

# SOC Design Lab3 FIR Report

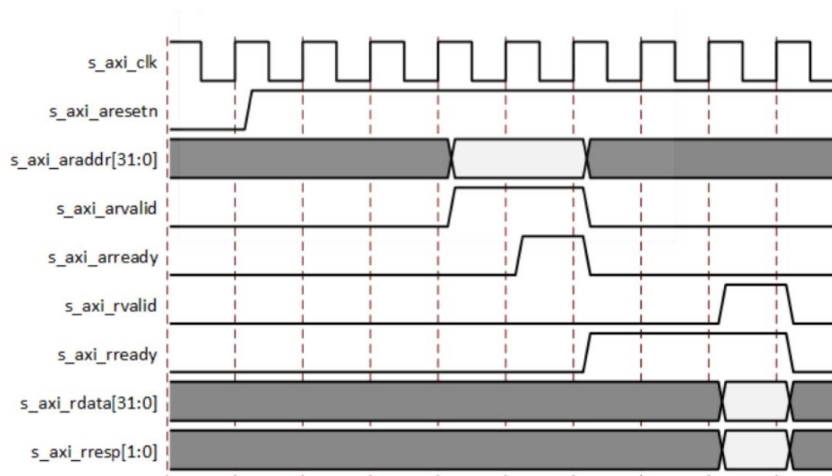
112061533 潘金生

## Protocol introduction:

### 1. AXI-Lite Read transaction:

- Arvalid : master put address on the read access channel
- Rready : master is ready to receive data from slave
- Arready : slave is ready to receive the address
- Rvalid : slave's data channel is valid

When a,c happens, handshake occurs. When b,d happens, finish the transaction.



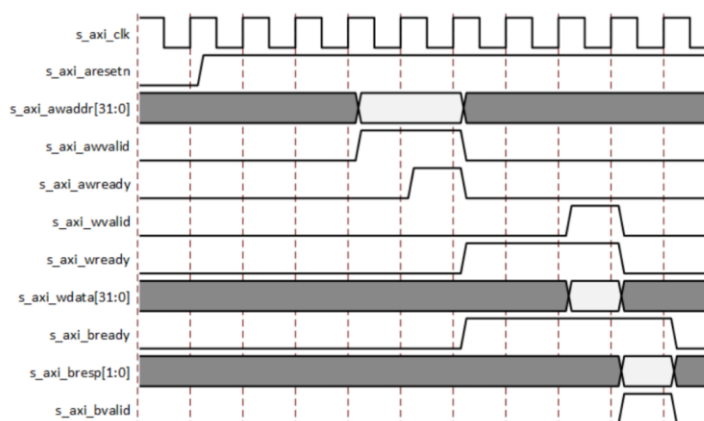
圖一 AXI-Lite Read protocol

### 2. AXI-Lite Write transaction:

- Awvalid, wvalid : Master the address/data is valid.
- Awready, wready : slave asserts.

handshake occurs and finish the transaction.

## AXI4-Lite Write Transaction

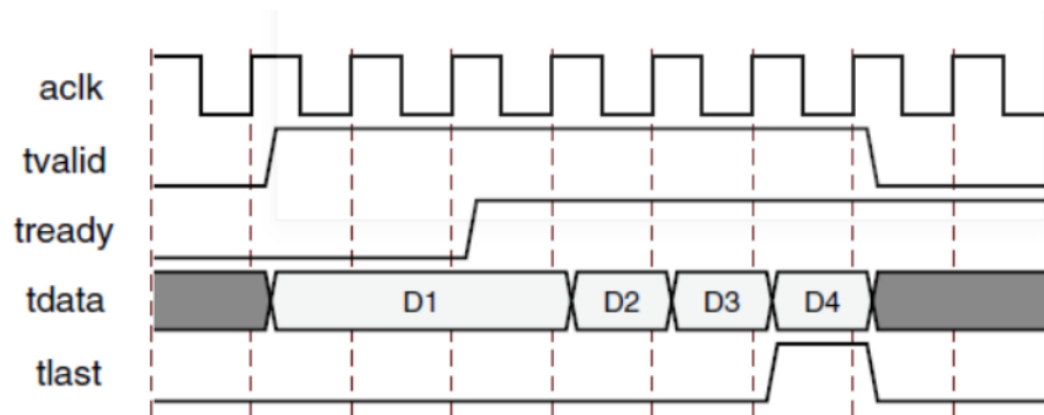
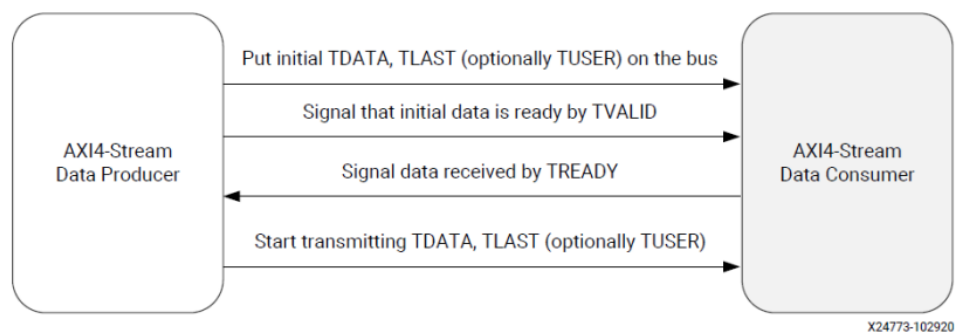


