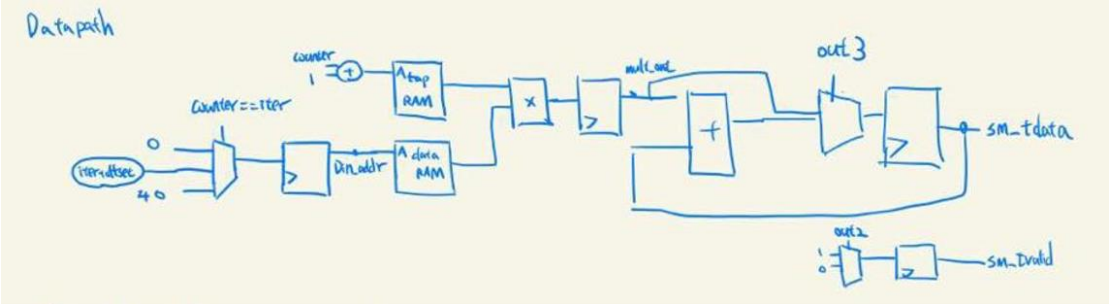
Report

312510145 劉峻瑋

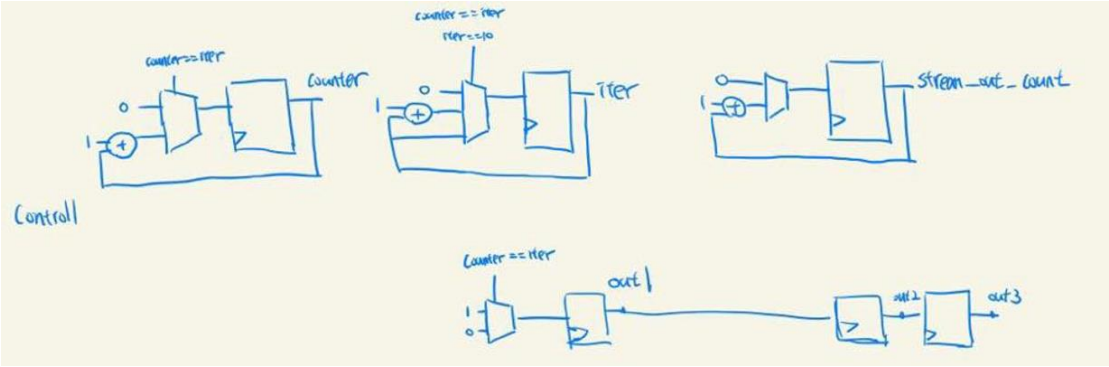
Block Diagram

• Datapath – dataflow



我的 fir 會分為三級的 pipeline，第一級是算出 data RAM 和 tap RAM 的 address，第二級會對 tap RAM 和 data RAM 的輸出做乘法，第三級則是加法器對乘法的輸出做累加，若有新的 Yn 被算出來，sm_tvalid 會被設成 1。

• Control signals

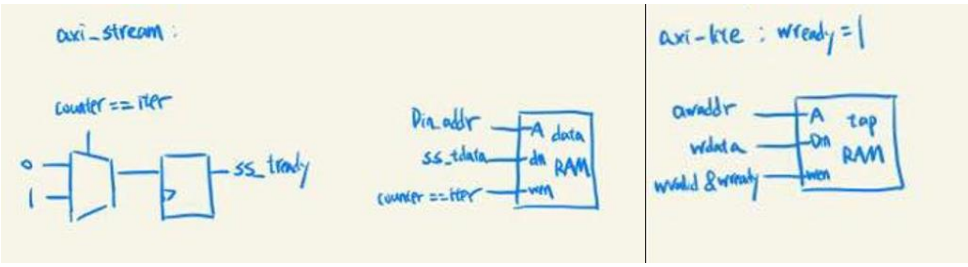


控制訊號用來紀錄時序的過程來控制 dataflow pipeline 的行為，iter 用來表示這次的 linear conv 需要計算幾次，FIR start 的時候會被 reset 成 1，表示第一個輸出只需要乘加一次，最高會到 10，表示後面的輸出需要被乘加 10 次；counter 用來表示這次的 linear conv 做到第幾次，若 counter==iter，表示在算完這組數字後會有新的 Yn 被算出，因為再 2 個 cycle 值才會被算出來、再 3 個 cycle 才需要將 adder 的 output 歸零，因此我會用 out1，out2，out3 這三個 shift register 來記錄並控制相關的行為。

最後是 stream_out_count 來記錄輸出了幾個 output，若 stream_out_count==length，FIR 會被停止並進入下一個 state。

