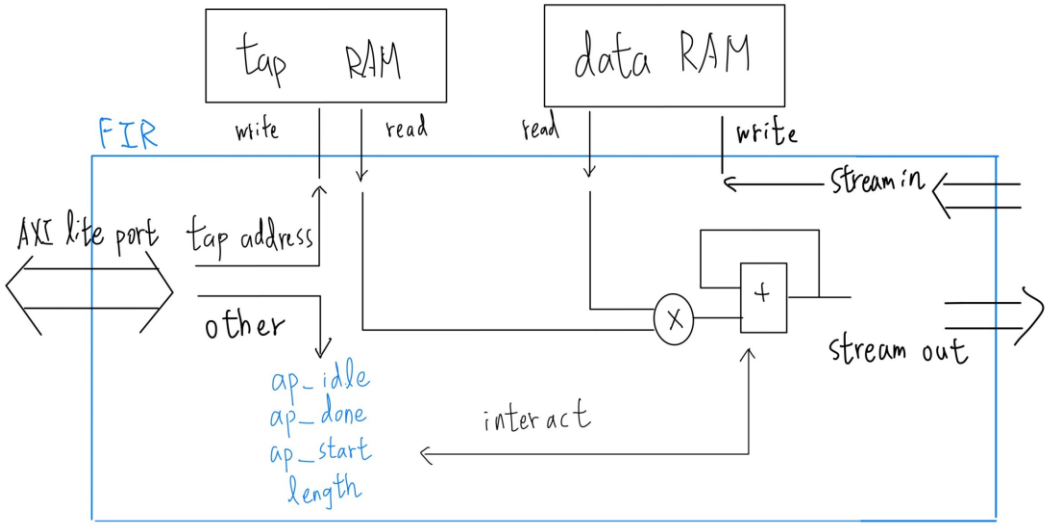
Report

1. Block Diagram

我的電路基本上就是遵照下面這張圖設計的:

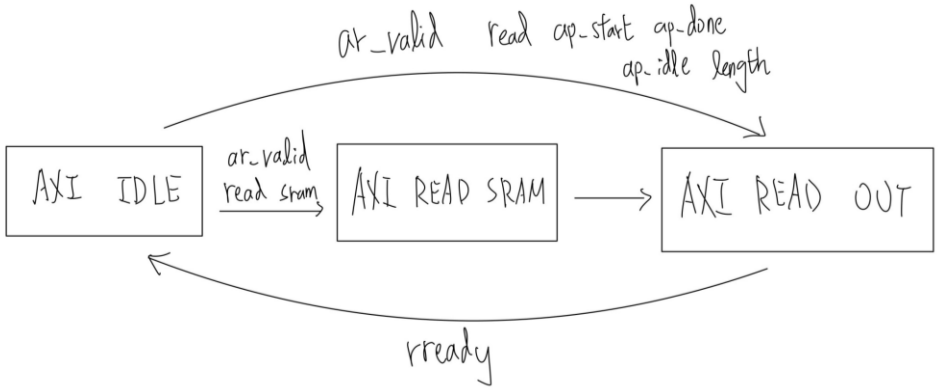


Axi lite 的 read port 跟 write channel 完全獨立，用一個小的 finite state machine 來控制，運算的部分則是用另一個 finite state machine 控制。

2. Describe operation

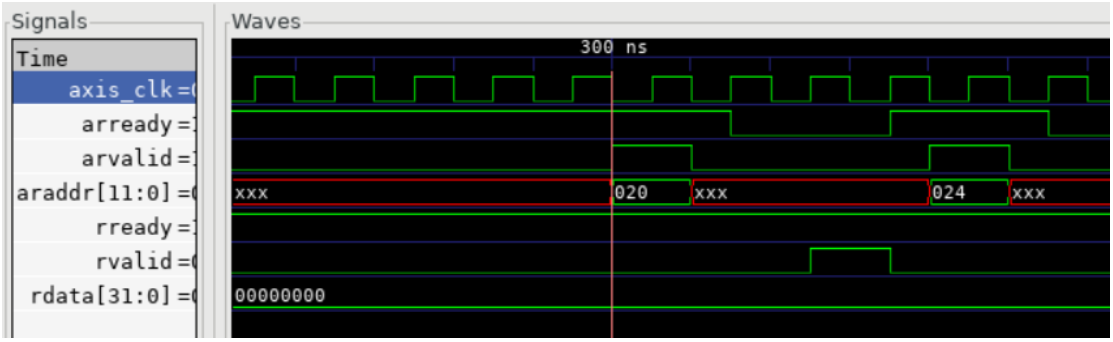
(1) AXI lite read

Finite state 如下:



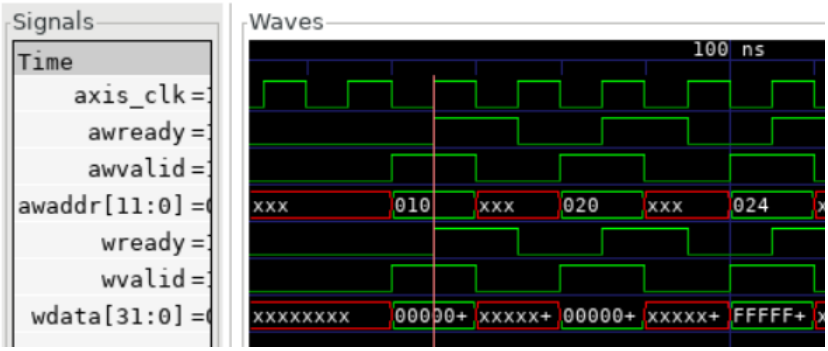
等 ar_valid 到之後，根據 araddr 去判斷要讀的是甚麼，如果是 ap_start、ap_done、ap_idle、length 的話，因為資料放在 register，直接進入 output state 輸出，等到 rready 起來就返回 idle state，如果發現要讀的是 sram 的值，就進入 read sram state，因為從 sram 讀資料會有一個 clock 的 delay，下個 cycle 就進 output state，等 rready 起來就返回 idle state。輸出訊號就是如果再 output state 就輸出資料，並拉高 rvalid，而 arready 則是在 idle state 就拉高。

模擬的波形如下:



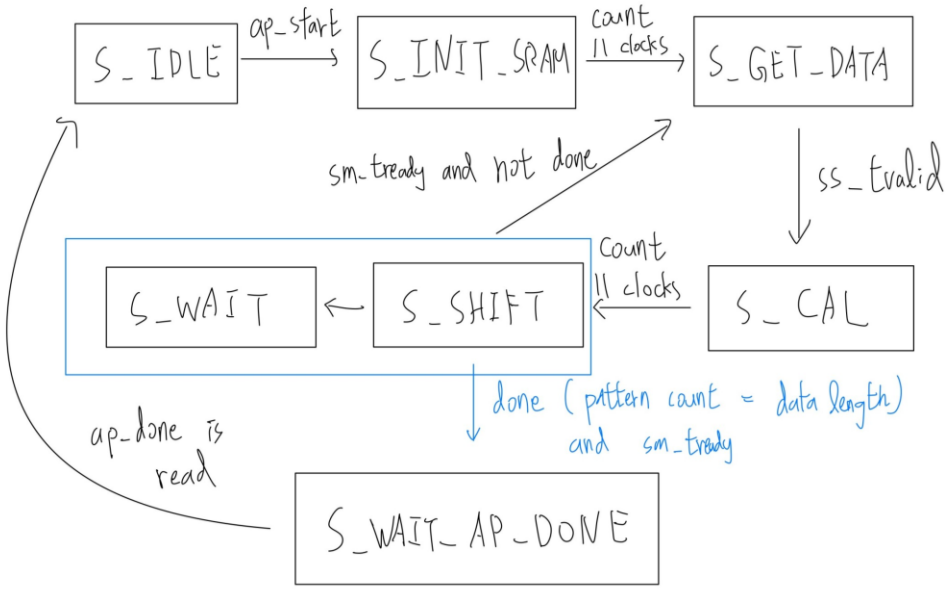
(2) AXI lite write

AXI write 的部分我就沒有用 finite state machine 來控制了，我的 awready 和 wready 要 wvalid 和 awvalid 同時為 1 的下一個 clock 才會一起拉起來，而我再會被寫入的 register 及 sram write enable 的訊號控制都是 wvalid && awvalid && address 為對應的 address，這樣就能實現 axi lite 的 write port 了。模擬的波形如下:



(3) Core design

我的設計主要的 finite state 如下:



當 ap_start 為 1 時，會進入 init sram state 先去初始化 data sram，花費 11 個

clock 把 data sram 寫入 0，接著進入 get data state，透過 stream input 拿取一筆資料，接著進入 calculate state，根據 spec 透過一個乘法器和一個加法器，花費 11 個 clock 算出一筆輸出資料，接著進入 shift state，在 shift data pointer 的同時透過 stream 來輸出剛剛算好的資料，如果 stream output 的 tready 不是 1 的話則進入 wait state 來等 tready，若資料以經送出則返回 get data state 來接著做下一次運算，若是發現已經做了 data length 次的運算，也都 stream 輸出了，就進入 wait ap done state，做各個 register 的 reset 動作同時等待 ap done 透過 AXI lite protocol 被 testbench 讀走，一旦確認被讀走，就進入 idle state，等待下一個 ap start。

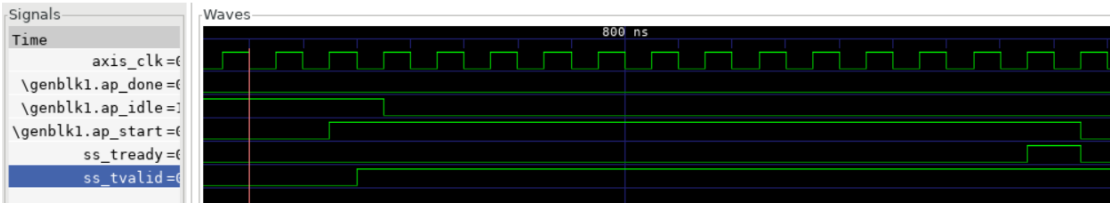
(4) AP register

ap_idle: 如果發現 ap_start 是 1，則變成 0，如果發現 state 在 wait ap done 就代表運算已經做完，ap_idle 改回 1，其他狀況維持不變。

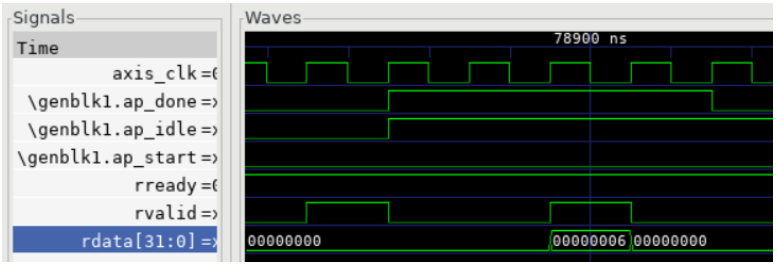
ap_start: 根據 AXI lite 的 flag 和 addr 來寫入，一旦發現有 stream input 被讀取的情況，就變成 0，其他狀況維持不變。

ap_done: 如果發現 state 在 wait ap done，代表運算已經完成，變成 1，如果發現 state 在 idle state，則變成 0，其他狀況維持不變。

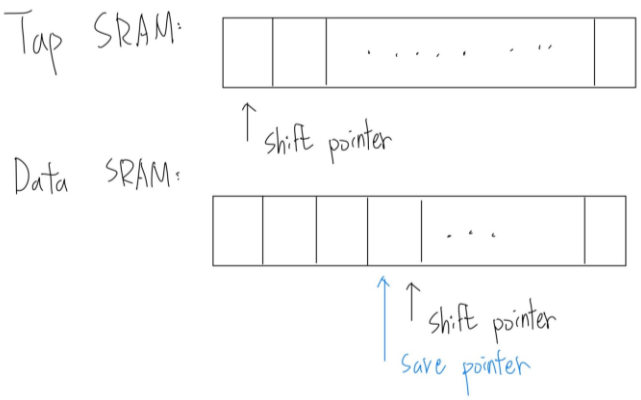
ap start 被 program 的波形:



運算結束後 ap_done 的波形:



(5) Shift SRAM



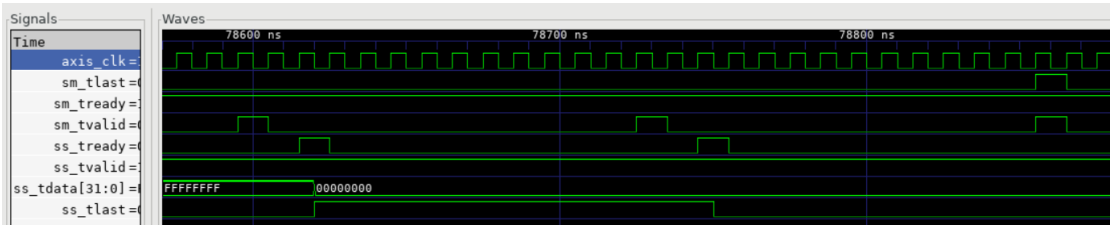
Tap SRAM: 每次運算都固定從位址 0，每個 clock shift 一個 word，用一個位址 register 即可完成。

