COL215 Hardware Assignment 3 Part 1

Arinjay Singhal 2023CS10041
Vihaan Luhariwala 2023CS10151

September 2024

1 Introduction

The objective of the assignment is: implementation of AES decryption operation. The components involved are memory elements (RAM, ROM and registers) and logical unit.

AES decryption scheme consists of the following steps:

1. AddRoundKey
2. InvShiftRows
3. InvSubBytes
4. InvMixColumns

We have implemented these modules along with RAM and ROM modules.

2 Design

2.1 AddRoundKey

2.1.1 Logic

The AddRoundKey module takes a 8 bit input along with a single element of round key (8 bit) and returns the XOR of the two.

2.1.2 Simulations

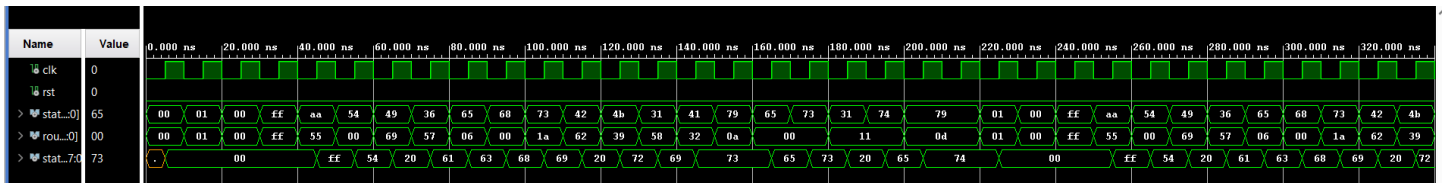

Figure 1: AddRoundKey Simulation

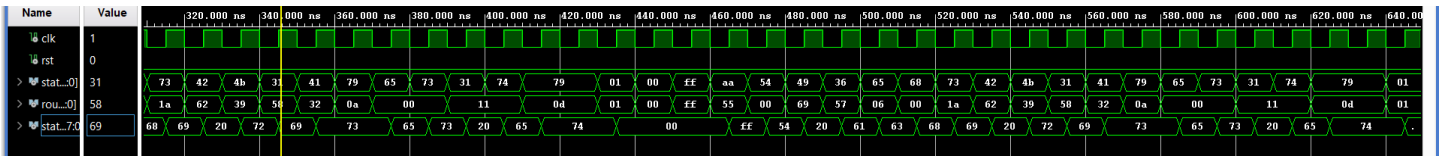

Figure 2: AddRoundKey Simulation

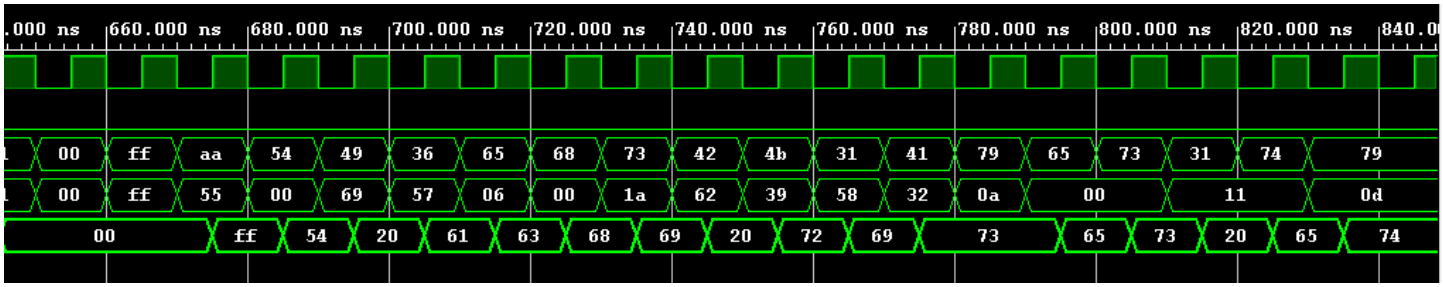

Figure 3: AddRoundKey Simulation

2.1.3 Block Diagram

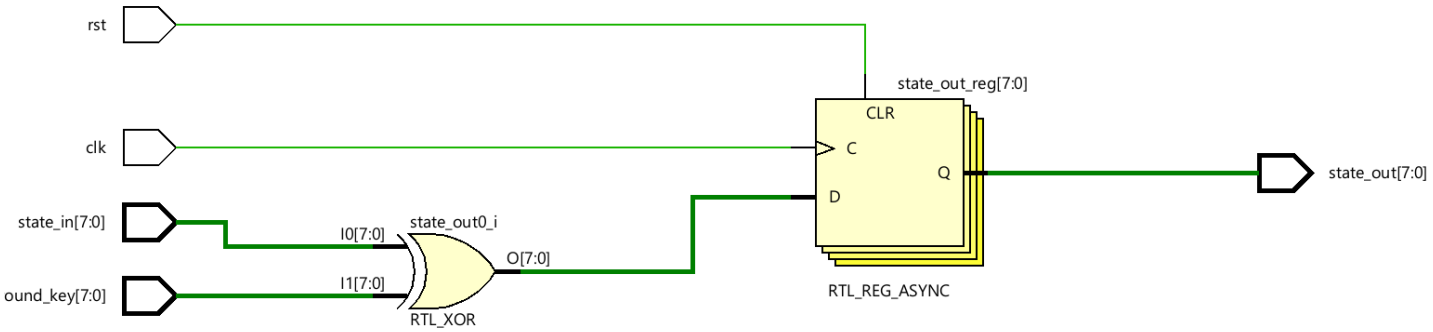

Figure 4: AddRoundKey Block Diagram

2.2 InvShiftRows

2.2.1 Logic

The InvShiftRows module takes a 32 bit input (which represents a row in the state matrix), along with the number of shifts to be performed on the row. It returns the row after the shifts. We break the 32 bit input into 4 bytes and then construct the output by shifting the bytes by the required amount.