Describe operation

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



Data in:

Data in 是經由 axi-stream 來接收，當我的控制訊號 counter==iter 時，會把 ss_tready 設為 1，來告訴 host 這筆資料已經接收完成，準備接收下一筆資料。我使用 bram11 作為 Data RAM，D_address 會接到我算好的 address，原則上第一筆為 0，第二筆為 4…第 11 筆為 40，然後第 12 筆再回到 0…，Din 接到 wdata，wen 則是當 counter==iter 時設成 1。

Tap in:

Tap in 經由 axi-lite 來接收，我會將 wready 一直設為 1，因為再接收 tap parameter 時不需要延遲，告訴 host 我下一個 cycle 就可以接收下一筆 tap parameter，T_addr 會接到 awaddr，tap_in 接到 wdata，wen 則是當 wvalid 和 wready 為 1 時設成 1。

• How to access shiftram and tapRAM to do computation

利用 counter 和 iter 這兩個控制訊號來決定 ram address 來直接使用 ram out 來做計算，shiftram address=iter-offset，tapram address=counter+1。因此不需要額外暫存器。

• How ap_done is generated.

當 stream_out_count==600，fir 會停止，進入到下一個 state，這個 state 會將 ap_done 設成 1。另外當 fir 收到 ap_start=1 時，也會進入到下一個 state，將 ap_start 設為 1，ap_done 設為 0。

• Resource usage: including FF, LUT, BRAM

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	214	0	0	53200	0.40
LUT as Logic	214	0	0	53200	0.40
LUT as Memory	0	0	0	17400	0.00
Slice Registers	92	0	0	106400	0.09
Register as Flip Flop	92	0	0	106400	0.09
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

Flip-flop 用在控制訊號和 2 個加法器乘法器輸出的 32bit 暫存器。

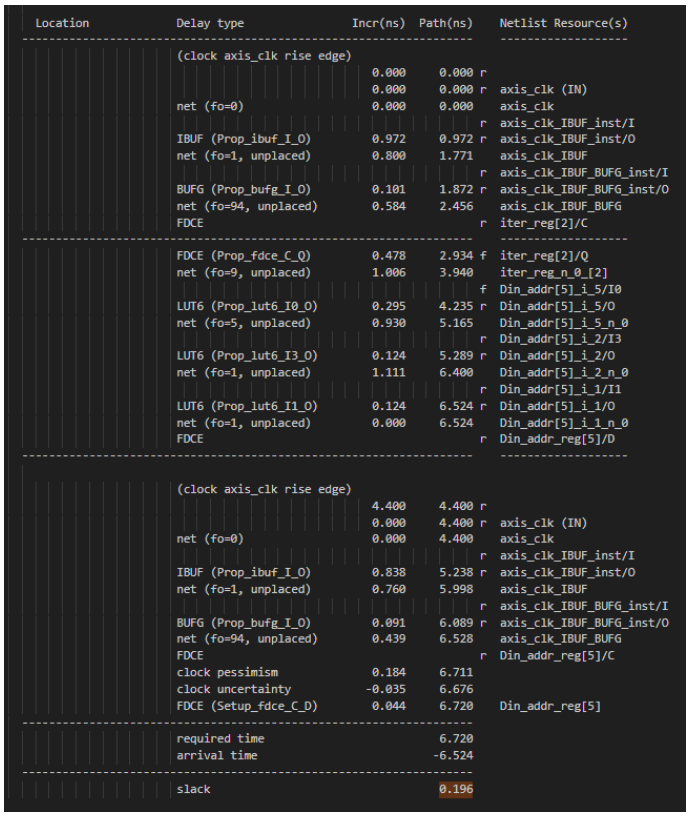
BRAM 使用一個 bram11 一個 bram12

• Timing Report

• Try to synthesize the design with maximum frequency

My minimum cycle time is 4.4ns.

