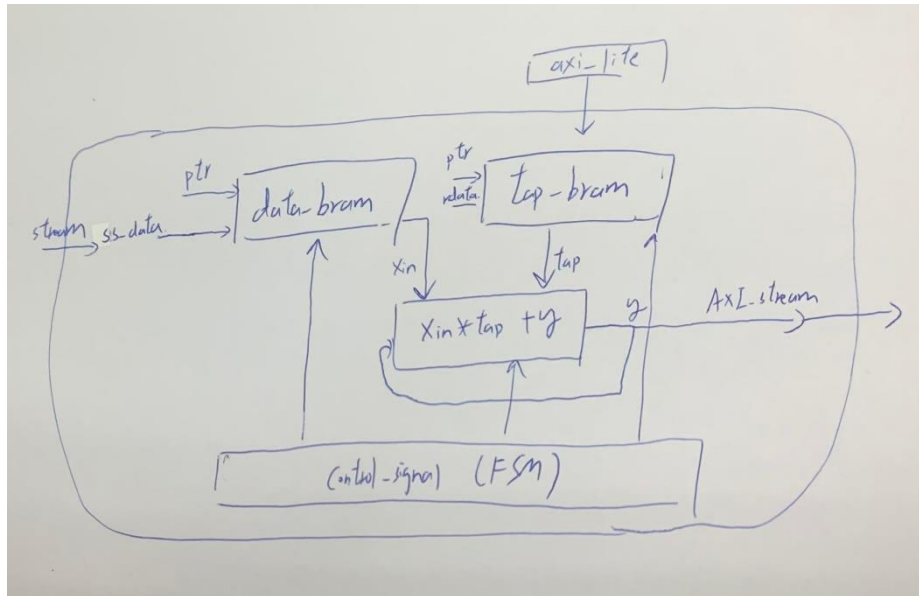


一、Block Diagram



二、Describe operation, e.g.

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

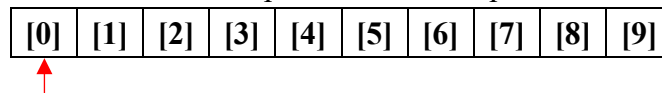
1. data-in

data-in(xin)從，axi-stream 來，把 ss_tready 拉 1，tb 端就會一直丟 xin 進來，把 ss_data 直接對接 bram 的 data_di，我會把 bram 裡的值全都歸零在一開始讀 coefficient 的時候，然後會有一個紀錄現在該寫入的地址，每寫入一個 xin 他就會往下個地址指(0、4、8...)，藉此將 xin 依序寫入 bram。寫滿 11 個的話就會把最前面的 data 蓋掉。

2. tap parameters

data-length 跟 coefficient 從 axi-lite 拿，data-length 用一個 flip-flop 存起來，而 coefficient 存進去 Bram11 來儲存，coefficient 後面都可以固定不變。

- How to access shiftram and tapRAM to do computation



這邊我們需要 shift 來做 data 的計算，但每次計算都把整個 ram 的 data shift 一個其實很耗 power，假設上面的 data 我們每次計算只拿最左邊的值，且做完要全部往左 shift 1 格的方式依序拿 tap[0]~[10]，我們其實可以不用 shift，反而用 ptr 去指我們現在該讀哪一個值就好。Ptr 從 0 依序加到 10，每次讀值根據 ptr 當 address 去從 bram 將資料讀出。

Ex: 計算第一次用[0]，第二次用[1]我們可以讓 ptr 依序往右指即可依序從[0]讀到[10]。

➤ How ap_done is generated

最後一筆 xin 進來時，tb 會將 tlast 拉起來跟我告知這是最後一筆資料，所以當 tlast 拉起來，電路可以把 tlast 存起來，界已知道這是最後一次的運算，算完就可以跳到狀態 done 並把 ap-done 拉起來。

➤ How ap_start is generated

當 axi-lite 傳送地址是 0x00 時，且 wvalid 跟 wready、wdata[0]=1 表示 tb 要傳送 start 訊號，ap-idle 是 1 的話，就可以把 tb 發送的 start 記錄在電路中，ap_start 拉 1(這邊我電路的 ap_start 只會拉一個 cycle)

