Polynomial Testing of Packet Switching Networks

JYH-CHARN LIU, STUDENT MEMBER, IEEE, AND KANG G. SHIN, SENIOR MEMBER, IEEE

Abstract—**A functional testing method called *polynomial testing* is proposed to test packet switching networks (PSN's) used in multiprocessor systems. For the purpose of concreteness, we focus on applying the method to packet switching multistage interconnection networks (PMIN's). A multiple stuck-at (MSA) fault model is developed first, and then faults are diagnosed at two different levels: *network level* and *switch level*. The former uses each processor as a tester and can test part of the network concurrently with the normal operations on the remaining part of the network. On the other hand, the latter uses switches in the network as testers and is inherently an autonomous testing method. To facilitate the network level testing, the routing dynamic in a PMIN is eliminated by synchronizing switch operations. The network is then decomposed into *routes*, each of which is tested after transforming it into a polynomial calculator. For switch level testing, a built-in tester (BIT) is embedded into each switch's structure to provide self-testing capabilities. Network level testing is distributed and suitable for concurrent testing, whereas switch level testing is off-line with a small testing time.**

Index Terms—**Built-in tester, concurrent testing, linear feedback shift register, multistage interconnection network, packet switching, polynomial generator, polynomial testing, stuck-at routing fault, switch self-testing.**

I. INTRODUCTION

DESPITE the continuing improvement in semiconductor device speed, use of multiple processors and memories is an attractive alternative to meet ever-increasing needs of computing speed and reliability. Interconnection networks are one of the most important components of such multiprocessor systems and are made feasible by the advancement in VLSI technology. Since VLSI technology greatly degrades testability, an interconnection network must have a structure that is easily testable.

There are two well-known switching methods for interconnection networks: *circuit switching* and *packet switching*. To distinguish these two methods, a *path* and a *route* for a source–destination pair are defined as follows. A path is a physically-established communication medium between the source and destination to transfer a request/data. A route is a logical path which can transfer a request from a source to its destination without total dedication to it; resources on a route are time-shared among several packets. In a circuit switching network (CSN), the path from a source to its destination is physically set up *a priori* and dedicated to a request until the request is completely serviced. By contrast, no complete physical path is established *a priori* for a request in a packet switching network (PSN). A packet switching multistage interconnection network (PMIN) is composed of a large number of links and switches with buffers. Each PMIN switch is essentially an $r \times r$ crossbar, in which a queue is placed at each input port to store packets. A request/message is decomposed into several packets, each of which is independently transferred through an available route.

Many PSN's have an undesirable effect called the *routing dynamic*: the order of arrival of packets at the destination may be different from the order of their transmission from the source. Although PMIN's can be designed not to have the routing dynamic, the routing dynamic will be considered in our testing method to provide better versatility. Clearly, a PMIN with the routing dynamic is an asynchronous sequential machine. Although a sequential machine can be fully tested with a checking sequence derived from its state-transition table [1], no feasible checking sequence seems to be derivable for large scale asynchronous sequential machines like PMIN's. Functional testing is an alternative to prove the correctness of some of the machine's functions within a finite time period.

Several researchers have proposed functional testing procedures for specific networks. Error control codes are popular for on-line fault detection [2], [3]. A comprehensive method for diagnosing the baseline CSN's with 2×2 switches was introduced by Feng and Wu [4]. A simplified version of the fault model in [4] and the corresponding testing strategy can be found in [5]. Davis *et al.* proposed some fault location techniques for distributed routing control networks [6]. Lee and Shen modeled a CSN using 2×2 switches as an ILA [7]. Low-order switches, e.g., 2×2 switches, can be completely tested with a constant number of patterns. Agrawal and Leu used the dynamic full accessibility of MIN's to test their connectivity [8]. Several high-level testing strategies for general PMIN's have also been studied [9]–[15], most of which are adaptive procedures requiring human assistance.

Most existing methods are centralized and off-line, i.e., the whole network is tested off-line by *one* tester. Since there are $N/r \log_r N$ switches in a PMIN, the complexity of the network testing problem is $O(N/r \log_r N)$. Centralized testing methods are usually very inefficient for large networks, because the problem to be handled by the tester grows exponentially with the size of the network. To improve testing efficiency, we

Manuscript received August 31, 1986; revised February 28, 1987. This work was supported in part by the Office of Naval Research under Contract N00014-85-K-0122 and NASA under Grants NAG-1-296 and NAG-1-492. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

The authors are with the Real Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8824535.

propose a two-level testing strategy: *network level* and *switch level testing*.[1] In the network level, every processor can serve as a tester to test part of the network; thus, there are N testers for the network. Assuming that testers are homogeneous, the complexity of the testing problem in each tester is reduced to $O(1/r \log_r N)$. In the switch level testing, switches are used as testers and designed to have autonomous testing capability [17]. In other words, the complexity of the testing problem in each tester is independent of the network size and is fixed. The characteristics of the network level testing are that it can test the network concurrently, but may have a lower fault coverage than the switch level testing. On the other hand, the switch level testing is an off-line method with a small testing time but has high fault coverage.

The network level testing is based on the topology and functions of the network, because processors must cooperate to test the network. To eliminate the routing dynamic, network operations are first synchronized. Then, an $N \times N$ blocking network is decomposed into N^2 routes, $NT = \{\mathrm{RT}_{ij} | 1 \le i, j \le N\}$, where RT_{ij} is the route from source i to destination j. RUT_{ij} is the route RT_{ij} under test, and the testing processors are the processors connected to the route under test (RUT). In the network level testing, faults in RUT_{ij} are tested without stopping the normal operations on $NT - \{\mathrm{RUT}_{ij}\}$, where $\mathrm{RUT}_{ij} \in NT$ and $\{\mathrm{RUT}_{ij}\} \neq NT$. To test a route without interrupting, or being interrupted by, normal operations, the testing processors should be able to lock/unlock the RUT. A RUT can be locked by activating the busy signals of the switches on the RUT. Locking a route prevents unexpected packets from entering the route. As shown in Fig. 1, a route can be viewed as a cascaded shift register array. The register array can then be easily modified into *divisors, multipliers,* or other similar structures for *polynomial testing*.

In the switch level testing, each switch is a tester and switches are assumed to be homogeneous. Thus, the logic structure, instead of its topology, of the network is the main concern. To obtain high fault coverage with a small testing time, each switch is designed to have self-testing capabilities. A switch is composed of buffers, a routing control unit (RCU), and output ports consisting of multiplexers-demultiplexers (MUDEX's). Since thorough testing of the RCU may require an intractable testing length, an on-line checker is proposed to detect malfunctions in the RCU. For the rest of the network, queues are first self-tested by polynomial generation and comparison. If the queues are fault-free, they are then used to generate test patterns for links and MUDEX's. The testing responses of switches at one stage are verified at the next stage.

The rest of this paper consists of four sections. Section II gives a brief review of the polynomial operations necessary for our testing method. Section III introduces the network fault models which are to be tested with operations on polynomials. Testable designs and the corresponding network and switch level testing are presented in Section IV. The paper concludes with Section V.

[1] This term should not be confused with the switch-level fault model for MOS circuits [16].

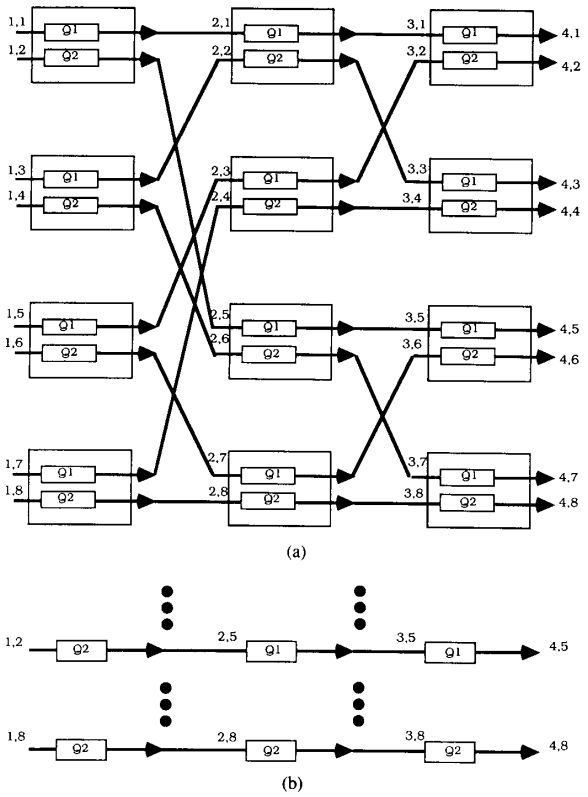


Fig. 1. A baseline PMIN with switch permutation E_0 and the corresponding cascaded shift register arrays. (a) A baseline PMIN with switch permutation E_0. (b) The corresponding cascaded shift register array of the PMIN.

II. Principles of Polynomial Testing

Basic polynomial operations and their implementations are briefly discussed below. Use of the polynomial ring $GF(2)[x]$ is well-known for error control codes [18]. Only those properties useful for testing PSN's will be introduced below for completeness.

Definition 1: A polynomial $P_b(x) = \sum_{i=0}^{n} b_i x^i$ in $GF(2)[x]$ is said to be a *bit polynomial* if each of its coefficients is a bit, i.e., $b_i \in \{0, 1\}$, $\forall 0 \le i \le n$. A *word polynomial* is the one whose coefficients are words instead of bits, i.e., $P_w(x) = \sum_{i=0}^{n} w_i x^i$, where for every $i \in I_n = \{0, 1, \cdots, n\}$, w_i = ONE or ZERO, and ONE is a b-bit vector of *arbitrary* pattern and ZERO = $\overline{\mathrm{ONE}}$, i.e., ZERO is bitwise complemented to ONE. Thus, any two words with maximum Hamming distance can be used as ONE and ZERO, respectively.

For notational convenience, let $W_n(x)$ denote a polynomial $\sum_{i=0}^{n} c_i x^i$, c_i = 1 or ONE, $\forall i \in I_n$. $\bar{P}(x) = \sum_{i=0}^{n} \bar{c}_i x^i = P(x) \oplus W_n(x)$ is the complement of $P(x)$, and the symbol "$\oplus$" represents the addition in $GF(2)$. Unless otherwise specified, we will use the term "polynomial" to represent both bit and word polynomials. The mechanisms to manipulate polynomials are called their *calculators*. The contents of a calculator before operating on its input are called the *initial state*, which

will always be assumed, for clarity of presentation, to be all zeros. A calculator with the zero initial state is called an *inert linear machine* [19]. When a word polynomial operation is applied to a faulty circuit, the closure property of $GF(2)[x]$ may not hold. However, when a word polynomial is applied to a nonfaulty circuit, the resulting polynomial belongs to $GF(2)[x]$. Calculators are more hardware efficient if ONE and ZERO are composed of all 1's and 0's, respectively, because for each operation every bit will require an identical circuit.

