COL215 Hardware Assignment 3b: AES Decryption

Vanshika (2023CS10746)
Laksh Goel (2023CS10848)

November 10, 2024

1 Task

1. Combine Dedicated units to perform various round operations. Like multiplexer, gate array and Multiplier block for GF(28) (MAC): to perform element-wise multiplication to get one output value , into single control unit , and perform the round 10 times.
2. Control the read/write from the memories into the ports of control unit.

2 Design Decisions and Third level abstraction modules

2.1 InvMixColumns

We implemented the module for multiplication of one 8-bit entry of matrix with another 8-bit entry. For this, we divided multiplying into three operations, namely shift by one , shift by two and shift by three.

Shift by one shifts the entry by one value left and takes XOR with 0x1B if highest bit is on. Further shift two uses two instances of shift one to perform shift two, similarly shift three used three instances of shift one to perform shift three.

Inside multiplier module, we perform all three shifts to get three signals. Further, we take xor of whichever three signals out of these are required.

2.2 InvRowShift

After InvMixColumns operation, we are required to rotate elements in a row from right to left. Number of elements to be rotated depends on the row number, as shown in the figure.

We used 4X1 multiplexer for this operation. A 32-bit row and Row number was sent as input, which decides which select vector will be sent to multiplexers. Four multiplexers were required for this purpose, one for each element of a row.

2.3 InvSubbytes

In this, each element is replaced by a value determined using the INV_SBOX matrix, which is stored in ROM as a 256-element vector, each a hexadecimal value.

Input of this code segment is an 8-bit hexadecimal value, which is sent as address to ROM, which then gives required value at output.

2.4 XOR operation

We implemented two XOR modules.
The first one takes two 8 bit hexadecimal and simply XOR them with each other.
Another XOR module takes input as two 128-bit hexadecimal strings and XOR 8-bits at a time, using the clock and assigns the final XOR to a result signal after all the 16 8-bit XOR are computed.

2.5 RAM controller

For reading and writing memory into ram we made a separate module. This module, is an fsm design where there are read , write and wait states. From the initial state according to write enable we move to write state or read state , and after each 8 bit write or read we wait for two wait states. This module using fsm returns me 4 vectors of 32 bit each .

2.6 ROM controller

For reading from ROM we have a separate module which reads from rom 8 bits at a time using fsm (READ and WAIT states) and returns 4 vectors of 32 bit each.

3 second level abstration modules

3.1 Four_row

This module is for shifting the rows of matrix. We send 4 row vector of 32 bit each in this module and using abstraction from part 1 we shift all rows and send new 4 row vectors to my fsm controller.

3.2 Four_sub

This module takes 4 vectors of 32 bit each and uses abstraction of inverse sub_box from hardware 1 to make new 4 vectors of 32 bit each.

3.3 Four_mult

This module takes 4 row vectors and performs multiplication using abstraction from the first part of the assignment and then returns 4 vectors of 32 bit each.

3.4 New_design

In this module we use 4 vectors of 32 bit each and using clock after every 1 second we change the display segment to be displayed and send it to the display plain text module from assignment 2 .

4 Top level of abstraction

4.1 FSM controller

This has an FSM which performs the above task and decrypts the cipher text using the sequence of operations performed given using various states and WAITS .

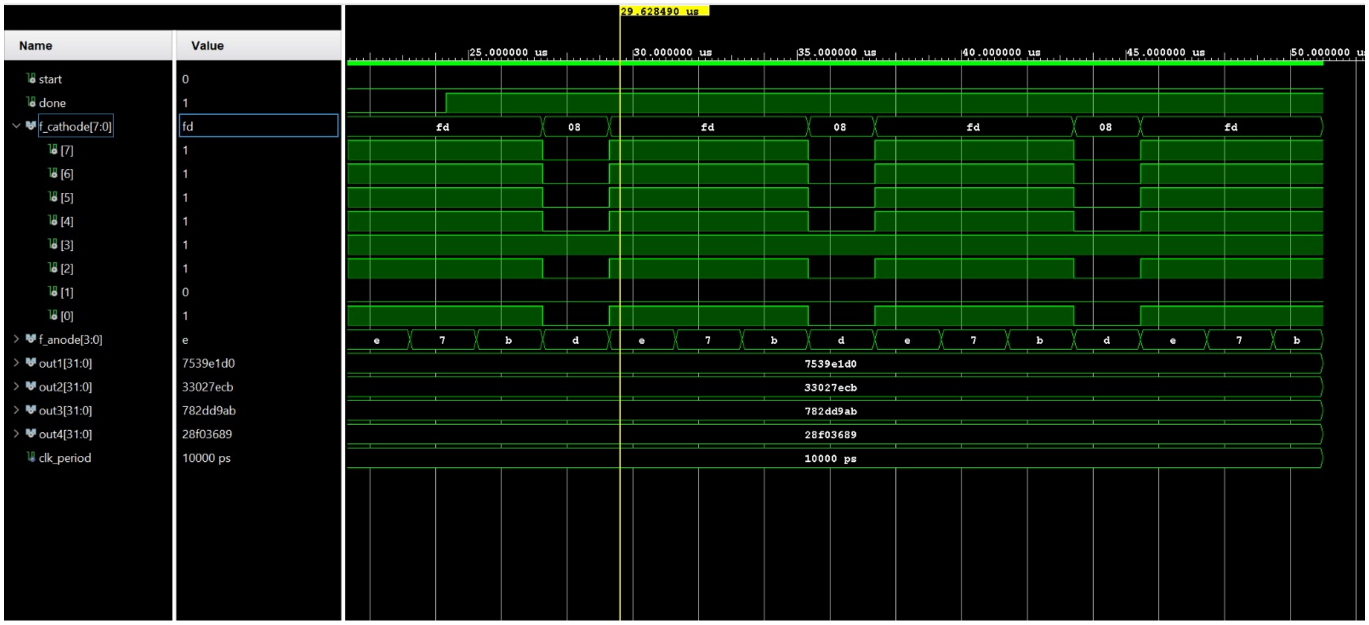
5 Displaying the plain text

We did the following abstraction :

1. One module for converting input 8-bit hexadecimal into it's corresponding binary of 4-bit using ASCII codes.
2. Another module for seven-segment single display from hardware 2
3. Another module for 4 displays from hardware 2
4. Another module that combines all these. It determines scrolling of one character within some constantxclock period time. Then takes 4 displays which are required to displayed at that time. Converts them to 4-bit using the first module. Then sends this to 4 display which further uses seven-segment single display to get cathodes for each character.

6 Simulations

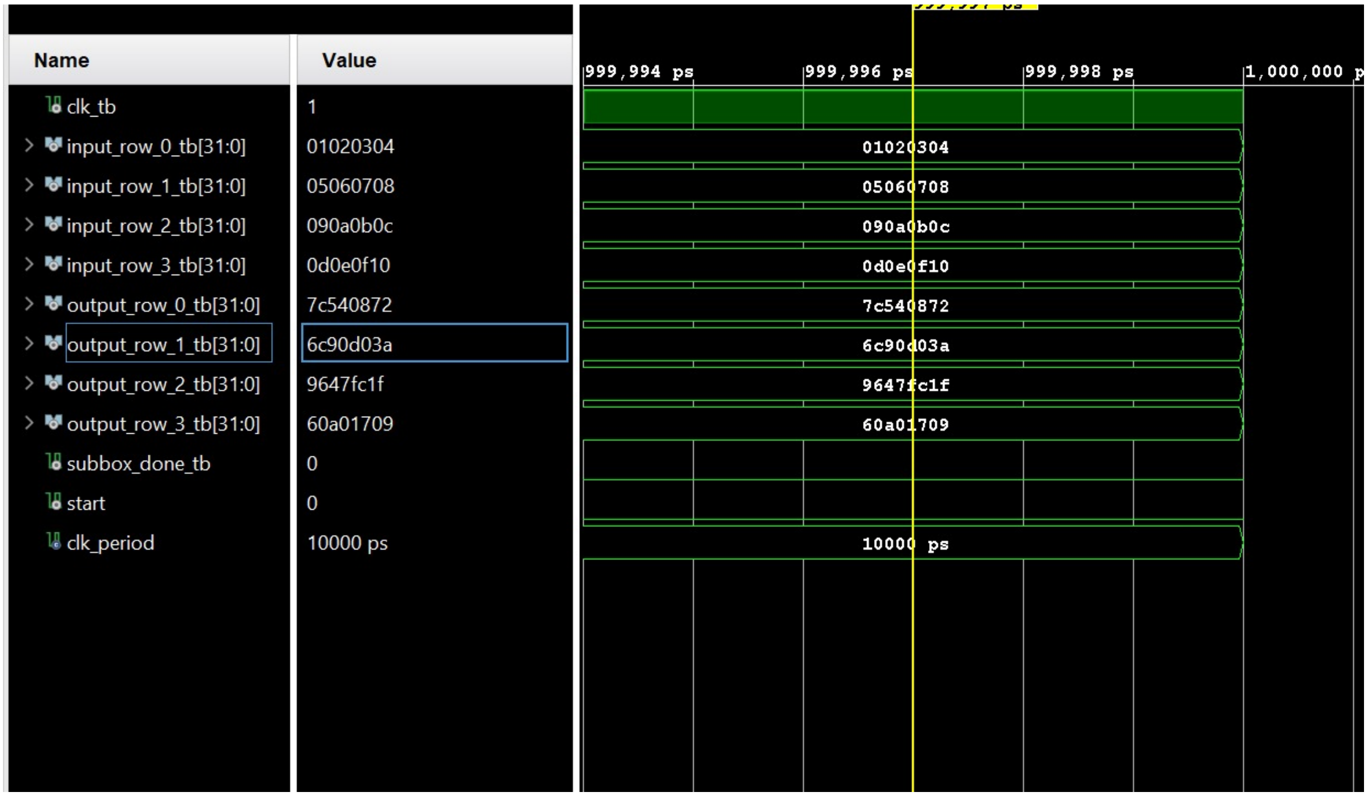
6.1 FINAL OUTPUT ACCORDING TO ROW MAJOR, FOR SOME INPUT



