# A VLSI Implementation of a Parallel, Self-Organizing Learning Model

Tony R. Martinez
martinez@cs.byu.edu

George L. Rudolph

Linton G. Salmon

Matthew G. Stout

# A VLSI Implementation of a Parallel, Self-Organizing Learning Model

Matthew G. Stout
Linton G. Salmon

Department of
Electrical and Computer Engineering
Brigham Young University
Provo, UT 84602

George L. Rudolph
Tony R. Martinez

Department of Computer Science
Brigham Young University
Provo, UT 84602

## Abstract

*This paper presents a VLSI implementation of the Priority Adaptive Self-Organizing Concurrent System (PASOCS) learning model that is built using a multi-chip module (MCM) substrate. Many current hardware implementations of neural network learning models are direct implementations of classical neural network structures—a large number of simple computing nodes connected by a dense number of weighted links. PASOCS is one of a class of ASOCS (Adaptive Self-Organizing Concurrent System) connectionist models whose overall goal is the same as classical neural networks models, but whose functional mechanisms differ significantly. This model has potential application in areas such as pattern recognition, robotics, logical inference, and dynamic control.*

## 1 Introduction

Over the past few years, many companies and researchers have announced hardware implementations of neural networks [1, 2, 3, 4]. The integrated circuits (ICs) cited are representative of much current neural network implementation research. They are direct implementations of classical neural network structures—a large number of simple computing nodes connected by a dense number of weighted links [4, 5, 6]. The node function is typically a variation on sum-of-products [7]. Early learning models and implementations support only static topologies in the sense that learning does not involve changes in the network topology. More recent models support dynamic topologies in learning, and therefore implementations should also support this. The style of implementation presented here accomplishes this goal.

The implementation described is based on the connectionist architecture Adaptive Self-Organizing Concurrent System (ASOCS) [8, 9, 10, 11]. The primary goal of an ASOCS is similar to the goals of many decision-making connectionist systems—the system attempts to learn an arbitrary set of input-to-output vector mappings. However, an ASOCS differs from many other connectionist systems in the manner that this task is accomplished. ASOCS is essentially a rule-based system. That is, an ASOCS learns by the introduction, storage, and comparison of rules. The system is able to learn by keeping itself consistent with the rules and by dynamically changing its topology as new rules are introduced. In this way, an ASOCS can change its structure to suit a particular problem.

This paper presents an implementation of a version of ASOCS called PASOCS (Priority ASOCS) [10, 11] that is built on a multi-chip module (MCM) interconnection substrate. Although this implementation is modeled specifically after the PASOCS model, the general design of this MCM-based system is versatile and can be modified to reflect other connectionist models [10, 11, 12]. This model has potential application in areas such as pattern recognition, robotics, logical inference, and dynamic control.

Due to space considerations, background information dealing with the PASOCS learning model is not presented here but can be found in [11, 13]. The hardware implementation is described in Section 2, and the PASOCS MCM system, its test results, and ideas for related research are presented in Section 3.

## 2 Implementation

The IC described in this section was fabricated in $2\mu m$ digital CMOS. It was designed using a VLSI layout editor and simulated using an event-driven switch-level simulator. The circuit was constructed using simple cells (NAND gates, NOR gates, inverters, etc.) from standard cell libraries and no special purpose cells were required.

Each IC contains the functional hardware for one node of a PASOCS. The nodes of the PASOCS are connected together in a binary tree structure (the logic network), and the root node is connected to a general purpose logic analyzer that acts as a *control unit*. The node can be described as a combinational logic block with internal latches for information storage. Information presented to the PASOCS is broadcast from the control unit (CU) to the logic network (LN), and information from the LN that must be sent to the CU flows up from the bottom of the tree (the information is gathered.)

| *d* | *SId* | *SIc* | Output | *d* | *SId* | *SIc* | Output |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | match | 1 | 0 | 0 | match |
| 0 | 0 | 1 | match | 1 | 0 | 1 | no match |
| 0 | 1 | 0 | match | 1 | 1 | 0 | match |
| 0 | 1 | 1 | no match | 1 | 1 | 1 | match |

Table 1: The match function for one variable. The abbreviations are as follows: *d*, the value of the environment variable; *SId*, the data bit of the SI; *SIc*, the care/don't care bit of the SI.

The general operation of the PASOCS is asynchronous, that is, a global clock is *not* used to drive the system into sequential states. Rather, the clock is replaced by a global mode (sub-mode) signal that places the entire network into a particular mode of operation. Thus, the mode signal is an asynchronous signal that drives the system. Once the CU generates this mode signal, it broadcasts information to the LN, the nodes operate on the new information, and the results of these operations are gathered and eventually reach the CU. After the CU has waited a sufficient amount of time for information to be gathered, the CU generates the next mode signal.

