Design and Evaluation of Accelerator Organizations for Binarized Neural Networks

Somar Iskif

September 19, 2022

Department of Computer Science
TU Dortmund, Germany

Contents

Introduction

- The neural network (NN)
  - Computer Model $\Leftrightarrow$ Human Brain
  - Consist of:
    - Neurons
    - Layers of Nodes
    - Weights
  - Face Recognition
  - Pattern Recognition
  - Speech Recognition

Motivation

- Training a neural network
  - Feeding it some data and the correct answer
  - NN predict the correct answer for future data
- Example:
  - Train a NN to recognize objects in images (Dogs)
    - The NN sees thousands of dogs and finds a way to generalize a dog's image.

Motivation

- How to conclude all types of dogs?
  - Train the NN with all Dog's types
  - New Data
  - Retrain the neural network again
    - Time
    - Resources (Energy, Register area)
  - BNN architecture addresses these issues
  - Design and train neural networks at a larger scale

BNN

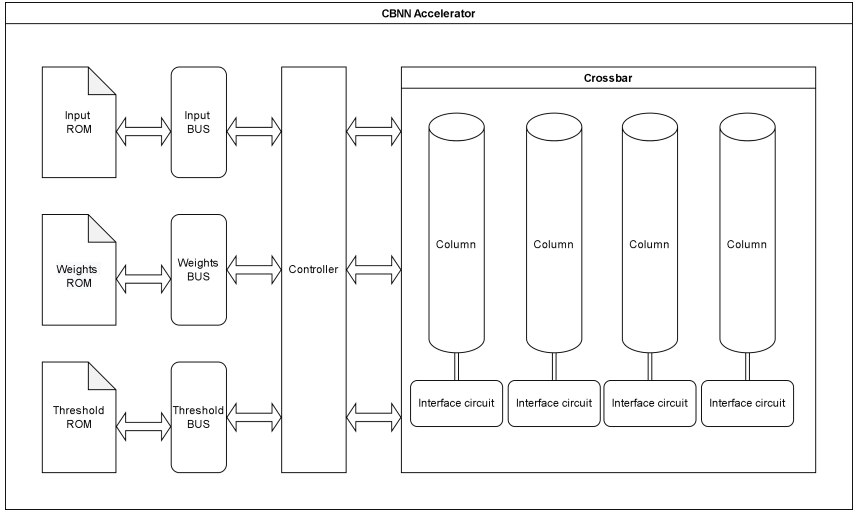
- Instead of using floating-point or integer numbers, BNN uses just one Bit('0' or '1')
- Fantastic opportunity for the realization of compact and energy efficient inference hardware
- The binary representation can be multiplied quickly and easily.
- Disadvantage of BNN, it is less accurate

BNN

- Internally, the BNN works with 0s and 1s.
- 0 represents a -1 in the NN and 1 represents 1.

