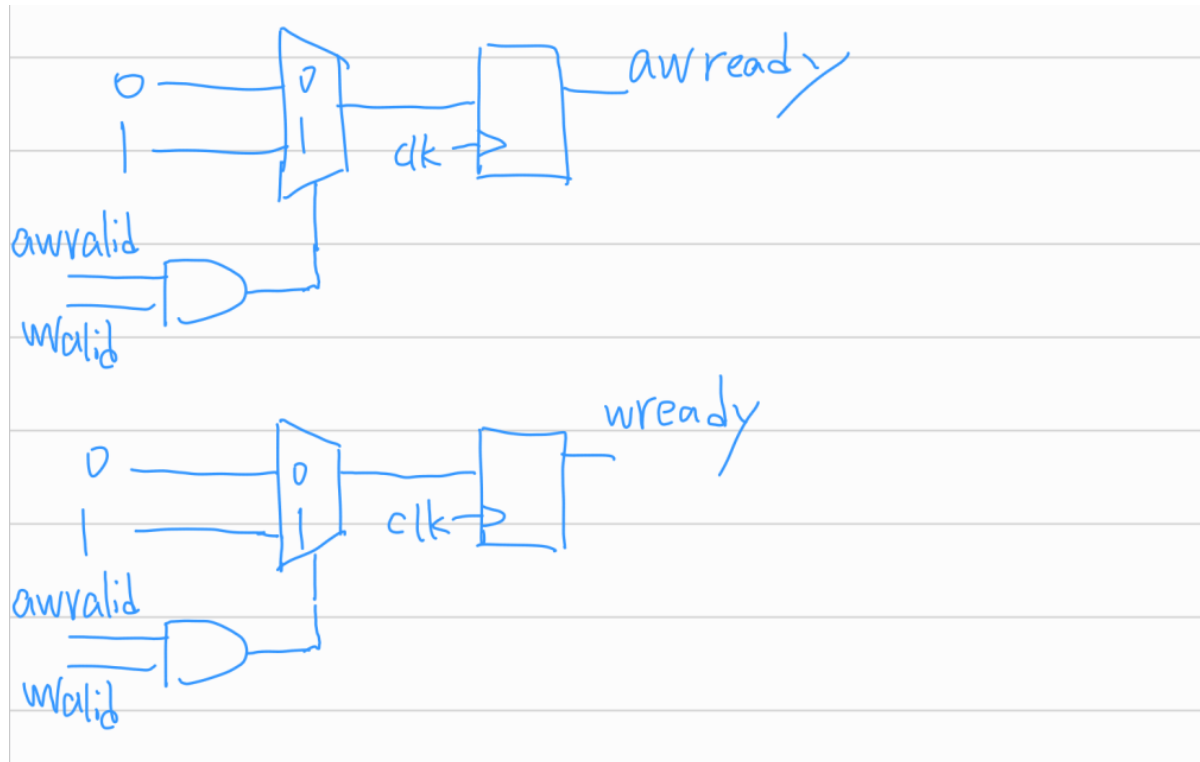


Lab 3 report

B09901196 黃秉璿

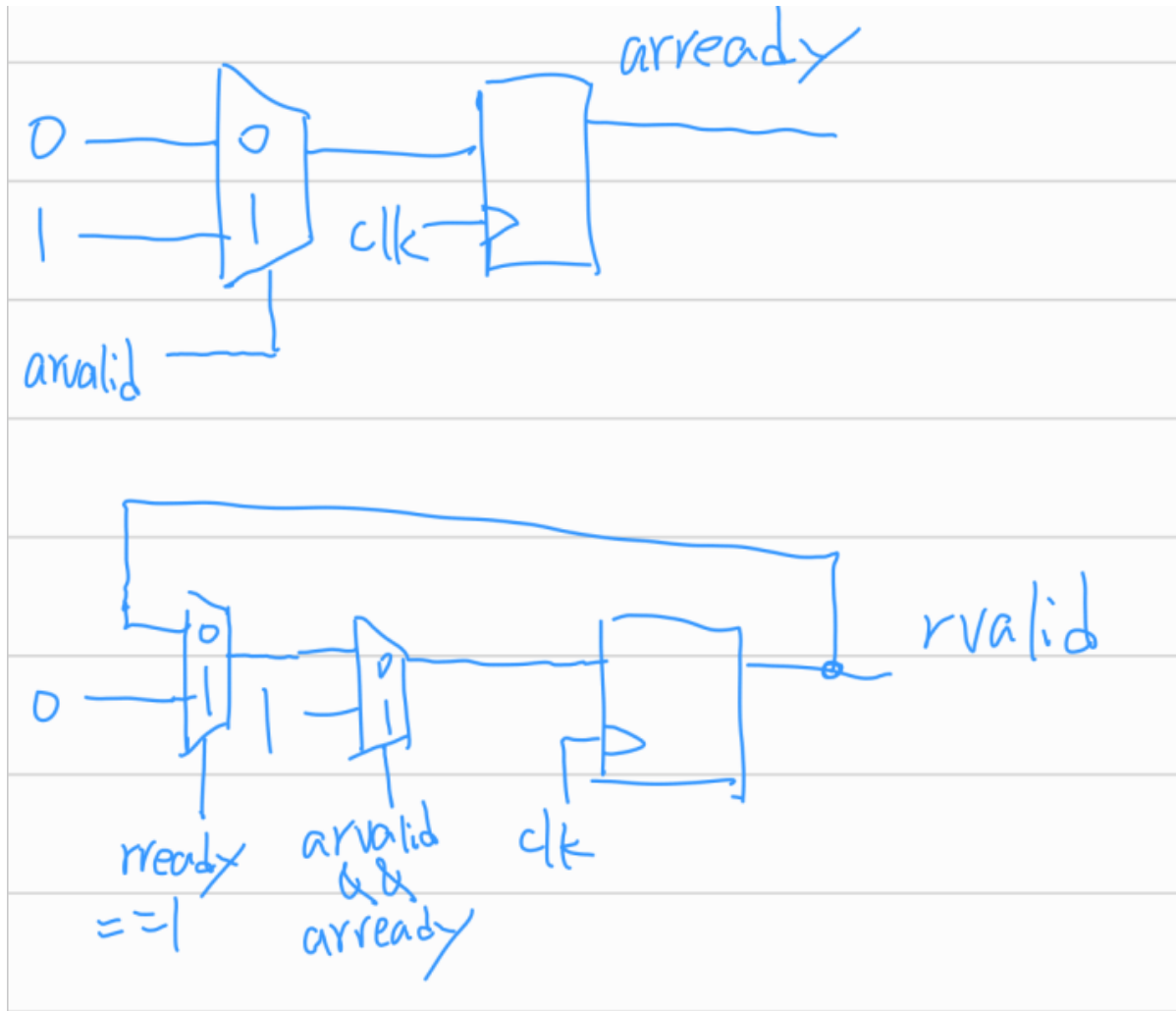
Block Diagram

