EL6463- Advanced Hardware Design

Final Project Part-3 Report

Date: 15 December 2017

Submitted By:

Anthony Fisher

Parul Raj

Sumesh Manjunath

Xinyu Ma

Prakhar Pandey

Yan Li

Xiaochen Zhang

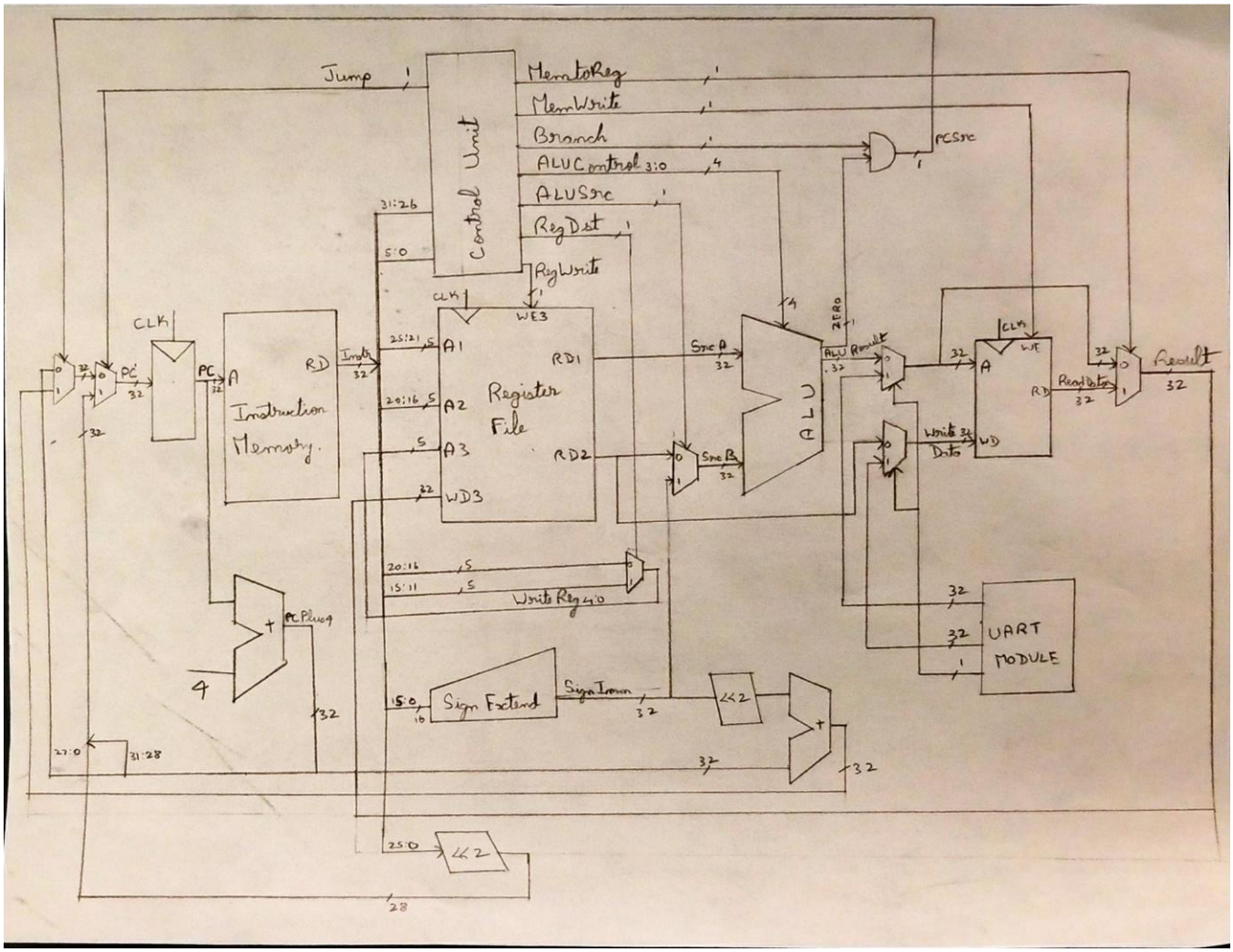
Introduction

We have created all individual components of processor in VHDL and then integrated them under a top module. We have done functional and timing simulation of individual components as well as for the integrated processor design.

