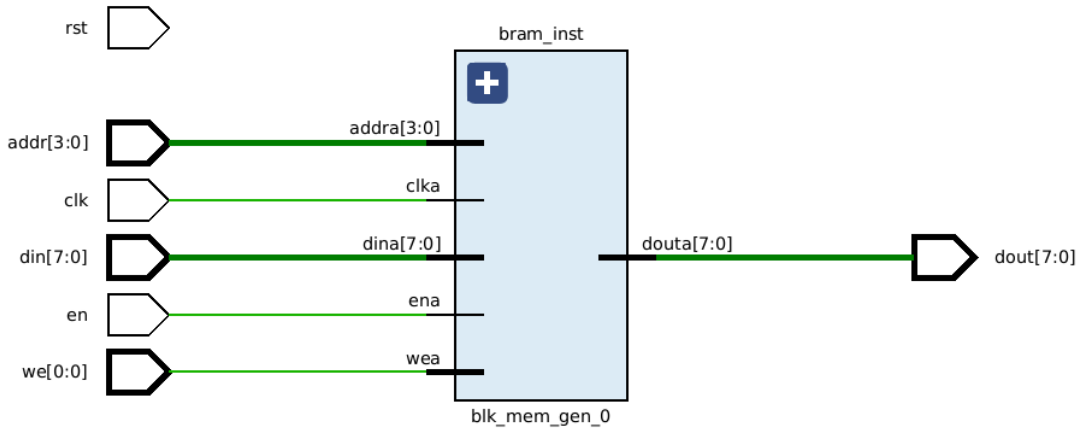
Hardware Assignment-3 Part-2
AES decryption operation
We took input of 128 bits for each component, and read RAM block 8 bits at a time.

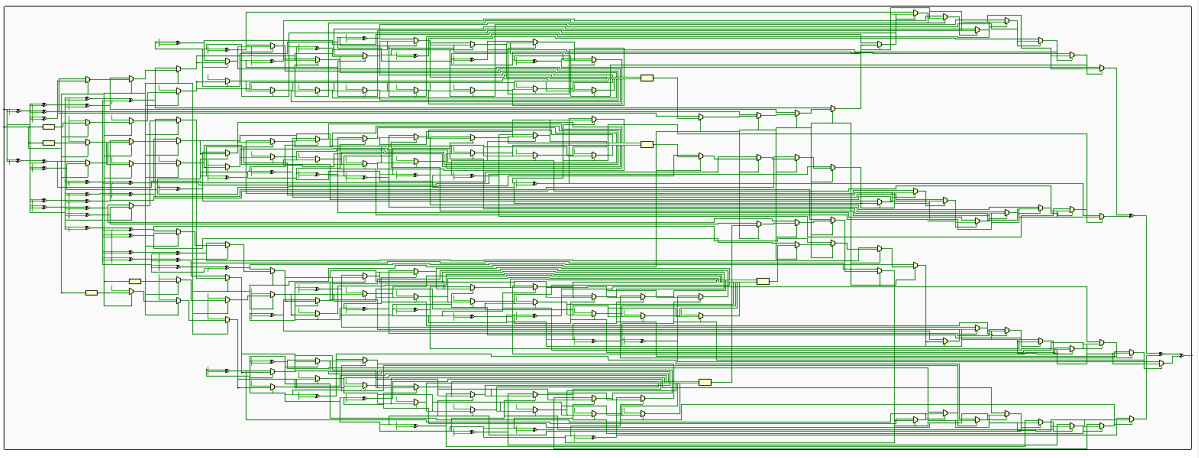
Explanation of each sub-component

1. BRAM Instantiate



2. Inverse Mixed Columns

Galois Field Multiplication
Here, we implemented GF(2^8) for specific row of inverse mix columns matrix and column input matrix.