Data SRAM: 我用兩個位址 register 來完成，一個 address saver 用來儲存起始位址，每做完一次運算之後會位移到下一個位址，並存入新的 Data，並用另外一個位址 pointer 從 saver 位址開始每一個 clock shift 一個 word。

(6) AXI Stream

Stream 的控制都由 Finite state 控制，我都有設計 stream 輸出跟輸入的 state，所以 sm_tvalid 和 ss_tready 就是由是否在那個 state 來控制，而 sm_tlast 則是由我的 pattern count 以及 state 來控制。

最後一筆 stream in 及 stream out 的波形如下:



(7) Test bench Design

```
initial begin
    initial_task;
    for (e = 0; e < RUN_TIME;e=e+1) begin
        wait_idle_task;
        program_param_task;
        program_ap_start_task;
        fork
            transmit_xn_task;
            receive_yn_task;
            polling_ap_done_task;
        join
    end
    repeat(10) @(negedge axis_clk);
    $finish;
end
```

基本上我的 testbench 是完完全全照著 spec 中去設計，先等 design 進入 idle，之後用 AXI lite 去 program tap sram、ap protocol register 及 data length，並透過 AXI read 來確認值有正確 program，接著 program ap_start，接著用 fork join 同時 stream input、stream output 以及不停 polling ap_done，接著重複幾次流程。

3. Resource Usage

Resource	Utilization	Available	Utilization %
LUT	243	53200	0.46
FF	179	106400	0.17
DSP	3	220	1.36
IO	329	125	263.20

4. Timing Report

Clock Period = 6ns

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.752 ns	Worst Hold Slack (WHS):	0.140 ns	Worst Pulse Width Slack (WPWS):	2.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	272	Total Number of Endpoints:	272	Total Number of Endpoints:	178

All user specified timing constraints are met.

Longest Path:

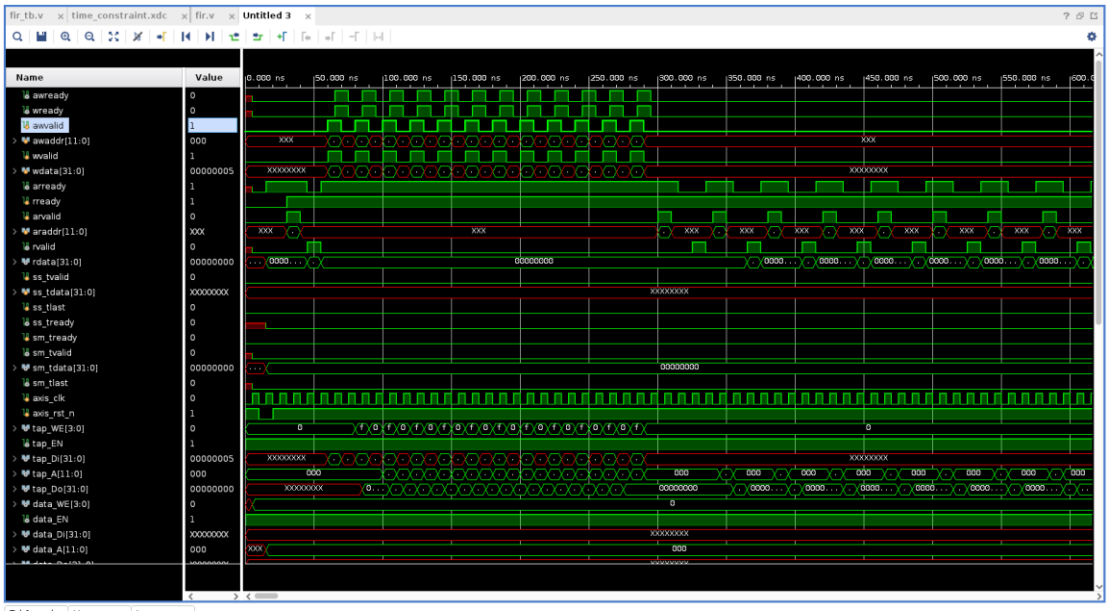
Name	Path 1
Slack	0.752ns
Source	genblk1.length_reg[4]/C (rising edge-triggered cell FDRE clocked by axis_clk {rise@0.000ns fall@
Destination	genblk1.current_state_reg[2]/D (rising edge-triggered cell FDRE clocked by axis_clk {rise@0.000
Path Group	axis_clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	6.000ns (axis_clk rise@6.000ns - axis_clk rise@0.000ns)
Data Path Delay	5.112ns (logic 3.071ns (60.074%) route 2.041ns (39.926%))
Logic Levels	10 (CARRY4=8 LUT6=2)
Clock Path Skew	-0.145ns
Clock Un...rtainty	0.035ns

