HARDWARE IMPLEMENTATION OF REAL TIME COMMUNICATION USING UART WITH AES 128 ENCRYPTION/DECRYPTION IN VERILOG

ADVANCED ENCRYPTION STANDARD (AES)

- BLOCK CIPHER: PROCESSES BLOCKS OF TEXTS OF FIXED SIZE OF 128 BITS (16 BYTES) AND OUTPUTS A NEW BLOCK OF THE SAME SIZE.
- WORKS ON THE METHODS OF SUBSTITUTION AND PERMUTATIONS.
- SYMMETRIC ENCRYPTION ALGORITHM, SAME KEY FOR BOTH ENCRYPTION AND DECRYPTION WHICH IS CONFIDENTIAL TO SENDER AND RECEIVER.

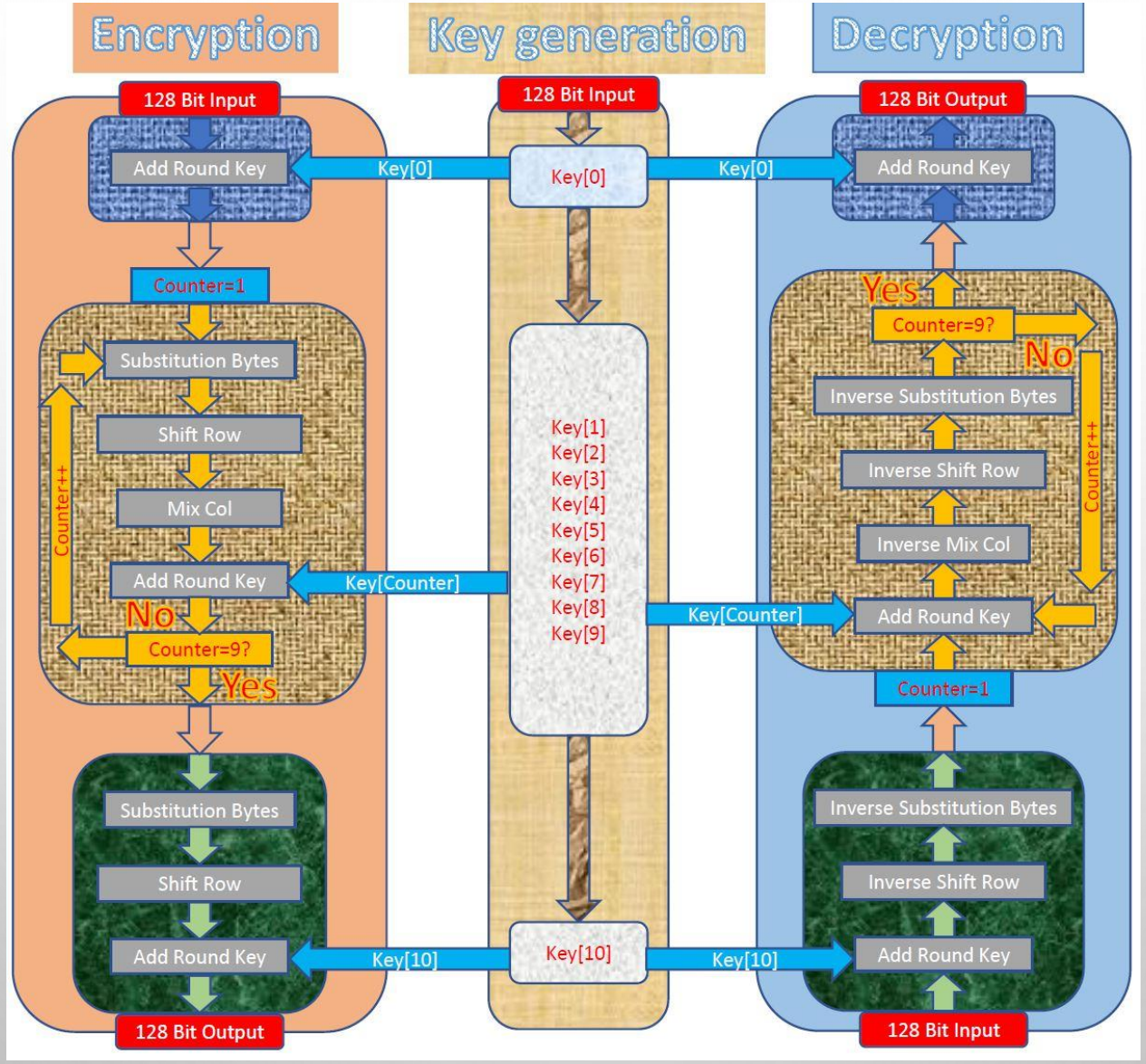


Figure: AES Encryption and Decryption

AES ADD ROUND KEY

THE INPUT DATA IS BITWISE XOR'ED WITH KEY IN THIS STAGE

Table 1: Add Round Key

Input Data				⊕	Key				=	Output			
D0	D4	D8	D12		K0	K4	K8	K12		A0	A4	A8	A12
D1	D5	D9	D13		K1	K5	K9	K13		A1	A5	A9	A13
D2	D6	D10	D14		K2	K6	K10	K14		A2	A6	A10	A14
D3	D7	D11	D15		K3	K7	K11	K15		A3	A7	A11	A15

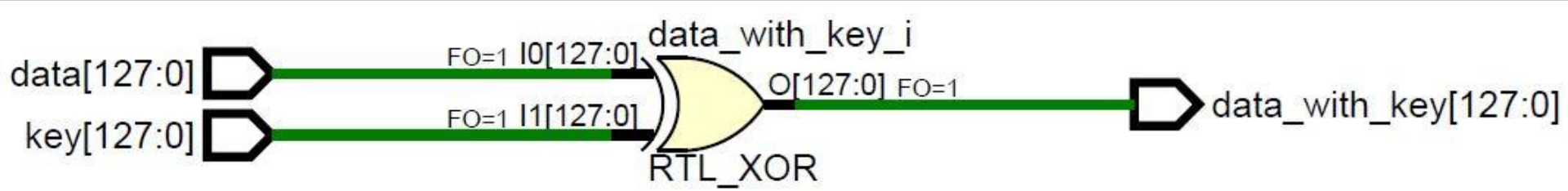
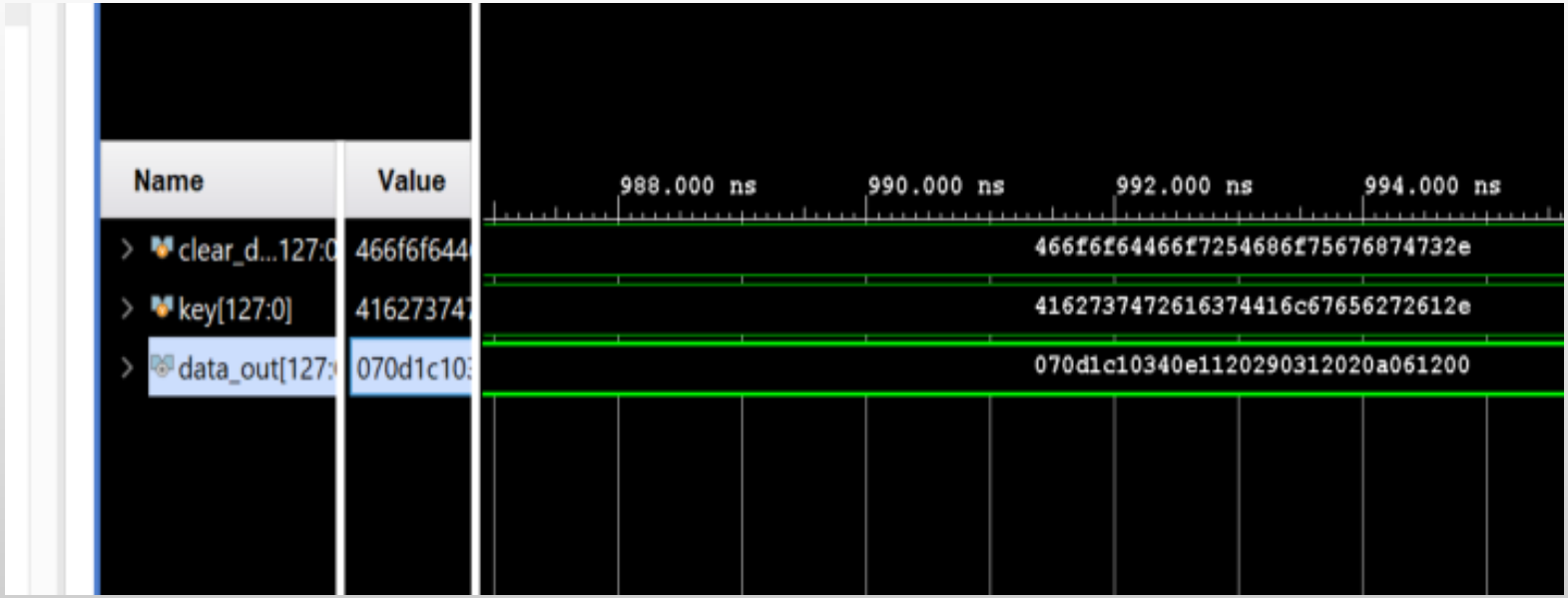


Table 1: AES Transformation Method

Each box represent 8 bit out of 128 bit													
in_00	in_04	in_08	in_12	→	S_00	S_04	S_08	S_12	→	out_00	out_04	out_08	out_12
in_01	in_05	in_09	in_13	→	S_01	S_05	S_09	S_13	→	out_01	out_05	out_09	out_13
in_02	in_06	in_10	in_14	→	S_02	S_06	S_10	S_14	→	out_02	out_06	out_10	out_14
in_03	in_07	in_11	in_15	→	S_03	S_07	S_11	S_15	→	out_03	out_07	out_11	out_15
Input				→	Process				→	Output			

Simulation: ADD Round Key



SUBSTITUTION BYTES ( ) TRANSFORMATION

•Each byte substituted using a well-defined substitution box called S-box.

•Contains mapping of each byte from 00 to FF.

•For example: {56} is substituted by looking at row 5 and column 6 in the 16 * 16 S-box, the substituted byte is the intersection of row and column.

S0	S4	S8	S12
S1	S5	S9	S13
S2	S6	S10	S14
S3	S7	S11	S15

=

A0	A4	A8	A12
A1	A5	A9	A13
A2	A6	A10	A14
A3	A7	A11	A15

DATA IS MAPPED TO 4*4 MATRIX

	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

Fig : **AES Substitution box**

INVERSE SUBSTITUTION BYTES ( ) TRANSFORMATION

- Inverse of the substitution Bytes Transformation.
- Inverse Substitution Box is the inverse mapping of the substitution box.
- For example: The byte {B1} is transformed to {56} after this operation.

