frontiers in
NEUROSCIENCE

ORIGINAL RESEARCH ARTICLE
published: 21 November 2014
doi: 10.3389/fnins.2014.00379

# FPGA implementation of a biological neural network based on the Hodgkin-Huxley neuron model

***Safa Yaghini Bonabi[1]*, Hassan Asgharian[2], Saeed Safari[3] and Majid Nili Ahmadabadi[1,4]***

[1] *Cognitive Robotic Lab., School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran*
[2] *Research Center of Information Technology, Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran*
[3] *High Performance Embedded Computing Lab., School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran*
[4] *School of Cognitive Sciences, Institute for Research in Fundamental Sciences, IPM, Tehran, Iran*

***Edited by:***
*Tim Pearce, University of Leicester, UK*

***Reviewed by:***
*Alejandro Linares-Barranco, University of Seville, Spain*
*Guillaume Garreau, Johns Hopkins University, USA*

****Correspondence:***
*Safa Yaghini Bonabi, Cognitive Robotic Lab., School of Electrical and Computer Engineering, College of Engineering, University of Tehran, North Kargar, Tehran 14395-515, Iran*
*e-mail: safa.yaghini@ut.ac.ir*

A set of techniques for efficient implementation of Hodgkin-Huxley-based (H-H) model of a neural network on FPGA (Field Programmable Gate Array) is presented. The central implementation challenge is H-H model complexity that puts limits on the network size and on the execution speed. However, basics of the original model cannot be compromised when effect of synaptic specifications on the network behavior is the subject of study. To solve the problem, we used computational techniques such as CORDIC (Coordinate Rotation Digital Computer) algorithm and step-by-step integration in the implementation of arithmetic circuits. In addition, we employed different techniques such as sharing resources to preserve the details of model as well as increasing the network size in addition to keeping the network execution speed close to real time while having high precision. Implementation of a two mini-columns network with 120/30 excitatory/inhibitory neurons is provided to investigate the characteristic of our method in practice. The implementation techniques provide an opportunity to construct large FPGA-based network models to investigate the effect of different neurophysiological mechanisms, like voltage-gated channels and synaptic activities, on the behavior of a neural network in an appropriate execution time. Additional to inherent properties of FPGA, like parallelism and re-configurability, our approach makes the FPGA-based system a proper candidate for study on neural control of cognitive robots and systems as well.

**Keywords: Hodgkin-Huxley, neural pool, neural network, digital hardware implementation, FPGA**

## INTRODUCTION

Developing computational tools for simulating the brain networks is of a very special interest, because the models provide powerful means for investigating different characteristics of the neural system. For example, they can be used to find the effect of malfunctioning voltage-gated channels on network level behaviors in specific brain diseases or are employed to track effects of learning on synaptic efficacies and neural behavior. Using the computational tools before undertaking biological experiments can also give some insights into the results of experiments. In addition, computational models can be used as controllers for cognitive robots.

Neurons and neural pools are basis of computational models. Neurons receive sensory signals, process the information, excite/inhibit each other through a complex electrochemical process (Kandel et al., 2000). A neural pool is a group of neurons sharing excitatory or inhibitory property. Neural pools can inhibit or excite each other by means of output signals. Therefore, activity of each neuron can affect the behavior of its pool and other pools in the brain, so specific behavior of the neural networks emerge from interaction of neurons and neural pools (Buzsáki, 2004).

A neural network can be implemented on software or hardware. Due to the sequential execution of software, the parallel nature of neural networks is affected which leads to reduction of execution speed. Implementing neurons on hardware can provide several benefits; including high-speed modeling of big neural networks and preparing responses in real-time, etc. (Indiveri et al., 2011). In the hardware implementation techniques, digital implementations are more preferred vs. analog implementations, based on some of the digital advantages such as noise-robustness, more flexibility, simple real-world interfaces, and easier testability (Muthuramalingam et al., 2008). In addition, digital implementations are more cost-effective and less time consuming (Gatet et al., 2009). One of the other benefits of the digital implementations is their capability for fast development (Indiveri et al., 2011). There are different methods for digital implementation such as ASIC (Application Specific Integrated Circuit), DSP (Digital Signal Processing), and FPGA (Field Programmable Gate Array). DSP-based implementations are not suitable for modeling the parallel behavior of the neurons because of their sequential nature. ASIC implementation is more efficient than FPGA in terms of power and area, but it suffers from lack of re-configurability (Wanhammar, 1999). FPGA has some benefits over DSP and ASIC that we are interested in: it is reconfigurable, so it is useful for rapid prototyping of neural networks (Wang et al., 2014), and it has parallel processing architecture.

Therefore, FPGA could be an appropriate solution for hardware implementation of neural networks (Muthuramalingam et al., 2008).

FPGAs have limited usable area and design tool chains, which create difficulty in implementation of large neural networks. Therefore, designs should be optimized in size to be implemented on an FPGA with limited number of resources. In this paper, we design and implement a biologically plausible neural network on an FPGA. There are different biological neuron models; however, we opt for Hodgkin-Huxley (H-H) neural model because of its biological plausibility and inclusion of synaptic details. Due to the FPGA area limitation, we use the reduced order of the previous implementation (Bonabi et al., 2012a). The implemented neural network in this work is a modified model used in Moldakarimov et al. (2005). The implemented network is the basic component of many neural networks; it has two mini-columns, each has two neural pools: an excitatory pool with 60 neurons and an inhibitory pool with 15 neurons. The reason that we are interested in this model is that this model can be used to investigate the competition between neural pools in the brain (Bakhtiari et al., 2012). Using the hardware as an accelerator for reducing the computation time of such neural networks is our main objective. The basis of single pool implementation are stated in Bonabi et al. (2012b). We use MATLAB simulations for high-level design of neural network and the results of simulations are used as a golden model to check correctness of our implementation.

In Materials and Methods, we introduce the model at the neural and the network levels. FPGA implementation of the network is introduced in this section, too. In Results, the validation processes and the implementation results for a set of variables are given. In addition, the hardware system and the speed of processing are discussed. Discussions and Conclusions are given in the last section.

## MATERIALS AND METHODS

### NEURON MODEL

The neuron model used in this implementation is the reduced version of the model introduced in Traub and Miles (1991). Ermentrout and Kopell (1998) and Börgers et al. (2005) also used the reduced model to investigate the dynamical behaviors of neural networks. The structure of the model for both of the inhibitory and excitatory neurons is the same. The membrane potential of the neurons follows the H-H equations (Hodgkin and Huxley, 1952). The ionic current that is mainly composed of sodium and potassium ions controls the membrane voltage. Moreover, this voltage regularizes the current flow of the ions by means of voltage-dependent ion channels. There are other ionic currents such as chloride, which their gating variables are independent of the membrane voltage. These ions constitute the leak current. Inside and outside of the membrane do not have equal concentration of ions, which results in an electrical potential. Both concentration gradient and electrical potential controls the current flow. The H-H model is shown in **Figure 1**.

