VHDL implementation of the multi-timescale neuronal model

Rodolfo Rocco
rocco.1494012@studenti.uniroma1.it

22 luglio 2016

SAPIENZA
UNIVERSITÀ DI ROMA

Multi-timescale Adaptive Threshold (model)

Features

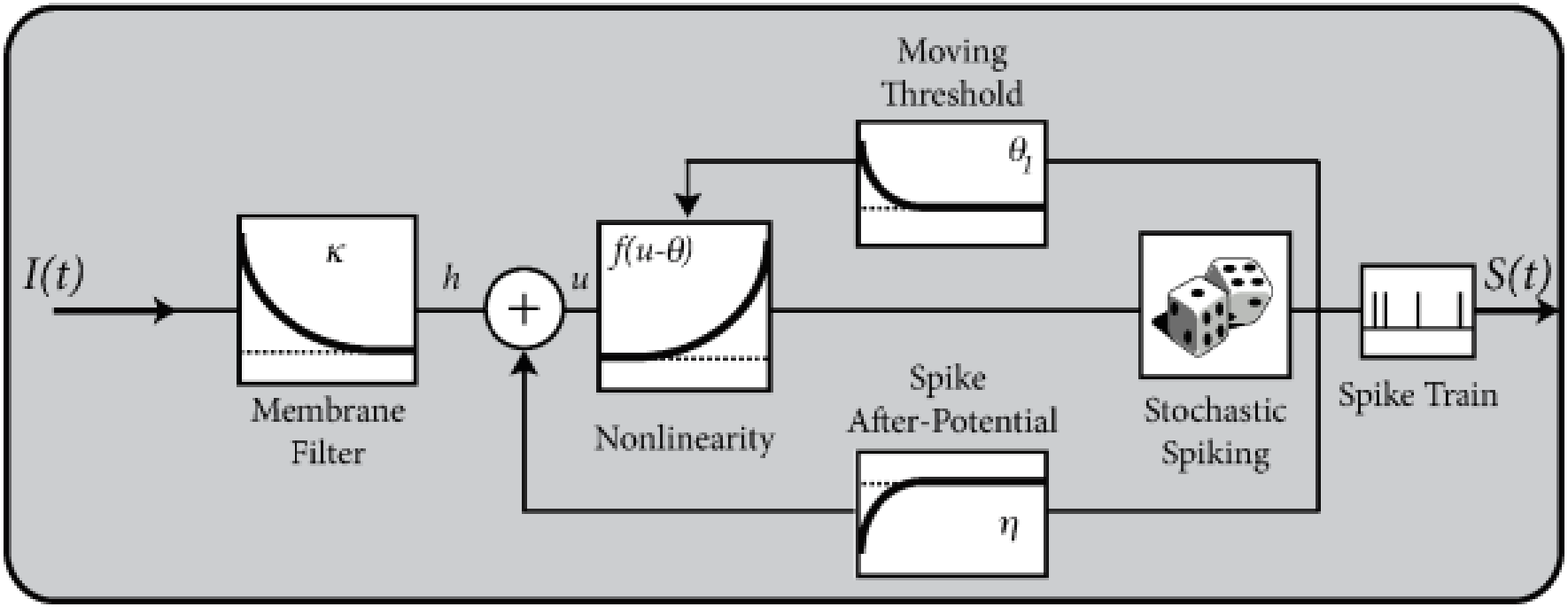
- It is a *phenomenological model* with a *dynamic threshold*
- It is a special kind of the *Spike Response Model*
- It has a threshold described by the sum of *decaying exponentials* with different τs

Dynamic multi-time scale threshold

$$\theta(t) = \theta_0 + \sum_{j=1}^{L} \alpha_j exp\left(-\frac{t - t^f}{\tau_j}\right)$$

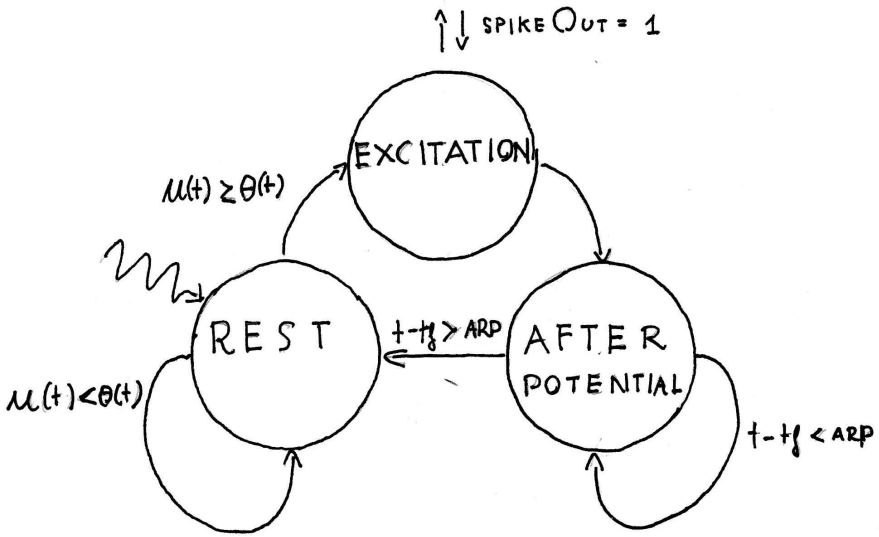
- The MAT was the winner of the *Single Neuron Modelling Competition 2009*, winning against HH (difficult to optimized), LIF (dynamics not complex enough), SRM (similar to MAT but with only one exp)
- If the convolution operation is implemented using LUTs the number of exponentials has no effect on complexity/performance

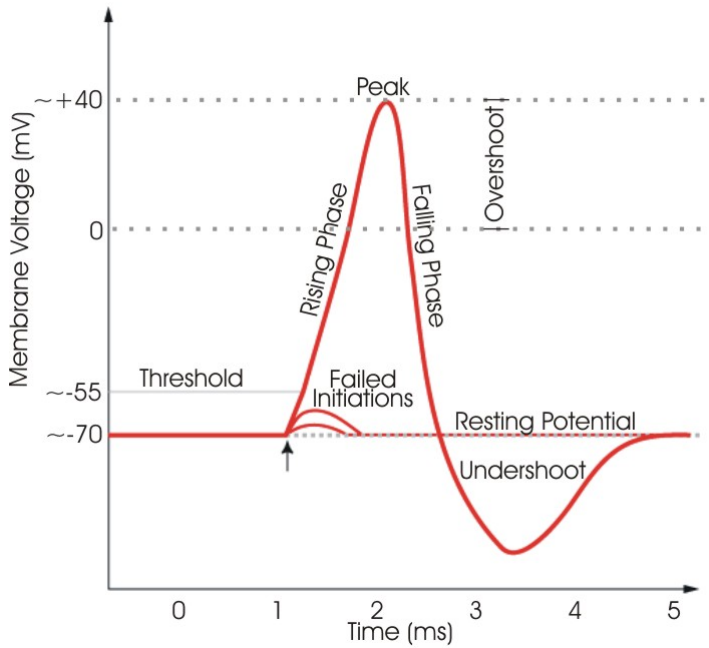
Spike Response Model

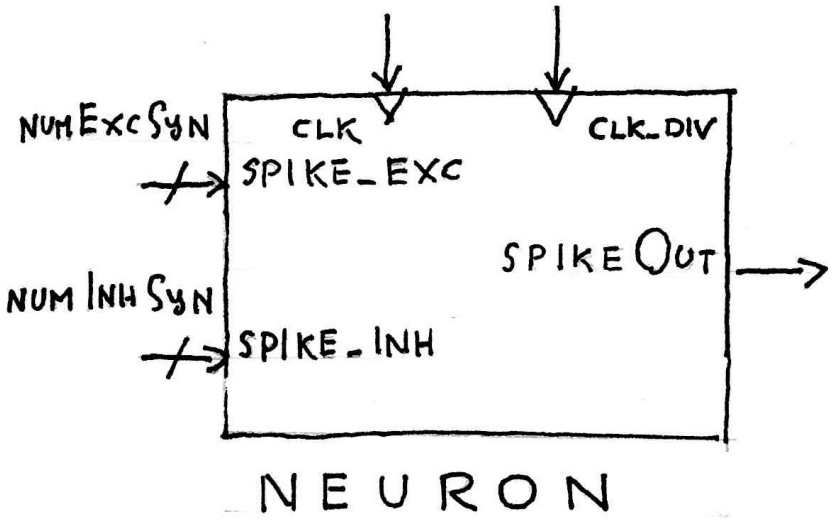


Designing a hardware implementation of the SRM is trivial since the model is given by a succession of filters (the post-potential can be included in the dynamic-threshold)