- {56} {B1} (Encryption)
- {B1} {56} (Decryption)

S0	S4	S8	S12
S1	S5	S9	S13
S2	S6	S10	S14
S3	S7	S11	S15

=

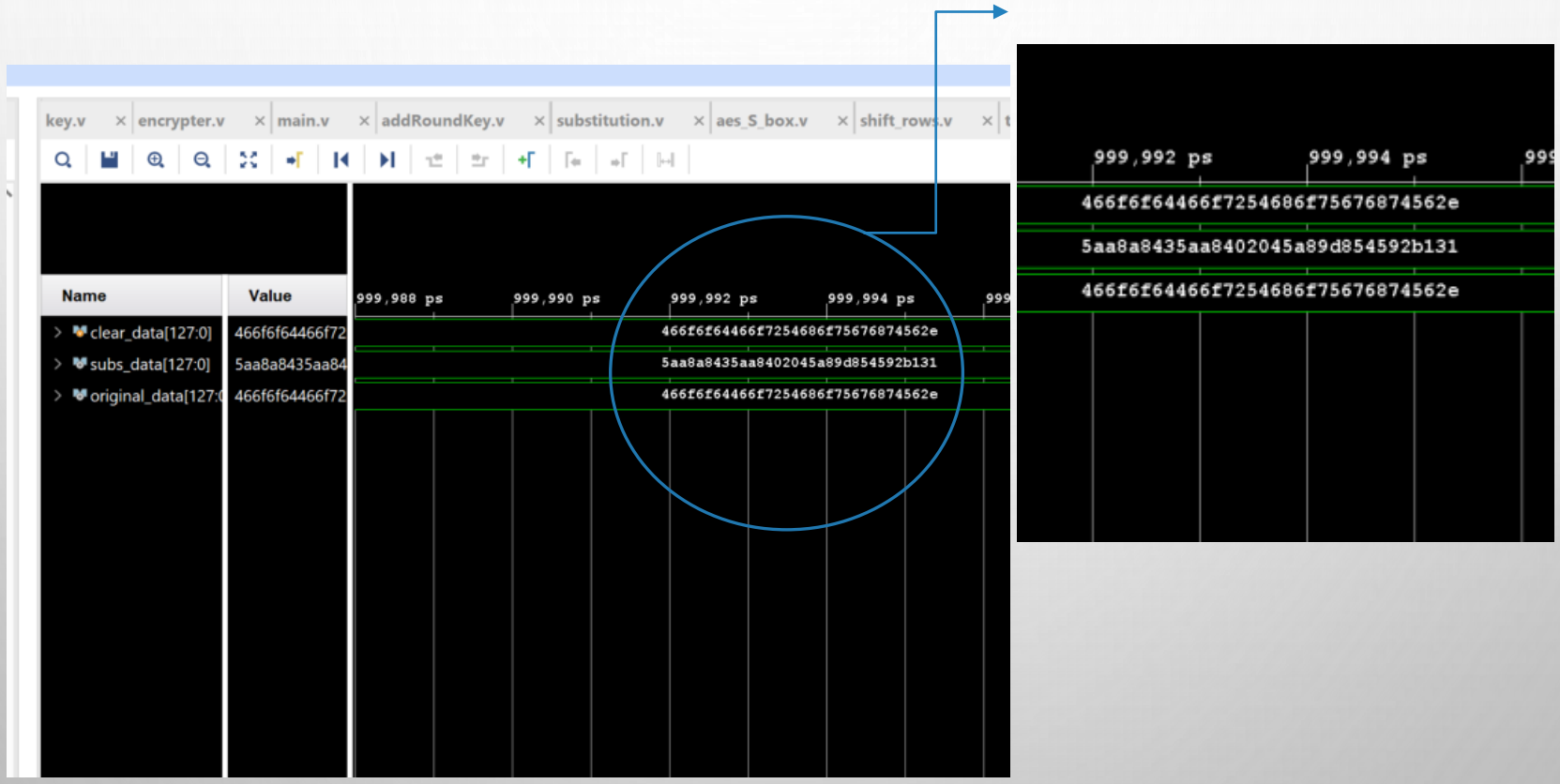
A0	A4	A8	A12
A1	A5	A9	A13
A2	A6	A10	A14
A3	A7	A11	A15

DATA IS MAPPED TO 4*4 MATRIX

	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	08	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
f	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Fig : **AES Inverse Substitution box**

VERIFICATION OF SUB BYTE AND INVERSE SUB BYTE TRANSFORMATION



SHIFT ROWS ( ) TRANSFORMATION

- The 16-byte data (or say 128-bit data) is arranged in a 4 x 4 grid.

- In each of the four rows,
  - 1st row remains unchanged.
  - 2nd row, each byte is shifted one position to the left.
  - 3rd row, each byte is shifted two positions to the left.
  - 4th row, each byte is shifted three position to the left.

A1	**A5**	**A9**	**A13**
A2	A6	A10	A14
A3	A7	A11	A15
A4	A8	A12	A16

Fig : 4 x 4 array of input data

A1	**A5**	**A9**	**A13**
A6	A10	A14	A2
A11	A15	A3	A7
A16	A4	A8	A12

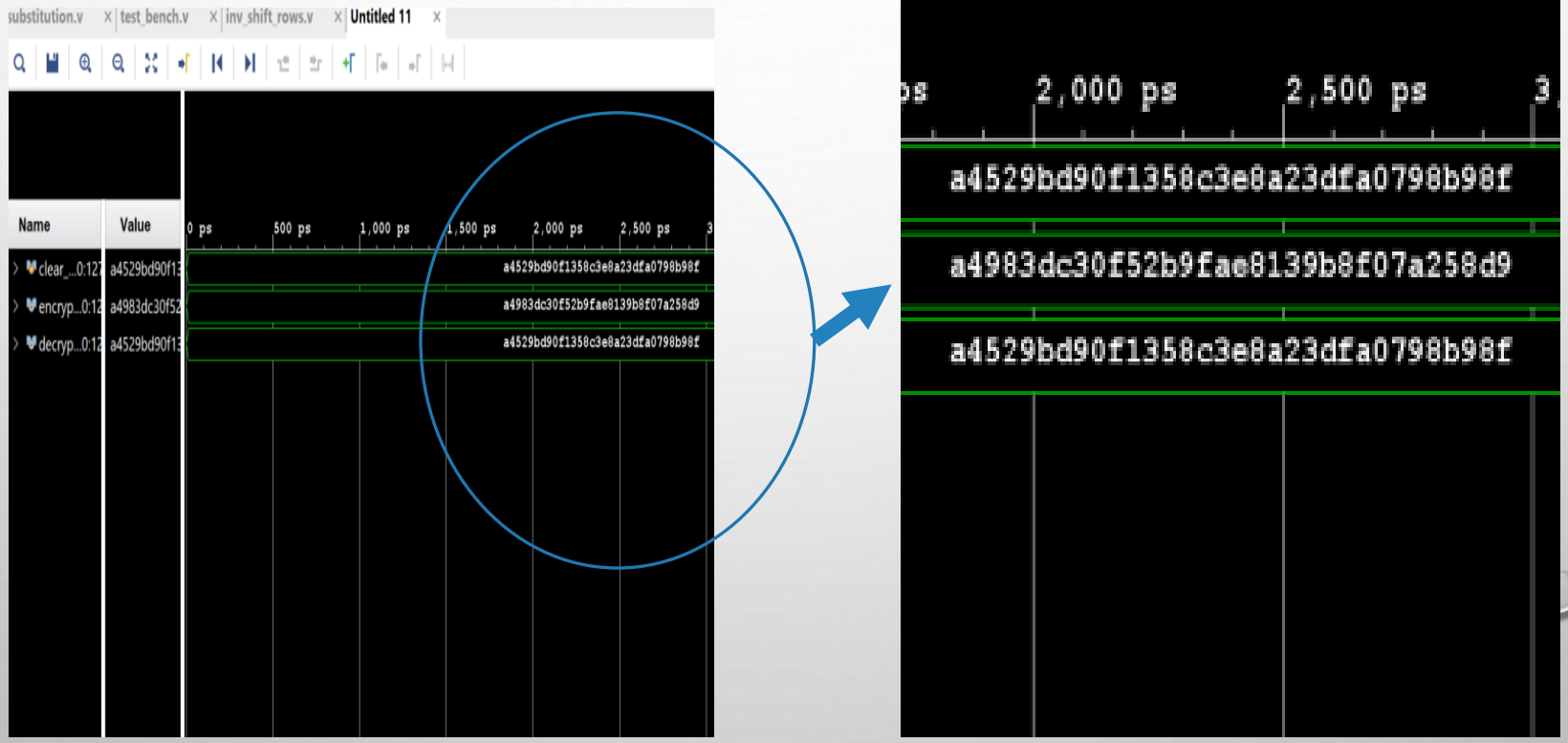
Fig: 4 x 4 array of shifted data

INVERSE SHIFT ROWS ( ) TRANSFORMATION

- Similar to the Shift Rows ( ) transformation in functionality, except the rows are shifted in the reverse order.

- In each of the four rows,
  - 1st row remains unchanged.
  - 2nd row, each byte is shifted one position to the right.
  - 3rd row, each byte is shifted two positions to the right.
  - 4th row, each byte is shifted three position to the right.

Verification of shift row() and inverse shift row() transformation



Mix_Column

Mix Col Matrix			
02	03	01	01
01	02	03	01
01	01	02	03
01	01	02	03

Input Data Matrix			
D0	D4	D8	D12
D1	D5	D9	D13
D2	D6	D10	D14
D3	D7	D11	D15

- Performs mixing of column wise data in finite field (GF^8)

Operation

Table 9: Column wise multiplication in Mix Col Operation

Mix Col Matrix					Input Col		Output Col
02	03	01	01		D_n		O_n
01	02	03	01	$\times$	D_{n+1}	$=$	O_{n+1}
01	01	02	03		D_{n+2}		O_$n+2$
01	01	02	03		D_{n+3}		O_{n+3}

$n \in \{0, 4, 8, 12\}$

Mix Col Matrix					Input Data Matrix					Output Data Matrix			
02	03	01	01		D0	D4	D8	D12		O0	O4	O8	O12
01	02	03	01	$\times$	D1	D5	D9	D13	$=$	O1	O5	O9	O13
01	01	02	03		D2	D6	D10	D14		O2	O6	O10	O14
01	01	02	03		D3	D7	D11	D15		O3	O7	O11	O15

•Irreducible Polynomial

- Irreducible Polynomial
  The irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ is utilized if our multiplication operation exceeds 8 bits. Since, our multiplication only involves multiplication by 01,01 and 03, the irreducible operation only needs to be subtracted once. This will provides us the remainder when our 9 bit output is divided by the irreducible polynomial. But, since we onlyt need 8 bit as our output, we can ignore the ninth bit from the beginning and just utilize $x^4 + x^3 + x + 1$ as our irreducible polnomial.

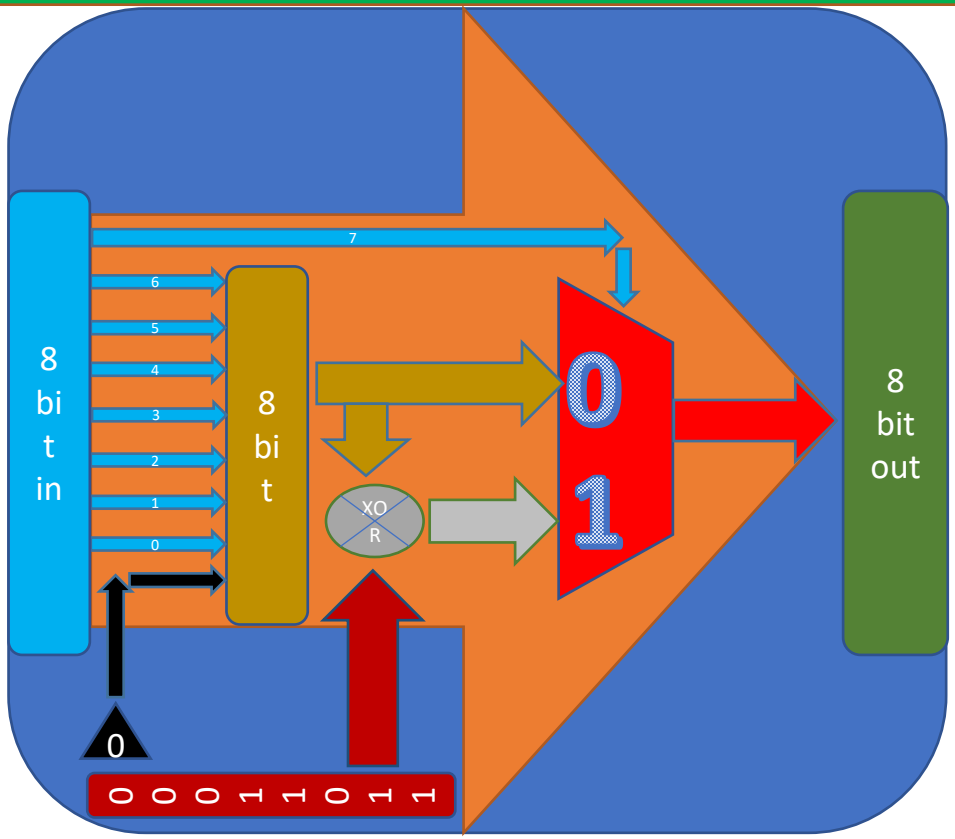
Binary Position	8	7	6	5	4	3	2	1	0
8	1	0	0	0	0	0	0	0	0
(8+4+3+1+0) * 0	1	0	0	0	1	1	0	1	1
XOR Remainder	0	0	0	0	1	1	0	1	1
Binary	0	0	0	0	1	1	0	1	1
Hex	1 B								

•Multiplication by 2

Table 11: Algorithm for Multiplication by 2

Input	a	b	c	d	e	f	g	h
	is a=1 or 0							
temp	b	c	d	e	f	g	h	0
	if a is 1							
Output	temp ⊕ irreducible polynomial							
	if a is 0							
Output	temp							

```
function [7:0] multiply_02(input [7:0]a);
    begin
    if (a[7]==0) begin
        multiply_02=a<<1;
    end else begin
        multiply_02=(a<<1)^8'b00011011;
    end
    end
endfunction
```