A. Operations on Polynomials

A *periodic* polynomial with period p is the series $\Sigma_{i=1}^{\infty} c_i x^i$ where $c_i = c_{i+p}$, $\forall i \in I$, and I is the set of integers. It can be generated by a linear (or nonlinear) feedback shift register (LFSR) called a *polynomial generator* (PG). Registers in a PG can be implemented by different types of flip-flops, and apparently different test patterns are needed for different implementations. However, as shown in Appendix A, at most two inputs are needed to detect faults in a master-slave SR flip-flop. Since the network level testing deals with the network topology, we will consider only the input and output stuck-at faults of registers, i.e., not the stuck-at faults inside registers. However, the same test patterns can test all the faults in those registers implemented with the master-slave flip-flops shown in Appendix A.

Two polynomials $P_1(x) = \Sigma_{i=0}^{n} c_{1,i}x^i$ and $P_2(x) = \Sigma_{i=0}^{n} c_{2,i}x^i$, are *equal* iff $c_{1,i} = c_{2,i}$, $\forall i \in I_n$. Two polynomials can be compared for equality by XOR gates. The following operations are useful for our discussion.

Addition and Boolean: Let $\{P_j(x) = \Sigma_{i=0}^{n} c_{j,i}x^i\}_{j=1}^{k}$ be k polynomials in $GF(2)[x]$. $P_3(x)$ is the *addition* of $P_1(x)$ and $P_2(x)$, denoted by $P_3(x) = \Sigma_{i=0}^{n} c_{3,i}x^i = P_1(x) \oplus P_2(x)$, if for each $i \in I_n$ $c_{3,i} = c_{1,i} \oplus c_{2,i}$. Addition can be implemented with XOR gates. If a Boolean operation Δ is applied to $P_1(x)$, $P_2(x)$, $\cdots$ $P_k(x)$, the resulting polynomial $P(x) = \Sigma_{i=0}^{n} c_i x^i$ is calculated by $c_i = c_i^1 \Delta c_i^2 \Delta \cdots \Delta c_i^k$, $\forall i \in I_n$, and Δ is a bitwise operation when c_i is a word. Only AND and OR, the two most important operations, will be considered in this paper.

Division and Multiplication: Given $P(x) = \Sigma_{i=0}^{n} p_i x^i$ and $M(x) = \Sigma_{i=0}^{n} m_i x^i$ in $GF(2)[x]$, the *multiplication* of $M(x)$ (multiplier) to $P(x)$ (multiplicand) is $P_3(x) = P(x)M(x) = \Sigma_{i=0}^{n} p_{3,i}x^i$, where $\forall i \in I_n$, $p_{3,i} = p_i m_0 \oplus p_{i-1}m_1 \oplus \cdots \oplus p_1 m_{i-1} \oplus p_0 m_i$. On the other hand, given two polynomials $P(x)$ (dividend) and $D(x) = \Sigma_{i=0}^{r} d_i x^i \neq 0$ (divisor) in $GF(2)[x]$ there exist two polynomials $Q(x)$ and $R(x)$ in $GF(2)[x]$ such that $P(x) = D(x)Q(x) + R(x)$ where $R(x) = 0$ or $\deg R(x) < \deg D(x)$. In this process, $P(x)$ is said to be *divided* by $D(x)$, yielding a quotient $Q(x)$ and a remainder $R(x)$.

A *bit divisor* (*multiplier*) divides (multiplies) an input stream by a fixed bit polynomial. Similarly, a *word divisor* (*multiplier*) performs divisions (multiplications) between two word polynomials. In a word polynomial divisor/multiplier (PDM), operands are ONE or ZERO instead of 1 or 0. It has logic operations similar to those of a bit PDM, but special mechanisms are necessary to preserve the properties of the polynomial ring.

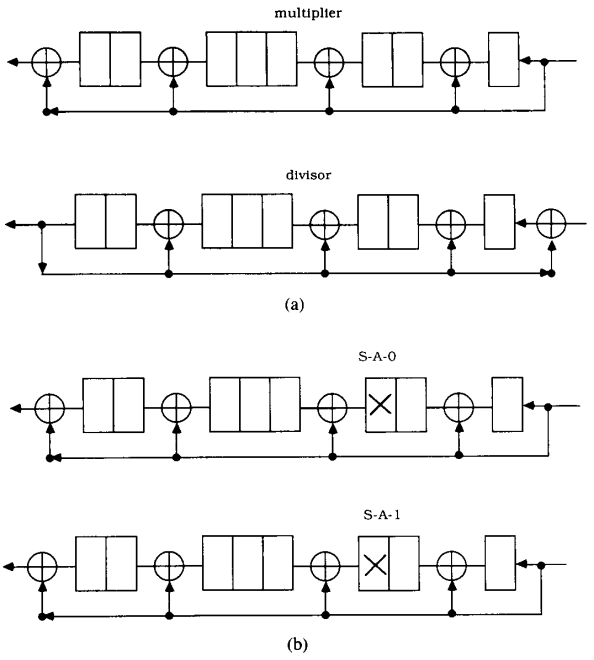


Fig. 2. The structure of faulty and nonfaulty multipliers and divisors. (a) Normal multiplier and divisor. (b) Two faulty multipliers.

The final contents of a PDM will henceforth be represented by $R(x)$, the input stream will be represented by $P(x)$, and the output stream by $Q(x)$. The general structures of a bit divisor and a bit multiplier are shown in Fig. 2(a). $M(x)$ and $D(x)$ in Fig. 2(a) are $1 + x^2 + x^5 + x^7 + x^8$ and $x^8 + x^6 + x^3 + x + 1$, respectively. The lowest order position is located in the input (output) port of the divisor (multiplier). There is an XOR gate, denoted by $\oplus$, at the D-type flip-flop's (DFF's) output of stage i only when m_i or d_i is 1. A *block* B_i is the collection of DFF's between the $(i-1)$th and ith XOR gates, counting from the lowest order position, in a PDM. Thus, a PDM is composed of a set of blocks $\{B_i\}$. Let the order (the number of stages) of B_i be r_i. In the multiplier of Fig. 2(a), $r_1 = 2$, $r_2 = 3$, $r_3 = 2$, and $r_4 = 1$.

Since a RUT is to be transformed into a polynomial calculator for testing, the effects of DFF's multiple stuck-at (MSA) faults on a PDM are discussed as follows. An MSA fault f_M in a block B_i is composed of multiple single stuck-at (SSA) faults, i.e., $f_M = \{f_s^i\}$, where f_s^i is an SSA fault in B_i. Let k be the faulty position nearest to the output port of B_i. Then, $f_s^k \in f_M$ will block the effects of all the other SSA faults in f_M. Such an f_s^k is called the *leading* SSA fault in B_i. There are $2r_i$ possible leading faults in B_i, and, thus, there are $2r_i$ distinguishable stuck-at faults in the block, where r_i is the number of stages in B_i.

An s-a-0 FL changes the attached XOR gates into null operators. Thus, for a multiplier, $M'(x) = \Sigma_{i=1}^{k} m_i' x^i$, where $m_i' = 0$ if the FL attached to the XOR gate at x^i is stuck at 0, and $m_i' = m_i$ otherwise. It is shown in Lemma 1 that multiple s-a-0 at the FL-inputs of XOR gates can be tested by the impulse

polynomial. On the other hand, when the FL is s-a-1, $M'(x)$ becomes $\Sigma_{i=1, i\neq m_f}^{k} m_i x^i + \Sigma_{\{m_f\}} x^{m_f} W(x)$, where m_f is the location of an XOR gate whose input from the FL is fixed at 1 due to an s-a-1 fault. Thus, when an all zero input stream, i.e., $\Sigma_{i=1}^{k} 0x^i$, is applied to the PDM, the locations of those XOR gates affected by the FL s-a-1 faults can be uniquely determined by the corresponding output stream.

Lemma 1 [19]: The impulse response of a multiplier is $m_0 m_1 \cdots m_n 0 \cdots 0$, where the impulse polynomial is $P_I(x) = 10 \cdots 0$.

Clearly, an unknown multiplier can be uniquely identified by its impulse response, and the multiplication of $P_I(x)$ to $M(x)$ can be viewed as a discrete convolution between them.

Lemma 2: When a PDM is an inert machine and an s-a-0 fault occurs in B_k, the multiplier $M(x) = \Sigma_{i=0}^{n} m_i x^i$ is changed to a new multiplier $M'(x) = \Sigma_{i=0}^{k-1} m_i x^i$.

Lemma 3: Let l be the number of fault-free DFF's between B_i's output and the leading faulty DFF. Then, when the leading faulty DFF^l in B_k is s-a-1, $Q(x) = M'(x)P(x) \oplus x^{r'} W_n(x)$, where $r' = l + \Sigma_{j<k} r_j$ and $M'(x)$ is the new multiplier whose highest order position is located at the leading faulty DFF in B_k.

Proof: Let the input (or output) of B_i be I_i (or O_i). Then we have $I_{i-1} = O_i \oplus P(x)$ and $O_{i-1} = x^{r_{i-1}} I_{i-1}$. When DFF^l is s-a-1, $O_k = W_n(x)$. Thus, $I_{k-1} = P(x) \oplus x^l W_n(x)$ and $O_{k-1} = x^{r_{k-1}}(P(x) + x^l W_n(x))$. By induction, we can show that $Q(x) = M'(x)P(x) \oplus x^{r'} W_n(x)$. ■

When x^k of a divisor is s-a-1, the output $Q(x) = P'(x)/D'(x)$, where $P'(x) = \{x^l (\Sigma_{i=k+1}^{r} n_i x^{i-k}) \oplus W_l(x)\}$, $\Sigma_{i=k+1}^{r} n_i x^{i-k}$ is the initial state of the divisor and l is the polynomial length that is sufficient for testing, and $D'(x)$ is the new divisor with its lowest order position at the output of B_k. Similarly, an s-a-0 fault at x^k makes $Q(x)$ periodic, i.e., $Q(x) = x^l (\Sigma_{i=k+1}^{r} n_i x^{i-k})/D'(x)$, where the degree of the faulty divisor is $r_d = \Sigma_{j>k} r_j$. Note that the output $Q(x)$ is independent of the input stream. The structures of s-a-0 and s-a-1 multipliers are shown in Fig. 2(b). The resulting $M_0'(x)$ (for s-a-0) and M_1' (for s-a-1) are $1 + x^2 + x^5$ and $M_0' \oplus x^r W(x)$, respectively.

For testing purposes, it is assumed that every DFF on a route can be simultaneously set to ZERO by an external signal. Signature analysis examines $R(x)$ after the testing polynomial $P(x)$ is applied to a circuit under test. The final contents of each DFF must be directly read out for signature analysis. Unfortunately, this will greatly increase the number of I/O terminals of a network. Thus, signature analysis or other similar methods requiring direct access to DFF's are not followed here and interested readers are referred to other articles, such as [20].

