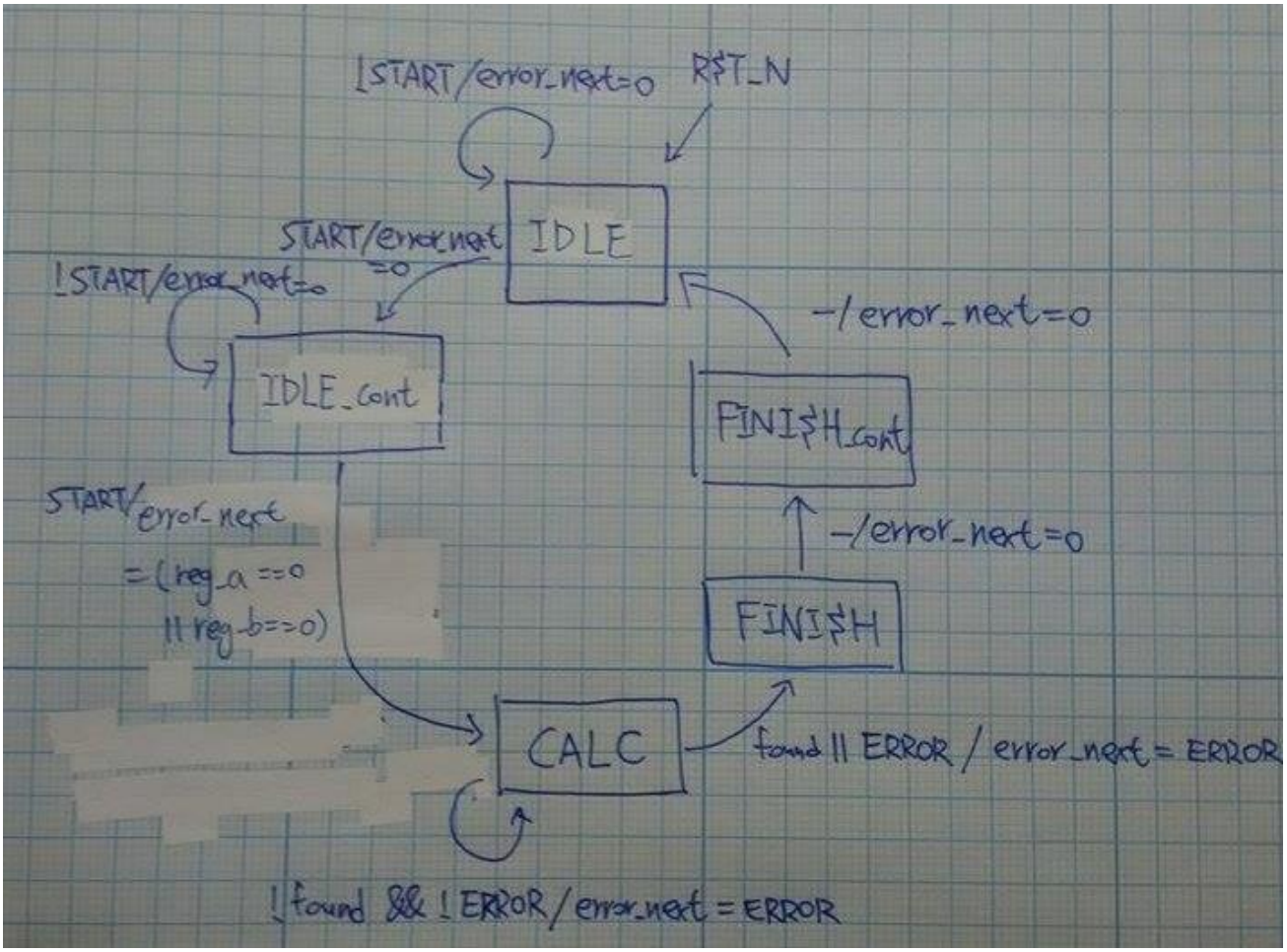
Lab 2: Greatest Common Divisor Engine

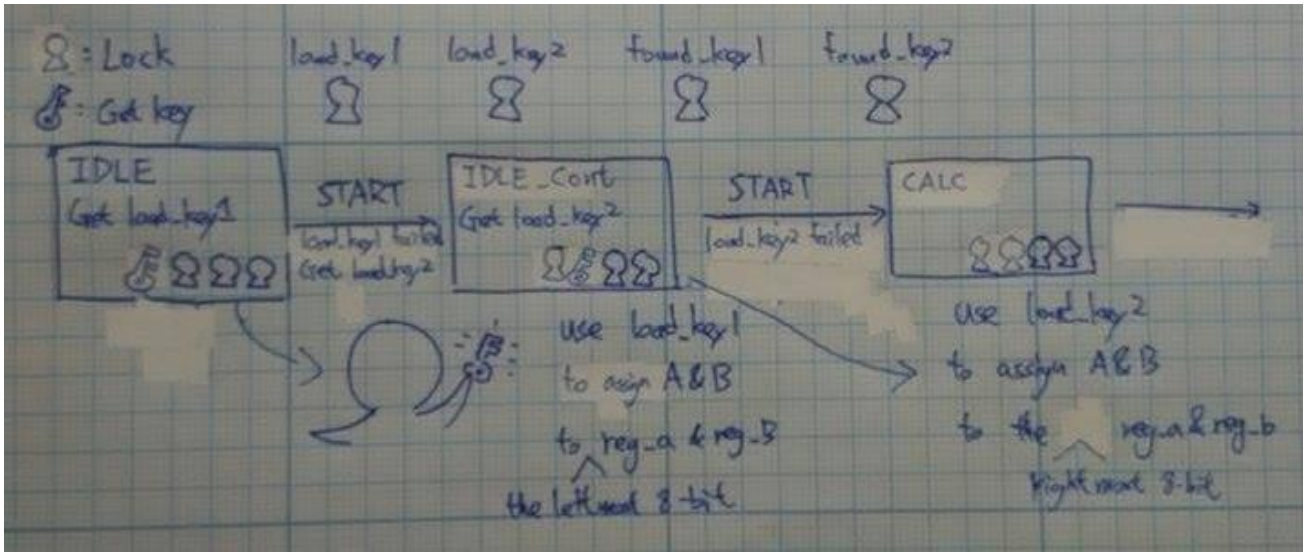
18:30, October 15, 2016

Design Concepts

1. Because of the provided template of PartA, it makes me less effort to complete the design block (by following the provided FSM).
2. PartB need to extend the previous GCD engine to support calculation of two 16-bit positive integers but remaining the bit-number of input A&B and output Y, so I extend the FSM to 5 states as shown below.



3. Refer to the concept of critical section design, using 4 disposable keys (load_key1, load_key2, found_key1 and found_key2) to load and output data.
   Take the procedure of loading data as a example:



4. Different from PartA, the comparison of A or B equal to zero or not must waiting until the A and B is loading complete. So I move these expression to “IDLE_cont” state. The code of “IDLE_cont” state is shown below:

```
IDLE_cont: begin
        DONE = 0;
        load_key1 = 0;
        load_key2 = 1;
            if (START) begin
                state_next = CALC;
                error_next = (reg_a == 0 || reg_b == 0) ? 1'b1 : 0;
            end else begin
            state_next = IDLE_cont;
            error_next = 0;
            end
```

