Digital Hardware Implementation of Bus-based, Field-Ordered Hopfield Network using VHDL

Kuek Chian Shiun, Wan Ahmad Tajuddin Wan Abdullah and Burhanuddin Kamaluddin
Computational Physics Laboratory, Physics Department, Science Faculty, University of Malaya, 50603 Kuala Lumpur.
Email: chianshiun@yahoo.co.uk

Abstract This paper presents some new approach in designing hardware discrete Hopfield network. For the network architecture, the bus-based approach is used to largely reduce the number of connection count. Field-ordered updating algorithm is used in placed of the typical random-ordered updating due to its added advantage in finding global minimum. As for the implementation, VLSI language has been used due to its ease in describing and documenting complex digital design. Besides, modern CAD tools enables the synthesis of VHDL codes into layout plans on customized FPGA chips and making corresponding timing simulation.

I. INTRODUCTION

VLSI hardware implementation has been an elegant solution for the neural networks due to it’s ability to enable parallel processing of information, speeding up processing time by a ratio of 100 to 1000 times as compared to software solution. Among several optional technologies, digital solution is found to be most robust especially if the network is large [1]. Meanwhile, adding number of neurons in a network will increases its processing ability. This can means the need to implement the network in multiple chips. Yet, implementing a large-scale, densely-connected network, such as the Hopfield network, becomes a major problem. The author estimates that the network size for VLSI implementation of Discrete Hopfield Neural Network is limited to about 1000++ neurons with current technology.

However, Hopfield network as an advantage: One of the unique characteristics of Hopfield net is that only one neuron is allowed to “fire” a signal (being updated) at any one time [2]. Having such characteristic means though there’s direct connection between each neuron, only 1 set of output connections will be utilized at any one time. We can exploit this by merging the connections into 1 common data-bus, thus greatly saving resources spent on connections without compensating on the processing speed.

Wan A. Tajuddin has done an extensive study on various possible functional algorithm of Hopfield network [3]. In his search, the field-ordered updating algorithm (founded by him) is found to be superior among the various known methods in increasing the network chance to converge to the global minimum. Besides, it gives one of the fastest convergence in terms of average-updates per-neuron, thus reducing processing speed.

In addition, a typical Hopfield network requires the neurons to be updated in random order. This means an additional random number generator is needed to generate random update sequence for each update cycles (if a good quality random sequence is desired). Field-ordered algorithm likewise eliminates the needs of this random generator unit.

CS Kuek has further developed these ideas by designing a 16-neuron digital system using VHDL. Binary discrete Hopfield Network is used. The design is being compiled using Xilinx ISE 5.2i CAD tool [4] and timing simulation based on Spartan-IIE 2S300E FPGA chip produced by the same cooperation. The synaptic weights are obtained using Hebb Rule [5].

II. DESIGN

The fundamental element of artificial neural network – neuron – is very much based on the architecture of biological neuron (Fig. 1)[6]. Signal pulses from other neurons are transmitted into the cell via dendrites and the net intensity is summed at soma. If it exceeds a certain threshold, the cell body fires a signal down the axon to other neurons via synapse. The effect upon other neuron varies depending the enzyme release by the synapse, which can prohibit or inhibit the next neuron to fires a signal. This mechanism is commonly referred as synaptic
It is possible to design a neuron circuit that mimics such operation. McCulloch-Pitts [7] has conceptualized it as the way shown in Fig. 2. The input signals are define as $x_n$, synaptic weight as $w_n$ and threshold as $\theta$. Output $y_i$ fires a signal if the dot product of input and weight equal or exceeds threshold.

A typical method of implementing such structure digitally is as shown in Fig. 3. The synaptic weight is loaded into register $w_{ij}$ and sum $\sum x_i w_{ij}$ is reset to zero prior to operation. Then, each neuron takes turn to transmit output into neuron-j via input $x_i$. At the same time, co-responding tri-state buffer turns from ‘high impedance’ to ‘on’ to allow signal propagation from multiplexer into the adder.

Consider that we need to implement a 6-neuron system; we can utilize the architecture as shown in Fig. 4. We conclude that the number of chip pin-outs required would be $n$, where $n$ is the number of neurons (assuming control circuits are not a problem). Likewise, current industrial world holds the manufacturing limit to about 1000++ [8]. While this value is fairly large for current standard, demands for network with much greater size is imminent in the future. A better solution is required to break-through this limit.

Fortunately, if we reconsider Hopfield network’s operational algorithm, we find a hint of hope: When the network operates, only one neuron is allowed to fires an update at any one time. If all neurons fires simultaneously, it could cause the network to iterate into an infinite loop or oscillation. If we analyzed Fig. 4, we would find that only one pin-out will carries a signal at any one time. We can exploit this unique character by merging the output connection into one common bus. The source of signals likewise would be identified with a corresponding index bus (receiver neurons receive the output on the bus along with the sender neuron’s address). This greatly reduced the number of required chip pin-outs to $(\log_2 n) + 1$. Fig. 5 summarized the
resulting architecture (index bus not included).

