Monostable Multivibrator Networks:
extremely low power inference at the edge with timer neurons

L. Keuninckx\textsuperscript{1}, M. Hartmann\textsuperscript{1}, P. Detterer\textsuperscript{2}, A. Safa\textsuperscript{1,3}, I. Ocket\textsuperscript{1}

\textsuperscript{1} imec Leuven, Belgium; \textsuperscript{2} imec Holst Centre, Eindhoven, The Netherlands; \textsuperscript{3} Department of Electrical Engineering, Katholieke Universiteit Leuven (KUL), Belgium.

What’s all this neuromorphic stuff, anyway?

Energy/inference $\downarrow \Rightarrow$ applications, market opportunities $\uparrow$

- Neurons are living cells **first** and only **then** computational units
- In the end, the **only thing that counts** are the computational properties of the units we employ and how easy we can build & organize them in networks
- What if... **we reverse the question**:

**From:** how to implement neurons (+ synapses) in electronics?

**To:** which known electronic building blocks can be useful as artificial “neuron”?

Model: Hodgkin-Huxley, Fitzhugh-Nagumo, Leaky I&F,...

Neuron-like computational properties?

Known electronic building block
Non-retriggerable* monostable multivibrator
It’s just a timer!

- Two possible states: idle, triggered
- Two inputs: EXCitatory and INHibitory
- If idle and EXC spike in → triggered
- *If triggered and EXC in → ignore
- After \(T\): Spike and return to idle
- If triggered and INH spike in → return to idle, don’t spike
- OR signals together → overlapping input spikes? Don’t care!
- “Non-biologically inspired spiking neuron”
Motivation: MMV = just a counter in digital hardware

Earlier work:

- Dynamical behavior of MMV networks: rate equations, equilibria of recurrent networks...
- Rate-based backpropagation
- Many fun problems for mathematicians & physicists 😊

- #MMVs ≈ #Flipflops/\log_2(T_{\text{max}})
- Artix XC7A100T FPGA (ARTY-board) → ~10k
- ~100s of MMVs in a recurrent network:

“Monostable multivibrators as novel artificial neurons”, Neural Networks, 2018.

🔍 How to put MMVs to good use in event-driven networks to actually solve problems?
Example: a spike-interval discriminator

- Spike out if input spike interval between Tmin, Tmax
- Conditions: \(d > \text{Tmax} - \text{Tmin}, \text{Tmin} > \text{Tmax}/2\)
- But...how to train large event-driven networks?
Goal: Optimized Recurrent MMV Networks

- OR-ing type crossbar
- No synaptic addition in the network
- Overlapping spikes? We don’t care!
- How to find connections and T’s?

Look Mom, No multipliers! No adders!
Training method overview

**MMV model:**
- Backpropagation with surrogate gradients → “get over” step-function discontinuity
- Spike, EXC, INH = 3 separate step-functions needed
- Use multiplicative integration rather than additive timing process → retain insensitivity to long T

**Network:**
- Separate EXC/INH inputs and handle as events via surrogate gradients → handle overlaps/OR-ed signals
- Slow binarization of connections & rounding of periods over many epochs → force connections \{0, 1\} vs. real-valued weights

→ ALL elements are needed for the training to work!
TRICK #1: the NRMMV model
Trigger, Reset, Spike = 3 discontinuous events

<table>
<thead>
<tr>
<th>Activation function:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm 1 Activation Function</strong></td>
</tr>
<tr>
<td>FORWARD_ACT(x)</td>
</tr>
<tr>
<td>save x as context</td>
</tr>
<tr>
<td>return 1 if x &gt; 0, else 0</td>
</tr>
<tr>
<td>BACKWARD_ACT(g)</td>
</tr>
<tr>
<td>hyperparameters: amplitude: A, spread: σ</td>
</tr>
<tr>
<td>x ← retrieve context</td>
</tr>
<tr>
<td>return g · A exp (-x^2/σ^2)</td>
</tr>
</tbody>
</table>

**NRMMV:**

- Separate EXC/INH inputs
- Multiplying accumulator
- EXC and INH are separate EVENTS
- Processing order determines “corner cases”

