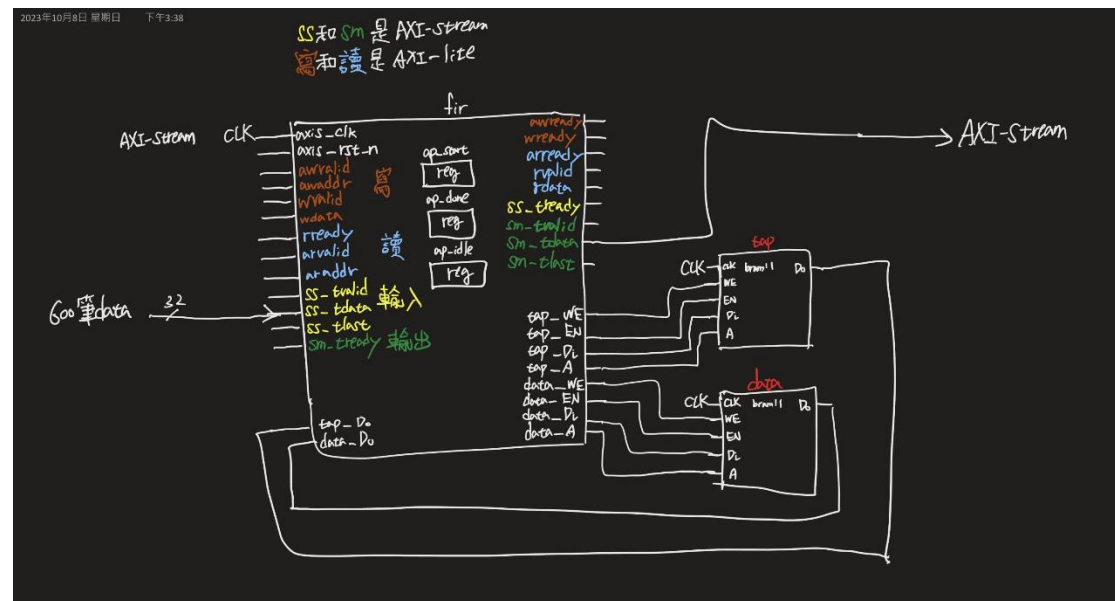


## Block Diagram

- Datapath – dataflow:



(圖一、block diagram)

- Control signals:

FIR 的控制訊號如上圖一，有 `tap_WE`、`tap_EN`、`tap_DI`、`tap_A`、`data_WE`、`data_EN`、`data_DI`、`data_A`，可以控制兩個 BRAM 什麼時候進行讀取還是讀取+寫入、對哪個地址進行、寫入的資料等等。

而 `ss_tready` 則是控制何時可以傳入資料、`sm_tvalid`、`sm_tdata`、`sm_tlast` 則是控制何時輸出什麼資料且何時輸出完最後一筆資料。

`awready` 和 `wready` 則是在寫資料中扮演重要的角色，`awready` 和 `wready` 為 1 時，代表 slave 準備好接收寫入的地址與寫入的資料。

`arready`、`rvalid`、`rdata` 則是再讀資料扮演重要的角色，`arready` 和 `rvalid` 為 1 時，代表 slave 準備好接收地址，master 準備好傳資料並且 `rdata` 是讀取出來的資料。

## Describe operation

- How to receive data-in and tap parameters and place into SRAM:

Tap parameters 是利用 AXI-LITE 寫入，也就是 `fir_tb` 中的 `config_write` 來執行，`awvalid` 和 `wvalid` 先維持 1 且 `awaddr` 和 `wdata` 也準備好，等到 `wready` = 1 時，即可寫入 SRAM。至於 `data_in` 則是接收 AXI-Stream 的 `ss_tdata` 以及 `shift` 的方式來接收資料，在 `fir_tb` 中，`ss` 用來執行傳送 `ss_tdata` 的功能，首先會將 `ss_tvalid` 和 `ss_tdata` 準備好，等待 `ss_tready` = 1 時才傳入。

- How to access shiftram and tapRAM to do computation:

在執行時，我用 33 個 CLK 將資料寫進且 shift data，來將 data\_ram 的資料寫好，並且在隨後的 22 個 CLK 將相對應的 11 個 RAM 的 data\_Do 和 tap\_Do 相乘並存在一個暫存器，把各相乘的值加起來即可得到結果。

- How ap\_done is generated:

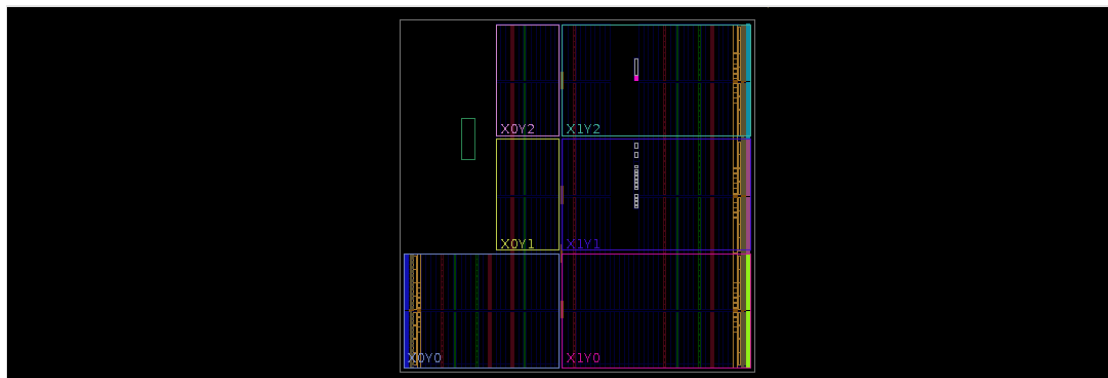
我是利用 length\_counter 來計數現在進行到第幾筆資料運算，當到達了 data\_length 的長度時，即完成運算，使 ap\_done = 1。

- How shiftRAM be shifted:

在此實驗的 bram 中，讀取要兩個 CLK，寫需要一個 CLK，因此如果先寫入的話，讀出來的會是剛剛寫入的資料，也就不能把原本的資料往後傳，所以我採取先讀再寫的方法，一步一步把資料往後傳。過程中會需要用到 read\_temp 和 write\_temp 來完成。在 shift 時，一個 data\_A 需要維持三個 CLK，第一個 CLK 開始讀，第二個 CLK 將讀出來的 data\_Do 傳到 read\_temp，第三個 CLK 將 write\_temp(前一個位置所存的資料)傳到 data\_Di 來寫入資料。

## Resource usage

下圖為合成後的電路示意圖



(圖二、合成後電路圖)

**BRAMs:** 280 (col length: RAMB18 60 RAMB36 30)

### Report Cell Usage:

	Cell	Count
1	BUFG	7
2	CARRY4	20
3	DSP48E1	3
4	LUT1	2
5	LUT2	90
6	LUT3	9
7	LUT4	55

8	LUT5		19
9	LUT6		95
10	FDRE		49
11	FDSE		1
12	LD		1
13	LDC		205
14	LDCP		3
15	IBUF		159
16	OBUF		169
+-----+-----+-----+			

## Timing Report

- Try to synthesize the design with maximum frequency:

在尋找最大操作頻率時，我們必須先找到最長路徑，在觀看合成報告時，可以明顯看到 FIR 的 32bit 乘加運算會造成很大的 data path delay，比對後發現此路徑就是最長的路徑，因此我們必須把 clock cycle time 提高，在這裡我的 data path delay 是 15.285ns，再考慮一些 tolerance，我的最高操作頻率大概落在 16ns/次，也就是 62.5MHz。

在觀看 timing report 時，我發現如果 destination 如果是 latch，那這個 timing report 將不會考慮資料到 latch 的時間，因此不太精準，因此必須將 latch 改掉，如此才能知道更精確的 delay 時間。得知正確的 delay time 後，也必須對 clock cycle time 進行修正來維持功能的正常。

- Report timing on longest path, slack:

可以看到 data path delay = 15.285ns, input delay = 2ns, output delay = 1ns, clock uncertainty = 0.035ns。我給定 clock cycle = 20ns，扣掉(input delay + data path delay + output delay + clock uncertainty)後，也就是  $20 - 2 - 15.285 - 1 - 0.035 = 1.679$  是我最整個電路最長路徑的 slack。

(clock axis\_clk rise edge)

		20.000
20.000		
	clock pessimism	0.000
20.000		
	clock uncertainty	-0.035
19.965		
	output delay	-1.000
18.965		

