ECE385

Fall 2020

Experiment 9

SOC With Advanced Encryption Standard in SystemVerilog

Name: Zhou Qinren
Zhang Yichi
Lab Section: LA3
TA's Name: Yu Yuqi

Dec. 8^{th} 2020

1 Introduction

In this experiment, we implemented a 128-bit advanced encryption standard using software as an intellectual property core. We also implemented a corresponding decryption module. The AES encryptor receives a plain sequence and encrypts it and stores the encrypted information in some registers. The decryptor fetches the encrypted code in the registers and then decrypts it.

2 Written Description and Diagrams of the AES encryptor/decryptor

2.1 Software Encryptor

The NIOS receives the plain information via console, and then encrypts it and store the encrypted information into some specific registers. The C code consists of some encryption help functions and a main function. The user enters a plain message using console, then the C code will store the message into an array of unsigned char. Then the user inputs the key, which will also be stored into an array. After that, the C code will generate the key expansion of the input key, and then perform the encryption. The process includes AddRoundKey, SubBytes, ShiftRows and MixColumns. AddRoundKEy takes the current message and a set of key as input, and performs an XOR operation. SubBytes substitute the current information with the information in a look-up table. ShiftRows shifts each row in the updating state by some offsets. MixColumns transforms the state via a separate invertible linear field. After the message is encrypted, the C code will store the encrypted message into some specific registers addressed by the pointer AES_PTR.

2.2 Hardware Decryptor

The hardware consists of a register file and a controller. The register file is used to store the key, encrypted message, decrypted message, and control signals. The controller takes charge of the decryption process. The controller is essentially an FSM, and when the control signal Run is high, the controller starts the decryption, and go through many states. In our design, the FSM includes 7 states: Stop, KesExpansion, AddRoundKey, InvShiftRows, InvSubBytes, InvMixColumns and End. Each state does exactly what its name suggests, and some states also checks some other condition, so as to follow the right flow.

2.3 Hardware/Software Interface

The avalon_aes_interface module is the interface that connects the software and hardware. It contains a register file of 16 32-bit registers. 0-3 stores the key, 4-7 stores the encrypted message, 8-11 stores the decrypted message, 14 stores the Start signal, and 15 stores the Done signal. NIOS sends data to the hardware using a pointer AES_PTR. The pointer points to the base address of the hardware. When sending data, we have to specify the relative address of the registers, e.g., AES_PTR[2] = something. For the hardware to send data to NIOS, it works similarly, e.g., some variable = AES_PTR[2].

