Politecnico di Torino

III Facoltà di Ingegneria

Universitat politecnica de Catalunya

Izhikevich neural model and STDP learning algorithm mapping on spiking neural network hardware emulator

Thesis project carried out online at the UPC
Master degree in Electronic Engineering

Advisor: Jordi Madrenas
Advisor: Guido Masera

Author: Antonio Caruso

November 29, 2020

Contents

Abbreviations

AER Address Event Representation

ALU Arithmetic and Logic Unit

BRAM Block Random Access Memory

EPSP Excitatory Post-Synaptic Potential

HEENS Hardware Emulator of Evolved Neural System

IF Integrate and Fire

IPh Initialization Phase

IPSP Inhibitory Post-Synaptic Potential

ISA Instruction Set Architecture

LFSR Linear-Feedback Shift Register

LIF Leaky Integrate and Fire

LIFO Last In First Out

LSB Less Significant Bit

LSW Less significant word

ND Neural processing Device

MC Master Chip

MSB Most Significant Bit

MSW Most significant word

PE Processing Element

PSP Post-Synaptic Potential

SIMD Single Instruction Multiple Data

SNN Spiking Neural Network

VHDL Very High speed integrated circuits Hardware Description Language

Abstract

The proposal of this thesis is to embed the Izhikevich neuron model and a full custom "Spike timing dependent plasticity" (STDP) learning algorithm in an architecture called HEENS (Hardware Emulator of Evolved Neural System). HEENS is a multi-chip structure developed at the "Universitat Politècnica de Catalunya" (UPC) based on a ring link topology connecting several SIMD processors reproducing each one a group of neuron of a Spiking neural network (SNN). The Izhikevich neuron model is a worldwide adopted mathematical model for reproducing the neural membrane potential evolution, observed in some mammalian cortex, along time and according to external stimuli. STDP is a biological learning algorithm which shapes the strength of a synaptic connection according to the timing with which that connection takes part to the overall spiking activity of the post and pre-synaptic neurons.
This master thesis project, in particular, acts at algorithm level and at instruction level as well at architectural level. It takes place analysing the mathematical models for the right data parallelism, writing the assembly program describing the neural routine, modifying the instruction set and the existing hardware of the HEENS architecture, in order to fulfil the biological model computational needs. The comparison between the actual behaviour of HEENS to that of the mathematical models is performed via MatLAB scripts. In the following:
Chapter 1 reviews the basic concepts of the neural networks and briefly details the HEENS architecture for what is of close interest to this text.
Chapter 2 deals with the Izhikevich neural model, which is first analysed in order to detect the minimum data parallelism needed, then a rounding scheme is introduced at hardware level and a custom LFSR-made gaussian noise generator is designed at hardware/firmware level to accomplish with the biological background noise, finally the developed assembly program is described and a neural network is simulated at RTL level and validated in MatLAB.
Chapter 3 first details the STDP rule adopted, then the integration of the model in HEENS and the ISA enhancement are presented, the developed assembly program is therefore explained and few test networks are set up in order to better show and demonstrate the algorithm characteristics and outcomes. Finally, in **Chapter 4** conclusions and future work are presented.
The performed simulations showed how the results in HEENS, obtained with a fine tuned fixed point arithmetic, well approximate the outcomes of floating point MatLAB model in every application. This project allows HEENS to embody the Izhikevich neural model with good biological consistency and flexibility. This is assured by the possibility to randomize or fix neural parameters, which allow to represent a mixed-neuron-type neural network or a precise kind of neurons.
Anty-simmetric Hebbian STDP has been integrated and tested with different granularity: from connection level, to neuron level, to the support of the entire network, which guarantees the best performance. The test networks, mapped in HEENS and simulated at RTL level in **Chapter 3**, allowed to validate the algorithm and clearly visualize the trend of the main STDP parameters.

CHAPTER 1

Introduction

From the 20th century, biological mechanisms of the brain behaviour have become more and more interesting for the research communities in information fields due to the computational power of the systems they inspire. In fact, despite the lack of consensus about the information processing actually involved in brain, biological processes have served as reference for recent computational models. The first Artificial Neural Networks (ANNs) were developed as simplified versions of biological neural networks in terms of structure and function. Today, the third generation of artificial network is that of the Spiking Neural Networks (SNNs), which reach a more realistic modelling by utilizing true biological features, like spikes, to transmit information between neurons.
The proposal of this thesis is to embed the Izhikevich neuron model and a full custom "Spike timing dependent plasticity" (STDP) learning algorithm in an architecture called HEENS (Hardware Emulator of Evolved Neural System). HEENS is a multi-chip structure developed at the "Universitat Politècnica de Catalunya" (UPC) based on a ring link topology connecting several SIMD processors reproducing each one a group of neuron of a Spiking neural network.
This chapter has two different purposes, addressed in its two sections:

- first section aims to introduce the basic concepts of the neural networks. In particular, an insight of the features of the Spiking neural networks (SNNs) and a review of the principal neuron models in literature.
- In the second section, the HEENS architecture is briefly detailed for what is of close interest.

1.1 Review of Neural Networks

The idea of being inspired by the way in which the brain performs neural computation for building computer-based algorithms has been present for more than eighty years [14]. These bio-inspired algorithms of neural computation are referred to as ANNs; they contain a set of computational units (neurons) interconnected via directed edges (synapses between neurons) which process information according to a specified set of rules and equations (the model of information processing in neural circuits). Different choices for their connectivity or dynamics have given rise to a vast range of different types of models which have been thoroughly studied in computer science over the last decades, and which have enjoyed a revived interest in recent years due to the success of Deep Learning [7]. With the aim of a brief Historical classification, the three neural networks discussed in the following are represented in Fig.1.1.

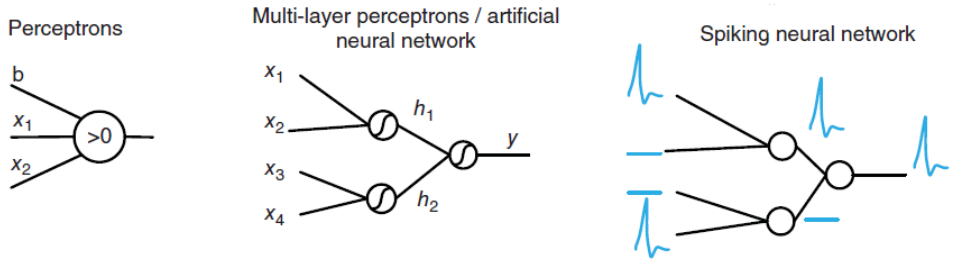

Figure 1.1: a) Perceptron model b) Multi-perceprons model or ANN c) Spiking neural network. [20]

The simplest neural network can be identified in the perceptron model, where hand-crafted features are employed as input to the network [20]. Outputs of the perceptron are binary numbers obtained through hard thresholding. The second type of neural network is sometimes called a multilayer perceptron (MLP). In an MLP, a non-linear activation function (or transfer function) is associated with each neuron. Popular choices for the non-linear activation function are the sigmoid function, the hyperbolic tangent function, and the rectifier function. The output of each neuron is a continuous variable instead of a binary state[20]. The MLP is widely adopted by the machine learning community, as it can be easily implemented on general-purpose processors. This type of neural network is so popular that the phrase "artificial neural network" (ANN) is often used to specify it exclusively. Advances in consolidating the vast number of new findings and insights from neuroscience into such computational models in a biologically plausible way have been largely lacking in the ANN community. While it has been known for long that neurons communicate with spikes (electrical pulse emitted by a neuron typically after that a potential function overcomes a threshold), it was only in the early 1990s when studies found evidence for biological brains making use of the exact timing of single spikes to encode information [1]. This observation gave rise to SNNs which, compared to the previous two types of neural networks, resembles more to a biological neural network in the sense that single spikes are used to transport information, instead of the average of them as in ANNs. It is well-known that SNNs are more powerful and advanced than ANNs, as the dynamics of an SNN is much more complicated and the information carried by an SNN could be much richer. Today, however, several critical issues and open questions in the SNNs field regarding, for example, the way in which the spikes are encoded and decoded for information transport are still present.

1.1.1 The Neural model

As discussed in the previous section, in a SNN, a neuron is a computational unit of the network with a certain activation function which determines the way in which the neuron evolves along time, receives and manages information under the form of a spike and spikes in turn. A stylized biological structure of a neuron is presented in Fig.1.2 together with a mathematical modelled version.
In a first approximation, neuron can be thought composed by:

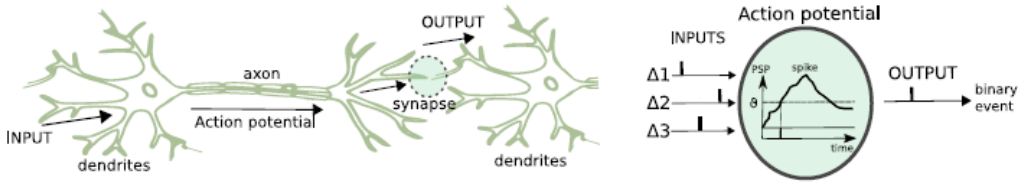

Figure 1.2: a)Stylized biological structure of a neuron b) Neuron modelled as a computational unit which receive inputs, manage them and evolve according to an activation function, produce a spiking output when a certain quantity exceeds a threshold. [6]

.

- A central nucleus or "Soma".
- Several dendrites which interact with outside as input/output nodes.
- The "Axon", physical link between cells where electrical impulsive quantity travel.
- The "synapses", terminations of the dendrite link which actually transmit biologically the spike between neurons.

The Neural mathematical model is instead better explained in Fig.1.3. At each input (synapse) can be assigned some weights and delays (W, d) which can evolve according to some rules, changing therefore the strength of the link. The inputs received will be added up, some neural parameters representing the biological state of the neuron (usually its membrane potential) therefore will evolve according to an activation function and eventually a spike is produced when the membrane potential overcame a threshold, this spike will in turn propagate toward the connected neurons.
In the next sections, some possible models for the neural activation function and a possible rule for synapse weight evolution will be presented.

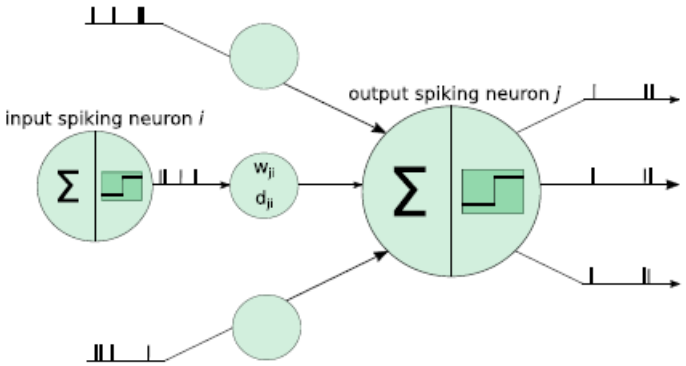

Figure 1.3: Neural model as weighted sum of incoming spikes, [13]

1.1.2 Spiking neural models

As said, the general model for a spiking neuron can be the one in Fig.1.3. What is still to be fixed is the activation function of the neuron itself, i.e. the way in which the neuron evolves with time and stimuli. The key parameter for characterizing the behaviour of the neuron is its membrane potential, which biologically represents the electric potential of the membrane which above a threshold induces the spike. Together with the membrane potential, other potential, parameters or non-synaptic input (modelled as noise) can characterize the cell. Some very popular models are now summarized.

1.1.3 Leaky Integrate and Fire (LIF)

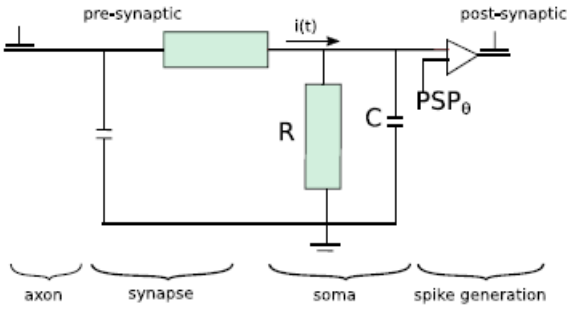

Figure 1.4: LIF circuit model [13]

In Leaky Integrate-and-Fire (LIF) model, a neuron is considered as an electrical circuit. Model consists of capacitor C in parallel with resistor R, driven by a current I(t):

$$I(t) = I_R + I_{cap} \tag{1.1}$$

The standard form of the model is defined as:

$$\tau_m du/dt = -u(t) + RI(t) \tag{1.2}$$

where $\tau_m = RC$ is the membrane time constant.
Spikes events are characterized by a firing time and after, the potential is reset to a resting potential u_r. Also, a refractory period can be included, during which the neuron is insensible to external stimuli. LIF model is simple and computationally effective, and it is the most widely used spiking neuron model despite other more biologically realistic models.

1.1.4 Hodgkin–Huxley

In Hodgkin–Huxley [8] model, a semipermeable cell membrane separates the interior of the cell from the extracellular liquid, acting as a capacitor (C). When an input current I(t) is injected into the cell, it may add further charge on C, or leak through the channels in the cell membrane. Because of active ion transport through the cell membrane, the ion concentration inside the cell is different from that in the extracellular liquid. The Nernst potential generated by the difference in ion concentration is represented by a battery. Without cover detail of minor interest for this text, can be stressed how this model, even if of good precision, uses one equation for the current and three differential equations for other parameters becoming very time and resource consuming.

1.1.5 Izhikevich

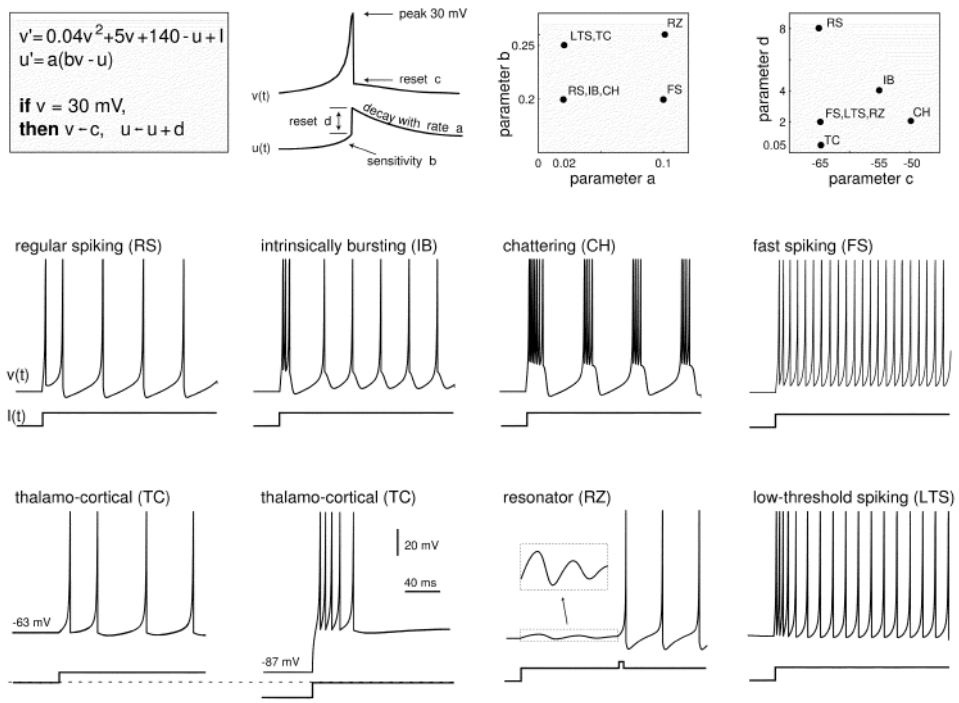

Figure 1.5: Known types of neurons correspond to different values of the parameters a, b, c, d. RS, IB, and CH are cortical excitatory neurons. FS and LTS are cortical inhibitory interneurons. Each inset shows a voltage response of the model neuron to a step of dc current I = 10 (bottom).[9]

As biological plausible as the Hodgkin–Huxley model, Izhikevich model [9] presents the computational efficiency of LIF models. It is defined by the following set of differential equations:

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{1.3}$$

$$u' = a(b * v - u) \tag{1.4}$$

with the auxiliary after-spike resetting:

$$if\ v \geq 30mV,\ than\ v = c\ and\ u = u + d \tag{1.5}$$

Here, v and u are dimensionless variables, and a, b, c, and d are dimensionless parameters called "neural parameters", v' is the derivative with respect to time. The variable "v" represents the membrane potential of the neuron and "u" represents a membrane recovery variable. The parameter "a" influences proportionally the time scale of the recovery variable, "b" describes the sensitivity of the recovery variable to the sub threshold fluctuations of the membrane potential, "c" describes the after-spike reset value for membrane potential (typical value is $-65mV$), "d" describes after-spike reset of the recovery variable (typical value is 2). "I" term represents the external stimuli, sum of synaptic currents or injected DC currents.

Depending basically on four parameters, the model can reproduce spiking and bursting (repetitive and discrete groups of spikes) behaviour of known types of cortical neurons, as illustrate in Fig.1.5. Neocortical neurons in the mammalian brain can be, indeed, classified into several types according to the pattern of spiking and bursting seen in intracellular recordings [9]. Each type can be modelled

according to a specific combination of the neural parameters, they are:
• RS (regular spiking) neurons are the most typical neurons in the cortex. When presented with a prolonged stimulus (injected step of dc-current) the neurons fire a few spikes with short inter-spikes period and then the period increases. In the model, this corresponds to c = 65 mV (deep voltage reset) and d = 8 (large after-spike jump of u).
• IB (intrinsically bursting) neurons fire a stereotypical burst of spikes followed by repetitive single spikes. In the model, this corresponds to c = 55 mV (high voltage reset) and d = 4 (large after-spike jump of u). During the initial burst, variable u builds up and eventually switches the dynamics from bursting to spiking.
• CH (chattering) neurons can fire stereotypical bursts of closely spaced spikes. The inter-burst frequency can be as high as 40 Hz. In the model, this corresponds to c = 50 mV (very high voltage reset) and d = 2 (moderate after-spike jump of u).
All inhibitory cortical cells are instead divided into the following two classes:
• FS (fast spiking) neurons can fire periodic trains of action potentials with extremely high frequency practically without any adaptation (slowing down). In the model, this corresponds to a between 0 and 1 (fast recovery).
• LTS (low-threshold spiking) neurons can also fire high-frequency trains of action potentials, but with a noticeable spike frequency adaptation. These neurons have low firing thresholds, which is accounted for by b between 0 and 25 in the model. To achieve a better quantitative fit with real LTS neurons, other parameters of the model need to be changed as well.

1.1.6 Spike Timing Dependent Plasticity (STDP)

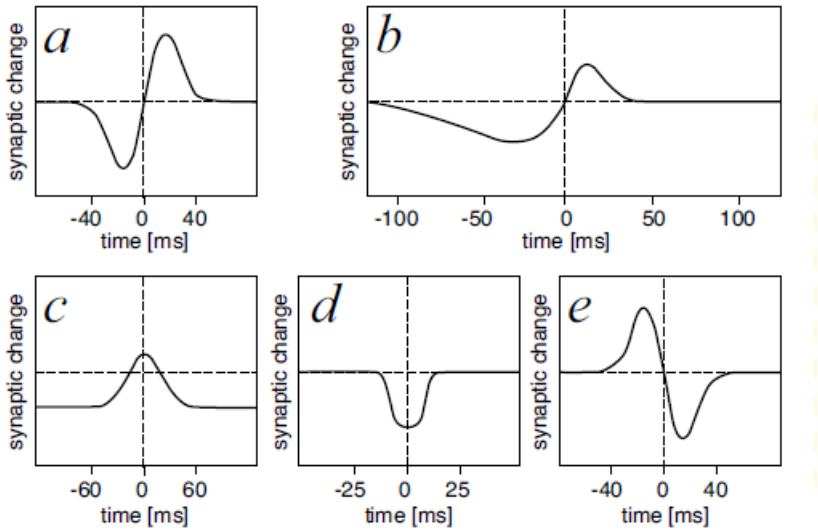

Figure 1.6: STDP rules. Positive times correspond to the postsynaptic spike following the presynaptic spike, negative times the opposite. (a): antisymmetric Hebbian rule; (b): antisymmetric Hebbian rule with differential dynamics; (c): symmetric Hebbian rule; (d): symmetric anti-Hebbian rule; (e): asymmetric anti-Hebbian rule. modified from [15]

Synapses change their strength according to the activity of both pre-synaptic and post-synaptic neurons. This property, called plasticity, is assumed to be associated with synapse formation, pruning and learning in general. Plasticity can introduce two kinds of alteration in the synaptic transmission: long-term potentiation (LTP), and long-term depression (LTD); these concepts are associated with a persistent increase or decrease in the amplitude of the excitatory postsynaptic potentials (EPSP). Don-

ald Hebb [3] was the first to emphasize a causality relation between the presynaptic and postsynaptic spikes suggesting that the efficiency of a connection from a pre- to a postsynaptic neuron is increased if the presynaptic cell contributes persistently to firing the postsynaptic cell. Hebb, however, did not provide a rule for the decreasing of the strength, nor did he address the issue of the effective time window of this causality. Recent experiments suggest that both strengthening, and depression obey the timing of pre– and postsynaptic spikes. First observed by Bell et al. [2], Spike-timing-dependent synaptic plasticity (STDP) is a mechanism to explain the synaptic strength modification based on such spiking order. What has been observed is that in some cases the correlation is consistent with the Hebb's one for which the pre- before post- spike enforce the link, others revealed the opposite. In addition, an Antisymmetric Hebbian rule has been observed in the mammalian cortex, and in cultured hippocampal neurons. It has been proposed to explain the origin of LTP, i.e. a mechanism for reinforcement of synapses repeatedly activated shortly before the occurrence of a postsynaptic spike [12]. It has also been proposed to explain LTD, which corresponds to the weakening of synapses strength whenever the presynaptic cell is repeatedly activated shortly after the occurrence of a postsynaptic spike. This is the rule adopted for our model. The following figures 1.6 show the different kind of Hebbian relations previously presented.

1.2 Review of the HEENS architecture

This thesis project aims to improve the HEENS (Hardware Emulator of Evolvable Neural Systems) architecture, that has been developed by the Advanced Hardware Architecture team of the Department of Electronics Engineering of Universitat Polithecnica de Catalunya (UPC), in order to make it able to embody, and thus execute as algorithms, the Izhikevich neuron model and the STDP rule.
HEENS is an evolution of a previous architecture, called SNAVA ("Spiking Neural-Networks Architecture for Versatile")[16], in which the functionality and the resource occupancy of the elementary processing element simulating the behaviour of the single neuron (PE) is improved, and the architecture can be on-line reconfigured. In fact, HEENS allows to load and run via software different models of neurons and change their synaptic interconnection without re-synthesize the project. This chapter deals with the principal features of the HEENS architecture, better explaining the different phases of execution of the neural algorithm and the design solutions at hardware level which make this architecture specialized and efficient for the neural network purpose. In particular, it reviews the features of the architecture strictly needed in order to understand which are the results of this project and the environment in which it is developed.

1.2.1 HEENS network topology

HEENS is a multi-chip architecture of SNN emulator in real time. The multi-chip structure is implemented by multiple FPGA connected in a ring topology which communicate each other exchanging information under the form of spiking events. In particular, the "Spikes" are encoded in a serial bus with the Address Event representation (AER) protocol, which associates with the information of a spike, the address of the neuron from which it was generated, sending thus its address and not the spike itself.
The ring topology is a Point to Point communication scheme with a Master chip (MC), which accounts for the set-up operations of the entire network, and several slave chips called Neuromorphic chips (NC). Each chip can be seen as a node of the network and during the execution of the neuronal operations the MC acts as a normal NC. The ring topology is a convenient scheme, because it allows to change the desired number of connected FPGAs, and thus to expand the network, without any change in the communication protocol (i.e. the way in which the information is shared and different chips communicate) if not the updating of the number of "nodes" in the network. Information transit across every chip of the network and, inside the NCs, the neuronal algorithm is executed by many neurons (i.e. PE) in a SIMD multiprocessor array called PEs array. Fig.1.7, gives an idea of the ring topology.

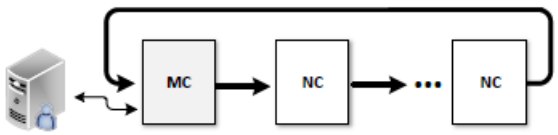

Figure 1.7: Point to Point ring topology with a Master chip and several Slaves chips [5]

The details of the AER protocols, which are explained in [5], are not treated in the following. In the chapter are instead detailed the operation phases of the architecture and the hardware details of the array of processing elements (PEs array) inside each NC.

1.2.2 Operation phases of HEENS

HEENS founds its functioning emulating the biologic behaviour of the neurons.

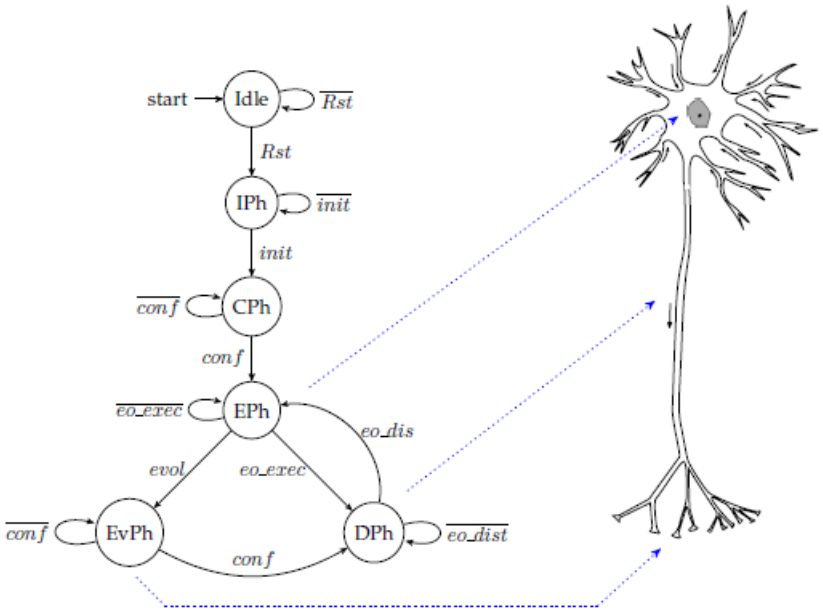

Figure 1.8: Operation cycle and the corresponding of each phase in the biologic structure of the neuron. [18]

Its operation cycle includes five phases executed strictly in a sequential way:

- *Initialization (IPh)*: All the operation for the multi-chip platform set up are executed. Each node is identified with an ID and the size of the ring is specified. Thus, basically it configures the ring network.
- *Configuration (CPh)*: All the configuration data needed by each PE inside each chip are now dispatched. This data are the neural parameters, synaptic connection and weights as well as the full instruction code to be executed for the desired neural algorithm.
- *Execution (EPh)*: This phase acts to emulate the operations done by the soma. The neuronal algorithm is processed and variables like membrane potential, recovery potentials, synaptic weights are updated. Each PE uses a complete set of its own parameters.
- *Evolution (EvPh)*: At the end of each cycle every NC becomes receptive with respect to an evolution command sent by the MC. If it is present, the involved neurons are modified according to incoming information before of the new cycle.
- *Distribution (DPh)*: The spikes arisen during the execution phase are dispatched, according to the synaptic connections fixed during the configuration phase, from a PE to another placed inside the same NC or in a different one. In both cases the dispatch is done through the serial bus. If the spike comes from a neuron (i.e. PE) of the same NC the spike is said to be "local", otherwise it is "global". This phase can be thought as corresponding to the axon.

Fig.1.8 remarks the correspondence between the different phases of operation and the biological structure of a neuron, explaining the evolution of the operation cycle driven by some control signals, used as flags to jump from a phase to another. Once that the IPh and CPh are done, the network

basically executes repeating an emulation cycle, made by EPh and DPh. The Fig.1.9 better explains this concept in the form of a timing diagram.

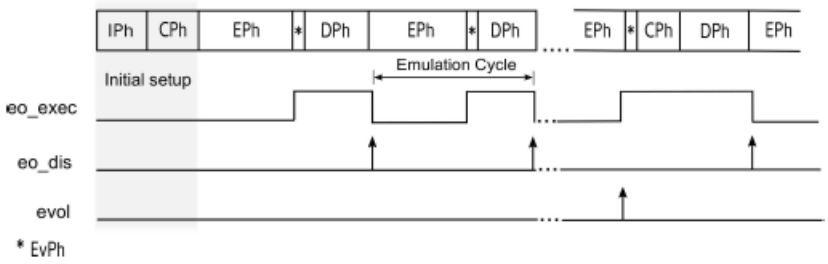

Figure 1.9: Operation cycle can be thought as the sum of two components: The setup made by IPh and CPh, the emulation cycle made by EPh and DPh. According to the control signal "evol", the evolution phase can be executed.[18]

1.2.3 HEENS multi-processor

The scheme of the multi-processor which corresponds to each NC of the network is presented in Fig.1.10. Each multiprocessor executes the neuronal task according to the user settings. The HEENS multiprocessor is a SIMD array, with a single control unit and an array of PEs, each identified according to its position in terms of row and column. The SIMD structure is very useful to reduce area without losing performance according that every neuron executes the same program. In fact, no Jumps are executed if a condition is taken, simply the PEs that do not execute the condition are disabled (frozen). Each multi-processor is basically made by:

- Communication buses
- PEs array
- Control unit
- AER-SRT controller

The **communication buses** allow the transition of configuration data and execution data from and toward the array of PEs. As shown in Fig.1.10, $HEENS\_add$ and $HEENS\_data$ are received by each of PE using an address/data format. These data are multiplexed by the "config" signal, dependent on

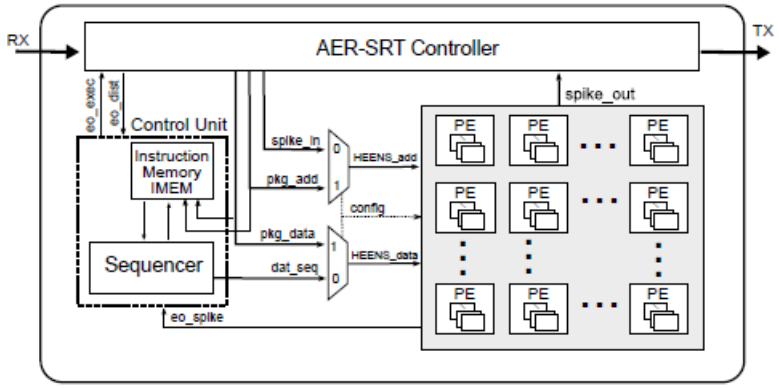

Figure 1.10: HEENS multi-processor blocks view.[18]

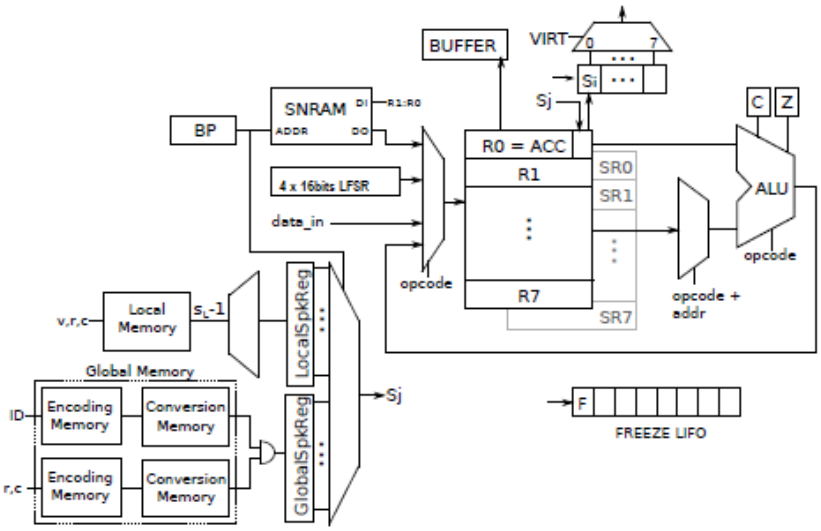

Figure 1.11: Internal structure of a single PE.[18]

the phase of operation, which chooses between configuration signals (*pkg_data*, *pkg_add*) or instructions from the sequencer (*data_seq*) and the incoming spike (*Spike_in*). *Spike_in* encodes the spike information under the form of chip address, row, column, virtual layer (more in the following). The **AER-SRT controller** is the interface between the NC and the serial bus, which receives with the signal *Spike_out* the spikes produced into the NC. Further details about the encode of instructions and control signals are not strictly necessary for the continuation, so they are not treated. See [5] for further details.

PE array is a matrix of PEs. There are NxM physical PEs, organized in a square matrix logic structure. Each PE, however, is multiplexed in time in order to be able to execute up to 8 neurons laying in different virtual layers. Each PE (neuron) is thus addressed by its row (from 0 to $N-1$), its column (from 0 to $M-1$), its virtual layer (from 0 to $vlayers-1$). Each of this set up parameters is fixed at configuration step and editable by the user in the python script "param.py" in appendix B. This file is heavily commented and will be considered again in these pages to better explain the size and configuration of the network.
The single **PE** has the structure shown in Fig.1.11. Basically, it contains a bank of general-purpose registers with shadow registers supporting "move" instructions from and toward the register R0. R0 is, in fact, the accumulator where the ALU result is stored, and arithmetic instructions execute an operation involving R0 and the chosen register. The *ALU* has a Zero flag and a Carry flag, it performs 16 bits fixed point logic and arithmetic operations. The architecture can perform signed addition, subtractions and multiplications (implemented with DSPs) and, during this project, unsigned additions and multiplications have been introduced. The register bank input is multiplexed between ALU result, data coming from outside the PE, the content of the SNRAM memory and the content of four LFSR register used to implement Gaussian Noise (more in future chapters). *BP* is the "block ram pointer" (Bram pointer), addressing the SNRAM. *SNRAM* is the Synaptic Neural Memory where neural parameters, synapses parameters, seeds of the LFSRs and shadow registers for all the virtual layers of the PE are stored. As said before, the PE can receive local spikes (i.e. coming from the same NC) and global spikes (i.e. coming from other nodes). *local memory* stores in each row the synapse number associated with the incoming spikes. For example, if the neuron 0,3,0 (virt, row, col) address

the lcl memory, at that position of the memory it is written which is the synapse associated with that neuron. The number of bits for local memory is:

$$Nofbits = 2^{v+r+c} * log_2(S_L - 1) \tag{1.6}$$

Where S_L is the total number of synapses supported. After the decoding of the $S_L - 1$ word, the synapse position associated with the incoming spike is stored in the "local Spike register", each of whose bits are inputs for the multiplexer with BP as selection signal. For each memory position, the corresponding bit of the "local Spike register", which is basically a flag, becomes the output of the multiplexer and indicates the presence of a spike. When the $LOADSP$ instruction is performed, the Spike flag (Sj) is stored in the LSB of the accumulator (R0(0)), and the registers R0 and R1 are filled with the content of the row of the $SNRAM$ addressed by BP. *Global memory block* model the connections between different NCs in a similar way. In this case there are more fields to be decoded. This project does not deal with global connections, for further details refer to [5] and [16].
The $freeze\ LIFO$ allows the processor to execute nested conditional instructions. In HEENS there are no jump instructions, but all the PEs execute the same code. If a condition is taken, part of code is executed as it is, if it is un-taken, that part of code is replaced by NOPs and the hardware is frozen. There are 8 levels of nested freeze conditions in HEENS, supported thanks to a LIFO in which the sequencer PUSH a '1' when the condition is taken and then POP it when the conditional part is over. Conditional operations depend on the status of the "Z" (Zero) and "C" (Carry) flags of the ALU. They are *FREEZEC* (freeze if Carry is '1'), *FREEZENC* (freeze if Carry is '0'),*FREEZEZ* (freeze if Zero flag is '1'),*FREEZENZ*(freeze if Zero flag is '0'). The conditional part ends with the instruction *UNFREEZE*.