According to Börgers et al. (2005); Izhikevich (2007) the relationship between input current, membrane voltage, and the complete set of H-H equations come in (1a–4c):

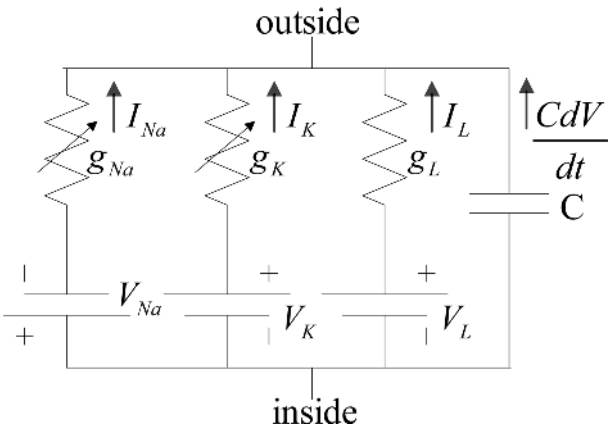


**FIGURE 1 | Hodgkin and Huxley proposed circuit for squid giant axon.** $g_K$ and $g_{Na}$, are voltage-dependent conductance (Hodgkin and Huxley, 1952).

$$\frac{CdV}{dt} = I - I_{Na} - I_K - I_L \tag{1a}$$

$$I_{Na} = \overline{g_{Na}} m^3 h\,(V - V_{Na}) \tag{1b}$$

$$I_K = \overline{g_K} n^4\,(V - V_K) \tag{1c}$$

$$I_L = g_L\,(V - V_L) \tag{1d}$$

Where,

$$m = m_\infty\,(V) = \frac{\alpha_m\,(V)}{[\alpha_m\,(V) + \beta_m\,(V)]} \tag{2a}$$

$$\alpha_m\,(V) = \frac{0.32\,(V + 54)}{1 - exp\,[-0.25\,(V + 54)]} \tag{2b}$$

$$\beta_m\,(V) = \frac{0.28\,(V + 27)}{exp\,[0.2\,(V + 27)] - 1} \tag{2c}$$

$$h = max\,(1 - 1.25n, 0) \tag{3}$$

$$\frac{dn}{dt} = 0.7\,(\alpha_n\,(V)\,(1 - n) - \beta_n\,(V)\,n) \tag{4a}$$

$$\alpha_n\,(V) = \frac{0.01\,(V + 34)}{1 - exp\,[-0.1\,(V + 34)]} \tag{4b}$$

$$\beta_n\,(V) = 0.125exp\,[-0.0125\,(V + 44)] \tag{4c}$$

The typical values and units of the parameters that used in the above equations according to Börgers et al. (2005) are as follows: $C = 1\,\mu F/cm^2$, $\overline{g_{Na}} = 100\,mS/cm^2$, $V_{Na} = 50\,mV$, $\overline{g_K} = 80\,mS/cm^2$, $V_K = -100\,mV$, $g_L = 0.1\,mS/cm^2$, $V_L = -67\,mV$.

The units of the letters $V$, $t$, and $I$ are $mV$, $ms$, and $\mu$A/cm$^2$ respectively.

### NEURAL NETWORK MODEL

The implemented neural network is composed of two similar mini-columns. **Figure 2** shows the network model. Each mini-column has an excitatory and an inhibitory pool. All of the

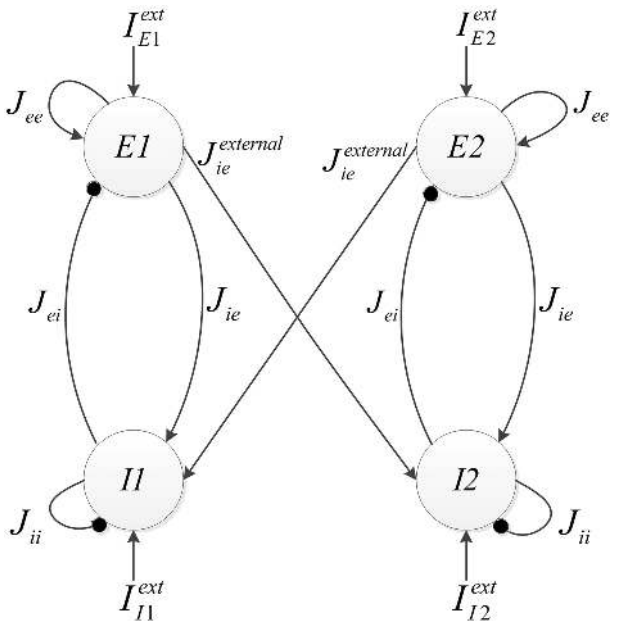


**FIGURE 2 | The structure of neural network, consist of two competitive mini-columns.**

neurons in each pool are connected to each other. By means of specialized structures called synapses, the information is transmitted among neurons. According to Börgers et al. (2005) neurons in each pool are related to each other with neurotransmitter, AMPA (E → E and E → I) and $GABA_A$ receptors (I → I and I → E). In order to create a relationship between the two mini-columns, an excitatory synaptic current is given to the inhibitory pool in the adjacent mini-column. In **Figure 2**, $J_{ee}$, $J_{ie}$, $J_{ii}$, $J_{ei}$, and $J^{external}_{ie}$ are the weights of neuron connections. These coefficients indicate the strength of synaptic connections of the excitatory neurons to each other, the synaptic strength of the excitatory neurons to the inhibitory neurons in the same mini-column, the strength of synaptic connections of the inhibitory neurons to each other, the synaptic strength of the inhibitory neurons to the excitatory neurons, and the synaptic strength of the excitatory neurons to the inhibitory neurons in the adjacent mini-column, respectively.

Each excitatory pool receives two synaptic currents: one from its own neurons and the other from the neurons of the inhibitory pool in the same mini-column (see **Figure 2**). Equation (5) shows these currents. In this equation, $V_{ee}$ and $V_{ei}$ are equal to 0 and −80, respectively.

$$I^{(1)}_{syne} = J_{ee} g^{(1)}_e \left(V_{ee} - V_e[j]\right) + J_{ei} g^{(1)}_i \left(V_{ei} - V_e[j]\right) \quad (5)$$

Each inhibitory pool receives three synaptic currents as described in Equation (6). One synaptic current is exerted from the excitatory pool in the same mini-column, the second one is from its own neurons, and the last one is from the excitatory pool of the adjacent mini-column.

$$I^{(1)}_{syni} = J_{ie} g^{(1)}_e \left(V_{ie} - V_i[j]\right) + J_{ii} g^{(1)}_i \left(V_{ii} - V_i[j]\right) + J^{external}_{ie} g^{(2)}_e \left(V_{ie} - V_i[j]\right) \quad (6)$$

In Equation (6) $V_{ie}$ is 0 and $V_{ii}$ is −80. Equation (5) is added to Equation (1a) for excitatory pools and Equation (6) is added to Equation (1a) for inhibitory pools. As $V_{ii}$ and $V_{ei}$ are considered less than the minimum value of the action potential of a neuron, the corresponding synaptic current is always negative, so it is always opposed to increase the voltage value. Thus, this can be an expression of the inhibitory synaptic current.

The total effect of the synaptic variable of the excitatory and inhibitory pools on post-synaptic neurons is shown by $g_X$, $X \in \{e, i\}$ in Equation (7). Based on all-to-all connections in each pool, $g_X$ is equal for all neurons in the same pool and it is obtained by the average of pre-synaptic neurons effects in each pool. In Equation (7), $N_X$ is the number of neurons in each pool.

$$g_X = \frac{\sum_{k=1}^{N_X} s_X[k]}{N_X} \quad (7)$$

According to Börgers et al. (2005), $s_X[k]$ is the gating variable, which is calculated by the following equation. $\tau_R$ and $\tau_D$ for AMPA receivers are equal to 0.2, 2 and for $GABA_A$ receivers are equal to 0.5, 10, respectively.

$$\frac{ds}{dt} = \frac{1 + \tanh\left(\frac{V}{10}\right)}{2} \frac{1 - s}{\tau_R} - \frac{s}{\tau_D} \quad (8)$$

The data transmission rate of neurotransmitters between the pre-synaptic and post-synaptic neurons is modeled by the mechanism of synapse. The term [1 + tanh(V/10)]/2 can be assumed as a normalized neurotransmitter concentration (Börgers et al., 2005).

