TRIỂN KHAI THUẬT TOÁN MẬT MÃ AES TRÊN FPGA

Trần Văn Nghĩa*, Vũ Tất Điệp†, Phạm Minh Quý†, Nguyễn Tiến Minh†
Email: nghiamosmipt@gmail.com

Ngày tòa soạn nhận được bài báo: 02/11/2023
Ngày phản biện đánh giá: 18/06/2024
Ngày bài báo được duyệt đăng: 27/06/2024

DOI: 10.59266/houjs.2024.416

Tóm tắt: *Với sự phổ biến hiện nay của mạng máy tính, hệ thống truyền thông không dây, v.v., bảo mật dữ liệu đã trở thành một chủ đề đặc biệt quan tâm trong các cơ sở hạ tầng thông tin mà chúng ta đang xây dựng, đang sử dụng và đang dựa vào trong cuộc sống hàng ngày. Việc triển khai phần cứng mật mã phải đối mặt với những yêu cầu nghiêm ngặt về giá thành thấp và độ phức tạp tính toán lớn. Bài báo này trình bày một kiến trúc mã hóa/giải mã rất nhỏ gọn cho thuật toán tiêu chuẩn mã hóa tiên tiến (AES). Kiến trúc AES đề xuất là một kiến trúc đường ống, được triển khai hiệu quả trong FPGA giá thành thấp.*

Từ khóa: *Tiêu chuẩn mã hóa tiên tiến (AES), mật mã khối, FPGA, RS232, UART.*

I. Đặt vấn đề

Thông tin là một thứ vô cùng quý giá, nên có khả năng rất cao xảy ra các hoạt động truy cập thông tin trái phép hoặc thay đổi thông tin này bởi một người dùng không chủ ý. An toàn thông tin trong truyền thông với thông tin nhạy cảm không chỉ cần thiết cho quân đội, các cơ quan chính phủ mà còn cho cả trong lĩnh vực doanh nghiệp và các cá nhân. Đồng thời, sự ra đời của các siêu máy tính lượng tử cỡ lớn nhờ vào thành tựu khoa học công nghệ đã dẫn đến các thuật toán mật mã thông tin cũ như DES (data encryption standard) [1] không an toàn. Trong hai thập kỷ gần đây, các nhà nghiên cứu đã phát triển nhiều thuật toán mật mã. Tuy nhiên, AES (advanced encryption standard) được coi là thuật toán mật mã an toàn nhất và dễ dàng triển khai hơn cho cả phần cứng và phần mềm [2-5].

Cho đến nay, nhiều cách tiếp cận đã được đề xuất để tạo ra các kiến trúc AES nhỏ gọn, tiêu thụ công suất chip thấp và đạt tốc độ xử lý cao [2-7]. Trong số đó, tiêu tốn tài nguyên chip là một tiêu chí luôn được quan tâm cải thiện nhất.

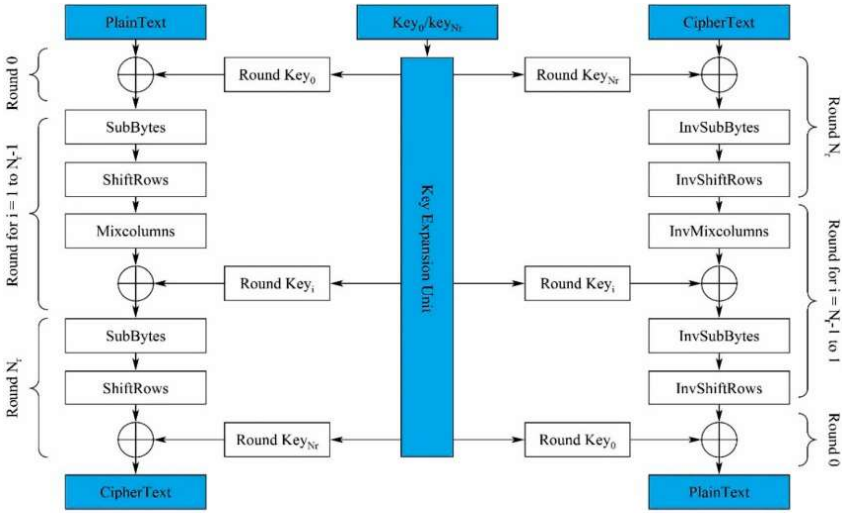
Trong khuôn khổ bài báo này, nhóm nghiên cứu sẽ trình bày quy trình thiết kế, đề xuất kiến trúc mã hóa/giải mã AES rất

