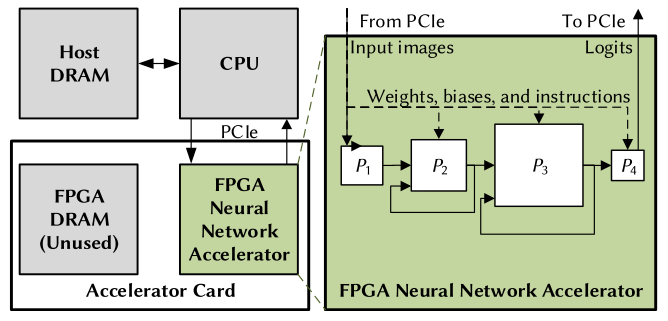
Compute-Efficient Neural-Network Acceleration

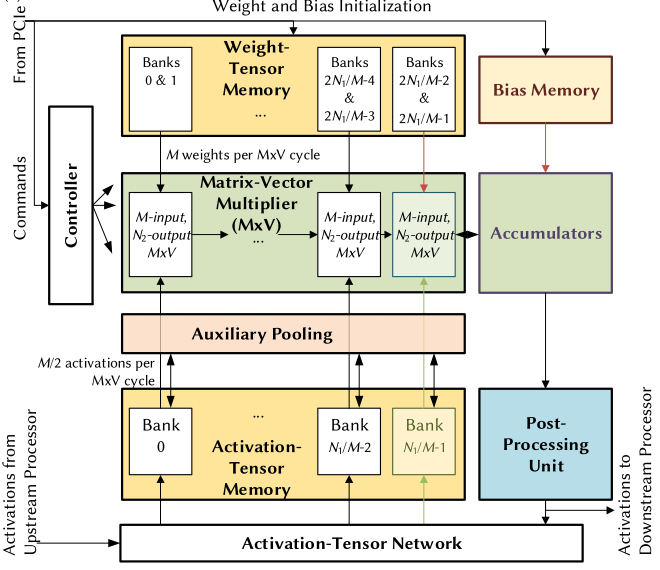
Xilinx FPGA2019

1. Contribution
   1. Implement a convolution neural network accelerator for image classification on VCU1525 card which houses a Xilinx VU9P FPGA.
   2. Describe a schedule that keeps compute utilization high.
2. Design Overview
   1. Accelerator processor chain
      1. The accelerator is a pipeline of four processors, labeled P1 to P4 in the figure. P1, P2 and P3 are convolution processors whereas P4 computes only fully-connected layers(it lacks a pointer generator to read convolution input patches from memory).
      2. The design does not use any DRAM on the accelerator card, this accelerator accepts commands, weights and input images from CPU through PCIe.



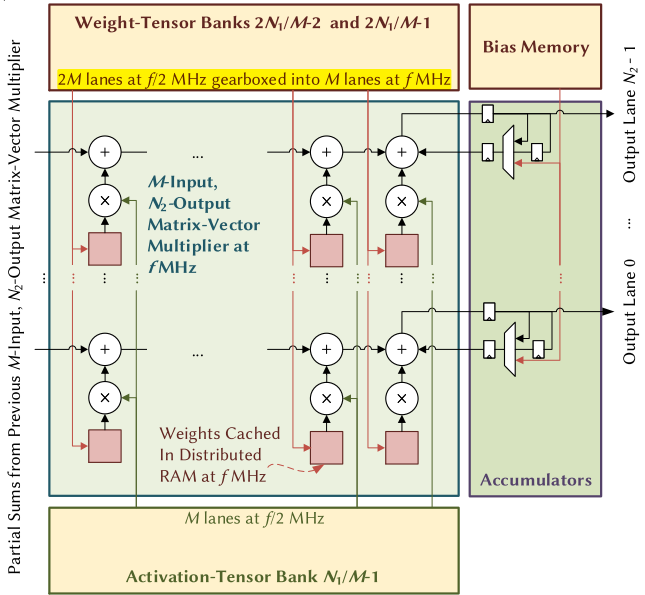
Accelerator

* 1. Processor architecture
     1. Each processor consists of a controller, weight-memory, activation-memory, a block-floating-point matrix-vector multiplier, a non-linear unit and an activation-tensor network that organizes activation tensors in SRAMs.