圖二 AXI-Lite Write protocol

3. AXI-Stream Transfer:

- a. Tready is driven by slave side
- b. Tvalid is driven by master side
- c. Send the last data, tlast=1

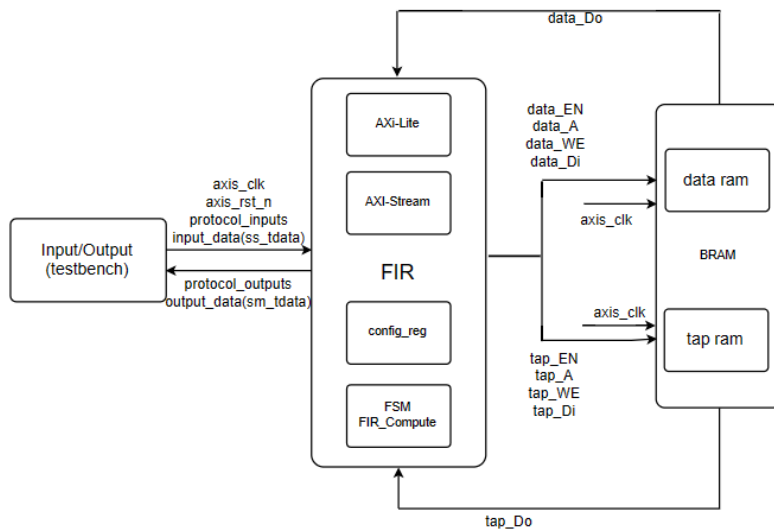
When a,b happens, handshake occurs.