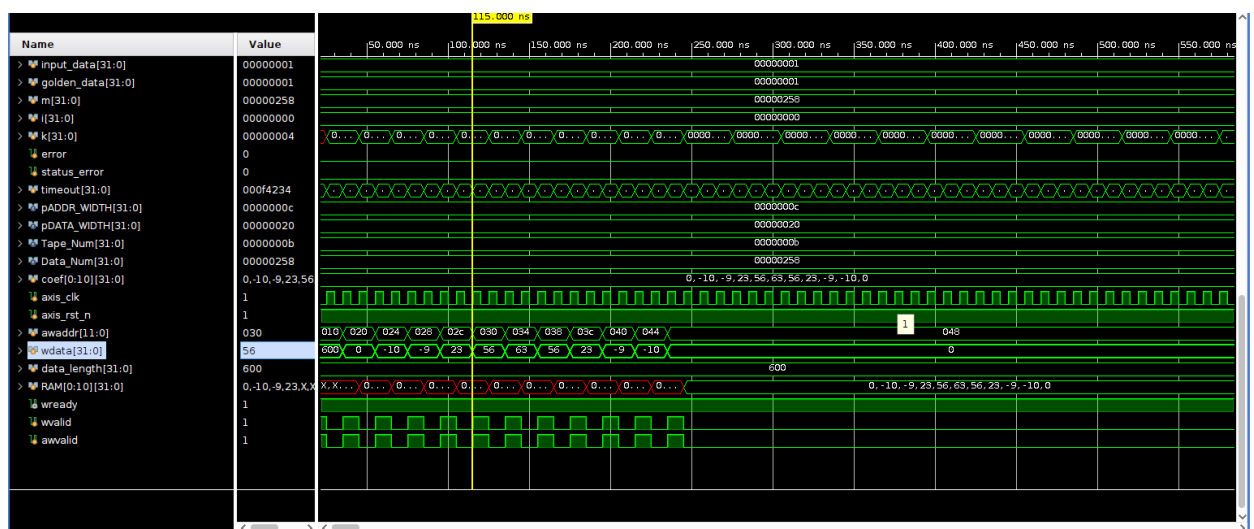
-----

	required time	
18.965		
	arrival time	-
17.285		
<hr/>		
	slack	1.679

## Simulation Waveform

### • Coefficient program, and read back:

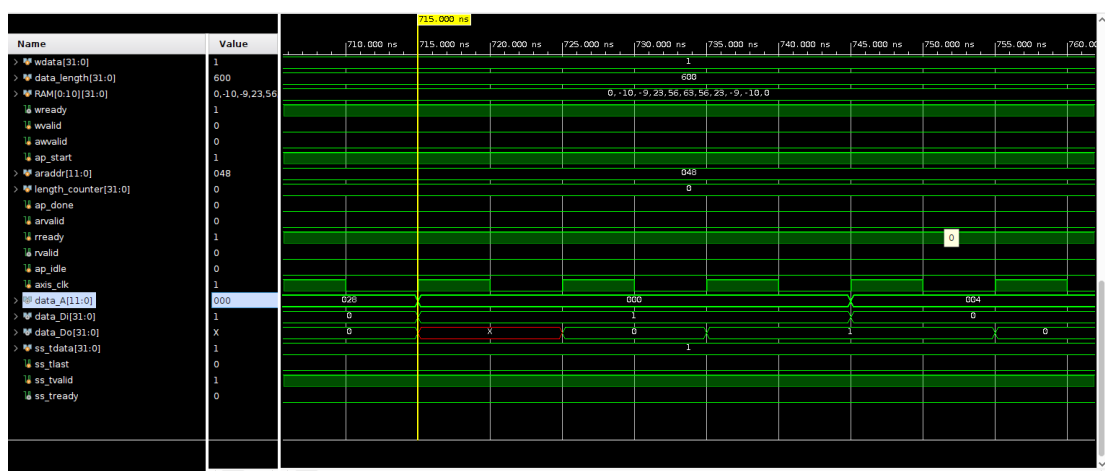
當 awaddr 是寫入的地址，wdata 是寫入的資料，可以看到 tap\_RAM 內部的資料依序寫入，到 0x48 時，已經擁有完整的 11 個 coefficient 在裡面。



(圖三、Coefficient program, and read back)