Control Unit manages the flow of data and instructions toward the PE array thanks to the Sequencer activity. HEENS-MP is a SIMD Harvard architecture, the processing elements have their own parameters in SNRAM but global data and instructions are fetched from a single memory block called "IMEM". These data arrive to the PE in the signals "data_in" and "opcode". For sake of simplicity, no further details are treated here. The appendix A shows the full custom Instruction Set, the kind of operations which the architecture is able to perform, with relative explanation.

1.2.4 System Setup and Software contribute

System Setup is fully made by means of several python scripts, some of the most important files they receive in input are:

- Assembly file: describes the routine executed by all the neurons.
- Netlist file: defines the synaptic connections between the neurons of the network.
- Neuron file: declares the value of the parameters to be stored in SNMEM.
- Mnemonic file of instructions declares instruction mnemonic words and their machine-level values.

Appendix B shows the file *params.py*, which fixes the dimension of the PE array and of the memory blocks inside the PE. During the Setup of the system, the **assembly code** is read and compiled in order to assign the right space in memory to store neural and synaptic parameters and to write the compiled machine code in the "IMEM". The assembly files developed in this project are reported in appendices and will be described in future chapters.

The **Netlist file**, shown in appendix D, is read in order to write each connection in the *lclmemory* and store the synaptic parameters, in it specified, in the *SNRAM*. The **Neuron file**, shown in appendix E, fixes and organizes the neural parameters to be stored in SNRAM specifying their memory location. The two appendices before cited, explain in detail the structure of the two files.

1.3 Motivation and objectives

This chapter presented the main properties of the NNs and of the HEENS architecture, focusing on the SNN which the HEENS architecture aims to emulate. The SIMD structure, the AER protocol with the addressing of the spikes and the neural phases presented, make HEENS able to be intrinsically efficient in the main SNN tasks and in reproducing its biologic features. If, therefore, the architecture is efficient and well customized on its purpose, an equally efficient and biologically faithful model must be adopted in order to emulate in the architecture the actual neural activity. As it has been discussed, using biophysically accurate Hodgkin–Huxley-type models is computationally prohibitive, since it can simulate only a handful of neurons in real time. On the other end, using an integrate-and-fire model is computationally effective, but the model is unrealistically simple and incapable of producing the dynamics exhibited by cortical neurons. The need for more biological accuracy makes the Izhikevich model preferred and a good trade-off between cost and performance. The proposal of this project is thus, as first objective, to map in HEENS architecture the Izhikevich model described in 1.1.5. This allows to simulate large-scale networks of spiking neurons with a "one-fits-all" choice of the function in 3.2, but also to reproduce the behaviour of a single neuron of multiple types according to different parameter values. The second objective of this project is the implementation in HEENS of an STDP algorithm inspired by that shown in [10] and customized to the resources of the architecture. For a neural network is indeed fundamental to be able to evolve according to some rules which consider the behaviour of the system as a whole. As already described, STDP is a rule which shapes the strength of a synaptic connection and thus the connectivity between neurons according to the inter-spike timing. To embody this model, therefore, makes it possible for the network to evolve over time according to its activity, specializing in a task with a biologically recognised dynamic.
In this project several neural models have been developed and several networks have been tested combining the different properties until now described, in order to appreciate the different dynamics. In fact, all the tested models integrates the Izhikevich neuron, while the outcomes of the application of the STDP with different granularity (to the entire network, or to selected neurons, or to selected connections) have been tested.

CHAPTER 2

Izhikevich algorithm mapping on HEENS

2.1 Izhikevich algorithm

One of the targets of this thesis project is the mapping on HEENS architecture of the Izhikevich neural model, which has been already presented in section 1.1.5. In order to better describe its algorithmic features, we can refer to the folowing MatLAB script, originally included in a paper published by Izhikevich in 2003. Since the routine performed in HEENS has the same structure, it can be useful to analyze in detail the different blocks of this program, each shown with a different colour:

```
% Created by Eugene M. Izhikevich, February 25, 2003
% Excitatory neurons Inhibitory neurons
Ne=800; Ni=200;
re=rand(Ne,1); ri=rand(Ni,1);
a=[0.02*ones(Ne,1); 0.02+0.08*ri];
b=[0.2*ones(Ne,1); 0.25-0.05*ri];
c=[-65+15*re.^2; -65*ones(Ni,1)];
d=[8-6*re.^2; 2*ones(Ni,1)];
S=[0.5*rand(Ne+Ni,Ne), -rand(Ne+Ni,Ni)];

v=-65*ones(Ne+Ni,1); % Initial values of v
u=b.*v; % Initial values of u

firings=[]; % spike timings

for t=1:1000 % simulation of 1000 ms
I=[5*randn(Ne,1);2*randn(Ni,1)]; % thalamic input
fired=find(v>=30); % indices of spikes
firings=[firings; t+0*fired,fired];
v(fired)=c(fired);
u(fired)=u(fired)+d(fired);
I=I+sum(S(:,fired),2);

v=v+0.5*(0.04*v.^2+5*v+140-u+I); % step 0.5 ms
v=v+0.5*(0.04*v.^2+5*v+140-u+I); % for numerical
u=u+a.*(b.*v-u); % stability

end;
plot(firings(:,1),firings(:,2),'.');
```

The first block initialize the number of "excitatory" and "inhibitory" Neurons. Even if the definition of excitatory or inhibitory is not well placed from a biological point of view, due to only the synapses

can be of a type, in this model we imagine that excitatory neurons create only connections with a positive synaptic weight and inhibitory with a negative sign. According to several biologic results, is assumed the presence in the Brain of a quarter inhibitory synapses w.r.t the excitatory ones. As can be noticed, this ratio is maintained in this program and will be in whole this project.
In the first block, all the neural parameters and the synaptic weights "S" are generated randomly according to the rectangular distribution "rand". These values are as well printed on file in order to be used by HEENS architecture for its simulation (more in the following). Note that Inhibitory Synapses (the Ni columns from the column $Ne+1$ of the S matrix) are stronger then excitatory ones (the first Ne colums).
In the Second block, membrane potential "v" and recovery potential "u" are initialized. The initial membrane potential of all the neurons is fixed at $-65uV$. The "firings" matrix is then initialize as an empty structure, it will be filled at each cycle with the addresses of the spiking neurons in order to print the spike pattern. In the third block is described the execution loop. This program aims to simulate 1000 cycles, and due to biologically every cycle is pretended to be executed in $1ms$, the simulation is of $1000ms$. In the third block, thalamic input noise "I" is defined according to a MatLAB Gaussian distribution function called "randn", then the neuron which fired (whose membrane potential overcomes the threshold of 30uV) are identified and stored in the "firings" matrix and their potentials are updated according to:

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.1}$$

$$u' = a(b * v - u) \tag{2.2}$$

with the auxiliary after-spike resetting:

$$if\ v \geq 30uV,\ than\ v = c\ and\ u = u + d \tag{2.3}$$

The parameter "c" represent thus the resting potential, i.e. the potential of reset after spike, different for all the neurons. It worth to stress that the thalamic input is stronger (the randn function is multiplied by 5) in the excitatory neurons than in the inhibitory ($randn * 2$). The fourth block includes the numeric version of the Izhikevich set of differential equations, in which, for sake of stability, the membrane potential routine is divided in two steps. It is worth to stress that in the "I" term are contained both the thalamic input and the sum of the synaptic weights of the fired connections. At the end of the file a firing pattern is plotted, it is a graphical representation of what neuron has spiked and in which time instant. An example for a network of 40 neurons and a simulation of $1000ms$ is shown in Fig.2.1

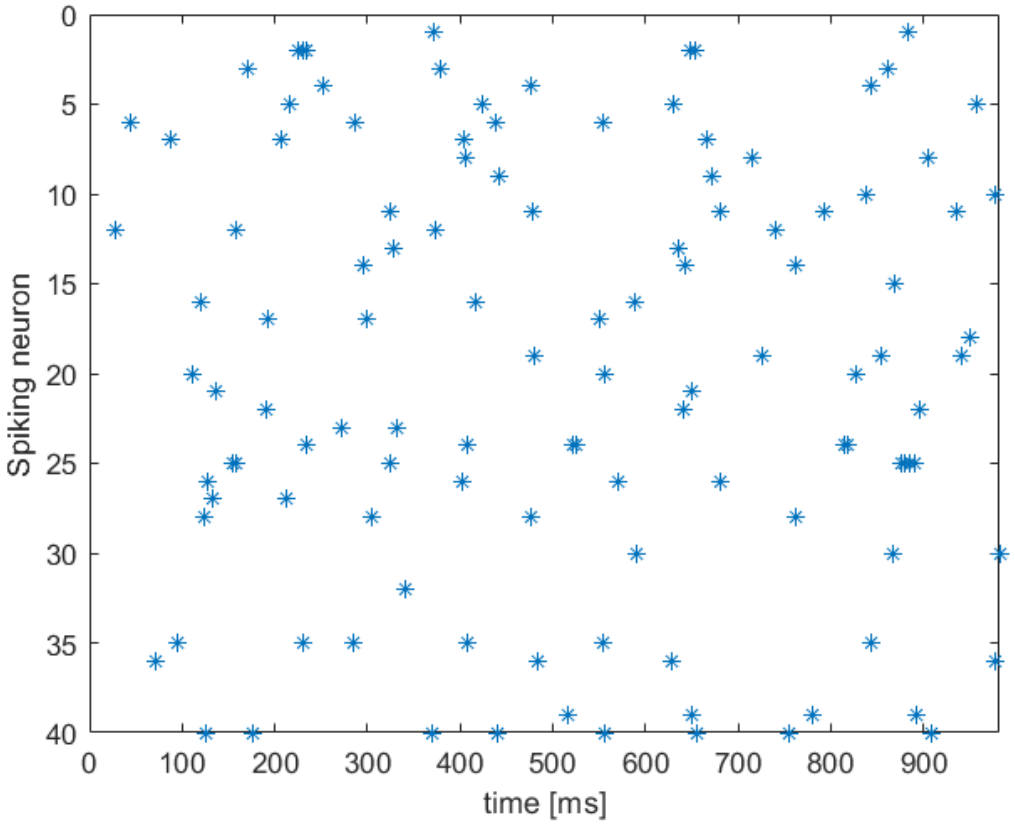

Figure 2.1: Example of the firing pattern for a network of 40 neurons and a simulation of $1000ms$

2.2 Matching HEENS and MatLAB model

Once analyzed the algorithm, must be clarified what grade of compatibility this can have with the HEENS architecture. In fact, there some important differences between the already presented algorithm and what can be performed in HEENS:

- MatLAB model supports floating point operations, this is not reproducible in HEENS where only fixed point operations can be performed. A fixed point model must be inserted in the MatLAB script.
- HEENS architecture doesn't perform any kind of hardware rounding. This could affect the possibility to obtain improved results in the hardware emulator and match the MatLAB results. Some kind of rounding must be inserted in HEENS.
- The Gaussian Noise by which is modeled the thalamic input is not supported in HEENS. A Hardware Gaussian Noise generator must be designed and its model must be inserted in MatLAB in order to make the HEENS and MatLAB simulations compatible.

Based on what has been seen, in order to simulate the algorithm before of the actual mapping, make some important design choices and then compare the floating point "infinite precise" model with the fixed point one actually implemented in HEENS, it is of great interest reproduce HEENS computational capability in MatLAB. This means introducing in the MatLAB script some custom functions which emulate real operations in HEENS. The following sections deal with this necessity.

2.2.1 Fixed point model

The "fixed point designer" app of the MatLAB suite allows the generation of a custom MatLAB function executing operations with fixed precision. The user is requested to specify the format and decimal precision for each variable, the parallelism of arithmetic operations, the type of rounding to be performed and several other parameters (For further detail refer to Appendix I). These kind of features allow to reproduce the arithmetic capability of HEENS and eventually to shape it according to needs.
In order to find the minimum decimal precision able to well approximate the floating point model, three fixed point membrane potential models are tested, with 5, 6 or 7 fractional bits respectively in a 16 bits word. In all the three cases, the tested network features 16 fully connected neurons with randomly generated parameters. A 8-decimal digit model is not considered because, analyzing the floating point model, can be found that the membrane potential minimum value is around -90uV. With such dynamic, an 8-fractional digits representation in HEENS would for sure saturate performing the Izhikevich algorithm. The results are shown in Figs.2.2, 2.3, 2.4, which represent the membrane potential of three neurons for each resolution value. Only the version with 7-fractional bits gives reasonable performance and doesn't present parasitic spikes.
A binary number with 7 fractional bits represents numbers with a resolution of $7.81 * 10^{-3}$, and thus, this number can be considered the "unitary scale value" of the system. Considering the physical quantity of the neural model, in order to facilitate debugging and design, the final chosen resolution is the near $10 * 10^{-3}$, in Volt $10 * 10^{-3} mV$. In HEENS, therefore, every Voltage quantity of the algorithm will be expressed in multiple of $10uV$ and the LSB, thus, corresponds to this value.

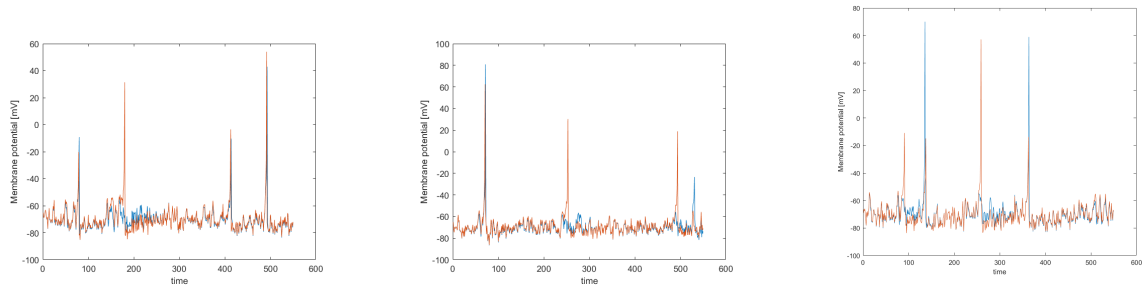

Figure 2.2: Comparison between floating point and 5-decimal digit fixed point models for 3 neurons.

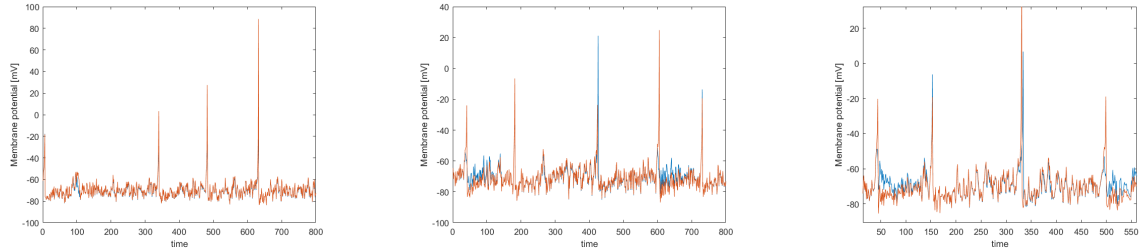

Figure 2.3: Comparison between floating point and 6-decimal digit fixed point models for 3 neurons.

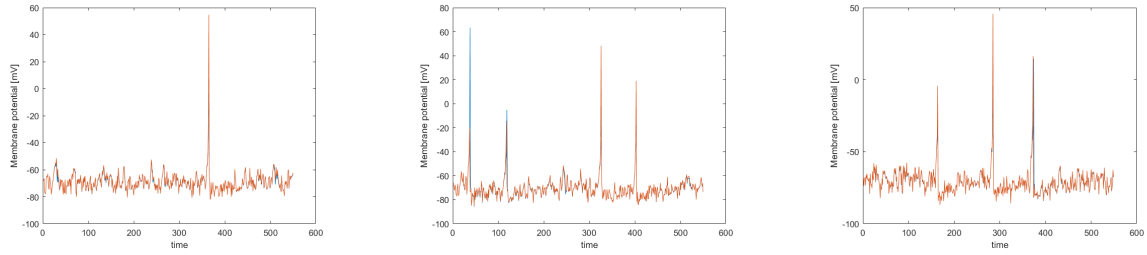

Figure 2.4: Comparison between floating point and 7-decimal digit fixed point models for 3 neurons.

2.2.2 Introduction of rounding in HEENS

In the section 3.9 it is shown as the MatLAB fixed point model with "Nearest" rounding (which rounds toward the nearest integer) has exactly the same spiking pattern of the infinite precise version, while the model with the "Floor" rounding (which simply truncates the results) makes several mistakes. To better emulate the ideal performance, it is thus necessary to introduce in HEENS some form of rounding with the minor possible hardware cost. With this aim, the following rounding solutions have been developed and introduced in the architecture:

- Rounding to the nearest integer when the arithmetic shift "SHRAN" instruction is performed. No rounding is executed with the non-arithmetic shift instruction "SHRN".
- Conditional rounding when "MUL" or "MULS" instruction are performed. The MSbit of the LSW of the result is put in the carry flag after each multiplication, with a "FREEZEC" in-

struction the content of the flag can be verified and eventually the MSW is incremented by one.

These solutions provide an "on demand" rounding when multiplications are performed, with thus a code overhead, and an automatic rounding when a shift is performed. Due to the most of operations are performed with shifts, a good precision is achieved without a big code overhead.
For sake of completeness, the Appendix M shows the ALU of the PE described in VHDL. The rounding of the result of an arithmetic right shift, in particular, is described from line 329.

2.2.3 Gaussian noise introduction

As said before, Izhikevich algorithm includes a term called "I" which is mathematically represented with a Gaussian Noise of different intensity if considered applied to an excitatory or an inhibitory neuron.
HEENS Processing element features a rectangular (or uniform) noise generator, made by a 64-bits LFSR, which is not actually needed by the Izhikevich model. It is thus replaced by an algorithmic emulation of a Gaussian Noise obtained thanks to four 16-bits LFSRs in Galois configuration.
As well known, an LFSR is simply a shift register whose input bit, at each clock cycle, is a linear function of its previous state. The initial value of the LFSR is called "seed" and, due to the stream of values produced by the register is completely determined by its previous state, the operation of the register is deterministic and thus the same state can reappear after a complete cycle. However, with a well chosen feedback function, the repetition cycle of the input pattern is very long, and thus it can be considered a generator of pseudo-random values. Galois LFSR is simply an LFSR characterized by a particular feedback function, which is shown in 2.5.

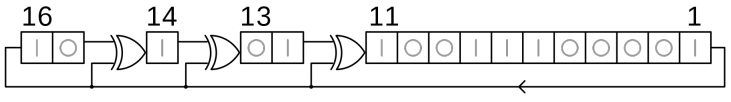

Figure 2.5: Galois 16-bit LFSR [17]

In [11] Gaussian noise is obtained from the uniform noise made by four LFSRs by means of shift and additions exploiting basically the central limit theorem. The logic structure proposed in [11] and used in our design is that of 2.6.
However, this kind of implementation would need the presence of three additions, two made in parallel, and thus the physical presence of at least two arithmetic units, which makes it quite hardware expensive. If compared with the actual noise generator, moreover, this solution appear to be far more expensive. In order to maintain approximately the same cost of the old implementation, all the operations are executed in firmware by means of the unique ALU of the Processing element. Gaussian noise generator is therefore realized as a routine of the assembly source file, whose functioning is based on a matrix of four VHDL described 16-bits LFSRs, whose states are loaded in parallel in the register file.

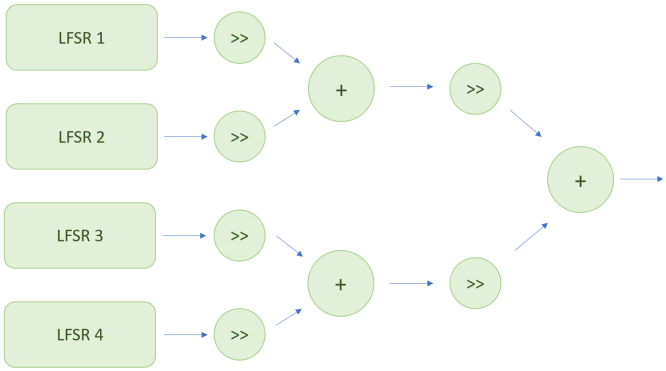

Figure 2.6: Model for generating Gaussian noise distribution with 4 LFSRs

Once realized a MatLAB model of the matrix of four galois LFSRs (see Appendix J), the real quality of the noise generated is verified thanks to the custom MatLAB script attached in Appendix K, where a distribution of 10^5 pseudo-random values obtained by simulation is mapped against the best-fit Gaussian distribution (MatLAB *fitdist* function). The first picture of Fig. 2.7 show the histogram obtained by the 10^5 samples randomly generated by the 4-LFSRs system, together with the line interpolating the histogram (in red) and the best-fit Gaussian shape (in yellow). The second picture of Fig. 2.7 shows the error (i.e difference) obtained comparing, in the center of each bin of the histogram, the value of the bin and the value of the interpolating shape. Considering the mean value of the square of the error just described, can be obtained the MSE (Mean square error), which is the average squared difference between the estimated values and the actual value and thus corresponds to the expected value of the squared error loss. When the Gaussian distribution generated by the 4-LFSRs model is compared with MatLAB best-fit function the MSE, results to be:

$$MSE_{LFSR} = 1/N * \sum_{n=1}^{N} (Y_{LFSR} - Y_{Gaussian})^2 = 0.0461 \; with \; N = samples \tag{2.4}$$

In order to appreciate the quality of the results, this value can be compared with the MSE obtained considering the Gaussian distribution made by the "randn" MatLAB function, which results to be of the order of $MSE_{RANDN} = 0.0130$.
The two pictures in the bottom of Fig.2.7, instead, show the shapes of the Gaussian distribution resulting from the 4-LFSRs model and the "randn" MatLAB function in the case of 10^6 samples.
In fig.2.8 the main logic blocks of the *Gaussian_noise* subroutine developed in HEENS are described. At the beginning of the algorithm an instruction called SEED initialize all the LFSRs, then two instructions called "RANDON" and "RANDOFF" enable the evolution of the states, then "LLFSR" instruction loads the state of the four Galois LFSRs in registers R0, R1, SR0, SR1. An algorithm of shifts and additions is thus performed together with a final portion of the routine which recognizes what kind of neuron (exc. or inh.) is the PE emulating (remember that all the PEs execute exactly the same algorithm) and multiplies it by different coefficients. This implementation actually allows to obtain a good and low cost Gaussian noise, with the overhead of 35 clock cycles, which is a reasonable outcome.
The *Gaussian_noise* subroutine is included in the assembly file of Appendix C and its full custom MatLAB model called "*LFSR_Gaussian_noise*" is introduced in *izhi_net.m* program in order to compare the simulations made in MatLAB and those made in HEENS, at RTL level, according to exactly the same noise source.
The *izhi_net.m* program and its simulation results are the topic of the next section.

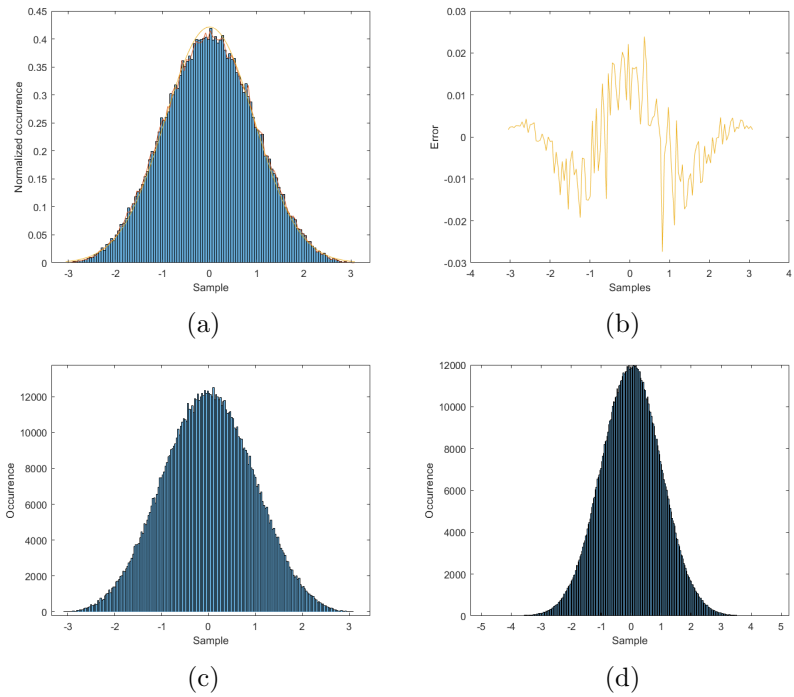

Figure 2.7: a) Distribution of pseudo-random values obtained by the 4-LFSR model is mapped against the MatLAB best-fit Gaussian distribution. b) Error between 4-LFSRs model distribution and the best-fitted Gaussian shape. Distribution made by 4-LFSRs model (c) and by "randn" function (d) with 10^6 samples.

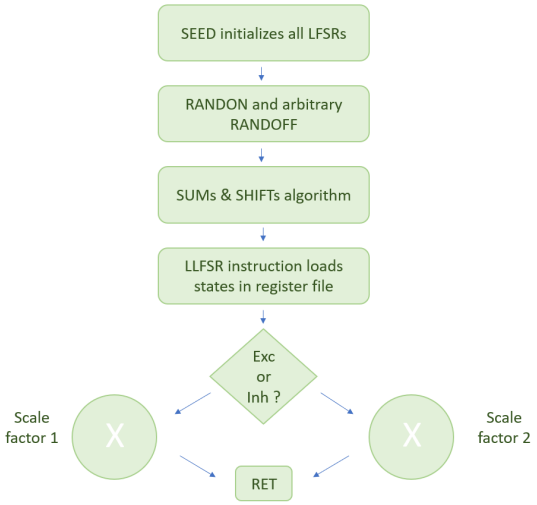

Figure 2.8: *Gaussian_noise* subroutine blocks view

2.3 MatLAB simulation

Appendix H contains the entire MatLAB code of the *izhi_net.m* program including both the floating point "infinite precise" and the HEENS-like versions of the Izhikevich neural algorithm. The program is inspired by that presented in the first section and modified according to what said in the previous section. Thus, the fixed-point precision of HEENS is emulated with the 7-fractional part function described before (which corresponds to have a resolution of $7.81uV$), and the Gaussian noise is generated by the 4-LFSR structure of the previous section. Just setting the Number of excitatory and inhibitory Neurons (again, this is not an accurate definition as said in 1.1.5) and running the program (the STDP part is to be considered commented in this section), a fully connected network is simulated for the number of cycles decided in the "for" instruction. In addition, at the end of the code, the program prints in the same graph the membrane potential temporal evolution of all the neurons of both models, and the superimposition of their spiking patterns. The figures in 2.9 show the comparison between the spiking pattern of the floating point model (o) and the fixed point model ($*$) for a $550ms$ simulation. The figures show the results in two case: in the left the fixed point model uses the "Floor" rounding which simply truncates the result, in the right the model supports the rounding toward the nearest integer value, called "Nearest". As can be seen, the first case presents several parasitic spikes which would totally drive out the network evolution, while in the second case the two spike patterns are identical. Therefore, the insertion in HEENS of the two rounding techniques already discussed in 2.2.2 is necessary.

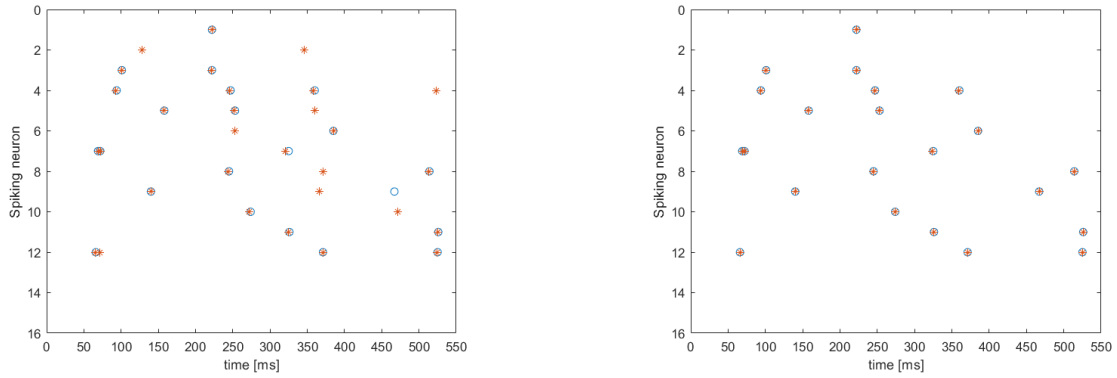

Figure 2.9: Comparison between the spiking pattern of the floating point model (o) and the fixed point model (*). a) "Floor" rounding performed, b) "Nearest" rounding performed.

It has been thus demonstrated that the fixed point model with a $7.81uV$ resolution and the "Nearest" rounding can emulate faithfully the floating point model. Thus, these are the kind of features that must be mapped on HEENS.
Next section details the generation of input files for HEENS architecture via MatLAB scripts. Then, in 2.4, the mapping of these parameters in HEENS is detailed.

2.3.1 MatLAB setup scripts

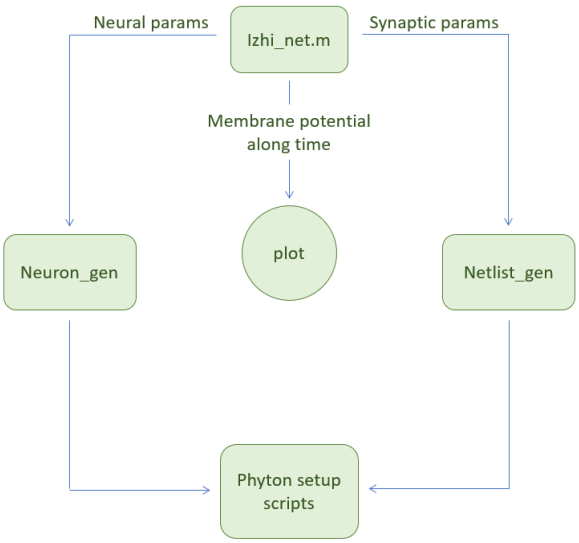

Figure 2.10: Purpose of the MatLAB scripts in this project

As said in 1.2.4, the entire HEENS architecture is setup by means of python scripts, whose principal input files have been already described and among which there are the files *Neuron.txt* and *Netlist.lst*. As can be noticed looking at the MatLAB program described in 2.1, some of the neural parameters to be stored in *Neuron.txt* and *Netlist.lst* need to be randomly generated according to bio-inspired rules and therefore the entire files need to be automatically generated for sake of simplicity. For example, as described in Appendix D and E, each of the number in the *Neuron* file is the decimal conversion of two randomly generated and concatenated 16-bits binary numbers representing neural parameters a, b, c, d, u, v and so on.
In order to easily generate these random parameters, *Netlist* and *Neuron* files are generated via the MatLAB scripts *netlist_gen* and *Neuron_gen* shown in **Appendix F** and **Appendix G**. In this way, the entire Neural Network is simulated in HEENS and in MatLAB starting from the same parameters, and executing exactly the same algorithm, in order to verify and compare the correct functioning and the reliability of the results obtained in HEENS.
Fig.2.10 shows how the effort of MatLAB in this project can be summarized. First it is executed the program *izhi_net*, which generates all the parameters and simulates the network, then the scripts *netlist_gen* and *Neuron_gen* before discussed generate the *Netlist.lst* and *Neuron.txt* input files needed by python scripts. Next section describes how these parameters are stored and managed in HEENS.

2.4 SNMEM organization for Izhikevich model

This section aims to describe how data necessary to execute the Izhikevich algorithm are stored and managed in HEENS. With the already cited file *Neuron.txt*, the data to be stored in memory are provided to the compiler together with their destination address. Synaptical neural memory contains 32-bit words, at each "LOAD" instruction the 16 most significant bits (16MSb) are moved to R1, the 16 less significant bits (16LSb) are moved to R0. In each memory row is therefore a 32-bit word which is the concatenation of two 16-bit words. In particular, for each neuron, the parameters of 2.1 are specified:

Memory row	16 MSBs	16 LSBs
$NEUaddr0$	membrane potential "v" $Q16.0$	recovery potential "u" $Q16.0$
$NEUaddr0+1$	parameter "b" $Q16.0$	parameter "d" $Q16.0$
$NEUaddr0+2$	parameter "a" $Q16.0$	parameter "c" $Q16.0$
...	...	...
$Noiseaddr0$	16 bit seed	16 bit seed
$Noiseaddr0+1$	16 bit seed	16 bit seed

Table 2.1: Neural parameters stored in memory

Where the format $Q16.0$ indicates an integer number with the sign bit, 15 integer digits and '0' decimal digits. It is worth to stress again that the four seeds are loaded in the four LFSRs used for Gaussian noise generation.
All the neural parameters before described are loaded in register file at the beginning of the operations and moved during the algorithm execution, with the particular case of the membrane potential which is the only data which never change position but always remain in R2 in order to be monitored during simulation (more details in future). The synapses as well are written in SNRAM through the file *Netlist.lst* and the memory organization for each neuron "i" linked to the neuron "j" with $0 \leq j \leq 15$ (where $J = 0$ stays for the neuron itself), results to be:

Memory row	16 MSBs	16 LSBs
SYN addr0	weight $P_{0,i}$	S_0
SYN addr0 +1	weight $P_{1,i}$	S_1
SYN addr0 +2	weight $P_{2,i}$	S_2
...	...	...
SYN addr +15	weight $P_{15,i}$	S_{15}

Table 2.2: synapses stored in memory

The work of this project is mostly focused at algorithmic and instruction level. However, In order to fulfil to computational exigences of the algorithm such as the presence of the square elevation, of conditional instructions, of the noise routine, some modifications in hardware and an extension of the instruction set were necessary. Without going into detail, beyond the already described rounding techniques, the original instruction set has been expanded with an unsigned multiplication and an unsigned addition, together with a modification of the "LLFSR" instruction which loads the content of the four registers in parallel into R0,R1,SW1 and SW2.
Next section details the mapping of the Izhikevich model in HEENS through the assembly program, in which the whole neural algorithm is described at instruction level.

2.5 Assembly file

The assembly program executing the Izhikevich neural model is included in Appendix C. As said, this assembly file describes the entire neural routine performed in HEENS. Just to sum it up, it has been mentioned before that during the set up operation of the architecture (made in the CPh), the assembly program, the netlist and the neuron files are the inputs for some phyton scripts which creates the memories of each PE, compile the assembly code creating the machine level program to be written inside the "IMEM" of the NC and set up the array of PEs. In this section, a brief overview of the structure of the assembly program is done. Each block is described in order to better explain its function, the hardware resources exploitation, the main program tasks and compare this program with its MatLAB version $izhi\_net.m$.