The proposed network level testing is to diagnose the network by appropriate operations on the output stream. After the testing polynomial $P(x)$ is applied to a RUT, a fault f_i changes $Q(x)$ into $Q_i(x)$, where $Q(x)$ [or $Q_i(x)$] is the correct (or faulty) output polynomial of the RUT. The procedure is then to find a testing polynomial $P(x)$ and an operation Θ_{f_i} such that $\Theta_{f_i}(P(x), Q(x)) = Q_i(x)$. The combination of $P(x)$, its output $Q(x)$, and the operation Θ_{f_i} is called a *testing routine* for the fault f_i.

III. Fault Models

A PSN is composed of links and switches. There are $r!$ possible interconnection patterns within an $r \times r$ switch. There are then $(r!)^{(N/r)\log_r N}$ different conflict-free interconnection patterns in an $N \times N$ PMIN. Links' stuck-at faults are equivalent to stuck-at faults of the switches to which they are attached. Thus, only switch faults are considered for the network level testing. That is, link stuck-at faults are implicitly included in the switch fault models.

Permanent *multiple stuck-at, delay, partial setting, blocking, merging, broadcasting*, and *misrouting* faults are all considered in this paper. An MSA fault occurs when one or more signal lines are fixed at 0 or 1. A delay fault occurs when the operation speed of some component(s) is slower than the specified and, thus, erroneous operations result. A partial setting fault occurs when some of the identical components in a unit do not provide the same operation as the others. A blocking fault occurs when an appropriate route within a switch cannot be established for a request. A handshake signal deadlock is an example of blocking fault. A switch has a merging (broadcasting) fault when two or more input (output) ports are connected to one output (input) port. A misrouting fault represents the case when packets are misdirected to incorrect output ports. Stuck-line faults at gate level are tested at the switch level testing.

IV. PMIN Diagnosis

As mentioned earlier, our testing strategy is divided into two levels: network and switch levels. At each of these two levels, we present testable designs and testing methods on the basis of the polynomial operations and the fault models introduced in Sections II and III, respectively. The network is designed such that all signal lines have only two states, i.e., 1 or 0, whether or not they are used to transfer data. The output port of a switch is a combination of multiplexers and demultiplexers (MUDEX's). A MUDEX is basically composed of AND and OR gates. When multiple requests are assigned to an output port, a combination of OR/AND functions among the requests will take place.

A. Network Level Diagnosis

Assume that the PMIN under test connects N sources and N destinations and is built with $r \times r$ switches. The number of stages in the PMIN is $k \equiv \log_r N$. To describe the PMIN's topology and permutation, the input (output) ports of all switches in each stage are vertically indexed. The number assigned to an input (output) port is called its *global index*. For each $r \times r$ switch, there is a one-to-one correspondence between the global index and the input/output port number: $f_i(j) = m$, where j is the port number of the ith switch at a stage, and m is the port's global index. A *link permutation* T_i, $1 \le i \le k$, is a one-to-one mapping from the output ports at stage $i-1$ to the input ports at stage i. On the other hand, a *switch permutation* $E_m^i: f_i(j) \rightarrow f_i((j + m) \text{ MOD } r)$ is a one-to-one mapping from input ports of a switch to its output ports, $0 \le m \le r - 1$. For simplicity, all the switches on the RUT are assumed to have an identical permutation, i.e., $i_1 =$

i_2 for all $E^{i_1}_m$, $E^{i_2}_m \in$ RUT, and E_m will henceforth be used to denote E^i_m. More general cases than this can be easily derived by using the actual permutation at each stage. To allow for simultaneous diagnosis and normal operation at the network level, the testing processors should be equipped with complete information of link and switch permutations.

1) Testable Design: Links are passive components and can be treated as data paths of switches, whereas switches make all switching decisions and also contain memory elements. To make the network easily testable, switches are designed to have two operational modes: *normal* and *testing* modes.

As mentioned in the Introduction, a RUT can be viewed as a cascaded shift register array. FL and XOR gates must be added to transform a 1-bit wide RUT into a bit PDM. Since links are the predominating cost factor of a PMIN, the link overhead in improving testability must be kept as small as possible. A tracer in each switch is thus proposed to minimize the width of FL. A tracer is composed of a testing pattern masker and mapper, a feedback/feedforward selector (F-selector) and a modulo TWO adder, where TWO = {ONE, ZERO}. The masker examines if bits of the testing pattern are identical and maps the testing pattern from ONE (ZERO) to 1(0) for FL. The mapper transforms 1(0) to ONE (ZERO) to use the adder. The F-selector determines the transmission direction of FL.[2] An adder is necessary for each switch to form a block on a route for data path diagnosis.

Four possible operational states, S, A, X, and N, are assigned to a switch when the network is being tested. Once a switch in a RUT is in state S, the switch will not allow any packets, except those from the same RUT, to enter the RUT, and the operations of switches on the route are synchronized. State S can be taken as a suboperation of the other states, because the tracer in the other states is activated *and* switch operations are synchronized. When the switch is in state N, only FL and the F-selector are activated. When a switch at stage i is in state A, the F-selector blocks the FL signals from stage $i + 1$, and the current switch's output is led to FL. When the switch is in state X, the data on FL are mapped, by the mapper, from 1 (0) to ONE (ZERO), and the logic operation $BU_0 \leftarrow P_{in} \oplus FL$ is performed at the input of the queue, where BU_0 is the input of the queue and P_{in} is the input packet. Fig. 3 shows these switch operations in different states. The logic diagram in Fig. 4 shows a switch design example of the network level testing.

A switch can enter/exit the testing mode by command packets. Two formats, *data packets* and *command packets*, are used to control the switch operations. A command packet is composed of routing tags and a command array $\{\mathbf{CA}(1), \cdots, \mathbf{CA}(k)\}$, where k is the number of stages of the network and $\mathbf{CA}(i)$ is a 2-bit command word associated with stage i. A switch at stage i will enter states S, A, N, and X, when $\mathbf{CA}(i) = 00$, 11, 10, and 01, respectively. The type of packets can be identified by a one-bit flag in each packet. As shown below, this testing method can also identify a misinterpreted command array (by a faulty switch).

[2] The F-selector can be eliminated if the RUT is to be transformed into either a multiplier or divisor, but not both.

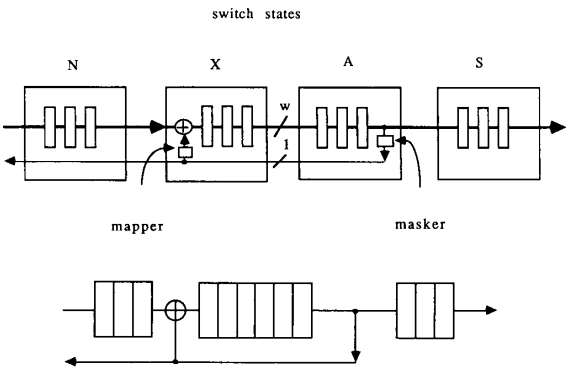


Fig. 3. Switches on a RUT and the corresponding word divisor.

Theorem 1: All misinterpreted command packets can be tested in one testing routine.

Proof: Once a RUT is transformed into a multiplier, the test pattern for misinterpreted command packets becomes an impulse polynomial. From Lemma 1, $M(x)$ of the RUT can be uniquely identified. ■

2) Data Path Stuck-at Faults: All switches are in state X when data path stuck-at faults are being tested. An SSA fault at the network level represents a stuck-at fault(s) in a *single* switch. But an MSA fault at the network level implies stuck-at faults in more than one switch. In a conventional approach, upon detection of a fault on some route, test patterns must be submitted from processors on different routes to locate the fault. It is shown below that the fault location with the polynomial testing is much easier than that with the conventional approach.

SSA Faults: Every switch is set to an identical permutation. When $r \times r$ switches are used, r different switch permutations $\{E_i | 0 \le i \le r - 1\}$ are necessary to test every data path within a switch. For any input port of a switch, its data paths to all the output ports are included in $\{E_i | 0 \le i \le r - 1\}$. Thus, in these r permutations every data path from each input port to every output port is tested.[3] The procedure can be generalized as follows: in testing routine m, the switch permutation E_m, $0 \le m \le r - 1$, is performed first. Then, the connection of source i to destination j is specified by $j = T_k E_m T_{k-1} E_m \cdots T_2 E_m T_1(i)$. The special case of $r = 2$ allows data path stuck-at faults to be detected in two permutations, each of which is composed of two steps [4].

Theorem 2: When a locked RUT_{ij} is configured as a multiplier, an SSA fault on the data path can be located by processor j in one testing routine.

Proof: The testing polynomial for the data path SSA fault is $W_n(x)$, where n is the total length of buffers on RUT_{ij}. As discussed earlier, RUT_{ij} can be expressed as $M_{ij}(x) = \sum_{i=0}^{n} m_i x^i$. The output at the destination j becomes $Q(x) = \sum_{l=0}^{n} m_l x^l W_n(x)$. $Q(x)$ should then have the format of $1\cdots10\cdots01\cdots$, where a 0 (1) $\rightarrow$ 1 (0) transition takes place

[3] Only r permutations are needed to test a data path, although $r!$ permutations are required to test the routing functions.

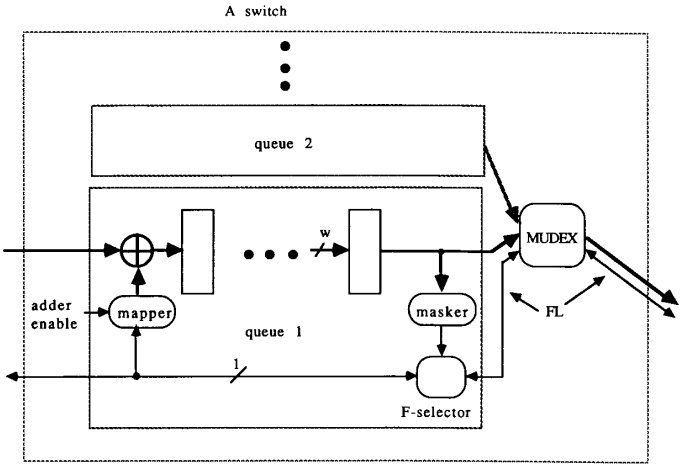


Fig. 4. A testable design of switches for concurrent testing.

at each position of an XOR gate on RUT_{ij} and the number of consecutive 1's (0's) in the ith block is the size of B_i. For example, the output stream of the multiplier in Fig. 2(a) is 10011100. When M_{ij} changes to $M'_{ij} \neq M_{ij}$ due to an SSA fault, there must be at least one i such that $m_i \neq m'_i$, $1 \leq i \leq k$, by Lemmas 2 and 3. When the number of $0 \rightarrow 1$ transitions is m_f, the faulty switch can be located by $s_f = (\Pi_{i=0}^{m_f} E^{-1} T_{m_f-i}^{-1})(j)$, where T_i^{-1} is the inverse of permutation T_i. ■

MSA Faults: An MSA fault on a data path cannot be determined in one testing routine. However, the polynomial testing can be applied to a sequential repairing procedure which locates and then replaces leading faulty switches/links in each testing routine.