• Report timing on longest path, slack

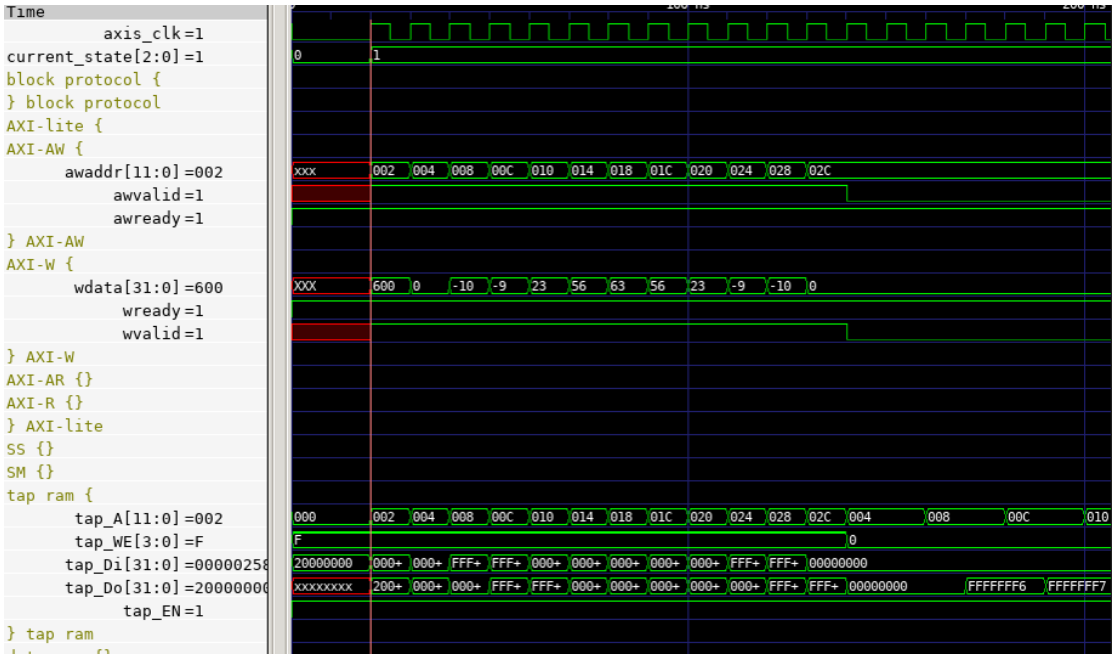


Location	Delay type	Incr(ns)	Path(ns)	Netlist Resource(s)
	(clock axis_clk rise edge)			
		0.000	0.000 r	
		0.000	0.000 r	axis_clk (IN)
	net (fo=0)	0.000	0.000	axis_clk
			r	axis_clk_IBUF_inst/I
	IBUF (Prop_ibuf_I_O)	0.972	0.972 r	axis_clk_IBUF_inst/O
	net (fo=1, unplaced)	0.800	1.771	axis_clk_IBUF
			r	axis_clk_IBUF_BUFG_inst/I
	BUFG (Prop_bufg_I_O)	0.101	1.872 r	axis_clk_IBUF_BUFG_inst/O
	net (fo=94, unplaced)	0.584	2.456	axis_clk_IBUF_BUFG
	FDCE		r	iter_reg[2]/C
	FDCE (Prop_fdce_C_Q)	0.478	2.934 f	iter_reg[2]/Q
	net (fo=9, unplaced)	1.006	3.940	iter_reg_n_0_[2]
			f	Din_addr[5]_i_5/I0
	LUT6 (Prop_lut6_I0_O)	0.295	4.235 r	Din_addr[5]_i_5/O
	net (fo=5, unplaced)	0.930	5.165	Din_addr[5]_i_5_n_0
			r	Din_addr[5]_i_2/I3
	LUT6 (Prop_lut6_I3_O)	0.124	5.289 r	Din_addr[5]_i_2/O
	net (fo=1, unplaced)	1.111	6.400	Din_addr[5]_i_2_n_0
			r	Din_addr[5]_i_1/I1
	LUT6 (Prop_lut6_I1_O)	0.124	6.524 r	Din_addr[5]_i_1/O
	net (fo=1, unplaced)	0.000	6.524	Din_addr[5]_i_1_n_0
	FDCE		r	Din_addr_reg[5]/D
	(clock axis_clk rise edge)			
		4.400	4.400 r	
		0.000	4.400 r	axis_clk (IN)
	net (fo=0)	0.000	4.400	axis_clk
			r	axis_clk_IBUF_inst/I
	IBUF (Prop_ibuf_I_O)	0.838	5.238 r	axis_clk_IBUF_inst/O
	net (fo=1, unplaced)	0.760	5.998	axis_clk_IBUF
			r	axis_clk_IBUF_BUFG_inst/I
	BUFG (Prop_bufg_I_O)	0.091	6.089 r	axis_clk_IBUF_BUFG_inst/O
	net (fo=94, unplaced)	0.439	6.528	axis_clk_IBUF_BUFG
	FDCE		r	Din_addr_reg[5]/C
	clock pessimism	0.184	6.711	
	clock uncertainty	-0.035	6.676	
	FDCE (Setup_fdce_C_D)	0.044	6.720	Din_addr_reg[5]
	required time		6.720	
	arrival time		-6.524	
	slack		0.196	