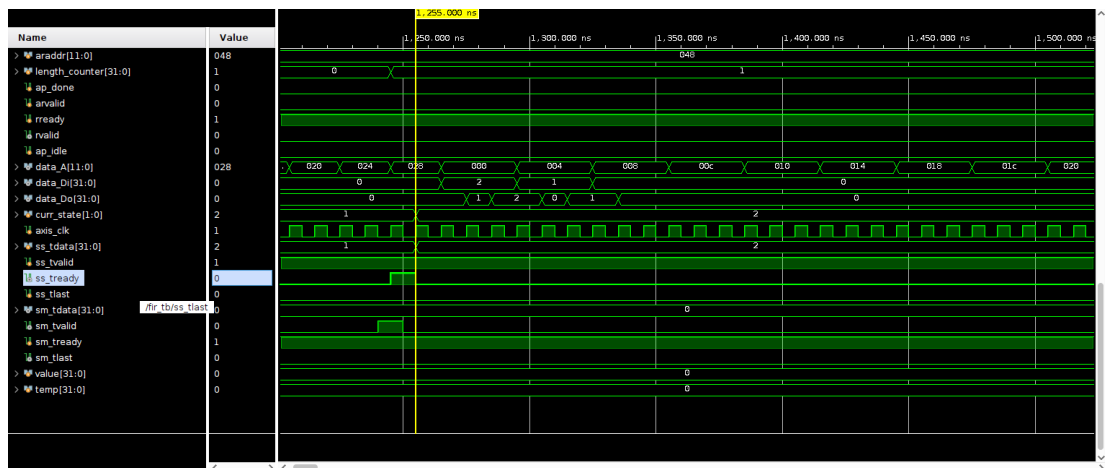
### • Data-in stream-in:

可以到相對應的 data\_A，有相對應的 data\_Di，這裡的資料是第一筆 ss\_tdata 進入。



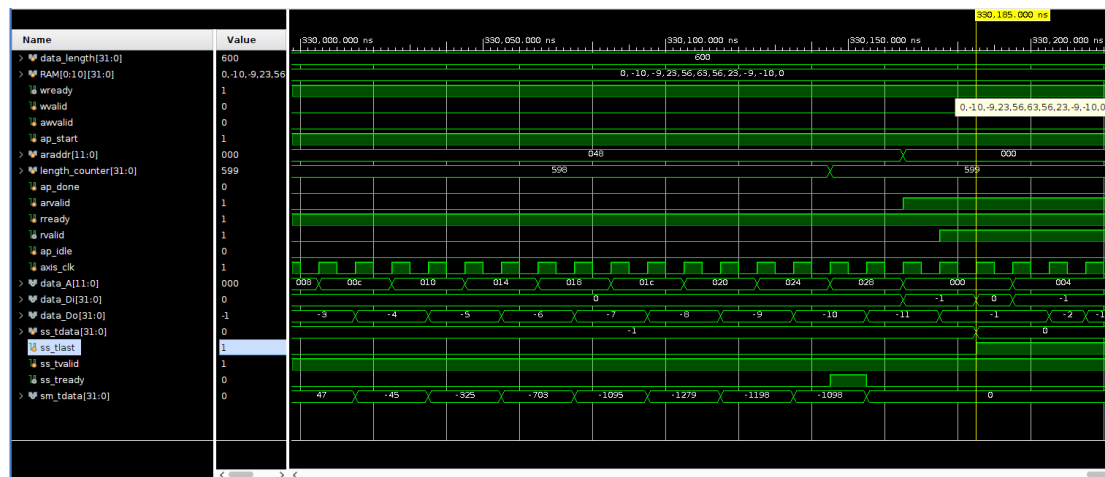
(圖四、當 ss\_tdata = 1 時，data\_ram 寫入情況)

當 ss\_tready=1 時，下一個 CLK 就讓 ss\_tdata 進入下一筆資料。



(圖五、當 ss\_tready=1 時，ss\_tdata 傳送下一筆資料)

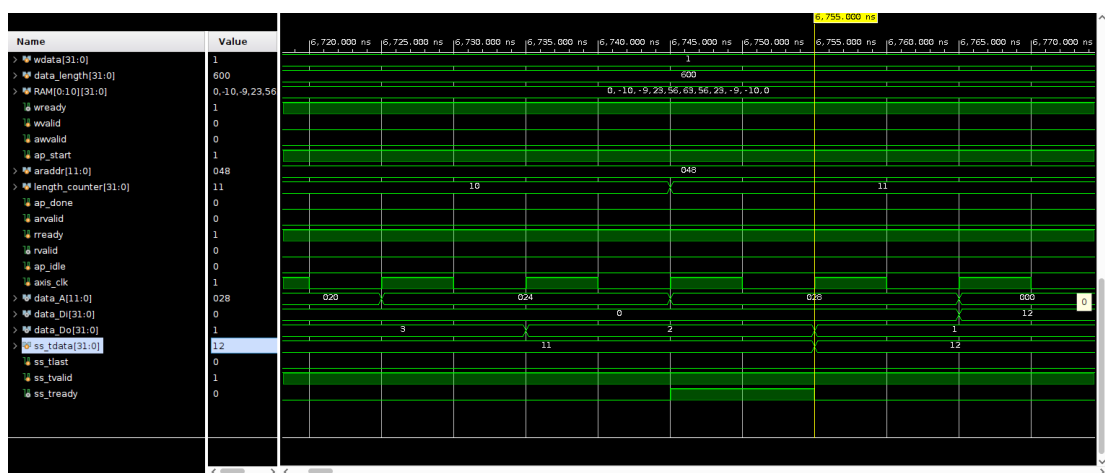
當 length\_counter=599，代表輸入最後一筆資料，此時將 ss\_tlast 升起