2.2.2 Simulations

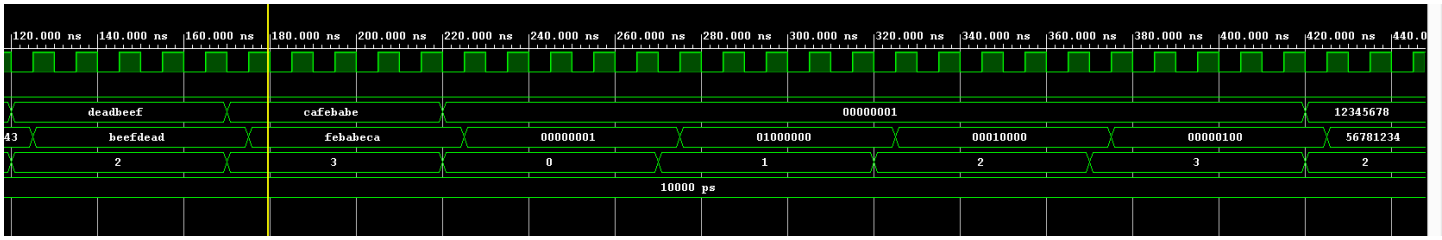

Figure 5: InvShiftRows Simulation

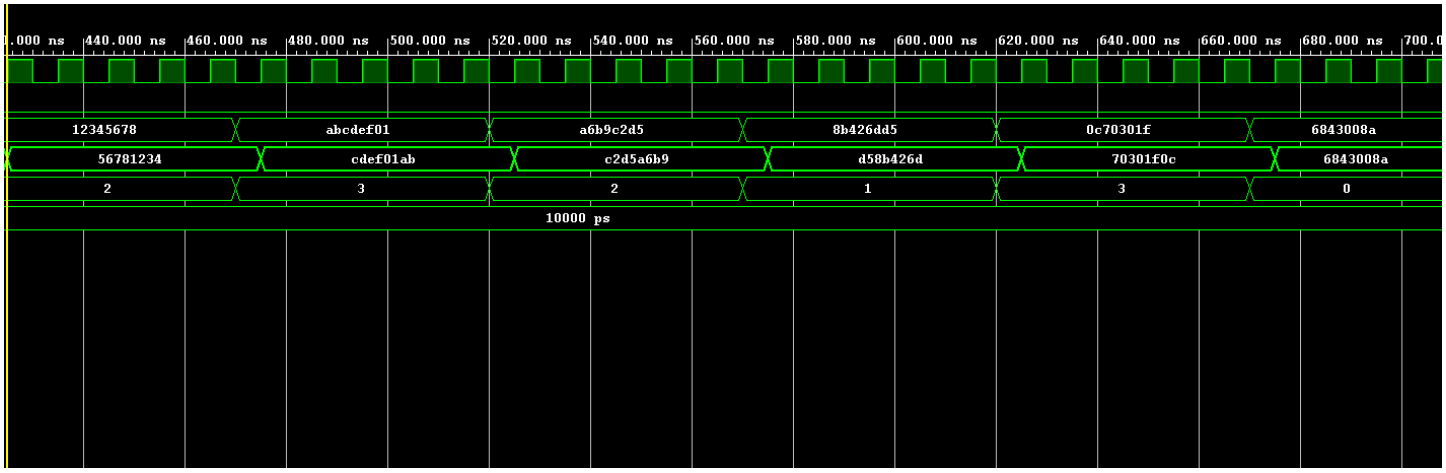

Figure 6: InvShiftRows Simulation

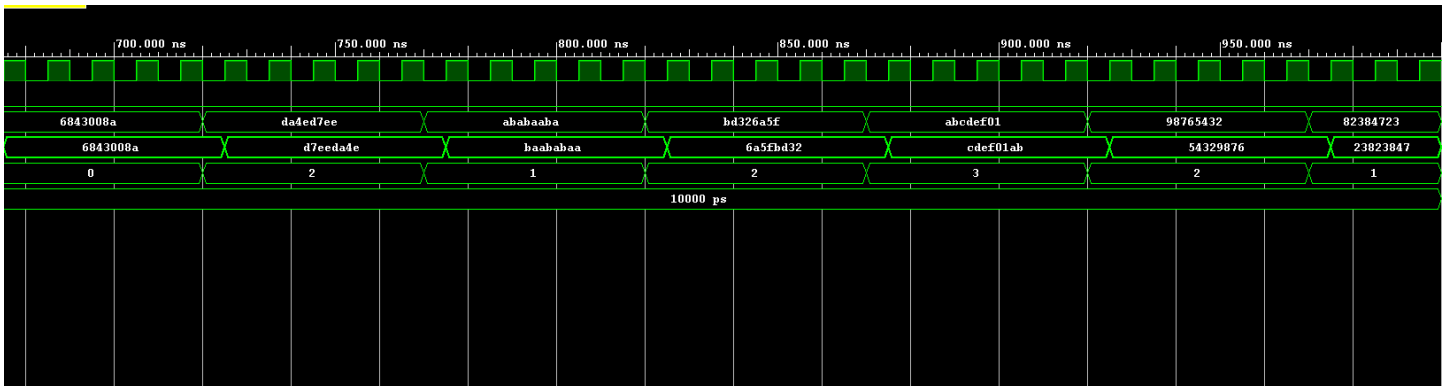

Figure 7: InvShiftRows Simulation

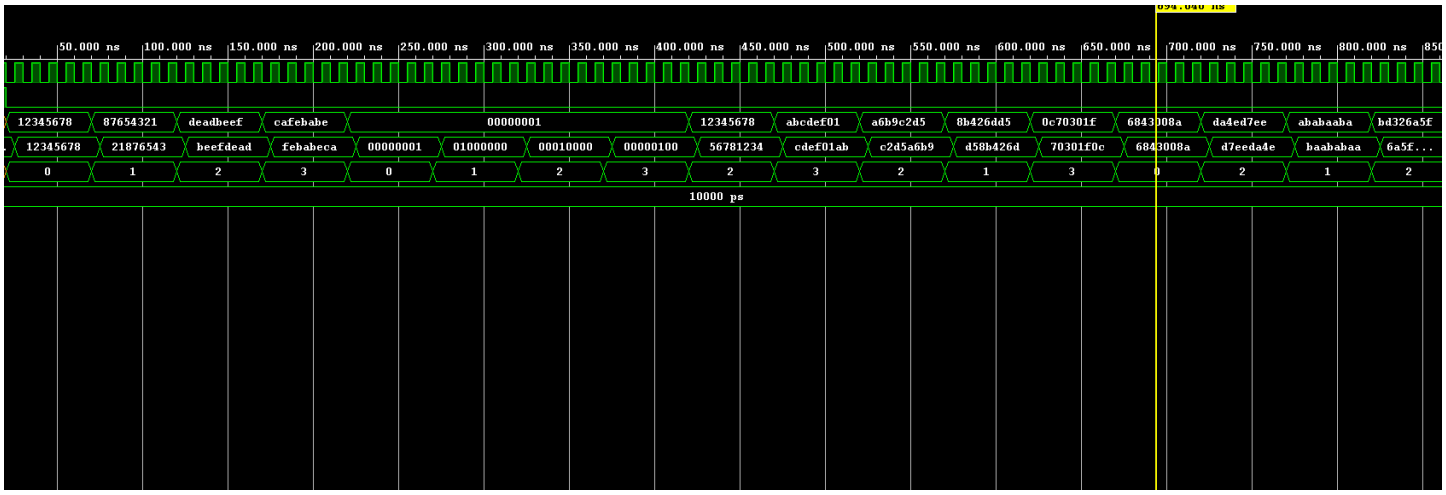

Figure 8: InvShiftRows Simulation

2.2.3 Block Diagram

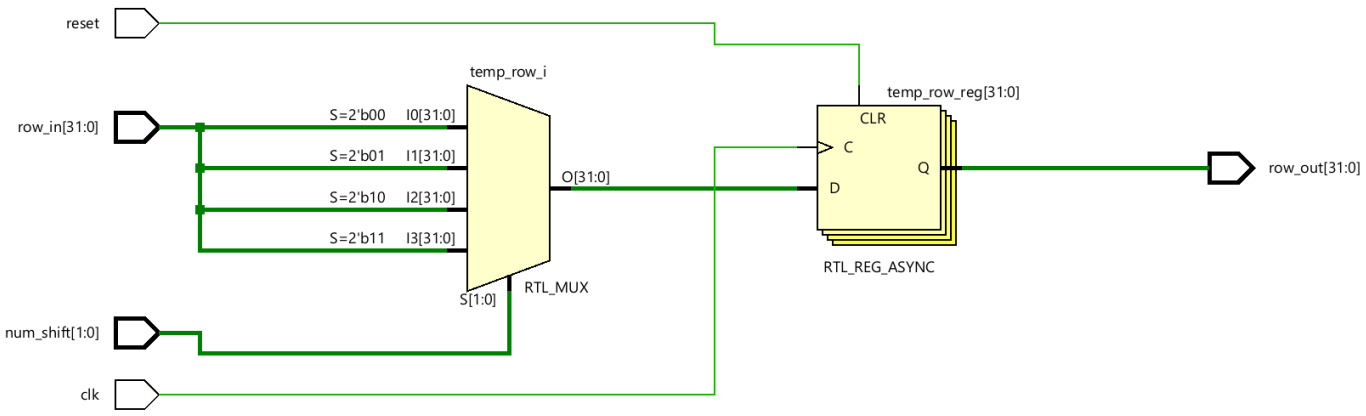

Figure 9: InvShiftRows Block Diagram

2.3 InvSubBytes

2.3.1 Logic

The InvSubBytes module takes a 8 bit input and returns the corresponding value from the inverse sbox. The sbox is stored in a ROM (sbox_inv). We find the index of the input in the sbox from the given input by 16*(column number) + (row number). The ROM is given this index and it returns the corresponding value.

