Built-In Self Test Architectures for Multistage Interconnection Networks

E. Bernard,*
OEN TN ETD3
Siemens AG
81359 Munich

S. Simon, and J. A. Nossek
Inst. Network Theory
TU Munich
80290 Munich

Abstract

A novel built-in self test architecture for locally controlled cube-type $N \times N$ multistage interconnection networks (MINs) is presented. First, a state-based pseudoexhaustive test procedure for this class of MINs is outlined. Then, a labelling algorithm on a binary n-cube is described which generates the necessary inputs for the tests. From the dependence graph of this algorithm a tree architecture is derived which results in a hardware overhead of $O(1/\log N)$.

1 INTRODUCTION

The still rapidly growing complexity of integrated circuits results in a steady increase of system performance. However, these merits are drastically reduced due to the susceptibility of these circuits to physical hardware defects [1]. In order to ensure reliable function testing is indispensable. Since failures also occur in field operation, the systems have to be tested periodically during operation. But as it is impossible in many casesto provide expensive automatic test equipment, failures either have to be detected by on-line error detection using the actual data, or the system has to test itself, i.e., a built-in self test (BIST) is implemented with additional hardware.

In the past years MINs have gained significant importance in high performance communication systems [2, 3]. Multistage interconnection networks (MINs) are well suited for being tested efficiently due to their modularity and regularity [4].

For locally controlled cube-type MINs a variety of test procedures has been presented (e.g. [5, 6, 7]). However, no method has been presented so far to generate the test stimuli. Thus, the only alternative for the implementation of BIST is to store the stimuli on the chip or in some additional component; a quite hardware-intensive method.

In this paper, we first propose a state-based pseudoexhaustive test procedure for locally controlled MINs capable of detecting failures associated with pure transmission of data [5], but which also can detect functional faults of the switching elements independently of the actual implementation. A graph labelling algorithm [7] for generating the test stimuli automatically is outlined. Based on the dependence graph of this procedure an efficient tree-like hardware architecture is derived for generating the test stimuli on-chip. If additionally the fault free test results are generated in the same way, an efficient BIST architecture is obtained.

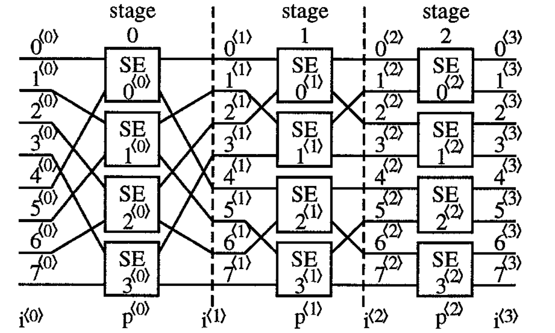

stage 0
stage 1
stage 2
SE $0^{\langle 0\rangle}$
SE $1^{\langle 0\rangle}$
SE $2^{\langle 0\rangle}$
SE $3^{\langle 0\rangle}$
SE $0^{\langle 1\rangle}$
SE $1^{\langle 1\rangle}$
SE $2^{\langle 1\rangle}$
SE $3^{\langle 1\rangle}$
SE $0^{\langle 2\rangle}$
SE $1^{\langle 2\rangle}$
SE $2^{\langle 2\rangle}$
SE $3^{\langle 2\rangle}$
$0^{\langle 0\rangle}$ … $7^{\langle 0\rangle}$
$0^{\langle 1\rangle}$ … $7^{\langle 1\rangle}$
$0^{\langle 2\rangle}$ … $7^{\langle 2\rangle}$
$0^{\langle 3\rangle}$ … $7^{\langle 3\rangle}$
$i^{\langle 0\rangle}$
$p^{\langle 0\rangle}$
$i^{\langle 1\rangle}$
$p^{\langle 1\rangle}$
$i^{\langle 2\rangle}$
$p^{\langle 2\rangle}$
$i^{\langle 3\rangle}$

Figure 1: Topology of a MIN and index notation for $N = 8$.

