Team01 Lab2 Report

– 使用所需器材與架設方式

這次實驗用到的器材有DE2-115 FPGA 板、電源線、USB傳輸線與RS232傳輸線。FPGA版接上電源線後，打開電源並將左下角的開關切到RUN。利用 USB 傳輸線將電腦與 FPGA 板連接後，用Quartus把程式燒到FPGA上。成功之後透過RS232將FPGA板與PC端連接，便可以開始進行RSA256的解密。

DE2-115 FPGA 板

電源線

USB 傳輸線

RS232 ↔ USB

– 使用方式與詳細步驟

Key0：RESET 按鈕

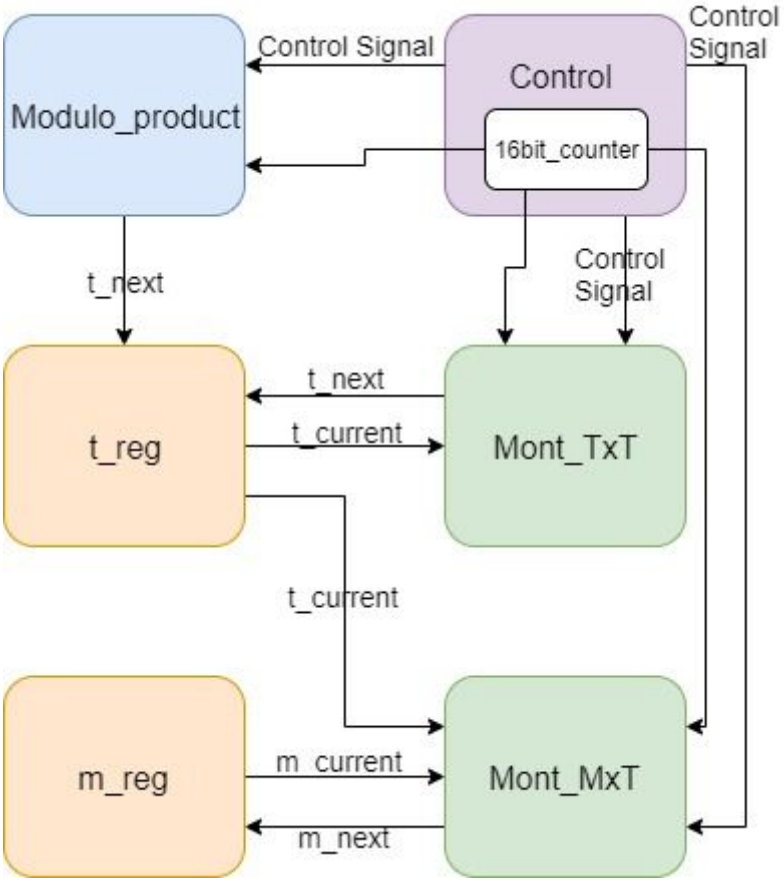
使用步驟：

1. 使用前先按下Key0，做 RESET的動作。
2. 在 pc_python/rs232.py 中，fp_key 是解密的金鑰，fp_enc是待解的密文，而最後解密完之後會寫在fp_dec。輸入以下的指令執行rs232.py：python rs232.py COM? (COM? 是OS指派的port 代號)
3. 如果想要解不同的密文，要修改fp_enc = open() 中的檔案路徑，到你要測試的檔案(goden/enc?.bin)。不用再按一次RESET按鈕(BONUS)，就可以執行rs232.py做下一次解密。

