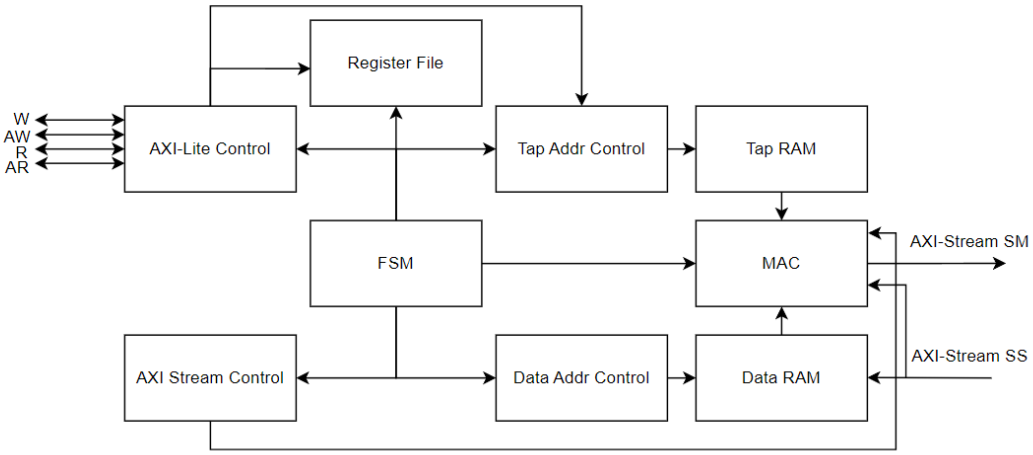
SoC Lab3 Verilog FIR Filter

姓名：江承祐 學號：R11921075

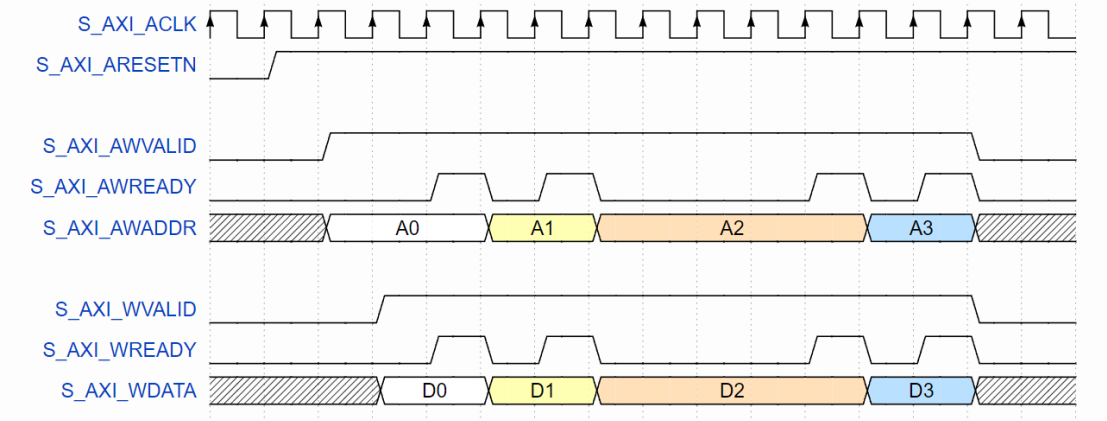
- **Block Diagram**



- **Describe Operation**
  - ✓ **AXI-Lite**

    1.Write Operation

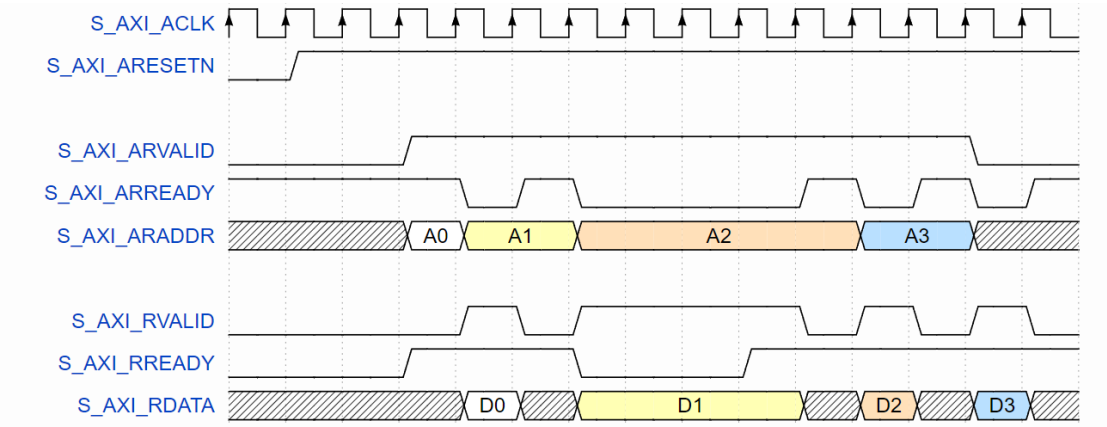
    在本次實作中，對於 Write Operation 我使用了以下的 timing diagram：