In **blue** the first piece of code of the program is shown :

- A $define$ section states how many virtual layers are involved in this model, the number of global synapses and the number of steps in which the membrane potential algorithm is performed. According to the MatLAB program published by Izhikevich, the membrane potential evolution is divided in two steps for numerical stability.
- For sake of simplicity and graphical result fruition in "QuestaSim" environment, in which HEENS architecture has been simulated, only one virtual layer is used in this program. Nevertheless the whole model is designed to support 8 virtual layers, in fact 16 synapses are assigned to each virtual layer in the section "Virtual layers".
- Constants and parameters are then defined. Each voltage quantity is of course written as multiple of the $10uV$ unit, each number $n \geq 1$ is written as it is, and each number $n \leq 1$ is written as itself multiplied for 2^{16} in order to be right shifted after the multiplication taking only the 16 MSBs of the 32-bit result.

```
; Network definitions

define virtual_layers 0 ; From 0 up to 7
define gsynapses 2 ; Up to 32 global synapses
define n_step 1; number of steps-1

.DATA

; Virtual layers
V0 = "0000000F" ; Number of assigned synapses (s-1) to the main layer
[..]
V7 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 7
VLAYERS="00000000" ; Number of virtual layers (n-1).

; Membrane potential parameters common to all neurons

;multiple of 10uV
VTHRES="FFFFF63C"  ; Threshold voltage -25 mV
N70="00001B58"; 70mV,
N0002="000068DC";
N001="00000028F"; 0.01, The noise generated by LFSR is 100 time bigger than necessary.
REST_POT="FFFFE69C"; -6500
N5="00000005";
;
; General constants
```

```
[..]
;
```

The next part of the code is shown in **green**. The first address for synaptic parameters in SNRAM is fixed for all virtual layers. Our algorithm without plasticity involves 1 row for each synapse, thus a total of 16 rows in memory per virtual layer. The same is then specified for Neural parameters in SNRAM: in our model there are 3 memory rows of synaptic parameters stored in memory, thus $NEU\_ADDR1 - NEU\_ADDR0 = 3$.

```
; Neural and Synaptic RAM addresses
SYN_ADDR0="00000000"    ; First address of Synaptic parameters in SNRAM for V = 0.
SYN_ADDR1="00000010"    ; First address of Synaptic parameters in SNRAM for V = 1.
[..]
SYN_ADDR7="00000070"    ; First address of Synaptic parameters in SNRAM for V = 7.
GSYN_ADDR="00000090"    ; First address of Global Synaptic parameters in SNRAM.
NEU_ADDR0="000003E3"    ; First address of Neural parameters in SNRAM (995) for V = 0.
NEU_ADDR1="000003E6"    ; First address of Neural parameters in SNRAM (997) for V = 1.
NEU_ADDR2="000003E9"    ; First address of Neural parameters in SNRAM (999) for V = 2.
[..]
NEU_ADDR7="000003F8"    ; First address of Neural parameters in SNRAM (1009) for V = 7.

SEED_ADDRL = "000003FD" ; Address of noise seed in SNRAM
SEED_ADDRH = "000003FE";Address of noise seed in SNRAM
PEID   = "000003FF" ; Address of  PE Identifier number
;
```

In the following portion of code in **violet** all the subroutines are defined.

```
.CODE
;
GOTO MAIN ; Jump to main program
;
; ************************** PROCEDURES BEGIN ***************************
;
.RANDOM_INIT   ; Uses R0 and R1
[..]
RET
.LOAD_NEURON     ; Uses R0, R1, R2, R3, R5
[..]
RET
.DETECT_SPIKE ; Uses R0,R3,SW5, SW6 and R2
[..]
RET
.SYNAPSE_CALC    ; at time: 15916...
[..]
RET
.GAUSS_NOISE ; Uses SW0, R2, R6 and R7
[..]
RET
.MEMBRANE_POTENTIAL ;Uses R0,R4,R7  32808
[..]
RET
.RECOVERY_UPDATE ;uses R3,R5,R6
[..]
RET
.SUM_NOISE_AND_Ws
[..]
RET
.STORE_NEURON    ;uses R0,R3 and R1
```

```
[ . . ]
RET
; ************************** PROCEDURES END ****************************
```

The **Maroon block** shows the main program which execute the "emulation cycle".

- $LAYERV$ states and saves in memory the information of how many virtual layers are used.
- The subroutine $RANDOM\_INIT$ initialize all LFSRs with the seeds defined in the file $Neuron.txt$.
- The instruction for the compiler $.EXECLOOP$ starts the neural execution loop.
- $LOOPvirtuallayers$ ensures the repetition of the same routine for all the virtual layers.
- Routine starts with $LOAD\_NEURON$ subroutine which stores in register file the neural parameters and the membrane potentials contained in SNRAM. Access in memory of course is time consuming, thus, in order to avoid it, all the needed data are now and not again loaded.
- The subroutine $DETECT\_SPIKE$, verifies if the membrane potential of the neuron overcomes the threshold and eventually push a spike in the LIFO. If a spike is detected, the potentials are updated according to:
$$v(t+1) = c\ ;\ u(t+1) = u(t) + d \tag{2.5}$$
- The first address of synaptic parameters in SNRAM is read by $READMPV\ SYN_{ADDR0}$ and a $LOOPV$ instruction executes the $SYNAPSE\_CALC$ subroutine for the neuron of the virtual layer. In $SYNAPSE\_CALC$ the weights of the synapses which have spiked are added together creating the final synaptic term to be added to the membrane potential of the post-synaptic neuron i. It is:
$$P_i(t) = \sum_{j=0}^{N-1} W_{ji} * S_j(t). \tag{2.6}$$
where $P_i(t)$ is the final input term, j is the index of the pre-synaptic neuron, N is the number of connections (equal to the number of neurons in our fully connected network), W_{ji} is the weight of the synaptic connection from the neuron j to the neuron i, $S_j(t)$ is a binary value which indicates if the neuron has fired or not.
- A sample following the Gaussian distribution is then generated, as already described, by the $GAUSS\_NOISE$ subroutine.
- In the following, the "LOOP" instruction executes twice the half step of the membrane potential algorithm described by the subroutine $MEMBRANE\_POTENTIAL$. it is:
$$v(t+1) = v(t) + 1/2 * (0.004v(t)^2 + 0.5 * v(t) - u(t) + 140). \tag{2.7}$$
- The incoming synaptic input $P_i(t)$ and the thalamic one are added to the membrane potential in the subroutine $SUM\_NOISE\_AND\_Ws$. This last subroutine basically embody the term I in the MatLAB Izhikevich algorithm.
- The $RECOVERY\_UPDATE$ routine execute the membrane recovery variable differential equation:
$$u(t+1) = u(t) + a * (b * v(t) - u(t)). \tag{2.8}$$

- The membrane and neural parameters are stored back in memory with the $STORE_{NEURON}$ routine and the "virtual layers" loop is ended after the increment of the virtual number. When all the virtual layers are executed, the "execution loop" ends and then spiking distribution is performing, thus completing the emulation loop.

```
; ************************** MAIN PROGRAMME BEGIN ***********************
.MAIN
;
; Virtual operation init
LAYERV virtual_layers ; Init sequencer vlayers. It is 0 for non-virtual operation
LDALL ACC, VLAYERS ; Load defined virtual layers to PE array
SPMOV 0 ; VIRT <= ACC

; Initial instructions
GOSUB RANDOM_INIT  ; For noise initialization

.EXEC_LOOP ; Execution loop
 LOOP virtual_layers     ;Neuron loop for virtual operation
     NOP     ;to prevent pipeline error
     GOSUB LOAD_NEURON
     GOSUB DETECT_SPIKE
   [..]
     READMPV SYN_ADDR0
     LOADBP
     LOOPV V0   ;synaptic loop. Reads number of current-layer synapses
  NOP          ;to prevent pipeline error
         GOSUB SYNAPSE_CALC
     ENDL
  SWAPS R3 ;SYNAPSE sum to SW3
  GOSUB GAUSS_NOISE
     LOOP n_step
      NOP
  GOSUB MEMBRANE_POTENTIAL   ;Calculate membrane potential
      GOSUB SUM_NOISE_AND_Ws
 ENDL
 GOSUB RECOVERY_UPDATE
 GOSUB STORE_NEURON
 INCV
 ENDL
.FINISH
NOP          ;Empty pipeline wait NOPs
NOP
NOP
SPKDIS         ;Distribute spikes
GOTO EXEC_LOOP       ;Execution loop
```

Finally, it worth to stress that this program executes the same tasks, in the same order, of its MatLAB version.

2.6 Simulation in QuestaSim

QuestaSim is a multi-language HDL simulation environment by Mentor Graphics for hardware description languages such as VHDL or System Verilog. It offers high-performance and advanced debugging capabilities, it is used in multi-million gates design. Our entire Architecture is loaded in QuestaSim, compiled and executed with tcl macros, the result of the RTL simulations is visible in a timing diagram form with the "Wave" tool of the suite. The wave tool of QuestaSim is the only one used to verify and debug the HEENS architecture during all the phases of this project. In particular, with this tool it is possible to known the content of each register, the ALU operations and results and the presence of spikes can be immediately recognized graphically. Moreover, The "analog format" option allows us to see the time evolution of the neural variables (such as the membrane potential) in a quantized cartesian graph, appreciating its typical shape in a very user-friendly way.
Once the assembly program has been translated and the memories of each PE are written, the VHDL files describing the architecture are compiled and the network is simulated for an user defined time lapse, allowing to visualize as many emulation cycles as needed. Fig.2.11 shows the spike pattern obtained for a simulation in Questasim of 550 cycles. The network is the same already simulated with MatLAB, shown in 2.9 and reported here for clarity. As it can be noticed, the two spike patterns are almost identical.

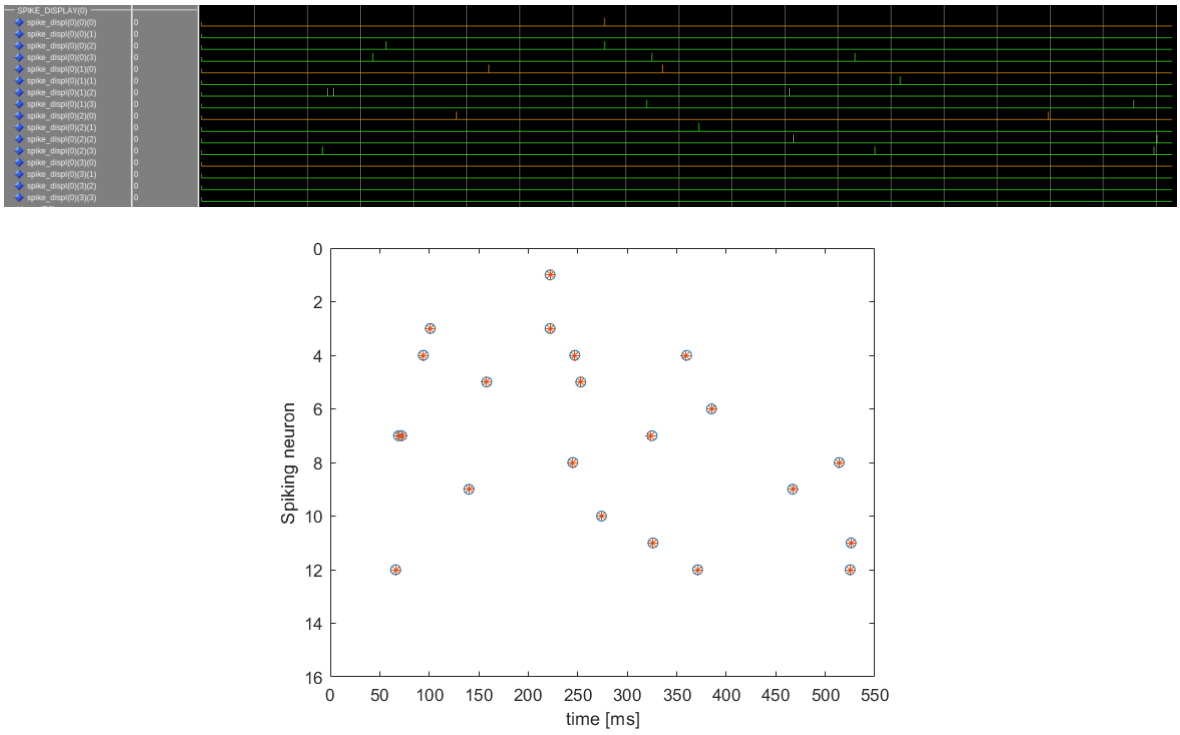

Figure 2.11: a)Spiking pattern obtained simulating a 16 neurons fully connected network in Questasim b) MatLAB simulation of the same Network with floating point arithmetic (o) and fixed point arithmetic (*).

Fig. 2.12 shows the analog representation of the membrane potential of all the 16 neurons. As it can be noticed, noise continuously affects membrane potential and each peak corresponds to a spike.

Figure 2.12: Membrane potential of all the 16 neurons of the Network

2.7 Simulating different types of Neuron

As discussed in 1.1.5 and shown in 1.5, Izhikevich model aims to imitate different types of cortical neurons according to well chosen a, b, c and d parameters. In 1.5 in particular, the characteristic membrane behaviour of three types of excitatory neurons called RS, IB and CH, and two kind of inhibitory called FS and LTS are presented. The aim of this section is to simulate in MatLAB and HEENS some of these known neurons using the parameters already indicated in 1.1.5, in order to demonstrate the capability of our model in reproducing some of their characteristics, such as spike pattern and membrane recovery interval. In particular, in Fig.2.14 three kinds of neurons, two excitatory and one inhibitory, are simulated individually in MatLAB. Each simulation is driven by a constant $10mV$ input and the following neural parameters:

- **CH**: $a = 0.02$ $b = 0.2$ $c = -50$ $d = 2$
- **RS**: $a = 0.02$ $b = 0.2$ $c = -55$ $d = 4$
- **FS**: $a = 0.02$ $b = 0.2$ $c = -65$ $d = 8$

In "Appendix L" the used assembly program is shown. This is actually identical to the more generic previously presented, with the only exceptions that here all the parameters are defined in the assembly program and stored as constants in IMEM and the noise sub-routine simply add the constant $10mV$ value. The fig.2.13 shows the membrane potential and the spike pattern of two neurons for each type, obtained in Questasim. The two neurons differ only for the initial membrane values, they are, in fact, basically shifted in time, but they behave in the same way. Note that at the "e" of Fig.2.14 the blue and the orange curves are perfectly superimposed and so the reader may be misled not seeing the blue curve. The coincidence of the two results can be verified looking at the spiking pattern in "f".

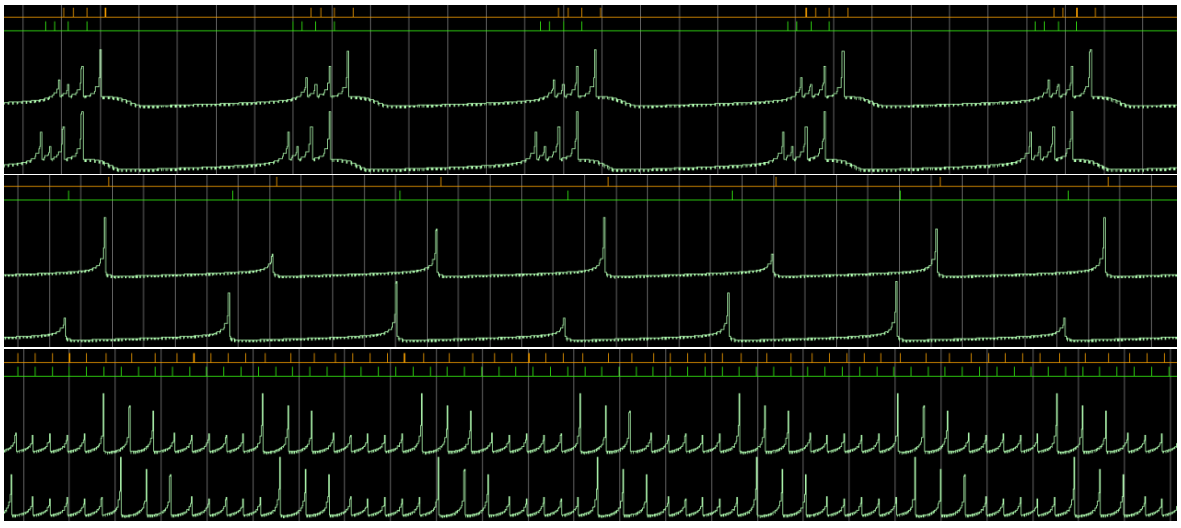

Figure 2.13: Membrane potential behaviour and the spike pattern obtained in HEENS for: a) CH neuron; b) RS neuron; c) FS neuron.

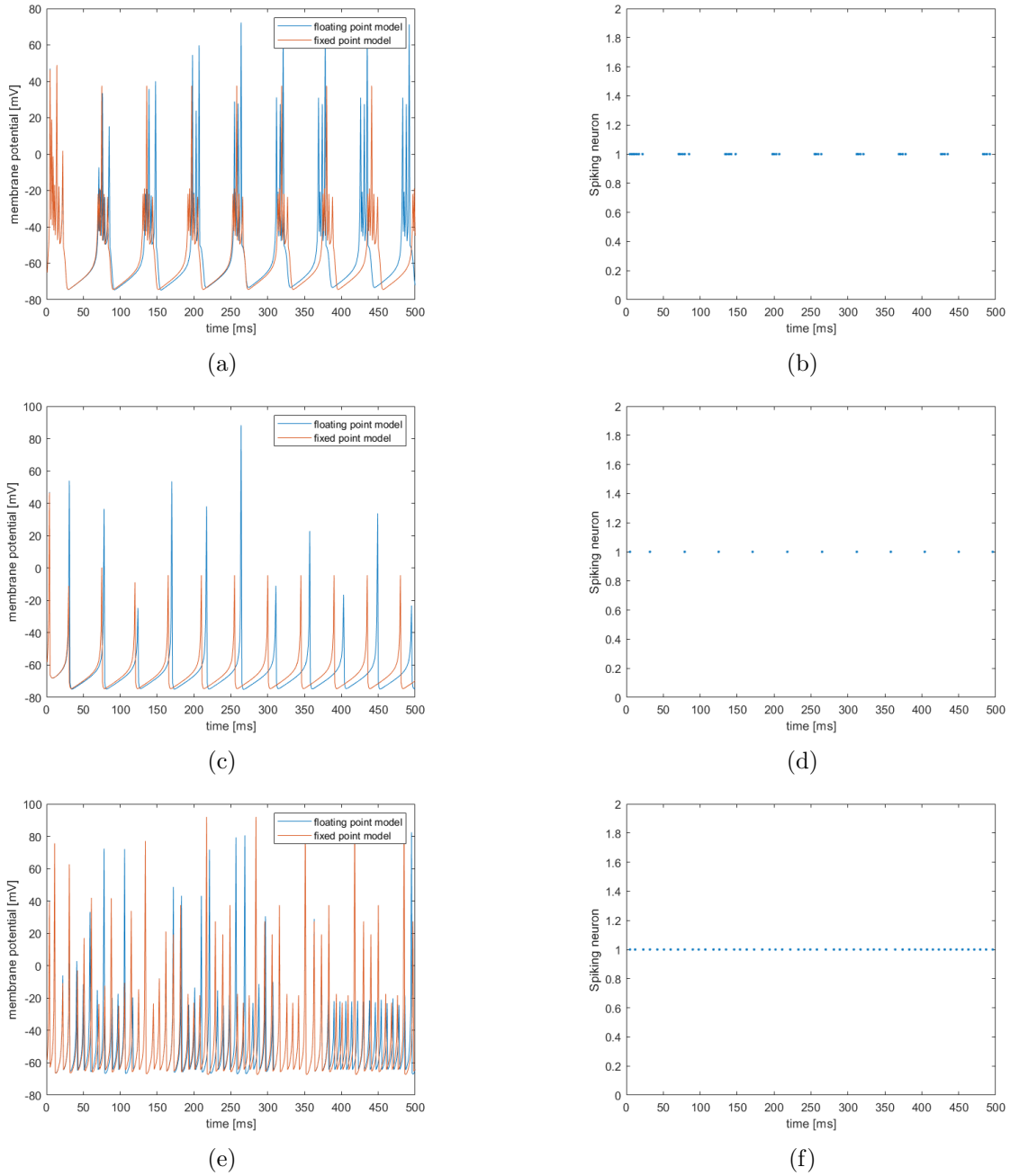

Figure 2.14: Membrane potential behaviour and the spike pattern obtained in MatLAB for: a,b) CH neuron; c,d) RS neuron; e,f) FS neuron. In blue, as usual, is shown the floating point model, in orange the fixed point model. In the right column, for sake of clarity, is shown only the outcome of the floating point.

CHAPTER 3

STDP algorithm mapping on HEENS

3.1 STDP algorithm

As discussed in 1.1.6, "Spike timing dependent plasticity" is a property of synapse connections consisting in a variation of their strength due to the relative time lapse between presynaptic and postsynaptic neurons' spikes. As already said, in nature, many types of relations which can impact in different ways on the strength of a link have been found, which in different ways keep account of the spikes in order to introduce some mechanisms that cause those long period depression and potentiation phenomena already described. The relation chosen in our model is called Anti-symmetric Hebbian rule and it is described by the Fig. 3.1.
This kind of relation can be easily visualized considering that positive time values relate to a positive value of the quantity: $t_{postsynspike} - t_{presynspike}$, and thus, for example, $40ms$ means that the postsynaptic spike happened $40ms$ after the pre-synaptic one and vice versa the opposite value. According to the previous assumption, in the Antisymmetric Hebbian relation, a link is as more reinforced as more the presynaptic neuron spikes immediately before the postsynaptic, or in other words, as more the pre-spike determines the post-spike. However, the figure 3.1 shows that for very small value of time there is not significant change in the synapse. This keeps in consideration, if present, a possible refractory, and thus "insensible", period immediately after spike. Once qualitatively understood the relation occurring between spikes, must be identified an analytical rule and then obtain from it an algorithmic version able to be inserted in HEENS and to be fully compatible with the Izhikevich model. The Izhikevich model, indeed, features some constant synapse weights which are added together by the receptive neuron when there are spikes. Can be thus simply written for each synapse:

$$W_{ji} = S_j * P_{ji} \tag{3.1}$$

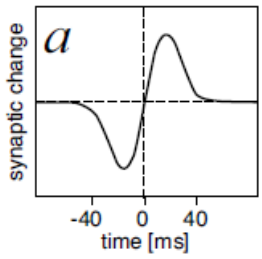

Figure 3.1: Anti-symmetric Hebbian rule. modified from [15]

Where "W" is the synaptic weight to be added and coming from the neuron "j", "S" is a binary variable equal to '1' if the neuron fired, "P" is the intrinsic weight of the synapse. Inserting STDP makes "W" changing according to time and pre/post-synaptic spikes and thus consider a new term called L(j) which describes this variation. The relation thus became:

$$Wji(t) = S_j * P_{ji} * L_{ij}(t, S_i, S_j) \tag{3.2}$$

which states that the weight of the link which connect the presynaptic neuron "j" and the postsynaptic one "i" differs from zero only when "j" spikes and change its original value "P" according to the parameter L which is function of times and spikes.
Javier Iglesias in [10] proposes an STDP model in which "W" depends on an integer variable called "A" which change its value when a real values variable exceeds some threshold. In our model we don't use any threshold or quantized integer value, but the same real values variable introduced by Iglesias becomes our "L". This means basically to admit a continuous variation of "W" with "L". The equation which describe this variation is:

$$L_{ji}(t+1) = L_{ji} * K_{act} + S_i(t) * M_j(t) - S_j(t) * M_i(t). \tag{3.3}$$

Here, "L" changes at each algorithmic step according to a decay coefficient K_{act} and according to some quantities called M_i and M_j added if a post or pre-spike happens. The coefficient $M(t)$ may be viewed as the "memory" of the latest inter-spike interval, and assumes the form: (in this case written for i, but equal for j)

$$M_i(t+1) = S_i(t) * M_{max} + (1 - S_i(t)) * M_i(t) * K_{syn} \tag{3.4}$$

M_i assumes a reset maximum value M_{max} after each spike, decreasing otherwise according to a decay constant called K_{syn}. Summarizing, with the generation of a postsynaptic spike (i.e. when Si = 1), the value Lji receives an increment which is a decreasing function of the elapsed time from the previous presynaptic spike. On the other hand, when a spike arrives at the synapse, the variable "L" decreases as function of the elapsed time from the previous postsynaptic spike. The trends of "M" coefficients and the resulting variations of "L" can be better visualized with the Fig.3.2. At each spike, "M" coefficients reset and "L" is decreased or increased according to "M" values. If no one spike occurs, "L" decreases slowly according to the K_{act} coefficient.

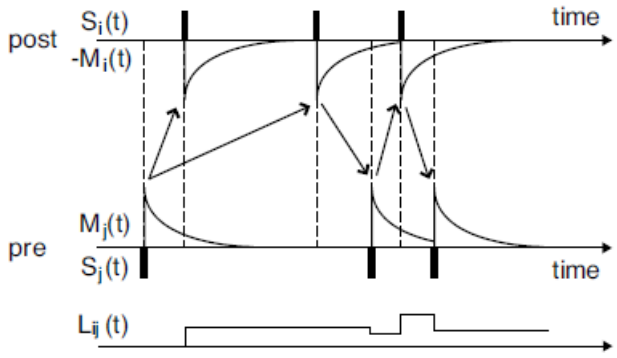

Figure 3.2: "M" and "L" trends when spikes occurs. At each pre(post)synaptic spike, $M_{i(j)}$ reset its value and "L" receives a decrement(increment) according to the value of M at the spike instant. If no spikes occurs M coefficients decay with K_{syn} and "L" decay very slowly with K_{act} (not shown). [10]

Iglesias in [10] describes "K" coefficients with exponentials:

$$K_{syn} = e^{-t/\tau_{syn}} \qquad K_{act} = e^{-t/\tau_{act}} \tag{3.5}$$

Our choice is to assume "K" constants equal to all neurons and, according to values considered reasonable by the literature, equal to $\tau_{syn} = 40\ ms$ and $\tau_{act} = 11000\ ms$. Moreover, for an algorithmic purpose it is convenient to approximate the exponential behaviour with a linear one (i.e. Taylor series approach), writing:

$$\begin{gathered} y = e^{-t/\tau} \\ \frac{de^{-t/\tau}}{dt} = -\frac{-1}{\tau} * e^{-t/\tau} \\ \frac{dy}{dt} = \frac{-y}{\tau} \\ \Delta y \simeq \frac{-y * \Delta t}{\tau} \\ y(n+1) - y(n) \simeq \frac{-y(n)}{\tau} \\ y(n+1) \simeq y(n)(1 - \frac{1}{\tau}) = \alpha y(n) \\ with\ \alpha = \frac{\tau - 1}{\tau} \end{gathered}$$

In our case, the constants are:

$$K'_{syn} = \tau_{syn} - 1/\tau_{syn} = 39/40 \qquad K'_{act} = 10999/11000. \tag{3.6}$$

The relations 3.3 and 3.2, considering t and $t+1$ an arbitrary algorithmic step and its next, thus simply become:

$$L_{ji}(t+1) = L_{ji} * K'_{act} + S_i(t) * M_j(t) - S_j(t) * M_i(t). \tag{3.7}$$

$$M_{i(j)}(t+1) = S_{i(j)}(t) * M_{max} + (1 - S_i(t)) * M_{i(j)}(t) * K'_{syn} \tag{3.8}$$

It worth to be stress that the sign "-" in the equation 3.7 realize the anti-symmetric rule.
It is clear that the value of M_{max}, gives a scale of how much STDP is effective, of how much at most L changes its value for each spike. Moreover "L" must describe both the potentiation and depression phenomena according to its value, i.e. according to a value bigger or minor than 1. Considering the mapping of this algorithm in HEENS, due to Fixed point arithmetic is naturally able only to deal with integer number, our design choice is to give to "L" an initial value of $L_{init} = 20$ and to M a maximum value of $M_{max} = 2$. Re-writing 3.2 as:

$$Wji(t) = \frac{S_j(t) * P_{ji} * L_{ij}(t, S_i, S_j)}{20} \tag{3.9}$$

and thus:

$$Wji_{init}(t) = \frac{S_j(t) * P_{ji} * L_{ij,init}}{20} = \frac{S_j(t) * P_{ji} * 20}{20} = S_j(t) * P_{ji}; \tag{3.10}$$

and re-writing 3.4 as:

$$M_i(t+1) = S_i(t) * 2 + (1 - S_i(t)) * M_i(t) * K'_{syn} \tag{3.11}$$

The normalized form in 3.9 has been chosen because it is easy and user friendly in the development phase, precise and efficient because it needs no rounding if not at the final normalizing division.

Another equally reasonable choice is to set $Mmax = 1$ and thus to have basically a less effective STDP. The details about the introduction of the STDP algorithm in HEENS are treated in future sections. Before, as usual, the model is inserted and simulated in MatLAB. This is the topic of the next section.

3.2 MatLAB simulation

In APPENDIX H is described the MatLAB program simulating a 16 neurons SNN supporting the Izhikevich neural model and STDP. The program has been already presented in Chapter 3, but ignoring the sections which start at the lines "64" and "74" which describe STDP routine for the floating point and the fixed-point models of the membrane potential. Considering for example the piece of code relative to the floating point part (line "64"), the two equations already presented in 3.4 and 3.5 are executed for all neurons through a for cycle and then Sf which represents our $P_{ji} * L_{ji}$ product is updated. Note that *alpha* and *beta* are respectively the decay constants K'_{act} and K'_{syn} and the fixed-point section executes the same operation.
The Fig.3.3 plot on the same graph the spike pattern of the floating point model (o) and of the fixed point model (*) both implementing STDP for $30.000ms$ and thus 30.000 cycles, as can be seen the two models behave in a similar way but moving forward with time there are some errors. This can reassure us about the reliability of our model but at the same time it reveals that our fixed-point model is not able to perfectly replicate the ideal behaviour when very small changes appear as in the STDP routine.

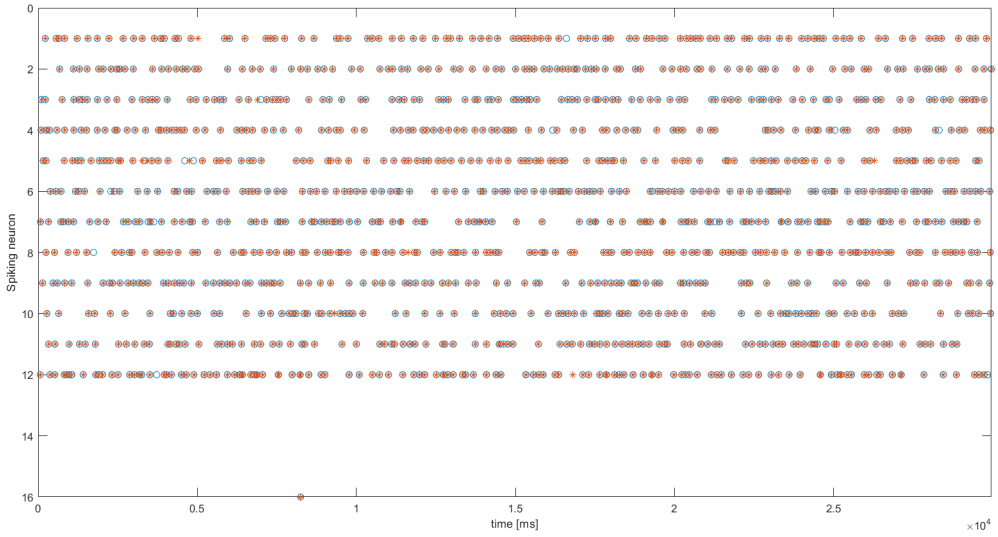

Figure 3.3: $30.000ms$ simulation of a 16-neurons SNN supporting STDP. (o) is the floating-point model of the network, (*) the HEENS-like fixed point model.

It can be furthermore interesting to compare two networks different only by whether to implement STDP and evaluate their different evolution in time. This kind of comparison is done in Fig.3.4,3.5,3.6 for different size of network and simulation time. As can be noticed, despite in a first moment there are not relevant differences between the models, moving forward with time the two networks start to behave in a different way, some spikes are shifted in time or not present, new spikes appear. However, it is still very complicated to recognise the STDP features clearly from these kinds of simulations,

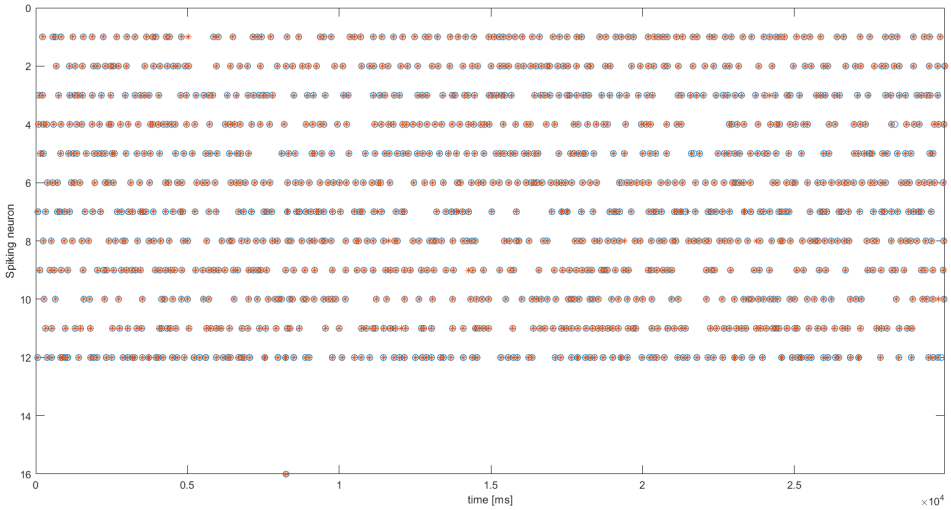

Figure 3.4: 30.000ms simulation of a 16-neuron SNN, (o) is the spike symbol for the network supporting STDP, (*) is the spike symbol for the network not supporting STDP.

and thus, in the following, some examples of network will be presented in order to stress, and clearly visualize, the STDP features.
In order to measure the growth or degrowth rate of the synapse weights with respect their initial value, it can be considered the matrix S of initial synapse weights and the matrix Sii of the final weights and calculate.

$$Svar = ((Sii - S)./S) * 100 \tag{3.12}$$

Where $Svar$ stays for "variation of S" and the symbol "." means that the operation is done on every element of Sii and S in the same position. Taking the maximum and minimum of $Svar$, we found the most grown and the most degrown synapse in percentage. Considering for example the 16-neurons network, it results to be:

$$Svar_{MAX} = -48\% \qquad Svar_{MIN} = -100\% \tag{3.13}$$

after 30 ms, thus, all the synapses are decreased, but in different measures.

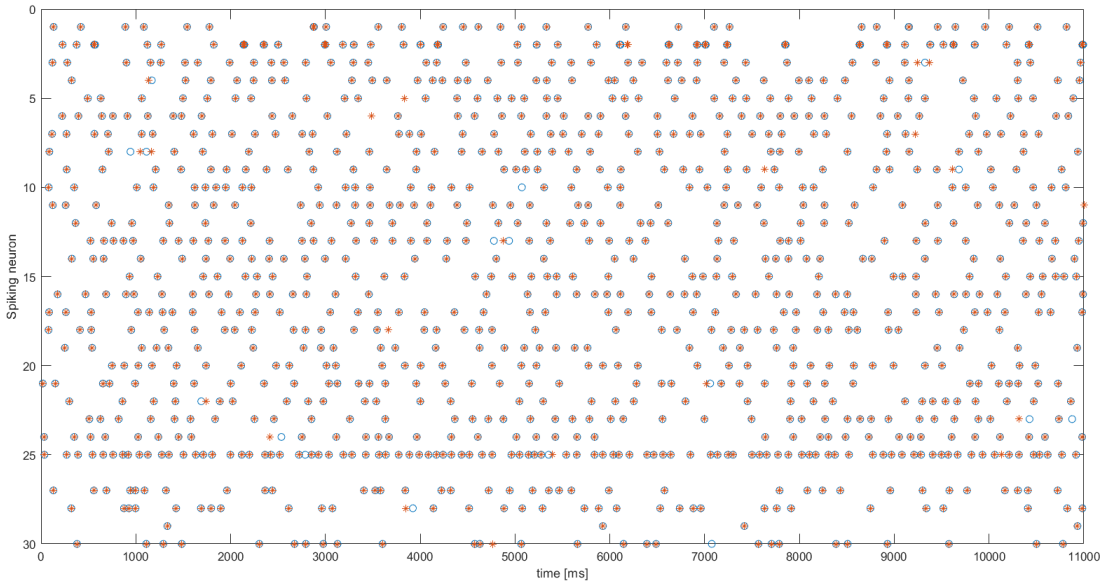

Figure 3.5: 11.000ms simulation of a 30-neuron SNN, (o) is the spike symbol for the network supporting STDP, (*) is the spike symbol for the network not supporting STDP.