Name	Value
start	0
done	1
f_cathode[7:0]	fd
[7]	1
[6]	1
[5]	1
[4]	1
[3]	1
[2]	1
[1]	0
[0]	1
f_anode[3:0]	e
out1[31:0]	7539e1d0
out2[31:0]	33027ecb
out3[31:0]	782dd9ab
out4[31:0]	28f03689
clk_period	10000 ps

Figure 1: simulation

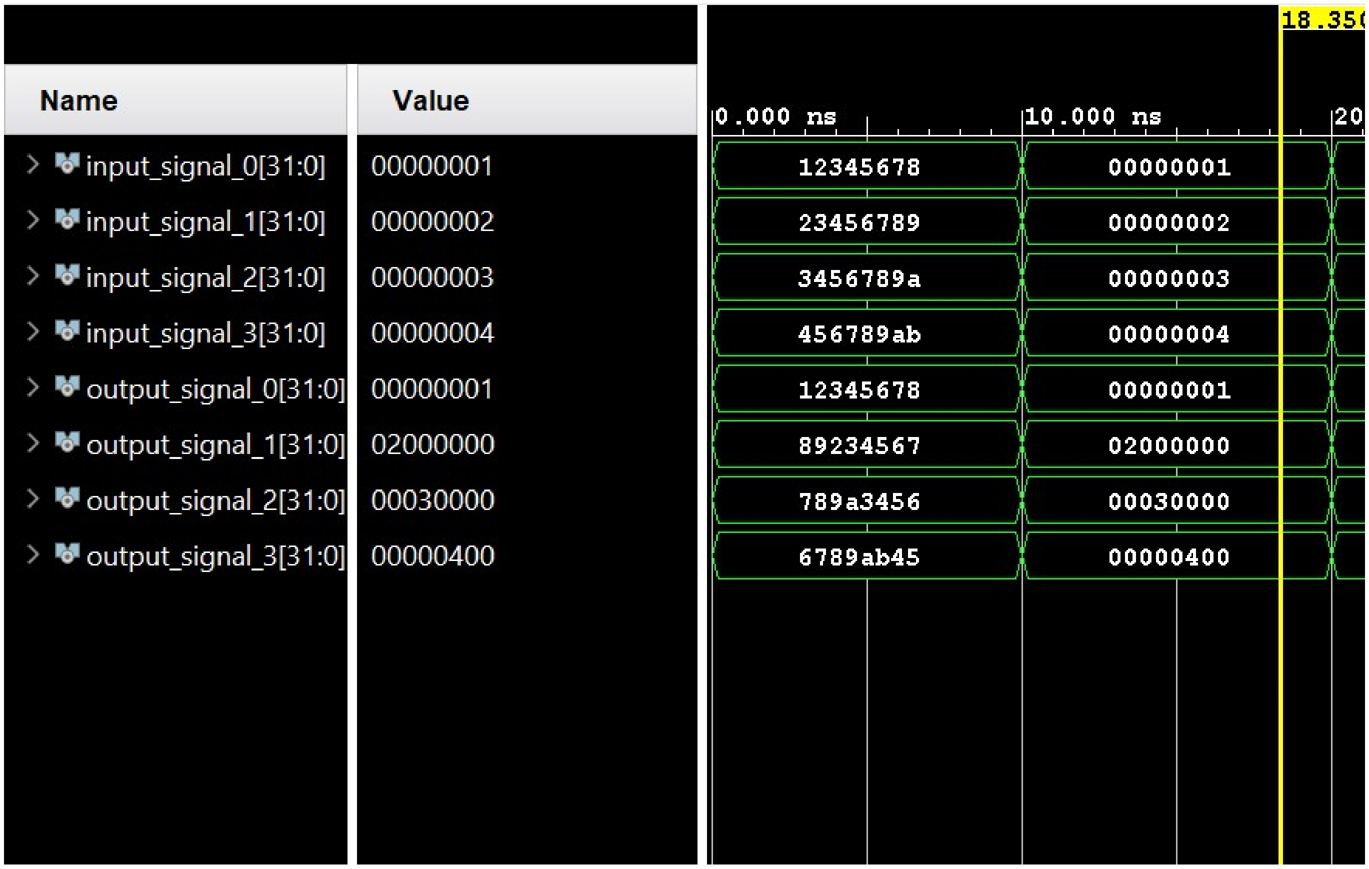
6.2 Inv. sub of whole matrix



Name	Value
clk_tb	1
input_row_0_tb[31:0]	01020304
input_row_1_tb[31:0]	05060708
input_row_2_tb[31:0]	090a0b0c
input_row_3_tb[31:0]	0d0e0f10
output_row_0_tb[31:0]	7c540872
output_row_1_tb[31:0]	6c90d03a
output_row_2_tb[31:0]	9647fc1f
output_row_3_tb[31:0]	60a01709
subbox_done_tb	0
start	0
clk_period	10000 ps

