數位電路實驗

LAB 2: RSA256 解密機

B06901190 陳昱仁, B06505021 楊欣哲, B06505011 傅敬倫

一. 實驗目的

1. 了解 RSA256 運算結構
2. 了解不同運算方式對硬體效率的影響
3. 了解使用 Qsys

二. 實驗原理

1. RSA 演算法

在傳遞訊息時，可以利用公鑰(N,e)、私鑰(N,d)進行加密解密。

會先利用公鑰對要傳遞的訊息 n 加密成訊息 c，方法為:

$$n^e \equiv c(mod\ \ N)$$

利用私鑰對加密的訊 c 進行解密，方法為:

$$c^d \equiv n(mod\ \ N)$$

2. 拿到公鑰加密後的訊息 y、N

在此實驗會拿到公鑰加密後的訊息 y 與 N，此時可以利用下方的演算法找出我們要的 m，而在計算y^d時，可以藉由 d 的 bit 來計算，舉例來說:

如果要計算y^{25}，則 d=25 換成二進位表示$(11001)_2$，可以將問題轉成:

$$y^{25} = 1 \times y^{16} + 1 \times y^8 + 0 \times y^4 + 0 \times y^2 + 1 \times y^0$$

其中y^{2^k}的計算只需在每次迴圈中執行$y \leftarrow y + y$

Algorithm 1 RSA256 with exponentiation by squaring

```
1: function ExponentiationSquaring(N, y, d)
2:     t ← y
3:     m ← 1
4:     for i ← 0 to 255 do
5:         if i-th bit of d is 1 then
6:             m ← m · t (mod N)
7:         end if
8:         t ← t^2 (mod N)                ▷ t → t^2 → t^4 → ···
9:     end for
10:    return m
11: end function
```

3. 優化 Modulo of Product

可以發現在上述的演算法中紅色框起來的部分是要算兩個數字相乘的餘數，但乘法對於硬體來說消耗太大，因此可以改成用加減法的方式改進。舉例來說，計算 y*25 時，可以將他拆成:

$$y \times 25 = 1 \times 16y + 1 \times 8y + 0 \times 4y + 0 \times 2y + 0 \times y$$

其中$2^k y$的計算只需在每次迴圈中執行$y \leftarrow y + y$

其演算法如下:

Algorithm 2 Modulo of products

```
1: function ModuloProduct(N, a, b, k)                ▷ k is number of bits of a
2:     t ← b
3:     m ← 0
4:     for i ← 0 to k do
5:         if i-th bit of a is 1 then
6:             if m + t ≥ N then
7:                 m ← m + t − N                     ▷ perform modulo operation in each iteration
8:             else
9:                 m ← m + t
10:            end if
11:        end if
12:        if t + t > N then
13:            t ← t + t − N                         ▷ perform modulo operation in each iteration
14:        else
15:            t ← t + t
16:        end if
17:    end for
18:    return m
19: end function
```

可以發現在每次跳到下一位時都會先檢查目前的餘數是否大於 N，如果大於就減掉，不會像 2.到最後乘完之後才去減掉 N 的倍數

4. Montgomery Algorithm

可以發現在上述的演算法中還是需要比較餘數是否超過 N，而 Montgomery Algorithm 是在計算 ab(mod N)時，可以先對 ab 都先乘上一個數字，以這次實驗來說是2^{256}，則有:

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

因為有2^{-256}出現，因此可以將乘法改成除法，就可以確保每個階段算出的 m 都在 0-N 之間，其演算法如下:

Algorithm 3 Montgomery algorithm for calculating $ab2^{-256}$ mod N

```
1: function MontgomeryAlgorithm(N, a, b)
2:     m ← 0
3:     for i ← 0 to 255 do
4:         if i-th bit of a is 1 then
5:             m ← m + b
6:         end if                                    ▷ 4~6: replace multiplication with
                                                        successive addition
7:         if m is odd then
8:             m ← m + N
9:         end if
10:        m ← m/2                                   ▷ 7~10: calculate the modulo of a·2^−1
                                                        → Montgomery reduction
11:    end for
12:    if m ≥ N then
13:        m ← m − N
14:    end if
15:    return m
16: end function
```

5. Final Algorithm

   最後結合 3 和 4 可以將一開始的演算法寫成下面這個演算法:

Algorithm 4 RSA256 with exponentiation by squaring and Montgomery algorithm

```
1: function RSA256MONT(N, y, d)
2:     t ← ModuloProduct(N, 2^256, y, 256)
3:     m ← 1
4:     for i ← 0 to 255 do
5:         if i-th bit of d is 1 then
6:             m ← MontgomeryAlgorithm(N, m, t)
7:         end if
8:         t ← MontgomeryAlgorithm(N, t, t)
9:     end for
10:    return m
11: end function
```

