COL-215
Hardware Assignment - 3 Report

Shreeraj Jambhale 2023CS50048
Pallav Kamad 2023CS51067

Table Of Contents

1 Build Seven Segment Decoder

1.1 Aim

The aim of this assignment is to implement the AES decryption operation, focusing on designing memory components (RAM and ROM) and logical units required for decryption. The goal is to decrypt given ciphertext into plaintext and display it on a seven-segment display on the Basys 3 board, ensuring accurate functionality and adherence to the AES algorithm's decryption process. This part only focuses on implementing smaller module which will eventually be used in Part-II

1.2 Approach

The Approach is written in parts like flow of control

1.2.1 Round Key – XOR

- Input Setup: Take an 8-bit input from the AES state matrix (1 hexadecimal code).
- Round Key: Input the corresponding 8-bit round key for the current decryption round.
- XOR Operation: Perform a bitwise XOR operation between the 8-bit input and the 8-bit round key.
- Output: The result of the XOR operation forms part of the decrypted output for this step.

1.2.2 Inverse Shift Rows - InvShiftRows

The InvShiftRows transformation is the inverse of the Shift Rows step in AES encryption. In InvShiftRows, the bytes in each row of the state are shifted to the right by different offsets, in our implementation we provide one row at a time to the module and also provide the row number so to inform about the shift number

- Input: row_in (32 bits) representing the current row.
- Shift index: shift_idx (2 bits) to specify the shift amount.
- Output: row_out (32 bits) as the shifted row.

Used if-else blocks to execute the process

- If shift_idx is "00", no shift is applied.
- If shift_idx is "01", it shifts by 8 bits (1 byte).
- If shift_idx is "10", it shifts by 16 bits (2 bytes).
- If shift_idx is "11", it shifts by 24 bits (3 bytes).

1.2.3 Inverse Sub Bytes – Inv S-Box

This code implements an inverse S-Box lookup function in AES decryption using a single-port ROM. Here's a breakdown of the approach and how it works:

- Input Splitting: The 8-bit input is split into two 4-bit parts, dig1 and dig2. This is done to access the specific row and column of the S-Box.
- Address Formation: The dig2 (lower 4 bits) and dig1 (higher 4 bits) are concatenated to form an 8-bit intermediate_add, which serves as the address for the ROM.
- ROM Lookup: The rom_invSbox component is instantiated as a single-port ROM, using intermediate_add as the address input (addra). This ROM is generated using a Block Memory Generator, preloaded with the inverse S-Box values.
- Output: The douta from the ROM outputs the S-Box-transformed value for the given input.

1.2.4 Inv Mix Columns

1.2.4.1 GF8_2Mul

The GF8_2Mul function multiplies two 8-bit vector inputs within the Galois Field returning an 8-bit result. The function achieves this by applying bitwise shifts and XOR operations to perform the multiplication and reduction steps.

1.2.4.2 Inv Mix Col

The Inv Mix Col function leverages the GF8_2Mul function to compute the Galois Field sum of four separate products. It combines the outputs from four calls to GF8_2Mul to produce a final 8-bit vector output, which replaces a single hexadecimal value in the 4x4 matrix during the transformation process.

1.2.5 Display

This VHDL project converts a 32-bit hexadecimal input into ASCII characters and scrolls the result across four 7-segment displays on the Basys 3 board, showing four characters at a time with a 1-second interval. The purpose is to enable full data visibility by cycling through the ASCII-converted data continuously. Key steps include:

- Clock Divider: Generates a 1-second delay for smooth scrolling.
- Scrolling Index: Selects and updates the displayed 4-character segment every second.
- ASCII Conversion: Maps each 4-bit hexadecimal digit to its ASCII equivalent for display.
- Display Logic: Cycles through the ASCII characters, presenting them on the 7-segment displays.

This design enables scrolling for large ASCII data on limited hardware.