0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

*

8B	0C	68	DA
42	70	43	4E
6D	30	00	D7
D5	1F	8A	EE

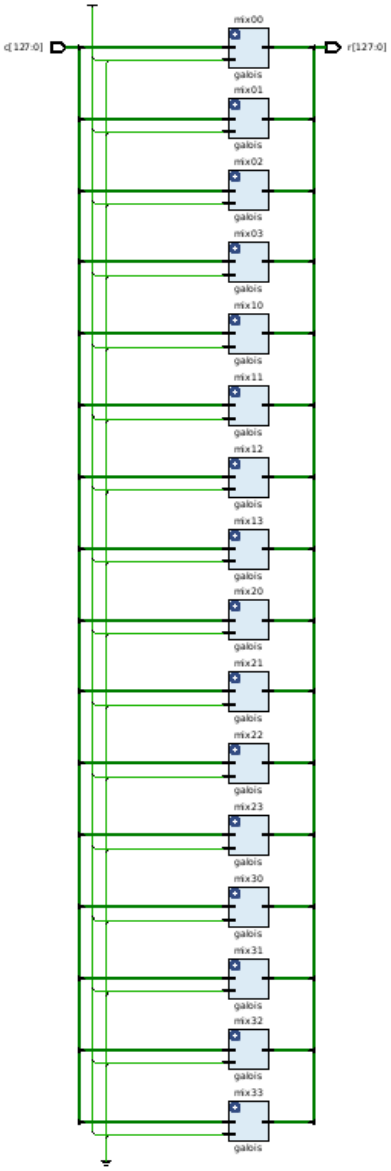
=

63	F9	5B	6F
A2	AA	12	63
67	63	6A	23
D7	63	82	82

Here M_{00} = xtimes_0e(c_{00}) ⊕ xtimes_0b(c_{01}) ⊕ xtimes_0d(c_{02}) ⊕ xtimes_09(c_{03}) and so on.

Where M is output matrix.
We chose to define polynomial $x^8 + x^4 + x^3 + x + 1$ as std_logic_vector 100011011, and took modulo after shifting and xor operations.
For example, for multiplication with x0e, i.e. 1110 (x^3+x^2+x), we xor'ed element shifted by 1, shifted by 2 and shifted by 3, then took the modulo with 9-bit polynomial.

-For Inverse Mixed Columns we finally combined all Galois field operations in one matrix

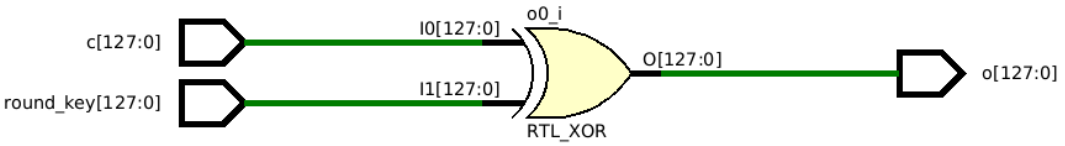


3. InvShiftRows

No shift for 0th row, shift last element to left for 1st row, shift last 2 elements to left for 2nd row, and shift last 3 elements to left for 3rd row.
For Inverse Shift Rows, we did not use muxes. We used logic_vector slicing and concatenation operator &.

4. Add Round Key

We simply xor'ed 128 bit input with round key.



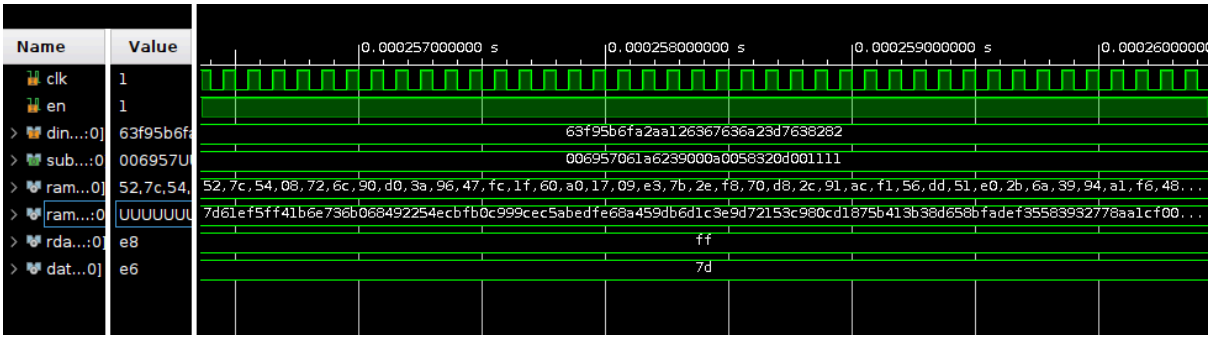
5. Display unit

Since we are supposed to represent only ASCII characters 0-9 and A-F, blank space and - for the rest, we considered hex values 20 (blank), 30-39 (0-9), and 41-46 (A-F).
Then, we created truth table for the same and implemented a, b, c, d, e, f, g for 7-segment decoder through if-else statements.
We used simultaneous 4 digit display from HW-2.
For scrolling, we implemented timing block to display each row of matrix for 1-2 s each (for simulation).