2 Test Procedures For MINs

First, we will describe the architecture of the networks considered in this paper. A MIN for interconnecting N inputs with N outputs is composed of $n = \log_2 N$ stages with $N/2$ 2×2 switching elements (SEs) each as depicted in Fig. 1. The input links of the various stages are referred to by an index i, $0 \leq i < N$. The stage index is s, with $0 \leq s < n$, and the index of the SEs within a stage is p, $0 \leq p < N/2$. A superscript in angle brackets denotes the stage s to which an input $i^{\langle s\rangle}$ or an SE $p^{\langle s\rangle}$ belongs. This index notation is also shown in Fig. 1. Since N is a power of two, the indices can be expressed in a dyadic representation, i.e., $i = (i_{n-1}, \ldots, i_1, i_0)$. The data dependencies between the stages can be realized by shuffle permutations [8]. Thus, the indices of the input and output links of an SE $p^{\langle s\rangle}$ differ exactly in bit i_{n-s-1}.

We will restrict in this paper to ATM-like destination tag routing, but other routing strategies can be handled with our procedure as well [9, 7]. A data packet comprises of q bit-parallel data words with b bits each. The data words either can be assigned to the pay load information or to the routing header which specifies the address (i.e. index) of the output link the packet is destined for. A specific bit of the routing header is inspected by the SEs of a particular stage and if this bit is 0 (1) the packet is routed to the upper (lower) output of the SE. If two data packets are destined for the same output of an SE one packet at the input has to be blocked. Accordingly, a SE has to assume one of the eleven states $A_0 - A_{10}$ depicted in Fig. 2 a). A line through the SE denotes an interconnection and the ”⊣” symbol denotes a

*The author has been with the Institute for Network Theory, TU Munich, where this work has been performed.

1066-1409/96 $5.00 © 1996 IEEE

blocked input. The state transition diagram in shown in Fig. 2 b). Transition arcs originating and terminating at the same node are not depicted. Edges traversed in both directions are denoted by double-headed arrows. The conditions for state transitions are coded as follows: by a single letter (u or l) if a transmission ends at an upper or lower input. A beginning transmission is represented by two letters, denoting the input and output of an SE, e.g. uu: request from upper input to upper output.

In order to test an SE it is sufficient to traverse all transition edges of the state diagram [10]. For testing all SEs of the network we propose the following procedure:

Test Procedure: In a first test phase all SEs traverse the same state sequence $A_0 \rightarrow A_6 \rightarrow A_0 \rightarrow A_5 \rightarrow A_0$. In order to bring all SEs to state A_6, all routing headers have to be identical to the index of the input link at the input of the network. For bringing all SEs to state A_5, all routing headers have to be the bit-wise complement of the input index. In this phase, all bit lines for a data word can be tested [6, 5]. For this task the pay load information has to contain the bit-wise complement of the routing header (for detecting stuck-at faults), a data word with all bits 0 or 1 (depending on whether the input index has even or odd parity [11]) and $\lceil \log_2 b \rceil$ data words so that all pairs of bit lines have at least once different values (for detecting bridging faults). The total number of data words for a packet is thus $q = \lceil \log_2 b \rceil + 3$. If no fault can be detected, all bit-lines are fault free and all SEs can assume the states A_5 and A_6 correctly. Otherwise the fault can be located [7], if necessary.

In the second test phase all remaining transitions have to be traversed. This can be done by the four cycles:

$$\begin{aligned}
&A_0 \xrightarrow{\mathrm{ul}} A_4 \xrightarrow[\mathrm{lu}]{\mathrm{u}} A_2 \xrightarrow[\mathrm{l}]{\mathrm{uu}} A_3 \xrightarrow[\mathrm{ll}]{\mathrm{u}} A_1 \xrightarrow[\mathrm{l}]{\mathrm{ul}} A_4 \xrightarrow[\mathrm{lu}]{} A_5 \xrightarrow{\mathrm{u}} A_2 \\
&\quad \xrightarrow[\mathrm{l}]{} A_0 \xrightarrow[\mathrm{ll}]{} A_1 \xrightarrow{\mathrm{uu}} A_6 \xrightarrow[\mathrm{l}]{} A_3 \xrightarrow{\mathrm{u}} A_0 \\
&A_0 \xrightarrow{\mathrm{uu}} A_3 \xrightarrow[\mathrm{ll}]{} A_6 \xrightarrow{\mathrm{u}} A_1 \xrightarrow[\mathrm{l}]{} A_0 \xrightarrow[\mathrm{lu}]{} A_2 \xrightarrow{\mathrm{ul}} A_5 \xrightarrow[\mathrm{l}]{} A_4 \\
&\quad \xrightarrow[\mathrm{ll}]{\mathrm{u}} A_1 \xrightarrow[\mathrm{l}]{\mathrm{uu}} A_3 \xrightarrow[\mathrm{lu}]{\mathrm{u}} A_2 \xrightarrow[\mathrm{l}]{\mathrm{ul}} A_4 \xrightarrow{\mathrm{u}} A_0 \\
&A_0 \xrightarrow[\mathrm{lu}]{\mathrm{uu}} A_9 \xrightarrow{\mathrm{u}} A_2 \xrightarrow{\mathrm{uu}} A_8 \xrightarrow[\mathrm{l}]{} A_3 \xrightarrow[\mathrm{lu}]{} A_9 (\xrightarrow{\mathrm{u}} A_2 \xrightarrow[\mathrm{l}]{} A_0) \\
&A_0 \xrightarrow[\mathrm{ll}]{\mathrm{ul}} A_{10} \xrightarrow{\mathrm{u}} A_1 \xrightarrow{\mathrm{ul}} A_7 \xrightarrow[\mathrm{l}]{} A_4 \xrightarrow[\mathrm{ll}]{} A_{10} (\xrightarrow{\mathrm{u}} A_1 \xrightarrow[\mathrm{l}]{} A_0).
\end{aligned} \tag{1}$$

Transitions in brackets are used to return to the initial state A_0.

With each of these transitions only one of the inputs of an SE is affected by a particular action (beginning or end of a packet). Thus, a specific state transition can only be initiated for all SE of a single stage simultaneously, but not for all SEs of the entire network. Accordingly, the number of tests is of $O(\log N)$. All other SEs which are not in the stage which is tested assume the states A_6 or A_0 which are fault free as proved in the first test phase. This is shown by the following lemma.

Lemma 1 *If at all SEs $p^{(s)}$ of a stage s data packets enter and leave on the same input and output, respectively, all other SEs $p^{(s')}$ with $s' \neq s$ are either in state A_0 or A_6.*

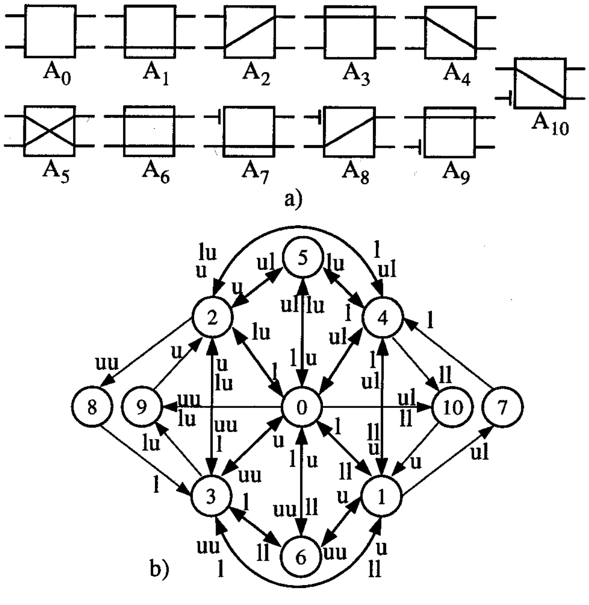


Figure 2: a) Possible states for an SE and b) state transition diagram.

Proof: Without loss of generality it is assumed that only those input links $i^{(s)}$ are affected for which $i_{n-s-1} = 0$ holds. Equivalently, $i_{n-s-1} = 0$ is assumed to hold for the output links $i^{(s+1)}$. Since only $N/2$ links are affected for which $i_{n-s-1} = 0$ holds, there are $N/4$ pairs of indices which differ in exactly one bit i_{n-j-1}, $j \neq s$, and the corresponding packets are inputs of the same SE in stage $n-j-1$. Thus, the SEs in stage $n-j-1$ either have two inputs nor none. The same holds if other values are chosen for the bit i_{n-s-1} of the input and output indices. □

An example illustrates the tests for the second test phase for $N = 8$ and the state sub-sequence $A_0 - A_3 - A_6 - A_1$ in stage 1, as depicted in Fig. 3. First, four packets represented by their routing headers are applied resulting in the solid paths and setting all SEs in stage 1 test to A_3. Then, while holding the previous paths (denoted by *), a second set of four packets is applied resulting in the dashed paths and state A_6 in stage 1. Finally, by dropping the solid paths (denoted by □) and holding the dashed ones, all SEs in stage 1 get in state A_1. Obviously, the SEs in stage $s = 1$ assume the desired states and all other SEs either are in state A_0 or A_6.

Since only the functionality of an SE is considered for this test procedure it clearly is independent of the actual hardware implementation.

3 Test Vector Generation

In this section, a labelling algorithm [12] for deriving the input vectors for the tests for locally controlled MINs is outlined. The algorithm can be described conveniently as a node labelling procedure on a binary n-cube. The nodes of the graph are indexed by v, $0 \leq v < 2^n = N$, and represent those input links of the MIN which have the same index $i^{(0)} = v$. Labels assigned to the nodes ($lab(v)$) represent the data words and the routing headers of the packets applied

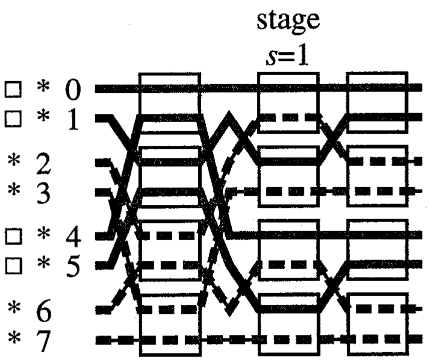


Figure 3: Example for the second test phase on stage $s = 1$, the state sequence $A_0 - A_3 - A_6 - A_1$ and $N = 8$.

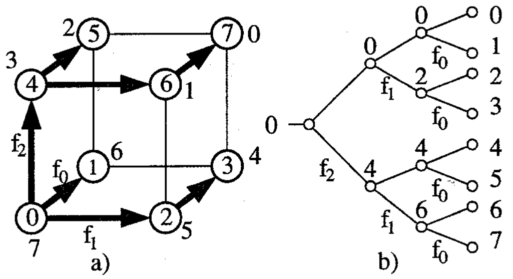


Figure 4: a) Example of labelling algorithm, and b) its dependence graph.

to the corresponding input links. The edges of the graph are incident to nodes the indices of which differ exactly by the value 2^k, $0 \leq k < n$, and thus can be associated with the SEs in network stages $s = k$. The value k is called the dimension of the link.

The procedure starts at an initial node with an initial label and visits every node exactly once. From an already visited node v_{pre} all nodes v_{suc} not yet visited are visited, if v_{pre} and v_{suc} are incident to a common edge. The label of v_{suc} is derived from v_{pre} by a labelling function depending on the dimension k, i.e., $lab(v_{suc}) = f_k(lab(v_{pre}))$. An example is given in Fig. 4 a) for $N = 8$ and the routing headers for bringing all SEs to state A_6, i.e., the routing headers are identical to the index of the respective input link. The labelling function is defined by

$$lab(v_{suc}) = f_k(lab(v_{pre})) = lab(v_{pre}) \oplus 2^k, \quad 0 \leq k < n.$$

The starting node is 0 with label $N - 1$. The arrows in Fig. 4 a) denote the labelling operations.

During the labelling procedure only a subset of all edges is traversed. However, the labels of all other nodes which are incident to a common edge have to be linked by the labelling function. The necessary and sufficient condition for this is given in the following lemma.

Lemma 2 *The labels of all pairs of nodes incident to a common edge are linked by the labelling function after the labelling algorithm, iff all pairs of labelling function are commutative, i.e.,*

$$f_i(f_j(x)) = f_j(f_i(x)), \quad 0 \leq i, j < n. \qquad (2)$$

Proof: For proving the necessity of (2), first consider an arbitrary node u with the distance two from the starting node v. The label $lab(u)$ is obtained from $lab(v)$ by $lab(u) = f_j(f_i(lab(v)))$. Both nodes u and v have a common edge with node w, the label of which is obtained from $lab(v)$ by $lab(w) = f_j(lab(v))$. Although the edge between w and u is not traversed by the algorithm, the respective labels have to be linked by the labelling function f_i, i.e., $lab(u) = f_j(f_i(lab(v))) = f_i(lab(w)) = f_i(f_j(lab(v)))$. Since this has to hold for all subcubes with distance two, the necessity of (2) is shown.