* Học viện Phòng không – Không quân
† Trường Đại học Mở Hà Nội

nhỏ gọn, được triển khai một cách hiệu quả trong FPGA giá thành thấp. Cụ thể, phép nhân trường Galois trong phép biến đổi Mix-Columns và Inverse Mix-Columns sẽ được tối ưu hóa bằng phép dịch bit và cộng, trong khi là hoạt động đòi hỏi khắt khe nhất về lượng tiêu thụ tài nguyên chip và tốc độ xử lý.

II. Thuật toán AES

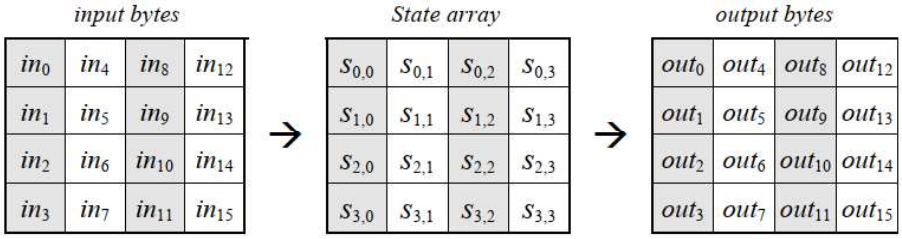
AES là một thuật toán mật mã đối xứng, trong đó một khóa mã được sử dụng cho cả hoạt động mã hóa và giải mã. AES có ba phiên bản, AES-128, AES-192 và AES-256, khác nhau về độ dài bit khóa. Số vòng hoạt động mã hóa và giải mã N_r cho mỗi phiên bản cũng khác nhau tương ứng 10, 12 và 14. Hình 1 thể hiện quá trình mã hóa và giải mã của thuật toán AES. Ngoại trừ vòng cuối cùng, tất cả các vòng sẽ có bốn hoạt động sau: SubBytes/ InvSubBytes, ShiftRows/InvShiftRows, MixColumns/InvMixColumns, và AddRoundKey.



Hình 1. Sơ đồ các khối của hệ thống

Phép biến đổi thay thế SubBytes/ InvSubBytes

Trước khi thực hiện quá trình mã hóa/giải mã, 128 bit bản rõ/bản mã được nhóm thành khối theo ma trận có bậc 4×4 được gọi là một trạng thái (state). Một state bao gồm bốn hàng byte, mỗi hàng chứa N_b byte, $N_b = 4$, có nghĩa là $0 \le c \le 3$. Trong khối state (S), mỗi byte riêng lẻ có hai chỉ số: Chỉ số hàng r, $0 \le r \le 3$, và chỉ số cột c, $0 \le c \le N_b - 1$. Khi bắt đầu mã hóa và giải mã, đầu vào (in_0 ... in_{15}) được ánh xạ vào state (Hình 2). Sau khi tất cả các biến đổi của khối state được hoàn thành, giá trị cuối cùng của nó được ánh xạ đến đầu ra (out_0 ... out_{15}).



Hình 2. Biểu diễn khối dữ liệu state

Phép biến đổi SubBytes() là một phép thay thế byte phi tuyến tính hoạt động trên từng byte riêng biệt của state để tạo ra một giá trị byte mới tạo thành bước quan trọng trong việc bảo toàn dữ liệu gốc khỏi bị truy cập trái phép. Mỗi byte trong ma trận state được coi là một đa thức trong Trường Galois GF(2^8). Tiếp theo, phép biến đổi affine và phép nhân ma trận được sử dụng để biến đổi byte như trong (1).