AXILITE write



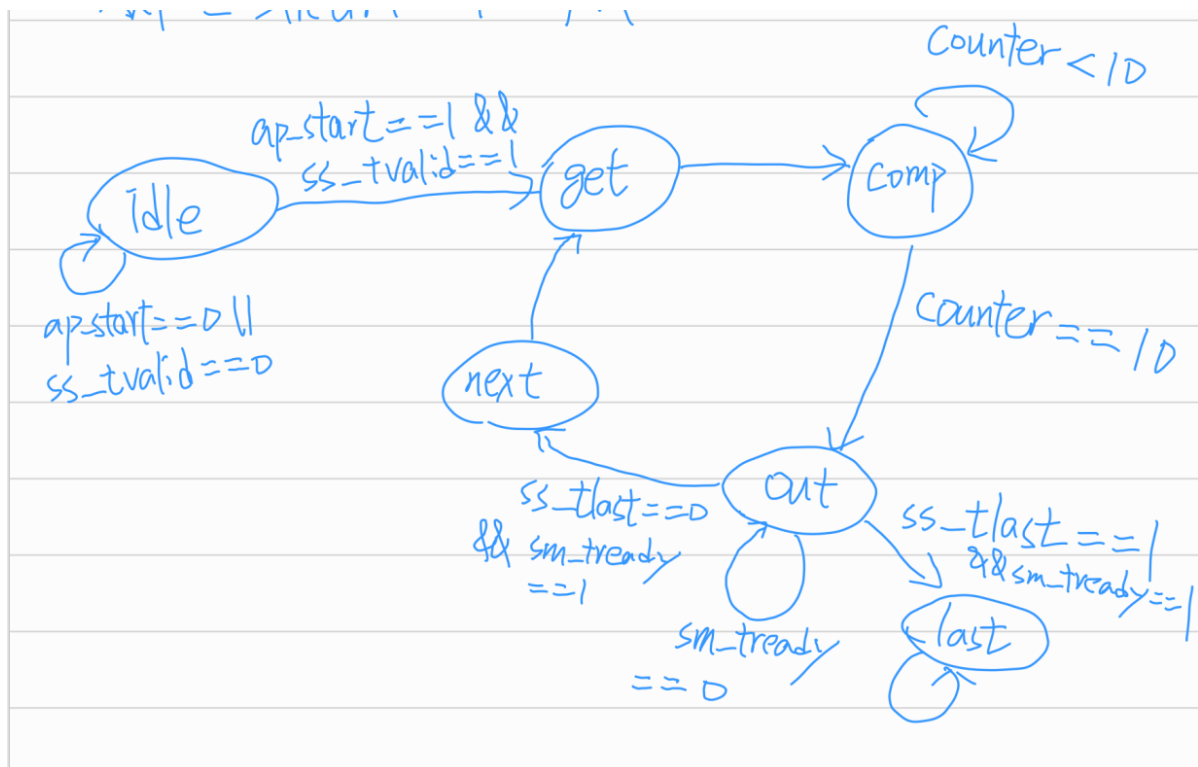
因為valid 訊號一旦升高就要等到 ready 才能降低，因此可以等到地址跟資料都是valid時再一次寫入就好，這樣就不必存地址或資料

