University of Illinois at Urbana-Champaign

ECE385
Digital Systems Laboratory

Lab7: SOC With Advanced Encryption Standard in SystemVerilog

TA: Abigail Wezelis, Harris Mohamed

Yan Miao, Guangxun Zhai
yanmiao2, gzhai5

April 2021

Contents

1 Introduction

During this lab, we implemented a 128-bit Advanced Encryption Standard (AES) encryptor and decryptor using C and SystemVerilog. In the first week, we worked on C code to write a software encryptor; and in the second week, we focused on hardware coding and implemented decryption. The inputs for the lab are message and key, and our goal is to encrypt/decrypt them.

2 Written Description and Diagrams

2.1 Written description of the software encryptor

With software encryption, we first get the 32-byte message and key from the keyboard using the JTAG-UART set up in Lab6. Then we used a charToHex function to convert the 32-byte(256-bit) ASCII value to 16-byte(128-bit) HEX value so that we can apply the AES-128 encryption method. With the preprocessing data ready, then we followed the exact AES encryption flowchart provided in the document: keyExpansion(), 9 rounds of substitute(), shift(), mix_column() and add_key() and a final slightly-modified round operation to get our final encrypted message.

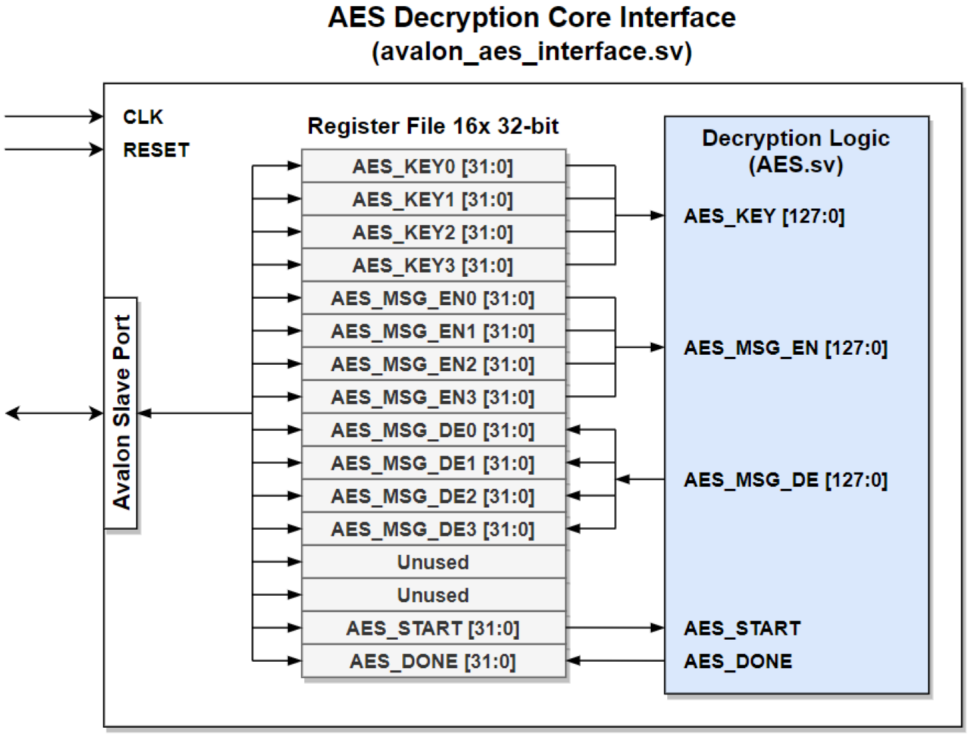
2.2 Written description of the hardware decryptor

With hardware decryption, we did almost the exact inverse operation of encryption. The major difference is we instantiate hardware components and a state machine to finish the operation instead of writing software functions(For example, the keyExpansion() function in C will be replaced by a SystemVerilog module). The state machines will have a counter(replacing for loops in software decryptor) and multiple control signals telling us which state(operation) the decryptor is executing. In all, it still goes through similar process as software encryptor: keyExpansion(), 9 rounds of inverse_substitute(), inverse_shift(), inverse_mix_colunb() and add_key() and a final slightly-modified round operation to get our final decrypted message.