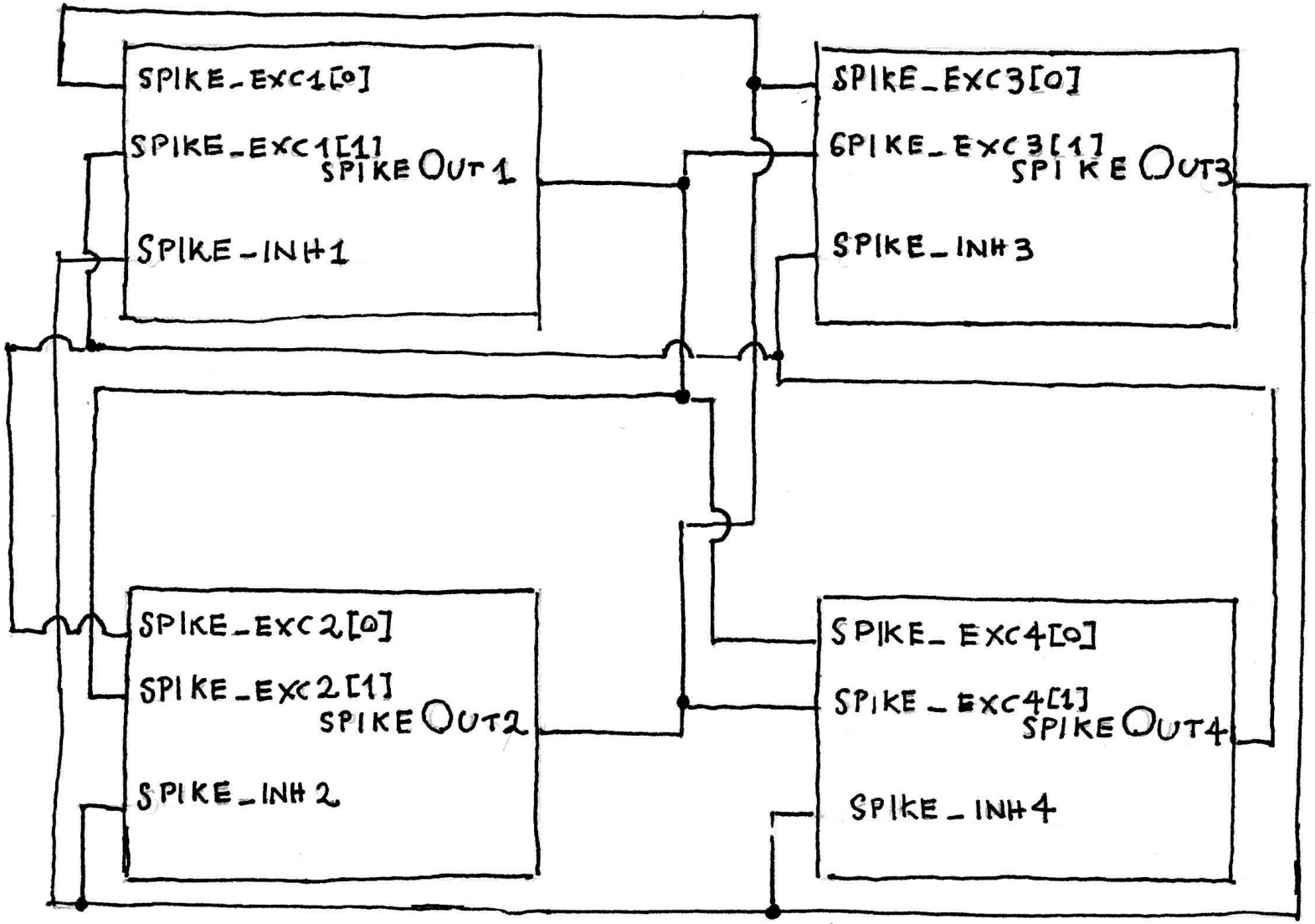
The starting point is **FSM** which describes the essential features of the neuronal dynamics







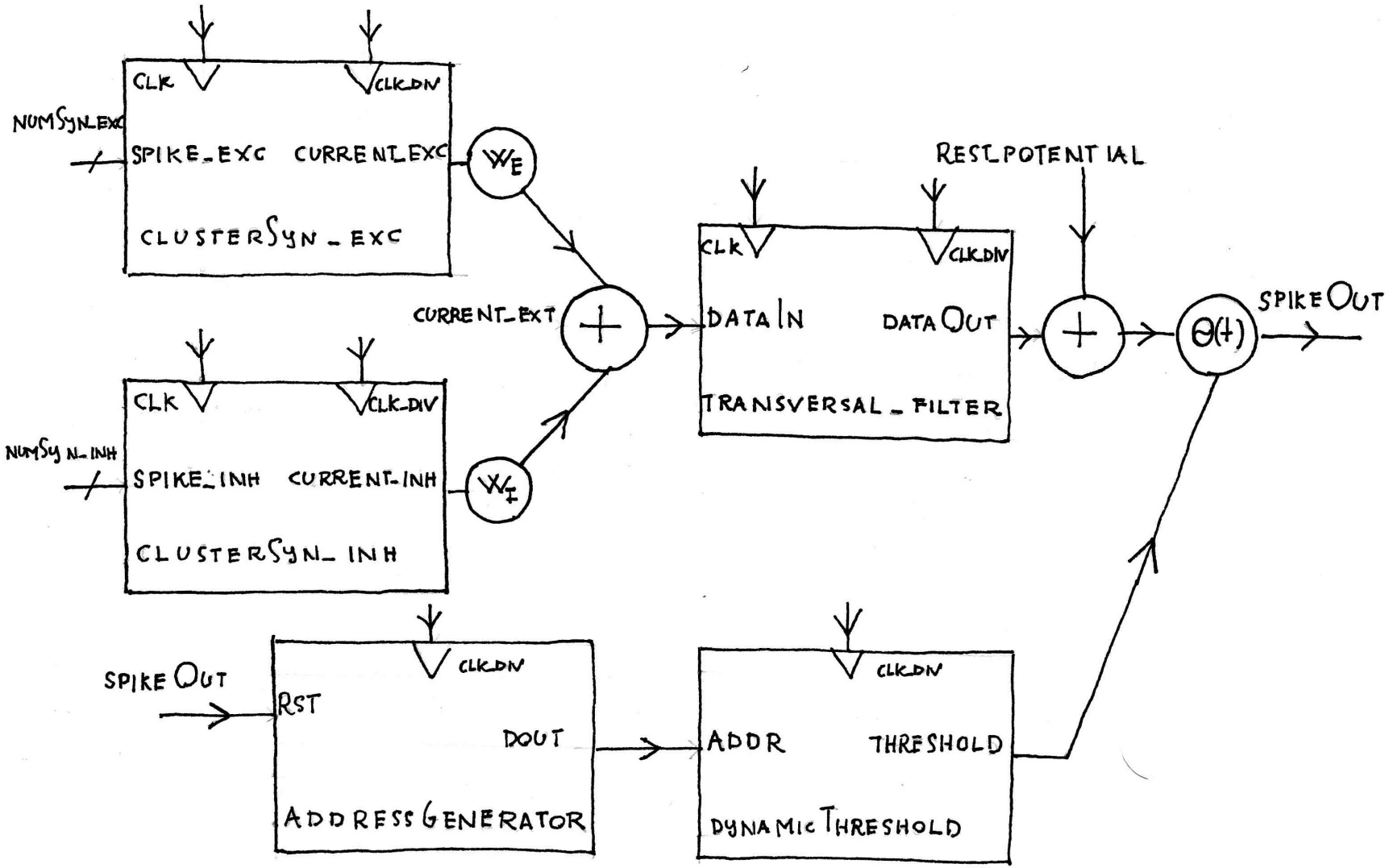
- **spikeOut**: high when the neuron fires
- **spike_exc(inh)**: collects the *spikeOut* of the presynaptic neurons
- **clk_div**: sampling rate of the neurons
- **clk**: frequency of the computing units

