Distributed Reconfiguration of Fault Tolerant VLSI Multipipeline Arrays with Constant Interstage Path Lengths

Hussain Al-Asaad, Mankuan Vai , and James Feldman

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115

Abstract

A new fault tolerant multipipeline array architecture and its diagnosis/reconfiguration algorithm will be presented. This multipipeline array design methodology is characterized by constant, fault distribution independent interstage path lengths. Other features include a low hardware overhead and a high survival rate when it is compared to existing approaches.

1: Introduction

The pipeline stages of a multipipeline array are separated from each other by interconnection networks as shown in Fig. 1. Interconnection networks ranging from simple feed-through connections to crossbar interconnections can be used. A balanced tradeoff between the degree of fault tolerance and the overhead should be located.

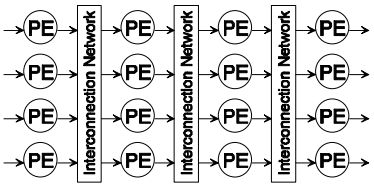

Fig. 1 A general model of multipipeline arrays.

The problems of reconfiguring functional pipelines out of an array with faults have received much attention [1-6]. Some reconfiguring algorithms assume fault-free switches and interconnections. A representative algorithm is described in [1]. This algorithm works in phases. Each phase consists of sequentially setting rows of switches. For an $N \times M$ multipipeline array, N sequential phases are required.

Other algorithms consider switch and interconnect faults. The algorithm proposed in [6] does not produce optimal results and is complex and difficult to be implemented distributively. Another algorithm based on finding the maximum flow in a flow network is presented in [2]. This algorithm, while being optimal, suffers from its complexity and difficulty in a distributed implementation

Another problem with known reconfiguring algorithms is that the interconnection lengths of a reconfigured array could become significantly longer than the original array.

2: New multipipeline array design methodology

The design methodology in this paper provides a fault distribution independent interconnection length between pipeline stages. Fig. 2 uses a 3×4 multipipeline array to show this new architecture. It is obvious that all interconnects are of equal lengths except for the wraparound ones.

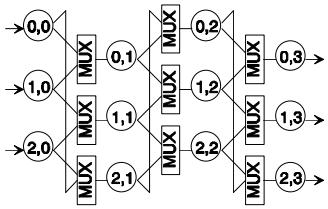

Fig. 2 The new multipipeline architecture illustrated for a 3×4 array.

A function that maps the PE's in a logical architecture into a corresponding physical array has been developed so that all the interconnections can be made into equal lengths by rearranging the PE's. Let (a, b) be the indices of a PE in the logical architecture and (x, y) be the PE indices in the physical implementation. The coordinates a and x are vertical indices ($0 \le a, x \le N - 1$) and b and y are horizontal indices ($0 \le b, y \le M - 1$). The following

mapping function guarantees constant length interconnections:

$$x = \begin{cases} 2a, & \text{if b is even, N is even, } a \le (N-1)/2, \\ 2a, & \text{if b is even, N is odd, } a \le (N-1)/2, \\ 2N-1-2a, & \text{if b is even, N is even, } a > (N-2)/2, \\ 2N-1-2a, & \text{if b is even, N is odd, } a > (N-1)/2, \\ 2a+1, & \text{if b is odd, N is even, } a \le (N-2)/2, \\ 2a+1, & \text{if b is odd, N is odd, } a \le (N-3)/2, \\ 2N-2a-2, & \text{if b is odd, N is even, } a > (N-2)/2, \\ 2N-2a-2, & \text{if b is odd, N is odd, } a > (N-3)/2, \end{cases}$$

and $y = b$. Fig. 3 demonstrates the mapping of the logical array in Fig. 2 into a physical array with constant length interconnects.

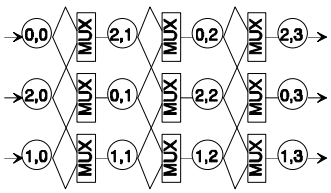

Fig. 3 The mapping of the array in Fig. 2.

The fault model considered here assumes that the reconfiguration control is fault-free due to its simplicity. Multipipeline diagnosis can be classified into being distributed or host driven.

The structure of a PE module designed for a distributed diagnosis is shown in Fig. 4. The multiplexer, self testing circuit, status flip-flop FS, and interconnections are assumed to be fault free. The status of each PE is determined by a self testing circuit.

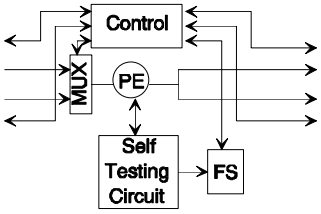

Fig. 4 The structure of a PE module.

The responsibility of error diagnosis can also be assigned to the host. With the help of a host, interconnection faults can also be detected. Diagnosis can thus be performed whether or not the assumption of fault-free interconnections is retained.

With the interconnections assumed to be fault free, the module shown in Fig. 4 can be modified to handle the host control of the status flip-flops by connecting all the status flip-flops in a column of PE's into a scan path. The host will set the status flip-flops according to the result of applying test vectors to the multipipeline, and activate the execution of the reconfiguration algorithm.

A bypassing path between the input and output of each PE can be provided as shown in Fig. 5 to test the interconnections. The contents of flip-flops FU and FD will be propagated to the upper and lower input PE's of the previous stage, respectively. A fault in this module will be represented by the values of FU and FD according to Table 1.

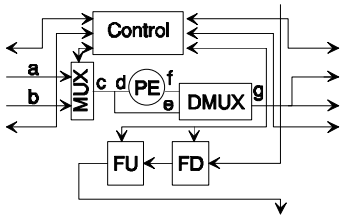

Fig. 5 The module structure modified for testing interconnections.

Table 1 Different types of faults indicated by the values of FU and FD.

Type of Fault	FU Value	FD Value
No fault	Good	Good
PE is faulty	Bad	Bad
MUX is faulty	Bad	Bad
DEMUX is faulty	Bad	Bad
Line a is faulty	Bad	Good
Line b is faulty	Good	Bad
Line c is faulty	Bad	Bad
Line d is faulty	Bad	Bad
Line e is faulty	Good	Good
Line f is faulty	Bad	Bad
Line g is faulty	Bad	Bad

The host will set the status flip-flops according to the result of applying interconnection and PE test vectors, and activate the execution of the reconfiguration algorithm.

The reconfiguration algorithm is implemented in the control part of a module. Each module communicates with its two neighbors from the previous stage and its two neighbors in the next stage. The control signals are illustrated in Fig. 6.

X receives two pipeline engagement requests *I_REQ_U* and *I_REQ_D* from *A* and *B*, respectively. It uses *O_ACK_U* and *O_ACK_D* to acknowledge one of these requests. *X* also receives the status of *C* and *D* by acknowledgment signals *I_ACK_U* and *I_ACK_D*, respectively. Based on the reconfiguration algorithm and its status, *X* will use *O_REQ_U* or *O_REQ_D* to request either *C* or *D* to be engaged in its pipeline.

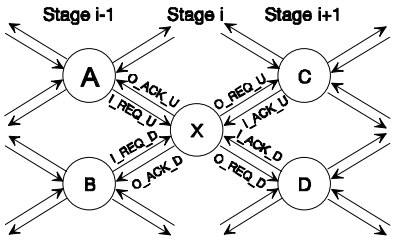

Fig. 6 Control signals between modules.

In the case of using a single status flip-flop, if *X* in stage *i* is faulty it will not acknowledge any of the requests from *A* and *B* in stage *i* - 1. Similarly if both acknowledgment signals from the two closest modules *C* and *D* of the next stage $i+1$ are negated, there will be no acknowledgement. On the other hand if *X* in stage *i* is healthy and at least one of the acknowledgment signals from *C* and *D* in stage *i* + 1 is asserted, a pipeline containing module *X* as its stage *i* can be formed. The module acknowledges one request, if it exists, to *A* or *B* in stage *i* - 1 with a preference assigned to the upper module. The module will also request one of *C* and *D* in stage *i* + 1 to be engaged in the pipeline with a preference given to the upper module.

Alternatively if two status flip-flops are used to represent the status of a PE, the reconfiguration algorithm will be modified slightly. If FU is set, then *X* in stage *i* cannot acknowledge the request from the upper module *A* in stage *i* - 1. If FD is set, then *X* cannot acknowledge the request from the lower module *B* in stage *i* - 1.

Two examples demonstrating these two algorithms are shown in Figs. 7a and b, respectively. The best solutions have been found in both examples.

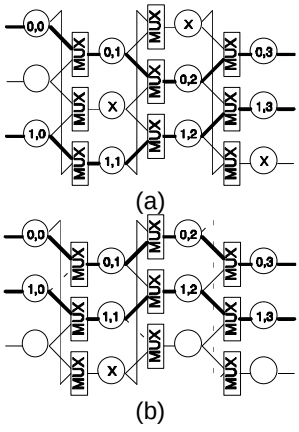

Fig. 7 Reconfiguration examples obtained with (a) the distributed algorithm and (b) the host-driven algorithm.

3: Evaluation

We use the common figures of merit: *simplicity*, *efficiency*, *area*, and *locality* (as described in [7]), as well as reliability, to compare this (HJM) with the multipipeline design (GUPTA) described by GUPTA *et al.* in [1]. A non-fault tolerant design and a design with crossbar interconnections between stages (MIN and MAX, respectively) are also included.

Both HJM and GUPTA are fast because of their hardware implementation, but HJM is simpler. HJM is a parallel distributed algorithm in which each PE performs the reconfiguration in parallel. GUPTA needs a sequence of reconfiguration phases.

A simulation is conducted for an 8×8 multipipeline to compare the efficiency of HJM to GUPTA. The expected number of recovered pipelines, normalized to the total number of pipelines supplied, is plotted as a function of *F* (the ratio of faulty PE's to all PE's) and shown in Fig. 8. It is concluded from this result that GUPTA has a better performance than HJM if $F > 0.1$. However, *F* is typically less than 0.1 in the case of a run time operation [8], so both algorithms have approximately equal performance for run time operations.

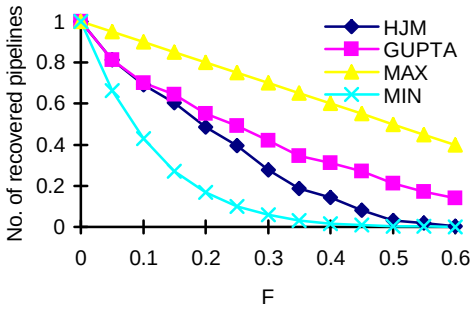

Fig. 8 The expected number (normalized) of survived pipelines.

HJM design uses simple multiplexers (2 T-gates) instead of the switches (10 T-gates) typically used. HJM does not need the control logic that controls the reconfiguration phases in GUPTA.

The biggest advantage of HJM over GUPTA is its locality. The interconnection length between any consecutive stages is always a constant in HJM.

The reliability of HJM is compared to that of GUPTA. The reliability is calculated using a Markovian model for an 8×8 multipipeline. A system failure occurs when the number of working pipelines dropped below a certain number S_m. The reliability of a multipipeline is defined as follows:

$$R(t) = \text{Prob}\{S(t) \geq S_m\},$$

where $S(t)$ is the number of survived pipelines at time t. The results of a simulation are provided in Fig. 9, which show that HJM has a reliability comparable to GUPTA. Simulations are also performed by varying M, N, and P (failure probability of a PE). The results are shown in Fig. 10.

To emphasize on the fact that the yield rate is approximately constant, the yield is plotted for $M = 8$ and different values of N and P and shown in Fig. 11. Each curve in Fig. 11 corresponds to a different set of N and P and can be used as a design guideline for fault tolerant multipipelines.

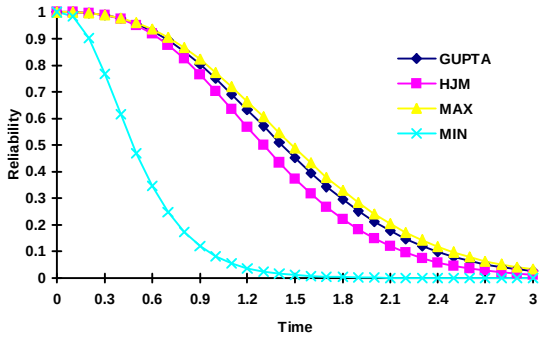

Fig. 9 The reliability of an 8×8 multipipeline with a PE failure rate of 0.1 failure per unit time and $S_M = 6$.

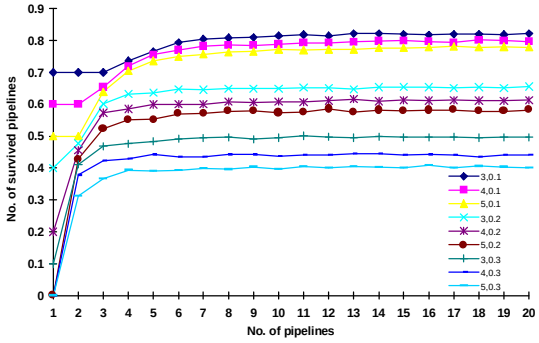

Fig. 10 The normalized number of survived pipelines for HJM.

4: Conclusion

A new design for fault-tolerant multipipelines was developed. This design is simpler than other designs described in the literature. A special advantage of the new design is that it guarantees uniform interconnect lengths that are independent of the fault distribution. Other features include a less overhead, a comparable efficiency, and a good reliability.

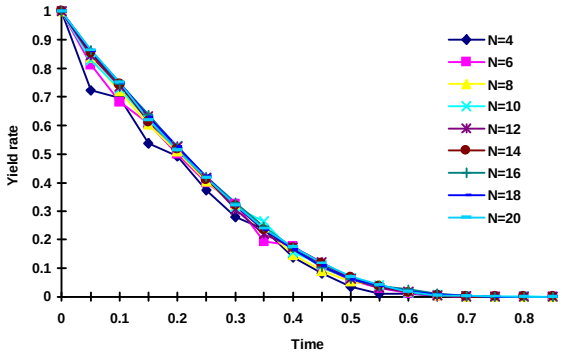

Fig. 11 The expected yield of an $N \times 8$ multipipeline.

References

[1] R. Gupta, A. Zorat, I. Ramakrishnan, "Reconfigurable Multipipelines for Vector Supercomputers," *IEEE Transactions on Computers*, Vol. 38, No. 9, September 1989, pp. 1297-1307.

[2] H. Lin, F. Lombardi, M. Lu, "On the Optimal Reconfiguration of Multipipeline Arrays in the Presence of Faulty Processing and Switching Elements," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 1, No. 1, March 1993, pp. 76-79.

[3] P. Koo, F. Lombardi, Y. Shen, "Approach for the Reconfiguration of Multipipeline Arrays," *IEE Proceedings-E*, Vol. 138, No. 3, May 1991, pp. 131-137.

[4] Y. Choi, "Reconfigurable VLSI/WSI Multipipelines," *Parallel Computing*, Vol. 17, 1991, pp. 941-952.

[5] Y. Choi, "Fault Diagnosis of Reconfigurable Multipipelines Using Boundary Scans," *Computers & Electrical Engineering*, Vol. 18, No. 2, 1992, pp. 119-130.

[6] Y. Choi, "Reconfigurable Multipipelines," *International Conference on Parallel Processing*, 1991, pp. 556-570.

[7] M. Chean, J. Fortes, "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," *IEEE computer*, January 1990, pp 55-69.

[8] B. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Welsley Publishing Company, Reading, MA., 1989.