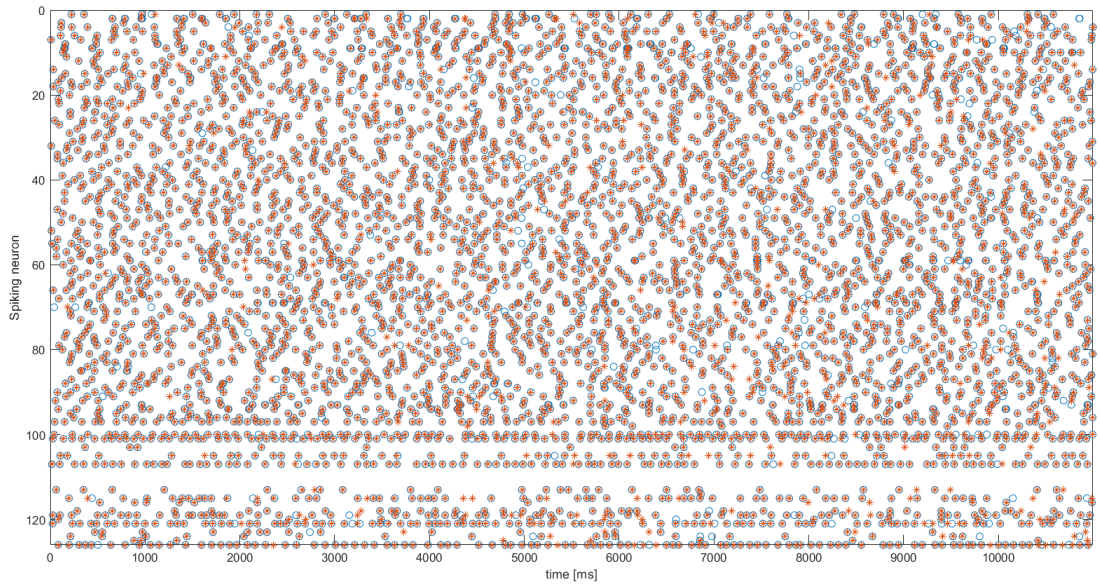

Figure 3.6: 11.000ms simulation of a 128-neuron SNN, (o) is the spike symbol for the network supporting STDP, (*) is the spike symbol for the network not supporting STDP.

3.3 STDP in HEENS architecture

In this Chapter have already been presented the equations describing the STDP model first adopted and then customized for this project. The initial value of the STDP parameters "L" and "M" have been defined as $L_{init} = 20$, $M_{init} = M_{max} = 2$. The decay time constants chosen are, instead, $\tau_{act} = 11000ms$ for the decay of "L" parameter and $\tau_{act} = 40ms$ for what concerns "M" parameter. As said, Synapse strength decays according to an exponential trend, in our algorithm, instead, this kind of behaviour is approximate piecewise linear with the coefficients $K'_{act} = \frac{11000-1}{11000}$ and $K'_{syn} = \frac{40-1}{40}$. This data must be encoded in HEENS to allow good precision in calculations and algorithmic efficiency. Thanks to the introduction of the unsigned multiplication, the two rational decay constants can be stored in HEENS as:

$$K'_{act} = \frac{11000-1}{11000} * 2^{16} = (0.99991 * s^{16})_{10} = (65530.04)_{10} = (FFFA)_{hex} \tag{3.14}$$

$$K'_{syn} = \frac{40-1}{40} * 2^{16} = (0.975 * s^{16})_{10} = (63897.6)_{10} = (F999)_{hex} \tag{3.15}$$

It is worth to stress again that these coefficients must be seen as unsigned numbers (otherwise they would appear as negative numbers) and thus managed with unsigned arithmetic. Another option would have been to store half their values, this would allow the use of signed arithmetic, but would have meant less precision.
At each step, this constant must be eventually multiplied with the synapse variables "L" and "M". To appreciate the variation deriving, the "L" and "M" parameters must be stored with a compatible precision and thus in a format containing fractional digits enough to be sensitive to the operation. In particular, the main synaptic parameters are encoded as follows:

- **L** is an unsigned number, its initial value is 20 and it must be able at least to duplicate its value. There is therefore no need for a sign bit and its integer part can be reasonably encoded in 6 bits. When multiplied by the coefficient K'_{act} which can be approximated as $K'_{act} = 0.9999$, it needs theoretically one millionth of precision. Our choice is to store L in an entire memory row (32bits) and align, in the most significant word, its fractional digits to those of "M" parameters. This means provide it basically 26 fractional digits, since the 10 still available using a single 16 bits word would not be enough.

- **M** is an unsigned number, its initial and maximum value is 2. There is therefore no need for a sign bit and its integer part can be reasonably encoded in 2 bits. When multiplied by the coefficient K'_{syn} which is $K'_{act} = 0.975$, it needs theoretically one thousandth of precision, which corresponds to 10 decimal digits ($2^{-10} = 1/1024$). Thus, the bits needed to encode the value are actually 12, we can therefore store the value in a 16 bits word and encode in the same word also, for example, the spike flag bit. This will be shown in the next section.

- **P** represents the "static" synaptic strength. It is a signed number and its value is biologically hypothesized to be between $-1mV$ and $1mV$. It must be thus encoded as a number between $-100 \leq P \leq 100$ and thus need at least 7 bits. However, for sake of simplicity and flexibility the value is stored as 16 bits signed integer.

It is worth to stress that supporting STDP doesn't need any substantial hardware modification but only a small modification in the selection signal of the MUX downstream of the *lcl spike register*.
In fact, the STDP subroutine performs two accesses in memory, but just the first one is a $LOADSP$ instruction which loads the Sj pre-synaptic spike flag into the first position of the accumulator (See

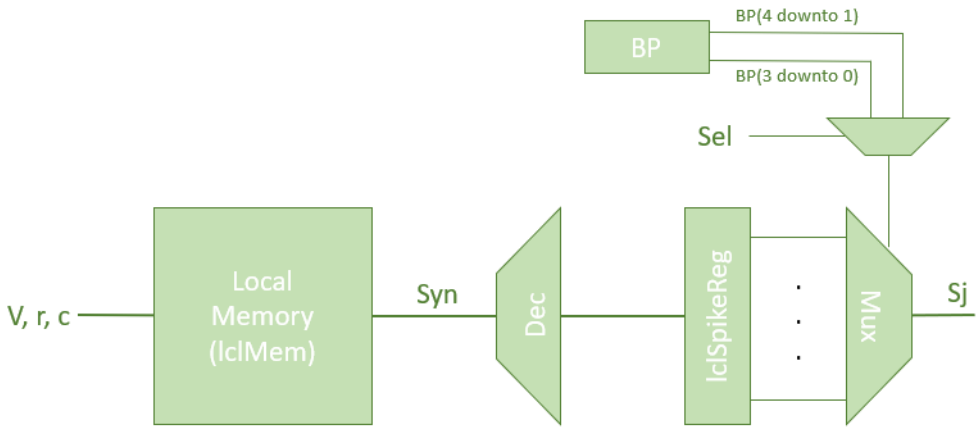

Figure 3.7: Spike decoding chain. In the top of the picture, the part of "BP" which drives the "MUX" change from BP(3 downto 0) (NO STDP) to BP(4 downto 1) (STDP)

APPENDRIX A). Therefore, due to, as said in **Chapter 1**, the selection signal of the MUX is the BP, it must select a different value of the *lcl spike register* every multiple of two of the address in memory. In order to do this, the selection signal simply changes from $BP(3downto0)$ to $BP(4downto1)$. In order to not re-synthesize the entire architecture for this small change, moreover, the two portions of the signal can be simply multiplexed by a selection signal external at the PE. The Fig.3.7 shows the structure (modified from an idea of [19]).
Next section details the organization in SNMEM of all the parameters already described.

3.4 SNMEM organization for SDTP

This section aims to describe how data needed to execute the STDP algorithm are stored and managed in HEENS. This section is in addition to and expand what has already been described in 2.4, which is recommended to read first. With the introduction of STDP, the content of the SNRAM for each neuron of a virtual layer becomes:

Memory row	16 MSbs	16 LSbs
$NEUaddr0$	membrane potential "v" $Q16.0$	recovery potential "u" $Q16.0$
$NEUaddr0+1$	parameter "b" $Q16.0$	parameter "d" $Q16.0$
$NEUaddr0+2$	parameter "a" $Q16.0$	parameter "c" $Q16.0$
$NEUaddr0+3$	STDP flag*	M_i unsigned Q6.10
...	...	...
$Noiseaddr0$	16-bit seed	16-bit seed
$Noiseaddr0+1$	16 bit seed	16 bit seed

Table 3.1: Neural parameters stored in memory

Where the format unsigned $Q6.10$ indicates an unsigned integer number with 6 integer digits and 10 decimal digits. And the initial value of M_i is $4*2^{10}$, stored for algorithm efficiency as double its value, to be right shifted during the algorithm obtaining the value of $2*2^{10}$, with thus 10 digits of decimal precision.
*The STDP flag is present only in the version of the program which supports the presence of STDP

at neuron level (more details in the next section).
In the algorithm supporting STDP, synaptic parameters are allocated into two memory rows of SNRAM instead that in a single row. Allocation in SNMEM of the synaptic parameters of each neuron "i" linked to the neuron "j" ($0 \leq j \leq 15$ where $J = 0$ stays for the neuron itself) becomes:

Memory row	16 MSBs	16 LSBs
SYN addr0	weight $P_{0,i}$ &S_p*	M_0 unsigned $Q5.10$ & S_0
SYN addr0 +1	$L_{0,i}$ unsigned $Q6.10$	$L_{0,i}$ LS 16 bits
SYN addr0 +2	weight $P_{1,i}$	M_1 unsigned $Q5.10$ & S_1
SYN addr0 +3	$L_{1,i}$ unsigned $Q6.10$	$L_{1,i}$ LS 16 bits
SYN addr0 +4	weight $P_{2,i}$	M_2 unsigned $Q5.10$ & S_2
SYN addr0 +5	$L_{2,i}$ unsigned $Q6.10$	$L_{2,i}$ LS 16 bits
...	...	...
SYN addr0 +30	weight $P_{15,i}$	M_{15} unsigned $Q5.10$ & S_{15}
SYN addr0 +31	$L_{15,i}$ unsigned $Q6.10$	$L_{15,i}$ unsigned $Q0.16$

Table 3.2: synapses stored in memory

*The spike flag S_p is present only in the version of the program which supports the plasticity at connection level.
As reported in APPENDIX D, initial synaptic parameters are written in memory by *netlist.lst* and each line of the file is of the form:

```
#|id|v|r|c| |id|v|r|c| synN MSw1, LSw1, MSw2, LSw2
```

and for example the first line is:

```
0 0 0 0 0 0 0 0 0 9 2048 20480 0
```

where $P = 9$, $M_{j,init} = M_{max} = 2 * 2^{10}$ $L_{ji,init} = 20 * 2^{10}$.

3.5 Assembly code

This section refers to the common features of the assembly programs in APPENDIX N, APPENDIX P, APPENDIX Q. These three programs describe the neural routine of networks supporting STDP at different level:

- APPENDIX N shows a program when STDP is supported by default by the entire network.
- APPENDIX P shows a program when STDP is supported at a neuron level. This means that according to the $STDP\ flag$, presented in the previous section and stored, for each neuron, in the SNMEM as shown in 3.1, the STDP is applied to all the pre-syn connection of the neuron.
- APPENDIX Q shows a program when STDP is supported at connection level. This means that according to the flag S_p presented in the previous section, stored for each connection as LSB of the "P" parameter and shown in 3.2, the STDP is applied to the single connection.

Based on the above, in the first case the program executes automatically STDP, in the second it tests for each neuron, in the third case it tests for each synapse according to S_p and thus the efficiency of the algorithm is reduced.

All three programs, however, are like the one previously detailed in chapter 3 and shown in the APPENDIX C, with the difference that the STDP equations, if supported, must be executed for each synapse according to the pre-synaptic activity. This basically introduces changes in each sub-routine, because more parameters must be handled, but most of all are different the operations done in $STDP\_SYNAPSE\_CALC$, the sub-routine now in charge of the synaptic operations. During it, both the rows of the SNMEM storing "P", "M" and "L" are updated, together with the usual operations of routing operated in Synaptic Local Memory. An additional sub routine called Mi updates the M parameter of the post-synaptic neuron (i.e. the neuron which is operating the routine) if in turn the neuron spikes. The sections of the assembly program are:

```
; Network definitions

define virtual_layers 0 ; From 0 up to 7
define gsynapses 2 ; Up to 32 global synapses
define n_step 1;

.DATA

; Virtual layers
V0 = "0000000F" ; Number of assigned synapses (s-1) to the main layer
[...]
V7 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 7
VLAYERS="00000000" ; Number of virtual layers (n-1).

; Membrane potential parameters common to all neurons

;multiple of 10uV
VTHRES="FFFFF63C"   ; Threshold voltage -25 mV
N70="00001B58"; 70mV,
N0002="000068DC";
N001="00000028F"; 0.01, The noise generated by LFSR is 100 time bigger than necessary.
REST_POT="FFFFE69C"; -6500
N5="00000005";
```

```
; Neural and Synaptic RAM addresses
SYN_ADDR0="00000000"    ; First address of Synaptic parameters in SNRAM for V = 0.
SYN_ADDR1="00000020"    ; First address of Synaptic parameters in SNRAM for V = 1.
SYN_ADDR2="00000040"    ; First address of Synaptic parameters in SNRAM for V = 2.
[...]
SYN_ADDR7="000000E0"    ; First address of Synaptic parameters in SNRAM for V = 7.
GSYN_ADDR="00000090"    ; First address of Global Synaptic parameters in SNRAM.
NEU_ADDR0="000003E3"    ; First address of Neural parameters in SNRAM (995) for V = 0.
NEU_ADDR1="000003E7"    ; First address of Neural parameters in SNRAM (999) for V = 1.
NEU_ADDR2="000003EB"    ; First address of Neural parameters in SNRAM () for V = 2.
[...]
NEU_ADDR7="000003FF"    ; First address of Neural parameters in SNRAM () for V = 7.

SEED_ADDRL = "000003FD" ; Address of noise seed in SNRAM
SEED_ADDRH = "000003FE";Address of noise seed in SNRAM
PEID   = "000003FF" ; Address of  PE Identifier number

; STDP constants
K_syn= "0000F99A"; (40ms-1ms/40)
K_act= "0000FFFA"; (11000-1/11000)
N025= "00000666"; 1/40=0.025 to be multiplied to sum of weights
```

Which differs from the previous version because the space in SNMEM for synaptic parameters is double, and the plasticity constants must be declared.

```
.CODE
;
GOTO MAIN ; Jump to main program
;
; *************************** PROCEDURES BEGIN ****************************
;
.RANDOM_INI
[..]
RET
.LOAD_NEURON
[..]
RET
.DETECT_SPIKE
[..]
RET
.STDP_SYNAPSE_CALC
[..]
RET
.GAUSS_NOISE
[..]
RET
.MEMBRANE_POTENTIAL
[..]
RET
.Mi
RET
.RECOVERY_UPDATE
[..]
RET
.SUM_NOISE_AND_Ws
[..]
RET
.STORE_NEURON
[..]
```

```
RET
; ************************** PROCEDURES END ****************************
```

Every subroutine is slightly different from the previous version. Furthermore "Mi" subroutine is added.

```
.MAIN
;

; Virtual operation init
LAYERV virtual_layers ; Init sequencer vlayers. It is 0 for non-virtual operation
LDALL ACC, VLAYERS ; Load defined virtual layers to PE array
SPMOV 0 ; VIRT <= ACC

; Initial instructions
GOSUB RANDOM_INIT  ; For noise initialization

.EXEC_LOOP ; Execution loop

 LOOP virtual_layers     ;Neuron loop for virtual operation
     NOP     ;to prevent pipeline error
     GOSUB LOAD_NEURON
     GOSUB DETECT_SPIKE
  SWAPS R3     ;SYNAPSE Sum will be store IN R3
  MOVA R5
  SHRN 5
  SHRN 5
  MOVR R5
     READMPV SYN_ADDR0
     LOADBP
     LOOPV V0  ;synaptic loop. Reads number of current-layer synapses
  NOP          ;to prevent pipeline error
         GOSUB STDP_SYNAPSE_CALC
     ENDL
  SWAPS R3 ;SYNAPSE sum to SW3
  GOSUB Mi
  GOSUB GAUSS_NOISE
     LOOP n_step
      NOP
  GOSUB MEMBRANE_POTENTIAL  ;Calculate membrane potential
      GOSUB SUM_NOISE_AND_Ws
 ENDL
 GOSUB RECOVERY_UPDATE
 GOSUB STORE_NEURON
 RST R3;
 MOVSR R3;
 INCV
 ENDL
.FINISH
NOP         ;Empty pipeline wait NOPs
NOP
NOP
SPKDIS        ;Distribute spikes
GOTO EXEC_LOOP       ;Execution loop
```

Similar to the previous one for what concerns order and tasks.

STDP routine performs two accesses in memory for each synapse, for a total number of 32 accesses in the case of a fully connected 16 neurons network, and for each synapse it tests if there is a spike

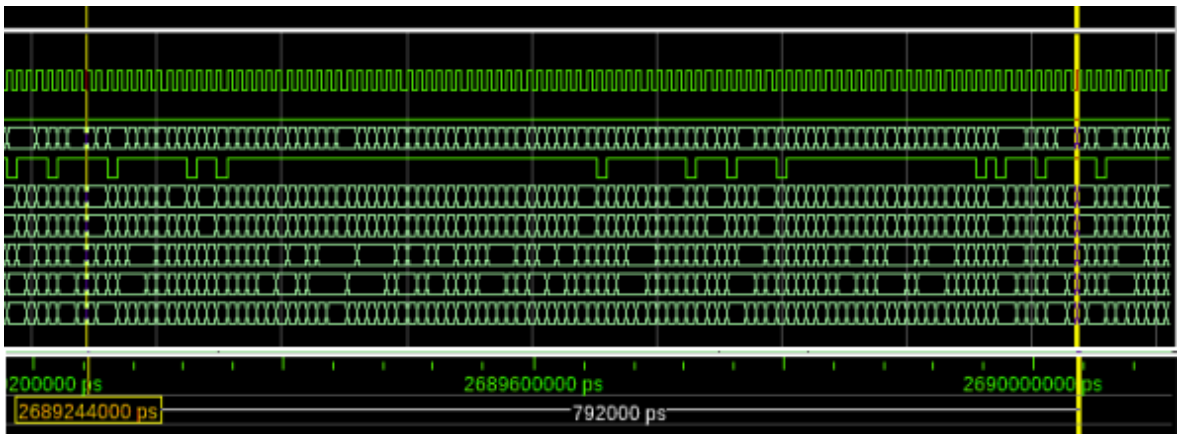

Figure 3.8: Synaptic subroutine duration

incoming and eventually sum up it to an accumulator. This subroutine is for sure the more time-consuming task in the emulation cycle.
The emulation cycle duration is an important factor, both to correctly size the length of the simulation, both to be sure that biological performances are fulfil. It can be determined experimentally simply looking at the wave window and detecting the time lapse before the algorithm is repeated. For the network entirely supporting STDP, the cycle duration turns out to be approximately equal to $15,54us$. Assuming in a first approximation that the only part of the program strictly depending from the number of neurons is the *synaptic_calculation* subroutine, which is repeated as many times as many are the neurons, it is interesting consider the cycle length as a function of the number of neurons as follows:

$$T_C(N) = N * t_s + t_{const} = 15.54us \tag{3.16}$$

where T_C is the emulation cycle duration, N is the number of neurons, t_s is the duration of the *synaptic_calculation* subroutine, t_{const} is the duration of the part of code which does not depend on the number of neurons in the network. In our simulation, $t_s \approx 0.8us$ as shown in 3.8.

Using this formula, if it is requested to know the cycle length of a 16 physical neurons network using all the 8 virtual layers, it can be considered as a 128 neurons fully connected network and it can be written:

$$T_C(16 * 8) = 16 * 8 * t_s + t_{const} \tag{3.17}$$

or writing according to the result 3.16

$$T_C(16 * 8) = 16 * 8 * t_s + t_{const} = 15.54 - 16 * 0.8 + 128 * (0.8) = 105.14us \tag{3.18}$$

Which provides a worst-case prediction.
As can be noticed, T_C in this equation grows linearly with the increase of the size of the network. It is clear therefore that is very important have an efficient *synaptic_calculation* algorithm and this will be again treated in a following section.

3.6 Simulation in QuestaSim

In this chapter the STDP formulas and the implementation of the algorithm in HEENS have been presented. It has been shown, moreover, that in a wide fully connected network, as that simulated in 3.2, it is really hard to identify the outcomes of plasticity and predict the evolution of the synaptic connections, and therefore of the spiking pattern, in an intuitive way. For these reasons, this section aims to show the actual functioning of the developed algorithm, and therefore the main features of the STDP rule, through two example networks and the graphical representation of the neural variables offered by the "Wave" tool of Questasim. The two networks to be analysed supports STDP at connection level. The first Network is that of 3.9. According to the structure of the *Netlist.lst* file already described; it is defined as:

```
#|id|v|r|c| |id|v|r|c| synN MSw1, LSw1, MSw2, LSw2
0 0 0 0 0 0 0 0 1 1000 2048 20480 0
0 0 0 0 0 0 0 1 1 5000 2048 20480 0
0 0 0 2 0 0 0 2 1 1000 2048 20480 0
0 0 0 2 0 0 0 3 1 5001 2048 20480 0
0 0 0 1 0 0 1 0 1 5000 2048 20480 0
0 0 0 3 0 0 1 0 2 2000 2048 20480 0
```

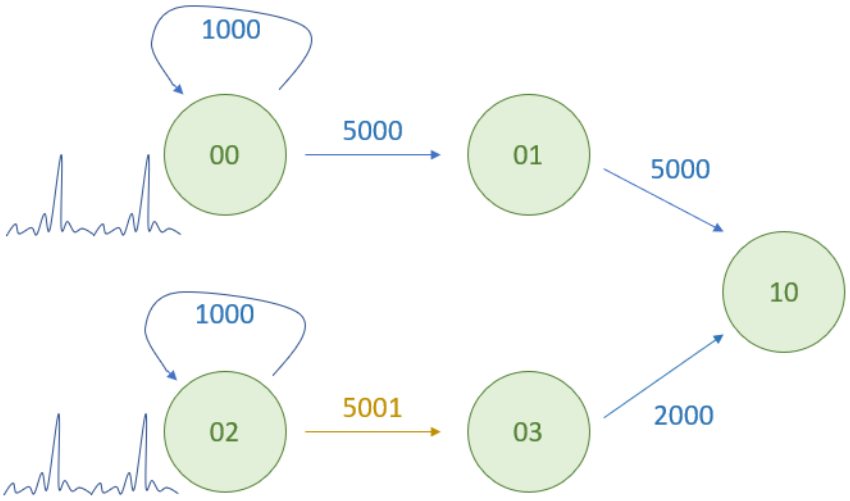

Figure 3.9: First example network, the orange connection support STDP property, which is encoded in the "1" value of the LSB

Our interest is to underline the modification introduced by plasticity and, with this aim, the couples of neurons "00"-"01" and "02"-"03" are designed exactly identical (they have the same neural parameters and the same noise seeds), with the only exception that the link between "02" and "03" supports plasticity and thus it is expected to present a different evolution along time. The neurons "00" and "02" are sources of spikes realized imposing a reset membrane potential near to the threshold and a feedback self-spike big enough to overcame it and generate a new spike. The sources are, however, not continuous due to at each spike emitted the membrane recovery variable "u" is increased of the amount described in the parameter "d" and, according to 3.2, the recovery variable "u" is at each step subtracted to the membrane potential "v". Thus, at a certain moment, the "u" variable will be

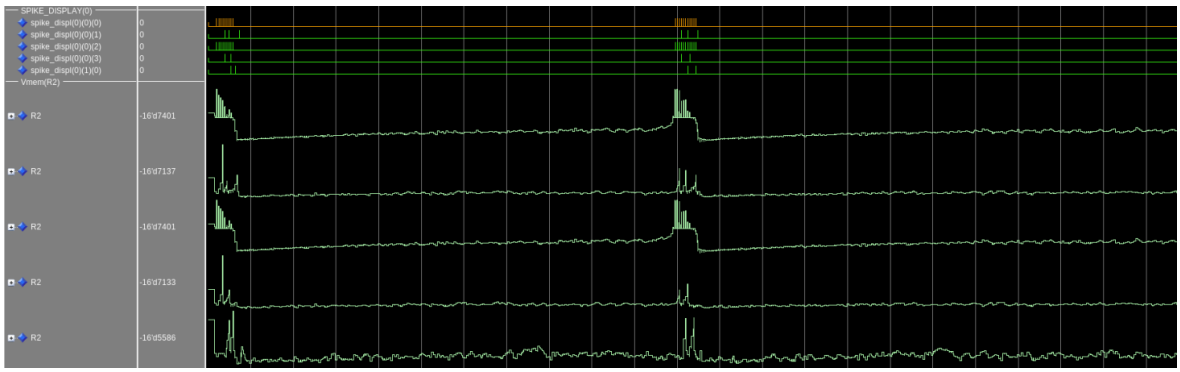

Figure 3.10: Questasim simulation of the first example network. In the top the spike pattern, in the bottom the analog shape of the membrane potential.

big enough to prevent a new spike.
Fig.3.10 shows the network simulated in Questasim. As can be seen, in absence of plasticity (couple "00"-"01") the behaviour of the system is almost periodical. After the first burst of spikes, the membrane potential of the neuron "00" increases slowly according to the differential equation in 3.2, with only the background noise as external stimulus, then the burst repeats almost identical.
Even if in a fist moment the behaviour of the couple "02"-"03" is of the same kind, then it deviates from that of other couple for action of plasticity. Indeed, the second spike of the neuron "03" is shifted and this happens also in the second burst with in turn also a different behaviour of the neuron "10".

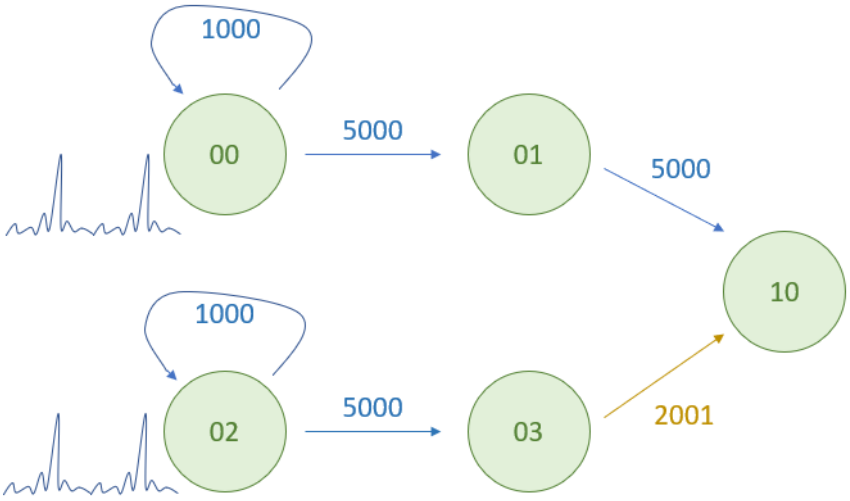

Figure 3.11: Second example network, the orange connection support STDP property, which is encoded in the "1" value of the LSB

We can analyse the same kind of behaviour under another perspective with the network in Fig. 3.11. In this case, the two couples "00"-"01" and "02"-"03" are totally identical, while the connection "03"-"10" supports STDP. It can be therefore studied the variation of the spiking pattern of the neuron "10", the trend of its plasticity parameter "M" and the variation of the parameter "$L_{03,10}$". It is worth to stress that it has been associated a bigger weight to the "01-10" connection (5000) w.r.t the "03-10" connection (2000) in order to make the "10" dependent more on "01" and, in this way, favour the decrease of L when "10" spike due to "01" instead of "03". The simulation of this second network

is shown in the Fig.3.12. In the top is shown the usual spiking pattern, in the mid of the picture is represented the "$M_0$3" and "$M_1$0" parameters (pre and post-synaptic M parameter of the connection "03"-"10"), in the bottom the "$L_{03,10}$" parameter.

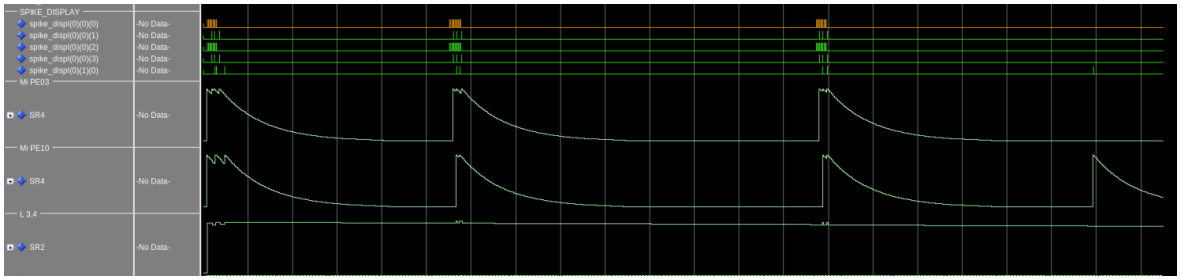

Figure 3.12: Questasim simulation of the second example network. In the top is shown the usual spiking pattern, in the mid of the picture is represented the "$M_0$3" and "$M_1$0" parameters, in the bottom the "$L_{03,10}$" parameter.

As can be seen, at each spike of the "i" neuron, the "M_i" parameter is reset to the maximum value and otherwise it decreases with an approximated exponential behaviour. At the same time, looking at the "$L_{03,10}$" parameter, it is decreased of the relative "M" amount after each spike of the neuron "03" and increased at each spike of the neuron "04".
The behaviour of the "M" parameters can be better analysed considering the Fig. 3.13. As can be seen, due to the first neuron emits initially a spike at each cycle, the M parameter remains constant and equal to its maximum value.

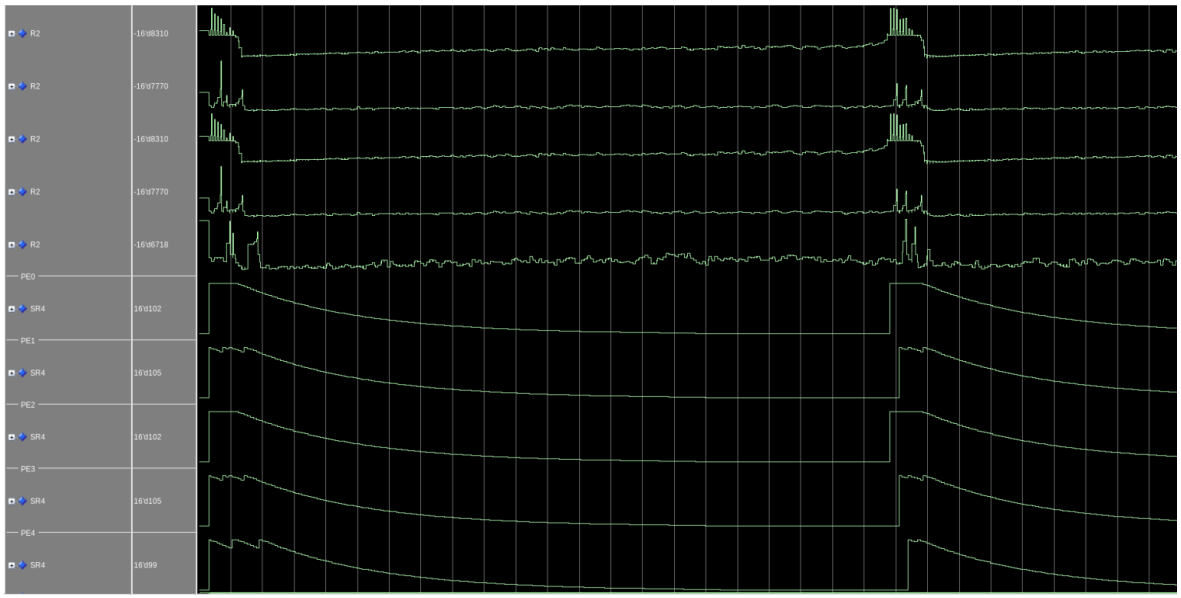

Figure 3.13: Analog representation of the membrane potential and of the "M" plasticity parameters for all the neurons.

The behaviour of the "L" parameter can be, instead, better analysed considering the Fig.3.14, zoom of the first burst in 3.12. It must be stressed that "L" decreases the next cycle w.r.t the pre-syn spike, while it is increased in the same cycle in which happens the post-syn spike, and this happens immediately before of spike representation on the screen, because the spike is dispatched at the end of the cycle. According to this, the first variation of L can be explained as follows: after the first spike of the neuron "30", the "L" parameter is decreased. Then, the second spike of "30" and the first of "10" compensate each other because they happen in sequence and L does not change, finally the L

is increased by the second spike of "10". With the same logic it can be explained the entire trend of "L".
The figure is not able to well describe it, but must be stressed that in the meanwhile L is decreasing very slowly according to its decay coefficient K'_{act} and this is the reason why there are tiny negative steps along the line, which must be evaluated as a lack of resolution of the "wave tool".

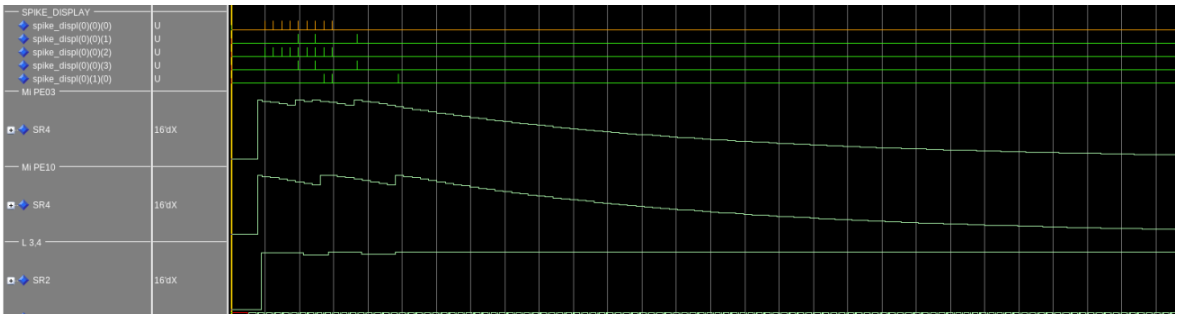

Figure 3.14: Zoom of the first burst in 3.12

What said demonstrate that the link is effectively strengthened or weakened according to the pre-synaptic and post-synaptic activity and the plasticity parameters evolves according to the equations 3.4 and 3.5.

CHAPTER 4

Conclusions and future work

This thesis presented the introduction in the HEENS multi-chip architecture of the Izhichevich neural model and of the "spike timing dependent plasticity" (STDP) learning rule. In particular, an algorithmic version of the two models has been described at instruction level in order to be executed by the HEENS architecture inside an assembly routine which describes the evolution along time and according to external stimuli of each neuron of a SNN. During the development phase, in both cases the project followed the same approach:

1. The Model is described as algorithm in MatLAB and tested with floating point precision.
2. The fixed-point computational features and the data organization in HEENS are emulated in MatLAB.
3. The two models obtained in MatLAB are compared in order to make some important design choice about parallelism, and evaluate the necessity of new hardware resources.
4. The algorithm is described at instruction level, and the entire routine of each neuron of the network is described in an assembly program.
5. The same networks are simulated in MatLAB and, at RTL level, in QuestaSim and the two outcomes are compared.

