Indian Institute of Technology (IIT-Kharagpur)

AUTUMN Semester, 2025

COMPUTER SCIENCE AND ENGINEERING

Computer Organization and Architecture Laboratory

Verilog Assignment 6

August 26, 2025

Instructions: *Please make an individual submission of this assignment. You must demonstrate your working design to a TA for evaluation. Submit a single zipped folder named* `VerilogAssignment6_<Your_Roll_No>.zip` *containing all well-commented source files. Ensure each source file includes a header with your name and roll number.*

Overall Cipher Architecture

The Simplified Advanced Encryption Standard (S-AES) is a pedagogical version of the real AES cipher. It operates on a smaller 16-bit block size and uses a 16-bit key, making it ideal for understanding the principles of a modern Substitution-Permutation Network (SPN) cipher. In this assignment, you will build a hardware implementation of the S-AES encryption process, focusing on the efficient use of FPGA resources by storing lookup tables in memory.

Encryption Flow:

1. **Key Expansion:** The Master Key is used to generate three Round Keys (K_0, K_1, K_2).
2. **Pre-Round:** The Plaintext is XORed with Round Key K_0.
3. **Round 1:** The state undergoes four transformations: `NibbleSub`, `ShiftRows`, `MixColumns`, and `AddRoundKey` (using K_1).
4. **Round 2 (Final Round):** The state undergoes three transformations: `NibbleSub`, `ShiftRows`, and `AddRoundKey` (using K_2). The `MixColumns` step is skipped. The result is the Ciphertext.

Detailed Component Design and Implementation

You will create a separate Verilog module for each component. The following sections describe their function and a table-based guide for implementation.

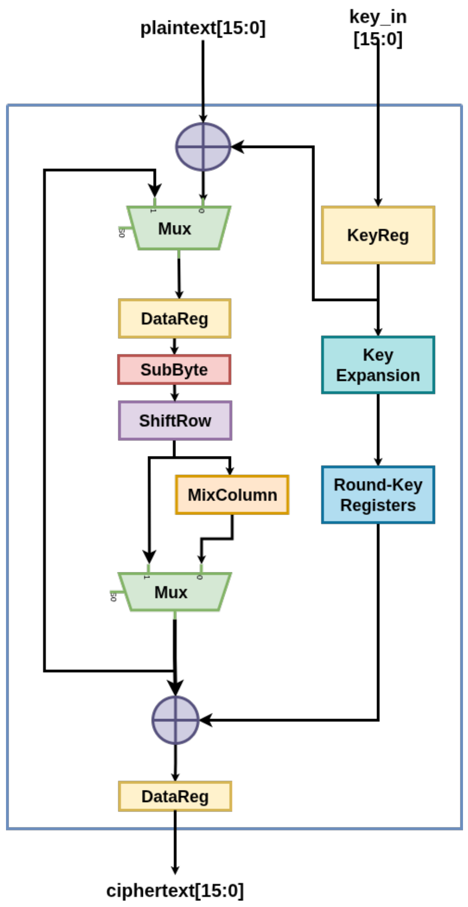

Figure 1: The overall data flow of the S-AES encryption process.

S-Box (Nibble Substitution)

- **Purpose:** To provide non-linear substitution, which is the core of the cipher's security.
- **Table-Based Implementation:** The S-Box is a fixed lookup table. For any 4-bit input nibble, it outputs a corresponding 4-bit nibble. In hardware, this is a **16x4-bit Read-Only Memory (ROM)**.
- **Note on State Processing:** Since the state is 16 bits (four 4-bit nibbles), the full `NibbleSub` operation requires four parallel S-Box lookups to transform the entire state in one clock cycle. Your top-level S-Box module should manage these four instances.

Example:

- **Input Nibble:** `B` (binary `1011`)
- **Process:** Use the input `B` (decimal 11) as the address for the table.

Table 1: S-Box Lookup Table

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	9	4	A	B	D	1	8	5	6	2	0	3	C	E	F	7

- **Output:** The value at address `B` is `3`. So, $S\text{-}Box(B) = 3$.

