ECE 385

Fall 2021

Experiment #9

Lab 9

SOC with Advanced Encryption Standard in SystemVerilog

Guanshujie Fu 3190110666

Xiaomin Qiu 3190110717

Lab Section D231 18:00 – 21:00

TA: Yuchuan Zhu

1. Introduction

i. Briefly summarize the operation of the AES encryptor/decryptor.

The operation of the AES encryptor is done in software on the NIOS processor and decryptor is done in hardware implementation, which could accelerate process in this case. The operation requires 128-bit message and key, together with the encrypted and decrypted message, it could get the original message which should be the decrypted message. With the help of an IP core, it could read and write to the register file in SystemVerilog.

2. Written Description and Diagrams of the AES encryptor/decryptor

i. Written description of the software encryptor

a. Describe the role of the NIOS processor as well as the basic functionality of your C code

The software encryptor of the experiment starts with the console instructing us to read the data from the keyboard to get the required key and plaintext message. Then, the main function calls the encryption function that converts the message and the key into a char array of hexadecimal values. At the same time, we create a new variable to hold the entire keyschedule and initialize a message variable, msgChar, to hold the current message. In addition, we initialize the values it contains in the function setAllKeys. After that, the program calls the key extension function, which uses the cryptographic key to generate the keys for all other 9 rounds. After that, we first call the addRoundKey function, which does a bitwise XOR between the message and the roundKey. after that, we enter a loop that runs 9 times, each time executing the SubBytes, ShiftRows, MixColumns and addRoundKey functions. The roundKey function takes the given msg input and XORs it with a round key and stores it in the msg. For the Subbytes function, we use a lookup table to find the multiplicative inverse of each byte in the message. Then for the ShiftRows function, we perform a circular left shift for all rows, with row 0 being shifted 0 times and row 3 being shifted 3 times, and for the MixColumns function, we again use a lookup table to make the necessary column changes to msgchar as needed for encryption. After this loop, we finally call SubBytes, ShiftRows and addRoundKey again to get the final result. Now we have the encrypted message. We store it in the input variable msg_enc by converting the 4 char entries in msgchar to 1 int entry in msg_enc and do the same for the key. All C programs are executed on top of NIOS II. In order for the scanf function to work from our computer console, we must remember to add a JTAG UART module. It communicates with the Avalon bus and sends the encrypted messages to the bus, while receiving the decrypted ones. This stores the encrypted information and the first round of keys in a register file, allowing the decryptor to use it when needed.

ii. *Written description of the hardware decryptor*

a. *Describe the basic steps of decryption and how this is controlled and computed in hardware*

The hardware decryption part of this experiment is handled by the state machine in AES.sv. In the AES module, we instantiate some modules while using a finite state machine to compute the decryption. With the InvMixColumns, InvShiftRows, KeyExpansion and InvSubBytes modules, we are able to compute the inverse results of all the encryption functions to achieve decryption. For the finite state machine, we use a counter in order to keep track of which round we have been in. We take the key from registers 0 to 3 and the encrypted information from registers 4 to 7. We also took the LSBs of registers 14 and 15, the Start and Done registers, to start and pause the state machine in order to load variables and store information correctly. We start with the KeyExpansion module, for which we set aside 11 clock cycles so that it finishes before starting the decryption process. After one round of key generation, we did addRoundKey. then we performed 9 cycles of InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns. We only allowed one instantiation of InvMixColumns, so we had 4 states for this operation, each doing 1 column matrix, and only one for the other three operations. After the loop, we execute InvShiftRows, InvSubBytes and AddRoundKey one last time to get the final decryption information, which we then store in registers 8 to 11 in the register file.

iii. *Written description of the hardware/software interface (avalon_aes_interface.sv)*

