ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ
----------o0o----------

ĐỒ ÁN 2

THIẾT KẾ PHẦN CỨNG HÀM BĂM SHA3-256 BẰNG THUẬT TOÁN KECCAK

GVHD: ThS. Nguyễn Tuấn Hùng
SVTH: Hoàng Minh Chiến
MSSV: 2112932

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2024

LỜI CẢM ƠN

Trong suốt thời gian làm đồ án này em đã nhận được rất nhiều sự hỗ trợ giúp đỡ và đóng góp ý kiến từ các thầy cô và bạn bè.

Em xin chân thành cảm ơn thầy Nguyễn Tuấn Hùng đã dành thời gian để hướng dẫn em trong suốt quá trình thực hiện đồ án. Nhờ những kiến thức và sự chỉ bảo của thầy, em đã học hỏi được rất nhiều kiến thức, kinh nghiệm bổ ích để hoàn thiện đồ án.

Em cũng xin gửi lời cảm ơn tới các thầy cô của bộ môn Điện tử - Trường Đại học Bách khoa đã luôn nhiệt tình với sinh viên.

Lời cảm ơn cuối cùng tới tất cả thầy cô và bạn bẽ đã đồng hành cùng em tới tận bây giờ!

Tp. Hồ Chí Minh, ngày 24 tháng 12 năm 2024.

Sinh viên

TÓM TẮT ĐỒ ÁN

Đồ án này trình bày về việc thiết kế và mô phỏng thuật toán băm SHA3-256 dựa trên cấu trúc Keccak bằng ngôn ngữ mô tả phần cứng Verilog. Thuật toán SHA3-256, thuộc chuẩn băm mật mã SHA-3, được xây dựng trên nguyên tắc xử lý dữ liệu bằng ma trận trạng thái $5 \times 5 \times 64$. Quá trình triển khai bao gồm các giai đoạn: nén dữ liệu đầu vào, chuyển đổi trạng thái qua các bước chi tiết như theta, rho, pi, chi, và iota, từ đó tạo ra giá trị băm đầu ra với độ dài 256-bit.

Trong đồ án, thiết kế được thực hiện theo phương pháp mô đun hóa, chia thành các khối xử lý riêng biệt để đảm bảo tính linh hoạt và dễ dàng kiểm tra. Kết quả mô phỏng trên ModelSim và Xcelium xác nhận thiết kế hoạt động chính xác theo yêu cầu của thuật toán SHA3-256. Tuy nhiên, đồ án mới dừng lại ở bước mô phỏng, chưa thực hiện tổng hợp (synthesis) và triển khai trên FPGA để đánh giá hiệu năng thực tế.

Đồ án cung cấp một nền tảng cơ bản cho các nghiên cứu và ứng dụng trong lĩnh vực bảo mật dữ liệu, đặc biệt là tích hợp thuật toán SHA3-256 vào các hệ thống phần cứng. Trong tương lai, việc hoàn thiện bước tổng hợp, tối ưu hóa tài nguyên và tốc độ xử lý bằng song song hóa sẽ được tiến hành để phát triển thiết kế cho các ứng dụng trong lĩnh vực mật mã học.

MỤC LỤC

DANH SÁCH HÌNH MINH HỌA

DANH SÁCH BẢNG SỐ LIỆU

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 này 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 Nhiệm vụ đề tài

Nội dung 1: Tìm hiểu nguyên lý, lý thuyết về hàm băm.

Nội dung 2: Tìm hiểu lý thuyết và nguyên lý về hàm bọt biển (Sponge Function).

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

Nội dung 4: Thiết kế hàm băm SHA3-256 sử dụng thuật toán Keccak theo chuẩn FIPS 202.

Nội dung 5 : Tìm hiểu lý thuyết và cách thức song song hóa SHA3 (Đồ án tốt nghiệp).

Nội dung 6: Thiết kế song song hóa cho tất cả các SHA3 đầu ra phổ biến là 224-bit, 256-bit, 384-bit, 512-bit, và hai giải thuật mở rộng SHAKE-128, SHAKE-256 (Đồ án tốt nghiệp).

2. LÝ THUYẾT

2.1 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.

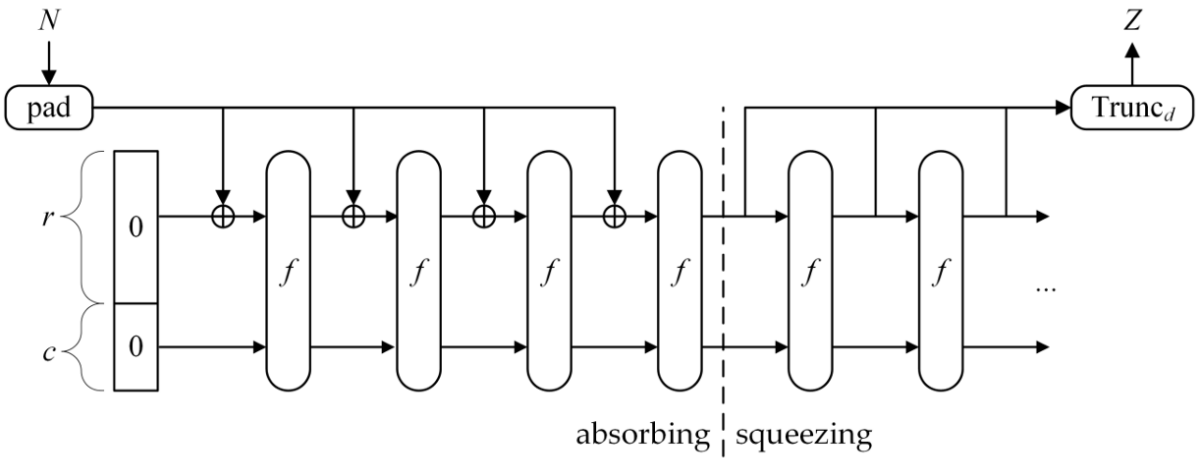
Từ đoạn mã "Hello" ban đầu sau quá trình băm thì trở thành một chuỗi số mà chúng ta nhìn vào sẽ không thể hiểu được. Điều này có vẻ giống với việc mã hóa (encrypt) nhưng khác biệt ở chỗ, ở mã hóa ta có thể giải mã được từ chuỗi số khó hiểu đó còn ở hàm băm thì không như tính chất đã trình bày ở trên. 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.

2.2 Hàm bọt biển "Sponge Function"

Hàm bọt biển (Sponge Function) hay cấu trúc bọt biển (Sponge Construction) là một thuật toán nhận độ vào có độ dài tùy ý và tạo ra đầu ra có độ dài bất kỳ. Cấu trúc bọt biển có vai trò quan trọng trong thiết kế Keccak và hàm băm SHA-3.

Cấu trúc này thực hiện bằng cách xử lý dữ liệu đầu vào qua các giai đoạn "hấp thụ" (absorb), "vắt ra" (squeez), với ba thành phần chính:

- Hàm chuyển đổi trạng thái f:



Hình 2-1 Cấu trúc bọt biển

  - Đây là một hàm có độ dài cố định, ký hiệu ở đây là b, còn được gọi là độ rộng (width), có thể hoán vị một chuỗi $b - bit$ sang chuỗi $b - bit$ khác.
  - Phép hoán vị $f: \{0,1\}^b \rightarrow \{1,0\}^b$ để biến đổi trạng thái.
- Tốc độ r (bitrate) và dung lượng c (capacity)
  - Tốc độ r là số bit của dữ liệu đầu vào hoặc đầu ra được xử lý trong mỗi bước, r là một số nguyên dương, nhỏ hơn độ rộng b.
  - Dung lượng c được xác định bằng $c = b - r$, đại diện cho phần bảo mật của hàm, $b = r + c$.
- Hàm đệm "padding"
  - Ký hiệu "pad", là một quy tắc đảm bảo dữ liệu đầu vào $N - bit$ được mở rộng sao cho độ dài của nó là bội số của r. Điều này cho phép chia dữ liệu thành các khối $r - bit$.
  - Trong thiết kế hàm băm SHA-3, "padding" được mô tả và thực hiện bởi thuật toán Keccak, để đảm bảo tính nhất quán.

Hàm được tạo ra từ các thành phần này được gọi là hàm sponge (sponge function), ký hiệu là $SPONGE[f, pad, r]$.

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 "vắt" (squeezing).

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 hàm bọt biển

Giải thuật hàm bọt biển $SPONGE[f, pad, r](N, d)$ có thể được trình bày như sau:

Đầu vào: chuỗi N, số nguyên không âm d.

Đầu ra: chuỗi Z sao cho $len(Z) = d$.

Các bước thực hiện:

1. Đặt $P = N||pad(r, len(N))$.
2. Đặt $n = len(P)/r$.
3. Đặt $c = b - r$.
4. Đặt $P_0, P_1, \dots, P_{n-1}$ là dãy duy nhất bao gồm các chuỗi có độ dài r sao cho $P = P_0||P_1|| \dots ||P_{n-1}$.
5. Đặt $S = 0^b$ (trạng thái ban đầu là chuỗi bit 0 với độ dài b).
6. Với i từ 0 đến $n - 1$: Cập nhập $S = f(S \oplus (P_i||0^c))$.
7. Đặt Z là chuỗi rỗng.
8. Cập nhập $Z = Z||Trunc_r(S)$ (lấy r bit đầu tiên của S nối vào Z) : $Trunc_r(S)$ là cắt $r - bit$ trong chuỗi S.
9. Nếu $d \leq |Z|$, trả về $Trunc_d(Z)$, nếu không thì tiếp tục.
10. Cập nhập $S = f(S)$, sau đó quay lại bước 8.

Lưu ý rằng đầu vào d xác định số lượng bit mà ngõ ra thuật toán trả về, không ảnh hưởng đến giá trị của chúng. Về nguyên tắc, đầu ra có thể được xem như một chuỗi vô hạn, nhưng trong thực tế, việc tính toán sẽ dừng lại sau khi tạo ra số lượng bit đầu ra mong muốn.

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 "vắt". Hình sau chỉ ra giải thuật cho giai đoạn "hấp thụ".

Algorithm 3 The absorbing function ABSORB$[f, r]$

Require: $r < b$

Interface: $s = \text{absorb}(P)$ with $P \in \mathbb{Z}_{2^r}^*$ 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 Giải thuật hàm hấp thụ

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. Hình dưới chỉ ra giải thuật cho giai đoạn "vắt".

