Team04_lab2_report

1. File structure

team04_lab2

|-team04_lab2_report.pdf

|-src

  |-DE2_115

    |-DE2_115.sv

 |-Rsa256Wrapper.sv

 |-Rsa256Core.sv

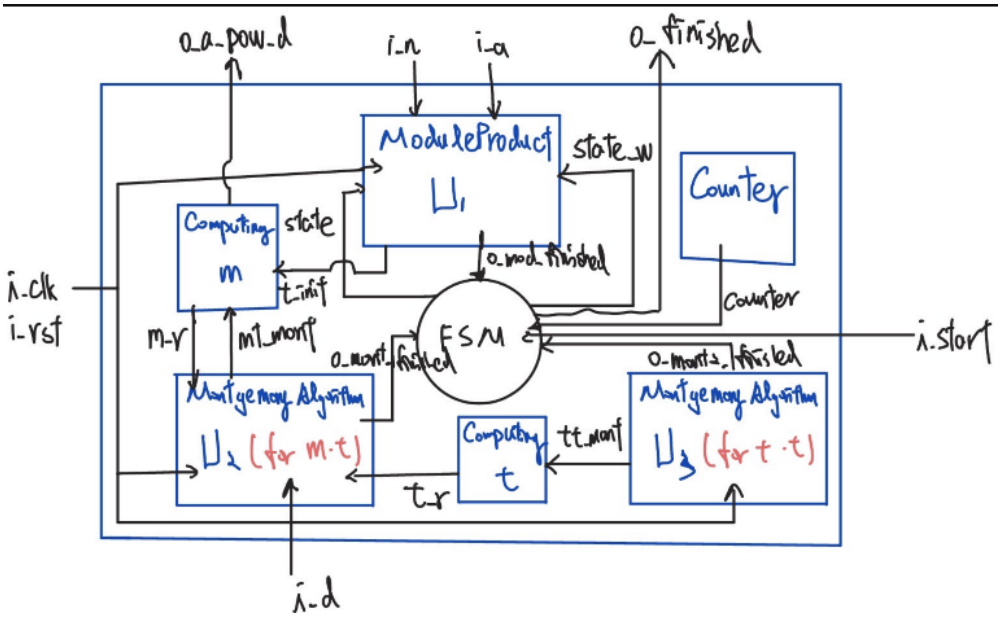
 |-Montgometry.sv