The results showed that the fixed point HEENS arithmetic can well approximate the floating-point version of the Izhikevich model according to a resolution of $10uV$, 16 bits words and a round to nearest integer technique.
The Izhikevich model has been introduced with success and the Gaussian noise generator needed to imitate the thalamic input in the brain has been designed and implemented in a hybrid hardware-firmware configuration based on four $LFSRs$.
Different types of neurons have been simulated obtaining a very similar behaviour with respect to the biological one. In particular, two types of excitatory neuron (RS, CH) and one inhibitory (FS) have been tested.
In **Chapter 2** it has been simulated, both in MatLAB and HEENS, a 16-neuron SNN executing the Izhikevich routine, obtaining optimal matching between the two outcomes. At the same time, the hardware computational cycle duration has proven to be able to reproduce biologic performance and thus to perform the entire network routine in less than $1ms$. Furthermore, even supporting STDP rule, it has been demonstrated that for sure also a 128 neurons SNN can achieve this goal.
The STDP rule has been introduced in the MatLAB program and some useful information about

the LTP and LTD of the neurons along the simulation time have been obtained, showing that in general, but even more in small networks, the depression phonomenon is more incisive. Moreover, in **Chapter 3** some networks have been tested in order to better show the results of the mapping of the STDP in HEENS. In this case, particular attention has been given to the trend of the main plasticity parameters (M and L), and how the spike pattern is modified by the insertion of the plasticity rule in the entire network, or at neuron level, or at connection level.
These kinds of results are a good entry point for the obtainment of a biologically accurate Spiking neural network, but there are other possible improvements to be addressed. Indeed, if the Izhikevich neuron model developed is widely adopted for generic purpose and guarantees good performance, the STDP introduced realizes only the antisymmetric Hebbian rule, while, as said, in the mammalian cortex other types of relationships have been identified. However, different Hebbian rules can be obtained simply changing the sign of the term added in formula 3.4, and so are ready to be used without great effort. Furthermore, the STDP routine developed allows to apply the plasticity rule to all the desired connections but does not consider differences in the rule of the inhibitory connections w.r.t those excitatory. In [4], for example, it is in fact hypothesised a variation of the strength of the inhibitory connections as a function of the strength of the excitatory connections of the same neuron, and other types of different relation are described in literature.
On the architecture side, moreover, there are several possible solutions to explore in order to speed up execution and improve precision. The STDP routine in HEENS, for example, make access twice per neuron to the memory. This, for sure, reduces performance and makes also more difficult handle, update and store data in the right memory position without time consuming inter-transfers inside the register file. This could be improved with more registers in the register file, or a new bank of shadow registers and it would benefit from a greater improvement with a bigger size of the memory row. In fact, a double width memory with a 64 bit word per row, instead of the current 32-bit, and the parallel loading of each of the four 16 bits words contained in the register file, would speed up execution allowing a single access to the memory.
In addition, with the aim of address short term improvements, together with more registers available, would be useful realize "common instructions", i.e. instructions involving at the same time multiple registers, in order to execute common shift, movements and so on. For example, for the "Membrane potential" routine can be useful operate a simultaneous shift of R0 and R1, and for the "synaptic calculation" routine, moreover, would be useful a simultaneous swap of R0 and R1.
During this project, it has been developed a first version of unsigned arithmetic in HEENS, with the insertion of an unsigned addition $ADDU$ and multiplication MUL. Future improvement could include more unsigned instructions and achieve better and more efficient hardware integration of these tasks. Considering the rounding, for example, the techniques until now adopted include "to nearest integer" scheme for the right shift instruction, but only an algorithmic rounding for the multiplication. A hardware circuit for the rounding of the result of the multiplication would improve the precision. On the hardware side, an important weight must be given to the test and the integration of the system in FPGA, operation which could not be done during this project but which must be addressed in each step of the development, in order to assure a good resources exploitation and the actual functioning of the architecture. Another interesting aspect in which put more attention would be that of precision. A bigger test campaign, simulating more and wider networks, indeed, would allow to identify the more suitable resolution for our model and at the same time to shape resolution according to the actual need of a certain application.
The development of applications for the neural network, and specifically tasks naturally sized for the SNNs, is the goal both in the mid-term and in the long-term. In the mid-term, some tasks to be

addressed can be, as an example, filter transfer functions and generic data processing. In the long-term, the aim is to integrate a SNN in a real biological system, making it communicate with the nervous system. Together with all these improvements and objectives, a separate mention must be reserved to the development of a graphical interface and a user-friendly configuration environment for the architecture. This would allow a better control, an easier test and for sure a general shorter development time, which is a fundamental objective.

APPENDIX A

Instruction Set

The instruction set of HEENS is described in the following. The instructions which have been modified during this project are shown in orange. The flag field stresses if the instruction can change the values of the "Carry" and "Zero" flags. In the right column, the operations done by each instruction are described.

	Instruction	Group	Class	Format	Opcode	Hex	Flags*	En**	Function
0	NOP	SEQ	0	NOP	000000	00			No operation
1	LDALL	REGISTERS	1	LDALL reg ****	000001	01	Z***	/F	reg <= DMEM (from sequencer)
2	LLFSR	MOVEMENT	0	LLFSR	000010	02	Z	/F	ACC<=LFSR(0);R1<=LFSR(1);SW0<=LFSR(2);SW1<=LFSR(3);
3	LOADSP	LOADSP	0	LOADSP	000011	03		/F	R1 & ACC(15:1) <= BRAM(BP,31:1); ACC(0) <= spike_register(BP(3:0))
4	STOREB	STOREB	0	STOREB	000100	04			EXT_BUFFER <= ACC
5	STORESP	STORESP	0	STORESP	000101	05		/F	BRAM(BP) <= R1 & ACC; BP <= BP + 1
6	STOREPS	STOREPS	0	STOREPS	000110	06		/F	AER_FIFO <= ACC(0) (post-synaptic Si)
7	RST	REGISTERS	1	RST reg	000111	07	Z***	/F	reg <= "0000"
8	SET	REGISTERS	1	SET reg	001000	08	Z***	/F	reg <= "FFFF"
9	SHLN	REGISTERS	2	SHLN n	001001	09	C,Z	/F	ACC <= ACC << n, (1 <= n <= 7), (n = number of positions)
10	SHRN	REGISTERS	2	SHRN n	001010	0A	C,Z	/F	ACC <= ACC >> n, (1 <= n <= 7), (n = number of positions)
11	RTL	REGISTERS	0	RTL	001011	0B	C,Z	/F	ACC <= ACC <<, carry = ACC(msb) Rotate Accumulator Left
12	RTR	REGISTERS	0	RTR	001100	0C	C,Z	/F	ACC <= ACC >>, carry = ACC(lsb) Rotate Accumulator Right
13	INC	ARITHMETIC	0	INC	001101	0D	C,Z	/F	ACC <= ACC + 1
14	DEC	ARITHMETIC	0	DEC	001110	0E	C,Z	/F	ACC <= ACC - 1
15	LOADSN	LOADSN	0	LOADSN	001111	0F	C,Z	/F	R1 & ACC <= BRAM(BP)
16	ADD	ARITHMETIC	1	ADD reg	010000	10	C,Z	/F	ACC <= ACC + reg (Saturated addition)
17	SUB	ARITHMETIC	1	SUB reg	010001	11	C,Z	/F	ACC <= ACC – reg (Saturated subtraction)
18	MUL	ARITHMETIC	1	MUL reg	010010	12	Z	/F	ACC & R1 <= ACC * reg (Unsigned product)
19	MULS	ARITHMETIC	1	MULS reg	010011	13	Z	/F	ACC & R1 <= ACC * reg (signed product)
20	AND	LOGIC	1	AND reg	010100	14	Z	/F	ACC <= ACC AND reg
21	OR	LOGIC	1	OR reg	010101	15	Z	/F	ACC <= ACC OR reg
22	INV	LOGIC	1	INV reg	010110	16	Z	/F	ACC <= INV reg
23	XOR	LOGIC	1	XOR reg	010111	17	Z	/F	ACC <= ACC XOR reg
24	MOVA	MOVEMENT	1	MOVA reg	011000	18	Z	/F	ACC <= reg
25	MOVR	MOVEMENT	1	MOVR reg	011001	19		/F	reg <= ACC
26	SWAPS	MOVEMENT	1	SWAPS reg	011010	1A	Z***	/F	reg <=> shadow_reg (Swap register)
27	MOVRS	MOVEMENT	1	MOVRS reg	011011	1B	Z***	/F	reg <= shadow_reg
28	LOOP	SEQ	5	LOOP n	011100	1C			Push LOOP_BUFFER(n-1);Push PC_BUFFER(PC+1)
29	LOOPV	SEQ	0	LOOPV ****	011101	1D			Push LOOP_BUFFER(DMEM-1);Push PC_BUFFER(PC+1)
30	ENDL	SEQ	0	ENDL	011110	1E			If LOOP_BUFFER = 0 then pop LOOP_BUFFER; pop PC_BUFFER; else LOOP_BUFFER <= LOOP_BUFFER – 1; PC <= PC_BUFFER
31	GOSUB	SEQ	4	GOSUB addr	011111	1F			PC <= addr; Push PC_BUFFER(PC+1)
32	RET	SEQ	0	RET	100000	20			PC <= PC_BUFFER
33	FREEZEC	CONDITIONAL	0	FREEZEC	100001	21	F		If C=1 then F <= 1; push F_BUFFER(1)
34	FREEZENC	CONDITIONAL	0	FREEZENC	100010	22	F		If C=0 then F <= 1; push F_BUFFER(1)
35	FREEZEZ	CONDITIONAL	0	FREEZEZ	100011	23	F		If Z=1 then F <= 1; push F_BUFFER(1)
36	FREEZENZ	CONDITIONAL	0	FREEZENZ	100100	24	F		If Z=0 then F <= 1; push F_BUFFER(1)
37	UNFREEZE	CONDITIONAL	0	UNFREEZE	100101	25	F		F <= pop F_BUFFER
38	HALT	SEQ	0	HALT	100110	26			INT<=1;sequencer halted until external input signal INT_ACK=1
39	SETZ	FLAGS	0	SETZ	100111	27	Z	/F	Z <= 1
40	SETC	FLAGS	0	SETC	101000	28	C	/F	Sets the carry flags C <= 1
41	CLRZ	FLAGS	0	CLRZ	101001	29	Z	/F	Clears the zero flags Z <= 0
42	CLRC	FLAGS	0	CLRC	101010	2A	C	/F	Clears the zero flags C <= 0
43	RANDON	RAND	0	RANDON	101011	2B		/F	random_en <= 1; LFSR becomes source register for LLFSR
44	SEED	MOVEMENT	0	SEED	101100	2C		/F	LFSR(0)(:)<= ACC; LFSR (1)(:)<=R1; LFSR(2)(:)<=LFSR(0)(:);LFSR(3)(:)<=LFSR(1)(:);
45	RANDOFF	RAND	0	RANDOFF	101101	2D		/F	random_en <= 0; LFSR_STEP <=0; LFSR disabled
46	SPKDIS	SEQ	0	SPKDIS	101110	2E			eo_exec <= 1, Stops the sequencer and stores spikes until input signal cam_en <= 0 (from AER control unit)

47	READMP	SEQ	4	READMP addr	101111	2F			DMEM <= BRAM(address)
48	RST_SEQ	SEQ	0	RST_SEQ	110000	30			Jumps to RESET state
49	ADDU	ARITHMETIC	1	ADDU reg	110001	31	C,Z	/F	Unsigned ADD
50	LAYERV	SEQ	2	LAYERV n	110010	32			VLAYERS <= n; CURR_VLAYER <= 0; defines number of virtual layers (currently 0 <= n <= 7)
51	GOTO	SEQ	4	GOTO addr	110011	33			PC <= addr
52	SHLAN	REGISTERS	2	SHLAN n	110100	34	C,Z	/F	ACC <= ACC << n, (1 <= n <= 7), Arithmetic shift
53	SHRAN	REGISTERS	2	SHRAN n	110101	35	C,Z	/F	ACC <= ACC >> n, (1 <= n <= 7), Arithmetic shift wih rounding
54	LOADBP	LOADBP	0	LOADBP ****	110110	36		/F	BP <= DMEM Loads PE BRAM pointer.
55	BITSET	REGISTERS	3	BITSET n	110111	37	Z	/F	ACC(n) <= 1
56	BITCLR	REGISTERS	3	BITCLR n	111000	38	Z	/F	ACC(n) <= 0
57	SPMOV	SPMOV	0	SPMOV n	111001	39		/F	Special MOVE. n = 0: VIRT <= ACC;
58	INCV	SEQ	0	INCV	111010	3A			VLAYER <= VLAYER + 1
59	READMPV	SEQ	4	READMPV addr	111011	3B			DMEM <= BRAM(address + VLAYER)
60	MOVSR	MOVEMENT	1	MOVSR reg	111100	3C		/F	shadow_reg <= reg

*Flags If the given instruction can change the indicated flag
** En
F: Frozen flag. /F= not(F) means unfrozen and the indicated instructions become enabled

*** Z can change only if ACC is set or reset (not in case of other registers)
**** See macros

MACRO INSTRUCTIONS: Conversion into elementary instructions.
It is recommended to use macro instructions instead of the associated simple instructions

1	LDALL			LDALL reg, const	reg <= DMEM(const) (from sequencer)
	Elementary instructions:			NOP	READ BRAM(addr(const)
				READMP const	DMEM <= const
				LDALL reg	reg <= const
29	LOOPV			LOOPV vp	Push LOOP_BUFFER(DMEM(vp)-1);Push PC_BUFFER(PC+1)
	Elementary instructions:			NOP	
				READMPV vp	
				LOOPV	
54	LOADBP			LOADBP bp	BP <= DMEM(bp) Loads PE BRAM pointer.
	Elementary instructions:			NOP	
				READMP bp	
				LOADBP	

APPENDIX B

PE parameters

```
# Chip parameters
rows = 4 #10
cols = 4 #10
vlayers = 8 # Number of virtual layers
synapses = 16 # Number of synapses per PE
# Limit (maxmium) parameters
row_max = 16 # Maximum number of rows that can be encoded
col_max = 16 # Maximum number of columns that can be encoded
virt_max = 8 # Maximum number of virtual layers
syn_max = row_max * col_max # Maximum number of synapses per PE (could be eventually more)
# Local spike memory
########################
syn_rows=2 #THE ONLY PARAMETER TO BE CHANGED IN ORDER TO USE DOUBLE ROWS NETLIST OR NOT ( 2 or 1)
########################
lcl_ram_width = 7 # local memory data bits of RAM
lcl_ram_size = 2**lcl_ram_width # Maximum number of synapses per PE
lcl_syn_memory = syn_rows*rows * cols #added to eventually write double rows
lcl_syn = rows * cols #16 # - 1 for no synapse. Number of local synapses,
lsyn_vl = [16, 16, 16, 16, 16, 16, 16, 16] # Local synapses assigned to each virtual layer
lsyn_base_addr = [0, 16, 32, 48, 64, 80, 96, 112] # Local synapses: Initial memory address for each virtual layer
lsyn_vl_addr=[lsyn_base_addr[i]*syn_rows for i in range(7)]
# Global spike memory
global_syn = 32 # - 1 for no synapse. Number of global synapses
# Synapse and neural memory
sn_ram_size = 1024  # synaptic_neural memory addresses
sn_ram_width = 32   # synaptic_neural memory data width
# PE array parameters
# PEID_bits = int(math.log(rows * cols, 2))  # number of bits for the PE ID
PEID_bits = 4  # number of bits for the PE ID

# Auxiliary functions
def text_2_int(data):
    if data[0] == '-': # Negative value
        data = data[1:]
        if data.isdigit():
            int_number = - int(data)
        else:
```

```
38              print('Error: Not an integer number -', data)
39              int_number = 'ERR'
40      elif data.isdigit():
41          int_number = int(data)
42      elif data[0:2] == '0X':
43          int_number = int(data[2:], 16)
44      else:
45          print('Error: Not an integer number ', data)
46          int_number = 'ERR'
47      return int_number
```

APPENDIX C

Izhikevich model assembly program

```
; GOTO CODE
;
; Izhikevic model for 16 fully connected neurons network
;
define virtual_layers 0 ; From 0 up to 7
define gsynapses 2 ; Up to 32 global synapses
define n_step 1;

.DATA

; Virtual layers

V0 = "0000000F" ; Number of assigned synapses (s-1) to the main layer
V1 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 1
V2 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 2
V3 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 3
V4 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 4
V5 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 5
V6 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 6
V7 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 7
;VLAYERS="00000003" ; Number of virtual layers.
VLAYERS="00000000" ; Number of virtual layers (n-1).

; Membrane potential parameters common to all neurons

;multiple of 10uV
VTHRES="FFFFF63C"   ; Threshold voltage -25 mV
N70="00001B58"; 70mV,
N0002="000068DC"; 0.0002*2^27
N001="00000028F"; 0.01, The noise generated by LFSR is 100 time bigger than necessary.
REST_POT="FFFFE69C"; -6500
N5="00000005";

;
;
; Neural and Synaptic RAM addresses
SYN_ADDR0="00000000"    ; First address of Synaptic parameters in SNRAM for V = 0.
SYN_ADDR1="00000010"    ; First address of Synaptic parameters in SNRAM for V = 1.
SYN_ADDR2="00000020"    ; First address of Synaptic parameters in SNRAM for V = 2.
SYN_ADDR3="00000030"    ; First address of Synaptic parameters in SNRAM for V = 3.
SYN_ADDR4="00000040"    ; First address of Synaptic parameters in SNRAM for V = 4.
```

```
42 SYN_ADDR5="00000050"    ; First address of Synaptic parameters in SNRAM for V = 5.
43 SYN_ADDR6="00000060"    ; First address of Synaptic parameters in SNRAM for V = 6.
44 SYN_ADDR7="00000070"    ; First address of Synaptic parameters in SNRAM for V = 7.
45 GSYN_ADDR="00000090"    ; First address of Global Synaptic parameters in SNRAM.
46 NEU_ADDR0="000003E3"    ; First address of Neural parameters in SNRAM (995) for V = 0.
47 NEU_ADDR1="000003E6"    ; First address of Neural parameters in SNRAM (997) for V = 1.
48 NEU_ADDR2="000003E9"    ; First address of Neural parameters in SNRAM (999) for V = 2.
49 NEU_ADDR3="000003EC"    ; First address of Neural parameters in SNRAM (1001) for V = 3.
50 NEU_ADDR4="000003EF"    ; First address of Neural parameters in SNRAM (1003) for V = 4.
51 NEU_ADDR5="000003F2"    ; First address of Neural parameters in SNRAM (1005) for V = 5.
52 NEU_ADDR6="000003F5"    ; First address of Neural parameters in SNRAM (1007) for V = 6.
53 NEU_ADDR7="000003F8"    ; First address of Neural parameters in SNRAM (1009) for V = 7.
54
55 SEED_ADDRL = "000003FD" ; Address of noise seed in SNRAM
56 SEED_ADDRH = "000003FE";Address of noise seed in SNRAM
57 PEID  = "000003FF" ; Address of  PE Identifier number
58 ;
59
60
61 .CODE
62 ;
63 GOTO MAIN ; Jump to main program
64 ;
65 ; *************************** PROCEDURES BEGIN ***************************
66 ;
67 .RANDOM_INIT   ; Uses R0 and R1
68  LOADBP SEED_ADDRL
69  LOADSN
70  SEED
71  LOADBP SEED_ADDRH
72  LOADSN
73  SEED     ;
74 RET
75 ;
76 .LOAD_NEURON   ; Uses R0, R1, R2, R3, R5
77  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
78  LOADBP    ; SNRAM pointer to currently processed neuron
79  LOADSN   ; Load Neural parameters from SNRAM to R1<=u & ACC<=Vmem
80  STORESP          ; FAKE STORE ONLY TO MAKE PC+1
81  MOVR R2   ; Move Vmem from ACC to R2
82  MOVA R1   ; ACC<=u
83  MOVR R3   ; put r3<=u
84  LOADSN           ; load ACC<=d; R1<=b
85  STORESP          ; FAKE STORE ONLY TO MAKE PC+1
86  MOVR R5   ; R5<=d
87  SWAPS R5  ; SW5<=d
88  MOVA R1   ; ACC<=R1<=b;
89  MOVR R5   ; R5<=b
90  LOADSN
91  MOVR R6   ; R6<=C
92  SWAPS R6  ; SW6<=C
93  MOVA R1
94  MOVR R6   ; R6<=A
95  MARK
96 RET
97 ;
98 .DETECT_SPIKE  ; Uses R0,R3 and R2
99  LDALL ACC VTHRES
```

```
100  SUB R2   ; Compare Vth - Vmem
101  SHLN 1   ;subtraction sign to C flag
102  RST ACC
103  FREEZENC ; If positive, freeze
104    ;REMEMBER: C stays in SW6.
105    SWAPS R6
106    MOVA R6
107    MOVR R2
108    MOVA R3;   ACC<= u
109    ;REMEMBER: d stays in SW5.
110    SWAPS R5;    R5<=D
111    ADD R5;    ACC<= u+d
112    SWAPS R5;
113    MOVR R3;   u<= u+d
114    SET ACC
115  UNFREEZE
116  STOREPS   ; Push spikes
117 RET
118 ;
119 .GAUSS_NOISE ; Uses SW0, R2, R6 and R7
120  RANDON                    ;LFSR ON
121  RANDOFF                   ;LFSR OFF. Arbitrarily heres
122  LLFSR     ;Noise seeds to ACC, R1, SR0, SR1
123  ADD R1
124  SHRAN 1
125  MOVR R7
126  SWAPS R0
127  SWAPS R1
128  ADD R1
129  SHRAN 1
130  ADD R7
131  LDALL R7 N001; ACC=OUTPUT
132  MULS R7   ;NOISE IN ACC
133  FREEZENC
134    INC
135  UNFREEZE
136  MOVR R4   ;NOISE IN R4
137  SWAPS R6
138  MOVA R6
139  SWAPS R6
140  LDALL R7 REST_POT
141  SUB R7
142  MOVA R4
143       FREEZEZ
144    LDALL R7 N5;
145    MULS R7
146    NOP
147    MOVA R1
148    SHRAN 1
149  UNFREEZE
150  MOVSR ACC                 ;TO STORE 1/2 THE NOISE
151 RET
152 ;
153 .SYNAPSE_CALC
154      LOADSP   ; Load Synaptic parameters and spike to R1 & ACC
155      SHRN 1   ; Move spike to flag C
156      FREEZENC
157  MOVA R1 ; Synaptic parameter to ACC
```

```
158  ADD R3;
159  MOVR R3;
160      UNFREEZE
161      RST ACC
162      STORESP  ; Stores synaptic parameter and increases BP for next synapse processing
163 RET
164 ;
165 .MEMBRANE_POTENTIAL ;Uses R0,R4,R7
166  MOVA R2
167  MULS R0  ; v^2*2^12
168  NOP  ; Check if needed
169  ; Shift R0R1 4 positions left
170  SHLN 4  ; Shift Accumulator 2^4
171  MOVR R4
172  MOVA R1  ; Move LS part (R1) to R0 (2^16)
173  SHRN 4
174  SHRAN 4
175  SHRAN 4  ; 2^16/2^12 = 2^4
176  ADD R4  ; Combine and obtain v^2/2^12
177  LDALL R4 N0002 ; 0.0002*2^27 is in R4
178  MULS R4  ; v^2*2^(-12)*0.0002*2^27/2^16 = 0.0002*v^2*2^(-1)
179  NOP  ; Check if needed
180  SHLN 1  ; Shift Accumulator 2^1
181  MOVR R4
182  MOVA R1  ; Move LS part (R1) to R0 (2^16)
183  SHRN 5
184  SHRAN 5
185  SHRAN 5  ; 2^16/2^15 = 2^1
186  ADD R4  ; Combine and obtain 0.0002*v^2
187  MOVR R7;
188  MOVA R2;          ACC<=Vinit
189  SHRAN 2;          ACC<=0.25*Vinit;
190  ADD R2;  ACC<=ACC+Vinit=1.25*Vinit
191  SHLAN 1;
192  ADD R7;
193  LDALL R4 N70;    R4<=70
194  ADD R4;
195  MOVR R7
196  RST ACC
197  SUB R3;   ACC=- u
198  SHRAN 1;
199  ADD R7;
200  ADD R2;  ACC=ACC+Vinit
201  MOVR R2;  Back to R2 where membrane potential is stored
202 RET
203 ;
204 .SUM_NOISE_AND_Ws
205  MOVRS ACC                  ;NOISE TO ACC
206  ADD R2     ;ADD NOISE TO SIGNAL
207  SWAPS R3    ;SYN. contribute
208  ADD R3;     add to membrane potential
209  MOVR R2;    store membrane potential
210  SWAPS R3;
211 RET
212 ;
213 .RECOVERY_UPDATE ;uses R3,R5,R6
214  MOVA R2;           ACC<=Vinit
215  MULS R5                  ;ACC<=R5*ACC=B*Vinit
```

```
 FREEZENC
  INC
 UNFREEZE
 SUB R3                    ;ACC<= ACC-R3= ACC-Uinit
 ;REMEMBER: A is in R6
 MULS R6;                   ACC<=A*ACC;
 FREEZENC
  INC
 UNFREEZE
 ADD R3;                    ACC<=ACC+Uinit
 MOVR R3;                   Back to R3 where recovery value is stored
RET
;
.STORE_NEURON ; uses R0,R3 and R1
 MOVA R3   ;move u from R3 to acc
 MOVR R1   ;move u from ACC to R1
 MOVA R2   ; Move Vmem from R2 to ACC
 READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
 LOADBP ; SNRAM pointer to currently processed neuron
 STORESP   ; Store u&Vmem to SNRAM
RET
;
; *************************** PROCEDURES END ****************************

; *************************** MAIN PROGRAMME BEGIN ***********************
.MAIN
;

; Virtual operation init
LAYERV virtual_layers ; Init sequencer vlayers. It is 0 for non-virtual operation
LDALL ACC, VLAYERS ; Load defined virtual layers to PE array
SPMOV 0 ; VIRT <= ACC


; Initial instructions
GOSUB RANDOM_INIT  ; For noise initialization


.EXEC_LOOP ; Execution loop

 LOOP virtual_layers ; Neuron loop for virtual operation
      NOP           ;to prevent pipeline error
      GOSUB LOAD_NEURON
      GOSUB DETECT_SPIKE
      GOSUB GAUSS_NOISE
      READMPV SYN_ADDR0
      LOADBP
      SWAPS R3        ;SYNAPSE Sum will be store IN SW3
      LOOPV V0         ;synaptic loop. Reads number of current-layer synapses
          NOP          ;to prevent pipeline error
          GOSUB SYNAPSE_CALC
      ENDL
      MOVA R3        ;move to acc synapse sum
      SHRAN 1;
      MOVR R3        ;move to R3
      SWAPS R3       ;SYNAPSE sum to SW3
      LOOP n_step
       NOP
```

```
274     GOSUB MEMBRANE_POTENTIAL     ;Calculate membrane potential according izhikevic
275           GOSUB SUM_NOISE_AND_Ws
276    ENDL
277    GOSUB RECOVERY_UPDATE
278    GOSUB STORE_NEURON
279    INCV
280   ENDL
281  .FINISH
282 NOP             ;Empty pipeline wait NOPs
283 NOP
284 NOP
285 SPKDIS           ;Distribute spikes
286 GOTO EXEC_LOOP          ;Execution loop
```

APPENDIX D

Netlist file

Netlist file describes the connections between neurons through their synapses. The developed networks are fully connected, so each of the 16 synapses (as many as the neurons) is used. This appendix shows a small part of the entire file: from line 3 to 28 are described the connections of all neurons with the first one (neuron 0), then with the second one (neuron 1) and so on. MSW and LSW (most and less significant 16-bit words) are the initial content of the first and second rows of synaptic parameters stored in SNRAM (see **Chapter 3** for further details). First is shown the netlist for the network implementing only the Izhikevich model, then is shown the netlist for the network supporting also STDP. In this last case, the structure of the file becomes:

MSW1: synaptic initial weight, **LSW1**: synaptic recovery variable, **MSW2**: 16 MSBs of Plasticity variable, **LSW2**: 16 LSBs of Plasticity variable.

```
1  #presyn  postsyn
2  #i  v  r  c|i  v  r  c  synN MSW LSW
3  #N  izhikevic  model
4  0  0  0  0  0  0  0  0  0  9  0
5  0  0  0  1  0  0  0  0  1  3  0
6  0  0  0  2  0  0  0  0  2  9  0
7  0  0  0  3  0  0  0  0  3  30  0
8  0  0  1  0  0  0  0  0  4  50  0
9  0  0  1  1  0  0  0  0  5  17  0
10 0  0  1  2  0  0  0  0  6  40  0
11 0  0  1  3  0  0  0  0  7  35  0
12 0  0  2  0  0  0  0  0  8  19  0
13 0  0  2  1  0  0  0  0  9  40  0
14 0  0  2  2  0  0  0  0  10  22  0
15 0  0  2  3  0  0  0  0  11  3  0
16 0  0  3  0  0  0  0  0  12  -9  0
17 0  0  3  1  0  0  0  0  13  -55  0
18 0  0  3  2  0  0  0  0  14  -16  0
19 0  0  3  3  0  0  0  0  15  -42  0
20 0  0  0  0  0  0  0  0  1  0  9  0
21 0  0  0  1  0  0  0  0  1  1  3  0
22 0  0  0  2  0  0  0  0  1  2  9  0
23 0  0  0  3  0  0  0  0  1  3  30  0
24 0  0  1  0  0  0  0  0  1  4  50  0
25 0  0  1  1  0  0  0  0  1  5  17  0
26 0  0  1  2  0  0  0  0  1  6  40  0
27 0  0  1  3  0  0  0  0  1  7  35  0
28  ...
```

```
#|presyn|   |postsyn| synaptic parameters
#|i|v|r|c|  |i|v|r|c| synN MSW1, LSW1, MSW2, LSW2
0 0 0 0 0 0 0 0 0 9 2048 20480 0
0 0 0 1 0 0 0 0 1 3 2048 20480 0
0 0 0 2 0 0 0 0 2 9 2048 20480 0
0 0 0 3 0 0 0 0 3 30 2048 20480 0
0 0 1 0 0 0 0 0 4 50 2048 20480 0
0 0 1 1 0 0 0 0 5 17 2048 20480 0
0 0 1 2 0 0 0 0 6 39 2048 20480 0
0 0 1 3 0 0 0 0 7 34 2048 20480 0
0 0 2 0 0 0 0 0 8 18 2048 20480 0
0 0 2 1 0 0 0 0 9 39 2048 20480 0
0 0 2 2 0 0 0 0 10 22 2048 20480 0
0 0 2 3 0 0 0 0 11 3 2048 20480 0
0 0 3 0 0 0 0 0 12 -10 2048 20480 0
0 0 3 1 0 0 0 0 13 -56 2048 20480 0
0 0 3 2 0 0 0 0 14 -16 2048 20480 0
0 0 3 3 0 0 0 0 15 -42 2048 20480 0
0 0 0 0 0 0 0 1 0 20 2048 20480 0
0 0 0 1 0 0 0 1 1 9 2048 20480 0
0 0 0 2 0 0 0 1 2 18 2048 20480 0
0 0 0 3 0 0 0 1 3 15 2048 20480 0
0 0 1 0 0 0 0 1 4 17 2048 20480 0
0 0 1 1 0 0 0 1 5 15 2048 20480 0
0 0 1 2 0 0 0 1 6 17 2048 20480 0
0 0 1 3 0 0 0 1 7 7 2048 20480 0
0 0 2 0 0 0 0 1 8 18 2048 20480 0
...
...
```

APPENDIX E

Neuron file

This appendix shows a small portion of the file *Neuron.txt*. This file is used to store in SNRAM the neural parameters for each neuron and each virtual layer. Line 1 indicates the row in memory from which write the content of the file. Line 2 contains the initial membrane potential and recovery variable values for all the 16 neurons of the virtual layer 0. Line 3 contains neural parameters "b" and "d" for all the 16 neurons virt layer 0. Line 4 contains neural parameters "a" and "c" for all the 16 neurons virt layer 0. Line 5 contains syn parameter "Mi" for all the 16 neurons virt layer 0. Line 6 contains the initial membrane potential and recovery variable values for all the 16 neurons of the virtual layer 1 and so on...
From line 14 the seeds for all the LFSRs are stored.

```
1  @0x3E3
2  4209829532 4209829532 4209829532 4209829532 4209829532 4209829532 4209829532
       4209829532 4209829532 4209829532 4209829532 4209829532 4209567388 4206945948
       4205373084 4199474844
3  858981144 858981138 858981134 858980920 858980954 858981150 858980630 858980832
       858980824 858981148 858980706 858980628 862322888 889258184 905052360 963510472
4  85976750 85976764 85976774 85977312 85977226 85976738 85978034 85977530 85977550
       85976740 85977844 85978042 424208028 381085340 355853980 262334108
5  1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024
6  4209829532 4209829532 4209829532 4209829532 4209829532 4209829532 4209829532
       4209829532 4209829532 4209829532 4209829532 4209829532 4209829532 4209829532
       4209829532 4209829532
7  858980552 858980552 858980552 858980552 858980552 858980552 858980552 858980552
       858980552 858980552 858980552 858980552 858980552 858980552 858980552 858980552
8  85976732 85976732 85976732 85976732 85976732 85976732 85976732 85976732 85976732
       85976732 85976732 85976732 85976732 85976732 85976732 85976732
9  1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024
10 {
11 ...
12 ...
13 }
14 @0x3FD 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
15 994858813 375875946 1109291642 1161525770 1607624595 1173498293 548411170 2002736044
       1239901623 491085887 1535858153 1752576516 1830184192 953383350 762532042
       1299207043
16 211557160 1430131952 329865713 78534401 1127485282 891368206 1983513249 1978012910
       553203795 1647773751 899895854 1749166241 1365065912 1861289903 481394589
       305344913
```

APPENDIX F

Neuron gen.m

```
%Creator: Antonio Caruso, May 2020
%This program print the neural parameters and noise seeds in a file called neuron.txt
%It can be used stand alone or inside the "Main" script, from whose
%workspace it takes input values.
%
%
%binary values of the same parameters are stored in binary_comp.txt for
%debugging

%% ------------DEFINE GLOBAL PARAMETERS----------
virt=8;
virt_used=1;

%%DECOMMENT if use stand alone
% Ne=12;
% Ni=4;
% seed=uint16((2^(15)-1)*rand(4,Ne+Ni));
% re=rand(virt,Ne); ri=rand(virt,Ni);
% a=[0.02*ones(virt,Ne), 0.02+0.08*ri];
% c=[-65+15*re.^2, -65*ones(virt,Ni)];
% b=[0.2*ones(virt,Ne), 0.25-0.05*ri];
% d=[8-6*re.^2, 2*ones(virt,Ni)];
%%

%%DECOMMENT if used within main.m
seed=[ESeed,ISeed];
a=a.';
c=c.';
b=b.';
d=d.';
c(virt_used+1:virt,:)=-65;
a(virt_used+1:virt,:)=0.02;
b(virt_used+1:virt,:)=0.2;
d(virt_used+1:virt,:)=2;

%STDP parameters
empty=zeros(8,16);
Mi= (2^14)*ones(8,16);

%randomized membrane parameters
vinit=-65*ones(virt,Ne+Ni);
```

```
uinit=b.*vinit;

%% ----------DEFINE MEMORY LINES COMPOSITION----------
input1=[uinit;b;a;empty];
input2=[vinit;d;c;Mi];
type1=[1;0;0;2];%0 number, 1 voltage
type2=[1;1;1;2];
bit1=16;
bit2=16;
n_lines=size(type1,1);%numero linee di memoria per ogni layer
integer=zeros(n_lines,Ne+Ni,'uint32');

%% ---------PRINT SECTION
fileIDi = fopen('neuron.txt','w');
fileIDb = fopen('binary_comp.txt','w');%for debug
fprintf(fileIDi,'@0x3E3\n');

for v=1:virt
for i=1:n_lines
    for j=1:Ne+Ni
    [int,bin]=param_memory_line( input1(v+virt*(i-1),j), input2(v+virt*(i-1),j), bit1,
     bit2, type1(i), type2(i) );
    integer((v-1)*3+i,j)=int;
    binar((v-1)*3+i+j,:)=bin;
    fprintf(fileIDi,'%d%1s',int,' ');
    fprintf(fileIDb,'%s%1s',bin,' ');
    end
fprintf(fileIDi,'\n');
fprintf(fileIDb,'\n');
end
end

fprintf(fileIDi,'@0x3FD 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\n');
fprintf(fileIDb,'Hereinafter seeds\n');

for i=1:Ne+Ni
[seed1(i),sbin1]=param_memory_line(seed(2,i),seed(1,i),16,16,2,2);
fprintf(fileIDi,'%d ',seed1(i));
fprintf(fileIDb,'%s ',sbin1);
end
fprintf(fileIDi,'\n');
fprintf(fileIDb,'\n');
for i=1:Ne+Ni
[seed2(i),sbin2]=param_memory_line(seed(4,i),seed(3,i),16,16,2,2);
fprintf(fileIDi,'%d ',seed2(i));
fprintf(fileIDb,'%s ',sbin2);
end
fprintf(fileIDi,'\n@1023 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\n');
fprintf(fileIDi,'################');
fclose(fileIDi);
fclose(fileIDb);


%% -----FUNCTION DEFINITION
%This function will write the binary word (and the corresponding int)
%of (bin(First input) concatenated bin(second input)) in bit1+bit2 bits.
%
%param_memory_line(parameter 1, parameter 2,bitlenght1,bitlenght2,type1, type2)
```

```
99 %types:0 are constant<1 in HEENS, 1 are voltage, 2 are number;
100 %voltage are in mV;
101 function [int_word,bin_word]= param_memory_line(b,a,lb,la,t1,t2)
102
103 if t1==t2
104     if t2==1
105         param1_int=double(a)*100;%voltage to be written
106         param2_int=double(b)*100;
107     elseif t2==0
108         param1_int=param_conv(a);
109         param2_int=param_conv(b);
110     elseif t2==2
111         param1_int=double(a);
112         param2_int=double(b);
113     else
114         error("types of input must be or Number (2) or voltage (1) or numeric constant
       in HEENS (0)");
115     end
116 elseif (t1==0 && t2==1) %if one is a number and other is a voltage, the voltage must
      be the 2nd input
117     param1_int=double(a)*100;
118     param2_int=param_conv(b);
119 else
120     error("field 5 and 6 are types of input, they must be voltage,voltage or number,
      number or contant,voltage");
121 end
122
123 LS16= conve(param1_int,la) ;%custom function that converts binary 2'compl
124 MS16= conve(param2_int,lb) ;
125
126 word(1,1:la)=(LS16);
127 word(1,la+1:la+lb)=(MS16);
128
129 for i=1:(la+lb)
130     word_inv(i)=word((la+lb)+1-i);
131 end
132
133 bin_word=num2str(word_inv,'%1d');
134 int_word=double(bin2dec(bin_word));
135 end
```

