

SoC Lab3

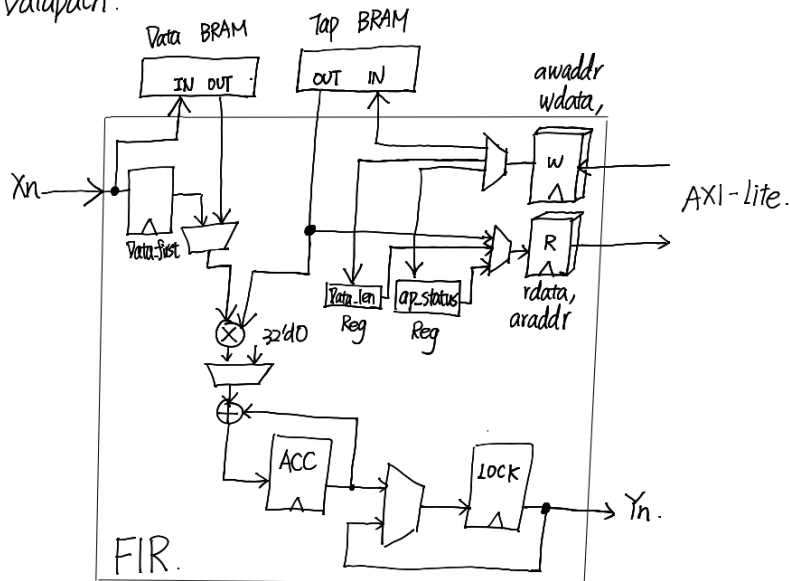
學號:111061605

姓名:周聖平

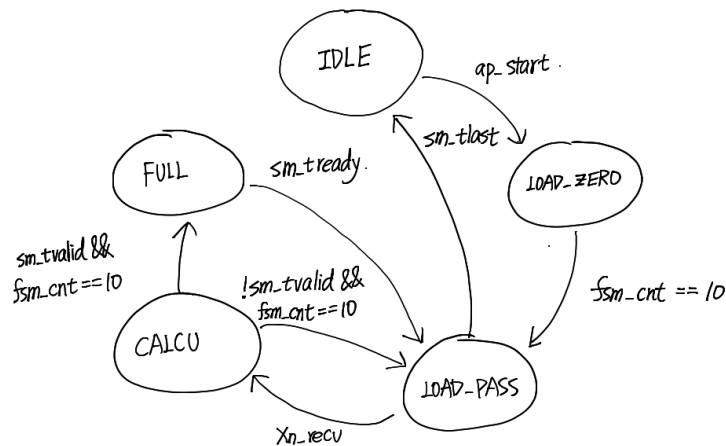
一、Block Diagram

• Datapath

Datapath.



• Control Signal



IDLE: 閒置狀態。

LOAD_ZERO: 負責將 DataRAM 初始化為 0。

LOAD_PASS: 等待 Xn 輸入，並將運算完的 Yn 傳送到下一級的 Reg 等待輸出。

CALCU: 負責運算 Yn。

FULL: 當下一級的 Reg 存有前一筆的 Yn 時(sm_tvalid=1)，就會跳到 FULL 狀態。此時，Adder 的輸入將會被切換成 32'd0，ACC Reg 會去鎖住當前運算的結果，並等到下一級的 Reg (Yn)輸出成功。

二、Describe Operation

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

當 Data-In 的 Address、Data 一進來後，就會有各自的 register 去負責暫存起來，目的是為了防止接收方無法及時處理，而只有當 Address、Data 都接收到資料後，才會寫入相對應的位址。寫入後，就會重新等待新一輪的資料。

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

DataRAM

當有新的 X_n 接收到後，就會寫入 DataRAM 中，同一時間新的 X_n 也會儲存到一個 register，目的是為了能快一個 clock 得到新的 X_n 。

而接收完 X_n 後，下一個 clock 就會進入運算模式開始運算第一次的累加，此時，第一筆的 X_n 就會從 register 拉，因為如果要從 RAM 拿的話，需要再多等一個 clock。此外，在算第一筆累加時，會同時去向 DataRAM 要第二筆的 X_n ，下個 clock 就能計算第二筆的累加。