AXILITE read



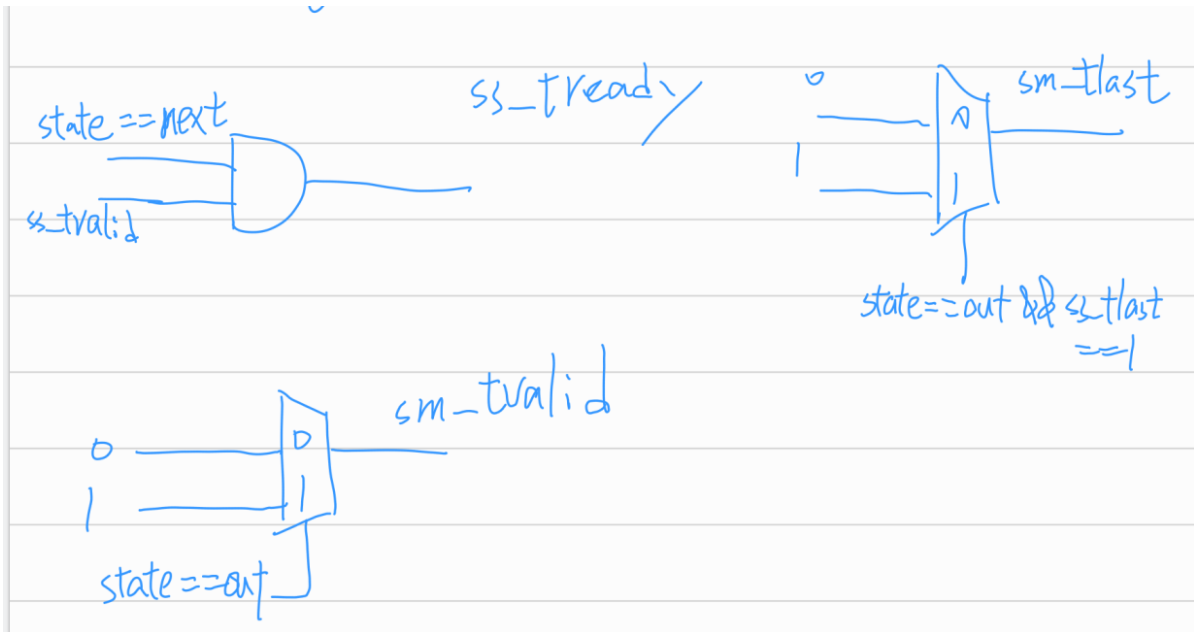
讀取也是類似，不過 `rvalid` 必須等 `arvalid` 和 `arready` 都升高才能升高，之後還要等 `rready` 升高才能降低，因此需要用 flip-flop 去存讀好的資料，等待 `rready`。

AXI stream FSM



1. idle : 初始狀態
2. get : 寫入新的 data 以及讀取 BRAM 數值
3. comp : 計算 fir 的輸出值 (tap 地址從 0x0 到 0xa , data 地址逐次減一)
4. out : 輸出 fir 數值 (等待 `ss_tready`)
5. next : 輸出 `ss_tready` 取得下一筆資料, 以及調整 data 輸入地址使得在 get 寫入時剛好取代舊資料
6. last : 所有資料都處理結束

