

Major Project Report

on

Spiking Neural Network (SNN) in Verilog HDL

Submitted by:

Sai Sathvik G B (21BEC042)

Under the guidance of:

Dr. Jagadish D N

Assistant Professor, Dept. of ECE



INDIAN INSTITUTE OF
INFORMATION
TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY (IIIT) DHARWAD

11 April 2025

Certificate

This is to certify that the project, entitled: **Spiking Neural Network (SNN) in Verilog HDL**, is a bonafide record of the Major Project coursework presented by the student mentioned below during the academic year 2024-2025 in partial fulfillment of the requirements of the Bachelor of Technology degree in Electronics and Communication Engineering.

Submitted by:

Sai Sathvik G B (21BEC042)

Dr. Jagadish D N

Project Supervisor

Contents

List of Figures	ii
List of Tables	ii
1 Introduction to the Project	1
1.1 About Neuromorphic Computing	1
1.2 About Spiking Neural Networks	2
1.3 About Spike Time Dependent Plasticity	4
2 Description of Related Works	5
3 Design and Implementation	6
3.1 Workflow Description	6
3.2 Details of the SNN model used	10
3.3 Details of the STDP algorithm used	12
4 Results and Conclusion	13
4.1 Training and Testing Results from Icarus Verilog	13
4.2 Concluding Discussions	16
4.2.1 Possible improvements to the project	16
4.2.2 Future Scope and Conclusion	16
References	17

List of Figures

1	Neuromorphic Computing vs. Traditional Computing	1
2	Comparison of SNN with ANN and MPNN	2
3	About the Leaky-Integrate-and-Fire Neuron	3
4	CMOS implementation of the Spiking Neuron	3
5	Formula for STDP Algorithm Weight Update	4
6	Height Encoding Logic	9
7	Weight Encoding Logic	10
8	Expected Classification Encoding Logic	10
9	SNN model implemented in the project	11
10	Output Decoding logic	11
11	STDP Implementation Flow Chart	12
12	Training Accuracy obtained	13
13	Testing Accuracy obtained	13
14	Final iteration of the Training Waveform	14
15	Complete Testing Waveform	14
16	Plot showing the training improvement	15

List of Tables

1	Training Data Table	8
2	Testing Data Table	9

1 Introduction to the Project

1.1 About Neuromorphic Computing

- This is a new & innovative approach to design computing systems, chiefly inspired by the function of the human brain; this is conceptualised by blending neurosciences with traditional computer sciences.
- They often are event-driven models, where information is processed only when an event (or) spike occurs; thereby performing tasks with far less energy than traditional computing systems.
- Such systems are used these days in the fields of robotics, machine learning, edge computing & artificial intelligence; and are starting to replace the traditional systems gradually due to their parallelism and efficiency.
- Although they are great in energy savings and parallel computing, they lag behind in achieving high accuracies as easily as the traditional ones due to present gap in making best adaptive learning algorithms possible.