Figure 2: simulation

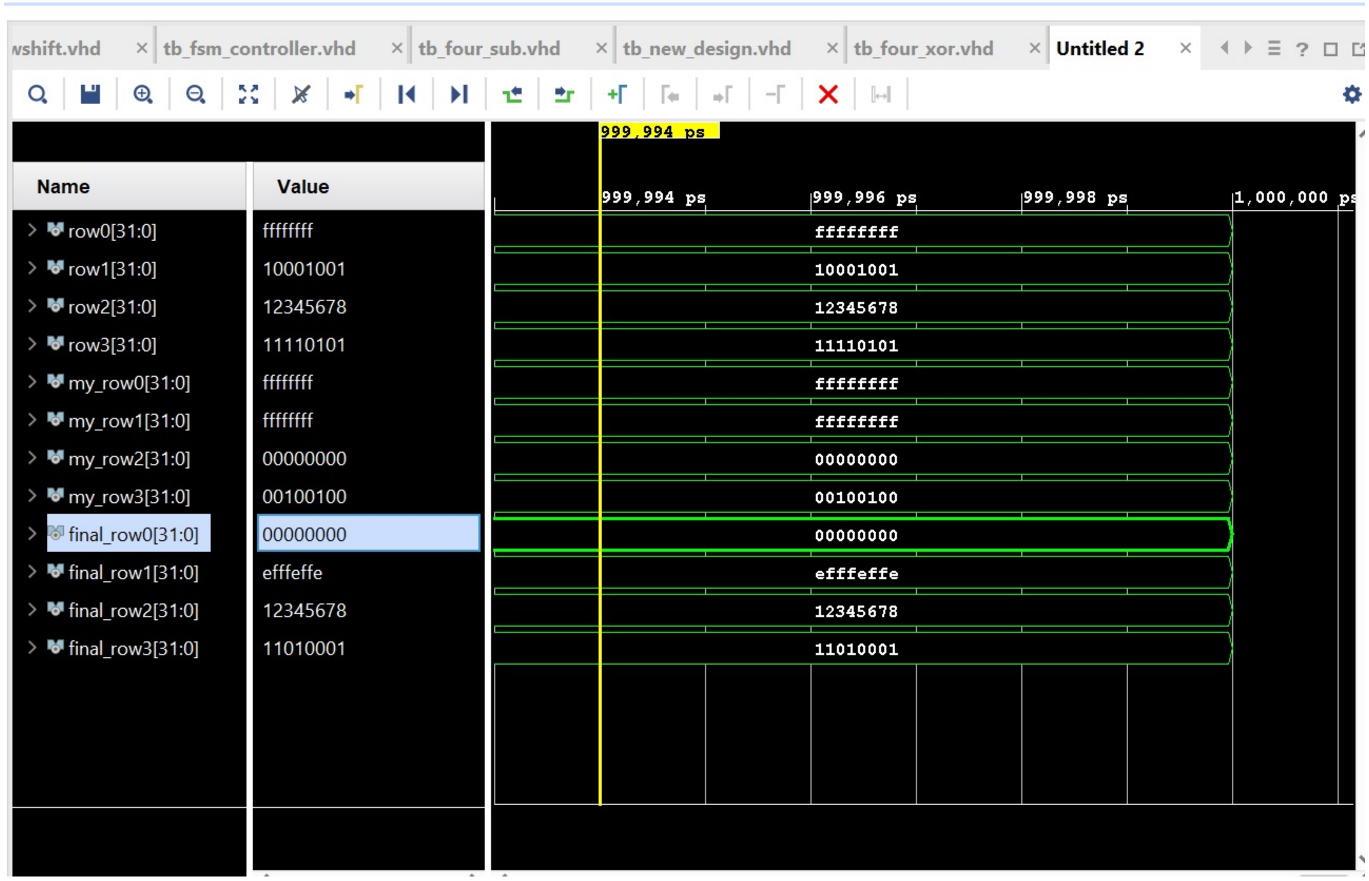
6.3 whole row shift



Name	Value	0.000 ns	10.000 ns
input_signal_0[31:0]	00000001	12345678	00000001
input_signal_1[31:0]	00000002	23456789	00000002
input_signal_2[31:0]	00000003	3456789a	00000003
input_signal_3[31:0]	00000004	456789ab	00000004
output_signal_0[31:0]	00000001	12345678	00000001
output_signal_1[31:0]	02000000	89234567	02000000
output_signal_2[31:0]	00030000	789a3456	00030000
output_signal_3[31:0]	00000400	6789ab45	00000400

Figure 3: simulation

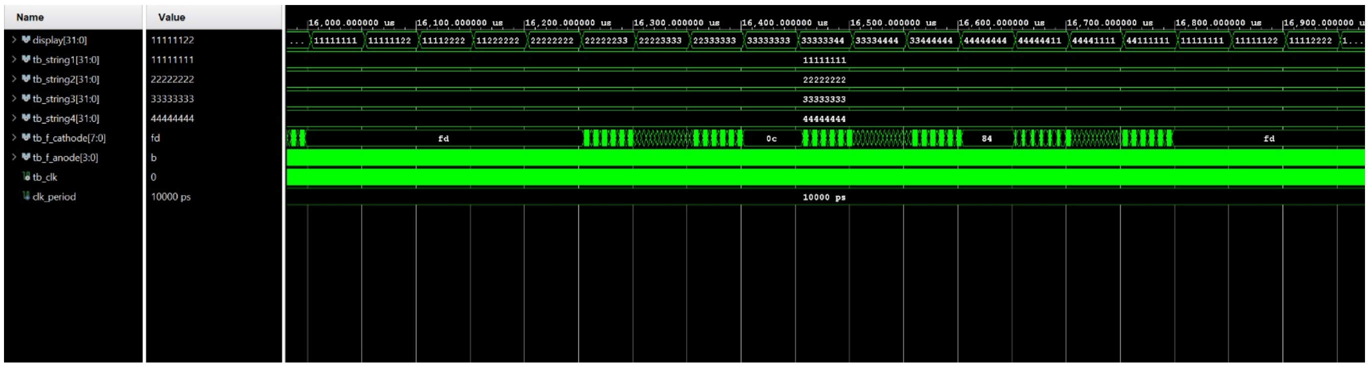
6.4 XOR simulation



Name	Value
row0[31:0]	ffffffff
row1[31:0]	10001001
row2[31:0]	12345678
row3[31:0]	11110101
my_row0[31:0]	ffffffff
my_row1[31:0]	ffffffff
my_row2[31:0]	00000000
my_row3[31:0]	00100100
final_row0[31:0]	00000000
final_row1[31:0]	efffeffe
final_row2[31:0]	12345678
final_row3[31:0]	11010001

Figure 4: simulation

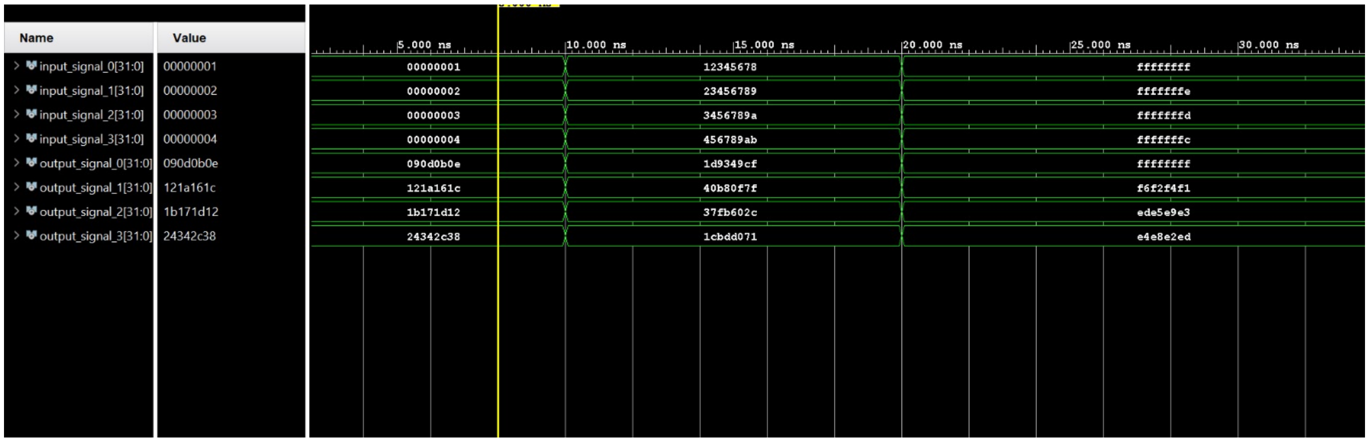
6.5 DISPLAY AND SCROLLING SHOWN



Name	Value
display[31:0]	11111122
tb_string1[31:0]	11111111
tb_string2[31:0]	22222222
tb_string3[31:0]	33333333
tb_string4[31:0]	44444444
tb_f_cathode[7:0]	fd
tb_f_anode[3:0]	b
tb_clk	0
clk_period	10000 ps

Figure 5: simulation

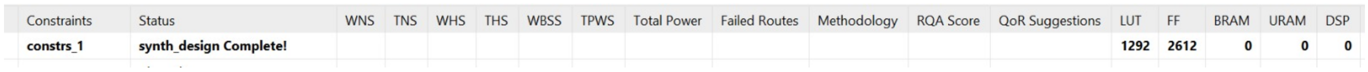
6.6 whole matrix multiplication (inv mix columns)



Name	Value			
input_signal_0[31:0]	00000001	00000001	12345678	ffffffff
input_signal_1[31:0]	00000002	00000002	23456789	fffffffe
input_signal_2[31:0]	00000003	00000003	3456789a	fffffffd
input_signal_3[31:0]	00000004	00000004	456789ab	fffffffc
output_signal_0[31:0]	090d0b0e	090d0b0e	1d9349cf	ffffffff
output_signal_1[31:0]	121a161c	121a161c	40b80f7f	f6f2f4f1
output_signal_2[31:0]	1b171d12	1b171d12	37fb602c	ede5e9e3
output_signal_3[31:0]	24342c38	24342c38	1cbdd071	e4e8e2ed