The sufficiency can be shown by considering all pairs of nodes u and w common to an edge e_j of dimension j. The labels of these nodes are obtained from $lab(v)$ by two sequences of labelling functions, i.e., $lab(u) = F_u(lab(v)) = f_i(\cdots(f_k(lab(v)))\cdots)$ and $lab(w) = F_w(lab(v))$. Again, $F_u(lab(v)) = f_j(F_w(lab(v)))$ has to hold. Since the set of edge dimensions by which u and w are reached from v are the same except for j, $f_j F_w$ can be transformed into F_u by exchanging pairs of functions f_k and f_l. Thus, the sufficiency of (2) is shown. □

The dependence graph of the labelling algorithm with its tree-like structure is shown in Fig. 4 b). Edges which are not labelled represent the identity function. The next section will show how this dependence graph can easily implemented as hardware architecture.

4 BIST Architecture

In order to achieve a BIST, the dependence graph of Fig. 4 b) can be implemented as hardware architecture as shown in Fig. 5 a). The basic component is a test generation module (TGM) which receives one input and produces two outputs one of which is identical to the input, the other is modified by the labelling function. Since all TGMs of a particular level of the tree architecture (corresponding to the dimension of an edge of the n-cube) realize the same function f_k, only one set of signals is sufficient to control all TGMs of that level. Thus, for controlling the entire tree architecture n sets of storage elements are required. Additionally, one storage element has to be provided for the initial label. In total, the tree architecture comprises of $N - 1$ TGMs and $n + 1$ sets of storage elements. In what follows, the tree architecture is outlined for both cases of bit-serial and block-sequential data format.

4.1 Bit-Serial Data Format

Bit-serial architectures are frequently used in order to save routing costs [13] despite of the increased block pipeline period which has to be tolerated. A TGM for a one bit signal is called test generation unit (TGU). For modifying the data in a TGU, an EXOR-gate (see Fig. 5 b)) is used, since this has a low transistor count and just one control signal is required. For breaking up the long data paths a buffer is required at each output of a TGU. For T tests with b bits each $Tb(n + 1)$ bits of storage capacity are required in addition to the $N - 1$ TGUs of the tree architecture.

In order to compare the amount of hardware of the proposed tree architecture (TA) with the method of simply storing the TbN bits for the N inputs separately we consider the

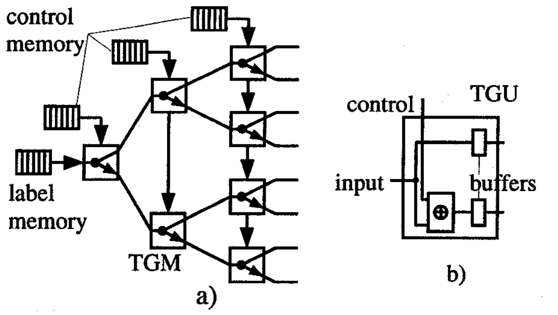


Figure 5: a) BIST architecture for $N = 8$ and b) structure of TGU.

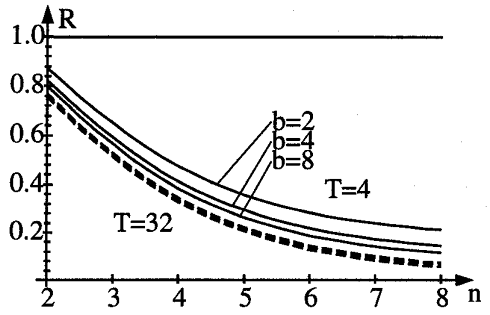


Figure 6: Ratio R of the transistor count for $b \in \{2, 4, 8\}$, $T = 4, 32$ over n.

ratio R of the number of transistors for both methods. For realizing the storage elements read only memories (ROMs) are suited best. The $n+1$ ROM-elements comprise of $T \times b$ arrays of one-transistor cells with appropriate row and column selection implemented as shift register chains. Further details are omitted due to lack of space. We do not take into account the routing area which is higher for the TA than for the storage method due to the global wires required. The routing area is proportional to the areas of the TGUs. The transistor count is dominated by the amount of hardware for the control signal memory and thus the routing area will not yield a significant contribution to the total area.

Fig. 6 shows the ratio R for different values of T and b over n. It becomes apparent that even for small values of the three parameters the TA is significantly more efficient than the storage method. For growing parameter values (especially T), as is usually the case, the advantage of the tree architecture becomes even more significant.

