VHDL implementation of the multi-timescale adaptive threshold neuronal model

Rodolfo Rocco

October 6, 2016

Contents

Acronyms

MAT	multi-timescale adaptive threshold
SRM	spike response models
ARP	absolute refractory period
LUT	look-up table
GLM	generalized linear model
RHS	right-hand side
FSM	finite-state machine
ROM	read only memory
FPGA	field programmable gate array
IF	integrate and fire

Abstract

In this work we describe the VHDL implementation of the MAT neuronal model. Firstly we describe the theoretical model on which the implementation is based. Secondly we describe the hardware architecture and we discuss the design choices that have been made in order to account for the hardware requirements of an eventual physical realization. Finally we discuss the predictive performance of the project when compared with that of the original software model. Given that the parameters of the model have not been computed through dynamic programming, as originally done by the authors of the MAT model [8], but rather are rough estimates obtained with a trial and error method, even though the predictive performance of the model is not on par with that of the software model, it is essentially in line with the expectations.

Chapter 1

The MAT model

1.1 Overview

The MAT model belongs to the family of the SRM [5]. Its main feature is the functional form of the filter which describes the dynamic threshold, which is not a simple exponential but rather the sum of different exponentials, each with its own time scale and coefficient; hence the name MAT. The parameters of the model which govern the neuronal dynamics are the aforementioned time scales and coefficients; in addition to these we must also consider the value of the static threshold. Finally there is the absolute refractory period (ARP), which is equally important to the firing of the neuron, even though it is possible to include the effects of refractoriness by raising the dynamic threshold, for the duration of the ARP, to a value that cannot be attained by the neuron [6].

Given the similarities between the two models, the following considerations, with the exception of those pertaining the dynamic threshold, which are exclusive to the MAT model, will apply to both of them.

1.2 The SRM

The SRM is a phenomenological model. Two are its principal characteristics: The first is the formalism used to describe the dynamics of the neuron, which is the same that is applied in the field of signal-processing; the second is the model modularity, that is to say the ability to add and remove certain features without compromising the basic functionality.

These two characteristics make the SRM the perfect candidate for a hardware implementation. The description of the neuronal dynamics in terms of filters lends itself to a direct and easy to understand mapping, where every filter is implemented in practice using a LUT. In addition to that the modularity allows to swap components very easily in order to evaluate the impact of a certain feature on the resources requirements, thus establishing the best trade off between predictive performance and model complexity.

1.2.1 Model dynamics

Two are the equations which describe the neuronal dynamics in the SRM:

$$u(t) = \int_0^\infty \eta(s)S(t-s)ds + \int_0^\infty \kappa(s)I(t-s)ds + u_{rest} \tag{1.1}$$

$$\theta(t) = \theta_0 + \int_0^\infty \theta_1(s)S(t-s)ds \tag{1.2}$$

A spike is emitted at time t whenever $u(t) \geq \theta(t)$ holds true, if spike emission is deterministic. Otherwise, if it is stochastic, we can define the firing intensity, ρ, as:

$$\rho(t) = f(u(t) - \theta(t)) \tag{1.3}$$

A common choice for $f(u(t) - \theta(t)$ would be

$$f(u(t) - \theta(t)) = \frac{1}{\tau_0}\exp[\beta(u(t) - \theta(t))] \tag{1.4}$$

The SRM equipped with stochastic firing is an example of a generalized linear model (GLM).

Let us go back to equations (1.1) and (1.2). Equation (1.1) describes the variation of the membrane potential over time. Three terms make up the equation. One, u_{rest}, is simply the rest potential, which is constant. $\int_0^\infty \eta(s)S(t-s)ds$ accounts for the spike and the after-potential. $\int_0^\infty \kappa(s)I^{ext}(t-s)ds$ is the response of the membrane filter to the input current. We will now examine these last two terms.

The kernel η has two contributions: the spike and the after-potential. Usually it is supposed that the shape of the spike remains always the same; therefore, to determine η, one strategy consists in extrapolating the after-potential by fitting the experimental data with, for example, an exponential decay like the one in equation (1.6).

$$\eta(t - t^f) = -\eta_0 \exp\left(-\frac{t - t^f}{\tau_{recov}}\right) \tag{1.5}$$

Since the spike and the subsequent after-potential need simply to be pasted in whenever the neuron emits a spike, their contribution is given by the convolution of the kernel η with the spike train $S(t) = \sum_f \delta(t - t^f)$, where t^f is the time of firing. In other words:

$$\int_0^\infty \eta(s)S(t-s)ds = \sum_f \int_0^\infty \eta(s)\delta(t - s - t^f)ds = \sum_f \eta(t - t^f) \tag{1.6}$$

Onto the second term of equation (1.1). Firstly, let us observe that $I(t)$ can be expressed as the sum of an external current, like for instance the one injected by the experimenter, and an internal one, that is to say the current produced by the arrival of the pre-synaptic action potentials at the synapses terminal. While the external current needs no clarification, it is convenient to write the internal current as a convolution of a kernel, α, with the pre-synaptic spikes, the reason being that the formalism reminds us of the hardware implementation that we will discuss in the next chapter. With that said, the internal current is now given by:

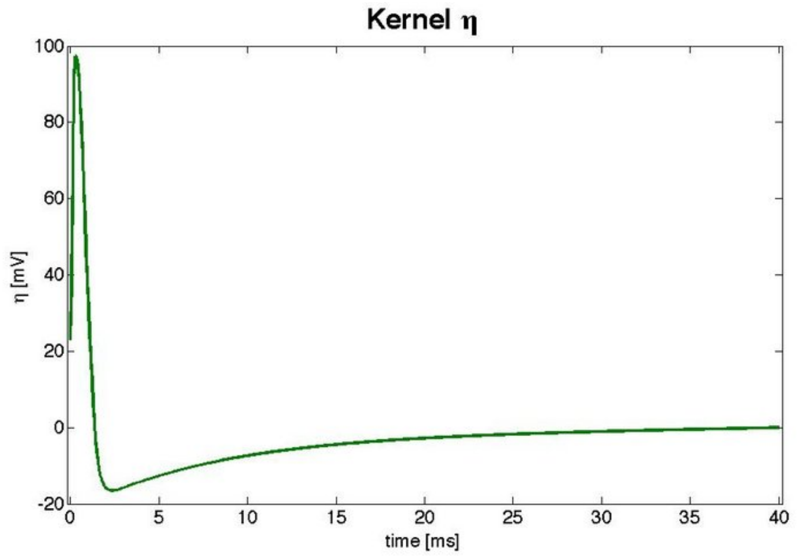

Figure 1.1: An example of the kernel η [5]