Theorem 3: An MSA fault on a data path can be repaired in k testing routines, where k is the number of stages of the network.

Proof: An MSA fault is the collection of multiple SSA faults. When the testing polynomial $W_n(x)$ is applied to a PDM, $Q(x)$ is uniquely determined by the type (s-a-0 or s-a-1) and the location of the leading stuck-at fault. In other words, the lowest order faulty switch can be located in each testing routine, regardless of the cardinality of the multiple fault. Since there are k switches on a route, at most k steps are required to repair the network. ■

Delay Faults: A delay fault on a data path is detectable when its operational speed is at least one clock cycle slower than specified.

Theorem 4: A single delay fault of longer than one clock cycle can be located in one testing routine.

Proof: The polynomial $P_E^1(x) = \Sigma_{i=0}^{k} x^{2i}$ can detect all delay faults. However, a polynomial $P_E^m(x) = \Sigma_{i=0}^{k} x^{(m+1)i}$ can be used to distinguish a $1 \rightarrow 0$ transition delay fault of m clock cycles from delay faults of less than m cycles. When an m unit delay fault occurs and $P_E^m(x)$ is applied, the faulty switch's output becomes $W(x)$. By forming a PDM on RUT_{ij}, a delay fault can be located in one testing routine. A testing polynomial for $0 \rightarrow 1$ delay transitions is complemented to become $P_E^m(x)$ and the output is $\bar{W}(x)$. ■

Like MSA faults, a multiple delay fault composed of different delay lengths can be repaired in k testing routines.

3) Routing Faults: Methods for locating routing faults are studied in this subsection. Switches are set to state S when routing functions are tested.

Merging and Broadcasting Faults: Depending on the implementation details, a merging fault can be located in one testing routine when appropriate polynomials are applied. A Δ-*merging* fault occurs when a Δ (i.e., AND or OR) operation results from the merging of two or more switch input/output ports.

Consider the effect of the OR merging first. For two routes RUT_{i_1} and RUT_{i_2}, they will topologically intersect in at most one switch when the network is not redundant.

Theorem 5: For a given permutation, a multiple OR-merging fault can be located in one testing routine for both distributed and centralized routing control PMIN's.

Proof: The testing polynomial at processor j is $P_j^N(x) = \Sigma_{i=1}^{N} c_i x^i$, where c_j = ONE and c_i = ZERO, $\forall i \neq j$. First, consider the case when two RUT's are merged. The two routes from i_1 and i_2 under the given permutation intersect at most once. When the intersecting switch has an OR-merging fault, and the testing polynomials $P_{i_1}^N(x)$ and $P_{i_2}^N(x)$ are applied, there will be an OR operation between these two polynomials. Without loss of generality, $P_{i_1}^N(x)$ can be assumed to be merged into $P_{i_1}^N(x)$, i.e., $P'_{i_2}(x) = P_{i_1}^N(x)$ OR $P_{i_2}^N(x)$. Since there is no overlap of the positions containing 1's in both $P_{i_1}^N(x)$ and $P_{i_2}^N(x)$, new information on the merging fault is added to $P'_{i_2}(x)$. Applying the XOR operation between $P'_{i_2}(x)$ and $P_{i_2}^N(x)$ at the destination of $P'_{i_2}(x)$, we get $P_{i_2}(x) = P'_{i_2}(x) \oplus P_{i_2}^N(x)$. A nonzero resulting polynomial implies that some polynomial is merged into $P_{i_2}^N(x)$. The switch with the merging fault is determined by the topology. That is, $P_{i_1}^N(x)$ merges with $P_{i_2}^N(x)$ at $S(i_f, j_f)$, where $S(i_f, j_f)$ is the j_fth switch located at stage i_f, when $(j_f - 1)r = \Pi_{i=1}^{i_f} E_m T_i(i_1) - \Pi_{i=1}^{i_f} E_m T_i(i_1)$ MOD r and $(j_f - 1)r = \Pi_{i=1}^{i_f} E_m T_i(i_2) - \Pi_{i=1}^{i_f} E_m T_i(i_2)$ MOD r. It is easy to see that no information will be lost when multiple mergings occur. Thus, all multiple merging faults can be determined in one testing routine. ■

If merging faults are assumed to be independent of the interconnection pattern, they can be located in one testing routine. Otherwise, we need $r!$ tests to set each switch to every interconnection pattern for fault location. The AND-*merging* fault can be diagnosed by the same method with the testing polynomial $\bar{P}_j^N(x)$.

A broadcasting fault at one input port of a switch implies a merging fault at the output port of the broadcast data path. Thus, broadcasting faults can be located by the same procedure used for testing merging faults.

Misrouting Faults: There are $r!$ possible permutations in an $r \times r$ switch. To locate a misrouting fault, the testing polynomial $P_i(x)$ for source i must be unique.

Theorem 6: One testing routine is sufficient to locate a multiple misrouting fault for both distributed and centralized routing control PMIN's.

Proof: The testing polynomial for merging faults can also be used for testing misrouting faults. $kr!$ permutation calculations are required in each testing routine. Given a permutation $j = T_kET_{k-1}E\cdots ET_1(i)$, a misrouting fault results when E becomes E', where $E' \neq E$ is a faulty permutation. The fault locating procedure is to find E' of a faulty switch. For a given processor j which receives an incorrect polynomial, all possible permutations have to be calculated to find E' of the faulty switch. Since each switch has $r!$ permutations, we need $kr!$ inverse permutations to locate the faulty switch. ■

A misrouting fault may be caused by either the misdecoding of a routing tag in the RCU of a faulty switch or a stuck-at link/switch which transmits the routing tag before the routing tag is actually decoded.

Blocking Faults: As mentioned earlier, the network is designed such that there are only two logic values, i.e., 0 and 1, in all signal lines. When a blocking fault occurs, a data path cannot be utilized, even though it is available.

Theorem 7: A blocked data path in a centralized routing control PMIN can be located in one testing routine.

The proof of this theorem is straightforward. In a centralized routing control network, a locked route can be established even when its data path is blocked. Since the output of a blocked switch is fixed at 1 or 0, it has the same output as a stuck-at data path. It is much more difficult to locate a blocking fault in a distributed routing control network, because routing tags and data are blocked at the same time. It can be located by a binary search which requires $\log_2 k$ testing routines.

Partial Setting Faults: When a data path is partially stuck, the testing procedures with multipliers can still be applied. Test patterns, however, must be determined by the design details of the masker and the mapper. In case of a partial fault, unaffected data bits have correct outputs but the stuck-at bit needs the same testing procedures as described above. In such a case, we have to examine a faulty bit(s) instead of a faulty word(s).

4) Pattern Generation: Test patterns are generated by pattern generators $\{G_i\}$ which are processors or dedicated hardware mechanisms. The cost of pattern generators is one of the most important factors for evaluating the performance of a testing method. Only two testing patterns $W_n(x)$ and $\{P_i^N(x)\}$ need to be generated for the network level testing. Both patterns can be easily generated when G_i's are *ringed* through a single bit control line. Denote the input and output of the ring in G_i by $D_{i(\text{in})}$ and $D_{i(\text{out})}$, respectively. $D_{N(\text{out})}$ is connected to $D_{1(\text{in})}$, and $D_{i(\text{out})}$ is connected to $D_{i+1(\text{in})}$, $\forall 1 \leq i \leq N-1$. To generate $\{P_i^N(x)\}$, the ring is initialized as $D_{1(\text{in})} = 1$, $D_{i(\text{in})} = 0$, $\forall i$, $i \neq 1$. Operations of G_i at the kth clock cycle are given as

$$OP1.\ P_i(k) = \begin{cases} \text{ONE when } D_{i(\text{in})} = 1 \\ \text{ZERO when } D_{i(\text{in})} = 0 \end{cases},$$

$$OP2.\ D_{i(\text{out})}(k) \leftarrow D_{i(\text{in})}(k)$$

where $P_i(k)$ is that the pattern generated by G_i at the kth clock cycle. The other test pattern $W_n(x)$ can be easily generated by the initialization $D_{i(\text{out})} = \text{ONE}$, $\forall i \leq N$, and applying OP1 and OP2 in each pattern generator. For a given permutation, there are only rk possible mergings on a route and the above testing polynomial is thus not optimal for testing OR-merging faults. For testing OR-merging faults, the length of the testing polynomial can be reduced to rk, when $P_k(x) \neq P_j(x)$ for any pair of polynomials $P_j(x)$ and $P_k(x)$ intersecting in a switch under a given permutation. However, the testing polynomial allows merging and misrouting faults to be tested simultaneously, and, thus, simplifies testing procedures. Moreover, G_i has a very simple structure and can be easily applied to various interconnection networks.

5) Testing Complexity: It is important to consider the testing complexity of the network level testing. The length of test patterns for data path stuck-at faults and misinterpreted command packets is km, where m is the queue length in each switch.[4] The calculation of a misinterpreted command packet is straightforward, because the coefficients of the multiplier can be identified directly from the output stream. The stuck-at-1 faults at the inputs of XOR gates, to which the FL are connected to, can be tested by an all zero polynomial, and its testing length is km. To test single data path stuck-at (delay) faults, we need one testing routine which is composed of at most k steps of inverse permutations. At most k testing routines are thus necessary to repair all multiple data path stuck-at faults, and each testing routine needs k inverse permutations. Thus, a total of $k^2 + k + 2$ inverse permutations is needed for data path diagnosis.

For routing faults, the test pattern length is N. One testing routine is sufficient to identify all merging and broadcasting faults. To locate a merging (broadcasting) fault, two RUT's are needed at a time. Since there are k switches on a RUT and each switch needs $r!$ inverse permutations, $k^2r!$ inverse permutations are required to locate a merging (broadcasting) fault. Finally, $kr!$ inverse permutations are required to locate the misrouting faults.

The network level testing is quite general to handle various circuit implementations and locate faults without completely stopping the normal operations of the network. The testing time varies with the size of the network. Note, however, that

[4] The queue lengths need not be identical.

