Đại học Quốc gia Thành phố Hồ Chí Minh

Trường Đại học Khoa học Tự nhiên

Khoa Điện tử Viễn thông

Bộ môn Điện tử

THIẾT KẾ LÕI TĂNG TỐC THUẬT TOÁN MẬT MÃ NHẸ ASCON ỨNG DỤNG NÂNG CAO BẢO MẬT DỮ LIỆU TRONG HỆ THỐNG INTERNET OF THINGS

Người thực hiện: LÊ ĐỨC DUY

Giảng viên hướng dẫn: PGS.TS. LÊ ĐỨC HÙNG

Mục lục

Danh mục hình ảnh

Danh mục bảng

1. Ý tưởng thiết kế

Trong bối cảnh đô thị thông minh ngày càng phát triển, việc đảm bảo an toàn và bảo mật dữ liệu trong các hệ thống Internet of Things (IoT) là một thách thức quan trọng. Các thiết bị IoT thường có tài nguyên hạn chế về bộ nhớ, hiệu suất tính toán và năng lượng, do đó, việc tích hợp các thuật toán mật mã nhẹ nhưng vẫn đảm bảo mức độ an toàn cao là yêu cầu cấp thiết.

Dự án này tập trung vào việc thiết kế và triển khai lõi tăng tốc phần cứng cho thuật toán mật mã nhẹ Ascon – một thuật toán được chuẩn hóa bởi NIST cho các ứng dụng bảo mật trong môi trường tài nguyên hạn chế. Lõi tăng tốc được thiết kế nhằm tối ưu hóa tốc độ xử lý, giảm mức tiêu thụ năng lượng và tăng cường khả năng bảo vệ dữ liệu trong hệ thống IoT của đô thị thông minh.

Bằng cách sử dụng kiến trúc phần cứng tối ưu, giải pháp này giúp các thiết bị IoT thực hiện các tác vụ mã hóa và xác thực dữ liệu một cách nhanh chóng và hiệu quả. Điều này không chỉ cải thiện hiệu suất của các hệ thống IoT mà còn tăng cường khả năng chống lại các cuộc tấn công mạng, góp phần xây dựng một nền tảng đô thị thông minh an toàn và đáng tin cậy hơn.

2. Đặc tính của thiết kế

Đặc tính của thiết kế được liệt kê như sau:

- Tốc độ mã hóa: 0.125 cycles/byte.
- Tần số hoạt động: 100 MHz (mục tiêu hướng đến)

3. Kiến trúc

Năm 2023, NIST đã công bố lựa chọn họ thuật toán Ascon được thiết kế bởi Dobraunig, Eichlseder, Mendel và Schlafffer để cung cấp các giải pháp mật mã cho các thiết bị bị giới hạn tài nguyên. Tiêu chuẩn này giới thiệu một họ dựa trên Ascon mới với mật mã có khóa đối xứng, cung cấp các khả năng mã hóa xác thực với associated data (Ascon-AEAD128), hàm băm và hàm băm có đầu ra mở rộng (Ascon-Hash256, Ascon-XOF128 và Ascon-CXOF128). Họ Ascon đặc trưng bởi các hàm dựa trên hóa vị nhẹ và cung cấp khả năng bảo mật, hiệu quả và tính linh hoạt mạnh mẽ, khiến nó trở nên lý tưởng cho các môi trường hạn chế về tài nguyên, như các thiết bị IOT, hệ thống nhúng và cảm biến công suất thấp.

Việc triển khai bộ tăng tốc thuật toán mật mã nhẹ Ascon là điều cần thiết để việc mã hóa Ascon hoạt động trên các thiết bị có tài nguyên hạn chế, nhưng lại cần được tính toán nhanh để xử lý được các chức năng phức tạp cần đến mã hóa dữ liệu khi truyền giữa các thiết bị hoặc lưu trữ, tránh các cuộc tấn công gây rò rỉ thông tin, nguy hại đến dữ liệu người dùng. Ở đây sẽ sử dụng phần cứng để thực hiện bộ tăng tốc có khả năng thực hiện 3 loại thuật toán Ascon được đề xuất dựa trên NIST SP 800-232 [1] là Ascon-AEAD128, Ascon-Hash256 và Ascon-XOF128.

Ý tưởng của các lõi tăng tốc thuật toán dành cho Ascon đã được công bố trước đây là sử dụng một lõi hoán vị Ascon-p một vòng để giảm số chu kì cần thiết để thực hiện hoán vị [2]. Tuy nhiên việc thực hiện thuật toán sẽ phụ thuộc vào CPU vì CPU sẽ tham gia vào quá trình thực hiện mã hóa. Điều đó sẽ khó mà tối ưu lượng tài nguyên của CPU dành cho việc thực hiện thuật toán.

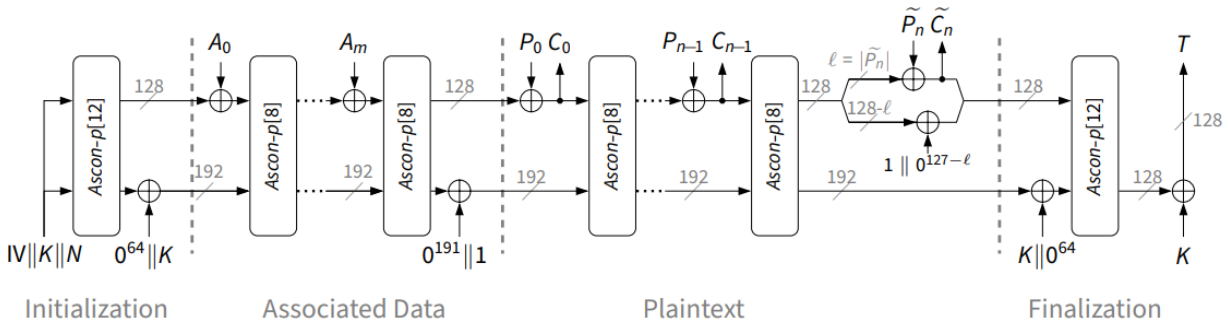
Kiến trúc phần cứng được đưa ra ở đây là việc sẽ thực hiện toàn bộ thuật toán Ascon chỉ bằng phần cứng, CPU chỉ có nhiệm vụ điều khiển, đưa dữ liệu vào, và lấy dữ liệu ra từ lõi phần cứng thực hiện thuật toán. Từ đó giảm đi lượng chu kì cần sử dụng khi thực hiện thuật toán Ascon.

3.1. Vấn đề khi thực hiện phần cứng cho lõi tăng tốc thuật toán

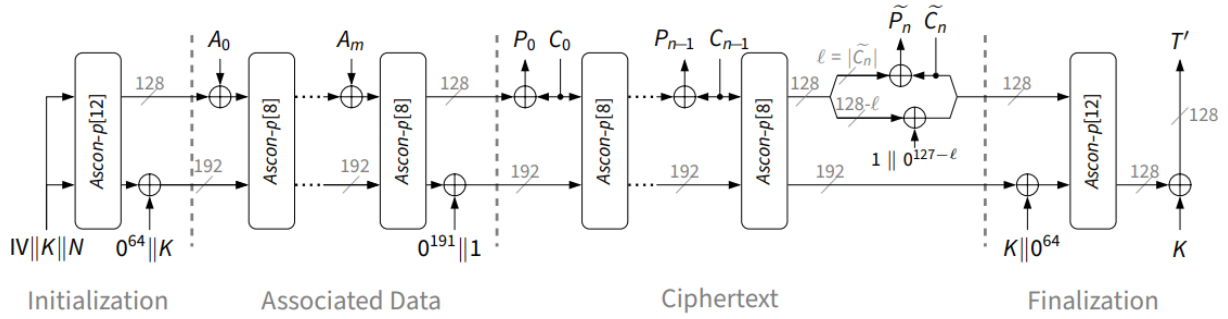
Để có thể triển khai phần cứng thực hiện tăng tốc thuật toán mật mã nhẹ Ascon, cần phải biết được các vấn đề liên quan và việc ứng dụng chúng trong các hệ thống như thế nào.

a. **Permutation.** Bộ hoán vị Ascon được đề xuất bởi NIST SP 800-232 tương tự với phiên bản Ascon v1.2, hoán vị mà các thuật toán được sử dụng là hoán vị 8 vòng và 12 vòng. Việc thực hiện phần cứng sẽ làm tăng diện tích do sử dụng nhiều cổng logic kết nối các khối Ascon-p 1 vòng lại với nhau.
b. **Vòng lặp.** Do phần cứng không thể lặp đi lặp lại để mở rộng liên tục, do đó ý tưởng được đưa ra ở đây là sử dụng các chu kì clock để thực hiện, mỗi vòng lặp sẽ ứng với số chu kì cần thiết để thực hiện vòng lặp thuật toán đó.
c. **Endianess.** Thuật toán Ascon được đề xuất bởi NIST SP 800-232 nhắm đến ứng dụng trên các thiết bị có tài nguyên hạn chế, do đó thuật toán đã được thiết kế hỗ trợ hệ thống có kiểu lưu trữ little-endian thay vì big-endian. Do đó, dữ liệu đưa vào phần cứng tăng tốc cũng phải thuộc dạng little-endian.