$$I_{int}(t) = \sum_j w_j \sum_f \int ds\alpha(s)\delta(t-s-t^f) = \sum_j w_j \sum_f \alpha(t-t^f) \tag{1.7}$$

j is the synapse index; w_j is the weight of the j-th synapse. Let us focus now on the second summation, $\sum_f \alpha(t-t^f)$. α is the kernel which describes the current response of a synapse to an incoming spike. Since α has no index j we suppose that the response is the same for every synapse of a certain kind, though it will be different for excitatory and inhibitory synapses. As a result of the convolution with the spike train, the kernel α is a function of the time elapsed since the arrival of the last spike. A typical shape for the kernel α in the following:

$$\alpha(t) = \frac{t}{\tau}\exp(-\frac{t}{\tau}) \tag{1.8}$$

Needless to say the weights w_j are such that the first member on the right-hand side (RHS) of equation (1.7) is dimensionally correct. Finally, if we put equations (1.7) and (1.6) inside equation (1.1) we get:

$$u(t) = \sum_f \eta(t-t^f) + \sum_j w_j \sum_f \int_0^\infty ds\kappa(s)\alpha(t-t^f-s) \tag{1.9}$$

In a last effort to get one step closer to the actual hardware implementation we discretise time, obtaining:

$$u(t) = \sum_f \eta(t-t^f) + \sum_j w_j \sum_f \sum_{s=0}^{N} \kappa(s)\alpha(t-t^f-s) \tag{1.10}$$

N is the order of the filter with kernel κ and input α. It has been observed [3] that a single exponential, like for instance $\kappa(t) = \frac{1}{C}\exp(-t/(\tau_m))$, well describes the filter kernel.

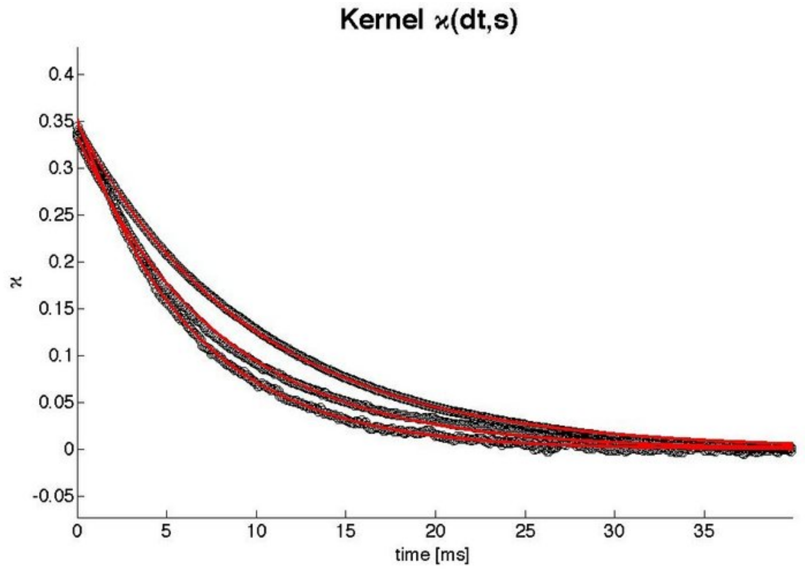

Figure 1.2: The kernel k, extracted from data for three different values of the time [5]

The second equation which describes the dynamics of the SRM is equation (1.2). The first term on the RHS is the static threshold, θ_0; the second one is the dynamic threshold, which is reset every time the neuron emits a spike. The latter is analogous to η, therefore we can write it using the same formalism:

$$\theta(t) = \theta_0 + \sum_f \theta_1(t - t^f) \tag{1.11}$$

While it is possible to modify at will the behaviour of the neuron by varying any of the parameters which appear in the preceding equations, the dynamics of the neuron are mainly dependent on the threshold. As a consequence we can attribute realistic values to the large majority of the parameters once and for all, while those which govern the dynamic threshold are optimized by comparing the output of the model with that of a real neuron stimulated with the same input. It is equally clear that the choice of a particular kernel θ_1 will play a vital role in achieving a satisfying predictive performance.

In the following session we discuss the MAT, whose strength lies in the choice a fairly complex θ_1, given by the sum of two or more exponentials with different time scales.

1.3 The dynamic threshold in the MAT

The MAT model belongs to the family of the spike response models. Its defining characteristic is the equation for the kernel θ_1 which appears in the definition of the dynamic threshold.

$$\theta_1(t) = \sum_{j=1}^{L} \alpha_j \exp(-t/\tau_j) \tag{1.12}$$

Thus, the kernel θ_1 is given by the sum of L exponentials, each with its own coefficient α_j and time scale τ_j. It has been shown [8] that the best predictive performance, defined as in [2], when neurons in layers 2/3 and 5 of the rat motor cortex are considered, can be achieved with two exponentials, with time scales equal to 10ms and 200ms. The coefficients $\alpha_{1,2}$ and the rest potential are the parameters which describe the dynamics of the neuronal model.

The MAT model fairs well when compared against other phenomenological models: in 2009 it placed first in a competition whose challenge was the prediction of the spike timing of a regular spiking L5 pyramidal cell [1].

Chapter 2

The hardware architecture

In this chapter we will detail the hardware architecture of our own implementation of the MAT. The starting point is the FSM which succinctly illustrates the neuronal dynamics and doubles as the highest abstraction layer.

2.1 General remarks

Firstly, let us dwell on some general considerations. For instance, should the computations be carried in the floating point representation or in the fixed point one? If we examine the relevant physical quantities, like the injected current or the membrane potential, we can observe that their values lie in the range of three orders of magnitude of one another, at most. This suggests that the disadvantages, in terms of hardware complexity and ease of implementation, of adopting the floating point representation may outweigh the advantage of the dynamic range. For this reason we have opted for the fixed point representation.

The chosen word length is 32 bits: 17 decimal bits and 14 integer bits. While it would have been convenient to have more than 32 bits, if we have to restrict ourselves to powers of 2, 64 bits would be decidedly wasteful. 32 bits are just enough to distinguish one increment of the membrane potential from the other and to prevent overflow when the firing threshold is being approached.