AXI stream control signals

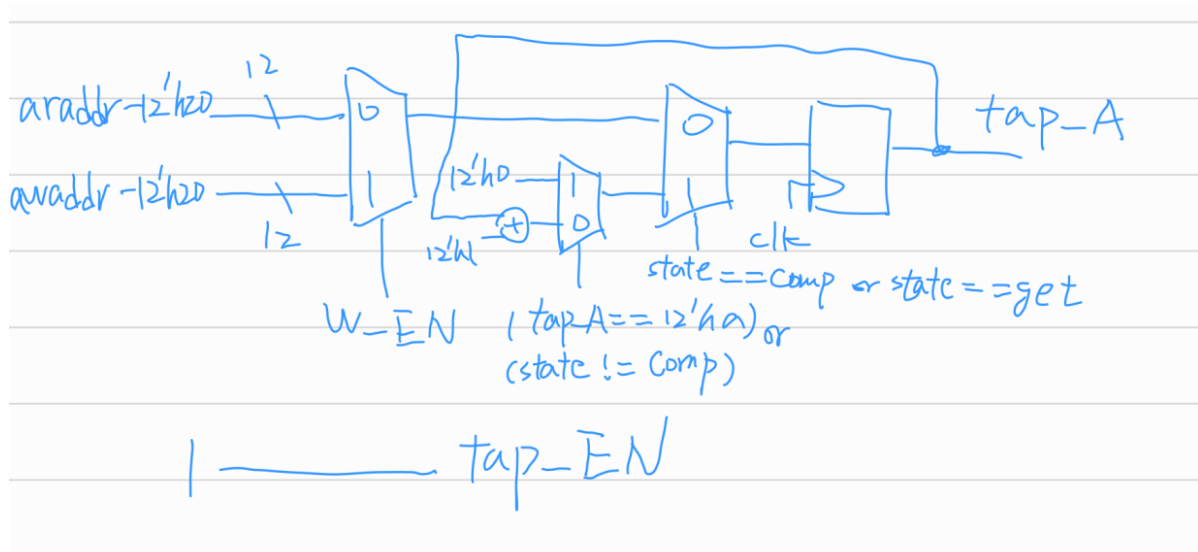
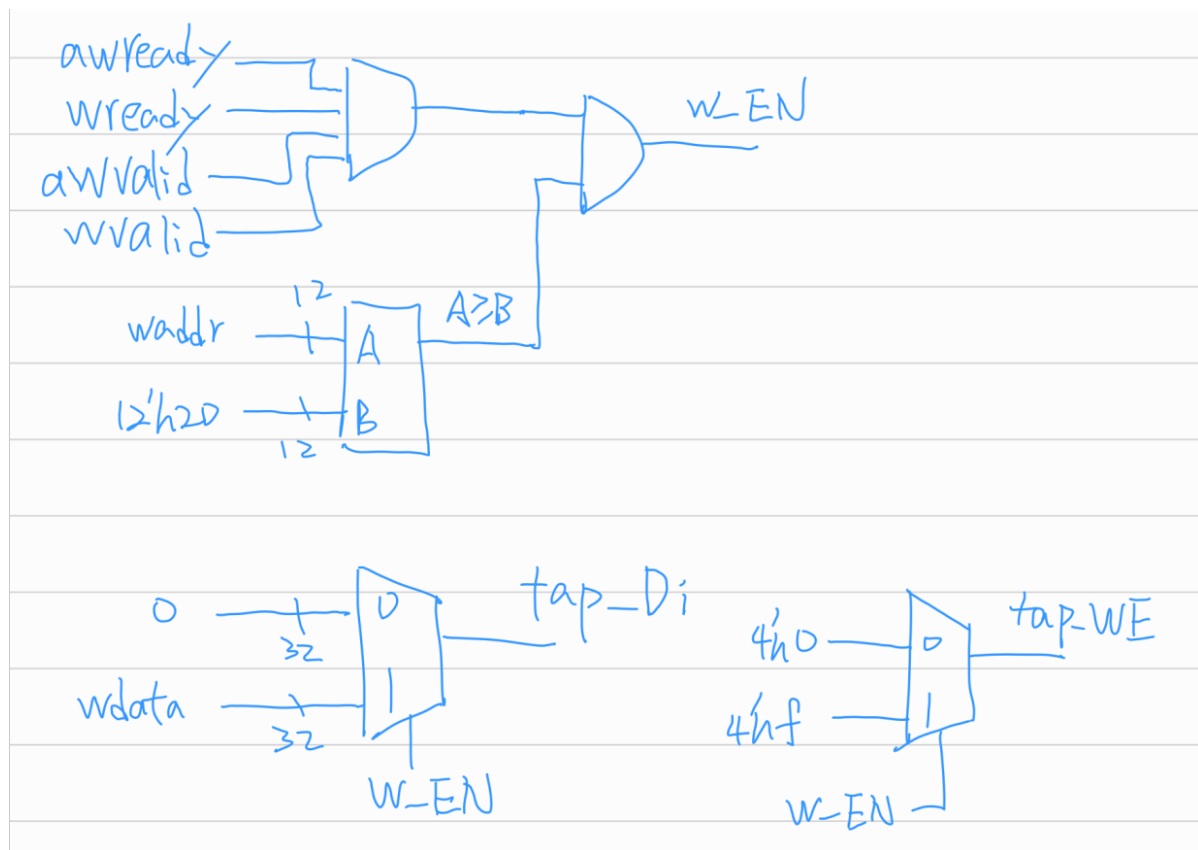


`ss_tready` 在 next 狀態下升高, 表示已經取得現在的輸入了

`sm_tvalid` 在 out 狀態下升高, 表示資料計算完成

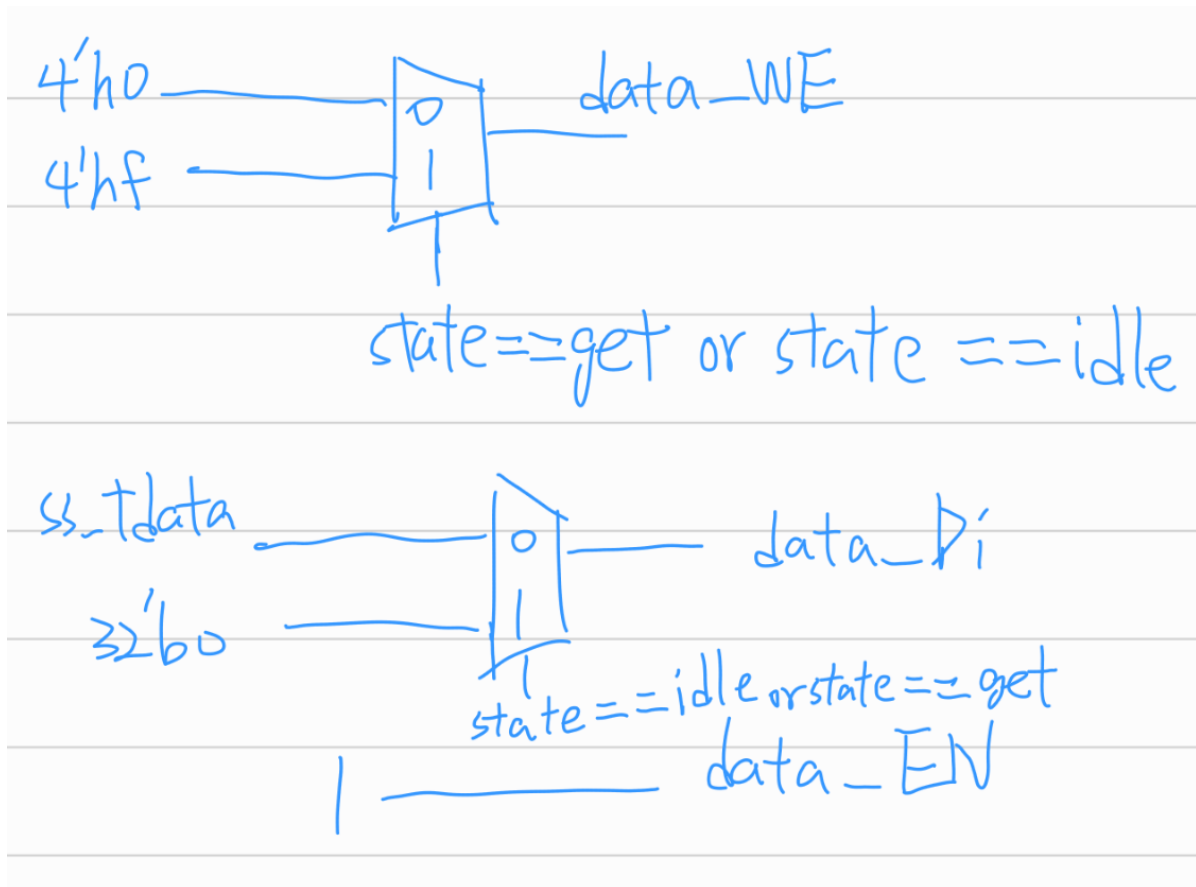
`sm_tlast` 在 `ss_tlast` 升高後的 out 狀態升高, 表示所有資料都算完了

