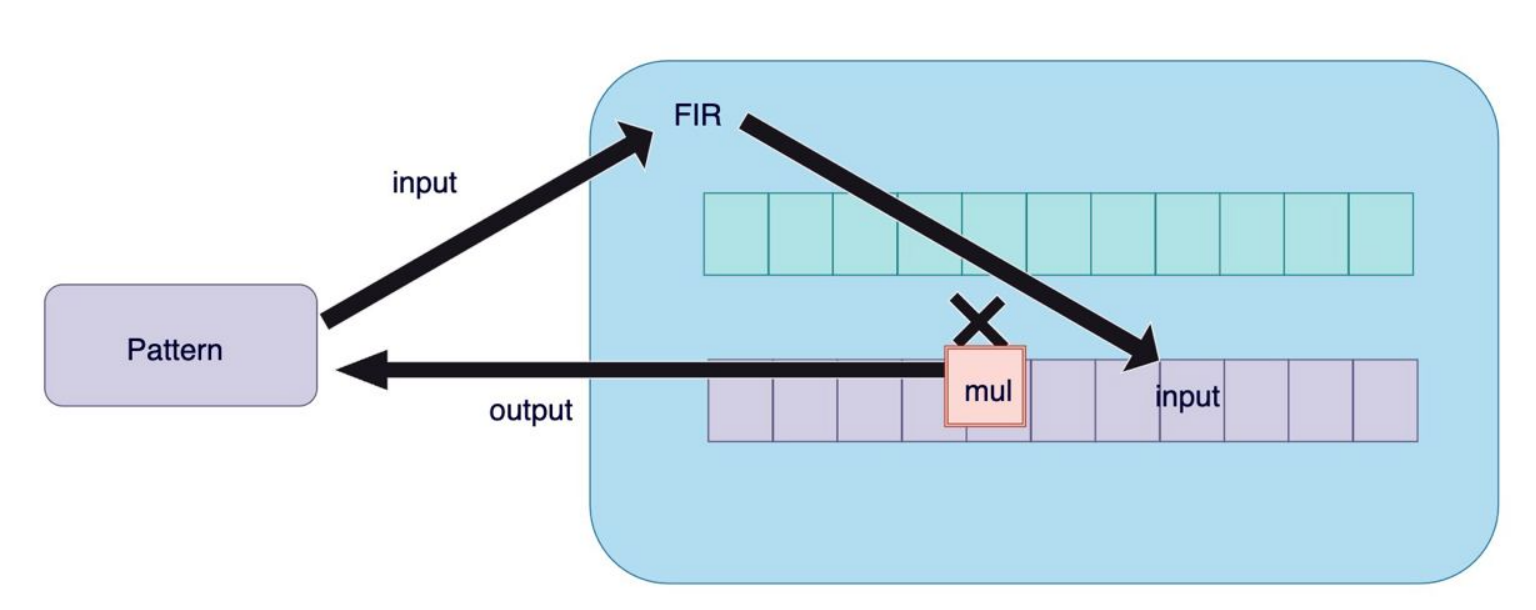
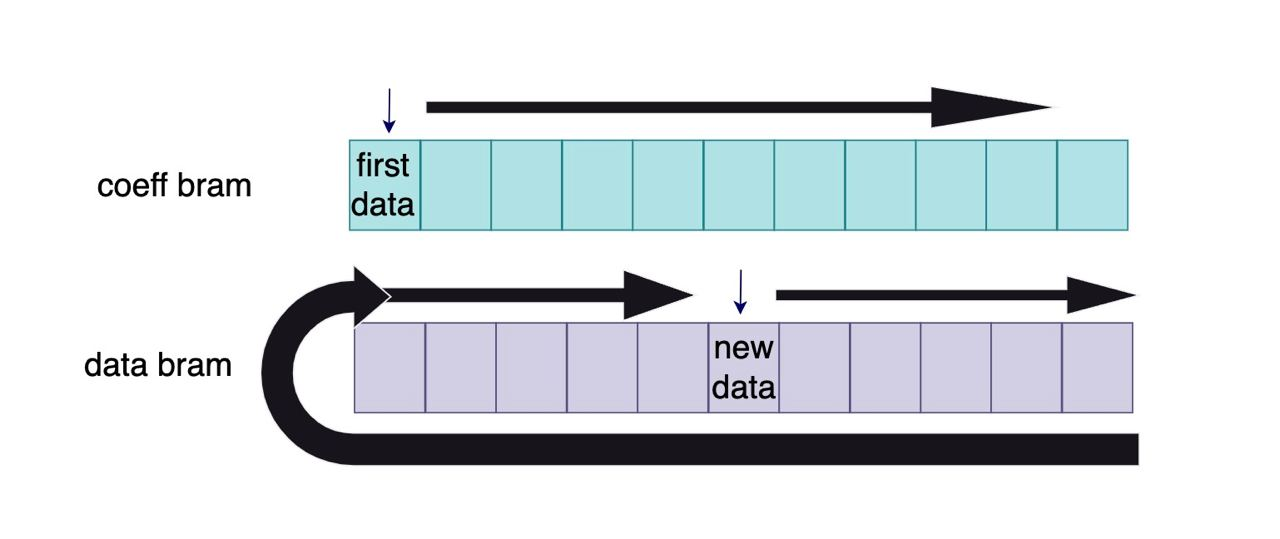
Dataflow:

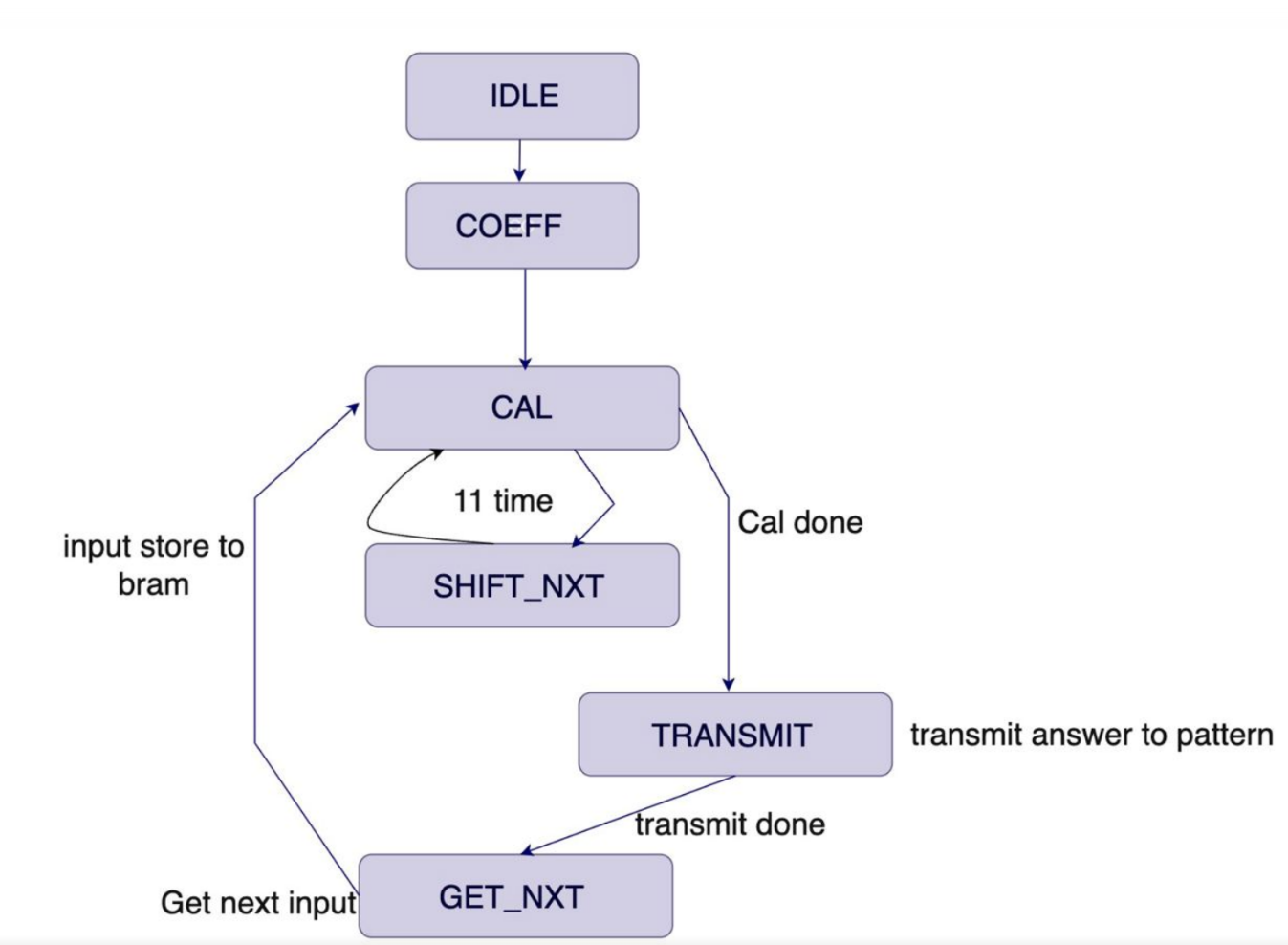


Bram operation



我使用兩個bram11來分別存coefficient和data。Coefficient的部分由於值不會變化，所以比較簡單。 而data bram的部分因為每次新data都會被存在新的位置，所以用一個new data address pointer來代表最新的值的addr。再進來新的一個值，就要再往下一個addr。而每次要算出一個結果，就要使用address shift register取bram值。取一個值要2cycles，所以算出一筆資料需要11\*2=22cycles。

FSM:



FSM: 我沒有用pipeline加速,由於之後的lab還會用到這個ip,所以我希望他的state可以很乾淨,debug起來不會很麻煩(也是因為還不確定後面的lab會如何驅動這個ip)。所以state跟state之間劃分得很乾淨,當然,隨之而來的弊端就是latency會比較長。各個state大致上在做以下的事情:

IDLE: reset後或是ap\_done拉起來,就會進入到idle。Idle 1 cycle後就會進入get coefficient。

COEFF: 接收coefficient並且存在coeff bram11中

CAL: 將coeff bram的值與data bram的值相乘到accumulation register中。如果已經是第11個數值就進入transmit。