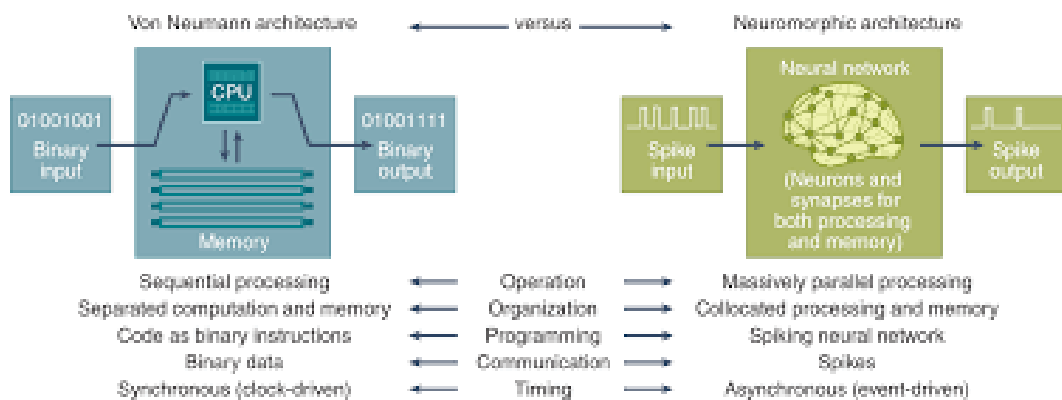


Figure 1. Neuromorphic Computing vs. Traditional Computing

1.2 About Spiking Neural Networks

- Spiking Neural Network (SNN) is one such approach of neuromorphic computing aiming to closely mimic the behavior of biological neurons, and involves discrete events (or) spikes for synaptic communication.
- The spiking neuron fires when its membrane potential reaches a certain threshold; otherwise it starts collecting membrane potential along with some leakage, the value of potential collected depends upon weighted sum.
- The spikes generated carry information about the intensity and timing of events; the processing occurs only when an event occurs (or) spike fires; making them highly save energy as it's being concerning these days.
- Learning algorithms for SNN's like Spike Time Dependent Plasticity (STDP) are adaptive [like the brain] which tends to adjust the neural connections over time; either supervised or unsupervised.