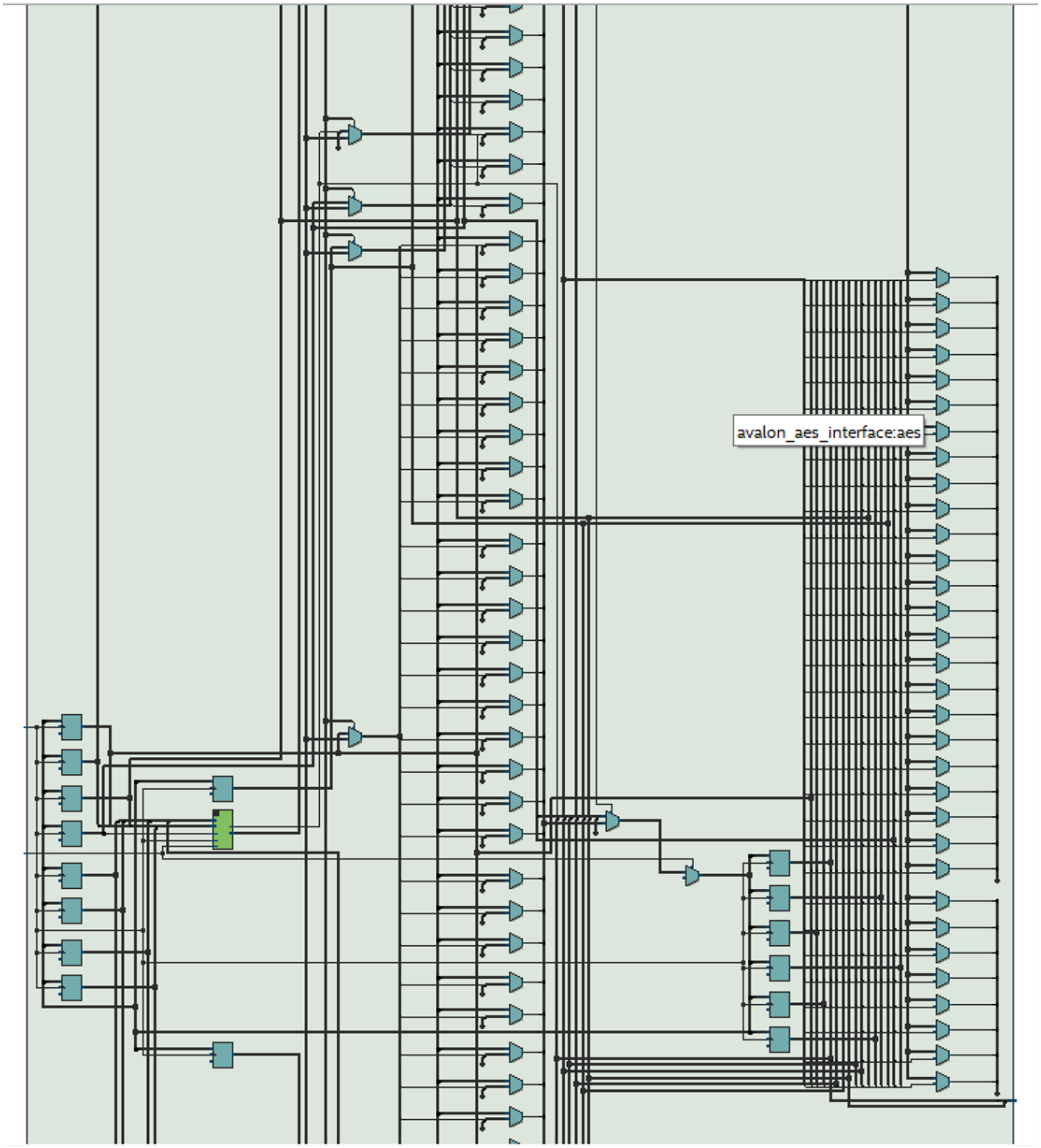


Figure 1: RTL view of avalon_aes_interface.

The original graph is too large, we only captured the most complex part of it. The rest part is similar but simpler.