Design Decisions

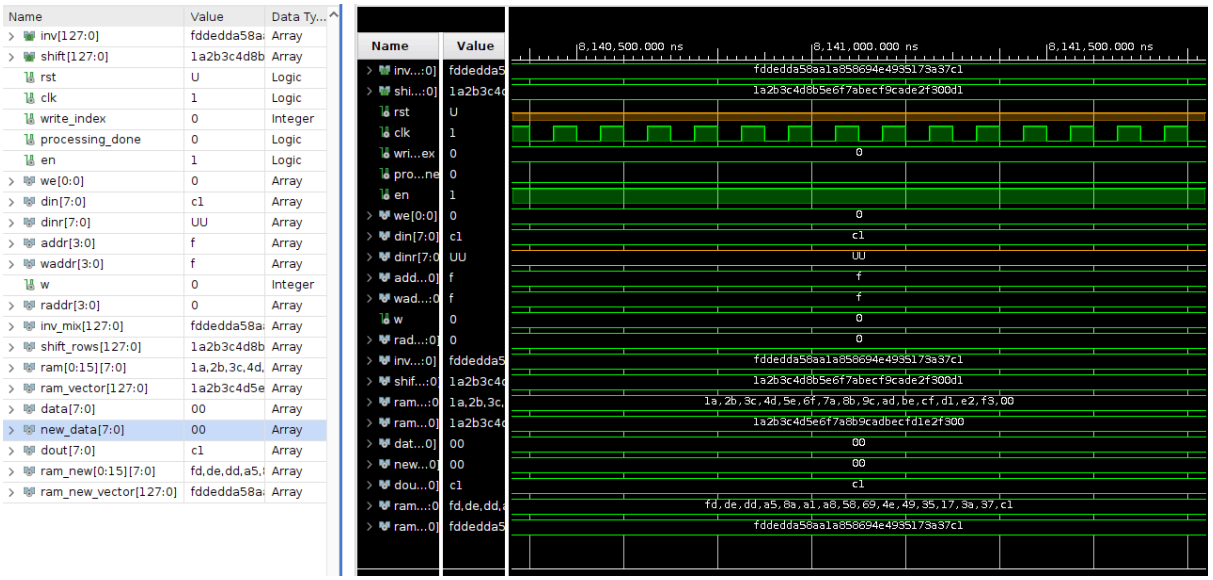
1. For Inverse SubBytes, considering input file to be column major, we took input of 128 bits. For ROM file reading we extracted 8 bits at a time and address of 8 bits. The value is stored in array ram and ram_vector, and operations performed to yield output subbytes.

