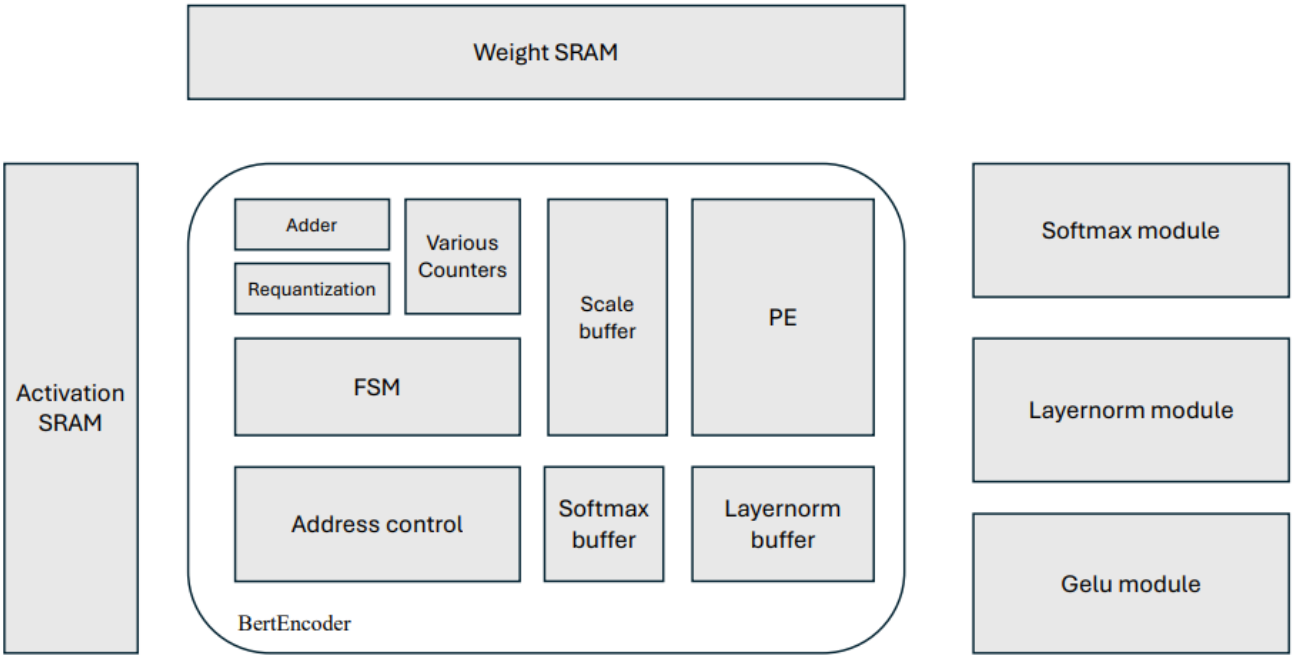
Final Project Report

Student ID: 112061620、111061642 Name: 郭邑哲、王煒翔

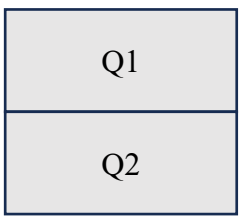
1. Design concept:
   - Block diagram:



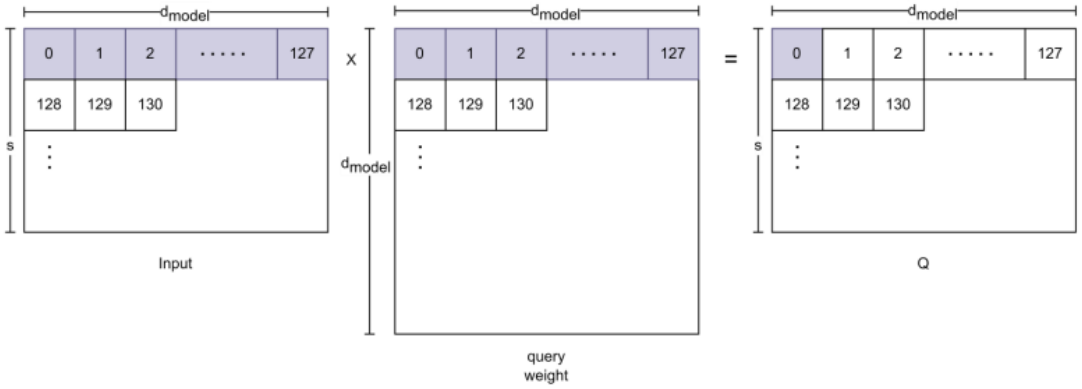
- Activation SRAM : We can perform two 128-bit memory accesses of separate addresses in the same clock cycle because it is a given dual-port SRAM. And in Final project, we use ”free to use” space to store some computation result.
- Weight SRAM : For scales, we store four 32-bit scales in each address. For weights, we store sixteen 8-bit weights in each address. For biases, sixteen 8-bit bias is stored in every address.
- PE :因為我們一次可以取出 2 組 16 個 8-bit 的 activation 做運算，因此這個 PE 是由 32 個乘法器所組成的，且在整個系統中只會使用一個 PE，以此達成硬體共用。另外，我們將它採取 adder tree 的方式，以此來減少 critical path。
- Scale buffer :用 buffer 來暫存從 Weight SRAM 的 offset:0-5 取出的 scale。
- Layernorm buffer : Layernorm module requires a total of 128 elements in the same row to start the computation, 而我們採取的方式為先將 data 分開送進 Layernorm module 中且過程中並不會存入 SRAM。因此，會需要用到一個 256bit 的 layernorm buffer 來儲存 data，及兩個

256bit 的 layernorm buffer 來分別儲存 Weight 和 bias。而從 Layernorm module 取出資料的方式則是連續 4 個 clock 取出並寫入 SRAM 中。

