INTEGRATION, the VLSI journal 54 (2016) 109–117

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Invited paper

Digital implementation of a virtual insect trained by spike-timing dependent plasticity ☆

P. Mazumder [a,*], D. Hu [a], I. Ebong [a], X. Zhang [b], Z. Xu [b], S. Ferrari [b]

[a] University of Michigan, Ann Arbor, MI 48109, USA
[b] Duke University, Durham, NC 27708, USA

ARTICLE INFO

Article history:
Received 25 June 2015
Accepted 18 January 2016
Available online 13 February 2016

Keywords:
Spike timing dependent plasticity
Neural network

ABSTRACT

Neural network approach to processing have been shown successful and efficient in numerous real world applications. The most successful of this approach are implemented in software but in order to achieve real-time processing similar to that of biological neural networks, hardware implementations of these networks need to be continually improved. This work presents a spiking neural network (SNN) implemented in digital CMOS. The SNN is constructed based on an indirect training algorithm that utilizes spike-timing dependent plasticity (STDP). The SNN is validated by using its outputs to control the motion of a virtual insect. The indirect training algorithm is used to train the SNN to navigate through a terrain with obstacles. The indirect approach is more appropriate for nanoscale CMOS implementation synaptic training since it is getting more difficult to perfectly control matching in CMOS circuits.

1. Introduction

The neural network approach to data processing has undergone continued research and development even with the widespread success of the von Neumann architecture, traditionally sequential in nature. Recent widespread advancement of the von Neumann architecture to utilize multi-core processors [2] is similar to the neural network approach, providing a much needed boost to the area. The difference between the parallelism of multi-core processors and that of neural networks is the latter uses much less complex processing elements, therefore, allowing opportunities for massively parallel structures. Hardware neural networks or neuromorphic circuits have been around for quite some time with proposals that span both digital CMOS and analog CMOS approaches [3–6]. Specific VLSI reviews and methodologies are provided in [7,8]. Although neural hardware have been proposed, implemented, and commercialized, their widespread adoption is still unrealized. Neural software implementations running on digital computers are much more prevalent, leaving hardware adoption behind. Hardware implementations have found niche uses in peripheral devices and various subsystems [3]. Software implementations have the advantage of ease of programming through well-known languages like C and C++, large software engineering support due to a lower barrier of entry for software engineers than those for hardware engineers, high precision calculations if the processing capabilities are present, and more flexibility regarding the implemented algorithm. Even with these advantages, hardware implementations are still sought after because of the speed associated with hardware computing; realization of adequate neural processors or neurocomputers will enable applications that require real-time processing, feedback, and learning. The most promising implementations use the digital components and have granted programmable neurocomputers like CNAPS [9] and SYNAPSE-1 [10]. Although neurocomputers are very powerful, efforts have been made to scale down applications to even less powerful machines, ones that run on battery and do not rely on a large number of processing elements. These efforts have led to the widespread appeal of spiking neural networks [4,11–15].

Spiking neural network (SNN) implementations provide a powerful computation fabric where a smaller number of processing elements can potentially be utilized in order to realize desired functionality. When working with SNN implementations, usually the designer provides different, specified goals during design phase. The goals determine the level of detail needed when designing the neuron and synapse behavior. Hardware neural networks have been used to study different phenomena in biological neural networks. This has brought about different neuron models with their hardware implementations. Hardware neural networks seek to be simple in functionality in order to minimize

☆This work was supported by the National Science Foundation under ECCS Grant 1059177 and CCF Grant 1421467.

* Corresponding author.
E-mail addresses: pinakimazum@gmail.com (P. Mazumder), hudi@umich.edu (D. Hu), idong@eecs.umich.edu (I. Ebong), xz70@duke.edu (X. Zhang), dec.ziyer@gmail.com (Z. Xu), sferrari@duke.edu (S. Ferrari).

http://dx.doi.org/10.1016/j.vlsi.2016.01.002
0167-9260/

the area associated with this processing element. This brings about the massive tradeoff between complex models like the Hodgkin–Huxley and the leaky integrate and fire (LIF) [15]. Since SNN solutions require numerical solutions with no closed form representations, their behavior in hardware is much harder to predict. Therefore, learning algorithms tenable to hardware adoption are crucial to pushing widespread SNN adoption. Software SNN implementations are widespread; hardware implementations need to catch up, hence the thrust behind this work. The contributions in this paper are: abstraction and mapping of a complex learning process to a digital spiking neural network fabric. A digital approach is used in order to encourage repeatability when dealing with complex SNNs. A virtual bug example is used to illustrate the strength of this algorithm. Section 2 will provide details on the model of the virtual insect. Section 3 will expand on the specific example by showing top level neural network organization, the training algorithm, and the CMOS circuit adaptation. Section 4 will provide simulation results and discussion, and Section 5 relays some concluding remarks.

2. Virtual insect model

The test setup and scenario chosen is a virtual insect (bug) model. The virtual insect model is constructed to demonstrate and evaluate the indirect training algorithm [1] and hardware-level rapid prototyping design. Offline training with limited information is adopted for the chosen application. After training the virtual insect, the virtual insect is used in a homing application where it is used to find a given target on a two-dimensional space with obstacles.