Algorithm 4 The squeezing function 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 $|Z|_r r < \ell$ **do**
  $s = f(s)$
  $Z = Z || \lfloor s \rfloor_r$
end while
return $\lfloor Z \rfloor_\ell$

Hình 2-4 Giải thuật hàm vắt ra

Tóm lại, hàm bọt biển trong thiết kế hàm băm SHA-3 hoạt động bằng cách “hấp thụ” (absorb) các dữ liệu đầu vào qua các khối $r - bit$ (bitrate), chuyển đổi trạng thái bằng một

hàm f với độ rộng cố định b, và "vắt ra" (squeez) chuỗi đầu ra có độ dài tùy ý. Nó sử dụng thêm quy tắc "padding" để đảm bảo dữ liệu đầu vào được căn chỉnh đúng độ rộng.

2.3 Quy tắc pad10*1

Quy tắc "padding", ký hiệu "pad" là một hàm tạo ra phần điền thêm, tức là một chuỗi có độ dài thích hợp để nối vào chuỗi khác. Trong nguyên lý hoạt động của sponge function, đầu vào có độ dài bất kỳ và ngõ ra sẽ có độ dài nhất định. Vì đầu vào bất kỳ nên ta phải tạo ra phần điền thêm để thêm những bit bổ sung, tạo thành khối có đủ độ dài để xử lý. Quy tắc pad trong giải thuật Keccak là "multi-rate padding", hay quy tắc pad10*1. Được thực hiện bằng cách bổ sung bit 1 vào trước MSB chuỗi ngõ vào, sau đó là một chuỗi các bit 0, và kết thúc bằng bit 1. Chuỗi bit 0 sẽ phụ thuộc vào độ dài ban đầu của ngõ vào. Sau khi padding sẽ tạo thành chuỗi đầu vào mới có độ dài là bội số $r - bit$. Quy tắc pad10*1 bổ sung tối thiểu là 2 bit và tối đa là $b + 1$ bit. Thuật toán quy tắc $pad10 * 1(x, m)$ như sau:

Đầu vào: Số nguyên dương x, số nguyên không âm m.

Đầu ra: Chuỗi P sao cho $m + len(P)$ là bội số dương của x.

Các bước thực hiện:

1. Đặt $j = (-m - 2)\ mod\ x$.
2. Trả về $P = 1||0^j||1$.

Số bit 0 có thể lược bỏ hoặc lặp lại khi cần thiết để tạo ra một chuỗi đầu ra có độ dài mong muốn

2.4 Hàm hoán vị Keccak

Hàm băm thiết kế theo thuật toán Keccak sử dụng cấu trúc bọt biển như đã trình bày, 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.

Phép hoán vị $KECCAK - p[b, n_r]$ được xác định với hai tham số:

- Độ dài cố định của các chuỗi được hoán vị, gọi là độ rộng của phép hoán vị (width) được ký hiện là b. Trong đó b thuộc tập $\{25,50,100,200,400,800,1600\}$.
- Số lần lặp lại của một phép hoán vị, gọi là một vòng (round), ký hiệu là n_r, với bất kỳ số nguyên dương nào cho n_r.

Một vòng của phép hoán vị KECCAK-p, được ký hiệu là Rnd, bao gồm một chuỗi năm phép biến đổi, được gọi là các bước ánh xạ (step mappings). Hàm vòng của nó có dạng theo tiêu chuẩn FIPS 202 như sau:

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

Trong đó, tầng tuyến tính của nó là kết hợp bởi một số thành phần tuyến tính như phép biển 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 sau.

Phép hoán vị được xác định thông qua một mảng các giá trị của $b - bit$, được cập nhật lặp lại trong quá trình thực hiện, gọi là trạng thái (state). Ban đầu, trạng thái này được khởi tạo bằng các giá trị đầu vào của phép hoán vị. Các đặc tả trong Tiêu chuẩn này chứa hai đại lượng khác liên quan đến b là $b/25$ và $\log_2(b/25)$, lần lượt được ký hiệu là w và l. Bảy giá trị có thể có của các biến này, được định nghĩa cho các phép hoán vị KECCAK-p, được trình bày trong hình dưới đây.

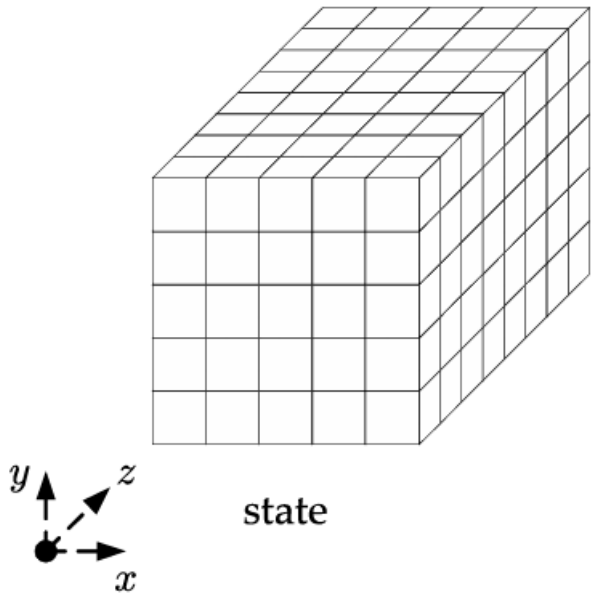
b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
ℓ	0	1	2	3	4	5	6

Hình 2-5 Độ rộng của Keccak-p và các đại lượng liên quan

Để thuận tiện thì ta biểu diễn trạng thái đầu vào và đầu ra của phép hoán vị dưới dạng các chuỗi $b - bit$, và biểu diễn các trạng thái đầu vào, đầu ra của các phép ánh xạ (step mapping) dưới dạng mảng $[5,5,w]$ các bit. Nếu S là ký hiệu biểu thị cho một chuỗi đại diện trạng thái thì các bit của nó được đánh số từ 0 đến $b - 1$. Do đó:

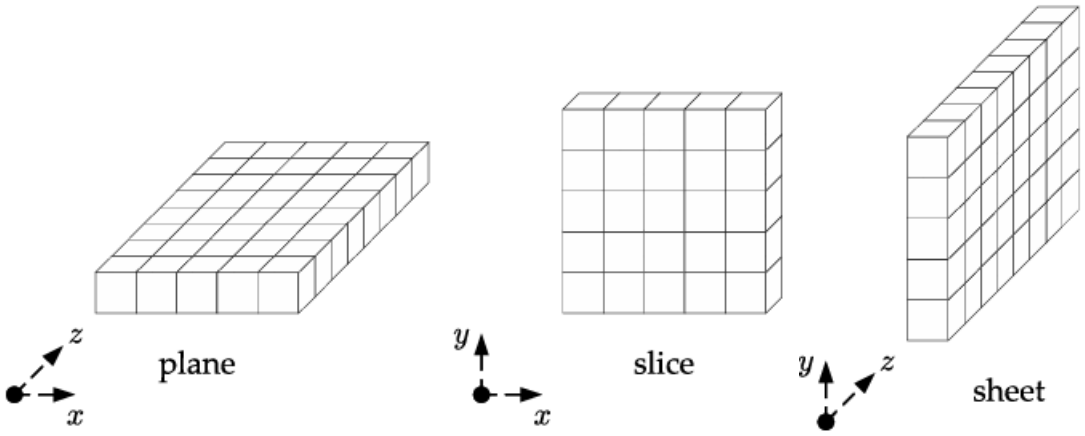
$$S = S[0] \,||\, S[1] \,||\, \dots \,||\, S[b-2] || \, S[b-1]$$

Và nếu A biểu diễn cho một mảng $[5,5,w]$ các bit đại diện cho trạng thái, thì các chỉ số của nó là bộ ba số nguyễn (x,y,z) thỏa mãn, $0 \leq x < 5, 0 \leq y < 5$ và $0 \leq z < w$. Bit tương ứng với (x,y,z) được ký hiệu $A[x,y,z]$. Một mảng trạng thái là một biểu diễn của trạng thái thông qua mảng ba chiều được đánh số theo quy tắc vừa nêu.

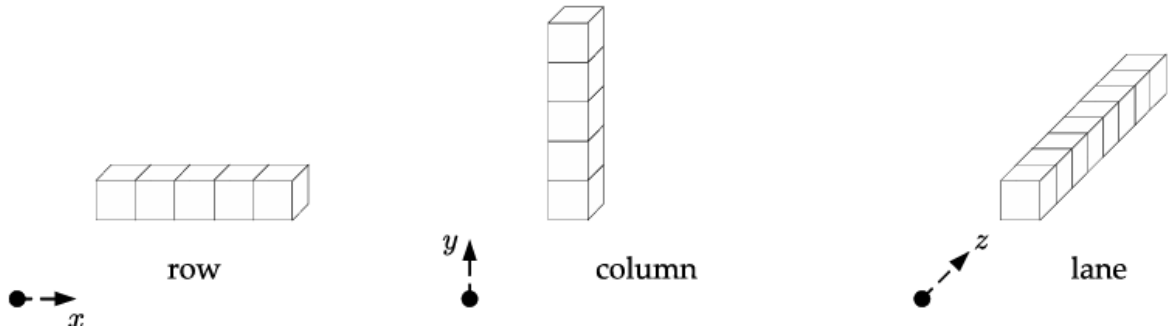


Hình 2-6 Mảng trạng thái 3 chiều

Hình trên là ví dụ của một mảng trạng thái 3 chiều với trường hợp $b = 200$, do đó $w = 8$. Và các mảng con hai chiều của mảng ba chiều trên sẽ được gọi là $sheets$, $planes$, $slices$, trong khi các mảng con một chiều được gọi là $rows, columns, lanes$. Hình ảnh minh họa các mảng con này được thể hiện:



Hình 2-7 Mảng con hai chiều của mảng trạng thái



Hình 2-8 Mảng con một chiều của mảng trạng thái

2.4.1 Chuyển đổi chuỗi trạng thái sang mảng trạng thái

Coi S là một chuỗi $b - bit$ đại diện cho trạng thái của phép hoán vị $Keccak - p[b, n_r]$. Mảng trạng thái tương ứng ký hiệu là A, được tính toán và định nghĩa như sau:

Với mọi (x, y, z) thỏa $0 \le x < 5, 0 \le y < 5, 0 \le z < w$. Mảng A được viết như sau:

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

Ví dụ, nếu $b = 1600$, do đó $w = 64$, nên:

$$\begin{array}{lll}
\mathbf{A}[0,0,0] = S[0] & \mathbf{A}[1,0,0] = S[64] & \mathbf{A}[4,0,0] = S[256] \\
\mathbf{A}[0,0,1] = S[1] & \mathbf{A}[1,0,1] = S[65] & \mathbf{A}[4,0,1] = S[257] \\
\mathbf{A}[0,0,2] = S[2] & \mathbf{A}[1,0,2] = S[66] & \mathbf{A}[4,0,2] = S[258] \\
\vdots & \vdots \quad \ldots & \vdots \\
\mathbf{A}[0,0,61] = S[61] & \mathbf{A}[1,0,61] = S[125] & \mathbf{A}[4,0,61] = S[317] \\
\mathbf{A}[0,0,62] = S[62] & \mathbf{A}[1,0,62] = S[126] & \mathbf{A}[4,0,62] = S[318] \\
\mathbf{A}[0,0,63] = S[63] & \mathbf{A}[1,0,63] = S[127] & \mathbf{A}[4,0,63] = S[319]
\end{array}$$

….

