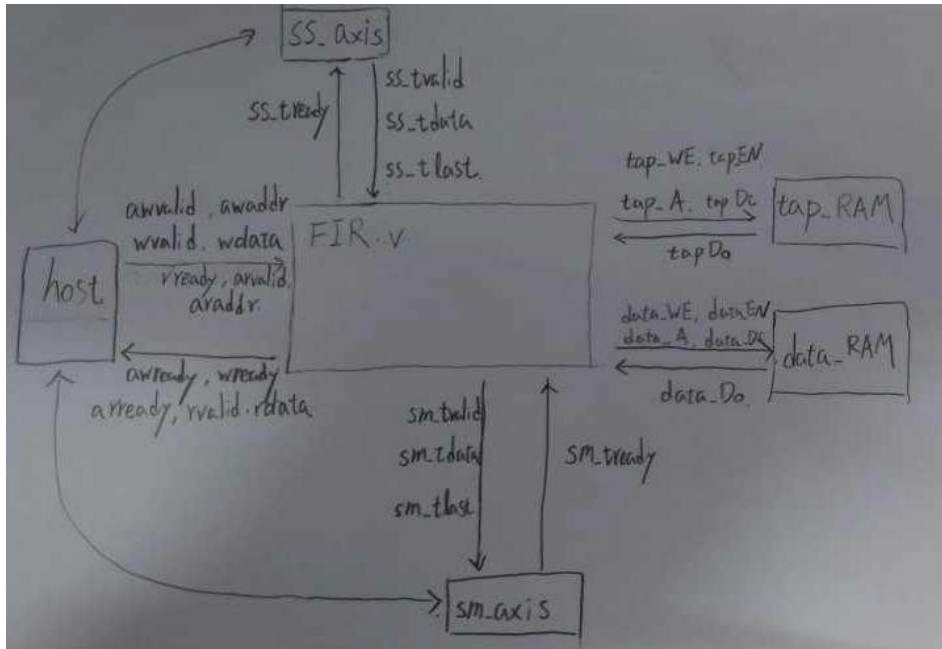


SOC Design Lab3

112061559 梁凱傑

- Block Diagram



我將 FIR 與外部溝通和計算功能全部都寫在 fir module 中。
以下是在 FIR 內的控制訊號：

reg [31:0] status, // 內部存著 FIR 的運作狀態，status[0] 為 ap_start (r/w)，status[1] 為 ap_done (ro)，status[2] 為 ap_idle (ro)，功能為workbook 中所敘述。Status中的其他 bits初始化為0，在此 FIR 設計中並無特定功用。

reg init, // reset後、ap_start被programmed成1後，init會設成1，此訊號是為了初始化 data_RAM內的資料為零，並在完成後將 init 設成0。(初始化為 1)

reg writing, // 從 host 收到 wdata後設為1，在完成 write 後設為0。為了避免 wdata 被後續輸入覆蓋，導致 write 失敗。(初始化為 0)

reg awriting, // 從 host 收到 awaddr後設為1，在完成 write 後前設為0。為了避免 awaddr 被後續輸入覆蓋，導致 write 失敗。(初始化為 0)

reg rr, // 從host 收到 araddr 後設為 1，在完成 read 後設為 0。為了避免 araddr 被後續輸入覆蓋，導致 read 失敗。(初始化為 0)

reg [5:0] readptr, // 為存取 tap_RAM 用的 address，用來 assign 給 tap_A。(初始化為 0)

reg [5:0] dreadptr, // 為存取 data_RAM 用的 address，用來 assign 給 data_A。(初始化為 Tap_Num - 1)

reg sswait, // 在ss_axis 輸入 data後設成1，在儲存完此data並計算完 FIR 後設成0，避免在計算 FIR 時有新的 input data進來。(初始化為 0)

reg smset, // 在完成一次 FIR 的計算結果後設成1，在將此結果輸出後設成0。(初始化為 0)

reg last, // 在收到ss的最後一份 data 後設成 1，在重新 program ap_start後設成0。此訊號用來協助控制 ap_done, ap_idle, sm_tlast。(初始化為 0)

reg WaitRD, // 在收到 FIR 的 data 後，因需要等一個 clock 的 BRAM 讀寫，故用此訊號控制。(初始化為 0)
reg Done, // 因計算過程須等 BRAM 和 乘法器、加法器的延遲，故用此訊號來提示即將完成 FIR 的計算。(初始化為 0)
reg backp, // 計算完成後的 output data 若還沒收走，新進來的 data 可能會蓋掉暫存的計算值，故用此訊號來阻擋 input data，值為 1 時，ss_tready 不會拉高。(初始化為 0)