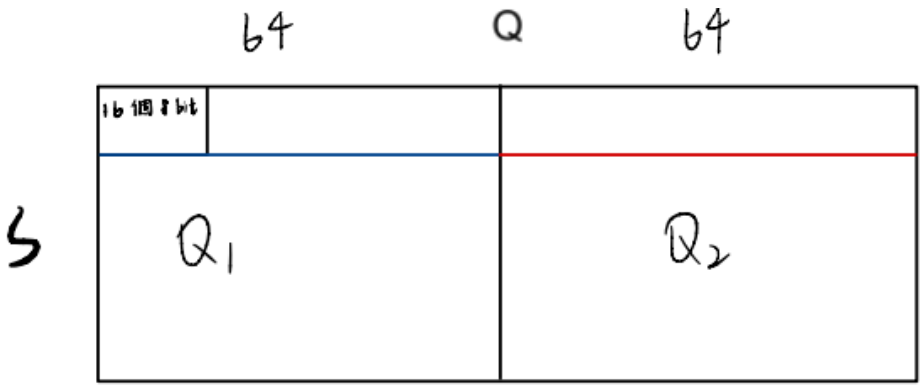
- Softmax buffer : The input of the softmax module is the result of the matrix multiplication and normalization of Query and Key. 輸出則是要與 V 做 Batch-matmul，而這個過程中，我們並沒有將其寫入 SRAM，而是透過 buffer 來儲存暫時運算出的結果並直接與 V 做運算。
- Requantization : In Final project, 我們選擇只使用一個 requantization 的硬體，來達成硬體共用，以此來最小化面積。
- Adder : 在做 Layernorm 之前，需要先將要傳入的兩個 input 做 requantization 及相加的動作，例如: 兩個輸入為分別從 Activation SRAM 中的 Quantized Input(offset:0-255)及 FC1 Output(offset:256-511)取出的資料，此時 dual-port 的 SRAM 並不像前面的運算方式能一次處理 32 筆 data，因為這兩組 data 都是從 Activation SRAM 中取出的，因此一次只能做 16 筆 data 的處理。所以總共有 16 個 8-bit 的 adder。
- FSM : In Final project, 設計了兩個 FSM 做互相控制的動作，首先第一組是對 data path 進行控制，這樣我們才能有效的使用 PE，例如:Compute state, normalize state, etc.第二組則是代表現在做到哪部份了，如果我們要共用硬體資源的話，勢必得將一些電路分開進行，例如: 我們會先將 Q 運算好放進 SRAM 後再進行 K 的運算，最後才執行 V 的運算。
- Address control : In Final project, 因為有許多 Transpose 及 multi-head 使用的過程，因此我們需要特別注意從 SRAM 取出 data 及放回 computation result 的方式，例如:當我們運算完 V 時，需要將它放進 SRAM 的方式做特別的處理，這樣我們將 V 取出和 QK 做運算時，會較為簡單且不易產生錯誤。
- Various Counters :用來當作 control signal 以及 address control.
  1. cnt :在運算矩陣相乘的 compute state 或是對 Adder 進行 requantization 的 requantization state 都不能在一個 cycle 就做完，因此我們需要 cnt 來控制它轉換 state 的時間和何時完成運算等等。
  2. Q1_Q2_cnt: In Multi-head attention, 我們會將 QKV 分別分成 Q1,Q2 及 K1,K2 及 V1,V2，而放進 SRAM 時， 設計 Q1 會放進 SRAM 較低的 offset 中，Q2 則放入 SRAM 較高的 offset，如下圖。Q1_Q2_cnt = 0 代表在運算 Q1 的狀態；1_Q2_cnt = 1 代表在運算 Q2 的狀態。

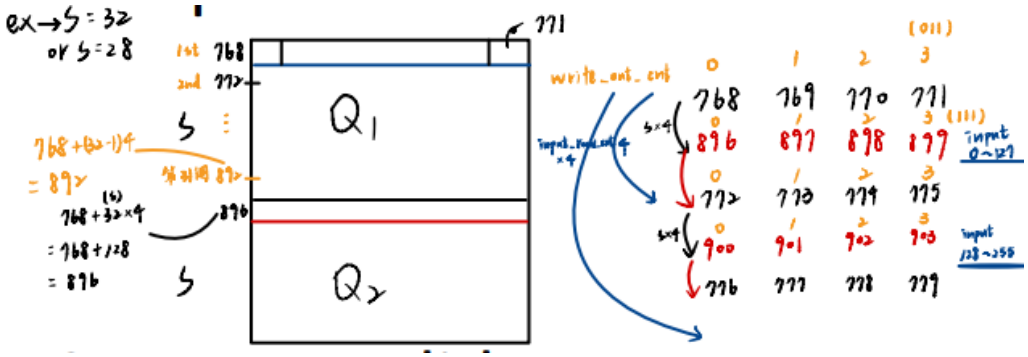


3. cnt5:把 head1 跟 head2 合併之後，後續寫地址都不用跳地址，這個是拿來給後續寫進去 SRAM 的 counter。
4. cnt6: 在跟 V 相乘時，用來判斷現在是否運算到 16 筆了，到了 16 筆就寫進去 SRAM。
5. write_cnt: 代表運算完了幾筆 data，可以傳入 SRAM 中。例如:在 Q 的運算時，因為我們是一次運算 16 筆 data 後將其透過 sram_act_wdata0 傳入 SRAM 中，因此 write_cnt 在 15 的時候即代表在處理最後一筆要傳入 SRAM 的資料。
6. write_cnt_bbb: 判斷現在要拿這個地址裡面的哪一個 bias。
7. write_bias_cnt: 代表現在是第幾個 bias 的地址。當我們算好 data 後，要將其加上一個 bias 才能傳入 SRAM，因此我們會設定這個 counter 在 data 傳入 SRAM 時，也做更新，這樣後面過程算好的 data 才能一直與正確的 bias 做相加。
8. weight_row_cnt: Input 與 Weight 做矩陣相乘時，我們會先固定 Input 的一個 row，並對 Weight 矩陣從最上層的 row 開始往下掃到最下層的 row 後，再換 Input 的第二個 row 以此類推。這樣做的原因是為了讓算好的 data 放進 SRAM 更有規律及方便性，下圖以 Q 為例子。



9. write_out_cnt: In Multi-head attention, Q 和 K 被分成 Q1,Q2 及 K1,K2，而我們在運算出 Output 的值時會遵循下圖的規律進行 Transpose，再將他們存進 SRAM，這樣對我們之後從 SRAM 取資料出來做運算也會更為方便。write_out_cnt 的功能就是判斷 Output 現在要寫入相鄰地址的哪一個地址，因此它的規律就是 0,1,2,3 這樣循環。下圖為 Q 經過 Transpose 的過程。

