Lab_3

臺灣科技大學　　學號：M11202103　　姓名：陳泓宇

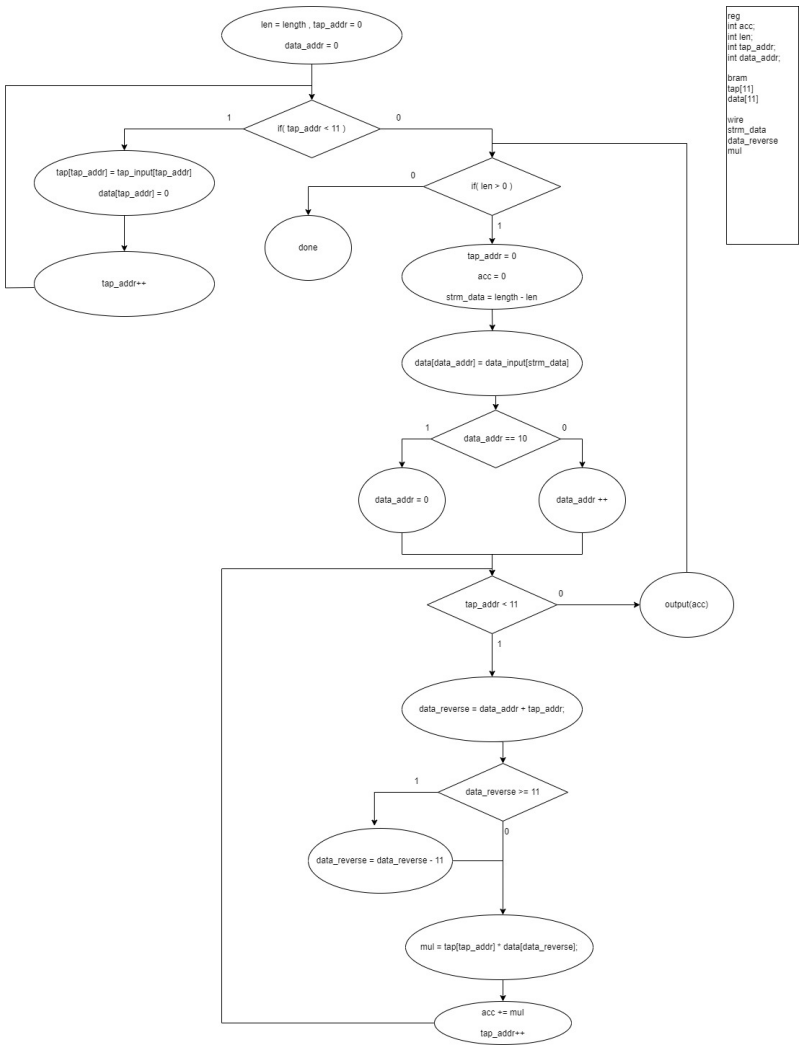
FIR(Finite impulse response)

