DLDCA LAB 3: Verilog

Group 1

August 20, 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/
|-outlab/
    |- module.v (TODO)
    |- testbench.v
    |- shiftrows_tb.v
    |- sbox_tb.v
    |- nextkey_tb.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 Mix-Columns

Each transformation is implemented as a separate module. Note that you may have implemented some of these modules in your in-lab, feel free to reuse those implementations.

Tasks

Lets begin with the simple tasks. Remember that your whole circuit should be combinational.

Task 1: AddRoundKey Module

Implement a combinational circuit that XORs the current state and roundKey to give out the next state.

```
module AddRoundKey(
    input  [63:0] currentState ,
    input  [63:0] roundKey     ,
    output [63:0] nextState
);
```

Task 2: SBox Module

Implement a substitution box transformation. It is a bijection from 4 bits $x_1x_2x_3x_4$ to 4 bits $y_1y_2y_3y_4$. 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

You can copy your implementation of SBox module from your inlab.

Task 3: ShiftRows Module

Implement a bit rearrangement that simulates shifting rows in the state matrix. The first row does not change. The second row shifts to the right by 1 nibble. Third row shifts to the right by 2 nibbles and the last row by 3. You can fill up the matrix starting from the MSB of the state. You can copy your implementation of ShiftRows module from your inlab.

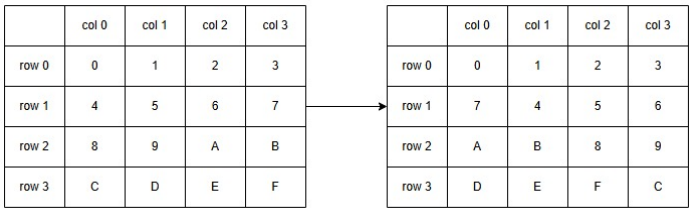

Figure 2: AES ShiftRows Operation

Task 4: NextKey Module

Each round uses a different key. The key for the current round is obtained by shifting the previous round's key to the left by 4 bits. This operation is also called as **key expansion**. Implement this module and you will be able to get the keys for all the 10 rounds. Beware that this is a rotation operation and not shift.

Task 5: Round Module

A round consists of 4 operations in the given order:

1. SubBytes
2. ShiftRows
3. AddRoundKey

This is the format for the 10 rounds.

Task 6: Encrypt Top-level Module

Integrate all your modules to build a full encryption datapath, including key expansion and round sequencing. The below image summarizes it quite well. The pre-round transformation just includes AddRoundKey. Later you have to apply 10 rounds on the output of pre-round transformation

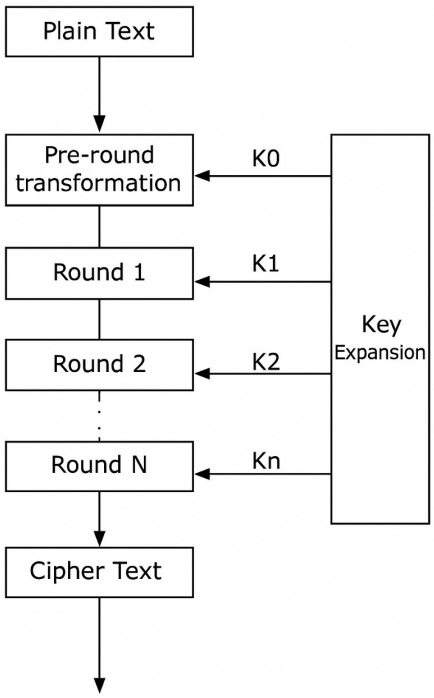

Figure 3: AES Encryption Datapath

```
module Encrypt(
    input [63:0]  plaintext  ,
    input [63:0]  secretKey  ,
    output [63:0] ciphertext
);
```

Hint: You can use generate block for applying the 10 rounds on the input plaintext and for key expansion.

You are expected to fill in the template code for all of the tasks and submit the same in the given format.