Journal of Engineering and Technology Research Vol. 4(1), pp. 11-21, January 2012
Available online at http:// www.academicjournals.org/JETR
DOI: 10.5897/JETR11.077
ISSN 2006-9790 ©2012 Academic Journals

Full Length Research Paper

Digital very-large-scale integration (VLSI) Hopfield neural network implementation on field programmable gate arrays (FPGA) for solving constraint satisfaction problems

A. Srinivasulu

Vaagdevi Institute of Technology and Science, Proddatur, India. E-mail: sreesri.avvaru@gmail.com.

Accepted 28 November, 2011

This paper discusses the implementation of Hopfield neural networks for solving constraint satisfaction problems using field programmable gate arrays (FPGAs). It discusses techniques for formulating such problems as discrete neural networks, and then it describes the N-Queen problem using this formulation. Finally results will be presented which compare the computation times for the custom computer against the simulation of the Hopfield network run on a high end workstation. In this way, the speed-up can be determined, that illustrate a speedup of up to 2 to 3 orders of magnitude is possible using current FPGAs devices.

Key words: Hopfield neural network, field programmable gate arrays (FPGA), N-Queen problem.

INTRODUCTION

Many practical optimization problems from business and industry can be formulated as standard mathematical programming problems using binary decision variables. Solution of these problems requires the use of heuristics or approximate algorithms due to the NP {(Neuro Psychology)-hard nature of their complexity; Fred and Gary, 1989}. Neural networks were proposed to solve such problems in 1985 (David, 1998), but the field has been plagued with problems of poor solution quality and inability to guarantee feasible final solutions (Hopfield and Tank, 1985). These initial problems have now been overcome. Techniques have been proposed to help the Hopfield neural network escape from local minima of its energy function and suitable construction of that energy function has been shown to guarantee the feasibility of solutions (Silvio, 1992). Using these improvements, neural network results have been obtained which compete effectively (and even outperform) other popular heuristics such as simulated annealing.

While most of the literature has focused on using Hopfield networks to solve the famous traveling salesman problem, a range of practical problems have also been solved with neural networks (Dijin, 1995; Silvio, 1992). The solutions to these problems were obtained by simulating the behavior of the Hopfield neural network (designed to be implemented in electrical hardware) on a conventional computer. However, while the algorithms generate good solutions, the computation times are extremely slow. If neural networks are to be applied routinely to practical problems, then the execution time must be reduced. There are a number of ways of accelerating the execution of the network algorithms, ranging from the use of high end parallel supercomputers, through to hardware implementations of the networks themselves using custom computing machines (CCMs). CCMs are attractive, because they have the potential to provide cheap high speed platforms for neural network based algorithms. However, until recently the cost of producing specific hardware has been high and the process errorprone.

Recently, the advent of high density field programmable gate arrays (FPGAs), in combination with new synthesis tools, have made it relatively easy to produce programmable custom machines without building specific hardware. FPGA based CCMs can provide high performance on certain problems, demonstrating speedups of orders of magnitude over conventional machines (David, 1998; Chen and du

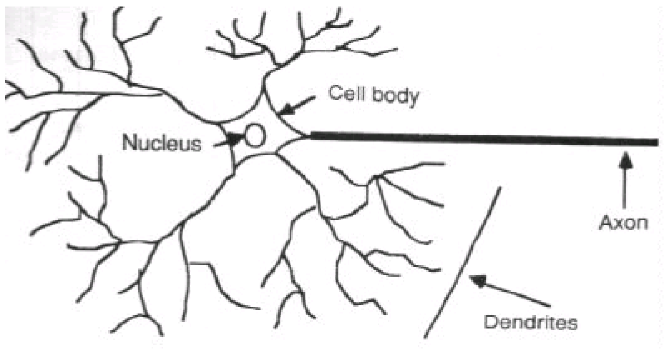


Figure 1. Components of a neuron.

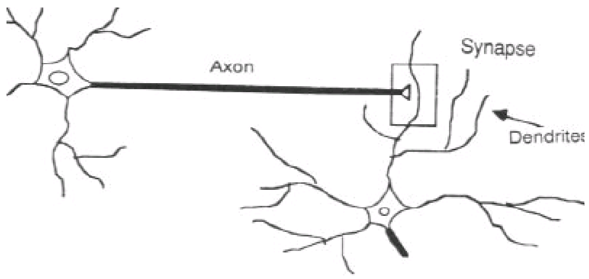


Figure 2. The synapse.

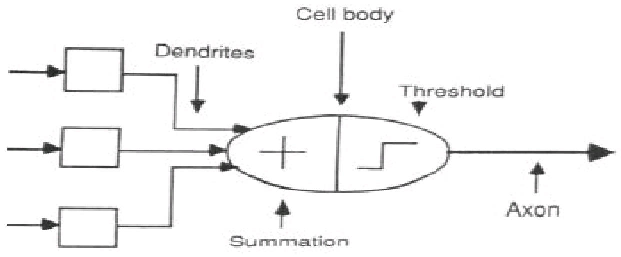


Figure 3. The neuron model.

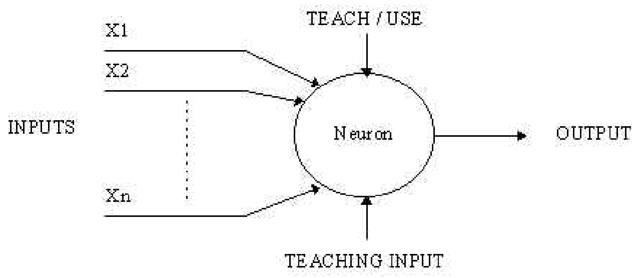


Figure 4. A simple neuron.

