ON-LINE RECONFIGURATION OF SYSTOLIC ARRAYS

KARUNESH PRATAP SINGH, B.Tech.

ON-LINE RECONFIGURATION OF SYSTOLIC ARRAYS

By

© Karunesh Pratap Singh, B.Tech.

A thesis submitted to the School of Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Engineering

Faculty of Engineering and Applied Science
Memorial University of Newfoundland
June, 1992

St. John's Newfoundland Canada

ISBN 0-315-78121-1

ABSTRACT

Various existing reconfiguration algorithms for array processors cannot be used efficiently for on-line reconfiguration of the array because they require a central processor to initiate and control the reconfiguration. In addition, most of the existing algorithms assume that the switching network is operationally fault-free.

This report presents an on-line reconfiguration scheme for array processors. The proposed algorithm can tolerate both processing element failure and switching network failure. The processing elements and switches are of a self-testing type and link failures are detected by the processing elements (by using parity bit checks).

The array is provided with a bottom row of spare cells and when a processing element detects either a self fault or a link failure, it invokes the reconfiguration. A downward global shift (for the particular column) is performed to accomplish the reconfiguration. A number of reconfiguration requests are generated by the processing elements and switch modules to facilitate the reconfiguration. The network is modified and links for propagation of reconfiguration request are added. This scheme makes full use of non-faulty partial results and it blocks the faulty partial results.

The reconfiguration in the case of a processing element failure is completed in two stages while the reconfiguration in the case of a link failure is completed in a single stage. The links are duplicated to achieve redundancy and in the case of a link failure the spare link is used.

ACKNOWLEDGEMENTS

I am grateful to Dr. R. Venkatesan for his guidance and constant encouragement throughout the duration of my program.

I wish to thank the School of Graduate Studies and Faculty of Engineering, Memorial University for the financial support during my program.

I am indebted to Dr. G.H. George for his invaluable help in making this thesis more readable and presentable.

Finally, I would like to thank my wife, Ranjana for her support and help during my stay in St. John's.

Contents

List of Figures

List of Tables

List of Symbols

PE	Processing Element
I/P	Input
O/P	Output
I/O	Input/Output
I_i^H	Horizontal input of row i to the array
I_j^V	Vertical input of column j to the array
O_i^H	Horizontal output of row i from the array
O_j^V	Vertical output of column j from the array
w_s	Static Coefficients
$PE_{i,j}$	PE with physical index (i,j)
$PE_{i,j}^L$	PE with logical index (i,j)
I_x^t	Input of PE_x at time t (when only one input is there)
O_x^t	Output of PE_x at time t (when only one output is there)
I_x^H, t	Horizontal input of PE_x at time t
I_x^V, t	Vertical input of PE_x at time t
O_x^H, t	Horizontal output of PE_x at time t
O_x^V, t	Vertical output of PE_x at time t
I_{PE0}^H, I_{PE1}^H, I_{PE2}^H	Horizontal input ports of PE
I_{PE0}^V, I_{PE1}^V	Vertical input ports of PE
O_{PE0}^H, O_{PE1}^H, O_{PE2}^H	Horizontal output ports of PE

O^V_{PE0}, O^V_{PE1}	Vertical output ports of PE
$w_{i,j}$	Static coefficient (weight), corresponding to $PE_{i,j}$
$S_{i,j}$	Switch module with index (i,j)
$S^H_{i,j}$	Horizontal switch module with index (i,j)
$S^V_{i,j}$	Vertical switch module with index (i,j)
I^H_{S0}, I^H_{S1}, I^H_{S2}, I^H_{S3}	Horizontal input data ports of switches
I^V_{S0}, I^V_{S1}, I^V_{S2}	Vertical input data ports of switches
O^H_{S0}, O^H_{S1}, O^H_{S2}, O^H_{S3}	Horizontal output data ports of switches
O^V_{S0}, O^V_{S1}, O^V_{S2}	Vertical output data ports of switches
L^y_x	Link, connecting the output of x to y
RR^y_x	Reconfiguration request, generated by x and fed to y
CLK_{PE}	Global clock to the PEs
CLK_S	Clock pulses to the switches
ST^H_x	State x of the horizontal switches
ST^V_x	State x of the vertical switches
FF	Fatal Failure
E_{LOGIC}	Error in logic circuit
SPE	Signal to the spare cells
LF^y_x	Link failure signal, failure detected by x and reported to y
E_{IH}	Error in horizontal input
E_{IV}	Error in vertical input

Chapter 1

INTRODUCTION

Von Neumann architecture restricts the speed of a memory based hardware system because of the limited number of interconnections. The system speed can be increased by reducing the number of memory interactions. Systolic arrays accomplish this and thereby improve the system performance. Here, the interaction with the outside world occurs only at the boundary cells of the array and once the data are fed to the array, the intermediate results are not passed on to the memory devices. A systolic array is an array of similar processing elements, where every element performs the same basic operation.

Systolic arrays can be classified under various categories depending on the data flow inside the array. The most common type is that of moving result, static weights. In this type of arrays, the partial results move in a pre-specified way and the weights stay in the processing elements. The various types are described in Chapter 2.

These arrays have a number of similar processing elements, so some spare cells can conveniently be introduced. In the case of a cell failure, the spare cell can replace the faulty one, thus improving the system reliability.

Various reconfiguration schemes (described in Chapter 2) have been proposed for the reconfiguration of these arrays in the event of a fault occurrence. Most of the proposed reconfiguration schemes use hardware redundancy (spare cells) and in case of a fault detection, the reconfiguration algorithm is performed on the array by an external central processor (which maintains information about the operational effectiveness of processing elements). The external processor is capable of changing the data routing. The reconfiguration algorithm changes the data routing paths and makes the array operational if the algorithm is successful. The reconfigured array is flushed to clear the partial results and the array can then be used again.

Since the array is flushed after every fault occurrence, these reconfiguration algorithms cannot be used effectively during run-time. In addition, these algorithms assume a fault free routing network, which is difficult to achieve. These two major shortcomings restrict the use of the above algorithms to production time yield improvement.

An on-line reconfiguration scheme should preferably be capable of utilizing those partial results which were not affected by the fault occurrence (referred to as non-faulty partial results in this report). In addition, the faulty partial results should be blocked by the algorithm to ensure the proper operation of the array. If a faulty partial result is passed on to the next processing element, it would make the final results erroneous.

An on-line reconfiguration algorithm is presented in this report which accomplishes the above-mentioned tasks and, in addition, tolerates switching network failures. This algorithm requires an additional row of processing elements, called spare cells (and this row is the bottom most row of the array). When a processing element failure is detected, the spare cell of the corresponding column is used to replace the faulty cell. Similarly, redundant links are provided to ensure the tolerance of link failures.

The processing element and switch modules are redesigned to accomplish the generation of above system. Each processing element performs a self-test and invokes reconfiguration (by generating reconfiguration requests) when it detects a self fault.

It is assumed that a central processor is linked to the array, which is capable of controlling the clock pulses to the array. In the event of a detected failure, the central processor is informed about the failure and it delays the clock pulses (as will be explained in chapters 3 and 4).

The processing element which detects the fault informs the neighbouring processing elements and switches about the fault occurrence. These neighbouring elements and switches generate reconfiguration requests again (if required) and inform the other elements and switches. The reconfiguration request keeps on propagating in this manner until it reaches the central processor.

1.1 Thesis Organization

The thesis is divided into six chapters. This chapter gives an introduction to the research topic. Chapter 2 gives an overview of systolic designs and explains various existing reconfiguration schemes and fault detection schemes. In this chapter it is shown that most of the existing reconfiguration schemes cannot be used effectively during run-time. An on-line reconfiguration algorithm for processing element failures is proposed in Chapter 3. In addition, chapter 3 describes the changes (in the design of processing elements, network and switch modules) required for the implementation of this algorithm. In this chapter it is proved that the recommended changes are sufficient to facilitate the reconfiguration. Chapter 4 explains the reconfiguration algorithm for failures in processing elements and links. Various control circuits (for *PEs* and switches) are designed in this chapter. The proposed algorithm is evaluated in Chapter 5 and conclusions are presented in Chapter 6.

Chapter 2

LITERATURE REVIEW

In this chapter, Section 2.1 explains the basic concept of systolic arrays. Section 2.2 describes various fault detection schemes for these arrays and Section 2.3 gives a summary of various well-known reconfiguration schemes and compares them.

2.1 Concept of Systolic Arrays

When memory-based hardware is used, typical Von Neumann bottleneck comes into the picture because of the limited number of interconnections which can be supported by conventional electronics based technology. In memory-based systems, the memory-access time restricts the speed of the system. This can be further explained with the help of the classical finite state machine, shown in Figure 2.1.a. This machine consists of several storage elements, M, a logic unit, inputs (I/P), outputs (O/P) and interconnections. In this scheme all the memory elements are updated and/or read simultaneously in parallel without any addressing. This configuration is not feasible if the number of memory elements is large. So addressing is used to reduce the requirement of parallel lines. This scheme is shown in Figure 2.1.b. Here additional address lines are used and data is fetched in parallel to all the memory elements by a bus and, similarly, a bus carries the output from the memory to the logic circuit.

Here, the number of required lines is reduced but the system performance has

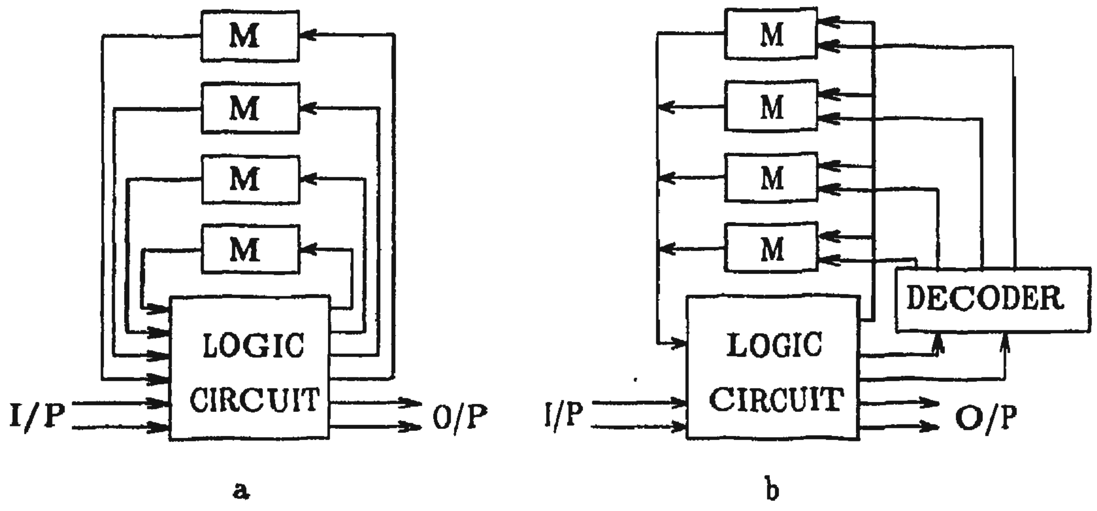


Figure 2.1: Finite State Machine

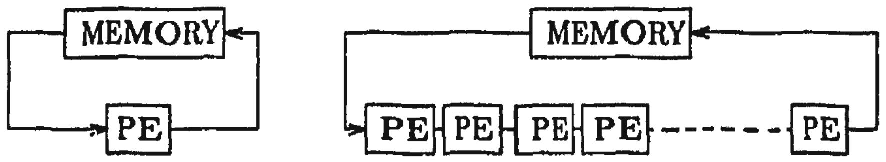


Figure 2.2: Reduced Memory Interaction

degraded because now only one memory element can be addressed at a time; as well an address is required to access the memory locations. This results in an increased memory-access time [1].

The Von Neumann problem can be solved by reducing the number of memory interactions. To explain this, we will consider a processing element which requires at least two memory interactions per operation. If memory access time is 100 ns, we would get a maximum of 5 million operations per second (assuming that the processing element takes negligible time compared to the memory access time). But, if the data are returned to the memory after n such operations, the speed becomes $5n$ million operations per second (see Figure 2.2).

All computations can be classified either under the *compute bound* category

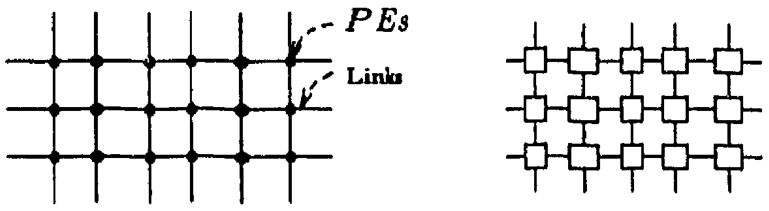


Figure 2.3: Systolic Array Representation

or under the *I/O bound* category of computations. Compute bound computations involve more computations than the required I/O operations (such as matrix-matrix multiplication). In I/O bound tasks, the number of computations is less than the I/O requirements (such as matrix addition).

Systolic arrays reduce the number of memory interactions. In a systolic array, once data are taken out from the memory, they are pumped through a number of processing elements before the final result goes back to the memory. The flow of data in a systolic array resembles the blood flow in the body and the term *systolic* shows the analogy with cardio-vascular biological system. The term *array* is used to show the resemblance of the systolic array to a grid, as shown in Figure 2.3, in which each junction point represents a processing element and the lines represent the links between the processing elements.

Systolic arrays consist of a set of interconnected processing elements, each element capable of performing some basic operations. The data flow in a pipelined manner within a systolic array and communication with the outside world occurs only at the boundary cells [2]. The memory requirement is reduced because the intermediate results are not passed on to the memory. Other than the reduced memory requirements, we get the following advantages:

- modular expandability
- regular and simple data flow

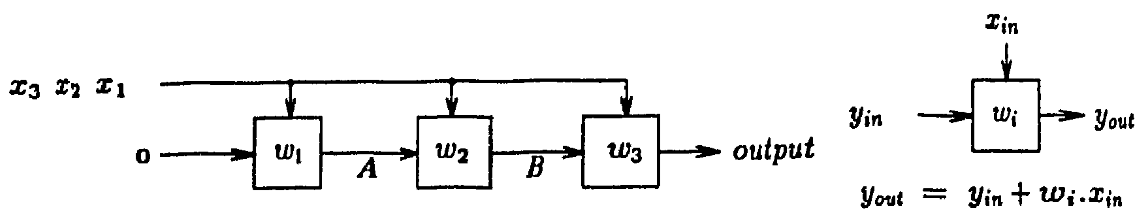


Figure 2.4: Design 1; Broadcast inputs, Results move and Weights stay

- use of simple and uniform cells.

Systolic arrays can be of many types (the types are defined based on the movement of data through the array). Some of the basic types are discussed in the following subsections. Consider a simple computation, given below:

$$y_i = w_1.x_i + w_2.x_{i+1} + \ldots + w_k.x_{i+k-1}, \tag{2.1}$$

where w's are the pre-specified weights and x's are the input data sequence. Here we would take k=3 for simplicity. So,

$$y_i = w_1.x_i + w_2.x_{i+1} + w_3.x_{i+2}. \tag{2.2}$$

Many types of systolic arrays can be designed to accomplish this task [2].

2.1.1 Broadcast inputs, move results and weights stay

The systolic array with this design and basic cell operation are shown in Figure 2.4. In this design, one of the basic criteria of systolic designs is not satisfied, still it works on the same principle that the intermediate results are stored in the array itself. The inputs are broadcast to all the cells at the same time, which is not acceptable for systolic arrays. Due to this shortcoming, this design is classified as *semi-systolic design.*

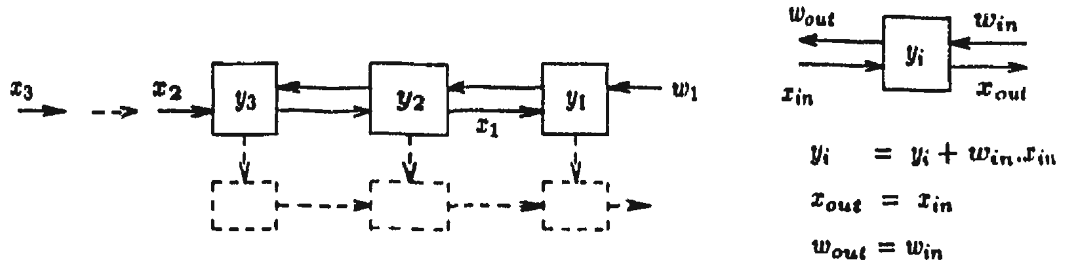


Figure 2.5: Design 2; Results stay, Inputs and Weights move

The data move at the tick of the clock pulse. The data present at the check points A, B and output (shown in Figure 2.4), with reference to the clock are listed below:

CLK.	A	B	Output
0	w_1x_1	w_2x_1	w_3x_1
1	w_1x_2	$w_1x_1 + w_2x_2$	$w_2x_1 + w_3x_2$
2	w_1x_3	$w_1x_2 + w_2x_3$	$w_1x_1 + w_2x_2 + w_3x_3$
3	w_1x_4	$w_1x_3 + w_2x_4$	$w_1x_2 + w_2x_3 + w_3x_4$
4	w_1x_5	$w_1x_4 + w_2x_5$	$w_1x_3 + w_2x_4 + w_3x_5$
5	w_1x_6	$w_1x_5 + w_2x_6$	$w_1x_4 + w_2x_5 + w_3x_6$
6	w_1x_7	$w_1x_6 + w_2x_7$	$w_1x_5 + w_2x_6 + w_3x_7$

We notice that from clock 2 onwards we get one correct output per clock cycle. There are many variations of semi-systolic designs but for the sake of brevity we will not discuss them here.

2.1.2 Results stay, inputs and weights move in opposite directions

This design is shown in Figure 2.5.

It is difficult to implement the previously explained semi-systolic design if the number of cells is large, because of the global broadcast bus requirement.

In this design, consecutive x's and w's are separated by two clock cycles to get the proper results. When x_i and w_i meet at a cell, the cell multiplies them and

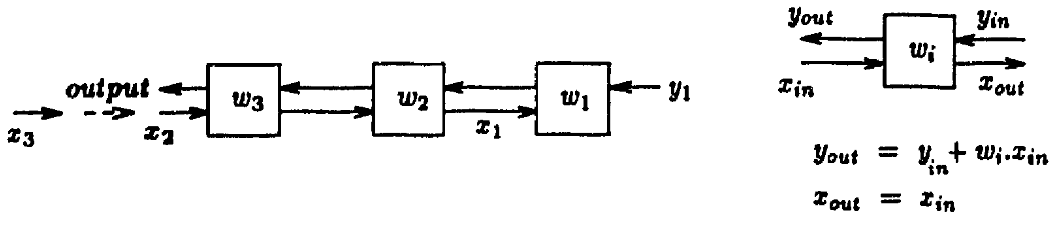


Figure 2.6: Design 3; Weights stay, Results and Inputs move

adds the result to the previously stored result. When w_1 reaches a cell, it outputs the stored values in the cell to the latch (shown below the cell in Figure 2.5) and resets the cell before getting multiplied by x_i. Here the path (shown by broken lines) is used for collecting the final outputs.

Usually, the results and inputs move and the weights stay in the array. This type of array is discussed next.

2.1.3 Weights stay, results and inputs move in opposite directions

This array is shown in Figure 2.6 and here the results and inputs move systolically in opposite directions. This type of design is most suited for on-line arrays and it is used when the same set of coefficients is used to operate on different input data (for example: recursive filtering, polynomial division etc.).

The other types are not discussed here for the sake of brevity.

Systolic arrays can be used for a number of processing operations. These arrays ensure multiple computations per memory interaction. They are particularly suited for FIR, IIR filtering, convolution operations and various matrix operations, like matrix transpose, matrix vector multiplication, matrix matrix multiplication, matrix inversion etc. [2] to [5]. These arrays can be used for any compute bound problem, which is regular (that is, one where repetitive computations are performed on a large set of data).

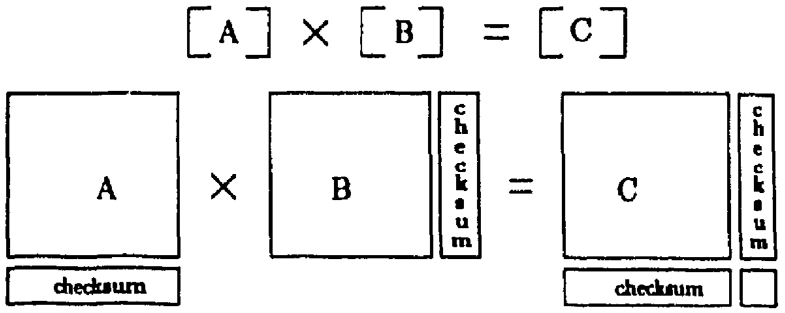


Figure 2.7: Matrix Encoding Method

2.2 Fault Detection Schemes

Systolic arrays are almost always designed to perform special purpose computations, so algorithm based fault detection schemes can be applied to them with very little hardware and time overheads. Some fault detection techniques are discussed in the following subsections.

2.2.1 Matrix Encoding Methods

In this scheme, the matrix is encoded by adding some checksums. Consider the matrix-matrix multiplication shown in Figure 2.7. During encoding, a checksum row is added to matrix A and a checksum column is added to matrix B. After the multiplication, the result matrix, C would have a checksum row and a checksum column. An example of this is given below:

$$\begin{bmatrix} 2 & 4 & 1 \\ 3 & 2 & 4 \\ \{5 & 6 & 5\} \end{bmatrix} \times \begin{bmatrix} 1 & 2 & \begin{Bmatrix} 3 \\ 6 \\ 4 \end{Bmatrix} \\ 2 & 4 & \\ 3 & 1 & \end{bmatrix} = \begin{bmatrix} 13 & 21 & \begin{Bmatrix} 34 \\ 37 \\ 71 \end{Bmatrix} \\ 19 & 18 & \\ \{32 & 39\} & \end{bmatrix}$$

Here, the checksums are shown in curly brackets.

This method can be used for those matrix multiplication arrays where the results stay. An error is detected by checking the checksums and it is located at the intersection of the inconsistent row and inconsistent column. For an $n \times n$ multiplication, an $(n+1) \times (n+1)$ array is required (overhead of $(2n+1)$ cells) [6].

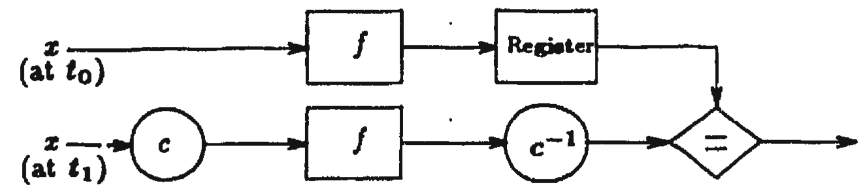


Figure 2.8: Recomputation with Shifted Operands

2.2.2 Recomputing with Shifted Operands (RESO)

Though many codes are available for concurrent error detection in addition and subtraction arrays, they cannot be used for multiplier and divide arrays because they unduly increase the complexity of the circuit.

For such arrays, RESO is an efficient scheme. The basic concept of this scheme is shown in Figure 2.8. First the function f (which is the required operation on the operands) is performed on the data x and the result is stored. The data x is then encoded by c and f is performed on this encoded data. The final result is decoded by c^{-1} and the decoded result is compared with the stored result. Any mismatch in these two values shows the error [7] [8]. Here, the coding c is performed by shifting the operands.

If many operands are used as inputs, then it may not be possible to shift all the input operands equally. In this case the operands can be assumed to be shifted by $k_1, k_2, k_3 \ldots$ and the result obtained by using these shifted operands would be shifted by r bits. This scheme is known as *RESO* $(k_1, k_2, \ldots, r)$ [9].

If E_0 and E_1 are the set of all possible erroneous outputs of $f_F(x)$ and $f'_F(x)$ respectively due to a fault F after the computations, where $f(x)$ is the required function, then the errors are detectable iff $E_0 \cap E_1 = \phi$ (which means that any possible output of the repeated step, $f'_F(x)$ must not be an element of E_0).

The potential error set (as explained in [8]) of the first unshifted computation

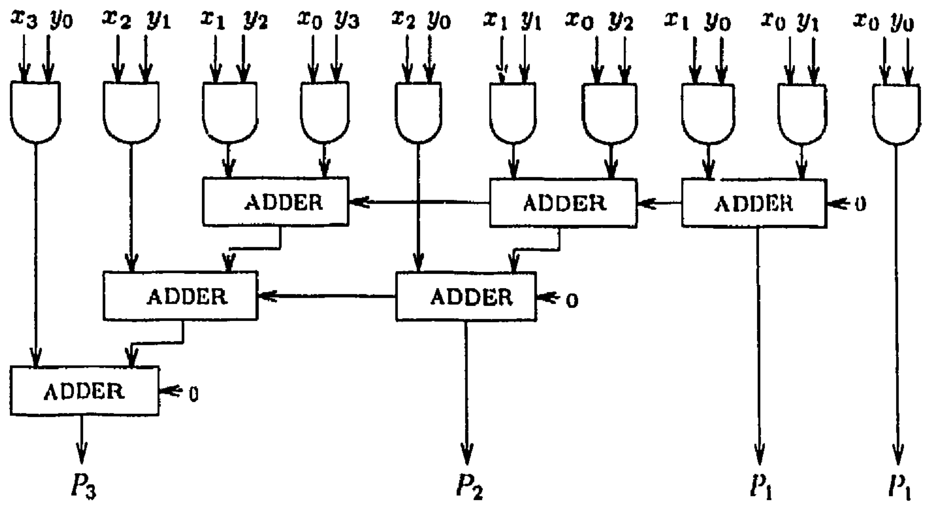


Figure 2.9: Multiplier with Ripple Carry Adder

can be written as:

$$E_0 = \{\pm 2^i \times q | q = 1, 2, \ \ldots, u\}, \tag{2.3}$$

where i is the minimum of the bit slice index of the fault module and u is the maximum error factor (which reflects the integer value of the affected output bit due to the fault). To make it more clear, we can consider the circuit shown in Figure 2.9.

Here, when one adder cell i fails, it tries to change the value of the output. The i^{th} adder chip failure can result in an error in the i^{th} sum bit or the carry bit (which affects the $(i+1)^{th}$ bit). So, there are two possible bits which can be affected and these bits have weights 2^i and 2^{i+1}. This gives four possible combinations:

- both bits correct; error = 0,
- bit 2^i has error, 2^{i+1} correct; error $= \pm 2^i$,
- bit 2^{i+1} has error, 2^i correct; error $= \pm 2^{i+1}$,
- both have error; error $= +2^i \pm 2^{i+1}$ or $-2^i \pm 2^{i+1} = \pm 2^i, \pm 3 \times 2^i$.

So, the result is in error by any one of the error set $\{0, \pm 2^i, \pm 2^{i+1}, \pm 3 \times 2^i\}$

So, for an adder, u=3. In the earlier discussion, we neglected the element 0 of the set, because this identifies a correct output.

In the recomputation step, the result is shifted left by r bits with respect to the original unshifted result. So potential error set of the recomputation is:

$$E_1 = \{\pm 2^{i-r} \times q | q = 1, 2, \ \ldots, u\}. \tag{2.4}$$

Now, the disjointness of E_0 and E_1 can be ensured by making sure that the maximum element in E_1 is less than the minimum element in E_0.

Using this strategy, arrays can be designed, whose faults can be diagnosed by RESO method [9].

2.2.3 Triple Data Redundancy

This scheme uses the basic modular property of systolic array (that all the processing elements, PEs, perform the same operation) to detect (and mask few) errors. It is suitable for one dimensional arrays.

In this scheme, three PEs perform the same computation on the same data at a time and they pass on their results to the next 3 PEs, which compare these 3 results and then perform the computation on the majority-voted input. Since this scheme uses three PEs to perform the same operation on the same data, it can mask the presence of a single fault and detect double faults [10].

The input is given to PE_1, PE_2 and PE_3 simultaneously (Figure 2.10) and they perform their portion of work on this data and then

- PE_1 passes on the result to PE_2, PE_3 and PE_4,
- PE_2 passes on the result to PE_3 and PE_4 and
- PE_3 passes on the result to PE_2 and PE_4.

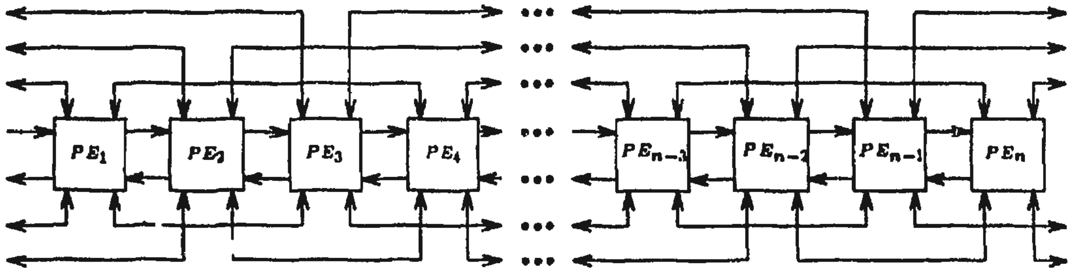


Figure 2.10: Triple Time Redundancy

So, at the next clock pulse, PE_2, PE_3 and PE_4 get three identical inputs (if no fault is present) and each one of them votes on the data and then performs the computation on the voted data.

In the case of a detected error, the PE which detects the error informs the central processor that the data output from PE_x is wrong. After receiving this message, the central processor attaches a flag (indicating a fault in the PE) to PE_x and ignores any further information about PE_x's health. In addition, the central processor maintains a table of the health of all $PE's$. Whenever it receives an error message, it checks the table and if the faulty PE falls within a distance of 2 from another faulty PE, reconfiguration is done by removing 3 PEs from the array. Each reconfiguration removes three PEs from the array. If in the array, shown in Figure 2.10, all PEs are working properly initially and then PE_{n-2} fails, the central processor marks it in the table and next if either PE_{n-1} or PE_n fails, the reconfiguration removes PE_{n-2}, PE_{n-1} and PE_n from the array. Similarly, if in this case (with PE_{n-2} as first failure) either PE_{n-3} or PE_{n-4} fails, the reconfiguration removes PE_{n-4}, PE_{n-3} and PE_{n-2} from the array. So a reconfiguration removes exactly 2 faulty cells and 1 non-faulty cell from the array. The reconfiguration reduces the array size and this necessitates a restructuring of the algorithm executing on the array. So, after every reconfiguration, the full array is flushed and the restructuring algorithm is run. This is done by the central processor [10].

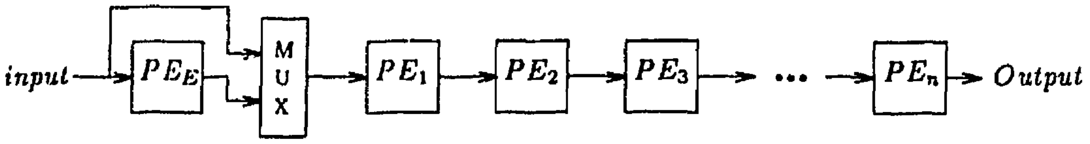


Figure 2.11: Comparison with Concurrent Redundant Computation (CCRC)

2.2.4 Comparison with Concurrent Redundant Computation (CCRC)

This scheme can be used for those systolic arrays, in which the results move and the weights stay in the *PEs* [11]. Here, the same computation is done by PE_i and PE_{i-1} at the same time and the results are compared (Figure 2.11). This algorithm assumes that only one *PE* fails at a time, so if PE_i is faulty, it will be detected by comparing y_i and y_{i-1}.

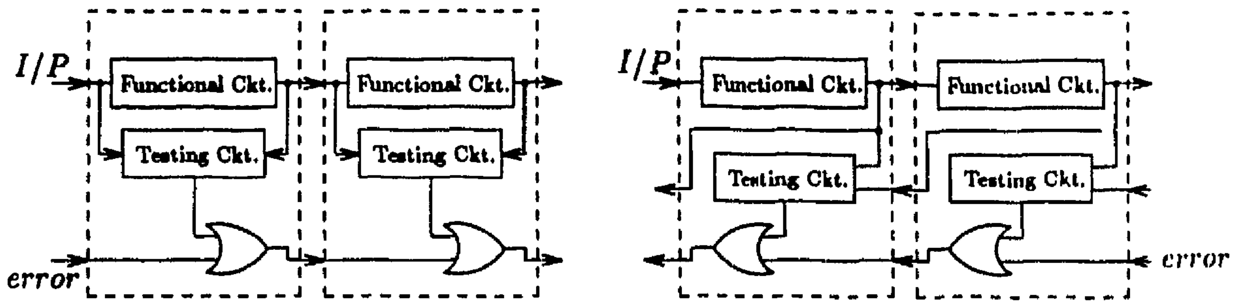
To implement this, the same input is given to the array twice. This can be done in many ways. One of the methods is shown in Figure 2.11. Here, PE_E is the extra *PE*, which is used to introduce the proper delay and calculate the first partial result.

The comparison can be done in two different ways. The scheme, shown in Figure 2.12.a, assumes that even when PE_i is faulty, its comparator is working. This condition is difficult to achieve. The scheme, shown in Figure 2.12.b, does not assume this, but it requires an additional link between the *PEs*.

This scheme generates an asynchronous error signal, which is necessary. In this case the fault is detected even before the error propagates to the output and corrupts the next stage of the system [11].

2.2.5 Double Calculation in the Same *PE*

This scheme is suitable for the systolic arrays, where the results stay in *PEs* and the coefficient and data streams move [12]. In such systolic arrays, the partial



a b

Figure 2.12: CCRC: Comparison Schemes

results stay in the *PEs* and when final result becomes available, it is passed on to the output register from where it is scanned out.

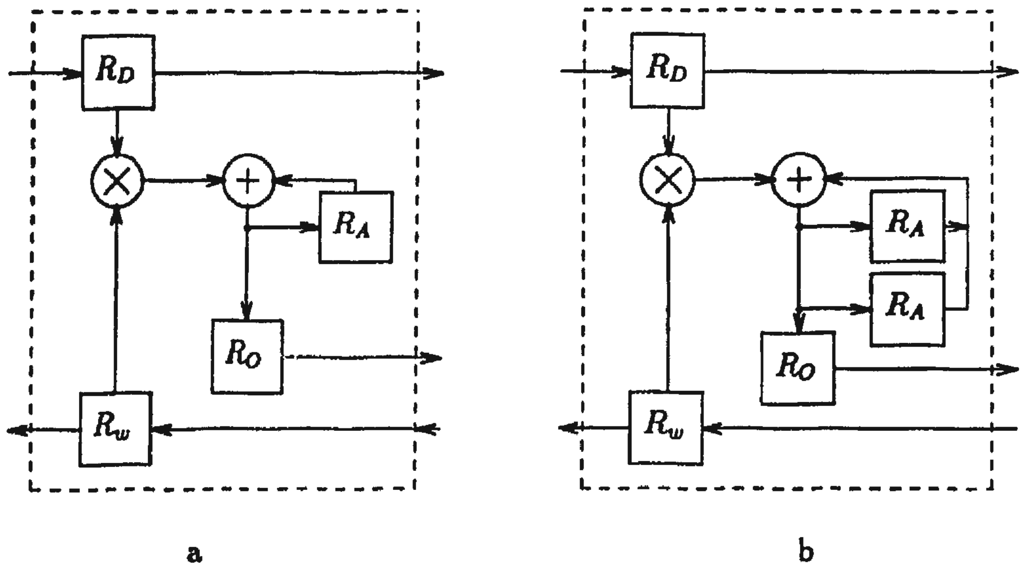
Now, consider an FIR filter:

$$y_n = \sum_{i=0}^{N} a_i x_{n-i}, \tag{2.5}$$

where N is the number of *PEs*.

To implement this filter, we have to separate the adjacent coefficient terms and data terms by two cells. This cell separation feature can be used to get fault tolerance. To add this additional feature, some extra hardware is required. Without fault tolerance, the normal *PE* looks as in Figure 2.13a. A second accumulator is added to store the results of a second calculation, Figure 2.13.b. Each accumulator R_A, feeds the adder and accepts its output during alternate clock cycles. So two independent calculations can be performed in each *PE*. When the calculation of an output term is completed, the adder output is sent directly to the output register R_O, while the accumulator containing the parallel result is reset.

Data flow is shown in Figure 2.14. Two boxes are shown for each *PE* and the content of each box represents the content of an accumulator in the *PE*. In the figure, *ij* means $x_i.a_j$; for an example, 32 would mean $x_3.a_2$.



R_D : Data register R_w : Weight register

R_A : Accumulator R_O : Output Register

Figure 2.13: Recomputation in the same PE

It is clear from this flow diagram that every output is available from two different PEs. These outputs can be compared to detect a fault.

Here, it is not possible to locate the faulty PE, because only 2 copies of the result are available, but whenever a fault is detected, the faulty PE can be located by running some exhaustive checking algorithm.

2.3 Reconfiguration Schemes

Fault tolerance is incorporated in a systolic array to achieve two basic goals:

- to improve the system reliability and
- to improve the yield of VLSI and WSI chip production.

To improve the chip density it is required that the physical dimensions of the transistor level circuitry be reduced making the manufacturing process more error-prone.

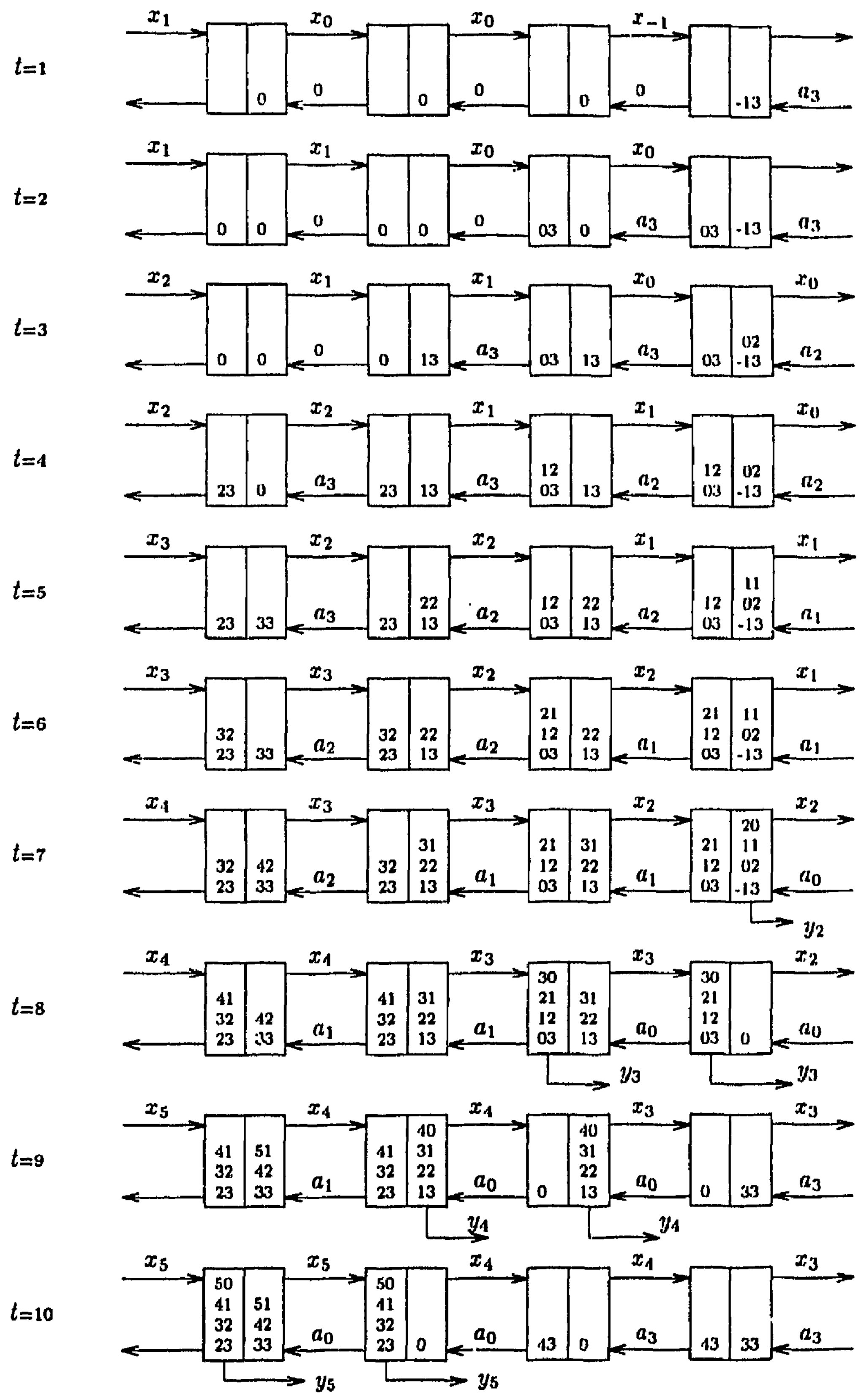


Figure 2.14: Data Flow Diagram (for recomputation in the same PE)

For a typical bulk CMOS process, the following is a brief list of common defects:

- *Photolithography Defect*: it causes missing or extra patterns on a mask layer. Common sources of this are mask defects, dirt particles and uneven etching.