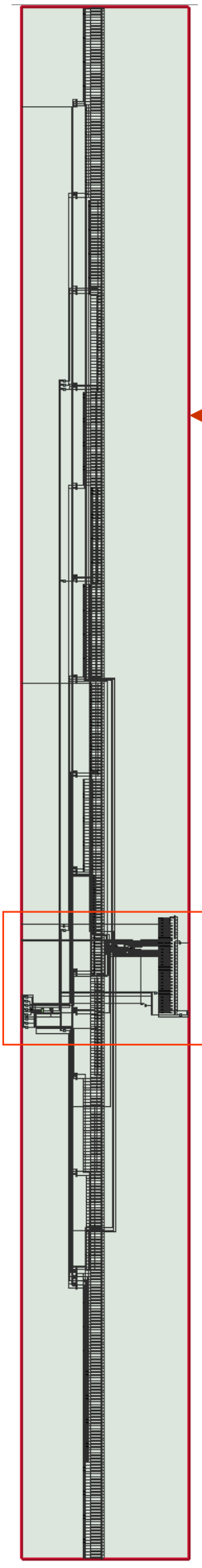
a. *Describe how the system sends data between NIOS and the hardware decryptor and how the register file is designed*

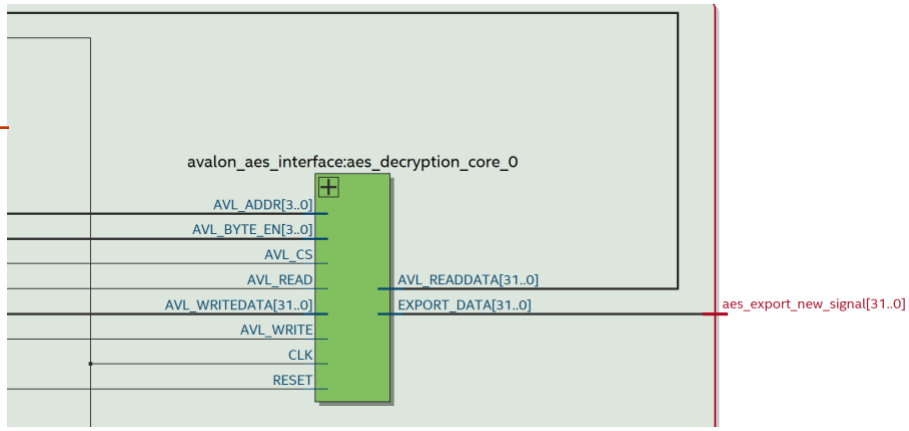
The hardware/software interface is built on the avalon_aes_interface module. In this module, we create 16 32-bit registers. It receives CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, AVL_BYTE_EN, AVL_ADDR, AVL_WRITEDATA, AVL_READDAT signals to decide whether to write to the register file, what registers to write and which bits to change, and outputs a 32-bit register named EXPORT _DATA, which is a 32-bit register used to store data for hexadecimal display. After the NIOS II has finished encrypting, registers 0 to 3 hold the key, 4 to 7 hold the encryption information, and 8 to 11 hold the decryption information. Register 14 is used by the C program to signal the hardware decryption to start the process. Register 15 is used by the hardware decryption module to inform the C program that it has finished decrypting the message. Any other data is sent through unused registers 12 and 13. The register file is written by using the data in the AVL_WRITEDATA modified in the NIOS