2.4.2 Chuyển đổi mảng trạng thái sang chuỗi trạng thái

Mảng trạng thái được ký hiệu là A, chuỗi trạng thái S có thể được xây dựng lại từ $lanes, planes$ của mảng A.

Với mỗi cặp số nguyên (i, j) sao cho $0 \le i < 5$ và $0 \le j < 5$, định nghĩa được $Lane(i, j)$ như sau:

$$Lane(i, j) = A[i, j, 0] \,||\, A[i, j, 1] \,||\, A[i, j, 2] \,||\, \ldots \,||\, A[i, j, w - 2] \,||\, A[i, j, w - 1].$$

Ví dụ, nếu $b = 1600$, do đó $w = 64$, nên:

$$Lane\,(0,0) = \mathbf{A}[0,0,0] \,||\, \mathbf{A}[0,0,1] \,||\, \mathbf{A}[0,0,2] \,||\, \dots \,||\, \mathbf{A}[0,0,62] \,||\, \mathbf{A}[0,0,63]$$
$$Lane\,(1,0) = \mathbf{A}[1,0,0] \,||\, \mathbf{A}[1,0,1] \,||\, \mathbf{A}[1,0,2] \,||\, \dots \,||\, \mathbf{A}[1,0,62] \,||\, \mathbf{A}[1,0,63]$$
$$Lane\,(2,0) = \mathbf{A}[2,0,0] \,||\, \mathbf{A}[2,0,1] \,||\, \mathbf{A}[2,0,2] \,||\, \dots \,||\, \mathbf{A}[2,0,62] \,||\, \mathbf{A}[2,0,63]$$

…..

Và với mỗi số nguyên j với $0 \le j < 5$, ta định nghĩa được $Plane(j)$ như sau:

$$Plane(j) = Lane(0,j) \,||\, Lane(1,j) \,||\, Lane(2,j) \,||\, Lane(3,j) \,||\, Lane(4,j).$$

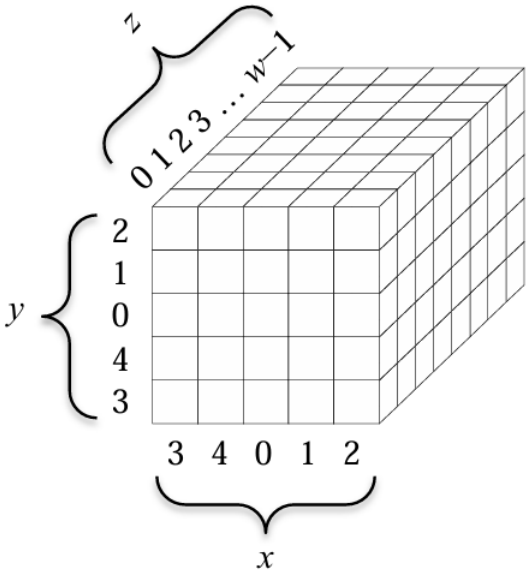
Sau đó:

$$S = Plane(0) \,||\, Plane(1) \,||\, Plane(2) \,||\, Plane(3) \,||\, Plane(4).$$

Cũng cùng ví dụ trên khi $b = 1600$, nên $w = 64$, vậy:

$$\begin{aligned} S = \quad & \mathbf{A}[0,0,0] \,||\, \mathbf{A}[0,0,1] \,||\, \mathbf{A}[0,0,2] \,||\, \dots \,||\, \mathbf{A}[0,0,62] \,||\, \mathbf{A}[0,0,63] \\ & ||\, \mathbf{A}[1,0,0] \,||\, \mathbf{A}[1,0,1] \,||\, \mathbf{A}[1,0,2] \,||\, \dots \,||\, \mathbf{A}[1,0,62] \,||\, \mathbf{A}[1,0,63] \\ & ||\, \mathbf{A}[2,0,0] \,||\, \mathbf{A}[2,0,1] \,||\, \mathbf{A}[2,0,2] \,||\, \dots \,||\, \mathbf{A}[2,0,62] \,||\, \mathbf{A}[2,0,63] \\ & ||\, \mathbf{A}[3,0,0] \,||\, \mathbf{A}[3,0,1] \,||\, \mathbf{A}[3,0,2] \,||\, \dots \,||\, \mathbf{A}[3,0,62] \,||\, \mathbf{A}[3,0,63] \\ & ||\, \mathbf{A}[4,0,0] \,||\, \mathbf{A}[4,0,1] \,||\, \mathbf{A}[4,0,2] \,||\, \dots \,||\, \mathbf{A}[4,0,62] \,||\, \mathbf{A}[4,0,63] \\ & \\ & ||\, \mathbf{A}[0,1,0] \,||\, \mathbf{A}[0,1,1] \,||\, \mathbf{A}[0,1,2] \,||\, \dots \,||\, \mathbf{A}[0,1,62] \,||\, \mathbf{A}[0,1,63] \\ & ||\, \mathbf{A}[1,1,0] \,||\, \mathbf{A}[1,1,1] \,||\, \mathbf{A}[1,1,2] \,||\, \dots \,||\, \mathbf{A}[1,1,62] \,||\, \mathbf{A}[1,1,63] \\ & ||\, \mathbf{A}[2,1,0] \,||\, \mathbf{A}[2,1,1] \,||\, \mathbf{A}[2,1,2] \,||\, \dots \,||\, \mathbf{A}[2,1,62] \,||\, \mathbf{A}[2,1,63] \\ & ||\, \mathbf{A}[3,1,0] \,||\, \mathbf{A}[3,1,1] \,||\, \mathbf{A}[3,1,2] \,||\, \dots \,||\, \mathbf{A}[3,1,62] \,||\, \mathbf{A}[3,1,63] \\ & ||\, \mathbf{A}[4,1,0] \,||\, \mathbf{A}[4,1,1] \,||\, \mathbf{A}[4,1,2] \,||\, \dots \,||\, \mathbf{A}[4,1,62] \,||\, \mathbf{A}[4,1,63] \\ & \\ & \vdots \\ & \\ & ||\, \mathbf{A}[0,4,0] \,||\, \mathbf{A}[0,4,1] \,||\, \mathbf{A}[0,4,2] \,||\, \dots \,||\, \mathbf{A}[0,4,62] \,||\, \mathbf{A}[0,4,63] \\ & ||\, \mathbf{A}[1,4,0] \,||\, \mathbf{A}[1,4,1] \,||\, \mathbf{A}[1,4,2] \,||\, \dots \,||\, \mathbf{A}[1,4,62] \,||\, \mathbf{A}[1,4,63] \\ & ||\, \mathbf{A}[2,4,0] \,||\, \mathbf{A}[2,4,1] \,||\, \mathbf{A}[2,4,2] \,||\, \dots \,||\, \mathbf{A}[2,4,62] \,||\, \mathbf{A}[2,4,63] \\ & ||\, \mathbf{A}[3,4,0] \,||\, \mathbf{A}[3,4,1] \,||\, \mathbf{A}[3,4,2] \,||\, \dots \,||\, \mathbf{A}[3,4,62] \,||\, \mathbf{A}[3,4,63] \\ & ||\, \mathbf{A}[4,4,0] \,||\, \mathbf{A}[4,4,1] \,||\, \mathbf{A}[4,4,2] \,||\, \dots \,||\, \mathbf{A}[4,4,62] \,||\, \mathbf{A}[4,4,63] \end{aligned}$$

Trong các sơ đồ trạng thái thì quy ước $lane$ tương ứng với tọa độ $(x,y) = (0,0)$ được biểu diễn ở trung tâm của các $slices$. Hình sau mô tả đầy đủ của các tọa độ (x,y,z) trong trạng thái.



Hình 2-9 Các chiều của mảng trạng thái

2.5 Step Mapping

Như đã đề cập thì 5 phép ánh xạ bước sẽ tạo thành một vòng của phép hoán vị $Keccak - p[b, n_r]$, các phép toán ánh xạ được ký hiệu là Theta (θ), Pi (π), Rho (ρ), Iota (ι), Chi (χ). Thuật toán cho mỗi phép ánh xạ bước (step mapping) nhận một mảng trảng thái, ký hiệu A, làm đầu vào và sau khi thực hiện phép toán thì sẽ trả ra một mảng trạng thái mới là A'. Đặc biệt đối với phép toán ánh xạ Iota (ι), sẽ có một đầu vào thứ hai là một số nguyên, số nguyên này được gọi là chỉ số vòng (round index), ký hiệu i_r. Và các phép toán ánh xạ bước khác sẽ không phụ thuộc vào chỉ số vòng.

2.5.1 Biến đổi Theta (θ)

Thuật toán biến đổi theta θ(A)

Đầu vào: Mảng trạng thái A.

Đầu ra: Mảng trạng thái A'.