2.3.2 Simulations

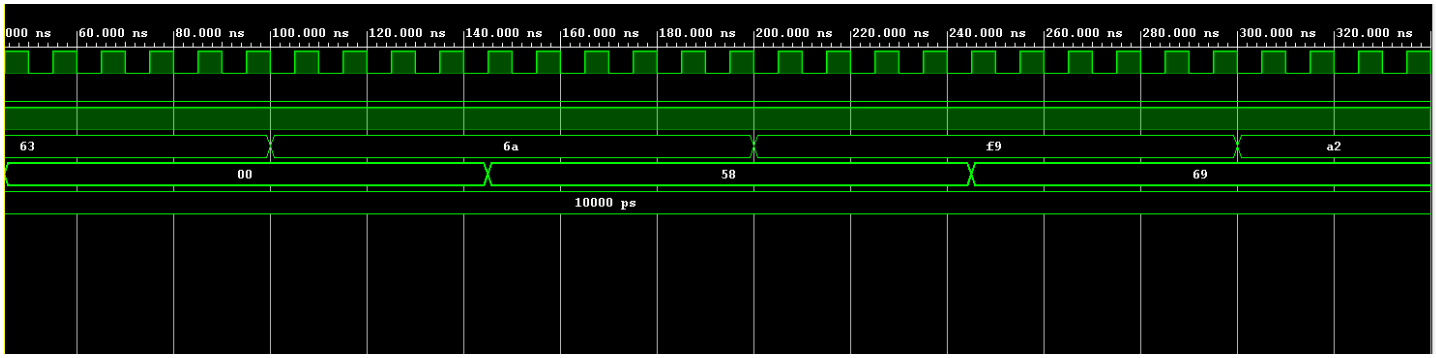

Figure 10: InvSubBytes Simulation

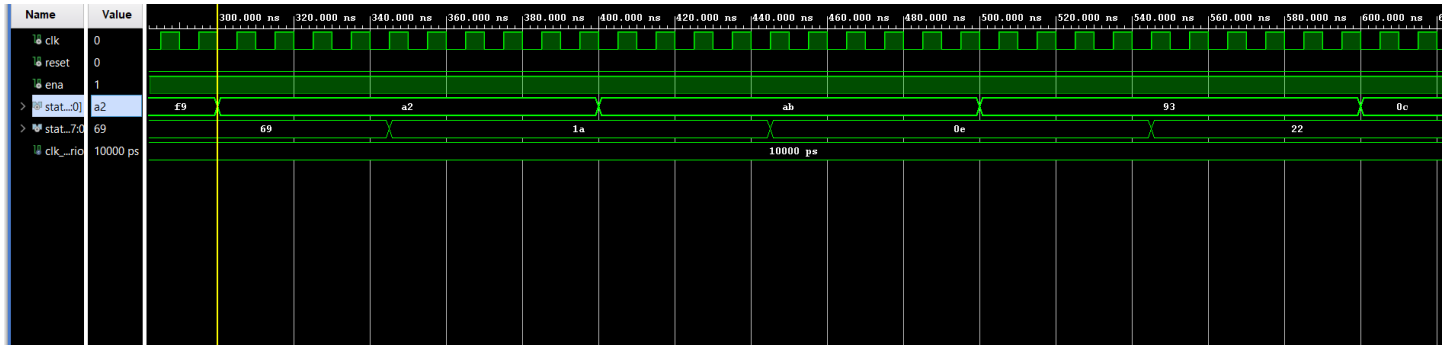

Figure 11: InvSubBytes Simulation

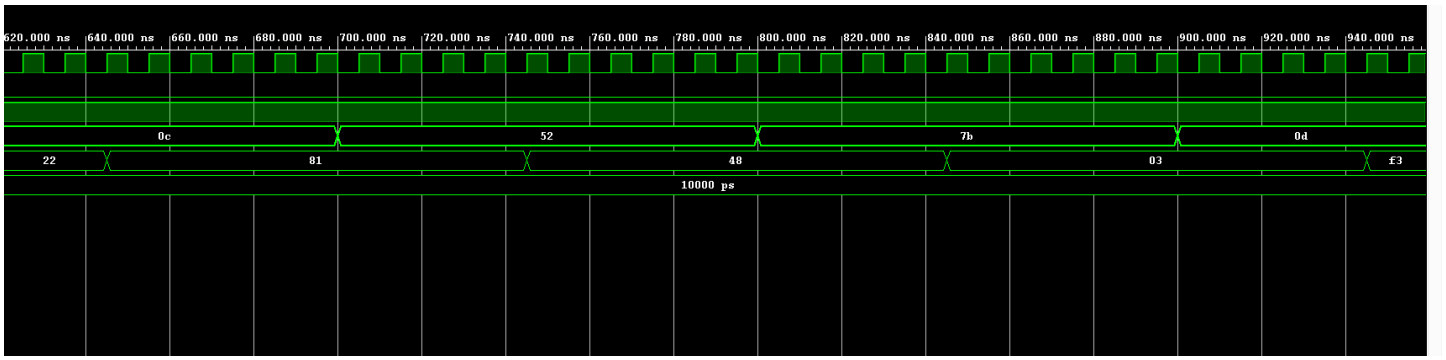

Figure 12: InvSubBytes Simulation

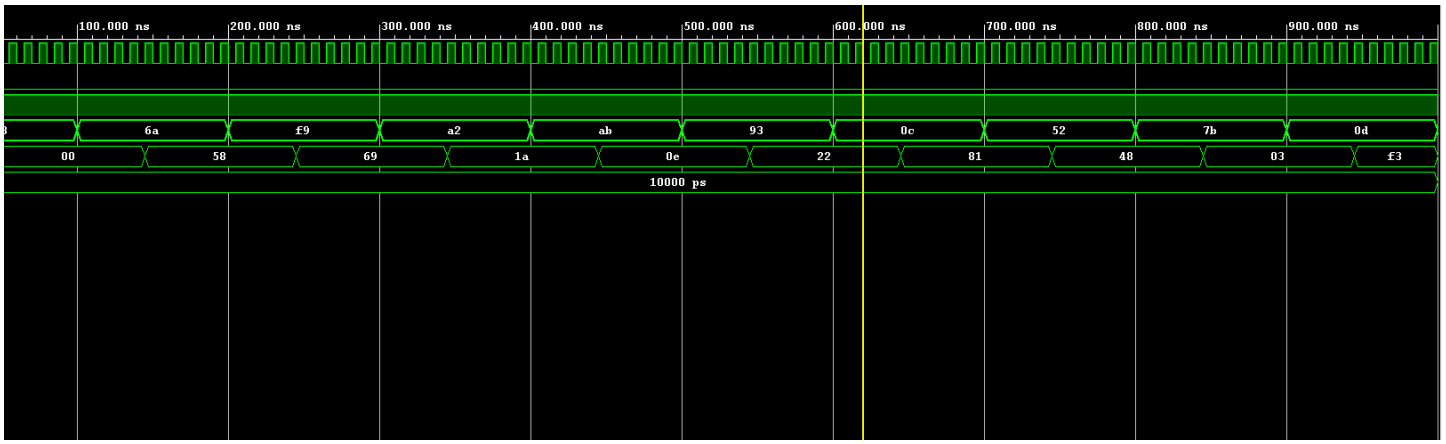

Figure 13: InvSubBytes Simulation

2.3.3 Block Diagram

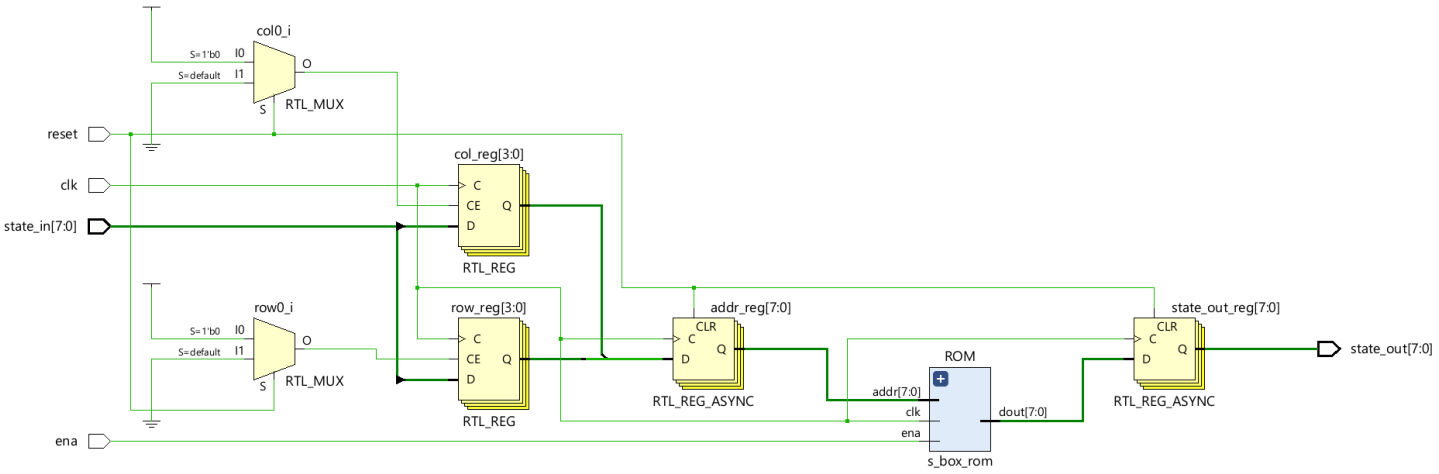

Figure 14: InvSubBytes Block Diagram

2.4 InvMixColumns

2.4.1 Logic

The InvMixColumns module takes a 32 bit input (which represents a column in the state matrix) and multiplies it with the inverse mix column matrix in GF(2^8). The output is a 32 bit column. We first implement the multiplication by 2 in GF(2^8). This is used to calculate the multiplication by 4 and 8. Multiplication by 9, b, d, e is done by adding the input to the output of the multiplication by 8, 2, 4, 1 respectively as given in handout. The resulting values are then XORed to get the final output for each cell of the column.