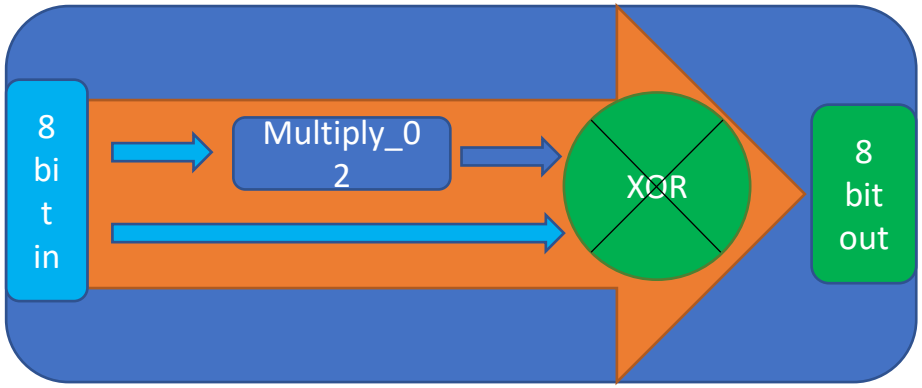
•Multiplication by 3

- Multiplication By 03
  In Galios Field(2^8) , multiplication is distributive over addition. Hence multiplication by 03 can be converted to multiplication by one and two.

$$input \times 03 = input \times (11) = input \times (10 \oplus 01) = input \times 10 \oplus input \times 01 = input \times 2 \oplus input$$

Thus, utilizing multiplication by 02, we can achieve multiplication bu 03.

```
function [7:0] multiply_03(input [7:0]a);
    reg [7:0]temp;
    begin
    temp=multiply_02(a);
    multiply_03=temp^a;
    end
endfunction
```



Inverse_Mix_Column

Inv Mix Col Matrix			
0E	0B	0D	09
09	0E	0B	0D
0D	09	0B	0D
0B	0D	09	0E

Input Data Matrix			
D0	D4	D8	D12
D1	D5	D9	D13
D2	D6	D10	D14
D3	D7	D11	D15

- Performs mixing of column wise data in finite field (GF^8)

•Multiplication by 0B

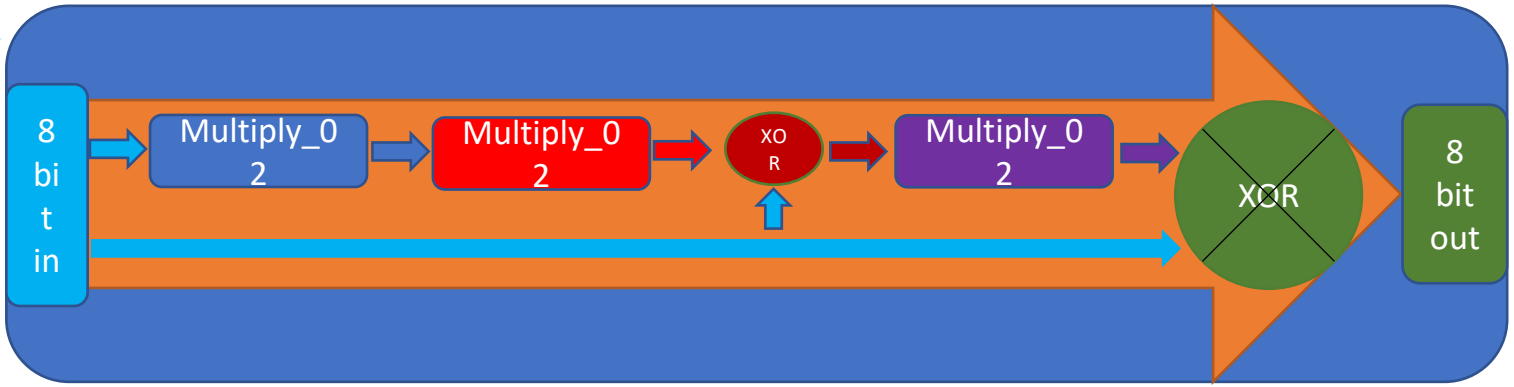
$$input \times 0B = input \times (00001011) = input \times (1000 \oplus 0010 \oplus 0001) = (input \times 1000) \oplus (input \times 0010) \oplus (input \times 0001)$$

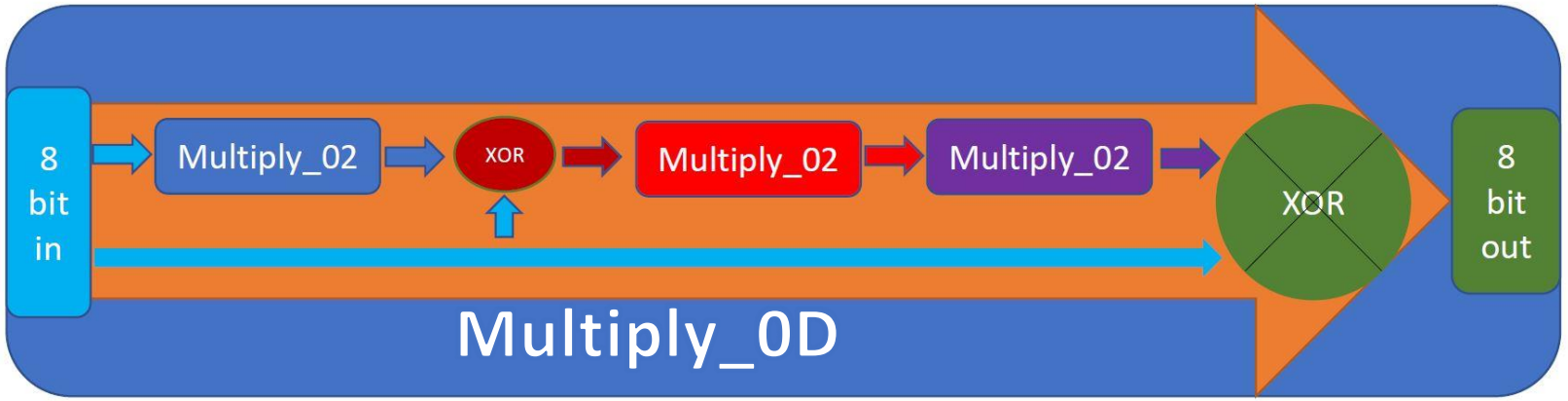
$$input \times 0B = (input \times 2 \times 2 \times 2) \oplus (input \times 2) \oplus (input)$$

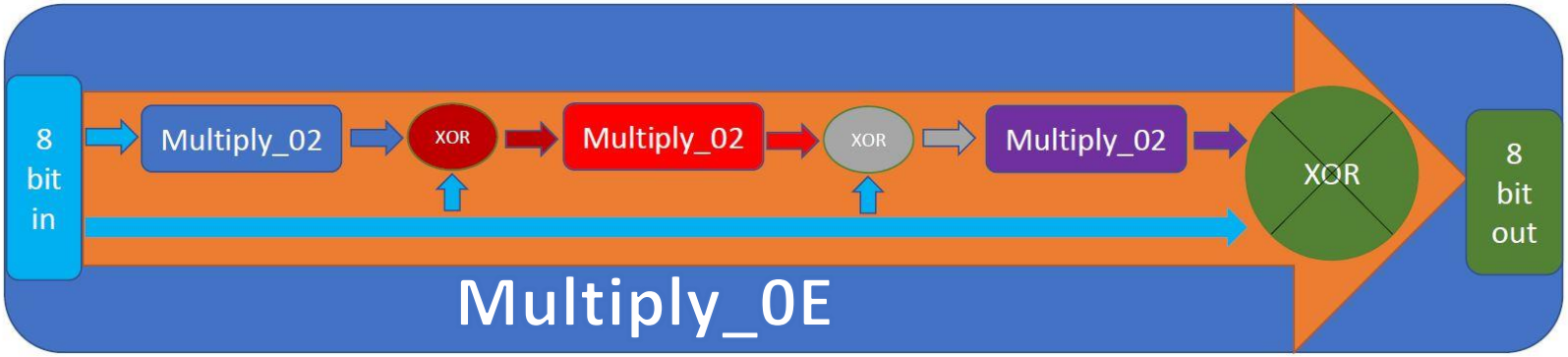
Smart Implementation by reducing repetition:

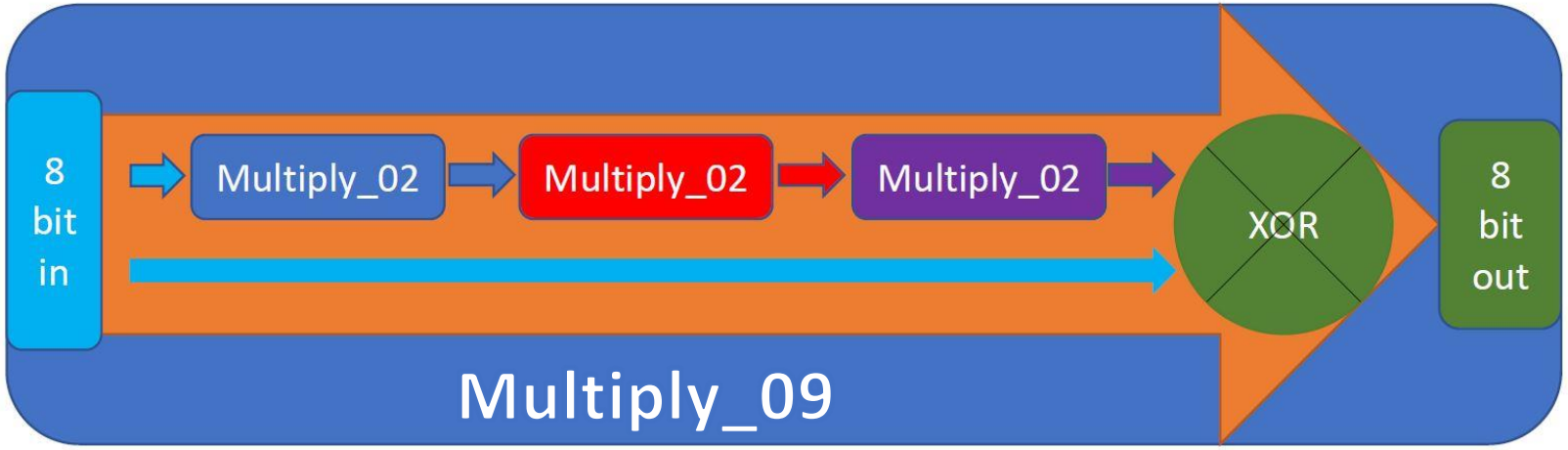
$$input \times 0B = (((input \times 2 \times 2) \oplus input) \times 2) \oplus input$$

```
function [7:0] multiply_0B(input [7:0]a);
    begin
    multiply_0B=(multiply_02(multiply_02(multiply_02(a))^a)^a);
    end
endfunction
```