三. 實驗器材

1. Ncverilog:

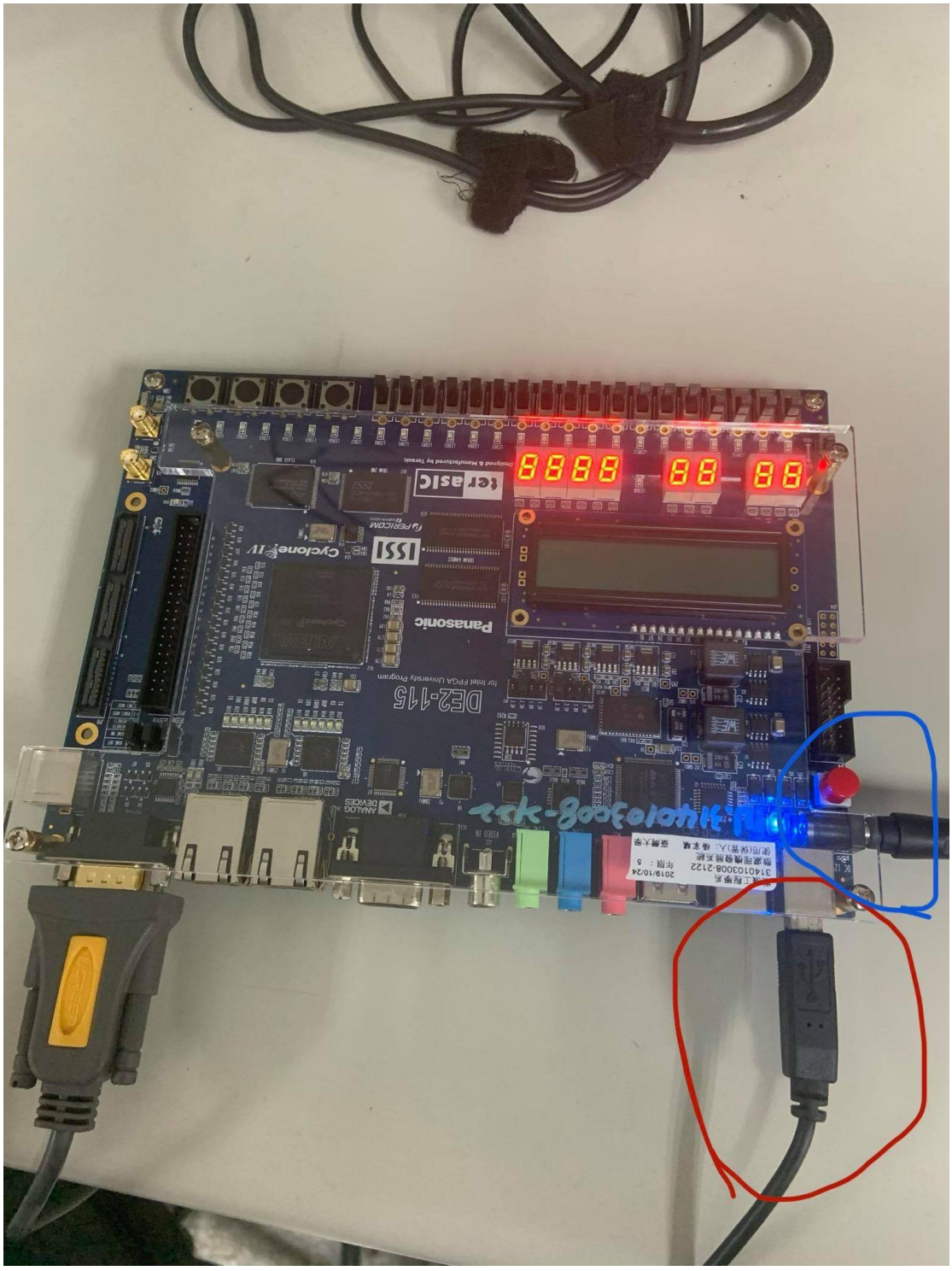
   用來 debug

2. Quartus:

   將 verilog code 合成為可供 FPGA 使用的 sof 檔案

3. FPGA
4. USB cable
5. 12 volt adapter

四. 架設方式

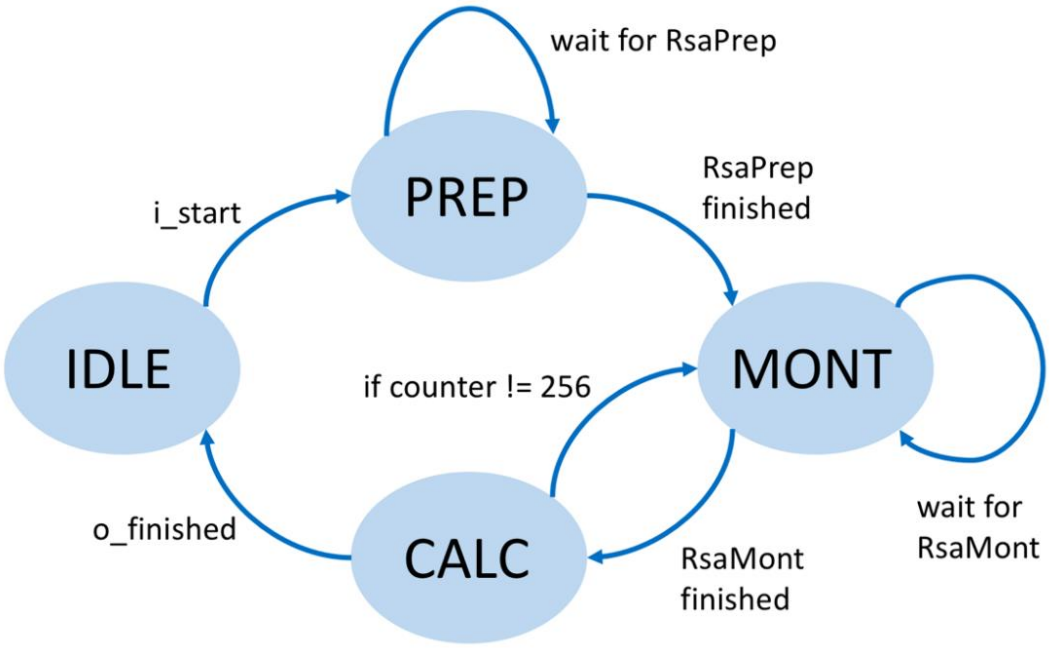


紅色部分為 USB cable，用來連接電腦與 FPGA

藍色部分為 12 volt adapter，用來提供 FPGA 電源

五. 硬體架構

RSA256CORE:



共分四個 state:

1. IDLE:起始 state，等待 i_start 訊號則進入到 PREP state
2. PREP:實作 Modulo of Product(詳見上方)
3. MONT: 實作 Montgomery Algorithm(詳見上方)
4. CALC:將 MONT state 的參數歸零，並計算總共進入 MONT state 次數，當達到 256 次時結束，回到 IDLE

RSAWrapper:

1. Avalon Memory-Mapped Interface

   Avalon Memory-Mapped Interface（以下簡稱 Avalon MM）實際上提供了很多功能，不過這次實驗中使用的 UART 模組只使用了以下輸出入介面：

	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）