$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \quad (1)$$

trong đó $0 \le i \le 8$, b_i là bit thứ i của byte và c_i là bit thứ i của byte c với giá trị {63} hoặc {01100011}.

Bảng 1. Bảng tra Sbox

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Bảng 2. Bảng tra InvSbox

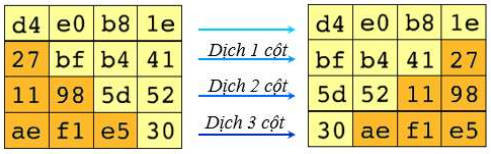
hex \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	8	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61

Phép biến đổi SubBytes() cũng có thể được tiến hành trực tiếp bằng cách sử dụng bảng thay thế Sbox [2] (Bảng 1). Sbox này có thể đảo ngược để nhận được InvSbox dùng cho giải mã (Bảng 2), được xây dựng bằng phép nghịch đảo nhân trong trường hữu hạn GF(2^8). Áp dụng phép biến đổi affine trên GF (2) được định nghĩa trong (2)

$$b_i' = b_{(i+2)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus d_i \quad (2)$$

Phép biến đổi dịch hàng ShiftRows/ InvShiftRows

Các hàng của ma trận khối dữ liệu state được dịch chuyển định kỳ. Ở đây, các hàng thứ hai, thứ ba và thứ tư được dịch chuyển sang trái một, hai và ba lần tương ứng, trong khi đó hàng đầu tiên vẫn không bị ảnh hưởng trong phép dịch này (Hình 3). Theo đó, các hàng được dịch chuyển vòng trái để hoàn thành hoạt động giải mã. Phép biến đổi dịch vòng hàng là quá trình khuếch tán đầu tiên trong thuật toán AES.



Hình 3. Chu trình ShiftRows

Phép biến đổi cột MixColumns/ InvMixColumns

Thao tác trộn cột được thực hiện ở cấp cột thông qua phép nhân ma trận. Hoạt động MixColumns/InvMixColumns

được biểu thị trong các biểu thức (3) và (4) tương ứng.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad (3)$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0\boldsymbol{b} & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad (4)$$

với $0 \le c < N_b$.

Phép nhân được thực hiện từng cột (4 byte). Mỗi cột được nhân với ma trận nên tổng cộng 16 phép nhân là được yêu cầu. Phép biến đổi này dựa trên phép nhân trường Galois (GF). Mỗi byte trong cột được thay thế bằng một giá trị khác là một hàm của tất cả bốn byte trong cột đã cho.

Dễ thấy rằng, mỗi byte trong cột được thay thế bằng một giá trị khác là một hàm của tất cả bốn byte trong cột đã cho. Do đó, bằng việc thực hiện phép nhân trong trường hữu hạn Galois, mỗi khối 4 byte đầu vào sẽ cho một khối 4 byte ở đầu ra với tính chất mỗi byte ở đầu vào đều ảnh hưởng tới cả 4 byte đầu ra. Cùng với bước ShiftRows, MixColumns đã tạo ra tính chất khuếch tán của thuật toán.

Phép biến đổi AddRoundKey

Thuật toán AES gồm nhiều vòng mã hóa liên tiếp nhau. Mỗi vòng mã hóa có một mật mã riêng được gọi là khóa con (Round Key) có cùng kích thước với khối dữ liệu đang được xử lý và được phát sinh từ bảng mã khóa cơ sở (Cipher Key). Phép biến đổi AddRoundKey sẽ thực hiện kết hợp khóa con với các khối state hiện thời để tạo nên khối state mới. Quá trình kết hợp được thực hiện bằng phép tính XOR từng bít của khóa con với khối dữ liệu như trong (5).

$$\left[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}\right] = \left[S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}\right] \oplus \left[w_{round} * N_b\right] \quad (5)$$