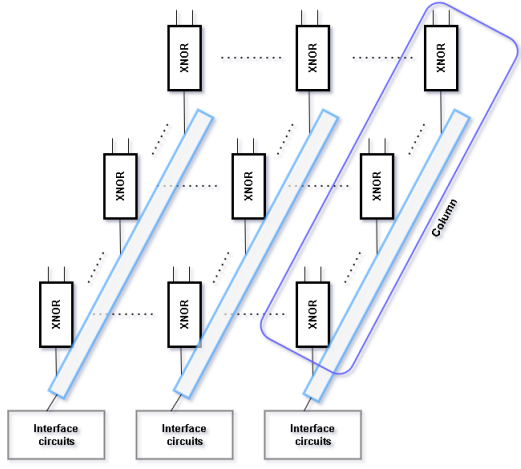
internal value	real value
0	-1
1	1

CBNNA Design



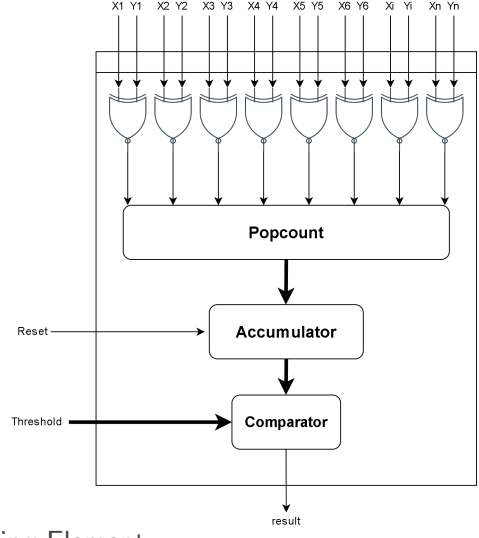
Design of Crossbar-based BNNA

- One Column processes the workload of one neuron
- m Columns
- n XNOR Gates
- $(m \times n) = (column \times XNOR)$
- Interface Circuit



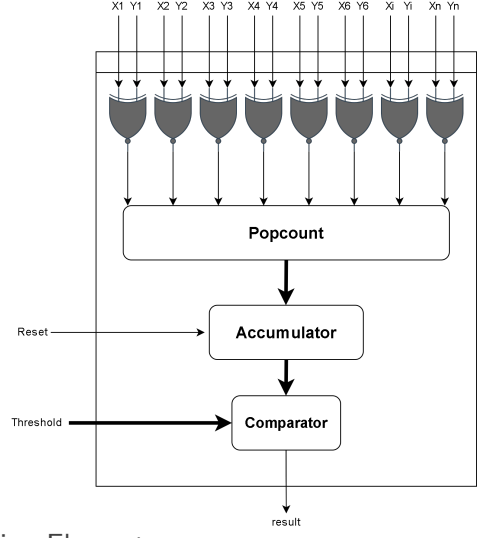
Column and Interface Circuits (Processing Element)

- Column
  - Xnor gates
- Interface Circuits (IC)
  - Popcount
  - Accumulator
  - Comparator



Processing Element

- Processing Element (PE) includes 4 modules:
  - Xnor gates
  - Popcount
  - Accumulator
  - Comparator



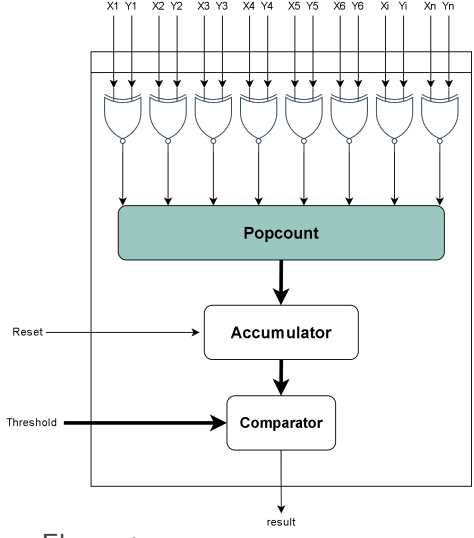
XNOR

- The XNOR gate is one of the two components responsible for vector multiplication.
- One XNOR gate multiplies one row of the two given vectors.
- x XNORS gates can multiply two vectors of size x.

$$XNOR(\begin{pmatrix}0\\0\\1\\1\end{pmatrix}, \begin{pmatrix}0\\1\\0\\1\end{pmatrix}) = \begin{pmatrix}1\\0\\0\\1\end{pmatrix} \Leftrightarrow \begin{pmatrix}-1\\-1\\1\\1\end{pmatrix} \times \begin{pmatrix}-1\\1\\-1\\1\end{pmatrix} = \begin{pmatrix}1\\-1\\-1\\1\end{pmatrix}$$

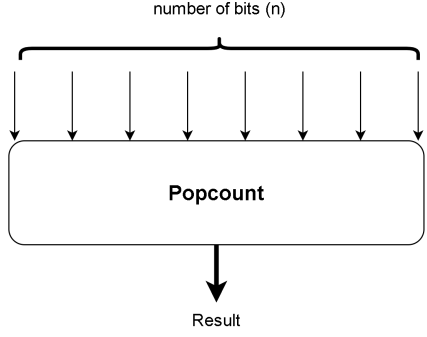
Processing Element

- Processing Element (PE) includes 4 modules:
  - Xnor gates
  - Popcount
  - Accumulator
  - Comparator