The operation of the system is divided into three general modes: initalization, execution, and learning. In addition, the learning mode is divided into six sub-modes. The functions of the modes and sub-modes are summarized in this section.

## 2.1 Execution mode

The execution mode (*ex*) is the default mode of the system. If the system is not involved in learning, then it constantly updates its output in response to changes in the environment state. The output of the system consists of the right-hand side (RHS) of the winning node's stored instance (SI) and its corresponding priority. (The winning node is the node with the highest priority whose SI matches the environment state.) If the environment does not match the SI of any node, then a priority of zero is received by the CU.

Each variable of the SI (and the new instance in the learning sub-modes) consists of two bits: a "data" bit and a "care/don't care" bit. The data bit is the actual value of the variable. However, if the value of the care/don't care bit is zero, then the variable is a "don't care". The match output of every node during *ex* for a single variable is summarized in Table 1.

This prototype has an instance size of four input variables and one output variable. Therefore, the four-variable match function is accomplished by using four single-variable match functions as shown at the top of Figure 1. In this figure, *d0-3* represent the four environment variables, *SId0-3* the four SI data bit values, and *SIc0-3* the four SI care/don't care bit values.

The four single-variable match functions and the AND gate at the top of Figure 1 determine whether the left-hand side (LHS) of the SI matches the environment state. The priority is passed to the Priority

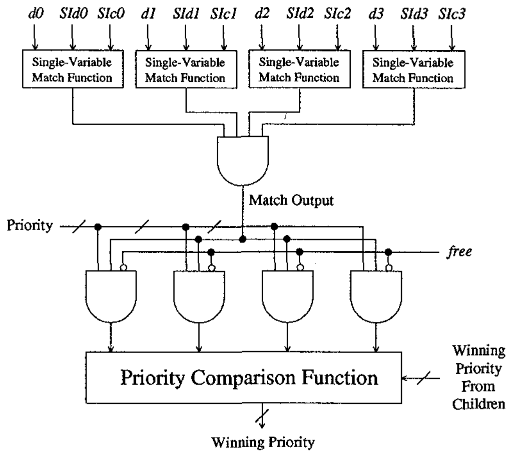
d0 SId0 SIc0 d1 SId1 SIc1 d2 SId2 SIc2 d3 SId3 SIc3
Single-Variable Match Function
Match Output
Priority
free
Priority Comparison Function
Winning Priority From Children
Winning Priority

Figure 1: Block diagram of execution mode.

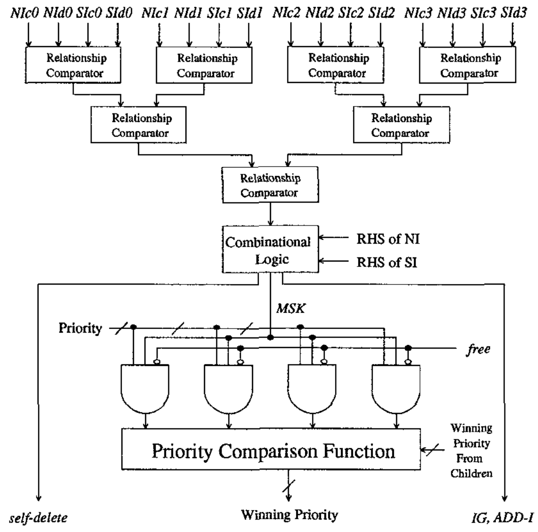
NIc0 NId0 SIc0 SId0 NIc1 NId1 SIc1 SId1 NIc2 NId2 SIc2 SId2 NIc3 NId3 SIc3 SId3
Relationship Comparator
Combinational Logic
RHS of NI
RHS of SI
MSK
Priority
free
Priority Comparison Function
Winning Priority From Children
self-delete
Winning Priority
IG, ADD-I

Figure 2: Block diagram of learning sub-mode 1.

Comparison Function only if a match occurs and the node is *not* free—that is, if the node contains a SI. Otherwise, the priority is masked to the value of zero. The Priority Comparison Function compares the priority of the node with the priorities of its children, and the winning (or highest) priority and corresponding RHS is sent to its parent. This process is continued for each node until the CU receives the RHS and priority of the LN's winning node.