2.4 Block Diagram

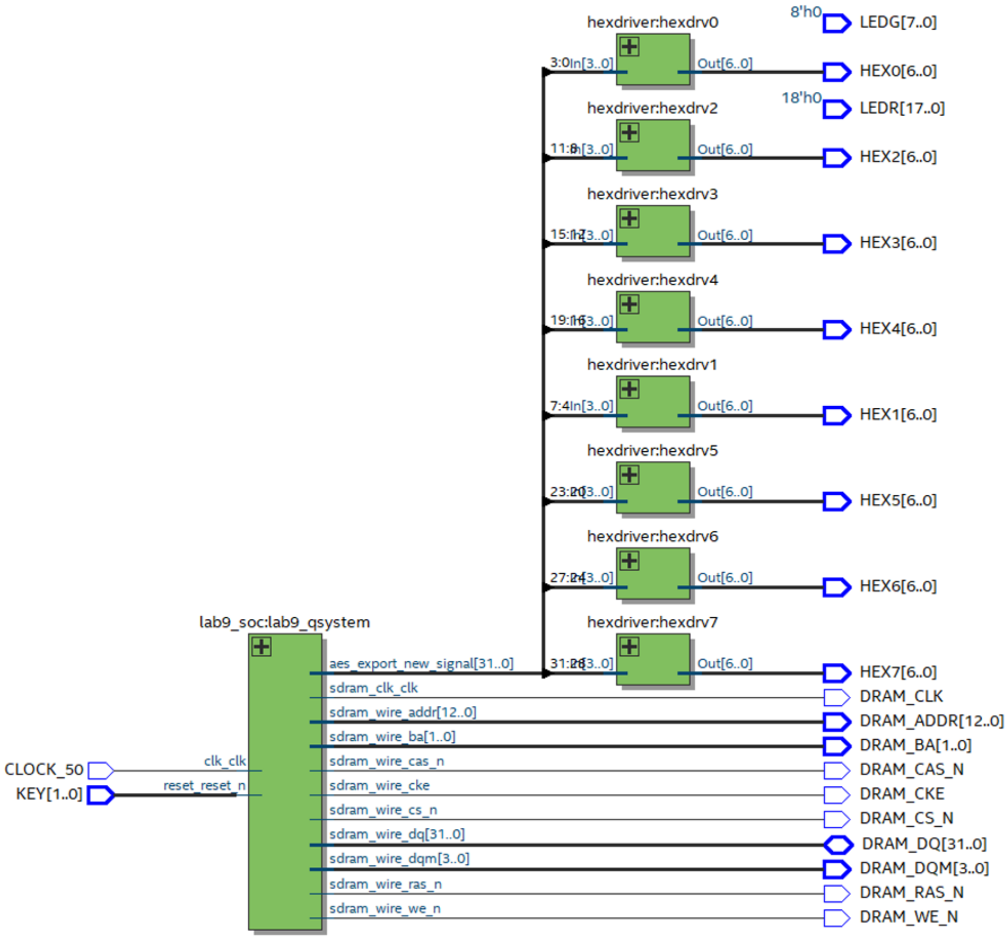


Figure 2: Top level block diagram.

2.5 State Diagram of AES Decryptor Controller

The first FSM is the outer state machine, it takes Start and Reset as input signals. If Start is high, then it goes to Execute state and triggers the inner FSM to start. When the decryption completes, the FSM goes to Done state, and outputs the Done signal.

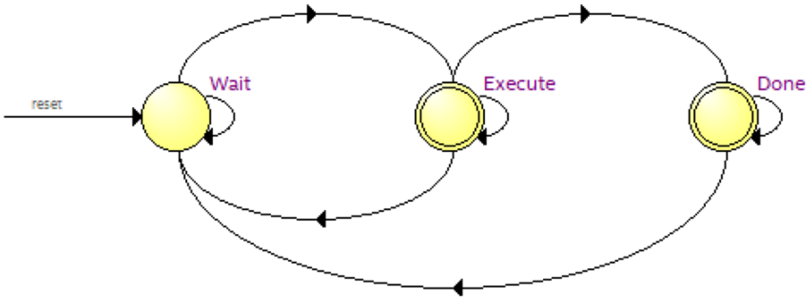


Figure 3: Outer FSM.

The inner FSM is the whole process of decryption. It stays in Stop state until the outer FSM triggers. Then it goes to many states to go through the decryption state.

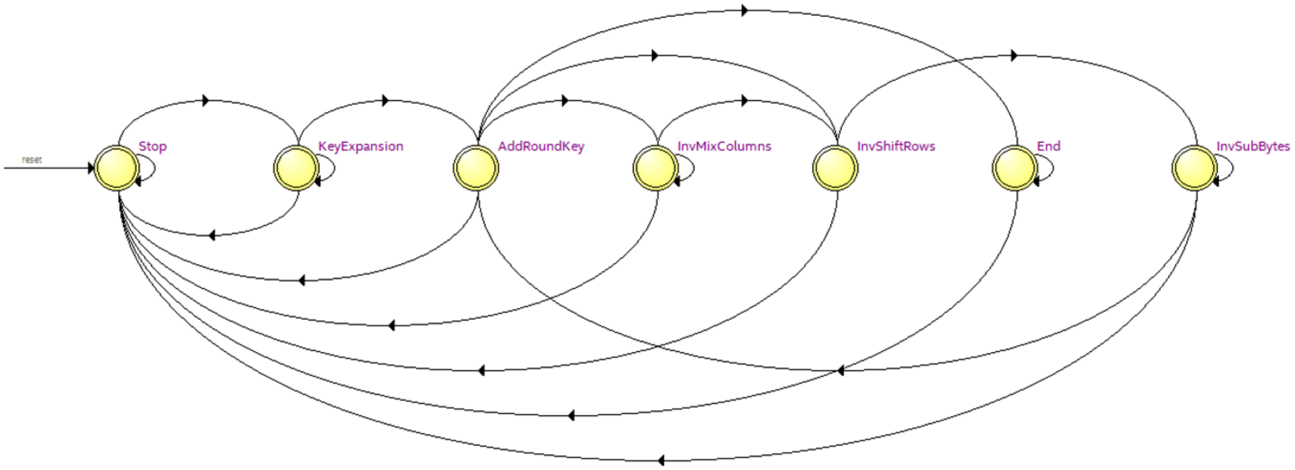


Figure 4: Inner FSM.