Các bước biến đổi:

1. Với tất cả các cặp (x, z) sao cho $0 \leq x < 5$ và $0 \leq z < w$, ta có:

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

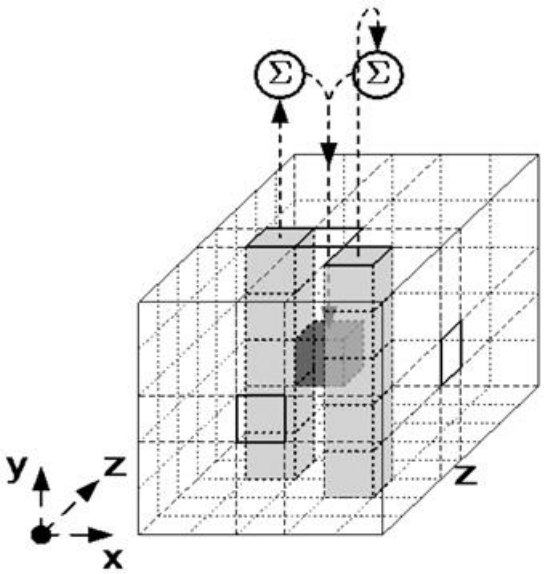
2. Với tất cả các cặp (x, z) sao cho $0 \leq x < 5$ và $0 \leq z < w$, ta có:

$$D[x, z] = C[(x - 1) \; mod \; 5, z] \oplus C[(x + 1) \; mod \; 6, (z - 1) \; mod \; w.$$

3. Với tất cả các bộ ba (x, y, z) với $0 \leq x < 5, 0 \leq y < 5$, và $0 \leq z < w$, ta có

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

Tác dụng của phép ánh xạ θ là thực hiện phép XOR trên từng bit trong trạng thái với tính chẵn lẻ của hai cột trong mảng, cụ thể đối với bit $A[x_0, y_0, z_0]$, tọa độ x của một trong các cột là $(x_0 - 1)\ mod\ 5$, với tọa độ z là z_0. Tọa độ x của cột còn lại là $(x_0 + 1)\ mod\ 5$, với tọa độ z là $(z_0 - 1)\ mod\ w$.



Hình 2-10 Minh họa phép biến đổi theta với w = 8

2.5.2 Biến đổi Rho (ρ)

Thuật toán biến đổi ρ:

Đầu vào: Mảng trạng thái A.

Đầu ra: Mảng trạng thái A'.

Các bước thực hiện:

1. Với tất cả z sao cho $0 \leq z < w$, ta có $A'[0,0,z] = A[0,0,z]$.
2. Có $(x, y) = (1,0)$.
3. Với t từ 0 đến 23
   a. Với tất cả z sao cho

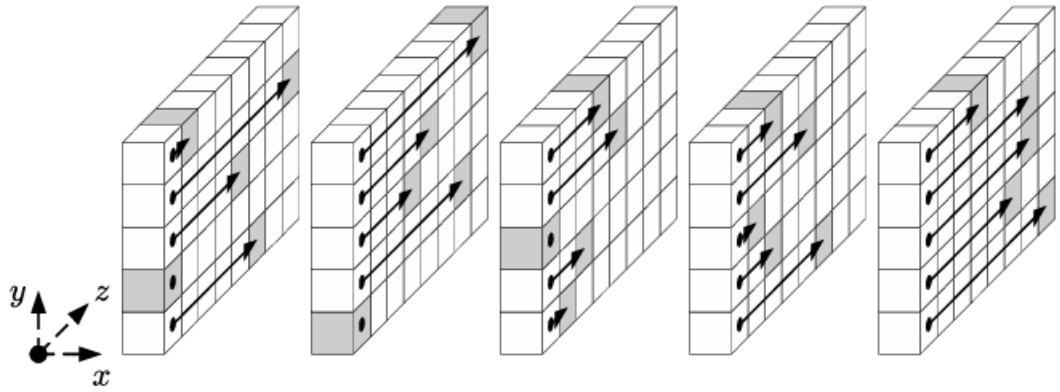
      $0 \leq z < w,\ A'[x, y, z] = A[x, y, (z - (t + 1)(t + 2)/2)\ mod\ w]$;

   b. Có $(x, y) = (y, (2x + 3y)\ mod\ 5)$.
4. Trả kết quả A'.

Tác động của phép biển đổi ρ là xoay các bit của $lane$ theo một độ dài, gọi là $offset$, phụ thuộc vào tọa độ cố định x và y của hàng đó. Tương tự đối với mỗi bit trong $lane$, tọa độ z được thay đổi bằng cách cộng thêm $offset$, theo modulo kích thước của $lane$. Bảng dưới đây là ví dụ của các lệnh được tính toán ở bước 3a với $w = 8$.

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	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 1 Thông số biến đổi rho với w = 8



Hình 2-11 Minh họa phép biến đổi rho với w = 8

Trong đề tài đồ án này, hàm băm SHA3-256 được xây dựng trên $Keccak = f[1600]$ với $w = 64$, nên ta có thể tính toán và xây dựng được các giá trị từ bước 3a cho $w = 64$.

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	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 Thông số biến đổi rho với w = 64

2.5.3 Biến đổi Pi (π)

Thuật toán biến đổi π:

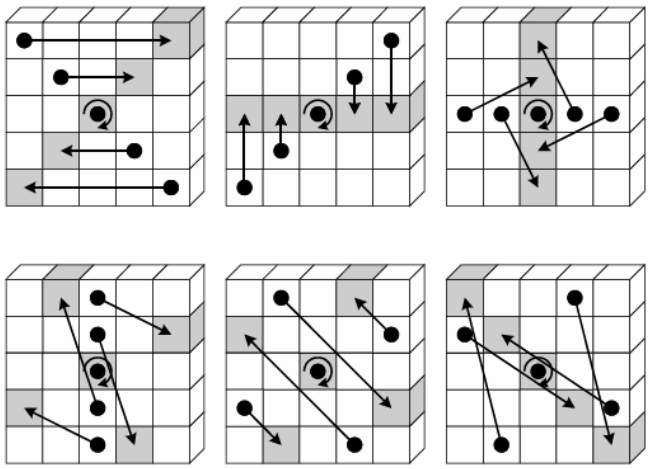
Đầu vào: Mảng trạng thái A.

Đầu ra: Mảng trạng thái A'.

Các bước thực hiện:

1. Với tất cả các bộ ba (x, y, z) sao cho $0 \leq x < 5, 0 \leq y < 5$ và $0 \leq z < w$, có:
$$A'[x, y, z] = A[(x + 3y)\ mod\ 5, x, z].$$
2. Trả kết quả A'.

Phép biến đổi π là thay đổi vị trí của các $lane$, như được minh họa các $slices$ trong hình dưới đây. Việc thay đổi vị trí này là giống nhau cho toàn bộ w $slices$ trong mảng trạng thái.



Hình 2-12 Minh họa phép biến đổi pi với w = 8

2.5.4 Biến đổi Chi (χ)

Thuật toán biến đổi χ:

Đầu vào: Mảng trạng thái A.

Đầu ra: Mảng trạng thái A'.

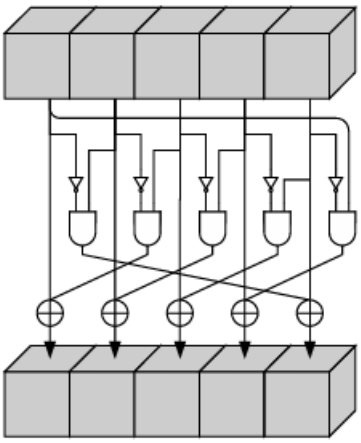
Các bước thực hiện:

1. Với tất cả các bộ ba (x, y, z) sao cho $0 \leq x < 5, 0 \leq y < 5$ và $0 \leq z < w$, có:

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

2. Trả kết quả A'.

Dấu chấm ở phía bên phải của phép gán ở bước 1 chỉ ra phép nhân nguyên, tương đương với phép toán Boolean "AND". Thuật toán χ, là thực hiện phép XOR giữa mỗi bit với một hàm phi tuyến của hai bit khác nhau trong cùng $lane$ của nó.



2

Hình 2-13 Minh họa biến đổi chi với w = 8

2.5.5 Biến đổi Iota (ι)

Như đã đề cập ở trên thì biến đổi ι được tham số hóa bởi chỉ số vòng (round index). 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 hàm $rc(t)$:

Đầu vào: số nguyên t.

Đầu ra: bit $rc(t)$.

Các bước thực hiện:

1. Nếu $t \ mod \ 255 = 0$, trả kết quả là 1.
2. Cho $R = 10000000$.
3. Với i từ 1 tới $t \ mod \ 255$, có:
   a. $R = 0 \ || \ R$;
   b. $R[0] = R[0] \oplus R[8]$;
   c. $R[4] = R[4] \oplus R[8]$;
   d. $R[5] = R[5] \oplus R[8]$;
   e. $R[6] = R[6] \oplus R[8]$;
   f. $R = Trunc_8[R]$;
4. Trả kết quả $R[0]$.

Thuật toán hàm $\iota(A, i_r)$:

Đầu vào: Mảng trạng thái A; chỉ số vòng (round index) i_r.

Đầu ra: Mảng trạng thái A'.

Các bước thực hiện:

1. Với tất cả bộ ba (x, y, z) sao cho $0 \leq x < 5, 0 \leq y < 5$ và $0 \leq z < w$, có $A'[x, y, z] = A[x, y, z]$.
2. $RC = 0^w$.
3. Với j từ 0 tới l, $RC = [2^j - 1] = rc(j + 7i_r)$.
4. Với tất cả z sao cho $0 \leq z < w, A'[0,0,z] = A'[0,0,z] \oplus RC[z]$.
5. Trả kết quả A'.

Ảnh hưởng của phép biến đổi ι là thay đổi một số bit của $lane(0,0)$ phụ thuộc theo chỉ số vòng i_r. 24 $lane$ còn lại không bị ảnh hưởng bởi phép biến đổi ι.

2.5.6 Hàm hoán vị $Keccak - p[b, n_r]$

Là một hàm biến đổi một chuỗi bit 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 (số vòng lặp lại trong biến đổi). $Keccak - p$ sử dụng hàm vòng với 5 phép biến đổi đã đề cập ở trên:

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

Thuật toán hàm hoán vị $Keccak - p[b, n_r](S)$:

Đầu vào: chuỗi bit S có độ dài b; số vòng lặp n_r.

Đầu ra: chuỗi bit S' có độ dài b.