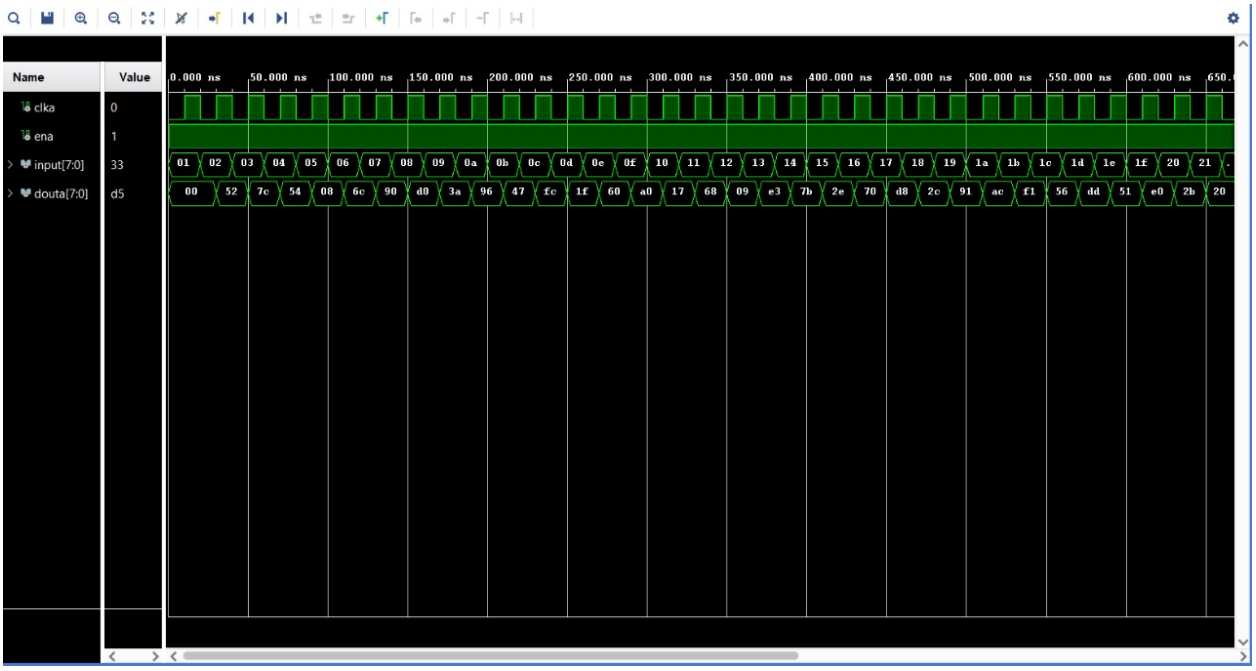
Digit	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
A/a	1	1	1	0	1	1	1
B/b	0	0	1	1	1	1	1
C/c	1	0	0	1	1	1	0
D/d	0	1	1	1	1	0	1
E/e	1	0	0	1	1	1	1
F/f	1	0	0	0	1	1	1
-	1	1	1	1	1	1	0

1.3 Simulation

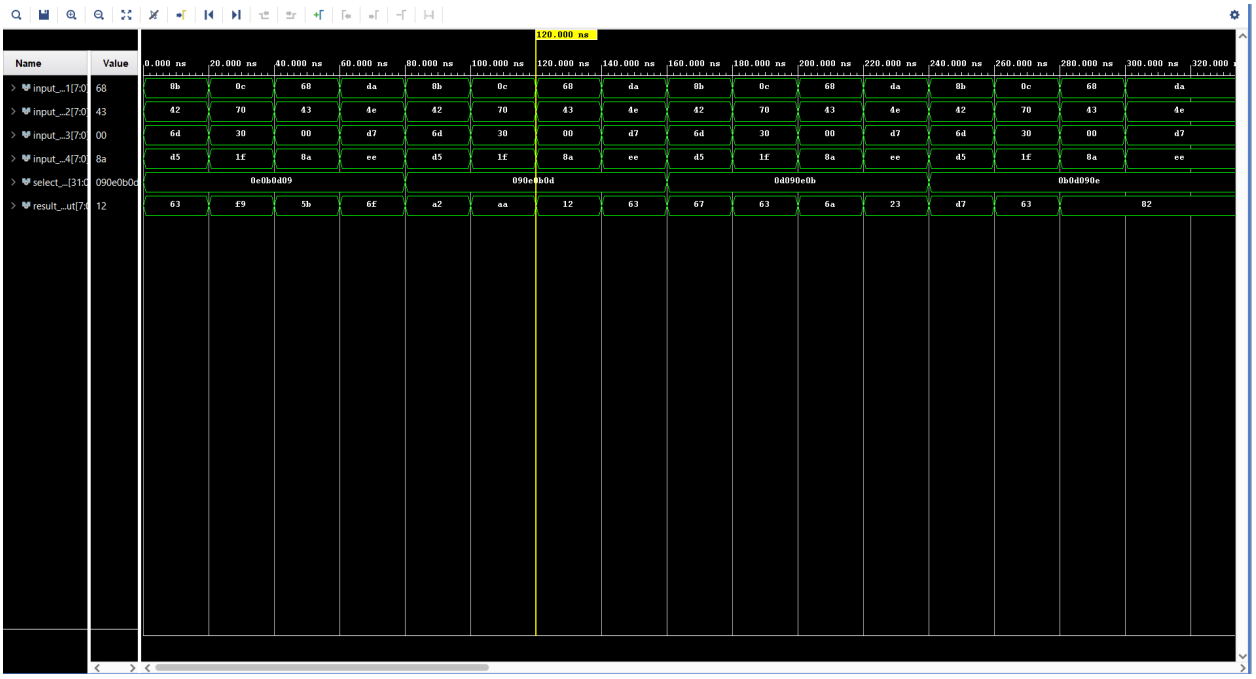
1.3.1 Round Key (XOR)