Introduction

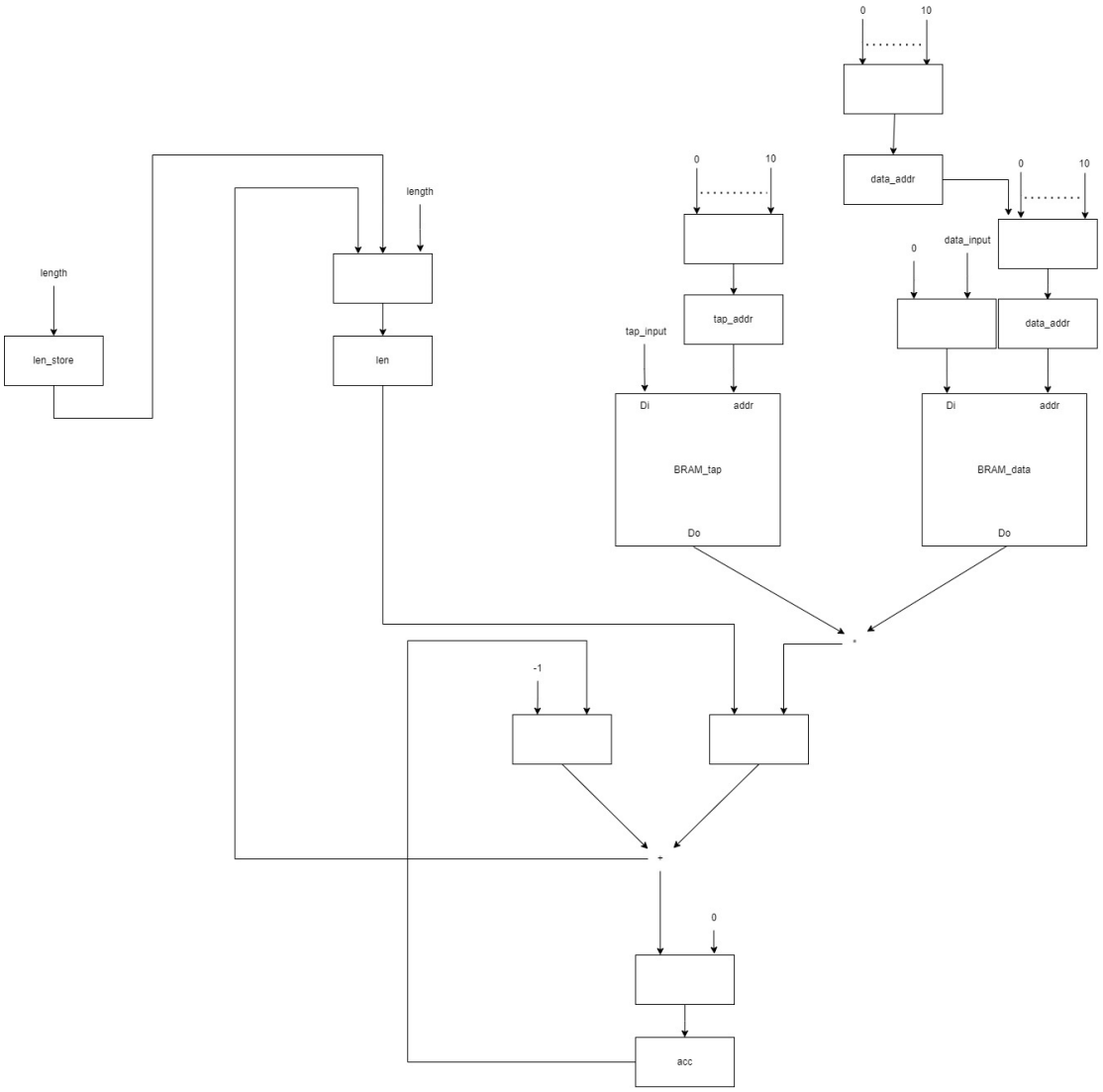
這次 Lab 的是使用 verilog 來完成 FIR 的硬體電路，其中的限制是不能夠使用移位暫存器來做資料的暫存，需使用 BRAM 來完成移位暫存器的功能，以及使用一個加法器和乘法器來完成卷積的操作。

Block Diagram

- **Datapath**



這次作業我先使用了 c 語言做功能的模擬，除了卷積功能的模擬以外還有做到 BRAM 的位址該如何選擇的部分，而不使用移位的方式來做值的讀取以及卷積，大致將功能描述出來後畫流程圖，如上圖。

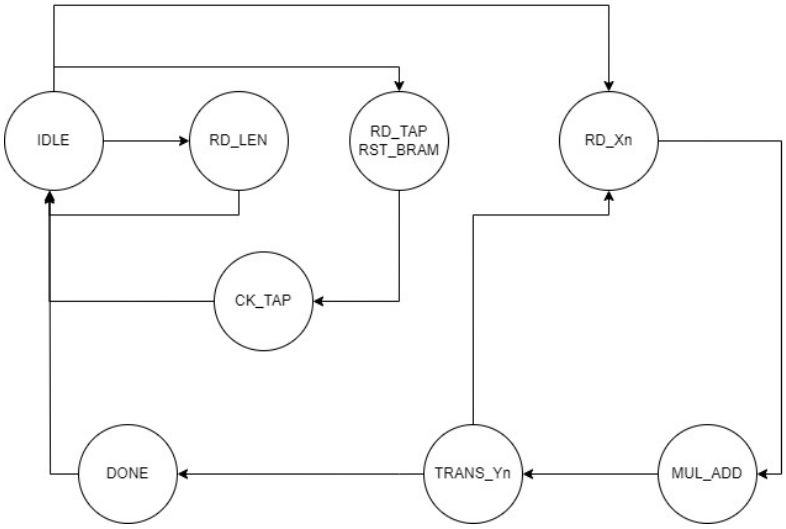


這次的作業我設計成全部只使用一個加法以及一個乘法，並用 resource sharing 的方式共用加法器的部分，而會用到加法器的有兩個部分，第一個就是卷積的部分，需要做加法，另一個是紀錄目前是第幾筆資料的暫存器，而我使用減法的方式來更新目前筆數的資料，因此我的加法是丟入-1 也就是 32'hFFFF_FFFFF 的方式來做到減一的操作。

