ECE 3170 Cryptographic Hardware for Embedded Systems

Fall 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Lab 2, 100 pts.

Due Tuesday September 30 prior to 11:55pm

(please turn in homework electronically on Canvas)

Verifying RSA in VHDL

In this lab, you will simulate and verify a VHDL implementation of Rivest Shamir Adleman (RSA) algorithm, as well as implement RSA on your DE-10 Board.

Recall in HW2, you have chosen two prime numbers p and q. Then you calculated the encryption key "e" and decryption key "d". Please modify **tb_rsa_core.vhd** (testbench) to use the encryption and decryption keys you have calculated in HW2. You should also use the same n (n = p*q). For your plaintext, you should use each segment of the same plaintext message in HW2 (originally in step 7 we only used segment 2). This means you will do multiple encryptions and decryptions. Keep in mind that the provided VHDL code only supports 32-bits input message, 32-bits encryption key (e), 32-bits decryptions key (d), 32-bits modulus key (n). Use the following instructions to run the RSA code in ModelSim.

1. Open ModelSim
2. Once ModelSim is open, create a new project by clicking on: File --> New --> Project...
3. Give the project a name, say "rsa", and click OK.
4. After creating the project, we have to add all the VHDL files to it. Do so by clicking on: Add Existing File --> Browse...
5. Choose all the VHDL files in the directory, **except for rsa_core_top.vhd and dec_7seg.vhd**. Click Open --> OK.
6. Once you are done adding the VHDL files, click Close.
7. If you haven't edited the testbench yet, do so now. Ensure you test all segments of the message from HW2 (m = 3655802389472199)
8. Now we have to compile the design files. To do so click on: Compile --> Compile All. Check the transcript window to make sure all files were successfully compiled.
9. Now we can start the simulation by clicking on: Simulate --> Start Simulation...
10. In the Start Simulation window, make sure you are on the design tab, expand the work library, choose the testbench for this code named: "tb_rsa_core", and click OK.
11. ModelSim will change view into simulation mode and a couple of other windows show up.

12. Our next step is to add some signals of interest to a wave window to monitor their changes as simulation proceeds. Before doing so, let us create a dump file that will be storing the results of our simulation as we perform it. To do so, type the following in the command line of the "Transcript" window:
    a. vcd file sim_results.vcd
    b. vcd add tb_rsa_core/*
    c. vcd add tb_rsa_core/uut/*
13. Now let us add our signals of interest to the wave window to monitor their changes as simulation proceeds. To do so, go to the "sim" window and click on the instance named "tb_rsa_core" to add the testbench signals. Next, open the "Objects" window and choose all the signals (inputs, outputs, and internal). Right-click on the selection and click on Add Wave. Check that the signals have been successfully added to the "Wave" window.
14. Our final step is to run the simulation for a specific time. For this testbench, running the simulation for 100 us should be enough. To do so, type run 100 us in the command line of the "Transcript" window. Please double check that you are getting the expected ciphertextout and plaintextout. It may be possible that your design needs more than 100 us, so you may want to increase the simulation time.
15. Navigating back to the "Wave" window will now show you the result of the simulation for all the signals that we added. To better read the values, select all the signals and right-click, then change the Radix to decimal. Also, towards, the bottom left of the signals pane (to the left of where it says "Now"), there is a blue button that has a description of "Toggle leaf names <-> full names" if you hover the mouse over it. Click on that button to show the signal names only without the hierarchy.
16. Look through the wave window and try to understand how the signals are changing values with respect to the simulation time. Specifically, look for the output signal that shows the encrypted/decrypted value.
17. Now that we have run the simulation, make sure that you set the zoom of the wave window to a full view. To do so, right-click anywhere in the wave window and click on Zoom Full. Export an image of your simulated waveform. To do so, click on File --> Export --> Image... and save it as an image. Submit waveforms in the lab report verifying one encryption testcase and one decryption testcase of your chosen parameters from HW2. Note that you may need to export multiple images to cover all signals in the waveform.
18. Before ending the simulation, open the transcript window and verify that the no reports are generated by the testbench indicating a failure of any of the test cases.
19. Finally, to end the simulation and correctly save the results in the VCD file, click on: Simulation --> End Simulation.

