Design of Digital Spiking Neural Networks for FPGA implementation

Ajinkya Gorad (140110033)
Amritesh Sharma (14D260012)
Shyam Ramamoorthy (140070042)
Dipti Ranjan Sahu (140070038)
Under the guidance of Prof. Sachin Patkar

Table of Contents

Introduction

Spiking Neural Network (SNN) are networks of ‘neurons’ which use spikes in form of spatiotemporal pattern to encode information. They are the biological representation model of actual neurons similar to our brain. Understanding the computations in their domain is still hot and active emerging field. Such network could be simulated on the PC, to realise the solution of real-time processing, it can be implemented on hardware barring the limitations on the resources on hardware.

Biological Background

A human neuron can basically be divided in three parts:

- Soma - The main body of the neuron
- Axon - part of neuron that transports the action potential from the soma to other cell
- Dendrites - part which receives action potentials from other cells and pass on to the soma

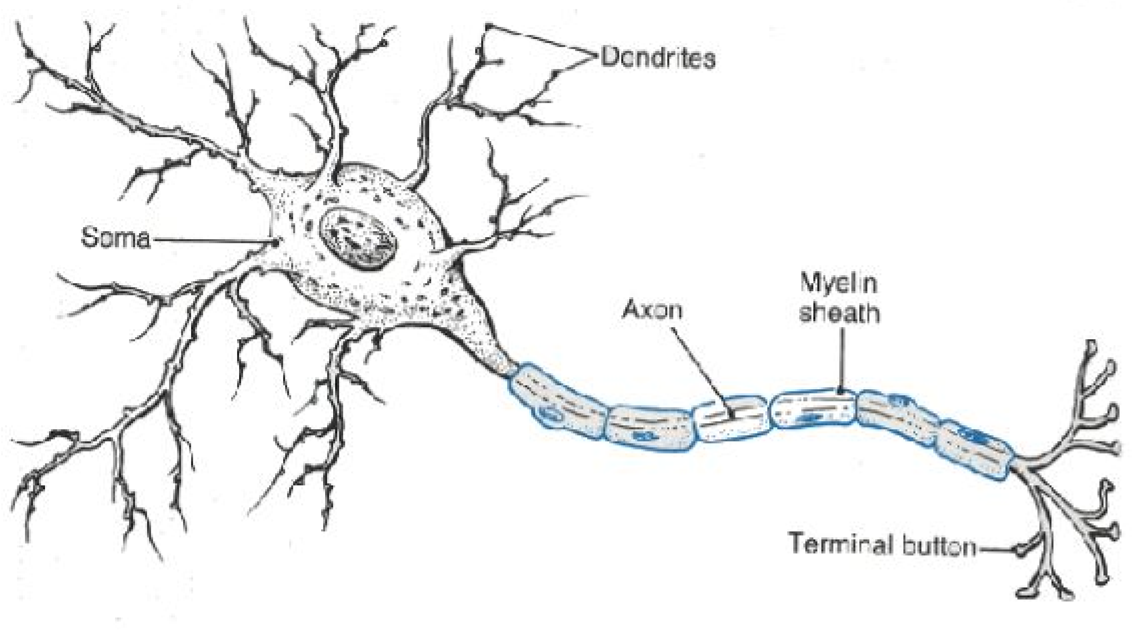


Image Courtesy: Javier Gómez Casado, Implementation of an artificial neuron in VHDL, 2008

The central part of the cell: the soma sums up (often non-linearly) the stimulus received from all of its dendrites and generates an impulsive spike if it crosses a certain threshold. This spike is nothing but a small period electrical pulse which is passed down the axon to other neurons. The dumping of these pulses happen at the site called synapse. And it is here, where the dendrites of the following neuron (also called the post-synaptic neuron) collect the spike message. As a matter of fact these pulses have an amplitude of 100mV and period of about 1-2ms.

In a matrix of these neurons in humans or any organism for that matter in actual time the neuron here and there keep firing up. Information is believed to be contained in series of impulses: the timing and the number. As aforementioned, the summation is non-linear, some of the inputs have more relevance and so are weighed more. So actually its a weighted summation. In real life of organism, these weights usually change while learning which are aptly captured by the most of artificial models used today.

Architecture

We implement the LIF model of neuron[ref] digitally. We take 1ms as timestep of time evolution of our network.
There are several components of the architecture. They are:

Neuron (Soma)

The working of the soma is represented by the following equation:

$$V_{n+1} = \alpha V_n + I_{app} + I_{syn} \qquad (LIF\ model)$$
$$V_n > V_{Th} \Rightarrow spike = 1$$

The voltage at the (n+1)th time interval is updated based on the voltage at the nth time interval and also the two currents, Iapp and Isyn. If the voltage at any particular instant crosses the threshold voltage Vth, the neuron will spike. There is a minimum current, at which a neuron can spike for the LIF model.

It can be represented by the following block diagram. Clearly there is some feedback in the circuit, which is to be expected given that the present voltage has a linear dependence on the past voltage.

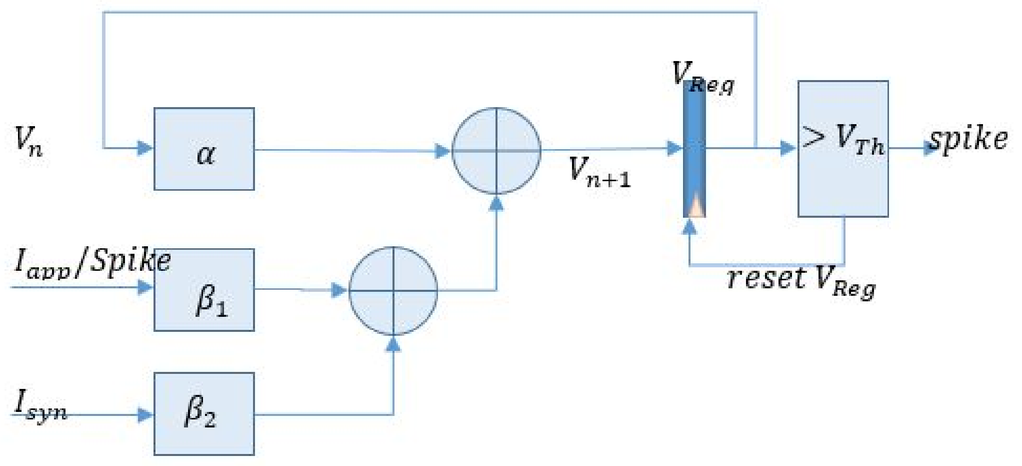
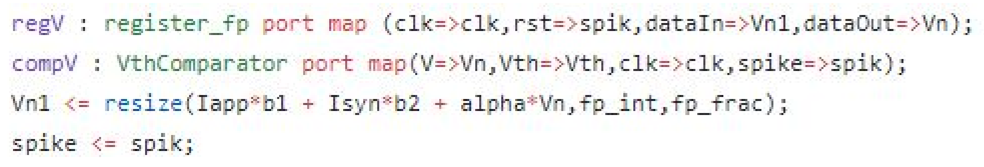


Fig. Digital block diagram for Neuron(soma)

The VHDL code for the neuron(soma) is in the neuron.vhd file.

--



```
regV : register_fp port map (clk=>clk,rst=>spik,dataIn=>Vn1,dataOut=>Vn);
compV : VthComparator port map(V=>Vn,Vth=>Vth,clk=>clk,spike=>spik);
Vn1 <= resize(Iapp*b1 + Isyn*b2 + alpha*Vn,fp_int,fp_frac);
spike <= spik;
```

--

The register_fp entity (fixed point register) is used to store the values of the voltage and allows us to implement the feedback that we need. The VthComparator entity is used to compare the current neuron voltage to the threshold voltage. If Vn (current voltage) is greater than the Vth (threshold voltage), the spike signal is set to ‘1’, ie. the neuron spikes.