換句話說：

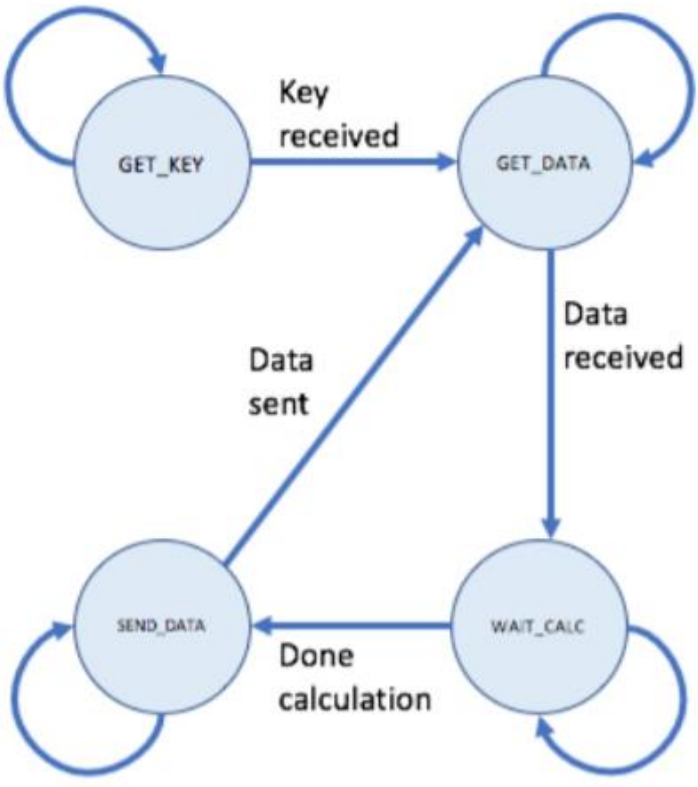
要讀取狀態時，將 read←1、write←0、address←8。
要接受資料時，將 read←1、write←0、address←0。
要傳送資料時，將 read←0、write←1、address←4。

以下是 UART 模組的暫存器內容：

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						

圖中紅色方框內是本實驗會用到的部分。
當 rrdy=1 時代表 RX 已經讀取完畢，可以進行 read 操作。
值得注意的是當進行 read 操作後，rrdy 會自行變回 0。 當 trdy=1 時代表 TX 已經完成傳送資料到 buffer，可以進行 write 操作。

2. FSM



了解 Avalon MM 的工作方式後，可以設計出以下四個狀態：

1. GET_KEY：等待金鑰 (N,d) 傳送完畢。
2. GET_DATA：讀取資料直到讀取完 32 byte。
3. WAIT_CALC：等待 Core 計算完畢。
4. SEND_DATA：等待 31 byte 的解密資料傳送完畢。

六. 使用方式

在 cmd 執行 python 檔案 rs232.py，值得注意的是需要先行安裝 pip，我們當初沒有安裝，之後才把這個問題解決。

七. 問題發現與解決

這次的實作並不難，重點在於數字較為龐大，做起來比較難 debug，花比較多時間在處理 RS232，指令較為繁瑣，不小心按錯一步就要重新來過，最後我們耐心地完成了。