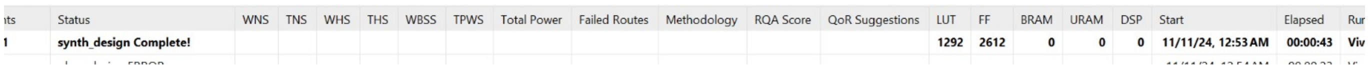
Figure 6: simulation

7 Resource utilization



Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP
constrs_1	synth_design Complete!												1292	2612	0	0	0

Figure 7: resource1



ts	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Ru
1	synth_design Complete!												1292	2612	0	0	0	11/11/24, 12:53 AM	00:00:43	Viv

Figure 8: resource2

8 Schematic

8.1 FSM

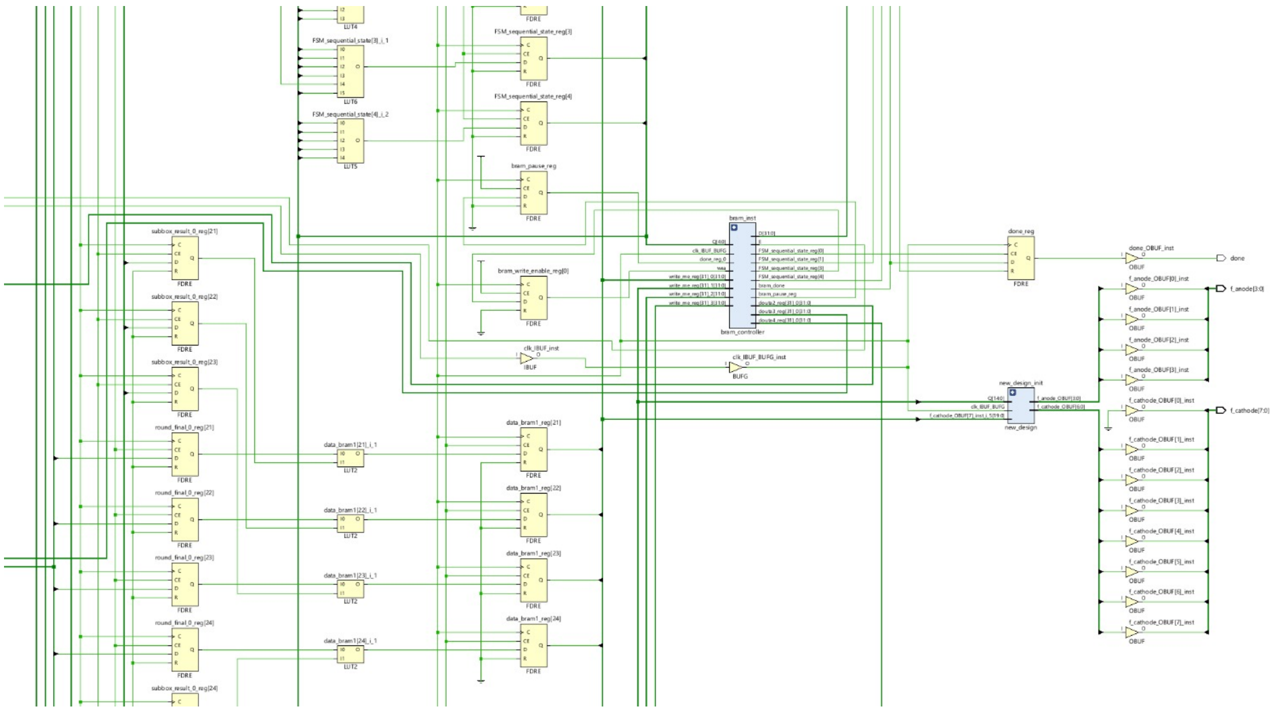


Figure 9: FSM schematic 1

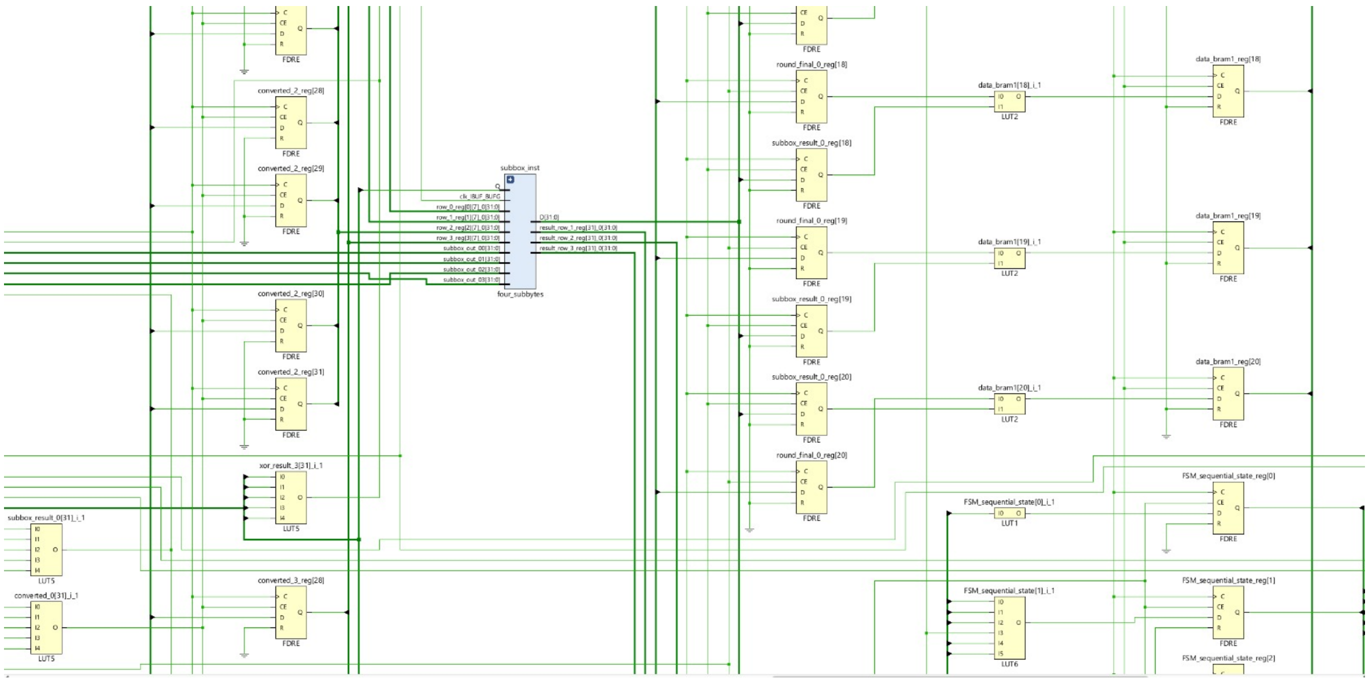


Figure 10: FSM schematic 2

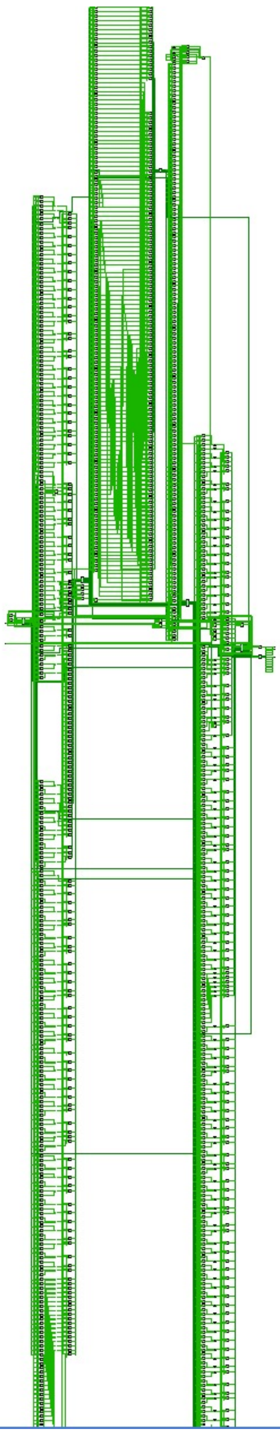


Figure 11: FSM schematic 3

9 Block diagram

9.1 whole matrix multiplication (inv mix columns)