Quy trình sinh khóa KeyExpansion

Quy trình sinh khóa để tạo ra tổng số $N_b(N_r+1)$ byte từ 4 byte (32 bit). Thuật toán yêu cầu một tập N_b từ ban đầu, và mỗi vòng trong số N_r vòng yêu cầu N_b từ của dữ liệu khóa. Kết quả sinh khóa bao gồm một mảng tuyến tính các từ 4 byte, được ký hiệu là w[i], $0 \le i < N_b(N_r+1)$

$$w[i] = \text{SubWord}\left(\text{Rotword}\left(w[i-1]\right)\right) \oplus w[i-N_k] \oplus \text{Rcon} \quad (6)$$

với $i \ge N_1$, $N_1 = \{4, 6, 8\}$ với AES kích thước 128, 192, 256 tương ứng.

Trong đó, RotWord() thực hiện thao tác dịch chuyển vòng các byte thành phần trong một từ, nghĩa là các byte (a, b ,c, d) của từ sẽ thu được một từ các byte (b, c, d, a) ở đầu ra trong quá trình mã hóa, hoặc (d, a, b, c) khi giải mã, SubWord() áp dụng phép thay thế dựa trên bảng Sbox hay InvSbox, Rcon là một hằng số cho trước (7).

$$\begin{cases} \text{Rcon}[i] = (\text{RC}[i], \{00\}, \{00\}, \{00\}) \\ \text{RC}[i] = x^{(i-1)} \\ 1 \le i \le \dfrac{N_r N_b}{N_k} \end{cases} \quad (7)$$

III. Mô tả triển khai phần cứng thuật toán AES trên FPGA

Để tận dụng khả năng xử lý song song đầy đủ của FPGA, khối dữ liệu state cần phải thực hiện tất cả các phép biến đổi trên 128 bit. Triển khai phép thay thế SubBytes/InvSubBytes chiếm nhiều tài nguyên logic nhất vì nó là một thao tác tra cứu bảng, được thực hiện trong FPGA. Bảng Sbox được triển khai dưới dạng một hàm Sbox() được mô tả trong một thư viện (Hình 4, tương tự với InvSbox). Mỗi byte trong 16 byte của state đầu vào được trả về giá trị tương ứng với bảng 1. Như vậy, tổng cộng 16 byte trong state sẽ yêu cầu 16 bảng tra.

Dễ dàng nhận thấy rằng, đối với ShiftRows/InvShiftRows, hàng đầu tiên được gán trực tiếp đến hàng đầu ra, 3 hàng còn lại được gán sang các vị trí mới tương ứng. Như vậy, phép biến đổi ShiftRows/ InvShiftRows hoàn toàn không làm mất tài nguyên chip FPGA.

```
function SboxLookup (xy : s8)
return s8 is
  variable t : s8;
begin
  case xy is
    when x"00" => t := x"63";
          ...
    -- All 256 combinations from the Sbox table
  end case;
return t;
end function SboxLookup;
```

Hình 4. Mã nguồn hàm đóng gói bảng tra cứu Sbox

Triển khai phần cứng phép biến đổi cột MixColumns/InvMixColumns sẽ tiêu tốn nhiều bộ nhân trong chip FPGA (3), (4) làm giảm nghiêm trọng tốc độ xử lý của FPGA. Tuy nhiên có thể thấy rằng, ma trận trong (3) và (4) xử lý với phép nhân giữa các giá trị cột và tất cả ba hằng số {01}, {02}, {03} trên trường Galois GF. Phép nhân với {02} có thể được thực hiện ở mức byte dưới dạng dịch trái một bit và XOR từng bit có điều kiện với {1b}. Thành ra, Triển khai phần cứng của MixColumns/InvMixColumns chỉ sử dụng các phần tử logic XOR (Hình 5).

```
function PolyMult (
          a : s8;
          b : s8)
return s8 is
  variable t : s8;
  variable andMask : s8;
begin
  andMask := b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7);
  case a(3 downto 0) is
    when "0001" => t := b;
    when "0010" => t := b(6 downto 0) & '0' xor (("00011011") and andMask);
    when "0011" => t := b(6 downto 0) & '0' xor (("00011011") and andMask) xor b;
    when others => t := (others => '0');
  end case;
return t;
end function PolyMult;
```