3 Module Description

Module: KeyExpansionOne
Inputs: clk, [127:0] oldkey, [7:0] Rcon
Outputs: [127:0] newkey
Description: It calls SubBytes and search the LUT Rcon to calculate a 128-bit key.
Purpose: It takes in an old key and outputs a new key for next AddRoundKey.

Module: KeyExpansion
Inputs: clk, [127:0] Cipherkey
Outputs: [1407:0] KeySchedule
Description: It constantly calls KeyExpansionOne.

Purpose: It calculates all 11 keys needed and combines and outputs them as a 1407 bits KeySchedule.

Module: SubBytes
Inputs: clk, [7:0] in
Outputs: [7:0] out
Description: This is a 16 × 16 LUT.
Purpose: It helps us to convert a 8-bit input to a specific byte for encryption.

Module: InvSubBytes
Inputs: clk, [7:0] in
Outputs: [7:0] out
Description: This is a 16 × 16 LUT.
Purpose: It helps us to convert a 8-bit input to a specific byte for decryption.

Module: InvSubState
Inputs: clk, [127:0] in
Outputs: [127:0] out
Description: It constantly calls InvSubBytes.
Purpose: It helps us to convert a 128-bit input to specific 16 bytes for decryption.

Module: InvShiftRows
Inputs: [127:0] data_in
Outputs: [127:0] data_out
Description: Consider and store the 128-bit input as a 4 × 4 matrix, each element is 2 bytes. This module keeps the first row unchanged, circular right shift the second row once, circular right shift the third row twice and circular right shift the fourth row three times.
Purpose: This is a step in decryption.

Module: InvMixColumns
Inputs: [31:0] in
Outputs: [31:0] out
Description: Search the GF_MUL LUT to convert a 32-bit input to 4 specific bytes.
Purpose: It does inverse mix columns for one column.

Module: AddRoundKey
Inputs: [127:0] in, [1407:0] keys, [3:0] round
Outputs: [127:0] out
Description: It calculates the output of AddRoundKey according to the round the FSM is in.
Purpose: This is a step in decryption.

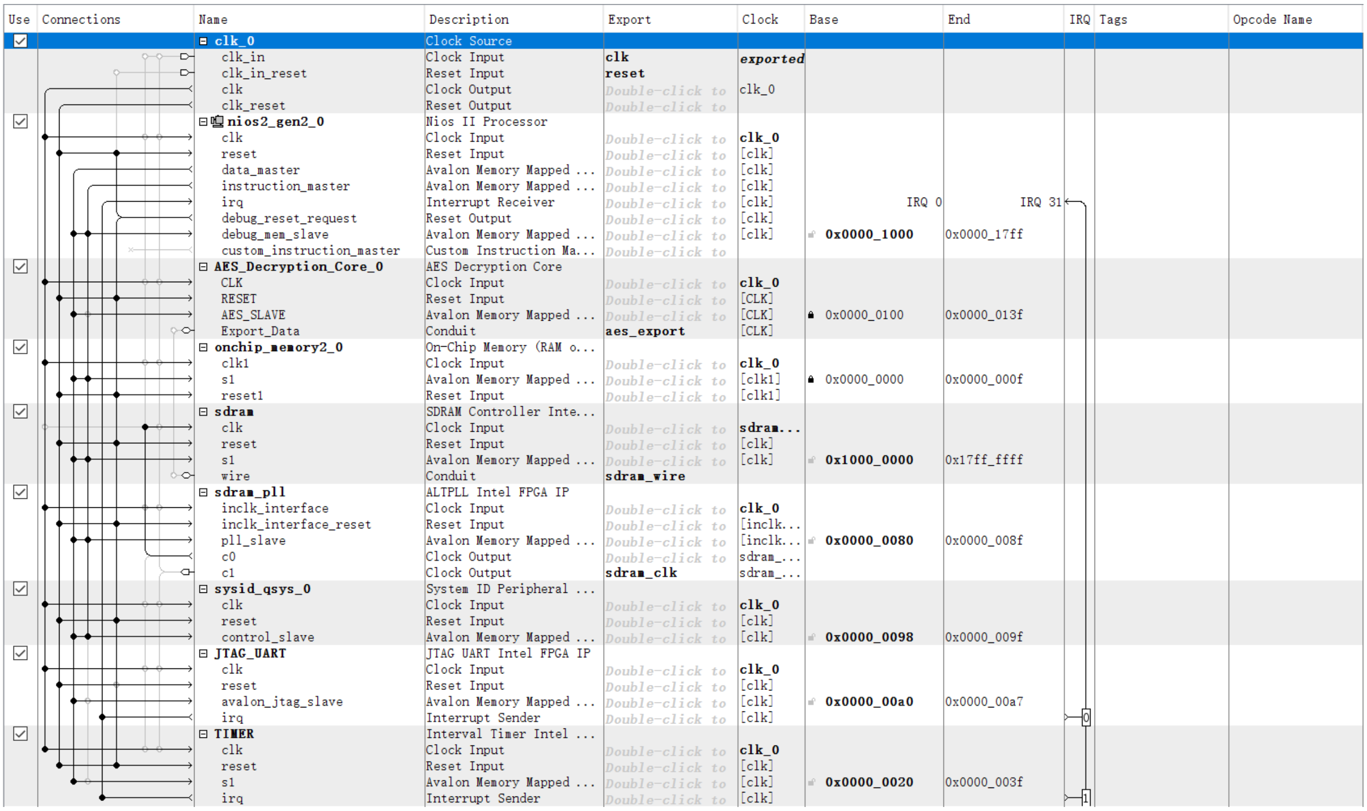
Module: counter
Inputs: CLK, RESET, in
Outputs: [3:0] out
Description: A counter. Every time the in signal is high, we add 1 to the counter.