4.2 Block-Sequential Data Format

In order to reduce the block pipeline period of an architecture, a block-sequential data format is advantageous. There, each of the T data packets comprises of q data words which are b bits wide. In the most general case, each TGM must be composed of b TGUs, one for each bit line. The resulting transistor count ratio R is depicted in Figs. 7 a) and b) for the parameters $T = 4, 64$, $b \in \{2, 4, 8, 18\}$ over n for $q = 1$ and $q = 4$, respectively. It turns out, that in this case the TA is just moderately efficient, because only for relatively large values of T and n a significant reduction of the transistor count can be achieved.

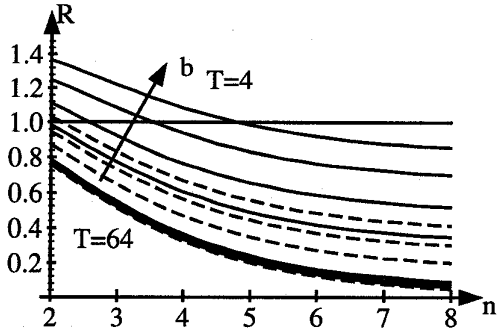
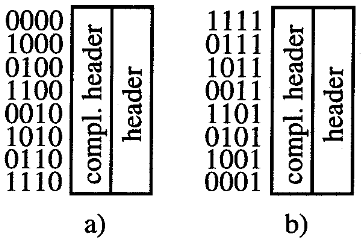


Figure 7: Ratio R of the transistor count for $b \in \{2, 4, 8\}$, $T = 4, 32$ over n for a) $q = 1$ (solid lines) and b) $q = 4$ (dashed lines).



0000
1000
0100
1100
0010
1010
0110
1110

compl. header

header

a)

1111
0111
1011
0011
1101
0101
1001
0001

compl. header

header

b)

Figure 8: The two types of different data words.

However, the complexity of the network can be reduced significantly, if the fault detection procedure presented in Section 2 is considered. In the following we will show this for the generation of the routing header and the additional data words of the pay load information, which have been described briefly in Section 2.

For generating the routing header only one TGU per TGM is required since only one particular bit is affected in each level of the tree architecture. This illustrates the commutativity feature stated in Lemma 2. For generating the data word which is the bit-wise complement of the routing header the same TGU can be used since simply an appropriate initial data word is required. The generation of the all zero or all one data words and the data words for detecting bridging faults more intricate considerations are necessary. Basically, two different sets of data words are required which are shown in Fig. 8 for $b = 8$. Except for the complemented routing header the data words of the payload have to be the bit-wise complement of each other. One type of these data words has to be comprised in packets on input links the dyadic index of which has even parity, the other set of data words belongs to packets on the other inputs.

Each of the data words can again be generated by a tree-like structure, since the bits have different values only if their dyadic index differs in a particular bit. The most efficient way to generate these data words is to embed the tree-like structure in the last $\lceil \log_2 b \rceil$ stages of the original TA which is used for the routing headers, as shown in Fig. 9, where only the uppermost TGMs of the last $\lceil \log_2 b \rceil = 3$ stages are depicted. According to their function there are two types of

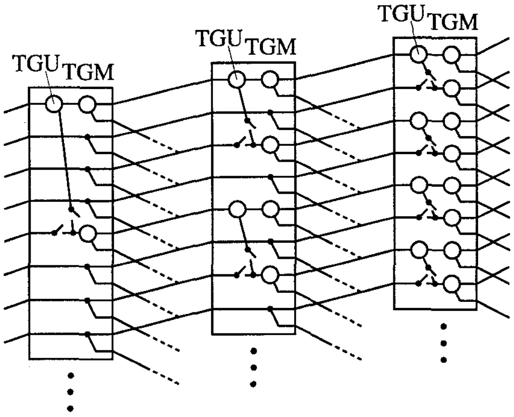


Figure 9: Schematic representation of the TGMs in the last $\lceil \log_2 b \rceil$ levels of the TA for block-sequential data format.

