SCALABLE PULSED MODE COMPUTATION ARCHITECTURE USING INTEGRATE AND FIRE STRUCTURE BASED ON MARGIN PROPAGATION

By

Thamira Hindo

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Electrical Engineering- Doctor of Philosophy

2014

ABSTRACT

SCALABLE PULSED MODE COMPUTATION ARCHITECTURE USING INTEGRATE AND FIRE STRUCTURE BASED ON MARGIN PROPAGATION

By

Thamira Hindo

Neuromorphic computing architectures mimic the brain to implement efficient computations for sensory applications in a different way from that of the traditional Von Neumann architecture. The goal of neuromorphic computing systems is to implement sensory devices and systems that operate as efficiently as their biological equivalents. Neuromorphic computing consists of several potential components including parallel processing instead of synchronous processing, hybrid (pulse) computation instead of digital computation, neuron models as a basic core of the processing instead of the arithmetic logic units, and analog VLSI design instead of digital VLSI design. In this work a new neuromorphic computing architecture is proposed and investigated for the implementation of algorithms based on using the pulsed mode with a neuron-based circuit.

The proposed architecture goal is to implement approximate non-linear functions that are important components of signal processing algorithms. Some of the most important signal processing algorithms are those that mimic biological systems such as hearing, sight and touch. The designed architecture is pulse mode and it maps the functions into an algorithm called margin propagation. The designed structure is a special network of integrate-and-fire neuron-based circuits that implement the margin propagation algorithm using integration and threshold operations embedded in the transfer function of the neuron model. The integrate-and-fire neuron units in the network are connected together through excitatory and inhibitory paths to impose constraints on the network firing-rate. The advantages of the pulse-based, integrate-and-fire margin propagation (IFMP) algorithmic unit are to implement complex non-linear and dynamic programming functions in a scalable way; to implement functions using cascaded design in parallel or serial architecture; to implement the modules in low power and small size circuits of analog VLSI; and to achieve a wide dynamic range since the input parameters of IFMP module are mapped in the logarithmic domain.

The newly proposed IFMP algorithmic unit is investigated both on a theoretically basis and an experimental performance basis. The IFMP algorithmic unit is implemented with a low power analog circuit. The circuit is simulated using computer aided design tools and it is fabricated in a 0.5 micron CMOS process. The hardware performance of the fabricated IFMP algorithmic architecture is also measured.

The application of the IFMP algorithmic architecture is investigate for three signal processing algorithms including sequence recognition, trace recognition using hidden Markov model and binary classification using a support vector machine. Additionally, the IFMP architecture is investigated for the application of the winner-take-all algorithm, which is important for hearing, sight and touch sensor systems.

To my husband Sabah Almasraf
and my daughters
Nora, Basma and Nagham

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Problem and key solution

Recognition and classification are the basic sensing applications in biological sensory organs. Extensive researches are implemented to achieve these applications using artificial neural network algorithms (ANN) and the technology used in man machine interface . Yet, the processing time and architectures of these applications are far beyond the "real time" processing and "scalable structures" as in biology. In order to gain real time processing, the algorithms executed in software using Von Neumann architecture have to be replaced by hardware implementation. In addition to that, the computational algorithms themselves have to be scalable to gain efficient hardware implementation. Real time processing and scalability are the goal of implementing signal processing on a chip in general and implementing neuromorphic systems as a specific implementation. The goal of the neuromorphic systems is to build a silicon chip that mimics the brain in order to implement sensory devices in an efficient way as in biology [1, 2, 3, 4, 5]. Therefore, the architecture of the new systems has to be similar to the neuronal architecture systems which is different from the traditional Von Neumann architecture such as asynchronous- parallel processing instead of synchronous-single processing, hybrid (pulse mode or analog and digital mode) computation instead of digital computation, neuron model as a basic core of the processing instead of the arithmetic logic unit and finally, analog VLSI design instead of digital VLSI. As a contribution in this large effort, a novel and scalable algorithm and its hardware implementation is proposed to compute non-linear functions as an important procedure to implement signal processing algorithms in the sensory applications. To meet the above objective, the proposed algorithm is designed in pulse mode that can map the functions into an algorithm called margin propagation using an integrate and fire structure. Figure 1.1 shows the three concepts of the proposed algorithm that is firstly based on pulse computation, secondly mapped into Margin Propagation as an approximation method to log-sum- exp and thirdly the proposed module is based on an integrate and fire structure as a core unit. This chapter introduces

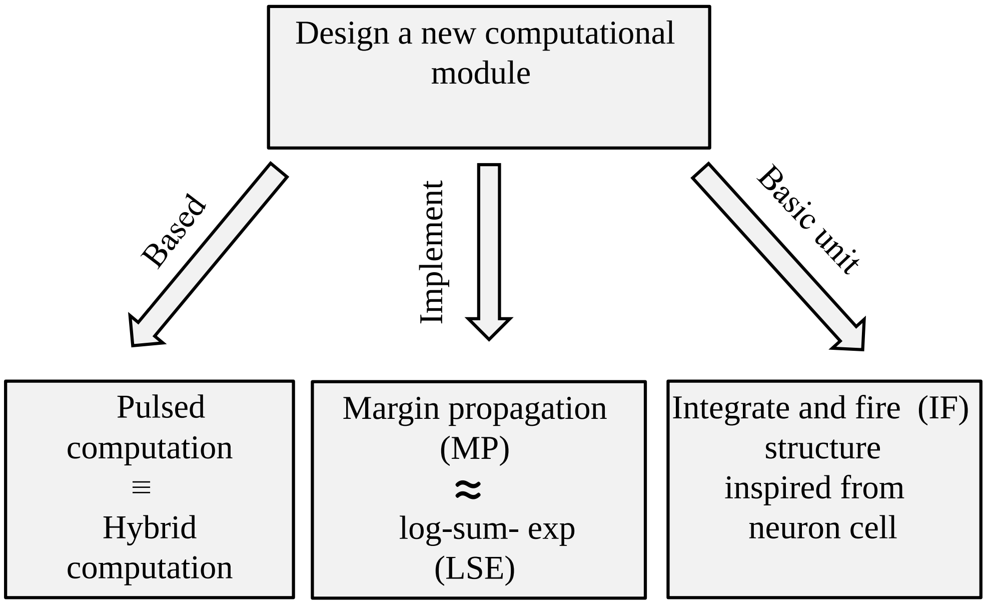

Design a new computational module
Based
Implement
Basic unit
Pulsed computation ≡ Hybrid computation
Margin propagation (MP) ≈ log-sum- exp (LSE)
Integrate and fire (IF) structure inspired from neuron cell

Figure 1.1: Main concepts of the proposed module

the following: Section 1.2 starts initially with the definition of pulse computation and how the variables are represented and the reasons behind using the pulse computation module. Then the section is followed by the background of the pulse computation architectures in the research domain and the objective of the proposal. The following two sections 1.3 and 1.4 include concepts that meet the objective of the proposal. Sections 1.3 includes the log-sum-exp function and margin propagation algorithm. Section 1.4 includes a brief description of the integrate and fire structure as the core unit of the proposed algorithm. Section 1.5 includes the tasks and methodology to verify the concept. Finally, section 1.6 states the organization of the consecutive chapters of the proposal.

1.2 Pulse computation

The pulse computation algorithm is introduced to mix the advantages of both digital and analog design. The proposed algorithm is mapped into hardware as digital in amplitude level and analog in the variable time between these levels. The main goal for using pulse computation is to propagate easily the computed parameters between successive modules because the type of the input/ output signals are digital. Before explaining the details of the computational module, it is important to define and describe with an example the importance of pulse computation systems, background of the pulse computations, the objective of the proposal and finally the significant issues that meet the objective.

1.2.1 What and why pulse computation

Pulse mode computation is an analog/digital (hybrid) computation which is digital(binary) in voltage level and analog (variable) quantity in time intervals between digital levels. In this mode, the data used is represented by a stream of pulses which are asynchronous and arrived at arbitrary times. Example of such data is the stream of pulses generated from neuron structure which is driven by a continuously varying current.

A pulse stream is represented as a stochastic stream of ones and zeros in which the probability of generating ones is considered as a pulse rate value. The pulse rate values are considered as the input /output variables of the pulse computation mode. In other words, the input/output variables are assigned as the probability of ones in a stream of random sequence of zeroes and ones as shown in the following mathematical procedure. If x is a vector of random inputs between 0 and 1 ($x \in u[0,1]$), y is the generated pulse stream, then the probability of generating a stream y of rate L_1 is calculated as,

$$y = 0.5[\ sgn(x - 1 + L_1) + 1] \Longrightarrow 2y - 1 = sgn(x - 1 + L_1)$$

$$p[\text{y} = 1] = p[sgn(x - 1 + L_1) = 1],\ \text{and}\ \ [\text{y} = 0] = p[sgn(x - 1 + L_1) = -1]$$

therefore the probability that the stream y has logic 1 is equal to the rate L_1 calculated as following,

$$p[\text{y} = 1] = p[x > (1 - L_1)] \Longrightarrow \int_{1-L_1}^{\infty} u(0,1)dx = \int_{1-L_1}^{1} dx = L_1$$

Pulse computation is a promising research topic since it mixes the advantages of analog and digital designs [6, 7]. The noise accumulation in analog stages can be eliminated by digital noise immunity. The analog design has the advantage of small area, low cost and low power especially if the design of computational units is implemented in weak inversion mode of complementary metal oxide semiconductors (CMOS).

Table 6.1 shows the main differences between the digital and analog systems. The first three items show the benefits of analog devices in computations while the last two items show the advantages of the digital systems.

Table 1.1: Comparison between analog and digital systems.

Analog	Digital
Efficient computations use continuous values (currents and voltages).	Computations uses discrete values (Logic 0 and 1).
Computations arise from physics of computing devices (capacitors, resistors..etc.).	Computations arise from the mathematics of Boolean logic(AND, OR..etc.).
Designs have high density, small size and low power (addition requires only wires, multiplication requires 7 transistors).	Designs have less density, larger size and higher power (Addition require 18 transistors and 2 bits multiplications requires 60 transistors).
Noise is due to thermal fluctuation in physical devices and it accumulates in the multi-stage designs.	Noise is due to round off error and it does not accumulate.
Signal is not restored at each stage of the computation.	Signal is restored at each stage of the computation.

1.2.2 Example of a pulse computational system

The brain is an example of hybrid computation because the type of signals transmitted between neurons are spikes (or pulses) and the processing of signals inside the neuron is analog. Next, a discussion of the power and efficiency of the brain in processing compared to computers.

Computers are no real competition for the human brain in areas such as vision, hearing, pattern recognition, and learning. Computers, for instance, cannot match our ability to recognize a friend from a distance merely by the way he walks. And when it comes to operational efficiency, there is no contest at all. The "100 trillion synapses and 100 billion neurons in the brain process information faster and with a higher density (data stored in the synapses) than the software simulation of the neural networks using mathematical models. For example: 2 duo core chips are needed to simulate only thirty neurons, a 32 chip board is needed to simulate 302 neurons, and a 64 board rack of super computers is needed to simulate 1 million neurons. For the super computer, the cost is in millions of dollars and the power bill costs thousands of dollars in order to simulate the brain of a bee (1 million neuron). A comparison of processing data between the brain and computers based on the technical parametric and functionality are listed below: [8].

1.2.2.1 Technical specifications

Energy cost: The computational power measured in Joules per operation in the brain is 10^8 times smaller than the microprocessor. To explain that, let's take an example showing the difference in energy between the brain and processor taking into consideration that each synaptic junction between neurons represents an instruction operation in the microprocessors [9, 10]. Given the followings information: If the neurons fire at an average of 10 Hz in a human brain that has about 10^{12} neurons, 10^{15} synapses and uses an approximate power of 10W. Assuming that the dominant energy consumer is synaptic activation and that a basic operation is a synaptic activation (synapse is the junction between a pre and post neuron synaptic). Then the energy cost of this activation is computated in Joules/Operation and compared with the cost per Multiply-Accumulate (MAC) of contemporary microprocessors, e.g. 100 W and 1 Hz for 1 G operation [10].

For human brain;

Energy cost = Energy/Number of synapses

Energy cost = $(10\ \text{W}) .(0.1\ \text{sec})/10^{15} = 10^{-15}$J/operation

For Multiply-Accumulate (MAC) in microprocessors;

Energy cost = Energy/Number of operations

Energy cost = $(100\ \text{W}) . (1\ \text{sec})/10^{9} = 10^{-7}$J/operation

Therefore, the energy - cost for the human (10^{-15} J/operation) is less than the energy cost in the microprocessor (10^{-7} J /operation) by an order of eight.

Power: The power consumption in the brain is much lower than the computer processor. The power consumption in the brain is 25 W, while in Pentium, the power needed is 130 W. The power needed in software simulation to mimic the function of brain in computation is about 10^{10} W [3].

Frequency: The spike frequency in the brain is between 1-100 Hz while the clock frequency in Pentium is in GHz.

Size: The size of brain is just 1 liter while in computers technology, a typical room-size supercomputer weighs roughly 1,000 times more, occupies 10,000 times more space than the cantaloupe-size lump of neural tissue that makes up the brain [3].

Cross connection: The synapse in neurons is equivalent to executing an instruction in the microprocessor. Synaptic activity is 10^{16} neural connection /sec [3]. It would take one million Intel Pentium processors to match that rate besides tens of megawatt (cost of time, power and energy) in software to process these information.

1.2.2.2 Functional features

The computer system is a powerful machine in calculation and processing data, yet, it is limited in the tasks which the brain is capable to perform. The attractive features which support such tasks in the brain have different functions from the processor as follows:

Parallel processing: The neurons work in parallel rather than sequentially, as in the processors. In other words there is no central processing unit to schedule the tasks serially. The access time of the Input/ output information is the same in both the silicon memory and the sensory/motor biology neural system which is measured in nano seconds. However, the parallel architecture of the brain enhance the processing of the information compared with computers. It is well known that computers are no real competition for the human brain in areas such as vision, hearing, pattern recognition, and learning.

Clock orientation: Neurons are self timed while computers are oriented systems that run on the global clock.

Memory: It is available during computation in neurons within synapse connections, while in computers the data is distant from computation.

Adaption and modification: In the brain, learning and adaptation are continuous while the processors are fixed as they are designed and fabricated.

Determination of signal state: The type of signal is stochastic (i.e. digital analog-digital) while being digitally deterministic in processors.

Tables 1.2 and 1.3 show the technical specifications and functional differences between the brain and computer.

The above specifications are motivation in designing new architecture in the neuromorphic systems in which the design principles and architecture are neuronal inspired. Scientists in the area of Information Theories, Computer Science, Communication, Mathematics, Biology and Engineering are all working together to mimic the human brain on a silicon chip. The primary reason for implementing neural models is attempting to gain possible brain like performance. It is well known that the type of signal in the brain is spike or pulse based. Therefore, processing the information is needed to exploit methods in pulse mode processing.

Pulse system research is concentrated in two directions, first the development of computational algorithms to encoding information [11, 12, 13, 14, 15]; and second to morph

Table 1.2: Comparison of parametric specification between the brain and the computer for simulating human cortex. The computational energy/cost in brain is 10^8 times smaller (efficient) than computers.

Specification	Computer simulation of human cortex	Brain
Power	10^{10} W consumed in running software simulation in addition to the power needed in the system hardware.	25 W.
Size	A typical room-size supercomputer weighs roughly 1,000 times more, occupies 10,000 times more space.	1liter.
Energy cost	10^{-7}J/operation,for $1G$ operation in $1s$ and power =100 W.	10^{-15} J /operation, for a fire rate =10 Hz, power $\approx$ 10 W, and a total number of synapses = 10^{15}.

the above algorithms on silicon elements to encode the input stimulus and then process the spikes to mimic the biological sensory systems [16, 4, 3]. This thesis mixes between the two research directions. It introduces a novel pulse based computation algorithm and verifies the algorithm on the circuit level. But, what are the pulse computation algorithms in the research field and what is the contribution of the proposed thesis? The answers for these questions are explored in the next section.

1.2.3 Survey of pulse computation

The goal of pulse computation algorithm is to introduce efficient hardware architecture to implement signal processing tasks such as recognition and classification. The efficiency means

Table 1.3: Comparison of functional specification between the brain and the computer for simulating human cortex.

Computer	Brain
It is a global clock oriented system, in GHz.	It is a self timed system, 1-100 Hz.
It has a 1- 8 core processors and it is based on serial computation.	It has distributed computational units and it is based on parallel computation.
The buses share several components.	There are dedicated local point to point connections.
The signal is digital and it is a time-discrete.	The signal is digital in levels (spikes) and it is analog in time.
The memory is distant from computation.	The memory is available at computation location.
The memory and processor are separated.	The storage and computation happen at the same time and in the same place.
Devices are fixed as fabricated. They are programmed devices.	It continuously adapts, modifies and processes information.
It is sensitive to noise.	Noise is a beneficial parameter [5].

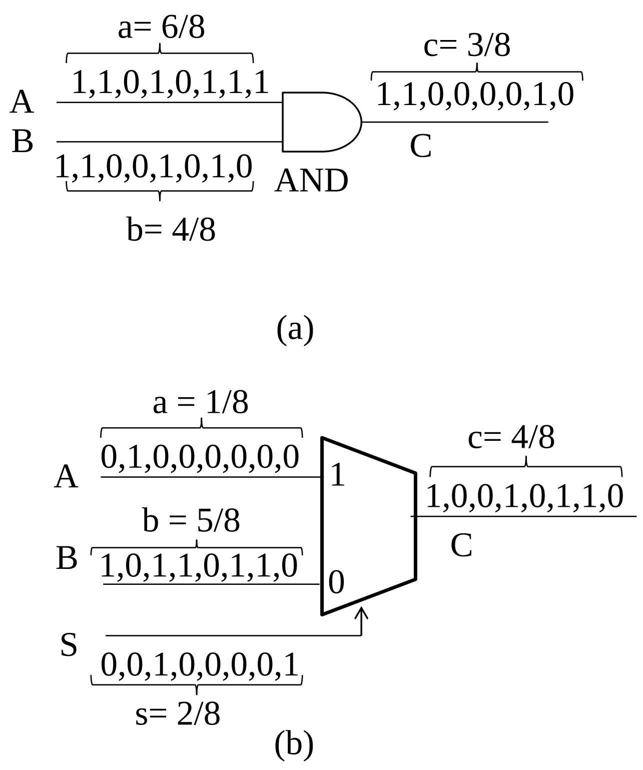
low power, compact and scalable architecture. The architecture of pulse computational algorithms are introduced in different categories:

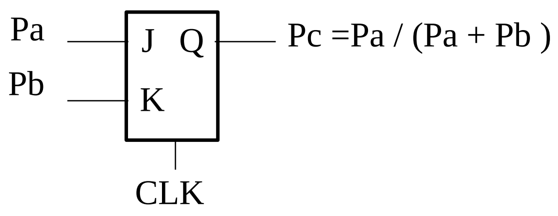
- Pulse computation is based on digital gates using stochastic computations.
- Pulse computation is based on analog circuits using finite state machine.
- Pulse computation is based on analog circuits using neural integration.

1.2.3.1 Pulse computation based on stochastic computations

: Stochastic computation (SC) was introduced in the 1960's as a method to design low precision digital circuits. The essential aspect of the SC is the possibility of performing complex computations using only simple circuitry. In SC, the input variables are represented as a probability of logic one in a stream of inputs of zeros and ones. The multiplication, scaled addition and division can be implemented using "AND", "Multiplexers" and "JK-flip flops" respectively as shown in Fig.1.2. For the AND gate shown in Fig. 1.2-(a) for example , if the input rates are 6/8 and 4/8, then the output rate is $6/8 \times 4/8 = 3/8$, as expected for the multiplication operation. Another example for the multiplexer shown in 1.2-(b), if the input rates are 1/8, 5/8, and 2/8, then the output rate is $2/8 \times 1/8 + (1 - 2/8) \times 5/8 = 4/8$, as expected for the scaled addition.

Experiments on polynomial functions used in image processing show that SC method produces circuits that are highly tolerant of input errors. The accuracy degrades gracefully with the error rate [17, 18, 19, 20]. SC has recently shown to be able to provide near optimal decoding performance for decoding with respect to sum product algorithms with small coding and specific applications, but it fails to build huge decoding systems due to SC weakness in computational precision and dynamic range. SC fails for some of the computation process, for example, the multiplication of two independent streams (p) such as $p^2 = p \times p$ using logical bitwise AND operation lead to misleading computation. In addition to that, SC fails because the parameters used in computations should be scaled up or down depending on the math operator, for example, the addition cannot be performed directly since the addition of two stream rates might be greater than one, therefore scaled addition must be used to implement the addition operator.





J	K	Q
1	0	1
0	1	0
0	0	No change
1	1	Reverse

(c)

Figure 1.2: Stochastic computation (a): Multiplication stochastic bit streams with an AND gate. (b): Scaled addition on stochastic bit streams, with a multiplexer (MUX). (c): Stochastic division [17, 21]

Another limitation of the SC is the low dynamic range of the computation in the sum -product algorithms. In such algorithms, the multiplication of two probabilities (represented as two bit streams) tends into a smaller quantity and hence multiple multiplications tend to much smaller value which results into an under flow problem in computation. In SC, the normalization step could be implemented using JK flip flops as division operator and multiplexer as scaled addition to realize the data in computations. But this process tends to be a difficult problem as the design size of the applications is huge (for example, in the case of hundred states, the implementation needs hundred JK flip flops and hundred to one multiplexer). In addition to that, the huge number of states and the number of input stream need a storage media such as registers. This will definitely increases the size of the design. Therefore, the low dynamic range of computations, the under flow problem and the normalization difficulties lead into false computation, low precision and long time to reach convergence (or never reach the convergence stage) in factor graph computations.

1.2.3.2 Pulse computation based on finite state machine

: A new technique was proposed to introduce pulse computation using finite state machine. This computational domain may hold the possibility of discovering silicon analogues to the enormously complex computational operations. In [22], elementary computational units named as "Macher" and "Differencer" are implemented in hardware using CMOS $2\mu m$. These units are the basic units to form more complex processing structures.

The foundation of the pulse computation using state machine is the operation of pulse matching and difference. Fig.1.3 (a) illustrates matched and unmatched pulses, given two signal lines $P1$ and $P2$, then the signals are matched by pairing a pulse on $P1$ with a pulse on $P2$. A pulse on $P1$ is unmatched, by contrast, if another pulse arrives on it and there are no pulses on $P2$ during the successive time period. Fig. 1.3 (b) shows the state transition diagram of the "Matcher" where the matched input pulses produce pulse outputs on line $O2$. if the machine is in $P1$ mode (state $S1$) and receives an input pulse on $P1$, no output is produced; but, if the pulse arrives instead on $P2$, an output pulse is produced on $O2$ and the machine transitions to $P2$ mode (state $S2$).

Fig. 1.3(c) shows the state machine of the "Differencer". If the machine is in '$P1$ mode' (state Sl), it will transition to '$P2$-mode' (state $S2$) if a pulse received on input $P2$, and remain in $S1$ either if nothing happens, or if a pulse is received on input $P1$. Output (a

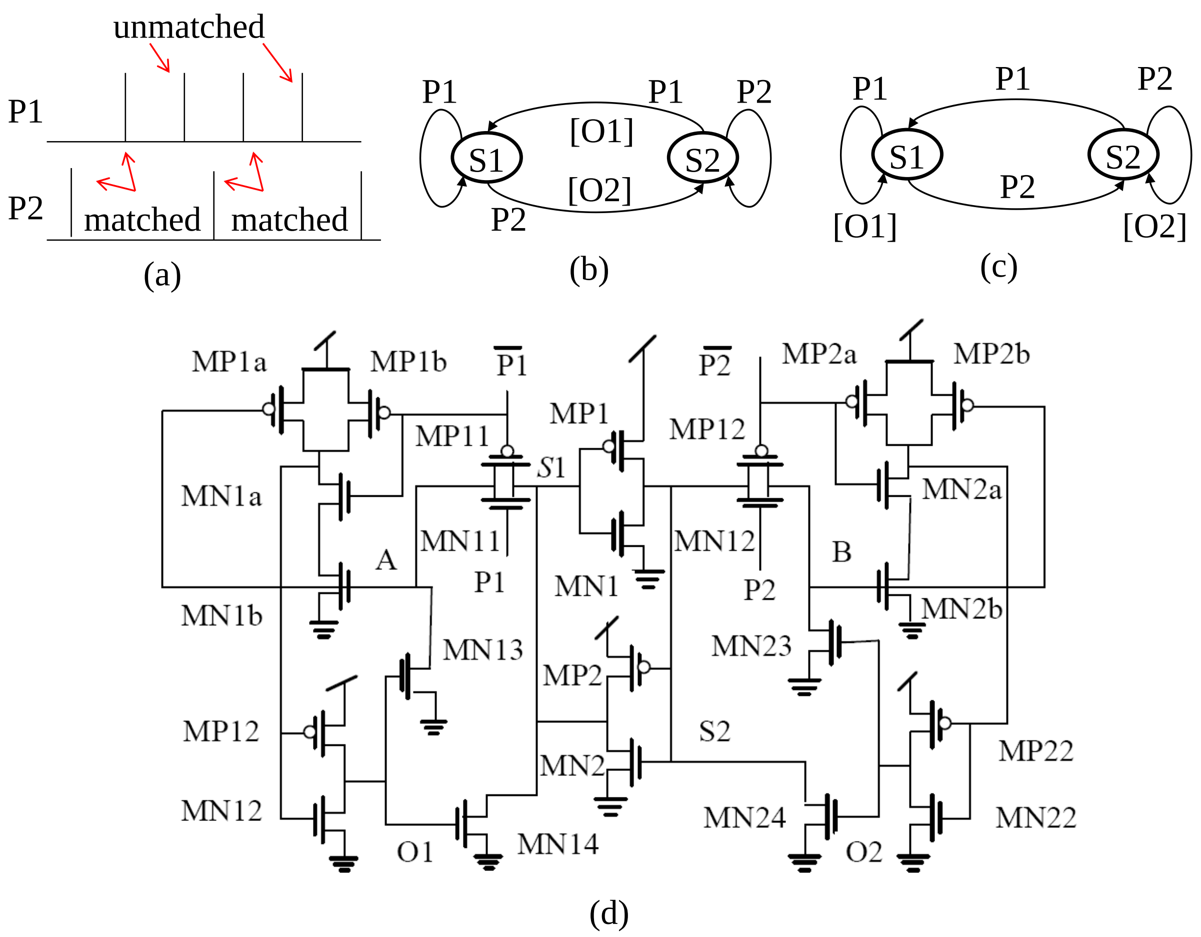


Figure 1.3: Pulse computation using state machines. (a): matched and unmatched pulses. (b): simplified state diagram for the matcher. (c): simplified state diagram for the differencer. (d): matched circuit [22]. (For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation)

pulse on line $O1$ or $O2$) is shown as $[O1]$ or $[O2]$; so if the machine is in $S1$ and receives a pulse on $P1$, it will output a pulse on $O1$. Therefore, this device outputs pulses only when unmatched input pulses are received.

The analog circuit and the oscilloscope traces from matcher is shown in Fig. 1.3(d) and Fig. 1.4(a) respectively. The circuit of the matcher is an SRAM cell constructed from cross-coupled inverters (transistors $MP1$, $MP2$, $MN1$, $MN2$) to hold the state. States (Sl) and ($S2$) are marked on the diagram; ($S2$) high means the circuit is in $P2$-mode. In $P2$-mode, a pulse arriving on $P1$ first opens, then closes the left-side pass-gate ($MP11$, $MN11$). This charges the dynamic node A high. When $P1$ falls low again (at the end of the incoming pulse,) the output of the NAND gate formed by ($MP1a, MP1b, MN1a, MN1b$) then drops low; this signal is inverted by ($MP12$, $MN12$) onto $O1$. An output high on $O1$ discharges the dynamic node through $MN13$, causing the system, including $O1$, to reset. The result is a brief voltage pulse, $[O1]$, which also (through $MN14$) sets the SRAM to state $S1$. If the incoming pulse had instead been on $P2$, the right-side pass-gate ($MP21, MN21$) would have opened; but since the dynamic node B would not have been raised high no output nor change of state would have been produced.

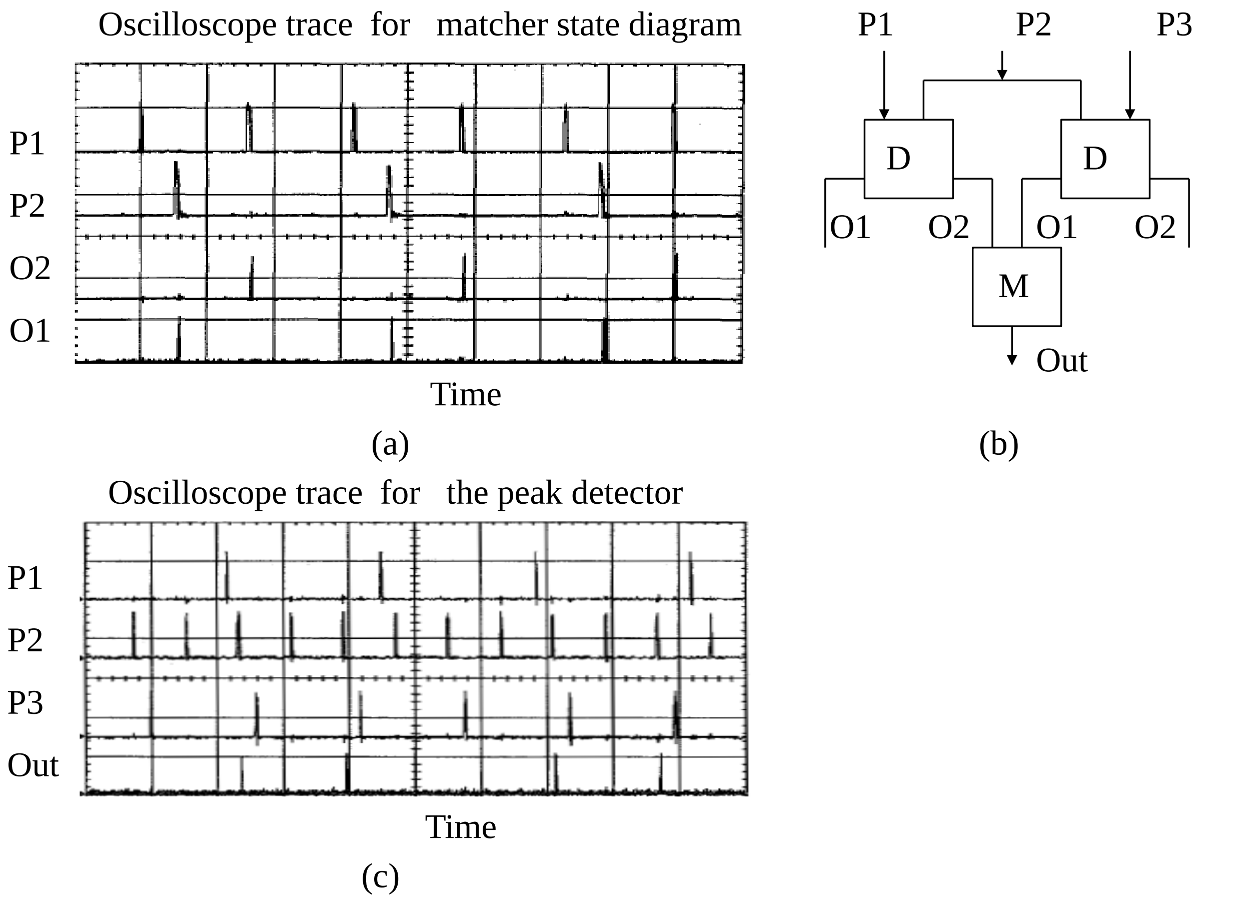


Figure 1.4: (a): Oscilloscope traces from matcher. P1 and P2, inputs; O2 and O1 outputs of states S1 and S2 for the matcher state diagram. (b): Block diagram of the peak detector application. (b) Oscilloscope traces from shows peak detection, P1, P2, P3,are the inputs; P4 is the output state diagram [22]

The basic elementary circuits are used to built a more complex application, for example, peak detector is designed using mixed units of differencer and matcher shown in the block diagram of Fig. 1.4 (b). The design considers the conversion of the signal amplitude into frequency and uses the concept of edge detection to find the highest frequency or the highest amplitude signal. In the figure, the outputs of the two differencers are the edge detectors between inputs $P1, P2$ and between $P2, P3$ respectively. The matcher generates an output pulse, with an average frequency which reflects the strength of the peak as shown in Fig. 1.4 (c) where the first three traces are the inputs and the last trace is the output. The occurrence of the output pulses indicates the input trace with the highest frequency. Although the above method is attractive in accuracy and programmability but the limitation here is in mapping the mathematical expressions needed in the applications of the signal processing algorithms. Also, the above computation unit is not scalable which means that any new application to design in hardware requires a new architecture of state machine and new analog circuits.

1.2.3.3 Pulse computation based on neural integration

Pulse stream computation was first introduced in the context of neural integration by Murry in 1987 [23]. Since then it has been used by a number of other groups [24, 25, 26, 27]. Before introducing the concept of neural pulse computation, the neuron itself is explored first as a pulse computation unit then an explanation of how to address the pulse computation among the digital and analog computation methods.

Pulse computation in biological neuron was first noticed, modeled and verified in experiments made by Hodgkin and Huxly in 1952 [28, 29, 30]. It is well known that the neurons communicate with each other through pulses. The pulses in the axon release neurotransmitters from the pre to the post synapse to generate pulses in the post neuron. As the neurotransmitters increase, the rate of the output pulses increase. The increment of neurotransmitters in biological experiments is represented by increasing the value of the injected current in the neuron membrane. It was proven in biological experiments on a giant squid axon and compared with a simulated neuron model that the transfer function of a neuron between the activation input current and the output spike rate is close to the logarithm function [28]. That means the addition of two input rates in one neuron represent the multiplication of the two rates [31, 29]. The fact that neuron multiplication of the input pulses realize specific tasks, for example, a motion detection task based on the correlation principle that multiply the input signal with the delayed version of the same or another input signal [30]. On the other hand, the decision made by one neuron is a function of the weighted sum of all the inputs in the neural population. If Vj is the input pulse stream from neuron j to neuron i, and the weight of the synapse between these two neuron is denoted by T_{ij}, then the output pulse stream V_i is a function of the weighted sum activity x_i such that,

$$V_i = f(x_i) = f(\sum_{j=1}^{j=n} T_{ij}V_j) \tag{1.1}$$

The above is the weighted sum equation which is the core of artificial neural network (ANN) algorithms that is realized in hardware as digital, analog and hybrid VLSI computation.

Digital techniques are used in the implementation of the weighted sum, the hardware platform can be implemented on a special chip, for example, the connected network of adaptive processors (CNAPS) chip is based around the processor node shown in Fig. 1.5 (b). The node comprises a multiplier (to form $T_{ij}V_j$), an "Adder" to perform the summation, RAM memory device, input/output control and an arithmetic shifter. Each CNAPS comprises a

64 processor Node. The device is configurable via microcode to define the architecture of CNAPS based system. CNAPS is one of the first commercial neurocomputers. This design puts the emphasis on a very limited connectivity: only two shared input and output buses connect the processing elements. Since there is no dedicated weight port on each processing element, weight memory is internal to the processing elements and, as such, rather limited [32, 33]. In addition to that the digital computations occupy large area and consume power for designing dense parallel networks [32, 33, 34].

Bacause of the limitation of the connectivity in digital architecture, analog techniques are used to design fully parallel architectures such as in Hopfield network shown in Fig. 1.5 (a). Although analog architecture sacrifices the computation precision, the low power encourages a massive dense network to be built. Yet, the Hopfield network was abandoned for a decade because of limitations in programmability. In the Hopfield network, the computation of the synapse weight is represented by the resistors and the op-amp which imply limited programmability. Once the resistors are fabricated within the CMOS process, the weight value and the network functionality are fixed which causes a lack in the dynamic change of the weight. However, introducing an array of resistors and switches to select the proper weight, increases the complexity and the size of hardware. Later on, the resistors in the above design are replaced by weak inversion MOSFET operation which take small space on the hardware chip. Mead [16] was the pioneer in introducing that since he found the similarity in operation between the transistors and nerve cell in which both has exponential transfer function. Yet, the computation in analog has limited programmability and storage. Many researchers later on used the floating gate that can dynamically change the weight. However, cascading modules of computational networks as well as the storage capability in analog hardware computation are still an open research topic.

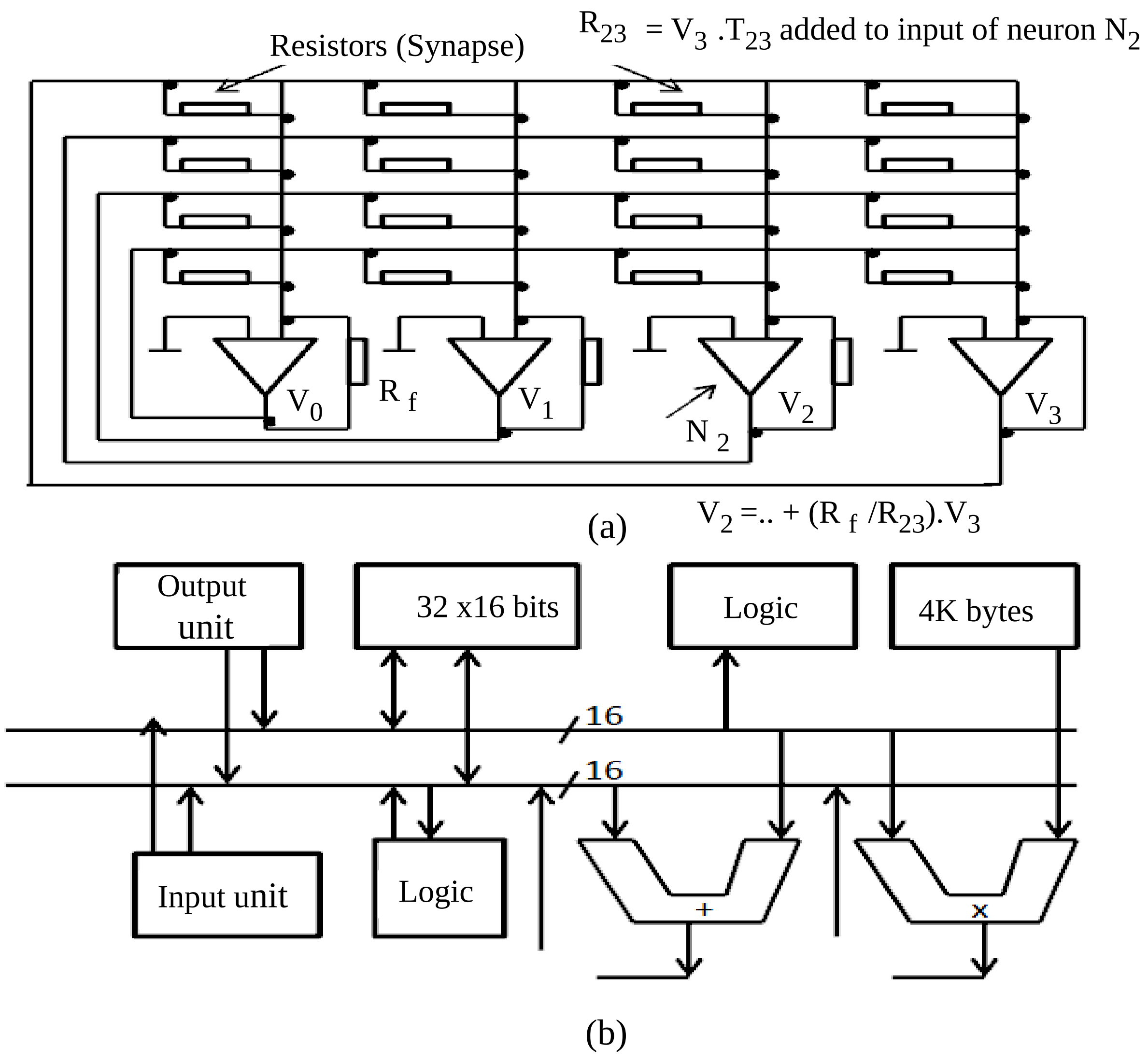


Figure 1.5: (a): Resistor array in Hopfield as fixed function analog as (b): Processor node in digital VLSI

To summarize the above, digital circuitry offer advantages to compute $T_{ij}V_j$ in terms of simple design and accuracy, but does not allow more than few hundred parallel multiply-add operations on a chip. However, analog VLSI allow massively parallel arrays of interconnected neurons in terms of thousands synapse that can be integrated on a single chip. These considerations have led to adapt a pulse stream computation technique which perform analog multiplication under digital control.

The underlying principles for using pulse coded stream in neural computations are the followings:

1. Analog computation is attractive in neural VLSI, for reasons of compactness, potential speed, asynchronous, and lack of quantization effects.

2. Analog signals are far from robust against noise and interference, are susceptible to process variations between devices and are not robust in the inter chip communication.

3. Digital silicon processing is more readily available than analog.

4. Digital signals are robust, easily transmitted and regenerated.

5. Digital consume more power and area than analog.

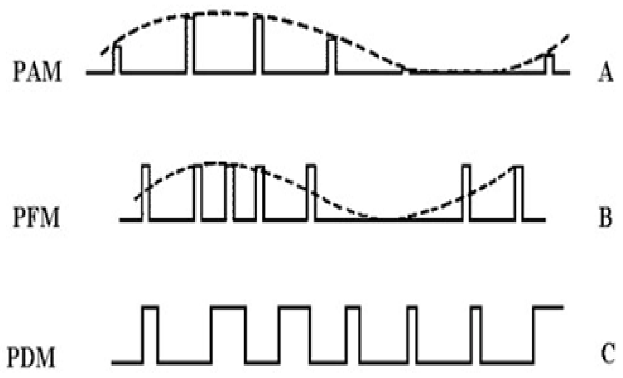


Figure 1.6: Some of the pulse coding, A is the pulse amplitude modulation (PAM), B is the pulse frequency or rate modulation (PFM),and C is the pulse duration modulation (PDM)

These considerations encourage a hybrid approach, seeking to blend the merits of both digital and analog technology. The pulse stream technique uses digital signals to carry information and control analog circuitry, while storing further analog information on the time axis. A number of possible techniques exist for coding a neural signals shown in Fig.

1.6 such as Pulse Amplitude (PAM), Pulse Width (PDM), and Pulse Frequency (PFM). The PFM is defined as the number of pulses per unit time corresponding to the amplitude of the analog signal. PFM is the coding mode used in the proposed computation model and it is referred as the rate or density modulation.

In the following, the approach of the pulse computation is introduced in mapping a dense computation of the weighted sum in the network architecture [35].

Fig.1.7 (a) shows the architecture of a single network of n totally interconnected neurons. Neurons, represented by circles, signal their states $\{V_i\}$ upward into a matrix of synaptic operators. The state signals are connected to an n-bit horizontal bus running through this synaptic array, with a connection to one synaptic operator in every column. Each column consists, therefore, of n operators, denoted by squares, each adding a new contribution T_{ij} to the running total of activity for the neuron i at the foot of the column. The function of the neuron is therefore to apply a sigmoid function to this activity to determine a neural state V. The synaptic function is to multiply a neural state V_p by a synaptic weight T_{ip} (stored in memory local to the synaptic operator), and add the result to a running total.

The neuron function is illustrated in Fig. 1.7 (c). The neuron is shown as receiving excitatory and inhibitory inputs, and producing a state output. The synaptic function is also straightforward at the functional level. Fig. 1.7 (b) shows a single synapse block. The (positive or negative) synaptic weight T_{ip} is stored in digital memory. To form the product $T_{ip}V_p$, the presynaptic neural state is gated according to the chopping signals derived from T_{ip}. The resultant product, $T_{ip}V_p$ is added to the running total propagating down either the excitatory or inhibitory activity channel, to add one term to the running total, as shown. One binary bit (the MSBit) of the stored T_{ij} determines whether the contribution is excitatory or inhibitory. The circuit of neuron that implements the sigmoid function and the circuit of

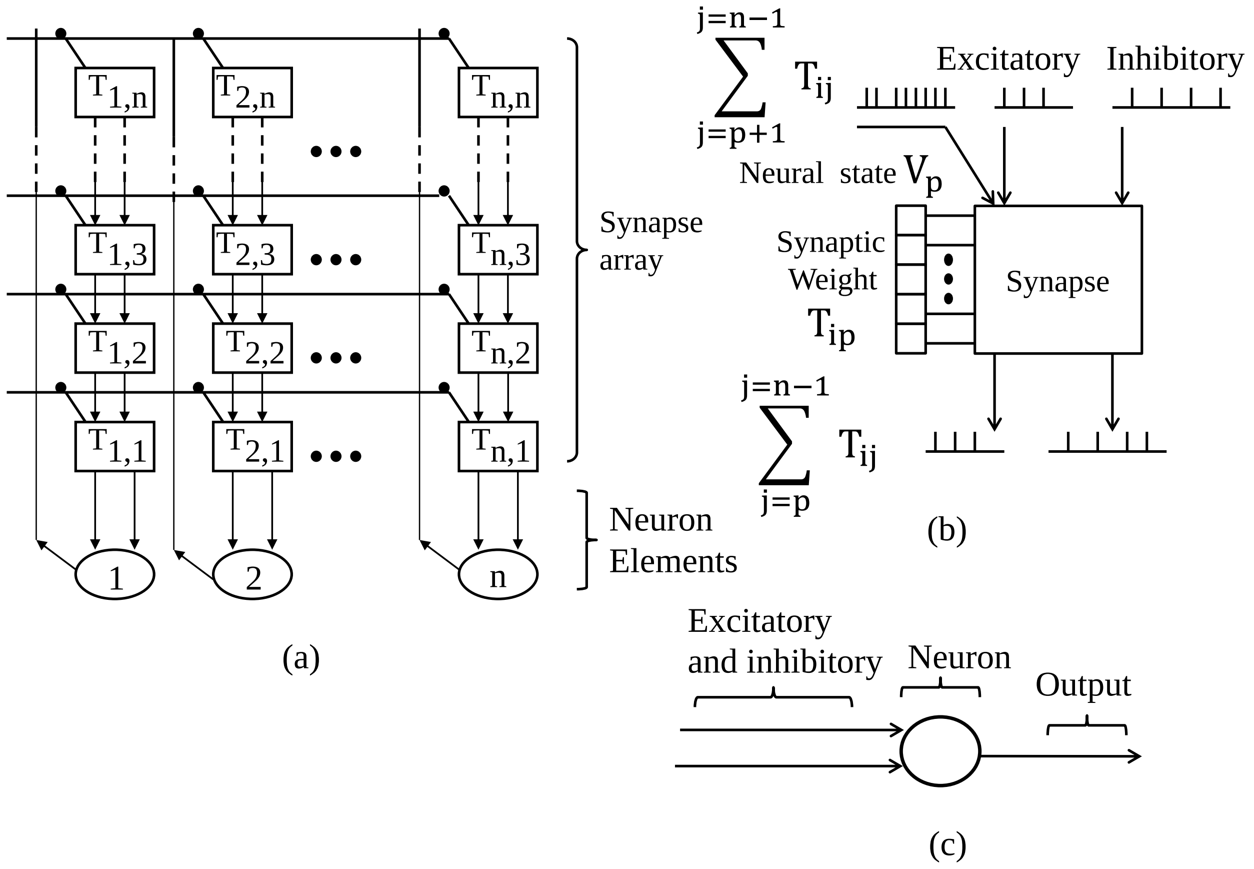
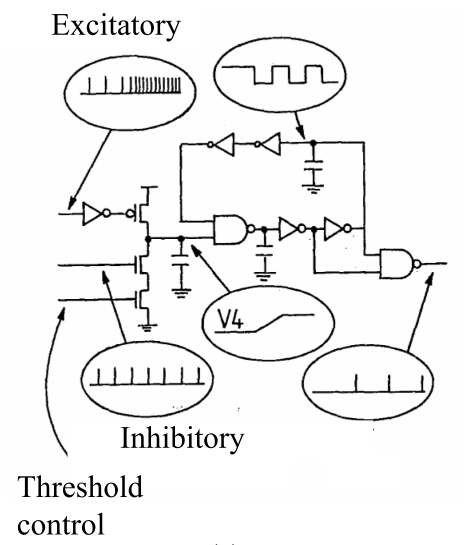


Figure 1.7: (a): Schematic architecture for a pulse - stream neural network. Neurons are denoted as a circle (c) and synaptic operators is denoted as square shape (b)[36]

the synapse that introduces the weighted sum is shown in Fig.1.8 (a,b) respectively [36, 34].

Another computational model of pulse coded neural network is introduced in [27, 37]. The levels of input pulses that represent the output from pre neuron V_i and the weight T_i introduce current into the synapse- neuron circuit which encode the input current into pulse density coding shown in Fig. 1.9(a, b). The output of the neuron in Fig. 1.9(b)is transmitted to other neurons via synapses. The weight inputs represents the connection strength (synaptic weight) between a particular neuron to another. Currents from all synapses connected to a neuron are summed at the input. The net current modulates the pulse train emanating from the neuron. As an application to this weighted sum circuit, a pulse coded



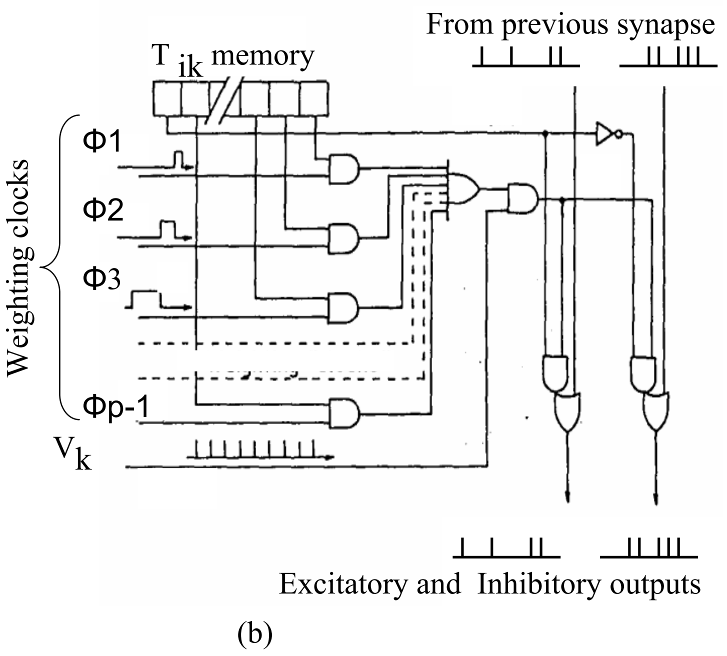


Figure 1.8: Neuron and synapse circuit in (a) and (b) respectively [36]

winner-take-all network (PWTA) has been implemented in a 2-micron CMOS process [27]. Fig. 1.9(b) shows a block diagram of the PWTA. The three neurons receive weighted inputs from V_1 and V_2. These three neurons in turn feed their outputs (Y_1, Y_2, and Y_3) to a global inhibition generator. This global inhibition generator feeds back inhibition signals to all the neurons in the network. Thus, if the output of one neuron is high, the I_{inh} inputs of all the neurons will be forced high. This will in turn discharge all the integrating capacitors in the network and thereby effecting the inhibition of all the neurons.

Many applications are implemented in signal processing and ANN in digital, analog VLSI and in pulse computation, but there is no scalable computation algorithm to implement more than the "weighted sum function". The limitation is that they are specific to certain application (algorithms based on weighted sum function) which means that any change in the applications needs to change the hardware structure and build a new system design. To

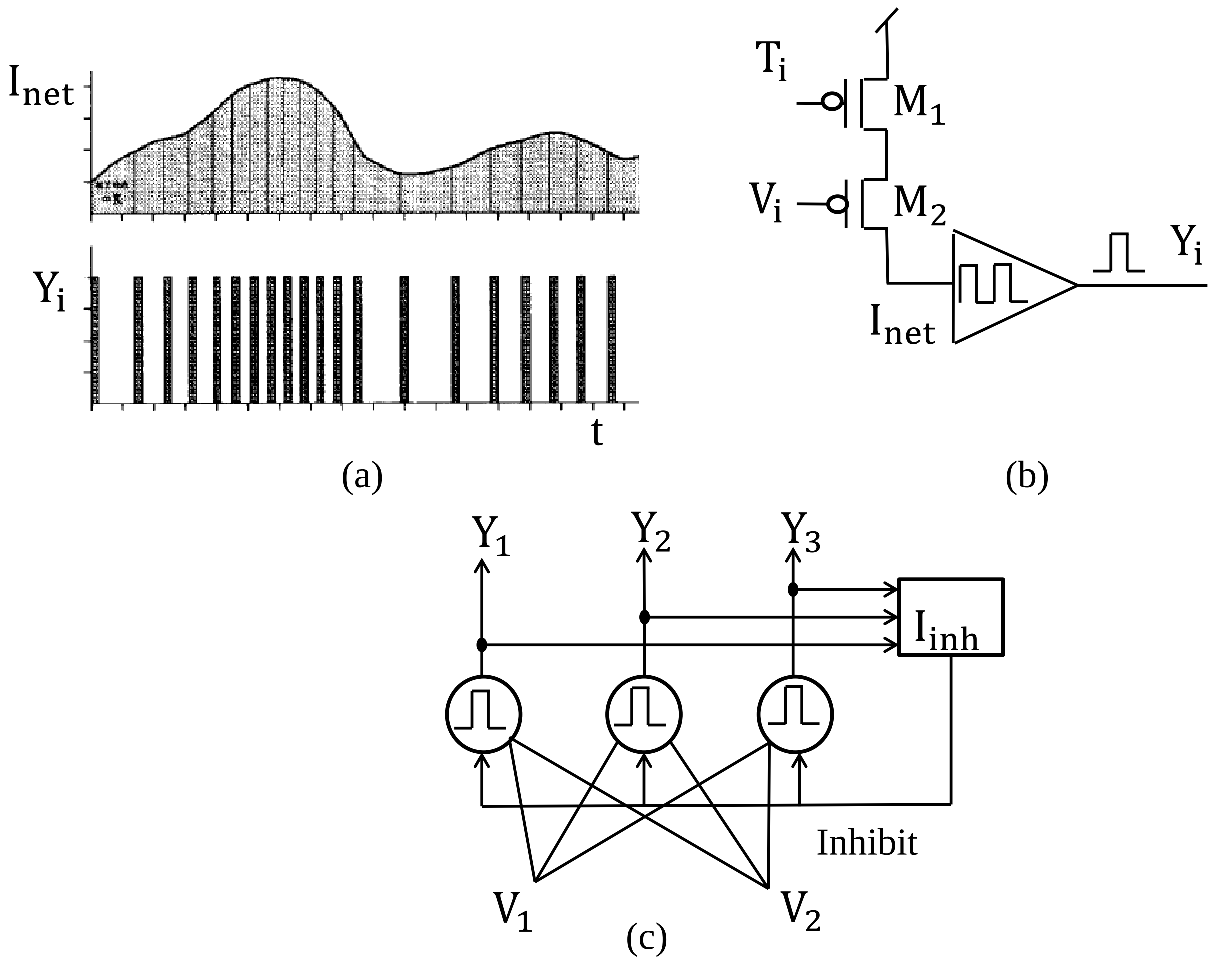


Figure 1.9: (a):Output pulse generated from input current (b):Synapse connection. (c):Pulse coded winner-take-all block diagram [37, 27]

address this limitation, an array of scalable -pulse computational modules are introduced to implement nonlinear functions (not restricted on computing only the weighted sum) and to map the algorithm into analog VLSI hardware. In order to achieve that, the proposed algorithm has to meet the items discussed in the next section.

1.2.4 Thesis objective

In this thesis, a scalable pulse computation algorithm and its hardware implementation based on integrate and fire structure and Margin propagation is developed to meet the following:

1. Convergence property: As a stochastic pulse stream computation, the algorithm has to meet the convergence and stability performance for the local unit and whole structure of the module.

2. Cascading and scalability property: The module can be cascaded in serial and parallel. The serial cascading supports the scalability property which means that the basic unit of the module can be cascaded to handle a growing amount of work in a capable manner.

3. Computation properties: As a new algorithm, the module has to meet the basic mathematical properties: scaling property, monotonicity, convexity, superposition and offset property.

4. Basic computation: The proposed module is able to map the basic computations (addition, subtraction, multiplication and division) and math expressions (power, polynomial and inner product) in pulse mode.

5. Computation capabilities: The computation capability means that the module is not restricted on computing only the weighted sum terms, but this unit can be mapped into other types of computations such as the dynamic equations in the Hidden Markov model and recursion equations in the Support vector machine. In addition to that, the IFMP algorithm generates an output that is equal to the maximum rate among the inputs with an adjustment of certain parameters.

6. Hardware realization: The basic concept of the thesis is the capability of mapping the algorithm into analog circuits.

7. Power and size efficiency: Since the algorithm can be verified in analog VLSI circuits, then it can be easily said that the design is compact (occupies small size) and low power since the design is adjusted in sub threshold or weak inversion mode.

8. The significance of the new pulse computation design: Because of the importance of the spiking networks with real-time and random input pulses, it is important to show the impact and significance of this design in the research space.

To meet the above objectives and implement the pulse computation algorithms, the equations of the algorithm have to be first mapped into the Log-sum-exp function and then mapped into margin propagation procedure and finally use the integrate and fire structure as the basic core to implement the algorithm. In the next two sections (sections 1.3 and 1.4), it is important to explain the reason of using log-sum-exp and how is this function related to both the margin propagation and the integrate and fire structure. In section 1.5, the methodology and tasks are listed to achieve the thesis objectives.

