DCLab2

Team04 韓秉勳 蔡昕宇 解正平

User Manual

1. Prepare text and RSA key file
    - Generate different length RSA Hex key from websites including N, e, and d value. Put them in three lines into text file naming keyLENGTH.txt {LENGTH = 128, 256, 512, 1024. Ex：key256.txt}
      URL：http://www.mobilefish.com/services/rsa_key_generation/rsa_key_generation.php
    - Create a file (command fsutil file createnew filename length)preparing for encoding and name decLENGTH.txt. However, we need to be aware of the bits size in the file must be the multiple of (LENGTH/8)-1 {LENGTH = 128, 256, 512, 1024. Ex: dec256.txt size=31n)
    - Run rsa_new.py to encode file with keyLENGTH.txt and decLENGTH.txt following generating keyLENGTH.bin and encLENGTH.bin.
      Ex：for LENGTH = 256 python2 rsa_new.py e 256
    - Run rsa_new.py to decode file with keyLENGTH.txt and encLENTH.bin following generating keyLENGTH.bin and decLENGTHo.txt
      Ex：for LENGTH = 256 python2 rsa_new.py e 256

2. Setting FPGA
    - Ensure using RS232 wire to connect to the computer and the LED is lighting showing that FPGA is ready.
    - Use FPGA switch from SW[7] to SW[10] to set what key length you may decode. Ex: SW[8] indicates that key length is 2^8 = 256 and you can see the number on Seven-segment display.
    - If we finish decoding but the LED is not lighting in normal, you can press KEY[0] to reset FPGA.

3. Run python code
    - Use pip2 to Install PySerial to make sure no errors.
    - Check what the port name is in your computer.
    - Put the keyLENGTH.bin and encLENGTH.bin in the same folder with the file rs232_esc.py to ensure that we can get file and decode another file without pressing reset.
    - Run rs232_esc.py to transmit data into FPFA and get decoding response.
      Ex：for LENGTH = 256 python2 rs232_esc.py PORT 256

Teaching Manual

I. 實驗目的

1. 了解 RSA 加密演算法的基本原理
2. 了解如何利用數學方法， 避開對硬體資源要求較多的乘法運算和模運算。
3. 練習利用 Qsys 套用 Altera 提供的 UART module
4. 設計 protocol 藉由 Avalon MM 介面進行 RS-232 傳輸

II. 實驗方法

透過電腦傳輸加密檔案，FPGA 設計 RSA 對資料解密，並回傳結果回電腦。

III. 實驗原理

A. RSA

1. RSA 基本演算法

基本上傳送者將內容加密，選兩個數$d, e \leqslant N$，使得$a^{de} \equiv a\ (mod\ N)$，其中$N$為兩個質數相乘。欲傳送的資料$x$，透過$y \equiv x^e\ (mod\ N)$，得到加密的$y$。那接收端得到加密的資料後，以$x = y^d\ (mod\ N)$還原原始傳送的資料$x$。在本實驗中即實作解密的動作。

2. 快速冪演算法

計算y^d時，由於 d 可能是一個很大個數值，因此需要有效的方法計算。可以將 d 以 2 進位表示。舉例來說計算 $y^{28} = y^{(1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0)}$，其中 28 = $(11100)_2$。我們可以初始值給t 給y，連乘 y, 得到 y^2、y^4、...。另一個要回傳的值如果指數遇到 1 時，乘上t即可。

在此當中我們遇到乘法的運算，為了加快運算速度，可以利用上述指數連乘運算方式的改念，利用累加的方式來達到相同的目的。舉例來說如果要計算 $28y = 1 * 16y + 1 * 8y + 1 * 4y + 0 * 2y + 0 * y$，28 = $(11100)_2$，我們可以初始值給t 給y，連加 y, 得到$2y$ 、4y、...。回傳值是遇到 1 時加起來即可得到答案。

另外考慮$mod\ N$的情況，在後者的狀況下，假設原本值為$a \leqslant N$，$2a \leqslant 2N$，因此我們考慮當加起來時大於N的情況，將其減去N，得$2a - N$。綜合起來即是快速冪演算法的整體架構。