Hình 5. Mã nguồn thực hiện các phép nhân trên trường GF trong FPGA

Phép biến đổi AddRoundKey sử dụng các phần tử logic XOR thực hiện biểu thức (5). Module sinh khóa và đảo ngược quá trình sinh khóa sử dụng thao tác tra bảng Sbox/InvSbox và các phần tử logic XOR.

IV. Thử nghiệm thiết kế thuật toán AES trên bo mạch FPGA

Sơ đồ khối thử nghiệm, đánh giá thiết kế khối mã hóa/giải mã thuật toán AES được trình bày trong Hình 6. Thử nghiệm được thực hiện với thuật toán AES 128 dựa trên chip Xilinx Artix-7 FPGA XC7A35T-1CPG236C. Thiết kế khối mã hóa/giải mã thuật toán AES được thực hiện bằng ngôn ngữ mô tả phần cứng VHDL. Giao diện người dùng truyền thông nối tiếp được tạo bằng ngôn ngữ lập trình C# có thể gửi và nhận dữ liệu với khối AES128 thông qua cổng RS232 với bo mạch FPGA. Bo mạch Basys 3 Artix-7 FPGA Board là thành phần phần cứng chính được sử dụng trong thử nghiệm thiết

kế. Cổng USB để lập trình FPGA và đồng thời làm truyền thông RS232 với máy tính tạo nên sự linh hoạt cho bo mạch. Một công tắc chuyển mạch được sử dụng để bắt đầu quá trình mã hóa/giải mã.

Mô-đun mã hóa có các cổng vào/ra sau:

Inputs

clk - Xung nhịp với tần số 50 MHz.

rst – reset hệ thống.

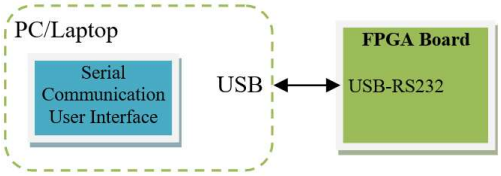
plaintext(127: 0) - Dữ liệu đầu vào được sử dụng để mã hóa (bản rõ).

round(3: 0) - Số vòng mã hóa nhận được từ bộ đếm bên ngoài.

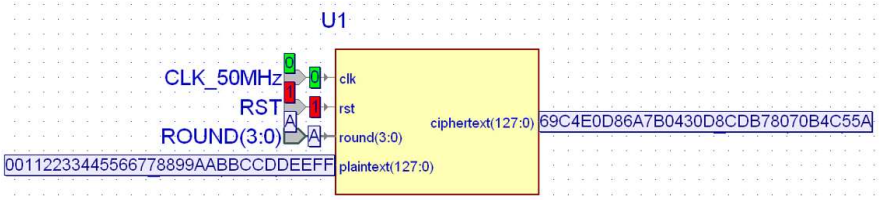
Output

ciphertext(127: 0) - Bản được mã hóa là kết quả của quá trình mã hóa.