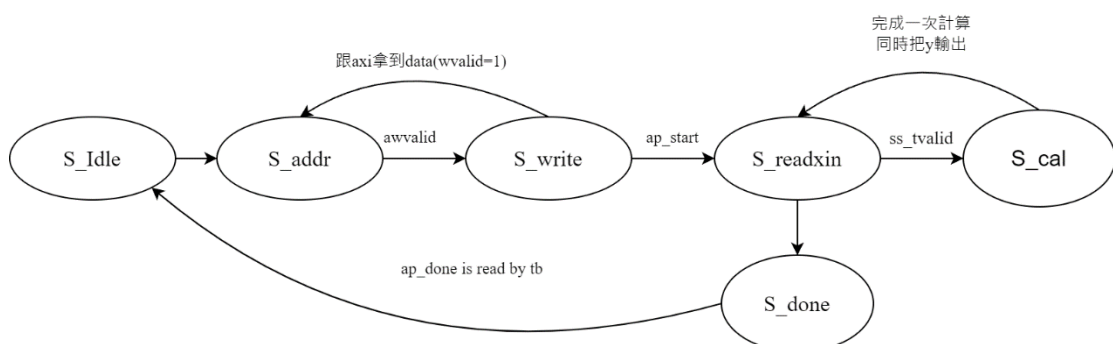
➤ How ap_idle is generated

當 sample 電路的 ap_start 拉 1，表示電路開始算 fir 了 idle 拉 0，直到跳到狀態 done，最後一筆計算完畢且輸出 idle 拉 1。

```
// ap.start idle done // // ap[0]-start ap[1]-done ap[2]-idle
always@(posedge axis_clk or negedge axis_rst_n)begin
    if(!axis_rst_n)
        ap[2:0] <= 3'b100; //set idle = 1
    else
        if((awaddr_reg == 12'h00) &&wvalid &&wready&& ap[2])//ap start //ap[2]idle=1 start is valid
            ap[2:0] <= wdata[2:0];
        else if (ap[0])
            ap <= ap&(3'b000); //when sample ap-start set idle to 0
        else if (ns==S_done) //ap[1] done ap[2]idle
            ap <= ap|3'b110;
        else if(ns == S_Idle) // ap[1] done is transfer to tb
            ap <=ap &3'b101;
        else
            ap <= ap;
end
```

➤ 電路運作

```
parameter S_Idle = 4'd0 ; //idle
parameter S_addr = 4'd1 ; //axi handshake addr
parameter S_write = 4'd2 ; //write tap to bram
parameter S_readxin = 4'd3; //read xin
parameter S_cal = 4'd4; //calculation y = xin*tap+y
parameter S_done =4'd5;
```



S_idle： 電路 idle

S_addr、S_write：跟 axi 握手拿 data_length、coeffient、ap_start。

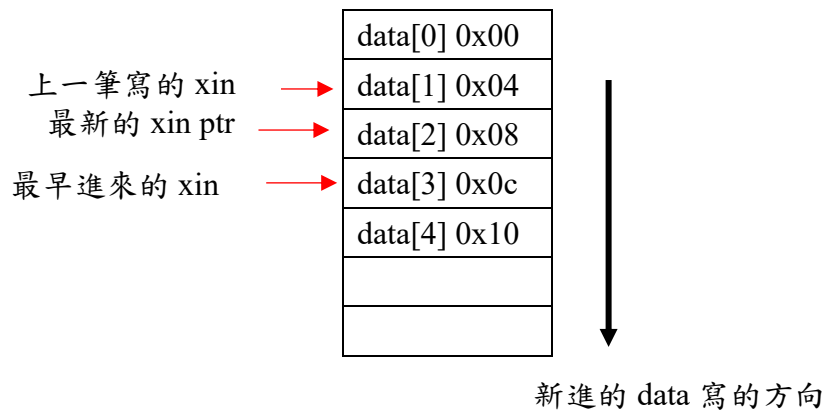
S_addr：確認握地址，awready=1。

S_write：前一狀態拿到地址，這個狀態跟 axi 收 data(wready=1)
拿到 data 如果不是 ap_start 就跳回 S_addr 繼續讀下一筆
tap 或 data_length，直到收到 tb 傳 ap_start 的訊號。

S_readxin：此狀態為讀 xin，在此狀態藉由 stream 收 input-data，
ss-ready=1，直到我們確認從 stream 收到 data，在這邊會
同時把收到的 data 存入 bram 以便後續的運算。