Plessis, 2002). There is great potential to apply these techniques to neural network based algorithms, however, research must be conducted to determine the appropriate methods.

This paper aims to demonstrate the potential of a custom computer based on FPGA technology for solving a classical constraint satisfaction problem: the N-Queen problem. The Hopfield neural network will be briefly described, and we will show how the N-Queen problem can be mapped onto the architecture. The issues involved in designing the custom computer will be discussed. Finally results will be presented which compare the computation times for the custom computer against the simulation of the Hopfield network run on a high endworkstation. In this way, the speed-up can be determined.

HOPFIELD NEURAL NETWORKS

Hopfield neural networks are a biologically inspired mathematical tool which can be used to solve difficult optimization problems.

Artificial neurons

In the human brain, a typical neuron collects signals from others through a host of fine structures called dendrites shown in Figure 1. The neuron sends out spikes of electrical activity through a long, thin stand known as an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse (shown in Figure 2) converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected neurons. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

We conduct these neural networks by first trying to deduce the essential features of neurons and their interconnections shown in Figure 3. We then typically program a computer to simulate these features. However because our knowledge of neurons is incomplete and our computing power is limited, our models are necessarily gross idealizations of real networks of neurons (Limin, 2003).

A simple neuron

An artificial neuron is a device with many inputs and one output shown in Figure 4. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not (Hopfield and Tank, 1985).

Neural networks in business

1) Marketing
2) Credit evaluation

Hopfield nets

Mainly the Hopfield nets are used as autoassociators. In addition to serving as auto associators, Hopfield networks can be applied to optimization and constraint satisfaction problems (David, 1998; Hopfield and Tank, 1985). The idea is to encode each hypothesis as a unit and to encode constraints between hypotheses by weights. Positive weights encode mutual supporting relationships, where as negative weights encode incompatible relationships. As the Hopfieldnet settles into a stable state, the state reflects the assignment of truth and falsity to the various hypotheses under constraints.

So, the Hopfield nets are useful both for auto association and for optimization tasks. It applies the concept of energy surface minimization in physics to finding stable solutions in the neural networks. Also, this network is relatively easy to implement in VLSI chips (Chen and du Plessis, 2002).

Architecture

The main concept underlying the Hopfield network is that a single network of interconnected, binary-valued neurons can store multiple stable states. Suppose we create a network of binary-valued neurons, where each neuron is connected to the others but not back to it. Assume all the connection weights are symmetric. That is$T_{jk}=T_{kj}$. This network can have a set of stable states. For each stable state, each binary neuron takes on a value (either 0 or 1) so that when it acts on its neighbors, the values of each neuron do not change.

The architecture of the Hopfield net is shown in Figure 5. The number of the network units is the same as that of the bits or values contained in each pattern (David, 1998). Units update their states asynchronously or sychronously by receiving inputs from other units. Once set, the weights in the Hopfield net are not trainable (Hopfield and Tank, 1985).

Hopfield net algorithm for optimization

Weight assignments

1) Write energy function based on problem constraints.

2) Compare the above energy function with the following energy function of the Hopfield net (a Liapunov function) to determine the weights

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij} v_i v_j - \sum_{i=1}^{N} I_i v_i$$

Calculation of activation

1) at time t=0, $v_j(0)$ = a randomized small value. Where $v_j(t)$ is the activation level of unit j at time t.
2) At time t(t>0),

$$u_i(t+1) = u_i(t) + \Delta t\left(\sum_{j=1}^{N} T_{ij} v_j + I_i\right)$$

$v_i(t+1) = g(u_i)$

Where the function g (u_i) is a hard-limiting function

$$v_i = g(u_i) = \begin{cases} 1, & u_i > 0 \\ 0, & u_i \le 0 \end{cases}$$

3) Repeat step 2 until equilibrium (that is, the activation levels of nodes remainunchanged with further iterations). Then, the pattern of activations upon equilibrium represents the optimized solution.

Improving solution quality

Many variations of the Hopfield network have been proposed for improving the solution quality. These approaches can be broadly categorized as either deterministic or stochastic. There have also been developments in hardware implementation that have enabled local minima to be avoided and problem-specific theoretical work on basins of attraction that enable the initial states of the network leading to good quality solutions to be calculated (Yegnanarayana, 2004). The deterministic approaches include problem-specific enhancements such as the “divide and conquer” method for solving the TSP, deterministic hill-climbing such as the “rock and roll” perturbation method and the use of alternative neuron models within the Hopfield network such as the winner-take-all neurons used to improve the feasibility of the solutions (Yegnanarayana, 2004). Stochastic approaches address the problem of poor solution quality by attempting to escape from local

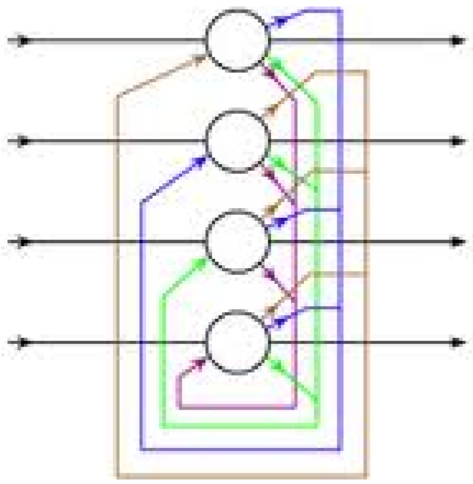


Figure 5. Architecture of Hopfield network.

minima. There are basically four main methods found in the literature to embed stochastic (Yegnanarayana, 2004) into the Hopfield network:

1) Replace sigmoid activation function with a stochastic decision-type activation function,
2) Add noise to the weights of the network,
3) Add noise to the external inputs (biases) of the network, and
4) Any combination of the above methods.

The Boltzmann machine utilizes the first method based on a discrete Hopfield model (Yegnanarayana, 2004). The inputs are fixed, but the discrete activation function is modified to become probabilistic. Much like simulated annealing, the consequence of modifying the binary activation level of each neuron is evaluated according to the criteria of the Boltzmann probability factor. This model is able to escape from local minima, but suffers from extremely large computation times (Yegnanarayana, 2004).

In order to improve the efficiency and speed of the Boltzmann machine, Akiyama et al. (1989) proposed Gaussian machines that combine features of continuous Hopfield networks and the Boltzmann machine (Yegnanarayana, 2004). Gaussian machines have continuous outputs with a deterministic activation function like the Hopfield network, but random noise is added to the external input (bias) of each neuron. This noise is normally distributed (or Gaussian) with a mean of zero and a variance controlled by a temperature parameter. However, based upon fast simulated annealing, which uses Cauchy noise to generate new search states and requires only at/log(*t*) cooling schedule, the Cauchy machine was proposed as an improvement to solution quality (Yegnanarayana, 2004). The Cauchy distribution is said to yield a better chance of convergence to the global minimum than the Gaussian distribution (Yegnanarayana, 2004).

Furthermore, Cauchy noise produces both local randomwalks and larger random leaps in solution space, whereas Gaussian noise produces only local random walks. The noise is incorporated into the activation function, while the outputs of the Cauchy machine are binary. In the high-gain limit of the gradient of the stochastic activationfunction, the Cauchy machine approaches the behavior of the discrete (and deterministic) Hopfield network. Anotherstochastic approach that has been very successful is mean fieldannealing, so named because the model computes the mean activation levels of the stochastic binary Boltzmann machine. Often, however, stochastic neural networks designed to“kick” a solution out of a local minimum suffer from instability.In previous work, we have suggested a modification to the internal dynamics of the modified Hopfield network (with the feasibility guaranteed)that permits escape from local minima through hill climbing (Hopfield and Tank, 1985; (Yegnanarayana, 2004).

Clearly, there are many approaches to improving thesolution quality of the Hopfield network through escape from local minima of the energy function and embedding stochastic into the dynamics of the network. The valid subspace approach has resulted in a guarantee of feasibility as well. Thus, the initial problems that have plagued the reputation of the Hopfield network have now been resolved.

Many optimization problems can be readily represented on Hopfield nets, by transforming the problem into variables such that the desired optimization corresponds to the minimization of the respective Lyapunov function (Hopfield and Tank, 1985). In this representation, the dynamics of change in network state with time takes the system to a local energy minimum. If this local minimum is also the global minimum, the solution of the desired optimization task has been carried out by the convergence of the network state (Yegnanarayana, 2004). Indeed, the energy function can be thought of as a programming language for transforming optimization problems into a solution method applying network dynamics. The resulting network could be either built in analog hardware or implemented in software on a digital machine.

Linear programming, the worker assignment problem (Dijin, (1995), and decomposing signals into a basis set can all be solved exactly by Hopfield networks because the Lyapunov function for these problems can be constructed with a single (and thus global) minimum. When more computationally difficult problems are programmed using this approach, the Lyapunov function often has multiple local minima, and the dynamics of the network may converge to a local minimum rather than to the global minimum. Finding a good half-tone image from a gray-scale photograph and the n-queens chess problem can be programmed in this way (Limin, 2003). How effective such a network can be in finding a good solution is strongly dependent on the problem class.

Biological modeling of the human brain is attempted by utilizing a fully inter-connected system of N neurons. Neuron i has internal state u_i and output level v_i (which can be either binary valued in the discrete model or real valued bounded by 0 and 1 in the continuous model). The internal state u_i incorporates a bias current (or negative threshold) denoted by I_i, and the weighted sums of outputs from all other neurons. The weights, which determine the strength of the connections from neuron i to j, are given by T_{ij}. The relationship between the internal state of a neuron and its output level is determined by an activation function $g(u_i)$. The nature of this activation function depends on whether the Hopfield network is discrete or continuous.

Commonly,

$$v_i = g(u_i) = \frac{1}{2}\left(1 + \tanh\left(\frac{u_i}{\tau}\right)\right) \qquad (1)$$

Is used for the continuous model, where τ is a parameter used to control the slope (or gain) of the activation function.

For discrete Hopfield networks, the activation function is usually a discrete threshold function:

$$v_i = g(u_i) = \begin{cases} 1, & u_i > 0 \\ 0, & u_i \le 0 \end{cases} \qquad (2)$$

The neurons update themselves (either sequentially orin parallel) according to the following rule:

$$u_i(t+1) = u_i(t) + \Delta t\left(\sum_{j=1}^{N} T_{ij} v_j + I_i\right) \qquad (3)$$

$$v_i(t+1) = g(u_i)$$

and in doing so, the network of neurons will convergeto a local minimum of the following energy function overtime:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij} v_i v_j - \sum_{i=1}^{N} I_i v_i \qquad (4)$$

Provided the weights are symmetric $T_{ij} = T_{ji}$.

If neurons are updated in parallel (or synchronously) then the possibility of convergence to a two-cycle exists.Both of the network states which comprise the two-cyclewill be local minima of the energy function however.