- *Contact and Via Defects*: these are the windows between different layers for providing interlayer connections. The defects in these can result in shorter/larger window area causing the shorting of other connections.

- *Gate Oxide Defect*: charge trapping in gate-oxide regions of MOS devices results in threshold voltage shifts which can lead to reduced noise margins and malfunctioning of gates.

Because of these reasons, the production of VLSI/WSI chips does not always give a yield at an acceptable level. To improve the yield, the chip is designed to be fault-tolerant [13].

To achieve fault tolerance we have to provide redundancy, which can be of two types:

- *hardware redundancy*: in this case, spare cells and the corresponding interconnection network are provided and in the case of a fault, reconfiguration is done.

- *time redundancy*: here, the processing elements are provided a number of processing states. Working elements perform the functions of faulty cells if any fault occurs. In this case, the number of elements does not increase but the interconnection network becomes very complex. Also, the processing speed decreases drastically, so it is not suitable for systolic arrays.

Usually, hardware redundancy is provided in an array and in case of a fault, reconfiguration is done. The goal of the reconfiguration is to achieve 100% spare utilization (i.e. if N spare cells are available, the array should survive up to N faults).

In discrete system architecture, 100% spare utilization is possible and also desirable because here the cost of the processing element is much higher than that of interconnection network and usually in this case each processing element is a CPU, so the re-routing can be performed by one of the working *PEs*. If the CPU is an extremely simple device (which can not perform the re-routing), the reconfiguration is not needed because in this case the reliability of the system will be extremely high due to the simple CPU design.

In the case of a systolic array, though the utilization of spare cells is still important, it is also necessary to maintain the locality of interconnections. Here, it is essential to use simple routing devices to minimize the time delays and silicon area (it has been proved that excessive increase of chip area due to fault tolerance related circuits has a negative effect on overall device reliability).

So, for a systolic array, the reconfiguration process has to provide a compromise between the reconfiguration-effectiveness and algorithm complexity. This compromise depends on the approach adopted for the reconfiguration, namely:

- static reconfiguration, performed at production time,
- dynamic reconfiguration driven by a host computer at run time and
- dynamic reconfiguration, performed on-chip at run time.

Static reconfiguration is uniquely determined at production time and for this the testing is performed externally (so no on-chip control circuitry is required). The complexity of the reconfiguration algorithm is not a critical issue because it does not affect either circuit complexity or operation speed.

For the second case, it is assumed that the external host can perform reconfiguration-controlling actions on the basis of the available error information.

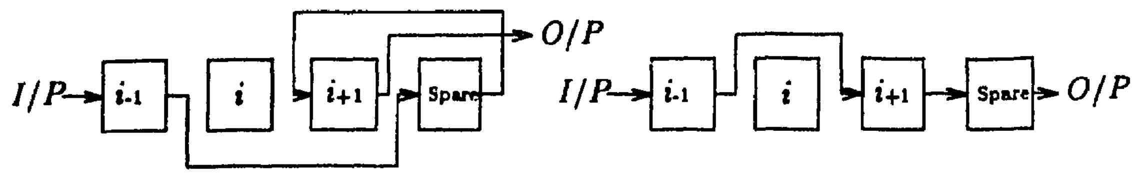


Figure 2.15: Direct Replacement and Global Deformation

The third case introduces additional costs for self testing and self reconfiguration.

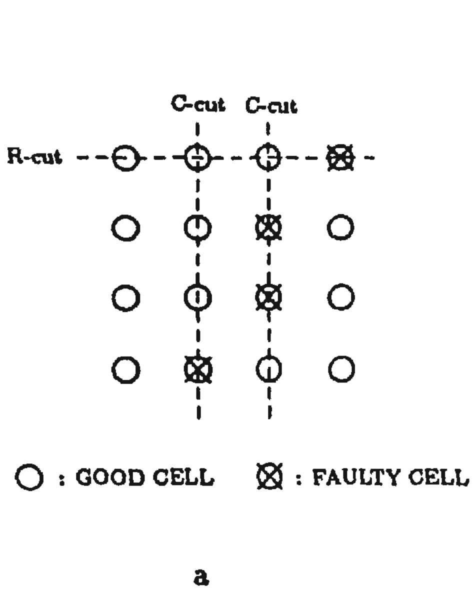
For all the dynamic reconfiguration algorithms, the problem of error-latency (defined as the time that passes before the array is operational again after the occurrence of a fault) has to be considered. Any reconfiguration approach involves two problems: the first problem is that of routing data through the reconfigured array. It involves introduction of redundant links and routing devices. The locality of the interconnection network is maintained by using the global deformation technique in place of the direct replacement technique. In the global deformation technique if cell i is faulty (see Figure 2.15), cell $(i+1)$ assumes the role of cell i and cell $(i+2)$ performs the functions of cell $(i+1)$ and so on. The spare cell performs the function of cell N. In the direct replacement technique, the spare cell has to perform the function of cell i and this disturbs the uniform data flow assumption of the systolic array.

The second problem is that of the reconfiguration computation as related to fault distribution. It involves the implementation of the reconfiguration algorithm [14].

An $M \times N$ faulty array is said to be reconfigurable into an $m \times n$ array iff m horizontal and n vertical data flow paths can be achieved by reconfiguration.

There are two major types of reconfiguration schemes:

- *Set Switching Schemes*: here, a faulty cell is replaced by logically removing a set of cells (row, column, block etc.), that contains the faulty cell. It is easily



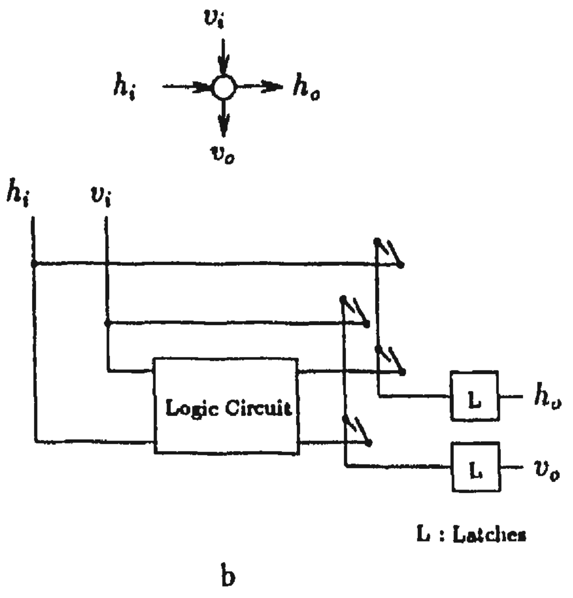


Figure 2.16: Row Column Cut Method

implemented but the waste of non-faulty cells is laıge.

- *Processor Switching Schemes*: here the replacement scheme proceeds in a chain fashion such that a faulty cell is replaced by (shifted to) an immediate neighbour and so on until the spare cell is reached [15].

Various reconfiguration schemes are discussed in the following sub-sections.

2.3.1 RC Cut (Row Column Cut) Method

A cut is defined as a set of cells, such that bypassing them leads to an array with one less data flow path. A horizontal (vertical) cut removes one horizontal (vertical) data flow path from the original array. Horizontal (vertical) cut is also called row (column) cut.

In this method, for a faulty cell all the cells in the same row/column are taken to be in a cut. So, in the array, shown in Figure 2.16.a, one horizontal and two vertical paths are involved in cuts. This results in a reconfigured 3×2 array from a 4×4 array.

The routing arrangement is shown in Figure 2.16.b. It is clear from the figure that any cell can be bypassed by simple switch controls. The architecture and path generation are simple but this algorithm wastes a large number of non-faulty cells. Particularly, for a large array (suppose a 10×10 array), the failure of just one cell removes a large number of cells (in this case 10) from the array [16].

2.3.2 RCS (Row, Column Slanted) Cut Method

This is also known as Kung and Lam approach. Here, the cells contributing to a cut may not be from the same row or column but they satisfy the following conditions:

- a cut must contain one cell per row (vertical cut) or one cell per column (horizontal cut) and the slope of the line containing the cells in the cut must be non-negative and
- the inclination of the line connecting the cells in the cut between the successive columns must be 0 or 45 degrees for horizontal cuts and 90 or 45 degrees between successive rows for vertical cuts.

One such vertical cut is shown in Figure 2.17.a. Here, the 4×4 array (used as an example in RC-cut subsection) is reconfigured into a 4×3 array. The routing arrangement for an RCS cut is shown in Figure 2.17.b. It is clear that the utilization of cells is improved in this method, but the routing complexity is also increased. It is difficult to get an optimum cut in this method and for fewer faults this scheme also wastes a large number of non-faulty cells [16].

2.3.3 Kuo-Fuchs Method

Now, consider the $(7+2) \times (9+3)$ array shown in Figure 2.18.a. Here $(7+2) \times (9+3)$ means that it is a 7×9 array, having seven rows, R_1 through R_7 and nine columns, C_1 through C_9, with two spare rows, S_{R1} and S_{R2}, and three spare columns, S_{C1}, S_{C2} and S_{C3}). Only faulty PEs are shown in the figure.

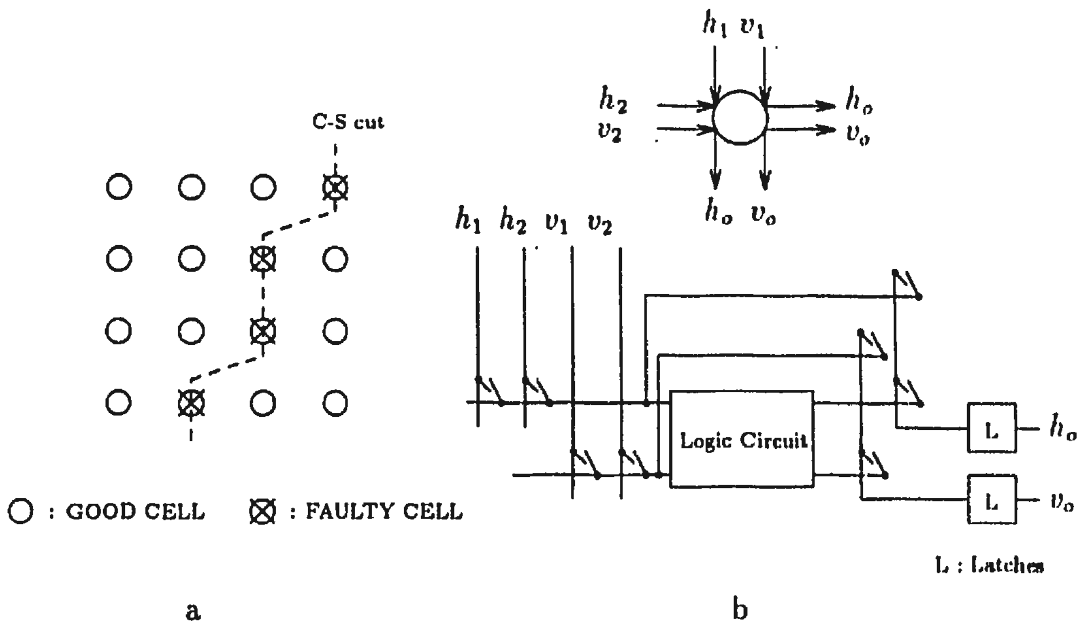


Figure 2.17: Row Column Slanted Cut Method

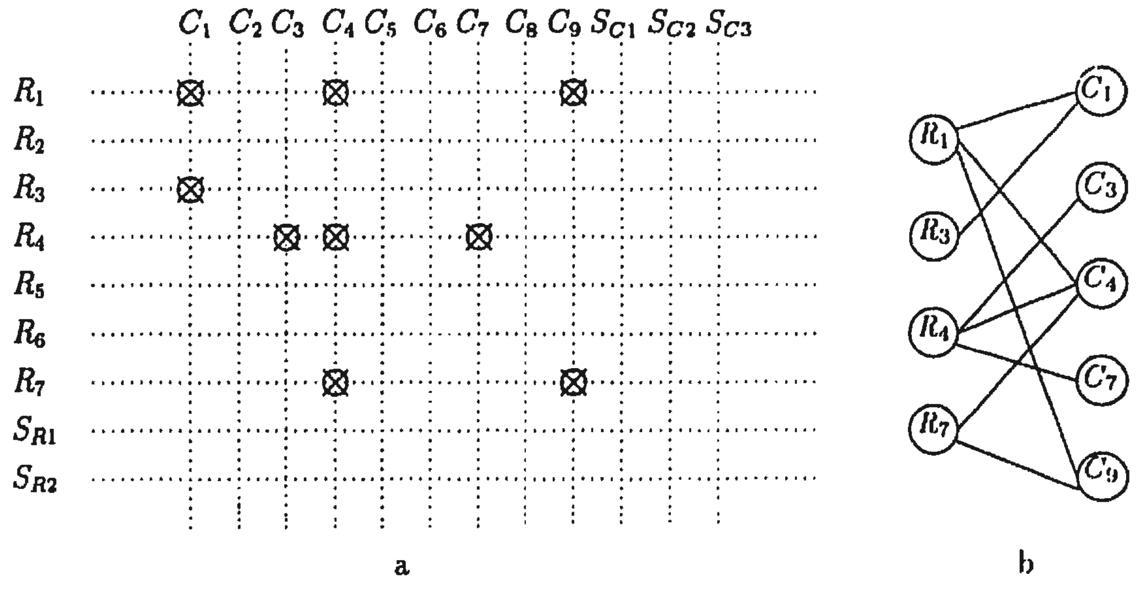


Figure 2.18: Kuo-Fuchs Method

A general set replacement algorithm replaces faulty rows/columns by proceeding from left to right and top to bottom - so, rows 1 and 3 would be replaced by the spare rows and columns 3, 4 and 7 would be replaced by the spare columns. Obviously, it does *not* reconfigure the array.

In the Kuo-Fuchs method, the rows/columns that contain the maximum number of faulty cells are replaced first. To implement this, the array is modelled as a bipartite graph, whose two sets of nodes are array rows and columns that contain faulty cells. Edges of this graph refer to the faulty cells. The bipartite graph of the $(7+2) \times (9+3)$ array is shown in Figure 2.18.b.

This method first chooses the nodes with maximum number of branches and replaces them. Here, first R_1 and R_4 are replaced with spare rows and then C_1, C_4 and C_9 are replaced with spare columns. This achieves a successful reconfiguration [17].

In all the above-mentioned schemes the utilization of non-faulty cells is very poor. Next, some processor switching schemes are discussed, where an available spare cell directly or indirectly replaces a faulty cell. Because of this, for these methods, the reconfiguration efficiency is good.

2.3.4 Diogenes Method

In this approach the array is laid out in a line with bunches of wires, called *bundles*, running above the line (the *PEs* need not literally lie in a line), as shown in Figure 2.19.

Each *PE* has some number of lines entering it (connecting it to the *PEs*, that lie to its left in the line) and some number of lines leaving it (connecting it to the *PEs*, that lie to its right in the line). These entering and leaving sets of lines are connected to the bundles through switches that are set by external control. The *PEs* are scanned in a row and the faulty *PEs* are not connected to the bundle. So, the utilization of the spares is maximized.

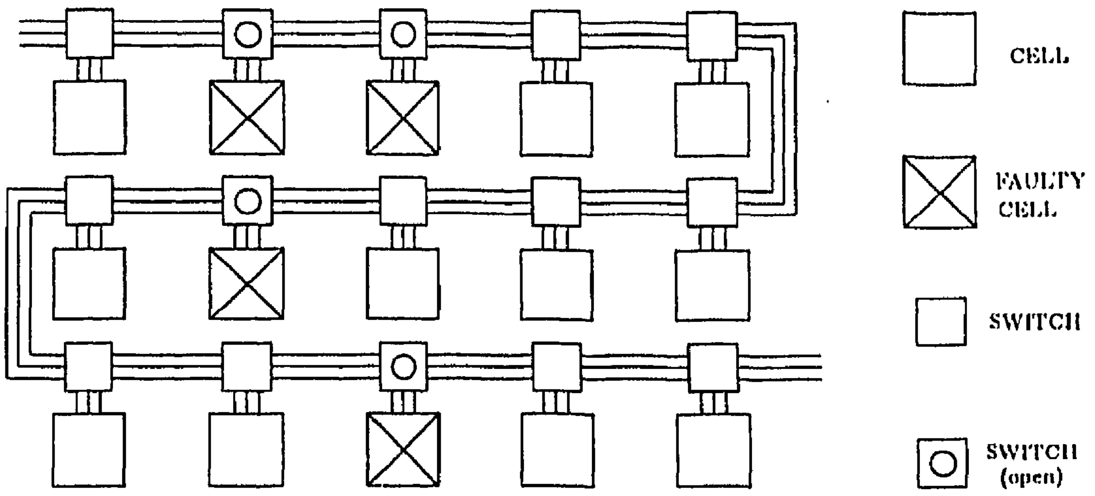


Figure 2.19: Diogenes Method

In this method, the *PEs* are tested first and the outcomes of the tests are available to the buses via control lines $GOOD_i$ that indicate the presence or absence of fault in the i^{th} *PE*. If PE_i is fault free, the corresponding control line would be high and PE_i would be hooked to the bundle. A *PE* is hooked to the bundle only if the corresponding line, $GOOD_i$ is high. This feature facilitates the testing also. Any *PE* can be isolated and tested by setting its $GOOD_i$ line to '1' and other $GOOD_i$ lines to '0'.

This scheme requires a large silicon area for the switch bus that might itself fail. In the presence of consecutive faulty *PEs*, logically adjacent *PEs* can be far apart physically, reducing the system speed [18].

2.3.5 Fault Stealing Methods

These are also known as index-mapping schemes. Here, for an array of $M \times N$ cells, the spares are organized along the $(M+1)^{th}$ row and the $(N+1)^{th}$ column. Reconfiguration is performed by mapping the array functions onto the working cells by means of a global renaming process. Whenever a given algorithm does not complete this mapping onto correctly working cells, a *fatal failure* condition is said

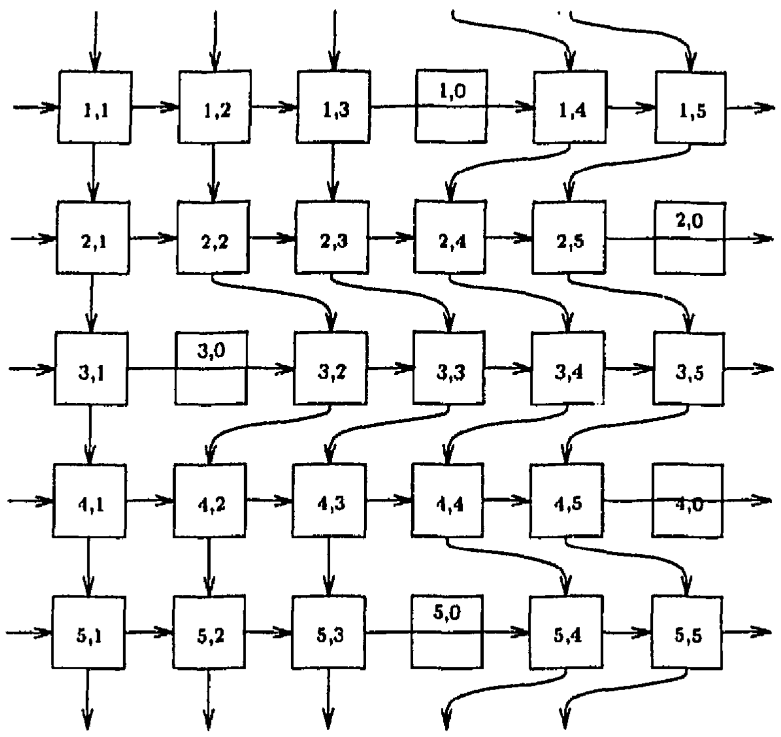


Figure 2.20: Simplest Fault Stealing Method

to occur.

In this scheme, the physical and logical indices are defined first for each cell. The physical indices (i, j) denote the position in the physical array consisting of all cells and the logical indices (i', j') denote position in the logical array consisting of working cells only and implement all the functions required by the array.

Consider the simplest case, in which a spare column is added. If a cell (i, j) is faulty, it is bypassed and logical indices (i', j') are associated with cell $(i, j + 1)$ for cells $(i, k), k > j$. Figure 2.20 shows the result of one such reconfiguration. The fatal failure condition is reached whenever there are two faulty cells in a row. This problem can be overcome by adding one spare row and one spare column to the array and slightly modifying the algorithm. The modified algorithm is as follows:

- the array is scanned from top to bottom

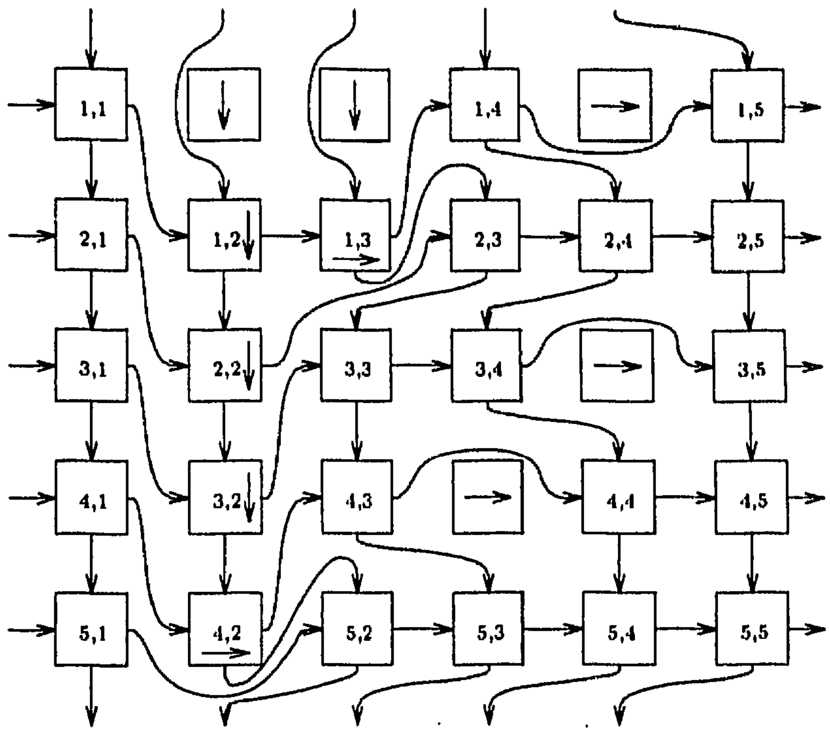


Figure 2.21: Modified Fault Stealing Method

- if in row i there is only one faulty/stolen cell, rightward reconfiguration is performed for that row,
- otherwise, the rightmost faulty or stolen cell invokes rightward reconfiguration, while all other faulty or stolen ones steal the functions of cells in the corresponding positions of row $(i + 1)$ making them stolen cells. Stealing by (i, j) implies associating logical indices (i, j) with the stolen cell.

Figure 2.21 shows one such reconfiguration.

In this case, a fatal failure condition is reached when a stolen cell is faulty. The locality is high in this case also. Here, a faulty cell (i, j) can be shifted to a fault free cell $(i, j+1)$ or $(i+1, j)$ The set consisting of cells (i, j), $(i, j+1)$ and $(i+1, j)$ is referred as an *adjacency domain.* This adjacency domain can be extended and the algorithm can be modified to get more spare utilization. The modified approach is called *complex fault stealing* [14].

2.3.6 CFS (Complex Fault Stealing) Method

In this scheme, a spare row and a spare column are provided to the $N \times N$ array and the algorithm is as follows:

- assume that in row i, $1 \leq i \leq N$ there are faulty or stolen cells (i, k_1), ..., (i, k_s)
- for each $k_i, 0 < i < s$:

  a- if $(i+1, k_i)$ is fault free, (i, k) is shifted to it,

  b- else, if $(i+1, k_i+1)$ is fault free, (i, k) is shifted to it,

  c- otherwise, (i, k_i) is shifted right.

- if no cell is shifted right along the row as per the previous rule, then (i, k_s) is shifted right. Otherwise (i, k_s) is shifted downwards to either $(i+1, k_s)$ or $(i+1, k_s+1)$.

An example of this algorithm is shown in Figure 2.22. Here, $(1,2)$ is shifted twice, first to $(1,3)$ and then to $(2,4)$. The interconnection links required by this algorithm are very complex [15] [19].

2.3.7 FUSS (Full Use of Suitable Spares) Method

This scheme uses an indicator vector, called the ***surplus vector*** to guide the replacement of faulty cells in an array. In its ideal case, FUSS achieves 100% spare survivability. In FUSS-C, the array is an $M \times (N+C)$ array, where C is the number of spare columns (spare rows are not used). First, the surplus vector of the array is computed. Let f_i be the number of faulty cells in row i. The surplus vector (S-vector) is defined as

$$s = [s_1, s_2, \ldots, s_M]^T,$$

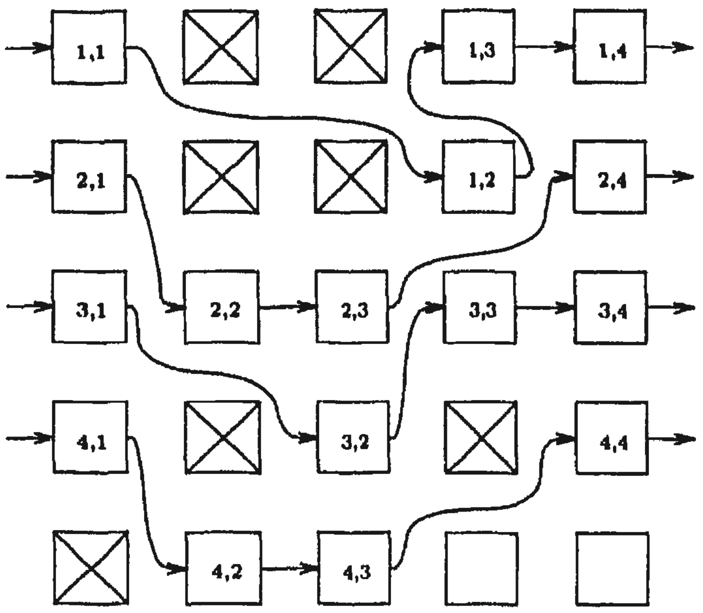


Figure 2.22: Complex Fault Stealing Method

where $s_i = \sum_{j=1}^{i}(C - f_i)$ is the surplus of row i.

Next,

- if $s_i > 0$, then the sum of spares in rows 1 through i is greater than the number of faulty cells in row 1 through row i; so row i has extra cells available for use by faulty cells in rows $i+1, i+2, \ldots M$,
- if $s_i < 0$, then row i has a deficit and needs to use available cells from row $i+1, i+2, \ldots, M$,
- if $s_M < 0$, then the total number of spares in the array is less than the number of faulty cells. In this case the array is not reconfigurable and fatal failure occurs.

In FUSS-C, an unavailable cell (i, j) can be shifted down to $(i+1, j)$ if s_i is negative or shifted up to $(i-1, j)$ if s_{i-1} is positive.

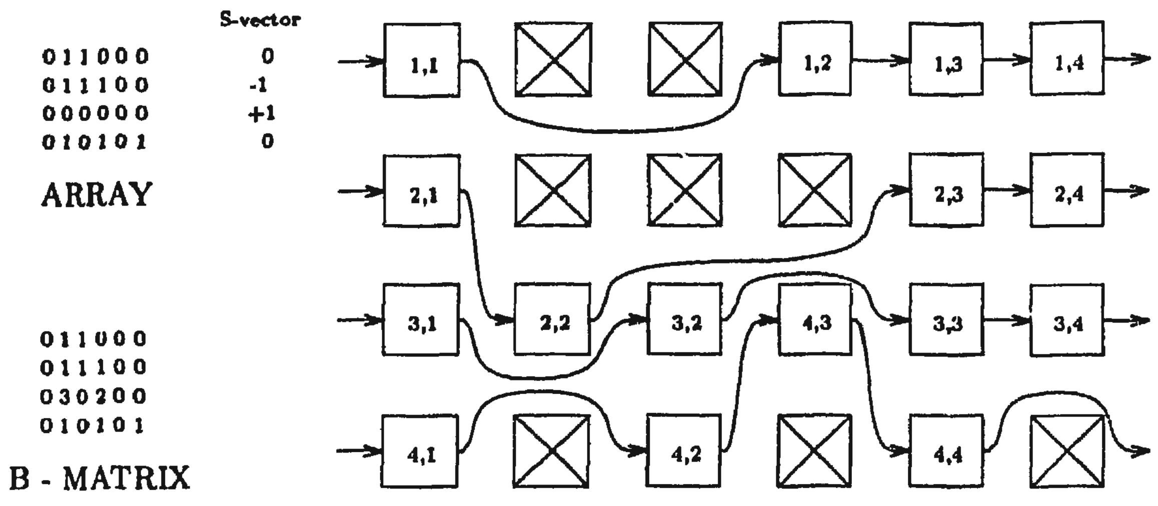


Figure 2.23: FUSS Scheme

After each step the corresponding entry in the surplus vector is readjusted towards zero. Its effect can be described as a cell migration from regions having most faulty cells to regions having less faulty cells.

Consider a $4 \times (4 + 2)$ array shown in Figure 2.23 (FUSS-2 Scheme), where '0' represents a good cell and '1' represents a faulty cell. The reconfiguration is executed as follows:

- scan the array downwards. When $s_i < 0$, shift a number equal to $|s_i|$ of unavailable cells to row $i+1$ and when successful, reset s_i to 0. Here, $s_2 = -1$, so one cell $(2, 2)$ is shifted down from row 2 to row 3 and this is assigned a status code of 3,

- scan the array upwards. When $s_i > 0$, shift $|s_i|$ unavailable cells in row $i + 1$ to row i; s_i is reset to 0 when all s_i cells are shifted successfully. Here, $s_3 = 1$, so one cell from row 4, cell $(4, 4)$ is shifted up to cell $(3, 4)$ which assumes the status code of 2; s_3 is readjusted to 0.

Now, the surplus vector is 0, which means that the fault shifting is successful. The

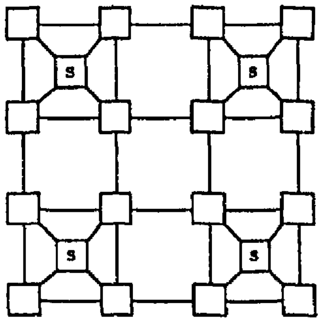


Figure 2.24: Interstitial Redundancy Scheme

status matrix (B-matrix) has the status codes that guide the cell interconnection phase of FUSS. Entry b_{ij} has the following meaning:

- $b_{ij} = 0$, if (i, j) is fault-free
- $b_{ij} = 1$, if (i, j) is faulty
- $b_{ij} = 2$, if (i, j) replaces $(i + 1, j)$ and
- $b_{ij} = 3$, if (i, j) is replacing $(i - 1, j)$.

Now, since the status of the cells is known, it is easy to derive the interconnection between the cells. In this algorithm, the probability of survival improves and fewer cells are wasted. However, the algorithm becomes more complex and the interconnection requirement is increased.

2.3.8 Local Redundancy Methods

In these schemes, the array is partitioned into smaller arrays, each of which can be reconfigured independently. The main objective of these schemes is the minimization of the interconnection delays. One such scheme is discussed next.

The scheme is called *interstitial redundancy* and it maintains short interconnection links.

The array is divided into a number of subarrays (clusters) and one spare is allocated to each cluster. The array shown in Figure 2.24 has 25% redundancy. Each cluster is independent and it can tolerate one faulty cell. The spares are physically close to the faulty cell they replace [20].

In these schemes, if reconfiguration is not possible within a block, the system fails unless the faulty block can be replaced by a functional one. To avoid this, the array can be organised in a hierarchical way. One such scheme is CHiP (configurable highly parallel) architecture, made up of building blocks, each of which is a two dimensional CHiP array [21].

The cut methods are simple but they are not efficient. In the slanted R-S cut method, sometimes it is difficult to get an optimum cut. The switching scheme is very simple for these methods.

The fault stealing and FUSS methods are very efficient but their algorithms and switching structures are complex.

Some of the above schemes cannot be used effectively during run-time because every time a fault occurs, the full algorithm has to run and it may completely change the previous reconfiguration. These algorithms are suitable for improving the production time yield.

In the next chapter an on-line reconfiguration scheme is proposed for PE failures.

Chapter 3

ON LINE RECONFIGURATION

On line reconfigurat. ›n is performed to increase the reliability of the system for the full duration of a mission. Here, in the case of a fault detection, the array is not flushed as required by the previous algorithms.

The reconfiguration scheme should be capable of:

- *fault detection*: if the fault is not detected, the array fails and this failure cannot be detected by the central processor; this is an *unsafe failure*;
- *fault location*: the fault location is important in order to replace the faulty PE by a non-faulty PE;
- *re-routing*: the scheme should be capable of mapping the new logical index on to the physical index and
- *fault blocking*: to ensure that the faulty data are not passed on to the next PE, otherwise all the further computations would use the faulty data and all the results would be faulty.

A major concern for an on-line reconfiguration is *complete use of non-faulty partial results.* During reconfiguration the fault-free partial results should be handled properly.

The reconfiguration scheme should have following attributes:

- *simplicity of algorithm*: the algorithm should be simple, so that it causes little *disturbance* in the array. Here, disturbance refers to the total number of PEs, for which the logical index changes.
- *minimal additional hardware*: any additional capability requires some extra hardware, which depends on the algorithm. The algorithm should use minimum additional hardware otherwise the additional hardware would bring down the array reliability instead of improving it.
- *use of fault-free partial results*: in systolic arrays, partial results are passed on to the next cell as input. In the case of a fault-occurrence, the faulty partial results should be blocked and the fault free partial results should be ideally utilized to best advantage.
- *locality*: the locality of the data is one of the major attributes of systolic arrays and the reconfiguration algorithm should maintain it. It is maintained by using the *global deformation* instead direct replacement.

A scheme is proposed in the following section for on-line reconfiguration that has these attributes.

3.1 On-Line Reconfiguration Scheme

This scheme does not perform any on-line testing, so self-testing PEs are required. When a PE detects any fault, it invokes the reconfiguration. The following assumptions are made.

Assumptions:

- the faults are occurring one at a time;
- the links and the switching network are fault-free;

- once a fault occurs, it is detected by the PE;
- the control circuitry of PEs never fails;
- a central processor provides input and clock to the array and it receives output and fault occurrence signals from the array and
- the occurrence of a failure is reported to the central processor before the arrival of the next rising clock edge.

The array is provided with an extra row of PEs (called spares) and these spares do not perform any useful operation during the normal operation. These cells do only self-testing and remain non-active for other operations. Once a $PE_{i,j}$ ($PE_{i,j}$ denotes the PE whose physical index is (i,j) and $PE^{L}_{i,j}$ denotes the PE with logical index (i,j)) detects a fault, it marks itself as bad and the reconfiguration is done as follows:

- if $PE_{i,j}$ is a non-active spare, no shift is done;
- if a working $PE_{i,j}$ fails and the spare cell, $PE_{row,j}$, is available, $PE_{i,j}$ invokes a downward shift;
- else a fatal failure occurs.

For example, if in the array shown in Figure 3.1, $PE_{3,2}$ fails, no shift is performed but it is marked as a bad PE. But when $PE_{2,1}$ fails, it checks the availability of spare cell, $PE_{3,1}$, and since this spare is available, the reconfiguration is done and a downward shift is performed for all $PE_{x,1}$, where $i \leq x \leq row - 1$ (here, i=2 and row=3). After this failure, if any PE fails in column 1, the algorithm cannot tolerate the fault and a fatal failure occurs.

Similarly if a spare, such as $PE_{3,1}$, fails first, then any further failure in column 1 would result into a fatal failure.

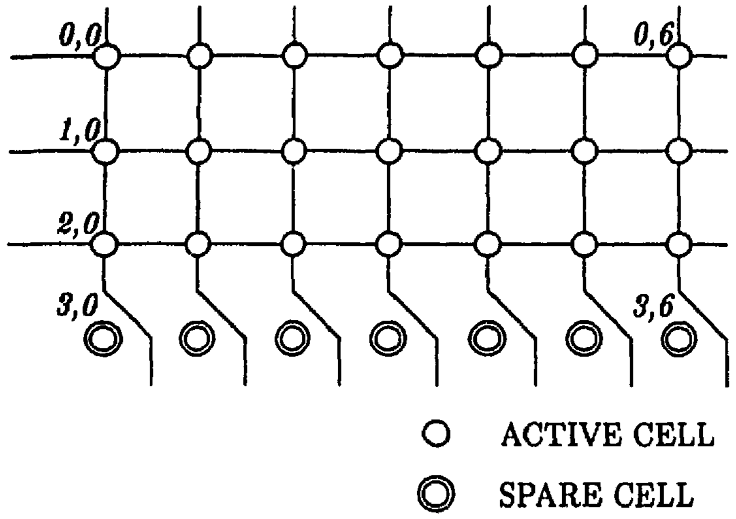


Figure 3.1: Proposed On-Line Reconfiguration Scheme

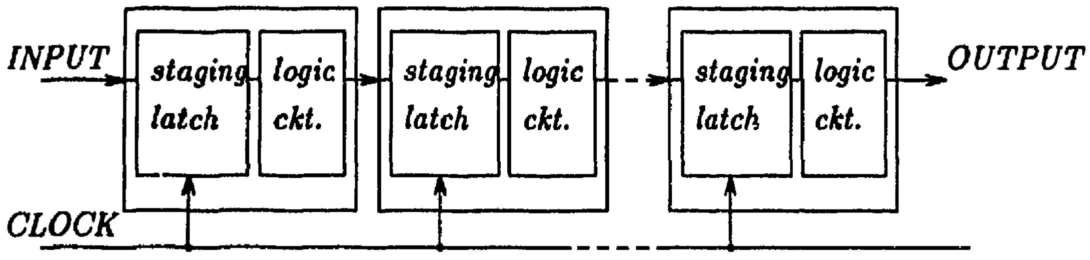


Figure 3.2: Staging Latch Position in Normal Arrays

3.2 Implementation

In most systolic arrays, staging latches are provided at the input end of the PE, as shown in Figure 3.2.

The clock is applied to these latches for propagation of data. When a clock edge arrives, PE_i latches the data from PE_{i-1} and it is available to PE_i for the full duration of a clock pulse.

Now, consider the one-dimensional pipeline shown in Figure 3.3.

During normal operation, each PE_i gets input from the output of the previous PE_{i-1}. Here, inputs and outputs are written as I_x^t and O_x^t, meaning that I_x^t is the

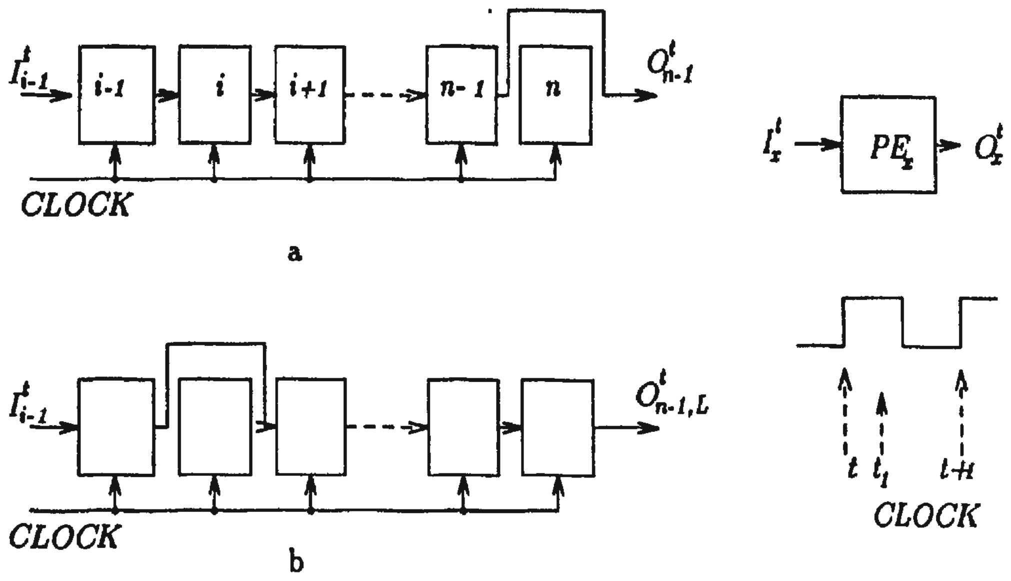


Figure 3.3: Pipeline, Before and After reconfiguration

input of PE_x at time t and O^t_x is the output of PE_x at time t. Similarly, $I^t_{x,L}$ and $O^t_{x,L}$ denote the input and output of PE^L_x at time t respectively. For the pipeline, shown in Figure 3.3, at any time t, $I^t_1 = O^t_0$, $I^t_2 = O^t_1$ ...and so on. At any time t_1 ($t < t_1 < t+1$), each PE_i is processing the data, which was available at its input at time t. Since we have the staging latches at the input end, the failure of PE_i at time t_1 makes the data available on link L^{i+1}_i (the link between PE_i and PE_{i+1}) erroneous. If a spare is available at the rightmost position of this pipeline, a rightward global shift is performed and the pipeline would look as in Figure 3.3(b).