 |-ModulePruduct.sv

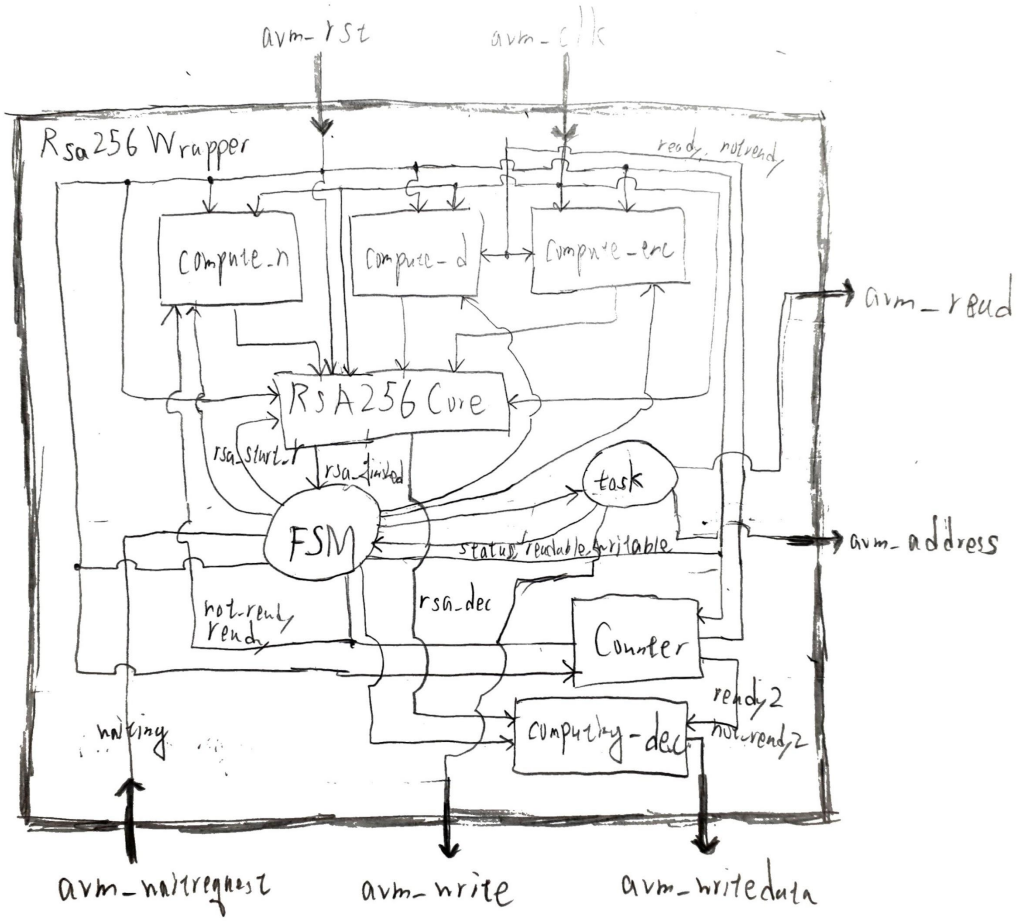
 |-rsa_qsys.v

2. Block Diagram

Rsa256Core



Rsa256Wrapper

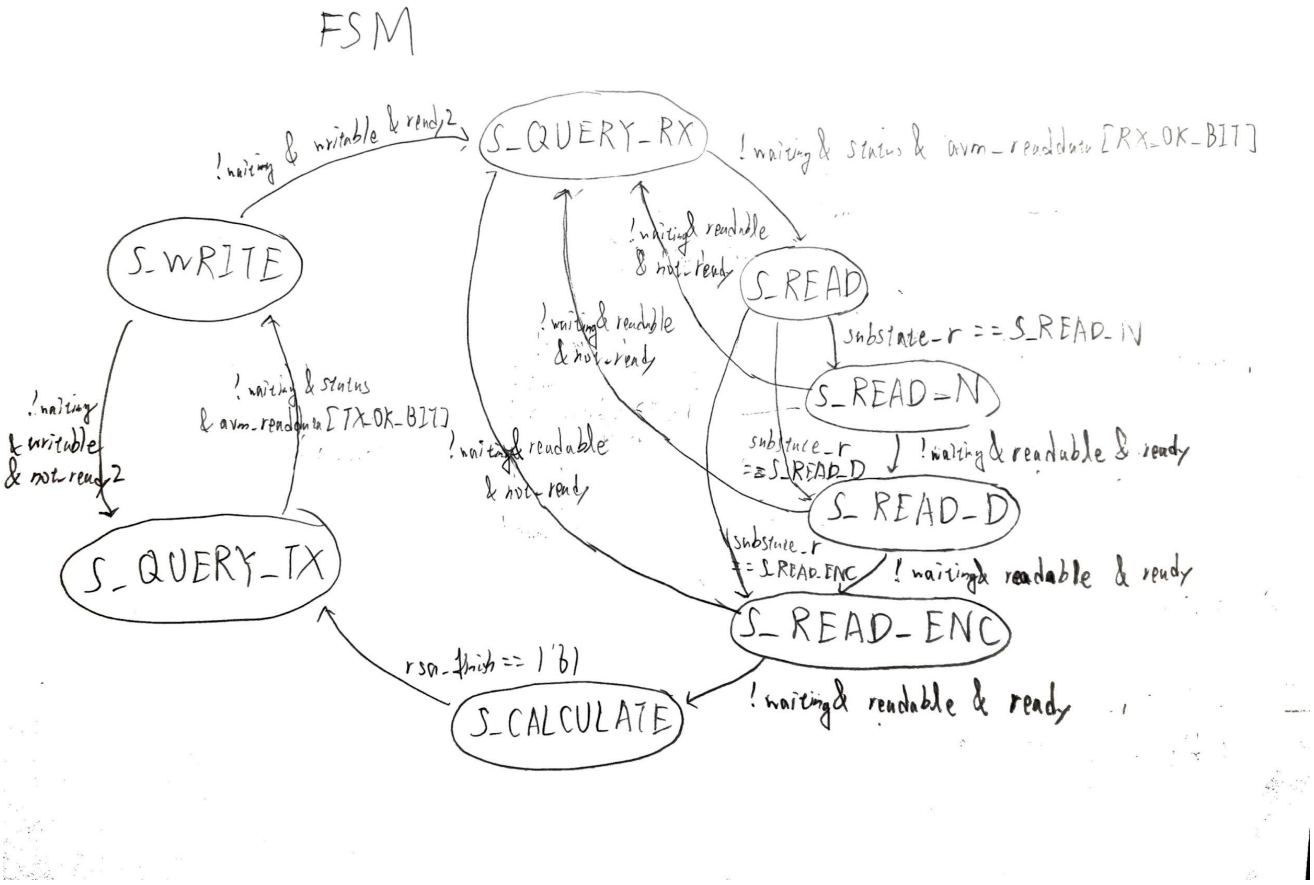


3. FSM and Hardware Scheduling

Rsa256Wrapper:

我們的Wrapper總共有5個state, 分別為S_QUERY_RX, S_READ, S_CALCULATE, S_QUERY_TX, S_WRITE, 在S_QUERY_RX裡面, 會等avm_waitrequest為0, address為STATUS_BASE 且 readdata[7] = 1時, 才能進行讀;而在S_READ就要進行讀, 要讀三筆資料, 分別是n, d, enc, 於是用三個substate, 有S_READ_N, S_READ_D, S_READ_ENC, 各自讀一筆資料, 每筆要讀32次, 用一個counter來計算, 每次讀8bits, 讀完就要將address重設成STATUS_BASE, 回到S_QUERY_RX等待下一次讀。每個substate讀完32次以後就跳到下一個讀下一個資料的substate, 當enc也讀完, 就會跳到S_CALCULATE, 進行解密運算, 此時控制Rsa256Core module開始運算的控制訊號rsa_start就會拉高, 等到運算完成後, Rsa256Core送的rsa_finished訊號就會拉高, 使得進入下一個state, 將得到的dec進行寫的動作。在S_QUERY_TX跟等待讀很類似, 也要等avm_waitrequest變為0, address為STATUS_BASE且readdata[6] = 1時, 才能進行write, 會用task將avm_write的訊號拉高;在

S_WRITE就要進行寫，將dec的8bit資料送出去當writedata，要寫32次，每次寫完就要再回到S_QUERY_TX再等待寫。



控制訊號:

waiting = avm_waitrequest

ready = (bytes_counter_r == 7’d31)

not_ready = (bytes_counter_r < 7;d31)

ready2 = (bytes_counter_r == 7’d30)

not_ready = (bytes_counter_r < 7;d30)

readable = (avm_address_r == RX_BASE)

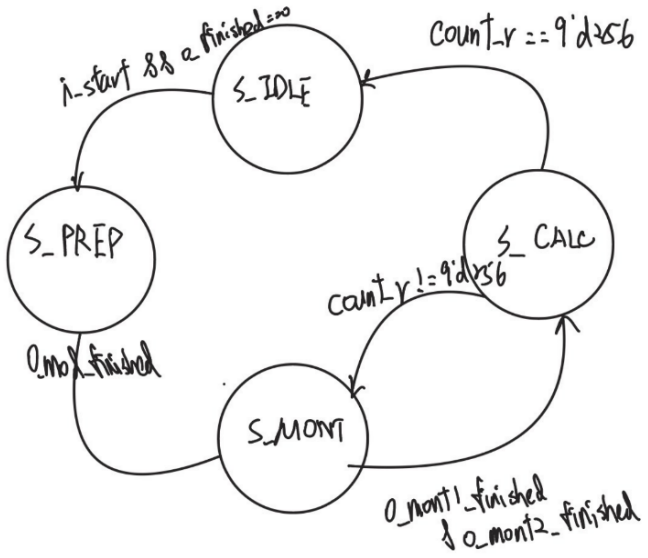
writable = (avm_address_r == TX_BASE)

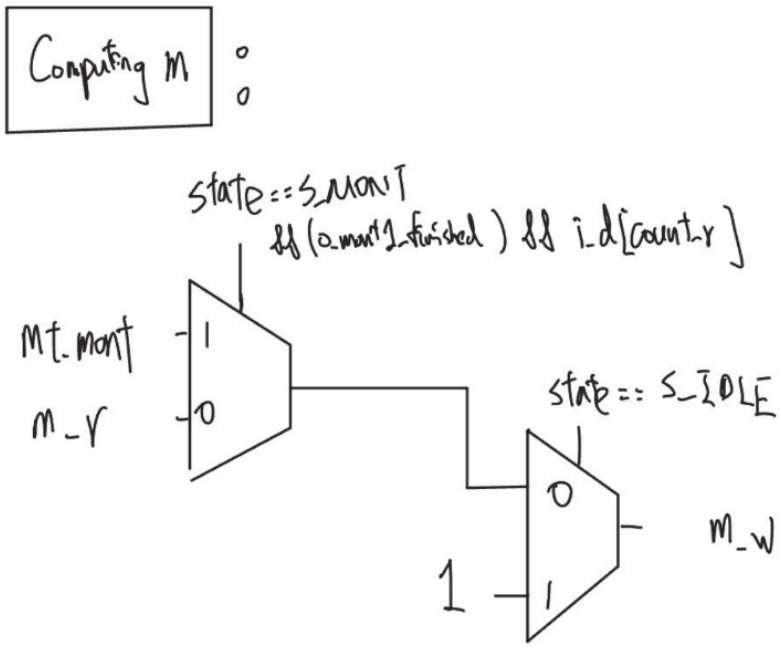
status = (avm_address_r == STATUS_BASE)