除了前述訊號外，內部還加了4組 registers：

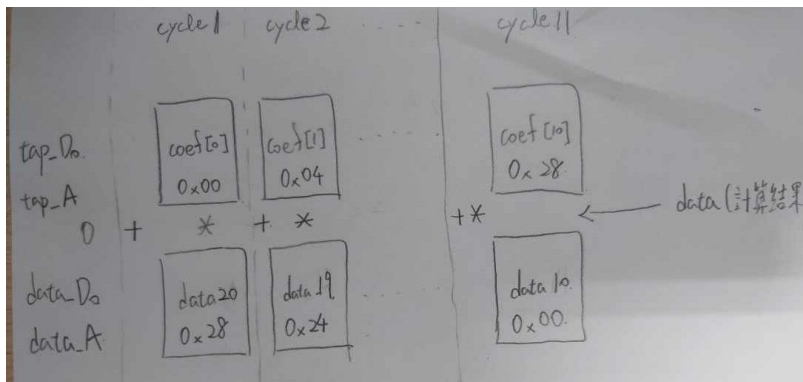
reg [31:0] datalength, // 存入 host 提供的 data 總數量。
reg [31:0] cnt, // 每從ss收一份 data就會 +1，直到數字與 datalength 相同，就會將 last拉起。在 ap_start 被 programmed 時設成 0。
reg [31:0] temp, // 存入 data 經過一次乘法和加法的計算值。
reg [31:0] result // 存入最終計算完的 output data。

• Operation

1. 在計算開始前，會收到由 host 提供的 data_length 和 tap coef 並伴隨著它們的地址。awriting 在 0 時，awready 為 1，並在 awvalid 為 1 的同時收入 awaddr，收入後將 awriting 設為 1，awready 則變為 0，等待寫入完成後再將 awriting 設回 0。writing 在 0 時，wready 為 1，並在 wvalid 為 1 的同時收入 wdata，收入後將 writing 設為 1，wready 則變為 0，等待寫入完成後再將 writing 設回 0。在 writing && awriting 時，則開始進行寫入，此時 tap_Di = wdata, tap_EN = 1, tap_WE = 4'b1111, tap_A = awaddr - 12'h20 (這是因為 tap coef 的 awaddr 從 0x20 開始，所以直接用一個 offset 來將 awaddr 接到 tap_A)，data_length (0x10)則存在 datalength 的 registers。
2. 然後是 host 將存入的 tap coef 的 read back 處理。在 rr 為 0 時，arready = 1, rvalid=0，並在 arvalid 為 1 的同時收入 araddr。這個 araddr 會直接連到 tap_A (一樣有 offset = -0x20)，此時 tap_EN = 1, tap_WE = 4'b0000，這樣 tap_RAM 就可以先準備。下一個 clock 會將 rr 設為 1，此時 arready = 0, rvalid = 1，剛才的 araddr 對應到此時的 tap_Do，並會直接連到 rdata。rr 會在 rvalid && rready 後設 0，此時 arready = 1, rvalid = 0，準備收入下一個 araddr。
3. 再來是 program ap_start。ap_start 設成 1 之後的下個 clock，會開始初始化 (init = 1)、ap_idle 設為 0、ap_start 設回 0。初始化會將 data_RAM 內的所有資料清零 (data_A = from 0x00 to 0x28, data_Di = 32'b0, data_EN = 1, data_WE = 4'b1111)。初始化結束後 init 設成 0，並拉高 ss_tready 開始收入資料進行計算。

```
assign ss_tready = (!status[2] && !sswait && !init && !last && !backp) ? 1 : 0;
```