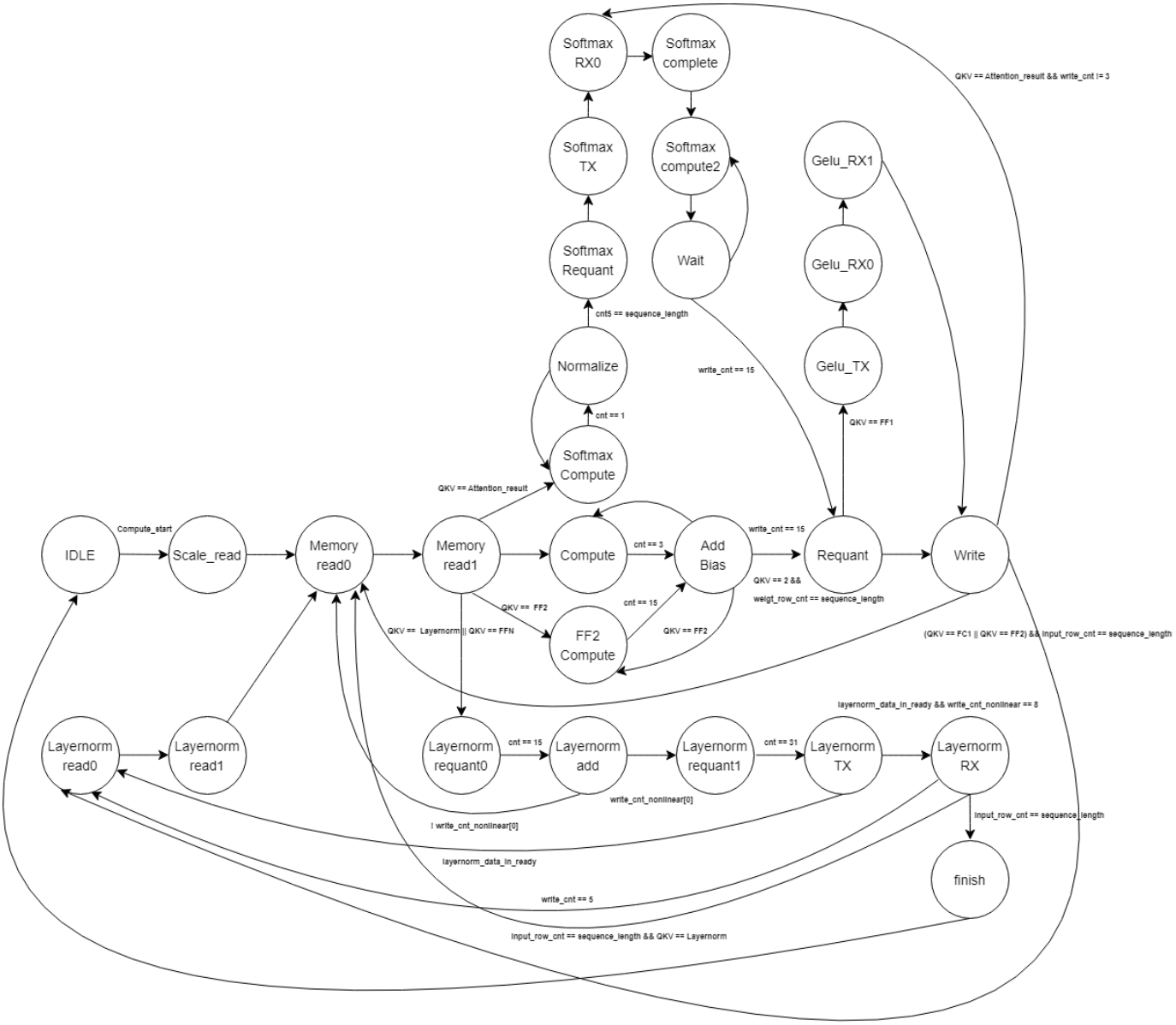




10. input_row_cnt: 用來判斷 input 現在算到第幾個 row，當 input_row_cnt 等於 input 的 sequence 時，我們就能知道現在執行最後一 row 的運算，也就是我們這個 input 的運算已經快要完成，可以接續做其他 state 的事情--在 FSM 中很常使用它作為控制訊號。
11. write_cnt_nonlinear: 用來給 non-linear model 使用的，例如: 在 Layernorm module，輸入我們採取分開送資料的方式，而 layernorm module 要收到 128 elements 才會開始運算，也就是說需要 8 組 8 個 16-bit 的 elements 才會開始運算，此時 write_cnt_nonlinear 就可以用來當作使 layernorm module 開始運算的控制訊號。

- FSM:

1. Data Flow 的 FSM



- IDLE :系統初始狀態--收到 compute_start 後，系統開始運作。
- Scale_read:讀取一開始運算時所需的 scale，並將其存在一個 32bit 的暫存器，實現 scale 的硬體共用。
- Memory_read0、Memory_read1 :從 SRAM 的 address 讀取相對應的 data 所需等待的時間，因為 SRAM 為 dual-port，因此地址會以+2 的方式更新。
- Compute、FF2_Compute :PE 開始運算的狀態，在此時 pe_enable 會被拉為 1。會開始將運算的值進行累加，並在 compute_state 的中間完成一個 data output 的運算，此時下一個 cycle 會將 PE 暫存器中累加的值歸 0，再進行下一筆 data output 的運算。

- Add_Bias :此時 pe_enable 會為 0，停止 PE 的運算。同時，在這個 cycle 將 PE 算好的 Output 與從 Weight SRAM 讀出來的 bias 進行相加，這樣就不需要額外使用硬體儲存 bias，以此節省面積。下一個 cycle 會回到 Compute state，做 requantization，以及繼續做下一個 PE 運算的 data output—這樣我們 requantization 在整個系統中僅需使用一個共用的硬體；另一方面，因為在同步進行 PE 運算以此來減少 cycle 數。
- Requant :為最後一筆 data output 算好之後，需花費一個 cycle 進行 requantization。
- Write :在此系統設計中寫回 SRAM 的方式大多採取 single-port 寫回，因此一次最多能寫回 16 筆 data output；在 Layernorm 時，因為我們為連續從 Layernorm module 讀取資料同時連續寫回 SRAM，因此採取 dual-port 寫回以此來節省 cycle 數以及面積。另外在儲存 K 時，考慮到後續要做 QK 的 matmul，因此將 K 儲存在 Weight SRAM，這樣才能實現 SRAM dual-port 的功能。