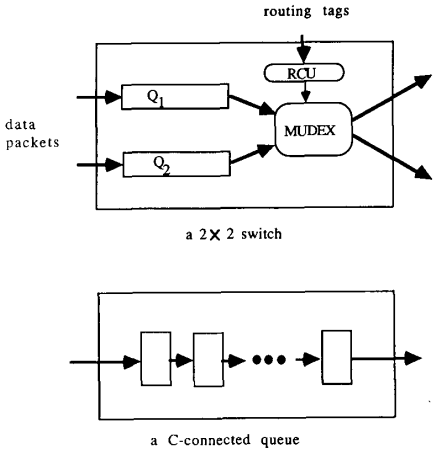


Fig. 5. The structure of a 2 × 2 switch and a C-connected queue.

the network level testing may not detect all possible faults for different circuit implementations. When the network level testing fails to locate some faults, a fast off-line testing method with high fault coverage needs to be called for. The switch level testing described below meets this very need.

B. Switch Level Testing

A switch is composed of data paths and a RCU. Data paths consist of links, queues, and MUDEX's. A pool of buffers, BU_i^j, $1 \le i \le m$, in a switch constitutes the jth queue of the switch, where m is the number of buffers within the queue. A buffer can store one w-bit packet. There are then at least $Nwm \log_r N$ memory bits in an $N \times N$ PMIN built with $r \times r$ switches, and a CSN is the special case of $m = 0$. Let BU_0^j and BU_{m+1}^j denote, respectively, the input and output ports of a switch. It is shown in Fig. 5 that these buffers are cascaded, or *C-connected*, and formally described by CN:$BU_i \rightarrow BU_{i+1}$, where "→" denotes an interconnection within a queue, called an *interlink*.

Different implementations of registers need different test patterns. We use random testing to test the queues. However, when specific test patterns like the one in Appendix A is needed, they are also easy to generate. In each switch, queues are tested by generation and comparison of polynomials. For the generation of a polynomial we can use the natural structure of a queue. The basic idea is to convert the queue into two PG's. A queue can be taken as a $w \times m$ matrix M in which each column is a buffer of w DFF's. Note that DFF's in each row j (collection of the jth DFF's of m buffers), $1 \le j \le w$, of the matrix are cascaded by its natural structure. Assuming w to be even, two PG's, PG_1 and PG_2, are formed by properly cascading the rows of M.

Two symmetric PG's can be obtained by 1) horizontally halving the buffers in the queue, 2) connecting $M(i + 1, 1)$ to $M(i, m)$ $\forall i \le w/2$ for PG_1, and $M(i + 1, m)$ to $M(i, 1)$, $\forall i \ge (w/2) + 1$ for PG_2, 3) identically connecting registers' outputs to the feedback XOR gates in PG_1 and PG_2, and 4) connecting the output of the XOR gate outputs of PG_1 and PG_2 to $M(1, 1)$ and $M((w/2)+1, 1)$, respectively. It is well-known that the maximum period of the output stream of a PG can be obtained when $2^l - 1$ is a prime number, where l is the PG's length, and the PG's characteristic function is irreducible [18]. A fault is detectable when it yields different output sequences in the two PG's.

The PGs' outputs form a 1-out-of-2 codeword when an inverter is added to one XOR gate's output. An XOR gate with n inputs needs $n + 1$ test patterns when n is odd; on the other hand, three test patterns are sufficient for an XOR gate with an even number of inputs. The test patterns for the XOR gate with an odd and an even number of inputs are $\{0\cdots0, 10\cdots0, 010\cdots0, \cdots, 0\cdots01\}$, and $\{0\cdots0, 1\cdots1, I_s\}$, respectively, where I_s is any input with an odd number of 1's. The test patterns for the XOR gate of a PG can be easily generated by setting the PG's initial state. Since every component in the PG's is tested, there is no hardcore in this design.

When two symmetric PG's are used, unidirectional stuck-at faults in a buffer cannot be detected. To solve this problem, PG_1 can be modified such that the outputs $M(i, m/2)$, $\forall i \le w/2$, are connected to the XOR gate whose output is then connected to $M(1, (m/2) + 1)$. Although the physical interconnection of $M(1, 1)$ to $M(w/2, m)$ is different from that of $M((w/2) + 1, 1)$ to $M(w, m)$, both PG_1 and PG_2 still have an identical structure. Such a modification can now detect the unidirectional faults mentioned above. Symmetric and asymmetric PG configurations are illustrated in Figs. 6(a) and (b), respectively.

The optimal testing length of a PG and its fault coverage are important performance parameters. Any DFF in the MSA fault model can be s-a-1, s-a-0, or fault-free. To evaluate the MSA fault coverage of the proposed method, we only need to consider the type and position of leading faulty DFF's in a block. Consider a pair of leading faulty DFF's, s_1 and s_2, which are in x^i and x^j positions of PG_1 and PG_2, respectively. The effects of faults in s_1 and s_2 can be distinguished only when they yield different outputs for at least one clock cycle. We begin with the simplest special case of the MSA fault model, i.e., the SSA fault model.

Theorem 8: All SSA faults are detectable, and the maximum testing length is $r + 1$, where r is the order of the PG.

Proof: An SSA fault in a PG is detectable when it generates an output different from that of the other PG. Let the initial state of the PG be $\Sigma_{i=1}^{r} n_i x^i$, where $n_1 = 1$ and $n_i = 0$, $\forall i \ne 1$. When an s-a-0 is located at output of x^i, n_1 is falsely inverted at the ith shift. The fault cannot be revealed during the first $i - 1$ clock cycles, because the s-a-0 is the same as the preset value of a fault-free circuit. The worst case occurs when the s-a-0 is located at the output of x^r, and, thus, r is the maximum testing length.

When an s-a-1 is located at the output of x^i, it will change the parity of the output immediately when it propagates to a feedback line. The worst case occurs when feedback lines emanate from x^1 and x^r, and the s-a-1 is present at the input of x^1. The output of the faulty PG is the same as the nonfaulty one until the $r + 1$th clock cycle. Thus, the maximum testing length for SSA faults is $r + 1$. ■

To calculate the MSA fault coverage, the position and type

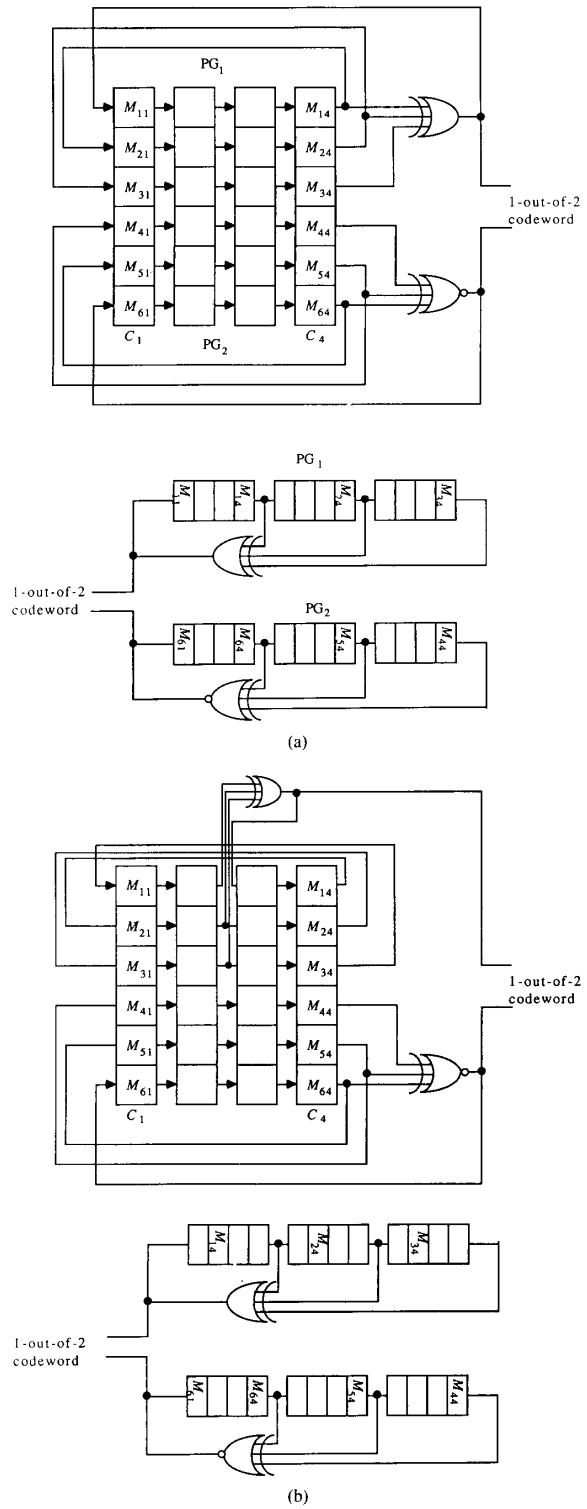


Fig. 6. A queue transformed into two symmetric and asymmetric PG's. (a) Symmetric PG's. (b) Asymmetric PG's.

TABLE I

MSA FAULT COVERAGES OF PG's. (a) MSA FAULT COVERAGE (C) OF DIFFERENT NUMBERS AND LOCATIONS OF FEEDBACK LINES. (b) $k = 8$ WITH THREE FEEDBACK LINES. (c) PG's TESTED TWICE BY TWO FEEDBACK CONFIGURATIONS. (d) FEEDBACK LINES AT k, 0, AND THE PG's ARE TESTED TWICE WITH TWO DIFFERENT INITIAL STATES.

feedback(f_i)	Stage No.(k)							
	3	4	5	6	7	8	12	16	20
k,0	.728	.715	.710	.709	.708	.708	.708	.708	.708
k,1,0	.780	.788	.793	.794	.794	.795	.795	.795	-
k,2,1,0	*	.827	.828	.831	.831	.832	.831	-	-

(a)

f_1, f_2, f_3	8,1,0	8,2,0	8,3,0	8,4,0	8,5,0	8,6,0
C	.795	.785	.780	.777	.770	.746

(b)

$\underset{C}{(k,f_1',f_2')}$; $f_1=k, f_2=0$

(3,2,1) .868	(4,2,1) .876	(5,2,1) .877	(6,2,1) .876	(7,2,1) .875	(8,2,1) .875	(9,2,1) .875	(10,2,1) .875	(11,2,1) .875	(12,2,1) .879
(8,1,0) .875	(8,3,1) .876	(8,3,2) .876	(8,4,1) .879	(8,4,3) .877	(8,5,1) .886	(8,5,3) .885	(8,5,4) .879	(8,6,1) .900	(8,6,3) .899
(8,6,5) .874	(8,7,0) .801	(8,7,1) .875	(8,7,5) .883	(8,7,6) .869	(16,1,2) .875	(18,10,9) .875	(19,10,9) .875	(20,11,10) .875	(21,12,11) .875

(c)

$\underset{C}{(k,in_2=2^n)}$; first initial condition $in_1 = 2^0$

(3,1) .762	(3,2) .809	(4,1) .748	(4,2) .764	(4,3) .805	(5,1) .745	(5,2) .749	(5,3) .764	(5,4) .803	(6,1) .743	(6,2) .745
(6,3) .749	(6,4) .764	(6,5) .803	(7,1) .743	(7,3) .745	(7,5) .764	(7,6) .803	(8,1) .743	(8,2) .745	(8,4) .745	(8,6) .764
(8,7) .803	(9,1) .743	(9,3) .743	(9,5) .745	(9,7) .764	(9,8) .803	(10,1-5) .743	(10,6) .745	(10,7) .749	(10,8) .764	(10,9) .803
(12,1-7) .743	(12,8) .745	(12,9) .749	(12,10) .764	(12,11) .803	(16,1-10) .743	(16,11) .743	(16,12) .745	(16,13) .749	(16,14) .764	(16,15) .803

(d)

"–": not computed because of excessive simulation time requirements.
"*": not applicable.

of leading faulty DFF's must be considered. Each DFF can be s-a-1, s-a-0, or fault-free, and the number of MSA faults in a queue is $3^{mw} - 1$. Due to the fault masking effect, the actual computing time is $K \prod_{i=1}^{k} (2r_i + 1)^2$, where r_i is the order of block B_i and K is the computing time required for each iteration. As shown in Table I, various testing strategies are simulated to examine their MSA fault coverages. The initial state of each simulation is $n_1 = 1$, $n_i = 0$, $\forall i \neq 1$. From the simulation results, the following three conjectures are made.