Purpose: This can be the counter counting 10 rounds for KeyExpansion and counting 9 rounds of InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns.

Module: AES
Inputs: CLK, RESET, AES_START, [127:0] AES_KEY, [127:0] AES_MSG_ENC
Outputs: AES_DONE, [127:0] AES_MSG_DEC
Description: An FSM.
Purpose: This is the FSM for decryption. It takes in the encrypted message, calls the helper modules and outputs the decrypted message.

Module: avalon_aes_interface
Inputs: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, [3:0] AVL_BYTE_EN, [3:0] AVL_ADDR, [31:0] AVL_WRITEDATA
Outputs: [31:0] AVL_READDATA
Description: It calls the AES module to do the decryption and stores the result to the SDRAM. Also, it writes the data into the SDRAM or reads the data out of the SDRAM based on the avalon signals.
Purpose: This is the interface that connects the software and hardware. The hardware writes data into SDRAM through this module and the software writes data into or reads data out of SDRAM through this module.

Module: lab9_top
Inputs: CLOCK_50, [1:0] KEY
Outputs: [7:0] LEDG, [17:0] LEDR, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [6:0] HEX4, [6:0] HEX5, [6:0] HEX6, [6:0] HEX7, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK
Inouts: [31:0] DRAM_DQ
Description: It connects the signal of all the hardwares.
Purpose: This is the top level module for lab9.