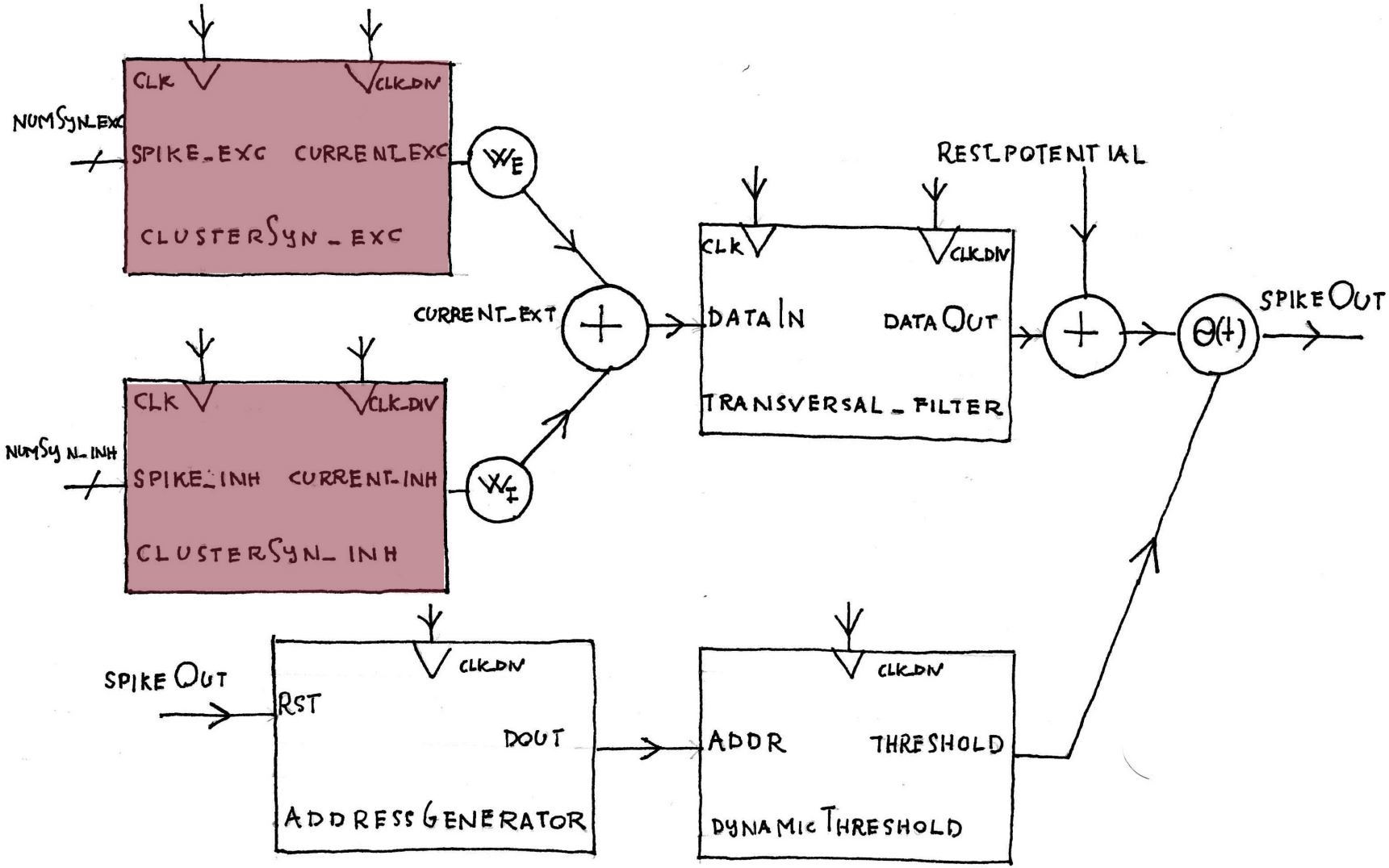

SPIKE_EXC1[0]
SPIKE_EXC1[1]
SPIKE OUT 1
SPIKE_INH1
SPIKE_EXC3[0]
SPIKE_EXC3[1]
SPIKE OUT3
SPIKE_INH3
SPIKE_EXC2[0]
SPIKE_EXC2[1]
SPIKE OUT2
SPIKE_INH2
SPIKE_EXC4[0]
SPIKE_EXC4[1]
SPIKE OUT4
SPIKE_INH4

Some additional infos...

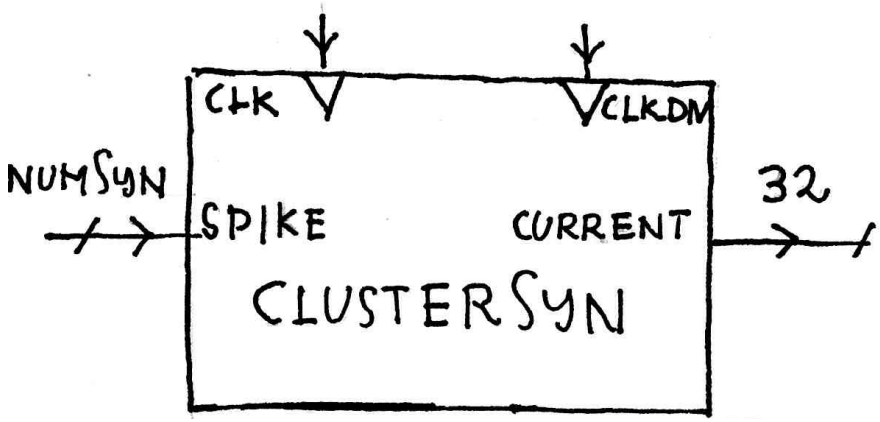
- **Fixed point representation**
  Since there's no interest in describing the potential beyond the threshold, the potential ranges between -70mV and -55mV; currents are within a range of three orders of magnitude
- **32 bits word**
  17 bits for the decimal part, 14 for the integer one. (Barely) adequate precision, 64 bits would be overkill

Spike generation




CLK
CLK_DIV
NUM_SYN_EXC
SPIKE_EXC
CURRENT_EXC
CLUSTER_SYN_EXC
W_E
REST_POTENTIAL
CLK
CLK_DIV
CURRENT_EXT
DATA_IN
DATA_OUT
SPIKE_OUT
Θ(t)
CLK
CLK_DIV
NUM_SYN_INH
SPIKE_INH
CURRENT_INH
W_I
CLUSTER_SYN_INH
TRANSVERSAL_FILTER
SPIKE_OUT
CLK_DIV
RST
DOUT
ADDRESS_GENERATOR
CLK_DIV
ADDR
THRESHOLD
DYNAMIC_THRESHOLD

clusterSyn



Receives as inputs the *spikeOut* signals of the presynaptic neurons, and produces the response current of the postsynaptic neuron