2.4.2 Simulations

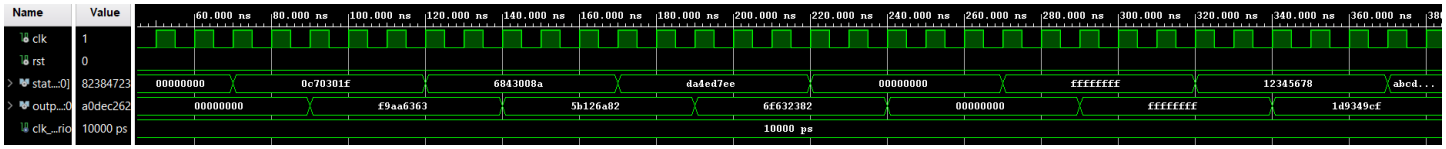

Figure 15: InvMixColumns Simulation

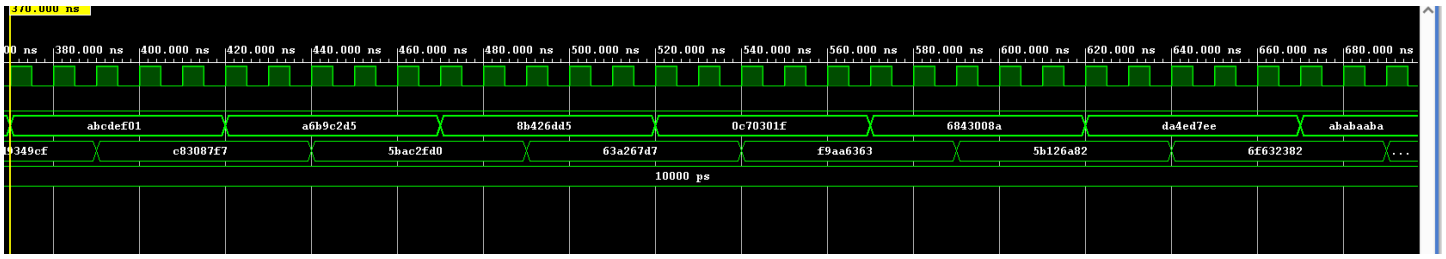

Figure 16: InvMixColumns Simulation

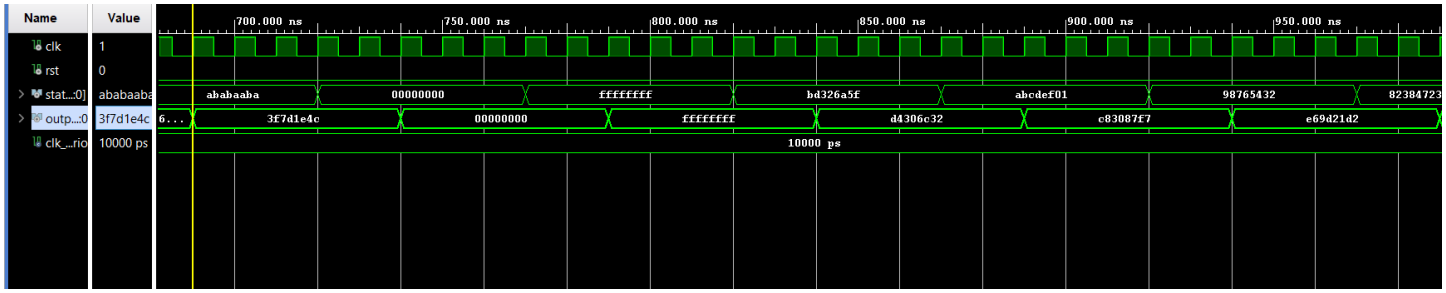

Figure 17: InvMixColumns Simulation

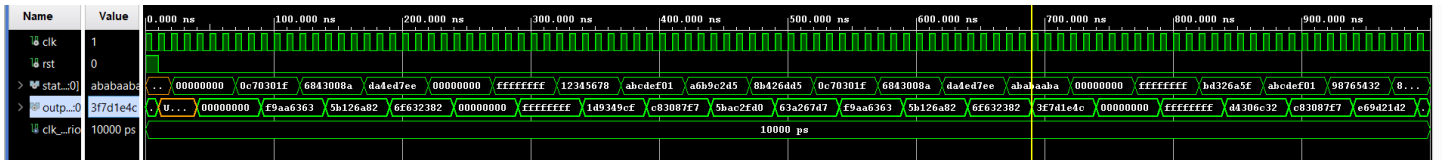

Figure 18: InvMixColumns Simulation

2.4.3 Block Diagram

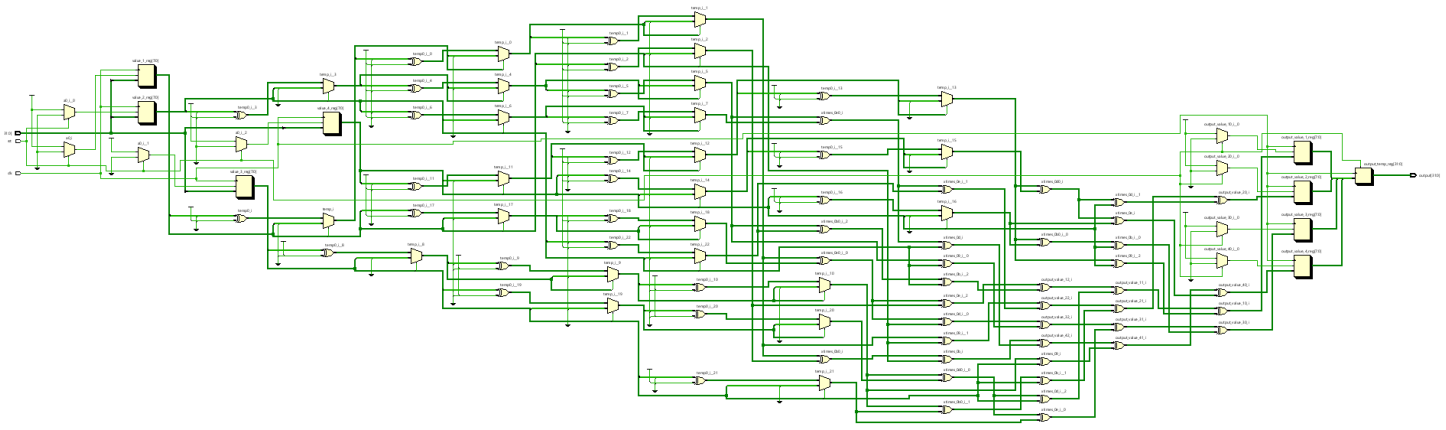

Figure 19: InvMixColumns Block Diagram

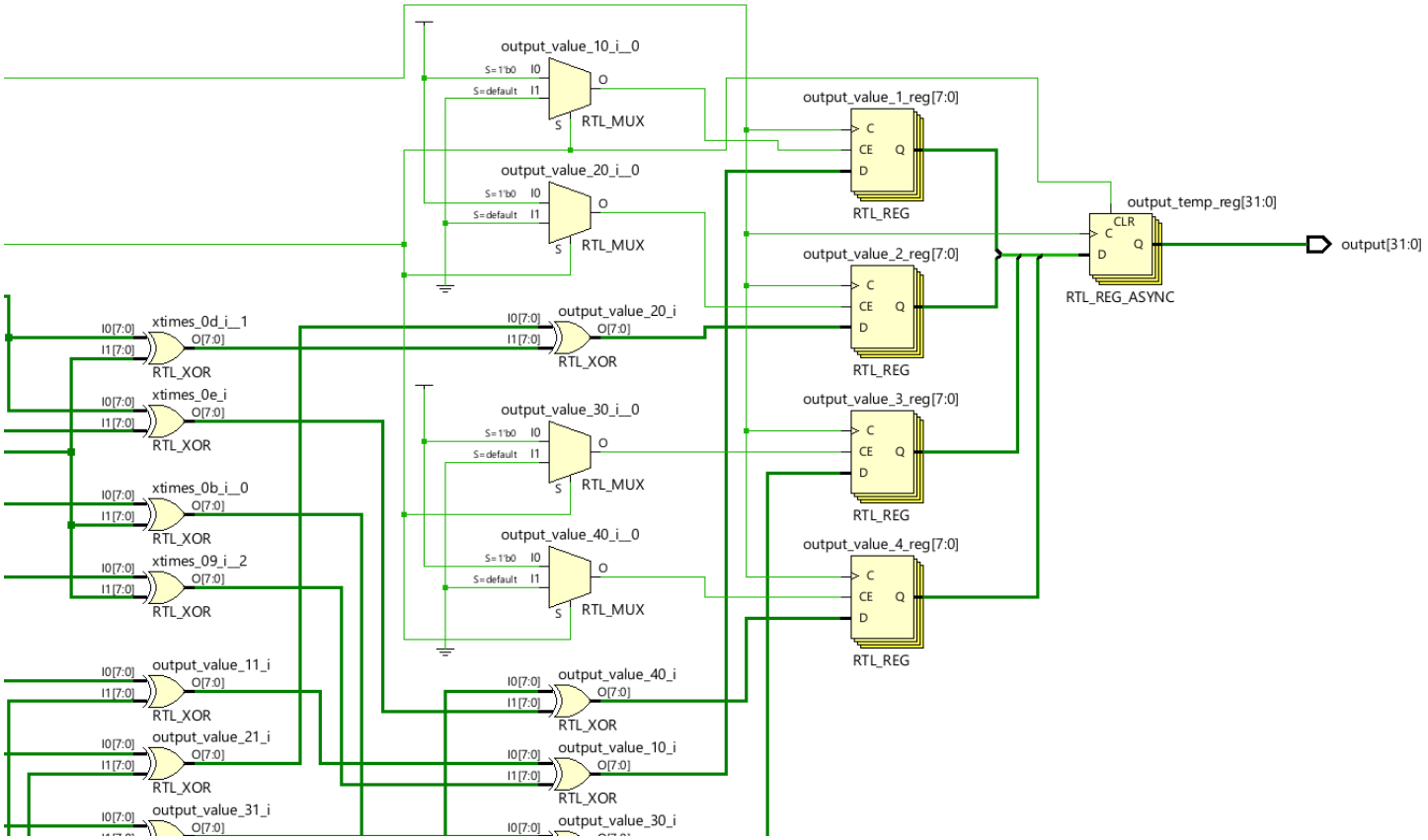

Figure 20: InvMixColumns Block Diagram

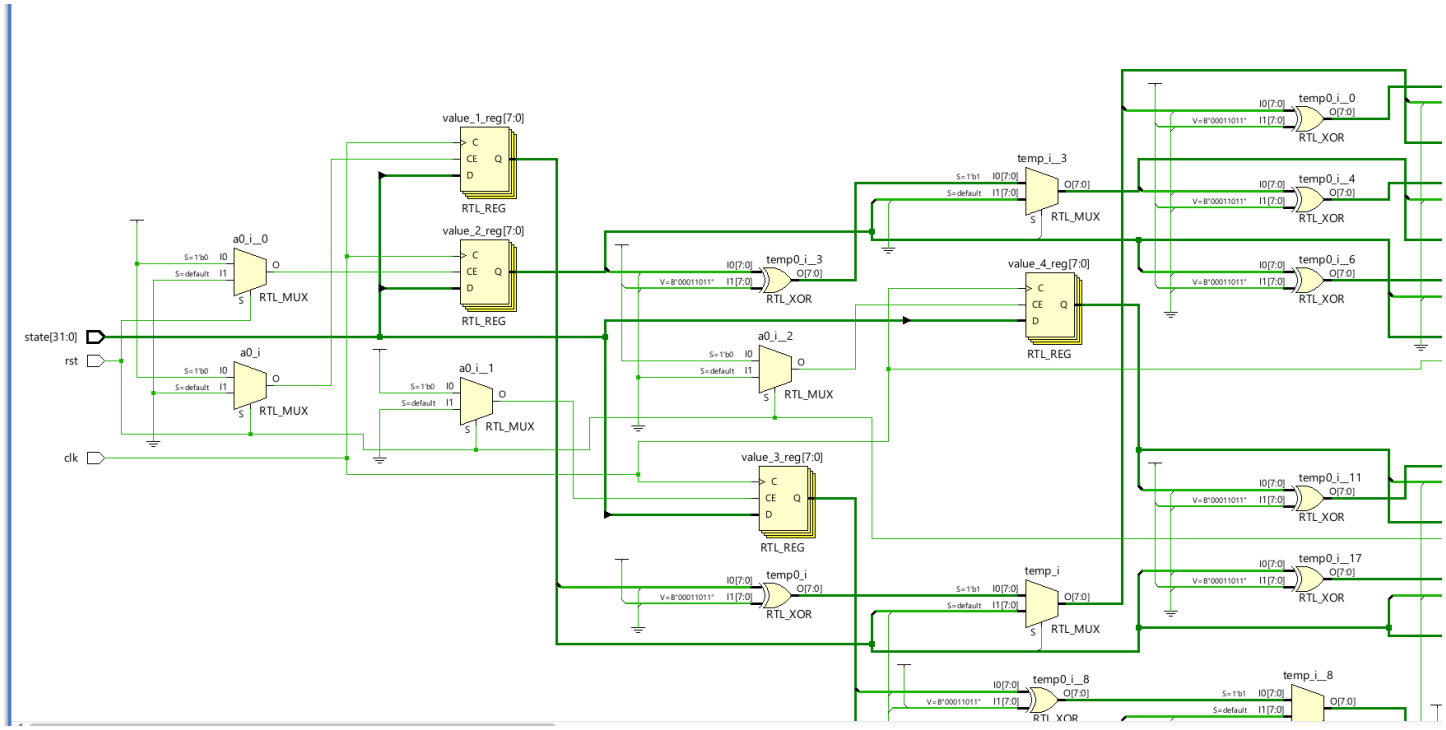

Figure 21: InvMixColumns Block Diagram

2.5 Display

2.5.1 Logic

The Display module takes a 32 bit input (1 row of the state matrix) and displays it in hexadecimal format. We first break the 32 bit input into 4 bytes and then convert the bytes to their ascii values. We then set the cathode pins of the 7 segment display to the corresponding ascii values. If the ascii value lies out of range, we display a dash. As the cathode pins are common to all the 7 segment displays, we have to display the values one by one in a cycle. We use a counter to keep track of the current display and update the counter based on clk signal. The anode pins are also updated based on the counter value.