(圖六、當 ss\_tdata 傳送完最後一筆資料，ss\_tlast 升起)

### • Data-out stream-out:

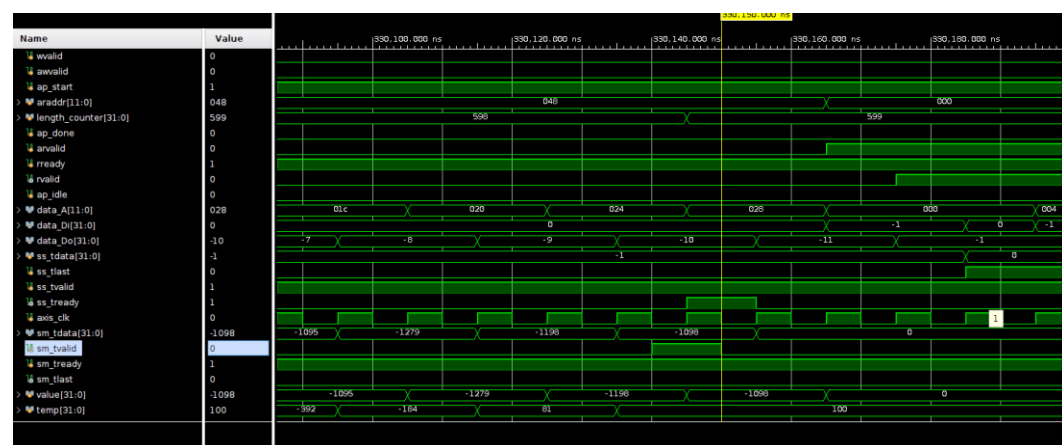
這裡是正在讀取的階段，可以看到相對應的位置讀取出相對應的 data\_Do。



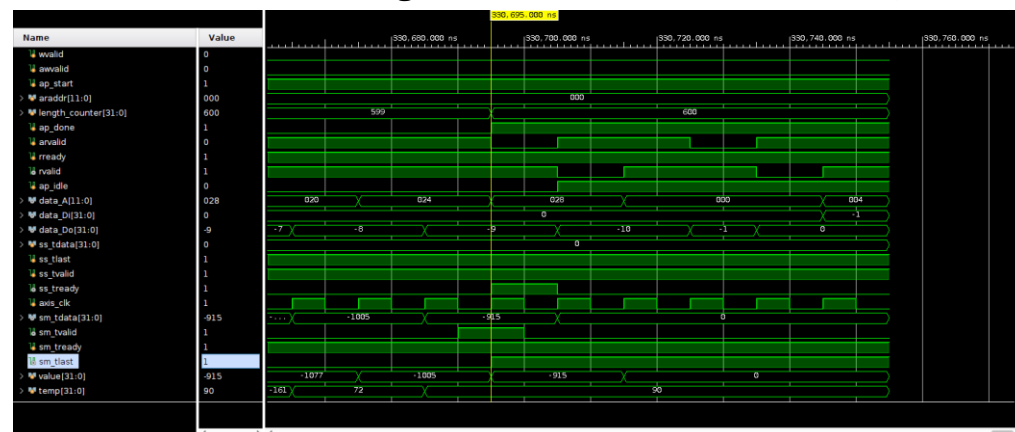
(圖七、由 data\_Do 可看出 data\_ram 的讀取狀態)

Name	Value	
▶ wdata[31:0]	1	
▶ data_length[31:0]	600	
▶ RAM[0:10][31:0]	0..10..9..23..56	
wready	1	
wvalid	0	
awvalid	0	
ap_start	1	
wready[1:0]	048	
▶ length_counter[31:0]	14	
ap_done	0	
anvalid	0	
rready	1	
rvalid	0	
ap_idle	0	
axis_clk	1	
▶ data_A[11:0]	000	
▶ data_D[31:0]		
data_Do[31:0]	15	
ss_data[31:0]	15	
ss_tlast	0	
ss_tvalid	1	
ss_tready	0	