暫存器的部分我使用了五組個暫存器，其中 tap_addr 和 data_addr 是使用了 4bits，紀錄目前的位址，以及一個 4bits 的暫存器用於紀錄這次的 data_addr 該從哪個位址開始，剩下三個 32bits 的暫存器用於紀錄目前筆數、此次輸入的 length 值、以及用於卷積計算的累加器。

除了紀錄輸入的 length 值的暫存器外，都使用 mux 來做選擇輸入的動作，因此控制訊號就是圍繞這些 mux，以及我使用了 enable 來控制是否寫入此暫存器。

- **FSM**



- **IDLE：**
  在 IDLE 中會等待 awaddr 以及 wdata 來做功能的選擇，能夠進到 RD_LEN、RD_TAP、RD_Xn，此狀態會將 ap_idle 拉起，讓 tb 知道可以進行其他操作。
- **RD_LEN：**
  讀取 length 的值，保存至暫存器，完成後回到 IDLE。
- **RD_TAP：**
  讀取 11 個 tap 的值到 tap BRAM，並同時將 data BRAM 清零，完成後進到 CK_TAP。
- **CK_TAP：**
  將讀取到 tap BRAM 的數值傳至 tb，在經由 tb 做 tap 的核對，完成後進到 IDLE 等待新的指令。
- **RD_Xn：**
  由 stream-in 讀取 data 後將 data 寫入至 data BRAM 中正確的位置，讀到資料後進到 MUL_ADD，進行卷積運算。
- **MUL_ADD：**
  開始執行 11 個週期的卷積運算，每個週期會讀出對應的 data 和 tap 做相乘，最後在將乘法的結果和累加器的結果相加後放入累加器暫存器中，完成後進到 TRANS_Yn。
- **TRANS_Yn：**
  將計算的結果經由 stream-out 送到 tb 端，並等到 tb 端回傳 ready 訊號後，若全部筆數已完成，則進入 DONE，否則就是回到 RD_Xn 繼續下一個讀取、做卷積的循環，此狀態在將要進入 DONE 狀態時會將 ap_done 拉起。
- **DONE：**
  此狀態會將 data BRAM 的值在次清空，這樣到 IDLE 時若要直接進到 RD_Xn 做讀 data，卷積的動作時結果才會正確。

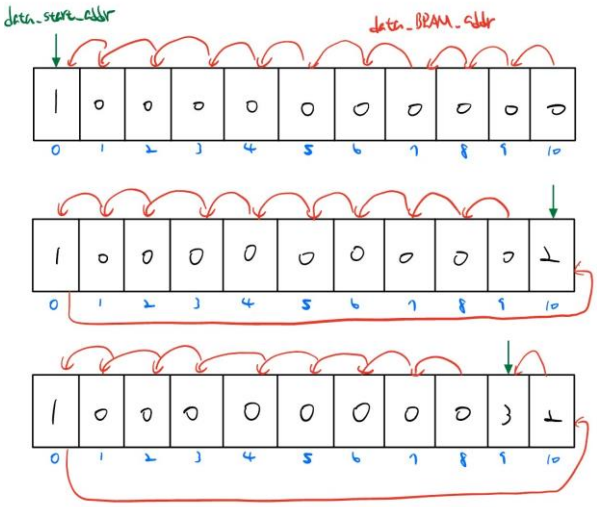
Descibe operation

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

  我使用 FSM 的方式，會有一個狀態是在讀取 tap 的狀態，持續 11 個週期，同時將 data BRAM 的值清成 0，並且我使用另一個類似 FSM 的方式去選擇 addr。再使用一個狀態是讀取 stream 資料進去 data BRAM，並且記錄每次卷積的起始位置，就能完成移位暫存器的效果。