In one occasion we had to use a scaling factor in order to sum the synaptic currents, which are internally computed in nF, and the external current, which is provided in pF.

As we noticed previously, many of the parameters of the model can be set preliminarily and left unchanged for the duration of the simulation. In the following table we report the values of such parameters.

parameter	value	unit
neuronal sampling rate	5	kHz
excitatory synapses time scale	1	ms
inhibitory synapses time scale	3	ms
excitatory synapses weight	0.1	nA
inhibitory synapses weight	0.033	nA
rest potential	-65	mV
membrane time scale	5	ms
membrane capacitance	0.5	nF
absolute refractory period	2	ms

2.2 The FSM

The FSM is comprised of three states: rest potential (R), excitation (E), after-potential (A). The transition from state R to state A happens when $u(t) \geq \theta(t)$. Whenever the machine enters state E it emits an impulse, which in our digital world plays the role of the action potential. Transition from state E to state A happens immediately, regardless of the value of the potential or any other variable. The machine remains in state A until the ARP is elapsed, then it transits to state A. The machine is susceptible to stimulation only in state A.

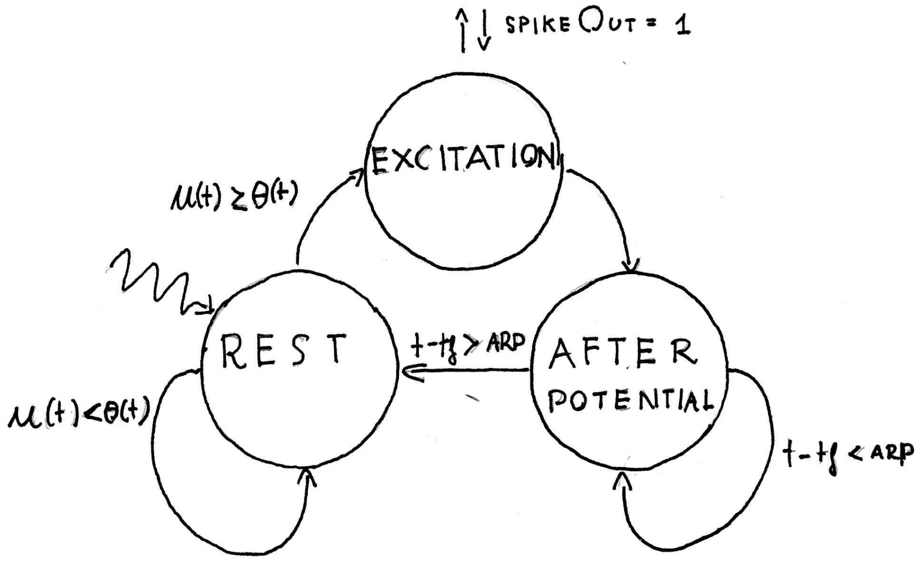

Figure 2.1: The FSM describing the neuronal dynamics

2.3 The pins

Every neuron has one single-bit output. As noted in the previous section the neuron signals its entering state E by emitting an impulse; therefore, whenever a spike is excited, the neuron simply