    可以看到在這種實作方式，Write Operation 最短需要兩個 cycle 才可以實現。如圖，為了控制方便與節省儲存資源我會等到 AW_valid 與 W_valid 同時為 1 的下一個 cycle 同時將 AWR_ready 與 W_ready 拉高，如此一來在拉高的 cycle，我能確保 AW 與 W 的 Handshake 同時發生，並且在該 cycle 讀取 awaddr 的數值並且對 tap_RAM 或是 register 進行寫入的動作。

    2.Read Operation

    在本次實作中，對於 Read Operation 我使用了以下的 timing diagram：



可以看到我的 AR_ready 與 R_valid 互為反相。這樣可以保證 AR 與 R 的 Handshake 並不會同時間發生。當系統讀取到 AR_valid 與 AR_Ready 同時為 High 的時候，此時 AR 的通道 Handshake 發生，系統可以在這個時間將 register 或是 Tap_RAM 的數值在下一個 cycle 放到 R 通道的 Data，同時也將 R_valid 拉高。由於 R_valid 與 AR_ready 永遠為反向的緣故，我們可以保證在下一個 cycle 中，AR 通道不可能會發生 Handshake。R_valid 會一直為 High 直到 R_ready 為 High，意即 R 通道的 Handshake 成立。當 Handshake 成立之後 R_valid 會拉 Low，AR_ready 會再度拉 High，讓下一個 Request 可以順利進來。

✓ **AXI-Stream**

由於 AXI-Stream 是用於我們處理 FIR Filter 的 Flow control，故是否要拉 SM_Valid 與 SS_ready 的時機完全取決於我們何時有辦法處理新一筆的 Data。在這個實作中，我利用 Counter 去控制何時要將 SM_Valid 與 SS_ready 拉起來。當 Counter 為 0 的時候，SS_ready 將會拉 High，當 SS 端的 Handshake 建立後，SS_data 將會存入 register 中暫存。當 Counter 數到 12 的時候，SM_valid 會拉高並且等待 SM 的 Handshake，當 Handshake 成立的下一個 cycle，Counter 又會回到 0，代表準備好接收下一筆資料。這個循環會一路的持續到每一筆 Data 都處理完為止。

✓ **FSM**

我的狀態機設為三種狀態：Idle、Start 與 Done

Idle：在 Idle 階段會優先的將 Data_Mem 中的每一個數值清為 0，並且等待 ap_start 被拉成 1。這個階段 AXI_Lite 的傳輸持續進行，外部的 Master 會將 Tap 參數與 ap_start 利用 AXI_Lite 傳入 FIR 中。

Start：在 Start 中，FIR 開始處裡來自 AXI_Stream 的 Data，每處理完一筆 Data，fir_data_cnt 將會+1，直到加到 fir_data_cnt == length - 1。這時若最後一筆 Data 也計算完畢並且藉由 SM 傳輸出去後，就會進到 Done State。

Done：這個 cycle 將 ap_done 與 ap_idle 設定為 1，並且進到 Idle 階

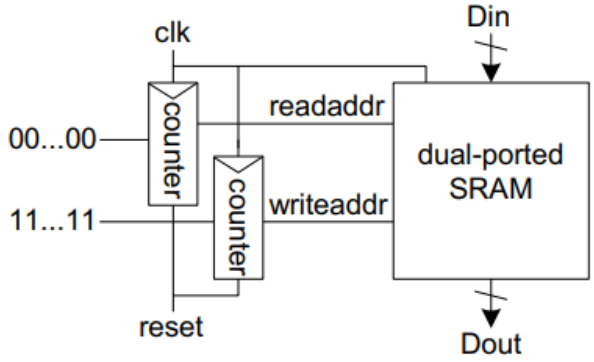
段。

- ✓ **MAC**

  我遵循作業規定只能使用一個加法器與乘法器的規定，在每個 cycle 送入不同的 Tap 參數與 Data_Mem 的數值相乘，並且存在 temp_register 中，算到 fir_counter 為 12 的時候將 sm_valid 拉高，並且將 FIR Stall 住，等待外部接收。

