Team09 Lab2 Report

b11901003 方嘉麟
b11901091 鄭淳芸
b11901148 李承彥

1. File Structure
   team09_lab2
   | - team09_lab2_report.pdf
   | - src/
      | - Rsa256Core.sv
      | - Rsa256Wrapper.sv

2. System Architecture

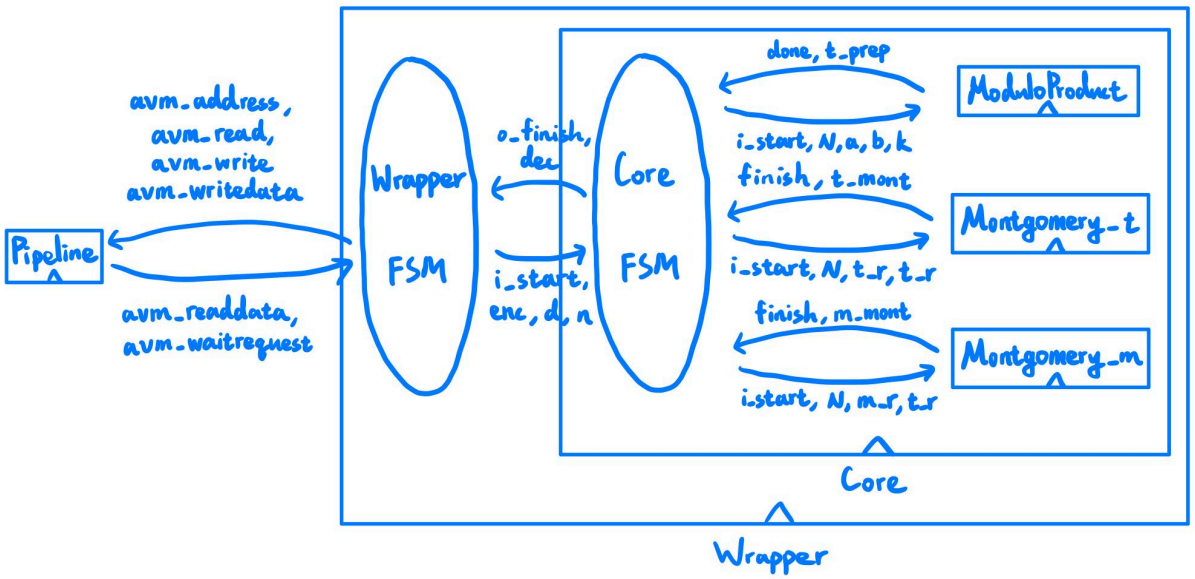


Figure 1: System Architecture

3. Hardware Scheduling
   a. FSM

i. Wrapper

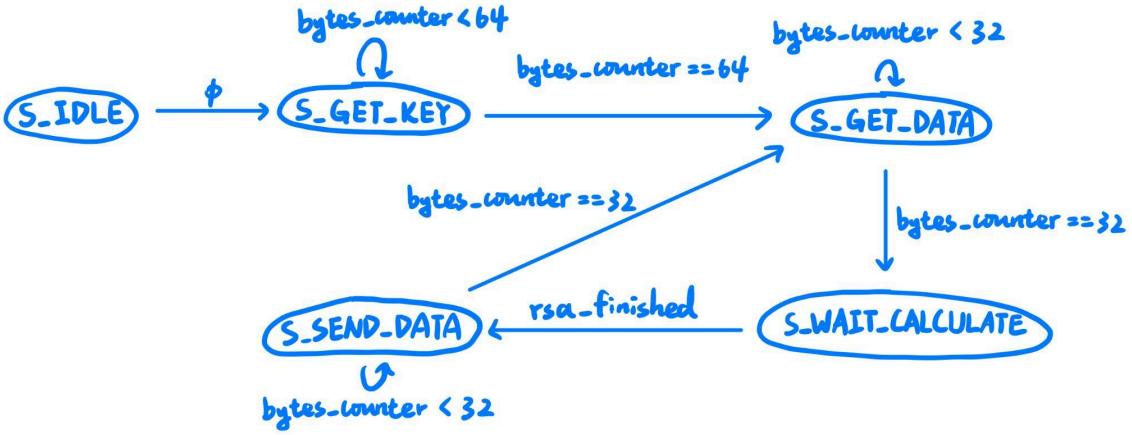
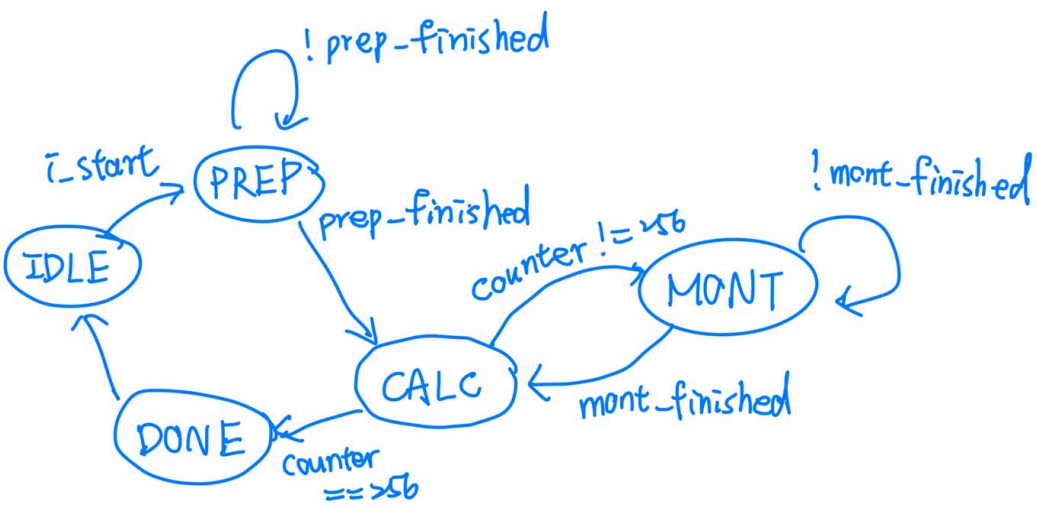


Figure 2: FSM of Wrapper.

ii. Core



IDLE: wait for i_start signal.
PREP: calculate t = y*2^256 (mod N).
CALC: assign result from ModuloProduct and Montgomery to t and m of the next iteration.