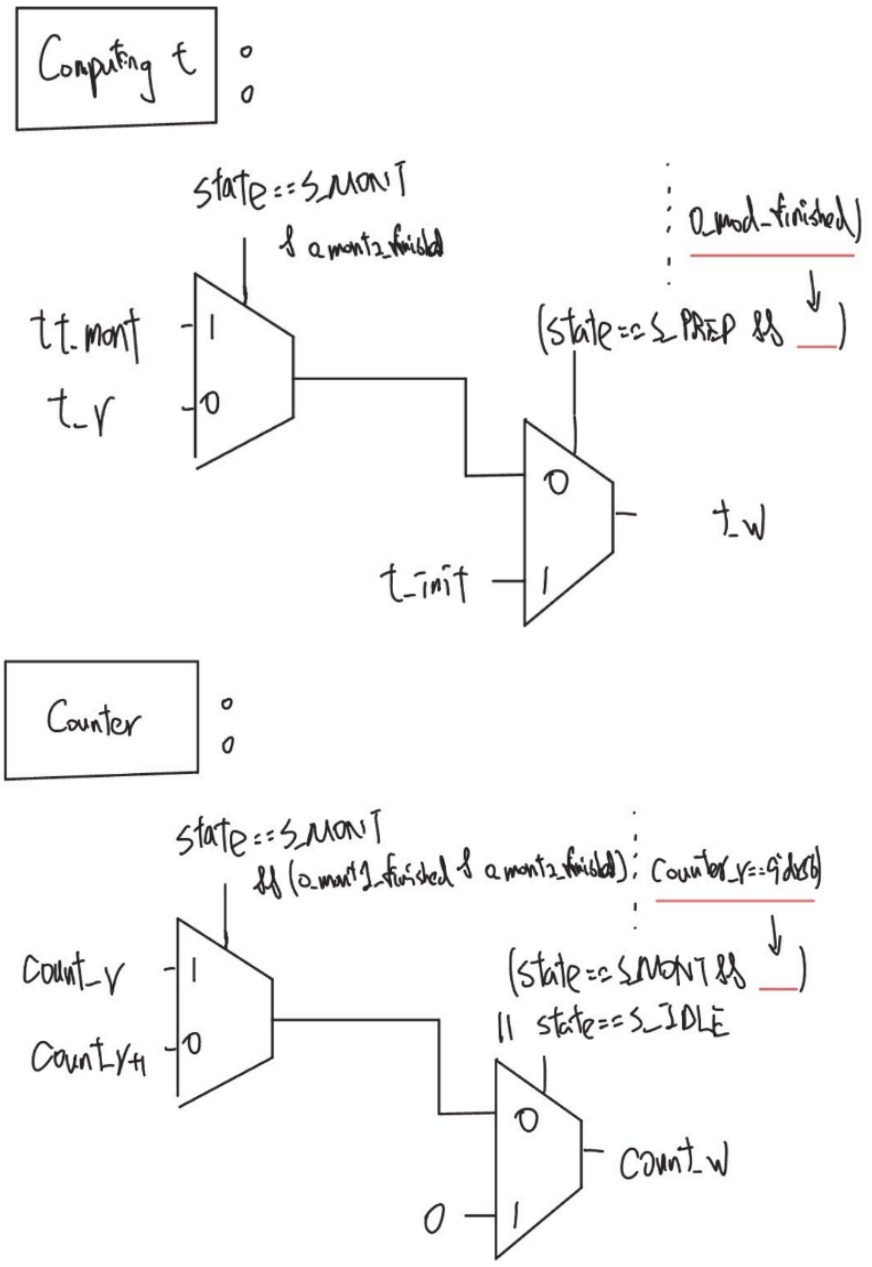
Rsa256Core:

我們的core總共有4個state，分別為S_IDLE、S_PREP、S_MONT、S_CALC。其中S_IDLE為initial state，當(i_start == 1 && o_finished == 0)時會進入S_PREP。S_PREP會等待ModuleProduct去計算initial t，並等待o_mod_finished == 1，表示ModuleProduct算完了。此時，會拉高i_mont_start，並進入S_MONT。S_MONT這個state負責計算

Montgomery Algorithm, 包含m*t 和 t*t兩個phase, 分別為Computing m和Computing t兩個block, 但兩者會平行計算。在這個state, Rsa256Core會等待o_mont1_finished和o_mont2_finished拉高成high, 再進入下個state—S_CALC。S_CALC是去判斷是否counter已經數到9’d256, 若數到9’d256則跳會S_IDLE, 反之回到S_MONT繼續更新m和t。