The resize function of the ieee_proposed is used to adjust the number of integral/fractional bits in the fixed point representation of V(n+1).

Following shown is the fixed point simulation of neurons for fractional depths of 10,11,12,13,14 bits from bottom up, in Matlab.

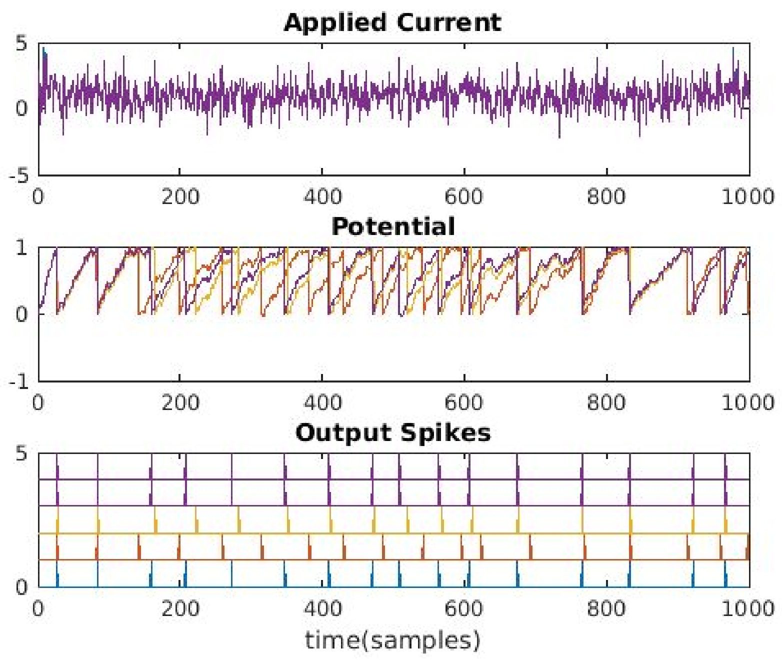


Fig. Fixed point simulation of neuron(soma) entity in matlab for fractional bits 10,11,12,13,14. Shown is the applied current to the neuron(top). Potential of the LIF neuron for the respective applied current (middle). Effect of changing bit depts of fractional part of fixed point is visible in the variations in spike timing of the neurons (bottom).

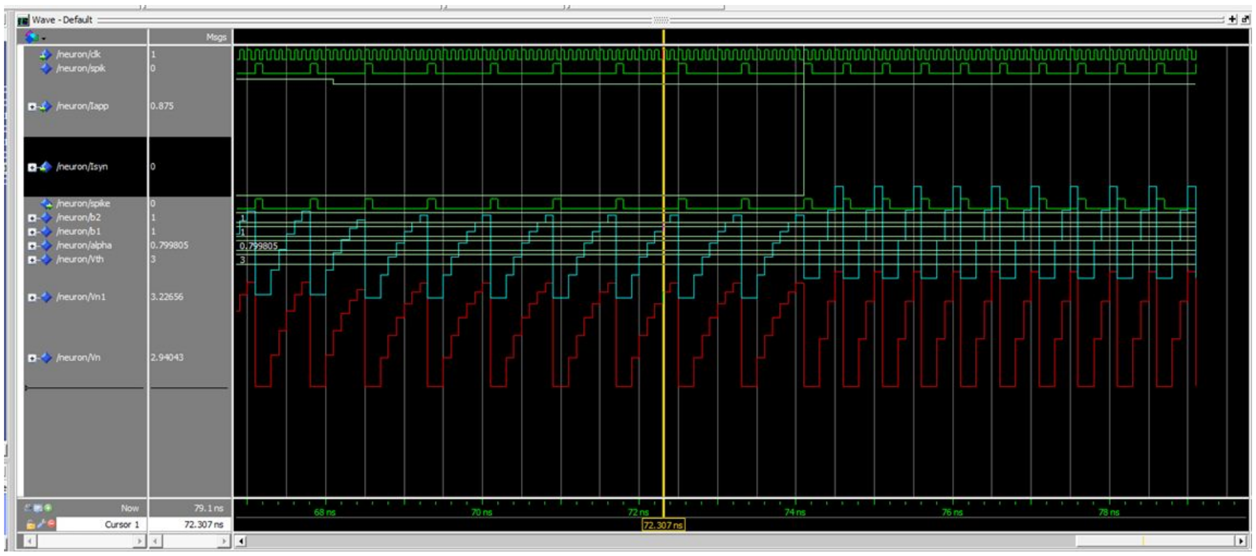


Fig. RTL simulation of the Neuron(soma) unit with some applied input current. Shown are the potential and spike waveforms in Modelsim

Synapse

The working of the synapse is represented by the following set of equations:

$$spike_{arrived} = delay(spike, timesteps)$$
$$v_{1_{n+1}} = \alpha_1 v_{1_n} + spike_{arrived}$$
$$v_{2_{n+1}} = \alpha_2 v_{2_n} + spike_{arrived}$$
$$v_n = v1_n - v2_n$$
$$I_{syn_{ij}} = \Sigma w_{ij} * v_n$$

The spike(arrived) signal is an input to the synapse which is fed to both the paths within the synapse. This is then added to two exponentials (represented by their decay constants alpha1 and alpha2). The difference of these two paths (v1-v2) is taken and multiplied by the weight coefficients Wij.

