Highlights

On Training Networks of Monostable Multivibrator Timer Neurons

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

- We explore the use of monostable multivibrators, which are simple timers, as non biologically-inspired spiking neurons and present a training algorithm for classification tasks.

- The timer neurons have only two digital inputs, excitatory and inhibitory and only one trainable parameter, the timer period, making them readily implementable in digital hardware.

- Signals to the input of a timer neuron are logically OR-ed together, thus avoiding synaptic addition within the network.

- We benchmark our training algorithm on the MNIST handwritten digits, Google Soli radar gestures, IBM DVS128 gestures, and Yin-Yang classification tasks. For the MNIST task, the estimated energy consumption is 855pJ per inference, excluding the linear readout layer, for a test accuracy of 98.61% with a reconfigurable network of 500 timer neurons mapped on a 28nm process.

On Training Networks of Monostable Multivibrator Timer Neurons

Lars Keuninckx[a,*], Matthias Hartmann[a], Paul Detterer[b], Ali Safa[a,d], Wout Mommen[a,c], Ilja Ocket[a]

[a] *Interuniversity Microelectronics Centre (imec), Kapeldreef 75, Leuven, 3001, Belgium*
[b] *imec Holst Centre, High Tech Campus 31, Eindhoven, 5656 AE, The Netherlands*
[c] *Vrije Universiteit Brussel, Pleinlaan 2, Elsene, 1050, Belgium*
[d] *Hamad Bin Khalifa University, Al Rayyan, Qatar*

Abstract

An important bottleneck in present-day neuromorphic hardware is its reliance on synaptic addition, which limits the achievable degree of parallelization and thus processing throughput. We present a network of monostable multivibrator timers, whose synaptic inputs are simply OR-ed together, thus mitigating the synaptic addition bottleneck. 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. Our training algorithm resolves temporally overlapping input events. We demonstrate our approach on the MNIST handwritten digits, Google Soli radar gestures, IBM DVS128 gestures and Yin-Yang classification tasks. The estimated energy consumption for the MNIST handwritten digits task, excluding the final linear readout layer, is 855pJ per inference for a test accuracy of 98.61% for a reconfigurable network of 500 units, when mapped to the TSMC HPC+ 28nm process.

Keywords: spiking neural networks, neuromorphic, recurrent networks, monostable multivibrators, edge computing

1. Introduction

Neuromorphic systems, often understood as hardware performing computations via spiking neurons, have gained much attention during the past decade (Schuman et al., 2020; Monroe, 2014; Mead, 1990). This interest is driven by the evergrowing need for low power and fast inference at the edge.

It is known that the brain with its 86 billion neurons (von Bartheld et al., 2016; Herculano-Houzel, 2009) operates very efficiently at only approximately 20 W, while performing remarkable computational feats. The main enabler of this energy efficiency, it is believed, is event-driven computation by spiking neurons: if nothing interesting happens, nothing needs to be activated.

The field of bio-inspired computing now puts much of its focus on spiking neural networks (SNNs) built around some variation of the (leaky) integrate and fire (LIF) neuron. This makes sense: by abstracting a presynaptic spike x_i to an all-or-nothing binary event, the input activation sum $a = \sum_i w_i x_i$ is rid of multiplication, thus saving chip area and power. Introducing statefulness in the neuron model also enables temporal dependencies in the computation.

Biological neurons are first and foremost living cells, that have to operate within many constraints imposed by their biological reality. Not being *only* computational units, they may be limited in their computational abilities in ways that artificial neurons needn't be. Hence, deriving too much inspiration from nature, might result in missed opportunities.

In (Keuninckx et al., 2018), 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 (Keuninckx et al., 2018) investigate the monostable multivibrator (MMV) timer. This well-known electronic structure is implementable in digital hardware using a counter. Next to many dynamical properties of MMV networks, a rate-based classifier with fixed connections was demonstrated. However, rate coding is not optimal for energy efficiency and speed. In the quest to make every spike count, many other event encoding methods have been proposed (Kittler and Illingworth, 1986; Thorpe, 1990; Rieke et al., 1999; Dupeyroux, 2021; Auge et al., 2021). Note that in this work we restrict ourselves to a fully digitally implementable solution and do not consider analog or mixed-signal implementations. (Schuman et al., 2022; Indiveri et al., 2011).

(Keuninckx et al., 2018) ends with the question of how MMV networks can be trained in an event-driven manner, making proper use of the natural tendency of MMV networks to spread their spiking activity over time. We provide an answer to that question and show that arithmetic synaptic addition within the neural network itself can be avoided.

1.1. Related work

Many neuromorphic hardware approaches try to alleviate the burden of synaptic addition, which scales quadratically with the number of neurons. Reducing the precision of synaptic

*Corresponding author
Email address: lars.keuninckx@imec.be (Lars Keuninckx)

weights to even as low as 4 bits, can result in almost no performance penalty, if the training algorithm is made aware of this (Stuyt et al., 2021; Moons et al., 2016). Another method, which allows for a much higher neuron count, works by dividing the neural population in small fully connected tiles or cores and implementing a network-on-chip messaging between the cores (Davies et al., 2018; Akopyan et al., 2015; Yousefzadeh et al., 2022). By time-multiplexing a single physical neuron, the chip-area can be drastically reduced at the cost of execution time (Frenkel et al., 2019). Delay-line based reservoir computing (DLRC) has this idea at its core, as a complete recurrent network is replaced by a single nonlinear node placed within a delayed feedback loop (Soriano et al., 2015; Appeltant et al., 2011). Optimal results will likely require careful balancing of several techniques. Still, all these approaches have in common that some form of synaptic addition needs to happen. Recently, many new successful approaches to training and applying SNNs have been introduced. Shao et al. (2023) introduce a novel learning algorithm, Joint Excitatory Inhibitory Cycle Iteration Learning, significantly improving bio-mimicry and adaptability of spiking neuron models. Li et al. (2024a) propose a pruning technique for SNNs that works both on the neuron and channel level, as to take full advantage of sparsity. Jiang et al. (2024) introduce and excitation-inhibition mechanism-assisted hybrid learning (EIHL) algorithm for SNNs, that balances global and local learning through network connectivity. Li et al. (2024b) propose a structured way of pruning deep SNNs based on kernel activity levels, called Spiking Channel Activity-based (SCA), which is inspired by synaptic plasticity mechanisms. Xu et al. (2024b) introduce a method to integrate knowledge distillation and connection pruning for SNNs, to dynamically optimize the sparsity. Xu et al. (2024a) propose an efficient recurrent SNN (RSNN) to reduce the loss of time domain information across slice samples and thus retain spatio-temporal information throughout the network. Xu et al. (2020) introduce ConvDenseSNN; a hybrid CNN/SNN approach, where the continuous-valued CNN functions as feature extractor, feeding into a unsupervised trained SNN classifier, thus adding biological plausibility.

Currently, neuromorphic computing as a field is expanding in a wide range of very different directions. One direction of research is on translations of traditional convolutional neural networks into rate based SNNs by using the correspondence between the I&F input-output rate and the ReLU activation function. Another direction, which is highly application dependent, is exploring different encodings for the input and output data. Model compression by enhancing sparsity and weight quantization and representation are important for both storage and speed. Network Architectural Search and data movement optimization enhance mapping onto existing hardware. Upcoming memristor based approaches lend heavily from material science and force training algorithms to take into account the precision with which weights can be represented. All these directions are driven by the wish to save energy, the brain consuming only 20 to 40 Watts is often given as the ultimate motivation. The alternative approach of MMV networks in this article does not start from bio-inspiration, however two basic principles from biology are still retained: parallel computation and time-based processing.

Other recent approaches, similar to ours in that they eschew biological inspiration altogether, are those of Deep Differentiable Logic Gate Networks (LGNs) (Petersen et al., 2022, 2024) and Lookup Table Networks (LUTs) (Wang et al., 2019, 2020). Also here, the design is fully digital and no clearly discernible synaptic addition is present in the networks. An important difference is that for these networks are made from fixed random connections, resulting in large networks, whereas in our approach, the connections between the units themselves are trained.

1.2. Outline

We propose a training algorithm for recurrent networks of MMVs or timer neurons in which the units are OR-ed together, thus avoiding the typical synaptic input adder reduction tree. The final output is still calculated from either a linear readout or, alternatively, a population-coding style summing readout, which take as their input the vector of accumulated spikes over time of the individual MMVs.

This article is organized as follows. Next, we reiterate the basic operation of an abstract non-retriggerable MMV and illustrate this with a spike interval detector consisting of three units. Then we show how MMV networks can be modeled and trained via back-propagation and surrogate gradient techniques with an autogradient framework such as PyTorch (Paszke et al., 2019). By employing a slow weight binarization technique spread out over many epochs, we end up with fully binarized and event-driven digital networks void of synaptic addition. We prove the effectiveness of this approach on various tasks. We then provide estimates for the energy per inference for the MNIST handwritten digits task. Finally, we summarize our conclusions and outline future work.

2. Monostable Multivibrators As Computational Units

2.1. Basic MMV Operation

Figure 1 shows the operation and symbol of an abstract non-retriggerable MMV. The MMV can be in one of two states, idle or triggered and has only two digital inputs: excitatory (EXC) and inhibitory (INH). When in the idle state, an excitatory input event ① triggers the MMV. After time interval T it returns to the idle state, while firing ②. A firing is also indicated by the downward facing arrow in the state transition from triggered to idle. If after an input trigger ③ a new excitatory event happens ④ within an interval T, then this last event is ignored and the output will fire ⑤ in response to the earlier triggering. This is the non-retriggerable part of the operation. Lastly, if within period T after triggering ⑥ an inhibitory event ⑦ occurs, then the MMV returns to idle without firing ⑧. Note that the INH and EXC inputs act upon the MMV in a fundamentally different way. Due to the OR-ed interconnect we describe in Section 2.3, there will be no difference in having one or more events on an input line simultaneously, as far as the operation of the MMV is concerned.

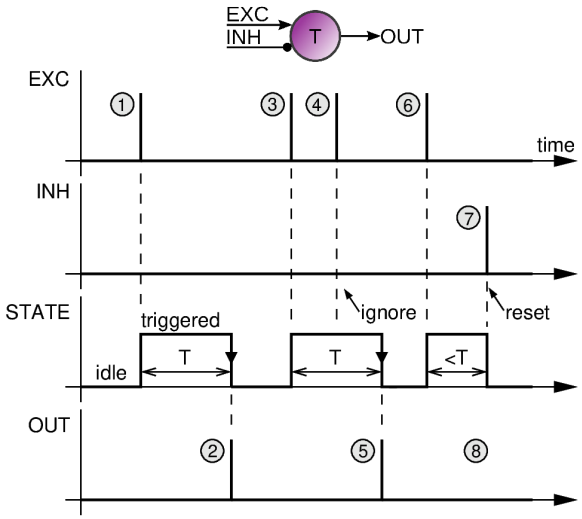


Figure 1: Symbol and operation of an abstract non-retriggerable monostable multivibrator: the unit has an excitatory (EXC, triangle arrow) and an inhibitory (INH, dotted arrow) input and one parameter, its period T. Starting from the idle state, and EXC event ① triggers the MMV. After period T, it fires ② and returns to the idle state. State transitions from triggered to idle that lead to a spike are indicated with a downward pointing arrow. Should after a trigger ③ an excitatory event happen within period T ④, then this last event is ignored and only the first one leads to an output spike ⑤. If within a period after a trigger ⑥ an INH event occurs ⑦, then the unit returns to the idle state without firing ⑧.