--Softmax

- Softmax_compute :從 Activation SRAM 讀取 Q 的 data；從 Weight SRAM 讀取 K 的 data，進行 QK matmul 的運算，同樣地，此時 pe_enable 會為 1，PE 會開始運算。
- Normalize : 將 QK matmul 的值進行 Normalize—將其右移 3 位達成/sqrt(64)的功能。
- Softmax_requant :為最後一筆 data output 算好之後，需花費一個 cycle 進行 requantization。
- Softmax_TX :此時 softmax_data_in_valid 為 1，代表可以傳值給 Softmax non-linear model，當 softmax_data_in_ready 為 1 時，代表 Softmax non-linear model 已經收到值了，此時 TX state 可以轉變為 RX state，準備接收從 Softmax non-linear model 運算好的值。
- Softmax_RX :此時 softmax_data_out_ready 為 1，代表 Bert_encoder model(my design)已經準備好接收從 Softmax non-linear model 運算好的值，當 softmax_data_out_valid 為 1 表 Softmax non-linear model 傳值給 Bert_encoder model。
- Softmax_complete :這個 state 代表完成了 Softmax 的運算。此次設計並沒有選擇將 Softmax 運算出的值存入 SRAM，而是運算出來後，直接與 V 做 matmul，以此來節省 cycle 數。
- Softmax_compute2 : 從 Activation SRAM 讀取 V 的 data，並與剛運算好 QK matmul 的值進行 matmul，此時 pe_enable 會為 1，PE 會開始運算。
- Wait: 設一個 cycle 給 PE 中的暫存器歸 0。

--Layernorm

- Layernorm_read0、Layernorm_read1 :從 Weight SRAM 讀取 Layernorm model 所需的 Layernorm weights 和 Layernorm bias。
- Layernorm_requant0 :將從 Activation SRAM offset:0-255 讀出的 16 筆 Quantized input 先經

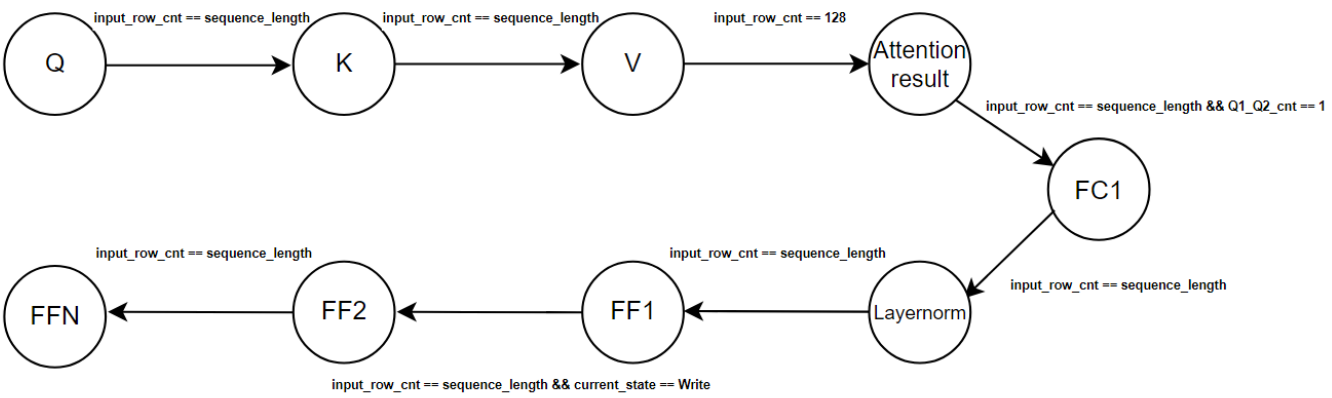
過 requantization，因為我們只設計一個 requantization 硬體，因此需耗費 16 個 clk。

- Layernorm_add :將經過 requantization 的 Quantized input 與從 Activation SRAM offset:256-511 讀出的 16 筆 FC1 Output 進行相加，並存進 layernorm buffer 中。
- Layernorm_requant1 :Layernorm model 要開始運算需要 128 個 elements，而我們設計分 4 次傳送 32 個 elements 給 Layernorm model。將將經過 requantization 的 Quantized input 與從 Activation SRAM offset:256-511 讀出的 16 筆 FC1 Output 進行相加的 layernorm buffer 中的值進行 requantization(attention_add)，耗時 32clk 後傳入給 Layernorm model。
- Layernorm_TX : 此時 layernorm_data_in_valid 為 1，代表可以傳值給 Layernorm non-linear model，當 layernorm_data_in_ready 為 1 時，代表 Layernorm non-linear model 已經收到值了，此時 TX state 會回到 Layernorm_read0 重新進行下一個 32 elements 的運算，當將 128 個 elements 運算好就可以轉變為 RX state，準備接收從 Layernorm non-linear model 運算好的值。
- Layernorm_RX : 此時 layernorm_data_out_ready 為 1，代表 Bert_encoder model(my design) 已經準備好接收從 Layernorm non-linear model 運算好的值，當 layernorm_data_out_valid 為 1 表 Layernorm non-linear model 傳值給 Bert_encoder model。由於從 Layernorm non-linear model 讀值出來我們採取連續讀出，因此 layernorm_data_out_valid 會維持 4clk，共讀出 4 組 32 elements。

--Gelu

- Gelu_TX :此時 gelu_data_in_valid 為 1，代表可以傳值給 Gelu non-linear model，當 gelu_data_in_ready 為 1 時，代表 Gelu non-linear model 已經收到值了，此時 TX state 可以轉變為 RX state，準備接收從 Gelu non-linear model 運算好的值。
- Gelu_RX :此時 gelu_data_out_ready 為 1，代表 Bert_encoder model(my design)已經準備好接收從 Gelu non-linear model 運算好的值，當 gelu_data_out_valid 為 1 表 Gelu non-linear model 傳值給 Bert_encoder model。

2. 各個執行步驟的 FSM



使用 QKV 這個暫存器儲存目前執行的步驟。因為許多硬體是共用的，因此需要用 QKV 此暫存器來決定執行的先後順序。

- Q: Query -- Multi-head attention
- K: Key -- Multi-head attention
- V: Value -- Multi-head attention
- Attention_result: 此 state 表運算 QK 的 matmul 並將值傳進 Softmax model 後，取出 Softmax model 運算後的值並與 V 做 matmul 後存進 SRAM。
- FC1: Fully connected layer
- FF1、FF2、FFN: Fully connected layer -- Feed Forward
- Layernorm: 執行 Layernorm -- Add&Norm

- Please explain your dataflow in detail.

  How many cycle do you need to read/write weights/activations?