## 2.2 Learning sub-mode 1

In learning sub-mode 1 (*l1*), the relationship of the SI to the new instance (the instance to be learned) must be determined. The relationship is determined with a three-level logic structure (shown at the top of Figure 2) that operates only on single variables of the new instance (NI) and SI. Figure 3 is a sequential state-flow diagram that clarifies the process of determining the relationship between the LHSs of two instances using single-variable comparisons. From the

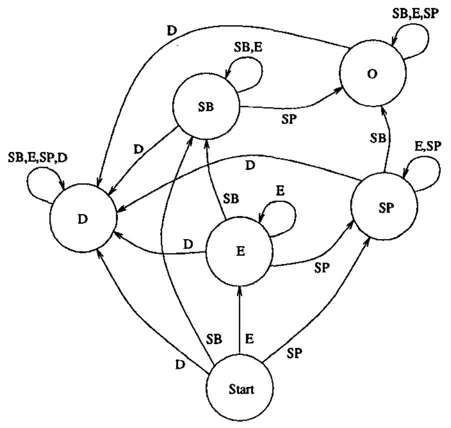

Figure 3: A sequential view of the parallel relationship comparison circuit (from *l1*) that shows how the relationship between two instances can be determined using only single-variable comparisons. The abbreviations are as follows: SB, Subset; E, Equal; SP, Superset; D, Discriminated; O, Overlap.

| | Subset | Equal | Superset | Discr. | Overlap |
|---|---|---|---|---|---|
| Con. | *ND/IG* | *D/MSK* | *D/MSK* | *ND/MSK* | *ND/MSK* |
| Dis. | *ND/ADD-I* | *D/MSK* | *D/MSK* | *ND/MSK* | *ND/ADD-I* |

Table 2: The actions generated by the node. The abbreviations are as follows: Discr., Discriminated; Con., Concordant; Dis., Discordant; *D*, Delete node; *ND*, Do not delete node; *MSK*, Mask priority to zero; *IG*, Ignore NI; *ADD-I*, Add the NI with a priority one higher than the highest priority of any inconsistent SI.

"Start" state, the four possible single-variable comparisons determine the path to the next state. For example, if the first-order relationship is "SB" (subset), then the path from "Start" to "SB" is followed. All successive comparisons determine the path to the next state. For example, if the next comparison results in "D" (discriminated), then the path to "D" is followed. Every successive comparison will remain at "D".

The relationship between the SI and NI determines whether the action signals *MSK*, *IG, ADD-I*, and *self-delete* (shown in Figure 2) for each node are asserted, as shown in Table 2. Once the values of these signals are determined, each node compares its action and priority with that of its children, resulting in a winning action and corresponding priority for the entire system.

The next mode or sub-mode of operation depends on the winning action reported to the CU in *l1*. That is, if the action *IG* is reported, then none of the following learning sub-modes are performed. If the reported action is *ADD-I*, or if a priority of zero is received by the CU (*MSK*), then the remaining learning sub-modes are performed.

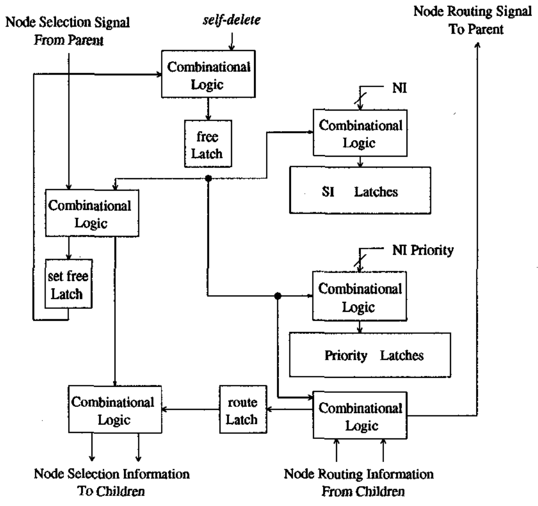

Figure 4: Block diagram of learning sub-modes 2–6.

## 2.3 Learning sub-modes 2–6

The function of the remaining sub-modes is to keep the system minimal and consistent. This is done by allocating a new node that contains the NI and corresponding priority and by deleting redundant or contradicting information from the system. After this is done, the CU proceeds to the default mode (*ex*) or to *l1* if additional learning is desired.

The block diagram of learning sub-modes 2–6 is shown in Figure 4. In learning sub-mode 2 (*l2*), the free latch is toggled to a high value if the *self-delete* signal was asserted in *l1*, effectively deleting the node from the system. In learning sub-mode 3 (*l3*), the NI is stored in the latches of every free node in the system. In *l4*, each node sends its routing information to its parent. At the end of this sub-mode, every route latch in the system contains information that determines a unique path from the root node down through the tree. This unique path is guaranteed to contain at least one free node that now has the NI and its priority stored in its latches. From the information contained in the newly acquired routing information and the present state of the free latch, the next state of the free latch of each node is determined in learning sub-mode 5 (*l5*). This state is temporarily stored in the set free latch until its value is needed in learning sub-mode 6 (*l6*). Finally, in *l6*, the set free latch's value is used to update the free latch's value. In other words, a single free latch's value is set to a low logic level, effectively adding it to the tree.

## 3 System design, testing, and ongoing research

Three of the ICs described in Section 2 have been interconnected on a multi-chip module interconnection substrate (see Figure 5) in a three-node feasibility study, and this small PASOCS has been tested. To test the system, various combinations of instances were presented, and the system's outputs were observed.

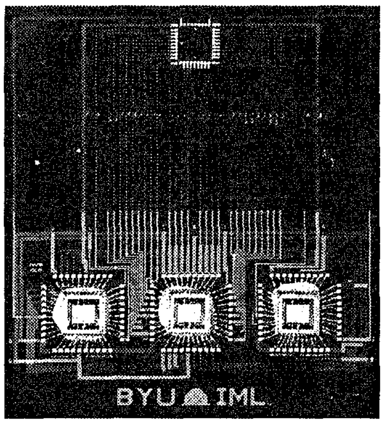

Figure 5: The PASOCS MCM prototype. The dimensions of the MCM are 2.5cm × 2.3cm.

The execution mode of the PASOCS operates as predicted by the software simulation, but two functions of the learning sub-modes (associated with overall network minimization and rule relationships) do not operate as predicted. These problems can cause the system to store invalid information and therefore return incorrect information in the execution mode. It should be noted that these are problems with the specific implementation of the ICs and not with the functionality of the PASOCS model or with the original conceptual design of the ICs, described in more detail in [13]. Despite these problems, most of the functions of the three-node PASOCS are functioning according to original design specifications. A detailed report of the test results can be found in [13].

In addition to the research reported here, ASOCS models other than PASOCS are also being investigated. These and other more classical neural network models can benefit from the results presented in this paper. Current research seeks to extend the general ideas presented here to other models.

Other related research seeks high-density interconnect technologies that can be exploited to create larger neural network systems [13, 14]. MCM interconnection and packaging techniques offer a promising solution to the high interconnect and processing element densities required for such hardware implementations. The MCM for this prototype was fabricated in the Integrated Microelectronics Laboratory at Brigham Young University. Other models that can benefit from MCM characteristics are also being investigated.

## 4 Conclusion

This paper described a VLSI implementation of a connectionist system that was built using an MCM interconnection substrate. The IC differs significantly from many other commercially available ICs and research projects dealing with connectionist architectures. Many of these are modeled after the classical "weighted connection" neural systems. However, this IC is modeled after PASOCS (Priority Adaptive Self-Organizing Concurrent System) and represents a different approach to implementation.

## Acknowledgements

This research was supported in part by the Endowed Chair of Engineering, occupied by Dr. Linton G. Salmon, and by grants from the National Science Foundation and Novell, Inc.

## References

[1] Intel Corporation. *80170NX Electrically Trainable Analog Neural Network*, Jun 1991.

[2] American NeuraLogix, Inc. *NLX420 Neural Processor Slice*, Feb 1992.

[3] M. Wright. Neural-network IC architectures define suitable applications. *EDN*, pages 84–90, Jul 1991.

[4] M. Verleysen and P. G. A. Jespers. An analog VLSI implementation of hopfield's neural network. *IEEE Micro*, pages 47–55, Dec 1989.

[5] D. Hammerstrom, T. Leen, and E. Means. Dynamics and VLSI implementation of self-organizing networks. In *Advanced Neural Computers*, pages 185–92. North-Holland, 1990.

[6] U. Ramacher. Hardware concepts for neural networks. In *Advanced Neural Computers*, pages 209–18. North-Holland, 1990.

[7] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* MIT Press, 1988.

[8] T. R. Martinez and D. M. Campbell. A self-adjusting dynamic logic module. *Journal of Parallel and Distributed Computing*, 11(4):303–13, 1991.

[9] T. R. Martinez and D. M. Campbell. A self-organizing binary decision tree for incrementally defined rule based systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1231–7, Sep/Oct 1991.

[10] T. R. Martinez. Consistency and generalization of incrementally trained connectionist models. In *International Symposium on Circuits and Systems*, pages 706–9, 1990.

[11] T. R. Martinez, D. M. Campbell, and B. W. Hughes. Priority ASOCS. *Journal of Artificial Neural Networks*, 1(3), 1994.

[12] G. Rudolph and T.R. Martinez. An efficient static topology for modeling ASOCS. *Artificial Neural Networks*, pages 729–34, 1991.

[13] M. G. Stout. Multi-chip module design for neural networks. Master's thesis, Brigham Young University, 1994.

[14] M. G. Stout, G. L. Rudolph, L. G. Salmon, and T. R. Martinez. A multi-chip module implementation of a neural network. In *Proceedings of the IEEE Multi-Chip Module Conference*, pages 20–5, 1994.