ECE 385

Fall 2020

Experiment #9

SOC with Advanced Encryption Standard in System Verilog

Name: Xiao Shuhong & Lu Yicheng

Lab Section: D225

1. Introduction

1.1. Briefly summarize the operation of the AES encryption/decryption

In lab9, we are going to implement a 128-bit AES as an IP core. We design encryption and decryption in different way: encryption is achieved using software while decryption part is down by hardware. The algorithm of encryption and decryption are quite similar but due to the different implement method, decryption procedure will be much faster as we use hardware accelerator. The procedure can be generalized as first what have our message and key for encryption, then encryption method will encrypt message, using the decryption hardware, we can turn the decrypted message back.

2. Written Description and Diagrams of the AES encryption/decryption

2.1. Written Description of the software encryptor

My C code contain two parts: user interface part and encryption algorithm part.

The user interface part is mainly described in main function. It has two different mode: encryption/decryption function test mode and procedure speed test mode. In function test mode, we are asked to input our message and key, function encryption and decryption are called and decrypted message and encrypted message can be shown; in speed mode, a time function will be called from library to see how fast will the encryption and decryption function be. For function encryption, it contains of several sub-function calling: first is KeyExpansion, which used to build the key schedule; then a AddRoundKey by applying a XOR between input message and round key. After that, we get 9 loops with each contains SubBytes, ShiftRows, MixColumns and AddRoundKey. For the SubBytes function, we use a LUT to find the multiplicative inverse of each byte in the message. For ShiftRows, we get a circle shift for all rows with row 0 shift 0, row 1 shift 1 and so on. For MixColumns, we do a multiply in GF (2^8) and the multiplicand achieved by another LUT. Then we get the final loop with no MixColumns. After the Encryption function, we also have a Decryption function

which do not do the inverse process as encryption but just retrieve data processed by hardware and output when be calling by main function. Besides, there are helper functions charToHex and charsToHex which together convert 2 characters to a byte for one time.

The above is all about our c code, for the NOIS processor, it works as a CPU interacts with memory through Avalon Bus, control the system clock.

2.2. Written description of the hardware decryptor

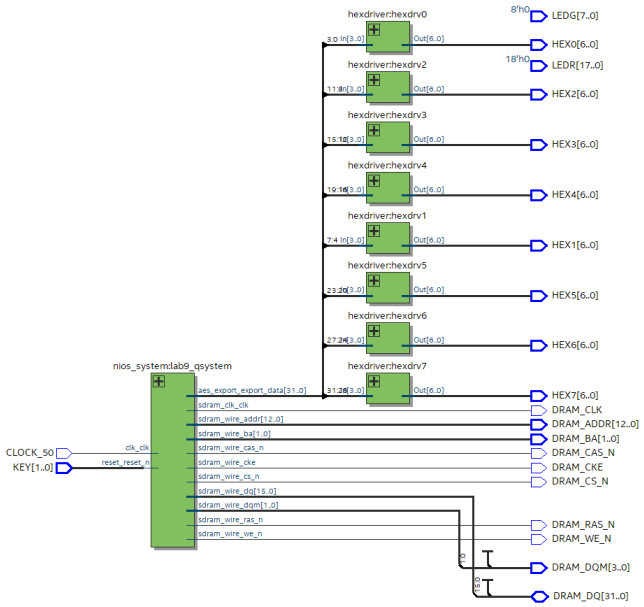
In this lab, we design the decryption part by hardware in order to learn how to speed our task by hardware accelerator. The process is very similar to encryption part and we are given the basic process: AddRoundKey, InvShiftrows, InvSubBytes and InvMixColumn. We instantiated all these modules in our AES.sv together with a state machine to control the algorithm flow. Noticed all the above function can be achieved just in one clock cycle, and for convenience, we named first AddRoundKey as AddroundKey0 and the looped one AddroundKey1 and the final one AddroundKey2, they are just a little bit difference in the next coming state and the loop one has a iterator to count loop times while other two don't have. Like this, we named looped InvShiftrows, InvSubBytes as InvShiftrows0 and InvSubBytes0 while the last two InvShiftrows1 and InvSubBytes1. One more thing to entire the state machine is we should first form our key schedule by KeyExpansion, after simulating, we discovered it takes eleven clock cycle to finish, so we use another counter here. For the detailed part, a loop mux will decide while column of key schedule will be used for AddRoundKey taken loop number as its selection signal. Both AddRoundKey, InvShiftrows, InvSubBytes and InvMixColumn are connected to a function result mux choosing which output to be given due to what process we are now given under the control of state machine.