圖三 AXI-Stream protocol

**Block diagram:**

1. Datapath:

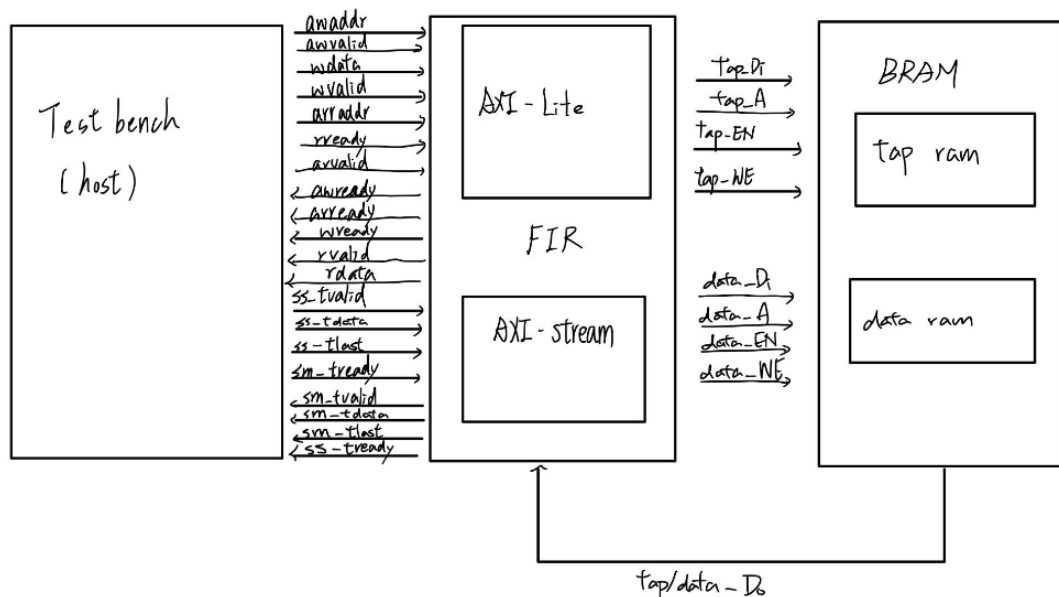


圖四 block diagram

## 2. Control signals:

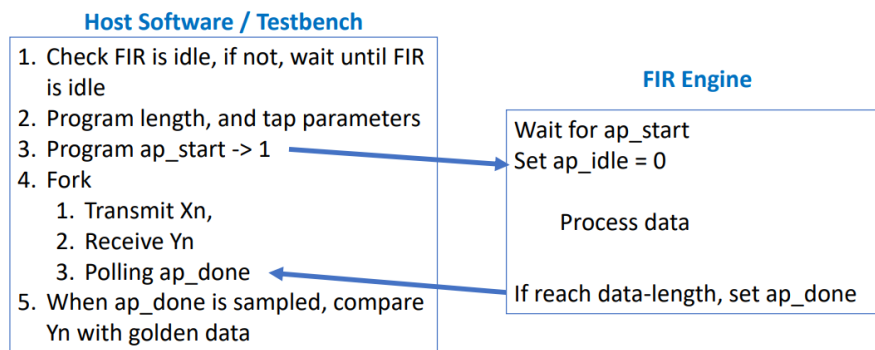
控制信號主要可以分為兩類:

- 對 RAM 的控制: 透過控制訊號 data\_EN, data\_WE, tap\_EN, tap\_WE 來進行 RAM 的操作, tap/data\_WE 為 1 時, 才能寫入資料, tap/data\_EN 為 1 時, 才能寫入或讀出資料。其中 tap/data\_Di 為寫入 ram 資料, tap/data\_Do 為讀出 ram 資料, tap/data\_address 為 ram 地址。
- 對輸入/輸出的控制: 透過控制訊號 ss\_tready, ss\_tvalid, sm\_tready, sm\_tvalid 來控制何時輸入資料和輸出資料。Ss 代表輸入, sm 代表輸出, 相關 protocol 如上述 introduction。當進行最後一筆資料完成, ss/sm\_tlast=1。
- 對係數和 configuration register 的控制: 透過控制訊號 awready, wready, awvaild, wvalid, arready, arvalid, rvalid, rready 控制何時寫入資料和讀取資料。Aw 代表寫入, ar 代表讀出, 相關 protocol 如上述 introduction。



## Describe Operation:

### Host software / Testbench Programming Sequence



上圖為 FIR 的操作流程，先把 tap 存進 tap ram 中，等到 ap\_start，開始傳送資料並進行 shift，和進行乘加運算，當運算到 data\_length 的時候，ap\_done 為 1，並和正解進行比較。

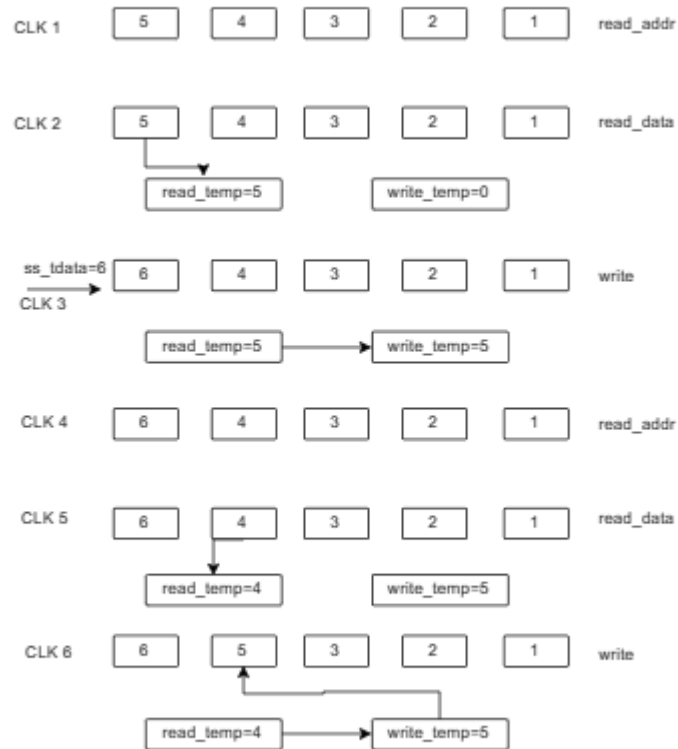
#### 1. How to receive data-in and tap parameters and place into SRAM:

首先，Tap parameters 採用 AXI-Lite interface 寫入，透過 tb 的 for(k=0; k< Tape\_Num; k=k+1) begin config\_write(12'h20+k\*4, coef[k]); end 來執行，AXI-Lite Write 的方式是當 awvalid 和 awready 為 1 且 wvalid 也為 1，此時 awaddr 和 wdata 也準備好，等到 wready 為 1 時，此時就能寫入 SRAM。

Data\_in 是透過 AXI-Stream interface 寫入，透過 tb 的 task ss 來執行，當 ss\_tvalid=1 時，data 會透過 ss\_tdata 傳遞，當 ss\_tready=1 時才開始傳入資料，傳至最後一筆時，ss\_tlast 會等於 1。

#### 2. How to access shiftram and tapRAM to do computation:

Shiftram 的完成方式可以參考下圖，首先會透過 33 個 CLK 寫入資料和 shift data，再者，由於 tapRAM 資料由新到舊的地址為 0x28→0x00，而 dataRAM 資料由新到舊 0x00→0x28，根據 FIR 公式可以發現 tapRAM 和 dataRAM 相同地址取出資料即可相乘，存在暫存器後，把相乘的值累計相加即是結果，因此需要花費 22 個 CLK。



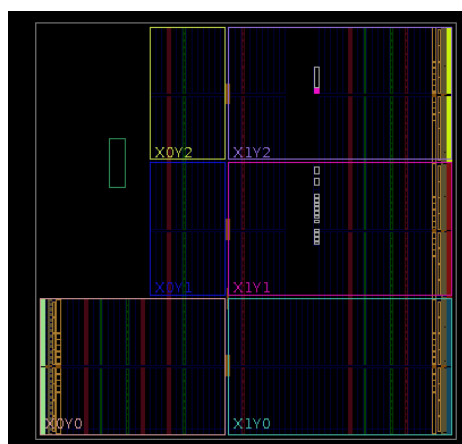
圖五 shiftram 實作方法

### 3. How ap\_done is generated.

透過 counter 紀錄運算到第幾筆資料，到達寫入的 data length，便紀錄 ap\_done。

### Resource Usage:

下圖為合成後的電路示意圖：



圖六 合成後電路圖

| Report Cell Usage: |         |       |
|--------------------|---------|-------|
|                    | Cell    | Count |
| 1                  | BUFG    | 8     |
| 2                  | CARRY4  | 21    |
| 3                  | DSP48E1 | 3     |
| 4                  | LUT1    | 4     |
| 5                  | LUT2    | 46    |
| 6                  | LUT3    | 41    |
| 7                  | LUT4    | 47    |
| 8                  | LUT5    | 32    |
| 9                  | LUT6    | 76    |
| 10                 | FDCE    | 15    |
| 11                 | LD      | 208   |
| 12                 | LDC     | 64    |
| 13                 | LDP     | 1     |
| 14                 | IBUF    | 161   |
| 15                 | OBUF    | 169   |

圖七 Cell Usage

## 1. Slice Logic

| Site Type             | Used | Fixed | Prohibited | Available | Util% |
|-----------------------|------|-------|------------|-----------|-------|
| Slice LUTs*           | 201  | 0     | 0          | 53200     | 0.38  |
| LUT as Logic          | 201  | 0     | 0          | 53200     | 0.38  |
| LUT as Memory         | 0    | 0     | 0          | 17400     | 0.00  |
| Slice Registers       | 288  | 0     | 0          | 106400    | 0.27  |
| Register as Flip Flop | 15   | 0     | 0          | 106400    | 0.01  |
| Register as Latch     | 273  | 0     | 0          | 106400    | 0.26  |
| F7 Muxes              | 0    | 0     | 0          | 26600     | 0.00  |
| F8 Muxes              | 0    | 0     | 0          | 13300     | 0.00  |

## 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  |

圖八 Logic and memory

## Timing Report:

### 1. Slack & Max delay path

根據 timing report 報告中顯示，可以發現 Data Path Delay 時間為 13.770ns，為我的 Data arrival time，而根據我所設置的 Input delay, Output Delay, Clock Uncertainty 時間分別為 2ns, 1ns, 0.035ns，我給定 clock cycle time 是 20ns，因此我的 Data required time 為 16.965ns(20-2-1-0.035)，因此我整個電路最長路徑的 slack 約為 3.194ns。

| Setup                                | Hold                             | Pulse Width                                       |
|--------------------------------------|----------------------------------|---|
| Worst Negative Slack (WNS): 3.194 ns | Worst Hold Slack (WHS): 1.970 ns | Worst Pulse Width Slack (WPWS): 9.500 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: 78        | Total Number of Endpoints: 78    | Total Number of Endpoints: 16                     |

All user specified timing constraints are met.

