Monitoring of MTL Specifications With IBM's Spiking-Neuron Model

Konstantin Selyunin*, Thang Nguyen†, Ezio Bartocci* Dejan Nickovic‡ and Radu Grosu*
* Vienna University of Technology, Austria, † Infineon Austria, Villach, Austria
‡ Austrian Institute of Technology, Vienna, Austria
Email: *{Konstantin.Selyunin,Ezio.Bartocci,Radu.Grosu}@tuwien.ac.at,
†Thang.Nguyen@infineon.com, ‡Dejan.Nickovic@ait.ac.at,

***Abstract*— This paper shows how to use the IBM's TrueNorth spiking neuron model, for monitoring if a digital signal satisfies a metric temporal-logic (MTL) specification. TrueNorth spiking neurons are universal computation blocks, which can perform a variety of deterministic or stochastic tasks (e.g., Boolean/arithmetic operations, filtering, and convolution) depending on the configuration of their parameters. We show how to set these parameters for the deterministic TrueNorth neural-model in order to recognize MTL operators. A TrueNorth circuit then behaves as a runtime MTL monitor. We demonstrate how to translate the neural monitor to synthesizable HDL-code on Xilinx's Zedboard using high-level synthesis. To the best of our knowledge, this is the first application of the IBM's TrueNorth model for runtime monitoring. It also demonstrates the complete flow from a high-level specification to the implementation of a neural monitor in FPGA. As a byproduct, the paper also introduces the first open-source FPGA implementation of the deterministic TrueNorth model. We demonstrate the usefulness of our approach on a case study, the launching of a missile from a battle ship.**

***Index Terms*—Runtime Monitoring, TrueNorth, MTL**

I. Introduction

Dating back to Leonardo da Vinci and his *"Codex on the Flight of Birds"* (c.a. 1505), inspiration from nature greatly affected technological development. Human brain, as the most complex and advanced information processing system, is a research subject in itself [1] and a source of inspiration not only in natural sciences but also in hardware engineering.

In a recent series of papers [2]–[4], IBM has revealed its novel, human-brain-inspired, TrueNorth hardware architecture. A versatile and configurable neuron model [2] is at the core of the architecture and allows to build a rich suite of applications. IBM plans to commercialize a neuro-synaptic chip built upon this model, allowing cognitive ways of computing in hardware. We believe that brain-inspired architectures are the future of technological development with enormous research potential.

However, neither neuro-synaptic hardware nor its simulation environment are available today for public use. In this work we implemented the spiking TrueNorth neuronal model from [2] in C++, and make it available to the research community. We believe this will foster the interest of researchers in neuro-synaptic hardware and in novel hardware architectures. We also show how the deterministic part of the model can be translated to synthesizable HDL code and deployed in FPGA.

The universality of the TrueNorth model is particularly appealing: It allows both deterministic and stochastic computation, depending on the neurons' configuration. In this paper we show how to apply the TrueNorth model for runtime monitoring of Metric Temporal Logic (MTL) properties. Having identified how to recognize MTL operators with TrueNorth, we are able to build neural monitors from MTL formulae. As our ultimate goal is to build hardware monitors that provide both qualitative and quantitative information, with regard to a mixed-signal and a sighal-temporal-logic (STL) specification, we see TrueNorth as a perfect candidate for our framework. In this paper we tackle only the qualitative side of the problem.

The paper contributions can be summarized as follows:

1) To the best of our knowledge, we are the first to provide a publicly available implementation of TrueNorth model.
2) We present a fresh view of the MTL runtime-monitoring problem in terms of spiking neurons and their circuits.
3) We give the complete flow from the C++ implementation of neural circuits to synthesizable in FPGA monitors.
4) We demonstrate the usefulness of our approach on a case study, the launching of a missile from battle ship.

The rest of the paper is organized as follows: Section II discusses related work. Section III recalls the TrueNorth model and presents our hardware generation flow. Section IV focuses on applying the TrueNorth model for monitoring MTL specifications. Section V presents the case study and the experimental results. Section VI offers our concluding remarks.

II. Related Work

Runtime monitoring (or runtime verification [5]) is a light-weight state-of-the-art technique, used to ensure the conformance of a system, w.r.t. its specification. Low overhead, the ability to check a system as a "black-box" without the system's model, made runtime monitoring very appealing, both from a theoretical [6], [7] and a practical point of view [8], [9].

The authors in [6] explore overhead-accuracy trade-off of a runtime software monitor and use HMMs to retain high probability of being accurate while introducing gaps in observations. In [10] the authors provide a procedure to directly check properties of signals in continuous time and monitor mixed-signal specifications. Fast, hardware-based, online monitoring has also drawn a great deal of attention in recent years [11]–[13]. In particular: 1) Synthesis of monitors for safety and liveness LTL properties [13], 2) Design of sophisticated architectures to record events for MTL property checking [11], 3) Synthesis of checkers for PSL assertions [12]. Although concerned with hardware monitoring, all this work did not, however, consider applying neuronal architectures for this task.

978-3-9815370-6-2/DATE16/

In [2] IBM presents the TrueNorth neuronal model for their hardware architecture and states that the model, beside reproducing relevant biological behaviors, can also be used for deterministic and stochastic computation. While the use of the model in capturing biological behaviors is described in some depth, this is not the case for the definition of logical or arithmetic functions. In this paper we investigate how to configure the model to obtain the behaviors of interest. In [3] the authors consider the application of IBM's neuronal architecture for digit and tone recognition, HMM sequence modelling, and eye detection, with special spiking retina sensors. The authors develop these applications on very high levels of abstraction, by using so-called "corelets". Their description does not reveal implementation details or consider monitoring as a possible application. In contrast, we build our monitors on top of the TrueNorth neural model, starting from single neurons.

The leaky-integrate-and-fire (LIF) models capture relevant behavior of neural cells [1]: 1) Neuronal dynamics can be seen as summation process (i.e integration); 2) Membrane potential of neurons leaks over time to its resting value; 3) Neurons communicate information via spikes, the form of which is not important, and only their number over time is of relevance. LIF models capture this behavior in three steps:

- *Synaptic Integration:* gather inputs from other neurons;
- *Leak:* decrease membrane potential by a leak amount;
- *Firing & Reset:* spike and reset of excited neurons.

III. TrueNorth Neuron Model and its Hardware Implementation

We use the TrueNorth spiking neuronal model and give its concise description below for consistency of presentation. For extended explanation the reader should refer to [2]. The TrueNorth neural model extends all the stages of the LIF model and operates in purely digital fashion.

A. The Model

1) Synaptic Integration (Eq. 1): A TrueNorth neuron can differentiate between four types of axons. According to [2], each neuron can have up to 255 input connections, although an axon type $G_i \in \{0,1,2,3\}$ must be assigned to each connection. Computation on neurons from the same axon type G_i is either deterministic or stochastic, depending on the value of the flag $b_j^{G_i}$: In *deterministic* mode a neuron computes the sum of its inputs A_i weighted by $s_j^{G_i}$. In *stochastic* mode a neuron updates its state value (i.e. integrates) when a weight $s_j^{G_i}$ is greater than a random number $\rho_{i.j}$ drawn from a uniform distribution. Integration in stochastic mode [2] is a sum of signum functions of weights $\text{sgn}(s_j^{G_i})$:

$$V_j(t) = V_j(t-1) + \sum_{i=0}^{255} A_i(t) w_{ij} \big[(1 - b_j^{G_i}) s_j^{G_j} + b_j^{G_i} F(s_j^{G_i}, \rho_{i,j}) \text{sgn}(s_j^{G_i})\big] \quad (1)$$

2) Leak Integration (Eqs. 2–3): A TrueNorth neuron j can have either positive, negative or zero leak λ_j. Negative leak can be seen as an energy loss over time, while positive leak is self-stimulation. Leaking can be either deterministic or stochastic, depending on the value of the flag c_j^λ. In *deterministic* mode, neuron j changes its state V_j by a value λ_j. In *stochastic* mode a neuron leaks by value $\text{sgn}(\lambda_j)$ when leak value λ_j is greater than a random number ρ_j^λ drawn from a uniform distribution.

"Leak-reversal" regime [2] (controlled by a flag ϵ_j) is an extension of the LIF model to capture divergent and convergent leak behaviors. Leak sign in this mode depends on the sign of a membrane potential V_j: λ_j changes its sign whenever the sign of the membrane potential has been changed. When V_j and λ_j are of the same sign, the divergent leak behavior is obtained; when the signs are different, leak force V_j to converge to zero. In this regime the membrane does not leak when V_j is zero.

$$\Omega_j = (1 - \epsilon_j) + \epsilon_j \text{sgn}(V_j(t)) \quad (2)$$

$$V_j(t) = V_j(t) + \Omega \left[(1 - c_j^\lambda)\lambda_j + c_j^\lambda F(\lambda_j, \rho_j) \text{sgn}(\lambda_j)\right] \quad (3)$$

3) Threshold, Fire, Reset (Eqs. 4–9): TrueNorth model has both deterministic and stochastic firing/reset behavior. We describe the *deterministic* behavior first. A neuron j has two thresholds: a positive threshold α_j and a negative threshold β_j. When the neuron's state V_j exceeds the positive threshold α_j the spike is generated on an axon, and a neuron executes one of reset modes depending on a value γ_j. In reset mode **0**, V_j is reset to the value R_j; in a reset mode **1**, the membrane potential V_j is linearly decreased by the value α_j and in a reset mode **2**, V_j is kept unchanged.

When the membrane potential falls below the negative threshold β_j, the negative reset is executed, although no spike is generated on the output. Depending on the value of the flag κ_j membrane potential is either bounded by its negative threshold β (i.e. saturated) or it is updated according to a value of γ_j. In the mode **0**, V_j is updated to $-R_j$, in the mode **1**, V_j is increased by the value of β_j, and the mode **2**, is a non-reset, which means that the value of V_j does not change.

The *stochastic* reset and firing behavior is governed by stochastic thresholds: η_j is a random variable that adds stochasticity to α_j and β_j. The value of the η_j is computed as a bitwise `AND` of a random sample ρ_j^T and a bitmask M_j.

$$\eta_j = \rho_j^T \,\&\, M_j \quad (4)$$

$$\texttt{if} \quad V_j(t) \geq \alpha_j + \eta_j \quad (5)$$

$$\text{SPIKE} \quad (6)$$

$$V_j(t) = \delta(\gamma_j) R_j + \delta(\gamma_j - 1)(V_j(t) - (\alpha_j + \eta_j)) + \delta(\gamma_j - 2) V_j(t) \quad (7)$$

$$\texttt{elseif} \quad V_j(t) < -\left[\beta_j \kappa_j + (\beta_j + \eta_j)(1 - \kappa_j)\right] \quad (8)$$

$$V_j(t) = \beta_j \kappa_j + \big[- \delta(\gamma_j) R_j + \delta(\gamma_j - 1)(V_j(t) + (\beta_j + \eta_j)) + \delta(\gamma_j - 2) V_j(t)\big](1 - \kappa_j) \quad (9)$$

After having identified the model, we now formulate the problem and show top-level view of our solution.

B. Problem Definition

Let $\mathbb{T}$ denote a discrete finite time interval $[0, r] \cap \mathbb{N}$, $\mathbb{B} \in \{0, 1\}$. A Boolean signal is a function $w : \mathbb{T} \to \mathbb{B}^n$. Given a MTL specification φ and a signal w, out task is to devise a monitor that delivers a verdict whether w violates φ. In the

paper we are interested in building synthesizable in FPGA hardware MTL monitors using the TrueNorth model.

We use neurons to recognize sub-formulae of φ: a neural circuit then represents a monitor. At each time step a TrueNorth neuron either spikes or not, which can be seen as outputting a binary signal. Since neurons compute on outputs of parent neurons, we can supply an external signal w into the circuit using special "input" neurons (`MockTrueNorthNeuron`'s in our implementation) which behave in accordance with w. At each time step the neural monitor accepts w as input and outputs a SPIKE if the specification has been fulfilled. The absence of a SPIKE at the output of the neural monitor corresponds to a specification violation.

In order to build neural monitors for MTL specifications using TrueNorth, we have to be able to express logical and (bounded) temporal operators in terms of the model [2].

C. Solution: Top Level View

In this section we describe the flow from getting a natural language specification of a property to a hardware TrueNorth neural monitor in an FPGA (see Fig. 1).

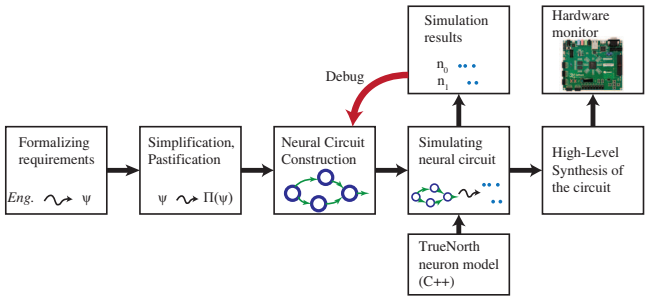

Figure 1: Neural Monitor Generation Flow

1) Formalizing requirements: In the first step a natural-language requirement must be converted to an MTL formula to eliminate ambiguity and possible misinterpretation. It is a non-trivial problem and still an open issue [14] how to express mutual dependencies and temporal behavior and translate it in a formal language. We illustrate how the conversion is done in a case study in Section V.

2) Pastification, Simplification: As we are able to devise a verdict based on the behavior that has already happened, every bounded future formula must be pastified to obtain an equisatisfiable formula containing only past operators. We use two-step pastification procedure from [15]: (i) compute temporal depth D of the formula and (ii) convert a bounded-future formula to a past one.

3) Constructing a neural monitor: We then define a circuit topology and instantiate the parameters of TrueNorth neurons that will correspond to a monitor of a past-MTL specification obtained in the previous step. We use configuration parameters for logical and temporal operators from Section IV and traverse the parse tree of the formulate replacing each node in the tree with a neural monitor.

4) Circuit simulation: We implemented the model described in Section III-A in C++ and use this implementation to simulate circuits of TrueNorth neurons (available in [16]). We instantiate neurons as objects of the `TrueNorthNeuron` class, set their parameters and define neurons' connections. We are then able to test the circuit on the external input and observe outputs for every neuron, which is useful for debugging and attesting configurations of neurons.