Hardware Task: Implement this table in a **BRAM** resource. Create a `.coe` file with the 16 output values and use the Vivado IP Generator to create a Block ROM.

Shift Rows

- **Purpose:** To provide diffusion by transposing the data across columns.
- **Table-Based Implementation:** This operation can be described as a fixed permutation of the nibble positions in the 2x2 state matrix.

Table 2: Nibble Position Permutation

Initial Position	N_{00} N_{01} N_{10} N_{11}
Final Position	N_{00} N_{01} N_{11} N_{10}

Example:

- **Input State:** `E793`, Matrix: $\begin{pmatrix} \texttt{E} & \texttt{7} \\ \texttt{9} & \texttt{3} \end{pmatrix}$
- **Process:** The top row (`E, 7`) is unchanged. The bottom row (`9, 3`) swaps its nibbles.
- **Output State:** `E739`, Matrix: $\begin{pmatrix} \texttt{E} & \texttt{7} \\ \texttt{3} & \texttt{9} \end{pmatrix}$

Hardware Task: Implement this as a simple wire-swapping operation in Verilog. For a 16-bit state `in[15:0]`, the logic is `assign out = {in[15:12], in[11:8], in[3:0], in[7:4]};`.

Mix Columns

- **Purpose:** To provide further diffusion by mixing data within each column. This operation acts independently on each of the two columns.
- **Table-Based Implementation:** The complex math is simplified using a lookup table for the "multiply by 4" function. The formulas for a column $\begin{pmatrix} N_1 \\ N_2 \end{pmatrix}$ are: $N_1' = N_1 \oplus (4 \cdot N_2)$ and $N_2' = (4 \cdot N_1) \oplus N_2$

Example Walkthrough: Assume the 16-bit state is `6E39`.

1. **Arrange into Matrix:** $\begin{pmatrix} \texttt{6} & \texttt{E} \\ \texttt{3} & \texttt{9} \end{pmatrix}$

Table 3: "Multiply by 4" Lookup Table

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	0	4	8	C	3	7	B	F	6	2	E	A	5	1	9	D

2. **Process Column 0** $\begin{pmatrix} \texttt{6} \\ \texttt{3} \end{pmatrix}$:
   - Lookup $4 \cdot 6 \rightarrow$ `B`.
   - Lookup $4 \cdot 3 \rightarrow$ `C`.
   - New Top: $\texttt{6} \oplus \texttt{C} = \texttt{A}$.
   - New Bottom: $\texttt{B} \oplus \texttt{3} = \texttt{8}$.
3. **Process Column 1** $\begin{pmatrix} \texttt{E} \\ \texttt{9} \end{pmatrix}$:
   - Lookup $4 \cdot \texttt{E} \rightarrow \texttt{9}$.
   - Lookup $4 \cdot \texttt{9} \rightarrow \texttt{2}$.
   - New Top: $\texttt{E} \oplus \texttt{2} = \texttt{C}$.
   - New Bottom: $\texttt{9} \oplus \texttt{9} = \texttt{0}$.
4. **Combine Results:** The new state is `AC80`.

Hardware Task: Implement the "multiply by 4" table as a combinational block (e.g., a `case` statement). Use its outputs to perform the XOR operations.

Key Expansion

- **Purpose:** To generate three 16-bit round keys from the 16-bit master key.
- **Table-Based Implementation:** This process uses the **S-Box** and a **Round Constant (Rcon)** table.

Table 4: Round Constant (Rcon) Table

Round 1	1000 0000 (0x80)
Round 2	0011 0000 (0x30)

Example Walkthrough: Assume the Master Key is `2D65`.

1. **Generate Round Key 0** (K_0)**:**
   - Split the key: w_0 `= 2D`, w_1 `= 65`.
   - $K_0 =$ `2D65`.
2. **Generate Round Key 1** (K_1)**:**

- **Calculate g(w_1)** where w_1 = `65`:
  - Rotate Nibbles: `65` $\rightarrow$ `56`.
  - S-Box Substitute `5` $\rightarrow$ `1` and `6` $\rightarrow$ `8`. Result is `18`.
  - XOR with Rcon(1): `18` $\oplus$ `80` $=$ `98`.
- **Calculate w_2:** w_2 = $w_0 \oplus$ `g(`w_1`)` $=$ `2D` $\oplus$ `98` $=$ `B5`.
- **Calculate w_3:** w_3 = $w_2 \oplus w_1 =$ `B5` $\oplus$ `65` $=$ `D0`.
- $K_1 =$ `B5D0`.