- **How to access shiftram and tapRAM to do computation**

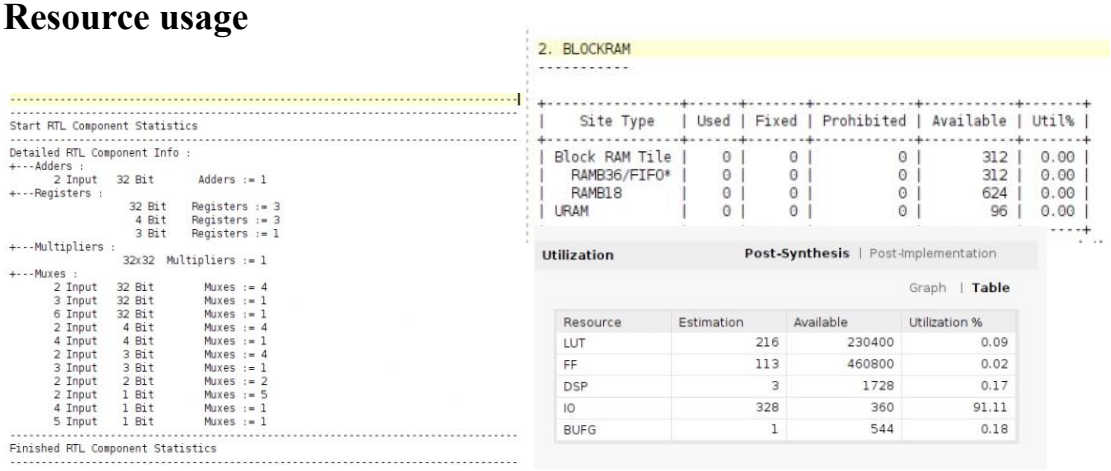
  我使用的方式是固定每次 tap 送出的位置(值)，然後選擇 data BRAM 輸出的值，而 data BRAM 都是固定的規律，所以只需要紀錄每次卷積的起始位置，就可以讓 addr 進入 FSM，然後完成 11 個週期的卷積功能。



- **How ap_done is generated**

  我有一個 DONE 狀態是在做 reset data BRAM 的動作，而一旦進到此狀態就代表傳送已經都完成，此時我的電路會回傳 ap_done 的狀態，但還沒回傳 ap_idle，ap_idle 是要回到 IDLE 狀態時才會回傳。

Resource usage



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

2. BLOCKRAM

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	312	0.00
RAMB36/FIFO*	0	0	0	312	0.00
RAMB18	0	0	0	624	0.00
URAM	0	0	0	96	0.00

Utilization Post-Synthesis | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	216	230400	0.09
FF	113	460800	0.02
DSP	3	1728	0.17
IO	328	360	91.11
BUFG	1	544	0.18

從報告中可以看出整個 FIR 只使用了一個加法器和一個乘法器，符合作業要求，而多用了兩個 32bits 暫存器的部分是為了儲存 length 的部分，若不管 length 的值，轉而去讀取 tlast 信號的話可以再少用 64bits 的暫存器，這部分是我能夠做的更好的。

Timing Report

- **maximum frequency**

```
Clock  Waveform(ns)       Period(ns)      Frequency(MHz)
-----  -------------      ----------      --------------
clk    {0.000 2.310}      4.620           216.450
```

- **Report timing slack**

Summary

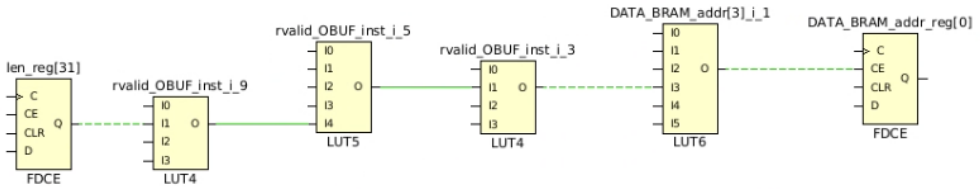
Name	Path 1
Slack	0.006ns
Source	len_reg[31]/C (rising edge-triggered cell FDCE clocked by clk {rise@0.000ns fall@2.310ns period=4.620ns})
Destination	DATA_BRAM_addr_reg[0]/CE (rising edge-triggered cell FDCE clocked by clk {rise@0.000ns fall@2.310ns period=4.620ns})
Path Group	clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	4.620ns (clk rise@4.620ns - clk rise@0.000ns)
Data Path Delay	4.232ns (logic 1.145ns (27.056%) route 3.087ns (72.944%))
Logic Levels	4 (LUT4=2 LUT5=1 LUT6=1)
Clock Path Skew	-0.145ns
Clock Un...rtainty	0.035ns