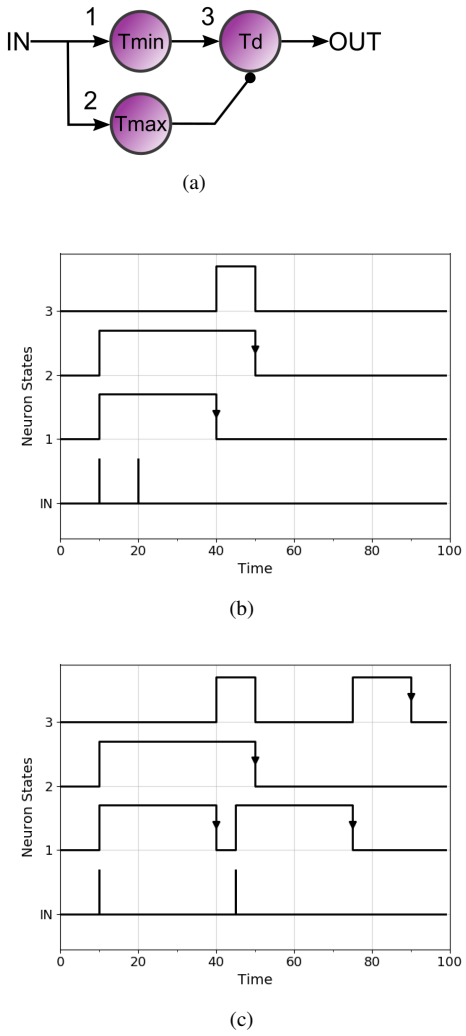


Figure 2: (a) An MMV spike interval detector: with the periods chosen as $T_{\max} > T_{\min} > T_{\max}/2$ and $T_{\mathrm{d}} > T_{\max} - T_{\min}$, this circuit produces a spike at the output only if, starting from the idle state, the interval ΔT_{in} between two input spikes is in $[T_{\min}, T_{\max}[$. Here we chose $T_{\max} = 40$, $T_{\min} = 30$ and $T_{\mathrm{d}} = 15$. (b) The network produces no spike ($\Delta T_{\mathrm{in}} = 10$) or (c) a single spike at its output ($\Delta T_{\mathrm{in}} = 35$). MMV state transitions from a triggered state (high) to the idle state (low) which are accompanied by an output spike or firing (thus a state change not caused by an inhibitory input) are indicated by an downward pointing arrow.

It is clear that such a device is easy to implement in a digital form using a binary counter at its core. When considering discrete-time implementations, for completeness we must define what happens when EXC, INH and firing events coincide. We arbitrarily choose this as follows: if an INH event happens at the same time step as an EXC event, then the INH event takes precedence and the MMV either returns to or stays in the idle state. In the MMV model (see further) this is handled by the order in which the events are handled. An EXC event at the same time as a firing leads to a retrigger. A firing event takes precedence on an INH. Our experiments suggest these rules are not critical, however for completeness such boundary situations must be defined.

2.2. MMV Circuit Example

We illustrate how MMVs perform computation in-time using the three-unit circuit shown in Figure 2a. Given the right periods, this circuit operates as a spike interval detector, outputting a spike only when two input spikes enter the network within the interval $[T_{\min}, T_{\max}[$. Here we have chosen $T_{\max} = 40$, $T_{\min} = 30$ and $T_{\mathrm{d}} = 15$ time steps. The diagrams of Figure 2b and Figure 2c show the states of the MMVs in the circuit. A firing or spike produced at a triggered (high) to idle (low) state transition is indicated with a downward pointing arrow. Transitions from triggered to idle without firing, the result of an INH event on an MMV input, are shown without an arrow. Starting from all units in the idle state, an input event at $t_1 = 10$ triggers units 1 and 2, in both Figure 2b and Figure 2c. At $t_1 + T_{\min} = 40$, unit 1 fires and triggers unit 3. In Figure 2b, a second input event at $t_2 = 20$ enters the circuit but is ignored because both units 1 and 2 are still triggered. At the end of the period of unit 2, $t_1 + T_{\max} = 50$, it fires, thereby inhibiting unit 3 and the circuit produces no output spike.

Now consider the situation in Figure 2c, where the second input event enters at $t_2 = 45$, in between the firings of units 1 and 2. In other words, the input spike interval $t_2 - t_1 = 35$ is between $T_{\min} = 30$ and $T_{\max} = 40$. Therefore the second input event in Figure 2c retriggers only unit 1 and not unit 2. Unit 2 then also goes on to inhibit the firing of unit 3 at $t = 50$. However, since unit 1 is now still triggered, some time later, at $t_2 + T_{\min} = 45 + 30 = 75$, it fires and retriggers unit 3. This leads to the network producing an output event at $t_2 + T_{\min} + T_{\mathrm{d}} = 45+30+15 = 90$. In general, with all units starting from the idle state and the periods chosen such that $T_{\max} > T_{\min} > T_{\max}/2$ and $T_{\mathrm{d}} > T_{\max} - T_{\min}$, unit 3 only fires if two input spikes on

units 1 and 2 arrive within the interval $[T_{\min}, T_{\max}[$.

In summary, MMV networks work by shifting and selectively blocking or passing events, depending upon the time and location of their occurrence and the state of the circuit at that time.

2.3. Large reconfigurable MMV networks

Figure 3 shows what a large reconfigurable MMV network looks like schematically. The MMVs are placed adjacent to the interconnect network. Each MMV is outfitted with a spike counter to allow for post-processing of the collected spikes. There is no synaptic addition as in conventional spiking (L)IF based neuromorphic networks. Inputs to the MMVs are OR-ed together. To an MMV there either is or isn't an event on its EXC/INH input at a certain time step or clock event. For a network of N MMVs, the $2N^2 - 2N$ configuration bits (excluding self-connections) are stored in a chain of D-flipflops that runs in a zigzag fashion through the network. A configuration bit gates the output of one unit $i = 1 \ldots N$ to an EXC or INH line of another unit $j = 1 \ldots N$ by an AND operation. External connections that carry the input events $1 \ldots K$ to the MMVs, not shown to avoid overloading the figure, are handled similarly. Symbolically, the total input to MMV j is described by:

$$\begin{aligned} \mathrm{EXC}_j &= \mathrm{OR}_{i=1}^{N} \left(C_{\mathrm{net,i,j}}^{\mathrm{EXC}} \text{ AND } \mathrm{OUT}_i \right) \\ &\quad \mathrm{OR}_{k=1}^{K} \left(C_{\mathrm{in,k,j}}^{\mathrm{EXC}} \text{ AND } \mathrm{IN}_k \right) \end{aligned} \tag{1}$$

$$\begin{aligned} \mathrm{INH}_j &= \mathrm{OR}_{i=1}^{N} \left(C_{net,i,j}^{\mathrm{INH}} \text{ AND } \mathrm{OUT}_i \right) \\ &\quad \mathrm{OR}_{k=1}^{K} \left(C_{\mathrm{in,k,j}}^{\mathrm{EXC}} \text{ AND } \mathrm{IN}_k \right), \end{aligned} \tag{2}$$

where $C_{\mathrm{net}}^{\mathrm{EXC}}$ and $C_{\mathrm{net}}^{\mathrm{INH}}$ are binary matrices representing the recurrent excitatory and inhibitory connections and $C_{\mathrm{in}}^{\mathrm{EXC}}$ and $C_{\mathrm{in}}^{\mathrm{INH}}$ are binary matrices representing the connections from the external input to the network.

Thus the problem at hand is the following: how can we train the connections in an MMV network and their time periods, such that the spiking activity in the network solves a given computational task, e. g. a classification? We show our approach in the next section.

3. Methods

Our training algorithm is based on back-propagation combined with surrogate gradients (Neftci et al., 2019), gradual connection binarization and period rounding techniques. In this way, an autogradient framework such as PyTorch (Paszke et al., 2019) can be applied. Our goal is to train an MMV network such as shown in Figure 3. Three problems need to be addressed:

A) modeling the discontinuous operation of the MMV,

B) modeling the OR-ing network,

C) finding integer values for the MMV periods and binary values for the network connections.

Solutions to these three problems are described next and are common to all practical demonstrations in this article.

3.1. Modeling the MMV

Spiking neurons are commonly modeled with a step-like activation function (a Heaviside step function) which is discontinuous and thus non-differentiable (Gerstner and Kistler, 2002). This poses a problem, since during the backward pass of the back-propagation algorithm the derivative of the activation function is needed to calculate the sensitivity of the loss function with regards to a certain weight value.

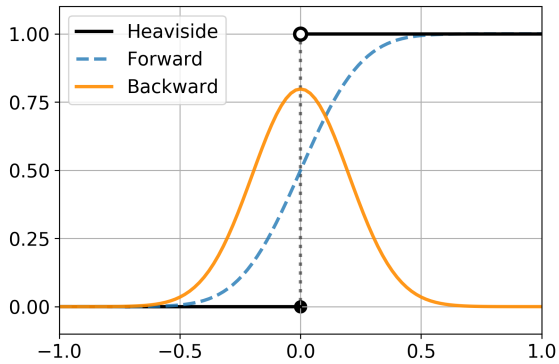


Figure 4: The surrogate gradient technique: the step-like forward-pass activation function of a spiking neuron is discontinuous and therefore non-differentiable at the origin. Implicitly this activation function can be approximated by a smooth differentiable function in the forward pass. During the backward pass of the back-propagation algorithm, the derivative of the activation function is explicitly replaced by the smooth gradient that is the result from the differentiation of the approximated activation function.

A recently introduced solution to this problem, such that spiking networks can be back-propagated using autogradient frameworks, is called the surrogate gradient technique (Neftci et al., 2019; Yin et al., 2021). The approach is outlined in Figure 4. The Heaviside function (black line) can be approximated by a smoothed function having the same limits. Here we used the cumulative distribution function of a Gaussian or normal distribution with mean zero (blue dashed line). This approximation is implicit, as during the forward pass the original discontinuous activation function is used. Then, during the backward pass the derivative of the smoothed activation function, here a Gaussian, is explicitly surrogated (orange line). The spread *sigma* and amplitude A of the surrogate gradient are hyperparameters to be tuned in the code. Algorithm 1 describes the surrogate gradient technique. Here, g is being propagated in the backward pass.

Algorithm 1 Activation Function

FORWARD_ACT(x):
  save x **as context**
  return 1 **if** $x > 0$, **else** 0

BACKWARD_ACT(g):
  hyperparameters: amplitude:A, spread: σ
  $x \leftarrow$ **retrieve context**
  return $g \cdot A \exp\left(-x^2/\sigma^2\right)$

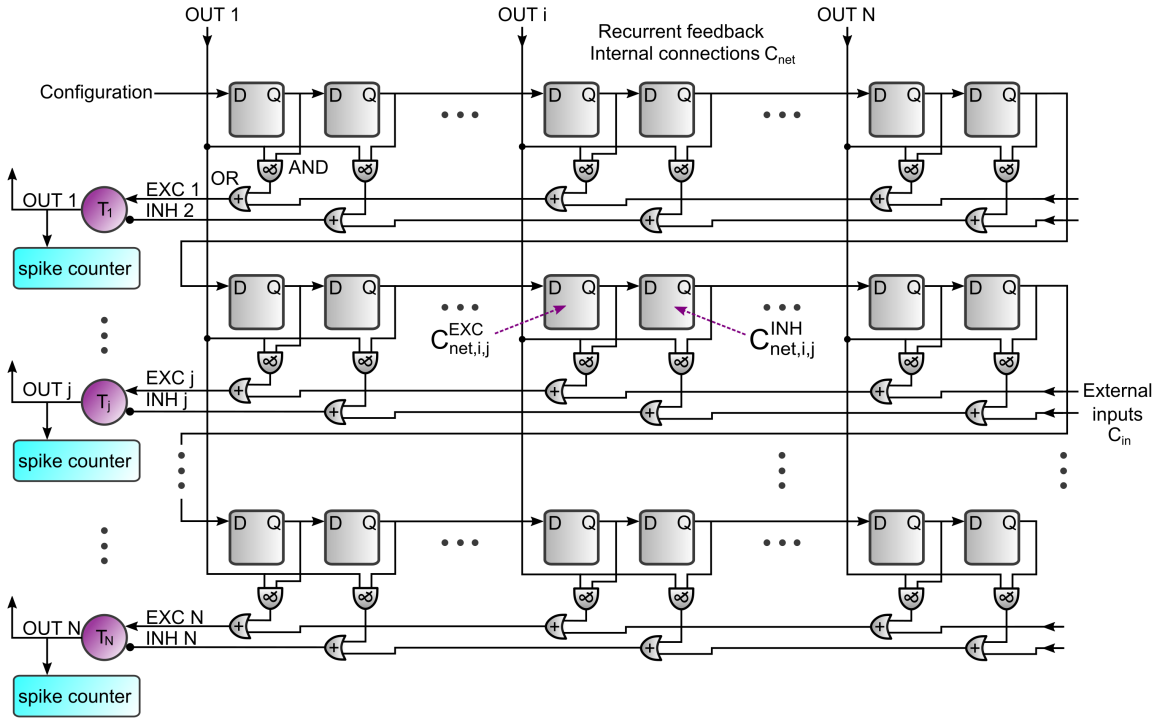


Figure 3: A reconfigurable MMV network: unlike conventional neuromorphic (L)IF networks, there is no synaptic addition as inputs to the MMVs individual EXC/INH lines are OR-ed together. Spike counters on the outputs of the MMVs allow for post-processing of the total number of spikes generated. The configuration is stored in a chain of D-flipflops that zigzags through the network. Note self-connections $C^{\text{EXC}}_{\text{net},i,i}$ and $C^{\text{INH}}_{\text{net},i,i}$ need not be implemented.

