LAB2

1. File structure

DE2_115.sv — rsa_qsys.v — Rsa256Wrapper.sv — Rsa256Core.sv

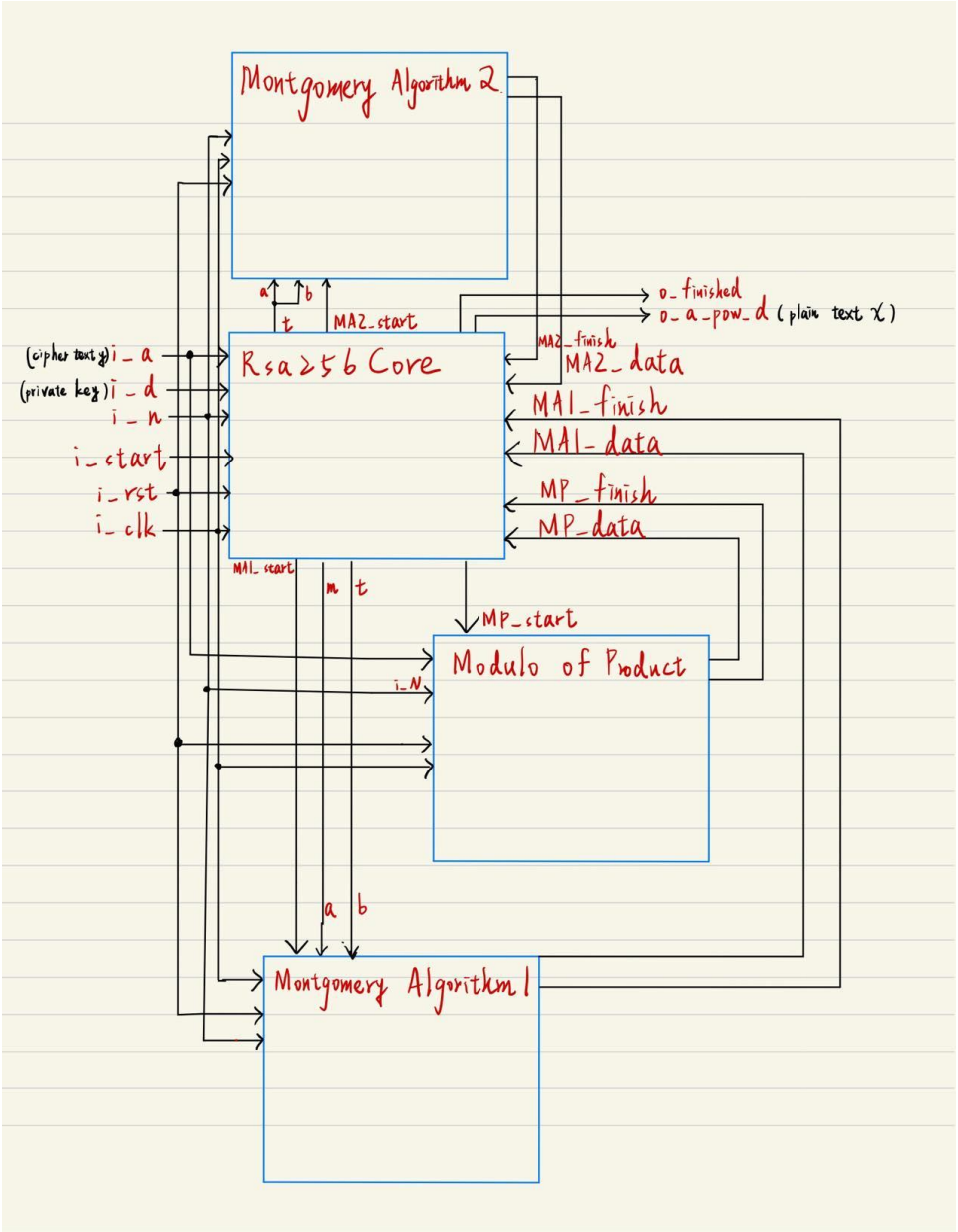
Rsa256Wrapper.sv : controller of RS232 protocol, read & write data and check bits.

Rsa256Core.sv : RSA256 decryption algorithm includes Montgomery algorithm and Modulomfo of Product.

