# ASIC/FPGA Hardware Implementation of Izhikevich-based Spiking Neural Network Using CORDIC Algorithm

Thesis submitted in fulfillment of the requirements for
the award of the Bachelor Degree in
Nanotechnology and Nanoelectronics Engineering
By:

Abdelrahim Mohamed Elnabawy
Hussien Abdelrazik
Moatasem Moustafa
Mostafa Elbediwy

Supervisors:
Dr. Amr Helmy
Dr. Hassan Mostafa


B.Sc in Nanotechnology and Nanoelectronics Engineering
University of Science and Technology at
Zewail City for Science and Technology

JUNE 2018

## ACKNOWLEDGEMENTS

First of all, we are so grateful for working under the supervision of Dr. Amr Helmy and Dr. Hassan Mostafa. We would like to express our gratitude to the supervisors for their guidance during the whole project period. Dr. Amr Helmy is the first one to believe in us showing his support starting from the summer period of 2017. He dedicated some of his worthy time to meet us even outside the official time for graduation project meeting. In addition, he is the one who introduced this interesting topic to consider it as our graduation project. We are so thankful to him for his support during this long journey. Dr. Hassan joined us from the first day of the official graduation project time. Fortunately, He had a lot of experience and solid knowledge in this field, so he put us on the right track. Moreover, he supported us with contacts of research assistants and technical engineers to help us during our project. So, we have to admit that we gained a solid experience from working under Dr. Hassan supervision. Furthermore, both supervisors helped us to get two funds from NTRA and ITIC and publish our first conference paper in NEWCAS conference. It was a great pleasure working under the supervision of both, Dr. Amr and Dr Hassan. It was an exciting journey that we managed to go through it thanks to them.

We are delighted to be supported from Dr. Hatem Fayed and Dr Ahmed Abd Elsamea from the Mathematical Department at Zewail City for Science and Technology. They invited us to meet in their offices. In spite of their busy schedule, we brainstormed and discussed several ideas during the meeting. They gave us a mathematical point of view to start from. So, we would like to thank them for their encouragement and support. We also would like to thank Dr. Amr Bayoumi, the Nanotechnology Department Director at Zewail City for Science and Technology, for providing us with all technical facilities during our project. He provided us with a separated electronics lab supplied with all the required tools such as FPGA kits, oscilloscopes, power supplies, Analog kits and Breadboards. We would like to show our gratitude to Eng. Ahmed abd El Rabou and Eng. Kirollos Ernest as they were the responsible engineers for our lab. We also want to thank our colleagues for their support during the previous year by their encouragement, knowledge and experience.

We would like to convey our thankful feelings for Eng. Salma Hassan Sayed, Research Assistant from Cairo University, for helping us to start our track in this project. She provided us with helpful resources to understand the deep learning concepts and the neural network types. Also, Eng. Noha Gamal from Mentor Graphics shared with us her experience with ISE tool to program the FPGA kit. So, we would like to offer both of them special thanks for their support and guidelines.

Finally, we would like to express our sincere appreciation to Dr. Shady Agwa. He dedicated several hours of his valuable time to debug and solve a problem we faced in the FPGA kit. He did not leave us until the problem is solved.

## PUBLICATION

We received a notification that our conference research paper is accepted to be published in NEWCAS conference held in Montreal, Canada on June 24-27 2018.

A. Elnabawy, H. Abdelmohsen, M. Moustafa, M. Elbediwy, A. Helmy and H. Mostafa, "A Low Power CORDIC-Based Hardware Implementation of Izhikevich Neuron Model", in *IEEE International NEWCAS*, Montréal, Canada, 2018, In Press.

## FUNDING AWARDS

This project is awarded two funding grants of 10,000 L.E each from:

i. National Telecom Regulatory Authority (NTRA).


ii. Information Technology Academia Collaboration (ITAC) Program at Information Technology Industry Development Agency (ITIDA).


## ABSTRACT

The project presents a significant contribution to the design and Very Large-Scale Integration (VLSI) implementation of Spiking Neural Networks (SNN) with low power consumption and better area utilization. The project aims to approximate a certain neuron model which is the building block of the neural network to reduce the network complexity while maintaining an adequate level of accuracy. Furthermore, the thesis contributes with original novel work to the neuromorphic computing field which is briefly explained below.

In this project, an efficient CORDIC-based hardware implementation of the Izhikevich neuron model is introduced. The CORDIC (COordinate Rotation Digital Computer) algorithm is used to approximate the square term in Izhikevich equations that describe the neuron response. The approximation is evaluated by defining four types of errors where the CORDIC approximation shows significant improvement in error performance compared to the Piecewise Linear (PWL) model [1]. Two additional approximation algorithms, Integral Sum and Iterative Logarithmic, have been proposed other than CORDIC algorithm. Yet, CORDIC algorithm has been proved the most accurate one.

For ASIC flow, the power consumption of the CORDIC-based neuron hardware implementation ranges from 0.26 mW to 0.4 mW whereas the PWL-based neuron as well as the original Izhikevich neuron hardware implementations consume 0.3 mW and 1.06 mW, respectively. A Figure of Merit (FoM) is defined to show the trade-off among errors, power and area. By comparing with the PWL-based neuron hardware implementation, it is found that the CORDIC-based model is preferred as an approximation method from FoM perspective.

In order to further investigate the performance of the CORDIC-based approximation of the neuron model against other approximation models, the different approximations of the Izhikevich neuron model have been implemented on Xilinx ZYNQ-7 ZC702 Evaluation FPGA Board. The original Izhikevich neuron model has exhibited a high-power consumption (3.73 mW) and number of LUTs (1030). Both the

CORDIC-based neuron model and the PWL-based neuron model performed significantly better than the original model from power and area perspective. Although the PWL-based neuron model consumed less power than the CORDIC-based neuron, the latter used a lower number of LUTs.

Furthermore, a feedforward neural network of two layers with 210 neurons in total (200,10) is simulated using the original Izhikevich neuron model, the PWL-based neuron as well as the CORDIC-based neuron. MNIST dataset is used to train and test the network [2]. The original Izhikevich neuron-based network achieves an accuracy of 89% while the CORDIC-based network achieves a better accuracy (86.5% to 88%) than the PWL-based one (85.5%).

Finally, real time FPGA implementation outputs are monitored using an oscilloscope for the purpose of behavioural verification. Spartan-6 FPGA SP605 Evaluation Kit is programmed with a single CORDIC-based Izhikevich neuron and connected to an oscilloscope in Digilent Analog Discovery 2 kit. It is found that the CORDIC-based Izhikevich neuron can exhibit the tonic spiking behavior correctly.

## TABLE OF CONTENT

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF SYMBOLS

| | |
|---|---|
| $w$ | Connection weight |
| $I$ | Neuron input current |
| $C_m$ | Membrane capacitance |
| $V_m$ | Membrane voltage |
| $V_{th}$ | Threshold voltage |
| $t_{ref}$ | Refractory period |
| $v$ | Membrane potential |
| $u$ | Recovery variable |
| $a$ | Izhikevich model parameter |
| $b$ | Izhikevich model parameter |
| $c$ | Izhikevich model parameter |
| $d$ | Izhikevich model parameter |
| $\Delta t$ | Time step |
| $m$ | Limit on the integer term |
| $n$ | Limit on the Fraction term |
| $ERRp$ | Local minimum error |
| $ERRt$ | Time difference error |
| $MERRt$ | Mean time difference error |
| $\sigma$ | Sigmoid function |

## LIST OF ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| SNARC | Stochastic Neural Analog Reinforcement Calculator |
| NN | Neural Network |
| ANN | Artificial Neural Network |
| CNN | Convolutional Neural Network |
| SNN | Spiking Neural Network |
| CPU | Central Processing Unit |
| GPU | Graphical Processing Unit |
| ReLU | Rectified Linear Units |
| LIF | Leaky Integrate and Fire |
| HH | Hodgkin-Huxley |
| FM | Frequency Modulated |
| DAP | Depolarized after-potential |
| IF | Integrate and Fire |
| FPGA | Field Programmable Gate Array |
| ASIC | Application Specific Integrated Circuit |
| IOT | Internet of Things |
| PWL | Piece-Wise Linear |
| CORDIC | COordinate Rotation Digital Computer |
| ILM | Iterative Logarithmic Method |

| | |
|---|---|
| HDL | Hardware Description Language |
| AdEx | Adaptive Exponential |
| MAE | Mean Absolute Error |
| RSEE | Relative Spike Energy Error |
| FoM | Figure of Merit |
| GM | Geometric Mean |
| AM | Arithmetic Mean |
| $GM_W$ | Weighted Geometric Mean |
| LUT | Look Up Table |
| DSP | Digital Signal Processor |
| MNIST | Modified National Institute of Standards and Technology |
| NIST | National Institute of Standards and Technology |
| GPIO | General Purpose Input/Output |
| USD | United States Dollar |
| CAGR | Compound annual growth rate |

# CHAPTER 1

# Introduction

## 1.1 The definition of Artificial Intelligence

Human beings are distinguished from other living things by their intelligence. As a society of intelligent creatures, we are capable of thinking, planning, learning, problem solving and many other activities that stimulate our minds.

Inspired by human brain, scientists and engineers are trying to understand how the brain works, mimic its functionality and build an artificial one that provides machines with the ability to think and decide. Started early in the 50's, Artificial Intelligence, abbreviated as "*AI*", is one of the recently established fields in science and engineering. *AI* is a new branch of Computer Science that aims to build intelligent machines that can think, learn and manipulate [3].

Traditional machines are instruction-based. By following restrict instruction, they can perform a certain task or function. Instruction-based machines produce a predicted well-defined output. Using AI, machines can learn, think and decide how the output should look like. Such a capability enables the machines to perform complex tasks such as recognition and classification.

## 1.2 The history of Artificial intelligence

The first attempt that could be considered as AI was performed by Warren McCulloch and Walter Pitts (Figure 1.1 (a)) in 1943. Their knowledge was based on information from physiology, proposition logic and Turing's theory of computation. They managed to create the first artificial neuron model that described how the neuron behaved. Their model was the early start of building artificial neural networks later [3].

In 1949, Donald Hebb (Figure 1.1 (b)) proposed a rule, called "*Hebbian learning rule*", that governed the connection strength between neurons. His rule, states that the neural connection strength is increased if the neurons fire simultaneously. Hebb's rule is a fundamental rule in AI that is used till now [3].

After a year, Marvin Minsky (Figure 1.1 (c)) managed to build a computer based on neural network with the help of Dean Edmonds. *SNARC* (Stochastic Neural Analog Reinforcement Calculator) is a neural network machine that simulates a network of 40 neurons. Their work is considered the first AI-based product [3].

One of the earliest pioneers in the 50's is Alan Turing (Figure 1.1 (d)). His inspiration paved the road for scientists and engineers to develop and flourish the field of AI. His contributions to AI field included, but not limited to, the Turing Test, machine learning, genetic algorithms, and reinforcement learning [3].

The official birth of Artificial intelligence was in 1956 when John McCarthy (Figure 1.1 (e)) used the term "Artificial intelligence" in his speech when he invited inspired scientists to a 2-month workshop. During this workshop, they tried to establish the basics of how machines could learn, think and solve problems [3].


Figure 1.1 The founders of Artificial Intelligence. (a) Warren McCulloch. and Walter Pitts. (b) Donald Hebb. (c) Marvin Minsky. (d) Alan Turing. (e) John McCarthy.

## 1.3 Classification of Artificial intelligence

After several decades, the science of AI has developed significantly to include different aspects. AI is divided mainly into two categories, Symbolic Learning and Machine Learning, as shown in Figure 1.2. The difference between the two categories is in the way of learning that the former is symbolic-based learning while the latter is data-based learning. In the Symbolic Learning, the machines learn the ability to move, avoid obstacles in the Robotics and recognize objects in computer vision. This aspect requires previous knowledge from Image Processing field [3].

The other main aspect is Machine learning that is mainly concerned about recognizing patterns. Machine learning is divided into two main topics, Statistical learning and Deep Learning or Neural Networks. In statistical learning, machines learn how to understand and recognize voices as well as natural languages. Deep learning builds a network of neurons that are capable of doing a certain function. Neural Network (NN) includes, but not limited to, three types which are Artificial Neural Network (ANN),

Convolutional Neural Network (CNN) and Spiking Neural Network (SNN) which are discussed in details in the next chapter [3].

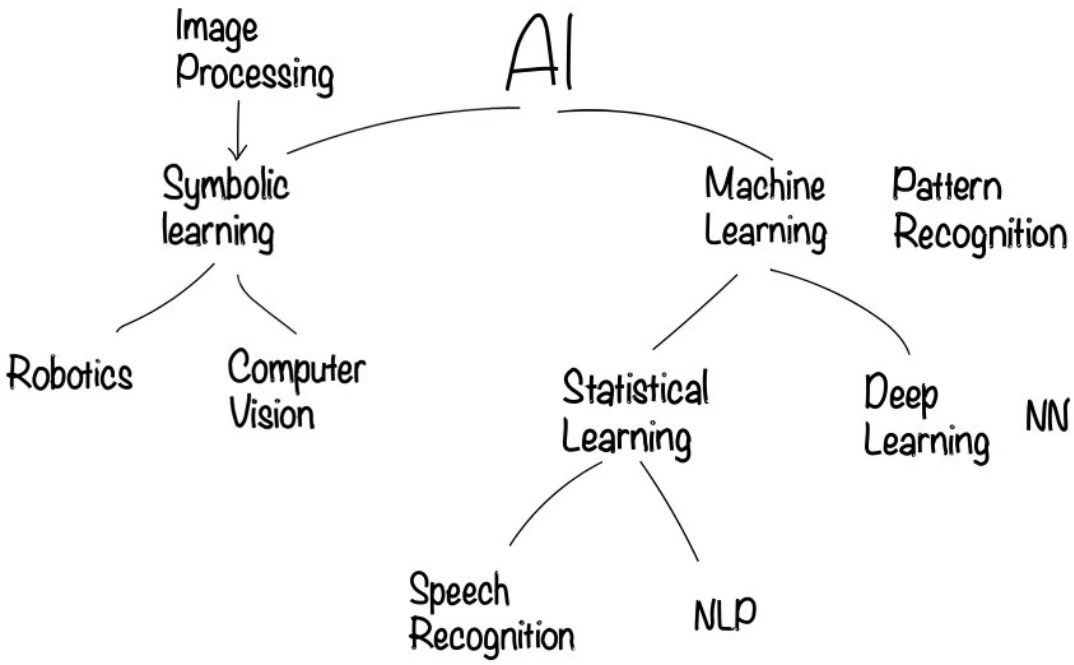


Figure 1.2 Classification of Artificial Intelligence.

## 1.4 Software versus Hardware- based Artificial Intelligence

Recently, A lot of great applications have been implemented as a software program. Software-based AI has accomplished great achievements; however, software has limitations. It is limited by the power consumption and CPU/GPU speed. As an alternative, companies started to focus on implementing AI as a hardware product [4].

Several recent commercial hardware products have been revealed. Companies moved towards hardware to take advantage of higher speed and lower power consumption. This turn will require exploring new hardware architecture than the traditional ones. A lot of research will be conducted to explore new custom hardware that is specialized for Artificial Intelligence to hit the performance limits [4]. The hardware-based AI growth and its market share are discussed in details later on in the economic analysis chapter.

# CHAPTER 2

# Literature review

## 2.1 The Biological neuron:

The biological neuron is the building block of our human brain. The brain consists of billions of neurons that are connected together to form a neural network responsible for thinking and making decisions. The biological neuron consists of three main components as shown in Figure 2.1 [5]:

a) Dendrites: receives signals from neighbour neurons and synapses.
b) A cell body (Soma): processes the signals received.
c) An axon: sends processed signals out to other neurons.

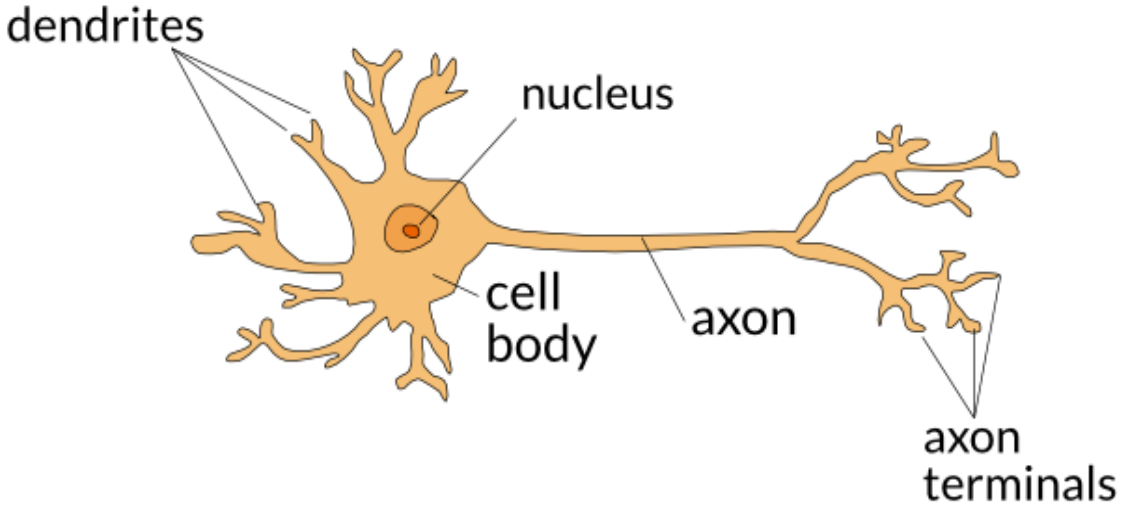


Figure 2.1 The biological neuron.

## 2.2 The artificial neuron:

Mimicking the biological neuron, the artificial neuron also has a number of inputs, a processing block and an output that can be connected to multiple neurons as shown in Figure 2.2. Different neuron and neural network models differ in the design of the processing block; however, the main concept is the same and similar to that of the biological neuron. The artificial neuron as well as the neural network will be discussed in details later in this chapter [6].

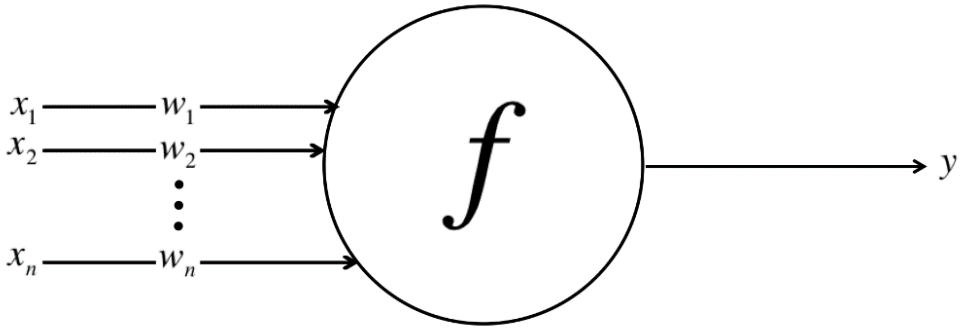


Figure 2.2 The artificial neuron.

## 2.3 Types of Neural Networks

Neural networks are one of the basics principles in the field of Artificial Intelligence [5]. There are several types of neural networks including, but not limited to, Artificial Neural Network (ANN), Convolutional Neural Network (CNN) and Spiking Neural Network (SNN).

### 2.3.1 Artificial Neural Network (ANN)

ANN is considered as a basic computing system inspired by the biological neural network. The basic component of ANN is the artificial neuron. Those neurons are implemented to mimic the computation process done inside the soma of the biological neuron. ANN is a matrix constructed by interconnecting many of those artificial neurons. Artificial neurons differ from each other according to the computation function used. To understand how the neuron works, a simple form of the artificial neuron, called the Perceptron, is illustrated below in Figure 2.3 [5].

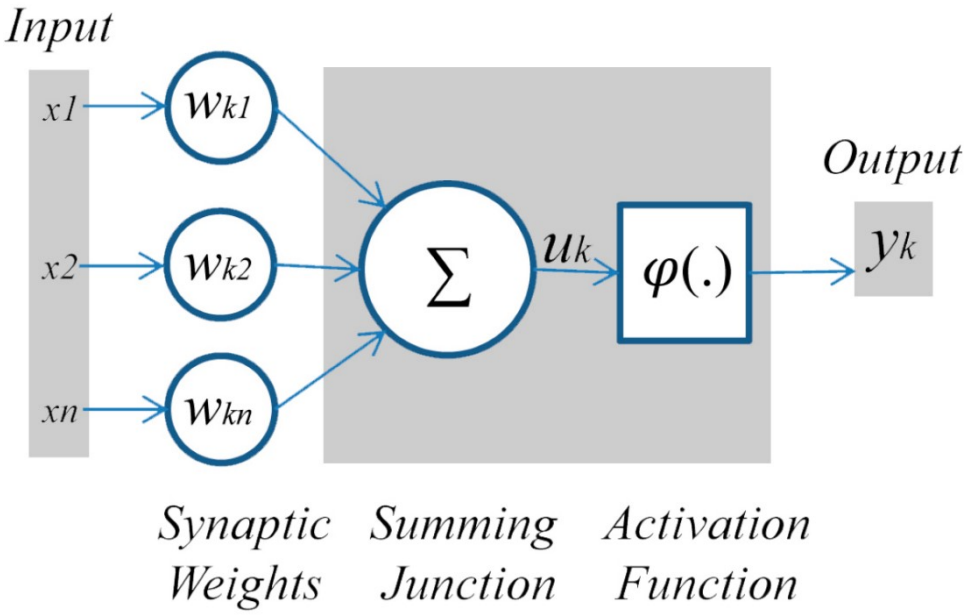


Figure 2.3 The perceptron.

The perceptron is considered as an early version of the sigmoid neuron. It takes several inputs and produces a single output as a binary digit "0" or "1". Each input has a weight that represents the input strength. Inputs are multiplied by their weights. The output of the perceptron is determined by summing all the weighted inputs. If the summation is greater than the threshold, the result is logic 1 and if it is less than the threshold, the result is logic 0. The threshold is a neuron-related parameter used to limit the firing. The perceptron mathematical form is represented in (1) as follows [7]:

$$output = \begin{cases} 0 & \sum_j w_j x_j \leq threshold \\ 1 & \sum_j w_j x_j > threshold \end{cases} \quad (1)$$

Where $x_j$ represents the input's value and the $w_j$ represents the weight's value of neuron $j$.

On the other hand, the perceptron model is a simple form of the neuron that it has some drawbacks. it is unrealistic to mimic the biological neuron as a binary number either "1" or "0". In addition, the function is very sensitive that the output can flip from "1" to

"0" or the opposite if a slight change in the inputs or the weights occurs. This has led to develop the sigmoid neuron [7].

Sigmoid neuron is the most commonly used neuron in ANN. Sigmoid neuron is considered an improved version of the perceptron. The aim is to overcome the sensitivity problem of the perceptron. Consequently, any small change in the inputs or the weights will result in a small change in the output. The sigmoid neuron looks like the perceptron in its structure that it takes many inputs and produces a single output. Unlike the perceptron, the output will not be in a binary form. Instead, the output can take any value between 0 and 1. The sigmoid function is defined in (2) as follows [7]:

$$\sigma(w.x + b) = \frac{1}{1 + e^{-(\Sigma_j w_j x_j + b)}} \tag{2}$$

Where $b$ is an overall bias applied to the neuron.

The sigmoid function formula looks difficult, but it is similar in shape to the perceptron when plotted. The perceptron is plotted as a step function, but the sigmoid function is plotted in Figure 2.4 as follows [7]:

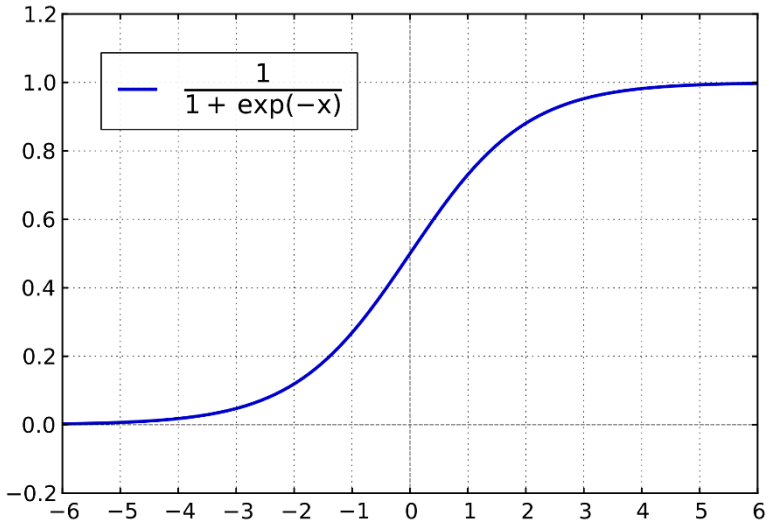


Figure 2.4 The sigmoid function.

The architecture of ANN is divided into several layers. Each layer has its name and function. The leftmost (first) layer is called the input layer because it is the layer that deals with the inputs directly. The rightmost (last) one is called the output layer because it is the one that produces the output. Any middle layer is called a hidden layer. The network could have many hidden layers or no hidden layer at all. Each neuron in any layer is connected to all neurons in the next layer as well as the previous layer. Such a structure is called feedforward that will be discussed in details later on this chapter. The network is represented in Figure 2.5 as follows [7]:

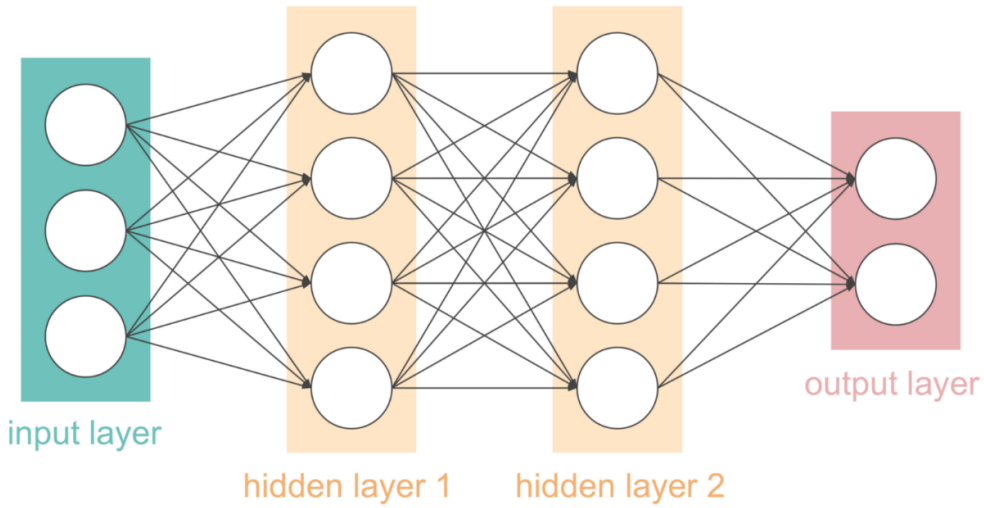


Figure 2.5 Feedforward structure of ANN.

Those layers do some computation according to the applied input and then, send their results to the next layer which performs the same process until the final network output is ready at the output layer [7].

### 2.3.2 Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) is one of the currenlty used type of neural networks. CNN is mainly designed for image recognition applications. This type of neural network is inspired biologically from the visual cortex. The input of this network is always an array of pixels' values. The number of pixels is dependent on the resolution and the size of the image. The value of the pixel is dependent on the features inside the image. The output of the CNN is a probability of some classes. CNN has a distinct structure not like any other neural network as shown in Figure 2.6 [8].

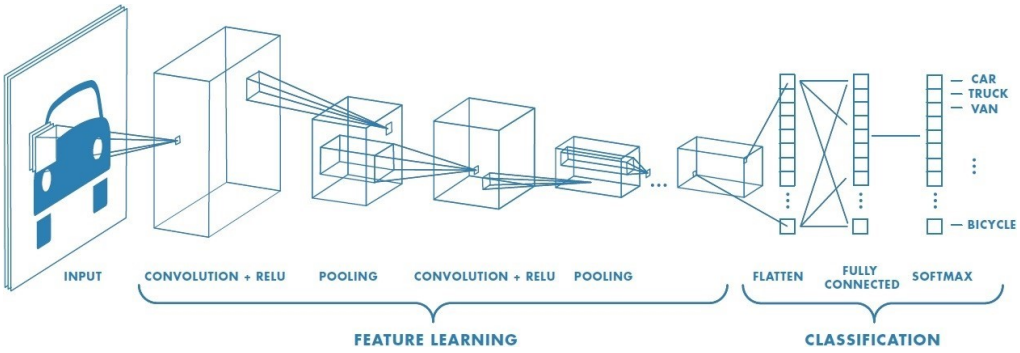


Figure 2.6 Convolutional Neural Network (CNN).

The structure of the CNN consists of a series of layers including, convolutional layer, Nonlinear layer, Pooling layer, Dropout layer and Fully connected layer. Each layer has its functionality and could be replicated more than one time [8].

The convolutional layer is directly connected to the input image. This layer is considered as a filter to scan a specific feature in the image. The filter is an array of weights applied to the inputs. The size of the chosen filter is compared with the original image and the desired resolution. The filter is moved to scan the whole image. The output of the filter is only one pixel. Accordingly, the image size will decrease at the end. Such an operation is called activation map illustrated in Figure 2.7 [8].

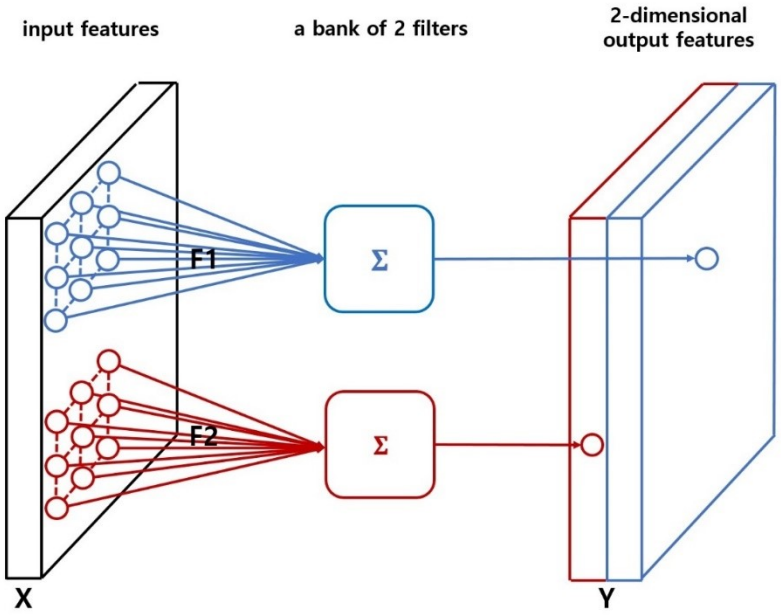


Figure 2.7 Activation map.

Rectified Linear Units (ReLU) layer is a nonlinear layer. This layer is interspersed between these convolutional layers. They provide nonlinearity, preservation of dimension and improve robustness. The ReLU layer applies the function f(x) = max (0, x) to all of the elements in the input volume [8].

Pooling layer is considered a down sampling layer as shown in Figure 2.8. This layer uses a filter to choose and compute the maximum-valued pixel. Moreover, pooling layer is used to control the overfitting problem. Such a problem occurs when the network cannot produce a correct output when a test set is applied. The reason is that training set makes the network over-biased [8].

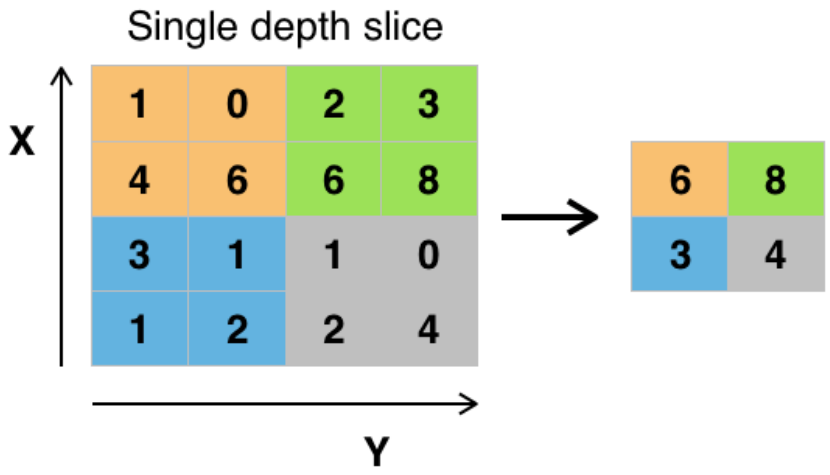


Figure 2.8 Pooling layer.

Dropout layer is used during the training time, but never used during the test time. This layer performs a specific function in the CNN where it “drops out” a random set of activations in that layer by setting them to zero. It helps in solving the problem of overfitting [8].

Fully Connected Layer works as a network itself. It works by taking the output of the previous layer then, determines which feature is the most correlated one to a particular class [8].

### 2.3.3 Spiking Neural Network (SNN)

Spiking Neural Network (SNN) is considered as the third generation of neural networks where the level of the realism has been increased. SNN can do not only recognition, but also data analysis and learning. The spiking neuron is the main component of SNN where it mimics the neuron of human's brain. Like the perceptron and sigmoid neurons, the spiking neuron can take many inputs and produce one output. The input of the spiking neurons is a current and the output is a train of spikes. This type of neuron is not supposed to fire each propagation cycle, but only when the membrane potential exceeds a specific value [9].

The behaviour of the spiking neuron is shown in Figure 2.9. when the neuron's membrane voltage is less than the threshold, the voltage accumulates till it reaches the threshold as long as a stimulus current is applied. when the membrane voltage reaches the threshold, the membrane voltage overshoots until it reaches the peak voltage, which is the maximum allowable value for the membrane voltage. Then, the membrane voltage drops down to a negative value during the refractory period until it reaches its resting state at the end [9].

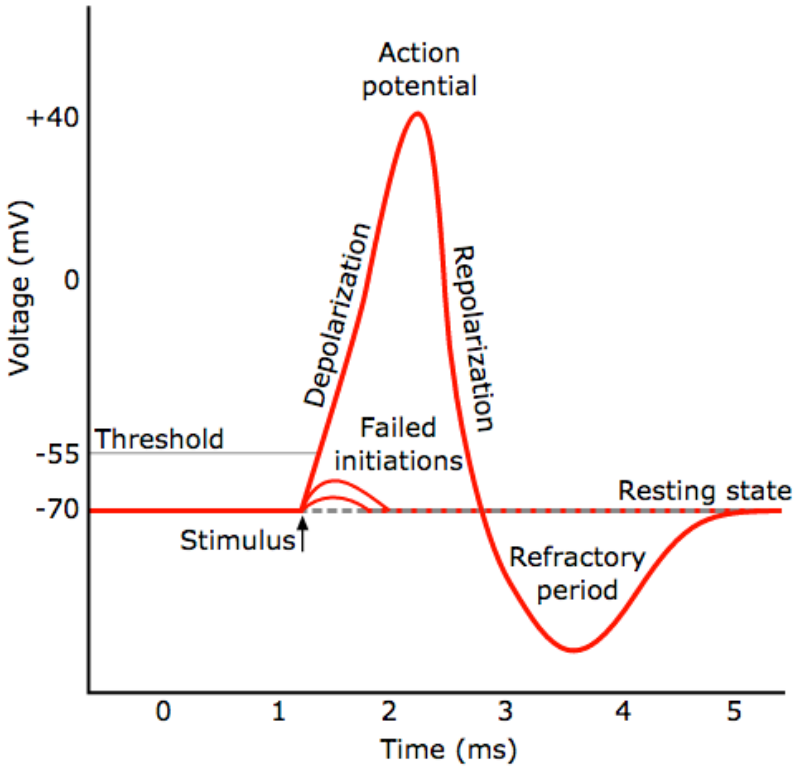


Figure 2.9 The behaviour of the biological neuron.

Each spiking neuron consists of three main computational stages as shown in Figure 2.10. The first stage is the sum of all inputs' currents. The second one does a certain computation. It integrates the output of the first stage over the time. The third stage is responsible for emitting the spikes and resetting the value of the membrane potential after a spike is fired [9].

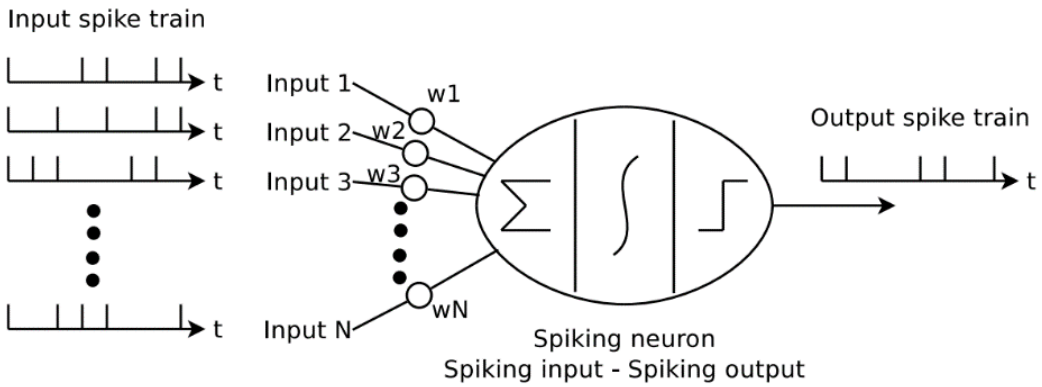


Figure 2.10 The spiking neuron's main components.

There are several mathematical models for the spiking neuron such as Integrate and Fire (IF) model, Hodgkin-Huxley (HH) model and the Izhikevich model. Each model has advantages and disadvantages. The IF model is very simple, but it does not produce an accurate neural behaviour. The HH model can produce a very accurate neural behaviour, but it is very complex. The Izhikevich model is considered as the trade-off between the two previous models that it can produce an accurate behaviour while being quite simple [10].

The structure of the SNN is not different from the structure of the ANN. It consists of layers connected to each other including, input layer, hidden layer(s) and output layer. Also, there could be more than one hidden layer or no hidden layer at all.

### 2.3.4 Neural Network Comparison

As explained before, each type of neural network has its advantages and disadvantages. ANN is the simplest one from implementation perspective since its neuron model is the simplest. In addition, ANN accuracy reached high levels. However, ANN cannot mimic the biological neuron and produce accurate neural behavior. Although CNN is compatible with image recognition applications, CNN is more complex than ANN. The SNN is the highest complexity compared with ANN and CNN. However, SNN is the most capable network to mimic the human brain behavior. In addition, SNN is the only type of neural networks that introduced the concept of the time. All previous advantages make SNN the most promising neural network towards more realistic applications. A lot of research has to be done in order to decrease SNN complexity.

## 2.4 Different spiking behaviours of the biological neuron

The Biological neuron is capable of exhibiting different spiking behaviours in response to various DC currents. Although the interactions between billions of neurons results in tremendous spiking behaviours, there exist 20 prominent spiking behaviours that are known and used in creating artificial neural networks [11]:

A) **Tonic Spiking:** The neuron is inactive in general unless the input current is on. In that case, the neuron fires a train of spikes. The neuron fires two rapid spikes at the beginning then, it fires a spike once every certain constant period [11].

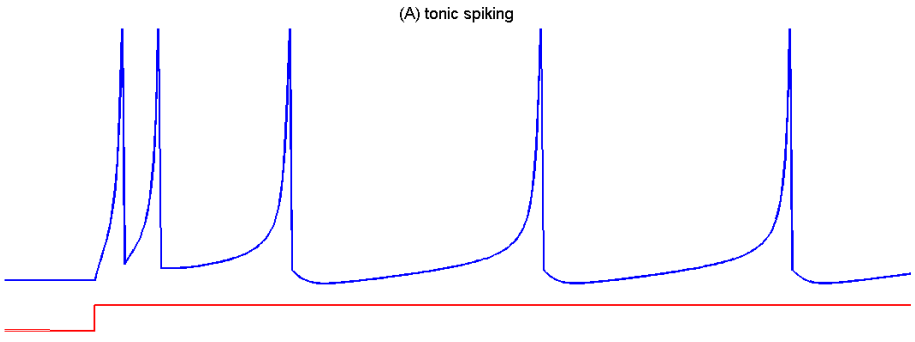


Figure 2.11 Tonic Spiking.

B) **Phasic Spiking:** The neuron fires only once when the input current is turned on then, it becomes inactive again, whether the input current is still on or changed to

off. It is useful in detection of the current stimulation start. The inter-burst frequency may get as high as 50 Hz [11].

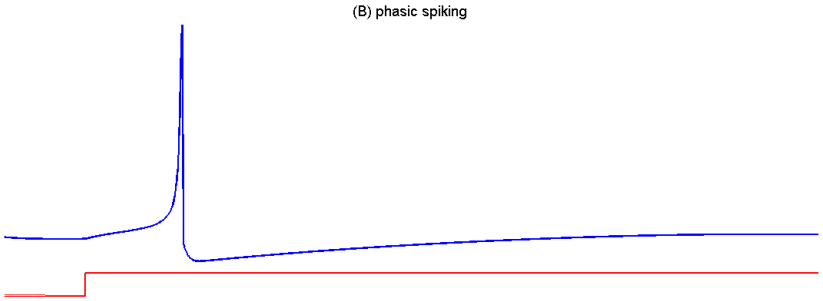


Figure 2.12 Phasic Spiking.

C) **Tonic Bursting:** The neuron fires periodic bursts of spikes when stimulated with input current [11].

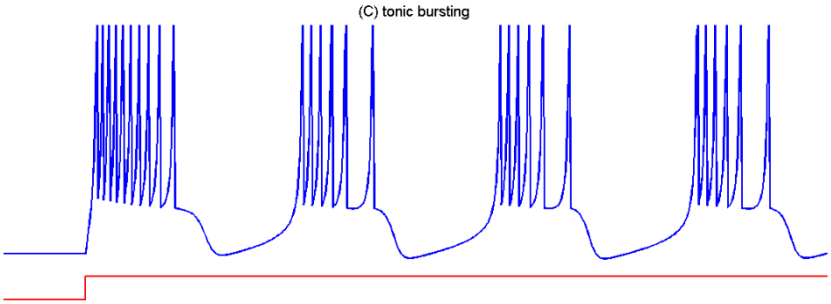


Figure 2.13 Tonic Bursting.

D) **Phasic Bursting:** Similar to phasic spiking but, it produces a burst of spikes at the beginning of the current stimulation only. However, bursts are favoured due to its immunity against neural noise [11].

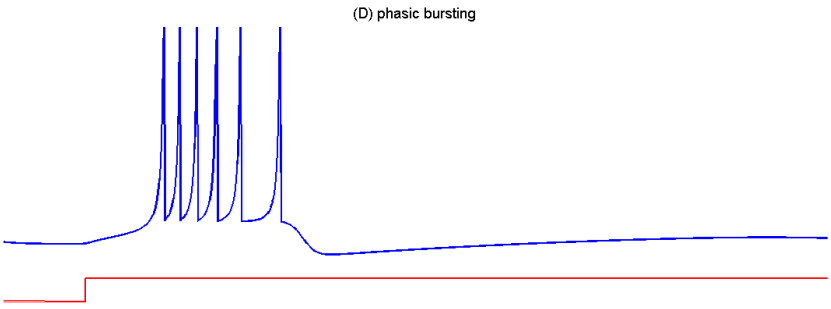


Figure 2.14 Phasic Bursting.

**E) Mixed Mode:** The neuron combines the behaviour of both phasic bursting and tonic spiking. It produces a burst of spikes at the start of the stimulation then, moves to the tonic spiking behaviour by producing a spike train as long as the stimulation is going on [11].

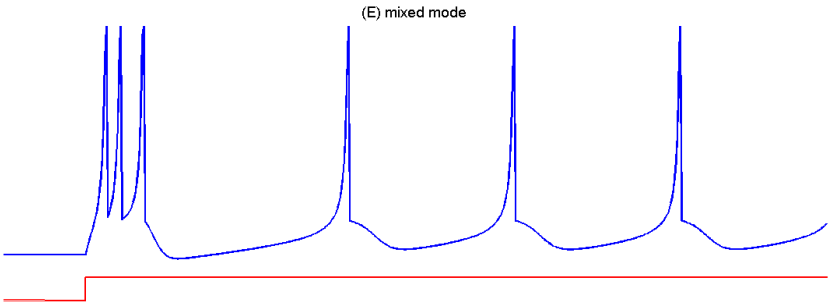


Figure 2.15 Mixed Mode.

**F) Spike Frequency Adaptation:** The neuron fires as tonic spiking but, with a decreasing frequency. It starts with a high frequency at the start of the stimulation then, decreases with time till the end of the current stimulation [11].

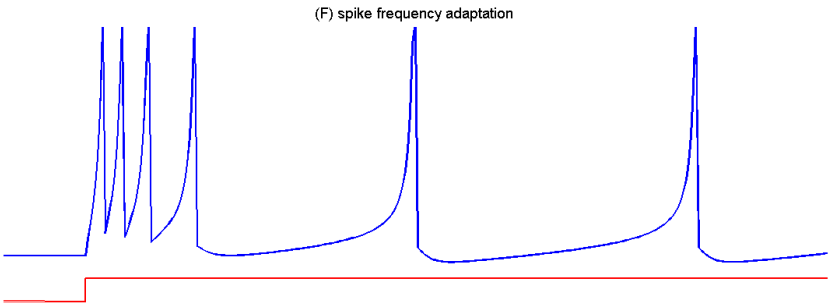


Figure 2.16 Spike Frequency Adaptation.

G) **Class 1 Excitability:** It performs a tonic spiking but, with a variable frequency that depends on the strength of the input stimulation. The spikes frequency ranges from 2 to 200Hz [11].

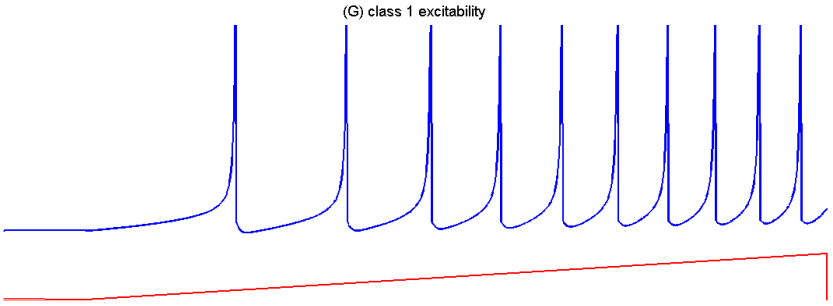


Figure 2.17 Class 1 Excitability.

H) **Class 2 Excitability:** the neuron should work the same as Class 1 Excitability but, it cannot fire a very low frequency spike trains [11].

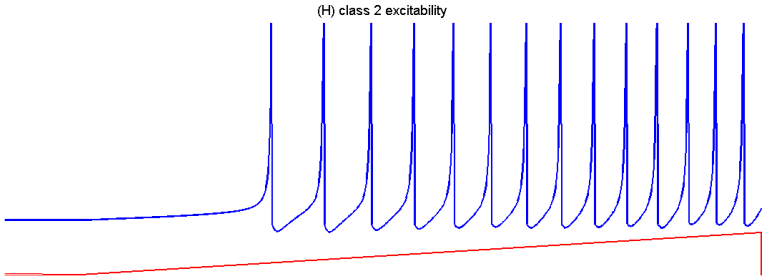


Figure 2.18 Class 2 Excitability.

I) **Spike Latency:** The neuron fires a delayed spike after the stimulation. The delay is proportional to the strength of the stimulation current [11].

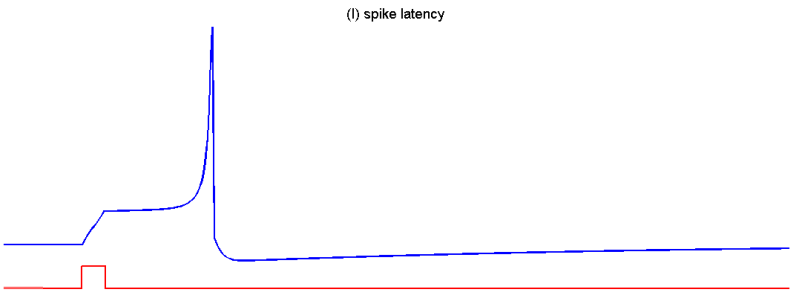


Figure 2.19 Spike Latency.

J) **Subthreshold Oscillations:** Neurons are able to exhibit oscillatory potential. The neurons behave like a bandpass filter where the frequency of the oscillations is an important characteristic [11].

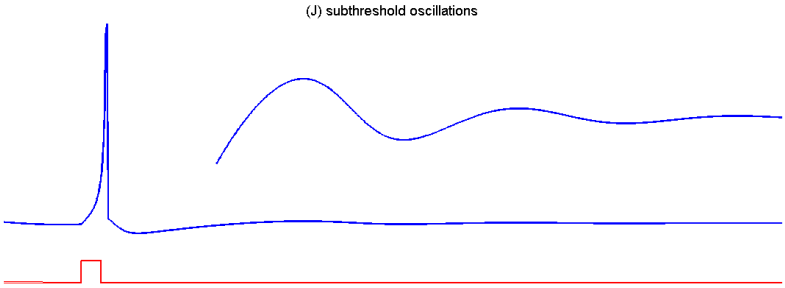


Figure 2.20 Subthreshold Oscillations.

K) **Frequency Preference and Resonance:** Neurons have a selective frequency of stimulation spikes that resonate only with their subthreshold oscillations causing a spike to be fired at their outputs. These neurons can represent a Frequency Modulated (FM) signals [11].

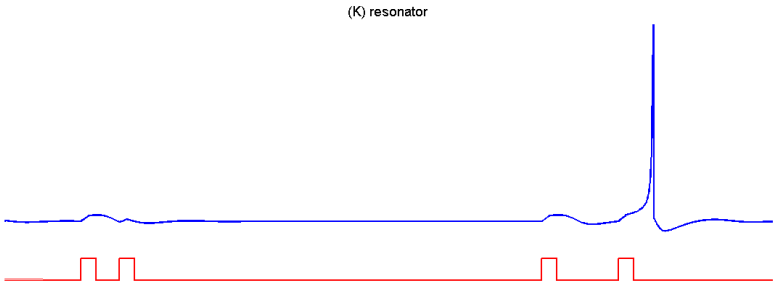


Figure 2.21 Frequency Preference and Resonance.

L) **Integration and Coincidence Detection:** Neurons that do not produce oscillatory potential and act as integrators. It fires more likely with high frequency input. It is useful for detecting coincident spikes [11].

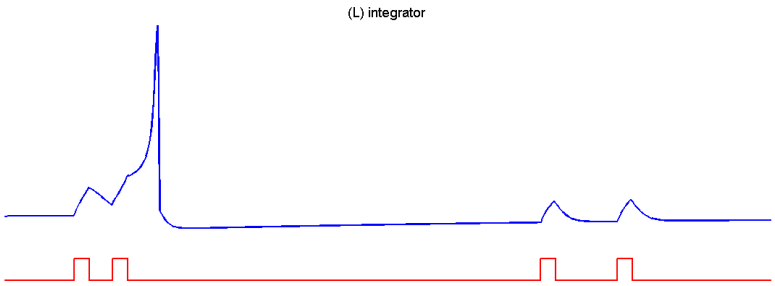


Figure 2.22 Integration and Coincidence Detection.

**M) Rebound Spike:** Neurons fire a rebound spike after receiving an inhibitory input [11]. It can be used as a detection for inhibitory current.

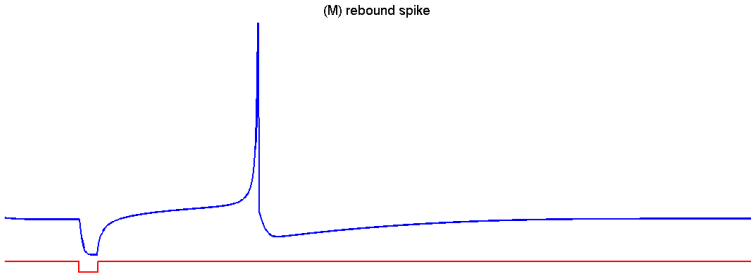


Figure 2.23 Rebound Spike.

**N) Rebound Burst:** Similar to Rebound Spike but, it produces a rebound burst of spikes after receiving an inhibitory input [11].

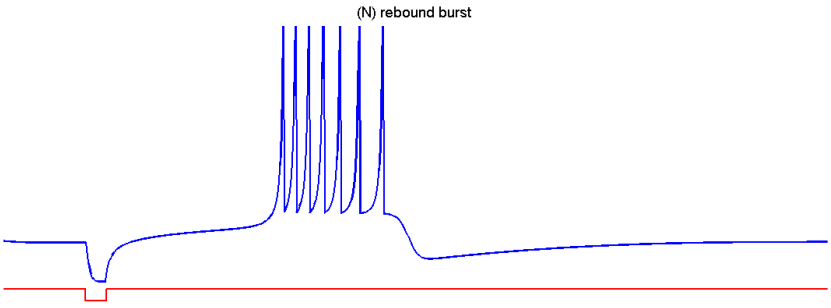


Figure 2.24 Rebound Burst.

**O) Threshold Variability:** A very common misconception is that the neurons have fixed threshold. In this neural behaviour, the neuron is being exposed to a specific excitatory input, no spikes are noticed which means the input does not exceed the threshold. Another inhibitory stimulation is applied to the neuron before applying

the same old excitatory input one more time. The neuron now fires a spike which means the threshold has been reduced after the exposure to inhibitory input. Similarly, an excitatory input might cause the threshold to increase and the neuron becomes less excitable [11].

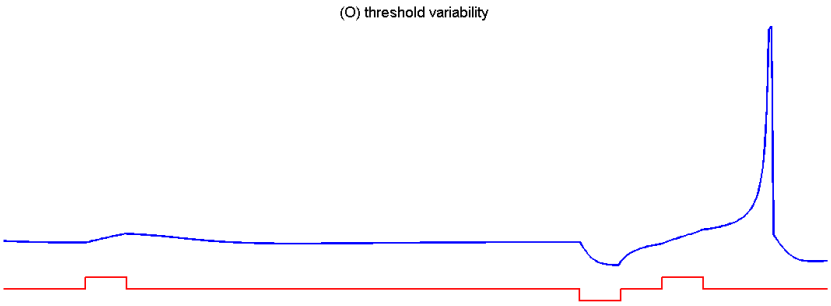


Figure 2.25 Threshold Variability.

**P) Bistability of Resting and Spiking States:** Some neurons can exhibit multiple modes of operation (e.g. resting and tonic spiking). An input pulse (inhibitory or excitatory) can result in a switch between modes which opens the door for a short-term memory behaviour. However, a switch from the tonic spiking to resting mode requires the input stimulation to arrive at an appropriate phase of oscillation which highlights the importance of spike timing in processing [11].

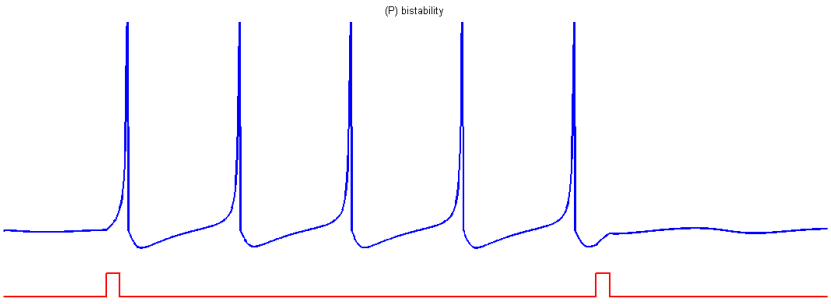


Figure 2.26 Bistability of Resting and Spiking States.

**Q) Depolarizing After-Potentials:** The neuron membrane potential exhibits a prolonged depolarized after-potential (DAP). DAPs appear as a result of a high-threshold input current activated during the spike [11].

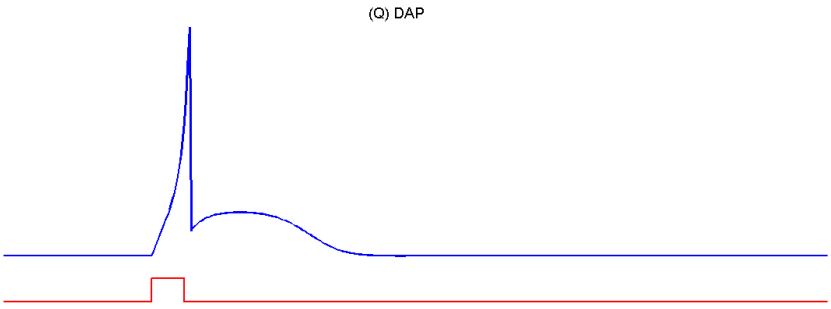


Figure 2.27 Depolarizing After-Potentials.

**R) Accommodation:** Although neurons are sensitive to short stimulation pulses, a strong but slowly increasing ramped input current may not cause a spike fire. Throughout the ramp stimulation, the inward membrane currents have enough time to inactivate while outward membrane currents have enough time to activate, which means that the neuron accommodates and becomes less excitable [11].

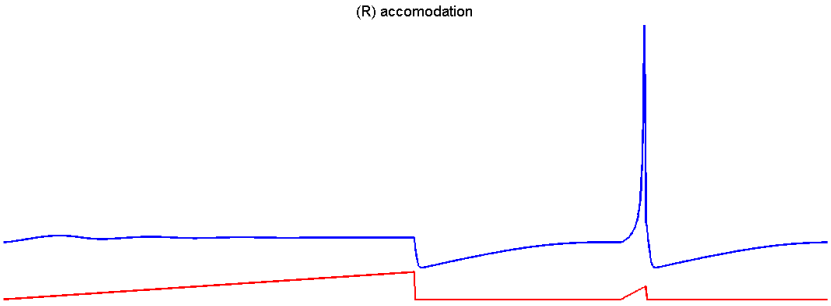


Figure 2.28 Accommodation.

**S) Inhibition-Induced Spiking:** When the injected current activates the h-current and deactivates calcium T-current, tonic spiking is fired even if the input stimulation is inhibitory [11].

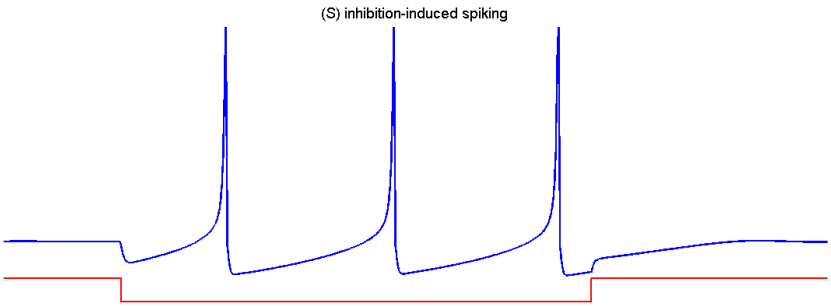


Figure 2.29 Inhibition-Induced Spiking.

T) **Inhibition-Induced Bursting:** Similar to Inhibition-Induced Spiking but, it produces a burst train of spikes instead [11].

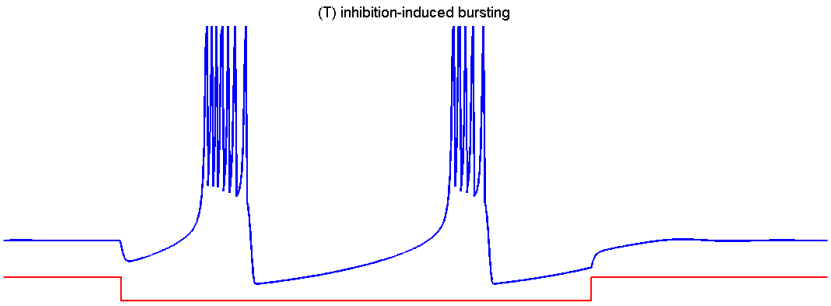


Figure 2.30 Inhibition-Induced Bursting.

## 2.5 Different Spiking Neuron Models

Some of the previously mentioned neuron properties are mutually exclusive. A neuron cannot be a resonator and an integrator at the same time, that is why no model can exhibit all these properties simultaneously. Nevertheless, some models can exhibit different computational properties by tuning some model parameters that control the neuron model behaviour as shown in Figure 2.31. Three common neuron computational models are: Integrate and Fire (IF), Hodgkin and Huxley (HH) and Izhikevich [11].

| Models | biophysically meaningful | tonic spiking | phasic spiking | tonic bursting | phasic bursting | mixed mode | spike frequency adaptation | class 1 excitable | class 2 excitable | spike latency | subthreshold oscillations | resonator | integrator | rebound spike | rebound burst | threshold variability | bistability | DAP | accommodation | inhibition-indiced spiking | inhibition-induced bursting | chaos | # of FLOPS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| integrate-and-fire | - | + | - | - | - | - | - | + | - | - | - | - | + | - | - | - | - | - | - | - | - | - | 5 |
| integrate-and-fire with adapt. | - | + | - | - | - | - | + | + | - | - | - | - | + | - | - | - | - | + | - | - | - | - | 10 |
| integrate-and-fire-or-burst | - | + | + | | + | - | + | + | - | - | - | - | + | + | + | - | + | + | - | - | - | | 13 |
| resonate-and-fire | - | + | + | - | - | - | - | + | + | - | + | + | + | + | - | - | + | + | + | - | - | + | 10 |
| quadratic integrate-and-fire | - | + | - | - | - | - | - | + | - | + | - | - | + | - | - | + | + | - | - | - | - | - | 7 |
| Izhikevich (2003) | - | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | 13 |
| FitzHugh-Nagumo | - | + | + | - | | - | - | + | - | + | + | + | - | + | - | + | + | - | + | + | - | - | 72 |
| Hindmarsh-Rose | - | + | + | + | | | + | + | + | + | + | + | + | + | + | + | + | + | + | + | | + | 120 |
| Morris-Lecar | + | + | + | - | | - | - | + | + | + | + | + | + | + | | + | + | - | + | + | - | - | 600 |
| Wilson | - | + | + | + | | | + | + | + | + | + | + | + | + | + | + | | + | + | | | | 180 |
| Hodgkin-Huxley | + | + | + | + | | | + | + | + | + | + | + | + | + | + | + | + | + | + | + | | + | 1200 |

Figure 2.31 Comparison between different spiking neuron models in their ability to exhibit certain spiking behaviors and their computational complexity.

### 2.5.1 Integrate and Fire (IF) Neuron Model:

It is considered one of the simplest models that describe the neuron behavior. The neuron is defined in (3) as:

$$I(t) = C_m \frac{dV_m(t)}{dt} \tag{3}$$

Which is basically the derivative of the capacitance law with respect to time. When an input current is applied to the neuron, the membrane voltage increases until it reaches a certain value, threshold voltage, then a spike is be fired and the membrane voltage is reset to the resetting potential value [11].

The model at this form is not accurate enough as the firing frequency increases linearly with the input current without any upper bound for the firing frequency. To enhance the model and accommodate this problem, the concept of *refractory period* ($t_{ref}$) is introduced. The refractory period is the amount of time after the spike during which firing new spikes is not allowed. The firing frequency can be formulated as in (4) [11]:

$$f(I) = \frac{I}{C_m V_{th} + t_{ref} I} \tag{4}$$

### 2.5.2 Hodgkin–Huxley (HH) Neuron Model:

Hodgkin–Huxley (HH) is one of the most complicated, but accurate models in computational neuroscience. It consists of four differential equations that describe the membrane potential, activation of sodium and potassium currents and inactivation of sodium current. The model has tens of parameters that are tuned to achieve the neural properties discussed in previous sections. Another good advantage of the HH model is that all its parameters are biologically meaningful and measurable [11].

### 2.5.3 Izhikevich Neuron Model:

As illustrated in Figure 2.32, this model can produce the 20 essential biological neuron behaviors discussed earlier in this chapter. The Izhikevich model consists of two differential equations describing the biophysical neuron behavior as in (5), (6) and (7) [10].

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{5}$$

$$u' = a(bv - u) \tag{6}$$

$$if\ v \geq 30\ mV, then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{7}$$

(5) and (6) describe how the membrane potential $v$ and the membrane recovery variable $u$ change. (7) is the after-spike resetting equation. The membrane potential $v$ and the recovery variable $u$ depend on the following four parameters $(a, b, c\ and\ d)$ and the current *I* as explained in [10].

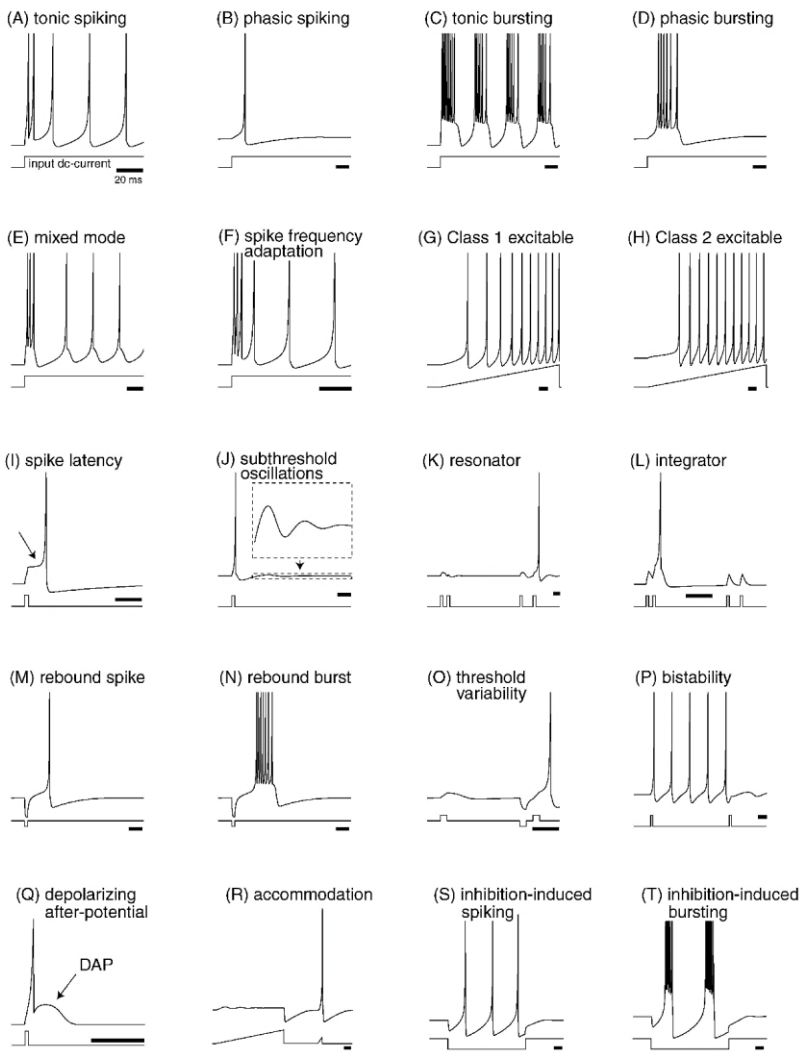


Figure 2.32 20 biological neuron behaviors produced by Izhikevich neuron model.

As illustrated in Figure 2.33, '*a*' represents the recovery time of the neuron controlling the neuron recovery speed after firing. '*b*' controls the sensitivity of the membrane recovery variable *u*. Its value must be determined carefully to avoid fluctuations or subthreshold firing of the membrane potential $v$ . '*c*' is the after-spike reset potential of *v*. '*d*' determines the reset value of the variable $u$ when the after-spike resetting condition is met. '*I*' is the synaptic current where it can be excitatory or inhibitory one [3]. Figure 2.33 shows how tuning the four basic parameters *a,b,c* and *d* can help achieve different neuron behaviors discussed earlier [10].

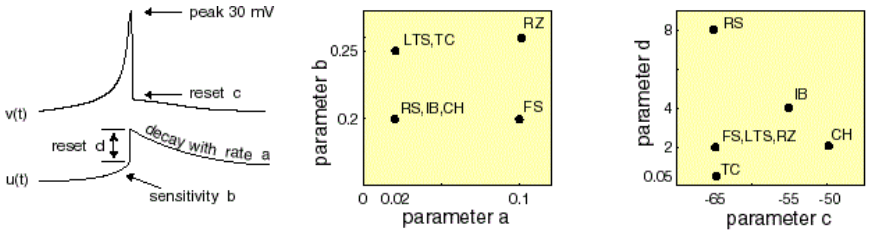


Figure 2.33 Izhikevich model paramters values to generate different neural behaviors like the biological neuron.

In this work, Euler method is used to solve numerically the differential equations with time step, $\Delta t$ as Izhikevich suggested in [11]. (8) and (9) show Izhikevich model in Euler formula as follows:

$$v_{t+\Delta t} = v_t + \Delta t(0.04{v_t}^2 + 5v_t + 140 - u_t + I) \tag{8}$$

$$u_{t+\Delta t} = u_t + \Delta t * a(bv_t - u_t) \tag{9}$$

### 2.5.4 Comparison between the neuron models:

As discussed before, different neuron models offer a variety of computational accuracy as well as a variety of model complexity. Figure 2.34 shows the trade-off between the complexity and accuracy in different neuron models. It is noticeable that Izhikevich model offers a high accuracy, almost the same as HH model, while costing a low complexity compared to other neuron models. That is why Izhikevich model is preferred in neural computations over the other models [11].

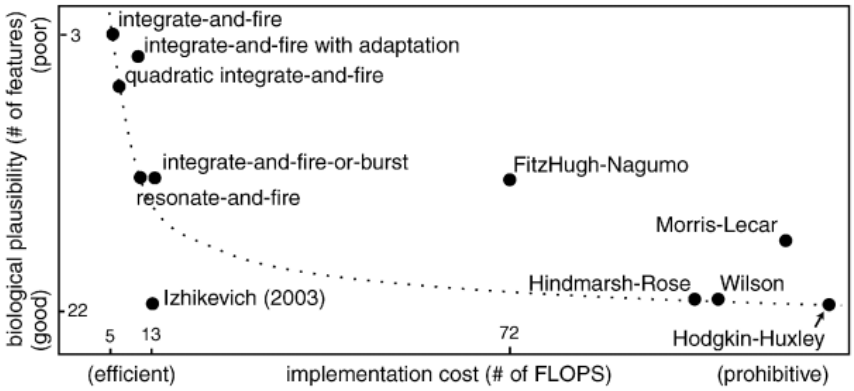


Figure 2.34 Accuracy versus Complexity of different neuron models.

## 2.6 Types of Learning Algorithms

Learning algorithm is the stage of teaching the neural network to perform a certain task. The learning algorithm is mainly responsible for updating the weights of layers connections. The weights increase in value if a certain input strongly affects the output and decrease otherwise. There are three learning algorithms that are commonly used in neural networks [9].

### 2.6.1 The Supervised Learning

This type of learning is characterized by presence of the desired output as shown in Figure 2.35. Using this algorithm, the network is able to configure itself by comparing the actual output with the desired output. Then, the network calculates the error with which the network weights are updated [9].

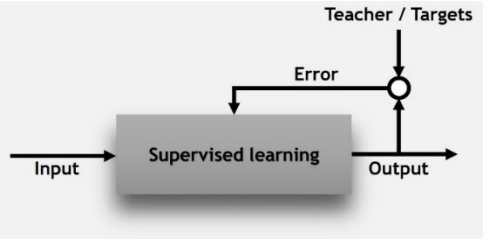


Figure 2.35 The Supervised Learning.

### 2.6.2 The Unsupervised Learning

Unlike the supervised learning, this learning algorithm does not have a desired output as shown in Figure 2.36. Unsupervised learning is characterized by the presence of only the inputs where the network is supposed to learn by itself. The network learns by searching for similarities and patterns in the applied input that's why it is also called self-organizing learning [9].

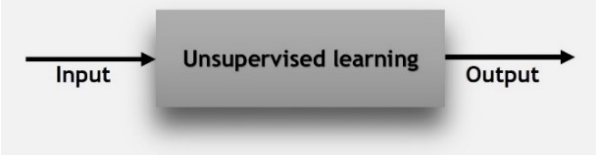


Figure 2.36 The Unsupervised Learning.

### 2.6.3 The Reinforcement Learning

The idea behind this learning algorithm comes from the punishment and reward learning method from the psychology field. The network learns by trial and error as shown in Figure 2.37. The right behavior of the network gets a positive reward while the undesired wrong behavior gets a punishment. Sometimes, Reinforcement learning is considered as subcategory of supervised learning. [9].

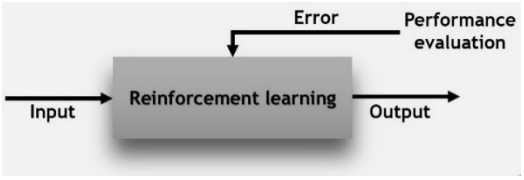


Figure 2.37 The Reinforcement Learning.

In Figure 2.38, it is shown that the human brain does not only use one type of learning, instead, the brain uses the three types of learning. Each learning algorithm occurs in a different part inside the brain [12].

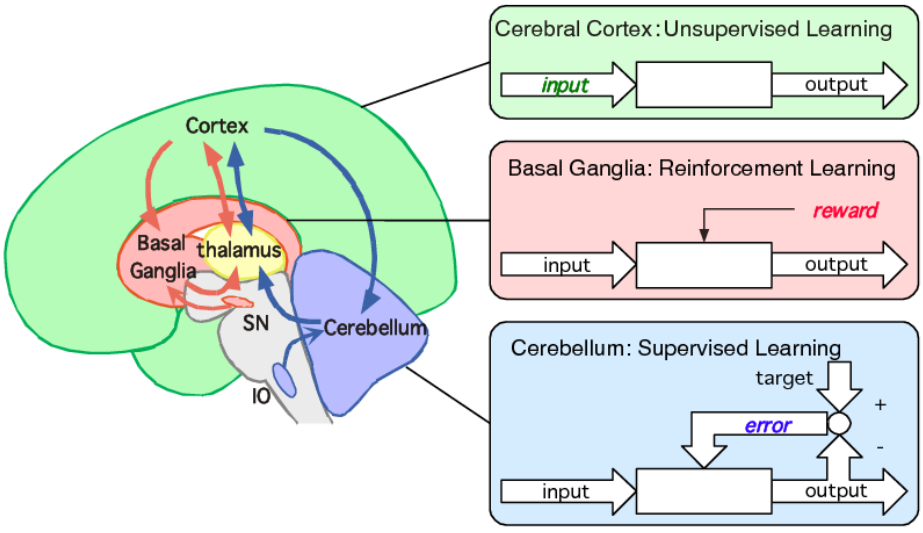


Figure 2.38 Specialization by Learning Algorithms by Doya in 1999.

## 2.7 Online Learning Versus Offline Learning

From the implementation perspective, there are two main techniques to do learning which are on-chip (Online) learning and off-chip (offline) learning. The offline learning is commonly used than the online learning. The idea of the off-chip training is based on simulating and training the network using a high-level language till it reaches the desired accuracy. Then, the resulted optimum weights are stored into a memory to be used in hardware implementation. This technique has several advantages that it is very fast compared to the online learning and provides accurate weights without truncation errors. On the other hand, offline learning requires a large memory to store the weights where such a memory takes a large area and increases the power consumption. In addition, the weights will be fixed. If the network is implemented on a FPGA, this is not a problem because it can be reprogrammed easily. However, if the network is implemented on ASIC platform, there is no way to modify the weights after implementation. This introduces significant limitation to the network design [13].

The online learning performs the network training on hardware. It has a great advantage that weights are updated instantaneously. Also, if the network is implemented on a non-reprogrammable platform such as ASIC, there is no worry about updating the weights because learning occurs online. On the other hand, this technique has several disadvantages because it will increase the time delay that's why online learning is significantly slower than the offline learning. Another problem is that it requires additional area and power due to the additional implemented logic and circuit of the learning [13].

## 2.8 Different Network structures

Neural networks can be classified according to their structure. The network structure differs in the connections between the layers. There are two main topologies in the neural networks:

### 2.8.1 Feedforward Structure

The feedforward topology network is a unidirection topology in which each neuron is connected to all the neurons in the following layer as shown in Figure 2.39. Within the same layer, neurons are not allowed to connect to each other. In addition, neurons are not allowed to connect to the preceding neurons. Also, a neuron cannot make cycles or loops. [6].

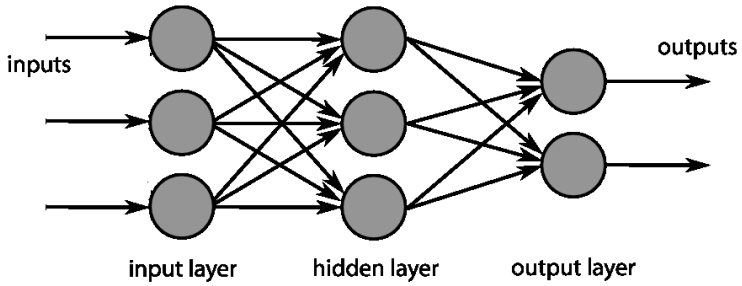


Figure 2.39 Feedforward Neural Network Structure.

### 2.8.2 Recurrent Structure

In the recurrent topology network, the interconnections have a higher level of freedom compared with the feedforward one. There is no single path direction that a neuron can connect either to a following or a preceding layer as shown in Figure 2.40. Also, cycles and loops are allowed. This could be thought of as a sort of feedback mechanism, but this topology will increase the complexity of the network [6].

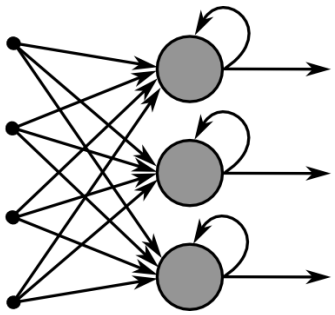

Figure 2.40 Recurrent Neural Network Structure.

# CHAPTER 3

## Problem definition and our contribution

### 3.1 Model accuracy versus power consumption

As discussed before, there are several types of Neural Networks such as ANN, CNN and SNN. Recently, SNN is expected to replace ANN and CNN as it can simulate the human brain behavior better than the previous generations of neural network. SNN can achieve high accuracy as its neuron models mimic the biological neuron behaviors perfectly. Regrettably, the spiking neuron models may consume higher power than the artificial neurons in ANN.

The trade-off between the accuracy and the power consumption leads us to the main neoteric objective in the spiking neural networks, whereas a variety of spiking neuron models exists that can generate a poorly approximated to exactly simulated neuron behaviors according to the required accuracy. The main features required while choosing an appropriate spiking neuron model are simplicity, power consumption and number of supported behaviors.

The model that is suitable to achieve most of these features is Izhikevich neuron model. The reason is that Izhikevich neuron model has a moderate simplicity, is represented by two differential equations only and produces various neural behaviors.

### 3.2 Problem in Izhikevich model

The selected model to implement our spiking neural network is the Izhikevich neuron model which offers suitable performance and accuracy compared to the other neuron models. On the other hand, all spiking neuron models, including Izhikevich model itself, consume higher power than the artificial neuron in ANN due to the used differential equations and multipliers. Despite of Izhikevich model simplicity compared to other spiking models, it still consumes power due to the squaring term $v^2$ and several constant

multipliers. These high-power components in the neuron model won't be acceptable according to the market needs as neural networks, in general, is required in low power systems like IOT and biological sensors. In addition, all neural networks that are used nowadays in CPUs and cell phones, are designed to be ultra-low power to save the device's battery life time.

As a solution, the constant multipliers are implemented using fixed shifters and adders. Due to bits limitation and truncation, an error affects the neuron accuracy compared to the original Izhikevich model. The original Izhikevich model refers to the exact Izhikevich model without approximating the parameters values. In addition, we have to introduce some approximation techniques to decrease the power consumption of the remaining power-hungry term, the squaring term.

### 3.3 Our contribution

The only approximation method applied to the Izhikevich model in previous research work is the Piece-Wise Linear approximation (PWL) [1]. This technique converts the squaring term into a sum of straight lines. This method increases the accuracy of the approximation by increasing the number of straight lines. There are three different orders of piece wise linear system discussed in [1]. The $2^{nd}$ order PWL uses two straight lines, the $3^{rd}$ order PWL uses three straight lines while the $4^{th}$ order PWL uses four straight lines. In this project, only the $4^{th}$ order PWL is used during comparison and evaluation. We will refer to the $4^{th}$ order PWL in the upcoming sections as PWL only for simplicity. The reader should keep in mind that we are talking about the $4^{th}$ order PWL approximation when writing PWL abbreviation.

The second technique is COordinate Rotation Digital Computer (CORDIC) which splits the number to be squared into a fraction part and an integral part. After that, each part is divided into a summation of powers of two to be implemented using shifters and adders instead of multipliers.

The third technique is called Iterative Logarithmic Method (ILM) which aims to convert the number to the largest summation of powers of two. This technique subtracts a power of two term from the number then, considers the remainder as an error term thereafter, it iterates on this subtraction operation until reaching a desired minimum error value.

The last method is called Integral Sum which modifies the squaring to a simple iterative addition operation. This conversion is based on the fundamental principle of multiplication that any multiplication operation can be converted to an addition operation. This methodology has also multiple approaches to achieve acceptable accuracy. From a hardware perspective, all these techniques try to replace the huge multiplier circuit with a set of shifters and adders. These approximation methods are discussed in details in the next chapter.

## 3.4 The flow of our work

The following workflow diagrams show our progress during this project. Figure 3.1 shows the prerequisite skills needed before going through this neuromorphic computing project.

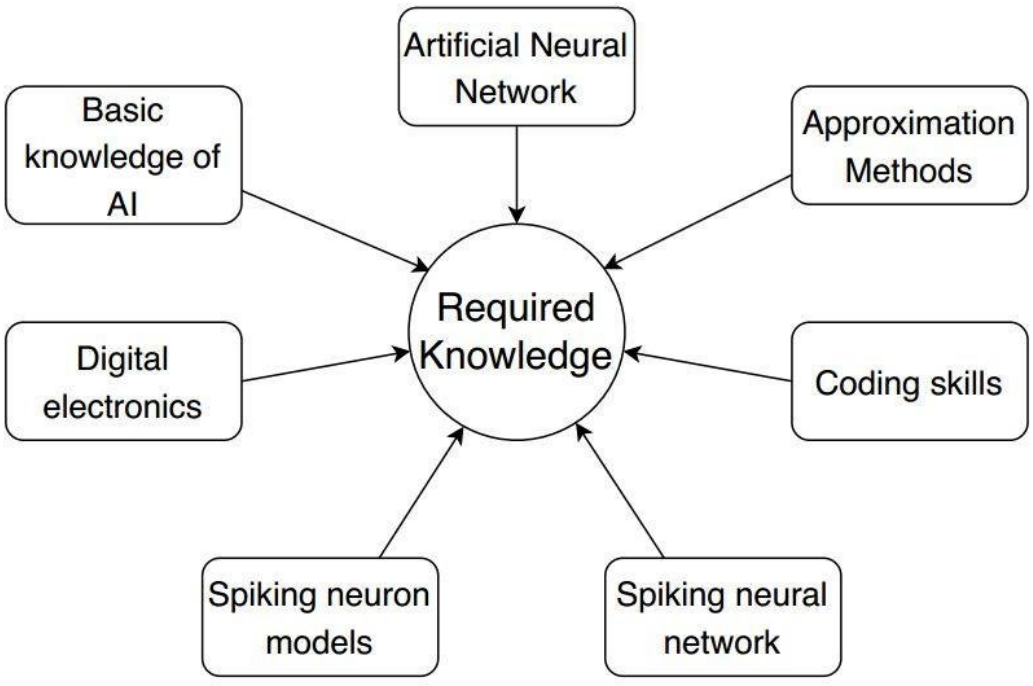


Figure 3.1 The basic required knowledge and skills before starting our neuromorphic computing project.

Figure 3.2 illustrates the first step in this project which is choosing the suitable type of network to be implemented. In this work, Spiking Neural Network (SNN) is chosen.

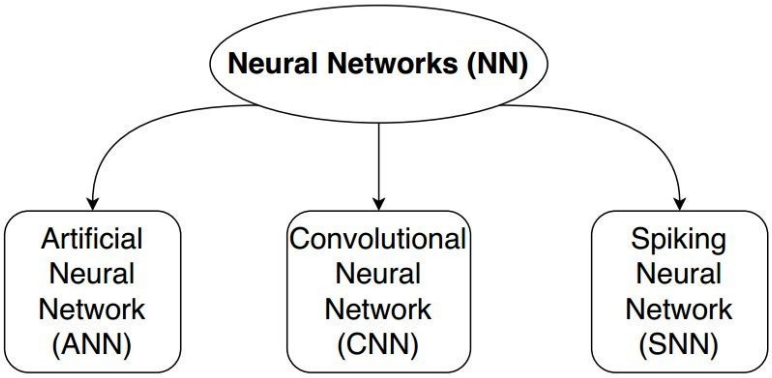


Figure 3.2 Determining the type of network to be implemented.

In Figure 3.3, the spiking neuron model is determined according to our accuracy and power guidelines. In this project, the Izhikevich neuron model is chosen.

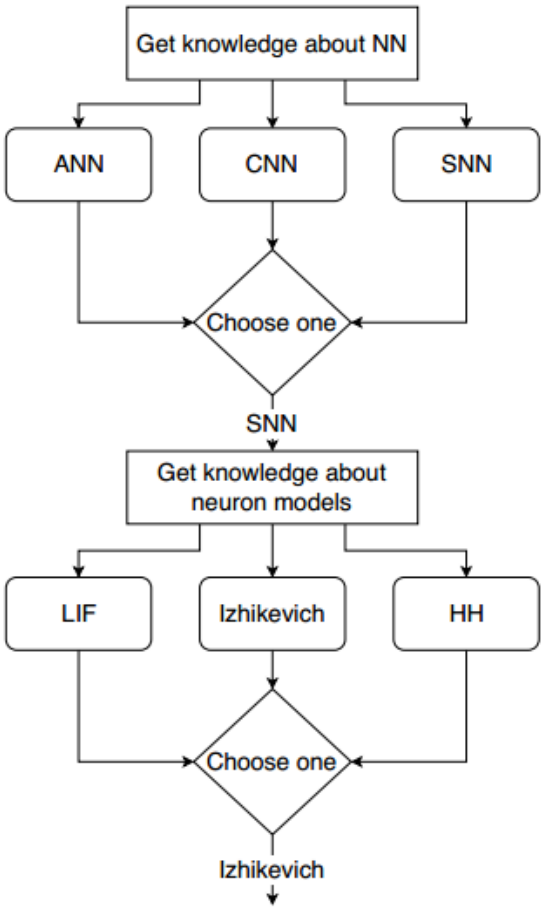


Figure 3.3 After choosing Spiking Neural Network, we have to choose the spiking neuron model that is suitable for our performance matrix.

Figure 3.4 highlights the process of proposing new approximation techniques and taking the final decision to implement our spiking neuron. A CORDIC-based hardware implementation of the Izhikevich neuron model is introduced

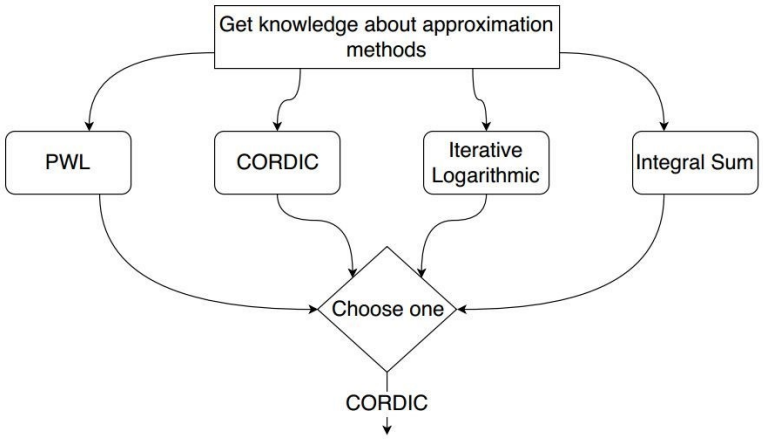


Figure 3.4 After choosing Izhikevich neuron model, propose approximation methods and choose the most accurate one.

To determine the applicability of each approximation method, a high-level language, MATLAB, is required. Using MATLAB, each method is evaluated to determine the most accurate approximation algorithm to be used as shown in Figure 3.5. Our neuron accuracy is compared with the original and the PWL-based neurons.

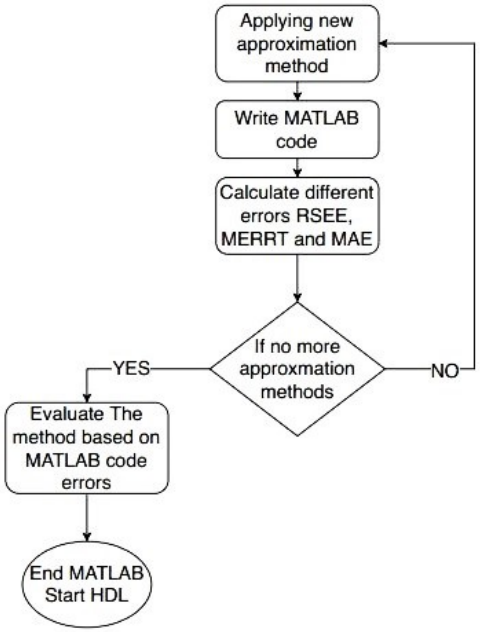


Figure 3.5 High level language optimization and comparison among different approximation methods.

As illustrated in Figure 3.6, the CORDIC-based, PWL-based and original neuron models are written as a VERILOG code and we calculate again the accuracy for each method based on the HDL output data. Then, the spiking neuron is implemented on ASIC/FPGA platforms. A performance analysis is held among the three models to show the power and area reduction due to the CORDIC and the PWL compared with the original neuron in ASIC and FPGA platforms.

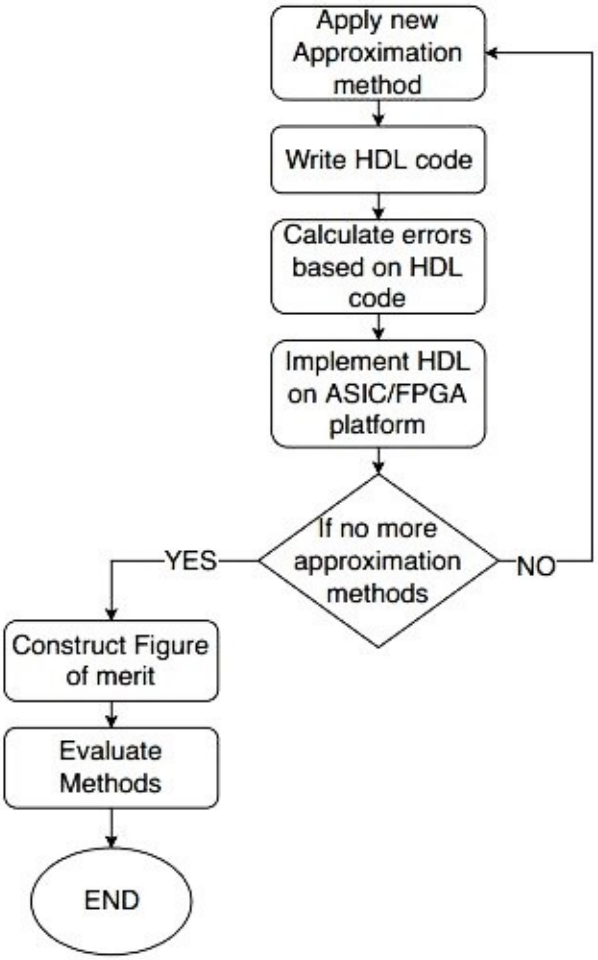


Figure 3.6 Implementation of the most suitable approximation methods to test the hardware implemented in ASIC/FPGA platforms.

After that, the spiking neural network is constructed using the CORDIC-based, PWL-based and original neuron models. Then, the accuracy of each network is calculated using MATLAB to determine the most preferable approximation technique. Finally, the power and area values are estimated to show up the reduction occurred while implementing the network with the approximation methods.

# CHAPTER 4

## Single Izhikevich Neuron

### 4.1 The proposed approximation methods

#### 4.1.1 Piece-Wise Linear approximation (Previous work)

The Piece-Wise Linear (PWL) converts the squaring term into a sum of straight lines. This method is applied to the quadratic term in the Izhikevich model before in [1]. Depending on the number of straight lines, PWL can be second, third and fourth order. Increasing the number of straight lines results in better accuracy and resemblance to the square term. Yet, this leads to more complexity in terms of hardware implementation [1].

**A) 2nd order Piece-Wise Linear approximation**

The first meaningful PWL approximation to the quadratic equation in Izhikevich model is to use two straight lines as shown in Figure 4.1. This approximation is formulated in (10) as follows [1]:

$$f'(V) = k_1|V + 62.5| - k_2 - u + I \quad (10)$$

Where $k_1$ and $k_2$ are the parameters that represent the degrees of freedom produced by the PWL approximation. Tuning these parameters is essential to achieve the closest behavior to the original model [1].

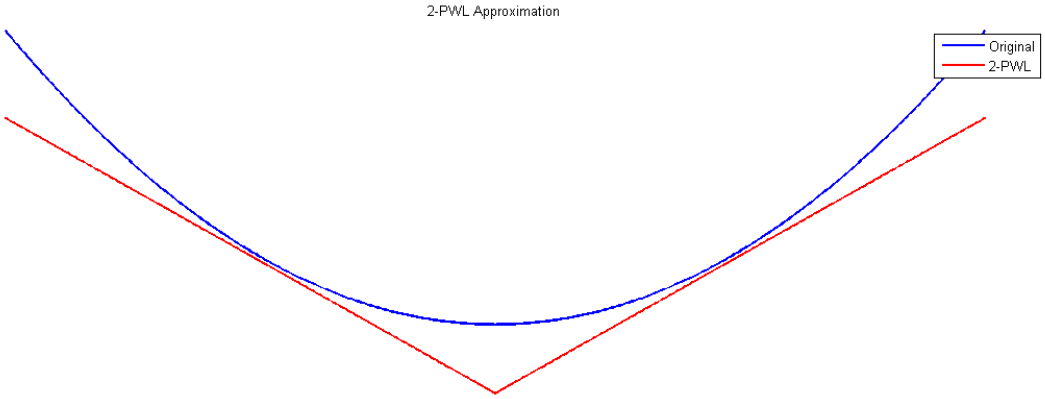


Figure 4.1 2nd order PWL approximation.

### B) 3rd order Piece-Wise Linear approximation

The same quadratic term can be approximated by PWL model with a better accuracy by increasing the number of straight lines from two lines to three lines. This third order approximation is shown in Figure 4.2 and can be formulated in (11) as follows [1]:

$$f'(V) = k_1(|V + 62.5 + k_2| + |V + 62.5 - k_2|) - k_1k_2k_3 - u + I \qquad (11)$$

It is noticeable that the number of fitting parameters ($k_1k_2k_3$) has increased, meaning that this approximation has more degrees of freedom than the previous one [1].

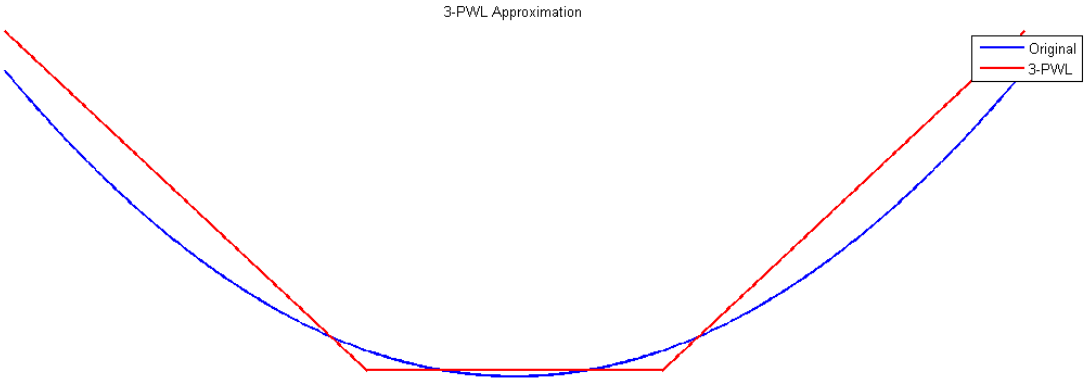


Figure 4.2 3rd order PWL approximation.

### C) 4th order Piece-Wise Linear approximation

Following the same pattern, the accuracy can be increased by increasing the order of PWL approximation. It is proposed in [1] that 4th order PWL approximation is a good trade-off between accuracy and complexity. The 4th order PWL approximation is illustrated in Figure 4.3. The approximation can be formulated in (12) as follows:

$$f'(V) = k_2(|V + 62.5 + k_3| + |V + 62.5 - k_3|) + k_1|V + 62.5| - 4k_2k_3 - u + I \quad (12)$$

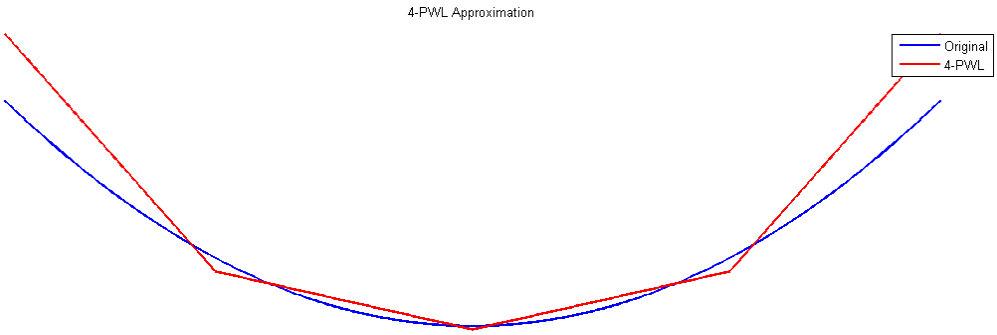


Figure 4.3 4th order PWL approximation.

### 4.1.2 COordinate Rotation Digital Computer algorithm

The COordinate Rotation Digital Computer (CORDIC) algorithm is used in approximating the exponential term in the Adaptive Exponential Integrate and Fire (AdEx) neuron model [14]. The algorithm is used in approximating different functions (e.g., multiplication, exponential function and hyperbolic function) [14]. To understand how the CORDIC algorithm works with multiplication, assume that the two multiplicands $A_1$ and $A_2$ are divided into two terms: integer term $I$ and fraction term $F$ in (13) as follows:

$$A_1 \times A_2 = (I_1 + F_1) \times (I_2 + F_2) \quad (13)$$

To convert the multiplication to shifters, the integer and fraction terms are written as a sum of powers of two in (14) and (15).

$$I = I[1] + I[2] \times 2 + I[3] \times 4 \ldots = \sum_{i=1}^{m} I[i] \times 2^{i-1} \tag{14}$$

$$F = \frac{F[1]}{2} + \frac{F[2]}{4} + \frac{F[3]}{8} \ldots = \sum_{i=1}^{n} F[i] \times 2^{-i} \tag{15}$$

Where $I[i]$ and $F[i]$ are either 0 or 1. $m$ and $n$ define the limits of the integer and the fraction terms, respectively. Since the Izhikevich model contains a square term ($v \times v$), only one $v$ is written as the form in (14) and (15) and the other $v$ is just a shifted version. Figure 4.4 shows the pseudocode of the CORDIC-based squaring.

```
I = Integer(V); // Determine the integer part
F = Fraction(V); // Determine the fraction part
m = number of bits for integer; // Endpoint for integer
n = number of bits for fraction; // Depends on accuracy
poweroftwo = 2^(m-1); // initial value
inverse_poweroftwo = 0.5; // initial value
square = 0;
// Calculate the integer part and multiply by the input
for i from 1 to m do{
    if (I ≥ poweroftwo){
        I = I - poweroftwo;
        square = square + V * poweroftwo;
    }
    poweroftwo = poweroftwo / 2;
}
// Calculate the fraction part and multiply by the input
for j from 1 to n do{
    if (F ≥ inverse_poweroftwo){
        F = F - inverse_poweroftwo;
        square = square + V * inverse_poweroftwo;
    }
    inverse_poweroftwo = inverse_poweroftwo / 2;
}
```

Figure 4.4 The pseudocode of the CORDIC-based squaring.

In this work, the CORDIC algorithm is applied to the Izhikevich model by substituting in (8) with (14) and (15). Accordingly, (8) is approximated in (16) as follows:

$$v_{t+\Delta t} = v_t + \Delta t \left( 0.04 v_t \left( \sum_{i=1}^{m} I[i] \times 2^{i-1} + \sum_{i=1}^{n} F[i] \times 2^{-i} \right) + 5 v_t + 140 - u_t + I \right) \tag{16}$$

Where $m$ and $n$ are predetermined before the hardware implementation.

### 4.1.3 Iterative Logarithmic method

The iterative logarithmic method (ILM) is one of proposed techniques used to approximate the $v^2$ term in the Izhikevich neuron model and simplify the multiplication operation on the hardware level. The method is an iterative algorithm where the user determines the required output accuracy. This method could get the exact value depending on the number of iterations. The multiplicands are represented in the logarithmic form [15]. Then, all the operations are converted into shifters and adders. To understand how this method works, let us assume we multiply $N_1 * N_2$ where each number is represented in (17) as follows:

$$N_1 = 2^{k_1} + N_1^{(1)} \; and \; N_2 = 2^{k_2} + N_2^{(1)} \tag{17}$$

$N_1^{(1)}$ and $N_2^{(1)}$ are the reminders of $N_1 - 2^{k_1}$ and $N_2 - 2^{k_2}$, respectively.

Hence, the product of $N_1 \; and \; N_2$ is formulated in (18) and (19) as follows:

$$P_{true} = N_1 * N_2 = \left(2^{k_1} + N_1^{(1)}\right) * \left(2^{k_2} + N_2^{(1)}\right) \tag{18}$$

$$P_{true} = 2^{k_1+k_2} + 2^{k_1} * N_2^{(1)} + 2^{k_2} * N_1^{(1)} + N_1^{(1)} * N_2^{(1)} \tag{19}$$

As seen above, $P_{true}$ is a set of several shift operations that are easily implemented in hardware except for the last term $N_1^{(1)} * N_2^{(1)}$ . Thus, $P_{true}$ is approximated by ignoring the last term in (20) as follows:

$$P_{approx}^{(0)} = 2^{k_1+k_2} + 2^{k_1} * N_1^{(2)} + 2^{k_2} * N_1^{(1)} \tag{20}$$

Where $P_{true} = P_{approx}^{(0)} + E^{(0)}$ and $E^{(0)} = N_1^{(1)} * N_1^{(2)}$.

To improve the accuracy and decrease the error, the previous steps are applied again on the error term, $E^{(0)}$ as in (21) where $N_1^{(1)} = 2^{l_1} + N_1^{(2)}$ and $N_2^{(1)} = 2^{l_2} + N_2^{(2)}$.

$$E^{(0)} = 2^{l_1+l_2} + 2^{l_1} * N_2^{(2)} + 2^{l_2} * N_1^{(2)} + N_1^{(2)} * N_2^{(2)} \quad (21)$$

As we did previously, the error term, $E^{(0)}$ is approximated by ignoring the last term, $N_1^{(2)} * N_2^{(2)}$ as in (22).

$$C^1 = 2^{l_1+l_2} + 2^{l_1} * N_2^{(2)} + 2^{l_2} * N_1^{(2)} \quad (22)$$

Where $E^{(0)} = C^1 + E^{(1)}$.

Similarly, the method is applied once more time until the error becomes small enough to be acceptable. The final result can be formulated in (23) as follows:

$$P_{approx}^{(i)} = P_{approx}^{(0)} + \sum_{j=1}^{i} C^{(j)} \quad (23)$$

Figure 4.5 below shows a pseudocode that describes how to approximate the $v^2$ using the iterative logarithmic method.

```
V = Input voltage; // to be squared
square = 0; // the initial square value of input V
P_log_approx = the result of the approximation;
// the logarithm representation of the multiplicands
m = number of bits for integer; //endpoint for integer
n = number of iterations; // to determine the accuracy
// try to write the input in this form: V = 2^k + N
for j from n to 1 do{
    for i from m to 1 do{
        if (V > 2^i){
            k = i;
            N = V - 2^k;
            break;
        }
    }
    square = square + 2^(2*k) + 2*N*2^k;
}
```

Figure 4.5 The pseudocode of the iterative logarithmic-based squaring.

To clarify the method, let us take an example where $N_1 = 7$ and $N_2 = 9$.

$$N_1 = 2^2 + 3 \text{ and } N_2 = 2^3 + 1$$

Their product equals to $P_{true} = N_1 * N_2 = (2^2 + 3) * (2^3 + 1)$

$$P_{true} = 2^5 + 2^2 * 1 + 2^3 * 3 + 3 * 1$$

The approximated product equals to $P_{approx}^{(0)} = 2^5 + 2^2 * 1 + 2^3 * 3 = 60$

Where $P_{true} = P_{approx}^{(0)} + E^{(0)}$.

The error from the first iteration is $E^{(0)} = N_1^{(1)} * N_1^{(2)} = 3 * 1$. Hence, apply the method again on the error term where $N_1^{(1)} = 3 = 2^1 + 1$ and $N_2^{(1)} = 1 = 2^0$.

$$C^1 = 2^1 + 2^0 * 1$$

Where $E^{(1)} = 0$.

$$E^{(0)} = C^1 + E^{(1)} = 2^1 + 2^0 * 1 = 3$$

So, the final result equals to

$$P_{approx}^{(i)} = P_{approx}^{(0)} + \sum_{j=1}^{i} C^{(j)} = 60 + 3 = 63$$

As seen above, the final results agree with the correct answer of multiplying 7 by 9.

### 4.1.4 Integral Sum

The integral sum algorithm is considered a new methodology to calculate the squaring. The idea behind this method is based on the multiplication principle. Let us take this simple example to demonstrate the idea behind the algorithm. Assume $V = 10$ is to be squared.

$$10^2 = 10 * 10 = 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 = 100$$

Instead of multiplying 10 by 10, you can add it ten times. As a result, the multiplication operation is converted into addition operation as shown above. This conversion is formulated in (24) as follows:

$$V^2 = \sum_{i=0}^{|V|} V \tag{24}$$

From the hardware perspective, the critical path depends on the value of $V$. The reason is that if you square a small number, you will need a small number of iterations. On the other hand, if you square a large number, you will need a large number of iterations. To solve this problem, we can do another mathematical trick which is to add the number 10 five times.

$$10^2 = 10 * 10 = 10 + 10 + 10 + 10 + 10 = 50$$

As a result, the solution lost half its value, but the number of iterations decreased to half improving the performance in hardware. To accommodate the loss occurred, we can multiply the solution by 2 as follows:

$$10^2 = 10 * 10 = 2(10 + 10 + 10 + 10 + 10) = 100$$

The multiplication by 2 should not be considered as a complex multiplication operation. The reason is that multiplication by 2 is realized in hardware as a left shift operation. The previous trick is formulated in (25) as follows:

$$V^2 \cong 2 * \sum_{i=0}^{\left|\frac{V}{2}\right|} V \tag{25}$$

To extend the previous formula, we can replace the division and multiplication by 2 with a variable $m$ as in (26).

$$V^2 \cong m \sum_{i=0}^{\left\lfloor \frac{V}{m} \right\rfloor} V \tag{26}$$

where $m$ = 1,2,4,8,16, 32, ..

The parameter $m$ takes only values that are multiples of 2. Increasing the value of $m$ will decrease the number of iterations required, improving the hardware performance. However, as the value of m increases, the error introduced to the squaring term increases. A pseudocode of the integral sum algorithm is shown in Figure 4.6 below.

```
1  V = input voltage; // to be squared
2  m = parameter for iterations reduction;
3  //shift right using m to reduce the iterations
4  halfV = V * 2^(1-m);
5  square = 0; // initial value
6  // to extract the integer part only
7  halfV = fix(halfV);
8  for i from 1 to halfV do{
9
10         square = square + abs(V);
11 }
12 // shift left due to reduction parameter
13 square = square * 2^(m-1);
14
```

Figure 4.6 The pseudocode of the integral sum-based squaring.

## 4.2 Error definition and calculation

Various errors are defined to assess the different approximation methods. The first two errors, $ERR_p$ and MAE examine the deviation occurred in $f(v)$ introduced in (27) where another two errors, RSEE and MERRt are applied on the Izhikevich model equations in (5), (6) and (7).

$$f(v) = 0.04v^2 + 5v + 140 \tag{27}$$

### 4.2.1 $ERR_p$ error definition

As illustrated in Figure 4.7, this error is defined in [1] as the difference between the original and the approximated curves $f(v)$ at the local minimum point (at $v = -62.5$) as in (28). This point is the initial excitation point where the neuron will require a stronger input I if the error is significant as in [1]. This error is formulated as follows:

$$ERR_p = |f(-62.5)|_{Exact} - |f(-62.5)|_{Approximated} \quad (28)$$

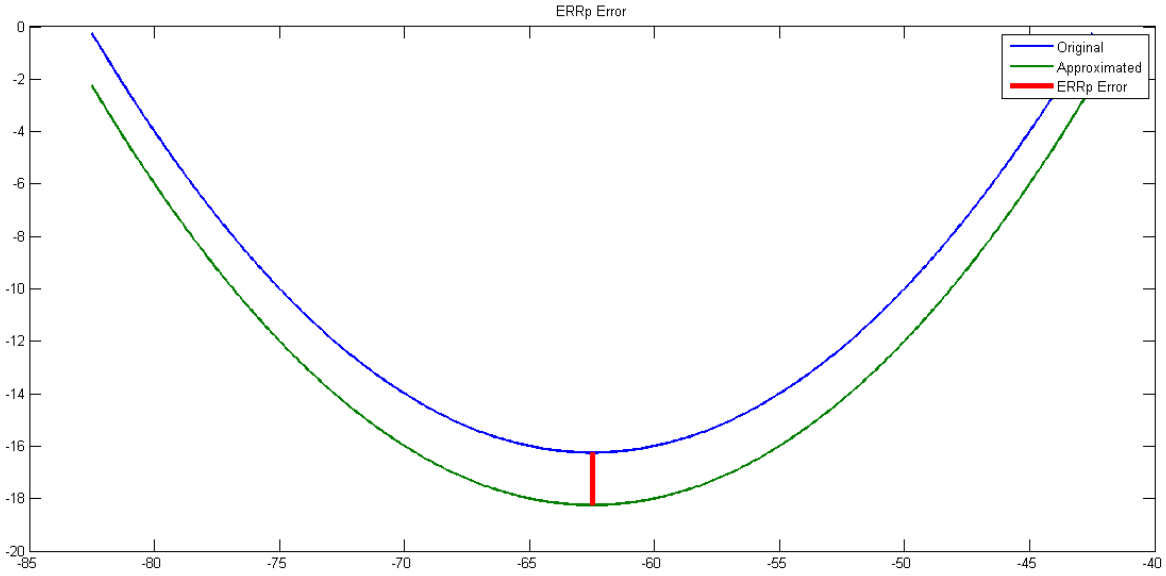


Figure 4.7 $ERR_p$ error.

### 4.2.2 MAE error definition

To generalize the concept of ERRp, Mean Absolute Error calculates the average of the differences between $f(v)$ curves for both the original and the approximated models over the range of $v$. MAE is formulated in (29) as follows:

$$MAE = \frac{\sum_i^n |f(V_i)|_{Exact} - |f(V_i)|_{Approximated}}{n} \quad (29)$$

### 4.2.3 RSEE error definition

Relative Spike Energy Error is inspired by a common concept in signal processing called “Signal Energy” which is an indication for the resemblance in shape between the spikes generated from both the original and the approximated models [16]. RSEE is unaffected by any spike timing shifts between the two models. This error is formulated in (30) as follows:

$$RSEE = \frac{\sum_i |V_i^2|_{Exact} - \sum_i |V_i^2|_{Approximated}}{\sum_i |V_i^2|_{Exact}} \times 100 \quad (30)$$

### 4.2.4 MERRt error definition

As illustrated in Figure 4.8, ERRt measures the time difference between only the first two consecutive spikes as in [14]. To make the error definition more realistic, ERRt is applied on all the spikes fired in a specific time interval where MERRt is the mean value of ERRt. This error is formulated in (31), (32) and (33) as follows:

$$ERRt = \left| \frac{\Delta t_p - \Delta t_e}{\Delta t_e} \right| \times 100 \quad (31)$$

$$where\ \Delta t = t_{apex2} - t_{apex1} \quad (32)$$

$$MERRt = \frac{1}{n} \sum_i^n ERRt_i \quad (33)$$

Where $\Delta t_p$ and $\Delta t_e$ represent the time intervals between two consecutive spikes in the approximated and the original models, respectively in (31).

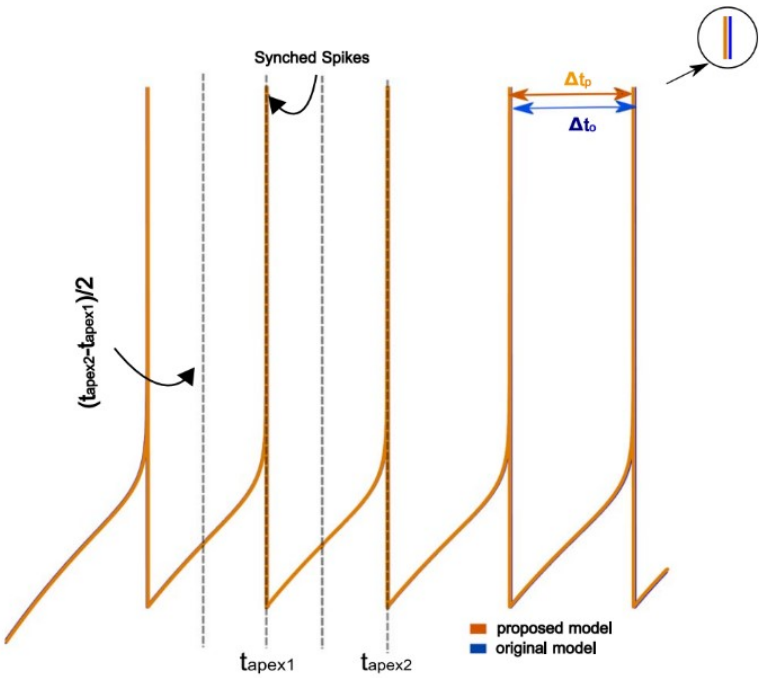


Figure 4.8 ERRt error.

### 4.2.5 MATLAB-based error calculation

After defining several errors, the different approximation methods, proposed in previous sections, are evaluated as shown in Table 4.1. Although the CORDIC algorithm has a higher MERRt error than the iterative logarithmic method, the former demostrates the least RSEE and MAE errors. As a result, CORDIC algorithm is used during this work as it shows better error results than the other methods.

Table 4.1 Error calculation of the approximation methods based on MATLAB results.

| Error type | CORDIC | | | | | Iterative Logarithmic | | | | | PWL | Integral sum | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Accuracy factor | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | ---- | 2 | 3 |
| RSEE | 3.23 | 1.27 | 0.52 | 0.19 | 0.03 | 2.73 | 1.21 | 0.67 | 0.39 | 0.1 | 1.3 | 4.53 | 9.5 |
| MERRt | 0.68 | 0.23 | 0.23 | 0.23 | 0.23 | 30.5 | 24.61 | 3.95 | 0.23 | 0 | 0.25 | 10.75 | 22 |
| MAE | 0.51 | 0.22 | 0.07 | 0 | 0 | 0.64 | 0.09 | 0.04 | 0.06 | 0.09 | 46.94 | 1.16 | 2.4 |

## 4.3 Design and system architecture

The neuron architecture is divided into four main blocks as shown in Figure 4.9. The "Reset" signal is triggered to initialize $u$ and $v$. Module 1 calculates the value of $(0.2 * v)^2$ using any of the approximation methods described before. Subsequently, modules 2 and 3 calculate the next value of $v$ and $u$, respectively. Finally, module 4 decides whether to fire a spike and reset $v$ and $u$ or to pass $v$ and $u$ without firing a spike.

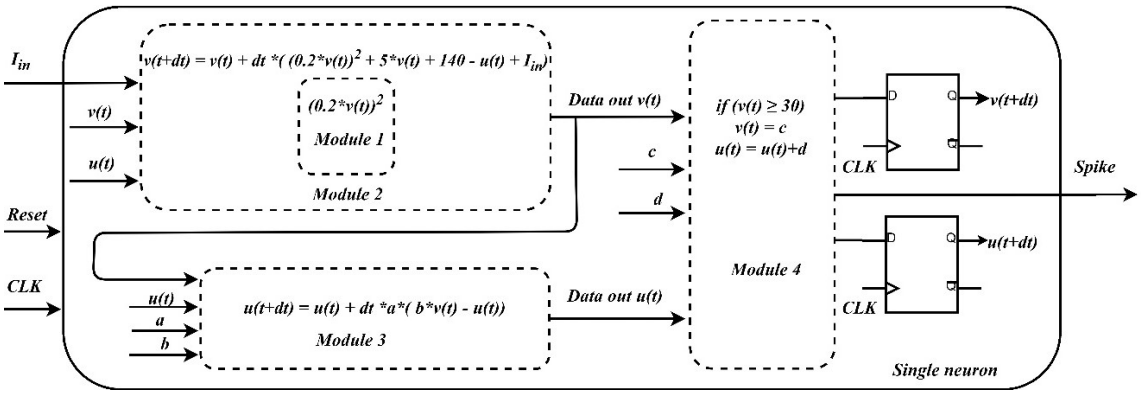


Figure 4.9 Izhikevich neuron architecture (The exact connection is not shown to avoid any confusion).

## 4.4 VERILOG code simulation and ASIC/FPGA implementation

As illustrated in the previous sections, the model uses decimal fraction numbers such as “a” and “b” parameters. Decimal fraction numbers increase system complexity in hardware level, so choosing their representation method is crucial. There are two ways of decimal fraction number representation, floating-point and fixed-point representations as shown in Figure 4.10. In this project, fixed-point representation is chosen because it is simpler, less complex in hardware level and less power consumption than floating point representation [17].

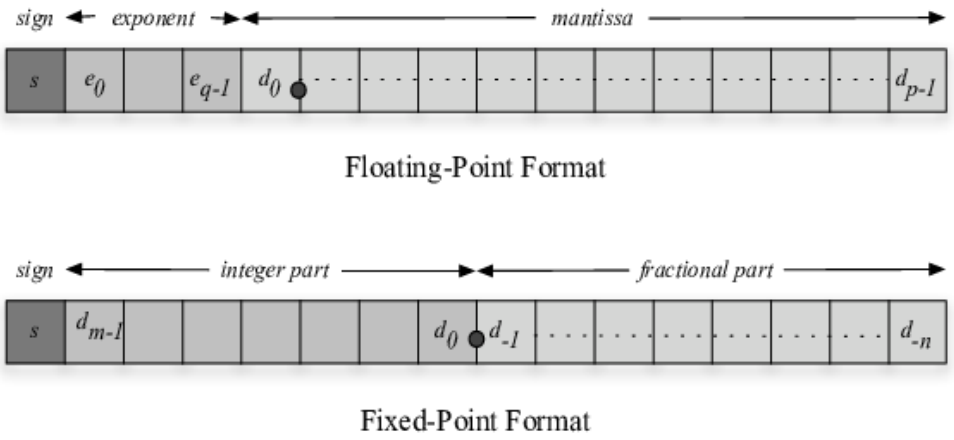


Figure 4.10 Floating-point and fixed-point representations.

In this work, Euler’s time step is set to 0.25 ms. The CORDIC-based, the PWL-based and the original Izhikevich models are hardware implemented using VERILOG to compare the error, power and area in the ASIC/FPGA platform. All constants multiplication is to be implemented using shifters and adders for all models except for the original model which uses multipliers without any approximation. For all models, the architecture is not pipelined. For the PWL-based and the CORDIC-based Izhikevich neuron, the number of bits is equal to 22 bits (12.10).

The original model is implemented with a number of bits that makes the MATLAB code results matches the VERILOG code results. It should be noted that the original model HDL code will always suffer from error due to the nature of fixed point representation. However, the error is kept as small as possible. ModelSim 10.4a is used in writing and simulation of the VERILOG code.

To verify that the VERILOG code exhibits a correct behavior like the MATLAB code, a signal called “spike” is defined that is equal to logic 1 if a spike occurs and is equal to logic 0 otherwise. As illustrated in Figure 4.11, the VERILOG code demonstrated the same behavior as the MATLAB code behavior in Figure 2.11. In this case, tonic spiking behavior is examined. To get a closer look, two rapid spikes are fired at the beginning as shown in Figure 4.12. then, a relatively slower spike is fired every certain time period.

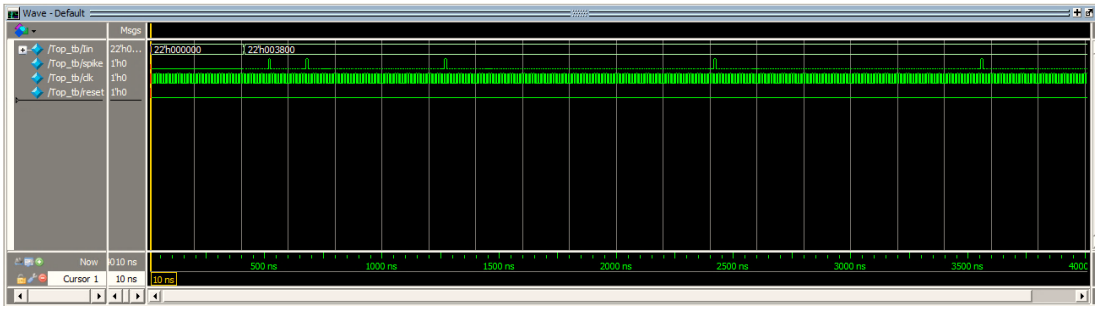


Figure 4.11 Tonic spiking behavior simulated in ModelSim.

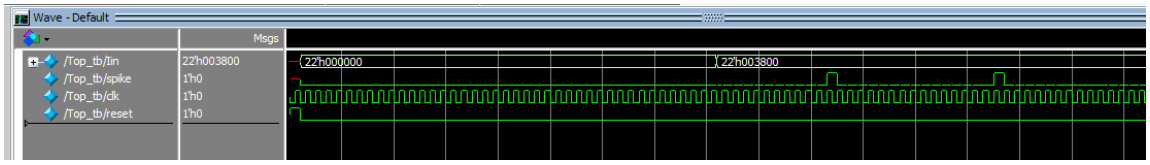


Figure 4.12 Two rapid spikes fired by the Izhikevich neuron.

After illustrated in Figure 4.13, the CORDIC-based, the PWL-based and the original Izhikevich models are hardware implemented to compare the error versus the power and the area in the ASIC platform using UMC 130 nm technology. DC Compiler Version B-2008.09 and SoC Encounter 8.1 are used in the ASIC synthesis and Place and Route (PnR) stages, respectively.

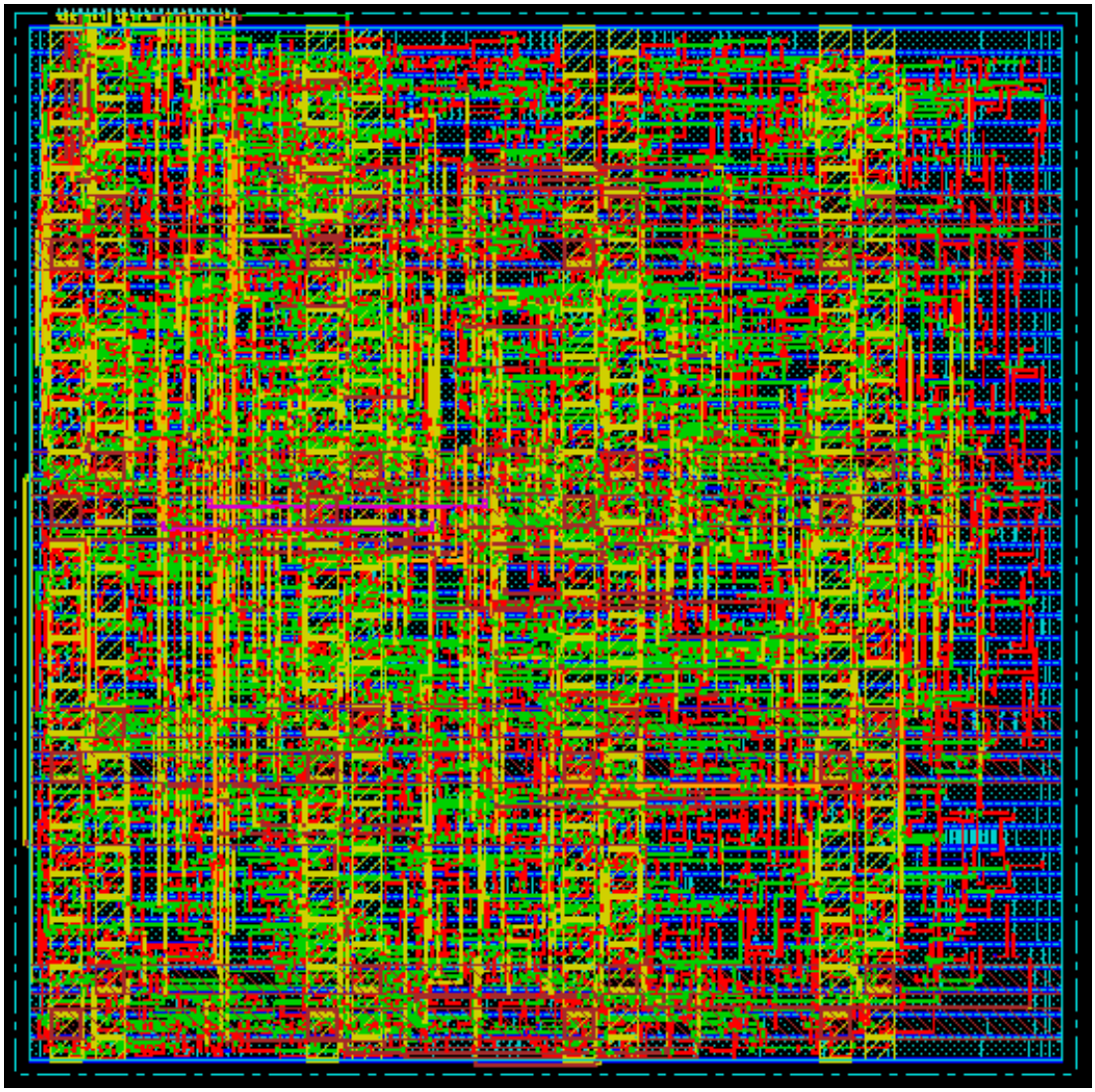

Figure 4.13 ASIC chip LAYOUT.

Furthermore, the CORDIC-based, the PWL-based and the original Izhikevich models are hardware implemented to compare the error versus the power and the area in the FPGA platform using Xilinx Zynq-7000 SoC ZC702 FPGA as illustrated in Figure 4.14. Vivado HLS 2015.2 is used in FPGA programming.

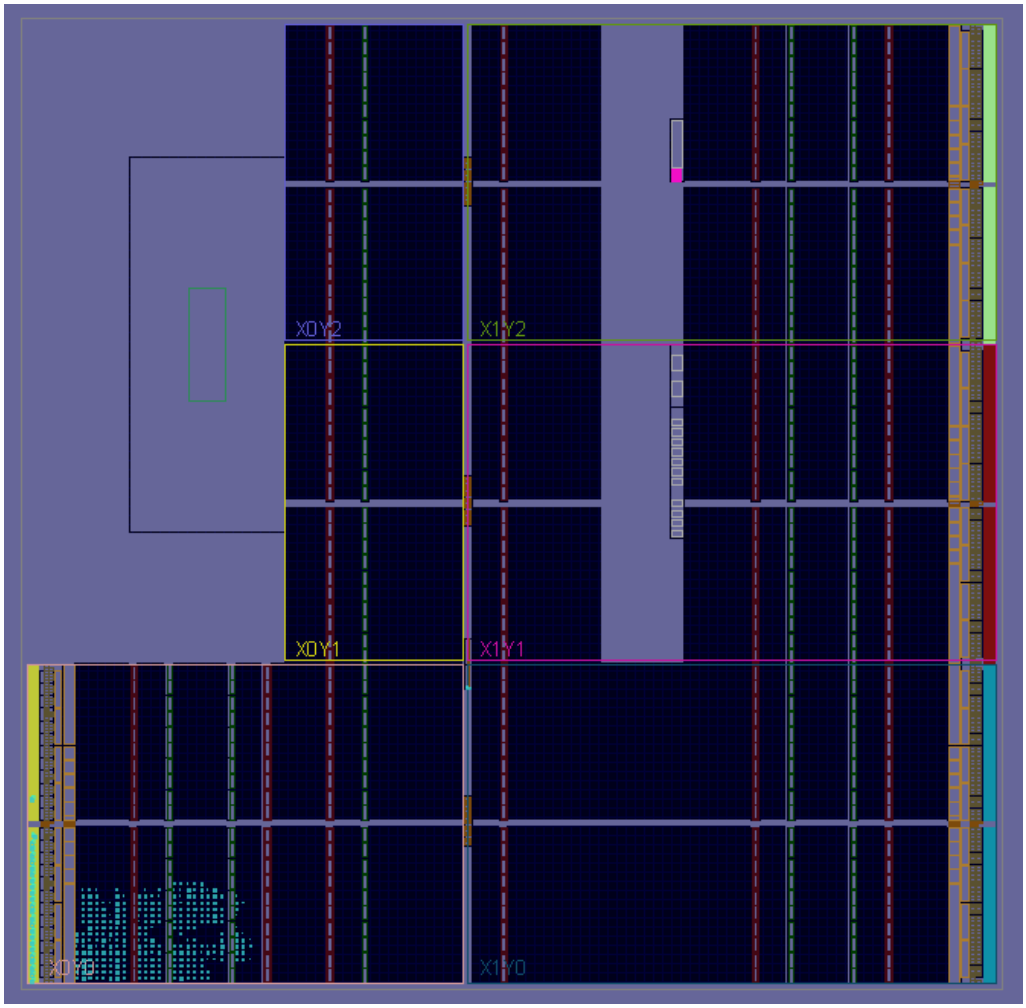


Figure 4.14 FPGA chip LAYOUT.

## 4.5 Comparison between CORDIC at *n*=10, PWL and original neuron models

### 4.5.1 Error comparison based on VERILOG results

As shown in Table 4.2, the CORDIC-based and the PWL-based models exhibits $ERR_p$ error of 0 and 0.25, respectively. This means that the initial excitation point of the CORDIC-based Izhikevich model matches the original Izhikevich model, however the PWL model is shifted by 0.25. Regarding MAE, the PWL approximation suffers from a large error compared to the CORDIC approximation. This means the PWL approximated parabola, $f(v)$ strongly deviates from the original parabola, $f(v)$.

Table 4.2 ERRp and MAE for the PWL-based and the CORDIC-based ($n$ =10) Izhikevich models.

| | PWL | CORDIC |
|---|---|---|
| **$ERR_p$** | 0.25 | 0 |
| **MAE** | 46.85 | 0.005 |

According to Table 4.3, the CORDIC-based Izhikevich model shows less error than the PWL-based Izhikevich model which means the spike in the former model suffers less from signal loss as well as timing shift.

Table 4.3 RSEE and MERRt for the PWL-based and the CORDIC-based ($n$ =10) Izhikevich models in VERILOG.

| | RSEE (%) | | MERRt (%) | |
|---|---|---|---|---|
| | **CORDIC** | **PWL** | **CORDIC** | **PWL** |
| **TS** | 0.06 | 1.18 | 1.56 | 1.11 |
| **TB** | 0.44 | 5.52 | 0 | 7.28 |
| **MM** | 0.80 | 2.46 | 2.60 | 3.28 |
| **FFA** | 0.24 | 0.10 | 1.13 | 0.74 |
| **Mean Value (%)** | 0.39 | 2.32 | 1.32 | 3.10 |

### 4.5.2 Power/Area comparison in ASIC/FPGA platforms

According to Table 4.4, the CORDIC-based Izhikevich model hardware implementation shows significant improvement compared to the original Izhikevich model hardware implementation in ASIC. Although the PWL-based hardware implementation of the Izhikevich model shows slight improvement than the CORDIC-based hardware implementation of the Izhikevich model, the latter model is more accurate as in Table 4.2 and Table 4.3.

Table 4.4 The ASIC implementation comparison for the original, the CORDIC-based ($n$ =10) and the PWL-based Izhikevich models at frequency = 9.1 MHz.

| | Original | CORDIC | PWL |
|---|---|---|---|
| **Area ($\mu m^2$)** | 69149 | 25894 | 21076 |
| **Power (mW)** | 1.06 | 0.40 | 0.30 |

In order to further investigate the hardware implementation performance of the three designs, the designs are also implemented on Xilinx Zynq-7000 SoC ZC702 FPGA where a comparison between their power and area is performed in Table 4.5. The area in FPGA is represented by the number of LUTs, registers and DSP. According to Table 4.5, the CORDIC-based Izhikevich neuron model shows significant improvement compared to the original Izhikevich neuron model in power and area perspective. However, the PWL-based Izhikevich neuron model consumes less power and area than the CORDIC-based Izhikevich neuron model in FPGA platform.

Table 4.5 The FPGA implementation comparison for the original, the CORDIC-based ($n$ =10) and the PWL-based Izhikevich models at frequency = 9.1 MHz.

| | **Original** | **CORDIC** | **PWL** |
|---|---|---|---|
| **Power (mW)** | 3.73 | 2.726 | 1.984 |
| **No. of LUTs** | 1030 | 961 | 811 |
| **No. of registers** | 64 | 44 | 44 |
| **No. of DSPs** | 10 | 0 | 0 |

## 4.6 Power/Area/Error trade-off in ASIC platforms

As seen in the previous section, the CORDIC-based neuron model consumes higher power and area than the PWL-based neuron model. Consequently, a sweep over $n$ from 0 to 10 is performed to decrease the power and area of the CORDIC approximation at the cost of additional error. $n$ = 0 means the approximation is performed on integer term only as if there is no fraction term. When implemented in ASIC platform, the CORDIC-based Izhikevich model power ranges from 0.26 mW to 0.40 mW when $n$ is changed from 0 to 10. The study performed in this section is based on ASIC results.

A Figure of Merit (FoM) is defined to show the tradeoff among error (ERR), power (P) and area (A) and determine the value of $n$ at which the FoM is minimum. ERR is the arithmetic mean of RSEE and MERRt. as in (34).

$$ERR = \frac{1}{2}(RSEE + MERRt) \tag{34}$$

The arithmetic mean (AM) is used in ERR because RSEE and MERRt have the same

numeric range (0 – 100 %). However, when combining the power and the area, the geometric mean (GM) is preferred over the AM due to having different numeric ranges. GM prevents factors with large values from dominating in mean calculation. In addition, a weighted GM ($GM_W$) is used to increase the contribution of one term over the others depending on the design demands as in (35). In this work, it is chosen to make the power and area of higher weight than the error as given in (36) since the paper main focus is on the low power hardware implementation.

$$GM_W = \sqrt[\Sigma_{i=1}^{n} w_i]{\prod_{i=1}^{n} x_i^{w_i}} \tag{35}$$

$$FoM = \sqrt[5]{ERR * P^2 * A^2} \tag{36}$$

According to Figure 4.15, the value of $n$ at which the FoM is minimum is 5. Depending on the problem constraints, one can choose whether to reduce the power consumption (decrease $n$) or to increase the accuracy (increase $n$) where such a tradeoff offers design flexibility.

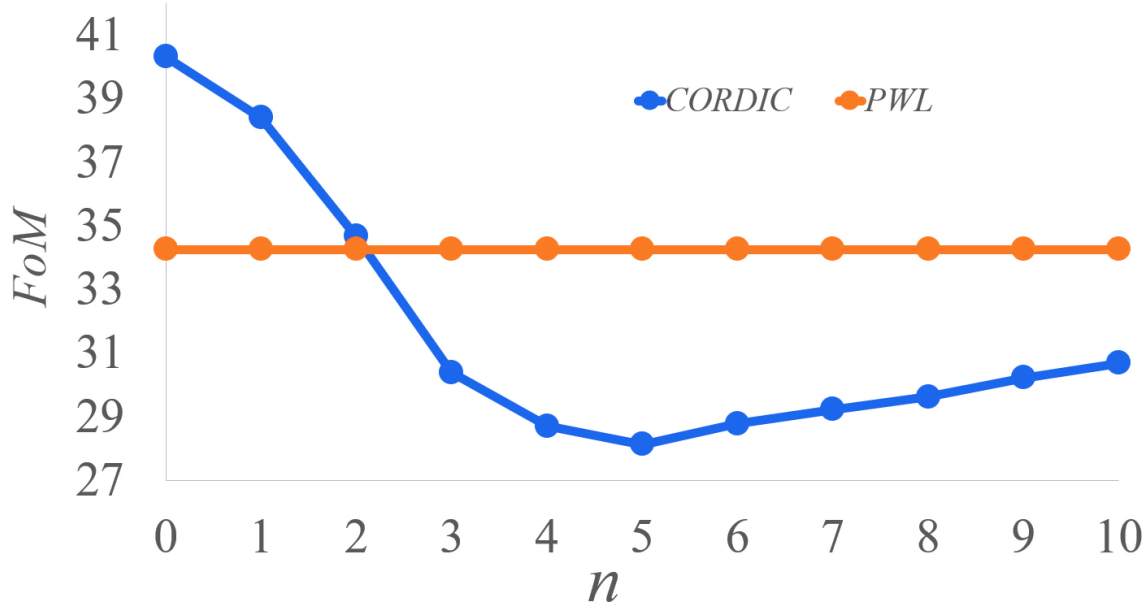


Figure 4.15 FoM versus n where the point at which ERR, P and A are minimum is $n = 5$

## 4.7 Comparison between CORDIC at *n*=5, PWL and original neuron models

### 4.7.1 Error comparison based on VERILOG results

According to Table 4.6 and Table 4.7, the CORDIC approximation at $n$ = 5 has less error than the PWL for the four types of errors.

Table 4.6 ERRp and MAE for the PWL-based and the CORDIC-based ($n$ =5) Izhikevich models.

| | PWL | CORDIC |
|---|---|---|
| **$ERR_p$** | 0.25 | 0 |
| **MAE** | 46.85 | 0.17 |

Table 4.7 RSEE and MERRt for the PWL-based and the CORDIC-based ($n$ =5) Izhikevich models in VERILOG.

| | RSEE (%) | | MERRt (%) | |
|---|---|---|---|---|
| | **CORDIC** | **PWL** | **CORDIC** | **PWL** |
| **TS** | 0.23 | 1.18 | 1.56 | 1.11 |
| **TB** | 0.33 | 5.52 | 0.24 | 7.28 |
| **MM** | 1.20 | 2.46 | 3.35 | 3.28 |
| **FFA** | 1.13 | 0.10 | 0.40 | 0.74 |
| **Mean Value (%)** | 0.72 | 2.32 | 1.39 | 3.10 |

### 4.7.2 Power/Area comparison in ASIC/FPGA platforms

According to Table 4.8, the CORDIC-based Izhikevich neuron model consumes almost the same power and area as the PWL-based Izhikevich neuron model when implemented in ASIC platform. Also, both of them still show better hardware performance than the original Izhikevich model.

Table 4.8 The ASIC implementation comparison for the original, the CORDIC-based ($n$ =5) and the PWL-based Izhikevich models at frequency = 9.1 MHz.

| | Original | CORDIC | PWL |
|---|---|---|---|
| **Area ($\mu m^2$)** | 69149 | 22088 | 21076 |
| **Power (mW)** | 1.06 | 0.33 | 0.30 |

Then, a comparison between their power and area is performed in Table 4.9, when implemented on FPGA. According to Table 4.9, the CORDIC-based Izhikevich neuron model shows significant improvement compared to the original Izhikevich neuron model in power and area perspective. The PWL-based Izhikevich neuron model still consumes less power and area than the CORDIC-based Izhikevich neuron model in FPGA platform, however, the 2 designs have become closer in hardware performance when CORDIC is implemented with $n = 5$ than the previous study when CORDIC is implemented with n = 10.

Table 4.9 The FPGA implementation comparison for the original, the CORDIC-based (n =5) and the PWL-based Izhikevich models at frequency = 9.1 MHz.

| | **Original** | **CORDIC** | **PWL** |
|---|---|---|---|
| **Power (mW)** | 3.73 | 2.319 | 1.984 |
| **No. of LUTs** | 1030 | 848 | 811 |
| **No. of registers** | 64 | 44 | 44 |
| **No. of DSPs** | 10 | 0 | 0 |

# CHAPTER 5

## Spiking Neural Network

### 5.1 MNIST database

The learning methodology used in simulating the neural network is supervised learning. Hence, a dataset has to be used to test and train the network. One of the most widely used datasets in machine learning systems is MNIST data set. It consists of handwritten numbers from zero to nine with different styles as illustrated in Figure 5.1. This means the network output layer contains 10 neurons, where each neuron represents a number from 0 to 9. The MNIST data set is inspired from NIST's data set which is collected from American employees (training data) and American high school students (testing data). However, MNIST data is black and white images only with fixed and limited number of pixels. The training data set consists of 60,000 images while the testing data set consists of 10,000 images. Half of the training data set as well as the testing data set are extracted from NIST's training and testing data sets. Each set is accompanied by its labels to train and test the network. Our network uses all the training and testing images [2].

Figure 5.1 Sample from the MNIST dataset.

## 5.2 Input image preparation

Before the network starts the computation process, the input image has to be prepared as illustrated in Figure 5.2. The images are converted from gray scale to black and white scale. The conversion is performed by multiplying the image by a step function to determine the black and white regions of the pixels. Each image has 28*28 pixels (784 pixels) and an equivalent label. Each squared-image is converted into a single columned-image of 784 pixels in total.

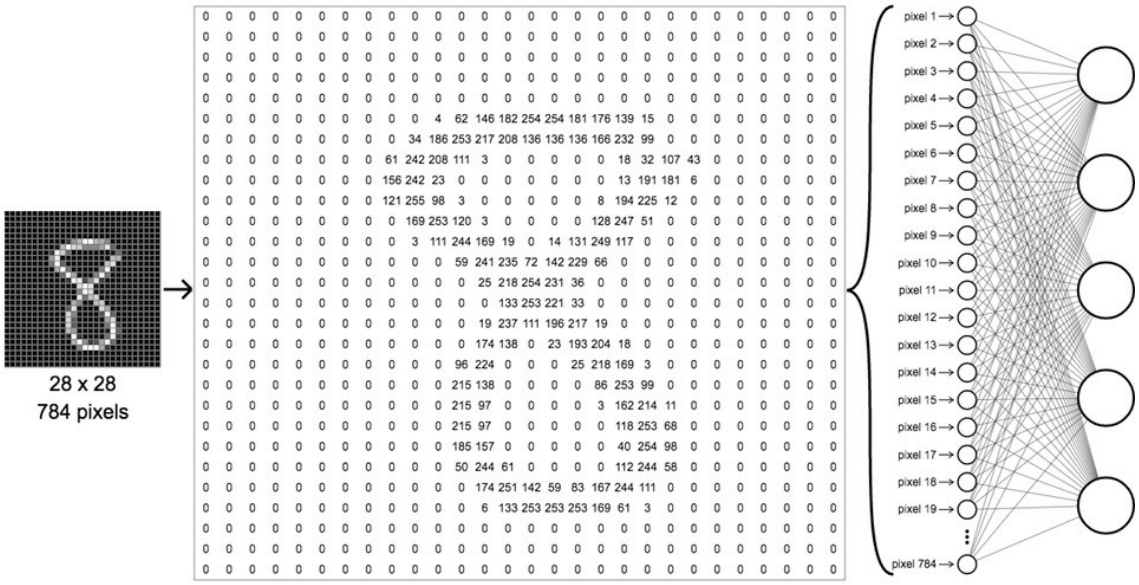


Figure 5.2 Input image pixel map of MNIST dataset.

## 5.3 Network structure

As illustrated in Figure 5.3, the network structure is feedforward and consists of two neural layers (The input pixels are not counted as a layer). The first layer, which is called the hidden layer, contains 200 neurons connected to the input image pixels through 200 synapses.

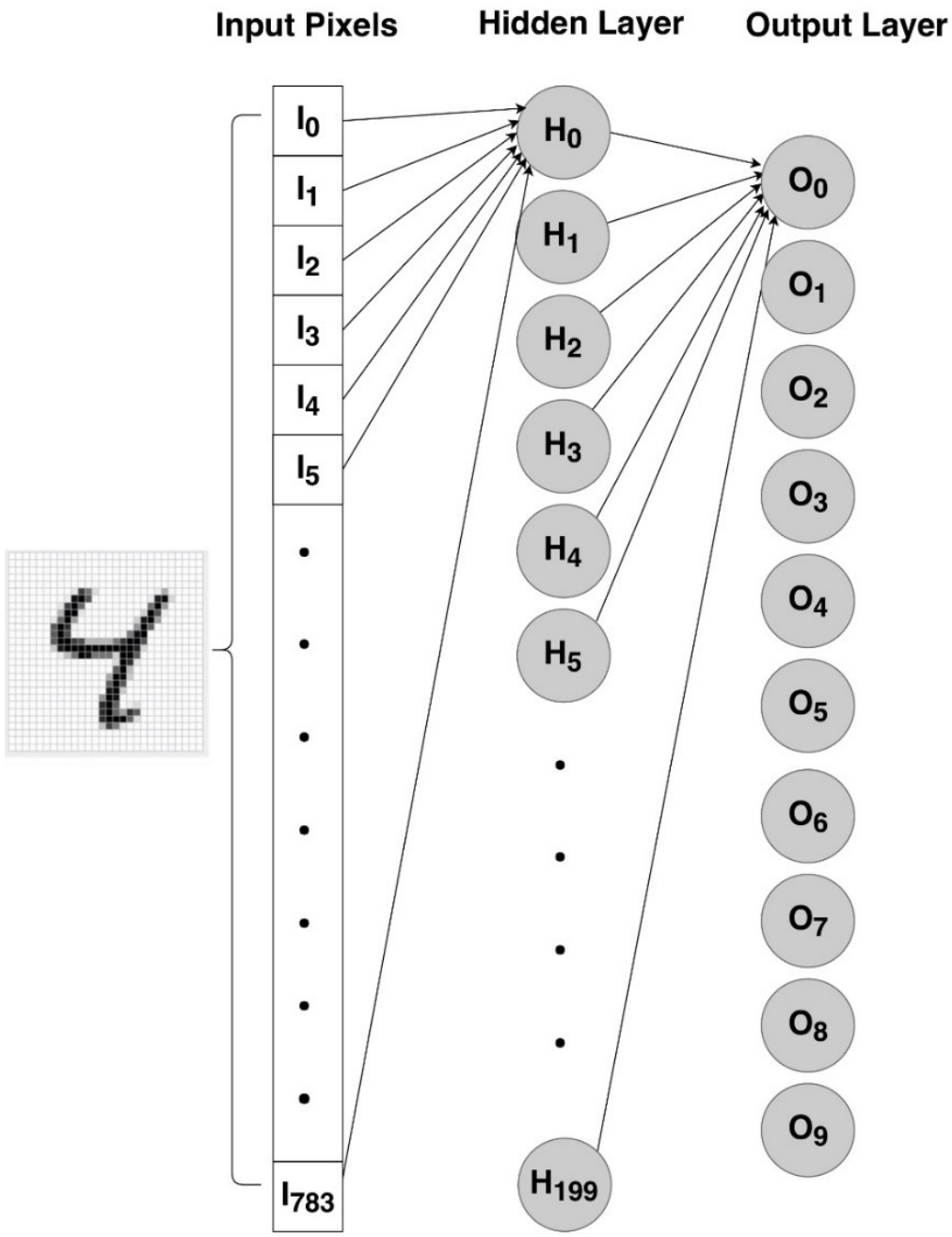


Figure 5.3 Our Feedforward Spiking Neural Network Structure.

Each arrow in Figure 5.3 represents a synapse whose objective is to convert each pixel into an input current governed by the following equation in (37):

$$I_i = I_0 + \sum_{j=1}^{S} W_{i,j} * Ij \tag{37}$$

where $I_i$ is the input current for the neuron $i$. $I_0$ represents the initial biasing current for the neuron used to make sure the neuron is always in the on-state. $Ij$ refers to the pixel value which is equal to 1 if white and 0 if black. The weights matrix, connecting the input pixels with the first layer, is $W_{i,j}$. $W_{i,j}$ represents the connection strength between $Ij$ (pixel) and $I_i$ (neuron). the parameter $S$ is equal to the total number of pixels. As a result, the total number of pixels is determined earlier before the network operates.

The second layer, which is called the output layer, consists of 10 neurons where each neuron represents a digit from zero to nine. Similarly, there are synapses between the hidden layer and the output layer. The biasing current is the same as before but, the dimensions of the weights matrix is changed.

## 5.4 Rate-based Neural coding and Backpropagation algorithm

As we mentioned before, each neuron fires a train of spikes. So, how does the neural network translate these trains of spikes into a decision? The neural network translates the neurons' output using a concept called neural coding. Our spiking neural network uses one type of neural coding called rate coding where the network decision is based on neurons' firing rates. The rate coding states that as the neuron input current increases, the neuron's frequency increases. As a result, if a neuron's input current is higher than another neuron's input current, the former's frequency will be higher than the latter's frequency. So, the firing frequency is function of the input current which is modified by increasing/decreasing the current. To modify the input currents, rate coding modifies the weights connecting the neurons. The neuron frequency is defined as the number of spikes fired divided by the time interval as illustrated in Figure 5.4 [18].

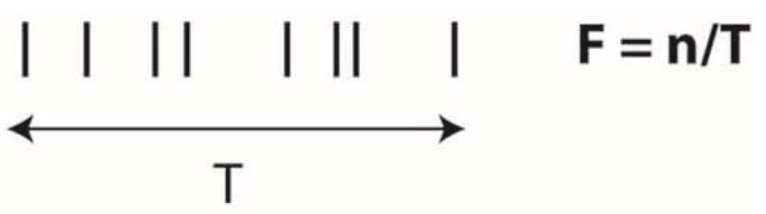


Figure 5.4 Neuron frequency calculation.

Furthermore, the backpropagation algorithm is used to calculate the error where the weights modification is based on this error as shown in (38). The following equation determines the delta weights to reach the desired output from the network [1].

$$\Delta W_{i,j} = \alpha * I_i * (f_{actual} - f_{desired}) \tag{38}$$

where $\alpha$ is the learning rate that is determined before constructing the network and fixed during the learning process. The observed frequency from each neuron is called $f_{actual}$ while the target frequency is called $f_{desired}$. The high frequency is the target frequency for the correct output neuron and the low frequency is the target frequency for the other neurons. In this network, the high frequency is chosen to be 30 spikes per 100 ms (300 Hz) while the Low frequency is 1 spike per 100 ms (10 Hz). to put an upper bound and a lower bound for the weights, a sigmoid function can be used during the learning process. Sigmoid function, $\sigma(I_i)$ is applied to the input current, $I_i$ in the delta weights equation. The modified weight is formulated in (39) as follows [1]:

$$W_{i,j\,modified} = W_{i,j\,current} - \alpha * \sigma(I_i) * (f_{actual} - f_{desired}) \tag{39}$$

For more details, the proof of (39) is explained in [1].

## 5.5 A case study of training and testing the Spiking Neural

To understand the rate coding, Figure 5.5 shows the behavior of two neurons in the output layer before and after the learning operation. First, an image containing number 5 is inserted to the network then, the behavior of these two neurons is observed. If the first neuron fires spikes with a high frequency, this means the input image is “1”.

Similarly, if the second neuron fires spikes with a high frequency, this means the input image is “5”. As shown in Figure 5.5 (a) and (b), the network can’t figure out the number whether it is one or five because both neurons fire with a high frequency. After that, the learning process takes place so that the weights can be updated. when the network is tested again, the neuron #5 has a higher firing rate than neuron #1 as illustrated in Figure 5.5 (c) and (d). Finally, the network decides that the input number is 5.

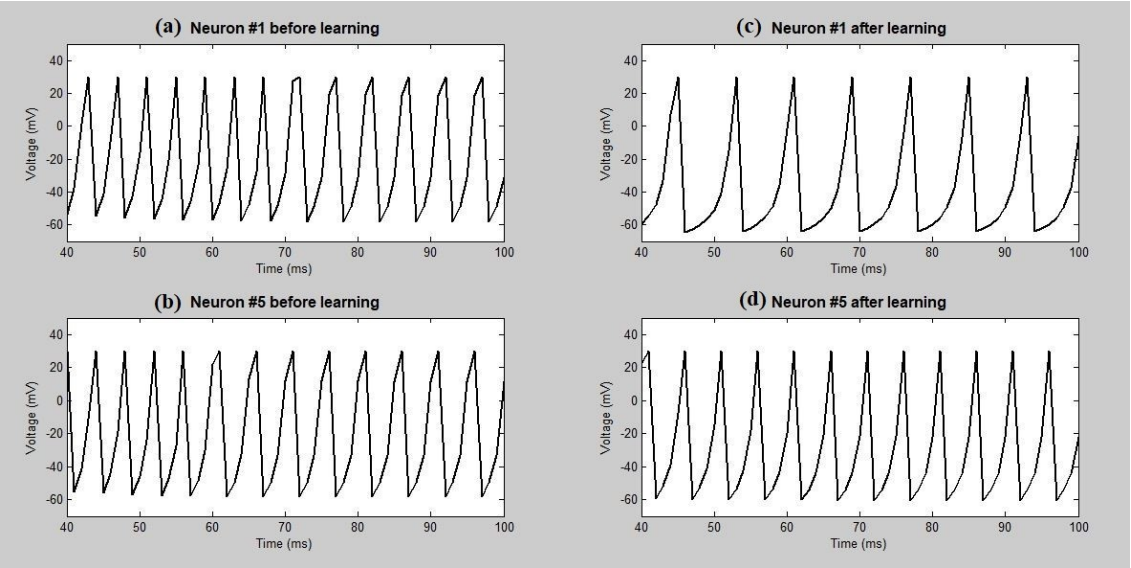


Figure 5.5 This study is conducted using only one training image. (a) The behavior of neuron #1 before the learning process. (b) The behavior of neuron #5 before the learning process. (c) The behavior of neuron #1 after the learning process. (a) The behavior of neuron #5 after the learning process.

After training the network with the whole MNIST dataset (60,000 images), the network is tested using an unknown new handwritten style of number 5. As shown in Figure 5.6, the frequency of the target neuron #5 is higher than the frequency of neuron #1. This means that the learning process is going well and the network is able to determine the number #5 whether it saw the exact handwritten style before or the handwritten style is new to the network as illustrated in Figure 5.5 and Figure 5.6.

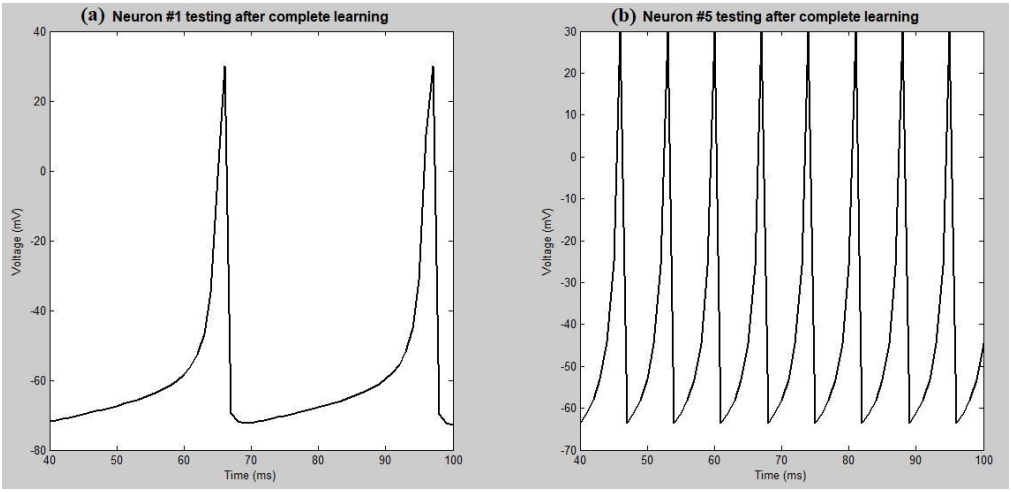


Figure 5.6 This study tests the network using a new image of number five after training the network with 60,000 images. (a) The low frequency of neuron #1. (b) The high frequency of neuron #5.

## 5.6 Spiking Neural Network error evaluation

### 5.6.1 Error based on the used number of training images versus the network accuracy

This study is conducted to show how the network accuracy is affected by the number of training images. During this study, the network is provided with a batch size equals 5000 then, the network accuracy is tested. As illustrated in Figure 5.7, the network accuracy increases as the number of images increase. At the beginning, the accuracy is badly affected by the low number of training images. Then, the accuracy reaches a saturation value where the improvement in accuracy is small.

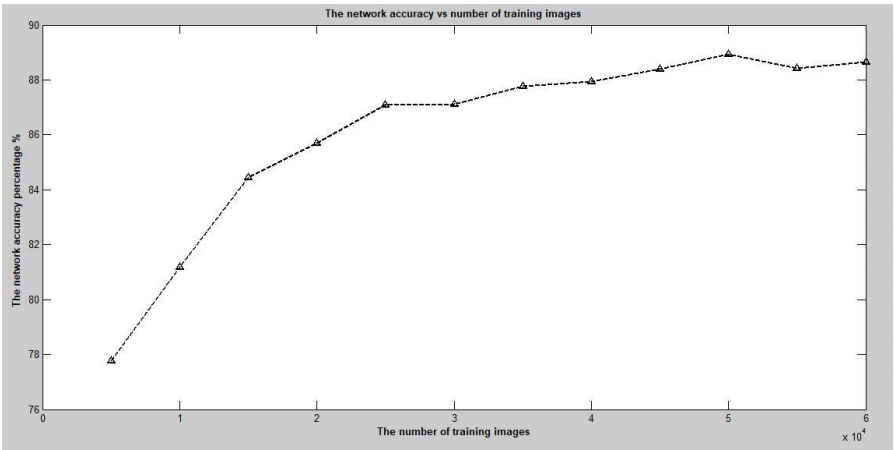


Figure 5.7 The number of taining images versus the network accuracy.

### 5.6.2 Error based on the frequency of the target neuron versus the number of learning iterations

Ideally, the target neuron fires with $f_{high}$ = 300 Hz. However, the network does not behave correctly at the beginning where the target neuron does not fire spikes with the ideal $f_{high}$. Based on this observation, a relative error function is defined as in (40).

$$E = \frac{f_{high} - f_{actual}}{f_{high}} \tag{40}$$

where $f_{high}$ is the target neuron (ideal) high frequency, fixed at 300 Hz, while $f_{actual}$ is the target neuron actual frequency. This error decreases as the network trains more. As illustrated in the Figure 5.8, the error decreases as the number of learning iterations increases until the error settles down around 30%. This means after a certain number of images, the training process does not affect the network. It should be noted that the batch size used in this study equals to 100 images.

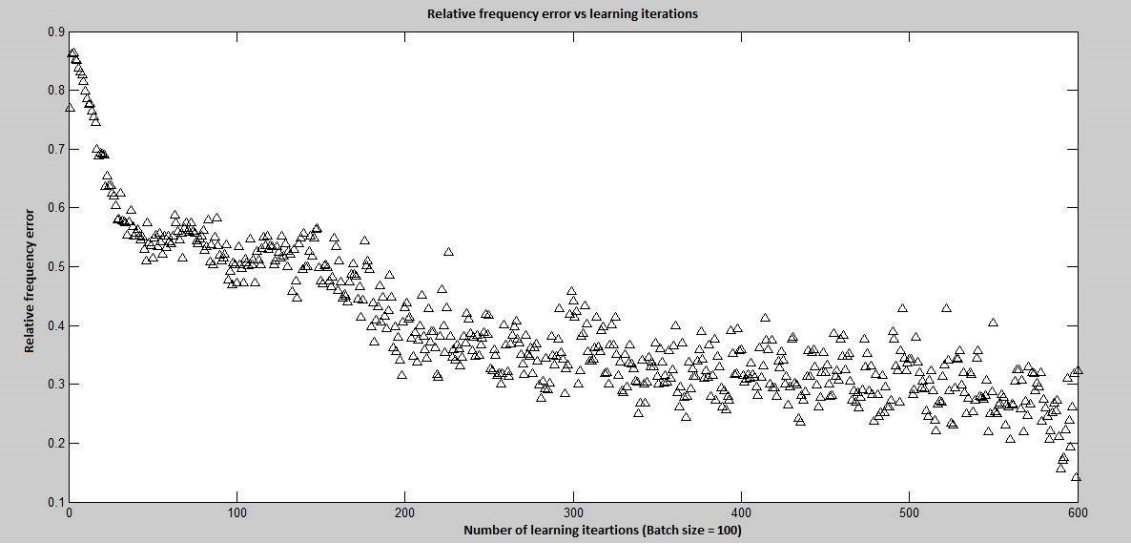


Figure 5.8 Relative frequency error versus the number of learning iterations. After training with 60,000 images, the error saturates around 30% at the end.

### 5.6.3 Error based on the output value versus the number of learning iterations

Another average error is defined as the difference between the output number and the target number. As shown in Figure 5.9, the value of the average error decreases as the learning iterations increase. When the error tends to zero, the network shows a correct decision. Also, this type of error saturates around a certain number at the end of the learning process.

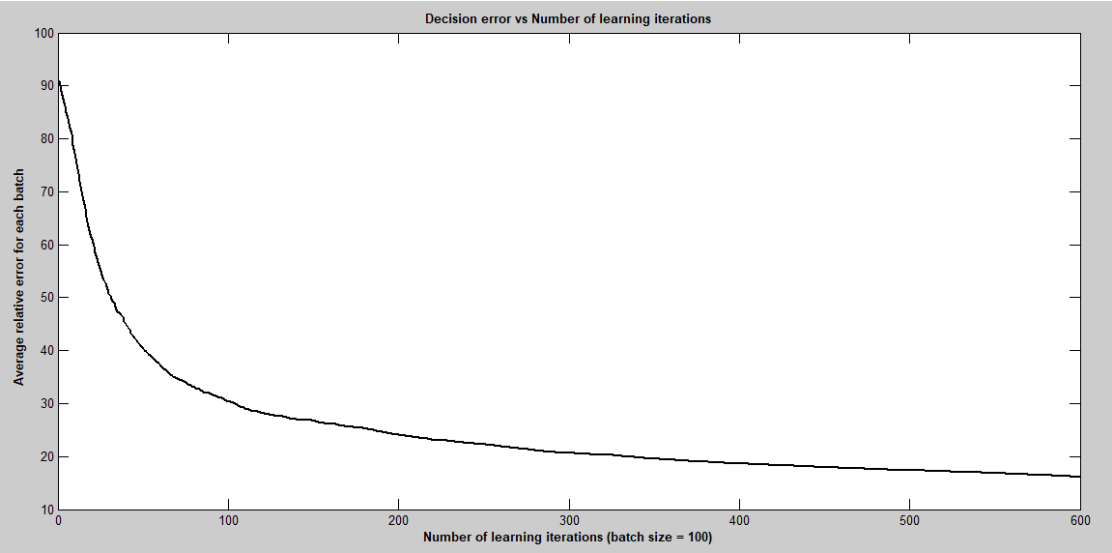


Figure 5.9 The relation between the output number and the target number versus the learning iterations.

### 5.6.4 The effect of error reduction on the delta weight value

Due to the error reduction, the value of the delta weight decreases during the learning process as the delta weight is function of the error. As the delta weights decreases, this means the network is close to the correct weights. When the network reaches the exact correct weights, the weights saturate at their current values. As shown in Figure 5.10, the delta weight is relatively large at the beginning of the learning operation then, it decreases until saturation occurs. The curve does small oscillations around the zero delta weight which means there is no longer weight modification. Occasionally, the network has learned and reached the correct weights.

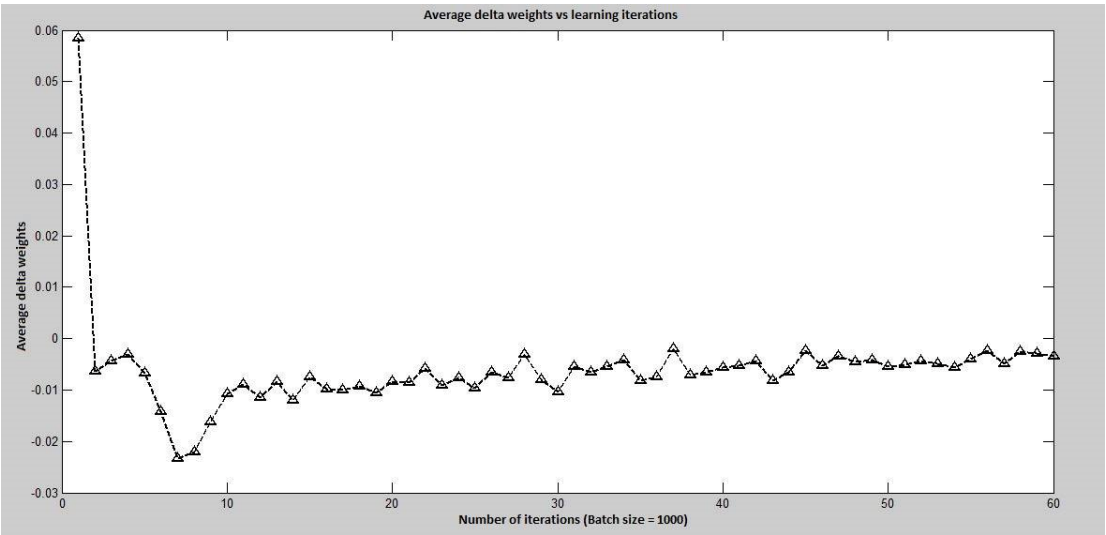


Figure 5.10 The delta weights changes during the learning process.

## 5.7 Spiking Neural Network comparison

Three independent spiking neural networks are constructed using original, CORDIC-based and PWL-based neurons. Each one takes the learning and testing operations separately. There are three factors used to compare the simulation results of each network, the accuracy, the estimated power consumption and the estimated area. After completing the learning operation using 60,000 images, the testing process begins with 10,000 images in which the first factor, the accuracy, is calculated. The network accuracy ($Accuracy_{Network}$) is defined in (41) as the number of correct network's decisions ($n_{Correct}$) divided by the total number of testing images ($n_{Testing}$ = 10,000).

$$Accuracy_{Network} = \frac{n_{Correct}}{n_{Testing}} * 100\ \% \tag{41}$$

Since the network is simulated in MATLAB, there is no way to calculate the actual power consumption. Instead, the second factor, which is the estimated power consumption, is calculated. We assume that the main power consuming source in the network is the neuron. So, the estimated network power ($P_{Network}^{Estimated}$ ) is determined in (42) by the number of used neurons ($n_{Neurons}$) multiplied by the power of a single neuron ($P_{Neuron}$).

$$P_{Network}^{Estimated} = n_{Neurons} * P_{Neuron} \quad (42)$$

Of course, this value isn't accurate for determining the actual network power consumption. Although it is a rough estimation, the estimated value gives us an indication about the expected true power consumption value. Similarly, the third factor, which is the estimated area, is calculated as the estimated power. The estimated area ($A_{Network}^{Estimated}$) equals to the number of neurons ($n_{Neurons}$) multiplied by the area of a single neuron ($A_{Neuron}$) as shown in (43).

$$A_{Network}^{Estimated} = n_{Neurons} * A_{Neuron} \quad (43)$$

The simulation results in Table 5.1 show us the original based SNN achieves the highest accuracy which is 89%. However, its estimated power and area is significantly high compared to CORDIC and PWL based SNN. CORDIC-based SNN achieves power reduction percentage from 60% to 74% of the original based SNN power consumption according to the accuracy parameter, *n*. Besides, CORDIC-based SNN achieves area reduction percentage from 62% to 76% of the original-based SNN area according to the accuracy parameter, *n*. Table 5.1 also shows that the accuracy of the CORDIC-based SNN ranges from 86.5% to 88% over the parameter *n* sweep. The estimated power of the CORDIC-based SNN ranges from 55 mW to 84 mW when *n* is changed from 0 to 10.

Table 5.1 Spiking Neural Network performance based on CORDIC, PWL and Original neuron models.

| | **Original** | **CORDIC** | | | **PWL** |
|---|---|---|---|---|---|
| **Accuracy parameter** | - | *n* = 0 | *n* = 5 | *n* = 10 | - |
| $\boldsymbol{Accuracy_{Network}}$ **(%)** | 89 | 86.5 | 87 | 88 | 85.5 |
| $\boldsymbol{P_{Network}^{Estimated}}$ $\boldsymbol{(mW)}$ | 210 | 55 | 69 | 84 | 63 |
| $\boldsymbol{A_{Network}^{Estimated}}$ $\boldsymbol{(mm^2)}$ | 14.5 | 3.5 | 4.5 | 5.5 | 4.5 |

# CHAPTER 6

## Real time FPGA outputs demonstration

### 6.1 The experimental setup of the FPGA implementation

For further verification, the FPGA is programmed with a single CORDIC-based Izhikevich neuron and connected to an oscolloscope to see real time outputs. The purpose is to make sure the neuron exhibits the biological behaviors correctly, In this study, the tonic spiking behavior is investigated. Spartan-6 FPGA SP605 Evaluation Kit is used during this study as shown in Figure 6.1 and programmed using ISE Design Suite 14.7. A signal called spike in the VERILOG code is connected to a (General-Purpose Input/Output) GPIO pin in the FPGA. Spike signal equals logic 1 if there is a spike and logic 0 otherwise.


Figure 6.1 Spartan-6 FPGA SP605 Evaluation Kit.

Then, the GPIO pin is connected to Digilent Analog Discovery 2 kit shown in Figure 6.2. This kit has several functionalities such as oscilloscope, function generator, logic analyzer and others. In this study, Digilent Analog Discovery 2 kit is used as an oscilloscope to demonstrate the spike signal and see the tonic spiking behavior.

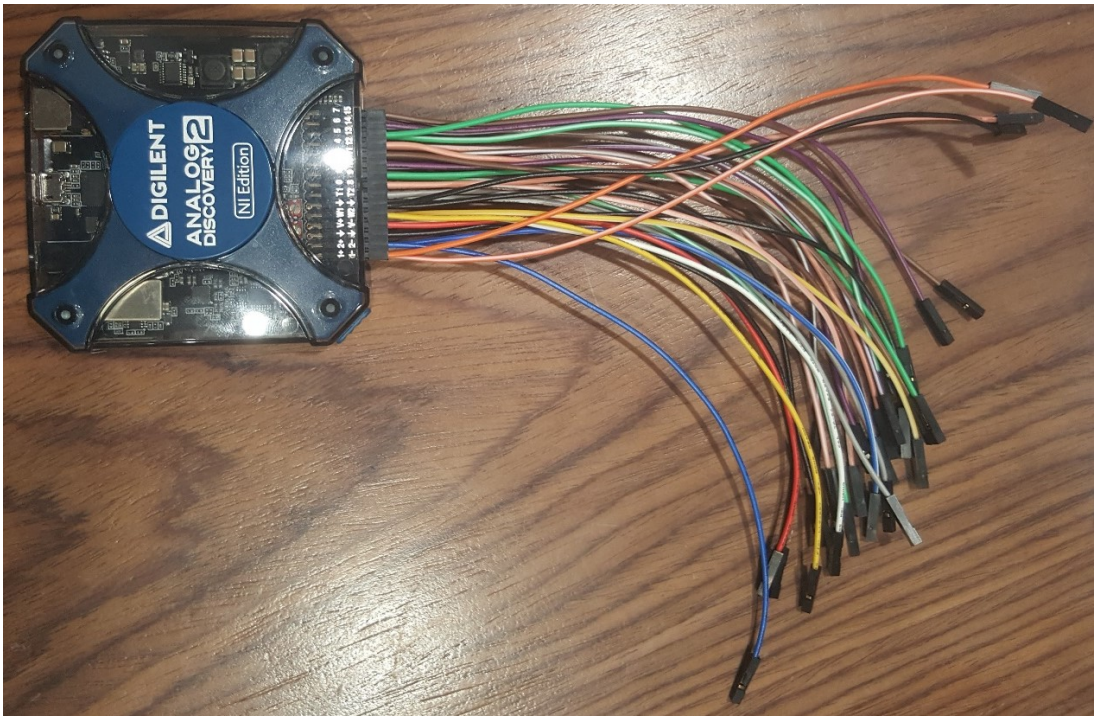


Figure 6.2 Digilent Analog Discovery 2 kit.

The connection between the FPGA and the discovery kit is simple as illustrated in Figure 6.3. The oscilloscope is a differential pair oscilloscope which has positive and negative terminals. The positive terminal of the oscilloscope is connected to the GPIO pin where the GPIO pin is connected to the spike signal in the design. The negative terminal of the oscilloscope, the ground of the discovery kit and the ground of the Spartan FPGA are all connected together to make a common ground connection.

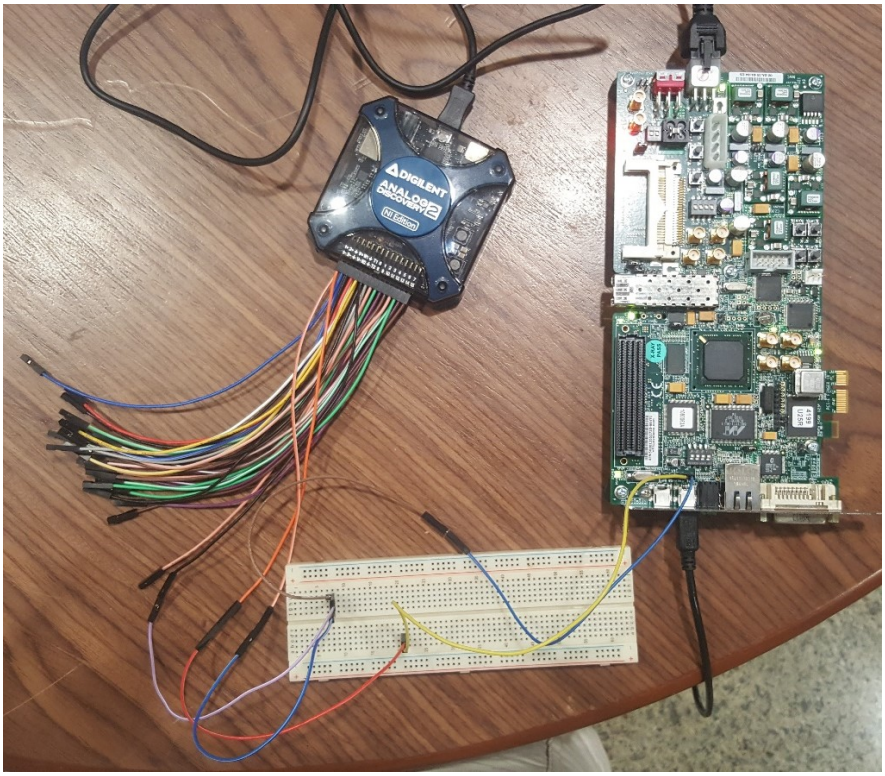


Figure 6.3 The connection between the Spartan FPGA and the Discovery kit to demonstrate the spike signal on the oscilloscope.

. As illustrated in Figure 6.4, this is the whole system where the laptop on the right is used to program the Spartan FPGA and the laptop on the left is used to see the output of the oscilloscope of the Discovery kit.

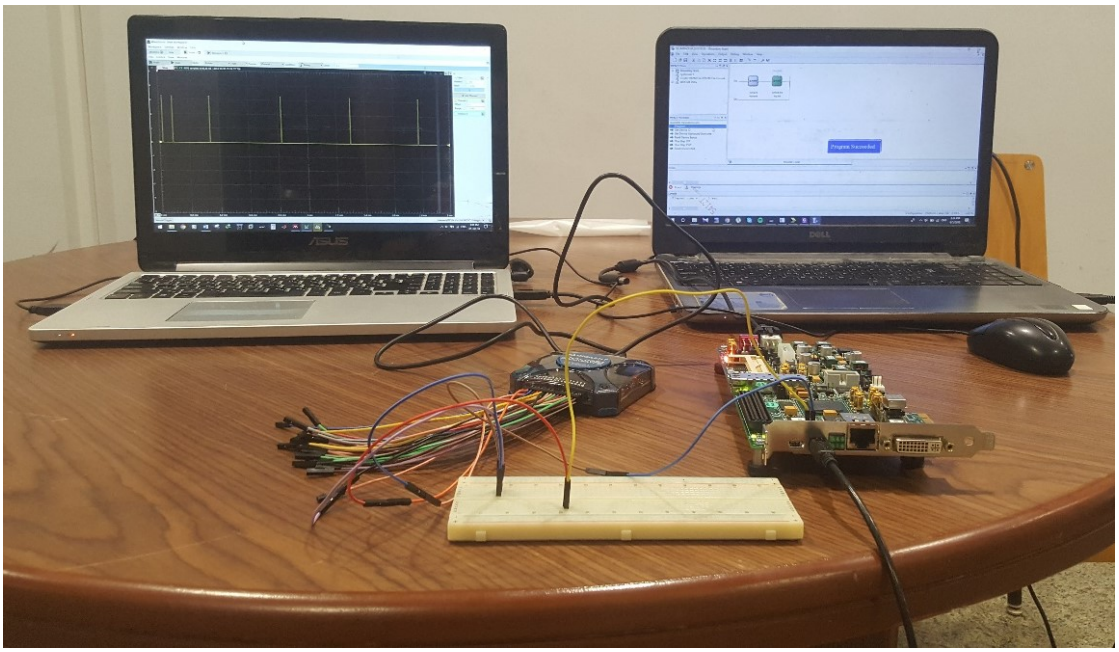

Figure 6.4 The whole system altogether.

## 6.2 The real time results of the FPGA implementation

By comparing Figure 6.5 with Figure 2.11, the CORDIC-based Izhikevich neuron can demonstrate the tonic spiking behavior correctly. 2 rapid spikes are fired at the beginning, then a constantly periodic spike is fired.

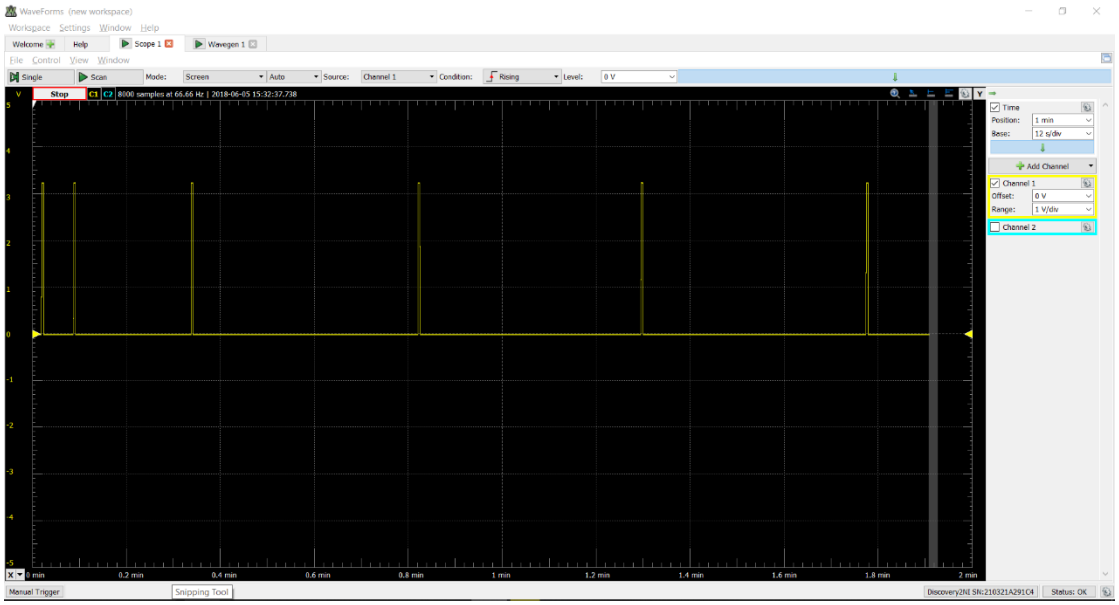

Figure 6.5 The tonic spiking behavior demonstrated by the spike signal.

To take a closer look, Figure 6.6 illustrates the 2 rapid spikes fired at the beginning. As seen below, the pulse of the spike signal looks perfect and does not suffer from noise.

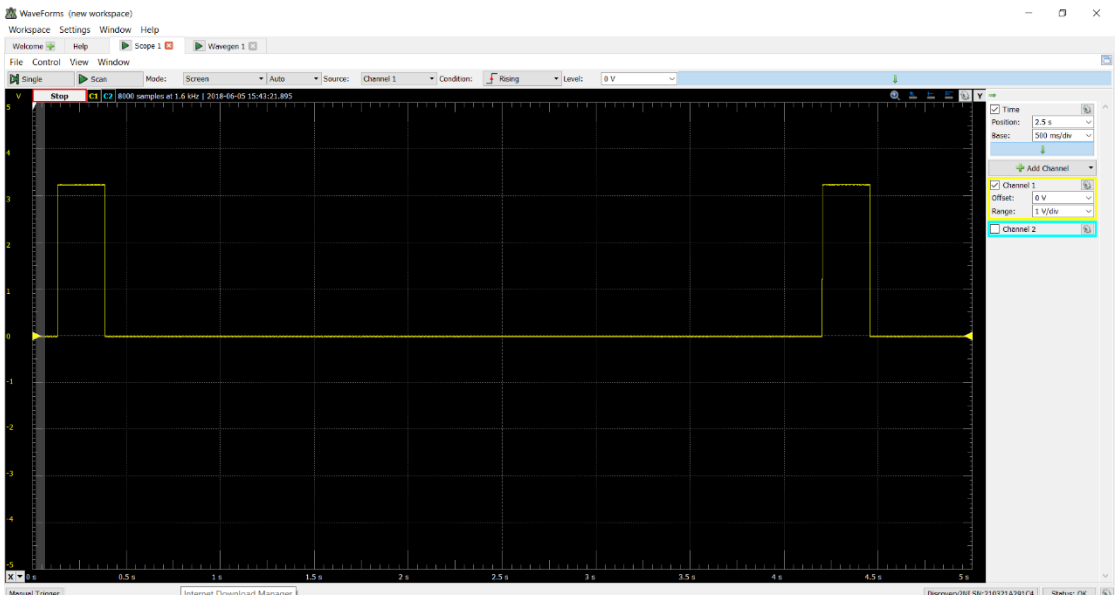

Figure 6.6 A closer look on the 2 rapid spikes fired at the beginning in the tonic spiking behavior.

# CHAPTER 7

## Economic analysis

### 7.1 The market share of Deep Neural Network

The huge amount of data generated each day is the main resource for the AI systems which supports the NN growth. Deep neural network market size in 2016 is estimated to be about 270 USD million as in [19]. In addition, the compound annual growth rate (CAGR) of neural network market is estimated to be 26% till 2021 [20]. The reason for such a huge investment is that there are many promising applications in which neural network can be applied. Figure 7.1 shows the market share of the deep neural network software in 2016 [20]. The healthcare category, for example, is in need for low power AI systems. These low power devices are implanted inside the human body.

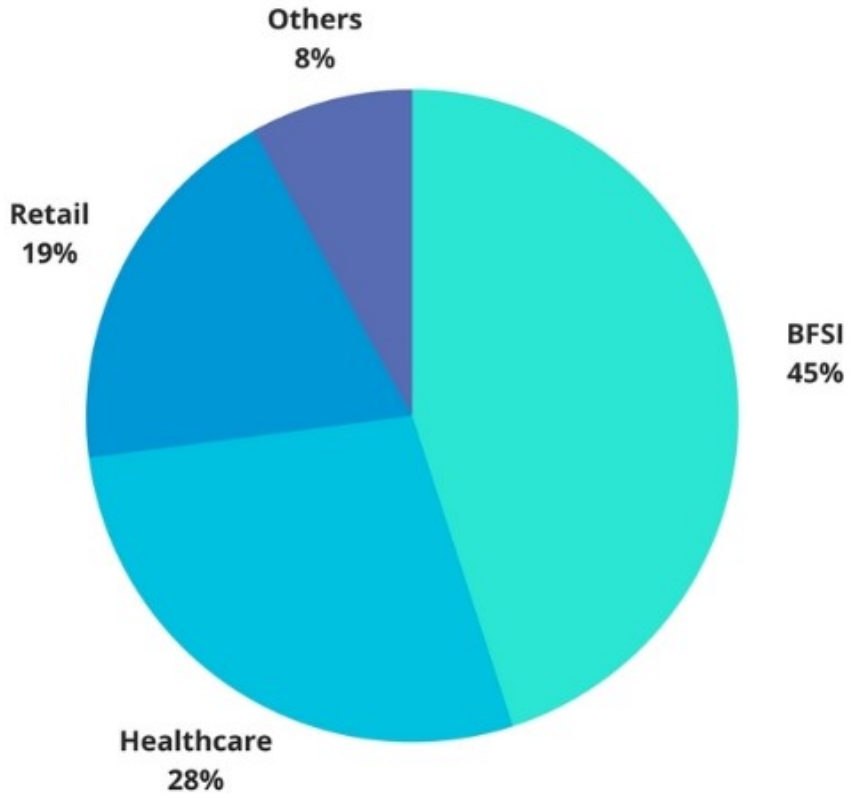


Figure 7.1 The market share of global neural network software in 2016.

## 7.2 The Investment Growth of Hardware-based Deep Learning Applications

As shown in Figure 7.2, the services provided by deep learning technology are limited due to lack of well-defined learning algorithms during the period from 2014 to 2017. Unfortunately, computer engineers haven't covered all concepts of neural networks, yet. Till now, there are still new learning algorithms introduced to deep learning. On the other hand, there is a solid base of deep learning systems that can be used in applications now and provide great results. This figure illustrates the growth of deep learning hardware applications. Starting from 2016, the demands of converting the software systems to real time hardware systems is increased. The needs of hardware deep learning are due to its low power with efficient performance. Nowadays, the market demands the lowest possible power systems that can be reached. So, hardware deep learning is expected to have a high growth rate in the upcoming years [19].

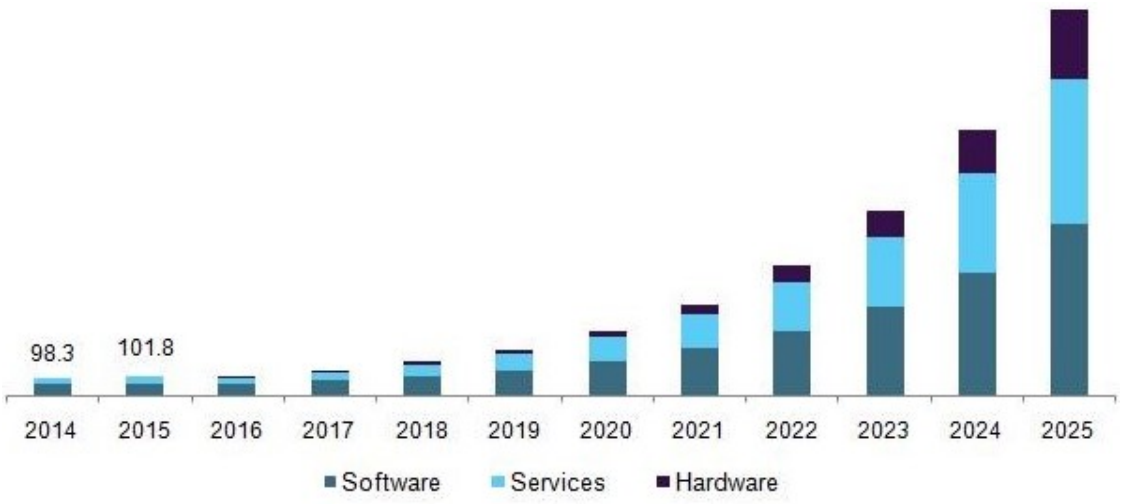


Figure 7.2 The investment growth of Hardware-based Deep Learning Applications from 2014 to 2025 in USD million.

## 7.3 The estimated fabrication cost of hardware implementation in ASIC and FPGA platforms

Table 7.1 represents the estimated fabrication cost of implementing our SNN on ASIC and FPGA platforms. The FPGA estimated cost is based on the Xilinx Zynq-7000 SoC ZC702 Evaluation Kit which is suitable for the number of LUTs used in our design. Fabrication cost in ASIC platform is determined from Sigenics website. As shown in Table 7.1 , it is clear that the cost of implementing a prototype using FPGA is much less than implementing it with ASIC platform due to the Non-recurring engineering (NRE) cost of ASIC. Table 7.2 shows the estimated fabrication cost of each chip in a production line of 1,000 chips. The fabrication cost using ASIC platform decreases with increasing the production capacity as the NRE cost is divided on a large number of chips. In contrast, the fabrication using FPGA platform remains the same [21].

Table 7.1 The estimated fabrication cost for the prototype in ASIC and FPGA platforms.

| Implementation Platform | SNN Fabrication Cost |
|---|---|
| FPGA<br>(Xilinx Zynq-7000 SoC ZC702 Evaluation Kit) | 895$ |
| ASIC | NRE cost = 234,964$<br>Production die cost = 5,300$ |

Table 7.2 The estimated fabrication cost for each individual SNN in the production line of 1,000 chips.

| Implementation platform | SNN Fabrication Cost per chip |
|---|---|
| FPGA<br>(Xilinx Zynq-7000 SoC ZC702 Evaluation Kit) | 895$ |
| ASIC | 240$ |

# CHAPTER 8

## Conclusion

In this work, several approximation methods such as the CORDIC, the Iterative logarithmic and the integral sum approximations for the $v^2$ term in the Izhikevich model are proposed. The CORDIC-based Izhikevich neuron model produces the spiking neuron behaviors like the original model efficiently. The original, the CORDIC-based and the PWL-based Izhikevich models are tested using both the ASIC & FPGA platforms to perform power/area comparison. The CORDIC-based Izhikevich model exhibits less power and area compared to the original Izhikevich model with an acceptable error. In addition, a Figure of Merit among Power, Area and error is defined to show how the CORDIC algorithm is flexible where the CORDIC approximation achieves better results than the PWL approximation.

Furthermore, the performance of each approximation in neural networks has been tested by constructing the network using the CORDIC-based neuron, PWL-based neuron and the original neuron. The CORDIC-based spiking neural network consumes less power and area than the original and the PWL-based spiking neural networks. Also, the CORDIC-based spiking neural network has been found to be more accurate than the PWL-based one.

Finally, real time FPGA outputs is demonstrated using an oscilloscope for the purpose of behavioural verification. It is found that the CORDIC-based Izhikevich neuron can exhibit the tonic spiking behavior correctly.

# REFERENCES


[1] H. Soleimani, A. Ahmadi, and M. Bavandpour, "Biologically inspired spiking neurons: Piecewise linear models and digital implementation," *IEEE Trans. Circuits Syst. I Regul. Pap.*, 2012.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, 1998.

[3] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Third edition*. 2014.

[4] H. Bauer, G. Richter, J. Wüllenweber, M. Breunig, D. Wee, and H. Klein, "Smartening up with Artificial Intelligence (AI) - What's in it for Germany and its Industrial Sector?," 2017.

[5] K. Gurney, *An Introduction to Neural Networks*. 1996.

[6] R. T. J. Monson and Drew J. Philip, "Artificial neural networks," *Surgery*, 2000.

[7] M. Nielsen, "Neural Networks and Deep Learning," *Determ. Press*, 2015.

[8] A. Deshpande, "A Beginner's Guide To Understanding Convolutional Neural Networks Part 1," *https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/*, 2016. .

[9] S. Davies, "Learning in Spiking Neural Networks," *Thesis Diss.*, 2012.

[10] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Networks*, 2003.

[11] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE Trans. Neural Networks*, 2004.

[12] K. Doya, "What are the computations of the cerebellum, the basal ganglia and the cerebral cortex?," *Neural Networks*. 1999.

[13] J. Wang, A. Belatreche, L. Maguire, and M. McGinnity, "Online Versus Offline Learning for Spiking Neural Networks: A Review and New Strategies," *IEEE 9th Int. Conf. Cyberntic Intell. Syst.*, 2010.

[14] M. Heidarpour, A. Ahmadi, and R. Rashidzadeh, "A CORDIC Based Digital Hardware for Adaptive Exponential Integrate and Fire Neuron," *IEEE Trans. Circuits Syst. I Regul.*

*Pap.*, 2016.

[15] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocess. Microsyst.*, 2011.

[16] R. C. Guido, "A tutorial on signal energy and its applications," *Neurocomputing*, 2016.

[17] C. Inacio and D. Ombres, "DSP decision: fixed point of floating?," *IEEE Spectr.*, 1996.

[18] R. Brette, "Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain," *Front. Syst. Neurosci.*, 2015.

[19] "'Deep Learning Market Size & Growth | Industry Research Report, 2025,' Agriculture Drones Market Size, Share | Industry Report, 2024." [Online]. Available: https://www.grandviewresearch.com/industry-analysis/deep-learning-market. [Accessed: 05-Jun-2018].

[20] "Global Neural Network Software Market 2017-2021," Technavio," 2017. [Online]. Available: https://www.technavio.com/report/global-neural-network-software-market. [Accessed: 05-Jun-2018].

[21] "PeopleVine, S. (2018). Sigenics - Custom Integrated Circuits." [Online]. Available: http://www.sigenics.com/. [Accessed: 05-Jun-2018].

# APPENDIX
# MATLAB SOURCE CODE

```
%   This MATLAB file generates figure 1 in the paper by
%               Izhikevich E.M. (2004)
%   Which Model to Use For Cortical Spiking Neurons?
%   use MATLAB R13 or later. November 2003. San Diego, CA

%%%%%%%%%%%%%%% (A) tonic spiking %%%%%%%%%%%%%%%%%%%%
subplot(5,4,1)
a=0.02; b=0.2;  c=-65;  d=6;
V=-70;  u=b*V;
VV=[];  uu=[];
tau = 0.25; tspan = 0:tau:100;
T1=tspan(end)/10;
for t=tspan
    if (t>T1)
        I=14;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
end;
plot(tspan,VV,[0 T1 T1 max(tspan)],-90+[0 0 10 10]);
axis([0 max(tspan) -90 30])
axis off;
title('(A) tonic spiking');

%%%%%%%%%%%%%%%% (B) phasic spiking %%%%%%%%%%%%%%%%%%%%%
subplot(5,4,2)%
a=0.02; b=0.25; c=-65;  d=6;
V=-64; u=b*V;
VV=[];  uu=[];
tau = 0.25;tspan = 0:tau:200;
T1=20;
for t=tspan
    if (t>T1)
        I=0.5;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
```

```
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 max(tspan)],-90+[0 0 10 10]);
axis([0 max(tspan) -90 30])
axis off;
title('(B) phasic spiking');

%%%%%%%%%%%%%% (C) tonic bursting %%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,3)
a=0.02; b=0.2;  c=-50;  d=2;
v=-70;  u=b*v;
vv=[];  uu=[];
tau = 0.25; tspan = 0:tau:220;
T1=22;
for t=tspan
    if (t>T1)
        I=15;
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 max(tspan)],-90+[0 0 10 10]);
axis([0 max(tspan) -90 30])
axis off;
title('(C) tonic bursting');

%%%%%%%%%%%%%% (D) phasic bursting %%%%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,4)
a=0.02; b=0.25; c=-55;  d=0.05;
v=-64;  u=b*v;
vv=[];  uu=[];
tau = 0.2;  tspan = 0:tau:200;
T1=20;
for t=tspan
    if (t>T1)
        I=0.6;
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
        vv(end+1)=30;
        v = c;
```

```
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 max(tspan)],-90+[0 0 10 10]);
axis([0 max(tspan) -90 30])
axis off;
title('(D) phasic bursting');


%%%%%%%%%%%%%%% (E) mixed mode %%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,5)
a=0.02; b=0.2;  c=-55;  d=4;
v=-70;  u=b*v;
vv=[];  uu=[];
tau = 0.25; tspan = 0:tau:160;
T1=tspan(end)/10;
for t=tspan
    if (t>T1)
        I=10;
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 max(tspan)],-90+[0 0 10 10]);
axis([0 max(tspan) -90 30])
axis off;
title('(E) mixed mode');


%%%%%%%%%%%%%%%% (F) spike freq. adapt %%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,6)
a=0.01; b=0.2;  c=-65;  d=8;
v=-70;  u=b*v;
vv=[];  uu=[];
tau = 0.25; tspan = 0:tau:85;
T1=tspan(end)/10;
for t=tspan
    if (t>T1)
        I=30;
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
```

```
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 max(tspan)],-90+[0 0 10 10]);
axis([0 max(tspan) -90 30])
axis off;
title('(F) spike freq. adapt');

%%%%%%%%%%%%%%%% (G) Class 1 exc. %%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,7)
a=0.02; b=-0.1; c=-55; d=6;
v=-60; u=b*v;
vv=[]; uu=[];
tau = 0.25; tspan = 0:tau:300;
T1=30;
for t=tspan
    if (t>T1)
        I=(0.075*(t-T1));
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+4.1*v+108-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 max(tspan) max(tspan)],-90+[0 0 20 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(G) Class 1 excitable');

%%%%%%%%%%%%%%%% (H) Class 2 exc. %%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,8)
a=0.2;  b=0.26; c=-65;  d=0;
v=-64;  u=b*v;
vv=[];  uu=[];
tau = 0.25; tspan = 0:tau:300;
T1=30;
for t=tspan
    if (t>T1)
        I=-0.5+(0.015*(t-T1));
    else
        I=-0.5;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
```

```
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 max(tspan) max(tspan)],-90+[0 0 20 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(H) Class 2 excitable');

%%%%%%%%%%%%%%%% (I) spike latency %%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,9)
a=0.02; b=0.2;  c=-65;  d=6;
v=-70;  u=b*v;
vv=[];  uu=[];
tau = 0.2; tspan = 0:tau:100;
T1=tspan(end)/10;
for t=tspan
    if t>T1 & t < T1+3
        I=7.04;
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 T1+3 T1+3 max(tspan)],-90+[0 0 10 10 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(I) spike latency');


%%%%%%%%%%%%%%%% (J) subthresh. osc. %%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,10)
a=0.05; b=0.26; c=-60;  d=0;
v=-62;  u=b*v;
vv=[];  uu=[];
tau = 0.25; tspan = 0:tau:200;
T1=tspan(end)/10;
for t=tspan
    if (t>T1) & (t < T1+5)
        I=2;
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
```

```
    if v > 30
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 (T1+5) (T1+5) max(tspan)],-90+[0 0 10 10 0 0],...
      tspan(220:end),-10+20*(vv(220:end)-mean(vv)));
axis([0 max(tspan) -90 30])
axis off;
title('(J) subthreshold osc.');


%%%%%%%%%%%%%%%%% (K) resonator %%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,11)
a=0.1;  b=0.26; c=-60;  d=-1;
v=-62;  u=b*v;
vv=[];  uu=[];
tau = 0.25; tspan = 0:tau:400;
T1=tspan(end)/10;
T2=T1+20;
T3 = 0.7*tspan(end);
T4 = T3+40;
for t=tspan
    if ((t>T1) & (t < T1+4)) | ((t>T2) & (t < T2+4)) | ((t>T3) & (t < T3+4)) | ((t>T4)
& (t < T4+4))
        I=0.65;
    else
        I=0;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 T1 T1 (T1+8) (T1+8) T2 T2 (T2+8) (T2+8) T3 T3 (T3+8) (T3+8) T4 T4
(T4+8) (T4+8) max(tspan)],-90+[0 0 10 10 0 0 10 10 0 0 10 10 0 0 10 10 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(K) resonator');

%%%%%%%%%%%%%%%%% (L) integrator %%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,12)
a=0.02; b=-0.1; c=-55; d=6;
v=-60; u=b*v;
vv=[];  uu=[];
tau = 0.25; tspan = 0:tau:100;
T1=tspan(end)/11;
T2=T1+5;
```

```
T3 = 0.7*tspan(end);
T4 = T3+10;
for t=tspan
    if ((t>T1) & (t < T1+2)) | ((t>T2) & (t < T2+2)) | ((t>T3) & (t < T3+2)) | ((t>T4)
& (t < T4+2))
        I=9;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+4.1*V+108-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
end;
plot(tspan,VV,[0 T1 T1 (T1+2) (T1+2) T2 T2 (T2+2) (T2+2) T3 T3 (T3+2) (T3+2) T4 T4
(T4+2) (T4+2) max(tspan)],-90+[0 0 10 10 0 0 10 10 0 0 10 10 0 0 10 10 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(L) integrator');

%%%%%%%%%%%%%%%% (M) rebound spike %%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,13)
a=0.03; b=0.25; c=-60;  d=4;
V=-64;  u=b*V;
VV=[];  uu=[];
tau = 0.2;  tspan = 0:tau:200;
T1=20;
for t=tspan
    if (t>T1) & (t < T1+5)
        I=-15;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
end;
plot(tspan,VV,[0 T1 T1 (T1+5) (T1+5) max(tspan)],-85+[0 0 -5 -5 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(M) rebound spike');

%%%%%%%%%%%%%%%% (N) rebound burst %%%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,14)
a=0.03; b=0.25; c=-52;  d=0;
```

```
V=-64;  u=b*V;
VV=[];  uu=[];
tau = 0.2;  tspan = 0:tau:200;
T1=20;
for t=tspan
    if (t>T1) & (t < T1+5)
        I=-15;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
end;
plot(tspan,VV,[0 T1 T1 (T1+5) (T1+5) max(tspan)],-85+[0 0 -5 -5 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(N) rebound burst');

%%%%%%%%%%%%%%%% (O) thresh. variability %%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,15)
a=0.03; b=0.25; c=-60;  d=4;
V=-64;  u=b*V;
VV=[];  uu=[];
tau = 0.25; tspan = 0:tau:100;
for t=tspan
   if ((t>10) & (t < 15)) | ((t>80) & (t < 85))
        I=1;
    elseif (t>70) & (t < 75)
        I=-6;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
end;
plot(tspan,VV,[0 10 10 15 15 70 70 75 75 80 80 85 85 max(tspan)],...
          -85+[0 0  5  5  0  0  -5 -5 0  0  5  5  0  0]);
axis([0 max(tspan) -90 30])
axis off;
title('(O) thresh. variability');
```

```
%%%%%%%%%%%%%% (P) bistability %%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,16)
a=0.1;  b=0.26; c=-60;  d=0;
V=-61;  u=b*V;
VV=[];  uu=[];
tau = 0.25; tspan = 0:tau:300;
T1=tspan(end)/8;
T2 = 216;
for t=tspan
    if ((t>T1) & (t < T1+5)) | ((t>T2) & (t < T2+5))
        I=1.24;
    else
        I=0.24;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
end;
plot(tspan,VV,[0 T1 T1 (T1+5) (T1+5) T2 T2 (T2+5) (T2+5) max(tspan)],-90+[0 0 10 10 0 0
10 10 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(P) bistability');


%%%%%%%%%%%%%% (Q) DAP %%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,17)
a=1;  b=0.2; c=-60;  d=-21;
V=-70;  u=b*V;
VV=[];  uu=[];
tau = 0.1; tspan = 0:tau:50;
T1 = 10;
for t=tspan
     if abs(t-T1)<1
        I=20;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
end;
plot(tspan,VV,[0 T1-1 T1-1 T1+1 T1+1 max(tspan)],-90+[0 0 10 10 0 0]);
axis([0 max(tspan) -90 30])
```

```
axis off;
title('(Q) DAP         ');



%%%%%%%%%%%%%% (R) accomodation %%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,18)
a=0.02;  b=1; c=-55;  d=4;
V=-65;  u=-16;
VV=[];  uu=[];  II=[];
tau = 0.5; tspan = 0:tau:400;
for t=tspan
    if (t < 200)
        I=t/25;
    elseif t < 300
        I=0;
    elseif t < 312.5
        I=(t-300)/12.5*4;
    else
        I=0;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*(V+65));
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
    end;
    uu(end+1)=u;
    II(end+1)=I;
end;
plot(tspan,VV,tspan,II*1.5-90);
axis([0 max(tspan) -90 30])
axis off;
title('(R) accomodation');

%%%%%%%%%%%%%% (S) inhibition induced spiking %%%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,19)
a=-0.02;  b=-1; c=-60;  d=8;
V=-63.8;  u=b*V;
VV=[];  uu=[];
tau = 0.5; tspan = 0:tau:350;
for t=tspan
       if (t < 50) | (t>250)
        I=80;
    else
        I=75;
    end;
    V = V + tau*(0.04*V^2+5*V+140-u+I);
    u = u + tau*a*(b*V-u);
    if V > 30
        VV(end+1)=30;
        V = c;
        u = u + d;
    else
        VV(end+1)=V;
```

```
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 50 50 250 250 max(tspan)],-80+[0 0 -10 -10 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(S) inh. induced sp.');

%%%%%%%%%%%%% (T) inhibition induced bursting %%%%%%%%%%%%%%%%%%%%%%%%
subplot(5,4,20)
a=-0.026;  b=-1; c=-45;  d=-2;
v=-63.8;  u=b*v;
vv=[];  uu=[];
tau = 0.5; tspan = 0:tau:350;
for t=tspan
       if (t < 50) | (t>250)
        I=80;
    else
        I=75;
    end;
    v = v + tau*(0.04*v^2+5*v+140-u+I);
    u = u + tau*a*(b*v-u);
    if v > 30
        vv(end+1)=30;
        v = c;
        u = u + d;
    else
        vv(end+1)=v;
    end;
    uu(end+1)=u;
end;
plot(tspan,vv,[0 50 50 250 250 max(tspan)],-80+[0 0 -10 -10 0 0]);
axis([0 max(tspan) -90 30])
axis off;
title('(T) inh. induced brst.');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set(gcf,'Units','normalized','Position',[0.3 0.1 0.6 0.8]);
```