A Cellular Automata FPGA Architecture that can be Trained with Neural Networks

J. Lyke¹

Air Force Research Laboratory
Space Vehicles Directorate
3550 Aberdeen Ave SE
Kirtland AFB, NM 87117-5776
(505)846-5812
james.lyke@kirtland.af.mil

Abstract—It is possible to derive a simple FPGA architecture from 1-D cellular automata structures in which a 2-D spatial feed-forward network is formed. By permitting each site to take on any possible function in its input space (through LUT substitution) an interesting new Boolean network concept is produced. It can be viewed as an FPGA, and it can be refined in a number of ways to accommodate the addition of configuration circuitry and registration structures. The interesting features of this FPGA include its low descriptive complexity/high regularity, low interconnect demand, interchangeability of logic/routing resources, and defect tolerance. By exploiting a connection between the Vapnik-Chervonenkis dimension of (at least) low-order LUTs and perceptron neural networks, it is relatively straightforward to model these Boolean networks with equivalent artificial neural networks, which can be trained using traditional approaches, such as the back-propagation algorithm. This paper reviews the derivation of this architecture and demonstrates examples of evolved circuit designs.

TABLE OF CONTENTS

1. INTRODUCTION
2. CELLULAR AUTOMATA FPGA
3. LOOK-UP TABLES (LUTs) AND THE FORM OF APPROXIMATING NEURAL NETWORKS
4. DESIGNING TILED LUT NETWORKS WITH ANNs
5. RESULTS / DISCUSSION
6. CONCLUSIONS

1. INTRODUCTION

Improved techniques for the automated design of integrated circuits (ICs) are eagerly sought, particularly for advanced digital standard cell and field programmable gate arrays (FPGAs). FPGAs, which are customized by software after fabrication, are of particular interest. A sequence of steps are involved in very large scale integration (VLSI) design, most of which are NP-complete [1], and hence heuristics are required to obtain adequate, albeit not usually optimal, solutions. The most common techniques in FPGA design employ standard methods in computer science, whether it is dynamic programming for logic decomposition or perhaps

greedy techniques with simulated annealing for routing [1]. The heuristics are often confined to a particular part of the design process, such as routing, which, for example, presupposes the completion of the technology mapping and sometimes placement calculations. In some cases, the choices made in the prior calculations might lead to intractability in a subsequent step. As such, it is often necessary to backtrack not just for new solutions within a particular heuristic, but sometimes back to the beginning of the whole design sequence to find a solution. In some classes of FPGA architectures, logic and routing resources are interchangeable, and the decisions made in placement further constrain routability.

Non-traditional approaches have been applied to these problems as well. Recent work has been described involving evolutionary approaches (e.g., genetic algorithms), in which the configuration of an FPGA is altered in-circuit to construct tonal discriminators [2]. Limited work has been presented on other potential design approaches, such as neural networks.

The intent of this paper is to describe a technique involving artificial neural networks (ANN) to design FPGA circuits using a standard back-propagation network. This work has been applied to a special FPGA architecture, described in [3] but resummarized briefly to emphasize the characteristics that make it particularly attractive for emulation by ANNs. In examining FPGA emulation, it is important to understand the expressive capacity of both the FPGA and ANN. This paper examines recent results on the Vapnik-Chervonenkis (VC) dimension to arrive at minimal neural subnetworks, each of which are capable of modeling a single lookup table (LUT) in the original FPGA. A simple procedure is then described for: (1) setting up a complete neural network that is capable of completely modeling a section of the related FPGA; (2) training the neural network with the desired behavioral specification, which must be expressed as a truth table; and (3) recovering FPGA designs from convergent ANNs. Results of simple examples are described as a largely empirical exercise, and the limitations and restrictions of this technique are further discussed.

¹ U.S. Government work not protected by U.S. copyright.

2. CELLULAR AUTOMATA FPGA

Rather than specifying metallization patterns or locations for individual transistors, FPGA designs use software to select or configure a complex array of prefabricated digital circuit resources. In this process, a user's design is entered with design tools using methods similar to that employed in standard integrated circuits. The completion of this process results in a configuration bitstream, which specifies the behavior of the FPGA when it is transferred into the FPGA electrically.