圖九 slack

| Max Delay Paths     |  |
|---------------------|--|
| Slack (MET) :       | 3.194ns (required time - arrival time)   |
| Source:             | data_Do[16]<br>(input port clocked by axis_clk {rise@0.000ns fall@10.000ns period=20.000ns})   |
| Destination:        | sm_tdata[31]<br>(output port clocked by axis_clk {rise@0.000ns fall@10.000ns period=20.000ns}) |
| Path Group:         | axis_clk   |
| Path Type:          | Max at Slow Process Corner   |
| Requirement:        | 20.000ns (axis_clk rise@20.000ns - axis_clk rise@0.000ns)                                      |
| Data Path Delay:    | 13.770ns (Logic 10.689ns (77.625%) route 3.081ns (22.375%))                                    |
| Logic Levels:       | 10 (CARRY4=4 DSP48E1=2 IBUF=1 LUT2=1 LUT6=1 OBUF=1)  |
| Input Delay:        | 2.000ns  |
| Output Delay:       | 1.000ns  |
| 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  |

圖十 max delay paths

| (clock axis_clk rise edge) |        |          |
|----------------------------|--------|----------|
|                            | 20.000 | 20.000 r |
| clock pessimism            | 0.000  | 20.000   |
| clock uncertainty          | -0.035 | 19.965   |
| output delay               | -1.000 | 18.965   |
| -----                      |        |          |
| required time              |        | 18.965   |
| arrival time               |        | -15.770  |
| -----                      |        |          |
| slack                      |        | 3.194    |

圖十一 slack 計算

## 2. 最大頻率

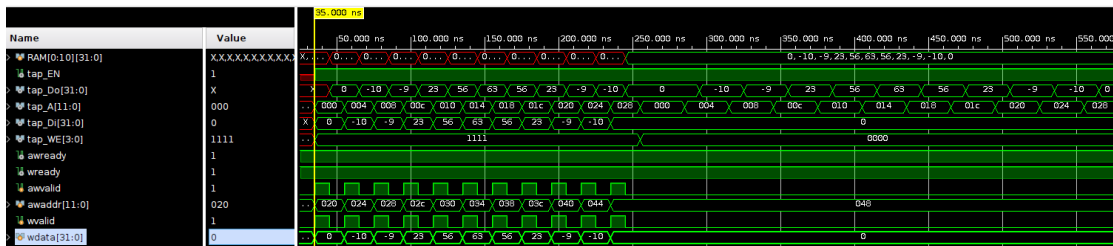
尋找最大操作頻率時，需要先找到最長延遲路徑，從 Timing report 可以發現，FIR 運算會造成相當大的 Delay 且與其他比對後，發現他是最長路徑，因此我們需要把 clock cycle time 提高，從報告可以發現，我的 Data path delay 為 13.770ns，因此最高操作頻率，大概可以落在 71.4MHz。

## Simulation Waveform:

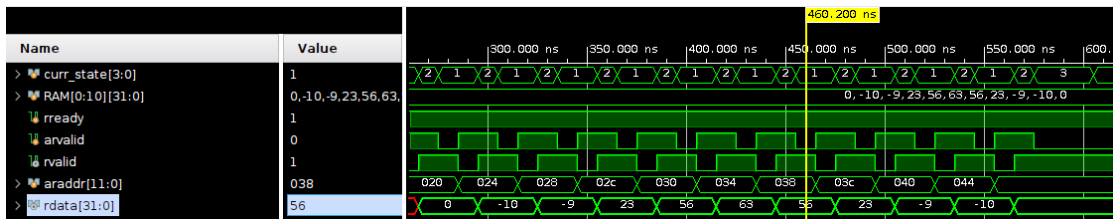
### 1. Coefficient Program and read back:

當 awready, awvalid, wready, wvalid 等於 1 的時候，寫入地址 awaddr 和資料 wdata，可以看下圖的 tap\_RAM 裡有存好的資料，擁有完整的 11 個係數在裡面。

