JOURNAL OF ELECTRONIC TESTING: Theory and Applications, 1, 103-123 (1990)

A Hierarchical Test Generation Methodology for Digital Circuits

DEBASHIS BHATTACHARYA
Department of Electrical Engineering, Yale University, New Haven, CT 06520

JOHN P. HAYES
Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109

Received March 13, 1989. Revised December 18, 1989 and January 5, 1990.

Abstract. A new hierarchical modeling and test generation technique for digital circuits is presented. First, a high-level circuit model and a bus fault model are introduced—these generalize the classical gate-level circuit model and the single-stuck-line (SSL) fault model. Faults are represented by vectors allowing many faults to be implicitly tested in parallel. This is illustrated in detail for the special case of array circuits using a new high-level representation, called the modified pseudo-sequential model, which allows simultaneous test generation for faults on individual lines of a multiline bus. A test generation algorithm called VPODEM is then developed to generate tests for bus faults in high-level models of arbitrary combinational circuits. VPODEM reduces to standard PODEM if gate-level circuit and fault models are used. This method can be used to generate tests for general circuits in a hierarchical fashion, with both high- and low-level fault types, yielding 100 percent SSL fault coverage with significantly fewer test patterns and less test generation effort than conventional one-level approaches. Experimental results are presented for representative circuits to compare VPODEM to standard PODEM and to random test generation techniques, demonstrating the advantages of the proposed hierarchical approach.

Key words: digital circuits, fault modeling, hierarchical testing, high-level circuit models, test generation.

1. Introduction

The great complexity of modern digital circuits has made classical test generation methods too time consuming for many applications. While large circuits are often designed and analyzed at a register or functional level of abstraction, classical algorithmic test generation techniques usually require a more complex *gate-level* model of the circuit in which logic gates (AND, OR, NOT, NAND, etc.) are the primitive components. The resulting increase in the number of components in the circuit model—a gate-level model contains perhaps ten times as many components as the equivalent register-level model—tends to significantly increase the test generation time. Moreover, important simplifying features of the circuit under test, such as the presence of a hierarchical structure or repeated subcircuits, must be ignored for test generation purposes.

Existing test generation algorithms usually assume a single line in a gate-level model of the circuit under test to be permanently stuck at logic level 0 or 1. This is known as the *single-stuck-line* or SSL fault model. Two well-known test generation algorithms that employ the SSL model are the D-algorithm [14] and PODEM [9]. The major problem of generating tests using a gate-level model is the exponential growth of computation time as the number of circuit components increases.

Several proposals have been made to reduce the complexity of the test generation problem by using high-level circuit models and extensions of classical test generation algorithms [15, 12, 11]. A major drawback of these approaches is their mixing of concepts from different modeling levels. Although the basic modules in the circuit description are largely at a level higher than the gate level, their inputs and outputs are defined, implicitly or explicitly, in terms of single-bit lines. More recently, various other high-level test generation techniques have been proposed that differ significantly from the classical approaches [5, 16, 18]. An important class of such techniques [7, 8, 16] construct pseudo-exhaustive test sets for iterative array circuits using only a high-level description of the input/output behavior of the modules in the array. Another method, proposed in [18], requires an instruction-level description of the circuit under test. Two major limitations of these techniques are their lack of generality, and the difficulty of comparing their fault coverage, i.e., the percentage of faults detected, to fault coverage obtained using more traditional test generation techniques.

In this article, we develop a new test generation algorithm called VPODEM based on the high-level circuit and fault modeling methodology introduced by us in [2]. VPODEM can generate tests for general combinational circuits using a bus-fault model, and takes explicit advantage of any hierarchical structure present in the circuit. In the special case of k-regular circuits [19, 20], which includes iterative array circuits as a proper subclass, our approach yields an efficient new high-level circuit representation called the modified pseudo-sequential or MPS model. Moreover, VPODEM reduces to standard PODEM if the gate-level circuit model is used. Hence, a hierarchical testing strategy which generates tests using two or more levels of abstraction can be adopted for general circuits. First, tests are generated for a high-level model of the circuit, and the SSL fault coverage provided by these tests is estimated. Next, using the same algorithm, tests are generated for the remaining undetected SSL faults in the gate-level model, so 100 percent coverage of detectable SSL faults can always be obtained. Thus, our approach minimizes the mixing of concepts from different levels, and reduces to a traditional test generation algorithm when low-level circuit and fault models are used. Furthermore, the number of test patterns generated by this hierarchical approach is significantly smaller than the number of patterns generated by conventional algorithmic or random test generation techniques.