In this report we will submit the following:

1. Block Diagram of our design
2. Simulation Screen Shots
3. Performance and Area Analysis of Designs
4. Description of RC5 implementation in assembly
5. Description of Processor Interfaces
6. Details about verification of overall designs
7. For the following test-vector provide the number of cycles for encryption, decryption, round key generation and clock frequency of the processor during the project demo:
   Ukey <= x"00001111 44002222 00033000 44001111"
   Din <= x"dd001234 56561111"
8. Link of demo video

Block Diagram



Instruction Memory

Instruction memory is byte addressable and the size of memory is 2047 blocks. It takes 32-bit address as input and outputs 32-bit instruction.

The instructions are divided into four parts. Rotate block, keygen block, encryption block and decryption block. The starting address of each blocks are

Block	Starting Address(Hex)	Ending Address(Hex)
Rotate	0	00000297
Keygen	00000338	000003F3
Encryption	00000298	00000337
Decryption	000003F4	000004C7

Register File

Register File contains 32 registers of 32-bit each. R[0] = 0 always.
A1, A2, A3 are addresses for read, read and write, respectively and WD3 is the 32-bit data to write into A3 address.

Data Memory

Data Memory is word addressable and the size of memory is 64 blocks. It takes 32-bit address and 32-bit data as inputs and based on WE signal, it either writes the data to that address or read the data from that address and outputs at RD.
The user inputs and the program outputs are stored in data memory and their locations are as follows

Block	Starting Block	Ending Block
Input	dMem[0]	dMem[1]
Skey	dMem[2]	dMem[27]
User key	dMem[32]	dMem[35]
encryption	dMem[30]	dMem[31]
decryption	dMem[28]	dMem[29]

UART description is given at the end of report.

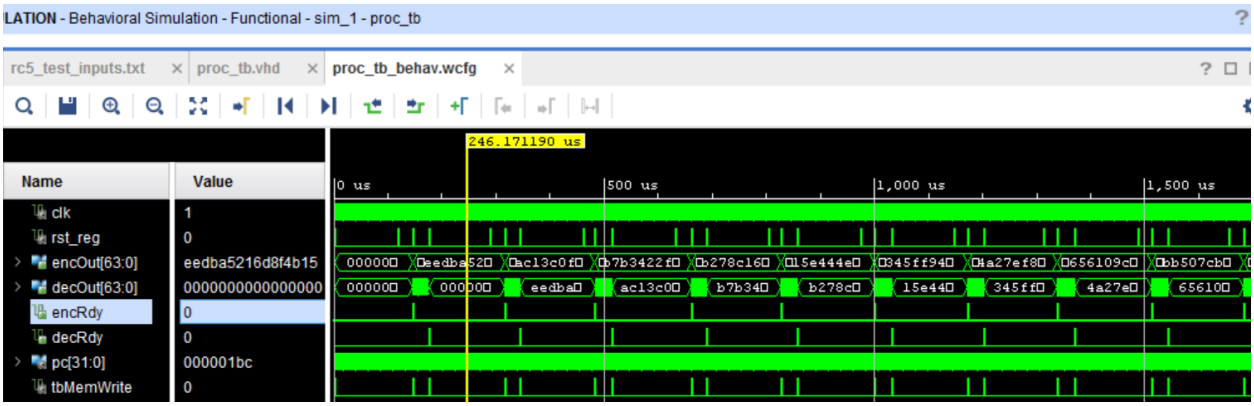
Simulation of RC5 using the processor

Functional Simulation:

Results for 10 test cases consecutively:

ukey	din	Encryption
00000000000000000000000000000000	0000000000000000	EEDBA5216D8F4B15
915F4619BE41B2516355A50110A9CE91	EEDBA5216D8F4B15	AC13C0F752892B5B
783348E75AEB0F2FD7B169BB8DC16787	AC13C0F752892B5B	B7B3422F92FC6903
DC49DB1375A5584F6485B413B5F12BAF	B7B3422F92FC6903	B278C165CC97D184
5269F149D41BA0152497574D7F153125	B278C165CC97D184	15E444EB249831DA
2AB516CF7389A2532CBDAC8F09A5261B	15E444EB249831DA	345FF9461B419FF9
D35E287696D8972EF364D8D2C73EA8A6	345FF9461B419FF9	4A27EF8548ABCB1D
E659A39507CDD7156DE3875D7A092305	4A27EF8548ABCB1D	656109CBCC75FD65
9B9323BFEE4147A30D2FCDC3E3235C9B	656109CBCC75FD65	BB507CB2FAF7F22D
3BD2F73E45C2866A606EAE5AAA10DEC2	BB507CB2FAF7F22D	7027EA2702380640

In following timing screen shot we can see **encOut** which is Encryption in above table and **decOut** is din in above table. So, every <din, Encryption> tuple is encrypted and decrypted back:

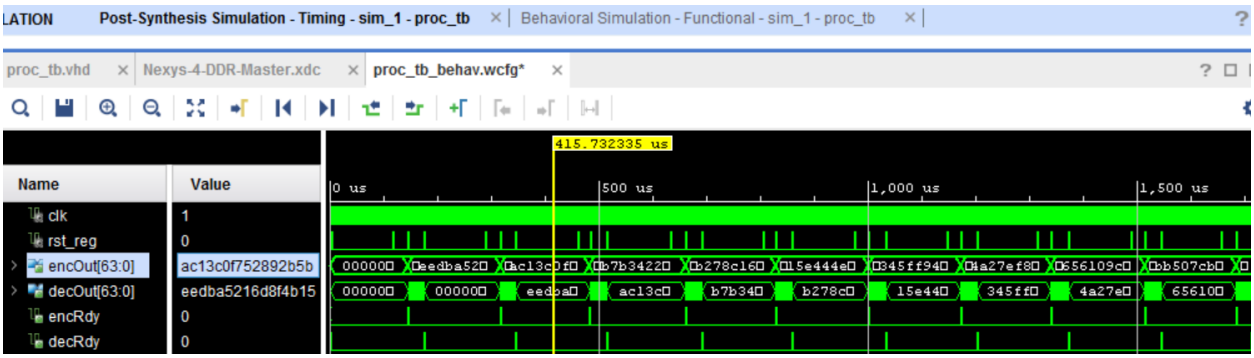


Timing Simulation:

Results for 10 test cases consecutively:

ukey	din	Encryption
00000000000000000000000000000000	0000000000000000	EEDBA5216D8F4B15
915F4619BE41B2516355A50110A9CE91	EEDBA5216D8F4B15	AC13C0F752892B5B
783348E75AEB0F2FD7B169BB8DC16787	AC13C0F752892B5B	B7B3422F92FC6903
DC49DB1375A5584F6485B413B5F12BAF	B7B3422F92FC6903	B278C165CC97D184
5269F149D41BA0152497574D7F153125	B278C165CC97D184	15E444EB249831DA
2AB516CF7389A2532CBDAC8F09A5261B	15E444EB249831DA	345FF9461B419FF9
D35E287696D8972EF364D8D2C73EA8A6	345FF9461B419FF9	4A27EF8548ABCB1D
E659A39507CDD7156DE3875D7A092305	4A27EF8548ABCB1D	656109CBCC75FD65
9B9323BFEE4147A30D2FCDC3E3235C9B	656109CBCC75FD65	BB507CB2FAF7F22D
3BD2F73E45C2866A606EAE5AAA10DEC2	BB507CB2FAF7F22D	7027EA2702380640

In following timing screen shot we can see **encOut** which is Encryption in above table and **decOut** is din in above table. So, every <din, Encryption> tuple is encrypted and decrypted back:



Performance & Area Analysis of Design

After Synthesis:

Design clock period is 20 ns.