SHIFT\_NXT: 同時將coefficient bram和 data bram shift到下一個bram address取值

TRANSMIT: 將算好的一個值傳送給Pattern。傳好就進入GET\_NXT。

GET\_NXT: 取得下一個data input。並且存在data bram中。

AXI:

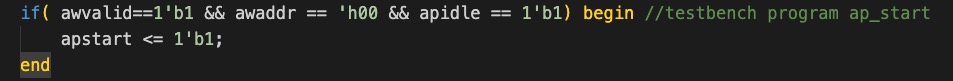
AXI-stream: 用來接data。主要使用ss\_tready和ss\_tvalid的handshake來判斷。而ss\_tlast是代表最後一個input的訊號。

AXI\_lite( configure write ): 用來傳coefficien，使用aw,ar,w,r等channel來與pattern溝通。在這個lab助教給的pattern中，awready和arready這兩個signal 沒有用到，不知道之後的lab會不會用到。

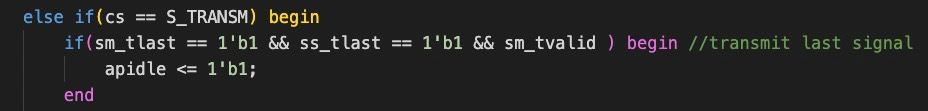
AP:

都是使用prototcal的訊號判斷拉起或歸0。

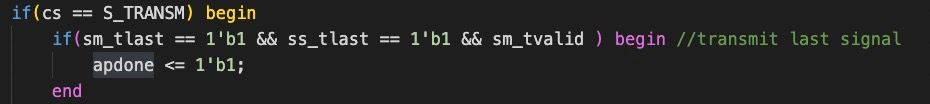
Ap\_start:



Ap\_idle:

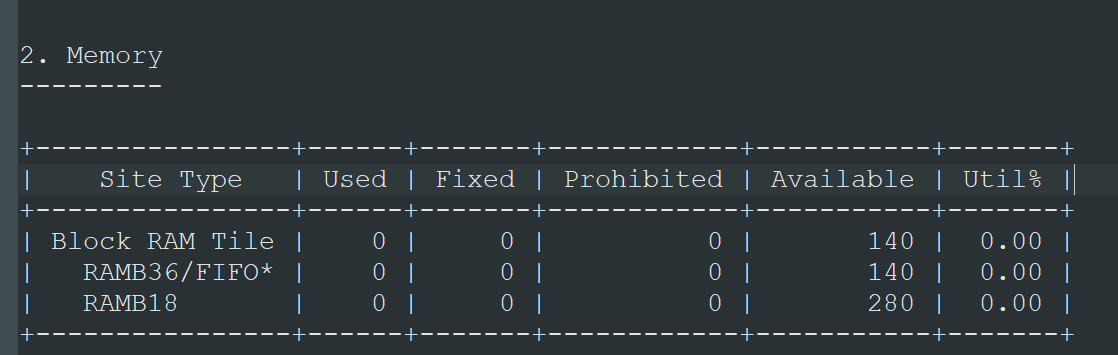


Ap\_done:

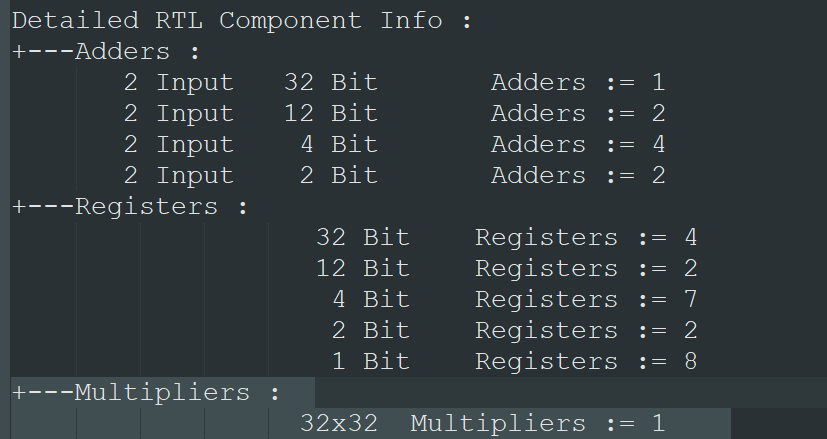


Resource Utilization:

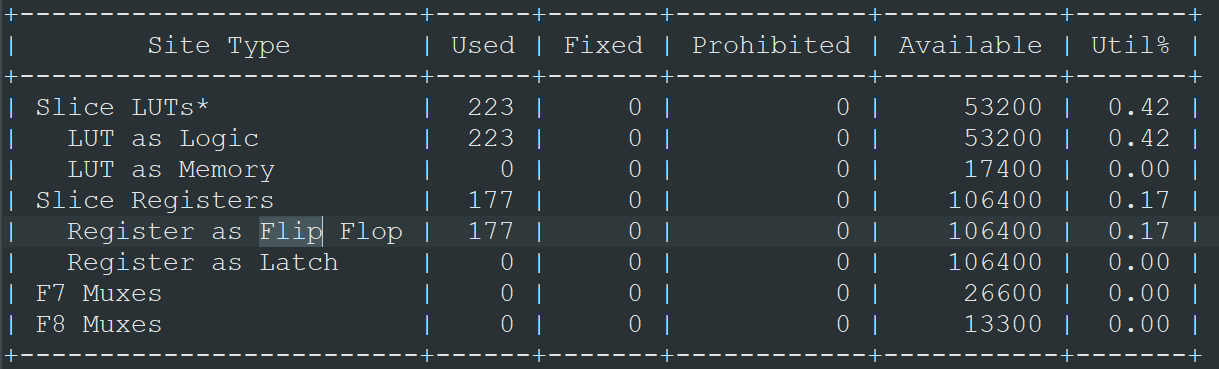
Memory: 2 Block RAM



Multipiler: 1



LUT and FF



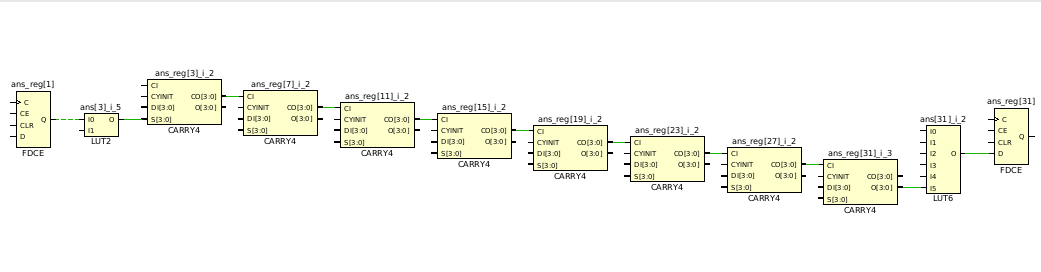
Timing:

就是在取bram值並且相乘再存到accumulated register的那條path。

花費3.761ns 留下了6.103ns的slack

表示最快大約可以synthesis到4ns左右(留一點timing給APR)





Waveform:



From ap\_start to ap\_done: 10209 cycles for 600 inputs。 Average 17.6 cycles for an output

Simulation result:

