FAULT SIMULATION OF A WAFER-SCALE NEURAL NETWORK

Norman L. May
B.S., Iowa State University, 1983

A thesis submitted to the faculty
of the Oregon Graduate Center
in partial fulfillment of the
requirements for the degree
Master of Science
in
Computer Science & Engineering

February, 1988

The thesis "Fault Simulation of a Wafer-Scale Neural Network" by Norman L. May has been examined and approved by the following Examination Committee:

Dan Hammerstrom
Associate Professor
Thesis Research Advisor

Robert Babb II
Associate Professor

Richard Hamlet
Professor

Kit Bradley
Tektronix

ACKNOWLEDGEMENTS

I would like to thank Dan Hammerstrom for his guidance throughout the project. I am grateful to Tektronix for allowing the time to work on this project. Also, many thanks to my wife, Donna, for all her patience and help.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Fault Simulation of a Wafer-Scale Neural Network

Norman L. May, M.S.
Oregon Graduate Center, 1988

Supervising Professor: Dan Hammerstrom

The Oregon Graduate Center's Cognitive Architecture Project (CAP) is developing a flexible architecture to evaluate and implement several types of neural networks. Wafer-scale integrated silicon is the targeted technology, allowing higher density and larger networks to be implemented more cheaply than with discrete components. The large size of networks implemented in wafer-scale technology makes it difficult to assess the effects of manufacturing faults on network behavior. Since neural networks degrade gracefully in the presence of faults, and since in larger networks faults tend to interact with each other, it is difficult to determine these effects analytically. This paper discusses a program, FltSim, that simulates wafer manufacturing faults. By building an abstract model of the CAP architecture, the effects of these manufacturing faults can be determined long before proceeding to implementation. In addition, the effects of architectural design trade-offs can be studied during the design process.

1. INTRODUCTION

The Oregon Graduate Center (OGC) Cognitive Architecture Project (CAP) is developing a flexible architecture to evaluate and implement several types of neural networks. Wafer-scale integration is the targeted technology for implementing the architecture, allowing higher density and larger networks to be implemented than with discrete components. As the size of the networks implemented increases, the effects of processing faults on the architecture become more difficult to evaluate. Neural networks degrade gracefully in the presence of faults, making analysis difficult. Also, especially in larger networks, faults tend to interact with each other. To what extent processing faults will effect the operation of the network is the question the fault simulator, Fltsim, answers.

Neural networks are fault tolerant and are scalable. Each processing node is working asynchronously on part of the problem to be solved. Messages, (current node output states) are passed between nodes, but the actual function and memory of the network are completely distributed[Ham86a]. This node independence allows additional nodes to be added to the architecture with little or no overhead, thus achieving scalability. The node independence also improves the fault tolerance of the network. If any of the nodes are damaged, the entire function is not lost, but nodes may participate in several representations, only decreasing the fault tolerance if the node is damaged.

*This work was supported in part by the Semiconductor Research Corporation contract no. 86-10-097, and jointly by the Office of Naval Research and Air Force Office of Scientific Research, ONR contract no. N00014 87 K 0259.

The neural network can be visualized as a large, multidimensional, directed graph of connection nodes (CNs), called the *n-graph*. The physical network is comprised of a repeated pattern of processing nodes (PNs) interconnected by bus structures. The interconnections between the PNs form a graph referred to as the *p-graph*. Typically, the n-graph is much larger than the p-graph, so that a subset of connection nodes in the n-graph is mapped onto a physical node (p-graph node). The number of CNs in each PN may vary. One extreme uses one PN to implement all the CNs, one connection at a time, which is too slow for large networks. The opposite extreme is a "direct" implementation using one CN per PN, which requires more silicon area for all the PNs and PN connections.

The fault tolerance of the architecture is affected by the p-graph to n-graph mapping. Mapping a subset of CNs onto a PN reduces the amount of fault tolerance in the network implementation. If a PN is defective due to processing faults, the entire CN subset is defective, having more impact on the operation of the network. Although, some fault tolerance is preserved, since the the function and memory of the physical network are distributed over the PNs. Losing one PN will not cause the entire network function to be lost. The mapping of the n-graph to the p-graph has a major effect on the fault tolerance of the network and can be evaluated using Fltsim.

The main limitation in the production of cost effective wafer-scale integrated devices is the processing faults that occur. Each wafer has defects that cause malfunctions in their operation. Some architectures that are implemented using wafer-scale integration try to route around dead cells and have redundant nodes that can be swapped in to replace these dead cells[Lei85a,Har88a]. Swapping cells involves effort to determine

which cells are dead and redundant hardware and communication paths to route around the dead cells. The cost for this extra effort and hardware redundancy made wafer-scale integration more expensive than discrete implementations. Neural networks, however, are inherently fault tolerant and do not require as much redundant hardware. The amount of redundant hardware required can be evaluated using Fltsim.

Fault simulation of the CAP architecture is used to predict the operation of the network containing manufacturing defects. These predictions can be used to improve the fault tolerance of the networks by providing feedback before the design has been implemented. Large networks can be simulated using Fltsim, due to the scalability of the architecture (e.g., all the PNs have the same structure). More realistic faults can be modeled in the architecture using the fault characteristics of wafer-scale integration and by taking fault interactions into account.

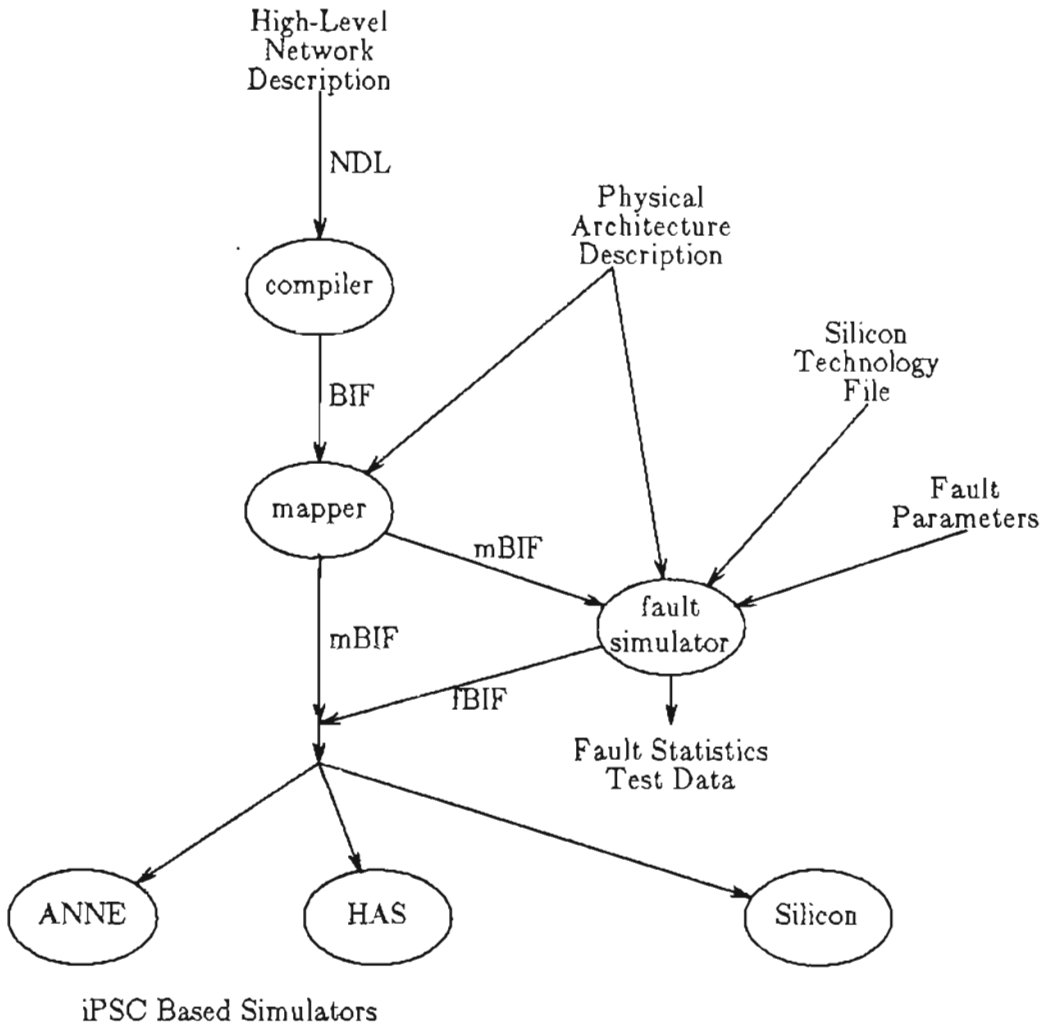
A fault simulator program tool developed to evaluate the CAP architecture is described in this thesis. The purpose of the fault simulator is to use standard models to model the faults typically found in a wafer, not to develop new ways to model faults in a wafer circuit. Chapters 2, 3 and 4 provide background information on the simulation environment, the CN/PN models and the fault models, respectively. Chapter 5 discusses the design and operation of the fault simulator, followed by the results of the simulation in Chapter 6 and a summary in Chapter 7. For the purposes of this thesis, it is assumed that the reader is familiar with neural network concepts.

2. SIMULATION ENVIRONMENT

The Cognitive Architecture Project group at the Oregon Graduate Center has developed tools and languages used to evaluate, simulate, and implement several different types of neural network architectures, as shown in Figure 1. These tools and languages are general in nature, allowing several different types of neural networks to be simulated and evaluated. A brief overview of these tools will help in understanding how the fault simulator interfaces with them.

A major goal in the design of the CAP tools is flexibility. To achieve this goal, much of the information the tools need for modeling functions of the network is read from data files. Obtaining the information from input files allows many architectures and fault models to be simulated more quickly than if the models were built into the actual code of the simulator. Also, some of the files are read by several different tools being developed at OGC, avoiding redundant information between files and helping to ensure information consistency between tools. Appendix B: Fault Simulator File Formats discusses the file formats in more detail.

A network specification begins with general descriptions and proceeds to greater levels of detail. A user first specifies a network with NDL, an extensible Network Description Language. NDL is then translated and expanded into a BIF file, which contains network structure, state, and state transition information describing the n-graph. In order for the simulators to use the information thus generated, a computer program,



NDL - Network Description Language
BIF - Beaverton Intermediate Form
mBIF - mapped BIF
fBIF - BIF fault fields
ANNE - Another Neural Network Emulator
HAS - Hardware Architecture Simulator

Figure 1 - Neural Network Tool Interaction.

"Mapper", maps CNs to physical computational elements, PNs, using a PAD (Physical Architecture Description) file and places the mapping information in the mBIF file[Bai88a]. A physical computational element corresponds to a single processor on a multi-processor machine, a device on a chip, or any other kind of processing element that simulates a connection node. The input to either simulator is then a BIF file augmented with physical node mappings (mapped mBIF file).

The PAD file describes the physical implementation of the architecture. It contains the number of PNs on the wafer and their geometry, the maximum number of CNs in a PN, the number of data bits/signal lines for each word or communication path, and the connectivity for the communication paths. From this description, a complete block diagram of the circuitry (PNs and their interconnectivity) can be built.

There are two architecture simulators, each serving a different purpose. The more general purpose simulator, ANNE (Another Neural Network Emulator), allows for the expedient testing and debugging of a wide variety of connectionist/neural network models[Bah88a]. Models can therefore be "stress tested" before committing them to the more special purpose simulator, HAS (Hardware Architecture Simulator), which simulates network behavior using a chronologically correct software emulation of the targeted wafer-scale hardware[Jag88a]. HAS provides the user with performance assessments of hardware design choices and points out potential weaknesses. Each simulator provides an overall structure to emulate the network. Within the CNs in the network are various functions to calculate a CN output. These functions are provided through user routines, which are supplied by the user and called by the architecture simulators.

The fault simulator uses a PAD file, a silicon technology file and a mapped BIF file to generate a physical representation of the neural network on the wafer. To convert the blocks of the block diagram described in the PAD file into actual physical representations of the architecture, the sizes of the blocks must be known. The size information is read from a technology file. It contains sizes for memory cells, buffers, and all the other basic elements that comprise the hardware blocks. These sizes are multiplied by the number of devices internal to the block to obtain the block size. Faults are generated and located in the physical representation using the characteristics of wafer-scale statistical fault models. The fault parameters required to generate the faults in the physical model are read from the fault parameters file. Parameters such as the average defect density, fault clustering coefficients and ratio of fault types are included.

The faulted BIF file, fBIF, which contains the fault fields for the mBIF file, is written by the fault simulator. The network simulators, HAS and ANNE, read both the mBIF file and the fault fields in the fBIF file to modify their operation accordingly. Differences in network operation due to faults can be evaluated to determine the impact of the faults and hence the impact of certain design decisions.

The fBIF file contains the fault fields to be included in the user routines of the architecture simulators HAS and ANNE. The user routine will make subroutine calls to system fault routines at various points in the CN calculation. The fault routine calls will access the fault fields contained in the fBIF file to simulate the faults in the hardware. The user routine will call the fault routine several times, passing different parameters each time to model faults in various hardware blocks which affect different sections of the n-graph. The appendices provide more detail on the interface between the fault simula-

tor and the architecture simulators, and how faults in the various hardware blocks are modeled in the n-graph.

Fltsim can generate two other output files, *fstat* and *test*. The *fstat* file contains all the fault statistics for the fault simulation and *test* contains intermediate Fltsim values, which gives more detail about the network size and fault calculations.

The fault statistics summarize the faults in the physical system and how these faults affected the n-graph. They also indicate the n-graph utilization of the p-graph. These statistics list each fault type, the section of the hardware block that it occurred in, and where in the n-graph it was mapped. Physical faults can affect the n-graph in multiple areas depending on the mapping of the n-graph onto the p-graph. If multiple faults affect a single n-graph section, the worst case fault is determined and is modeled in the network. The worst case fault is selected by either combining the faults into one fault, or determining which of the faults has more impact on the network. The statistics file indicates the physical defects that were combined to fault a single BIF section.

The utilization of the p-graph by the n-graph is listed with the fault statistics to help evaluate the faults that occurred in the network. For example, a small n-graph mapped onto a large p-graph will result in few faults in the p-graph affecting the operation of the network. When faults in the p-graph do not have much affect on the n-graph, it may mean either that the p-graph is underutilized or the design is fault tolerant.

The *test* file contains intermediate values used in the fault simulator. Input file values are echoed in the *test* file, such as the sizes of the PN blocks and the actual fault locations. The *test* file can be used to debug the system or to give greater information about the fault generation in the network.

3. NEURAL NETWORK MODEL

Neural Model

A neural network model is comprised of many processing units, referred to as CNs, operating asynchronously. Each CN transforms its inputs into a single output value using non-linear functions. The function that is used to calculate the node output value depends on the type of neural network used. These CN functions are derived so that the overall function of the network is to map a set of input values to a desired set of output values using a "best match" selection. The information stored in the network that most closely matches the input selection criteria is selected as the output of the network.

The CNs in the neural network are interconnected by direct, node-to-node links. Although there is large connectivity, it is not total, i.e., not all nodes are connected together[Ham86a]. Figure 2 shows the conceptual model of a connection node (CN). Separate site functions, S_{site}, and a CN function, $\int_{CN}$, are shown in the figure. The outputs of each site function are used as the inputs for the CN function. A single value is calculated in the CN function to be passed to the output site. The output site passes the CN output to the next destination, another CN input site or the output of the network. If the destination is another CN, the output site signal will excite or inhibit the destination CN(s).

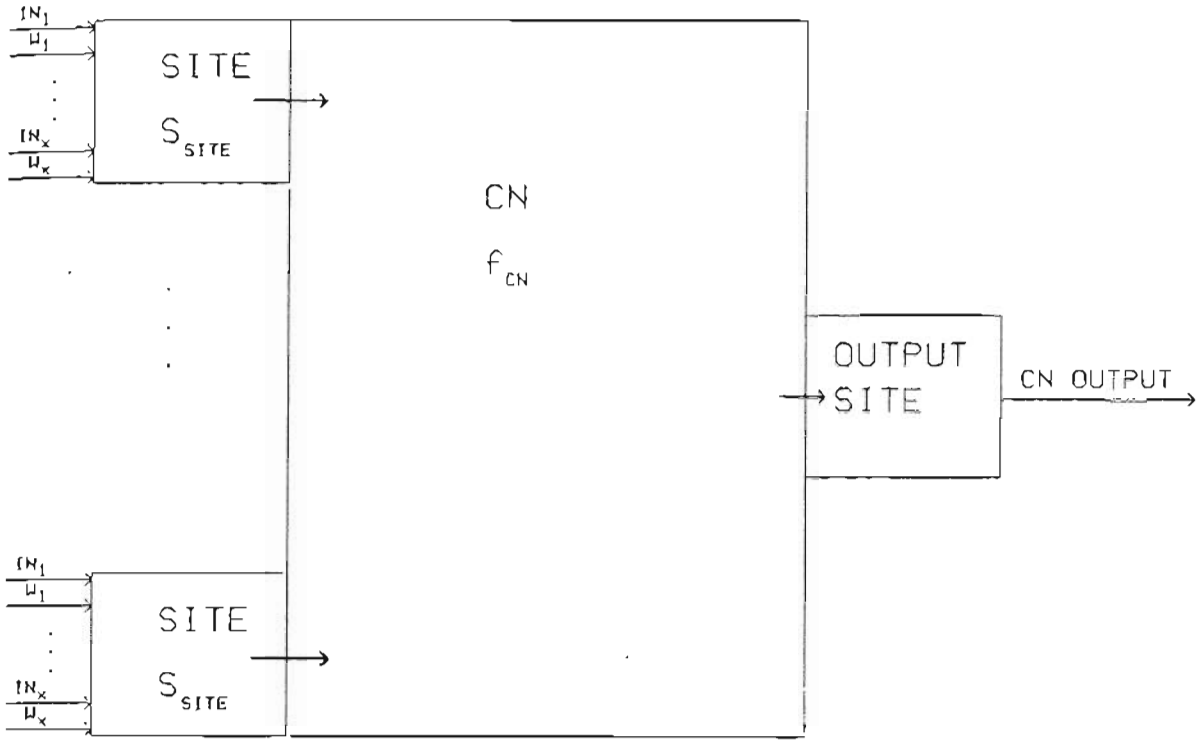


Figure 2 - Connection Node Model.

Hardware Implementation

An n-graph to p-graph mapping combines groups of CNs into Processing Nodes (PNs) as shown in Figure 3. The CN interconnections would be inefficient to implement directly with current silicon technology due to their large number. Silicon provides a small number of high bandwidth connections, but CNs require a large number of low bandwidth connections. Therefore, a connectivity mismatch exists between silicon technology and the required architecture of the network. Interconnection buses are multiplexed since metal lines are too expensive to dedicate to a single CN connection[Bai86a]. By combining CNs into PNs and using a multiplexed interconnection scheme, the efficiency of the network is preserved.

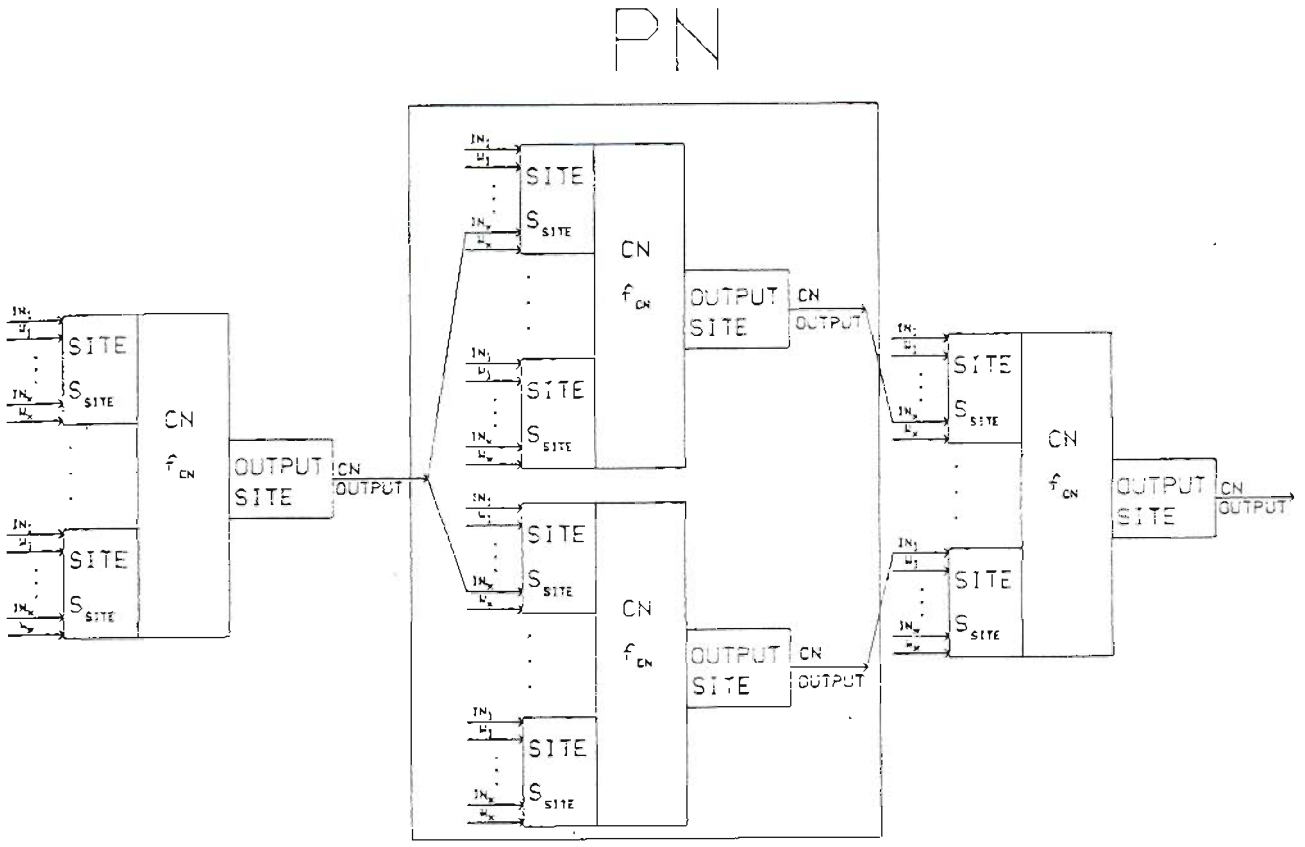


Figure 3 - N-graph to P-graph mapping.

Figure 4 shows a partial block diagram of the hardware implementation of a CN. The CN input is received from another CN output or is an external input to the network. For the assumed PN model, this value is stored in the CN memory. A corresponding weight value is stored in another memory block. A Digital to Analog Converter (DAC) is used to convert these binary numbers to an analog signal corresponding to the multiplication of the CN value and the weight. Each analog signal is combined in the Analog to Digital Converter (ADC) which acts as an analog arithmetic logic unit, to convert the output back to a digital format. The ADC calculates the CN output using

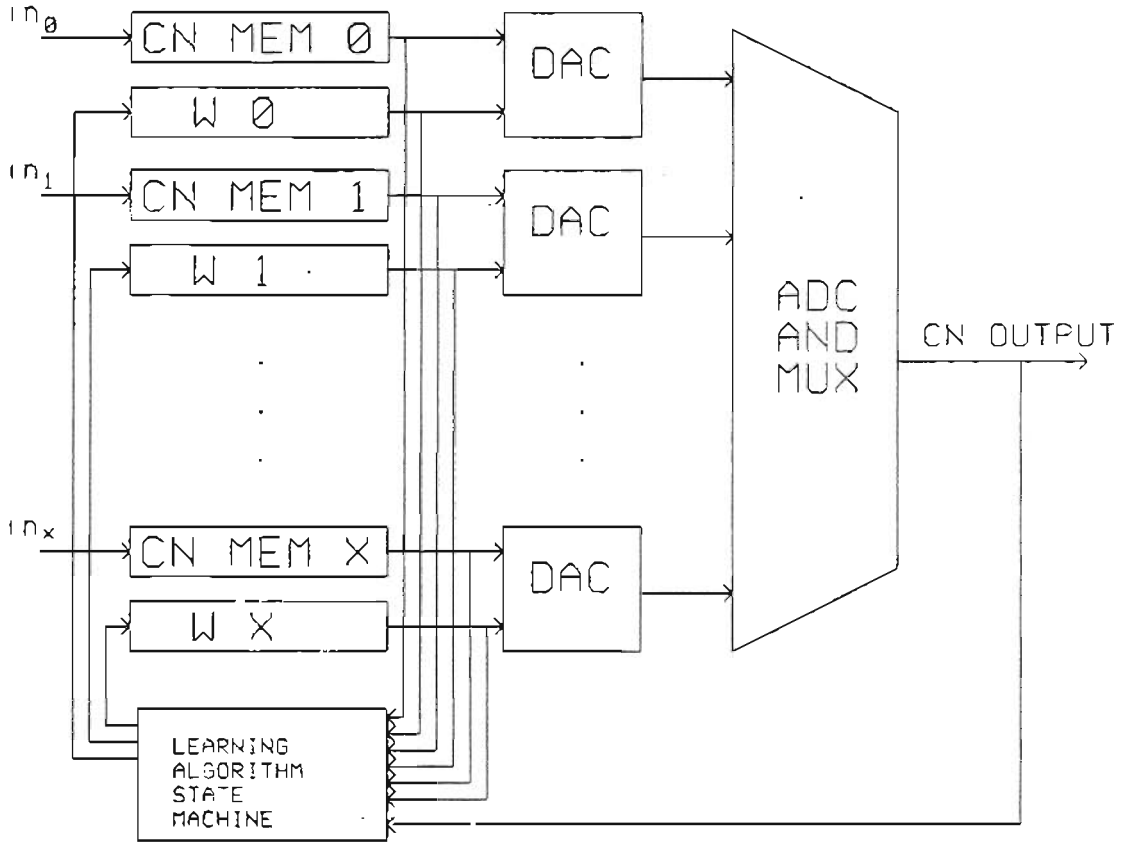


Figure 4 - Partial Hardware Block Diagram for a CN.

the equation:

$$out = f_{CN}(S_1(in_1, w_1, ..., in_{n_1}, w_{n_1}), \cdots S_x(in_1, w_1, ..., in_{n_x}, w_{n_x})),$$

where f_{CN} is the CN function and each S is a site function. The site function uses as arguments the link inputs and their respective weights from the other CNs. One arithmetic unit calculates several CN function outputs in a time multiplexed fashion. For

example, the initial networks at OGC use the site and CN functions shown below:

$$S_{site} = \sum_{x} (in_x \times w_x)$$

$$f_{CN} = \frac{1}{1 + e^{-S_{site}}}$$

Arithmetic is performed using analog techniques instead of digital in order to save silicon space on the wafer, increase fault tolerance and increase the speed of the network.[1] Although digital signals are preferred because digital signals are easier to multiplex over several interconnections and provide more reliable communication.

The Learning Algorithm State Machine, LSM, implements the weight adjustment or learning algorithm for the CN. Most learning algorithms use the current output for the CN, the current input, a learning rate constant, and a second order term not included here. The arithmetic operations typically performed by the learning state machine include multiplication and subtraction, and perhaps others, depending on the learning algorithm. Therefore, the learning state machine contains multiplier and subtraction circuitry, tailored to the learning algorithm to be used and a Programmable Logic Array, PLA, is used to implement the LSM control. The arithmetic circuitry calculates a new weight to be stored in the WEIGHT memory hardware block. The LSM operates concurrently and asynchronously with the other CN functions.

Figure 5 shows the PN block diagram. Several CNs (shown in Figure 4) are mapped onto this block diagram. No global control signals are needed for the PNs, and each PN operates asynchronously with respect to the rest of the PNs in the network.

[1] Patents Pending - OGC

Only the messages that are passed between PNs require synchronization.

Two modes of communication between PNs are implemented. One uses a grid network, shown in Figure 6, which is called Point-To-Point (PTP) communication and the second is a tree-like structure, shown in Figure 7, called the PN Broadcast Hierarchy,

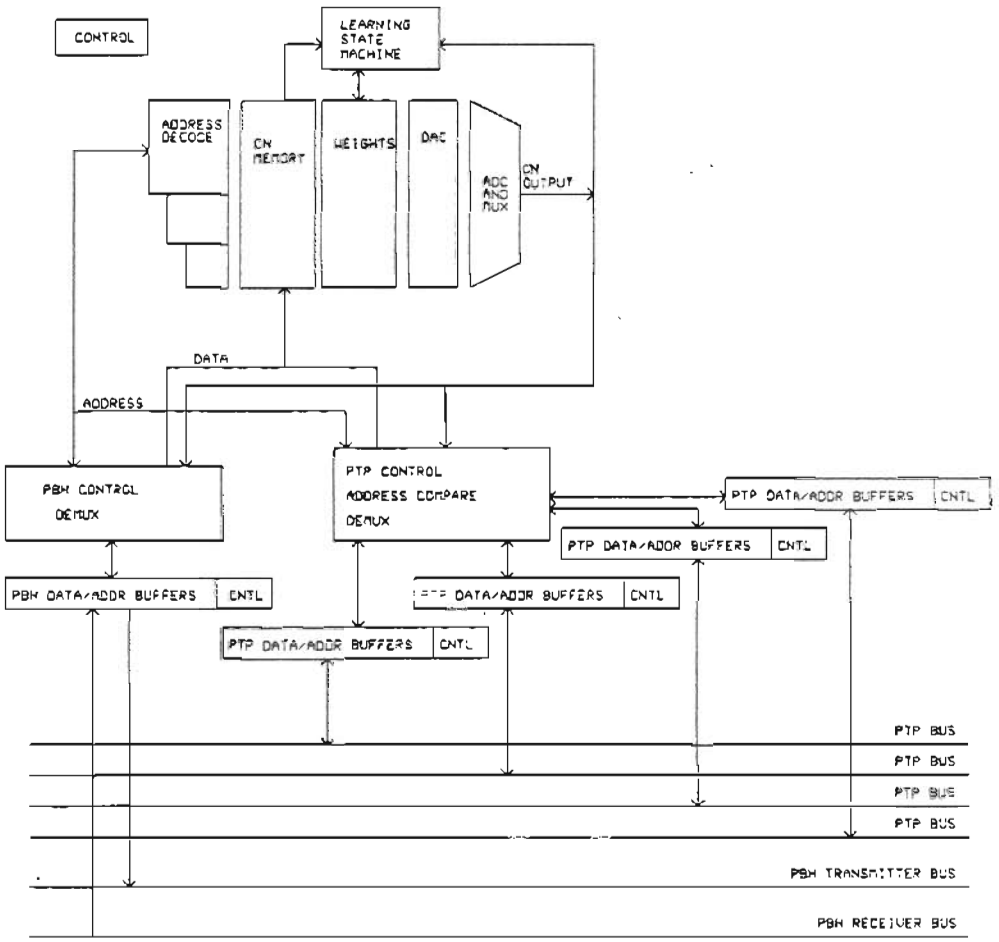


Figure 5 - PN Block Diagram.

(PBH).[2] The PTP network connects a PN with each of its four nearest neighbors. Messages include a destination PN address that is used to route the message through each PN. The PN receives a message and determines whether it is the destination PN. If the PN was not the final destination, the message is retransmitted to the next PN using a predefined routing algorithm.

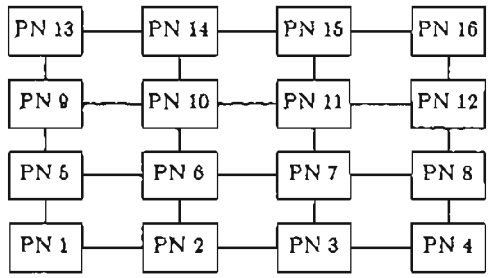


Figure 6 - PTP Bus communication.

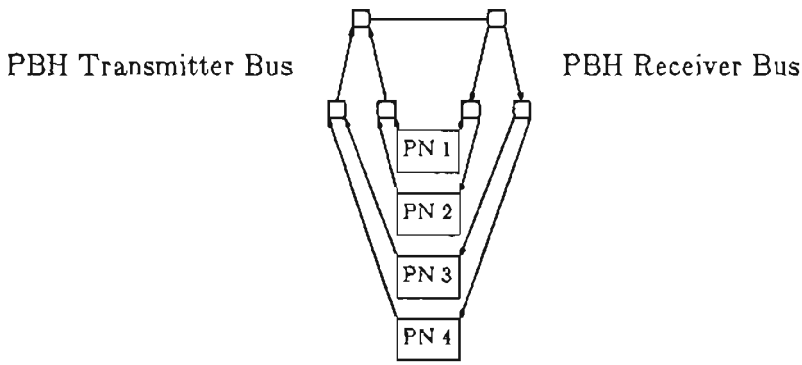


Figure 7 - PBH Bus communication.

[2] PTP and PBH have Patents Pending - OGC

The PBH is used to broadcast messages to several PNs simultaneously, updating many CNs with one message. PNs are grouped into PBH broadcast regions that are physically connected by a common PBH bus. When any of the PNs in the region sends a message, all the PNs in the region receive it. The PBH bus is split into transmitter and receiver link sections. To broadcast a message using the PBH network, a PN sends its own source CN address along with the data onto the PBH transmitter bus. It is received by a concentrator which retransmits the message to the next higher level concentrator in the tree. Each concentrator accepts messages from two lower level concentrator, allowing only one of the two to transmit at a time. At the top node, the message is then sent to the receiver links. Messages are received by deconcentrators and are retransmitted to two lower level deconcentrators. All the PNs in the region receive the message from the broadcast tree. For each message address that matches an entry in the CN address decoder, the data is stored in the CN MEMORY hardware block. PN messages use the PBH transmitter line to traverse up to the top node in the tree, and descend the receiver bus, communicating simultaneously to all connected PNs. Control lines are used to avoid collisions and perform arbitration as the messages traverse up the tree.

The PBH regions may overlap, allowing PNs to belong to multiple PBH regions. The PN will transmit and receive messages from all the PBH regions it belongs to. Each CN in the PN will determine which of the PBH regions to transmit messages to and which regions to receive messages from.

In Figure 5, the PBH Control/Demux and PTP control/address compare/demux control their respective communication channels, both in the sending and receiving of messages. One set of data lines is used to send both the address (CN #) and data (CN

output value) information in each of the PTP and PBH communication schemes. This information may be sent in a serial mode, depending on the architecture modeled. The width of the data bus is read in from an architecture description file. Control lines are used to handshake the data (i.e., Data Valid and Data Accepted).

The information from other CNs is routed though the PTP or PBH section in the PN. The address field is separated from the incoming word and sent to the address decoder to check for a match. The address decoder uses a Content Addressable Memory, CAM, to check for the presence of that CN number (address). If present, the data information is loaded into the matching CN MEMORY location(s), which contain other CN output values.

As mentioned before, the contents of the CN MEMORY and corresponding weights are used by the computation unit to calculate each CN output. The computation is performed by the DAC and ADC blocks. Each CN in the PN has its output calculated in a cyclic manner. Each output and its CN number are then routed to either the PTP or PBH communication channel, as predetermined by the type of CN routing, to be passed on to other CNs.

A Learning State Machine monitors the CN outputs to calculate the new weight for that CN. As each CN output is calculated, the LSM calculates the new weight value using a predetermined learning algorithm. The new weight value to be used for the next incoming CN value is stored in the WEIGHTS memory.

A PN control block is included to represent any control signals that are used throughout the PN. The control circuitry represented is the portion of the PN circuitry that coordinates the operation of all the hardware blocks within a PN. For example,

circuits controlling the timing of data transfer between all the hardware blocks in a PN would be represented in the PN control block. Control circuitry local or affecting only one PN hardware block should be included in that hardware block.

4. DEFECT FAULT MODELS

Originally, defects in integrated circuit fabrication were considered to be purely random. As the defect densities were reduced by better process control, it was assumed that those defects were random and could be modeled using a Poisson distribution[Sta86a]. Later, it was discovered that the defects were not random. As integrated circuit size increased, it was discovered that the defect distributions deviated from the simple Poisson distribution model. Larger circuits exhibit fault clustering which is not modeled using simple Poisson distributions and a more detailed model must be used. A compound Poisson distribution can be used in which a wafer is sectioned into areas with the average number of faults in each area specified by a variable. Clustering can be modeled as independent regions with varying numbers of faults[Sta86a, Che87a]. Within each area, the Poisson distribution can be calculated as before.

CMOS circuit technology is the process chosen to implement the neural networks at OGC. A typical p-well CMOS process with one metal layer requires 7-8 processing steps and masks[Wes85a]. Each of these steps can potentially add new defects to the wafer. There are two categories of faults that can occur in processing a wafer, global defects and local defects[Har88a, Che87a]. *Global defects* affect the operation of the entire wafer and are generally catastrophic in nature. Global defects are generally process defects and include problems such as mask misalignment and oxide thickness defects. All, or most of the cells on the whole wafer will have the same fault defect present. The number of wafers with global defects can be derived statistically and affects the yield

directly. Thus, global defects are not considered by the fault simulator.

The second category, *local defects*, are those that occur at single points on the wafer. Local defects include extra or missing material defects, oxide pinholes, or via resistance faults[McD86a], which result in opens and shorts in the circuit. The fault simulator will model local defects, as these affect a single PN or groups of PNs which may not critically affect the output of the network. The network will operate despite faults due to the inherent fault tolerance of the neural network, but the performance will be degraded. Fault simulation will provide an analysis tool to determine the degree of performance degradation in large physical implementations of neural networks.

Fault Distribution

Local processing defects in a wafer can be characterized by statistical studies of defects on other wafers, which indicates that the fault density increases towards the edge of a wafer and that faults tend to cluster in groups[Sta86a]. These characteristics are used in the fault description model to determine what the effects of actual processing faults would be on a particular network. Defects tend to cluster within wafers and among wafers in a batch. Clustering can be attributed to the batch oriented process, where the processing conditions vary from lot to lot[Wal86a], or from concentrations of impurities in the air or in the process.

To get some idea of the fault spatial distribution, F. M. Armstrong and K. Saji examined 12 blank wafers from a manufacturing line to determine the location of all defects, which lead to circuit faults[Sta86a]. Each wafer was sectioned off into smaller areas referred to as *quadrats*, and the number of defects in each quadrat was counted.

Various quantities of quadrats were used for each wafer, consisting of a 12x12, 8x8, 6x6, 4x4 and a 2x2. The distributions of the numbers of defects per quadrat were tabulated. The mean, variance and mean-to-variance were compared for each quadrat size. The larger quadrats, (quadrats with the fewest grids, such as the 2x2), had the greatest deviation from Poisson statistics. This deviation indicates faults were clustered within the quadrats[Sta86a]. The goodness of fit between the statistical model and the data was determined using the chi-square test. The tabulated defect statistics were analyzed to determine which of four different compound distributions provided the best fit for these data. Of the twelve wafers tested, four wafers were best modeled by a mixed Poisson-binomial distribution, four others by a Neymann Type A distribution, three others by a negative binomial distribution, and one wafer fit all of these distributions equally well. None of the compound distributions matched the data significantly better than the other distributions. But, for all the wafers, each of these distributions gave a much better fit than Poisson's distribution[Sta86a].

Several yield models based on mixed Poisson statistics are derived from observed statistical data and not directly from the wafer fabrication process[Har88a]. Since fault distributions vary between and within process lines and product lines, fault models tend to vary, causing some dispute on their validity. The end result of the model though is to simplify the physical process of fabricating a circuit, and as long as the model fits the actual data within the given tolerances, the model is valid. Since all the distributions modeled the fault clustering similarly and all of them did better than the Poisson distribution, any of the fault models can be used to model the fault distribution. For the analysis in this thesis, a Poisson-negative binomial was used to model the fault clustering.

The probability of finding x defects in a wafer quadrat is:

$$Pr(x) = \frac{\Gamma(\alpha+x)}{x!\Gamma(\alpha)} \frac{(\lambda/\alpha)^x}{(1+\lambda/\alpha)^{x+\alpha}}$$

where λ is the expected (average) number of defects in an area and α is the clustering coefficient between areas on the wafers. Lower values of α correspond to greater clustering. The variance for the number of faults found in an area is:

$$var(x) = \lambda(1+\lambda/\alpha)$$

These equations were found to fit Stapper's test data and were verified using the chi-square test. Stapper concluded that these tests were conservative, and that actual wafers would exhibit even more fault clustering traits[Sta86a].

A second trait of the spatial distribution of faults is related to the distance from the center of the wafer. Defects in a wafer are more common around the edge of the wafer and exhibit a radial distribution of the form:

$$h(r) = c_1 + c_2 e^{c_1 r}$$

where $h(r)$ is the probability of a defect occurring at a given distance r away from the center of the wafer, and c_1 and c_2 are constants[Wal86a]. There are many factors that contribute to this effect: the edge being more exposed to air, tilting of the silicon wafers while processing, the lithography defocusing towards the edge of the wafer, or the handling of wafers by the edges.

For this thesis, both fault clustering and the radial distribution of faults are modeled. The fault simulator combines Stapper's fault clustering model with the radial

fault distribution model, which is the same concept used by Harden and Strader[Har88a].

Fault clustering is modeled in the neural network wafer by dividing it into a 12x12 grid with varying defect densities in each area. The number of faults in each area will be determined by a two zone radial distribution where the density of faults will be greater towards the edge of the wafer. The inside zone will have a lower defect density than the outside zone. The average total number of defects (ATD) to model in the network is equal to the defect density (defects per square inch) multiplied by the area required for the network. Each quadrat starts with a common base defect density equal to ATD/144, or the total number of defects divided by the number of grid areas, or quadrats. The common defect density for each quadrat will produce the fault clustering effect using the Pr(x) equation to calculate the number of faults per quadrat area. This defect density is multiplied by a radial distribution modifier that is greater than one for the outside zones and less that 1 for the inside zones, resulting in a higher average number of defects for the outside zones, and lower average number of defects per quadrat for the inner areas. The constraint is that the sum of the probabilities for each quadrat (defect density) must equal unity to preserve the total average number of defects per wafer.

The radial distribution zones and quadrat grid are shown in Figure 8. The circular shape of the wafer is approximated in the simulation by a square area as shown by the outside box. Likewise, the radial zones are approximated with squares as shown by the inner box. Dashed lines represent the 144 quadrat boundaries. Once the number of defects have been determined for the quadrat area, the defects are placed randomly inside this area.

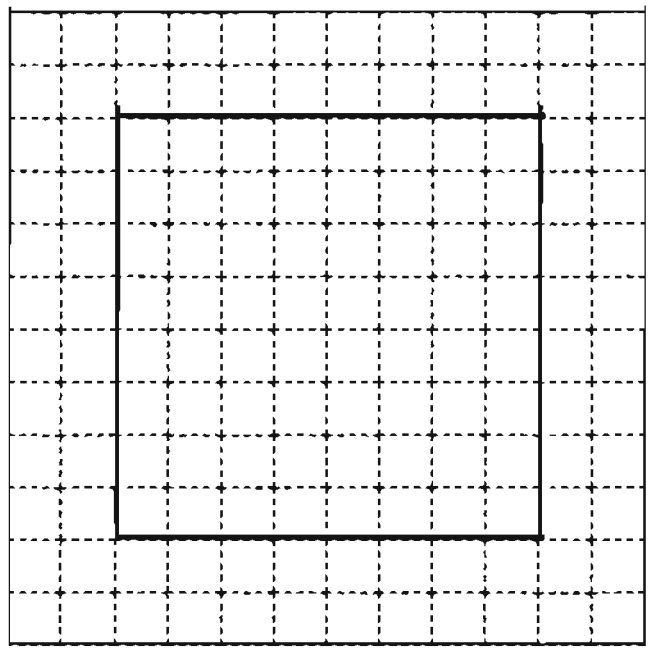


Figure 8 - Wafer radial zone and quadrat grid.

Physical Faults

The basic local defects in a wafer modify the operation of the circuit. Most of the defects manifest themselves as either opens in the signal path or shorts between signals or between signals and fixed sources. For example, extra polysilicon or metal can short two lines on the same layer, extra polysilicon can cause an open by forming a new transistor if over an active wire, extra material can cover a via or missing material can cause opens in signal lines. Walker discusses these defects in depth and their implications on circuit behavior.[Wal86a]. Another study examined 43 failed circuits in a 4-bit microprocessor chip. The results of the observed failure modes are shown in Figure 9[Gal80a]. The physical fault types are modeled using the logic fault models, which modify the logical operation of the network.

Short between metalizations	39%
Open metalization	14%
Short between diffusions	14%
Open between diffusions	6%
Short between metalization & substrate	2%
Unobservable	10%
Insignificant	15%

Figure 9 - Faults in a 4-bit microprocessor.

Logic Fault Models

One approximate model for physical defects is where signal paths are shorted or opened. This model is adequate when the actual physical layout of the wafer is known, so that it can be determined which signal paths are likely to be shorted together. The fault simulator works from a higher level description of the network, and the actual layout has not been developed yet. Therefore, the short/open model physical defect types are not used in the fault simulator. Instead a stuck-at model augmented with special fault models is used.

Stuck-At Model

As the level of integration increases, the stuck-at model becomes progressively less accurate[Gal80a]. There are two reasons for the inaccuracies. First, not all faults can be modeled using the stuck-at fault model. Some faults will actually change the function of the gates, and not always force it to a high or low state, and some faults create state-dependent behavior. For example, in Figure 10, if the transistor with input e is shorted so it is always on, whenever f is low, the path to GND is completed, forcing the output low. If e was low, the output is correct, otherwise the operation of the circuit is defective.

Second, a topology for the transistor circuit has to be assumed for the logic gate topology. Figure 10 shows a logic function and the transistor circuit to implement it. Faults are generally modeled using the logic schematic, but this does not always correspond to the transistor circuit which, in turn, does not always correspond to the lay-out. The X's shown in the logic schematic and transistor schematic indicate points that cannot be modeled in the corresponding schematic.

CMOS uses a complementary set of transistors for connecting the output to either to VDD or VCC. A fault in the path connecting the output to VDD will cause the output node never to be discharged via that path. Yet, a parallel path connecting the output to VDD will discharge the node, with seemingly correct operation. The fault is only detected when the faulty path is supposed to be activated, and the output is still high. The stuck-at model does not model this type of "intermittent" defective operation. By using the stuck-at model for "intermittent" nodes like this in the fault simulation, the

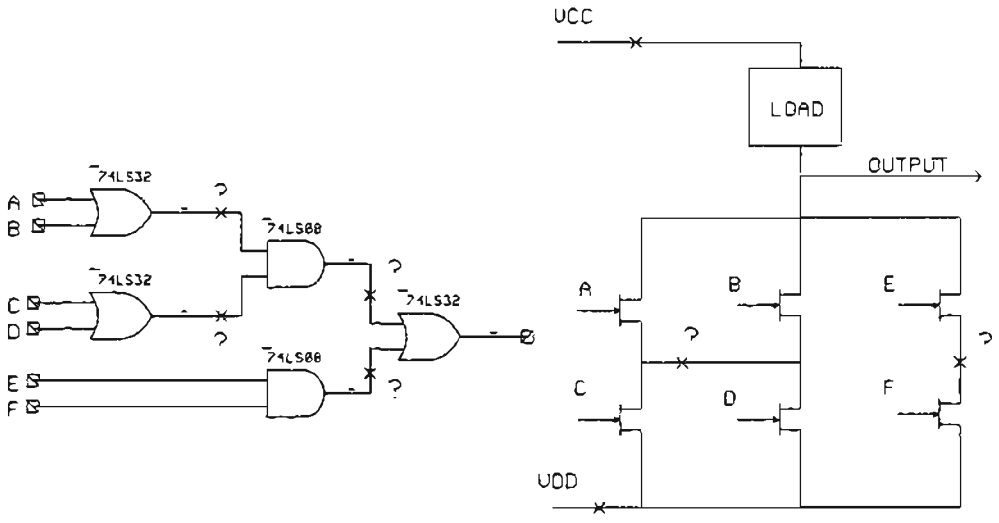


Figure 10 - Logic vs Electrical topology.

worst case operation for the node can be modeled. The model implies that the node is always stuck high or low, but it is actually stuck only for a subset of input combinations.

A tradeoff exists between the accuracy of the simulation and the amount of information about the physical architecture to be supplied. To improve simulation accuracy, more information is required. But, the purpose of the fault simulator is to model an architecture before the design is complete to ensure fault tolerance, therefore, many simplifying conservative assumptions were made.

The fault simulator will produce a first cut yield estimate for the wafers. Fault modeling is best done using a high level fault mechanism versus a more detailed model that is more accurate, but requires a more detailed architectural description and design. The accuracy of the model will be limited due to this abstracted architectural description.

The stuck-at model forces a signal to an always high (S-A-1) or always low (S-A-0) state, allowing single bits in data words to be faulted. A single physical defect will map to a hardware block in the wafer, where a stuck-at fault will be assigned. Usually a single physical defect will map to a single bit in a data and/or address field. But potentially, a single physical defect can cause multiple stuck bits. For example, multiple stuck bits occur when a data word is transmitted in portions using a multiplexed data bus with a defective signal line or a large defect can bridge several devices or wires. Each data word portion will have the same faulty bit position.

Physical defects can prevent data transfers between hardware blocks; these are usually defects in handshake or control signals. Faults can inhibit the transfer of the weight data from the learning state machine to the weight memory or data from one PN to

another PN site input. One method of modeling the inhibited data transfer is to fault all the bits in the data word. All the data lines would be stuck-at 0 or 1. The destination hardware block of the transfer would be updated with a new value that is either all high or all low. In reality, the inhibited data transfer will cause the destination node to keep its old value, ignoring the new value. The fault simulator uses a "NO CHANGE" flag which allows data transfers to be inhibited, forcing the destination to keep its old value.

5. THE FAULT SIMULATOR

Several of the goals for the fault simulator development were:

- Flexibility
- High level architecture description input
- Modular routines
- Efficient memory storage (for large networks)
- Efficient execution time (for large networks)
- Worst case fault model

The fault simulator provides a basic framework for modeling faults in a neural network wafer described by a high level architecture description. This architecture will gain more detail as the design progresses. The fault simulator can adapt to the changes without a major redesign of the concepts used in the simulator. As the architecture becomes more detailed, the model and fault simulation should become more accurate. This flexibility is accomplished by using a modular structure for the simulator routines. For example, a single routine calculates the size of the PN. As this calculation becomes more accurate, this routine can be modified to the new, more accurate model. The fault generation routine is another example of a modular routine. The original fault generation package placed faults using a completely random distribution. The second fault generation package, which incorporated fault clustering and radial distribution, required that only one routine be replaced.

The fault simulator will be used to analyze ultra-large-scale integrated silicon neural networks. Networks to be simulated at OGC will have 128 PNs, 4K CNs (16 CNs per PN), and 2 million connections. The BIF file to describe this network would require over 26 megabytes of data. Therefore, utilizing the memory required to run the fault simulation efficiently is a constraint. Usually as memory is conserved, the execution time is increased, which is another constraint. A proper balance of the required memory space and program execution time was needed. Minimizing the memory requirements is the more important constraint to allow simulation of larger networks.

Fltsim reduces the amount of memory and execution time by using a high level architecture description to model the hardware. For example, since the network is comprised of an array of similar PNs, only one set of hardware block sizes internal to the PN is calculated. Only the outside boundaries for the PNs are replicated for the entire wafer, and not all the internals for the wafers. Also, to reduce the amount of memory used, the entire mBIF file is not stored in memory, but only the connectivity of the n-graph is stored.

A fault field in the *fBIF* file consists of two numbers, a fault index and a fault modifier. The fault index indicates the type of fault (S-A-1, S-A-0 or NO CHANGE), and the type of target value to be faulted. The types of target values that can be faulted are a data word, a message address or both the address and data portions. The fault modifier indicates the specific bits to be modified in the targeted value. As an example, the index may indicate a S-A-0 fault in a data word. The modifier will contain 0's for the faulty bits that are stuck-at-0 and the rest are 1's. The target value is AND'ed with the modifier to give a new target value. The faulted bits are always low. The S-A-1

fault is modeled in the same way, except 1's are OR'ed with the target value. Another type of fault is the "NO CHANGE" fault, which forces the target value not to be updated with new values. NO CHANGE faults are implemented one of two ways. For output links, the message's address field is set to an invalid node in the network, causing the message to be lost. For input values, the new value is set to its previous value. For example, if a handshake line is faulty, message transfers between PNs will not occur, so that the target value is not updated. The update is inhibited by sending the message to an invalid node in the network. As such, the target value in the destination CN does not get updated and will keep its old value. NO CHANGE faults do not use the fault modifier field.

Faults in the hardware of the network can be modeled as faults in the n-graph model of the network. Since BIF describes this n-graph model, faults will be mapped to the n-graph and then to the BIF file.

In the n-graph, faults can manifest themselves in the CN output, the links between nodes, the weight fields, or the output of the site to the CN. Figure 11 shows the diagram of the n-graph with X's indicating where faults are to be modeled. To model n-graph faults, four fault fields are required in the BIF file, one for each area mentioned, the CN, Site, Link and Weight. These fields allow multiple faults to occur in different areas of a CN, but only one fault field per area is allowed, to reduce the complexity of the system. Faults that modify a common n-graph area are combined using a set of worst case fault rules. These rules combine, if possible, two faults to effect all faulty bits from both fault modifiers, otherwise they choose a fault that has more impact on the operation of the network. Faults will be combined if the fault fields of the indexes are

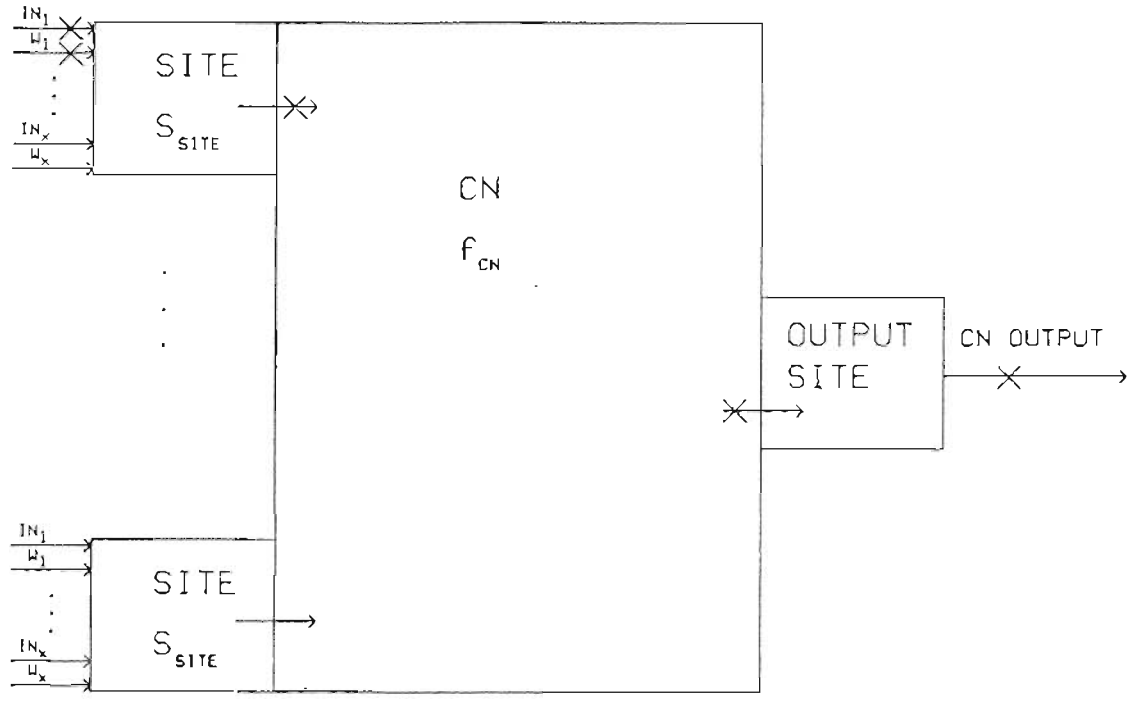


Figure 11 - Fault Locations in the n-graph.

equal, i.e., if both faults modify the same address and/or data portions of the target values. The rules for determining the worst fault are listed in order of precedence below:

1. "NO CHANGE" faults
2. Address fault over Data faults
3. Address and Data faults over Data faults
4. S-A-0 over S-A-1 faults and combine faulty bits
5. Combine faulty bits

"NO CHANGE" faults are assumed to be the worst type of fault, since they are the result of a control or handshake fault, which affects all the bits in the word. Faults modifying only the message address bus have more impact than faults that modify the data words. Faults modifying both the address and data are worse than faults that just modify the data. If the faults modify the same address and data values, the fault

modifiers are combined. If several faults are all S-A-1 (S-A-0) faults, the modifiers are combined, faulting all the stuck bits. If several faults have a combination of S-A-1 and S-A-0 faults, S-A-0 is assumed and the faulty bits are combined so that they will all be stuck at 0. The main idea is to provide a single worst case fault with only one field. So, although the type of fault is changed, the impact on the network will be the worst case.

Assumptions

Some hardware and fault modeling assumptions had to be made when developing the fault simulator. These assumptions were needed to simplify the design of the fault simulator and also because the hardware architecture is not yet completely defined. As the hardware becomes better defined, it can be modeled more accurately. Assumptions relating to how the physical hardware faults are modeled and how they affect the operation of the network will be reassessed later.

The assumptions can be divided into two categories. The first category of assumptions simplified the hardware model and the second category simplifies the modeling or representation of the faults in the network. Each impacts the modeling of the network and is now described in more detail.

The connectivity for the PTP connections was assumed by the PN network locations in the network as specified in the PAD file. Although the PTP connectivity is explicitly stated in the PAD file for the PTP connections, a simplification was to use the PN x,y location in the network and assume physically adjacent PNs in the wafer are connected by a PTP bus. Minimal area is used when connecting adjacent PNs and thus will be the most common configuration. The networks currently planned to be modeled at

OGC will connect adjacent PNs.

The message routing algorithm for the PTP communication is assumed. A message sent from one PN to another will travel in the x direction first until the correct column is reached, then in the y direction until the destination PN is reached.

The PAD file specifies the number of LSMs in each PN. If one of the LSMs has a defect, all the weights updated by the defective LSM will be faulted. The assumption that each LSM has an equal number of weights is assumed. So one defective LSM will cause 1/(# LSMs) of the weights to be faulted.

The structure of both the PBH Transmitter bus and the Receiver bus was assumed to be a binary tree. Concentrators have two input links from lower bus levels and one output link to a higher level. Transmitters have one input link from a higher bus level and two output links to lower levels in the bus.

PBH regions must have a common structure. That is, each region must contain the same number of PNs and each level the same number of data and control signals, which reflects the assumed symmetry in the n-graphs to be emulated.

The assignment of the CNs, Sites, Links and Weights to specific locations within a PN was assumed to be in the order listed in the BIF file. The BIF file uses a hierarchical notation to list the n-graph sections; a CN section is first, then all the Sites for the CN, followed by all the Links for each Site. Thus, all the information is grouped in order in the BIF file, and will therefore be adjacent in the PN hardware.