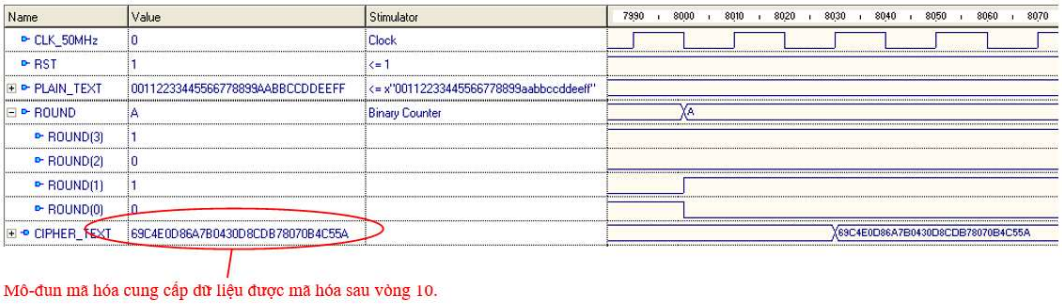
Mô-đun mã hóa có cổng đầu vào plaintext(127:0) nhận 128 bit dữ liệu từ máy tính. Hình 7 và 8 biểu diễn cho mô-đun mã hóa trong giai đoạn mô phỏng. Bản rõ đầu vào plaintext(127:0) nhận dữ liệu ở dạng hexa và bản mã đầu ra ciphertext(127:0) trả về dữ liệu được mã hóa.



Hình 6. Kết nối thử nghiệm thiết kế



Hình 7. Mô-đun mã hóa trong 10 vòng



Hình 8. Kết quả mô phỏng VHDL mô-đun mã hóa

Dữ liệu bản rõ cho mô phỏng được thiết lập với giá trị 00112233445566778899aabbccddeeff ở mã hexa. Dữ liệu xuất ra sau 10 vòng mã hóa là 69c4e0d86a7b0430d8cdb78070b4c55a.

Mô-đun giải mã bao gồm các cổng vào/ra sau đây:

Inputs

clk - Xung nhịp với tần số 50 MHz.

rst – reset hệ thống.

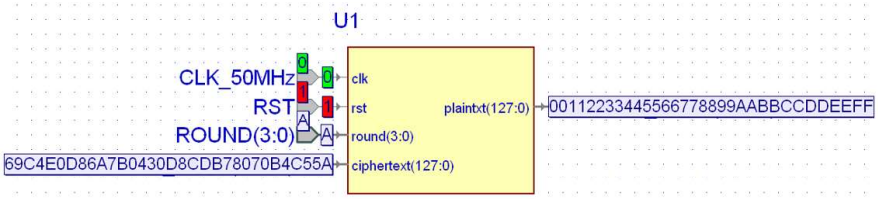
ciphertext(127:0) - Dữ liệu đầu vào được sử dụng để giải mã (bản mã).

round(3: 0) - Số vòng mã hóa nhận được từ bộ đếm bên ngoài.

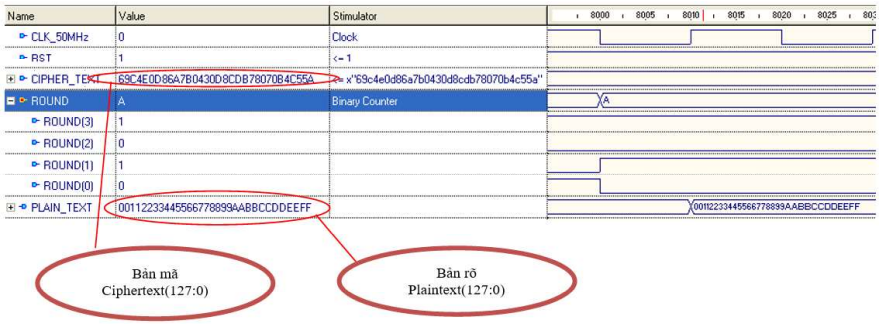
Output

plaintext(127:0) - Bản được giải mã là kết quả của quá trình giải mã.