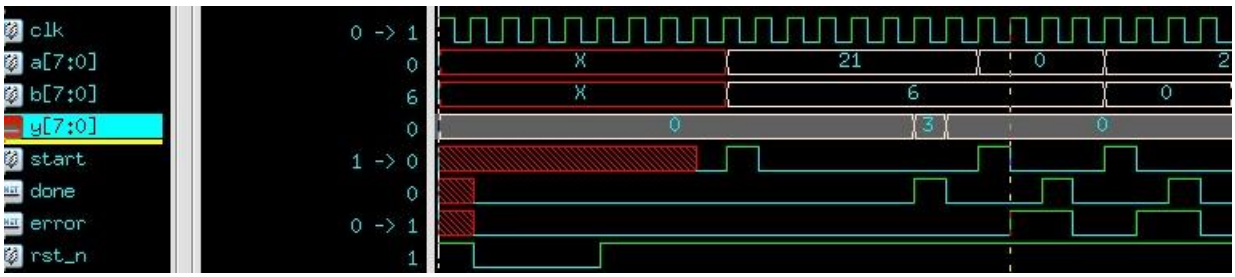
Stimulation Patterns

➢ Part A – My Testing Condition (8bit input range: 0 ~ 255)

A	B	A	B
21	6	21	21
0	6	6	21
21	0	0	0
233	144		

- Part A – Condition Discussion (General)

A	B	Y(expection)	Y(reality)	ERROR(expection)	ERROR(reality)
21	6	3	3	0	0



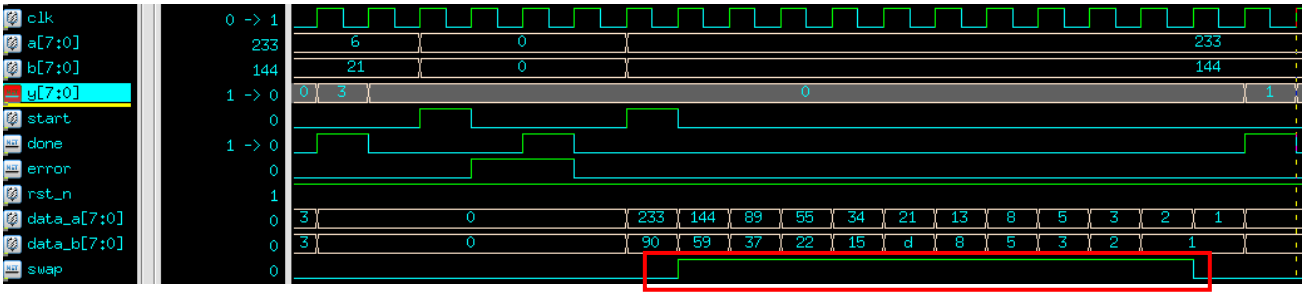
- Part A – Condition (witn zero value)

A	B	Y(expection)	Y(reality)	ERROR(expection)	ERROR(reality)
21	0	0	0	1	1
0	6	0	0	1	1

- Part A – Condition (adjacent Fibonacci number)

  Use Fibonacci number because I want to test the swap function can work or not !