Các bước thực hiện:

1. Biến đổi chuỗi S thành mảng trạng thái ba chiều A.
2. Với i_r từ $12 + 2l - n_r$ đến $12 + 2l - 1$, đặt $A = Rnd(A, i_r)$.
3. Biến đổi mảng trạng thái ba chiều A thành chuỗi S' có độ dài b.
4. Trả kết quả S'.

2.5.7 Hàm hoán vị $Keccak - f[b]$

$Keccak - f$ chính là $Keccak - p[b, n_r]$ trong trường hợp $n_r = 12 + 2l$:

$$Keccak - f[b] = Keccak - p[b, 12 + 2l]$$

Do đó phép hoán vị $Keccak - p[1600,24]$ là cơ sở của các hàm SHA-3, tương đương $Keccak - f[1600]$.

2.5.8 Hàm hoán vị $Keccak[c]$

Như đã định nghĩa thì $Keccak$ là họ các hàm sponge sử dụng hoán vị $Keccak - p[b, 12 + 2l]$ làm cơ sở và quy tắc cho đệm pad10*1.

Hàm hoán vị này được tham số hóa bằng cách lựa chọn bất kỳ r (bitrate) và c (capacity) sao cho $r + c$ nằm trong tập $\{25,50,100,200,400,800,1600\}$. Trong trường hợp $b = 1600$ bộ đệm sử dụng là Pad10*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.

$$Keccak[c] = SPONGE[Keccak - f[1600], pad10*1, 1600 - c]$$

$Keccak[c]$ nhận đầu vào là N và d trong đó N là chuỗi bit đầu vào và d là độ dài của đầu ra.

$$Keccak[c](N.d) = SPONGE[Keccak - f[1600], pad10*1, 1600 - c](N, d)$$

2.5.9 Thiết kế hàm băm SHA3-256

Với một đoạn thông điệp ký hiệu làM, hàm băm mã hóa chính SHA3-256 được định nghĩa bằng phép hoán vị $Keccak - [c]$ như sau:

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

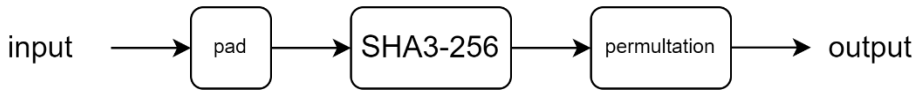
Instance	Output Size d	Rate r-Block Size	Capacity c
SHA3-224(M)	224	1152	448
SHA3-256(M)	256	1088	512
SHA3-384(M)	384	832	768
SHA3-512(M)	512	576	1024

Bảng 3 Thông số cho các giải thuật SHA-3

3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

3.1 Cách thức thực hiện thiết kế

Từ những lý thuyết về hàm băm SHA3 đã trình bày ở mục 2, cấu trúc cơ bản của hàm băm sử dụng giải thuật Keccak có thể trình bày bằng sơ đồ khối tổng quát sau:



Hình 3-1 Sơ đồ cơ bản hàm băm SHA3-256

3.2 Yêu cầu thiết kế

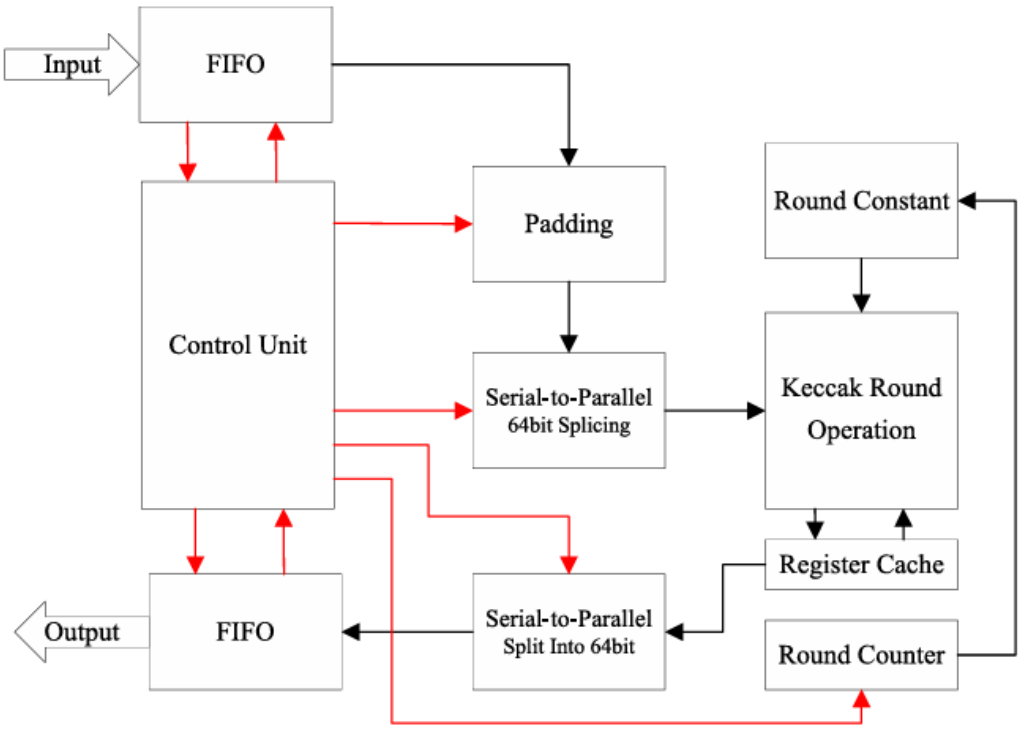
Thiết kế hàm băm SHA3-256 bằng giải thuật Keccak.

Thiết kế có thể tổng hợp và tối ưu tốc độ, tiết kiệm tài nguyên trên FPGA Altera Cyclone V De10-Standard.

3.3 Phân tích thiết kế

Vì ngõ vào của SHA3-256 là một chuỗi bit bất kỳ, để thực hiện thiết kế trên phần cứng FPGA ta không thể đọc cùng lúc lượng lớn bit ngõ vào. Do đó để tối ưu hóa phần cứng của SHA-3, FIFO được sử dụng như một bộ nhớ đệm (cache), dữ liệu cần tính toán được lưu trữ trước trong FIFO, dưới dạng từng khối 64-bit và kết quả cũng được lưu trong FIFO. Thêm đó, phương pháp hợp nhất (merge processing) được áp dụng, và năm bước $\theta, \rho, \pi, \chi, \iota$ được hợp nhất thành một vòng, sao cho một phép toán lặp lại được thực hiện trong một chu kỳ đồng hồ. Toàn bộ phép toán Keccak chỉ cần 24 chu kỳ, giúp nâng cao hiệu suất hoạt động. Tiếp theo, đối với giao diện dữ liệu I/O, chiều rộng 64 bit được sử dụng làm chuẩn cho việc nhập và xuất dữ liệu đồng nhất, và khi thực hiện $Keccak - f[1600]$, dữ liệu được ghép lại thành 1600 bit và sau khi tính toán hoàn tất, được chia thành 64 bit để giảm thiểu chi phí tài nguyên và cải thiện băng thông.

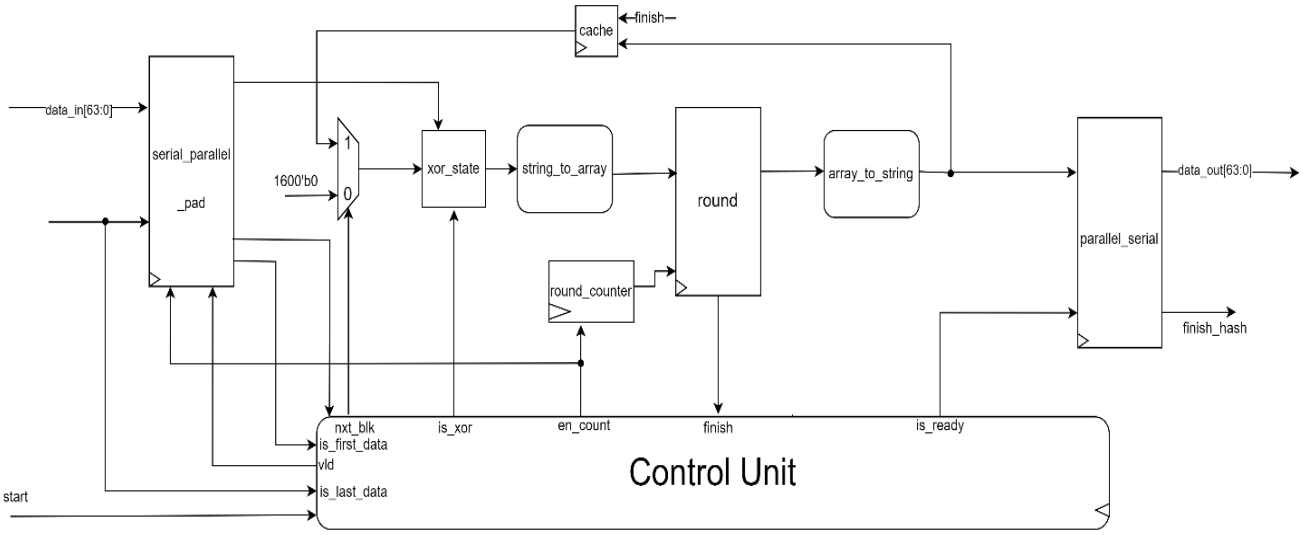
Do đó, toàn bộ cấu trúc phần cứng SHA-3 có thể được chia thành các thành phần chính: một module điều khiển (control unit), một module pad (padding unit), một module xử lý hoán vị Keccak (Keccak round operation unit) và một module chuyển đổi nối tiếp sang song song (serial-to-parallel conversion unit).



Hình 3-2 Sơ đồ khối tổng quát SHA3-256

Trong đó các khối chính thực hiện chức năng cơ bản như:

1. Control Unit Module: Kiểm soát và điều khiển toàn bộ quá trình thực thi của hàm băm SHA-3 gồm: nhập và xuất dữ liệu, thêm padding, chuyển đổi dữ liệu, thực hiện các phép tính toán vòng.
2. Serial-Parallel-Pad Module: Thực hiện chuyển đổi dữ liệu đầu vào và padding cho đầu vào, với SHA3-256 thêm padding với 0x06 và 0x80.
3. Parallel-Serial Module: Tách dữ liệu kết quả ra thành các khối 64-bit để xuất kết quả vào FIFO.
4. Keccak-round: Hàm Rnd thực hiện tính toán Keccak, với 24 lần lặp.