**Surrogate gradient**

**Leaky Integrate&Fire:**

<table>
<thead>
<tr>
<th>Algorithm 2 Leaky Integrate &amp; Fire Neuron</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE_LIF_STATE(a)</td>
</tr>
<tr>
<td>hyperparameters: leakiness: α ≤ 1, threshold: L</td>
</tr>
<tr>
<td>state ← α · state + a</td>
</tr>
<tr>
<td>spike = FORWARD_ACT(state - L)</td>
</tr>
<tr>
<td>state ← 0 if spike = 1, else state</td>
</tr>
<tr>
<td>return spike</td>
</tr>
</tbody>
</table>

**True adder needed!**
TRICK #1½ : multiplicative vs additive accumulation*

Preserve gradient sensitivity to input spikes in the past

From PyTorch Autograd

```python
# Multiplicative:
# m = 1.0
# dm/dx = 0.99999988807907104 --> keeps sensitivity!
# dm/dT = -0.09531016647815704 --> -ln(ALPHA)

# Additive:
# m = 1.0000001192092896
# dm/dx = 0.0500000074505806 --> 1/T, less sensitive!
# dm/dT = -0.0500000074505806 --> 1/T
```

*This is only for the model

Unrelated to the hardware implementation!
TRICK #2: network model

An MMV cannot distinguish between 1, 2 or more simultaneous EXC/INH inputs!
There is NO addition of synaptic inputs
Hence NO adders are needed in the network!
Synapses are simple OR-operations

start with random real-valued weight matrices
Split EXC (>0) and INH (<0) weights
Process separately, inputs are positive, Equivalent to #activated lines

Algorithm 4 MMV network propagation

NET_FORWARD(input_spikes, internal_spikes)
parameter: connections $C_{in}, C_{net}$

split:
$C_{in}^{EXC}, C_{in}^{INH} \leftarrow \text{Eqs. (9,10)}$
$C_{net}^{EXC}, C_{net}^{INH} \leftarrow \text{Eqs. (9,10)}$

calculate activations:
$act_{exc} = C_{in}^{EXC} \cdot \text{input_spikes} + C_{in}^{EXC} \cdot \text{internal_spikes}$
$act_{inh} = C_{in}^{INH} \cdot \text{input_spikes} + C_{in}^{INH} \cdot \text{internal_spikes}$

calculate new internal spikes:
internal_spikes =
UPDATE_MMV_STATE(act_exc, act_inh)

return internal_spikes
TRICK #3: slow weight binarization & period rounding

- Slowly “crystalize” the weights over many epochs, $\gamma$: 1→0, to connections
- Forward step: use partially quantized weights and rounded periods
- Backward and update step: operate on full precision $C$

$$C_{i,j} \left\{ \begin{array}{ll}
\gamma C_{i,j} & \text{if } -\frac{1}{2} \leq C_{i,j} \leq \frac{1}{2}, \\
\gamma C_{i,j} + 1 - \gamma & \text{if } C_{i,j} > \frac{1}{2}, \\
\gamma C_{i,j} - 1 + \gamma & \text{if } C_{i,j} < -\frac{1}{2}, 
\end{array} \right.$$ 

$$T_i \leftarrow \gamma T_i + (1 - \gamma) \max \{1, \lfloor T_i \rfloor \}$$

![Graph showing output weights vs. input weights for different values of $\gamma$.](image1)

![Graph showing output periods vs. input periods for different values of $\gamma$.](image2)
MNIST handwritten digits

- Threshold pixels at 0.5
- Stream in row or column-wise
- 500 NRMMVs
- Count spikes
- Linear readout: ~98.6%
- Population count readout: ~97.3%
MNIST digits - some network metrics

Spike pattern

Period distribution

Network connections

Spikes/inference distribution

Input connections
MNIST handwritten digits

- Streaming in rows & columns
- $2 \times 500$ NRMMVs + linear readout $1000 \rightarrow 10$
- Data augment train set every epoch: shift $\pm 2$, rotate $\pm 15^\circ \rightarrow \sim 99\%$
Yin-Yang segmentation task

A nicer “XOR”