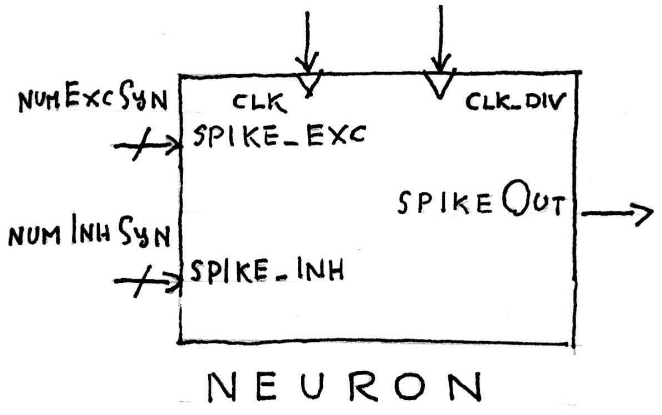

Figure 2.2: The pins configuration of the *neuron* component

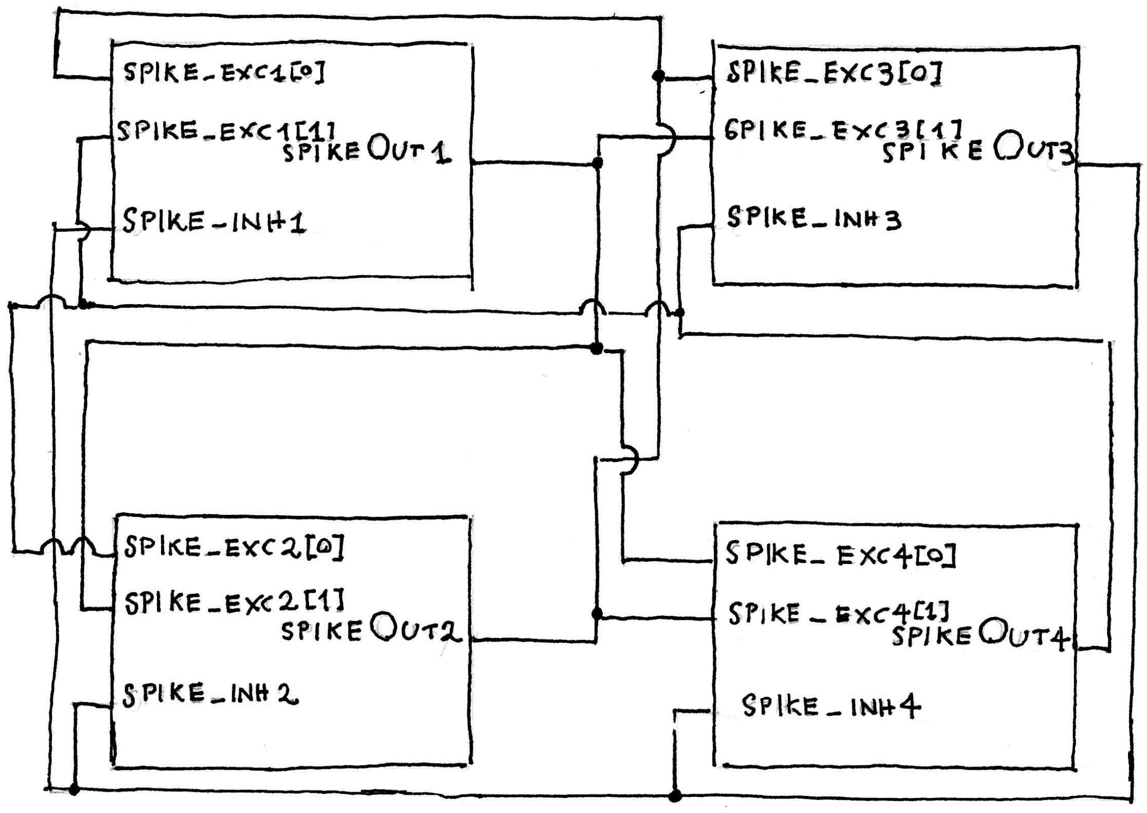

Figure 2.3: Example of four interconnected neurons. The network has full connectivity. The one illustrated is just one of the possible networks. For simplicity's sake we have omitted the clocks.

asserts the *spikeOut* signal. We remind that we are merely interested in the spike timing and not in the value of the membrane potential.

Inputs are fairly more complex. We have two different clocks, *clk* and *clk_div*. The second one is obtained from the first by the use of an external frequency divider. The reason for the presence of two different clocks is readily explained: since we want to process the post-synaptic potentials serially, as shown in one of the following sections, we need an additional clock (*clk*) to serve each synapse's request within the time frame allowed by the neuronal sampling rate (*clk_div*).

Finally we have two vectorial inputs, *spike_exc* and *spike_inh*: each of them is a collection of impulses coming at a certain time from a number of synapses, which can be either excitatory or inhibitory, equal to the respective vector's length. The neurons are networked together by connecting their *spikeOut* ports to another neuron's *spike_exc* or *spike_inh* port, as shown in figure 2.3.

One last note: the neuron can be stimulated using an external current. This current is passed to the neuron in the form of a data file, whose location can be specified by the use of generics.

2.4 The internal organization

We describe the internal organization of the neuron by following the evolution of the FSM when the potential is nearing the threshold.

At any moment t_n which, by necessity, will be an integer multiple of the main clock, that is to say the sampling rate of the neuron *clk_div*, the inputs are updated. When this happens the two vectors, *spike_exc* and *spike_inh*, are passed to two different instances of the same component, *clusterSyn*. The purpose of *clusterSyn* is that of producing the synaptic current by convoluting the pre-synaptic spike with the impulse response α, as in equation (1.7). The two components differ in the fact that two different kernels are adopted, depending on whether the synapses are excitatory or inhibitory.

The currents thus generated are sum with the external current and the result is passed to the membrane filter, named *transversalFilter* in the vhd file.

The membrane filter, as the name suggests, generates the membrane potential. Its input is the total current, defined as above, and its internal architecture is that of a common transversal filter. Looking back at the mathematical model, the equation which describes the behaviour of the filter is equation (1.10), while its kernel is that depicted in figure 1.2.

The membrane potential is sum with the rest potential and the result is compared with the dynamic threshold, whose values are sampled from equation (1.12) and stored in the LUT named *dynamicThreshold*. A new address is generated to access the LUT at every *clk_div* by incrementing the old one, beginning with the null address. The component in charge of this is aptly named *addressGenerator*. The incrementation is halted when the machine enters state A; it resumes from the first address, which points to the greatest value of the dynamich threshold, when the machine leaves it.

If the comparison indicates that the total potential is greater than the dynamic threshold, *spikeOut* is asserted. After that the machine immediately enters state A. Meanwhile the current is not passed to the membrane filter and the dynamic threshold is not updated. When the ARP has elapsed, the machine leaves state A and enters state R. From there, everything proceeds as before.

If the potential is under the dynamich threshold, no change in the configuration of the FSM occurs.

