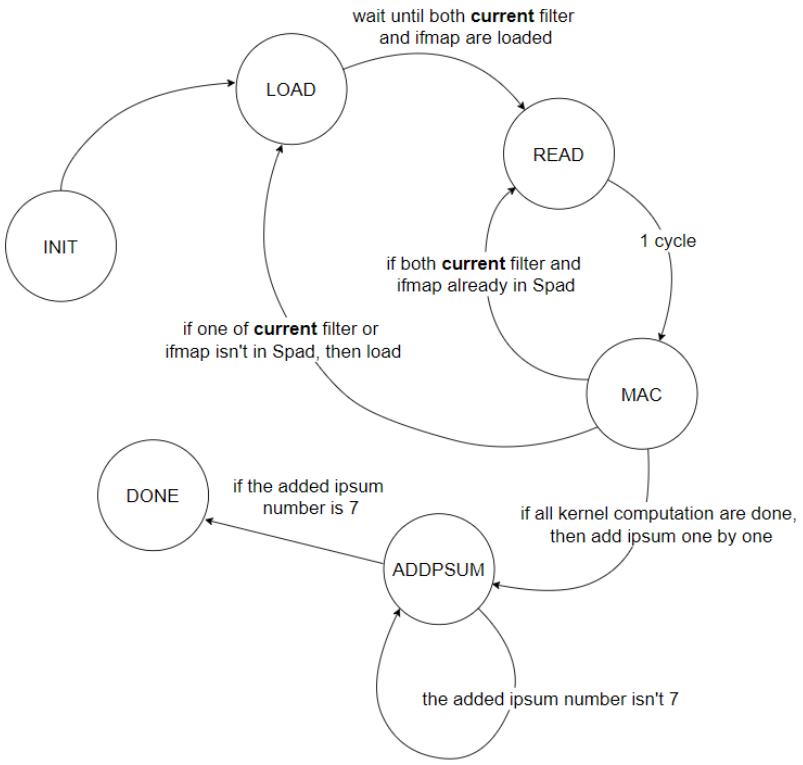
Implement PE

Yu-Chi Chu

28 Apr 2024

1. Draw flow chart, FSM of your design.

This design uses a straightforward state machine. For each kernel operation, it first checks if the current filter and ifmap have been loaded into the SPAD. If one or both are not in the SPAD, it goes to the LOAD state, then READ to perform the MAC operation. Otherwise, if both are in the SPAD, it jumps from MAC to READ, skipping the LOAD step. Before this, all kernel operation results are stored in the corresponding location in the psumSPAD. After all kernel operations are completed, the FSM jumps to the ADDPSUM state to add the ipsum sent from other PEs and output to opsum.



2. Describe how your PE work.

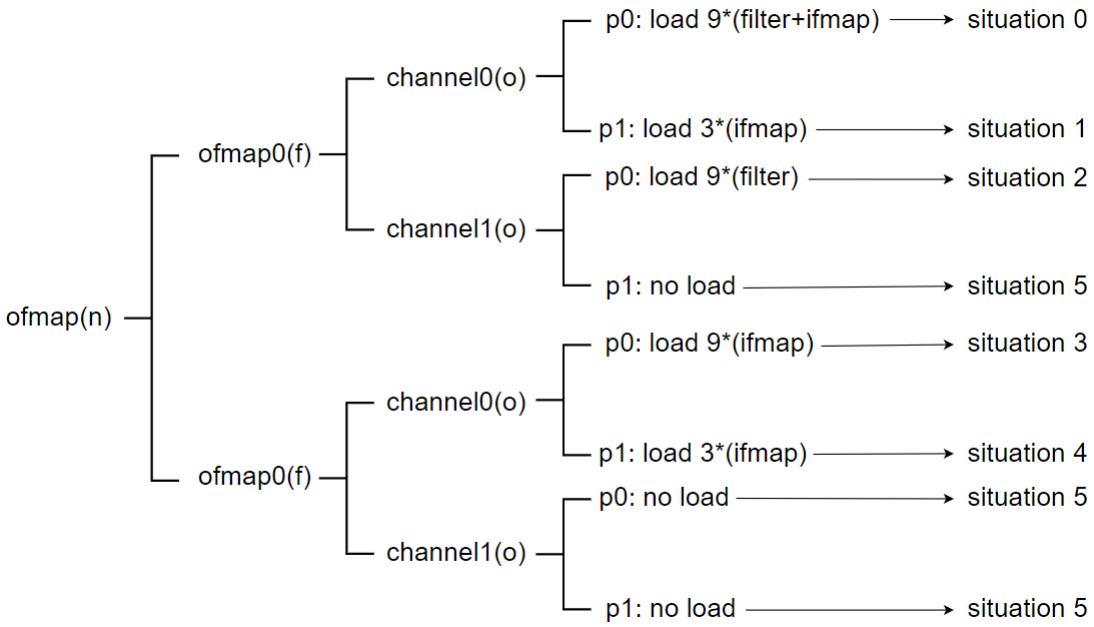
a) Mechanism for Determining LOAD or READ Transition:

To optimize data loading for the Multiply-Accumulate (MAC) operations, a mechanism is implemented to decide whether to load data from memory or read it directly from the local scratchpad memory (SPAD). This mechanism avoids redundant memory accesses, thus improving performance.

The process begins by constructing a dependency tree representing the filters and ifmaps required for processing all output feature map (ofmap) pixels. This tree structure helps in efficiently managing and tracking data dependencies.

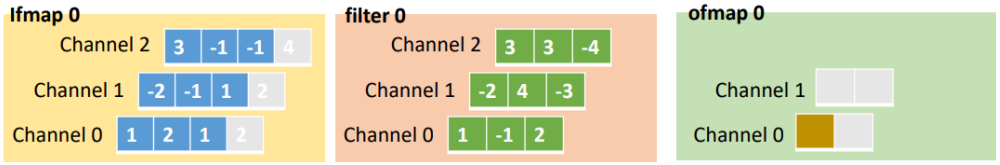
The decision to transition to either the LOAD state (to fetch data from external memory) or the READ state (to access data already in the SPAD) is made based on the "situation." This "situation" is determined by a combination of factors:

- **Current ofmap:** The specific output feature map being processed.
- **Current output channel:** The output channel being computed.
- **Current output pixel:** The specific pixel within the output feature map.
- **Kernel calculation counter:** A counter tracking the number of kernel computations performed so far.

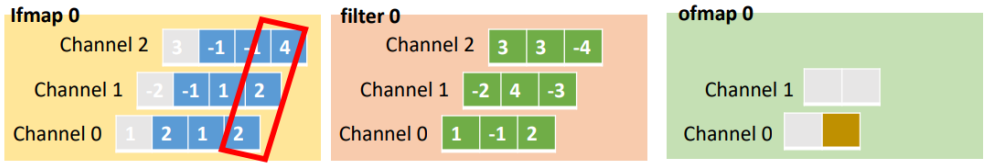


```
365     // situation
366     always@(*) begin
367         if(curOfmap == 4'd0 && curOChannel == 4'd0 && curOPixel == 4'd0) situation = 4'd0; // 9 (filter and ifmap)
368         else if(curOfmap == 4'd0 && curOChannel == 4'd0 && curOPixel > 4'd0) situation = 4'd1; // 3 ifmap
369         else if(curOfmap == 4'd0 && curOChannel == 4'd1 && curOPixel == 4'd0) situation = 4'd2; // 9 filter
370         else if(curOfmap == 4'd1 && curOChannel == 4'd0 && curOPixel == 4'd0) situation = 4'd3; // 9 ifmap
371         else if(curOfmap == 4'd1 && curOChannel == 4'd0 && curOPixel > 4'd0) situation = 4'd4; // 3 ifmap
372         else situation = 4'd5; // no load situation
373     end
```

1) **Situation 0: Initial Load:** The required filter and ifmap data are not yet present in the SPAD. The sequence is LOAD (fetch filter and ifmap from external memory) → READ (transfer data from SPAD to processing units) → MAC (perform multiply-accumulate operations). This loop continues until the current kernel computation is complete, at which point the control logic transitions to the next situation.



2) **Situation 1: Ifmap Update (Partial):** The required filter is in the SPAD, but a portion of the ifmap needs to be updated. The sequence is READ (read existing filter from SPAD) → MAC (perform MAC operations) loop. When the kernel calculation count reaches 5 (or a predefined threshold), it indicates that 3 new ifmaps need to be loaded into the SPAD. The state transitions to MAC → LOAD (fetch the 3 new ifmaps) → READ (transfer the new ifmaps to processing units). This loop continues until the current kernel computation is complete, then transitions to the next situation.



3) **Situation 2: Filter Map Update:** A new filter map needs to be loaded. The sequence is LOAD (fetch the new filter map) → READ (transfer the new filter map to processing units) → MAC (perform MAC operations). This loop continues until the current

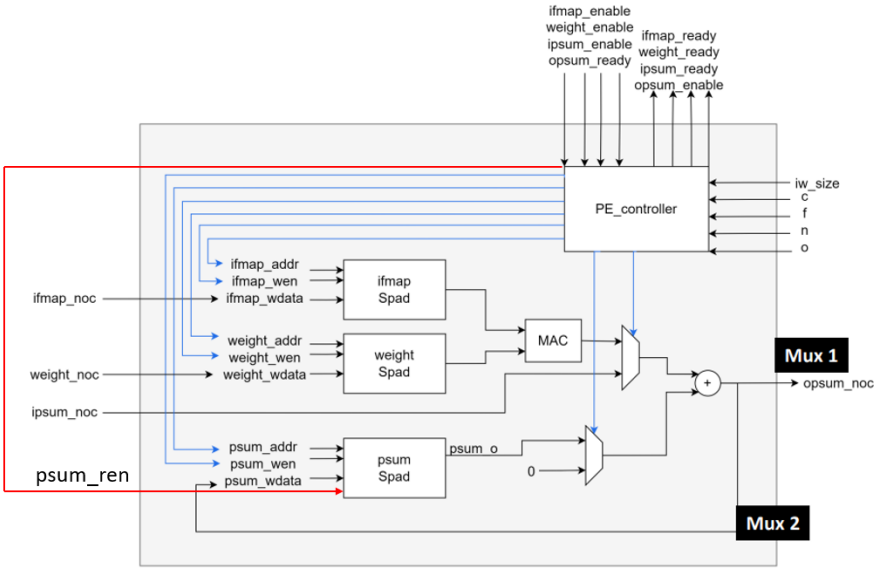
kernel computation is complete, then transitions to Situation 5 (no load).

4) **Situation 3: Ifmap Update (Full):** A completely new ifmap needs to be loaded. The sequence is LOAD (fetch the new ifmap) → READ (transfer the new ifmap to processing units) → MAC (perform MAC operations). This loop continues until the current kernel computation is complete, then transitions to the next situation.

5) **Situation 4: Ifmap Update (Partial - Alternative Condition):** Similar to Situation 1, the filter is in the SPAD, but a portion of the ifmap needs to be updated. The sequence is READ (read existing filter from SPAD) → MAC (perform MAC operations) loop. When the kernel calculation count reaches 5 (or a predefined threshold), it indicates that 3 new ifmaps need to be loaded into the SPAD. The state transitions to MAC → LOAD (fetch the 3 new ifmaps) → READ (transfer the new ifmaps to processing units). This loop continues until the current kernel computation is complete, then transitions to the next situation.

6) **Situation 5: No Load:** Both the required filter and ifmap data are already present in the SPAD. No loading is required. The operations likely proceed with READ and MAC operations until the current kernel is complete.

b) PE Architecture: The PE architecture is generally similar, however, the design of the psumSPAD (partial sum Scratchpad) differs from the ifmapSPAD (input feature map Scratchpad) and weightSPAD (weight Scratchpad) due to the need for simultaneous read and write operations. The psumSPAD requires an additional read enable signal to prevent data hazards.