1.3 Log-sum-exponent and margin propagation

To meet the objective, the proposed algorithm and hardware architecture uses log-sum-exp and margin propagation. The first term "log-sum-exp" abbreviated as LSE is an important function because of the following:

1. LSE is used in factor graph and signal processing applications such as Bayesian belief propagation [38]. In factor graph algorithms, the basic functions used are the sum of

product terms (probability terms) which are used to evaluate the probabilities or the marginal functions of passing messages between the nodes and variables of the factor graph. Since the product of probability terms tends to decrease as the number of probability terms increase, then a problem of underflow will be popped up and cause false computations. Therefore, such algorithms use the log-likelihood computation to eliminate the underflow problem.

2. LSE is used to increase the dynamic range of variables in the computation process since it depend on logarithm functions of the input parameters.

3. For n input parameters, the LSE function evaluate the max value of the input parameters as shown in the following proof:

Let $P_1, P_2, P_3..P_n$ represent probability variables and Let

$$f = \sum_k P_{1k}.P_{2k}.. \longrightarrow P_{nk} \tag{1.2}$$

Then the LSE expression is represented as,

$$log(f) = log \sum_k e^{log(P_{1k})+log(P_{2k})+..\longrightarrow log(p_{nk})} \tag{1.3}$$

Now Let $a = log(P_{1k}) + log(P_{2k}).. + log(P_{nk})$ and set $b = max(a_{ik})$ then

$$log(f) = log(e^b \sum e^{a_{ik}-b}) = b + log(\sum e^{a_{ik}-b}) \simeq b \tag{1.4}$$

The last term is approximately equal to b if all the terms of a_{ik} (except the max value) are too small. The above trick is to shift the terms of a_{ik} to the right by a maximum value of inputs equal to b, evaluate the log-sum-exp then shift back to the original position.

In this work, the LSE expression is the first step to map non-linear functions. t can be said the following : "If a linear/ non-linear function can be mapped into LSE, then the function can be implemented in the proposed architecture".

An example below demonstrates a mapping process of any function into LSE.

Example: Let a,b,c and d $\in R$ and let $f = ab + cd$, then

$$log(f) = log(ab + cd) \tag{1.5}$$

$$z = log(f) = log[e^{log(a)+log(b)} + e^{log(c)+log(d)}] \tag{1.6}$$

The last term of the example is called the LSE expression.

As mentioned earlier, the LSE expression is used in factor graph algorithms. Next, the factor graph algorithm is defined and how it can be represented in LSE.

A factor graph is a graphical representation that shows how a function of several variables can be factorized into a product of smaller functions and it is useful because it simplifies complicated efficient algorithms. In the factor graph, the nodes (rectangles) represent the factors or functions while the edges represent the variables [39]. The proposed model can be applied in factor graphs such as in the inference problems. In such problems, the probability of the message propagated between two independent graph sets can be evaluated based on belief propagation. An example shows the concept of factorization by evaluating the probability in any path of the graph using sum product rule [39]. The main idea of using the factor graph is to extract some variables by sum operators that reduce the computation of messages passing through nodes and edges. Lets take a function $f(x_1, x_2, x_3..x_n)$ where f is the probability density function (PMF) of discrete random variable $x_1, x_2, x_3..x_n$, then in order to compute $f_k(x_k)$ such that (example: $n = 4$ and $k = 3$ in Fig 1.10),

$$f_k(x_k) = \sum_{x_1..x_n except \ \ x_k} f(x_1, x_2, x_3..x_n) \tag{1.7}$$

where $f_k(x_k)$ is the PMF of x_k. If $f_k(x_k)$ can be divided into two independent graph sets, then it can be inferred that the probability of the message propagated between the two sets is calculated by multiplying the probability of the two sets. This can be calculated by using sum-product algorithm in which the large marginalization is split into a sequence of small marginalization. For example, assume that

$$f_k(x_k) = \sum_{x_1..x_n except \ \ x_k} f(x_1..x_k)f(x_k..x_n) \tag{1.8}$$

and the marginal probabilities $p_a(k)$ and $p_b(k)$ are denoted as,

$$p_a(x_k) = \sum_{x_1..x_n except \ \ x_k} f(x_1, ..x_k) \tag{1.9}$$

$$p_b(x_k) = \sum_{x_k..x_n except \ \ x_k} f(x_k..x_n) \tag{1.10}$$

then the probability of the message passing in path k is:

$$p(x_k) = p_a(x_k)p_b(x_k) \tag{1.11}$$

The above equations are expressed in log-sum-exp as following:

$$log(p_a(x_k)) = log(\sum_{x_1..x_k except \ \ x_k} e^{log(f(x_1..x_k))}) \tag{1.12}$$

$$log(p_b(x_k)) = log(\sum_{x_k..x_n except \ x_k} e^{log(f(x_k..x_n))}) \tag{1.13}$$

$$log(p(x_k)) = log(p_a(x_k)) + log(p_b(x_k)) \tag{1.14}$$

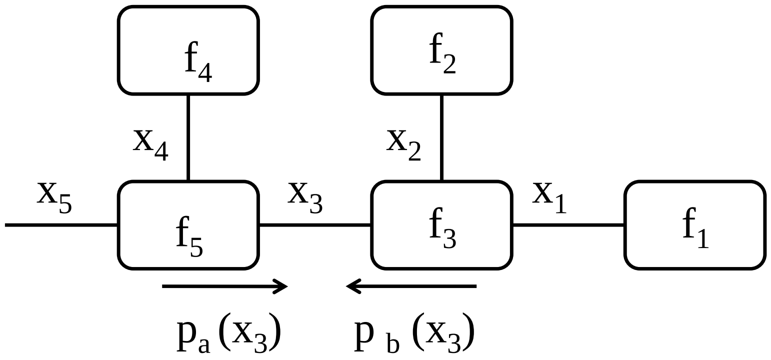


Figure 1.10: Factor graph representation to show that $p(x_3) = p_a(x_3).p_b(x_3)$

In factor graph, the algorithms are applied extensively using analog input parameters. But in this work, the input parameters are represented as a stream of random pulses and applied in scalable architecture. Since it is difficult to implement modularity in LSE, an approximation method to the LSE is needed. Therefore, Margin propagation method is introduced as an approximation method to LSE. In previous work [40], it was proven that margin propagation is an approximation method to LSE. The input/output variables in that work are represented as currents and the computation procedure is implemented using Kirchoff's current law. Margin propagation was implemented in these circuits to achieve scalability in the decoding algorithms. In this work, the concept in pulse computation mechanism of margin propagation as an approximation method to LSE is introduced.

The core of the margin propagation (MP) algorithm is based on the concept of the reverse water-filling (RWF) algorithm, [41]. Given a set of random inputs (scores) $L_i \in R; i = 1 : N$, the RWF algorithm computes the solution z according to the constraint

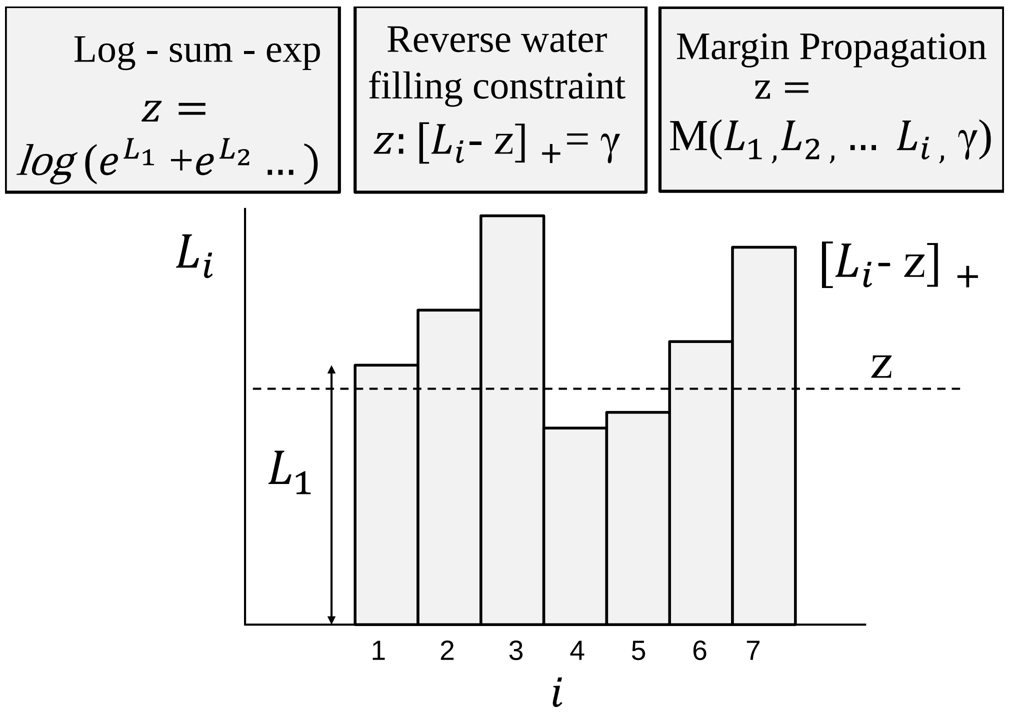

Log - sum - exp
$z = log(e^{L_1} + e^{L_2} ...)$
Reverse water filling constraint
$z: [L_i - z]_+ = \gamma$
Margin Propagation
$z = M(L_1, L_2, ... L_i, \gamma)$
L_i
$[L_i - z]_+$
z
L_1
1 2 3 4 5 6 7
i

Figure 1.11: Water level filling constraint

$$\sum_{i=1}^{N}[L_i - z]_+ = \gamma \tag{1.15}$$

where $[.]_+ = max(.,0)$ denotes a threshold operation and $\gamma \geq 0$ represents a parameter of the algorithm. Note that z is in the log domain. The solution of the equation 1.15 can be visualized using Fig. 1.11, where the cumulative scores beyond z (shown by the shaded area) equal γ. The limit of z is such that

$$z \leq \underset{i}{max} L_i$$

Equation 1.15 can be rewritten in the form of

$$z = M(L_1, L_2, ..., L_N, \gamma) \tag{1.16}$$

where M denotes the MP function and N denotes the number of operands in the MP function. Then LSE is represented as following

$$z = log(\sum_{i=1}^{N} e^{L_i}) \simeq M(L_1, L_2, L_3....L_N, \gamma). \tag{1.17}$$

To summarize the above, it can be deduced that LSE is a useful function to map signal processing equations or factor graph algorithms. Yet, it is difficult to represent a cascaded LSE in analog circuits. In this work, the LSE is approximated into MP algorithm which is verified to implement scalable computation. The approximated concept from LSE into MP is realized in hardware using a novel design based on integration and threshold operation. This operation is realized using integrate and fire structure (explained in the next section) as the core unit of the computational module.

1.4 Integrate and fire structure

In order to realize the threshold operation used in the MP algorithm discussed in the previous section, it is important to implement this operation in the circuit design level. The integrate and fire (IF) structure is introduced to calculate the threshold operation. IF model is a simple representation of a neuron, it is extensively used as a neuron model in spiking neural networks [42, 43, 44] and neuromorphic systems [45, 4]. The neuron model is an important computational structure and it is considered as the basic core in designing ANN which has huge applications in processing biological, statistical and engineering applications.

To understand the operation of IF neuron model, the structure of neuron is explained briefly. The neuron has three parts as shown in Fig. 1.12:

1. Dendrites: These branchlike projections of the cell make connections to other cells and allow the neuron to talk with other cells .

2. Soma: It is the place that joins the signals from other neurons via dendrites. The soma and nucleus do not play a role in the transmission of neural signals but they maintain the cells functionality.

3. Axon: It is located at the end of the soma. It controls the firing of the neuron. If the total potential reaches a certain threshold, then the neuron will fire. It propagates the spikes to other cells.

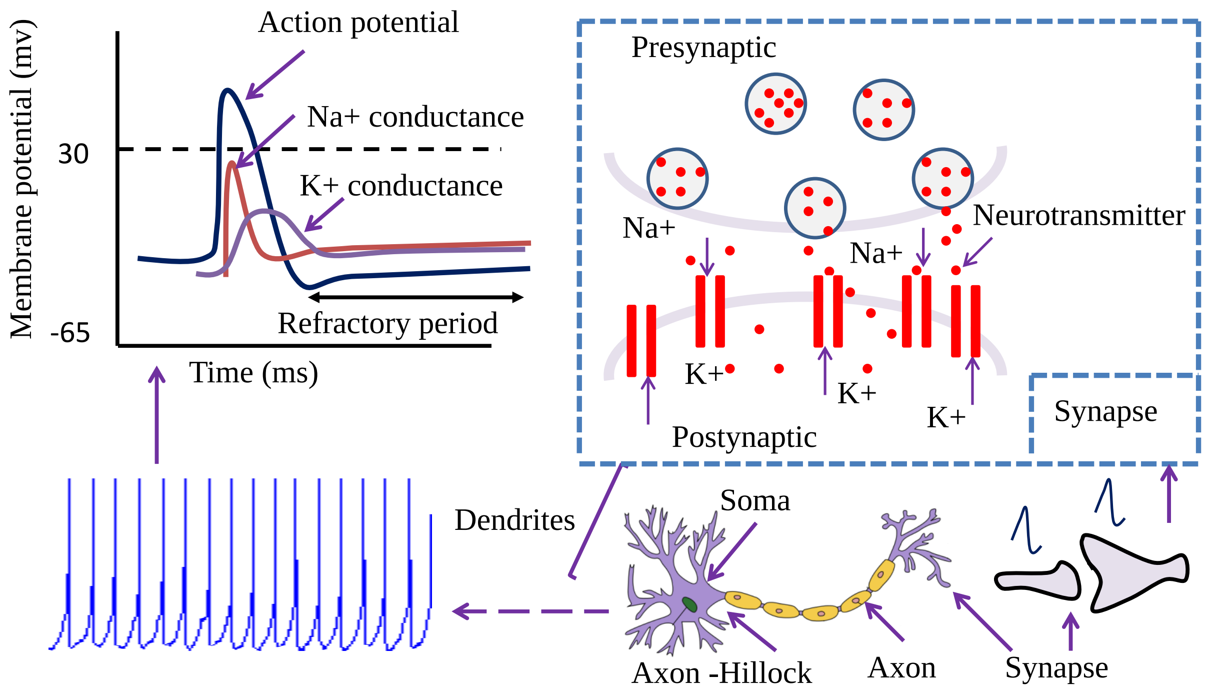


Figure 1.12: Mechanism of spike-generation and signal propagation through synapses and neurons

Nerve cells communicate with one another using spikes (pulses) passed through specialized junctional structures called synapses. The spikes are generated when pulses exit neurotransmitters in the presynapse and release ions to the postsynapse as shown in Fig.1.12. The underlying mechanism of spike or action-potential generation is due to unbalanced movement of ions across the membrane which alters the potential difference between the inside and the outside of the neuron. In the absence of any stimuli to the neuron, the potential of the membrane with respect to the potential outside the membrane is about -65 mV, also referred to as the resting potential. This potential is increased by the influx of sodium ions (Na^+) inside the cell causing depolarization whereas the potential is decreased by the efflux of potassium ions (K^+) outside the cell causing hyper polarization. Once the action potential is generated, the (Na^+) ion channels are unable to reopen immediately until a built-up potential is formed across the membrane. The delay in reopening the sodium channels results in a time-period called refractory period, as shown in Fig. 1.12, where the neuron cannot spike. There are two types of synapses typically encountered in neurobiology: excitatory synapses and inhibitory synapses. For excitatory synapses, the membrane potential of the post-synaptic neuron (referred to as the excitatory post-synaptic potential or EPSP) increases, whereas for inhibitory synapses, the membrane potential of the post-synaptic neuron (referred to as the inhibitory post-synaptic potential or IPSP) decreases. It is important to note that the underlying dynamics of EPSP, IPSP and the action potential are complicated and several books have been written to discuss only the mathematical model [46, 43]. Therefore, for the sake of brevity, a simple integrate-and-fire neuron model is described which is used in the design of neuromorphic sensors [4].

A schematic diagram of the IF model is shown in Fig. 1.13 (a). The basic circuit inside the dashed circle represents the IF model. The IF structure is described as following: when a current $I(t)$ charges the capacitor c representing the membrane of a neuron, a voltage $v_m(t)$ across the capacitance (points) is built and compared to a threshold. If $v_m(t) = v_{th}$ at firing time t_i^f, then an output delta type pulse $\delta(t - t_i^f)$ is generated. The generated

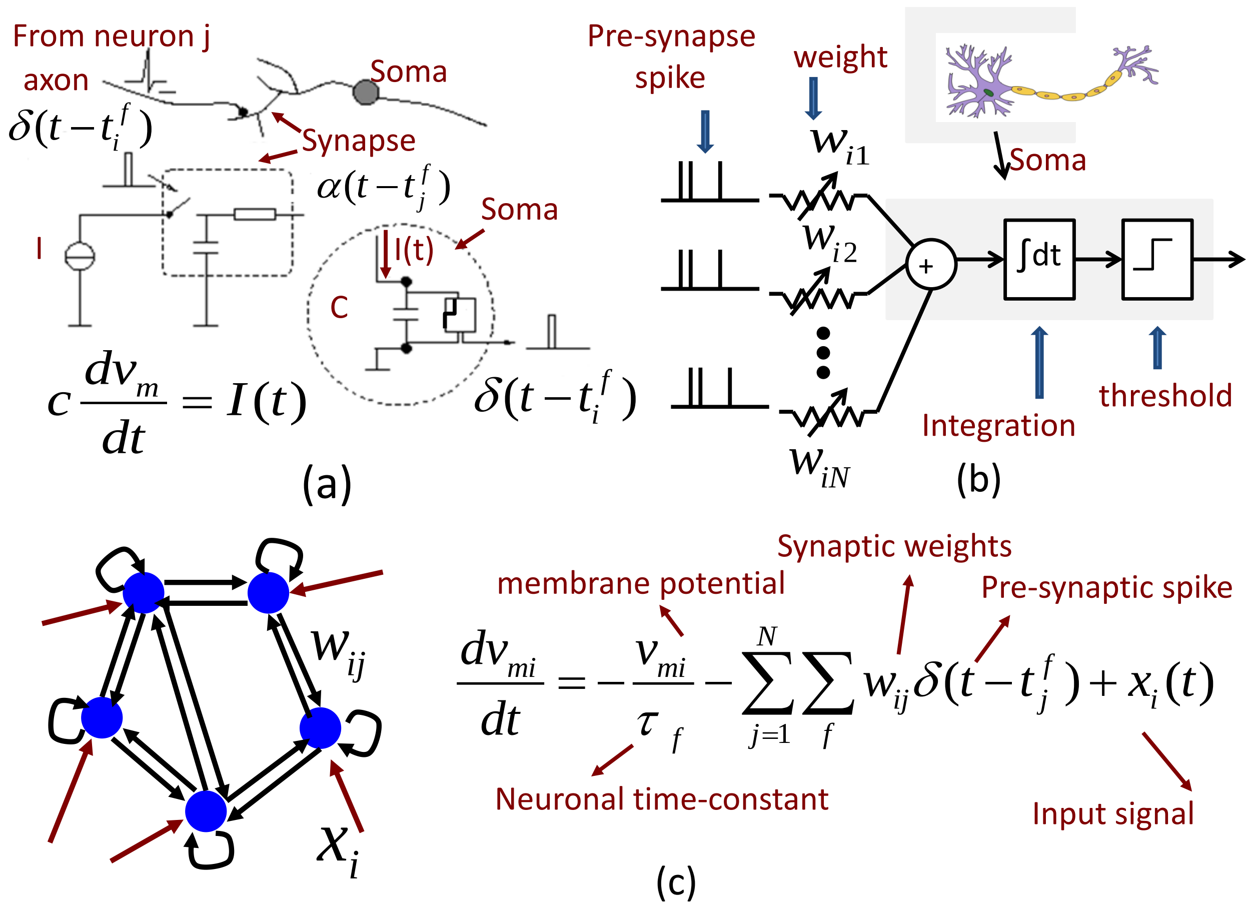


Figure 1.13: (a): Simplest representation of IF representation as neuron model neuron. The basic circuit is the module inside the dashed circle on the right-hand side. A current $I(t)$ charges the C circuit and a voltage across the capacitance is compared to a threshold v_{th}. Left part: A presynaptic representation as a low-pass filter. (c): The dynamic equation of the membrane voltage in a population of spiking neurons in (b)

pulse resets the capacitor voltage to its initial value and hence the pulses are continuously generated as long as the capacitor is charging/discharging. In the same figure (Left), the synapse is represented as a low-pass filter whose input pulses are $\delta(t-t_j^f)$ and the output pulses are $\alpha(t-t_j^f)$. The membrane voltage v_m is simply denoted by the following equation that describe the charging of the membrane capacitance c as,

$$c\frac{dv_m}{dt}=I(t) \tag{1.18}$$

The above equation is written in discrete time at sampling time n and sampling time t_s as,

$$v_m[n] - v_m[n-1] = \frac{I[n].t_s}{c} \tag{1.19}$$

In Fig. 1.13 (a), the input current I to the postsynaptic neuron i is due to the spikes of presynaptic neurons j. In the simplest model of a synapse, each presynaptic spike arrival evokes a postsynaptic current with a standard time course α. The total input to neuron i is denoted as,

$$I_i = \sum_{j,f} w_{ij} \alpha(t - t_j^f) \tag{1.20}$$

Where the sum runs over all firing times t_j^f of all presynaptic neurons. The factor w_{ij} is the synaptic efficacy of a connection from a presynaptic neuron j to a postsynaptic neuron i. In Fig. 1.13 (b), the neuronal population is shown and the membrame potential for neuron i in Fig. 1.13 (c) is denoted as [23],

$$\frac{dv_{mi}}{dt} = -\frac{v_{mi}}{\tau} - \sum_{j=1}^{N} \sum_{f} w_{ij} \delta(t - t_j^f) + x_i(t) \tag{1.21}$$

Where v_{mi} is the membrane voltage for neuron i, w_{ij} is the weight from neuron j to neuron i that could be positive/ negative depending on the excitatory/ inhibitory connection of the presynapse, t_j^f is the firing time of neuron i, $\delta(t - t_j^f)$ is the pre synapse pulses and $x_i(t)$ is an external exciting potential to the neuron cell. Different mathematical models have been introduced and implemented in hardware to simulate the dynamic characteristics of the IF model [43, 42]. In this work, a simple mathematical equation (1.19) is used to represent the IF model since it implements the integration and threshold operations.

In analog circuits, the first analog IF model was introduced by Carver Mead in 1989 as a

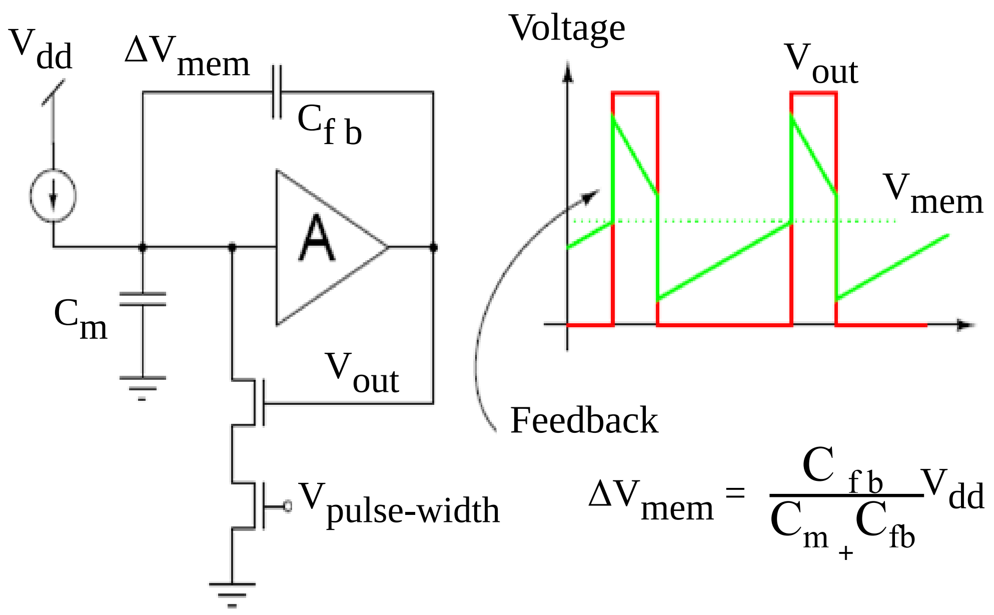


Figure 1.14: Conceptual schematic of a circuit that generate pulses due to positive feedback through capacitor c_{fb} across high gain buffer. The lower transistor adjust the pulse width during discharge phase of the capacitors

core to a dense layout using a VLSI neuromorphic chip. The designed circuit shown in Fig. 1.14 is simple and easy to implement in hardware [1]. It consists of an integrating capacitor (representing the membrane capacitance of a neuron), a simple high gain amplifier (double inverter) that switches when the integrated input current exceeds its threshold voltage and a feedback capacitor that adds extra charge to the membrane capacitor to stabilize the firing state. The reset is achieved through a leakage that is turned on while the output is high. That leakage current determines the pulse length of the output pulse. When the output voltage is high, it resets the membrane voltage and the output pulse resets to the starting point again. At this point, the input current charges the capacitor c_{in} until the input voltage of the amplifier reaches the threshold level at which in which it turns the output V_{out} into logic high. The feedback capacitor (C_{fb} and membrane capacitor C_m are considered here a capacitor voltage divider which changes the timing duration of pulses and adjusts the change in slope of membrane voltage V_{mem} which decides the output pulse state V_{out} of the amplifier.

Conceptual, this model is considered in two ways, the first one is to be considered as an analog-to-analog converter that transforms an analog current into an analog time interval. The pulse frequency is proportional to the analog input current. One advantage of this representation in the time domain is that one can convey analog data with a digital signal, for example for transmission via a digital wireless link. The second view is conceptually the same but has different names in that the IF model can be viewed as an analog-to-digital converter, where the output pulses of the network faithfully encodes different features of the input analog sensory stimuli into a time encoded signal or into a train of spikes [47, 48]. IF circuit is asynchronous, meaning it has no clocks, as compared to the conventional ADCs. The asynchronous property leads to a significantly lower output data rate and lower power consumption as compared to conventional ADCs [49]. In this work as explored in chapter 2, a more dense circuit of the IF is designed to implement the threshold operation. The designed circuit has only one small feedback capacitor equal to 200 fF.

1.5 Tasks and Methodology

The methodology of achieving the proposed pulse computational model and hardware implementation is to design the analog circuit of the "IF" unit as the core unit in the computational model, verify the dynamic characteristics and the convergence property in the cadence tool and then apply it in a novel design to implement spike time dependency. The "IF" unit is then simulated in Matlab to verify and analyse the main concept of the "IFMP algorithm".

For hardware implementation of the IFMP algorithm, the simulated "IFMP" is mapped and verified in analog circuit. The convergence of the "IF" unit and the "IFMP" algorithm are proven mathematically. In order to show that the concept of IFMP algorithm is applicable in computations and applications, the IFMP structures are simulated in Matlab to verify the basic computations (addition, multiplication, power, inner product and polynomial) and applications (winner takes all, decoding and classification). An array of IFMP units is fabricated on a chip using 0.5 μ CMOS circuitry to verify the concept in hardware.

The tasks to achieve the proposed computational model include the followings:

1. Design the analog circuit of the integrate and fire structure (IF), which is the basic unit of the pulse computational algorithm IFMP. The design of the analog IF implements the integration and threshold operations, which refer to the operations of integration the currents caused by the input pulses and then comparing the voltage produced by the integration with a certain threshold to produce output pulses. The first step in the successful IF design is to verify the logarithmic transfer function of the IF as the pulse rate output versus input current. The second step is to verify the convergence characteristic of spiking IF structure which implies that the expected value of a train of input pulses converges to the expected value of the output pulses. Here the expected value of pulses is defined as the average of pulse levels over a time period of these pulses.

2. Verify the convergence of one IF structure for more than one input in which the sum of expected values of input pulses are converges to the expected value of the output pulses.If the input pulses are denotes as L_i, then the $E(L_1) + E(L_2) + ...E(L_n) \longrightarrow E(output)$, where 'i' is the index of the input pulses, 'E' is the expected or average

value of the pulses and 'n' is the number of inputs.

3. Verify the concept of IF structure as a computational unit itself in the spike dependency plasticity, which is the learning concept in synaptic junctions of neurons.

4. Simulate steps 1, 2 and 3 in Matlab and verify the concept of approximating the log-sum-exp into margin propagation algorithm in pulse mode. The design of the technique is implemented using a network of IF structures (IFMP) and simulated to verify the convergence of margin equation for all values of the adjusting parameter γ.

5. Derive and simulate how much the difference is between the IFMP algorithm and the "log-sum -exp" function.

6. Using MOSFET transistors, design, simulate and verify the dynamic characteristics of an analog circuit implementation of the basic pulse computational algorithm IFMP.

7. Prove mathematically the convergence of one IF structure and IFMP algorithm. Specifically it is shown by induction that the convergence is valid if the integration of step voltage is bounded between two limits.

8. Prove and simulate the concept of cascading multi - two inputs IFMP structures which is equivalent to the parallel inputs to the IFMP. The importance here is to built scalable units of an IFMP array on silicon in order to implement computational function.

9. Simulate in Matlab and circuits the basic computation of nonlinear functions such as multiplication, power, polynomial and inner product by mapping the functions into the margin propagation method.

10. Simulate in Matlab and circuits the applications of sequence and trace detection using Hidden Markov model (HMM) by mapping the algorithm of HMM into margin propagation.

11. Simulate in Matlab and circuits binary classification using a Support vector machine method (SVM) by mapping the algorithm of SVM into margin propagation.

12. Show the significance of the IFMP algorithm in dynamic range capability compared with stochastic pulse computation using HMM decoding algorithm.

13. Show the significance of the IFMP algorithm that can implement easily spiking winner take all network compared with other architectures.

14. Design an array of analog IFMPs on a 0.5 μ CMOS process for testing and verification the dynamic characteristics of a single IFMP.

15. Design an array of analog IFMP on a 0.5 μ CMOS process for testing and verification the simulation procedure of HMM.

1.6 Thesis organization

The thesis is organized as following: Chapter 2 introduces the circuit diagram of an integrate and fire model, the convergence performance of one neuron and an application of a single integrate and fire neuron model. In the application, a floating gate transistor is used as a storage device to modify the weight in a novel architecture of synapse plasticity (modification of weight strength between synapses). However, the above design is limited to a specific task. The goal here is to built a scalable computational unit that can be plugged into many applications. Therefore, the main topic of the thesis is introduced in chapter 3 to built a scalable computational unit, "the integrate- fire structure and margin propagation method (IFMP)". The chapter includes a proof of the convergence of the dynamic performance, the IFMP parameters and cascading, the properties of MP and basic computations examples of the IFMP. Chapter 4 introduces three examples of the proposed pulse computation algorithm to implement non-linear functions of signal processing algorithms. As a circuit simulation and layout design on a chip, the first example is a novel implementation of the Hidden Markov Model algorithm (HMM) to recognize sequences in pulse based computation. The first example is based on discrete probability observation while the second example is based on the continuous probability observation. The third example is an implementation of a support vector machine to classify two sets of data types. Chapter 5 explores the significance roles of using IFMP algorithmic module in the research space. In other words, how can the IFMP modules enhance the performance and add specific properties to the applications. Finally, chapter 6 conclude the thesis with a summary and future work recommendations.

Chapter 2

Integrate-Fire Model and Synaptic Plasticity

A new design of the integrate and fire (IF) model is introduced with two concepts:

1. First concept: The generated pulses are proportional to the injected current to the neuron.
2. Second concept: The generated pulses are proportional to the input pulses which is refereed to a convergence status. Convergence status mean that the output pulse rate convergence in value to the input pulse rate.

It is shown in this chapter that the first concept of the IF structure has computational capability in changing the synapse weight, yet it has a limited capability in implementing other functions. Therefor, the second concept of the IF model is introduced, in the next chapter, as the core of a scalable pulse computational algorithmic module which is the main topic of this thesis. In section 2.1, the new design of the IF model and the proof to the convergence status are discussed. In section 2.2, the first concept of generating the IF model is introduced to change the synapse weight in a mechanism called the spike time dependency plasticity (STDP). Section 2.3 is the summary of the chapter.

2.1 Pulse based IF

The concept of the integrating and fire neuron model (IF) is to integrate the injected currents to increase the membrane voltage until reaching a threshold. Then the output of the IF model fires and resets the membrane voltage into its initial value. The membrane voltage starts increasing again according to the values of the injected currents. This concept is translated into analog circuit using an integrator and inverter as a comparetor circuit to compare the membrane voltage V_m with the threshold value (Fig. 2.1-b).

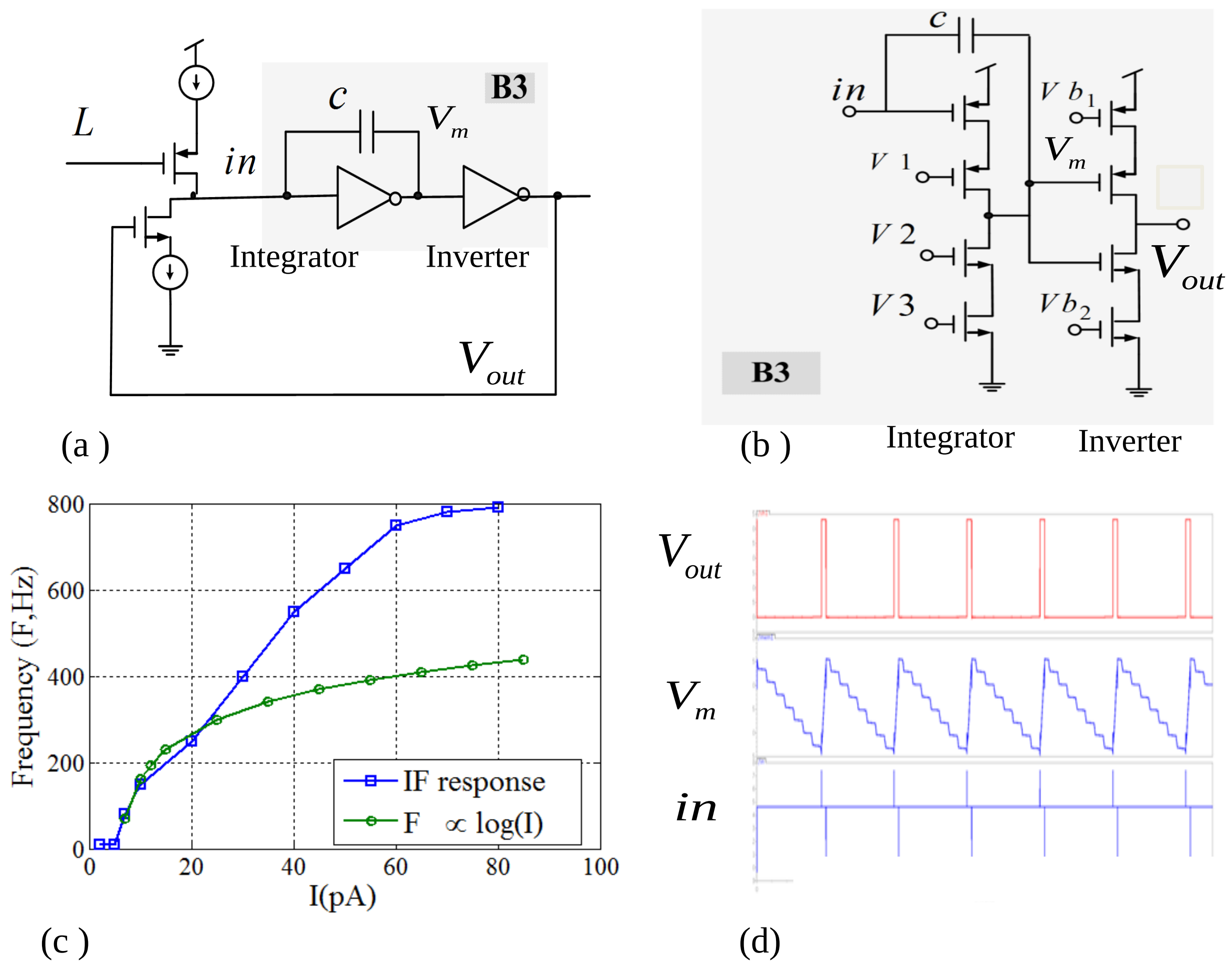


Figure 2.1: (a): Integrate and fire schematic circuit. (b): Integrator and cascoded inverter. (c): Transfer function of the IF unit. (d): The output voltage V_{out}, the membrane voltage V_m and input pin in of the integrator

The lower boundary is adjusted through biasing voltages V_{b1} and V_{b2}. The upper bound is equal to $V_{dd} - 2V_{eff}$, where $V_{dd} = 3.3V$, and $V_{eff} \approx 0.5V$. Note that $V_{eff} = V_{gs} - V_{th}$. The biasing V_1, V_2, V_3 are used to adjust the integrator in the high gain region. In this circuit, the biasing is adjusted so that the integration is bounded between $2.3V$ and $0.9V$. Here, the lower bound is the threshold value and the upper limit is the reset value of the membrane. In other words, the integration of currents cause the membrane voltage to decrease from it's restart value $2.3V$ until it reach the lower bound or the threshold (0.9 V).

Initially, if the input of the integrator is zero, the outputs of the integrator and the cascoded inverter are equal to $3.3V$ and zero volt respectively. If the input voltage increases and reaches the high gain region of integrator amplifier, then the integration phase starts to built up which is considered as the discharging phase of the feedback capacitor. When the output voltage of the integrator V_m reaches the lower bound, then the outputs of the inverter V_{out} is turned into logic one ($3.3V$). At this point, the output of the inverter is logic one which resets the output voltage of the integrator to the upper bound (charging phase of capacitor) through the feedback to the NMOS shown in Fig. 2.1(a). The cycle of charging and discharging the feedback capacitor C is repeated according to the amount of the injected current to the inputs of the integrator ('in' node) as shown in Fig. 2.1(d). The injected currents to the integrator are applied respectively during off and on states of the input pulses L for the excitatory path (PMOS transistors) and through feedback pulses (output pulses) in the inhibitory path (NMOS transistors).

Fig. 2.1(c) shows the relation between the injected current and the output rate. For a single neuron, it had been remarked that the relation between external stimulus and neural response is approximately logarithmic [50, 51] which has been noticed in the designed IF model too. This means that addition of two neural signals is performing multiplication of the input signal. For example if x_1 is the input to neuron A and x_2 is the input to neuron B and if it is considered that the response of neuron is exactly log, then the sum of the output of neurons A and B is a multiplicative relation in log domain; $log(x_1)+log(x_2) = log(x_1x_2)$. Thus multiplication and division can be implemented by adding and subtracting outputs of neurons with a logarithmic transfer function [29]. The importance of the above is to show the computation aspect of the IF model that can encode two separate inputs into one output which is a detecting process (i.e: detecting the occurrence of timely simultaneous inputs yet spatially separate input signals).

Next, the concept of the IF structure as pulse response with respect to pulse rate is introduced and analyzed. The change of the applied current to the integrator occurs by applying a stream of input pulses to the gates of the NMOS and PMOS switches at the input paths of the integrator. The change of the input rate is caused by changing the stream of the input pulses L.

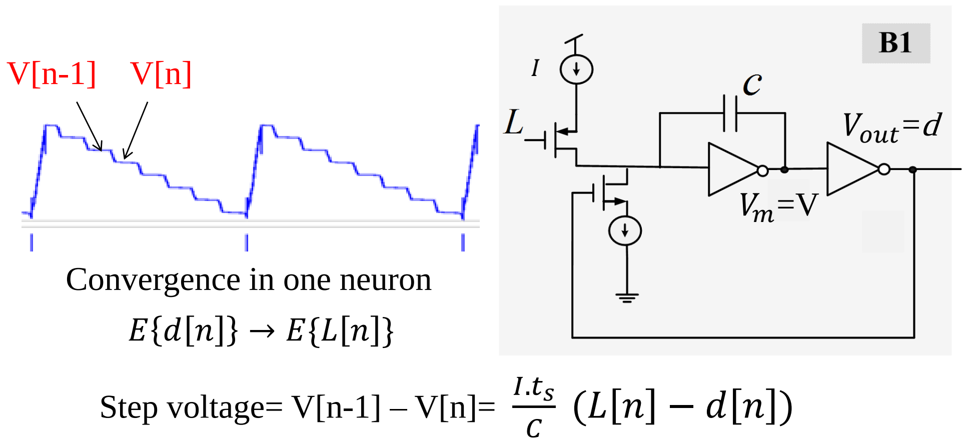
The circuit is designed to fulfill the convergence property for pulse rate convergence. It is important to set the parameters (sampling time of integration, capacitance, currents) of Fig 2.1 (a) such that the expected rate of the output pulses convergence to the expected rate of the input pulses. In order to do that the currents of both excitatory and inhibitory inputs are set to the same value. The convergence of IF means that the expected value of output spikes $d[n]$ is equal to the expected value of the input spike $L_i[n]$ overall the samples,

$$\mathcal{E}_n\{L[n]\} = \mathcal{E}_n\{d[n]\} \tag{2.1}$$

To prove the convergence of IF unit, the equation for the change in the step voltage is shown in Fig. 2.1 (d) can be written as,

$$V[n] = V[n-1] - \frac{t_s I}{C}(L[n] - d[n]) \tag{2.2}$$

where V is the membrane voltage, n is the instance time, C is the capacitance that represent



V[n-1]
V[n]
Convergence in one neuron
$E\{d[n]\} \rightarrow E\{L[n]\}$
B1
I
L
C
$V_{out}=d$
V_m=V
Step voltage= V[n-1] – V[n]= $\frac{I.t_s}{C}(L[n] - d[n])$

Figure 2.2: IF model rate convergence

the membrane, t_s is the time sampling, I is the biasing current in the NMOS and PMOS paths, $L_i[n]$ is the Bernoulli random input variables and $d[n]$ is the output spikes that take the values { 0,1 } according to the threshold voltage (V_{th}) and the membrane voltage V, where,

$$d[n] = 0.5[sgn(V_{th} - V[n]) + 1] \tag{2.3}$$

Then it must be proven that when the membrane voltage $V[n]$ is bounded, then the expected value of the output $d[n]$ is equal to the expected value of the input $L[n]$:

$$\text{if} \quad |V[n]| \leq c \quad \text{then} \quad \mathcal{E}_n\{L_i[n]\} = \mathcal{E}_n\{d_i[n]\}$$

where c is a constant. Let's write the recursive equation 2.2 of the membrane potential, sum them to deduce the expectation equation of input/ output and let $\alpha = \frac{t_S I}{C}$, then:

$$V[n] = V[n-1] - \alpha(L[n] - d[n]) \tag{2.4}$$

$$V[n-1] = V[n-2] - \alpha(L[n-1] - d[n-1]) \tag{2.5}$$

$$V[n-2] = V[n-3] - \alpha(L[n-2] - d[n-2]) \tag{2.6}$$

Until

$$V[1] = V[0] - \alpha(L[1] - d[1]) \tag{2.7}$$

When these equations are added, divided by the number of samples N and then performing the $\lim_{N\to\infty}$ of both side of the equation, the resultant equation will be as following:

$$\lim_{N\to\infty}\Big[\frac{V[n]}{N}\Big] = \lim_{N\to\infty}\Big[\frac{V[0]}{N}\Big] - \alpha\Big[\frac{1}{N}\sum_{n=1}^{N} L[n] - \frac{1}{N}\sum_{n=1}^{N} d[n]\Big] \tag{2.8}$$

If $|V[n]|$ is bounded (if $|V[n]| \leq constant$), then the equation 2.8 is written as:

$$\lim_{N\to\infty}\Big[\frac{1}{N}\sum_{n=1}^{N} L[n]\Big] = \lim_{N\to\infty}\Big[\frac{1}{N}\sum_{n=1}^{N} d[n]\Big] \tag{2.9}$$

Now, Let us return to equation 2.4: $V[n] = V[n-1] - \alpha(L[n] - d[n])$ where the output d_n is equal to:

$$d[n] = 0.5[sgn(V_{th} - V[n-1]) + 1] \tag{2.10}$$

Let $V'[n] = V_{th} - V[n]$ and $V'[n-1] = V_{th} - V[n-1]$. If the above equations are substituted in equation 2.4 and 2.10 then the following equations are introduced as,

$$d[n] = 0.5[sgn(V'[n-1]) + 1] \tag{2.11}$$

and

$$V'[n] = V'[n-1] + \alpha(L[n] - d[n]) \tag{2.12}$$

Now let

$$d'[n] = sgn[V'[n-1]] \tag{2.13}$$

Substitute equation 2.13 in 2.11

$$d[n] = 0.5[d'[n] + 1] \tag{2.14}$$

Substitute equation 2.14 in 2.12

$$V'[n] = V'[n-1] + \alpha(L[n] - 0.5d'[n] - 0.5) \tag{2.15}$$

$$2V'[n] = 2V'[n-1] + \alpha(2L[n] - 1 - d'[n]) \tag{2.16}$$

Now Let

$L'[n] = 2L[n] - 1$, $V''[n] = 2V'[n]$ and

$$V''[n-1] = V'[n-1] \quad \text{then} \tag{2.17}$$

$$V''[n] = 2V''[n-1] + \alpha(L'[n] - d'[n]) \tag{2.18}$$

Now substitute 2.13 and 2.17 to find $d'[n]$ such that $d'[n] = sgn[V''[n-1]]$ too.

$$V''[n] = V''[n-1] + \alpha(L'[n] - sgn(V''[n-1]))$$

If $V''[n-1] > 0$ then

$$V''[n] = V''[n-1] + \alpha(L'[n] - 1)$$

$$\text{and } V''[n] \leq V''[n-1]$$

if

$$V''[n-1] \leq c$$

$$\text{then } V''[n] \leq V''[n-1] \leq c$$

and if $V''[n-1] < 0$

$$\text{then } V''[n] = V''[n-1] + \alpha(L'[n] + 1)$$

$$\text{and } V''[n] \geq V''[n-1]$$

and if

$$V''[n-1] \geq c$$

$$\text{then } V''[n] \geq V''[n-1] \geq c$$

Therefore $V[n]$ is bounded since $|V[n]| \leq c$.

and hence,

$$\mathcal{E}_n\{L[n]\} = \mathcal{E}_n\{d[n]\}$$

2.2 Spike time dependency plasticity

In this section, the designed model of spiking IF neuron is used as a computational model itself to change the weight of synapses. The mechanism of changing the strength is called "the spike time dependency plasticity" (STDP) which is implemented by changing the amount of the injected current into the neuron depending on the time difference between the pre and post pulses of the neurons. The change of the injected current depends on previous weight of the synapse that must be stored as reference. For that purpose, the change of weight is implemented using a storing mechanism using floating gate MOSFETs. The concept of synapse plasticity and a novel mechanism of changing the plasticity using floating gate transistors will be discussed next.

The STDP is a process that adjusts the strength of connections between neurons in the brain. The adjustment of the connectivity strength is based on the relative timing of a particular neuron's output and input action potentials (or spikes). The STDP process is a tentative candidate for a hypothesis that partially explains the development of an individual's brain, especially with regards to long-term potentiation and long-term depression [52, 53].

While different models have been reported for describing STDP, all of them agree on the causal relationship between the pre- and post-synaptic spikes. The increase or decrease in the strength of the synapse (W) depends on whether the pre-synaptic spike arrives before or after the post-synaptic spike and the time-duration between the pre and post synaptic spike. If the pre-synaptic spike was generated at time t_j and the post-synaptic spike was generated at time t_i, then one form of STDP can be mathematically expressed as

$$\Delta W_{ij}(\Delta t) \propto \mathrm{sgn}(\Delta t).A.e^{|\Delta t|/\tau}, \tag{2.19}$$

where $\Delta t = t_j - t_i$, sgn(.) denotes the sign of Δt, while the parameters A and τ are the amplitude and the time constant of synapse potential respectively. Depending on the retention time of the synapse, the increase in the weight value is referred to the short-term potentiation (STP) or long-term potentiation (LTP). On the other hand, the decrease of the weight value is referred to the short-term depression (STD) or long-term depression (LTD). Fig. 2.3 and 2.6 (a) show the behavior of changing the weight corresponding to the difference between the time difference between the pre and the post pulses.

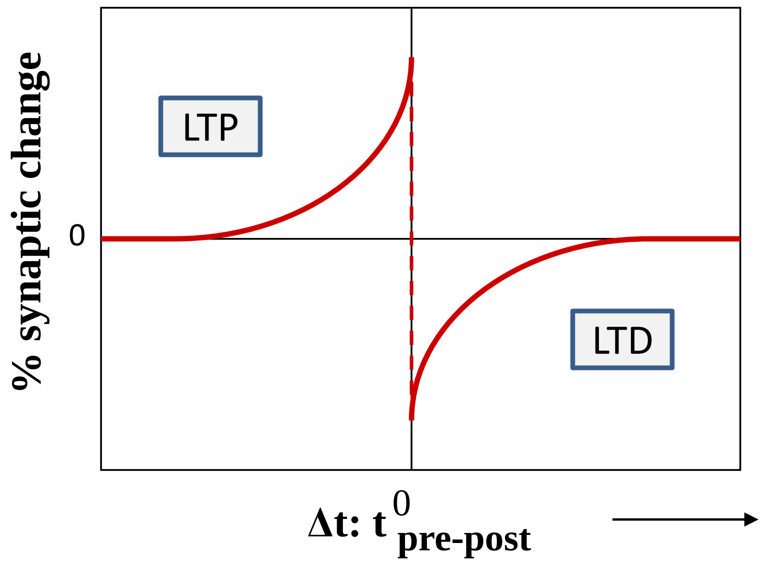


Figure 2.3: The STDP modification function. The synaptic change with ΔT, where $\Delta T = t_{pre} - t_{post}$

In this section, it is shown a novel synapse circuit to change the strength of synaptic connectivity using a nonvolatile memory of the floating gate type. Before introducing the synapse circuit, an overview of the floating gate will be explained below.

The floating-gate MOSFET (FGMOS) is a field-effect transistor, whose structure is similar to a conventional MOSFET. The FGMOS is used as storage element in digital EPROM and flash memories or in analog elements in neuronal computational networks [54]. The gate of the FGMOS is electrically isolated, creating a floating node. A simplified implementation of the conventional FG memory cell is shown in Fig. 2.4, where the gate of a PMOS transistor M1 is used for storing charge. Because the gate is completely insulated by high quality silicon-dioxide, any charge trapped on the gate is retained for a long period of time (8 bits retention accuracy for about 8 years) [55, 56]. The charge on the floating-gate can be modified either by inducing a large electric field at the drain of the transistor for hot electron injection or by Fowler-Nordheim (FN) tunneling of electrons through a parasitic capacitor C_{tun} [57]. The total amount of charge on the floating-gate and the magnitude of the applied control gate voltage V_{cg} determines the floating-gate voltage V_{fg}. The drain current of M1 which is a function of V_{fg} can be copied and used for biasing or for auxiliary analog signal processing functions (shown in Fig.2.4).

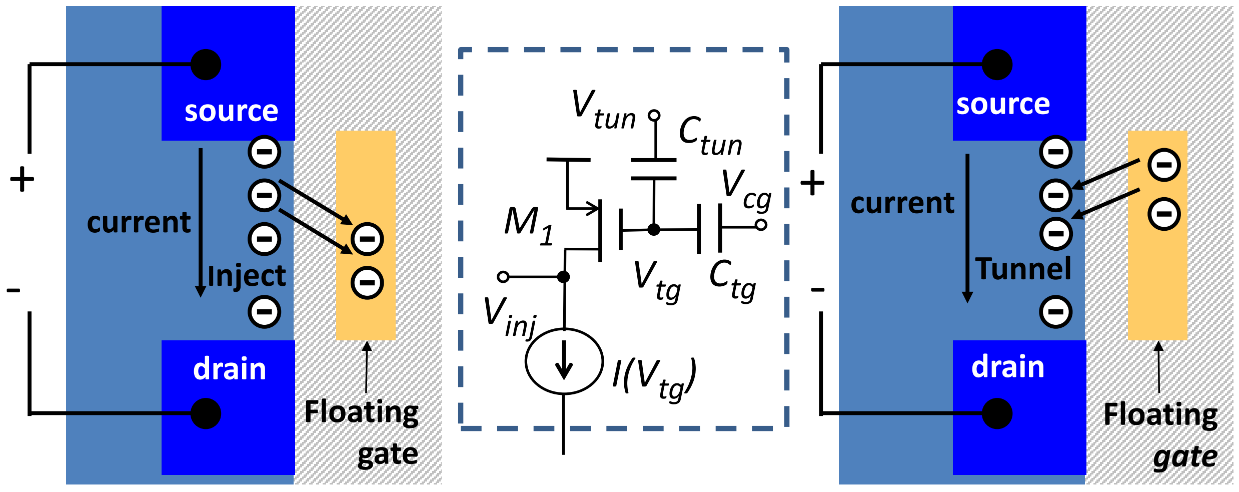


Figure 2.4: Floating gate memory

Usually Fowler-Nordheim tunneling and hot-carrier injection mechanisms are used to modify the amount of charge stored in the FG [58]. The charge on the floating-gate is modified using hot-electron injection voltage ($4.2V$ in $0.5\mu m$ process) and tunneling voltage ($15V$ in $0.5\mu m$ process).

Since the first reported floating-gate structure in 1967 [59], floating- gate transistors have been used widely to store digital information for long periods in structures such as EPROMs and EEPROMs. Recently, floating-gate devices have found applications as analog memories, analog and digital circuit elements, and neuronal computational networks as adaptive processing elements. In neuronal networks, the choice of changing the weight in the learning rule is the (STDP) rule. Recent implementations of STDP learning in VLSI spiking networks demonstrate the use of these networks in classification and computational tasks [60, 61, 62]. However, the slowest weight update rate and time constant in most of these implementations is limited to the transistor leakage currents in the chosen fabrication process. To obtain longer update time constants for the synaptic weights, the use of floating-gate technology to update the weight is described as another implementation of the STDP learning rule. The synaptic weight is set by the current flowing through a pFET transistor whose gate is a floating node. The charge on the floating node is removed or added using tunneling and injection. Two main reasons to use the floating gates in STDP learning mechanism in the neuronal networks; the storage facility of the weight and good energy efficiency [53, 54, 57, 63, 64, 65].

Toward the above goals, a synapse design is suggested for the STDP learning rule. The circuit uses the floating gate as a storage device for the weight. The adjustment of the weight in the designed circuit is implemented in a different way such that the injection and the tunneling processes change the current to the IF neuron discussed earlier. The adjustment of weight is implemented in an instantaneous way by applying the following pulses: pre synapse pulse, post synapse pulse, pre shaped synapse pulse and post-shaped synapse pulse. The pre synapse pulse, post synapse pulse are input/ output pulses of a neuron respectively. The pre shape pulses and post synapse pulses impose a delay to the incoming signals in order to allow weight modification before losing the occurrence of these pulses. The detail of the designed circuit is listed below, the circuit description and the simulation result of each part is demonstrated. The synapse design is fabricated on (1.5mm * 1.5mm) chip as an array of 12 synapse * 8 neurons and also on (3mm*3mm) chip as an array of 25 synapse * 30 neurons. The connectivity between the neurons on the fabricated chip can be implemented using a look up table and state machine program configured in field-programmable gate array (FPGA). The role of FPGA is to process the output of the chip and route the pulses between the rows and columns of synapses which are connected directly as a column vector to the neurons.

Fig 2.5 is the architecture for the STDP learning that include the following units:

1. Synapse unit (S): it represents the junction between neurons. The inputs to the synapse unit are the pre pulse signal, post pulse signal, pre shape signal and post shape signals. The output of the synapse is the current applied to the neuron. The current here is either to increase or decrease the pulse rate of IF neuron. Note that more than one synapse is connected to the input of neurons. The concept here is that the pulses are applied in time multiplexing using a high speed FPGA for generating pulses as if the pulses are generated simultaneously. This is the concept of distributed parallel processing that is used in neuromprphic design.

2. Neuron unit (N): It represents the IF unit discussed in the previous section.

3. Post shape pulse unit: it receives the neuron output pulses and introduces a delay so that the synapse unit keeps track of timing sequences between the post pulse and

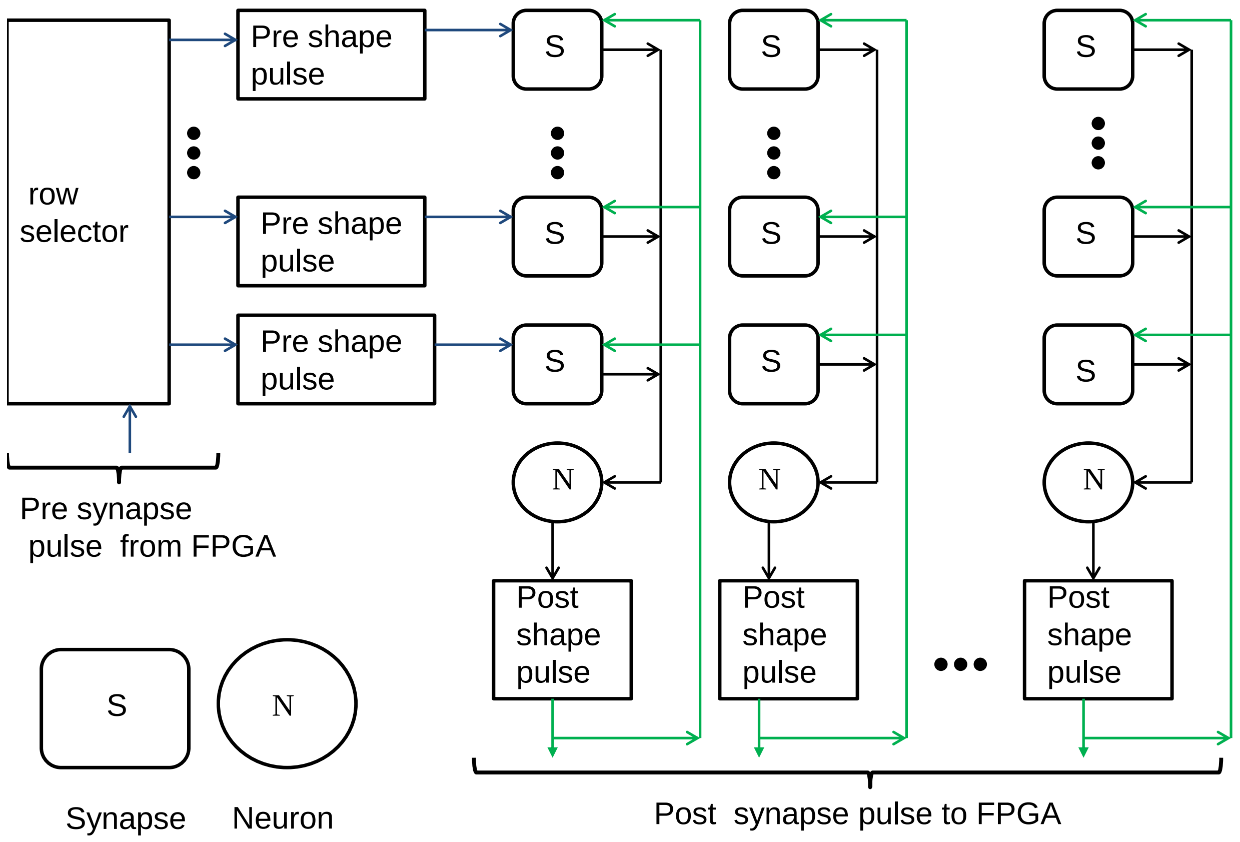


Figure 2.5: STDP learning using neuronal architecture showing the connectivity between the synapses, neurons, post shape, pre shape and shift register modules

pre pulse. This Process is necessary for the tunneling phase in the FG because the tunneling happens when the post pulse come before the pre pulse. Therefore, the post pulse has the chance to stay a longer time than the pulse itself so that the difference voltage between the pre pulse and post shape signal is high enough for tunnelling. The duration of tunnelling phase and the amount of tunnelling voltage decide how much increment in the floating gate voltage that deceases the current applied to the neuron and eventually decrease the pulse rate. The change of the weight depends on the time difference between t_{pre} and t_{post} shown in the LTD part of the Fig. 2.6 (a). In the tunneling mode of the floating gate, the required voltage VT to transfer the electron

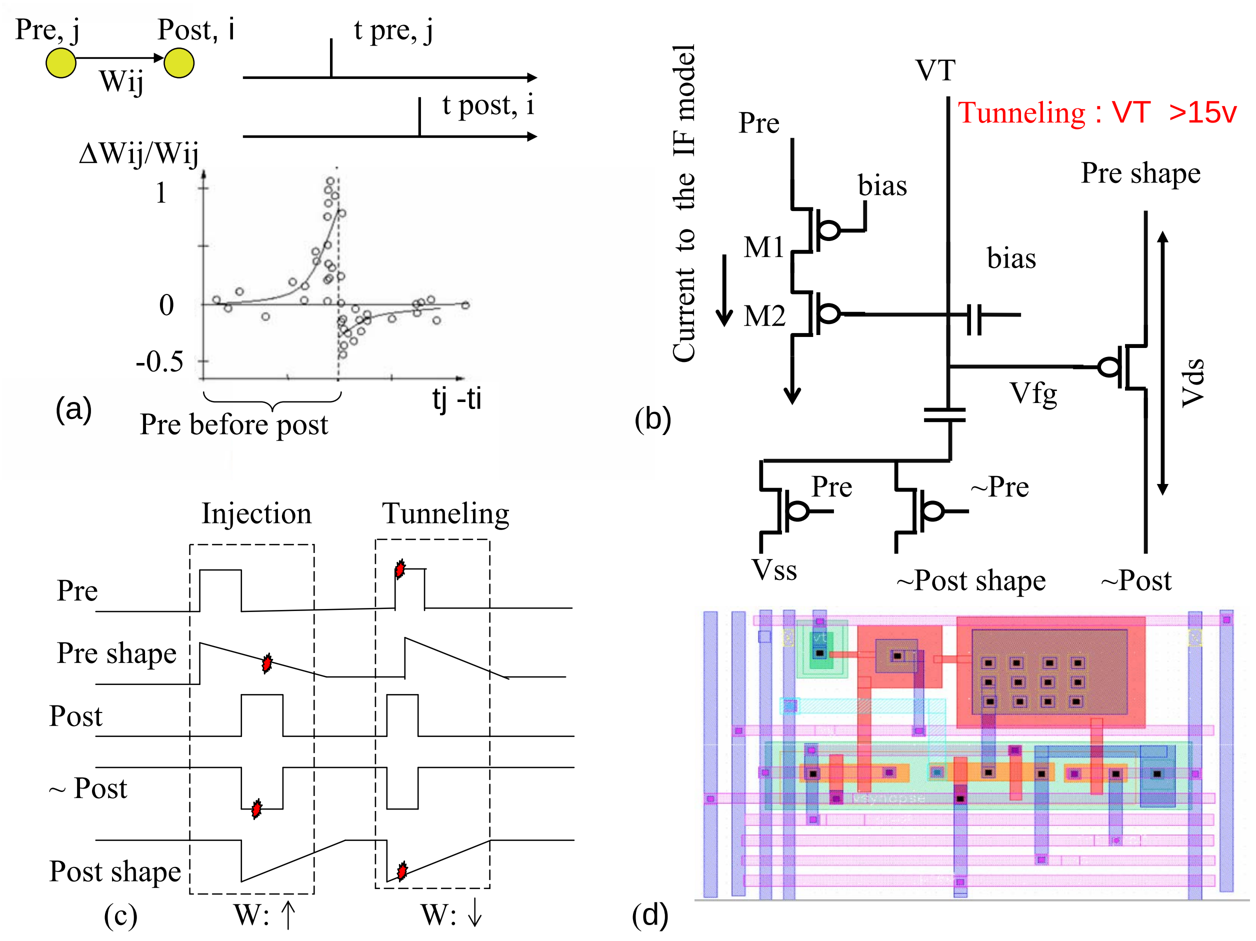


Figure 2.6: (a): Learning curve of STDP as the change in the weight versus the time interval between t_{pre} and t_{post}. (b): The synapse circuit shows the injection and tunneling needed in the floating gate to adjust the current in pMOS $M1$ and $M2$. (c): Timing sequence for the injection and tunneling in the synapse -floating gate of (b). (d): The Layout of synapse of (b)

out of the gate, where VT is $> 15v$. The post shape circuit, the layout and a graphic simulation of the shaped circuit is shown in Fig. 2.7

4. Row selector: It represents the shift register to enable and select the rows to input the pre pulse from the FPGA into pre shape stage.

5. Pre shape pulse unit: It includes first a level shifter to raise the voltage level to a required level (from 3.3 v to 6 v) to validate the required voltages for injection and tunnelling of electrons (in 0.5μm process), hot-electron injection requires a voltage

difference of $4.2v$ between the source and drain. This unit is used to introduce a delay time to the pre pulse in order to keep track of the timing sequence between the pre pulse and the post pulse. This Process is necessary for the injection phase in the FG because the injection happens when the pre pulse comes before the post pulse. Therefore, the pre pulse has the chance to stay longer time than the pulse it self so that the difference voltage between the pre pulse and post shape signal is high enough for injection. The duration of injection phase and the amount of injection voltage decide how much decrements in the the floating gate voltage that increase the current applied to the neuron and eventually increase the pulse rate. Also, the change of the weight depends on the time difference between t_{pre} and t_{post} in the potentiation of weight (”pre before post” part shown in Fig. 2.6 (a)). The pre shape circuit, the layout and a graphic simulation of the shaped circuit are shown in Fig. 2.8.

An example of circuit simulation for STDP design is shown in Fig. 2.9. In this experiment ten synapse circuits are connected to one IF neuron model. In the simulation result, it is shown that the floating gate voltage (v_{fg}) decreases whenever there is a pre pulse *pre* at which the integration step occurs. Notice that as the v_{fg} decrease more ,the current increases in the PMOS transistor M_1 and M_2 in Fig. 2.6 (b). Thus the integration of the current in the IF capacitor increases and causes the membrane voltage of the integrator to increase too. That will cause the membrane potential to reach the threshold level earlier and generate output pulse earlier too. Hence the time between the generating output pulses by the IF will decreases (lower time intervals between the post pulses ”*post*”) and in other word, the pulse rate of the IF will increase. In the figure, it is shown two time intervals, the first one is $70ms$ where $t_{pre} - t_{post}$ is less than that in the second interval $100ms$. This means that the

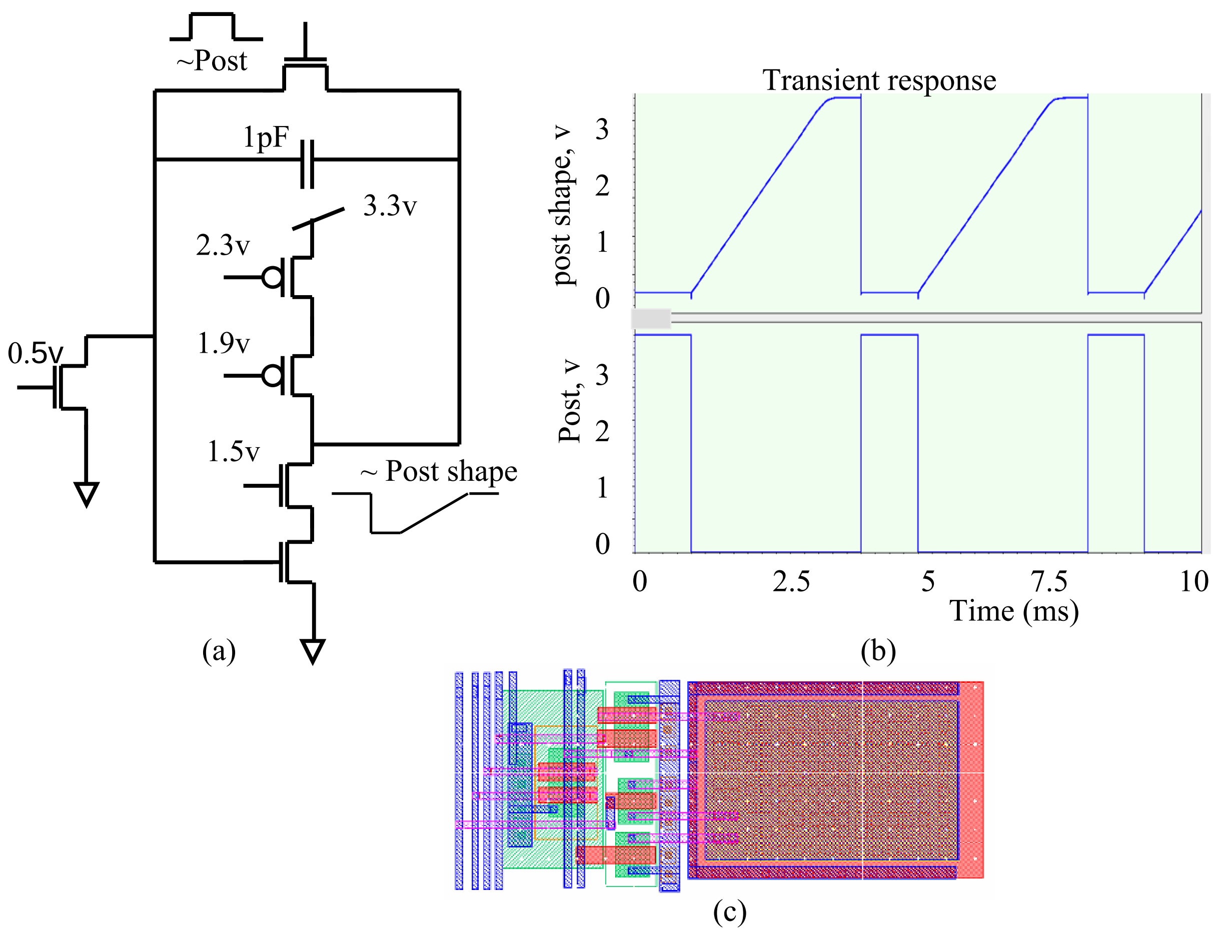


Figure 2.7: Post pulse shape circuit

integration occurs earlier in the first case and hence the time to reach the threshold to spike is shorter and thus the rate is faster. Note of the first case has almost simultaneous intervals "*preshape*" and "*post*" pulses which realize the injection mode of the floating gate while in the second interval the *post* pulse comes before the *pre* pulse which realize the tunnel mode of floating gate. The integration in the membrane capacitor v_{mem} is shown correspondingly in the same figure.

The layout for the fabricated chips are shown in Fig.2.10 for 12 synapses * 8 neuron and for 25 synapses * 30 neuron. The above implementation is to show the computation power of the IF-synapse module in the learning field at the level of one neuron. The synapse

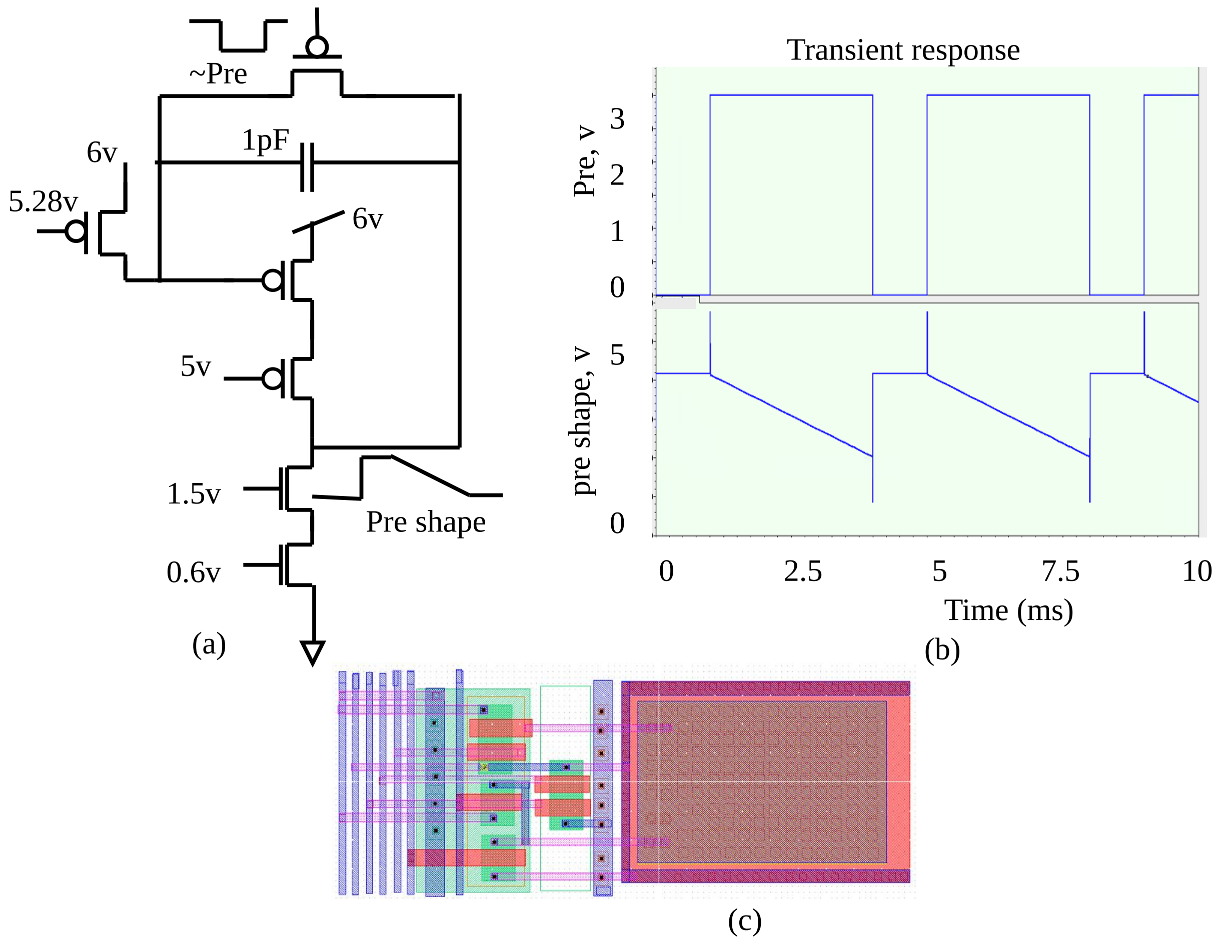


Figure 2.8: Pre pulse shape circuit

unit and the IF structure modify the weight by finding the time difference between post and pre pulses, decide the change of the weight which is translated to a certain amount of an injected current and modifies the membrane potential in the IF unit. However, the computation complexity increases in a population of the neuronal network. In the next chapter, a network of IF units is designed as a basic novel pulse based computational unit to encode, compute, recognize input events, map and implement non linear functions.

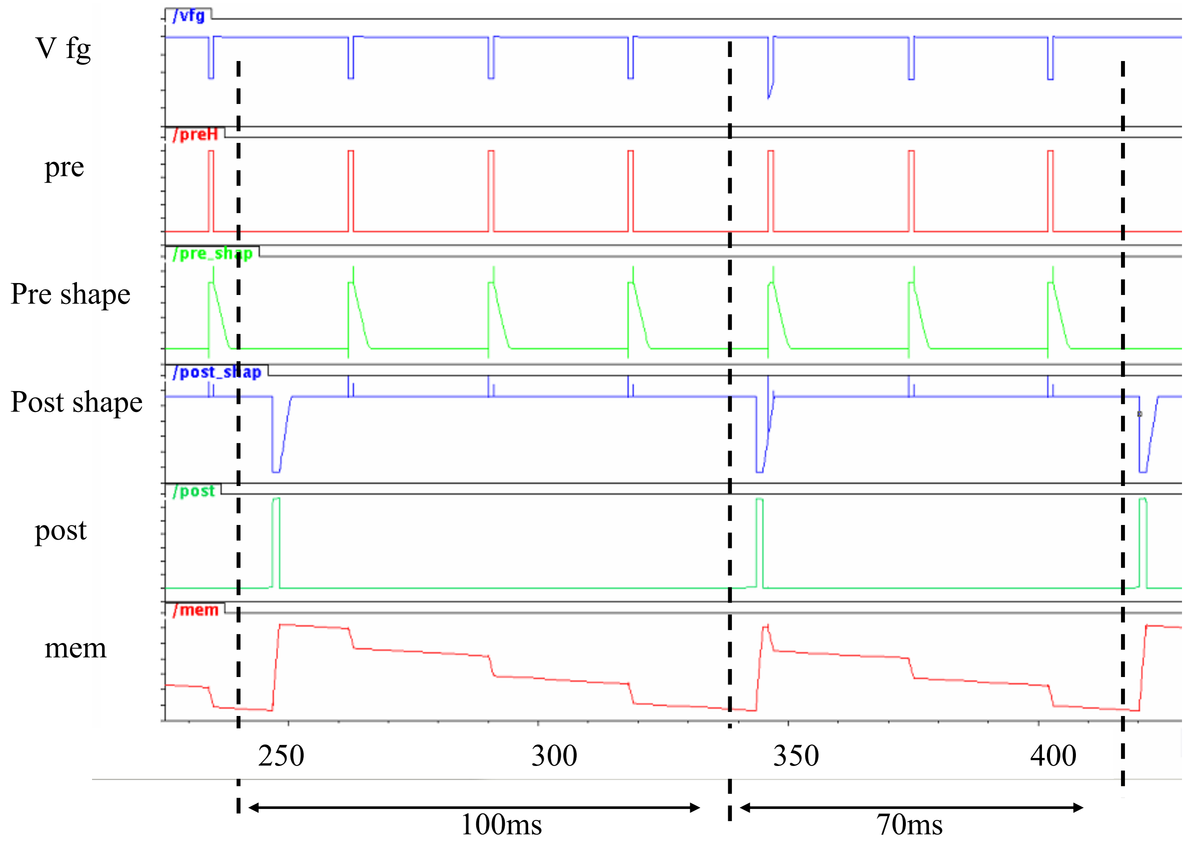
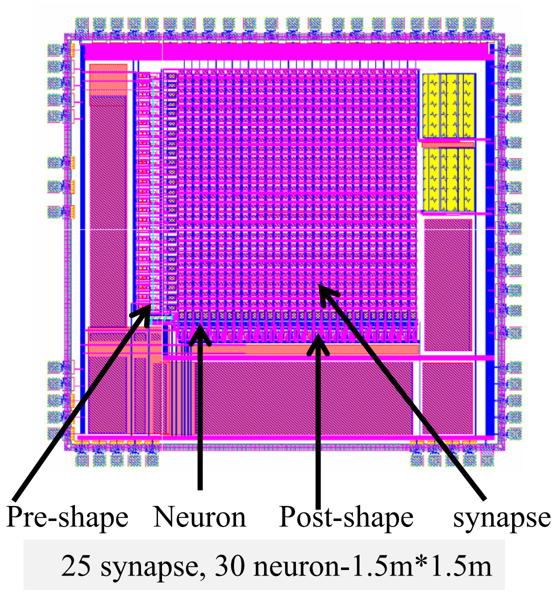


Figure 2.9: Transient simulation of the STDP circuit

2.3 Summary

In this chapter, a new model of the integrate and fire neuron is designed. The model is applied to change the synapse weight using a storage transistor " floating gate" using a novel synapse circuit. The applied example discussed in the chapter is to show the computational capability of the integrate and fire model. Yet, the integrate and fire model is limited in computing other functions. Therefore, the integrate and fire model is reconsidered to verify the convergence capability in order to use this model as the core of the basic designed pulse computational algorithmic model in this thesis. The convergence capability for the integrate and fire neuron means that the expected rate of the output pulse approach the expected rate



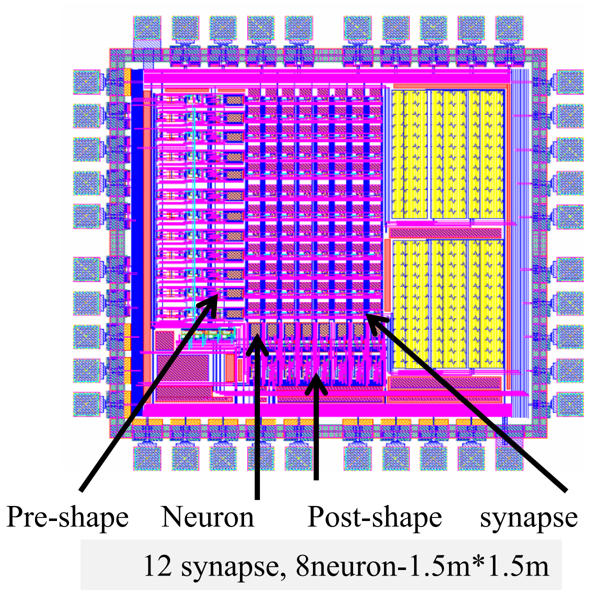


Figure 2.10: Layout of the STDP architecture. The two chips are identical but are different in density

of the input pulses. The applied pulses is stochastic where the input/ output parameters are quantified by the average number one's to the time period.

Chapter 3

IFMP

In this chapter, a scalable pulse computational module using Integrate - Fire structure and Margin Propagation (IFMP), as the main topic of the thesis, will be discussed in details. Section 3.1 includes a description of the dynamic performance, the analysis of different parameters in the module and the cascading concept of the IFMP. Section 3.2 includes the proofs and simulations of the margin propagation properties. Section 3.3 demonstrates examples of the basic computations using IFMP including the addition, multiplication, division, power, polynomial, and inner product functions. Section 3.4 shows the analog circuit of IFMP module and the measurement test of the IFMP chip. The final section summaries the chapter.

3.1 IFMP: concepts and analysis

The strategy of processing information and the connectivity between neurons in a population of neurons in the brain is not known yet. There are different types of connectivity schemes between neurons in the population of spiking neural networks. Some of these neurons act as an excitatory and others as inhibitory neurons. If the excitatory/inhibitory neurons are connected to a particular neuron in the population through synapse, then the spike rate of that neuron will increase/decrease accordingly. The proposed model is inspired from the neuronal computation of the receptive field in the biological sensory system. In the receptive field, the afferent neurons are excited by the external stimuli. The excited spot of the input stimulus causes high spike rate generated by the afferent neurons of the receptive field. These neurons inhibit the surrounded afferent neurons through inhibitory synapses in order to maximize its effect and transmit the spikes to the higher neuronal level as shown in the neurobiology sensory system figure 3.1 [5]. This observation matched with the log-sum-exp function discussed earlier which maximize the input value.

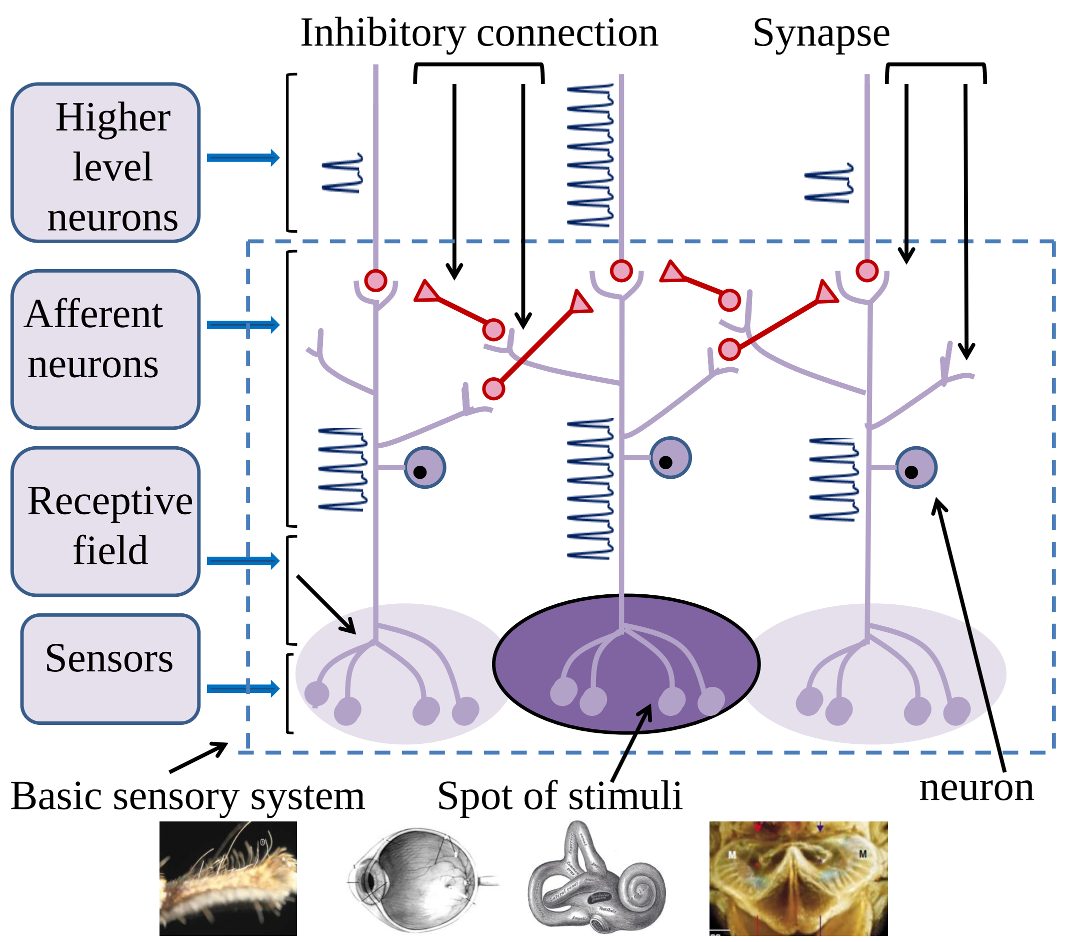


Figure 3.1: Organization of a generic neurobiological sensory system, image source: Wikipedia and Ref. [5, 66]

It is well known that the simplest form of neural coding is the rate-based encoding [11] which computes the instantaneous spiking-rate of the i^{th} neuron $R_i(t)$ according to

$$R_i(t) = \frac{1}{T}\int_t^{t+T} \rho_i(t)dt, \tag{3.1}$$

where $\rho_i(t)$ denotes the spike-train generated by the i^{th} neuron using a sequence of time-shifted Kronecker-delta functions as

$$\rho(t) = \sum_{m=1}^{\infty} \delta(t - t_m) \tag{3.2}$$

where $\delta(t) = 0$ for $t \neq 0$ and $\int_{-\infty}^{+\infty} \delta(\tau)d\tau = 1$. t_m is the firing time of the spike or the delta pulse.

Since pulse rate is one way to encode information, a pulsed neuronal model is proposed to implement pulse rate computation with a novel strategy of excitatory and inhibitory neurons [67]. In this proposal, an architecture is presented which uses the integrate-and-fire structure to encode the analog input currents passing through switches controlled by a stream of input pulses. These neuronal structures are connected with lateral inhibition which prevents them from spiking simultaneously, and has the effect of distributing their sampling uniformly in time. The new structure has N feed back inhibition which is less than the full connection of N(N-1) inhibition connections used for example in [68].

Fig. 3.2 shows a schematic diagram of the proposed spiking network. The network is referred to as an IFMP model and consists of three integrate-and-fire structures N_1, N_2 and N_3. The excitatory/inhibitory inputs are represented by black (white) triangles and only modules N_1 and N_2 have self-inhibitory feedback connections. Given the rate of input spike-trains $L_1[n]$ and $L_2[n]$ with n being a discrete time-index, it can be shown that firing-rate of

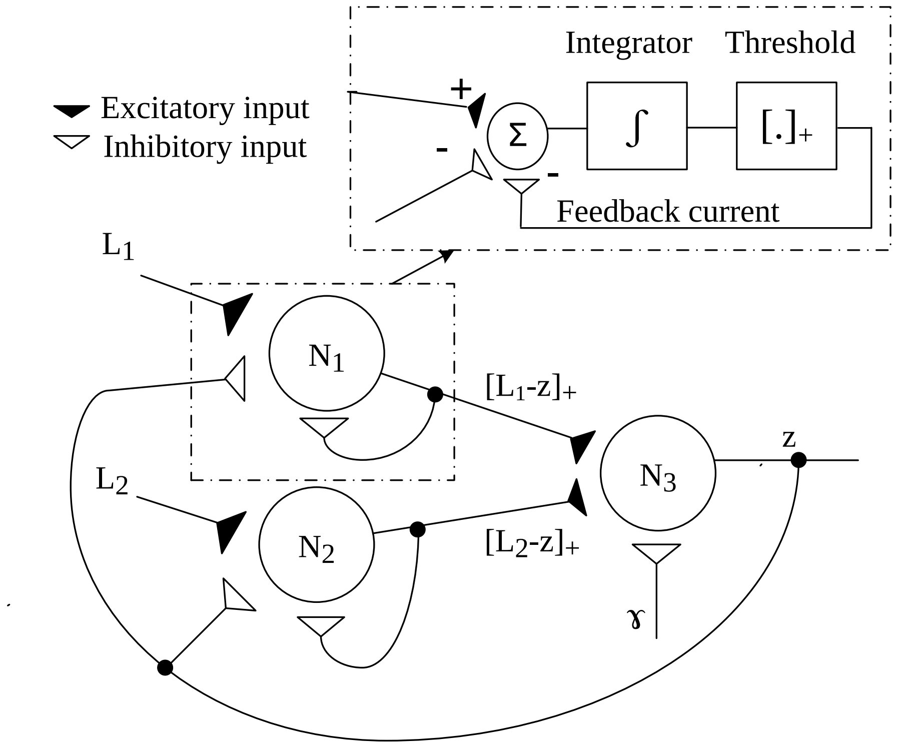


Figure 3.2: Schematic of the proposed IFMP unit comprising of three integrate-and-fire modules

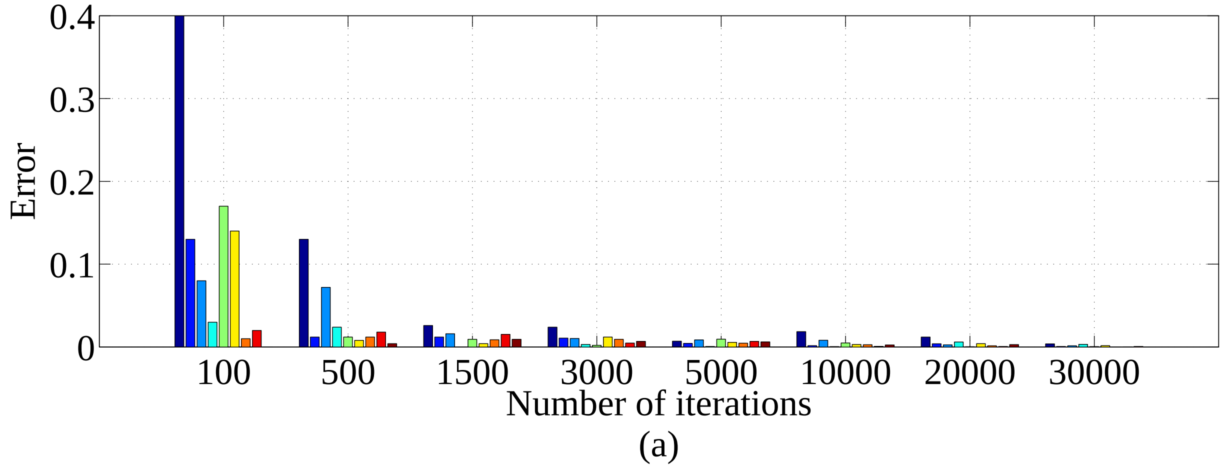
the output $L_z[n]$ (denoted by $\mathcal{E}(L_z) = \lim_{T\to\infty} \frac{1}{T}\sum_{n=1}^{T} L_z[n]$) asymptotically satisfies the following equation,

$$[\mathcal{E}(L_1[n]) - \mathcal{E}(L_z[n])]_+ + [\mathcal{E}(L_2[n]) - \mathcal{E}(L_z[n])]_+ \longrightarrow \mathcal{E}(\gamma[n]) \tag{3.3}$$

and in general for p inputs,

$$\sum_{i=1}^{P} [\mathcal{E}(L_i[n]) - \mathcal{E}(L_z[n])]_+ \longrightarrow \mathcal{E}(\gamma[n]) \tag{3.4}$$

Note that the equation 3.4 converges only in probability. The difference between the left



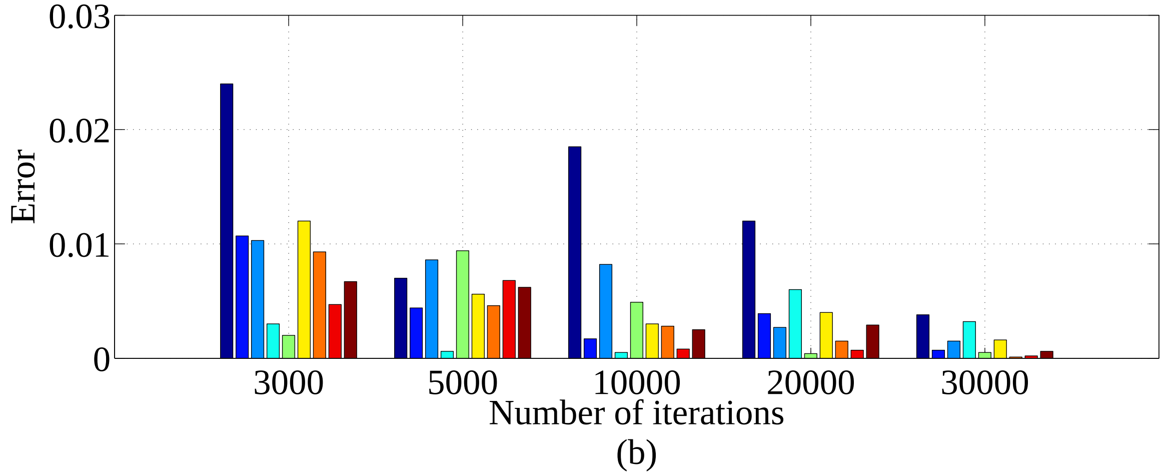


Figure 3.3: a: The error bar between the theoretical and practical rates with respect to iterations for iterations greater than 3000. b: Error bars for iterations greater than 100

and right hand side of the above equation decreases as the time increases (or the number of stream sequence of the random inputs increases denoted as n). Hence the summation of the expected values of the input stream converges to the expected values of the output stream. Fig. 3.3-a shows the absolute error bars versus the number of iteration for the input stream where the error is calculated as the difference between the theoretical and the simulated output rate. In the figure, it is shown that the absolute error bars decrease with iteration number starting from 100 to 30,000. The nine bars for each iteration represent the error when the input rate for one input changes from [0.1:0.9] where the bound for the absolute error is less than 0.003. In Fig. 3.3-b, it is shown a zoomed out picture for iterations starting from 3,000 to 30,000 and how is the error decreasing significantly..

Fig.3.4 shows the plot of instantaneous spiking-rates for N_1,N_2 and N_3, when input rate of the inputs are varied as shown in Fig. 3.4 (below). In this experiment, $\gamma = 0.3$ and the input rate $L_2 = 0.5$ for N_2 while input rate L_1 for N_1 increase from 0 to 1. The dynamic of the figure follows the IFMP equation 3.4 such that $[L_1 - z]_+ + [L_2 - z]_+ = \gamma$. Initially, when L_1 is between 0 and 0.3, then the output rate of N_1, N_2 and N_3 is equal to 0, 0.3 and 0.2 respectively. When L_1 is 0.35, then the output rate of units N_1, N_2 and N_3 are $0.05, 0.25$ and 0.25 respectively. When L_1 is 0.6, then the output rate of units N_1, N_2 and N_3 are $0.2, 0.1$ and 0.4 respectively and so on the sum of the output rates for the first two IF units convergence to a rate equal γ all the time in which the dynamics of IFMP satisfies equation 3.4 as shown in Fig. 3.4.

To prove the convergence of equation 3.4 above, it is necessary to prove initially the convergence of each neuron and then the convergence of the IFMP network. The convergence of one neuron was proven in chapter 2. The convergence equation for the IFMP in Fig. 3.2 is listed as, $\sum_{i=1}^{P} [\mathcal{E}(L_i[n]) - \mathcal{E}(L_z[n])]_+ \longrightarrow \mathcal{E}(\gamma[n])$ or

$$[\mathcal{E}_n\{T_1[n]\} + \mathcal{E}_n\{T_2[n]\}] = \mathcal{E}\{\gamma[n]\} \tag{3.5}$$

where, $T_1[n] = \mathcal{E}(L_1[n]) - \mathcal{E}(L_z[n])_+$ and $T_2[n] = \mathcal{E}(L_2[n]) - \mathcal{E}(L_z[n])_+$

The equation for the membrane voltage for neurons 1, 2 and 3 can be written as,

$$V_1\,[n] = V_1\,[n-1] - \alpha(L_1[n] - T_1[n] - z[n]) \tag{3.6}$$

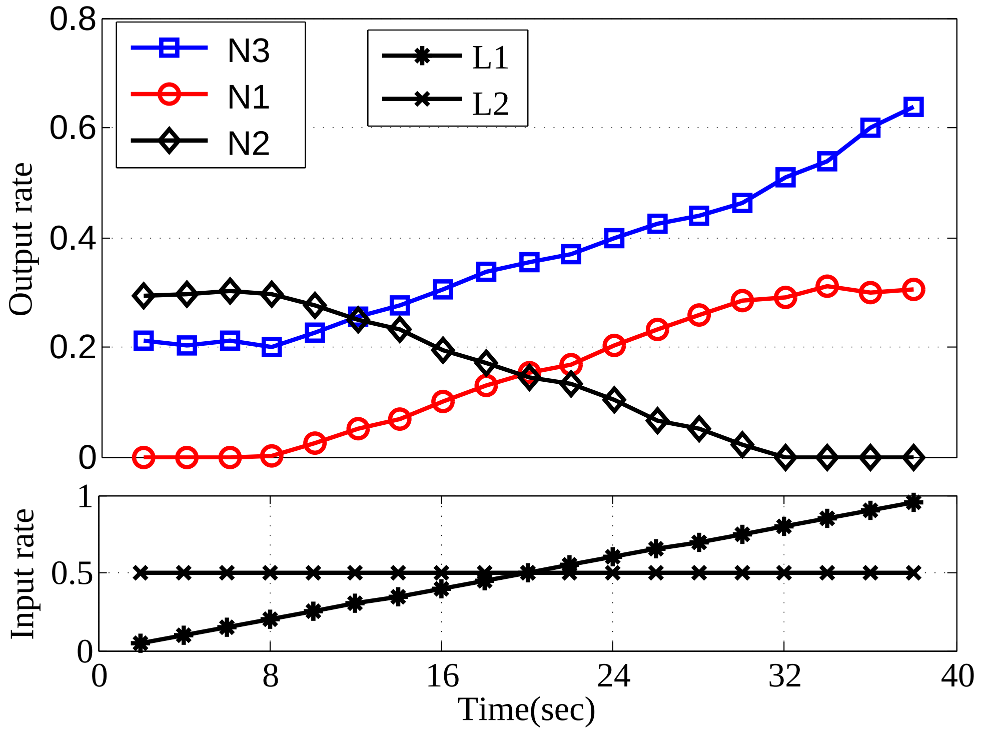


Figure 3.4: Spike-rates for neurons N_1, N_2 and N_3 when the spike-rate of L_1 is monotonically increased. For interpretation of references to color in this figure and all other figures, the reader referred to the electronic version of this thesis

$$V_2\,[n] = V_2\,[n-1] - \alpha(L_2[n] - T_2[n] - z[n]) \tag{3.7}$$

$$V_3\,[n] = V_3\,[n-1] - \alpha(T_1[n] + T_2[n] - \gamma[n]) \tag{3.8}$$

The membrane potential $V_3\,[n]$ in the IFMP is designed to be bounded between threshold voltage and restart voltage (initial voltage). Let T_1, T_2, and z be the output spike rate for structures N_1, N_2 and N_3 respectively and γ is the inhibitory input rate for unit N_3. Then resumes proving the convergence by writing recursively the membrane equation for N_1 starting from the first recursion as follows,

$$V_1\,[n-1] = V_1\,[n-2] - \alpha(L_1[n-1] - T_1[n-1] - z[n-1]) \tag{3.9}$$

$$V_1\,[n-2] = V_1\,[n-3] - \alpha(L_1[n-2] - T_1[n-2] - z[n-2]) \tag{3.10}$$

Writing the recursive equations for the previous step times will get eventually the initial recursive equation as follows,

$$V_1\,[1] = V_1\,[0] - \alpha(L_1[1] - T_1[1] - z[1]) \tag{3.11}$$

Next, adding all the above recursive equations and divide them by N will have the followings,

$$\frac{V_1\,[n]}{N} = \frac{V_1[0]}{N} - \alpha(\frac{1}{N}\sum_{i=1}^{N} L_1[i] - \frac{1}{N}\sum_{i=1}^{N} T_1[i] - \frac{1}{N}\sum_{i=1}^{N} z[i]) \tag{3.12}$$

since $|V_1[n]|$ is bounded, then the $\lim_{N\to\infty}$ of the above equation becomes as,

$$[\mathcal{E}\{L_1[n]\} - \mathcal{E}_n\{z[n]\}] = \mathcal{E}_n\{T_1[n]\} \tag{3.13}$$

Similarly for N_2,

$$[\mathcal{E}_n\{L_2[n]\} - \mathcal{E}_n\{z[n]\}] = \mathcal{E}_n\{T_2[n]\} \tag{3.14}$$

For N_3, the membrane potential is listed below:

$$V_3\,[n] = V_3\,[n-1] - \alpha(T_1[n] + T_2[n] - \gamma[n]) \tag{3.15}$$

Since V_3 is bounded, then the expected value of the output for neuron 3 will be as :

$$[\mathcal{E}_n\{T_1[n]\}] + \mathcal{E}_n\{T_2[n]\} = \mathcal{E}_n\{\gamma[n]\} \tag{3.16}$$

Now, substitute equations (3.13) and (3.14) in (3.16), then the convergence expectation equation is written as follows,

$$[\mathcal{E}_n\{L_1[n]\} - \mathcal{E}_n\{z[n]\}] + [\mathcal{E}_n\{L_2[n]\} - \mathcal{E}_n\{z[n]\}] = [\mathcal{E}_n\{\gamma[n]\}] \tag{3.17}$$

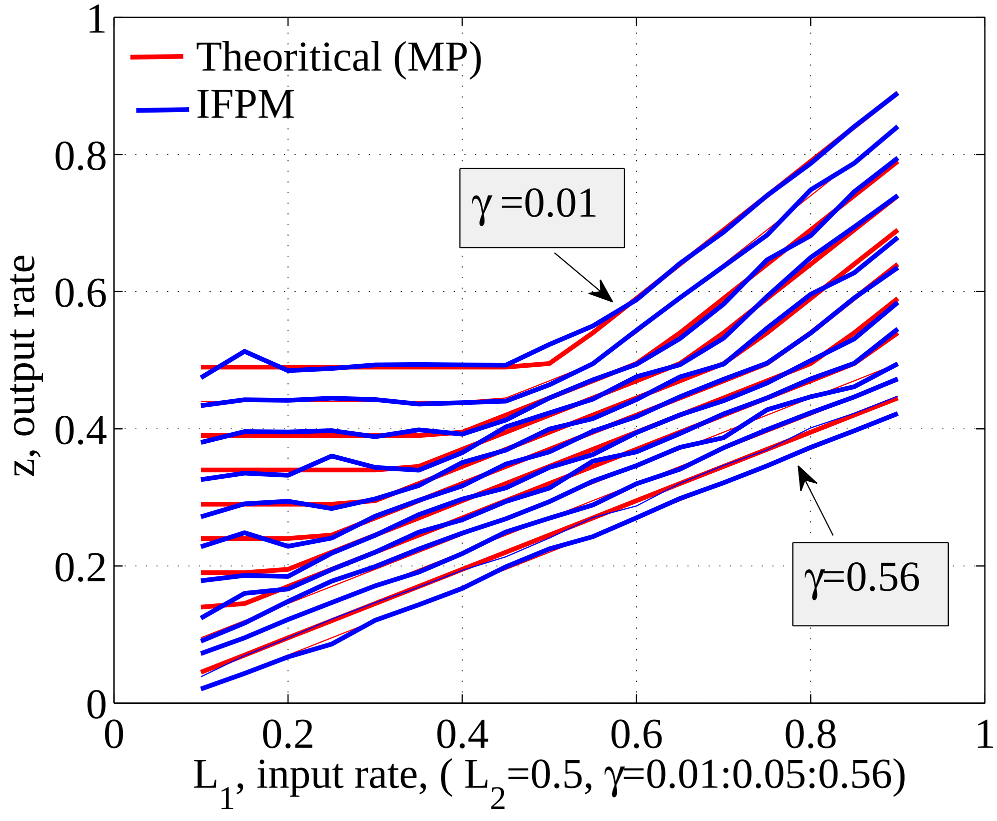


Figure 3.5: Dynamic characteristics of IFMP unit for different values of γ for MP and IFMP

Fig.3.5 shows the plot of instantaneous spiking-rates for N_3, when spiking-rate of the input rate L_1 is varied from 0.01 to 0.99. For this result, the spiking-rate for input L_2 is kept constant to 0.5 as γ changes from $0.01 : 0.05 : 0.56$. The plot shows that the spiking-rate of N_3 increases according to a piece-wise linear approximation to the margin propagation function.

It is proven in [40] that the margin propagation (MP) is an approximation to the log-sum-exp. But how much the approximation is and what are the parameters involved in the approximation? How successful is the cascading of the approximated model? To answer these questions, the followings are stated: Let z_{MP} is the output pulse rate for an IFMP unit of m inputs of inputs $L_1, L_2..L_m$ and let $z_{LSE} = log(\sum_i^m e^{L_i})$ is a solution to

$$\sum_i^m e^{[L_i - z_{LSE}]_+} = 1 \tag{3.18}$$

where $\{z_{LSE} : R^m \longrightarrow R\}$, then using the fact that

$$e^{[L_i - z_{LSE}]} \geq [1 + L_i - z_{LSE}]_+$$

then

$$\sum_i^m [1 + L_i - z_{LSE}]_+ \geq 1 \tag{3.19}$$

To make the normalization ideal, the above equation is equated to one,

$$\sum_i^m [1 + L_i - z_{LSE}]_+ = 1 \tag{3.20}$$

$$\sum_i^m [L_i - z_{MP}]_+ = 1 \tag{3.21}$$

where z_{MP} is the approximated MP value to the log -sum -exp as $z_{MP} = z_{LSE} - 1$

If the normalization factor changes to $z_{LSE}/\gamma = log(\sum_i^m e^{(L_i/\gamma)})$ as a solution to

$$\sum_i^m e^{[L_i - z_{LSE}]/\gamma} = 1 \tag{3.22}$$

MP is approximated as following, (see Fig. 3.7-a)

$$\sum_{i}^{m}[1 + (L_i - z_{LSE})/\gamma]_+ \geq 1 \tag{3.23}$$

$$\sum_{i}^{m}[\gamma + L_i - z_{LSE}]_+ \geq \gamma \tag{3.24}$$

$$\sum_{i}^{m}[\gamma + L_i - z_{LSE}]_+ = \gamma \tag{3.25}$$

$$\sum_{i}^{m}[L_i - z_{MP}]_+ = \gamma \tag{3.26}$$

such that $z_{MP} = z_{LSE} - \gamma$

Fig. 3.6 shows the above approximation which is equal to γ between z_{LSE} and z_{MP}, z_{IFMP} where $z_{IFMP} = z_{MP}$

The MP formulation can be mapped onto a cascaded topology by rewriting the LSE equation " $z_{LSE} = log(\sum_i^n e^{(L_i)})$ " in a recursive form as following,

$$z_{LSE}[n] = log(\sum_{i}^{n-1} e^{(L_i)} + e^{(L_n)}) \tag{3.27}$$

$$z_{LSE}[n] = log(e^{log(\sum_i^{n-1} e^{(L_i)})} + e^{(L_n)}) \tag{3.28}$$

$$z_{LSE}[n] = log(e^{(z_{LSE[n-1]})} + e^{(L_n)}) \tag{3.29}$$

Above is the equation for cascading the MP units. The margin of the first unit (Fig. 3.7-b) is represented as ,

$$[L_1 - z_{MP1}]_+ + [L_2 - z_{MP1}]_+ = \gamma \tag{3.30}$$

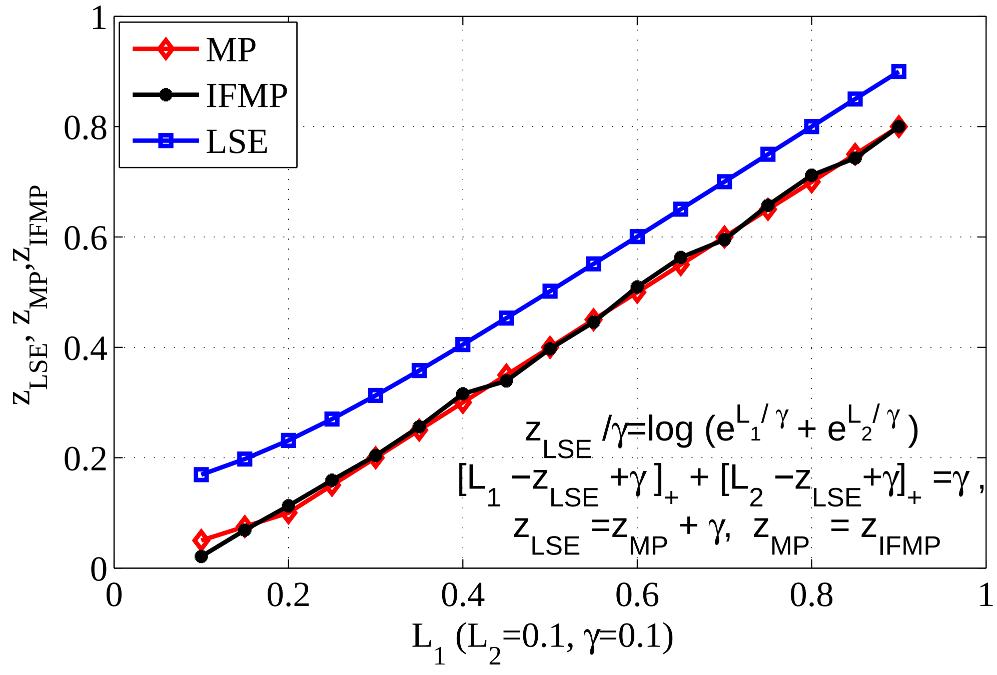


Figure 3.6: Approximation of rate z between the log-sum-exp rate (z_{LSE}), Margin Propagation rate (z_{MP}) and Integrate-Fire Marginn Propagation (z_{IFMP})

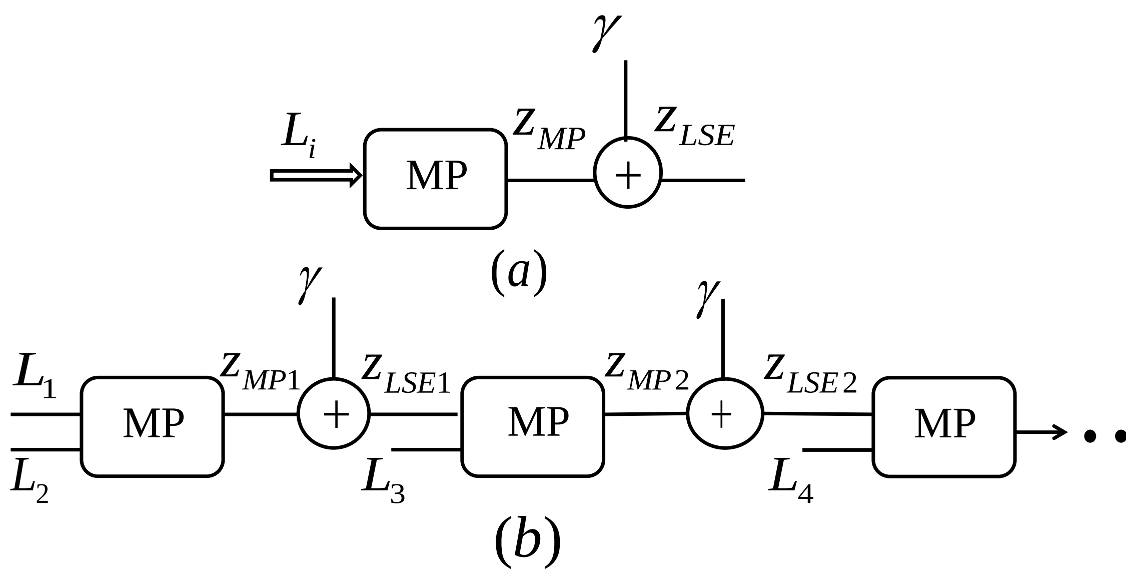


Figure 3.7: (a): Block diagram that shows the approximation between LSE and IFMP. (b): Cascading of IFPM structure

Let $L_2 = -\infty$ for the sake of clarity to show the cascading property. It was shown earlier that $z_{LSE} = z_{MP} + \gamma$ then for the second block,

$$[z_{LSE1} - z_{MP2}]_{+} + [L_2 - z_{MP2}]_{+} = \gamma \tag{3.31}$$

$$[z_{LSE1} - [z_{LSE2} - \gamma]]_{+} + [L_2 - [z_{LSE2} - \gamma]]_{+} = \gamma \tag{3.32}$$

$$[z_{LSE1} - z_{LSE2} + \gamma]_{+} + [L_2 - z_{LSE2} + \gamma]_{+} = \gamma \tag{3.33}$$

Denote $z_{LSE[n]}$ and $z_{LSE[n-1]}$ are equal to $z_m[n]$ and $z_m[n-1]$, therefore,

$$[z_m[n-1] - z_m[n] + \gamma]_{+} + [L_n - z_m[n] + \gamma]_{+} = \gamma \tag{3.34}$$

Equation 3.34 can now be implemented using only two-input IFMP (2-IFMP) units in which it is extended to implement a complex spiking neural network as shown in Fig.3.7 (b). An array of 2-IFMP units is integrated on silicon and the connectivity to the array is potentially achieved using an FPGA. The FPGA could also implement the synapse by storing the synaptic weights on a high-density digital memory. Fig. 3.8 shows the response of five inputs to log-sum exp function that is identical with cascaded margin in series, cascaded margin in parallel, cascaded pulsed margin (IFMP) in series and cascaded pulsed margin (IFMP) in parallel.

In [40], it is shown that the approximation between log-sum-exp and MP is enhanced as the number of inputs increases. In this work, the relation is shown between the γ constraint and the number of inputs for the exact match. Let $L_1, L_2..L_n$ be the input scores to the log-sum-exp formula and these inputs are of equal values then,

$$z_{LSE} = log(e^{(L_1)} + e^{(L_2)} + ..e^{(L_n)}) \tag{3.35}$$

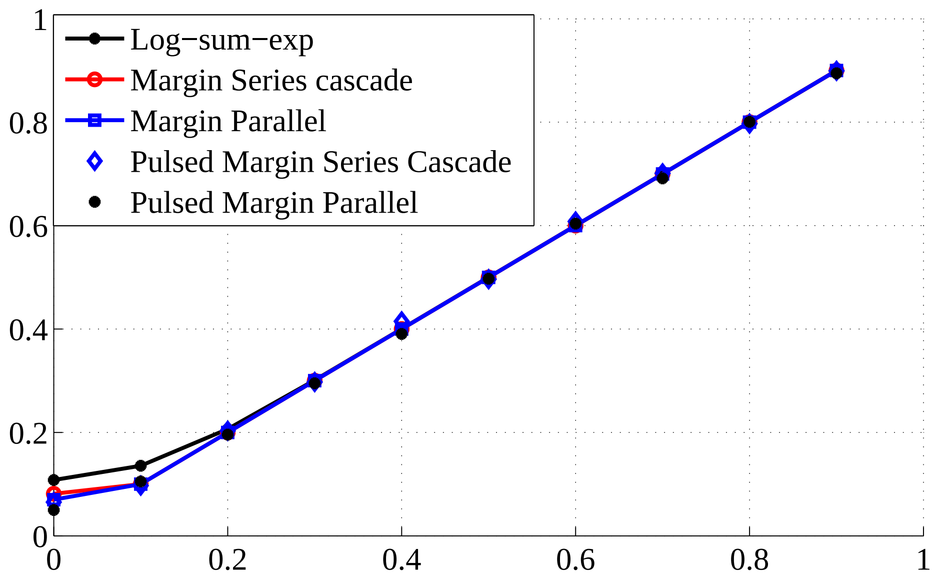


Figure 3.8: Implementation results of pulsed serial cascaded and parallel IFMP module approximation compared to the log-sum-exp function and the real values serial and parallel MP, The y-axis represents the output rate (z) for the different cases showed in the legend and the x-axis represents the change of rate for one input while keeping the other inputs at constant rates

$$z_{LSE} = log(ne^{(L)}) \tag{3.36}$$

In MP with the same assumption related the input scores assuming that the inputs are greater that z_{MP} (see Fig. 3.8)

$$n[L - z_{MP}]_{+} = \gamma \tag{3.37}$$

$$n[L - z_{LSE} + \gamma]_{+} = \gamma \tag{3.38}$$

The following equation is the result of substituting equation 3.36 in the above equation,

$$nL - n(log(ne^{(L)}) + n\gamma = \gamma \tag{3.39}$$

Then gamma needed for exact matching corresponds to the number of input scores (n) as following:

$$\gamma = -nlog(n)/(1-n) \tag{3.40}$$

For two different input scores L_1 and L_2, the value of gamma that matches the exact log-sum-exp value of z_{LSE} to the approximated value of z_{MP} will substituted in the following log-sum-exp and the margin equations respectively as,

$$z_{LSE} = log(e^{L_1} + e^{L_2}) \tag{3.41}$$

$$[L_1 - z_{MP}]_+ + [L_2 - z_{MP}]_+ = \gamma \tag{3.42}$$

Then the above equation is substituted with the exact value of z_{MP},

$$[L_1 - z_{LSE} + \gamma]_+ + [L_2 - z_{LSE} + \gamma]_+ = \gamma \tag{3.43}$$

If the input scores are greater than z_{MP}, then the value of γ will be,

$$\gamma = 2z_{LSE} - L_1 - L_2 \tag{3.44}$$

In general, for n input scores $L > z_{MP}$ and m input scores $L < z_{MP}$, γ is calculated for best matching between the exact log-sum-exp and the approximated margin propagation values as following: For the log-sum-exp value,z_{LSE} is calculated as,

$$z_{LSE} = \sum_{i=1}^{N} log(e^{L_i}) \tag{3.45}$$

Here N=n+m input scores where n and m represent the number of inputs greater and less than z_{MP} respectively. Therefore, the approximated margin propagation values z_{LSE} for n inputs is written as,

$$\sum_{i=1}^{n} L_i - nz_{LSE} + n\gamma = \gamma \tag{3.46}$$

$$z_{LSE} = \frac{\sum_{i=1}^{n} L_i + (n-1)\gamma}{n} \tag{3.47}$$

Equate the two equations 3.45 and 3.47 will get the followings,

$$\sum_{i=1}^{N} log(e^{L_i}) = \frac{\sum_{i=1}^{n} L_i + (n-1)\gamma}{n} \tag{3.48}$$

$$\gamma = \frac{n\sum_{i=1}^{N} log(e^{L_i}) - \sum_{i=1}^{n} L_i}{n-1} \tag{3.49}$$

In the above derivations, it is shown the convergence and approximation characteristics of the IFMP related the constraint γ. In the next section, the properties of the IFMP is shown with proofs and simulation plots.

3.2 IFMP properties

In [69], the properties and proofs were mentioned for analog margin propagation. In this work, the same properties are tested in pulse computation mode. Experiments are implemented for both the analog and pulse based margin propagation and shows the similarity between the two modes. The properties and proofs in [69] are listed below:

Since margin approximation uses thresholds $[x]_+ = max(x;0)$, then two lemmas are listed as facts to be used later for proving some of the other properties.

Lemma 1: $\forall a, b \in R, [a]_+ - [b]_+ \leq [a-b]_+$.

Lemma 1: $\forall a, b \in R, [a]_+ + [b]_+ \leq [a-b]_+$.

<u>**Property 1 (P_1, Scaling Property):**</u> For any $\alpha \in R, \alpha > 0$ and a set of series $\mathcal{L} = \{L_i\}, i = 1..N, M(\alpha\mathcal{L}, \alpha\gamma) = \alpha M(\mathcal{L}, \gamma)$. Here $\alpha\mathcal{L} = \{\alpha L_i\}, i = 1, .., N$.

The proof of this property is simple: if the condition $\sum_{i=1}^{N} [L_i - z]_+ = \gamma$ is satisfied, then the following condition

$\sum_{i=1}^{N} [\alpha L_i - \alpha z]_+ = \alpha\gamma$ is also satisfied. This property indicates that the threshold z scales with the scaling of the log-likelihood scores and the parameter γ.

<u>**Property 2 (P_2, Monotonicity):**</u> Given a set of scores $\mathcal{L} = \{L_i\}, i = 1, ..., N$, and if $\gamma_1 \geq \gamma_2 \geq 0$, then $M(\mathcal{L}, \gamma_1) \leq M(\mathcal{L}, \gamma_2)$.

One of the important implications of the monotonicity is the asymptotic property when $\gamma \to 0$ and is given by $\lim_{\gamma \to 0} M(\mathcal{L}, \gamma) = max(\mathcal{L})$.

Proof: Given

$$\sum_{i=1}^{N} [L_i - z_1]_+ = \gamma_1, \ \sum_{i=1}^{N} [L_i - z_2]_+ = \gamma_2, \ \gamma_1 \geq \gamma_2 \geq 0, \tag{3.50}$$

To prove the equation $z_1 \leq z_2$., then the two equations in (3.50) are subtracted to get the following,

$$\sum_{i=1}^{N} [L_i - z_1]_+ - [L_i - z_2]_+ = \gamma_1 - \gamma_2 \geq 0, \tag{3.51}$$

Based on Lemma 1, $\sum_{i=1}^{N} [z_2 - z_1]_+ \geq \sum_{i=1}^{N} [L_i - z_1]_+ - [L_i - z_2]_+$. combined with (3.51), it is deduced that $\sum_{i=1}^{N} [z_2 - z_1]_+ \geq 0$ Thus $z_1 \leq z_2$.

<u>**Property 3 (P_3, Convexity Property):**</u> Given a set of coefficients $\{\alpha_k\}$ satisfying $0 \leq \alpha_k \leq 1$ and $\sum_k \alpha_k = 1$, given a set of hyper-parameters $\{\gamma_k\}$, $M(\mathcal{L}, \sum_k \alpha_k \gamma_k) \geq$

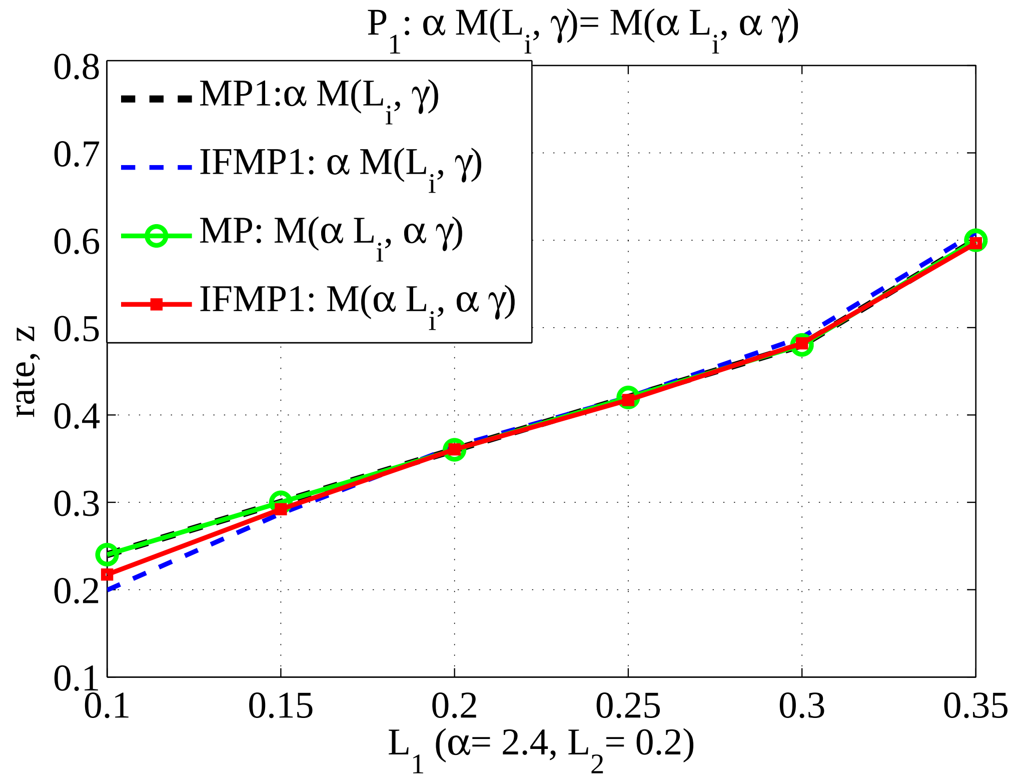


Figure 3.9: Scaling property for spiking IFMP compared to the continous values of margin propogation

$\sum_k \alpha_k M(\mathcal{L}, \gamma_k)$.

Proof: Given the same group of L_is, and a set of $\gamma_k, k = 1, .., n$.

$$\sum_{i=1}^{N}[L_i - z_1]_+ = \gamma_1, ... \sum_{i=1}^{N}[L_i - z_n]_+ = \gamma_n. \tag{3.52}$$

Based on property 1, equation (3.52) is transformed as,

$$\sum_{i=1}^{N}[\alpha_1 L_i - \alpha_1 z_1]_+ = \alpha_1\gamma_1, ... \sum_{i=1}^{N}[\alpha_n L_i - \alpha_n z_n]_+ = \alpha_n\gamma_n.$$

Add up the above equations, and according to Lemma 2 will get,

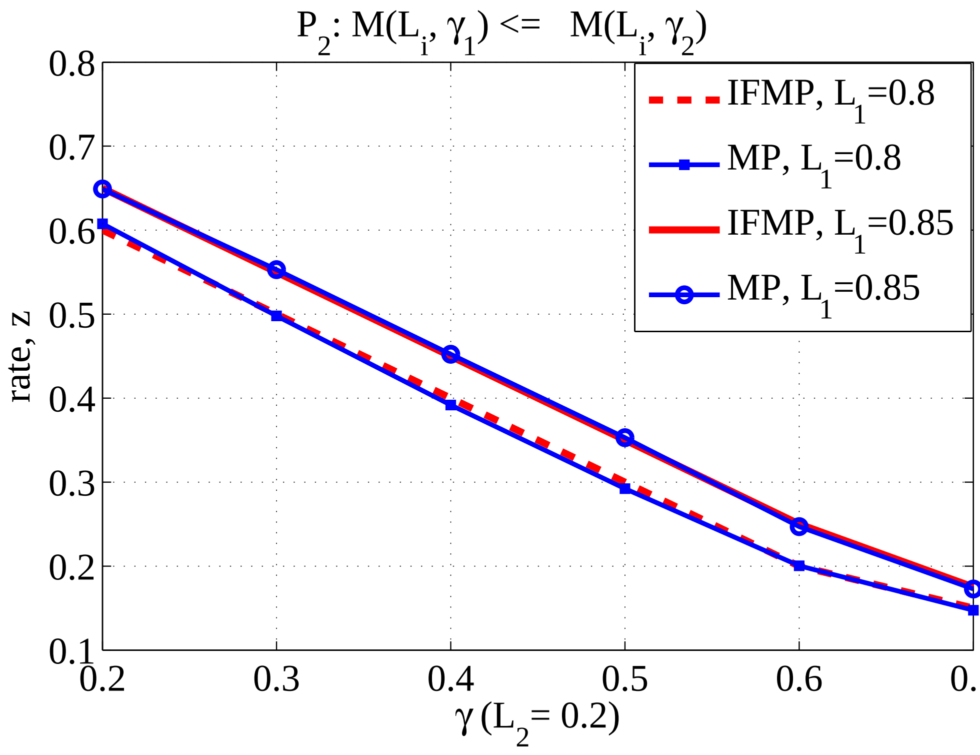


Figure 3.10: Monoticity propertiy for spiking IFMP compared to the continous values of margin propogation

$$\sum_{i=1}^{N}[L_i - \sum_{k=1}^{n} \alpha_k \mathbf{z}_k]_+ \geq \sum_{k=1}^{n} \alpha_k \gamma_k. \tag{3.53}$$

Denote z' satisfies $\sum_{i=1}^{N}[L_i - z_1']_+ = \sum_{k=1}^{n} \alpha_k \gamma_k$. Based on property 1, the following equation $z' \geq \sum_{k=1}^{n} \alpha_k \mathbf{z}_k$ is obtained. Therefore, $M(\mathcal{L}, \sum_k \alpha_k \gamma_k) \geq \sum_k \alpha_k M(\mathcal{L}, \gamma_k)$

Property 4 (P_4, Superposition Property): Given two sets of scores $\mathcal{L}$ and $\mathcal{G}$of size N with a well defined ordering, and if $\mathcal{L} + \mathcal{G}$ represent an element by element scalar addition, then $M(\mathcal{L} + \mathcal{G}, \gamma) \leq M(\mathcal{L}, \gamma) + M(\mathcal{G}, \gamma)$. **Proof:** Given,

$$\sum_{i=1}^{N}[L_i - z_1]_+ = \gamma, \quad \sum_{i=1}^{N}[g_i - z_2]_+ = \gamma, \tag{3.54}$$

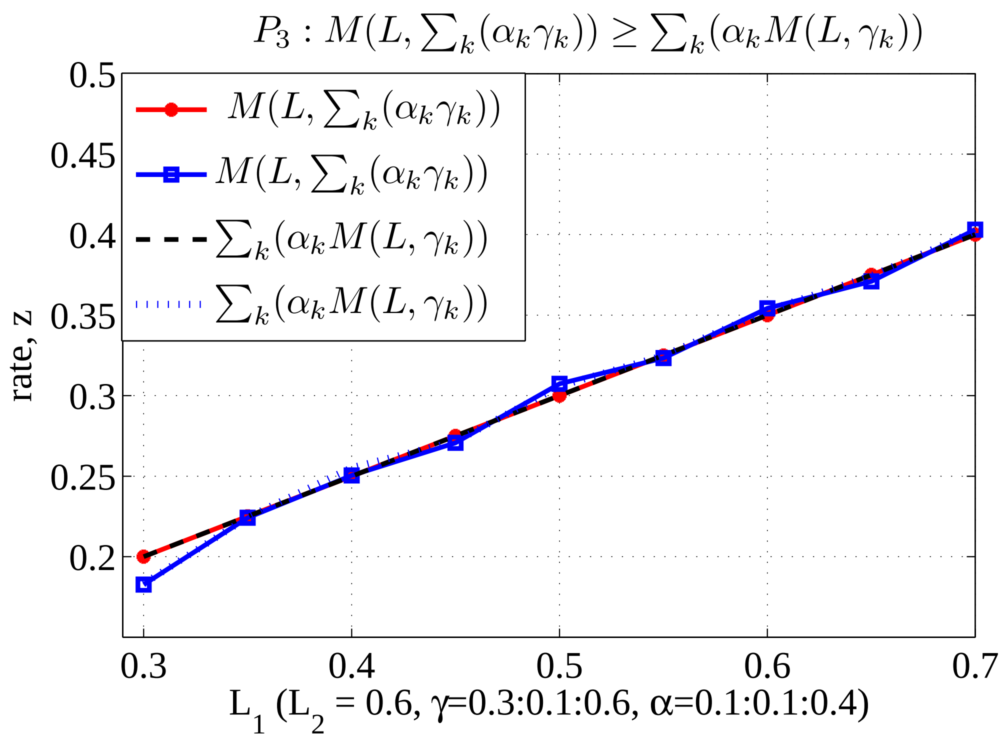


Figure 3.11: Convexity property for spiking IFMP compared to the continous values of margin propogation

What is needed to prove is: if

$$\sum_{i=1}^{N}[L_i + g_i - z_3]_+ = \gamma, \tag{3.55}$$

then $z_3 \leq z_1 + z_2$.

All the $L_i s$ are sorted and rearranged in a decreasing order so that $L'_1 \geq L'_2 \geq L'_3 ... \geq L'_N$, then N_1 is used to represent the number of L_i above the threshold z_1. The same operations are done to $g_i s$. All $g_i s$ are stored and rearranged decreasingly as $g'_1 \geq g'_2 \geq g'_3 ... \geq g'_N$. And the number of $g'_i s$ above the threshold z_2 is N_2. Rewrite equation (3.54) as,

$$\sum_{i=1}^{N} L'_i - N_1 z_1 = \gamma, \quad \sum_{i=1}^{N} g'_i - N_2 z_2 = \gamma, \tag{3.56}$$

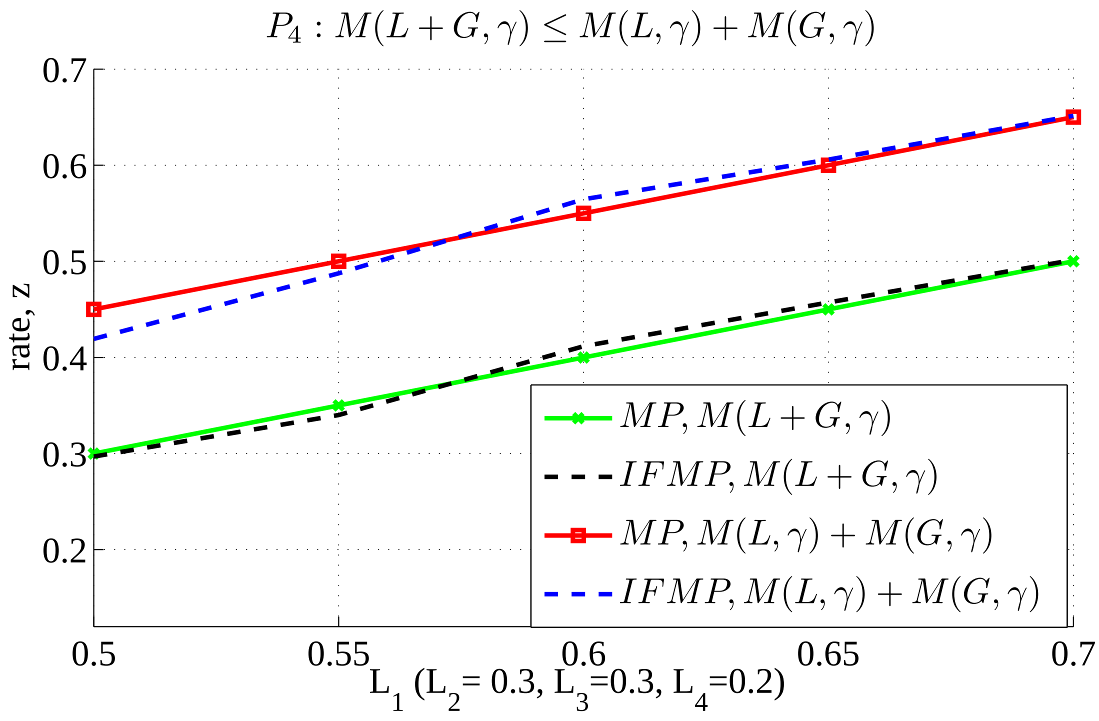


Figure 3.12: Margin propogation property for spiking IFMP compared to the continous values of margin propogation.

Assume $N_2 \geq N_1$ then from the latter equation in (3.56), the following will have,

$$0 \leq \sum_{i=1}^{N_1} g'_i - N_1 z_2 \leq \gamma. \tag{3.57}$$

Thus by adding the two equations in (3.56), the following will have,

$$\gamma \leq \sum_{i=1}^{N_1} L'_i + \sum_{i=1}^{N_1} g'_i - N_1 z_1 - N_1 z_2 \leq 2\gamma. \tag{3.58}$$

Equation (3.58) can be transformed as $\sum_{i=1}^{N_1}[L'_i + g'_i - (z_1 + z_2)] \geq \gamma$ from which can be deduced that

$$\sum_{i=1}^{N} [L_i + g_i - (z_1 + z_2)]_+ \geq \gamma, \tag{3.59}$$

Given (3.55) and (3.59), based on Property 2, it can be deduced that $z_3 \leq z_1 + z_2$ and vice versa if $N1 \geq N_2$ is assumed. If written in MP function, it is $M(\mathcal{L} + \mathcal{G}, \gamma) \leq M(\mathcal{L}, \gamma) + M(\mathcal{G}, \gamma)$.

Property 5 (P_5, Offset property): Given a set of scores $\mathcal{L}$ of size N and a scalar $g \in \mathcal{R}$, then $M(\mathcal{L} + g, \gamma) = M(\mathcal{L}, \gamma) + g$. Property 5 implies that if a constant offset to all the elements of input set leads to an equivalent offset in the output of the margin approximation function.

Proof: Based on the proof of Property 4, given

$\sum_{i=1}^{N} [L_i - z]_+ = \gamma$, if $\mathcal{G}$ is a scalar set with each $g_i = g, g \in \mathcal{R}$, equation (3.55) can be rewritten as $\sum_{i=1}^{N} [L_i + g - (z + g)]_+ = \gamma$. Written into MP function format, it becomes $M((\mathcal{L} + g), \gamma) = M(\mathcal{L}, \gamma) + g$.

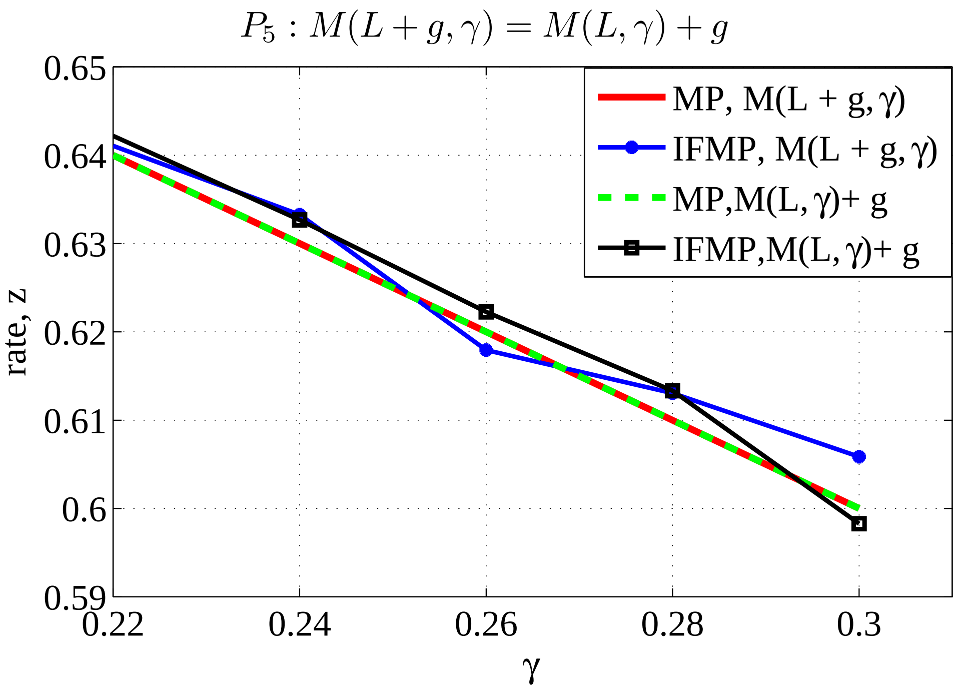


Figure 3.13: Super position property for spiking IFMP compared to the continous values of margin propogation.

The computational properties above are applied successfully to the spiking networks based on the proposed IFMP model. It is shown in Fig.3.9, 3.10, 3.11, 3.12 and 3.13 respectively the close match between the approximated continuous MP and the pulse-mode margin propagation (IFMP) for all of the above properties. Because of the limitation in representing the probabilities between [0,1], the above properties, especially the scaling and the offset properties, are used to represent the input parameters to approximate the nonlinear computational functions. In the next section, examples of nonlinear functions are approximated using IFMP pulse computations.

3.3 Examples of basic computations using IFMP

Before applying the model into algorithmic applications, it is necessary to show how the IFMP unit is used to approximate the basic computations such as addition,subtraction, multiplication, division, power, inner product and polynomial. The inputs to these units are probabilities and must be converted into logarithmic format. Then the resultant negative values are scaled using the properties of scaling and offset, as discussed above, in order to keep the input values between zero and one. To retrieve the correct computation values, the output of the computational function is scaled back in the reverse order of input scales as shown in Fig.3.14. The basic computations are implemented using log-sum-exp, analog margin propagation and pulsed margin propagation as an approximated operation as well as the error for each of the above computations will be shown graphically for different values of input probability rates.

Addition: Let $f = a + b = e^{log(a+b)}$ then the exponent z can be represented in log-sum-

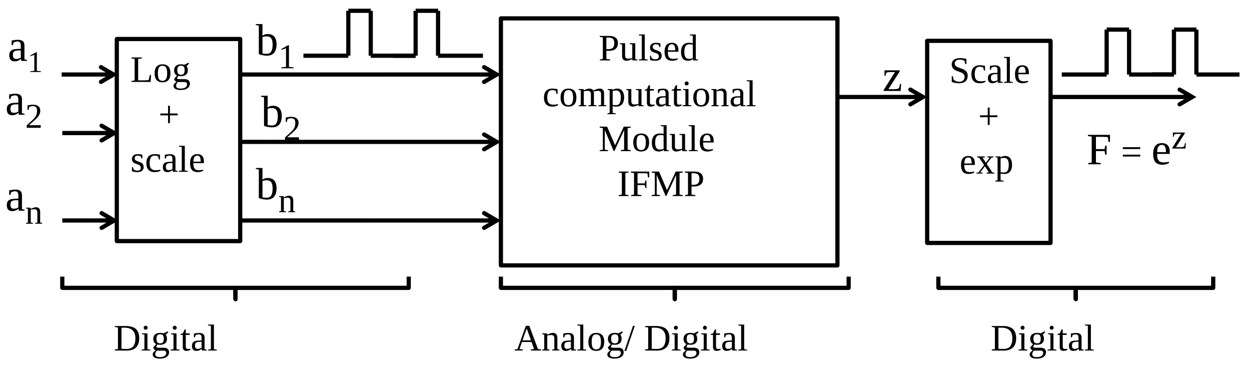


Figure 3.14: Input/output stages in the Pulsed computational unit.

exp and margin format as followings(see Fig. 3.15-a)

$$z = log(f) = log(e^{log(a)} + e^{log(b)}) \tag{3.60}$$

$$z = log(e^{L(a)} + e^{L(b)}) \tag{3.61}$$

$$z = M(L_a, L_b, \gamma) \tag{3.62}$$

where L stand for log function.

Multiplication: Let $f = a.b = e^{log(a.b)}$ then the exponent z can be represented in log-sum-exp and margin format as followings (see Fig. 3.15-b)

$$z = log(e^{L(a)+L(b)}) \tag{3.63}$$

$$z = M(L_a + L_b, \gamma) \tag{3.64}$$

Division: Let $f = a/b$ and $c = 1/b$ then $f = ac$. Then the above can be represented as

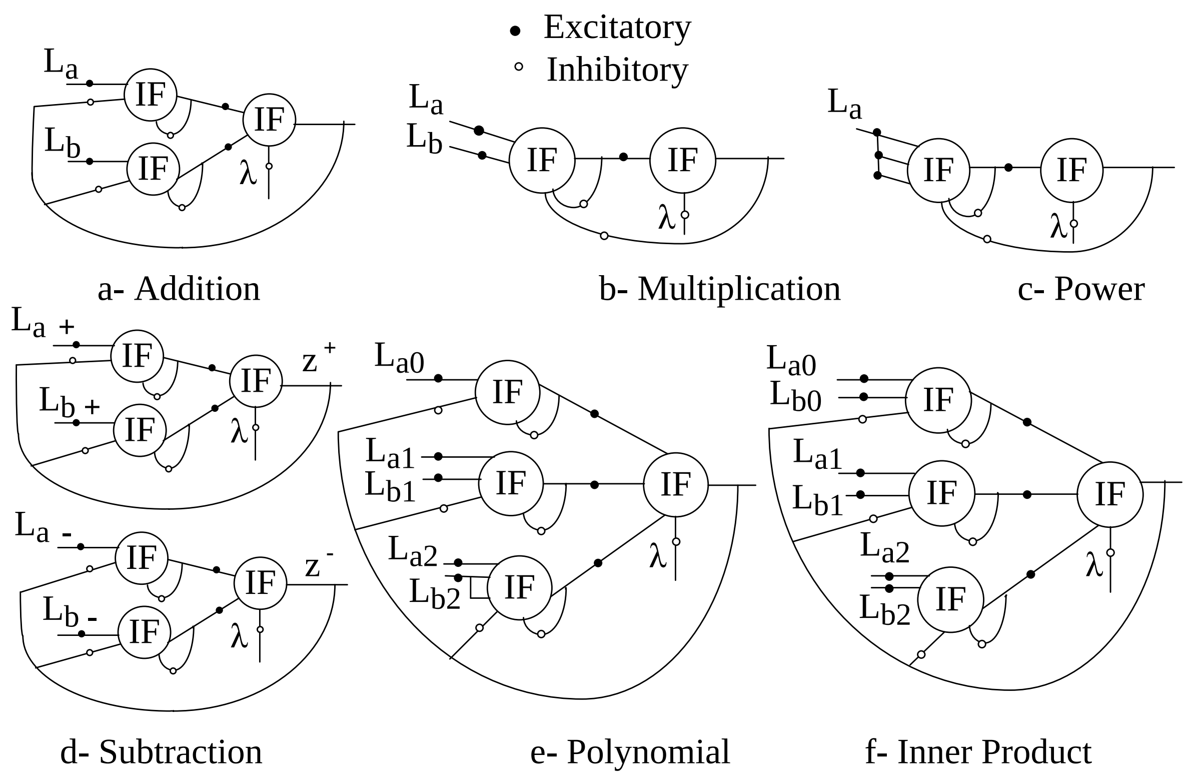


Figure 3.15: Basic computational architecture using IFMP.

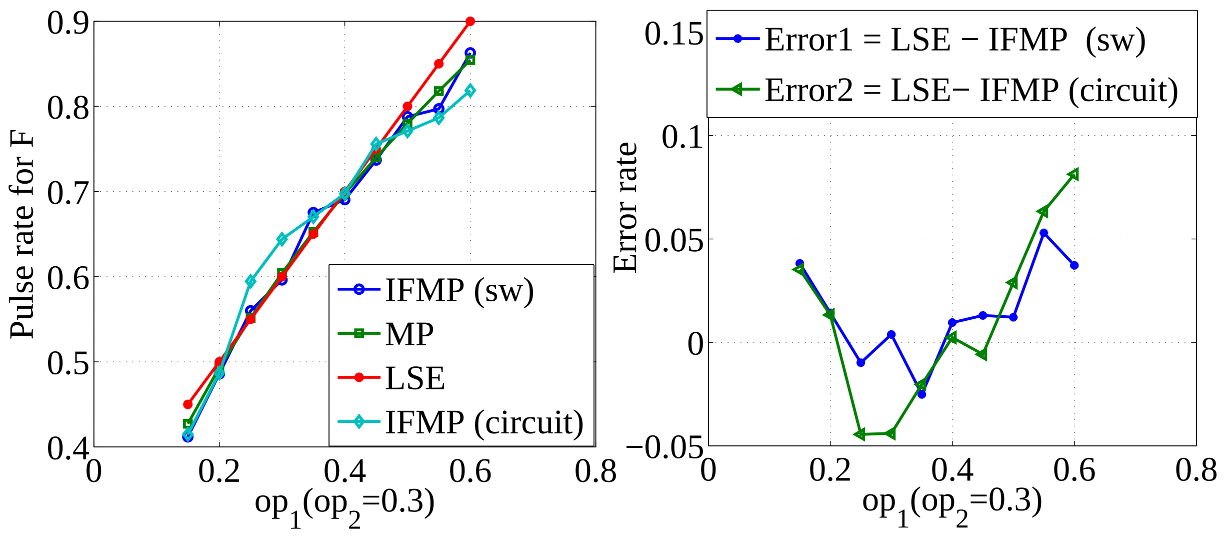


Figure 3.16: Computational functions for addition using $IFMP_{sw}, MP, LSE$ and $IFMP_{circuit}$. The right figures represent the error rate between the LSE computation and $IFMP_{sw}$ (Error1), and $IFMP_{circuit}$ (Error2)

a multiplication operation in IFMP representation as,

$$z = M(L_a + L_c, \gamma) \tag{3.65}$$

Power: Let $f = a_1 a_2 a_3 ... = e^{log(a_1 a_2 a_3 ...)}$ then the exponent z can be represented in log-sum-exp and margin format as followings (see Fig. 3.15-c)

$$z = log(e^{L(a_1)+L(a_2)...}) \tag{3.66}$$

$$z = M(L_{a_1} + L_{a_2} + ..., \gamma) \tag{3.67}$$

Subtraction: In the subtraction function, the differential form is used such that $f = a - b$ can be written as $f = (a^+ - a^-) - (b^+ - b^-) = (a^+ + b^-) - (b^+ + a^-)$

$$f = e^{log(a^+ + b^-)} - e^{log(b^+ + a^-)} \tag{3.68}$$

The following two exponents z^+ and z^- are then evaluated using two IFMP's as shown in Fig. 3.15 (d) such that

$$z^+ = M(L_{a^+}, L_{b^-}, \gamma). \tag{3.69}$$

and

$$z^- = M(L_{b^+}, L_{a^-}, \gamma). \tag{3.70}$$

Polynomial: Let $f = a_0 + a_1 b_1 + a_2 b_2^2 + a_3 b_3^3$ then the exponent z can be represented in log-sum-exp and margin format as followings (see Fig. 3.15-e)

$$z = log(e^{L_{a_0}} + e^{L_{a_1} + L_{b_1}} + e^{L_{a_2} + 2L_{b_2} + ...}) \tag{3.71}$$

$$z = M(L_{a_0}, L_{a_1} + L_{b_1}, L_{a_2} + 2L_{b_2}.., \gamma) \tag{3.72}$$

<u>**Inner Product:**</u> Let $f = a_0b_0 + a_1b_1 + a_2b_2 + ...$ then the exponent z can be represented in log-sum-exp and margin format as followings (see Fig. 3.15-f)

$$z = log(e^{L_{a_0}+L_{b_0}} + e^{L_{a_1}+L_{b_1}} + e^{L_{a_2}+L_{b_2}} + ...) \tag{3.73}$$

$$z = M(L_{a_0} + L_{b_0}, L_{a_1} + L_{b_1}, L_{a_2} + L_{b_2}..., \gamma) \tag{3.74}$$

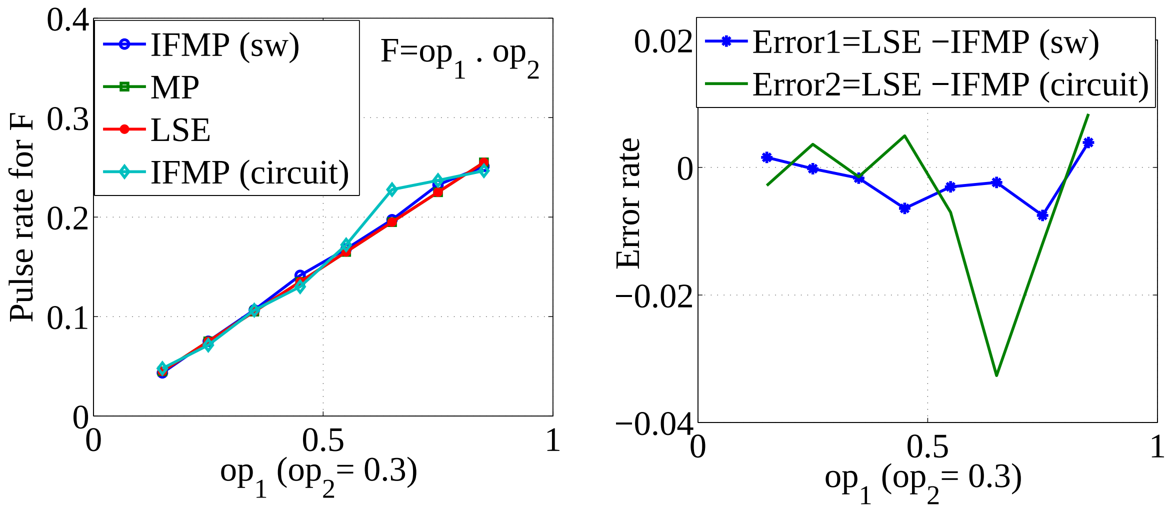


Figure 3.17: Computational functions for multiplication using $IFMP_{sw}, MP, LSE$ and $IFMP_{circuit}$. The right figure represent the error between the LSE computation and $IFMP_{sw}$ (Error1), and $IFMP_{circuit}$ (Error2)

The above computations are simulated using log-sum-exp (LSE), margin propagation for real values and pulsed IFMP. The left parts of Fig.3.16 ,3.17, 3.18, 3.19 and 3.20 represent the computational function for addition, multiplication, power, inner product and polynomial for pulse-based IFMP software simulation, continuous values MP, LSE and pulse-based hybrid circuit simulation IFMP, while the right part represents the error between the LSE and pulse based IFPM values for both software and hybrid circuit simulation. The absolute

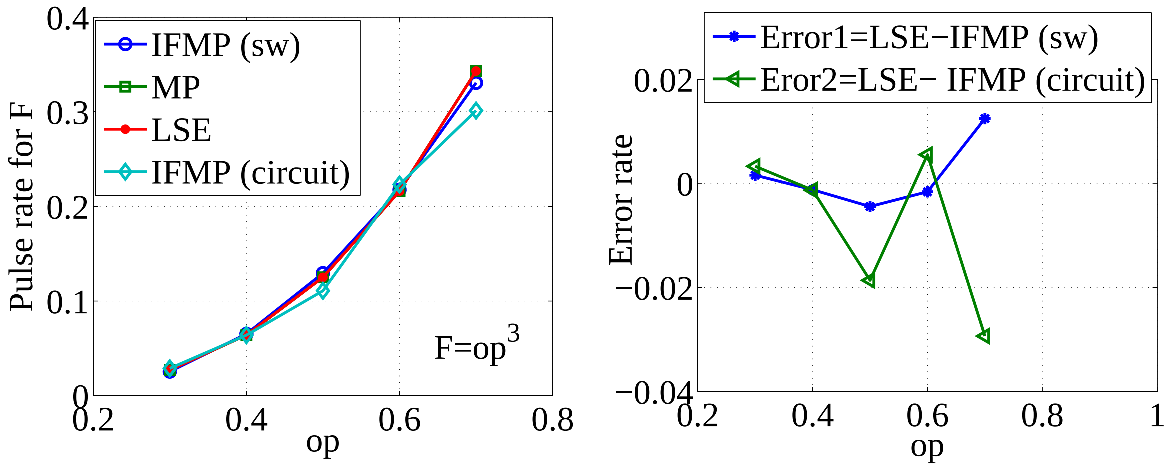


Figure 3.18: Computational functions for power using $IFMP_{sw}, MP, LSE$ and $IFMP_{circuit}$. The right figure represent the error between the LSE computation and $IFMP_{sw}$ (Error1), and $IFMP_{circuit}$ (Error2)

errors between the software simulation and the theoretical output rates for all computations are less than 0.03, where the number of iteration in these cases are 10000. On the other hand, the number of iterations for the circuit simulation in cadence are applied for 3000 for the sake of storage space allocated for research purpose. Therefore, the absolute errors in the cases of circuit simulation are higher which are less than 0.07.

The rates in the above experiments are chosen so that the output computational results are in the limits of probability values of [0:1]. If the first operator in the addition case for example is 0.3, then the second operator in the experiment is between [0:0.6]. In the multiplication and power cases, the computational results have no conflict with the probability limit. As experimental procedure for the multiplication operation, one operand is equal to 0.3 and the other is equal between [0.1:0.9]. For the power function, the range of operand in the experiment is between [0.3:0.7]. For the inner product function, there are four operands as discussed in the mapping above,(op_1, op_2, op_3, op_4), three of them are fixed to 0.3, 0.4 and 0.5 respectively and the forth operand varies between [0.1:0.8]. For the polynomial function,

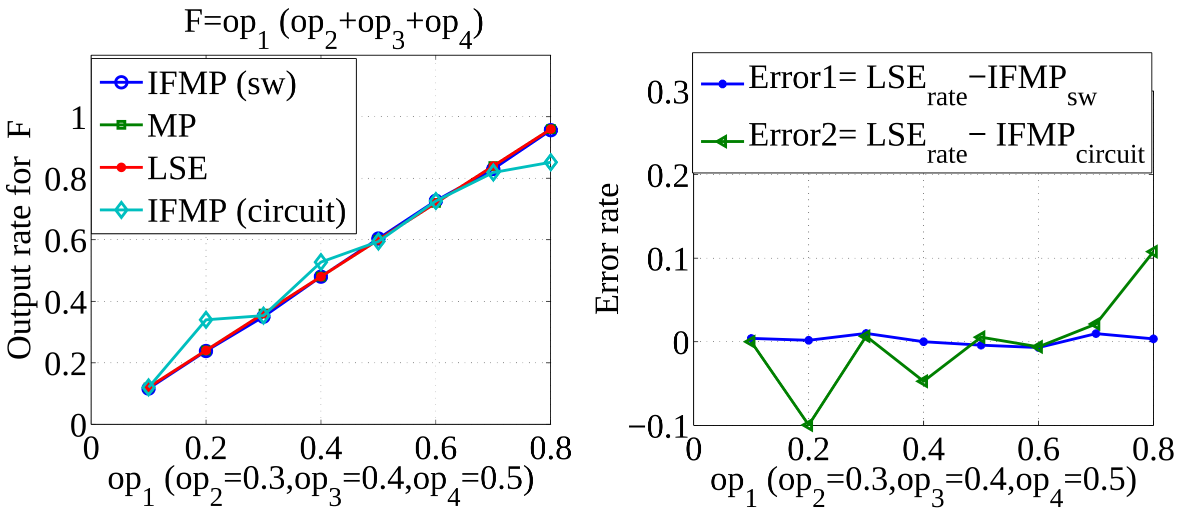


Figure 3.19: Computational functions for inner product using $IFMP_{sw}, MP, LSE$ and $IFMP_{circuit}$. The right figure represent the error between the LSE computation and $IFMP_{sw}$ (Error1), and $IFMP_{circuit}$ (Error2)

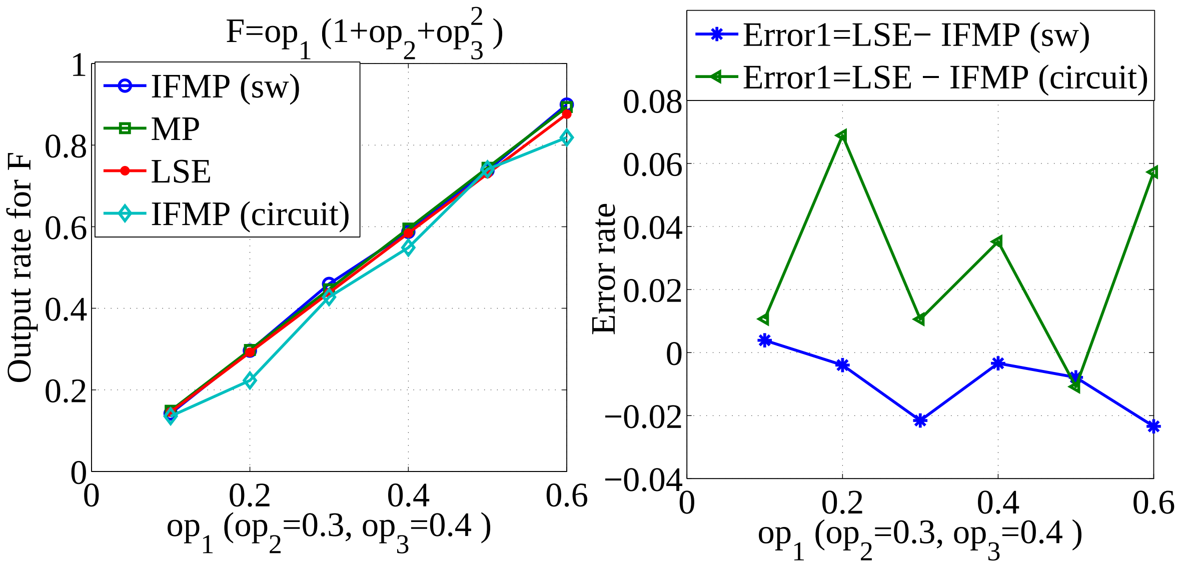
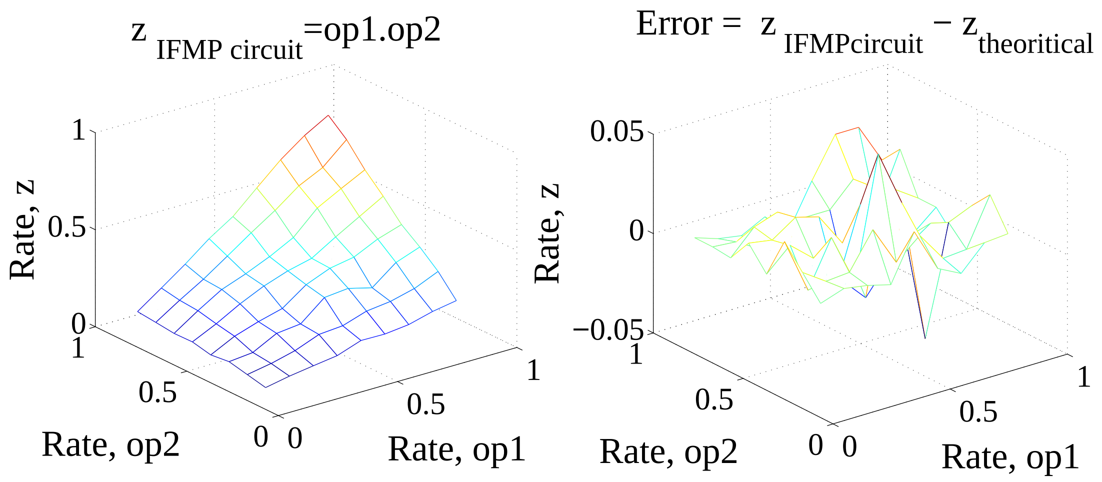


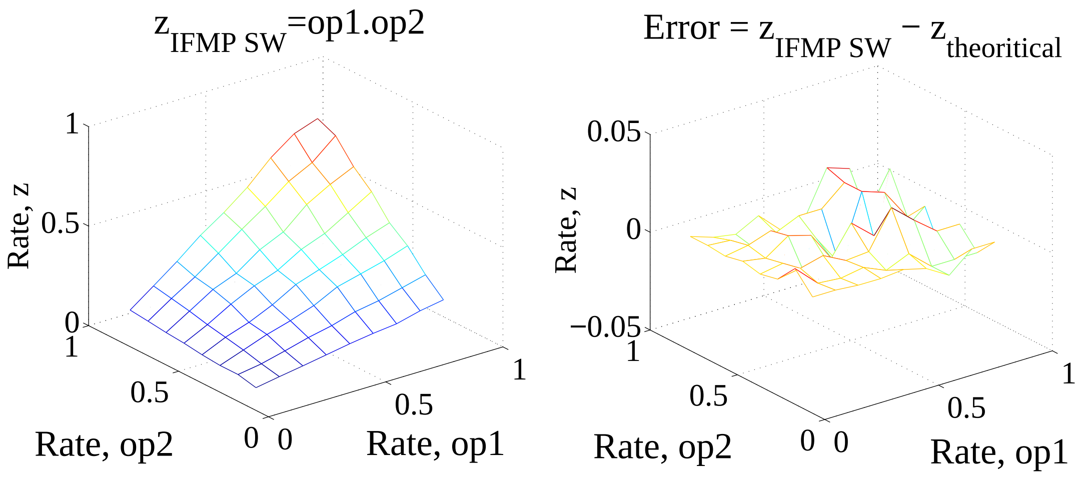
Figure 3.20: Computational functions for polynomial using $IFMP_{sw}, MP, LSE$ and $IFMP_{circuit}$. The right figure represent the error between the LSE computation and $IFMP_{sw}$ (Error1), and $IFMP_{circuit}$ (Error2)

there are three operands as discussed in the mapping above,(op_1, op_2, op_3), two of them are fixed to 0.3 and 0.4 and the third operand varies between[0.1:0.6].

For more comprehensive display of the pulse based computations explained above, ex-



(a)



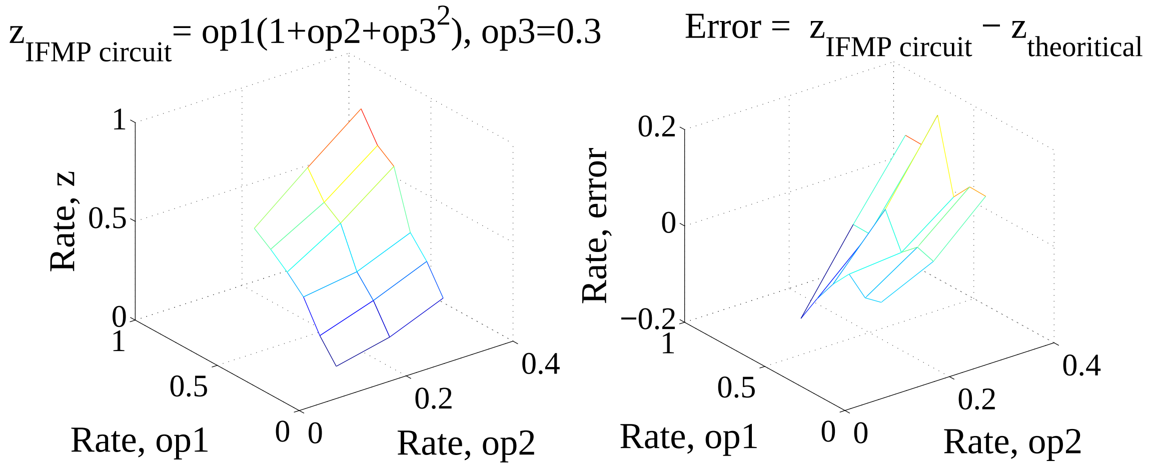
(b)

Figure 3.21: Output Rates and error corresponding to the theoretical computation for both the circuit simulation in (a) and software simulation in (b) to evaluate the multiplication function using pulse computation

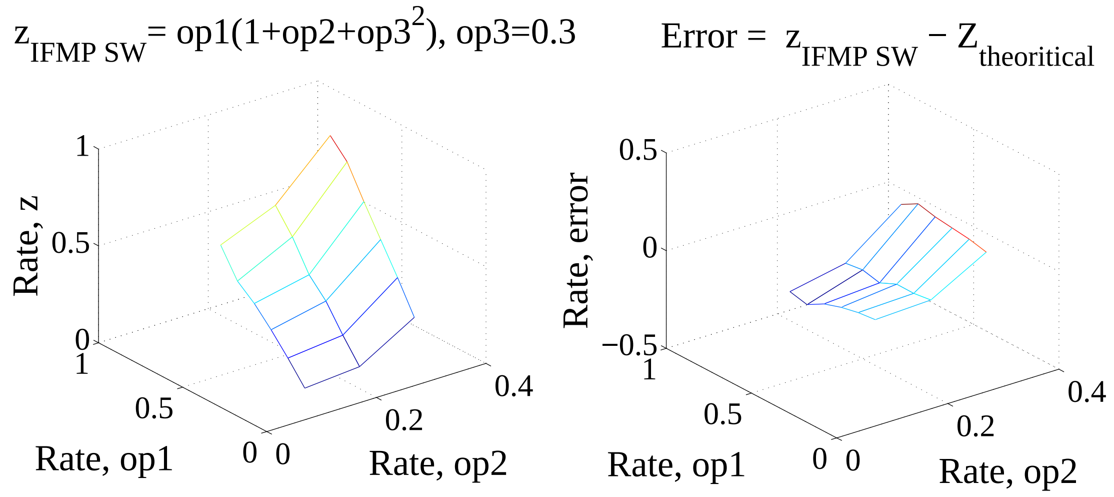
periments are implemented when both of the IFMP inputs are changing. For example, experiments are implemented to evaluate the output rates for the pulse based IFMP circuits and theoretical modules to show the dynamic characteristics for the multiplication operation and polynomial operation as in Figure3.21, 3.22 and 3.23 respectively. In the multiplication function, the range of first and the second operand is between [0.1, 0.9]. In the polynomial function, one operand is fixed on 0.3 and the other two operands are in the range of [0.1:0.3] and [0.1:0.6] respectively. In the inner product function, two operands are fixed to 0.4 and 0.5 and the other two operands are in the range of [0.1:0.6] and [0.1:0.7] respectively.The absolute errors in the above cases are less than 0.1. The errors are due to either the number of random pulses in the simulation and circuit simulation or the balance accuracy between the excitatory and inhibitory paths of the IFMP circuit.

3.4 IFMP: circuit, layout and tested measurements

The analog circuit of IFMP module in Fig. 3.24-(a) is shown in Fig. 3.24-(b). Each neuron in the IFMP module represents the integration and threshold operations that are designed between two bounds $(2.34v, 0.9v)$. Initially, if the input of the integrator is zero, the outputs of the integrator and the cascoded inverter are equal to 3.3 and zero volts respectively. If the input voltage increases and reaches the high gain region of integrator amplifier, then the integration phase will built up which is considered as the discharging phase of the capacitor. The input current is integrated and the output voltage of the integrator discharges to the lower bound. At this point, the output of the cascoded inverter turned into logic one which will turn the output voltage of the integrator to the upper bound (charging phase of capacitor). The cycle of charging and discharging the capacitor C is repeated according to the amount of the current injected to the inputs of the integrator ('in' node). The injected currents to the three integrators are applied respectively during off and on states of the input pulses for the excitatory path (PMOS transistors) and inhibitory path (NMOS transistors). Modules N_1, N_2 have excitatory inputs (PMOS path), one self feedback inhibitory input (NMOS path) and one output feedback inhibitory input (NMOS path)



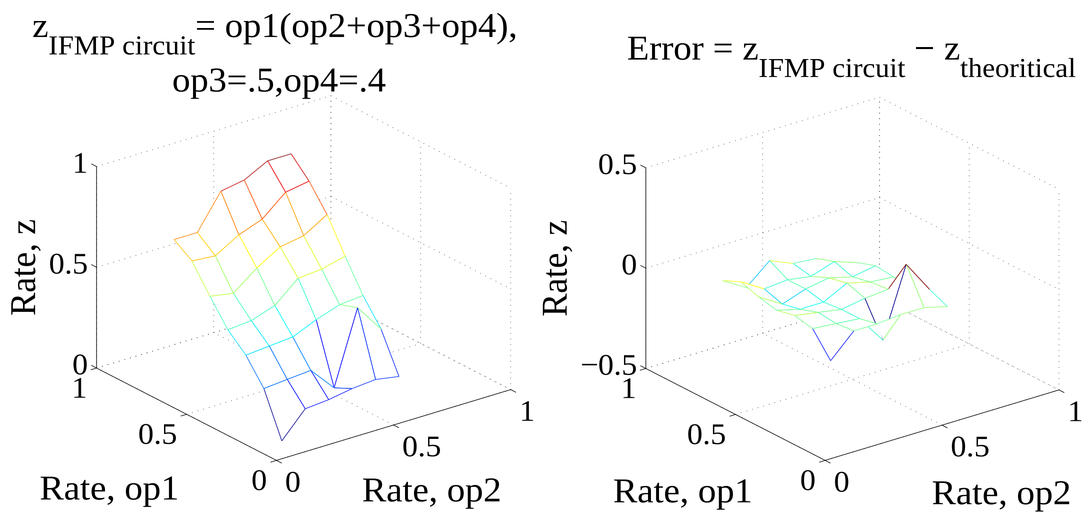
(a)



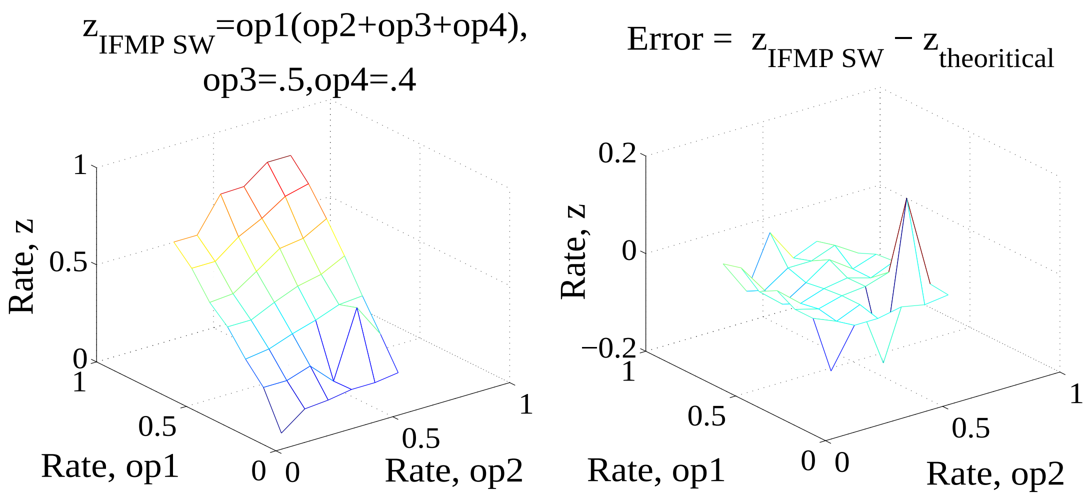
(b)

Figure 3.22: Output Rates and error corresponding to the theoritical computation for both the circuit simulation in (a) and software simulation in (b) to evaluate the polynomial function using pulse computation

while Module N_3 has two excitatory inputs (PMOS paths) and one inhibitory input. The last inhibitory input is represented by an adjustable constrain rate γ explained earlier.The values of the capacitance, current, W/L and the biasing voltages in Figure 2(3) are as follows: $C = 200 * 10^{-15}F, I = 10^{-12}A, W/L = 6/3m, Vb_1 = 0, Vb_2 = 3.3v, V3 = 0.7v, V2 = 1.37v, V1 = 1.4v.$



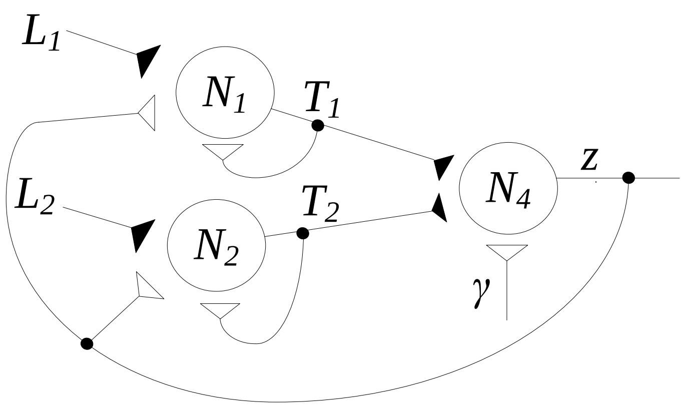
(a)



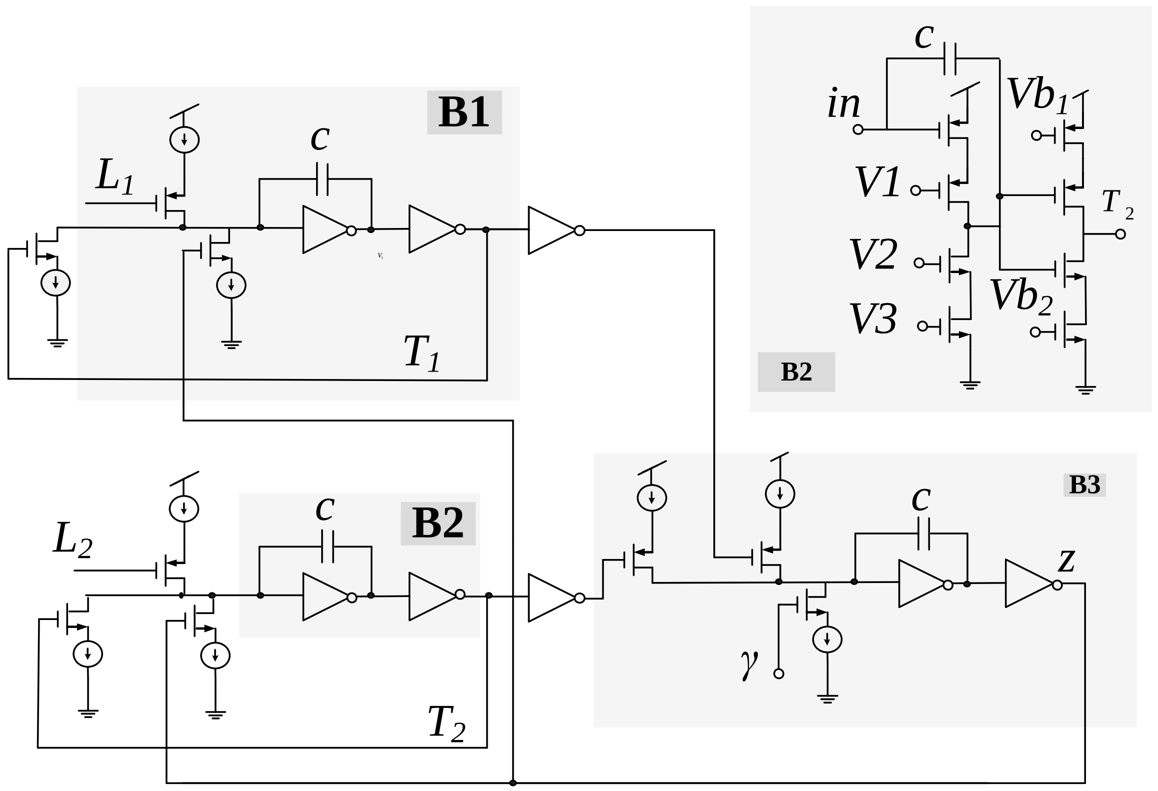
(b)

Figure 3.23: Output Rates and error corresponding to the theoritical computation for both the circuit simulation in (a) and software simulation in (b) to evaluate the inner product function using pulse computation

The dynamic characteristics of the IFMP module are verified in theoretical and layout design on standard 0.5m CMOS technology shown in 3.25 (b). In the experiments using the structure in 3.25(a), the chip is biased to ensure the balance between the excitatory and inhibitory parts using National instrument data acquisition input/ output. The biasing voltages are applied utilizing a Matlab program to initialize the ADC and DAC converters.



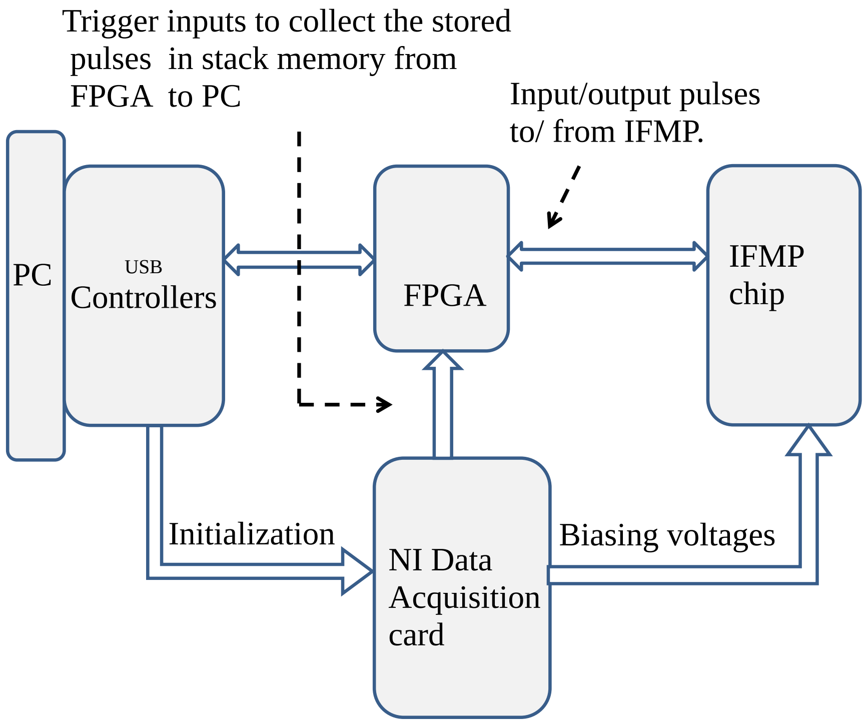
(a)



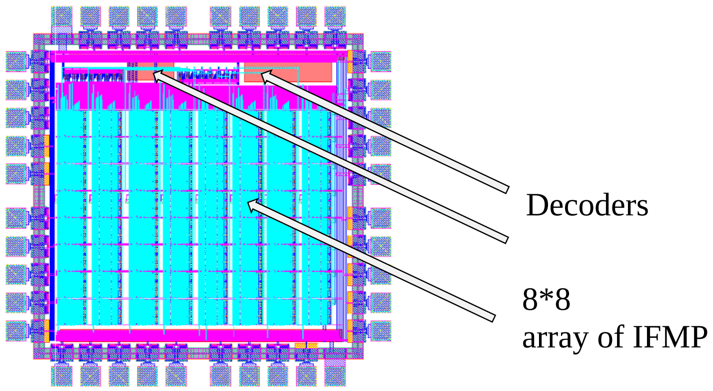
(b)

Figure 3.24: (a): Two inputs IFMP module. (b): Analog circuit for the IFMP module

The USB is supported with functions to communicate the input / output between th e PC and FPGA. The inputs are applied as random pulses with certain specified rates which are


Trigger inputs to collect the stored pulses in stack memory from FPGA to PC
Input/output pulses to/ from IFMP.
PC
USB Controllers
FPGA
IFMP chip
Initialization
NI Data Acquisition card
Biasing voltages

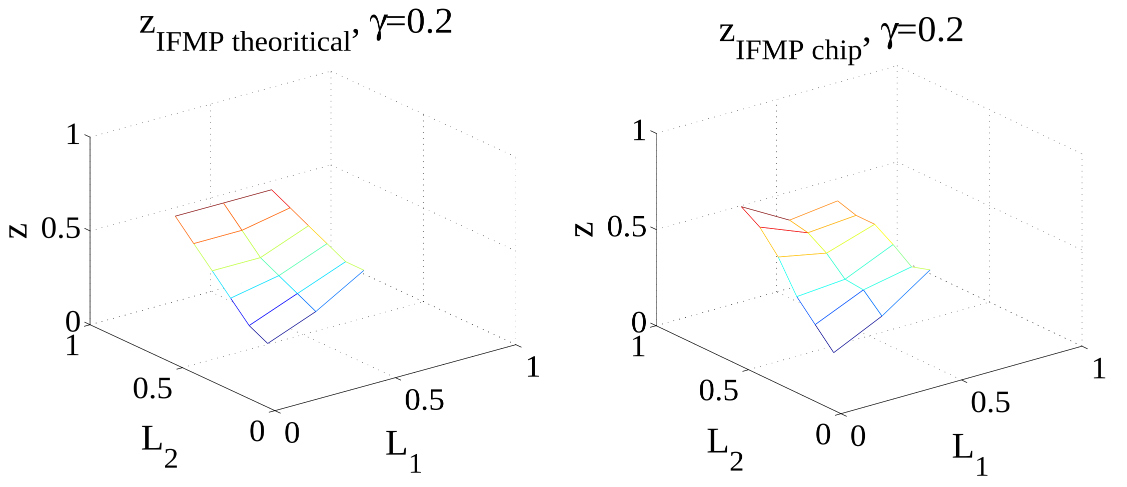
(a)



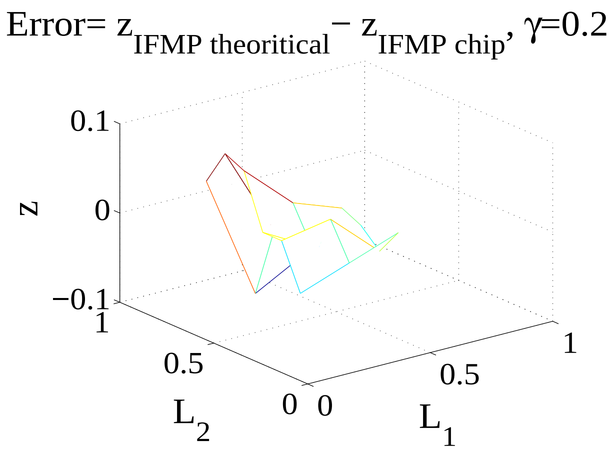
(b)

Figure 3.25: (a): Interfacing boards for testing the IFMP chip. The PC and FPGA are used to generate random pulses/getback the output pulses to/from the IFMP circuit. The National instrument interfacing card is used to generate voltage biasing to the transistors of the IFMP as well to intiate the stack memory to pull the stored pulses to the PC. (b): IFMP chip includes array of 8*8 IFMP chip and decoders

generated using Verilog hardware description language (Verilog HDL). The HDL program is converted into a bit file and is reconfigured into hardware components on FPGA using



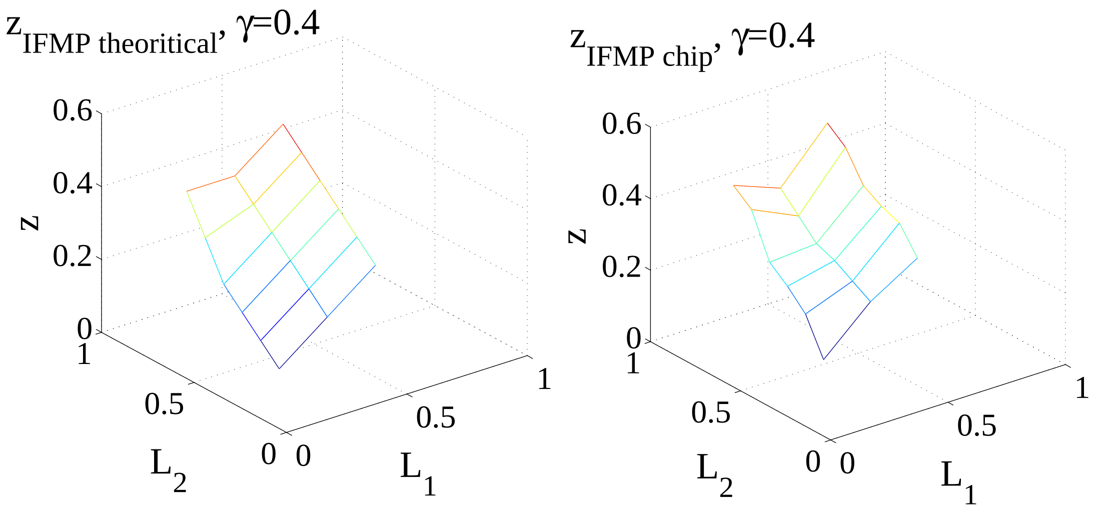
(a)



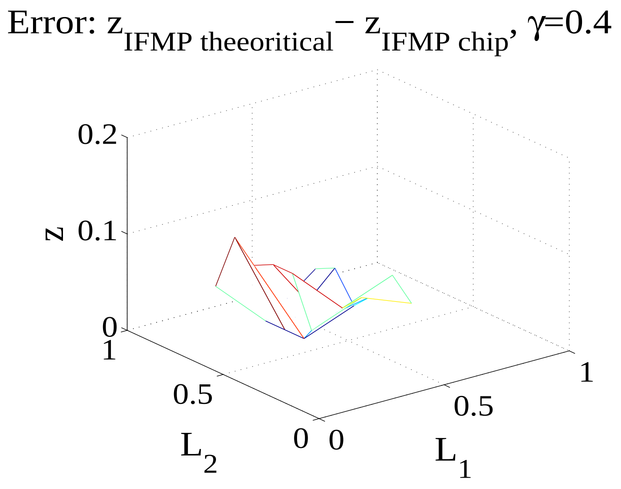
(b)

Figure 3.26: Sample tests of dynamic characteristics of IFMP when $\gamma = 0.2$ in (a) respectivally and the deference between the dynamic characteristics of the measured values, theoritical rates in (b)

Xilinx Integrated Software Environment (ISE 9.2) software. Experiments are implemented and shown in Figure 3.26, 3.27 and 3.28 to test the convergence of the dynamic characteristics of IFMP chip which match the theoretical convergence rates. In the figures, it is shown the measured output rates, theoretical rates and the difference between these two rates in when the input rates changes between 0 and 1 for three gamma rate (0.2, 0.4 and 0.6). The



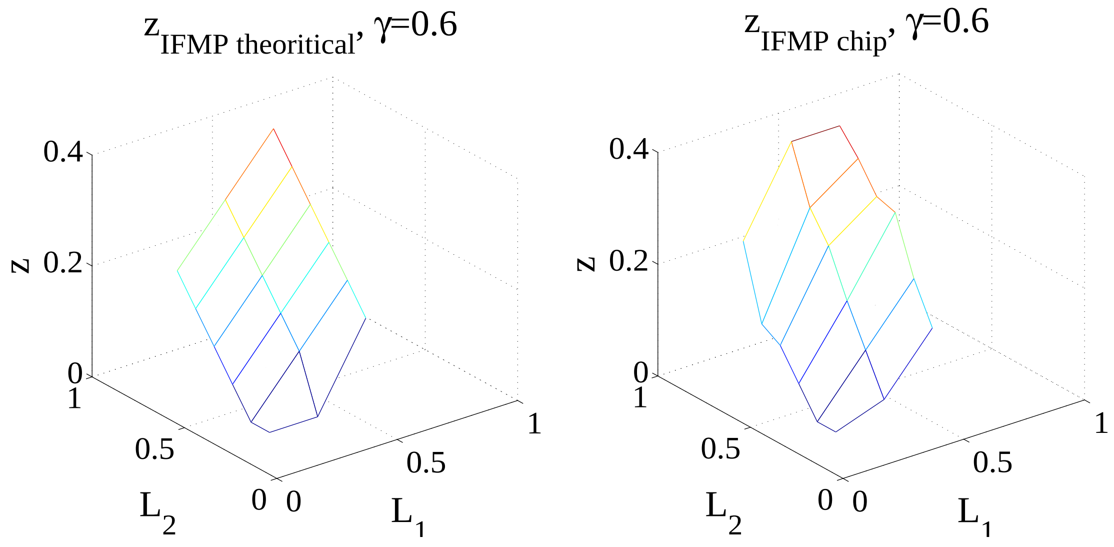
(a)



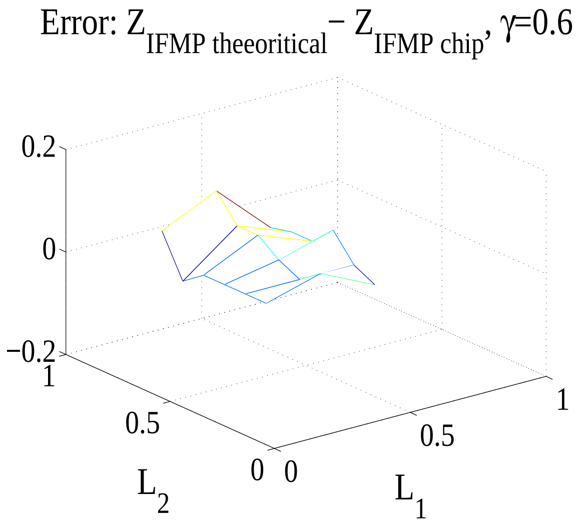
(b)

Figure 3.27: Sample tests of dynamic characteristics of IFMP when $\gamma = 0.4$ in (a) respectively and the deference between the dynamic characteristics of the measured values, theoritical rates in (b)

experiments are implemented for the three values of gamma when the range of input rates L_1 and L_2 are chosen between [0.2:0.6] and [0.3:0.8] respectively. The absolute errors for the three experiments are less than 0.1. In addition to the reason of the convergence error caused by the value of the iterations assigned to the stream inputs, there is another reason for the error in the IFMP layout. The main reason for the error in the IFMP layout is to



(a)



(b)

Figure 3.28: Sample tests of dynamic characteristics of IFMP when $\gamma = 0.6$ in (a) respectively and the diference between the dynamic characteristics of the measured values, and theoritical rates in (b)

maintain the balance in current between the exitatory and inhibitory paths of the IFMP circuit. The balance achieved by fixing the biasing voltages to certain accurate values. In circuit simulation, there is no problem in fixing these values but a problem in the circuit layout chip appears because the biasing voltages are common for all the paths due to pins limitation of the chip. This may not cause a problem if there is no mismatch in the physical performance of the transistors on a chip but that is not the case in real world. Therefore the number of pulses to coverage the input/output rates and the mismatch of transistors in hardware are the main reasons for the error in the measurement results of IFMP simulation and circuit layout.

It is well known that the low current biasing in subthreshold is important in low power applications, particularly in neuromorphic applications that simulate various aspects of brain function [1], [70]. Since the new designed modules, discussed above, are verified in low power circuits, which imply low current and/or voltage levels, then the designed circuits are more susceptible to the effect to noise. Therefore, It is important to suppress the effect of the noise in the design since the noise due to the thermal change can not be avoided. In this novel design, the type of connectivity of the IFMP network reduce the effect of the noise because the inhibitory connection suppress the noise to maintain the desired dynamic characteristics as will be discussed next.

The main parameter that causes the noise in the weak inversion mode is the thermal effect. A formula for subthreshold noise in MOS transistors has been derived by Enz [71] and Vittoz [72] from the consideration that the channel of transistor as being composed of a series of resistors. The integrated thermal noise of all these resistors yields the net thermal noise in the transistor. In [73], the white- noise is derived for the subthreshold MOS transistors that include the thermal effect as the sum of fluctuation components from the forward and reverse currents.

However, it is proven in neuronal population that the inhibitory connections between neurons shape the noise or reduce the noise and thus increase the signal to noise ratio (SNR). The SNR is improved if either the output signal is increased or the noise is reduced. In [74], [75], the authors proved that a method named noise shaping is used to improve the SNR in neuronal population. In the above approach, the inhibitory couplings between neurons are used to shift the noise power from one part of the spectum to another, thereby shaping the spectrum and more importantly, lowering the noise at the frequencies of interest. Therefore, it is possible in the noise shaping method to suppress the noise power within the signal bandwidth and thus improve the SNR. In [76], the author shows that the noise shaping is used to reduce the noise effect in performance of the analog circuit design fabricated on 0.5 μ m CMOS process.

In [5], the noise-shaping mechanism is discussed using the integrate-and-fire model. Consider a neuronal network consisting of N integrate-and-fire neurons. Each neuron is characterized by its intrinsic voltage $v_i(t)$, $i \in [1, N]$, and the neuron fires whenever v_i exceeds a threshold V_{th}. Between consecutive firings, the dynamics of the membrane potential can be expressed using the integrate-and-fire model as [74]:

$$\frac{dv_i(t)}{dt} = -v_i(t)/\tau_m - \sum_{j=1}^{N} W_{ij} \exp(-(t - t_j^m)\ /\tau_s) + \alpha x_i(t). \tag{3.75}$$

Here, t_j^m are the set of firing times of the j^{th} neuron and τ_m denotes the time-constant of the neuron capturing the leaky nature of integration denoted by the leaky potential of the membrane $v_i(t)/\tau_m$. The exponential term and the related time constant τ_s models the pre-synaptic filter $h(t)$ in Eq. (4.8) and time constant of the pre-synaptic spike-train. The parameter set W_{ij} denotes the synaptic weights between the i^{th} and j^{th} neurons and also denotes the set of learning parameters for this integrate-and-fire neural network.

To show how the synaptic weights W_{ij} influence noise-shaping, consider two specific cases: (a) when $W_{ij} = 0$, implying there is no coupling between the neurons and each neurons fires independently of the other; and (b) when $W_{ij} = W$ implying that the coupling between the

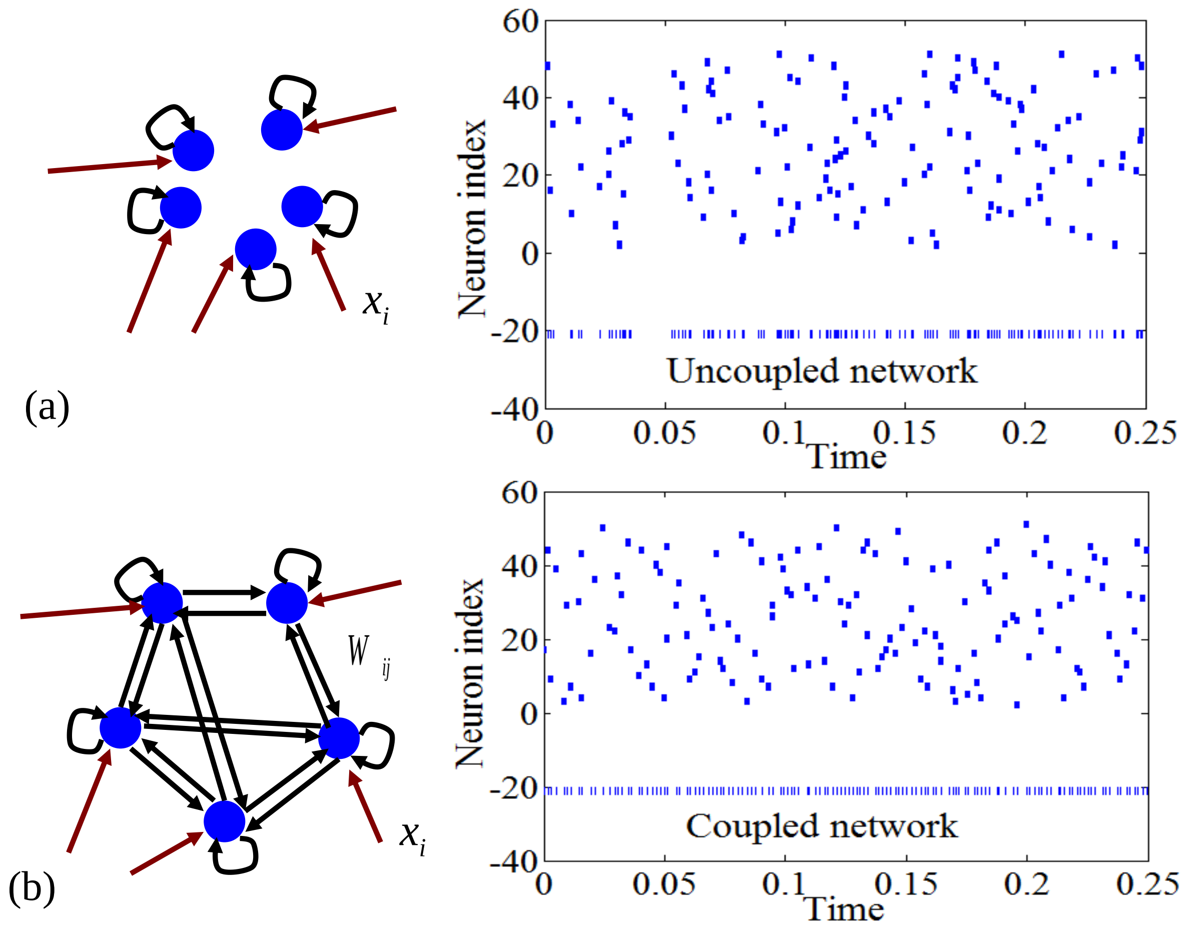


Figure 3.29: (a): Uncoupled connection between neurons and the coresponding raster plot. (b): Coupled connection between neurons and the coresponding raster plot showing the uniform and non uniform occurance of pulses at the bottom of the raster plots

neurons is inhibitory and constant. For a simple demonstration, τ_m is set to $1ms$ and N is set to 50 neurons. When the input $x_i(t)$ is applied, the raster plot is used to indicate the firing signal of the 50 neurons for both the uncoupled case and the coupled case as shown in Fig. 3.29 (a) and (b) respectively. In the raster plot of the two figures, the y-axis represents the neuron N_i that fires with respect to time, where i is the index of neurons. The lower parts of the raster plots present firings time of the neurons. For the uncoupled case, the population firing show clustered behavior where multiple neurons fire could fire in close proximity, whereas for the coupled case, the firing rates are respectively uniform indicating that the inhibitory coupling reduce correlation (introduce temporal correlation) between the neuronal firings. In the frequency domain, these correlations shift the noise power from one part of the spectrum to another, thereby "shaping" the spectrum and, more importantly increase the SNR ratio.

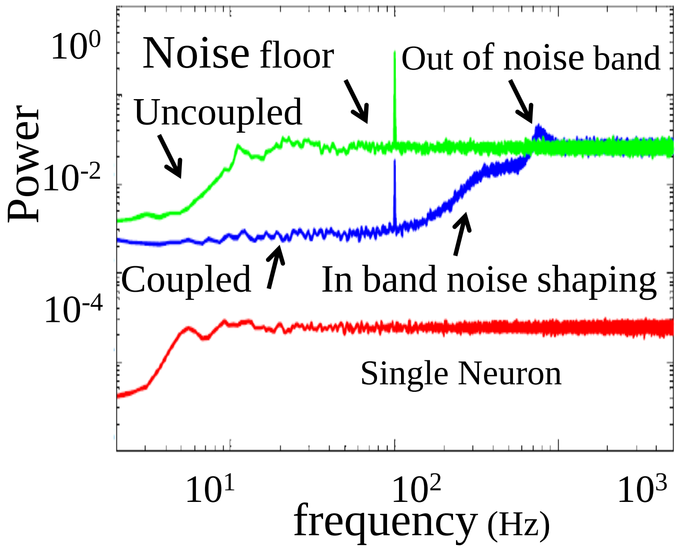


Figure 3.30: Power spectrum to show that the noise is suppressed in coupled connection of neurons compared with the uncoupled connection of neuron shown in Fig. 3.29

To understand the implication of the inhibitory coupling for noise-shaping, a sinusoidal input at frequency $f_0 = 1$ kHz is applied to all the neurons and the population firing rates are analyzed in frequency domain using a short time Fourier transform. Fig. 3.30 shows a comparison of the power spectrum for a single neuron, a neuron in a population of a coupled and uncoupled network respectively. The spectrum for a single neuron shows that it is unable to track the input signal since its bandwidth (1 kHz) is much larger than the firing rate of the neuron, whereas for the uncoupled/coupled neurons in a population case the input signal can be easily seen. For the uncoupled case, the noise floor, however, is flat, whereas for the coupled case, the noise floor from the signal band is shifted in the higher frequency range. The shaping of the in-band noise-floor enhances the SNR ratio of the network for a large network [74]. therefore, It can be deduced that the inhibitory connections enhance the performance of the IFMP network in the existence of thermal noise.

3.5 summary

To summaries this chapter, the characteristics of the new designed pulse computational IFMP module is verified in code simulation, mapped and verified into analog circuit as simulation and chip layout. The results in all the above simulations proved the convergence equation of the IFMP in verifying the dynamic characteristics and approximating the non-linear functions. The convergence of IFMP module depends on three main points:

1- The first parameter is the number of iterations of random pulses per time that represents the probability to satisfy the convergence. This effect is verified in software simulation and showed that the convergence is enhanced as the number of random input pulses to represent the rate increases above 3,000. The convergence can be achieved for less than 3,000 iterations (1,000 iterations), yet these lower iterations is not stable on repeatedly running the experiments. The reason is that the type of input pulses are random and that the number of pulses per time period in repeating the experiment is not constant. As a results of the randomness of the inputs, the chance of the potential membrane potential to reach the threshold and generate pulses is not fixed when the experiments are repeated. Hence, the average number of pulses per time might not be to satisfy the convergence.

2- The second parameter is the accuracy of the current balance between the excitatory and the inhibitory paths of the analog IFMP module. This effect is very important in the circuit and layout design. If the biasing current which is measured in pico-amperes

(20PA) and balanced accurately, then the convergence tends to be stable and close to the software simulation. Therefore, it is recommended, for future work, to design a fixed biasing techniques to stabilize the analog circuit biasing voltages. Therefore, this second parameter effect is the reason in the difference error between measured and simulated results discussed in this chapter.

3- The mismatch of the transistors in the hardware implementation causes difference biasing in the excitatory and inhibitory paths. This mismatch is caused by the common biasing for all the neuron modules in the IFMP array because of the limitation of the pins on the chip packages. The last reason is the cause of the error in hardware implementation besides the above two reasons for the error.

Chapter 4

IFMP Applications

This chapter shows three applications for the newly designed pulse computational architecture, the first two applications are concerned with factor graph algorithm in recognition of discrete sequences and trajectory traces using Hidden Markov Model (HMM) while the third one is a binary classification using support vector machine (svm). The purpose of applying the above applications is to show that pulse computations are applicable to complicated dynamic algorithms. If the newly designed pulsed computational architecture is applied and verified on a silicon chip then this unit could be the core of sensory designs of the neuromorphic applications. The chapter includes an introduction to the HMM, sequence recognition HMM, trajectory detection HMM, introduction to svm, binary classification using svm and finally a summary to the chapter.

4.1 Introduction to the Hidden Markov Model

In this section, a brief explanation of the Markov model and the hidden Markov model algorithm is introduced. Markov model is a probabilistic model of symbol sequences in which the probability of the current event is conditioned only by the previous event. Consider a sequence of random variables $s_1, s_2, , s_N$ in which the subscripts indicates word-position in a sentence. Here, the random variable is a function, and in this case its range is the vocabulary of the language. Then the probability of words $s_1, s_2..s_t$ according to Bayes formula is equal to,

$$p(s_1 = w_1, s_2 = w_2, ..s_t = w_t) \quad = p(s_t = w_t \mid s_1 = w_1, s_2 = w_2, ..s_{t-1} = w_{t-1})^* \tag{4.1}$$

$$p(s_1 = w_1, s_2 = w_2, ..., s_{t-1} = w_{t-1})$$

$$p(s_1 = w_1, s_2 = w_2, ..s_t = w_t) \quad = p(s_t = w_t \mid s_1 = w_1, s_2 = w_2, ..s_{t-1} = w_{t-1}) *$$

$$p(s_{t-1} = w_{t-1} \mid s_1 = w_1, s_2 = w_2, ..s_{t-2} = w_{t-2}) * \tag{4.2}$$

$$......p(s_1 = w_1)$$

Therefore, the probability of the last state can be recursively written as,

$$p(s_t = w_t \mid s_1 = w_i, s_2 = w_2, ..s_{t-1} = w_{t-1}) \quad = p(s_t = w_t \mid s_{t-1} = w_{t-1}) \tag{4.3}$$

On the other hand, the Hidden markov Model (HMM) is a probabilistic algorithm that is used to estimate the probabilities of unobserved events (in order to be decoded or recognized) using probabilities of observed events. For examples: in speech recognition, the observed data is the acoustic signal and the spoken sequence of words are the hidden parameters and the goal is to find the maximum probability out of all possible sequences of words. In handwriting recognition, the observed data is the designed images of words and the hand written words to be decoded or recognized are the hidden parameters. The estimation of the probabilities for unobserved events can be evaluated using the forward recursion algorithm. The following shows how to calculate the probability using the HMM- forward algorithm

[77]. Given the following parameters for the HMM states $s_1 : s_n$ in Fig.4.1(a):

1. Initial state distribution for the first state.

2. Transition probabilities from one state to the next or self state : $p(s_k/s_{k-1})$

3. Emission or observation probabilities: $x_{1:k}$

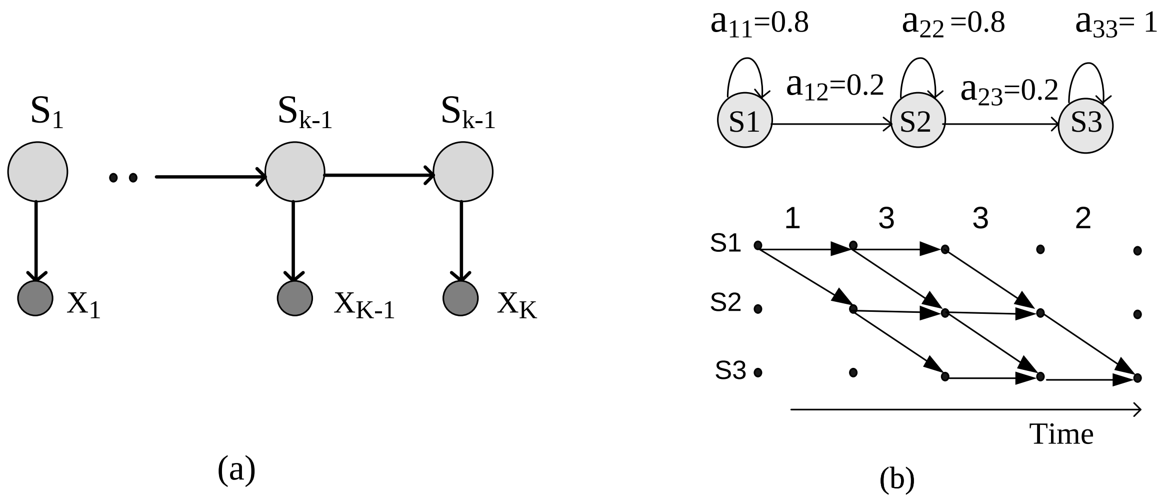


Figure 4.1: (a): HMM states(s)and emission events (x). (b): Three state HMM and it's representation using trellis with four input sequence to be recognized (1,3,3,2)

The goal is to compute the probability of $s_k/x_{1:n}$ according to the marginalization property as $p(s_k/x_{1:n}) = \sum_{s_{k-1}=1}^{m} p(s_k, s_{k-1}, x_{1:k-1})$ for $k = 1 : m$, then the probability is factorized and normalized as following;

$$p(s_k, x_{1:k}) \quad = \sum_{s_{k-1}=1}^{m} p(x_k/s_k, s_{k-1}, x_{1:k-1}) * \tag{4.4}$$

$$p(s_k/s_{k-1}, x_{1:k-1}) * p(s_{k-1}/x_{1:k-1}) * p(x_{1:k-1})$$

In the first term $p(x_k/s_k, s_{k-1}, x_{1:k-1})$, x_k is independent on both s_{k-1} and $x_{1:k-1}$. Similarly, in the second term $p(s_k/s_{k-1}, x_{1:k-1})$, s_k is independent of $x_{1:k-1}$. The terms

$p(s_{k-1}/x_{1:k-1}) * p(x_{1:k-1})$ is equivalent to $p(s_{k-1}, x_{1:k-1})$, therefore,

$$p(s_k, x_{1:k}) \quad = \textstyle\sum_{s_{k-1}=1}^{m} p(x_k/s_k) * p(s_k/s_{k-1}) * p(s_{k-1}, x_{1:k-1}) \tag{4.5}$$

Let $\alpha_{k-1}(s_{k-1}) = p(s_{k-1}, x_{1:k-1})$ and

$\alpha_k(s_k) = p(s_k, x_{1:k})$ then

$$\alpha_k(s_k) \quad = \textstyle\sum_{s_{k-1}=1}^{m} p(x_k/s_k) * p(s_k/s_{k-1}) * \alpha_{k-1}(s_{k-1}) \tag{4.6}$$

The above equation represents the HMM recursive equation where $\alpha_{k-1}(s_{k-1})$ is the pre or initial state distribution, $p(s_k/s_{k-1})$ is the transition probability and $p(x_k/s_k)$ is the emission probability. Thus the forward algorithm evaluate recursively the probabilities from the current state to backward state. HMM is feasible in recognition but it has limits in scalability for implementing large networks in analog circuit implementations using current-mode or voltage-mode circuits [78]. In the next section, it can be shown how to implement an asynchronous analog HMM circuit using log-likelihood decoding directly in time-domain using binary pulses. The core of the implementation is an integrate-and-fire margin propagation circuit that can approximate a soft-max function and can be cascaded together to build large, complex HMMs. The simulation results obtained at a circuit level show that the implementation of the asynchronous HMM is equivalent to the results obtained from a software implementation.

4.2 Sequence detection

In this application, the detection for sequences might be a recognition of slices to recognize a character, recognition of letters in certain sequence to recognize a word or determination of certain order in DNA sequence. In these applications, a sequence of numbers are needed for recognition. If it is intended to detect a specific discrete number in a sequence, then the required observations for such sequence are trained to get the highest probability of occurrence for successful detection. The resultant probability can be assigned in a matrix named as an observation probability matrix. The observation probability, as discussed in the previous section, is one of the main parameters in HMM. The dimension of the observation matrix depends on the number of HMM states and the size of numbers to be detected. For example, if the number of HMM states are three and the discrete numbers to be detected are a set of {1,2,3...10}, then the size of observation matrix is (3x10), where the row is denoted to the state and the column is denoted to the sequence number. Note that the sum of observation probability for each state (row) in the matrix is equal to 1. In this section, the number of states are chosen as three states and the the numbers to be detected in a sequence are chosen as set of {1,2,3}. Therefore, the dimension of the observation probability matrix is (3 x 3). Another parameter in the HMM sequence detection is the transition probability matrix which includes two types of probabilities: 1- the transition probability from one state to the next state. 2- the probability of being in the same state. Since there are three states in the HMM model, then the size of the observation matrix is (3x3), where the rows and columns denoted to the state number. The same condition related the sum of probabilities in each row or state in the observation matrix is valid in the transition matrix too (ie, the sum of probabilities in a row is equal to one). Next, the HMM model for detecting a sequence of numbers using the newly designed pulse computational IFMP algorithm will be discussed in details.

A generic architecture of an HMM is similar to that of a finite-state-machine with multiple states $s_1, .., s_k$ where the transition between the states occurs with a finite probability and each of the states generates random observations or symbols according to a well-defined probability distribution [79]. As an example, a three-state HMM is shown in Fig.4.1(b) which label the transition probability between states s_i and s_j. For each state, the probability distribution associated with it is based on which an observation is generated. If the probability distribution is continuous then the observations are the elements of real numbers vector. If the probability distribution is discrete then the observations are symbols drawn from a finite alphabet. In the example, 3-states HMM shown in Fig.4.1(b) can be mapped into trellis as three states to detect a four items sequence with time, the discrete probability distribution over three symbols is compactly represented using a observation probability matrix. An example of the observation matrix is:

$$O = \begin{bmatrix} 0.9 & 0.1 & 0.0 \\ 0.0 & 0.2 & 0.8 \\ 0.0 & 0.9 & 0.1 \end{bmatrix}$$

where the element $O_{i,x}$ represents the observation probability of generating symbol $x \in \{1, 2, 3\}$ while being in state S_j. The transition and observation probabilities for an HMM (discrete or continuous) are typically estimated using a training procedure based on a pre - collected sequence of observations. During the decoding procedure, the trained HMM model H_1 is used for estimating the posterior probability $p(x|H_1)$ of generating an observation sequence $x =\in \{1, 2, 3\}$ given the HMM. In practice, estimation of $p(x|H_1)$ is implemented using a forward recursion algorithm which can be expressed as,

$$p_j[n] = \sum_{i \in C_j} p_i[n-1] a_{ij} O_{j,x[n]} \tag{4.7}$$

where $p_j[n]$ represents the probability of being in state S_j at time instant n, a_{ij} represents the transition probability from state S_i to state S_j, $O_{j,x[n]}$ is the emission probability of an observation x in state S_j at instant n and C_j is the connection index for state S_j. For example, in Fig 4.1, the recursion for state S_2 can be simplified to

$$p_2[n] = p_1[n-1] a_{12} O_{2,x[n]} + p_2[n-1] a_{22} O_{2,x[n]} \tag{4.8}$$

Practical implementations of HMM decoding algorithms map equation 4.8 into log-domain such that the product of probabilities does not lead to precision or truncation errors. Define $L(p_j[n]) = log(p_j[n]), L(a_{ij}) = log(a_{ij}), L(O_{j,x[n]}) = log(O_{j,x[n]})$, then equation 4.7 can be expressed in the log-domain as,

$$L(p_j[n]) = log(\sum_{i \in C_j} exp(L(p_i[n-1]) + L(a_{ij}) + L(O_{j,x[n]}))) \tag{4.9}$$

Equation 4.9 involves a logarithm of a sum of exponents, abbreviated as log-sum-exp, for which a piece-wise linear approximation called margin propagation is discussed earlier. Given a general form of log-sum-exp function by

$$z_{log} = log(\sum_{i=1}^{P} exp(L_i)) \tag{4.10}$$

where L_i, $i = 1 : P$ represents the log-likelihood scores as in equation 4.9, margin propagation generates an approximation $z \approx z_{log}$ where z is computed as a solution to the following equation

$$\sum_{i=1}^{P}[L_i - z]_+ = \gamma \tag{4.11}$$

where γ is an adjustable parameter which is chosen to improve the quality of approximation. The operation $[.]_+$ denotes a threshold operation whose output value is generated if the input is positive, otherwise equals zero. Using equation 4.11 to approximate the equation 4.9 the following recursion is obtained.

$$\sum_{i \in C_j}[L(p_i[n-1] + L(a_{ij}) + L(O_{j,x[n]})) - L(p[n])]_+ = \gamma \tag{4.12}$$

A basic circuit of margin propagation (IFMP) is designed for two inputs $P = 2$ indicated in equation 4.11. Any state of the three states in Fig. 4.2 (below) represents an IFMP model and consists of three integrate-and-fire structures. For the first structures of state 1 $(N_1, N_2, N_4, N_5, N_7, N_8)$, there are three excitatory inputs (black triangle) and two inhibitory inputs (white triangle): the self-feedback and the feedback from the output of the IFMP structure, where the output here is fed from the third structure of the local state. The third one of each state (N_3, N_6, N_9), has two excitatory inputs and one inhibitory input. Let us denote $L_1[n]$ and $L_2[n]$ with n being a discrete time-index as the rate of input spike-train for the first two structures, then it can be shown that firing-rate of the output $L_z[n]$ (denoted by $\mathcal{E}(L_z) = lim_{T \longrightarrow \infty}\frac{1}{T}\sum_{n=1}^{T} L_z[n]$) asymptotically satisfies.

$$\sum_{i=1}^{P}[\mathcal{E}(L_i[n]) - (L_z[n])]_+ \longrightarrow \mathcal{E}(\gamma[n]) \tag{4.13}$$

Note that equation 4.13 convergences only in probability. The parameters in the above equations represent the expected values of binary (pulse) sequences. For example, a rate of $L = 0.5$ means that half of the equally space bits in a stream is equal to logic 1.

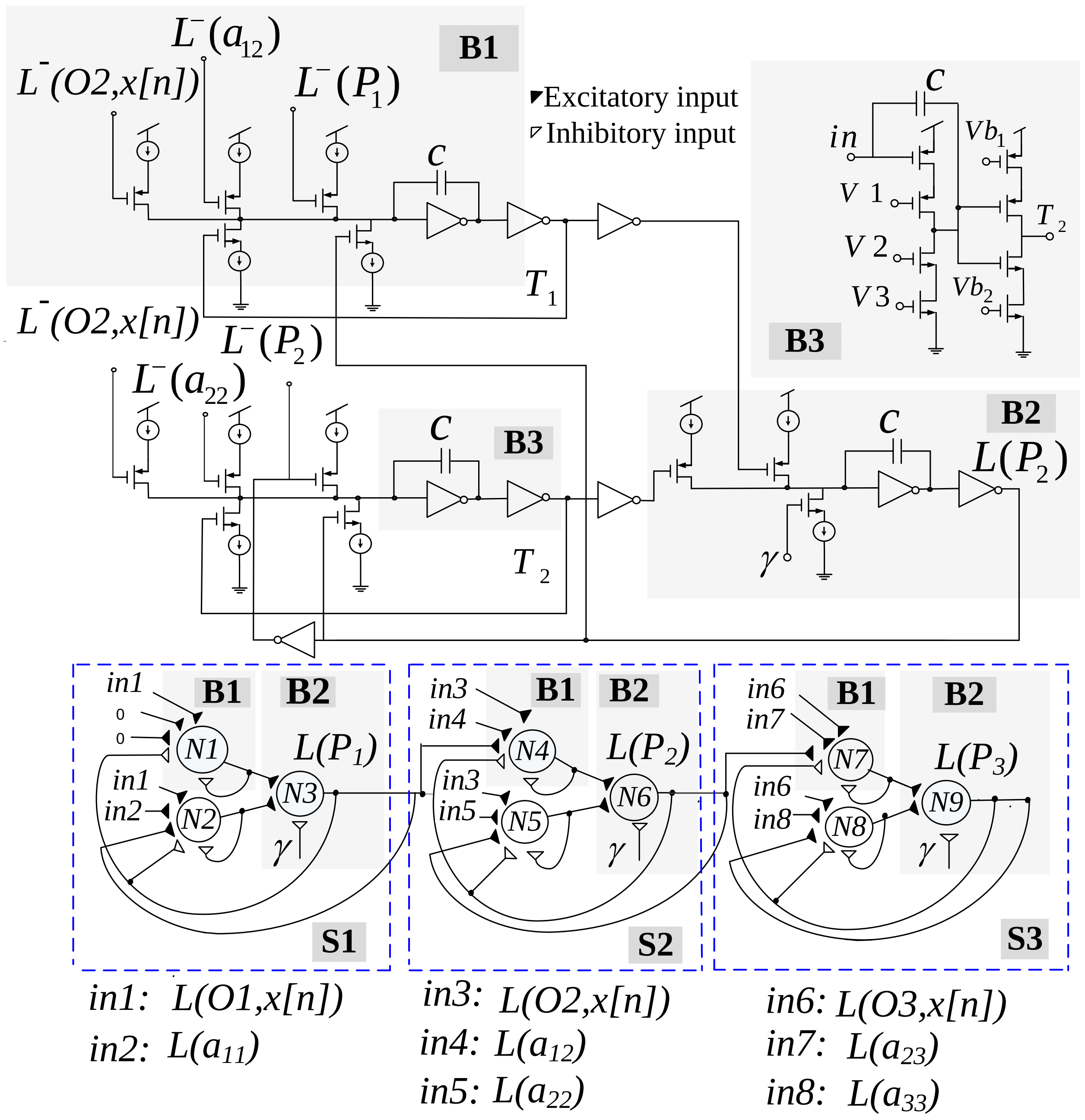


Figure 4.2: Three state HMM cascaded using IFMP structure to decode sequence and the analog circuit (below) for each cascaded state

The analog circuit of IFMP module shown in Fig. 4.2 (upper) (explained earlier in chapter 2) is mapped to each state of the HMM states shown in 4.2 (lower). The inputs of the analog circuit shown in the figure represent the input terms for state $S2$. Each state of the HMM states is represented by "two-inputs IFMP" unit. One input is fed by the sum-log of self feedback pulses of the state, observation probability to an input sequence and self transition probability. The second input is fed by the sum-log of the previous pulse state, observation probability to an input sequence and transition probability from a previous state to present state. These input are applied into the IFMP as expected value of pulses as $\mathcal{E}(L(O_{j,x[n]})) + \mathcal{E}(L(p_i[n-1]) + \mathcal{E}(L(a_{ij}))$. where j represents the present state, i represents the previous and the present states as two terms and $x[n]$ is the input sequence to the HMM states. As an example of the inputs to state $S2$, j and i in the above equation are denoted as states 2 and 1 respectively. Then, the above terms of the equation are represented as $\mathcal{E}(L(O_{2,x[n]})) + \mathcal{E}(L(p_1[n-1]) + \mathcal{E}(L(a_{12}))$ and $\mathcal{E}(L(O_{2,x[n]})) + \mathcal{E}(L(p_2[n-1]) + \mathcal{E}(L(a_{22}))$ respectively.

In the next section, the procedure and results of HMM decode sequence are demonstrated when the proposed model in Fig. 4.2, maps the equation below such that the expected value of the output spike is proportional to the expected values of the input spikes,

$$\sum_{i \in C_j} [\mathcal{E}(L(O_{j,x[n]})) + \mathcal{E}(L(p_i[n-1]) + \mathcal{E}(L(a_{ij})) - \mathcal{E}(L(p_j[n]))]_+ = \gamma \tag{4.14}$$

Procedure and results: The IFMP module discussed above represents one state machine of three HMM states. Here, the circuit of three states HMM is implemented to find the output rates $p1, p2, p3$ of states s_1, s_2, s_3 of Fig 4.2 (b- below) which is substituted from the recursive equations 4.9 and 4.12 as,

$$L(p_1)[n] = L[e^{[L(O_{1,x[n]})+L(p_0)]} + e^{[L(O_{1,x[n]})+L(p_1[n-1])+L(a_{11})]}] \tag{4.15}$$

$$L(p_2)[n] = L[e^{[L(O_{2,x[n]})+L(p_1[n-1])+L(a_{12}]} + e^{[L(O_{2,x[n]})+L(p_2[n-1])+L(a_{22})]}] \tag{4.16}$$

$$L(p_3)[n] = L[e^{[L(O_{3,x[n]})+L(p_2[n-1])+L(a_{23})]} + e^{[L(O_{3,x[n]})+L(p_3[n-1])+L(a_{33})]}] \tag{4.17}$$

The mapping of the parameters in the above equations is shown in the Fig. 4.2. Where $L(p_0)$ is the initial probability for the first state. In the figure, it is shown that each of the IFMP units has two terms; the pre state unit and the self state unit in addition to the related transition and observation probabilities for each state.

Two HMM ($H_1 and H_2$) are tested to observe the sequence $\{1, 3, 3, 2\}$ where the rows in the probability observation matrix represent the states and the columns represent the item sequence as a digit $\in \{1, 2, 3\}$. In HMM H_1, it is trained to maximize the probability in the third state according to trained observation. The second HMM H_2 has identical transition probabilities as H_1 but with different observation matrix that does not recognize the above sequence. The second observation is given as:

$$O = \begin{bmatrix} 0.9 & 0.1 & 0.0 \\ 0.1 & 0.9 & 0.0 \\ 0.9 & 0.1 & 0.0 \end{bmatrix}$$

In both observations, the excitatory inputs corresponding to each state S_j are the recursive feed-back term $L(p_i[n-1])$, the transition probability score $L(a_{ij})$ and the observation probability score $L(O_{j,x[n]})$. At the end of the HMM decoding (after N sequences have been

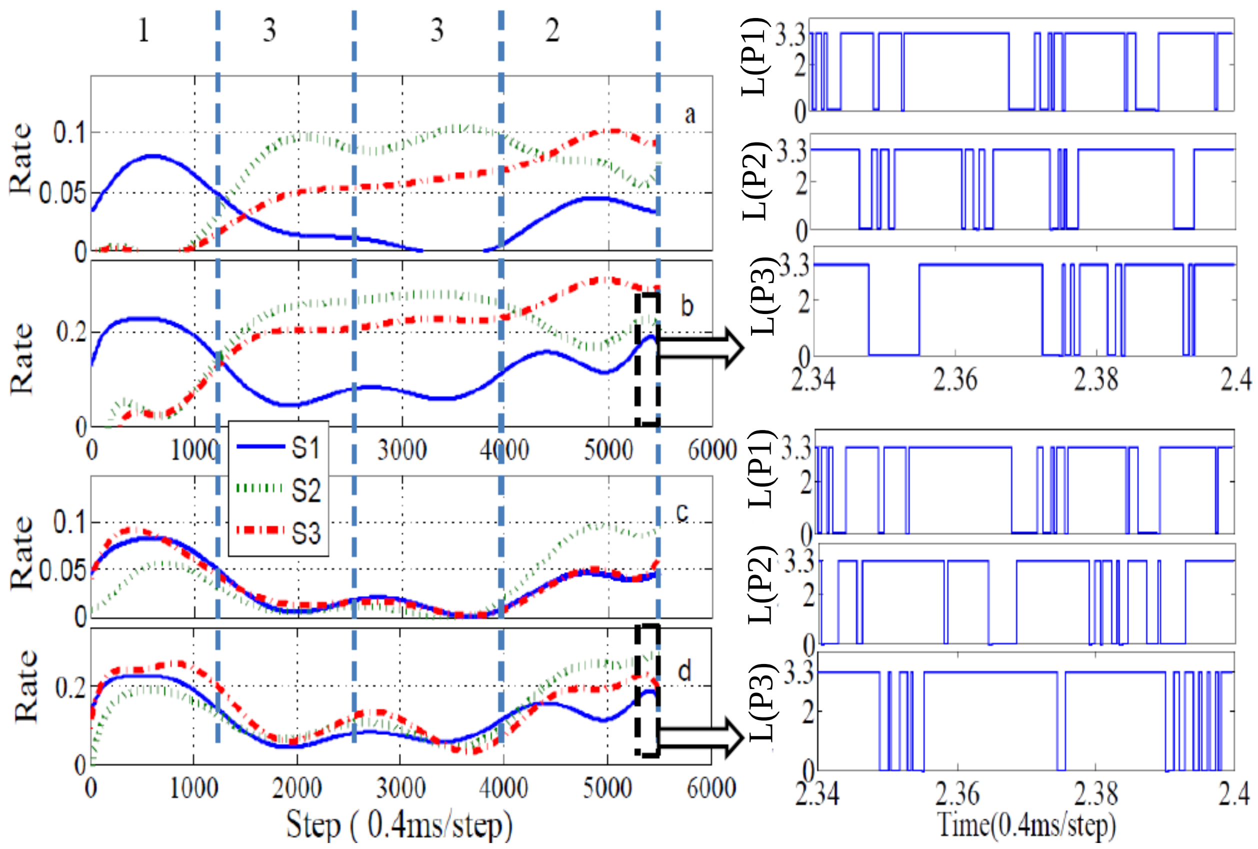
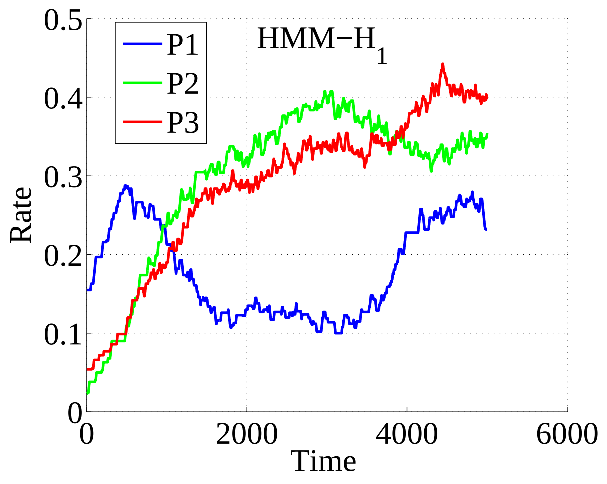


Figure 4.3: (Left): Rates of the states in Fig. 4.2 for two sequences recognition with both matched and unmatched observation using pulse based HMM in software and schematic simulation. Time scale: 0.4ms/step. (a): output rates of HMM state $H1$ software simulation. (b): output rates of HMM state H_1 cadence simulation. (c): output rates of HMM state H_2 software simulation. (d): output rates of HMM state H_2 cadence simulation. (right): A window of the output pulses for the three states for HMM H_2 which view the sparse of pulses between state s_1 as Lp_1 rate , state s_2 as Lp_2 rate and state s_3 as Lp_3 rate

received, here $N = 4$, $p(x|H_1)$ can be estimated according to

$$log(p(x|H_1)) \approx L(p_3[N]) \tag{4.18}$$

Which is then compared against the likelihood scores generated by other HMMs before the decision is made. Fig. 4.3 shows the instantaneous firing-rate computed for each of the modules S_1, S_2, S_3 in HMMs H_1 and H_2 when an observation sequence $x = \{1, 3, 3, 2\}$ is received. For this particular experiment, only H_1 is trained to recognize this sequence (which can be easily verified by inspecting the observation matrix. As described in equation 4.9, the final likelihood score can be estimated using $L(p_3[N])$ which is the firing-rate of the third IF module of third state. As shown in Fig.4.3, the HMM H_1 which is trained to recognize the observation sequence produces a higher probability (output p_3) compared to output produced by HMM H_2 for both the software and schematic simulations. In other words the output rate of recognizing HMM H_1 in the third state is higher than that for HMM H_2. Fig 4.3 (a, b, c, d) shows successively the output rates of HMM H_2 states in software simulation, the output rates of HMM H_2 states in cadence simulation, the output rates of HMM H_1 states in software simulation and the output rates of HMM H_1 states in cadence simulation respectively. In the same Figure (right) a window of $60ms$ is shown for the output pulses (circuit simulation) for the three states of HMM H_2 and H_1 .



