**Digital Circuit Lab — Lab 2**

RSA256解密機

(RSA256 Decryptor)

Team: 06

Member: 劉容均 b05901084、吳映葦 b05901100、鄧宇凡 b05901183

Github: <https://github.com/ywwu928/DClab>

**• Introduction**

RSA, named after its designers Ron Rivest, Adi Shamir and Leonard Adleman, is a public key cryptosystem used for secure data transmission. In this lab, we will implement a digital circuit system on the FPGA board that can decrypt messages from ciphertext when given the receiver’s private key.

• **Hardware**: Altera DE2-115 FPGA board

• **Software**: Quartus II, Python 3

• **HDL (Hardware Description Language)**: System Verilog

**Part I: User Manual**

**• Usage: A step-by-step guide of using the RSA256 Decryptor**

- Clone the folder “lab2” from Github

- Install Quartus II on either Windows or Linux

- Install the USB driver

- Connect DE2-115 FPGA board to your computer by USB cable

- Program the FPGA board

1. Open Quartus II，choose project file “DE2\_115.qpf ”
2. Compile the project by pressing Ctrl + L
3. Open Tools ➝ Programmer
4. Click on “Hardware Setup”, choose the FPGA board which is connected to your computer by USB
5. Click on “Add File”, select “DE2\_115.sof ” under directory “output\_files”
6. To upload codes to FPGA board temporally, select “JTAG” mode, switch FPGA board to “run” mode, and click “Start” button in Programmer to start uploading
7. To upload codes to FPGA board permanently, select “Active Serial Programming” mode, switch FPGA board to “Program” mode, and select file “DE2\_115.pof ” to upload

* To convert .sof files into .pof files, simply click on File ➝ Convert Programming Files

- Install Python 3 from the internet

* Remember to set up paths (路徑) and environmental variables (環境變數) in order to run python program in the Command Line (命令提示字元)

- FPGA board and computer operation

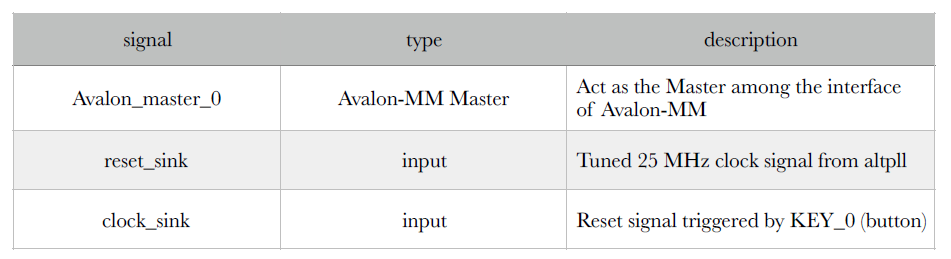
1. Connect DE2-115 FPGA board to your computer through RS232 ↔ USB transmission cable
2. Rename the binary ciphertext file you want to decrypt “enc.bin” and put it under the directory lab2\pc\_python
3. Go to My Computer (我的電腦)→right click on your mouse(按滑鼠右鍵)→content (內容)→Device Manager (裝置管理員)→連接埠(COM和LPT) and check the port section (COM? ) assigned to Prolific USB-to-Serial Common Port
4. Open Command Line from the start menu (開始功能表)
5. Change directory to lab2\pc\_python
6. Press the reset button “KEY0” for initialization purpose
7. Type “python rs232.py [name of your port (ex: COM?)]” in Command Line and press Enter
8. The decrypted binary plaintext file “dec.bin” will appear under the directory lab2\pc\_python

**Part II: Tutorial**

**• Introduction of Protocols and Tools**

- **Avalon-MM (Avalon Memory Mapped) interface**:

Avalon-MM interface is a bus interface set up by Altera. By observing its specifications, we can easily set up communication between our own modules and other predefined modules. Avalon-MM interface can be classified into two categories: Master and Slave. Master components can send I/O request to the bus, while Slave components only perform passive read/write operation. Note that when creating new components, you must set the signal interfaces and signal type properly so that it can work correctly. For further description of the signals used in Rsa256Wrapper of this lab, please refer to the table below.