Conjecture 1: The fault coverage is dominated by the number of feedback lines. It monotonically increases with the number of feedback lines. The length of a PG has little effect on the fault coverage.

Conjecture 2: For a given PG of length r, and l_f feedback lines, the MSA fault coverage attains a maximum when feedback lines are located at x^1, x^2, $\cdots$ x^{lf-1}, and x^r.

Conjecture 3: The MSA fault coverage increases with the number of testing routines, each of which uses a different initial state. The optimal testing length for MSA faults is r for a given initial state and a feedback configuration.

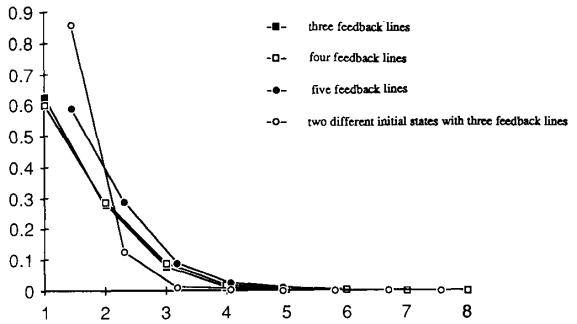


Fig. 7. Detected-faults/detectable-faults versus number of shifts when $r = 8$.

For a given PG configuration and the initial state, theoretically, 2^r shifts are required to exercise all the states of the PG. However, our simulation results show that testing lengths are rarely required to be longer than the length of the PG. Although the choice of an initial state affects the fault coverage, the number and location of feedback lines are the dominating factors in the fault coverage. It is shown in Fig. 7 that about 65 percent of detectable MSA faults are immediately detected for most cases. From Theorem 8 and the above conjectures, each testing length is found to be $r + 1$.

Unlike the off-line testing of data paths, a faulty RCU can be detected on-line. An *RCU checker* is proposed to detect faults in the RCU using its output signals. An RCU has an r $\log_2 r$-bit input and an r^2-bit output. The RCU output signals are denoted by E_{ij}, $1 \le i, j \le r$, where $E_{ij} = 1$ if queue j is connected to output port i, and $E_{ij} = 0$ otherwise. For any fixed k, $\{E_{ik}\}$ or $\{E_{ki}\}$, $1 \le i \le r$, forms a 1-out-of-r codeword. Thus, $2r$ 1-out-of-r self-checking checkers, one for each $\{E_{ik}\}$ or $\{E_{ki}\}$, are needed to detect all noncodeword outputs.

The outputs of the RCU and queues are the inputs of the MUDEX to which they are connected. The RCU and queues can be tested first using the above procedures. If they are fault-free, then the MUDEX and the links connected to the MUDEX are tested by using the RCU and queues to generate test patterns for the the MUDEX and its links. For output verification, the streams from the MUDEX's of stage i are transmitted through the links and then verified at stage $i + 1$ with special mechanisms.

Before we develop the test method for MUDEX's and links, it is necessary to find the test patterns of the $r \times 1$ multiplexer shown in Fig. 8, where E_i and D_i are the enable and data of the ith input, respectively. The $r \times 1$ multiplexer is implemented by r two-input AND gates and an OR gate.

Lemma 4: All SSA faults in the multiplexer of Fig. 8 can be detected in $r + 2$ steps.

Proof: After fault collapsing, the faults that need to be tested are 1) s-a-0 and s-a-1 primary output, i.e., output of the OR gate in Fig. 8, 2) s-a-0 A_i, $\forall i \le r$ in Fig. 8, and 3) s-a-1 $D_i(E_i)$, $\forall i \le r$. Test patterns can be derived as follows.

PT(1): $E_iD_i = 10$, $\forall i \le r$,

PT(2): $E_iD_i = 01$, $\forall i \le r$,

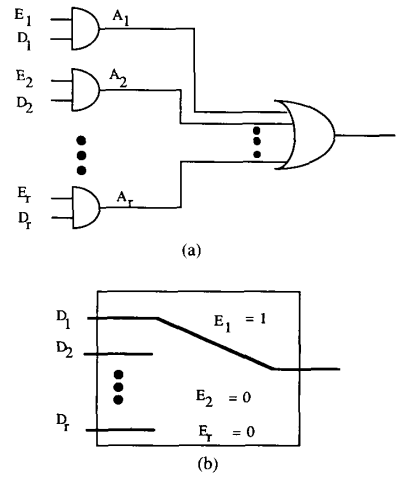


Fig. 8. The logic and functional diagrams of a multiplexer with r data inputs and r enable signals. (a) Logic diagram. (b) Functional diagram.

PT(3): $E_1D_1 = 11$, and $E_iD_i = e_id_i$ for $i \ne 1$

PT(4): $E_2D_2 = 11$, and $E_iD_i = e_id_i$ for $i \ne 2$

... ...

PT($r+2$): $E_rD_r = 11$, and $E_iD_i = e_id_i$ for $i \ne r$,

where $d_i = 0$ or $e_i = 0$, $\forall 1 \le i \le r$.

An $r \times r$ MUDEX connects r queues' outputs to r links. The MUDEX can be implemented by two-level AND and OR gates, where each MUDEX's output port is basically a multiplexer. An example design of MUDEX is shown in Fig. 9, where E_{ij} is the enable signal from the RCU to route the packet at queue j to output port i. E_{ij} fans out to w branches to simultaneously enable the w bits of queue j.

Theorem 9: Any SSA fault on links or MUDEX's can be tested in $r + 2$ clock cycles.

Proof: Since operations to be applied to each of the w bits of a packet are identical, it is sufficient to discuss only one bit of the packet. Each output port of the 1-bit MUDEX is an $r \times 1$ multiplexer, and there are a total of r multiplexers in a MUDEX. Test patterns derived in Lemma 4 can be directly applied to test the MUDEX. However, it is important to minimize the test length when one selects test patterns. The proposed testing procedures are as follows. At clock cycle 1, all the RCU's outputs are set to 1 and the queue outputs to 0. Queue outputs are fixed at 1 for the rest of the procedures. At cycle 2, all the RCU's outputs are set to 0. During the remaining r cycles, the RCU performs permutation $i \rightarrow (i + j - 1)$ MOD r at cycle j, $3 \le j \le r + 2$. When the network uses distributed routing control, the queues for storing routing tags can be used to generate the desired routing requests to the RCU. By this permutation and the data queue setting, the r multiplexers in a MUDEX are tested simultaneously. The testing procedures are shown in Fig. 10. ■

The MUDEX's output stream is two 0's followed by r 1's. Since both 0 and 1 appear at each switch's output, and thus, at each link, the links can be tested without introducing any

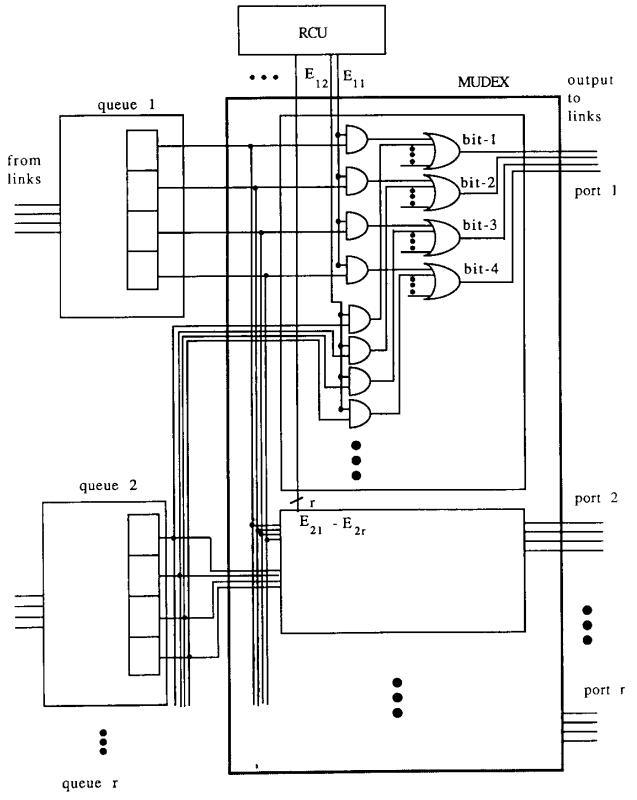


Fig. 9. An example of the MUDEX in an $r \times r$ switch.

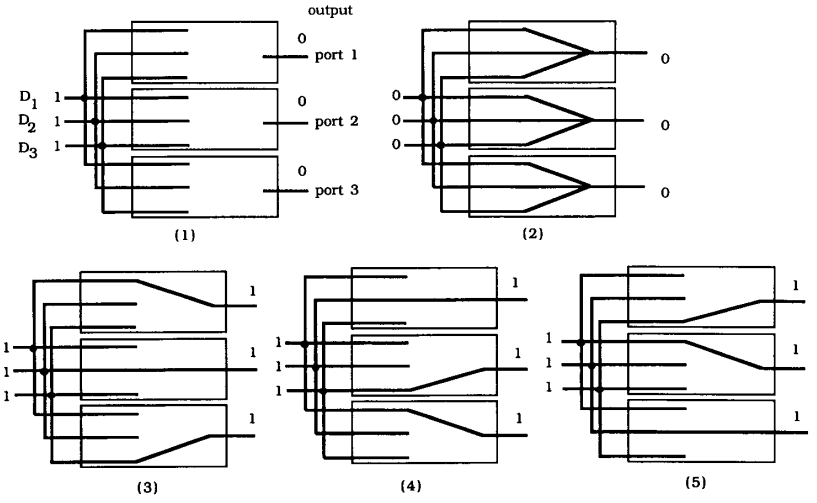


Fig. 10. The testing procedures for a 3 × 3 MUDEX.

additional cost. For test verification, it should be noted that all links in the network have identical outputs. Thus, the comparison method to verify the test results of queues can be applied similarly. Without loss of generality, the number of links from each switch is assumed to be even. Half of the links are connected to the primary inputs of a fan-out-free XOR tree, and the rest are connected to the primary inputs of the other fan-out-free XNOR tree. The outputs of the two fan-out-free networks form a 1-out-of-2 codeword. A design example for this method is given in Fig. 11(a).