接下來，就依序地去拿資料做累加運算。

TapRAM

當有新 X_n 進來後，就要先向 TapRAM 拿第一筆的 coef，因為 RAM 需要等待一個 clock 才会有資料輸出，所以需要提前拿資料才能來的及跟 X_n 做運算。接下來，就跟 DataRAM 一樣依序地去拿資料做累加運算。

- **How to transfer Y_n .**

在 Y_n 的部分，會有一個 register 負責去儲存完成的 Y_n 等待著傳送出去，而在等待的同時，下一筆 Y_n 就能開始運算，這樣就不會因為 Y_n 沒有接收而卡在那邊。

- **How ap_start is generated.**

H: 只有在 $ap_idle=1$ 時，才能夠透過 AXI 修改 ap_start 的值成 H。

L: 當 $ap_start=H$ 時， ap_start 將會被拉為 L。

- **How ap_done is generated.**

H: 當最後一筆 Y_n 運算完畢後，就會將 ap_done 拉為 H

L: 當 ap_done 為 H 並且 ap_done 被讀取後， ap_done 將會被拉為 L。

- **How ap_idle is generated.**

H: 當 ap_done 為 H 並且 ap_done 被讀取後， ap_idle 將會被拉為 H。

L: 當 ap_start 被修改為 H 時， ap_idle 將會被拉為 L。

三、 Resource Usage

LUT: 336、FF: 240

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	336	0	0	53200	0.63
LUT as Logic	336	0	0	53200	0.63
LUT as Memory	0	0	0	17400	0.00
Slice Registers	240	0	0	106400	0.23
Register as Flip Flop	240	0	0	106400	0.23
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: 0

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

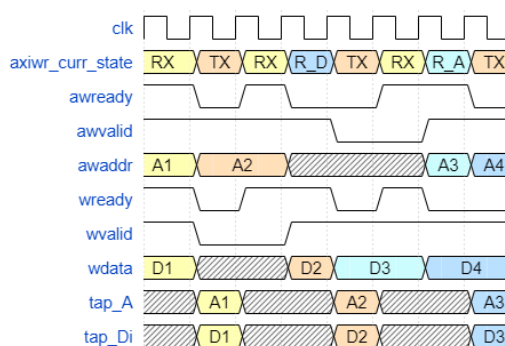
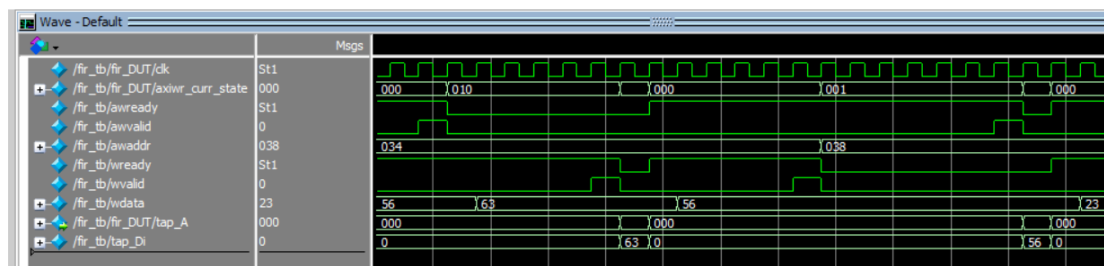
四、 Timing Report

Longest Path: 12.912ns、Slack:0.983ns@clock period=14ns

Max Delay Paths				
Slack (MET) : 0.983ns (required time - arrival time)				
Source: curr_state_reg[0]/C				
(rising edge-triggered cell FDCE clocked by axis_clk (rise@0.000ns fall@5.000ns period=14.000ns))				
Destination: Yn_acc_reg[29]/D				
(rising edge-triggered cell FDCE clocked by axis_clk (rise@0.000ns fall@5.000ns period=14.000ns))				
Path Group: axis_clk				
Path Type: Setup (Max at Slow Process Corner)				
Requirement: 14.000ns (axis_clk rise@14.000ns - axis_clk rise@0.000ns)				
Data Path Delay: 12.912ns (logic 8.748ns (67.749%) route 4.164ns (32.251%))				
Logic Levels: 12 (CARRY4=5 DSP48E1=2 LUT2=1 LUT3=2 LUT4=1 LUT5=1)				
Clock Path Skew: -0.145ns (DCD - SCD + CPR)				
Destination Clock Delay (DCD): 2.128ns = (16.128 - 14.000)				
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)