3. Montgomery 演算法

Montgomery 演算法是為了計算 $ab\ (mod\ N)$的演算法。已知 $A \equiv a \times 2^{256}\ (mod\ N)$，$B \equiv b \times 2^{256} (mod\ N)$，可得到 $AB \equiv ab \times 2^{256}\ (mod\ N)$，再推回$ab\ (mod\ N)$。由模的反元素運算，可進一步得到：

$$ab \times 2^{256}\ (mod\ N) \equiv AB * 2^{-256} (mod\ N)$$

另外，右式的部分可得到

$$AB \times 2^{-256} = \left(\sum_{i=0}^{255} A_i 2^i\right) \times B \times 2^{-256} = \left(\sum_{i=0}^{255} A_i 2^{i-256}\right) \times B$$

$$= (((A_0 \times B) \times 2^{-1} + A_1 \times B) \times 2^{-1} + \cdots) \times 2^{-1}$$

發現可以跟前面的演算法有類似共通之處，把 recursive 迴圈中改成 $(t + A_0 B) \times 2^{-1}$，即可 recursive 的呼叫。

4. 結合 Montgomery 和快速冪

觀察兩者的結構可發現，如果初始化的 t 給 $y \times 2^{256}$，則 Montgomery 的演算法中, a = t, b = t, 可以化簡成

$$y^2 \times 2^{256}, y^4 \times 2^{256}, \ldots$$

另外一個在 Montgomery 的演算法中, a = 要回傳值, b = t, 可發現 2^{256} 被抵消了。即可得到我們要的解密過程。

B. Avalon Memory-Mapped Interface

1. Input/Output Interface

	方向	功能
clk	input	clock
address	output	需要 read/write 的記憶體位置 （UART 模組對應的記憶體位置參考下圖）
read	output	傳給 Avalon MM Slave 需要讀取資料的訊號
readdata	input	讀取到的資料
write	output	傳給 Avalon MM Slave 需要寫入資料的訊號
writedata	output	要寫入／傳送的資料
waitrequest	input	當 waitrequest 為 1 時，read/write 動作都暫停 （在 waitrequest 回到 0 之前 writedata 要維持不變）

此為本實驗有使用的功能，主要用來與 FPGA 的 memory 連結拿資料。方式非常簡單，以下是大概的介紹：

要讀取資料 -> address(status) 確認 read ready ->
read(1)write(0)給 memory 說要讀取 -> address(read) 到讀取地址拿資料
要傳送資料 -> address(status) 確認 transmit ready ->
read(0)write(1) 給 memory 說要傳送 -> address(write) 到傳送地址給資料

2. Register information

Offset	Register Name	R/W	Description/Register Bits													
			15:13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO	Reserved					1	1	Receive Data						
1	txdata	WO	Reserved					1	1	Transmit Data						
2	status 2	RW	Reserved	eop	cts	dcts	1	e	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW	Reserved	ieop	rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor 3	RW	Baud Rate Divisor													
5	endof-packet 3	RW	Reserved					1	1	End-of-Packet Value						

此為本實驗中與 memory 溝通的 register 內容，若要確認是否有 read ready 或是 transmit ready，register 要配置 8 號的記憶體位址，並分別確認 7 號及 6 號 bit。當確認可以再配置 0 號(讀取)或 4 號(傳送)的地址，並且只能接受或寫入 0-7 號 bit，也就是說每次都只能使用 8 個 bit 來移動資料，因此需要特別注意每個 package 的 data 都會切成 8 bit 來移動，所以需要常常確認 ready bit 是否可以再移動下一份 8 bit。