- Recurrent NRMMV net + linear readout layer
- Direct binary encoding $[0,1] \rightarrow 0.2^{12} - 1$ (12 bits)
- $82 \pm 24$ spikes, active connections: ~26%

<table>
<thead>
<tr>
<th>Network size</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>96.26</td>
</tr>
<tr>
<td>200</td>
<td>98.43</td>
</tr>
<tr>
<td>500</td>
<td>99.16</td>
</tr>
</tbody>
</table>
Google Soli radar gestures

- Input: $32 \times 32$ Range-Doppler maps $[0..1]$, 25ms/map, 11 gestures
- Preprocessing:
  - limit input image to first 384 pixels
  - #timesteps (frames) rescaled from 28...145 → 50
  - Any value $>0$ → spike in

Google Soli radar gestures

- Stream in recurrent NRMMV net + linear readout
- 10× cross validated

<table>
<thead>
<tr>
<th>Reference</th>
<th>Network/Algorithm</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>100 MMVs + linear readout</td>
<td>97.62 ± 0.33</td>
</tr>
<tr>
<td>This work</td>
<td>250 MMVs + linear readout</td>
<td>98.01 ± 0.26</td>
</tr>
<tr>
<td>This work</td>
<td>500 MMVs + linear readout</td>
<td>98.32 ± 0.29</td>
</tr>
<tr>
<td>Yin et al., Nat. Mach. Intell 2021</td>
<td>Spiking RNN</td>
<td>91.9</td>
</tr>
<tr>
<td>Tsang et al., MDPI Electronics 2021</td>
<td>LSM with 460 neurons</td>
<td>98.6 ± 0.7</td>
</tr>
<tr>
<td>Wang et al., UIST 2016</td>
<td>CNN+RNN</td>
<td>87.17</td>
</tr>
</tbody>
</table>
IBM DVS128 gestures

- Group events in frames of 50ms
- \( \text{pol}=1 \rightarrow \text{pixel}++ \), \( \text{pol}=0 \rightarrow \text{pixel}-- \), clamp at 0, threshold to spikes
- 2.0s/gesture, 40 frames, 0.25s hop, 32x32 pixels \( \rightarrow \) flatten to 1024 wide x 40 timesteps
- 22347 train, 5870 test gestures

https://research.ibm.com/interactive/dvsgesture/
IBM DVS128 gestures

- 500 NRMMVs + linear readout
- Data augmentation: shift ±3 pixels, rotate ±15°
- 22347 train, 5870 test gestures

<table>
<thead>
<tr>
<th>Reference</th>
<th>Network/Algorithm</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>500 MMVs + linear readout</td>
<td>92.91</td>
</tr>
<tr>
<td>Safa et al., 2022</td>
<td>LIF+STDP</td>
<td>92.5</td>
</tr>
<tr>
<td>Amir et al., IEEE CVPR 2017</td>
<td>CNN</td>
<td>91.77</td>
</tr>
<tr>
<td>Amir et al., IEEE CVPR 2017</td>
<td>CNN+postprocessing</td>
<td>94.59</td>
</tr>
<tr>
<td>Samadzadeh et al., 2021</td>
<td>Spiking ResNet</td>
<td>96.7</td>
</tr>
</tbody>
</table>

Reference Network/Algorithm Accuracy [%]

<table>
<thead>
<tr>
<th>Reference</th>
<th>Network/Algorithm</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safa et al., 2022, arXiv.2111.00791</td>
<td>LIF+STDP</td>
<td>92.5</td>
</tr>
<tr>
<td>Amir et al., IEEE CVPR 2017</td>
<td>CNN</td>
<td>91.77</td>
</tr>
<tr>
<td>Amir et al., IEEE CVPR 2017</td>
<td>CNN+postprocessing</td>
<td>94.59</td>
</tr>
</tbody>
</table>
MNIST digits - energy/inference

- PyTorch → Verilog
- Mapping to TSMC 28nm HPC+, 500 MMV recurrent reconfigurable network with D-FF & OR-ing trees: **855pJ/inf.**
- Linear/population count readout *guesstimate* using figures from another project: 2.8pJ/MAC, 1.4pJ/ADD, BF16 format, 100MHz