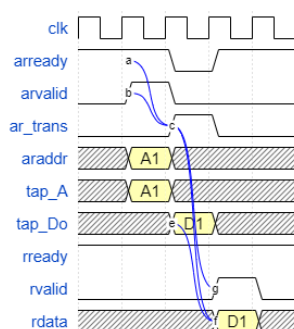
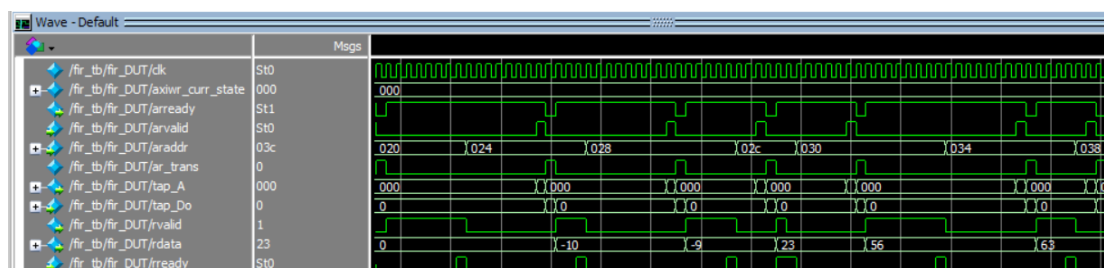
五、Simulation Waveform

- Coefficient program, and read back



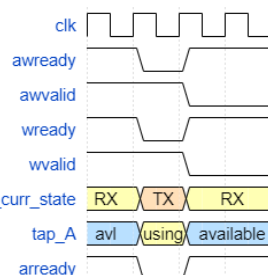
AXI WR 的部分共有三種接收狀態 第一種是 RX 它能夠同時去接收 addr、data，當同時收到 addr、data 時，就會跳到 TX 將資料寫入。

而若只收到其中一項時，就會跳到接收另一項的狀態，例如：接收到 addr -> 跳到 RECV_DATA(接收 data 狀態)，直到收齊 addr、data 時才會跳到 TX 將資料寫入。

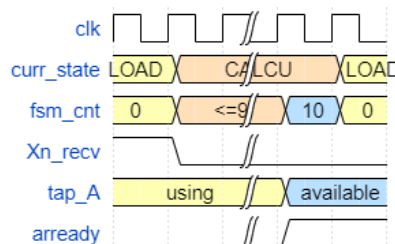


當 arready && arvalid 時，會同時去寫入 tap_A，而當 tap 輸出 data 後，會用 reg 去暫存起來，是因為不能確保 rready 什麼時候會為 H，所以 rdata 需要使用 reg 去保存起來，這樣就能釋放對 TapRAM 的控制

而由於 TapRAM 是共用的，因此 arready 在某些情況下需要被關閉的，第一 ar_trans 為 H、第二 AXIWR 占用、第三則是 FIR 占用，在這三種情況下，AXIRD 都不能去接收新的 Request。