2.3 Written description of the hardware/software interface

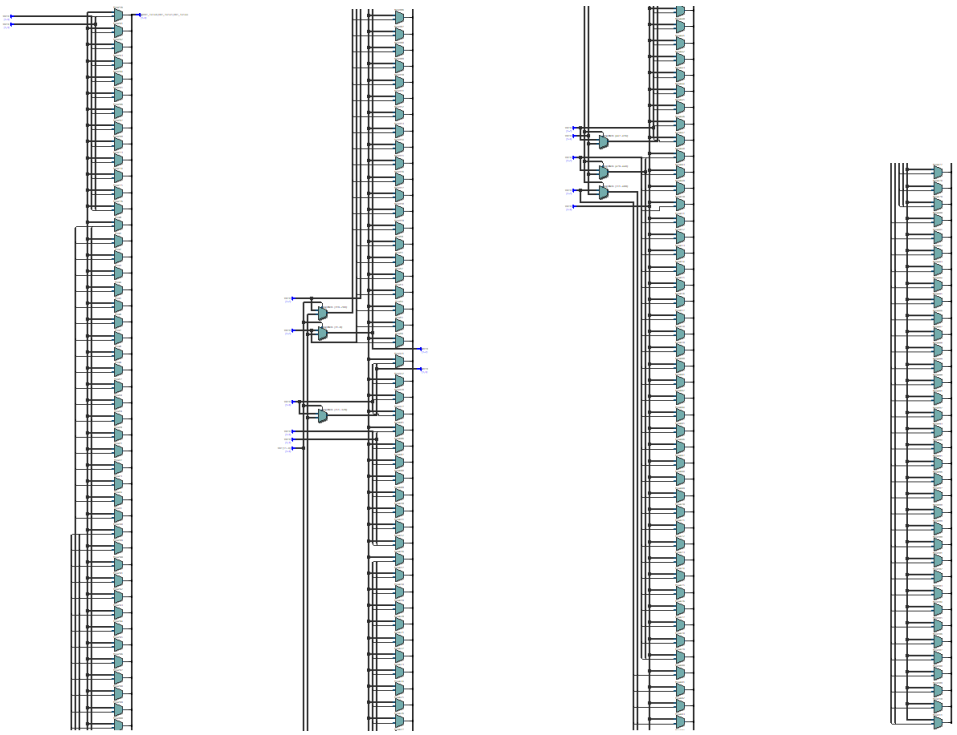
Like the LC3 Register File design in Lab5, we used a combination of the packed and the unpacked array to design the reg file: 16 registers, each with 32bits.

```
1         logic [31:0] registers [15:0];
```