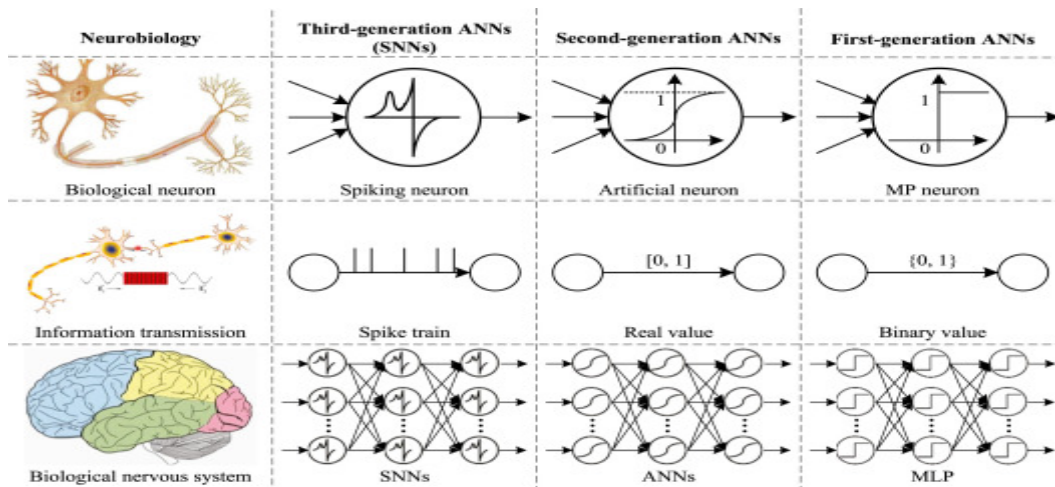


Figure 2. Comparison of SNN with ANN and MPNN

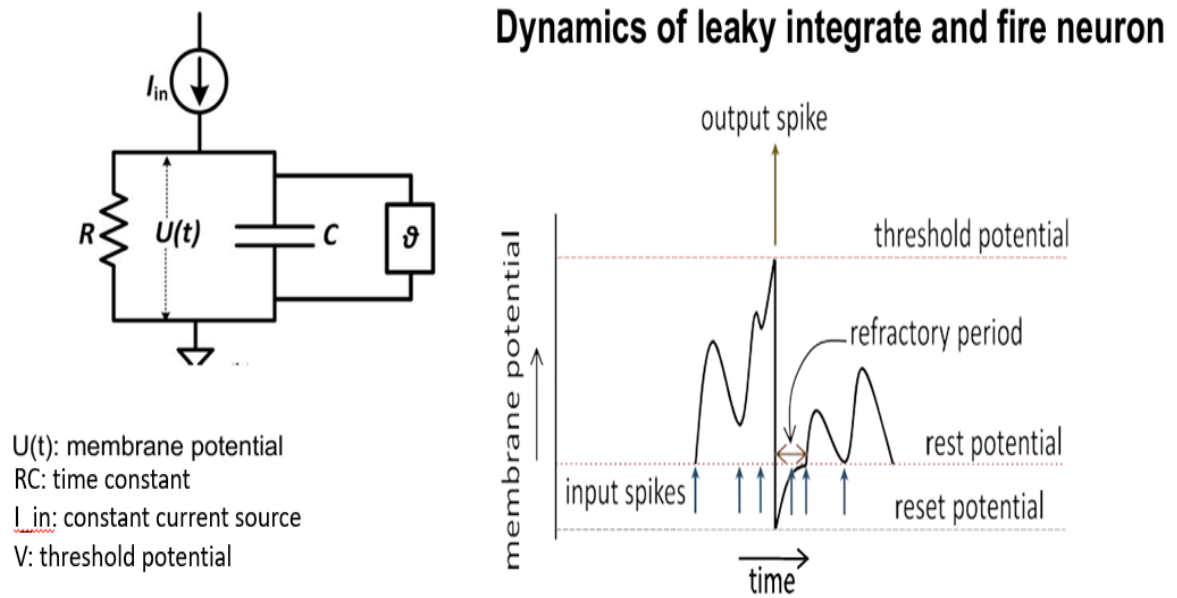


Figure 3. About the Leaky-Integrate-and-Fire Neuron

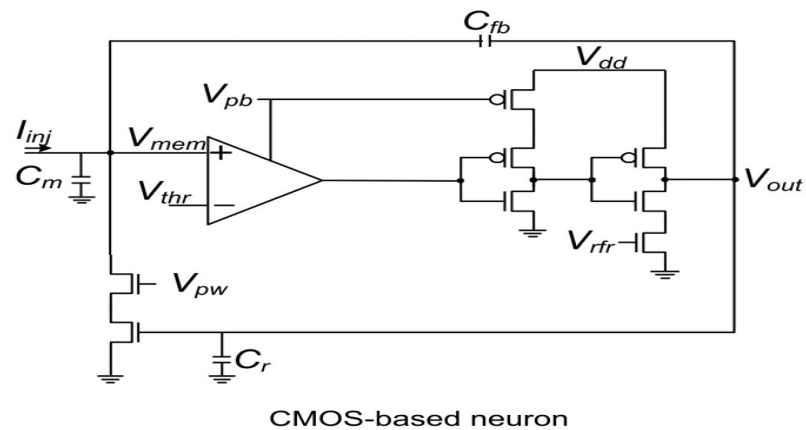


Figure 4. CMOS implementation of the Spiking Neuron

1.3 About Spike Time Dependent Plasticity

- Spike-Time Dependent Plasticity (STDP) is a learning algorithm which refers to the synapses strengthened or weakened based on timing difference between pre-synaptic and post-synaptic spikes.
- It consists of two conditions for learning: strengthening (or) long-term potentiation [pre-synaptic neuron spiked before post-synaptic neuron] and weakening (or) long-term depression [pre-synaptic neuron spiked after post-synaptic neuron]; this will update weights as shown below.
- Actually, the STDP is believed to be one of the fundamental mechanism behind learning in the brain and helps form appropriate synaptic connections and self-organised neural networks based on activity patterns.
- STDP is being applied widely in Spiking Neural Networks, and together are gradually getting replaced by traditional Artificial Neural Networks (ANN's); thereby making Machine Learning energy-efficient.

STDP Learning

$$\Delta w = \begin{cases} A_+ \exp\left(\frac{-\Delta t}{\tau_+}\right), & \Delta t > 0 \\ -A_- \exp\left(\frac{-\Delta t}{\tau_+}\right), & \Delta t < 0 \end{cases}$$

Macro-modelling
astrocyte functionality:

$$A_+ = \frac{K}{1 + e^{-\Delta f}}$$

Figure 5. Formula for STDP Algorithm Weight Update

2 Description of Related Works

- [1] contains the Verilog implementation of an SNN with Leaky-Integrate-and-Fire (LIF) spiking neuron model which is close to ideality (but can be made close to reality); it involved single hidden layer and one output layer.
- [2] describes one of the methods to implement Spiking Neural Network on the Xilinx FPGA; the work involved using the Izhikevich spiking neuron (which balances biological realism and hardware efficiency); the system comprises 1,000 neurons and 1 million synapses, with 54% resources used.
- [3] describes one of the methods to apply the STDP learning algorithm in Verilog; their work primarily involved reduced use of memory and hardware resources by simplifying the exponential calculation, thereby leading the way to low-power neuromorphic computing; the far-future.
- [4] describes the comparison of SNN's with ANN's and the human brain; and highlights the SNN's energy-efficiency and real-time decision making abilities; depicting SNN's showing resemblances to the brain than ANN's.

The work described by this report is primarily inspired by [1] and [3] and implemented in simpler fashion than the originals.

3 Design and Implementation

3.1 Workflow Description

- The objective of this Spiking neural network design is to classify the height and weight of 6-year old kids as underweight, overweight or perfect weight; the 6-bit input given to the SNN comes out as 3-bit output, which is further decoded into another 3-bit value that classifies the input.
- The height (in cm), weight (in kg) and expected classifications are encoded into 6-bit and 3-bit sequences respectively using python scripts for training and testing separately. Also, the weights after complete training are also extracted using python script. These are all stored in separate ".mem" files.
- The training and testing designs are not entirely different; the testing one is dependent on the training one for its weights. Plot (in ".jpg" file) that shows the improvements of training is also generated using python script.

The following tables (in ".mem" files) are the data used in the project.

Ht. (cm)	Wt. (kg)	Type
111.7	19.8	Perfect
106.1	19.2	Under
112.2	22.6	Perfect
106.2	18.1	Perfect
109.9	17.4	Under
121.6	26.9	Over
114.8	22.5	Perfect

109.1	16.8	Under
115.0	20.9	Perfect
105.2	26.1	Over
105.9	19.0	Perfect
105.5	18.0	Under
110.6	21.3	Perfect
105.9	25.3	Over
113.1	21.5	Perfect
117.8	22.8	Perfect
109.3	16.5	Under
112.4	20.3	Perfect
117.8	22.4	Perfect
110.9	17.6	Under
118.3	23.6	Perfect
113.9	26.7	Over
115.7	21.9	Perfect
116.2	24.7	Over
106.5	24.2	Over
110.2	20.9	Perfect
114.1	19.9	Under
109.7	22.1	Perfect
119.0	26.5	Over
107.4	18.7	Perfect
120.0	24.9	Over
118.2	22.9	Perfect

119.7	23.3	Perfect
117.1	25.1	Over
118.9	19.7	Under
119.2	25.5	Over
120.1	24.3	Perfect
109.3	21.2	Over
116.4	23.6	Over
111.5	18.9	Under
108.9	19.5	Perfect
119.2	22.1	Perfect
115.5	25.2	Over
113.7	21.7	Perfect
121.3	24.7	Perfect
107.8	18.2	Under
118.5	23.9	Over
104.6	17.3	Perfect
122.4	26.1	Over
109.8	20.4	Perfect

Table 1: Training Data Table

Ht.(cm)	Wt.(kg)	Type
120.1	25.2	Perfect
106.9	17.9	Under

109.4	26.0	Over
112.5	20.3	Perfect
107.9	18.1	Under
120.0	28.3	Over
111.5	20.9	Perfect
108.9	17.5	Under
114.0	23.1	Over
113.4	21.8	Perfect
118.7	26.2	Over
107.2	18.0	Under
112.1	22.4	Perfect
116.3	24.5	Over
118.0	21.5	Under

Table 2: Testing Data Table

```

#conditions for encoding height: range is 105 to 120 cm for 6-year olds
#encode unique 3-bits for every 2.5cm
if height <= 105.0:
    ht_bits = "000"
elif height > 105.0 and height <= 107.5:
    ht_bits = "001"
elif height > 107.5 and height <= 110.0:
    ht_bits = "010"
elif height > 110.0 and height <= 112.5:
    ht_bits = "011"
elif height > 112.5 and height <= 115.0:
    ht_bits = "100"
elif height > 115.0 and height <= 117.5:
    ht_bits = "101"
elif height > 117.5 and height <= 120.0:
    ht_bits = "110"
elif height > 120.0:
    ht_bits = "111"
else: #if anything else is entered, throws error

```

Figure 6. Height Encoding Logic

```

#conditions for encoding weight: range is 17 to 26 kg for 6-year olds
#encoded unique 3-bits for every 1.5kg
if weight <= 17.0:
    wt_bits = "000"
elif weight > 17.0 and weight <= 18.5:
    wt_bits = "001"
elif weight > 18.5 and weight <= 20.0:
    wt_bits = "010"
elif weight > 20.0 and weight <= 21.5:
    wt_bits = "011"
elif weight > 21.5 and weight <= 23.0:
    wt_bits = "100"
elif weight > 23.0 and weight <= 24.5:
    wt_bits = "101"
elif weight > 24.5 and weight <= 26.0:
    wt_bits = "110"
elif weight > 26.0:
    wt_bits = "111"
else: #if anything else is entered, throws error

```

Figure 7. Weight Encoding Logic

```

#conditions for encoding expected output: overweight, underweight or perfect
if kind == "Over":
    kind_bits = "100"
elif kind == "Perfect":
    kind_bits = "010"
elif kind == "Under":
    kind_bits = "001"

```

Figure 8. Expected Classification Encoding Logic

3.2 Details of the SNN model used

- The Spiking neural Network (SNN) implemented in this project has single input layer, hidden layer and output layer respectively; there are 6 neurons each in the input and hidden layer but only 3 neurons in the output layer.
- Leaky-Integrate-and-Fire (LIF) spiking neuron model has been applied in the project for all the 9 neurons (of hidden and output layers, since input layer only pushes the input to the SNN); with all parameters 20-bit each.
- All the 54 weights (36 between input and hidden layers, 18 between hidden and output layers) are 12-bit signed integers (-2048 to +2047); these weights are updated with 50 data samples iterated for 217 times during training.

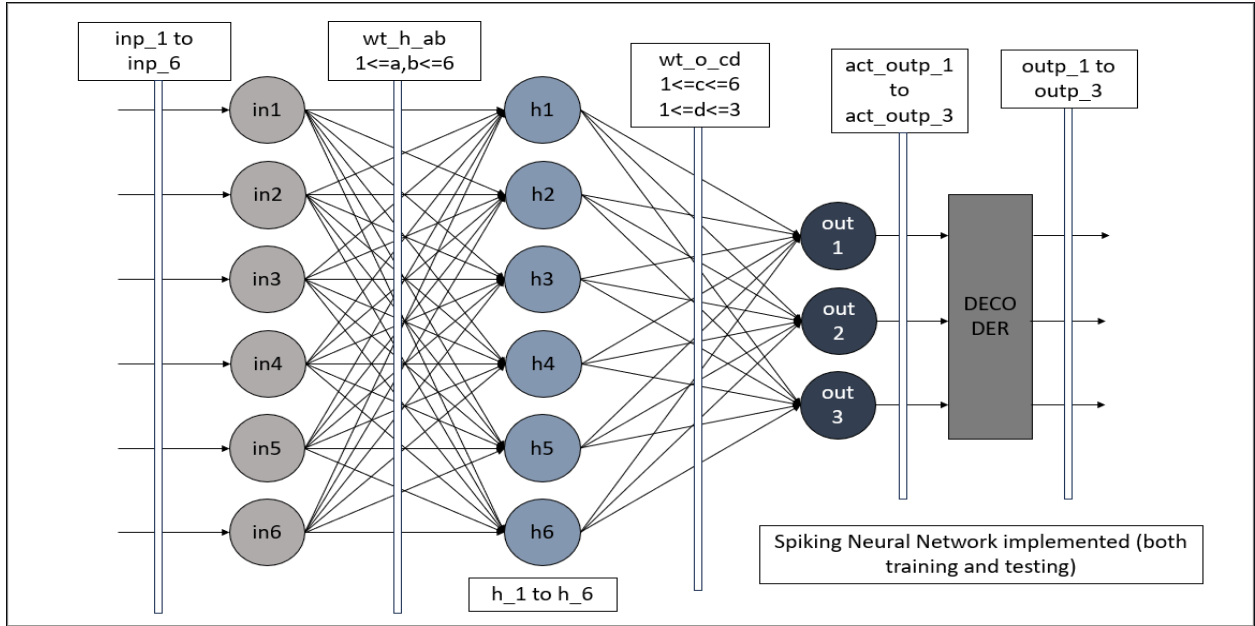


Figure 9. SNN model implemented in the project

```
//decoder logic to deduce output from the actual output to improve further iterations
always@(posedge clock)//decode the outputs in positive clock edge
begin
    casex({act_outp_1,act_outp_2,act_outp_3})//this section is to decode the output
        3'b1xx: {fin_outp_1,fin_outp_2,fin_outp_3} = 3'b100;//100,101,110,111 are all 100
        3'b01x: {fin_outp_1,fin_outp_2,fin_outp_3} = 3'b010;//010,011 are all 010
        3'b001: {fin_outp_1,fin_outp_2,fin_outp_3} = 3'b001;//001,000 are all 001
        default: {fin_outp_1,fin_outp_2,fin_outp_3} = 3'b000;//by default this is 000
    endcase
end
```

Figure 10. Output Decoding logic

3.3 Details of the STDP algorithm used

- The STDP algorithm has been used in the training only; with workflows each for long-term potentiation, long-term depression, no-synapse and both-synapse occurrence scenarios respectively.
- The change in weight is also 12-bit signed, while some parameters like t_+ and t_- are taken as +2 and -2 respectively so as to apply as right shifts.

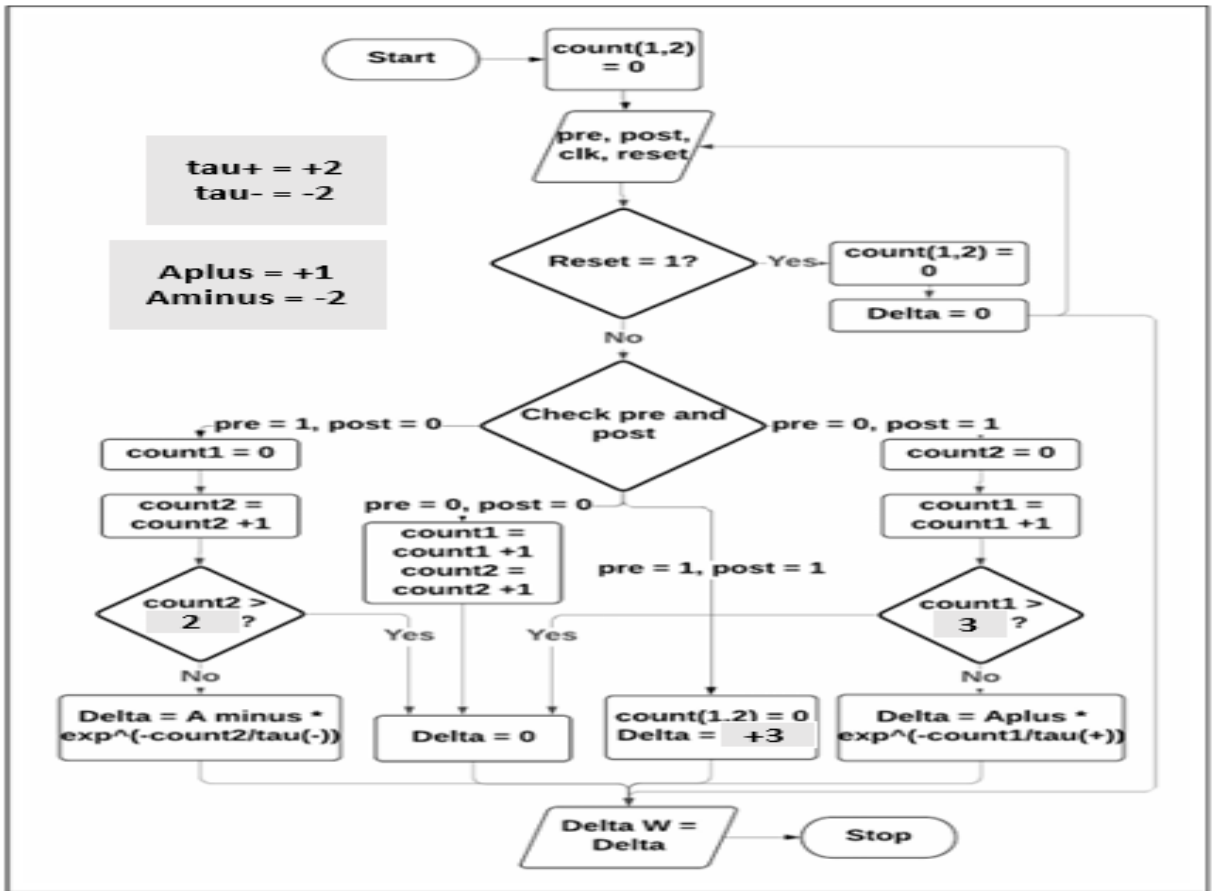
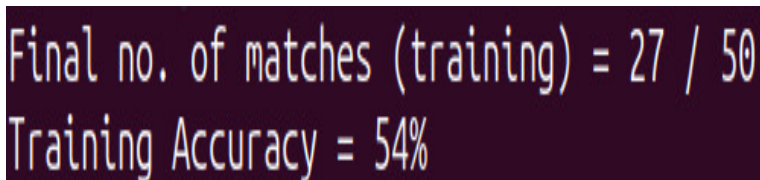


Figure 11. STDP Implementation Flow Chart

4 Results and Conclusion

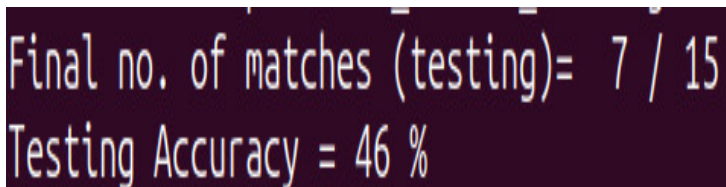
4.1 Training and Testing Results from Icarus Verilog

- Training was done for 217 iterations using 50 data samples; the initial iterations didn't spike the output neurons at all, but with time they started to spike. In final iteration, training accuracy was 54% (27 of 50 matched).
- It has been observed that as no. of iterations are increased (like 300 and more), the outputs started to get stuck at a single value although the weights were getting updated (mostly start to overflow). The previously mentioned 54% taken was recorded as the highest even till 347 iterations.
- Testing was done on 15 unique data samples and the accuracy witnessed was 46% (7 of 15 matched); there was the same problem as seen in training.



```
Final no. of matches (training) = 27 / 50
Training Accuracy = 54%
```

Figure 12. Training Accuracy obtained



```
Final no. of matches (testing)= 7 / 15
Testing Accuracy = 46 %
```

Figure 13. Testing Accuracy obtained



Figure 14. Final iteration of the Training Waveform

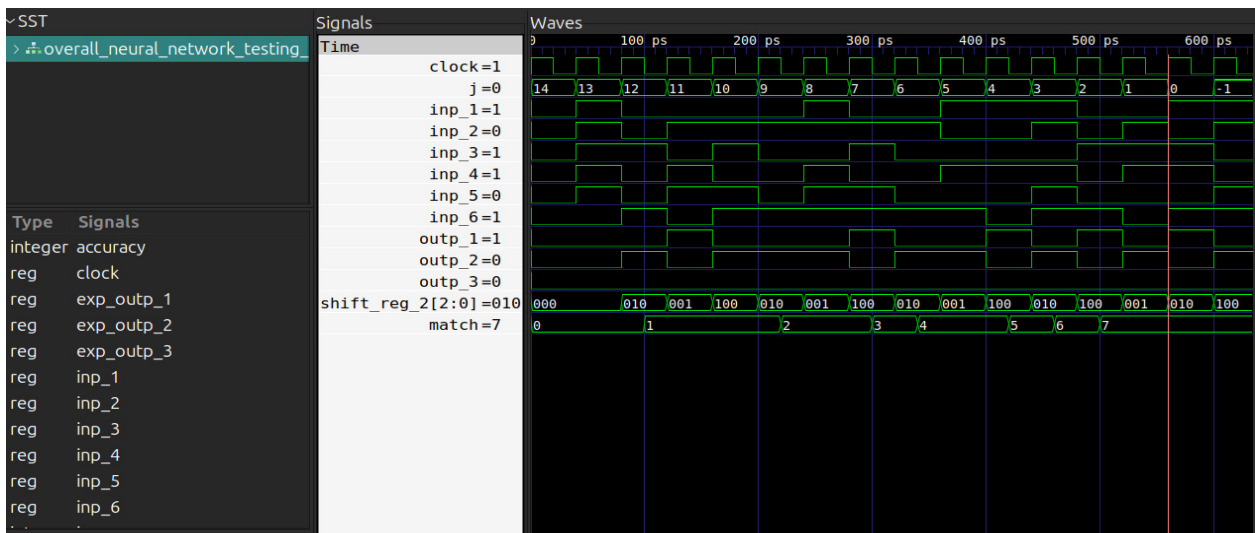


Figure 15. Complete Testing Waveform

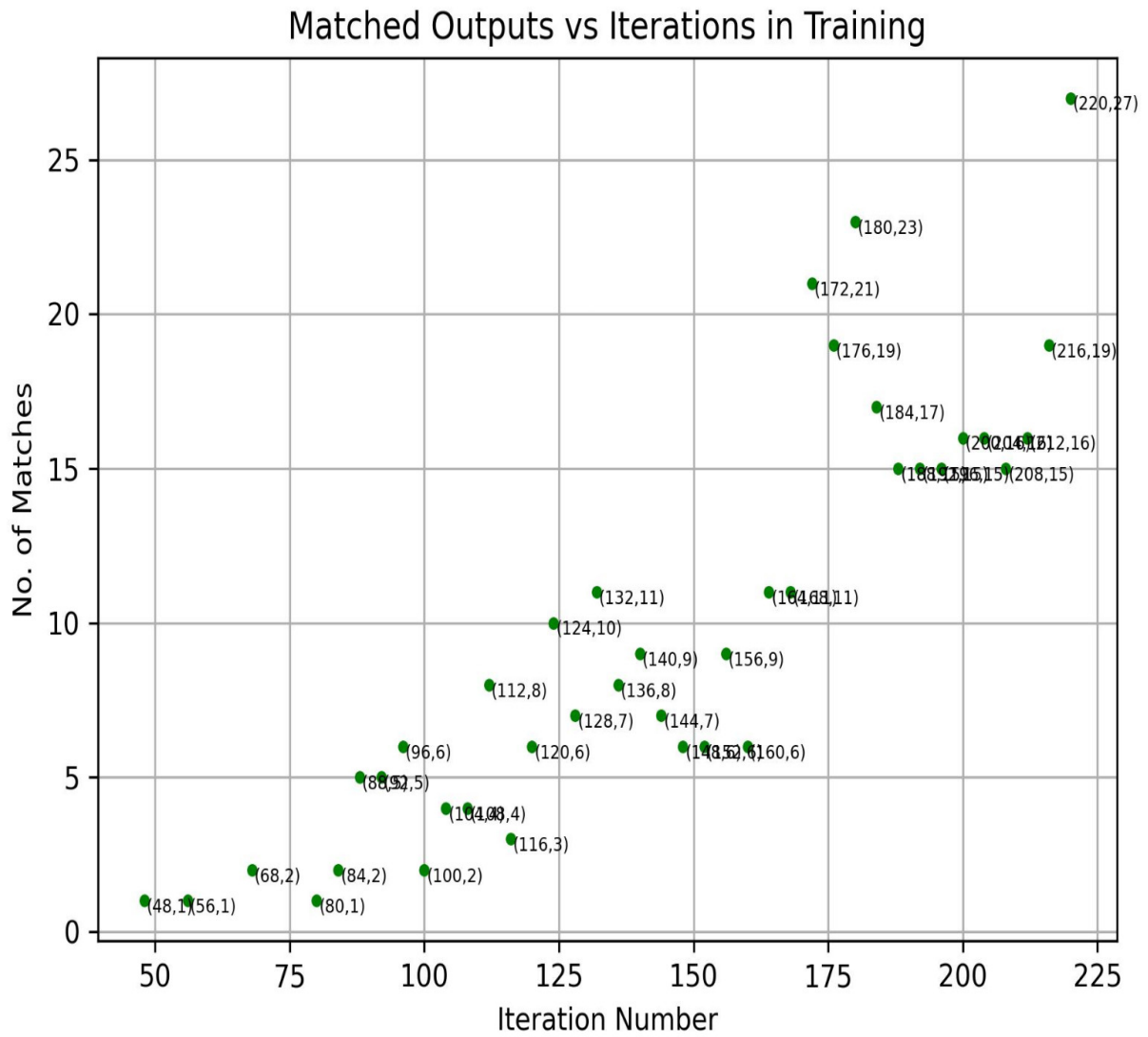


Figure 16. Plot showing the training improvement

4.2 Concluding Discussions

4.2.1 Possible improvements to the project

- Since the STDP learning algorithm implemented in this project is unsupervised, there can be possibility of replacing by it's supervised version (might be challenging if hidden layers exist), which could help SNN learn better.
- The Leaky-Integrate-and-Fire Spiking neuron can be replaced by the ideal one or the Izhikevich one; although the latter is considered complex in HDL due to multiple differential equations, it's close to human neuron's working.
- The training methodology can be improved so that there won't be such observations of outputs getting stuck; the weights can be 32-bit floating point format so as to improve update and reduce overflows.

4.2.2 Future Scope and Conclusion

- Low-Power optimisation techniques can be applied to this implementation to make it more energy-conserving and futuristic; due to the current demand of low-power designs it can be of immense use.
- Popular Machine-Learning models like AlexNet, VGGNet, etc. can also be implemented using Spiking neurons provided there's a better algorithm for both implementation and learning altogether in Verilog HDL.

This concludes that although SNN's are hard to train and implement in HDL's, they are better in terms of energy-savings & parallelism.

References

- [1] Nikita Buzov. Spiking neural net, 2023. URL <https://github.com/nikitabuzov/SpikingNeuralNet>.
- [2] Jungmin Choi, Minwook Ahn, and Jong Tae Kim. Implementation of hardware model for spiking neural network. In *Proceedings of the International Conference on Artificial Intelligence (ICAI'15)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.
- [3] Amrutha Manoharan, Gadamsetty Muralidhar, and Binsu J Kailath. A novel method to implement stdp learning rule in verilog. In *2020 IEEE Region 10 Symposium (TENSYP)*. IEEE, 2020.
- [4] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking neural networks and their applications: A review. *brain sciences*, 12(7), 2022.