Now, PE_{i+1} acts as PE^L_i and since the partial result generated by PE_i is faulty at time t_1, it must be recomputed by PE^L_i. For generating $O^{t_1}_i$, PE_{i+1} requires the same input, which was available to PE_i at time t, but this data is not available at t_1 because at time t it was generated as O^t_{i-1} by PE_{i-1} and after the clock edge the PE_{i-1} receives new input I^t_{i-1} and changes the output.

To overcome this problem, the staging latches are shifted from the input side to the output side and the new pipeline is shown in Figure 3.4.

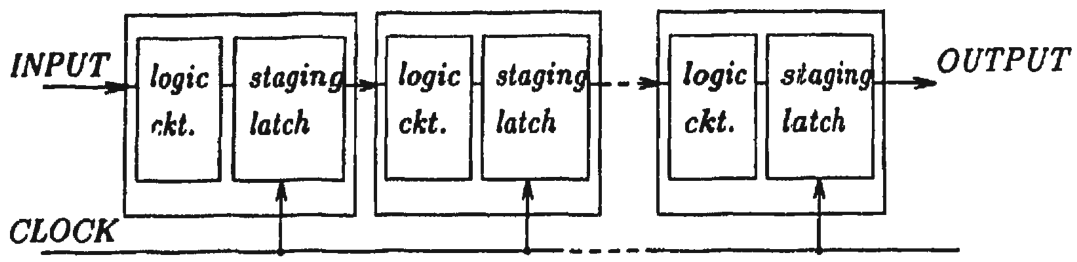


Figure 3.4: Modified Staging Latch Position

In this case, the links and output ports never carry the faulty data, because the moment a fault is detected by any PE, the PE requests the central processor to block the clock. Here, in the case of PE_i failure at time t_1, O^{t}_{i-1} is available at the output port of PE_{i-1} and it can be used by the PE^{L}_{i}. Once PE_i changes its logical index, it has to use the weight (static coefficient), which was being used by PE_{i-1}. This is discussed in the next subsection.

3.2.1 Loading of Weights

When an array is implemented, it is not possible to connect all the static coefficient latches to the external ports (which are used to connect the array and the central processor) due to extensive link requirements. So usually the input line is used to load the static coefficients in the array before the array begins processing data. In most systolic arrays, one of the data streams (either vertical or horizontal) passes through the array without getting modified and this feature is used to load the static coefficients. In the following discussion, it is assumed that the vertical data stream is not modified. This can be generalized to the horizontal data stream also. A simplified block diagram of a PE is shown in Figure 3.5.

We can use either of the following two methods for loading static coefficients in the array.

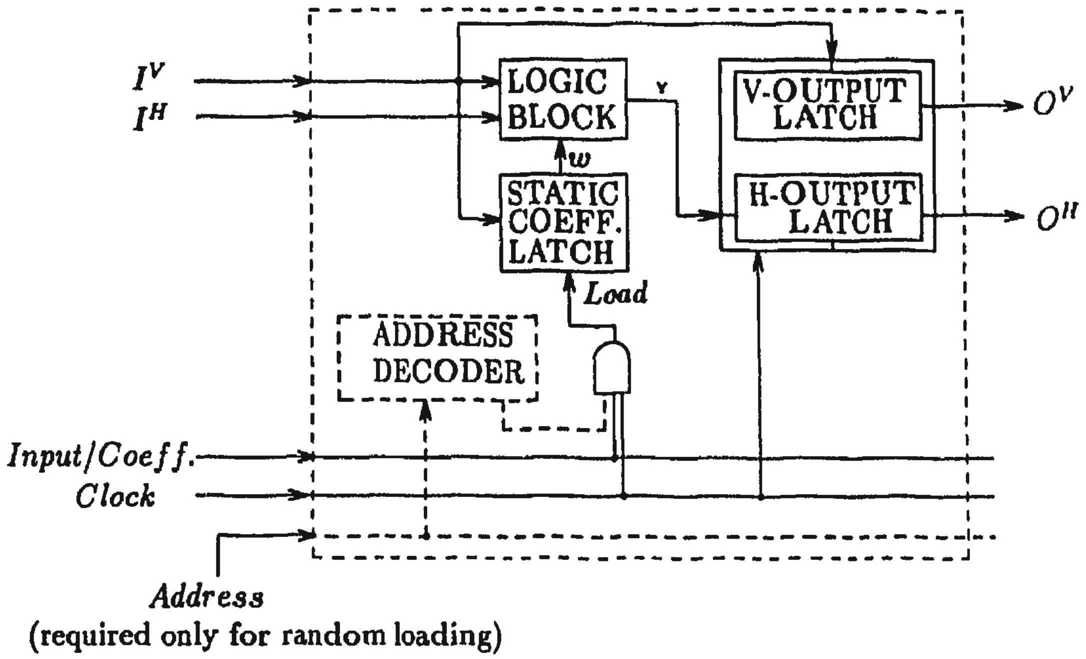


Figure 3.5: Block Diagram of PE (with emphasis on Coefficient Loading Circuit)

Method 1 (Sequential Loading) -

Here, the coefficients, $w_{i,j}$ are loaded into $PE_{i,j}$ by presenting $w_{i,j}$ on vertical input line I_j^V in sequence $w_{m-1,j}, w_{m-2,j}$ ... and after $m-1$ clock pulses (m is the total number of rows in the array, one bottom row of spares is added - making the total number of rows $m+1$), each $PE_{i,j}$ would have its static coefficient $w_{i,j}$ at its input port. Now the *input/coefficient* line is made valid for coefficient (informing the PEs that the data available at their vertical input port is the static coefficient) and the clock is applied once. The clock causes the PEs to store the data available at the vertical input port into the static coefficient latch.

Method 2 (Random Loading) -

In this method, some extra hardware is added in the PE and an additional address bus is provided which carries the address of the PE, to which the static coefficient available on the input port (I_j^V) belongs (see Figure 3.5).

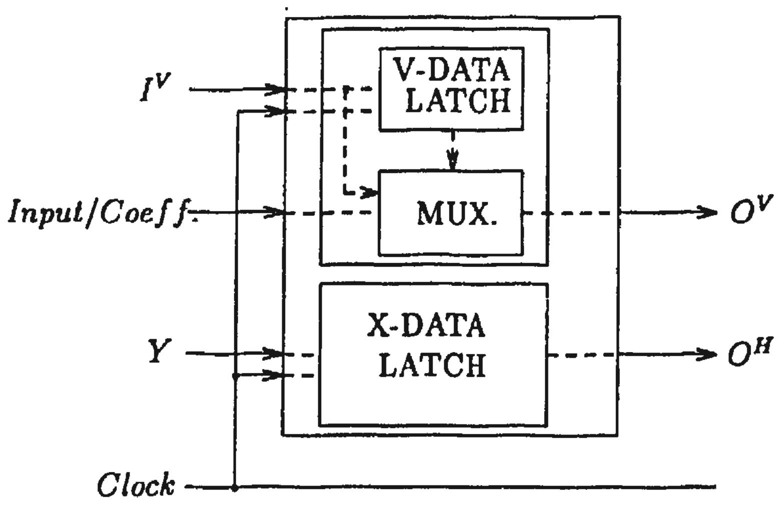


Figure 3.6: Output Latch Block for Random Coefficient Loading

A multiplexer is used in the output latch block to bypass the output latch (see Figure 3.6), when coefficients are being loaded. In this case, each $PE_{i,j}$ $(0 \leq i \leq m; 0 \leq j < n)$ gets the same data which is available on input port I_j^V. Firstly, weight $w_{i,j}$ is put on port I_j^V and then the address of $PE_{i,j}$ is put on the address bus and clock is applied to store $w_{i,j}$ in $PE_{i,j}$. This scheme requires extra hardware and random loading is not essential in most cases, so it is rarely used.

When the array is operational, it is not possible to load the static coefficients without losing some information available in the PEs, because the $PE's$ output ports carry the partial results. So when a shift is performed in the case of $PE_{i,j}$ failure, it is not possible to load the new weight $w_{x-1,j}$ in $PE_{x,j}$ $(i < x \leq m)$ without losing some of the partial results. To overcome this problem, one more static coefficient latch is added in the PEs and the latches are called *static coefficient latch '0'* and *static coefficient latch '1'*. Initially the $PE_{i,j}$ uses the static coefficient latch '0' (carrying $w_{i,j}$) and in the case of a $PE_{i,j}$ failure, the $PE_{x,j}$ $(i < x \leq m)$ start using the static coefficient latch '1' (carrying $w_{x-1,j}$). An additional line *select 0/1* is used to help the proper storing of static coefficients. This avoids the need

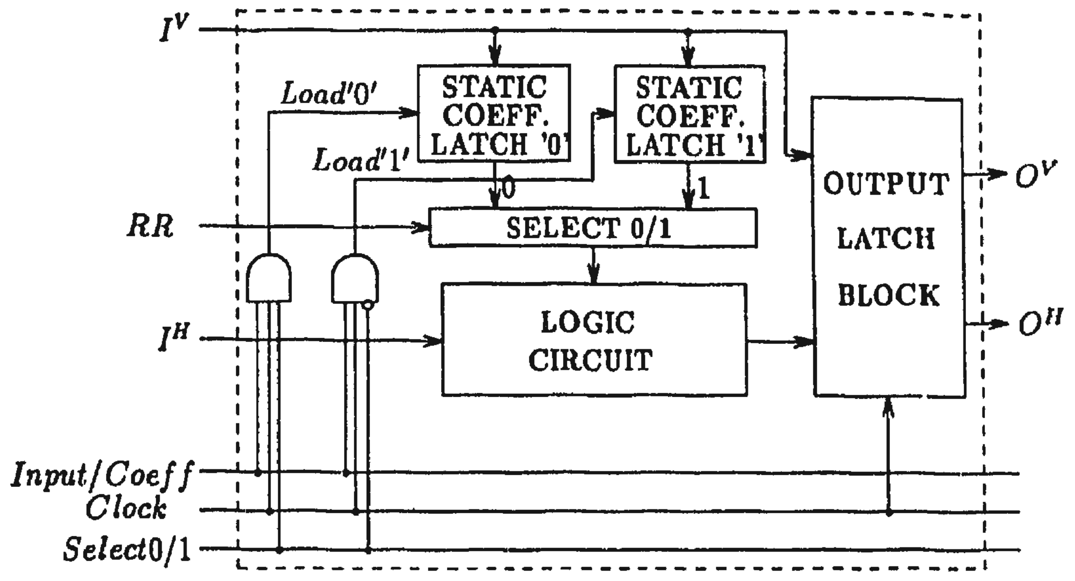


Figure 3.7: Block Diagram of PE With Two Static Coefficient Latches

to reload at the time of reconfiguration. The block diagram of PEs is shown in Figure 3.7. RR (Reconfiguration Request) is a signal, which comes to $PE_{x,j}$ $(i < x \leq n)$ in the case of $PE_{i,j}$ failure (it is explained in next subsection).

In this case the coefficients are loaded initially using method 1 (explained earlier). Initially *select 0/1* line is made valid for latch 0, so at clock $m-2$ (because there are m active rows in the array, namely row 0 through row $(m-1)$ and clock pulses are counted from pulse 0), $w_{i,j}$ is loaded in $PE_{i,j}$ $(0 \leq i < m)$ and $w_{i,j}$ $(0 \leq i < m)$ appears at the input of $PE_{i+1,j}$. At this point, the line *select 0/1* is made valid for latch 1 and the next clock pulse, $m-1$, loads $w_{i-1,j}$ in $PE_{i,j}$ $(0 < i \leq m)$.

During reconfiguration, rerouting of data is done, so a switching network is added to facilitate the rerouting. For an active array of size $m \times n$, a physical array of size $(m+1) \times n$ ($PE_{0,0}$ through $PE_{m,n-1}$) is required and to support the routing, a switch array of size $(m+2) \times (n+1)$ ($S_{0,0}$ through $S_{m+1,n}$) is required. The complete array is shown in Figure 3.8. I_i^H and I_j^V represent the horizontal input of row i and vertical input of column j from the central processor respectively. Similarly, O_i^H and O_j^V represent the horizontal output of row i and vertical output of column j from the array respectively.

Each switch module shown in Figure 3.8 is a pair of switches (one is used for vertical routing and the other for horizontal routing). For the sake of clarity the vertical and horizontal paths are shown separately in Figure 3.9.

In the next subsection a scheme is proposed for proper handling of partial results in the case of PE failure.

3.2.2 Handling of Partial Results

Consider the array shown in Figure 3.8. When a $PE_{i,j}$ fails, it invokes a downward shift (if the bottom row spare is available) and the logical index of $PE_{x,j}$ $(i < x \leq m)$ changes from (x, j) to $(x-1, j)$. For the sake of clarity, vertical and horizontal partial result handling are explained separately.

Handling of Vertical Partial Result

At any time t_1 ($t < t_1 < t+1$; shown in Figure 3.10), the PEs are processing the data which are available at their input ports at time t_1 because the data were latched by the output latches of the previous cells at time t and they remain there till the next clock edge, $t+1$ comes. $I_{i,j}^{H,t}$ and $I_{i,j}^{V,t}$ denote the horizontal and vertical inputs of $PE_{i,j}$ at time t respectively and $O_{i,j}^{H,t}$ and $O_{i,j}^{V,t}$ represent the horizontal and vertical outputs of $PE_{i,j}$ at time t respectively. Similarly, $I_{i,j,L}^{H,t}, I_{i,j,L}^{V,t}, O_{i,j,L}^{H,t}$ and $O_{i,j,L}^{V,t}$ denote the horizontal input, vertical input, horizontal output and vertical output of $PE_{i,j}^L$ (PE with logical index (i,j)) at time t respectively. When $PE_{i,j}$ fails at time t_1, it immediately generates a *Reconfiguration Request (RR)* and passes it to the central processor, which delays the next clock edge, $t+1$ for a pre-specified duration (which depends on the time taken for the switch settings and the processing time

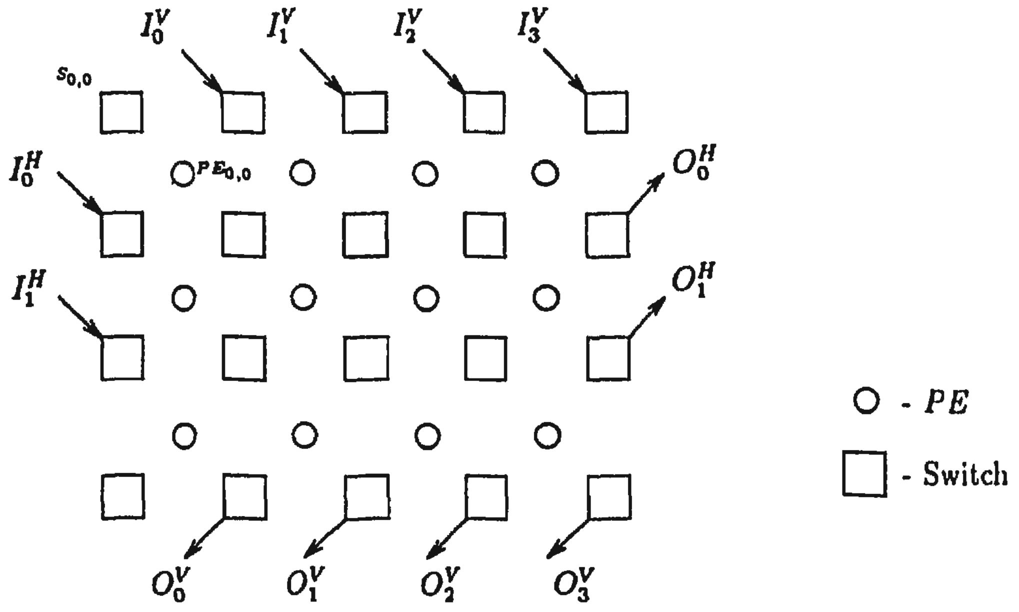


Figure 3.8: Basic Array with Switch Modules

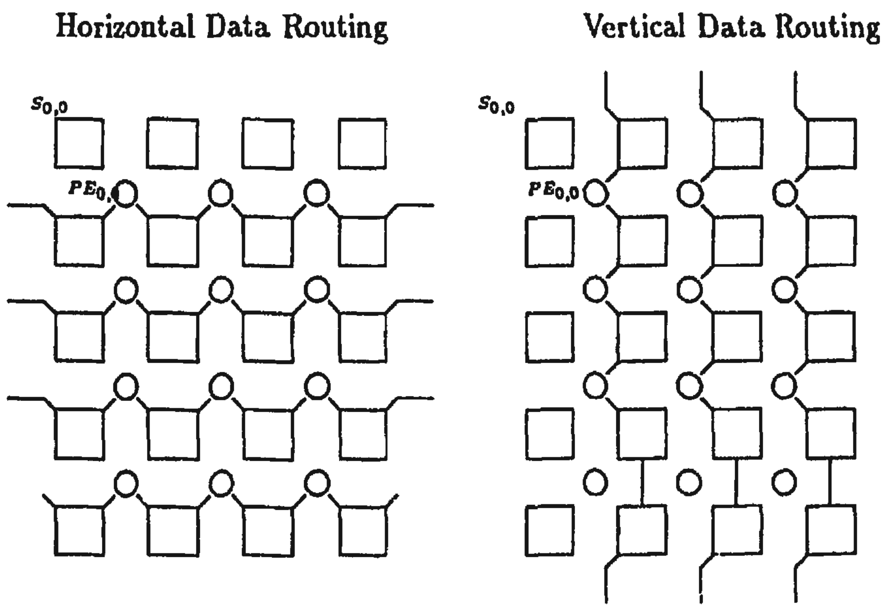


Figure 3.9: Vertical and Horizontal Data Paths

of each PE). RRs are written as RR_X^Y, which means that the RR is generated by X and it is fed to Y (for example, $RR_{PE_{i,j}}^{S_{i,j+1}}$ denotes the reconfiguration request generated by $PE_{i,j}$ and it goes to switch $S_{i,j+1}$). Since the logical index of $PE_{x,j}$ $(i < x \leq m)$ has changed from (x, j) to $(x-1, j)$ at t_1, the $PE_{x,j}$ $(i < x \leq m)$ has to process the same data, which $PE_{x-1,j}$ was processing at time t_1; for instance, after t_1, $PE_{i+1,j}$ should get $O_{i-1,j}^{V,t}$, $PE_{i+2,j}$ should get $O_{i,j}^{V,t}$ and so on, meaning that $I_{i+1,j}^{V,t} = O_{i-1,j}^{V,t}$, $I_{i+2,j}^{V,t} = O_{i,j}^{V,t}$ and so on. To accomplish this, an intermediate state of the vertical path is provided (shown in Figure 3.10), which is called the first or intermediate stage of rerouting. At t_1, the switches $S_{x,j+1}$ $(i \leq x \leq m+1)$ are set to provide this routing and the next clock is applied at $t+1$, which causes the intermediate results to appear on the output ports of the PEs. At this time the switches $S_{x,j+1}$ $(i < x \leq m+1)$ are set again to get the final reconfigured vertical path (shown in Figure 3.10). After final routing $I_{i,j,L}^{V,t} = O_{i-1,j,L}^{V,t}$ $(I_{i+1,j}^{V,t} = O_{i-1,j}^{V,t})$, and so on.

The horizontal partial result handling is explained in the next subsection.

Handling of Horizontal Partial Result

After the $PE_{i,j}$ failure at t_1, each $PE_{x,j}$ $(i < x \leq m)$ has to work as $PE_{x-1,j}^L$ and each $PE_{x,j}$ $(i < x \leq m)$ has to get the same horizontal input as $PE_{x-1,j}$ was getting at time t_1, i.e., $I_{i+1,j}^{H,t_1} = I_{i+1,j}^{H,t} = O_{i,j}^{H,t}, I_{i+2,j}^{H,t_1} = I_{i+2,j}^{H,t} = O_{i+1,j}^{H,t}$ and so on. To accomplish this, an intermediate horizontal routing is done at t_1 (as shown in Figure 3.11) and at $t+1$ the final routing is done to get the final reconfiguration, so that each $PE_{x,j+1}$ $(i \leq x < m)$ gets horizontal partial result from $PE_{x+1,j}$.

Lemma 3.1 - Reconfiguration in the case of a PE failure requires a maximum of two stages of rerouting.

Proof - There are only two combinations of PE failure: either a spare PE fails or an active PE fails.

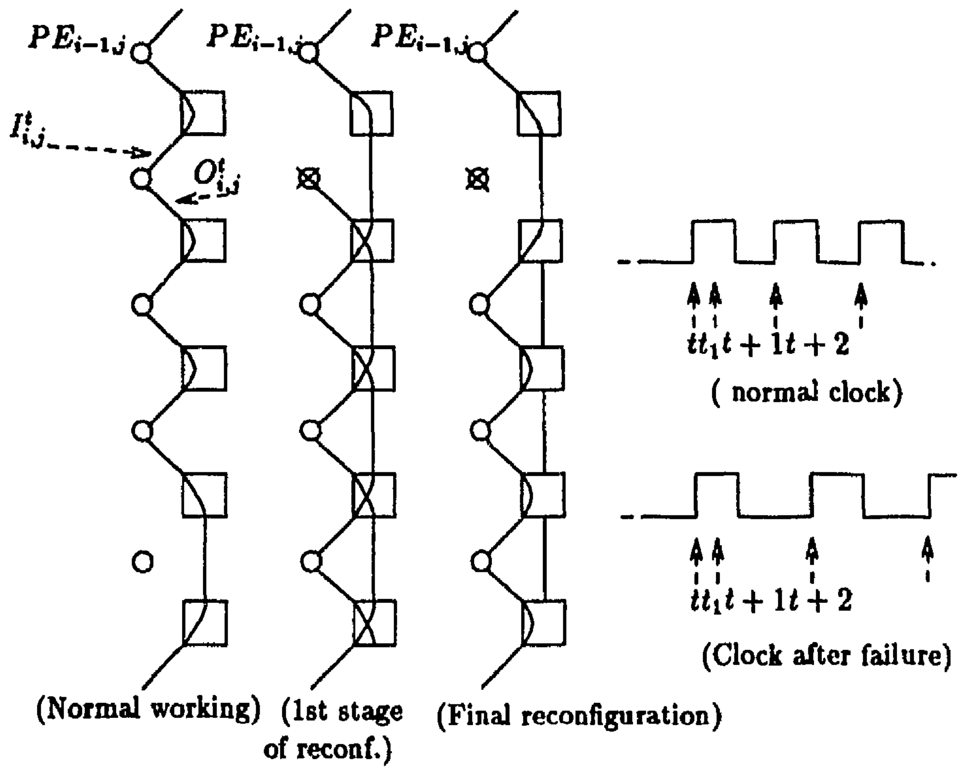


Figure 3.10: Vertical Partial Result Handling

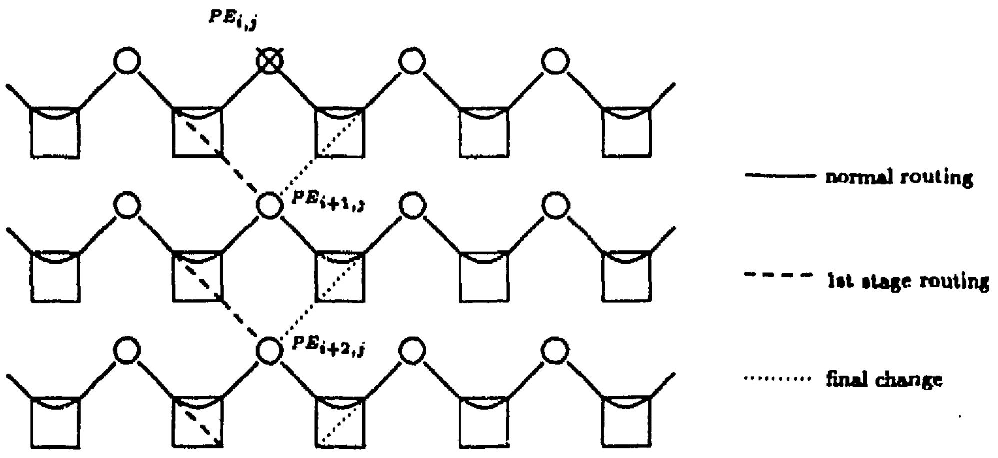


Figure 3.11: Horizontal Partial Result Handling

1. When a spare PE fails, it does not invoke any reconfiguration and

2. when an active PE fails, it invokes the reconfiguration and as explained earlier (in vertical and horizontal partial result handling subsections), any such failure requires two stages of rerouting (intermediate stage and final stage).

□

In the next subsection switch modules are discussed.

3.2.3 Switch Module

As explained earlier, each switch module consists of two switches. One of them is used exclusively for horizontal data routing and the other is used for vertical data routing. Both of them are discussed separately in the following subsections.

Vertical Data Routing Switch

Consider the array shown in Figure 3.12 (only vertical data paths are shown). Here, I_0^V, I_1^V, I_2^V ... are the input data from the central processor to the array and O_0^V, O_1^V, O_2^V ... form the final output from the array.

At time t_1, $PE_{i+1,j-2}$ has already failed (and has been reconfigured) and $PE_{i,j}$ fails at time t_1 causing the first stage of rerouting to be done. So in this figure, column $(j-1)$ of the switches shows the vertical data path, which is fully reconfigured and column $(j+1)$ of the switches shows the vertical data path in the intermediate stage. To support the reconfiguration, the network shown in Figure 3.13 is provided.

It is clear from the network that the switch modules $S_{x,0}$ $(0 \leq x \leq m+1)$ need not have the vertical data routing switch. Each switch is a 2×2 switch, the inputs are denoted as I_{S0}^V, I_{S1}^V and the outputs are written as O_{S0}^V and O_{S1}^V. The vertical input of the array, I_x^V is given to the I_{S0}^V input of $S_{0,x+1}^V$ $(0 \leq x < n)$ and the I_{S1}^V of these switches is not used. Final vertical output, O_x^V is taken from the array by

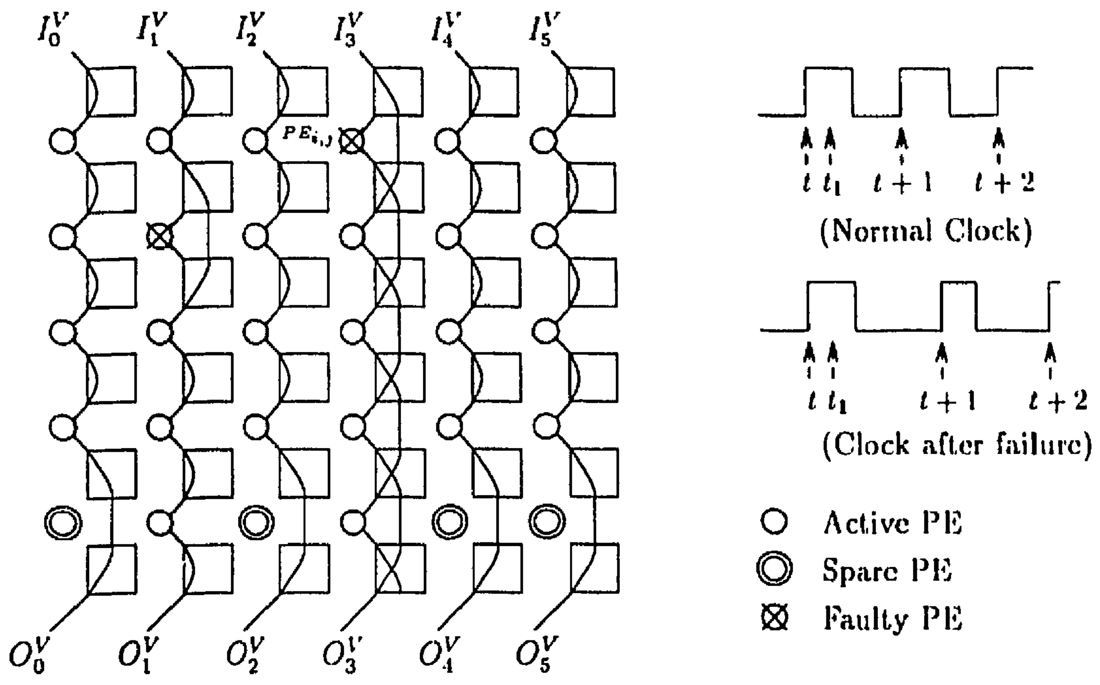


Figure 3.12: Vertical Data Routing Path

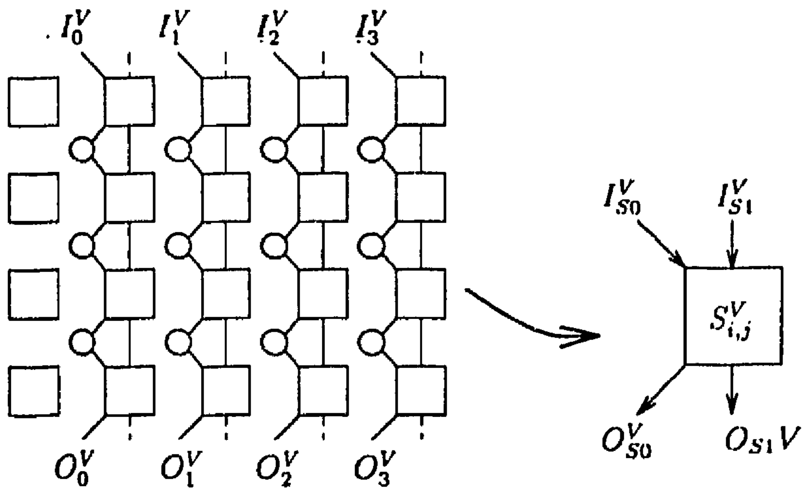


Figure 3.13: Network for Vertical Data Handling during Reconfiguration

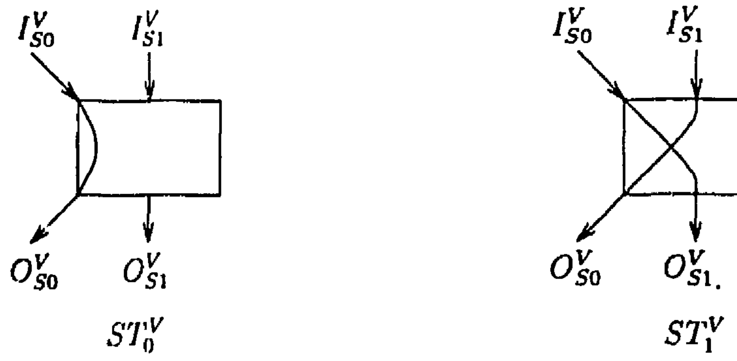


Figure 3.14: States of Vertical Switches (For PE failure algorithm)

O_{S0}^V of $S_{m+1,x+1}^V$. In order to get all the required connections, the vertical switches have two states (shown in Figure 3.14).

Initially, all the switches $S_{i,j}^V$ $(0 \leq i \leq m-1$ *and* $0 \leq j \leq n)$ are in state ST_0^V, the switches $S_{i,j}^V$ $(m \leq i \leq m+1$ *and* $0 \leq j \leq n)$ are in state ST_1^V and when a $PE_{i,j}$ fails at t_1, it changes the states of switches $S_{x,j+1}^V$ $(i \leq x \leq m-1)$ from ST_0 to ST_1. At $t+1$, switches $S_{x,j+1}^V$ $(i+2 \leq x \leq m+1)$ are brought back to state ST_0^V.

Lemma 3.2 - The two proposed states (ST_0^V and ST_1^V) of vertical switches are sufficient to support the algorithm.

Proof - As shown in Lemma 3.1, a PE failure requires two stages of rerouting so a vertical data path can be in either of the following three states:

1. the particular data path doesn't have any faulty PE;
2. the particular data path has a faulty PE and the reconfiguration is in the intermediate stage or
3. the particular data path has a reconfigured faulty PE.

The data paths required by these states are shown in Figure 3.12. Since $PE_{i+1,j-2}$ has been reconfigured completely, column $(j-1)$ of the switches shows the data paths required by final stage. $PE_{i,j}$ failure has gone through the first stage of rerouting only, so column $(j+1)$ of switches shows the data paths required by the intermediate stage of rerouting. Other columns of switches show the normal data routing. It is obvious from Figure 3.12, Figure 3.13 and Figure 3.14 that the proposed two states of the switches provide all the required data paths. For a column x of PEs, if no $PE_{i1,x}$ $(0 \leq i1 \leq m-1)$ is faulty, the spare PE, $PE_{m,x}$ is bypassed by bringing switches $S^V_{m,x+1}$ and $S^V_{m+1,x+1}$ to ST^V_1. Other switches of column $(x+1)$ would be in ST^V_0. $PE_{i+1,j-2}$ failure is reconfigured completely and the data paths, required for this are provided by bringing $S^V_{i+1,j-1}$ and $S^V_{i+2,j-1}$ to ST^V_1. Other switches of column $(j-2)$ stay in ST^V_0. $PE_{i,j}$ failure is in intermediate stage and data paths are provided by bringing switches $S^V_{i1,j+1}$ $(i \leq i1 \leq m+1)$ to ST^V_1. Other switches of column $(j+1)$ stay in ST^V_0. □

Horizontal Data Routing Switch

The horizontal routing is shown in Figure 3.15. At t_1, $PE_{i,j-2}$ has already failed and has been reconfigured completely and $PE_{i,j}$ fails at this point, causing the first stage of reconfiguration. The network, illustrated in Figure 3.16 is provided to support the algorithm. It is clear that the switches $S_{0,j}$ $(0 \leq j < n)$ need not have the horizontal switch. The horizontal input to the array, I^H_i comes to the I^H_{S0} port of switch $S^H_{i+1,0}$ for all $(0 \leq i < m)$ and the output O^H_i is taken from the O^H_{S0} port of switch $S^H_{i,n}$. The various switch states for a switch $S^H_{i,j}$ are shown in Figure 3.17.

When a $PE_{i,j}$ fails at t_1, it changes the states of $S^H_{x,j}$ $(i < x \leq m+1)$ from ST^H_0 to ST^H_1 and at time $t+1$ next clock edge is given which changes the switches $S^H_{x,j+1}$ $(i < x \leq m+1)$ from ST^H_0 to ST^H_1. At the time of switch settings, the PEs are informed to use the proper input port, on which the correct data is available.

The above scheme is valid when for a $PE_{i,j}$ failure, there is no faulty $PE_{i1,j-1}$

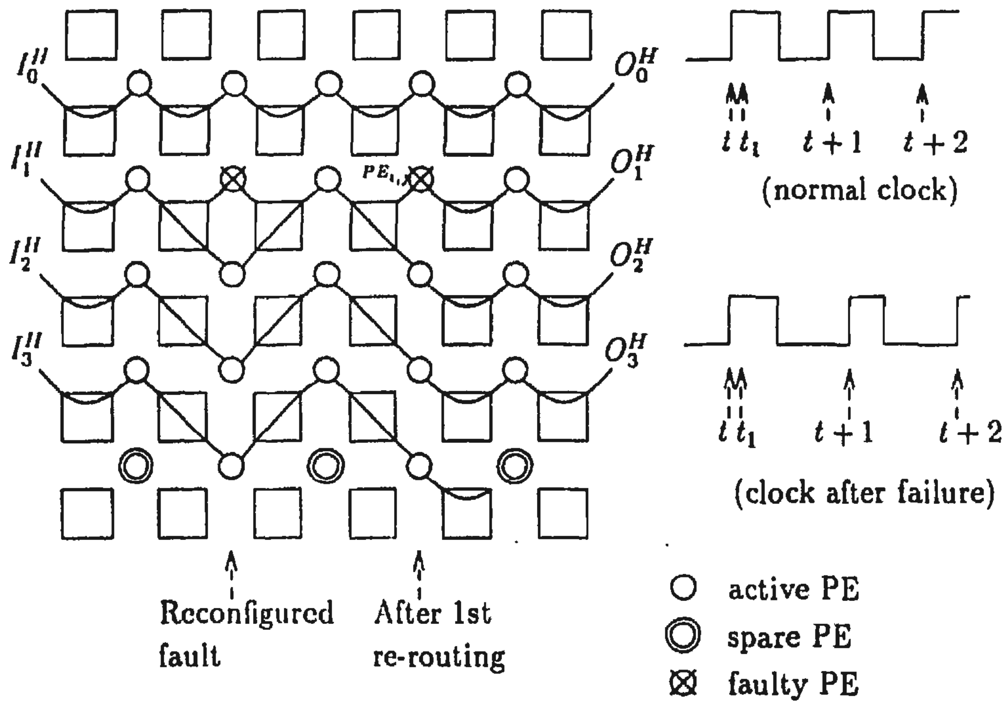


Figure 3.15: Horizontal Data Routing Path

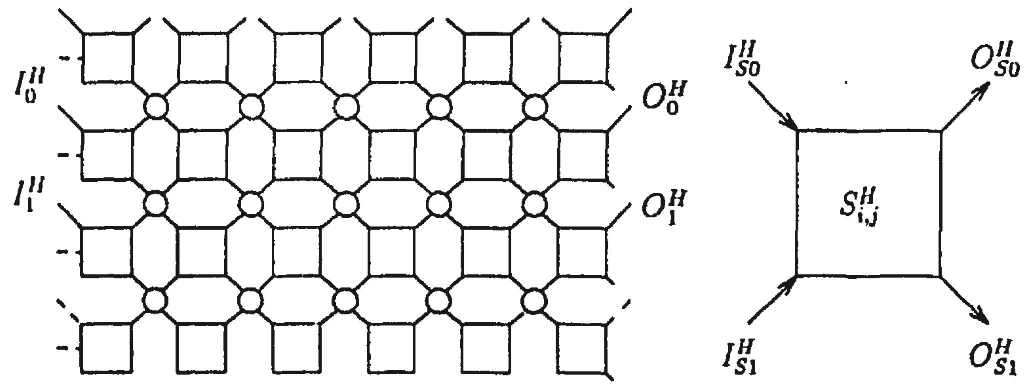


Figure 3.16: Network for Horizontal Data Handling during Reconfiguration

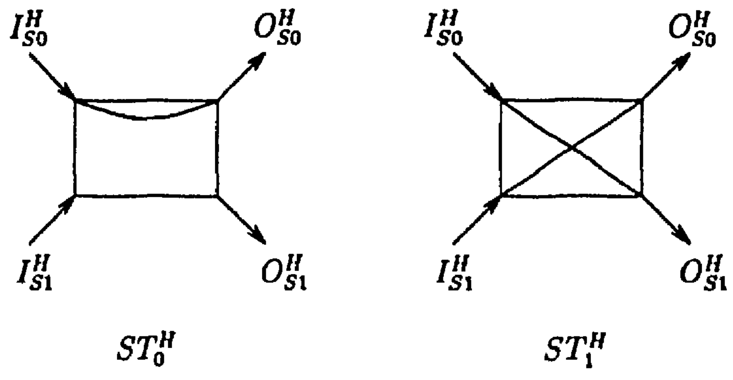


Figure 3.17: States of Horizontal Switches (For PE failure algorithm)

or $PE_{i1,j+1}$ present. In presence of any such faulty PE, the algorithm is changed. Both of these cases are explained below:

a. $PE_{i1,j-1}$ **Faulty**: the array is shown in Figure 3.18. After the intermediate rerouting at t_1, $PE^L_{x,j}$ ($i \leq x \leq m$; which is $PE_{x+1,j}$) should get data from $PE^L_{x,j-1}$ (which is $PE_{x+1,j-1}$), so the switches $S^H_{x,j-1}$ ($i < x \leq m+1$) change state either from ST^H_1 (caused by previous reconfiguration due to $PE_{i1,j-1}$ failure) to ST^H_0 or from ST^H_0 to ST^H_1. The final reconfiguration at $t+1$ changes the states of the switches $S^H_{x,j+1}$ ($i < x \leq m+1$), from ST^H_0 to ST^H_1.

b. $PE_{i1,j+1}$ **Faulty** - the array is shown in Figure 3.19. Here, at t_1 all the switches $S^H_{x,j-1}$ ($i < x \leq m+1$) change state from ST^H_0 to ST^H_1. After the rerouting at $t+1$, $PE^L_{x,j+1}$ ($i \leq x \leq m$) should get data from $PE^L_{x,j}$. To achieve this, at $t+1$ the switches $S^H_{x,j+1}$ ($i < x \leq m+1$) change state either from ST^H_0 to ST^H_1 or from ST^H_1 to ST^H_0.

Lemma 3.3 - The two proposed states (ST^H_0 and ST^H_1) of the horizontal switches are sufficient to support the algorithm.

Proof - Horizontal data routing, in the case of a $PE_{i,j}$ failure depends on earlier failures. There are only four combinations of this occurrence, which are listed below:

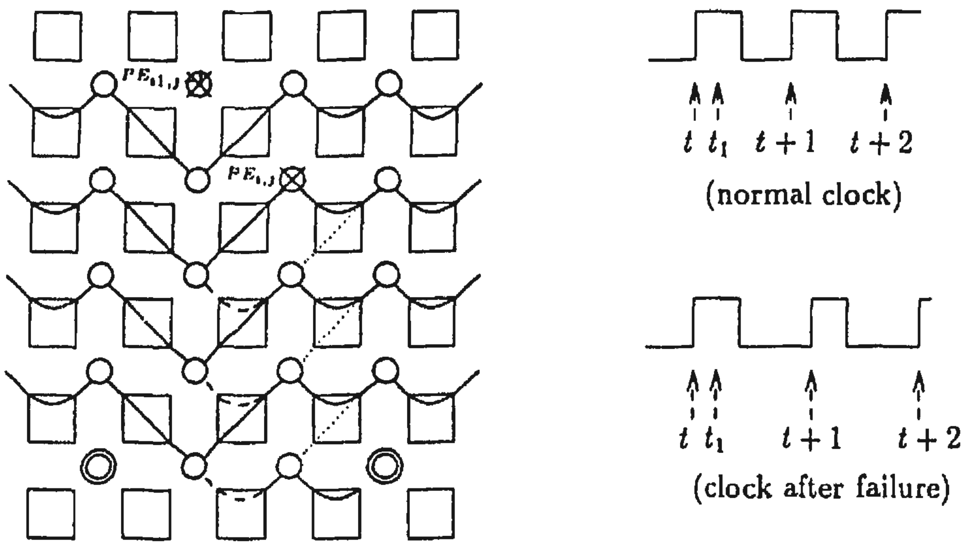


Figure 3.18: Horizontal reconfiguration for $PE_{i,j}$ in presence of faulty $PE_{i1,j-1}$

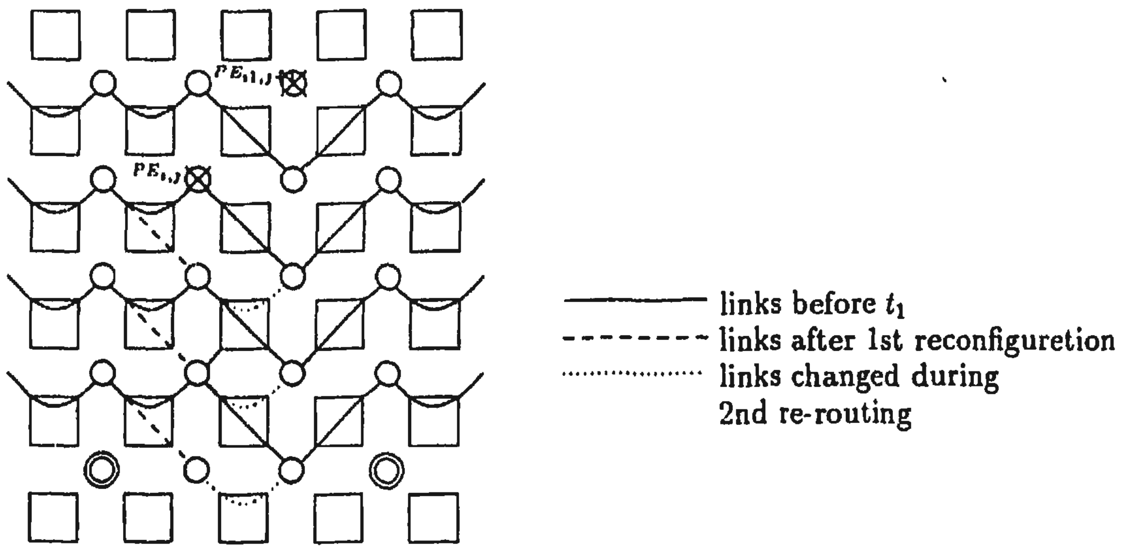

$PE_{i1,j}$
$PE_{i,j}$
links before t_1
links after 1st reconfiguretion
links changed during 2nd re-routing

Figure 3.19: Horizontal reconfiguration for $PE_{i,j}$ in presence of faulty $PE_{i1,j+1}$

1. no PE in columns $(j-1)$ and $(j+1)$ is faulty;

2. column $(j-1)$ has a faulty PE $(PE_{i1,j-1})$ and it has been reconfigured (it is assumed that faults occur one at a time);

3. column $(j+1)$ has a faulty PE $(PE_{i1,j+1})$ and it has been reconfigured and

4. both columns $(j-1)$ and $(j+1)$ have faulty cells ($PE_{i1,j-1}$ and $PE_{i2,j+1}$ respectively).

As shown in Lemma 3.1, only two stages of rerouting are required in the case of a PE failure reconfiguration. For horizontal data rerouting, in the case of $PE_{i,i}$ failure, the intermediate stage requires modification of data links between PEs of column $(j-1)$ and PEs of column j and the final stage of rerouting requires modifications of data links between PEs of column j and PEs of column $(j+1)$. Cases 1, 2 and 3 are shown in Figures 3.15, 3.18 and 3.19 respectively and it is clear that the proposed two states of horizontal switches are capable of providing all required data links. Case 4 is the combination of Case 2 and Case 3 and since for horizontal data rerouting, intermediate and final stages of rerouting are mutually exclusive (the intermediate stage requires state changes of switches in column j and the final stage requires state changes of switches in column $(j+1)$), the intermediate rerouting in this case would be similar to that of Case 2 and final rerouting would be similar to that of Case 3. So the proposed two states (ST_0^H and ST_1^H) would provide all horizontal data paths required by the algorithm. □

So, in the case of a $PE_{i,j}$ failure at t_1, the switches $S^H_{x,j-1}$ $(i < x \leq m+1)$ change state either from ST_0^H to ST_1^H or from ST_1^H to ST_0^H at t_1 and at $t+1$ switches $S^H_{x,j+1}$ perform the same.

The switches are finite state blocks. In the next subsection various changes in the basic network, processing element and switch modules are explained.

3.2.4 Network

The network is modified to implement the algorithm and it is shown in Figure 3.20.

In this figure, global clock line (CLK_{PE}), *input/coefficient* line, select 0/1 line (used for loading the coefficients initially), reset line (used for resetting all the flip flops initially) and fatal failure line (explained later) are not shown. Various control lines for a PE and switch module are shown in Figure 3.21.

$RR_{PE_{i,j}}^{PE_{i+1,j}}$ is the reconfiguration request from $PE_{i,j}$ to $PE_{i+1,j}$ and $RR_{PE_{i,j}}^{S_{i1,j1}}$ is the reconfiguration request from $PE_{i,j}$ to switch $S_{i1,j1}$. FF is connected to the fatal failure line, which indicates the occurrence of fatal failure. Once a PE fails, the reconfiguration starts and it is done based on the information available on these lines. When a faulty $PE_{i,j}$ receives an RR from $PE_{i-1,j}$, it generates FF (fatal failure signal) and puts it on the FF line, which carries it to the central processor.

3.2.5 Processing Element

The block diagram of the processing element is shown in Figure 3.22. Each $PE_{i,j}$ has two static coefficient latches and two horizontal inputs (I_{PE0}^{H} and I_{PE1}^{H}) and the selection is done by using the signal *select*$_{0/1}$ line, which becomes high when $PE_{i,j}$ receives $RR_{PE_{i-1,j}}^{PE_{i,j}}$.

The SPE (spare PE) signal is applied to the spare cells initially and it is latched to derive SPE_L, which is used to ensure that the spare cells do not invoke reconfiguration. The *PE test circuit* checks the state of the PE and when it detects a fault in the logic circuit, it generates E_{LOGIC}, which remains high for the full duration of the array operation. The block diagram of the control circuit is given in Figure 3.23 and timing diagram of various signals is shown in Figure 3.24. Once a fault is detected by the self-test circuit of $PE_{i,j}$, it passes this information on to the control block of the $PE_{i,j}$ using the line E_{LOGIC}. When the control circuit of $PE_{i,j}$ receives this E_{LOGIC} (see Figure 3.24.a), it generates $RR_{PE_{i,j}}^{PE_{i+1,j}}$, $RR_{PE_{i,j}}^{S_{i,j+1}}$

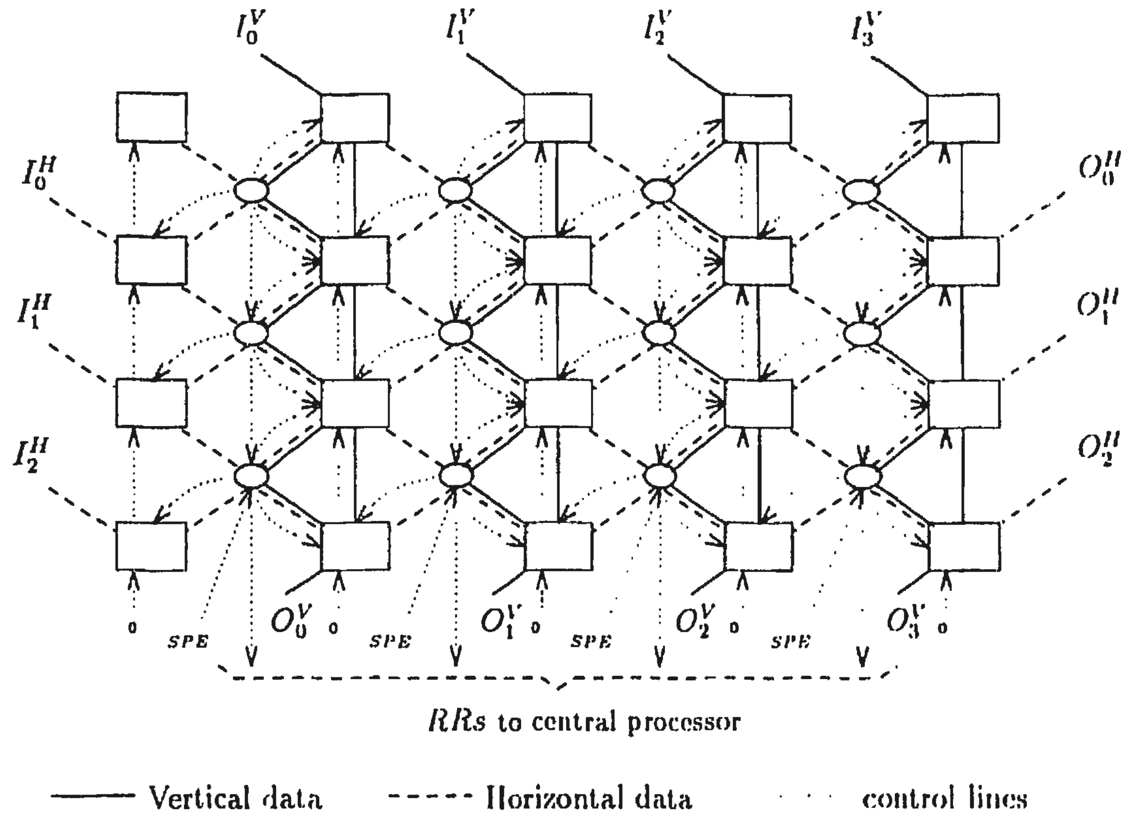
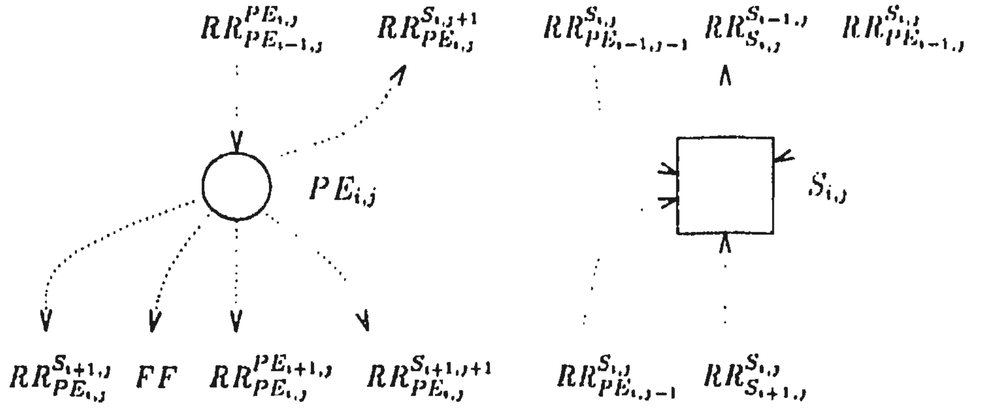


Figure 3.20: Modified Network (for supporting PE failure algorithm)



$RR_{PE_{i-1,j}}^{PE_{i,j}}$
$RR_{PE_{i,j}}^{S_{i,j+1}}$
$RR_{PE_{i-1,j-1}}^{S_{i,j}}$
$RR_{S_{i,j}}^{S_{i-1,j}}$
$RR_{PE_{i-1,j}}^{S_{i,j}}$
$PE_{i,j}$
$S_{i,j}$
$RR_{PE_{i,j}}^{S_{i+1,j}}$
FF
$RR_{PE_{i,j}}^{PE_{i+1,j}}$
$RR_{PE_{i,j}}^{S_{i+1,j+1}}$
$RR_{PE_{i,j-1}}^{S_{i,j}}$
$RR_{S_{i+1,j}}^{S_{i,j}}$

Figure 3.21: Control Lines for PE and Switches (PE failure algorithm)

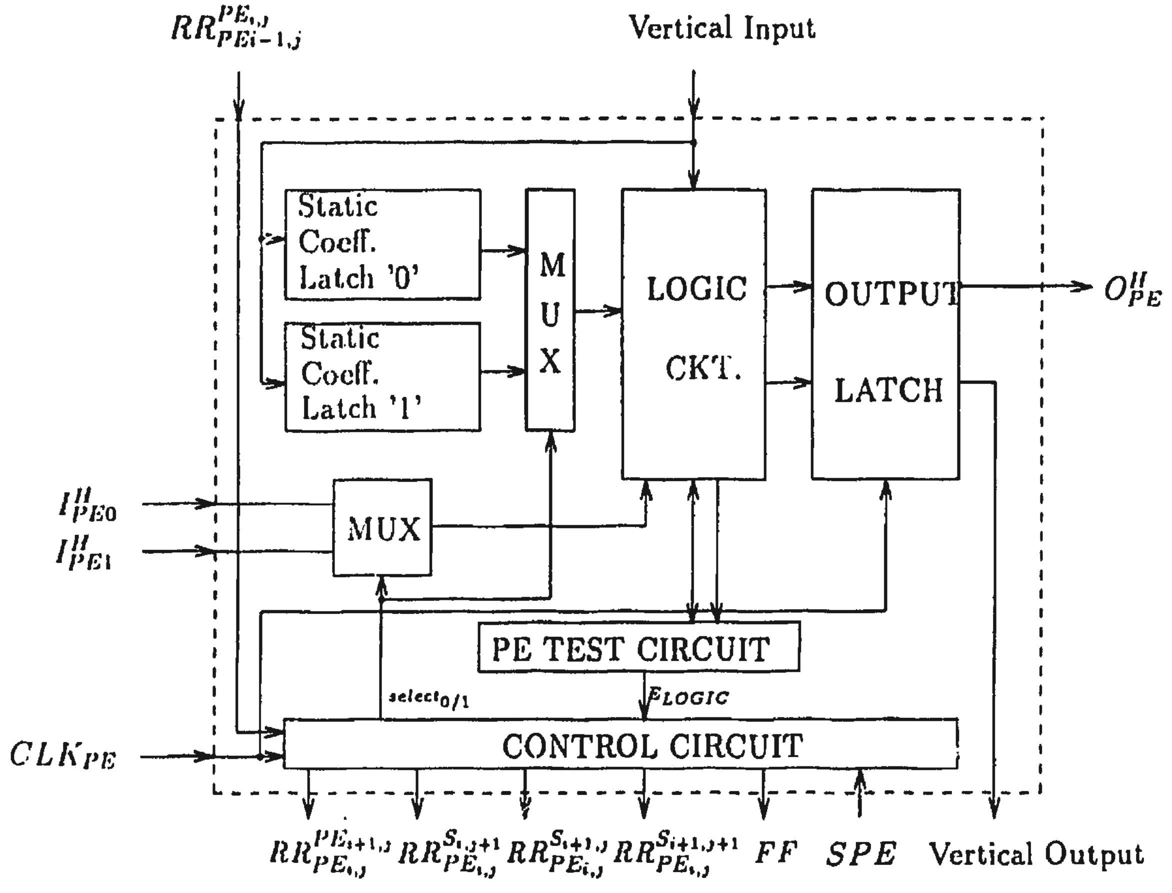


Figure 3.22: Complete Block Diagram of Modified PE (PE failure algorithm)

and $RR^{S_{i+1,j}}_{PE_{i,j}}$. These RRs are reset at $t+1$, but E_{LOGIC} stays high. Now, if $PE_{i,j}$ receives $RR^{PE_{i,j}}_{PE_{i-1,j}}$, it would generate the FF signal (because this indicates two faulty PEs in the same column).

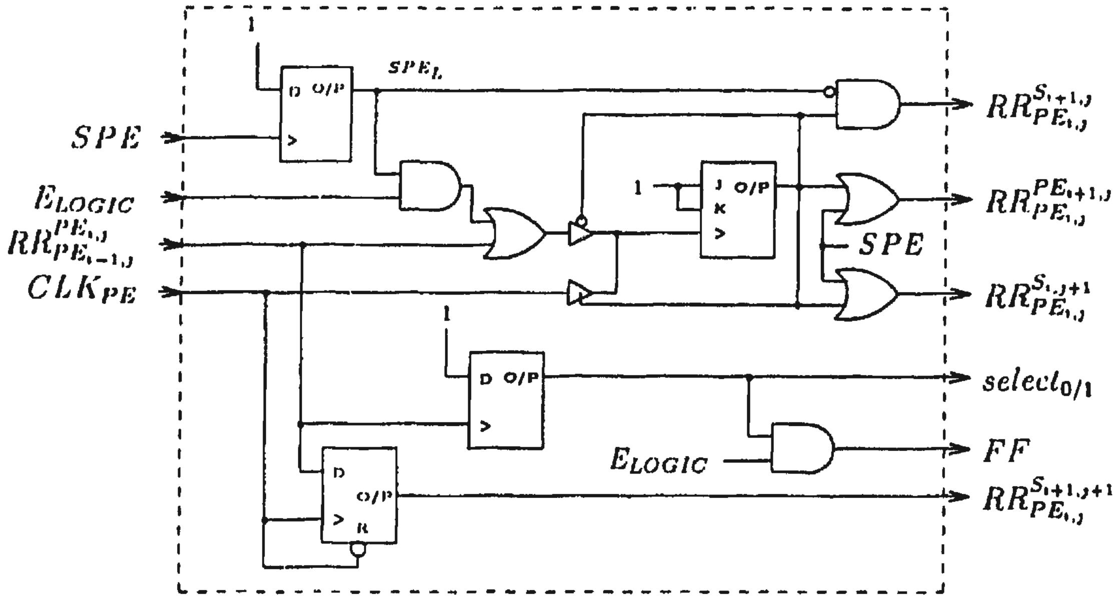
When $PE_{i,j}$ receives $RR^{PE_{i,j}}_{PE_{i-1,j}}$ at t_1 (see Figure 3.24.b), it generates $RR^{PE_{i+1,j}}_{PE_{i,j}}$, $RR^{S_{i,j+1}}_{PE_{i,j}}$ and $RR^{S_{i+1,j}}_{PE_{i,j}}$ at t_1 and at the next clock edge, $t+1$ it resets these RRs and generates $RR^{S_{i+1,j+1}}_{PE_{i,j}}$, which is reset at the next falling edge of the clock, at t_2. In this case, if $PE_{i,j}$ fails at t_3, it generates FF.

3.2.6 Switch

The block diagram of the switch module is given in Figure 3.26. Each switch consists of three basic circuits: one control circuit, one horizontal switch (used for horizontal data routing) and one vertical switch (used for vertical data routing). The control circuit is very simple in this case and the horizontal switch toggles from one state to the other, when either $RR^{S_{i,j}}_{PE_{i-1,j}}$ or $RR^{S_{i,j}}_{S_{i+1,j}}$ comes. The vertical switch goes to state ST^V_1 when $RR^{S_{i,j}}_{PE_{i,j-1}}$ comes and it goes back to ST^V_0 at the arrival of $RR^{S_{i,j}}_{PE_{i-1,j-1}}$.

Consider a portion of the array as shown in Figure 3.25. When $PE_{i,j}$ fails at t_1, various RRs are generated. The control circuit (shown in Figure 3.23) is used to generate these signals. The $RR^{PE_{m+1,j}}_{PE_{m,j}}$ of the bottom row of cells is connected to the central processor, which delays the next rising edge $(t+1)$ of the clock. This delay is the sum of the switch settling time and the processing time of a PE. The arrival of pulse $t+2$ is also delayed by the same amount of time and after that the clock resumes its normal speed.

The central processor gives a signal called SPE (spare PE) to the spare PEs and it is used to bring $S^V_{m,j}$ and $S^V_{m+1,j}$ to ST^V_1 initially. SPE is latched as SPE_L, which is used to ensure that no RR is generated, when a spare cell detects a self-fault. The $RR^{S_{i,j}}_{S_{i+1,j}}$ input of switches, $S_{m+1,j}$ $(0 \leq j < n)$ is pulled low and it is



1
SPE_L
SPE
E_{LOGIC}
$RR^{PE_{i,j}}_{PE_{i-1,j}}$
CLK_{PE}
D O/P
J O/P
K
1
SPE
D O/P
1
D O/P R
E_{LOGIC}
$RR^{S_{i+1,j}}_{PE_{i,j}}$
$RR^{PE_{i+1,j}}_{PE_{i,j}}$
$RR^{S_{i,j+1}}_{PE_{i,j}}$
$select_{0/1}$
FF
$RR^{S_{i+1,j+1}}_{PE_{i,j}}$

Figure 3.23: Control Circuit of PE (PE failure algorithm)

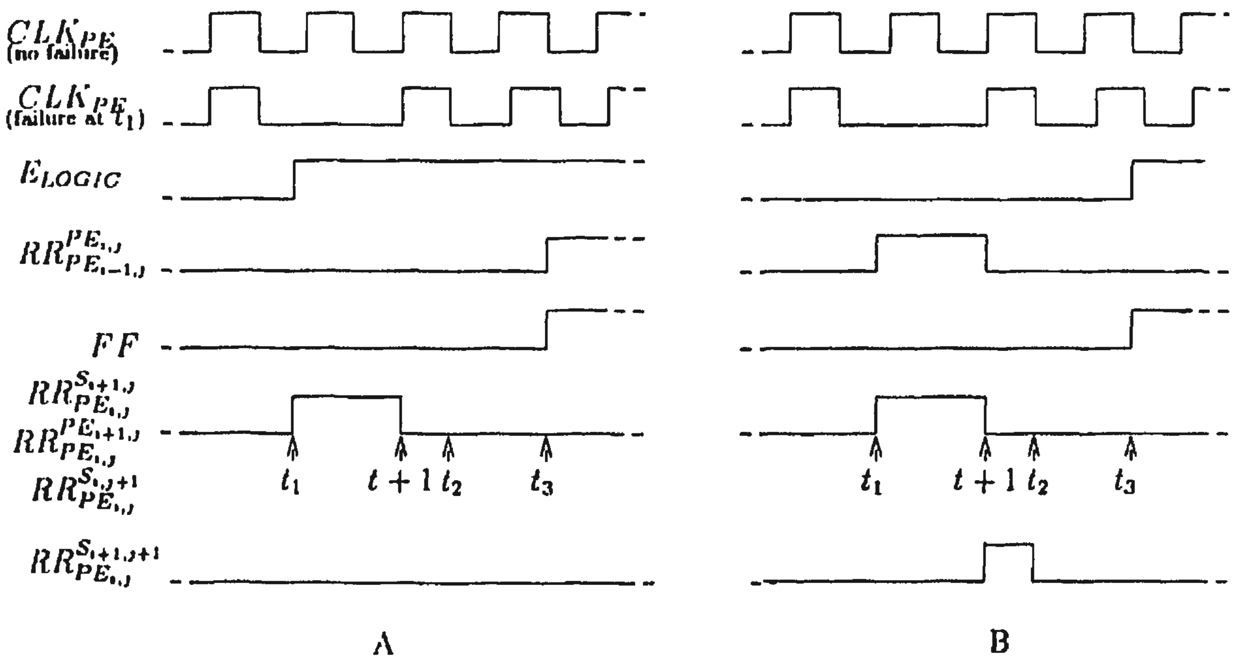
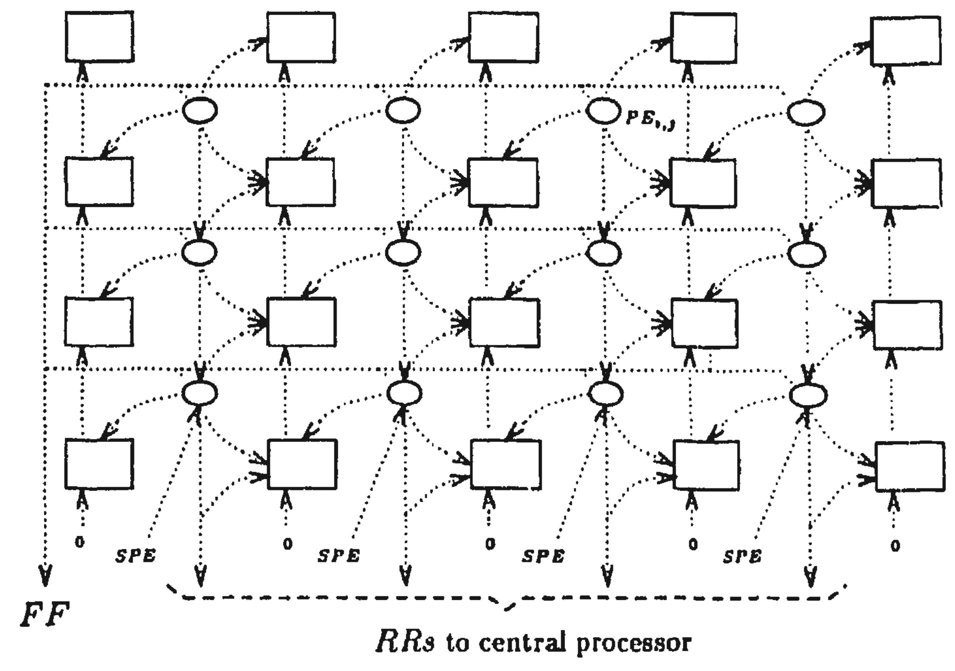


Figure 3.24: Signal Waveforms (Output of the $PE's$ control circuit)



——— Vertical data ----- Horizontal data ········· control lines

Figure 3.25: Reconfiguration Request Propagation (for PE failure algorithm)

called $RR_0^{S_{m+1,j}}$. SPE_L and $RR_0^{S_{m+1,j}}$ ensure that $S^H_{m+1,j}$ $(0 \leq j < n)$ do not change state (these switches always remain in ST^H_0).

The switches are finite state blocks as shown in Figure 3.26 and the states of the switches, depending on the RR lines, are shown in Figure 3.27. Case A shows the vertical switch state change for switches $S^V_{i,j+1}$ and $S^V_{i+1,j+1}$ in the case of $PE_{i,j}$ failure at t_1. At t_1, these switches go to ST^V_1 and stay there. Case B shows the vertical switch state changes for switches $S^V_{i_x,j+1}$ $(i + 1 < i_x < m)$, in the case of $PE_{i,j}$ failure at t_1. At t_1, these switches go to ST^V_1 and come back to ST^V_0 at $t + 1$. Case C shows the state transition of switches $S^H_{i_x,j}$ $(i < i_x \leq m)$. These switches toggle from one state to the other at t_1 and remain in this state. Case D shows the state transition of switches $S^H_{i_x,j+1}$ $(i < i_x \leq m)$. These switches toggle from one state to the other at $t + 1$ and remain in that state.

In the next section, the full algorithm is detailed.

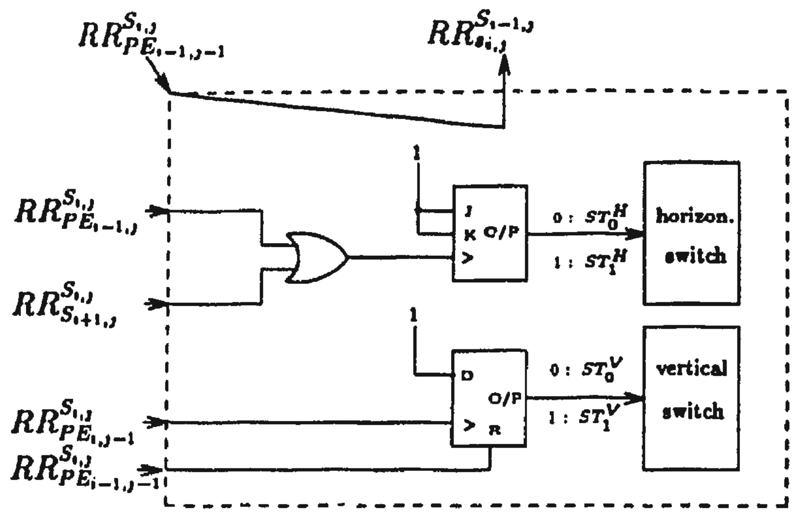

$RR^{S_{i,j}}_{PE_{i-1,j-1}}$
$RR^{S_{i-1,j}}_{S_{i,j}}$
$RR^{S_{i,j}}_{PE_{i-1,j}}$
$RR^{S_{i,j}}_{S_{i+1,j}}$
$RR^{S_{i,j}}_{PE_{i,j-1}}$
$RR^{S_{i,j}}_{PE_{i-1,j-1}}$
1
J
K O/P
0 : ST^H_0
1 : ST^H_1
horizon. switch
1
D
O/P
R
0 : ST^V_0
1 : ST^V_1
vertical switch

Figure 3.26: Block Diagram of the Switch module (for PE failure algorithm)

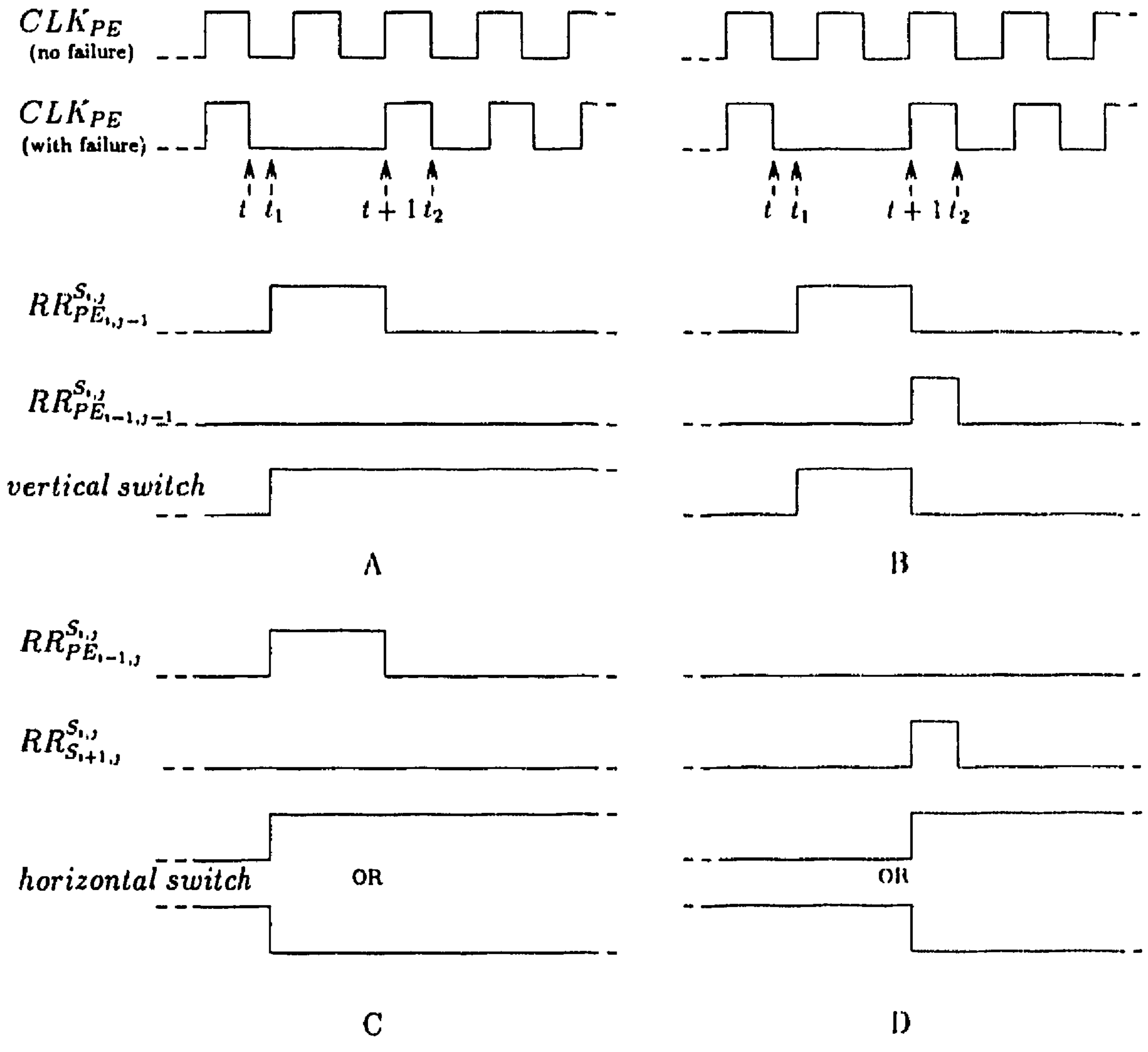
3.3 Operation of the Algorithm

Consider an $m \times n$ array (with m active rows of PEs, 0 through $m-1$, and n active columns of PEs, 0 through $n-1$).

Initially, all the horizontal switches $S^H_{i,j}$ and vertical switches $S^V_{i,j}$ $(0 \leq i \leq m+1\ ;\ 0 \leq j \leq n)$ are brought to state ST^H_0 by applying a pulse at the global reset line. Then the static coefficients are loaded in the array by using vertical input and *input/coefficient* lines as explained in the subsection 3.2.1 ($PE_{i,j}$ $(0 \leq i < m; 0 \leq j < n)$ contains the static coefficient of $PE^L_{i,j}$ in accumulator '0' and of $PE^L_{i-1,j}$ in accumulator '1'). Spare PEs, $PE_{m,j}$ $(0 \leq j < n)$ do not have any valid data in accumulator '0' and accumulator '1' contains the static coefficient of $PE^L_{m-1,j}$.

Next, the vertical switches, $S^V_{i,j}$ $(m \leq i \leq m+1\ ;\ 1 \leq j < n)$ are brought to state ST^V_1 by giving a pulse to the SPE inputs of the spare cells. SPE gets latched as SPE_L. This prepares the array for operation.

When a $PE_{i,j}$ $(i \neq m)$ fails at t_1, it issues RRs to $S_{i,j+1}, S_{i+1,j+1}, S_{i+1,j}$ and $PE_{i+1,j}$. After receiving this request $PE_{i+1,j}$ generates RRs to switches and to



CLK_{PE} (no failure)
CLK_{PE} (with failure)
t t_1 $t+1$ t_2
t t_1 $t+1$ t_2
$RR^{S_{i,j}}_{PE_{i,j-1}}$
$RR^{S_{i,j}}_{PE_{i-1,j-1}}$
vertical switch
A
B
$RR^{S_{i,j}}_{PE_{i-1,j}}$
$RR^{S_{i,j}}_{S_{i+1,j}}$
horizontal switch
OR
OR
C
D

Figure 3.27: State Transition of The Switches (for PE failure algorithm)

$PE_{i+2,j}$. In this way the reconfiguration request goes from $PE_{i,j}$ to the spare, $PE_{m,j}$. If on its way it encounters a faulty cell, a fatal failure occurs and the FF signal is given to the central processor. When a switch, $S_{i,j}$, receives $RR_{PE_{i-1,j-1}}^{S_{i,j}}$ it generates $RR_{S_{i,j}}^{S_{i-1,j}}$ to $S_{i-1,j}$, which is used to decode the relative location of $S_{i-1,j}$ with respect to the failed PE.

Once the reconfiguration request reaches $PE_{m,j}$, $PE_{m,j}$ generates $RR_{PE_{m,j}}^{PE_{m+1,j}}$, which is given to the central processor. The central processor delays the next CLK_{PE} edge, $t+1$. Each $PE_{x,j}$ $(i < x \leq m)$ generates the RRs to the switches and the switches are reconfigured in two stages:

1. At t_1, the vertical switches, $S_{x,j+1}^{V}$ $(i \leq x \leq m-1)$ are brought to state ST_1^V, $S_{m,j+1}^{V}$ and $S_{m+1,j+1}^{V}$ remain in ST_1^V and the horizontal switches, $S_{x,j}^{H}$ $(i \leq x < m+1)$ toggle either from state ST_0^H to ST_1^V or from ST_1^H to ST_0^H. At the same time, $PE_{x,j}$ $(i < x \leq m)$ start using the static coefficient latch '1' and select the horizontal input port I_{PE1}^{H} for use.

2. At $t+1$, the vertical switches, $S_{x,j+1}^{V}$ $(i+1 < x \leq m+1)$ change state from ST_1^V to ST_0^V and the horizontal switches, $S_{x,j+1}^{H}$ $(i < x \leq m+1)$ toggle either from state ST_0^H to ST_1^H or from ST_1^H to ST_0^H. This completes the reconfiguration and then the normal clock speed is resumed.

3.4 Concluding Remarks

In this chapter an on-line reconfiguration algorithm for PE failures was discussed. Here an extra row of cells (called spares) is provided to the array and in the case of a detected PE failure global shift is performed for the corresponding column.

The staging latches were shifted from the input side to the output side to facilitate the full use of non-faulty partial results. The PEs are provided with an additional static coefficient latch to avoid reloading of static coefficients in the case of a PE failure. The testing circuit and control circuit are added in the PEs to detect the fault and generate the reconfiguration requests. In addition, the control circuit selects the proper input data ports after the rerouting.

The network is modified to support the algorithm and switches are designed as finite state machines. It was proved that the reconfiguration requires a maximum of two stages of rerouting and the proposed two states of vertical and horizontal switches provide the required data paths.

It is assumed that the control circuit of the *PEs* never fails, it is essential to ensure the proper operation of the algorithm. Failure of control circuit may lead to an unsafe fatal failure. To achieve this feature, control circuit can be provided with active redundancy. The assumption of sequential failures (faults occurring one at a time) is made to simplify the modelling of the algorithm. This algorithm can tolerate simultaneous multiple failures if the failures are not in adjacent columns.

It is assumed that the occurrence of a failure is reported to the central processor before the arrival of next clock edge. This can be ensured by making the clock period slightly longer. The time between the occurrence of a failure and fault reporting depends upon the number of rows in the array. Therefore for an array with small number of rows the speed reduction due to extended clock will be very little.

The above algorithm is modified to accommodate the link failures too and the modified algorithm is discussed in the next chapter.

Chapter 4

ALGORITHM FOR *PE* AND LINK FAILURE TOLERANCE

The basic principle of this algorithm is the same as explained earlier: a bottom row of spares is provided to the array of size $(m \times n)$ and if $PE_{i,j}$ fails, $PE_{i,j}$ is replaced by $PE_{m,j}$ if $PE_{m,j}$ is available.

A link failure for the link $L_{PE_{i-1,j}}^{PE_{i,j}}$ is detected by $PE_{i,j}$ by using parity bit checks. To tolerate the link failures, each link is duplicated.

Here, the following assumptions are made:

Assumptions:

- the faults are occurring one at a time;
- the link failures are detected by PEs (here even an intermittent data error is taken as link failure);
- switches perform self-test only for the control circuit (they do not test the actual switching circuitry because any fault in a switching circuit results in a data error, which is detected by PEs);
- once a PE fails, it detects the self-fault;
- the control circuitry of a PE never fails;
- the self-testing blocks of PEs and switches never fail;

- a central processor provides input and clock to the array and it receives output and fault occurrence signals from the array;
- the occurrence of a failure is reported to the central processor before the arrival of next rising clock edge and
- the central processor provides clock pulses (CLK_S) to switches also, if any PE fails.

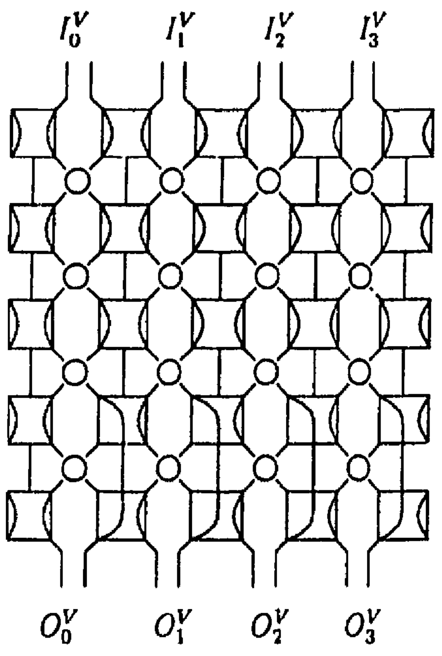
4.1 Data Routing

As explained earlier, the algorithm (for PE failures) needs one vertical and two horizontal links between PEs; consequently now two vertical and four horizontal links are provided (link redundancy). The vertical and horizontal data routings are discussed separately in the following subsections.

4.1.1 Vertical Data Routing Path (for PE and Link failures)

The network for vertical data is shown in Figure 4.1.

The input links from the central processor and output links to the central processor are also duplicated. Each PE has two vertical inputs (I^V_{PE0} and I^V_{PE1}) and two vertical outputs (O^V_{PE0} and O^V_{PE1}) as shown in Figure 4.1. Similarly, each vertical switch is a 3×3 switch (with three inputs, $I^V_{S0}, I^V_{S1}, I^V_{S2}$ and three outputs $O^V_{S0}, O^V_{S1}, O^V_{S2}$). To support the algorithm, a total of eight states of the vertical switches are provided as shown in Figure 4.2. Initially, all switches $S^V_{i,j}$ ($0 \leq i \leq m-1; 0 \leq j \leq n$) are in state ST^V_0, switches $S^V_{m,j}$ ($1 \leq j \leq n$) are in ST^V_1 and switches $S^V_{m+1,j}$ ($1 \leq j \leq n$) are in state ST^V_2 (see Figure 4.1). The switches, $S^V_{m,0}$ and $S^V_{m+1,0}$ are in ST^V_0. The vertical input to the array is applied through the I^V_{S0} and I^V_{S2} ports of the switches $S_{0,j}$ ($0 \leq j \leq n$). The output O^V_j appears at O^V_{S0} of $S_{m+1,j+1}$ ($0 \leq j < n$) in the case of no output link failure.



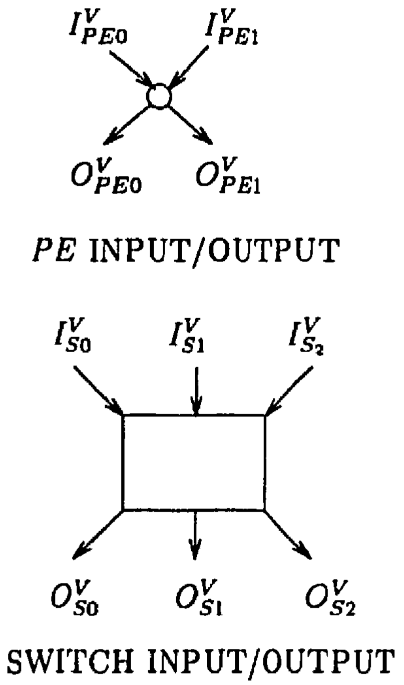


Figure 4.1: Vertical Data Path (for PE and Link failures)

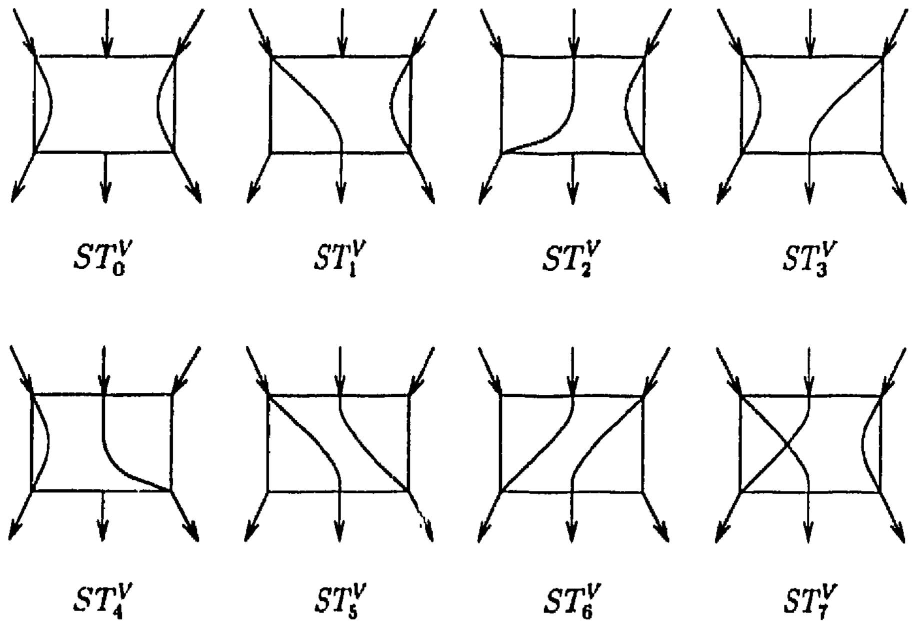


Figure 4.2: States of the Vertical Switches (For combined PE and Link failures)

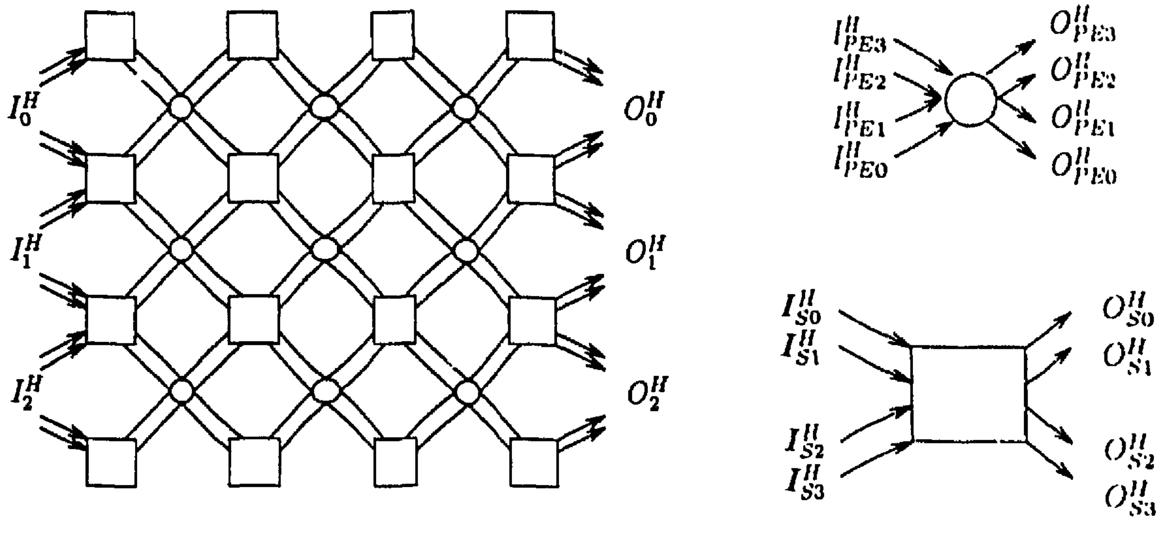


Figure 4.3: Horizontal data Path (Combined PE and Link failure)

4.1.2 Horizontal Data Routing Path (for PE and Link failures)

The network is shown in Figure 4.3. The horizontal inputs and outputs of the PEs and switches are also shown in Figure 4.3. Each PE has two pairs of horizontal inputs (I_{PE0}^H, I_{PE1}^H and I_{PE2}^H, I_{PE3}^H). To support the algorithm, a total of two states of the horizontal switches are provided as shown in Figure 4.4. Initially all the horizontal switches are in state ST_0^H.

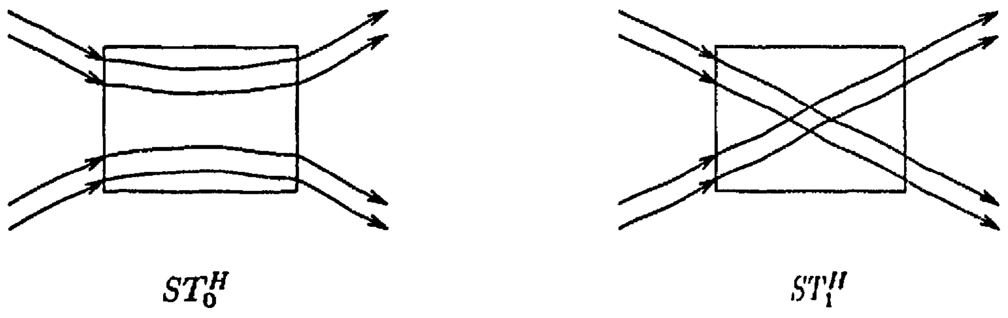


Figure 4.4: Horizontal Switch States (Combined PE and Link failure)

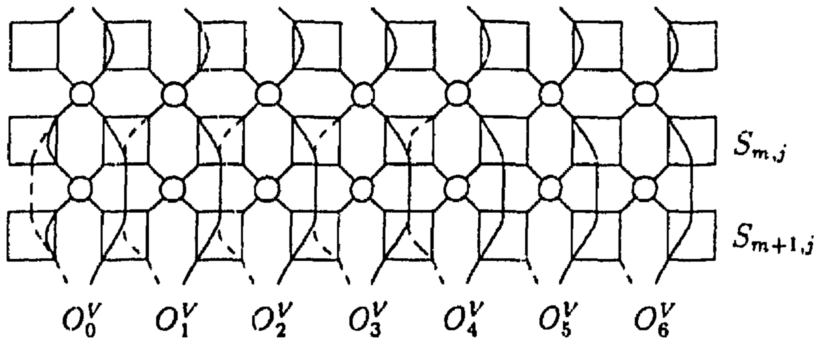


Figure 4.5: Switch State Changes (Combined PE and Link failure)

In the next section, handling of link failure is explained.

4.2 Handling of a Link failure

Normally, $PE_{i,j}$ processes the data available at its I_{PE0}^V and I_{PE0}^H ports and when it detects a fault in the data, the PE selects port I_{PE1}^H for horizontal input (in the case of a horizontal data fault). For a vertical data fault, it checks the switch $S_{i,j+1}^V$ and if $S_{i,j+1}^V$ is in state ST_0^V, it selects the vertical input port I_{PE1}^V. For an output O_j^V fault (as stated earlier, the central processor detects this fault), switch $S_{m+1,j+1}^V$ is checked and since it is in state ST_2^V, it invokes a reconfiguration of vertical switches because here, it cannot use the data available at I_{PE1}^V. In this case, $S_{m+1,j+1}^V$ and $S_{m,j+1}^V$ change the states of $S_{m+1,j}^V$ and $S_{m,j}^V$ from ST_0^V & ST_2^V to ST_4^V and from ST_0^V & ST_1^V to ST_3^V respectively. These switches $S_{m+1,j}^V$ and $S_{m,j}^V$ change the states of $S_{m+1,j-1}^V$ and $S_{m,j-1}^V$ again and so on, until $S_{m+1,y}^V$ finds a switch $S_{m+1,y-1}^V$ in state ST_0^V or in ST_4^V (here the algorithm assumes that though the link $L_{PE_{m-1,y-1}}^{O_{y-1}}$ is faulty, the switches $S_{m,y}^V$, $S_{m+1,y}^V$ and link $L_{S_{m,y}}^{S_{m+1,y}}$ may not be faulty).

If in the array, shown in Figure 4.5, O_2^V fails, the algorithm changes the states of $S_{m+1,2}^V, S_{m+1,1}^V$ and $S_{m+1,0}^V$ to ST_4^V and switches $S_{m,2}^V, S_{m,1}^V$ and $S_{m,0}^V$ are brought to state ST_3^V and outputs O_j^V $(0 \leq j \leq 2)$ are taken through the second output port.

Next, when O_4^V becomes faulty, it changes the states of $S_{m+1,j}^V$ $(2 < j \leq 4)$ to ST_4^V and $S_{m,j}^V$ $(2 < j \leq 4)$ to ST_3^V and outputs O_j^V $(2 < j \leq 4)$ are taken through the second port.

The PE failure and the link failure algorithms are combined in the next section.

4.3 Combined PE and Link Failure

Here PE and Link failures are discussed separately for the sake of clarity.

4.3.1 PE Failure (in presence of faulty Links)

As explained earlier, a PE failure is handled in two stages.

Vertical Data Routing - A $PE_{i,j}$ failure affects the states of switches $S_{x,j+1}^V$ $(i \leq x \leq m+1)$. These switches can be in any of the states ST_0^V, ST_3^V, ST_4^V and ST_5^V depending on the occurrence of earlier faults (they cannot be in states ST_6^V and ST_7^V because these states can be reached only if there is a failed $PE_{x,j}$ $(0 \leq x \leq m-1)$, in which case the algorithm fails now due to the non-availability of a spare cell).

When a $PE_{i,j}$ fails at t_1, it starts the re-routing, which is done in two stages. The changes required for the intermediate stage and the final stage are listed below:
Intermediate stage - all switches $S_{x,j+1}^V$ $(i \leq x \leq m+1)$ change state depending on their current state. $S_{i,j+1}^V$ changes from ST_0^V to ST_1^V or from ST_4^V to ST_5^V. If it is in ST_3^V, it stays in ST_3^V. Other switches $S_{x,j+1}^V$ $(i < x \leq m+1)$ change from ST_0^V, ST_1^V and ST_2^V to ST_7^V, from ST_3^V to ST_6^V and from ST_4^V to ST_5^V. If $S_{x,j+1}^V$ is in ST_5^V, it stays in ST_5^V. When a switch $S_{i1,j+1}^V$ $(i1 > i)$ is in state ST_4^V or in ST_5^V, the partial result of the $PE_{i1-2,j}$ does not reach $PE_{i1,j}$ by above changes. So when a switch $S_{i1,j+1}^V$ is in state ST_4^V, the algorithm checks the switches $S_{i1,x}^V$ $(0 \leq x \leq j)$ and finds a switch $S_{i1,j1}^V$ $(0 \leq j1 \leq j)$ (nearest to $S_{i1,j+1}^V$), which is not

in ST_2^V (when $S_{i1,j+1}^V$ is in ST_4^V, the algorithm checks $S_{i1,j}^V$ and if $S_{i1,j}^V$ is in ST_2^V, the algorithm checks $S_{i1,j-1}^V$. Again, if $S_{i1,j-1}^V$ is in ST_2^V, the algorithm checks $S_{i1,j-2}^V$. In this way, the algorithm goes towards $S_{i1,0}^V$ and finds $S_{i1,j1}^V$, which is not in ST_2^V).

Lemma 4.1 - The switch $S_{i1,j1}^V$ cannot be in ST_2^V, ST_3^V, ST_6^V and ST_7^V; so it can be only in one of the states ST_0^V, ST_1^V, ST_4^V and ST_5^V.

Proof - If a switch S_{i1,j_x}^V is in ST_2^V, the algorithm checks S_{i1,j_x-1}^V as specified in the algorithm and S_{i1,j_x}^V is not defined as $S_{i1,j1}^V$.

When a switch S_{i1,j_x}^V is in ST_3^V or ST_6^V, it means that S_{i1,j_x+1}^V is in ST_1^V and then S_{i1,j_x+1}^V would be taken as $S_{i1,j1}^V$ and the algorithm will not check S_{i1,j_x}^V.

The switch, $S_{i1,j1}^V$ cannot be in ST_7^V, because it is assumed that failures occur one at a time (and a switch can be in ST_7^V only during the intermediate stage of re-routing). □

Corollary 4.1.1 - The switch, $S_{i1-1,j1}^V$ cannot be in states ST_4^V, ST_5^V and ST_7^V.

Proof - When a switch, S_{i1-1,j_x}^V is in ST_4^V or ST_5^V, it means that S_{i1-1,j_x+1}^V is in ST_2^V and then S_{i1,j_x+1}^V cannot be in ST_2^V (as will be shown in Lemma 4.2). So, here the algorithm will take S_{i1,j_x+1}^V as $S_{i1,j1}^V$ and it will not check S_{i1,j_x}^V. This proves that the switch, $S_{i1-1,j1}^V$ cannot be in ST_4^V or in ST_5^V. $S_{i1-1,j1}^V$ cannot be in ST_7^V because only one failure occurs at a time. □

Corollary 4.1.2 - When the switch, $S_{i1,j1}^V$ is in ST_0^V, $S_{i1-1,j1}^V$ must be either in ST_0^V or in ST_2^V.

Proof - If $S_{i1-1,j1}^V$ is in ST_1^V, it would require $S_{i1,j1}^V$ to be in ST_2^V, which is not possible. Similarly, $S_{i1-1,j1}^V$ cannot be ST_3^V or ST_6^V, because these conditions require $S_{i1,j1}^V$ to be in ST_4^V.

So, $S_{i1-1,j1}^V$ must be either in ST_0^V or in ST_2^V. □

Corollary 4.1.3 - When the switch, $S_{i1,j1}^V$ is in ST_1^V, $S_{i1-1,j1}^V$ can be only in ST_0^V.

Proof - When $S_{i1,j1}^V$ is in ST_1^V, the switch, $S_{i1+1,j1}^V$ would be in ST_2^V. So, $S_{i1-1,j1}^V$ cannot be in ST_1^V or ST_2^V because in a column only one switch can be in ST_1^V and

ST_2^V (as will be proved in Lemma 4.2). The switch $S_{i1-1,j1}^V$ cannot be in ST_3^V or ST_6^V, because it would require $S_{i1,j1}^V$ to be in ST_4^V.

So, $S_{i+1,j1}^V$ must be in ST_0^V. □

Corollary 4.1.4 - When the switch, $S_{i1,j1}^V$ is in ST_4^V or in ST_5^V, $S_{i1-1,j1}$ must be in ST_3^V. □

If the switch $S_{i1,j1}^V$ is in state ST_0^V or ST_1^V, the switches $S_{i1,y}^V$ $(j1 \le y \le j)$ change state from ST_0^V and ST_2^V to ST_4^V or from ST_1^V to ST_5^V and switches $S_{i1-1,y}^V$ $(j1 \le y \le j)$ change state from ST_0^V and ST_1^V to ST_3^V or from ST_2^V to ST_6^V. At the same time $PE_{i1,y}$ $(j1 < y \le j)$ start using the second vertical input port I_{PE1}^V.

If the switch $S_{i1,j1}^V$ is in state ST_4^V or ST_5^V (meaning that $S_{i1,j1-1}^V$ is in ST_2^V and the link, $L_{PE_{i1-2,j1}}^{PE_{i1,j1}}$ is faulty), the switches $S_{i1,y}^V$ $(j1 < y \le j)$ change state from ST_2^V to ST_4^V and switches $S_{i1-1,y}^V$ $(j1 < y \le j)$ change to ST_3^V. Here it assumes that though the link $L_{PE_{i1-2,j1}}^{PE_{i1,j1}}$ is faulty, the switches $S_{i1-1,j1+1}^V$, $S_{i1,j1+1}^V$ and link $L_{S_{i1-1,j1+1}}^{S_{i1,j1+1}}$ may not be faulty. At the same time, $PE_{i1,y}$ $(j1 < y \le j)$ select their second vertical input port. If after selecting the second port, any PE detects an input error, a fatal failure occurs.

Final Stage - At the next clock edge, $t+1$, the algorithm does the following:

It changes $S_{x,j+1}^V$ $(i+2 \le x \le m+1)$ from ST_7^V to ST_0^V, from ST_5^V to ST_4^V or from ST_6^V to ST_3^V and in the case of a change from ST_5^V to ST_4^V of switch $S_{i1,j+1}^V$, if $i1 > i+1$, the switches $S_{i1,y}^V$ and $S_{i1+1,y}^V$ (y was defined earlier in the intermediate stage) are brought back to the states in which they were before intermediate re-routing. $S_{i,j+1}^V$ does not change state during the final stage and $S_{i+1,j+1}^V$ changes state to ST_2^V if it were in ST_7^V.

Lemma 4.2 - In a column j, only one switch $S_{i,j}^V$ can be in state ST_1^V.

Proof - Switch $S_{i,j}^V$ goes to state ST_1^V if and only if vertical data routing path requires bypassing of $PE_{i,j-1}$.

When column $(j-1)$ of the PEs does not have any faulty PE, the spare cell, $PE_{m,j-1}$, is bypassed by bringing $S_{m,j}^V$ to ST_1^V and $S_{m+1,j}^V$ to ST_2^V.

When column $(j-1)$ of the PEs has a faulty cell, $PE_{i,j-1}$, then this PE is bypassed by bringing $S_{i,j}^V$ to ST_1^V and $S_{i+1,j}^V$ to ST_2^V. In this case the spare cell, $PE_{m,j-1}$ becomes an active cell and switches $S_{m,j}^V$ and $S_{m+1,j}^V$ go to ST_0^V. Now, if another switch, $S_{i1,j}^V$ is in ST_1^V, it means that $PE_{i1,j-1}$ is faulty and it implies that column $(j-1)$ of PEs has two faulty cells. Since the algorithm can tolerate only one PE failure in a column, this condition leads to a fatal failure.

So, in a working array, only one switch in a column can be in ST_1^V. □

Corollary 4.2.1 - In a column j, only one switch can be in state ST_2^V. □

Corollary 4.2.2 - In a column j, only one switch can be in state ST_3^V.

Proof - As shown in Lemma 4.2, column $(j+1)$ can have only one switch, $S_{i,j+1}^V$ in state ST_1^V. In this case, $S_{i+1,j+1}^V$ would be in ST_2^V and these two switches provide the link $L_{PE_{i-1,j-1}}^{PE_{i+1,j+1}}$ (vertical data path between $PE_{i-1,j-1}$ and $PE_{i+1,j-1}$; $PE_{i,j-1}$ is bypassed).

At this stage, when $PE_{i+1,j-1}$ detects vertical data error (vertical input link failure), the algorithm brings $S_{i,j}^V$ to ST_3^V (which will be discussed in the next subsection - link failure handling). When $S_{i,j}^V$ goes to ST_3^V, $S_{i+1,j}^V$ goes to ST_4^V and these two provide an alternative data path between $PE_{i-1,j-1}$ and $PE_{i+1,j-1}$. Since in a column $(j+1)$, only one switch $S_{i,j+1}^V$ can be in ST_1^V, column j can have only one switch $S_{i,j}^V$ in ST_3^V. □

The following three corollaries can be proved similarly.

Corollary 4.2.3 - In a column j, only one switch can be in state ST_4^V. □

Corollary 4.2.4 - In a column j, only one switch can be in state ST_5^V. □

Corollary 4.2.5 - In a column j, only one switch can be in state ST_6^V. □

Lemma 4.3 - In the event of an active $PE_{i,j}$ failure, switch $S_{i,j+1}^V$ can not be in state ST_1^V, before the reconfiguration starts.

Proof - Since before the reconfiguration starts, $PE_{i,j}$ is active and it is getting data from $PE_{i-1,j}$ and providing the partial results to $PE_{i+1,j}$, the switch, $S_{i,j+1}$ cannot be in state ST_1^V. □

The following corollary can be proved similarly.

Corollary 4.3.1 - In the case of an active $PE_{i,j}$ failure, switch $S_{i+1,j+1}^V$ can not be in ST_2^V, before the reconfiguration starts. □

Some representatives cases of $PE_{i,j}$ failure handling are discussed next (see Figure 4.6).

Case-A shows the array reconfiguration, where the column $(j+1)$ of switches provides all data links required by the intermediate and final stages. Here, before the failure, the switches, $S_{i_r,j+1}^V$ $(i \leq i_r < m)$ are in ST_0^V, $S_{m,j+1}^V$ is in ST_1^V and $S_{m+1,j+1}^V$ is in ST_2^V.

When $S_{i,j+1}^V$ is in state ST_4^V (Case-B), then also all the paths are made available by changing $S_{i,j+1}^V$ to ST_5^V and other switches to ST_7^V for the intermediate stage and then by changing $S_{i+1,j+1}^V$ to ST_2^V from ST_7^V and other $S_{r,j+1}^V$ $(i+2 \leq r \leq m+1)$ from ST_7^V to ST_0^V for final stage.

But when a switch, $S_{i1,j+1}^V$ $(i1 > i)$ is in state ST_4^V, the path $L_{PE_{i1-2,j}}^{PE_{i1,j}}$ cannot be provided by the switches in the column $j+1$. So other switches in row i and $i-1$ are modified.

In Case-C, $PE_{i,j}$ fails and the intermediate stage requires a link between $PE_{i1-2,j}$ and $PE_{i1,j}$, which cannot be provided by the switches in the $(j+1)th$ column. So the algorithm finds $S_{i1,j1}^V$ in state ST_0^V. Due to an earlier $PE_{i1-1,j-1}$ failure, $S_{i1-1,j}^V$ is in ST_1^V and $S_{i1,j}^V$ is in ST_2^V. For intermediate stage routing, $S_{i1-1,j-1}^V$ and $S_{i1-1,j}^V$ are brought to ST_3^V and $S_{i1,j1}^V$ and $S_{i1,j}^V$ are brought to ST_4^V.

Case-D is similar to Case-C, but here $S_{i1,j1}^V$ is in ST_1^V. So, $S_{i1,j1}^V$ is changed to ST_5^V. The other changes are the same as written for Case-A.

Case-E is another variation of Case-C, here $S_{i1-1,j1}^V$ is in ST_2^V. $S_{i1-1,j1}^V$ is changed

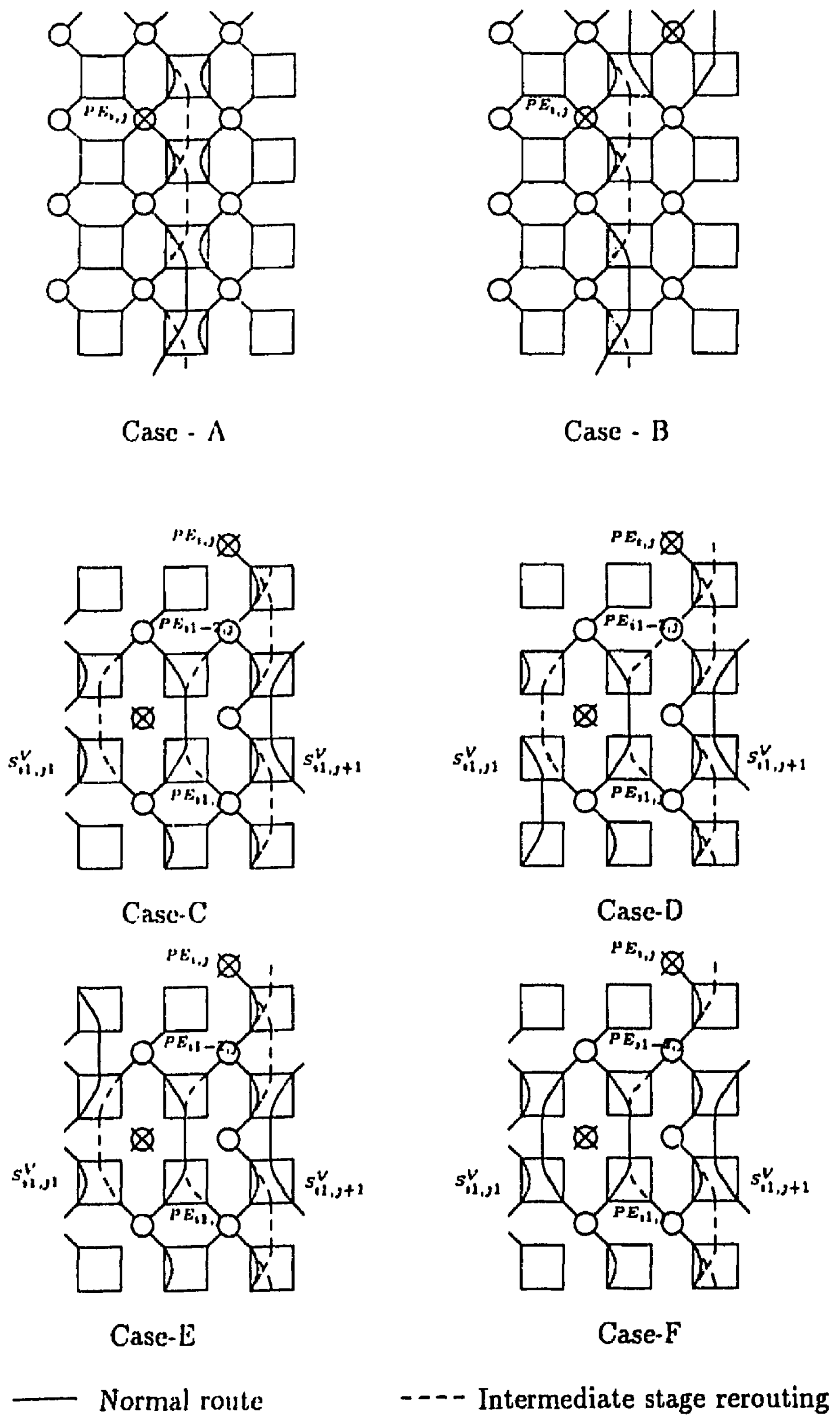


Figure 4.6: Vertical Switch State Changes (Combined PE and Link failure)

to ST_6^V and other changes are same as written in Case-A.

In Case-F, $S_{i1-1,j1}^V$ is in ST_3^V and $S_{i1,j1}^V$ is in ST_4^V. Here $S_{i1-1,j}$ is changed to ST_3^V and $S_{i1,j}^V$ is changed to ST_4^V. The switches $S_{i1-1,j1}^V$ and $S_{i1,j1}^V$ are in states ST_3^V and ST_4^V because $PE_{i1,j1}^V$ detected a link failure earlier (and $PE_{i1-1,j1+1}$ was faulty that time). The latest reconfiguration assumes that though the link $L_{PE_{i1-2,j-1}}^{PE_{i1,j-1}}$ is faulty, link $L_{S_{i1-1,j}}^{S_{i1,j}}$ is not faulty. In the case of faulty $L_{S_{i1-1,j}}^{S_{i1,j}}$ a fatal failure occurs.

If the links, which are newly generated by using the switches of columns j_r ($j_x < j$), are required by the final stage (when $i1 = i + 1$), the algorithm does not change the states of the switches generating these links. Otherwise at $t + 1$, these switches go back to their prior-to-t_1-state.

Horizontal Data Routing - This is exactly similar to the horizontal data routing explained in the previous algorithm (only PE fail algorithm). The reconfiguration, invoked due to $PE_{i,j}$ failure, changes the states of the switches $S_{i_r,j}^H$ ($i < i_r \leq m+1$), either from ST_1^H to ST_0^H or from ST_0^H to ST_1^H during the intermediate stage of rerouting. During the final stage, the states of the switches, $S_{i_r,j+1}^H$ ($i < i_r \leq m+1$) is changed, either from ST_0^H to ST_1^H or from ST_1^H to ST_0^H.

4.3.2 Link Failure (in presence of faulty PEs)

As explained earlier, each link is duplicated here. When a PE detects an error in the data, available at its first input port, it invokes a reconfiguration and selects the second input port. If a PE is using second input port, input data error leads to fatal failure. The vertical and horizontal data paths are discussed separately.

Vertical Path - When a $PE_{i,j}$ detects a fault in its input data (at port I_{PE0}^V), it does the following:

- if $S_{i,j+1}^V$ is in ST_0, ST_3^V or in ST_4^V, then the $PE_{i,j}$ simply selects the other input port (I_{PE1}^V),

- else if $S^V_{i,j+1}$ is in ST^V_2, ST^V_6 or in ST^V_7, then the algorithm checks $S^V_{i,j}$ and if it is in ST^V_2, the algorithm checks $S^V_{i,j-1}$ and so on, until it finds a switch $S^V_{i,x}$ $(0 \leq x < j)$, which is not in state ST^V_2 (here, $S^V_{i,x}$ would be in one of the states ST^V_0, ST^V_1, ST^V_4 or ST^V_5, lemma 4.1 shows this) and then:
  - if $S^V_{i,x}$ is in ST^V_0, it changes all $S^V_{i-1,y}$ $(x \leq y \leq j)$ from ST^V_0 to ST^V_3, from ST^V_7 to ST^V_6 or from ST^V_2 to ST^V_6 and changes all $S^V_{i,y}$ $(x \leq y \leq j)$ to ST^V_4 from ST^V_0 and ST^V_2 or to ST^V_5 from ST^V_7 and select input port I^V_{PE1} for all $PE_{i,y}$ $(x \leq y \leq j)$,
  - else if $S^V_{i,x}$ is in ST^V_1, it changes $S^V_{i-1,x}$ to ST^V_3, $S^V_{i,x}$ to ST^V_5 and changes all $S^V_{i-1,y}$ $(x < y \leq j)$ to ST^V_3 from ST^V_1, to ST^V_6 from ST^V_7 and all $S^V_{i,y}$ $(x < y \leq j)$ to ST^V_4 from ST^V_2, to ST^V_5 from ST^V_7 and select input port I^V_{PE1} for $PE_{i,y}$ $(x \leq y \leq j)$,
  - else if $S^V_{i,x}$ is in state ST^V_4 or in ST^V_5, it changes $S^V_{i-1,y}$ $(x < y \leq j)$ to ST^V_3 from ST^V_1, to ST^V_6 from ST^V_7 and all $S^V_{i,y}$ $(x < y \leq j)$ to ST^V_4 from ST^V_2, ST^V_5 from ST^V_7 and select input port I^V_{PE1} for $PE_{i,y}$ $(x < y \leq j)$.

Some link failure reconfigurations are shown in Figure 4.7. Only the paths, which are modified, are shown.

In Case-A, link failure is detected by $PE_{i,j}$, but since the switch $S^V_{i,j+1}$ is in ST^V_0, no reconfiguration is done, $PE_{i,j}$ simply selects the other input port. Similarly, in Case-B, $S^V_{i,j+1}$ is in ST^V_4, so no re-routing is done and the second input port is selected.

In Case-C, link failure is detected by $PE_{i,j}$ and the switch $S^V_{i,j+1}$ is in ST^V_2 due to the earlier failure of $PE_{i-1,j}$. Now the algorithm finds $S^V_{i,j-1}$ in ST^V_0 and modifies $S^V_{i-1,j-1}$, $S^V_{i-1,j}$ to ST^V_3 and $S^V_{i,j-1}$, $S^V_{i,j}$ to ST^V_4.

Case-D is similar to Case-C, but here $S^V_{i-1,j-1}$ is in ST^V_2, so it is brought to ST^V_6.

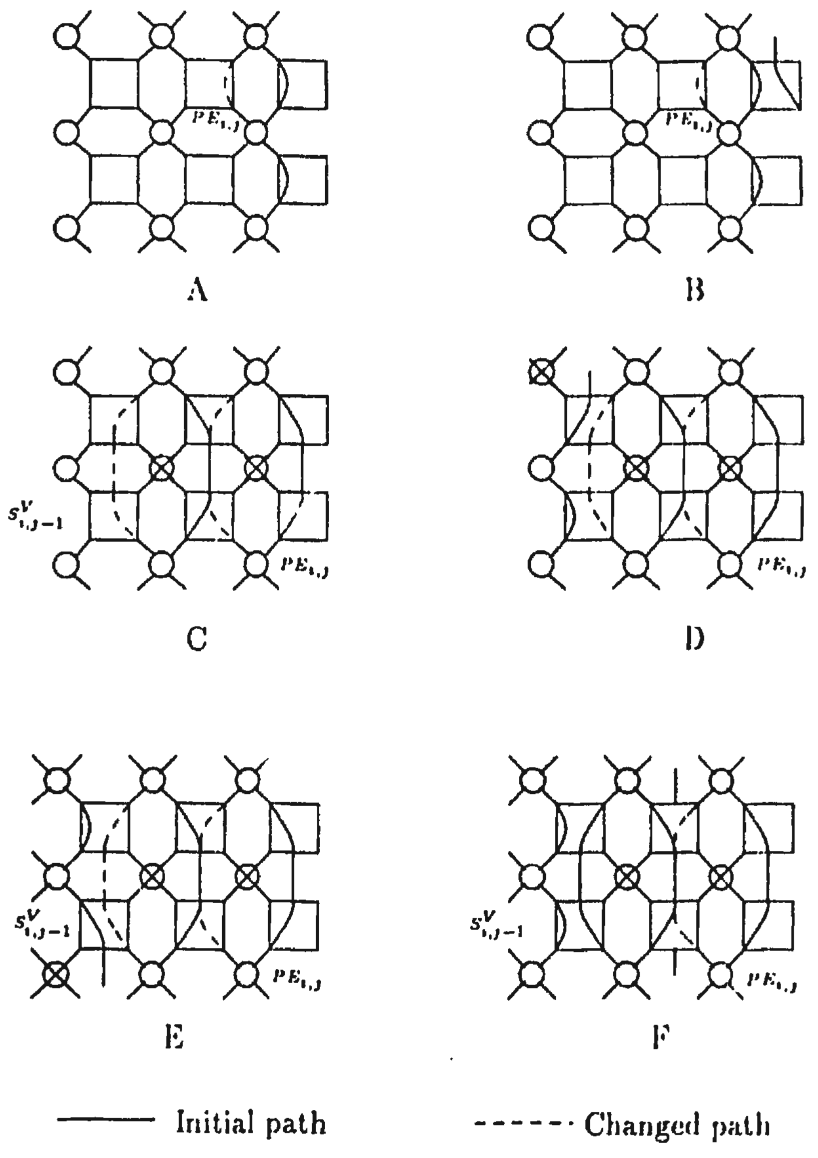


Figure 4.7: Link failure Reconfigurations

In Case-E, $PE_{i,j}$ detects the link failure and $S^V_{i,j+1}$ is in ST^V_2, so the algorithm finds $S^V_{i,j-1}$ in ST^V_1 and changes it to ST^V_5. The states of $S^V_{i-1,j-1}$, $S^V_{i-1,j}$ are changed to ST^V_3 and $S^V_{i,j}$ is changed to ST^V_4.

In Case-F, $S^V_{i,j+1}$ is in ST^V_2 and $S^V_{i,j-1}$ is found in ST^V_4. Here. S^V_{ij} is changed to ST^V_4 and $S^V_{i-1,j}$ is changed to ST^V_3. It is assumed that though the link $L^{PE_{i-1,j-1}}_{PE_{i,j-1}}$ is faulty, the link, $L^{S_{i-1,j}}_{S_{i,j}}$ is not faulty. If this link is also faulty, $PE_{i,j}$ would again detect vertical input error and it would cause a fatal failure.

Horizontal Path - When a $PE_{i,j}$ detects a fault in its horizontal input data, it performs the following operations:

- if it is using I^H_{PE0}, it selects I^H_{PE1},
- else if it is using I^H_{PE2}, it selects I^H_{PE3},
- otherwise the algorithm fails and fatal failure occurs.

Theorem 4.1 - The cases shown in Figure 4.6 represent all the possible combinations of vertical switch states, in the case of $PE_{i,j}$ failure.

Proof - The reconfiguration, in the case of $PE_{i,j}$ failure reroutes the vertical data by changing the states of the switches in column $(j+1)$. The states of the switches in column $(j+1)$ depend on earlier PE and link failures in column $(j+1)$ of PEs. There are only four combinations of failures in column $(j+1)$ of PEs and these are listed below.

1. Column $(j+1)$ of PEs has no faulty PE or no faulty link,
2. column $(j+1)$ of PEs has a faulty PE,
3. column $(j+1)$ of PEs has only link failures and
4. column $(j+1)$ of PEs has both PE and link failures.

The effect of each of these failures on the reconfiguration (which is invoked due to $PE_{i,j}$ failure) is discussed separately.

Column $j+1$ has no faulty PE or no faulty link - here, column $(j+1)$ of switches would be in its initially set state and this corresponds to Case-A of Figure 4.6.

Column $j+1$ has only one faulty PE - here also, column $(j+1)$ of switches would be in its initially set state and this corresponds to Case-A of Figure 4.6.

Column $j+1$ has only link failures - here the output links may be or may not be faulty. When the output link is not faulty, column $(j+1)$ of switches would not be disturbed by these link failures and this is covered in Case-A of Figure 4.6.

When the output link is faulty, $S^V_{m,j+1}$ would be in ST^V_3 and $S^V_{m+1,j+1}$ would be in ST^V_4. In this case, the link $L^{CP}_{PE_{m-1,j}}$ (the link that carries the vertical result to the central processor from $PE_{m-1,j}$) is not provided by the switches in column $(j+1)$. Here, all switches, S^V_{m+1,j_x} $(0 \leq j_x \leq j+1)$ would be in ST^V_4 (as explained in the subsection 4.2). This corresponds to Case-F of Figure 4.6. Here $j1 = j$, so no switches would change state. The only difference here is that for Case-F, it was assumed that though the link, $L^{PE_{i1-2,j1}}_{PE_{i1,j1}}$ is faulty, link $L^{S_{i1,j1}}_{S_{i1-1,j1}}$ is not faulty, but now this assumption is not required because $S^V_{m+1,j}$ did not reach ST^V_4 due to the link failure detected by column j of switches.

Column $j+1$ has both faulty PE and faulty links - in this case, the failures affect column $(j+1)$ of switches only if $PE_{i2,j+1}$ fails and vertical input error is detected by $PE_{i2+1,j+1}$. It brings $S^V_{i2,j+1}$ to ST^V_3 and $S^V_{i2+1,j+1}$ to ST^V_4.

When $(i2 < i-1)$, the switches, $S^V_{i_x,j+1}$ $(i \leq i_x \leq m+1)$ are not affected by the above mentioned failures. This condition corresponds to Case-A of Figure 4.6.

When $(i2 = i-1)$, the switch, $S^V_{i-1,j+1}$ would be in ST^V_3 and $S^V_{i,j+1}$ would be in ST^V_4. This condition corresponds to Case-B of Figure 4.6.

When $(i2 > i-1)$, one of the links required by the intermediate stage (namely link $L_{PE_{i2-1,j}}^{PE_{i2+1,j}}$) would not be provided by the column $(j+1)$ of switches. Here, switch $S_{i2+1,j+1}^{V}$ is renamed as $S_{i1,j+1}^{V}$ for the sake of clarity and it can be either in ST_4^V or in ST_5^V. Once the algorithm finds $S_{i1,j+1}^{V}$ in ST_4^V or in ST_5^V, it checks $S_{i1,j}^{V}$ and if it is in ST_2^V, the algorithm checks $S_{i1,j-1}^{V}$ and so on, until it finds a switch, $S_{i1,j1}^{V}$, which is not in ST_2^V. Clearly all switches, S_{i1,j_x}^{V} $(j1 < j_x \leq j)$ would be in ST_2^V and S_{i1-1,j_x}^{V} would be in ST_1^V. The switch, $S_{i1,j1}^{V}$ can be in any of the states ST_0^V, ST_1^V, ST_4^V and ST_5^V (as proved in Lemma 4.1).

When $S_{i1,j1}^{V}$ is in ST_0^V, $S_{i1-1,j1}^{V}$ can only be either in ST_0^V or in ST_2^V (as proved in Corollary 4.1.2) and these two conditions correspond to Case-C and Case-D of Figure 4.6 respectively.

When $S_{i1,j1}^{V}$ is in ST_1^V, $S_{i1-1,j1}^{V}$ would be in ST_0^V (as proved in Corollary 4.1.3) and it corresponds to Case-E of Figure 4.6.

When $S_{i1,j1}^{V}$ is in ST_4^V or in ST_5^V, $S_{i1-1,j1}^{V}$ would be in ST_3^V (as proved in Corollary 4.1.4) and it corresponds to Case-F of Figure 4.6. □

Theorem 4.2 - The cases shown in Figure 4.7 represent all the possible combinations of vertical switch states in the case of a vertical link failure detected by $PE_{i,j}$.

Proof - When a vertical link failure is detected by $PE_{i,j}$, the reconfiguration depends on the state of switch $S_{i,j+1}^{V}$. It can be in any state depending on the occurrence of earlier faults.

When $S_{i,j+1}^{V}$ is in ST_0^V, ST_3^V or in ST_4^V, it means that $PE_{i,j}$ is receiving the input from $PE_{i-1,j}$ and it corresponds to Case-A of Figure 4.7.

When $S_{i,j+1}^{V}$ is either in ST_1^V or in ST_5^V, $PE_{i,j}$ is faulty and no reconfiguration is invoked.

When $S_{i,j+1}^{V}$ is in ST_2^V, ST_6^V or ST_7^V (here it can be in ST_7^V because the intermediate stage of rerouting always instructs the PEs to use the switches of column $(j+1)$ and it may have faulty links before the failure of $PE_{i,j}$), $PE_{i,j}$ receives vertical input from $PE_{i-2,j}$ and in this case an alternative vertical data path is required. For this, the algorithm checks the switch, $S^V_{i,j}$ and if it is in ST^V_2, the algorithm checks $S^V_{i,j-1}$. If $S^V_{i,j-1}$ also is in ST^V_2, the algorithm checks $S^V_{i,j-2}$ and so on, until it find a switch, $S^V_{i,j1}$, which is not in ST^V_2 ($S_{i,j1}$ can be in any of the states ST^V_0, ST^V_1, ST^V_4 or ST^V_5, as proved in Lemma 4.1).

When $S^V_{i,j1}$ is in ST^V_0, $S^V_{i-1,j1}$ can be either in ST^V_0 or in ST^V_2 (Corollary 4.1.2), and these conditions correspond to Case-C and Case-D of Figure 4.7 respectively (switch, $S^V_{i,j+1}$ is shown in ST^V_2; when it is in ST^V_5 or ST^V_7, the changes would be the same).

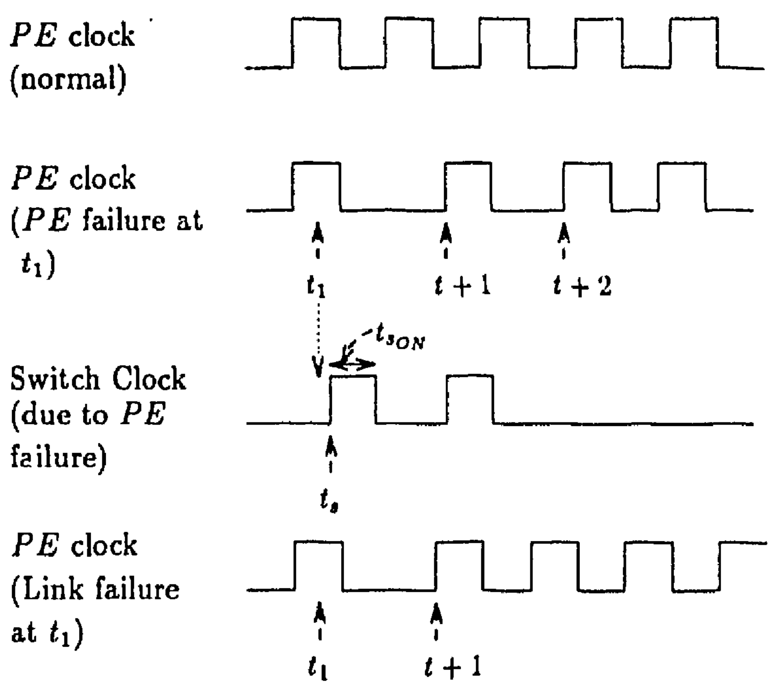
When $S^V_{i,j1}$ is in ST^V_1, $S^V_{i-1,j1}$ would be in ST^V_0 (Corollary 4.1.3) and it corresponds to Case-E of Figure 4.7.

When $S^V_{i,j1}$ is in ST^V_4 or in ST^V_5, $S^V_{i-1,j1}$ would be in ST^V_3 (Corollary 4.1.4) and it corresponds to Case-F of Figure 4.7. □

In the next section, a scheme for implementing this algorithm is proposed.

4.4 Implementation

A scheme is proposed here to implement the above algorithm for combined PE and link failure handling. The proposed scheme uses an external clock (CLK_S) for the switch state changes in the case of PE failures. Reconfiguration in the case of a link failure does not need any external clock, but when there is a PE failure at t_1 (see Figure 4.8), two clock pulses are provided to the switches and the next clock edge $t+1$ is delayed. At t_s, the first clock edge is applied to the switches to complete the intermediate stage of reconfiguration. The on-time of t_s depends on the time taken by the PEs to check their inputs. If input checking time is t_c, then the on-time of t_s, $t_{s_{on}} = t_c + \delta t$, where δt depends on $RR-$ propagation time and switch settling time. This is done to ensure the proper routing of vertical data in the case of a vertical link failure detection between intermediate and final stages of



PE clock (normal)
PE clock (PE failure at t_1)
t_1
$t+1$
$t+2$
t_{sON}
Switch Clock (due to PE failure)
t_s
PE clock (Link failure at t_1)
t_1
$t+1$

Figure 4.8: Various Clock Signals

PE failure reconfiguration (due to an earlier link failure). At $t+1$, the second clock edge arrives to the switches and completes the final stage of the reconfiguration. The next pulse $t+2$ to the PEs is also delayed to accommodate the switch settling time.

In the case of a link failure at t_1, the next clock edge, $t+1$ to the PEs is delayed and no separate clock is given to the switches. Various changes required in the network, processing element and switch module are given in the following subsections.

4.4.1 Network

The network is made capable of:

- informing the central processor of the occurrence of PE and link failure;
- informing the central processor of the occurrence of fatal failure;
- invoking the reconfiguration,

- providing the clock pulses to the switches and
- initiating the switches.

The central processor provides horizontal and vertical data inputs, PE-clock (CLK_{PE}) and switch-clock (CLK_S) to the array and it receives reconfiguration requests and output from the array. It provides a signal called SPE (Spare PE) to the PEs of the bottom-most row. This signal brings the switches $S^V_{m,j}$ and $S^V_{m+1,j}$ $(0 \leq j \leq n)$ to states ST^V_1 and ST^V_2 initially.

The network is shown in Figure 4.9.

4.4.2 Processing Element

Various control and data lines for a processing element are shown in Figure 4.10. Each PE gets four horizontal inputs ($I^H_{PE0}, I^H_{PE1}, I^H_{PE2}$ and I^H_{PE3}) and two vertical inputs (I^V_{PE0} and I^V_{PE1}). Similarly each PE has four horizontal output ports ($O^H_{PE0}, O^H_{PE1}, O^H_{PE2}$ and O^H_{PE3}) carrying the same data and two vertical output ports (O^V_{PE0} and O^V_{PE1}) carrying the same information.

Each $PE_{i,j}$ gets two control signals, $RR^{PE_{i,j}}_{PE_{i-1,j}}$ and $SVI^{PE_{i,j}}_{S_{i,j+1}}$. $RR^{PE_{i,j}}_{PE_{i-1,j}}$ is the reconfiguration request from $PE_{i-1,j}$ and $SVI^{PE_{i,j}}_{S_{i,j+1}}$ is the command for selecting the proper vertical input port. There is no such SVI control input for horizontal input selection because the horizontal input port selection is done by the PE itself.

$PE_{i,j}$ issues various control signals (reconfiguration requests) to other switches and PEs. It generates $LF^{S_{i,j+1}}_{VPE_{i,j}}$ (link failure for vertical data) in the case of a detected vertical input data error and it is sent to $S_{i,j+1}$. Another signal $LF^{CP}_{PE_{i,j}}$ is sent to the central processor in the case of an input error (vertical or horizontal). The central processor delays the next clock edge, $t+1$ to the PEs after receiving this signal. This delay time depends on the time required for RR propagation and switch settling time. If $PE_{i,j}$ detects a self fault, it sends $RR^{S_{i,j+1}}_{PE_{i,j}}$ to $S_{i,j+1}$, $RR^{S_{i+1,j+1}}_{PE_{i,j}}$ to $S_{i+1,j+1}$, $RR^{S_{i+1,j}}_{PE_{i,j}}$ to $S_{i+1,j}$ and $RR^{PE_{i+1,j}}_{PE_{i,j}}$ to $PE_{i+1,j}$.

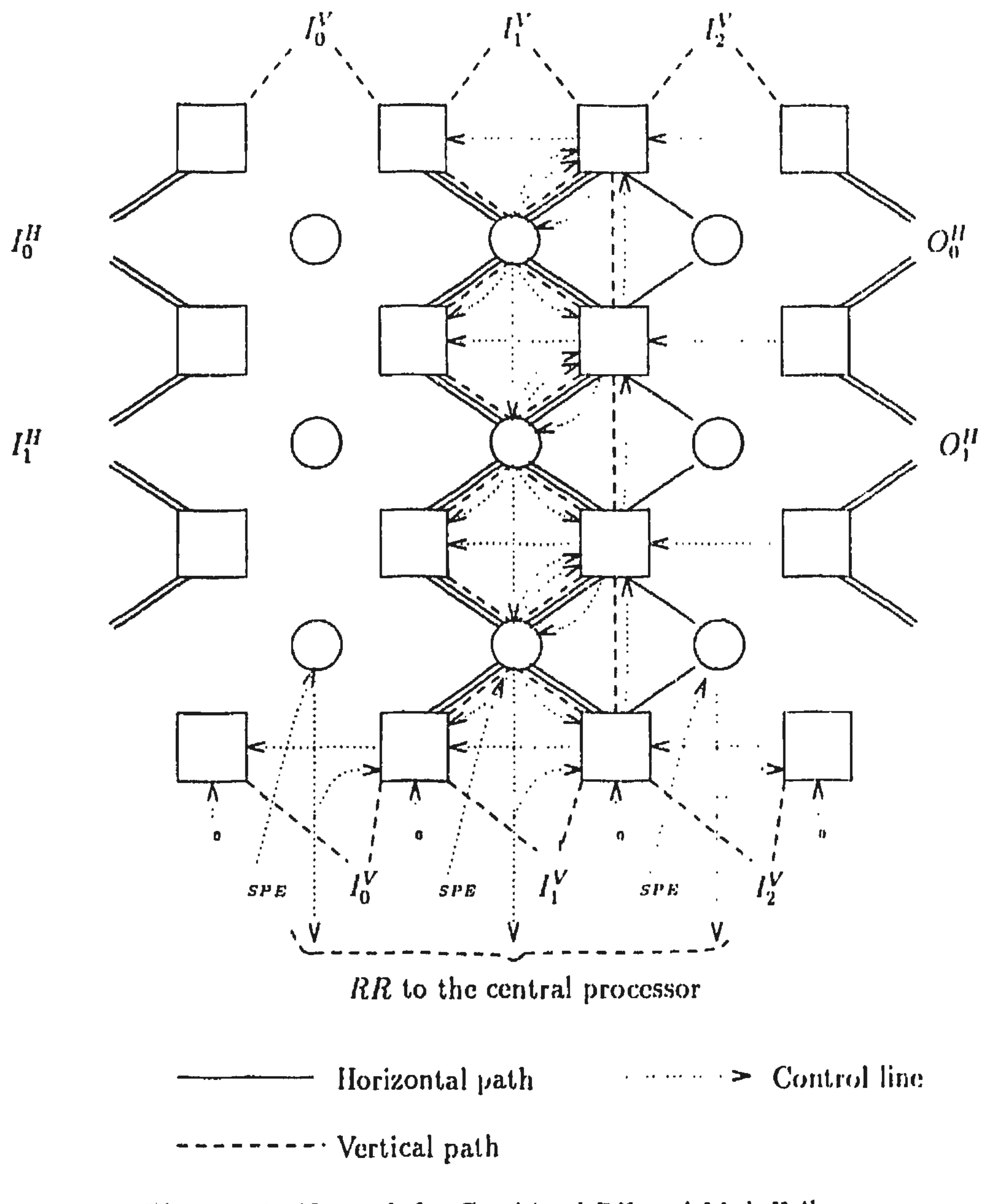


Figure 4.9: Network for Combined PE and Link Failure

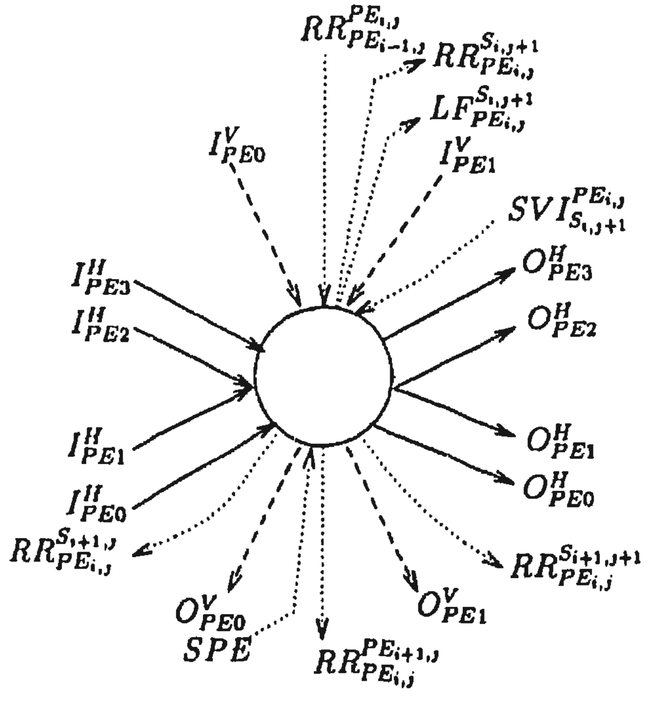

$RR^{PE_{i,j}}_{PE_{i-1,j}}$
$RR^{S_{i,j+1}}_{PE_{i,j}}$
$LF^{S_{i,j+1}}_{PE_{i,j}}$
I^V_{PE0}
I^V_{PE1}
$SVI^{PE_{i,j}}_{S_{i,j+1}}$
I^H_{PE3}
O^H_{PE3}
I^H_{PE2}
O^H_{PE2}
I^H_{PE1}
O^H_{PE1}
I^H_{PE0}
O^H_{PE0}
$RR^{S_{i+1,j}}_{PE_{i,j}}$
$RR^{S_{i+1,j+1}}_{PE_{i,j}}$
O^V_{PE0}
O^V_{PE1}
SPE
$RR^{PE_{i+1,j}}_{PE_{i,j}}$

Figure 4.10: Processing Element Lines for Combined PE and Link Failure

The block diagram of $PE_{i,j}$ is given in Figure 4.11. I^H_{PE0} and I^H_{PE1} inputs to $PE_{i,j}$ come from $S_{i+1,j}$ and I^H_{PE2} and I^H_{PE3} come from $S_{i,j}$ (see Figure 4.9) and depending on the earlier reconfiguration either one of the pairs (I^H_{PE0}, I^H_{PE1} or I^H_{PE2}, I^H_{PE3}) is selected by using $MUX - C$. Initially, $PE_{i,j}$ receives horizontal data from $S^H_{i+1,j}$ using I^H_{PE0} input port and when it detects a horizontal input error, its E_{IH} (error in horizontal input) line becomes high and $PE_{i,j}$ selects I^H_{PE1} in place of I^H_{PE0}. Now, if $PE_{i,j}$ receives $RR^{PE_{i,j}}_{PE_{i-1,j}}$, the H_S line is reset and H'_S is made high, which selects port I^H_{PE2} for horizontal input.

Once either I^H_{PE1} or I^H_{PE3} is selected, E_{IH} becomes 0 because the new data are correct (it is assumed that only one failure can occur at a time), but H_S remains high. Next, if the same PE detects another horizontal input error, it again makes E_{IH} high and the RR generating circuit (shown in Figure 4.12) generates a *fatal failure* signal.

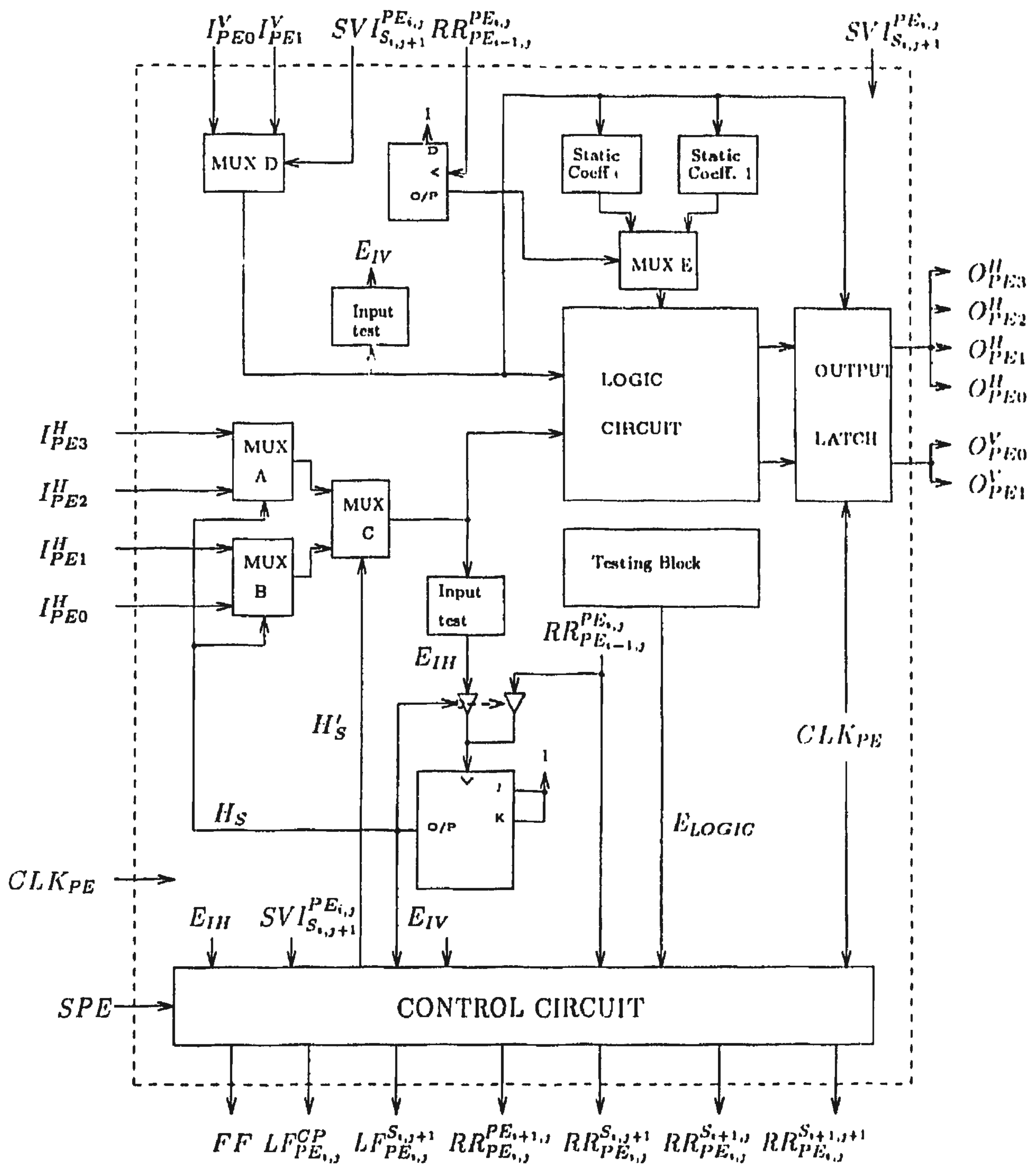


Figure 4.11: Block Diagram of the Processing Element (combined PE and link failure algorithm)

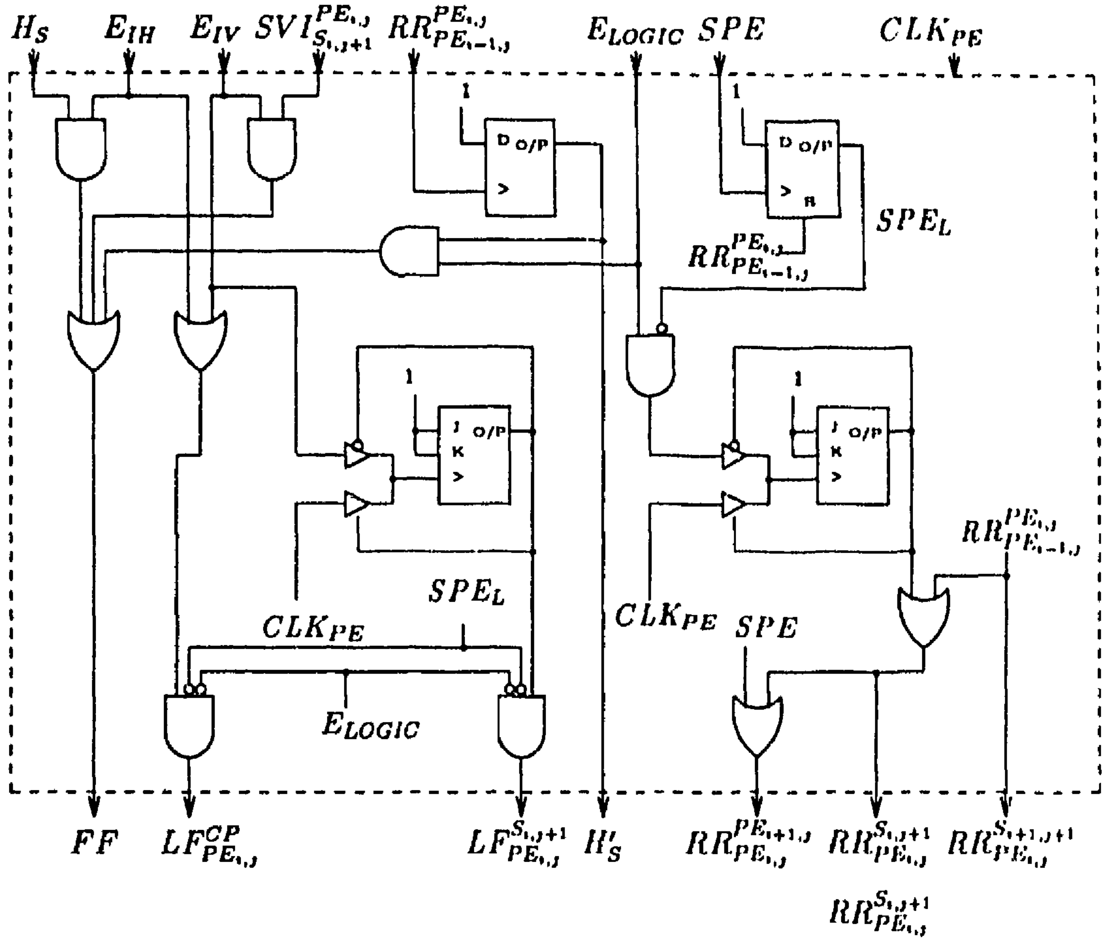
I^V_{PE0} and I^V_{PE1} come to $MUX - D$ and if vertical input error is detected, E_{IV} becomes high and it is passed on to $S_{i,j+1}$ as $LF^{S_{i,j+1}}_{PE_{i,j}}$, which changes the switch states, if required. As explained previously, a PE failure may also require other PEs to select their other vertical input port. This is done by the switch $S_{i,j+1}$. $S_{i,j+1}$ generates $SVI^{PE_{i,j}}_{S_{i,j+1}}$, which is used by the PE to select the proper vertical input port (when $SVI^{PE_{i,j}}_{S_{i,j+1}}$ is low, I^V_{PE0} is selected and a high SVI selects I^V_{PE1}) . If a vertical input error is detected while SVI is high, *fatal failure* occurs.

When the testing block of a PE detects a fault in $PE's$ logic circuit, it generates an error signal, E_{LOGIC} (this signal remains valid until the array is taken off-line) which is used to generate the RRs.

After loading the coefficients, the central processor sends a signal, SPE (spare PE) to the PEs of the bottom most row (to spare cells), which is latched as SPE_L. SPE makes $RR^{S_{i,j+1}}_{PE_{i,j}}$ and $RR^{CP}_{PE_{i,j}}$ lines high for some time, so that CLK_S brings $S^V_{m,j}$ and $S^V_{m+1,j}$ $(0 \leq j < n)$ to ST^V_1 and ST^V_2 respectively. SPE_L is used to ensure that no RRs are generated by a spare cell, when it detects a self-fault or input error.

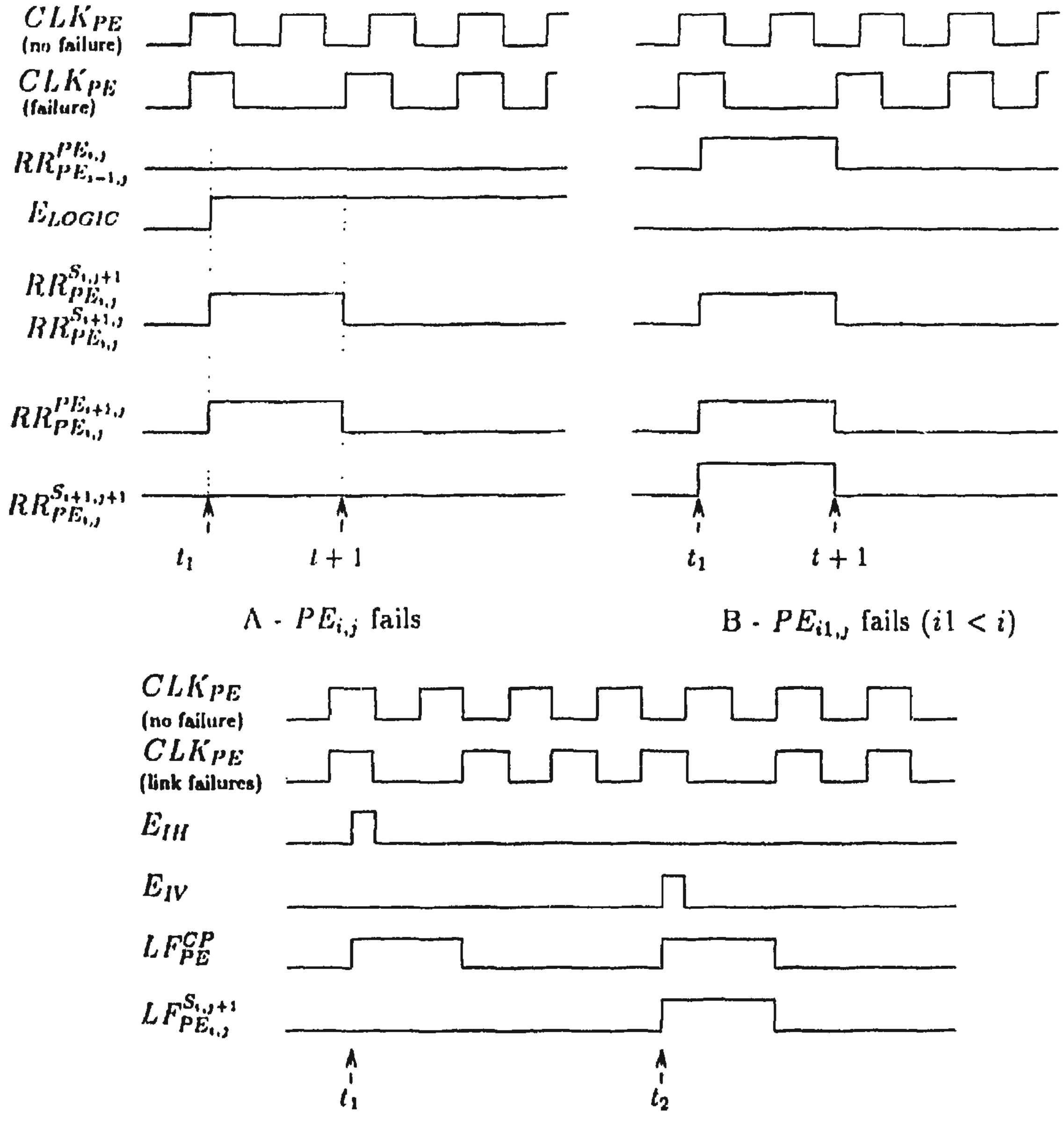
The RR- generating circuit is shown in Figure 4.12. Various inputs and outputs are shown in the block diagram and the timing diagram of the output control signals is shown in Figure 4.13. Figure 4.13.a shows the RRs generated by $PE_{i,j}$ when it fails at time t_1 and Figure4.13.b shows the RRs generated by $PE_{i,j}$ when it receives $RR^{PE_{i,j}}_{PE_{i-1,j}}$ at t_1 due to failure of $PE_{i1,j}$ $(i1 < i)$. When a faulty $PE_{i,j}$ receives $RR^{PE_{i,j}}_{PE_{i-1,j}}$, it generates *fatal failure* signal.

Figure 4.13.c shows CLK_{PE}, $LF^{CP}_{PE_{i,j}}$ and $LF^{S_{i,j+1}}_{PE_{i,j}}$ in the case of link failures. At t_1, a horizontal data error is detected and the central processor is informed but no information is sent to $S_{i,j+1}$. At t_2, a vertical data error is detected and both the central processor and $S_{i,j+1}$ are informed about this failure. All RRs are reset at time $t + 1$.



H_S
E_{IH}
E_{IV}
$SVI^{PE_{i,j}}_{S_{i,j+1}}$
$RR^{PE_{i,j}}_{PE_{i-1,j}}$
E_{LOGIC}
SPE
CLK_{PE}
SPE_L
$RR^{PE_{i,j}}_{PE_{i-1,j}}$
CLK_{PE}
SPE_L
E_{LOGIC}
CLK_{PE}
SPE
$RR^{PE_{i,j}}_{PE_{i-1,j}}$
FF
$LF^{CP}_{PE_{i,j}}$
$LF^{S_{i,j+1}}_{PE_{i,j}}$
H'_S
$RR^{PE_{i+1,j}}_{PE_{i,j}}$
$RR^{S_{i,j+1}}_{PE_{i,j}}$
$RR^{S_{i+1,j+1}}_{PE_{i,j}}$
$RR^{S_{i,j+1}}_{PE_{i,j}}$

Figure 4.12: Schematic of the RR generating Circuit (combined PE and link failure algorithm)


CLK_{PE} (no failure)
CLK_{PE} (failure)
$RR^{PE_{i,j}}_{PE_{i-1,j}}$
E_{LOGIC}
$RR^{S_{i,j+1}}_{PE_{i,j}}$
$RR^{S_{i+1,j}}_{PE_{i,j}}$
$RR^{PE_{i+1,j}}_{PE_{i,j}}$
$RR^{S_{i+1,j+1}}_{PE_{i,j}}$
t_1
$t+1$
CLK_{PE} (no failure)
CLK_{PE} (link failures)
E_{IH}
E_{IV}
LF^{CP}_{PE}
$LF^{S_{i,j+1}}_{PE_{i,j}}$
t_1
t_2

A - $PE_{i,j}$ fails

B - $PE_{i1,j}$ fails ($i1 < i$)

C - Link Failures at t_1 and t_2

Figure 4.13: Timing Diagram of the RR generating Circuit

4.4.3 Switch Module

As explained earlier, each switch module, $S_{i,j}$ is a pair of switches, $S^H_{i,j}$ and $S^V_{i,j}$, which are used to route the horizontal and vertical data respectively. Various data and control lines for a switch module $S_{i,j}$ are shown in Figure 4.14. It gets $LF^{S_{i,j}}_{PE_{i,j-1}}$ and $RR^{S_{i,j}}_{iPE_{i,j-1}}$ from $PE_{i,j-1}$, $RR^{S_{i,j}}_{PE_{i-1,j-1}}$ from $PE_{i-1,j-1}$, $RR^{S_{i,j}}_{PE_{i-1,j}}$ from $PE_{i-1,j}$, $RR^{S_{i,j}}_{S_{i+1,j}}$ from $S_{i+1,j}$ and $RR^{S_{i,j}}_{S_{i,j+1}}$ from $S_{i,j+1}$ as control inputs and based on these data, it changes the switch state and generates $RR^{S_{i-1,j}}_{S_{i,j}}$, $RR^{S_{i,0}}_{S_{i,j}}$ and $SVI^{PE_{i,j-1}}_{S_{i,j}}$.

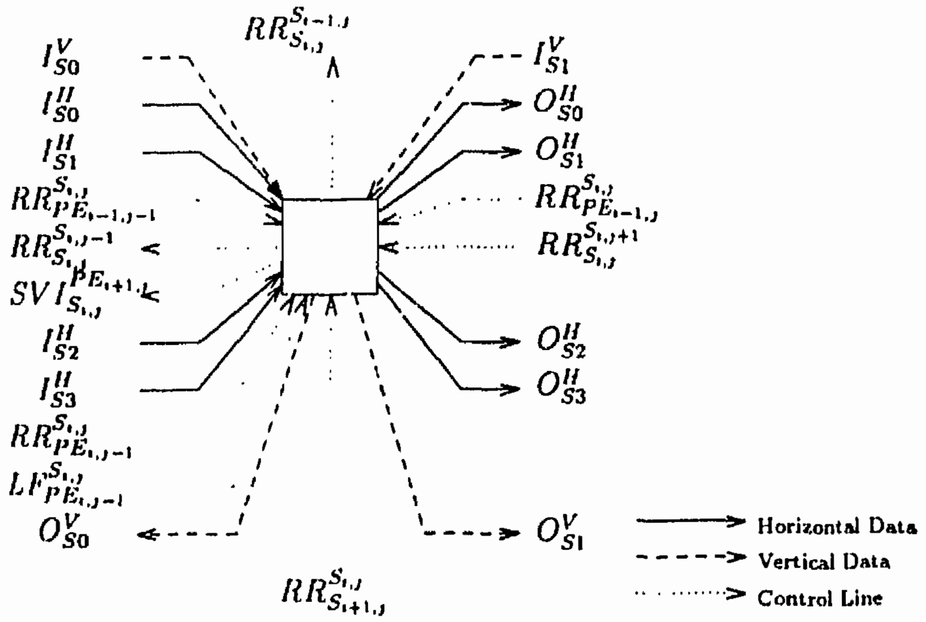
If $PE_{i,j}$ fails at t_1 (see Figure 4.14), the central processor provides two clock pulses to the switches by using a global switch clock line at t_s and $t+1$. At t_s, the intermediate stage of the rerouting is completed and at $t+1$ the final stage is completed.

The delay, $t_s - t_1$ depends on the switch settling time and the number of columns in the array, because the RRs go from $S_{i1,j+1}$ to $S_{i1,0}$ ($i1 > i$), if a switch $S^V_{i1,j+1}$ is in ST^V_4 (as explained in the algorithm). Similarity, $(t+1) - t_s$ depends on the PE processing time and switch settling time. The next clock edge to the PEs, $t+2$ is also delayed to allow for the switch settling time during the final stage of reconfiguration. After $t+2$, the clock resumes its normal speed.

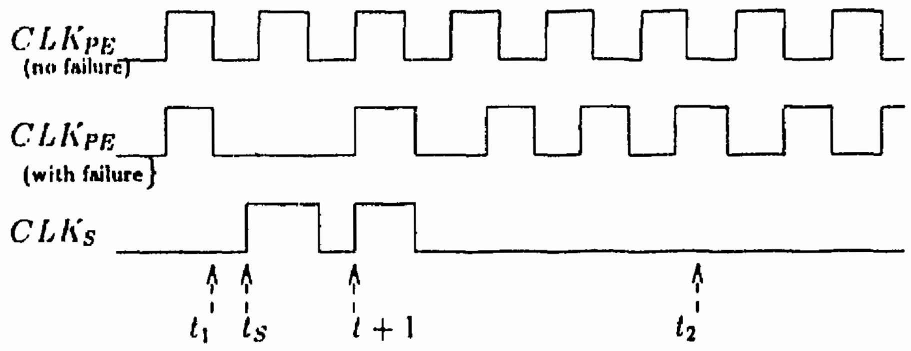
If a link fails at t_2, next clock edge, $t+1$ to the PEs is delayed by a pre-specified time to provide sufficient time for RR propagation and switch settling and after $t+1$, the clock resumes its normal speed.

Vertical and horizontal switches are discussed separately.

Vertical switch - It has two sub-circuits: the control circuit and the switching circuit. The control circuit changes the states of the switches and generates various RRs, while the switching circuit provides the proper input-output connections based on the state-data made available by the control circuit. Various switch state changes are described in the algorithm and to achieve the proper changes, the block diagram shown in Figure 4.15 is proposed for the control circuit. For the sake of


$RR_{S_{i,j}}^{S_{i-1,j}}$
I_{S0}^V
I_{S0}^H
I_{S1}^H
$RR_{PE_{i-1,j-1}}^{S_{i,j}}$
$RR_{S_{i,j}}^{S_{i,j-1}}$
$SVI_{S_{i,j}}^{PE_{i+1,j}}$
I_{S2}^H
I_{S3}^H
$RR_{PE_{i,j-1}}^{S_{i,j}}$
$LF_{PE_{i,j-1}}^{S_{i,j}}$
O_{S0}^V
I_{S1}^V
O_{S0}^H
O_{S1}^H
$RR_{PE_{i-1,j}}^{S_{i,j}}$
$RR_{S_{i,j}}^{S_{i,j+1}}$
O_{S2}^H
O_{S3}^H
O_{S1}^V
$RR_{S_{i+1,j}}^{S_{i,j}}$
Horizontal Data
Vertical Data
Control Line

Controls and Data lines for a Switch



Switch Clock due to PE and Link Failure

Figure 4.14: Data and Control Lines for a Switch (combined PE and link failure algorithm)

clarity, various signals are renamed as shown below:

$$\begin{array}{llll} A_I = & RR^{S_{i,j}}_{S_{i,j+1}} & A_O = & RR^{S_{i,j-1}}_{S_{i,j}} \\ B_I = & RR^{S_{i,j}}_{S_{i+1,j}} & B_O = & RR^{S_{i-1,j}}_{S_{i,j}} \\ C = & RR^{S_{i,j}}_{PE_{i-1,j-1}} & D = & RR^{S_{i,j}}_{PE_{i,j-1}} \\ E = & LF^{S_{i,j}}_{PE_{i,j-1}} & X = & SVI^{PE_{i,j-1}}_{S_{i,j}} \end{array}$$

Once a $PE_{i,j}$ fails at t_1, various RRs are generated. If a link fails, E arrives and it is latched as E_L, which generates an A_O signal, if $S^V_{i,j}$ is in ST^V_2 or in ST^V_7. E_L is reset at the next falling edge of the switch clock (CLK_S) for the switch $S_{i,j}$, if $S^V_{i,j}$ is in the intermediate stage of rerouting due to $PE_{i-1,j-1}$ failure. Otherwise it gets reset at $t+1$, when E goes low.

A_I appears at $S_{i,j}$, if $S_{i,j+1}$ needs the state change of $S_{i,j}$. In the block diagram, $s_2 s_1 s_0$ inform the present state of the vertical switch.

In the case of $PE_{i,j}$ failure, the changes required by the reconfiguration algorithm depend on the index of the switch. These changes are listed in Table 4.1.

X represents that the switch cannot be in this state. No switch $S^V_{i,j+1}$ can be in states ST^V_6 or ST^V_7 without an earlier PE failure in column j, which causes fatal failure now. Similarly, $S^V_{i,j+1}$ cannot be in ST_1 or ST^V_2 (Lemma 4.3), because if there is no previously failed PE in column j, only $S^V_{m,j+1}$ would be in ST^V_1 and only $S^V_{m+1,j+1}$ would be in ST^V_2. In this case, only the failure of $PE_{m,j}$ would find $S_{m,j+1}$ in state ST^V_1 and since $PE_{m,j}$ is the spare cell, no reconfiguration is invoked. Similarly, $S_{i+1,j+1}$ cannot be in ST^V_2 (Corollary 4.3.1).

When $S^V_{i1,j+1}$ ($i1 > i$) (which is either in ST^V_4 or in ST^V_5) receives D, it generates A_O. After receiving A_I, $S^V_{i1,j}$ changes state either to ST^V_4 from ST^V_0 & ST^V_2 or to ST^V_5 from ST_1 and it issues B_O to $S^V_{i1-1,j}$, which causes $S^V_{i1-1,j}$ to change state either to ST^V_3 from ST^V_0 & ST^V_1 or to ST^V_6 from ST^V_2. If $S^V_{i1,j}$ is in ST^V_2, it again generates A_O and RR propagates towards $S_{i1,0}$ in this manner until a switch, $S^V_{i1,j1}$

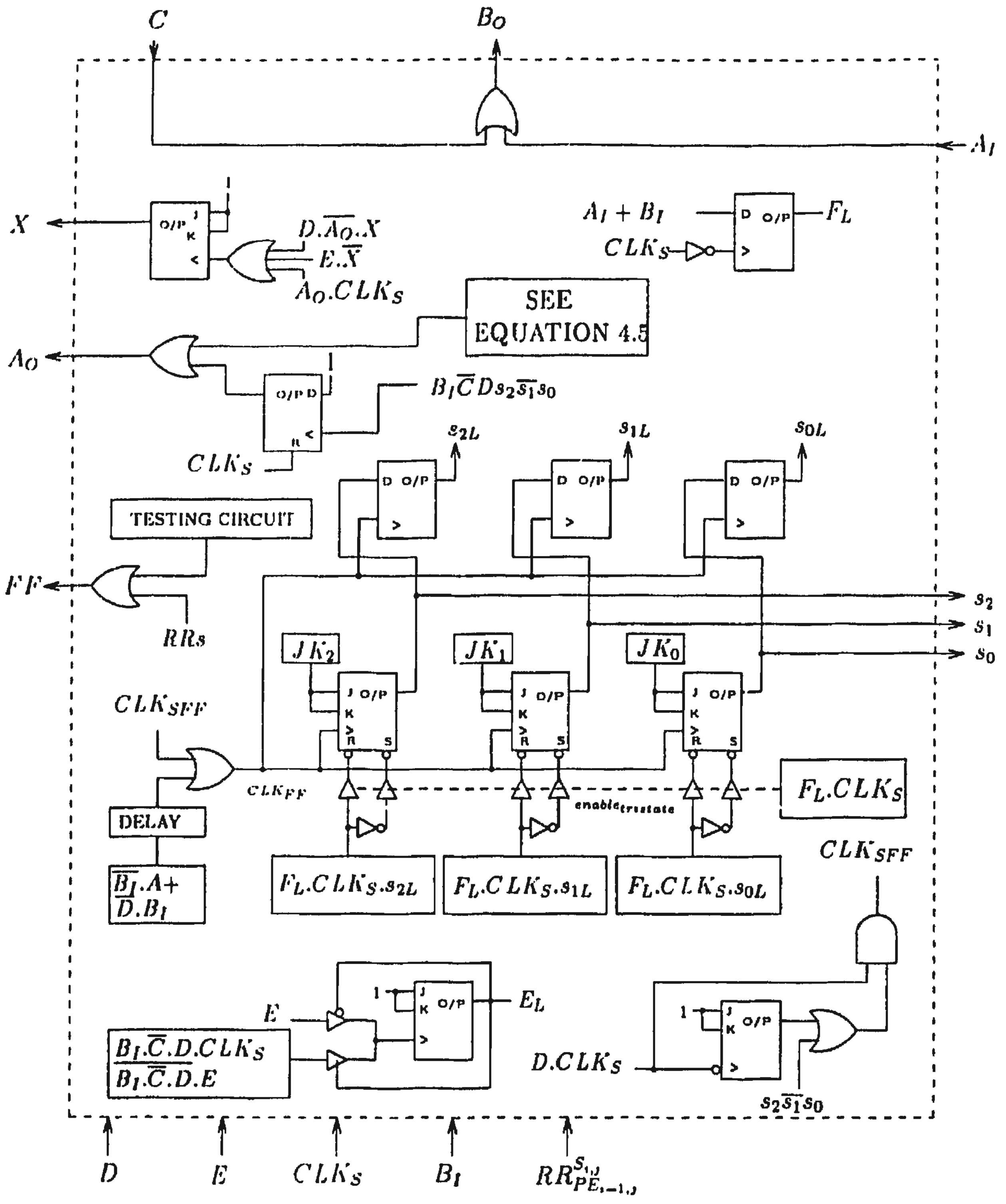


Figure 4.15: Block Diagram of the Vertical Switch (combined PE and link failure algorithm)

is found, which is not in ST_2^V. $S_{i1,j1}$ does not generate A_O and at the same time $S_{i1,y}$ $(j1 < y \leq j+1)$ instruct $PE_{i1,y-1}$ to use the second vertical input port by setting SVI signals. A_O and B_O change the state of the switches without using the external clock. This is the reason behind delaying t_s after t_1, so that all the switches, which need to generate A_O and B_O can generate these signals and A_I and B_I get sufficient time to change the state of the switches (because at t_s, the switch $S_{i1,j+1}^V$ changes state and A_O generated by this may not stay after that).

A_I and B_I are latched as F_L at the falling edge of CLK_S and F_L is used to bring the switch in its prior-to-t_1-state, if the final stage requires so. If these newly generated paths (generated by A_I and B_I) are required by the final stage of reconfiguration (when $i_1 = i+1$), $S_{i1,j+1}^V$ resets A_O at t_s and F_L remains low.

A_I and B_I change the states of the switches according to Table 4.2. The next clock edge $t+1$, to the switches brings the changes, listed in Table 4.3 (the states of the switches before $t+1$ is taken from Table 4.1).

At $t+1$, switches having $F_L = 1$ go back to their prior-to-t_s-state. At $(t+1)^+$ the reconfiguration is complete and all the RRs are reset.

In the case of a link failure communicated by $PE_{i,j}$ to $S_{i,j+1}$, A_O is generated by $S_{i,j+1}^V$ if it is in ST_2^V, ST_6^V or ST_7^V. This A_O propagates towards $S_{i,0}$ and generates B_O as explained earlier, until it finds a switch $S_{i,j1}^V$, which is not in ST_2^V. The changes caused by A_I and B_I are listed in Table 4.2.

Since the changes depend on the physical index of the switch in the case of a $PE_{i,j}$ failure, B_I, C and D are used to decode the position of the switches. For $S_{i,j+1}$, B_I and C are low and D is high (i.e. $\overline{B_I}\ \overline{C}\ D$ is high). For $S_{i+1,j+1}$, $B_I\ \overline{C}\ D$ is high and for $S_{i1,j+1}$ $(i1 > i+1)$, $B_I\ C\ D$ is high.

Three $J-K$ flip flops (s_2, s_1 and s_0) are used to store the current state of the switch and 3 D-latches (s_{2L}, s_{1L} and s_{0L}) are used to store the previous state of the switches.

Table 4.1: State changes for Intermediate Stage

State before t_1	State after t_s		
	$S_{i,j+1}$	$S_{i+1,j+1}$	$S_{i1,j+1}$ $(i_1 > i+1)$
0	1	7	7
1	X	7	7
2	X	X	7
3	3	6	6
4	5	5	5
5	X	5	5
6	X	X	X
7	X	X	X

Table 4.2: State changes due to A_I and B_I

State before the arrival of A_I and B_I	State after the arrival of A_I and B_I	
	Due to A_I	Due to B_I
0	4	3
1	5	3
2	4	6
3	X	3
4	4	X
5	5	X
6	X	6
7	X	X

Table 4.3: State changes for Final Stage

State before $t+1$	State after $t+1$		
	$S_{i,j+1}$	$S_{i+1,j+1}$	$S_{i1,j+1}$ $(i_1 > i+1)$
0	X	X	X
1	1	X	X
2	X	X	X
3	3	X	X
4	X	X	X
5	5	4	4
6	X	6	3
7	X	2	0

For bringing the switches to their prior-to-t_s-state, *reset* and *set* inputs of the flip-flops are used. For all the other changes the switches are clocked into their new states. Since the external switch-clock, CLK_S comes only in the case of a $PE_{i,j}$ failure and is supposed to modify the switches of column $(j+1)$ only, it is AND-ed with D. A_I and B_I work as clock for all the switches $S_{i1,j1}$ $(j1 \neq j+1)$. So, $\overline{D} \cdot (A_I + B_I)$ is delayed and OR-ed with $D \cdot CLK_S$ to get the final clock to the switches. $\overline{D} \cdot (A_I + B_I)$ is delayed to ensure the presence of proper input at flip-flops' inputs before the clock edge, CLK_{FF} appears.

If the first rising edge of CLK_S is blocked from reaching the $J-K$ flip flops for the switch, $S^V_{m,j+1}$ if it is in ST^V_5, then various state changes (at the clock edge) can be listed as in Table 4.4 (for generating this table, tables 4.1, 4.2 and 4.3 are combined). When $S^V_{m,j+1}$ is in ST^V_5, the first edge of CLK_S is blocked by using a $J-K$ flip flop and two gates. This circuit allows only the second rising edge of CLK_S to appear as CLK_{SFF}, if the switch is in ST^V_5.

At $t+1$, F_L brings the switches to their prior-to-t_s-state using *set-reset* inputs of the flip flops. F_L uses the outputs of D-latches, s_{2L}, s_{1L} and s_{0L} for bringing the switch to prior-to-t_s-state. For the JK-flip flops, *set-reset* inputs are listed in Equation 4.1.

$$\begin{aligned} \overline{R_2} &= F_L \cdot CLK_S \cdot s_{2L}, & \overline{S_2} &= R_2, \\ \overline{R_1} &= F_L \cdot CLK_S \cdot s_{1L}, & \overline{S_1} &= R_1, \\ \overline{R_0} &= F_L \cdot CLK_S \cdot s_{0L} \text{ and} & \overline{S_0} &= R_0. \end{aligned} \tag{4.1}$$

The *set-reset* inputs go through tri-states to the flip flops and the tri-states are activated only when these changes are required.

$$enable_{tristate} = F_L \cdot CLK_S. \tag{4.2}$$

Similarly, $J-K$ inputs are derived and written in Equation 4.3.

Table 4.4: State changes due to clock edge

State before the clock edge	State after the clock edge				
	$\overline{B_I}\,\overline{C}\,D$	$B_I\,\overline{C}\,D$	$B_I\,C\,D$	$\overline{D}\,A_I$	$\overline{D}\,B_I$
0	1	7	7	4	3
1	1	7	7	5	3
2	X	X	7	4	6
3	3	6	6	X	3
4	5	5	5	4	X
5	5	4	4	5	X
6	X	6	3	X	6
7	X	2	0	X	X

Table 4.5: State changes using *set-reset* inputs

State	State after the change		
	$F_L.CLK_S$	$A_I.\overline{B_I}.s_2.s_1.s_0$	$\overline{A_I}.B_I.s_2.s_1.s_0$
0		X	X
1	*goes*	X	X
2	*to*	X	X
3		X	X
4	*prior-to*	X	X
5	t_s	X	X
6	*state*	X	X
7		5	6

Table 4.6: Generation of A_O

State	$\overline{B_I}\,\overline{C}D$	$B_I\overline{C}D$	B_ICD	$A_I\overline{D}$	E_L
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	1	0	0
5	0	*	1	0	0
6	0	0	0	0	0
7	0	0	0	0	1

$$
\begin{aligned}
J_2 = K_2 = JK_2 = &\ B_I\overline{C}D(\overline{s_2} + s_2s_1s_0) + B_ICD(\overline{s_2} + s_1) + \overline{D}A_I(\overline{s_2}) + \overline{D}B_I(\overline{s_2}s_1\overline{s_0}) \\
J_1 = K_1 = JK_1 = &\ B_I\overline{C}D(\overline{s_2}\ \overline{s_1}) + B_ICD(\overline{s_2}\ \overline{s_1} + s_2s_1s_0) + \overline{D}A_I(s_1) + \overline{D}B_I(\overline{s_1}) \\
J_0 = K_0 = JK_0 = &\ \overline{B_I}\ \overline{C}D(\overline{s_1}\ \overline{s_0}) + B_I\overline{C}D(\overline{s_1}\ \overline{s_0} + \overline{s_2}s_1 + s_2s_0) + \\
&\ B_ICD(\overline{s_1}\ \overline{s_0} + s_1 + s_2) + \overline{D}B_I(\overline{s_1}\ \overline{s_0})
\end{aligned} \tag{4.3}
$$

B_O is generated, whenever C or A_I is there, therefore:

$$B_O = C + A_I. \tag{4.4}$$

A_O is generated depending on the switch state and various inputs. In the case of $PE_{i,j}$ failure, A_O is generated by the switch $S_{i1,j+1}$ $(i1 < i)$, which is either in state ST_4^V or in ST_5^V. When a switch, which is in ST_2^V or ST_6^V, receives A_I, it generates A_O. Any A_O generated by $S_{i1,j+1}$ $(i1 > i + 1)$, stays high till $t + 1$, so that F_L stays high for bringing various switches to their prior-to-t_s-state, if required. Various combinations for generating A_O are listed in Table 4.6. The entry for $B_I \cdot C \cdot D$, corresponding to state 5 is marked as '*', because this condition generates A_O, which stays high only until the rising edge of CLK_S arrives (even though the switch remains in ST_5^V).

Consequently,

$$
\begin{aligned}
A_O = &\ A_{O1} + A_{O2}, \\
A_{O1} = &\ B_I\overline{C}D(s_2\overline{s_1}s_0) \text{ - high from } t_1 \text{ to } t_s \text{ and} \\
A_{O2} = &\ B_I\overline{C}D(s_2\overline{s_1}\ \overline{s_0}) + B_ICD(s_2\overline{s_1}) + A_I(\overline{s_2}s_1\overline{s_0}) + E_L(\overline{s_2}s_1\overline{s_0} + s_2s_1s_0).
\end{aligned} \tag{4.5}
$$

$SVI_{S_{i,j}}^{PE_{i,j-1}}$ is generated by $S_{i,j}^V$ and it is used by $PE_{i,j-1}$ for selecting the proper vertical input port. If $PE_{i,j-1}$ is using input port I_{PE0}^V and input error is detected, $SVI_{S_{i,j}}^{PE_{i,j-1}}$ is made high and I_{PE1}^V is selected. Now, if $S_{i,j}^V$ receives D, it resets $SVI_{S_{i,j}}^{PE_{i,j-1}}$ and again I_{PE0}^V is selected. When $S_{i,j}^V$ is in ST_2^V or in ST_7^V and a link

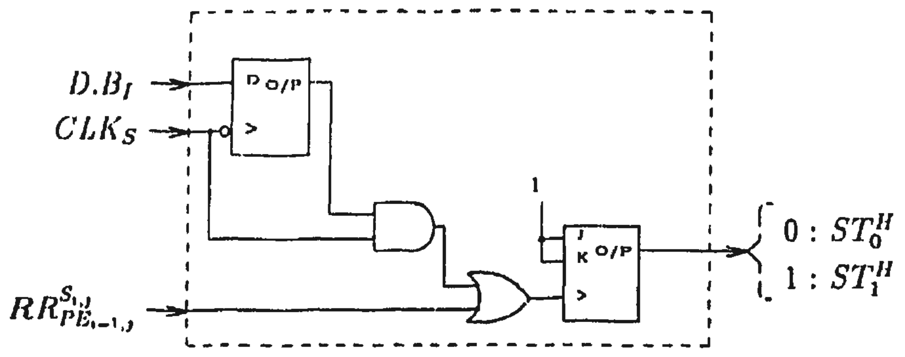


Figure 4.16: Block Diagram of the Horizontal Switch (combined PE and link failure algorithm)

failure signal arrives, A_O is generated, which moves towards $S_{i,0}$, until it finds $S^V_{i,j1}$, which is not in ST^V_2. In this case, if $SVI^{PE_{i,j1}}_{S_{i,j1+1}}$ is high, it means that $PE_{i,j1}$ is using I^V_{PE1} due to earlier link failure. In this case, during the final rerouting, $SVI^{PE_{i,j1}}_{S_{i,j1+1}}$ is not reset.

Horizontal Switch - This also has two sub-circuits: control circuit and switching circuit. The block diagram of the control circuit is given in Figure 4.16.

When $S^H_{i,j}$ receives $RR^{S_{i,j}}_{PE_{i-1,j}}$, it toggles either from ST^H_0 to ST^H_1 or from ST^H_1 to ST^H_0. Similarly, when $D \cdot B_I$ is '1', the $t+1$ edge of CLK_S toggles it from one state to the other. For this, $D \cdot B_I$ is latched at the falling edge of CLK_S and the latched signal is $ANDed$ with CLK_S to generate the clock for the flip flop.

The testing block of the switches tests the logic circuit of the switches and the FF signal is generated, when a faulty switch receives any reconfiguration request. The operation of the algorithm is shown in the next section.

4.5 Operation of the Algorithm

Consider the array, shown in Figure 4.17, which has no faults at t_1, at which time $PE_{i,j}$ detects a vertical input error. Immediately, $PE_{i,j}$ generates $LF^{CP}_{PE_{i,j}}$ (which delays the next rising edge, $t+1$ of CLK_{PE}) and $LF^{S_{i,j+1}}_{PE_{i,j}}$. When $S_{i,j+1}$ receives

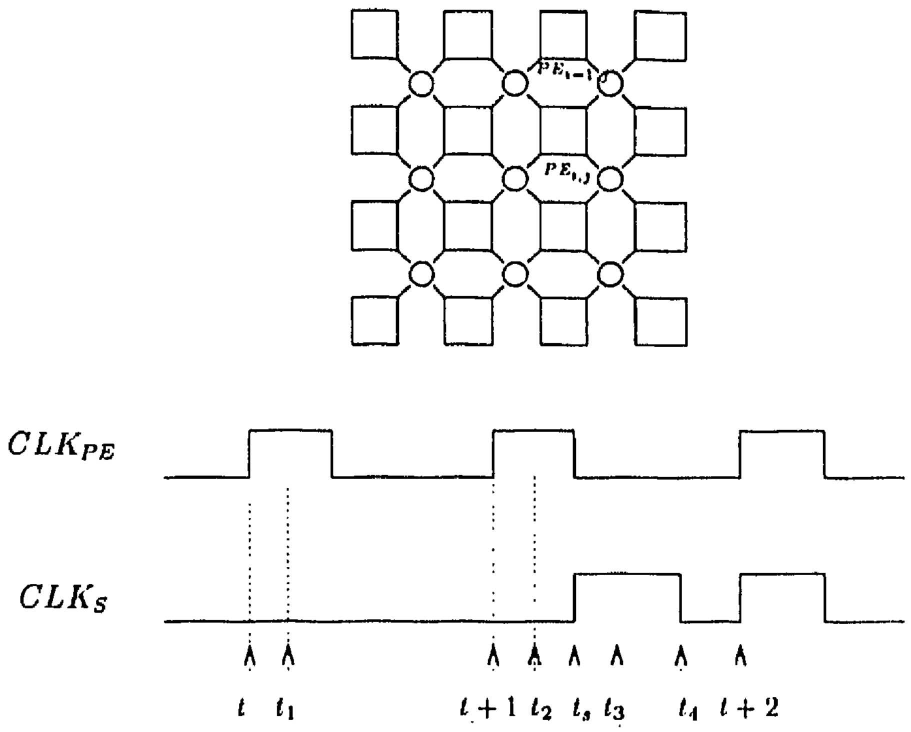


Figure 4.17: Operation of the Algorithm (combined PE and link failure algorithm)

E, it makes E_L and X high. X is fed back to $PE_{i,j}$, which selects I^V_{PE1} in place of I^V_{PE0} and it resets E, because the second link is non-faulty.

Let us assume that at t_2, $PE_{i-1,j}$ fails and various RRs are generated. $S_{i-1,j+1}$ receives D, $S_{i,j+1}$ receives B_l & D and $S_{i_x,j+1}$ $(i < i_x \le m+1)$ receive B_l, C & D. Once $S_{i,j+1}$ receives D, it resets X and $PE_{i,j}$ is forced to select I^V_{PE0} again. B_l, C & D change the inputs of $J-K$ flip flops and the new inputs are listed in Table 4.7.

At t_S, the positive edge of CLK_S arrives and the switches change their states depending on the $J-K$ inputs. These changes are listed below:

- $S^V_{i-1,j+1}$ goes to ST^V_1 and
- $S^V_{i_x,j+1}$ $(i \le i_x \le m+1)$ go to ST^V_7.

At t_2, $PE_{i_x,j}$ $(i-1 \le i_x \le m)$ generate $RR^{S_{i_x+1,j}}_{PE_{i_x,j}}$, which brings $S^H_{i_x,j}$ $(i \le i_x \le m+1)$ to ST^H_1 (required for the horizontal data intermediate stage routing). $RR^{PE_{i_x+1,j}}_{PE_{i_x,j}}$

Table 4.7: $J-K$ flip flop inputs at t_2

Flip-Flop Inputs	Inputs at t_2				
	$S^V_{i-1,j+1}$	$S^V_{i,j+1}$	$S^V_{i_x,j+1}$ $i < i_x < m$	$S^V_{m,j+1}$	$S^V_{m+1,j+1}$
JK_2	0	1	1	1	1
JK_1	0	1	1	1	0
JK_0	1	1	1	0	1

makes $PE_{i_x+1,j}$ select the other horizontal input port pair (here it selects "I^H_{PE2} and I^H_{PE3}").

Since the earlier vertical input error was detected by $PE_{i,j}$, it reappears again at t_3 (after the re-routed data are checked). There are two possibilities of its detection and they are written next (if the earlier line failure was due to switching circuit failure, it would not appear during the intermediate stage because the switch has changed state and a different path is in use).

Case A - If the first vertical input error was due to the failure of link $L^{S_{i,j+1}}_{PE_{i-1,j}}$, then $PE_{i+1,j}$ would now detect a vertical input error at t_3. In this case, at t_3, $PE_{i+1,j}$ generates $LF^{CP}_{PE_{i+1,j}}$ (which delays the next CLK_{PE} and CLK_S edges) and $LF^{S_{i+1,j+1}}_{PE_{i+1,j}}$. Once $S_{i+1,j+1}$ receives E, it latches it as E_L and generates X, which makes $PE_{i+1,j}$ select I^V_{PE1} in place of I^V_{PE0}. At the same time, $S_{i+1,j+1}$ generates A_O, which is passed on to $S_{i+1,j}$, which in turn generates B_O and feeds it to $S_{i,j}$. A_I and B_I change the inputs of $J-K$ flip flops of $S^V_{i+1,j}$ and $S^V_{i,j}$ respectively. The new inputs are listed in Table 4.8. These A_I and B_I appear as CLK_{FF} after a delay (which ensures the presence of proper information at the flip flop inputs) and it changes the states of $S^V_{i+1,j}$ and $S^V_{i,j}$ to ST^V_4 and ST^V_3 respectively. The previous states get latched in the D-flip flops. $(A_I + B_I)$ gets latched as F_L at the falling edge of CLK_S (at t_4). Here, $(A_I + B_I)$ would be *high* at t_4, because E_L gets reset at $t+2$ and A_O (of $S_{i+1,j+1}$) is *high* at t_4.

Table 4.8: $J-K$ flip flop inputs due to A_l and B_l

Flip-Flop Inputs	Inputs due to A_l and B_l	
	$S^V_{i+1,j}$	$S^V_{i,j}$
JK_2	1	0
JK_1	0	1
JK_0	0	1

Table 4.9: $J-K$ flip flop inputs at $t+2$

Flip-Flop	Inputs at $t+2$		
	$S^V_{i-1,j+1}$	$S^V_{i,j+1}$	$S^V_{i_x,j+1}$ $i < i_x \leq m+1$
JK_2	0	1	1
JK_1	0	0	1
JK_0	0	0	1

Case B - If the first vertical input error occurred due to the failed link $L^{PE_{i,j}}_{S_{i,j+1}}$, then $PE_{i,j}$ will report vertical input error again and the same changes occur, which are explained earlier in Case A, but here $A_l + B_l$ would not stay as F_L till $t+2$, because E_L gets reset at t_4 and it resets A_O (of $S_{i,j+1}$).

At t_S^+, the $J-K$ inputs of the flip flops change again due to a change in the switch state. The new inputs are listed in Table 4.9. At $t+2$, the next clock edge of CLK_S appears and it completes the final stage of reconfiguration by changing $S^V_{i,j+1}$ to ST^V_2 and $S^V_{i_x,j+1}$ $(i < i_x \leq m+1)$ to ST^V_0. At the same time, if the previously explained Case-A is valid, $S^V_{i+1,j}$ and $S^V_{i,j}$ are brought back to ST^V_0 (because F_L is high at $t+2$, which enables the *set-reset* tri-states and these asynchronous inputs of the flip flops load the previous states in these flip flops) and X (of $S_{i+1,j+1}$) is reset by $A_O \cdot CLK_S$. There is no change in the states of $S^V_{i+1,j}$ and $S^V_{i,j}$ for Case-B at $t+2$. Clock edge $t+2$ changes $S^H_{i_x,j+1}$ $(i-1 < i_x \leq m+1)$ to ST^H_1 now and it completes the total reconfiguration.

4.6 Concluding Remarks

In this chapter an on-line reconfiguration scheme for PE and link failures was discussed. Here an extra row of cells (called spares) is provided to the array and in the case of a detected PE failure global shift is performed for the corresponding column. The links are duplicated to provide link redundancy and link failures are detected by checking parity bits. The redundant vertical link is taken through different data path than the original link because in this configuration, the complete failure of a switch block will have lesser effect on the overall reliability. When a horizontal link fails, the PE automatically selects the other horizontal input port and when a vertical link fails, the PE informs the neighboring switch about this failure. The neighboring switch invokes the switch state changes and commands the PE to select the other vertical input port.

The control circuit for the PEs and switches were designed and the network was modified to support the algorithm. It was proved that the proposed eight states of the vertical switches and two states of the horizontal switches are sufficient to support the algorithm.

Here it is assumed that the link failures are detected by the PEs by using parity bit checks. The number of parity bits can be chosen depending upon the reliability requirement. With one parity bit only odd number of bit errors can be detected.

The algorithm is evaluated in the next chapter.

Chapter 5

Algorithm Evaluation

The reconfiguration algorithms are evaluated based on the following criteria [19]:

- *probability of survival* - defined as the probability of correct reconfigurations in the presence of x faults, $x \leq S$, where S is the number of spare cells in the array,
- *locality of interconnections*,
- *time complexity of reconfiguration algorithm* and
- *area complexity of the switching and routing circuits.*

These features are conflicting. It is possible to develop an algorithm which is simple and maintains high locality, but the probability of survival degrades in this case for an increasing number of faults.

The proposed algorithm maintains high locality by allowing only one downward shift in the case of a failure. The algorithm introduces very small time delays when a fault occurs. It is assumed that the central processor provides the clock pulses to the PEs and switches. When a PE fails, the central processor reduces the clock speed for next two clock periods. This can be achieved by simply blocking the *on-period* of the clock, when the delay is required. If this method is used, the failure of any PE would introduce a delay of 2 clock periods in the operation and a link failure would introduce a delay of 1 clock period in the operation.

The increase in the complexity of the switching circuit is not large for the *only PE failure algorithm*, but the switches and the network are slightly more complex for the *combined PE and link failure algorithm.*

The probability of survival is derived analytically first and then simulation results are presented.

5.1 Analytical Results

We consider a 4×4 active array, which needs a physical array of size 5×4. The array has 4 spare cells, 20 vertical active links and 20 horizontal active links (including input and output links).

The PE and link failures are considered separately in the following subsections.

5.1.1 Probability of Survival After a PE Failure

The above mentioned array cannot tolerate more than four PE failures, because it has only four spares. As explained earlier, each column can tolerate only one faulty PE.

One Failure - The probability of survival in this case is 100%, because the first fault is always tolerated.

Two Failures - If the first fault is in column 0 and second fault occurs in one of the remaining columns, the array can tolerate these two faults and the total number of combinations for this occurrence is 5×15, because there are five PEs in column 0 and fifteen PEs in other columns. Similarly, if the first fault is in column 1 and second fault occurs in any one of the remaining columns, the array can tolerate these two faults. Since, the case of one fault in column 0 and the other fault in column 1 is included earlier (where column 0 has the first fault and column 1 has the second fault), the number of combinations for the occurrence of two reconfigurable faults (which are tolerated by the algorithm), with first fault in column 1 is 5×10, because there are five PEs in column 1 and ten PEs in column 2 and column 3. So, the total number of combinations for two reconfigurable faults can be written as:

$$Success_2 = 5 \times 15 + 5 \times 10 + 5 \times 5.$$

Total number of combinations for 2 faults is $\begin{pmatrix} 20 \\ 2 \end{pmatrix}$.
The probabilty of survival for two failures is:

$$P_2 = \frac{Success_2}{\begin{pmatrix} 20 \\ 2 \end{pmatrix}} = \frac{75 + 50 + 25}{190} = 0.7895 = 78.95\%. \qquad (5.1)$$

Three Failures - The number of combinations for three reconfigurable faults can be written as:

1. one fault in column 0, one in column 1 and one fault either in column 2 or in column 3 ; $5 \times 5 \times 10 = 250$,

2. one fault in column 0, one fault in column 2 and one fault in column 3 ; $5 \times 5 \times 5 = 125$ and

3. one fault in column 1, one fault in column 2 and one fault in column 3: $5 \times 5 \times 5 = 125$.

So,

$$Success_3 = 250 + 125 + 125 = 500.$$

The probability of survival in the presence of three faults can be written as:

$$P_3 = \frac{Success_3}{\begin{pmatrix} 20 \\ 3 \end{pmatrix}} = \frac{500}{1140} = 0.4386 = 43.86\%. \qquad (5.2)$$

Four Failures - The number of combinations for four reconfigurable faults can be written as:

1. one fault in column 0, one fault in column 1, one fault in column 2 and one fault in column 3: $5 \times 5 \times 5 \times 5 = 625$.

So,

$$Success_4 = 625.$$

The probability of survival in the presence of four faults can be written as:

$$P_4 = \frac{Success_4}{\binom{20}{4}} = \frac{625}{4845} = 0.1290 = 12.90\%. \tag{5.3}$$

5.1.2 Probability of Survival After a *Link* Failure

There are 20 vertical and 20 horizontal active links in the array. When an active link fails, it is replaced by the spare link and the spare is then called the active link. The algorithm checks only the active links. So for this calculation, only the active links are considered. The links are designated depending on their destination. For example, the link carrying the vertical input from the central processor to $PE_{0,0}$ is named as *V_link (0,0)*. Similarly, a link carrying the horizontal data from $PE^L_{0,0}$ to $PE^L_{0,1}$ is named as *H_link (0,1)*. So, the links can be taken as array elements and vertical link array (*V_link array*) would be a 5×4 array with elements from $(0,0)$ through $(4,3)$. The bottom most row of elements represents the links, which connect the vertical output of the array to the central processor and since the switches, providing these links, are in ST^V_1 and ST^V_2, it is assumed that the last row of links (array elements) can survive only one faulty link (element). All other array elements can survive one fault. Since in the event of a link failure, the spare link replaces the faulty link and the spare is given the same index (making that element of the array active again), each element of the array can fail twice. The second failure of any array element leads to fatal failure. Similarly, the horizontal links can be written as a 4×5 array (*H_link array*), where the elements of column

4 represent the links, connecting the horizontal output of the array to the central processor. Here, each array element can survive one failure and the second failure of the same element leads to a fatal failure.

One Fault - The probability of survival in this case is 100%, because one fault is always tolerated.

Two faults - When element (0,0) of the *V_link array* fails first, the failure of any other element in *V_link array* and *H_link array* is tolerated, but the next failure of element (0,0) of the *V_link array* leads to fatal failure. The number of the combinations of two reconfigurable link failures, with *V_link*(0,0) as the first failure is:

19 (remaining *V_link array* elements) + 20 (*H_link array* elements) = 39

and with (0,0) as the first failure, there are 40 combinations of two failures.

Similarly, when element (0,1) of *V_link array* fails first, the number of combinations of two reconfigurable faults would be 18+20=38. Failure of element (0,0) is not included here as the second failure because this combination (failure of (0,0) and (0,1)), is already included in the first case (where (0,0) is the first failure). The number of possible combinations of two failures in this case is 39.

So, the total number of reconfigurable two failures is:

$$Success_2 = \sum 39 - \sum 3 = 774,$$

where $\sum 3$ is the number of combinations of two faults, with both faults in the bottom most row of *V_link array* (it is assumed that the bottom most row can tolerate only one fault).

The total number of combinations for two link failures is $\sum 40 = 820$.

So the probability of survival in presence of two faulty links is:

$$P_2 = \frac{774}{820} = 0.9439 = 94.39\%. \qquad (5.4)$$

Three Faults - The number of combinations for three reconfigurable faults can be derived as follows:

1. when *V_link (0,0)* is one of the faulty links, the number of combinations would be $\binom{39}{2} - \binom{4}{2} = 735$, where $\binom{4}{2}$ is the number of combinations with two faults in the bottom most row of *V_link array*,

2. when *V_link (0,1)* is one of the faulty links (but *V_link (0,0)* is not faulty), the number of combinations would be $\binom{38}{2} - \binom{4}{2} = 697$ and so on.

So, the number of combinations for three reconfigurable faults can be written as:

$$Success_3 = \left[\binom{39}{2} + \binom{38}{2} + \binom{37}{2} + \ldots + \binom{2}{2}\right] - B \text{ and}$$

$$B = 16 \times \binom{4}{2} + \binom{22}{1} + \binom{21}{1} + \binom{20}{1} + \binom{21}{1} + \binom{20}{1} + \binom{20}{1},$$

where B is the number of combinations of two or more faults in the bottom most row of *V_link array*.

The number of combinations for three faults is:

$$Combinations_3 = \sum 40 + \sum 39 + \sum 38 + \ldots + 1.$$

So, the probability of survival for three link failures is:

$$P_3 = \frac{Success_3}{Combination_3} = \frac{9660}{11480} = 0.8415 = 84.15\%. \qquad (5.5)$$

It can be observed that the analysis becomes increasingly complex as the number of failures increase. Therefore simulation is used to get the values of probability of survival for a greater number of faults.

5.2 Analysis of Simulation Results

As explained earlier, it is difficult to calculate the probability of survival, for large number of faults and large arrays using an analytical method. So a computer program is written (the basic control flow of the program is given in Appendix A) to simulate the algorithm with a view to calculate the probability of survival.

5.2.1 Simulation Software Outline

The simulation program injects the specified number of faults randomly and checks the outcome of the algorithm. For an example, when the program needs to inject one PE and one link failures, it generates a random number PE_LINK, which can be either '0' or '1'. When it is '0', a PE failure is injected in the array. For this, the index value (i, j) is generated randomly and failure of $PE_{i,j}$ is injected and reconfiguration algorithm is performed on the array.

When PE_LINK is '1', a link failure is injected. Here, another random number, H_V is generated. When H_V is '0' ('1'), horizontal (vertical) link failure is injected by randomly generating the index (i, j) of the link. This program does not assume that the bottom-most row of the *V_link array* can tolerate only one fault (as was assumed for the analytical calculation). Instead, here a random number is generated, which provides the information about the outcome of the algorithm for the failure of O_j^V link in presence of faulty O_{j1}^V $(j1 < j)$. When O_j^V fails in presence of faulty O_{j1}^V $(j1 > j)$, fatal failure occurs. The program simulates the algorithm completely by changing the states of the switches, PEs and links and checking the outcome.

The program injects the specified number of faults for a specified number of times n (by going in the same loop) and every time it starts with a fresh array (fault free array).

Every time the program enters the loop (the process is called a *trial*), it returns one of the two possible outcomes, *success*, S or *failure*, F. An outcome of S informs that the reconfiguration attempt was successful and F indicates the occurrence of a fatal failure.

5.2.2 Confidence Level of the Simulation

In this simulation, the trials are independent of each other, because every trial starts with a fresh array and thus the probability of success remains constant from trial to trial. These trials are called *Bernoulli trials* and the random variable X, which denotes the number of successes in n trials has a binomial distribution given by $p(x)$ and:

$$\begin{aligned} p(x) &= \binom{n}{x} . p^x . (1-p)^{n-x}, & x &= 0, 1, 2, \ldots, n \\ &= 0 & & \text{otherwise,} \end{aligned}$$

where, p is the probabilty of success of any random trial.

The binomial distribution approaches the normal distribution in the limit as n becomes large. In general, the approximation is fairly good as long as $n \cdot p > 5$ and $n \cdot q > 5$, where $q = (1-p)$.

The probability density allows one to find the probability that the data would assume some value within a specified range at any time. A normal density function $f(z)$ (shown in Figure 5.1) determines the shape of the plot. When the number of successful trials is X for n trials, the probability of success for a randomly selected trial can be estimated as:

$$\hat{p} = \frac{X}{n}, \tag{5.6}$$

where $\hat{p}$ is called the estimate of p.

Now, two values p_1 and p_2 (which are functions of $\hat{p}$) can be determined in such a way that the probability of p lying between p_1 and p_2 is $(1-\alpha)$. That is:

$$P\left(p_1 \leq p \leq p_2\right) = 1 - \alpha.$$

Therefore (p_1, p_2) forms an interval, which has the probability $(1-\alpha)$ of capturing the true value of p. This interval is called the *confidence interval* and $(1-\alpha)$ is called the confidence coefficient (confidence level) [22].

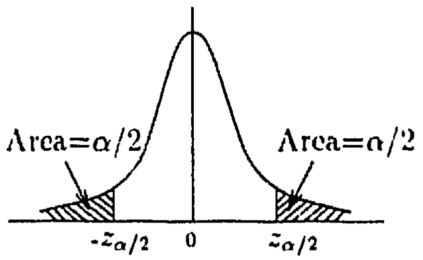


Figure 5.1: Normal Density Function

The confidence interval for p can be written as:

$$\frac{X}{n} - z_{\alpha/2}\cdot\sqrt{\frac{\hat{p}\cdot\hat{q}}{n}} \leq p \leq \frac{X}{n} + z_{\alpha/2}\cdot\sqrt{\frac{\hat{p}\cdot\hat{q}}{n}}, \tag{5.7}$$

where $\hat{p}$ and $\hat{q}$ are the estimated values of p and q and $z_{\alpha/2}\cdot\sqrt{\frac{\hat{p}\cdot\hat{q}}{n}}$ is the margin of error E in the estimated value. So,

$$E \leq z_{\alpha/2}\sqrt{\frac{\hat{p}\cdot\hat{q}}{n}}, \tag{5.8}$$

which gives Equation 5.9.

$$n \geq \left(\frac{z_{\alpha/2}}{E}\right)^2 \cdot \hat{p}\cdot\hat{q}. \tag{5.9}$$

For the simulation, the number of trials is calculated based on equation 5.9. The maximum value of $\hat{p}\cdot\hat{q}$ is 0.25, when $\hat{p} = \hat{q} = 0.5$. If we want the confidence level to be 95% and the half width of the confidence interval to be 2%, then

$$n \geq \left(\frac{z_{\alpha/2}}{0.02}\right)^2 \times 0.25. \tag{5.10}$$

For a confidence level of 95% ($\alpha = 0.05$), $z_{\alpha/2} = 1.96$ (from the cumulative normal distribution table [22]). So,

$$\begin{aligned} n &\geq \left(\tfrac{1.96}{0.02}\right)^2 \times 0.25 \\ &\geq 2401. \end{aligned} \tag{5.11}$$

Now, if the total number of trials is more than 2401, it can be said confidently that the probability of success in any random trial is $\hat{p} \pm 2\%$, 19 times out of 20.