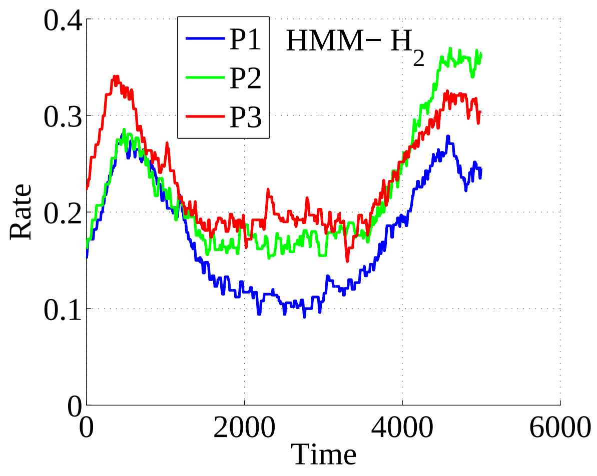


Figure 4.4: Instantaneous firing-rates for neurons N_1 to N_3 using the IFMP network implementing HMM $\mathcal{H}_1$(left) and HMM $\mathcal{H}_2$ (right) for the observation sequence {1,3,3,2}.

Another experiment is implemented for the same input sequence {1,3,3,2} shows a successful detection. The result of the experiment is shown in Fig. 4.4. The figure represents the moving average window for the probabilities over the time operation which shows that the probability of the third state is higher in HMM H_1 than HMM H_2. Probability of third state is the resultant sum of product over the observations, transition probabilities and iterations. It is shown in the figure the instantaneous firing-rates for neurons N_1 to N_3 using the IFMP network implementing HMM $\mathcal{H}_1$(Left) and HMM $\mathcal{H}_2$ (right) for the observation sequence {1,3,3,2}.

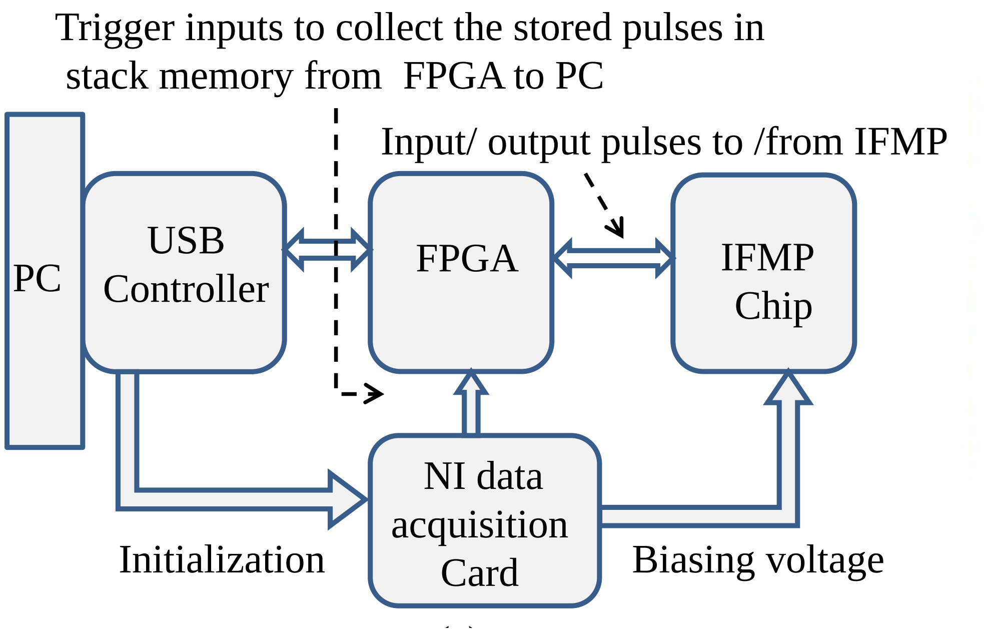
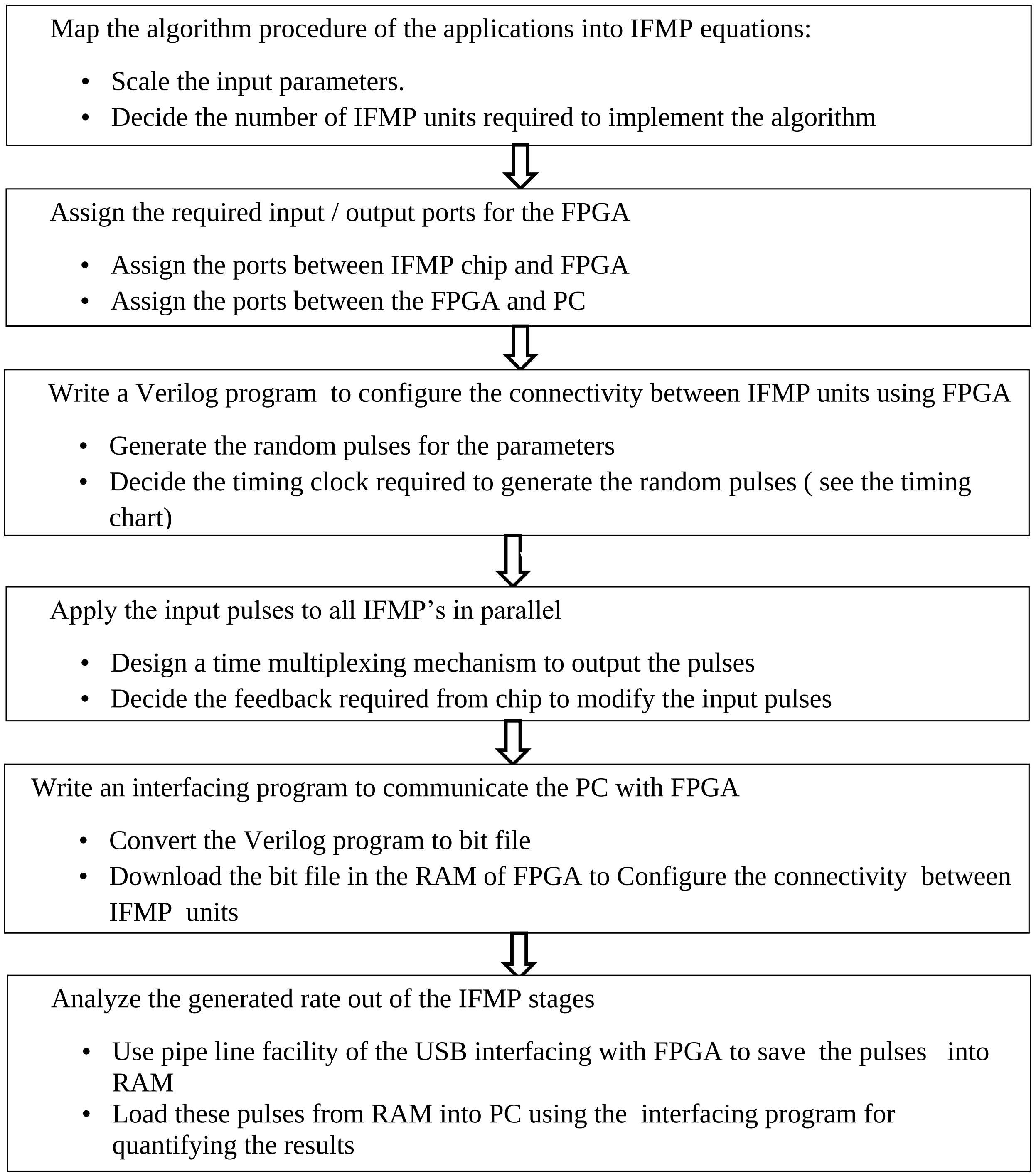

Trigger inputs to collect the stored pulses in stack memory from FPGA to PC
Input/ output pulses to /from IFMP
PC
USB Controller
FPGA
IFMP Chip
NI data acquisition Card
Initialization
Biasing voltage

Figure 4.5: Interfacing boards for testing the IFMP chip. The PC and FPGA are used to generate random pulses/getback the output pulses to/from the IFMP circuit. The National instrument interfacing card is used to generate voltage biasing to the transistors of the IFMP as well to intiate the stack memory to pull the stored pulses to the PC

4.2.1 Chip simulation

For the hardware verification of the IFMP pulse computational modules, the sequence detection example is implemented using arrays of IFMP's on a chip. The units of the IFMP in the array are chosen using row and column decoders. The IFMP chip is interfaced to the FPGA and National instrument (NI) ADC/ DAC boards shown in Figure 4.5 to test and


Map the algorithm procedure of the applications into IFMP equations:

- Scale the input parameters.
- Decide the number of IFMP units required to implement the algorithm

Assign the required input / output ports for the FPGA

- Assign the ports between IFMP chip and FPGA
- Assign the ports between the FPGA and PC

Write a Verilog program to configure the connectivity between IFMP units using FPGA

- Generate the random pulses for the parameters
- Decide the timing clock required to generate the random pulses (see the timing chart)

Apply the input pulses to all IFMP's in parallel

- Design a time multiplexing mechanism to output the pulses
- Decide the feedback required from chip to modify the input pulses

Write an interfacing program to communicate the PC with FPGA

- Convert the Verilog program to bit file
- Download the bit file in the RAM of FPGA to Configure the connectivity between IFMP units

Analyze the generated rate out of the IFMP stages

- Use pipe line facility of the USB interfacing with FPGA to save the pulses into RAM
- Load these pulses from RAM into PC using the interfacing program for quantifying the results

Figure 4.6: Flowchart of the supported software needed to implement applications using IFPG board and the array of IFMP modules on a chip

implement the HMM algorithm. The NI instrument is used to apply the voltage biasing to the IFMP chip while the FPGA system is used to apply inputs to the IFMP units, route the pulses between the IFMP units and store the output pulses of the IFMP units for analysis and verification of results. In this experiment, the input parameters are applied in parallel to the three HMM state machine discussed earlier (the three IFMP modules). The inputs are generated using pseudo random generators written in verlog hard description language (Verlog HDL). While the inputs are applied in time sharing strategy for each input pulse period, the output pulses from the three states are considered as feedback signals according to the forward HMM equations.

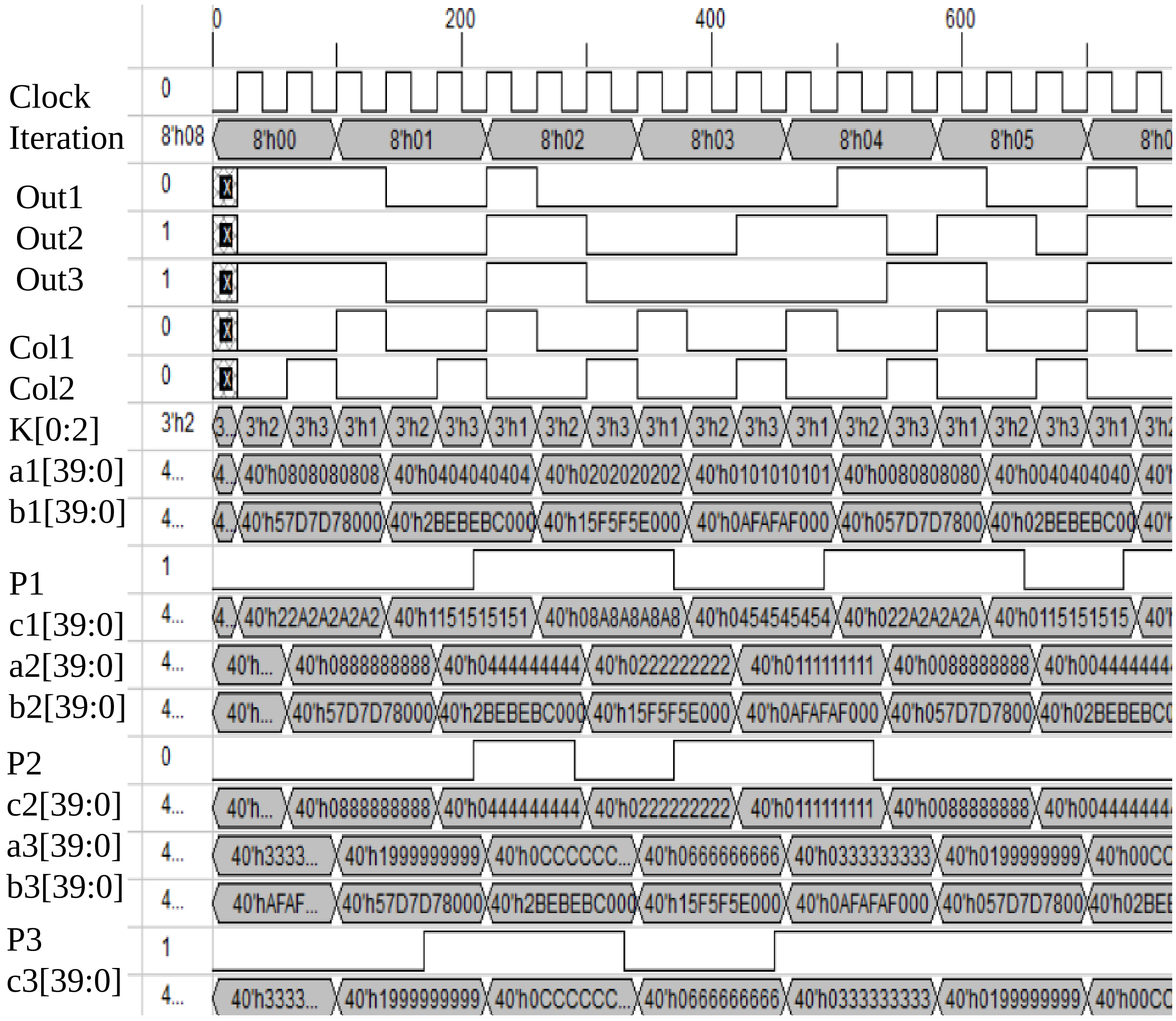


Figure 4.7: Timing simulation of Verilog program to generate random pulses to the three units of IFMP in time sharing mode where col1 and col2 select the IFMPunit on the chip

The flow chart shown in Figure 4.6 explains the procedure needed to implement applications using IFMP chip supported by the FPGA board as a device to select the connectivity and route the pulses between the IFMP units on a chip according to a particular application. The flow chart for implementing applications started by scaling the input parameters and assigning the iteration number of the random input pulses. Then the interfacing ports need to be assigned between the IFMP chip and FPGA as well as between FPGA and PC . Next, a verlog program is written to generate the pseudo random pulses in time-share multiplexing. The Time step of the generating pulses depend on the frequency of applying these pulses. The step time used in software simulation is the same step time used in hardware implementation which is equal to 0.2ms. Therefore, the input frequency of 5KHz is needed for the HDL program to generate the pseudo random pulses. By simulating the HDL program, In Figure 4.7, the following pulse signals are explained as,

1. Generated pulse streams for the IFMP states(a1, b1, c1, a2, b2, c2, a3, b3 and c3): in the timing diagram, three random input rates are generated per IFMP state. Two of these inputs (a1, b1) are the excitatory inputs of the first two IF model of the IFMP, while the third input (c1) is the gamma constraint applied to the inhibitory input of the third neuron of the IFMP. The other inputs (a2, b2, c2, a3, b3 and c3) are applied in the same way to the other two IFMP states. For experimental purpose to verify the HMM operation in pulse mode, these nine inputs are generated using pseudo random generator. These nine inputs are applied in parallel to the three IFMP units in time share multiplexing mode where each pulse width represent a time of 0.2ms which is the step time implemented in software simulation and circuit simulation.

2. The output pulses from the IFMP states (P1, P2 and P3): these outputs are fed back

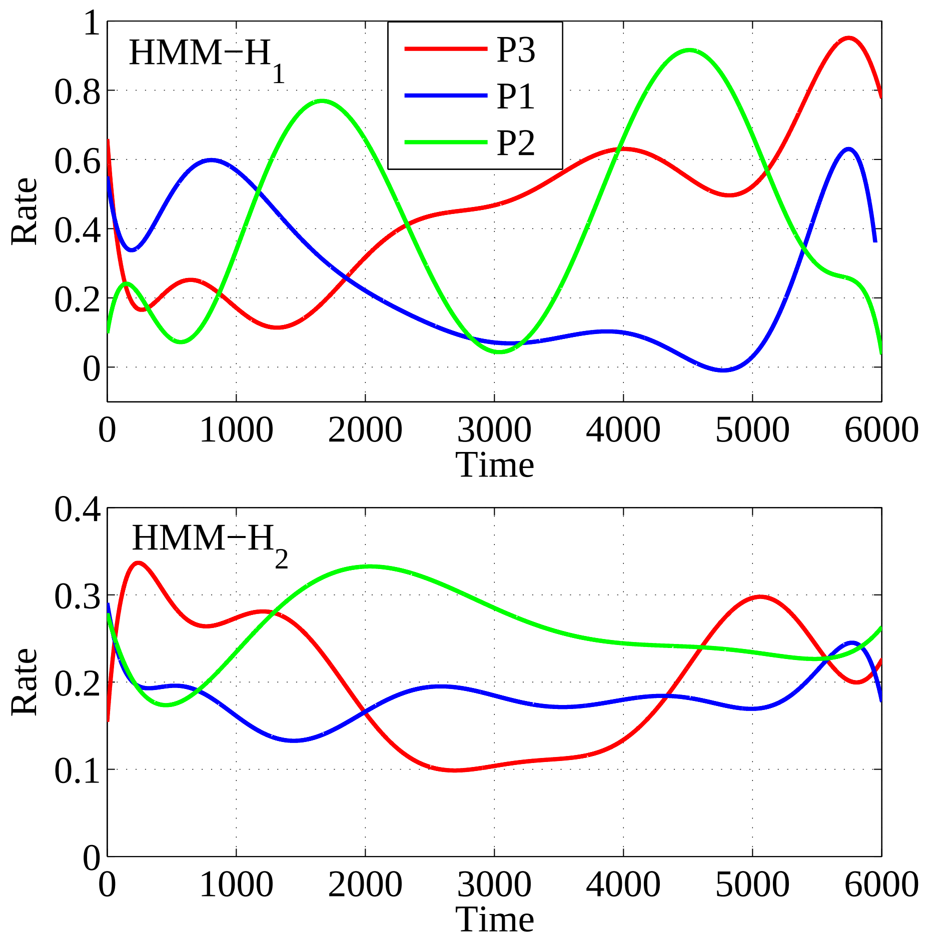


Figure 4.8: The moving average window for the three output states of HMM with two observations: the HMM-H1 and HMM-H2 respectively

and ORED with the input pulses to the IFMP units according to the HMM states algorithm.

3. The selection signals k, col1 and col2: These signals are used to select a specified IFMP unit in the IFMP array.

4. The clock pulse: this input signal is chosen equal to 5KHz which is the same frequency used in the software and circuit simulation.

5. The iteration signal: this signal is to repeat the pseudo random pulses inputs.

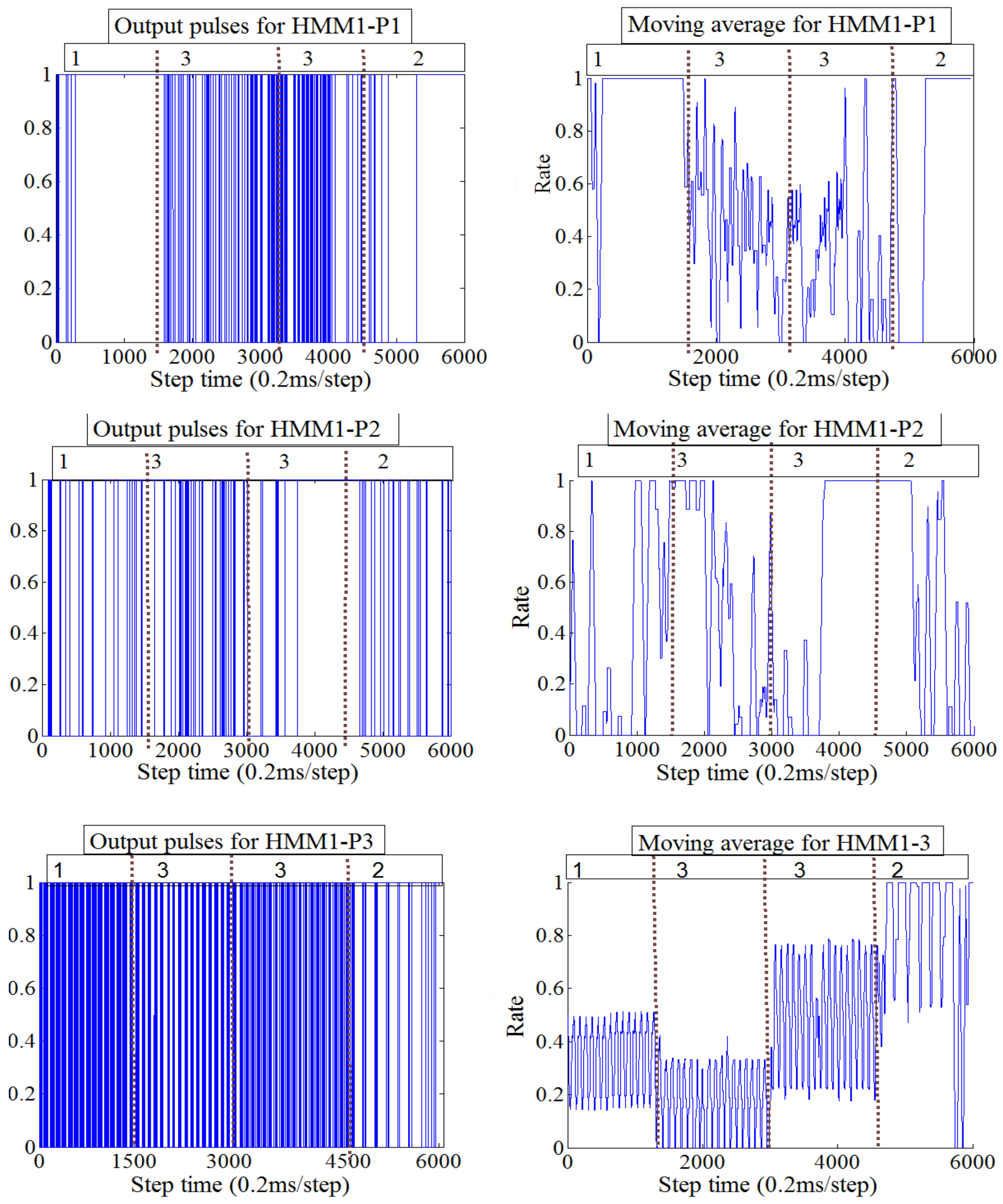


Figure 4.9: The output pulses and moving average window for the three states for HMM-H1

The last steps in the flow chart is to write an interfacing program in Matlab to communicate the PC with the FPGA, configure the HDL program on FPGA and analyzing the generated rates from the IFMP stages. The last step is utilized using the pipe line facility of software functions (designed by opal kelly to interface the PC with FPGA through USB) to save the pulses into RAM of the FPGA then load these pulses from RAM into PC using the interfacing program for quantifying the results.

An example of detecting a sequence of numbers is succeeded by evaluating the rates of the third states for the two observations HMM H_1 and HMM H_2. As the expected outputs in simulation, a higher probability or rate is observed using the input parameters of the first observation HMM H_1 (which is the matched observation for the input sequence [1,3,3,2]) rather than using the parameters of the second observation HMM H_2 as shown in Figure 4.8. In the figure, the P3 trace represents the probability of the third state of the Markov state machine, while the P1 and P2 traces represent the rate of the first and the second states respectively . In the experiment, the number of samples /input is equal to 1500 steps each has duration time of 0.2ms which is the same step time used in the software simulation in Matlab and in circuit simulation using cadence (discussed in the previous section). For both of the observations HMM H_1 and HMM H_2, it is shown respectively in Figures 4.9 and 4.10 the output pulses of the third state of HMM and the moving average window which from which the three traces of the three states clarify the matching detection of the HMM circuit.

To summarize this section, the implementation of HMM sequence decoding task is demonstrated in pulse based using analog circuit as cadence simulation and chip layout hardware which realize the modular characteristics of IFMP. Achieving MP based computation in spike-domain implies the possibility of implementing hybrid neuromorphic architectures where large-scale machine learning algorithms can now be integrated on spiking neural-

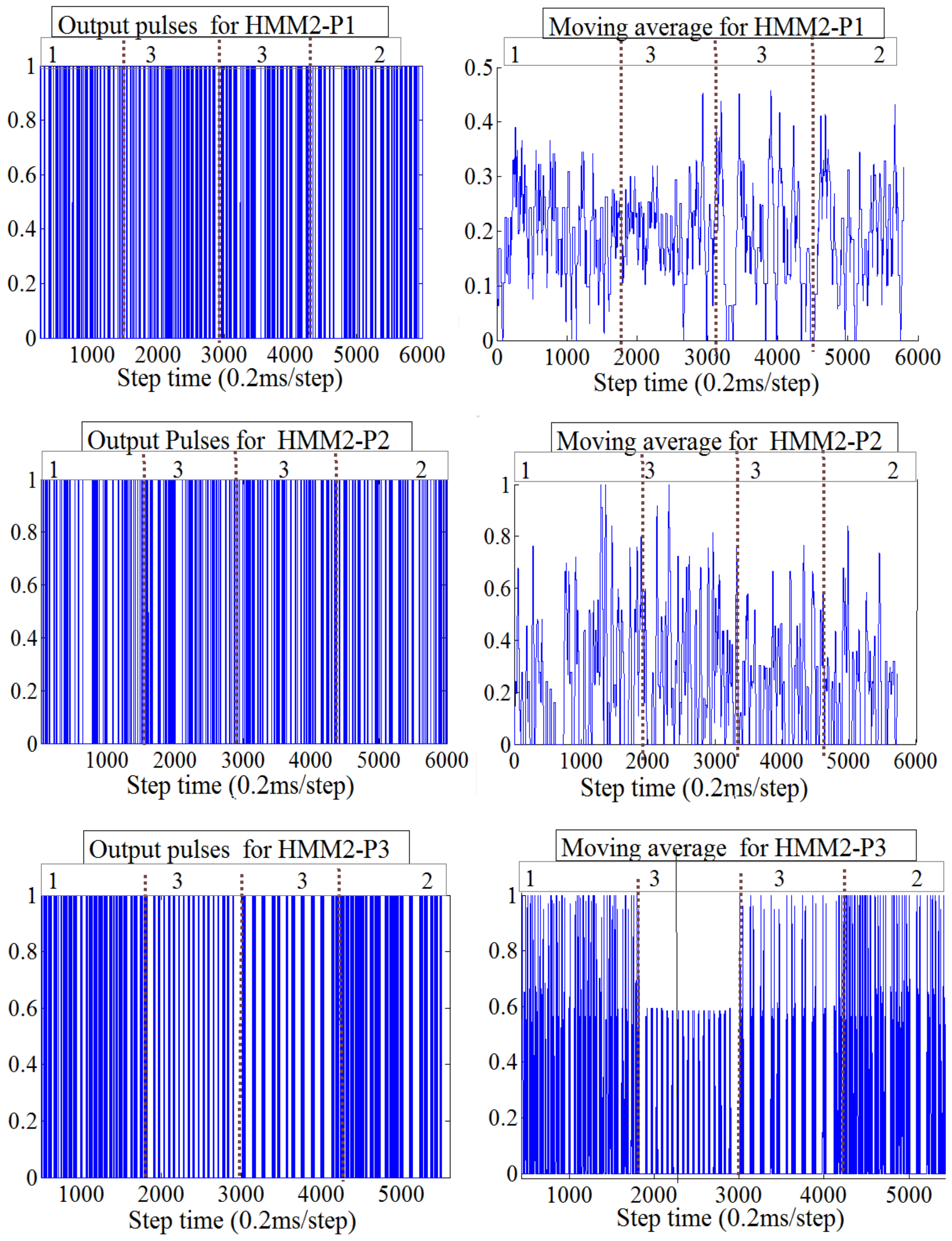


Figure 4.10: The output pulses and moving average window for the three states for HMM-H2

networks. The observation probability matrix in the above application shows discrete or fixed probability to the observed input items. In the next section, another detection application is verified in HMM model using IFMP pulse computation module where the observation probability are real values and obtained using Gaussian distribution.

4.3 Trajectory detection

In this application, the forward algorithm of HMM Model discussed in the sequence detection application is used to detect a trajectory traces. Since the trajectory has infinity points in the trace, the change in the curvature shape is considered as separate state. Each state is trained by selecting the mean and variance of the neighboring points using Gaussian distribution. Fig. 4.11 shows the states $s_j : j = 1 : 17$ of the HMM model used to detect eight trajectories $P_i : i = 1 : 8$. The model is left-to-right Markov models with three states per one path. Each state, j , is characterized by the following.

1- A state transition vector, a_i, with components a_{ji} = probability of making a transition to state i (at the next transition instant), given that the system is currently in state j.

2- A state observation continuous density, $b_j(O)$, of the form of,

$$b_j(O) = \frac{\prod_{d=1}^{D} exp[-(O(d) - \mu_{jd})^2/(2\sigma_{jd}^2]}{2\pi^{1/2}(\prod_{d=1}^{D} \sigma_{jd}^2)^{1/2}} \tag{4.19}$$

where O is the observation vector, μ_{jd} is the mean vector and σ is the standard deviation in state j and D ($D = 2$ for this experiment) is the component of dimension of μ_{jd}. In this application, the sets of mean and standard deviation for each trace is trained in order to find the standard deviations that are close to the trace. The mean, standard deviation and the observed dimension of the event are used to evaluate the observation probability. In the experiment, the standard deviation used for training the neighboring trace is chosen as 0.2

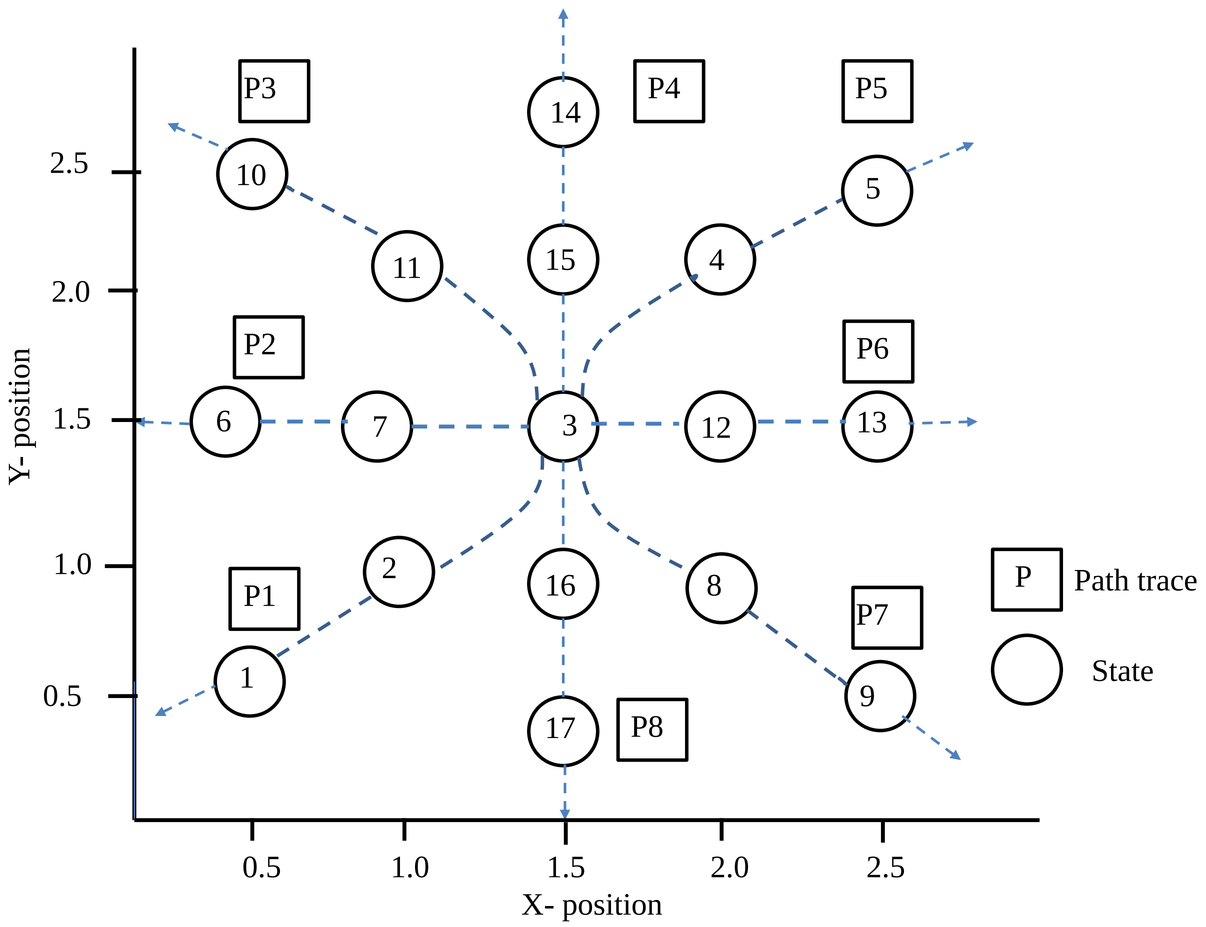


Figure 4.11: Traces for eight trajectories starting from state 3 using three HMM states per path

unit so that there is no conflict with other traces. The mean values in the two dimension graph shown in Fig. 4.11 are chosen for the eight paths as following,

1. Path1: {(1.5,1.5); (1.0,1.0); (0.5,0.5)}
2. Path2: {(1.5,1.5); (1.0,1.5); (0.5,1.5)}
3. Path3: {(1.5,1.5); (1.0,2.0); (0.5,2.5)}
4. Path4: {(1.5,1.5); (1.5,2.0); (1.5,2.5)}
5. Path5: {(1.5,1.5); (2.0,2.0); (2.5,2.5)}

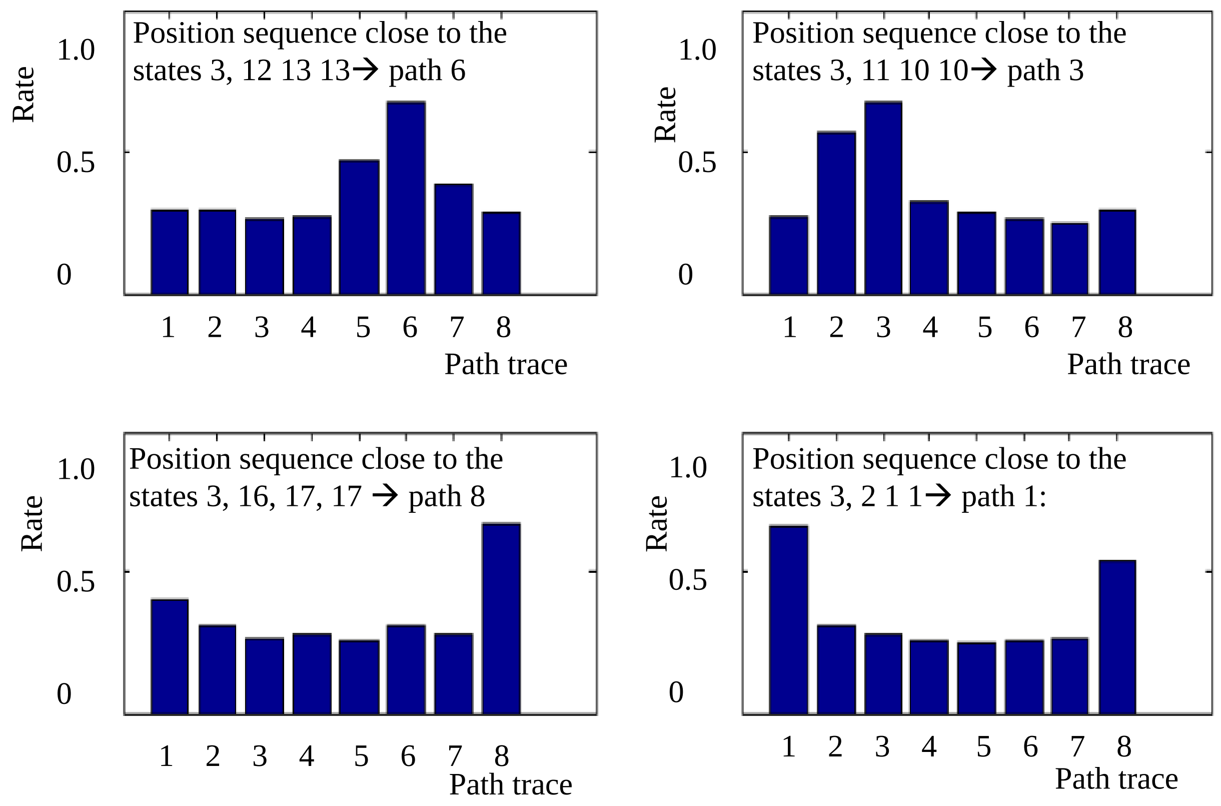


Figure 4.12: Four detecting experiments for different trajectory paths in Fig 4.11. The x-axis and Y-axis reprsent the eight traces and the probabilities for each trace path in the experiment

6. Path6: {(1.5,1.5); (2.0,1.5); (2.5,1.5)}

7. Path7: {(1.5,1.5); (2.0,1.0); (2.5,0.5)}

8. Path8: {(1.5,1.5); (1.5,1.0); (0.5,0.5)}

In order to check which path is belong to given points or traces, the points as an observation values are applied in Gaussian equation (4.19) to calculate the observation probability, given the variance σ_{jd} and the mean values for the paths listed above. Once the observation probability and the transition probability are known, then it is possible to apply the forward algorithm HMM to calculate the probability of the being in any states having the input observation of the trace point.

Thus, the observation probability and the transition probability for all traces are mapped into the recursive HMM-IFMP formula to decide the highest probability for the observed event or trace among the eight traces as shown in Fig. 4.12. In this application, four experiments are implemented to check the path to four different input traces. In the first experiment, the sequence of the trajectory points near by the states 3, 12, 13, 13 are required to be detected and decide the closest path. For this purpose, the HMM model is implemented to find the probabilities for the eight paths. In this experiment, it is shown the max probability matches at path 6 which is the closest path to the input trace points as shown in the trace figure 4.11 and 4.12. The same procedures are implemented for the other three experiments to detect respectively the trajectory points near by the following states [3,11 ,10 ,10], [3 ,16 ,17 ,17] and [3 ,2 ,1 ,1]. The figure shows that the maximum probabilities for the above traces are corresponding paths 3, 8, and 1 respectively.

4.4 Support vector machine

Support vector machines (svm) are one of the most popular supervised learning instances. The supervised learning methods are comprised of two discreet phases, the training and the classification [80]. The svm training phase is responsible for the identification of these data points, called support vectors, that can best build a separation model for the classes. These vectors are then used to predict the class of any future data point during the classification phase. In this section, a general idea of svm learning machine and a brief description of the learning and classification will be explained.

The General idea of classifying data using support vector machine it to map the data from the original input space into some higher-dimensional feature space where the training set is separable. For example in Fig. 4.13 (a), the data is mapped from one dimension into two dimension space and in Fig. 4.13 (b), the data is mapped from two dimension into three dimensional space. Every data point is mapped into high-dimensional space via some transformation $\phi : x \longrightarrow \phi(x)$ such that:

$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ where $K(x_i, x_j)$ is some function called a kernel function that corresponds to an inner product in some expanded feature space. Example: Two dimensional vectors $x = [x1 \quad x2]$; let $K(x_i, x_j) = (1 + x_i^T x_j)^2$,

Starting from the second order polynomial kernel $K(x_i, x_j) = (1 + x_i^T x_j)^2$, it can be shown that $K(x_i, x_j) = \phi(x_i)^T (x_j)$ as following,

$$\begin{aligned}
K(x_i, x_j) &= (1 + x_i^T x_j)^2 \\
&= 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} \\
&= [1 \quad x_{i1}^2 \quad \sqrt{2}x_{i1}x_{i2} \quad x_{i2}^2 \quad \sqrt{2}x_{i1} \quad \sqrt{2}x_{i2}]^T * \qquad (4.20) \\
&\quad [1 \quad x_{j1}^2 \quad \sqrt{2}x_{j1}x_{j2} \quad x_{j2}^2 \quad \sqrt{2}x_{j1} \quad \sqrt{2}x_{j2}] \\
&= \phi(x_i)^T \phi(x_j)
\end{aligned}$$

where $\phi(x) = [1 \quad x_1^2 \quad \sqrt{2}x_1 x_2 \quad x_2^2 \quad \sqrt{2}x_1 \quad \sqrt{2}x_2]$

The kernel function types listed below are used in training and classification in svm,

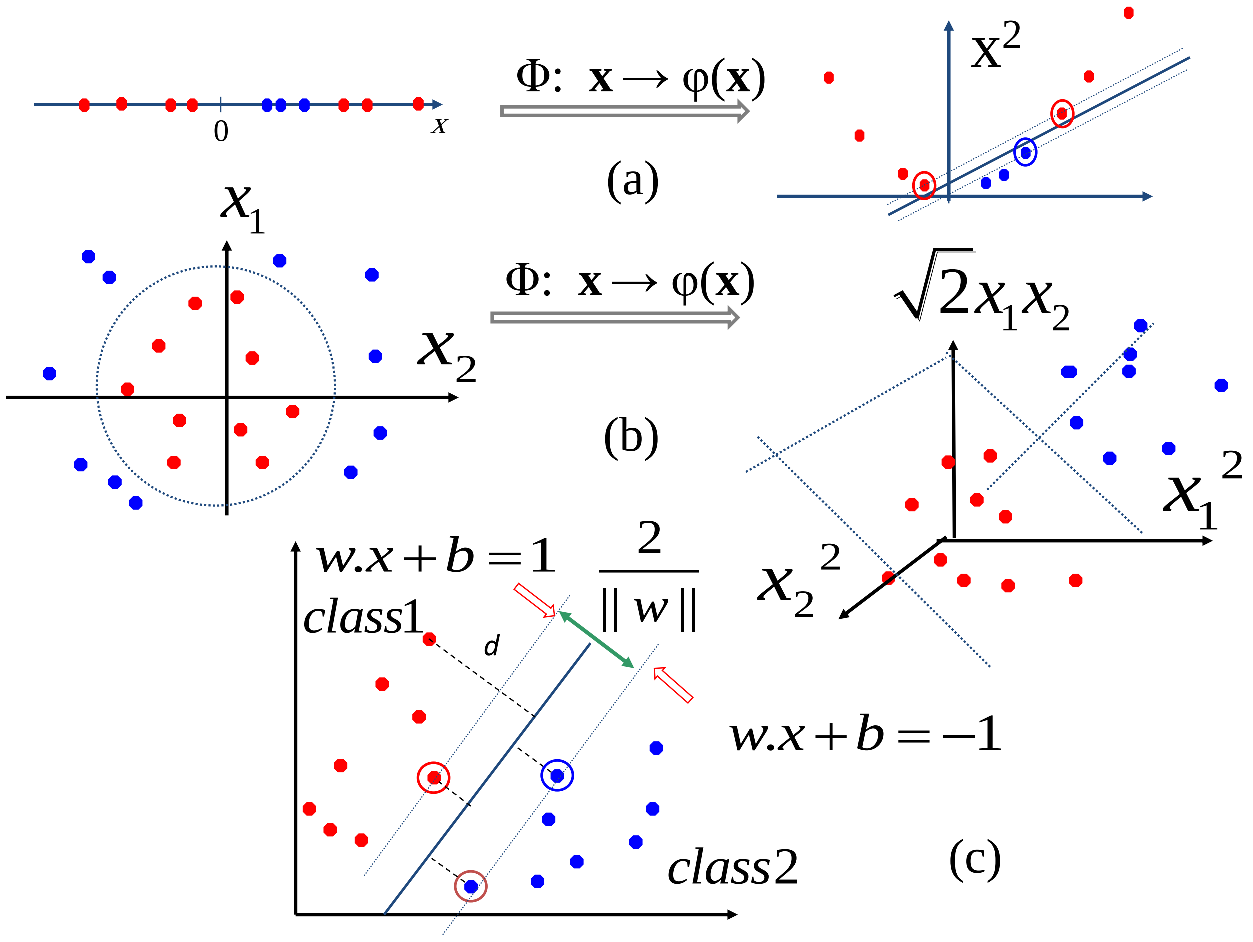


Figure 4.13: (a, b): Two examples of mapping a non separable data (left) into a linearly separable data (right) by mapping the lower dimension data into higher dimensions. (c): Binary classification based on linear huperplan svm

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial of power p: $K(x_i, x_j) = (1 + x_i^T x_j)^p$
- Gaussian (radial-basis function network): $K(x_i, x_j) = exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$
- Sigmoid: $K(x_i, x_j) = tanh(\beta_0 x_i^T x_j + \beta_1)$

In the IFMP application for classification, a second order polynomial is implemented as will be discussed in the next section.

The svm training builds a model that is able to distinguish the belonging class of any future data based on the support vectors obtained by the training data set. On a two-class classification problem, the svm objective is the construction of a separating hyperplane $w.x - b = 0$ to attain maximum separation between the classes, as shown in Fig. 4.13(c). The classes' hyperplanes are parallel to the separating one, lying on each of its sides. The Euclidean distance between the two hyperplanes is $\frac{2}{\|w\|}$, thus the objective of svm is to maximize the distance between the classes hyperplanes or, in other words, to minimize $\parallel w \parallel$

$$min\frac{1}{2} \parallel w^2 \parallel, \;\; s.t. \;\; y_i(w.x_i - b) \geq 1, \;\; 1 \leq i \leq N \tag{4.21}$$

where $(\cdot)$ denotes an inner product, x_i is the training data and label y_i denotes the belonging class of data x_i and takes the values -1 or 1. While w is a perpendicular vector to the hyperplane direction, b is the offset to the origin, and N is the training set size. The svm training phase focuses on the identification of the support vectors, which are the training samples that lie closest to the hyperplane and determine its direction. Many optimization algorithms have been proposed to find the parameters of svm [81, 82, 83]. Whenever the training data are not linearly separable in the input space, the input space is mapped to a higher-dimensional one, where a linear separation may be feasible. Support vector Machine employs kernel functions discussed above $K(x_i, x_j)$ to replace the inner products in the optimization problem 4.21

$$min\frac{1}{2} \parallel w^2 \parallel, \;\; s.t. \;\; y_i(k(w, x_i) - b) \geq 1, \;\; 1 \leq i \leq N \tag{4.22}$$

On the classification phase, any new data x is classified according to the output of the decision function;

$$F(x) = \mathrm{sgn}(\sum_{i=1}^{N_{sv}} y_i \alpha_i k(x_i, x) + b) \tag{4.23}$$

where α_i and b are the optimization parameters and "N_{sv}" is the set of the support vectors identified in the training phase. It is easily derived that the computational time of the non-linear svm classification task is linearly dependent on the size of the classification dataset, on the SV population "N_{sv}" and on the problem's dimensionality. In the next section, the svm classification using the IFMP computation modules will be shown.

4.5 Classification using support vector machine

A multi-class svm classifier computes matching scores (kernels) between the input vectors $\mathbf{y}$ and a set of template vectors (support vectors) $\mathbf{x}_i$. Then the kernel of the two vectors x and y are combined linearly to produce a score corresponding to class j according to

$$f_j(\mathbf{y}) = \sum_{i=1}^{N_{sv}} \alpha_{ij} k(x_i, y) + b_j \tag{4.24}$$

where b_j is a class specific bias parameter and α_{ij} are the weights corresponding to the support vector x_i and class j. In this work, a quadratic kernel $K(\mathbf{x}_i, \mathbf{y}) = (\mathbf{x}_i \cdot \mathbf{y})^2$ is implemented, satisfying the Mercer condition [Mercer condition only tells whether the kernel function is actually an inner product in some space and therefore admissible for use in a support vector machine. However, it tells nothing how to find $\phi(x)$. To find support vectors, there are different training packages for training the input data based on the optimization problem to maximize the marginal distance from the hyper plane] [84]. The svm training procedure automatically selects support vector templates from the training examples, and derives values for the weights α_{ij} and the bias parameter b_j accordingly. During the training procedure, several hardware related constraints can be imposed. For instance, all the the weights, biases and support vectors are positive so that they can be mapped directly onto an IFMP architecture using the inner-product in the kernel $(\mathbf{x}_i \cdot \mathbf{y})$. The polynomial operation $(.)^2$ in the kernel is mapped onto multiplication because IFMP computations operate in a logarithmic domain. For a spiking architecture, the mapping simply results in multiple fan-outs (copies) of the spike-train. The multiplication of the weights α_{ij} with the kernels according to equation 4.24 can be viewed as an inner-product computation and hence the mapping to IFMP described in chapter 2 is also applicable.

As an example of applying IFMP as pulse computational unit, a binary linear classifying has been implemented using svm algorithm. The following procedure shows how to derive the parameters to be applied in the IFMP based svm experiment: Let y denotes a vector drawn from the input space and let $\{\phi(y_j)\}_{j=1}^{m_1}$ denotes a set of non-linear transformations from the input space to a feature space , m_1 is the dimension of the feature space. For such set of non-linear transformations, a hyperplane is defined as the decision surface as:

$$\sum_{j=i}^{m_1} \bar{\alpha}\phi(y_j) + b = 0 \tag{4.25}$$

where $\bar{\alpha}$ denotes the vector of non-linear weights connecting the feature space to the output space, and b is the bias. In this example, a second order polynomial kernel of inner product is introduced between the input vector y and support vectors x to decide the hyperplane as

$$z = \sum_{i=1}^{N} \alpha_i (x_i.\bar{y})^2 + b \tag{4.26}$$

To find the parameters for the IFMP units, the logarithm function of both sides of the above equation must be applied as following,

$$L_z = L(\sum_{i=1}^{N} \alpha.(x_i.\bar{y})^2 + b) \tag{4.27}$$

$$L_z = L(\sum_{i=1}^{N} e^{(\log(\alpha_i)+2.\log(x_i.\bar{y}))} + e^{(\log(b))}) \tag{4.28}$$

where N represent the number of support vectors. The parameters for margin (M_1) are expressed as,

$$M_1(\{\log(\alpha_i) + 2.\log(x_i.\bar{y})\}, \log(b), \gamma) \tag{4.29}$$

Now, the kernels k_i are calculated as the Margin parameter (M_2) for the inner product term such that

$$K_i = \log(\sum_{j=1}^{m} x_{i,j}.y_j) \tag{4.30}$$

$$K_i = \log(\sum_{j=1}^{m} e^{(\log(x_{i,j})+\log(y_j))}) \tag{4.31}$$

Therefore, the parameters for Margin (M_2) are the set of kernels K_i such that,

$$K_i = M_2(\{\log(x_{i,j}) + \log(y_j)\}, \gamma) \tag{4.32}$$

Since the values of bias b and optimization factors α_i might be negative, these values are simulated in differential form ($b^+ - b^-$ and $\alpha_i^+ - \alpha_i^-$). Therefore, equation 4.27 is derived for two differential form as following,

$$L_{z+} = L(\sum_{i=1}^{N} \alpha^{+}.(x_i.\bar{y})^2 + b^{+}) \tag{4.33}$$

$$L_{z-} = L(\sum_{i=1}^{N} \alpha^{-}.(x_i.\bar{y})^2 + b^{-}) \tag{4.34}$$

The difference between the above two equations are used to decide the classification of data as shown in Fig. 4.14.

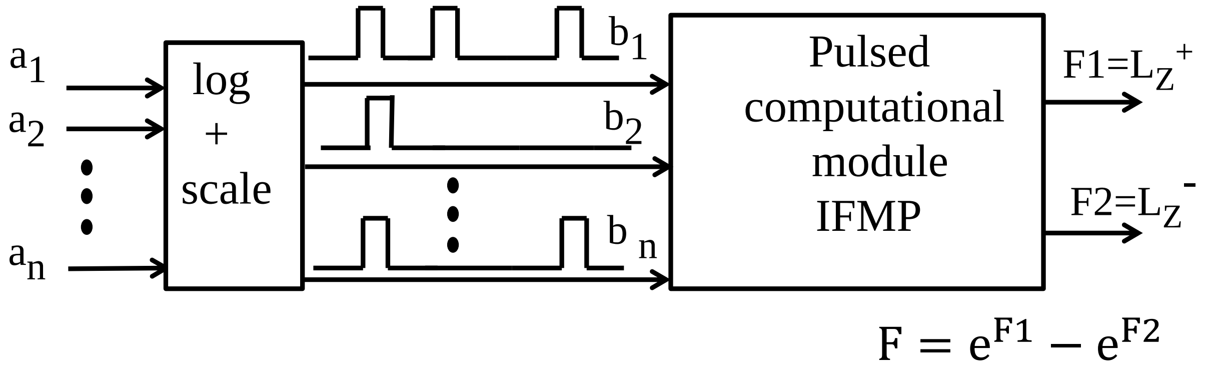


Figure 4.14: The classification using IFMP and support vector machine to compare between the differential outputs

As an example, the IFMP based svm network is applied for a binary classification task where the two-dimensional Iris data (shown in Fig. 4.16(a)) are used for training and validation. A "*Gini*" svm training algorithm is used to obtain the weights α_{ij} and identify the support vectors $\mathbf{x}_i$ [84]. The complexity of the IFMP depends on the number of support vectors and number of sparse data. In this application, nine support vectors are obtained from the training program " *Gini*" svm and 22 units of IF neural model are needed for the classification architecture. For this application, the architecture shown in Fig. 4.15 is designed for two - class classification using spiking svm network based on IFMP modules. The architecture follows a modular approach to compute first the kernel inner-product and secondly the inner-product between the weights and the kernels. The first computations shown in the architecture is implemented using nine IFMP modules to implement the kernel inner product with nine support vectors mentioned in equation (4.32). These nine units ($k_1 : k_9$) are common units between the two differential hyper plane equations (4.33) and (4.34). The input parameters for the positive hyper plan are : $b_1^+, \alpha^+, k_1 : k_9$, where k represents the kernels or the inner products of the support vectors and the input data to be classified. In the same way, the input parameters for the negative hyper plan are : $b_1^-, \alpha^-, k_1 : k_9$. For this architecture note the followings:

1. For the IFMP architecture, a quadratic operation is implemented using a simple fan-out which replicates the spike-train twice.

2. The two outputs z^+ and z^- in the figure are in log domain and they are corresponding to the output of the IFMP networks for each class.

3. All the inputs are quantified as random pulse rate.

Fig. 4.16(b) shows the raster-plot of the spikes generated by the neurons in the svm network [raster plot: is a fig that shows the occurrence of spikes for neurons in population with respect to time]. The spike-train at the top of the plot corresponds to one of the output of the network (neurons 11 and 22 in Fig 4.15) . Visually, it is inferred that when the vector corresponding to each of the classes is presented, the output spiking rate illustrates the network's discrimination capability. Fig. 4.17 shows the difference between the firing-rates $z^+ - z^-$ computed for the input data. The result shows that the network is correctly classifying both types of Iris petal data and hence the result validates the proof-of-concept of an IFMP-based spiking SVM.

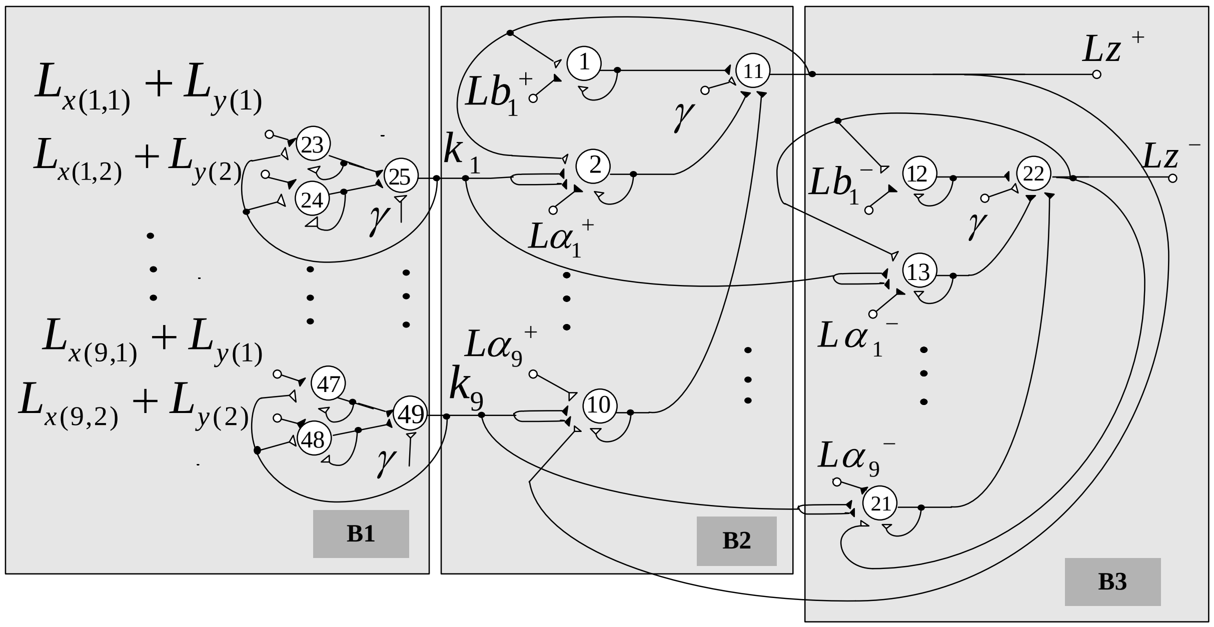
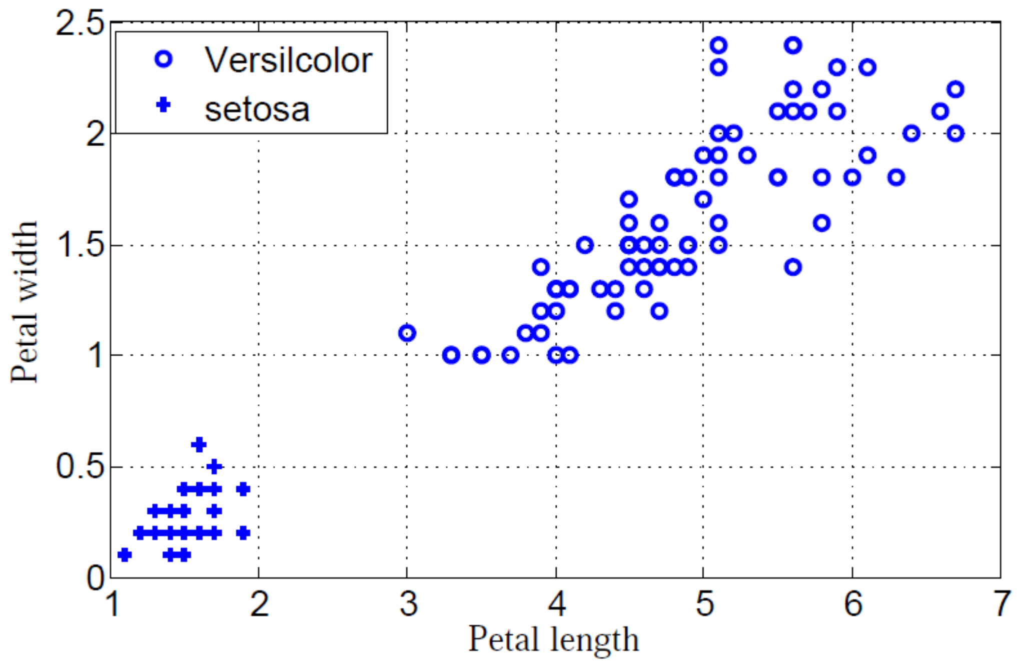
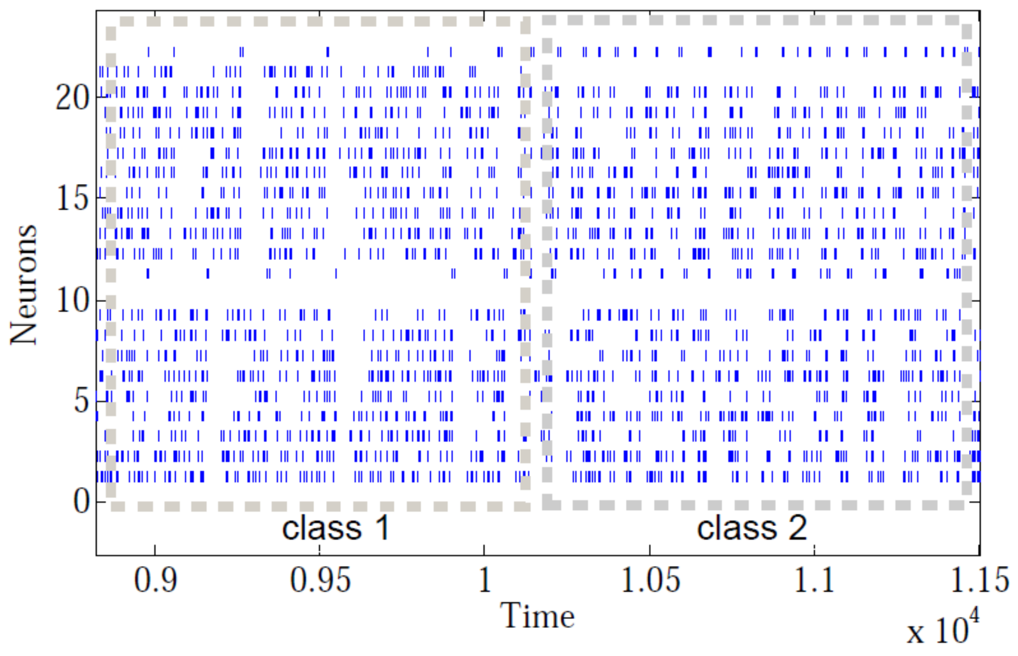


Figure 4.15: svm based classification using parllel cascading of IFMP modules. $k_1 : k_9$ are the kernels. The difference between the firing-rates of the output $z^+ - z^-$ determines the classification of input data. $B1$: Input vector (y) and support vector (x), $B2$: Kernel function (k), output decision (z^+) $B3$: Kernel function (k), output decision (z^-)

On the other hand the classification has been implemented for cascading architecture of two input- IFMP units shown in Fig. 4.18, where the support vectors are applied to a series of IFMP units with positive and negative values of alpha and bias values. Fig. 4.19 shows successively the classification of the data and raster- plot of the cascaded IFMP units as in the first case where neurons 30 and 81 are the output neurons of the two differential parts of the IFMP based SVM implementation. In this case, 81 neurons are used to simulate SVM with nine support vectors, nine values of alpha and one bias value, which are all in the differential form , to test the classification of input data.



(a)



(b)

Figure 4.16: Results obtained from the svm based classification using IFMP network. (a): Two dimensional Iris data set; (b): Raster-plot of spikes generated by the neurons in the network; and (c) Difference between the firing-rate of the outputs z^+ and z^-

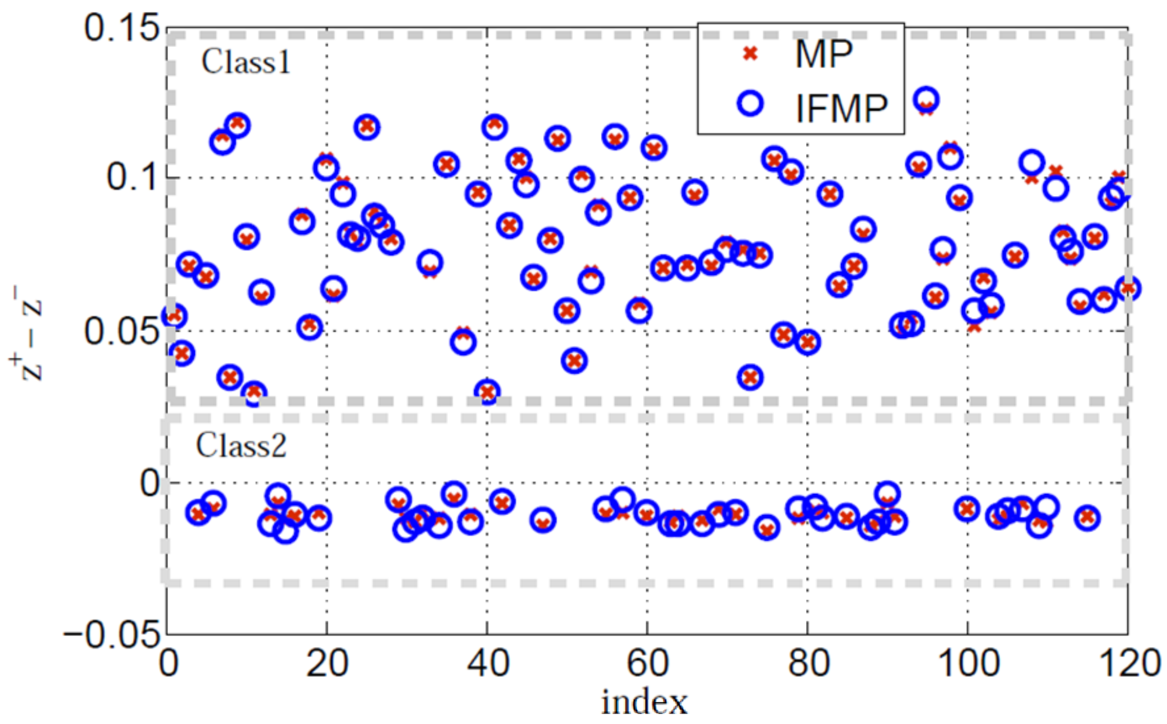


Figure 4.17: Continued- Difference between the firing-rate of the outputs z^+ and z^-

4.6 Summary

In this chapter, it is shown that the new designed pulse computational unit is not only approximating the basic nonlinear functions but it can also cascaded in serial architecture or parallel architecture to approximate more complicated algorithms. Three applications are implemented: the sequence decoding, the trace detection and data classification. The first two applications are implemented using forward recursion algorithm of hidden Markov model and the third application is implemented using support vector machine algorithm. It is not intended to implement these applications to replace artificial neural networks rather to show that the new designed pulse computation module is applicable in complicated algorithms that include state machine design and recursive equations in their procedures. Moreover, all the parameters in these applications are represented by the rate of a real time random pulses. Another important goal in this chapter is to verify that such applications can be applied in

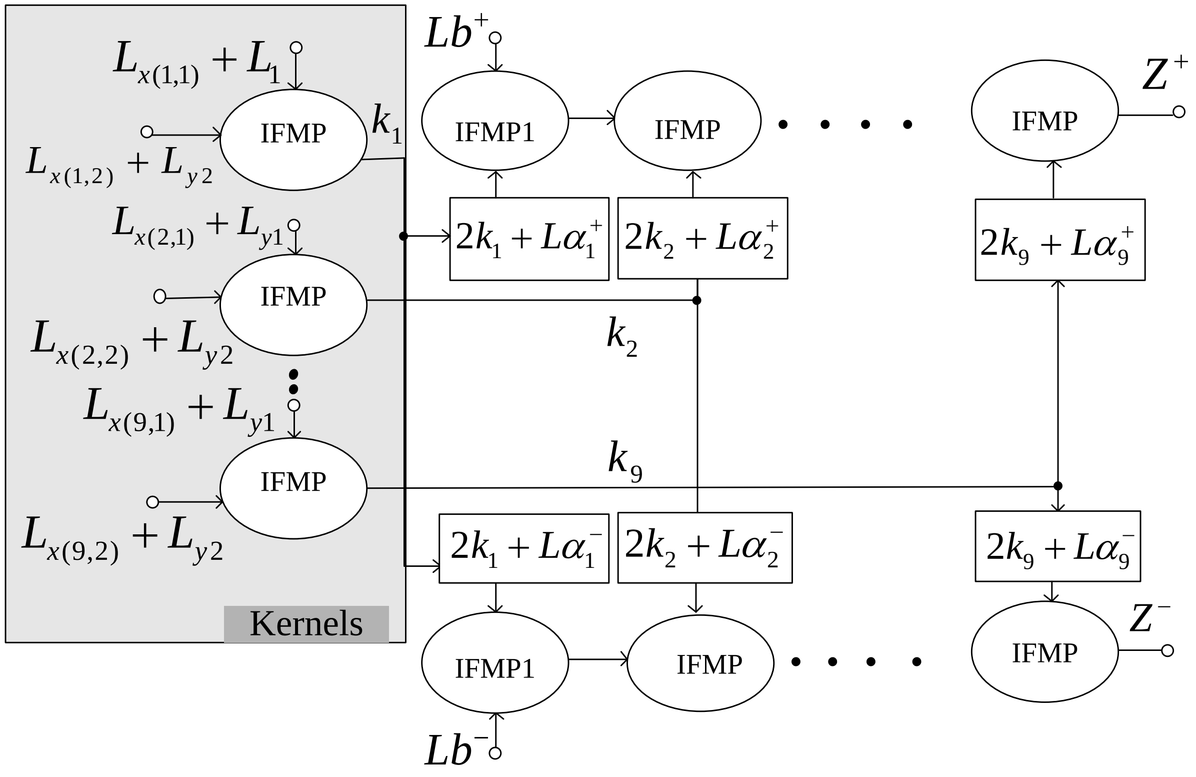
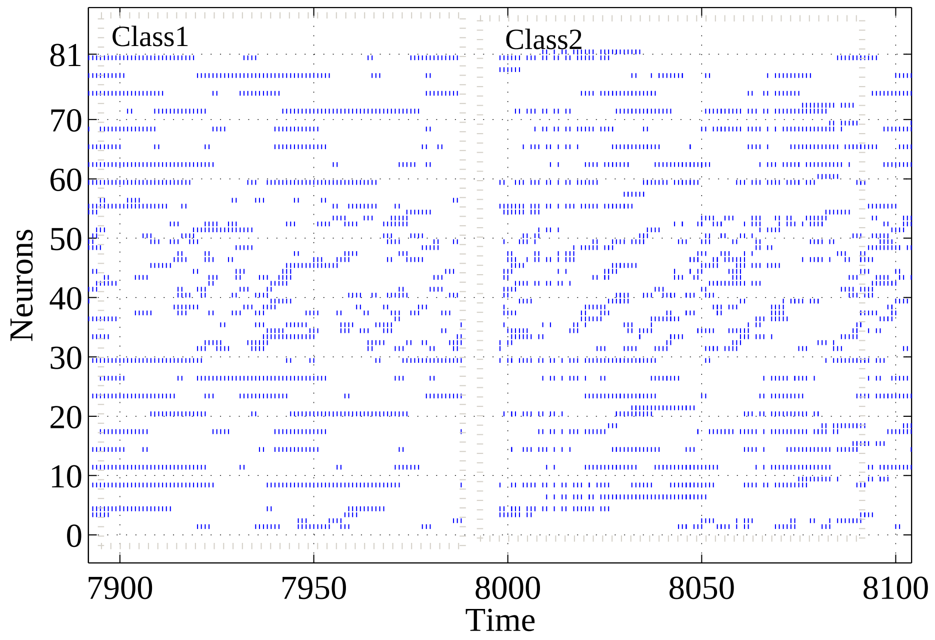
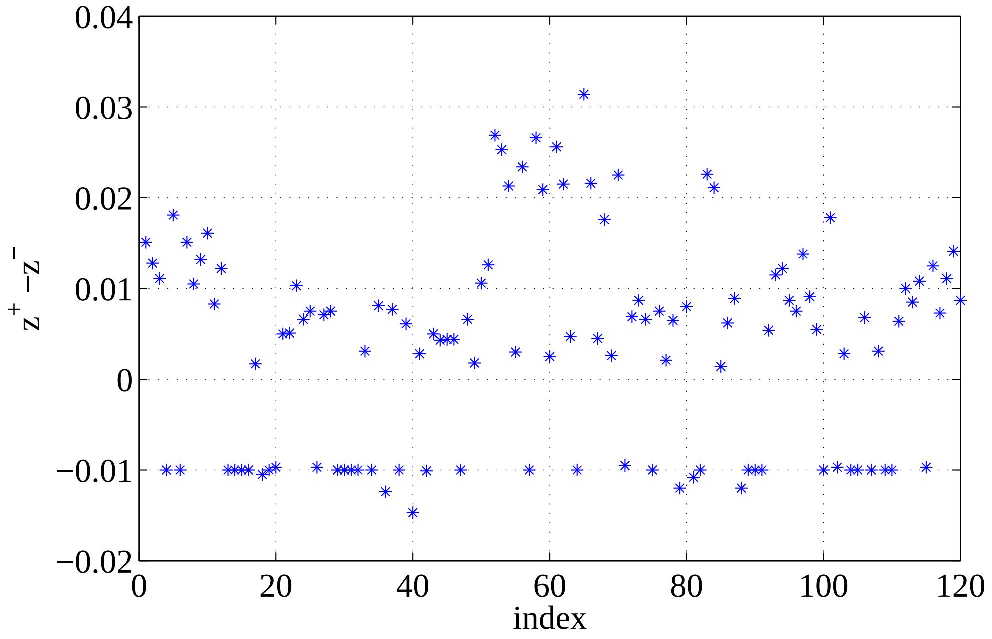


Figure 4.18: Classification for serial cascaded architecture using IFMP. $k_1 : k_9$ are the kernels. The biasing and weights (b and α) are in the differential forms to find the output z^+ and z^- that decides the classification of input data

hardware. In the hardware implementation, the designed IFMP algorithm is mapped and verified into analog circuit, and then design an architecture composed of IFMP array to implement procedural applications. The implementation of applications is possible because first the IFMP units are scalable modules and secondly the connectivity between modules are implemented using high speed programmable gate array "FPGA". The FPGA helps to apply high speed random pulses, consider the output pulses from the IFMP modules as feedback in processing and finally analyses the output rates according to the required algorithm.



(a)



(b)

Figure 4.19: (a): Classification for the serial cascaded architecture of two classes of Iris data set. The negative and positive values of $(z^+ - z^-)$ represent Setosa and Versilcolor types respectively using two models,the MP (red marker) and IFMP (plus marker). (b): Raster plot for spikes of neurons 30 and 81 decide the classification of two samples of input Iris data set: the first and the second half of the raster plot respectively

Chapter 5

Significance of IFMP

There are two significant roles of the new designed IFMP module in pulse computation algorithms. The first significant role is the enhancement of dynamic range in computations compared with an existed pulse-based stochastic computation applications. The second significant role is the direct and simple implementation of the ”winner take all” design compared with other existing designs. For the first significance of the designed IFMP module, it is shown with an example that the factor graph application reported to fail in pulse based stochastic computation regardless of the simplicity in hardware structure. In section one, a solution is suggested to this failure by using the IFMP pulse computation module. The second role of the IFMP designed module is compared with the existing ”winner take all” designs which are important in designing neuromorphic architecture such as the receptive field structures of retina and cochlea. A background of designing winner take all is presented starting from continuous time design presented by Lazzaro, 1989[85] to the spiking network design presented by; Oster, 2009), [86]. It is shown in section 2 with an example, the significant difference of spiking network design between Oster’s design and the new IFMP design.

5.1 Dynamic range enhancement in pulse computation

5.1.1 Applications of stochastic computations

Stochastic computations are pulse computational structures where the probabilities are represented as streams of random digital bits using Bernoulli sequences and the information is contained in the statistics of the bit stream. Using this representation, complex operations on probabilities such as multiplication and division are converted to operations on bits which can easily be manipulated using simple stochastic gates.

Stochastic computation is reported to implement hardware structures for some applications of factor graph algorithms. The factor graphs, introduced in chapter one, originally lie in the coding theory but they offer an attractive notation for a wide variety of signal processing problems. These problems are viewed as instances of the summary-product algorithm which operates by message passing in a graphical model. Specific instances of such algorithms include Kalman filtering and smoothing; the forward-backward algorithm for hidden Markov models; probability propagation in Bayesian networks; and decoding algorithms for error-correcting codes such as the Viterbi algorithm, the BCJR algorithm, and the iterative decoding of turbo codes, low-density parity-check (LDPC) codes, and similar codes [39][17].

However, stochastic decoding methods could only work either on very simple short codes or for decoding some specific error-correcting codes on trellis graphs. The reason for that is because the stochastic computations have false computations and might not be successful for other decoding applications. The false stochastic computations are caused by the similar pulsed-input codes in the multiplication operation and implementing scaling addition in the addition operation. In addition to that, the main problems in the stochastic computation are the long time convergence and low dynamic range of processed parameters. To address the low dynamic range problem, the new pulse computational module "IFMP" is introduced to map the input/output parameters in logarithmic domain.

In the next section, the basic units in stochastic computations are introduced in order to design the required dynamic or the recursive equations for the algorithmic applications. An application of sequence decoding using Hidden Markov model is implemented in both pulse-based stochastic computation and pulse- based IFMP modules to compare the performance in both cases.

5.1.2 Significant difference between IFMP and SC

In stochastic representation, the multiplication function is implemented with simply the "AND" logic gate. In Fig. 5.1 (a) the multiplication is implemented on two input values represented by stochastic bit streams. In the figure, with bit streams of length 8, the values have a resolution of 1/8. Assuming that the two input stochastic bit streams A and B are independent, then the rate c is represented by the output stochastic bit stream C as $P(C = 1)$ such that,

$$c = P(C = 1) = P(A = 1 and B = 1)$$
$$= P(A = 1)P(B = 1) = a.b$$

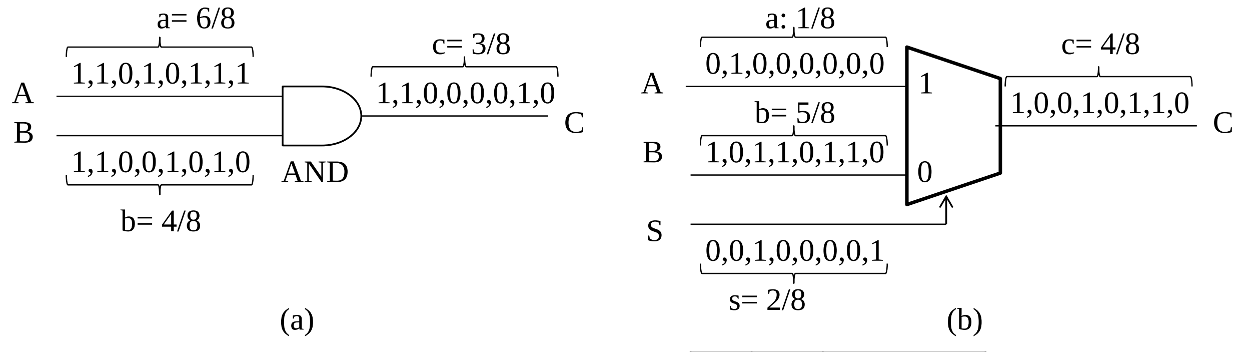


Figure 5.1: (a): Multiplication on stochastic bit streams with an AND gate. (b): Scaled addition on stochastic bit streams, with a multiplexer (MUX)

In stochastic computation, it is not possible to add two probability values directly because the output result might be greater than one, which cannot be represented as a probability value. Therefore, the normal addition is replaced by the scaled addition which is verified using multiplexer (MUX) . The multiplexer shown in Fig. 5.1 (b) is a digital construct that selects one of its two input values to be the output value, based on the selecting input value S. When $S = 1$, the output $C = A$. Otherwise, when $S = 0$, the output $C = B$. The Boolean function implemented by the multiplexer is $C = (A \wedge S) \vee (B \wedge \overline{S})$. With the assumption that the three input stochastic bit streams A, B, and S are independent, the number represented by the output stochastic bit stream C is

$$\begin{aligned} c &= P(C = 1) \\ &= P(S = 1 \wedge A = 1) + P(S = 0 \wedge B = 1) \\ &= P(S = 1)P(A = 1) + P(S = 0)P(B = 1) \\ &= s.a + (1 - s).b \end{aligned}$$

Thus, with this stochastic representation, the computation performed by a multiplexer is the scaled addition of the two input values a and b, with a scaling factor of (s) for (a) and $(1 - s)$ for (b).

Since the multiplication and the addition operations are verified in stochastic pulse computation, then it is possible to implement more complicated equations such as the recursive equations. It is intended in this section to map the forward HMM recursion algorithm (equations 4.7 and 4.8 in chapter 4) in stochastic computation to verify the sequence detection application. Recall that the HMM recursion equation is the following,

$$p_j[n] = \sum_{i \in C_j} p_i[n-1] a_{ij} O_{j,x[n]}$$

Then the above equation is expanded into the following three equations;

$$\begin{cases} p_1[n] = p_0[n-1]O_{1,x[n]} + p_1[n-1]a_{11}O_{1,x[n]} \\ p_2[n] = p_1[n-1]a_{12}O_{2,x[n]} + p_2[n-1]a_{22}O_{2,x[n]} \\ p_3[n] = p_2[n-1]a_{23}O_{3,x[n]} + p_3[n-1]a_{33}O_{3,x[n]} \end{cases}$$

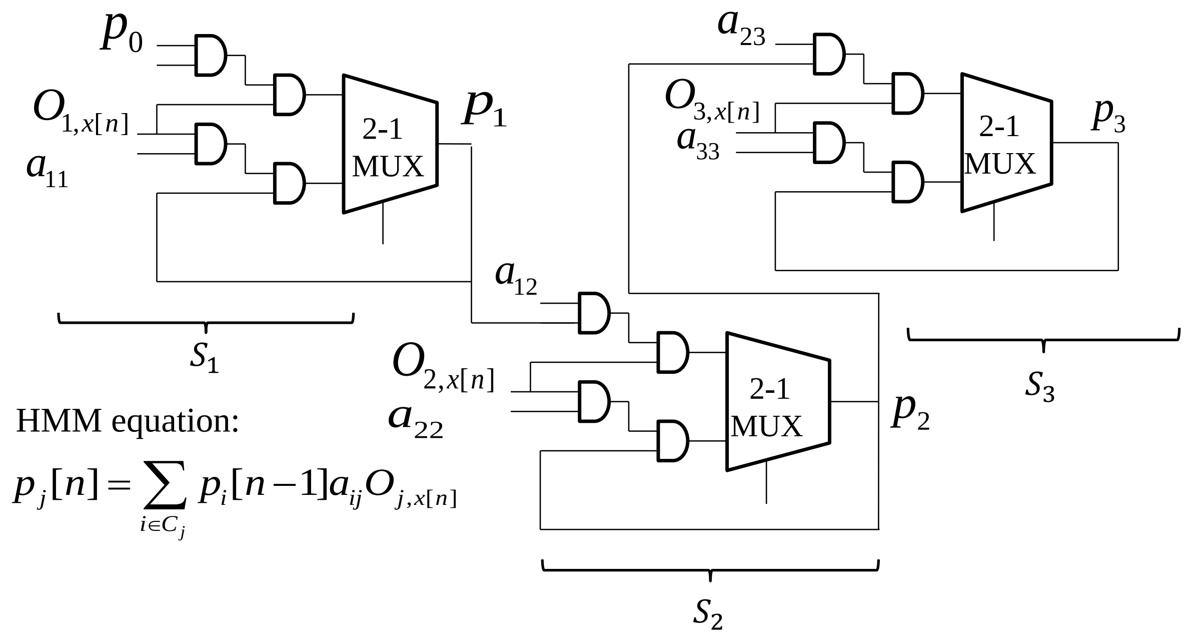


Figure 5.2: Stochastic computation to implement forward HMM algorithm HMM

The above three equations are simulated in stochastic computation using the AND logic units as multiplication operations and multiplexer units as the addition operations shown in Figure 5.2. This pulse-based architecture is compared with the architecture of pulse based IFMP computational modules discussed in chapter four and shown in Figure 4.2. The implemented example is to maximize a sequence that match the HMM observations. It is shown in the lower plots of Fig. 5.3, the output rates or probabilities of the three states of the HMM H_1 and HMM H_2 for the stochastic simulation which can not recognize a simple sequence [1,3,3,2]. It is shown in the figure that the output rates for HMM states are low rates that make it impossible to distinguish the input sequence if it is matched or unmatched with the given observations HMM H_1 and HMM H_2. The low rates of the HMM based stochastic computations are caused by the multiplication of the following probabilities in the forward HMM equation: observation probability, transition probability and the pre-time state probability. Multiplying the above probabilities reduce the probability more or reduce the dynamic range of the computations that make it difficult to recognize the posterior probabilities for different HMM observations. On the other hand, mapping the HMM recursive equation to the log-exp- sum or the margin propagation enhances the dynamic range and make it possible to distinguish the different HMM observation probabilities as shown in upper plots for Fig. 5.3. In the figure of IFMP implementation, the output rate of the third state $P3$ in the HMM H_1 observation is higher than the output rate in the case of observation HMM H_2. In the case of HMM H_1, it is shown how the the rate of P_3 increasing as the input sequence [1,3,3,2] are applied with respect to time. In addition to the dynamic range characteristics between the pulse based computations and the pulse base IFMP architecture, there are other differences in implementation such as the hardware dense, the power consumption and the rate convergence of the HMM states. Table 5.1 lists the differences in implementation between the pulse based stochastic computation structures and the pulse based IFMP computation modules.

5.2 Novel pulse-mode Winner take all (WTA)

The second significance of the new designed IFMP module is the simplicity and the direct implementation of the winner take all network. The definition of the winner take all concept and it's background in the design and the contribution of the new designed pulse mode IFMP

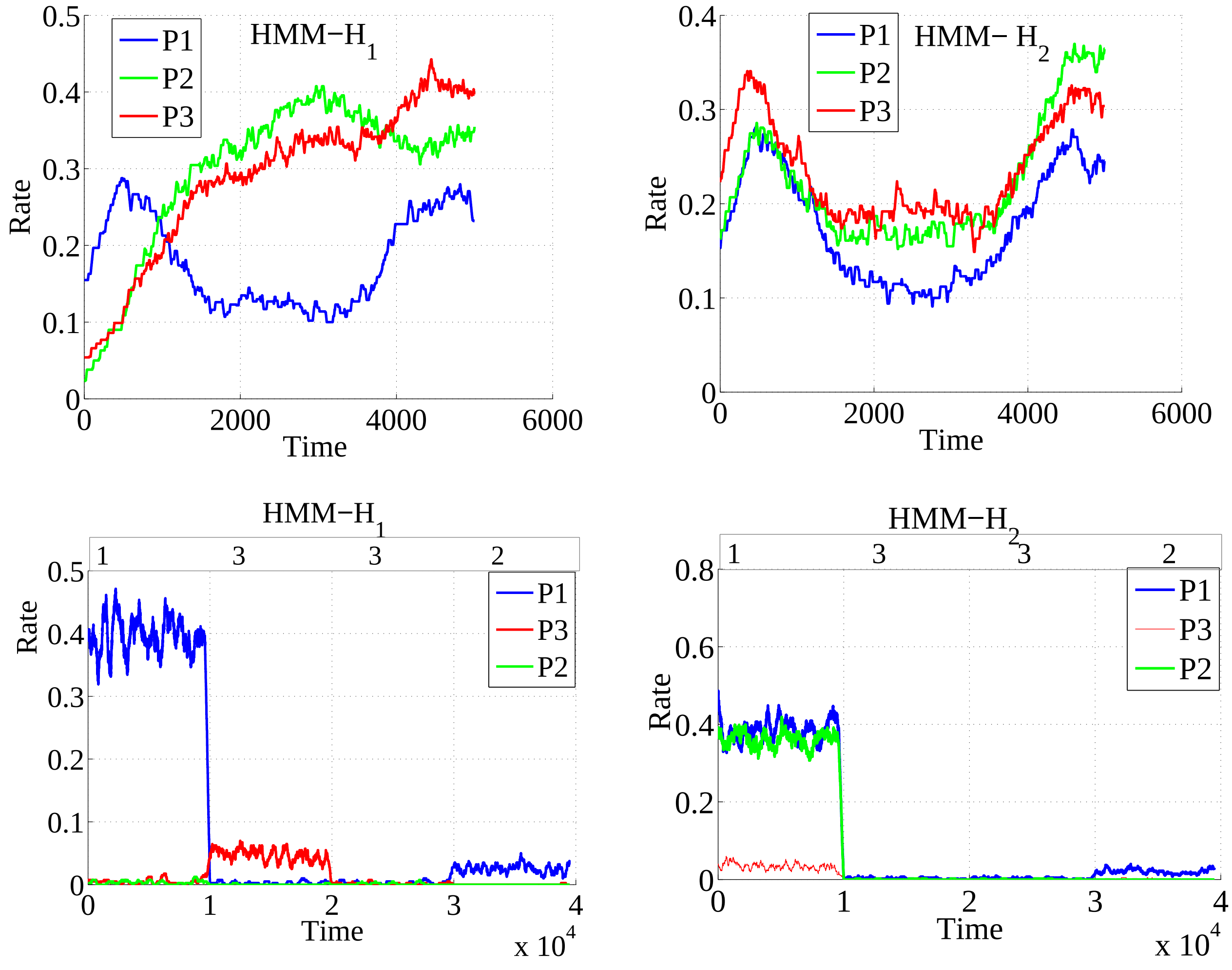


Figure 5.3: (Upper):the observation sequence {1,3,3,2} of the Instantaneous firing-rates for neurons N_1 to N_3 using the IFMP architecture. (Lower): the observation sequence {1,3,3,2} of the Instantaneous firing-rates for neurons N_1 to N_3 using stochastic computation architecture

architecture is introduced in the successive sections.

Winner take -all networks are a case of competitive learning in recurrent neural networks. Output nodes in the network mutually inhibit each other, while simultaneously activating themselves through reflexive connections. After some time, only one node in the output layer will be eventually active, namely the one corresponding to the strongest input. Thus the network uses nonlinear inhibition to pick out the largest of a set of inputs. Winner-take-all is a general computational primitive that can be implemented using different types

Table 5.1: Differences in implementation between the pulse based stochastic computation structures and the pulse based IFMP computation modules

Pulse - based IFMP Computation	Pulse - based Stochastic Computation
Number of Transistors = 100 (more dense in layout chip design)	Number of Transistors = 300 (less dense)
Analog computations, low power dissipation = 1.6 μ W, operation current =200nA	Digital Computations, Higher power dissipation in 100s mW, operation current in hundreds μ A
Implementing the computations by taking the log of input rates (probabilities) introduce wide range(Ex: log .000001= -6)	Implementing the computations prohibits the decoding process (Ex: multiplication of probabilities reduce the values causing underflow)
Increasing the number of pulses to represent the rate improves the convergence equation and eventually enhances the decoding applications.	Increasing the iterations does not improve the decoding output.
The IFMP module is scalable, meaning that the IFMP units has the same structure each implement partial computation of any mapped algorithm (Ex. HMM and SVM algorithm)	Simple design but each algorithm has different structure as hardware.

of neural network models, including both continuous-time and spiking networks (Grossberg, 1973; Oster, 2009)[87] [86].

5.2.1 Importance of WTA

1. WTA has a role in the sensing of the external stimulus in the receptive fields of cochlea, retina and skin [88]. [Zaghlol 2006, Wen, 2003, Dungem, 2005]. A winner-take-all circuit finds the maximum among a set of inputs, shutting off the other inputs. This

is a useful function in many sensory processing tasks that try to focus attention on the most interesting salient of an array of sensory inputs. Solving this task on-chip with dedicated parallel computing elements is much faster than processing the raw sensor data serially on a computer.

2. WTA has a role in cortical processing. Decision processes in the brain are not localized in one specific region but evolve in a distributed manner when different brain regions cooperate to reach a consistent interpretation. The winner-take-all circuit with both cooperative and competitive properties is a main building block that contributes to this distributed decision process in [89] [90].

3. WTA has a role in a hierarchical model of vision in cortex. Computational visual attention is the process of selecting a part of the available visual information for localization, identification and understanding of objects in an environment or visual scene. This process allows the visual system to capture visual input preferentially by shifting attention about an image, giving more attention to salient locations and less attention to unimportant regions [91] [92] [93].

4. WTA circuit conceived as part of an imager chip to process current input from a motion detection array, thus detecting the row and column of maximum change of illumination. The fact that this WTA processes analog input and produces spike output is most convenient for the address event interface (AER) that conveys the WTA output off-chip.

5.2.2 Background of WTA (continuous input/output variables)

Because of the importance of WTA networks, these networks have been of great interest to researchers. Yuille and Grzywacz (1989) [94] and Ermentrout (1992) [95] are classical references to theoretical analyses. In these early models, the neurons are non spiking, that is, they receive an analog input and have an analog output. The analog WTA computation can be efficiently implemented in very-large scale integrated (VLSI) transistor circuits. With initial circuits described in [96], a whole series of analog models [97, 98, 99] and implementations has been developed in [100, 101, 102, 103, 104, 105, 106].

5.2.3 Case study to design WTA (spiking networks)

In the past decade, spiking neuron models and their electronic counterparts have gained increasing interest. Spike-based networks capture the asynchronous and time-continuous computation inherent in biological nervous systems. Neuron models with analog inputs and analog outputs can be converted into models with spiking output if a thresholding operating is introduced to the neuron. Coultrip, Granger, and Lynch (1992) is an early theoretical analysis, with further descriptions in Jin and Seung (2002) [107] and Yu, Giese, and Poggio (2002) [108] and VLSI implementations in Chicca, Indiveri, and Douglas (2004), Abrahamsen, Hafliger, and Lande (2004) [109], and Oster, Wang, Douglas, and Liu (2008) [110]. The next theoretical considerations are models with both spiking input and spiking output. Previous theoretical studies focused on population models (e.g., Lumer, 2000) [111], where the population firing represents a graded analog value. Indiveri, Horiuchi, Niebur, and Douglas (2001) [104] and Chicca, Lichtsteiner, Delbruck, Indiveri,and Douglas (2006) [112] are VLSI implementations that use the firing rate as an analog input and output encoding.

A special case is presented in Carota, Indiveri, and Dante (2004) [113] and Bartolozzi and Indiveri (2004) [114], in which the core of the winner-take-all is analog but the signals are converted to spike rates for communication with the outside world. Table 5.2 present the summary of the above background of WTA design.

5.2.4 Contribution and Significance of IFMP as WTA

No analysis until now has considered the effect of single spikes and spike timing on the winner-take-all computation with random spiking inputs and outputs. Gautrais and Thorpe (1998) [115] start their analysis from a similar point of view, that is, how the network decides which of two input spike rates is higher, but they do not consider sampling of this estimation in the output spikes (this analysis could be classified as spiking input and analog output). The emergence of multichip spiking systems that incorporate the WTA as part of their decision process (Serrano-Gotarredona et al., 2005 [116]; Choi, Merolla, Arthur, Boahen, and Shi, 2005 [117]; Chicca, Lichtsteiner, Delbruck, Indiveri, and Douglas, 2006 [118]; Vogelstein, Mallik, Culurciello, Cauwenberghs,and Etienne-Cummings, 2007 [119]) highlights the necessity for theoretical quantification of a WTA network based on different network parameters (e.g., CAVIAR), especially if these systems are to be used in different applications.

To address this need a framework is developed by Oster et al. (2001-2009) [86] for quantifying the performance of a spiking WTA network with regular spiking inputs. Our contribution is to develop a new pulse computational architecture that implement WTA computation as one application to our model. The new aspect here is that the input pulses to the network is not necessary to be regular which is the basic of Oster architecture in which the recognition of the highest rate is based on receiving the earliest signal in time among

Table 5.2: Survey of Winner take all design started from theoritical analaysis into spiking populaion of neural networks

Research of WTA development	Achievement of WTA
Yuille and Grzywacz (1989);Ermentrout (1992)	Models as theoretical analyses: In these early models, the input and output signals of neuron models are analog rather than spiking signals.
Lazzaro, Ryckebusch, Mahowald, and Mead (1989); Kaski and Kohonen (1994); Barnden and Srinivas (1993); Hahnloser, Sarpeshkar, Mahowald, Douglas, and Seung (2000)	The WTA circuits are initially described using analog computation.
He and Sanchez-Sinencio (1993); Starzyk and Fang (1993); Serrano-Gotarredona and Linares-Barranco (1995); Kincaid, Cohen, and Fang (1996); Indiveri (1997,2001); Moller, Maris, Tomes, and Mojaev (1998); Hahnloser, Sarpeshkar, Mahowald, Douglas, and Seung (2000); Liu (2000, 2002)	The WTA are efficiently implemented in very-large scale integrated (VLSI) transistor circuits
Coultrip, Granger, and Lynch (1992); Jin and Seung (2002); Yu, Giese, and Poggio (2002);Chicca, Indiveri, and Douglas (2004); Abrahamsen, Hafliger, and Lande (2004); Oster, Wang, Douglas, and Liu (2008), Lumer (2000), Indiveri, Horiuchi, Niebur, and Douglas (2001); Chicca, Lichtsteiner, Delbruck, Indiveri, and Douglas (2006)	Early theoretical analysis with VLSI implementation of spiking neuron models when the threshold operating point is introduced to the neuron. The studies are focused on spiking population models, where the population firing represents a graded analog value.
Oster et al. (2001-2009)	quantifying the performance of a spiking WTA network with regular spiking inputs

set of incoming sources of spikes. If the input spikes are random, then Osters architecture will not recognize the highest rate as will explained in the next section. In the new designed IFMP architecture, the inputs to our computing architecture is quantified as the rate of random spikes in which a simple procedure is implemented to recognize the highest rate even if the input spikes are random pulses in time. Table 5.2 list the summary of the WTA background.

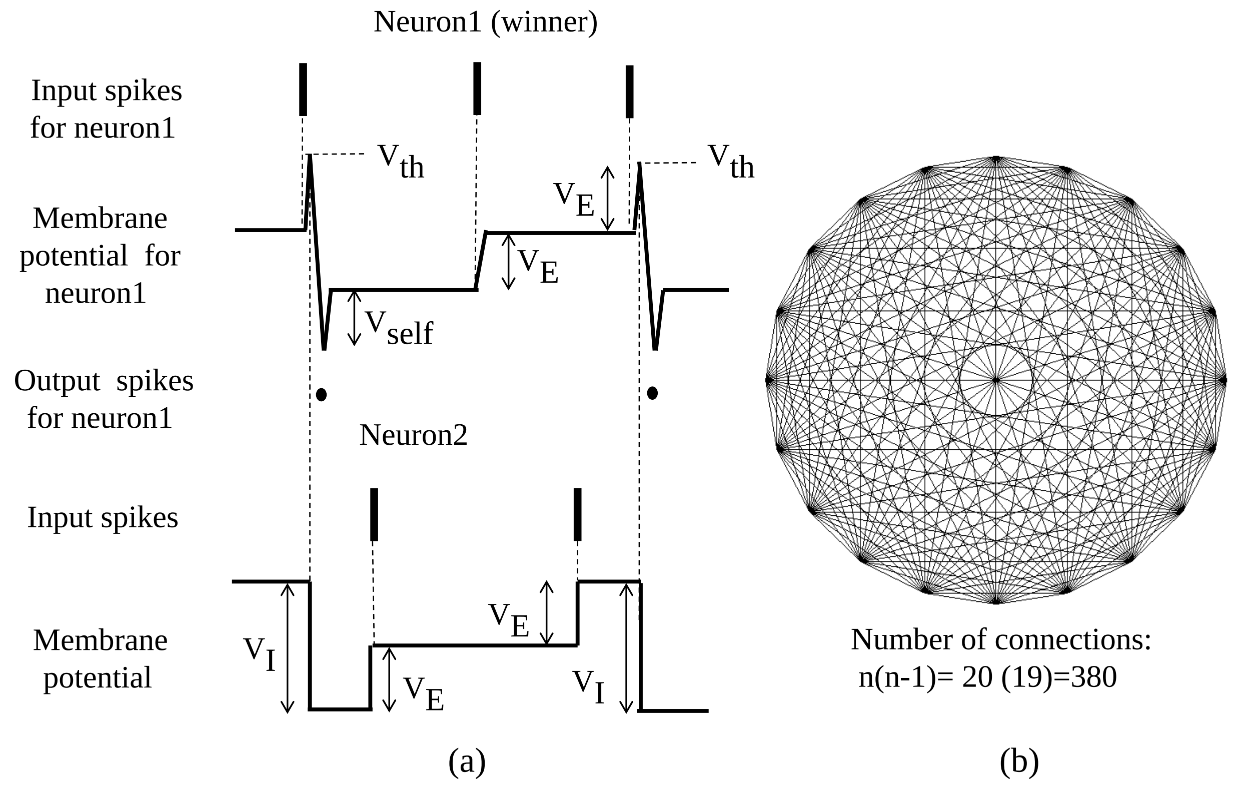


Figure 5.4: (a): Input spikes, membrane potential and output spike of neuron1 and neuron2. It shows how is the winner neuron1 inhibit the membrane of neuron2.(b): Full connectivity between 20 neuorons for processing the input pulses shown in (a)

5.2.5 Experiment

A framework is developed by Oster et al. (2001-2009) for quantifying the performance of a spiking WTA network with regular spiking inputs. Our contribution is to develop a new pulse computational architecture that implement WTA computation where the input pulses to the network is not necessary to be regular which is the basic of Oster architecture in which the recognition of the highest rate is based on receiving the earliest signal in time among set of incoming sources of spikes. The problem of the above architecture is that if the input spikes are random, then Osters architecture will not recognize the highest rate. Experiment are implemented to simulate both Oster's architecture and IFMP architecture for a network of 20 neurons. For each case the simulation of the neuronal population of neurons are considered with no connection between neurons and with specific connection according to the type of architecture. In order to understand the difference in the architecture and operation between the newly designed IFMP and Oster's design, both of the above architectures are explained next.

1. To explore the concept of Oster's architecture, Fig. 5.4(b) shows the full connectivity between neurons in Oster's design. The number of connectivity in this architecture is simply calculated as $n(n-1)$, where n is the number of neurons. The operation of this network is explained using Fig. 5.4(a) which demonstrate the membrane behavior for a couple of neurons when the input spikes are applied. The input spikes have different frequency and therefore the winner neuron is the one with the highest input frequency or the least in time period as explained in the followings:

   (a) Neuron 1 (the winning neuron) spikes when its membrane potential exceed threshold. After every spike, the neuron is reset to V_{self}.

   (b) As soon as neuron 1 spikes , no other neuron is able to spike because of the inhibitory connection from neuron 1 reset the membrane potential of other neurons (neuron 2 in the figure).

(c) When the input spikes are applied to the neurons, the membrane potential of the neurons increase by a step voltage V_E. As the membrane potential reaches the threshold, the neuron generates a spike and resets the membrane voltage again to V_{self}. Then the above procedures are repeated and therefore the shortest time period of the spike is considered as the winner neuron.

(d) If the duty cycle for the input spikes are not equal to 50% or the input spikes are irregular, then there will be more than one winner. Fig. 5.5 shows that when the input pulses are irregular and applied to three neurons (for example), then there will be more than one winner. Therefore it can be deduced that Oster's network can not recognize the highest rate for streams of random pulses.

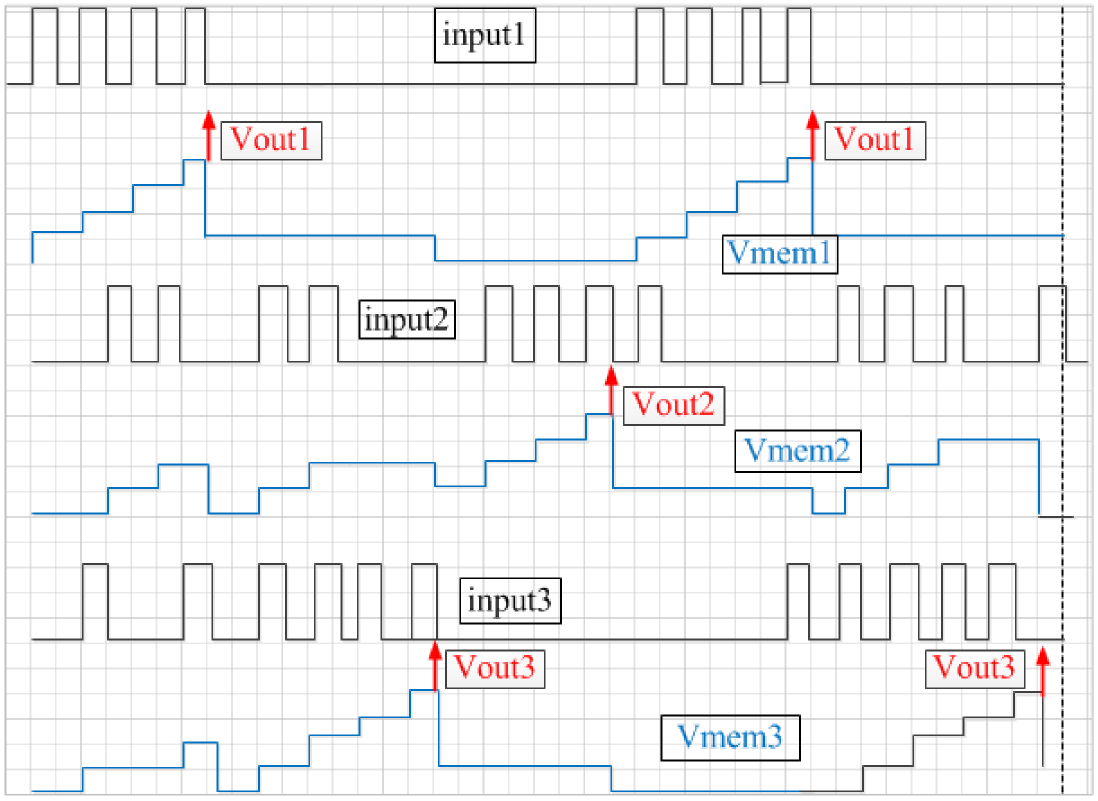


Figure 5.5: More than one winner in Oster's architecture when three random input streams are applied to three neuron.

The above procedure are simulated and implemented with two types of connectivity between the IF neurons. The first type of connectivity are implemented when there is no weight between neurons (for time less than 400 units) and the second type when there is full connectivity (for time greater than 400 units). Two experiments are implemented, the first one when the input spikes are regular and the second one when the input spikes are irregular (random). As a result of the first experiment, a raster plot in Fig. 5.6 shows that the neurons spike according to regular input spikes when there is no connectivity and there is only one neuron spike (winner) when there is full connectivity. On the other hand, the second experiment shows the raster plots for four simulations in Fig. 5.7where the neurons spikes when there is no connectivity and there in different winners with the full connectivity architecture.

2. The concept of pulse based computation of the IFMP architecture is discussed earlier in details. In Fig. 5.8(a), it is shown the network architecture between neurons (n) in the IFMP design where the number of connectivity in this architecture is calculated as $3n$. The concept of inhibitory connection between the neurons is similar in both the IFMP network and the receptive field of the sensory system. In Fig. 5.8(c), the neurons of the highest rate inhibit the neighboring neurons in order to propagates the highest input stimuli(highest rate). For three input IFMP module shown in Fig. 5.8(b), the output of N4 inhibits neurons N1, N2 and N3 according to the convergence equation $[L_1 - z]_+ + [L_2 - z]_+[L_3 - z]_+ = \gamma$. The output neuron N3 in this case propagates the highest input rate when the constraint γ is approximately zero. Fig. 5.9 shows the membrane potential for neurons N1, N2 and N3 for a three-inputs IFMP where the output rate is the highest rate between the input rates. For comparison

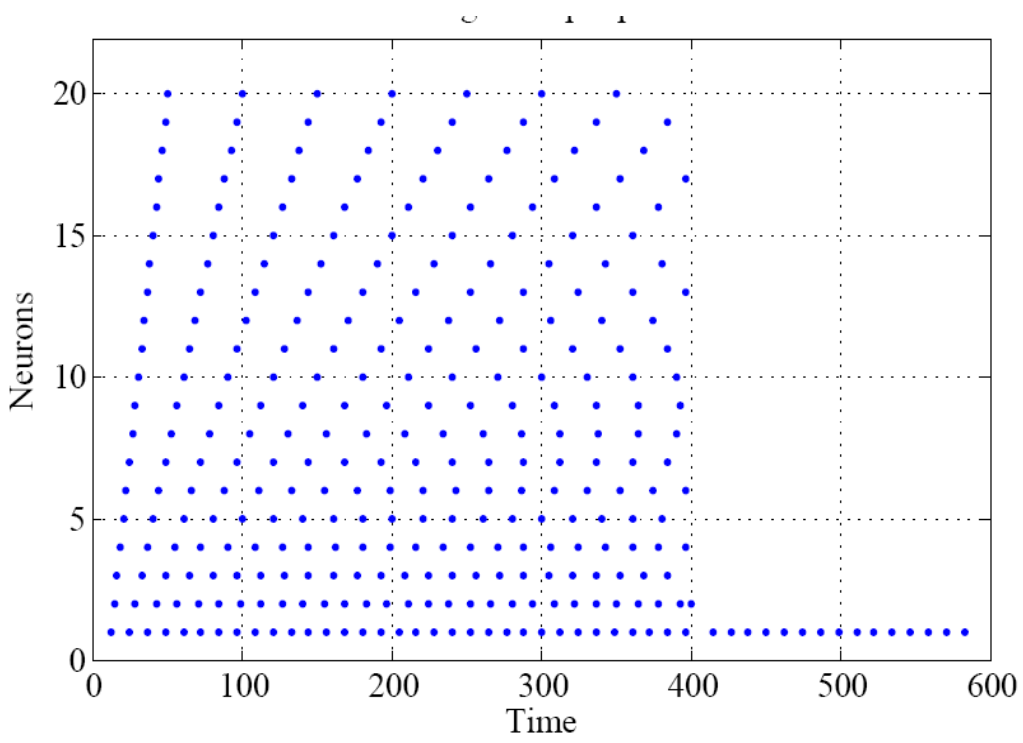


Figure 5.6: Raster plot for 20 neurons based on Oster's spiking neuranal architecture. Two types of connectivity are implemented between the IF neurons, the first part (less than 400 units) are implemented when there is no weight between neurons. The second part (greator than 400 units) in the full connectivity and implementation of Oster's archhitecture showing that there is only one winner neuron

reasons, 20 neurons of the IFMP modules are simulated to test the spiking capability of the neurons when these neurons have inhibitory connections and when there is no connectivity between these neurons. Fig. 5.10(a)shows that there is one winner between the 20 neurons for the inhibitory connections while Fig. 5.10(b) shows that the neuron spikes according to the membrane potential and the threshold reached as the normal operation for neurons.

Finally table 5.3 lists the difference comparison between the temporal spiking network in Oster's architecture [86] and the IFMP architecture.