1.3.2 InvShiftRows

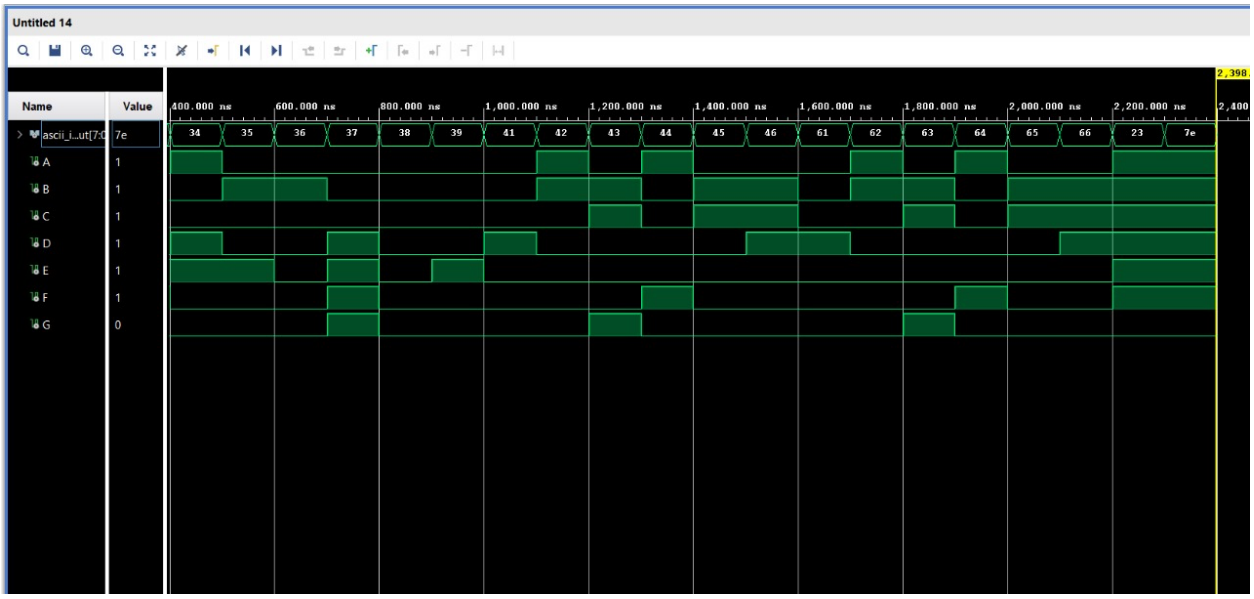
1.3.3 Inv S-Box (outputs are shifted by some Clk cycle because of ROM access delay)