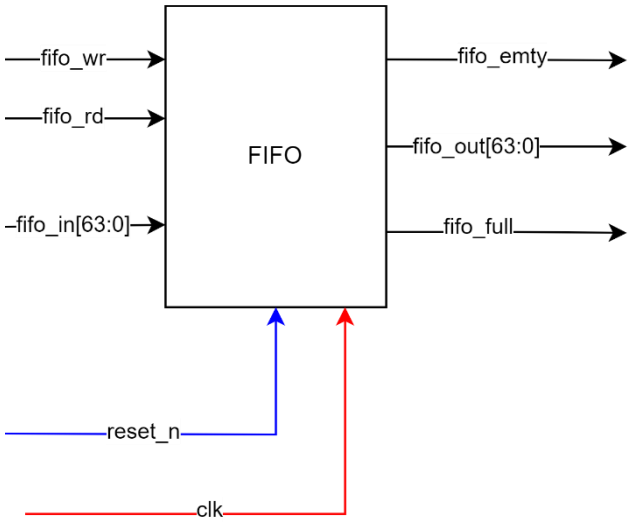


Hình 3-3 Sơ đồ khối của thiết kế

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

3.4.1 Khối lưu trữ dữ liệu đầu vào và đầu ra FIFO

3.4.1.1 Sơ đồ khối chi tiết



Hình 3-4 Sơ đồ khối khối FIFO

3.4.1.2 Mô tả chân I/O

Tín hiệu	I/O	Số bit	Mô tả
fifo_wr	Input	1	Cho phép ghi dữ liệu vào FIFO
fifo_rd	Input	1	Cho phép đọc dữ liệu từ FIFO
fifo_in	Input	64	Dữ liệu đầu vào FIFO
fifo_empty	Output	1	Tích cực nếu FIFO đang rỗng
fifo_full	Output	1	Tích cực nếu FIFO đã đầy
fifo_out	Output	64	Dữ liệu đọc ra từ FIFO

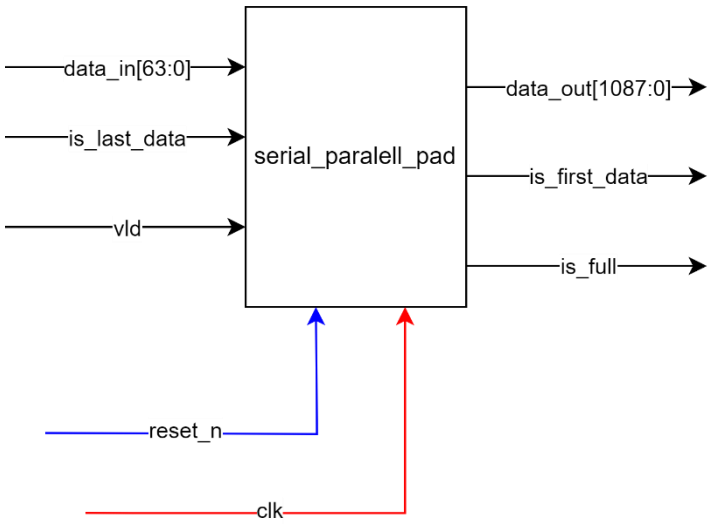
Bảng 4 Chân kết nối của khối FIFO

3.4.1.3 Mô tả chi tiết chức năng

Như đã đề cập, FIFO đóng vai trò như bộ nhớ đệm để lưu từng khối 64-bit đầu vào và đầu ra. Điều này có thể giúp tổng hợp thiết kế trên phần cứng của FPGA. FIFO có độ sâu là 16 và độ rộng 64, tương ứng với việc lưu trữ 1024-bit. Do kích thước khối đầu vào của SHA3-256 tuân thủ HMAC nên chỉ cần 136 bytes để lưu trữ (tương ứng 1088-bit). Các tín hiệu *fifo_full* hay *fifo_empty* sẽ cho biết FIFO đang đầy hay rỗng.

3.4.2 Khối Serial-Parallel-Pad

3.4.2.1 Sơ đồ khối chi tiết



Hình 3-5 Sơ đồ khối khối serial-parallel-pad

3.4.2.2 Mô tả chân I/O

Tín hiệu	I/O	Số bit	Mô tả
is_last_data	Input	1	1 nếu khối data 64-bit đưa vào là khối cuối cùng
vld	Input	1	Cho phép nhận dữ liệu từ FIFO
data_in	Input	64	Dữ liệu đầu vào
data_out	Output	1088	Dữ liệu ngõ ra tương ứng SHA3-256
is_full	Output	1	1 nếu khối đã đầy dữ liệu vào
is_first_data	Output	1	Cho biết khối dữ liệu 64-bit đầu tiên được đưa vào xử lý

Bảng 5 Chân kết nối của khối serial-parallel-pad

3.4.2.3 Mô tả chi tiết chức năng

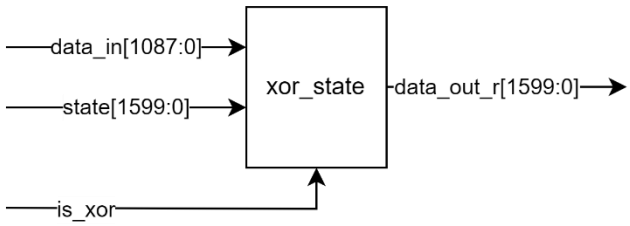
Module đóng vai trò đệm tất cả dữ liệu đưa từ FIFO qua và thực hiện padding với quy tắc pad10*1 như đã đề cập ở phần lý thuyết. Tín hiệu *is_last_data* cho biết khối dữ liệu 64-bit đưa vào đã là khối cuối cùng cần xử lý, lúc này module dừng đọc dữ liệu và sẽ gom tất cả tín hiệu và thực hiện padding, tín hiệu *is_full* cũng được tích cực sau *is_last_data*. Tín hiệu *is_first_data* sẽ cho biết khối ngõ vào 64-bit đầu tiên để thực hiện phép XOR, sẽ đề cập ở mục 3.4.3.3. Ngõ ra là một số 1088-bit tương ứng với 136 bytes của SHA3-256 đã nói đến.

Hash Function	SHA3-224	SHA3-256	SHA3-384	SHA3-512
Block Size (bytes)	144	136	104	72

Bảng 6 Kích thước ngõ vào theo HMAC

3.4.3 Khối XOR trạng thái

3.4.3.1 Sơ đồ khối chi tiết



Hình 3-6 Sơ đồ khối khối xor_state

3.4.3.2 Mô tả chân I/O

Tín hiệu	I/O	Số bit	Mô tả
data_in	Input	1088	Dữ liệu đầu vào xor_state
state	Input	1600	Dữ liệu trạng thái để XOR với *data_in*
is_xor	Input	1	Tín hiệu cho phép hoạt động
data_out_r	Output	1600	Dữ liệu trạng thái sau khi XOR để xử lý Round

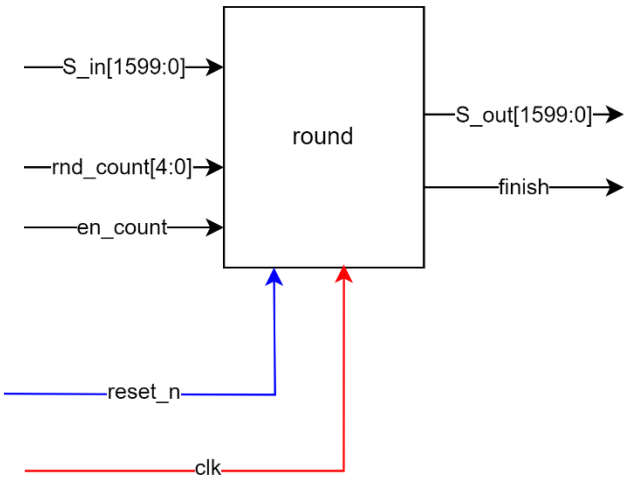
Bảng 7 Chân kết nối của khối xor_state

3.4.3.3 Mô tả chi tiết chức năng

Chuỗi dữ liệu 1088-bit từ serial-parallel-pad sẽ dược đưa vào *data_in* khối *xor_state*. Theo lý thuyết bọt biển, tại vòng lặp đầu tiên thì $r - bit$ sẽ kết hợp với $c - bit$ 0 của chuỗi trạng thái ban đầu, là 1600 bit 0. Những vòng lặp tiếp theo, $c - bit$ 0 sẽ được thay bằng chuỗi trạng thái đã tính toán từ vòng lặp trước đó, nên tín hiệu *is_first_data* từ serial-parallel-pad sẽ cho biết điều này. Và để tạo thành chuỗi 1600-bit trước khi đưa vào xử lý Keccak, tùy vào khối dữ liệu là đầu tiên hay không mà *state* sẽ quyết định chuỗi bit để XOR.

3.4.4 Khối Round

3.4.4.1 Sơ đồ khối chi tiết



Hình 3-7 Sơ đồ khối khối round

3.4.4.2Mô tả chân I/O

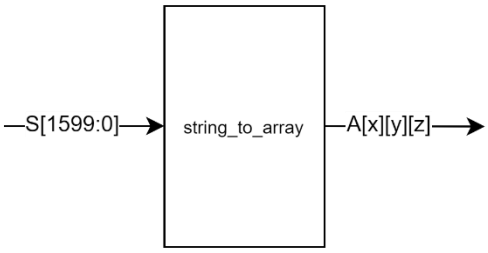
Tín hiệu	I/O	Số bit	Mô tả
S_in	Input	1600	Dữ liệu đầu vào thực hiện tính toán Keccak
rnd_count	Input	5	Bộ đếm số thứ tự thực hiện vòng lặp
S_out	Output	1600	Chuỗi ngõ ra sau khi tính toán Keccak
finish	Output	1	Cho biết đã kết thúc tính toán

Bảng 8 Chân kết nối của khối round

3.4.4.3 Mô tả chi tiết chức năng

Module *round* thực hiện tính toán chính của thuật toán Keccak, gồm 5 phép hoán vị đã đề cập là theta, rho, pi, chi, iota. Trước khi thực hiện tính toán, ta cần phải đưa chuỗi ngõ vào 1600-bit thành mảng trạng thái 3 chiều $5 \times 5 \times 64$. Và kết quả cuối cùng cũng phải chuyển về chuỗi trạng thái 1600-bit để xuất ra kết quả. Module *round constant* được dùng riêng cho phép biến đổi Iota, với 24 vòng lặp ra cần 24 constant để tác động lên *lane* gốc của phép biến đổi Iota. Module *round counter* sẽ đảm nhiệm việc đếm này. Tín hiệu *en_count* sẽ điều khiển module *round counter*.