So, minimum period = 20 -7.143 = 12.857 ns

Critical path delay = 12.857 ns

Speed of Operation = Maximum Frequency= 1/ Minimum period = 1/12.857 = 77.778 MHz

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.143 ns	Worst Hold Slack (WHS):	0.237 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	8943	Total Number of Endpoints:	8943	Total Number of Endpoints:	4246

All user specified timing constraints are met.

Area Analysis:

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
processor	2194	718	273	91	31	3

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
processor	3.46%	0.57%	0.86%	0.57%	14.76%	9.38%

After Post & Route:

Timing Analysis:

This timing analysis is for the board clock at 10ns period. The processor is running on a clock with double that period. The analysis shows that the board clock could run as fast as (10 - 1.949) = 8.051ns period, so the processor clock would have a 16.102ns period, which is a frequency of **62.1MHz.**

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.949 ns	Worst Hold Slack (WHS):	0.057 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	951	Total Number of Endpoints:	951	Total Number of Endpoints:	505

All user specified timing constraints are met.

Area Analysis:

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (1585 0)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
processor	2001	718	273	91	634	1697	304	334	31	3

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (1585 0)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
processor	3.16%	0.57%	0.86%	0.57%	4.00%	2.68%	1.60%	0.53%	14.76%	9.38%

RC5 assembly code

all_c5_assembly.xlsx is submitted on GitHub and in code as well which has all the comments and explanation with assembly code.

Convert your RC5 assembly code into machine code

We have two python files which convert assembly code to machine code:

Code_Assembly_Updated_rd_rs_rt.py → For format rd, rs, rt

Code_Assembly_rs_rt_rd.py → For format rs, rt, rd

Both files take input.txt having assembly code and write a new file output.txt with machine code. They have been uploaded with the code and on GitHub as well.

Cycles for Encryption, Decryption and Key Generation

We made a testbench to run automated test for multiple test cases. The Testbench is submitted separately with code:

For team assigned test vector **ukey<= x"00001111 44002222 00033000 44001111" and din<= x"dd001234 56561111"**:

Cipher Text <= 755EB0769316A3CB

Tests complete. 1 cases tested at clock period of 20 ns.

0 errors detected.

Average key generation time: 111445 ns

Average key generation clock cycles: 5572

Average encryption time: 26050 ns

Average encryption clock cycles: 1302

Average decryption time: 29170 ns

Average decryption clock cycles: 1458

In Behavioral, for 10, 000 test cases output is:

Tests complete. 10000 cases tested at clock period of 20 ns.

0 errors detected.

Average key generation time: 113765.103 ns

Average key generation clock cycles: 5688

Average encryption time: 25860.124 ns

Average encryption clock cycles: 1293

Average decryption time: 29962.506 ns

Average decryption clock cycles: 1498

In Post-Route, for 10 test cases output is:

Tests complete. 10 cases tested at clock period of 20 ns.

0 errors detected.