Source Clock Path

```
From Clock:  clk
  To Clock:  clk

Setup :            0  Failing Endpoints,  Worst Slack        0.006ns,  Total Violation        0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.132ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack        1.810ns,  Total Violation        0.000ns
```

- **Report timing on longest path**

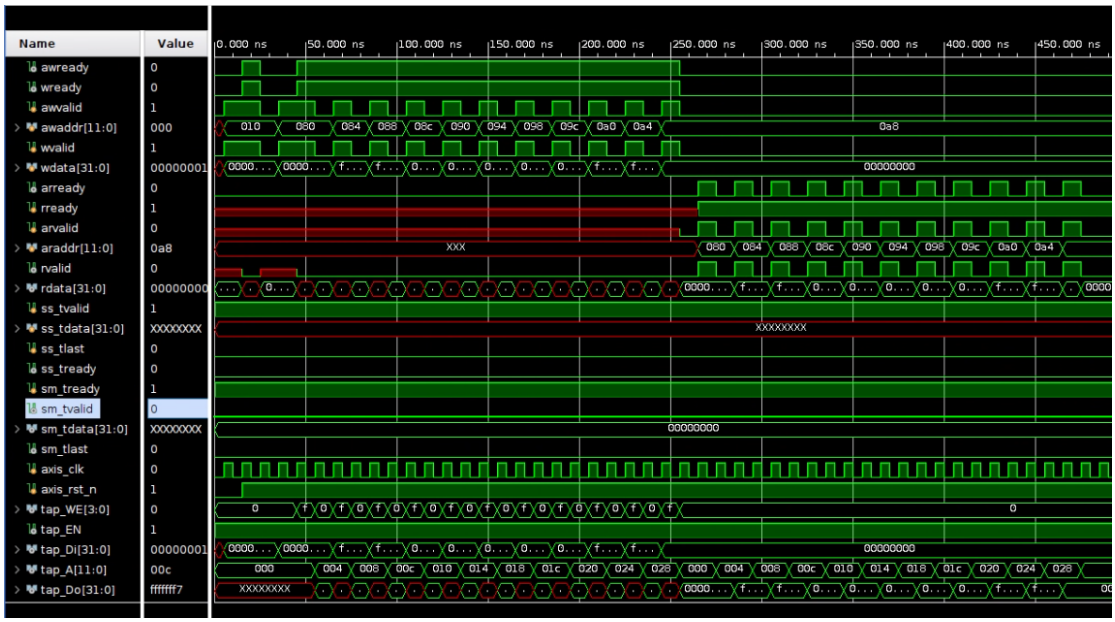



```
Max Delay Paths
--------------------------------------------------------------------------------
Slack:                    inf
  Source:                 awvalid
                            (input port)
  Destination:            awready
                            (output port)
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        6.476ns  (logic 3.978ns (61.433%)  route 2.497ns (38.567%))
  Logic Levels:           5  (IBUF=1 LUT2=1 LUT5=1 LUT6=1 OBUF=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                                                      0.000     0.000 r  awvalid (IN)
                         net (fo=0)                   0.000     0.000    awvalid
                                                                     r  awvalid_IBUF_inst/I
                         IBUF (Prop_ibuf_I_O)         0.972     0.972 r  awvalid_IBUF_inst/O
                         net (fo=5, unplaced)         0.800     1.771    awvalid_IBUF
                                                                     r  awready_OBUF_inst_i_5/I1
                         LUT2 (Prop_lut2_I1_O)        0.124     1.895 f  awready_OBUF_inst_i_5/O
                         net (fo=1, unplaced)         0.449     2.344    awready_OBUF_inst_i_5_n_0
                                                                     f  awready_OBUF_inst_i_4/I5
                         LUT6 (Prop_lut6_I5_O)        0.124     2.468 f  awready_OBUF_inst_i_4/O
                         net (fo=1, unplaced)         0.449     2.917    awready_OBUF_inst_i_4_n_0
                                                                     f  awready_OBUF_inst_i_1/I4
                         LUT5 (Prop_lut5_I4_O)        0.124     3.041 r  awready_OBUF_inst_i_1/O
                         net (fo=2, unplaced)         0.800     3.841    wready_OBUF
                                                                     r  awready_OBUF_inst/I
                         OBUF (Prop_obuf_I_O)         2.634     6.476 r  awready_OBUF_inst/O
                         net (fo=0)                   0.000     6.476    awready
                                                                     r  awready (OUT)
  -------------------------------------------------------------------    -------------------
```