A	B	Y(expection)	Y(reality)	ERROR(expection)	ERROR(reality)
233	144	1	1	0	0

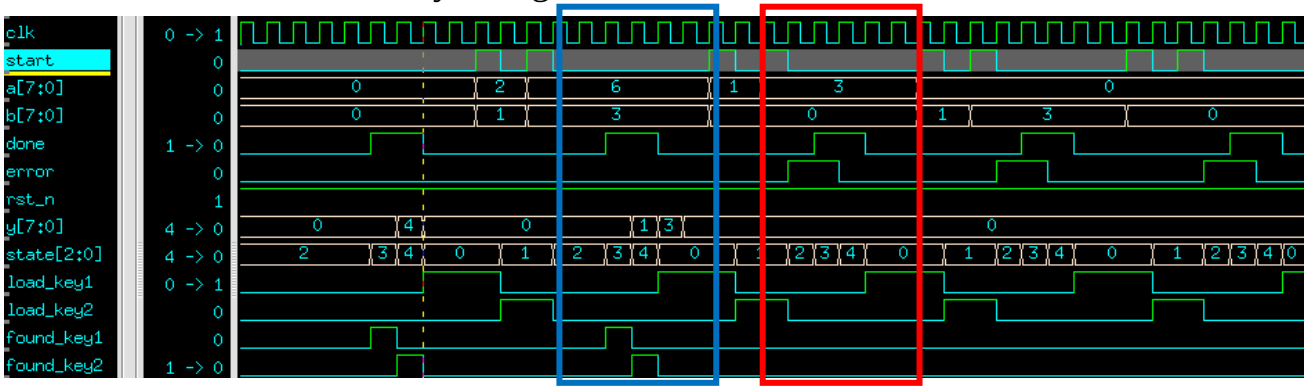


And it swaps as many times as possible as my expectation ! =D

- Part B – My Testing Condition (16-bit input range: 0 ~ 65535)

A	B	A	B
1024	8192	0	0
518	259	9487	9487
259	0	46368	28657
0	259		

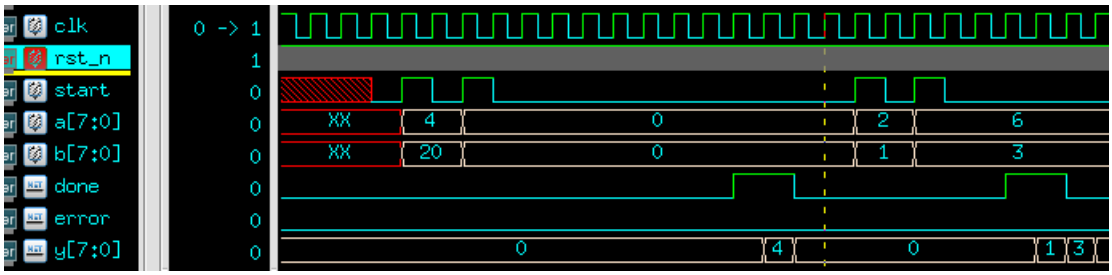
- Part B – The Behavior of Key Using



If get the found_key1&2 then output the result y(blue block), and 0 otherwise(red block).

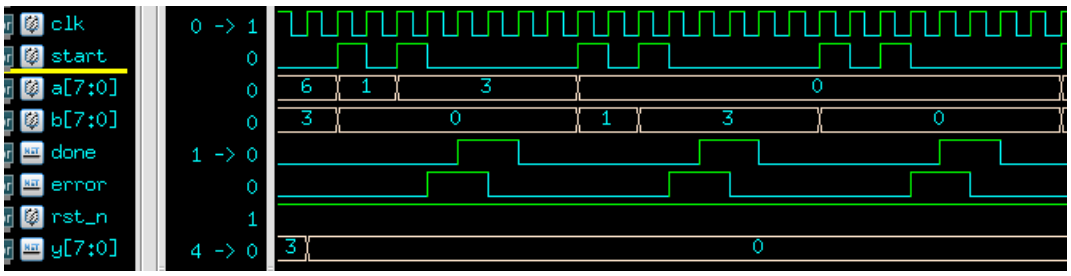
- Part B – Condition (General)

A	B	Y(reality)	ERROR(reality)
16' b0000_0100_0000_0000 (1024)	16' b0010_0000_0000_0000 (8192)	16' b0000_0100_0000_0000(1024)	0
16' b0000_0010_0000_0110 (518)	16' b0000_0001_0000_0011 (259)	16' b0000_0001_0000_0011(259)	0



- Part B – Condition (witn zero value)

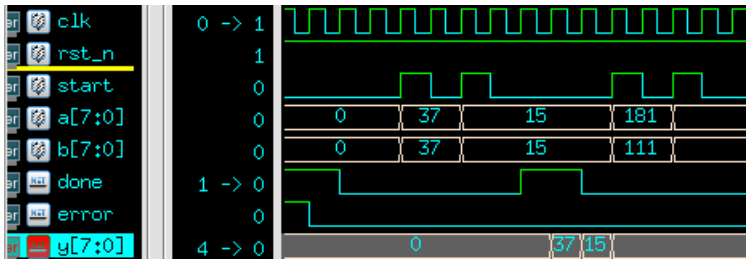
A	B	Y(reality)	ERROR(reality)
16' b0000_0001_0000_0011	0	0	1
0	16' b0000_0001_0000_0011	0	1
0	0	0	1



If one of A or B is zero, then output Y is zero !!!