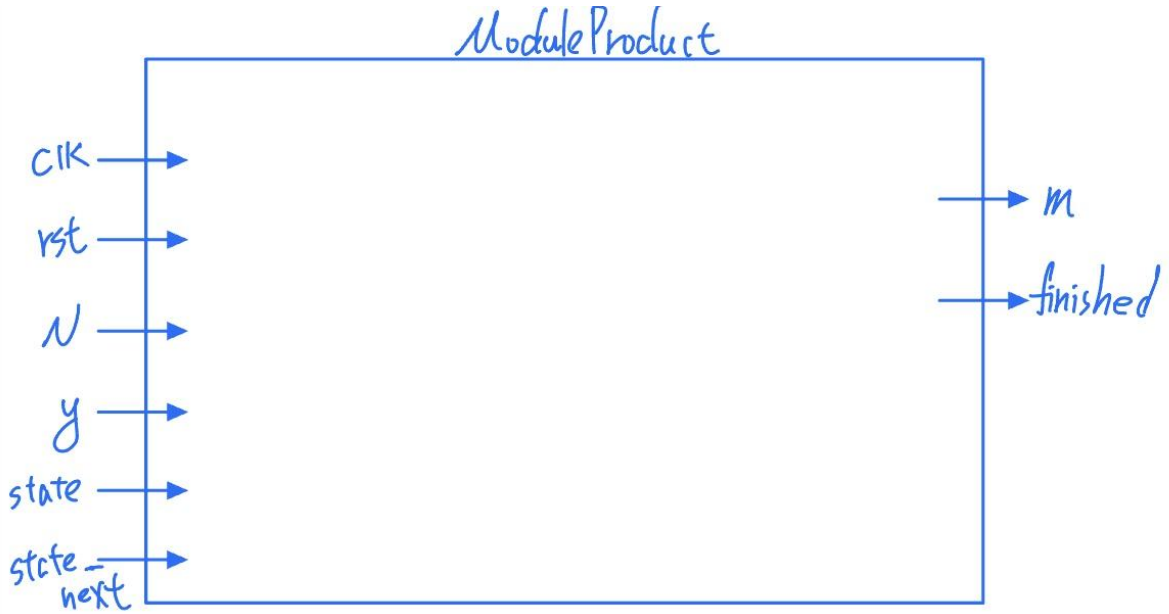


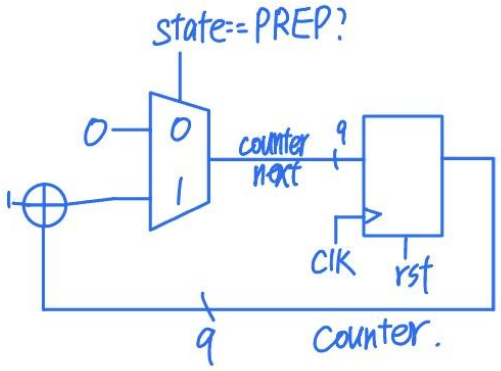


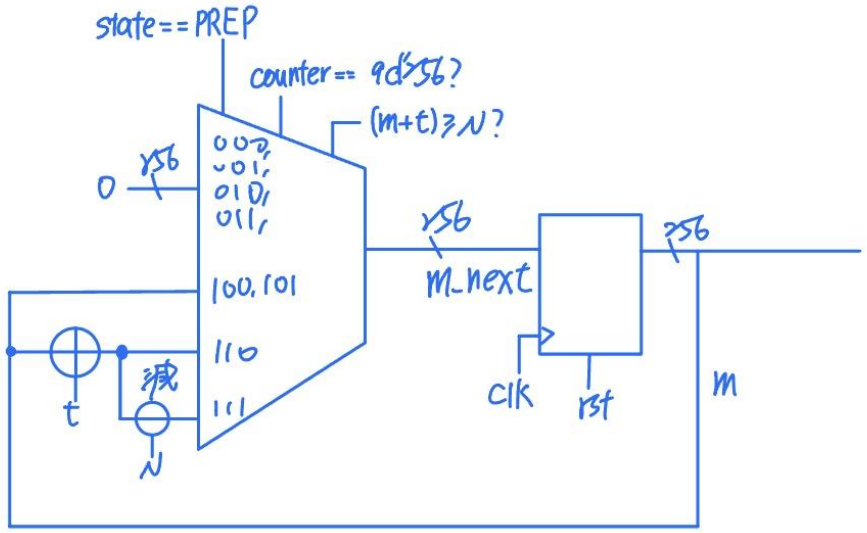


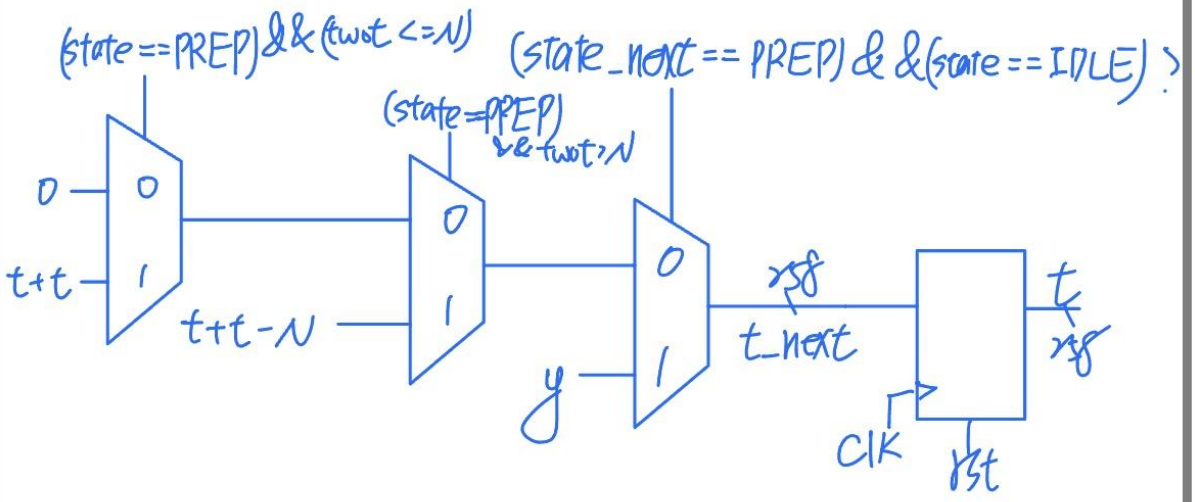
ModuleProduct:

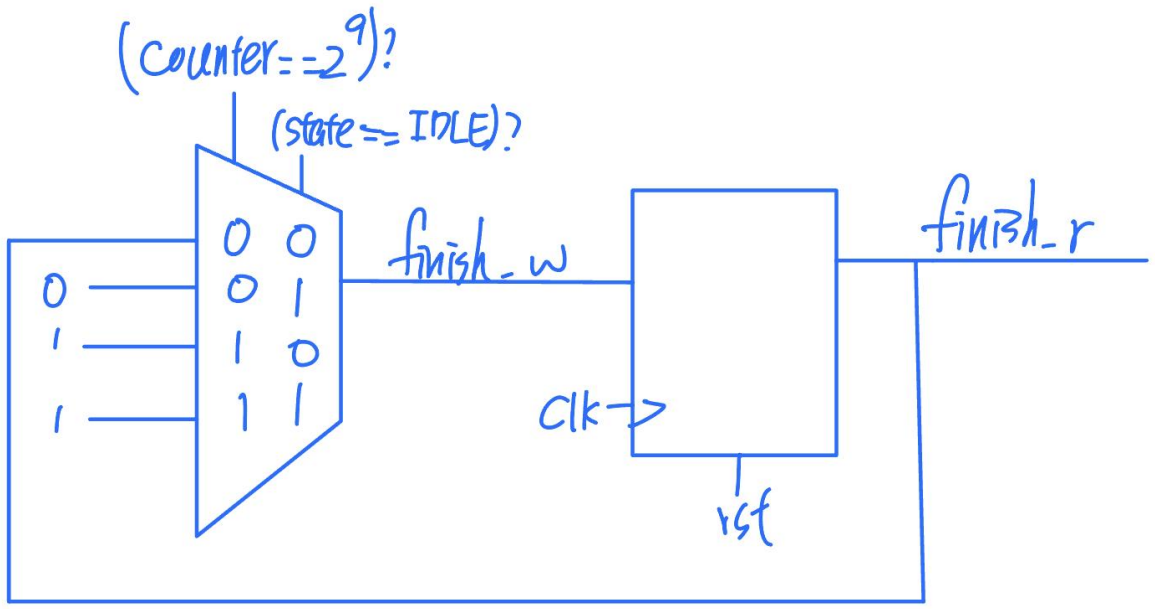
在此module裡面，我們進行(y * 2^256) mod(N) 的計算，其中y為input資料i_a。我們會有從Rsa256Core進來的state訊號，若state變成PREP時，就會先初始化t的值變y，並且會用一個counter從0開始跑到256，每個cycle都會更新t的值，若t + t > N，t就賦值成t + t - N，反之，賦值成t + t；而m的值在reset時被初始化成0，在counter跑到256時，檢查如果m + t >= N的話，m就賦值成m + t - N，反之，賦值成m + t，此時finish也會拉高，並在下一個cycle再output出去。


Module Product
clk
rst
N
y
state
state_next
m
finished


state==PREP?
0
0
1
counter next
q
clk
rst
q
counter.

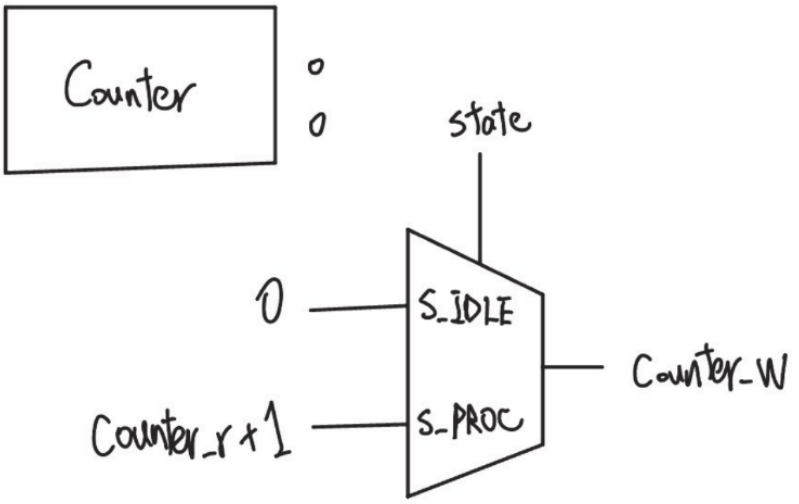

state == PREP
counter == 9'd256?
(m+t)>N?
0
256
000,
001,
010,
011,
100, 101
110
111
减
t
N
256
m_next
256
clk
rst
m