The discrete model has an advantage over the continuousmodel in terms of the number of updates required toconverge to a local minimum. For this reason, and others related to hardware constraints will be discussed, we have chosen to use a discreteHopfield network for solving the N-Queen problem. We have also chosen to update the neurons in a parallel operation rather than sequentially since it is our ultimate intention to solve large scale problems as rapidly aspossible. Parallel implementation involves calculating allof the U updates then all of the v updates, as opposed tothe sequential update which calculate the u and v updatefor each neuron one at a time.

Hopfield and Tank (1985) showed that if a 0-1 optimization problem can be expressed in terms of anenergy function of the form given by (4), then a Hopfield network can be used to find locally optimal solutions of the energy function. This may translate to local minimum solutions of the optimization problem.

Typically, the network energy function is made equivalent to the objective function of the optimization problem, while the constraints of the problem are included in the energy function as penalty terms. The network parameters can then be inferred by comparison with the standard energy function given by (4). The weights of the network, T_{ij} are then the coefficients of the quadratic term, $v_i v_j$ and the external bias currents,I_i, for each neuron i, are the coefficients of the linearterms v_i, in the chosen energy function. The network can be initialized by setting the activity level v_i, of each neuron to an unbiased state. Updating the network according to equation (3) will then allow a minimum energy state to be attained, since the energy level never increases during state transitions.

However, these stable states may not necessarily correspond to feasible or good solutions of the optimization problem, and this is one of the major pitfalls of the H-T formulation. Because the energy function comprises several terms (each of which is competing to be minimized), there are many local minima, and a tradeoff exists between which terms will be minimized. An infeasible solution to the problem will arise when at least one of the constraint penalty terms is non-zero. If this occurs, the objective function term is generally quite small, because it has been minimized to the detriment of the constraint terms, thus the solution is “good” but not feasible. Alternatively, all constraints may be satisfied, but a local minimum may be encountered that does not globally minimize the objective function, in which case the solution is feasible but not “good.” Certainly, a penalty parameter can be increased to force its associated term to be minimized, but this generally causes other terms to be increased. The solution to this trade-off problem is to find the optimal values of the penalty parameters that balance the terms of the energy function and ensure that each term is minimized with equal priority. Only then will the constraintterms be zero (a feasible solution), and the objective function be also minimized (a “good” solution). The derivation of the weights and external biases for the N-Queen problem are provided subsequently.

$$W_{ij} = \begin{matrix}
-2 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & -1 \\
-1 & -2 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 \\
-1 & -1 & -2 & -1 & 0 & -1 & -1 & -1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\
-1 & -1 & -1 & -2 & 0 & 0 & -1 & -1 & 0 & -1 & 0 & -1 & -1 & 0 & 0 & -1 \\
-1 & -1 & 0 & 0 & -2 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & 0 & -1 & 0 \\
-1 & -1 & -1 & 0 & -1 & -2 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & -1 & -1 & -1 & -2 & -1 & 0 & -1 & -1 & -1 & -1 & 0 & -1 & 0 \\
0 & 0 & -1 & -1 & -1 & -1 & -1 & -2 & 0 & 0 & -1 & -1 & 0 & -1 & 0 & -1 \\
-1 & 0 & -1 & 0 & -1 & -1 & 0 & 0 & -2 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
0 & -1 & 0 & -1 & -1 & -1 & -1 & 0 & -1 & -2 & -1 & -1 & -1 & -1 & -1 & 0 \\
-1 & 0 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -2 & -1 & 0 & -1 & -1 & -1 \\
0 & -1 & 0 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -2 & 0 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & -1 & 0 & -1 & 0 & -1 & -1 & 0 & 0 & -2 & -1 & -1 & -1 \\
0 & -1 & 0 & 0 & 0 & -1 & 0 & -1 & -1 & -1 & -1 & 0 & -1 & -2 & -1 & -1 \\
0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -2 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -2
\end{matrix}$$

Figure 6. Weight matrix.

N-QUEEN PROBLEM

The N-Queen problem is a classical constraint satisfaction problem, whose goal is to place N Queens on an NxN chess board in mutually non-attacking positions. Since a Queen can only attack horizontally, verticallyand diagonally, the constraints can be stated as:

*there can be only one Queen in each row
* there can be only one Queen in each column
* there can be only one Queen in each diagonal (ascending and descending).

Suppose we define a binary decision variable:

X_{ij}= 1 if a Queen is positioned in row i and 0 otherwise

There are several ways of combining all of theseconstraints in an objective function as penalty terms, andone such function is:

$$f(x) = \frac{A}{2}\sum_{i=1}^{N}\left(\sum_{j=1}^{N} x_{ij} - 1\right)^2 + \frac{B}{2}\sum_{j=1}^{N}\left(\sum_{i=1}^{N} \mathrm{x_{ij}} - 1\right)^2 + \frac{C}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{p\neq 0\\ 1\leq i+p\leq N\\ 1\leq j-p\leq N}} x_{ij}x_{i+p,j-p} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{p\neq 0\\ 1\leq i+p\leq N\\ 1\leq j+p\leq N}} x_{ij}x_{i+p,j+p}\right) \tag{5}$$

The first two terms ensure that the rows and columns sum to one, while the final term counts the cost of Queens on each diagonal (ascending and descending). The value of this function isexactly zero for a non-attacking solution, and will be greater if some of the constraints are not satisfied. The values of A**,** B and C are selected to balance the relative importance of each constraint, and we have fixed these tounity.