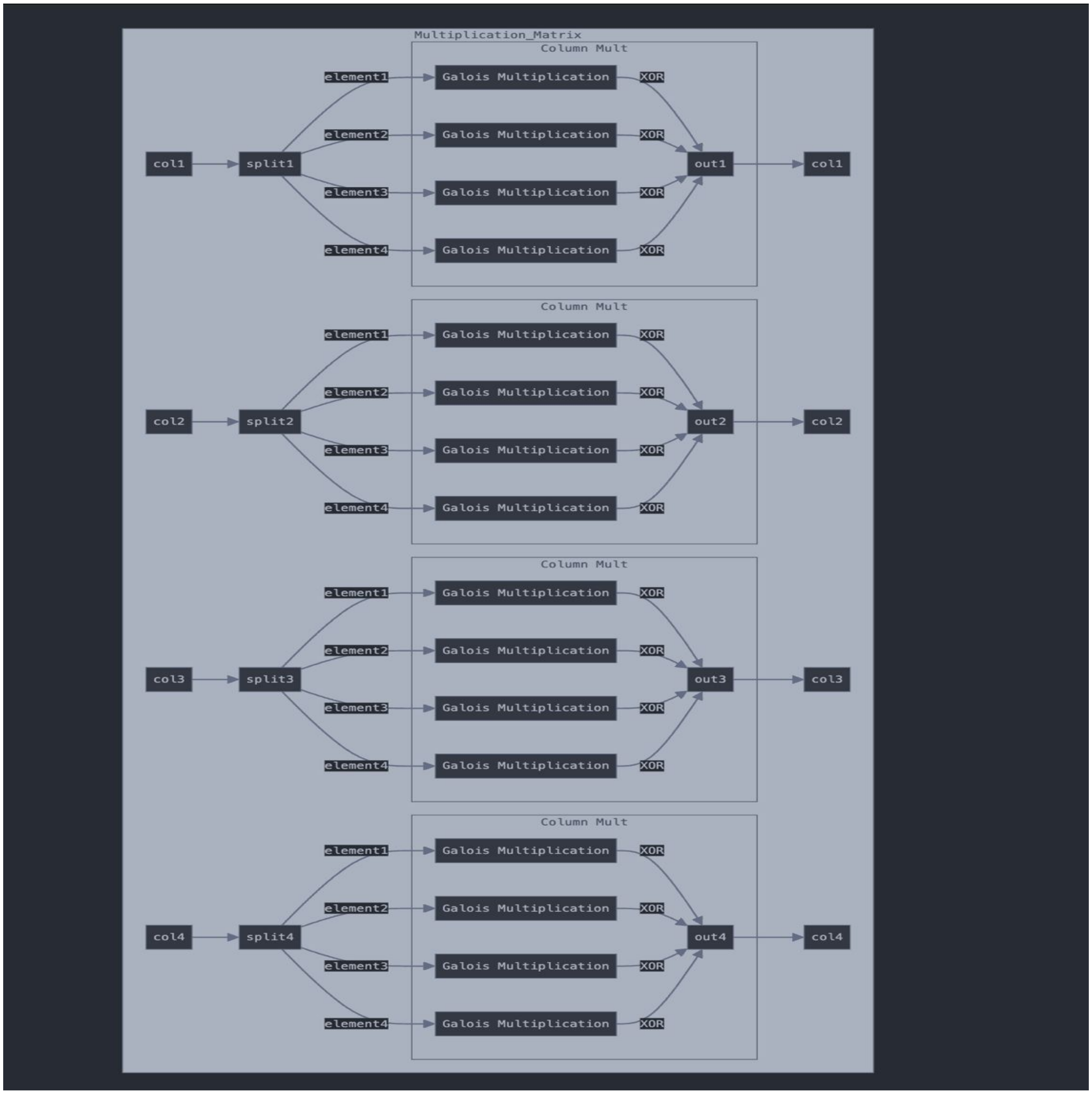


Figure 12: block diagram

9.2 Inv. sub of whole matrix

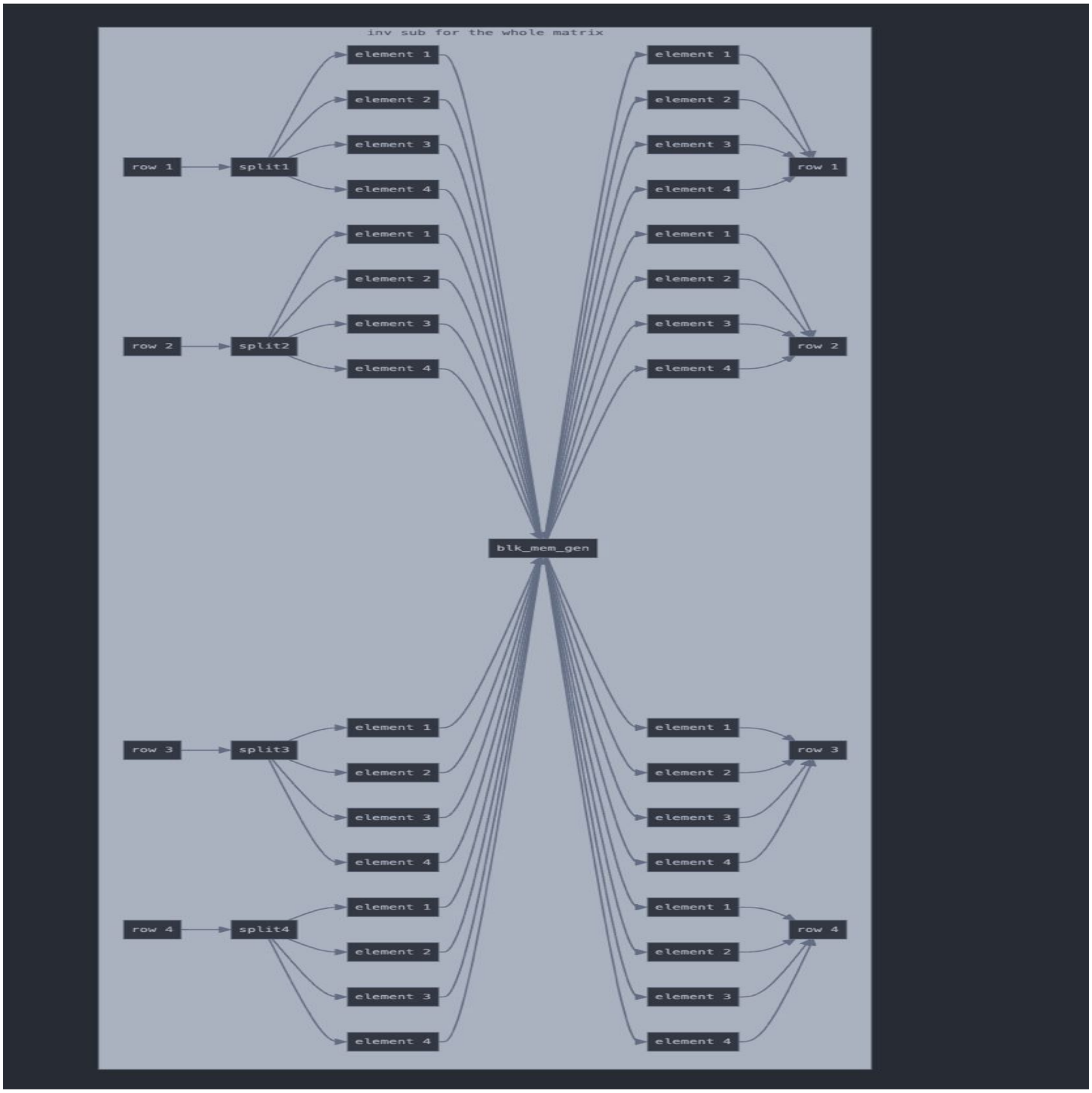


Figure 13: block diagram

9.3 whole row shift

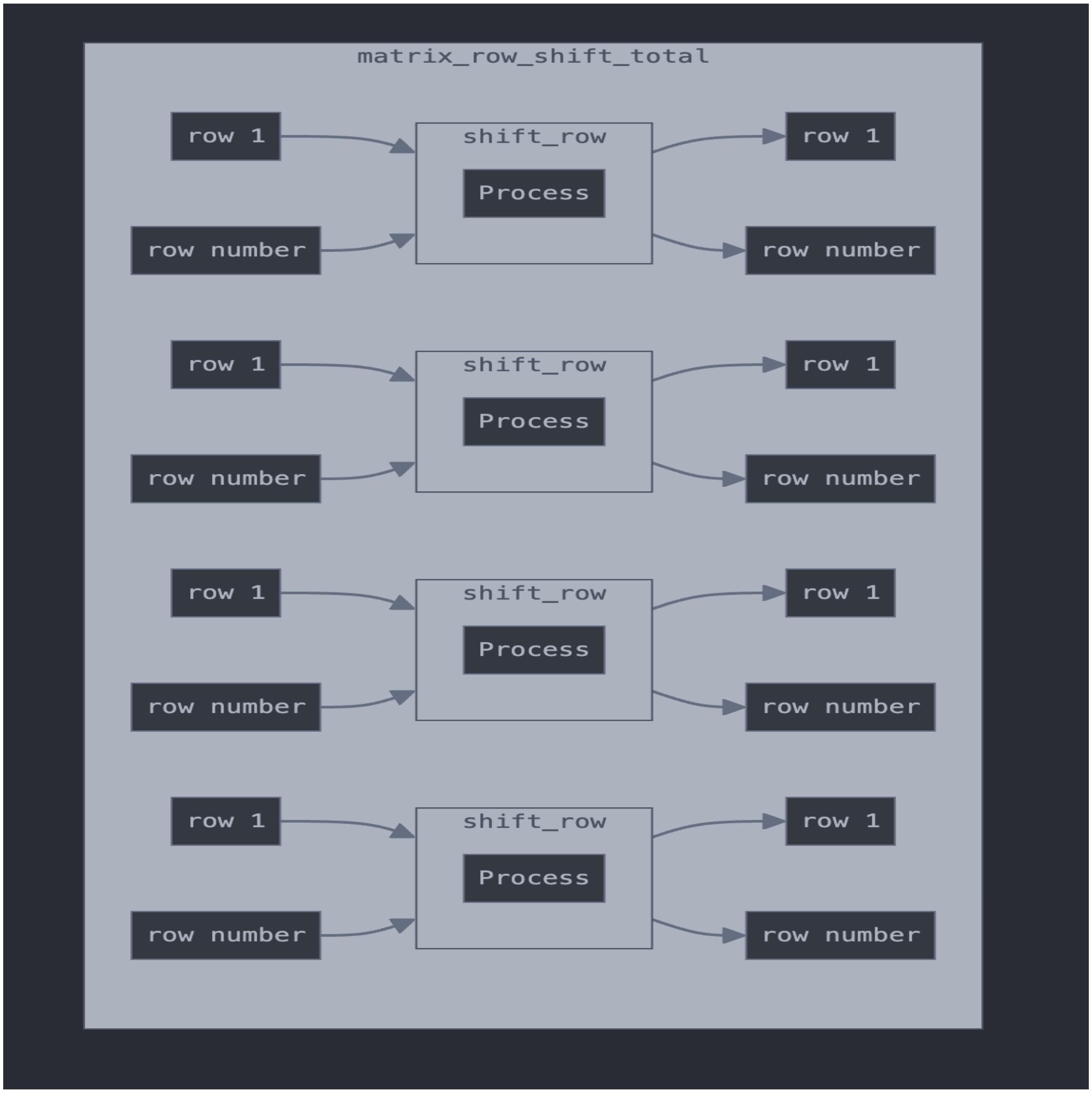


Figure 14: block diagram

9.4 XOR

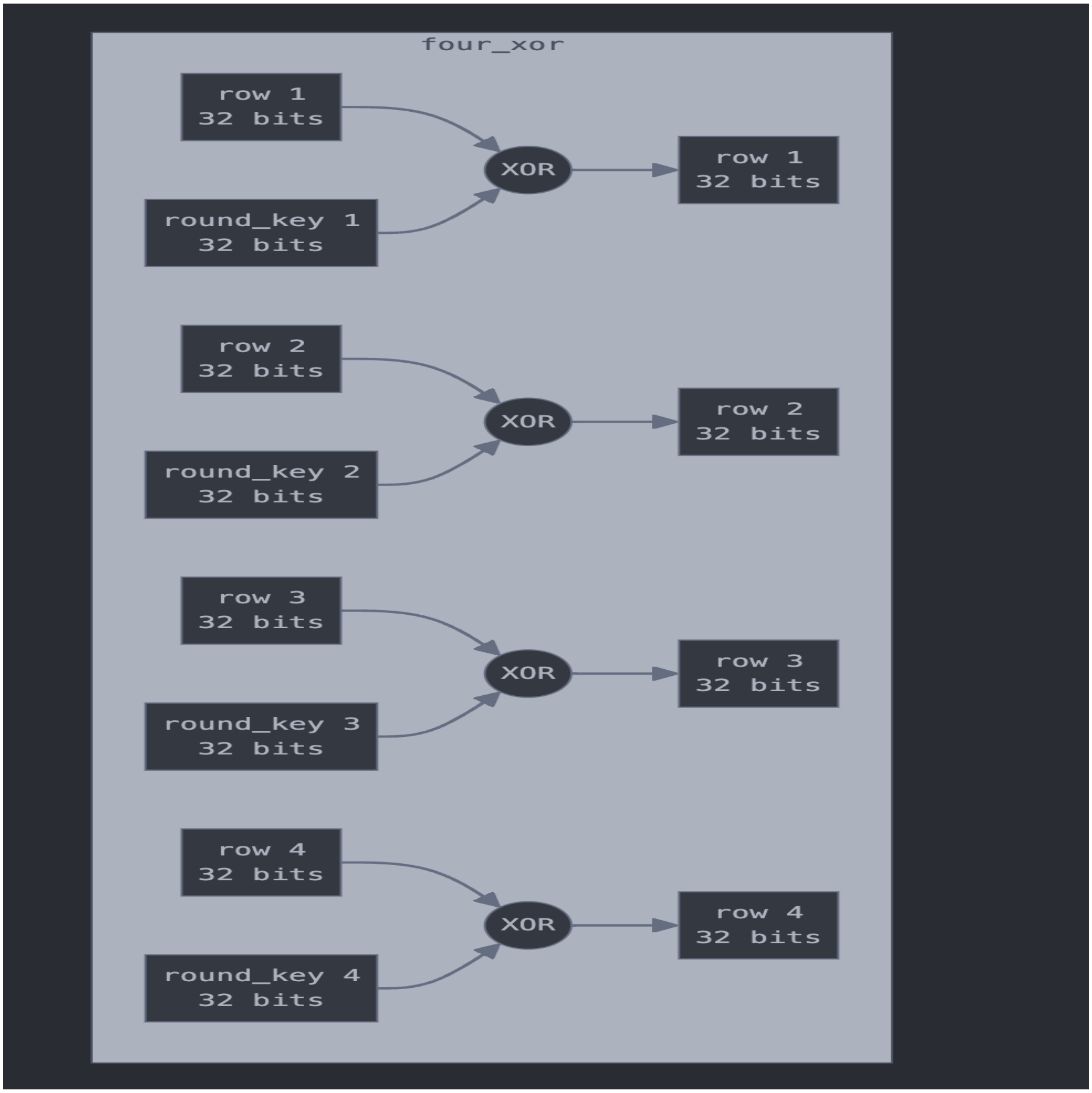