2.5.2 Simulations

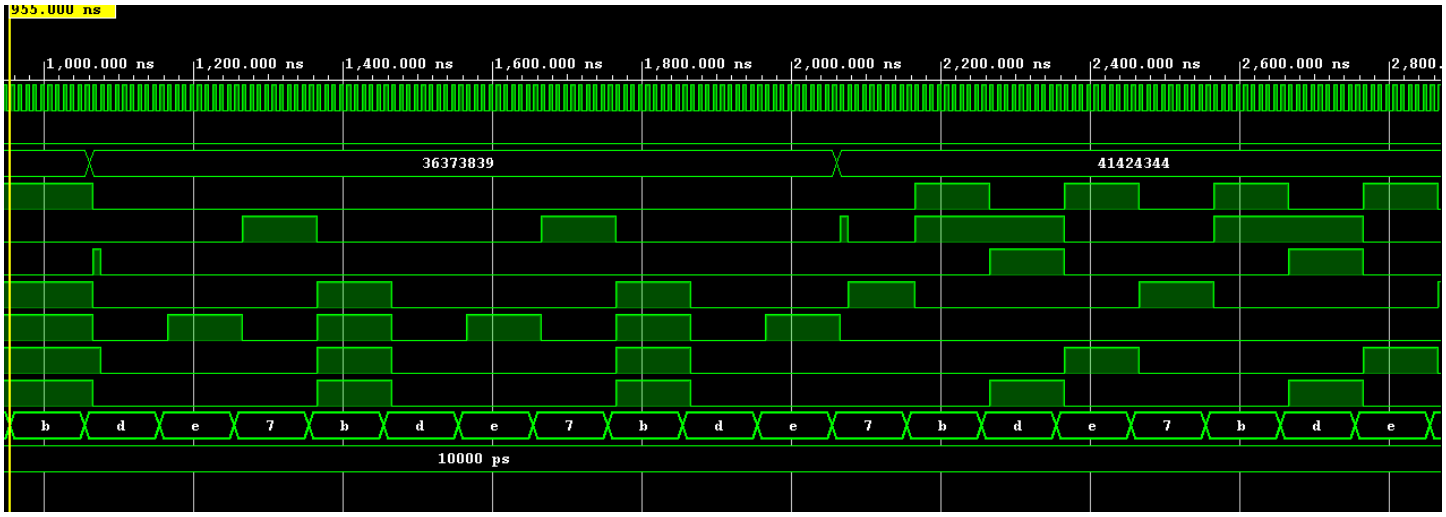

Figure 22: Display Simulation

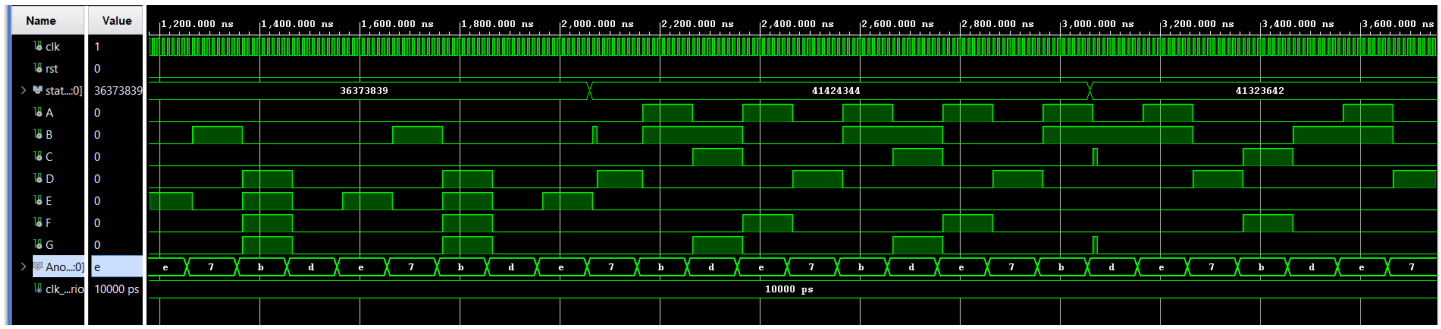

Figure 23: Display Simulation

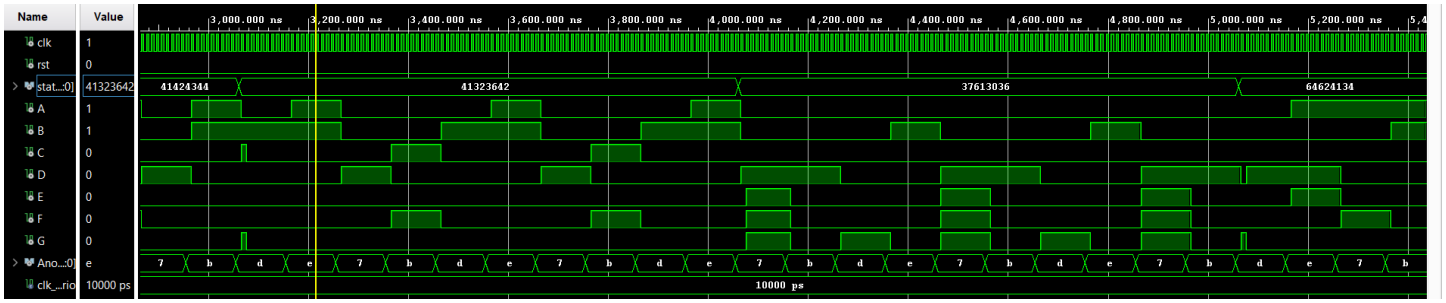

Figure 24: Display Simulation

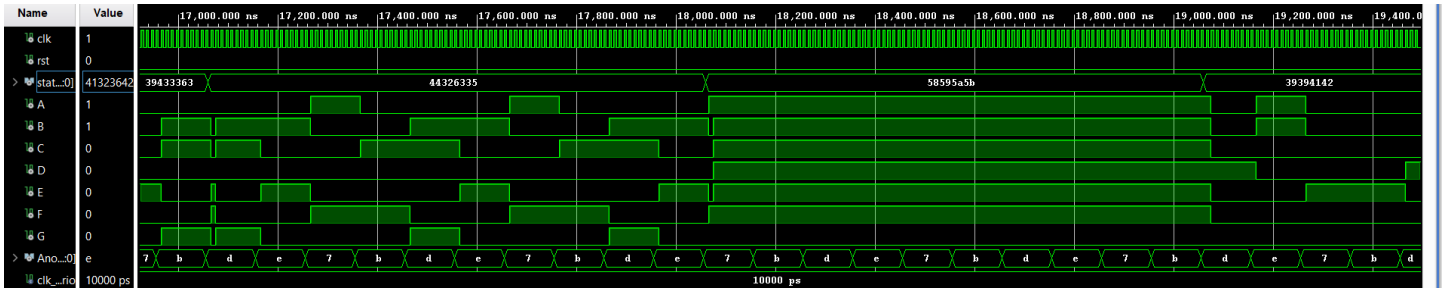