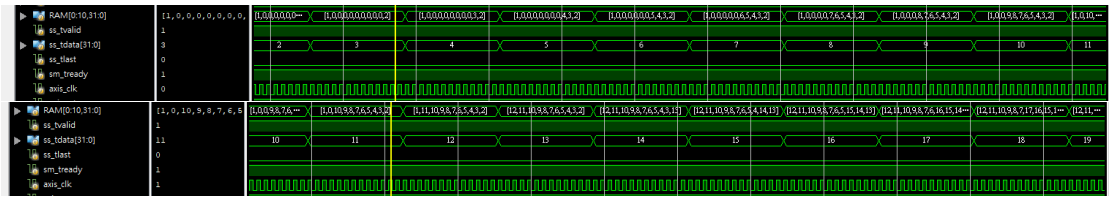
Simulation Waveform, show

- **Coefficient program, and read back**



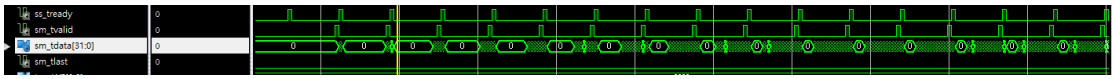
最前面 awaddr 0x10 的部分是在寫入 length，而後面 0x80 開始則是寫入 tap 的值，而寫完 11 個 tap 後會作驗證 tap 是否正確的操作，也就是 aready 以下的訊號在將 tap 回傳並讓 tb 確認值是否正確。

- **Data-in stream-in**

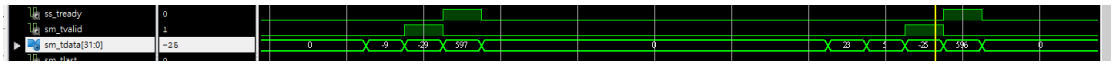


由這兩張圖片可以看出經由 stream-in 傳入的 data 被送入 data RAM 中。

- **Data-out stream-out**



由上圖可以看出資料傳遞。



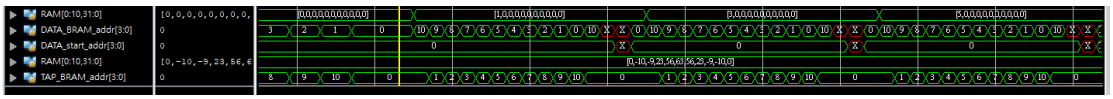
由上圖可以看出傳遞了-29 和-25 到 tb 端。

```
[PASS] [Pattern   1] Golden answer:   -10, Your answer:   -10
[PASS] [Pattern   2] Golden answer:   -29, Your answer:   -29
[PASS] [Pattern   3] Golden answer:   -25, Your answer:   -25
[PASS] [Pattern   4] Golden answer:    35, Your answer:    35
```