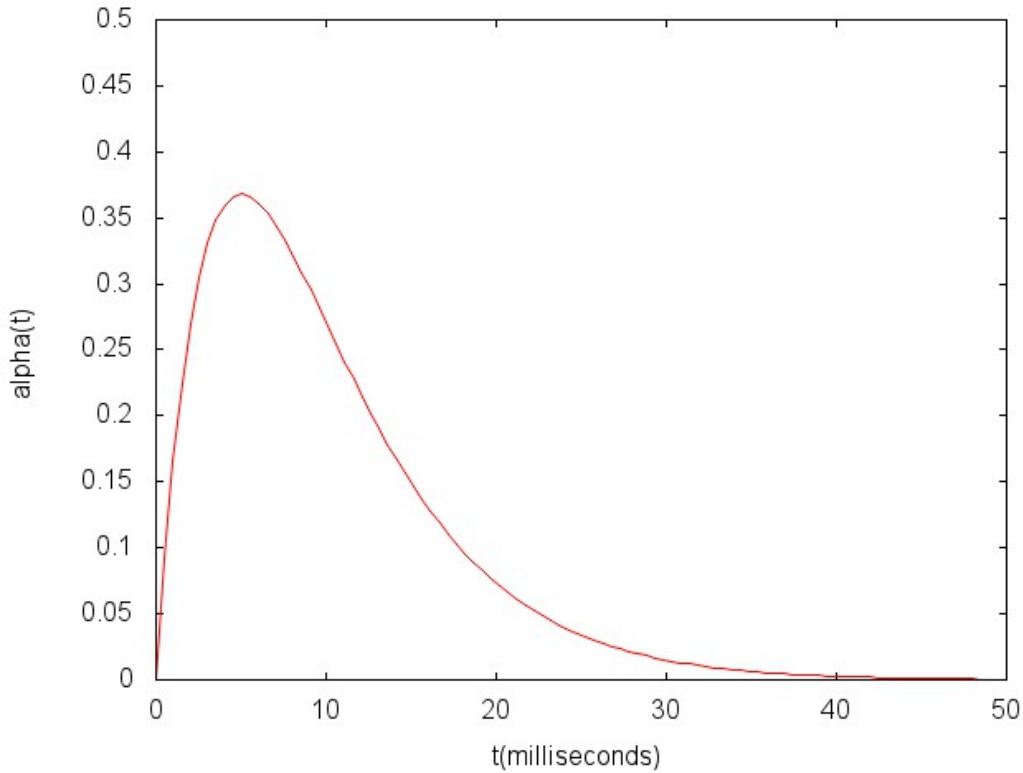
The spikes of the presynaptic neuron induce a response current:

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

$$= \sum_j w_j \alpha(t - t_j^f)$$

- w_j is the synaptic weight j
- $\delta(t - t^f)$is the spike
- $\alpha(t) = \frac{1}{\tau} \exp(-\frac{t}{\tau})$ is the response current to the spike

For each synapse ...



- t is the time elapsed since the reception of the spike
- $\alpha(t)$ is the adimensional response current

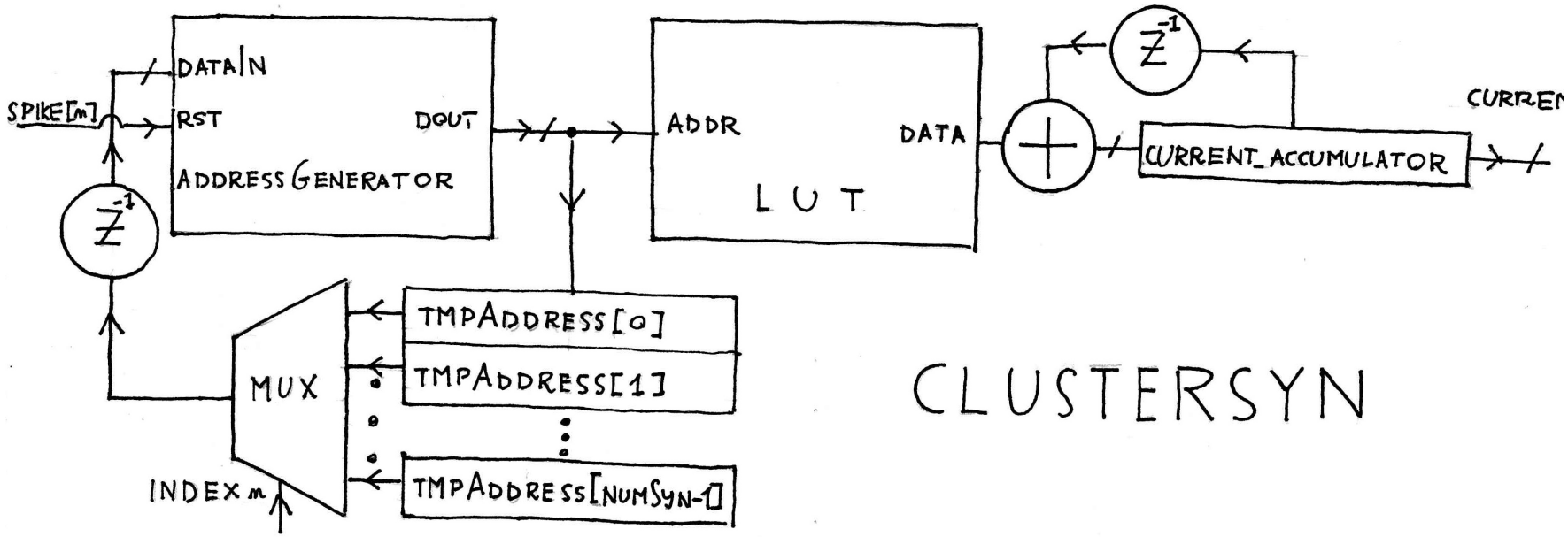
Implementation is straightforward when done with **LUTs** (but access time may be long...)

What do we need?

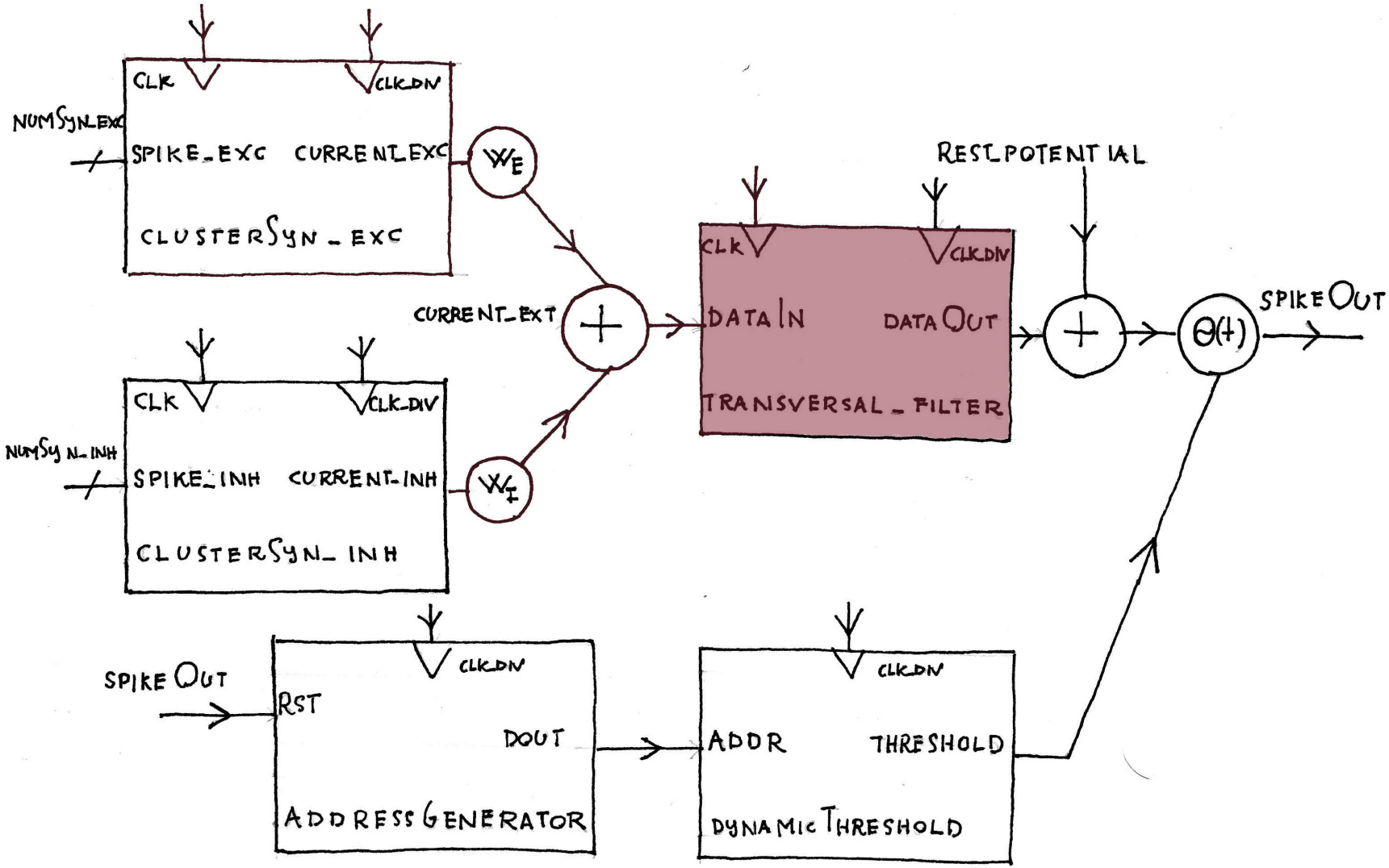
- Addresses generator
- Memory
- Adder for the currents
- Multipliers for the weights