Figure 1. (a) Portion of FPGA, illustrating two look-up tables (LUTs) and some associated routing resources. Small hollow (filled) circles are programmable (fixed) connections. (b) Example routing for F31=AB; F41=CD, simplified view. (c) Same routing showing actual routing resources consumed to form terminal connections (from [3]).

In FPGA architectures, the LUT is one of two essential structures for implementing complex digital designs. Since it is not possible to implement large LUTs (say 1000-input) in physical devices, it is necessary to use a large number of smaller LUTs (say, 3- to 5-input) to implement complex designs. The process of redefining an arbitrary Boolean description in terms of k -input LUTs is referred to as *logic decomposition*. *Technology mapping* refers to the process of mapping this design to specific LUTs in the FPGA. The other important FPGA structure pertains to the interconnection between all LUTs in an architecture. *Routing* refers to the process of determining how specified input/output termini relationships between LUTs are satisfied.

We will illustrate the use FPGAs to realize designs with an example. Figure 1 illustrates a typical FPGA implementation of two Boolean functions. First, the general "fabric" is shown (Figure 1a), containing two, four-input LUTs and a variety of routing resources. Hollow circles indicate potential connections, which are implemented by a programmable semiconductor switch. A viable (non-unique) solution to a two-function design (F31=AB; F41=CD) is shown in Figure 1b, but the routing is drawn similar to the way a designer might think about the problem. In Figure 1c, the true routing situation is illustrated in terms of resources used, albeit with a greater loss in clarity.

FPGAs are heavily dominated by interconnection resources

and are highly irregular, making it difficult to apply non-traditional heuristics in their design. A new architecture, the *cellular automata* (CA) FPGA, is derived from CA structures, as it implements a periodic architecture with local-only interconnectivity. CAs, in particular binary CAs, are discretized points in a regular m -dimensional lattice, whose values evolve at discrete time points [4]. The values depend only on nearest neighbors, making CAs a convenient abstraction for modeling a localized network of nodes involving usually identical Boolean functions. In CA-based FPGAs (CAFPGAs), architectures are formed by

directly replacing each discrete point in a CA lattice space where a computation would be done by a LUT with an arity equal to the neighborhood of the corresponding CA.

To illustrate these CAs, Figure 2a illustrates the template of a 1-D CA with 3-neighborhood and its repetition to form an infinite network. Typical CA structures have a feedback characteristic. Feedback behavior, which is responsible for the rich behavior of CAs (Figure 2b), is difficult to analyze in traditional VLSI design. For this reason most computer-aided design is based on synchronous circuitry, in which all feedback is synchronized with a global clock (analogous to recurrent neural networks [7]). While the power of feedback remains a subject of rich *future* exploration, the present work emphasizes only the case of feed-forward behavior.

One way to eliminate feedback behavior in the Figure 2a template involves exploiting a second spatial dimension. Specifically, rather than feed backwards (onto itself) the results of the computational cycle, it is possible to break the links and feed *forward* those results onto another copy of the 1-D structure. This operation is reflected schematically in Figure 3a. The iteration of n sites this template in m rows forms a 2-D feed-forward network (Figure 3b), where each row corresponds to the behavior of an n -site CA at the i th time step. When a LUT is placed at each site, the $m \times n$ structure represents a tile of a cellular automata FPGA (CAFPGA).

Unlike traditional CA structures, each site in a CAFPGA architecture can be programmed with a different function. Since the sites are each realized with a LUT, all Boolean functions of the same arity can be implemented. While any

Figure 2. 1-D binary cellular automata. (a) Template of single 1-D CA site (neighborhood 3) and its overlapping tessellation to form an infinite 1-D CA structure. (b) Time evolution "strip chart" of multi-site CA structure is shown. Each site possesses an identical function (parity) at each site, with a single non-zero initial condition (time increasing in the downward direction) (c) Expanded version of sample, displaying fractal behavior in the parity function. The results may also be interpreted as a count of all downward paths from the initial seed to the corresponding point in time-space, modulo 2 (from [4]).

site can implement an "elemental" Boolean function, more complex Boolean functions (with an input space larger than the arity of a single function) can be implemented by decomposing the function in terms of simpler Boolean functions, each of which can "fit" onto single sites, and arranging the Boolean functions onto the "lattice" of available sites. This process, equivalent to the technology mapping step in standard FPGAs, is significantly constrained by the connective limitations of the CAFPGA structure.

CAFPGA architectures are perhaps the simplest possible specification for an FPGA architecture. The most striking difference between CAFPGA and traditional architectures is

that the CAFPGA does not employ distinct routing resources. In traditional FPGAs, it is not uncommon to find 90% of the IC "real estate" to be dedicated to routing resources [5]. In the CAFPGA, which has no dedicated routing resources, wires must be implemented "virtually," i.e., through logic. By programming LUTs with the truth tables that correspond to a wire, the corresponding LUTs are "sacrificed" to realize routing. As such, a 3-input LUT can "wire" an output virtually to any of its three inputs by simply configuring the LUT to repeat the value desired input as the output function.

Having introduced the fundamental concepts of standard FPGAs and the prospective CAFPGA, it is possible to explore ANN implementations. ANNs approximate complex functions and their ability to do so is best when the expressive capability of the network exceeds the complexity of the function being approximated. Boolean functions represent an interesting possibility for ANNs, particularly if a connection between the two could be exploited for digital design. Though the application of ANNs to Boolean logic has received attention [6] [7], little work on their application to FPGAs has occurred, primarily due to the complex, irregular arrangements of logic and interconnection resources in typical architectures. The CAFPGA may uniquely admit an opportunity for ANN-based design, thanks to its high regularity. To understand how to model even a section of this FPGA, however, it is necessary to develop a better understanding of the LUT itself.

Figure 3. Extension of 1-D CA structure in temporal-spatial CA FPGA. (a) Conversion from 1-D feedback template to feedforward template. (b) FPGA based on tiled LUTs employing feed-forward template.

3. LOOK-UP TABLES (LUTs) AND THE FORM OF APPROXIMATING NEURAL NETWORKS

Boolean combinational circuit networks, expressed as $\{0,1\}^N \rightarrow \{0,1\}^M$, map an N -dimensional input space to an M -dimensional output space. The case in which $M=1$ is a single Boolean function, also referred to as a *dichotomy*.

There are 2^N points in the input space, with 2^{2^N} possible dichotomies of N points or "colorings" of those points to zero or one. LUTs are simply memories with a k -bit address space and (for this discussion) a single output bit. LUTs can be thought of as structures that implement any truth table of a k -bit Boolean function, since a k -bit address space corresponds to 2^k memory locations. Any particular pattern of memory locations defines a single dichotomy. There are obviously 2^{2^k} possible specifications or dichotomies for a k -input LUT (k -LUT).

Figure 4. Neural network model of lookup table. (a) Lookup table (LUT). (b) Form of artificial neural network capable of implementing it.

What is the simplest neural network that can implement a k -LUT? This implies that some minimal network can implement all 2^{2^k} possible dichotomies of a k -LUT, since a LUT is not itself a fixed Boolean function, but rather a structure that can express arbitrary functions of its input space. The Vapnik-Chervonenkis (VC) dimension is a measure of expressive capacity, i.e., the ability of an ANN to implement a range of functions [8]. Carter and Oxeley describe the VC dimension as mapping of the separating capacity [9] of an ANN. In traditional perceptron-based neural networks, it is well-known that a single neuron can only resolve dichotomies that are linearly separable, i.e. those cases where

$$\bar{x}^T \bar{w} > t \Rightarrow \bar{x} \in C_1$$

and

$$\bar{x}^T \bar{w} < t \Rightarrow \bar{x} \in C_2,$$

where $\bar{x}$ and $\bar{w}$ are k -dimensional input and weight vectors, respectively, t is a bias scalar, and C_1, C_2 represent classes. The equation of the separating hyperplane [11] is given by $\bar{x}^T \bar{w} = 0$.