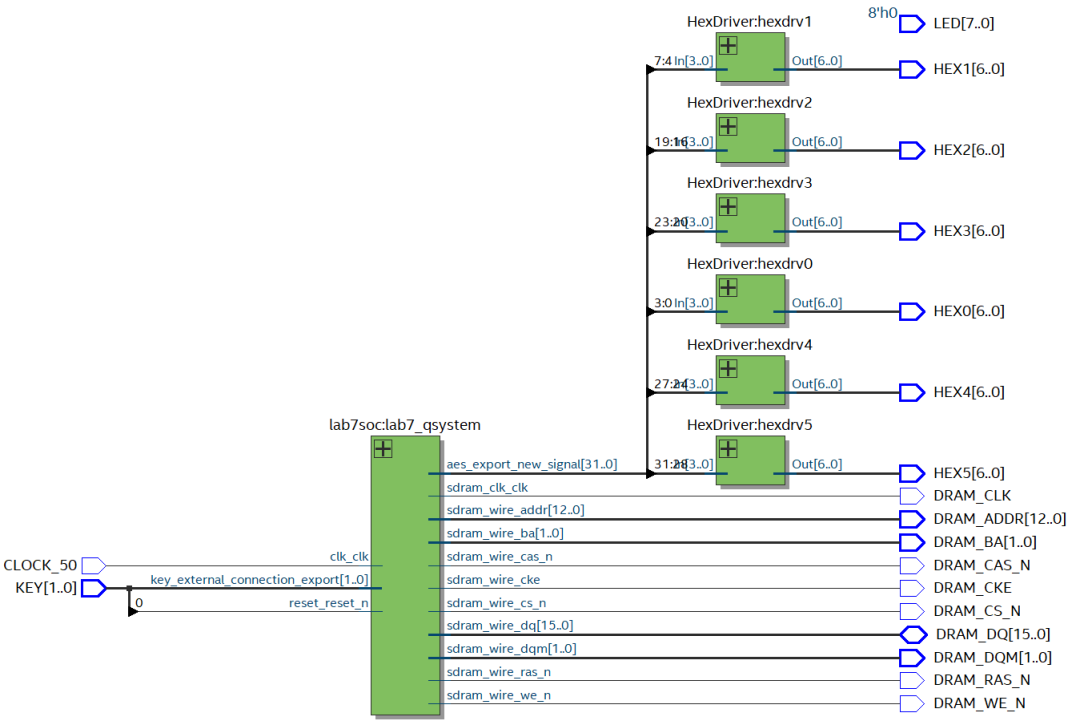


Since we need a 0-cycle read latency, we decided to put read data in an always_comb block; and we need a 1-cycle write latency, we decided to sync the write operation in an always_ff block. Also given the fact we need to support write byte and half, we also integrate the write_enable_signal to make sure we always write to the correct location of the correct register. Certain registers(E.g. the one holding key and msg) will be directly connected to the AES module where we designed our state machine and instantiate the decryption module so that the result message key and done flag can be directly written by the AES module. The software can access and modify the register file using the exported ptr we set in the platform designer, similar to how we directly wrote to the LED value in Lab6.1. (E.g. AES_PTR[0] = 0x8, is setting the first register value to 8 in the register file.