2.3. Written description of interface

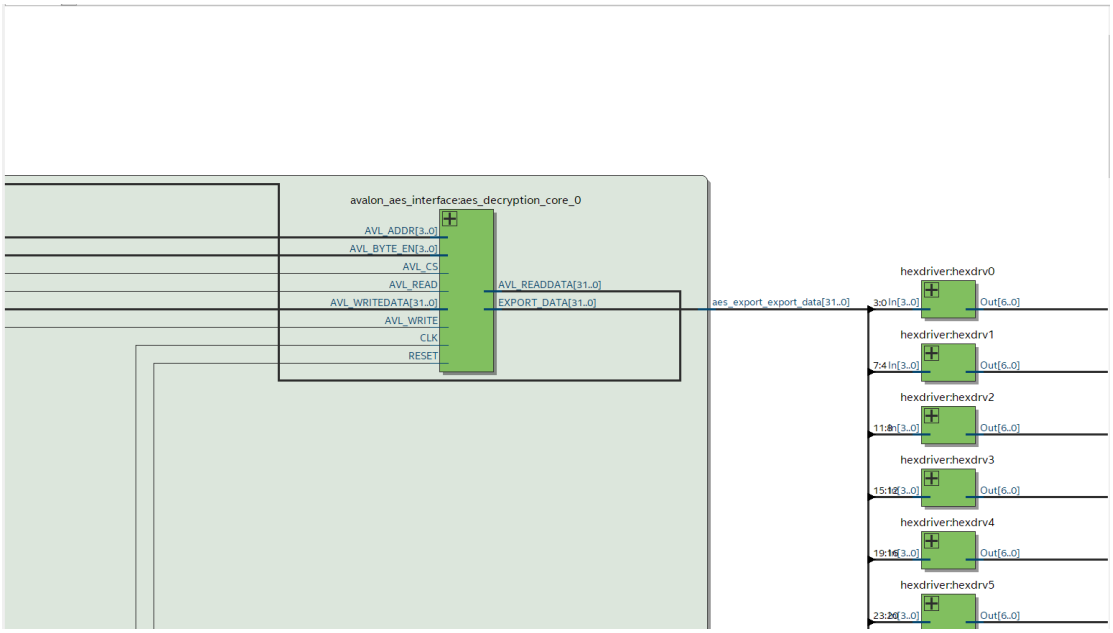
2.3.1. Describe how the system sends data between NOIS and the hardware decryptor and how the register file is designed.

In this lab, we achieved the data transfer in the module Avalon_aes_interface.sv. Here we form 16 32-bit register, with the first four store AES key after NOIS finish encryption, and the encrypted message are stored at the five to eight registers, the nine to twelve registers are used for decrypted message given by decryption logic, thirteen and fourteen are empty with fifteen stores start signal and sixteen stores done signal.