## FPGA IMPLEMENTATION

In this section, the FPGA implementation of the neural network, shown in **Figure 2**, is described. According to **Figure 2**, this network is made of two similar mini-columns, each has two neural pools: excitatory and inhibitory. The neurons of the excitatory and inhibitory pools have the same structure and there is no significant difference in their synaptic mechanisms. The selected single neuron model for implementation is described completely in Bonabi et al. (2012a) and we made a few changes in the model in this work. We reduced the order of dynamics of the system to make it simpler for the implementation of big neural networks. As Equations (1a–4c) describe, the dynamics of $m$ and $h$ are neglected. The most difficult part in the implementation of equations is the implementation of the exponential function. The accuracy and performance in the implementation of the exponential function has an intensive impact on the results. We use hyperbolic Coordinate Rotation Digital Computer (CORDIC) algorithm, which could be implemented using simple shifters and adders, to calculate the function. Equations (9a–c) shows the CORDIC algorithm used in our implementation (Ercegovac and Lang, 2003).

$$x[j+1] = x[j] + \sigma_j 2^{-j} y[j] \quad (9a)$$

$$y[j+1] = y[j] + \sigma_j 2^{-j} x[j] \quad (9b)$$

$$z\left[j+1\right] = z\left[j\right] - \sigma_j tanh^{-1}(2^{-j}) \tag{9c}$$

In order to minimize the required FPGA resources, we use LUT for implementing hyperbolic tangent inverse function in the Equation (9c). Therefore, we achieve less running time, because only a simple search is required instead of a mathematical calculation. CORDIC algorithm can calculate the value of functions with a reasonable error only when the inputs are in a limited range (Ercegovac and Lang, 2003). According to Equation (10), in order to have a bigger range of inputs, we separate each input into two parts: integer part and fractional part. We precalculate the exponential of the integer part using MATLAB and save them in a LUT. The exponential of the fractional part is calculated using the implemented CORDIC module. Then, in order to produce the final value of the exponential function, the output of LUT is multiplied by the output of the implemented CORDIC module.

$$A = B + C \Longrightarrow \exp(A) = \exp(B) \cdot \exp(C) \tag{10}$$

Another module needed to produce new values of $n$, $s$, and $V$ from their dynamics is the integrator. The method used to implement this module is the same as our previous work (Bonabi et al., 2012a). Equation (11) is used to implement the integrator.

$$x(t + \Delta t) = x(t) + \Delta t.\dot{x}(t) \tag{11}$$

In this work, we add a multiplexer (Mux) in order to insert the initial values for the dynamics $n$, $s$, $V$. **Figure 3** shows the block diagram of the implemented module for calculating the integral function. In order to calculate the integral of each step, a register is used to save the previous values of the integral.

The basis of implementations of the excitatory and the inhibitory pools are the same and it is similar to the method presented in Bonabi et al. (2012b). We only change the number of neurons and the number of synaptic currents in each pool. Neurons are connected to each other with neurotransmitters, which make synaptic current. Synaptic current, as shown in Equations (5, 6), depends on the total effects of the synaptic variable and the gating variable. Therefore, for calculating synaptic current, we have to implement a module that could calculate the gating variable. The block diagram in **Figure 4** is used to implement the gating variable, $s$, which is shown in Equation (8).

In order to calculate the hyperbolic tangent block in **Figure 4**, we use Equation (12) by employing our implemented exponential function.

$$tanh(x) = \frac{exp(2x) - 1}{\exp(2x) + 1} \tag{12}$$

To reduce the number of multipliers, a one-bit left shifter is used to duplicate the input of hyperbolic tangent. These currents are added to the implemented pools in order to make a connection between the neurons in each pool, between excitatory and inhibitory pools in each mini-column, and excitatory and inhibitory pools in the different mini-columns. Because of the limited usable area, the implemented mini-column has five neurons, four of them are in excitatory pool, and the last one is in inhibitory pool. The block diagram of this neural pool is shown in **Figure 5**.

**Figure 6** shows the controller that is designed for this mini-column. This mini-column finishes its work when both of the pools finish their processing.

To implement the neural network with 150 neurons, two mini-columns with 75 neurons are needed which are connected to each other by a synaptic current. Because of the resource limitation of FPGA, we share resources by multiplexing in time to implement this neural network on a single chip. **Figure 7** shows the architecture that is used to increase the number of neurons in each mini-column. In this architecture, we use a ROM to put the