Figure 15: block diagram

9.5 FSM

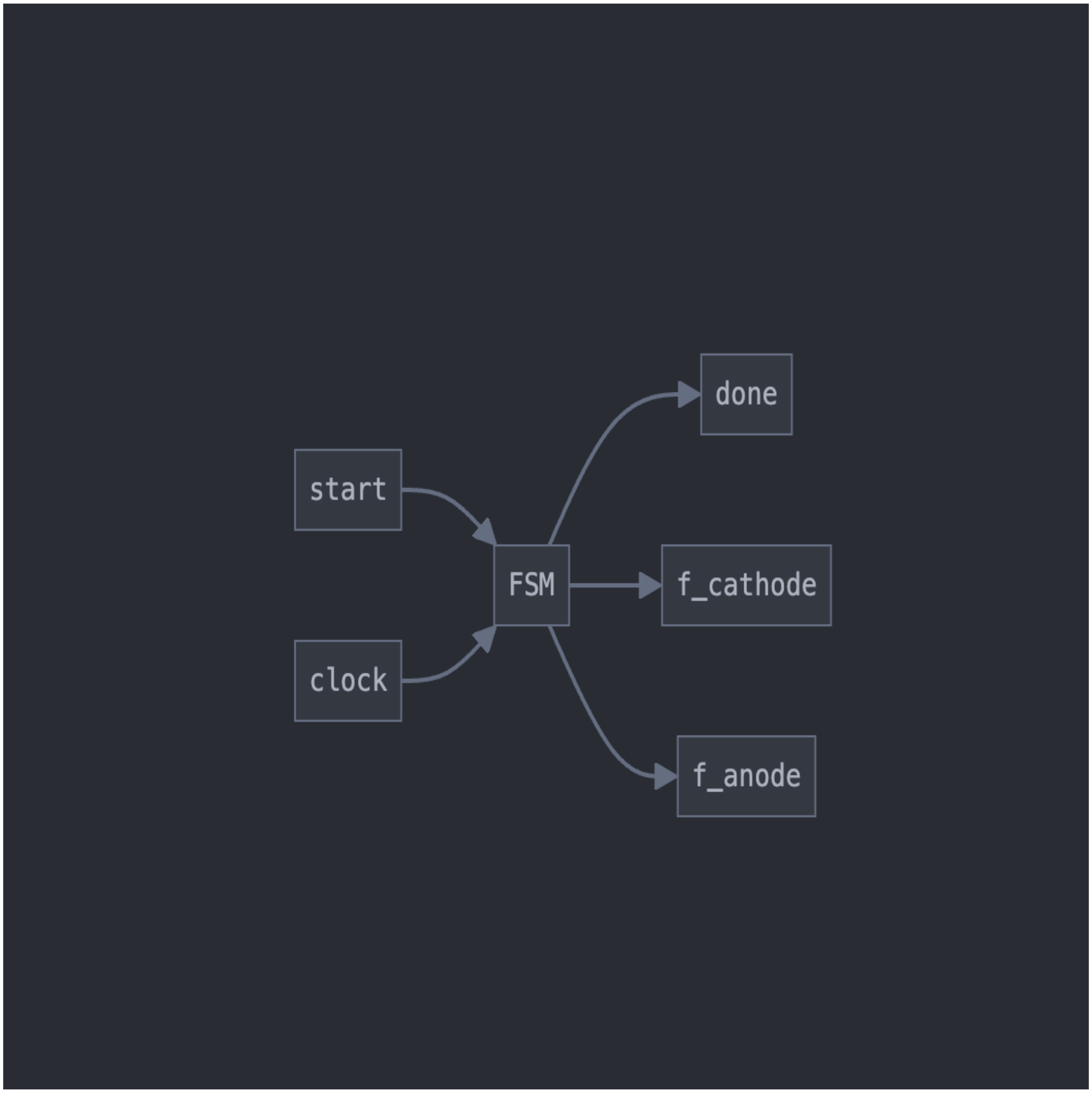


Figure 16: block diagram

10 simulation depicting the change of states in final fsm

NOTE: STATE 1 = READ
STATE 2 = XOR
STATE 3= MULTIPLY
STATE 4 = INVERSE ROW SHIFT
STATE 5 = INVERSE SUB BYTES
STATE 6 = WRITE
STATE 7 = PAUSE(DONE)