The second category of assumptions concerned fault modeling. The defects modeled on the wafer are point faults with zero diameter. Actual faults have a non-zero variable diameter size. If a defect is located on a signal line, the defect diameter and the

line width will determine if the line is completely severed or just partially nicked. If the defect is between two metal runs, the defect diameter and the line spacing will determine if the two lines are shorted together. The defect size, line width and line spacing are not modeled explicitly in the fault simulator.

Faults are modeled as single bit faults, such as defects in a single data bit in a memory, a single DAC or ADC output or a single data buffer. These single bit faults may affect several bits depending on how the hardware is used, but only one bit is defective in the hardware block per physical defect. For example, the RAM structures may have defects in the row or column address decoding, disabling a whole set of bits for the entire PN. These defects affect the control structure of the individual hardware blocks. A future enhancement would be to add probabilities of faults in the control structures for individual hardware blocks that would affect sets of bits. For the LSM, a probability has been defined by catastrophic faults in the LSM. This concept could be expanded for other hardware blocks.

Each hardware block contains the circuitry to perform the indicated function. The area inside a block does not include any free area. Defects that occur inside a hardware block will affect the function of the block. No allowance for free areas between the lines and devices is made. This free area can be compensated for by decreasing the defect density.

The fBIF file conveys the fault actions to be performed in the architecture simulators. Only one fault field was allowed per n-graph section. Either multiple faults that affect a common n-graph section are combined, or the worst case fault is used. The rules for choosing the worst case fault are assumed to produce a fairly accurate model of the

real system.

Basic Fault Simulator Processes

Figure 12 shows the basic execution flow of the fault simulator. Fltsim starts by

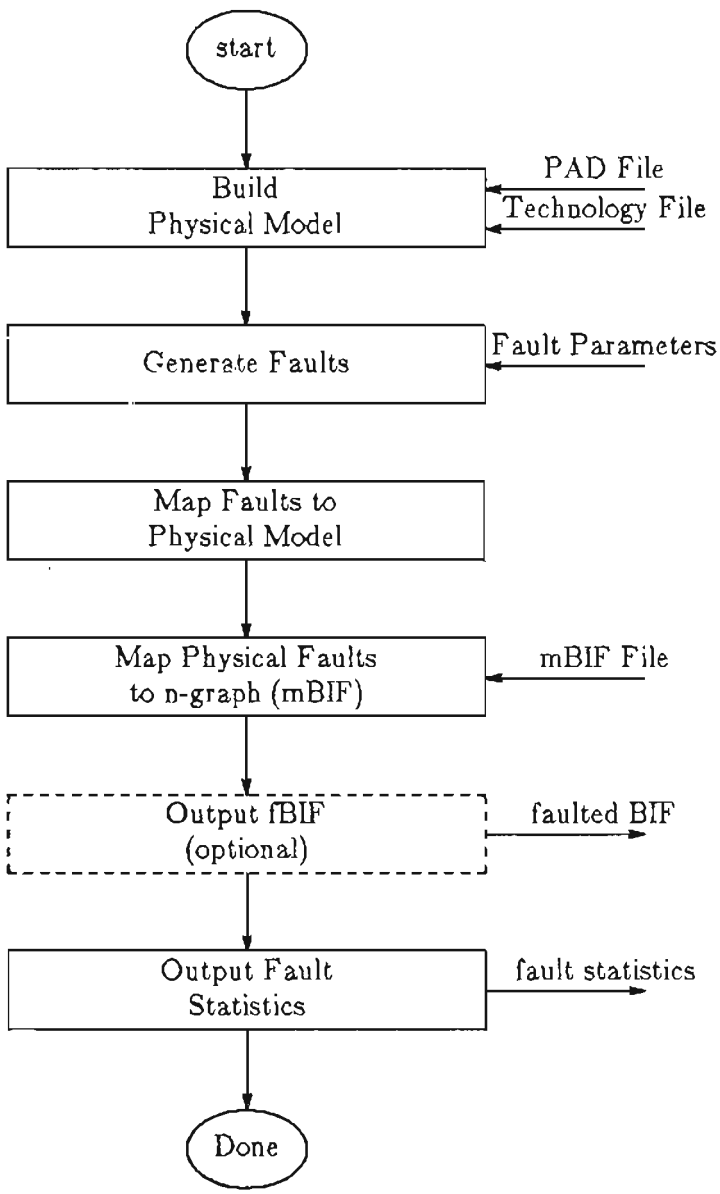


Figure 12 - Fltsim Processes.

building a physical model of the circuitry. Size information read from a technology file and a physical architecture description, PAD, file are used to construct the model. The fault model parameters are obtained from a fault parameter file that describes the characteristics of the faults. Stuck-at-1 and Stuck-at-0 faults are generated and placed on the wafer using an x,y coordinate system. These defects are then mapped into the physical hardware blocks in the network. The impact on the operation of the p-graph network is assessed to determine how the n-graph is affected by the faults. Faults in the p-graph are mapped to the n-graph. Knowing the effects of the faults in the n-graph, the fBIF file can be created including a fault index and fault modifier to modify the n-graph operation. These fault fields will indicate to the architecture simulators, HAS and ANNE, how to model the faulted network. Fault statistics are generated to provide the needed feedback to evaluate the new operation of the network simulation by HAS or ANNE. The following sections will describe each Fltsim execution phase in more detail.

Physical Model

The fault simulator builds a block representation of the circuitry to be modeled. Each PN's internal circuitry is identical, each PTP bus is identical, and each PBH bus communication region is assumed to be identical. Therefore, to drastically reduce the number of calculations and stored information, only one PN hardware block representation and one PTP and PBH bus representation are calculated. The technology and PAD files contain the required information to build these representations.

The PN is modeled as a rectangular area containing the hardware blocks. An aspect ratio determines the x and y dimensions for the PN. Given the PN x,y dimensions, each PN in the network is assigned a physical location on the wafer. The PNs are

separated on each side by a bus communication area, which is calculated from the bus line width and the number of bus lines. Note that the locations of the hardware blocks within a PN are not assigned. The locations of the PNs, PN hardware blocks and bus areas are discussed further in the placement of the faults in the hardware.

The technology file contains information describing the sizes of the basic elements used in the blocks. The sizes are dependent on the technology used to fabricate the device. It contains fields that describe line widths, sizes of memory cells, DAC cells, ADC cells and buffers. Basic element sizes are multiplied by the hardware block dimensions given in the PAD file. Not only will the basic element size indicate the space required for its basic function, but it will also contain an added amount for the control of the function. For example, a memory cell can be implemented using a fixed amount of area. Also included in the memory cell area is an amount for the row and column buffers and address decoding that will be part of that hardware block. Some basic elements can be thought of as a bit slice processor, where sections of components are added to expand its capabilities. The LSM is an example. As each LSM is added, a new separate structure is added to the existing circuit, expanding the LSM's capabilities. Expanding the LSM's capabilities allows additional learning algorithms to be used.

The PAD file describes the p-graph for the network, contains information used to organize the basic elements into the hardware block sizes, and indicates the interconnectivity of the network. The PAD file describes the PNs and CNs in the network, the layout of the PNs in the network, the PTP communication structure and the PBH communication structure. Information regarding the dimensions of the blocks in a PN, such as the number of bits in the weight field and the maximum number of CNs in a PN is

included in the PAD file. Figure 13 shows the dimensions of the blocks in a PN. The actual values are given in the PAD file.

The amount of silicon area covered by the PTP and PBH bus structures between the PNs is calculated. The PTP uses a simple grid network where the bus area between two PNs is the product of the number of bus lines, the line width, and the distance between the PNs. The total bus area between PNs is multiplied by the number of PTP buses between PNs.

The PBH bus size calculation is more involved, since it uses a tree structure communication path to broadcast messages to several PNs simultaneously. To model faults in the PBH transmitter bus, the bus area for each level of both the PBH transmitter bus and the receiver bus must be calculated, as faults in various levels affect the network differently. Bus signal lengths increase towards the top concentrator and deconcentrator nodes in the PBH network. Every second level, while ascending the PBH transmitter bus, the bus increases in length exponentially. If we assume the x and y dimensions are the same for the PN and level 0 has length 1, then level 1 will have length 1, level 2 length 2, level 3 length 2, and level 4 length 4. If the PN dimensions are pn_x and pn_y, and pn_sep is the distance between the PNs, Figure 14 shows a table of the bus lengths for increasing levels. The relative bus lengths are shown in Figure 15. Dark lines represent the PBH buses and the squares are the PNs array in the network. Bus faults are more likely to occur in the upper levels of the PBH network due to the longer bus lines and bigger buffers required to drive the longer lines. Worse yet, these upper level faults corrupt a higher percentage of the PN messages in that PBH region.

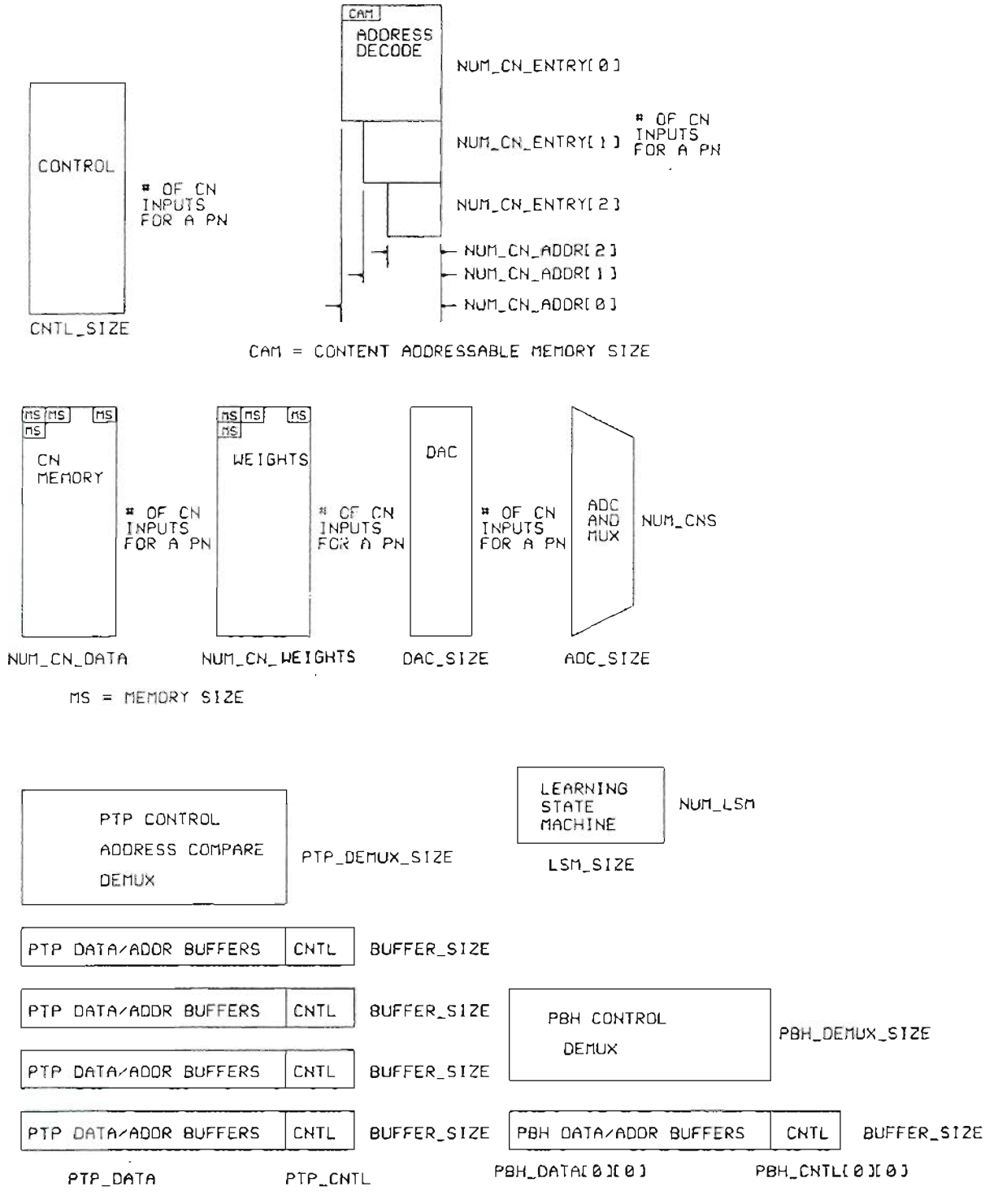


Figure 13 - PN Block Sizes.

Level	Length
0	pn_sep
1	pn_sep
2	2*(pn_x + pn_sep)
3	2*(pn_y + pn_sep)
4	4*(pn_x + pn_sep)
5	4*(pn_y + pn_sep)
6	8*(pn_x + pn_sep)

Figure 14 - Exponential PBH bus length.

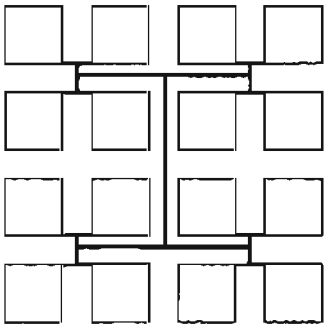


Figure 15 - PBH Bus Lengths.

Fault Generation

The fault generator produces a list of defects using the fault models discussed earlier. The fault parameters used to calculate the fault locations and types are read from a fault parameter file. Parameters can be varied quickly, without recompiling Fltsim, to determine how the fault parameters affect the performance of the network. Of primary interest is how the fault density and fault clustering affect the operation of the networks. The fault parameter file will specify the fault density for a network. The fault density is multiplied by the size of the network to determine the average number of defects to place in the network. The number of defects in the network is divided by the number of quadrats to determine the base average number of quadrat faults. The base number is adjusted for inner and outer zones to model the radial distribution.

An array of normalized fault location coordinates is generated, i.e., each coordinate ranges from 0 to 1. The normalized coordinate is multiplied by the network overall dimensions to get the actual physical x,y fault coordinates. With each fault the fault generator associates the fault type, S-A-1 or S-A-0. The ratio of the fault types is specified in the fault parameters file.

Mapping Faults to Physical Model

The physical coordinates for each fault are used to determine which hardware section contains the fault. The fault can occur in a PN or the bus area between the PNs. If the fault is in a PN, the hardware block that the fault is in is determined statistically as a random number with uniform distribution. The probability that the fault is in a block is given by:

$$Pr(block) = \frac{block_{area}}{PN_{total\ area}}$$

Thus, on the large scale, faults use the wafer-scale fault model characteristics, but within the PNs, the faults are placed according to the block sizes. Once the fault is isolated to a hardware block, the fault is mapped to specific CNs within the PN which determines how the CNs are affected.

If the fault is located between PNs, it is modeled in the PTP or PBH bus structure or the unused area. Faults in the unused area do not have any impact on the network. The bus structure that faulted is determined, along with a faulty bus segment within the bus structure. The faulty bus segment determines which PN messages to corrupt. Bus segments include the PTP bus between two adjacent PNs, the PBH bus between a PN

and the concentrator/deconcentrator nodes or the bus between concentrator/deconcentrator nodes. A uniformly distributed random number is generated to determine in which bus and bus portion to place the fault. The probability of a fault in a bus is given by:

$$Pr(bus\ segment) = \frac{area\ of\ the\ bus\ segment}{total\ area\ between\ PNs},$$

where neither the PTP or PBH bus may be affected if the fault occurs in an area with no bus lines. For a PTP fault, the faulty segment indicates on which side of the PN the fault occurred. Four PTP buses are associated with each PN, one on each side. For the PBH transmitter and receiver buses, the level in the PBH tree is indicated along with the closest PN to the fault. The closest PN to the fault indicates which PBH region that contains the fault. If the PN is in overlapping PBH regions, one of the PBH regions is chosen, with equal likelihood, to contain the fault. The faulty PBH level and closest PN indicate which part of the PBH subtree is affected. The bus area required for each PBH level for each transmitter and receiver bus determines the probability of a fault occurring in that bus portion.

The buses that connect the PNs together are the most critical area to model in this architecture. While faults in a PN will generally cause a single CN or PN to fail, a fault in the bus area will cause several PNs to receive faulted data information. This is especially true for PBH structures where a message is sent to several PNs simultaneously. The bus signal area itself is not the critical factor for faults, as buses can be expanded to reduce bridging and open bus faults. What is more likely to fail are active devices[Lei85a]. Bus wires only require a few masking steps, versus active devices which

require many more. For the MIT Lincoln Laboratories project[Raf85a], yields were predicted at 30 to 50 percent for cells and 95 percent for wires. Each PBH concentrator node and deconcentrator node contain buffers to drive the bus line to the next node. Also, in the concentrator nodes some form of contention avoidance circuitry is used to avoid bus collisions, which adds more circuitry and area. Larger bus buffers towards the top nodes in the tree will be required to drive the longer bus lines. These concentrator/deconcentrator nodes are more susceptible to faults than just simple bus lines. A PBH region connects several nonadjacent PNs. Bus line lengths increase by $O(2^n)$ for every second level. It does not take many levels to make this bus circuitry large and susceptible to faults.

Mapping faults to the n-graph

A mBIF description is read which describes the n-graph. Subsections in the mBIF file describe each CN, each CN site, and all site links and weights. As each subsection of the mBIF file is read, the list of physical faults is checked for the faults pertaining that mBIF subsection. If multiple faults are found to affect a common subsection, the faults are combined into one set of fault fields to model the worst case operation of the network. Each mBIF subsection is read, checked for faults, and the fault fields written before proceeding to the next subsection to reduce the memory requirements of large networks. Figure 16 shows a partial mapping of hardware block faults to n-graph areas affected. For example, each link input value is stored in the CN MEMORY hardware block. If a CN MEMORY word is faulty, the corresponding link input to the CN will have a S-A fault modeled in the input message. More detail on how the faults are mapped is provided in Appendix D.

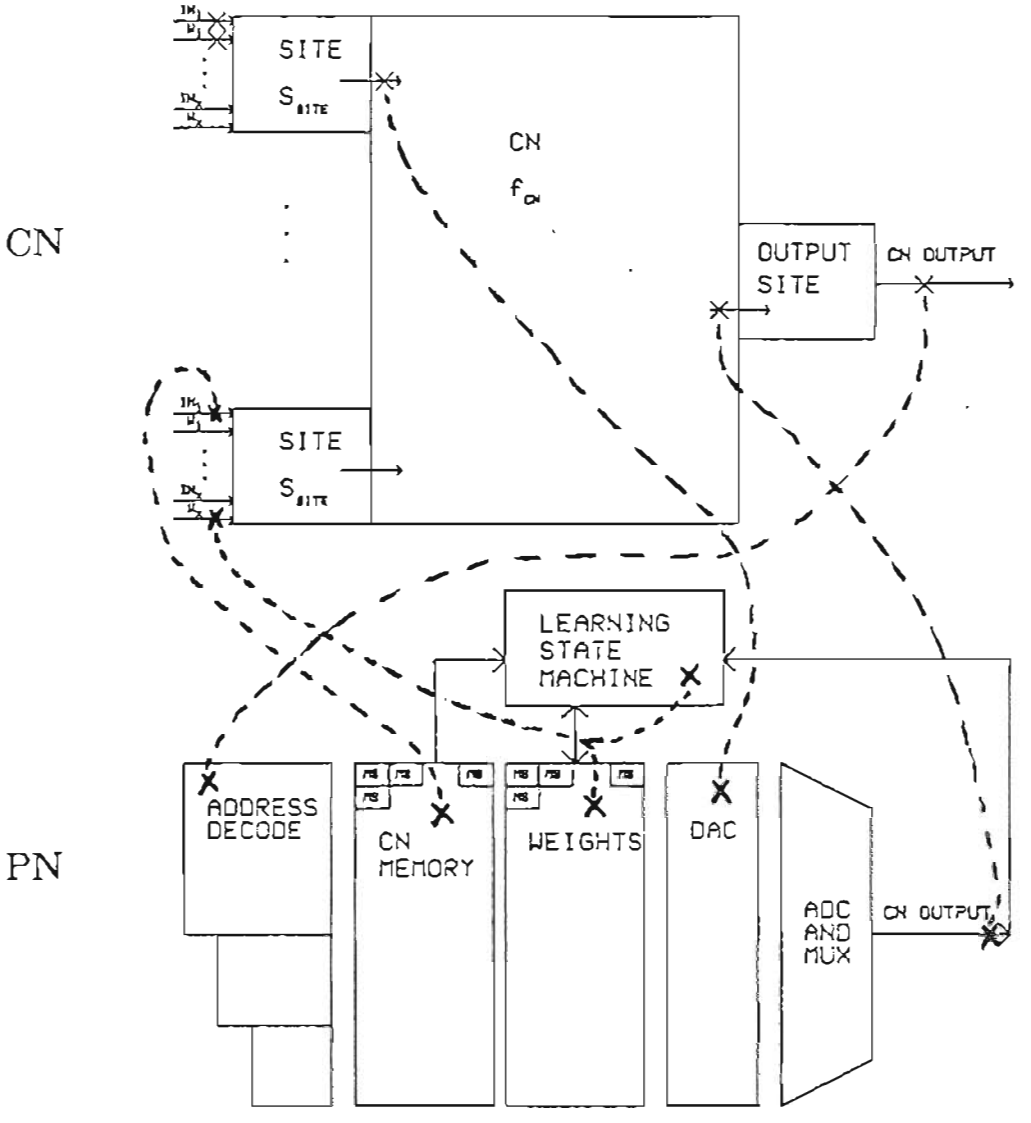


Figure 16 - Hardware fault to n-graph mapping.