MONT: calculate m*t (mod N) and t^2 (mod N).
DONE: output final answer and set o_finished to high.

iii. ModuloProduct

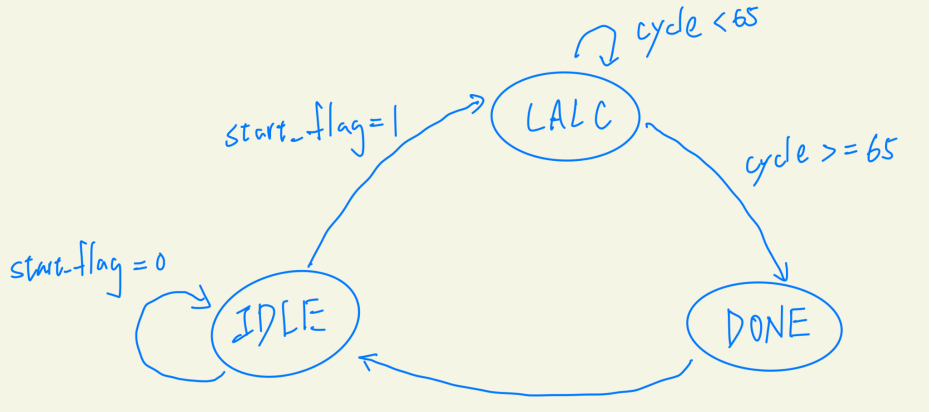


Figure 4: FSM of ModuloProduct.

IDLE : Wait for incoming numbers, raise start_flag if i_start is enabled

CALC: Calculate the result for 256/4 + 1 cycles

DONE: Tell the core the result is ready

iv. Montgomery

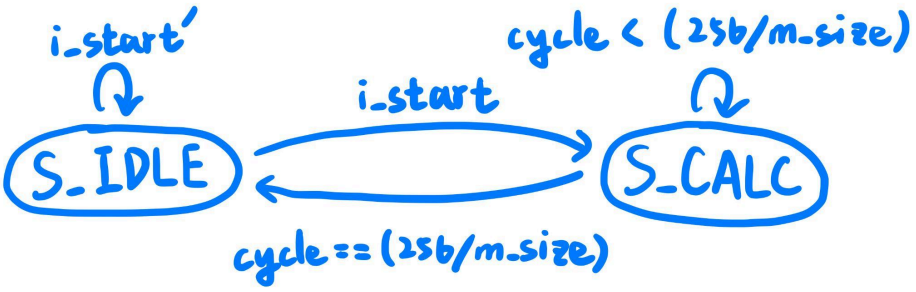


Figure 5: FSM of Montgomery. (m_size: the number of operations within one cycle)

b. Algorithm

i. Wrapper

1. *S_IDLE*

   StartRead(*STATUS_BASE*), then go to *S_GET_KEY*.