Two approaches

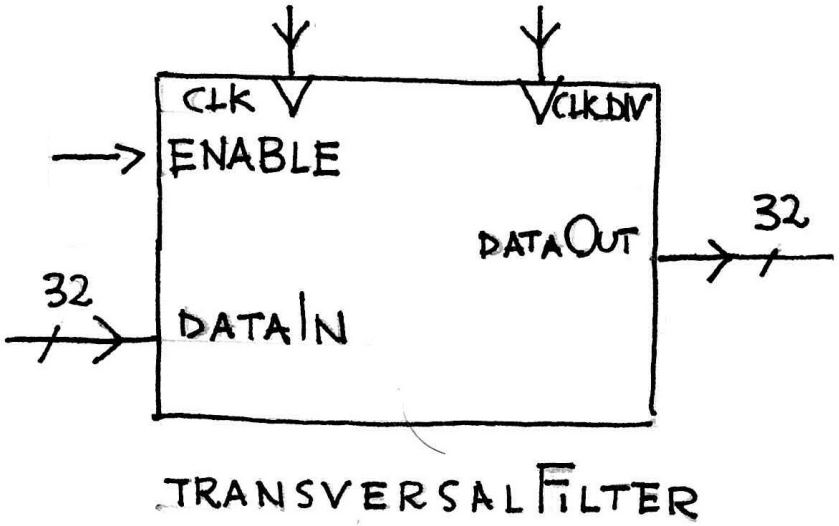
- Parallel synapses: expensive (in terms of resources utilization) but needs only one clock
- Serial synapses: less resources needed but a **faster clock** is needed as well as **additional registers**



1. The address generator is reset every time a new spike is received
2. The address is sent to the LUT and saved in a register
3. The value obtained from the LUT is sent to the adder
4. If no spike arrives the address generator increments the address stored in the register


CLK
CLK_DIV
NUMSYN_EXC
SPIKE_EXC
CURRENT_EXC
CLUSTERSYN_EXC
W_E
CURRENT_EXT
REST_POTENTIAL
CLK
CLK_DIV
DATA IN
DATA OUT
TRANSVERSAL_FILTER
$\Theta(t)$
SPIKE OUT
CLK
CLK_DIV
NUMSYN_INH
SPIKE_INH
CURRENT_INH
CLUSTERSYN_INH
W_I
SPIKE OUT
CLK_DIV
RST
DOUT
ADDRESS GENERATOR
CLK_DIV
ADDR
THRESHOLD
DYNAMIC THRESHOLD

transversal_filter

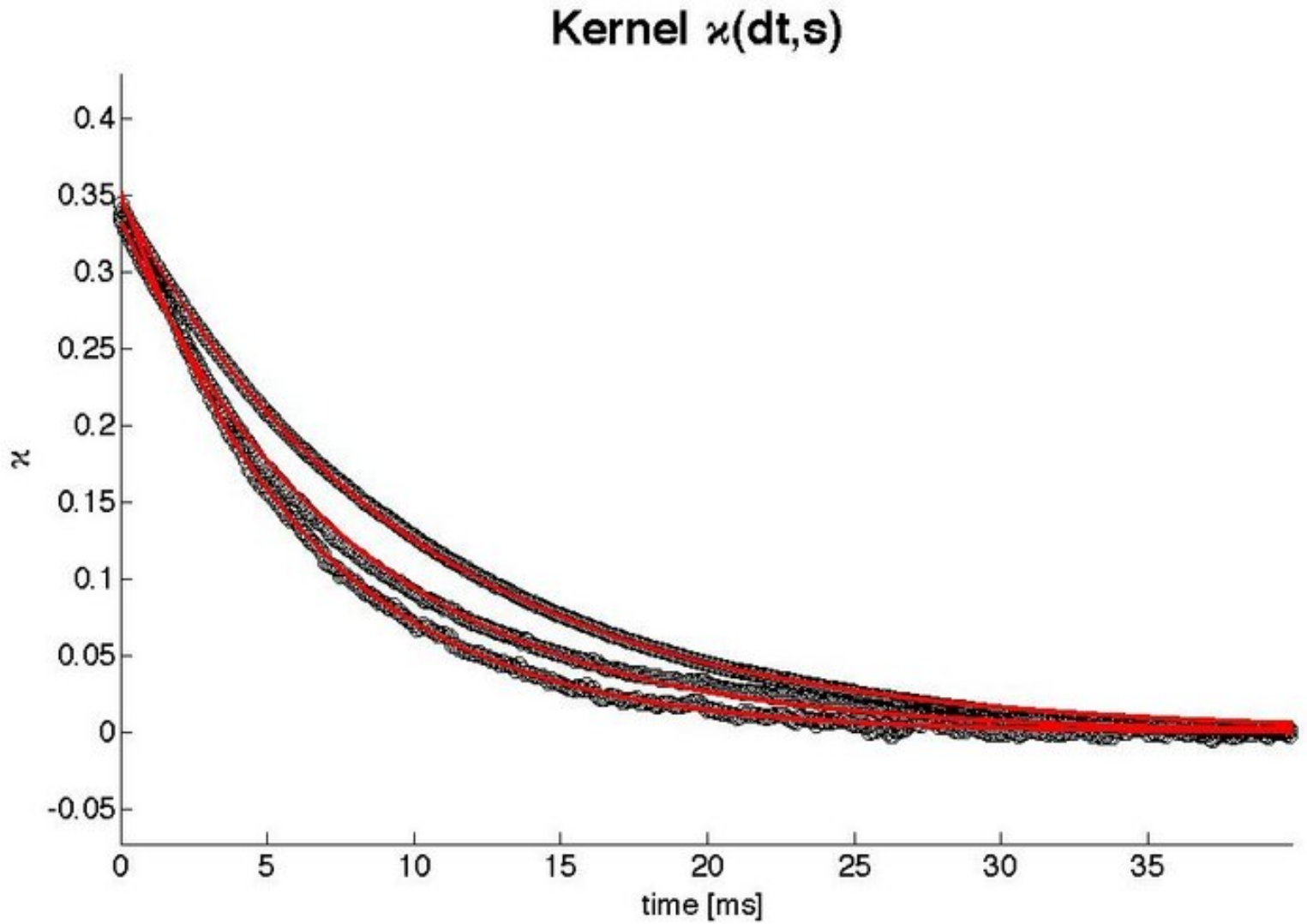


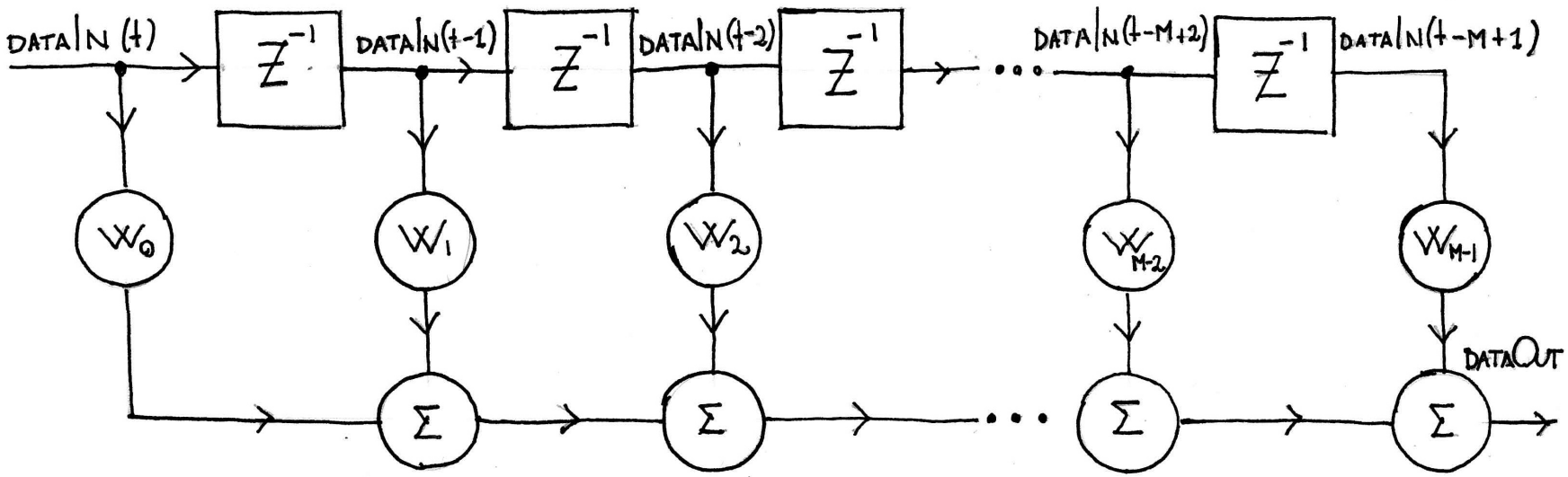
Receives as input the sum of the currents generated by *clusterSyn* and produces the membrane potential