Understanding the VHDL code

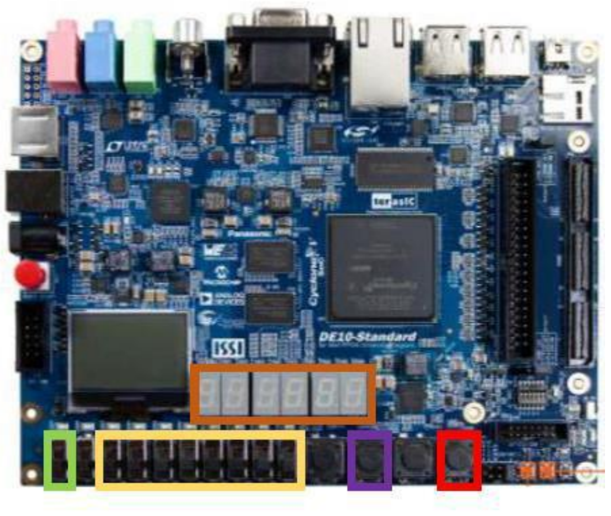
Please answer the following questions in the lab report.

Please read RSA hardware implementation, then describe the purpose of each VHDL file and how each VHDL file relates to each other. For the reading, please focus on RL binary algorithm (Section 4) and interleaving multiplication and reduction (Section 7.1). Given the two algorithms (RL binary algorithm and interleaved multiplication and reduction), please identify which algorithm is Block_2.vhd implementing. Please also identify which algorithm is Block_3.vhd implementing.

Please explain why the rsa_core does not have an input signal indicating the current mode of operation (encryption or decryption). In your answer, please explain using mathematical equations with parameters (ciphertext, plaintext, encryption key (e), modulus (n), and decryption key (d).

Implementing RSA on the DE-10 Standard

In Quartus, start a new project named "RSA". Add all VHDL files to the project except the test bench (tb_rsa_core.vhd). This project utilizes the additional files **rsa_core_top.vhd** and **seg_7dec.vhd** to use the switches on the DE-10 as shown in the picture below to implement RSA, while outputting the result to the seven-segment displays.



🟨	msgin_data(7-0)
🟧	msgout_data(23-0)
🟩	Encrypt/decrypt
🟥	msgin_valid (start encryption/decryption)
🟪	reset

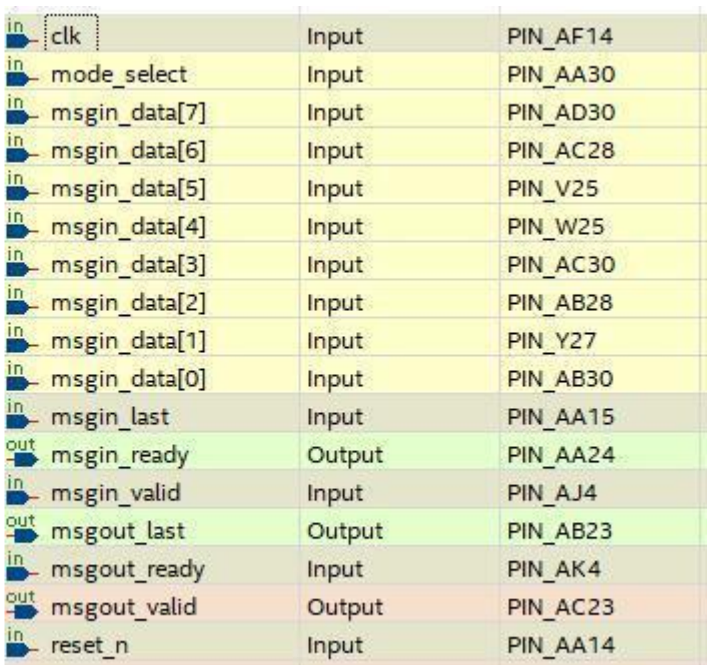
Notice that “msgin_data” and “msgout_data” are not 32 bits. The most-significant 24 bits for “msgin_data” are hardcoded into the file **rsa_core_top.vhd** as a string of ‘0’. This effectively means this circuit can only encrypt or decrypt values 8-bits or less. The switch shown in green changes which key is used as input into the circuit (e or d). Both of these keys are hardcoded before synthesis as well as your value n. When the switch is ‘1’ (“up” when viewed from the same direction as the above picture) the encryption key ‘e’ is selected.