5.2.3 Probability of Failure

The probability of survival goes down with the increasing number of faults but the probability of occurrence of a large number of faults also goes down. In any array as the failures can be reasonably assumed to be independent, the binomial distribution can be used to calculate the probability of occurrence of x failures (the active array size is $m \times n$).

If we consider the PE failures, total number of PEs in the array is $(m+1) \times n$, so probability of x PE failures is,

$$P_{x,PE} = \begin{pmatrix} (m+1).n \\ x \end{pmatrix} \cdot p_{PE}^{x} \cdot q_{PE}^{(m+1).n-x},$$

where p_{PE} is the probabilty of failure for PEs. Here, the value of p_{PE} would be very small therefore the Poisson distribution can be used to approximate the binomial distribution and then

$$P_{x,PE} = \frac{e^{-\lambda_{PE}} \cdot \lambda_{PE}^{x}}{x!}$$

where $\lambda_{PE} = (m+1) \cdot n \cdot p_{PE}$.

Similarly, the probability of x link failures is

$$P_{x,link} = \frac{e^{-\lambda_{link}} \cdot \lambda_{link}^{x}}{x!}$$

where $\lambda_{link} = [(m+1) \cdot n + m \cdot (n+1)] \cdot p_{link}$. Here $(m+1) \cdot n$ is the number of active vertical links and $m \cdot (n+1)$ is the number of active horizontal links. It is assumed that the probabilities of a horizontal link failure and a vertical link failure are both equal to p_{link}.

Since the occurrence of PE failures and link failures are independent of each other, the probability of x_1 PE failures and x_2 link failures can be written as:

$$\begin{aligned} P_{PE=x_1,link=x_2} &= P_{x_1,PE} \times P_{x_2,link} \\ &= \frac{e^{-\lambda_{PE}} \cdot \lambda_{PE}^{x_1}}{x_1!} \times \frac{e^{-\lambda_{link}} \cdot \lambda_{link}^{x_2}}{x_2!}. \end{aligned}$$

The probability of occurrence of various failures for a 4×4 array are listed in Table 5.1 (assumed $p_{PE} = 10^{-4}$ and $p_{link} = 10^{-6}$). The probability of switch and link failures is less than that of PEs because the PE circuitry is more complex than that of switches and links in most cases.

5.2.4 Simulation Results

The results of the simulation program are listed in Table 5.1 for various values of n for a 4×4 array (the maximum number of injected PE faults is four and injected link faults is three). The first column in the table gives the number of faulty PEs (i) and the second column gives the number of faulty links (j). The joint probability of i PE failures and j link failures is listed in column 3. The estimated probability of survival is listed in the other columns for various values of n. It can be seen that the estimated value of p becomes stable, once n becomes large. The complete table of outputs (for $n = 3000$, array size $= 4 \times 4$, maximum number of PE faults $= 4$ and maximum number of link faults $= 7$) is given in Table 5.2 and various confidence intervals are calculated and listed in the same table (for 95% confidence level). The first two columns of the table give the number of PE and link faults. The estimated probability of survival ($\hat{p}$, output of the simulation) is given in column 3. The confidence-interval is calculated based on $\hat{p}$ and listed in column 4.

It can be seen that the analytically calculated values of probability of survival are well within the confidence interval (calculated from the simulation results) for PE failures but they are below the confidence interval for link failures. It is because of the assumption, that the bottom most row of *V_link array* can survive only one fault, which was made for the analytical calculation.

The overall probability of survival for a 4×4 array is calculated based on tables 5.1 and 5.2 and it is 99.903%.

Table 5.1: Estimated Values of Probabilities of Survival (Array Size=4 × 4)

Number of PE Faults (i)	Number of link faults (j)	Probability of this occurrence *	Estimated value, $\hat{p}$ (%)				
			n=10	n=100	n=1000	n=2100	n=5000
0	0	9.97×10^{-1}	100.00	100.00	100.00	100.00	100.00
	1	3.99×10^{-5}	100.00	100.00	100.00	100.00	100.00
	2	7.98×10^{-10}	100.00	92.00	95.30	95.00	95.82
	3	1.06×10^{-14}	50.00	87.00	88.80	88.76	88.12
1	0	1.99×10^{-3}	100.00	100.00	100.00	100.00	100.00
	1	7.98×10^{-8}	100.00	100.00	99.70	99.43	99.50
	2	1.60×10^{-12}	90.00	95.00	94.10	95.76	95.38
	3	2.13×10^{-17}	90.00	86.00	87.80	87.42	87.90
2	0	1.99×10^{-6}	80.00	77.00	77.10	76.86	78.60
	1	7.98×10^{-11}	40.00	78.00	78.40	79.19	78.18
	2	1.60×10^{-15}	90.00	72.00	71.00	75.43	74.54
	3	2.12×10^{-20}	80.00	71.00	69.70	69.62	67.48
3	0	1.33×10^{-9}	40.00	49.00	43.40	45.00	44.78
	1	5.32×10^{-14}	40.00	40.00	41.80	42.43	43.24
	2	1.06×10^{-18}	60.00	36.00	41.50	40.81	41.02
	3	1.42×10^{-23}	40.00	41.00	39.90	35.71	37.40
4	0	6.65×10^{-13}	0.00	22.00	14.90	12.71	12.82
	1	2.66×10^{-17}	0.00	11.00	12.20	12.90	12.32
	2	5.32×10^{-22}	20.00	14.00	12.00	9.86	12.32
	3	7.10×10^{-27}	30.00	10.00	11.50	11.76	10.70

Table 5.2: Estimated Values of Probabilities of Survival (Array Size=4 × 4)

Number of *PE* Faults	Number of link Faults	Estimated Probability, $\hat{p}$ (from simulation)	Confidence Interval n=3000 (*Confidence level* = 95%)
0	0	100.00	100.00 - 100.00
	1	100.00	100.00 - 100.00
	2	95.77	95.05 - 96.19
	3	87.50	86.32 - 88.68
	4	77.30	75.80 - 78.80
	5	65.80	64.10 - 67.50
	6	55.13	53.35 - 56.91
	7	43.47	41.69 - 45.24
1	0	100.00	100.00 - 100.00
	1	99.33	99.04 - 99.62
	2	95.17	94.40 - 95.93
	3	88.40	87.25 - 89.55
	4	77.30	75.80 - 78.80
	5	64.67	62.96 - 66.38
	6	54.80	53.02 - 56.58
	7	41.73	39.97 - 43.50
2	0	79.17	77.71 - 80.62
	1	77.77	76.28 - 79.25
	2	74.50	72.94 - 76.06
	3	69.00	67.34 - 70.66
	4	61.67	59.93 - 63.41
	5	51.37	49.58 - 53.16
	6	43.90	42.12 - 45.68
	7	35.27	33.56 - 36.98
3	0	43.47	41.69 - 45.24
	1	43.93	42.16 - 45.71
	2	42.47	40.70 - 44.24
	3	37.83	36.10 - 39.57
	4	33.47	31.78 - 35.16
	5	28.50	26.88 - 30.12
	6	24.57	23.03 - 26.11
	7	20.53	19.09 - 21.98
4	0	12.07	10.90 - 13.23
	1	12.90	11.70 - 14.10
	2	15.20	13.92 - 16.48
	3	10.40	9.31 - 11.49
	4	9.80	8.74 - 10.86
	5	8.03	7.06 - 9.01
	6	7.27	6.34 - 8.20
	7	6.13	5.27 - 6.99

Various probabilities of survival for different array sizes are listed in Appendix B for 95% confidence level.

5.3 Concluding Remarks

In this chapter, the proposed algorithm was evaluated. It was shown that this algorithm introduces a delay of two clock periods for PE failures and of one clock period for link failures. Therefore it can be inferred that the time overhead is very small.

The locality of interconnections is maintained here by using global deformation. The amount of increase in the hardware is very small for the *only PE failure algorithm* but it is slightly more for the *combined PE and link failure algorithm* due to complex vertical switch control circuit.

It can be seen that though the probability of survival is less for large number of faults, the probability of this occurrence is also low. The overall probability of survival of this algorithm for a 4×4 array is 99.903% (assumed $p_{PE} = 10^{-4}$ and $p_{link} = 10^{-6}$).

Chapter 6

CONCLUSIONS

The processing speed of a computation can be increased by ensuring multiple computation per memory access. Systolic arrays accomplish this and in addition these arrays provide modularity and regular data flow.

To improve the yield and reliability, various fault detection and reconfiguration schemes are used. In Chapter 1, the concept of systolic arrays was explained and various existing reconfiguration algorithms were discussed. It can be seen that most of the existing schemes are efficient for improving the production time yield but they are not suitable for run-time reliability improvement because they need an external processor to run the algorithm. In addition these schemes assume the network to be always fault-free, which is difficult to achieve. The scheme, proposed here, does not assume a perfect switching network and it is capable of tolerating the link failures also.

The scheme proposed in this report can be used efficiently for on-line reconfiguration to improve run-time reliability. The algorithm for *PE* failures was presented in Chapter 2. A bottom row of spares is provided to the array and in the case of a *PE* failure, a global shift is performed, if the spare cell (for the particular column) is available. The *PEs* are of a self-testing type and in the event of a fault detection, *PEs* invoke the reconfiguration by generating the reconfiguration requests.

An algorithm for *PE* and link failures was presented in Chapter 3. Here, each link is duplicated and a bottom row of spare cells is provided to the array. In the case of a PE failure a global shift is performed if the spare cell (for the particular column) is available. The link failures are detected by using parity bits, the PEs perform parity checks on incoming data and any error in the incoming data is taken as the incoming link failure. In the event of a horizontal link failure, the processing element simply selects the second input port, if it is using the first input port. If the PE is using the second input port and it detects an input data error, a fatal failure occurs. In the case of vertical link failure, the PE invokes a reconfiguration by generating a reconfiguration request. Various states were defined for the switches and it was proved that the proposed number of switch states is sufficient to implement the algorithm.

A central processor is linked to the array for providing the inputs and receiving the outputs. The central processor controls the clock input of the array and when a fault occurs, the central processor inserts delays in the clock as required by the reconfiguration algorithm. This algorithm makes full use of non-faulty partial results after the occurrence of a fault and it does not require flushing of the array every time a fault occurs.

The probability of survival for this algorithm was calculated analytically in Chapter 4. Next, the simulation results were presented. The simulation program injects random faults in the array and checks the outcome of the algorithm. The probabilities of survival were estimated based on the outcome of the random fault injection and a 95% confidence interval was defined for each estimated value. The number of trials was calculated based on a maximum margin of error of 2% and on the required value of the confidence level (which is assumed to be 95% here). The simulation results were analyzed in Chapter 4 and it was shown that the overall probability of survival is approximately 99.903% for a 4×4 array (assumed probability of PE failure $= 10^{-4}$ and probability of line failure $= 10^{-6}$).

It was shown that the probability of survival after a fault occurrence decreases with the increasing number of faults but it is overshadowed by the fact that the probability of occurrence of faults also decreases with increasing number of faults.

In the next chapter some suggestions for further research are given.

Bibliography

[1] A. Huang, "Architectural Considerations Involved in the Design of an Optical Signal Computer," *Proc. of the IEEE*, vol. 72, no. 7, pp. 780-786, July 1984.

[2] H.T. Kung, "Why Systolic Architectures?," *Computer*, pp. 37-45, January 1982.

[3] P.O. Dianne, "Systolic Arrays for Matrix Transpose and other Reorderings," *IEEE Transactions on Computers*, vol. c-36, no. 1, pp. 117-122, January 1987.

[4] R.B. Urquhart and D.Wood, "Systolic Matrix and Vector Multiplication Methods for Signal Processing," *IEE Proceedings*, vol. 131, pt. F, no. 6, pp. 623-631, October 1984.

[5] J.C. Ward, J.V. McCanny and J.G. McWhirter, "Bit Level Systolic Array Implementation of the Winograd Fourier Transform Algorithm," *IEE Proceedings*, vol. 132, pt. F, no. 6, pp. 473-479, October 1985.

[6] K.H. Huang and J.A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Trans. Computers*, vol. C-33, no. 6, pp. 518-528, June 1984.

[7] J.H. Patel and L.Y. Fung, "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands," *IEEE Trans Computers*, vol. C-31, No. 7, pp. 589-595, July 1982.

[8] J.H. Patel and L.Y. Fung, "Concurrent Error Detection in Multiply and Divide Arrays," *IEEE Trans Computers*, vol. C-32, no. 4, pp. 417-422, April 1983.

Chapter 7

SUGGESTIONS FOR FURTHER RESEARCH

In this report an algorithm for on-line reconfiguration was presented. This algorithm can be extended in various directions depending upon the requirements. Some of the extensions are listed below.

- The proposed algorithm uses only one row of spare cells therefore it would not be very effective for high failure rate of *PEs*. For such cases a greater number of spare cells is required. An additional column of spares can be added and the algorithm can be modified to make effective use of column and row spares.

- When a spare column and a spare row of *PEs* are used, each *PE* requires 3 or more static coefficient latches (the exact number depends on the algorithm). This requirement can be reduced by modifying the algorithm, so that only two copies of each static coefficient are kept in the array. When a *PE* fails, the static coefficients can be moved from one cell to the other (if required). This requires some additional time for reconfiguration but the hardware is reduced.

[9] S.-W. Chan, S.S. Leung and C.-L. Way, "Systematic Design Strategy for Concurrent Error Diagnosable Iterative Logic Arrays," *IEE Proceedings*, pt. E, vol. 135, no. 2, pp. 87-94, March 1988.

[10] A. Majumdar, C.S. Raghvendra and M.A. Breuer, "Fault Tolerance in Linear Systolic Array using Time Redundancy," *IEEE Trans. Computers*, vol. 39, no. 2, pp. 269-276, February 1990.

[11] R.K. Gulati and S.M. Reddy, "Concurrent Error Detection in VLSI Array Structures," *ICCD 1986*, pp. 488-491.

[12] R.J. Cosentino, "Concurrent Error Correction in Systolic Architecture," *IEEE Trans. CAD*, vol. 7, no. 1, pp. 117-125, January 1988.

[13] R.M. Lea and H.S. Bolouri, "Fault Tolerance : Step Towards WSI," *IEE proceedings*, pt. E, vol. 135, no. 6, pp. 289-297, November 1988.

[14] R. Negrini, M. Sami and R. Stefanelli, "Fault Tolerance Techniques for Array Structures Used in Supercomputing," *Computer*, vol. 19, no. 2, pp. 78-87, February 1986.

[15] M. Chean and J.A.B. Fortes, "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," *Computer*, vol. 23, no. 1, pp. 55-69, January 1990.

[16] C.W.H. Lam, H.F. Li and R. Jayakumar, "A Study of Two Approaches for Reconfiguring Fault-Tolerant Systolic Arrays," *IEEE trans. Computers*, vol. C-38, no. 6, pp. 833-844, June 1989.

[17] S.Y. Kuo and W.K. Fuchs, "Efficient Spare Allocation for Reconfigurable Arrays," *IEEE Design and Test*, vol. 4, no. 1, pp. 24-31, February 1987.

[18] A.L. Rosenberg, "The Diogenes Approach to Testable fault-Tolerant Arrays of Processors," *IEEE Trans. Computers*, vol. C-32, no. 10, pp. 902-910, October 1983.

[19] F. Lombardi, M. Sami and R. Stefanelli, "Reconfiguration of VLSI arrays by Covering," *IEEE trans. CAD.*, vol. 8, no. 9, pp. 952-964, September 1986.

[20] A.D. Singh, "Interstitial redundancy : An Area Efficient Fault Tolerance Scheme for Large Area VLSI Processor Arrays," *IEEE Trans. Computers*, vol. C-37, no. 11, pp. 1398-1410, November 1988.

[21] L. Snyder, "Introduction to the Configurable Highly Parallel Computer," *IEEE Computer*, vol. 15, no. 1, pp. 47-56, January 1982.

[22] W. W. Hines, *Probability and Statistics in Engineering and Management Science*, John Wiley & Sons, pp. 240-256, 1980.

Appendix A

Program Structure

This program calculates the probability of survival by injecting random faults in the array and running the reconfiguration algorithm.

The basic flow of controls is given below:

```
Start :
                get the array dimensions from the terminal;
                get the number of trials from the terminal;
                get the maximum number of faults, (maxfault_PE, maxfault_link)
                                        to be injected;
                initialize various variables;

Fault :
                start with maximum number of PE faults, fault_PE to be 0 and
                maximum number of link faults, fault_link to be 1.

Trial :
                generate arrays;        /* one array each for PEs, vertical
                                           switches, horizontal switches,
                                           vertical links and horizontal links */
Loop :
                decide randomly, which fault (PE or link) to inject;
                if link fault has to be injected, go to Link_fault;

PE_fault :
                generate the index (i,j) of the failed PE randomly;
                if $PE_{i,j}$ is non-faulty then go to PE_algorithm;
                go to PE_fault;

PE_algorithm :
                check the success of the algorithm;
                if algorithm is successful, go to Successful_PE;
                increment the number of attempts for (fault_PE, fault_link);
                go to Next_trial;

Successful_PE :
                modify the $PE$, switch and link arrays;
                if number of $PE$ and link faults, already injected
                                    = (fault_PE, fault_link), go to success;
                increment the number of injected $PE$ faults;
                go to loop;

Link_fault :
                decide randomly, which failure (horizontal link or vertical
                                    link) to inject;
                if vertical link failure has to be injected, go to Vertical_link;

Horizontal_link :
                generate the index (i,j) of the failed link randomly;
                check the success of the algorithm;
                if algorithm is successful, go to Successful_link;
                increment the number of attempts for (fault_PE, fault_link);
                go to Next_trial;

Vertical_link :
                generate the index (i,j) of the failed link randomly;
                check the success of the algorithm;
                if algorithm is successful, go to Successful_link;
                increment the number of attempts for (fault_PE, fault_link);
                go to Next_trial;

Successful_link :
                modify the corresponding switch and link array;
                if number of $PE$ and link faults, already injected
                                    = (fault_PE, fault_link), go to Success;
                increment the number of injected link faults;
                go to Loop;

Next_trial :
                reset the number of faults, already injected;
                increment the number of trials, already attempted;
                go to Trial;

Success :
                increment the number of successes recorded for (fault_PE, fault_link);
                increment the number of attempts for (fault_PE, fault_link);
                increment the number of trials, already attempted;
                if number of trials, already attempted is less than user
```

```
                        specified number of trials, go to Trial;
            if (fault_PE, fault_link) is less than
                        (maxfault_PE, maxfault_link), go to Next;
            calculate estimated probability of survival for each
                        combination of (fault_PE, fault_link);
            go to End;

Next :
            If fault_link is less than maxfault_link, go to increment_link;
            increment fault_PE;
            reset fault_link;
            go to Reset_trial;

Increment_link :
            increment fault_link;
Reset_trial :
            reset number of trials, already attempted;
            reset number of faults, already injected;
            go to Trial;

End :       Stop.
```

Appendix B

Probability of Survival

Here the simulation results are listed for various array sizes. The first two columns of the table show the number of PE and link faults corresponding to the particular row. The other columns give the 95% confidence interval of the probability of survival (calculated from the simulation results) for different array sizes (number of trials = 3000).

Number of PE Faults	Number of link Faults	Estimated Probability Confidence Interval (*Confidence level* = 95%)			
		5×5	6×6	10×10	20×20
0	0	100.00 – 100.00	100.00 – 100.00	100.00 - 100.00	100.00 - 100.00
0	1	100.00 – 100.00	100.00 – 100.00	100.00 - 100.00	100.00 - 100.00
0	2	96.35 – 97.58	97.57 – 98.56	99.04 - 99.62	99.69 - 99.98
0	3	91.28 – 93.19	93.26 – 94.94	97.28 - 98.32	98.88 - 99.52
0	4	82.86 – 85.47	87.60 – 89.86	94.33 - 95.87	98.37 - 99.16
0	5	74.03 – 77.10	79.63 – 82.44	91.91 - 93.76	97.88 - 98.79
0	6	61.58 – 65.02	71.45 – 74.62	88.96 - 91.11	95.37 - 96.76
0	7	53.15 – 56.71	63.60 – 67.00	83.38 - 85.96	95.59 - 96.95
0	8	41.33 – 44.87	60.83 – 64.30	79.75 - 82.53	86.32 - 88.68
0	9	31.65 – 35.02	58.78 – 62.28	75.22 - 78.25	94.36 - 95.90
0	10	24.56 – 27.71	48.28 – 51.86	70.22 - 73.44	93.93 - 95.53
0	11	25.67 – 28.86	29.48 – 32.79	64.84 - 68.22	82.00 - 84.67
0	12	21.20 – 24.20	21.07 – 24.06	58.35 - 61.85	86.32 - 88.68
0	13	13.02 – 15.52	15.33 – 18.00	51.85 - 55.42	85.45 - 87.88
0	14	8.48 – 10.58	12.12 – 14.55	48.84 - 52.42	71.75 - 74.92
0	15	8.48 – 10.58	8.93 – 11.07	39.17 - 42.69	69.85 - 73.08

Number of *PE* Faults	Number of link Faults	Estimated Probability Confidence Interval (*Confidence level* = 95%)			
		5 × 5	6 × 6	10 × 10	20 × 20
1	0	100.00 – 100.00	100.00 – 100.00	100.00 – 100.00	100.00 – 100.00
1	1	99.08 – 99.65	99.55 – 99.92	99.46 – 99.87	99.64 – 99.96
1	2	96.72 – 97.88	96.79 – 97.94	98.41 – 99.19	99.42 – 99.85
1	3	90.33 – 92.34	92.76 – 94.51	96.57 – 97.76	98.84 – 99.49
1	4	81.72 – 84.41	87.57 – 89.83	94.11 – 95.69	98.14 – 98.99
1	5	73.38 – 76.48	80.18 – 82.95	91.42 – 93.32	97.61 – 98.59
1	6	64.41 – 67.79	72.46 – 75.60	87.95 – 90.18	96.28 – 97.52
1	7	53.35 – 56.91	65.72 – 69.08	84.38 – 86.89	94.11 – 95.69
1	8	43.29 – 46.85	55.60 – 59.14	79.36 – 82.18	93.47 – 95.13
1	9	33.19 – 36.61	46.61 – 50.19	74.37 – 77.43	92.19 – 94.01
1	10	25.54 – 28.72	39.94 – 43.46	69.82 – 73.05	90.01 – 92.06
1	11	17.50 – 20.30	31.55 – 34.92	64.57 – 67.96	88.82 – 90.98
1	12	13.18 – 15.69	24.27 – 27.40	57.58 – 61.09	85.31 – 87.75
1	13	9.94 – 12.19	21.10 – 24.10	54.99 – 58.54	84.79 – 87.27
1	14	5.81 – 7.59	15.88 – 18.58	47.08 – 50.66	78.57 – 81.43
1	15	4.16 – 5.71	10.55 – 12.85	43.52 – 47.08	79.94 – 82.73
2	0	82.83 – 85.44	85.52 – 87.95	89.66 – 91.74	95.15 – 96.58
2	1	80.08 – 82.86	82.96 – 85.57	89.21 – 91.33	93.76 – 95.[illegible]
2	2	77.54 – 80.46	81.14 – 83.86	88.33 – 90.53	93.22 – 94.[illegible]
2	3	73.69 – 76.78	78.16 – 81.04	88.40 – 90.60	93.26 – 94.94
2	4	69.14 – 72.39	72.91 – 76.03	84.17 – 86.70	92.76 – 94.51
2	5	60.67 – 64.13	68.06 – 71.34	83.03 – 85.63	91.45 – 93.35
2	6	53.59 – 57.15	63.29 – 66.71	79.08 – 81.92	90.85 – 92.81
2	7	44.62 – 48.18	56.03 – 59.57	75.66 – 78.67	90.75 – 92.72
2	8	37.52 – 41.01	47.31 – 50.89	71.72 – 74.88	87.98 – 90.22
2	9	27.67 – 30.93	42.95 – 46.51	68.02 – 71.31	87.74 – 89.99
2	10	22.67 – 25.73	33.19 – 36.61	63.16 – 66.57	83.52 – 86.08
2	11	17.27 – 20.06	26.20 – 29.40	59.15 – 62.65	83.03 – 85.63
2	12	13.14 – 15.66	20.65 – 23.62	52.95 – 56.51	81.48 – 84.18
2	13	8.96 – 11.11	17.27 – 20.06	49.41 – 52.99	80.63 – 83.37
2	14	4.65 – 6.28	13.02 – 15.52	45.91 – 49.49	77.47 – 80.39
2	15	3.24 – 4.63	9.98 – 12.22	39.74 – 43.26	75.36 – 78.38
3	0	50.48 – 54.05	59.15 – 62.65	72.91 – 76.03	86.18 – 88.56
3	1	49.61 – 53.19	54.89 – 58.44	72.46 – 75.60	84.62 – 87.11
3	2	48.58 – 52.16	56.33 – 59.87	70.19 – 73.41	84.03 – 86.57
3	3	47.21 – 50.79	52.78 – 56.35	69.65 – 72.89	83.03 – 85.63
3	4	41.43 – 44.97	50.28 – 53.85	67.89 – 71.18	83.41 – 85.99
3	5	39.50 – 43.03	47.28 – 50.86	65.62 – 68.98	82.00 – 84.67
3	6	34.68 – 38.12	42.29 – 45.84	64.03 – 67.43	79.49 – 82.31
3	7	28.39 – 31.67	39.90 – 43.43	60.40 – 63.87	80.08 – 82.86
3	8	24.33 – 27.47	31.98 – 35.36	57.41 – 60.93	79.60 – 82.40

Number of *PE* Faults	Number of link Faults	Estimated Probability Confidence Interval (*Confidence level* = 95%)			
		5×5	6×6	10×10	20×20
3	9	19.48 – 22.39	27.80 – 31.06	53.96 - 57.51	78.60 - 81.46
3	10	15.56 – 18.24	23.74 – 26.86	50.68 - 54.25	77.95 - 80.85
3	11	11.03 – 13.37	19.90 – 22.83	49.95 - 53.52	74.71 - 77.76
3	12	8.61 – 10.72	15.04 – 17.69	43.85 - 47.42	74.34 - 77.40
3	13	5.31 – 7.03	14.01 – 16.59	42.56 - 46.11	72.29 - 75.44
3	14	4.65 – 6.28	10.33 – 12.61	36.73 - 40.21	70.12 - 73.34
3	15	2.81 - 4.12	7.28 – 9.25	32.93 - 36.34	67.68 - 70.98
4	0	21.01 – 23.99	30.76 – 34.11	52.42 - 55.98	71.95 - 75.11
4	1	21.49 – 24.51	29.67 – 32.99	50.85 - 54.42	70.94 - 74.13
4	2	22.28 – 25.32	28.46 – 31.74	47.98 - 51.56	70.70 - 73.90
4	3	21.62 – 24.64	26.98 – 30.22	50.91 - 54.49	69.95 - 73.18
4	4	18.34 – 21.19	27.02 – 30.25	47.98 - 51.56	68.22 - 71.51
4	5	16.82 – 19.58	24.79 – 27.94	46.18 - 50.05	69.78 - 73.02
4	6	14.04 – 16.62	21.85 – 24.88	45.41 - 48.99	68.06 - 71.34
4	7	11.06 – 13.41	18.31 – 21.16	42.92 - 46.48	67.68 - 70.98
4	8	11.22 – 13.58	15.82 – 18.52	43.22 - 46.78	66.16 - 69.50
4	9	10.71 – 13.02	16.20 – 18.93	36.06 - 39.54	64.61 - 67.99
4	10	6.21 – 8.05	12.82 – 15.31	36.46 - 39.94	65.89 - 69.24
4	11	4.99 – 6.67	10.49 – 12.78	33.56 - 36.98	63.39 - 66.81
4	12	3.09 – 4.45	8.55 – 10.65	29.90 - 33.23	61.71 - 65.16
4	13	1.88 – 2.98	7.19 – 9.15	27.93 - 31.20	62.72 - 66.15
4	14	2.06 – 3.21	5.09 – 6.78	27.57 - 30.83	58.72 - 62.22
4	15	1.15 – 2.05	3.97 – 5.49	23.12 - 26.21	57.58 - 61.09
5	0	4.65 – 6.28	10.61 – 12.92	31.51 - 34.89	58.85 - 62.35
5	1	4.68 – 6.32	9.63 – 11.84	30.17 - 33.50	56.40 - 59.93
5	2	4.19 – 5.74	10.71 – 13.02	29.80 - 33.13	56.30 - 59.83
5	3	3.76 – 5.24	9.66 – 11.88	28.69 - 31.98	57.01 - 60.53
5	4	3.60 – 5.06	9.85 – 12.08	28.79 - 32.08	56.54 - 60.06
5	5	3.27 – 4.67	7.31 – 9.29	27.41 - 30.66	53.92 - 57.48
5	6	3.30 – 4.70	7.85 – 9.88	27.11 - 30.35	55.60 - 59.14
5	7	2.75 – 4.05	7.22 – 9.18	24.40 - 27.54	54.29 - 57.84
5	8	2.09 – 3.24	6.93 – 8.87	24.76 - 27.91	52.72 - 56.28
5	9	1.62 – 2.65	5.62 – 7.38	22.41 - 25.46	54.73 - 58.27
5	10	0.78 – 1.55	4.72 – 6.35	22.11 - 25.15	51.85 - 55.42
5	11	1.35 – 2.31	3.39 – 4.81	19.45 - 22.35	50.55 - 54.12
5	12	0.73 – 1.47	3.54 – 4.99	19.19 - 22.08	48.34 - 51.92
5	13	0.59 – 1.28	1.91 – 3.02	18.11 - 20.95	48.61 - 52.19
5	14	0.43 – 1.04	1.59 – 2.61	16.75 - 19.51	47.08 - 50.66
5	15	0.10 – 0.50	1.15 – 2.05	14.46 - 17.07	45.88 - 49.45

Number of PE Faults	Number of link Faults	Estimated Probability Confidence Interval ($Confidence\ level\ =\ 95\%$)			
		5 × 5	6 × 6	10 × 10	20 × 20
6	0	0.00 - 0.00	1.65 – 2.69	15.78 - 18.48	45.01 - 48.59
6	1	0.00 - 0.00	1.50 – 2.50	15.91 - 18.62	43.59 - 47.15
6	2	0.00 - 0.00	1.27 – 2.20	16.27 - 19.00	43.55 - 47.11
6	3	0.00 - 0.00	1.30 – 2.24	13.85 - 16.42	42.56 - 46.11
6	4	0.00 - 0.00	1.38 – 2.35	16.53 - 19.27	40.20 - 43.73
6	5	0.00 - 0.00	1.21 – 2.12	14.56 - 17.17	41.06 - 44.60
6	6	0.00 - 0.00	0.70 – 1.43	14.53 - 17.14	42.46 - 46.01
6	7	0.00 - 0.00	0.75 – 1.51	11.89 - 14.31	40.33 - 43.87
6	8	0.00 - 0.00	0.64 – 1.36	12.79 - 15.28	39.01 - 42.53
6	9	0.00 - 0.00	0.81 – 1.59	11.00 - 13.34	40.30 - 43.83
6	10	0.00 - 0.00	0.40 – 1.00	10.87 - 13.20	36.86 - 40.34
6	11	0.00 - 0.00	0.53 – 1.20	10.45 - 12.75	38.05 - 41.55
6	12	0.00 - 0.00	0.40 – 1.00	8.70 - 10.83	37.35 - 40.85
6	13	0.00 - 0.00	0.22 – 0.71	8.42 - 10.51	36.10 - 39.57
6	14	0.00 - 0.00	0.27 – 0.79	7.94 - 9.99	34.81 - 38.26
6	15	0.00 - 0.00	0.22 – 0.71	7.60 - 9.60	34.12 - 37.55
7	0	0.00 - 0.00	0.00 – 0.00	6.62 - 8.51	29.31 - 32.62
7	1	0.00 - 0.00	0.00 – 0.00	6.09 - 7.91	29.80 - 33.13
7	2	0.00 - 0.00	0.00 – 0.00	5.65 - 7.42	28.26 - 31.54
7	3	0.00 - 0.00	0.00 – 0.00	5.31 - 7.03	29.51 - 32.82
7	4	0.00 - 0.00	0.00 – 0.00	6.43 - 8.30	29.90 - 33.23
7	5	0.00 - 0.00	0.00 – 0.00	6.81 - 8.72	29.31 - 32.62
7	6	0.00 - 0.00	0.00 – 0.00	5.03 - 6.71	28.13 - 31.40
7	7	0.00 - 0.00	0.00 – 0.00	5.06 - 6.74	29.44 - 32.76
7	8	0.00 - 0.00	0.00 – 0.00	5.06 - 6.74	27.44 - 30.69
7	9	0.00 - 0.00	0.00 – 0.00	4.59 - 6.21	28.03 - 31.30
7	10	0.00 - 0.00	0.00 – 0.00	4.68 - 6.32	27.47 - 30.73
7	11	0.00 - 0.00	0.00 – 0.00	4.13 - 5.67	26.16 - 29.37
7	12	0.00 - 0.00	0.00 – 0.00	3.45 - 4.88	26.26 - 29.47
7	13	0.00 - 0.00	0.00 – 0.00	3.36 - 4.77	24.33 - 27.47
7	14	0.00 - 0.00	0.00 – 0.00	2.90 - 4.23	26.03 - 29.23
7	15	0.00 - 0.00	0.00 – 0.00	2.48 - 3.72	24.33 - 27.47
8	0	0.00 - 0.00	0.00 – 0.00	1.70 - 2.76	18.80 - 21.67
8	1	0.00 - 0.00	0.00 – 0.00	1.85 - 2.95	19.28 - 22.18
8	2	0.00 - 0.00	0.00 – 0.00	2.00 - 3.13	18.18 - 21.02
8	3	0.00 - 0.00	0.00 – 0.00	1.53 - 2.54	17.30 - 20.10
8	4	0.00 - 0.00	0.00 – 0.00	1.44 - 2.43	18.31 - 21.16
8	5	0.00 - 0.00	0.00 – 0.00	2.06 - 3.21	19.02 - 21.91
8	6	0.00 - 0.00	0.00 – 0.00	1.47 - 2.46	18.37 - 21.23
8	7	0.00 - 0.00	0.00 – 0.00	1.65 - 2.69	18.11 - 20.95
8	8	0.00 - 0.00	0.00 – 0.00	1.53 - 2.54	17.86 - 20.68

Number of PE Faults	Number of link Faults	Estimated Probability Confidence Interval ($Confidence\ level\ =\ 95\%$)			
		5×5	6×6	10×10	20×20
8	9	0.00 - 0.00	0.00 - 0.00	1.21 - 2.12	18.73 - 21.60
8	10	0.00 - 0.00	0.00 - 0.00	0.95 - 1.78	17.95 - 20.78
8	11	0.00 - 0.00	0.00 - 0.00	0.98 - 1.82	16.33 - 19.07
8	12	0.00 - 0.00	0.00 - 0.00	1.12 - 2.01	16.72 - 19.48
8	13	0.00 - 0.00	0.00 - 0.00	0.81 - 1.59	17.01 - 19.79
8	14	0.00 - 0.00	0.00 - 0.00	0.89 - 1.71	16.85 - 19.62
8	15	0.00 - 0.00	0.00 - 0.00	1.04 - 1.90	14.66 - 17.28
9	0	0.00 - 0.00	0.00 - 0.00	0.00 - 0.26	11.54 - 13.93
9	1	0.00 - 0.00	0.00 - 0.00	0.22 - 0.71	11.73 - 14.13
9	2	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	11.35 - 13.72
9	3	0.00 - 0.00	0.00 - 0.00	0.38 - 0.96	11.54 - 13.93
9	4	0.00 - 0.00	0.00 - 0.00	0.22 - 0.71	11.28 - 13.65
9	5	0.00 - 0.00	0.00 - 0.00	0.15 - 0.58	10.10 - 12.36
9	6	0.00 - 0.00	0.00 - 0.00	0.30 - 0.84	10.93 - 13.27
9	7	0.00 - 0.00	0.00 - 0.00	0.25 - 0.75	11.57 - 13.96
9	8	0.00 - 0.00	0.00 - 0.00	0.13 - 0.54	10.07 - 12.33
9	9	0.00 - 0.00	0.00 - 0.00	0.10 - 0.50	10.58 - 12.88
9	10	0.00 - 0.00	0.00 - 0.00	0.08 - 0.45	9.43 - 11.63
9	11	0.00 - 0.00	0.00 - 0.00	0.08 - 0.45	9.56 - 11.77
9	12	0.00 - 0.00	0.00 - 0.00	0.13 - 0.54	9.59 - 11.81
9	13	0.00 - 0.00	0.00 - 0.00	0.13 - 0.54	9.69 - 11.91
9	14	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10	8.93 - 11.07
9	15	0.00 - 0.00	0.00 - 0.00	0.17 - 0.63	9.24 - 11.42
10	0	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	6.81 - 8.72
10	1	0.00 - 0.00	0.00 - 0.00	0.00 - 0.16	5.68 - 7.45
10	2	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10	6.18 - 8.02
10	3	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10	6.49 - 8.37
10	4	0.00 - 0.00	0.00 - 0.00	0.00 - 0.26	6.31 - 8.16
10	5	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10	6.02 - 7.84
10	6	0.00 - 0.00	0.00 - 0.00	0.00 - 0.26	6.02 - 7.84
10	7	0.00 - 0.00	0.00 - 0.00	0.03 - 0.16	5.43 - 7.17
10	8	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	5.62 - 7.38
10	9	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	6.56 - 8.44
10	10	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	5.37 - 7.10
10	11	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10	5.34 - 7.06
10	12	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	6.53 - 8.41
10	13	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	5.46 - 7.20
10	14	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10	5.18 - 6.89
10	15	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	4.81 - 6.46

Number of PE Faults	Number of link Faults	Estimated Probability Confidence Interval ($Confidence\ level\ =\ 95\%$)			
		5×5	6×6	10×10	20×20
11	0	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	3.94 - 5.46
11	1	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.87 - 4.19
11	2	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.51 - 3.76
11	3	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	4.13 - 5.67
11	4	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.72 - 4.01
11	5	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	4.87 - 6.53
11	6	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.60 - 3.87
11	7	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	3.05 - 4.41
11	8	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.60 - 3.87
11	9	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.93 - 4.27
11	10	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.96 - 4.30
11	11	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	3.05 - 4.41
11	12	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	3.21 - 4.59
11	13	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.69 - 3.98
11	14	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.06 - 3.21
11	15	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.03 - 3.17
12	0	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	2.15 - 3.32
12	1	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.15 - 2.05
12	2	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.15 - 2.05
12	3	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.09 - 1.97
12	4	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.92 - 1.74
12	5	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.95 - 1.78
12	6	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.38 - 2.35
12	7	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.41 - 2.39
12	8	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.01 - 1.86
12	9	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.04 - 1.90
12	10	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.84 - 1.63
12	11	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.81 - 1.59
12	12	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.01 - 1.86
12	13	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.75 - 1.51
12	14	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	1.24 - 2.16
12	15	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.75 - 1.51
13	0	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.51 - 1.16
13	1	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.48 - 1.12
13	2	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.62 - 1.32
13	3	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.17 - 0.63
13	4	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.32 - 0.88
13	5	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.53 - 1.20
13	6	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.02 - 0.31
13	7	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.27 - 0.79
13	8	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.20 - 0.67
13	9	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.43 - 1.04

Number of *PE* Faults	Number of link Faults	Estimated Probability Confidence Interval ($Confidence\ level = 95\%$)			
		5×5	6×6	10×10	20×20
13	10	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.25 - 0.75
13	11	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.40 - 1.00
13	12	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.32 - 0.88
13	13	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.45 - 1.08
13	14	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.48 - 1.12
13	15	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.53 - 1.20
14	0	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.04 - 0.36
14	1	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.08 - 0.45
14	2	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.15 - 0.58
14	3	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.15 - 0.58
14	4	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.10 - 0.50
14	5	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.02 - 0.31
14	6	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00
14	7	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.17 - 0.63
14	8	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.15 - 0.58
14	9	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00
14	10	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.04 - 0.36
14	11	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.08 - 0.45
14	12	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00
14	13	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.06 - 0.41
14	14	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.10 - 0.50
14	15	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.10 - 0.50
15	0	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.04 - 0.36
15	1	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10
15	2	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.26
15	3	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.16
15	4	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10
15	5	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10
15	6	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.16
15	7	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10
15	8	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.26
15	9	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.10
15	10	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.16
15	11	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00
15	12	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00
15	13	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00
15	14	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00	0.00 - 0.00

Since the probability of occurrence of more number of faults than this is very small, the Table is truncated here.