– 實作設計技術細節與巧思

Core：

下圖是Core Module的架構

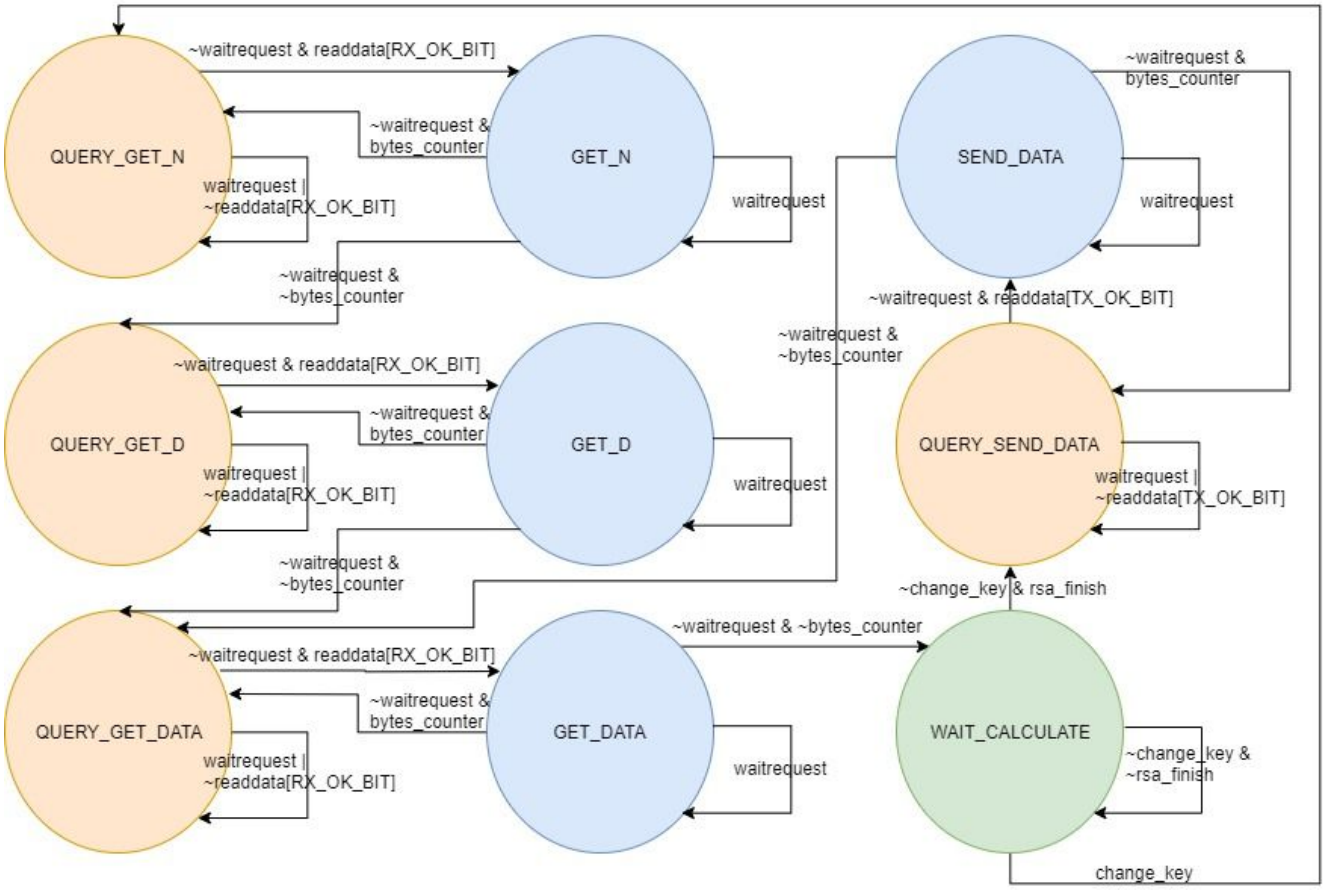


計算的過程分為三個state，IDLE、Modulo Product(以下簡稱Mod)及Montgomary(以下簡稱Mont)，其中Mod state需要256個cycle，而Mont state則因為MxT與TxT可以同時做運算，只需要256(外層的for迴圈)*256(一次Mont運算)個cycle，cycle的計數由Control內的Counter統一控制。在Mod state，當counter數至255時，將Mod運算結果存入t_reg中，並跳至Mont state。在Mont state，每當counter的8個LSB為255，t_reg的值

會被新計算出的TxT更新，而m_reg則會依據counter的8個MSB(即是pseudo code中的i)及d值決定要不要將儲存的值更新為計算出的MxT。最後，counter數至256*256後會跳回IDLE state並輸出m_reg的值。