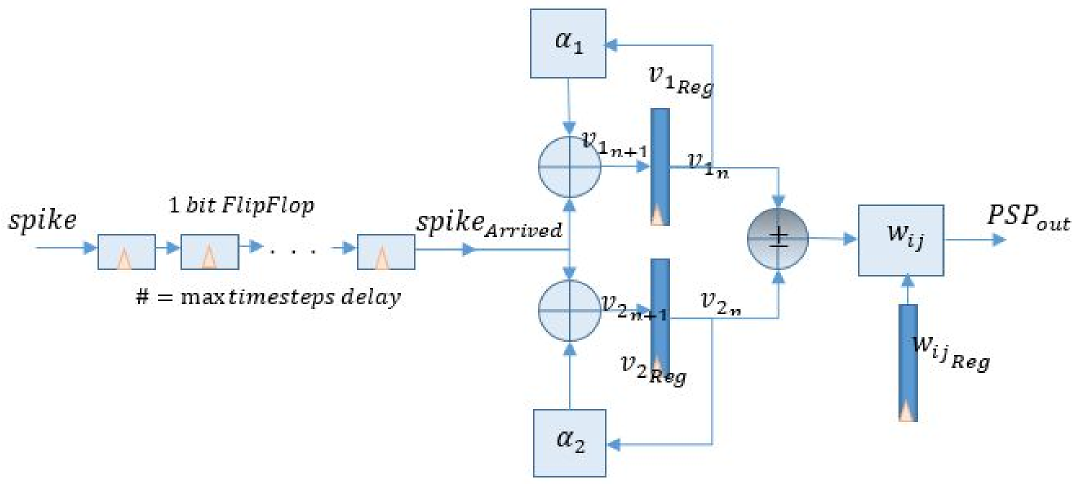
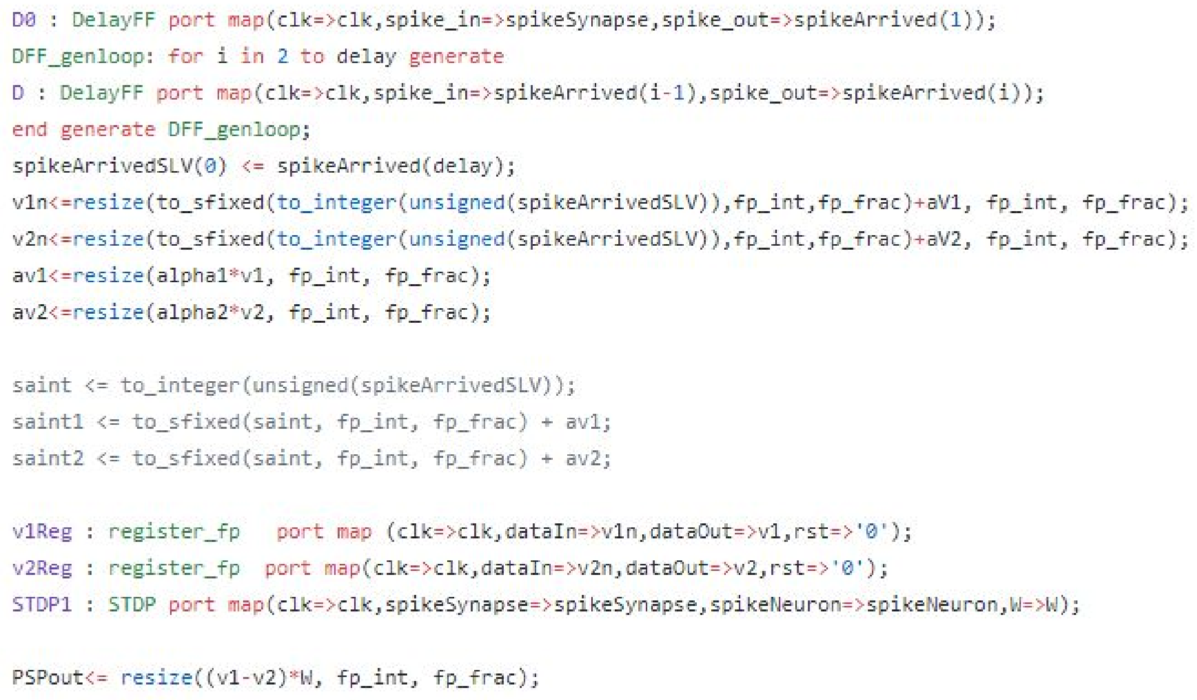


Fig. Digital block diagram for Synapse unit

This is the block diagram representation of the synapse.The output of the synapse PSPout is the post synaptic potential of the synapse.

The VHDL code for the synapse is in the synapse.vhd file



```
D0 : DelayFF port map(clk=>clk,spike_in=>spikeSynapse,spike_out=>spikeArrived(1));
DFF_genloop: for i in 2 to delay generate
D : DelayFF port map(clk=>clk,spike_in=>spikeArrived(i-1),spike_out=>spikeArrived(i));
end generate DFF_genloop;
spikeArrivedSLV(0) <= spikeArrived(delay);
v1n<=resize(to_sfixed(to_integer(unsigned(spikeArrivedSLV)),fp_int,fp_frac)+aV1, fp_int, fp_frac);
v2n<=resize(to_sfixed(to_integer(unsigned(spikeArrivedSLV)),fp_int,fp_frac)+aV2, fp_int, fp_frac);
av1<=resize(alpha1*v1, fp_int, fp_frac);
av2<=resize(alpha2*v2, fp_int, fp_frac);

saint <= to_integer(unsigned(spikeArrivedSLV));
saint1 <= to_sfixed(saint, fp_int, fp_frac) + av1;
saint2 <= to_sfixed(saint, fp_int, fp_frac) + av2;

v1Reg : register_fp   port map (clk=>clk,dataIn=>v1n,dataOut=>v1,rst=>'0');
v2Reg : register_fp  port map(clk=>clk,dataIn=>v2n,dataOut=>v2,rst=>'0');
STDP1 : STDP port map(clk=>clk,spikeSynapse=>spikeSynapse,spikeNeuron=>spikeNeuron,W=>W);

PSPout<= resize((v1-v2)*W, fp_int, fp_frac);
```

The floating point registers v1Reg and v2Reg are used to store the values of v1n and v2n. The resize, * and to_sfixed operations of the ieee_proposed library are used to perform the operations of the equations listed above.

Shown below is the fixed point simulation in Matlab for Synapse.

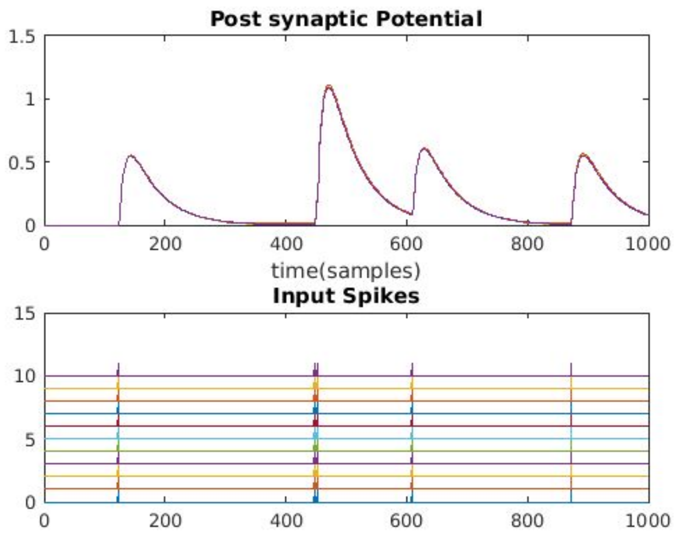
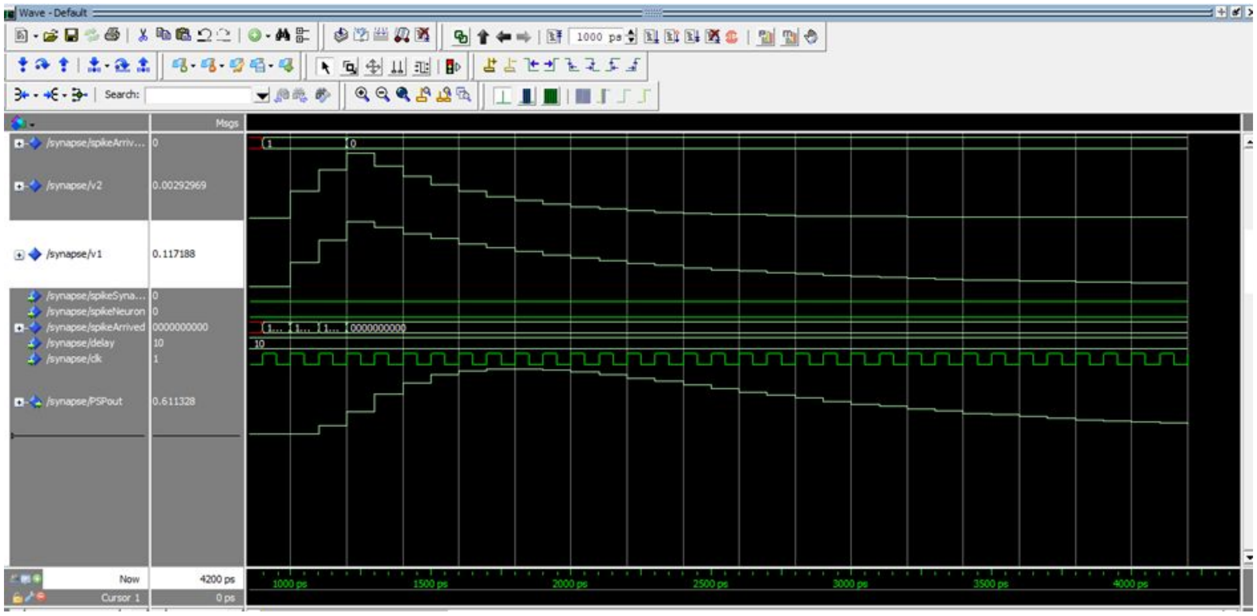


Fig. Fixed point simulation in Matlab to different depths of fractional bits (from 9,10,...,19 down to up). Shown the postsynaptic potential due to input spikes(top), and the input spikes(bottom)

We observe that post synaptical is effectively insusceptible to depths of fractional bits (post synaptic potential is almost the same for every depth shown).

The synapse also contains an STDP (spike timing dependent plasticity) unit which is described in the next section.



```
/synapse/spikeArriv...   0
/synapse/v2              0.00292969
/synapse/v1              0.117188
/synapse/spikeSyna...    0
/synapse/spikeNeuron     0
/synapse/spikeArrived    0000000000
/synapse/delay           10
/synapse/clk             1
/synapse/PSPout          0.611328
Now                      4200 ps
Cursor 1                 0 ps
```

Fig. RTL simulation of the Synapse given a input spike. Shown are the variables in registers v1 & v2 and the postsynaptic potential.

STDP (Spike Timing Dependent Plasticity)

Spike-timing-dependent plasticity (STDP) is a biological process that adjusts the strength of connections between neurons in the brain. The process adjusts the connection strengths based on the relative timing of a particular neuron's output and input action potentials (or spikes).

The relevant equations for STDP are below:

$$v_{3_{n+1}} = v_{3_n} + spike_{arrived}$$
$$v_{4_{n+1}} = v_{4_n} + spike_{Neuron}$$

$$\Delta w_{ij} = \pm v_{3_n}\delta(spike_{Neuron}) \pm v_{4_n}\delta(spike_{arrived})$$

Here is the block diagram for STDP:

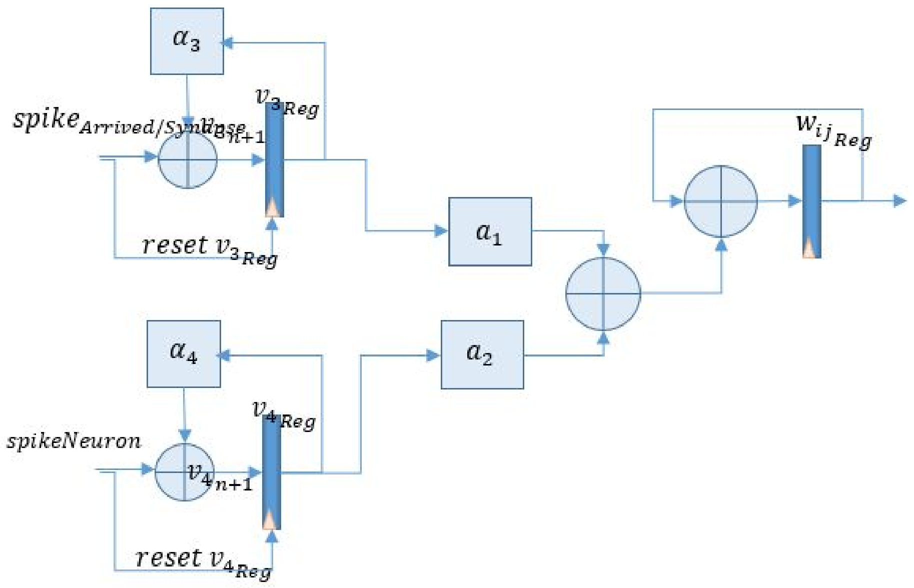
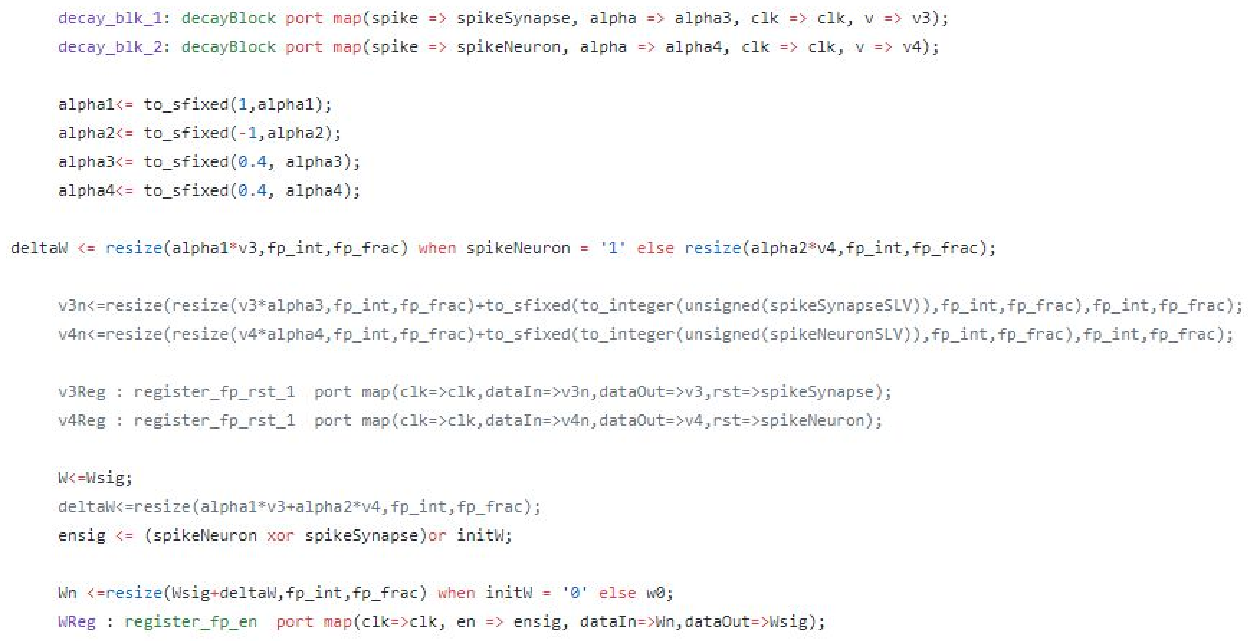


Fig. Digital block diagram for STDP unit

The code for the STDP is in the stdp.vhd file.



```
        decay_blk_1: decayBlock port map(spike => spikeSynapse, alpha => alpha3, clk => clk, v => v3);
        decay_blk_2: decayBlock port map(spike => spikeNeuron, alpha => alpha4, clk => clk, v => v4);

        alpha1<= to_sfixed(1,alpha1);
        alpha2<= to_sfixed(-1,alpha2);
        alpha3<= to_sfixed(0.4, alpha3);
        alpha4<= to_sfixed(0.4, alpha4);

 deltaW <= resize(alpha1*v3,fp_int,fp_frac) when spikeNeuron = '1' else resize(alpha2*v4,fp_int,fp_frac);

        v3n<=resize(resize(v3*alpha3,fp_int,fp_frac)+to_sfixed(to_integer(unsigned(spikeSynapseSLV)),fp_int,fp_frac),fp_int,fp_frac);
        v4n<=resize(resize(v4*alpha4,fp_int,fp_frac)+to_sfixed(to_integer(unsigned(spikeNeuronSLV)),fp_int,fp_frac),fp_int,fp_frac);

        v3Reg : register_fp_rst_1  port map(clk=>clk,dataIn=>v3n,dataOut=>v3,rst=>spikeSynapse);
        v4Reg : register_fp_rst_1  port map(clk=>clk,dataIn=>v4n,dataOut=>v4,rst=>spikeNeuron);

        W<=Wsig;
        deltaW<=resize(alpha1*v3+alpha2*v4,fp_int,fp_frac);
        ensig <= (spikeNeuron xor spikeSynapse)or initW;

        Wn <=resize(Wsig+deltaW,fp_int,fp_frac) when initW = '0' else w0;
        WReg : register_fp_en  port map(clk=>clk, en => ensig, dataIn=>Wn,dataOut=>Wsig);
```

Shown below is the Fixed point simulation of STDP unit in Matlab for different fixed points. We simulate for two kinds of conditions, one in which the register corresponding to v1 & v2, (in Fig.) is reset to 1 after each incoming spike, and other in which the registers do not reset, hence giving two different types of STDP rules which are shown as follows, for a1 = +0.1 , a2=-0.1, tau = 50 cycles for both

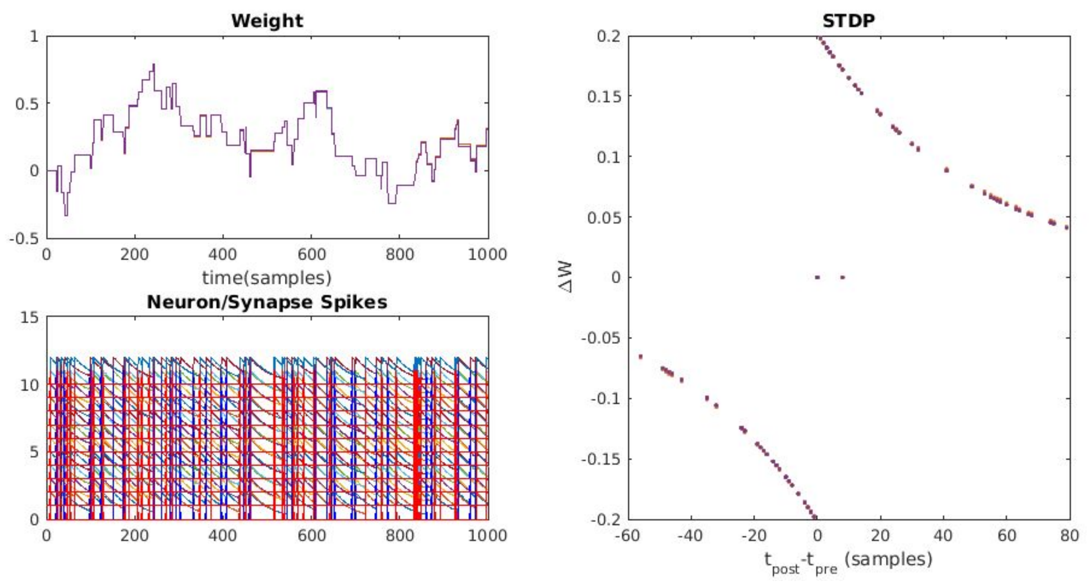


Fig. STDP unit behaviour when the registers are reset to 1 after incoming spike. Shown top left is the weight updates over time from two random poisson input spike trains. Bottom left shown is the spikes and corresponding v1 & v2 values. Right shown is the STDP behaviour.

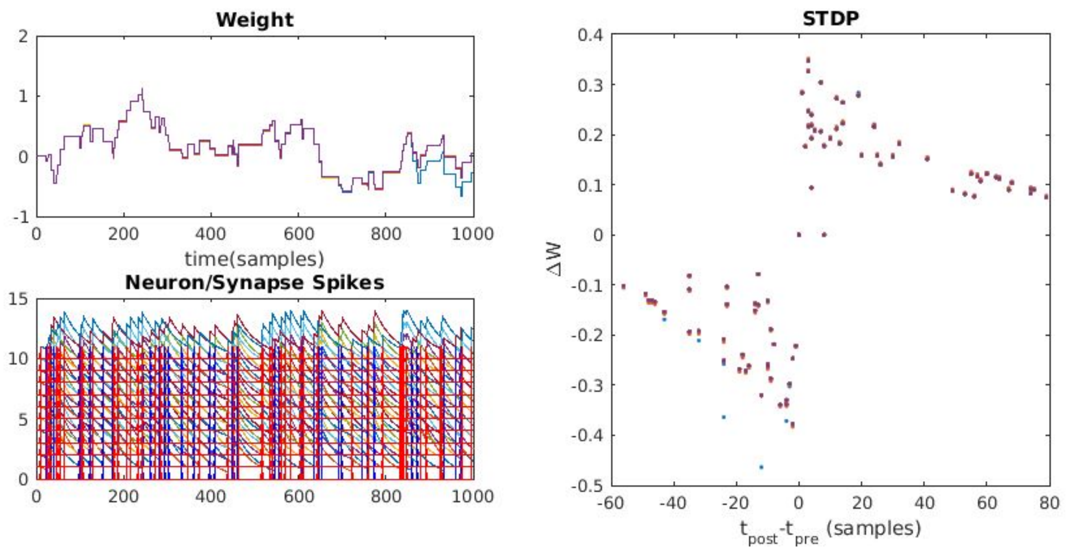


Fig. STDP unit behaviour when the registers do not reset after incoming spike Shown top left is the weight updates over time from two random poisson input spike trains. Bottom left shown is the spikes and corresponding v1 & v2 values. Right shown is the STDP behaviour.

Both the STDP behaviours are useful, but second one is more biological, as it represents the randomness in the weight update.

Optimized Design

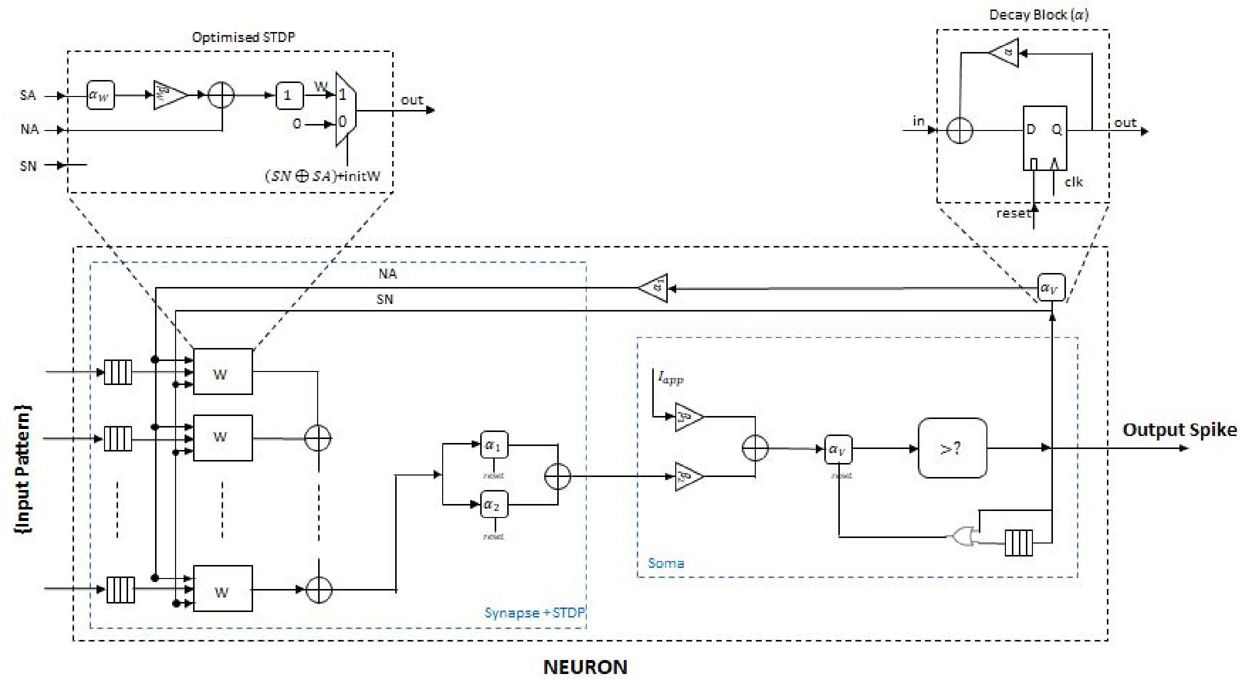


Fig. Block diagram of the optimized design for a neuron with synapses

To reduce the hardware costs, we were able to optimise it till where no performance is lost, considering the neural network parameters. These were possible because of the following considerations

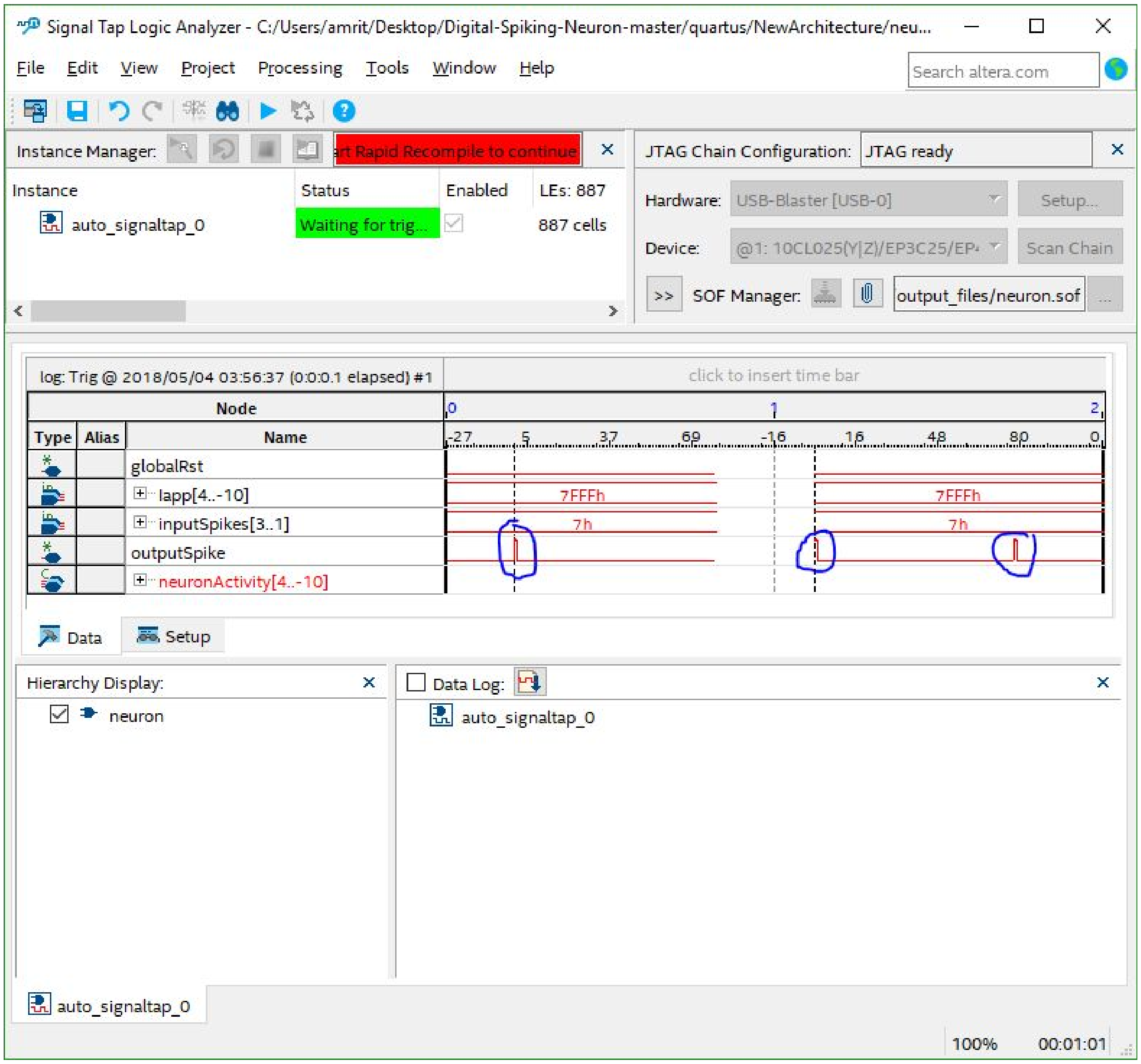
- Time constants of the synapse which are assumed to be same for all synapses for a neuron
- Assumed that STDP for anticausal events is same for all synapses for a neuron

This allows us to reduce the number of registers in STDP blocks by almost half, reduce the number of synapse registers almost by number of synapses for each neuron in the network.

Shown in Fig. The block diagram of the optimized neuron, which in turn consist unit of Soma , synapse and STDP. Figure also shows the block diagram of Decay Block and Optimized STDP block.

We show that our optimized design takes less hardware estate per synapse for large number of synapses which is evident from the optimization, and is one of the useful results, as generally a single neuron is generally connected by hundredths and thousands of synapses.

Running Signal-Tap for single neuron


Signal Tap Logic Analyzer - C:/Users/amrit/Desktop/Digital-Spiking-Neuron-master/quartus/NewArchitecture/neu...
File Edit View Project Processing Tools Window Help
Search altera.com
Instance Manager:
rt Rapid Recompile to continue
JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0]
Setup...
Device: @1: 10CL025(Y|Z)/EP3C25/EP4
Scan Chain
SOF Manager: output_files/neuron.sof
Data
Setup
Hierarchy Display:
neuron
Data Log:
auto_signaltap_0
auto_signaltap_0
100%
00:01:01

Instance	Status	Enabled	LEs: 887
auto_signaltap_0	Waiting for trig...	✓	887 cells

log: Trig @ 2018/05/04 03:56:37 (0:0:0.1 elapsed) #1

click to insert time bar

Type	Alias	Name	-27 … 0
		globalRst	
		lapp[4..-10]	7FFFh / 7FFFh
		inputSpikes[3..1]	7h / 7h
		outputSpike	
		neuronActivity[4..-10]	

Fig. Signal Tap output for running single neuron. A constant input current of 1.5 in value was given to the neuron, we can find that it spikes reappear after intervals of 78 units.

Scalability considerations

We have wrote a matlab script which generates a scalable VHDL code for Neural Networks using our optimized design.

Following figure shows the percent LUT usage in the FPGA (EP4CE22F17C6 DE0 Nano board)which contains total of 22,320 logic elements.

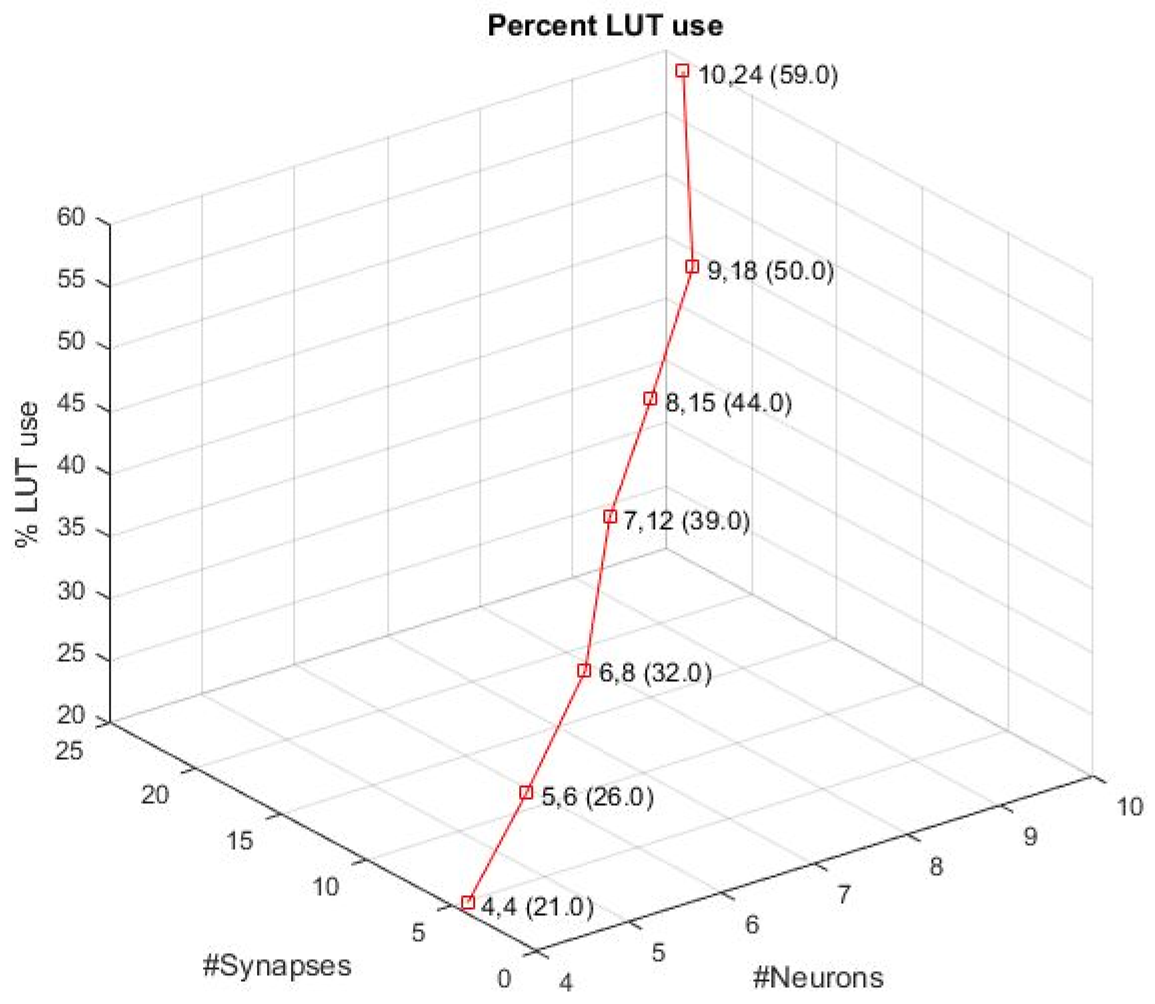


Fig. Percent LUT use for our model (given in brackets) after pair of numbers representing total number of neurons and synapses respectively (#neurons,#synapses) in the network

We estimate the percent occupied on our hardware by following relation, where N is the number of neurons and Nsyn is the number of synapses .

$$\%LUT_{DE0Nano} = 4.629N + 0.5051N_{syn}$$

$$LUT = 1033N + 112N_{syn} \text{ logic elements}$$

Hence cost per synapse is relatively small, but both the costs are relatively large for larger scale networks comprising of 1000’s of neurons.

<u>Goals Achieved</u>

- We have delivered a neural network code which is scalable and only limited by the estate requirements of the FPGA, and limitations of the compiler. Where each neuron can be configured for its desired parameters of operations

- We proposed a new optimized design, which reduces the hardware used
- Code can be generated using matlab script, which takes any arbitrary network as input

Future Work

- Since our FPGA runs at tens or hundreds of MHz, and our neuron model emulates realtime neuron (time evolution step) at 1KHz, we can, for this model, run it faster that 1000x for this kind of approach. To have real time processing, we can have 1000x more neurons/synapses by doing time multiplexing approach, where we have the shared resources, hence reducing the hardware costs and effectively more entities
- Reduce hardware costs also by maintaining the full parallelism as in real neurons, by effective implementation of fixed point multiplier or even a different approach in computing dynamics

Contributions from each team member

Ajinkya Gorad :

- Defined the initial block diagram for LIF neuron model with Soma, Synapse and STDP
- Wrote starting code for all the above model, which were to go under changes and corrections
- Made sure that weekly log is maintained by the team, and hence every progress we made whenever we worked
- Looked for ieee_proposed fixed point library which contributes to the essence of every building block in the design
- Found a workaround to wrapper based coding by manually forcing the signals in modelsim
- Figured out a way to optimize the hardware used in the initial design, common functional blocks were pulled out in the interest of economy
- Performed Matlab simulations for verifying the suggested model and its dependence on depth of fractional representation in bits, for Synapse, Neuron and STDP
- Wrote a script to write the VHDL code for arbitrary networks for our optimized model
- Worked on the final report, with sections including architecture introduction, matlab plots for fixed point simulations, section on optimized design

Amritesh Sharma :

- We decided on to use the ieee_proposed library, so wrote a very initial code for fixed_point adder, which was although not used in the final project but nevertheless helped understand the library functions.
    - Also through this came to know that generics and fixed_point are not compatible with simulations invoked in modelsim by testbench

    - After a discussion with TAs came to know that only std_logic interface is compatible
    - Wrote a wrapper for fixed_point adder and got its testbench working invokable through modelsim
- Subsequently wrote wrappers for each element of the code to provide an *std_logic* and *non-generic* interface with modelsim.
- The other workaround to writing wrappers was to manually force the signals in ModelSim. So, Automated the process of manually forcing the signals/ports using tcl/tk based ModelSim scripting
- Prepared Tutorial slides for IEEE_PROPOSED library and basic ModelSim Scripting
- Designed the optimal block diagram for the new optimised neuron.
- Wrote the VHDL code for optimised neuron and initial specific Neural Net of two neurons using the better design which was later scaled up the script generated by Ajinkya
- Worked on the final report's Biological background section and Appendix (project files organization and instructions on how to run the project.)

Shyam Ramamoorthy :

Worked on coding synapse.vhd
Worked on coding stdp.vhd
Created top level entity for testing the overall design
Worked to incorporate the ieee_proposed library into the project to make use of fixed point algebra
Helped in curing the error of 'could not find mytypes.vhd' during Modelsim simulations
Simulated and tested the neuron entity
Simulated and tested the synapse entity
Simulated and tested the top level entity
Worked on the final report, including the sections explaining the architecture and block diagram of the various components

Dipti Ranjan Sahu :

Worked on coding the Neuron entity- neuron.vhd
Worked on coding a wrapper for the entity neuron
Created testbench for neuron, simulated and tested the entity neuron
Checked and debugged the overall testbench and the overall design
Computed the hardware cost for different neural networks using MATLAB and quartus. Plotted the hardware cost with respect to number of neurons and synapse
Worked on completing the report

Team :

Team used Github for exchanging software and updating it with developed/better program whenever possible

Acknowledgement

We thank WEL lab for providing us the DEO Nano FPGA board.

Resources Generated

We have created some resources which we encouraged to be looked upon and could be used if useful for the user, apart from this work, they include

- Tutorial slides for using fixed poi nt using IEEE Proposed Fixed Point library
- Using Modelsim scripting which is better than testbenches, and allows use of generics to be played with, avoiding adding of wrappers
- Scalable VHDL network generator Matlab.m script file

References

1. Cassidy, Andrew, and Andreas G. Andreou. "Dynamical digital silicon neurons." *Biomedical Circuits and Systems Conference, 2008. BioCAS 2008. IEEE*. IEEE, 2008.
2. Wang, Runchun Mark, et al. "An FPGA implementation of a polychronous spiking neural network with delay adaptation." *Frontiers in neuroscience* 7 (2013): 14.
3. Caron, Louis-Charles, Frédéric Mailhot, and Jean Rouat. "FPGA implementation of a spiking neural network for pattern matching." Circuits and Systems (ISCAS), 2011 IEEE International Symposium on. IEEE, 2011.