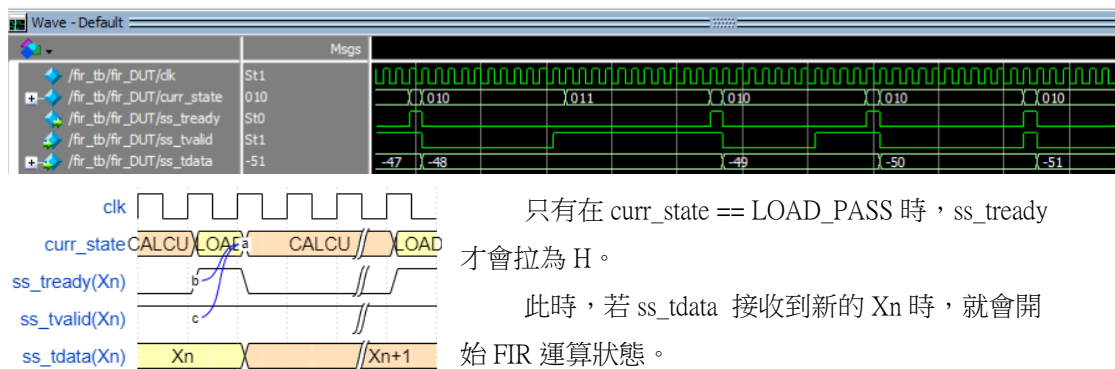


AXIWR 占用

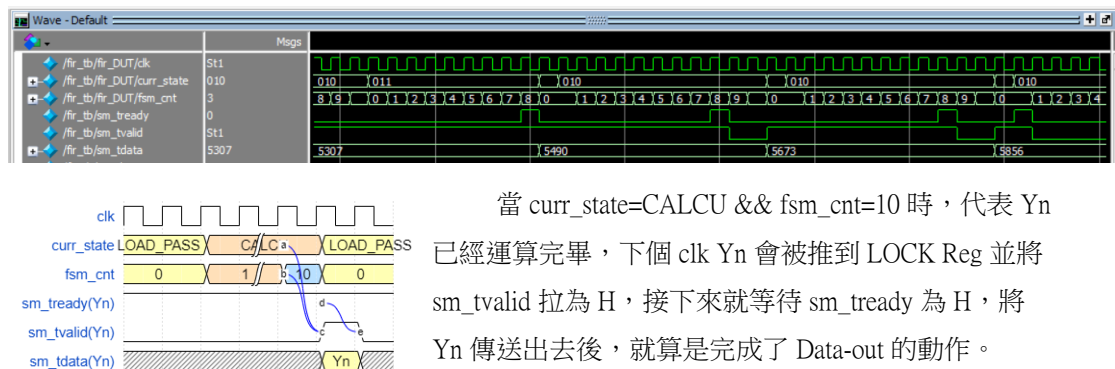


FIR 占用

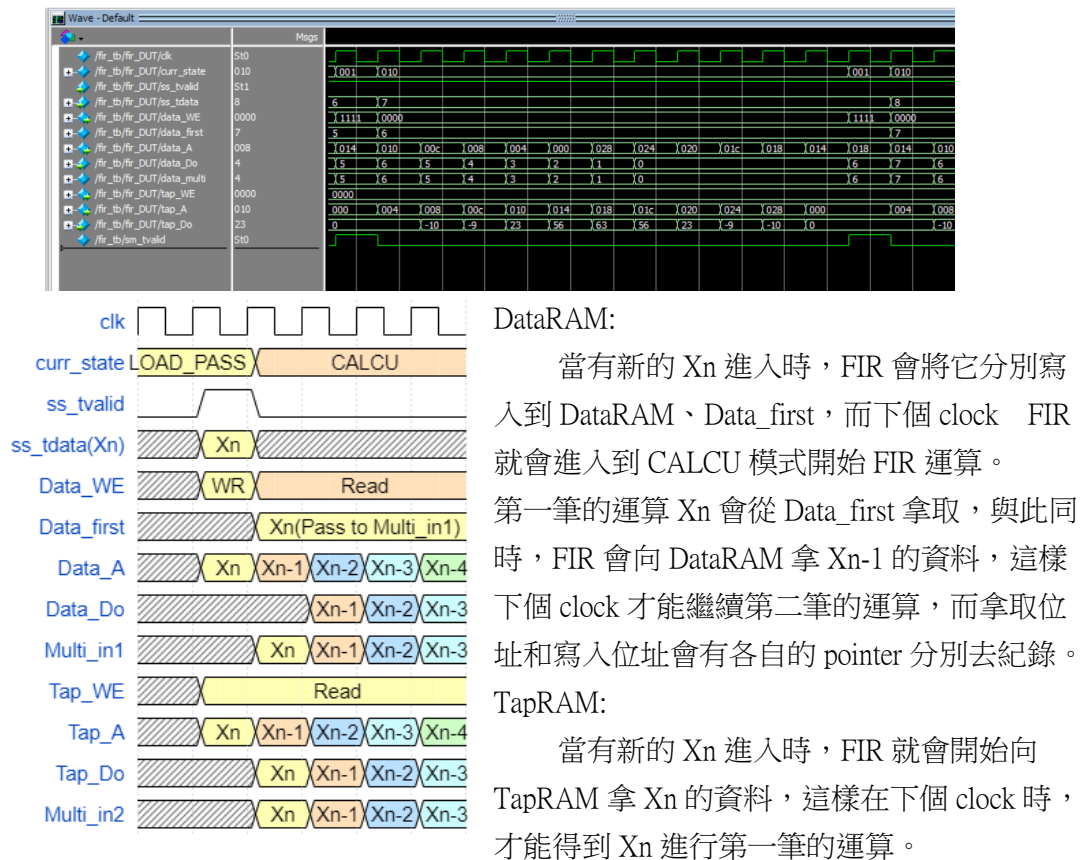
- Data-in stream-in



- Data-out stream-out

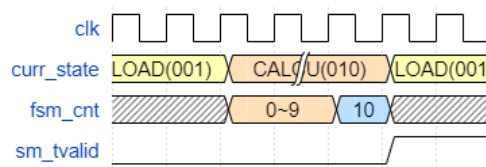
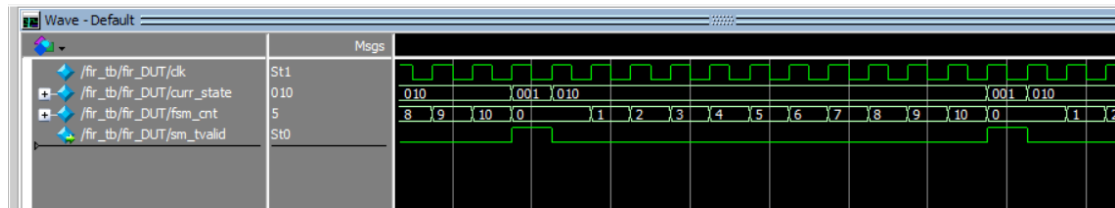


- RAM access control