當 arvalid, rready 等於 1 的時候，先讀出地址 araddr 等到下一個 clk 在讀出資料 rdata。



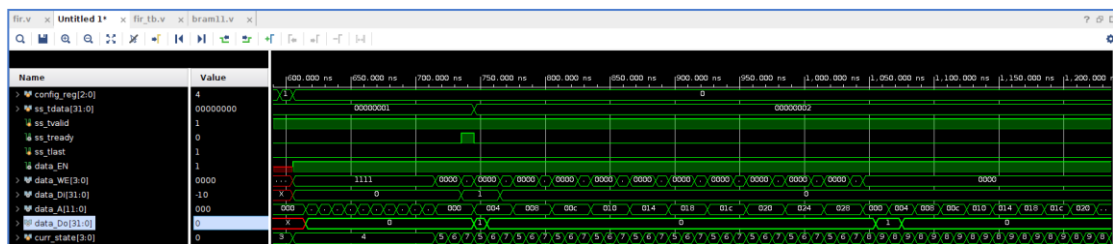
圖十二 寫入係數



圖十三 讀出係數

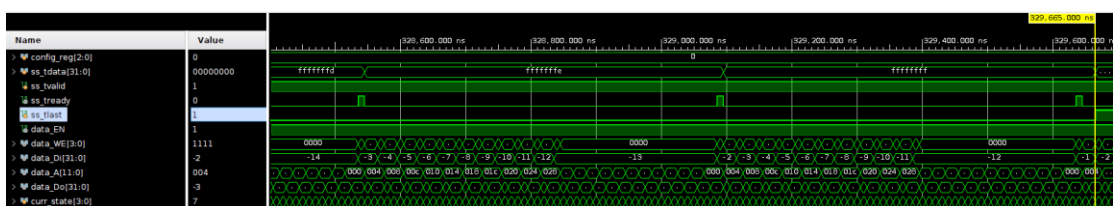
## 2. Data\_in, Stream in

如下圖，可以看到對應的位置有對應的資料，當 ss\_tready=1，便傳送資料。



圖十四 寫入資料

寫入最後一筆資料時，ss\_tlast = 1。

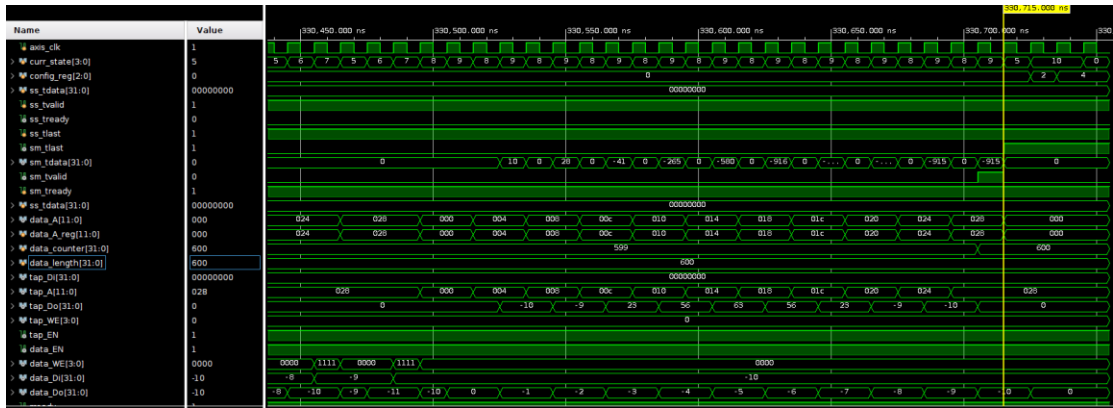


圖十五 寫入最後一筆資料

## 3. Data\_out, Stream out

如下圖，可以發現相對應的位置有相對應的輸出，並且最後一筆 data 時，sm\_tlast=1。



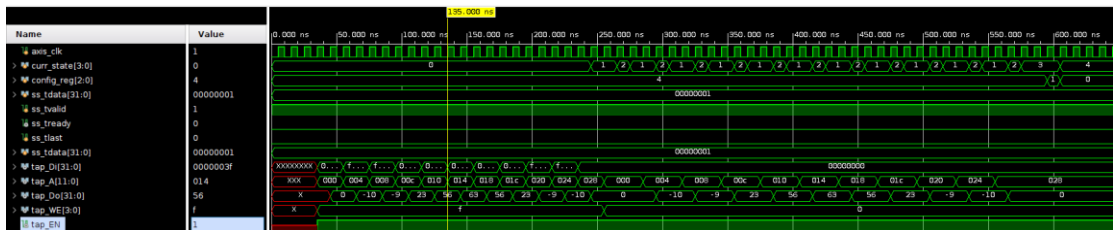


圖十六 輸出最後一筆資料

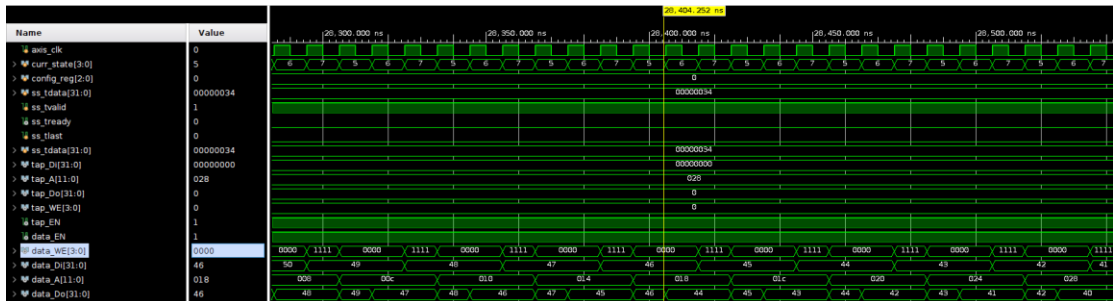
#### 4. RAM access control

寫入資料的時候 data\_WE=4'b1111, data\_EN=1, tap\_WE=4'b1111, tap\_EN=1

讀取資料的時候 data\_WE=4'b0000, data\_EN=1, tap\_WE=4'b0000, tap\_EN=1



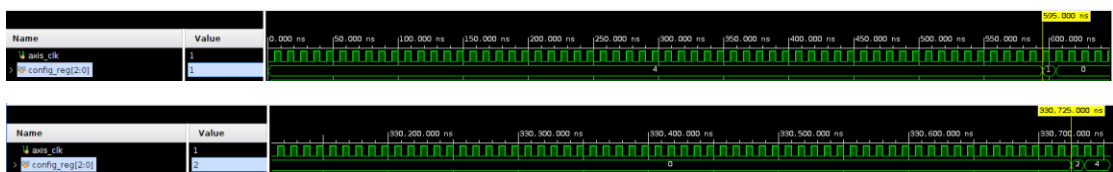