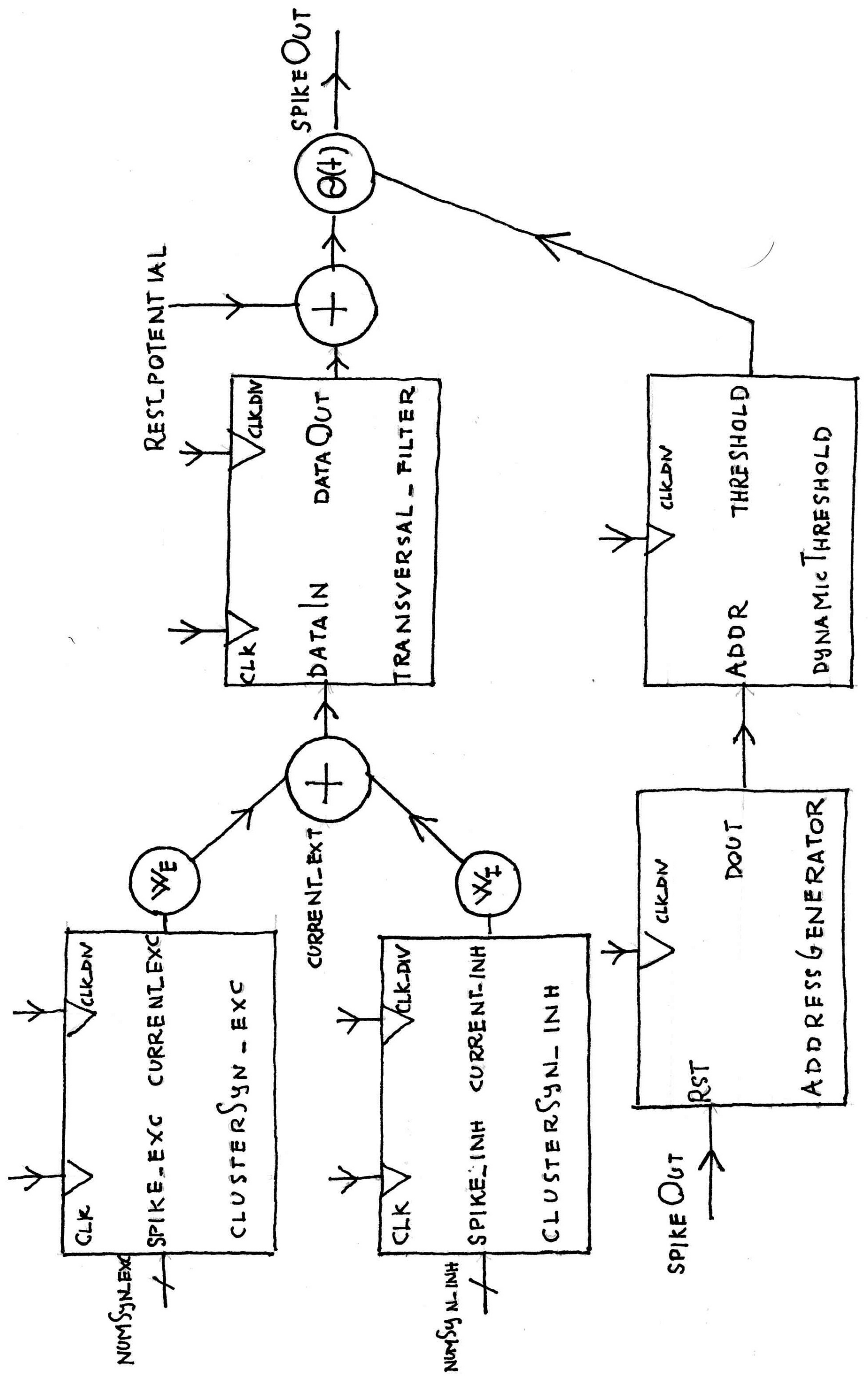

Figure 2.4: Circuit used to generate the *spikeOut* signal. W_e and W_i are the weights of the two kinds of synapses; each synapse has its own weight inside *clusterSyn*. The $\theta(t)$ symbol inside the circle means that *spikeOut* is generated only if the result of the sum is greater than the threshold θ

2.5 The components

The architecture we have illustrated makes use of several components. Some of these are rather basic, like the *LUT*, the *transversalFilter* and the incrementer, called *counter* in the vhd file. Some others are more complex, like the frequency divider (*freq_divider* in the file) and the synaptic terminal (*clusterSyn*).

In the following we will dedicate more space to the more complex components while we will simply outline the functionality of the other ones.

2.5.1 The *counter*

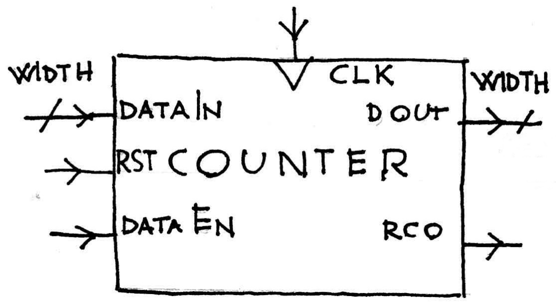

Figure 2.5: Pins configuration of the *counter*

The incrementer, though very basic in nature, has two distinguished qualities: the first one is that the incrementation can start from any given number, which can be passed through the *dataIn* port. This mode must be enabled by asserting *dataEn*. The second one is that when the highest number has been generated the incrementation process doesn't start afresh, as it would be expected; instead the same number is sent to the *dout* port until *rst* is asserted.

In the following we will occasionally call this component *addressGenerator*, to make its function immediately clear.

2.5.2 The *LUT*

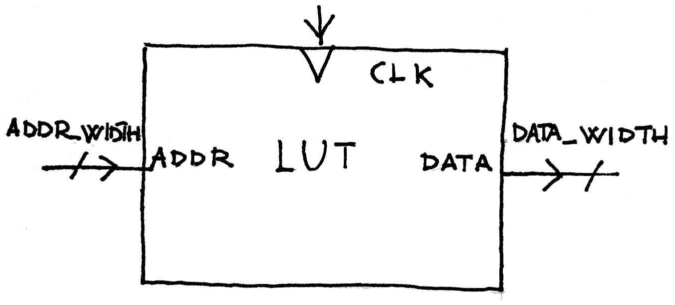

Figure 2.6: Pins configuration of the *LUT*

The LUT takes has as input an address and outputs the corresponding value stored in a read only memory (ROM). Address width and data width are specified via generics.

2.5.3 The *frequency_divider*

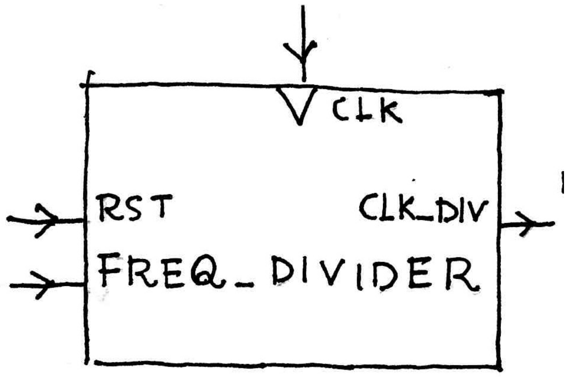

Figure 2.7: Pins configuration of *freq_divider*

The purpose of the *frequency_divider* is that of providing a clock whose period is an integer multiple of the input clock. Since this objective can be achieved through different strategy we will briefly explain the one adopted.

Firstly the component waits a number of *clk* periods equal to the generics named $factor - 1$. At the next rising edge of *clk*, *clk_div*, which is the longer clock produced by the frequency divider, is asserted. Starting from the same period of *clk*, *factor* is decremented, and when it equals zero, *clk_div* is de-asserted.

A reset signal can be sent at every time to stop the process and start it anew.

This implementation guarantees that there will be no delays between the two clocks rising edge, as shown in figure 2.8.