Average key generation time: 114678.24 ns

Average key generation clock cycles: 5733

Average encryption time: 25834 ns

Average encryption clock cycles: 1291

Average decryption time: 30076 ns

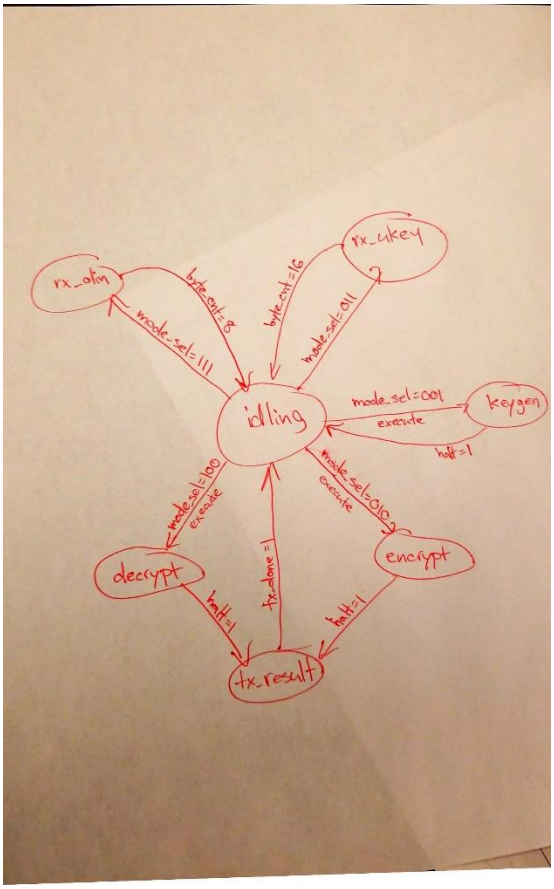
Average decryption clock cycles: 1503

Verification of Design

We verified the processor design using testbenches for individual components and the whole processor as well. Functional Simulation and Timing Simulation are giving expected results and are discussed in past project submissions. We also created automated testbench and tested 10,000 results and is discussed later in this report.

Operation Manual for RC5 using processor on FPGA and UART

The FSM for Input/Output is following:



For User Key Input: (128 bit)

Step 1: sw[14] sw[13] sw[12] <= 011

Step 2: sw[15] <= 1

Step 3: Send User key through UART

For Data Input: (64 bit)

Step 1: sw[14] sw[13] sw[12] <= 111

Step 2: sw[15] <= 1

Step 3: Send Data Input through UART

For KeyGen:

Step 1: sw[14] sw[13] sw[12] <= 001

Step 2: sw[15] <= 1

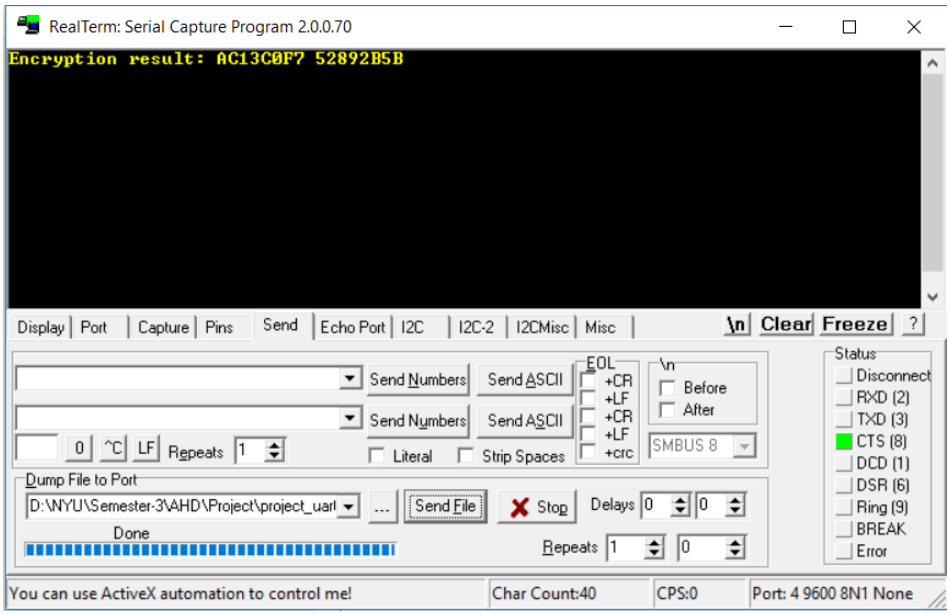
Step 3: skey[0] visible on 7-segment display

For Encryption:

Step 1: sw[14] sw[13] sw[12] <= 010

Step 2: sw[15] <= 1

Step 3: Encryption result visible on terminal (RealTerm) used as UART terminal.