Algorithm 2 shows how a (basic) LIF neuron can be modeled using the activation function of Algorithm 1. Here, the input activation a is added to the LIF's state. This input activation is the sum of the incoming spikes weighted with their respective synaptic strengths: $a = \sum_i w_i s_i$. The state is then compared against a threshold L via the spiking function of Algorithm 1. If the state exceeds the threshold, then a spike is returned and the state is reset to zero. This threshold is often fixed to unity, however per-neuron adaptive thresholds can also be used (Yin et al., 2021).

Algorithm 2 Leaky Integrate & Fire Neuron

UPDATE_LIF_STATE(a):
  hyperparameters: leakiness: $\alpha \leq 1$, threshold: L
  state $\leftarrow \alpha \cdot$ state $+ a$
  spike = FORWARD_ACT(state $- L$)
  if spike = 1: state $\leftarrow 0$
  return spike

The key observation that is needed for modeling the MMV in an autogradient framework, Algorithm 3, is that the two possible state changes, idle-to-triggered as driven by EXC and triggered-to-idle driven by INH, are also discontinuous events that require the use of a surrogate gradient. It is natural to model the MMV as a timer that, once triggered, integrates a state variable upwards until it crosses a threshold. Instead of an additive integrator update as in state$(t+1)$ = state$(t) + \delta$, we used a

Algorithm 3 Monostable Multivibrator Neuron

UPDATE_MMV_STATE(in_exc, in_inh):
  parameter: period: T
  hyperparameters: growth factor: $\alpha = 1.1$,
  input threshold: $\beta = 0.99$
  $\delta = \alpha^{-T+1/2}$
  state $\leftarrow \alpha \cdot$ state
  spike = FORWARD_ACT(state $- 1$)
  if spike = 1: state $\leftarrow 0$
  exc = FORWARD_ACT(in_exc $- \beta$)
  inh = FORWARD_ACT(in_inh $- \beta$)
  state $\leftarrow$ exc $\cdot \delta$ if state = 0, else state
  if inh = 1: state $\leftarrow 0$
  return spike

multiplicative timing process:

$$\text{state}(t+1) = \alpha\, \text{state}(t), \tag{3}$$

with growth factor $\alpha > 1$, because it gave better results for reasons that are explained below. We chose $\alpha = 1.1$ and the firing threshold equal to unity in all our experiments. An idle MMV has a state = 0. Triggering then means setting a small positive initial value δ to the state: state$(0) = \delta$. This value is related to the MMV period T as:

$$\delta = \alpha^{-T+1/2}, \tag{4}$$

such that T subsequent time steps result in the state crossing the

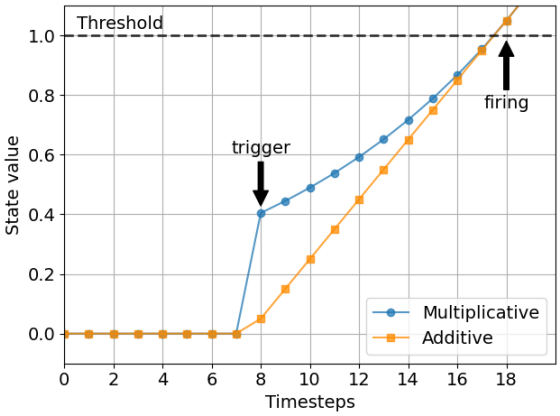


Figure 5: Multiplicative (blue) vs. additive (orange) timing process: An MMV with period $T = 10$ is triggered at $t = 8$ and fires at $t = 18$. As explained in the text, the multiplicative state integration retains its sensitivity to the incoming trigger event, regardless of the MMV period.

threshold:

$$\text{state}(T) = \alpha^{T}\alpha^{-T}\alpha^{1/2} > 1, \tag{5}$$

and $\text{state}(T-1) = \alpha^{-1/2} < 1$. This is illustrated in Figure 5, where an MMV with period $T = 10$ is triggered at $t = 8$ and fires at $t = 18$. It turns out that using a multiplicative instead of an additive integrating process avoids a bias towards lower MMV periods that could negatively impact the result of the training. To see this, consider how the state depends on an incoming triggering signal exc that is either 0 or 1:

$$\text{state}(t+1) = \begin{cases} \text{exc} \cdot \delta & \text{if state}(t) = 0, \\ \alpha\,\text{state}(t) & \text{otherwise.} \end{cases} \tag{6}$$

At $t = T$, the state has crossed the firing threshold and:

$$\frac{\partial \text{state}(T)}{\partial \text{exc}} = \frac{\partial \alpha^{T}\,\text{exc}\,\alpha^{-T}\alpha^{1/2}}{\partial \text{exc}} = \alpha^{1/2} > 1. \tag{7}$$

Compare this to an additive integrator where, for the state to have crossed the threshold T time steps after a triggering, we have to set $0 < \delta \leq T^{-1}$:

$$\text{state}(t+1) = \begin{cases} \text{exc} \cdot \delta & \text{if state}(t) = 0, \\ \text{state}(t) + T^{-1} & \text{otherwise.} \end{cases} \tag{8}$$

Here we have:

$$\frac{\partial \text{state}(T)}{\partial \text{exc}} \leq \frac{1}{T}. \tag{9}$$

In other words, by modeling the MMV using a multiplicative integrator, we avoid that the back-propagated gradients are downscaled by a factor $1/T$, thus enabling better long-term correlations. Note that this is only for assisting in the training and is unrelated to any hardware implementation, where typically the MMVs are implemented using binary counters.

Returning to Algorithm 3, the MMV model maintains a state and receives excitatory and inhibitory input events. Note that the actual value of the incoming excitatory or inhibitory events is of no concern once they are above unity, because of the activation function, thus this is where the OR-ing operation happens. The threshold for the input events in_exc and in_inh has been set slightly below unity at 0.99, such that a single unity-valued event can trigger or inhibit the MMV. The order in which the spiking, excitatory and inhibitory events are handled, is such that the model agrees with the operation of the MMV as discussed in Section 2. As explained in Section 4.1, MMV spikes are accumulated and further processed to form inferences. A practical Verilog implementation of a non-retriggerable MMV is given in Appendix A.4.

3.2. Modeling the OR-ing network

In this section we explain how we model the internal and external MMV network connections, such that they can be optimized via back-propagation. The internal network connections from the output of MMV $i = 1 \ldots N$ to the input of MMV $j = 1 \ldots N$ are recorded in an $N \times N$ matrix C_{net}. The entries of this matrix are $C_{\text{net},i,j} = -1$ for an INH connection or $C_{\text{net},i,j} = +1$ for an EXC connection, leaving 0 for no connection. In order to find these connections, we initially treat the values in C_{net} as continuous real-valued weights and apply a slow binarization process over many training epochs.

In a first step we initialize C_{net} with positive values drawn from a uniform distribution between zero and one, such that on average c entries per MMV are above 0.5. We experimented with $c = 2...8$. Since we found that the training converges faster if started without recurrent excitatory (hence positive) connections, in a second step the values of C_{net} above the diagonal are negated:

$$C_{\text{net}} \leftarrow \text{LT}(C_{\text{net}}) - \text{UT}(C_{\text{net}}), \tag{10}$$

where LT() and UT() denote matrices having the upper and lower triangular elements of C_{net} with the complementary entries set to zero. The input connections, represented by an $N \times K$ matrix C_{in}, are initially drawn from a uniform distribution such that c_e positive (EXC) and c_i negative (INH) connections per neuron have absolute values above 0.5. We found good results can be obtained from a wide range of initial values.

The network is then forward-propagated as shown in Algorithm 4. At every time step, a K-wide vector of input spikes and an N-wide vector of internally generated spikes stemming from the previous time step is processed. Note these event vectors are binary valued (0 or 1). The network matrices are first split into positive and negative values, representing the connections that lean more towards excitatory or inhibitory operation, before being multiplied with their respective events vector to calculate the MMV's input activations:

$$C^{\text{EXC}}_{\text{net},i,j} = \begin{cases} C_{\text{net},i,j} & \text{if } C_{\text{net},i,j} \geq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

and:

$$C^{\text{INH}}_{\text{net},i,j} = \begin{cases} -C_{\text{net},i,j} & \text{if } C_{\text{net},i,j} < 0, \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

and likewise for the input connections, represented by C_{in}.

This operation is crucial for understanding how the OR-ing operation in the network arises: by performing the split as in Equations (11) and (12), only positive values appear in the separate C^{INH} and C^{EXC} matrix multiplications that constitute the event propagation. This splitting operation allows back-propagation, as it merely consists of placing a minus sign in front of the negatively valued weights. Once the network is fully digitized, as explained further below, the results from these matrix multiplications represent the integer number of active EXC and INH signals that are OR-ed at the MMV inputs, at a certain time step. For the MMV modeled as in Algorithm 3, there is no difference between having a single or multiple event on an input line occurring during the same time step. Thus we obtain an OR operation in which the gradients of the dynamical behavior over time are traceable.

Algorithm 4 MMV network propagation

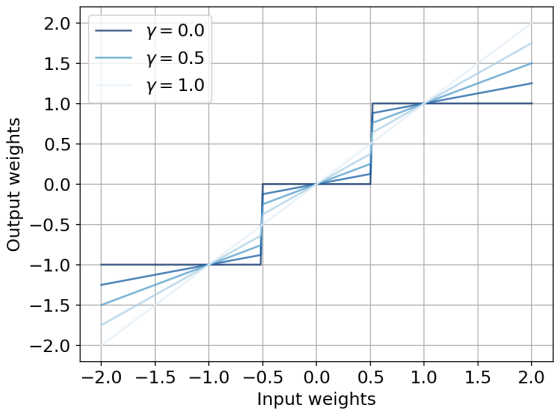
NET_FORWARD(input_spikes, internal_spikes)
parameter: connections C_{in}, C_{net}
split:
$C_{\text{in}}^{\text{EXC}}, C_{\text{in}}^{\text{INH}} \xleftarrow{\text{Eqs. (11,12)}} C_{\text{in}}$
$C_{\text{net}}^{\text{EXC}}, C_{\text{net}}^{\text{INH}} \xleftarrow{\text{Eqs. (11,12)}} C_{\text{net}}$
calculate activations:
act_exc $= C_{\text{in}}^{\text{EXC}} \cdot$ input_spikes $+ C_{\text{net}}^{\text{EXC}} \cdot$ internal_spikes
act_inh $= C_{\text{in}}^{\text{INH}} \cdot$ input_spikes $+ C_{\text{net}}^{\text{INH}} \cdot$ internal_spikes
calculate new internal spikes:
internal_spikes =
UPDATE_MMV_STATE(act_exc, act_inh)
return internal_spikes

At this point the connections given by the network matrices and the MMV periods still have real (non-integer) values.

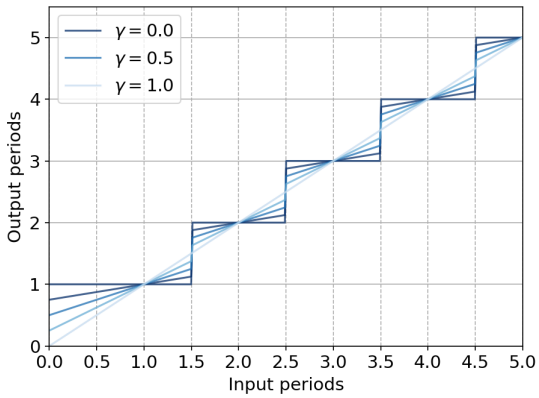
3.3. Binarizing the Connections and Rounding the Periods

We found that abruptly binarizing the weights did not lead to good results. Therefore, we employ a "soft" binarization method which squeezes the continuous weights to binary values, over the course of many epochs, as follows: after initializing the network with as explained above, we train for a certain number of epochs, meaning we apply the forward propagation as per Algorithm 4, while the backward step and connection value updates are handled in the traditional manner by the auto-gradient framework. Then, from a certain epoch e_1 until epoch e_2, in the forward step the network matrices are, replaced by a gradually more rounded analogue:

$$C_{i,j} \leftarrow \begin{cases} \gamma C_{i,j} & \text{if } -\frac{1}{2} \le C_{i,j} \le \frac{1}{2}, \\ \gamma C_{i,j} + 1 - \gamma & \text{if } C_{i,j} > \frac{1}{2}, \\ \gamma C_{i,j} - 1 + \gamma & \text{if } C_{i,j} < -\frac{1}{2}, \end{cases} \tag{13}$$



(a)



(b)

Figure 6: (a) Binarization scheme of Equation (13) for connections. (b) Rounding scheme for MMV periods as per Equation (15). During training, γ is slowly decreased from one to zero over many epochs, resulting in a fully digitized network.

with C either C_{in} or C_{net}, and γ a factor that decreases linearly between epochs $e = e_1 \dots e_2$ as:

$$\gamma(e) = \begin{cases} 1 & \text{if } e \le e_1, \\ \dfrac{e_2 - e}{e_2 - e_1} & \text{if } e_1 < e < e_2, \\ 0 & \text{if } e \ge e_2. \end{cases} \tag{14}$$

Both e_1 and e_2 are training hyperparameters. Figure 6a shows how Equation (13) changes the weights, as $\gamma = 1 \dots 0$. Note that the final values in the connectivity matrices are ternary, either $+1$, -1 or 0, representing an EXC, INH or no connection, respectively. We still call this a binarization scheme, since what we are doing is binarizing the INH and EXC connections as shown in Figure 3. Thus the network is gracefully forced to employ fully binarized connections. A similar method is used to force the MMV periods $T_{i=1\dots N}$ to integer values:

$$T_i \leftarrow \gamma T_i + (1 - \gamma) \max\{1, \lfloor T_i \rceil\}. \tag{15}$$

This is shown in Figure 6b. In future work, the linearly decreasing binarization speed, Section 14, might be changed to a

strategy where the step size follows a more complicated rule, with the goal of saving on training time. Furthermore, we use the Adam optimizer (Kingma and Ba, 2014) and the negative log likelihood loss function for classification tasks. This concludes the methods we use to model and train non-retriggerable MMV networks.

4. Results

4.1. MNIST Handwritten Digits

The MNIST handwritten digits classification task (Lecun et al., 1998) is the "hello world" application for novel neural network approaches. Figure 7 shows how an MMV network for this task is organized. A 28×28 grayscale image with pixel values normalized between 0 (black) and 1 (white) is first thresholded at 0.5 to obtain events. The input events are applied to the network of size N in a row-by-row fashion, leading to an N-entry wide integer-valued vector of spike counts after 28 time steps. These spike counts are then post-processed by a standard N-to-10 linear layer W_{out}. The highest output value of the linear layer then indicates the class $0 \ldots 9$ of the image. Note column-by-column streaming gives similar results. Superficially, the architecture resembles a liquid state machine (LSM) (Maass et al., 2002), however the difference here is that next to the readout layer's weights and biases, the internal network connections are also optimized. The linear output layer carries a cost of $N \times C$ multiply-accumulate (MAC) operations and C additions with N the number of neurons and C the number of output classes. At the time of writing, we could not get good results by constructing the output layer from MMV neurons alone, although a population-coding style output, explained further, does work well.

We initialized a network of $N = 500$ MMVs as follows: on average each MMV received 3 excitatory and 3 inhibitory connections with a value above 0.5 from the 28 input lines. Each MMV was given on average $c = 5$ connections to other MMVs, as explained in Section 3.2. The MMV periods were initialized randomly between $T = 2$ and $T = 8$.

The network is trained for 200 epochs on 50000 training images and tested on 10000 images using a batch size of 500. Between each exemplar the MMVs are reset to the idle state and the spike counters are cleared. The connection binarization and period rounding schemes run from epochs 80 to 160. Figure 8a shows how the training and testing accuracies evolve over time. The best test accuracy is 98.61%, found at epoch 195. Figure 8b shows the corresponding confusion matrix.

Furthermore, from epoch 170 on the periods are kept constant and the learning rate is gradually stepped down from $5 \cdot 10^{-3}$ to approximately $5 \cdot 10^{-4}$. We also apply data-augmentation in the form of randomly rotating (±15°), shifting (±2 pixels) and scaling (±5%) the training images, explaining why the test-performance line is *above* the train performance in Figure 8a. We found that these steps improve the final result marginally by about 0.5%. We also experimented with collecting extra spikes for some time steps after all rows of an image were shifted in, but could not find any improvement.

Table 1: MNIST handwritten digits

Network/Algorithm	Accuracy [%]
2×500 MMVs hor./ver. + linear readout	99.02
1000 MMVs + linear readout	98.80
500 MMVs + linear readout	98.61
500 MMVs + population coding readout	98.22
200 MMVs + linear readout	97.35
200 MMVs + population coding readout	94.88
Logistic regression (thresholded images)	91.28
Logistic regression (grayscale images)	92.00

Figures. 9a shows the resulting network connections C_{net} (-1 for INH, $+1$ for EXC). Approximately 28.4% of connections are active (either EXC or INH). Note that there are excitatory recurrent connections present after training. A typical spike pattern is shown in Figure 9b. A single digit of the test set generates mean ± std. dev. = 285 ± 27 spikes.

In Table 1, we list results for MMV networks of different sizes (best of completely binarized epochs, single run). Included are networks in which we replaced the linear readout layer with a more hardware-friendly population coding output, as follows: first the number of neurons N is chosen such that it is divisible by the number of classes C. Then the neurons, numbered $0 \ldots N-1$, are divided into C equally sized groups $iN/C \ldots (i+1)N/C - 1$, with $i = 0 \ldots C-1$. The spike counts are accumulated for each group of neurons. The group with the most spikes determines the class. We increased the softmax temperature to 10 for the population coding output network with $N = 500$ MMVs. The population coding output requires only N addition operations but, as is clear from Table 1, comes at a performance cost for smaller networks.

We also trained a network of 2×500 MMVs, in which one part was fed the images row-wise and the other column-wise. Their spike vectors were then concatenated and processed together in one 1000-to-10 linear readout. The resulting test accuracy surpasses 99% and outperformed a single 1000-MMV network. Included in Table 1 are the results for simple logistic regressions on, both the original grayscale and thresholded MNIST digits.

In Section 4.5, we discuss energy estimates for the MNIST task and compare to other published results.

4.2. Yin-Yang Task

The goal of the Yin-Yang task (Kriener et al., 2021) is to classify given points (x, y) inside the circular symbol as belonging to one of four possible regions. Figure 10 shows the result for this task using an $N = 100$ MMV network. The black circles indicate erroneous classifications during testing. Most errors are found near borders between two regions. The difficulty here is to encode the real input values into events. As shown in Figure 11, we binary encode an input tuple $(x, y) \in [0, 1]^2$ as two $b = 12$-bit wide integer numbers $(\lfloor x(2^b - 1) \rfloor, \lfloor y(2^b - 1) \rfloor)$ and concatenate these representations. The resulting 24-bits wide spike vector is presented at the input of an MMV network for

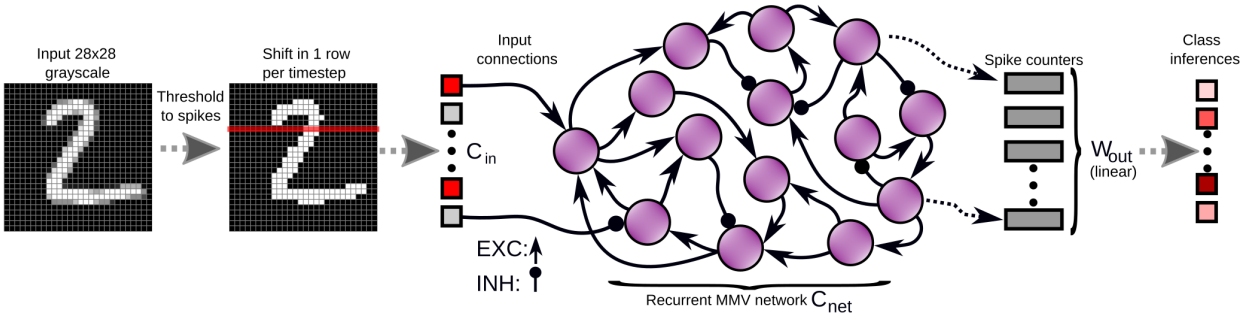
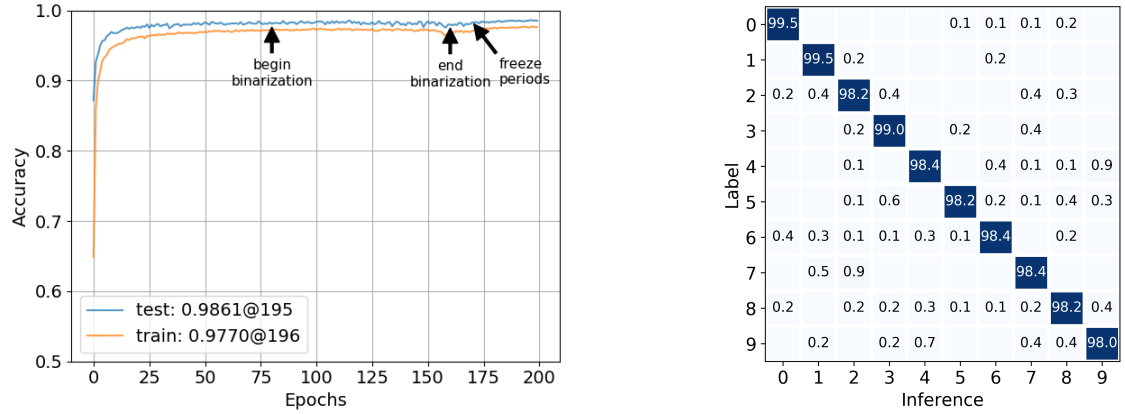
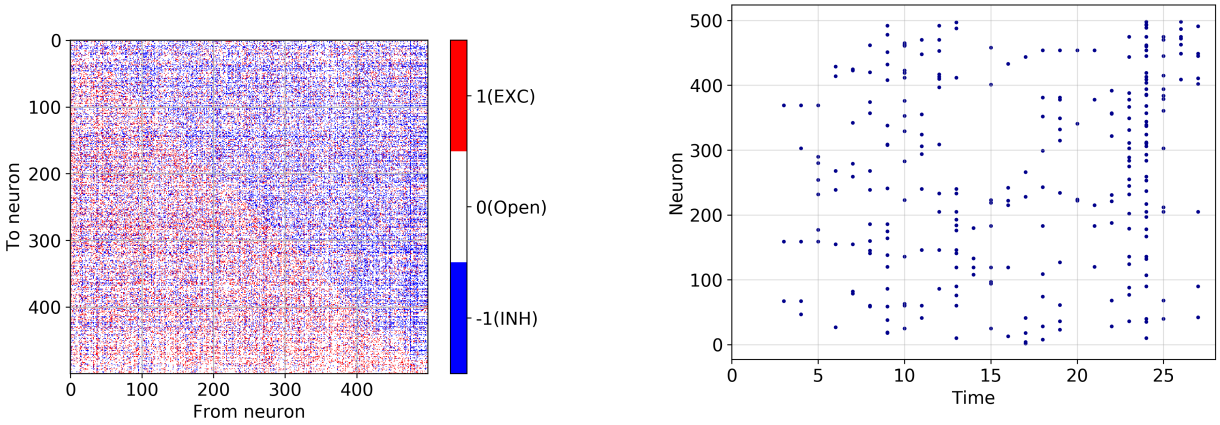


Figure 7: MNIST handwritten digits classification with a recurrent MMV network: the network starts with all MMVs in the idle state. An exemplar is shifted in one row per time step. After 28 time steps, the vector of spike counts is post-processed by either a linear layer or a population count coding.