  Scale : 最一開始要讀 Q 的 Scale，花費一個 cycle

  Quary、Key、Value、FC1、Output: 2 個 cycle 來讀 address，然後開始讀 input 跟 weight data，會花 5 個 cycle 做完一筆 8bit output，然後滿 16 個 output 寫進去一次，也就是 5*16 = 80 cycle，由於最後一筆需要等它做完 requant，所以從讀出來資料到寫進去 output，需要 2+60+2 = 64 cycle 為一個循環，addr 有 2*S*4 個 address，所以做完需要 512*S 個 cycle。

  Attention result : 最前面花 2 個 cycle 來讀地址，然後先算 Q*K，由於 Q 跟 K 的維度都為 S*S，所以做完一筆需要 3 個 cycle，由於 softmax 規定必須要一次傳入一個 row，所以進到 softamx 前需要 3*S 個 cycle，一樣要等 cycle 讓最後一筆 output 進到 softamx，softmax 需要等待 3 個 cycle 出去跟 V 相乘，由於 V_i 的維度為 64*S，做完一次 output 只需 2 個 cycle，一樣等到 16 筆 output 處理完，需要 2*16 個 cycle，所以從最一開始讀資料到寫進去需要 2 + 3*S + 3 + 2*16 + 2 = 3*S+39 個 cycle，但由於我們的 V_i 有 64 個 row，還沒有全部乘完，所以做完一個 row 乘完所有的 V，需要 4*(2*16 + 2) + 6 = 142 cycle，所以做完一整個大循環需要 2 + 3*S + 3 + 142 = 3*S + 147，我們的 Q 總共有 2*S 個 row，所以需要 2*S(3*S + 147) 個 cycle 來完成整個的 attention result。

  Layernorm: Layernorm model 一次運算為 128 elements，首先以前三組 32elements 講解，需要兩個 cycle 讀 weight 和 bias 以及 4 個 cycle 讀地址的值，還有兩個 16clk 和一個 32clk 做 requantization，最後加上兩個 add cycle 和一個 cycle 的 TX。最後一組 32elements，則是要考慮到 RX(包含寫回 SRAM)共為 6clk。因此 Layernorm 部分總共需要 ((2+4+16*2+32+2+1)*4+6)*S cycles，而此次 data flow 有兩個 Layernorm 步驟，因此總共耗時的 cycle 數為((2+4+16*2+32+2+1)*4+6)*S*2。

  Gelu: 前面跟第一個一樣 2 + 5*16 + 2，但這裡我算完進去 gelu 出來才會存進去 sram，gelu 跟 softmax 一樣等 3 個 cycle，所以寫進去一次 sram 要花 87 個 cycle，這裡有 S*32 個地址，所以需要 2784*S 個 cycle。

  FF2 : 這裡有點不一樣，由於一個 row 有 512 個，所以需要 16 個 cycle 算完一筆 output，寫進去一筆 output 需要 16 *16 ，output 有 S*8 個地址，所以做完需要 S*8*16*16 = 2048*S。

How do you compute the result in how many cycle?

以下使用 Query 做為範例:

首先固定 input 的一個 row，然後將 Weight 從最上層的 row 開始一路往下掃至第 16 個 row。圖 3-1 為運算出 Q 的第一個 row 的 16 個 elements 之時序圖，並將其存進 Activation SRAM offset:768 -- 一個 SRAM 的 addr 可以儲存 16 個 8-bit 的 elements。

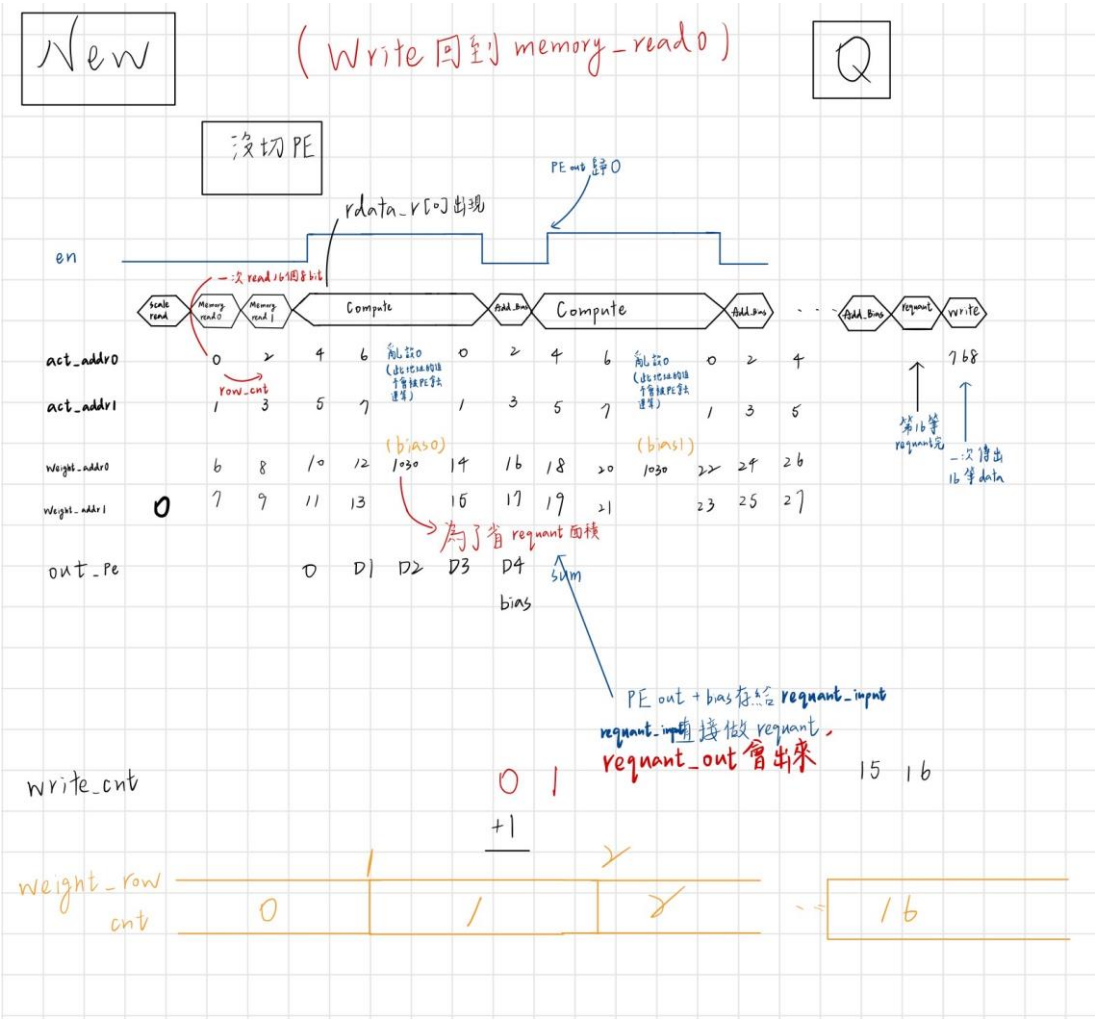


圖 3-1

從 Weight 的第 17 個 row 掃至第 32 個 row，而 Weight 共有 128 個 row，即可找到規律，運算出 Q 的第一個 row 的第二組 16 個 elements 之時序圖，並將其存進 Activation SRAM offset:769，Q 向右儲存的規律也為此，如圖 3-2。