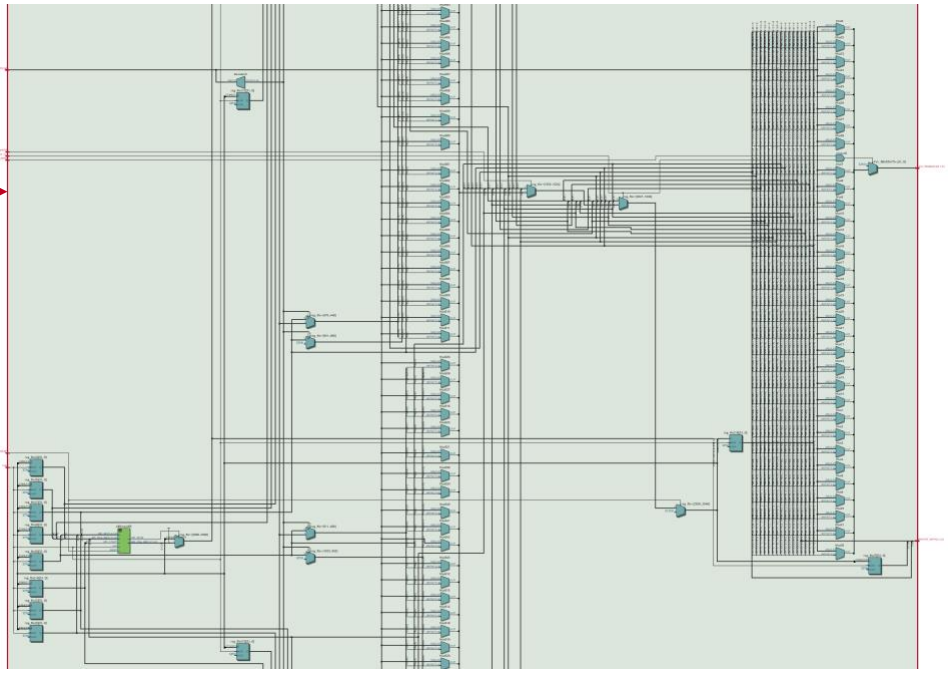
processor.The AVL_WRITE and AVL_CS signals are also modified in the NIOS processor so that we only write to it after the encryption process is complete.AVL_READ is used to check the data used in register AVL_ADDR, which the NIOS processor uses it, reads the data from the decrypted message, sends the encrypted message to the register file of the avalon_aes_interface module, and then the decryptor can take over and check the completion register to confirm that the decrypted message is valid.

iv. *Block diagram*

a. *Please include the RTL view of avalon_aes_interface.sv. The Qsys view of the NIOS processor or lab9_top.sv is not necessary for this portion.*