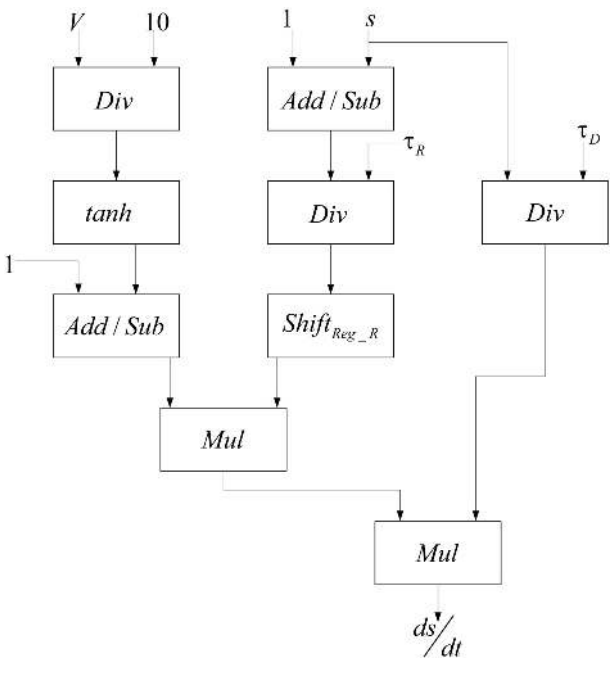


**FIGURE 4 | Designed module to implement the gating variable *s*.**

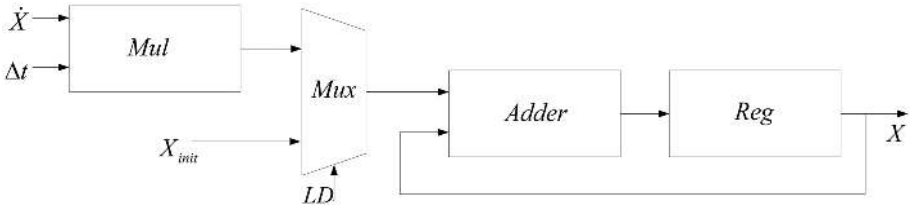


**FIGURE 3 | Block diagram of implemented module for calculating the integral function.**

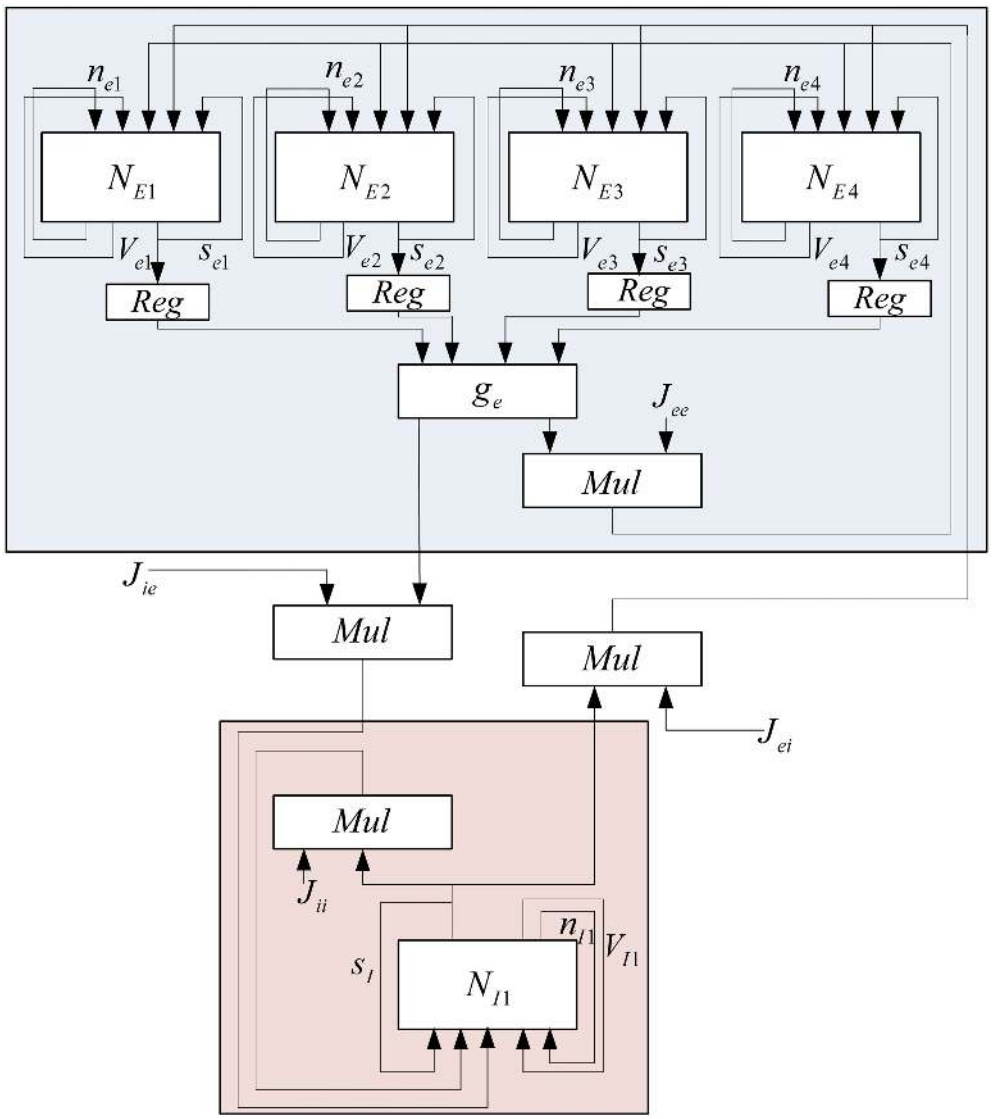


**FIGURE 5 | Block diagram of implemented mini-column with five neurons.**

initial values of membrane voltages and gating variables for each of the neurons. The RAM is used to write the responses of the neurons and use them for the next executions of the system. Both of the ROM and RAM have 15 rows; each row has data for all of the five neurons in the mini-column. Each row is used in each execution. The Mux is used to choose the data from ROM and RAM. The address generator generates the address to read from the ROM/RAM and write in the RAM.

The proposed architecture has a specific controller, shown in **Figure 8**, which controls all the processes. We have a counter in this controller, which manages the interactions of all components. The proposed architecture and the controller work in these steps:

- At first, the controller resets the circuit in order to set the values of all components to zero and the controller moves to the next state.
- In the second state (Read Input) the select input of Mux is set to zero and the inputs of the mini-column is read from the first line of the ROM. For the next 15 executions of this system, the select input of Mux is zero to choose the data from ROM. After first 15 executions, the select input is changed to one and data will be read from RAM. After reading the inputs of mini-column, the controller moves to the next state.
- The controller will stay in this state (Process) until the responses of neurons (five neurons in the mini-column) are ready. When the ready output of the mini-column is activated, the state of controller changes to the next state (Write Output).
- In the Write Output state, the answer of five neurons (membrane voltages and gating variables) is written in the same row of the RAM that data was read from it. If the address reaches to 15, the controller changes its state to Output Ready state; otherwise, the next state will be Check Status. Then, after a clock

cycle, the state of the controller will be changed to Read Input, address generator points to the next row of the ROM/RAM, and these steps will be repeated.

- If the value of the address generator reaches 15, the controller changes its state to Output Ready. In this state, the ready output of mini-column with 75 neurons is activated. Then, all of the synaptic currents will be calculated and given to the pools. The controller resets the counter in the address generator, so address generator will point to the first row of the RAM and these steps will be repeated until the desired final time.

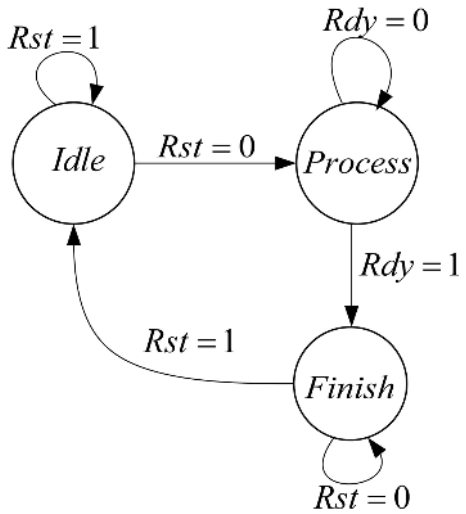


**FIGURE 6 | Designed controller for the implemented mini-column.**

In the neural network, two mini-columns should be connected to each other by synaptic currents. To implement this connection when $g_X$ is calculated in each mini-column, it will be entered to the inhibitory pool of the adjacent mini-column by coefficient of $J_{ie}^{external}$ and the synaptic current will be calculated in the inhibitory pools. The block diagram of the implemented neural network is shown in **Figure 9**.

**Figure 10** shows the designed controller for the implemented neural network. At first, the controller resets both of the mini-columns and they start to work from the $S_{Start}$ state. Then, mini-columns are activated together and work as a parallel system until both of them prepare the response of their 75 neurons. Each mini-column that finishes working, activates its ready output (Rdy). According to the ready output of the mini-columns, the controller changes its state. For example, if the ready output of the first mini-column is activated ($Rdy_1 = 1$), the controller changes its state to $S^B_{Rdy1}$. In the new state, the clock of this mini-column is deactivated. Then, the controller changes its state to the next state ($S_{Rdy1}$). State of the controller would not change until the responses of all of the neurons in the other mini-column are prepared. Then, the controller moves to the next state ($S^A_{Rdy1}$), the

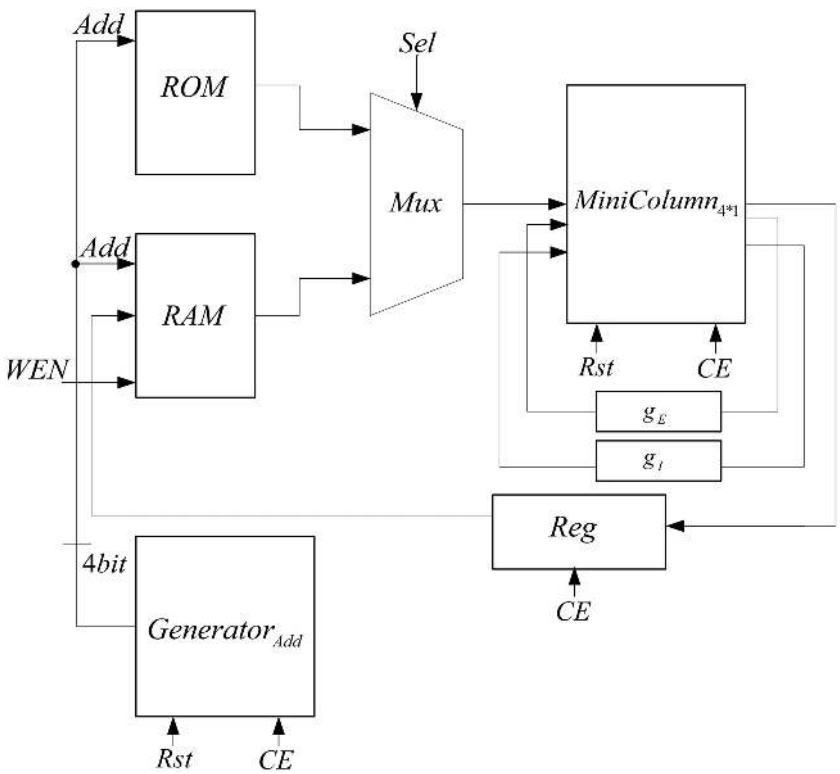


**FIGURE 7 | The proposed architecture to increase the number of neurons in the mini-column.**

clock pulse of both of the mini-columns are deactivated in this state, and after a clock cycle the controller moves to the final state ($S_{Finish}$). In the final state, the results are written in the memories of the mini-columns (RAM) and the controller changes its state to the first state ($S_{Start}$). The controller repeats this process until the desired time.

