2024 Digital IC Design

Homework 5: Advanced Encryption Standard

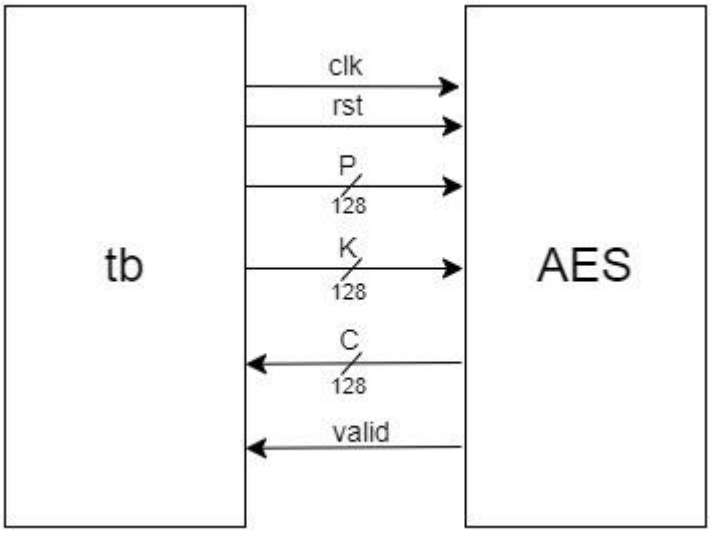
1. Introduction:

Information security issues occur everywhere in modern society, for example, the password of online apps, the balance shown in your bank account, or the selfies stored in your social software. How to protect our personal information is worthy of discussion. In 2001, the **National Institute of Standards and Technology** (**NIST**) in America announced the standard encryption scheme **Advanced Encryption Standard** (**AES**). It is widely used in every communication scenario in computer science, including private messages and data exchange. E.g., Secure Shell connections, Image Encryption, etc. Also, AES has several operation modes, including ECB, CBC, GCM, etc. However, in this practice, we neglect the variations of these modes, providing the pure AES-128 algorithm for implementing it. We expect that you could consider both the **Area** and **Timing** two criteria and design an AES-128 architecture for accelerating the provided algorithm.

2. Specification:

2.1. System specifications

We provide a testbench to check the correctness of your design. In this homework, you should pass the functional and gate-level simulations to get the whole score. In other words, the synthesized circuit is needed to be submitted. At the same time, the input delay will be considered, so you must remember to add Synopsys design constraint (SDC) before synthesis to avoid hold-time violations during gate-level simulation.



2.2. Input/Output

The goal of this homework is to implement an AES-128 algorithm. First, instead of considering different operation modes and manipulating the input format, your hardware architecture will directly read the 128-bit data and start encrypting. The input data includes plaintext (P in the table) and the initial key (K in the table). Then, the output will be expected to be 128-bit encrypted data "C" with the "valid" signal representing the data in effect. Also, because this is a single architecture design, only one clock is the primary clock source, and one reset signal resets the circuit, active high, synchronously.

Table I I/O specification of the AES

Signal	I/O	width	Description
clk	I	1	Input port connected to clock source. Active when positive edge.
rst	I	1	Input port to reset the circuit. Active when positive edge, synchronous reset.
P	I	128	Plaintext
K	I	128	Initial Key
valid	O	1	The signal tells the testbench your data is valid, and it will be

			checked. (active high)
C	O	128	Ciphertext

2.3. AES-128 algorithm

1. Plaintext and Initial Key are involved in AddRoundKey.
2. AddRoundKey results perform SubBytes.
3. SubBytes results perform ShiftRows.
4. ShiftRows results perform MixColumns.
5. MixColumns results perform AddRoundKey, with Subkey n.
6. Repeat 2~5 until the second last round.
7. Repeat 2~4 in the last round.

(Subkey n is generated by KeyExpansion)