3.4.5 Khối String to Array và Array to String

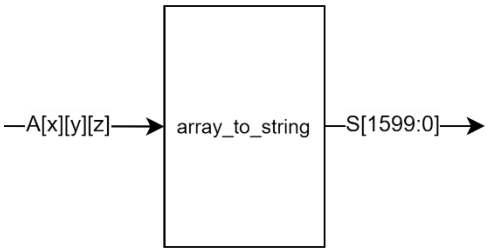


Hình 3-8 Sơ đồ khối khối string to array

Tín hiệu	I/O	Số bit	Mô tả
S	Input	1600	Chuỗi dữ liệu đầu vào
A	Output	1600	Mảng trạng thái 3 chiều ngõ ra với $A[x][y][z]$ sao cho $0 \leq x < 5, 0 \leq y < 5, 0 \leq z \leq 64$

Bảng 9 Chân kết nối khối string to array

Module String to Array thực hiện chuyển đổi chuỗi trạng thái 1600-bit thành mảng trạng thái ba chiều $A[x][y][z]$ trước khi tính toán bằng Keccak. Module Array to String thực hiện chuyển đổi mảng 3 chiều $A[x][y][z]$ thành chuỗi trạng thái 1600-bit để đưa ra kết quả hàm băm SHA3-256.

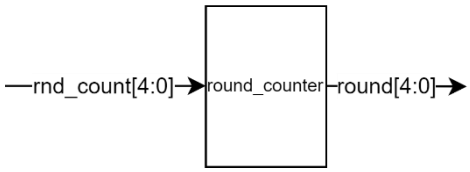


Hình 3-9 Sơ đồ khối khối array to string

Tín hiệu	I/O	Số bit	Mô tả
A	Input	1600	Mảng trạng thái 3 chiều ngõ ra với $A[x][y][z]$ sao cho $0 \leq x < 5, 0 \leq y <$
S	Output	1600	Chuỗi trạng thái ngõ ra

Bảng 10 Chân kết nối khối array to string

3.4.6 Khối Round Counter



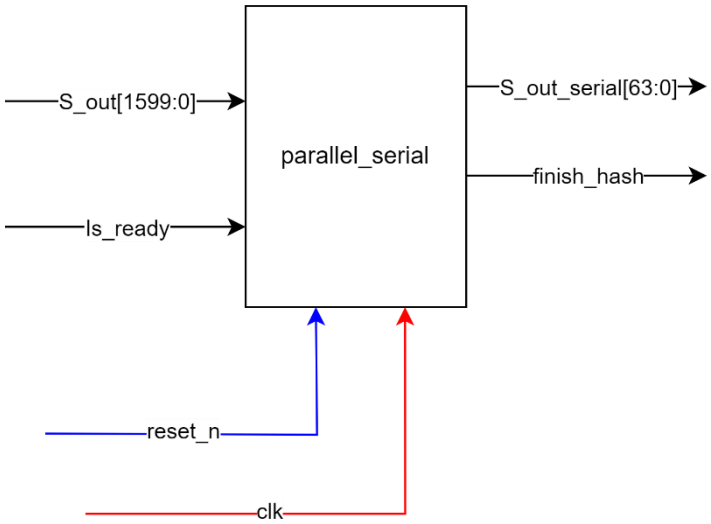
Hình 3-10 Sơ đồ khối khối round counter

Tín hiệu	I/O	Số bit	Mô tả
rnd_count	Input	5	Bộ đếm số vòng lặp từ 0 đến 23
round	Output	5	Ngõ ra của bộ đếm

Bảng 11 Chân kết nối khối round counter

3.4.7 Khối Parallel-Serial

3.4.7.1 Sơ đồ khối chi tiết



Hình 3-11 Sơ đồ khối khối Parallel-Serial

3.4.7.2Mô tả chân I/O

Tín hiệu	I/O	Số bit	Mô tả
S_out	Input	1600	Dữ liệu đầu vào cần chuyển đổi
is_ready	Input	1	Tín hiệu cho phép bắt đầu chuyển đổi dữ liệu
S_out_serial	Output	64	Chuỗi ngõ ra từng khối 64-bit
finish_hash	Output	1	Cho biết đã kết thúc tính toán hàm băm

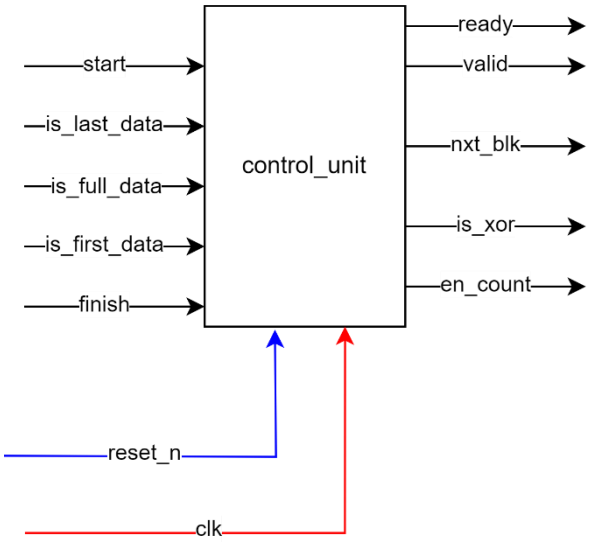
Bảng 12 Chân kết nối của khối Parallel-Serial

3.4.7.3 Mô tả chi tiết chức năng

Module *Parallel-serial* thực hiện chức năng tách dữ liệu sau khi băm (256-bit) thành các khối dữ liệu 64-bit qua từng *clock*. Điều này giúp giảm tải tài nguyên khi đưa ra kết quả hàm băm trên FPGA.

3.4.8 Khối Control Unit

3.4.8.1 Sơ đồ khối chi tiết



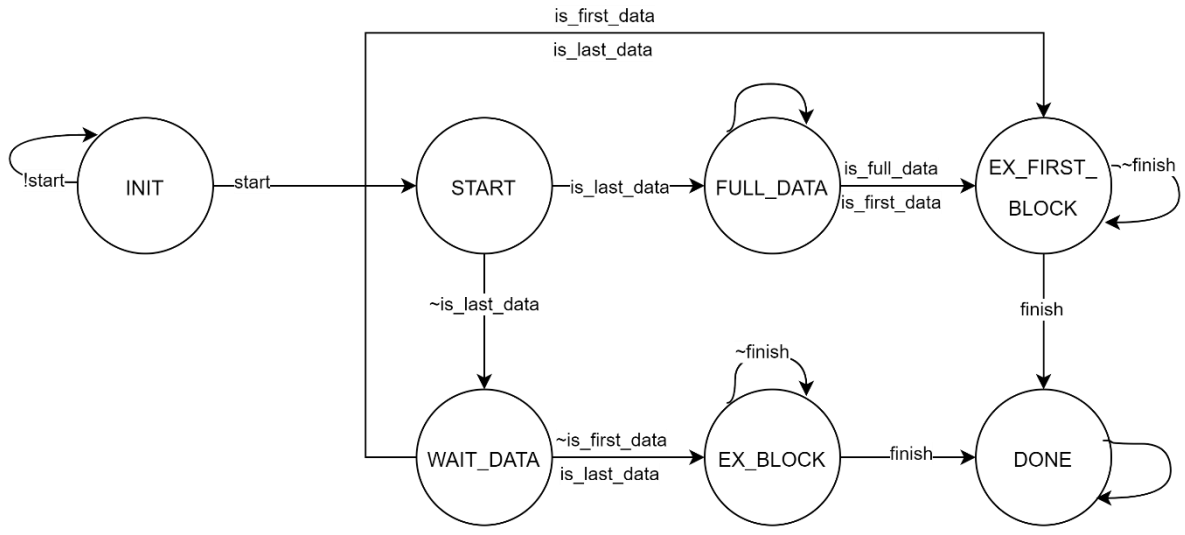
Hình 3-12 Sơ đồ khối control-unit

3.4.8.2 Mô tả chân I/O

Tín hiệu	I/O	Số bit	Mô tả
start	Input	1	Tín hiệu bắt đầu thực hiện
is_last_data	Input	1	Cho biết khối dữ liệu 64-bit là khối ngõ vào cuối cùng
is_full_data	Input	1	Cho biết đã quá dữ liệu nhận vào (>1024-bit)
is_first_data	Input	1	Cho biết khối 64-bit đầu vào đầu tiên được tính toán
finish	Input	1	Kết thúc tính toán
valid	Output	1	Cho phép khối serial-parallel-pad nhận đầu vào từ FIFO
nxt_blk	Output	1	1 nếu tính toán lại với chuỗi từ vòng lặp trước, 0 nếu là 1600-bit 0
is_xor	Output	1	Cho phép khối *xor_state* hoạt động
en_count	Output	1	Cho phép khối *round_counter* hoạt động
is_ready	Output	1	Cho phép khối *parallel-serial* xuất đầu ra

Bảng 13 Bảng chân kết nối Control Unit

3.4.8.3 Máy trạng thái Control Unit



Hình 3-13 Máy trạng thái Control Unit

3.4.8.3.1 Mô tả trạng thái

Trạng thái	Mô tả
INIT	Trạng thái ban đầu sau reset.
START	Trạng thái bắt đầu nhận đầu vào.
FULL_DATA	Trạng thái đã đủ số khối đầu vào cần tính toán.
WAIT_DATA	Trạng thái chờ cho đủ khối đầu vào cần tính toán.
EX_FIRST_BLOCK	Trạng thái thực hiện tính toán khối đầu tiên.
EX_BLOCK	Trạng thái thực hiện tính toán không phải khối đầu tiên.
DONE	Trạng thái đã tính toán xong.