2. *S_GET_KEY*

   Check if *avm_waitrequest* is low before each operation. If *rrdy_r* is low, check if avm_readdata[*RX_OK_BIT*] is high, if yes, set *rrdy_w* to be high, and StartRead(*RX_BASE*). If *rrdy_r* is high, StartRead(*STATUS_BASE*), and for the first 32 bytes, store them into *n*, for the 33~64 bytes, store them into *d*, and set *rrdy_w* to be low. After repeating 64 times, go to *S_GET_DATA*.

3. *S_GET_DATA*

   Check if *avm_waitrequest* is low before each operation. If *rrdy_r* is low, check if avm_readdata[*RX_OK_BIT*] is high, if yes, set *rrdy_w* to be high, and StartRead(*RX_BASE*). If *rrdy_r* is high, StartRead(*STATUS_BASE*), then store bytes into *enc*, and set *rrdy_w* to be low. After repeating 32 times, go to *S_WAIT_CALCULATE*.

4. *S_WAIT_CALCULATE*

   Wait until *rsa_finished* is high, then store core's output to dec and go to *S_SEND_DATA*.

5. *S_SEND_DATA*

   Check if *avm_waitrequest* is low before each operation. If *trdy_r* is low, check if avm_readdata[*TX_OK_BIT*] is high, if yes, set *trdy_w* to be high, and StartWrite(*TX_BASE*). If *trdy_r* is high, StartRead(*STATUS_BASE*), then store bytes into *enc*, and set *trdy_w* to be low. After repeating 32 times, go to *S_GET_DATA*.

ii. Core

After receiving a high signal of *i_start* in state IDLE, the state will be switched to PREP. Then, the *start* signal for Modulo_Product will be pulled high for 1 cycle. After receiving the done signal from Modulo_Product, the state will be switched to CALC. This is when the value of *m* and *t* for the next iteration is set. Then the state is put to MONT to calculate Montgomery, and the *start* signal for Montgomery will be pulled high for 1 cycle. A counter ensures that Montgomery is calculated exactly 256 times. After 256 times of calculation, we will obtain the answer. The state is then switched to DONE, during the cycle the decoded value will be at the output and *o_finished* signal will be set to high.

iii. ModuloProduct

After receiving a high signal of start in state IDLE, start_flag will be pulled high until finishing the calculation, and the state will be switched to CALC. During state CALC, the module will perform 4