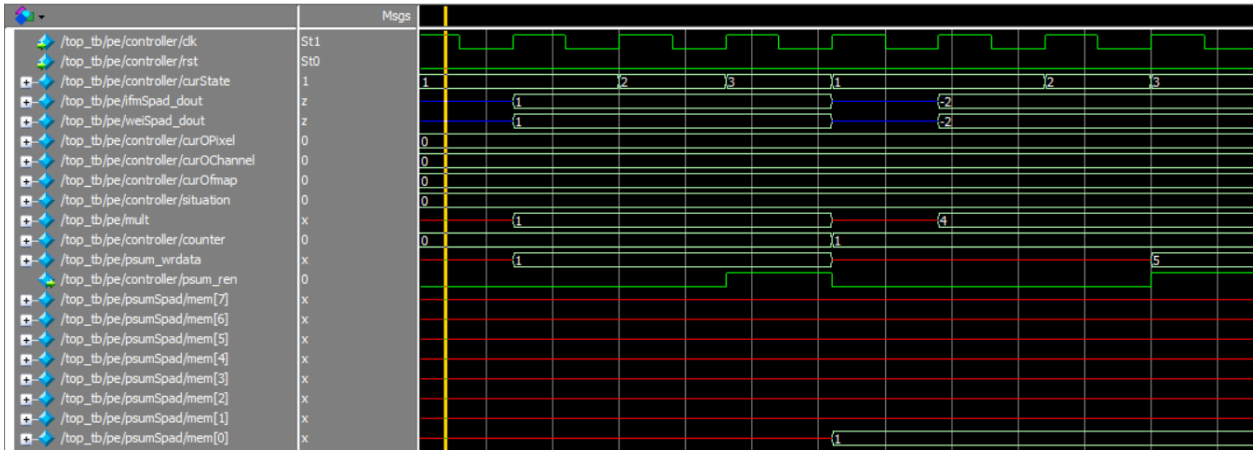


3. Explain the result by waveform.

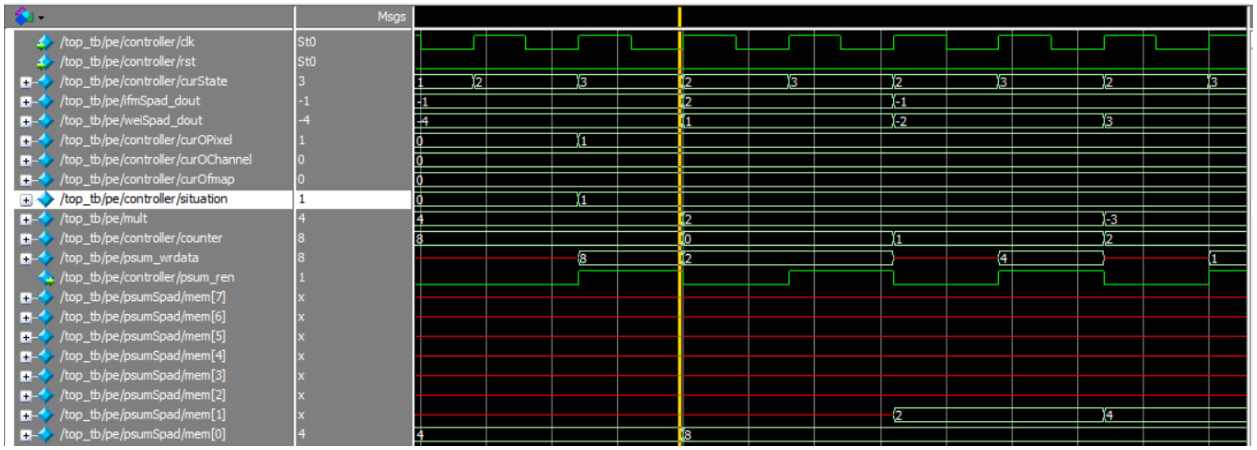
State 0: INIT
State 1: LOAD
State 2: READ
State 3: MAC
State 4: ADDPSUM
State 5: DONE