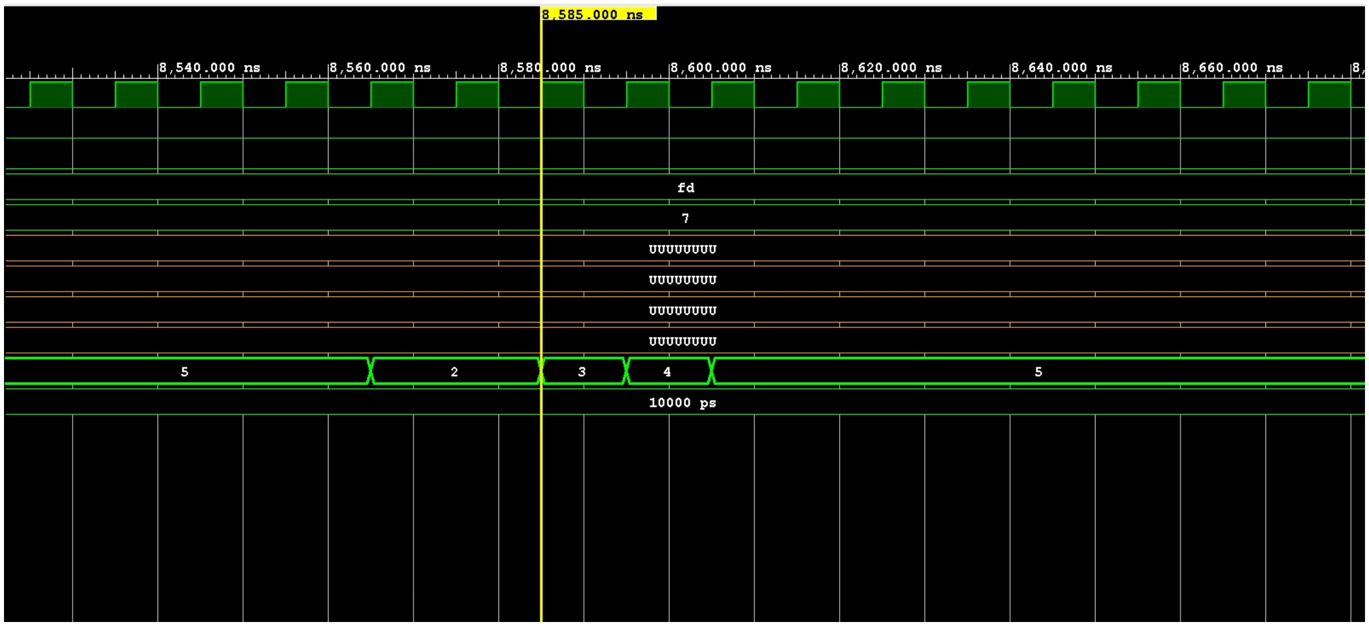


Figure 17: starts from 5 , goes to 2 , 3, 4 again 5

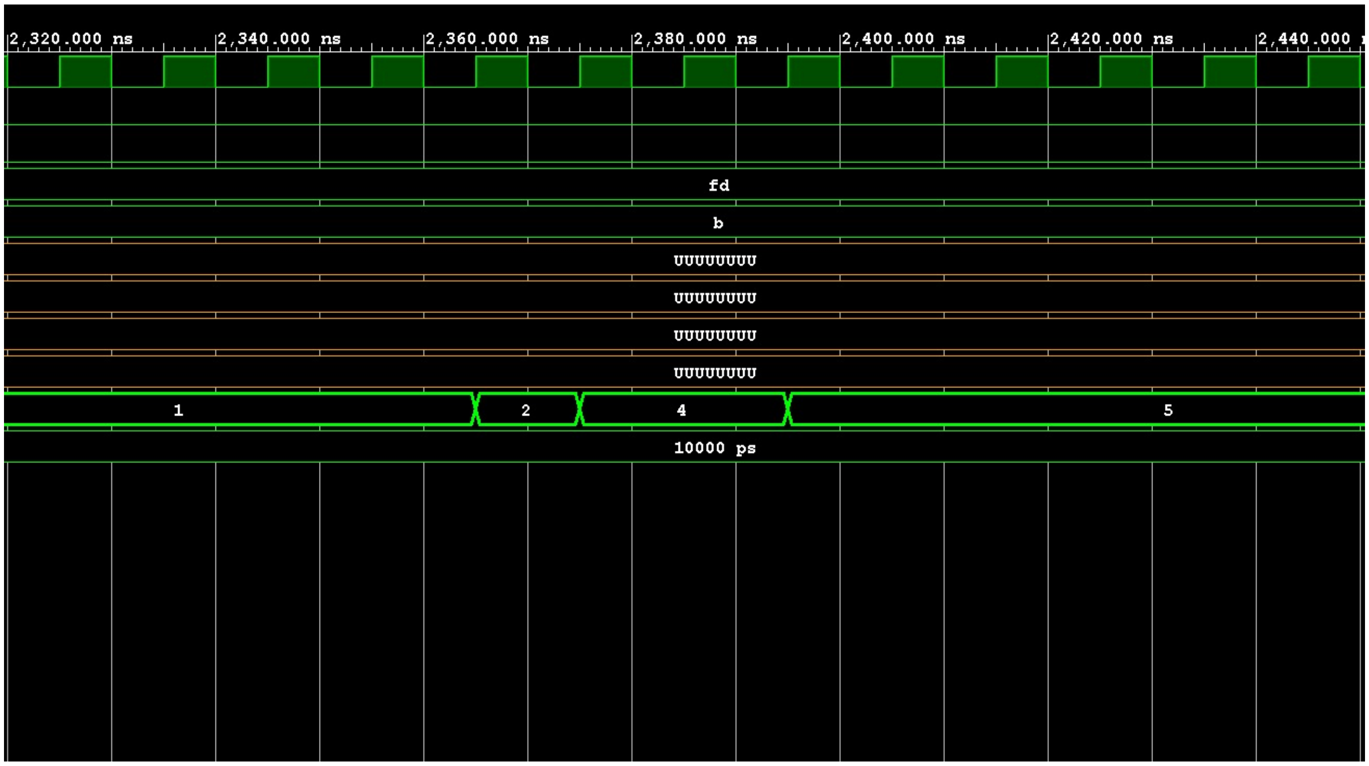


Figure 18: goes to state 1, then 2 , then 4 then 5

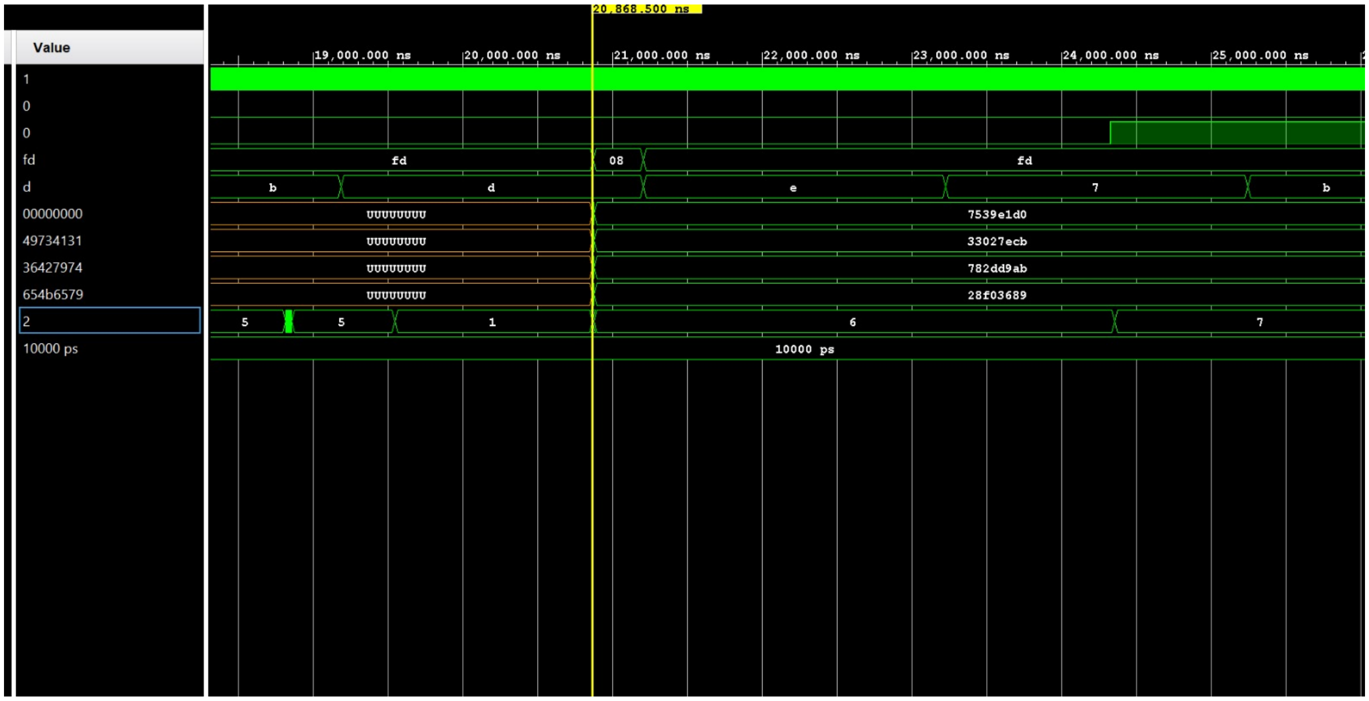


Figure 19: there are rounds, 5 to 5 then 5 