Bảng 14 Mô tả trạng thái Control Unit

3.4.8.3.2 Ngõ ra trạng thái

	valid	nxt_blk	is_xor	en_count	is_ready
INIT	0	0	0	0	0
START	1	0	0	0	0
FULL_DATA	0	0	1	0	0
WAIT_DATA	1	0	1	0	0
EX_FIRST_BLOCK	0	0	1	1	0
EX_BLOCK	0	1	1	1	0
DONE	0	0	0	0	1

Bảng 15 Giá trị ngõ ra máy trạng thái

4. THỰC HIỆN PHẦN MỀM

Yêu cầu đặt ra cho phần mềm: Phần mềm sẽ thực hiện hàm băm SHA3-256 bằng thuật toán Keccak. Việc thực hiện tính toán bằng phần mềm sẽ giúp so sánh kết quả sau khi thực hiện phần cứng với phần mềm.

Việc thực hiện bằng phần mềm có thể sử dụng ngôn ngữ Python, nhưng do chỉ kiểm tra kết quả sau khi băm nên em sẽ dùng các tools có sẵn để kiểm tra kết quả sau khi băm. Điển hình là https://emn178.github.io/online-tools/sha3_256.html và https://codebeautify.org/sha3-256-hash-generator

5. KẾT QUẢ THỰC HIỆN

5.1 Cách thức đo đạc, thử nghiệm

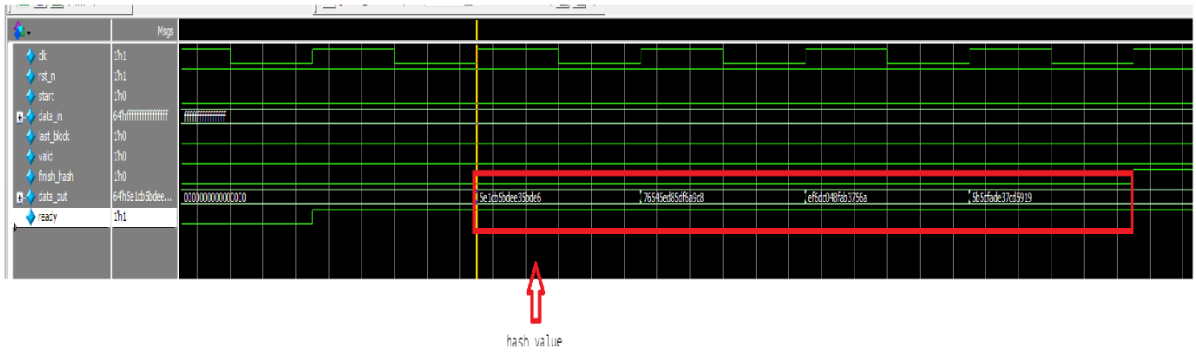
Sau khi hoàn thành, thiết kế được thử nghiệm bằng chương trình mô phỏng phần cứng. Ở đây em sử dụng Modelsim để thiết kế và mô phỏng phần cứng, website

https://codebeautify.org/sha3-256-hash-generator đối chiếu và kiểm tra tính chính xác của kết quả.

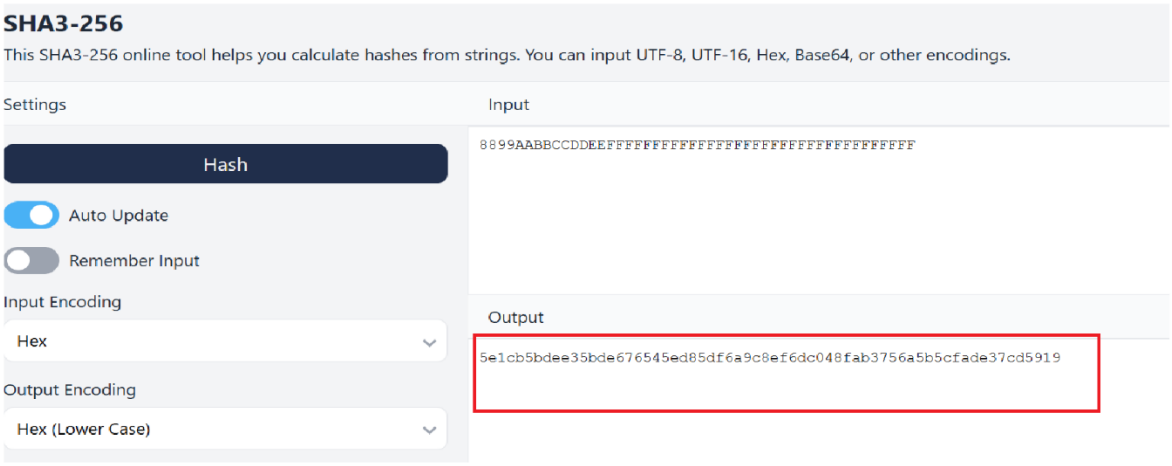
Về đo đạc các thông số như tài nguyên phần cứng hay tốc độ thì em sẽ tổng hợp thiết kế trên Quartus Prime với kit FPGA Cyclone V SX SoC—5CSXFC6D6F31C6N.

5.2 Kết quả đo đạc

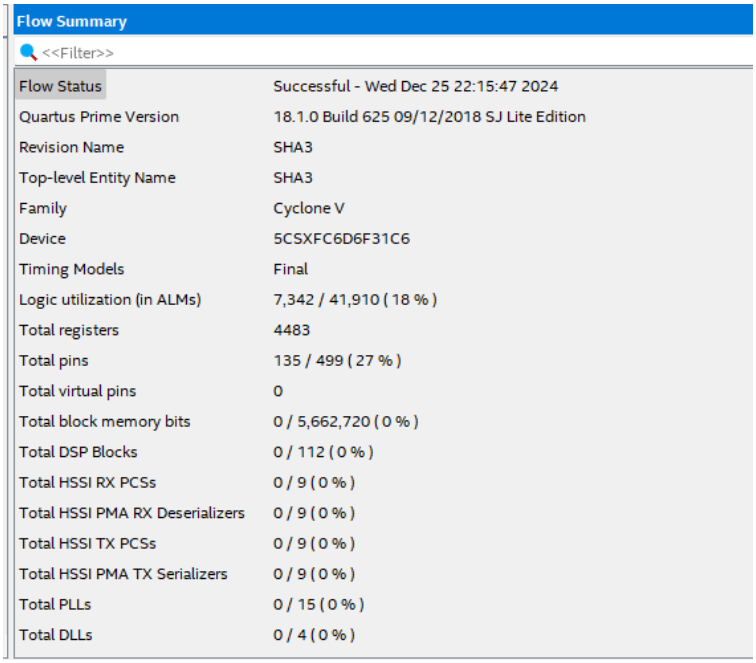
- Mô phỏng trường hợp ngõ vào "8899AABBCCDDEEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"



Hình 5-1 Hình ảnh mô phỏng



Hình 5-2 Đối chiếu kết quả



Flow Summary

<<Filter>>

Flow Status	Successful - Wed Dec 25 22:15:47 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	SHA3
Top-level Entity Name	SHA3
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	7,342 / 41,910 (18 %)
Total registers	4483
Total pins	135 / 499 (27 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

Hình 5-3 Tài nguyên sử dụng FPGA

6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

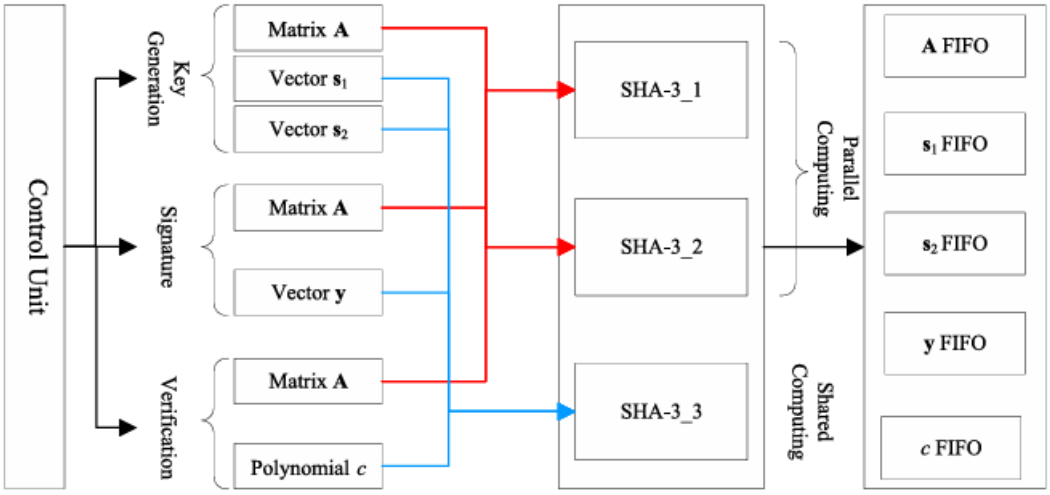
Quá trình thực hiện đồ án đã giúp em hiểu nguyên lý hoạt động, cấu trúc Keccak, các bước xử lý và tạo giá trị băm. Cơ bản đã thực thi được hàm băm SHA3-256 bằng giải thuật Keccak trên ngôn ngữ mô tả phần cứng.

Tuy nhiên thiết kế chỉ mới đạt được chức năng cơ bản của hàm băm chứ chưa đánh giá được hiệu năng xử lý và tối ưu hiệu năng, tài nguyên phần cứng.

Do đó trong tương lai (đồ án tốt nghiệp) em sẽ triển khai cấu trúc SHA3-Parallelization sẽ được trình bày trong 6.2

6.2 Hướng phát triển

SHA-3 Parallelization: Nhiều mô-đun SHA-3 được sử dụng để tính toán song song nhằm tăng tốc độ xử lý.



Hình 6-1 Cấu trúc tính toán song song Multi-SHA3

7. TÀI LIỆU THAM KHẢO

[1] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, Cryptographic sponge functions, January 2011, http://sponge.noekeon.org/CSF-0.1.pdf

[2] National Institute of Standards and Technology, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[3] Bin Li, Yunfei Yan, Yuanxin Wei, Heru Han, Scalable and Parallel Optimization of the Number Theoretic Transform Based on FPGA, September 2023, https://ieeexplore.ieee.org/document/10266840

[4] Jayaram Bhasker, Verilog Primer, 1997.