To achieve this, we need to alter the neuron structure shown earlier. Fig. 6 shows a possible solution. The major alteration is the synaptic weights are now stored in a RAM. After the system has been initialized, neurons will take turns to fires a signal (either ‘1’ or ‘0’) via common bus $x_i$ together with its unique index. The receiving neuron will capture the index and input into the address input of the RAM. Corresponding weight will be output to the multiplexer and the rest of the operation will be the same as structure in Fig. 3. Note that the circuit is now consumes less resources, much less connections and yet the processing speed is not compensated. For chips with 1000 pin-outs, we can now produce network size of $2^{1000}$ neurons!

To gain global minimum advantage, we further modify the neuron to accommodate for field-ordered operational algorithm. The essence of field-ordered algorithm is this: Instead of having a random updating sequence, the sequence is now governed by a factor call local field, $h$, where $h$ is defined as $h = \sum x_i w_{ij} - \theta$. Field-ordered algorithm says that among all neurons with imminent change (output to be transmitted is different from the former time-step value), the neuron with the highest magnitude of local field, $|h|$, will be selected to update its new output to the rest. If there is more than one neuron with the same highest magnitude of local field, an arbitrary selection is made among them [3]. Fig. 7 shows a possible implementation of field-ordered neuron.

There are two major modification happens here: First, we found that a subtractor can effectively replace the role of comparator. The MSB of $\sum x_i w_{ij} - \theta$ will produce the invert value of $y_j$ while producing the local field, $h$. The MSB value is compared against the previous output value, $\overline{y_1}$, using a XOR gate. If it is different, change is imminent and ‘change-status’ output turns into ‘1’ and the neuron is set to compete for updating opportunity. Note that if change-status = ‘0’ or if magnitude of local-field is not the greatest in the network, the value of previous output is retained. Here, $y_j$ is designated as $\overline{y_2}$, the potential output of the neuron that is yet to be updated.

We also noted that since only the neuron with highest local-field is allowed to update its value, we can sum up the total net-input, $\sum x_i w_{ij}$, at the beginning of the network operation. Then, in subsequent updates, we only add upon or minus out the corresponding weight value of the selected neuron. We see that $\overline{y_2}$ fit in the job nicely to produce the decision of whether to add-on or minus out corresponding weight value. Second modification involves changing the previous adder into adder/subtractor unit.

Field-ordered Hopfield network has also the great advantage in reducing processing time in 2 areas. To illustrate the first area, we study Fig. 8. Fig. 8 shows a typical updating flow-chart of 4-neuron Hopfield network. To simplify the discussion, we label the various network states with alphabets.

Now, consider that the network is being updated using original Hopfield algorithm and the neurons are labeled as $N_1$, $N_2$, $N_3$ and $N_4$. During the first iteration cycle, assume that we have the random update sequence of $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4$ and the initial pattern is $M(0011)$. We shall get the following computation:
Since at least a change of state is observed, test of convergence fails and the net needs to run a second iteration cycle. In the second iteration cycle, the update sequence is arbitral; either way, the net does not change state except to prove that the net has converged. As we observed, eight time-steps has been spent to have pattern M(0011) converged to B(1010). Now, compare with the Field-Ordered algorithm, we get the following computation:

<table>
<thead>
<tr>
<th>Time-Step</th>
<th>Update Index</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N3</td>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>2</td>
<td>N1</td>
<td>0011</td>
<td>1011</td>
</tr>
<tr>
<td>3</td>
<td>N4</td>
<td>1011</td>
<td>1010</td>
</tr>
<tr>
<td>4</td>
<td>N2</td>
<td>1010</td>
<td>1010</td>
</tr>
</tbody>
</table>

As we can see, only two time-steps were spent in this example, we save 75% of computational time. This is due to easier procedure that omits steps that doesn’t contribute to the effective change of network. Empirical result shows a discount of about 71.4% in processing time [3].

Second way computational time is saved is this: Considered neuron in Fig. 3. For every new updates, we need n-1 time-steps to compute the net-input, $\sum x_i w_{ij}$. The structure in Fig. 7 only needs 1 time-step.

### III. RESULTS AND DISCUSSION

A 16-neuron field-ordered Hopfield net has been converted into VHDL modules and supporting control circuits are added to sort for desired neuron for updating. The design passed compilation, functionality and timing simulation based on Spartan-IIIE 2S300E FPGA chip. Time-step used was 100ns and typical pattern recognition problem is solved within 50µs. Time-step length is possible to be further reduced by half considering longest delay-time but is not practice to cater for a greater margin for error. Delay-time is possible to be further reduced by using better design such as fast adders, using better FPGA chips such as Virtex series [4] or even custom-made chips. Total system size is about 16,000 gates.

![Typical updating sequence of a 4-neuron Hopfield network.](image)

### IV. CONCLUSION

We have presented some methods of constructing digital hardware Hopfield network that will greatly extend the limits of networks size, reduce resources spend and connection count through simpler architecture and much faster operational system while harvesting better results through increased chance of finding global minimum.

### V. ACKNOWLEDGEMENT

To God be the glory!

### REFERENCES