Expanding and rearranging this objective function so that it is expressed as the sum of a linear component and quadratic components yield:

$$f(x) = \frac{A}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N}\delta_{ik}x_{ij}x_{kl} - A\sum_{i=1}^{N}\sum_{j=1}^{N}x_{ij} + \frac{B}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N}\delta_{jl}x_{ij}x_{kl} - B\sum_{i=1}^{N}\sum_{j=1}^{N}x_{ij} + \frac{C}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N}x_{ij}x_{kl}(1-\delta_{ik})\left(\delta_{i+j,k+l}+\delta_{i-j,k-l}\right)\right)$$

$$= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N}\left(-\mathrm{A}\delta_{ik} - \mathrm{B}\delta_{jl} - \mathrm{C}(1-\delta_{ik})\left(\delta_{i+j,k+l}+\delta_{i-j,k-l}\right)\right)x_{ij}x_{kl} - \sum_{i=1}^{N}\sum_{j=1}^{N}(\mathrm{A}+\mathrm{B}) + constants \tag{6}$$

Where δ_{ij}, is the Kronecker-delta symbol equivalent to unity only if i =j and is zero otherwise. Comparing this expansion to the standard Hopfield network energy function (4), and noting the double subscript for this N Queen formulation, the weights and external biases can be read off as:

$$W_{ij,kl} = -\mathrm{A}\delta_{ik} - \mathrm{B}\delta_{jl} - \mathrm{C}(1-\delta_{ik})\left(\delta_{i+j,k+l}+\delta_{i-j,k-l}\right)$$

And I_{ij} = A+B (7)

In this project we are designing Hopfield network for solving 4 – Queen problem. So, the weight matrix order becomes 16 × 16 (as N increases the weight matrix order becomes $N^2 \times N^2$) and the matrix shows the derived weight matrix from equation (7) (Figure 6).

(Where i, j, k, l are from 1 to 4),

And the bias current, from Equation (7) is,

I_{ij} = 1 + 1 = 2.

Since the parameters A, B, Cand D are taken as unity. Therefore

I_{ij} = 2.

Where the negative sign indicates incompatibility (inhibition or weak) relationship between the neurons. According to the derived weights the connections are made. Once the weights are set, the adaptation of weights does not exist. Because we are implementing a Hopfield network, the interconnection between neurons is fixed and dictated by the nature of the constraints in the problem. This means that it can be wired into the implementation, which is particularly relevant for FPGA based machines. This not only reduces the amount of interconnection hardware required, but also reduces the time spent accumulating the input values to a neuron (Chen and du Plessis, 2002).

After designing the weights, the initial state of each neuron is selected randomly, and then the weighted sum of each neuron is calculated. Since we are designing the digital network, the step function is chosen as the nonlinear activation function to decide the state of each neuron whether it is fired or not. By applying this weights and bias to the network we can get the solution to our (4 – Queen) problem. But this may not be the optimal solution to the problem. That means we may get the local minima value, which is the disadvantage in the Hopfield neural network. To get the global or optimal solution to the problem, we have to eliminate the local minima. The techniques to overcome the local minima problem are already mentioned. So, by using one of those techniques we can eliminate local minima problem. In this project we are just designing the Hopfield network for solving 4-Queen problem and implementing on FPGA.

IMPLEMENTATION USING FPGAS

Architecture

There has been much architecture proposed for the implementation of neural networks over the years, including both digital and analogue circuits (Chen and du Plessis, 2002; Silvio, 1992). Most of these have concentrated on a hardware implementation of the neuron evaluation function, which consists of computing number ofmatrix-vector products. Thus, these systems involve the parallel and pipelined executions of a number of multiply operations together with a reduction sum operator. A small amount of work has been focused on the training aspects of networks, which can be extremely time consuming.

The work described in this paper differs from general neural networks in three important ways. The weights are small and can be represented using smallintegers. This means that the hardware responsible for the accumulation can be optimized for small integer values. This not only reduces the size of the arithmeticunits, but also reduces the carry propagation delays.

1) The neuron values are restricted to 0 or 1, rather than general fixed or floating point numbers.Consequently, the vector product becomes a set of conditional additions without the need to perform any multiplication operations. Multiplier units normally consume large amounts of logic, thus the savings hereare dramatic.
2) Because we are implementing a Hopfield network, the interconnection between neurons is fixed and dictated by the nature of the constraints in the problem. This means that it can be wired into the implementation, which is particularly relevant for FPGA based machines. This not only reduces the amount of interconnection hardware required, but also reduces the time spent accumulating the input values to a neuron.

The architecture chosen for this work consists of set of neurons, as described by the VHDL code, which are then interconnected when the weights are non-zero. Thus, neurons which have zero weights between them are not connected, limiting the number of inputs to a neuron to O(n) instead of O (N^2), where N is the number of neurons in the system.

Whilst the code for each neuron is identical, each one is specified with a different Weight-array, and thus the hardware generated for each will be slightly different.

Implementation

The consequence of the issues discussed in the earlier on means that it is possible to implement the neural network using FPGA devices. FPGAs, like the Xilinx family of parts, consist of a number of configurable logic blocks (CLBs) connected using a hierarchical, bus based, and wiring scheme. The neurons and their interconnections are specified in VHDL, which is synthesized and simulated using Xilinx ISE simulator and Spartan 3E tools for FPGA mapping and routing. Our hardware platform, an FPGA Spartan 3E reconfigurable logic board, routes the neuron outputs either to an array of LEDs or back to a host work station for display. Since all the weights are known at synthesis time, the synthesis tool's optimization features are exploited to automatically remove any additions involving zero-valued weights. The use *of* VHDL has a number of advantages over conventional hardware design techniques like schematic capture.

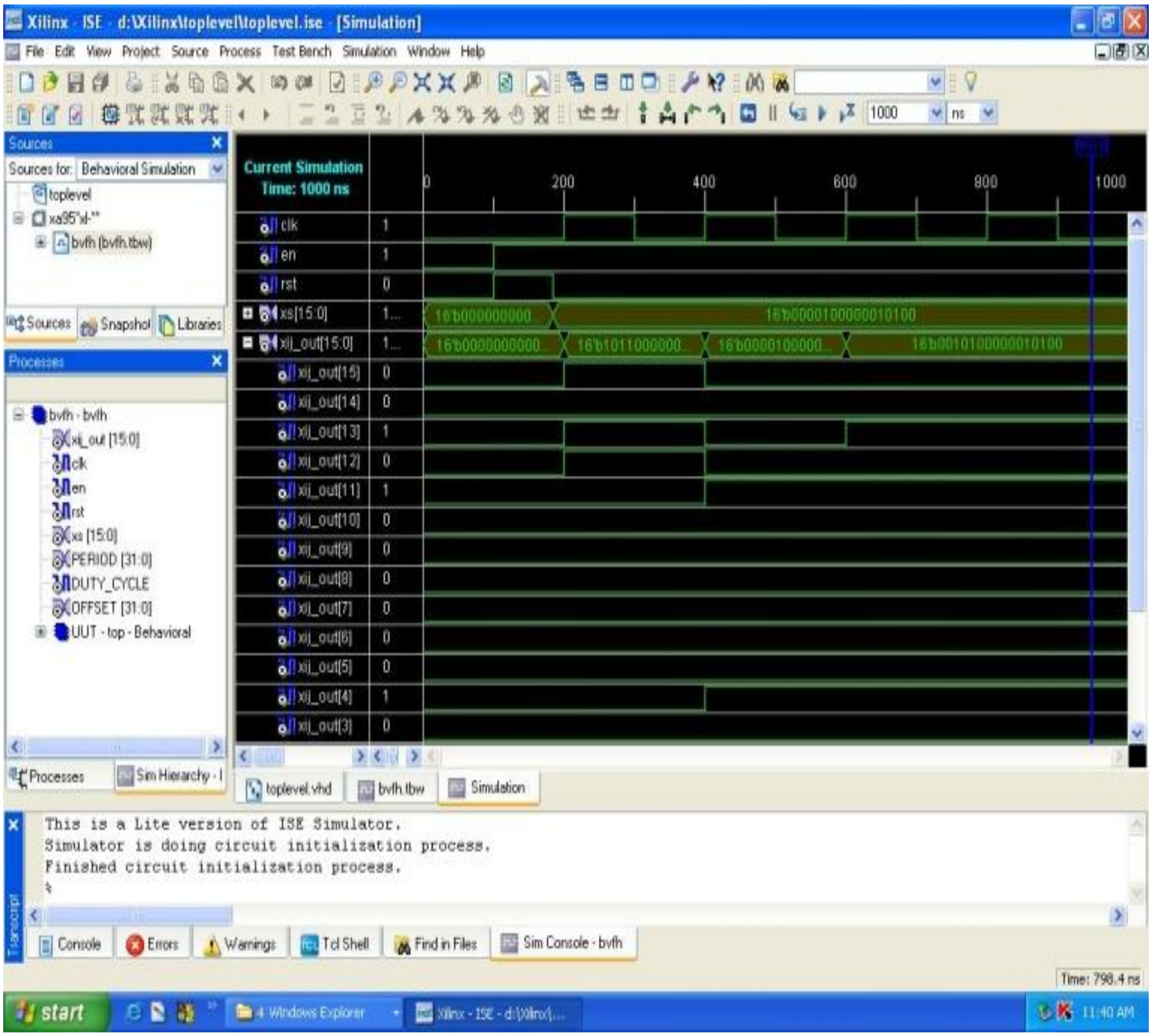
1) Its high level syntax is not very different from conventional imperative programming languages, thus the design effort is not significantly different from writing a software simulation of a neural network. This is an important design consideration when considering the development cost of application specific hardware.
2) The VHDL software supports extensive optimizations, thus the performance of the underlying hardware is optimized for each neuron depending on the weights on its inputs.
3) The VHDL compiler analyses the domain of the variables and generates optimal hardware without any further user interaction. For example, if the range of a variable is limited to the values from0 to 3, then the data path used to transmit and manipulate the variable will be automatically constrained to 2 bits.

PERFORMANCE RESULTS

We have performed experiments using 4, 8, and 16 - Queen’s problem. This code resembles the structure of the hardware solution. Using Xilinx 9.2i software the results are simulated on ISE simulator and implanted on FPGA Spartan 3E, xc 3s500e devices. The simulation result and the synthesis report is given as Figure 7 and the synthesis report is given as Figure 8.