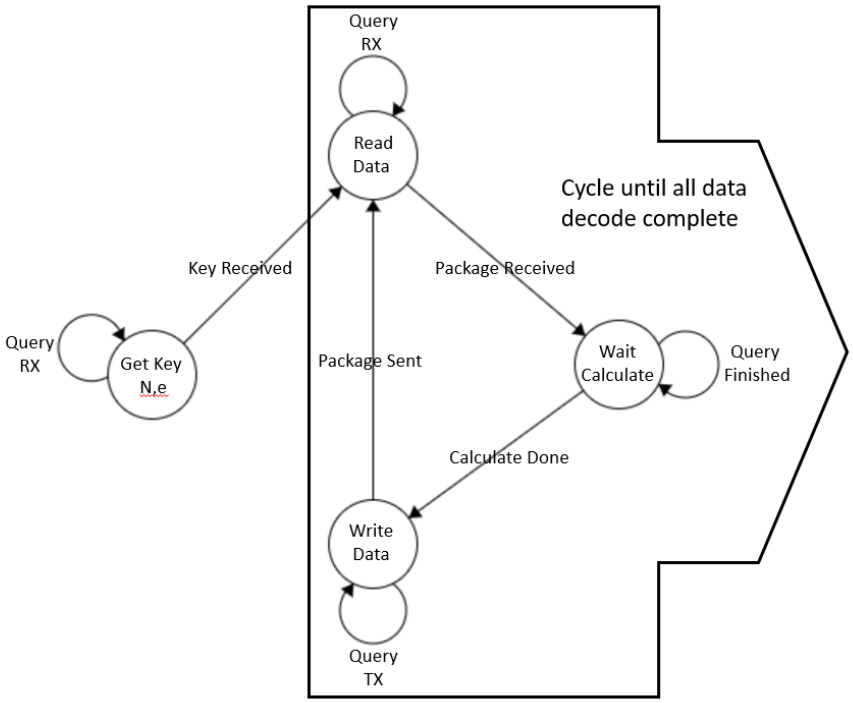
3. FSM diagram

以下是本實驗基本的 FSM，同學們可以根據自己的想法修正。

由於每次運算只需要一份 key 但 data 會切成很多份 package，我們將 read key 和 read data 分成兩個 state，並將 query RX 和 TX 寫在 read 裡面。當 key received，接下來就是處理每一份一份的 data package，一份 package 因為會切成每個 8 bit 在 memory 移動，所以會不斷 query，直到獲得整份 package，最後整份計算完再寫出 decode 好的整份 package，不斷循環直到 data 的所有 package 都 decode 完成。

a. Get Key N, e：query RX and wait for receiving all key package (per 8 bits)
b. Read data：query RX and wait for receiving data package (per 8 bits)
c. Wait Calculate：query Finished and wait for the package decoded
d. Write data：query TX and wait for transmitting data package (per 8 bits)

=> b.c.d. cycle until all data package had decoded



C. PySerial

1. Key length 以每份大小 8 bit 換算得 package size：
   因為 FPGA 裡面的 memory 每次只會以 8 bit 來移動資料，所以可以計算得知會要多少次 8 bit 才可以傳送完一份 key，我們設定這個次數當作整個 package size，可以用來衡量甚麼時候 key received。
2. Encode Data 切成每份大小與 key length 相同來 decode：
   我們設定每次 encode 的 size 和 key length 相同 bit 數量，因為讀檔每行讀進來是 2 bytes = 8 bits，所以讀的行數剛好可以用 package size 來衡量每次要幾行的資料來 decode，而且也可知道甚麼時候讀完整個 package 然後再進入 wait calculate。下圖是舉例和圖示：
   a. Key length = 256 bits
   b. Package size = 32
   c. Query to read one key iteration = 32 times (two key N, e = 64)
   d. Data size to encode = 32 line
   e. Query to read one package iteration = 32 times

|.....Package Size……..|

Key	8 bit	8 bit	…...	8 bit	8 bit

Encode Data			
1st package	2nd package		last package
8 bit	8 bit	…...	8 bit
…….	…….		…….
8 bit	8 bit		8 bit

IV. 額外功能

A. 支援連續傳檔案， 每份檔案之間不需要手動按下 reset 按鈕

由於之前的 FSM 觀察不難發現，每次進入 decode data 就無法再重新載入下一份 key 跟 data，只會不斷在迴圈中讀記憶體跟解密，直到按下 reset 重新跑程序。為了可以讓 FPGA 在接收完一個檔案後可以接受下個檔案， 基本上就是要讓 FPGA 端知道是不是已讀到檔案結尾， 以下是幾種特別的想法，當然還有很多方式同學們可以想想看。