to 5 and so on, after that it does reading(state 1) then the cursor marks the point 2 which final XOR and then final writing state 6 and then pause which is state 7

11 STATE TRANSITION DIAGRAM OF FSM

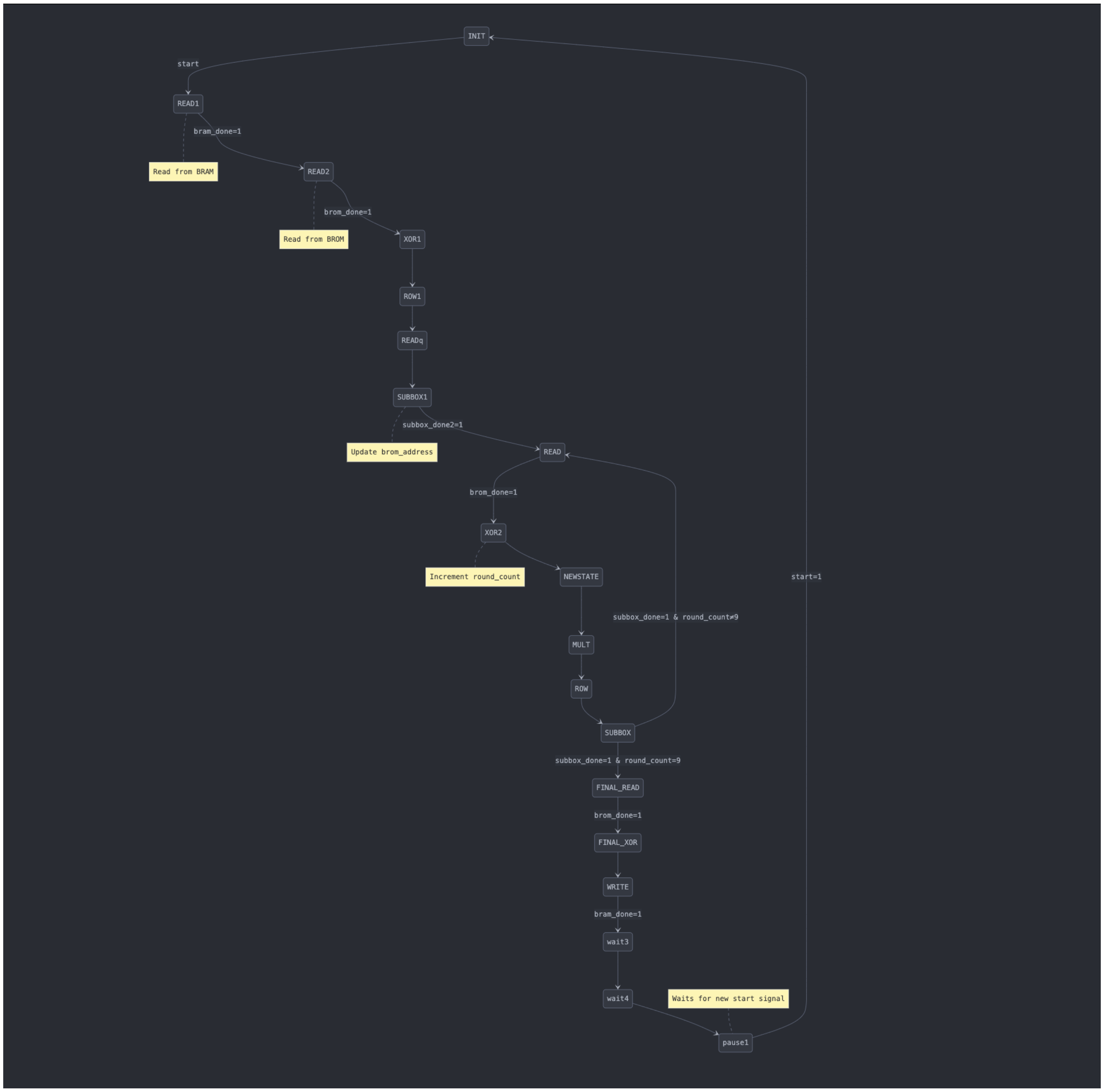


Figure 20: state transition diagram