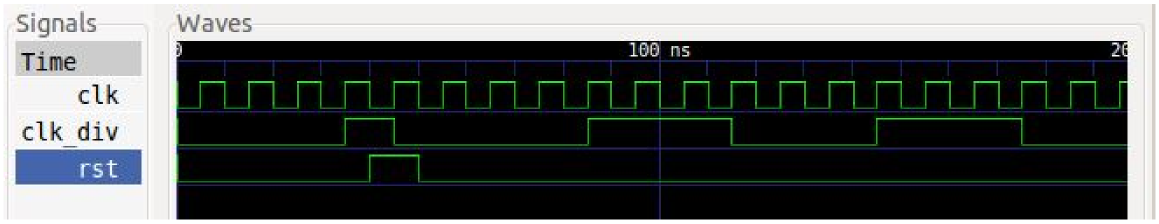

Figure 2.8: Timings of *frequency_divider*. Initially we test the *rst* signal, which interrupts the generation of *clk_div*. Then we verify that the longer clock is generated correctly. In this case $factor = 3$

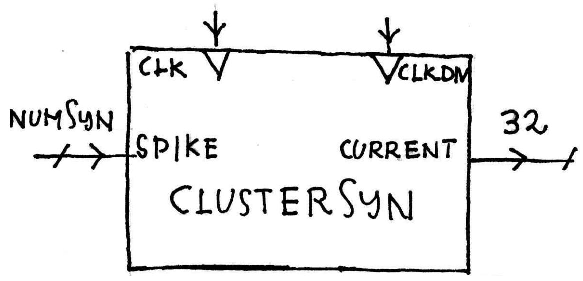

Figure 2.9: Pins configuration of *clusterSyn*

2.5.4 The cluster of synapses (*clusterSyn*)

The number of synapses which connect our hardware neuron to its neighbours can be specified via generics. Generally speaking we'd expect one single neuron to be connected with a great number of other neurons, even if the connectivity of the network is only a small percentage of that of a real neural network. For this reason, in order to mitigate the utilization of logic gates, we have decided to read and process the synapses' response to the upcoming action potentials serially. Obviously this solution comes at a cost: if we want our neuron to refresh its output at a given frequency we need a faster clock to serve the synapses' requests within the allocated time-frame.

It is exactly for this reason that our synaptic terminal has two clocks: *clk_div* is the sampling rate of the neuron, that is to say the rate at which the results for all the synapses must be ready; *clk* is the faster clock used by the internal units to compute the current response to a single action potential.

spike is a vector whose components are the impulses coming from the pre-synaptic neurons: each impulse is a spike which provokes a current response. The sum of the currents is sent to the output port, *current*, which is 32 bits long. *spike*'s length is exactly the number of synapses; an impulse is just one bit which signals whether or not the associated pre-synaptic neuron has fired.

Now we will describe briefly the internal workings of *clusterSyn*.

From equation (1.7) follows that the current response is the highest the moment a pre-synaptic neuron fires and then it decays exponentially. Equation (1.7) has been sampled at regular intervals equal to the inverse of the neuronal sampling rate and the values have been stored in a LUT. There is only one LUT for every *clusterSyn*, meaning that the only difference we make is that between excitatory and inhibitory synapses.

In the generation of the addresses needed to access the LUT we employ the same kind of *address-Generator* that we mentioned in the discussion of the dynamic threshold. For any given synapse there are two possible scenarios: either the pre-synaptic neuron has fired or it has not. In the first case we need to access the very first word of the LUT. The strategy we have adopted is that of connecting the component of *spike* associated with the synapse we are serving to the *rst* port of the *addressGenerator*. It goes without saying that, since we want to have one single *addressGenerator* for every *clusterSyn*, the input of the *rst* port will change at every rising edge of *clk*, that is to say every time we move on to a different synapse.

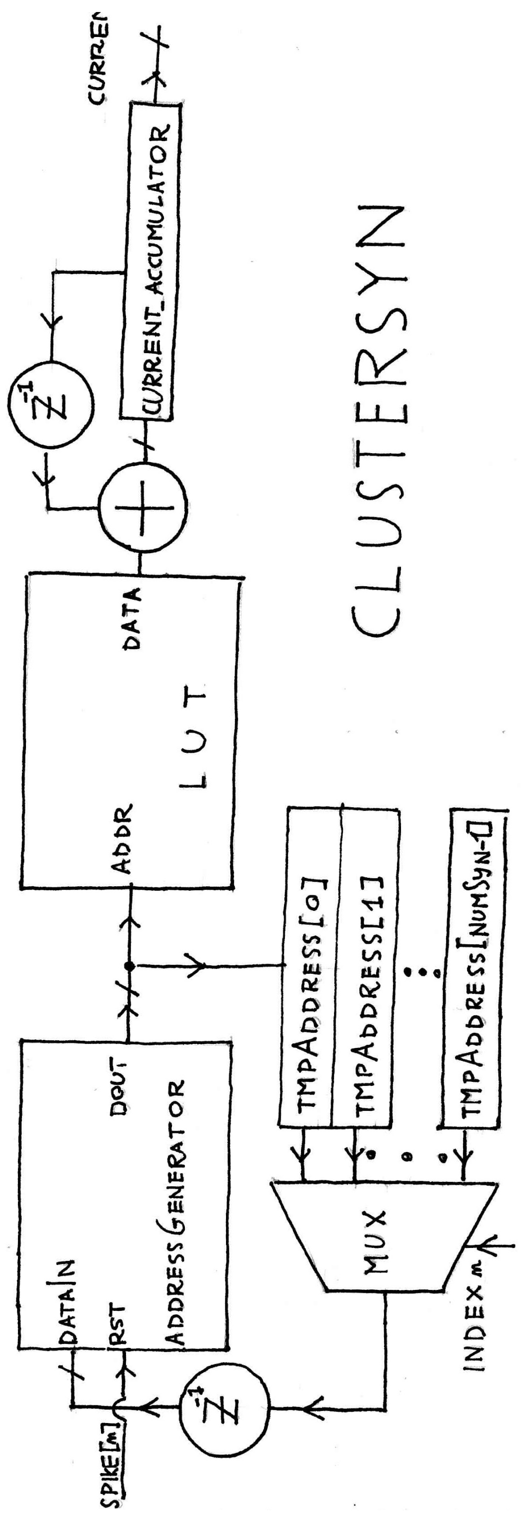

Figure 2.10: *clusterSyn* architecture. Some pins and most notably the clocks have been omitted for simplicity's sake. The z^{-1} indicates a delay.

The value provided by the LUT is then sent to an accumulator. The address is stored in a temporary register, in the case it will be needed the next time we have to process the same synapse (which won't happen until the next *clk_div*).

In the second case the pre-synaptic neuron has not fired. The address previously stored in the temporary register must be incremented. To this purpose we send it to the *dataIn* port of the *addressGenerator* and we assert *dataEn*, meaning that, instead of the last number produced, which was used to compute the current response of the previous synapse, it is the one that we provide that should be incremented. From here the process proceeds as before

When the last synapse has been served the content of the accumulator is sent to the output port *current*.