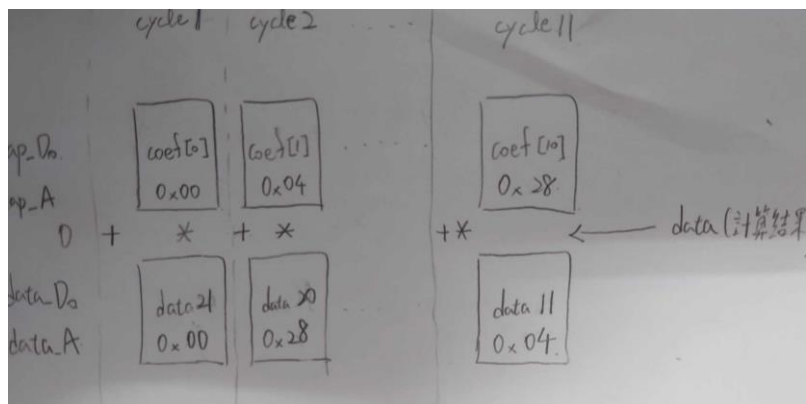
4. s 從 ss_tready 拉高的同時，會將收到的 ss_tdata 存入 $data_RAM$ 裡面、將計算結果 ($temp$) 清零， $cnt \leq cnt + 1$ 並在下一個 $cycle$ 開始計算過程， $sswait$ 此時也會設成 1。計算的方式如下 (假設本次 ss_tdata 收到的為 $data20$ 並存在 $data_RAM$ 的 $0x28$ 裡)：



tap_A 從 $0x00$ 累加到 $0x28$ ， $data_A$ 則是從 $0x28$ 累減到 $0x00$ 。每一次 $cycle$ 計算 $temp \leq tap_Do * data_Do + temp$ 。在 tap_A 數到 $0x28$ 後就會暫停計算，並拉高 $smset$ ， $smset$ 為 1 時， $sm_tvalid = 1$ ，而算完的 $temp$ 則會有兩種情況：

1. $result$ (output buffer) 為空或被讀取過：將此計算完的結果存入 $result$ ，並準備開始下一次運算。
2. $result$ 還存有上一次的計算結果，而且還沒被 sm 讀取過：將此次計算結果暫存在 $temp$ ，並拉高 $backp$ ，暫停 ss 的 $input$ 。等到讀取後再將 $temp$ 內的結果存入 $result$ 裡。

5. ss_tready 再次拉高， tap_A 會回到 $0x00$ ，而 $data_A$ 則是繼續停在上一組計算的最後一個 $address$ (因為那個 $data$ 已經用不到了)，並將 ss 收進來的 $data$ 存入該 $address$ 裡，然後再開始新一輪的計算 ($0x00$ 進行累減的話會拉回 $0x28$)。



6. 直到 $cnt == datalength$ ，這時可以判定整個 FIR 計算進入尾聲，將 $last$ 設成 1。 sm_tlast 則會與最後一個 FIR 的計算結果及 sm_tvalid 一起輸出 1，並代表計算結束。 $sm_tlast \&\& sm_tvalid \&\& sm_tready$ 後， ap_idle 和 ap_done 都會在下一個 $cycle$ 設回 1。此時 $host$ 可以讀取 ap_done ，也可以 $program$ ap_start 進行下一組 FIR。

```
always@(posedge axis_clk or negedge axis_rst_n) begin
    if(!axis_rst_n) begin
        status[1] <= 0;
    end
    else if (sm_tlast && sm_tvalid && sm_tready) begin
        status[1] <= 1; // set ap_done after last output is transferred
    end
    else if (araddr == 12'h0 && rready && rvalid) begin
        status[1] <= 0; // reset ap_done after status being read
    end
    else begin
        status[1] <= status[1];
    end
end
```

7. 在我的設計中，各個 read/write channel 基本上是能獨立運作的。但由於初始化 data_RAM (init = 1)時 和 FIR 計算中 (ap_idle = 0) 會對 BRAM 的 address 進行頻繁的變換，為了避免資料汙染，在這兩個狀況下，針對 BRAM 的讀寫是 forbidden 的。
8. 在我的設計中，計算會在 ss_tdata 收進來後的第 2 個 clock 開始計算，並在第12個 clock 得出計算結果。延遲一個 cycle 的原因是 BRAM 的讀取，因此不會影響從ss 收 data，故在最大的效率下 (ss 隨時輸入，sm 隨時輸出)，一個 data 的計算只需要 11 個 clocks，最大化了乘法器和加法器的利用率。