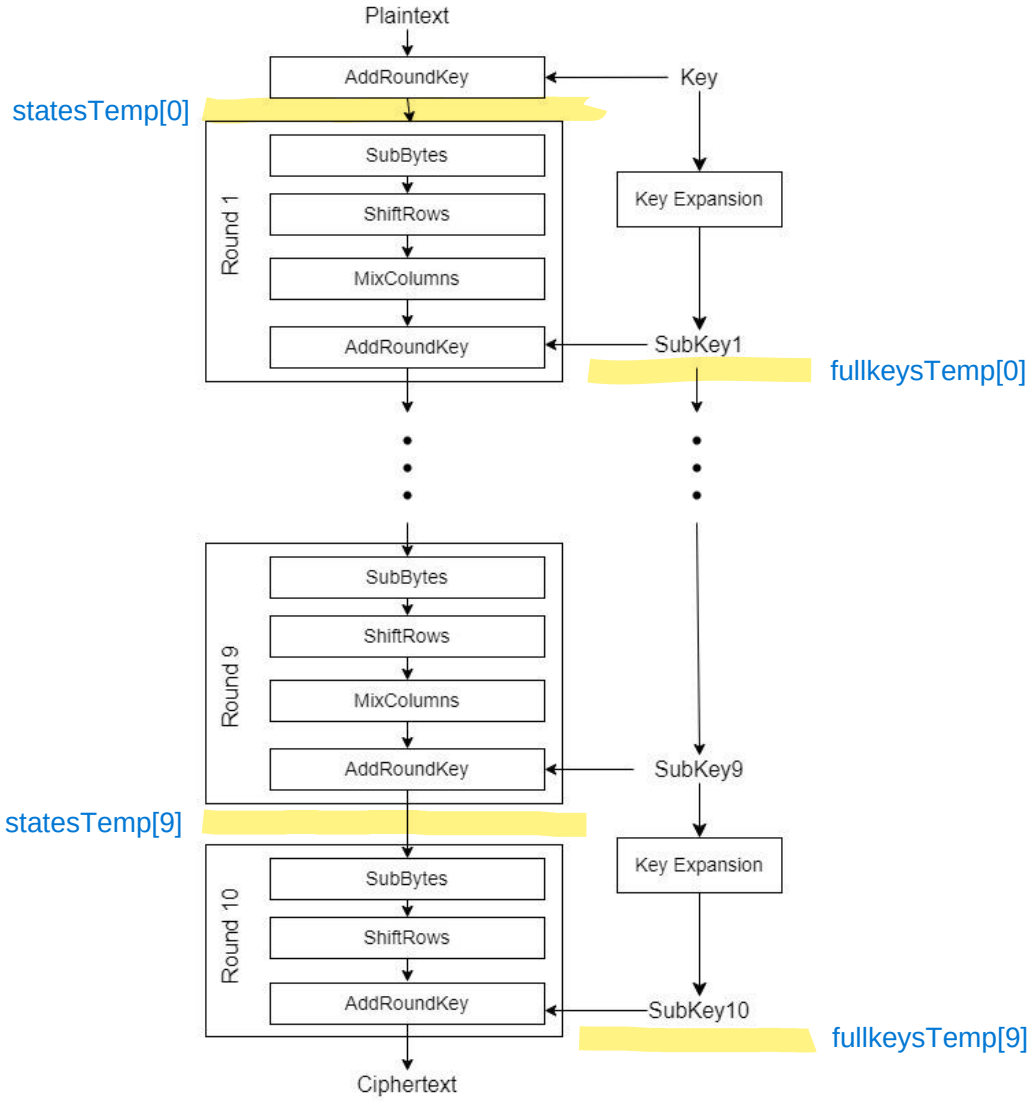


Fig. 1. AES Encryption path

2.4. State Array

Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of 4 rows of bytes, each containing 4 bytes.
The process of transforming input data into a state array and producing output data is illustrated in Fig.2

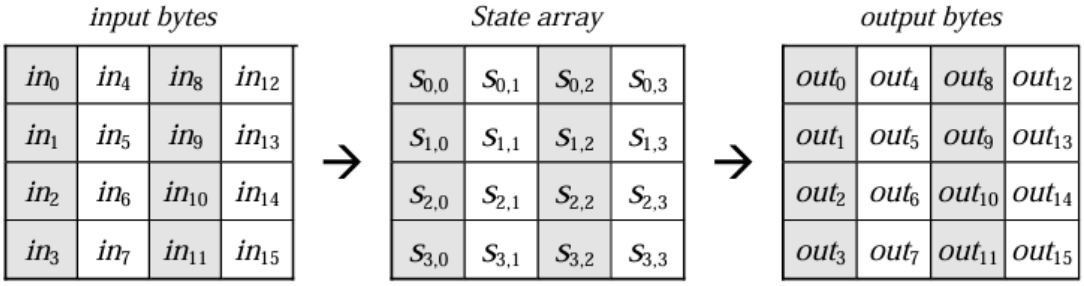


Fig. 2. State array input and output

2.5. Encryption

At the start of the Encryption, the input is copied to the State array. After an initial Round Key addition, the State array is transformed by implementing a round function 10 times with the final round differing slightly from the first 9 rounds(Fig. 1.). The final State is then copied to the output.
The regular round function contains 4 steps: SubBytes, ShfitRows, MixColumns, and AddRoundKey.

2.5.1. SubBytes

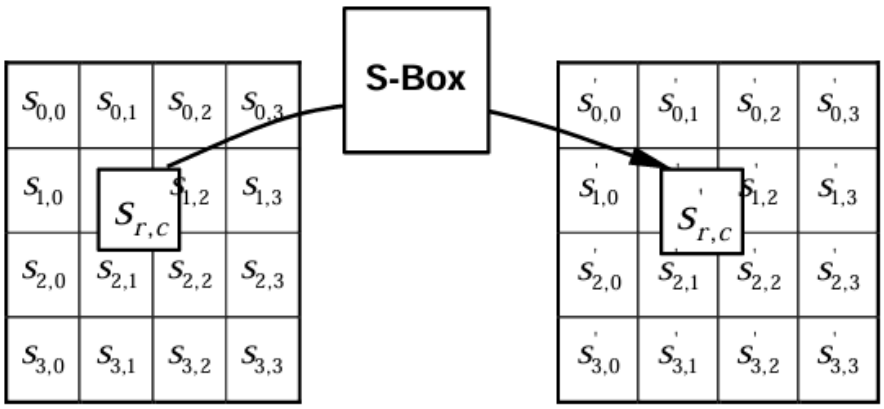


Fig. 3. SubBytes applies the S-box to each byte of the State.

The SubBytes transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution

table (S-box). This S-box (Fig. 4.), which is invertible, is constructed by composing two transformations: inverse, and affine transformation.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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

Fig. 4. S-box: substitution values for the byte xy (in hexadecimal format).

For example, if $s_{1,1}=\{53\}_{16}$, then the substitution value would be determined by the intersection of the row with index '5' and the column with index '3' in Fig. 4. This would result in $s'_{1,1}$ having a value of $\{ed\}_{16}$.