In order to augment the separating capacity of traditional perceptrons, the single-neuron perceptron is augmented with a single hidden layer, where each neuron can contribute a hyperplane in consideration of more complex separation landscapes. Carter and Oxeley recently established a method of evaluating the VC dimension using Poincare polynomials, which establish a count of compartments formed in the intersection of hyperplanes associated with particular neurons. When each layer is fully connected and the hidden layer contains k neurons, it is possible based on these findings [9] to implement all dichotomies of 2^k points. An example, based on a 3-input LUT, is shown in Figure 4. The key difference to this finding and some previous results on NNs relative to Boolean circuit complexity was that in [7], ANNs were restricted in connectivity (2-input), which leads to the standard result that most k -input Boolean functions require an exponential number of 2-input gates to implement [12].

In fact, Koiran has discussed the VC-dimension of circuits [10]. One version of a SELECTION function he defines as having $y + \log_2(y)$ inputs, where $y = 2^k$. The inputs are partitioned into sets Y (the first y inputs) and X (the remaining $\log_2(y)$ inputs). The output of SELECTION is defined as the x th bit of y , where x is any binary setting of each bit in X . If we consider that the inputs of Y are *restricted* or specified by the outputs of a $|Y|$ -bit memory and that only the remaining $|X|$ bits are free, then we have simply redefined the SELECTION function to be a $|X|$ -input look-up table (LUT). Since SELECTION represents a function with VC-dimension $2^{|X|}$, then we recognize similarly that a k -input LUT has a VC-dimension of 2^k , which reinforces the equivalence in VC dimension between the two approaches in Figure 4.

Is the Figure 4b representation truly minimal? It seems that the answer is "yes", unless one considers more sophisticated

Figure 5. A small ($m=4, n=5$) CAFFGA tile. (a) Schematic, depicting local connectivity. (b) Neural network formulation, based on substitution of k -LUTs ($k=3$) by neural subnetworks as described.

activation functions. Gaynier and Downs [13] discuss the fact that non-monotonic functions do have a higher VC-dimension, and they present at least one class of monotonic function with infinite VC-dimension, which employs what might be described as a smooth staircase activation

If the system in Figure 6 can be successfully implemented, then it is likely that a target design could have been realized by the original LUT network. In order to close this loop, it is necessary to "reverse engineer" what patterns are implied by the convergent ANN within the original LUTs of the CAFPGA. To establish this, a simple diagnostic algorithm is applied in situ to each neural subnetwork corresponding to an LUT in the original CAFPGA. This diagnostic simply isolates each neural subnetwork, scans all possible binary patterns into its inputs ("000" through "111"), and records the output values. These output values literally define the pattern of the corresponding LUT in the original CAFPGA. When repeated for all $m \times n$ LUTs, a complete picture of the design for the CAFPGA emerges.

5. RESULTS/DISCUSSION

A neural network simulation incorporating back-propagation capability was developed with an input file structure flexible enough to accommodate arbitrary networks corresponding to different m, n values for emulated CAFPGA tiles. Furthermore, simple auxiliary programs were developed to automatically generate complete neural network input files based on chosen m, n values. This paper only addresses results of a single case, with $m=4$ and $n=5$, corresponding to Figure 5, which has a corresponding neural network consisting of 5 input neurons, 75 hidden layer neurons, and 5 output neurons. Sigmoidal activation functions were employed on all non-input neurons, and a standard back-propagation algorithm was implemented [8].

The conclusions of [9] may generate controversy, as the ability to separate all 2^{2^k} inputs with $(k+1)$ neurons and (k^2+k) weights would imply exponential storage with only quadratic growth in (weight) resources, indeed an exciting result. While we will not pursue this point much further, it is easy to show that discretized representations in weight space cannot yield this result. If a weight is quantized to r bits, then the weight can take on 2^r values. If there are (k^2+k) weights in a perceptron configuration, then the amount of required storage is $(k^2+k)r$, which can be configured $2^{(k^2+k)r}$ possible ways. However, a k -input LUT has a VC-dimension of 2^k and requires 2^k storage. Having a VC-dimension of 2^k implies the ability to configure weights to produce 2^{2^k} distinct dichotomies, which is clearly impossible with (k^2+k) weights, since they can only be set $2^{(k^2+k)r}$ possible ways, and obviously $O((k^2+k)r) < O(2^k)$. Hence, it does not appear that the results of [9] can apply in all cases where weight values are discretized, except below some threshold k_{max} . We cannot make any conclusions for the case of real-valued weights based on these simple counting arguments.