2.5.5 The *transversalFilter*

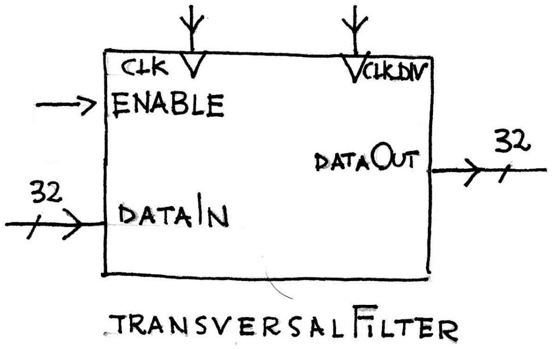

Figure 2.11: Pins configuration of *trasnversalFilter*

The only thing worth mentioning is that *transversalFilter* is definitely the most complex component. In our case the filter order is 256, which means that we have 256 multipliers, an equal amount of registers for the weights and a final adder which must be capable of doing the summation of 256 32 bits long words.

Chapter 3

The testbench

The main purpose of our testbench was that of proving that our hardware would be capable of producing a realistic response to a typical current stimulation. By saying this we mean that we wanted to ascertain whether the neuronal dynamics would respond well to significant change in the parameters of the model, which are the coefficients of the exponentials which make up the dynamic threshold and the time scales of said exponentials.

To this purpose we networked five neurons. We stimulated the network with an external current[1], which was generated in such a way as to simulate the mean field effect of a population of neurons. Four of our hardware neurons were provided with slightly different parameters, while the fifth neuron was given a different set of parameters.

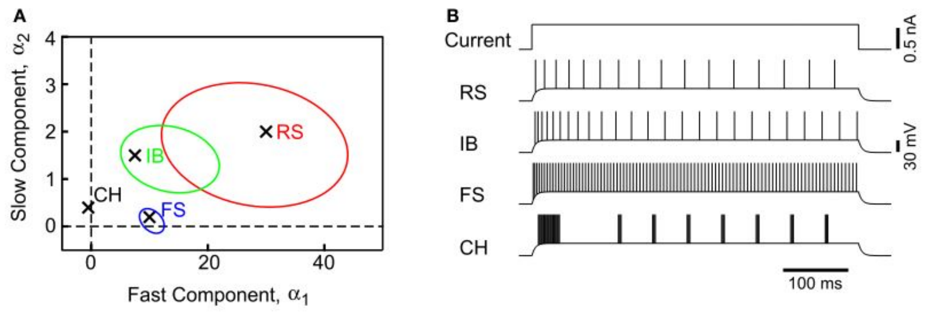

FIGURE 5 | Components of fast and slow dynamics and the model simulations. (A) Distributions of MAT* parameters α_1 and α_2 for RS, IB, and FS neurons are represented as the 1/2 quantiles of the Gaussian distributions fitted to the data in **Figure 3**. **(B)** Spikes generated by a rectangular current of 0.60 nA (500 ms): The parameters of the MAT* model are, RS: α_1 = 30 mV, α_2 = 2.0 mV, ω = 20 mV, IB: α_1 = 7.5 mV, α_2 = 1.5 mV, ω = 19 mV, and FS: α_1 = 10 mV, α_2 = 0.2 mV, ω = 10 mV. It is possible to mimic the chattering phenomenon by choosing a negative value for the fast component, CH: α_1 = –0.5 mV, α_2 = 0.4 mV, ω = 26 mV.

Figure 3.1: Phase space of the parameters and associated firing patterns [8]

As of figure 3.1 we would expect the fifth neuron to present a different firing pattern. By comparing the stereotypical firing illustrated on the right of the figure with the spike timings of our network, as shown in figure 3.2, it is immediately manifest that our implementation is successful in

[1]The data for the external current are the ones used in the Quantitative Single Neuron Modelling Competition of 2009, organized by the INCF. For more details see [1], [7]

simulating the expected neuronal dynamics. In particular we encourage the reader to ask himself to which class the two groups of neurons belong. The answer is that four of the five neurons are regular spiking neurons while the fifth is a fast spiking neuron. If the reader was able to predict the answer by merely observing the timing of our network then this means that our implementation of the MAT must be considered somewhat successful in reproducing the neuronal dynamics.

To conclude our considerations on this first testbench we'd like to stress that, in order to make the contribution of a single neuron of our small network comparable with that of the external current, which accounts for a whole population, the weights W_e and W_i for the clusters of synapses had to be increased to unrealistic levels. Otherwise it would have been impossible to appreciate variations in the timings of the regular firing neurons.

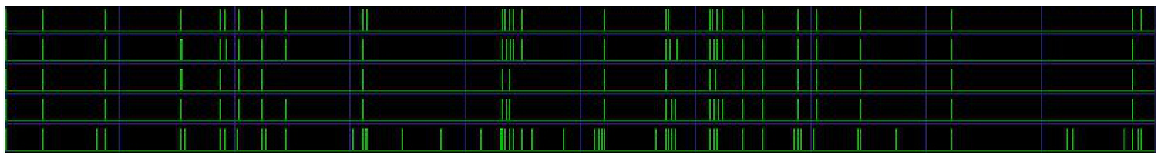

Figure 3.2: One of the five neurons features a markedly different set of parameters. Which one is it?

Another interesting testbench that we conducted was that of comparing the output of our neuron with that of a real neuron[2] stimulated with the same current. In this case the result of the comparison has been somewhat unsatisfying, due mainly to two reasons:

- Firstly the ability of our neuron to predict the spike timings of the real neuron is limited by the underlying mathematical model, the MAT. In the absence of a comparison between the real neuron and a software simulation of the MAT it is impossible to say how many of the observed discrepancies are due to a faulty implementation or to the inherit limits of the MAT.
- More importantly the parameters of our neuron were optimized by hand through a trial-and-error process. Needless to say chances are that the chosen parameters are suboptimal and that a better predictive performance can be obtained by a more analytic approach, as illustrated in many papers [8]

Optimization of the parameters by hand becomes more and more difficult as the runtime of the simulation increases. For this reason we considered only the first 360 ms

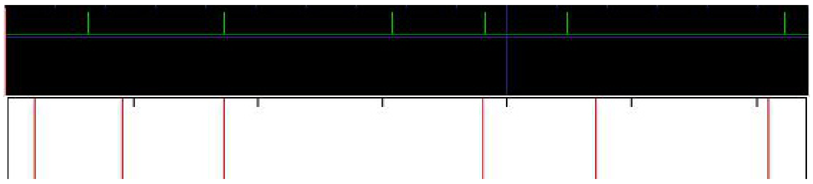

Figure 3.3: Comparison of the firing of a real neuron (bottom) with that of our hardware neuron (top) for the first 360 ms, when the two are subjected to the same external current

[2]More precisely of the primary somatosensory neocortex of a Wistar rat

On final word on the time it takes the simulation to complete. A simulation of 1 ms of length, which corresponds to 2 seconds of simulated neuronal activity, takes one minute to complete if we have only one neuron; five if a network of five neurons is being simulated. Therefore it appears that times scale linearly with the number of neurons.

Some things should be taken in consideration when evaluating the speed of the simulation. Firstly the simulation has been run on a virtual machine. Secondly, and more importantly, the LUT's are read one entry at a time. To the best of our knowledge the read speed of the hard disk where the LUT's are stored might be the limiting factor.

Chapter 4

Future work

Some areas of improvement are easy to identify.

Our circuit is actually a little bit more complex that it needs to. State A of the FSM can be discarded in favour of a momentary raise in the dynamic threshold to account for the effects of refractoriness. This has a double advantage: firstly it allows us to spare a little bit of gates and memory; secondly, and more importantly, there would be even fewer parameters to optimize, since we have eliminated the need for an ARP.

The filters have been implemented using LUT's, but it may be preferable to compute on the fly the impulse response, the reason being that the access times of the LUT's are quite slow while the additional complexity of the computational units should be sustainable as a consequence of our choice to use the fixed point representation.

Needless to say it is of vital importance to know precisely the requirements of our project in terms of gates and memory. Therefore it will be necessary to migrate the project to any one of the integrated developments environments and to test the field programmable gate array (FPGA) core. This will also shed some light on the physical time it takes our circuit to compute the shift in the membrane potential in response to an external current. Knowing this we will be able to formulate a more precise plan for the realization of a population of neurons.

For the time being we will restrict ourselves to some general considerations.

Our objective is that of designing an architecture which will allow us to have a whole population of neurons on one single next generation FPGA. A population is an mesoscopic abstraction well suited to the description of a given layer of a cortical column: it is made of a number of neurons in the order of the thousands; it presents little variation in the parameters of the neurons and its activity can be easily described as in equation (4.1) (N is the number of neurons, n(t) the total number of spikes emitted at time t and Δt is the inverse of the sampling rate of the neuron); neurons of the same populations are also subjected to the same external stimulus, for example the activity of another population.

$$A(t) = \frac{n(t)}{N * \Delta t} \tag{4.1}$$

The main observation is that the neuronal sampling rate is quite slow and we want our architecture to operate in real time. If we assume that the single neuronal circuit that we detailed in this work can be made to function at a rate of the order of the MHz, that would be roughly one thousand times faster than a real neuron (whose sampling rate is circa 5kHz). The idea of our proposal is that of taking advantage of this difference in the speed of computation, so to say, to serialize some of the computational units in order to lessen the requirements of the population in terms of logic gates.

As we noticed the main burden is represented by the transversal filter. In reality we predict that this will be the only component that will be necessary to share across multiple neurons. Having one single transversal filter for the whole population appears infeasible, but ten of them should probably cut it.

As things stand now memory constraints should not be a concern, since we have already determined to dispatch the LUT's in favour of doing the calculations on the fly. It should therefore be possible to have a few registers for each neuron to store the model parameters; this way it should be possible to have an heterogeneous population if the task at hand requires it. It is equally interesting to note that the predictive performance of the MAT is considerably higher than that of a common integrate and fire (IF); in turn this should hopefully result in a more realistic behaviour of the whole population (which is presumably a stringent requirements of the implantable biomimetic chips to be [4]).

The sharing of resources naturally suggests an organization of the neurons in clusters, which are characterized by having only one transversal filter. The neurons of a certain cluster would only talk between themselves, while clusters of neurons would communicate by means of the cluster activity, which would be defined just like the population activity of equation (4.1) (where N is now obviously the number of neurons in a cluster). In other words we are describing a hierarchy whose levels share some characteristics but differ in some others. At the base of such hierarchy there would be the single neuron, with its own parameters, and possibly equipped with the arithmetic units necessary to compute the dynamic threshold, but certainly deprived of the more expensive transversal filter. On a higher level there would be the cluster. For a population of a thousand neurons each cluster should probably have a hundred of them, one single transversal filter and the necessary arithmetic units to compute the current response to the others clusters' activity. This current is to be broadcast to the neurons of the cluster. Connectivity should be rather sparse when compared with that of the clusters themselves. The top level is the population, in this case made of ten clusters. Such a low amount of entities allow to employ full connectivity. The population activity is built treating each cluster as a neuron. Firing is assumed when the cluster activity goes beyond a certain threshold which, once again, could be made different for every cluster. The optimization of the parameters of a certain level of the hierarchy could be made dependant on the values of those at a higher level.

Our hope is that this organization will allow to identify some quantities, like perchance the aforementioned threshold for the cluster activity, which meaningfully link the parameters of the lowest level with the mean field dynamics of the higher ones, so that optimization could eventually be led on the base of observations at the mesoscopic or even macroscopic level.

Bibliography

[1] INCF quantitative single-neuron modeling 2009. `http://staging.incf.org/community/competitions/archive/spike-time-prediction/2009`. Accessed: 2016-06-14.

[2] Renaud Jolivet et al. A benchmark test for a quantitative assessment of simple neuron models, 2008.

[3] Skander Mensi et al. Parameter extraction and classification of three cortical neuron types reveals two distinct adaptation mechanisms. *Journal of Neurophysiology*, 2012.

[4] Theodore W. Berger et al. Brain-implantable biomimetic electronics as the next era in neural prosthetics, 2001.

[5] Wulfram Gerstner. Spike-response model. *Scholarpedia*, 3(12):1343, 2008.

[6] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics*. Cambridge University Press, 2014.

[7] Renaud Jolivet, Felix Schurmann, Thomas K. Berger, Richard Naud, Wulfram Gerstner, and Arnd Roth. The quantitative single-neuron modeling competition. *Biological Cybernetics*, 2008.

[8] Ryota Kobayashi, Yasuhiro Tsubo, and Shigeru Shinomoto. Made to order spiking neuron model equipped with a multi timescale adaptive threshold. *Frontiers in Computational Neuroscience*, 2009.