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
☑		⊟ **clk_0**	Clock Source							
		clk_in	Clock Input	**clk**	***exported***					
		clk_in_reset	Reset Input	**reset**						
		clk	Clock Output	*Double-click to*	clk_0					
		clk_reset	Reset Output	*Double-click to*						
☑		⊟ **nios2_gen2_0**	Nios II Processor							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		data_master	Avalon Memory Mapped ...	*Double-click to*	[clk]					
		instruction_master	Avalon Memory Mapped ...	*Double-click to*	[clk]					
		irq	Interrupt Receiver	*Double-click to*	[clk]	IRQ 0	IRQ 31			
		debug_reset_request	Reset Output	*Double-click to*	[clk]					
		debug_mem_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	**0x0000_1000**	0x0000_17ff			
		custom_instruction_master	Custom Instruction Ma...	*Double-click to*						
☑		⊟ **AES_Decryption_Core_0**	AES Decryption Core							
		CLK	Clock Input	*Double-click to*	**clk_0**					
		RESET	Reset Input	*Double-click to*	[CLK]					
		AES_SLAVE	Avalon Memory Mapped ...	*Double-click to*	[CLK]	0x0000_0100	0x0000_013f			
		Export_Data	Conduit	**aes_export**	[CLK]					
☑		⊟ **onchip_memory2_0**	On-Chip Memory (RAM o...							
		clk1	Clock Input	*Double-click to*	**clk_0**					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk1]	0x0000_0000	0x0000_000f			
		reset1	Reset Input	*Double-click to*	[clk1]					
☑		⊟ **sdram**	SDRAM Controller Inte...							
		clk	Clock Input	*Double-click to*	**sdram...**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	**0x1000_0000**	0x17ff_ffff			
		wire	Conduit	**sdram_wire**						
☑		⊟ **sdram_pll**	ALTPLL Intel FPGA IP							
		inclk_interface	Clock Input	*Double-click to*	**clk_0**					
		inclk_interface_reset	Reset Input	*Double-click to*	[inclk...					
		pll_slave	Avalon Memory Mapped ...	*Double-click to*	[inclk...	**0x0000_0080**	0x0000_008f			
		c0	Clock Output	*Double-click to*	sdram_...					
		c1	Clock Output	**sdram_clk**	sdram_...					
☑		⊟ **sysid_qsys_0**	System ID Peripheral ...							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		control_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	**0x0000_0098**	0x0000_009f			
☑		⊟ **JTAG_UART**	JTAG UART Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	**0x0000_00a0**	0x0000_00a7			
		irq	Interrupt Sender	*Double-click to*	[clk]			0		
☑		⊟ **TIMER**	Interval Timer Intel ...							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	**0x0000_0020**	0x0000_003f			
		irq	Interrupt Sender	*Double-click to*	[clk]			1		

Figure 5: Platform designer view.

This is the hardware platform for our lab9 design. We build this platform by deleting the otg_hpi modules and keycode module and adding AES_Decryption_core_0 and TIMER modules on the basis of the platform of lab8.

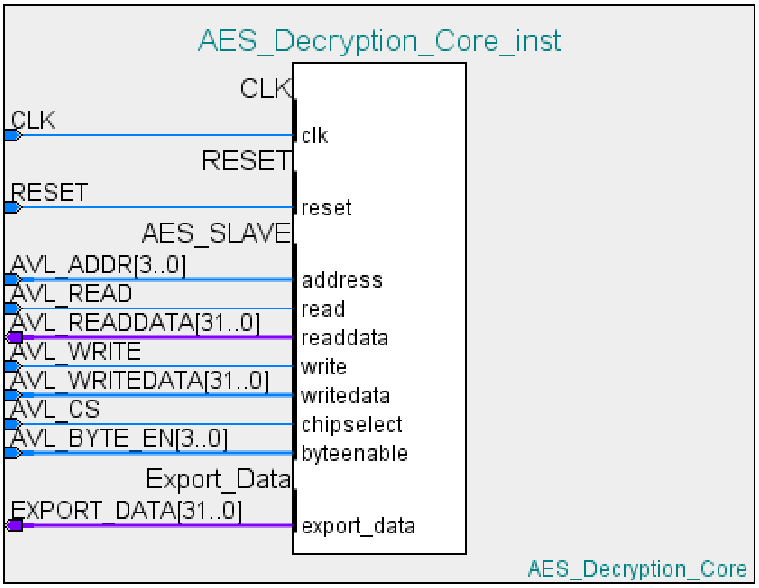


Figure 6: AES_Decryption_core_0 instance.

AES_Decryption_core_0 is an IP we made based on the avalon_aes_interface module. It is controlled by the NIOS-II through signals in Avalon-MM bus.

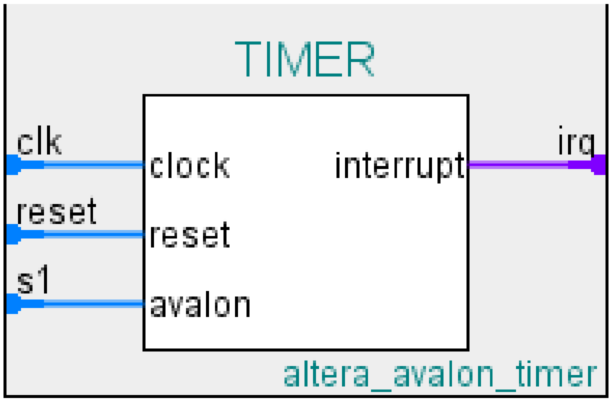


Figure 7: Timer instance.