The mBIF file describes the connections between CNs in the link subsection. This connection can be via the PTP or PBH communication networks, as indicated in the link subsection. The entire communication path that implements this connection must be checked for faults. If PTP communication is used, the PNs send a message that is potentially routed through several PNs in a grid network. If PBH is used, messages are routed through intermediate nodes. If a fault has occurred in any part of the communication path, the message will be corrupted or even lost. Corrupted or lost messages correspond

to the actual physical results of a defect. Therefore, the entire message path is checked for faults for each mBIF link connection.

Another consideration for modeling faults in the PBH network is whether to modify the source PN sending the message or the destination PN receiving the message. If a defect occurs in the PBH transmitter tree link, all PNs sending messages in that subtree will have their messages corrupted. PNs not using this faulty link can send messages that do not get corrupted. To model these corrupted messages, the output links for the PNs in the faulty subtree will be faulted. Defects that occur in a PBH receiver bus link for a given level will affect all messages sent to the PNs using that link. All PBH messages to PNs in a faulty subtree portion will be corrupted, while the PNs not in that subtree will receive the message uncorrupted. Therefore, receiver bus faults are modeled on the input links to the CNs.

Both the PTP and PBH messages are sent over multiplexed buses. The message is divided into subwords that are transferred over the bus. A defective data bus line will cause each subword sent over the bus to have the same fault, faulting bits in both the address and data fields of the message. Thus, one faulty signal line will cause a faulty bit in each subword. If the data and address are multiplexed using only one signal line and it is faulted, all the address and data bits in the message will be corrupted.

Output Fault Data

Two fault files are produced by Fltsim, *fBIF* and *fstat*. The fBIF file contains a section of C program code that is included by the architecture simulators to initialize an array of fault indexes and modifiers. Three other arrays are also included in the *fBIF* file

and are used to access the fault array by the architecture simulators. The modified operation of the network is evaluated using the architecture simulators. At various points in the network simulation, fault routines are called that modify intermediate values in each CN. For example, the output of a site function may be faulty. A site function routine calculates the site function output and passes it to a fault routine, which accesses the *fBIF* fault fields to potentially corrupt the site output. The fault routines modify the CN values using one of the logic fault models presented here, S-A-0, S-A-1 or "NO CHANGE".

The *fstat* file is the fault statistics file, which lists the defects and how the n-graph was modified along with a summary of the faults. The *fstat* summary includes the percentage of faulted CNs, links, sites, and weights in the network. More detail on the contents of the *fstat* file is in Appendix B, Fault Simulator File Formats. Understanding and predicting the network's performance can be done using the fault statistics.

A third file, *test*, can be generated which provides more detailed information on intermediate calculations used in the fault simulation. Sizes of the various hardware blocks and bus structures and actual fault locations are examples of the information contained in this file.

6. SIMULATION RESULTS

The fault simulator executes several basic processes to model faults in the neural network. (These processes were introduced in Chapter 5.) The results of each intermediate process are presented in this chapter.

The network discussed in this chapter is a feed forward 128 x 128 neural network. Three layers of CNs are present in the network, each with 128 nodes, for a total of 384 CNs. Feed forward implies all messages from a layer are sent to a higher layer. Each CN has one input site and one output site which receives/sends messages, for a total of 384 input sites and 384 output sites. Since no faults are modeled in the output sites, they are not included in the fault statistics.

Each CN in the first layer has one input link that receives the input to the network. Also, each CN in the first layer has an output link with each CN in the second layer, which accounts for 128^2 or 16,384 links. Likewise, each CN in the second layer has an output link to each CN in the third layer. Each CN in the third layer has one output link, which is the output for the network. A total of 32,896 links are present in the network. Since faults are modeled separately on the input and output links, they are counted separately for the fault statistics, and are referred to as IN LINKS and OUT LINKS. For this experiment, the simulated network used only PTP communication to implement the links. The PBH communication was not used.

The CNs are mapped to 8 PNs in the network. Each PN contains 48 CNs. Four of the PNs have 6144 input links, one has 6017 input links, one has 2207 and two have 48 input links. Varying quantities of input links is due to the first CN layer having fewer inputs.

The fault simulator first builds a physical model of the hardware to be faulted. The technology file and PAD file determine the required area for the various hardware blocks. A 1μ CMOS process is used to implement the network. As no actual hardware has been designed, estimated sizes, some from the advanced VLSI class project designs, (which are scaled to 1.25μ), are used in this thesis.

The sizes calculated by Fltsim for the network are shown in Figure 17. Preliminary size calculations show that the largest PN hardware block is the DAC, covering 90% of the PN area, and second largest is the PN control section, covering 6% of the

Section	Area (Square microns)	Percent
ADC	4800000	0.70
DAC	614400000	90.22
PN CONTROL	40960000	6.01
MEMORY	6553600	0.96
WEIGHT	6553600	0.96
LSM	1600000	0.23
ADDRESS DECODER	6144000	0.90
PTP_DEMUX	100	0.00
(1 of 4) PTP_CNTL	100	0.00
(1 of 4) PTP_BUFFER	100	0.00
PBH_DEMUX	0	0.00
PBH_CNTL	0	0.00
PBH_BUFFER	0	0.00
total PN	681012100	100

Figure 17 - PN block sizes with DAC = 75000.

	D0		P0	
Section	Area (Sq. microns)	Percent	Area	Percent
ADC	4800000	7.21	4800000	0.75
DAC	0	0.00	614400000	95.99
PN CONTROL	40960000	61.49	40960000	0.00
MEMORY	6553600	9.84	6553600	1.02
WEIGHT	6553600	9.84	6553600	1.02
LSM	1600000	2.40	1600000	0.25
ADDRESS DECODER	6144000	9.22	6144000	0.96
PTP_DEMUX	100	0.00	100	0.00
(1 of 4) PTP_CNTL	100	0.00	100	0.00
(1 of 4) PTP_BUFFER	100	0.00	100	0.00
PBH_DEMUX	0	0.00	0	0.00
PBH_CNTL	0	0.00	0	0.00
PBH_BUFFER	0	0.00	0	0.00
total PN	66612100	100	640052100	100

Figure 18 - PN block sizes with DAC = 0 and PN CONTROL = 0.

PN area. The remainder of the hardware block areas are insignificant. The area required by the DAC and PN CONTROL circuits were determined to be too large and would skew the results, so two additional sets of simulations were run, one with the DAC size set to 0 and one with the PN CONTROL set to 0. These additional simulations help determine how the DAC or PN CONTROL sizes effect the fault tolerance of the network. Figure 18 shows the resulting sizes of the networks. Although it is unreasonable to assume these two areas can be eliminated completely in the circuit, stricter design rules and redundancy can be used to effectively reduce the chance of faulty operation. The sizes of the hardware blocks will be further studied as either the inputs to calculate these sizes are inaccurate, or the design should be changed to reduce this area or increase the fault tolerance of the block. The results from these architectures will have skewed results until more accurate size information is available. For the purpose of this thesis, these sizes will be assumed to be correct. (The focus here is on the simulation tool

and not the specific architecture.)

To obtain a statistical sampling, the simulation was run 5 times with the original DAC and PN CONTROL sizes and 5 times with the DAC set to 0 square microns and 12 times with the PN CONTROL set to 0 square microns. The results of each set of simulations are compared in this chapter. The network with the original DAC and PN CONTROL sizes will be referred to as D75, the network with the 0 DAC size will be referred to as D0, and the network with the PN CONTROL size of 0 will be referred to as P0.

Figure 19 shows a completely random defect distribution on a wafer, where a "1" indicates a S-A-1 fault and a "0" indicates a S-A-0 fault. Compare this random distribution with a more accurate model using fault clustering and radial distribution characteristics shown in Figure 20. The input parameters for generating the fault distribution shown in Figure 20 specified a defect density of 15 defects per square inch, the percentage of S-A-0 faults is 30%, the clustering coefficient is 0.49, and the inner to outer zone fault density ratio is 1.0. These values are typical values. Fault densities observed in industry range from about 15 defects per square inch to about 35 defects per square inch. For the purposes of this simulation, the lower bound was chosen. The percentage of S-A-0 faults was chosen arbitrarily for these first simulations. This percentage does not have a major impact on the operation of the faulted network, as a fault will be modeled in the network regardless whether it is a S-A-0 or a S-A-1 fault. The fault clustering coefficient of 0.49 produces less of an even distribution of faults and is from Stapper's paper on fault clustering[Sta86a]. The first architectures that are being modeled do not cover an entire wafer, but represent a large die on a wafer. Since the radial distribution

model only accounts for the distribution of faults for an entire wafer, and not for individual die on a wafer, the radial distribution is not taken into account here by setting the inner and outer fault densities equal. The resulting fault distributions from the fault generator of the fault simulator correspond to the figures shown in Stapper's paper on fault distributions[Sta86a].

The calculated size of the neural network was 8.45 square inches for the D75 network, 0.83 square inches for the D0 network and 7.94 square inches for P0. With a fault density of 15 defects per square inch, an average of 126 faults should occur in the D75 network, 12.4 faults in the D0 network and 119 faults in P0. For the D75 network, the average number of physical faults was 123 with a range of 99 to 151. For D0, the average number of faults was 11.2 with a range of 5 to 18 faults. For P0, the average number of faults was 123.4 with a range of 101 to 142 faults. Removing the DAC circuitry reduces the total amount of silicon area considerably, thereby reducing the number of faults present in the D0 network, whereas removing the PN CONTROL has a smaller effect on the network size and number of faults. Due to the randomness of the fault simulation, a wide range of faults occur in the network. The actual number of faults varies from the predicted values, but they are reasonably close. The average number of faults for P0 is greater than for D75. This increase is due to the randomness in the simulation. More simulations should increase the average for D75 and reduce the average for P0.

As expected, for each network, there is a correlation between the relative hardware block size and the percentage of faults present in the block. Consequently, the majority of PN faults generated are either DAC or PN CONTROL faults. Figure 21 summarizes

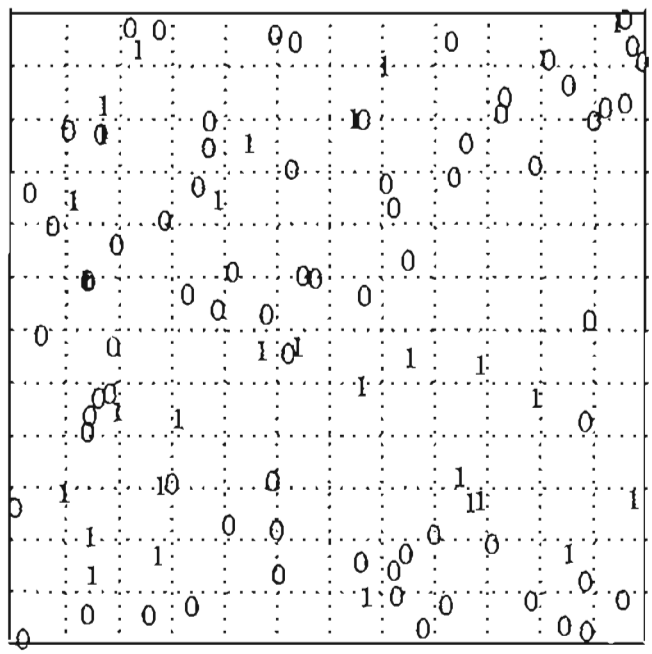


Figure 19 - Random Distribution of 100 faults.

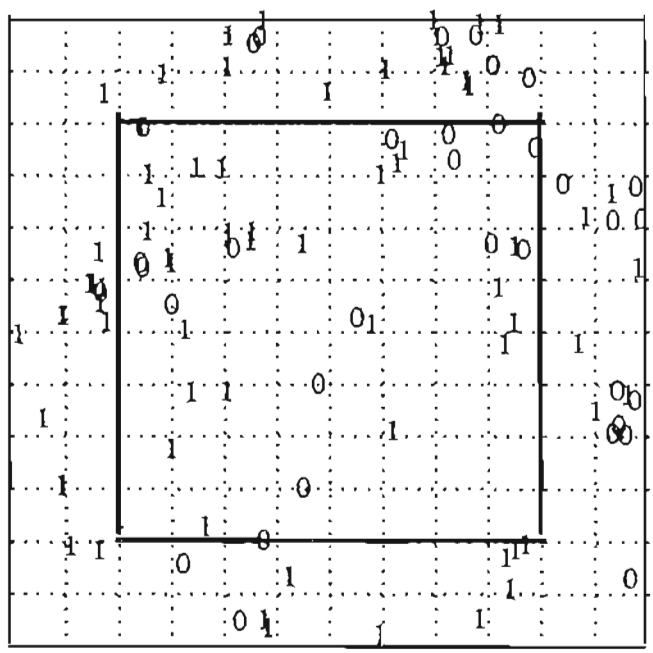


Figure 20 - Fault Simulator Fault Distribution.

the average number of faults that occurred in each hardware block for each of the simulations. These fault rates cn be compared to the hardware block sizes shown in Figures 17 and 18.

The impact of the faults on the p-graph is mapped to the n-graph. The physical faults generated an average of 25144.2 faults in the n-graph for the D75 network, 17702.8 faults for the D0 network and 1152.7 faults for the P0 network. Figure 22 shows a n-graph fault summary, indicating the number of entities in each n-graph section, the average number of entities that were faulted, the average percentage that were faulted and the range of faults that occurred in the n-graph section for the simulations. The sections consist of CNs, IN SITES, IN LINKS, OUT LINKS and WEIGHTS, which refer to the specific areas in the n-graph where the fault impacts are modeled.

	D75		D0		P0	
Section	Faults	Percent	Faults	Percent	Faults	Percent
ADC	1.2	0.98	0	0.00	0.58	0.47
DAC	107.8	87.64	0	0.00	119.58	96.89
PN CONTROL	8.4	6.83	7.2	64.29	0	0.00
MEMORY	2	1.63	1.2	10.71	0.67	0.54
WEIGHT	1	0.81	1.2	10.71	1.08	0.88
LSM	0.6	0.49	0.2	1.79	0.25	0.20
ADDRESS DECODER	1.8	1.46	1.4	12.50	1.25	1.01
PTP_DEMUX	0	0.00	0	0.00	0	0.00
(1 of 4) PTP_CNTL	0	0.00	0	0.00	0	0.00
(1 of 4) PTP_BUFFER	0	0.00	0	0.00	0	0.00
PBH_DEMUX	0	0.00	0	0.00	0	0.00
PBH_CNTL	0	0.00	0	0.00	0	0.00
PBH_BUFFER	0	0.00	0	0.00	0	0.00
total PN	123.0	100	11.2	100	123.42	100

Figure 21 - Hardware block faults.

D75						
Section	Number	Faulted	Percent	Range		
CN	384	0.8	0.21	0	-	2
IN SITES	384	92.4	24.06	76	-	111
IN LINKS	32896	1.0	0.00	1	-	1
OUT LINKS	32896	22582.4	68.65	14416	-	32656
WEIGHTS	32896	2467.6	7.50	1	-	6144

D0						
Section	Number	Faulted	Percent	Range		
CN	384	0.00	0.00	0	-	0
IN SITES	384	0.00	0.00	0	-	0
IN LINKS	32896	0.6	0.00	0	-	1
OUT LINKS	32896	16473.0	50.08	6446	-	26752
WEIGHTS	32896	1229.2	3.74	0	-	6144

P0						
Section	Number	Faulted	Percent	Range		
CN	384	0.4	0.11	0	-	2
IN SITES	384	102.1	26.58	90	-	111
IN LINKS	32896	0.3	0.00	0	-	2
OUT LINKS	32896	21.3	0.07	0	-	96
WEIGHTS	32896	1028.58	3.1	0	-	6144

Figure 22 - Fault statistics summary.

The number of faults modeled in the CN, IN LINKS and WEIGHTS were consistent for all three networks. These n-graph sections do not depend upon the size of the DAC or PN CONTROL hardware blocks. The number of faults in the CN and IN LINKS was low, which indicates that these sections should be reliable. The WEIGHTS section had a large average number of faults with a wide variation of faults between simulations. Some simulations had a large number of faulted weights and some did not have any weights faulted. This wide range of faults is due to the random quantity and placement of the faults in the network and how the network is modeled. For example, a

single fault may occur in the weight memory, faulting a single weight, or it may occur in a LSM, affecting all the weights in a PN. LSM defects fault all the weights that the LSM updates, which, if there are only a few LSMs in a PN, will be a large number of weights.

The number of faults modeled in the IN SITES was dependent on the size of the DAC. (Faults in the DAC fault the input SITE calculation.) Thus, the D75 and P0 networks had more of the input sites faulted than the D0 network. The DAC will be a critical section to make fault tolerant to ensure the integrity of the input site functions.

The number of faults modeled in the OUT LINKS was dependent on the size of the PN CONTROL. The large number of faulty output links is due to the large PN control hardware block. Defects in the PN control inhibit all the CNs in the PN, impacting a large section of the network, resulting in a high percentage of faulty output links. The size of the PN control will need to be examined to determine if this calculated size accurately models the function expected by the fault simulator, e.g., if a defect in the PN control section will not impact all the CNs in the PN, then it should be modeled in one of the other hardware blocks.

Due to the large fraction of output links faulted, the PN CONTROL section will be the most critical area to make fault tolerant. With the fault tolerance of this area increased, as implied by the P0 network, the number of OUT LINK fault becomes acceptable. A wide variation of faulted LINKS occurred for the D75 and D0 network. As with the WEIGHTS, this variation is due to the random quantity and placement of the faults. Also, for the output links, recall that the number of links in a PN varied from 48 to 6144. PN CONTROL faults in different PNs will fault varying quantities of OUT LINKS, which adds to the wide range of OUT LINKS faults.

A consistently large number of faulty n-graph sections or a wide variation in the number of faulty n-graph sections indicates that the hardware block contains critical logic. Using Fltsim, the DAC, PN CONTROL and LSM hardware blocks have been identified as containing critical logic for the current network. Either the areas required to implement these functions should be decreased or the amount of redundancy increased to alleviate these problems.