(a) (b)

Figure 8: (a) Accuracy over epochs for the MNIST handwritten digits task on a network of 500 MMVs. (b) Confusion matrix. Values lower than 0.1% are not shown.



(a) (b)

Figure 9: (a) Recurrent network connections of a 500 MMV network, as shown in Figure 7, trained for the MNIST handwritten digits recognition. (b) A typical spike pattern as it forms when shifting in a digit.

10 consecutive time steps. Thus on top of the actual segmentation, the MMV network has to learn how to "read" the binary input representation. To construct the output, we use a population coding scheme as discussed above. The accuracies from networks of various sizes are given in Table 2. We used $2 \cdot 10^5$ randomly generated points for training and 10^4 for testing, and trained for 100 epochs. The binarization scheme ran from epoch 40 until epoch 80. The network of 500 MMVs generated mean±std. dev. = 82±24 spikes per inference and 26.2% of the connections were active.

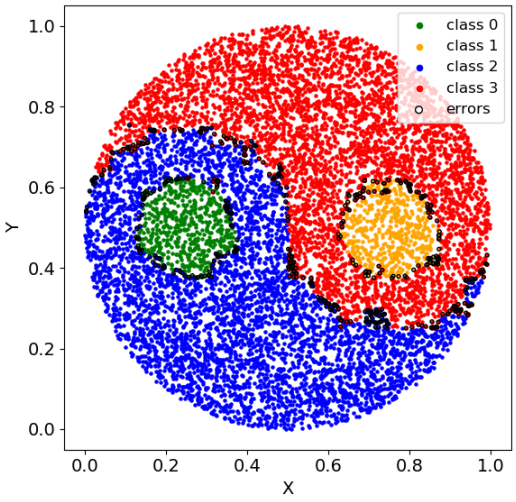


Figure 10: The Yin-Yang segmentation task: the goal is to classify points (x, y) inside the symbol as belonging to one of the four regions. The image shows the result for testing 10k points on a network of $N = 100$ MMVs with a population coding output, after training for 100 epochs on 200k points. Inference errors are shown as black circles.

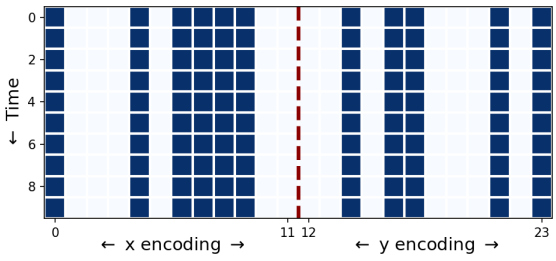


Figure 11: A point $(x, y) \in [0, 1]^2$ in the Yin-Yang symbol is binary encoded as two $b = 12$-bit wide integer numbers $(\lfloor x(2^b - 1) \rfloor, \lfloor y(2^b - 1) \rfloor)$. These are then concatenated and applied to the MMV network for 10 consecutive time steps. The point presented here is at $x = 0.54603 \sim 100010111100_2$ and $y = 0.17314 \sim 001011000101_2$.

Table 2: Yin-Yang segmentation task

Network Size N	Accuracy [%]
20	66.37
40	90.51
100	96.26
200	98.43
500	99.16

4.3. Google Soli Radar Gestures

A dataset of hand gestures as measured with the Google Soli Radar sensor is introduced in (Wang et al., 2016). The dataset contains sequences of 32×32-pixel sized range-Doppler maps, representing 11 different gestures, measured from ten different users over the four receivers of the Radar chip. The interval between the maps is 25 milliseconds. The number of maps per gesture varies from 28 to 145. We take a hint from (Tsang et al., 2021), where the Soli dataset is processed with a liquid state machine (LSM) and largely follow the same preprocessing steps: first the range-Doppler maps are limited to their first 12 rows as most of the activity is found in that region. These are flattened into rows of $32 \times 12 = 384$ pixels and all maps belonging to a single gesture are concatenated. Pixel values in the Soli dataset frames range from zero to one. Any value above zero is seen as an event or input spike (thus set to one). Then, this frame is rescaled in the time (vertical) dimension to a fixed length of 50 rows. Figure 12 shows some examples of the preprocessed gestures. These 384-wide rows of events are

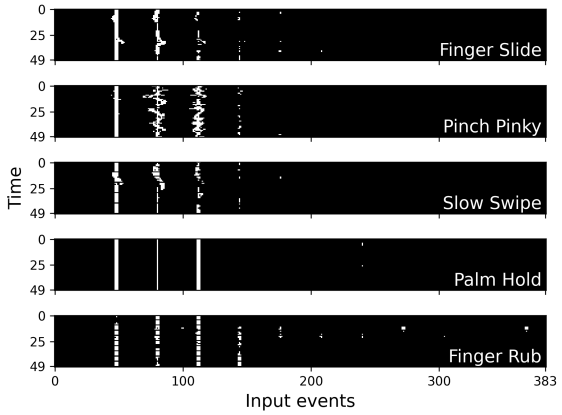


Figure 12: Some Soli dataset gesture examples after preprocessing to spikes. The examples are applied to the MMV network in a row-by-row fashion.

shifted in one-by-one, over the course of 50 time steps, into an MMV network with a similar architecture as shown in Figure 7. Again, spike counts are collected and fed into a linear readout layer that is optimized together with the connections of the MMV network itself. The dataset thus consists 22000 gestures, 1900 per class, except for class 11 ("palm hold") which contains only 1100 exemplars. In Table 3, we compare some results found in the literature to those we obtained from variously sized MMV networks. Our results are 10-fold cross-validated. This shows that even for a small network of only $N = 100$ MMVs good results are obtained. Figure 13 shows the confu-

Table 3: Google Soli Radar gestures

Reference	Network	Accuracy [%]
This work	100 MMVs + lin. readout	97.62 ± 0.33
This work	250 MMVs + lin. readout	98.01 ± 0.26
This work	500 MMVs + lin. readout	98.32 ± 0.29
Yin et al. (2021)	Spiking RNN	91.9
Tsang et al. (2021)	LSM, 460 neurons	98.6 ± 0.7
Wang et al. (2016)	CNN+RNN	87.17

sion matrix for the case of the $N = 500$ MMV sized network, with an overall accuracy of mean ± std. dev. = (98.32 ± 0.29)%. Values below 0.1% are omitted for clarity.

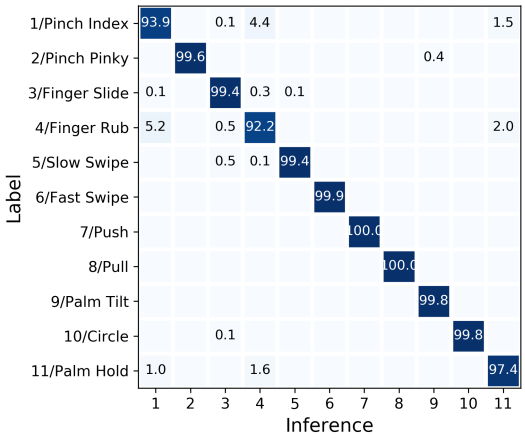


Figure 13: Confusion matrix resulting from a 10-fold cross validation of the Soli Radar gestures dataset on $N = 500$ sized MMV networks. Values below 0.1% are not shown.

For this task all MMV networks were trained for 200 epochs with initialization and training details similar to those given in Section 4.1. A typical learning curve is shown in Figure 14.

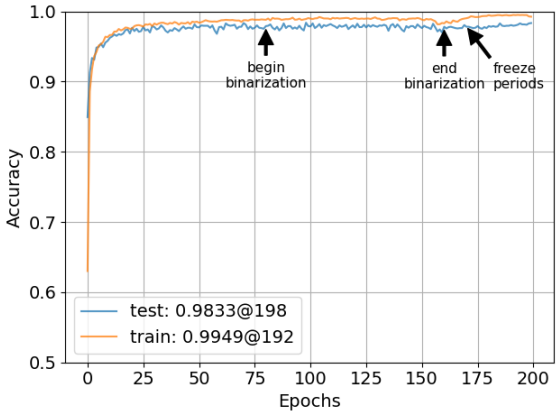


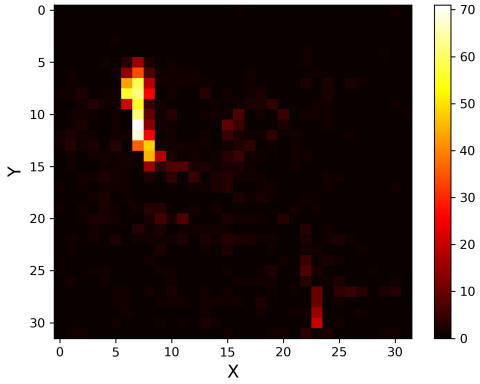
Figure 14: Typical learning curve resulting from training a $N = 500$ sized MMV network on the Soli Radar gestures dataset.

4.4. IBM DVS128 Gestures

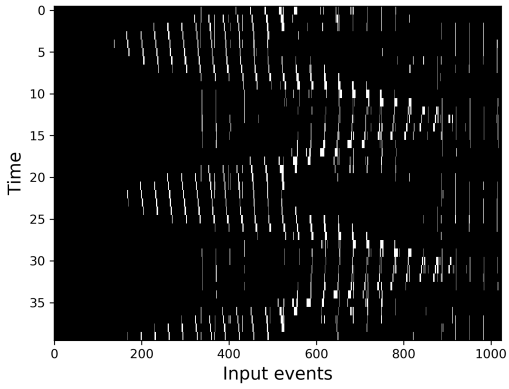
The IBM DVS128 gestures dataset (M., 2017; Amir et al., 2017) contains 11 gestures by 29 participants recorded under three different lighting conditions with an 128×128 pixels iniLabs DVS128 dynamic vision sensor camera. Instead of entire frames, this camera transmits events corresponding to intensity changes of individual pixels. The events also denote the direction or polarity of the changes in intensity.

In the dataset, 10 gestures are predefined movements, while the 11th is an improvisation freely chosen by the participants. In some works this 11th gesture is omitted as it lacks commonality among the participants. We choose to keep this 11th gesture.

To preprocess the dataset, the recorded events from a single gesture, which are tuples (x, y, t, p) indicating pixel position, time-of-arrival and polarity, are first accumulated into frames of 50ms duration. Each new frame starts with all entries at zero. A positively polarized event increments the position in the frame it addresses, while a negative one decrements it. The minimum value in a frame is clamped to zero. These frames are downscaled by averaging to a size of 32×32 pixels. Figure 15a shows a single frame of a "right hand clockwise" gesture. We take a block of 40 frames, corresponding to 2.0s, as a single gesture exemplar to be processed by the MMV network.



(a)



(b)

Figure 15: (a) A single frame of DVS camera events collected over 50ms and resized to 32×32 pixels. The activity shown is a person waving their right hand. (b) A complete preprocessed gesture consists of 40 frames that are flattened into rows and thresholded.

A block is reshaped by flattening the frames, thus we end up with a rectangular array of 40 rows of 1024-entries wide. These are then thresholded into events: all entries above the empirically derived value of 6.0 are interpreted as an excitatory input spike to the MMV network. Figure 15b shows an entire preprocessed"right hand clockwise" gesture.

Also, before flattening and thresholding, at each epoch, data augmentation is applied to the training set to prevent overfitting: all frames of a single gesture together undergo a random rotation (±15°), horizontal shift (±4 pixels) and vertical shift

(±3 pixels).