APPENDIX G

STDP netlist gen.m

```
%Creator: Antonio Caruso, May 2020
%this file generates a fully connected network netlist for izhikevich algorithm execution in HEENs

n=Ne+Ni;%Total number of neuron

Ni=floor(n/4);
Ne=n-Ni;
w=ceil(100*[0.5*rand(1,Ne),-rand(1,Ni)]);
p=round(100*S.');
Linit=20*2^10; % double line
M=2^11;

fileID = fopen('Double_netlist.lst','w');
fprintf(fileID,'#presyn postsyn \n#i v r c|i v r c s ph pl \n#N izhikevic model \n');
for i=1:n %exe 16 neuroni, 16 rows per ogni neurone
    c=0;
    rowi=floor((i-1)/4);
    coli=mod(i-1,4);
    for j=1:n
        rowj=floor((j-1)/4);
        colj=mod(j-1,4);
        %int_net= param_memory_line(p(j,i),Linit,8,8,2,2);
        %fprintf(fileID,'0 0 %d %d 0 0 %d %d %d %d %d\n',rowj,colj,rowi,coli,j-1,int_net,256);
        fprintf(fileID,'0 0 %d %d 0 0 %d %d %d %d %d %d 0\n',rowj,colj,rowi,coli,c,p(j,i),M, Linit);
        %fprintf(fileID,'+ 0 0 %d %d 0 0 %d %d %d %d 0 \n',rowj,colj,rowi,coli,c,Linit);
        %c=mod(c+1,16);
        c=c+1;
    end
end
fclose(fileID);
```

APPENDIX H

Izhi net.m

```
%Creator: Antonio Caruso, May 2020
%Starting from a nucleus of the script provided by Izhikevich in its pubblication, this algorithm reproduce Izhikevich algorithm execution in HEENS, comparing floating point model results with the fixed point one.A custom LFSRs inspired gaussian noise is introduced to exactly reproduce that acting in HEENS. STDP evolution is introduced for both models.

Ne=12; Ni=4;
clkcycles=2;

%STDP constant
alpha=10999/11000;%plasticity coefficient decay
beta=39/40; %synaptic recovery variable decay
%Initial STDP parameters for both models
L=20*ones(16,16);
M=2*ones(16,1);
Lf=20*ones(16,16);
Mf=2*ones(16,1);

%Izhichevik model parameters
re=rand(Ne,1); ri=rand(Ni,1);
a=[0.02*ones(Ne,1); 0.02+0.08*ri];
b=[0.2*ones(Ne,1); 0.25-0.05*ri];
c=[-65+15*re.^2; -65*ones(Ni,1)];
d=[8-6*re.^2; 2*ones(Ni,1)];

S=[0.5*rand(Ne+Ni,Ne), -rand(Ne+Ni,Ni)];
Sii=S;

v=-65*ones(Ne+Ni,1); % Initial values of v
u=b.*v; % Initial values of u
vf2=-65*ones(Ne+Ni,1);% Initial values of v for fixed point
uf2=b.*vf2;% Initial values of u for fixed point

f=zeros(Ne+Ni);
f2=zeros(Ne+Ni);
firings=[]; % spike timings
firings3=[];

ESeed=uint16((2^(15)-1)*rand(4,Ne));
ISeed=uint16((2^(15)-1)*rand(4,Ni));
```

```
38
39 Estate=int16(ESeed);%exctitatory
40 Istate=int16(ISeed);%inhibitory
41
42 %% simulation loop of 30000 ms
43 for t=1:30000
44
45 f1=zeros(16);
46 %LFSR_gaussian_noise is a custom function reproducing LFSR-made noise in HEENs
47 [ENoise,Estate]=LFSR_gaussian_noise(Estate,Ne,clkcycles);
48 [INoise,Istate]=LFSR_gaussian_noise(Istate,Ni,clkcycles);
49
50 I=[5.*ENoise;2.*INoise];% thalamic input
51 Iii=I;
52
53 fired=find(v>=-25);% indices of spikes for float
54 f1(fired)=1;
55 fired3=find(f2==1);% indices of spikes for fixed
56
57 firings=[firings; t+0*fired,fired];%matrix of spiked neuros
58 firings3=[firings3; t+0*fired3,fired3];%matrix of spiked neuros for fixed
59
60 %% SAMPLE PART FOR PLOT
61 vo(:,t)=v(:);%floating point version
62 vf2o(:,t)=vf2(:);%.7 fixed
63
64 %% STDP for floating point part
65  Lf=Lf.*alpha;
66  for i=1:16
67      for j=1:16
68          Lf(j,i)=Lf(j,i)+(Mf(j)*f1(i)-Mf(i)*f1(j));
69          if Lf(j,i)<0 %to prevent synapse from change sign
70              Lf(j,i)=0;
71          end
72      end
73   Mf(i)=2*f1(i)+(1-f1(i))*Mf(i)*beta;
74  end
75  Sf=S.*Lf./20;
76
77 %% STDP for fixed point part
78 L=L.*alpha;
79 for i=1:16
80     for j=1:16
81         L(j,i)=L(j,i)+(M(j)*f2(i)-M(i)*f2(j));
82         if L(j,i)<0 %to prevent synapse from change sign
83             L(j,i)=0;
84         end
85     end
86     M(i)=1*f2(i)+(1-f2(i))*M(i)*beta;
87 end
88 Sii=S.*L./20;
89
90
91 %% AFTER SPIKE UPDATE %%
92 v(fired)=c(fired);
93 u(fired)=u(fired)+d(fired);
94 vf2(fired3)=(c(fired3));
95 uf2(fired3)=(uf2(fired3)+d(fired3));
```

```
96
97 %% noise and synapse sum
98 I=I+sum(Sf(:,fired),2);%floatin
99 Iii=Iii+sum(Sii(:,fired3),2);
100
101 %%  MEMBRANE EVOLUTION %%%%%%%%%%%%%
102
103 %%% FLOATING POINT %%%
104 v=v+0.5*(0.04*v.^2+5*v+140-u+I); % step 0.5 ms, for numerical
105 v=v+0.5*(0.04*v.^2+5*v+140-u+I);%stability
106 u=u+a.*(b.*v-u);
107
108 %% FIXED POINT 7 decimals %%%
109 %%memb_potential_net_fixpt7 is a custom function emulating HEENs fixed point
        computational capability.
110  for i=1:16
111      [vf2(i),uf2(i),f2(i)]=memb_potential_net_fixpt7(vf2(i),uf2(i),a(i,1),b(i,1),Iii(i
         ,1));
112  end
113
114 end
115
116 %% plot part
117 t=1:550;
118  for i=1:16
119      figure('Name', "membrane potential");
120      plot(t,vo(i,t),t,vf2o(i,t));
121  end
122
123 figure('Name', "Spiking neurons comparison")
124 plot(firings(:,1),firings(:,2),"o");%floating point model spiking pattern
125 hold on
126 plot(firings3(:,1),firings3(:,2),"*");%fixed point model spiking pattern
127 set(gca, 'YDir','reverse')
128 xlabel('time [ms]');
129 ylabel('Spiking neuron');
130 axis([0 550 0 16]);
```

APPENDIX I

MATLAB Fixed point algorithm version

The "fixed point designer" app of the MATLAB suite allows the generation of a custom MATLAB function executing operation with fixed precision. The user is requested to specify the format and decimal precision for each variable, the parallelism, the kind of rounding and several other parameters. The **input function** is:

```
%Author: Antonio Caruso, April 2020
function v= memb_potential(a,b,c,d)
v=c;
u=b.*v;
vth=int16(-25);

for t=2:1000 % simulation of 1000 ms
v=v+0.5*(0.04*v.^2+5*v+140-u); % step 0.5 ms
v=v+0.5*(0.04*v.^2+5*v+140-u); % for numerical
u=u+a.*(b.*v-u); % stability

    if (v>vth)
        v=c;
        u=u+d;
    end
end
end
```

The **output function** is:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%           Generated by MATLAB 9.7 and Fixed-Point Designer 6.4                %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ v, u ,f ] = memb_potential_net_fixpt7(v_1,u_1,a,b,I)
fm = get_fimath();
u = fi(u_1, 1, 16, 7, fm);
v = fi(v_1, 1, 16, 7, fm);

f=0;
vth=int16(-25);

v=fi(v+fi(0.5, 0, 16, 16, fm)*(fi(0.04, 0, 16, 20, fm)*v.^2+fi(5, 0, 3, 0, fm)*v+fi
    (140, 0, 8, 0, fm)-u+I), 1, 16, 7, fm); % step 0.5 ms
v(:)=v+fi(0.5, 0, 16, 16, fm)*(fi(0.04, 0, 16, 20, fm)*v.^2+fi(5, 0, 3, 0, fm)*v+fi
    (140, 0, 8, 0, fm)-u+I); % for numerical
u=fi(u+a.*(b.*v-u), 1, 16, 7, fm); % stability
```

```
    if (v>vth)
        f(:)=1;
    end
end

%for better rounding you can use Nearest instead of Floor
function fm = get_fimath()
 fm = fimath('RoundingMethod', 'Floor',...
       'OverflowAction', 'Saturate',...
       'ProductMode','FullPrecision',...
       'SumMode','FullPrecision');
end
```

APPENDIX J

LFSR gaussian noise.m

```
%Author: Antonio Caruso, may 2020
%4-LFSRs model
LFSR=zeros(4,1);
LSB=LFSR;
samples=1000;

LFSR=[11249;30671;4088;23939];
LFSR=int16(LFSR);

for c=1:samples

for j=1:4
LSB(j)=bitand(LFSR(j),1);
end

%d=LFSR;
%step1=dec2bin(LFSR)
LFSR=bitsra(LFSR,1);
%step2=dec2bin(LFSR)

for j=1:4%4 lfsr
    if LSB(j)==1
        LFSR(j)= bitor(LFSR(j),int16(-32768));
        LFSR(j)=bitxor(LFSR(j),13312);
    else
        LFSR(j)= bitand(LFSR(j),32767);
    end
end%4 LFSR

LFSRs=double(LFSR);
y(c)=(( LFSRs(1)/2+LFSRs(2)/2 )/2 + (LFSRs(3)/2+LFSRs(4)/2)/2 )/2;

end%clock cicles
figure(2)
histogram(y)
```

APPENDIX K

LFSR test.m

```
%Author: Antonio Caruso, may 2020
%comparison between matlb random noise and 4-LFSRs model one
close all
hold on;

random=randn(100000,1);
random_pd=fitdist(random,'Normal');
%plot the histogram
h_random=histogram(random,'Normalization','pdf');

Nbins_r=h_random.NumBins;
[N,edges] = histcounts(random,Nbins_r,'Normalization','pdf');
edges = edges(2:end) - (edges(2)-edges(1))/2;
y_random=pdf(random_pd,edges);
%plot the line interpolating the histogram
plot(edges,N,edges,y_random);
hold off;

figure(2)
plot(edges, y_random - N);

MSE_R=mean((y_random - N).^2);


%----------LFSR----------------

LFSR=four_LFSR_test(100000);
LFSR=LFSR.'
LFSR_pd=fitdist(LFSR,'Normal');

%plot the histogram
figure(3)
h_LFSR=histogram(LFSR,'Normalization','pdf');
hold on;
Nbins_L=h_LFSR.NumBins;
[N,edges] = histcounts(LFSR,Nbins_L,'Normalization','pdf');
edges = edges(2:end) - (edges(2)-edges(1))/2;
y_LFSR=pdf(LFSR_pd,edges);
%plot the line interpolating the histogram
plot(edges,N,edges,y_LFSR);
hold off;
```

```
42
43 figure(4)
44 plot(edges, y_LFSR - N);
45 MSE_L=mean((y_LFSR - N).^2);
```

APPENDIX L

Simulating different types of neurons

```
1  ; GOTO CODE
2  ;
3  ; Integrate and fire. Both non-virtual and virtual
4  ; DEFAULT operation without virtual layers
5  ; REMOVE: 'semicolon%VIRT ' for operation with virtual layers
6
7  ; Network definitions
8
9  define virtual_layers 0 ; From 0 up to 7
10 define gsynapses 2 ; Up to 32 global synapses
11 define n_step 1;
12
13 .DATA
14
15 ; Virtual layers
16
17 V0 = "00000001" ; Number of assigned synapses (s-1) to the main layer
18 V1 = "00000001" ; Number of assigned synapses (s-1) to virtual layer 1
19 V2 = "00000001" ; Number of assigned synapses (s-1) to virtual layer 2
20 V3 = "00000001" ; Number of assigned synapses (s-1) to virtual layer 3
21 V4 = "00000001" ; Number of assigned synapses (s-1) to virtual layer 4
22 V5 = "00000001" ; Number of assigned synapses (s-1) to virtual layer 5
23 V6 = "00000001" ; Number of assigned synapses (s-1) to virtual layer 6
24 V7 = "00000001" ; Number of assigned synapses (s-1) to virtual layer 7
25 VLAYERS="00000000" ; Number of virtual layers (n-1).
26
27 ; Membrane potential parameters common to all neurons
28
29 ;multiple of 10uV
30 VTHRES="FFFFF63C"   ; Threshold voltage -25 mV
31 N70="00001B58"; 70mV,
32 N0002="000068DC"; 0.0002*2^27
33 ;N001="00000028F"; 0.01, The noise generated by LFSR is 100 time bigger than necessary
       .
34 REST_POT="FFFFE69C"; -6500
35 N5="00000005";
36
37 ;RS neural parameters:
38 izhi_B= "00003333"   ;0.2
39 izhi_D= "00000320"   ;8mV
40 izhi_C= "FFFFE69C"   ; -65mV resting potential
```

```
izhi_A= "0000051E"; 0.02
;FS neural parameters:
;izhi_B= "00003333"   ;0.2
;izhi_D= "000000C8"   ;2mV
;izhi_C= "FFFFE69C"   ; -65mV resting potential
;izhi_A= "00001999"; 0.1
;CH neural parameters
;izhi_B= "00003333"   ;0.2
;izhi_D= "000000C8"   ;2mV
;izhi_C= "FFFFEC78"   ; -50mV resting potential
;izhi_A= "0000051E"; 0.02

;
; Neural and Synaptic RAM addresses
SYN_ADDR0="00000000"    ; First address of Synaptic parameters in SNRAM for V = 0.
SYN_ADDR1="00000002"    ; First address of Synaptic parameters in SNRAM for V = 1.
SYN_ADDR2="00000004"    ; First address of Synaptic parameters in SNRAM for V = 2.
SYN_ADDR3="00000006"    ; First address of Synaptic parameters in SNRAM for V = 3.
SYN_ADDR4="00000008"    ; First address of Synaptic parameters in SNRAM for V = 4.
SYN_ADDR5="0000000A"    ; First address of Synaptic parameters in SNRAM for V = 5.
SYN_ADDR6="0000000C"    ; First address of Synaptic parameters in SNRAM for V = 6.
SYN_ADDR7="0000000E"    ; First address of Synaptic parameters in SNRAM for V = 7.
GSYN_ADDR="00000064"    ; First address of Global Synaptic parameters in SNRAM.
NEU_ADDR0="000003E3"    ; First address of Neural parameters in SNRAM (995) for V = 0.
NEU_ADDR1="000003E5"    ; First address of Neural parameters in SNRAM (997) for V = 1.
NEU_ADDR2="000003E7"    ; First address of Neural parameters in SNRAM (999) for V = 2.
NEU_ADDR3="000003E9"    ; First address of Neural parameters in SNRAM (1001) for V = 3.
NEU_ADDR4="000003EB"    ; First address of Neural parameters in SNRAM (1003) for V = 4.
NEU_ADDR5="000003ED"    ; First address of Neural parameters in SNRAM (1005) for V = 5.
NEU_ADDR6="000003EF"    ; First address of Neural parameters in SNRAM (1007) for V = 6.
NEU_ADDR7="000003F1"    ; First address of Neural parameters in SNRAM (1009) for V = 7.

SEEDH_ADDR = "000003FD" ; Address of noise seed in SNRAM
SEEDL_ADDR = "000003FE" ;
NOISE= "000003E8"; noise for test is 10mV, 1280
PEID  = "000003FF" ; Address of  PE Identifier number



.CODE
;
GOTO MAIN ; Jump to main program
;
; *************************** PROCEDURES BEGIN ***************************
;
.RANDOM_INIT   ; Uses R0 and R1
 LOADBP SEEDH_ADDR
 LOADSN
 SEED     ; High seed
 LOADBP SEEDL_ADDR
 LOADSN
 SEED     ; Low seed
RET
;
.LOAD_NEURON   ; Uses R0, R1, R2, R3, R5
 READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
 LOADBP    ; SNRAM pointer to currently processed neuron
 LOADSN   ; Load Neural parameters from SNRAM to R1<=u & ACC<=Vmem
```

```
 MOVR R2   ; Move Vmem from ACC to R2
 MOVA R1   ;ACC<=u
 MOVR R3   ;r3<=u
 MARK
RET
;
.MEMBRANE_POTENTIAL ;Uses R0,R4,R7
 MOVA R2
 MULS R0   ; v^2*2^12
 NOP   ;
 ; Shift R0R1 4 positions left
 SHLN 4   ; Shift Accumulator 2^4
 MOVR R4
 MOVA R1   ; Move LS part (R1) to R0 (2^16)
 SHRN 4
 SHRAN 4
 SHRAN 4   ; 2^16/2^12 = 2^4
 ADD R4   ; Combine and obtain v^2/2^12
 LDALL R4 N0002 ; 0.0002*2^27 is in R4
 MULS R4   ; v^2*2^(-12)*0.0002*2^27/2^16 = 0.0002*v^2*2^(-1)
 NOP   ;
 SHLN 1   ; Shift Accumulator 2^1
 MOVR R4
 MOVA R1   ; Move LS part (R1) to R0 (2^16)
 SHRN 5
 SHRAN 5
 SHRAN 5   ; 2^16/2^15 = 2^1
 ADD R4   ; Combine and obtain 0.0002*v^2
 MOVR R7;
 MOVA R2;          ACC<=Vinit
 SHRAN 2;          ACC<=0.25*Vinit;
 ADD R2;   ACC<=ACC+Vinit=1.25*Vinit
 SHLAN 1;
 ADD R7;
 LDALL R4 N70;     R4<=70
 ADD R4;
 MOVR R7
 RST ACC
 SUB R3;    ACC=- u
 SHRAN 1;
 ADD R7;
 ADD R2;   ACC=ACC+Vinit
 MOVR R2;   Back to R2 where membrane potential is stored
RET
;
.ADD_NOISE ; Uses R0, R2 and R5
 LDALL R7, NOISE
 MOVA R7;
 SHRAN 1; due to 2 steps
 ADD R2 ; Add to Vmem
 MOVR R2 ; Back to R2
RET
;
.SYNAPSE_CALC
    LOADSP   ; Load Synaptic parameters and spike to R1 & ACC
    SHRN 1   ; Move spike to flag C
    FREEZENC
 MOVA R1   ; Synaptic parameter to ACC
```

```
157         SWAPS R1;
158  ADD R1;
159  MOVR R1;
160  SWAPS R1;
161      UNFREEZE
162      RST ACC
163      STORESP   ; Stores synaptic parameter and increases BP for next synapse processing
164 RET
165 ;
166 .RECOVERY_UPDATE ;uses R3,R5,R6
167  MOVA R2          ;ACC<=Vinit
168  LDALL R5 izhi_B
169  MULS R5                  ;ACC<=R5*ACC=B*Vinit
170  SUB R3                   ;ACC<= ACC-R3= ACC-Uinit
171  LDALL R6 izhi_A
172  MULS R6                  ;ACC<=A*ACC;
173  ADD R3                   ;ACC<=ACC+Uinit
174  MOVR R3                  ;Back to R3 where recovery value is stored
175 RET
176 ;
177 .DETECT_SPIKE ; Uses R0,R3 and R2
178  LDALL ACC VTHRES
179  SUB R2   ; Compare Vth - Vmem
180  SHLN 1   ;subtraction sign to C flag
181  RST ACC
182  FREEZENC ; If positive, freeze
183   LDALL R2 izhi_C ; Vmem to resting potential
184   MOVA R3;  ACC<= u
185   LDALL R5 izhi_D
186   ADD R5;   ACC<= u+d
187   MOVR R3;  u<= u+d
188   SET ACC
189  UNFREEZE
190  STOREPS   ; Push spikes
191 RET
192 ;
193 .STORE_NEURON ; uses R0,R3 and R1
194  MOVA R3   ;move u from R3 to acc
195  MOVR R1   ;move u from ACC to R1
196  MOVA R2   ; Move Vmem from R2 to ACC
197  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
198  LOADBP ; SNRAM pointer to currently processed neuron
199  STORESP   ; Store u&Vmem to SNRAM
200 RET
201 ;
202 ; ************************** PROCEDURES END ****************************
203
204 ; ************************** MAIN PROGRAMME BEGIN ***********************
205 .MAIN
206 ;
207
208 ; Virtual operation init
209 LAYERV virtual_layers ; Init sequencer vlayers. It is 0 for non-virtual operation
210 LDALL ACC, VLAYERS ; Load defined virtual layers to PE array
211 SPMOV 0 ; VIRT <= ACC
212
213
214 ; Initial instructions
```

```
215 GOSUB RANDOM_INIT ; For noise initialization
216
217
218 .EXEC_LOOP ; Execution loop
219
220 LOOP virtual_layers ; Neuron loop for virtual operation
221     NOP   ;to prevent pipeline error
222     GOSUB LOAD_NEURON
223     GOSUB DETECT_SPIKE
224     READMPV SYN_ADDR0
225     LOADBP
226     LOOPV V0  ; synaptic loop. Reads number of current-layer synapses
227             NOP   ;to prevent pipeline error
228             GOSUB SYNAPSE_CALC
229     ENDL
230     SWAPS R1; take synapse weight
231     MOVA R1;  move to acc
232     SHRAN 1;  divide cause 2 steps
233     MOVR R1;  move to R1
234     SWAPS R1; move to SW1
235     LOOP n_step
236      NOP
237  GOSUB MEMBRANE_POTENTIAL ; Calculate membrane potential according izhikevic
238  GOSUB ADD_NOISE
239  SWAPS R1;
240  MOVA R2;
241  ADD R1;
242  MOVR R2;
243  SWAPS R1;
244     ENDL
245     GOSUB RECOVERY_UPDATE
246     GOSUB STORE_NEURON
247     INCV
248 ENDL
249 .FINISH
250 NOP  ; Empty pipeline wait NOPs
251 NOP
252 NOP
253 SPKDIS ; Distribute spikes
254 GOTO EXEC_LOOP ; Execution loop
```

APPENDIX M

ALU

```
1  ----------------------------------------------------------------------------
2  -- Project Name:  HEENS
3  -- Design Name:   ALU.vhd
4  -- Module Name:   ALU - behavioral
5  --
6  -- Creator: Sergi Juan, Mireya Zapata & Jordi Madrenas
7  -- Company: Universitat Politecnica de Catalunya (UPC)
8  --
9  --
10 -- Description:
11 -- ALU of the Processing Element
12 --
13 -- Revision 1: Jordi Madrenas
14 -- 09/07/2020: Connect to carry the MSbit of R1 in MUL/MULS to simplify R0 round (Most
       Significant 16-bit Word)
15 --
16 --Revision 2: Antonio Caruso
17 --10/08/2020: Rounding to nearest integer in SHRAN, Z flag fix, unsigned addition "
       ADDU",
18 --unsigned multiplication "MUL" with result to R0 R1, signed multiplication "MULS"
       with result to R0 R1.
19 --
20 -- Additional Comments:
21 --
22 --
23 ----------------------------------------------------------------------------
24
25 library ieee;
26 use ieee.std_logic_1164.all;
27 use ieee.std_logic_arith.all;
28 use ieee.std_logic_unsigned.all;
29
30 use work.SNN_pkg.all;
31
32 entity ALU is
33     port (
34           clk       : in  std_logic;
35           reset     : in  std_logic;
36           InA       : in  std_logic_vector(15 downto 0);
37           InB       : in  std_logic_vector(15 downto 0);
38           OP_CODE   : in  std_logic_vector(5 downto 0);
```

```
39          OutCarry : out std_logic;
40          OutZero  : out std_logic;
41          OutSolve : out std_logic_vector(31 downto 0)
42          );
43 end ALU;
44
45 architecture behavioral of ALU is
46
47     signal sumA          : std_logic_vector(15 downto 0);
48     signal sumB          : std_logic_vector(15 downto 0);
49     signal sumA_reg      : std_logic_vector(15 downto 0);
50     signal sumB_reg      : std_logic_vector(15 downto 0);
51     signal sgn           : std_logic_vector(15 downto 0);
52     signal res           : std_logic_vector(31 downto 0);
53  signal out_res    : std_logic_vector(31 downto 0);
54     signal res_aux_add   : std_logic_vector(16 downto 0);
55     signal res_aux_sub   : std_logic_vector(16 downto 0);
56     signal res_MUL    : std_logic_vector(31 downto 0);
57  signal res_MULS   : std_logic_vector(31 downto 0);
58     signal res_inc_dec   : std_logic_vector(15 downto 0);
59     signal r_MULS        : std_logic_vector(31 downto 0);
60  signal r_MUL         : std_logic_vector(31 downto 0);
61     signal max_res_add   : std_logic;
62     signal max_res_sub   : std_logic;
63     signal min_res_add   : std_logic;
64     signal min_res_sub   : std_logic;
65     signal ShiftingBits  : std_logic_vector(2 downto 0);
66     signal SetClr_Bit    : std_logic_vector(3 downto 0);
67     signal C_SHLN, C_SHLAN, C_SHRN, C_RTL, C_RTR, C_INC, C_DEC, C_ADD, C_ADDU, C_SUB :
        std_logic;
68     signal Z_RES,Z_SH, Z_AND, Z_OR, Z_XOR : std_logic;
69     signal S_RST, S_SET,S_SH , S_SHLN, S_SHLAN, S_SHRN, S_SHRAN, S_RTL, S_RTR, S_AND,
       S_OR, S_XOR, S_BITSET, S_BITCLR : std_logic_vector (15 downto 0);
70     signal signal_ADD, signal_ADDU: std_logic;
71     signal signal_SUB: std_logic;
72     signal signal_INC: std_logic;
73     signal signal_DEC: std_logic;
74     signal signal_MUL: std_logic;
75
76 begin
77
78  -- Opcode Decoding
79     signal_ADD <= '1' when (OP_CODE = OP_ADD) else
80                   '0';
81     signal_ADDU <= '1' when (OP_CODE = OP_ADDU) else
82                   '0';
83     signal_SUB <= '1' when (OP_CODE = OP_SUB) else
84                   '0';
85     signal_MUL <= '1' when ((OP_CODE = MUL) OR (OP_CODE = MULS)) else
86                   '0';
87     signal_INC <= '1' when (OP_CODE = INC) else
88                   '0';
89     signal_DEC <= '1' when (OP_CODE = DEC) else
90                   '0';
91
92
93  -- Carry out
94     with OP_CODE select
```

```
95      OutCarry <= C_SHLN   when SHLN,
96                  C_SHLAN  when SHLAN,
97                  C_SHRN   when SHRN,
98                  C_RTL    when RTL,
99                  C_RTR    when RTR,
100                 C_INC    when INC,
101                 C_DEC    when DEC,
102                 C_ADD    when OP_ADD|OP_ADDU,
103                 C_SUB    when OP_SUB,
104                 r_MULS(31) when MULS,  -- Introduced to allow for easy 16-bit rounding
105     r_MUL(31) when MUL,   -- Introduced to allow for easy 16-bit rounding
106                 '1'      when SETC,
107                 '0'      when others;
108
109 -- Zero out
110     with OP_CODE select
111     OutZero <= '1'    when SETZ | RST,
112                '0'    when CLRZ | SET,
113                Z_RES  when INC | DEC | OP_ADD | OP_ADDU | OP_SUB | BITSET | BITCLR,
114  Z_SH when SHLAN | SHRAN | SHRN |SHLN,
115                Z_AND  when OP_AND,
116                Z_OR   when OP_OR,
117                Z_XOR  when OP_XOR | INV,
118                '0'    when others;
119
120 -- Output (ALU result)
121     with OP_CODE select
122     OutSolve <= x"0000" & S_RST            when RST,
123                 x"0000" & S_SET            when SET,
124                 x"0000" & S_SHLN           when SHLN,
125                 x"0000" & S_SHLAN          when SHLAN,
126                 x"0000" & S_SHRN           when SHRN,
127                 x"0000" & S_SHRAN          when SHRAN,
128                 x"0000" & S_RTL            when RTL,
129                 x"0000" & S_RTR            when RTR,
130                 out_res     when OP_ADD | OP_ADDU | OP_SUB | MULS | MUL | INC | DEC |
    BITCLR | BITSET,
131                 x"0000" & S_AND            when OP_AND,
132                 x"0000" & S_OR             when OP_OR,
133                 x"0000" & S_XOR            when OP_XOR,
134                 x"0000" & S_XOR            when INV,
135                 x"00000000"                when others;
136
137
138 with OP_CODE select
139 S_SH<= S_SHLN          when SHLN,
140                 S_SHLAN         when SHLAN,
141                 S_SHRN          when SHRN,
142                 S_SHRAN         when SHRAN,
143         x"0000" when others;
144
145 -- Aux. signal InA
146     with OP_CODE select
147     sumA <= InA      when SHLN | SHLAN | SHRN | SHRAN | OP_ADD | OP_ADDU | OP_SUB | MUL
      | MULS | OP_AND | OP_OR | OP_XOR | BITSET | BITCLR | RTL | RTR | INC | DEC,
148             x"FFFF" when INV,
149             x"0000" when others;
150
```

```
151 -- Aux. signal InB
152     with OP_CODE select
153     sumB <= InB       when SHLN | SHLAN | SHRN | SHRAN | OP_ADD | OP_ADDU | OP_SUB | MUL
          | MULS | OP_AND | OP_OR | OP_XOR | BITSET | BITCLR | INV,
154             x"0001" when INC,
155             x"FFFF" when DEC,
156             x"0000" when others;
157
158 -- Number of bits to shift
159     ShiftingBits <= sumB(2 downto 0);
160     SetClr_Bit <= sumB(3 downto 0);
161
162 -- ALU operations
163
164 -- Carry out
165     with ShiftingBits select
166     C_SHLN <= sumA(N_bits - 1) when "001",
167               sumA(N_bits - 2) when "010",
168               sumA(N_bits - 3) when "011",
169               sumA(N_bits - 4) when "100",
170               sumA(N_bits - 5) when "101",
171               sumA(N_bits - 6) when "110",
172               sumA(N_bits - 7) when "111",
173               sumA(N_bits - 8) when others;
174
175     with ShiftingBits select
176     C_SHLAN <= sumA(N_bits - 2) when "001",
177                sumA(N_bits - 3) when "010",
178                sumA(N_bits - 4) when "011",
179                sumA(N_bits - 5) when "100",
180                sumA(N_bits - 6) when "101",
181                sumA(N_bits - 7) when "110",
182                sumA(N_bits - 8) when "111",
183                sumA(N_bits - 9) when others;
184
185     with ShiftingBits select
186     C_SHRN <= sumA(0) when "001",
187               sumA(1) when "010",
188               sumA(2) when "011",
189               sumA(3) when "100",
190               sumA(4) when "101",
191               sumA(5) when "110",
192               sumA(6) when "111",
193               sumA(7) when others;
194
195     C_RTL <= sumA(N_bits - 1);
196
197     C_RTR <= sumA(0);
198
199     with sumA select
200     C_INC <= '1' when x"7FFF",
201              '0' when others;
202
203     with sumA select
204     C_DEC <= '1' when x"8000",
205              '0' when others;
206
207     carry_add: process (sumA, sumB, res)
```

```
208     begin
209
210  if OP_CODE = OP_ADDU then
211   if ((sumA(15) = '1' OR sumB(15) = '1') AND res_aux_add(16) = '1') then
212               C_ADD <= '1';
213           else
214               C_ADD <= '0';
215           end if;
216  elsif OP_CODE = OP_ADD then
217   if ((sumA(15) = '0' AND sumB(15) = '0' AND res(15) = '1')
218                   OR (sumA(15) = '1' AND sumB(15) = '1' AND res(15) = '0')) then
219               C_ADD <= '1';
220           else
221               C_ADD <= '0';
222           end if;
223  else null;
224  end if;
225     end process;
226
227
228     carry_sub: process (sumA, sumB, res)
229     begin
230
231         if ((sumA(15) = '1' AND sumB(15) = '0' AND res(15) = '0')
232             OR (sumA(15) = '0' AND sumB(15) = '1' AND res(15) = '1')) then
233             C_SUB <= '1';
234         else
235             C_SUB <= '0';
236         end if;
237
238     end process;
239
240  -- Zero out
241
242  With OP_CODE select
243  out_res<= res when  OP_ADD | OP_ADDU | OP_SUB,
244   r_MULS when MULS,
245   r_MUL when MUL,
246   x"0000" & res_inc_dec     when INC | DEC,
247   x"0000" & S_BITSET        when BITSET,
248                   x"0000" & S_BITCLR        when BITCLR,
249   x"00000000"               when others;
250
251     with out_res select
252     Z_RES <= '1' when x"00000000",
253              '0' when others;
254
255     with S_SH select
256     Z_SH <= '1' when x"0000",
257       '0' when others;
258
259     with S_AND select
260     Z_AND <= '1' when x"0000",
261               '0' when others;
262
263     with S_OR select
264     Z_OR <= '1' when x"0000",
265              '0' when others;
```