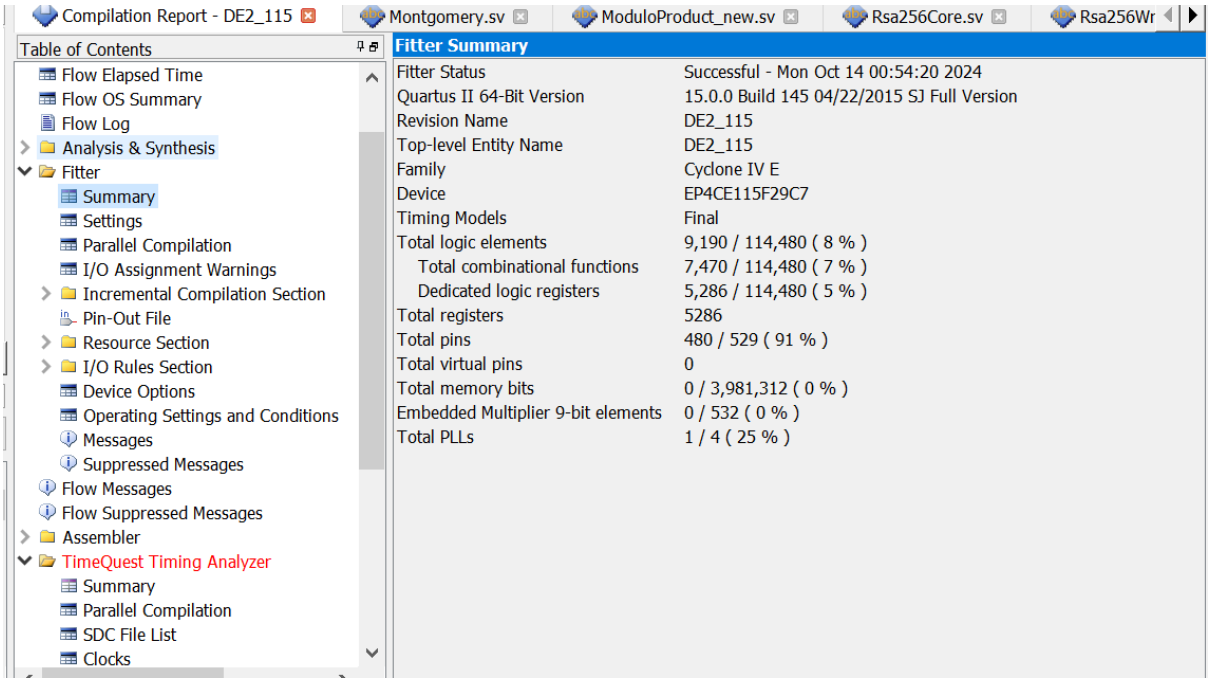
iterations. In each iteration, a bit of *a* will be checked and conduct the following algorithm as mentioned in the lecture document.

Implementation of the algorithm will be described as follows: Initially, both m and t are set to zero. Because we need to perform 4 iterations per cycle, wires a.k.a. temp_m and temp_t are needed to store the temporary result of m and t. These wires are also initialized to 0. In the first 3 iterations of a cycle, temporary results of m and t will be stored in the array of temp_m and temp_t, and at the end of the fourth iteration, m and t will be updated to be the same as temp_m and temp_t respectively. Since there are 257 bits to be checked, the module will keep calculating for 65 cycles. In the end, *done* will be pulled high to tell the core that *ModuloProduct* has finished the whole process.

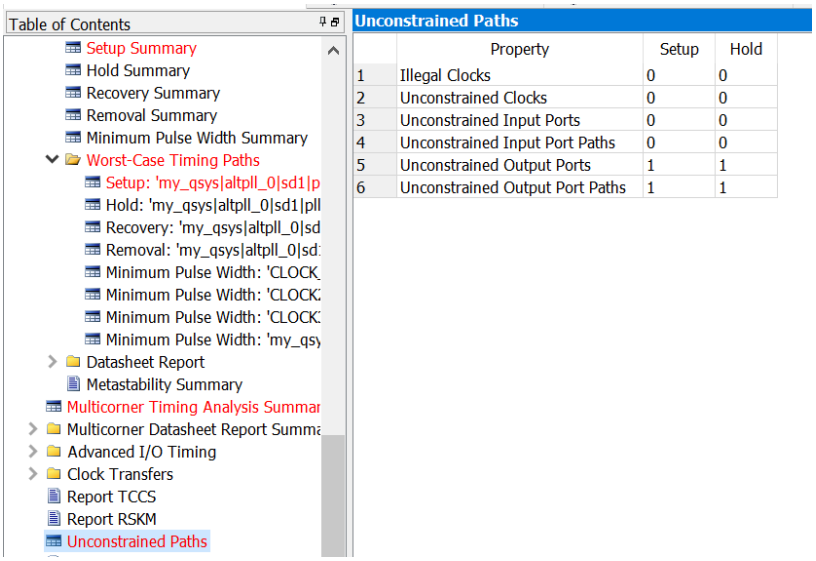
iv. Montgomery

After *i_start* is high, the state becomes *S_CALC*. To improve the generalizability, we use a variable *m_size* to set the number of operations within one cycle (so the number of needed cycles reduces to 256/*m_size*). Then for each *cycle* and for each *i* in *m_size*, if *a*'s (*cycle* + *i*)-th bit is one, tmp1[*i*] is assigned to be m[*cycle* + *i* - 1] + *b* (m[*cycle* + *i* - 1] otherwise), and if tmp1[*i*] is odd, tmp2[*i*] is assigned to be tmp1[*i*] + N (tmp1[*i*] otherwise), then m[*cycle* + *i*] is assigned to be tmp2[*i*] >> 1. In our code, we set *m_size* to be 4, then in watch cycle *m_r* -> tmp1[0] -> tmp2[0] -> tmp3[0] -> … -> tmp1[3] -> tmp2[3] -> tmp3[3] -> *m_w*. After 256/*m_size* cycles, return *m_r*, and the state becomes *S_IDLE*.