In order to add the effect of stochastic factors, we add a noise term to the input current of each neuron. This random term creates small differences between neurons, which makes the implementation more biologically plausible. The noises are generated from a zero mean Gaussian distribution for each neuron. Then, they are saved in a ROM that is shown in **Figure 9** ($I_{Noise}$). Each given address to the ROM by the address generator gives noises to the neurons of the two mini-columns.

## RESULTS

To validate the implemented neural network on FPGA, the deployed bit level simulations are compared with the numerical implementations of the mathematical models in MATLAB Simulink.

All of the hardware components are designed and implemented using VHDL modeling language. The main objective in the design of the low-level implementation is the accuracy of the output. Due to the hardware constraints, in the implementation of all modules minimum number of bits are used. More details of implementation are given in **Table 1** as synthesis results.

The membrane voltage of the implemented single neuron (blue line) and MATLAB simulation (red line) for $I_{ext} = 0.7$ mA are shown in **Figure 11**. According to this figure, there is a small difference between the membrane voltages of the implemented model on FPGA and MATLAB simulation, because of the rounding error in the digital implementation. However, the firing rates of both of the signals are equal.

In an excitatory pool (when there is no connection between the pools of a mini-column) each neuron receives an excitatory synaptic current and external stimulus. Therefore, a neuron in an excitatory pool receives two exciting current while a free single neuron receives just an external stimulus. Thus, the firing rate of a neuron in an excitatory pool is higher than a free single neuron. Each neuron in the inhibitory pool receives an inhibitory synaptic current that reduces the effect of external stimulus. Therefore, the firing rate of a neuron in the inhibitory pool would be less than the firing rate of a free single neuron. In order to investigate the effects of synaptic currents in the pools, we exerted a certain stimulation ($I_{ext} = 0.7$ mA) to a free single neuron, an excitatory pool, and an inhibitory pool. The result shows that in the excitatory pool the firing rate is more than the firing rate of the free single neuron and in the inhibitory pool it is less than the firing rate of the free single neuron. The firing rate in the single neuron, excitatory pool, and inhibitory pool are shown in the **Table 2**.

In each mini-column, the excitatory pool receives an inhibitory current from the inhibitory pool that reduces the firing rates of the neurons. **Figure 12** shows the membrane voltages of one neuron in the excitatory and inhibitory pools, given $I_E = 0.8$ mA, $I_I = 0.7$ mA, $J_{ie} = 0.7$, and $J_{ei} = 0.2$. According to the voltages in this figure, when the neurons in the inhibitory pool reach their maximum value, they prevent the neurons in the

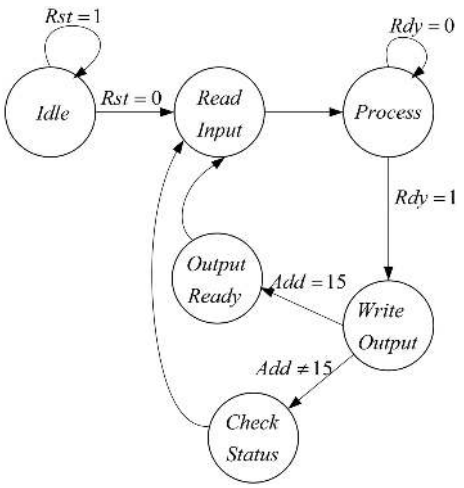


**FIGURE 8 | The designed controller for the implemented architecture to increase number of neuron in the mini-column.**

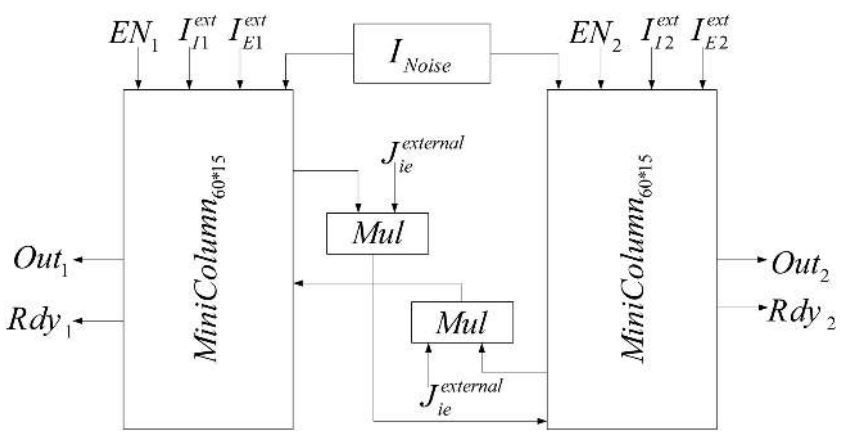


**FIGURE 9 | Block diagram of implemented neural network.**

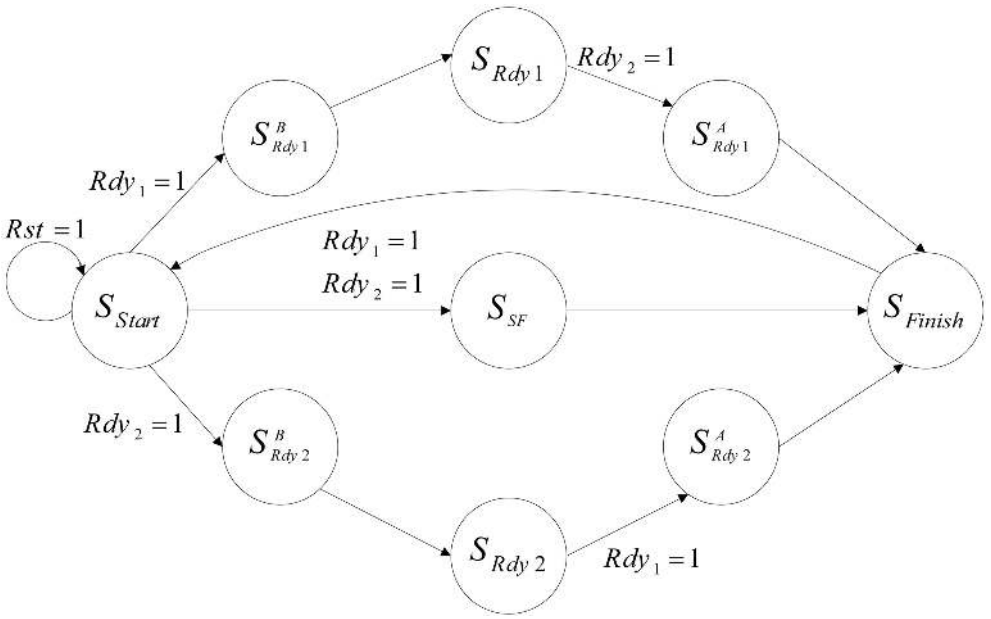


**FIGURE 10 | The designed controller, which is used to control the neural network.**

**Table 1 | Abstract of synthesis results.**

| Criteria | Virtex-7 | |
|---|---|---|
| | xq7k410t-2l-rf676 | |
| | Used | Utilization (%) |
| Frequency | 63.386 MHz | – |
| No. of LUTs | 86032 | 33 |
| No.of LUT-FF pairs | 30528 | 28 |
| No.of slice registers | 50228 | 9 |
| No.of DSP blocks | 1112 | 72 |
| No.of BRAM | 14 | 1 |

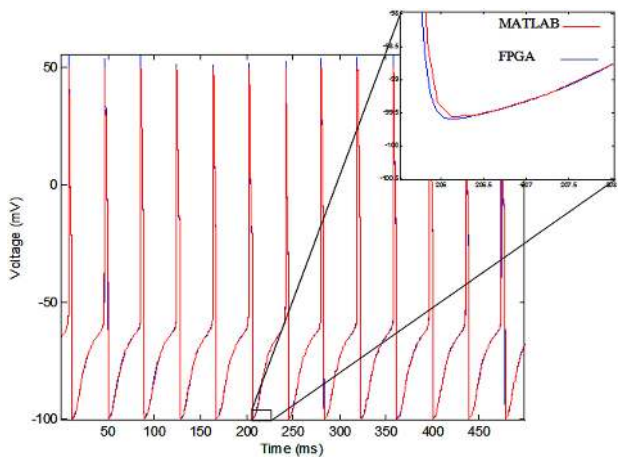


**FIGURE 11 | The voltage of single neurons, implemented single neuron (blue line) and MATLAB simulation (red line).**

excitatory pool to complete their firing. Therefore, as you can see in **Figure 12A** small bumps are created in the shape of the voltages of the excitatory neurons. In addition, the firing rate of the neurons in the excitatory pools is reduced.