```
266
267     with S_XOR select
268     Z_XOR <= '1' when x"0000",
269              '0' when others;
270
271 -- Auxiliary OutSolve out
272
273 -- Reset all out bits
274     S_RST <= x"0000";
275
276 -- Set all out bits
277     S_SET <= x"FFFF";
278
279 -- Sign vector to increase length
280     sgn <= (others => sumA(N_bits - 1));
281
282 -- Left shift
283     with ShiftingBits select
284     S_SHLN <= sumA(N_bits - 2 downto 0) & "0"         when "001",
285               sumA(N_bits - 3 downto 0) & "00"        when "010",
286               sumA(N_bits - 4 downto 0) & "000"       when "011",
287               sumA(N_bits - 5 downto 0) & "0000"      when "100",
288               sumA(N_bits - 6 downto 0) & "00000"     when "101",
289               sumA(N_bits - 7 downto 0) & "000000"    when "110",
290               sumA(N_bits - 8 downto 0) & "0000000"   when "111",
291               sumA when others;
292
293
294 -- Left arithmetic shift
295     with ShiftingBits select
296     S_SHLAN <= sgn(0) & sumA(N_bits - 3 downto 0)  & "0"         when "001",
297                sgn(0) & sumA(N_bits - 4 downto 0)  & "00"        when "010",
298                sgn(0) & sumA(N_bits - 5 downto 0)  & "000"       when "011",
299                sgn(0) & sumA(N_bits - 6 downto 0)  & "0000"      when "100",
300                sgn(0) & sumA(N_bits - 7 downto 0)  & "00000"     when "101",
301                sgn(0) & sumA(N_bits - 8 downto 0)  & "000000"    when "110",
302                sgn(0) & sumA(N_bits - 9 downto 0)  & "0000000"   when "111",
303                sumA when others;
304
305 -- Right shift
306     with ShiftingBits select
307     S_SHRN <= "0"         & sumA((N_bits - 1) downto 1) when "001",
308               "00"        & sumA((N_bits - 1) downto 2) when "010",
309               "000"       & sumA((N_bits - 1) downto 3) when "011",
310               "0000"      & sumA((N_bits - 1) downto 4) when "100",
311               "00000"     & sumA((N_bits - 1) downto 5) when "101",
312               "000000"    & sumA((N_bits - 1) downto 6) when "110",
313               "0000000"   & sumA((N_bits - 1) downto 7) when "111",
314               sumA when others;
315
316
317 S_SHRANround: process (ShiftingBits, sumA, sgn)
318  variable temp:  std_logic_vector (15 downto 0):=( others => '0');
319  begin
320  case ShiftingBits is
321            when "001" => temp :=sgn(1 downto 0)  & sumA((N_bits - 2) downto 1);
322               when "010" => temp :=sgn(2 downto 0)  & sumA((N_bits - 2) downto 2);
323               when "011" =>temp := sgn(3 downto 0)  & sumA((N_bits - 2) downto 3);
```

```
324                 when "100" =>temp := sgn(4 downto 0)  & sumA((N_bits - 2) downto 4);
325                 when "101" =>temp := sgn(5 downto 0)  & sumA((N_bits - 2) downto 5) ;
326                 when "110" =>temp := sgn(6 downto 0)  & sumA((N_bits - 2) downto 6) ;
327                 when "111" =>temp := sgn(7 downto 0)  & sumA((N_bits - 2) downto 7) ;
328                 when others => temp := sumA;
329 end case;
330
331 --rounding
332 if( sgn(0)='0') then
333  if ( conv_integer(unsigned(ShiftingBits)) > 0) then
334   if ( SumA(conv_integer(unsigned(ShiftingBits)) -1) = '1') then
335      temp:= temp + 1 ;
336   else null;
337   end if;
338  else null;
339  end if;
340 else null;
341 end if;
342 S_SHRAN<= temp;
343 end process;
344
345
346
347 -- Bit set
348    with SetClr_Bit select
349    S_BITSET <= sumA(N_bits - 1 downto 1)  & '1' when x"0",
350                sumA(N_bits - 1 downto 2)  & '1' & sumA(0) when x"1",
351                sumA(N_bits - 1 downto 3)  & '1' & sumA (N_bits - 15 downto 0) when x"
    2",
352                sumA(N_bits - 1 downto 4)  & '1' & sumA (N_bits - 14 downto 0) when x"
    3",
353                sumA(N_bits - 1 downto 5)  & '1' & sumA (N_bits - 13 downto 0) when x"
    4",
354                sumA(N_bits - 1 downto 6)  & '1' & sumA (N_bits - 12 downto 0) when x"
    5",
355                sumA(N_bits - 1 downto 7)  & '1' & sumA (N_bits - 11 downto 0) when x"
    6",
356                sumA(N_bits - 1 downto 8)  & '1' & sumA (N_bits - 10 downto 0) when x"
    7",
357                sumA(N_bits - 1 downto 9)  & '1' & sumA (N_bits - 9 downto 0)  when x"
    8",
358                sumA(N_bits - 1 downto 10) & '1' & sumA (N_bits - 8 downto 0)  when x"
    9",
359                sumA(N_bits - 1 downto 11) & '1' & sumA (N_bits - 7 downto 0)  when x"
    A",
360                sumA(N_bits - 1 downto 12) & '1' & sumA (N_bits - 6 downto 0)  when x"
    B",
361                sumA(N_bits - 1 downto 13) & '1' & sumA (N_bits - 5 downto 0)  when x"
    C",
362                sumA(N_bits - 1 downto 14) & '1' & sumA (N_bits - 4 downto 0)  when x"
    D",
363                sumA(N_bits - 1) & '1' & sumA (N_bits - 3 downto 0) when x"E",
364                '1' & sumA(N_bits - 2 downto 0) when x"F",
365                sumA when others;
366
367 -- Bit clear
368     with SetClr_Bit select
369    S_BITCLR <= sumA(N_bits - 1 downto 1)  & '0' when x"0",
```

```
370                 sumA(N_bits - 1 downto 2)  & '0' & sumA(0) when x"1",
371                 sumA(N_bits - 1 downto 3)  & '0' & sumA (N_bits - 15 downto 0) when x"
    2",
372                 sumA(N_bits - 1 downto 4)  & '0' & sumA (N_bits - 14 downto 0) when x"
    3",
373                 sumA(N_bits - 1 downto 5)  & '0' & sumA (N_bits - 13 downto 0) when x"
    4",
374                 sumA(N_bits - 1 downto 6)  & '0' & sumA (N_bits - 12 downto 0) when x"
    5",
375                 sumA(N_bits - 1 downto 7)  & '0' & sumA (N_bits - 11 downto 0) when x"
    6",
376                 sumA(N_bits - 1 downto 8)  & '0' & sumA (N_bits - 10 downto 0) when x"
    7",
377                 sumA(N_bits - 1 downto 9)  & '0' & sumA (N_bits - 9 downto 0)  when x"
    8",
378                 sumA(N_bits - 1 downto 10) & '0' & sumA (N_bits - 8 downto 0)  when x"
    9",
379                 sumA(N_bits - 1 downto 11) & '0' & sumA (N_bits - 7 downto 0)  when x"
    A",
380                 sumA(N_bits - 1 downto 12) & '0' & sumA (N_bits - 6 downto 0)  when x"
    B",
381                 sumA(N_bits - 1 downto 13) & '0' & sumA (N_bits - 5 downto 0)  when x"
    C",
382                 sumA(N_bits - 1 downto 14) & '0' & sumA (N_bits - 4 downto 0)  when x"
    D",
383                 sumA(N_bits - 1) & '0' & sumA (N_bits - 3 downto 0) when x"E",
384                 '0' & sumA(N_bits - 2 downto 0) when x"F",
385                 sumA when others;
386
387 -- Left circular shift
388     S_RTL <= sumA(N_bits - 2 downto 0) & sumA(N_bits - 1);
389
390 -- Right circular shift
391     S_RTR <= sumA(0) & sumA(N_bits - 1 downto 1);
392
393 -- Multiplication registers
394
395     sumA_reg_process : process
396     begin
397         wait until clk'event and clk = '1';
398         if (reset = '1') then
399             sumA_reg <= (others => '0');
400         elsif (signal_MUL = '1') then
401             sumA_reg <= sumA;
402         end if;
403     end process;
404
405     sumB_reg_process : process
406     begin
407         wait until clk'event and clk = '1';
408         if (reset = '1') then
409             sumB_reg <= (others => '0');
410         elsif (signal_MUL = '1') then
411             sumB_reg <= sumB;
412         end if;
413     end process;
414
415 -- Arithmetic operations
```

```
416     res_aux_add <= ( ('0' & sumA) + ('0' & sumB) );
417     res_aux_sub <= ( ('0' & sumA) - ('0' & sumB) );
418     res_inc_dec(15 downto 0) <=  res_aux_add(15 downto 0);
419  res_MUL <= sumA_reg * sumB_reg;
420  res_MULS <= signed(sumA_reg) * signed(sumB_reg);
421     r_MULS <= res_MULS(15 downto 0) & res_MULS(31 downto 16);
422     r_MUL <= res_MUL(15 downto 0) & res_MUL(31 downto 16);
423     max_res_add <= '1' when (((NOT sumA(15)) AND (NOT sumB(15))) AND ((NOT res_aux_add
        (16)) AND       res_aux_add(15))) = '1' else
424                    '0';
425     max_res_sub <= '1' when (((NOT sumA(15)) AND       sumB(15) ) AND (      res_aux_sub
        (16)  AND       res_aux_sub(15))) = '1' else
426                    '0';
427     min_res_add <= '1' when ((      sumA(15)  AND       sumB(15) ) AND (      res_aux_add
        (16)  AND (NOT res_aux_add(15)))) = '1' else
428                    '0';
429     min_res_sub <= '1' when ((      sumA(15)  AND (NOT sumB(15))) AND ((NOT res_aux_sub
        (16)) AND (NOT res_aux_sub(15)))) = '1' else
430                    '0';
431
432     arithmetic: process (signal_ADD, signal_ADDU, signal_SUB, res_aux_add, res_aux_sub
        , max_res_add, min_res_add, max_res_sub, min_res_sub)
433     begin
434
435         res <= (others => '0');
436
437  if (signal_ADDU = '1') then
438
439             res(15 downto 0) <= res_aux_add(15 downto 0);
440
441         elsif (signal_ADD = '1') then
442
443             res(15 downto 0) <= res_aux_add(15 downto 0);
444             if (max_res_add = '1') then
445                 res(15 downto 0) <= x"7FFF";
446             elsif (min_res_add = '1') then
447                 res(15 downto 0) <= x"8000";
448             end if;
449
450         elsif (signal_SUB = '1') then
451
452             res(15 downto 0) <= res_aux_sub(15 downto 0);
453             if (max_res_sub = '1') then
454                 res(15 downto 0) <= x"7FFF";
455             elsif (min_res_sub = '1') then
456                 res(15 downto 0) <= x"8000";
457             end if;
458
459         end if;
460     end process;
461
462
463  -- Logic AND operation
464     S_AND <= sumA AND sumB;
465
466  -- Logic OR operation
467     S_OR <= sumA OR sumB;
468
```

```
469  -- Logic XOR and INV operation
470      S_XOR <= sumA XOR sumB; -- Do both XOR and INV operations
471
472 end behavioral;
```

APPENDIX N

Network with STDP assembly program

```
1  ; GOTO CODE
2  ;
3  ; Izhikevic model for 16 fully connected neurons network
4  ; DEFAULT operation without virtual layers
5  ; REMOVE: 'semicolon%VIRT ' for operation with virtual layers
6
7  ; Network definitions
8
9  ;define virtual_layers 3 ; from 0 up to 7 (1 to 8 layers)
10 define virtual_layers 0 ; From 0 up to 7
11 define gsynapses 2 ; Up to 32 global synapses
12 define n_step 1;
13
14 .DATA
15
16 ; Virtual layers
17 V0 = "0000000F" ; Number of assigned synapses (s-1) to the main layer
18 V1 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 1
19 V2 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 2
20 V3 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 3
21 V4 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 4
22 V5 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 5
23 V6 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 6
24 V7 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 7
25 ;VLAYERS="00000003" ; Number of virtual layers.
26 VLAYERS="00000000" ; Number of virtual layers (n-1).
27
28 ; Membrane potential parameters common to all neurons
29
30 ;multiple of 10uV
31 VTHRES="FFFFF63C"  ; Threshold voltage -25 mV
32 N70="00001B58"; 70mV,
33 N0002="000068DC";
34 N001="00000028F"; 0.01, The noise generated by LFSR is 100 time bigger than necessary.
35 REST_POT="FFFFE69C"; -6500
36 N5="00000005";
37
38 ;____Randomized membrane value written in memory
39 ;IZH_B= 0.2
40 ;IZH_D= 4mV
41 ;IZH_C= -65mV resting potential
```

```
42 ;IZH_A= "0000051E"= 0.02
43 ;
44 ;
45 ; Neural and Synaptic RAM addresses
46 SYN_ADDR0="00000000"    ; First address of Synaptic parameters in SNRAM for V = 0.
47 SYN_ADDR1="00000020"    ; First address of Synaptic parameters in SNRAM for V = 1.
48 SYN_ADDR2="00000040"    ; First address of Synaptic parameters in SNRAM for V = 2.
49 SYN_ADDR3="00000060"    ; First address of Synaptic parameters in SNRAM for V = 3.
50 SYN_ADDR4="00000080"    ; First address of Synaptic parameters in SNRAM for V = 4.
51 SYN_ADDR5="000000A0"    ; First address of Synaptic parameters in SNRAM for V = 5.
52 SYN_ADDR6="000000C0"    ; First address of Synaptic parameters in SNRAM for V = 6.
53 SYN_ADDR7="000000E0"    ; First address of Synaptic parameters in SNRAM for V = 7.
54 GSYN_ADDR="00000090"    ; First address of Global Synaptic parameters in SNRAM.
55 NEU_ADDR0="000003E3"    ; First address of Neural parameters in SNRAM (995) for V = 0.
56 NEU_ADDR1="000003E7"    ; First address of Neural parameters in SNRAM (999) for V = 1.
57 NEU_ADDR2="000003EB"    ; First address of Neural parameters in SNRAM () for V = 2.
58 NEU_ADDR3="000003EF"    ; First address of Neural parameters in SNRAM () for V = 3.
59 NEU_ADDR4="000003F3"    ; First address of Neural parameters in SNRAM () for V = 4.
60 NEU_ADDR5="000003F7"    ; First address of Neural parameters in SNRAM () for V = 5.
61 NEU_ADDR6="000003FB"    ; First address of Neural parameters in SNRAM () for V = 6.
62 NEU_ADDR7="000003FF"    ; First address of Neural parameters in SNRAM () for V = 7.
63
64
65 SEED_ADDRL = "000003FD" ; Address of noise seed in SNRAM
66 SEED_ADDRH = "000003FE";Address of noise seed in SNRAM
67 PEID  = "000003FF" ; Address of  PE Identifier number
68
69 ; General constants
70 K_syn= "0000F99A"; (40ms-1ms/40)
71 K_act= "0000FFFA"; (11000-1/11000)
72 ;N012= "00000333"; 1/80
73 N025= "00000666"; 1/40=0.025 to be multiplied to sum of weights
74 ;N05=    "00000CCC"; 1/20
75 L_MASK_LS= "0000007F"; 7 bit mask
76 MMAX= "00000800"; 1024/2=2^9 to be left shifted
77
78
79 ;DEBUG
80 ;NOISE=  "00000500"; noise for test is 10mV, 1280
81 ;add1= "0000D511"
82 ;add2= "00000789"
83
84 .CODE
85 ;
86 GOTO MAIN ; Jump to main program
87 ;
88 ; ************************** PROCEDURES BEGIN **************************
89 ;
90 .RANDOM_INIT  ; Uses R0 and R1
91  LOADBP SEED_ADDRL
92  LOADSN
93  SEED
94  LOADBP SEED_ADDRH
95  LOADSN
96  SEED    ;
97 RET
98 ;
99 .LOAD_NEURON    ; Uses R0, R1, R2, R3, R5
```

```
100  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
101  LOADBP    ; SNRAM pointer to currently processed neuron
102  LOADSN    ; Load Neural parameters from SNRAM to R1<=u & ACC<=Vmem
103  STORESP           ; FAKE STORE ONLY TO MAKE PC+1
104  MOVR R2   ; Move Vmem from ACC to R2
105  MOVA R1   ; ACC<=u
106  MOVR R3   ; put r3<=u
107  LOADSN            ; load ACC<=d; R1<=b
108  STORESP           ; FAKE STORE ONLY TO MAKE PC+1
109  MOVR R5   ; R5<=d
110  SWAPS R5  ; SW5<=d
111  MOVA R1   ; ACC<=R1<=b;
112  MOVR R5   ; R5<=b
113  LOADSN
114  STORESP
115  MOVR R6   ; R6<=C
116  SWAPS R6  ; SW6<=C
117  MOVA R1
118  MOVR R6   ; R6<=A
119  LOADSN
120  MOVR R4   ; R4<=Mi
121  MOVSR R4  ; SW4<=Mi
122  MARK
123 RET
124 ;
125 .DETECT_SPIKE ; Uses R0,R3,SW5, SW6 and R2
126  LDALL ACC VTHRES
127  SUB R2    ;Compare Vth - Vmem
128  SHLN 1    ;subtraction sign to C flag
129  FREEZEC ; if doesn't spiked do:
130   SWAPS R5
131   RST R5
132   RST ACC; DA COMMENTARE
133  UNFREEZE
134  FREEZENC  ;If spiked, do:
135   ;REMEMBER: C stays in SW6.
136   SWAPS R6
137   MOVA R6
138   SWAPS R6
139   MOVR R2
140   MOVA R3   ;ACC<= u
141   ;REMEMBER: d stays in SW5.
142   SWAPS R5  ;R5<=D
143   ADD R5    ;ACC<= u+d
144   MOVR R3   ;u<= u+d
145   RST R0
146   INC   ; 1 in ACC
147   SHLN 5
148   SHLN 5
149   MOVR R5   ; SET 11th bit of SW5 in order to stress the presence of spike
150   SET ACC
151  UNFREEZE
152  STOREPS    ; Push spikes
153 RET
154 ;
155 .STDP_SYNAPSE_CALC
156  LOADSP    ; ACC<=Mj&&Sj(t), R1<=P
157  MOVSR R1  ; SW1<=P
```

```
158 MOVSR R0   ; Sw0<=Mj&&Sj(t)
159 SHRN 1     ; ACC<=Mj
160 MUL R5     ; ACC<=Mj*Si
161 NOP
162 MOVA R1
163 MOVR R7    ; R7<=PARTIAL<=  Mj*Si(t)!!
164 MOVRS R0   ; ACC<=Mj&&Sj(t)
165 BITCLR 0
166 LDALL R4 K_syn
167 MUL R4     ; ACC<= M*Ksyn=M(t+1)&Sj
168 ; the stored value of Sj is don't care, it will be took by lcl_mem
169 SWAPS R0   ; ACC<=Mj&&Sj(t), SW0<= M*Ksyn= M(t+1)
170 MOVR R1;
171 SHRN 1
172 FREEZENC   ; if there is spike
173  SWAPS ACC ; ACC<= M*Ksyn= M(t+1), SW0<=Mj&Sj(t) =Mj(t)
174  RST ACC   ; replace M*Ksyn with 2**11
175  INC
176  SHLN 7
177  SHLN 4    ; M=2**11
178  SWAPS ACC ; ACC<= Mj ,SW0<= 2**11= M(t+1)
179  MOVRS R4  ; R4<=Mi=SW4
180  MOVA R7   ; ACC<= Partial
181  SUB R4;   ; ACC<= PARTIAL <= Si(t)*Mj(t)-Sj(t)*Mi(t)
182  MOVR R7   ; R7<= PARTIAL
183 UNFREEZE
184 MOVA R1    ; ACC<= Mj&Sj ,SW0<= 2**11= M(t+1)
185 MOVRS R1   ; R1 <= P
186 SWAPS R0   ; ACC<= M(t+1), SW0<= Mj&Sj,
187 STORESP    ; MEM<=Mj&&Sj(t+1), MEM<=P
188 NOP
189 LOADSN    ; ACC<= L(15 downto 0), R1<= L(31 downto 16)
190 SWAPS R7  ; SW7<= PARTIAL
191 MOVR R7   ; R7 <= L(15 downto 0)
192 MOVRS ACC ; ACC<= Mj&Sj
193 SHRN 1
194 FREEZENC  ; se c'   lo spike
195  MOVA R1  ; ACC<= L(31 downto 16)= [L]*2^10
196  MOVR R4
197  SHRN 5;
198  SHRAN 5;
199  MOVRS R1; R1<=P
200  MULS R1  ; L*P
201  MOVA R1  ;
202  ADD R3   ; ADD SUM OF WEIGHTS
203  MOVR R3  ; STORE SUM OF WEIGHTS
204  MOVA R4
205  MOVR R1  ; R1<=L(31 downto 16)
206 UNFREEZE
207 LDALL R4 K_act   ; --------------------------
208 MOVA R1    ; ACC<= L(31 downto 16)
209 MUL R4;
210 MOVSR ACC ; SW0<= MSW of MSW
211 MOVSR R1  ; SW1<= LSW of MSW
212 MOVA R7   ; ACC<=LSW of L
213 MUL R4    ; ACC<= MSW of LSW), R1 unuseful
214 RST R4
215 MOVRS R1  ; R1<=LSW(MSW)
```

```
216  ADDU R1; ADD CHANGED IN ADDU NEW!!!
217  FREEZENC ; overflow
218   SET R4 ; R4<=-1 1741006799 ps
219  UNFREEZE
220  MOVR R7   ; R7<= L(15 downto 0)(t+1)
221  MOVRS ACC   ; ACC<= MSW(MSW)
222  SUB R4    ; ACC<= L(31 downto 16)(t+1)
223  ;RST R4
224  MOVR R1   ; R1<=L*2^10=
225  SWAPS R7 ; R7<= PARTIAL, SW7<=L(15 downto 0)(t+1)
226  MOVA R7          ; in this way ACC<= PARTIAL = M * 2^10
227  ADD R1
228  MOVR R1   ; R1<=L(31 downto 16)(t+1)
229  SWAPS R7 ;R7<=L(15 downto 0)(t+1)
230  MOVA R7
231  STORESP
232 RET
233 ;
234 .Mi
235  MOVRS R4   ; R4<= Mi
236  LDALL R7 MMAX
237  MOVA R5
238  MUL R7
239  MOVA R1
240  SHRAN 1; NOTE THAT MMAX is MiMAX*2 in order to be shifted
241  FREEZENZ
242   LDALL ACC K_syn
243   MUL R4
244  UNFREEZE
245  MOVR R4
246  MOVSR R4   ;SW4<= Mi
247 RET
248 ;
249 .GAUSS_NOISE ; Uses SW0, R2, R6 and R7
250  RANDON                    ;LFSR ON
251  RANDOFF                   ;LFSR OFF. Arbitrarily heres
252  LLFSR     ;Noise seeds to ACC, R1, SR0, SR1
253  ADD R1
254  SHRAN 1
255  MOVR R7
256  SWAPS R0
257  SWAPS R1
258  ADD R1
259  SHRAN 1
260  ADD R7
261  LDALL R7 N001; ACC=OUTPUT
262  MULS R7   ;NOISE IN ACC
263  FREEZENC
264   INC
265  UNFREEZE
266  MOVR R4   ;NOISE IN R4
267  SWAPS R6
268  MOVA R6
269  SWAPS R6
270  LDALL R7 REST_POT ; tipical value of resting potential to be compared with the actual
         one
271  SUB R7   ; compare it with the value of the parameter. if =, it is excitatory
272  MOVA R4
```

```
273         FREEZEZ
274    LDALL R7 N5;
275    MUL R7
276    NOP
277    MOVA R1
278    SHRAN 1
279   UNFREEZE
280   MOVSR ACC                    ;TO STORE 1/2 THE NOISE
281  RET
282  ;
283  .MEMBRANE_POTENTIAL ;Uses R0,R4,R7   32808
284   MOVA R2
285   MULS R0   ; v^2*2^12 (CHANGED MUL IN MULS!!)
286   NOP   ; Check if needed
287    ;Shift R0R1 4 positions left
288   SHLN 4   ; Shift Accumulator 2^4
289   MOVR R4
290   MOVA R1   ; Move LS part (R1) to R0 (2^16)
291   SHRN 4
292   SHRAN 4
293   SHRAN 4   ; 2^16/2^12 = 2^4
294   ADD R4   ; Combine and obtain v^2/2^12
295   LDALL R4 N0002 ; 0.0002*2^27 is in R4
296   MULS R4   ; v^2*2^(-12)*0.0002*2^27/2^16 = 0.0002*v^2*2^(-1)
297   NOP   ; Check if needed
298   SHLN 1   ; Shift Accumulator 2^1
299   MOVR R4
300   MOVA R1   ; Move LS part (R1) to R0 (2^16)
301   SHRN 5
302   SHRAN 5
303   SHRAN 5   ; 2^16/2^15 = 2^1
304   ADD R4   ; Combine and obtain 0.0002*v^2
305   MOVR R7;
306   MOVA R2;          ACC<=Vinit
307   SHRAN 2;          ACC<=0.25*Vinit;
308   ADD R2;   ACC<=ACC+Vinit=1.25*Vinit
309   SHLAN 1
310   ADD R7
311   LDALL R4 N70;     R4<=70
312   ADD R4;
313   MOVR R7
314   RST ACC
315   SUB R3;     ACC=- u
316   SHRAN 1;
317   ADD R7;
318   ADD R2;   ACC=ACC+Vinit
319   MOVR R2;   Back to R2 where membrane potential is stored
320  RET
321  ;
322  .RECOVERY_UPDATE ;uses R3,R5,R6
323   MOVA R2;           ACC<=Vinit
324   SWAPS R5;
325   MULS R5                    ;ACC<=R5*ACC=B*Vinit
326   FREEZENC
327    INC
328   UNFREEZE
329   SUB R3                     ;ACC<= ACC-R3= ACC-Uinit
330    ;REMEMBER: A is in R6
```

```
331  MULS R6;                    ACC<=A*ACC;
332  FREEZENC
333    INC
334  UNFREEZE
335  ADD R3;                     ACC<=ACC+Uinit
336  MOVR R3;                    Back to R3 where recovery value is stored
337 RET
338 ;
339 .SUM_NOISE_AND_Ws
340  MOVRS ACC                    ;NOISE TO ACC
341  ADD R2     ;ADD NOISE TO SIGNAL
342  MOVR R2
343  SWAPS R3    ;SYN. contribute
344  LDALL R0 N025 ; 1/40
345  MULS R3
346  ADD R2;
347  MOVR R2;    store membrane potential
348  SWAPS R3;
349 RET
350 ;
351 .STORE_NEURON    ;uses R0,R3 and R1
352  MOVA R3   ;move u from R3 to acc
353  MOVR R1   ;move u from ACC to R1
354  MOVA R2   ; Move Vmem from R2 to ACC
355  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
356  LOADBP    ; SNRAM pointer to currently processed neuron
357  STORESP   ; Store u&Vmem to SNRAM
358  NOP
359  LOADSN
360  NOP
361  STORESP   ;store second row
362  NOP
363  LOADSN
364  NOP
365  STORESP   ;store third row
366  MOVRS R4;
367  MOVA R4;
368  RST R1
369  STORESP
370 RET
371 ;
372 ; ************************** PROCEDURES END ****************************
373
374 ; ************************** MAIN PROGRAMME BEGIN ***********************
375 .MAIN
376 ;
377
378 ; Virtual operation init
379 LAYERV virtual_layers ; Init sequencer vlayers. It is 0 for non-virtual operation
380 LDALL ACC, VLAYERS ; Load defined virtual layers to PE array
381 SPMOV 0 ; VIRT <= ACC
382
383
384 ; Initial instructions
385 GOSUB RANDOM_INIT  ; For noise initialization
386
387 .EXEC_LOOP ; Execution loop
388
```

```
389 LOOP virtual_layers     ;Neuron loop for virtual operation
390       NOP     ;to prevent pipeline error
391       GOSUB LOAD_NEURON
392       GOSUB DETECT_SPIKE
393   SWAPS R3     ;SYNAPSE Sum will be store IN R3
394   MOVA R5
395   SHRN 5
396   SHRN 5
397   MOVR R5
398       READMPV SYN_ADDR0
399       LOADBP
400       LOOPV V0   ;synaptic loop. Reads number of current-layer synapses
401   NOP           ;to prevent pipeline error
402             GOSUB STDP_SYNAPSE_CALC
403       ENDL
404   SWAPS R3 ;SYNAPSE sum to SW3
405   GOSUB Mi
406   GOSUB GAUSS_NOISE
407       LOOP n_step
408         NOP
409   GOSUB MEMBRANE_POTENTIAL   ;Calculate membrane potential
410         GOSUB SUM_NOISE_AND_Ws
411  ENDL
412  GOSUB RECOVERY_UPDATE
413  GOSUB STORE_NEURON
414  RST R3;
415  MOVSR R3;
416  INCV
417  ENDL
418 .FINISH
419 NOP          ;Empty pipeline wait NOPs
420 NOP
421 NOP
422 SPKDIS         ;Distribute spikes
423 GOTO EXEC_LOOP        ;Execution loop
```

APPENDIX O

Network with STDP at neuron level assembly program

```
; GOTO CODE
;
; Izhikevic model for 16 fully connected neurons network
; DEFAULT operation without virtual layers
; REMOVE: 'semicolon%VIRT ' for operation with virtual layers

; Network definitions

define virtual_layers 0 ; From 0 up to 7
define gsynapses 2 ; Up to 32 global synapses
define n_step 1; Number of step -1.

.DATA

; Virtual layers
V0 = "0000000F" ; Number of assigned synapses (s-1) to the main layer
V1 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 1
V2 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 2
V3 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 3
V4 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 4
V5 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 5
V6 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 6
V7 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 7
VLAYERS="00000000" ; Number of virtual layers (n-1).

; Membrane potential parameters common to all neurons

;multiple of 10uV
VTHRES="FFFFF63C"   ; Threshold voltage -25 mV
N70="00001B58"; 70mV,
N0002="000068DC";
N001="00000028F"; 0.01, The noise generated by LFSR is 100 time bigger than necessary.
REST_POT="FFFFE69C"; -6500
N5="00000005";

;____Randomized membrane value written in memory
;IZH_B= 0.2
;IZH_D= 4mV
```

```
39 ;IZH_C= -65mV resting potential
40 ;IZH_A= "0000051E"= 0.02
41 ;
42 ;
43 ; Neural and Synaptic RAM addresses
44 SYN_ADDR0="00000000"    ; First address of Synaptic parameters in SNRAM for V = 0.
45 SYN_ADDR1="00000020"    ; First address of Synaptic parameters in SNRAM for V = 1.
46 SYN_ADDR2="00000040"    ; First address of Synaptic parameters in SNRAM for V = 2.
47 SYN_ADDR3="00000060"    ; First address of Synaptic parameters in SNRAM for V = 3.
48 SYN_ADDR4="00000080"    ; First address of Synaptic parameters in SNRAM for V = 4.
49 SYN_ADDR5="000000A0"    ; First address of Synaptic parameters in SNRAM for V = 5.
50 SYN_ADDR6="000000C0"    ; First address of Synaptic parameters in SNRAM for V = 6.
51 SYN_ADDR7="000000E0"    ; First address of Synaptic parameters in SNRAM for V = 7.
52 GSYN_ADDR="00000090"    ; First address of Global Synaptic parameters in SNRAM.
53 NEU_ADDR0="000003E3"    ; First address of Neural parameters in SNRAM (995) for V = 0.
54 NEU_ADDR1="000003E7"    ; First address of Neural parameters in SNRAM (999) for V = 1.
55 NEU_ADDR2="000003EB"    ; First address of Neural parameters in SNRAM () for V = 2.
56 NEU_ADDR3="000003EF"    ; First address of Neural parameters in SNRAM () for V = 3.
57 NEU_ADDR4="000003F3"    ; First address of Neural parameters in SNRAM () for V = 4.
58 NEU_ADDR5="000003F7"    ; First address of Neural parameters in SNRAM () for V = 5.
59 NEU_ADDR6="000003FB"    ; First address of Neural parameters in SNRAM () for V = 6.
60 NEU_ADDR7="000003FF"    ; First address of Neural parameters in SNRAM () for V = 7.
61 ;FOR STDP TEST
62 NEU_ADDR0_4="000003E6";
63
64
65 SEED_ADDRL = "000003FD" ; Address of noise seed in SNRAM
66 SEED_ADDRH = "000003FE";Address of noise seed in SNRAM
67 PEID  = "000003FF" ; Address of  PE Identifier number
68
69 ; General constants
70 K_syn= "0000F99A"; (40ms-1ms/40)
71 K_act= "0000FFFA"; (11000-1/11000)
72 N05=     "00000CCC"; 1/20
73 L_MASK_LS= "0000007F"; 7 bit mask
74 MMAX= "00001000"; 4096 to be right shifted
75
76
77 .CODE
78 ;
79 GOTO MAIN ; Jump to main program
80 ;
81 ; *************************** PROCEDURES BEGIN ***************************
82 ;
83 .RANDOM_INIT   ; Uses R0 and R1
84  LOADBP SEED_ADDRL
85  LOADSN
86  SEED
87  LOADBP SEED_ADDRH
88  LOADSN
89  SEED     ;
90 RET
91 ;
92 .LOAD_NEURON     ; Uses R0, R1, R2, R3, R5
93  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
94  LOADBP    ; SNRAM pointer to currently processed neuron
95  LOADSN   ; Load Neural parameters from SNRAM to R1<=u & ACC<=Vmem
96  STORESP           ; FAKE STORE ONLY TO MAKE PC+1
```

```
97  MOVR R2   ; Move Vmem from ACC to R2
98  MOVA R1   ; ACC<=u
99  MOVR R3   ; put r3<=u
100 LOADSN            ; load ACC<=d; R1<=b
101 STORESP           ; FAKE STORE ONLY TO MAKE PC+1
102 MOVR R5   ; R5<=d
103 SWAPS R5  ; SW5<=d
104 MOVA R1   ; ACC<=R1<=b;
105 MOVR R5   ; R5<=b
106 LOADSN
107 STORESP
108 MOVR R6   ; R6<=C
109 SWAPS R6  ; SW6<=C
110 MOVA R1
111 MOVR R6   ; R6<=A
112 LOADSN
113 MOVR R4   ; R4<=Mi
114 MOVSR R4  ; SW4<=Mi
115 MOVA R1;
116 MOVR R4;
117 MARK
118 RET
119 ;
120 .DETECT_SPIKE ; Uses R0,R3,SW5, SW6 and R2
121  LDALL ACC VTHRES
122  SUB R2    ;Compare Vth - Vmem
123  SHLN 1    ;subtraction sign to C flag
124  FREEZEC   ;if doesn't spiked do:
125   SWAPS R5
126   RST R5
127   RST ACC; DA COMMENTARE
128  UNFREEZE
129  FREEZENC ;If spiked, do:
130   ;REMEMBER: C stays in SW6.
131   SWAPS R6
132   MOVA R6
133   SWAPS R6
134   MOVR R2
135   MOVA R3   ;ACC<= u
136   ;REMEMBER: d stays in SW5.
137   SWAPS R5  ;R5<=D
138   ADD R5    ;ACC<= u+d
139   MOVR R3   ;u<= u+d
140   RST R0
141   INC   ; 1 in ACC
142   SHLN 5
143   SHLN 5
144   MOVR R5   ; SET 11th bit of SW5 in order to stress the presence of spike
145   SET ACC
146  UNFREEZE
147  STOREPS    ; Push spikes
148 RET
149 ;
150 .SYNAPSE_CALC
151 LOADSP    ; ACC<=Mj&&Sj(t), R1<=P
152 MOVSR R1   ; SW1<=P
153 MOVSR R0   ; Sw0<=Mj&&Sj(t)
154 MOVA R4; put 0 or 1 if this neuron has STDP or NOT
```