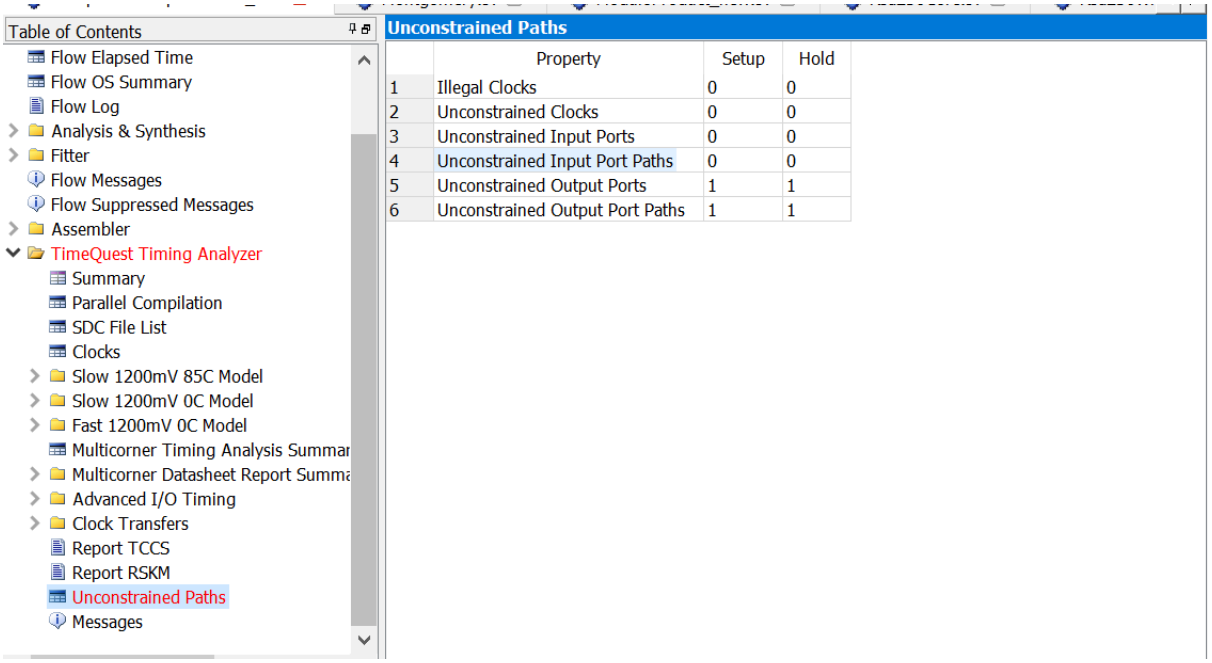
4. Fitter Summary 截圖



5. Timing Analyzer截圖



一個cycle進行4次運算的版本(加速版)
雖然有紅字，但確實是可以執行的



一個cycle算一次的版本(基礎版)

6. 遇到的問題與解決辦法
   a. Quartus編譯：我們的project不知為何compile了將近一個小時，起初compile停住時我們以為是code寫壞了，但檢查waveform都遲遲找不出問題。後來我們決定放著讓電腦compile然後去睡覺，結果隔天發現compile成功，並且功能也沒問題，因此知道之後要有耐心一些，┌應且實際上網查也有看過compile

7. Bonus

如前所述，我們針對ModuloProduct及Montgomery部分進行了運算的加速，在這兩者為運算耗時主要來源下，解密速度可以為平時的4倍。

8. 心得

本次實作過程相較Lab1多耗時不少，需要實作更完整的架構。但藉由這個過程，學習到的內容明顯比Lab1多上不少，也有機會了解Qsys跟古老溝通協定的模擬過程。本次實作我們也體會到運算速度與硬體面積的tradeoff，並發現這對compile耗時的顯著影響。整體來說，開發過程除了權衡面積與耗時，實作時的耐心也很重要，睡一覺可能問題就解決了。