2.5.2. ShiftRows

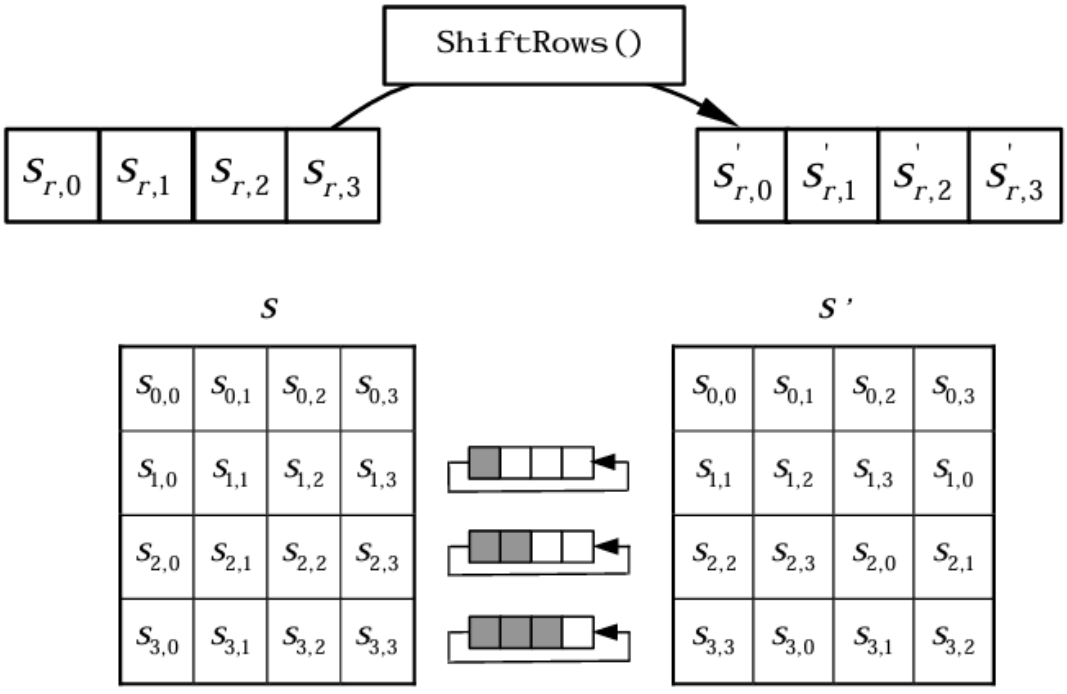


Figure 5. ShiftRows cyclically shifts the last three rows in the State.

In the ShiftRows transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, r = 0, is not shifted.

2.5.3. MixColumns

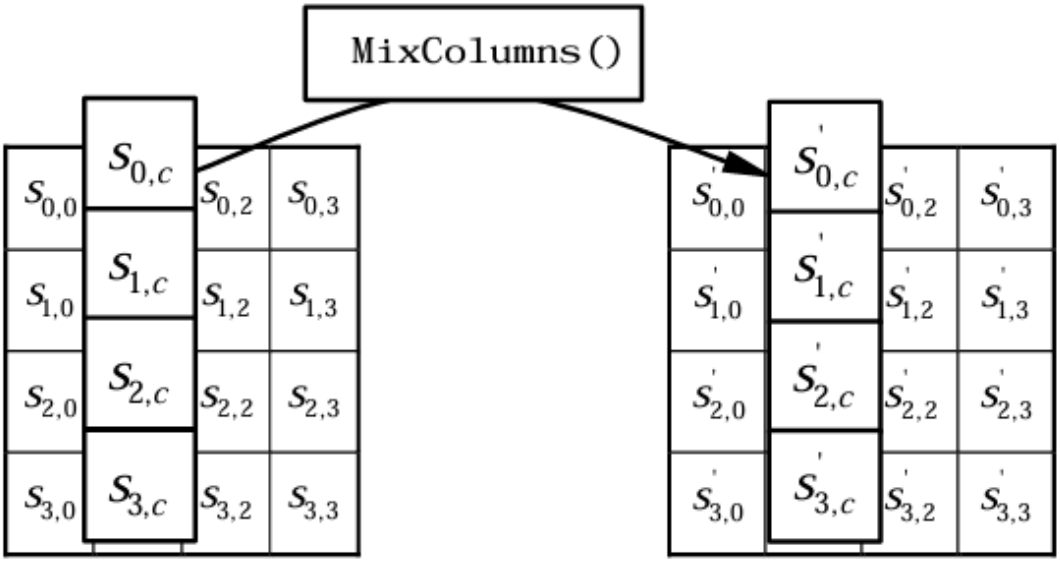


Fig. 6. MixColumns operates on the State column-by-column.

The MixColumns() transformation operates on the State column-by-column, treating each column as a four-term polynomial as described in Appendix A. The MixColumns() can be written as a matrix multiplication in GF(2^8).

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

2.5.4. AddRoundKey

In the AddRoundKey() transformation, a Round Key is added to the State by a simple bitwise XOR operation.