5) HLD code generation with High-Level Synthesis: Given the circuit from Step III-C4 and C++ implementation of TrueNorth model, our task in this step is to obtain an FPGA-ready synthesizable representation of a neural monitor. This is the final step of hardware implementation of a neural monitor. We use High-Level Synthesis (HLS) from Xilinx [17] to convert a C++ description of a neural circuit to a synthesizable HDL code (Verilog or VHDL). Given a C++ function that represents a monitor for a sub-formula of the initial specification, we generate HDL code with HSL according to the following steps: 1) Define a test suite to compare a function that uses hardware-specific (bit-precise) data types with a "golden" function that uses generic software types; 2) Generate HDL with RTL synthesis. 3) Compare the functionality of the synthesized code from 2) with the "golden" function. 4) Export the synthesized code as an IP (available upon request). These IPs can be then imported in development tools (e.g. Vivado or PlanAhead) for generating bit-stream for a specific chip.

IV. MONITORING WITH TRUENORTH MODEL

Our specification language in this paper is MTL [18] interpreted over discrete time. In this section we start with giving a formal definition of MTL and then show how to build temporal testers for past fragment of it using the TrueNorth model. Using the conversion procedure [15] we are not restricted only to past but are also able to monitor formulae, equisatisfiable to bounded future-MTL specifications, after linear time preprocessing in the size of the formula. The authors in [19] showed that MTL is decidable in discrete time.

A. Metric Temporal Logic

The syntax of an MTL formula φ with past and future operators over a set of boolean variables $P = \{p_1, \cdots, p_m\}$ is defined by the following grammar [20]:

$$\varphi := p \,|\, \neg\varphi \,|\, \varphi_1 \vee \varphi_2 \,|\, \varphi_1 \mathcal{U}_I \varphi_2 \,|\, \varphi_1 \mathcal{S}_I \varphi_2,$$

where $p \in P$, I is $[a, b]$, $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. Other MTL operators are derived from the definition in a standard way: $\top = \varphi \vee \neg\varphi$; $\bot = \neg\top$; eventually $\Diamond_I \varphi = \top \mathcal{U}_I \varphi$; once $\Diamond\!\!\!-_I \varphi = \top \mathcal{S}_I \varphi$; always $\Box_I \varphi = \neg \Diamond_I \neg\varphi$; historically $\boxminus_I \varphi = \neg \Diamond\!\!\!-_I \neg\varphi$.

The semantics of an MTL formula is defined as follows:

$$\begin{array}{lcl}
(w,i) \models p & \leftrightarrow & p[i] = \top \\
(w,i) \models \neg\varphi & \leftrightarrow & (w,i) \not\models \varphi \\
(w,i) \models \varphi_1 \vee \varphi_2 & \leftrightarrow & (w,i) \models \varphi_1 \text{ or } (w,i) \models \varphi_2 \\
(w,i) \models \varphi_1 \mathcal{U}_I \varphi_2 & \leftrightarrow & \exists j \in (i+I) \cap \mathbb{T} : (w,j) \models \varphi_2 \\
 & & \text{and } \forall i < k < j, (w,k) \models \varphi_1 \\
(w,i) \models \varphi_1 \mathcal{S}_I \varphi_2 & \leftrightarrow & \exists j \in (i-I) \cap \mathbb{T} : (w,j) \models \varphi_2 \\
 & & \text{and } \forall j < k < i, (w,k) \models \varphi_1
\end{array}$$

B. Parameter synthesis for neural monitors as ILPs

To configure TrueNorth neurons as monitors of MTL specifications, we need to find their parameters (i.e. synaptic weights, leak, thresholds, etc.). For each function (either logical or bounded temporal) our goal is to devise constraints that characterize its behavior: If, for example, for given combination of inputs we require a neuron to output a SPIKE then at the current time step the value of a membrane potential V_j of a neuron j after synaptic and leak integration steps should exceed positive threshold α_j. The configuration task can be seen as a search in parameter space for a point that satisfies all constraints. Since all the parameters of the TrueNorth model [2] are either integers or booleans, the configuration problem for a TrueNorth neuron can be stated as an *Integer Linear Program* (ILP) [21] with zero objective function.

1) Logical Operators with TrueNorth: To configure neurons for executing logical functions, we need to impose memoryless behavior on the TrueNorth model, which has memory (i.e. membrane potential V_j). This is achieved by forcing a neuron j to reset its internal state V_j at every time point by executing either positive or negative reset steps in the reset mode $\mathbf{0}$ ($\gamma_j = 0$): At each time step a combination of inputs should either exceed α_j (when SPIKE on this particular combination of inputs is required) or stay below β_j (when no SPIKE is required on the input). By assigning $R_j = 0$ we keep V_j at zero at the beginning of every time step.

Logical operators AND, OR, NOT, NAND, NOR and implication can be computed with one neuron.

a) Example: 2-NAND: To illustrate our approach, we configure a TrueNorth neuron to perform a two input NAND operation (see Fig. 2). The input neurons n_0, n_1 provide stimulus to the neuron n_2, the parameters of which we need to find in this example.