When mini-columns are connected to each other, and a stimulation excites one of the excitatory pools in the network, both of the inhibitory pools will be activated and they try to inhibit their excitatory pools. For example, consider a case when the excitatory pool in the second mini-column ($E2$) receives greater stimulation than the excitatory pool in the first mini-column ($E1$). $E2$ and $E1$ exert excitatory synaptic currents to both of the second and first inhibitory pools ($I2$ and $I1$ respectively) (see **Figure 2**). $I1$ and $I2$ try to suppress the effect of the stimulations to $E1$ and $E2$. Since $E1$ is less stimulated than $E2$, the inhibiting effect of $I1$ might be able to suppress the neural activity in $E1$. To investigate this on the implemented system, we injected the same currents to both of the inhibitory pools and $E1$ is stimulated with less external current than $E2$. As rastergrams in **Figure 13** shows, the neurons in $E1$ are strongly suppressed by $I1$, while $I2$ is not able to suppress $E2$. Rastergrams of $I1$ and $I2$ looks almost the same, because both of them are excited by $E1$ and $E2$ and they receive equal external stimulations. The external currents and weights of neuron connections are as follow:

$$I_E^{(2)} = 0.85\,mA,\ I_I^{(2)} = 0.7\,mA,\ I_E^{(1)} = 0.7\,mA, I_I^{(1)} = 0.7\,mA$$

$$J_{ee} = 0.1, J_{ie} = 0.7, J_{ii} = 0.02, J_{ei} = 0.2, J_{ie}^{external} = 0.45.$$

## DISCUSSION

There is a range of neuron models: from abstract to biophysically plausible ones (Bailey et al., 2011). The computational efficiency

of models is inversely proportional to inclusion of details in them. Therefore, simpler models are more widely used. Abstract neuron models are usually used in the artificial neural network implementations; see Khan et al. (2006); Sahin et al. (2006); Çavuşlu et al. (2011) as examples. Interest in implementation of spiking neuron models is also increased in the recent years (Yang et al., 2011; Ambroise et al., 2013).

Several implementations of biologically plausible neuron models and neural networks have also been proposed. For example, in Zou et al. (2006), the authors have implemented a neural ınetwork based on the H-H neuron model using analog ASIC and a computer. In another example, in Heo and Song (2012) a VLSI implementation of a biological neuron model is presented. Each specific ıimplementation has its own advantages over any general-purpose implementation, like realization on FPGA. Nevertheless, the inherent parallel processing nature of FPGA devices, in addition to their shorter design time and online re-configurability, make them a good alternative for implementation of biologically plausible neural networks.

There are FPGA implementations of simple neuron models; see Chen and Wang (2002) and Yang et al. (2011) for Integrate-and-Fire and Leaky Integrate-and-Fire models respectively. In Bailey et al. (2011) Cellular Automata is implemented on FPGA. In Cassidy and Andreou (2008); Mokhtar et al. (2008); Rice et al. (2009); Ambroise et al. (2013) the Izhikevich model, which is a compromise between biological and abstract neuron models, has been implemented on FPGA. We are interested to have a more biologically plausible neuron model; so the H-H neuron model is chosen. There are different FPGA implementations of the H-H neuron model (Graas et al., 2004; Zhang et al., 2009; Pourhaj et al., 2010; Saïghi et al., 2010a,b; Grassia et al., 2011). A LUT-based implementation of computing algorithms is used in Graas et al. (2004), that has better time complexity, but more area is needed compared to the direct implementation. In addition, it has lower accuracy than direct algorithmic implementation. Therefore, a large size of memory is used to save the pre-computed results with limited number of bits, which reduces the final accuracy because of the limitations on memory size. Moreover, in Graas et al. (2004) MATLAB System Generator (SG) is used to implement a neuron and it is obvious that SG cannot produce an optimal hardware. Pourhaj et al. (2010) used

**Table 2 | Firing rates of a neuron in single neuron and pools.**

| | Single neuron | Excitatory pool | Inhibitory pool |
|---|---|---|---|
| Firing rate (HZ) | 26 | 30 | 16 |

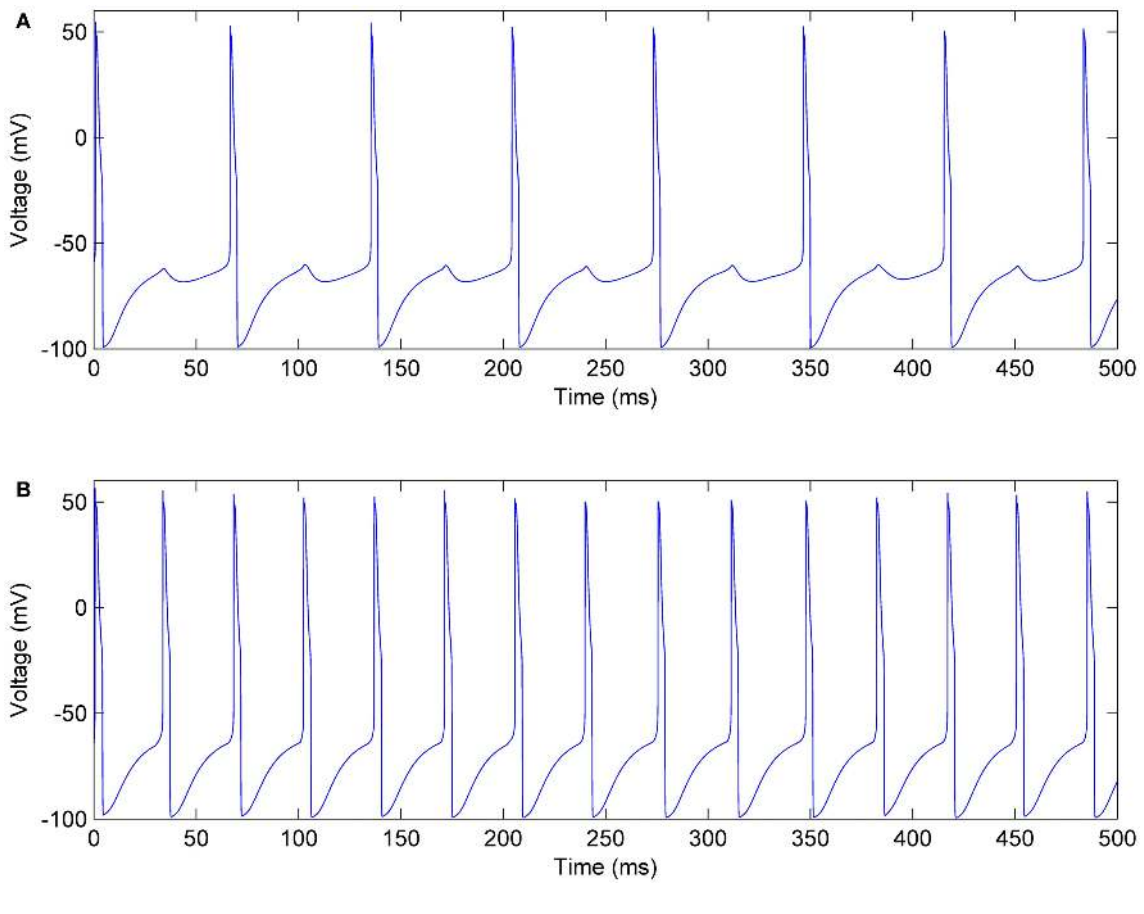


**FIGURE 12 | Membrane voltage of one of the neurons, (A) In the excitatory pool, (B) In the inhibitory pool of a mini-column.**