- **Qsys**:

Qsys is a system integration tool built in Quartus II. We use Qsys to wrap up our designs and set up connections between modules. The modules used in this project are described below.

►**ALTPLL**

ALTPLL is a clock rate conversion tool provided in Qsys. In this lab, we convert the original clock rate from 50 MHz to 25 MHz and distribute to other modules.



►**UART**

UART (Universal Asynchronous Receiver/Transmitter) serves as a communicator between serial data and parallel data. It converts data to serial or parallel form to help communication between devices. In this lab, we use RS232 as the serial data transmit interface. UART convert the RS232 serial signals into parallel form and send it to Rsa256Wrapper to continue on the decrypting progress.



**• System Verilog Files**

- **Rsa256Wrapper.sv**:

A user-defined component created in this project in order to wrap up the Rsa256 decoder circuit and provide I/O signals that agree with Avalon-MM interface.

- **Rsa256Core.sv**:

Given three 256-bit number inputs a, e and n, the program should calculate ae mod n using the “exponentiation by squaring method” and “Montgomery Algorithm”. When the calculation is finished, the signal o\_finished will turn from 0 to 1 and o\_a\_pow\_e will output the 256-bit result.

- **DE2\_115.sv**:

Uses Qsys file which has already set up connections between modules “Rsa256Wrapper” and “Rsa256Core”, and acts as an interface between Verilog codes and FPGA board.

**• Design**

**- RSA Cryptosystem:**

►**Key Generation**

The keys of RSA cryptosystem are generated in the following manner:

1. Choose two prime numbers p and q.

* For security purpose, these prime numbers should be large and chosen randomly.

1. Compute N = pq to be used as modulus for both public key and private key.
2. Compute r = largest-common-multiplier(p-1,q-1), and then choose an integer e such that 1 < e < r and greatest-common-divisor(e,r)=1.
3. Compute d = e-1 mod r.
4. N and e are public keys for anyone who would like to send messages to the receiver. d is the private key that should be kept secret by the receiver.

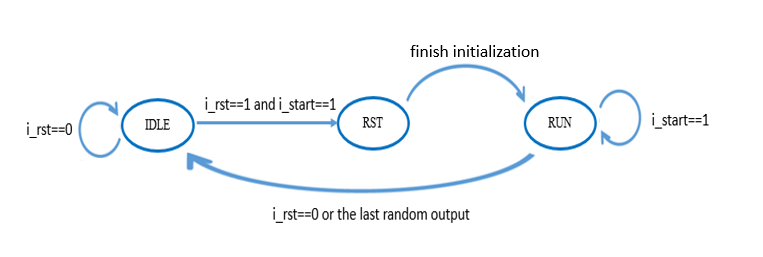
►**Encryption:**

For a message m, compute ciphertext y from the formula y = me mod N for encryption.

►**Decryption:**

For a ciphertext y, compute plain message m from the formula m = yd mod N for decryption.

**- Finite State Machine:**



* **Hint:** use “enum” in System Verilog to record your state.

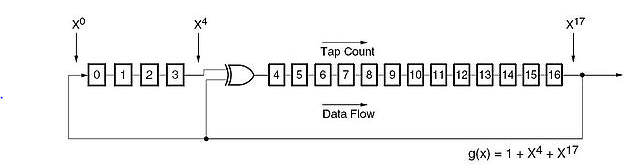
**- Circuit Implementation:**

****

* **Hint:** \_r stands for register and \_w stands for wire

**- Random Number Generator:**

We apply the linear feedback shift register (LFSR) to create pseudo-random numbers. The advantages of this method are easy implementation, fast operation and little memory requirement. Since the situation does not require high-quality randomness, the LFSR works pretty well. Also, by selecting an appropriate feedback function, we can produce a sequence of bits which appears more random and has a longer repeating cycle.



* **Note:** Do not assign 0 to all bits in the LFSR as the initial state (or seed); otherwise, all the bits will remain 0 forever.