在讀取的階段，會把每個相對應的 data\_Do 和 tap\_Do 讀取出來相乘相加，相乘的結果存在 temp，而把各個 temp 加起來後存到 value 中，且在最後的結果出來時，讓 sm\_tvalid 升起來把 value 送出去給 sm\_tdata



sm\_tlast 當上方的波型 length\_counter=600 時，sm\_tlast 上升到 1

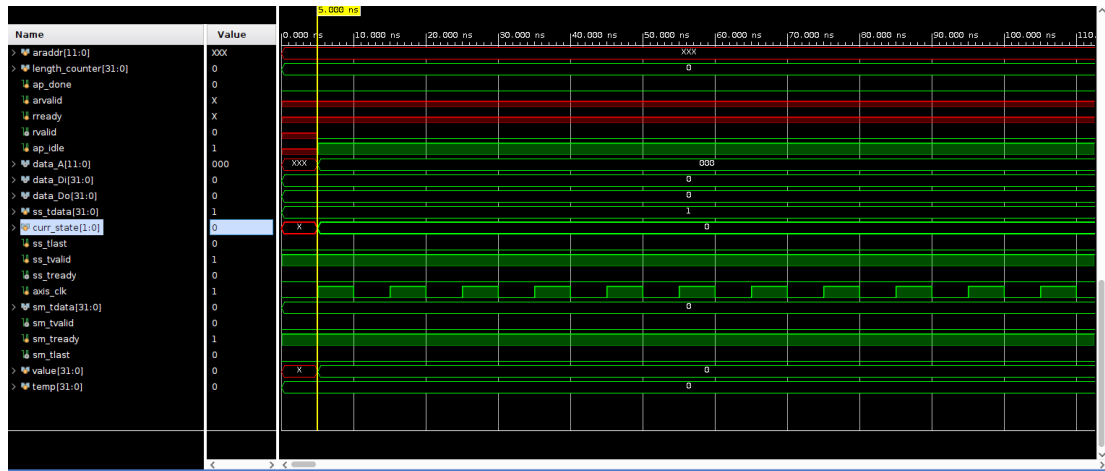


(圖十、送出最後一筆資料十，sm\_tlast 升起)