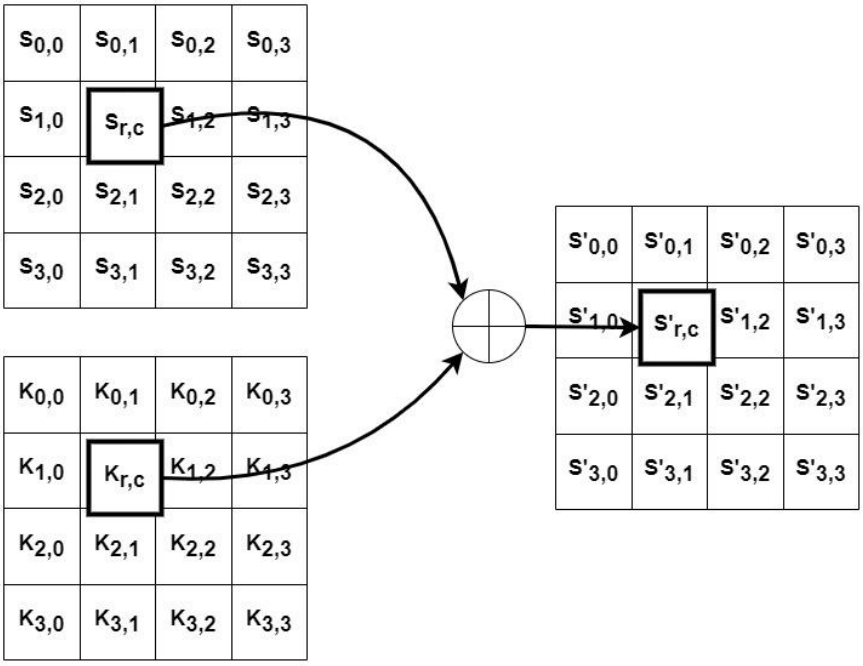
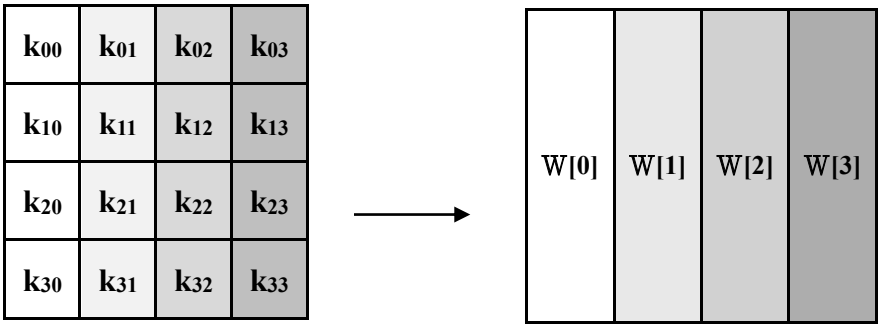


Fig. 7. AddRoundKey XORs each bytes of the State with the key schedule.

2.5.5. Key Expansion

In the AES-128 algorithm, the KeyExpansion function expands the key into an additional 10 sets of RoundKeys for use by Rounds 1 to 10 of AddRoundKey (the round0 uses the initial key). This function also includes rotation and SubBytes operations to increase data entropy.

First, the input key is divided into word arrays (32 bits each) labeled as W[0] to W[3].



Assuming the RoundKey to be used for the r-th round AddRoundKey is W[4r] to W[4r+3], where $0 \leq r \leq 10$. The next round's RoundKey (W[4r+4] to W[4r+7]) is derived from the current

round's RoundKey (W[4r] to W[4r+3]).

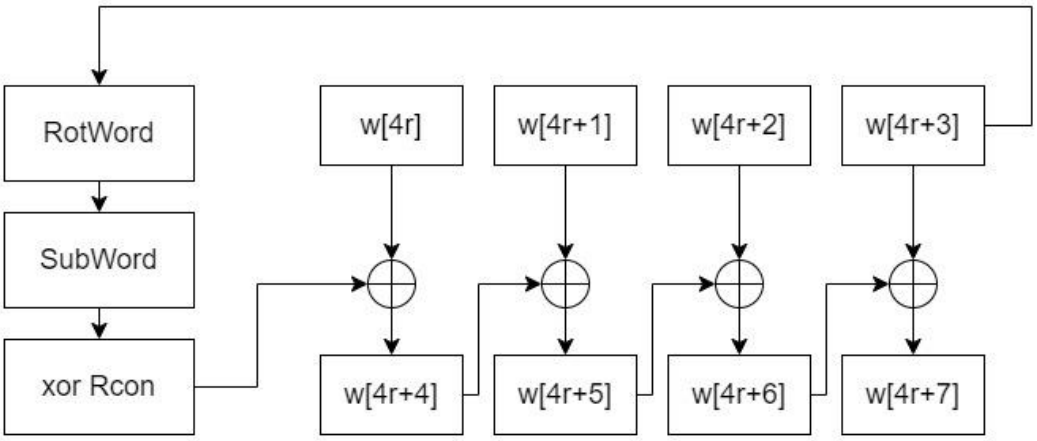


Fig. 8. Key Expansion data flow.

2.5.5.1. SubWord

The function SubWord is a function that takes a four-byte input word and applies the S-box to each of the four bytes to produce an output word.

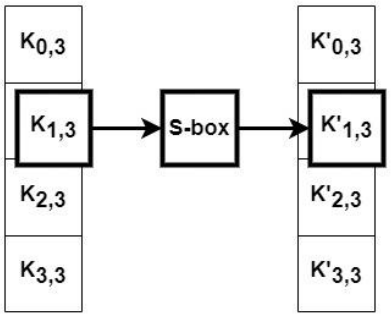


Fig. 9. SubWord applies the S-box to each byte of the word.