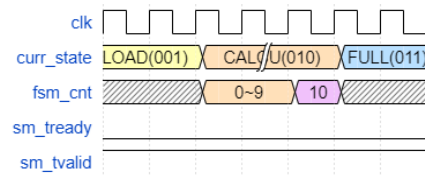
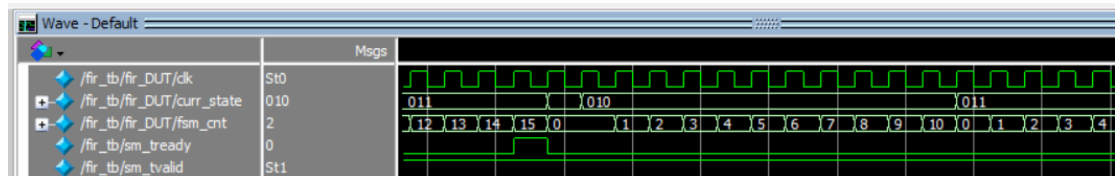
- FSM

CALCU(010) -> LOAD_PASS(001)



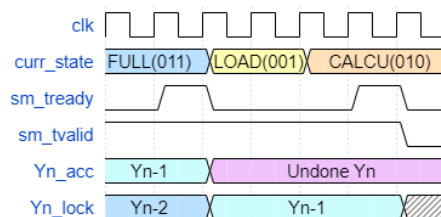
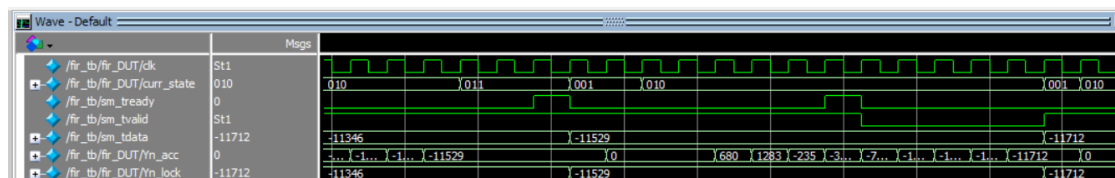
若 sm_tvalid 為 L，代表儲存 Y_n 的 LOCK Reg 是空的，當 FIR 完成計算後，就能將 Y_n 推入 LOCK Reg 等待傳輸，而接下來，就能開始下一筆的計算。

CALCU(010) -> FULL(011)