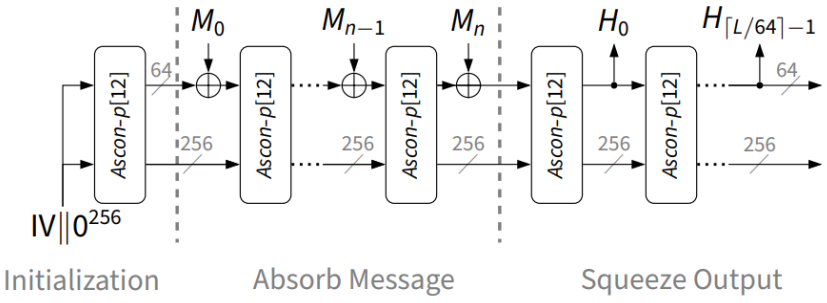
3.2. Mô hình thuật toán Ascon



Hình 1. Ascon-AEAD128 Encryption



Hình 2. Ascon-AEAD128 Decrytion



Hình 3. Ascon-Hash256 và Ascon-XOF128

3.3. Triển khai phần cứng thuật toán Ascon

3.3.1 Ascon Core (top module)

Khối Ascon Core có tác dụng thực hiện tính toán thuật toán Ascon dựa trên NIST 800-232, thực hiện 3 loại thuật toán chính là Ascon-AEAD128, Ascon-Hash256, Ascon-XOF128. Dựa vào mô hình thuật toán đã đưa ra ở trên, có thể tổng hợp lại các bước cần thiết để thực hiện mã hóa bao gồm:

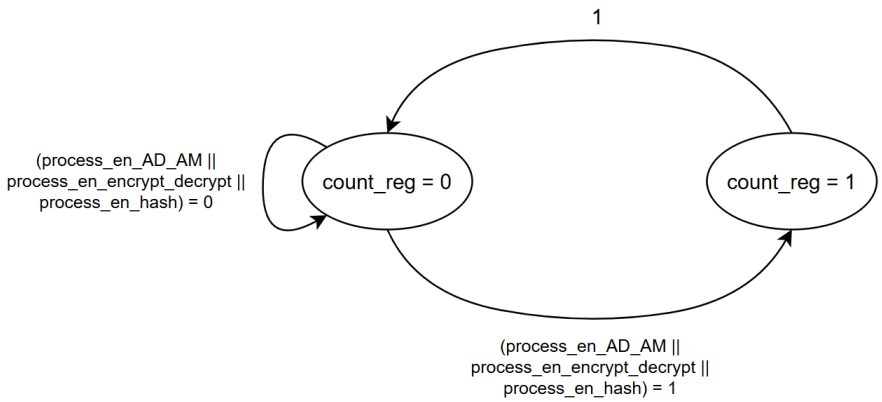
- Bước 1: khởi tạo giá trị
- Bước 2: hấp thụ dữ liệu có liên quan
- Bước 3: thực hiện mã hóa dữ liệu chính
- Bước 4: đối với mã hóa xác thực sẽ là bước tạo tag để xác thực dữ liệu đã giải mã đúng không bị mất hay sai sót trong quá trình truyền hay lưu trữ dữ liệu. Đối với hàm băm sẽ không có bước này.

Như vậy, ý tưởng ở đây là mỗi bước sẽ do một module thực hiện. Với bước khởi tạo giá trị, khối được sử dụng là Ascon-init có tác dụng tạo ra giá trị khởi tạo ban đầu, vì giá trị khởi đầu của mỗi loại thuật toán là khác nhau nên sẽ có các tín hiệu lựa chọn. Với bước hấp thụ các dữ liệu liên quan, do bước Associated Data ở Ascon-AEAD128 và Absorb Message của Ascon-Hash256 và Ascon-XOF128 có cấu trúc gần giống nhau do đó sẽ được gom vào chung một khối là Ascon-AD_AM. Về bước thực hiện mã hóa dữ liệu, ở đây sẽ chia ra thành 2 module bao gồm Ascon-Encrypt_Decrypt và Ascon-Hash để thực hiện mã hóa AEAD và hàm băm. Với bước cuối cùng tạo tag sẽ được thiết kế bởi một khối riêng là Ascon-final. Dưới dây là thiết kế của khối Ascon Core, về chi tiết những module bên trong sẽ được liệt kê chi tiết ở những phần sau.

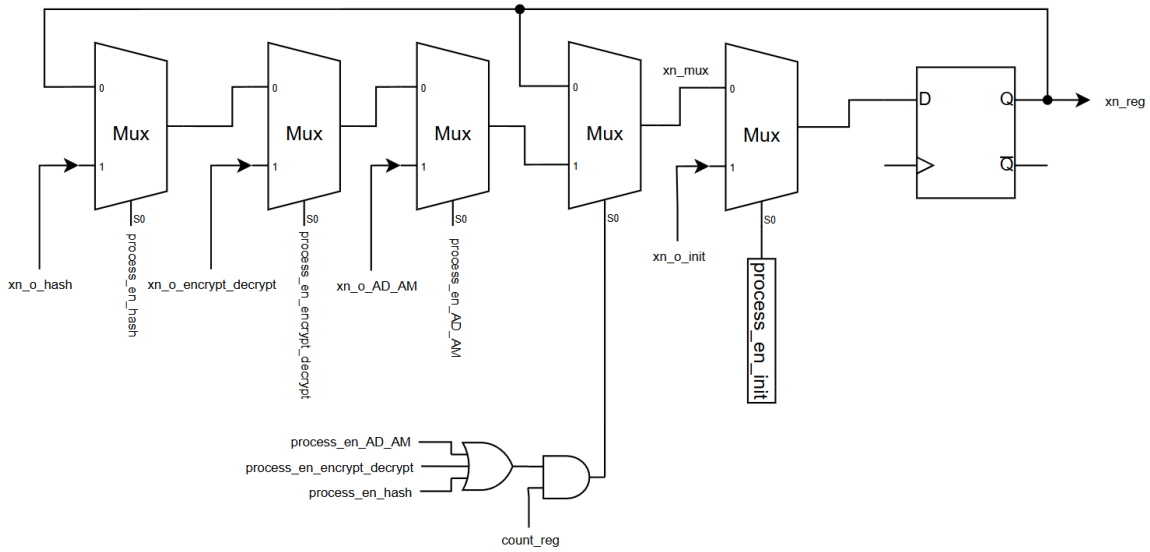
Trong quá trình thiết kế, nhận thấy các tính toán không thể hoạt động trong 1 chu kì do dữ liệu tính toán được vẫn chưa được lưu lại. Vì thế nên ở đây sử dụng 2 chu kì:

- Chu kì 1: tính toán dữ liệu, các khối tính toán dữ liệu và xuất ra đường dữ liệu có tên là xn_o_[tên khối]
- Chu kì 2: lưu dữ liệu vào thanh ghi xn_reg, sau khi các khối đã tính toán, lấy dữ liệu từ xn_o_[tên khối] lưu vào xn_reg, việc phân đường dữ liệu dựa trên các bit process_en_[tên khối]

Vì vậy mà khối FSM được thiết kế sao mà sau khi ghi dữ liệu vào thanh ghi sẽ mặc định quay về trạng thái cho phép tính toán dữ liệu.



Hình 4. Khối FSM thay đổi trạng thái giữa tính toán dữ liệu và lưu dữ liệu



Hình 5. Xử lý lưu dữ liệu đã tính toán được vào thanh ghi xn_reg (Ascon Core)

Khối trên xử lý việc phân chia đường dữ liệu dựa trên các bit process_en_[tên khối], do chu kì 1 sẽ là chu kì tính toán từ các khối do đó sẽ lấy count_reg để xác định xem tại thời điểm đó có thể lưu vào thanh ghi được hay không, nếu count_reg = 0 có nghĩa là chu kì đó đang diễn ra việc tính toán bên trong các khối, đợi đến khi count_reg =1 sẽ cho phép lưu dữ liệu đã tính toán đó vào bên trong thanh ghi xn_reg. Đối với dữ liệu initialization, chỉ cần 1 chu kì để có thể khởi tạo trạng thái ban đầu. Vì thế khối initialization sẽ không phụ thuộc vào count_reg.

Do các tín hiệu lưu vào thanh ghi mỗi lần sẽ chỉ lưu vào 1 trạng thái tính toán, đo đó cần phải có mỗi mạch kiểm tra các tín hiệu process_en_[tên khối] có đồng thời cho phép hay không, nếu có sẽ báo lỗi ra err.

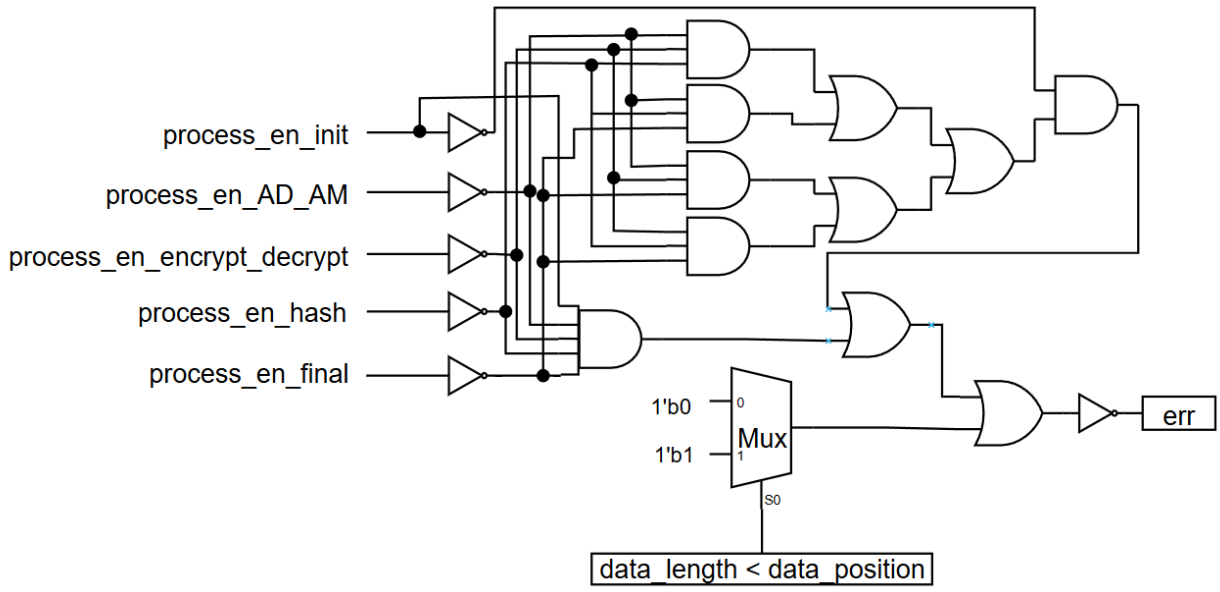
Process_en_init (A)	Process_en_AD_AM (B)	Process_en_encrypt_decrypt (C)	Process_en_hash (D)	Process_en_final (E)	Err (Y)
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0