Can be disable (useful during the ARP)

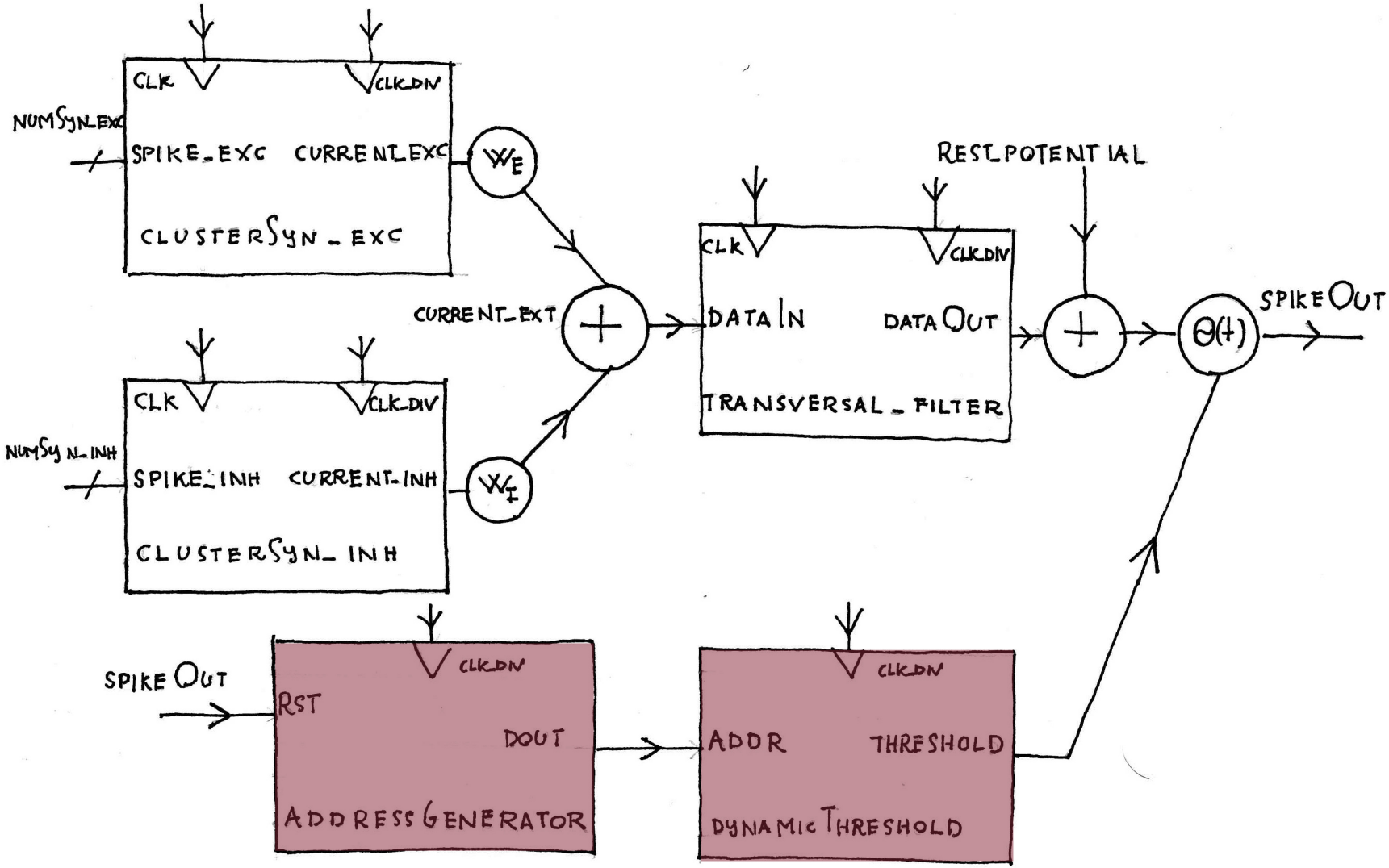
$$u(t) = \sum_j w_j \sum_{s=0}^{N} \kappa(s)\alpha(t - t_j^f - s)$$

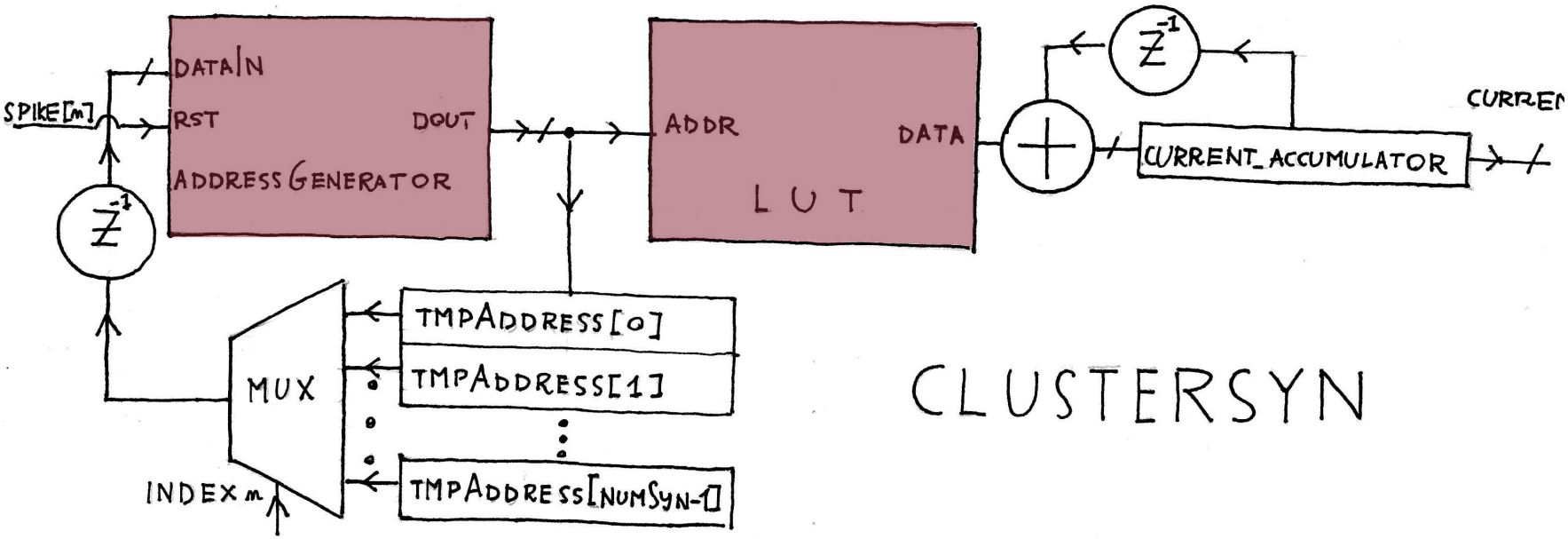
- w_j and α already introduced in *clusterSyn*
- $\kappa(t) = \frac{1}{C}\exp(-t/(\tau_m))$ is the kernel of the filter