- ✓ **Addr Control**

  在 Data_mem 的 addr control 中，我參考了 VLSI 課程中的 Dense Shift Register 架構，如下圖：



  不同的是由於提供的 SRAM 為 Single port，故我利用不同 counter 的數值去判斷該 cycle 需要進行寫入或是讀出

  我在 counter 為 1 到 10 時分別讀出 data 的不同數值，address 為每個 cycle 往回-4，當數到 0 時則回到 40。
  在 counter 為 11 的時候，為寫入資料，並且在寫入資料之後 address 往上數 4，並且在數到 40 時回到 0

  藉由這種巧妙的 address control，可以將 SRAM 轉換為 Shift register 的工作模式，在新的值寫進來的同時蓋過最舊的數值，並且每次 data 的處理都是照著由新到舊讀取 data。

- ✓ **Ap_port Control**

  Ap_port 由於只需要實現 3 個 port，故我只開三個 bit 去儲存，並且由於只有 ap_start 是可以被寫入的，其他的 bit 數皆沒有辦法藉由外部去寫入。
  我利用上述提到的 FSM 對於不同 ap_ports 進行控制：

  ap_start：在外部透過 axi_lite 拉至 1 之後，FSM 進入 Start 的狀態，並且 ap_start 在 HW 內部將其拉回 0

  ap_Idle：初始狀態為 1，並且在 FSM 狀態為 Idle 時設定為 1，當 ap_start 為 1 的時後將其設定為 0。

  ap_done：初始狀態為 0，並且在 FIR 處理最後一筆 Data 且 Yn 發

生 Handshake 的下一個 cycle 將其拉為 1。

✓ **Testbench specification**

首先會檢查 ap_idle = 1，一開始會先進行 axi_lite 的寫入，並且執行 read_confic_check 檢查 tap 參數是否正確，檢測完畢後 module 開始將 Data 透過 axi-stream 傳輸進來，在傳輸的同時，testbench 也透過 AXI-stream master 來接收與比對 fir 的計算結果，直到 ss channel 送出倒數第二筆 input data 後，會利用 read_config_check 檢測 ap_idle 和 ap_done 的數值是否為 0，若為正常的話才送出最後一筆 input data，然後 sm channel 在接收了 599 筆計算輸出後，也會馬上透過 AXI-lite 檢測 ap_idle 和 ap_done 同時為 0，因為這時候 fir 仍然在計算最後一筆數據，最後接收完畢最後一筆 fir 輸出後，再使用 AXI-lite 檢測 ap_idle 和 ap_done 同時為 1，接著進行第二次模擬，確保設計的狀態機有辦法處理多筆 data。

Resource Usage

Resource	Utilization	Available	Utilization %
LUT	277	53200	0.52
FF	178	106400	0.17
DSP	3	220	1.36
IO	329	125	263.20

- **Timing Report**

Period：13ns

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.078 ns	Worst Hold Slack (WHS):	0.137 ns	Worst Pulse Width Slack (WPWS):	6.000 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:	272	Total Number of Endpoints:	272	Total Number of Endpoints:	179

All user specified timing constraints are met.

Figure:Timing report

Name	Path 1
Slack	0.078ns
Source	fir_cycle_cnt_reg[4]/C (rising edge-triggered cell FDCE clocked by axis_clk {rise@0.000ns fall@6.500ns period=13.000ns})
Destination	FIR_temp_reg[29]/D (rising edge-triggered cell FDCE clocked by axis_clk {rise@0.000ns fall@6.500ns period=13.000ns})
Path Group	axis_clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	13.000ns (axis_clk rise@13.000ns - axis_clk rise@0.000ns)
Data Path Delay	12.817ns (logic 8.748ns (68.251%) route 4.069ns (31.749%))
Logic Levels	12 (CARRY4=5 DSP48E1=2 LUT2=2 LUT3=2 LUT5=1)
Clock Path Skew	-0.145ns
Clock Un...rtainty	0.035ns