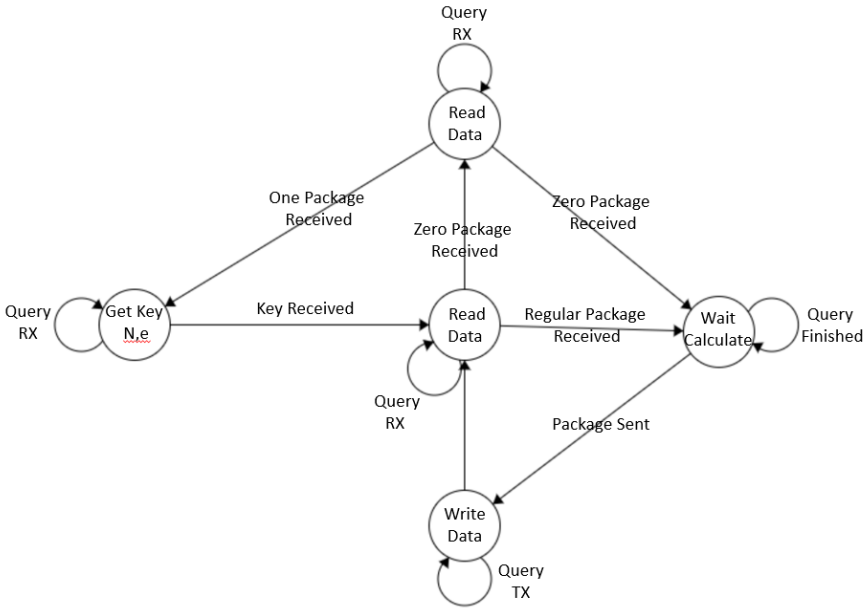
1. 事後傳輸 Finish 記號：

本實驗多加入 0 和 1 兩個 package 連續訊號當作 data 結尾記號，當看到兩個連續 package 結尾 8 bits 分別是 0 和 1 表示 data decode 完成，可以將 state reset 到 Get Key 狀態。

0 package：以 key length 表示的 0 (000…000) (結尾 8 bits 0000,0000)

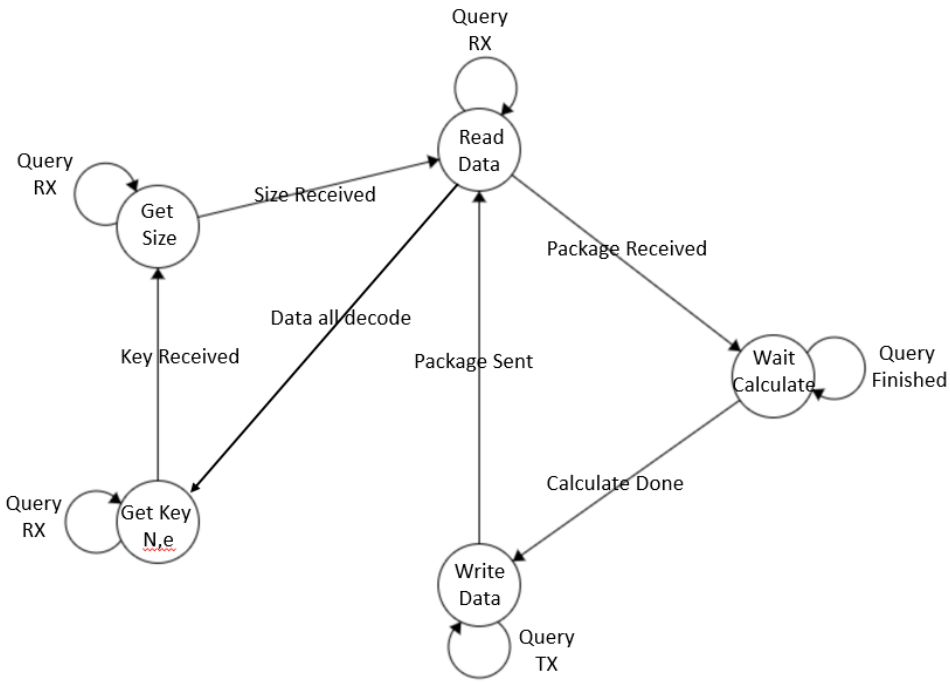
1 package：以 key length 表示的 1 (000...001) (結尾 8 bits 0000,0001)