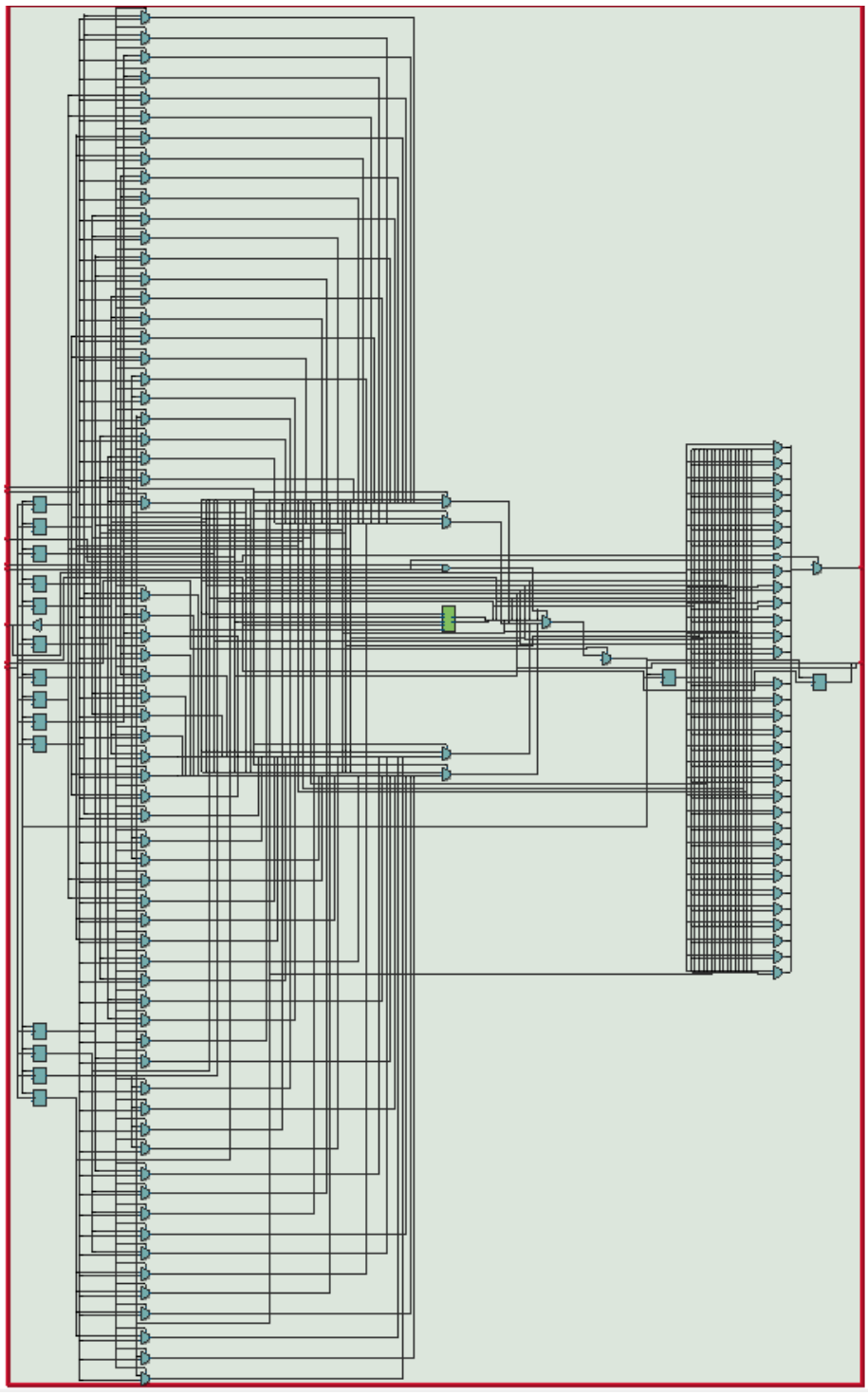
2.3.2. Block diagrams



top level diagram

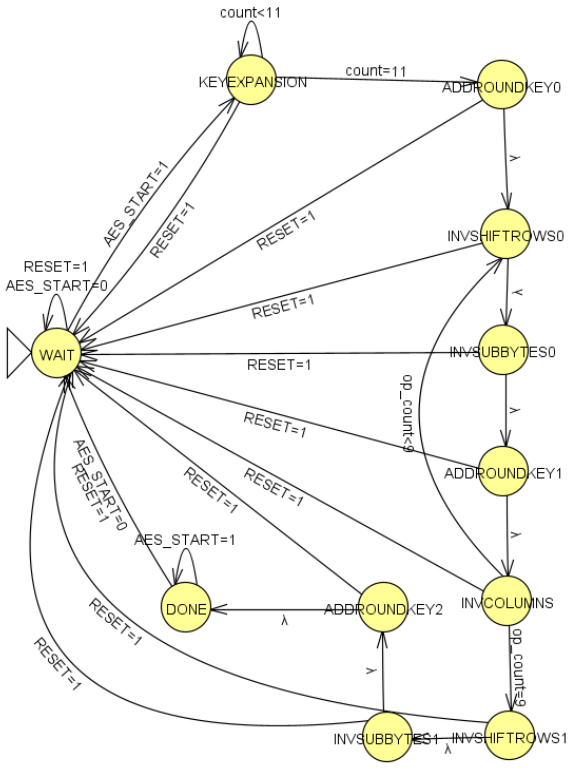


Avalon_aes_interface.sv



Avalon_aes_interface.sv details

2.3.3. State Diagram of AES decryptor controller



3. Module Description

Module: InvAddRoundKey.sv

Inputs: [127:0] state, roundkey

Outputs: [127:0] result

Description: do XOR to input and assignment result to output

Purpose: used as one step in decryption process

Module: SubBytes.sv

Inputs: clk, [7:0] in

Outputs: [7:0] out

Description: transform Byte according to one to one table achieve as a mux

Purpose: used as one step in decryption process, transform byte

Module: lab9_top.sv

Inputs: CLOCK_50, [1:0] KEY

Outputs: DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK, [7:0] LEDG, [17:0] LEDR, [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM

INOUT: [31:0] DRAM_DQ

Description: This is the top level of lab9, connect all the wires between hexdriver and sdram

Purpose: my Qsys system are instantiated here and key data are sent to hexdrivers to be display on board

Module: KeyExpansion.sv

Inputs: [127:0] data_in

Outputs: [127:0] data_out

Description: this module will generate the roundkey we used in decryption process

Purpose: we totally call AddRoundKey 11 times and all the roundkey with each 32 bits are generated together here

Module: InvShiftRows.sv

Inputs: CLOCK_50, [1:0] KEY

Outputs: DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK, [7:0] LEDG, [17:0] LEDR, [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM

INOUT: [31:0] DRAM_DQ

Description: This is the top level of lab9, connect all the wires between hexdriver and sdram

Purpose: my Qsys system are instantiated here and key data are sent to hexdrivers to be display on board

Module: InvMixColumns.sv

Inputs: [31:0] in

Outputs: [31:0] out

Description: This module accepts 32 bits data and modified it by multiplicand a square matrix

Purpose: used as one step in decryption process, modified 32 bits input and output 32 bits result

Module: HexDriver.sv

Inputs: [3:0] In0

Outputs: [6:0] Out0.

Description: This module achieves a mux to exchange output of our circuit to 7-bit display signal.

Purpose: This module helps to light the 7 bits segment display on the FPGA board.

Module: avalon_aes_interface.sv

Inputs: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, [3:0] AVL_BYTE_EN, AVL_ADDR, [31:0] AVL_WRITEDATA

Outputs: [31:0] AVL_READDATA, [31:0] EXPORT_DATA

Description: This module achieves a 16 32-bit register file to store key, encrypted message from nois and decrypted message from decryption hardware, as well as aes_start and aes_down signal.

Purpose: This module is the interface between hardware and software which control the data flow

Module: AES.sv

Inputs: CLK, RESET, AES_START, [127:0] AES_KEY, [127:0] AES_MSG_ENC

Outputs: AES_DONE, [127:0] AES_MSG_DEC

Description: This module describes the whole process of decryption, using a state machine to control the data flow

Purpose: all the decryption functions are instantiated and organized here to perform the decryption

Module: nios_system.v

Inputs: clk_clk, reset_reset_n,

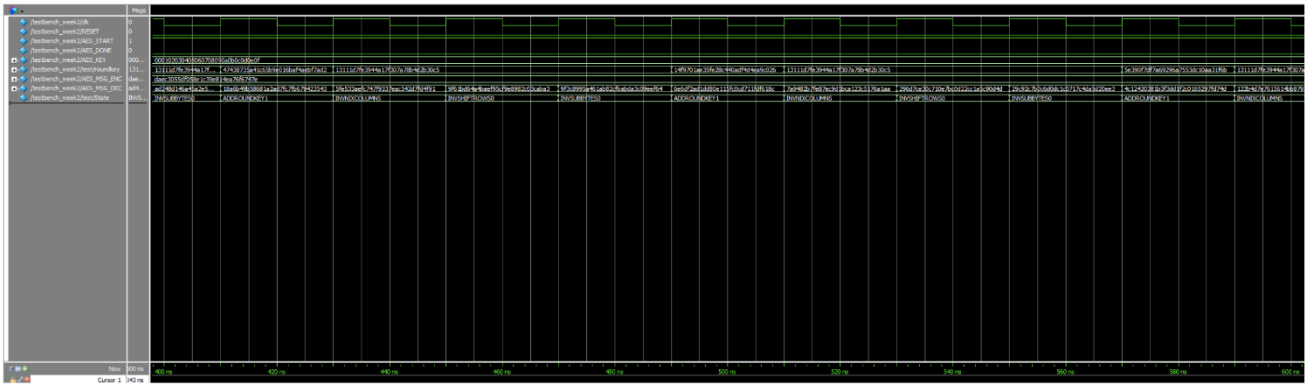
Outputs: sdram_clk_clk, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, sdram_wire_ras_n, sdram_wire_we_n, [31:0] aes_export_export_data, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, [1:0] sdram_wire_dqm