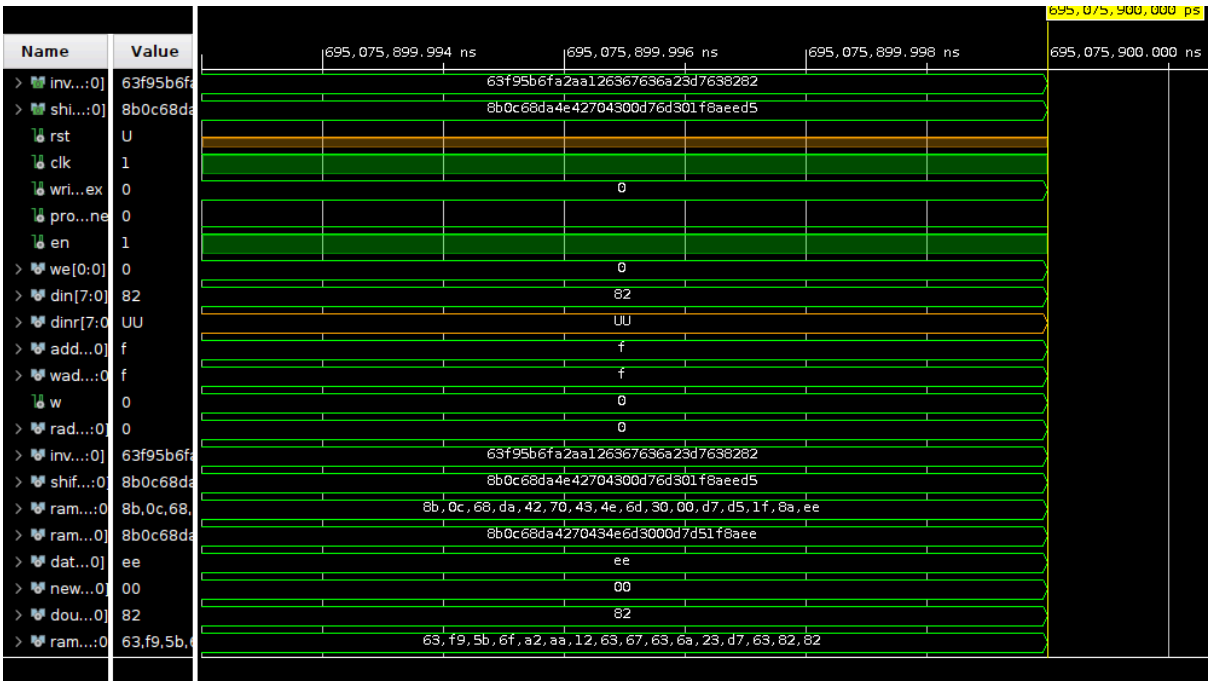
Inverse SubBytes (Reading from invsubbytes.coe)



2. Inverse Mixed Columns and
3. Inverse Shift Rows

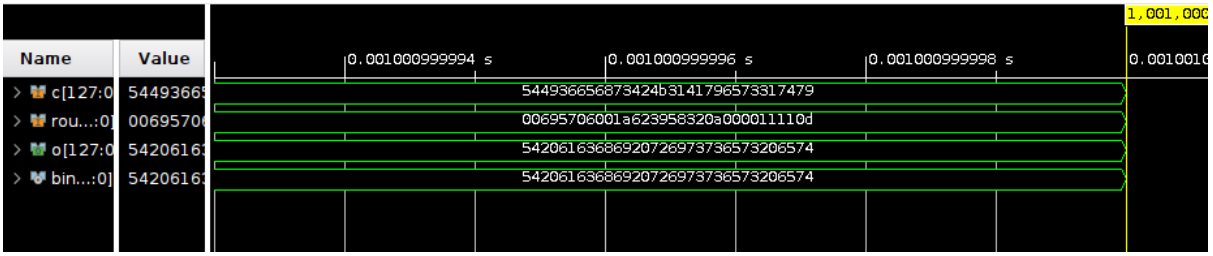
For Inverse Mixed Columns, we created component galois for implementing Galois Field Arithmetic, for each column of input and row of inverse matrix.
(Reading from sample.coe)
This is bram_access.vhd