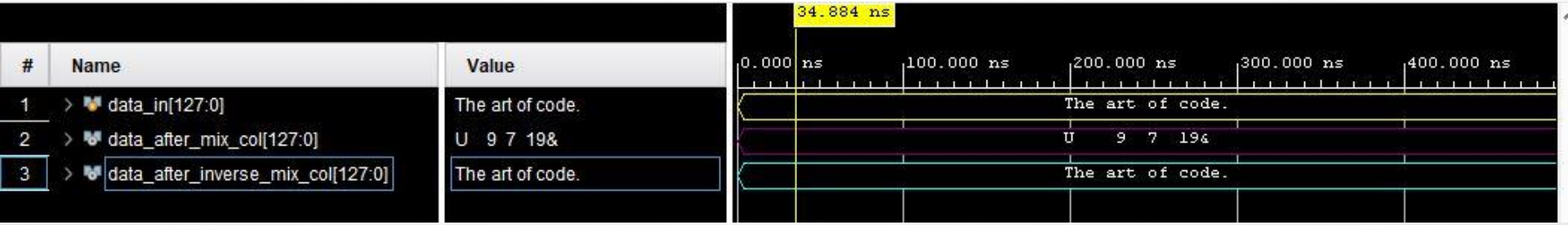

8 bit in
Multiply_02
XOR
Multiply_02
Multiply_02
XOR
8 bit out
Multiply_0D


8 bit in
Multiply_02
XOR
Multiply_02
XOR
Multiply_02
XOR
8 bit out
Multiply_0E

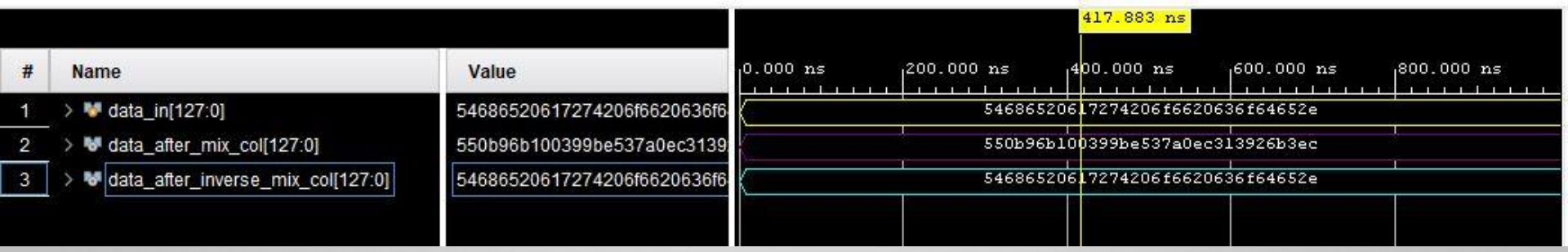

8 bit in
Multiply_02
Multiply_02
Multiply_02
XOR
8 bit out
Multiply_09

Simulation: Mix Column and Inverse Mix Column

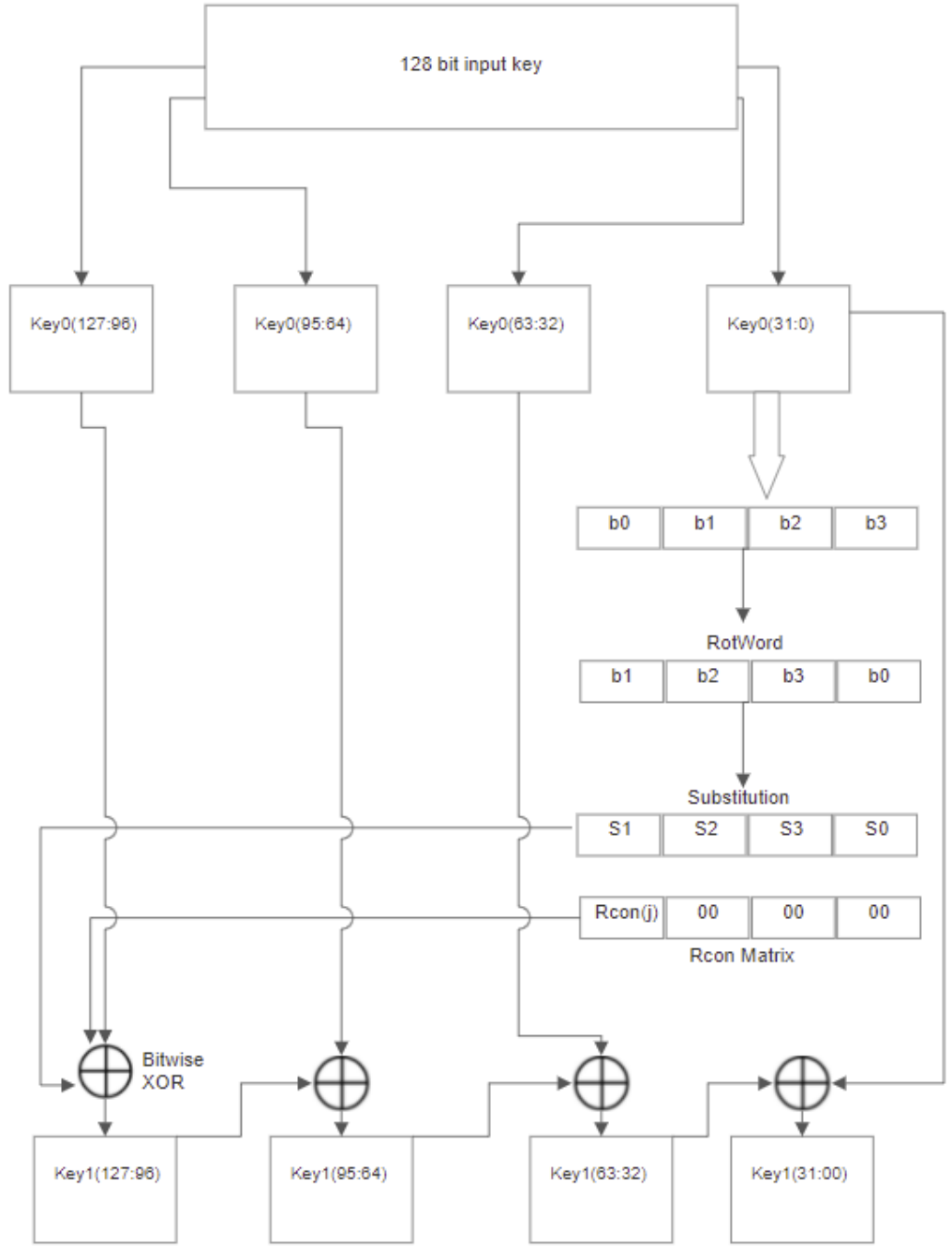
ASCII



Hexadecimal



KEY EXPANSION



Components

KEY EXPANSION

RCON Matrix:

$x^8 + x^4 + x^3 + x + 1$

- Three rightmost bytes are always 0.
- Varies for each iteration of key expansion round.
- Multiplication defined over the field GF(2^8).

$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \text{ and } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11B_{16} & \text{if } i > 1 \text{ and } rc_{i-1} \geq 80_{16} \end{cases}$$

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

Components

KEY EXPANSION

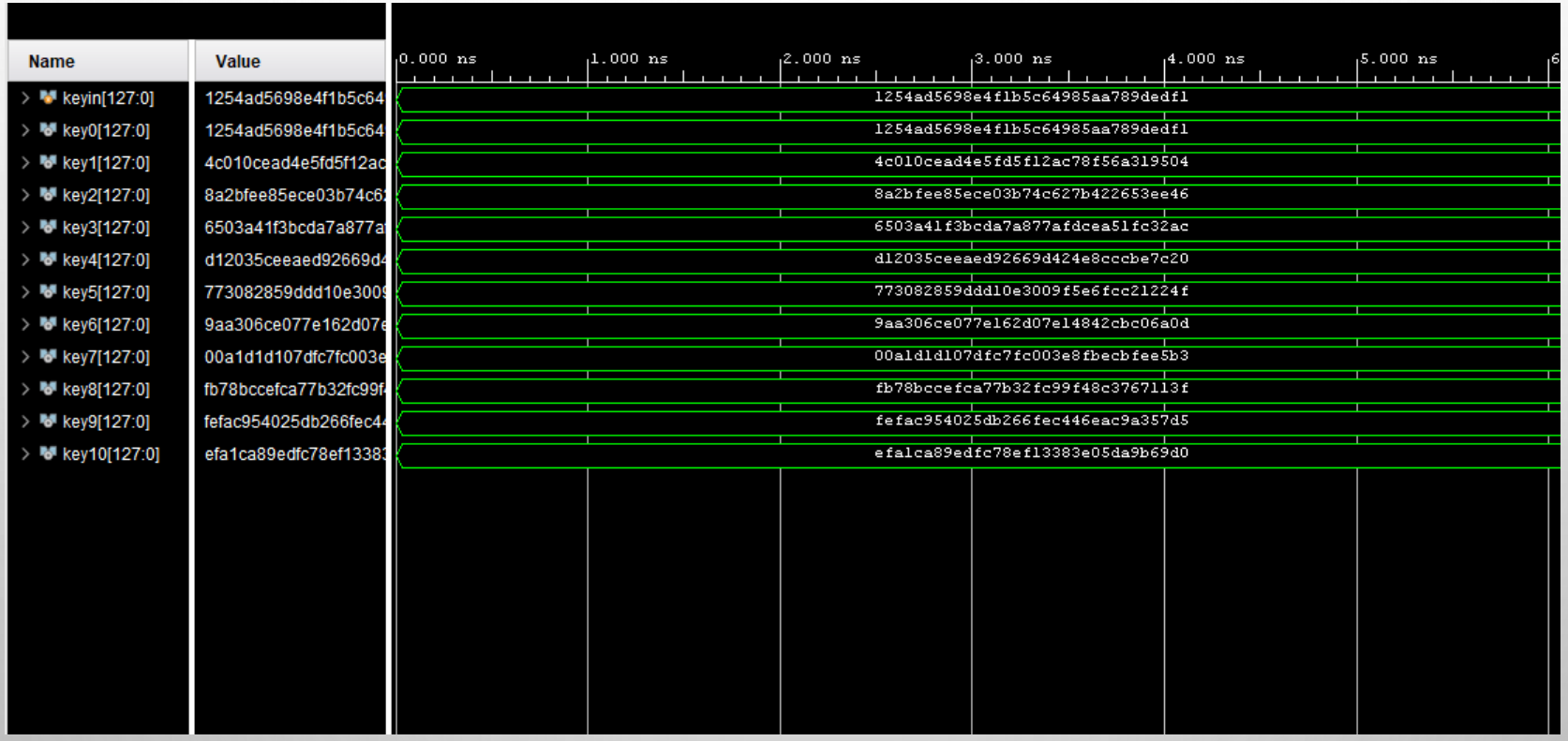
Substitution Byte:

- Performs byte substitution on input word with word from AES Substitution Box.

RotWord:

- RotWord performs a one-byte circular left shift on a word.
- This means that an input word [B0, B1, B2, B3] is transformed into [B1, B2, B3, B0].