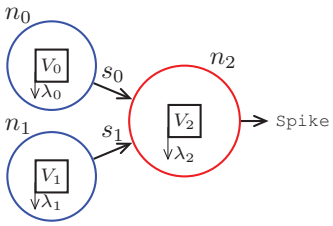

Figure 2: Two input TrueNorth circuit

To reproduce the behavior of interest, the output neuron n_2 must spike at every time step unless both n_0 and n_1 are active: in this case n_2 must be inhibited and its membrane potential V_2 must fall below β_2; In all other cases V_2 after leak integration V_2 must exceed α_2. LHS of the inequalities represent the value of V_2 after synaptic and leak integration for different combination of inputs. The ILP is given in Eq. 10.

$$\begin{array}{rrrcl} \multicolumn{5}{c}{\min 0 \quad \text{s.t.}} \\ & & \lambda_2 & \geq & \alpha_2 \\ & +s_1 & +\lambda_2 & \geq & \alpha_2 \\ s_0 & & +\lambda_2 & \geq & \alpha_2 \\ s_0 & +s_1 & +\lambda_2 & < & \beta_2, \end{array} \tag{10}$$

where s_0 and s_1 are synaptic weights of neurons n_0 and n_1 respectively; λ_2 is a leak weight of n_2; α_2 and β_2 are positive and negative thresholds of n_2 respectively. This problem can be now solved using a standard ILP solver (e.g. we use `intlinprog` from MATLAB). By similar argument we devise constraints and find parameters of different logical functions (see Table I for the results for two-input case). Cases with more input and composed logical functions can be also stated as ILPs and solved analogously.

Table I: Neural Parameters of Logical Operators

	s_0	s_1	λ_2	α_2	β_2	γ_2	c_2^λ	ϵ_2	M_2	κ_2
AND	9	9	-14	4	-4	0	0	0	0	0
OR	9	9	-5							
NOT	-4	-	4							
NOR	-9	-9	4							
NAND	-9	-9	13							
$\rightarrow$	9	-9	-5							

2) Temporal Operators with TrueNorth: We employ the idea of temporal testers [22] to construct monitors for MTL specifications in a compositional way. We build temporal testers on top of the TrueNorth model, i.e. use TrueNorth neurons to recognize past-LTL and past-MTL operators by leveraging the internal state V_j and different reset modes.

a) The Once *Operator* $\diamondsuit$*:* The semantics of the *Once* operator according to [23] is as follows: $(w,i) \models \diamondsuit\, p$ iff $(w,k) \models p$ for some $k, 0 \leq k \leq i$. The temporal tester for $\diamondsuit$ spikes continuously after p has happened. We need one TrueNorth neuron for this task. Fig. 3 shows a circuit where n_1 computes $\diamondsuit\, p$, where p denotes a predicate "n_0 spikes". The constraints that govern behavior of n_1 are as follows:

- To be non-forgetful, leak weight λ_1 must be zero,
- To fire when the first Spike from n_0 arrives, $s_{01} \geq \alpha_1$;
- To continue firing after the first occurrence of Spike on n_0, reset mode $\mathbf{0}$ with $R_1 > \alpha_1$, (see Tab. II).

b) The Previous *Operator* $\ominus$*:* A neural temporal tester for $\psi = \ominus\, \phi$ needs to postpone a satisfaction of ϕ for one time step. The circuit consists of two neurons (see Fig. 3) that compute in an inverse order: At each time step, n_2 computes on input from n_1, and n_1 compute on input neuron n_0, which results in shifting satisfaction of ϕ by one time step.

c) The Punctual Once *Operator* $\diamondsuit_{\{a\}}$*:* Since $\psi = \diamondsuit_{\{a\}}\, \varphi$ postpones satisfaction of φ over a time steps, we implement this operator as a cascade of previous operators as in [24]. This implementation requires $2a$ neurons.

$$\diamondsuit_{\{a\}}\, \varphi \quad = \quad \underbrace{\ominus\, \ominus \ldots \ominus}_{a}\, \varphi$$

d) The Bounded Once *Operator* $\diamondsuit_{[0,a]}$*:* The temporal tester for $\psi = \diamondsuit_{[0,a]}\, \varphi$ uses six TrueNorth neurons. According to its semantics, the tester should spike when φ is satisfied, and output spikes for a time steps after last satisfaction of φ. We build a circuit of four neurons that captures last satisfaction of φ: $\neg\varphi \wedge \ominus\, \varphi$. Its output activates special "bounded once" neuron, which operate in non-reset mode, has one leak weight and input weight from auxiliary circuit of a. Setting R to 1 allows us obtain a concecutive spikes after last satisfaction of φ. The output of the tester is an OR neuron, that combines input with output from a "bounded once" neuron.

e) The Historically *Operator* $\boxminus$*:* A temporal tester for $\psi = \boxminus\, \varphi$ takes a neuron n_0 and its an inverse n_1 as inputs (see Fig. 3). According to the semantics from [23], whenever a SPIKE from n_1 arrives at the tester, its membrane potential should fall below β_2 and its output never produces a SPIKE. To

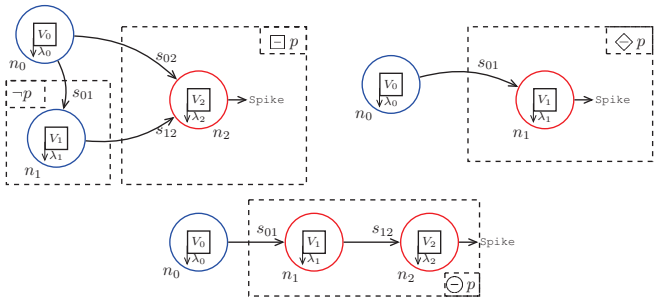

Figure 3: Neural Temporal Testers

configure a neuron as a tester, we find parameters that satisfy the constraints (see Tab. II):

$$\begin{array}{cccccc} & +s_{02} & & +\lambda_2 & \geq & \alpha_2 \\ & & +s_{12} & +\lambda_2 & < & \beta_2 \\ +R_2 & +s_{02} & & +\lambda_2 & \geq & \alpha_2 \\ +R_2 & & +s_{12} & +\lambda_2 & < & \beta_2 \\ -R_2 & +s_{02} & & +\lambda_2 & < & \beta_2 \\ -R_2 & & +s_{12} & +\lambda_2 & < & \beta_2 \end{array} \tag{11}$$

f) The bounded Historically *Operator* $\boxminus_{[0,a]}$*:* To construct a temporal tester for $\psi = \boxminus_{[0,a]}\,\phi$ we use the circuit in Fig. 3 but configure n_2 differently. We use the reset mode **2** and set the positive threshold $\alpha_2 = as_{02}$: this allows us to count satisfaction of ϕ over consecutive time steps. To be able to reset the state of n_2 when ϕ gets violated we set the negative threshold to zero, use saturate mode ($\kappa_2 = 1$) and the largest negative weight possible for s_{12} to execute negative reset mode (see Tab. II). To check satisfaction on $[a, b]$ we use the fact that $\boxminus_{[a,b]}\,\phi = \diamondsuit_{\{a\}}\,\boxminus_{[0,b-a]}\,\phi$.

g) The Since *Operator* $\mathcal{S}$*:* To build a temporal tester for $\psi = \phi_1\,\mathcal{S}\,\phi_2$ we need four neurons. The definition states that ϕ_2 happened at some time in the past and ϕ_1 held at every time point from the occurrence of ϕ_2 till present. We use an argument from [24] for constructing the tester: the tester must satisfy $\phi_2(t) \Leftrightarrow \psi(t)$ at the first time step and $\psi(t) \Leftrightarrow (\phi_2(t) \vee (\phi_1(t) \wedge \psi(t-1)))$ starting from the second time step. The tester comprises $\wedge$, $\vee$ and $\ominus$ neurons.

h) The bounded Since *Operator* $\mathcal{S}_{[0,b]}$*:* We implement a tester for $\psi = \phi_1\,\mathcal{S}_{[0,b]}\,\phi_2$ using a rewriting rule from [24]: $\phi_1\,\mathcal{S}_{[0,b]}\phi_2 = (\phi_1\,\mathcal{S}\phi_2) \wedge \diamondsuit_{[0,b]}\,\phi_2$. As a combination of two testers, this tester requires 11 neurons.

Table II: Neural Temporal Testers

	in_0	in_1	λ_i	α_i	β_i	γ_i	R_i	ϵ_i	κ_i
$\diamondsuit$	7	-	0	4	-4	0	5	0	0
$\ominus$	1	-	0	1	0	0	0	0	0
	1	-	0	1	0	0	0	0	0
$\diamondsuit_{\{a\}}$	a chain of $\ominus$								
$\diamondsuit_{[0,a]}$	combination of $\ominus, \wedge, \neg, \vee$ and a "core" neuron (below)								
	INT_MIN	a	-1	1	0	2	0	0	1
$\boxminus$	0	-18	4	4	-4	0	9	0	0
$\boxminus_{[0,a]}$	1	INT_MIN	0	a	0	2	0	0	1
$\mathcal{S}$	combination of $\wedge, \vee, \ominus$								
$\mathcal{S}_{[0,b]}$	combination of $\wedge$, $\mathcal{S}$, $\diamondsuit_{[0,a]}$								

V. Case Study and Experimental Results

As a case study we build a hardware neural monitor for a safety property that describes the launch of a missile from a ship. Suppose that the launch is governed by three signals: "launch enable": ℓ, "fire enable": f and "detonation": d. The signal ℓ characterizes whether the missile is allowed to be launched; the signal f describes the moment when a missile has been fired; the d signal is sent to trigger detonation.