For the preliminary networks simulated, there was no fault interaction; the number of combined faults for all the simulations was zero. The lack of faults being combined can be attributed to several factors. The foremost reason is the relative area of the bus structures is much smaller than the size of the PNs. Faults in the bus areas are most common faults to be combined, but due to the relatively small size of the bus, few faults occur in the bus. Also, the faults that occur in the PN CONTROL are modeled as NO CHANGE faults, which are not combined with other faults.

Fault clustering does have an impact on the operation of the network. Figure 23 shows a list of each PN and the number of faults occurring in the PN. PNs 3 and 4 have a minimal number of defects and have the greatest probability of operating normally. On the other hand, fault clustering caused PNs 0 and 2 to have a higher quantity of faults, resulting in a greater chance that those PNs will be unoperational. The actual fault impact would need to be examined in each case, as any fault may disable the entire PN.

Execution times for Fltsim are dependent upon the physical size of the network and the number of CNs, sites and links in the n-graph. As the size of the network increases, more faults are generated that require a longer search time for the n-graph section. Also,

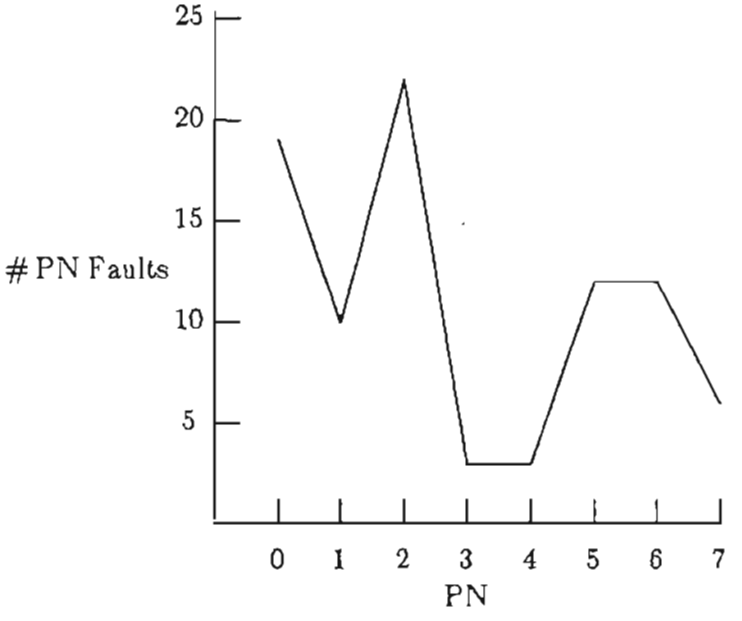


Figure 23 - Fault clustering in the PNs.

as the number of CNs, sites, and links in the n-graph increase, the greater the number of sections to check for faults.

Execution times were measured for Fltsim modeling a 12 PN, 1 CN per PN network and the 128 x 128 network, (with 8 PNs and 364 CNs). For the 12 PN network, the fault simulator with no test output executed in 8 seconds on a VAX 11/780 with 2.4 seconds of user time and 0.6 seconds of system execution time. Generating the test output increased the user time to 3.1 seconds, while the other times remained constant. For the 128 x 128 network, the execution time was 30 to 40 minutes. The total amount of area available on a 4 inch wafer is 12.5 sq. inches. The D75 network used 8.45 sq. inches, which is 68% of the total amount available. Using a similar network architecture, a full wafer could be simulated in about 1.5 hours on the VAX. Running Fltsim on a more powerful computer, such as a Sun 4, will reduce the time to 20 to 40 minutes. Larger wafers can then be simulated on the Sun computer.

Verification of the fault simulator is complicated by the size and complexity of the network being modeled. The only way to truly verify the correct modeling of process faults and their impact on the neural network architecture is to fabricate several wafers, identify physical fault locations and examine the faulted network's operation. Comparing the actual data collected to the fault statistics produced by the fault simulator and the operation of the architecture simulators would determine the accuracy of the fault simulator. Since actual implementation is not yet feasible, another verification method is required.

One other verification method is to place faults into the physical hardware model and determine the faulted operation by hand. Placing faults in each different area to calculate the effects of the network operation would be time consuming. Also, since processing faults tend to cluster, there is fault interaction where multiple faults occur in a single message path. Fault interaction and the large size of the networks prohibit a complete hand calculation of fault effects.

To verify the operation of Fltsim, several faults located by the fault generation routine were studied for their impact on the operation of the network. These faults modify the network operation according to predicted hand calculations.

Two extremes can be used to model the granularity of the circuitry in the architecture, as shown by Figure 24. One extreme is to model the circuit as implemented in the

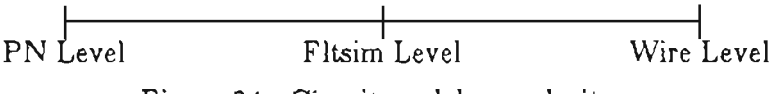


Figure 24 - Circuit model granularity.

silicon, as gates and wires. Although this model is the most accurate, it requires too much detail that is not yet available and would require extensive memory and time to simulate. The opposite extreme is to model the architecture at the PN level. The inputs or outputs of the PN could be modeled as containing the faults. This level is too coarse as each PN is comprised of several different functions. Faults in each of these functions affects the operation of the network differently. Fltsim is in the middle of these two approaches and models the network more accurately than the PN level, but not at the gate level. The question remains, how much accuracy is lost from the wire model? More research is required to answer this question thoughly. Fltsim uses the information available to model the network at its current level of implementation.

The ultimate goal of the fault simulator is to test the fault tolerance of the neural network architecture being developed. The defects that occur in the hardware do not need to be modeled exactly in the n-graph. The n-graph is faulted using the best approximation available, which is much better than introducing random faults into the network. The modified operation of the network will still determine the fault tolerance of the network, even though it is not a perfect model of the actual defective circuit operation.

7. SUMMARY AND CONCLUSIONS

A fault simulator tool has been developed that models worst case local defects in a wafer-scale integrated neural network emulation architecture. The fault simulator allows the fault tolerance of the neural network to be modeled at a high level before the network is actually implemented. The fault model used is a combination of the fault clustering model developed by Stapper and the radial distribution model.

The contributions made by this work aid the study of the fault tolerance of the Cognitive Architecture Project at OGC. Fltsim also provides a general technique for determining the impact of processing faults using a high level description of an architecture. The use of a high level description allows fault tolerance to be incorporated into the design at a higher level, where the fault tolerance is easier to implement. The fault process could be expanded to other wafer-scale integrated architectures where a repeated pattern of devices is arrayed on the wafer.

The fault simulator was originally designed to model faults in a silicon implementation of a neural network. Conceivably, Fltsim could be extended to model faults in a biological system. Several steps in the fault simulation would require new models, but the general processes would remain unchanged. An n-graph can be used to describe the biological nervous system since biological systems have much the same structure as described by the n-graph. Fltsim would, as before, fault the n-graph operation according to known biological defects.

Fltsim builds a model of how the silicon hardware blocks are interconnected and each block's size. The size of each block determines the probability of a defect in that area and the interconnectivity determines which messages are corrupted by defects. A biological system has a network of nodes or synapses that are interconnected. Sizes for the various regions can be assigned depending on the probability of defects in those regions. The fault distribution model can be altered to model the characteristics of biological defects. Stuck-at faults can model the incorrect activation between the synapses. The n-graph operation could be faulted as normally done, and HAS or ANNE could be used to simulate the faulted network.

Future Enhancements

As the network is refined, and can be modeled more accurately, the fault simulator can be enhanced to provide a better fault analysis. The communication structure is of primary concern, as it is the link for the neural network model, and ties many nodes together through time division multiplexing. Some assumptions were made to simplify the design of the fault simulator. The uniform PBH areas should be expanded to non-uniform PBH areas with varying numbers of data/control lines and different sizes of regions. The concentrator/deconcentrator nodes should be assigned sizes to add to the bus area. These concentrator/deconcentrator node sizes could increase in size towards the top nodes to model larger buffers sizes. Future versions of the PBH bus will possibly include a fat tree[Rud88a], where the width of the data bus increases towards the top nodes, resulting in a higher data bandwidth at the top nodes. The fat tree helps alleviate the bottleneck of many PNs sending messages using the PBH bus. Also, redundant root nodes or communication channels should be added to increase the fault tolerance of the

network. Redundant hardware can be modeled by not faulting the n-graph operation until a predetermined number of faults occur in the hardware block.

Bus line spacing, bus line width and defect sizes could be included in the simulation. Studies have been done on how these parameters relate to one another. Incompletely severed bus lines or partially damaged transistors could cause AC parameter faults. These faults could be modeled as delay faults in the network, where the signal gets to the proper value, but it takes longer to make the transition. HAS and ANNE already model delays in the network for normal message transfer times.

Faults in the hardware blocks could have different effects on the network operation according to predefined probabilities. Currently, single bit faults are modeled. A single defect may damage the control structure in a hardware block or a large defect could effect multiple bits. For example, in a RAM two adjacent bits may be faulty, or a whole row or column may be faulty. Information about the layout of the cell will indicate the probabilities for multiple bit faults.

A modification to reduce the execution time of Fltsim would be to sort or hash the fault list. For each section in the BIF file, the fault list is searched for faults that effect that section. If the list could be searched faster by sorting the list of pointers by specific types of faults, this search time would be reduced. Currently this search time is the major bottleneck for the simulation.

References

Ham86a.
Dan Hammerstrom, "Connectionist VLSI Architectures," Project Proposal, Dept. of Computer Science, Oregon Graduate Center (Aug 1986).

Lei85a.
Tom Leighton and Charles E. Leiserson, "Wafer-Scale Integration of Systolic Arrays," *IEEE Transactions on Computers* **C-34** pp. 448-461 (May 1985).

Har88a.
Jim C. Harden and Noel R Strader II, "Architectural Yield Optimization for WSI," *IEEE Transactions on Computers* **37**(1) pp. 88-110 (Jan 1988).

Bai88a.
Jim Bailey, "Mapper – A Program to Map CNNs to Physical Networks," Technical Report, Dept. of Computer Science, Oregon Graduate Center (1988).

Bah88a.
Casey Bahr, *ANNE: Another Neural Network Emulator,* MS Thesis, OGC (1988).

Jag88a.
Kevin Jagla, *Concurrent Neural Network Simulator - HAS,* MS Thesis, OGC (1988).

Bai86a.
Jim Bailey and Dan Hammerstrom, "How to Make a Billion Connections," Technical Report No. CS/E-86-007, Dept. of Computer Science, Oregon Graduate Center (August 1986).

Sta86a.
C. H. Stapper, "On yield, fault distributions, and clustering of particles," *IBM Journal of Research and Development* **30**(3) pp. 326-338 (May 1986).

Che87a.
Chen, Ihao and Strojwas, Andrzej J., "Realistic Yield Simulation for VLSIC Structual Failures," *IEEE Transactions on Computer-Aided Design* **CAD-6**(6) pp. 965-980 (Nov 1987).

Wes85a.
Neil Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design, A System Perspective*, Addison-Wesley (1985).

McD86a.
J.F. McDonald, Capt. B. J. Donlan, R. H. Steinvorth, H. Greub, M. Dhodhi, J. S. Kim, and A. S. Bergendahl, "Yield of Wafer-Scale Interconnections," *VLSI Systems Design,* pp. 62-66 (December 1986).

Wal86a.
D. M. H. Walker, *Yield Simulation for Integrated Circuits,* PhD Thesis, CMU (July

1986).

Gal80a.

J. Galiay, Y. Crouzet, and M. Vergniault, "Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability," *IEEE Transactions on Computers* **C-29**(6) pp. 527-531 (June 1980).

Raf85a.

J. I. Raffel, A. H. Anderson, G. H. Chapman, K. H. Konkle, B. Mathur, A. M. Soares, and P. W. Wyatt, "A wafer-scale integrator using restructurable VLSI," *Joint Special Issue on VLSI, IEEE Journal Solid-State Circuits and IEEE Trans. Electron Devices*, (Feb 1985).

Rud88a.

Mike Rudnick and Dan Hammerstrom, "Physical Broadcast Structure," OGC Technical Report No. CS/E-88-018 (April 1988).

APPENDIX A:

FAULT SIMULATOR COMMAND LINE DESCRIPTION

Fltsim reads and writes several files. Each input file contains information required by the simulator to build physical models of the architecture. The output files are used to determine the effects of the faults and as input files for the architecture simulators to simulate operation of the faulted network. The command line for the fault simulator includes several options for specifying source and destination file names. Listed below are the command line optional arguments with the default file names that are used if the option is not specified. All arguments in brackets are optional.

```
fltsim      [-b BIF.in]       [-s fstat.output]      [-T]
            [-t tech.in]      [-o fbif.output]       [-N]
            [-f fault.in]     [-d test.output]       [-h]
            [-p pad.in]
            [-S seed]

Where -T is TEST mode
      -N is No faulted BIF output
      -S is the random # generator seed
      -h is help
```

Calling Fltsim with the `-h` option returns the above text indicating all the options and default file names that can be specified. The `-b` option specifies the mapped BIF file input describing the n-graph, the `-t` is the technology file input information providing device size information, the `-f` is the input fault parameter file to characterize the fault distribution, and the `-p` option specifies the physical architecture data file describing the

physical hardware. The -s specifies the fault statistics output file to summarize the fault information, -o the faulted BIF fields to be used by the architecture simulators, HAS and ANNE, and -d the test output file to list intermediate Fltsim values. The default file names are listed inside the brackets. The format and content of these files are described in the Fault Simulator File Formats appendix.

A TEST mode can be specified to write intermediate values of the simulation to a file. TEST mode can be used to troubleshoot problems that occur, or to obtain more statistics on the characteristics of the faults or hardware model. Information such as the calculated size of the PNs, internal block sizes in the PNs, bus sizes and fault locations are written to this file. Also included is the parsed fBIF output, listing information about what the fault simulator thought each line type was (e.g. iotype field or link information.). The test file output is helpful if fltsim returns an execution error.

The TEST mode can be specified two ways. If the -T option is specified or if the -d option with a file name is specified, the TEST output will be written. If the -d option is listed in the command line, the -T is not required. The -T option writes the test information to the file named test.output. To direct this test output to a different file name, the -d option with a file name is specified.

Fltsim normally will produce a unique set of random numbers each time the procedure is invoked. Sometimes it is desirable to have repeatable faults to diagnose problems with input files or Fltsim routines. Repeatable random number generation is accomplished by specifying the -S option with the same seed value. The sequence of random numbers generated by the random number generator will be identical for each invocation of Fltsim with the same seed values. The seed value used is printed in the

standard output of the fault simulator and is printed in the fault statistics file to allow repeating the execution of any Fltsim run. The seed is contained in the fault statistics file because this file is always generated with each fault simulation. Given the same input file data, successive executions of Fltsim with identical seed values will produce output files with the same contents as prior executions.

Very large networks can be run though the simulator without generating the fBIF file. The -N option is then used to inhibit the writing of the faulted BIF output. This option does not effect the modeling of faults, and the fault statistics file, fstat, is still produced.

APPENDIX B:

FAULT SIMULATOR FILE FORMATS

PAD File

The PAD file describes the physical architecture of the neural network. It consists of sizes of the hardware blocks and the connectivity of the network. Comments are allowed to help describe the network information. Each comment section must be contained on a single line and start with a "/*" and end with a "*/". This is similar to comments in the C programming language, except for the fact that comments cannot span multiple lines. The PAD file has 4 major parts:

- The CN/PN network descriptions
- PTP Network description
- Local PAD region definitions
- PBH Network description

The PAD file starts with a description of the PNs and CNs. The total number of PNs in the network and how many are in the x and y dimensions is listed first. Next is the number of CNs in each PN of the network. Each CN has a number of inputs, which are values from other CNs. These other CN values are stored in a memory for later use. The number of bits in each CN value is identified by the CN_DATA field. Correspondingly, each input has a weight. The number of weight bits in each value is identified by the WEIGHT field. The total number of other CN values stored in any one PN is

```
/*****************************************************************/
/*             PAD VERSION # 6 example file        7/16/87       */
/*             This is an example PAD description file           */
/*****************************************************************/
/* Start with PN/CN descriptions                                 */
/*****************************************************************/
PN: 16;                 /* total number of PNs in the network */
PN_GEO: 4, 4;           /* x,y dimension of PNs */
CN: 2;                  /* number of CNs in each PN */
CN_DATA: 8;             /* represents current state of other CNs */
WEIGHT: 9;              /* # of weight bits */
CN_ENTRY: 7;            /* # entries of other CN states (input links) */
CN_ADDR: 15: 4, 9: 2;   /* # entries: # address lines */
LSM: 2;                 /* # of learning state machines in each PN */
/*****************************************************************/
/* Define Point-to-Point Communication Network                   */
/*****************************************************************/
PTP_DATA: 4;            /* # of PTP data lines between each PN */
PTP_CNTL: 2;            /* # of PTP control lines between each PN */
PN_CON: 0: 1,4;         /* connections between PNs */
PN_CON: 1: 0,2,5;       /* this will be used to map external PN # */
PN_CON: 2: 1,6,3;       /* number system to internal # system */
PN_CON: 3: 2,7;
PN_CON: 4: 0,5,8;       /* CONNECTIVITY FOR ALL PN'S */
PN_CON: 5: 1,4,6,9;
PN_CON: 6: 2,5,7,10;
PN_CON: 7: 3,6,11;
PN_CON: 8: 4,9,12;
PN_CON: 9: 5,8,10,13;
PN_CON: 10: 6,9,11,14;
PN_CON: 11: 7,10,15;
PN_CON: 12: 8,13;
PN_CON: 13: 9,12,14;
PN_CON: 14: 10,13,15;
PN_CON: 15: 11,14;
/*****************************************************************/
/* Define regions of PNs to be used in PBH definition            */
/*****************************************************************/
REGION: 2: 1, 2;                    /* list PNs in region 2 */
REGION: 1: 3, 6, 7;                 /* list PNs in region 1 */
/*****************************************************************/
/* Describe Physical Broadcast Hierarchy Communication Network   */
/*****************************************************************/
PBH_LEVELS: 2;              /* # levels in PBH hierarchy */

PBH_N: 0: 1: 2;         /* PBH PN netlist - includes Region 1 & PN 2 */
PBH_BDCAST: 7;
PBH_DATA: 2,4;          /* 2 data lines in lowest level, 4 in upper 2 */
PBH_CNTL: 3;            /* 3 control lines in all levels */

PBH_N: 1: 2: 3,4;       /* PBH PN net list - Regions 2 & PN 3 & 4 */
PBH_BDCAST: 12;
PBH_DATA: 3,5;
PBH_CNTL: 3,4;
```

Figure B1 - PAD File.

CN_ENTRY. CN_ENTRY is the total number of input links to all the Sites on a CN as shown in the n-graph. The network uses an addressing scheme where the incoming

message's address field is stripped off the message and sent to the address decoder. The address decoder uses a Content Addressable Memory (CAM) to determine the proper CN_MEMORY word to store the message's data field in. A reduction in transmission time and silicon area can be achieved if the number of address bits required is variable. The CN_ADDR field specifies the number of CN entries with the number of address bits used. The total number of CN_ADDR entries should add up to the total number of input links for the CN. An assumption that each CN within the PN has the same address decoder structure was made, allowing the network to be scaled easier. If the number of CNs in a PN is changed, the number of address entries changes by a corresponding amount.

The next block describes the PTP communication network. Each PN is connected via the PTP communication network to all its immediate neighbors. Each message uses handshake lines to communicate, for example, Data Valid and Data Received lines. Message destinations will not have to be the neighboring PN. The message may have to hop several PNs before reaching its final destination. Encoded in each message is the destination PN number and CN address. The destination address and data is multiplexed over a set of data lines. The PTP fields indicate the number of data lines and the number of control lines for each PTP connection. PN connectivity is listed in the PN_CON fields. For the example shown, PN # 2 is connected to PN # 1, 6 and 3. Fltsim requires knowledge of the x,y addresses of the PNs in the network, so the PNs listed in the PN_CON fields must be listed in physical order. That is, the PN at x,y location 0,0 is first, 1,0 next and so on.

The PAD file allows for shorthand notation for grouping regions. Grouping regions gives a specified number of PNs a given region number that can be referred to later. Larger numbers of PNs can be more easily developed using these regions.

The last section describes the PBH communication structure. The number of levels in the PBH communication tree is listed first, indicating the number of concentrator and deconcentrator nodes. Next is the information for each PBH Region in the network. Note here that the PBH Region is different from the regions defined earlier in the PAD file. Each PBH Region number is specified in the PBH_N field, with all the PNs listed for that PBH Region. The PN list starts with all the predefined regions from above listed first, separated by comas, followed by a colon, and then all the individual PNs listed. The number of broadcast data lines, PBH data lines, and control lines is similar. The number of lines for each level of the PBH hierarchy can be specified, allowing differing number of signal lines for each level. All higher levels from the last number listed will have the same number of signal lines. For example, the PBH_DATA of 2,4 indicates that the lowest PBH level has 2 data lines, the second and all higher levels have 4 data lines.

With all the information given in the PAD file, a block diagram of the physical hardware is possible. The block diagram will contain sizes of the hardware blocks internal to each PN and all the connectivity between the PNs.

Technology File

The technology file describes the sizes of the individual elements, such as memory cells or buffer sizes. Each parameter is specified by a keyword followed by a numeric

value indicating the size. Only one parameter is allowed per line and the text after the numeric value can be used as a comment. Entire lines may be used for comments and have to be preceded with a "/*".

The values in the technology file are multiplied by the dimensions given in the PAD file to determine the silicon areas required to implement the functions. The sizes for the components must be given in microns and square microns. The memory_size indicates the size for a single bit memory to store data in the CN_MEMORY block containing other CN values. The adc_size and dac_size are for the analog arithmetic unit to calculate the CN output value. The buffer_size is for the PTP and PBH data and control buffers to drive the bus to the concentrator nodes. The cntl_size indicates the amount of global control circuitry for a PN. The line_width is the line width for the PTP and PBH bus lines only. Line width is used to determine the total amount of silicon area a bus line occupies. Address decoding within a PN is done using a Content Addressable Memory cell. The size of the CAM cell is listed as the addr_dec_size field. Lsm_size is a scale

```
                    /* file:  technology file */
                       /* date:  27 Nov 87 */
memory_size          50   /* memory size */
adc_size            750   /* adc size */
buffer_size         100   /* buffer size */
cntl_size           800   /* size of control section */
line_width            2   /* line width */
dac_size            750   /* dac size */
addr_dec_size        75   /* address decode size */
lsm_size            960   /* learning state machine size */
weight_size          50   /* memory cell size for weight storage */
ptp_demux_size      100   /* ptp control, addr compare, & demux size */
pbh_demux_size      100   /* pbh control and demultiplexer size */
```

Figure B2 - Technology File.

factor for the amount the LSM block increases for each independent learning algorithm used in each PN. Weight_size indicates the memory bit size to store each bit of the weight field. Each PN has a PTP and PBH demultiplexer and control section. This section determines if it is the final destination for the message by examining the PN destination number. Once the message reaches the final PN destination, this section separates the address field and the data field and sends them to their correct destination, the address decoder and CN MEMORY. The size of this section is shown in the ptp_demux_size and pbh_demux_size fields.

Fault Parameter File

The fault parameter file includes all the parameters to be used in the fault generation routine. Each parameter is specified by a keyword followed by a numeric value indicating the size. Only one parameter is allowed per line and the text after the numeric value can be used as a comment. Entire lines may be used for comments and have to be preceded with a "/*". Fault parameters are used to generate the locations of the faults and the types of faults.

```
                    /* file:  fault file */
                    /* date:  27 Nov 87 */
defect_den   15     /* average defect density per sq inch */
s-a-0         0.30  /* fraction of open faults */
cluster       0.3   /* clustering coefficient between areas */
in out        0.40  /* inner/outer fault density ratio */
```

Figure B3 - Fault Parameter File.

The average defect density for the simulation is shown by the field defect_den. The defect density is multiplied by the area of the network to determine the average number

of defects to place. The fraction of Stuck-at-0 faults is a parameter, which also determines the fraction of Stuck-at-1 faults. Fault clustering can be varied by the cluster parameter. A typical value for the fault clustering coefficient for a wafer is around 0.3. The fault radial distribution is determined by the in_out parameter. The in_out parameter is the ratio of the defect density of the inner area divided by the defect density of the outside area. To simulate a wafer, the ratio of the inner area defect density to the outer area defect density will be less than 1. To simulate faults on a die in a wafer, the ratio should be set to 1, as the radial distribution only applies to the wafer model.

BIF File

The BIF file describes the n-graph of a neural network, which is described in more detail in Casey Bahr's Thesis, "ANNE: Another Neural Network Emulator" (OGC). The BIF file required by the fault simulator must be mapped to the physical implementation of the network. The mapper routine should be used to assign n-graph nodes the p-graph nodes needed by the fault simulator. BIF describes the n-graph Connection Node connectivity. The BIF lists Group information about the CNs, followed by a list of CNs, a list of Sites for each CN, and a list of links for each Site. Each list describes a different portion of the connections of the n-graph.

Fault Statistics

The fault statistics file contains the fault statistics of the faults in the network. The statistics show where faults occurred and what effects on the n-graph they had. When simulating the network using HAS or ANNE the statistics will provide information to aid in understanding the modified circuit operation. Figures B4 and B5 show a

portion of the fstat file.

```
fault statistics
run time = Mon Dec 21 11:42:37 1987

fault # 0
 DATA    S_A_0 in dac
fault index = 1
fault modifier = 0
fault input site
fault pn x,y = 0,0 which is PN # 0
fault offset = 182

fault # 1
   ALL VALUES IN THIS PN,DO NOT CHANGE DATA ADDR S_A_0 in pncntl
fault index = 195
fault modifier = 0xffffffff
fault output link
fault pn x,y = 1,1 which is PN # 3
fault offset = 0

*** END OF LIST FOR SINGLE FAULTS ***
*** FAULT COMBINATIONS LIST ***
   (determined when reading BIF file)
   (fault #'s correspond to faults listed above
    and the fault order below is the order faults
    are placed in the fBIF file)
CN index = 0
Site index = 0
Link offset = 0
weight (link) offset = 0
Site index = 1
Link offset = 0
weight (link) offset = 0
Link offset = 1
       *** New worst fault #1
weight (link) offset = 1
Link offset = 2
       *** New worst fault #1
weight (link) offset = 2
Link offset = 3
weight (link) offset = 3
weight (link) offset = 0

BIF file statistics
section        number         faulted        percent faulted
CN             12             1              8.33
SITES          24             0              0.00
LINKS          72             18             25.00
WEIGHTS        72             8              11.11
```

Figure B4 - Fault Statistics File.

```
BIF utilization of the Hardware defined in the PAD file
PAD:    16 CN's
        2 Sites (For now, max sites from BIF file)
        2048 Links
PN(0,0):
        2 CN's          12.50 percent utilization
        2 Sites's       100.00 percent utilization
        5 Input Links   0.24 percent utilization
PN(0,1):
        2 CN's          12.50 percent utilization
        2 Sites's       100.00 percent utilization
        8 Input Links   0.39 percent utilization
PN(0,2):
        2 CN's          12.50 percent utilization
        2 Sites's       100.00 percent utilization
        5 Input Links   0.24 percent utilization
PN(0,3):
        0 CN's          0.00 percent utilization
        0 Sites's       0.00 percent utilization
        0 Input Links   0.00 percent utilization
PN(1,0):
        2 CN's          12.50 percent utilization
        2 Sites's       100.00 percent utilization
        8 Input Links   0.39 percent utilization
PN(1,1):
        2 CN's          12.50 percent utilization
        2 Sites's       100.00 percent utilization
        8 Input Links   0.39 percent utilization
PN(1,2):
        2 CN's          12.50 percent utilization
        2 Sites's       100.00 percent utilization
        2 Input Links   0.10 percent utilization
PN(1,3):
        0 CN's          0.00 percent utilization
        0 Sites's       0.00 percent utilization
        0 Input Links   0.00 percent utilization
```

Figure B5 - Fault Statistics File (con'd).

The fault statistics are grouped into five sections:

- The header
- Hardware faults
- BIF faults
- BIF fault statistics
- BIF utilization

The fstat header consists of a description of what is contained in the file, "fault statistics", and the time the Fltsim program was executed. All the output files from Fltsim

contain a time stamp, which will be identical for all files generated from a single execution of Fltsim.

The next section lists each fault placed in the hardware circuitry with information describing the specific location and how it will effect the operation of the circuit. For example, fault #1 is a fault in the PN control hardware block of PN #3 with a x,y location of (1,1). This fault will be modeled by not allowing the output data links to change value. The actual fault index and modifier to be put into the fBIF file is listed, which indicates to the fault routines, how to model the fault. The fault offset indicates an offset within the hardware block that the fault occurred in, and is specific to the hardware block. See the Fault Effects appendix for more information of how to interpret the offset value.

The BIF faults list which hardware faults actually effect the operation of the n-graph. Two types of entries are used, "New worst fault" and "combine for new worst fault". "New worst fault" indicates the hardware fault listed has more impact on the operation of the network than the previous fault, or it is listed for the first fault to effect a BIF section. "Combine for new worst fault" indicates that the listed fault has been combined with the previous worst fault.

The BIF fault statistics summarize the faults placed in the BIF file. The total units for each section and the number and percentage of faulted units are listed.

The last section, BIF utilization, shows the utilization of the hardware circuitry by the BIF description. The number of sections available in the hardware (derived from the PAD file) is listed followed by the utilizations of the BIF sections for each PN. Extreme underutilization of the hardware by the BIF file may lead to invalid conclusions to be

made. For example, two networks that use the same PAD file are simulated, but one has a 50% utilization of the hardware and the other has a 100% utilization. Both circuits use the same PAD file, so the size of the hardware will be the same, resulting in the same average number of physical faults. Since only 50% of the hardware is used in the first network, a higher portion of the faults will be in unused hardware, which will not affect the n-graph, leading to a false sense of fault tolerance.

Faulted BIF

The faulted BIF file (fBIF), contains the fault information to be included by the architecture simulator fault routines. Arrays of fault information will be initialized to be accessed by the fault routines which are used to modify intermediate CN node calculations. An example if the fBIF file is shown in Figure B6.

The beginning of the fBIF file contains a comment section providing information about the generation of the fBIF file. The source BIF file used to generate the fBIF file is listed to coordinate the proper use of the same BIF file and fBIF file by the architecture simulator. Using the fBIf file with a different BIF file other than the one used to generate the fBIF file will cause unpredictable results.

Four arrays are initialized in fBIF, flts[], fcn_ptr[], fsite_ptr[][], and flink_num[][]. Flts[] is an array of fault indexes and modifiers to be used to determine how to modify the interface CN value. The other three arrays are indexes into the fault array, used by the fault routine to access the appropriate fault index and modifier. Fcn_ptr[cn_index] points to the CN fault data in the flts[] array. Fsite_ptr[cn_index][site_index] points to the site fault in the flts[] array. Flink_num[cn_index][site_index] contains the number of

```
/*********************************************************
 *
 * fbif file:  FAULT INDEXES AND MODIFIERS
 *
 *      source bif file - PF4.bif.0has.in
 *      run time - Mon Dec 21 11:42:37 1987
 *********************************************************/
static struct flts {
               unsigned short  index;
               unsigned int    mod;
} flt[] ={
0,0,
0,0,
195,-1,
0,0,
195,-1,
0,0,
0,0,
17,-513,
0,0};

static int fcn_ptr[] = {
0,
13,
26,
39,
52,
71,
109,
128,
167};

static int fsite_ptr[128][2] = {
{1,4},
{14,17},
{27,30},
{40,43},
{53,62},
{72,81},
{-1,-1},
{-1,-1},
{-1,-1}};

static int flink_num[128][2] = {
{1,4},
{1,4},
{4,4},
{4,1},
{0,0},
{0,0},
{0,0}};
```

Figure B6 - Faulted BIF File.

links for each site and is used to ensure that the link offset is valid. The link fault index into the flts[] array is calculated as an offset from the site fault index. For example, the first link fault value for each site is the first number after the site fault value. The

weight fault index is next followed by the link fault value for the second link.

Test Output

The test file output contains intermediate values used in the fault simulation process. Information about the sizes of the calculated physical network and its connectivity, parsed BIF output, fault location information and CN/PN mappings are included in the test file output. The test file can be used to verify operation of the simulator and to debug problems with parsing files, or other problems. Due to the length and variety of information contained in the test file, it is not listed here.

APPENDIX C:

ARCHITECTURE SIMULATOR

TO

FAULT SIMULATOR INTERFACE

The fault information that is to be modeled by the architecture simulators, HAS and ANNE, is conveyed though the fBIF file. Fault routines are called from the architecture simulator routines and user's routines to model the fault's operation. The process of calling these routines, and the information to be be conveyed to them will be described in this section. More detail about this interface is provided in the HAS and ANNE architecture simulator descriptions.

Three files are used by Fltsim to model faults by the architecture simulator: fault.h, fault.c and fbif.output. Fault.h defines constants used by the fault.c routines and Fltsim. Fault.c contains the subroutines to model faults in different n-graph sections. And fbif.output sets an array of fault fields for each subsection of the BIF file. This array of fault fields consists of a fault index and modifier for each CN, each CN Site, and each Site Link and Weight. In order for the architecture simulators to call the fault routines and access the fault fields, certain requirements must be satisfied. An include statement in fault.c includes the fBIF fault fields, which must correspond to the fBIF file produced by Fltsim to model the faults. Also, the fBIF file must have been generated from the

same BIF file as being used in the architecture simulator. The fault routines need to be linked and loaded with the user's fault routines. Each time the fBIF output changes, the fault.c file must be recompiled.

Each fault field in fBIF consists of two numbers, a fault index and a fault modifier. The fault index uses 8 bits to indicate the type of fault and where to model the fault. The file fault.h defines constants for the fault fields. Since the fault index value is also used internally to Fltsim, not all the bits will be used by the fault routines. All the bits will be defined here for completeness. Bits 0-3 are the fault locations that indicate if the fault was a data word fault or an addressing fault. Data word faults either modify the targeted value using bit operations, or do not update the value at all. Address faults change the routing of the network, so that the address that the message is being sent from/to will be altered. Bit 4 is set if a range of target values are to be faulted. Bit 4 is not used by the fault routines. Bit 5 indicates if the fault is a Stuck-at-1 (bit 5 high) or a Stuck-at-0 (bit 5 low) fault. Bit 6 if set high indicates that the target value should not be altered. That is, an old value and a new value are both passed to the fault routine, where if the NO CHANGE bit (bit 6) is asserted, the value returned is the old value. For example, the NO CHANGE fault is used when a handshake line is damaged and the destination node does not receive new values from the source node it is supposed to be connected to. The input to the site would always stay at the same value. Bit 7 indicates that all the CNs in a given PN are faulted when set high. Bit 7 is not used by the fault routines.

Fault Index

7	6	5	4	3 2 1 0
1 bit	1 bit	1 bit	1 bit	4 bits
ALL	NO_CHG	S_A_1	RANGE	Fault Fields

The NO CHANGE bit has the highest precedence in the fault index, so if it is set, the old value is returned, and the fault modifier is not used. Otherwise, the Stuck-At bit is used to determine the operation of the routine. For a data word fault, the Stuck-At bit will determine if the new value passed should be AND'ed with the fault modifier (S-A-0) or OR'ed with the fault modifier (S-A-1). The new value is so modified, and the routine is finished. If the fault is an address fault, the address routing is modified. Note that both the address and data can be corrupted. In this case, the fault modifier will be used to modify both the address and data fields.

The general steps for calling the fault routines are as follows:

1. For each input link:
   a. Fault the input link and corresponding weight.....`flt_lkwt()`
2. Update old input link values.
3. Calculate all the site functions for the CN.
4. Fault all site function outputs. .....`flt_site()`
5. Calculate the CN function.
6. Fault the CN function output. .....`flt_cn()`
7. For each output link:
   a. Fault the data and the destination address. .....`flt_olink()`
   b. Send the output to the next CN address
8. Calculate the weight functions

9. Fault the weight values.....flt_wt()

10. Update the old weight values.

These steps are performed for each CN calculation in the network. For step 1, faulting all the input links and weights to the CN, a link and weight fault routine, `flt_lkwt()`, is called for each input link to the CN. The fault routine will return the value to use as the link input and weight depending on the fault fields listed in the fBIF file. Figure C1 summarizes the parameters for all the fault routine calls. Six arguments are passed to the link/weight fault routine, the current CN index and site name that this link attaches to, the link index, the old (or previous) value of the link input, a pointer to the new link input and a pointer to the weight for the link input. After the subroutine returns, the contents of the pointer to the new link input will contain the value to use as the link input and the contents of the pointer to the weight will contain the new weight. Both the old value of the previous link input and the current link input are passed to the fault routine to model faults where the input link is defective, not allowing new inputs to be transmitted. Faulty input links which do not allow transfers to take place will set the new input equal to the old input value. The initial old value will need to be determined for the first call for each link input. After the link fault routine completes, the old value can be set to the current value for the next pass.

The site function for each site of the CN is calculated in step 3. The output of each site function is faulted in step 4. The site output is faulted by calling the site fault routine, `flt_site()`, with the current CN index and site name, and a pointer to the site output value. The fault routine will calculate the new site value, using the fault fields, to be returned as the contents of the site output pointer.

```
flt_lkwt(cnindex,site,link,old_link,new_link,weight)
  int   cnindex;        /* current cn index */
  short site;           /* current site name */
  short link;           /* current link index */
  short old_link;       /* previous link value */
  short *new_link;      /* new link value */
  short *weight;        /* weight for link */

flt_site(cnindex,site,new_site)
  int   cnindex;        /* current cn index */
  short site;           /* current site name */
  short *new_site;      /* pointer to the new site value */

flt_wt(cnindex,site,link,old_weight,new_weight)
  int   cnindex;        /* current cn index */
  short site;           /* current site name */
  short link;           /* current link index */
  short old_weight;     /* previous weight value */
  short *new_weight;    /* new weight value */

flt_cn(cnindex,new_cn)
  int   cnindex;        /* current cn index */
  short *new_cn;        /* new cn value */


flt_olink(cnindex,site,link,new_addr,new_data)
  int   cnindex;        /* current cn index */
  short site;           /* current site name */
  short link;           /* current link index */
  short *new_addr;      /* new address */
  short *new_data;      /* new output link value */
```

Figure C1 - Fault routine parameters.

Faulting the CN function is done similarly. The CN function calculates a new value and calls the CN fault routine, `flt_cn()`, passing it the CN index and a pointer to the new CN value. The fault fields are accessed to modify the CN output.

The output links are faulted by calling `flt_olink()`, which modifies the destination address and data value to send. The CN output would normally be sent to the destination CNs using the output links. Pointers to the destination CN address and the CN output are passed to a fault output link routine along with the current CN index, site name and link index. The address and CN output are modified using the fault fields in the fBIF file to change the routing and the CN output. The modified CN output is then sent to the destination CN using the modified routing.