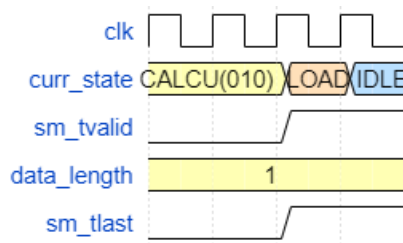
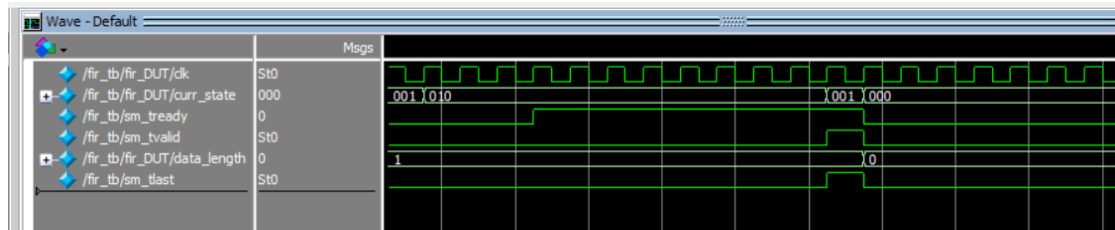
若 sm_tvalid=1 && sm_tready=0，代表前一筆 Y_n 還尚未輸出出去，如果此時 FIR 的計算完成，則需要等待前筆的 Y_n 輸出後，才能開始下筆的計算。

FULL(011) -> LOAD_PASS(001)



當 curr_state=FULL 時，會去等待 Y_n 接收，Y_n 接收完畢後，Y_n_acc 就會將上一筆計算完成的 Y_n 推到 Y_n_lock 等待接收，並且同時新的計算。

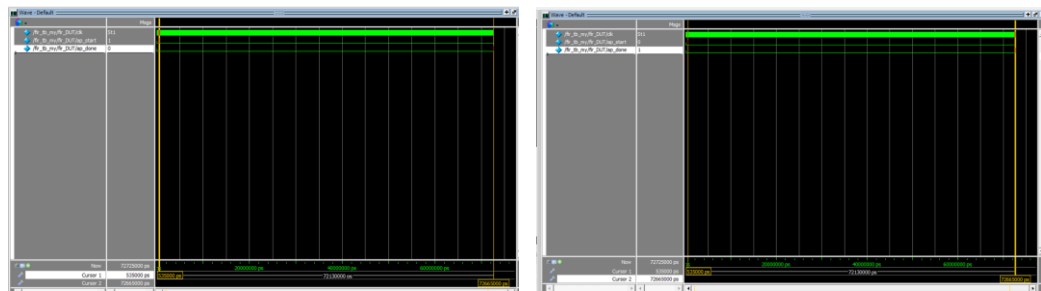
LOAD_PASS(001) -> IDLE(000)



由於每次 Y_n 傳輸出去後，data_length 就會減一，因此 data_length 為 1 時，代表著該筆 Y_n 為最後一筆，接下來就會跳至 IDLE 狀態，等待新的 FIR 運算。

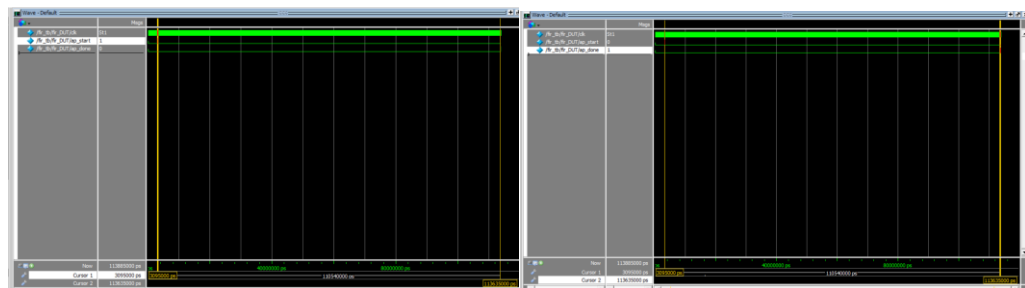
- ap_start, ap_done (measure # of clock cycles from ap_start to ap_done)

Testbench (from course)



7213 clocks

Testbench (modify from course)



11054 clocks