BRAM tap read/write



tap 的部分只用來存 fir 的係數，因此要確認地址是不是要寫到係數 ($\geq 0x20$)，再加上 axilite 的訊號作為寫入判斷訊號 **W_EN**，當 **W_EN = 1** 時為寫入，其他時候為讀取

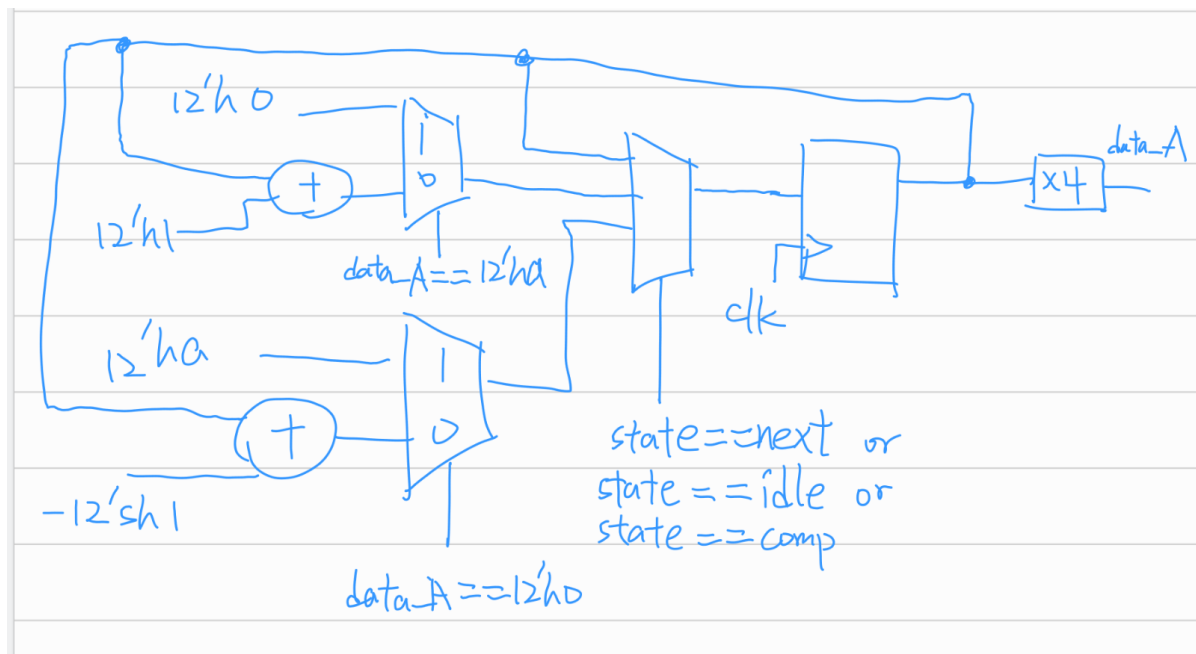
Bram data read/write



`data_WE` 在 idle 狀態會打開做初始化，get 狀態時也會打開，寫入 stream 輸入

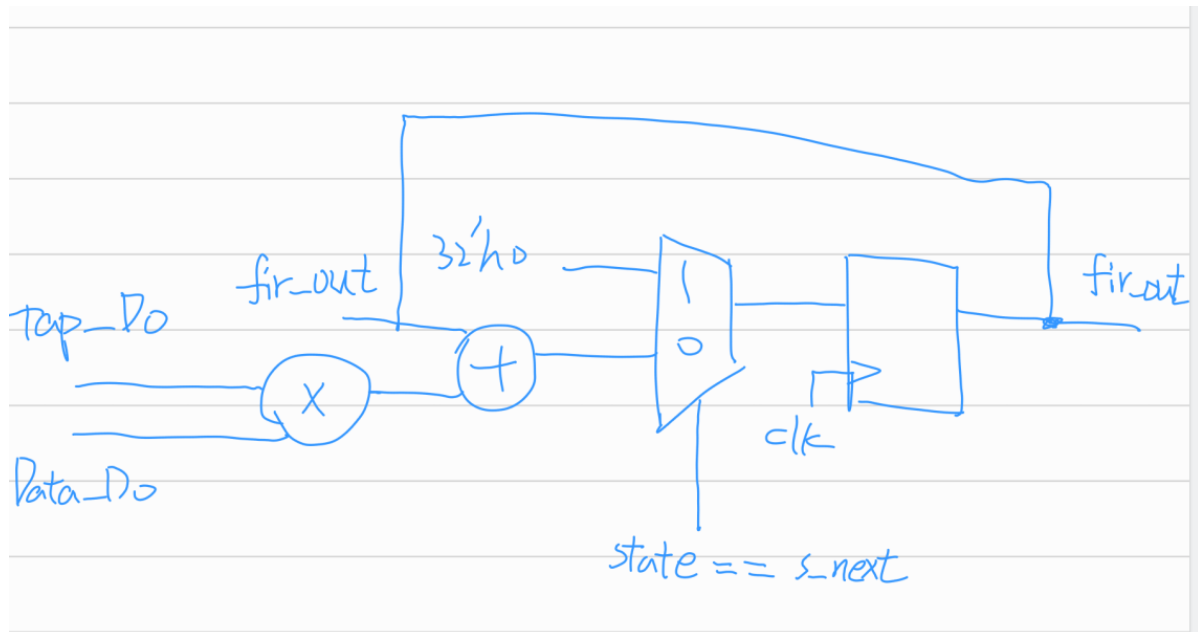