• Testbench

我的 Testbench 主要為以下功能：

1. 在一開始時將 datalength 和 tap coef 寫入 FIR 內部。
2. 向 FIR 確認 tap coef 寫入沒有問題。
3. Programmed ap_start，開始 FIR 的運算，將 samples_triangular_wave.dat 的 data 由 ss 端餵進去 (中間沒有等待，ss_tvalid 常為 1)。同時讀取 sm 端的輸出 (中間沒有等待，sm_tready 常為 1)，將其與 out_gold.dat 的 data 比對。
4. 在過程中讀取 ap_done、ap_idle，確認 FIR 的運作狀態。
5. 上述3.~4.再重複兩次。而在第三次的整個 FIR 運算會進行 wait state 的測試。最初五次 sm 端的讀取會延遲 5 個 clocks，第六到十次則是延遲 30 個clocks。倒數第4, 3, 2次 ss 端的輸入會分別延遲 15, 15, 30 個 clocks。

- Resource usage

```

-----
Start RTL Component Statistics
-----
Detailed RTL Component Info :
+---Adders :
    2 Input   32 Bit   Adders := 1
    2 Input   12 Bit   Adders := 2
    2 Input    6 Bit   Adders := 2
+---Registers :
    32 Bit   Registers := 2
    6 Bit    Registers := 2
    1 Bit    Registers := 10
+---Multipliers :
    32x32   Multipliers := 1
+---Muxes :
    2 Input   32 Bit   Muxes := 12
    2 Input   12 Bit   Muxes := 3
    2 Input    8 Bit   Muxes := 2
    2 Input    6 Bit   Muxes := 1
    5 Input    6 Bit   Muxes := 1
    2 Input    4 Bit   Muxes := 2
    5 Input    3 Bit   Muxes := 1
    2 Input    1 Bit   Muxes := 17
    5 Input    1 Bit   Muxes := 1
    3 Input    1 Bit   Muxes := 1
-----
Finished RTL Component Statistics
-----

```

32 bit adder is for FIR.

12 bit adders is for address offset.

6 bit adders is for read pointer for BRAM (readptr, dreadptr)

32 * 32 Multipliers is for FIR.

LUTs and FFs:

1. Slice Logic						
Site Type	Used	Fixed	Prohibited	Available	Util%	
Slice LUTs*	837	0	0	53200	1.57	
LUT as Logic	837	0	0	53200	1.57	
LUT as Memory	0	0	0	17400	0.00	
Slice Registers	153	0	0	106400	0.14	
Register as Flip Flop	153	0	0	106400	0.14	
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	

BRAM:

2. Memory						
Site Type	Used	Fixed	Prohibited	Available	Util%	
Block RAM Tile	0	0	0	140	0.00	
RAMB36/FIFO*	0	0	0	140	0.00	
RAMB18	0	0	0	280	0.00	

• Timing report

axis_clk is set to around 250MHz

Clock Summary			
Clock	Waveform(ns)	Period(ns)	Frequency(MHz)
axis_clk	{0.000 2.000}	4.000	250.000

WNS = 0.308 ns

Design Timing Summary							
WNS(ns)	TNS(ns)	TNS Falling Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Falling Endpoints	THS Total Endpoints
0.308	0.000	0	261	0.146	0.000	0	261

All user specified timing constraints are met.

WNS is at from clock pin of “readptr[0]” to D pin of “dreadptr[0]”