Bảng 1. Các tín hiệu process_en không gây lỗi hệ thống

Sử dụng K-map có thể tính được công thức sau:

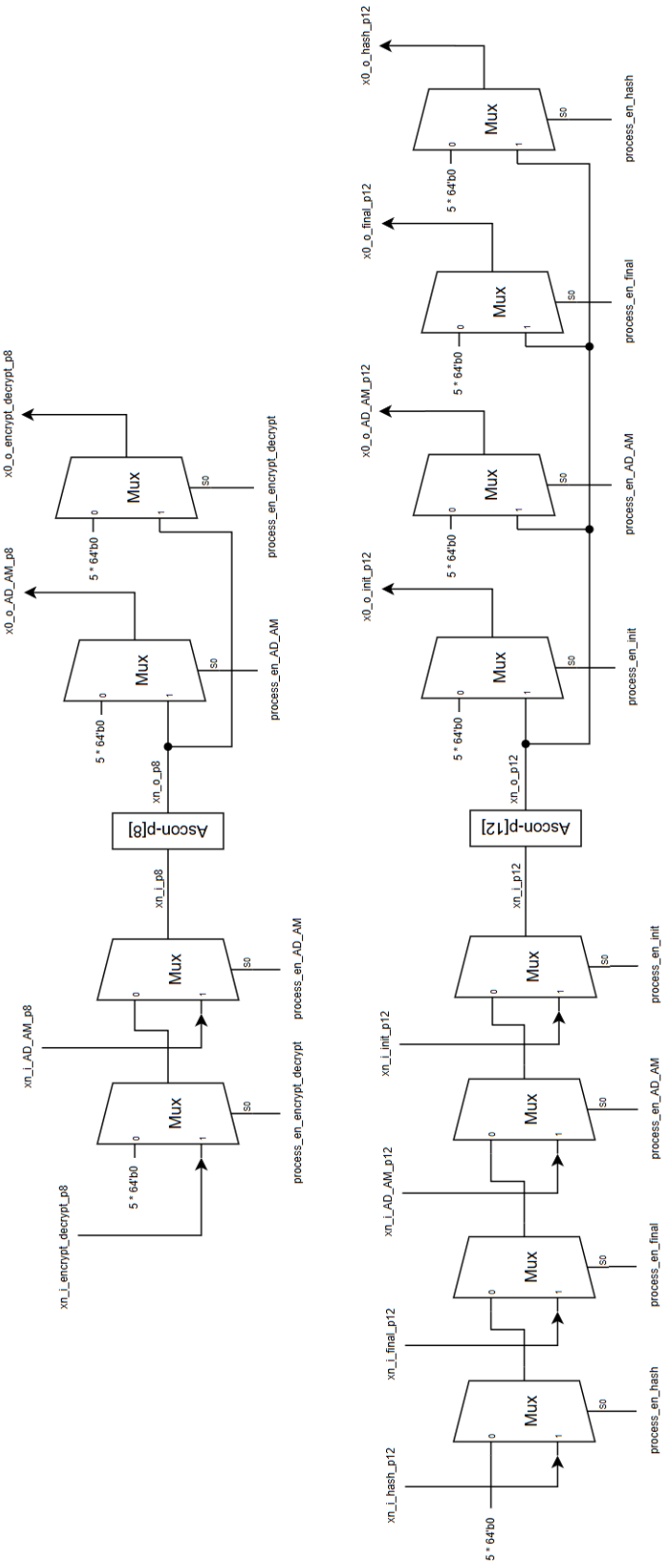
[A.(B'C'D'E') + A'(B'C'D' + B'D'E' + B'C'E' + C'D'E')]'

Do vấn đề về phát triển lên phần cứng và thuật toán hoạt động lặp đi lặp lại đến khi hết chuỗi dữ liệu cần mã hóa, do đó hai tín hiệu là data_length và data_position được đưa vào. Mục đích của hai tín hiệu này là xác định xem độ dài dữ liệu và lượng dữ liệu đã mã hóa tính theo byte từ đó có thể đưa vào các khối padding để căn chỉnh lại dữ liệu sao cho phù hợp với thuật toán được đưa ra. Vì vậy cần đảm bảo việc data_length >= data_position.

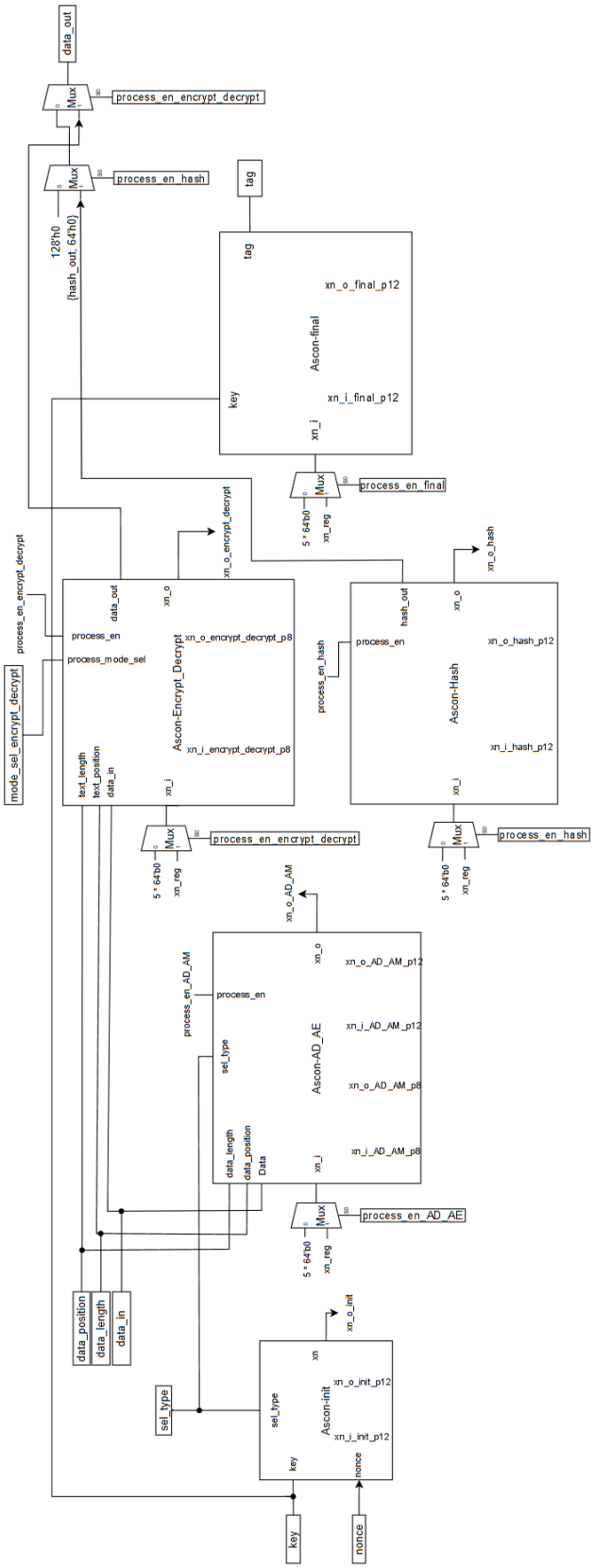


Hình 6. Khối xử lý tín hiệu err (Ascon Core)

Về phần các tín hiệu bên trong khối Ascon Core, để giảm lượng tài nguyên sử dụng, các module bên trong đều có chân vào ra để có thể đưa tín hiệu vào ra 2 module chịu trách nhiệm hoán vị là Ascon-p[8] và Ascon-p[12]. Vì vậy nên cần một mạch số chịu trách nhiệm phân chia đường dữ liệu vào ra các module Ascon-p. Hình bên dưới mô tả mạch số làm nhiệm vụ trên.



Hình 7. Phân chia đường dữ liệu đưa vào Ascon-p[8] và Ascon-[12] (Ascon Core)

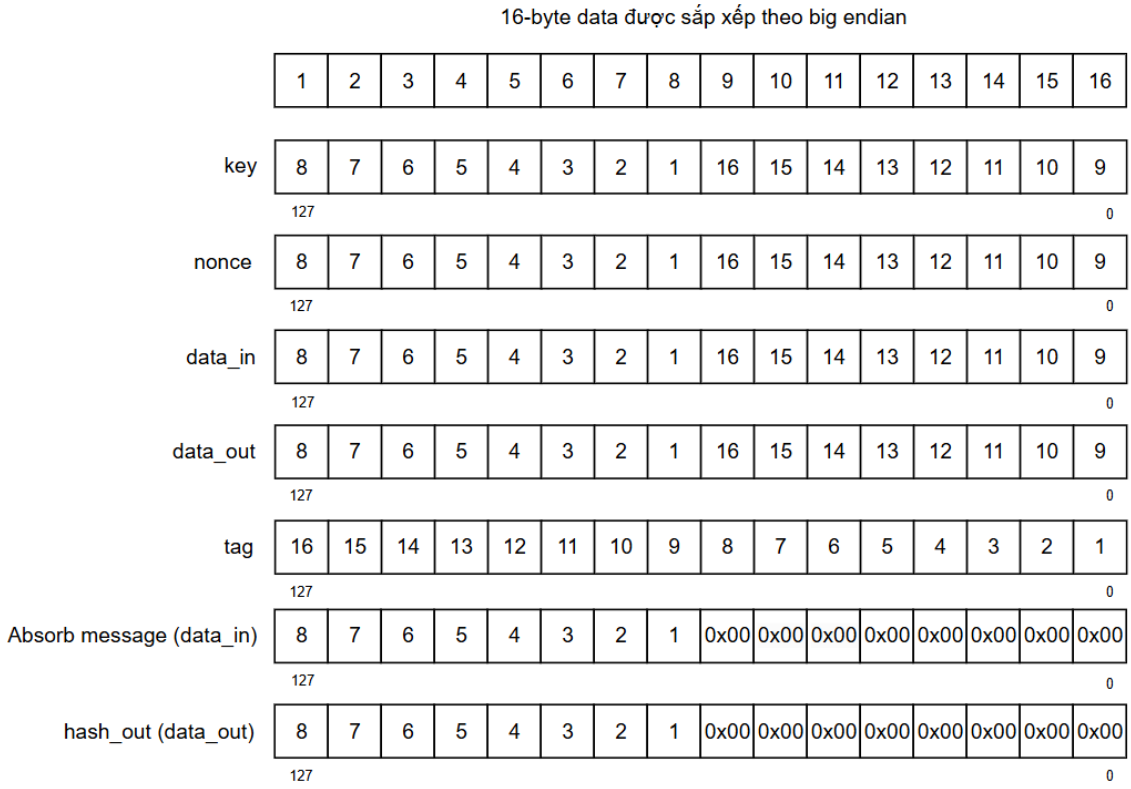


Hình 8. Kết nối các module chính và xử lý (Ascon Core)