`data_Di` 在初始化時寫0，其餘則為 stream 輸入

`data_EN` 則是一直保持開啓



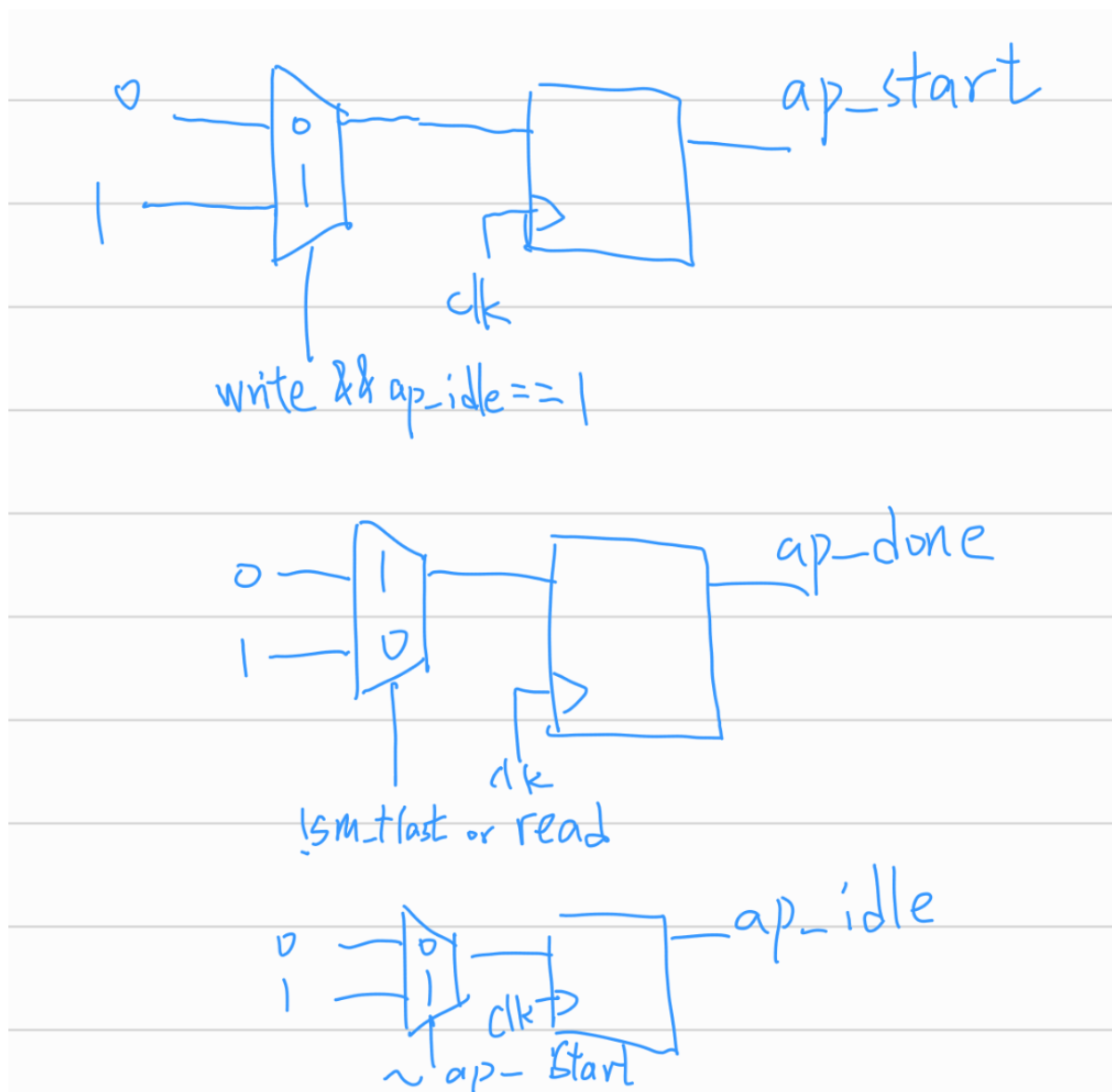
`data_A` 在 idle 時會以 $(addr + 1) \% 11$ 的方式寫入 0，之後在 stream 開啓後以 $(addr = (addr == 0) ? 10 : addr - 1)$ 的方式取讀取，達到 shift register 的效果，之後在 next 狀態會地址會+1，覆蓋最舊的輸入。