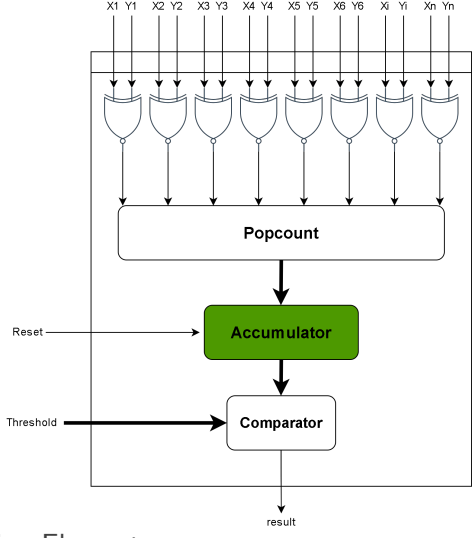
Popcount

- Counting 1's in a bit stream
- Result vector of the XNOR's gates
- Result can not be negative
- Outputs the result to the Accumulator



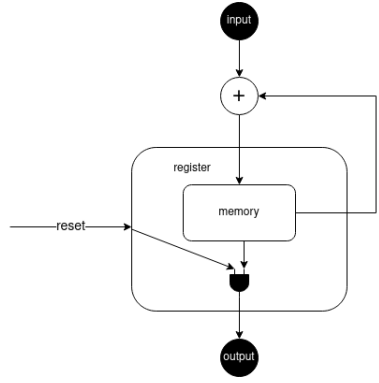
Processing Element

- Processing Element (PE) includes 4 modules:
  - Xnor gates
  - Popcount
  - Accumulator
  - Comparator



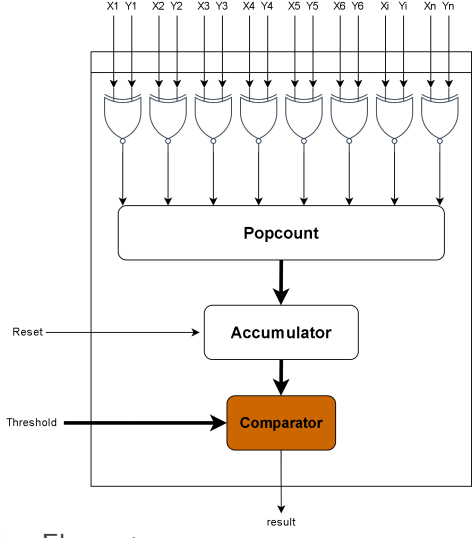
Accumulator

- The intermediate Accumulator sums up the results of the Popcount module and therefore the scalar product of the two vectors.
- If the Accumulator receives a reset signal from the controller, it outputs the value it's currently holding and resets its value to 0.



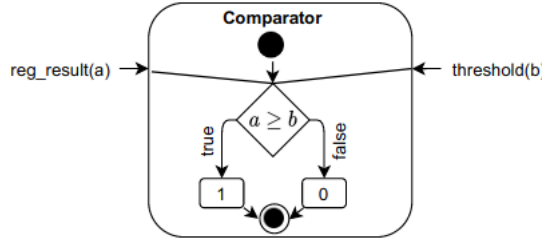
Processing Element

- Processing Element (PE) includes 4 modules:
  - Xnor gates
  - Popcount
  - Accumulator
  - Comparator

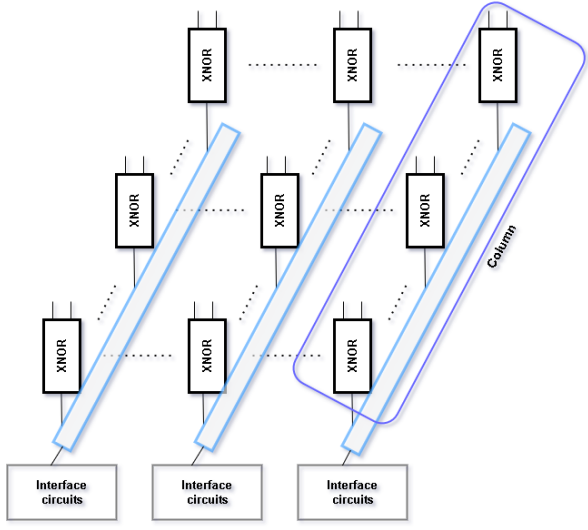


Comparator

- Compares output of register with threshold
  - '1' if output of register $\geq$ threshold
  - Else '0'



CBNNA



CBNNA: Processing Element

NN Workload

$$W_{(\alpha,\beta)} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,\beta} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,\beta} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\alpha,1} & w_{\alpha,2} & \cdots & w_{\alpha,\beta} \end{pmatrix} \begin{matrix} \\ \\ \\ \downarrow \alpha \end{matrix} \quad \overrightarrow{\beta}$$

$$I_{(\gamma,\delta)} = \begin{pmatrix} i_{1,1} & i_{1,2} & \cdots & i_{1,\delta} \\ i_{2,1} & i_{2,2} & \cdots & i_{2,\delta} \\ \vdots & \vdots & \ddots & \vdots \\ i_{\gamma,1} & i_{\gamma,2} & \cdots & i_{\gamma,\delta} \end{pmatrix} \begin{matrix} \\ \\ \\ \downarrow \gamma \end{matrix} \quad \overrightarrow{\delta}$$

- NN workload can be converted to this Notation ($\beta = \gamma$)
- Big numbers for alpha, beta and delta => big Matrices
- One Processing Element is often not enough or too slow
- Using Crossbars to process the Computation

Challenge and Solutions

- P: $\beta >> n =>$ "Crossbar array is not large enough to hold the weights of one convolution layer" [1]

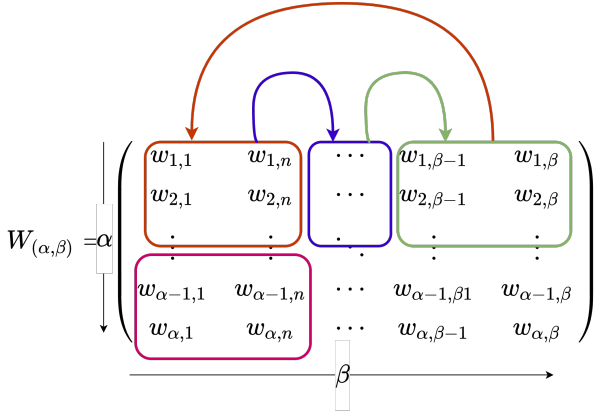
  How to partition the workload and map Computations to the crossbar array?
  - L1: "Partition computations and accumulate intermediate results (i.e., partial sums)" [1]
  - L2: Map the workload with clever methods

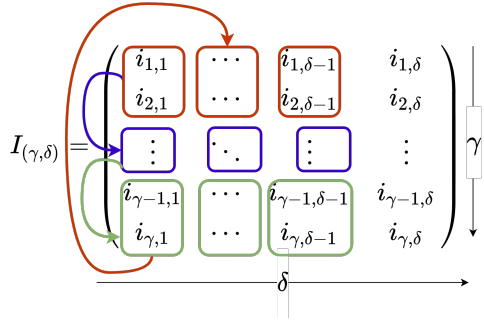
Methods

- Different ways of partitioning and mapping computations:
  - Vertical Move
  - Strided Move

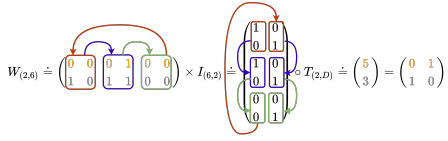
  They can have a significant impact on:
    - Performance
    - Energy

Vertical Move





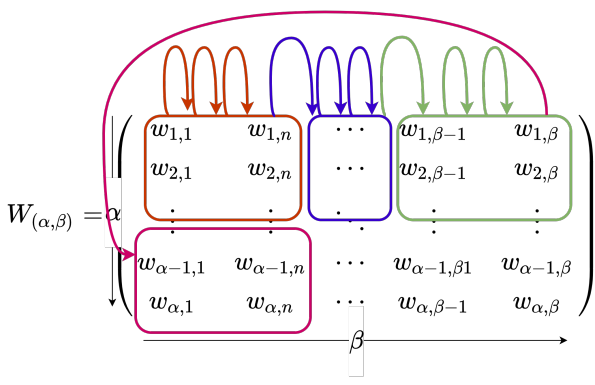
VM Example

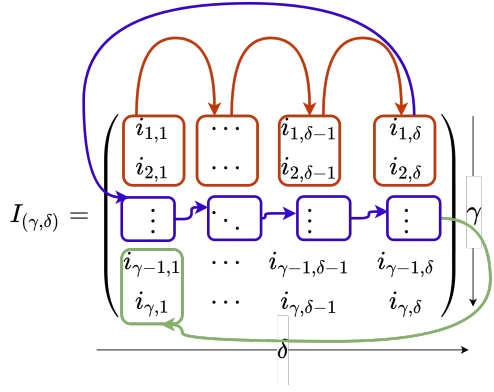

$W_{(2,6)} \doteq \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \times I_{(6,2)} \doteq$
$\circ T_{(2,D)} \doteq \begin{pmatrix} 5 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
First crossbar's column process first row
Second crossbar's column process second row
$(0 \quad 0) \oplus \begin{pmatrix} 1 \\ 0 \end{pmatrix} = popcount\,(0 \quad 1) = 1$
$(0 \quad 1) \oplus \begin{pmatrix} 1 \\ 0 \end{pmatrix} = popcount\,(0 \quad 0) = 0$
$(0 \quad 0) \oplus \begin{pmatrix} 0 \\ 0 \end{pmatrix} = popcount\,(1 \quad 1) = 2$
$accumulator\,(1, \quad 0, \quad 2) = 3$
$comparator\,(3)\,(5) = 0$
$(1 \quad 0) \oplus \begin{pmatrix} 1 \\ 0 \end{pmatrix} = popcount\,(1 \quad 1) = 2$
$(1 \quad 1) \oplus \begin{pmatrix} 1 \\ 0 \end{pmatrix} = popcount\,(1 \quad 0) = 1$
$(0 \quad 0) \oplus \begin{pmatrix} 0 \\ 0 \end{pmatrix} = popcount\,(1 \quad 1) = 2$
$accumulator\,(2, \quad 1, \quad 2) = 5$
$comparator\,(5)\,(5) = 1$
$(0 \quad 0) \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = popcount\,(1 \quad 0) = 1$
$(0 \quad 1) \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = popcount\,(1 \quad 1) = 2$
$(0 \quad 0) \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = popcount\,(1 \quad 0) = 1$
$accumulator\,(1, \quad 2, \quad 1) = 4$
$comparator\,(4)\,(3) = 1$
$(1 \quad 0) \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = popcount\,(0 \quad 0) = 0$
$(1 \quad 1) \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = popcount\,(0 \quad 1) = 1$
$(0 \quad 0) \oplus \begin{pmatrix} 0 \\ 1 \end{pmatrix} = popcount\,(1 \quad 0) = 1$
$accumulator\,(0, \quad 1, \quad 1) = 2$
$comparator\,(2)\,(3) = 0$

Vertical Move

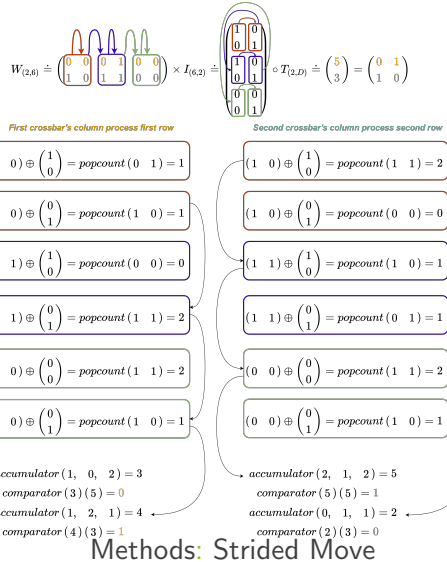
- The crossbar array needs to be re-programmed for every move
  - Increasing Time.
  - Increasing Energy.
- No Need to store Intermediate results from the previous Tile
  - Reducing the register area.

Strided Move





SM Example


First crossbar's column process first row
Second crossbar's column process second row
accumulator (1, 0, 2) = 3
comparator (3) (5) = 0
accumulator (1, 2, 1) = 4
comparator (4) (3) = 1
accumulator (2, 1, 2) = 5
comparator (5) (5) = 1
accumulator (0, 1, 1) = 2
comparator (2) (3) = 0

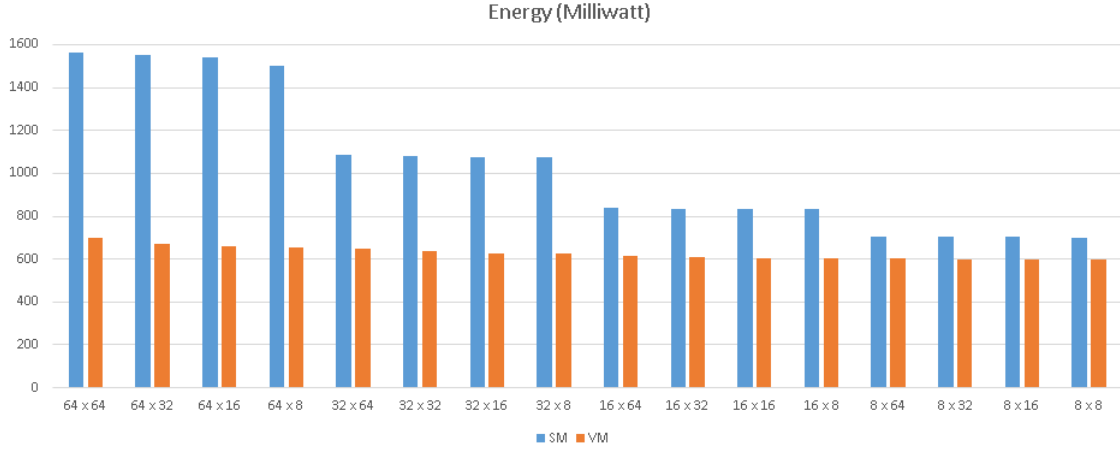
Strided Move

- Re-programming the crossbar array is not required
  - Reducing Energy.
- The weights stored in the crossbar array can be reused
  - Reducing Time.
- Intermediate results from the previous tile need to be stored in its exclusive registers
  - Increasing the register area.