(state == PREP) && (t+t <= N)
(state_next == PREP) && (state == IDLE)
(state == PREP) && (t+t > N)
0
t+t
t+t-N
y
9
t_next
clk
rst
t

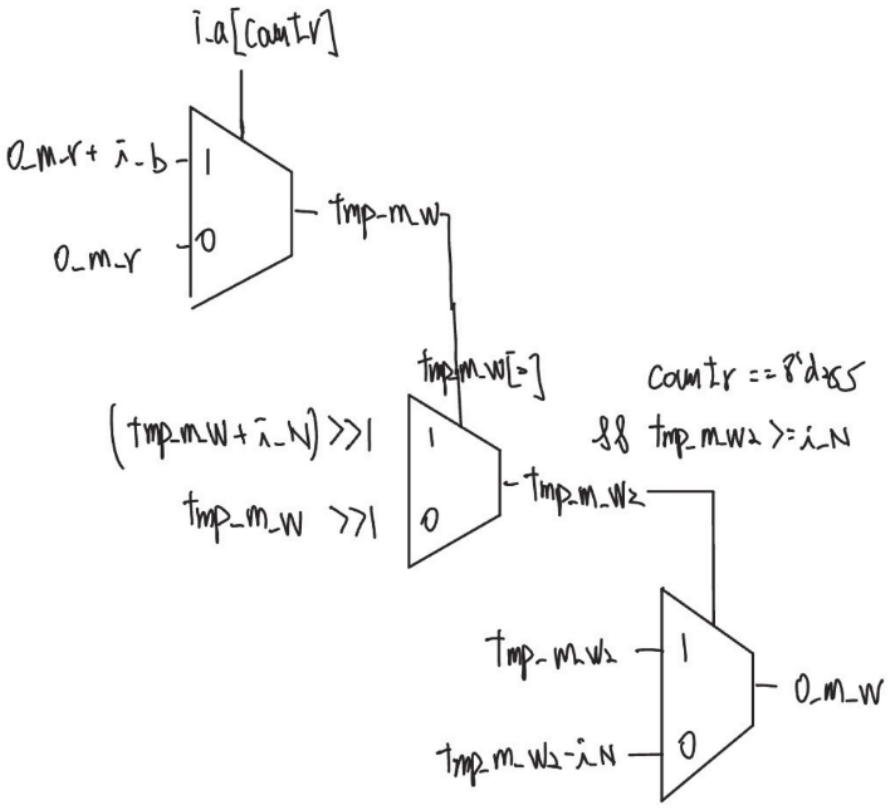


MontgemoryAlgorithm:

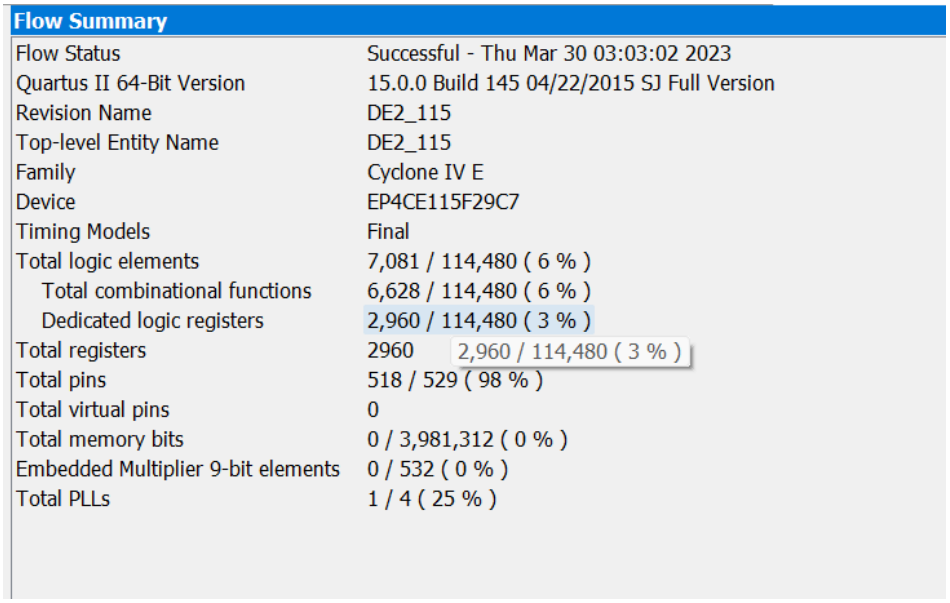
在此module裡面我們實現Montgomery algorithm，有兩個state，分別為S_IDLE和S_PROC，另外，為了計算最終的output o_m，我們用tmp_m_w和tmp_m_w2訊號來作運算。一開始reset時會處於S_IDLE的state，tmp_m_w, tmp_m_w2, o_m都會等於0，而當從Rsa256Core送進來的i_start控制訊號拉高時，就會開始運送，state跳到S_PROC，在S_PROC時，用一個counter count從0跑到255，在每一個cycle時，會看看送進來的input訊號i_a的第i個bit是不是1，其中i為counter的值，若是1的話，tmp_m_w 賦值為o_m + i_b，否則為o_m，接著再看tmp_m_w的第0個bit是不是1，也是看是否為odd，若是的話則賦值為(tmp_m_w + i_N) >> 1，否則為tmp_m_w >> 1。當count跑到255時，檢查tmp_m_w2是否>= i_N，若是的話o_m_w就賦值成tmp_m_w2 - i_N，否則為tmp_m_w2，此時finish訊號也會拉高，並在下一個cycle輸出o_m的值。