Port	Width	Direction	Function
Clk, rst_n	1	Input	Clock và reset
Key	128	Input	Key cho AEAD, không sử dụng trong Hash
nonce	128	Input	Nonce cho AEAD, không sử dụng trong Hash
Sel_type	2	Input	Lựa chọn loại mã hóa 2'b00: Ascon-AEAD128 2'b01: Ascon-Hash256 2'b10: Ascon-XOF128
mode_sel_encrypt_decrypt	1	Input	Khi thực hiện Ascon-AEAD128: - 0: mã hóa (encrypt) - 1: giải mã (decrypt)
data_length	32	Input	Chiều dài của data hiện tại
data_position	32	Input	Số lượng dữ liệu đã mã hóa
data_in	128	Input	Trong quá trình xử lý AD_AM ở chế độ AEAD thì data_in chính là associated data, ở chế độ Hash thì data_in[127:64] là absorb message. khi thực hiện mã hóa ở chế độ AEAD: - encrypt: input plaintext - decrypt: input ciphertext
data_out	128	Output	Trong khi thực hiện hash, thì data_out [127:64] là kết quả hash. Khi thực hiện mã hóa ở chế độ AEAD: - encrypt: output ciphertext - decrypt: output plaintext
process_en_init	1	Input	Cho phép khối Ascon-init tạo giá trị khởi đầu
process_en_AD_AM	1	Input	AEAD: thực hiện xử lý associated data Hash: thực hiện absorb message
process_en_encrypt_decrypt	1	Input	Cho phép khối Ascon-Encrypt_Decrypt thực hiện mã hóa hoặc giải mã
process_en_hash	1	Input	Cho phép khối Ascon-Hash thực hiện băm dữ liệu
process_en_final	1	Input	Tạo giá trị tag cho chức năng AEAD
tag	128	Output	Tag được tạo ra sau khi thực hiện chức năng AEAD
err	1	Output	Trả về khi data_length < data_position

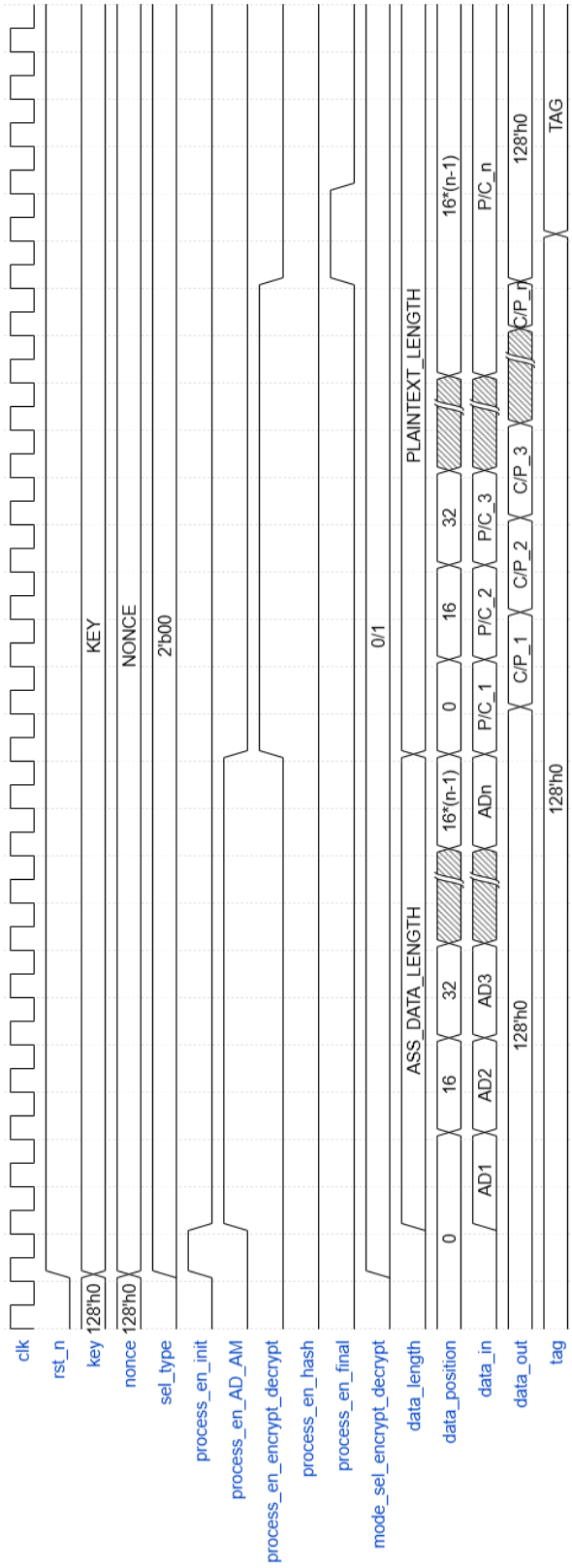
Bảng 2. IO port list Ascon Core

Về cách truyền dữ liệu vào lõi, dựa trên việc các vi xử lý thực hiện lưu trữ dữ liệu theo kiểu little endian, đường dữ liệu của lõi được thiết kế phù hợp cho việc truyền mỗi 64-bit dữ liệu được sắp xếp theo little endian. Hình dưới đây minh họa cho việc truyền dữ liệu vào và lấy dữ liệu ra tại data_in và data_out.

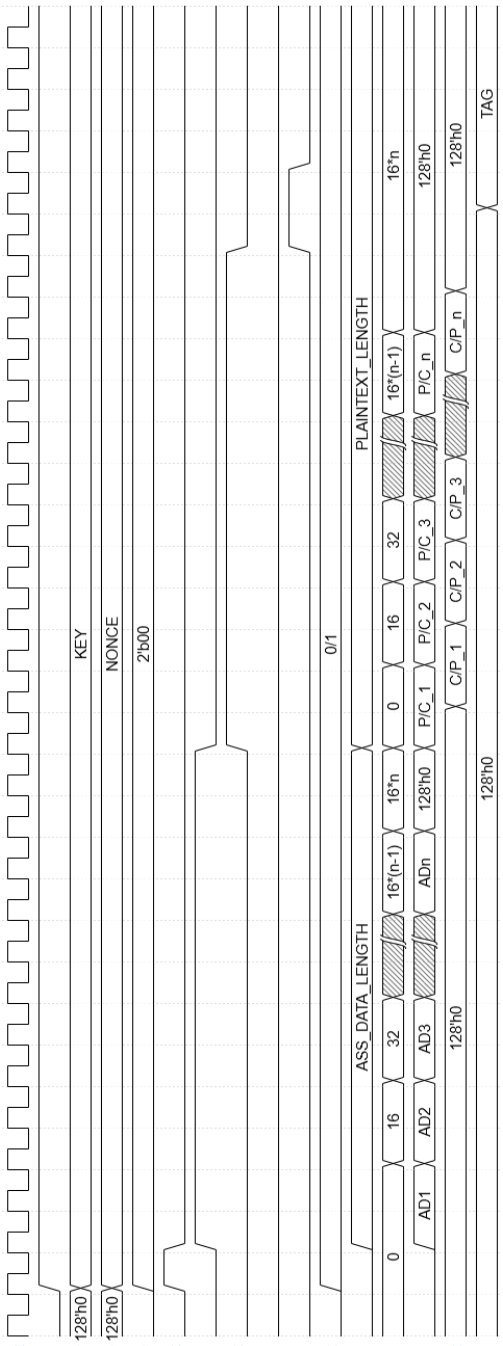


Hình 9. Mô tả vị trí byte khi truyền dữ liệu vào và lấy kết quả ra

Về hoạt động, dưới đây là dạng sóng của tín hiệu, dạng sóng được chia thành 2 dạng, một dạng của AEAD, 1 dạng của Hash. Mỗi dạng đầu có một trường hợp thường và 1 trường hợp đặc biệt. Đối với AEAD, trường hợp thường là khi chiều dài dữ liệu không là bội số của 16, trường hợp đặc biệt là khi chiều dài dữ liệu là bội số của 16. Đối với Hash, trường hợp thường là khi chiều dài dữ liệu không là bội số của 8, trường hợp đặc biệt là khi chiều dài dữ liệu là bội số của 8. Nguyên nhân dẫn đến việc chia thành 2 trường hợp như vậy là do tính chất của loại mã hóa này khi không đủ số byte (16 đối với AEAD, 8 đối với Hash) ở cuối cùng sẽ padding ra để đầy đủ dữ liệu, dẫn đến khi số byte cuối cùng đầy đủ, thuật toán vẫn tạo thêm 1 padding trống không có dữ liệu. Điều đó là nguyên nhân khi thực hiện lên phần cứng sẽ có 2 trường hợp cho 2 dạng mã hóa.