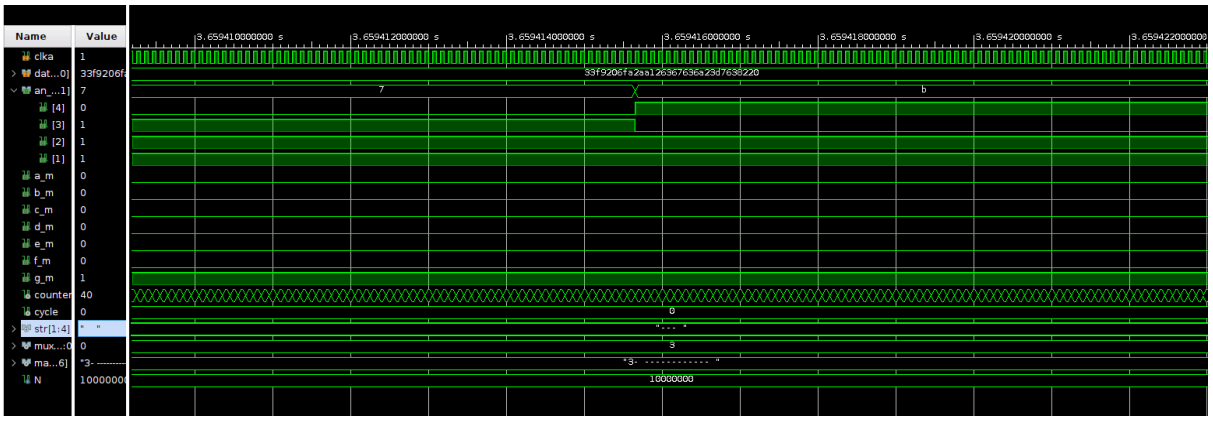


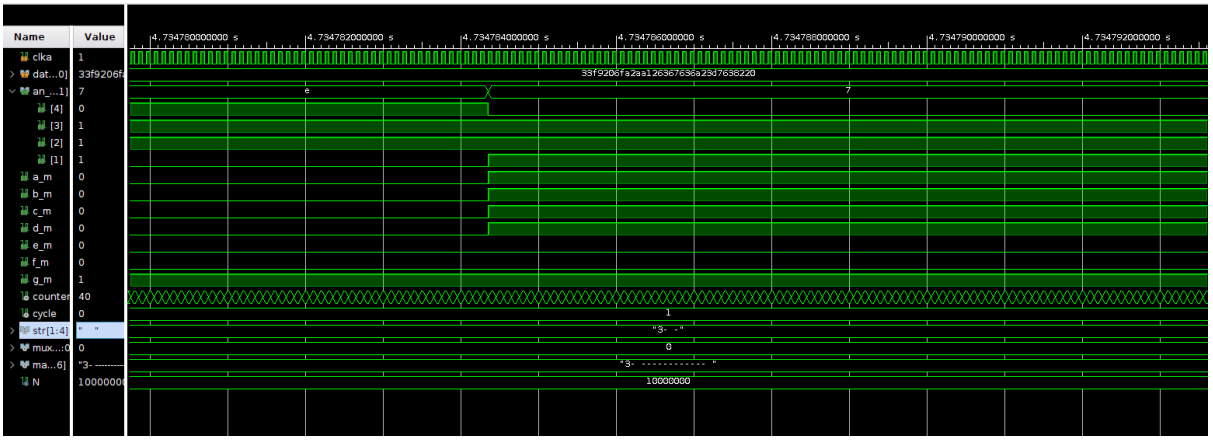
Here, ram_vector represents values read from initial .coe file, and ram_new_vector represents values read from updated .coe file after performing inverse mixed columns.

4. Add Round Key



5. Display ASCII on 7 segment (scrolling)





For FSM implementation:

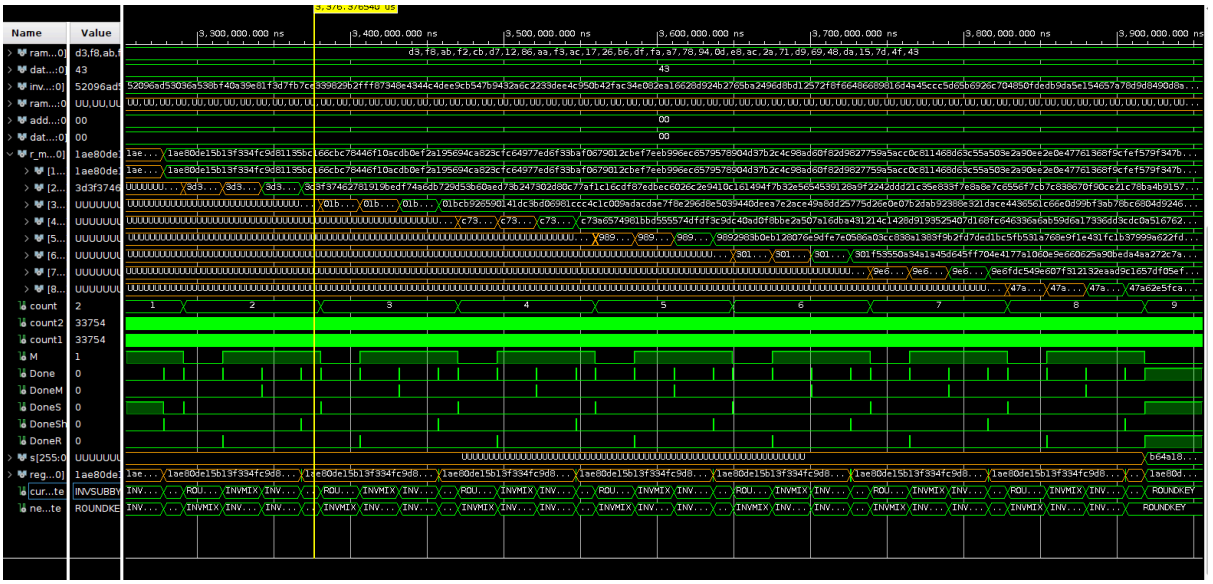
For 128 bits:

We maintained count from 0 to 8 rounds and performed 0th round and last round (9th) differently.

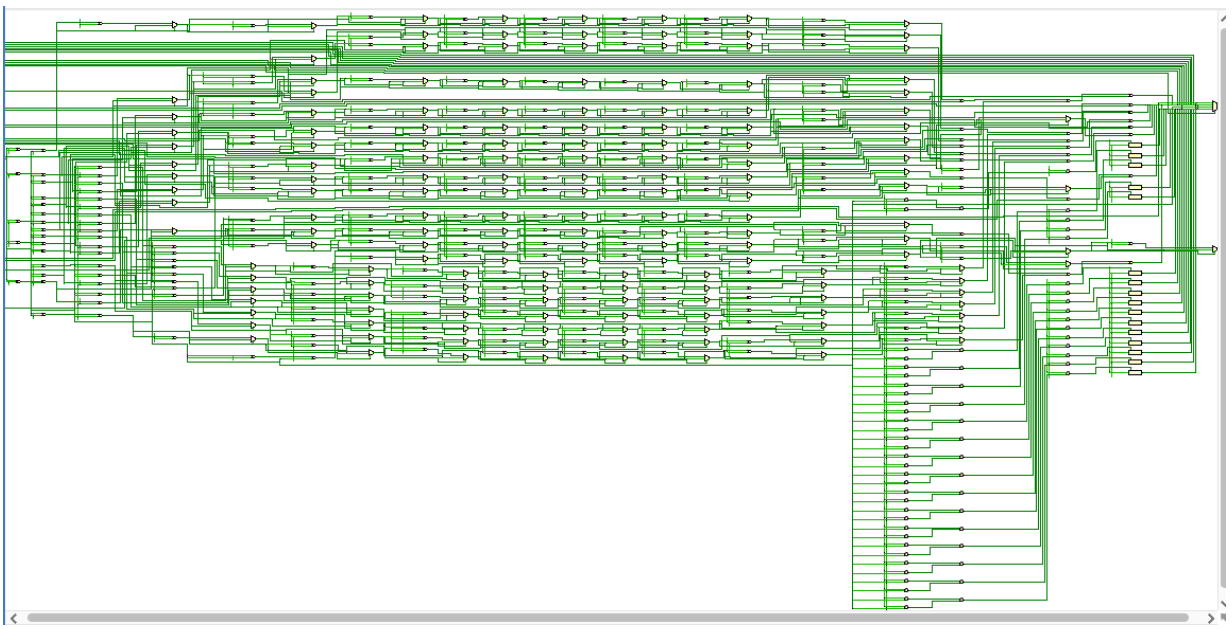
We defined r_main register (array of length 8 (round) and width 640). For each operation, it stores at first the input (639 downto 512), then the round key xor result (511 downto 384), the inverse mixed columns operation result (383 downto 256), inverse shift rows operation (255 downto 128) and finally the inverse sub bytes operation (127 downto 0).