Kernel κ(dt,s)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
κ
0
5
10
15
20
25
30
35
time [ms]

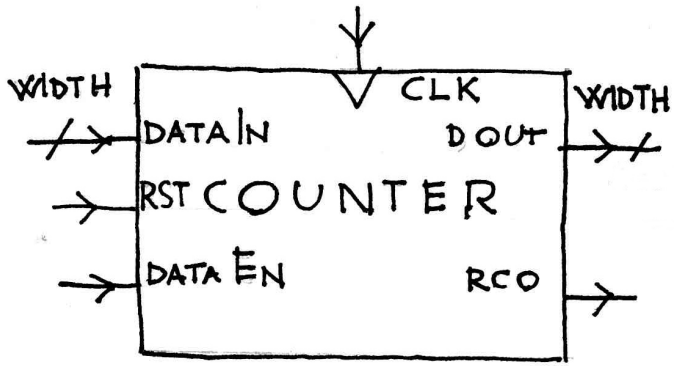


- Filter of order 256
- Fixed point, 32 bits multipliers and adders
- Is the most expensive component

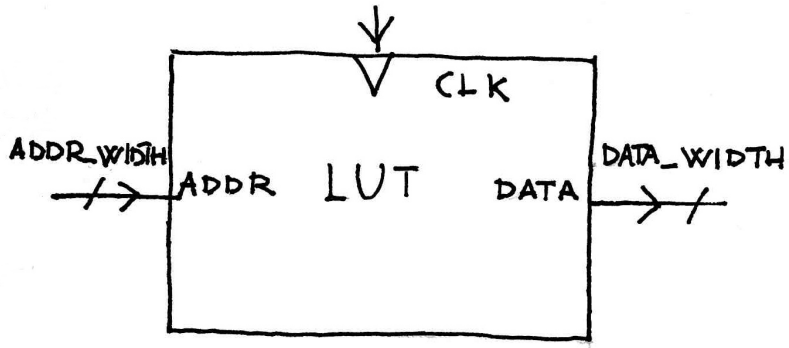

CLK
CLK_DIV
NUMSYN_EXC
SPIKE_EXC
CURRENT_EXC
CLUSTERSYN_EXC
W_E
CURRENT_EXT
REST_POTENTIAL
CLK
CLK_DIV
DATA IN
DATA OUT
TRANSVERSAL_FILTER
$\Theta(t)$
SPIKE OUT
CLK
CLK_DIV
NUMSYN_INH
SPIKE_INH
CURRENT_INH
CLUSTERSYN_INH
W_I
SPIKE OUT
CLK_DIV
RST
DOUT
ADDRESS GENERATOR
CLK_DIV
ADDR
THRESHOLD
DYNAMIC THRESHOLD


DATAIN
SPIKE[m]
RST
DOUT
ADDRESS GENERATOR
ADDR
DATA
LUT
CURRENT_ACCUMULATOR
CURREN
MUX
TMPADDRESS[0]
TMPADDRESS[1]
TMPADDRESS[NUMSYN-1]
INDEX m
CLUSTERSYN

counter & LUT

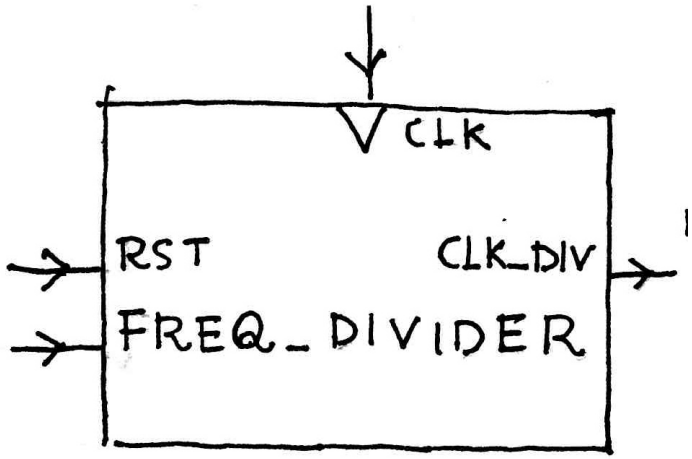


- It's an incrementer
- Can start from any number (*dataIn*)
- Once it reaches the end doesn't restart

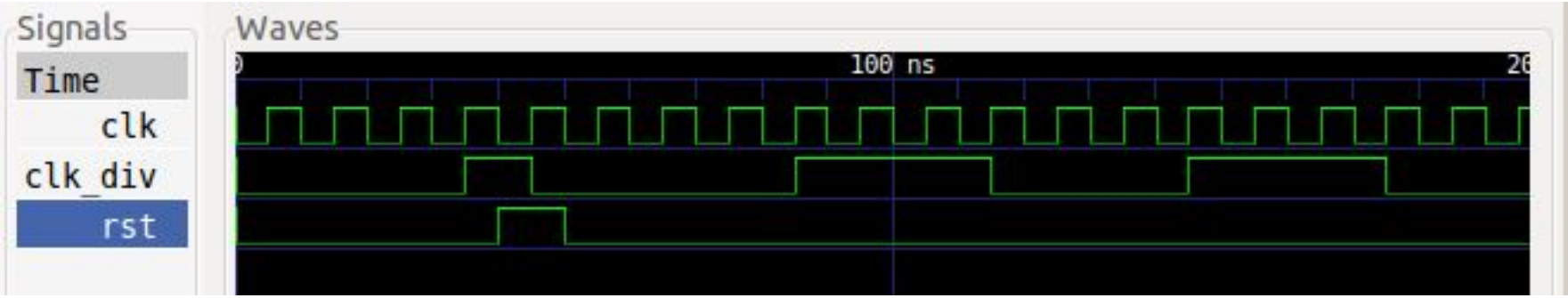


- Just a simple ROM with addresses decoding

frequency_divider



Takes the input clock and generates a clock with a period which is a multiple of that of the input



- **Main objective**

  Verifying the correct working by observing different firing patterns when changing parameters
- **Possibly...**