Encryption/Decryption is conducted by pressing on the red msgin_valid button.

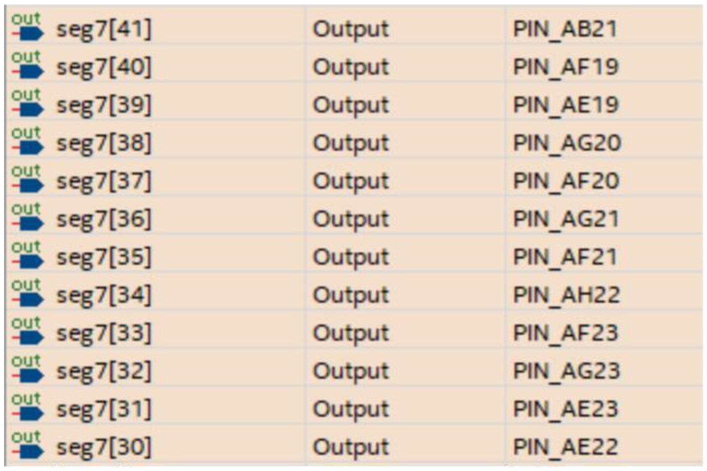
Alter the file rsa_core_top.vhd to utilize your values of e, d, and n from HW2. (**NOTE: if your value of n is greater than 8-bits, you may get an encrypted value c during operation which is impossible to enter using the input switches unless your resynthesize the project with new hardcoded input values for the most-significant 24 bits.**)

After altering rsa_core_top.vhd with your e,d, and n values, compile the design and program your board. Use the pinouts shown below, and remember to set the pin voltages to 3.3 V, and unused pins to “as input tri-stated”. These options are found in **Assignment -> Device -> Device and Pin options-> Unused Pins** and **Assignment -> Device -> Device and Pin options-> Voltage**. Take a screenshot of the resource utilization to include in the lab report.

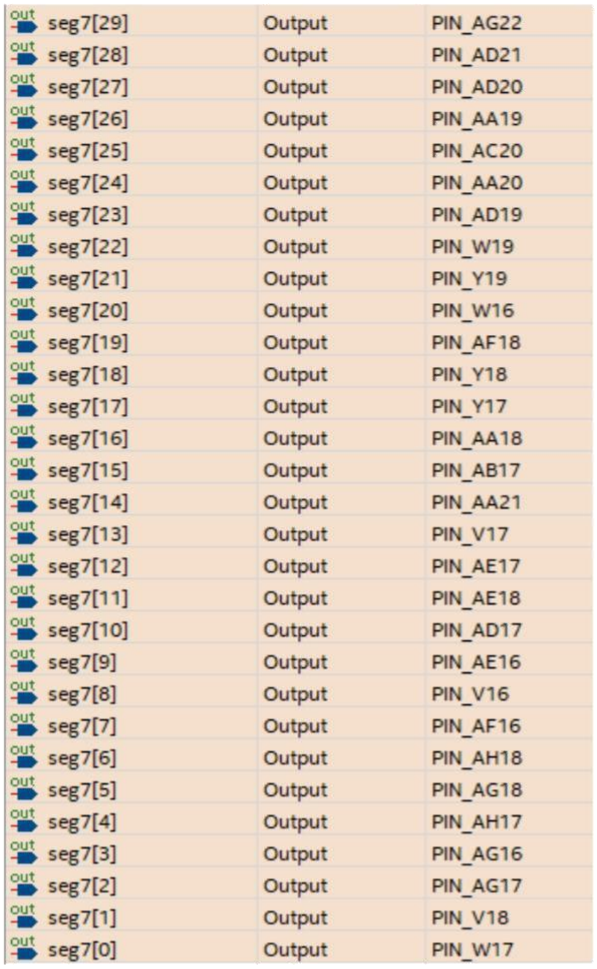
Encrypt the same segment you encrypted in HW2 on the board. Take a picture of the board showing the yellow switches set to your segment value, and the 7-segment displays showing the expected encryption value output. (The output shows the least-significant 24 bits, so even if your n is greater than 8-bits, everyone’s output e will appear correctly on the 7-segment displays).



