DLDCA LAB 3: Verilog

August 13, 2025

Introduction

In this lab, you will be implementing a simplified[1] version of AES (Advanced Encryption Standard) block cipher in Verilog. You would have to implement the encryption rounds, key expansion and various transformation operations.

Note that the implementation may not exactly follow the AES standard, this is by design, as we have simplified AES to make it easier to implement in verilog.

Necessary tools

We shall be using `icarus-verilog` alias `iverilog` inorder to compile verilog HDL. To visualise waveforms we shall use `gtkwave`.

Before starting the lab, ensure that both tools are installed in your system. This can be verified by running

```
$ iverilog
iverilog: no source files.
Usage: iverilog [-EiRSuvV] [-B base] [-c cmdfile|-f cmdfile]
           [-g1995|-g2001|-g2005|-g2005-sv|-g2009|-g2012] [-g<feature>]
                [-D macro[=defn]] [-I includedir] [-L moduledir]
                [-M [mode=]depfile] [-m module]
                [-N file] [-o filename] [-p flag=value]
                [-s topmodule] [-t target] [-T min|typ|max]
                [-W class] [-y dir] [-Y suf] [-l file] source_file(s)

See the man page for details.
```

[1] does not include `mixcolumns`

```
$ gtkwave --version
GTKWave Analyzer v3.3.116 (w)1999-2023 BSI

GTKWAVE | Use the -h, --help command line flags to display help.
WM Destroy
```

Structure of submission/templates

Submission format:

```
your-roll-no/
|- task1/
    |- module.v (TODO)
    |- testbench.v
|- task2/
    |- module.v (TODO)
    |- testbench.v
```

The folder structure of the expected submission for this lab is given above. We expect this folder to be compressed into a tarball with the naming convention of your-roll-no.tar.gz. The command given below can be used to do the same

```
$ tar -cvzf your-roll-no.tar.gz your-roll-no/
```

Cipher Overview

In digital systems, all data—whether textual, numerical, or multimedia—is ultimately represented as sequences of binary digits (bits), each having a value of 0 or 1. When such binary data is transmitted over a communication channel or stored on a medium in its plaintext form, any entity with access to that channel or medium can interpret its contents.

Encryption is the computational process of transforming plaintext into an unintelligible format called ciphertext, in such a way that reversing the transformation requires knowledge of a specific secret value known as a key. This process ensures confidentiality of information, even if the ciphertext is intercepted.

AES is widely adopted due to:

- High computational efficiency on both hardware and software platforms.

- Strong resistance to known cryptanalytic attacks.
- A well-defined mathematical structure that is publicly vetted.

Block size: AES operates on fixed-size blocks of 128 bits (16 bytes).

Key sizes: Supported key lengths are 128, 192, or 256 bits.

Number of rounds:

- 10 rounds for 128-bit keys
- 12 rounds for 192-bit keys
- 14 rounds for 256-bit keys

Each round applies a series of non-linear and linear transformations designed to progressively obscure the statistical structure of the input data.

The AES state is represented internally as a 4×4 matrix of bytes. Each round (except the final) applies the following operations:

- **SubBytes**:A non-linear byte substitution using a precomputed substitution box (S-box)
- **ShiftRows**:A cyclic left shift of each row of the state matrix by a different offset
- **MixColumns**: Skipped for simplicity
- **AddRoundKey**:A bitwise XOR of the state with a round-specific subkey derived from the main key via the AES key schedule.

In this lab, your AES implementation would have all the modules except the MixColumns(Not included at all), AddRoundKey(Part of outlab).

Tasks

Task 1: SBox Module

An S-Box is nothing but a look-up table. Given a 4-bit number(called nibble), it gives the 4-bit number mapped to it. The function itself is non-linear and helps in encryption of the message.

In order to implement the look-up table, for these following mappings, draw K-maps and find the corresponding boolean function.

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

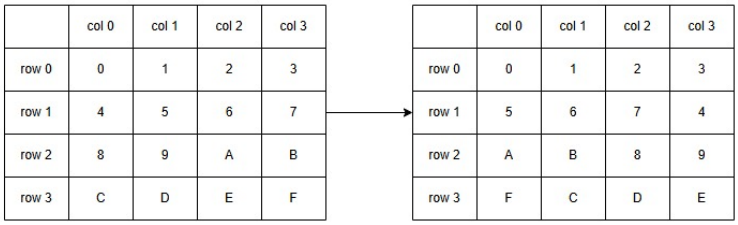
Figure 1: S-Box specification

Task 2: ShiftRows Module

This module is to shift around the bits in the row. The way that

- **Input**: currentState (64-bit) — the state matrix after SBox, arranged as 16 nibbles (4-bit each).
- **Output**: nextState (64-bit) — the state after shifting rows.
- **Function**: ShiftRows cyclically shifts each row of the 4×4 nibble matrix:
  - Row 0: no shift
  - Row 1: left shift by 1 nibble (4 bits)
  - Row 2: left shift by 2 nibbles (8 bits)
  - Row 3: left shift by 3 nibbles (24 bits)

Note: A nibble is a 4-bit chunk (half a byte). The 64-bit input is made up of 16 such nibbles, arranged logically as a 4×4 matrix in row-major order.



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

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

Figure 2: 4x4 Nibble matrix order