Hình 10. Dạng sóng khi thực hiện mã hóa/giải mã trường hợp thường

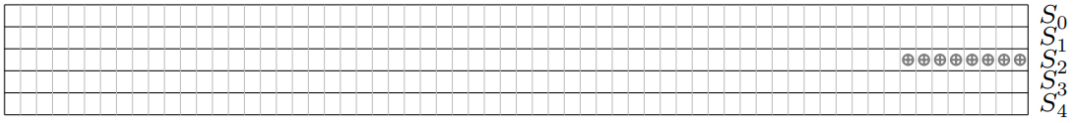


Hình 11. Dạng sóng khi thực hiện mã hóa/giải mã trường hợp đặc biệt

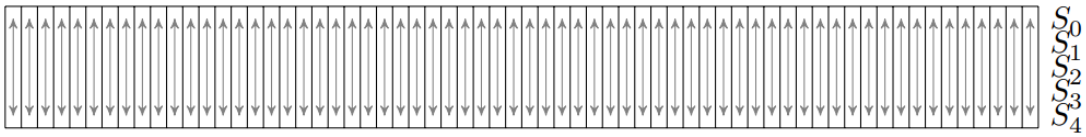
3.3.2 Ascon permutation

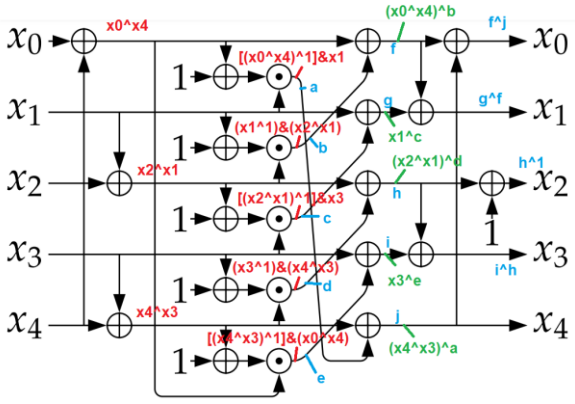
Ascon permutation (Ascon-p) là một khối thực hiện hoán vị 5 biến, mỗi biến dài 64-bit, tổng số bit được hoán vị là 320-bit, được mô tả là thực hiện 3 bước bao gồm:

- Bước 1: thêm hằng số vòng bằng cách sử dụng phép XOR hằng số với S2



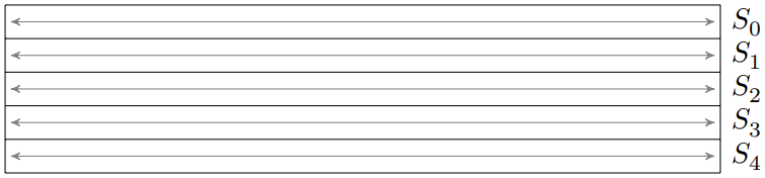
- Bước 2: hoán vị giữa các bit ở cùng vị trí của các biến với thuật toán như bên dưới





Hình 12. Substitution layer (S-box)

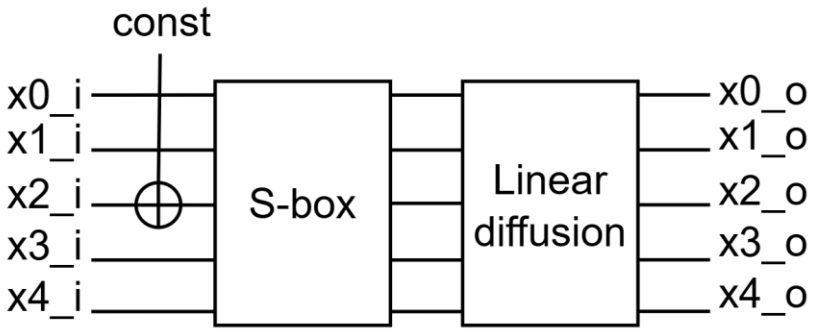
- Bước 3: hoán vị giữa các bit trong dùng một biến.



$$x_0 \leftarrow \Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$
$$x_1 \leftarrow \Sigma_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$
$$x_2 \leftarrow \Sigma_2(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)$$
$$x_3 \leftarrow \Sigma_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)$$
$$x_4 \leftarrow \Sigma_4(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$$

Hình 13. Linear Diffusion layer

Khi được đưa lên phần cứng, ở đây được thiết kế bằng cách tạo ra một module có input là dữ liệu các biến, output là kết quả sau hoán vị.

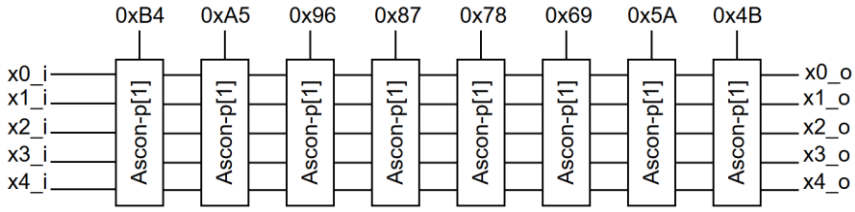


Hình 14. Sơ đồ khối Ascon-p[1]

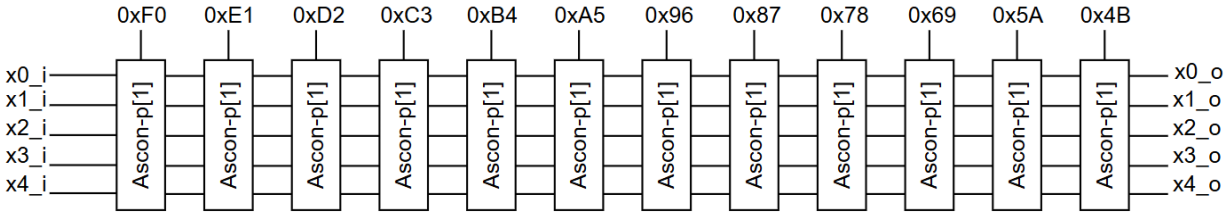
Port name	width	Direction
x0_in	64	Input
x0_in	64	Input
x0_in	64	Input
x0_in	64	input
x0_in	64	Input
x0_out	64	Output
x0_out	64	Output
x0_out	64	Output
x0_out	64	Output
x0_out	64	Output
CONSTN	8	input

Bảng 3. IO ports list của Ascon-p[1]

Để tạo ra được Ascon-p[8] và Ascon-p[12], có thể nối nhiều khối Ascon-p[1] lại với nhau. Như vậy có thể giải quyết được vấn đề cần nhiều chu kì để thực hiện hoán vị khi sử dụng phần mềm. Với cách làm này, chỉ cần có dữ liệu tại đầu vào input, ngay lập tức sẽ có kết quả đầu ra output, có thể nói Ascon-p[8] và Ascon-p[12] là hai khối mạch tổ hợp lặp đi lặp lại của khối Ascon-p[1]



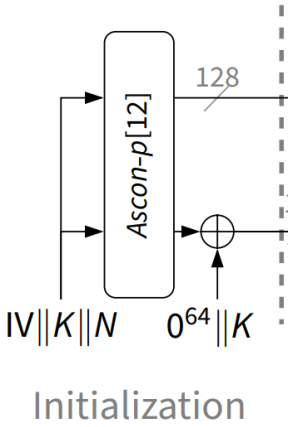
Hình 15. Sơ đồ khối Ascon-p[8]



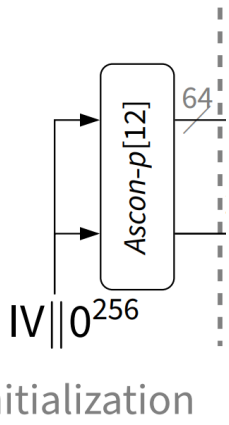
Hình 16. Sơ đồ khối Ascon-p[12]

3.3.3 Ascon initialization

Ascon initialization là bước đầu tiên của thuật toán Ascon có tác dụng khởi tạo giá trị hoán vị ban đầu, có trong cả 3 loại thuật toán Ascon-AEAD128, Ascon-Hash256 và Ascon-XOF128. Mỗi loại sẽ có cách khởi tạo giá trị khác nhau.

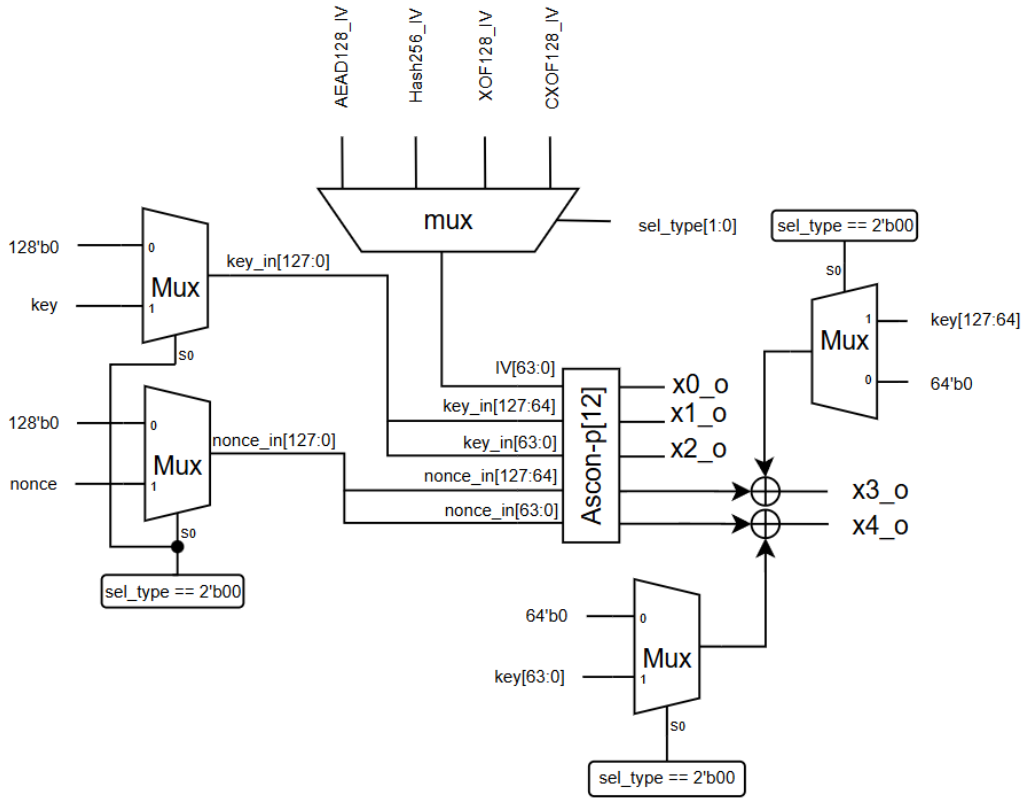


Hình 17. Ascon-AEAD128 init



Hình 18. Ascon-Hash256 và Ascon-XOF128 init

Ý tưởng cho khối phần cứng Ascon-init là tùy vào loại mã hóa mà người dùng cần sử dụng, sẽ cần các tín hiệu để người dùng có thể lựa chọn loại mã hóa và khối phần cứng này có chức năng xuất ra các tín hiệu khởi tạo đúng với loại mã hóa được chọn.