1) **Situation 0 (Initial Data Load):** During the LOAD state, both ifmSpad_dout (output from the input feature map SPAD) and weiSpad_dout (output from the weight SPAD) are in a high-impedance (high Z) state. This means that no data is being read from these SPADs yet. The loaded values are read out during the subsequent READ state. The multiplication and accumulation (MAC) operation occurs in the next cycle, during which the multiplier output

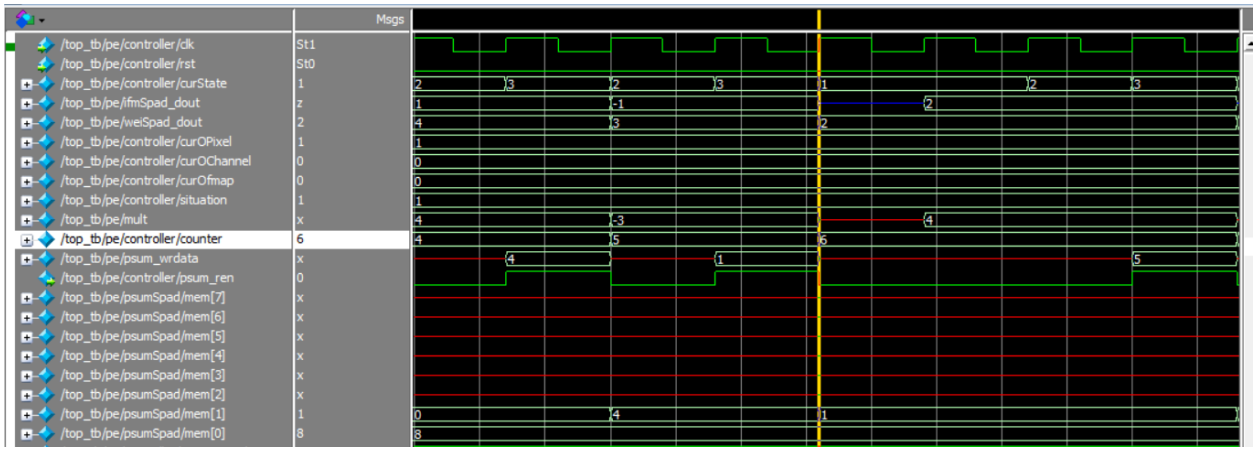
is stable. The resulting psum_wrdata (partial sum write data) is then written into the psumSpad (partial sum SPAD) in the following cycle.



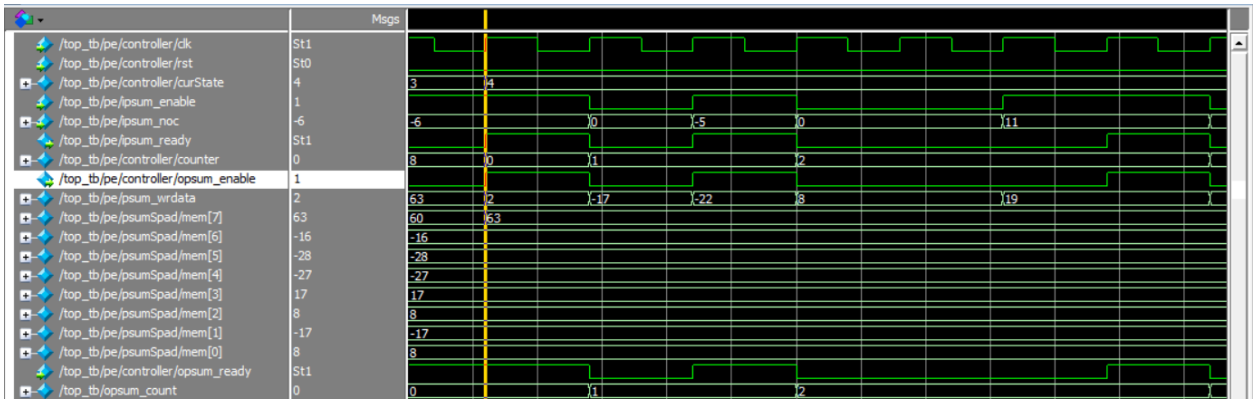
2) **Situation 1 (Ifmap Update - Partial):** When transitioning from Situation 0 to Situation 1, the state changes from MAC to READ because the current filter and weight are already present in the SPADs.