```
155 SHRN 1;
156 FREEZENC; if c=0 freeze, you must do STDP
157  MOVRS R0   ; ACC<=Mj&&Sj(t)
158  SHRN 1   ; Move spike to flag C
159       FREEZENC
160   MOVA R1 ; Synaptic parameter to ACC
161   ADD R3;
162   MOVR R3;
163          UNFREEZE
164 UNFREEZE
165 MOVA R4; put 0 or 1 if this neuron has STDP or NOT
166 SHRN 1;
167 FREEZEC; if c=1 freeze. you must not do STDP
168  MOVRS R0   ; ACC<=Mj&&Sj(t)
169  SHRN 1    ; ACC<=Mj
170  MUL R5    ; ACC<=Mj*Si
171  NOP
172  MOVA R1
173  MOVR R7    ; R7<=PARTIAL<=  Mj*Si(t)!!
174  MOVRS R0   ; ACC<=Mj&&Sj(t)
175  BITCLR 0
176  LDALL R4 K_syn
177  MUL R4    ; ACC<= M*Ksyn=M(t+1)&Sj
178  ; the stored value of Sj is don't care, it will be took by lcl_mem
179  SWAPS R0    ; ACC<=Mj&&Sj(t), SW0<= M*Ksyn= M(t+1)
180  MOVR R1;
181  SHRN 1
182  FREEZENC    ; if there is spike
183   SWAPS ACC ; ACC<= M*Ksyn= M(t+1), SW0<=Mj&Sj(t) =Mj(t)
184   RST ACC   ; replace M*Ksyn with 2*2**11
185   INC
186   SHLN 7
187   SHLN 5   ; M=(2*2**10)*2 MORE INCISIVE STDP
188   SWAPS ACC ; ACC<= Mj ,SW0<= 2*2**11= M(t+1)
189   MOVRS R4 ; R4<=Mi=SW4
190   MOVA R7   ; ACC<= PARTIAL
191   SUB R4;   ; ACC<= PARTIAL <= Si(t)*Mj(t)-Sj(t)*Mi(t)
192   MOVR R7   ; R7<= PARTIAL
193  UNFREEZE
194  MOVA R1   ;
195  MOVRS R1 ; R1 <= P
196  SWAPS R0 ; ACC<= M(t+1), SW0<= Mj&Sj,
197  STORESP   ; MEM<=Mj&&Sj(t+1), MEM<=P
198  NOP
199  LOADSN   ; ACC<= L(15 downto 0), R1<= L(31 downto 16)
200  SWAPS R7 ; SW7<= PARTIAL
201  MOVR R7   ; R7 <= L(15 downto 0)
202  MOVRS ACC ; ACC<= Mj&Sj
203  SHRN 1
204  FREEZENC   ; se c'   lo spike
205   MOVA R1 ; ACC<= L(31 downto 16)= [L]*2^10=20*2^10
206   MOVR R4
207   SHRN 5;
208   SHRAN 5;
209   MOVRS R1; R1<=P
210   MULS R1 ;
211   MOVA R1 ;
212   ADD R3   ; ADD SUM OF WEIGHTS
```

```
213   MOVR R3 ; STORE SUM OF WEIGHTS
214   MOVA R4
215   MOVR R1 ; R1<=L(31 downto 16)
216  UNFREEZE
217  LDALL R4 K_act   ; --------------------------
218  MOVA R1   ; ACC<= L(31 downto 16)
219  MUL R4;
220  MOVSR ACC ; SW0<= MSW of MSW
221  MOVSR R1 ; SW1<= LSW of MSW
222  MOVA R7   ; ACC<=LSW of L
223  MUL R4   ; ACC<= MSW of LSW), R1 unuseful
224  RST R4
225  MOVRS R1 ; R1<=LSW(MSW)
226  ADDU R1;
227  FREEZENC ; overflow
228   SET R4 ; R4<=-1 1741006799 ps
229  UNFREEZE
230  MOVR R7   ; R7<= L(15 downto 0)(t+1)
231  MOVRS ACC   ; ACC<= MSW(MSW)
232  SUB R4    ; ACC<= L(31 downto 16)(t+1)
233  ;RST R4
234  MOVR R1   ; R1<=L*2^10=
235  SWAPS R7 ; R7<= PARTIAL, SW7<=L(15 downto 0)(t+1) 2537731149 ps
236  MOVA R7          ; in this way ACC<= PARTIAL = M * 2^10
237  ADD R1
238  MOVR R1   ; R1<=L(31 downto 16)(t+1)
239  SWAPS R7 ;R7<=L(15 downto 0)(t+1)
240  MOVA R7
241  STORESP
242 UNFREEZE
243 RET
244 ;
245 .Mi
246  MOVRS R4   ; R4<= Mi
247  LDALL R7 MMAX
248  MOVA R5
249  MUL R7
250  MOVA R1
251  SHRAN 1; NOTE THAT MMAX is MiMAX*2 in order to be shifted
252  FREEZENZ
253   LDALL ACC K_syn
254   MUL R4
255  UNFREEZE
256  MOVR R4
257  MOVSR R4   ;SW4<= Mi
258 RET
259 ;
260 .GAUSS_NOISE ; Uses SW0, R2, R6 and R7
261  RANDON                    ;LFSR ON
262  RANDOFF                   ;LFSR OFF. Arbitrarily heres
263  LLFSR     ;Noise seeds to ACC, R1, SR0, SR1
264  ADD R1
265  SHRAN 1
266  MOVR R7
267  SWAPS R0
268  SWAPS R1
269  ADD R1
270  SHRAN 1
```

```
271  ADD R7
272  LDALL R7 N001; ACC=OUTPUT
273  MULS R7   ;NOISE IN ACC
274  FREEZENC
275   INC
276  UNFREEZE
277  MOVR R4   ;NOISE IN R4
278  SWAPS R6
279  MOVA R6
280  SWAPS R6
281  LDALL R7 REST_POT; tipical value of resting potential to be compared with the actual
       one
282  SUB R7   ; compare it with the value of the parameter. if =, it is excitatory
283  MOVA R4
284       FREEZEZ
285   LDALL R7 N5;
286   MUL R7
287   NOP
288   MOVA R1
289   SHRAN 1
290  UNFREEZE
291  MOVSR ACC                ;TO STORE 1/2 THE NOISE
292 RET
293 ;
294 .MEMBRANE_POTENTIAL ;Uses R0,R4,R7
295  MOVA R2
296  MULS R0   ; v^2*2^12
297  NOP   ;
298  ;Shift R0R1 4 positions left
299  SHLN 4   ; Shift Accumulator 2^4
300  MOVR R4
301  MOVA R1   ; Move LS part (R1) to R0 (2^16)
302  SHRN 4
303  SHRAN 4
304  SHRAN 4   ; 2^16/2^12 = 2^4
305  ADD R4   ; Combine and obtain v^2/2^12
306  LDALL R4 N0002 ; 0.0002*2^27 is in R4
307  MULS R4   ; v^2*2^(-12)*0.0002*2^27/2^16 = 0.0002*v^2*2^(-1)
308  NOP   ;
309  SHLN 1   ; Shift Accumulator 2^1
310  MOVR R4
311  MOVA R1   ; Move LS part (R1) to R0 (2^16)
312  SHRN 5
313  SHRAN 5
314  SHRAN 5   ; 2^16/2^15 = 2^1
315  ADD R4   ; Combine and obtain 0.0002*v^2
316  MOVR R7;
317  MOVA R2;          ACC<=Vinit
318  SHRAN 2;          ACC<=0.25*Vinit;
319  ADD R2;   ACC<=ACC+Vinit=1.25*Vinit
320  SHLAN 1
321  ADD R7
322  LDALL R4 N70;     R4<=70
323  ADD R4;
324  MOVR R7
325  RST ACC
326  SUB R3;    ACC=- u
327  SHRAN 1;
```

```
328  ADD R7;
329  ADD R2;   ACC=ACC+Vinit
330  MOVR R2;   Back to R2 where membrane potential is stored
331 RET
332 ;
333 .RECOVERY_UPDATE ;uses R3,R5,R6
334  MOVA R2;           ACC<=Vinit
335  SWAPS R5;
336  MULS R5                    ;ACC<=R5*ACC=B*Vinit
337  FREEZENC
338    INC
339  UNFREEZE
340  SUB R3                     ;ACC<= ACC-R3= ACC-Uinit
341  ;REMEMBER: A is in R6
342  MULS R6;                    ACC<=A*ACC;
343  FREEZENC
344    INC
345  UNFREEZE
346  ADD R3;                     ACC<=ACC+Uinit
347  MOVR R3;                    Back to R3 where recovery value is stored
348 RET
349 ;
350 .SUM_NOISE_AND_Ws
351  MOVRS ACC                    ;NOISE TO ACC
352  ADD R2     ;ADD NOISE TO SIGNAL
353  MOVR R2
354  SWAPS R3    ;SYN. contribute
355  MOVA R3
356  ADD R2;
357  MOVR R2;    store membrane potential
358  SWAPS R3;
359 RET
360 ;
361 .STORE_NEURON    ;uses R0,R3 and R1
362  MOVA R3   ;move u from R3 to acc
363  MOVR R1   ;move u from ACC to R1
364  MOVA R2   ; Move Vmem from R2 to ACC
365  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
366  LOADBP    ; SNRAM pointer to currently processed neuron
367  STORESP   ; Store u&Vmem to SNRAM
368  NOP
369  LOADSN
370  NOP
371  STORESP   ;store second row
372  NOP
373  LOADSN
374  NOP
375  STORESP   ;store third row
376  NOP
377  LOADSN
378  MOVRS R4;
379  MOVA R4;
380  STORESP
381 RET
382 ;
383 ; ************************** PROCEDURES END ****************************
384
385 ; ************************** MAIN PROGRAMME BEGIN ***********************
```

```
.MAIN
;

; Virtual operation init
LAYERV virtual_layers ; Init sequencer vlayers. It is 0 for non-virtual operation
LDALL ACC, VLAYERS ; Load defined virtual layers to PE array
SPMOV 0 ; VIRT <= ACC


; Initial instructions
GOSUB RANDOM_INIT  ; For noise initialization

.EXEC_LOOP ; Execution loop

 LOOP virtual_layers     ;Neuron loop for virtual operation
      NOP     ;to prevent pipeline error
      GOSUB LOAD_NEURON
      GOSUB DETECT_SPIKE
 SWAPS R3     ;SYNAPSE Sum will be store IN R3
 MOVA R5
 SHRN 5
 SHRN 5
 MOVR R5; put 0 o 1 in R5 if there is a spike or not
 READMPV SYN_ADDR0
      LOADBP
          LOOPV V0  ;synaptic loop. Reads number of current-layer synapses
  NOP          ;to prevent pipeline error
           GOSUB SYNAPSE_CALC
      ENDL
 MOVA R3     ;move to acc synapse sum
    SHRAN 1; save half the weight
      MOVR R3    ;move to R3
 READMPV NEU_ADDR0_4 ; Address of real neuron + virt (valid also for non-virtual)
 LOADBP    ; SNRAM pointer to currently processed neuron
 LOADSN  ; ACC<=Mi ;R1<=0 o 1 if must be done STDP or not
 MOVA R1
 SHRN 1;
 FREEZEC; if 0 you have to perform STDP
  LDALL R0 N05 ; 1/20
  MULS R3
  MOVR R3
  GOSUB Mi
 UNFREEZE
 SWAPS R3 ;SYNAPSE sum to SW3
 GOSUB GAUSS_NOISE
      LOOP n_step
       NOP
  GOSUB MEMBRANE_POTENTIAL  ;Calculate membrane potential
       GOSUB SUM_NOISE_AND_Ws
 ENDL
 GOSUB RECOVERY_UPDATE
 GOSUB STORE_NEURON
 RST R3;
 MOVSR R3;
 INCV
 ENDL
.FINISH
NOP         ;Empty pipeline wait NOPs
```

```
444 NOP
445 NOP
446 SPKDIS          ; Distribute spikes
447 GOTO EXEC_LOOP        ; Execution loop
```

APPENDIX P

Network with STDP at connection level assembly program

```
1  ; GOTO CODE
2  ;
3  ; Izhikevic model for 16 fully connected neurons network
4  ; DEFAULT operation without virtual layers
5  ; REMOVE: 'semicolon%VIRT ' for operation with virtual layers
6
7  ; Network definitions
8
9  define virtual_layers 0 ; From 0 up to 7
10 define gsynapses 2 ; Up to 32 global synapses
11 define n_step 1; Number of step -1.
12
13 .DATA
14
15 ; Virtual layers
16 V0 = "0000000F" ; Number of assigned synapses (s-1) to the main layer
17 V1 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 1
18 V2 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 2
19 V3 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 3
20 V4 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 4
21 V5 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 5
22 V6 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 6
23 V7 = "0000000F" ; Number of assigned synapses (s-1) to virtual layer 7
24 VLAYERS="00000000" ; Number of virtual layers (n-1).
25
26 ; Membrane potential parameters common to all neurons
27
28 ;multiple of 10uV
29 VTHRES="FFFFF63C"   ; Threshold voltage -25 mV
30 N70="00001B58" ; 70mV,
31 N0002="000068DC" ;
32 N001="00000028F" ; 0.01, The noise generated by LFSR is 100 time bigger than necessary.
33 REST_POT="FFFFE69C" ; -6500
34 N5="00000005" ;
35
36 ;____Randomized membrane value written in memory
37 ;IZH_B= 0.2
38 ;IZH_D= 4mV
```

```
39 ;IZH_C= -65mV resting potential
40 ;IZH_A= "0000051E"= 0.02
41 ;
42 ;
43 ; Neural and Synaptic RAM addresses
44 SYN_ADDR0="00000000"    ; First address of Synaptic parameters in SNRAM for V = 0.
45 SYN_ADDR1="00000020"    ; First address of Synaptic parameters in SNRAM for V = 1.
46 SYN_ADDR2="00000040"    ; First address of Synaptic parameters in SNRAM for V = 2.
47 SYN_ADDR3="00000060"    ; First address of Synaptic parameters in SNRAM for V = 3.
48 SYN_ADDR4="00000080"    ; First address of Synaptic parameters in SNRAM for V = 4.
49 SYN_ADDR5="000000A0"    ; First address of Synaptic parameters in SNRAM for V = 5.
50 SYN_ADDR6="000000C0"    ; First address of Synaptic parameters in SNRAM for V = 6.
51 SYN_ADDR7="000000E0"    ; First address of Synaptic parameters in SNRAM for V = 7.
52 GSYN_ADDR="00000090"    ; First address of Global Synaptic parameters in SNRAM.
53 NEU_ADDR0="000003E3"    ; First address of Neural parameters in SNRAM (995) for V = 0.
54 NEU_ADDR1="000003E7"    ; First address of Neural parameters in SNRAM (999) for V = 1.
55 NEU_ADDR2="000003EB"    ; First address of Neural parameters in SNRAM () for V = 2.
56 NEU_ADDR3="000003EF"    ; First address of Neural parameters in SNRAM () for V = 3.
57 NEU_ADDR4="000003F3"    ; First address of Neural parameters in SNRAM () for V = 4.
58 NEU_ADDR5="000003F7"    ; First address of Neural parameters in SNRAM () for V = 5.
59 NEU_ADDR6="000003FB"    ; First address of Neural parameters in SNRAM () for V = 6.
60 NEU_ADDR7="000003FF"    ; First address of Neural parameters in SNRAM () for V = 7.
61
62
63 SEED_ADDRL = "000003FD" ; Address of noise seed in SNRAM
64 SEED_ADDRH = "000003FE";Address of noise seed in SNRAM
65 PEID  = "000003FF" ; Address of  PE Identifier number
66
67 ; General constants
68 K_syn= "0000F99A"; (40ms-1ms/40)
69 K_act= "0000FFFA"; (11000-1/11000)
70 N05=    "00000CCC"; 1/20
71 L_MASK_LS= "0000007F"; 7 bit mask
72 MMAX= "00000800"; 4096/2=2^9 to be right shifted
73
74 .CODE
75 ;
76 GOTO MAIN ; Jump to main program
77 ;
78 ; ************************** PROCEDURES BEGIN **************************
79 ;
80 .RANDOM_INIT  ; Uses R0 and R1
81  LOADBP SEED_ADDRL
82  LOADSN
83  SEED
84  LOADBP SEED_ADDRH
85  LOADSN
86  SEED    ;
87 RET
88 ;
89 .LOAD_NEURON    ; Uses R0, R1, R2, R3, R5
90  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
91  LOADBP   ; SNRAM pointer to currently processed neuron
92  LOADSN  ; Load Neural parameters from SNRAM to R1<=u & ACC<=Vmem
93  STORESP          ; FAKE STORE ONLY TO MAKE PC+1
94  MOVR R2  ; Move Vmem from ACC to R2
95  MOVA R1  ; ACC<=u
96  MOVR R3  ; put r3<=u
```

```
 LOADSN              ; load ACC<=d; R1<=b
 STORESP             ; FAKE STORE ONLY TO MAKE PC+1
 MOVR R5   ; R5<=d
 SWAPS R5  ; SW5<=d
 MOVA R1   ; ACC<=R1<=b;
 MOVR R5   ; R5<=b
 LOADSN
 STORESP
 MOVR R6   ; R6<=C
 SWAPS R6  ; SW6<=C
 MOVA R1
 MOVR R6   ; R6<=A
 LOADSN
 MOVR R4   ; R4<=Mi
 MOVSR R4  ; SW4<=Mi
 MARK
RET
;
.DETECT_SPIKE ; Uses R0,R3,SW5, SW6 and R2
 LDALL ACC VTHRES
 SUB R2    ;Compare Vth - Vmem
 SHLN 1    ;subtraction sign to C flag
 FREEZEC   ;if doesn't spiked do:
  SWAPS R5
  RST R5
  RST ACC; DA COMMENTARE
 UNFREEZE
 FREEZENC ;If spiked, do:
  ;REMEMBER: C stays in SW6.
  SWAPS R6
  MOVA R6
  SWAPS R6
  MOVR R2
  MOVA R3   ;ACC<= u
  ;REMEMBER: d stays in SW5.
  SWAPS R5 ;R5<=D
  ADD R5   ;ACC<= u+d
  MOVR R3   ;u<= u+d
  RST R0
  INC   ; 1 in ACC
  SHLN 5
  SHLN 5
  MOVR R5   ; SET 11th bit of SW5 in order to stress the presence of spike
  SET ACC
 UNFREEZE
 STOREPS    ; Push spikes
RET
;
.SYNAPSE_CALC ;
LOADSP    ; ACC<=Mj&&Sj(t), R1<=P
MOVSR R1   ; SW1<=P
MOVSR R0   ; Sw0<=Mj&&Sj(t)
MOVA R1; put 0 or 1 if this neuron has STDP or NOT
SHRN 1;
FREEZEC; if c=1 freeze and do STDP
 MOVRS R0   ; ACC<=Mj&&Sj(t)
 SHRN 1   ; Move spike to flag C
      FREEZENC
```

```
155   MOVA R1 ; Synaptic parameter to ACC
156   SHRN 1; to delete the STDP flag
157   ADD R3;
158   MOVR R3;
159         UNFREEZE
160 UNFREEZE
161 MOVA R1; put 0 or 1 if this neuron has STDP or NOT
162 SHRN 1;
163 MOVRS R0; ACC<=Mj&&Sj(t)
164 FREEZENC; if c=1 do STDP
165  MOVRS R0  ; ACC<=Mj&&Sj(t)
166  SHRN 1    ; ACC<=Mj
167  MUL R5    ; ACC<=Mj*Si
168  NOP
169  MOVA R1
170  MOVR R7   ; R7<=PARTIAL<=  Mj*Si(t)!!
171  MOVRS R0  ; ACC<=Mj&&Sj(t)
172  BITCLR 0
173  LDALL R4 K_syn
174  MUL R4    ; ACC<= M*Ksyn=M(t+1)&Sj
175  ; the stored value of Sj is don't care, it will be took by lcl_mem
176  SWAPS R0   ; ACC<=Mj&&Sj(t), SW0<= M*Ksyn= M(t+1)
177  MOVR R1;
178  SHRN 1
179  FREEZENC   ; if there is spike
180   SWAPS ACC ; ACC<= M*Ksyn= M(t+1), SW0<=Mj&Sj(t) =Mj(t)
181   RST ACC   ; replace M*Ksyn with 2**11
182   INC
183   SHLN 7
184   SHLN 4    ; M=(2*2**10) LESS INCISIVE STDP
185   SWAPS ACC ; ACC<= Mj ,SW0<= 2**11= M(t+1)
186   MOVRS R4  ; R4<=Mi=SW4
187   MOVA R7   ; ACC<= PARTIAL
188   SUB R4;   ; ACC<= PARTIAL <= Si(t)*Mj(t)-Sj(t)*Mi(t)
189   MOVR R7   ; R7<= PARTIAL
190  UNFREEZE
191  MOVA R1
192  MOVRS R1  ; R1 <= P&F
193  SWAPS R0  ; ACC<= M(t+1), SW0<= Mj&Sj,
194  STORESP   ; MEM<=Mj&&Sj(t+1), MEM<=P
195  NOP
196  MOVA R1
197  SHRN 1
198  MOVR R1
199  MOVSR R1  ; R1<=P
200  LOADSN    ; ACC<= L(15 downto 0), R1<= L(31 downto 16)
201  SWAPS R7  ; SW7<= PARTIAL
202  MOVR R7   ; R7 <= L(15 downto 0)
203  MOVRS ACC ; ACC<= Mj&Sj
204  SHRN 1
205  FREEZENC  ; se c'   lo spike
206   MOVA R1  ; ACC<= L(31 downto 16)= [L]*2^10=20*2^10
207   MOVR R4
208   SHRN 5   ;
209   SHRAN 3  ; ACC<=L*2
210   LDALL R1 N05 ; 1/20 STORE +1 in order to delete the freeze
211   MULS R1  ; ACC<= L*4/20. 21532000 ps
212   FREEZENC
```

```
213   INC
214   UNFREEZE
215   MOVRS R1; R1<=P
216   MULS R1 ; R1<=L*P*4/20
217   MOVA R1
218   SHRAN 2 ;  L*P*1/20
219   ADD R3  ; ADD SUM OF WEIGHTS
220   MOVR R3 ; STORE SUM OF WEIGHTS
221   MOVA R4
222   MOVR R1 ; R1<=L(31 downto 16)
223  UNFREEZE
224  LDALL R4 K_act  ; ---------------------------
225  MOVA R1  ; ACC<= L(31 downto 16)
226  MUL R4;
227  MOVSR ACC ; SW0<= MSW of MSW
228  MOVSR R1 ; SW1<= LSW of MSW
229  MOVA R7  ; ACC<=LSW of L
230  MUL R4  ; ACC<= MSW of LSW), R1 unuseful
231  RST R4
232  MOVRS R1 ; R1<=LSW(MSW)
233  ADDU R1;
234  FREEZENC  ; overflow
235   SET R4 ; R4<=-1 1741006799 ps
236  UNFREEZE
237  MOVR R7  ; R7<= L(15 downto 0)(t+1)
238  MOVRS ACC  ; ACC<= MSW(MSW)
239  SUB R4   ; ACC<= L(31 downto 16)(t+1)
240  MOVR R1  ; R1<=L*2^10=
241  SWAPS R7 ; R7<= PARTIAL, SW7<=L(15 downto 0)(t+1)
242  MOVA R7          ; in this way ACC<= PARTIAL = M * 2^10
243  ADD R1
244  MOVR R1  ; R1<=L(31 downto 16)(t+1)
245  SWAPS R7 ; R7<=L(15 downto 0)(t+1)
246  MOVA R7
247  STORESP
248 UNFREEZE
249 RET
250 ;
251 .Mi
252  MOVRS R4  ; R4<= Mi
253  LDALL R7 MMAX
254  MOVA R5
255  MUL R7
256  MOVA R1
257  SHRAN 1; NOTE THAT MMAX is MiMAX*2 in order to be shifted
258  FREEZENZ
259   LDALL ACC K_syn
260   MUL R4
261  UNFREEZE
262  MOVR R4
263  MOVSR R4  ;SW4<= Mi
264 RET
265 ;
266 .GAUSS_NOISE ; Uses SW0, R2, R6 and R7
267  RANDON                  ;LFSR ON
268  RANDOFF                 ;LFSR OFF. Arbitrarily heres
269  LLFSR    ;Noise seeds to ACC, R1, SR0, SR1
270  ADD R1
```

```
271  SHRAN 1
272  MOVR R7
273  SWAPS R0
274  SWAPS R1
275  ADD R1
276  SHRAN 1
277  ADD R7
278  LDALL R7 N001; ACC=OUTPUT
279  MULS R7  ;NOISE IN ACC
280  FREEZENC
281   INC
282  UNFREEZE
283  MOVR R4  ;NOISE IN R4
284  SWAPS R6
285  MOVA R6
286  SWAPS R6
287  LDALL R7 REST_POT; tipical value of resting potential to be compared with the actual
        one
288  SUB R7  ; compare it with the value of the parameter. if =, it is excitatory
289  MOVA R4
290      FREEZEZ
291   LDALL R7 N5;
292   MUL R7
293   NOP
294   MOVA R1
295   SHRAN 1
296  UNFREEZE
297  MOVSR ACC                ;TO STORE 1/2 THE NOISE
298 RET
299 ;
300 .MEMBRANE_POTENTIAL ;Uses R0,R4,R7
301  MOVA R2
302  MULS R0  ; v^2*2^12
303  NOP  ; Check if needed
304  ;Shift R0R1 4 positions left
305  SHLN 4  ; Shift Accumulator 2^4
306  MOVR R4
307  MOVA R1  ; Move LS part (R1) to R0 (2^16)
308  SHRN 4
309  SHRAN 4
310  SHRAN 4  ; 2^16/2^12 = 2^4
311  ADD R4  ; Combine and obtain v^2/2^12
312  LDALL R4 N0002 ; 0.0002*2^27 is in R4
313  MULS R4  ; v^2*2^(-12)*0.0002*2^27/2^16 = 0.0002*v^2*2^(-1)
314  NOP  ; Check if needed
315  SHLN 1  ; Shift Accumulator 2^1
316  MOVR R4
317  MOVA R1  ; Move LS part (R1) to R0 (2^16)
318  SHRN 5
319  SHRAN 5
320  SHRAN 5  ; 2^16/2^15 = 2^1
321  ADD R4  ; Combine and obtain 0.0002*v^2
322  MOVR R7;
323  MOVA R2;         ACC<=Vinit
324  SHRAN 2;         ACC<=0.25*Vinit;
325  ADD R2;  ACC<=ACC+Vinit=1.25*Vinit
326  SHLAN 1
327  ADD R7
```

```
328  LDALL R4 N70;    R4<=70
329  ADD R4;
330  MOVR R7
331  RST ACC
332  SUB R3;    ACC=- u
333  SHRAN 1;
334  ADD R7;
335  ADD R2;   ACC=ACC+Vinit
336  MOVR R2;   Back to R2 where membrane potential is stored
337 RET
338 ;
339 .RECOVERY_UPDATE ;uses R3,R5,R6
340  MOVA R2;           ACC<=Vinit
341  SWAPS R5;
342  MULS R5                    ;ACC<=R5*ACC=B*Vinit
343  FREEZENC
344    INC
345  UNFREEZE
346  SUB R3                     ;ACC<= ACC-R3= ACC-Uinit
347  ;REMEMBER: A is in R6
348  MULS R6;                    ACC<=A*ACC;
349  FREEZENC
350    INC
351  UNFREEZE
352  ADD R3;                     ACC<=ACC+Uinit
353  MOVR R3;                    Back to R3 where recovery value is stored
354 RET
355 ;
356 .SUM_NOISE_AND_Ws
357  MOVRS ACC                    ;NOISE TO ACC
358  ADD R2     ;ADD NOISE TO SIGNAL
359  MOVR R2
360  SWAPS R3    ;SYN. contribute
361  MOVA R3
362  ADD R2;
363  MOVR R2;    store membrane potential
364  SWAPS R3;
365 RET
366 ;
367 .STORE_NEURON    ;uses R0,R3 and R1
368  MOVA R3   ;move u from R3 to acc
369  MOVR R1   ;move u from ACC to R1
370  MOVA R2   ; Move Vmem from R2 to ACC
371  READMPV NEU_ADDR0 ; Address of real neuron + virt (valid also for non-virtual)
372  LOADBP    ; SNRAM pointer to currently processed neuron
373  STORESP   ; Store u&Vmem to SNRAM
374  NOP
375  LOADSN
376  NOP
377  STORESP   ;store second row
378  NOP
379  LOADSN
380  NOP
381  STORESP   ;store third row
382  NOP
383  LOADSN
384  MOVRS R4;
385  MOVA R4;
```

```
386  STORESP
387 RET
388 ;
389 ; ************************** PROCEDURES END ****************************
390
391 ; ************************** MAIN PROGRAMME BEGIN ***********************
392 .MAIN
393 ;
394
395 ; Virtual operation init
396 LAYERV virtual_layers ; Init sequencer vlayers. It is 0 for non-virtual operation
397 LDALL ACC, VLAYERS ; Load defined virtual layers to PE array
398 SPMOV 0 ; VIRT <= ACC
399
400
401 ; Initial instructions
402 GOSUB RANDOM_INIT  ; For noise initialization
403
404 .EXEC_LOOP ; Execution loop
405
406  LOOP virtual_layers     ;Neuron loop for virtual operation
407       NOP     ;to prevent pipeline error
408       GOSUB LOAD_NEURON
409       GOSUB DETECT_SPIKE
410  SWAPS R3     ;SYNAPSE Sum will be store IN R3
411  MOVA R5
412  SHRN 5
413  SHRN 5
414  MOVR R5; put 0 o 1 in R5 if there is a spike or not
415  READMPV SYN_ADDR0
416       LOADBP
417           LOOPV V0  ;synaptic loop. Reads number of current-layer synapses
418   NOP          ;to prevent pipeline error
419            GOSUB SYNAPSE_CALC
420       ENDL
421  MOVA R3     ;move to acc synapse sum
422     SHRAN 1; save half the weight
423       MOVR R3  ;move to R3
424  SWAPS R3 ;SYNAPSE sum to SW3
425  GOSUB Mi
426  GOSUB GAUSS_NOISE
427       LOOP n_step
428         NOP
429   GOSUB MEMBRANE_POTENTIAL  ;Calculate membrane potential
430        GOSUB SUM_NOISE_AND_Ws
431  ENDL
432  GOSUB RECOVERY_UPDATE
433  GOSUB STORE_NEURON
434  RST R3;
435  MOVSR R3;
436  INCV
437  ENDL
438 .FINISH
439 NOP          ;Empty pipeline wait NOPs
440 NOP
441 NOP
442 SPKDIS        ;Distribute spikes
443 GOTO EXEC_LOOP       ;Execution loop
```

Bibliography

[1] M. Abeles et al. “Spatiotemporal firing patterns in the frontal cortex of behaving monkeys.” In: *Journal of Neurophysiology* 70 (1993), pp. 1629–1638.

[2] C. C. Bell et al. “Synaptic plasticity in a cerebellum-like structure depends on temporal order”. In: *Nature* 387 (1997), pp. 278–281.

[3] Hebb D. “The organization of behaviour”. In: (1949).

[4] J.A. D’amour and R. C. Froemke. “Inhibitory and Excitatory Spike-Timing-Dependent Plasticity in the Auditory Cortex”. In: *Neuron* 86.2 (2015), pp. 514–528. DOI: :10.1016/j.neuron.2015.03.014..

[5] Dorta, Zapata, and J. Madrenas. “AER-SRT: Scalable spike distribution by means of synchronous serial ring topology address event representation”. In: *Neurocomputing* 171 (2016), pp. 1684–1690.

[6] W. Gerstner and W. M. Kistler. “Spiking neuron models: Single neurons, populations, plasticity.” In: *Cambridge University Press* (2002).

[7] G. E. Hinton and R. R. Salakhutdinov. “Reducing the dimensionality of data with neural networks.” In: *science* 313 (2006), pp. 504–507.

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

[9] E. Izhikevich. “Simple model of spiking neurons.” In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572.

[10] Iglesias J. et al. “Dynamics of Pruning in Simulated Large-Scale Spiking Neural Networks”. In: *Biosystems* (2005).

[11] M. Kang. “FPGA implementation of Gaussian-distributed pseudorandomnumber generator”. In: *Conf. on Digital Content, Multimedia* (2010).

[12] S. R. Kelso, A. H. Ganong, and T. H. Brown. “Hebbian synapses in hippocampus.” In: *roc NatlAcad Sci U S A* 83 (1986), pp. 5326–5330.

[13] Jesus L. Lobo et al. “Spiking Neural Networks and online learning: An overview and perspectives”. In: *Neural Networks* 121 (2020), pp. 88–100.

[14] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity.” In: *The Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.

[15] P. Roberts and C. Bell. “Spike timing dependent synaptic plasticity in biological systems”. In: *Biol. Cybern.* 87 (2002), pp. 392–403.

[16] A. Sripad et al. “SNAVA—A real-time multi-FPGA multi-model spiking neural network simulation architecture”. In: *Neural NEtworks* 97 (2018), pp. 28–45.

[17] Wikipedia contributors. *Linear-feedback shift register — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Linear-feedback_shift_register&oldid=986763170.

[18] M. Zapata and J. Madrenas. “Unpublished material regarding HEENS architecture”. In: ().

[19] Zapata and J. Madrenas. “Compact Associative Memory for AER Spike Decoding in FPGA-Based Evolvable SNN Emulation”. In: *Artificial Neural Networks and Machine Learning* (2016).

[20] Nan Zheng and Pinaki Mazumder. *Learning in energy-efficient neuromorphic computing : algorithm and architecture co-design*. Wiley-IEEE Press, 2020. ISBN: 9781119507390.

Acknowledgments

At the end of this "long and winding road" it is time to look back for once and not let everything go unnoticed, but let it be remembered. This thesis is the last official assignment of my university career, and the only thing I can do is to say thank you to the people who left a lasting impression on it.
First, I want to start from the end and to thank my supervisor at UPC Jordi: even with all the limitations imposed by COVID, even if a project with these modality was a first time for both of us, even though a virtual link, Jordi has been always ready and available for my every need, patient and intellectually stimulating in every step of our project. Thank you, Jordi.
Thanks to the entire Polito for such a high level of education and thanks to my advisor Guido Masera.
Thanks to my friend Corrado, companion of thesis and of a missed adventure, thanks to Danilo, fellow of stupid things in quarantine, thanks to all the friends who shared the Turin experience and the Cernaia nights with me, such as Luca, Giuseppe, Roberta, la casta Antonio, Roberta Marineo, Ilaria, Antonella, Francesca, Paola.
Now that I am arrived at the end, it is time for the begin.
Grazie a Marco e Dario per tutto quello che ha significato per me conversare con loro per tutti questi anni.
Grazie a Danieli, Roberto, Totò, Luke (De Luca) per aver condiviso tutto sin dai banchi di Unipa: tutte le risate, le imitazioni, i soprannomi, le assurdità, le difficoltà, per essere stati i migliori "ingegneri atipici" che avrei potuto desiderare. Rivivrei ogni singolo giorno passato insieme.
Grazie ai Bazinga tutti perchè in qualche modo ci siete stati sempre e molti di voi si sono dovuti sorbire tutto il mio pessimismo pre-esame, ogni volta poi smentito.
Grazie ai 12BBR per avermi fatto vivere due vite.
Grazie a mia cugina Chiara per l'entusiasmo.
Un grazie immenso a Viviana per aver condiviso tutto, aver sopportato molto più di quanto fosse lecito ed essere stata ogni singolo passo di questo cammino.
Un grazie finale alla mia famiglia tutta per avermi permesso di studiare ed avermi supportato sempre, un grazie speciale a mia madre per essere la promotrice di questa ricerca per la felicità che è l'apprendimento.