It has been shown in [21] that a linear function implemented by two-input XOR gates needs at most four test patterns. The test patterns can be recursively derived from the primary output of the XOR (XNOR) tree to the primary inputs. Assume

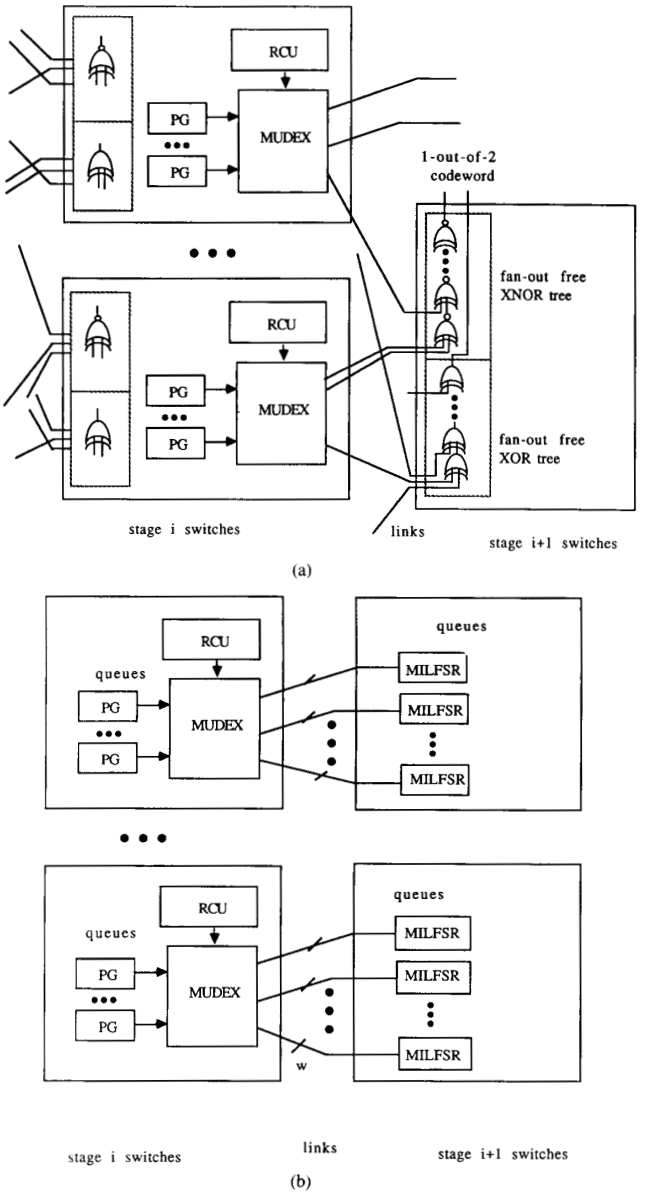


Fig. 11. Verification of testing response by comparison and signature analysis.

that a linear function P_n of n variables is implemented by an XOR tree as in Fig. 11(a). Then P_n can be recursively expressed by $P_n = x_n \oplus P_{n-1}$, where x_n is the nth primary input (variable), and P_{n-1} is the linear function implemented by the subnetwork excluding the primary output XOR gate and the primary input x_n. To test the primary output gate, it is sufficient to have $P_{n-1}x_n = 00, 01, 10, 11$. The input stream in x_n is then 0101, and P_{n-1} should be 0011. We want to derive a test pattern which can be easily generated, e.g., all inputs are identical, or, only one or two inputs are different from others. Thus, for $P_{n-1} = 0011$ and $P_{n-1} = P_{n-2} \oplus x_{n-1}$, we set $x_{n-1} = 0101$ (as x_n), and thus, $P_{n-2} = 0110$. It can be shown by induction that $x_i = 0101$, $\forall i \neq 1$, and $x_1 = 0110$ (0011) when the number of gates is even (odd).

It is now clear that we can eliminate the hardcore in the XOR and XNOR trees when their test patterns are applied. Assume that $w/2$ links are connected to inputs $x_1 \cdots x_{w/2}$ of the XOR tree. To test the XOR (XNOR) tree, we need to add one more input x_0 to the tree, and x_0 is controlled by the BIT. Since the output stream of MUDEX testing is composed of two 0's and r 1's, the XOR (XNOR) tree can be tested simultaneously with the MUDEX's and links when $0011 \cdots (0110 \cdots)$ are simultane-

ously applied to x_0 by the BIT. It requires wr XOR and XNOR gates in each switch to verify the test response. When the number of XOR (XNOR) gates is too high, the testing method can be decomposed into two phases as follows. In phase one (two), all the queues in even (odd) stage switches serve as pattern generators and those in odd (even)-stage switches serve as multiple input linear feedback shift registers (MILFSR's). To test MUDEX's and links, the outputs of the MILFSR's are compared in a way similar to the case of testing queues. Thus, the network can be tested in two phases, each phase requiring $r + 2$ clock cycles. An example design showing such a strategy is given in Fig. 11(b).

V. Conclusion

A two-level testing strategy is developed in this paper. The network level testing uses processors as testers, and the switch level testing uses switches to test the network. The network level testing is concurrent testing and the switch level testing is off-line testing.

The first step to ease the network level testing is to synchronize network operations. Then, the network is tested with different polynomial operations. Although only a small number of functional fault models and the corresponding testing procedures are discussed, this method can be easily extended to detect other faults at the cost of additional testing times. For example, it can be used for *conflict resolving* testing. When two or more requests in a switch requesting the same output port to use, the RCU should grant only one. Using the testing polynomials, $P_i^N(x)$ and conflicting routing requests, testing an $r \times r$ switch can be done in $r^r - r!$ testing routines. Obviously, conflict resolving testing is very expensive, when r is large.

Since testing polynomials can be easily generated by processors or a dedicated hardware, the proposed method can be used for both SIMD and MIMD machines. Most of network faults can be diagnosed in a decentralized manner by processors with the polynomial testing. A processor need not communicate with others except for the simultaneous submission of polynomials. Thus, when the network becomes faulty, the testing method can be used to identify, with different levels of accuracy, fault-free components, i.e., a faulty system can be reconfigured following the testing.

The costs of testable designs are measured by the extra hardware required for links and logic components in switches. When the packet width is more than one bit, one of the data links can be used as a feedforward line (but not as a feedback line). However, when a data line is to be transformed into a feedforward line, a multiplexer should be used to bypass the queue that the link is connected to. One masker, a mapper, and an adder of w bits are required for each data queue. For an $r \times r$ switch, at least r^2 logic gates are needed to implement the MUDEX, and the overhead, relative to the combinational circuit of the switch to implement the tracer, is $2/r$. It should be noted that the overhead is an upper bound, because hardware for the data buffers is not considered.

The goal of the switch level testing is to obtain high fault coverage with a small testing time. Queues are self-tested first by converting them into polynomial generators. Furthermore, test patterns for the MUDEX can also be generated by the PG's. For a C-connected queue, it needs w extra interlinks to convert a queue into PG's. Testing the RCU by the conventional method is very inefficient. It is proposed to use r 1-out-of-r codeword checkers to monitor the outputs of the RCU. Finally, the MUDEX's and links are tested simultaneously. Testing responses are verified by using XOR and XNOR trees. It is shown that the network testing time is independent of the network size and the design method can be applied to various types of switch. For the switch level testing, comparators in the switches are the predominating overhead, which is $wr/r^2 = 1/r$ for the combinational circuits of switches, and no extra links are needed.

In this paper, we have focused only on the development of testing strategies. To determine an optimal (in some sense) testing period, the tradeoff between the performance penalty and fault-detection time must be studied. This and modifications of the proposed testing method for CSN's are the subject of further inquiry.

Appendix A

Fault Coverage of Polynomial Testing

Fault coverage of polynomial testing may vary with different circuit implementations of the network. To obtain a concrete figure on its fault coverage at the logic level, consider an example LFSR design in Fig. 12. The basic structure of a shift register is essentially a master–slave S/R flip-flop. The ith gate in a flip-flop is denoted as G_i, and its output and jth input (indexed from top to bottom) by GO_i and GI_i^j, respectively. It should be noted that latches usually have two outputs Q and $\bar{Q}$. However, since the number of links is a major concern in the network design, only the Q output of the slave latch will be used, and $\bar{Q}$ is ignored. In other words, a fault is detectable only when an erroneous response can be observed at the Q output of the slave latch. During normal operations, the S and R inputs are in the form of $d\bar{d}$, 11, $d\bar{d}$, 11, $\cdots$, where $d \in \{0, 1\}$. It should be noted that the input 11 is inserted automatically when CLK = 0 (1) at the master (slave) latch, and the outputs of a latch are read out when SR = 11. 01, 10, and 11 at the SR inputs are referred to as r, s, and b, respectively.

To derive its test set, the slave latch composed of G_7, G_8 is considered first. Using the D-algorithm, test patterns for faults in the latch are summarized as follows.

faults/nodes	GI_7^1	GI_7^2	GO_7	GI_8^1	GI_8^2	GO_8
s-a-0	sb	sb	rb	rb	rb	sb
s-a-1	$sbrb$	$sbrb$	sb	sb	$rbsb$	rb

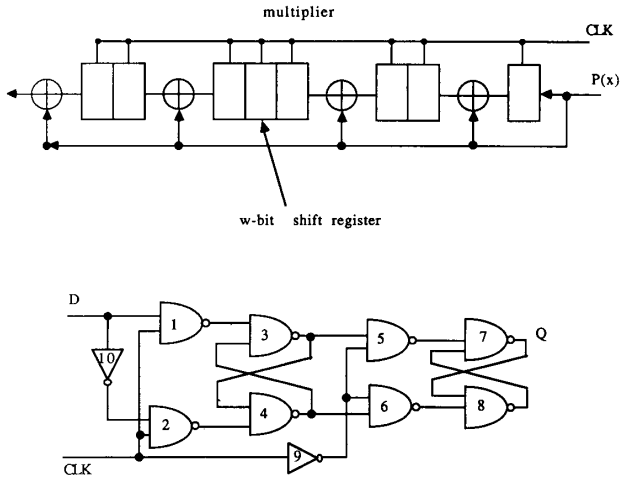


Fig. 12. An example LFSR implemented with master-slave *SR* latches.