## • FSM:

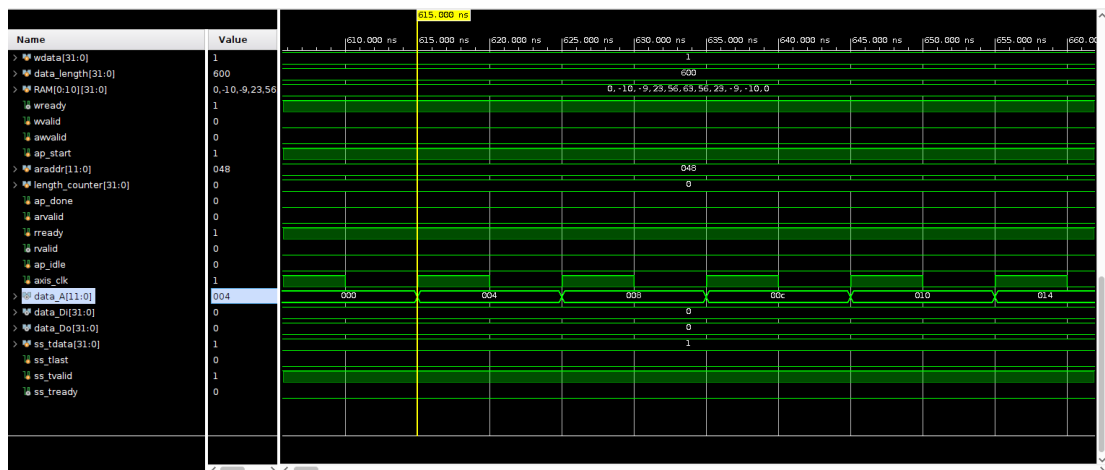
FSM 部份我分成三種狀態

第一種狀態  $S_0=2'b00$ ，在  $S_0$  狀態中會先將 data\_ram 寫入 11 個 0。



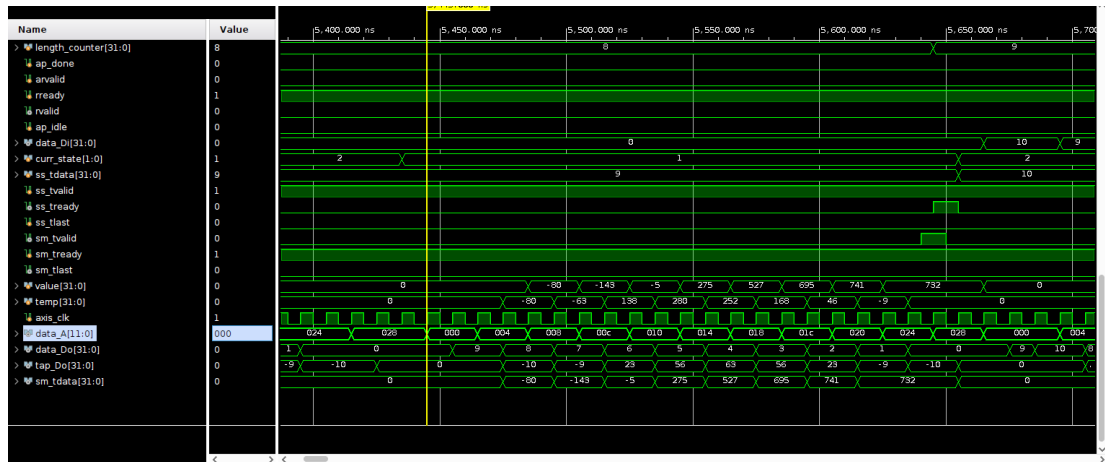
(圖十一、第一種狀態  $S_0$ ，data\_Di 皆寫入 0 來清零 data\_ram)

補充第一種狀態  $S_0$  的波形圖，此時會讓每個 data\_A 輸入 data\_Di = 0



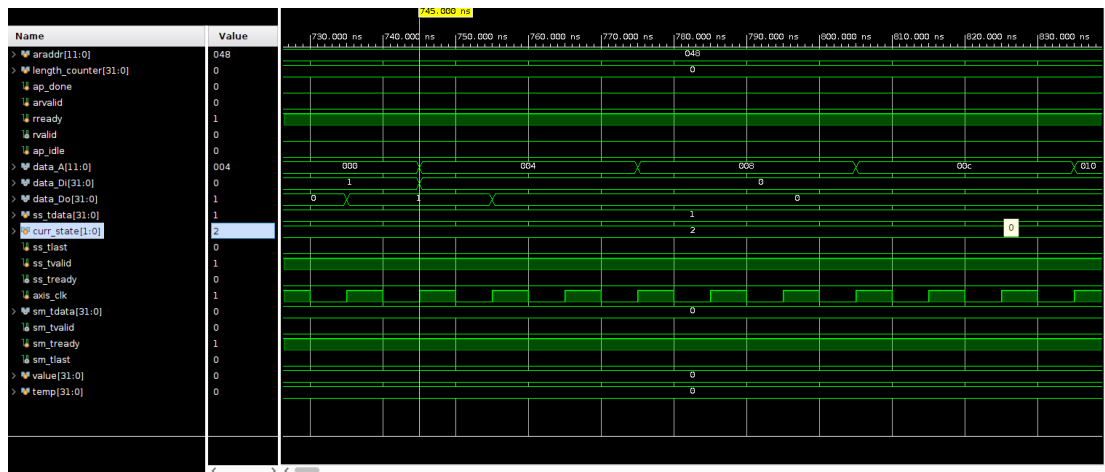
(圖十二、第一種狀態  $S_0$ ，相對應的 data\_A 的 data\_Di 皆為 0)

第二種狀態 S1=2' b01，在 S1 狀態中狀態中會將擺入正確資料的 tap\_ram 以及 tap\_data 的資料取出來相乘相加。



(圖十三、狀態 S1，將地址相對應的 data\_Do 和 tap\_Do 讀取出來相乘相加)

