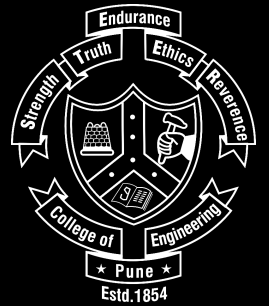
HDL Implementation of SNN for Event-based Dataset

Tarun Kumar Allamsetty - 111807029
Juhi Wani - 111807044
Rushikesh Gherde - 111807014



Aims & Objectives

Aim :

1. Training of an event based dataset on a Convolutional Spiking Neural Network and analysis of performance with change in hyper-parameters.
2. HDL implementation of the SNN and analysis of resource utilization.

Introduction

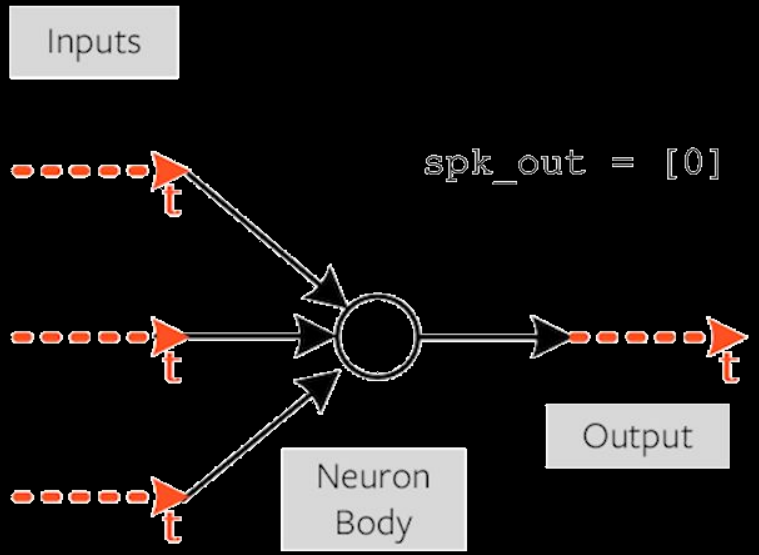
Introduction

Neuromorphic computing deals with modelling the elements in a system in close accordance with units present in the mammalian brain and nervous system. It functions over a network of spiking neurons.

	Neurogrid	BrainScaleS	TrueNorth	HiAER-IFAT	SpiNNaker
Technology	Analog	Analog	Digital	Analog	Digital
Feature size	180 nm	180 nm	28 nm	130 nm	130 nm
Chips	16	352	16	16	48
Neurons	1 M	200 k	16 M	1 M	768 k
Synapses	4 G	40 M	4 G	1 G	768 M
Power	3 W	500 W	3.2 W	40 W	80 W
Interconnect	Tree-multicast	Hierarchical	2D mesh-unicast	Hierarchical	2D mesh-unicast
Neuron model	Adaptive quadratic IF	Adaptive exponential IF	Configurable LIF	2-compartment LIF	Programmable

Background

Spiking Neural Network



Modelling SNN

SNN neurons are based on mathematical representations of biological neurons.

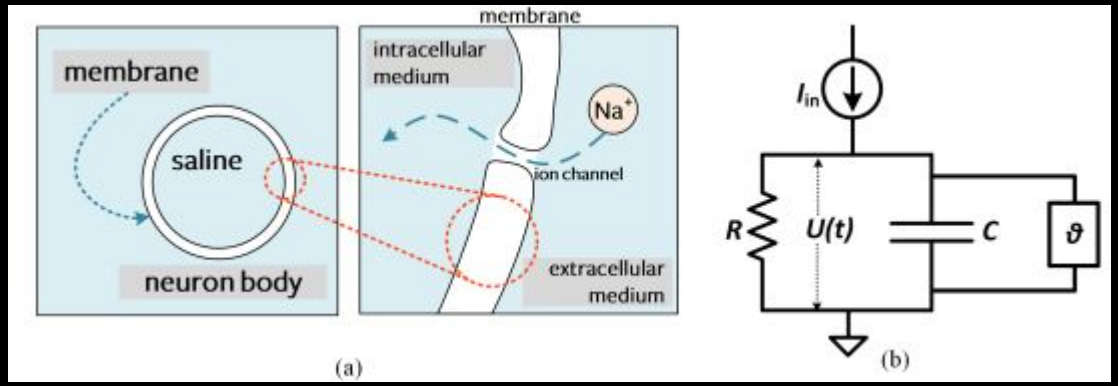
To model an SNN neuron, there are two major sets of approaches

1. **Conductance based models** describe how action potential of neuron are initiated and propagated. **E.g. Hodgkin-Huxley model, Izhikevich model**

2. **Threshold based models** generate an impulse at a certain threshold. **E.g Leaky Integrate and Fire (LIF)**. LIF is the most popular model used for SNN neuron.

In this project LIF is used to model SNN.

Leaky Integrate-and-Fire (LIF)



(a) Membrane of neuron (b) LIF circuit model

LIF can be modelled as a circuit. The differential equation RC circuit in Figure is

$$\tau \frac{dU(t)}{dt} = -U(t) + I_{in}(t)R$$

Solution of above equation for constant current is

$$U(t) = I_{in}(t)R + [U_0 - I_{in}(t)R]e^{-\frac{t}{\tau}}$$

On applying the forward Euler method above equation reduce to

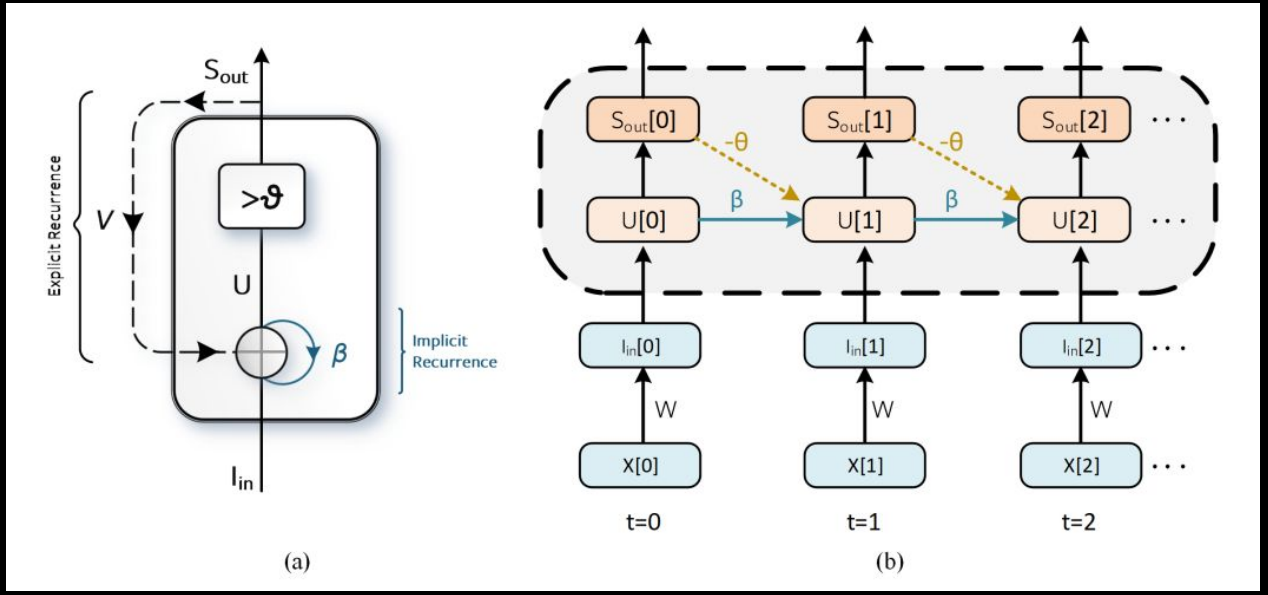
$$U[t] = \beta U[t-1] + (1-\beta)I_{in}[t]$$

Membrane potential of LIF is given as

$$U[t] = \beta U[t-1] + WX[t] - S_{out}[t-1]\theta$$

$$S_{out}[t] = \begin{cases} 1, & if\ U[t] > \theta \\ 0, & otherwise \end{cases}$$

When the membrane voltage exceed the threshold , then neuron generates a spike.



(a) Block diagram of LIF (b)Unrolled Recurrence block diagram of LIF

When a biological neuron spikes, then its membrane potential drops. This reset mechanism is captured in adjacent equation.

When spike is generated $Sout[t-1] = 1$ so θ is subtracted , else $Sout[t-1] = 0$ and no reset happens.

Learning Techniques in SNN

1. **Shadow training** - converting existing ANN to SNN
2. **Backpropagation using spikes** - training using error back propagation rule
3. **Locally learning rules** - Weight updates are a function of signals that are spatially and temporally local to the weight. E.g. STDP.

In this project back-propagation using spikes is used as a training method.

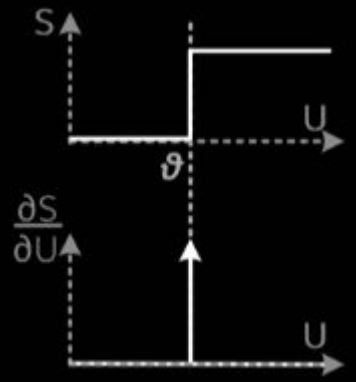
Dead Neuron Problem

Alternate representation of earlier threshold function is

$S[t] = \Theta(U[t] - U_{thresh})$ where Θ is a heavy side step function.

On taking derivative we get

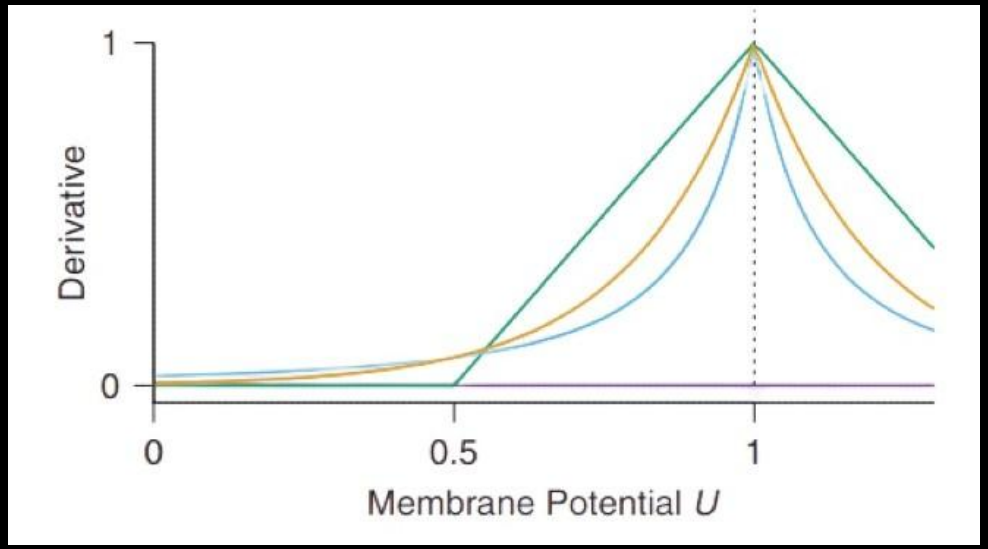
$$\frac{\partial S[t]}{\partial U} = \delta(U[t] - U_{thresh})$$



S vs U, ∂S / ∂U vs U

As seen in the figure gradient is zero most of the time. Hence no weight update take place. This is a "dead neuron" problem.
This problem is tackled by using Surrogate gradients. It replaces backward pass impulse function with functions like sigmoid, triangular, etc.
In this project we have used fast sigmoid function.

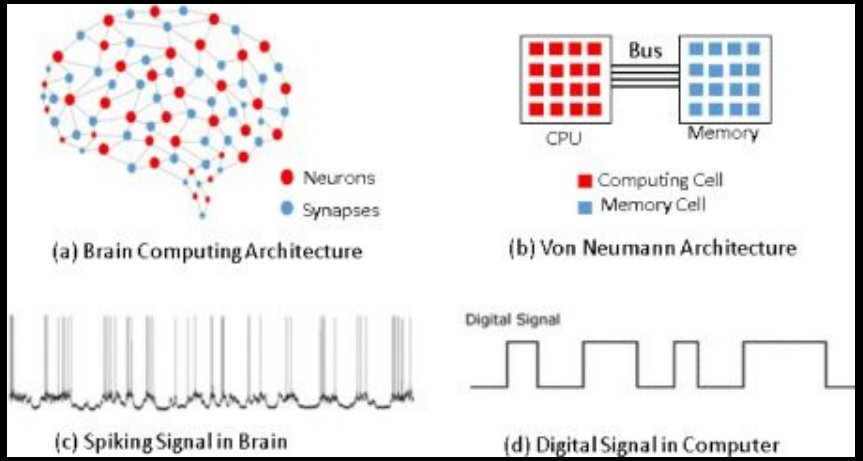
Surrogate Gradient



The step function has a zero derivative (violet) everywhere except at 0, where it is ill defined. The green (piecewise linear) , blue (derivative of a fast sigmoid), and orange (exponential) lines are examples of surrogate derivatives that have been used to train SNNs.

Motivation

- A blossoming research field in AI as well as hardware.
- Potential to address the approaching limit to Moore's Law.
- Addresses the Von neumann bottleneck problem by offering a non-Von neumann architecture.
- Less resource and power utilization compared to traditional ANNs.
- A network that is in biological consistency with the brain.



Why less power?

- Sparsification over time
    - Less Communication
- Less computation
    - Fewer memory lookups
- Cheaper Computation
    - Only sum instead of sum and multiply

Implementation and Results

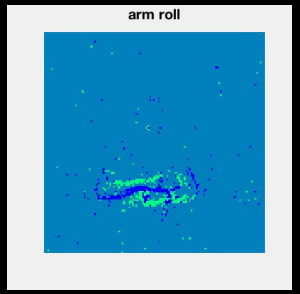
Software

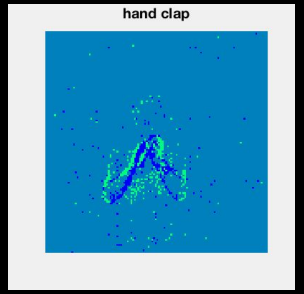
Dataset -

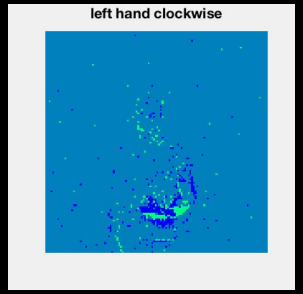
DVS128 Gesture dataset - It is Event-based Gesture dataset. The data was recorded using a DVS128. The dataset contains 11 hand gestures from 29 subjects under 3 illumination conditions. This dataset was used to build the real-time, gesture recognition system described in the CVPR 2017 paper titled "A Low Power, Fully Event-Based Gesture Recognition System." . Already split into Train and Test

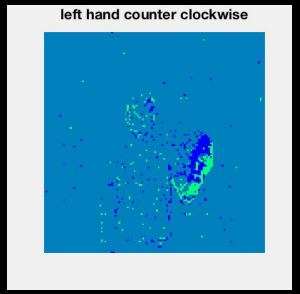
Camera sensor - The iniLabs DVS128 camera is a 128 × 128-pixel Dynamic Vision Sensor that generates events only when a pixel value changes magnitude by a user-tunable threshold [21, 22] (this work used the default settings provided by the cAER configuration software [23]). Each event encodes the spatial coordinates of a pixel reporting a change and a timestamp indicating when that change happened. The device offers a high dynamic range (120 dB) and a typical and maximum event rate of 100K and 1M events/sec.

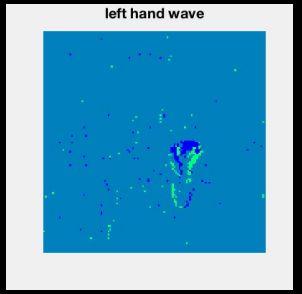
DVS128 Gesture dataset-

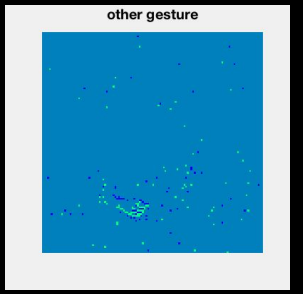


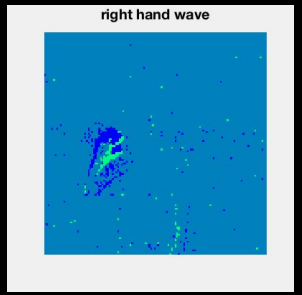


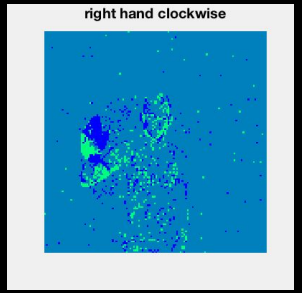


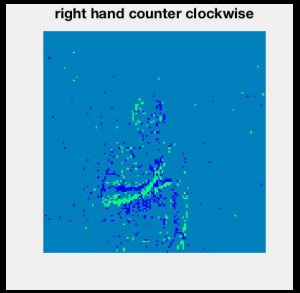


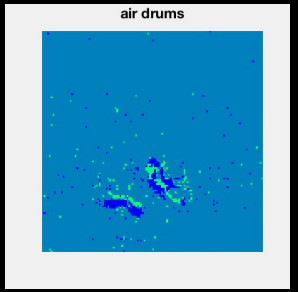


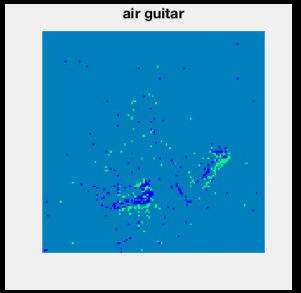








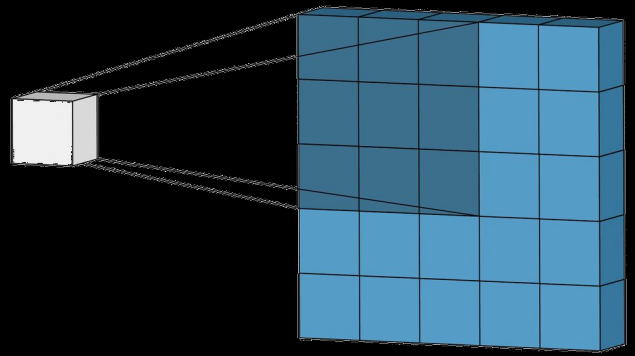


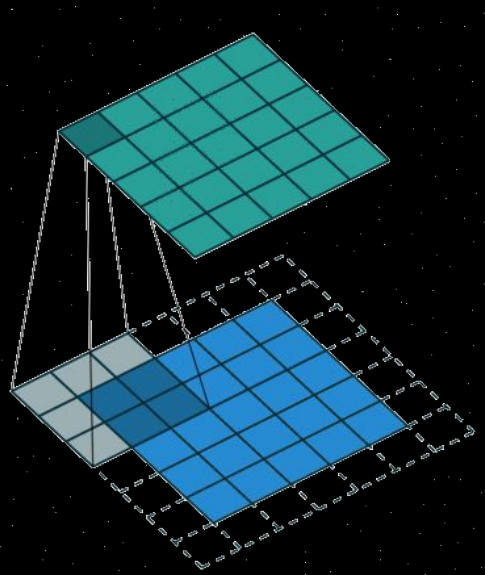


Convolution

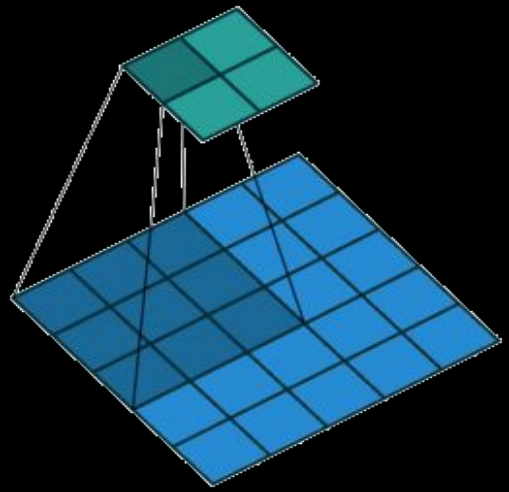
3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



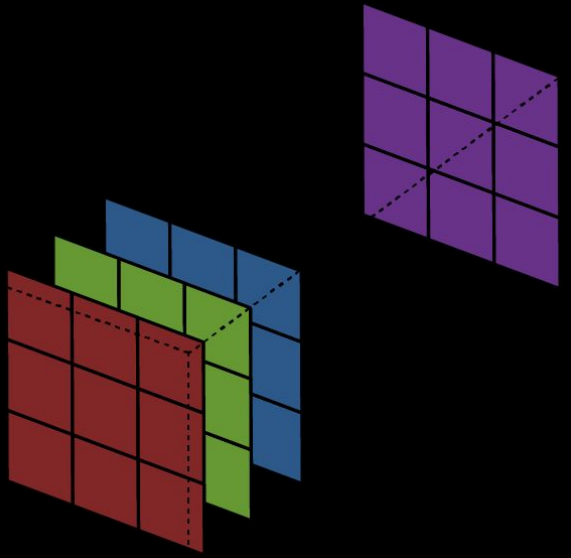


Padding = 1



Stride = 2

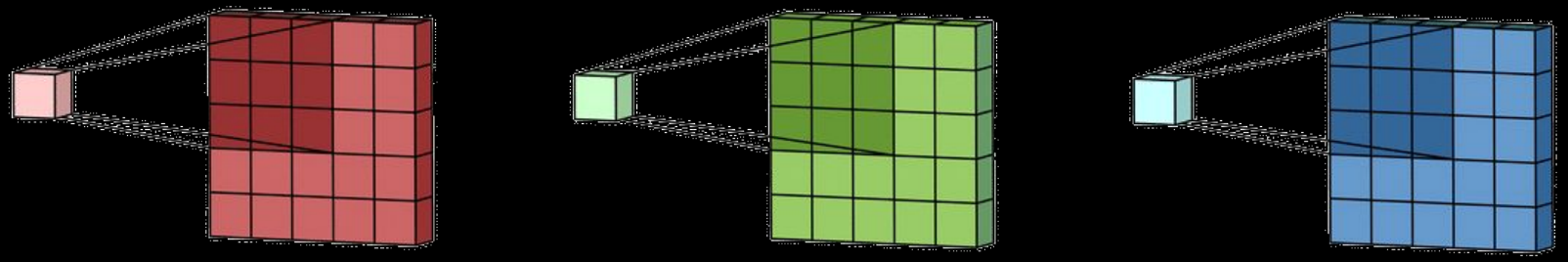
Multi-Channel Convolution

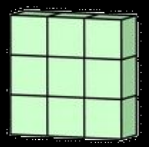


Filter is Collection of Kernels

2 Steps

1. 2D convolution for each kernel
2. Add all ouputs



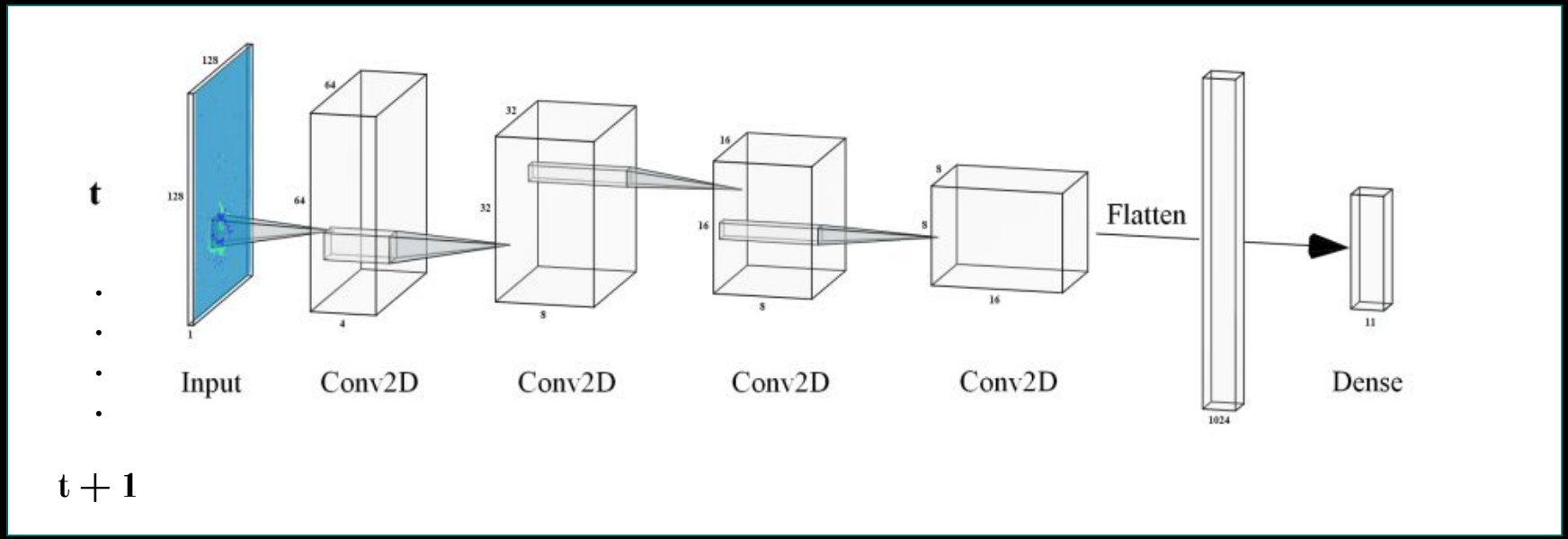

SNN to Convolution SNN

Convolution is a Linear operation, so it can be represented as a Matrix Multiplication

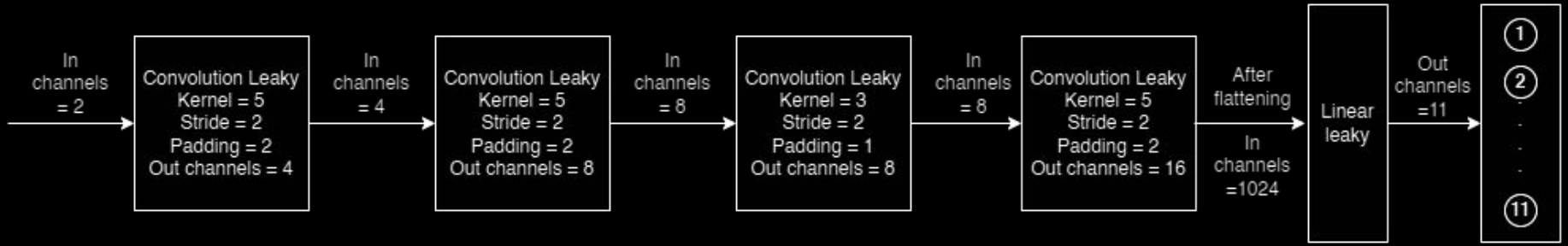
LIF equation - $U[t] = \beta U[t-1] + WX[t] - S_{out}[t-1]\theta$

W X[t] can be replaced with convolution operation.

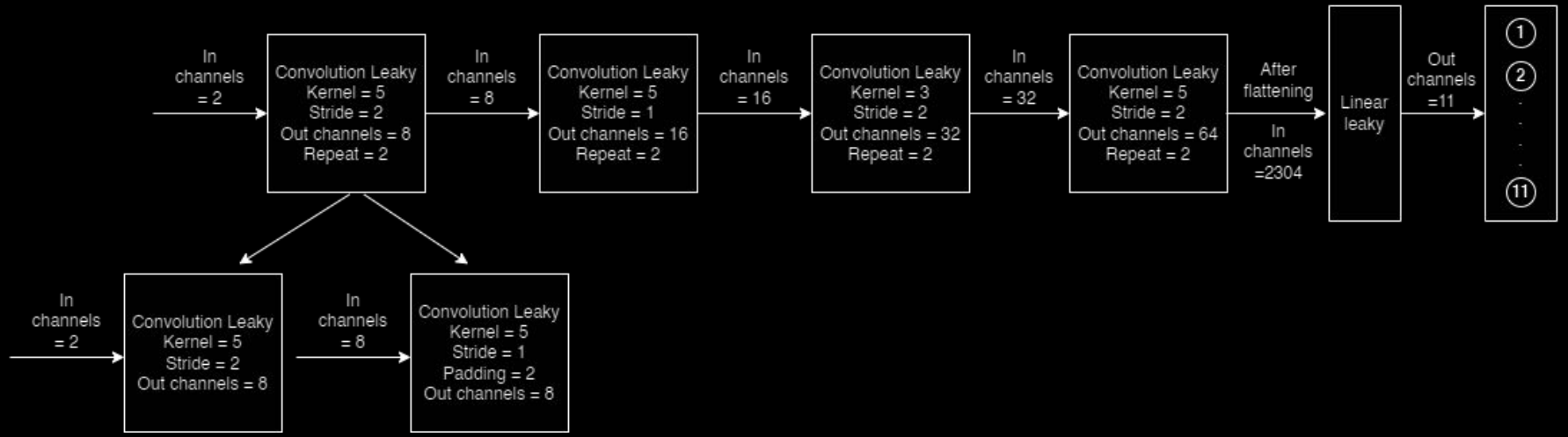
Architecture - Conv-SNN



Architecture	Parameters
4C5K2P–8C5K2P–8C3K1P–16C3K1P–D–1024FC11	14k



Architecture	Parameters
8C5K2R–D–16C5K2R–D–32C3K2R–D–64C3K2R–D – 2304FC11	106k



Training-

-Loss function

Cross Entropy Rate Loss

-Surrogate gradient descent

Due to dead neuron problem during backward pass the heavy side function is approximated as sigmoid function

- Optimizer - ADAM optimizer is used
- Batch size = 16
- Training dataset is split into Validation and Train dataset with ratio of 0.1

Hyper-parameters

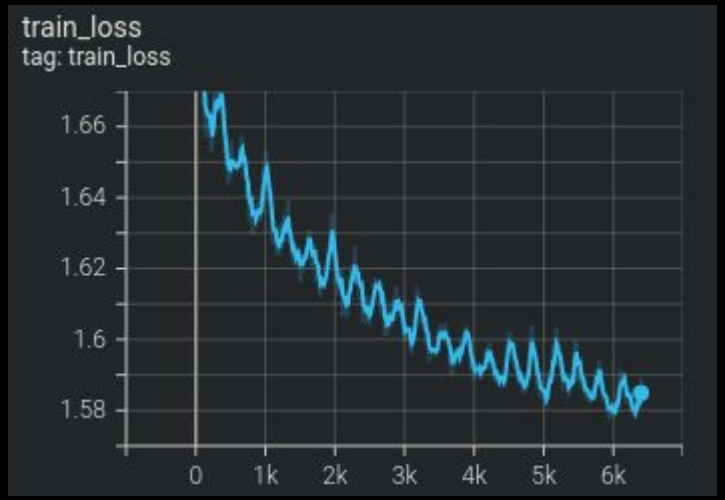
Hyper parameter	Priority
Slope of Surrogate	0.28
Beta	0.22
Threshold	0.22
Learning rate	0.16
Dropout rate	0.13

Tuning

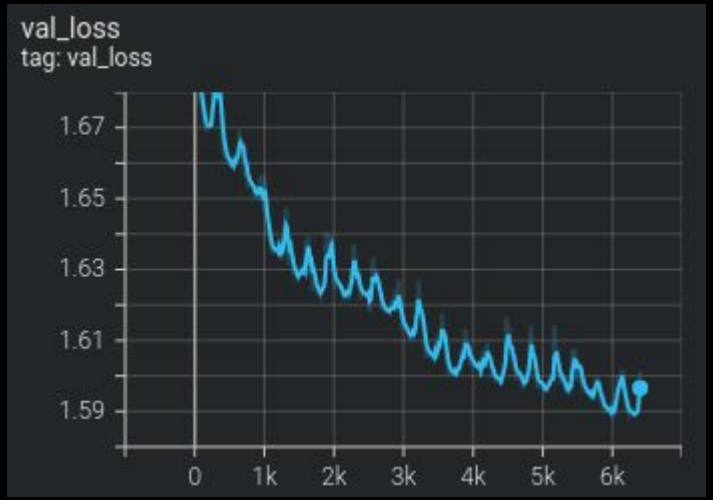

Intermediate Value
Step
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
10
20
30
40
50

Results - Loss

Loss : 1.589

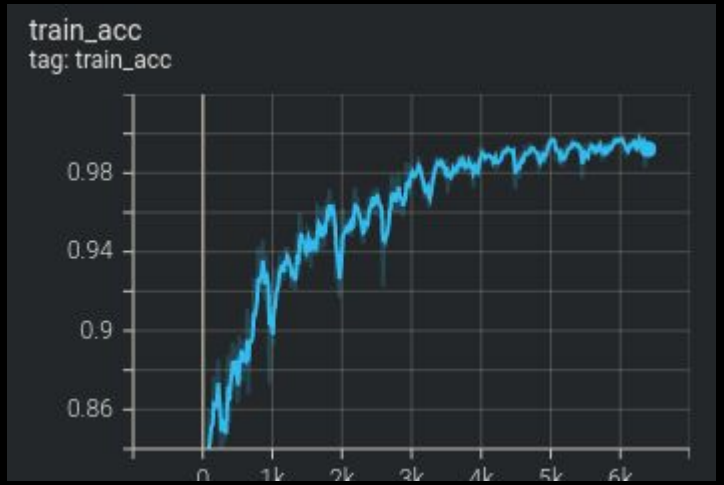


Loss : 1.603

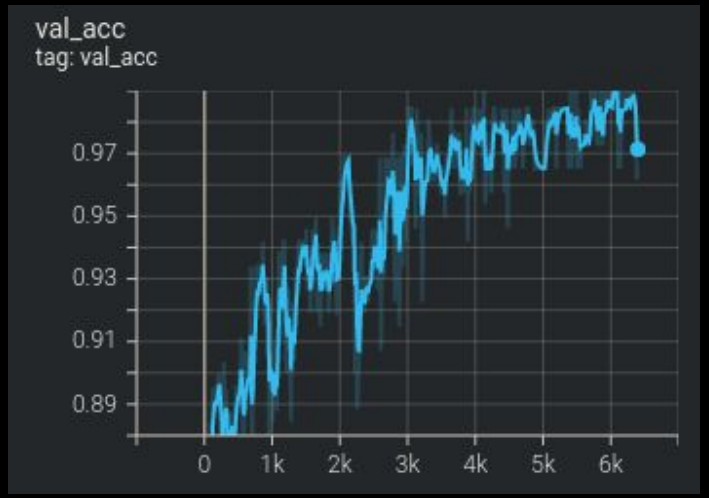


Results - Accuracy

Accuracy : 0.9951



Accuracy : 0.9879



Results - Final

Phase	Accuracy
Train	0.9951
Validation	0.9879
Test	0.8813

Libraries used -

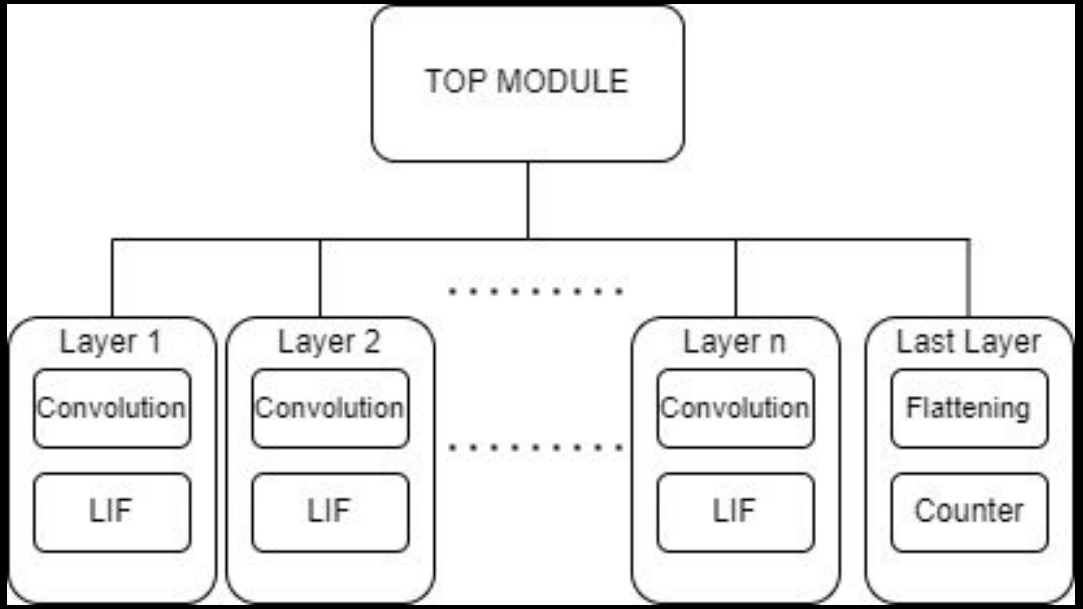
- Pytorch lightning
- SnnTorch
- Tonic
- Optuna
- QTorch

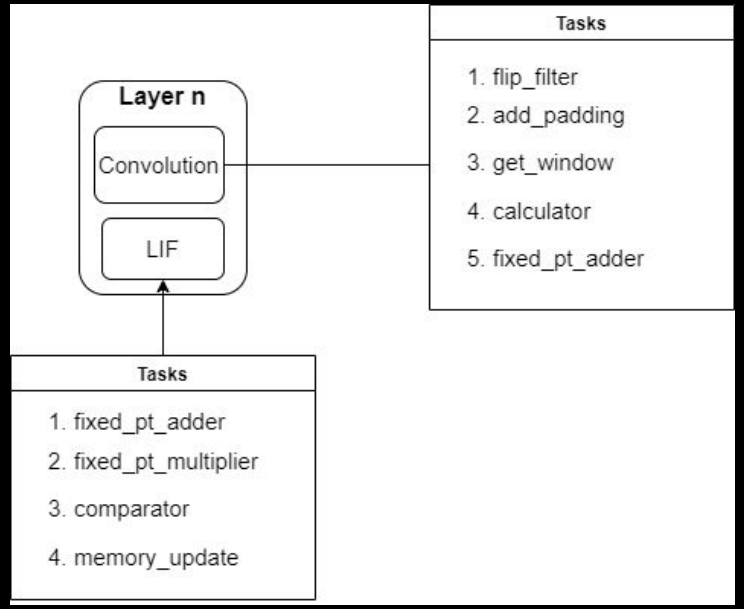
HDL Implementation

Hardware simulation

- A convolutional spiking neural network will be implemented in HDL.
- The script has to be synthesizable. Artix-7 FPGA has been targeted for synthesis and implementation.
- Values of input, all kernels and biases will be read from a file that is updated for each iteration.
- Aim is to make the code modular, flexible and scalable in terms of number of layers, kernel size, number of neurons and number of output labels.
- Data is passed serially to the modules.
- 35 bit Fixed-Point representation has been used.

Block Diagram

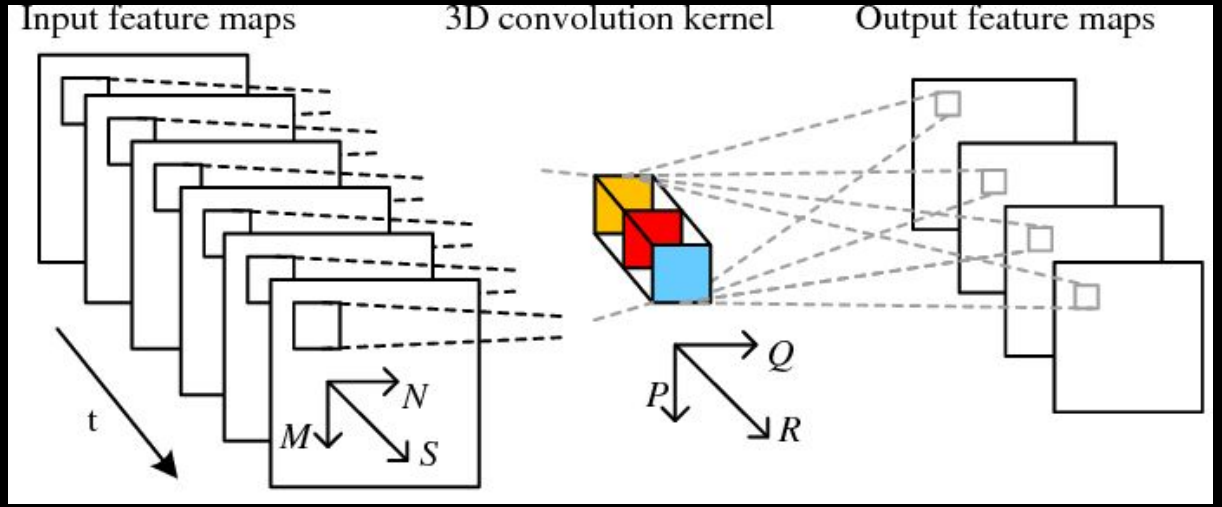


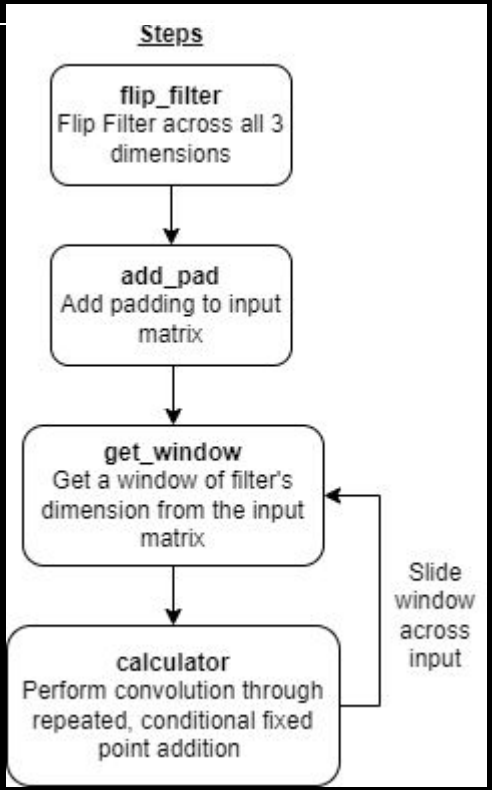


Block Diagram

Convolution Module

- Multiple input and output channel convolution is to be done.
- Due to spiking inputs, the traditional multiply-and-add operation is reduced to add-if-non-zero.

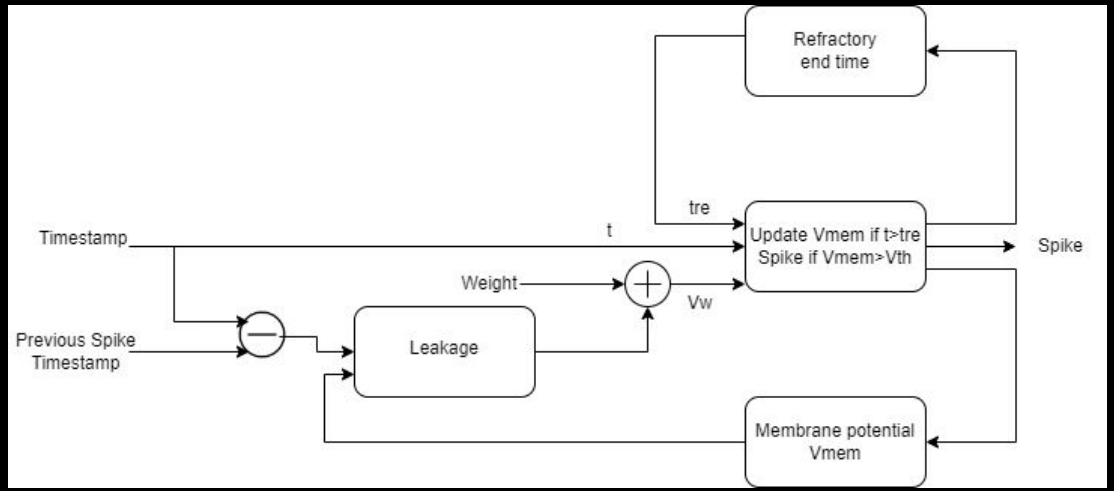


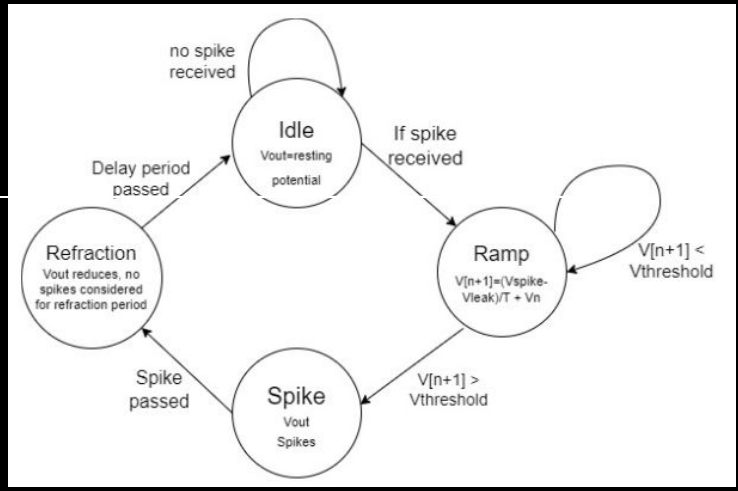


LIF Module

A layer of LIF neurons follows every convolution layer

Block Diagram





State Diagram

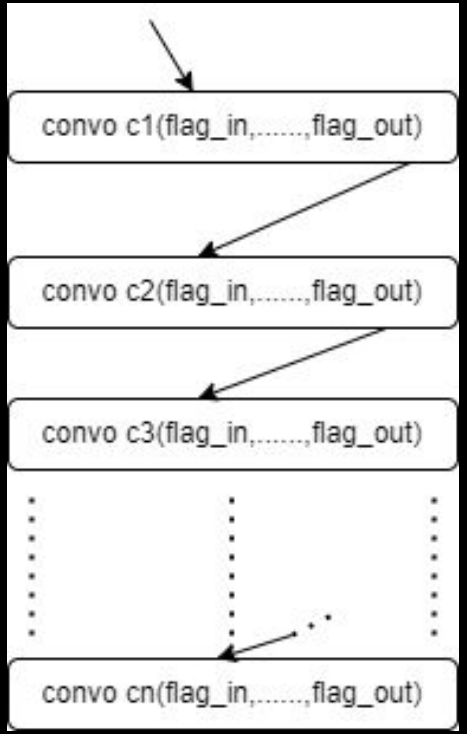
It requires a fixed point multiplier and adder, comparator and memory update operation per iteration.

Flattening and Counter Modules

- The flattening module acts as a linear layer converting large matrix arriving from the preceding layer to a linear array of size equal to the number of labels in the output: 11 in this case.
- The counter module keeps count of the ones appearing at each position in the flattened array across all iterations.
- The position holding maximum count corresponds to the output label.

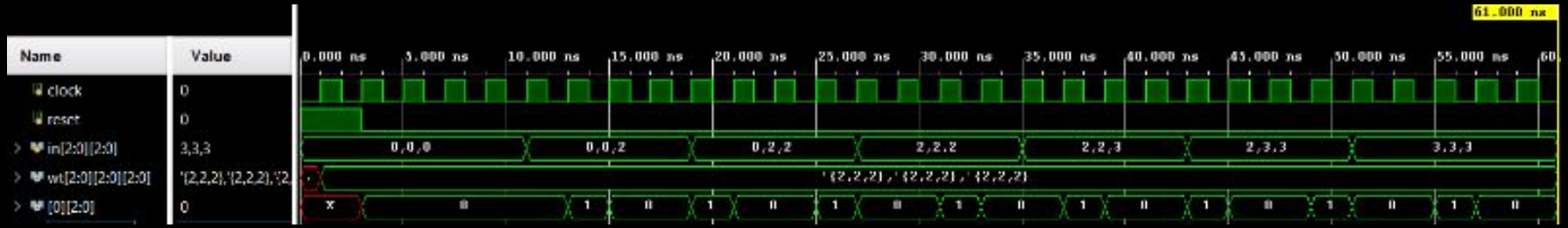
Method of Relaying Flags

- All modules for every layer are called sequentially.
- Execution of 1 module begins after the previous module has finished execution– indicated by setting a flag.



LIF neuron model:

- in[2:0][2:0] is the set of inputs with corresponding weights stored in wt[2:0][2:0][2:0].
- The variable out[0][2:0] goes to 1 when Vmembrane > Threshold voltage
- Vmembrane returns to resting value after a delay accounting for the refractory period.

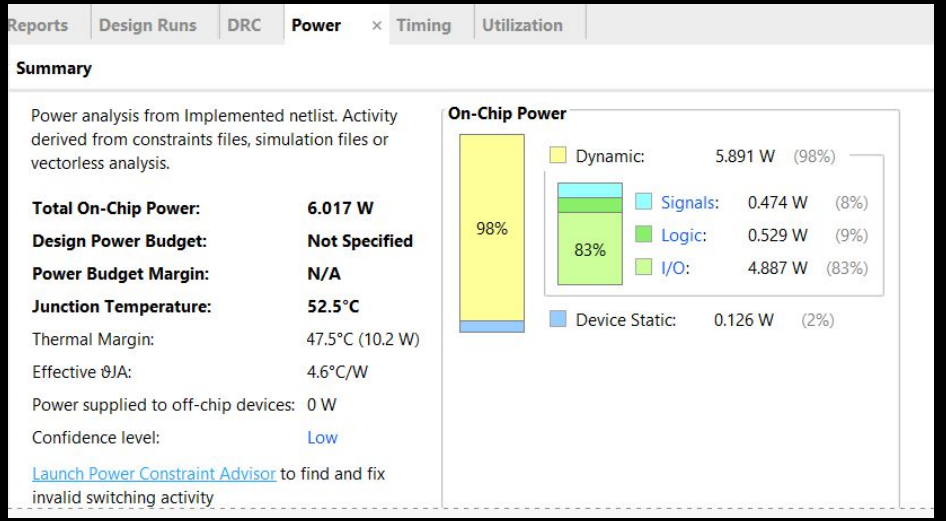


Synthesis report

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	57	0	47200	0.12
LUT as Logic	57	0	47200	0.12
LUT as Memory	0	0	19000	0.00
Slice Registers	0	0	94400	0.00
Register as Flip Flop	0	0	94400	0.00
Register as Latch	0	0	94400	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Resource utilization



Power consumption

Convolution Layer

- Results for a scaled down convolution operation.

```
---------------------------------------------------------
Detailed RTL Component Info :
+---Registers :
                   11 Bit    Registers := 446
                    5 Bit    Registers := 4
                    1 Bit    Registers := 2
+---Muxes :
       2 Input    1 Bit         Muxes := 608
---------------------------------------------------------
```

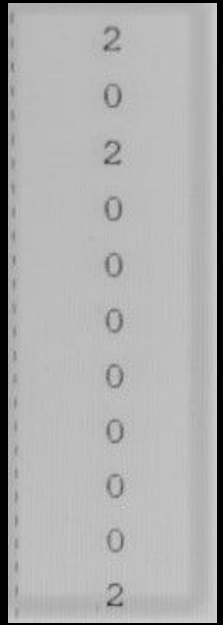
Resource utilization

```
Input Kernel:
  1   2   3
  1   2   3
  1   2   3
Flipped Kernel:
  3   2   1
  3   2   1
  3   2   1
Input matrix:
  1   1   1   1   1   1   1
  1   1   1   1   1   1   1
  1   1   1   1   1   1   1
  1   1   1   1   1   1   1
  1   1   1   1   1   1   1
  1   1   1   1   1   1   1
  1   0   0   0   1   0   0
Convolution:
 18
 18
 18
 18
 18
 18
 18
 15
 13
 15
```

Simulation Output

Flattening and Counter Layer

- The output of the matrix multiplication is passed through a LIF layer which returns output as either 0 or 1 for each of the 11 positions in the array.
- The counter module keeps count of the ones appearing at each position in the flattened array across all iterations. The position holding maximum count corresponds to the output label.
- Here, the maximum count 54 has been achieved for label 2, hence that is the output



counter state
7
54
8
2
11
23
5
13
7
14
6

Comparison with traditional convolution

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	484	0	47200	1.03
LUT as Logic	484	0	47200	1.03
LUT as Memory	0	0	19000	0.00
Slice Registers	1936	0	94400	2.05
Register as Flip Flop	1936	0	94400	2.05
Register as Latch	0	0	94400	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	0	0	47200	0.00
LUT as Logic	0	0	47200	0.00
LUT as Memory	0	0	19000	0.00
Slice Registers	0	0	94400	0.00
Register as Flip Flop	0	0	94400	0.00
Register as Latch	0	0	94400	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

3. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	180	0.00

3. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	121	0	180	67.22
DSP48E1 only	121			

Conclusion and Future Work

Conclusion and Future Work

- The DVS128 dataset developed by IBM has been used to train 2 architectures of convolutional SNN. The training has achieved a fair accuracy of around 0.88.
- The 4-layer architecture has been modelled in SystemVerilog and results obtained through simulation in Xilinx Vivado.
- The HDL code has been so designed such that it is completely synthesizable, modular as well as extensible.
- Future work involves further improving the training accuracy by extending the architecture, experimenting with hyperparameters.
- We also aim to carry out hardware optimization to reduce resource utilization and actually deploy the code on FPGA.

References

References

- E. O. Neftci, H. Mostafa and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks," in IEEE Signal Processing Magazine, vol. 36, no. 6, pp. 51-63, Nov. 2019.
- Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Frontiers in Computational Neuroscience, 9:99, 2015.
- S. Gupta, A. Vyas and G. Trivedi, "FPGA Implementation of Simplified Spiking Neural Network," 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2020, pp. 1-4, doi:10.1109/ICECS49266.2020.9294790.
- F. Akopyan et al., "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 10, pp. 1537-1557, Oct. 2015, doi: 10.1109/TCAD.2015.2474396.

References

- A Low Power, Fully Event-Based Gesture Recognition System
- Surrogate Gradient Learning in Spiking Neural Networks
- Hardware Design of a Leaky Integrate and Fire Neuron Core Towards the Design of a Lowpower Neuro-inspired Spikebased Multicore SoC
- A Low-Cost High-Speed Neuromorphic Hardware Based on Spiking Neural Network

Thank You!