The virtual insect is a moniker based on the given construct in Fig. 1, since the sensors are attached to the body like antennae on a biological insect. The virtual insect is modeled as a rigid object that can move in any direction on a map. Fig. 1 shows the external structure and environment of the virtual insect. The environment of the virtual bug consists of obstacles which are denoted as black objects and a target which is denoted as a bright spot on the map. The virtual bug has four sensors which provide terrain and target information. The bug has an elliptical shape and is symmetric along its major axis. On each symmetric half, a target sensor, a terrain sensor, and a motor is modeled. By convention, the labels for these sensors and motors are either “Left” or “Right,” depending on which half they reside as depicted in Fig. 1.

The target sensor generates a signal S_{target} based on the distance between the sensor and the target. By convention, the further the virtual insect is from the target, the higher the magnitude or intensity of S_{target}. The terrain sensor generates a signal $S_{terrain}$ based on the roughness of the map, with a rougher map corresponding to a more intense or higher magnitude of $S_{terrain}$. The two

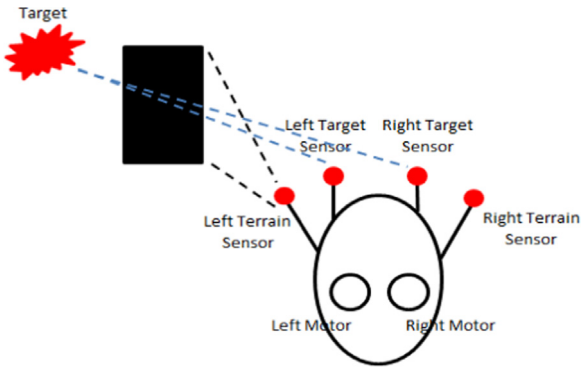

Fig. 1. External structure of the virtual insect.

motors effect the direct motion of the insect. Intuitively, if the left motor has a higher revolutions per minute (rpm) compared to the right motor, the insect will turn right. The insect turns left if the right motor rotates faster than the left motor. If the two motors have the same rpm, then the virtual insect will move forward in its direction of orientation. The motion of the virtual insect is restricted in that its motors are allowed to rotate in only one direction. Therefore, the insect is incapable of reversing (moving opposite its oriented direction). If the insect needs to follow the direction opposite its direction of orientation, it will need to turn towards that direction then move forward.

Since virtual insect motion is determined by both the relative angular velocities of the two motors and the current sensor inputs regarding proximity to obstacles and the target, the internal connection between the sensors and the motors is described.

When the insect moves in the prescribed environment, its motion can be described by (1), adapted from the modified unicycle robot locomotion in [16,'1]. By restricting the environment to a 2D Cartesian plane, when the insect moves, its linear velocity can be described as v. v can be decomposed into components in both the x-direction and y-direction, denoted in (1) as v_x and v_y, respectively. v_{left} and v_{right} are the speeds of the left and right motors, respectively. θ is the variable used to represent direction of orientation and is defined as the angle between the major axis of the elliptical insect and the x-axis. L, τ_{motor} and η are scaling constants; and t_L^f and t_R^f are the firing times of output neurons (more on this later).

$$\begin{cases} v_x = v \times \cos(\theta) \\ v_y = v \times \sin(\theta) \\ v = \frac{v_{left} + v_{right}}{2} \\ \Delta\theta = \frac{v_{right} - v_{left}}{L} \\ \Delta v_{left} = -\frac{v_L}{\tau_{motor}} + \eta * \left(t = = t_L^f\right) \\ \Delta v_{right} = -\frac{v_R}{\tau_{motor}} + \eta * \left(t = = t_R^f\right) \end{cases} \quad (1)$$

According to (1), Δv_{left} and Δv_{right} are always negative and become positive only when $t = t_L^f$ or when $t = t_R^f$, respectively. This translates to a motor's rpm, v_{left} or v_{right}, is always decreasing unless its corresponding output neuron spikes. Therefore, the more frequently an output neuron fires or spikes, the faster the speed of the corresponding motor. The next section will elucidate the connections between the output neurons and how its spiking events are controlled. Essentially, the dependence on the firing frequency of an output neuron on the synaptic weights of the neural network reduces the training of the virtual insect to weight parameter adjustment.

3. Spike-based training approach

3.1. Top level NN organization

In the previous discussion, the control of the motors was due to a spiking pattern of the outputs of some neural network. This section provides the top level architecture and inherent connectivity of the spiking neural network (SNN) controlling the motors. The SNN architecture (illustrated in Fig. 2) resembles a feedforward neural network with an input layer, a hidden layer and an output layer. The input layer interfaces with the four sensor inputs while the output layer interfaces with the two motors. The two output layers are shown as separate entities to make clear there is no interconnectivity between the two layers. Additionally, the structure of the SNN is flexible (i.e. each layer can have any number of neurons and can be of any shape, and the connections