clk	Input	PIN_AF14
mode_select	Input	PIN_AA30
msgin_data[7]	Input	PIN_AD30
msgin_data[6]	Input	PIN_AC28
msgin_data[5]	Input	PIN_V25
msgin_data[4]	Input	PIN_W25
msgin_data[3]	Input	PIN_AC30
msgin_data[2]	Input	PIN_AB28
msgin_data[1]	Input	PIN_Y27
msgin_data[0]	Input	PIN_AB30
msgin_last	Input	PIN_AA15
msgin_ready	Output	PIN_AA24
msgin_valid	Input	PIN_AJ4
msgout_last	Output	PIN_AB23
msgout_ready	Input	PIN_AK4
msgout_valid	Output	PIN_AC23
reset_n	Input	PIN_AA14



seg7[41]	Output	PIN_AB21
seg7[40]	Output	PIN_AF19
seg7[39]	Output	PIN_AE19
seg7[38]	Output	PIN_AG20
seg7[37]	Output	PIN_AF20
seg7[36]	Output	PIN_AG21
seg7[35]	Output	PIN_AF21
seg7[34]	Output	PIN_AH22
seg7[33]	Output	PIN_AF23
seg7[32]	Output	PIN_AG23
seg7[31]	Output	PIN_AE23
seg7[30]	Output	PIN_AE22



seg7[29]	Output	PIN_AG22
seg7[28]	Output	PIN_AD21
seg7[27]	Output	PIN_AD20
seg7[26]	Output	PIN_AA19
seg7[25]	Output	PIN_AC20
seg7[24]	Output	PIN_AA20
seg7[23]	Output	PIN_AD19
seg7[22]	Output	PIN_W19
seg7[21]	Output	PIN_Y19
seg7[20]	Output	PIN_W16
seg7[19]	Output	PIN_AF18
seg7[18]	Output	PIN_Y18
seg7[17]	Output	PIN_Y17
seg7[16]	Output	PIN_AA18
seg7[15]	Output	PIN_AB17
seg7[14]	Output	PIN_AA21
seg7[13]	Output	PIN_V17
seg7[12]	Output	PIN_AE17
seg7[11]	Output	PIN_AE18
seg7[10]	Output	PIN_AD17
seg7[9]	Output	PIN_AE16
seg7[8]	Output	PIN_V16
seg7[7]	Output	PIN_AF16
seg7[6]	Output	PIN_AH18
seg7[5]	Output	PIN_AG18
seg7[4]	Output	PIN_AH17
seg7[3]	Output	PIN_AG16
seg7[2]	Output	PIN_AG17
seg7[1]	Output	PIN_V18
seg7[0]	Output	PIN_W17

Analyzing the provided RSA Implementation

Answer the following questions in your lab report:

Our ability to encrypt and decrypt was limited on the DE-10 due to the small number of input switches. What is one way that the RSA implementation can be altered to operate on larger input sizes (i.e., greater than 8) while still using the DE-10 board? (Your proposed RSA alteration should be theoretically possible to implement, but you will not be graded on if it is a “good” idea or not.)

- How do you expect this to effect resource/LUT utilization of the project? How do you expect this to effect (if it does) maximum clock frequency?
- How do you expect this to effect throughput?

Additionally, comment on how the resource utilization of RSA compares to the utilization of DES from lab 1, noting the size of input data blocks of each implementation.

Required Documents to turn in

Modified **rsa_core_top.vhd**
Modified **tb_rsa_core.vhd**

Modelsim dump file **sim_results.vcd**
Lab report with

- Answers to the questions in “Understanding the VHDL code” and “Analyzing the provided RSA Implementation”
- Screenshot showing resource utilization
- Picture of your DE-10 Board after performing one encryption
- Waveforms saved from ModelSim

Please compress all files into a single folder and submit on Canvas.