由上圖可以看出 tb 端在收到資料後與 golden data 做核對後正確。

- **RAM access control**
  - **Data BRAM**

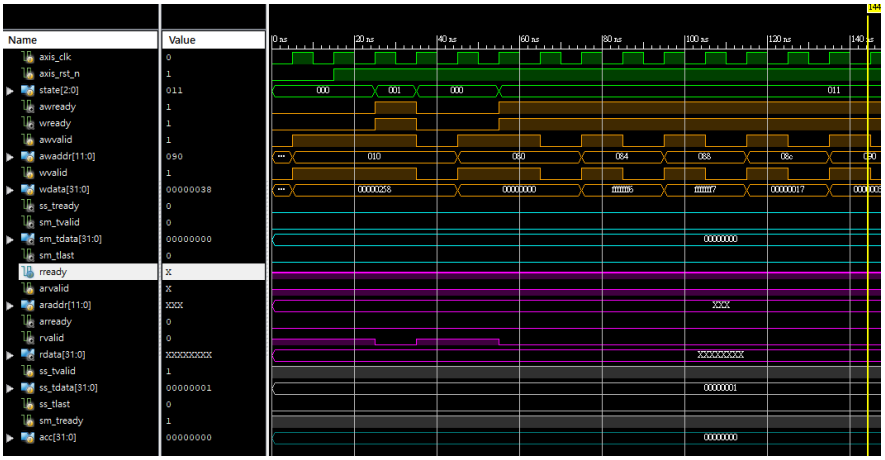
  - **tapBRAM**



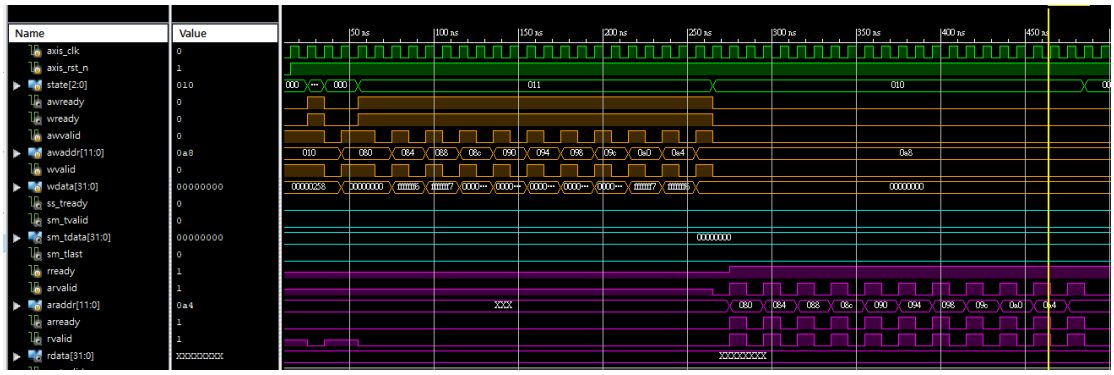
由上面兩張圖可以看出只有 data BRAM 的 addr 的起始位置不同，tap BRAM 的 addr 一直維持由零到十。

- **FSM**

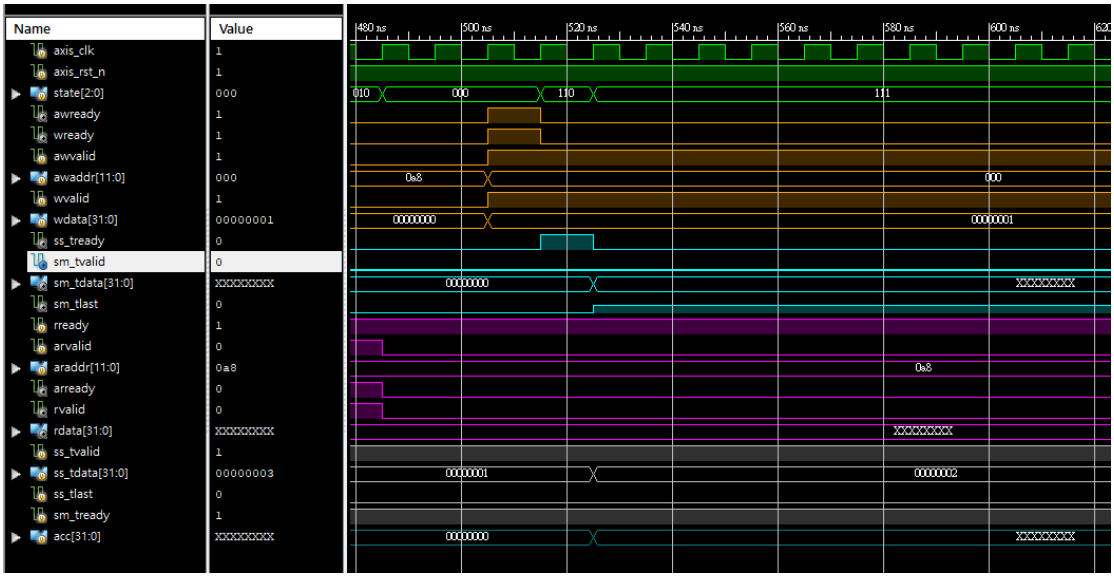
IDLE = 3'b000, RD_LEN = 3'b001, RD_TAP = 3'b011, CK_TAP = 3'b010, RD_Xn = 3'b110, MUL_ADD = 3'b111, TRANS_Yn = 3'b101, DONE = 3'b100;