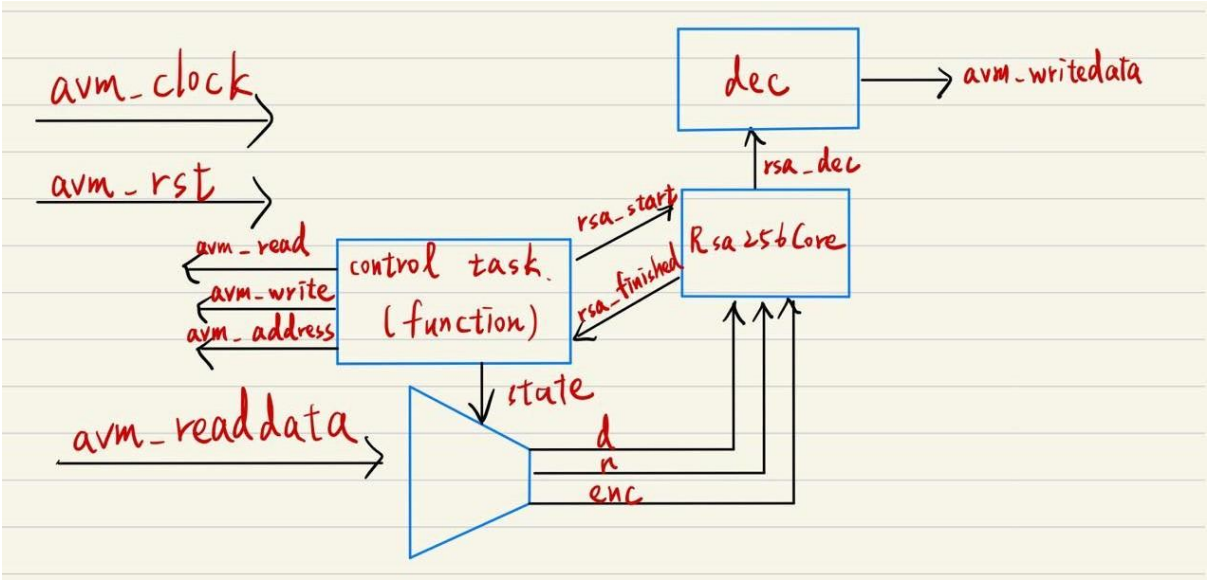
rs232.py : transmit key and encoded file parts to FPGA,then receive decoded file parts from FPGA and print them onto a file. Finally, transmit an ending signal as the end of this transmission.

2. System structure :

Core


Montgomery Algorithm 2
a
b
t
MA2_start
o_finished
o_a_pow_d (plain text ^)
MA2_finish
MA2_data
(cipher text) i_a
(private key) i_d
i_n
i_start
i_rst
i_clk
Rsa256 Core
MA1_finish
MA1_data
MP_finish
MP_data
MA1_start
m
t
MP_start
Modulo of Product
i_N
a
b
Montgomery Algorithm 1