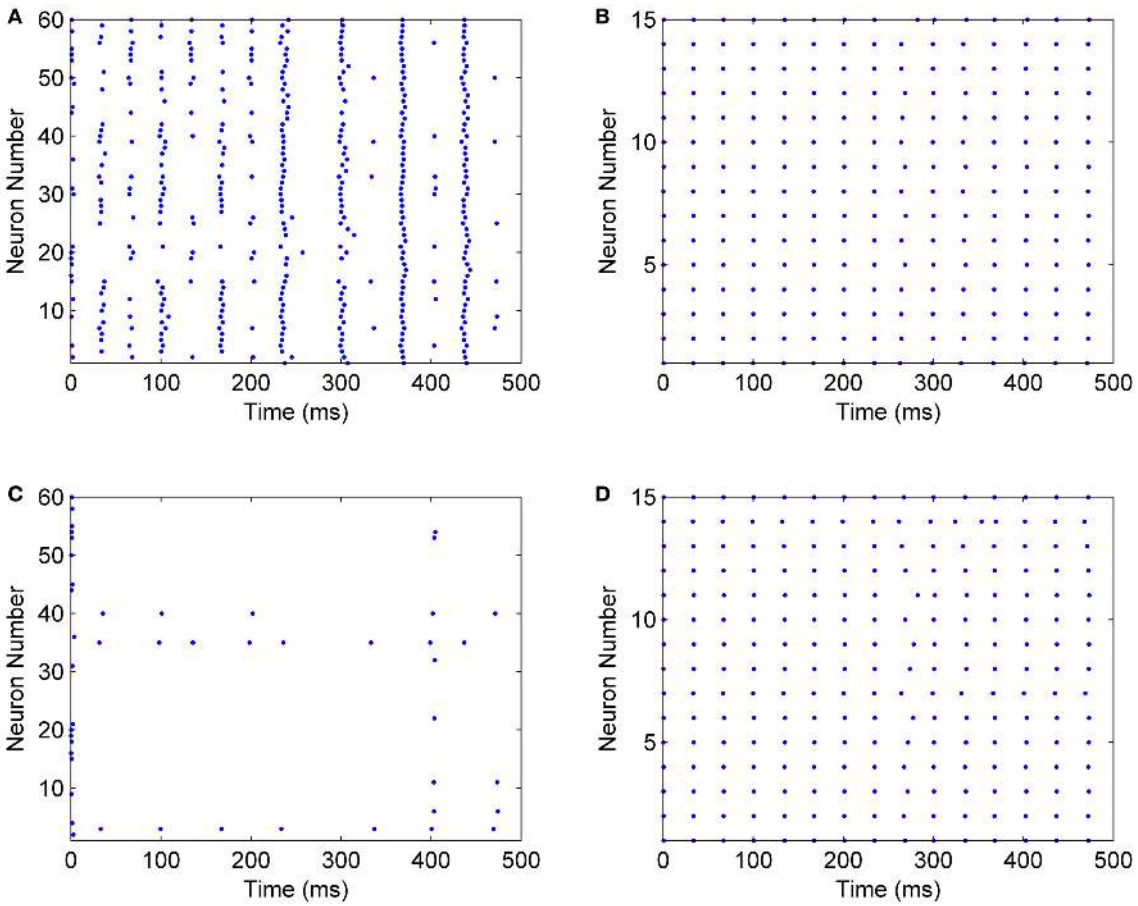


**FIGURE 13 | Rastergrams of the implemented neural network, for (A) The excitatory pool in mini-column 2, (B) The inhibitory pool in mini-column 2, (C) The excitatory pool in mini-column 1, (D) The inhibitory pool in mini-column 1.**

LUT-based implementation in different parts of the model and some equations with the exponential terms, which has a ıside effect on the final accuracy of implementation. The authors of Zhang et al. (2009) implemented a 32-bit floating point reconfigurable somatic neuroprocessor on an FPGA. In Saïghi et al. (2010a,b); Grassia et al. (2011) the H-H based neural network has been implemented on a combination of systems: digital hardware, analog hardware, and software. Due to data transmission between different interfaces, these kinds of implementations are more sensitive to noise and have lower speed compared to the network implemented on a single FPGA. Also, implementation, validation, and evaluation of these types of systems are harder than pure digital design on FPGA devises. In Bonabi et al. (2012b), a neural pool is implemented on FPGA. In this paper, we have upgraded this pool to a network composed of four different pools. It is done by improving our implementation method through using computational techniques, such as CORDIC algorithm and step-by-step integration in the implementation of arithmetic circuits, in addition to sharing resources. Our implementation makes increasing the network size possible while keeping the network execution speed close to real time and having high precision.

## CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a method to implement a biologically plausible neural network on an FPGA. The network consists of four neural pools; two excitatory; and two inhibitory pools. Each pool is implemented based on the H-H neuron model.

We used MATLAB simulation for validating our implementation. After simulating the network using MATLAB, we chose suitable number of bits for each variable in mathematical computations, in order to have less error in the results of the implemented network.

We described each part of the H-H model equations using VHDL as a hardware description language (Bottom-up approach). In order to have a neural network with the maximum number of neurons and accurate responses, we focused on having small error, rapid processing time, and using less hardware resources. In the implementation, both of the full implementation and LUT-based implementation were used to obtain high accuracy and low execution time. In addition, varieties of computational circuits were tested to have an implementation with optimal resource usage, such as CORDIC algorithm and step-by-step integrator, which are beneficial in having accurate response and low hardware resource usage. At every stages of the

implementation, range of the inputs and outputs are examined, in order to achieve the appropriate data representation and to have simplified arithmetic functions. For example, in the implementation of different modules, the LUT implementation is used instead of complete functional implementation, where the function has limited range of changes in the values of inputs and outputs. We achieved high-speed performance of running the neural network, based on the parallel processing nature of FPGA, which is practically impossible in sequential platforms. Moreover, our design has high accuracy in its output; i.e., in membrane voltage signals and firing rates.

The presented implementation technique has other benefits as well, such as scalability and extendibility. The number of neurons could be increased by increasing the number of repetitions in the address generator. It results in linear increase in response time of the system. The implemented network could be used as a pipeline system to raise its speed in the next steps of this research. Using our system for developing controllers for cognitive robots is among our research plans.

## REFERENCES


Ambroise, M., Levi, T., Joucla, S., Yvert, B., and Saïghi, S. (2013). Real-time biomimetic central pattern generators in an FPGA for hybrid experiments. *Front. Neurosci.* 7:215. doi: 10.3389/fnins.2013.00215

Bailey, J. A., Wilcock, R., Wilson, P. R., and Chad, J. E. (2011). Behavioral simulation and synthesis of biological neuron systems using synthesizable VHDL. *Neurocomputing* 74, 2392–2406. doi: 10.1016/j.neucom.2011.04.001

Bakhtiari, R., Sephavand, N. M., Ahmadabadi, M. N., Araabi, B. N., and Esteky, H. (2012). Computational model of excitatory/inhibitory ratio imbalance role in attention deficit disorders. *J. Comput. Neurosci.* 33, 389–404. doi: 10.1007/s10827-012-0391-y

Bonabi, S. Y., Asgharian, H., Bakhtiari, R., Safari, S., and Ahmadabadi, M. N. (2012a). "FPGA implementation of Hodgkin-Huxley neuron model," in *IJCCI*, eds A. C. Rosa, A. D. Correia, K. Madani, J. Filipe, and J. Kacprzyk (SciTePress), 522–528.