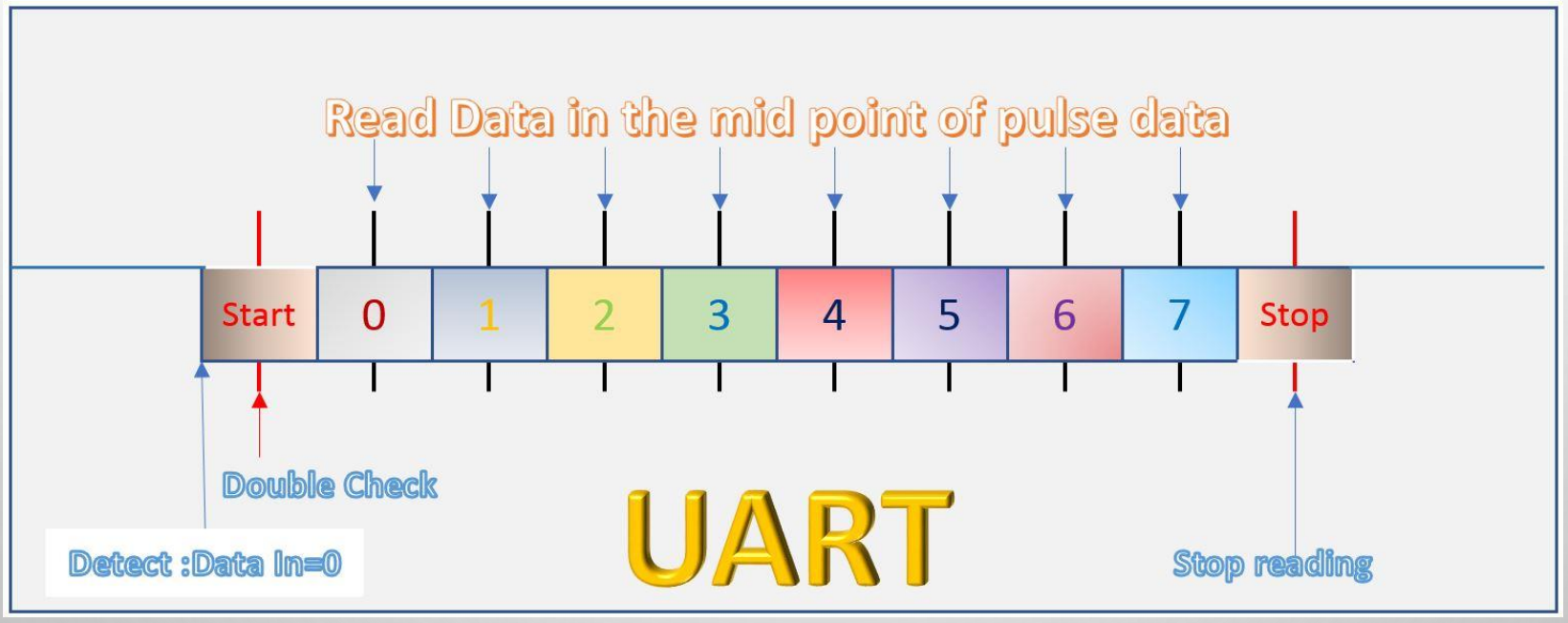
SIMULATION


Name
Value
0.000 ns
1.000 ns
2.000 ns
3.000 ns
4.000 ns
5.000 ns
keyin[127:0]
1254ad5698e4f1b5c64
1254ad5698e4f1b5c64985aa789dedf1
key0[127:0]
1254ad5698e4f1b5c64
1254ad5698e4f1b5c64985aa789dedf1
key1[127:0]
4c010cead4e5fd5f12ac
4c010cead4e5fd5f12ac78f56a319504
key2[127:0]
8a2bfee85ece03b74c6
8a2bfee85ece03b74c627b422653ee46
key3[127:0]
6503a41f3bcda7a877a
6503a41f3bcda7a877afdcea51fc32ac
key4[127:0]
d12035ceeaed92669d4
d12035ceeaed92669d424e8cccbe7c20
key5[127:0]
773082859ddd10e3009
773082859ddd10e3009f5e6fcc21224f
key6[127:0]
9aa306ce077e162d07e
9aa306ce077e162d07e14842cbc06a0d
key7[127:0]
00a1d1d107dfc7fc003e
00a1d1d107dfc7fc003e8fbecbfee5b3
key8[127:0]
fb78bccefca77b32fc99f
fb78bccefca77b32fc99f48c3767113f
key9[127:0]
fefac954025db266fec4
fefac954025db266fec446eac9a357d5
key10[127:0]
efa1ca89edfc78ef1338
efa1ca89edfc78ef13383e05da9b69d0

UART

- UART STANDS FOR UNIVERSAL ASYNCHRONOUS RECEIVER/TRANSMITTER
- DATA TRANSMITTED IN SERIAL STREAM REDUCING NUMBER OF WIRES USED TO TRANSFER DATA
- TRANSMITTER/RECEIVER AGREE ON CERTAIN PARAMETERS:
  - BAUD RATE (9600 BITS/SEC)
  - NUMBER OF DATA BITS (8 BITS)
  - PARITY BIT (0)
  - STOP BITS (1)
  - FLOW CONTROL (NONE)

UART PROTOCOL



FPGA clock speed	100MHZ
Baud_rate	9600 bits per seconds
clock_per_bits	10417 clock cycles per bit

UART Transmitter State Machine

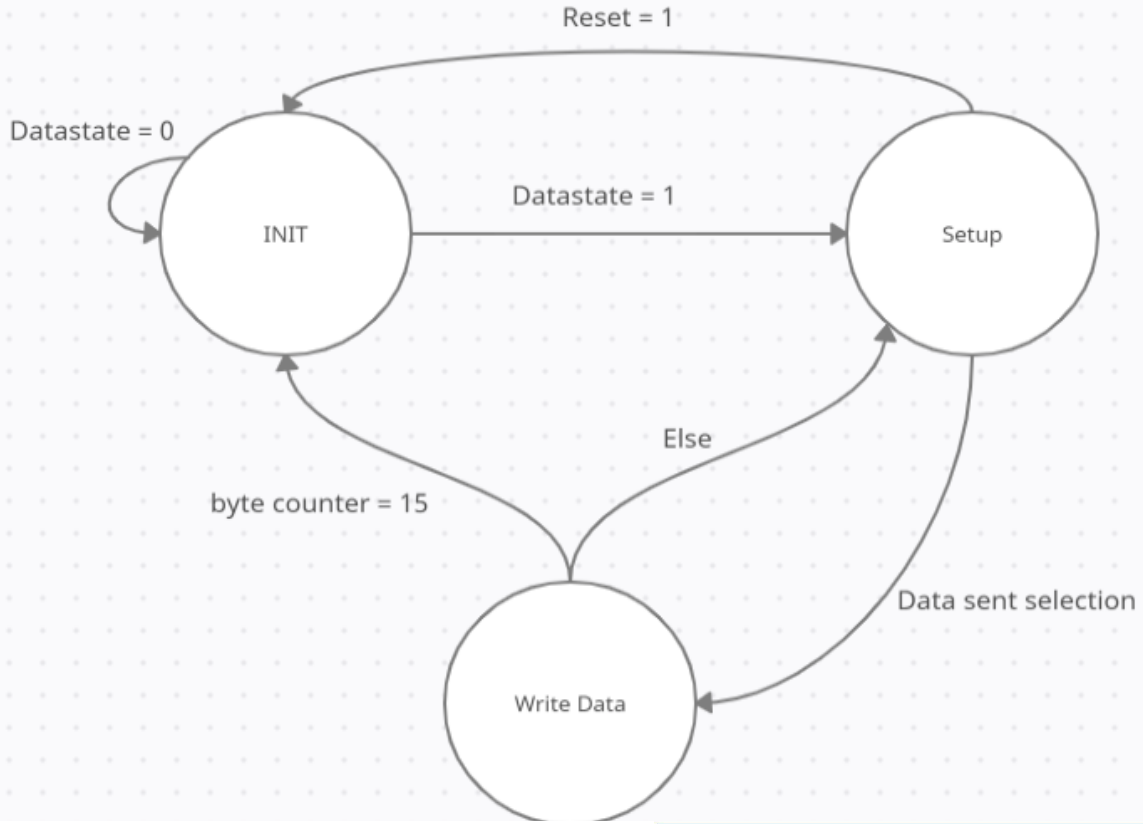
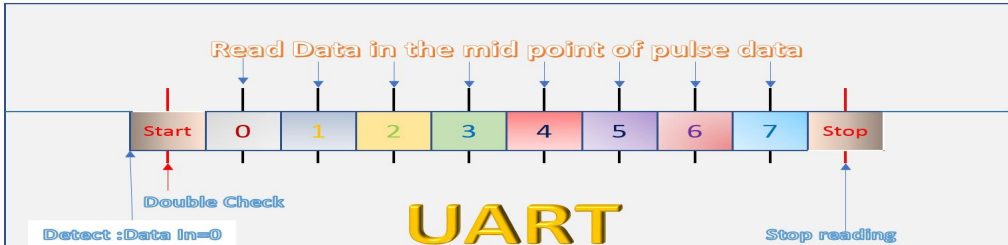


Table 18: Input/Output wire for Uart Transmitter

I/O	Wire	Size(bit)	Utilization
Input	clock	1	Clock for operations
	data	128	data that needs to be sent
	data_state	1	Data_ state is a flag that initiates the transmitter module.
	reset	1	Resets the transmitter to state Init.
Output	data_out_tx	1	Output pin.



UART Receiver

State Machine

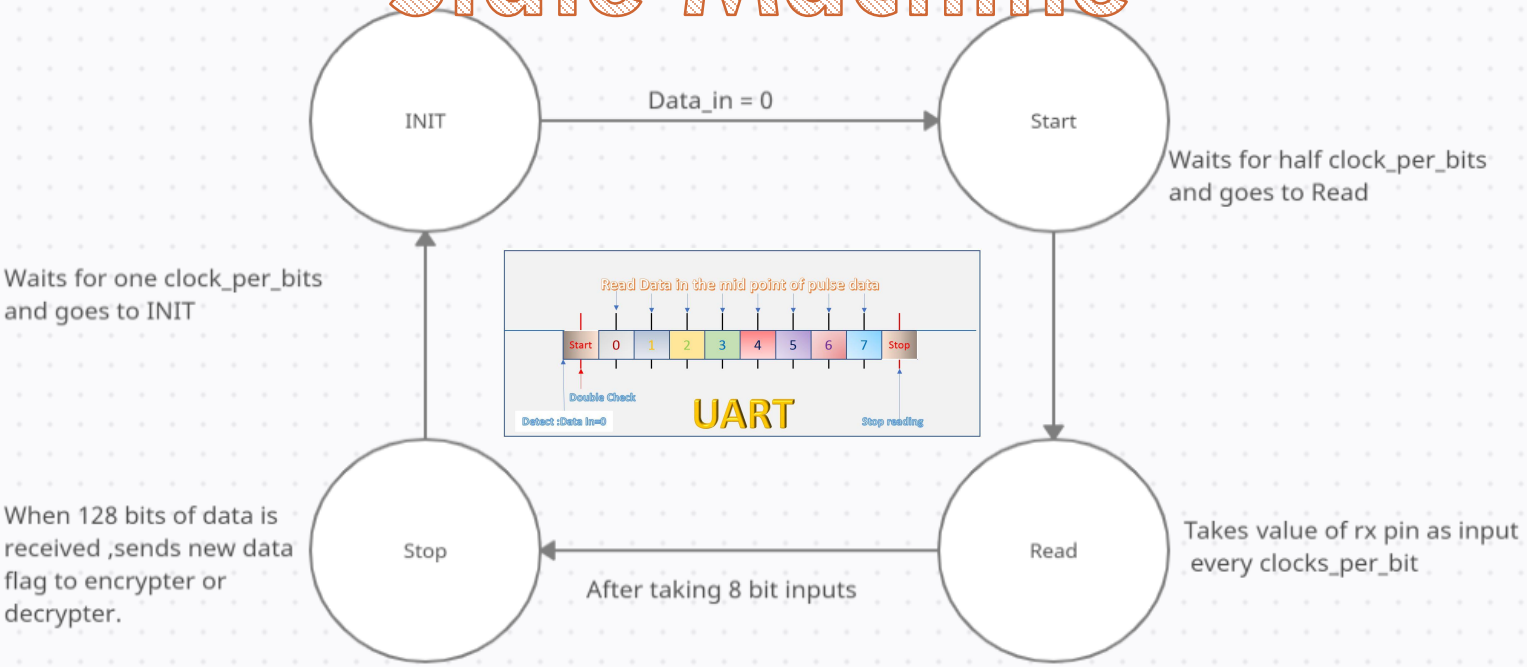


Table 19: Input/Output wire for Uart Reciever

I/O	Wire	Size(bit)	Utilization
Input	clock	1	Clock for operations
	data_in	1	It represents the pin that acts as reciever for FPGA.
Output	data_out	128	Output pin.
	data_state	1	Flag that represents new set of 128 bit has been received.