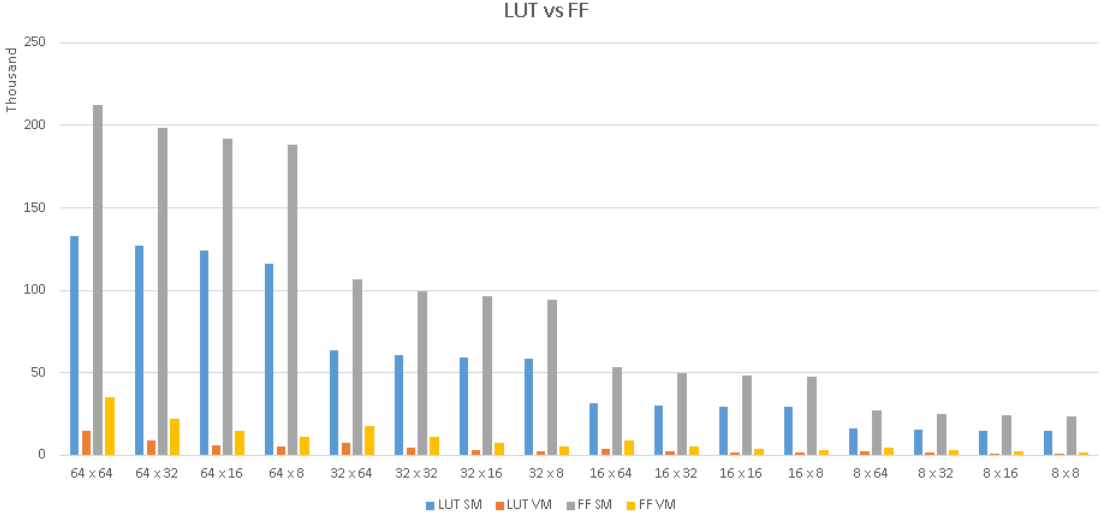
Evaluation Plan

- Measure and Compare execution times of different Configurations and Input's length
- Compare **Latency**, **Energy** and **Resource Usage** ("LUTs" and "FFs") of the following Methods:
  - **Strided Move**
  - **Vertical Move**
- What is better for **FPGA**, Vertical or Strided Move ?