S_cal：此狀態為計算，我們會依序從 2 個 bram 個別讀出 tap、xin
做相乘並累加，結束後跳回 S_readxin 讀新的 data。

S_done：在讀最後一個 xin 時會有 tlast 我們會把這個東西暫存起
來，在進去 S_cal 做一次運算，最後輸出在跳到 S_done。



Bram 每次寫入 xin 都會根據 xin ptr 去寫入，每寫完一次 xin，xin_ptr 就會
往下加(加到最底的話就回歸 0x00)，所以 xin_ptr 就是我們目前最新寫進來的
xin 他的上面是上一筆 xin。在 caculation 時最新的 data 就乘 tap[10]，最早進來
的乘 tap[0]、再來第二早乘 tap[1]依此類推，next_ptr 指到 xin_ptr 表示算完一輪
了就可以輸出。

● Resource usage

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	190	0	0	53200	0.36
LUT as Logic	190	0	0	53200	0.36
LUT as Memory	0	0	0	17400	0.00
Slice Registers	85	0	0	106400	0.08
Register as Flip Flop	85	0	0	106400	0.08
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

1. FF

用了 85 個暫存需要的 data

2. LUT

有數個 LUT (處理 bram address 按照順序 0x00 0x04 0x08.....)

3. 這個 lab 沒有用到 latch

4. 用了 2 個 bram 一個存 data 一個存 coefficient

● Timing Report

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.481 ns	Worst Hold Slack (WHS): 0.142 ns	Worst Pulse Width Slack (WPWS): 3.000 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: 89	Total Number of Endpoints: 89	Total Number of Endpoints: 86

All user specified timing constraints are met.

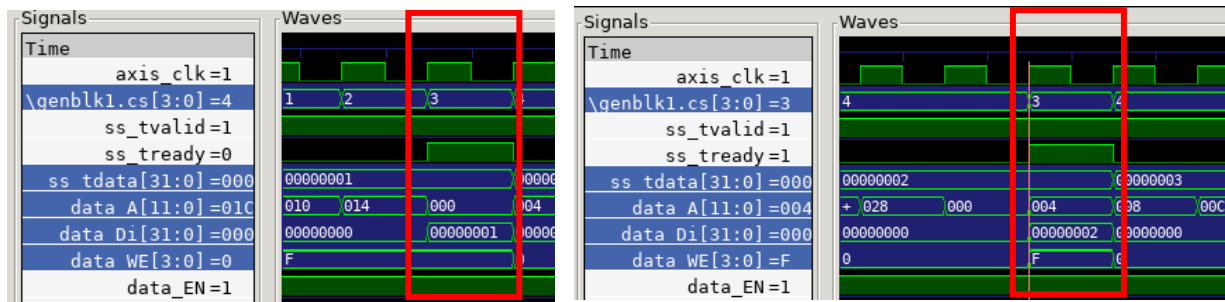
Max Delay Paths

```
Slack (MET) : 0.481ns (required time - arrival time)
Source:      genblk1.data_rdpnr_tristate_oe_reg[5]/C
              (rising edge-triggered cell FDCE clocked by axis_clk {rise@0.000ns fall@0.000ns})
Destination: genblk1.ap_reg[1]/D
              (rising edge-triggered cell FDPE clocked by axis_clk {rise@0.000ns fall@0.000ns})
Path Group:  axis_clk
Path Type:   Setup (Max at Slow Process Corner)
Requirement: 7.000ns (axis_clk rise@7.000ns - axis_clk rise@0.000ns)
Data Path Delay: 6.383ns (logic 2.093ns (32.790%) route 4.290ns (67.210%))
Logic Levels: 7 (CARRY4=1 LUT6=6)
Clock Path Skew: -0.145ns (DCD - SCD + CPR)
  Destination Clock Delay (DCD): 2.128ns = ( 9.128 - 7.000 )
  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
```

```
Slack (MET) : 0.481ns (required time - arrival time)
Source:      genblk1.data_rdpnr_tristate_oe_reg[5]/C
              (rising edge-triggered cell FDCE clocked by axis_clk {rise@0.000ns fall@0.000ns})
Destination: genblk1.ap_reg[2]/D
              (rising edge-triggered cell FDCE clocked by axis_clk {rise@0.000ns fall@0.000ns})
Path Group:  axis_clk
Path Type:   Setup (Max at Slow Process Corner)
Requirement: 7.000ns (axis_clk rise@7.000ns - axis_clk rise@0.000ns)
Data Path Delay: 6.383ns (logic 2.093ns (32.790%) route 4.290ns (67.210%))
Logic Levels: 7 (CARRY4=1 LUT6=6)
Clock Path Skew: -0.145ns (DCD - SCD + CPR)
  Destination Clock Delay (DCD): 2.128ns = ( 9.128 - 7.000 )
  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
```