Wrapper：

Wrapper的state graph如下圖所示



reset完之後會進入QUERY_GET_N的state，此時會把bytes_counter初始為32，等到readdata[RX_OK_BIT]為1時，會將bytes_counter的值減1，並且進入GET_N的state，此時將會把readdata[7:0]的值寫入n_w[(8 * bytes_counter_r + 7) -: 8]，若bytes_counter的值尚不為0，會回到QUERY_GET_N繼續進行讀取N的動作，否則才會進入QUERY_GET_D，並將bytes_counter重新設為32。QUERY_GET_D、GET_D與QUERY_GET_DATA、GET_DATA也是使用了上述邏輯去讀取D以及cipher text。

讀取完32個bytes的cipher text之後，即進入WAIT_CALCULATE的state，此時會將rsa_start設為1，使得core開始進行運算，並等待rsa_finish跳為1，才會進入QUERY_SEND_DATA的state以開始進行寫入。值得注意的是，此處將會判斷讀進來的cipher text是否為{256{1'b1}}（見下方註

解），若成立則會觸發change_key的訊號，使得Wrapper跳回QUERY_GET_N的state，重新讀取key。
QUERY_SEND_DATA、SEND_DATA則是負責將資料寫回。此處的邏輯與上述讀取的邏輯相同，不同的細節在於判斷valid時要讀取的位置是readdata[TX_OK_BIT]。另外一個不同處是bytes_counter應該要設為31，原因是傳回的數值只有31個bytes，最前面將會再pad上一個byte的0以避免overflow。當bytes_counter變為0時會回到QUERY_GET_DATA，再進行下一組32bytes的cipher text的讀取。

註：因cipher text為某個數字mod N所得到的值，又因N必定不會大於{256{1'b1}}，可知正常的cipher text不可能為{256{1'b1}}，所以我們可以利用此訊息作為重新讀取key的signal。

– 碰過的問題或挑戰與解決方式

我們在實驗過程中遇到了幾個問題：

1. Montgomary Alg.過程中的Overflow

在Montgomary Alg.中，若m+b為奇數，則要再加上n，我們原先只為加法的sum預留257個bit，但m+b+n最多是3n，有可能會使用到258個bit，因此造成Overflow的問題。

2. FPGA的行為與預期中不同

我們bonus的設計，是當收到的Cypher Text為{256{1'b1}}時會觸發特定行為，不過使用python執行解碼時，送出的{256{1'b1}}並不會觸發該行為。我們為了尋找問題點，送出了256'b0試著藉由解碼的結果確認解碼正確性以及電路的狀態，卻意外觸發該行為。起初由於在網路上查到RS232送出的Logic 1是負電壓，Logic 0是正電壓，與TTL相反，便以為是FPGA會將收到的訊號誤認為相反的邏輯值，不過後來經過詢問助教，以及與其他組別討論，並沒有相似的現象，而且送出的enc及key可以解出沒有invert的正確結果，因此可以排除傳出的Cypher Text在任一個環節被反相的可能性。經過助教的確認，verilog的寫法似乎並沒有問題，因此雖然Demo時可以實現我們設計的行為，我們仍不知道這個現象發生的原因。

3. rsa_finish應為triggering signal

一開始的core將rsa_finish設為1之後，下一個cycle時不會立即變回0，而是等到下一次rsa_start被設為1之後才會將rsa_finish歸0。此問題將會造成下一次wrapper進入WAIT_CALCULATE時，會以為core已經立即完成計算，所以接著會直接進入QUERY_SEND_DATA以準備進行資料的寫入，然而實際上core根本還沒對這次的cipher text進行運算，所以只是將上一批資料再寫入一次，而引發錯誤。正確的方式應該是rsa_finish被設為1的下一個cycle應馬上被設回0（意即triggering signal），如此一來才不會發生上述的問題。

4. 執行rs232.py時，程式會卡住沒有回應

把程式燒進FPGA板上後，要執行rs232.py時，我們發現不管等多久，檔案一直都在執行中沒有回應，我們嘗試用crtl+c直接砍掉也停不下來。我們還以為這個檔案本來就要解很久，所以就放著讓它一直跑個半小時，可是最後看dec.bin裡面居然還是空的什麼都沒有，實在無法理解。直到我們不小心按到key1，FPGA板上的七段顯示器開始亂跳之後，才發現這怎麼跟我們lab1做的這麼像。之後我們重新燒一次，居然就可以成功解密了。