Hình 19. Sơ đồ khối Ascon-init

Ascon variants	Initial value
Ascon-AEAD128	0x00001000808c0001
Ascon-Hash256	0x0000080100cc0002
Ascon-XOF128	0x0000080000cc0003
Ascon-CXOF128	0x0000080000cc0004

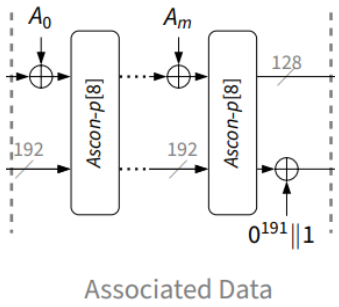
Bảng 4. Giá trị khởi tạo IV

Port	Width	Direction	Function
Clk, rst_n	1	input	Clock và reset
Sel_type	2	input	Lựa chọn chức năng của mã hóa + 2'b00: Ascon-AEAD128 + 2'b01: Ascon-Hash256 + 2'b10: Ascon-XOF128 + 2'b11: Ascon-CXOF128
Key	128	Input	Key
nonce	128	input	Nonce
X0, x1, x2, x3, x4	64	output	Giá trị initialization
x0_i_init_p12, x1_i_init_p12, x2_i_init_p12, x3_i_init_p12, x4_i_init_p12	64	output	Dữ liệu cần hoán vị đưa vào Ascon-p[12]
x0_o_init_p12, x1_o_init_p12, x2_o_init_p12, x3_o_init_p12, x4_o_init_p12	64	input	Dữ liệu đã hoán vị lấy ra từ Ascon-p[12]

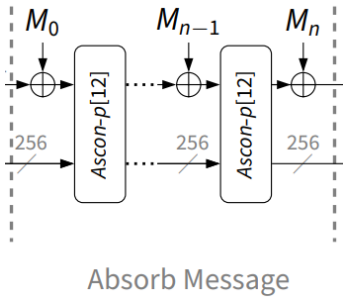
Bảng 5. IO ports list của Ascon-init

3.3.4 Ascon Associated Data – Ascon Absorb Message (Ascon-AD_AM)

Ascon Associated Data là loại dữ liệu khi mà muốn bảo vệ tính toàn vẹn của dư liệu nhưng không muốn mã hóa. Có thể là các header file, tên file… Còn Ascon Absorb Message là dữ liệu đầu vào khi cần băm.



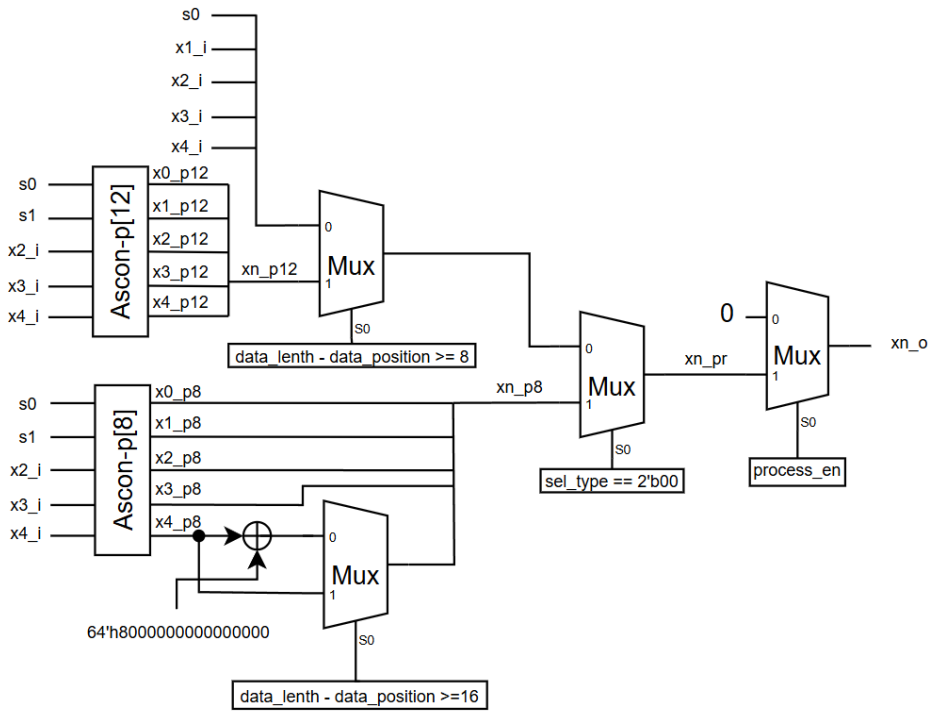
Hình 20. Associated Data (AD)



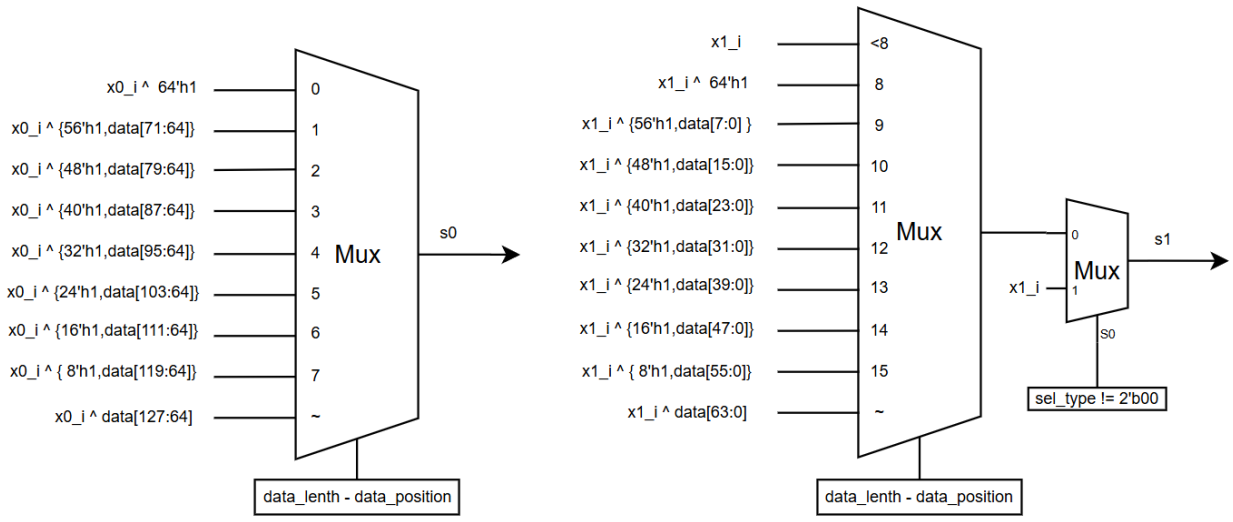
Hình 21. Asorb Message (AM)

Có thể nhận thấy quá trình xử lý AD với AM giống nhau, tuy nhiên chỉ khác nhau ở lượng dữ liệu đầu vào và bước cuối cùng quá trình xử lý. Quá trình xử lý AD có thể xử lý cùng lúc 128-bit, quá trình xử lý AM có thể xử lý cùng lúc 64-bit. Do đó có thể kết hợp xử lý AD và AM chung một khối với nhau.

Nhận thấy ở đây cần được lặp đi lặp lại nhiều lần tùy theo độ dài của dữ liệu, phần cứng được thiết kế để có thể xử lý mỗi chu kì có thể xử lý một lượng dữ liệu nhất định tùy theo loại mã hóa, như vậy có thể đáp ứng được độ dài dữ liệu biến thiên.



Hình 22. Sơ đồ khối Ascon-AD_AM



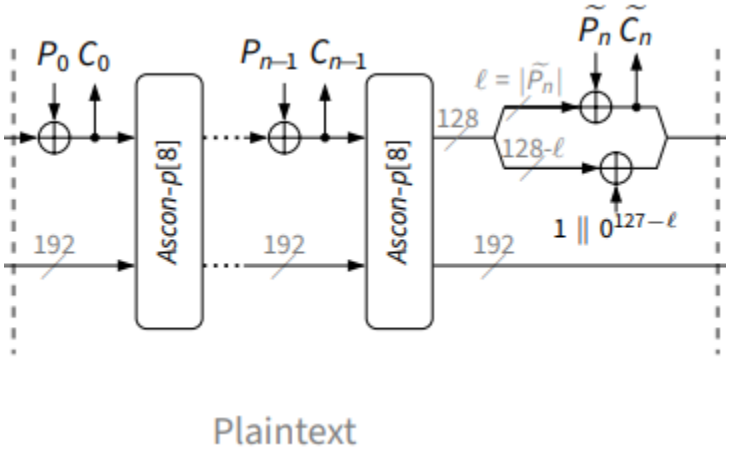
Hình 23. Khối thực hiện chức năng padding