Steps 8 through 10 are used for networks with dynamic weights that are calculated during execution of the network. Step 8 calculates the new weights for all the input links using a selected learning algorithm. Step 9 is to fault the weights by calling a fault weight routine, `flt_wt()`, passing the CN index, site name, link index, the old weight value, and a pointer to the new weight value. The fault fields in the fBIF file are used to potentially modify the contents of the new weight value. Some faults may cause the new weight not to be calculated or saved in memory, resulting in the weight never being updated. The old weight is used in this case, where the new value is set to the old value. After the fault weight routine completes, the old value is saved for the next pass. The weights are faulted twice, once with the input links and once when the weight values are updated. There are two reasons for doing the weight fault twice. First, steps 8-10 are optional for networks that do not calculate new weight values. Second, the weight values are accessed by the CN at two different times so the faulted value is required twice. So for consistency between dynamic weight networks and static weight networks, the weights are faulted twice. Faulting a value twice has no adverse effects, since the same fault index and modifier are accessed both times. And faulting a faulted value does not change the value. For example, AND'ing the fault modifier with a value a second time does not alter the number.

The actual code to implement the fault routines will depend upon the simulator used and the site/CN functions and data structures used. Some of the fault routines may be called from the architecture simulator, and hence should not be called by the user's routines. With the general guidelines presented here, the interface of the fault routines to the architecture simulator routines and the user's routines can be implemented. The

specific architecture simulator description should describe which fault routines are to be called by the user's routines.

Shown below is a simplified section of the user's routine for HAS with the fault routine calls shown in bold print:

```
if(userfx_mode == 1)
   { /* receive link input messages */
   L->old_inval = L->inval;    /* save old value for fault routine */
   L->inval = mes_value;       /* receive message input to node */
   }
else if(userfx_mode == 2)
   { /* Site function - Sum of Products (called once per site input */
   siteval = S->siteval;
   wt = L->inval;
  flt_lkwt(C->cnindex,S->sitename,L->link_index,
            L->old_inval,&(L->inval),&wt);
      /* fault input link and weight */
   inval = inval * wt;
   siteval += inval;
   S->siteval = siteval;
   }
else if(userfx_mode == 3)
   { /* CN function - Sigmoid function */
   siteval = S->siteval;
  flt_site(C->cnindex,S->sitename,&siteval);      /* fault site output */
   C->output = (1/(1 + exp(-1.0 * siteval)));
   }
else if(userfx_mode == 4)
   { /* send outputs to next CN's */
  flt_olink(C->cnindex,S->sitename,L->link_index,
              &(C->index),&(C->output));
      /* fault destination address and CN output */
   sprintf(buf,"%d %d %d",C->index,C->output,time);
   send_output(C->index);
   }
```

Since the example shown does not use dynamic weights, step 8-10 are omitted; the `flt_weight()` routine is not called.

APPENDIX D:

FAULT EFFECTS

The induced faults reflect the actual physical faults that will occur in the system. Some approximations were used in the fault model to simplify the design of the fault simulation and because the CAP network architecture is still in the design phase. This section will cover each section of the hardware block diagram, describing the general function of the block, what types of faults can occur in the block, how the faults effect the function, and how the faults will be modeled in the n-graph. The n-graph is used by the architecture simulators to simulate the operation of the network. The fault fields written in the fBIF file by Fltsim modify the n-graph operation. The hardware blocks described here, except for the bus structures, are replicated for each PN in the network. Potentially several CNs will be in each PN as described earlier in the paper. Some faults effect only one CN in the PN, whereas other faults will effect all the CNs in the PN.

There are three basic types of faults modeled here, Stuck-At-1, Stuck-At-0, and NO CHANGE. The S-A-1 and S-A-0 faults OR or AND a fault modifier with the value, forcing bits in the word to be always high or low. The NO CHANGE fault forces the value not to be updated, that is, it retains its previous value.

CN MEMORY

The CN MEMORY section is a RAM that stores values of other CN's function outputs or states. The CN MEMORY words stored here will be inputs that the CN function will use to calculate its output. There is one word for each input link to the CN. Each input link gets another CN's output to write a unique word in the CN MEMORY. Only those CNs connected to this CN store their values in this CN's memory.

The most common defect in RAM structures is to lose single bits in the stored data words. When a value is read from the RAM, a bit in the stored value will be always have a high or low value, regardless of what value was written. Defective bits could be caused in the physical hardware by shorts, opens or leakage current between cells. Faults not modeled by Fltsim are RAM control structure faults. Control structure faults will cause multiple faults in the RAM with only one fault in the device. Row or address decoding faults for the RAM are examples of control structures that could be defective.

Defective bits in the memory will change the input link value for a specific link. A random bit in the memory is chosen to be stuck high or low. The defective bit is mapped to the word the fault occurred in. The defective word corresponds to a specific input link. This input link in the n-graph will be faulted. The corresponding link will be determined by the listed order of the links in the BIF file. The first link listed in the BIF file for a particular PN will be the first word in the memory, the second link in the PN will be the second word in the memory, etc.

ADDRESS DECODER

The address decoder uses a Content Addressable Memory (CAM) structure to match the incoming CN address with the corresponding CN value in the CN MEMORY. The incoming CN value is stored in the CN memory word that has a matching address. If no address match is found, the value is not accepted by any of the CNs in the PN. The connectivity of the network is stored in the CAM that is described by the link section in the BIF file. To save silicon area, the addresses can be of variable length. The more common local addresses can be encoded with fewer address bits.

Faults in the address decoder will cause improper decoding of the CN address, resulting in CN data values being written to wrong memory words (incorrect links), no value being written, or multiple CN values being written at once. Faults in the global control of the CAM may cause entire rows or columns to malfunction. Single bit faults are the only faults currently considered by Fltsim.

To fault the address decoder, a random bit within the address decoder is chosen to be stuck high or low. The bad bit is located within the addresses stored in the CAM. Since variable length addresses are allowed, each different address length section has a different probability of a faulty address bit. The faulty address word and faulty bit position are identified. The address word is an offset within the address words for a PN. There is one address word per input link for each CN in the PN. The order of the CN address words (for each input link) corresponds to the order listed in the BIF file. The link address for the faulty address order is listed in the BIF file. The link address for the faulty address word is modified to have either a stuck high or stuck low bit. In the architecture simulators, the destination address will be modified so that if the new address

matches some PN/CN in the network, the message will be routed to it. In most cases, the new address will not match a PN/CN, and the message will be lost.

WEIGHTS

The WEIGHTS section stores all the current weights for each CN input link to be used in the site function. One weight word is used for each input link. The Weights section is a RAM that is addressed either when the site function is to calculated, or when the Learning State Machine is to read or update the value. Typically the weight is multiplied by the input link value in the site function.

Since the weights section is a RAM, faults here are similar to the faults in the CN MEMORY section. Any weight bit could be stuck, and would always read a 1 or a 0. Stuck bits will effect the site calculation and the weight update or learning algorithm. To model weight faults in the site calculation, the weight value is modified with either stuck high or stuck low bits before using the value in the site function. To model faults while calculating the new weight using the learning algorithm, the new weight value is faulted with the stuck bit after the new weight is calculated. That is, the weight was modified and stored as the current weight when the site function was called. The LSM used this faulted value to calculate the new weight, and that weight was faulted again, since it was stored in the same RAM word with the same fault. The second time the weight is modified, faults in the LSM are also taken into account.

LEARNING STATE MACHINE

The Learning State Machine (LSM) is a PLA running as a background task that samples and updates all the weights for the site function according to a predefined

learning algorithm. Various inputs are used to calculate a new weight value, and may include the current weight, the CN function output, and the CN link input. If any of the inputs to the LSM are faulted, the new weight value will be faulted. All the LSM input faults are modeled by the other sections in the PN. Only internal faults to the LSM are considered in this block. LSM faults could cause the LSM not to work at all or to update the weights with incorrect values. If the LSM does not work at all, none of the weights for the PN will be updated with the new weight value, otherwise the fault will cause the new weight values to have a stuck bit. A predetermined fraction of faults will cause the LSM not to function at all. If a fault occurs in one of the LSMs, all the CNs updated by that LSM will have the same fault in all the weight values. When a fault occurs in one of multiple LSMs in a PN, 1/(# LSMs) of the weights are faulted. See the WEIGHT section for more details of how LSM faults are modeled.

DAC

The Digital to Analog converter (DAC) will calculate the site functions for the CNs in the process of converting the digital words to analog signals. Corresponding words from the CN MEMORY and the WEIGHTS sections are input to the DAC, converting the digital signals to analog signals while performing the site function simultaneously. The analog output represents the output of the site function.

Faults in the DAC section will cause the DAC output to be at an incorrect level, resulting in an incorrect site function output. Incorrect output levels can be caused by any one of the input bits being stuck, which causes a fault identical to a bad bit in the CN MEMORY. Since the faults to the inputs are the same, input bit faults are modeled in the CN MEMORY and WEIGHTS sections. Also, the output of the DAC could be

stuck, causing its output to be always stuck at one of the supply rails, which is modeled as the output of the site function to always being all ones or all zeros.

ADC and MUX

The Analog to Digital Converter (ADC) will perform the CN function calculation with analog signals. The ADC will combine all the analog signals from the outputs of the DACs into one analog signal to be converted back to a digital word that represents the CN function output to be sent to other CNs. There is one ADC section for each CN in the PN.

Faults in the ADC section will cause the CN output value to be corrupted. Faults in the inputs to the ADC will be modeled in the DAC hardware block by changing the output of the site function. The analog CN function output signal could be faulty, resulting in the digital CN function output being all ones or all zeros. The faulted CN output is sent via the output link to other CNs in the network.

CONTROL

The Control section groups together the global control circuitry for the PN. This section will not necessarily be in one physical location in the PN, but will be spread over the entire PN circuit. Clock signals that pace the PN circuitry and enable and disable various functions will be the type of control signals that will be included in this section. Any faults on these control lines will have a major impact on the operation of all the CNs in the PN. So faults here simply disable the whole PN. None of the output links from the PN change state, resulting in a static PN state.

PTP BUS

The PTP communication bus lines transfer messages between PNs/CNs. The bus lines are driven by the PTP DATA/ADDRESS/CONTROL buffers. Four sets of bus lines are present for each PN to connect to each neighboring PN. Messages may travel through several PNs before reaching their destination.

Faults may occur in the bus in the form of opens or shorts. Shorts will likely connect two adjacent runs by excess material in fabrication. Opens will occur if a defect causes a discontinuity in the signal path. Shorts are modeled as stuck high and opens as stuck low signals in Fltsim. The control lines that handshake the data transfers may be faulted causing no data transfers to occur. As with the PTP Buffers, the entire message path is checked for faults. Any faults found in the message path will be modeled in the output link of the source CN. If any of the data/address lines are stuck, the output of the CN function will be altered by modifying bits in the message. If the control lines are stuck, the transfer will be completely disconnected.

PTP DATA/ADDR/CNTL BUFFERS

The PTP DATA/ADDR/CNTL BUFFERS are the interface for the PTP communication network. This section includes bidirectional buffers for each data/address line to send and receive the messages, and buffers to control the handshake of the data. The size of these buffers will be larger to drive the capacitance in the data bus, which will make them more prone to faults. Four sets of PTP buffers are used for each PN to send/receive messages from/to any neighboring PN.

Faults may occur in any of the four sets of buffers. Faults in the buffers will effect all the CNs communicating through this set of buffers. The CN does not have to be local to the PN. Messages may travel through several PNs before reaching the destination PN(CN). If any of the intermediate PTP buffers are faulty, the message will be corrupted. The message path is determined by traveling in the x direction first, then in the y direction. For the source and destination PNs, only one side is checked for faults. Intermediate PNs have two sides checked, as the message enters one side and exits on another. Any faults in the message path are modeled at the output link of the source CN. If the output buffer for any of the data/address signals is faulted, the output of the CN function will be modified. If the input buffer for any data/address signals is faulted, the output of the sending CN will be faulted. For data/address faults, bits are either stuck high or low. Since the data is multiplexed, several bits will be stuck. Fltsim does not distinguish between faults on input buffers or output buffers. If any of the control buffers for the handshake are faulted, all transfers of data using these buffers will be impaired. PTP control signal faults are modeled by not updating the input link values.

PTP CONTROL/DEMUX

The PTP Control/Demux hardware block controls the input/output operations of the PTP interface for the PN. Messages are received from the four sets of PTP buffers. The messages are multiplexed, so this unit reassembles the subwords into a message unit. The PN destination portion of the address field is checked to see if this PN is the destination. If this PN is not the destination, one of the four PTP buffers is selected to forward the message, and the message is broken into subwords to be multiplexed over the data lines. If the PN is the destination, the address field is sent to the address decoder

and the data field to the CN MEMORY. For output operations, the CN function output from the ADC is sent to the PTP control section. The destination address is added to the message and the message is multiplexed over one of the four PTP communication buses.

Faults in the PTP control will corrupt the PTP message. Potentially, faults in the PTP Control could cause either single bit faults in the message, or all the bits in the message to be faulted. More commonly, faults in this section will cause the whole section to malfunction, inhibiting all PTP communication for the PN. The individual stuck bits are accounted for in other hardware sections. Faults in the PTP Control will disable all the PTP transfers for this PN. All PTP messages routed through this PN will be disabled, and the fault modeled at the source PN PTP output link.

PBH TRANSMITTER BUS

Several separate PBH regions may exist in the network. Each operates and is modeled as a separate autonomous bus structure. Each PBH region has a binary tree structure to send messages to a top concentrator node via the PBH TRANSMITTER BUS. Then the message is sent over to a separate binary structured RECEIVER BUS to broadcast the message to all the PNs in the region. The PBH TRANSMITTER BUS contains the multiplexed data/address lines, and some handshake control lines to send the message to the top concentrator node.

Faults in the bus, as with the PTP BUS, can cause opens or shorts in these signals. The shorts and opens are modeled by assigning a stuck high or stuck low fault to one of the data signal lines in a particular level of the PBH tree. This will cause several faults

in the address and data fields due to the multiplexed transmission of the data. The fault is modeled in the output links of all the CNs that use this faulted portion of the PBH bus. For example, if a PBH region has 4 levels, there would be 16 PNs in the PBH region. If a fault occurs on level 2, 4 of the PNs use the faulted portion of the bus and have their output links faulted.

PBH RECEIVER BUS

The deconcentrator network or RECEIVER BUS, sends a message to all the PNs in the PBH region using a binary tree structure. The data/address is multiplexed as in the PBH DATA/ADDRESS BUS. Control lines handshake the data between the nodes.

Faults, as with the other bus structures discussed here, can be opens or shorts which are modeled as stuck high or low values. Both the address and data fields will be modified by the fault. The level in the broadcast bus tree will determine how many PNs are effected by the fault, as in the PBH TRANSMITTER BUS. The faults will be modeled in the input links for the receiving PNs.

PBH DATA/ADDR/CNTL BUFFERS

The PBH Buffers drive the signals from the PN onto the PBH communication network. Bidirectional buffers are used for the data/address interface, and control signals handshake the data/address transfers. The output buffers drive the signal onto the PBH Transmitter Bus and the input buffers receive signals from the PBH Receiver Bus.

Any of the PBH buffers may be defective, but Fltsim only models faults in the output buffers for simplicity. Faults in the buffers are modeled as stuck high or stuck low faults. The message is multiplexed for the PBH network, so a message is transmitted in

several subwords. A faulty buffer would cause several bits in the message to be faulty, so that both the address and data fields are modified by a faulty buffer. Since all the CNs in the PN that use the PBH network use a common set of buffers, if any of the buffers are faulty, all the CNs in the PN will be faulted. If an output buffer for any of the data/address signals is faulted, the output of the CN function and the message address will be modified. Thus, the faulted data will be sent using a faulty address. If any of the control lines are faulted, all transfers of data will be impaired.

PBH DEMULTIPLEXER

The PBH DEMULTIPLEXER controls the interface to the PBH bus. It controls both the sending and receiving of messages on the PBH bus. The PBH DEMUX sends the incoming message's address information to the ADDRESS DECODER and the data information to the CN MEMORY section. It will also combine the address and data information from the ADC section in order to broadcast to other CNs over the PBH network.

Faults in the PBH DEMUX will corrupt the PBH messages. Potentially, faults in the PBH DEMUX could cause single bits faults in the messages, or all the bits in the message may be faulted. Faults can also cause the whole section to malfunction, inhibiting all PBH communication for the PN. The individual stuck bits are accounted for in other hardware sections. Therefore, faults in the PBH DEMUX will disable all the PBH transfers for the entire PN. All PBH messages, both inputs and outputs, will be disabled.

Hardware Area	Location	N-graph	NO_CHG	ALL	Field
CN MEMORY	Link	In Link	N	N	DATA
ADDRESS DECODER	Link	Out Link	N	N	ADDR
WEIGHT	Link	Weight	N	N	DATA
LSM	LSM	Weight	P	P	DATA
DAC	Site	In Site	N	N	DATA
ADC and MUX	CN	CN	N	N	DATA
PN CONTROL	NA	Out Link	Y	Y	ADDR
PTP DATA BUS	Side	PTP Link	N	N	ADDR/DATA
PTP CNTL BUS	Side	PTP Link	Y	N	DATA
PTP CNTL/DEMUX	NA	PTP Link	Y	Y	DATA
PBH RECEIVER BUS	Level/Region	PBH In Link	N	N	ADDR/DATA
PBH TRANSMITTER BUS	Level/Region	PBH Out Link	N	N	ADDR/DATA
PBH CNTL BUS	Level/Region	PBH Out Link	Y	N	ADDR/DATA
PBH DEMULTIPLEXER	Level/Region	PBH Link	Y	N	DATA

Figure D1 - Faulted Hardware to Fault Representation.

Figure D1 shows how defects in the various hardware blocks are represented. A defect will occur at a location within the hardware block, as shown by the first two columns. From the area and location, specific n-graph areas and fields are identified to model the fault. If the target value to fault is not to be updated with new values, the NO_CHG column contains a Y. If updating the value is dependent on other factors, a P is indicated for a Potential NO CHANGE. Otherwise, the N indicates the value will be updated. Values that are updated will use the Stuck-At model to model the defects. The N/Y/P is similar for the ALL column. If all the CNs in a PN are to be faulted, a Y is shown. If faulting all CNs depends on other factors, a P is indicated. And if only the one CN is to be faulted, an N is indicated. The Field column indicates the target message field to fault. Either the address or data fields of the target value can be faulted.

Field	N-graph	ALL	Fault:
DATA	In Link	N	Input link
ADDR	Out Link	N	Output (Source) Address
DATA	Weight	N	Weights for input link or updated by LSM
DATA	Weight	Y	All PN Weights
DATA	In Site	N	Input Site function output
DATA	CN	N	CN function output
ADDR/DATA	PTP Link	N	Output msgs routed through faulty PTP link side
DATA	PTP Link	Y	Output msgs routed through faulty PN
ADDR/DATA	PBH In Link	N	Input msgs routed through faulty level
ADDR/DATA	PBH Out Link	N	Output msgs routed through faulty level
DATA	PBH Link	N	Input and Output msgs routed through faulty level

Figure D2 - Fault Representation to Fault Action.

Figure D2 lists the faults represented in the fault simulator and which n-graph area will model the fault. For example, a data word fault in the n-graph WEIGHT section not having the ALL bit set will be modeled in all weights specified by the link offset or a range of faults specified by the LSM. If the ALL bit is set, all the weights for the PN will be modified. Figure D2 also points out whether the input link to a CN or the output link from a CN will model fault associated in the CN links.

Biographical

The author was born the 23rd of January, 1962, in Princeton, Illinois. After moving to Iowa, he graduated from Nevada High School in 1980. He graduated with honors from Iowa State University in 1983 with a B.S. in Computer Engineering.

The summer following graduation was spent in Poughkeepsie, New York, working as a summer intern for IBM, writing PL/1 programs. That Fall, he entered the M.S.E.E. program at Oregon State University in Corvallis. After the completion of one quarter, he relocated to Vancouver, Washington to work at Tektronix. He is an electrical engineer designing automated test equipment, and has also been involved with the firmware effort in several products.

In the spring of 1984, the author began his studies at the Oregon Graduate Center as a part-time student while still working at Tektronix. He has been a member of the Cognitive Architecture Project.