For viewing skeys: (If not in STEP Mode)

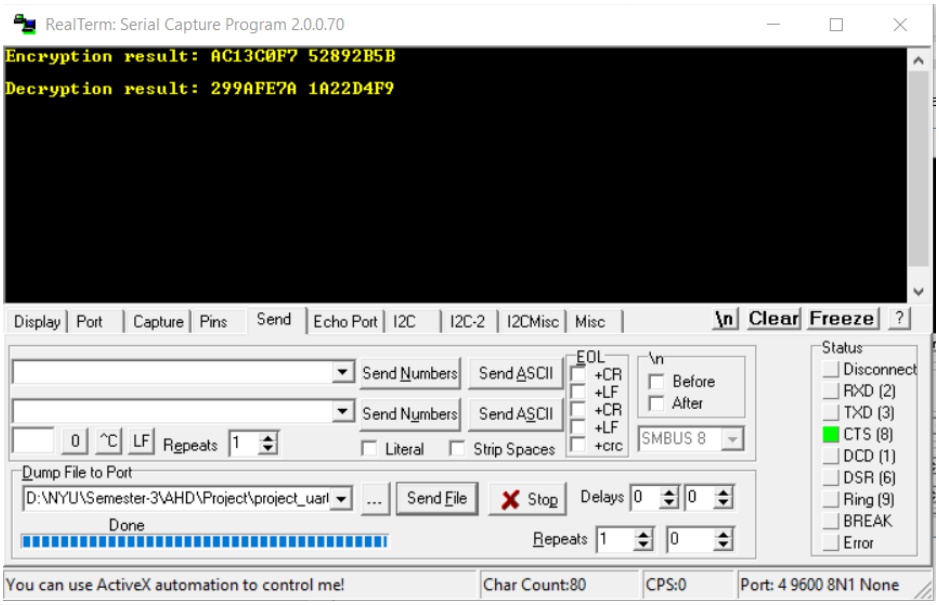
Step 1: sw[0] sw[1] sw[2] sw[3] sw[4] can be used as index of skey[0-25] array and corresponding skey value will be displayed on 7-segment display.

For Decryption:

Step 1: sw[14] sw[13] sw[12] <= 100

Step 2: sw[15] <= 1

Step 3: Decryption result visible on terminal (RealTerm) used as UART terminal.



For Step Operation:

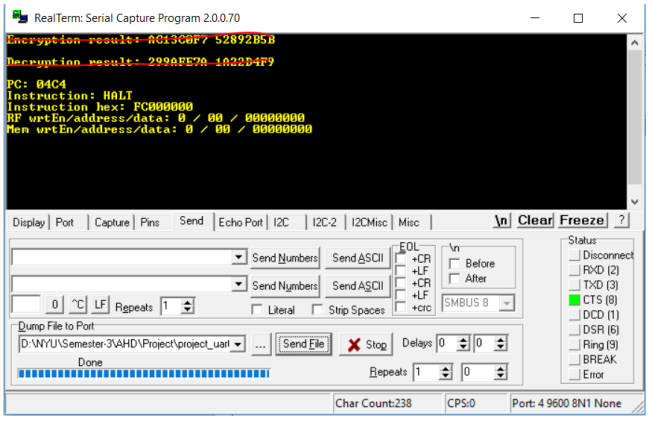
Step 1: sw[6] <= 1

Step 2: 7-segment display the PC.

Step 3: Press Center button

Step 4: 7-segment display the next PC.

Step 5: UART terminal will display following:

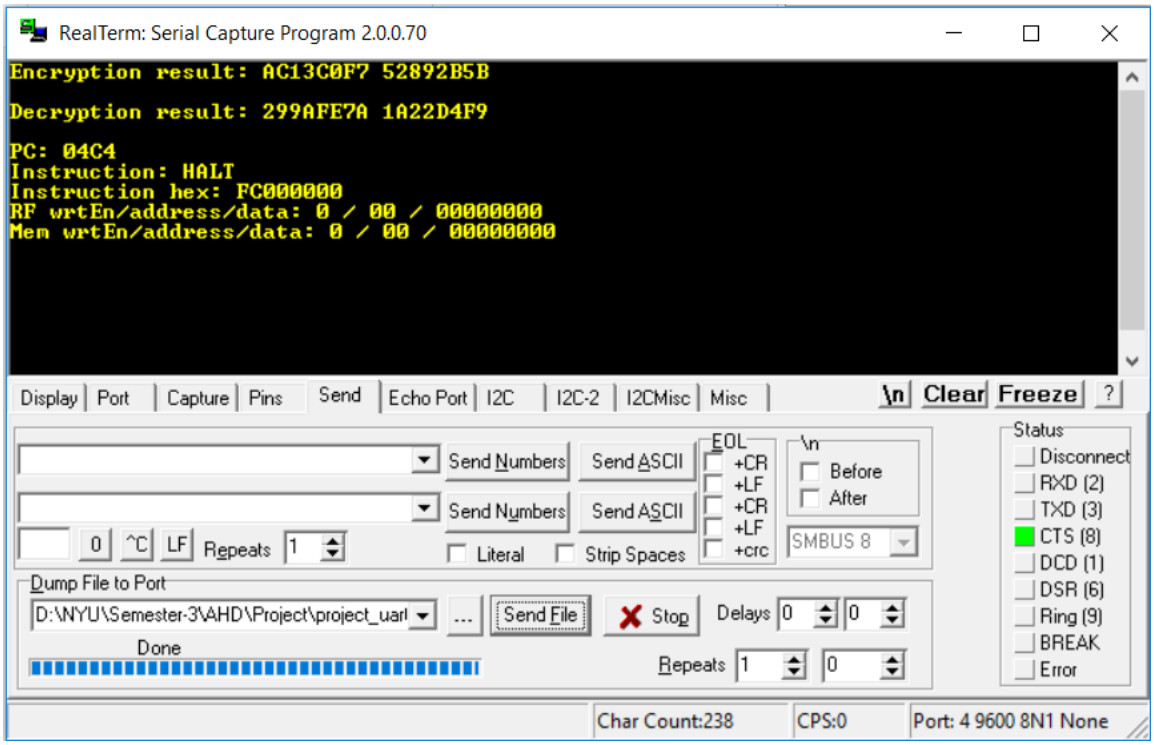


Step 6: Go to Step 3.

Extra Features

1) **UART:** For convenience, we chose to use the UART interface to display our outputs and send input data to the board. Since the input data is very long (192 bits for ukey and din combined), we thought it would be far too cumbersome to input using the 16 board switches. Instead, we can program hex files on the PC and send them to the fpga over the UART interface.

   When either of the encryption or decryption programs finishes execution, the result is displayed on the PC terminal. When the processor is in step mode, the result of each clock cycle is displayed in the terminal. The step mode data displayed is the current program counter value, the instruction in assembly language, the instruction in hex, and any value being written to data memory or the register file.



2) **Automated Testbench:** This testbench **proc_for_test_tb.vhd** is submitted separately. This file takes input as **rc5_test_inputs.txt** which contains the ukey, din and cipher text. The testbench reads the input data from the input file, runs key generation, encryption, and decryption using that data, and then compares the results with the expected values. It also keeps track of the time taken for each program and reports the average execution time and latency in the output file. This writes output in file **rc5_test_outputs.txt.** The output is generated in the format:

```
Tests complete. 10 cases tested at clock period of 20
ns.
0 errors detected.

Average key generation time: 114678.24 ns
Average key generation clock cycles: 5733

Average encryption time: 25834 ns
Average encryption clock cycles: 1291

Average decryption time: 30076 ns
Average decryption clock cycles: 1503
```

3) **Python Code to convert Assembly code to machine code:** We have two python files which convert assembly code to machine code:

   Code_Assembly_Updated_rd_rs_rt.py → For format rd, rs, rt

   Code_Assembly_rs_rt_rd.py → For format rs, rt, rd

   Both files take input.txt having assembly code and write a new file output.txt with machine code. They have been uploaded with the code and on GitHub as well.

Youtube Demo Link

---

https://youtu.be/zVQs-y0smLg

GitHub Link

---

https://github.com/prakhar1337/AHD_Group_Project_Team_3

References:

[1] Digital design and Computer Architecture by David Harris, Sarah L. Harris