Appendix

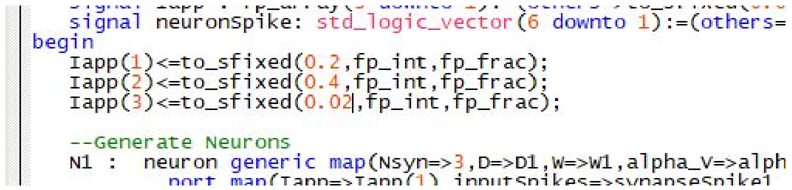
About the project files organization and instructions to run the project:

File Organization:

How to run the project(default)

Simulate fully connected network of 3 neurons (default Network)

- Go to quartus\NewArchitecture\ folder
- Load the project neuron.qpf from the provided project files
- Make sure that the ieee_proposed library has been included(if not follow the given IEEE_PROPOSED_LIBRARY tutorial)
- Also make sure myEntities.vhd, mytypes.vhd and W_STDP.vhd is included in the project from src project
- Make sure that the top level entity is neuron (Assignments->Settings->General)
- For demonstration purpose constant current input is given for each neuron

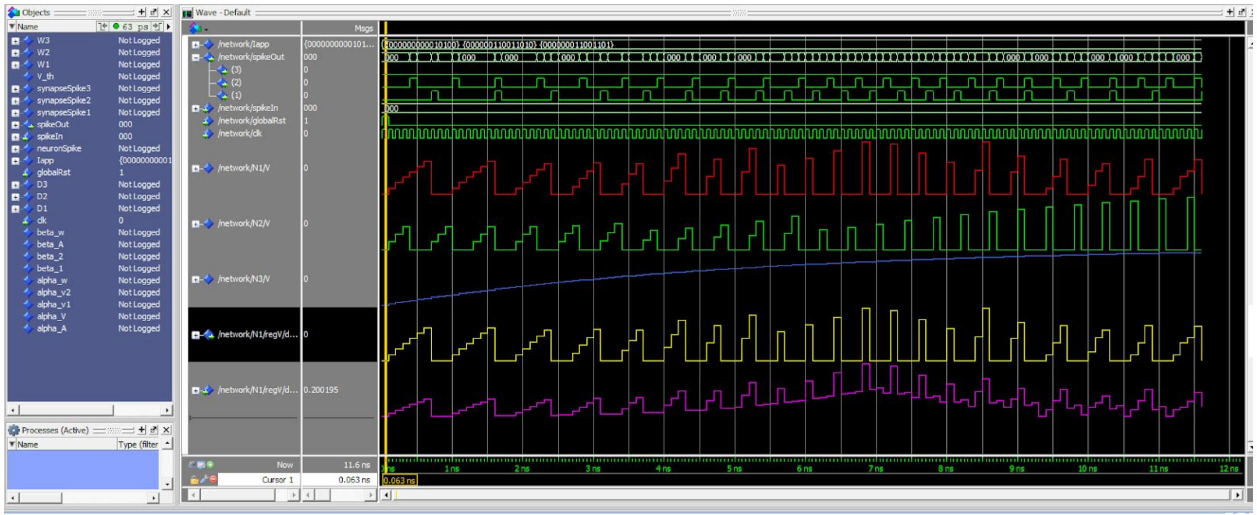


```
   signal neuronSpike: std_logic_vector(6 downto 1):=(others=
begin
   Iapp(1)<=to_sfixed(0.2,fp_int,fp_frac);
   Iapp(2)<=to_sfixed(0.4,fp_int,fp_frac);
   Iapp(3)<=to_sfixed(0.02,fp_int,fp_frac);

   --Generate Neurons
   N1 :  neuron generic map(Nsyn=>3,D=>D1,W=>W1,alpha_V=>alph
```

- Compile the project
- After compiling, run RTL simulation
- Copy the following modelsim script and paste it in Modelsim terminal

```
vsim rtl_work.network

add wave -position end  sim:/network/Iapp
add wave -position end  sim:/network/spikeOut
add wave -position end  sim:/network/spikeIn
add wave -position end  sim:/network/globalRst
add wave -position end  sim:/network/clk
add wave -position end  sim:/network/N1/V
add wave -position end  sim:/network/N2/V
add wave -position end  sim:/network/N3/V
add wave -position end  sim:/network/N1/regV/dataOut
add wave -position end  sim:/network/N1/regV/dataIn
add wave -position end  sim:/network/N1/SynapseGen_Loop(1)/Syn/Wn
force -freeze sim:/network/spikeIn 000 0
force -freeze sim:/network/globalRst 0 0
force -freeze sim:/network/globalRst 1 0
force -freeze sim:/network/clk 1 0, 0 {50 ps} -r 100
run
force -freeze sim:/network/globalRst 0 0
run 10000
```

- This will run the modelsim RTL simulation for the given project



Name	63 ps
W3	Not Logged
W2	Not Logged
W1	Not Logged
V_th	Not Logged
synapseSpike3	Not Logged
synapseSpike2	Not Logged
synapseSpike1	Not Logged
spikeOut	000
spikeIn	000
neuronSpike	Not Logged
Iapp	{00000000001
globalRst	1
D3	Not Logged
D2	Not Logged
D1	Not Logged
clk	0
beta_w	Not Logged
beta_A	Not Logged
beta_2	Not Logged
beta_1	Not Logged
alpha_w	Not Logged
alpha_v2	Not Logged
alpha_v1	Not Logged
alpha_V	Not Logged
alpha_A	Not Logged

	Msgs
/network/Iapp	{000000000101...
/network/spikeOut	000
(3)	0
(2)	0
(1)	0
/network/spikeIn	000
/network/globalRst	1
/network/clk	0
/network/N1/V	0
/network/N2/V	0
/network/N3/V	0
/network/N1/regV/d...	0
/network/N1/regV/d...	0.200195
Now	11.6 ns
Cursor 1	0.063 ns

Fig. Modelsim output after running the transcript (Follow ModelSimScript to convert binary signals to fixed point to visualize the result). Shown are the potential of 3 neurons, and their spike outputs.

To simulate single neuron, make neuron as top level entity.

How to run the project(any network)

- Load the project neuron.qpf from the provided project files
- Make sure that the ieee_proposed library has been included(if not follow the given IEEE_PROPOSED_LIBRARY.pptx tutorial)
- Open the matlab script generateNeuronNet.m in matlab folder. Define the directed edges (synapse) of the graph in network as X->Xn, provide corresponding delays and weights for each synapse in Tau and W respectively. Provide the input neurons and the output neurons. Neuron parameters can be changed in the script.
    - Run the script
    - Output file is script.vhd with entity of Network
- Include the code from script.vhd to neuronNet in current project
- Also make sure mytypes.vhd, myEntities.vhd, and W_STDP.vhd is included in the project from src project. Keep files in the order specified.
- Compile the project
    - There is one bug, that a neuron with single synapse, quartus is not capable of having single entity in an array, hence workaround is to append a one more value of similar kind. For eg.

(output)	(corrrected)
constant D1 :integerArray :=(1);	constant D1 :integerArray :=(1,1);

    - This is not being corrected in the script, as there may exist better alternatives to this kind of network generation method
    - Networks with more than 10 neurons take a while to compile
- After compiling, run RTL simulation
- Now one can manually force the signals and simulate the design or follow ModelSimScripting.pptx for script based simulations

-END-