The property we monitor is a safety property of the ship in a sense that it describes correct order of actions, timing and absence of damage to the ship from its missile:

"When the missile received the launch enable signal, it must see the fire enable signal followed within the next four time points. After fire en has arrived, no detonation is allowed for the next five time points." (see Fig. 4).

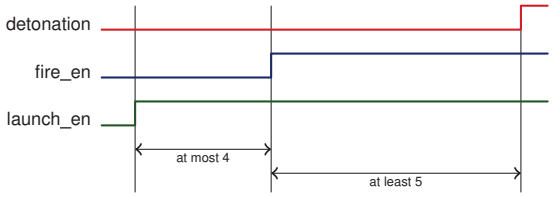

Figure 4: Missile timing property specification

The specification above can be almost directly expressed in MTL: the relation that the rising edge of f must appear within four time steps after the edge of ℓ is by definition bounded eventually; no detonation for five time points is a bounded always of negation of d signal:

$$\uparrow \ell \rightarrow \diamondsuit_{[0;4]} \left(\uparrow f \wedge \square_{[0;5]}\, \neg d\right), \tag{12}$$

where $\uparrow \varphi$ is a shortcut for MTL formula $\varphi \wedge \ominus \neg\varphi$ which denotes a rising edge of φ.

We then convert the bounded future MTL formula (Eq. 12) to an equisatisfiable past one. After performing a Step III-C2 we obtain an equisatisfiable past MTL specification:

$$\diamondsuit_{\{9\}} \uparrow \ell \rightarrow \diamondsuit_{[0,4]} \left(\diamondsuit_{\{5\}} \uparrow f \wedge \boxminus_{[0;5]}\, \neg d \right) \tag{13}$$

In the next step we construct a neural circuit which represents a monitor. Each sub-formula from Eq. 13 is converted to a corresponding neural circuit. After composing a neural circuit we simulate it using our C++ implementation of TrueNorth model (Fig. 5 shows simulation results).

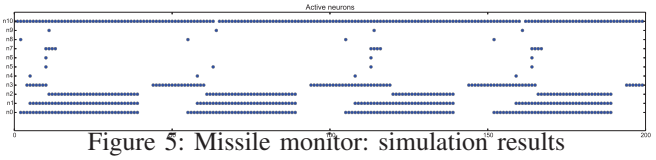

Figure 5: Missile monitor: simulation results

We use Zedboard from Xilinx and Vivado System Edition [25] (University License) for generating neural monitors (see Fig. 6). During HLS conversion we leveraged hardware specific data types to assign all the flags in the model exactly one bit-width and reduced native data types proportional to the width of Xilinx multipliers to save hardware resources. Each operator has been synthesized separately to foster reuse for other specifications. We then implemented the demonstrator on FPGA which generates ℓ, f and d signals and checks the monitor under nominal conditions and fault injections. FPGA resources for each operator are listed in Tab. III (FF and LUT stand for flip-flop and look-up tables, respectively).

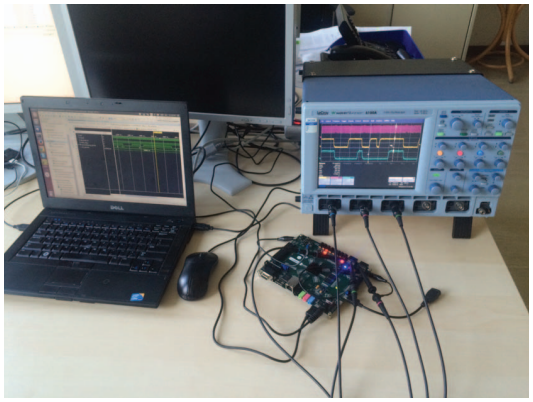

Figure 6: Experimental setup (see [26] for oscillograms)

Table III: Missile monitor: implementation results

Operator	# of neurons	FPGA resources	
		FF	LUT
$\rightarrow$	1	25	82
$\Diamond_{\{9\}}$	18	443	1623
$\uparrow \ell$	4	98	325
$\Diamond_{[0,4]}$	6	146	1085
$\wedge$	1	25	82
$\Diamond_{\{5\}}$	10	246	920
$\uparrow f$	4	98	325
$\boxminus_{[0;5]} \neg d$	2	48	164
Total:	46	1129 (1.1%)	4606 (8.6%)

VI. Conclusion and Outlooks

In this paper we applied TrueNorth, a novel versatile spiking neural model from IBM, for monitoring MTL specifications over digital signals, and producing synthesizable hardware monitors. We demonstrated our approach on a case study where we build a hardware runtime monitor for a missile launch from natural language requirement. In future we plan to investigate advanced functionality of the model and create a unified framework based on TrueNorth for both qualitative and quantitative reasoning about mixed-signals.

Acknowledgement. This research is supported by the project HARMONIA (845631), funded by a national Austrian grant from FFG (Österreichische Forschungsförderungsgesellschaft) under the program IKT der Zukunft.

References

[1] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. New York, NY, USA: Cambridge University Press, 2014.