2.4 Block diagram(s)

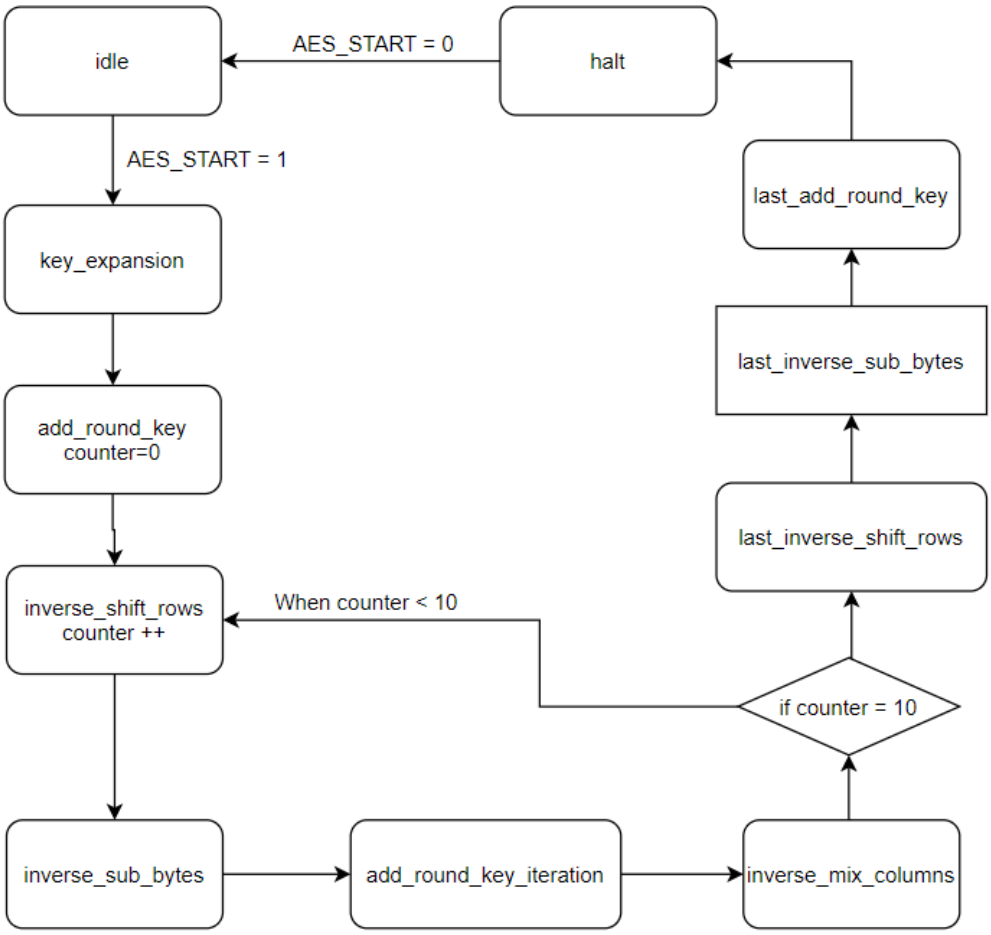


RTL view of avalon_aes_interface.sv



top-level RTL diagram

2.5 State Diagram of AES decryptor controller



3 Module Descriptions

1. **Module: lab7.sv**

   - Inputs:CLOCK_50, KEY, DRAM_DQ
   - Outputs: LED, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, DRAM_ADDR, DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_DQ, DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK
   - Description: This is the top-level file for lab7. Basically, it links every port with the FGPA and NIOS II so that we can run our project.
   - Purpose: serves as an interface between FGPA and the C code.

2. **Module: lab7soc.v**

    - Inputs: clk_clk, key_external_connection_export, reset_reset_n, sdram_wire_dq, spi0_MISO, usb_gpx_export, usb_irq_export
    - Outputs: aes_export_new_signal, hex_digits_export, leds_export, sdram_clk_clk, sdram_wire_addr, sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, sdram_wire_dq, sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n
    - Description: this file is generated by the platform designer for all the hardware components used in the lab 7 design. It includes all the wired connections of for all ports.
    - Purpose: provide all the wires and hardware for the top-level module to use and interact.

3. **Module: AddRoundKey.sv**

    - Inputs: state, round_key
    - Outputs: result
    - Description: This module basically just xor state and round_key and then output the result.
    - Purpose: do the operation of "AddRoundKey", which is one of the essential step in decryption.

4. **Module: InvMixColumns.sv**

    - Inputs: [31:0] in
    - Outputs: [31:0] out
    - Description: each Word of the updating State is considered as a polynomial over GF(2^8).
    - Purpose: do the operation of "InvMixColumns", which is one of the essential step in decryption.

5. **Module: InvShiftRows.sv**

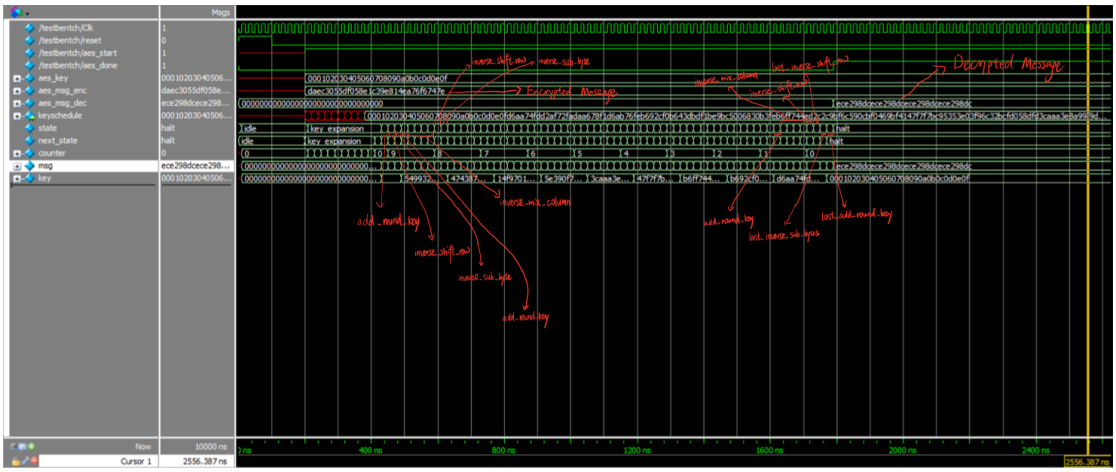
    - Inputs: [127:0] data_in
    - Outputs: [127:0] data_out
    - Description: shift the bytes for different rows. Specifically, row n is right-circularly shifted by n - 1 Bytes.
    - Purpose: do the operation of "InvShiftRows", which is one of the essential step in decryption.