Max Delay Paths				
Slack (MET) :	0.308ns	[required time - arrival time]		
Source:	genblk1.readptr_reg[0]/C	(rising edge-triggered cell FDCE clocked by axis_clk {rise@0.000ns fall@2.000ns period=4.000ns})		
Destination:	genblk1.dreadptr_reg[0]/D	(rising edge-triggered cell FDCE clocked by axis_clk {rise@0.000ns fall@2.000ns period=4.000ns})		
Path Group:	axis_clk			
Path Type:	Setup (Max at Slow Process Corner)			
Requirement:	4.000ns (axis_clk rise@4.000ns - axis_clk rise@0.000ns)			
Data Path Delay:	3.556ns (logic 1.145ns (32.199%) route 2.411ns (67.801%))			
Logic Levels:	4 (LUT4=1 LUT5=2 LUT6=1)			
Clock Path Skew:	-0.145ns (DCD - SCD + CPR)			
Destination Clock Delay (DCD):	2.128ns = (6.128 - 4.000)			
Source Clock Delay (SCD):	2.456ns			
Clock Pessimism Removal (CPR):	0.184ns			
Clock Uncertainty:	0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE			
Total System Jitter (TSJ):	0.071ns			
Total Input Jitter (TIJ):	0.000ns			
Discrete Jitter (DJ):	0.000ns			
Phase Error (PE):	0.000ns			
Location	Delay type	Incr(ns)	Path(ns)	Netlist Resource(s)
(clock axis_clk rise edge)				
		0.000	0.000 r	
net (fo=0)		0.000	0.000 r	axis_clk (IN)
		0.000	0.000 r	axis_clk
				r axis_clk_IBUF_inst/I
IBUF (Prop_ibuf_I_0)		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_0)		0.101	1.872 r	axis_clk_IBUF_BUFG_inst/O
net (fo=153, unplaced)		0.584	2.456	axis_clk_IBUF_BUFG
FDCE				r genblk1.readptr_reg[0]/C

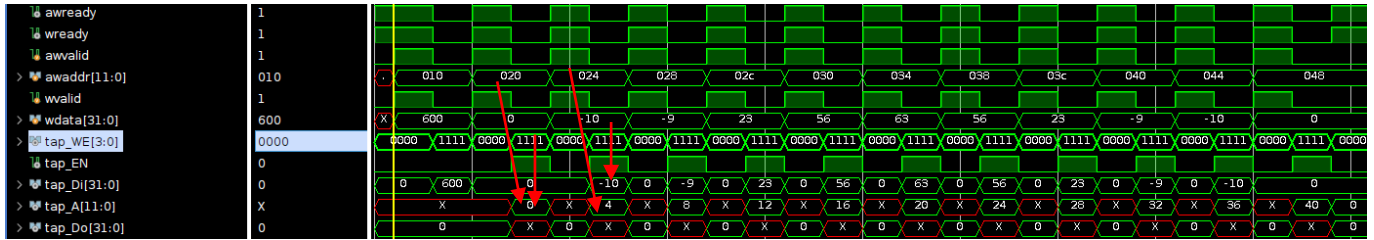
FDCE (Prop_fdce_C_0)		0.478	2.934 f	genblk1.readptr_reg[0]/Q
net (fo=8, unplaced)		1.003	3.937	genblk1.readptr_reg_n_0_[0]
				f genblk1.readptr[5]_i_3/I0
LUT6 (Prop_lut6_I0_0)		0.295	4.232 f	genblk1.readptr[5]_i_3_0
net (fo=6, unplaced)		0.481	4.713	genblk1.readptr[5]_i_3_n_0
				f genblk1.dreadptr[5]_i_4/I4
LUT5 (Prop_lut5_I4_0)		0.124	4.837 r	genblk1.dreadptr[5]_i_4_0
net (fo=3, unplaced)		0.467	5.304	genblk1.dreadptr[5]_i_4_n_0
				r genblk1.dreadptr[2]_i_2/I2
LUT4 (Prop_lut4_I2_0)		0.124	5.428 r	genblk1.dreadptr[2]_i_2_0
net (fo=2, unplaced)		0.460	5.888	genblk1.dreadptr[2]_i_2_n_0
				r genblk1.dreadptr[0]_i_1/I2
LUT5 (Prop_lut5_I2_0)		0.124	6.012 r	genblk1.dreadptr[0]_i_1_0
net (fo=1, unplaced)		0.000	6.012	genblk1.dreadptr[0]
FDCE				r genblk1.dreadptr_reg[0]/D

(clock axis_clk rise edge)				
		4.000	4.000 r	
net (fo=0)		0.000	4.000 r	axis_clk (IN)
		0.000	4.000 r	axis_clk
				r axis_clk_IBUF_inst/I
IBUF (Prop_ibuf_I_0)		0.838	4.838 r	axis_clk_IBUF_inst/O
net (fo=1, unplaced)		0.760	5.598	axis_clk_IBUF
				r axis_clk_IBUF_BUFG_inst/I
BUFG (Prop_bufg_I_0)		0.091	5.689 r	axis_clk_IBUF_BUFG_inst/O
net (fo=153, unplaced)		0.439	6.128	axis_clk_IBUF_BUFG
FDCE				r genblk1.dreadptr_reg[0]/C
		0.184	6.311	
clock pessimism		-0.035	6.276	
clock uncertainty		0.044	6.320	genblk1.dreadptr_reg[0]
FDCE (Setup_fdce_C_D)				
required time			6.320	
arrival time			-6.012	
slack			0.308	