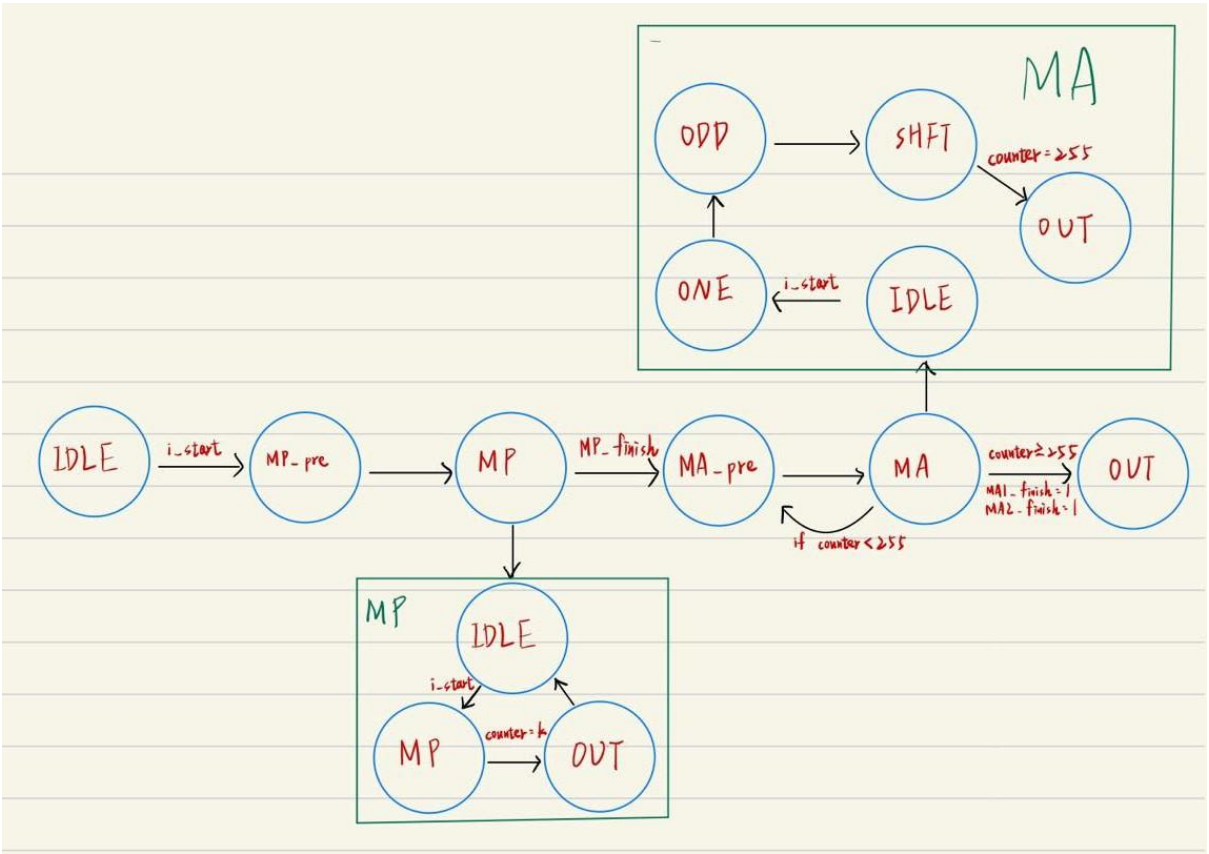
Wrapper



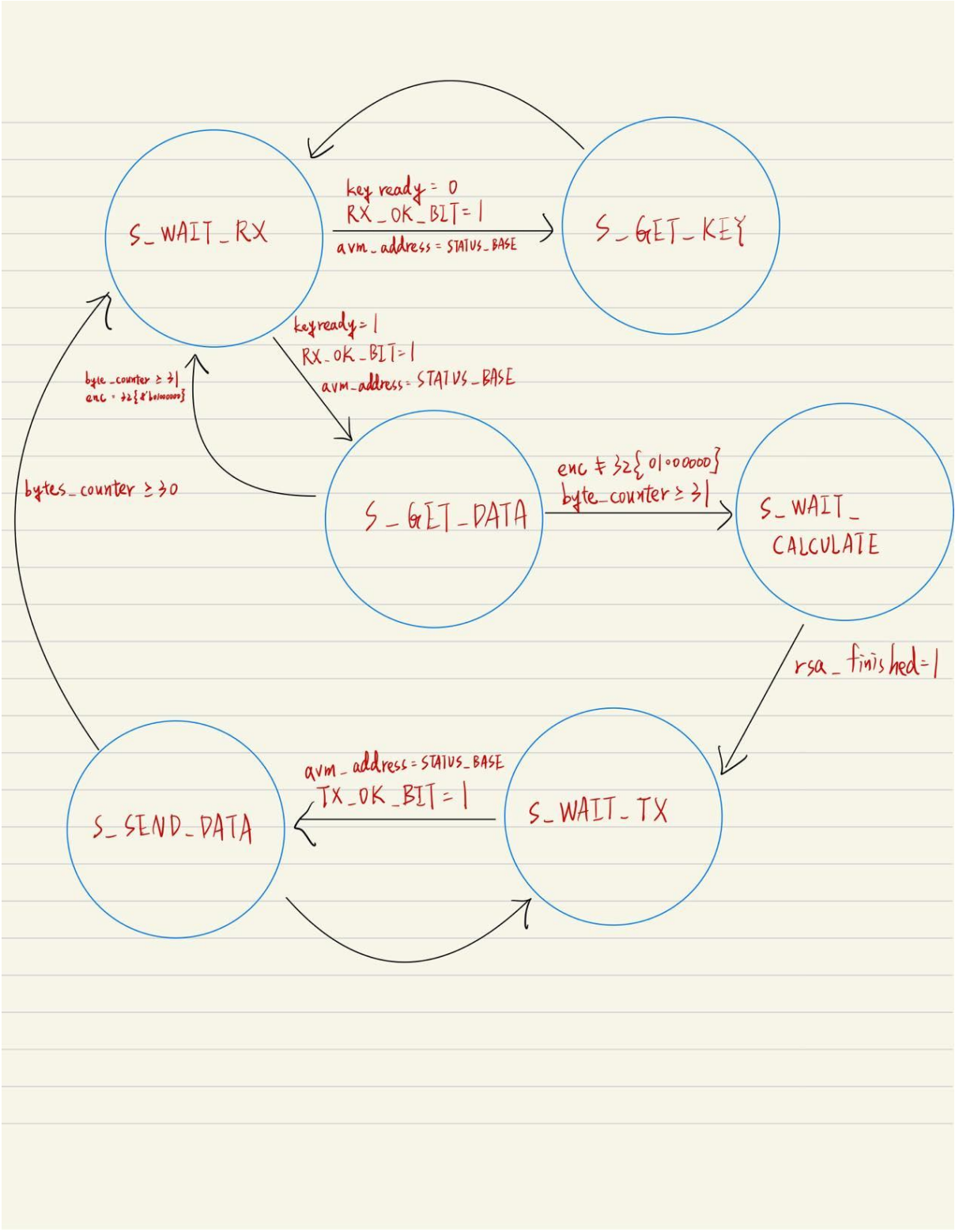
3. Hardware scheduling :