While it is therefore generally not possible to discretely simulate k -input LUTs with perceptrons having k -inputs in a single hidden layer, it may be possible to for small-arity cases. To establish the validity of the form of the Figure 4 network for the *specific* case of $k \leq 3$ using traditional floating-point discretization of the weights, simple empirical investigations were performed on neural subnetworks with 2 and 3 nodes in a single hidden layer to verify convergence on all 255 dichotomies of three-input variables. As expected, the sub-network with only 2 nodes in the hidden layer did not converge on all three-input functions (an example includes the XOR3 function in three inputs), whereas the latter (fully populated) subnetwork did converge in all cases with no difficulties. Therefore, $k_{max} \geq 3$ on this strictly empirical finding.

Next, larger scale LUT networks were simulated using ANNs in which the LUTs were replaced with these specific sub-networks. The ANN system shown in Figure 5 was trained using the Figure 6 approach on a number of simple examples, including routing-only examples, a 2-bit adder and a 2-bit multiplier. By using the diagnostic method described, the LUT specifications were recovered, and the set of all recovered LUT specifications is a Boolean circuit design. To verify that this procedure was correct, the ANN-produced design was hand-simulated to verify its correctness against the truth tables used to produce the design.

Convergence did not always occur, even in cases where a known hand-solution existed. In all cases where convergence did occur, correct designs were produced. A typical result implementing a 2-bit multiplier is shown in Figure 7. In this example, the X3-X2 inputs form the multiplicand, and the X1-X0 inputs the multiplier. The resulting multiplication result is presented as the juxtaposed binary number formed at outputs f_4 down to f_0 . The aforementioned training process was carried out, specifically by presenting 32 training examples, from 00000 to 11111 in binary notation. The unused input (X4) was augmented to the other input variables to form an input space that was larger than that required strictly for the multiplier example. The multiplication truth table from 00000 to 01111 and from 10000 to 11111 have identical form. In training, the truth table entries were randomly shuffled and no entries were omitted, as these initial experiments were not concerned with the possibility of generalization. Indeed, for Boolean designs of common interest, it is not certain what one would expect to see in a generalization process. Therefore, the training set and testing set used were identical.

While these designs are in fact correct, they do not resemble any designs a human would intentionally produce. For example, a grounded signal from the X4 input is unnecessarily OR'd with the X3 input, and some signals are inverted as many as three times, as they are passed from one

Figure 7. Design of two-bit multiplier in a small CAFPGA tile produced by neural networks. Multiplicand inputs are given by X_3, X_2 ; multiplier inputs given by X_1, X_0 .

stage to another. These odd results are due in part to the definition of training, which is to produce *error-free* designs, not necessarily the best or prettiest. But the results are encouraging because, as in the case of greedy algorithms, they do provide answers, which, when bolstered with additional heuristics, can produce better answers. The definition of "better" of course must be considered, since any solution implemented in the same structure will require the same number of LUTs and have the same propagation delay characteristics. It is possible that ANN-based designs could be supplied to other algorithms, which could further compact the design by removing extraneous circuit elements and shifting portions of the implementation through the re-definition of some LUT patterns.

6. CONCLUSIONS

This paper has described a particular form of FPGA with regular structure, amenable to emulation with an equivalent neural network that can be trained to "design" circuitry when given a complete enough truth table to reflect the use conditions of the circuit in practice. Incomplete truth tables can also be used, but results for input conditions that were not specified are unpredictable. In this sense, the neural network does not generalize or "fill in the blanks". If one wishes to design a multiplier, he cannot only give the network a subset of training patterns and expect the resulting design to generate the missing patterns.