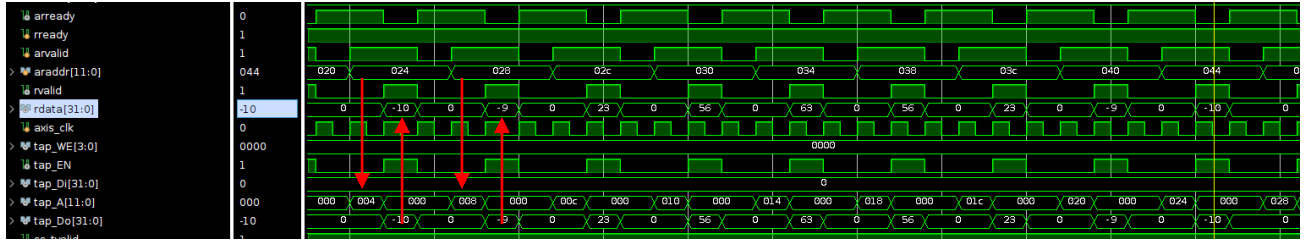
我猜想這可能是因為這條 path 有過多的 mux 控制，導致 net delay 偏長。

Waveform

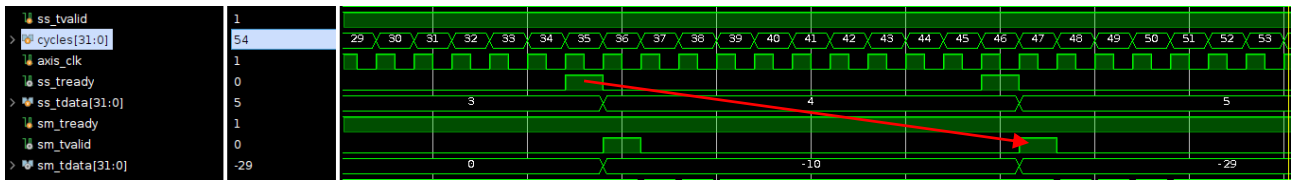
Coefficient program:



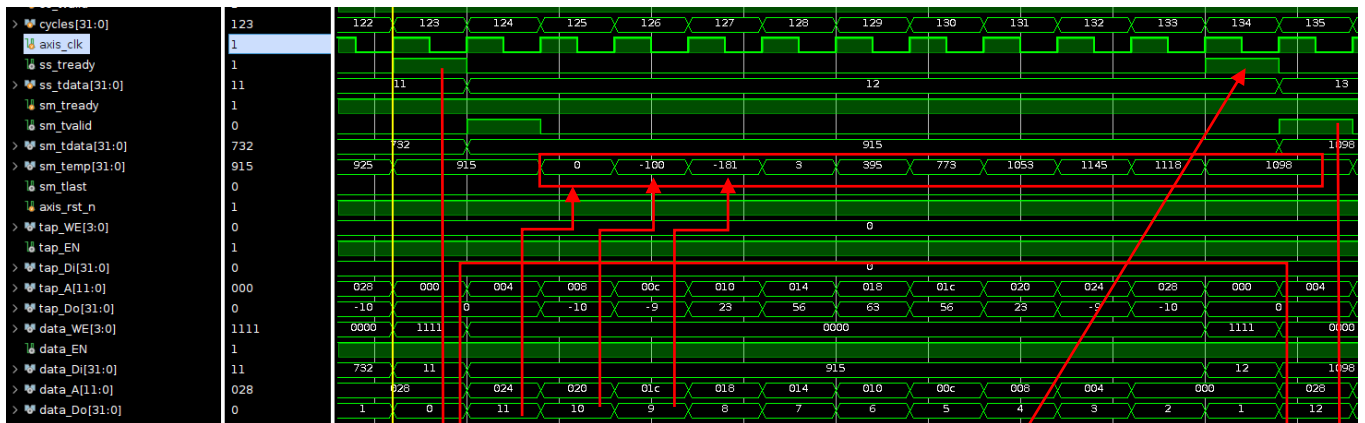
Coefficient read back:



Data stream-in and Data stream-out:



RAM access control when FIR computing:



Write
ss_data
to BRAM

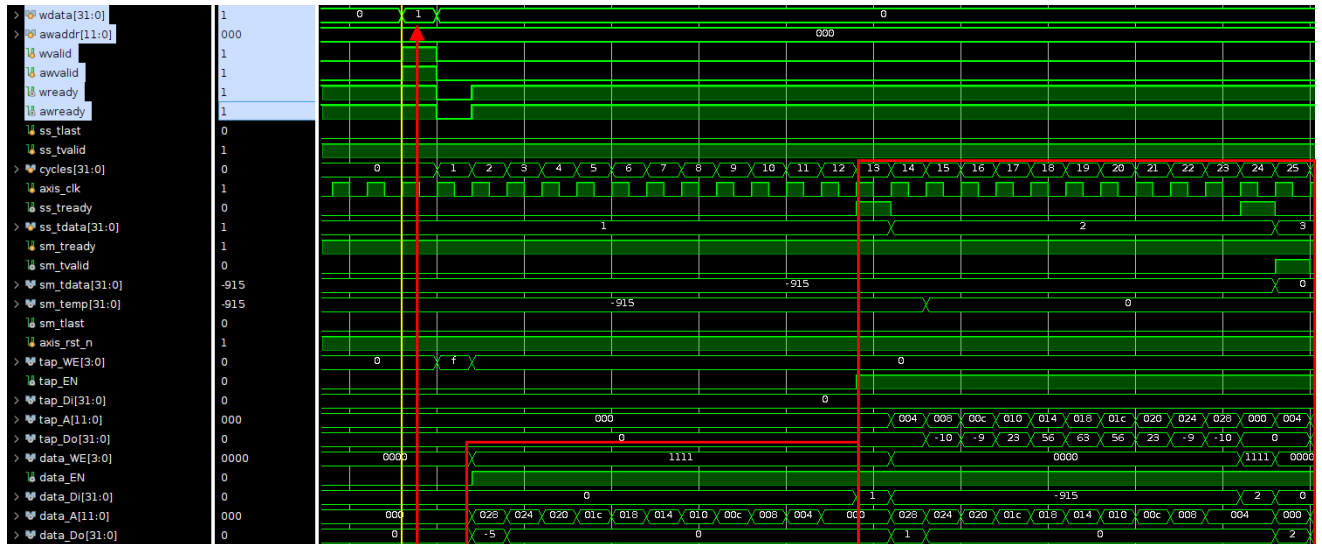
FIR
Computing

sm_data
Valid

Next FIR
start here

- Waveform

Program ap_start:

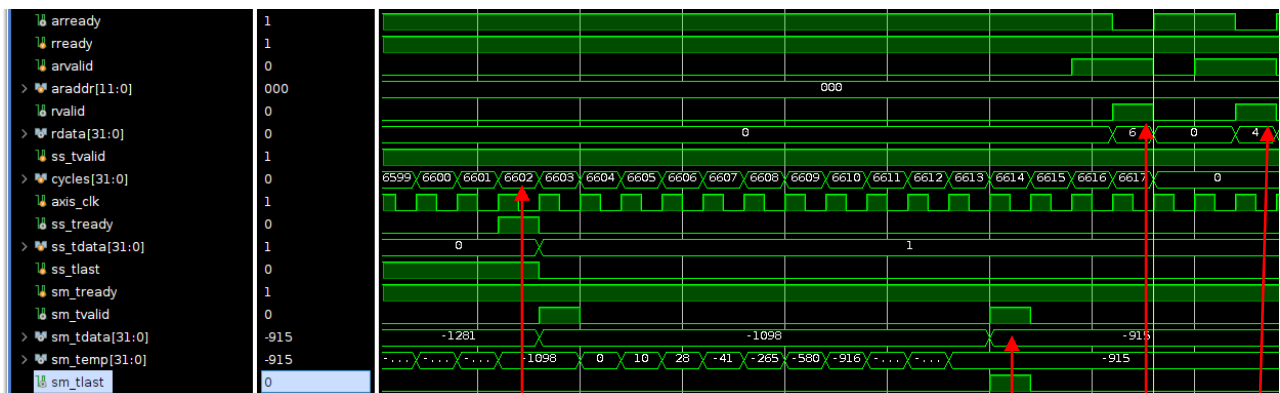


ap_start
programmed

Initialize
data BRAM

Start FIR
processing

Read ap_done and ap_idle:



read
last ss_data

read
last sm_data

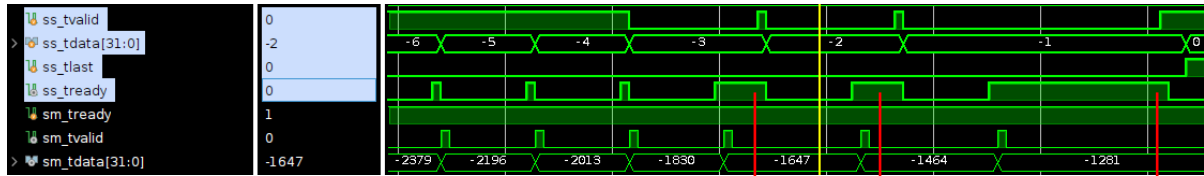
ap_idle
read

read ap_done
Note that ap_idle is also 1

Clock cycles used from ap_start to ap_done : 6617.

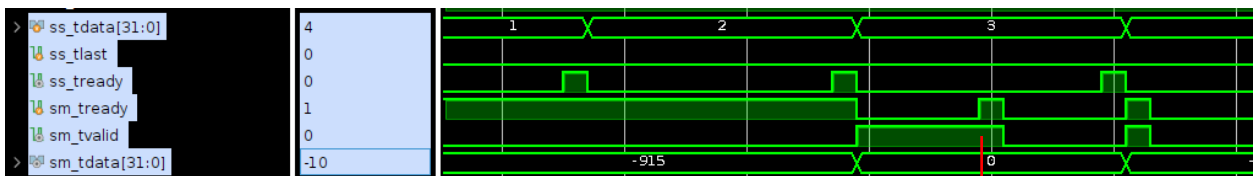
- Waveform

ss input wait state test:



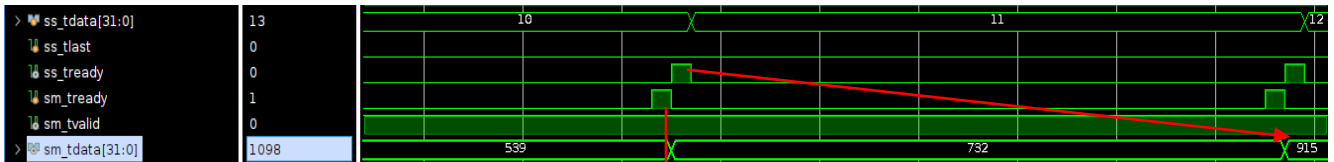
ss_tready waits for
ss_tvalid

sm output wait state test (5 cycles wait):



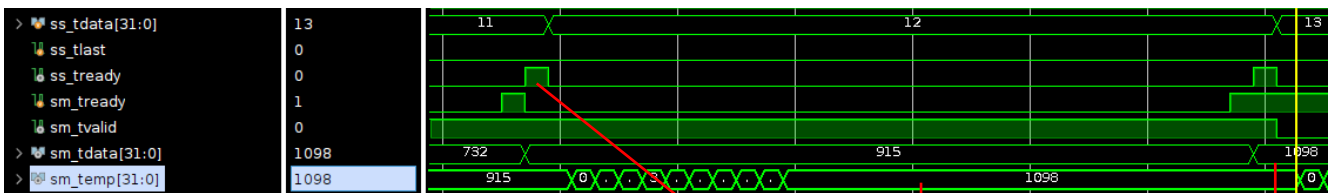
sm_tvalid waits for
sm_tready

sm output wait state test (30 cycles wait):



sm_tvalid waits for
sm_tready

The FIR
result of
"10" is here



The FIR
result of "11"
is here

Pause FIR since
there is no room

sm_tvalid is 0 after outputs
(915 and 1098) has been read.