Bonabi, S. Y., Asgharian, H., Bakhtiari, R., Safari, S., and Ahmadabadi, M. N. (2012b). "FPGA implementation of a cortical network based on the Hodgkin-Huxley neuron model," in *Neural Information Processing*, eds T. Huang, Z. Zeng, C. Li, and C. S. Leung (Berlin; Heidelberg: Springer), 243–250.

Börgers, C., Epstein, S., and Kopell, N. J. (2005). Background gamma rhythmicity and attention in cortical local circuits: a computational study. *Proc. Natl. Acad. Sci. U.S.A.* 102, 7002–7007. doi: 10.1073/pnas.0502366102

Buzsáki, G. (2004). Large-scale recording of neuronal ensembles. *Nat. Neurosci.* 7, 446–451. doi: 10.1038/nn1233

Cassidy, A., and Andreou, A. G. (2008). "Dynamical digital silicon neurons," in *IEEE Biomedical Circuits and Systems Conference, 2008. BioCAS 2008* (Baltimore, MD), 289–292. doi: 10.1109/BIOCAS.2008.4696931

Çavuşlu, M. A., Karakuzu, C., Şahin, S., and Yakut, M. (2011). Neural network training based on FPGA with floating point number format and it's performance. *Neural Comput. Appl.* 20, 195–202. doi: 10.1007/s00521-010-0423-3

Chen, K., and Wang, D. (2002). A dynamically coupled neural oscillator network for image segmentation. *Neural Netw.* 15, 423–439. doi: 10.1016/S0893-6080(02)00028-X

Ercegovac, M. D., and Lang, T. (2003). *Digital Arithmetic.* San Francisco, CA: Elsevier.

Ermentrout, G. B., and Kopell, N. (1998). Fine structure of neural spiking and synchronization in the presence of conduction delays. *Proc. Natl. Acad. Sci. U.S.A.* 95, 1259–1264. doi: 10.1073/pnas.95.3.1259

Gatet, L., Tap-Béteille, H., and Bony, F. (2009). Comparison between analog and digital neural network implementations for range-finding applications. *IEEE Trans. Neural Netw.* 20, 460–470. doi: 10.1109/TNN.2008.2009120

Graas, E. L., Brown, E. A., and Lee, R. H. (2004). An FPGA-based approach to high-speed simulation of conductance-based neuron models. *Neuroinformatics* 2, 417–435. doi: 10.1385/NI:2:4:417

Grassia, F., Lévi, T., Tomas, J., Renaud, S., and Saïghi, S. (2011). "A neuromimetic spiking neural network for simulating cortical circuits," in *45th Annual Conference on Information Sciences and Systems (CISS), 2011* (Baltimore, MD), 1–6. doi: 10.1109/CISS.2011.5766098

Heo, Y., and Song, H. (2012). Circuit modeling and implementation of a biological neuron using a negative resistor for neuron chip. *BioChip J.* 6, 17–24. doi: 10.1007/s13206-012-6103-x

Hodgkin, A. L., and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* 117, 500.

Indiveri, G., Linares-Barranco, B., Hamilton, T., VanSchaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Izhikevich, E. M. (2007). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. Cambridge, MA: The MIT press.

Kandel, E. R., Schwartz, J. H., and Jessell, T. M. (2000). *Principles of Neural Science. 4th Edn.* New York, NY: McGraw-Hill Medical.

Khan, F. A., Uppal, M., Song, W. C., Kang, M. J., and Mirza, A. M. (2006). FPGA Implementation of a neural network for character recognition. *Adv. Neural Netw.* 2006, 1357–1365. doi: 10.1007/11760191_197

Mokhtar, M., Halliday, D. M., and Tyrrell, A. M. (2008). Hippocampus-inspired spiking neural network on FPGA. *Evol. Syst.* 5216, 362–371. doi: 10.1007/978-3-540-85857-7_32

Moldakarimov, S., Rollenhagen, J. E., Olson, C. R., and Chow, C. C. (2005). Competitive dynamics in cortical responses to visual stimuli. *J. Neurophysiol.* 94, 3388–3396. doi: 10.1152/jn.00159.2005

Muthuramalingam, A., Himavathi, S., and Srinivasan, E. (2008). Neural network implementation using FPGA: issues and application. *Int. J. Inform. Technol.* 4, 86–92.

Pourhaj, P., Teng, D. Y., Wahid, K., and Ko, S. B. (2010). "A novel scalable parallel architecture for biological neural simulations," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)* (Paris), 3152–3155. doi: 10.1109/ISCAS.2010.5537951

Rice, K. L., Bhuiyan, M. A., Taha, T. M., Vutsinas, C. N., and Smith, M. C. (2009). "FPGA implementation of izhikevich spiking neural networks for character recognition," in *International Conference on Reconfigurable Computing and FPGAs, 2009. ReConFig'09* (Quintana Roo), 451–456. doi: 10.1109/ReConFig.2009.77

Sahin, S., Becerikli, Y., and Yazici, S. (2006). "Neural network implementation in hardware using FPGAs," in *Neural Information Processing*, eds I. King, J. Wang, L.-W. Chan, and D. Wang (Berlin; Heidelberg: Springer), 1105–1112.

Saïghi, S., Levi, T., Belhadj, B., Malot, O., and Tomas, J. (2010a). "Hardware system for biologically realistic, plastic, and real-time spiking neural network simulations," in *The 2010 International Joint Conference on Neural Networks (IJCNN)* (Barcelona), 1–7. doi: 10.1109/IJCNN.2010.5596979

Saïghi, S., Tomas, J., Bornat, Y., Belhadj, B., Malot, O., and Renaud, S. (2010b). "Real-time multi-board architecture for analog spiking neural networks," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)* (Paris), 1939–1942. doi: 10.1109/ISCAS.2010.5538039

Traub, R. D., and Miles, R. (1991). *Neuronal Networks of the Hippocampus.* Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511895401

Wang, R. M., Hamilton, T. J., Tapson, J. C., and van Schaik, A. (2014). A mixed-signal implementation of a polychronous spiking neural network with delay adaptation. *Front. Neurosci.* 8:51. doi: 10.3389/fnins.2014.00051

Wanhammar, L. (1999). *DSP Integrated Circuits.* Academic Press.

Yang, S., Wu, Q., and Li, R. (2011). A case for spiking neural network simulation based on configurable multiple-FPGA systems. *Cogn. Neurodyn.* 5, 301–309. doi: 10.1007/s11571-011-9170-0

Zhang, Y., Nunez-Yanez, J., McGeehan, J., Regan, E., and Kelly, S. (2009). "A Biophysically accurate floating point somatic neuroprocessor," in *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009* (Prague), 26–31. doi: 10.1109/FPL.2009.5272558

Zou, Q., Bornat, Y., Tomas, J., Renaud, S., and Destexhe, A. (2006). Real-time simulations of networks of Hodgkin-Huxley neurons using analog circuits. *Neurocomputing* 69, 1137–1140. doi: 10.1016/j.neucom.2005.12.061

**Conflict of Interest Statement:** The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

*Received: 17 July 2014; accepted: 05 November 2014; published online: 21 November 2014.*
*Citation: Yaghini Bonabi S, Asgharian H, Safari S and Nili Ahmadabadi M (2014) FPGA implementation of a biological neural network based on the Hodgkin-Huxley neuron model. Front. Neurosci. **8**:379. doi: 10.3389/fnins.2014.00379*

*This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience.*