  Verifying the predictive capabilities of the model

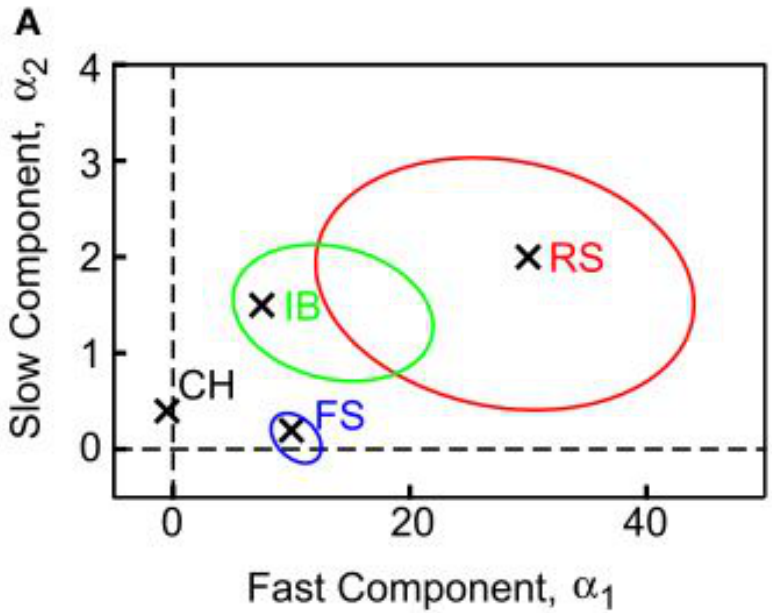
Time scale

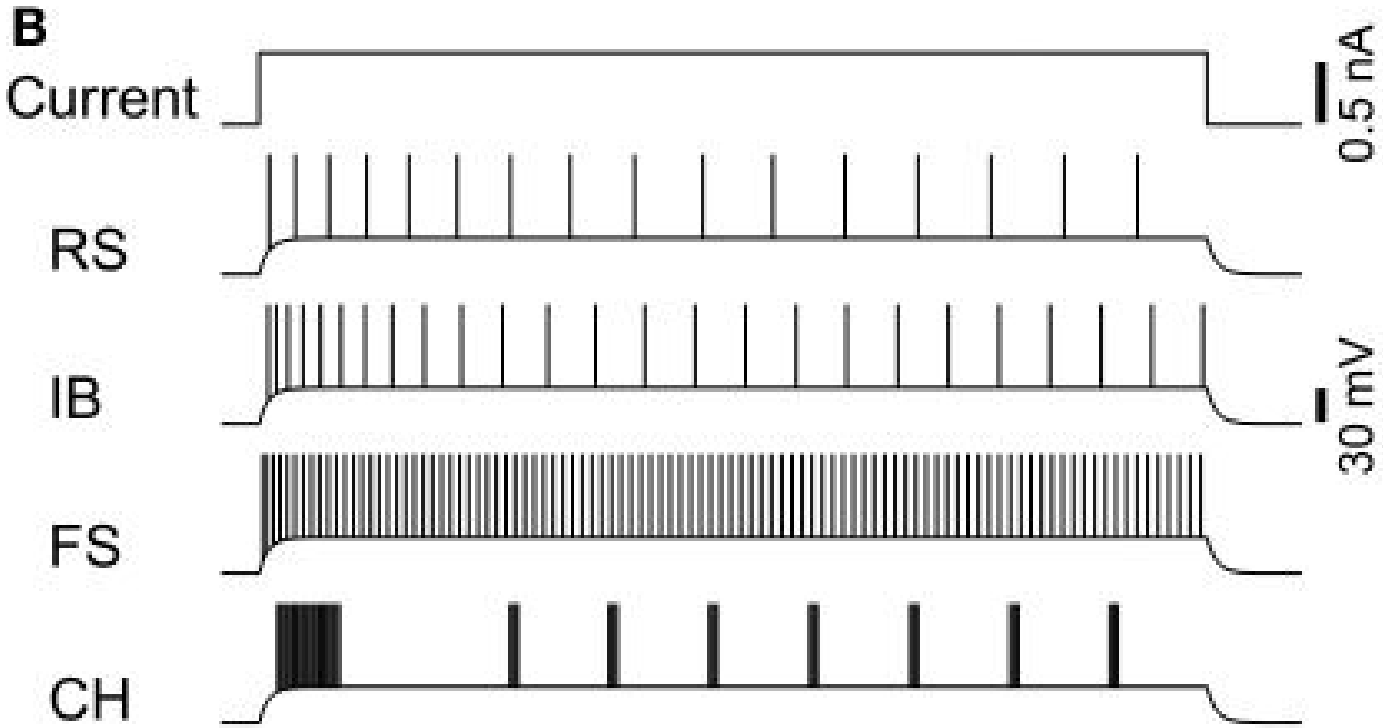
Model	Timescales of adaptive threshold (ms)	Predictive performance Γ_A
MAT(1)	10	0.72 ± 0.25
MAT(1)	50*	0.80 ± 0.23
MAT(1)	200	0.69 ± 0.22
MAT(2)	10, 50	0.82 ± 0.25
MAT(2)	10, 200*	0.89 ± 0.21
MAT(2)	50, 200	0.81 ± 0.22
MAT(3)	10, 50, 200	0.86 ± 0.22
	References	
HH	–	0.51 ± 0.26
LIF	–	0.66 ± 0.26
SRM	50*	0.70 ± 0.26

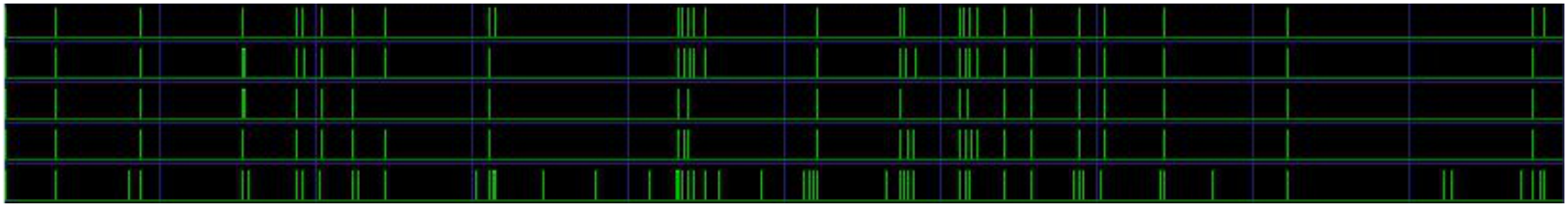
Best performance with $\tau_1 = 10$ mS and $\tau_2 = 200$ mS; however $\tau_2 = 2000$ mS is too slow (LUT gets very big) hence we chose $\tau_2 = 50$ mS (second best choice)

Main testbench

We created a small network of 5 neurons (all-to-all connectivity) and subjected to the same external current. Four of them had the same parameters; the fifth had a different set.

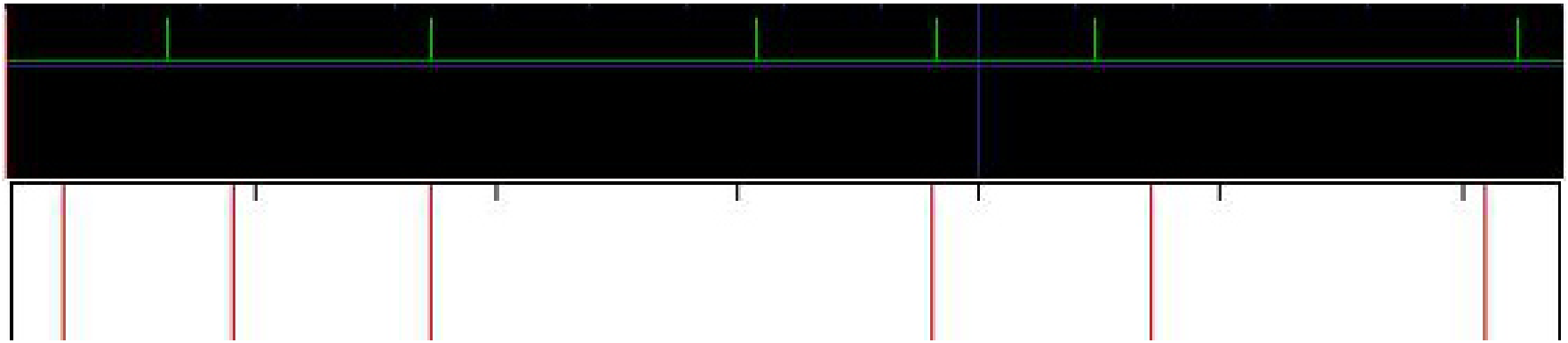



B
Current
0.5 nA
RS
IB
30 mV
FS
CH



Secondary testbench

Comparison of the timings of a hardware neuron with those of a real neuron subjected to the same stimulus



Only partially satisfying ...

- Two spikes correctly guessed
- Three spikes fired too soon or too late
- One spike missed and one spike too many

Possible explainations:

- Inherent limitation of the model
- Sub-optimal parameters
- Wrong implementation...

The equations we used...

$$u(t) = \sum_{j} w_j \sum_{s=0}^{Ntaps} \kappa(s)\alpha(t - t_j^f - s)$$

$$I(t) = \sum_{j}^{numSyn} w_j \alpha(t - t_j^f)$$

$$\theta(t) = \theta_0 + \sum_{i=1}^{Nexp} \alpha_j exp\left(-\frac{t - t^f}{\tau_i}\right)$$

...and those we should have used

$$u(t) = \sum_j w_j \sum_f \sum_{s=0}^{Ntaps} \kappa(s)\alpha(t - t_j^f - s)$$

$$I(t) = \sum_f \sum_j^{numSyn} w_j \alpha(t - t_j^f)$$

$$\theta(t) = \theta_0 + \sum_f \sum_{i=1}^{Nexp} \alpha_j exp\left(-\frac{t - t^f}{\tau_i}\right)$$

We considered only the last **spike** (heavy approximation)

Without approximation the design gets more complex but some simplifications are possible:

- **No multiplications** since the spike train is a sum of deltas: $S(t) = \sum_f \delta(t - t_f)$
- **The number of adders can be lower** that the order of the filter thanks to the ARP ($ARP = 2$ mS vs $\nu = 0.1$ mS)

For a filter of order 256 we need no more than $256 * 0.1/2 \approx 13$ adders