- Part B – Condition (equal number)

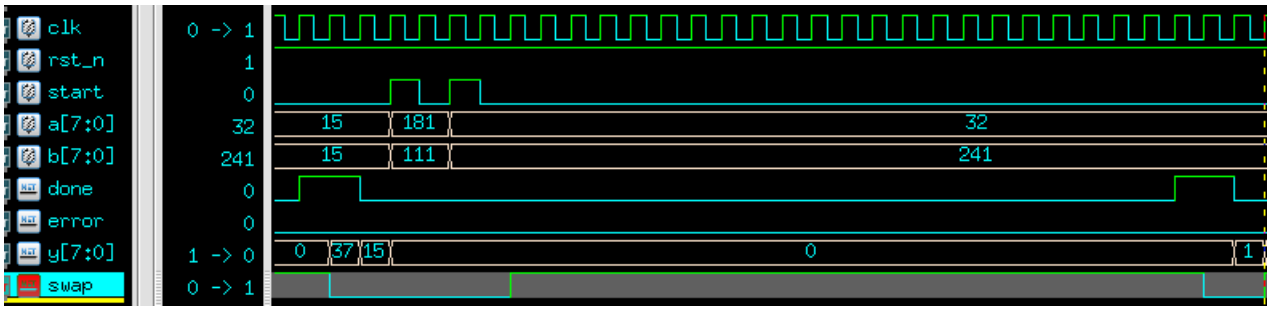
A	B	Y(reality)	ERROR(reality)
16’ b0010_0101_0000_1111 (9487)	16’ b0010_0101_0000_1111 (9487)	16’ b0010_0101_0000_1111 (9487)	0



16’ b0010_0101_0000_1111=9487;
8’b0010_0101 = 37;
8’b0000_1111 = 15;

- Part B – Condition (adjacent Fibonacci number)

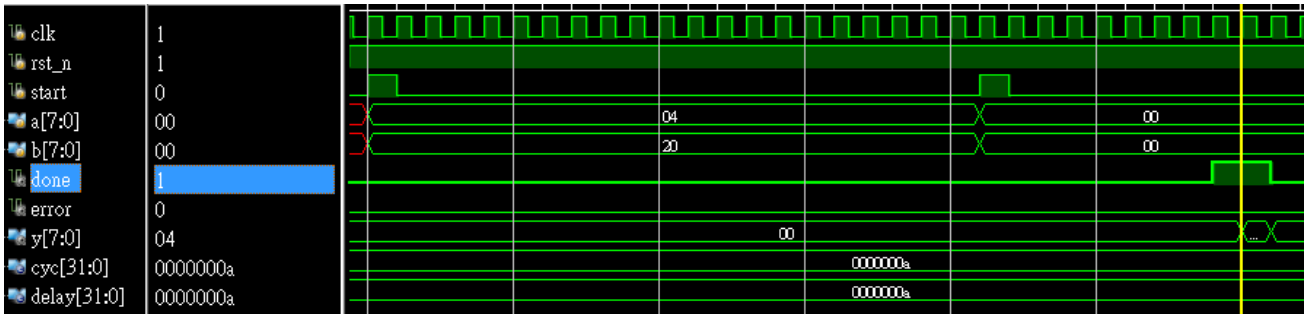
A	B	Y(reality)	ERROR(reality)
16’ b1011_0101_0010_0000 (46368)	16’ b0110_1111_1111_0001 (28657)	16’ b0000_0000_0000_0001	0