[2] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B. dayan Rubin, E. Mcquinn, W. P. Risk, and D. S. Modha, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *in International Joint Conference on Neural Networks (IJCNN). IEEE*, 2013.

[3] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. V. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico, S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. A. Kusnitz, T. M. Wong, W. P. Risk, E. McQuinn, T. K. Nayak, R. Singh, and D. S. Modha, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores." in *IJCNN*. IEEE, 2013, pp. 1–10.

[4] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, E. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-icaza, E. Mcquinn, B. Shaw, N. Pass, and D. S. Modha, "Cognitive Computing Programming Paradigm: A Corelet Language for Composing Networks of Neurosynaptic Cores," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013.

[5] M. Leucker, "Teaching Runtime Verification," in *Runtime Verification*, ser. Lecture Notes in Computer Science, S. Khurshid and K. Sen, Eds. Springer Berlin Heidelberg, 2012, vol. 7186, pp. 34–48.

[6] S. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. Smolka, and E. Zadok, "Runtime Verification with State Estimation," in *Runtime Verification*, ser. Lecture Notes in Computer Science, S. Khurshid and K. Sen, Eds. Springer Berlin Heidelberg, 2012, vol. 7186, pp. 193–207.

[7] K. Kalajdzic, E. Bartocci, S. Smolka, S. Stoller, and R. Grosu, "Runtime Verification with Particle Filtering," in *Runtime Verification*, ser. Lecture Notes in Computer Science, A. Legay and S. Bensalem, Eds. Springer Berlin Heidelberg, 2013, vol. 8174, pp. 149–166.

[8] C. Watterson and D. Heffernan, "A runtime verification monitoring approach for embedded industrial controllers," in *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, June 2008, pp. 2016–2021.

[9] D. Borrione, M. Liu, K. Morin-Allory, P. Ostier, and L. Fesquet, "On-line assertion-based verification with proven correct monitors," in *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on*, Dec 2005, pp. 125–143.

[10] O. Maler and D. Nickovic, "Monitoring properties of analog and mixed-signal circuits," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 3, pp. 247–268, 2013.

[11] T. Reinbacher, M. Fgger, and J. Brauer, "Runtime verification of embedded real-time systems," *Formal Methods in System Design*, vol. 44, no. 3, pp. 203–239, 2014.

[12] A. Dahan, D. Geist, L. Gluhovsky, D. Pidan, G. Shapir, Y. Wolfsthal, L. Benalycherif, R. Kamidem, and Y. Lahbib, "Combining system level modeling with assertion based verification," in *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on*, March 2005, pp. 310–315.

[13] K. Claessen, N. Een, and B. Sterin, "A circuit approach to LTL model checking," in *Formal Methods in Computer-Aided Design (FMCAD), 2013*, Oct 2013, pp. 53–60.

[14] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta, "Formalization and validation of safety-critical requirements," in *Proceedings FM-09 Workshop on Formal Methods for Aerospace, FMA 2009, Eindhoven, The Netherlands, 3rd November 2009.*, 2009, pp. 68–75.

[15] O. Maler, D. Nickovic, and A. Pnueli, "On synthesizing controllers from bounded-response properties," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds. Springer Berlin Heidelberg, 2007, vol. 4590, pp. 95–107.

[16] "Source Code," https://github.com/selyunin/truenorth_cpp_hls.git.

[17] "Vivado High-Level Synthesis," http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html *(Accessed 15.09.2015)*.

[18] J. Ouaknine and J. Worrell, "Some Recent Results in Metric Temporal Logic," in *Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems*, ser. FORMATS '08. Springer-Verlag, 2008, pp. 1–13.

[19] R. Alur and T. Henzinger, "Real-time logics: complexity and expressiveness," in *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, Jun 1990, pp. 390–401.

[20] O. Maler, D. Nickovic, and A. Pnueli, "From MITL to Timed Automata," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, E. Asarin and P. Bouyer, Eds. Springer Berlin Heidelberg, 2006, vol. 4202, pp. 274–289.

[21] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*. Department of operations and research and financial engineering, Princeton university, 2001.

[22] A. Pnueli and A. Zaks, "On the Merits of Temporal Testers," in *25 Years of Model Checking*, ser. Lecture Notes in Computer Science, O. Grumberg and H. Veith, Eds. Springer Berlin Heidelberg, 2008, vol. 5000, pp. 172–195.

[23] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.

[24] S. Jakšić, E. Bartocci, R. Grosu, R. Kloibhofer, T. Nguyen, and D. Ničković, "From Signal Temporal Logic to FPGA Monitors," in *International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2015, pp. 1–10.

[25] "Vivado System Edition," http://www.xilinx.com/products/design-tools/vivado.html *(Accessed 15.09.2015)*.

[26] "Oscillograms and Lab Results," https://www.dropbox.com/sh/s3jy1zux5y9uvk5/AAC9KOAWFzYSAXOJntR6CosIa?dl=0 *(Accessed 12.11.2015)*.