2.5.5.2. RotWord

The function RotWord takes a word $[a_0, a_1, a_2, a_3]$ as input, performs a cyclic permutation, and returns the word $[a_1, a_2, a_3, a_0]$.

2.5.5.3. Rcon

The round constant word array, Rcon[i], contains the values in Table II.

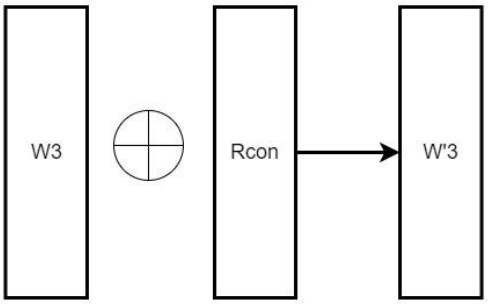


Fig. 10. XOR Rcon with W[4r+3] after RotWord and SubWord.

Table II Rcon value for each round

Round	Rcon
1	0x 01000000
2	0x 02000000
3	0x 04000000
4	0x 08000000
5	0x 10000000
6	0x 20000000
7	0x 40000000
8	0x 80000000
9	0x 1b000000
10	0x 36000000

2.6. Timing Diagram

After the system is reset, the plaintext data will input one by one.

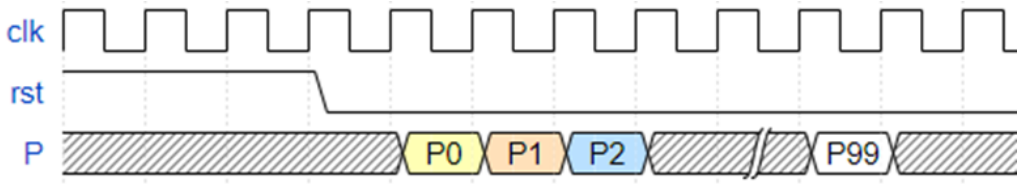


Fig. 11. Input timing diagram.

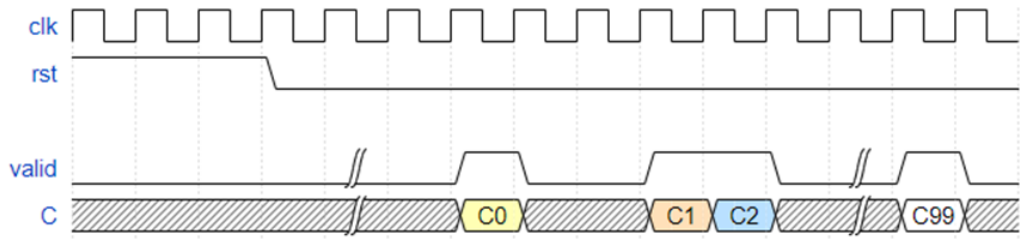


Fig. 12. Output timing diagram.

3. Scoring:

3.1. Functional Simulation **[60%]**

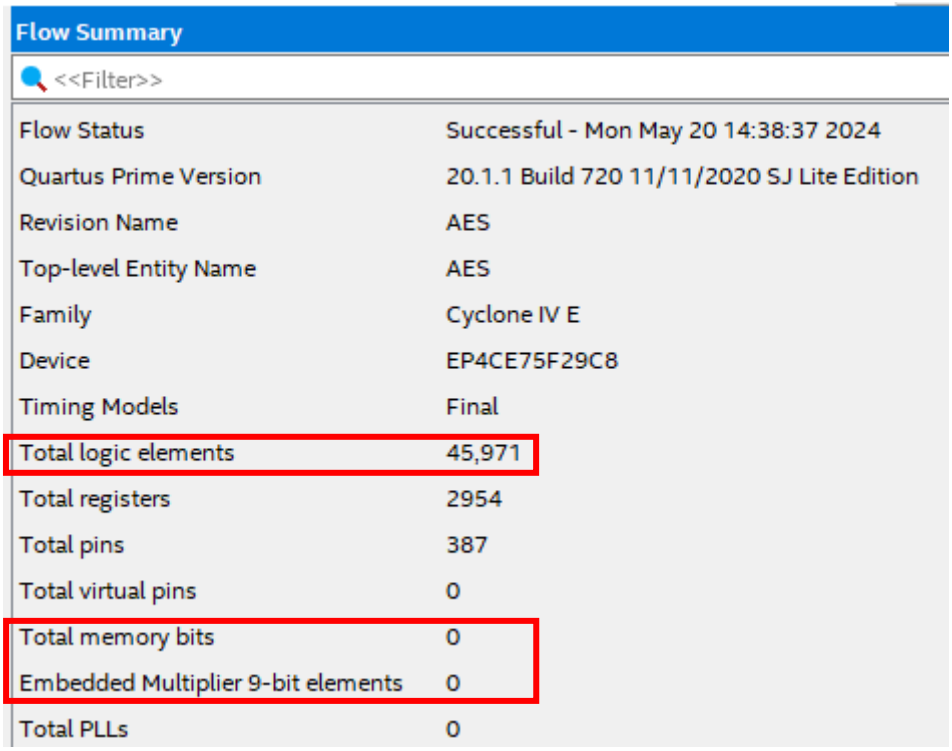
3.2. Gate level Simulation **[20%]**

Your code should be synthesizable. After it is synthesized in Quartus, a file named AES.vo will be obtained.