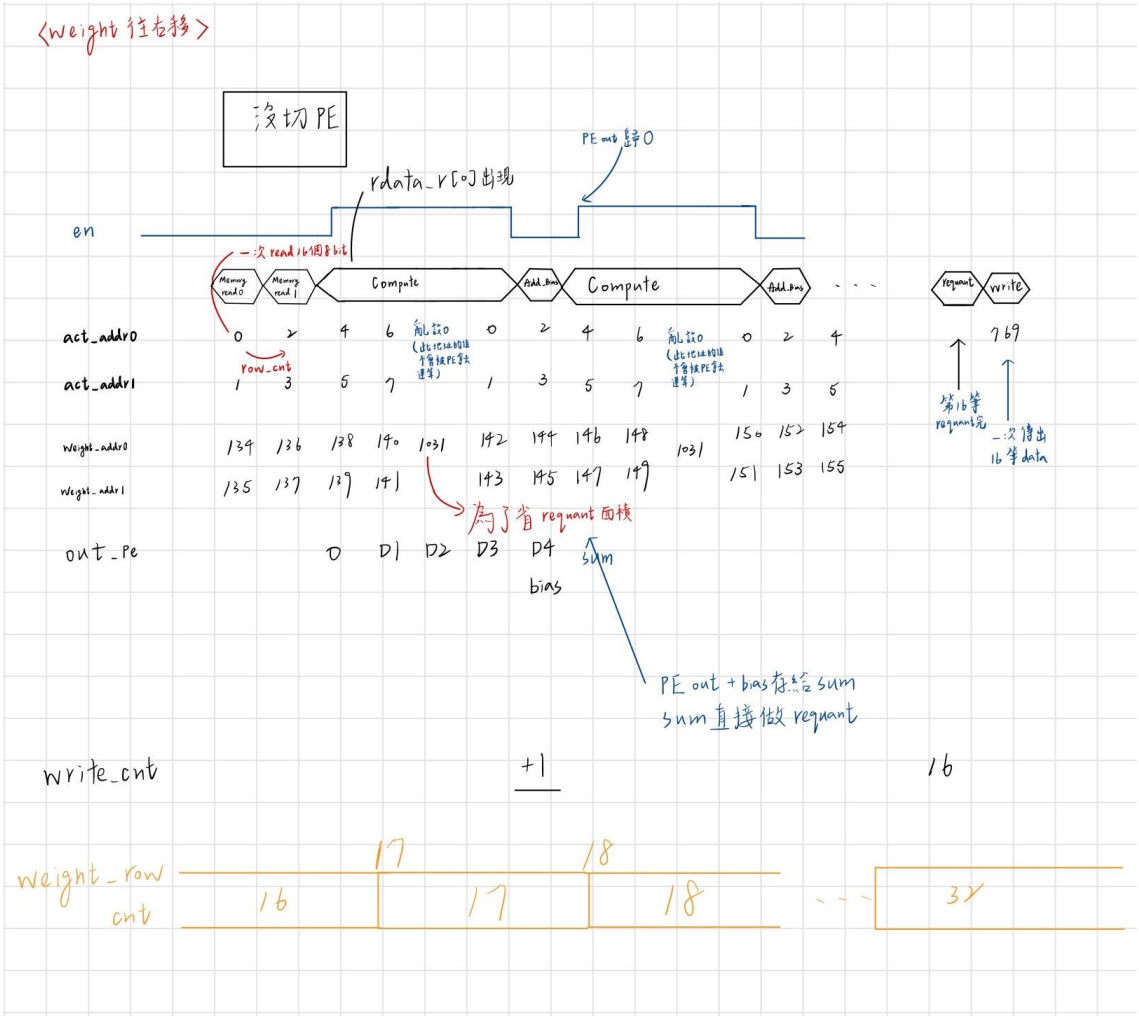
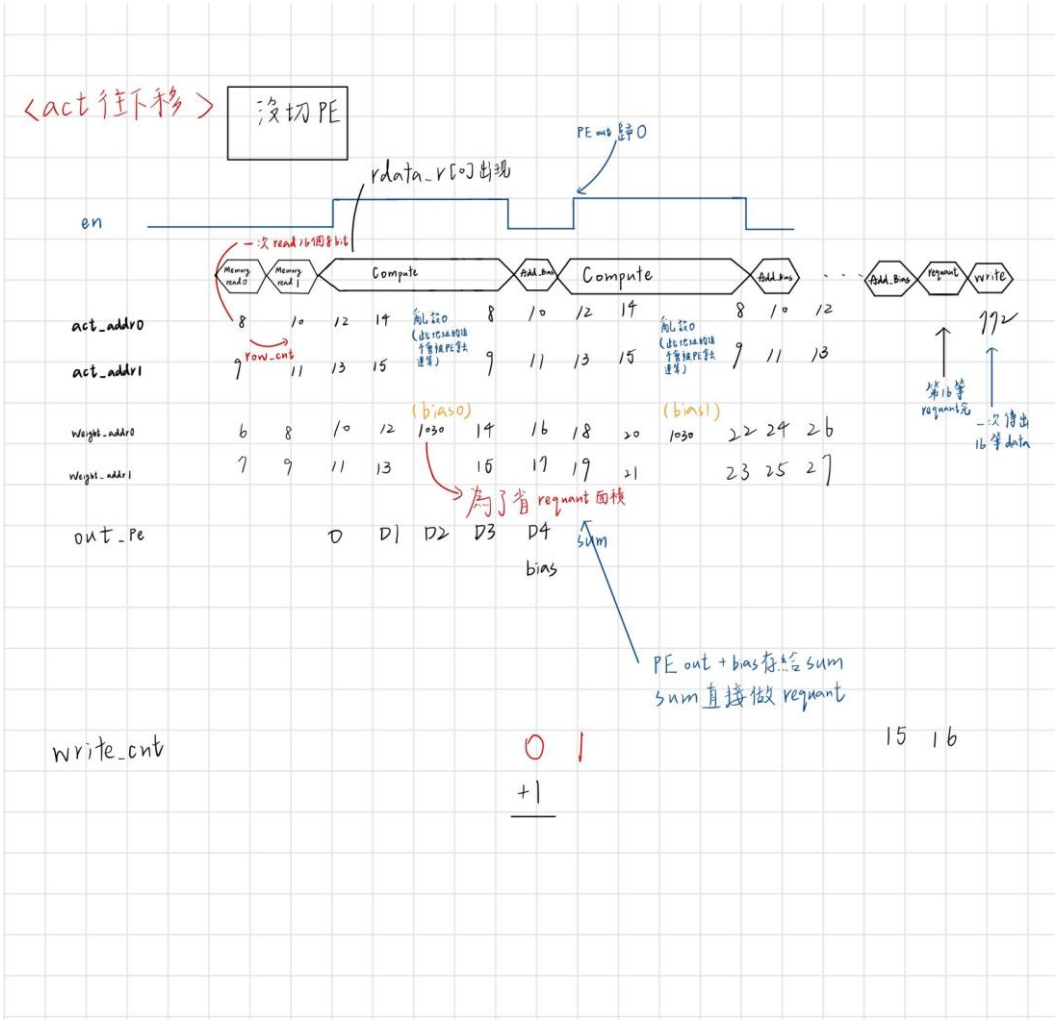