Timing summary

Speed Grade: -4



```
This is a Lite version of ISE Simulator.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

Figure 7. Simulation results

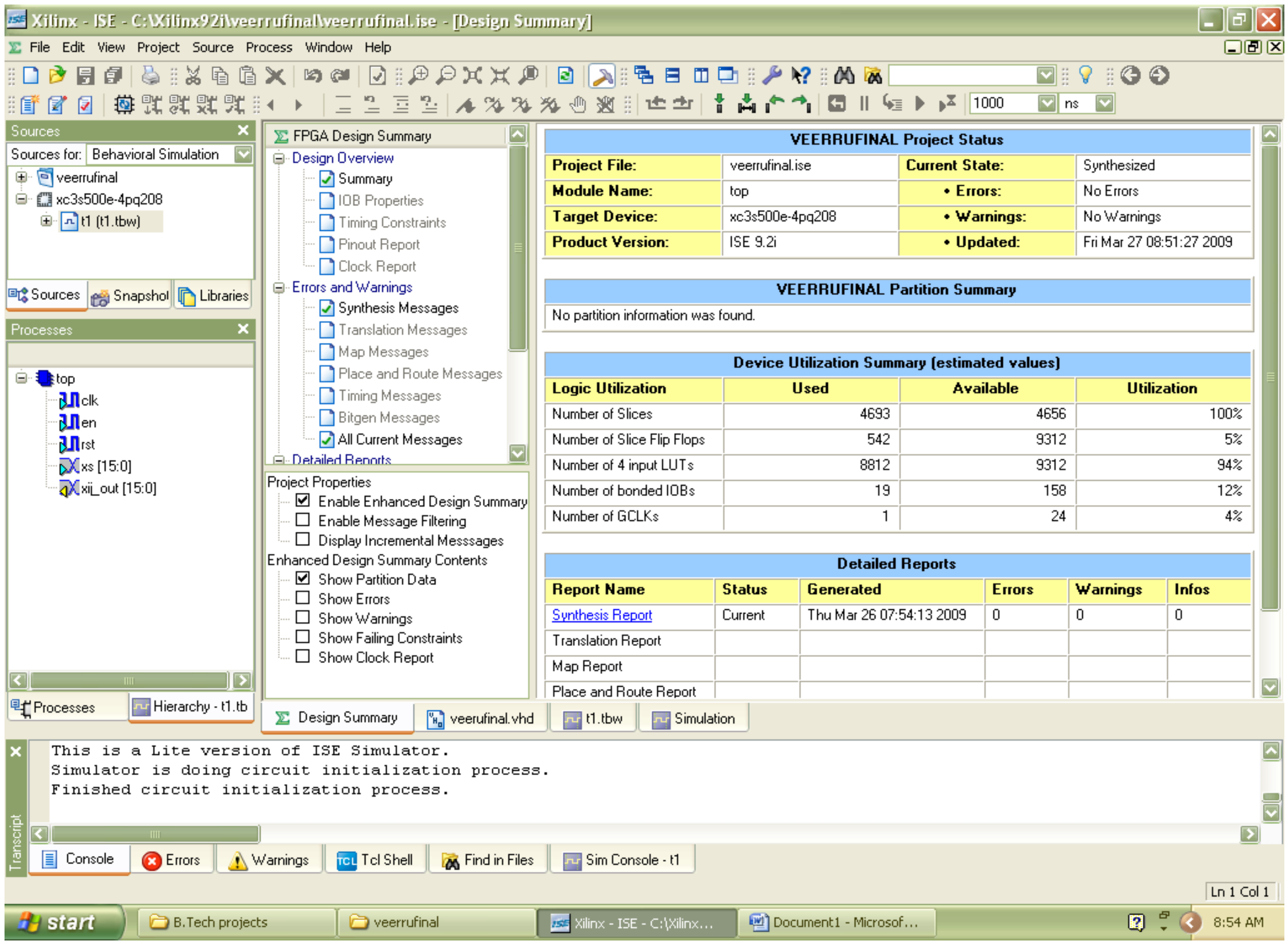
Minimum period: 64.978 ns
(Maximum Frequency: 15.390 MHz)
Minimum input arrival time before clock: 5.153 ns
Maximum output required time after clock: 8.180 ns
Maximum combinational path delay: No path found

Device utilization summary

Selected device: 3s500epq208-4
Number of slices: 4693 out of 4656; 100% (*)
Number of slice flip flops: 542 out of 9312; 5%
Number of 4 input LUTs: 8812 out of 9312; 94%
Number of IOs: 35
Number of bonded IOBs: 19 out of 158; 12%
Number of GCLKs: 1 out of 24; 4%

HDL synthesis report

Macro Statistics
# Adders/Subtractors: 256
32-bit adder: 256
# Accumulator: 16



VEERRUFINAL Project Status			
Project File:	veerrufinal.ise	**Current State:**	Synthesized
Module Name:	top	**• Errors:**	No Errors
Target Device:	xc3s500e-4pq208	**• Warnings:**	No Warnings
Product Version:	ISE 9.2i	**• Updated:**	Fri Mar 27 08:51:27 2009

VEERRUFINAL Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	4693	4656	100%
Number of Slice Flip Flops	542	9312	5%
Number of 4 input LUTs	8812	9312	94%
Number of bonded IOBs	19	158	12%
Number of GCLKs	1	24	4%

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Thu Mar 26 07:54:13 2009	0	0	0
Translation Report					
Map Report					
Place and Route Report					

```
This is a Lite version of ISE Simulator.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

Figure 8. Synthesis report.

32-bit up accumulator: 16
# Registers: 1
16-bit register: 1
# Comparators: 16
32-bit comparator greater: 16

Advanced HDL synthesis

Advanced HDL synthesis report
Macro statistics
# Adders/sub tractors: 256
32-bit adder: 256
# Accumulators: 16
32-bit up accumulator: 16
# Registers: 16
Flip-flops: 16
# Comparators: 16
32-bit comparator greater: 16

Figures 9 and 10 shows the computational time and speed up of the hardware over each of the software approaches written in VHDL and C code.

The results indicate that it is possible to gain between 2 and 3 orders of magnitude speed up, which is significant. The computation time and speed of the network depend on the number of iterations and N in N-Queen problem. As either number of iterations or N increases, the computation time increases and speedup decreases.