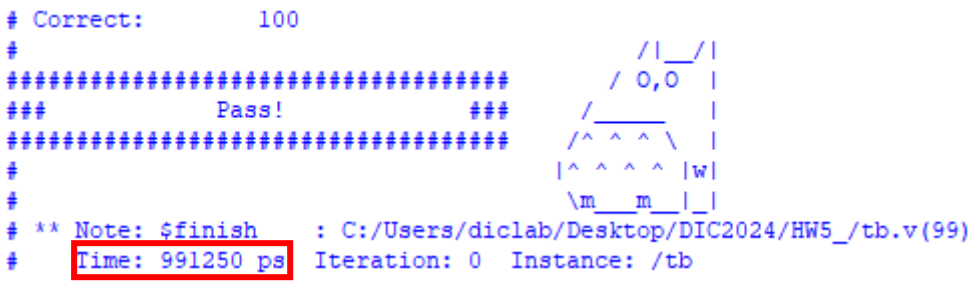
DEVICE：Cyclone IV E - EP4CE75F29C8

3.3. Performance **[20%]**

The scoring standard: (The smaller, the better)
Scoring = Area cost * Timing cost
Area cost = Total logic elements + total memory bits + 9*embedded multiplier 9-bit elements
Timing cost = simulation time



Flow Summary	
<<Filter>>	
Flow Status	Successful - Mon May 20 14:38:37 2024
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	AES
Top-level Entity Name	AES
Family	Cyclone IV E
Device	EP4CE75F29C8
Timing Models	Final
Total logic elements	45,971
Total registers	2954
Total pins	387
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0



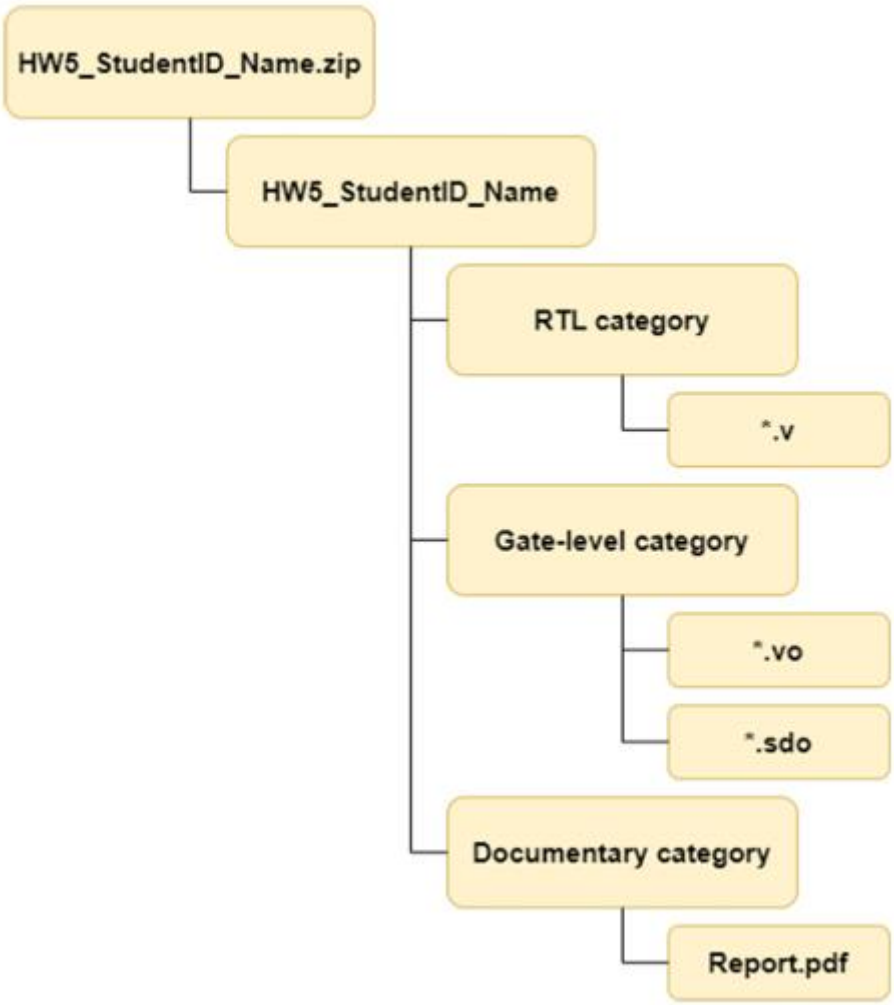
4. Submission

4.1. Submitted files

Please submit your files, followed by the illustration below.

For example, HW5_P78121506_廖國佑.zip. (Note: if your operating system is MACOS, please remove the directory "__MACOSX" when compressing your files.

	RTL category
*.v	All of your Verilog RTL code
	Gate-level category
*.vo	Gate-Level netlist generated by Quartus
*.sdo	SDF timing information generated by Quartus
	Documentary category
*.pdf	The report file of your design (in pdf).



5. Report File

Please follow the specifications of the report. You are asked to describe how the circuit is designed as detailed as possible.

5.1. Note

Please submit your .zip file to folder HW5 in moodle.

Deadline: 2024/6/24 23:55 p.m.

If you have any problem, please contact TA by email diclab65a01tas@gmail.com

Appendix A. Finite field

All bytes in the AES algorithm are interpreted as finite field elements. Finite field

elements can be added and multiplied, but these operations are different from those used for numbers. The following subsections introduce the basic mathematical concepts.