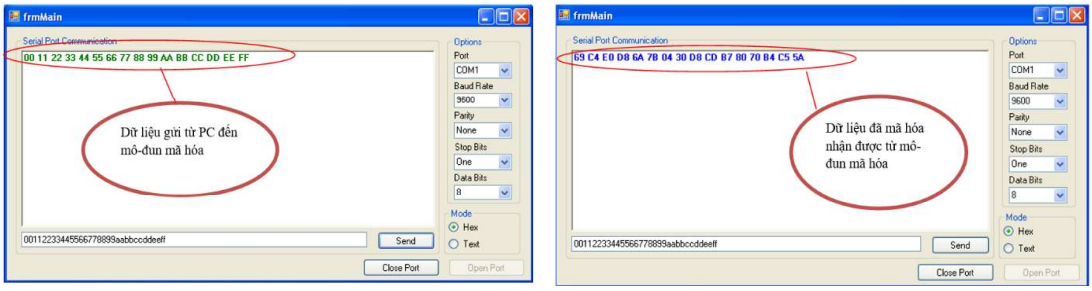
Để giải mã, sử dụng dữ liệu được tạo ra từ mã hóa (hình 7 và 8). Hình 9 và 10 minh họa kết quả mô phỏng VHDL mô-đun giải mã, trong đó bản mã ciphertext(127:0) lấy dữ liệu được mã hóa từ mô-đun mã hóa và dữ liệu bản rõ plaintext(127:0) được tạo ra từ giải mã. Từ các Hình 7-10 có thể thấy rằng, các kết quả nhận được từ các mô-đun mã hóa/giải mã là hoàn toàn tương ứng với tiêu chuẩn AES đưa ra [2].



Hình 9. Mô-đun giải mã trong 10 vòng

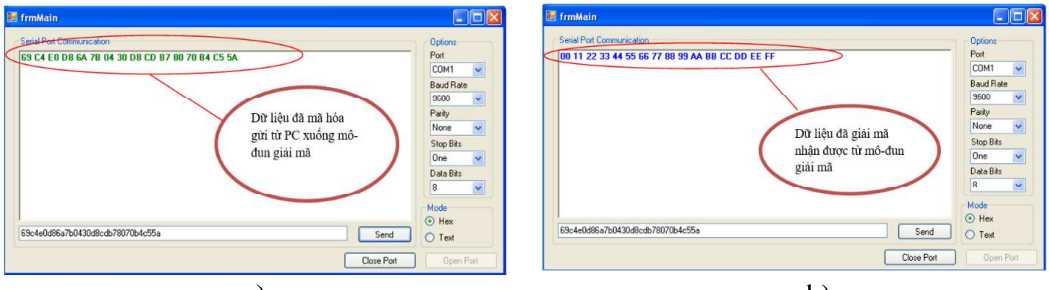


Hình 10. Kết quả mô phỏng VHDL mô-đun giải mã



a) b)

Hình 11. Đánh giá thiết kế trên FPGA: Dữ liệu được gửi từ PC đến mô-đun mã hóa (a); Dữ liệu được mã hóa từ FPGA gửi lên PC (b)



a) b)

Hình 12. Đánh giá thiết kế trên FPGA: Dữ liệu được gửi từ PC đến mô-đun giải mã (a); Dữ liệu được giải mã từ FPGA gửi lên PC (b)

Để kiểm tra hệ thống mã hóa, cần thiết lập trình thiết kế lên bo mạch Basys 3 FPGA. Các công tắc chuyển mạch cũng được đưa vào trong thiết kế để thực hiện các nội dung kiểm tra tính đúng đắn thiết kế. Sau khi lập trình, tiến hành kết nối bo mạch với máy tính và thay đổi các chuyển mạch thích hợp để thực hiện kiểm tra. Các kết quả đánh giá thiết kế trên bo mạch thể hiện trên các hình 11-14. Các kết quả nhận được này phù hợp với mô phỏng và thử nghiệm của tiêu chuẩn AES [2].

Hình 13 đưa ra tổng hợp các tài nguyên FPGA tiêu dùng cho hai quá trình mã hóa và giải mã. Từ hình 13 có thể thấy rằng, cấu trúc thiết kế đề xuất cho mã hóa/ giải mã chiếm một lượng tương đối nhỏ tài nguyên chip FPGA.