FIR



利用控制Bram輸入地址的方式，fir 只需要做累乘運算即可，當要進入下一輪運算時清空

ap_start, ap_done, ap_idle



參考 address map 我把三個訊號包成 `ap_config = {ap_idle, ap_done, ap_start}` 一開始 reset 時，只有 `idle = 1` (`ap_config = 100`)，等到 start 被寫入會有一個 cycle 變成 `001` 之後變成 `000`，完成運算後才會變為 `110` 表示 idle 以及 done

(註：因為 test bench 要測試 idle done 有沒有等於1，所以我把 ap_done 讀取後設為0註解掉了)

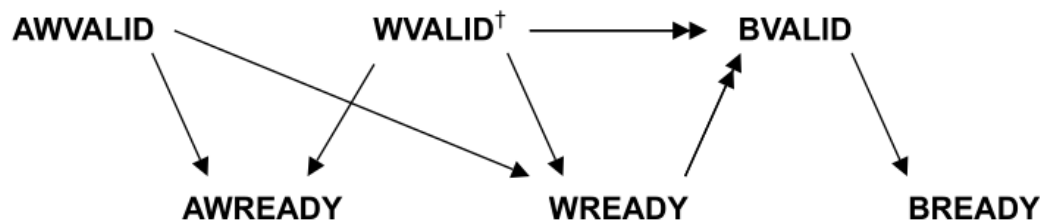
Describe operation

axi read write 的部分我參考了規格書的 dependency graph

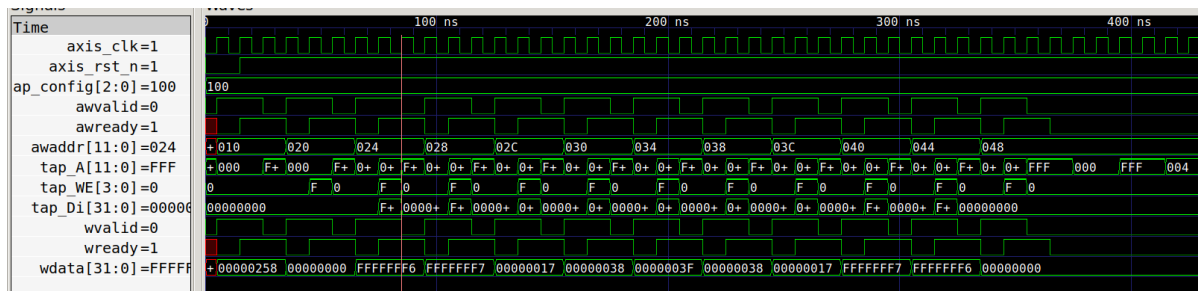
In the dependency diagrams:

- single-headed arrows point to signals that can be asserted before or after the signal at the start of the arrow
- double-headed arrows point to signals that must be asserted only after assertion of the signal at the start of the arrow.

AXILITE write



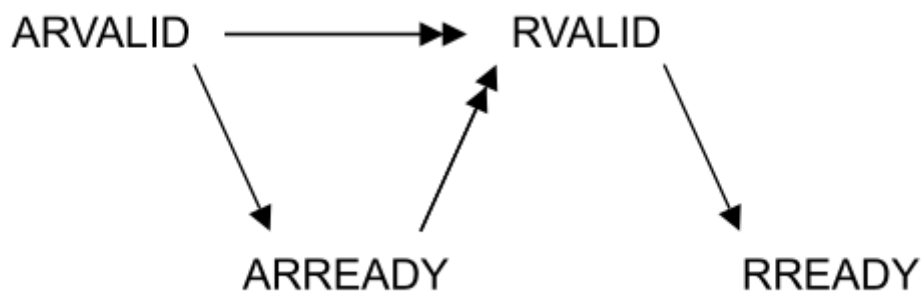
† Dependencies on the assertion of **WVALID** also require the assertion of **WLAST**

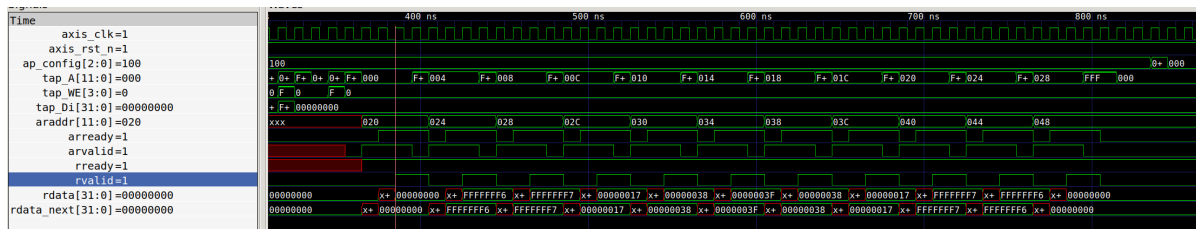


可以看到 **awready** 以及 **wready** 都等到 **awvalid** 以及 **wvalid** 升高才升高完成 handshake，並且當這四個訊號都為 1 時 **tap_WE = 1111** 寫入係數