圖十七 寫入係數和讀取係數



圖十八 寫入 data 和讀取 data

#### 5. Measure # of clock cycles from ap\_start to ap\_done

如下圖可知，clock cycle 為 $(330725-595)/10 = 33013$ ，需花費 33013clock cycle 來完成。



圖十九 ap\_start 和 ap\_done time

## 6. FSM

FSM 我主要分成 11 個狀態：

axilite\_w：寫入係數。

axilite\_addr\_r：讀取係數地址。

axilite\_data\_r：讀取係數。

start：等待 ap\_start。

stream\_init：初始化 dataRAM，存放 8 個 0。

stream\_addr\_r：讀取寫入資料地址和進行 shift 資料地址。

stream\_data\_r：讀取資料。

stream\_data\_w：寫入 0x00 地址資料和進行 shift。

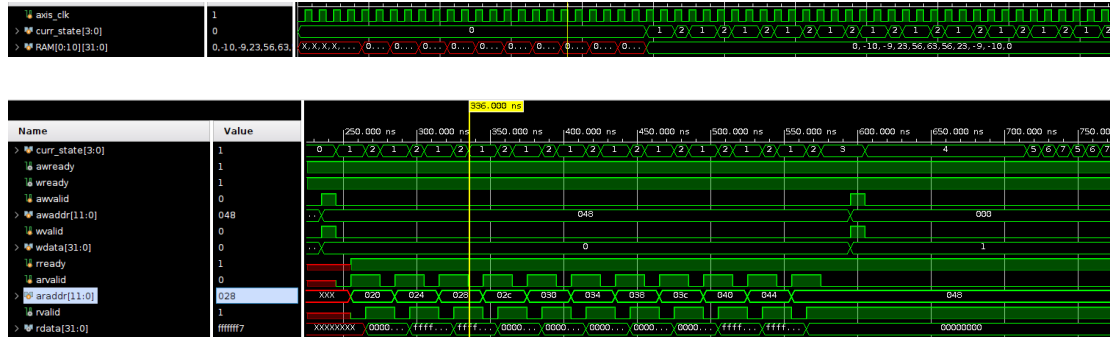
result\_addr：fir 運算地址。

result\_data：fir 累計乘加和結果。

final\_stage：fir 執行結束，處理 ap\_idle 和 ap\_done。

```
parameter axilite_w = 0;
parameter axilite_addr_r = 1;
parameter axilite_data_r = 2;
parameter start = 3;