Đối với mã hóa:
Selected Device : XC7A35T-1CPG236
Number of Slices: 1231 out of 5200 24%
Number of Slice Flip Flops: 2289 out of 41600 6%
Number of Block RAM/FIFO: 4 out of 50 8%
Đối với giải mã:
Selected Device : XC7A35T-1CPG236
Number of Slices: 1195 out of 5200 23%
Number of Slice Flip Flops: 2040 out of 9312 21%
Number of Block RAM/FIFO: 4 out of 50 8%

Hình 13. Tổng hợp tài nguyên tiêu thụ của chip FPGA

IV. Kết luận

Trong bài báo này, kiến trúc phần cứng dựa trên FPGA cho mã hóa/giải mã thuật toán AES đã được đề xuất. Cấu trúc phần cứng đề xuất nhằm triển khai hiệu quả cho phép xử lý cột của khối dữ liệu state MixColumns/ InvMixColumns, loại trừ yêu cầu các bộ nhân trong thiết kế.

Nhóm tác giả đã tiến hành triển khai thử nghiệm thuật toán AES128 trong mô phỏng và trên bo mạch FPGA. Kết quả thiết kế thể hiện hiệu quả về tài nguyên chiếm dụng trong chip FPGA. Khối mã hóa/giải mã FPGA đã được kiểm tra và xác minh bằng ứng dụng phần mềm giao diện người dùng được triển khai trên PC giao tiếp với bo mạch thông qua RS232 để thực hiện truyền thông. Các kết quả thử nghiệm trên chip FPGA cho thấy hoàn toàn phù hợp với bộ dữ liệu thử nghiêm trong tiêu chuẩn AES.

Tài liệu tham khảo:

[1]. Zeebaree, S.R.M. DES encryption and decryption algorithm implementation based on FPGA. *Indonesian Journal of Electrical Engineering and Computer Science*, 2020, 18, 2, 774–781, doi: 10.11591/ijeecs.v18.i2.pp774-781.

[2]. NIST, Advanced Encryption Standard, Federal Information Processing Standards Publication 197, May 2023.

[3]. Farooq, U.; Aslam, M.F. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA. J. King Saud Univ. Comput. Inf. Sci. 2017, 29, 295–302.

[4]. Visconti, P.; Capoccia, S.; Venere, E.; Velázquez, R.; Fazio, R.D. 10 Clock-Periods Pipelined Implementation of AES-128 EncryptionDecryption Algorithm up to 28 Gbit/s Real Throughput by Xilinx Zynq UltraScale+ MPSoC ZCU102 Platform. Electronics 2020, 9, 1665.

[5]. Shahbazi, K.; Ko, S.-B. Area-Efficient Nano-AES Implementation for Internet-of-Things Devices. IEEE Trans. Very Large Scale Integr. VLSI Syst. 2021, 29, 136–148.

[6]. Sikka, P.; Asati, A.; Shekhar, C. High-throughput field-programable gate

array implementation of the advanced encryption standard algorithm for automotive security applications. J. Ambient. Intell. Humaniz. Comput. 2021, 12, 7273–7279.

[7]. Murugan, C.A.; Karthigaikumar, P.; Priya, S.S. FPGA implementation of hardware architecture with AES encryptor using sub-pipelined S-box techniques for compact applications. Automatika 2020, 61, 682–693.

IMPLEMENTATION OF AES ENCRYPTION ALGORITHM ON FPGA

Tran Van Nghia[‡], Vu Tat Diep[§], Pham Minh Quy[§], Nguyen Tien Minh[§]

Abstract: *With the current ubiquity of computer networks, wireless communication systems, etc., data security has become a topic of particular concern in the information infrastructures we are building, using, and counting on in daily life. Cryptographic hardware implementations face strict requirements of low cost and high computational complexity. This paper presents a compact encryption/decryption architecture for an advanced encryption standard (AES) algorithm. The proposed AES architecture is a pipelined one, which is efficiently implemented in a low-cost FPGA.*

Keywords: *Advanced encryption standard (AES), cryptography, FPGA, RS232, UART.*

[‡] Air Force-Air Defense Academy

[§] Hanoi Open University