Conclusions

The aim of this work was to establish whether it was possible to achieve a reasonable speed up by implementing FPGA based Hopfield neural networks for some simple constraint satisfaction problems. The results are significant our initial implementation using standard

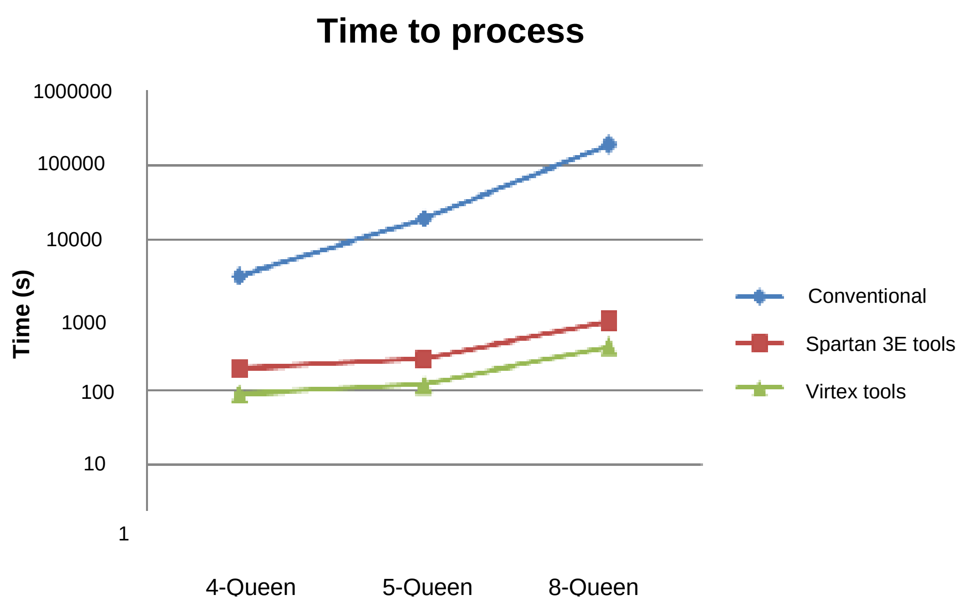


Figure 9. System performance based on computational time.

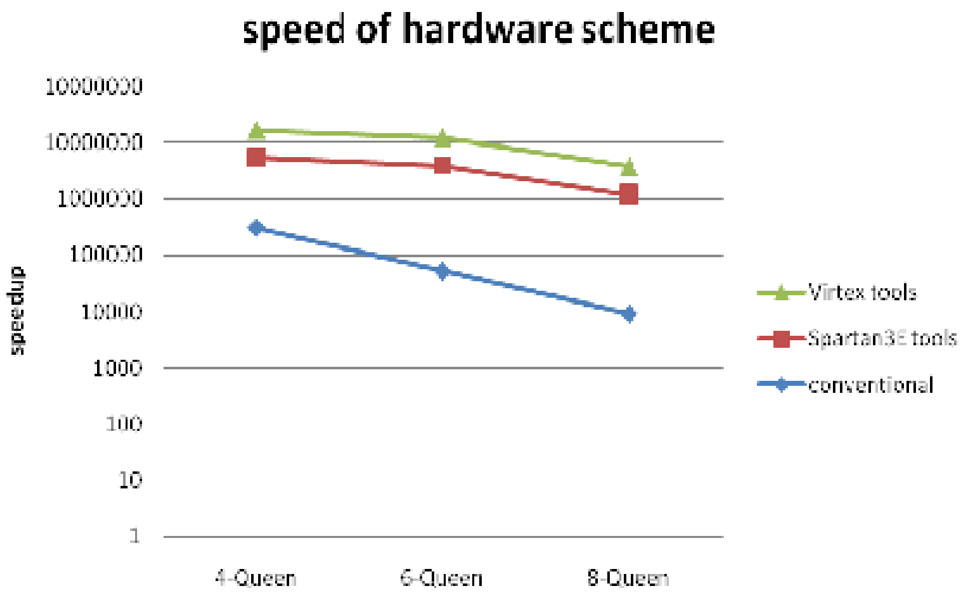


Figure 10. System performance based on speed.

XilinxFPGAs yielded 2 to 3 orders of magnitude speed up over the Sun Blade 2000 workstation comes with 1.2-GHz version of the 64-bit UltraSPARC III Cu processor.

The main problem with the work to date is that the problems are both unrealistically small and simplistic. That is the constraint on the N-Queen problemare simpler than those found in many real world scheduling applications. Thus, it is not clear whether we will be able to optimize the neuron structure for more complex problems since the weights matrix may not contain as many zero elements.

REFERENCES

Chen YJ, du Plessis WP (2002). “Neural network implementation on FPGA.” IEEE, pp. 337-342.

David A (1998). “FPGA based implementation of a Hopfield neural network for constraint satisfaction Problems.” IEEE, pp. 688-692.

Dijin G (1995). ”Neural network Approach for General Assignment Problem.” IEEE, 4: 1861 - 1866

Fred G, Gary AK (1989). “ Handbook of Mataheuristics.” By Kluwer International Series.

Hopfield JJ, Tank D (1985). “Neural computation of decisions in optimization problems.” “Artificial neural networks“, Yegnanarayana. Biol. Cybernet., 52: 141-152.

Limin F (2003). “Neural networks in computer intelligence.” By McGraw-Hill International Editions. Computer Science Series.

Silvio PE (1992).”Analog VLSI Neural networks: Implementation issues and examples in Optimization and Supervised learning.” IEEE, 39(6): 552 – 564.