1.3.4 Inv Mix Columns

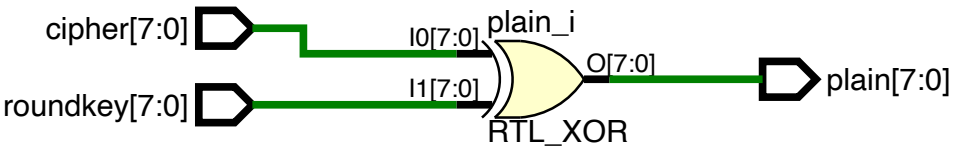


1.3.5 Display

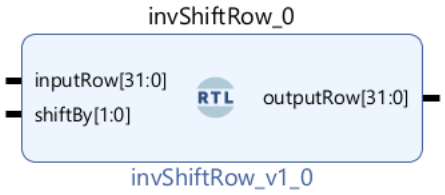


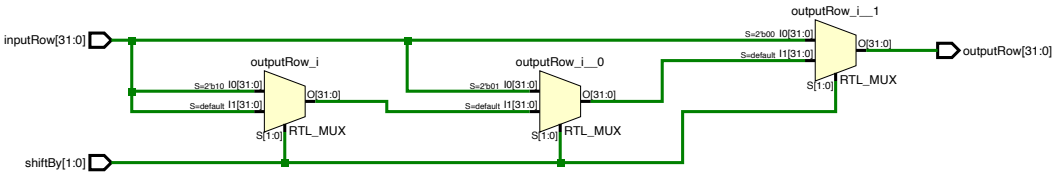
1.4 Block Diagrams

1.4.1 Round Key (XOR)

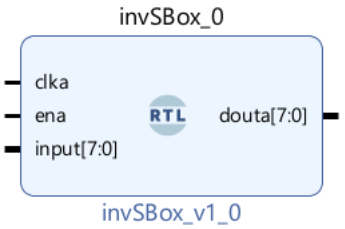


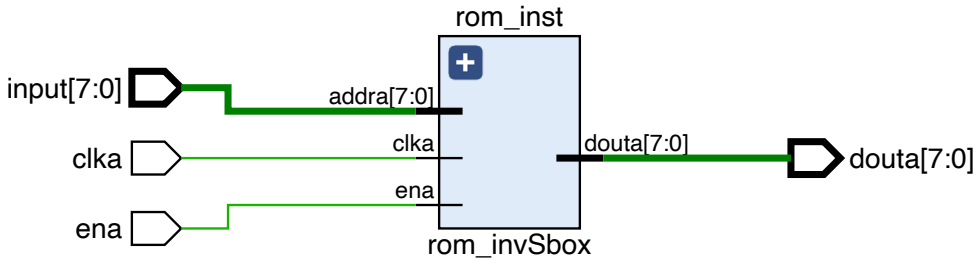
1.4.2 InvShiftRows



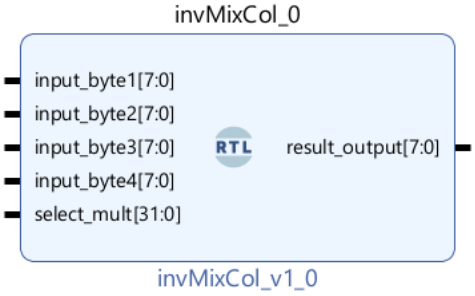


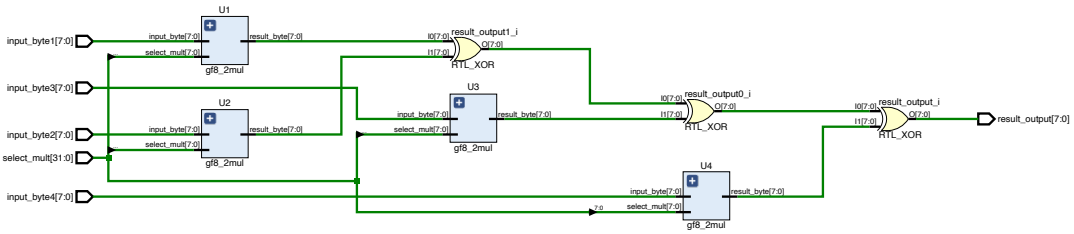
1.4.3 Inv S-Box





1.4.4 Inv Mix Columns





1.4.5 Display