然而因為原先檔案也有可能兩連續 package 最後 8 bits 是 0 和 1 但不是 data 結尾，我們將原本有 0 的訊號後面多加一個 0 package，這樣就可以知道這個 0 是有用的，不是當作結尾訊號，以下是修正的 FSM diagram。



2. 事前傳輸檔案大小：

   除了傳資料結尾的時候給出 Finish 記號，我們也可以直接先給 data 的 package 數量，這樣統計有處理過幾次 package 就可以知道是不是已經結尾，但很有可能因為 data 數量龐大，存 size 的 register 不夠大；或是這很有可能因為 data 中間傳輸斷掉，造成數量不對。以下是修正的 FSM diagram。



B. 支援切換不同 bit 數的 RSA key

直接將之前不管是 verilog 或是 python 檔中原本 256 bits 的地方改成可以設定即可，可以發現將 換成其他常見的 bit 數（如 128、 512、 1024）RSA 演算法也是成立的，但要注意更改一些原本是常數的東西要想怎麼修正，比如說 wrapper 的輸出不再是輸出固定的 [247-:8] 而要視情況而定。

另外記得測資要生成不同 length 的 key 以及 dec.txt，還有 rsa.py 也需要更改。

C. 支援顯示目前解密進度

使用 FPGA 上面的 LED 燈當作處理中，如果開始 read data 即變暗，直到 reset 發生才會再變亮，可以用來判斷說 data 是否已經處理完，顯示進度幫助解碼。

V. 常見問題

A. 如果 qsys 有新增除了原本助教範例的 I/O 要怎麼設定？

助教範例的 type 是 Avalon Memory Mapped Master，可以自己更改 type 成 conduit，但是產生的 .v 檔並不會有 wire 接線，要自己另外更改 code，將 top level 設定新的 I/O 連到原本設定未接的地方。

如在我們的 lab2 中，qsys 原本接線如下:(DE2_115.sv)

```
DE2_115_qsys my_qsys(
    .LED_ready(LEDG[0]),
    .i_RSA_BIT(RSA_BIT),

    .clk_clk(CLOCK_50),
    .rst_reset_n(KEY[0]),
    .uart_0_external_connection_rxd(UART_RXD),
    .uart_0_external_connection_txd(UART_TXD)
);
```

我們多了 LED_ready 與 i_RSA_BIT 要接，(分別控制 LED 燈與不同 bit 的解碼功能)

此時我們可以在 qsys 的.v 檔中，自行增加 wire，如圖:

```
module rsa_qsys (
        input  wire  [12:0] i_RSA_BIT,
        input  wire  clk_clk,                       //                    clk.clk
        input  wire  reset_reset_n,                 //                  reset.reset_n
        input  wire  uart_0_external_connection_rxd, // uart_0_external_connection.rxd
        output wire  uart_0_external_connection_txd,  //                            .txd
        output wire  LED_ready
    );
```

我們多加了 LED_ready 與 i_RSA_BIT，同時在下方 rsa_wrapper 模組將線接上:

```
Rsa256Wrapper rsa_wrapper_0 (
    .avm_address     (rsa_wrapper_0_avalon_master_0_address),     // avalon_master_0.address
    .avm_read        (rsa_wrapper_0_avalon_master_0_read),        //                .read
    .avm_readdata    (rsa_wrapper_0_avalon_master_0_readdata),    //                .readdata
    .avm_write       (rsa_wrapper_0_avalon_master_0_write),       //                .write
    .avm_writedata   (rsa_wrapper_0_avalon_master_0_writedata),   //                .writedata
    .avm_waitrequest (rsa_wrapper_0_avalon_master_0_waitrequest), //                .waitrequest
    .avm_clk         (altpll_0_c0_clk),                           //      clock_sink.clk
    .avm_rst         (rst_controller_reset_out_reset),            //      reset_sink.reset
    .i_RSA_BIT       (i_RSA_BIT),                                  //     conduit_end.beginbursttransfer
    .LED_ready       (LED_ready)                                   //                .writeresponsevalid_n
);
```

此乃完成新增 I/O 的設定。

B. Testbench 都過了可是跑完 python 出來結果不對怎麼辦？
高機率是 python 版本問題，請務必使用 python2 來執行，並且確認 code 並無錯誤，再來是確認板子開關是否搞錯，最後請更換一塊 FPGA 試試看。