3. **Generate Round Key 2 (K_2):**
   - **Calculate g(w_3)** where w_3 = `D0`:
     - Rotate Nibbles: `D0` $\rightarrow$ `0D`.
     - S-Box Substitute `0` $\rightarrow$ `9` and `D` $\rightarrow$ `E`. Result is `9E`.
     - XOR with Rcon(2): `9E` $\oplus$ `30` $=$ `AE`.
   - **Calculate w_4:** w_4 = $w_2 \oplus$ `g(`w_3`)` $=$ `B5` $\oplus$ `AE` $=$ `1B`.
   - **Calculate w_5:** w_5 = $w_4 \oplus w_3 =$ `1B` $\oplus$ `D0` $=$ `CB`.
   - $K_2 =$ `1BCB`.

Hardware Task: Create a module that performs these steps.

AddRoundKey

- **Purpose:** This is the operation that mixes the secret key with the data state. It is the only step that uses the round keys.

Example:

- **Input State:** `AC80`
- **Round Key (K_1):** `B5D0`
- **Process:** Perform a 16-bit XOR operation.
  `1010 1100 1000 0000` $\oplus$ `1011 0101 1101 0000`
- **Output State:** `0001 1001 0101 0000`, which is `1950`.

Hardware Task: Implement this as a purely combinational module that takes the 16-bit state and 16-bit round key as inputs. The logic is a single line: `assign out = state` $\oplus$ `round_key;`.

Hardware Implementation Architecture

To promote an efficient, resource-aware design, your implementation must be sequential and reuse a single set of core transformation modules. You are required to instantiate only one of each of the following:

- One top-level S-Box module (which internally manages the four parallel 4-bit S-Box lookups).

- One `ShiftRows` module.
- One `MixColumns` module.
- One `KeyExpansion` module.

This architectural constraint means you cannot create a fully unrolled, combinational datapath by chaining separate modules for each round. Instead, you must design control logic to manage the data flow. This controller will use a central state register, feeding its value back through this single set of transformation modules over multiple clock cycles to complete the encryption process.

Top-Level FPGA Design

Note on Key Input:

The FPGA development board have a limited number of switches (e.g., 16). Since the plaintext input requires all 16 switches, you will hardcode the Master Key inside your Verilog design instead of using physical switches for it.

- **Inputs:**
  - `clk`: 100MHz clock from the FPGA board.
  - `rst`: A reset signal (e.g., a button).
  - `sw[15:0]`: 16 switches for the **Plaintext** input.
- **Outputs:**
  - `led[15:0]`: 16 LEDs for the **Ciphertext** result.
- **Functionality:**
  1. Inside your top-level module, declare the 16-bit Master Key as a fixed parameter. For example:
     `localparam MASTER_KEY = 16'h2D65;`
  2. This `MASTER_KEY` will be the input to your `key_expansion` module.
  3. The output of the final round is directly connected to the `led` display.

Analysis and Report Questions

1. **BRAM vs. LUTs:** Explain the primary advantages of implementing the S-Box in BRAM instead of letting the synthesizer build it from LUTs.
2. **Resource Utilization:** Provide a summary from the Vivado Utilization Report. How many LUTs, Flip-Flops, and BRAM tiles did your design consume?
3. **Critical Path:** What is the critical path in your design?.

Submission Guidelines

Submit a single `.zip` file containing:

1. **Verilog Source Files (`.v`):** All your modules.
2. **Memory Initialization Files (`.coe`):** Files for the S-Box and Round Constants.
3. **Constraints File (`.xdc`):** The file mapping your I/O to the FPGA pins.
4. **A Report (PDF):** Containing your answers to the analysis questions.