10 (CARRY4=8 LUT6=2)

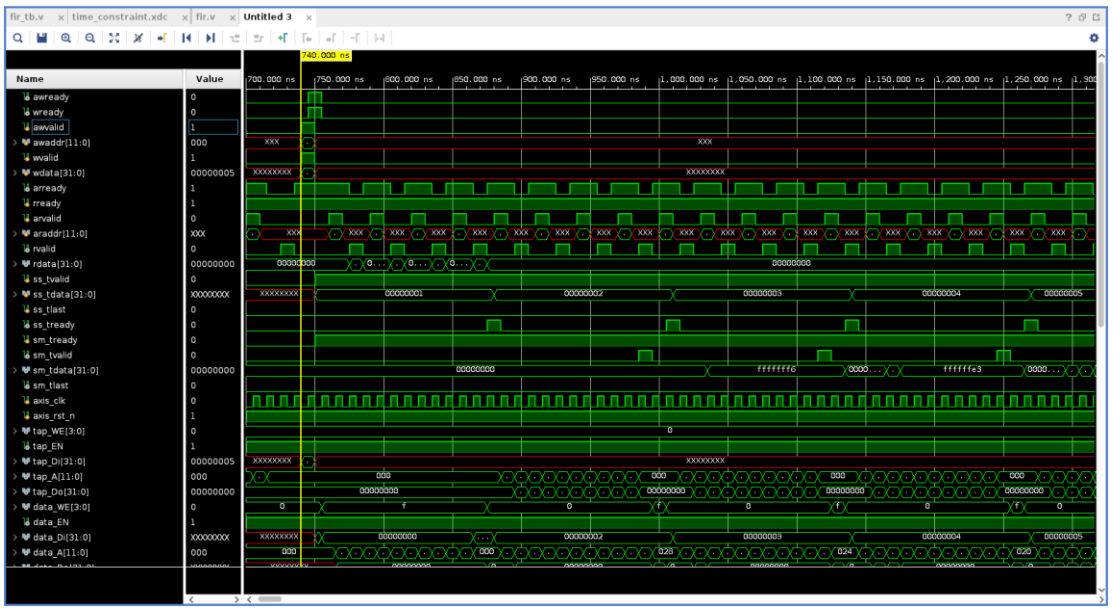
另外我的 design 所花費的 cycle 數為 7815。

5. Simulation Waveform

Coefficient Program and Read back:

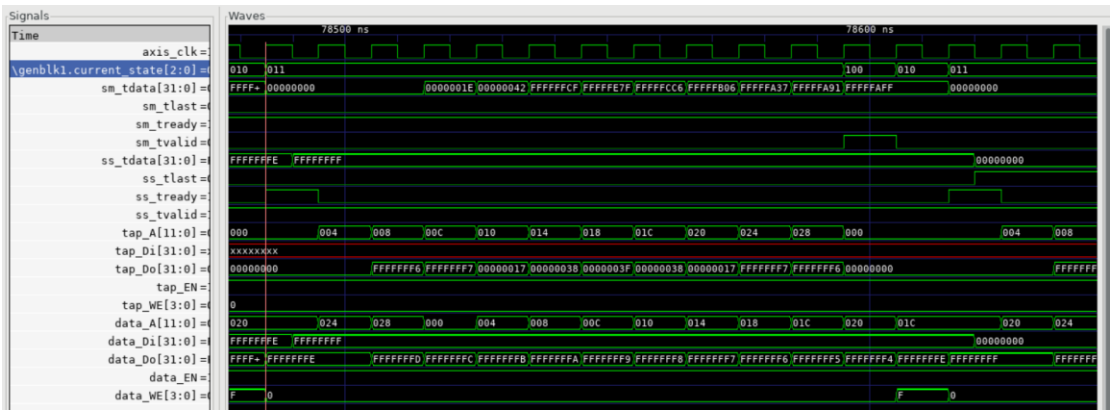


Program ap_start, data stream in and stream out:



RAM access control:

FSM:



更細節的 waveform 在 2. Describe Operation 中有更詳細的說明。