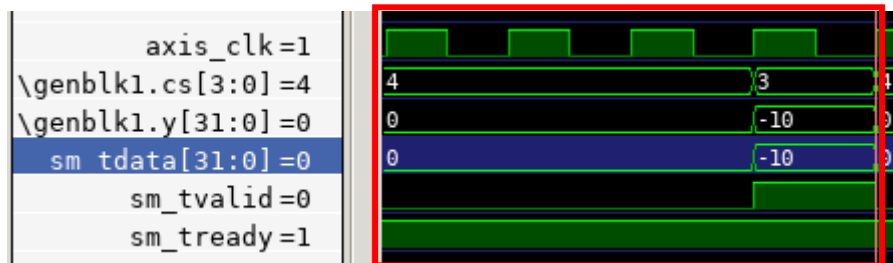
● Simulation Waveform

➤ Data-in stream-in



State3 為 state_read-xin，在這個 state 我們會從 tb 讀一個 xin，tvalid 跟 tready 都是 1 才是確定有讀到 xin，data_Di 會對接 ss_tdata，data_we、A 在這個 state 設定好。

➤ Data-out stream-out

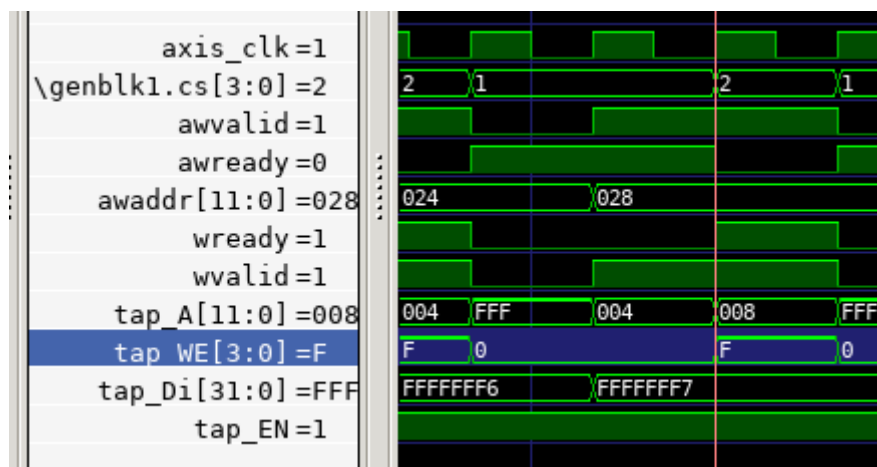


State4 (cal)跳 State3(read_xin)表示我計算完一筆資料了，要讀一個新的 xin 同時輸出算好的 y。

(sm_tvalid 只有在 cs_cal 跳到 red_xin 那時才輸出)

```
always@(posedge axis_clk)begin
    sm_tvalid <= ((cs == S_cal)&&(ns == S_readxin))?1:0;
end
```

➤ Bram access control



2 個 bram 一個就是按照前面 data_in 的方式存 xin，另一個是存 coefficient，我們有狀態 1(跟 axi 握 address，如果不是地址不是 0x00

► FSM

[illegible]

```
// ns //
always@(*)begin
    case (cs)
        S_Idle: ns = S_addr;
        S_addr: ns = (awvalid)?S_write:S_addr;
        S_write: ns = (wvalid)?
            ((awaddr_reg==12'h00 && wdata[0])? S_readxin:S_addr):
            S_write;
        S_readxin: ns = (sm_tvalid && last_out)?S_done:
            (ss_tvalid)? S_cal:S_readxin;
        S_cal : ns = (nxt_ptr == data_wrptr)?S_readxin: S_cal;
        //S_done: ns = S_done;
        S_done :ns = ((araddr_reg == 12'h00)&&(rready&&rvalid))?S_Idle: S_done;
    default: ns = S_cal;
endcase
end
```