TIMER module is used to measure the encryption and decryption time by sending periodic interrupts.

4 Annotated Simulation of the AES Decryptor

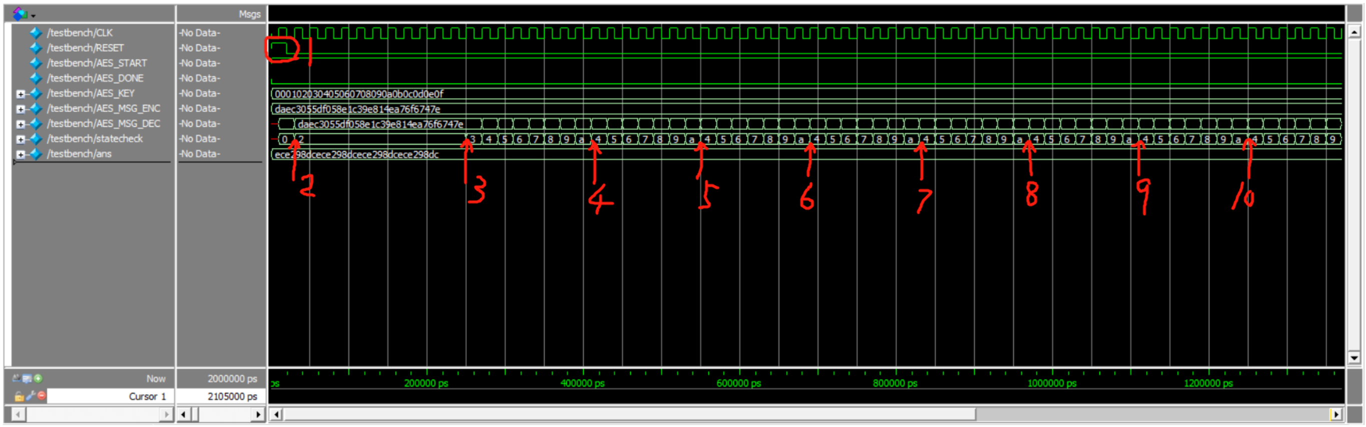

Msgs
/testbench/CLK -No Data-
/testbench/RESET -No Data-
/testbench/AES_START -No Data-
/testbench/AES_DONE -No Data-
/testbench/AES_KEY -No Data-
/testbench/AES_MSG_ENC -No Data-
/testbench/AES_MSG_DEC -No Data-
/testbench/statecheck -No Data-
/testbench/ans -No Data-
000102030405060708090a0b0c0d0e0f
daec3055df058e1c39e814ea76f6747e
daec3055df058e1c39e814ea76f6747e
0 2 3 4 5 6 7 8 9 a 4 5 6 7 8 9 a 4 5 6 7 8 9 a 4 5 6 7 8 9 a 4 5 6 7 8 9 a 4 5 6 7 8 9 a 4 5 6 7 8 9 a 4 5 6 7 8 9
ece298dcece298dcece298dcece298dc
1
2
3
4
5
6
7
8
9
10
Now 2000000 ps
Cursor 1 2105000 ps
ps
200000 ps
400000 ps
600000 ps
800000 ps
1000000 ps
1200000 ps

Figure 8: Annotated simulation1.