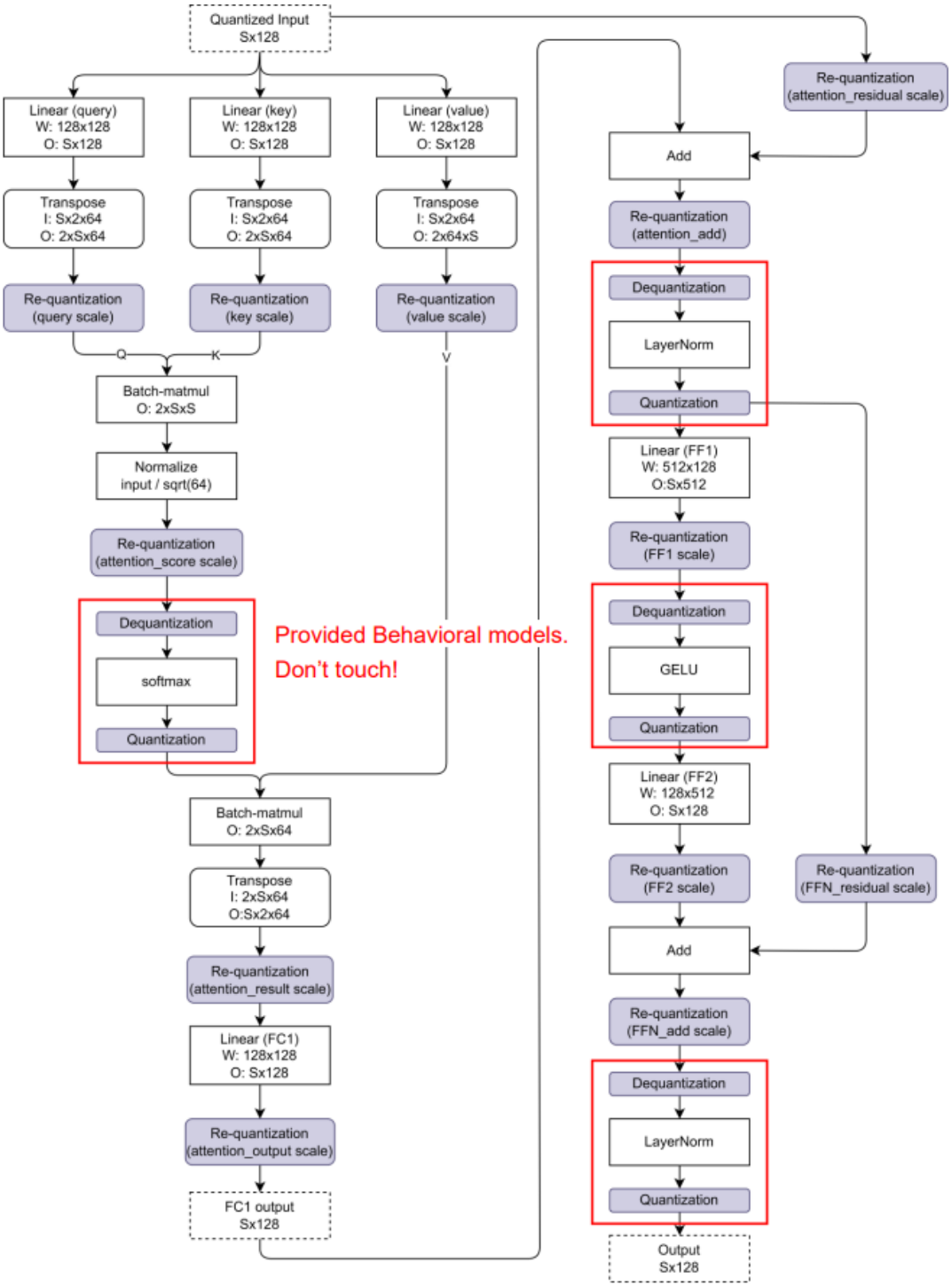