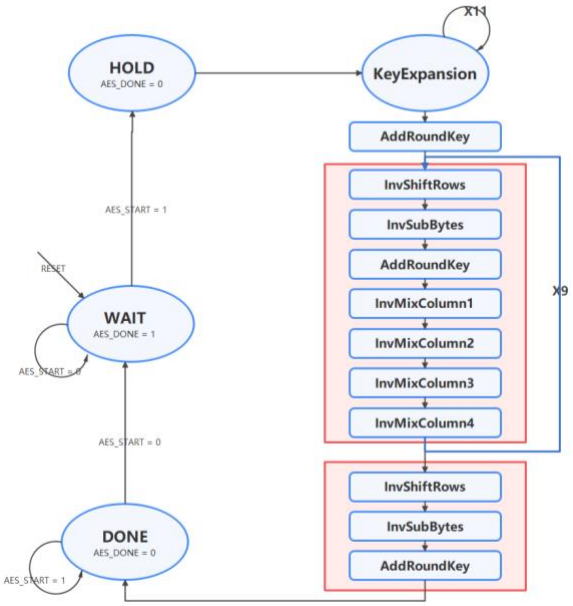


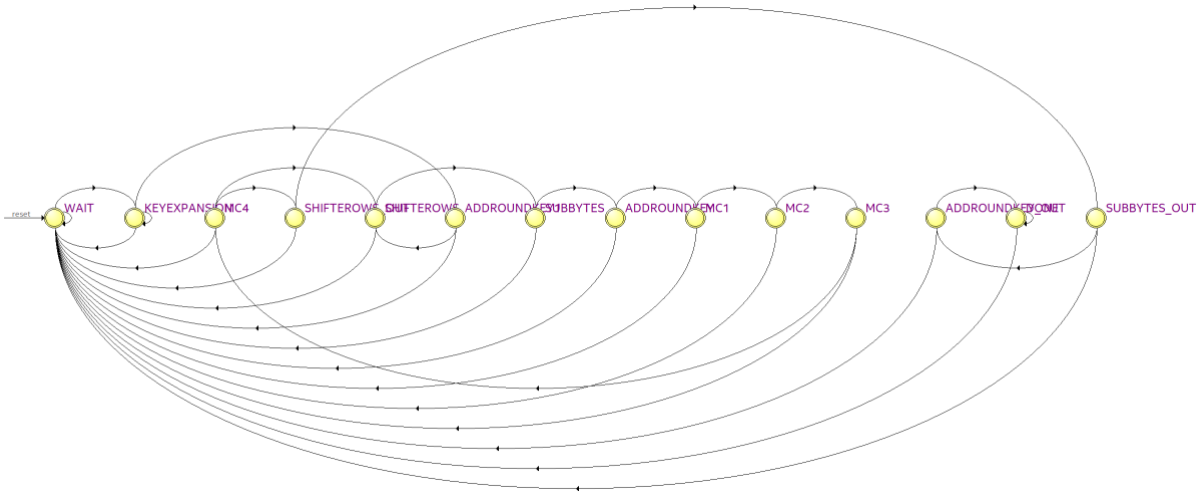

avalon_aes_interface:aes_decryption_core_0
AVL_ADDR[3..0]
AVL_BYTE_EN[3..0]
AVL_CS
AVL_READ
AVL_WRITEDATA[31..0]
AVL_WRITE
CLK
RESET
AVL_READDATA[31..0]
EXPORT_DATA[31..0]
aes_export_new_signal[31..0]



v. *State Diagram of AES decryptor controller*

a. *This is the state machine that was written in AES.sv. You may abbreviate the 9 looping rounds in the state diagram like in figure 9 on page IAES.9 of the lab manual.*





vi. *Module descriptions*

a. *A guide on how to do this was shown in the Lab 5 report outline. Do not forget to describe the Qsys generated file for your Nios system!*

Modules	**InvAddRoundKey.sv**
Inputs	input logic [127:0] state input logic [127:0] key
Outputs	output logic [127:0] result
Description	The module XORs the input information along with the round key.
Purpose	This module is used for the decryption process and we use

	it to subtract the round key from the message and output the result to the output.

Modules	**SubBytes.sv**
Inputs	input logic clk input logic [7:0] in
Outputs	output logic [7:0] out
Description	This module takes 8 bits of input, converges each cell of the decrypted message with the table used in AES, and outputs the 8-bit result with the modified data.
Purpose	This module will be used for the encryption of each temporary variable that houses the decryption of each cycle of the AES algorithm, thus helping us to perform the transformation in Rijndael's finite field.

Modules	**lab9_top.sv**
Inputs	input logic CLOCK_50 input logic [1:0] KEY
Outputs	output logic [7:0] LEDG output logic [17:0] LEDR output logic [6:0] HEX0 output logic [6:0] HEX1 output logic [6:0] HEX2 output logic [6:0] HEX3 output logic [6:0] HEX4 output logic [6:0] HEX5 output logic [6:0] HEX6 output logic [6:0] HEX7 output logic [12:0] DRAM_ADDR output logic [1:0] DRAM_BA output logic DRAM_CAS_N output logic DRAM_CKE output logic DRAM_CS_N inout logic [31:0] DRAM_DQ output logic [3:0] DRAM_DQM output logic DRAM_RAS_N output logic DRAM_WE_N output logic DRAM_CLK

Description	This module is the top-level of this lab. It provides the data as well as displays values on the hex displays on the FPGA and connects all the corresponding wires of hexdrivers and sdram.
Purpose	This module helps to display data on the hexadecimal display and instantiates the Qsys design for proper access and manipulation of memory.

Modules	**KeyExpansion.sv**
Inputs	input logic clk input logic [127:0] Cipherkey
Outputs	output logic [1407:0] KeySchedule
Description	This module instantiates the KeyExpansionOne module 10 times and generates the round key used in the decryption algorithm from the output of this function.
Purpose	This module is used for the AES algorithm, and the generated round keys are used in the addRoundKey module, where each round key is XORed with temporary information to generate the system's key plan.

Modules	**InvShiftRows.sv**
Inputs	input logic [127:0] data_in
Outputs	output logic [127:0] data_out
Description	This module creates a number of temporary variables and reverses the shift operations given in the AES algorithm. It takes 128 bits of input and outputs the modified data.
Purpose	This module does inverse shift rows operations through the 10 loops involved in the decryption process.

Modules	**InvMixColumns.sv**
Inputs	input logic [31:0] in
Outputs	output logic [31:0] out
Description	This module helps us to perform the decryption algorithm by instantiating several submodules that handle the unwinding of the column mixture performed during the encryption process. It accepts 32-bit input and outputs the result of the modified data.
Purpose	This module will be instantiated in the first 9 cycles of the

	AES decryption process and reverse multiplied with 1 column of the decrypted message at a time.

Modules	**hexdriver.sv**
Inputs	input logic [3:0] In
Outputs	output logic [6:0] Out
Description	This is a hex driver that transfers a 4-bit signal to represent a hex display on the FPGA. The 4 bits in are needed to have designations for hex values 0 to F.
Purpose	It helps to present the output hex-value on the board.

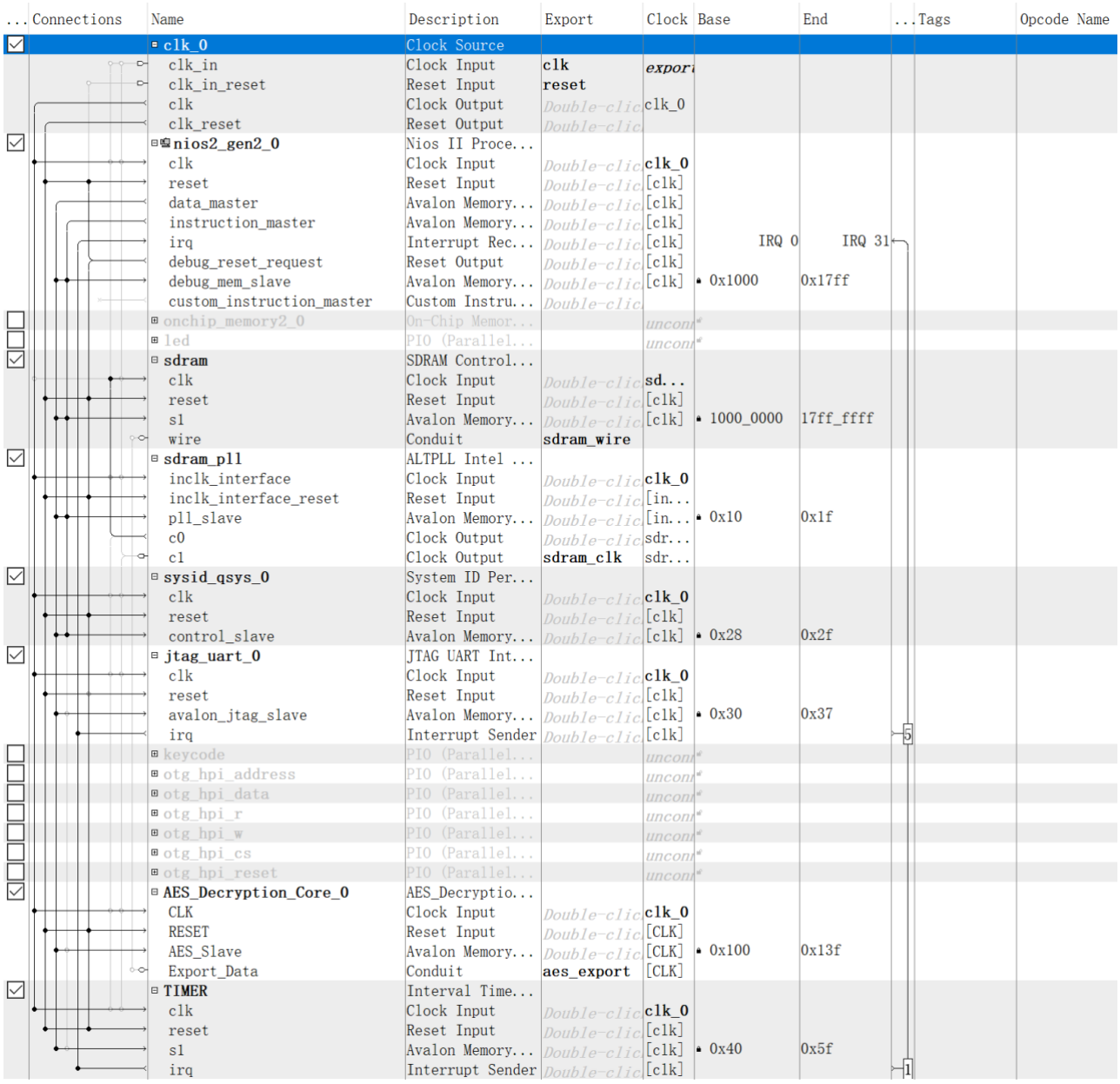
Modules	**avalon_aes_interface.sv**
Inputs	input logic CLK input logic RESET input logic AVL_READ input logic AVL_WRITE input logic AVL_CS input logic [3:0] AVL_BYTE_EN input logic [3:0] AVL_ADDR input logic [31:0] AVL_WRITEDATA
Outputs	output logic [31:0] AVL_READDATA output logic [31:0] EXPORT_DATA
Description	This module creates a register file consisting of 16 registers that act as gates between the hardware and the SystemVerilog program, each with a different purpose. It holds the logic for reading and writing in the various registers in the bank, and also instantiates AES.sv, which writes data to the registers according to the AVL_BYTE_EN signal. Finally, it is able to send the encrypted information to the AES module for decryption as well as store the final information in the registers for software access.
Purpose	This module controls the interface between the SystemVerilog, software and hardware processes in this lab and fully interacts with AES_Decryption_Core in QSYS. It controls the flow of data from the registers it creates to the hardware/software part of this lab. It also controls which data is to be output to the hex driver for

	display.

Modules	**AES.sv**
Inputs	input logic CLK input logic RESET input logic AES_START input logic [127:0] AES_KEY input logic [127:0] AES_MSG_ENC
Outputs	output logic AES_DONE output logic [127:0] AES_MSG_DEC
Description	This module is the central state machine for decryption and is responsible for overseeing the entire decryption process. And it instantiates the KeyExpansion, InvShiftRows, AddRndKey, InvMixColumns and InvSubBytes modules to fully decrypt the encrypted information of the given first key.
Purpose	This module is the main module and is able to control the data from several different decryption modules, doing most of the work for the decryption operation. It chooses which output from the different modules should be the next temporary message and sends the DONE signal at the end of the process.

Modules	**lab9_soc.v**
Inputs	input wire clk_clk, input wire reset_reset_n
Outputs	output wire [31:0] aes_export_new_signal output wire sdram_clk_clk output wire [12:0] sdram_wire_addr output wire [1:0] sdram_wire_ba output wire sdram_wire_cas_n output wire sdram_wire_cke output wire sdram_wire_cs_n inout wire [31:0] sdram_wire_dq output wire [3:0] sdram_wire_dqm output wire sdram_wire_ras_n output wire sdram_wire_we_n
Description	This is a system-on-a-chip module that houses the NIOS II

	chip and all other components that communicate with the Avalon bus.
Purpose	This module enables the software to write to the register file on the hardware and read the decrypted information through the Avalon_AES_interface module that can communicate with the hardware decryption module.



Clk_0: It serves as the functional clock for the entire system and as a reference for other modules.

nios2_gen2_0: It handles the conversion of the C code into system verilog, which can then be executed on the hardware FPGA board.

Sdram: It allows us to get available sdram on the FPGA board, since the on-chip memory is too small to successfully store and update the available programs. sdram is very useful due to its fast-processing time and low update and output latency.

sdram_pll: This block is a way to account for the small latency within the transfer of data to and from the SDRAM and serves as a separate clock for the system.

sysid_qsys_0: This block verifies correct software and hardware transfers by looking back and forth between C code and SystemVerilog to ensure that the data is transferred in the correct format.

jtag_uart_0: It allows terminal access and is used to debug the software.