A.1 Polynomials representation in GF(2^8)

All byte values in the AES algorithm will be presented as the concatenation of its individual bit values (0 or 1) between braces in the order $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. These bytes are interpreted as finite field elements using a polynomial representation:

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$$

For example, $\{F5\}_{16} = \{11110101\}_2$ identifies the specific finite field element $x^7 + x^6 + x^5 + x^4 + x^2 + 1$

A.2 Addition

The addition of two elements in a finite field is achieved by "adding" the coefficients for the corresponding powers in the polynomials for the two elements. The addition is performed with the XOR operation.

For example, the following expressions are equivalent to one another:

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$(polynomial notation)

$\{01010111\}_2$ xor $\{10000011\}_2 = \{11010100\}_2$ (binary notation);

$\{57\}_{16}$ xor $\{83\}_{16} = \{d4\}_{16}$ (hexadecimal notation).

A.3 Multiplication

In the polynomial representation, multiplication in GF(2^8) (denoted by •) corresponds with the multiplication of polynomials modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm, this irreducible polynomial is

$m(x) = (x^8 + x^4 + x^3 + x + 1)$

For example, $\{57\}_{16} \bullet \{83\}_{16} = \{c1\}_{16}$, because

$\{57\}_{16} \bullet \{83\}_{16} = (x^6 + x^4 + x^2 + x + 1) \bullet (x^7 + x + 1)$

$= x^{13} + x^{11} + x^9 + x^8 + x^7 +$

$x^7 + x^5 + x^3 + x^2 + x +$

$x^6 + x^4 + x^2 + x + 1$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

$$(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)\text{mod}(x^8+x^4+x^3+x+1)$$

$$= x^7+x^6+1 = \{c1\}_{16}$$

$$
\begin{array}{r}
 & & & & & x^5 & & +x^3 & \\
\hline
x^8+x^4+x^3+x+1 \;\big)\; x^{13}+x^{11}+x^9+x^8 & & & +x^6+x^5+x^4+x^3+1 \\
\underline{x^{13} \quad +x^9+x^8 \quad +x^6+x^5} & & & & & & & & \\
\hline
 & x^{11} & & & & & & +x^4+x^3+1 & \\
 & \underline{x^{11}} & & +x^7+x^6 & & & & & \\
\hline
 & & & x^7+x^6 & & & & & +1
\end{array}
$$

Appendix B - Key Expansion Examples

Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

w_0 = **2b7e1516** w_1 = **28aed2a6** w_2 = **abf71588** w_3 = **09cf4f3c**

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i–Nk]	w[i]= temp XOR w[i-Nk]
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17					28aed2a6	88542cb1
6	88542cb1					abf71588	23a33939
7	23a33939					09cf4f3c	2a6c7605
8	2a6c7605	6c76052a	50386be5	02000000	52386be5	a0fafe17	f2c295f2
9	f2c295f2					88542cb1	7a96b943
10	7a96b943					23a33939	5935807a
11	5935807a					2a6c7605	7359f67f
12	7359f67f	59f67f73	cb42d28f	04000000	cf42d28f	f2c295f2	3d80477d
13	3d80477d					7a96b943	4716fe3e
14	4716fe3e					5935807a	1e237e44
15	1e237e44					7359f67f	6d7a883b
16	6d7a883b	7a883b6d	dac4e23c	08000000	d2c4e23c	3d80477d	ef44a541
17	ef44a541					4716fe3e	a8525b7f
18	a8525b7f					1e237e44	b671253b
19	b671253b					6d7a883b	db0bad00
20	db0bad00	0bad00db	2b9563b9	10000000	3b9563b9	ef44a541	d4d1c6f8
21	d4d1c6f8					a8525b7f	7c839d87
22	7c839d87					b671253b	caf2b8bc
23	caf2b8bc					db0bad00	11f915bc
24	11f915bc	f915bc11	99596582	20000000	b9596582	d4d1c6f8	6d88a37a
25	6d88a37a					7c839d87	110b3efd
26	110b3efd					caf2b8bc	dbf98641
27	dbf98641					11f915bc	ca0093fd
28	ca0093fd	0093fdca	63dc5474	40000000	23dc5474	6d88a37a	4e54f70e
29	4e54f70e					110b3efd	5f5fc9f3

30	**5f5fc9f3**					**dbf98641**	**84a64fb2**
31	**84a64fb2**					**ca0093fd**	**4ea6dc4f**
32	**4ea6dc4f**	**a6dc4f4e**	**2486842f**	**80000000**	**a486842f**	**4e54f70e**	**ead27321**
33	**ead27321**					**5f5fc9f3**	**b58dbad2**
34	**b58dbad2**					**84a64fb2**	**312bf560**
35	**312bf560**					**4ea6dc4f**	**7f8d292f**
36	**7f8d292f**	**8d292f7f**	**5da515d2**	**1b000000**	**46a515d2**	**ead27321**	**ac7766f3**
37	**ac7766f3**					**b58dbad2**	**19fadc21**
38	**19fadc21**					**312bf560**	**28d12941**
39	**28d12941**					**7f8d292f**	**575c006e**
40	**575c006e**	**5c006e57**	**4a639f5b**	**36000000**	**7c639f5b**	**ac7766f3**	**d014f9a8**
41	**d014f9a8**					**19fadc21**	**c9ee2589**
42	**c9ee2589**					**28d12941**	**e13f0cc8**
43	**e13f0cc8**					**575c006e**	**b6630ca6**