Processor

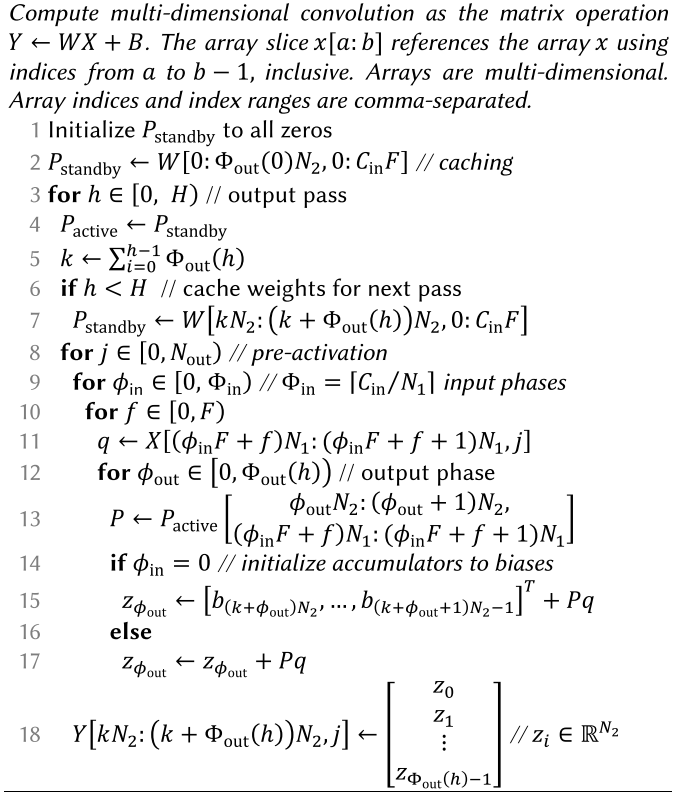
* + 1. Matrix-vector multiplier
       1. The matrix-vector is a N1 x N2 crossbar array of processing elements(N1 input lanes and N2 output lanes), segmented into M x N2 array slices that are aligned to the weight memory banks and activation memory banks.
       2. Each processing element has a 128-byte weight cache and a multiplier-add unit with a full-precision adder. The weight cache is double-buffered to accept new weights from one weight memory bank to one buffer while the matrix-vector multiplier reads from the other buffer.



The last M input lanes of the matrix-vector multiplier

* + - 1. Encode tensor elements as block-floating-point numbers. A block-floating-point number in a block of numbers is . is the block exponent and is an integer. The design rescales activations between layers with arithmetic shifting. Use GoogLeNet as an example, the design encode weights as signed 8-bit integers sharing the exponent -7, encode output activation as unsigned 8-bit integers with per-layer block exponents in the range [-2, 5]. The matrix-vector multiplier accumulates dot products in full-precision(int48).
    1. Weight and activation memory
       1. Channelize the activation tensor memory by dividing it into banks, each covering M of the N1 input lanes. Also partition the weight-tensor memory into banks similarly.
    2. Activation-tensor network
       1. The activation-tensor network delivers activations from post-processing unit to the activation-tensor memory. This network is usually inter-processor but sometimes also intra-processor.

1. Scheduling
   1. Algorithm
      1. Use block-matrix multiplication algorithm. Matrix-vector multipliers array caches a sub-matrix that includes all weights from a subset of output channels. With this sub-matrix, the array computes all corresponding pre-activations in one output pass. A weight stays with the same multiplier.



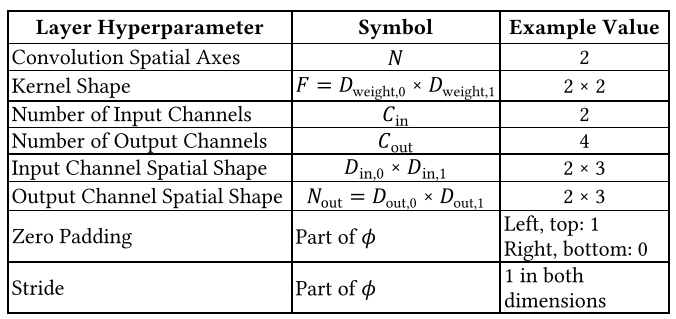
Scheduling Algorithm

* + 1. , a small integer, is the number of output phases in the output pass. The design constrain to be either two or three. is the number of input phases. In one output pass, it consumes the weight sub-matrix. It does so in output-phase-major order, therefore reusing each input activation vector on the input broadcast bus times.
    2. In one cycle, it computes:



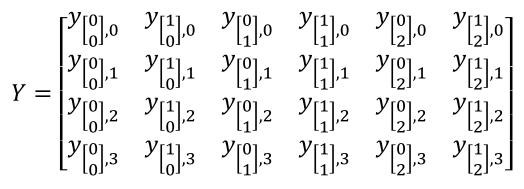
Matrix form of convolution

* 1. Example
     1. Convolution example

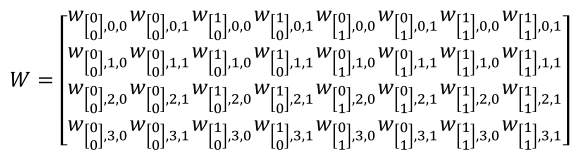


Parameters of the convolution

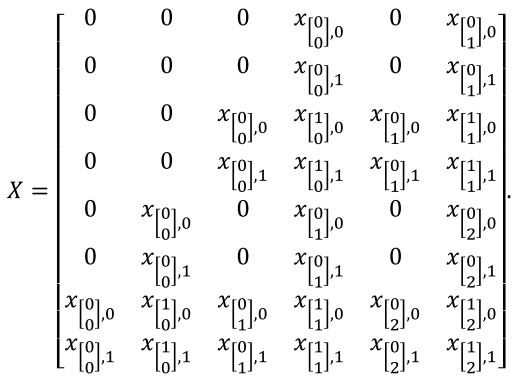
* + 1. Data arrangement



Matrix of output activations

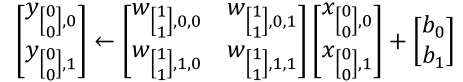


Matrix of weights

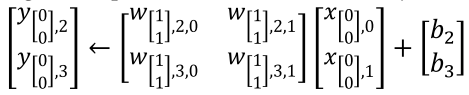


Matrix of input activations

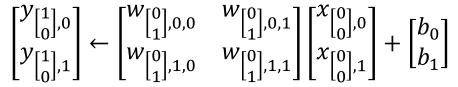
* + 1. Implement the convolution on matrix multipliers array with 2 input lanes and 2 output lanes. We need only 1 output pass and the output pass has 2 output phases. There is 1 input phase.
    2. The matrix multipliers array skips zero padding. The process of computing first 2 columns of is described below.



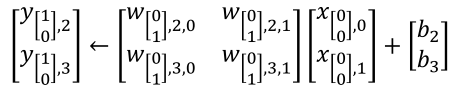
Cycle 1



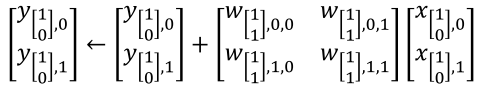
Cycle 2



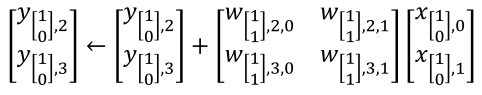
Cycle 3



Cycle 4

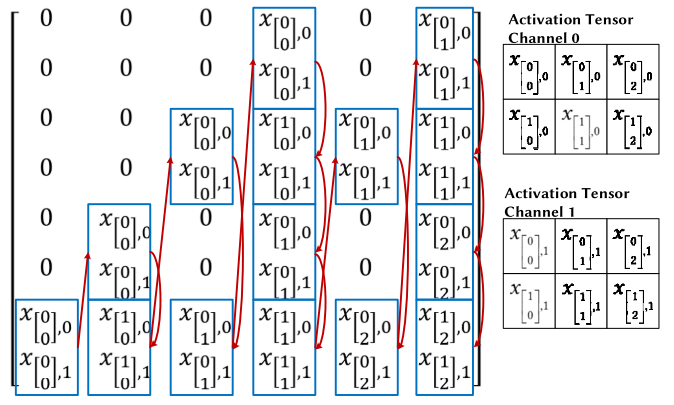


Cycle 5



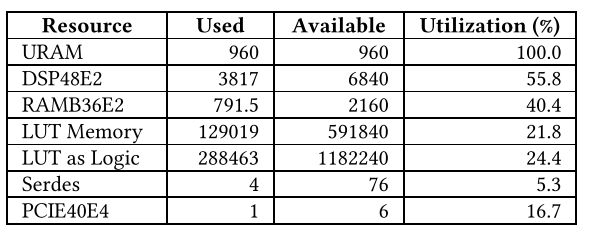
Cycle 6

* + 1. The N1 elements read from the input matrix in the same cycle share the same spatial coordinates.

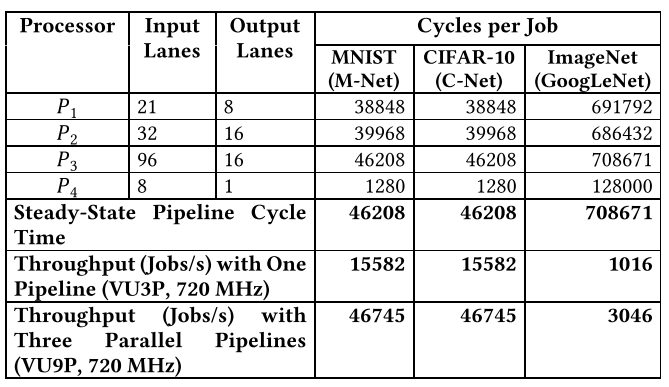


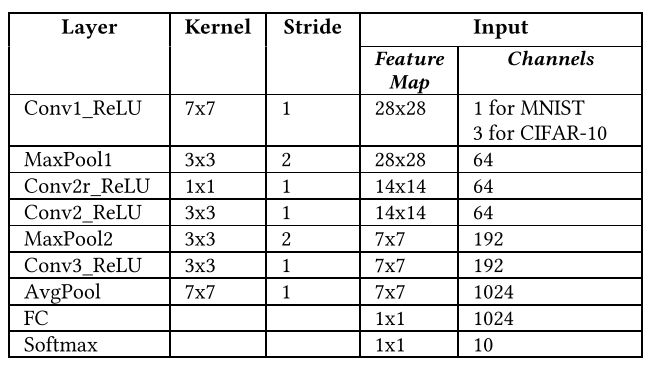
Activation read ordering

1. Result
   1. Resource utilization



* 1. Latency





Architecture of M-Net and C-Net