Figure:Longest Path

- **Simulation Waveform**

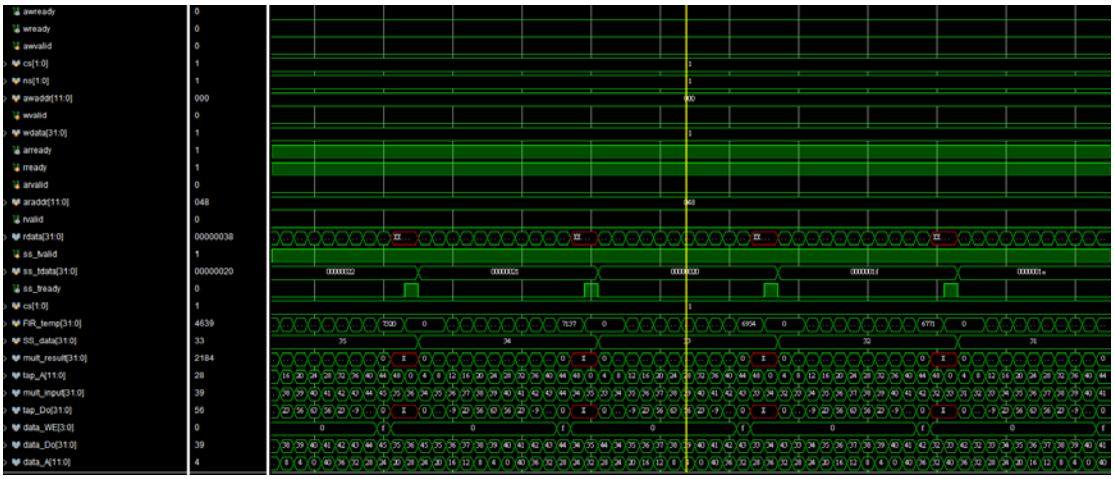
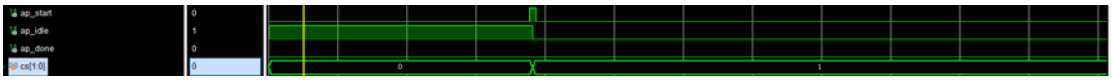


Figure1: FIR Engine Start

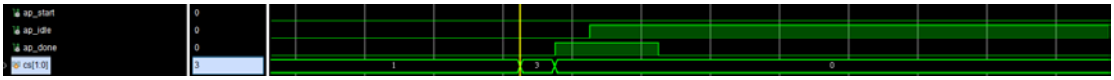
Data-in, stream-in

Data-out Stream-out

RAM access control



Ap_start asserted triggers state changes



Ap_Idle and ap_done asserted after processing data

FSM

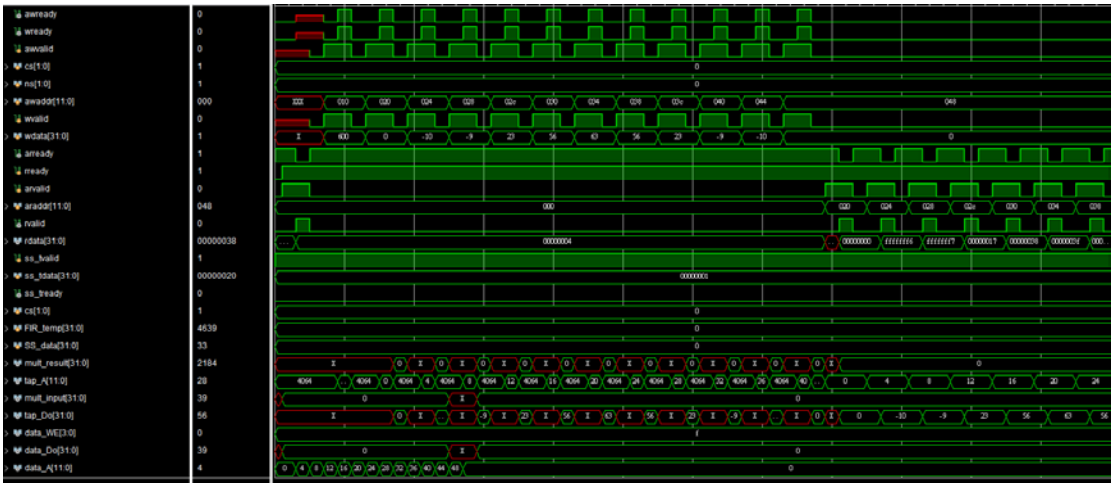


Figure2:AXI-Lite Configuration

Total Cycle: 15768