FSM :

Core



Wrapper



4. Fitter summary :

Fitter Summary	
Fitter Status	Successful - Wed Mar 29 22:09:29 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	9,606 / 114,480 (8 %)
Total combinational functions	8,068 / 114,480 (7 %)
Dedicated logic registers	5,714 / 114,480 (5 %)
Total registers	5714
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. Timing analyzer :

Slow 1200mV 85C Model Setup Summary			
	Clock	Slack	End Point TNS
1	my_qsys|altpll_0|sd1|pll7|clk[0]	13.276	0.000

心得

大致上跟隨投影片中的protocol與FSM來設計，再加上一些自訂的額外code，在這裡以介紹額外的部分為主。

(1)如何處理waitrequest訊號

可以把情況分為需要與傳輸資料互動與否來分成兩種情況，而用FSM的state便可以達成。

若需要與傳輸資料互動(寫入或讀取任何資料的state)，則在waitrequest訊號為1時FSM與其他reg都不採取任何行動(所有reg在下個cycle保持相同)

若不需要與傳輸資料互動(例如:wait_calculation state)，則不在乎waitrequest訊號，FSM與其他reg的計算則照原本的演算法繼續計算與更新。

(2)如何處理rx_ready與tx_ready的部分

我另外新增了兩個state: WAIT_RX,WAIT_TX, 在這兩個state中分別檢查rx_ready與tx_ready, 檢查為1時才進入讀取、寫入的state, 否則FSM就持續停在這兩個狀態。
(3)如何正確的寫入與讀取資訊到register
利用counter計算應該讀取的數量, 再用shift功能來依序讀取一個接一個的資料到register中。
(4)如何利用新的protocol來使每次傳輸都不須reset
原本的protocol有一個問題就是, 因為每次電腦都會重新傳送新的key進去, 但板子無法判斷收到的資訊是新的key還是上一次尚未接收完的data, 便會將新的key當成舊的data來繼續解碼, 造成錯誤。為了解決這情況, 我在電腦傳送資料完畢後, 多傳送一個停止訊號(我設定為32個@, 共32個byte), 當板子收到此訊號後, 便清除掉上次key的資料, 等待重新接收新的key(相當於reset), 因此板子每次不用再手動reset;除此之外, 也設定一個新的counter, 當等待資料時經過了50M個cycle(數秒)都沒有新的data的話, 也會強制重置。
(5)除了上述以外, 我也將key是否重置的訊號接到板子的LEDG上, 以此可以觀察是否板子已準備好接收新的一筆key與data, 也比較好debug 新的protocol是否有正常運作。