The exemplars are fed into the network in a row-by-row fashion, quite similar to how the Soli gesture dataset was processed in Section 4.3. In order to obtain more exemplars from this fairly small dataset, we generate exemplars by hopping 250ms within a single gesture recording file. We split the final dataset into 22347 train and 5870 test exemplars.

We trained a recurrent network of $N = 500$ MMVs, similar to what is shown in Figure 7, for 200 epochs on this dataset. As shown in Figure 16, we reach a testing accuracy of 92.91%. The confusion matrix for this test is shown in Figure 17. Un-

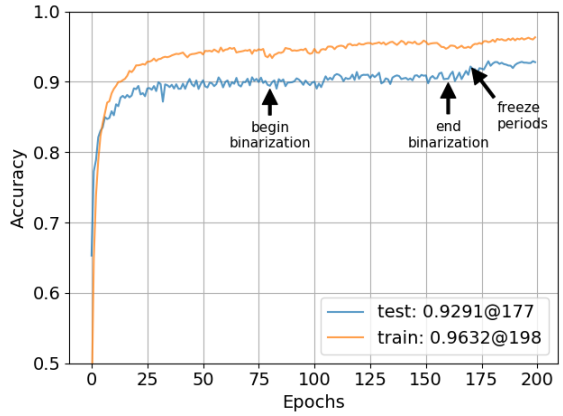


Figure 16: Accuracy over epochs for an $N = 500$ sized MMV network on the IBM DVS128 gestures dataset.

surprisingly, large gestures such as the hand waves are easier distinguished from one another than small ones for which the arms are closer to the body. In Table 4 we compare our results

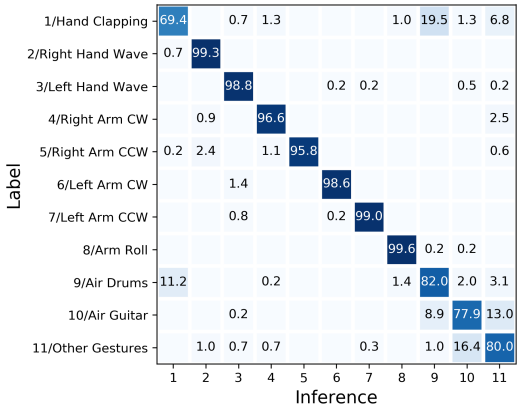


Figure 17: Confusion matrix result for the IBM DVS128 gestures dataset on an $N = 500$ sized MMV network. Values below 0.1% are not shown. The overall accuracy was 92.91%.

for this dataset to some recent results found in the literature. Note that in (Shen et al., 2024) 98.58% accuracy is reached, by using nonlinear adaptive dendritic techniques.

Table 4: IBM DVS128 gestures

Reference	Network/Algorithm	Accuracy [%]
This work	500 MMVs	92.91
Safa et al. (2021b)	LIF+STDP	92.5
Amir et al. (2017)	CNN	91.77
Amir et al. (2017)	CNN with postprocessing	94.59
Samadzadeh et al. (2020)	Spiking ResNet	96.7
Liu et al. (2024)	Spike Attention Coding ResNet 18	95.14
Zheng et al. (2021)	ResNet 17	96.87
Shen et al. (2024)	5 layer SNN with adaptive dentrites	98.58

4.5. *Energy Estimate*

In this section we offer a preliminary estimate for the energy per inference for the MNIST handwritten digit classification and compare it to published results using comparable chip technologies. We caution the reader that a fair apples-to-apples comparison to other neuromorphic approaches is difficult to achieve, as a wide variety of architectures, sensor interfacing, pre and post processing, network sizes, and accuracies are reported in a poorly stratified manner. It must be noted that the works referenced below might target other aspects of neuromorphic engineering, for example online learning or implementing a large number of (virtual) neurons. Such features understandably do not come for free and this further complicates comparison. Also, at this time we do not have actual hardware available, so we are comparing actual chip implementations to a result from a mapping tool in what follows, although this is believed to be accurate. Not included in this estimate, are the time t_{readout} and energy E_{readout} needed for the linear readout layer, having as input the vector of spike counts per neuron. This requires $N \times C$ MAC and C addition operations, with $N = 500$ the number of MMVs in the recurrent network and $C = 10$ the number of output classes. For a population coding style readout, only N additions are needed. In a first approximation, we envision the readout layer to be handled by a central system controller. Note that the population coding readout can be implemented directly in hardware by using C (N/C) binary encoders and C accumulators. In this particular case, every population of 50 neurons would connect to one binary encoder of $b = \lceil \log_2(50) \rceil = 6$ bits. The output of each encoder accumulates over 28 timesteps into a sum that maximally reaches $50 \times 28 = 1400$ spikes per accumulator and thus can be represented in an 11 bit register. Note that the population coding readout can be improved upon, while remaining hardware-friendly, for example by adding a preloaded bias term per group or by selectively masking out MMVs. We leave benchmarking for the energy E_{readout} and time t_{readout} required for the readout of the network as future work, as these are system specific and here we only aim at getting an idea of the performance of the MMV and OR-ing network. Also not included in this estimate are energy costs associated

Table 5: MNIST handwritten digits energy consumption estimate and comparison

Reference	Method/Conditions	Acc. [%]	Time/inf.	Energy/inf.
This work	500 MMVs, 28 nm HPC+, linear readout population coding readout	 98.61 98.22	330 ns + t_{readout}	855 pJ + E_{readout}
Chen et al. (2018)	4096-neuron, 1M synapse SNN, 10 nm	97.9	-	1.7 µJ
Park et al. (2019)	200+200+10 SNN, 28 nm	97.83	10µs	236.5 nJ
Frenkel et al. (2020)	SPOON, 28 nm event-driven CNN	97.5	117 µs	313 nJ
Stuyt et al. (2021)	"μBrain", 256+64+16 SNN, 40 nm	91.7	4.2 ms	308 nJ
Wang et al. (2025)	MorphBungee, 256+256+256+200 SNN, 65 nm	96.06	440µs	26.8 µJ
Renner et al. (2024)	Backpropagation on Loihi, 784 in+400+10	97.5	170µs	2.5 µJ
Huijbregts et al. (2024)	128 × 128 Compute-in-memory, FC 768+256+256+256+10, 3 nm FinFET	97.64	22.7 ns	607 pJ

with data movements to and from the MMV network.

A trained Pytorch 500-MMV network was converted to a Verilog HDL description representing a reconfigurable OR-ed network, as in Figure 3. Here, a large shift register of $2 \times 500 \times 500$ flipflops is used to gate the output of the MMVs with an AND gate followed by an OR tree. The MMV period counting registers were chosen to be 5 bits wide. One 6-bit wide spike counter per neuron is also included in the simulation. We verified that the spike counts from the PyTorch and Verilog models are identical. After synthesis it was mapped onto TSMC's 28 nm HPC Plus process, clocked at 100 MHz. The power was estimated using Cadence's Joules RTL to Power Flow tool, with the supply voltage and temperature set to 0.9 V and 25 °C, respectively. Cadence Joules uses the RTL level description of the design, technology characterization libraries, and RTL level simulation stimuli to synthesize the design, propagate the recorded activity to the netlist nodes and then estimate power average power consumption. The result is shown in Table 5.

The design area was reported to be 0.8 mm^2. Even when taking into account above caveats and the missing readout layer or population counters, these results suggests that the MMV OR-ed network approach is interesting, energy-wise. Only the result by Huijbregts *et al.* (Huijbregts et al., 2024) surpasses ours in energy efficiency, by using a multiport SRAM cell as well as surrounding circuitry designed in recently introduced 3 nm IMEC FinFET technology.

4.6. Number of trainable parameters

Consider a reconfigurable recurrent MMV network with N units and K inputs where each MMV has a maximum timer period of $T_{\max}$. Per input or recurrent connection, there are three possibilities (no connection, inhibitory or excitatory), thus requiring $\lceil N(K+N)\log_2(3)\rceil$ bits. Add to this the number of bits needed to store the timer periods $N\lceil\log_2(T_{\max})\rceil$, to find the total number of bits to store the network configuration. Additionally, a readout layer for C classes where each coefficient is stored in b bits, requires $(N+1)Cb$ bits.

As an example, take $N = 500$, $K = 28$ as for the MNIST handwritten digits processing example in Section 4.1, Figure 7 and $T_{\max} = 31$ so the timer periods fit in 5-bit registers, then the network configuration requires 51.4 Kbytes of storage. The readout layer, with the coefficients stored in FP16, would require 9.8 Kbytes, for a total of 61.2 Kbytes.

5. Conclusions

We presented an algorithm for training recurrent networks of monostable multivibrators, non biologically-inspired spiking timer neurons. Input signals to the timer neurons are logically OR-ed together, such that the typical synaptic addition inside the network is avoided. We showed on several datasets that by combining slow binarization and surrogated gradients these digital recurrent MMV networks can be trained to perform classification tasks with good accuracy. A preliminary estimate of the energy per inference showed that this approach is promising. We share the results of additional experiments in the supplementary material.

For now, our work is limited to networks having only non-retriggerable MMVs. Many other monostable units, with their own triggering, timing and firing rules can be thought up and are just as straightforward to implement in digital form. These novel computational units could be mixed in heterogenous networks, even with (leaky) integrate & fire neurons. Heterogeneous networks may offer the best of all solutions in terms of chip area and/or energy efficiency for a certain application. After all, the brain also holds many different types of neurons and glial support cells (Purves et al., 2017). It is however not immediately clear how to select the best network architecture, given certain requirements on accuracy, chip area and power. Here, the toolset of neural architecture search (Elsken et al., 2019; Wistuba et al., 2019) could be applied. Evolutionary optimization could also offer a differentiable model-free solution (Schu-

man et al., 2020; Schliebs and Kasabov, 2013; Storn and Price, 1997). Furthermore evolutionary learning can be tuned to keep the sparsity at a certain level, as shown by citeshen2023. How to optimally present the data to an MMV network and how to extract the output from the network dynamics is still an open question. This is a general problem common to all types of spiking or event-driven neural networks. As of yet there does not seem to be a consensus (Schuman et al., 2022). There is even evidence that preprocessing methods that work well for SNNs do not work equally well for CNNs and vice versa (Safa et al., 2021a). This makes sense, given the central role that time plays in SNNs. Some alternatives to simple thresholding are spike latency coding, rank order coding, time-to-first-spike (TTFS) coding (Masquelier and Thorpe, 2010; Thorpe, 1990; Auge et al., 2021), thermometer encoding (Buckman et al., 2018; Komkov et al., 2021) and Hough's spiking algorithm (Hough et al., 1999; Dupeyroux, 2021). Note that we only used the output spikes or events, however the MMV states, idle or triggered, could also hold useful information. Given that the dynamics of our MMV networks are quite different from those in regular recurrent spiking networks, an investigation of the computational complexity and storage capacity is also needed. On the implementation side one could for example turn to analog methods to implement the MMV timers. Many complex analog neuromorphic building blocks have already been constructed Indiveri et al. (2011). An analog approach to MMV timer neurons could lead to further architectural freedoms, for example programmable input thresholds, which themselves could lead to leaner network architectures and further power savings. Purely digital, one could implement graded output spikes and input thresholds, by means of subdivision of the timesteps into "microsteps". The MMV output spike grade would be equal to the amount of microsteps in would be active. Algorithmically it is straightforward to augment the described training algorithm in this direction. Recently also the application of nonvolatile memory (NVM) in neuromorphic engineering has started to gain attention Varshika et al. (2022).

We see all these as areas for future exploration. Likely, there is much about monostable multivibrator networks and their dynamics still to be discovered. In this article we took a first step by showing that event-driven networks of simple timers can effectively be trained without synaptic addition within the network itself.

Conflict of interests

All authors declare that they have no conflicts of interest.

Supplementary Material

Our code is available from the corresponding author upon request.

Acknowledgments

This work has received funding from the Flemish Government under "Onderzoeksprogramma Artificiële Intelligentie Vlaanderen".