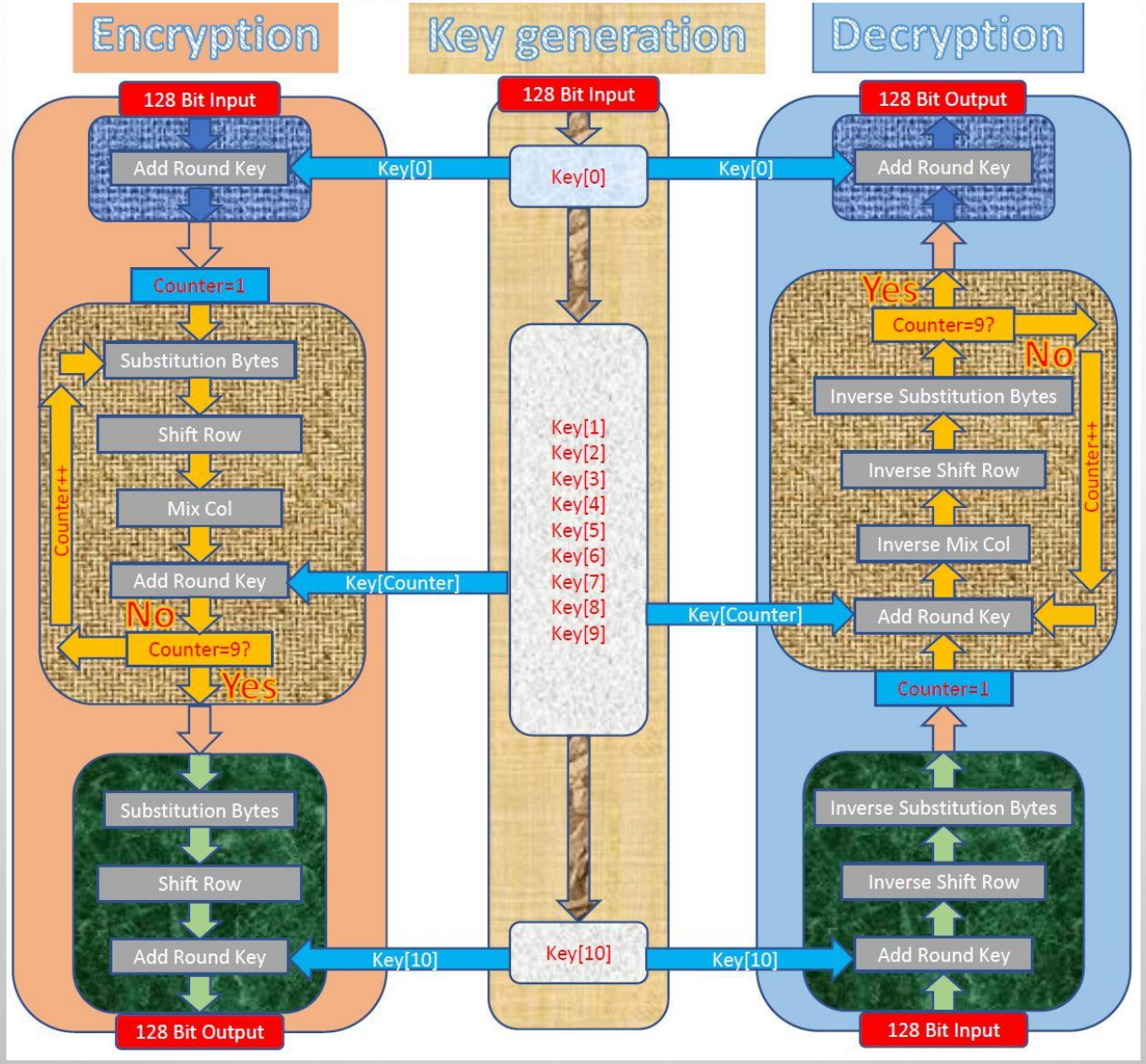
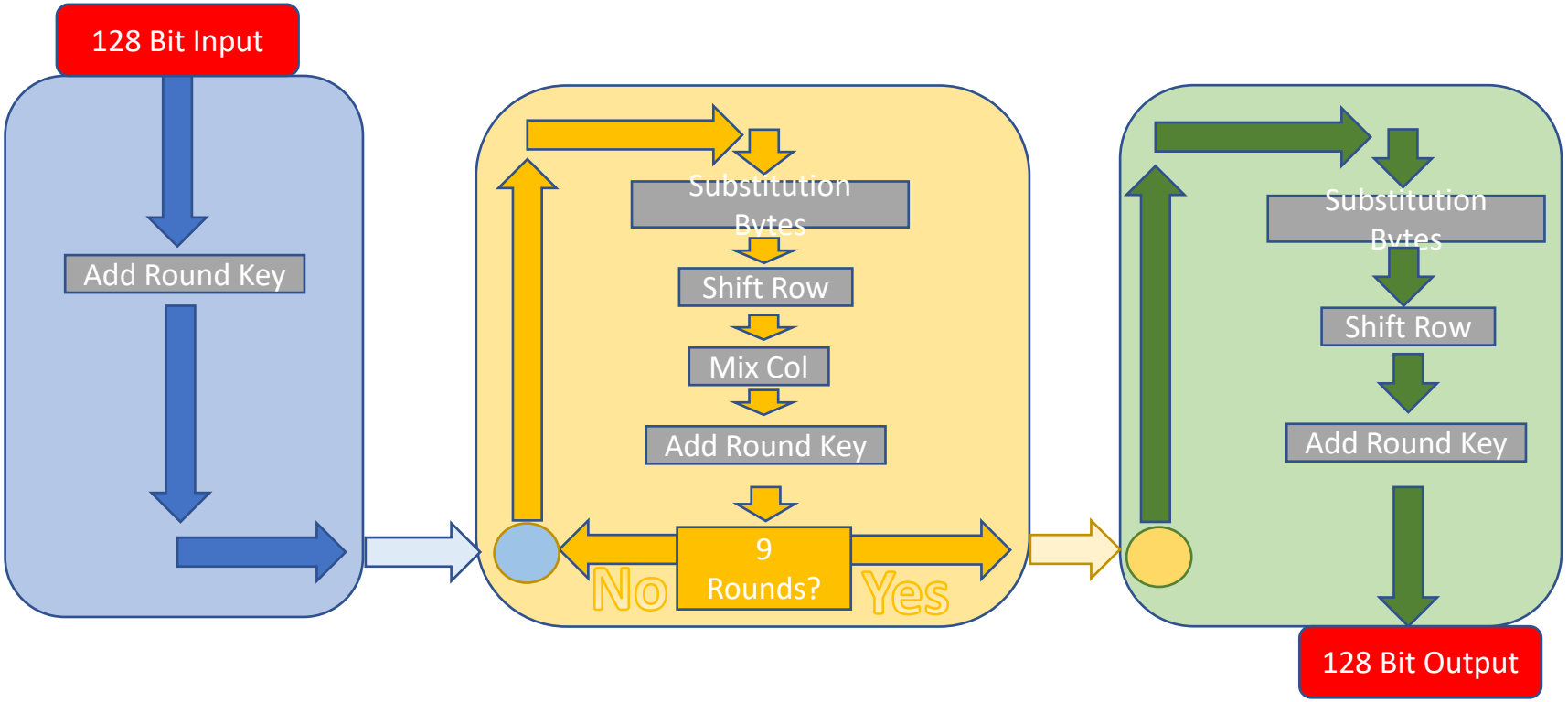
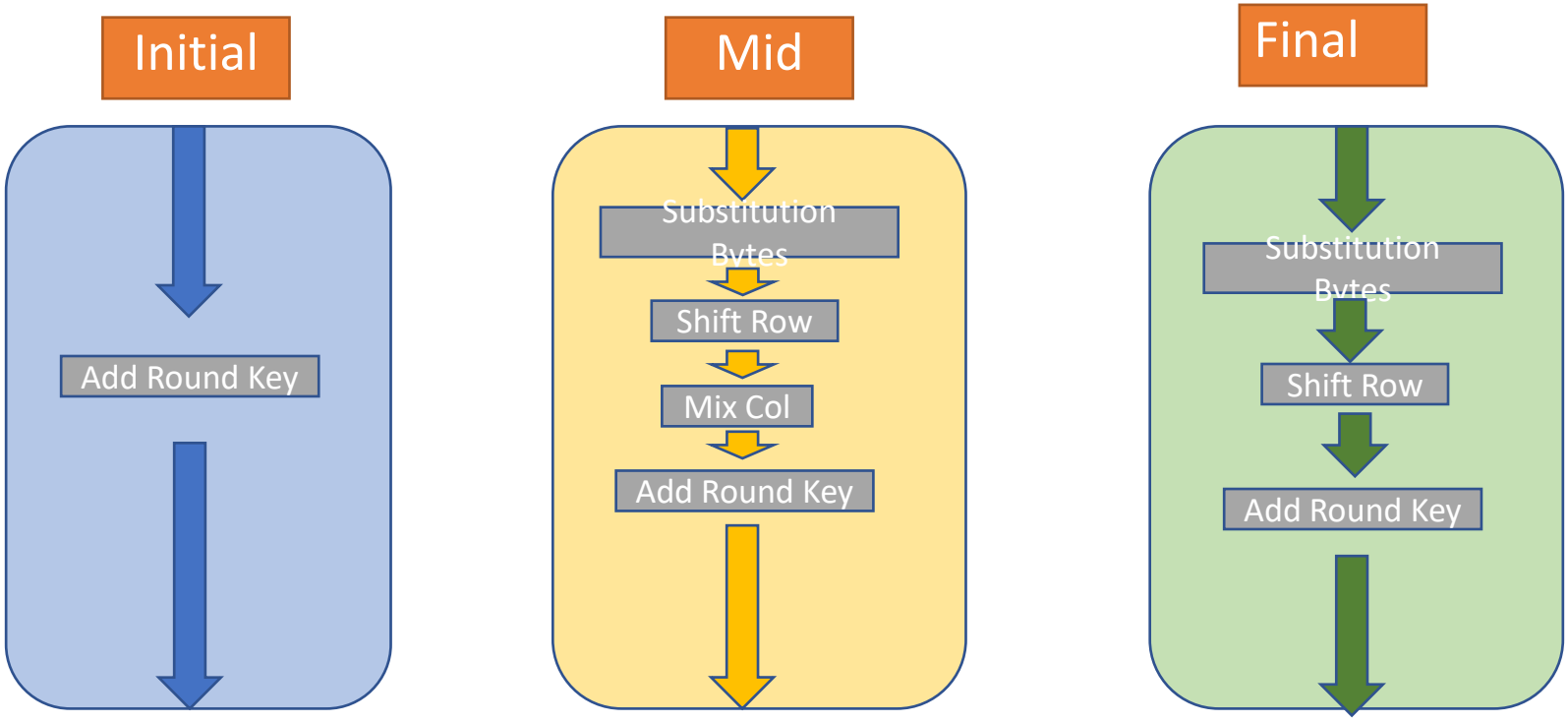


Figure: AES Encryption and Decryption

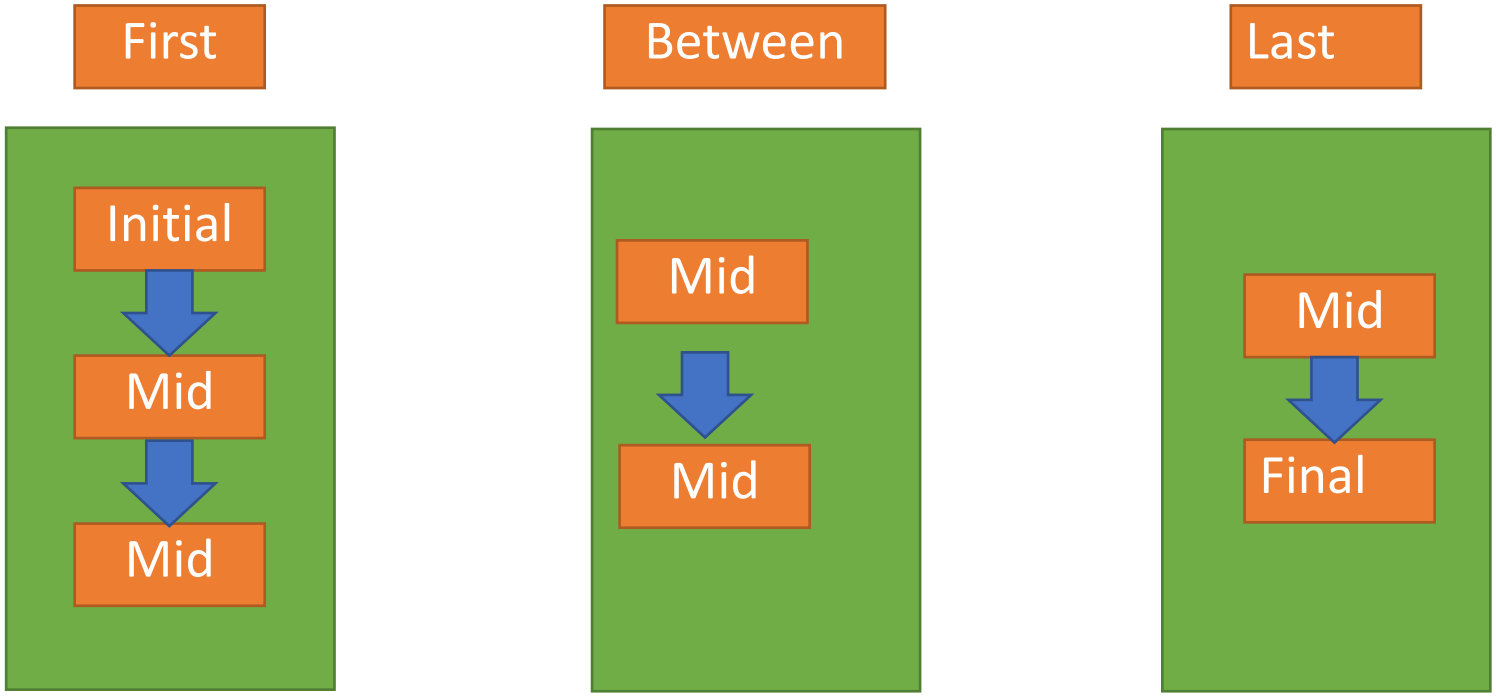
•Encryption


128 Bit Input
Add Round Key
Substitution Bytes
Shift Row
Mix Col
Add Round Key
9 Rounds?
No
Yes
Substitution Bytes
Shift Row
Add Round Key
128 Bit Output