Appendix C - Encryption Examples

Plaintext = 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

Round Number	Start of Round	After SubBytes	After ShiftRows	After MixColumns	Round Key Value	
input	32 88 31 e0 43 5a 31 37 f6 30 98 07 a8 8d a2 34				2b 28 ab 09 7e ae f7 cf 15 d2 15 4f 16 a6 88 3c	⊕ =
1	19 a0 9a e9 3d f4 c6 f8 e3 e2 8d 48 be 2b 2a 08	d4 e0 b8 1e 27 bf b4 41 11 98 5d 52 ae f1 e5 30	d4 e0 b8 1e bf b4 41 27 5d 52 11 98 30 ae f1 e5	04 e0 48 28 66 cb f8 06 81 19 d3 26 e5 9a 7a 4c	a0 88 23 2a fa 54 a3 6c fe 2c 39 76 17 b1 39 05	⊕ =
2	a4 68 6b 02 9c 9f 5b 6a 7f 35 ea 50 f2 2b 43 49	49 45 7f 77 de db 39 02 d2 96 87 53 89 f1 1a 3b	49 45 7f 77 db 39 02 de 87 53 d2 96 3b 89 f1 1a	58 1b db 1b 4d 4b e7 6b ca 5a ca b0 f1 ac a8 e5	f2 7a 59 73 c2 96 35 59 95 b9 80 f6 f2 43 7a 7f	⊕ =
3	aa 61 82 68 8f dd d2 32 5f e3 4a 46 03 ef d2 9a	ac ef 13 45 73 c1 b5 23 cf 11 d6 5a 7b df b5 b8	ac ef 13 45 c1 b5 23 73 d6 5a cf 11 b8 7b df b5	75 20 53 bb ec 0b c0 25 09 63 cf d0 93 33 7c dc	3d 47 1e 6d 80 16 23 7a 47 fe 7e 88 7d 3e 44 3b	⊕ =
4	48 67 4d d6 6c 1d e3 5f 4e 9d b1 58 ee 0d 38 e7	52 85 e3 f6 50 a4 11 cf 2f 5e c8 6a 28 d7 07 94	52 85 e3 f6 a4 11 cf 50 c8 6a 2f 5e 94 28 d7 07	0f 60 6f 5e d6 31 c0 b3 da 38 10 13 a9 bf 6b 01	ef a8 b6 db 44 52 71 0b a5 5b 25 ad 41 7f 3b 00	⊕ =
5	e0 c8 d9 85 92 63 b1 b8 7f 63 35 be e8 c0 50 01	e1 e8 35 97 4f fb c8 6c d2 fb 96 ae 9b ba 53 7c	e1 e8 35 97 fb c8 6c 4f 96 ae d2 fb 7c 9b ba 53	25 bd b6 4c d1 11 3a 4c a9 d1 33 c0 ad 68 8e b0	d4 7c ca 11 d1 83 f2 f9 c6 9d b8 15 f8 87 bc bc	⊕ =

6

f1	c1	7c	5d
00	92	c8	b5
6f	4c	8b	d5
55	ef	32	0c

a1	78	10	4c
63	4f	e8	d5
a8	29	3d	03
fc	df	23	fe

a1	78	10	4c
4f	e8	d5	63
3d	03	a8	29
fe	fc	df	23

4b	2c	33	37
86	4a	9d	d2
8d	89	f4	18
6d	80	e8	d8

⊕

6d	11	db	ca
88	0b	f9	00
a3	3e	86	93
7a	fd	41	fd

=

7

26	3d	e8	fd
0e	41	64	d2
2e	b7	72	8b
17	7d	a9	25

f7	27	9b	54
ab	83	43	b5
31	a9	40	3d
f0	ff	d3	3f

f7	27	9b	54
83	43	b5	ab
40	3d	31	a9
3f	f0	ff	d3

14	46	27	34
15	16	46	2a
b5	15	56	d8
bf	ec	d7	43

⊕

4e	5f	84	4e
54	5f	a6	a6
f7	c9	4f	dc
0e	f3	b2	4f

=

8

5a	19	a3	7a
41	49	e0	8c
42	dc	19	04
b1	1f	65	0c

be	d4	0a	da
83	3b	e1	64
2c	86	d4	f2
c8	c0	4d	fe

be	d4	0a	da
3b	e1	64	83
d4	f2	2c	86
fe	c8	c0	4d

00	b1	54	fa
51	c8	76	1b
2f	89	6d	99
d1	ff	cd	ea

⊕

ea	b5	31	7f
d2	8d	2b	8d
73	ba	f5	29
21	d2	60	2f

=

9

ea	04	65	85
83	45	5d	96
5c	33	98	b0
f0	2d	ad	c5

87	f2	4d	97
ec	6e	4c	90
4a	c3	46	e7
8c	d8	95	a6

87	f2	4d	97
6e	4c	90	ec
46	e7	4a	c3
a6	8c	d8	95

47	40	a3	4c
37	d4	70	9f
94	e4	3a	42
ed	a5	a6	bc

⊕

ac	19	28	57
77	fa	d1	5c
66	dc	29	00
f3	21	41	6e

=

10

eb	59	8b	1b
40	2e	a1	c3
f2	38	13	42
1e	84	e7	d2

e9	cb	3d	af
09	31	32	2e
89	07	7d	2c
72	5f	94	b5

e9	cb	3d	af
31	32	2e	09
7d	2c	89	07
b5	72	5f	94

⊕

d0	c9	e1	b6
14	ee	3f	63
f9	25	0c	0c
a8	89	c8	a6

=

output

39	02	dc	19
25	dc	11	6a
84	09	85	0b
1d	fb	97	32