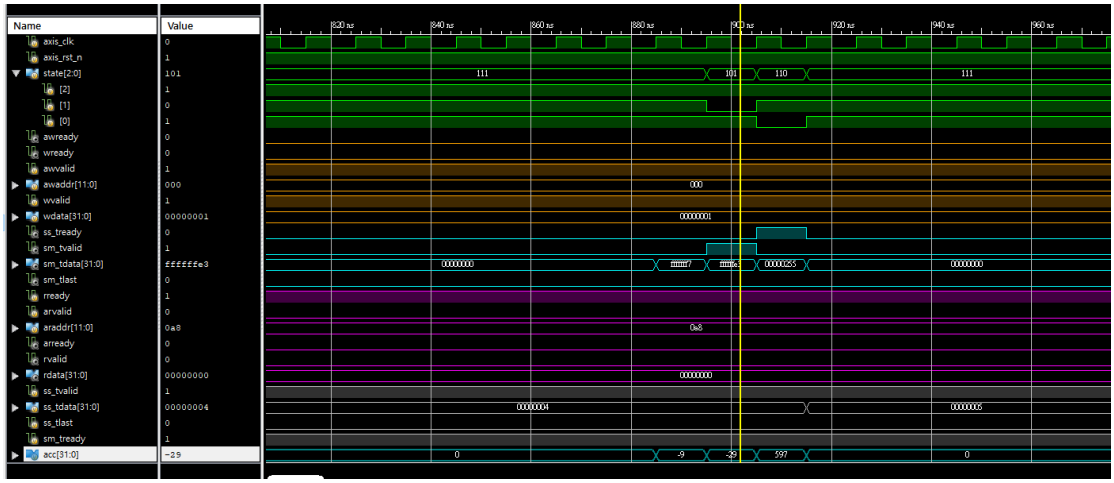
Reset 後會先進到 IDLE 的狀態，IDLE 時讀到 addr = 0x10 且 valid 進到 RD_LEN，讀取完 length 後回到 IDLE，IDLE 時讀到 addr = 0x80 且 valid 進到 RD_TAP，讀取 11 個 tap。



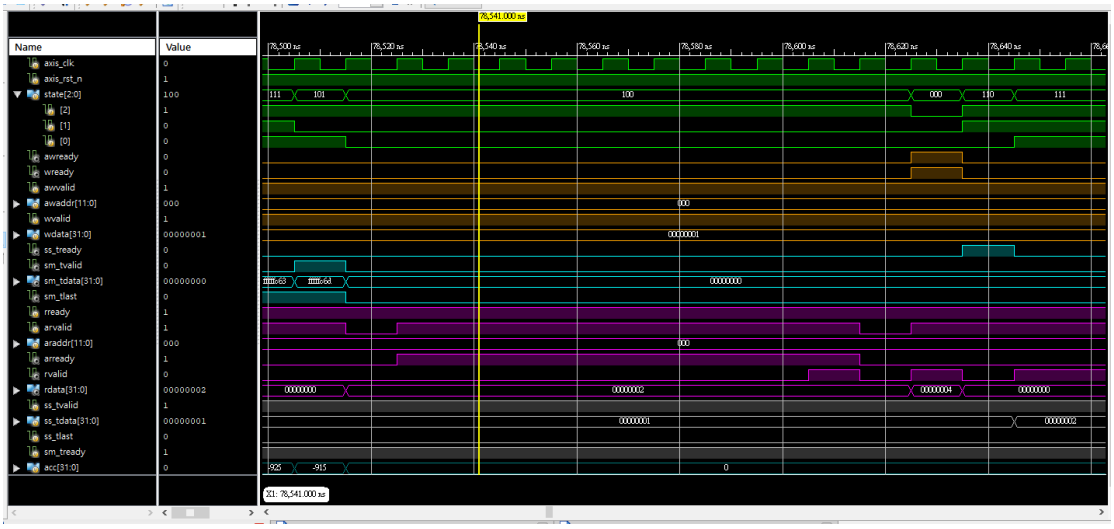
讀取完 tap 後進到 CK_TAP，將讀到 BRAM 的 tap 回傳給 tb，如上圖紫色部分。



CK_TAP 完成後回到 IDLE，收到 ap_start 的訊號後進到 RD_Xn，讀取 ss_tdata 並回傳 ready 後進到 MUL_ADD。



MUL_ADD 卷積結束後進到 TRANS_Y，將結果回傳到 tb，當 tb 接收到之後回傳 ready 後會回到 RD_Xn，如此往復 length 的筆數。



結束所有運算後會進到 DONE，將 data BRAM 的值清零，並將 ap_done 拉起，直到回到 IDLE 後再將 ap_idle 拉起，而 tb 看到 ap_idle assert 後會傳第二次的 ap_start，開始做第二、三次計算。