Montgomery Algorithm



4. Fitter Summary



Flow Summary

Flow Status	Successful - Thu Mar 30 03:03:02 2023
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115
Top-level Entity Name	DE2_115
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	7,081 / 114,480 (6 %)
Total combinational functions	6,628 / 114,480 (6 %)
Dedicated logic registers	2,960 / 114,480 (3 %)
Total registers	2960 2,960 / 114,480 (3 %)
Total pins	518 / 529 (98 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	1 / 4 (25 %)

5. Time analyer

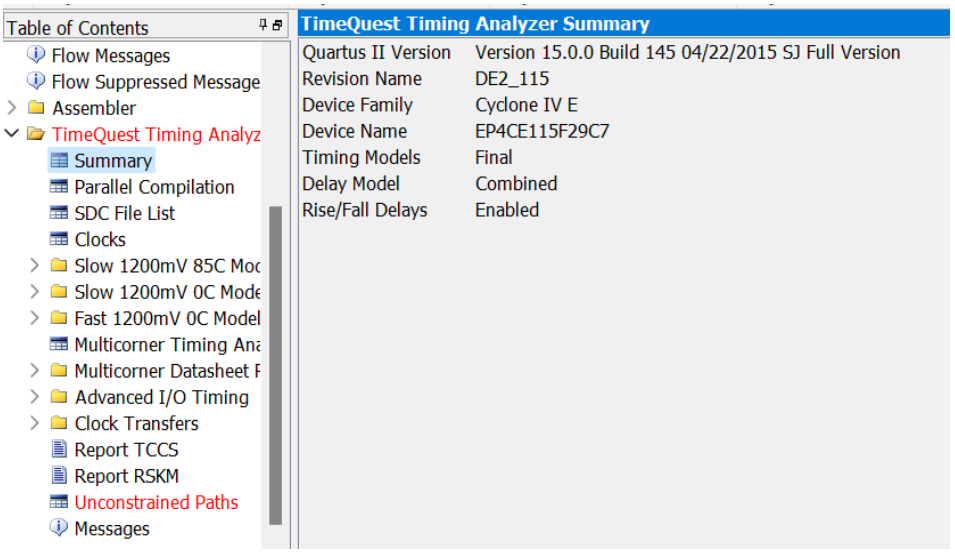


Table of Contents

- Flow Messages
- Flow Suppressed Message
- Assembler
- TimeQuest Timing Analyz
  - Summary
  - Parallel Compilation
  - SDC File List
  - Clocks
  - Slow 1200mV 85C Moc
  - Slow 1200mV 0C Mode
  - Fast 1200mV 0C Model
  - Multicorner Timing Ana
  - Multicorner Datasheet F
  - Advanced I/O Timing
  - Clock Transfers
  - Report TCCS
  - Report RSKM
  - Unconstrained Paths
  - Messages

TimeQuest Timing Analyzer Summary

Quartus II Version	Version 15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115
Device Family	Cyclone IV E
Device Name	EP4CE115F29C7
Timing Models	Final
Delay Model	Combined
Rise/Fall Delays	Enabled

6. 問題與心得

1. Rsa256Wrapper的module在讀或寫訊號時, 一次讀8個bit, 總共讀32次, 本來以為這32次可以連續讀寫, 也就是寫成每過一個cycle就讀寫一次, 但發現這每一次讀寫之前都還要再等avm_waitrequest變成0以後才可以再做一次讀寫。

2. 這次是我第一次打較大型的程式, 讓我發覺保持良好的coding style很重要, 其一是可讀性比較高, 而且也比較容易debug。
3. 此外, 透過這次程式, 也讓我更熟悉利用 i_start 和 finish訊號去控制每個module的運作。
4. 雖然助教在lab講解時就有提醒大家, 但我們一開始測驗測資時也忘記 reset, 因此產生許多亂碼。但是, 後來發現只要reset後, 就可以產生正常的output了。

5. 這次的實驗大且難了許多, 花了比lab1多了很多時間, 希望lab3可以安然度過。
6. ModuleProduct中的 t 沒有給多一點空間, 導致t + t時overflow, 下次要注意overflow的問題。