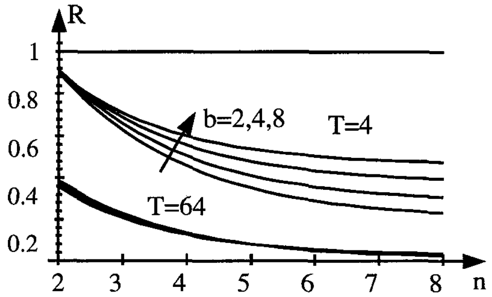


Figure 10: Transistor count ratio for $T = 4, 64, b \in \{2, 4, 8\}$ and $q = \lceil \log_2 b \rceil + 3$.

TGUs. First, there are those TGUs which are required for distributing the initial value in such a way that at the output of the TA the two types of data words are assigned to the inputs the indices of which have the respective parity. The second type of TGUs has to generate the different values within a data word. For this, two switches are needed to overwrite an existing value on a bit line. Since in level $n - \lceil \log_2 b \rceil - 1$ only one TGU of the second types exists an initial value has to be propagated in the preceding stages only on one bit line by one TGU of the first type per TGM.

Furthermore, the amount of storage for the control signals can be reduced significantly, since these data words are part of all data packets, so that they can be produced periodically. Without going into further details, Fig. 10 shows the ratio R of the transistor count for the TA and the simple storage of the test inputs for different parameter values. It becomes apparent that also for the case of block-sequential data format the TA is an efficient alternative to the storage architecture for the proposed testing procedure. This is particularly true for large values of T which usually occur since the number of tests in of $O(\log N)$ with a quite large constant.

By implementing the same structure for generating the network outputs in the fault free case and comparing these to the actual network outputs a simple and efficient built-in self test architecture is obtained. Certainly, the self test circuitry can also be affected by hardware defects, but the probability that the effects of the defects in the actual interconnection network and the test circuitry compensate and make the defects undetectable is practically zero.

5 CONCLUSION

In this paper a state-based test procedure for multistage interconnection networks with packet-oriented data transmission is presented. The entire functionality can be tested independently of the actual hardware implementation. Based on this procedure a tree-like hardware architecture is proposed which is suited for the implementation of a built-in self test due to its low hardware complexity compared to the only known method for pseudoexhaustive test, which is the storage of the test inputs.

References

[1] T. E. Mangir. Source of failures and yield improvement for VLSI and restructurable interconnects for RVLSI. *Proc. IEEE*, 72(6):690–708, 1984.

[2] F. A. Tobagi. Fast packet switch architectures for broadband integrated services digital networks. *Proc. IEEE*, 78(1):133–167, Jan. 1990.

[3] J. Hui. Switching integrated broadband services by sort-banyan networks. *Proc. IEEE*, 79(2):145–154, 1991.

[4] D. P. Agrawal. Testing and fault-tolerance of multistage interconnection networks. *IEEE Computer*, 15(4):41–53, 1982.

[5] N. Davis IV, W. Hsu, and H. Siegel. Fault location techniques for distributed control interconnection networks. *IEEE Trans. Comp.*, C-34(10):902–910, 1985.

[6] A. Mourad, B. Özden, and M. Malek. Comprehensive testing of multistage interconnection networks. *IEEE Trans. Comp.*, 40(8):935–951, 1991.

[7] E. G. Bernard. Efficient fault location for globally controlled and comparison-based multistage interconnection networks. accepted for publication in *IEEE Transactions on Computers*, 1995.

[8] M. C. Pease. The indirect binary n-cube microprocessor array. *IEEE Trans. Comp.*, C-26(5):458–473, 1977.

[9] E. G. Bernard, M. Sauer, and J. A. Nossek. Fault location in bitonic merging networks. In *Proc. ECCTD'95*, pages 929–932, August 1995.

[10] S. Naito and M. Tsunoyama. Fault detection for sequential machines by transition-tours. In *Proc. FTCS 81*, pages 238–243, 1981.

[11] T.-Y. Feng and C.-L. Wu. Fault-diagnosis for a class of multistage interconnection networks. *IEEE Transactions on Computers*, C-30(10):743–758, 1981.

[12] E. G. Bernard, M. Sauer, and J. A. Nossek. Fault detection in bitonic merging networks. In *Proc. ECCTD'93*, pages 665–670, 1993.

[13] F. A. Tobagi, T. Kwok, and F. M. Chiussi. Architecture, performance, and implementation of the tandem banyan fast packet switch. *IEEE Jou. Sel. Areas Comm.*, 9(8):1173–1193, 1991.