When the calculation counter reaches 5 (or a predefined threshold), the next state after MAC should be LOAD to fetch new ifmaps.



3) **ADDPSUM State (Final Accumulation):** After all kernel computations are completed, the state machine transitions to the ADDPSUM state to accumulate the incoming ipsum (intermediate partial sum) values. The psum_wrdata represents the correct output data only when opsum_enable (output sum enable) is asserted.



4. Q&A: From the fig(a) and fig(b), please try to explain (can draw the dataflow to explain)

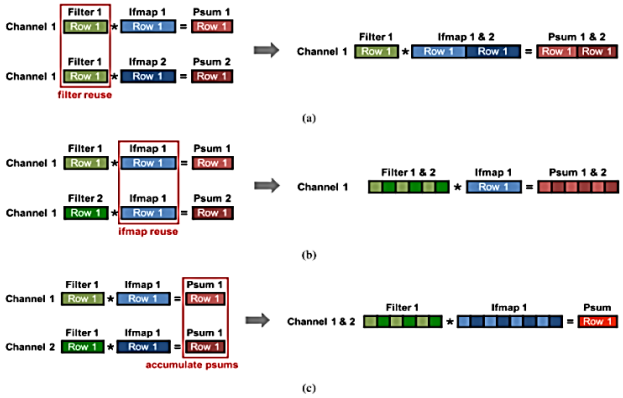


Fig. 6. Handling the dimensions beyond 2-D in each PE by (a) concatenating the ifmap rows, each PE can process multiple 1-D primitives with different ifmaps and reuse the same filter row and (b) time interleaving the filter rows, each PE can process multiple 1-D primitives with different filters and reuse the same ifmap row. (c) By time interleaving the filter and ifmap rows, each PE can process multiple 1-D primitives from different channels and accumulate the psums together.

The dimensionality of Deep Neural Network (DNN) computations extends beyond two dimensions due to factors such as batch size (N), number of channels (C), and number of filters (M). This necessitates strategies for efficient computation and memory management.

- **(a) Filter Reuse:** When the batch size (N) is greater than 1, filter reuse is employed. This means that the same filter weights are used for different input feature maps (ifmaps) within the batch. This re-utilization of filter weights reduces redundant computations and memory accesses, improving efficiency.
- **(b) Ifmap Reuse:** When the number of filters (M) is greater than 1, ifmap reuse is employed. This signifies that the same ifmap is repeatedly used with multiple different filters. This strategy minimizes the number of ifmap loads from memory, again contributing to enhanced computational efficiency.

5. Screenshot your simulation result and report your total cycle.

a. Total cycle = ___250___

b. simulation result

```
#
# result=   2, answer=   2, Correct!
# result= -22, answer= -22, Correct!
# result=  19, answer=  19, Correct!
# result=  18, answer=  18, Correct!
# result= -20, answer= -20, Correct!
# result= -18, answer= -18, Correct!
# result=   4, answer=   4, Correct!
# result=  41, answer=  41, Correct!
#
#
#
#         ****************************
#         **                        **       |__||
#         **  Congratulations !!    **      / 0.0  |
#         **                        **    /_____   |
#         **  Simulation PASS!!     **   /^ ^ ^ \  |
#         **                        **  |^ ^ ^ ^ |w|
#         ****************************   \m___m__|_|
#
#
# total cycle =                250
#
```