第三種狀態 S2=2' b10，在 S2 狀態中會將 ss\_tdata 的資料進來一筆並且將 ram 內部的資料 shift。

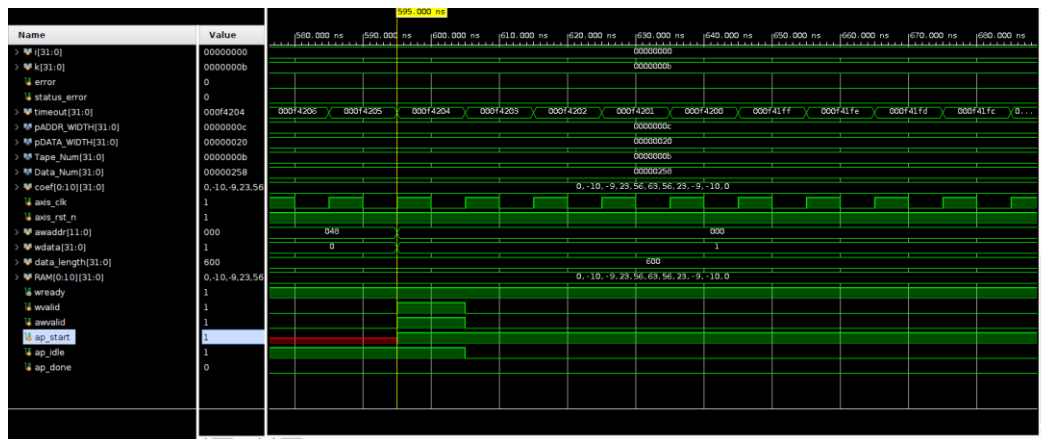


(圖十四、狀態 S2，進入一筆 ss\_tdata 並且 shift)



- ap\_start:

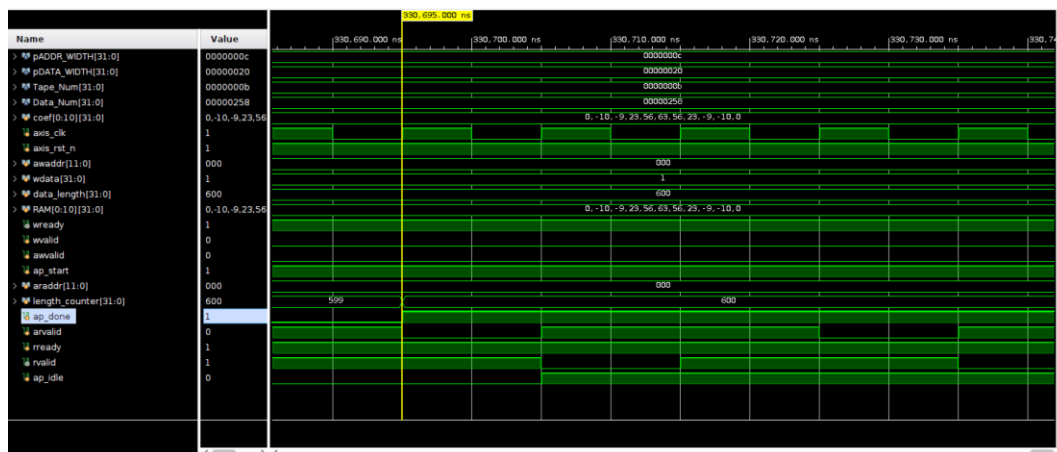
當 tap\_ram 寫入完成後，把 ap\_start 升起。



(圖十五、ap\_start 升起)

- ap\_done:

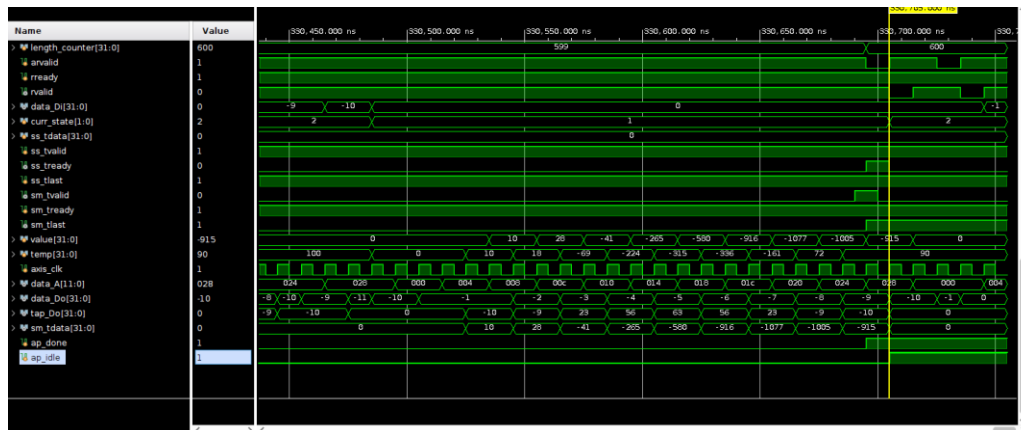
當輸入第 600 筆資料時，ap\_done = 1



(圖十六、ap\_done 升起)

- ap\_idle:

當 ap\_done=1 隨後把 ap\_idle 升起，以表示閒置



(圖十七、ap\_idle 升起)