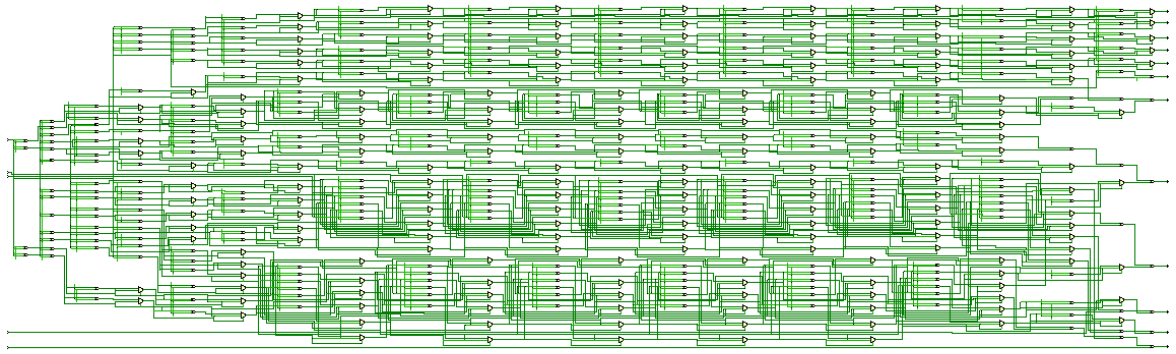
The DoneM, DoneS, DoneSh and DoneR signals store when each operation has been completed. The Done signal stores when final operations have been completed. It is only after this delay that the value enters the register.

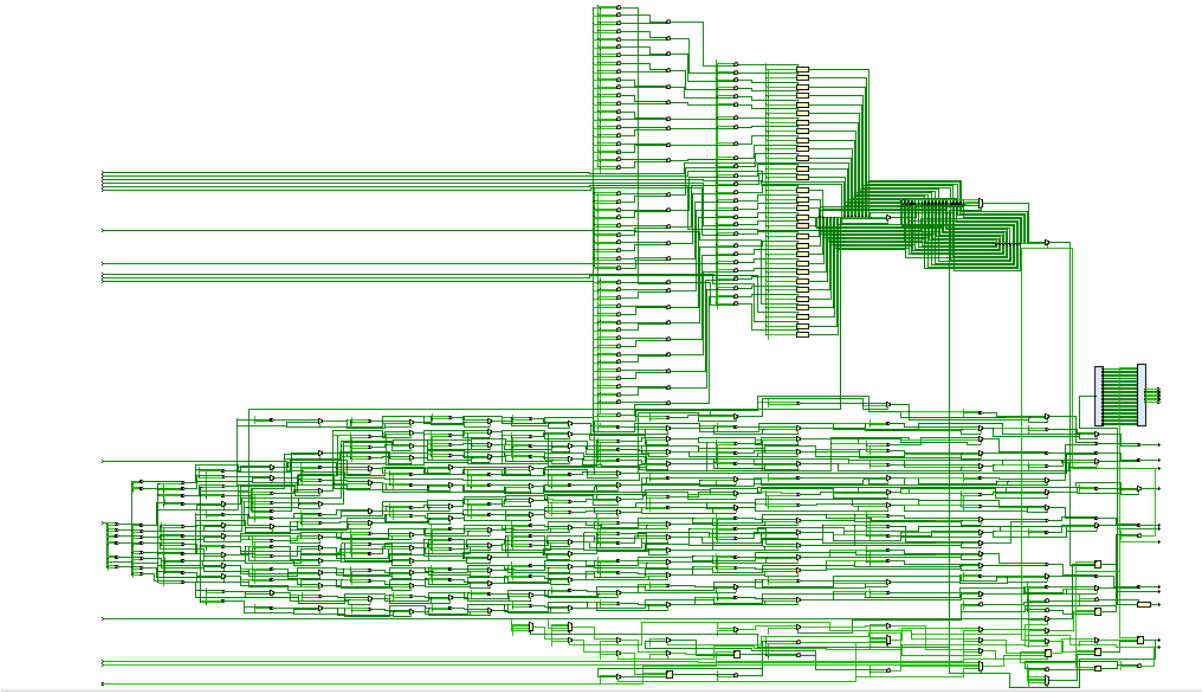
For 256 bits:
We considered 2 blocks of 128 bits, and otherwise implemented the same procedure as for 128 bits.
However, for key register (2559 downto 0), we created 2 repeating blocks i.e. 2 times the size of the previous register.
We defined r_main register (array of length 8 (round) and width 1280). For each operation, it stores at first the input (1279 downto 1024), then the round key xor result (1023 downto 768), the inverse mixed columns operation result (767 downto 512), inverse shift rows operation (511 downto 255) and finally the inverse sub bytes operation (255 downto 0).