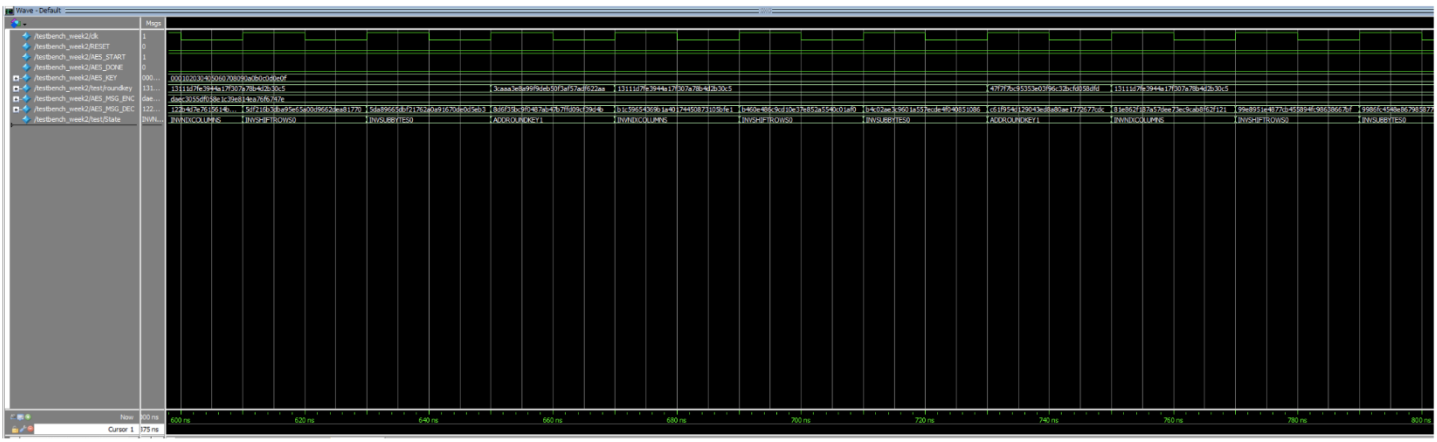
INOUT: [15:0] sdram_wire_dq

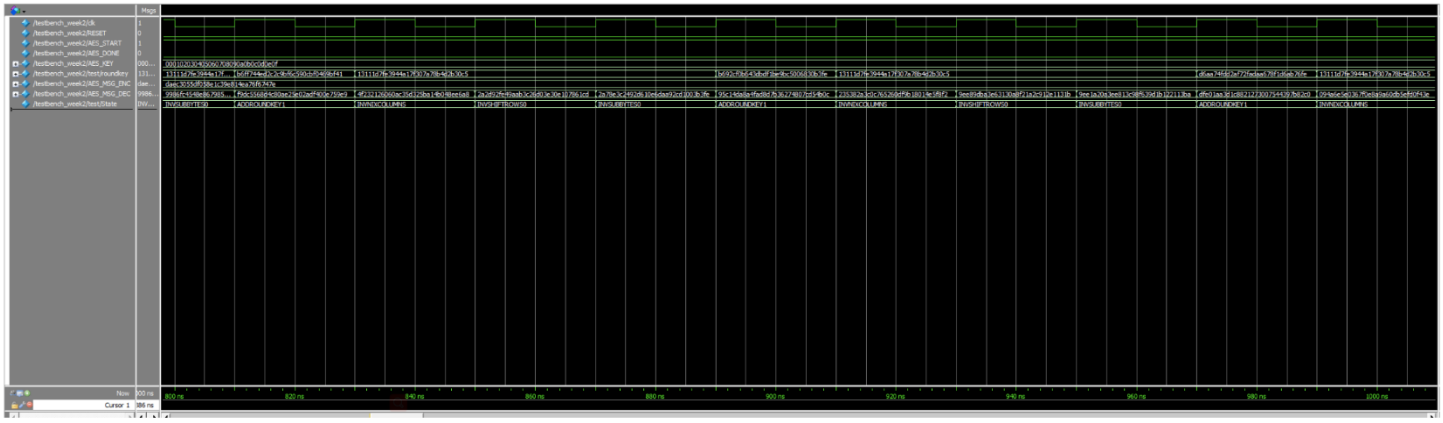
Description: This module is generated by platform designer automatically; it describes what we have added to connect the Avalon bus