(註：為了方便 debug，**tap_A** 沒用時會被改為 0xfff)

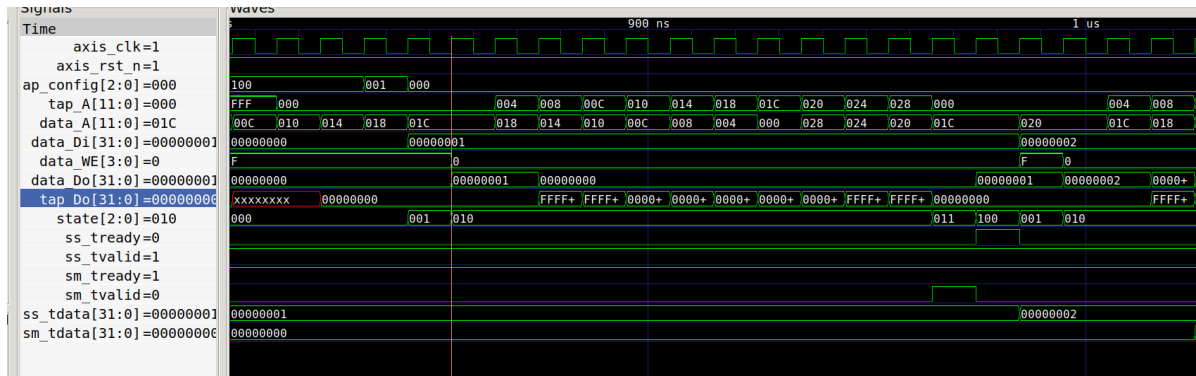
AXILITE read





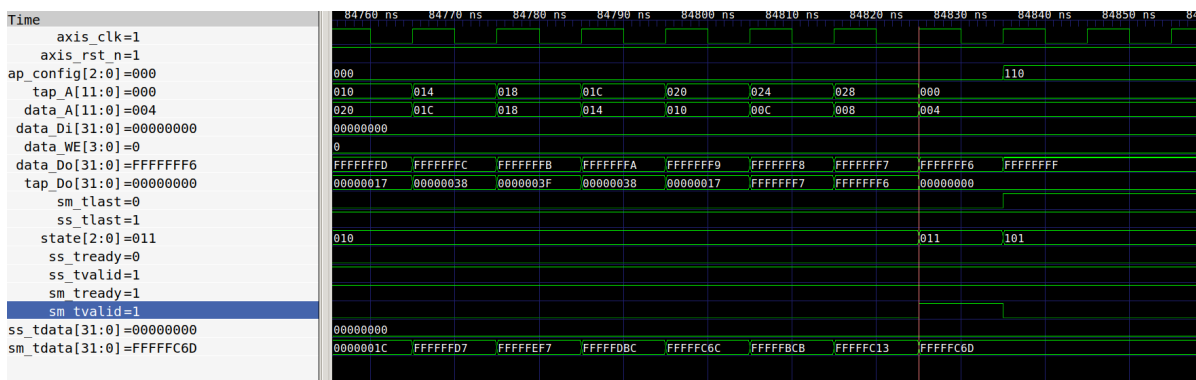
在讀取的過程，`arready` 等待 `arvalid`，當 `arvalid` 升高時，`tap_A` 就會輸入對應的地址，並在下一個rising edge 存到 `rdata` 中，之後拉高 `rvalid` 直到 `rready` 也為 1

AXI stream



(state = {0: idle, 1: get, 2: comp, 3: out, 4:next, 5,last})

當 `ap_start` 被打開後，下個cycle開始計算，當state 進入 out時 `sm_valid = 1`，因為這一個狀態要等到 `sm_ready = 1`，視另一端(testbench/host)而定可能會有多个 cycles，所以把 `ss_ready` 改到 next 狀態再升高，取得剛好一筆新資料



當 `ss_tlast` 升高後，下一個 out 狀態完會進入 last 狀態，並將 `sm_tlast` 打開，`ap_idle = 1`，`ap_done=1`，結束運算

Resource usage

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	217	53200	0.41
FF	130	106400	0.12
DSP	3	220	1.36
IO	330	125	264.00
BUFG	1	32	3.13

timing summary

```
clock constraint create_clock -period 4.000 -name axis_clk -waveform {0.000 2.000}
[get_ports axis_clk]
```

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	217	53200	0.41
FF	130	106400	0.12
DSP	3	220	1.36
IO	330	125	264.00
BUFG	1	32	3.13

LUT	FF	BRAM	URAM	DSP	S
217	130	0	0	3	

critical path

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[10]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 2	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[11]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 3	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[2]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 4	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[4]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 5	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[5]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 6	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[6]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 7	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[7]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 8	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[8]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 9	0.187	3	4	9	data_addr_reg[6]/C	data_addr_reg[9]/D	3.677	1.045	2.632	4.0	axis_clk	axis_clk
Path 10	0.449	3	4	3	tap_addr_reg[4]/C	tap_addr_reg[8]/D	3.415	1.787	1.628	4.0	axis_clk	axis_clk