圖 3-2

當 Weight 從第一個 row 掃至最後一個 row 後，表完成 Q 的第一個 row，接下來會從 input 的第二個 row 開始運算，並重複圖 3-1 及 3-2 的步驟。最後即可完成 Query 的運算。



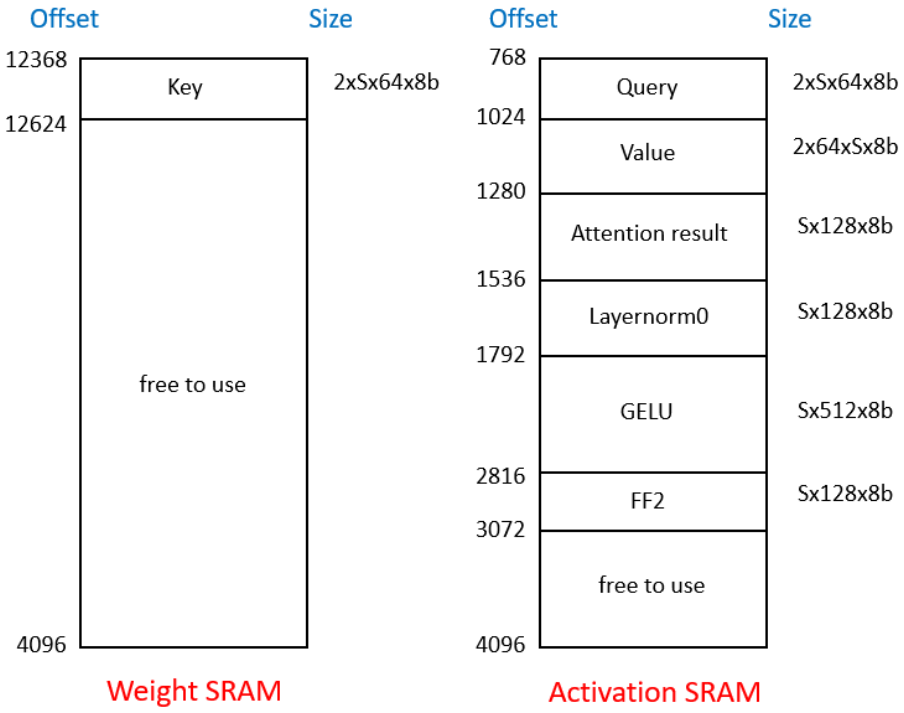
按照此次作業的 data flow，並考慮從 SRAM 讀取 data 及寫回 data，以及各個 state 運算的時間，考慮 S=28，我們最後可以得到總共需 244852cycles 完成運算。



What data do you keep in register for later reuse?

1. 使用一個 scale register 來給電路做使用，已達成硬體共用。在 layernorm 的過程中，因為需一次傳入四種 scale，因此另外多增加了四個。
2. Layernorm_buffer 代表儲存 layernorm 的 data 及 weight 及 bias，因為在 layernorm 的過程中，並沒有寫回 SRAM 做儲存，因此使用 Register 做儲存，以利後續使用。
3. Counter 也是使用 register 儲存現在所數到的數值，以便達到 data path control 的目的。
4. 儲存狀態的 register，以便達到 state control。
5. PE 進行累加時所需要的 Register，用來儲存矩陣相乘相加的結果。

- Please explain the usage of the free-to-use space in the SRAMs.



Weight SRAM：只有 key 存在這裡的原因是因為我們的 query 是因要跟 key 相乘的，所以放在 weight sram 這樣就可以一次取兩個地址出來相乘。

不過值得一題的是我們的 FC1 output 也可以存在這，原因在於 Add 的地方，input 跟 fcl output 都被指定存在 activation sram，所以我們要相加的時候一次只能取一個地址去做相加，如果選擇放在 weight sram，就可以一次取兩個地址出來相加，可以省下 cycle 數，不過如果這樣的話，我們就必須要再加一倍數量的電路來處理這裡的 add，最終決定犧牲 cycle 數，來換取小面積。

Activation SRAM:大部分都選擇放在這的原因在於後面都是 Fully connected layer，要跟 weight 去做相乘，所以必須放在這裡，Value 選擇放在這的原因在於，我們 QK 做完後進去 softmax 出來並沒有先存進去 sram，而是選擇直接跟 Value 相乘，再放入 sram，也就是我們的 attention result。

第一個 Layernorn 出來一定要存，因為後面 Feed Forward 出來要做 add&norm，需要這裡的 data，也就是 Layernorm0。

Gelu 出來到 FF2 之前要存的原因在於，我前面的所有 fully connected layer 之前都有把 data 存進去，所以這裡存進去就比較好沿用之前的狀態去延續，也就是 GELU。

做完 FF2 這裡也要存的原因在於，可以延續第一個 add &norm 的狀態，也就是 FF2，做完之後就可以存到最後的 Output 去做驗證。

- Any additional information you want to provide.

Spyglass rpt 討論:

a. Warning Input 'softmax_data_in_ready' declared but not read.
   由於助教們給的 nonlinear modle，in_vaild 起來時，in_ready 一定是 1，所以不需要去判斷 ready 是否等於 1，只需判斷 vaild 啥時為 1 就好，不過廣義來講必須考慮 ready，不過考慮這次的 spec 跟 design 效能，選擇不用。
b. Warning Input 'gelu_data_out_valid' declared but not read.
   這裡的原因在於當 in_vaild 起來時，過 2 個 cycle 出來，所以我這裡不用判斷 out_valid，用狀態去判斷啥時拿 gelu 的有效值。
c. Warning Input 'gelu_data_in_ready' declared but not read.這裡的解釋跟 a 一樣。
d. Warning Rhs width '27' with shift (Expr: '(temp >>> 16)') is more than lhs width '11'
   這裡警告說會有 overlflow，跟 hw4 一樣的問題，因為我們 requant 出來一定是 8bit，所以中間的乘法右移不需要那麼多 bit，所以我有砍 bit 數，不過有去測試幾筆 pattern 都沒有問題，所以這問題可以忽略。
e. Fatal Could not find clocks for all the flops. Please add clock SGDC constraint to the design
   這個 fatal 可以忽略，因為助教給的 constrain 裡面有檢查 clk，不過要檢查 clk 需要有 SGDC 檔，不過我們沒有，所以有 FATAL。
f. FATAL 'Advanced Lint Policy' not run due to unavailability of Auto_Verify license feature
   原因在於軟體限制，我們沒辦法用。
g. FATAL 'Advanced Lint Policy' not run due to unavailability of Auto_Verify license feature
   同 f。

2. Result

Item	Description	Unit
RTL simulation	PASS	---
Gate-level simulation	PASS	---
Gate-level simulation clock period	2.89	ns
Gate-level simulation latency	244852	cycles
Total cell area	57811.749690	um^2

3. Contribution (skip this part if you are in a 1-person group)

Item	郭邑哲	王煒翔
Architectural design	50%	50%
Coding	50%	50%
Report writing	50%	50%

4. END_CYCLE (optional)

- If you need a larger END_CYCLE than the default, please let us know what value you are using.

END_CYCLE	

5. Others (optional)

- Suggestions or comments about this class to teacher or TA.

謝謝助教，助教辛苦了!!