References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.J., Taba, B., Beakes, M., Brezzo, B., Kuang, J.B., Manohar, R., Risk, W.P., Jackson, B., Modha, D.S., 2015. TrueNorth: Design and tool flow of a 65mW 1 million neuron programmable neurosynaptic chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34, 1537–1557. doi:10.1109/TCAD.2015.2474396.

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., Modha, D., 2017. A low power, fully event-based gesture recognition system, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7388–7397. doi:10.1109/CVPR.2017.781.

Appeltant, L., Soriano, M.C., Van der Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C.R., Fischer, I., 2011. Information processing using a single dynamical node as complex system. Nat. Commun. 2, 468.

Auge, D., Hille, J., Mueller, E., Knoll, A., 2021. A survey of encoding techniques for signal processing in spiking neural networks. Neural Processing Letters 53, 4693–4710. doi:10.1007/s11063-021-10562-2.

von Bartheld, C.S., Bahney, J., Herculano-Houzel, S., 2016. The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. Journal of Comparative Neurology 524, 3865–3895. doi:https://doi.org/10.1002/cne.24040.

Buckman, J., Roy, A., Raffel, C., Goodfellow, I., 2018. Thermometer encoding: One hot way to resist adversarial examples, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=S18Su--CW.

Chen, G.K., Kumar, R., Sumbul, H.E., Knag, P.C., Krishnamurthy, R.K., 2018. A 4096-neuron 1m-synapse 3.8pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10nm finfet cmos. 2018 IEEE Symposium on VLSI Circuits , 255–256.

Cramer, B., Stradmann, Y., Schemmel, J., F., Zenke, 2022. Heidelberg spoken digits dataset. online URL: https://compneuro.net/datasets/hd_audio.tar.gz. [Online; accessed 30-May-2022].

Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.H., Wild, A., Yang, Y., Wang, H., 2018. Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro 38, 82–99. doi:10.1109/MM.2018.112130359.

Dupeyroux, J., 2021. A toolbox for neuromorphic sensing in robotics. CoRR abs/2103.02751. arXiv:2103.02751.

Elsken, T., Metzen, J.H., Hutter, F., 2019. Neural architecture search: A survey. Journal of Machine Learning Research 20, 1–21.

Frenkel, C., Lefebvre, M., Legat, J.D., Bol, D., 2019. A 0.086mm^2 12.7pJ/-SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm CMOS. IEEE Transactions on Biomedical Circuits and Systems 13, 145–158. doi:10.1109/TBCAS.2018.2880425.

Frenkel, C., Legat, J.D., Bol, D., 2020. A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas, in: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. doi:10.1109/ISCAS45731.2020.9180440.

Gerstner, W., Kistler, W.M., 2002. Spiking Neuron Models. 1st ed., Cambridge University Press.

Herculano-Houzel, S., 2009. The human brain in numbers: a linearly scaled-up primate brain. Frontiers in Human Neuroscience 3. URL: https://www.frontiersin.org/article/10.3389/neuro.09.031.2009, doi:10.3389/neuro.09.031.2009.

Hough, M., Korkin, M., Gers, F., Nawa, N., 1999. Spiker: Analog waveform to digital spiketrain conversion in ATR's artificial brain (cam-brain) project. International Conference on Robotics and Artificial Life .

Huijbregts, L., Liu, H.H., Detterer, P., Hamdioui, S., Yousefzadeh, A., Bishnoi, R., 2024. Energy-efficient SNN architecture using 3nm FinFET multiport SRAM-based CIM with online learning, in: Proceedings of the 61st ACM/IEEE Design Automation Conference, Association for Computing Machinery, New York, NY, USA. doi:10.1145/3649329.3656514.

Indiveri, G., Linares-Barranco, B., Hamilton, T., van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.C., Dudek, P., Häfliger, P., Renaud, S., Schemmel, J., Cauwenberghs, G., Arthur, J., Hynna, K., Folowosele, F.,

Saïghi, S., Serrano-Gotarredona, T., Wijekoon, J., Wang, Y., Boahen, K., 2011. Neuromorphic silicon neuron circuits. Frontiers in neuroscience 5, 73. doi:10.3389/fnins.2011.00073.

Jiang, T., Xu, Q., Ran, X., Jiangrong, S., Lv, P., Qiang, Z., Pan, G., 2024. Adaptive deep spiking neural network with global-local learning via balanced excitatory and inhibitory mechanism, in: The Twelfth International Conference on Learning Representations ICLR.

Keuninckx, L., Danckaert, J., Van der Sande, G., 2018. Monostable multivibrators as novel artificial neurons. Neural Networks 108, 224–239. doi:https://doi.org/10.1016/j.neunet.2018.08.014.

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. International Conference on Learning Representations .

Kittler, J., Illingworth, J., 1986. Minimum error thresholding. Pattern Recognition 19, 41–47. doi:https://doi.org/10.1016/0031-3203(86)90030-0.

Komkov, H., Pocher, L., Restelli, A., Hunt, B., Lanthrop, D., 2021. Reservoir computing using networks of cmos logic gates, in: International Conference on Neuromorphic Systems 2021, Association for Computing Machinery, New York, NY, USA. doi:10.1145/3477145.3477163.

Kriener, L., Göltz, J., Petrovici, M.A., 2021. The Yin-Yang dataset. doi:10.48550/ARXIV.2102.08211.

Lecun, Y., Cortes, C., Burges, C., 1998. MNIST database of handwritten digits. URL: http://yann.lecun.com/exdb/mnist.

Li, Y., Fang, X., Gao, Y., Zhou, D., Shen, J., Liu, J.K., Pan, G., Xu, Q., 2024a. Efficient structure slimming for spiking neural networks. IEEE Transactions on Artificial Intelligence 5, 3823–3831. doi:10.1109/TAI.2024.3352533.

Li, Y., Xu, Q., Shen, J., Xu, H., Chen, L., Pan, G., 2024b. Towards efficient deep spiking neural networks construction with spiking activity based pruning, in: Forty-first International Conference on Machine Learning ICML.

Liu, J., Hu, Y., Li, G., Pei, J., Deng, L., 2024. Spike attention coding for spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems 35, 18892–18898. doi:10.1109/TNNLS.2023.3310263.

Lyon, R.F., 1982. A computational model of filtering, detection, and compression in the cochlea, in: ICASSP, pp. 1282–1285.

M., F., 2017. IBM DVS128 gestures dataset. https://research.ibm.com/interactive/dvsgesture/. [Online; accessed 25-May-2022].

Maass, W., Natschläger, T., Markram, H., 2002. Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Computation 14, 2531–2560.

Masquelier, T., Thorpe, S., 2010. Learning to recognize objects using waves of spikes and spike timing-dependent plasticity. The 2010 International Joint Conference on Neural Networks (IJCNN) doi:10.1109/IJCNN.2010.5596934.

Mead, C., 1990. Neuromorphic electronic systems. Proceedings of the IEEE 78, 1629–1636. doi:10.1109/5.58356.

Monroe, D., 2014. Neuromorphic computing gets ready for the (really) big time. Commun. ACM 57, 13–15. URL: https://doi.org/10.1145/2601069, doi:10.1145/2601069.

Moons, B., De Brabandere, B., Van Gool, L., Verhelst, M., 2016. Energy-efficient ConvNets through approximate computing, in: 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 1–8. doi:10.1109/WACV.2016.7477614.

Neftci, E.O., Mostafa, H., Zenke, F., 2019. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. IEEE Signal Processing Magazine 36, 51–63. doi:10.1109/MSP.2019.2931595.

Park, J., Lee, J., Jeon, D., 2019. A 65nm 236.5nJ/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback, in: 2019 IEEE International Solid- State Circuits Conference - (ISSCC), pp. 140–142. doi:10.1109/ISSCC.2019.8662398.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035.

Petersen, F., Borgelt, C., Kuehne, H., Deussen, O., 2022. Deep differentiable logic gate networks, in: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 2006–2018.

Petersen, F., Kuehne, H., Borgelt, C., Welzel, J., Ermon, S., 2024. Convolutional differentiable logic gate networks, in: Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 121185–121203.

Purves, D., Augustine, G.J., Fitzpatrick, D., Hall, W.C., LaMantia, A.S., Mooney, R.D., Platt, M.L., White, L.E., 2017. Neuroscience. Oxford University Press, USA (October 12, 2017).

Renner, A., Sheldon, F., Zlotnik, A., 2024. The backpropagation algorithm implemented on spiking neuromorphic hardware. Nat Commun 15, 9691.

Rieke, F., Warland, D., de Ruyter van Steveninck, R., Bialek, W., 1999. Spikes: Exploring the Neural Code. MIT Press, Cambridge, MA, USA.

Safa, A., Corradi, F., Keuninckx, L., Ocket, I., Bourdoux, A., Catthoor, F., Gielen, G.G.E., 2021a. Improving the accuracy of spiking neural networks for radar gesture recognition through preprocessing. IEEE Transactions on Neural Networks and Learning Systems , 1–13doi:10.1109/TNNLS.2021.3109958.

Safa, A., Ocket, I., Bourdoux, A., Sahli, H., Catthoor, F., Gielen, G., 2021b. A new look at spike-timing-dependent plasticity networks for spatio-temporal feature learning. doi:10.48550/ARXIV.2111.00791.

Samadzadeh, A., Far, F.S.T., Javadi, A., Nickabadi, A., Chehreghani, M.H., 2020. Convolutional spiking neural networks for spatio-temporal feature extraction. doi:10.48550/ARXIV.2003.12346.

Schliebs, S., Kasabov, N., 2013. Evolving spiking neural network—a survey. Evolving Systems 4, 87–98. doi:10.1007/s12530-013-9074-9.

Schuman, C.D., Kulkarni, S.R., Parsa, M., Mitchell, J.P., Date, P., Kay, B., 2022. Opportunities for neuromorphic computing algorithms and applications. Nature Computational Science 2, 10–19. doi:10.1038/s43588-021-00184-y.

Schuman, C.D., Mitchell, J.P., Patton, R.M., Potok, T.E., Plank, J.S., 2020. Evolutionary optimization for neuromorphic systems, in: Proceedings of the Neuro-Inspired Computational Elements Workshop, Association for Computing Machinery, New York, NY, USA. doi:10.1145/3381755.3381758.

Shao, Z., Fang, X., Li, Y., Feng, C., Shen, J., Xu, Q., 2023. Eicil: Joint excitatory inhibitory cycle iteration learning for deep spiking neural networks, in: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 32117–32128.

Shen, G., Zhao, D., Zeng, Y., 2024. Exploiting nonlinear dendritic adaptive computation in training deep Spiking Neural Networks. Neural Networks 170, 190–201. doi:10.1016/j.neunet.2023.10.056, arXiv:2023-11-10.

Slaney, M., 1988. Lyon's cochlear model. https://engineering.purdue.edu/~malcolm/apple/tr13/LyonsCochlea.pdf. [Techn. Rep. 13].

Soriano, M., Ortín, S., Keuninckx, L., Appeltant, L., Danckaert, J., Pesquera, L., der Sande, G.V., 2015. Delay Based Reservoir Computing: Noise effects in a Combined Analog and Digital Implementation. IEEE TNNLS 26.

Storn, R., Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359. doi:10.1023/A:1008202821328.

Stuyt, J., Sifalakis, M., Yousefzadeh, A., Corradi, F., 2021. μbrain: An event-driven and fully synthesizable architecture for spiking neural networks. Frontiers in Neuroscience 15. doi:10.3389/fnins.2021.664208.

Thorpe, S., 1990. Spike arrival times: A highly efficient coding scheme for neural networks. Parallel Processing in Neural Systems and Computers .

Tkanov, D., 2022. Lyon's auditory model for python. online URL: https://github.com/sciforce/lyon. [Online; accessed 30-May-2022].

Tsang, I.J., Corradi, F., Sifalakis, M., Van Leekwijck, W., Latré, S., 2021. Radar-based hand gesture recognition using spiking neural networks. Electronics 10. doi:10.3390/electronics10121405.

Varshika, M.L., Corradi, F., Das, A., 2022. Nonvolatile memories in spiking neural network architectures: Current and emerging trends. Electronics 11. doi:10.3390/electronics11101610.