6. **Module: KeyExpansion.sv**
    - Inputs: clk, [127:0] Cipherkey
    - Outputs: [1407:0] KeySchedule
    - Description: take the key and performs a Key Expansion to generate a series of Round Keys (4-Word matrix) and store them into Key Schedule.
    - Purpose: do the operation of "KeyExpansion", which is one of the essential step in decryption.
7. **SubBytes.sv**
    - Inputs: clk, [7:0] in
    - Outputs: [7:0] out
    - Description: use a look-up table to convert each Byte into another value.
    - Purpose: do the operation of "SubBytes", which is one of the essential step in decryption.
8. **AES.sv**
    - Inputs: CLK, RESET, AES_START, AES_KEY, AES_MSG_ENC
    - Outputs: AES_DONE, AES_MSG_DEC
    - Description: this module includes a finite state machine that can go through every single operation for decryption. Either AES_START or AES_DONE signal is 1 bit. When the state halts and AES_START = 0, the state will still go back to the idle state and wait for the next decryption.
    - Purpose: write about the every step of the decryption.
9. **avalon_aes_interface.sv**
    - Inputs: CLK, RESET, AVL_READ, AVL_Write, AVL_CS, AVL_BYTE_EN, AVL_ADDR, AVL_WRITEDATA
    - Outputs: AVL_READDDATA, EXPORY_DATA
    - Description: this module has 4 registers to hold the AES key, encrypted message, and decrypted message, in total 12 registers each 32 bits. It also has 2 more unused registers and 2 more registers that control the AES_START, AES_DONE signals.
    - Purpose: serves as an interface to connect to the AES.sv. The AES.sv uses the register written here to do the decryption steps.

4 Annotated Simulation of the AES decryptor

```
initial begin

    reset = 1'b1;

    #10 ;

    reset = 1'b0;

    #10
    aes_start = 1'b1;

    aes_key = 128'h000102030405060708090a0b0c0d0e0f;
    AES_MSG_ENC = 128'hdaec3055df058e1c39e814ea76f6747e;


    // check the simulation
    #2000
    aes_start = 1'b0;

    #5
    // check the webister
    aes_key = 128'h3b280014beaac269d613a16bfdc2be03;
    AES_MSG_ENC = 128'h439d619920ce415661019634f59fcf63;

    #5
    aes_start = 1'b1;


end
```

test-bench codes



RTL simulation

5 Post-Lab Questions

(a)

LUT	7544
DSP	0
Memory (BRAM)	562,176
Flip-Flop	3542
Frequency	105.64MHz
Static Power	96.18mW
Dynamic Power	0.74mW
Total Power	105.98mW

Design Analysis

(b) **Which would you expect to be faster to complete encryption/decryption, the software or hardware? Is this what your results show?**

We expect the hardware to complete encryption/decryption faster, and this is just what our results showed. Because when doing the hardware, it only requires some combinational logic or logic gate to propagate data. While for the software, the NIOS II has to fetch the data from on-chip RAM and interpret the instructions, which costs more time.

(c) **If you wanted to speed up the hardware, what would you do?**

One of the drawbacks that we believe will influence our speed for the hardware is that we are only allowed to include one instance of InvMixColumns in our state machines. As a result, we created four states for InvMixColumns, and each state deal with 32 bits of messages. These extra three states will also go through 10 rounds, which in total 30 additional states running. As a result, if we pursue higher efficiency, we could write four modules of InvMixColumns, and each module process 32-bit-message.

6 Conclusion

6.1 Bugs Encountered

In Week1, we had a hard time understanding the column major order state. And we initially didn't understand how to use the gf_mul array to get the desire xtime value we need. This Tutorial and the intermediate_result.txt helped us a lot during debugging.

In week1, with the help of simulation, we were able to quickly discover and finish the state machine design. The only major bug we had is how to instantiate/use only 1 invMixColumn(32-bit) to handle 128-bit operation. We eventually solved that using 3 extra states.

6.2 Functionality of Our Design

Our design works both on encryption and decryption and we got all points during the demo in Week1 and Week2. We also fixed the bug of the ball when it bounces from the wall mentioned in the lecture. All provided programs can be executed with correct behavior, details of which can be found in the above report.

6.3 Suggestions to Lab-Manual and Given Document

We found a possible error during our demo when we want to copy and paste the message and key from the 385 website to the running program. Whenever we switch the window or type in the programs using non-English input, the Eclipse will crash. Therefore, we have to write the input in the c code instead of depending on "type functions" to get the values from keyboards. We hope the future documents could include this error or just cancel the type-in message and keys and replace it with given input codes so that we can also save some time typing the 32-bit characters.