Figure 25: Display Simulation

2.5.3 Block Diagram

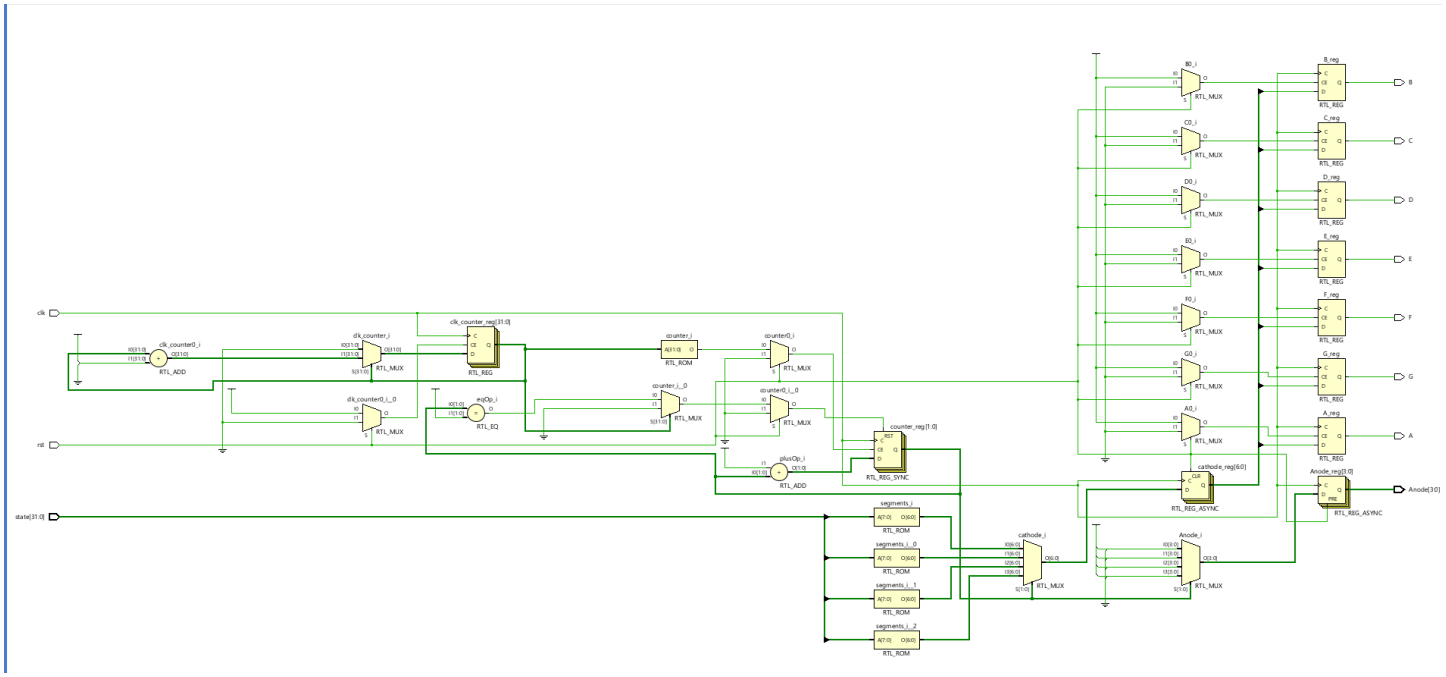

Figure 26: Display Block Diagram

2.6 RAM

2.6.1 Logic

The RAM is implemented using the IP catalog. The depth is set to 16 and the width is set to 16. It stores the state matrix.