<table>
<thead>
<tr>
<th>Reference</th>
<th>Method/Conditions</th>
<th>Accuracy [%]</th>
<th>Time per inference</th>
<th>Energy per inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>500 MMVs → linear readout, 28nm TSMC HPC+</td>
<td>98.61</td>
<td>330ns + t_readout = 330ns + (500x10 + 10)x10ns = 50.43μs</td>
<td>855pJ + E_readout = 855pJ + 10(500x2.8pJ + 1.4pJ) = 14.9nJ</td>
</tr>
<tr>
<td>This work</td>
<td>500 MMVs → population coding readout, 28nm TSMC HPC+</td>
<td>97.32</td>
<td>330ns + t_readout = 330ns + 500x10ns = 5.33μs</td>
<td>855pJ + E_readout = 855pJ + 500x1.4pJ = 1.56nJ</td>
</tr>
<tr>
<td>Frenkel et al., ISCAS 2020</td>
<td>SPOON, 28 nm event-driven CNN</td>
<td>97.5</td>
<td>117μs</td>
<td>313nJ</td>
</tr>
<tr>
<td>Park et al., ISCC 2019</td>
<td>200+200+10 SNN, 28 nm</td>
<td>97.83</td>
<td>10μs</td>
<td>236.5nJ</td>
</tr>
<tr>
<td>Chen et al., IEEE Symp. VLSI 2018</td>
<td>4096-neuron, 1M synapse SNN, 10 nm</td>
<td>97.9</td>
<td>not given</td>
<td>1.7μJ</td>
</tr>
<tr>
<td>Stuyt et al., Front. Neurosci 2021</td>
<td>“μBrain”, 256+64+16 SNN, 40 nm</td>
<td>91.7</td>
<td>4.2ms</td>
<td>308nJ</td>
</tr>
</tbody>
</table>
Take Home Messages

- MMVs = non-biologically inspired spiking “neurons” = simple timers

- MMV networks can be trained with PyTorch and implemented in digital hardware
  They work by setting up and testing timing conditions

- Extremely low energy/inference due OR-ing style interconnect instead of synaptic addition, at good accuracy, with current technology

- Many things still to be discovered: other types of MMVs, heterogenous MMV nets, MMV based CNNs, evolutionary optimization...
Not Biologically Inspired: 
On Training Networks of Monostable Multivibrator Timer Neurons

Lars Keuninckx, Matthias Hartmann, Paul Detterer, Ali Safa, Ilja Ocket

Abstract—Monostable multivibrators are simple timers which are easily implemented using counters in digital hardware and can be interpreted as non-biologically inspired spiking neurons. We show how fully binarized event-driven recurrent networks of monostable multivibrators can be trained to solve classification tasks. We mitigate an important bottleneck in neuromorphic hardware concepts by circumventing synaptic addition within the network. Here rather, input signals to a neuron are simply ORed together. Temporally overlapping input events are resolved at the neuron level. We demonstrate our approach on the MNIST handwritten digits, Google Sali radar gestures, IBM DVS128 gestures and Yin-Yang classification tasks, all with excellent results. The estimated energy consumption for the MNIST handwritten digits task, excluding the final readout layer, is 855pJ per inference for a test accuracy of 98.61% for a reconfigurable network of 500 units that was mapped to a 28nm process.

and probably even more unknown constraints imposed by their biological reality. Biological neurons are never only computational units and therefore may be limited in their computational abilities in ways their artificial counterparts needn’t be, but actually might become if we push the analogy too far.

In Ref. [6], the original bio-inspired question was reversed from “how can we build a system that somewhat behaves like a brain?” to “which fundamental building blocks that are easy to build and network in large quantities do we have at our disposal?”. As said there, whether such a unit should be called a neuron or not, is merely a matter of semantics.

We are only concerned with its computational or “neuronal” properties. The authors of Ref. [6] investigate the monostable

http://dx.doi.org/10.13140/RG.2.2.27417.70242
Thanks!

Ilja Ocket  Paul Detterer  Ali Safa  Matthias Hartmann


Copyright Notice

This presentation in this publication was presented as a tinyML® EMEA Innovation Forum. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyml.org