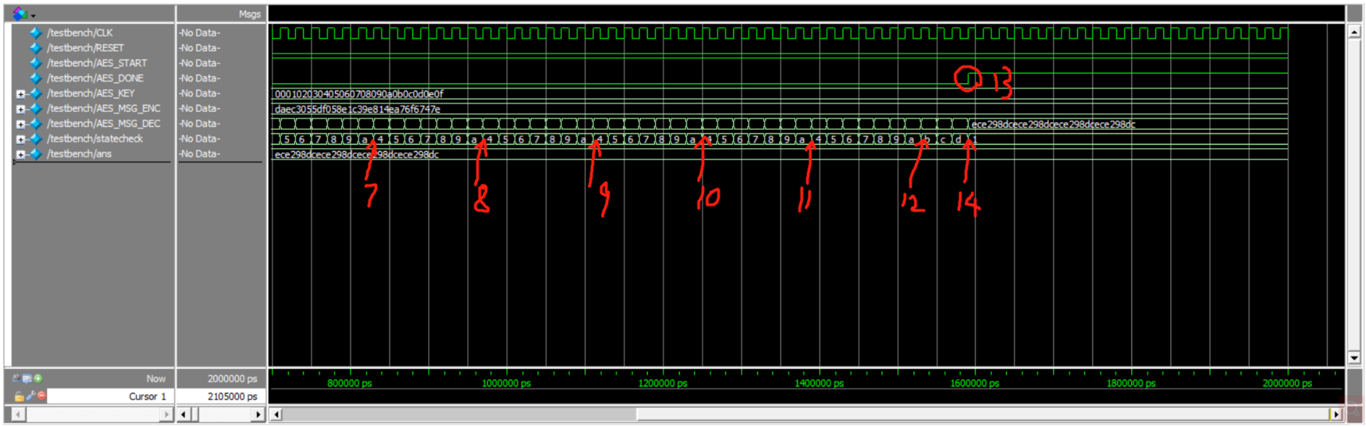


Figure 9: Annotated simulation2.

state 0: Halt
state 1: Done
state 2: KeyEx
state 3: firstAddRoundKey
state 4: nineInvShiftRows
state 5: nineInvSubBytes
state 6: nineAddRoundKey
state 7: InvMixColumns_1
state 8: InvMixColumns_2
state 9: InvMixColumns_3
state a: InvMixColumns_4
state b: lastInvShiftRows
state c: lastInvSubBytes
state d: lastAddRoundKey

The reset signal is set high at mark 1 for one cycle. Right after this cycle, at mark 2, the FSM enters the KeyExpansion state for 10 cycles where the module KeyExpansion calculate and store the keyschedule for later use. At mark 3, the FSM finishes KeyExpansion and starts to decrypt. After the firstAddRoundKey and nine rounds of InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns (from mark 3 to mark 12), the FSM then does the last InvShiftRows, InvSubBytes and AddRoundKey. Finally, at mark 13 and 14, the FSM enters the Done state where it holds the decrypted values and set the AES_DONE signal high. As we can see, the final result is the same as the ans signal where stores the real answer of decryption.

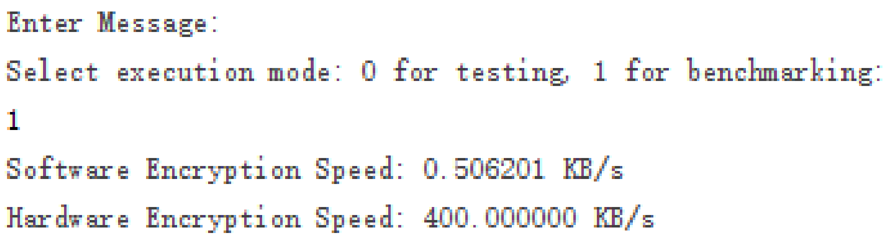
5 Post Lab Questions

LUT	5979
DSP	0
BRAM	571392
Flip-Flop	2877
Frequency	119.9MHz
Static Power	102.54mW
Dynamic Power	72.82mW
Total Power	243.48mW

Table 1: Design statistics table for the multiplier.

Question: 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)

Answer: We expect the hardware to be faster. Yes, the result approves that. We can see that the hardware is approximately 800 times faster.



```
Enter Message:
Select execution mode: 0 for testing, 1 for benchmarking:
1
Software Encryption Speed: 0.506201 KB/s
Hardware Encryption Speed: 400.000000 KB/s
```

Figure 10: Benchmark.

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

Answer: Build a complete LUT for InvMixColumns. Now we compute InvMixColumns column by column. If there is a complete LUT that takes in a 128-bit input, search the LUT and directly outputs the result, it will be much faster.

6 Conclusion

6.1 Functionality

In this lab, we first achieved encryption in software then decryption in hardware. When we built the FSM for decryption we made a mistake that we reset the state where we temporarily stored the result of decryption to be the encrypted message instead of 128-bit zeros. Later we found that when the reset is triggered the encrypted message had not been loaded. It was just some random numbers. We have to store the encrypted message to the state after the reset process. Finally, all the functions work well.

6.2 About the Lab Manual

It would be much nicer if the requirements are highlighted.