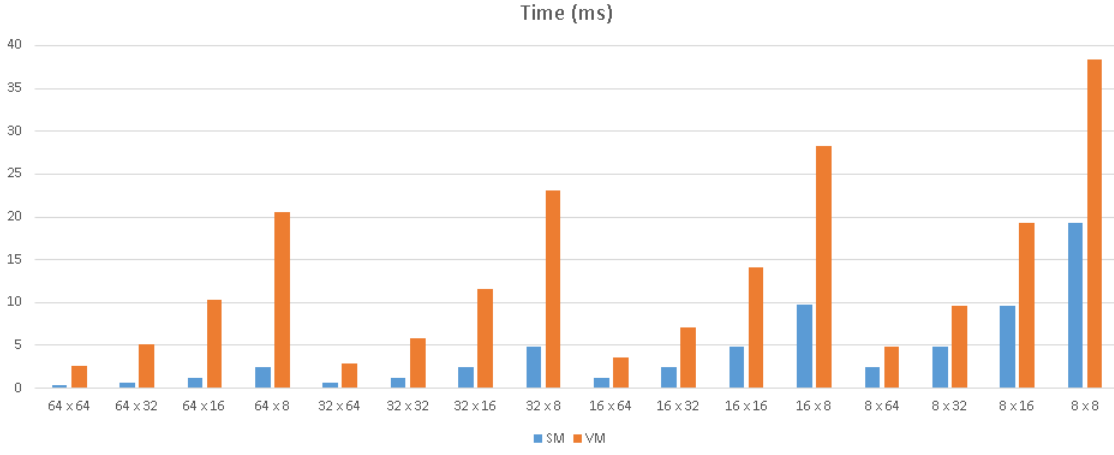
Energy Results



LUTs and FFs Results



Time Results



Evaluation Results

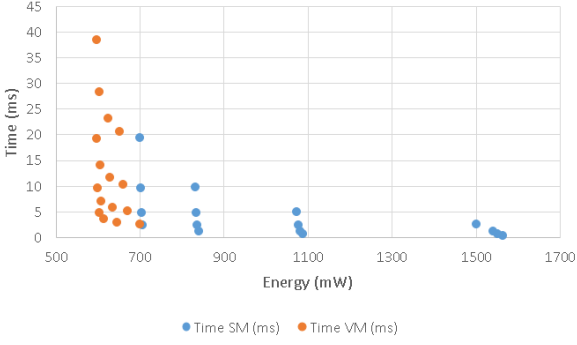


Figure: Energy vs. Time

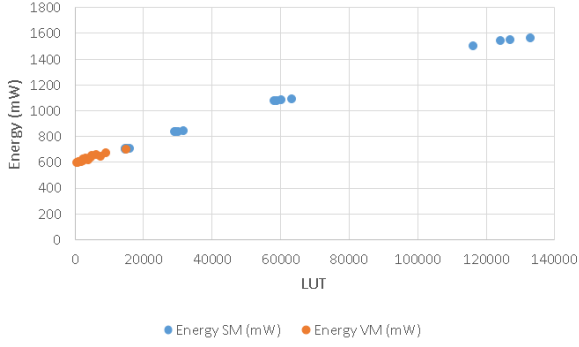


Figure: Energy vs. LUT

Evaluation Results

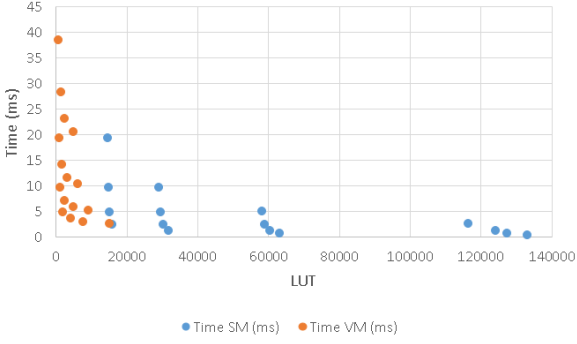


Figure: Time vs. LUT

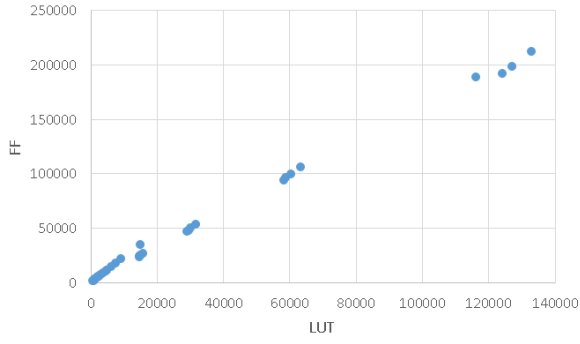


Figure: FF vs. LUT

Summary

- Define the Demands, Hardware Requirements
  - Time-Critical
  - Resource Efficiency
    - Energy
    - Register Area
- Available Resources?

Summary

- Vertical Move
  - Low Energy
  - Low Register Area
  - Slow Execution Time

Reason: Only m Registers, wait until the new weights are loaded

- Strided Move
  - High Energy
  - High Register Area
  - Fast Execution Time

Reason: High write energy, more LUTs and flip-flops for the δ registers, reuse the weights

Summary

- (Execution Time $\ggg$ Resource Usage) $\Rightarrow$ Strided Move
- (Execution Time $\lll$ Resource Usage) $\Rightarrow$ Vertical Move.
- Many FPGA types and different capacity
  - Not be able to run the strided move
    - lack of resources
    - Exceed the maximum energy

  $\Rightarrow$ VM better than SM for the FPGA's in general

Further Optimization

- Full Binary Neural Network
- Horizontal Move
- Systolic Array
- Approximate Computing (Stochastic Computing)
- Several Crossbars in Parallel

Github Repo

- The CBNNA Design is here available:
  `https://github.com/somar0/Crossbar-Design`

- **Thank you all for listening.**

[1] Chen X, Yin X, Niemier M, Hu XS. Design and optimization of FeFET-based crossbars for binary convolution neural networks. In 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE) 2018 Mar 19 (pp. 1205-1210). IEEE.

[2] Liang S, Yin S, Liu L, Luk W, Wei S. FP-BNN: Binarized neural network on FPGA. Neurocomputing. 2018 Jan 31;275:1072-86.

[3] Hirtzlin T, Penkovsky B, Bocquet M, Klein JO, Portal JM, Querlioz D. Stochastic computing for hardware implementation of binarized neural networks. IEEE Access. 2019 Jun 5;7:76394-403.

[4] Umuroglu Y, Fraser NJ, Gambardella G, Blott M, Leong P, Jahre M, Vissers K. Finn: A framework for fast, scalable binarized neural network inference. InProceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays 2017 Feb 22 (pp. 65-74).