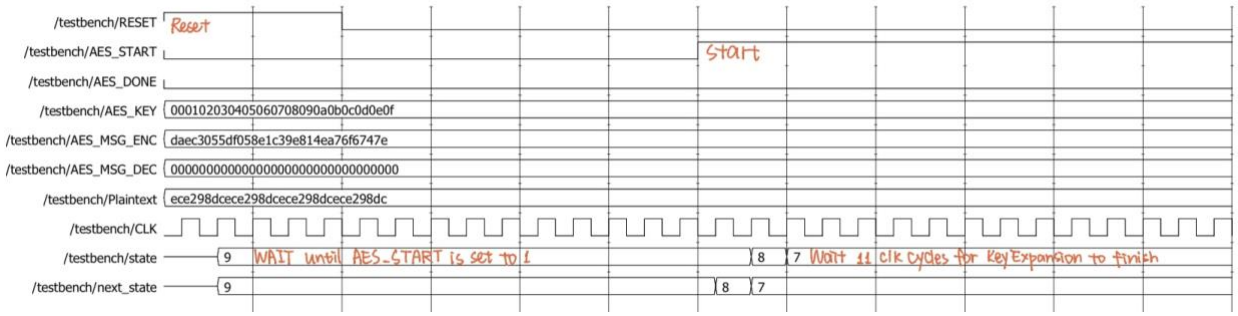
AES_Decryption_Core： It allows input of key schemes and encryption information to the hardware, which is then processed at a significantly faster rate than the software. It can also output decryption information.

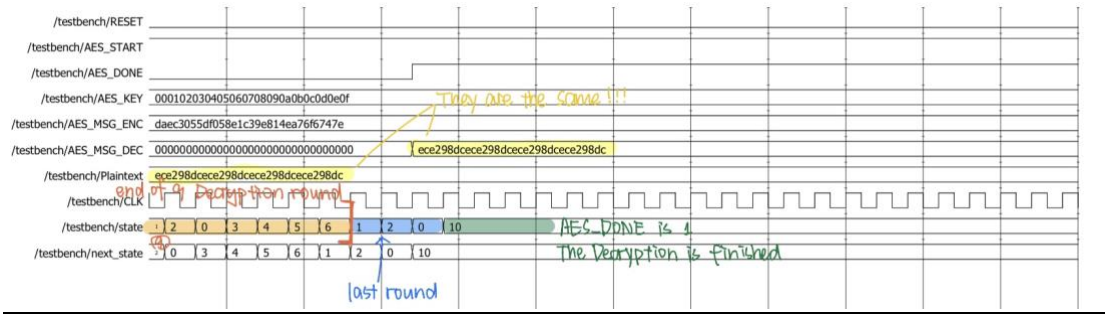
TIMER: It serves as a clock synchronization between the software and the hardware components as the software has no clock.

3. Annotated Simulation of the AES decryptor

i. *Write a testbench for AES.sv and set AES.sv as top level for simulation.*

ii. *In this simulation, you should display the input encrypted message, the input plaintext, the output decrypted message and the current state of the state machine. Notate various points of interest in the simulation (such as when the decryptor finishes decrypting, etc.).*





4. Answer to Post Lab Questions

i. *Which would you expect to be faster to complete encryption/decryption, the software or hardware? Is this what your results show? (List your encryption and decryption benchmark here)*

I expect the decryption process to run much faster when run on hardware. This is what our results show.

Encryption Speed: 0.567KB/s

Decryption Speed: 158KB/s

ii. *If you wanted to speed up the hardware, what would you do? (Note: restrictions of this lab do not apply to answer this question)?*

To speed up the hardware, we can use parallel operations to handle key extensions while the rest of the state machine is running. Also, we can instantiate InvMixColumns 4 times, and if we can declare 4 at a time, then we can finish the operation in one state of the state machine.

iii. *Fill out the design resources and statistics table (duplicated here for convenience).*

LUT	5935
DSP	0
Memory (BRAM)	125952
Flip-Flop	2866
Frequency	53.26MHz
Static Power	102.53mW
Dynamic Power	72.29mW
Total Power	242.91mW

5. Conclusion

i. *Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it*

In this lab, we implement AES encryption in C on a NIOS-II processor and decryption with hardware logic. As the methods of the two operations are symmetric to each other, their working efficiency can be compared with the run rate. In the encryption part, we implement those individual functions in a c file and then compile it with Eclipse. The goal is to encrypt the message with the specified key code. While this implementation has the advantage of better readability of the encoding, AES encryption only runs at a slow speed. Besides, decryption is

mainly implemented using hardware logic to decrypt the encrypted message with the given key. Although System Verilog is less readable than C, it has the advantage of running at a relatively high speed.

In conclusion, the design can successfully perform data encryption and decryption operations and all parts are functioning properly.

ii. *Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.*

Nope.