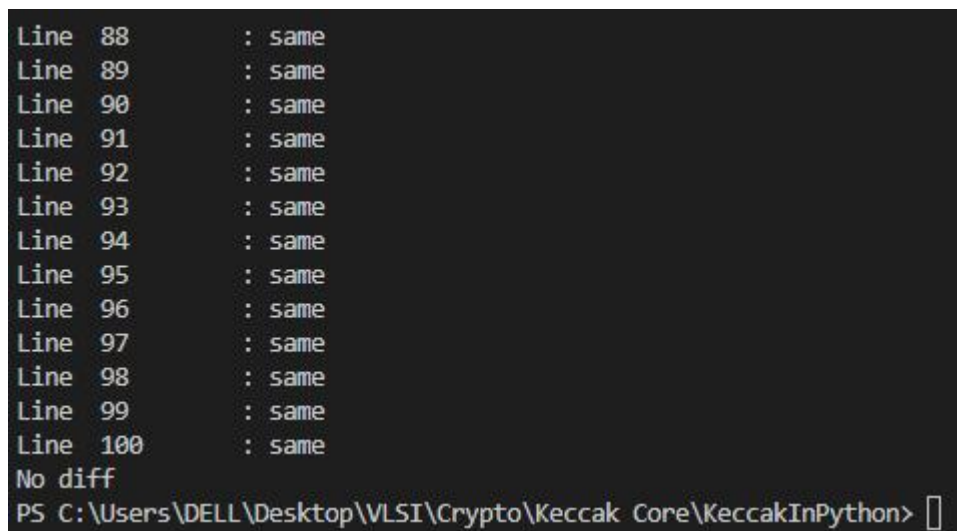
Để đo đạc các thông số về tài nguyên phần cứng, em sử dụng Quartus Prime 21.1 và thông số của kit FPGA 5CGXFC7C7F23C8 dòng Cyclone V của Intel.

6.2 Số liệu đo đạc

Kết quả mô phỏng cho thấy thiết kế chạy chính xác các MODE đã đề ra của SHA3 và SHAKE.



Hình 6 - 1 Kết quả tính toán hàm băm bằng phần cứng và phần mềm



Hình 6 - 2 Kết quả khi so sánh hai kết quả

Slow 1100mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	70.1 MHz	70.1 MHz	clk	

Hình 6 - 3 Tần số của hệ thống

Flow Summary	
<<Filter>>	
Revision Name	top_module
Top-level Entity Name	top_module
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	9,453 / 56,480 (17 %)
Total registers	4675
Total pins	116 / 268 (43 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Hình 6 - 4 Kết quả tổng hợp phân cứng

7. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

8. TÀI LIỆU THAM KHẢO

- [1] G.-P. M. G. T. George S. Athanasiou, “HIGH THROUGHPUT PIPELINED FPGA IMPLEMENTATION OF THE NEW SHA-3”.
- [2] J. H.-Y. S. S. A. C. Ming Ming Wong, “A New High Throughput and Area Efficient SHA-3 Implementation”.
- [3] D.-e.-S. k. A. A. Alia Arshad, “Compact Implementation of SHA3-512 on FPGA”.

- [4] H. M. B. B. M. M. Fatma Kahri, “High Speed FPGA Implementation of Cryptographic KECCAK Hash Function Crypto-Processor”.
- [5] F. E. M. A. F. Assad, “An optimal hardware implementation of the KECCAK hash function on virtex-5 FPGA”.
- [6] M. A. M. R. Kashif Latif, “Efficient Hardware Implementations and Hardware Performance Evaluation of”.
- [7] P. K. N. S. C. K. George Provelengios, “FPGA-Based Design Approaches of Keccak Hash Function”.
- [8] S. M. Atefeh Gholipour, “High-Speed Implementation of the KECCAK Hash Function on FPGA”.
- [9] R. C. Magnus Sundal, “Efficient FPGA Implementation of the SHA-3 Hash Function”.
- [10] K. I. T. M. T. K. Thuong Nguyen Dat, “Implementation of high speed hash function Keccak on GPU”.
- [11] J. D. M. P. G. V. A. Guido Berton, “Cryptographic sponge functions”.
- [12] J. D. M. P. G. V. A. Guido Berton, “The Keccak SHA-3 submission”.
- [13] J. D. M. P. G. V. A. Guido Berton, “Keccak sponge function family main document”.
- [14] J. D. M. P. G. V. A. Guido Berton, “The Keccak reference”.
- [15] N. I. o. S. a. Technology, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions”.