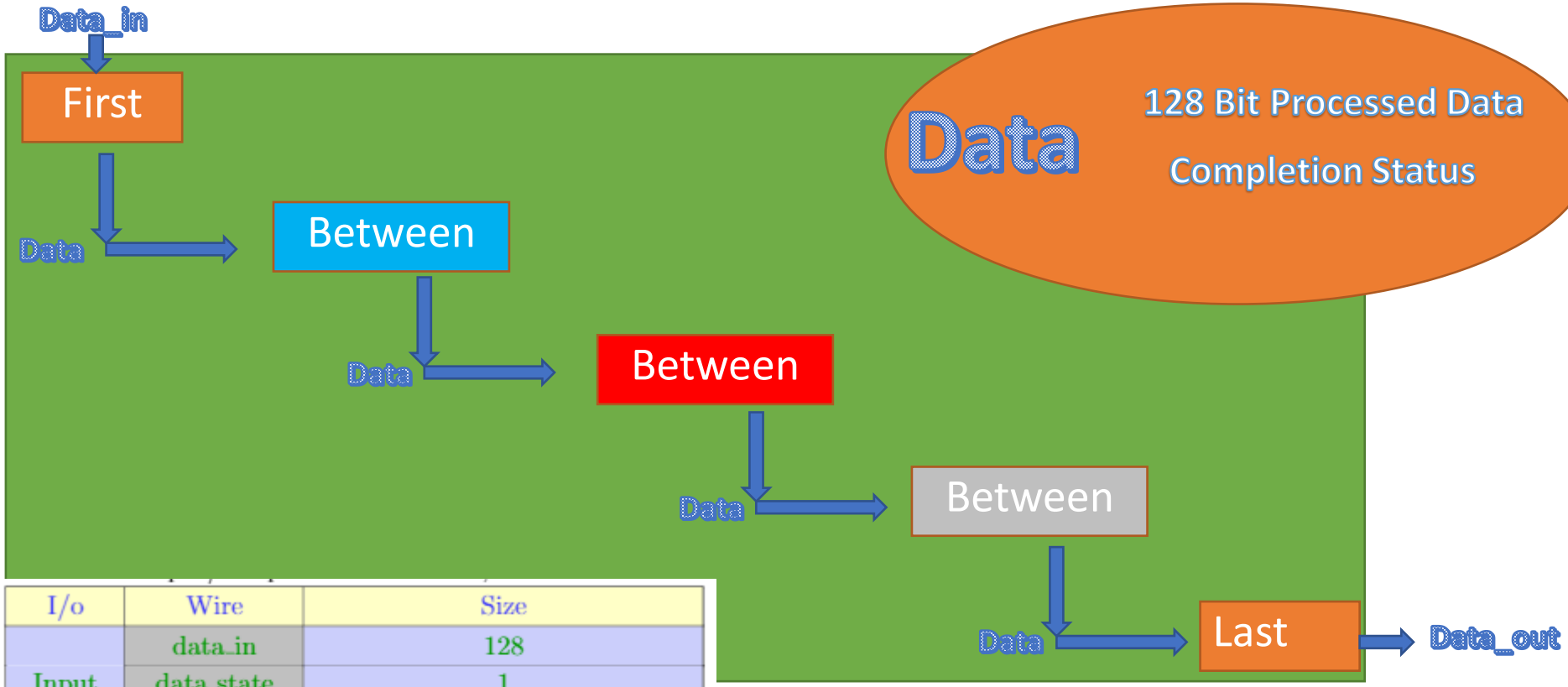
•Circuits


Initial
Add Round Key
Mid
Substitution Bytes
Shift Row
Mix Col
Add Round Key
Final
Substitution Bytes
Shift Row
Add Round Key

- FSM Circuits

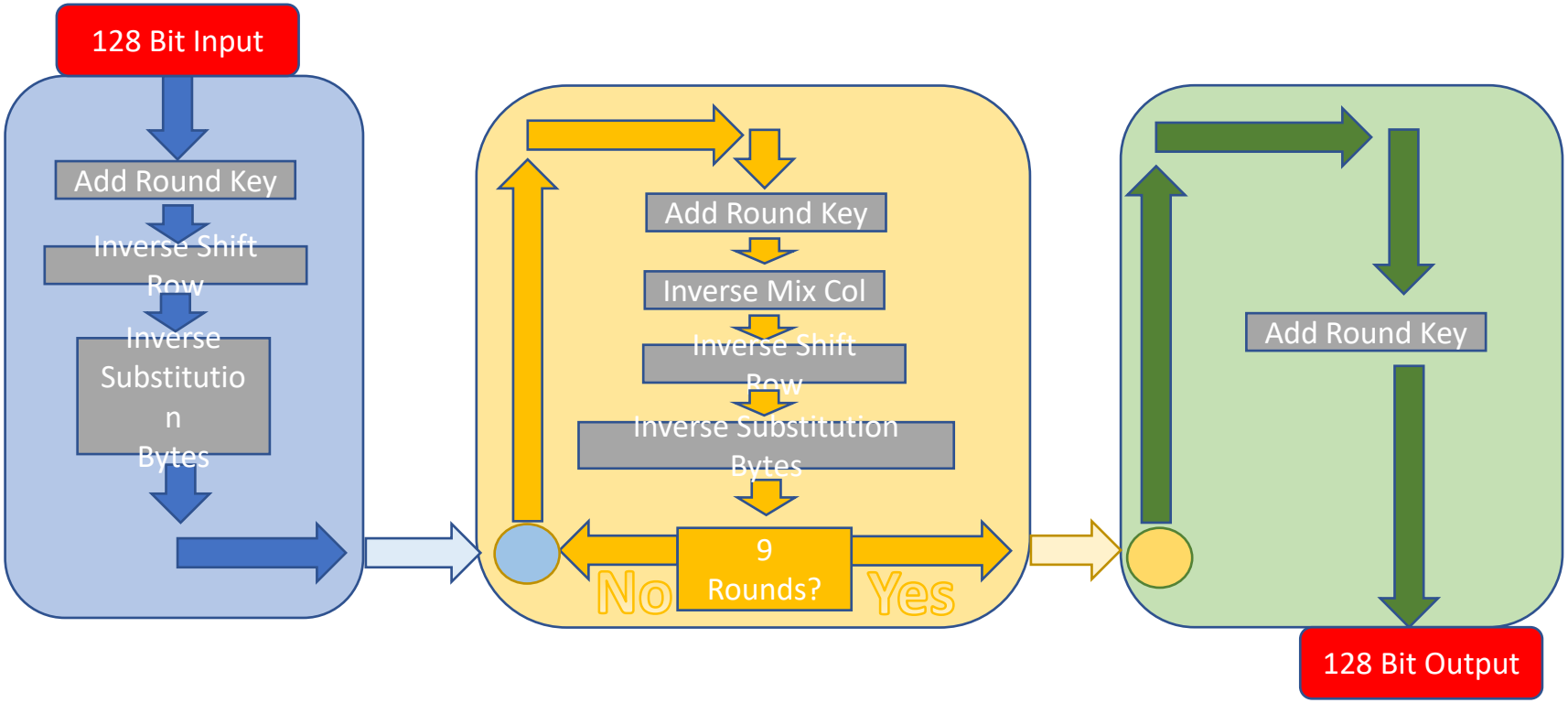


•Encrypter

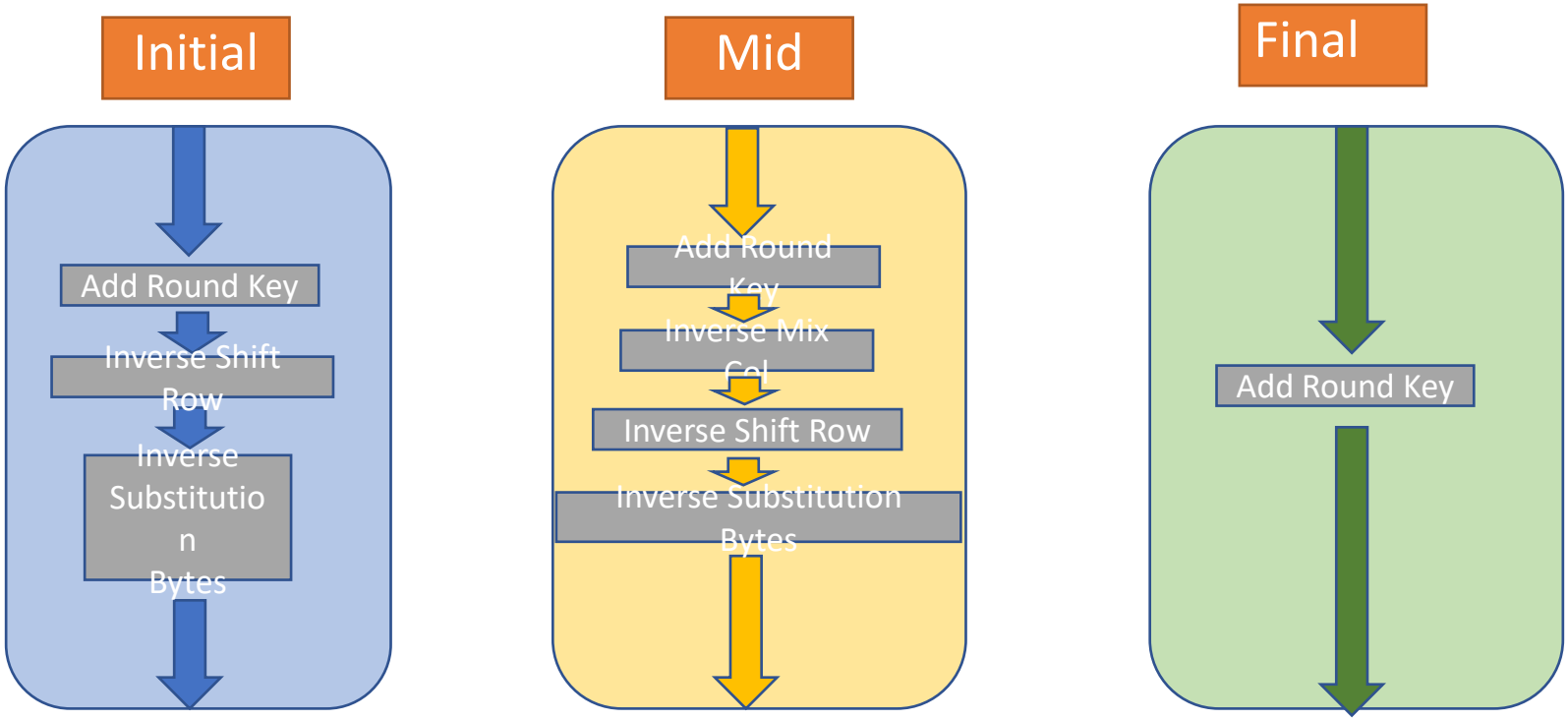


I/o	Wire	Size
	data_in	128
Input	data_state	1
	data_out	128
Output	encode_state	1
These are wires which are connected to internal registers		