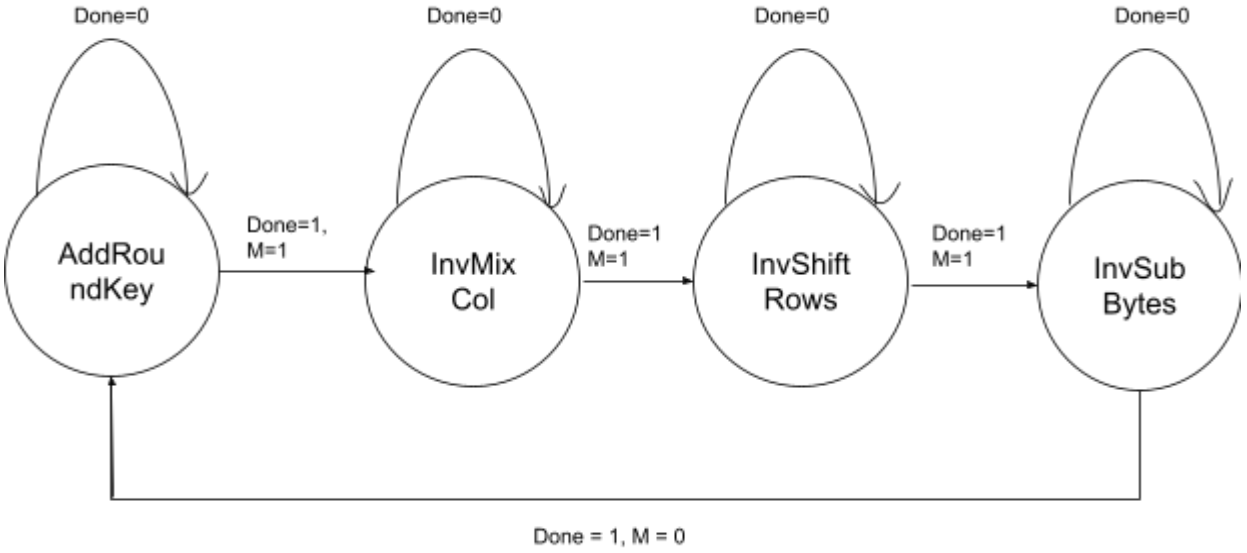
Schematic:







For 8 rounds,
State machine:



We maintained signal cur_state and next_state, wherein cur_state is updated to next state at each clock cycle and when Done=1.

Resource Utilisation:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N decrypt_FSM	3419	1676	304	104	1172	2779	640	1	12	1
DUT (display)	188	64	0	0	68	188	0	0	0	0
DUT1 (hex_to_ascii)	266	112	64	0	100	266	0	0	0	0
inst (bram)	4	0	0	0	2	4	0	0.5	0	0
inst1 (bram_test)	4	0	0	0	2	4	0	0.5	0	0

Slice LUTs

lame	Used
N decrypt_FSM	3419
Leaf Cells (2957)	2957
DUT1 (hex_to_ascii)	266
DUT (display)	188
inst (bram)	4
inst1 (bram_test)	4

Slice Registers

Name	Used
N decrypt_FSM	1676
Leaf Cells (1500)	1500
DUT1 (hex_to_ascii)	112
DUT (display)	64

Primitives

Ref Name	Used	Functional Category
FDRE	1555	Flop & Latch
LUT6	1466	LUT
RAMS32	640	Distributed Memory
LUT2	607	LUT
LUT4	586	LUT
LUT3	493	LUT
MUXF7	304	MuxFx
LUT5	250	LUT
CARRY4	187	CarryLogic
LDCE	112	Flop & Latch
MUXF8	104	MuxFx
LUT1	69	LUT
OBUF	11	IO
FDSE	9	Flop & Latch
RAMB18E1	2	Block Memory
IBUF	1	IO
BUFG	1	Clock