parameter stream_init = 4;
parameter stream_addr_r = 5;
parameter stream_data_r = 6;
parameter stream_data_w = 7;
parameter result_addr = 8;
parameter result_data = 9;
parameter final_stage = 10;
```

圖二十 State code

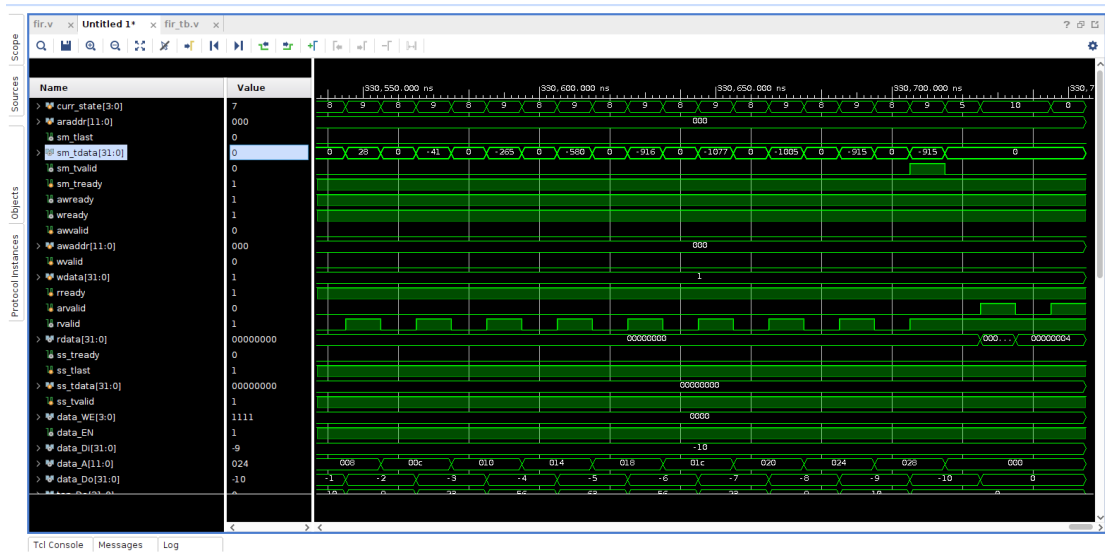


圖二十一 axilite\_w, axilite\_r, axilite\_data\_r 狀態



圖二十二 start, stream\_init, stream\_addr\_r, stream\_data\_r, stream\_data\_w, result\_addr, result\_data 狀

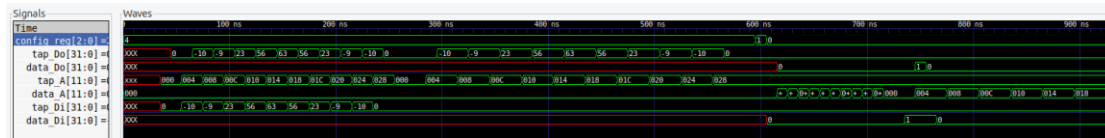
態



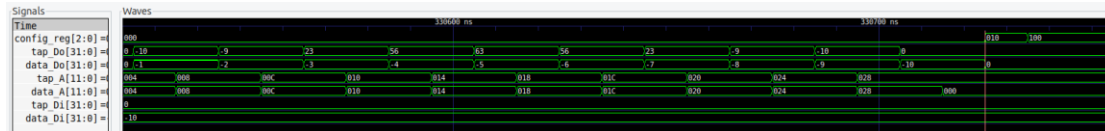
圖二十三 final\_stage 狀態

## 7. Configuration

Ap\_start 讀取到後，開始讀資料，讀完最後一筆資料後，設置 ap\_done，在設置為 ap\_idle。



圖二十四 ap\_start



圖二十五 ap\_done, ap\_idle

Github link : <https://github.com/shengxsheng/SOC-Design.git>