Wang, E., Davis, J.J., Cheung, P.Y.K., Constantinides, G.A., 2019. LutNet: Rethinking inference in FPGA soft logic, in: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 26–34. doi:10.1109/FCCM.2019.00014.

Wang, E., Davis, J.J., Cheung, P.Y.K., Constantinides, G.A., 2020. LutNet: Learning FPGA configurations for highly efficient neural network inference. IEEE Transactions on Computers 69, 1795–1808. doi:10.1109/TC.2020.2978817.

Wang, S., Song, J., Lien, J., Poupyrev, I., Hilliges, O., 2016. Interacting with soli: Exploring fine-grained dynamic gesture recognition in the radio-frequency spectrum, in: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, Association for Computing Machinery, New York, NY, USA. pp. 851–860. doi:10.1145/2984511.2984565.

Wang, T., Tian, M., Wang, H., Zhong, Z., He, J., Tang, F., Zhou, X., Lin, Y., Yu, S.M., Liu, L., Shi, C., 2025. MorphBungee: A 65nm 7.2mm^2 27microJ image digital edge neuromorphic chip with on-chip 802-Frames per second multi-layer spiking neural network learning. IEEE Transactions on Biomedical Circuits and Systems 19, 209–225. doi:10.1109/TBCAS.2024.3412908.

Wistuba, M., Rawat, A., Pedapati, T., 2019. A survey on neural architecture search. doi:10.48550/ARXIV.1905.01392.

Xu, Q., Fang, X., Li, Y., Shen, J., Ma, D., Xu, Y., Pan, G., 2024a. RSNN: Recurrent spiking neural networks for dynamic spatial-temporal information processing, in: Proceedings of the 32nd ACM International Conference on Multimedia, Association for Computing Machinery, New York, NY, USA. pp. 10602–10610. doi:10.1145/3664647.3680573.

Xu, Q., Li, Y., Fang, X., Shen, J., Zhang, Q., Pan, G., 2024b. Reversing structural pattern learning with biologically inspired knowledge distillation for spiking neural networks, in: Proceedings of the 32nd ACM International Conference on Multimedia, Association for Computing Machinery, New York, NY, USA. pp. 3431–3439. doi:10.1145/3664647.3680655.

Xu, Q., Peng, J., Shen, J., Tang, H., Pan, G., 2020. Deep CovDenseSNN: A hierarchical event-driven dynamic framework with spiking neurons in noisy environment. Neural Networks 121, 512–519. URL: https://www.sciencedirect.com/science/article/pii/S0893608019302618, doi:https://doi.org/10.1016/j.neunet.2019.08.034.

Yin, B., Corradi, F., Bohté, S.M., 2021. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. bioRxiv doi:10.1101/2021.03.22.436372.

Yousefzadeh, A., van Schaik, G.J., Tahghighi, M., Detterer, P., Traferro, S., Hijdra, M., Stuijt, J., Corradi, F., Sifalakis, M., Konijnenburg, M., 2022. Seneca: Scalable energy-efficient neuromorphic computer architecture, in: 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 371–374. doi:10.1109/AICAS54282.2022.9870025.

Zheng, H., Wu, Y., Deng, L., Hu, Y., Li, G., 2021. Going deeper with directly-trained larger spiking neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press. pp. 11062–11070. doi:10.1609/aaai.v35i12.17320.

Appendix A. Supplementary material

Here, we briefly present some additional experimental results.

Appendix A.1. Heidelberg Spoken Digits Keyword Spotting

The Heidelberg digits dataset (Cramer et al., 2022) contains aligned spoken digits in English and German. We restricted our test of MMV networks on this dataset to the English "zero" to "nine" utterances. Figure A.1a shows an example waveform of the digit "one" from the dataset. For preprocessing, we start by downsampling the audio recordings from 48ksps to 12ksps, which does not lead to loss of intelligibility. We then apply the Lyon passive cochlear model (Lyon, 1982; Slaney, 1988; Tkanov, 2022) with the decimation factor set to 64. This results in a spectrogram-like representation in which each column represents approximately 5.33 ms of sound. This Lyon representation, Figure A.1b, is rescaled so all entries are between zero and one. To speed up the processing in the MMV network somewhat, we also remove trailing columns with all-zero entries. To transform this representation into events, we apply a step-forward or delta encoding, with a threshold equal to 0.1 (Dupeyroux, 2021). This results in a rather sparse array of both positive and negative spikes, as shown in Figure A.1c. We chose the step-forward encoding because it is easy to implement in hardware and thus attractive for edge computing. Positive and negative events are then stacked above each other such that the final representation has only positive spikes, Figure A.1d. The training and testing set consisted of 4011 and 1079 exemplars, respectively. We trained MMV networks of $N = 200$ and 500 neurons, both with linear readouts and population coding readouts, as described in Section 4.2 for 200 epochs. For this experiment, we chose MMV periods in the range of 5 to 30. The spike rasters where shifted in column-wise. The remaining details of the training are identical to those found in Section 4.3. In Table A.1 we show our results. The network with 500 MMVs and a linear readout gave the best result at a testing accuracy of 99.26%. Remarkably, a logistic regression on the flattened binary-valued spike rasters resulted in an accuracy of 94.71%. This shows that the preprocessing described above works quite well. However, the logistic regression requires over 300k MAC operations per inference, while the linear readout of the $N = 200$ MMV network requires only 2000 MACs and still surpasses 98% test accuracy.

Table A.1: Heidelberg spoken digits (English)

Network/Algorithm	Accuracy [%]
200 MMVs + population coding readout	96.94
500 MMVs + population coding readout	98.52
200 MMVs + linear readout	98.70
500 MMVs + linear readout	99.26
Logistic regression	94.71

Appendix A.2. Repeated Application of Random Encoded Input

The Fashion-MNIST dataset contains grayscale images of 28×28 pixels of ten classes of clothing items. This dataset is similar to the MNIST handwritten digits, but with considerably more nuance in the pixel gray scales. We encode the images as follows: each pixel is randomly encoded with probability $P[\text{spike in}] = b$ with $b \in [0, 1]$ the pixel brightness. Then the randomly encoded image is applied column-by-column (and row-by-row) to the recurrent MMV network(s) as described in Section 4.1. This sequential processing is then repeated a few times, thus accumulating spikes over multiple applications of the same image, but each time with a different encoding. As before, a linear readout layer is applied to the total spike count to obtain the inference. Note that this is still sequential processing. Table A.2 shows our results. The readout layer is only calculated once per inference, so the cost for better accuracy is contained in having to generate the random encodings and run them through the network. For comparison, we include the result of a feedforward ANN with 2×500 neurons with arctan activation function followed by a linear layer.

Table A.2: Repeated applications of random encodings (Fashion MNIST)

Network	Rep.	Acc. [%]
500 MMVs rec. + lin. readout	5	87.25
500 MMVs rec. + lin. readout	8	87.69
500 MMVs rec. + pop. coding	8	85.88
500 MMVs rec. + lin. readout, thresholded	-	83.16
2×500 MMVs rec. + lin. readout	5	88.61
2×500 MMVs rec. + lin. readout	8	88.87
ANN 784 in $\rightarrow$ 500 $\rightarrow$ 500 + lin. readout	-	87.48

Appendix A.3. Mixed MMV and I&F Networks

MMVs with OR-ing interconnect can be combined with Integrate & Fire spiking neuron layers. In Table A.3 we show some early results on this approach on the MNIST handwritten digits dataset, sequentially processing the digits as outlined in Section 4.1. For these results, the I&F layer(s) had 4-bit quantized synaptic weights and 6-bit quantized firing thresholds.

Table A.3: Mixed MMV and I&F network results (MNIST handwritten digits)

Network	Acc. [%]
500 MMVs rec. + 10 ffwd I&F	98.59
2×256 MMVs rec. + 64 ffwd I&F + 10 ffwd I&F	98.82
4×128 MMVs rec. + 10 ffwd I&F	98.19

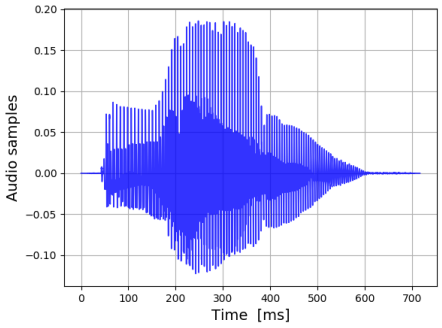
Appendix A.4. Verilog MMV Description

Here we give an example of a digital implementation of a non-retriggerable MMV in Verilog. A digital counter acts as a timekeeper. The period is given as a parameter to be chosen when instantiating the module. A second parameter sets the bit

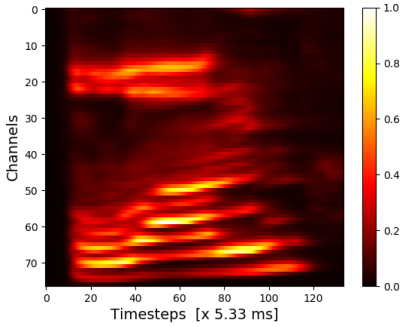
width of the registers. It is straightforward to make the period also programmable at run-time. Note a future research direction would be to investigate the use of the MMV state for gating signals, so this output has already been foreseen.

Listing 1: A minimal NRMMV implementation in Verilog

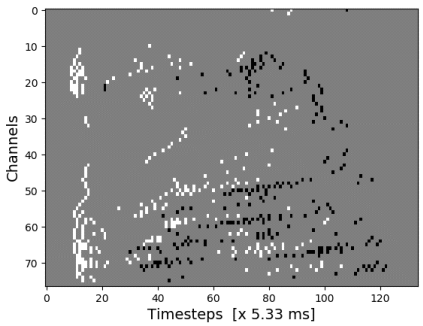
```
module nrmmv
#(parameter T = 10, parameter WIDTH=5)
(   input wire clock, //clock
    input wire reset, //asynchronous reset
    input wire exc,   //excitatory input
    input wire inh,   //inhibitory input
    output wire out,  //spike output
    output wire state //state output
);
reg [WIDTH-1:0] counter = {WIDTH{1'b0}};
reg spike;
reg spike2;

always @(posedge clock or posedge reset)
if (reset == 1'b1)
    begin
       counter <= {WIDTH{1'b0}};
    end
else
    begin
       if (inh)
          counter <= {WIDTH{1'b0}};
        else if (exc)
           begin
              if (counter==T)
                 //retrigger simultaneous
                 //with spike
                 counter <= 1;
              else
                 //counter is <T, so trigger
                 //if T was 0 or count up
                 counter <= counter + 1;
            end
        //when exc is 0
        else if (counter==T)
           counter <= {WIDTH{1'b0}};//to idle
           else if (counter!={WIDTH{1'b0}})
              //count up, if already triggered
              counter <= counter + 1 ;
           //else counter stays idle at zero
    end
//state output is 0=idle if counter is zero
assign state = (counter=={WIDTH{1'b0}})
                ? 1'b0:1'b1;

//change spike output at negedge
always @(negedge clock or posedge reset)
if (reset)
   begin
      spike <= 1'b0;
   end
else if (counter == T)
   begin
      spike <= 1'b1;
   end
      else
        begin
           spike <= 1'b0;
        end

assign out = spike;
endmodule
```



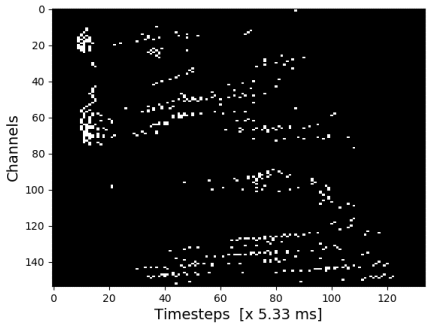
(a)



(b)



(c)



(d)

Figure A.1: (a) Waveform of an utterance of the word "one". (b) Lyon spectrogram of (a) normalized between zero and one. Each column represents 64 samples of sound at 12ksps or approximately 5.33 ms. (c) Step-forward spike encoding of (b), with positive (white) and negative (black) events. (d) Positive and negative events are stacked above each other into one raster of positive spikes, which is fed into the MMV network in a column-by-column fashion.