Port	Width	Direction	Function
Clk, rst_n	1	input	Clock và reset
Process_en	1	input	Cho phép tín hiệu được ghi vào thanh ghi kết quả tính toán
sel_type	2	input	Lựa chọn chức năng của mã hóa + 2'b00: Ascon-AEAD128 (Process associated data) + 2'b01: Ascon-Hash256 (absorb message) + 2'b10: Ascon-XOF128 (absorb message) + 2'b11: Ascon-CXOF128 (absorb message)
data_length	32	input	Chiều dài của data cần mã hóa tính theo byte (VD: mã hóa 51 byte data thì data_length = 51)
data_position	32	input	Số lượng byte đã mã hóa, các bước nhảy là 16 đối với Ascon-AEAD128 và 8 đối với các loại mã hóa còn lại. (VD: mã hóa 51 byte data thì data_length = 51, tại 16 byte đầu, data_position = 0, sau khi đã mã hóa 16 byte thì data_position = 16. tương tự với chức năng hash)
Data	128	input	Associate data cho chế độ Ascon-AEAD128 và message cho 3 mã hóa còn lại
x0_i, x1_i, x2_i, x3_i, x4_i	64	input	Mã hóa của những data data trước, với lần mã hóa đầu tiên thì input sẽ là giá trị initialization, tổng 320 bit
x0_o, x1_o, x2_o, x3_o, x4_o,	64	Output	Kết quả mã hóa, tổng 320 bit
x0_i_AD_AM_p8, x1_i_AD_AM_p8, x2_i_AD_AM_p8, x3_i_AD_AM_p8, x4_i_AD_AM_p8,	64	output	Dữ liệu cần hoán vị đưa vào Ascon-p[8]
x0_o_AD_AM_p8, x1_o_AD_AM_p8, x2_o_AD_AM_p8, x3_o_AD_AM_p8, x4_o_AD_AM_p8	64	input	Dữ liệu đã hoán vị lấy ra từ Ascon-p[8]
x0_i_AD_AM_p12, x1_i_AD_AM_p12, x2_i_AD_AM_p12, x3_i_AD_AM_p12, x4_i_AD_AM_p12,	64	output	Dữ liệu cần hoán vị đưa vào Ascon-p[12]
x0_o_AD_AM_p12, x1_o_AD_AM_p12, x2_o_AD_AM_p12, x3_o_AD_AM_p12, x4_o_AD_AM_p12	64	input	Dữ liệu đã hoán vị lấy ra từ Ascon-p[12]

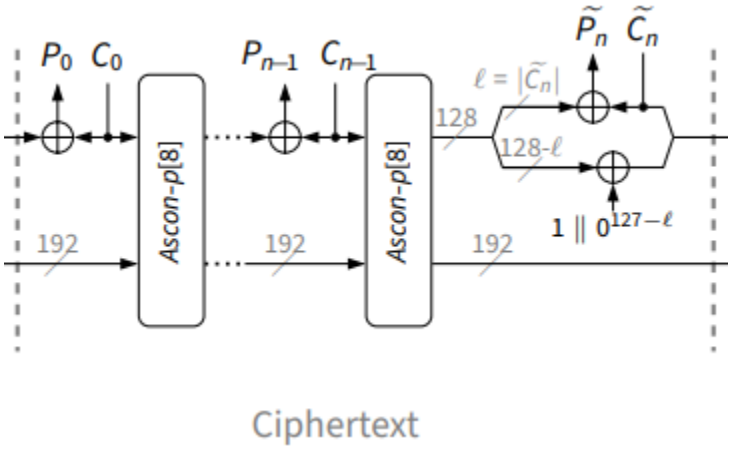
Bảng 6. IO ports list của Ascon-AD_AM

3.3.5 Ascon Encryption – Decryption

Ascon Encryption- Decryption là khối chịu trách nhiệm mã hóa hoặc giải mã dữ liệu khi thực hiện chức năng Ascon-AEAD128. Ý tưởng về cách thực hiện vòng lặp sẽ tương tự với khối Ascon-AD_AM.