令我意外的是我最長的 delay path 並非 32bits 的乘法，而是在計算當前的 data ram address，所需要的計算是((counter+offset)%11)<<2，需要在做加法後 round down 再前移 2 位。

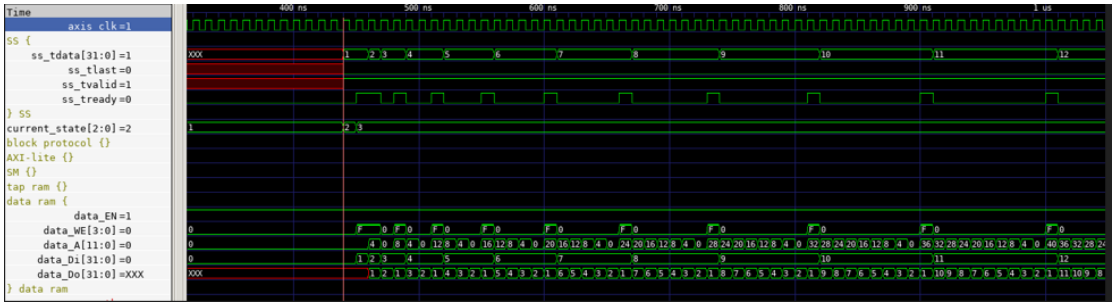
• Simulation Waveform, show

• Coefficient program, and read back



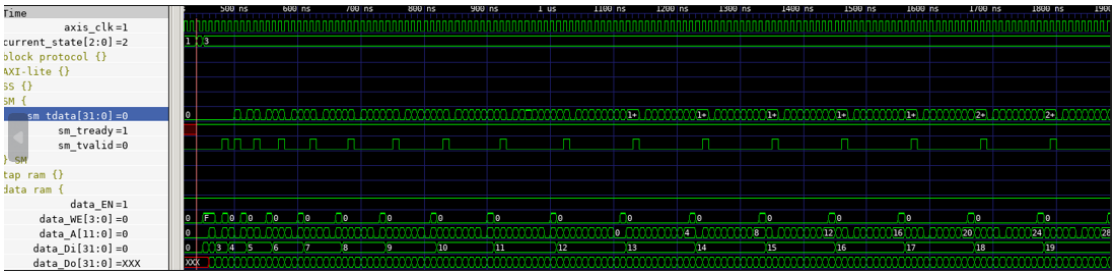
因為我的 tap_ram 可以連續的寫入，我把 testbench 的 axi-lite 改成當 awvalid 和 awready 都為 1 時下個 cycle 就可以輸出下一筆資料

• Data-in stream-in



因為 host 端隨時準備好輸出下一筆資料，所以 ss_tvalid 一直設為 1，因為題目規定不能拿額外的暫存器存 data 且每個 cycle 只能做一次乘加，所以我的 fir 會算完這筆 data 才準備好接收下一筆 data，ss_tready 為 1 時為準備好接收下一筆 data。

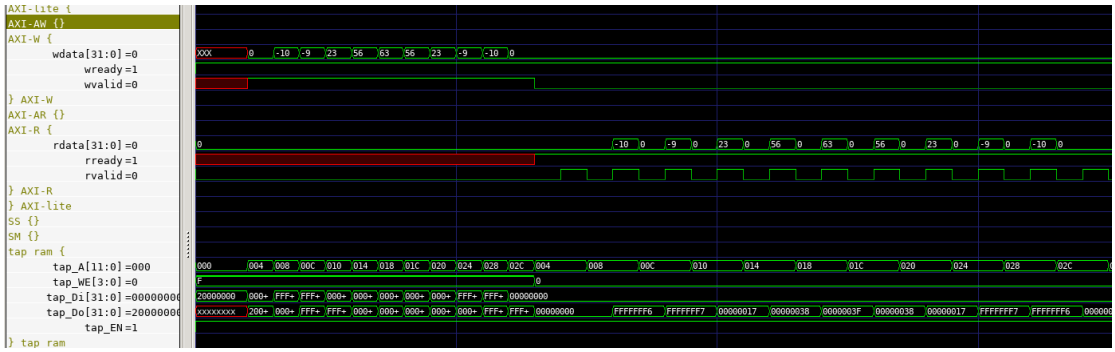
• Data-out stream-out



當一筆資料經過 10 個 cycle 後，這筆資料就準備好送回 host 端檢查，會將 sm_valid 設為 1。而 host 端隨時準備好接收資料，因此 sm_tready 一直設為 1。

• RAM access control

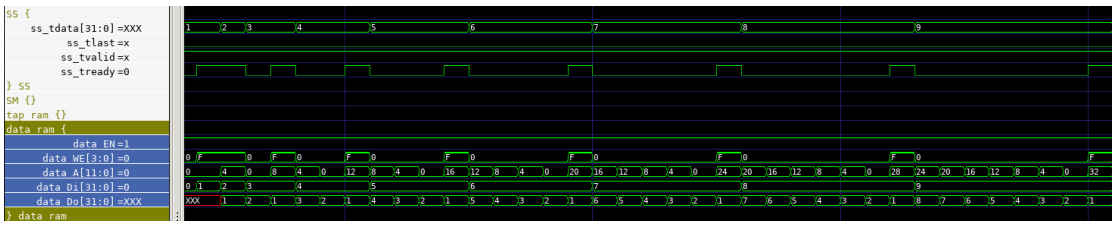
Tap_parameter:



Axi_lite write 時，tap_wen 設為 1，tap_A 設為 awaddr，tap_Di 設為 wdata。

Axi_lite read 時，tap_wen 設為 0，tap_A 設為 araddr，rdata 連接到 tap_Do。

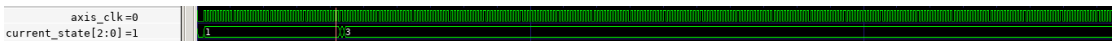
Data_in:



Data_in 時，data_we 設為 1，address 由 control 訊號做計算，data_Di 接到 ss_tdata。

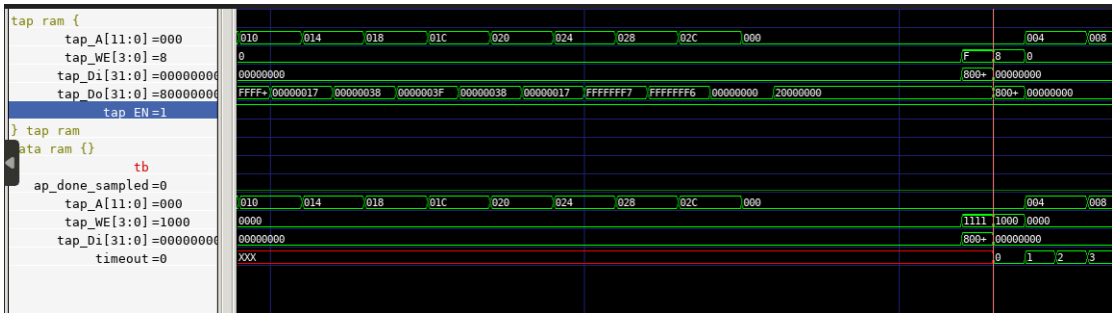
在做 fir 計算時，乘法器的 input 分別設為 tap_ram 和 data_ram 的 output。

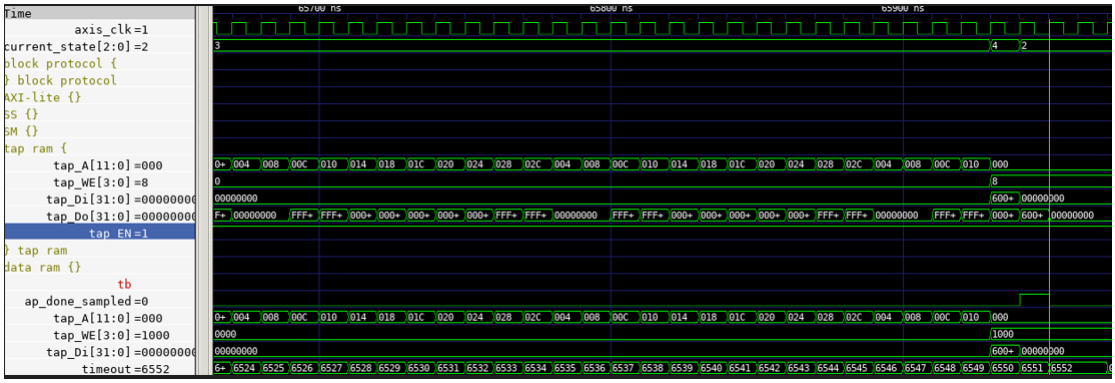
• FSM



我的 FSM 簡單分為 4 級，分別是一開始的 parameter setup、ap_start_setting、fir_executing 和 ap_done_setting。可以從上圖發現大部分的時間都在 fir_executing

• AP_start to AP_done





FIR cycle: 6552

Timeout 是在 testbench 用來記錄運行時間的參數，可以看見 ap_start pull up 的時候 reset 成 0，ap_done pull up 時停止紀錄並輸出紀錄結果。