Testing the master latch is more complicated, because the slave latch must be in an appropriate initial state, i.e., output, to propagate the erroneous response of the master latch to the Q output of the slave latch. For example, when $(Q, \bar{Q}) = (0, \bar{D})$ at the master latch,[5] the initial state of the slave latch must be (1, 0) to obtain D at the output of the slave latch. Otherwise, the slave latch's output is 01, meaning that the fault is not tested. In our case, the master and slave latches have identical test patterns.

Test patterns for G_5 and G_6 are summarized below:

faults/nodes	GI_5^1	GI_5^2	GO_5	GI_6^1	GI_6^2	GO_6
s-a-0	*sbrb*	*sbrb*	*sb*	*rbsb*	*rbsb*	*rb*
s-a-1	*rb*	*rbsb*	*sbrb*	*sbrb*	*sb*	*rbsb.*

It can be shown that the above pattern can test G_1, G_2 simultaneously. GI_{10} s-a-1 and GO_{10} s-a-0 can be tested by *sbrb*. The only undetectable faults are GI_{10} s-a-0 and GO_{10} s-a-1, because the erroneous responses can only propagate to the $\bar{Q}$ output of the slave latch. GI_9 s-a-1 and GO_9 s-a-0 will block the transmission of data, and thus, *sbrb* is sufficient to test them. GI_9 s-a-0 and GO_9 s-a-1 faults do not cause logic faults, and thus, are not detectable by the D-algorithm. However, the *memory* of the register is lost when the above two faults occur, i.e., the latch fails to hold the data for a specific period. Assume that the high (low) period t_T of testing clock is three times slower than the latch's transition time t_d. Then, such faults can be tested by the polynomial testing method. For example, when GO_9 is stuck-at-1, the data at the input of the register will be shifted to the slave latch's output after $2t_d$. At the third t_d, the data are erroneously shifted into the next register. Thus, the shift register fails to perform the *delay* function. Occurrence of such faults in a queue implies that the length of queue is reduced. Since the polynomial testing can identify the configuration of an LFSR, all such faults can be detected.

Although the polynomial testing is developed as a functional level testing, it can clearly detect all the detectable stuck-at faults at the logic level of registers. Since only two of the 56 faults are undetectable in a register, and all other faults, i.e., stuck-at faults on the XOR gates and feedforward lines, are detectable, the lower bound of the polynomial testing method is 96 percent.

[5] They are inverted to (1, D) before they enter the slave latch.

Appendix B

List of Symbols

A, N, S, X	Four states of a switch on the route under test.
B_i	The ith block in a PDM.
BU_i	The ith buffer in a queue.
$D_{i(\text{in})}$, $D_{i(\text{out})}$	The ring link input (output) of the ith pattern generator G_i.
DFF	A D-type flip-flop with a single input and a single output.
E_m	Switch permutation, $E_m^{i+1}: f(ir + j) \rightarrow f(ir + j + m \text{ MOD } r)$, where $f(ir + j)$ and $f(ir + j + m \text{ MOD } r)$ are the global indexes of the jth input port and the $(j + m \text{ MOD } r)$th output port of the $(i + 1)$th

	switch, respectively, where r is the switch order, and $\rightarrow$ is the interconection. E_m^i is denoted as E_m when all switches have the same permutation.
E_m^{-1}	The inverse of the switch permutation E_m.
E_{ij}	An RCU output which enables queue j to be connected to output port i.
$GF(2)[x]$	The polynomial ring.
I, I_n	I is the set of integers and $I_n = \{0, 1, 2, \cdots, n\}$.
M	An $m \times w$ matrix representing m buffers in a queue of w bits each. The ith row of M is the ith bits of all buffers and the jth column of M is the jth buffer in the queue.
$M_{ij}(x)$	The multiplier formed by the route connecting source i to destination j.
MUDEX	An $r \times r$ MUDEX is the combination of r multiplexers and r demultiplexers. It is used to direct packets in a PMIN switch.
ONE, ZERO	Coefficients of a word polynomial, ONE = $\overline{\text{ZERO}}$.
$P(x)$, $\bar{P}(x)$	A polynomial and its complement.
$P_j^N(x)$	$P_j^N(x) = \Sigma_{i=0}^N c_i x^i$, where $c_j = 1$, $c_i = 0$, $i \neq j$, is a test pattern submitted by processor j.
$P_E^m(x)$	$P_E^m(x) = \Sigma_{i=0}^k x^{(m+1)i}$, the test polynomial for m unit delay faults.
PDM	Polynomial multiplier or divisor.
PG	Polynomial generator.
PMIN	Packet switching multistage interconnection network.
$Q(x)$	The product of $P(x)$ and $M(x)$, or the quotient of $P(x)/D(x)$, or the output of a PDM.
$R(x)$	The remainder of the operation that $D(x)$ divides $P(x)$ or the final contents of a PDM.
RCU	Routing control unit.
RUT	Route under test.
r_i	The length or order of B_i.
SSA	Single stuck at fault.
T_i	The link permutation at stage i.
$W_k(x)$, $W(x)$	$W_k(x) = \Sigma_{i=0}^k x^i$. $W(x) = W_\infty(x)$.
$\Sigma_{i=0}^r n_i x^i$	The initial state of a PDM or a PG.

Acknowledgment

The authors are grateful to N. K. Jha for the valuable comments on the material in Section IV-C. They also want to thank M. H. Woodbury and referee A for numerous suggestions on the initial draft of this paper.

References

[1] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems*. Rockville, MD: Computer Science, 1976.

[2] W. K. Fuchs, J. A. Abraham, and K. H. Huang, "Concurrent error detection in VLSI interconnection networks," in *Dig. Papers, FTCS-13*, 1983, pp. 309-315.

[3] J. E. Lilienkamp, D. H. Lawrie, and P. C. Yew, "A fault tolerant interconnection network using error correcting codes," in *Dig. Papers, FTCS-12*, 1982, pp. 123-125.

[4] T. Y. Feng and C. L. Wu, "Fault-diagnosis of a class of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-30, pp. 743-758, Oct. 1981.

[5] D. P. Agrawal, "Testing and fault tolerance of multistage interconnection networks," *Computer*, pp. 41-53, Apr. 1982.

[6] N. J. Davis IV, W. T.-Y. Hsu, and H. J. Siegel, "Fault location techniques for distributed control interconnection networks," *IEEE Trans. Comput.*, Vol. C-34, pp. 902-910, Oct. 1985.

[7] D. C. H. Lee and J. P. Shen, "Easily-testable (N, K) shuffle/exchange networks," in *Proc. Int. Conf. Parallel Processing*, 1983, pp. 65-70.

[8] D. P. Agrawal and J.-S. Leu, "Dynamic accessibility testing and path length optimization of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-34, pp. 255-266, Mar. 1985.

[9] V. Cherkassky, E. Opper, and M. Malek, "Reliability and fault diagnosis analysis of fault-tolerant multistage interconnection networks," in *Dig. Papers, FTCS-14*, 1984, pp. 246-251.

[10] M. Malek and E. Opper, "Multiple fault diagnosis of SW-banyan networks," in *Proc. FTCS-13*, 1983, pp. 446-449.

[11] E. Opper and M. Malek, "Real-time diagnosis of banyan networks," in *Proc. Real Time Syst. Symp.*, 1982, pp. 27-36.

[12] W. Y.-P. Lim, "A test strategy for packet switching networks," in *Proc. Int. Conf. Parallel Processing*, 1982, pp. 96-98.

[13] V. Cherkassky and E. Opper, "Fault diagnosis and permuting properties of CC-banyan networks," in *Proc. Real-Time Syst. Symp.*, 1984, pp. 175-183.

[14] J. Y. Maeng, "Self-diagnosis of multistage network-based computer systems," in *Dig. Papers, FTCS-13*, 1983, pp. 324-331.

[15] S. Thanawastien and V. P. Nelson, "Diagnosis of multiple faults in shuffle/exchange networks," in *Proc. Real Time Syst. Symp.*, 1984, pp. 184-192.

[16] R. E. Bryant, "A switch-level model and simulator for MOS digital systems," *IEEE Trans. Comput.*, vol. C-33, pp. 160-177, Feb. 1984.

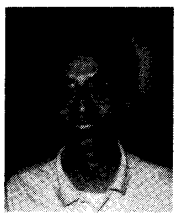
[17] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test," *IEEE Trans. Comput.*, vol. C-30, pp. 866-875, Nov. 1981.

[18] S. W. Golomb, *Shift Register Sequences*. San Francisco, CA: Holden-Day, 1967.

[19] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1978.

[20] J. E. Smith, "Measures of effectiveness of fault signature analysis," *IEEE Trans. Comput.*, vol. C-29, pp. 510-514, June 1980.

[21] J. P. Hayes, "On realizations of boolean functions requiring a minimal or near-minimal numbers of tests," *IEEE Trans. Comput.*, Vol. C-20, pp. 1506-1513, Dec. 1971.



Jyh-Charn Liu (S'84) was born in Kaohsiung, Taiwan, on December 6, 1956. He received the B.S. and M.S. degrees in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 1979 and 1981, respectively.

He was a system engineer of Siantek Co. Taiwan in 1983. Since 1984 he has been a Research Assistant at the University of Michigan, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. His research interests include fault-tolerant computing and easily testable architectures for real-time applications.

Mr. Liu is a student member of the IEEE Computer Society.

Kang G. Shin (S'74-M'78-SM'83) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

From 1970 to 1972 he served in the Korean Army as an ROTC officer and from 1972 to 1974 he was on the research staff of the Korea Institute of Science and Technology, Seoul, working on the design of VHF/UHF communication systems. From 1978 to 1982 he was an Assistant Professor at Rensselaer Polytechnic

Institute, Troy, NY. He was also a Visiting Scientist at the U.S. Airforce Flight Dynamics Laboratory in Summer 1979 and at Bell Laboratories, Holmdel, NJ in Summer 1980. Since September 1982, he has been with the Department of Electrical Engineering and Computer Science at The University of Michigan, Ann Arbor, MI, where he is currently a Professor. He has been very active and authored/coauthored over 140 technical papers in the areas of distributed fault-tolerant real-time computing, computer architecture, and robotics and automation. As an initial phase of validation of architectures and analytic results, he and his students are currently building a 19-node hexagonal mesh multiprocessor at the Real-Time Computing Laboratory (RTCL), The University of Michigan.

Dr. Shin is a member of the Association for Computing Machinery, Sigma Xi, and Phi Kappa Phi. He was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium and is the Guest Editor of the special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems which appeared in August 1987. In 1987, he also received an Outstanding Paper Award for a paper on robot trajectory planning published in IEEE TRANSACTIONS ON AUTOMATIC CONTROL.