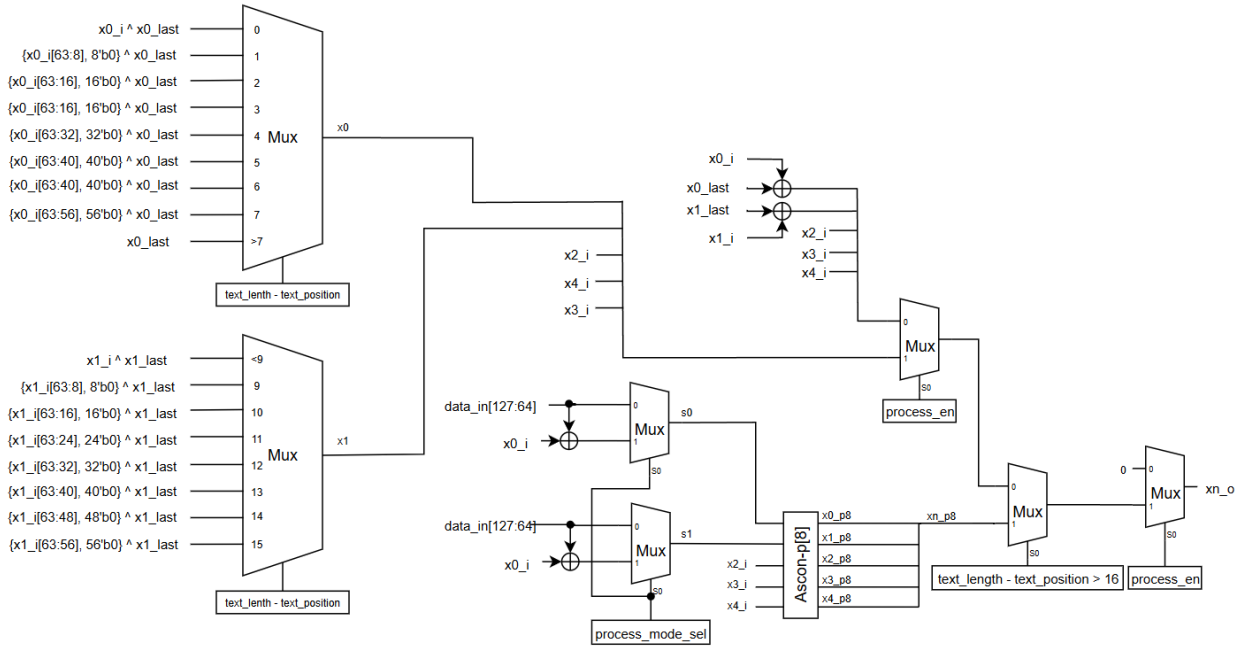


Hình 24. Ascon-AEAD128 Encryption

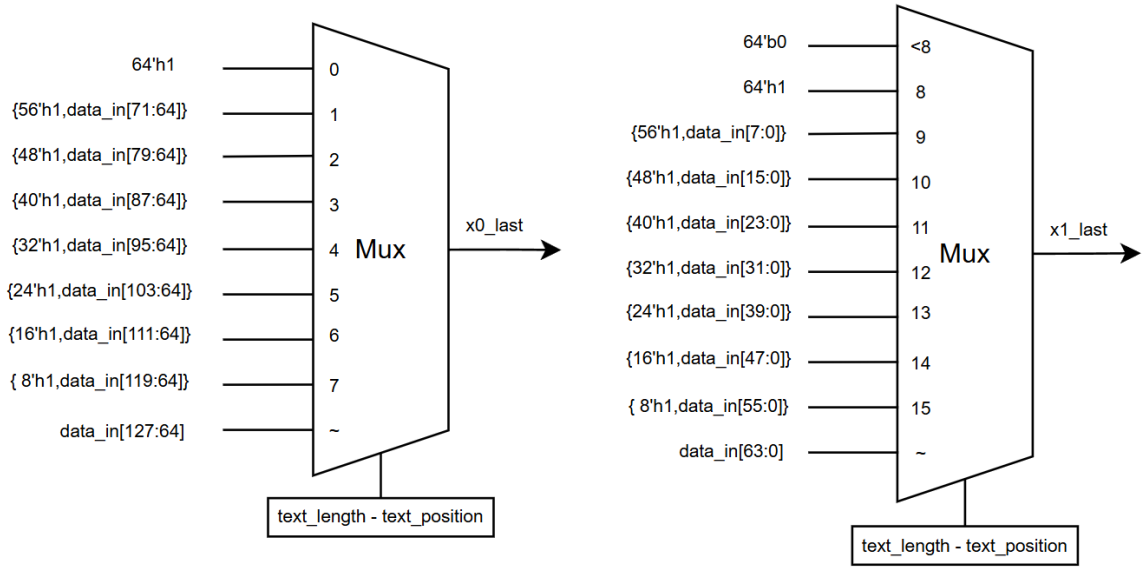


Hình 25. Ascon-AEAD128 Decryption

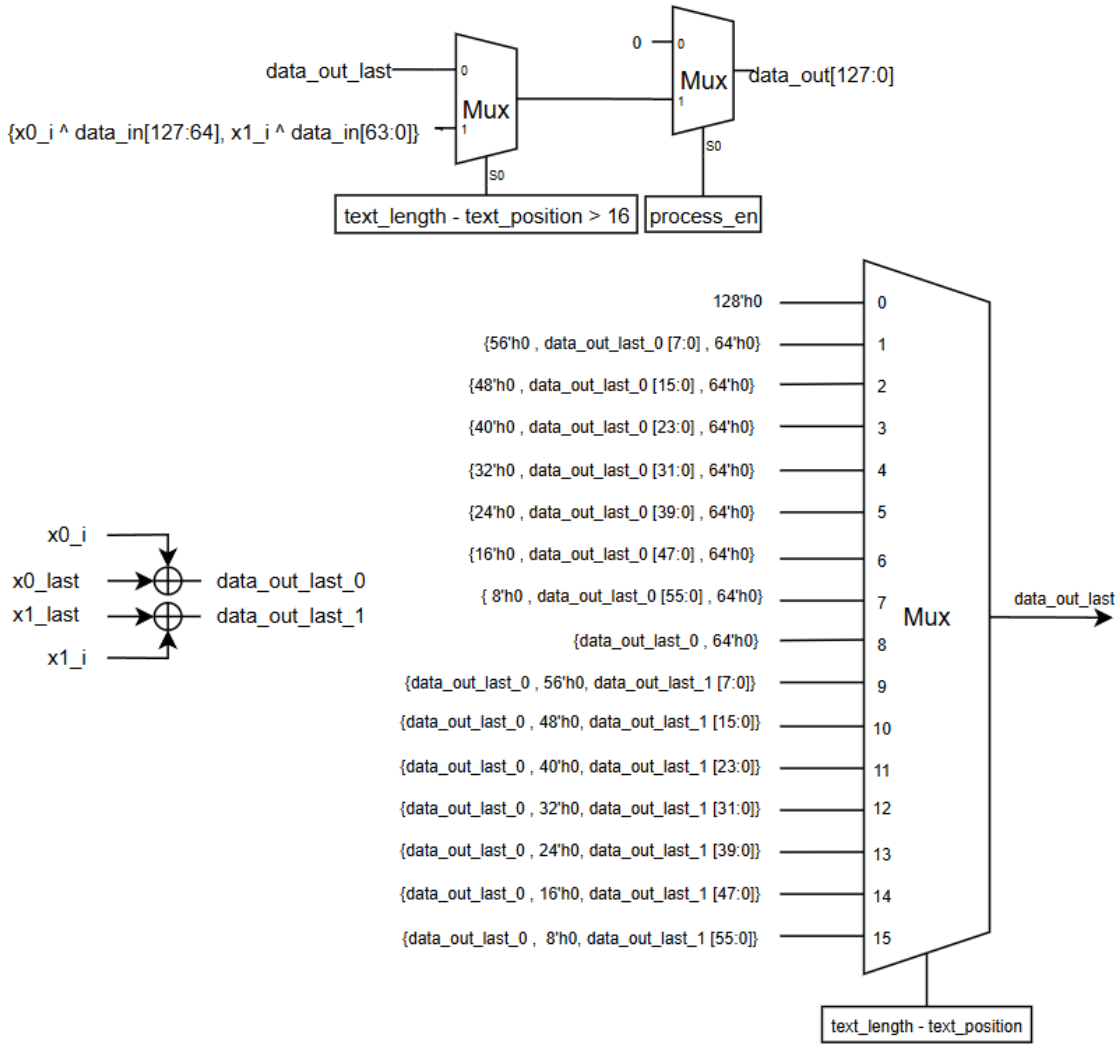
Có thể thấy được sự tương đồng trong thuật toán giữa Encryption và Decryption. Một ý tưởng được đặt ra là kết hợp hai thuật toán này vào trong một khối phần cứng, trong đó data_in của Encryption là plaintext, data_out của Encryption là ciphertext, data_in của Decryption là ciphertext, data_out của Decryption là plaintext.



Hình 26. Sơ đồ khối Ascon-Encrypt_Decrypt [1]



Hình 27. Sơ đồ khối Ascon-Encrypt_Decrypt [2]



Hình 28. Sơ đồ khối Ascon-Encrypt_Decrypt [3]

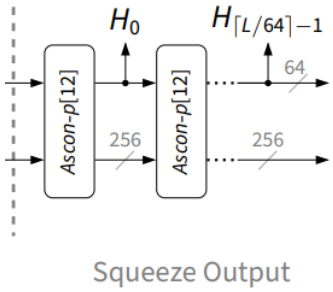
Port	Width	Direction	Function
Clk, rst_n	1	input	Clock và reset
Process_en	1	input	Cho phép tín hiệu được ghi vào thanh ghi kết quả tính toán
Process_mode_sel	1	input	0: encryption, 1: decryption
Text_length	32	input	Chiều dài của text cần mã hóa hoặc giải mã tính theo byte
Text_position	32	input	Số lượng byte text đã mã hóa hoặc giải mã
Data_in	128	input	Plaintext với chế độ Encrypt, ciphertext với chế độ decryption
Data_out	128	Output	Ciphertext với chế độ Encrypt, Plaintext với chế độ Decryption

x0_i, x1_i, x2_i, x3_i, x4_i	64	input	Mã hóa của những data trước, với lần mã hóa đầu tiên thì input sẽ là giá trị Associated Data cuối cùng
x0_o, x1_o, x2_o, x3_o, x4_o,	64	Output	Kết quả hoán vị
x0_i_encrypt_decrypt_p8, x1_i_encrypt_decrypt_p8, x2_i_encrypt_decrypt_p8, x3_i_encrypt_decrypt_p8, x4_i_encrypt_decrypt_p8,	64	output	Dữ liệu cần hoán vị đưa vào Ascon-p[8]
x0_o_encrypt_decrypt_p8, x1_o_encrypt_decrypt_p8, x2_o_encrypt_decrypt_p8, x3_o_encrypt_decrypt_p8, x4_o_encrypt_decrypt_p8	64	input	Dữ liệu đã hoán vị lấy ra từ Ascon-p[8]

Bảng 7. IO ports list của Ascon-Encryption_Decryption

3.3.6 Ascon Hash

Ascon Hash có tác dụng băm dữ liệu với kết quả là x0 có độ dài 64-bit sau mỗi lần hoán vị Ascon-p[12]. Với Ascon-Hash256 thì số lần băm sẽ là 4 lần, mỗi lần 64-bit, tổng số bit đạt được là 256-bit, tương đương với 32-byte dữ liệu. với Ascon-XOF128 thì số lần băm sẽ phụ thuộc vào CPU muốn băm bao nhiêu lần, mỗi lần 64-bit. Do đó, Ascon-XOF128 chính là phiên bản mở rộng dữ liệu băm của Ascon-Hash256, nên Ascon-Hash256 và Ascon-XOF128 sẽ dùng chung một hàm. Khi triển khai phần cứng, chỉ phát triển duy nhất hàm băm của Ascon-Hash256, để có được dữ liệu của Ascon-XOF128, chỉ cần thay đổi sel_type và lặp đi lặp lại nhiều lần không như Ascon-Hash256, có thể bé hơn, lớn hơn hay bằng 4 lần tùy theo độ dài yêu cầu.



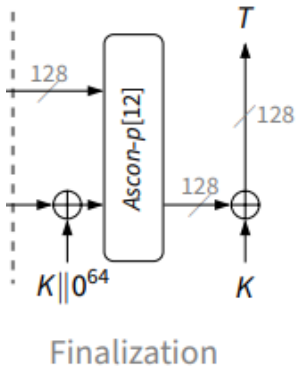
Hình 29. Ascon-Hash

Port	Width	Direction	Function
Clk, rst_n	1	input	Clock và reset
Process_en	1	input	Cho phép tín hiệu được ghi vào thanh ghi kết quả tính toán
Hash_out	64	Output	Dữ liệu băm
x0_i, x1_i, x2_i, x3_i, x4_i	64	input	Mã hóa của những data trước, với lần mã hóa đầu tiên thì input sẽ là giá trị Asorbs message cuối cùng
x0_o, x1_o, x2_o, x3_o, x4_o	64	Output	Kết quả hoán vị
x0_i_hash_p12, x1_i_hash_p12, x2_i_hash_p12, x3_i_hash_p12, x4_i_hash_p12	64	output	Dữ liệu cần hoán vị đưa vào Ascon-p[12]
x0_o_hash_p12, x1_o_hash _p12, x2_o_hash_p12, x3_o_hash_p12, x4_o_hash_p12	64	input	Dữ liệu đã hoán vị lấy ra từ Ascon-p[12]

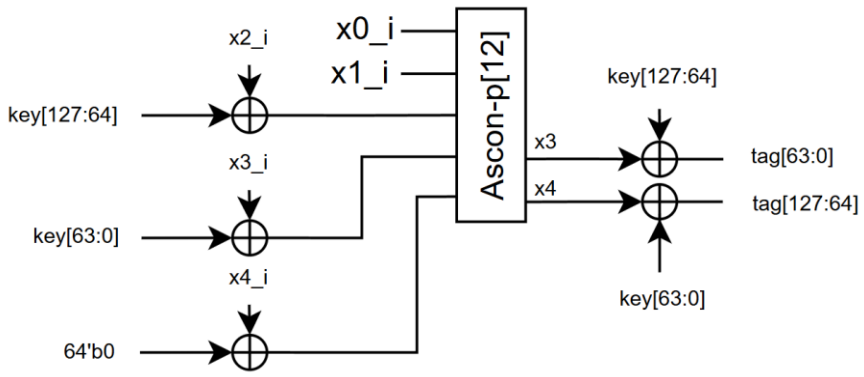
Bảng 8. IO ports list Ascon-Hash

3.3.7 Ascon Finalization

Khối Finalization tác dụng tạo ra Tag, Tag có tác dụng để kiểm tra lỗi trong quá trình mã hóa hoặc quá trình giải mã và quá trình xác minh, yêu cầu phải có Key secret giữa hai quá trình. Khối này chỉ có tác dụng trong khi sử dụng loại mã hóa Ascon-AEAD128



Hình 30. Ascon-final



Hình 31. Sơ đồ khối Ascon-final

Port	Width	Direction	Function
Clk, rst_n	1	input	Clock và reset
key	64	input	key
tag	64	Output	Dữ liệu tag
x0_i, x1_i, x2_i, x3_i, x4_i	64	input	Mã hóa của những data trước
x0_i_final_p12, x1_i_final _p12, x2_i_final _p12, x3_i_final _p12, x4_i_final _p12	64	output	Dữ liệu cần hoán vị đưa vào Ascon-p[12]
x0_o_final_p12, x1_o_final_p12, x2_o_final_p12, x3_o_final_p12, x4_o_final_p12	64	input	Dữ liệu đã hoán vị lấy ra từ Ascon-p[12]

Bảng 9. IO ports list Ascon-final

4. Tài liệu tham khảo

[1] Meltem Sönmez Turan, Kerry A. McKay, Jinkeon Kang, John Kelsey, "Ascon-Based Lightweight Cryptography Standards for Constrained Devices", NIST Special Publication 800-232, November 2024.

[2] Stefan Steinegger, Robert Primas, "A Fast and Compact RISC-V Accelerator for Ascon and Friends", 2020.

[3] M. Eichlseder, "Python implementation of Ascon," *GitHub*, Apr. 02, 2023. https://github.com/meichlseder/pyascon