16’ b1011_0101_0010_0000 = 46368
8’b1011_0101 = 181;
8’b0010_0000 = 32;

16’ b0110_1111_1111_0001 = 28657
8’b0110_1111 = 111;
8’b1111_0001 = 241;

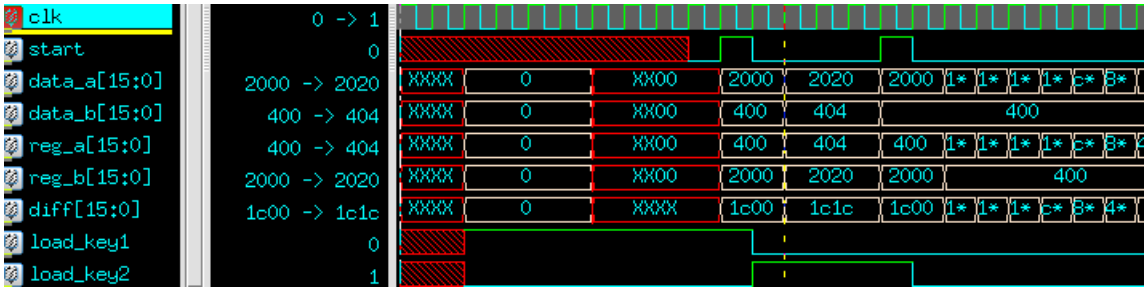
- Part B – Support the start control with separate active cycles



The GCD engine will start calculating when 2 start signals are asserted !!

Lab Review

在只要 dataA 或 dataB 的值有產生變動，diff 就會進行計算然後開始執行 Euclid's Algorithm 來求得 GCD，在思考如何讓 start 訊號允許分散輸入的控制上，key 的設計也幫助到了這點！



在輸入第一個 start 訊號取得 A&B[15:8]的數值後，握有的 key 會從 load_key1 置換成 load_key2，此時從上圖可以看到，雖然 diff 的確做了一次的減法(2020-404=1c1c)，但因為除非輸入下一個 start 訊號才會使 load_key2 失效，因此 dataA 與 dataB 只會一直做讀取 A&B[7:0] 的動作，而不會去進行 GCD 的減法計算，達到等待下一個訊號輸入的結果，而上圖可以看到 start 訊號在次輸入後，load_key2 失效，此時便會把計算求得的 diff 值輸入給 dataA，便開始執行 diff=dataA-dataB 的動作了，便達到了分散時間迴圈觸發的 optional！

Done signal 沒有去做可以允許 separate active cycles 的機制是我認為要做勢必還要增加新的 input 訊號(也有可能是我比較笨想不到不增加訊號怎麼讓程式知道你準備要接收下一段的值了)，但我覺得突然增加新的訊號去控制好像很老梗，因為 start 就展示大概是這樣的概念了，所以我就沒弄了(好啦是偷懶找藉口)

此外這次 Lab 是用 Euclid's Algorithm 來操作，計算時間複雜度為 O(logB)，比起 Binary GCD Algorithm 的計算時間複雜度為 $O((loga+logb)^2)$來說少了很多，因此我想在 diff 產生的方式上動手腳減少執行時間，應該就是這次 optional 需要的，然而利用教授提供的參考 code 去進行改寫，原本有思考過 diff 的產生用 while loop 去做減法，這樣比起原本的寫法每一次 clock cycle 只會做一次減法來說，他可以一次減到需要 swap 或直接顯示結果的地步會比較節省時間，在輸入非零值的時候結果非常滿意，只是輸入零值的時候就失敗了！在 diff 產生的同時因為用 loop 的關係，所以如果輸入 0 值，我的寫法都會造成無止盡的執行，然後就卡住了，或許要在多設定一些參數來控制吧！

最後特別感謝吳岳騏助教在 nWave 的操作上讓我做詢問！