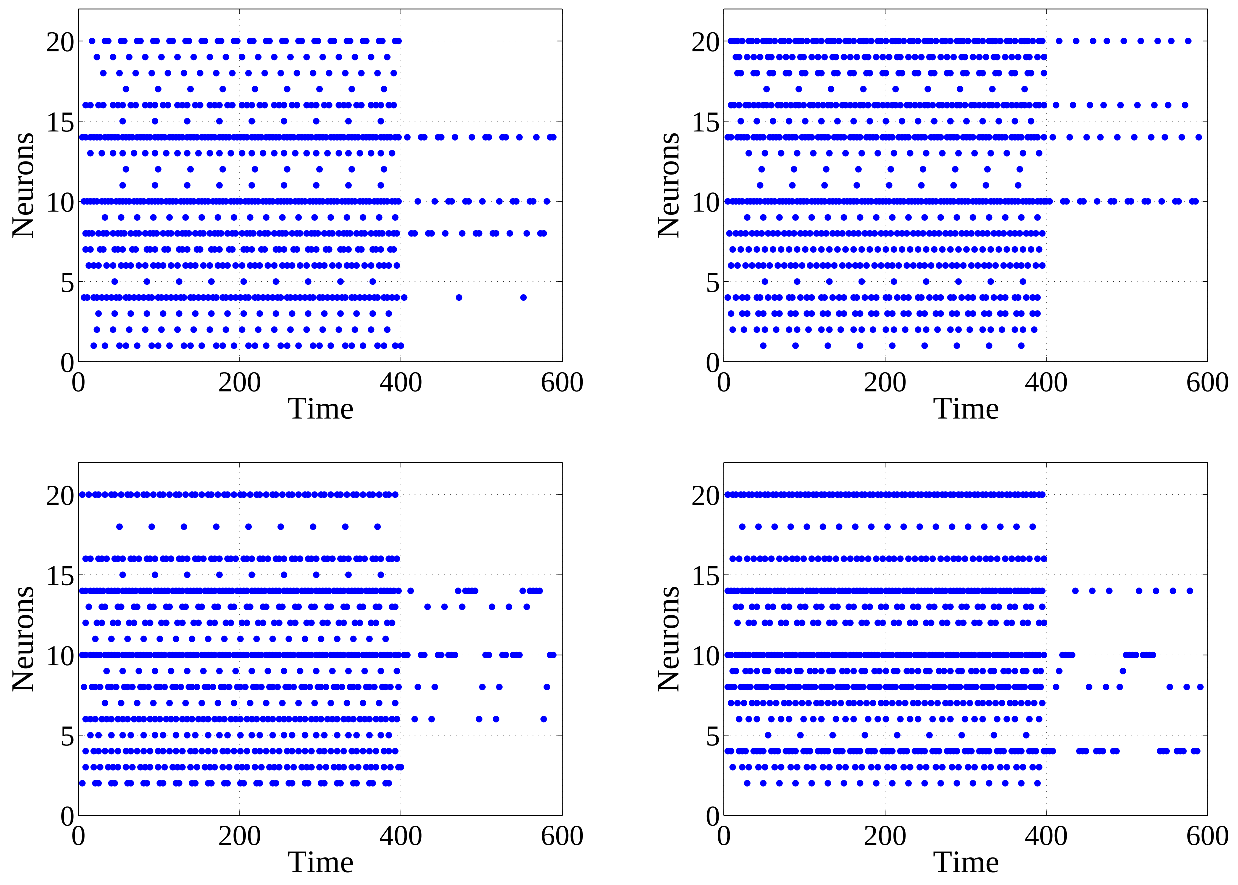


Figure 5.7: Raster plot for four experiment to the Oster's Architecture with stochastic inputs

5.3 Summary

In this chapter, two significant roles of the new designed pulse mode computation are explored. In the first significance role, it is shown that the designed pulse mode computation enhances the dynamic range of the signal processing applications that are based on probability or the rate of input/output parameters. The reason for the wide range enhancement is mapping the computational functions into log- sum- exp which eliminates the underflow problem caused by repeated multiplication of probabilities. It is shown using a simple sequence detection factor graph application that using stochastic computation failed to detect a simple sequence of numbers while the new designed pulse computational IFMP modules

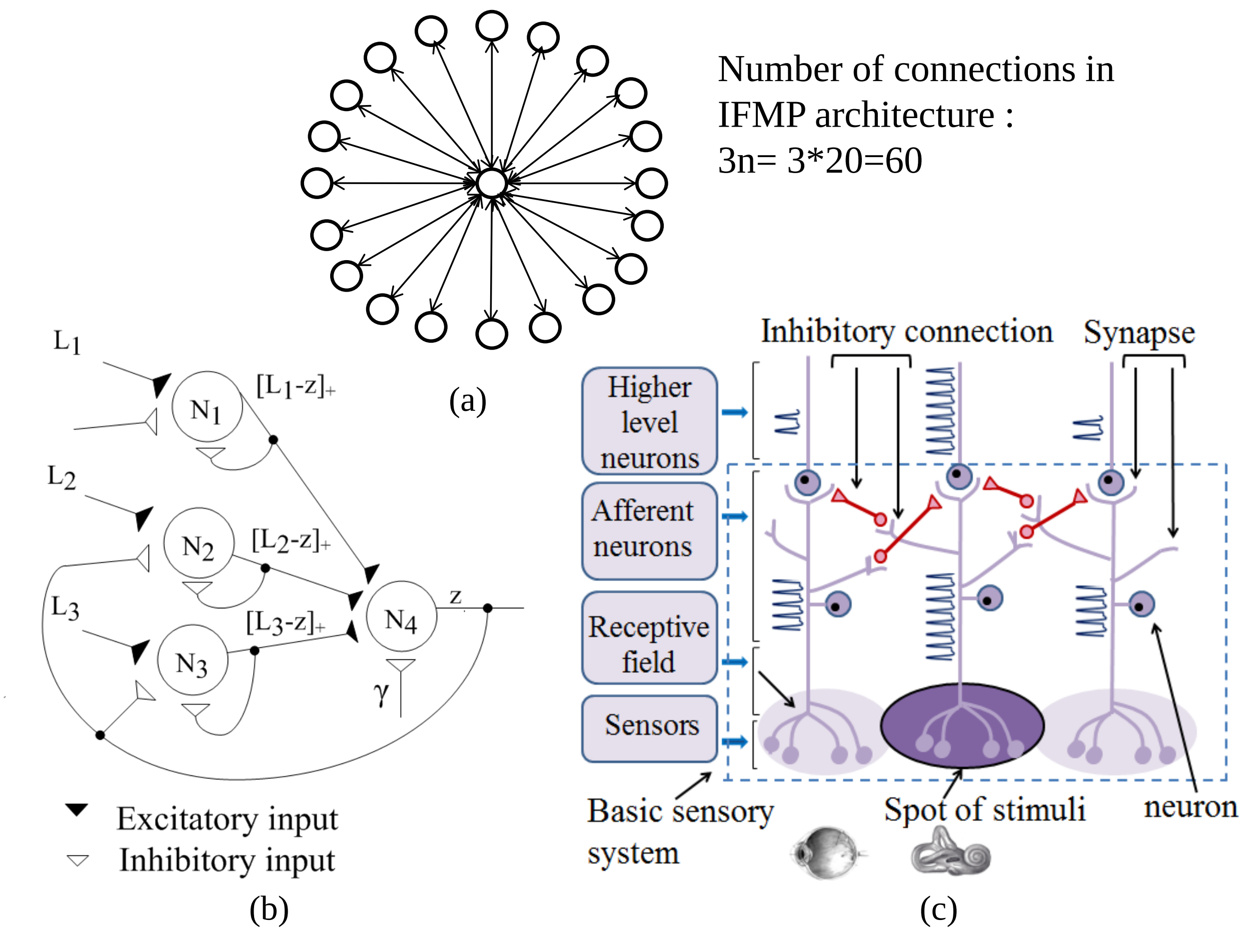


Figure 5.8: (a): Conection between 20 neurons in the IFPM architecture. (b): Inhibitary connections between neurons in the IFMP architecture similar to the basic sensory system in the receptive field shown in (c)

recognize the sequence directly.

The second significant role of the designed module is the direct and simple implementation of an important design in the bio sensory system named as winner take all. A background of WTA network has discussed in details showing that no research has yet designed the winner take all circuit as spiking network where the input/ outputs are stream of random pulses. In this section, the successful implementation of winner take all to random input pulses is compared to the unique study in this aspect that shows a limited capability in designing the spiking winner take all networks.

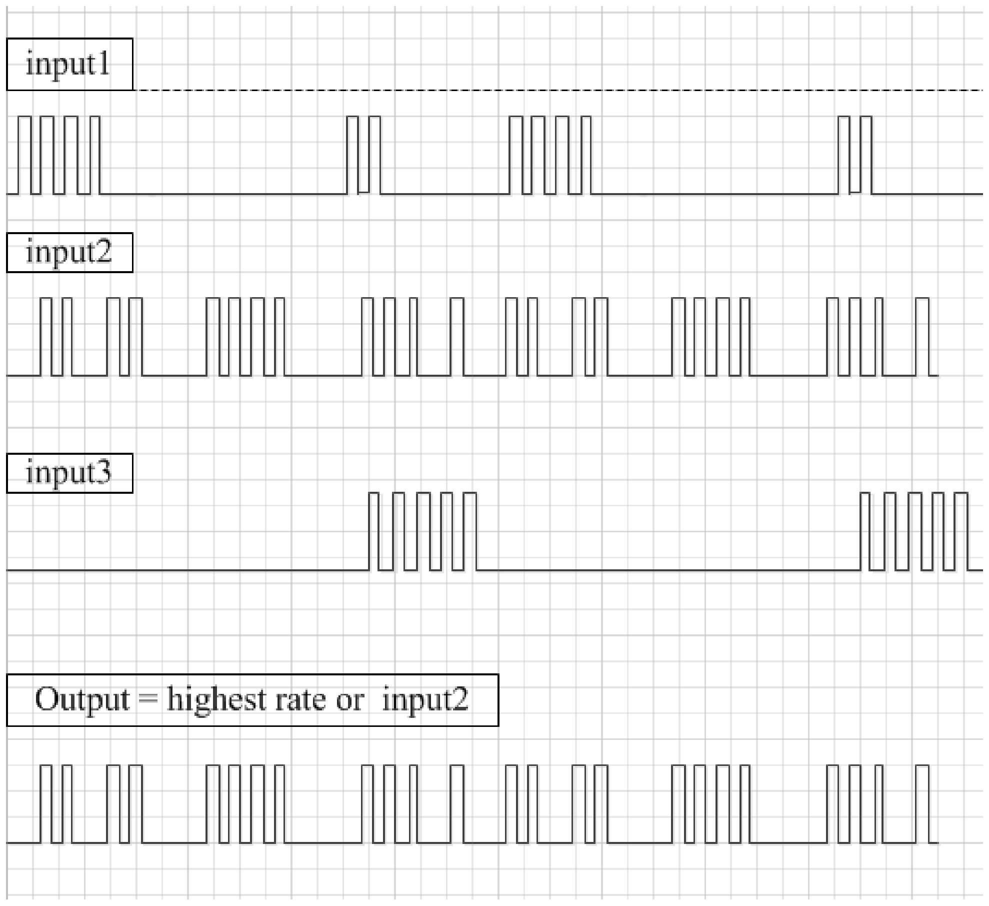


Figure 5.9: Pulse output response for three input IFMP architecure showing that the output rate is equal to the highest input rate

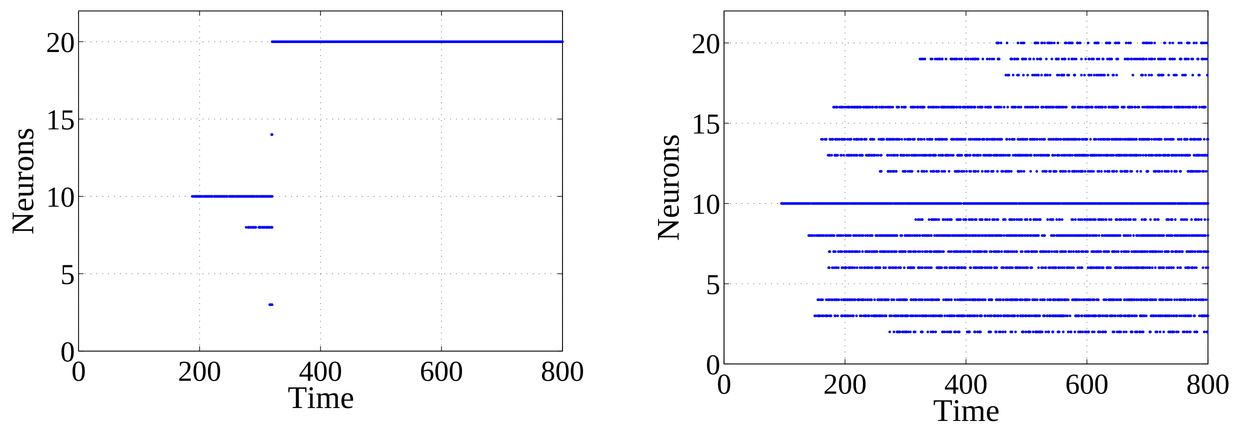


Figure 5.10: (a):Spikes of neuron with respect to time using IFMP architecture with inhibitory connections between IF neurons. (b):Spikes of neuron with respect to time using without the connection between the IF neurons

Table 5.3: Comparison between the temporal spiking network in Oster's architecture and the proposed work

WTA (IFMP)	WTA (Oster, Douglas, Shi Chii 2001-2012)
The activated input pulses are random where the probability of generating the pulses is equal to the rate of logic one over a time window.	The activated input pulses are assumed as regular rate which is not the real case in the biological neuronal population
The activating variables are currents injected by the random input / output pulses.	The activating variables are currents injected by of regular input / output pulses.
Competition between neurons is decided through a global neuron as shown in fig a. Ex: To detect the max input between N neurons, the number of connection are equal 3N connections	Competition between neurons is decided through full connections as shown in fig b. Ex: To detect the max input for 100 neurons, the number of connections is equal N (N-1) connections.
Simple procedure of detecting the max input by the excitatory synapses and the inhibitory feedback to/from a global neuron using the margin equation.	Complicated procedure of detecting the max input through self excitatory connection and inhibit all the neurons in the network.

Chapter 6

Summary and Future work

In chapter one, it is stated that real time processing and scalability are goals of implementing signal processing applications on a chip in general and implementing neuromorphic systems as specific computations. In order to achieve real time processing, the computations are processed in parallel and implemented in pulse mode. The pulse mode computations mix the advantages of analog and digital hardware designs as well as to propagate easily the computed parameters in real time between the successive modules. In order to achieve a scalable design, it is important to introduce a computation algorithm that can be cascaded to achieve partial processes of the algorithms and procedures.

As a contribution to this large effort, a novel and scalable mechanism of pulse computation algorithm and its hard ware implementation is designed to approximate the nonlinear functions as an important procedure to implement signal processing algorithms. The designed module is based on the integrate and fire structure and margin propagation algorithm (IFMP). In chapter one, the objectives and the tasks to implement the designed pulse computational module are addressed. In the chapters two, three, four and five, the analysis of these tasks are explored in details. In this chapter, the objective goals to design the scalable IFMP algorithmic module, the tasks and methodology to meet the objective and the significance roles of the designed algorithm are summarized with recommendations to the future work.

6.1 Objectives

In order to design a scalable pulse computation algorithm and its hardware implementation, it is important to meet the following objectives,

1. Convergence property: Since the computations are implemented with stochastic pulse computation, the designed algorithm has to meet the convergence property for each unit of the designed module.

2. Scalability property: Scalability means that the basic designed module can be cascaded to handle a growing amount of work in a capable manner.

3. Computation properties: As a new computation algorithm, the module has to meet some computational properties such as: scaling property, monotonicity, convexity, superposition and offset property.

4. Pulse computation verification: The pulse mode computational module has to verify the the approximation of the basic and non linear functions such as the addition, multiplication, power, polynomial and inner product.

5. Pulse computational applications: It is important to implement procedural algorithms to verify that the pulse computation module has the capability to implement signal processing applications such as decoding and classification.

6. Hardware realization: The basic concept of the designed module is the capability of mapping the algorithm into analog circuits.

7. Power and size efficiency: In order to design a dense array of the pulse mode-algorithm on a chip, it is recommended to map the algorithm into a compact and low power

analog circuits.

8. Significance role: The designed pulse module has the roles in dynamic range enhancement and unique design to maximize the highest input rate in spiking networks.

6.2 Implemented tasks

To meet the above objectives, the following tasks and methodologies are applied for the verification of the scalable pulse computational module:

Design the core unit of the IFMP: The basic unit of the IFMP module is designed using the concept of the integrate and fire structure. The goal here is to verify the convergence of input/output pulses such that $\mathcal{E}(L[n]) = \mathcal{E}(d[n])$, where $\mathcal{E}(L[n])$ and $\mathcal{E}(d[n])$ are the expected values of input pulses L and d over time period T as shown in following equations: $(\frac{1}{T}\sum_{n=1}^{T} L[n])$ and $(\frac{1}{T}\sum_{n=1}^{T} d[n])$ respectively.

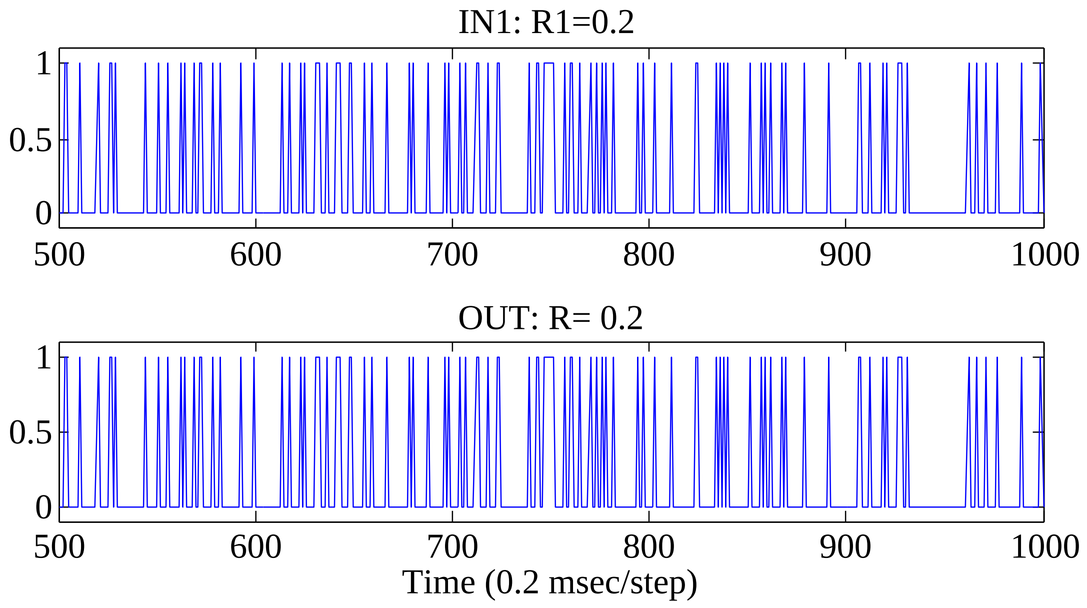


Figure 6.1: Convergence of one IF neuron with input rate equal 0.2. It shows that the output rate (R) is equal to the input pulse rate (R1)

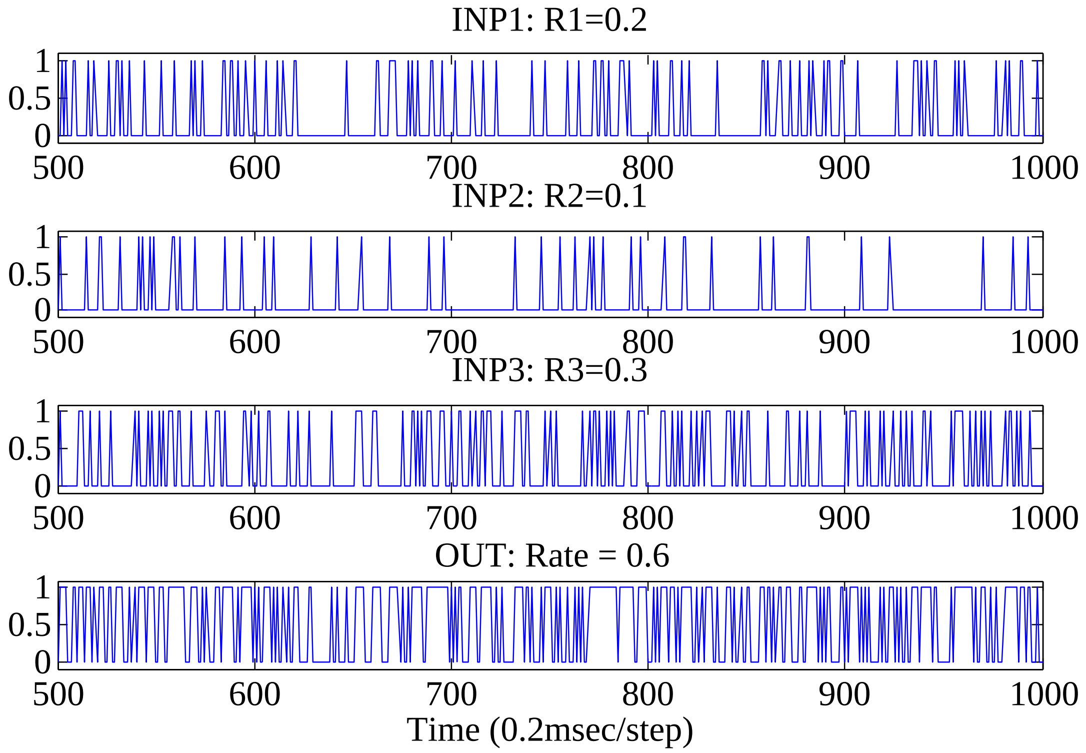


Figure 6.2: Convergence of one IF neuron with three inputs of rates 0.2, 0.1, and 0.3 respectively. It shows that the output rate (Rate) is convergence to the sum of the input pulse rates (R1, R2, and R3)

The above convergence is proven theoretically and verified experimentally. If the "IF" neuron has one input, then the output rate (OUT) converges to the input rate (IN1) as shown in Fig. 6.1, where both of the input and output rates are equal to 0.2. In addition to that, the convergence is verified for more than one input according to this equation: $\sum_{i=1}^{P} \mathcal{E}(L[n]) = \mathcal{E}(d[n])$, where (P) is the number of input signals to the neuron. In Fig. 6.2, an experiment is implemented using three inputs $(P = 3)$. It is shown that the sum of the expected values for these inputs is equal to the expected value of the output pulses.

IF application: As an application to the IF structure, a novel circuit is designed to adjust the weight of the synapse by changing the input current to the neuron. The injected current is changed according to the previous and post pulses to the neuron. The synapses weight is adjusted using storage transistors named as the floating gates. The above synapse weight adjustment is designed with ten synapses connected a IF neuron. The result of the experiment verifies the behavior of the synapse time dependency plasticity that represents the mechanism of learning process in neuronal population.

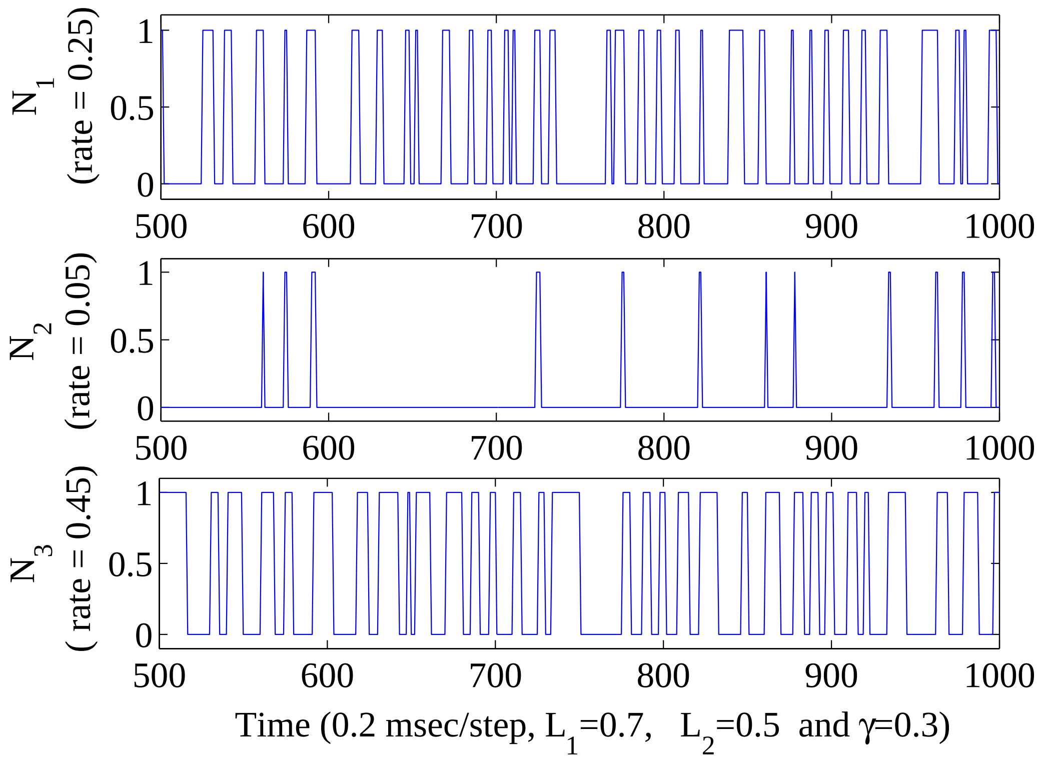


Figure 6.3: Convergence of the output rates for the three neurons $N_1 = 0.25$, $N_2 = 0.05$ and $N_3 = 0.45$ in the IFMP structure when the input rates γ, L_1, and L_2 are equal to $0.3, 0.7$ and 0.5 respectively

IFMP pulse computational unit: A special network of the IF neurons is designed to approximate non- linear functions in pulse mode. The designed algorithmic module is based on margin propagation that approximate the log-sum-exp function. The importance of the log-sum -exp is to map easily the nonlinear functions. Moreover, the importance of margin propagation is to map these function into analog circuit. In order to verify the algorithmic

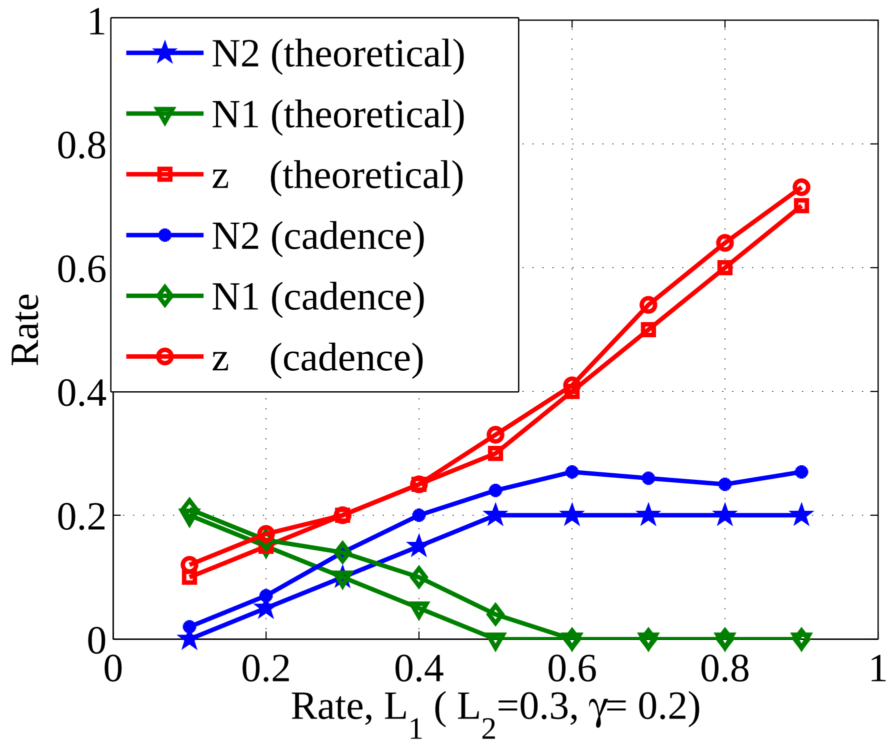


Figure 6.4: Dynamic characteristics for two-input IFMP module showing the convervence and the output rates for the three neurons $N1$, $N2$ and $N3$ in both theoretical and circuit(cadence) simulation

operation of IFMP, the following convergence equation must be satisfied,

$$\sum_{i=1}^{P} [\mathcal{E}(L_i[n]) - \mathcal{E}(L_z[n])]_+ \longrightarrow \mathcal{E}(\gamma[n])$$

where L_i, p, z, γ and n are the i'th input, number of inputs, output, constraint and time step respectively. The inputs, output and γ are quantified as the rate of ones in a stream of random pulses. The concept of the convergence in the above equation is proven theoretically, verified in software simulation and circuit simulation as explained in chapter 3. An experiment is implemented when the input rates γ, L_1, L_2 are equal to $0.3, 0.7$ and 0.5 respectively. The output pulses for the neurons converge to $N_1 = 0.25$, $N_2 = 0.05$ and $N_3 = 0.45$ as shown in Fig. 6.3.

Table 6.1: Pulse rates obtained in the theoretical "theo" and cadence "cad" simulations for the three neurons N_1, N_2 and N_3, note that z is the output rate of neuron N_3. Where $\gamma = 0.2$ and $L_2 = 0.3$

L_1	$N1_{cad}$	$N2_{cad}$	z_{cad}	$N1_{theo}$	$N2_{theo}$	z_{theo}
0.1	0.21	0.02	0.12	0.2	0	0.1
0.2	0.16	0.07	0.17	0.15	0.05	0.15
0.3	0.14	0.14	0.2	0.1	0.1	0.2
0.4	0.1	0.2	0.25	0.05	0.15	0.25
0.5	0.4	0.24	0.33	0	0.2	0.3
0.6	0	0.27	0.41	0	0.2	0.4
0.7	0	0.26	0.54	0	0.2	0.5
0.8	0	0.25	0.64	0	0.2	0.6
0.9	0	0.27	0.73	0	0.2	0.7

Another experiment is implemented to show the convergence of the IFMP algorithmic module in software and circuit simulations. Fig. 6.4 and table 6.1 shows the dynamic convergence of the output rates for the three neurons $N_1 = 0.25$, $N_2 = 0.05$ and $N_3 = 0.45$ in the IFMP structure when the input rates γ, L_1, and L_2 are equal to $0.3, 0.7$ and 0.5 respectively.

The figure shows a difference between theoretical and practical results of the rates as indicated in the legend and the corresponding table of numerical values. The reason of the difference in the circuit simulation is the requirement of the accurate balance for the IFMP convergence to fix the input currents to the inhibitory and excitatory paths of the IF. In addition to that, the accuracy of the output rates depends on the number of iterations for the input pulses. The implementation of IFMP in analog circuit is important because it tells that the IFMP concept can be mapped on a chip and thus achieve dense, high speed, and low power specification compared to the software simulation.

Cascading and approximation concept of IFMP: The importance of the cascading or the scalability property is to handle gnawing amount of designs and to implement different algorithms on the same hard ware array with different connectivity and settings. The connectivity and settings in this project are implemented using field-programmable gate array (FPGA).

In order to achieve the cascading property for the IFMP modules, first of all the approximation between the margin propagation and the log-sum-exp must be estimated in order to achieve the right computations. In chapter 3, it is proven that the estimation between the above function is equal to the rate γ. It is shown in the simulation too the successfully approximation for both the parallel and serial cascading.

Properties, computation and applications: As a pulse computation algorithmic model, it is important to identify and verify the properties of these computations. In chapter 4, the scaling, offset, monstrosity, convexity and supper position were discussed in details. The above properties were proven and simulated to show that the above properties are applicable for the designed computational algorithm in pulse mode. Scaling and offset properties are used frequently in the computations because the inputs, output and γ are probabilistic terms or pulse rate terms. In order to map the IFMP to the non-linear functions, all the parameters of the equation are in logarithmic values. Applying the logarithmic function to these terms will get negative numbers that require biasing adjustments by adding an offset and then scaling the rates to values within the probability range (0:1). Having the above properties for IFMP modules, the module can then be used to approximate the mathematical functions such as multiplication, addition, power, inner product and polynomial function. The above functions are verified in both the software simulation and the circuit simulation as explained in chapter 4. In addition to that, more complicated algorithm are verified using cascaded IFMP modules such as recursive dynamic equations and support vector machine algorithms for decoding and classification applications respectively.

6.3 significance of the IFMP algorithmic module

1. **Dynamic range improvement**

   To identify the difference that the pulse computational IFMP module would have to make in computations compared to other pulse computational modules, an application of sequence detection is applied in two pulse computation techniques: the stochastic pulse computations and the IFMP algorithmic module.

   The Stochastic computation is a new alternative approach for iterative computation on factor graphs. In this approach, the information is represented by the statistics of the bit stream which results in simple high-speed hardware implementation of graph-based algorithms. Stochastic arithmetic was introduced to design low-precision digital circuits. The important motivation for considering stochastic computation was the possibility of performing complex computations using only simple circuitry. In stochastic computation as in the IFMP pulse computations , probabilities are represented as streams of random digital bits. Using this representation, complex operations on probabilities such as multiplication and division are converted to operations on bits which can easily be manipulated using simple stochastic gates.

   Although the stochastic computations seems feasible but it fails to implement a simple detection application. In chapter 5, a simulation to the factor graph application ”sequence detection” is implemented using stochastic computation. The simulation shows that the low range of the processing parameters cause a failure to recognize an input sequence to the hidden Markov model with a matching observation. In the same chapter, it is shown the significance difference in detecting a simple sequence in both the pulse stochastic computation and the IFMP computations which tells that the logarithmic representation of the computational parameter enhance the range of the small values of the probabilities in computations.

2. **Novel design of spiking winner take all network**

   In chapter 5, the background of designing winner take all (WTA) in continuous time systems and spiking networks is discussed. Many attempts are listed in the background survey to design the WTA because of it's importance in the sensory system of the receptive fields. Many researchers are interested in the design of spiking networks but most of them looked to the WTA in a spiking architecture as a circuit that implement continuous signal with encoder/decoder for spikes to continuous/ continuous to spikes signal conversion. The only designed network for spiking WTA that addressed the spiking WTA is the Oster's architecture, where the input and output signals for the designed algorithm are quantified as spike rate. Yet, these input spikes have to be regular in order to identify the winner between the input streams of pulses. If the above architecture is designed for random input pulses, the design will fail to identify the winner.

   The significance and novelty of the pulse computational IFMP unit is the identification to the winner input even if the input streams are random pulses (variable time period between pulses/ stream). In chapter 5, it is shown that the designed IFMP module verifies the concept of winner take all (WTA) algorithm in pulsed computations and that the idea of WTA is embedded in the IFMP algorithm. The IFMP algorithm approximates the max input if the inhibiting parameter γ is chosen as a small rate (example: 0.001), thus it can be said that IFMP propagates the maximum input rate. The novelty in the designed winner take all is the implementation of the concept when the input pulses are **stochastic or random** compared to the unique designed found in the survey which implement spiking WTA based on the **regular** input pulses.

3. **Chip test verification:**

   The next significance role of the designed pulse computational module is the successful implementation and realization of the algorithmic concept into analog circuit and layout design. The design pulse computation algorithm is realized on a chip as an array of 64 units of two-input IFMP modules. Couple of the 3-8 decoders are used to select the IFMP unit in the array and 16- pull up transistors are used to propagate the output pulses to identify the x-y coordination of the IFMP that generates a spike in the array. If the IFMP units are verified as a stable unit in silicon layout as in the simulation, then the IFMP array is ready to approximate the basic functions and the algorithmic applications.

   In order to built an architecture for any application, a state machine is designed using hardware description language (HDL) that is configured on an interfacing field-programmable gate array (FPGA). The HDL program is designed to generate random pulses that are applied to the IFMP units in time multiplexing mode. The number of the required IFMP units for implementing applications depends on the complexity of the algorithmic application. For each step time unit, the output of the IFMP units is detected and routed using a lookup table that represents the connectivity of IFMP network. The output pulses are stored in the FPGA registers for the analysis and verification of the applied algorithmic application. Following the above procedure, the IFMP array has been tested and verified in the following,

(a) Testing and verification of the convergence property of the IFMP algorithmic module on a chip as discussed in chapter 3.

(b) Testing and verification of a sequence decoder application using HMM algorithm as discussed in chapter 4

The same chip can be used for other application in the same procedure discussed above. This is the scalability issue, which means that there is no need to redesign the hardware for different applications. For example, the scalability is essential in designing the neuromporphic architectures that require dense neuron modules to implement signal processing applications using FPGA or any other interfacing programmable controller.

6.4 Future work

Since the newly designed module is significant in spiking networks and it is scalable in detection and classification algorithms, then it is recommended the followings,

1- It is recommended to built speech recognition and image processing systems on the hardware level. A pre-stage is required for both applications to encode/decode the images and speeches into/from sequence of pulses.

2- In the neuromorphic applications, the designed arrays can be used as computational model in mimicking the biological sensory systems especially that the induced noise is considered as beneficiary to enhance performance.

3- In order to get accurate convergence of each neuron and each IFMP unit in the array in the hardware level, the biasing voltages can be fixed using floating gate transistors. This is a huge project in fixing the biasing using row and column shift registers to identify each biasing voltage.

6.5 Publications

chapter book

1. Thamira Hindo and Shantanu Chakrabartty, Noise-Exploitation and Adaptation in Neuromorphic Sensors (Biomimetics, and Bioreplication. Edited by Lakhtakia, Akhlesh, ELSEVIER 2012).

2. Thamira Hindo, " Basic Computations using a Novel Scalable Pulse-Mode Modules" an invited paper to IAENG Transaction on Engineering Technologies -Special Issue of the World Congress on Engineering and Computer Science 2012"

Conferences

1. Thamira Hindo and Shantanu Chakrabartty, Noise-Exploitation and Adaptation in Neuromorphic Sensors, Bioinspiration, Biomimetics, and Bioreplication. Edited by Lakhtakia, Akhlesh. Proceedings of the SPIE, Volume 8339, pp. 833905-833905-13,2012.

2. Thamira Hindo, An Asynchronous, Time- Domain Analog Hidden Markov Model Circuit Based On Integrate and Fire Margin Propagation, 5th International Conference on Computer and Electrical Engineering ICCEE, Hong kong, Oct, 2012.

3. Thamira Hindo, Scalable Pulsed Computational Module Using Integrate and Fire

Structure and Margin Propagation Algorithm, World Congress on Engineering and Computer Science, San Francisco, Oct, 2012

4. Thamira Hindo and Timothy Grotjohn, "Design of Scalable Pulse Computational Module Using Integrate and Fire Structure", Engineering Graduate Research Symposium, Michigan State University, East Lansing, Nov 2012.

5. Thamira Hindo and Timothy Grotjohn, "Design of pulse - based winner take all network using integrate and fire neuron and margin propagation algorithm", Fifth Annual Graduate Academic Conference, Michigan State University, Feb, 2013.

Journals

1. Thamira Hindo, Classification Using Novel Spiking Computation Module, Submitted to Electrical and Electronics Engineering: An International Journal (EEEIJ) ,ISSN:2200-5846, accepted on 6'th November 2012.

2. Thamira Hindo, Properties and Applications of Scalable Pulse Computational Module Using Integration and Threshold Operation, submitted to Neurocomputing Journal on 9/2/2012

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] C. A. Mead. *AnalogVLSI and Neural Systems.* Addison-Wesley, 1989.

[2] M. Mahowald. *VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function.* Technical Report. California Institute of Technology, Pasadena, 1992.

[3] K. Boahen. Neuromorphic microchips. *Scientific American SCIAM*, 2005.

[4] S-C Liu and T. Delbruck. Neuromorphic sensory systems. *Elsevier:Current Opinion in Neurobiology*, 20:1–8, 2010.

[5] T. Hindo and S. Chakrabartty. Noise-exploitation and adaptation in neuromorphic sensors. In *Proceeding of SPIE, Bioinspiration, Biomimetics, and Bioreplication*, volume 8339, March 2012.

[6] R. Sarpeshkar. Analog versus digital: Extrapolating from electronics to neurobiology. *Neural Computation*, 10(7):1601–1638, 1998.

[7] Y.W. Li, K.L. Shepard, and Y.P. Tsividis. Continuous-time digital signal processors. In *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, pages 138 – 143, march 2005.

[8] K. Boahen. "googling the brain on a chip, stanford university, youtube", 2009.

[9] S. Liu. Introdution to Neuromorphic and Bio-inspired Mixed signal VLSI. KLAB.Caltech.edu/ Shih, 2006.

[10] Shih-Chii Liu, Tobias Delbruck, Jorgene Kramer, Giacomo Indiveri, and Rodney Douglas. *Analog VLSI: Circuits and Principles.* MIT Press, Cambridge, MA, USA, 2002.

[11] W. Gerstner and W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, 1 edition, 2002.

[12] A.A. Lazar and E.A. Pnevmatikakis. Video time encoding machines. *Neural Networks, IEEE Transactions on*, 22(3):461 –473, march 2011.

[13] P.A. Cariani. Temporal codes and computations for sensory representation and scene analysis. *Neural Networks, IEEE Transactions on*, 15(5):1100 –1111, sept. 2004.

[14] John V. Arthur and Kwabena Boahen. Learning in silicon: Timing is everything. In *NIPS*, 2005.

[15] H. Paugam-Moisy and S. M. Bohte. Computing with spiking neuron networks. (to appear), 2011.

[16] C. A. Mead and M.A. Mahowald. A silicon model of early visual processing. *Neural Networks*, pages 91–97, 1988.

[17] S.S. Tehrani, S. Mannor, and W.J. Gross. Survey of stochastic computation on factor graphs. In *Multiple-Valued Logic, 2007. ISMVL 2007. 37th International Symposium on*, page 54, May 2007.

[18] S. Sharifi Tehrani, W.J. Gross, and S. Mannor. Stochastic decoding of ldpc codes. *Communications Letters, IEEE*, 10(10):716 –718, oct. 2006.

[19] S. Sharifi Tehrani. Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams. 2011.

[20] B.D. Brown and H.C. Card. Stochastic neural computation. i. computational elements. *Computers, IEEE Transactions on*, 50(9):891 –905, Sep 2001.

[21] Weikang Qian, Xin Li, Marc D. Riedel, Kia Bazargan, and David J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Transactions on Computers*, 60:93–105, 2011.

[22] J. Miller and W. Yang. Simple pulse asynchronous state machines. In *Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on*, volume 3, pages 405 –409 vol.3, may 1996.

[23] A.F. Murray and A.V.W. Smith. Asynchronous VLSI neural networks using pulse-stream arithmetic. *Solid-State Circuits, IEEE Journal of*, 23(3):688 –697, Jun 1988.

[24] J.L. Meador, A. Wu, C. Cole, N. Nintunze, and P. Chintrakulchai. Programmable impulse neural circuits. *Neural Networks, IEEE Transactions on*, 2(1):101 –109, Jan 1991.

[25] A. Hamilton, A.F. Murray, D.J. Baxter, S. Churcher, H.M. Reekie, and L. Tarassenko. Integrated pulse stream neural networks: results, issues, and pointers. *Neural Networks, IEEE Transactions on*, 3(3):385 –393, may 1992.

[26] L. M. Reyneri. Theoretical and implementation aspects of pulse streams: an overview. In *Proceedings of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*, MICRONEURO '99, pages 78–, Washington, DC, USA, 1999. IEEE Computer Society.

[27] N. Shaikh-Husin and Chang Wooi Po. Pulse coded neural network implementation in VLSI. In *TENCON 2000. Proceedings*, volume 3, pages 237 –241 vol.3, 2000.

[28] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 1952.

[29] D. Tal, E. L. Schwartz, and M. Greis. Computing with the leaky integrate and fire neuron: Logarithmic computation and multiplication. *Neural Computation*, 19(2):305–18, 1997.

[30] G. Bugmann. Biologically plausible neural computation. *Biosystems*, 40:11–19, 1997.

[31] Christof Koch and Tomaso Poggio. Multiplying with Synapses and Neurons. In Thomas Mckenna, Joel Davis, and Steven F. Zornetzer, editors, *Single Neuron Computation*, Neural Nets: Foundations to Applications, pages 315–345. Academic Press, 1992.

[32] D. Hammerstrom. A highly parallel digital architecture for neural network emulation. In J.G.Delgado-Frias and W.R.Moore eds., editors, *VLSI for Artificial Intelligence and Neural Networks*, pages 357–366. Plenum Press, 1991.

[33] J. Cabestany, P. Ienne, J.M. Moreno, and J. Madrenas. Is there a future for ann hardware?, 1996.

[34] A. Murry and L. Tarassenko. *Analogue Neural VLSI A pulse stream approach.* Chapmann and Hall, 1994.

[35] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[36] A.F. Murray and A.V.W. Smith. Asynchronous VLSI neural networks using pulse-stream arithmetic. *Solid-State Circuits, IEEE Journal of*, 23(3):688 –697, Jun 1988.

[37] P. Hylander, J. Meader, and E. Frie. VLSI implementation of pulse coded winner take all networks. In *Circuits and Systems, 1993., Proceedings of the 36th Midwest Symposium on*, pages 758 –761 vol.1, Aug 1993.

[38] R. P. N. Rao. neural models of bayesian belief propagation. *The Bayesian Brain: Probability Approaches to Neural Coding, K. Doya, S. Ishii, A. Pouget, R. P. N. Rao, eds., Cambridge, MA: MIT Press*, 2006.

[39] H. A. Loeliger. An introduction to factor graphs. *Signal Processing Magazine, IEEE*, 21(1):28–41, January 2004.

[40] Ming Gu and S. Chakrabartty. Synthesis of bias-scalable cmos analog computational circuits using margin propagation. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 59(2):243 –254, Feb. 2012.

[41] C. Kong and S. Chakrabartty. Analog iterative ldpc decoder based on margin propagation. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 54(12):1140 –1144, Dec. 2007.

[42] E.M. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569 – 1572, Nov. 2003.

[43] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, 1 edition, August 2002.

[44] Bruce Segee. Methods in neuronal modeling: From ions to networks, 2nd edition. *Computing in Science and Engineering*, 1:81, 1999.

[45] G. Indiveri and etal. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5:1–23, 2011.

[46] E. M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability And Bursting.* MIT Press, 2007.

[47] A.A. Lazar and L.T. Toth. Time encoding and perfect recovery of bandlimited signals. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, volume 6, pages VI – 709–12 vol.6, April 2003.

[48] M. Rastogi, A.S. Alvarado, J.G. Harris, and J.C. Principe. Integrate and fire circuit as an adc replacement. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 2421 –2424, May 2011.

[49] H.Y. Yang and R. Sarpeshkar. A bio-inspired ultra-energy-efficient analog-to-digital converter for biomedical applications. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 53(11):2349 –2356, Nov. 2006.

[50] F. Ratliff. *Mach bands: quantitative studies on neural networks in the retina.* MIT Press, 1965.

[51] A L Hodgkin and A F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[52] Shih-Chii Liu. Analog VLSI circuits for short-term dynamic synapses. *EURASIP Journal on Applied Signal Processing*, 7:620–628, Jan. 2003.

[53] J.M. Cruz-Albrecht, M.W. Yung, and N. Srinivasa. Energy-efficient neuron, synapse and stdp integrated circuits. *Biomedical Circuits and Systems, IEEE Transactions on*, 6(3):246 –256, June 2012.

[54] Paul E. Hasler, Bradley A. Minch, and Chris Diorio. Floating-gate devices: they are not just for digital memories any more. In *ISCAS (2)*, pages 388–391. IEEE, 1999.

[55] V. Srinivasan, G. Serrano, C.M. Twigg, and P. Hasler. A floating-gate-based programmable cmos reference. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 55(11):3448 –3456, Dec. 2008.

[56] T. Morie, O. Fujita, and K. Uchimura. Self-learning neural network lsi with high-resolution non-volatile analog memory. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1628 –1632 vol.4, Nov/Dec 1995.

[57] F. Tenore, R.J. Vogelstein, R. Etienne-Cummings, G. Cauwenberghs, and P. Hasler. A floating-gate programmable array of silicon neurons for central pattern generating networks. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4 pp. –3160, May 2006.

[58] M Lenzlinger. Fowler-nordheim tunneling into thermally grown sio2. *Journal of Applied Physics*, 40(1):278, 1969.

[59] D. Kahng and S. M. Sze. A floating-gate and its applications to memory devices. *The Bell System Technical Journal*, XLVI(6):1288–1295, July 1967.

[60] Giacomo Indiveri. Neuromorphic bistable VLSI synapses with spike-timing-dependent plasticity. In *in Advances in Neural Information Processing Systems*, volume 2002, pages 1091–1098, 2002.

[61] John V. Arthur and Kwabena Boahen. Learning in silicon: Timing is everything. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, 2005.

[62] A. Bofill and A. Murray. Synchrony detection by analogue vlsi neurons with bimodal stdp synapses. In *IN ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2003.

[63] C. Diorio, P. Hasler, B.A. Minch, and C.A. Mead. A floating-gate mos learning array with locally computed weight updates. *Electron Devices, IEEE Transactions on*, 44(12):2281 –2289, Dec 1997.

[64] P. Hafliger and C. Rasche. Floating gate analog memory for parameter and variable storage in a learning silicon neuron. In *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, volume 2, pages 416 –419 vol.2, jul 1999.

[65] Shih-Chii Liu and R. Mockel. Temporally learning floating-gate VLSI synapses. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 2154 –2157, May 2008.

[66] D. Robert and M. C. Göpfert. Novel schemes for hearing and orientation in insects. *Current Opinion in Neurobiology*, 12(6):715–720, December 2002.

[67] T. Hindo. Scalable pulsed computational module using integrate and fire structure and margin propogation algorithm. In *World Congress on Engineering and Computer Science 2012, San Francisco*, pages 10–16, 2012.

[68] J. Tapson and A. Schaik. An asynchronous parallel neuromorphic adc architecture. IEEE, ISCAS, 2012.

[69] Ming Gu, K. Misra, H. Radha, and S. Chakrabartty. Sparse decoding of low density parity check codes using margin propagation. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1 –6, 30 2009-Dec. 4 2009.

[70] M.D. Godfrey. CMOS device modeling for subthreshold circuits. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 39(8):532 –539, Aug 1992.

[71] C. C. Enz. *High Precision CMOS Micropower Amplifiers.* PhD thesis, 1989. No. 802.

[72] E. Vittoz. MOS Transistor -Intensive Summer course on CMOS VLSI Design - Swiss Federal Institute of Technology, 2007.

[73] R. Sarpeshkar, T. Delbruck, and C. Mead. White noise in MOS transistors and resistors. *IEEE Circuits and Devices*, 9(6):23–29, June 1993.

[74] D. J. Mar, C. C. Chow, W. Gerstner, R. W. Adams, and J. J. Collins. Noise shaping in populations of coupled model neurons. *Proceedings of the National Academy of Sciences*, 96(18):10450–10455, August 1999.

[75] M. J. Chacron, B. Lindner, L. Maler, A. Longtin, and J. Bastian. Experimental and theoretical demonstration of noise shaping by interspike interval correlations. In *luctuations and Noise in Biological, Biophysical, and Biomedical Systems III*, page 150. SPIE 5841, 2005.

[76] R. K Shaga. Noise- Shaping Optimization and Online Learning with Applications to Digitally Assisted Analog Circuits. A Thesis Submitted to Michigan State University, 2012.

[77] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 –286, Feb 1989.

[78] J. Lazzaro, J. Wawrzynek, and R.P. Lippmann. A micropower analog circuit implementation of hidden markov model state decoding. *Solid-State Circuits, IEEE Journal of*, 32(8):1200 –1209, Aug 1997.

[79] T. Hindo. An asynchronous, time- domain analog hidden markov model circuit based on integrate and fire margin propagation. In *5th international Conference on computer and electrical engineering ICCEE, Hong kong*, pages 10–16, 2012.

[80] Vladimir N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[81] John C. Platt. Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[82] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 200–209, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[83] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM.

[84] S. Chakrabartty and G. Cauwenberghs. Gini-support vector machine:quadratic entropy based multi-class probability regression. *Journal of Machine Learning Research*, 8:813–839, 2007.

[85] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead. Winner-take-all networks of o(n) complexity. In *Advances in Neural Information Processing Systems*, pages 703–711. Morgan Kaufmann, 1989.

[86] Matthias Oster, Rodney Douglas, and Shih-Chii Liu. Computation with spikes in a winner-take-all network. *Neural Comput.*, 21(9):2437–2465, September 2009.

[87] S. Grossberg. Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, 52(3):213–257, 1973.

[88] G. Indiveri, C. Rasche, and R. Douglas. Neuromorphic Sensory Processing using Analog Very Large-Scale Integration. In Et, editor, *Frontiers of Life*. Academic Press, San Diego, CA, 2002.

[89] Shin-Ichi Amari and Michael A. Arbib. *Competition and Cooperation in Neural Nets.* Springer, Berlin, 1982 1982.

[90] Rodney J. Douglas and Kevan A. Martin. Neuronal circuits of the neocortex. *Annual review of neuroscience*, 27(1):419–451, 2004.

[91] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex, 1999.

[92] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[93] L. Carota, G. Indiveri, and V. Dante. A software-hardware selective attention system. *Neurocomputing*, 58–60:647–653, Jun 2004.

[94] A. L. Yuille and N. M. Grzywacz. A winner-take-all mechanism based on presynaptic inhibition feedback. *Neural Computation*, 1(3):334–347, 1989.

[95] Bard Ermentrout. Complex dynamics in winner-take-all neural nets with slow inhibition. *Neural Networks*, 5(3):415–431, 1992.

[96] John Lazzaro, Sylvie Ryckebusch, Misha Mahowald, and Carver Mead. Winner-take-all networks of o(n) complexity. In David S. Touretzky, editor, *NIPS*, pages 703–711. Morgan Kaufmann, 1988.

[97] Samuel Kaski and Teuvo Kohonen. Winner-take-all networks for physiological models of competitive learning. *Neural Networks*, 1994.

[98] J. A. Barnden and K. Srinivas. Temporal winner-take-all networks: a time-based mechanism for fast selection in neural networks. *Trans. Neur. Netw.*, 4(5):844–853, September 1993.

[99] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. J. Douglas, and S. Seung. Digital selection and analog amplification co-exist in an electronic circuit inspired by neocortex. *Nature*, 405(6789):947–951, 2000.

[100] Y. He and E. Sanchez-Sinencio. Min-net winner-take-all cmos implementation. *Electronics Letters*, 29(14):1237 –1239, July 1993.

[101] Starzyk and X. Fang. CMOS current mode winner-take-all circuit with both excitatory and inhibitory feedback. *Electronic Letters*, 29(10):908–910, May 1993.

[102] R. Serrano-gotarredona, M. Oster, P. Lichtsteiner, A. Linares-barranco, R. Paz, F. Gmez-rodrguez, H. Kolle Riis, T. Delbrck, S. C. Liu, S. Zahnd, A. M. Whatley, R. Douglas, P. Hfliger, G. Jimenez-moreno, and A. Civit. Aer building blocks for multi-layer multi-chip neuromorphic vision systems. In *in Advances in Neural Information Processing Systems*, pages 1217–1224. MIT Press, 2005.

[103] Yuguang Fang, Michael A. Cohen, and Thomas G. Kincaid. Dynamics of a winner-take-all neural network. *Neural Networks*, 9(7):1141–1154, 1996.

[104] G. Indiveri, T. Horiuchi, E. Niebur, and R.J. Douglas. A competitive network of spiking VLSI neurons. In F. Rattay, editor, *World Congress on Neuroinformatics*, ARGESIM Report no. 20, pages 443–455, Vienna, 2001. ARGESIM / ASIM - Verlag.

[105] R. Mller, M. Maris, Oller M. Maris, J. Tomes, and A. Mojaev. A strong winner-take-all neural network in analogue hardware, 1998.

[106] S.C. Liu, S.C. Liu, and Institute of Neuroinformatics (Zürich). *A Winner-take-all Circuit with Controllable Soft Max Property*. INI's posters. Institute of Neuroinformatics, ETH/UNIZ, 2001.

[107] Dezhe Z. Jin and H. Sebastian Seung. Fast computation with spikes in a recurrent neural network. *Physical Review E*, 65(5):051922+, May 2002.

[108] Angela J. Yu, Martin A. Giese, and Tomaso A. Poggio. Biophysiologically plausible implementations of the maximum operation. *Neural Comput.*, 14(12):2857–2881, December 2002.

[109] Jens Petter Abrahamsen, Philipp Hfliger, and Tor Sverre Lande. A time domain winner-take-all network of integrate-and-fire neurons. In *ISCAS (5)*, pages 361–364, 2004.

[110] Matthias Oster, Yingxue Wang, Rodney Douglas, and Shih-Chii Liu. Quantification of a spike-based winner-take-all VLSI network, 2008.

[111] Erik D. Lumer. Effects of spike timing on winner-take-all competition in model cortical circuits. *Neural Comput.*, 12(1):181–194, January 2000.

[112] E. Chicca, P. Lichtsteiner, T. Delbruck, G. Indiveri, and R.J. Douglas. Modeling orientation selectivity using a neuromorphic multi-chip system. In *International Symposium on Circuits and Systems, ISCAS 2006*, pages 1235–1238. IEEE, 2006.

[113] L. Carota, G. Indiveri, and V. Dante. A software-hardware selective attention system, 2004.

[114] C. Bartolozzi and G. Indiveri. A neuromorphic selective attention architecture with dynamic synapses and integrate-and-fire neurons. In *Brain Inspired Cognitive Systems, BICS 2004*, volume BIS2.2, pages 1–6, Aug 2004.

[115] J. Gautrais and S. Thorpe. Rate coding versus temporal order coding: a theoretical approach. *Biosystems*, 48(1-3):57–65, 1998.

[116] Teresa Serrano-Gotarredona and Bernab Linares-Barranco. A modular current-mode high-precision winner-take-all circuit. In *ISCAS*, pages 557–560, 1994.

[117] T. Y. W. Choi, P. A. Merolla, J. V. Arthur, K. A. Boahen, and B. E. Shi. Neuromorphic implementation of orientation hypercolumns. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(6):1049–1060, June 2005.

[118] E. Chicca, P. Lichtsteiner, T. Delbruck, G. Indiveri, and R.J. Douglas. Modeling orientation selectivity using a neuromorphic multi-chip system. In *International Symposium on Circuits and Systems, ISCAS 2006*, pages 1235–1238. IEEE, 2006.

[119] R. Jacob Vogelstein, Udayan Mallik, Eugenio Culurciello, Gert Cauwenberghs, and Ralph Etienne-Cummings. A multichip neuromorphic system for spike-based visual information processing, 2007.