Using recent results on VC-dimensional analysis, it is possible to specify a simple perceptron network capable of exactly expressing any dichotomy corresponding to an m -input LUT. This understanding led to a way to prescribe a composite neural network corresponding to any $m \times n$ tiling of LUTs. Such a neural network was demonstrated to converge on a number of design examples using a simple back-propagation algorithm. Interestingly, it is straightforward to reverse-engineer the specification of each LUT corresponding to weights that are set in the various neural subnetworks during back-propagation. Since the current results were established on only a limited test case ($m=4$, $n=5$), it is not possible to assert that this technique would work for very deep LUT tiles ($m \gg 4$).

Neural networks do produce curious design results, certainly not the ones a human designer would usually prescribe. The fact that it works at all is intriguing, and it is surely more efficient than randomly specifying LUT patterns (i.e. guessing which possible of the 2^{mn} patterns might be viable), since the back-propagation approach provides meaningful cues that steer the design to convergence at least on the limited number of cases investigated. With more study and refinement, it could serve as a part of a more sophisticated heuristic that employs a neural network kernel as a greedy solver, whose results are fed into algorithms that could prune away some of the "nonsense" constructs initially produced and fine-tune or compact the remaining portions of the design. These results are encouraging, since they were easily obtained, and may provide a counterpoint against which more traditional approaches could be compared.

REFERENCES

- [1] Banerjee, P. *Parallel Algorithms of VLSI Computer-Aided Design*. Prentice Hall, Englewood Cliffs, NJ, 1994.
- [2] Miller, J.F. "On the Filtering Properties of Evolved Gate Arrays", *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, IEEE Computer Society Order Number PR00256, 19-21 July 1999, Pasadena, California.
- [3] Lyke, J. and G. Donohoe, "Cellular Automata Based Field Programmable Gate Array Architecture for VLSI and Nanoscale Implementations", *Proceedings of the 8th NASA Symposium on VLSI Design*, IEEE sponsored (contact dcox@mrcc.unm.edu), 20-21 October 1999, Albuquerque, New Mexico.
- [4] Wolfram, S. *Cellular Automata and Complexity*. Addison-Wesley, New York (1994).
- [5] DeHon, A. "Reconfigurable Architectures for General-Purpose Computing", Massachusetts Institute of Technology Artificial Intelligence Laboratory, *A.I. Technical Report No. 1586*, October 1996.
- [6] Chapman, S.P. "Capabilities of n -Input k -Nodes Layered Logic Unit Networks, MS thesis, University of New Mexico, EECE Department, December 1994.
- [7] Horne, W.G. "Recurrent Neural Networks: A Functional Approach". PhD Thesis, University of New Mexico EECE Department, May 1993.
- [8] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, New Jersey, 1999.
- [9] Carter, M.A. and M.E. Oxeley, "Evaluating the Vapnik--Chervonenkis Dimension of Artificial Neural Networks Using the Poincare Polynomial", *Neural Networks*, 12(1999): 403—408.
- [10] P. Koiran. VC dimension in circuit complexity. In *Proc. 11th IEEE Conference on Computational Complexity*, pages 81-85, 1996. ECCC technical report 95-051
- [11] Cover, T.M. "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition", *IEEE Transactions on Electronic Computers*, 6(1965): 326—334.
- [12] Wegener, I. *The Complexity of Boolean Functions*, John Wiley & Sons, New York, 1989.
- [13] Gaynier, R.J. and T. Downs. "Sinusoidal and Monotonic Transfer Functions: Implications for VC Dimension", *Neural Networks*, 8(6):901—904, 1995.

Jim Lyke is a senior electronics engineer who works for the Air Force Research Laboratory's Space Electronics and Protection Branch (Space Vehicles Directorate). He specializes in the development of microsystems approaches for space systems, involving a combination of advanced packaging, microelectronics, and micro-electromechanical systems and associated architectures. Mr. Lyke has initiated and/or managed dozens of technology programs, which have resulted in the development of over 60 different multi-chip module designs, including a number of spaceflight insertions. He has authored or co-authored over 50 publications, including three US patents and a book chapter on advanced packaging. He has a BSEE from the University of Tennessee, an MSEE from the Air Force Institute of Technology, and is a PhD candidate in electrical engineering at the University of New Mexico.