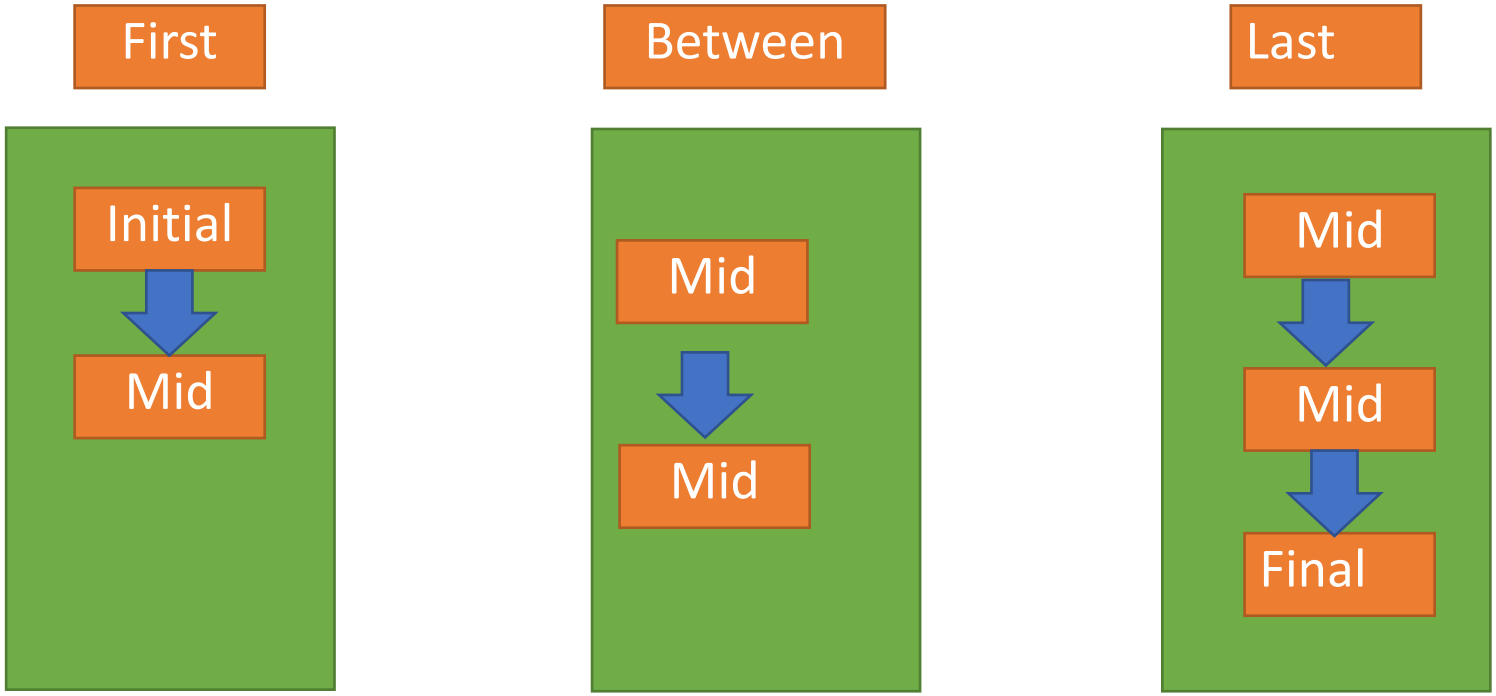
•Decryption


128 Bit Input
Add Round Key
Inverse Shift Row
Inverse Substitution Bytes
Add Round Key
Inverse Mix Col
Inverse Shift Row
Inverse Substitution Bytes
9 Rounds?
No
Yes
Add Round Key
128 Bit Output

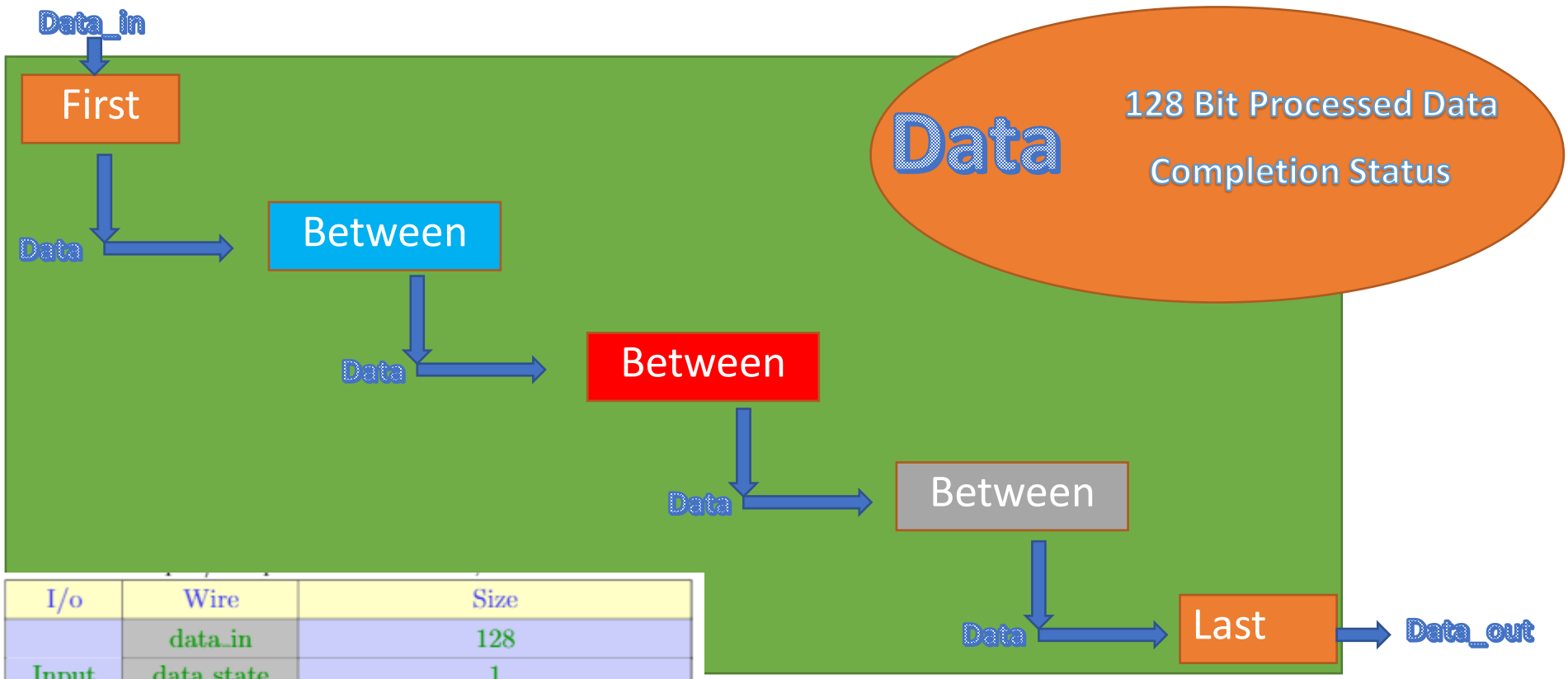
•Circuits


Initial
Add Round Key
Inverse Shift Row
Inverse Substitution Bytes
Mid
Add Round Key
Inverse Mix Col
Inverse Shift Row
Inverse Substitution Bytes
Final
Add Round Key

- FSM Circuits

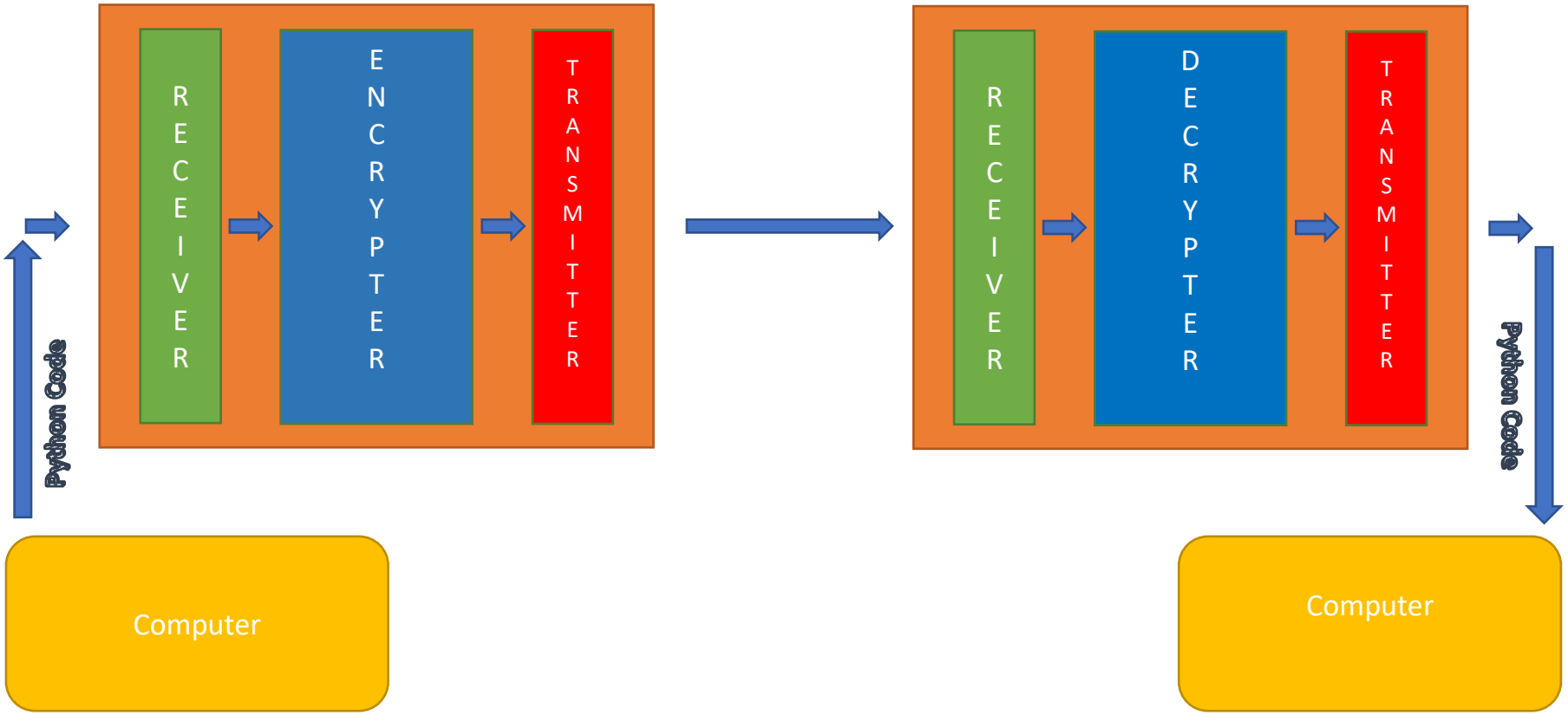


•Decrypter

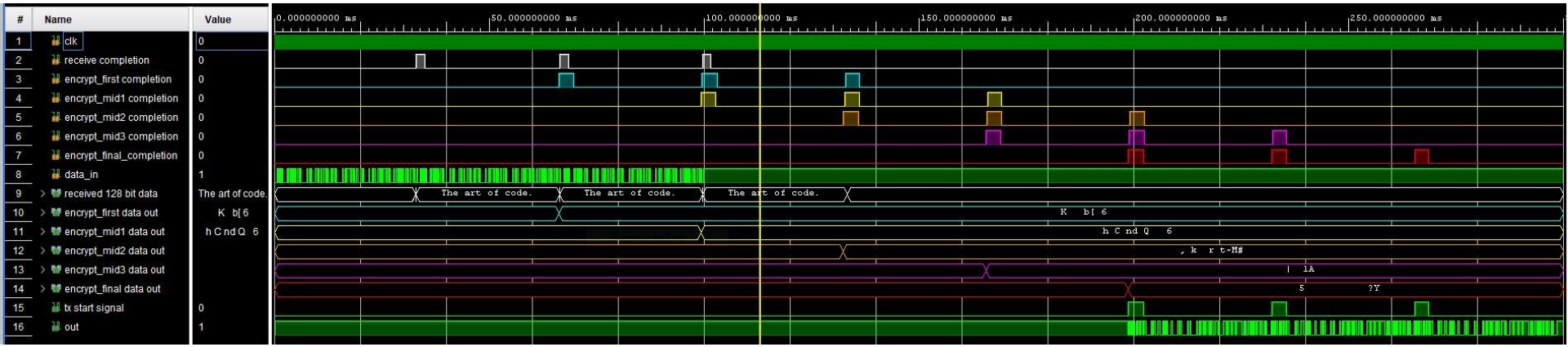


I/o	Wire	Size
	data_in	128
Input	data_state	1
	data_out	128
Output	encode_state	1
These are wires which are connected to internal registers		

•Complete Implementation



Encryption: Simulation



Decryption: Simulation

Challenges

Timing

Worst Negative Slack (WNS):	-18.233 ns
Total Negative Slack (TNS):	-2287.537 ns
Number of Failing Endpoints:	128
Total Number of Endpoints:	1695

Implemented Timing Report

Without Pipeline

Timing — **Setup** | Hold | Pulse Width

Worst Negative Slack (WNS):	-2.918 ns
Total Negative Slack (TNS):	-452.703 ns
Number of Failing Endpoints:	381
Total Number of Endpoints:	2128

Implemented Timing Report

2 Device Pipeline

Timing — **Setup** | Hold | Pulse Width

Worst Negative Slack (WNS):	2.19 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	2803

Implemented Timing Report

5 Device Pipeline