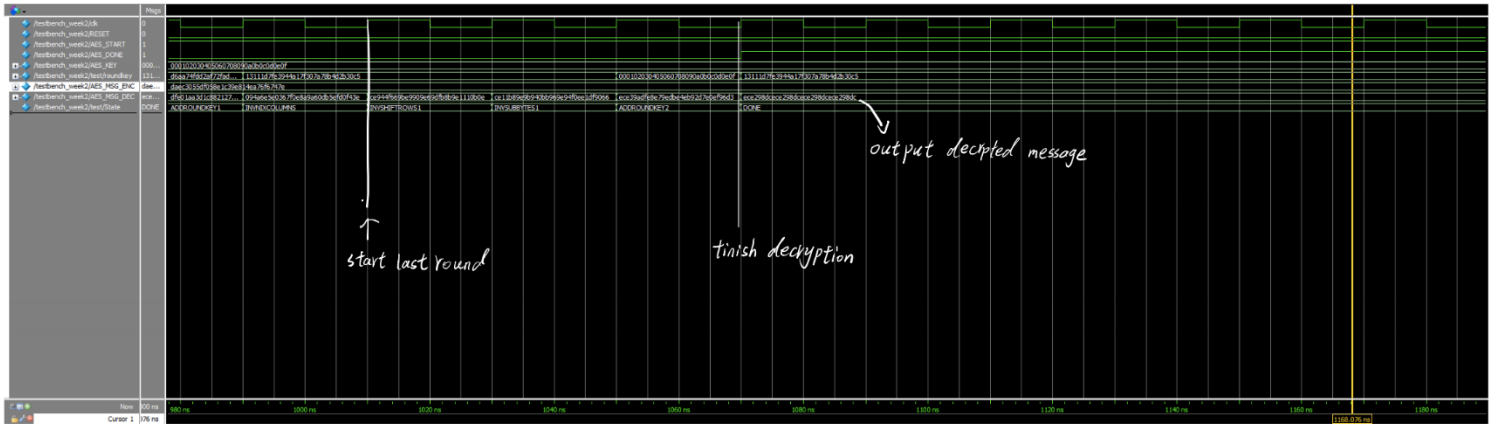
Purpose: our NOIS CPU control data through Avalon bus, module avalon_aes_interface are achieved as a IP core and added to the Avalon bus here and except this, this module contain all accelerator and controller we used

4，Annotated Simulation of the AES decryptor








output decipled message
start last round
finish decryption

5,Post-Lab Questions

LUT	2215
DSP	0
Memory (BRAM)	36864
Flip-Flop	1840bits
Frequency	52.4 MHz
Static Power	102.52 mW
Dynamic Power	72.36 mW
Total Power	243.69 mW

Problem encountered

1, We forgot to reverse the key schedule when dealing with decryption.

2, When creating custom IP, we find that we have to keep the port name consistent.

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)

hardware， yes. C program need to communicate with memory, but our hardware part can just visit registers directly. It is clearly that decryption time will much smaller than encryption time.

Encryption benchmark: 0.4940kb/s

Decryption benchmark: 100.00kb/s

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

Since we already have key schedule when doing encryption. When doing encryption, we can just store it to PIO. In this case, when doing decryption, we do not need to expand the key.

Also, we can just raise the clock to its top constraint and use NIOS-II/f.

The last one is that we can create four InvMixColumn modules. So that they can run in parallel.

6,Conclusion

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 have two parts. The software part is written in C and can accept a plaintext with key and encrypts plaintext base on the key. Another part is the hardware part. It gets the start signal, ciphertext and key from software part and do the encryption. After hardware part finishes its job, Software part can get the results. To summarize this lab, we gain the experience of construct an "Hardware accelerator" . So, if we have some tough and large task to deal with, we can let hardware to do it and return the results.

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.

It is quite clear. However, if the manual can introduce more about the mathematic background, it will be easier for us to understand.