2.6.2 Simulations

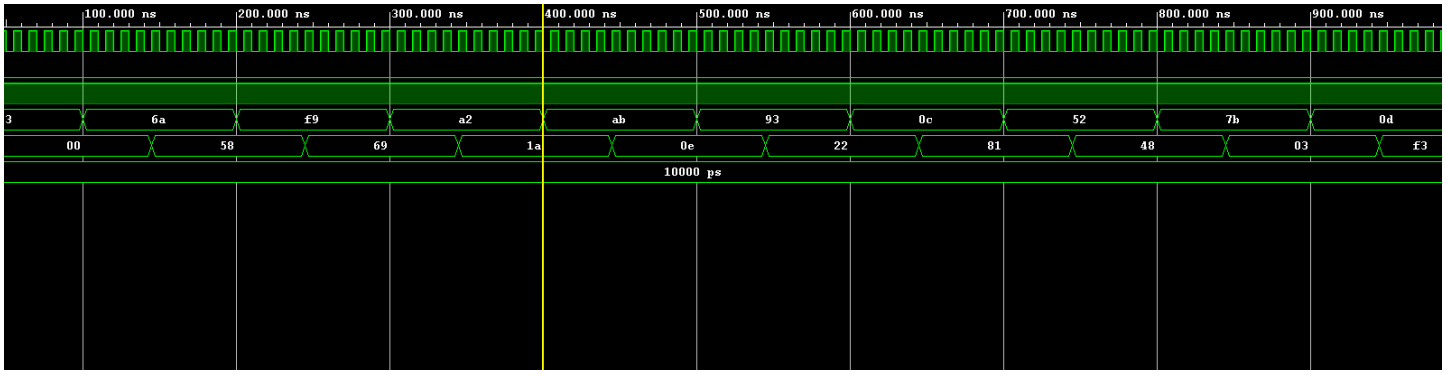

Figure 27: RAM Simulation

2.6.3 Block Diagram

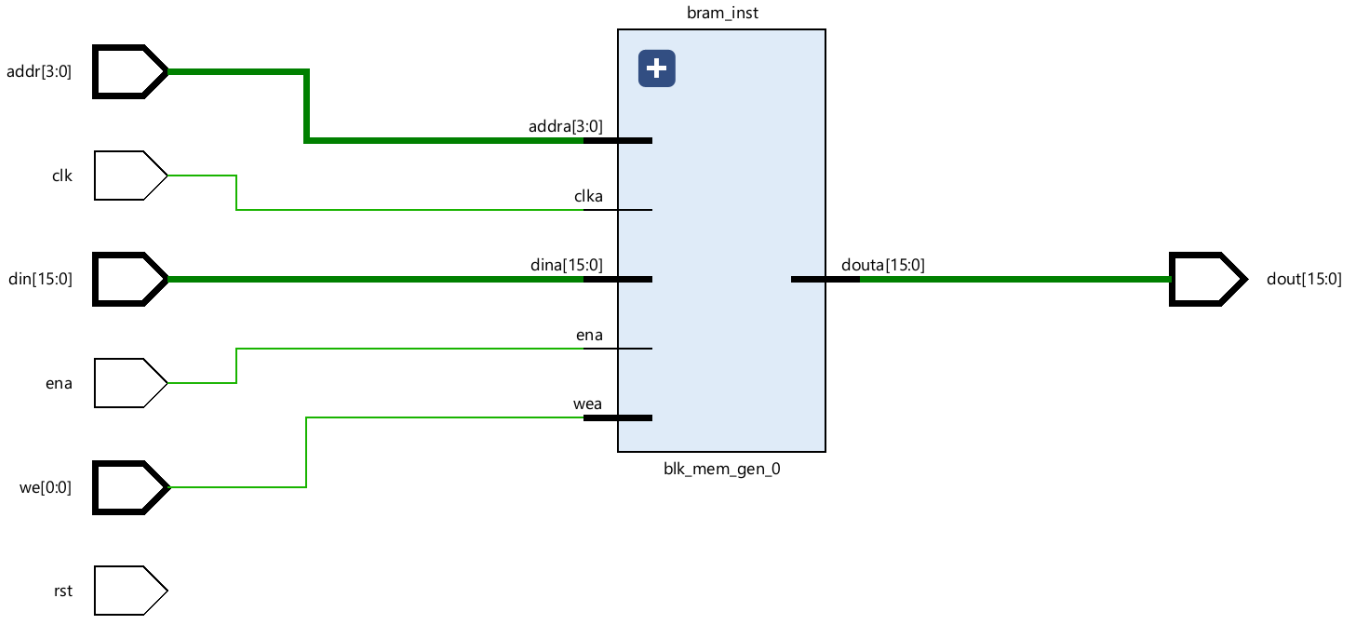

Figure 28: RAM Block Diagram

2.7 ROM

2.7.1 Logic

We have implemented sbox_inv_rom which is initialized using IP catalog and sbox_inv.coe file. The ROM is used to store the inverse sbox.

2.7.2 Simulations

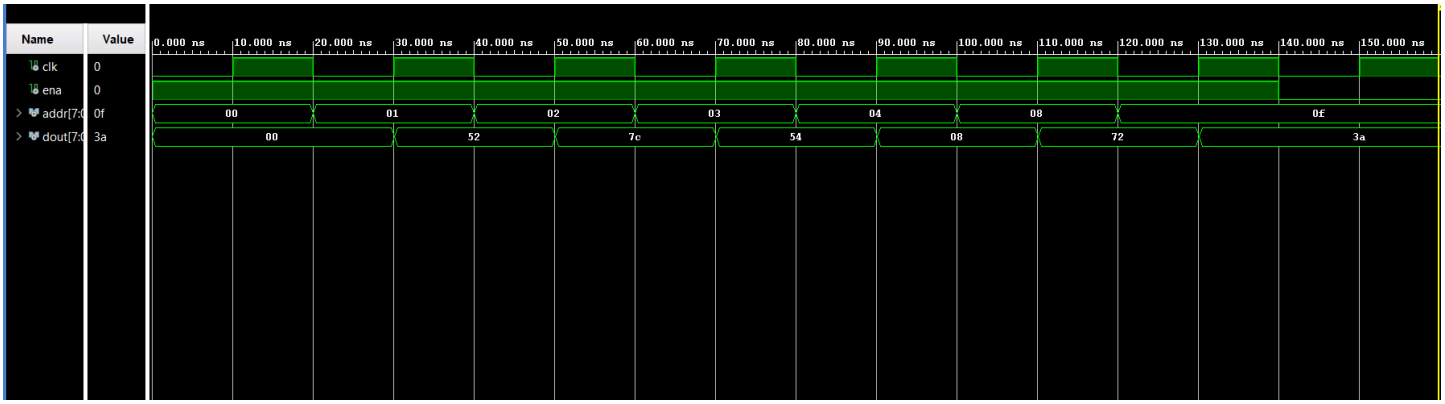

Figure 29: ROM Simulation

2.7.3 Block Diagram

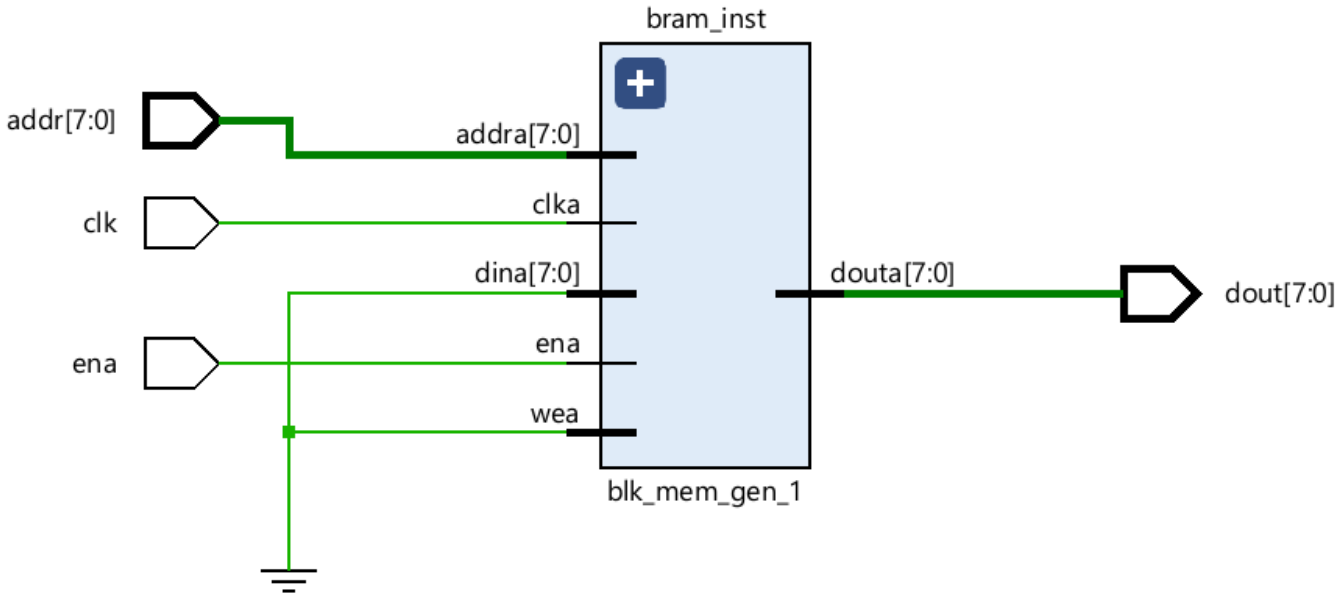

Figure 30: ROM Block Diagram

3 Note

All out modules are sensitive to the clock as they take cllk signal in the sensitivity list. This results in a time lag between the input and output. This is because the output is updated only when the clock signal changes. Typically, this lag is around 2 clock cycles.

4 Conclusion

We have implemented the AES decryption scheme using the modules AddRoundKey, InvShiftRows, InvSubBytes, InvMixColumns, RAM and ROM. The modules have been tested and verified using simulations.