Section 2 first summarizes a vector notation used to represent faults and tests in the high-level models. It then presents the proposed high-level circuit and fault modeling techniques. The modeling techniques are illustrated in section 3 by a detailed study of high-level model construction for k-regular circuits. Section 4 describes the VPODEM test generation algorithm. Key implementation details of VPODEM and the results of experiments using VPODEM are provided in section 5.

2. Circuit and Fault Models

We begin with a brief summary of the vector sequence (VS) notation, which provides a useful and compact way of describing circuit behavior hierarchically. The VS notation, first proposed in [10] and subsequently expanded by us in [2], employs $n \times m$ arrays called *vector sequences* (VSs), where n and m are dynamic parameters, as the fundamental information units. This notation is primarily intended to represent input/output behavior of circuits at different levels of complexity. For example, the behavior of the gate-level circuit of figure 1(a) can be written in the VS form as

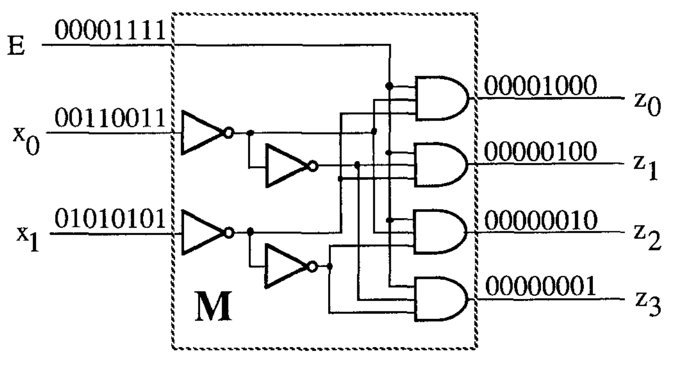
$$\begin{bmatrix} 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \end{bmatrix} \Bigg/ \begin{bmatrix} 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix},$$

where the left 3×8 array is a VS denoting the input (test) vectors, and the right 4×8 array is a VS denoting the corresponding output (response) vectors. The same circuit can be represented at a higher level as shown in figure 1(b), with the following VS pair representing its input/output behavior:

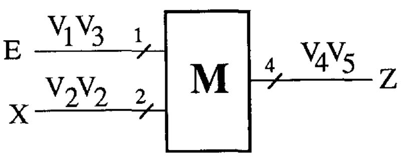
$$\begin{bmatrix} V_1 & V_3 \\ V_2 & V_2 \end{bmatrix} \Big/ [V_4 \quad V_5],$$

where

$$V_1 = [0 \quad 0 \quad 0 \quad 0],$$



(a)



(b)

Fig. 1. Two-to-four decoder: (a) gate-level model; (b) high-level model.

$$V_2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix},$$

$$V_3 = [1 \quad 1 \quad 1 \quad 1],$$

$$V_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$V_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

Six operators, called *external time expansion* $\cdot$, *external space expansion* $\odot$, *internal time expansion* $\times$, *internal space expansion* $\otimes$, *select* S_α, and *project* P_α have been defined on VSs [10, 2] which provide the ability to expand or contract VSs in various complex ways. The effect of applying these operators to VSs is illustrated below using the two VSs S and T where

$$S = T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}:$$

$$S \cdot T = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \quad S \odot T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$S \times T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad S \otimes T = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix},$$

$$S_{(1)}(S) = [1 \quad 0], \quad P_{(2)}(S) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Note that an index set α is provided for the operator select (project), which identifies the rows (columns) of the original VS that constitute the resultant VS. Further details about the VS operators can be found in [3].

Certain "standard" vector sequences have been defined using the expansion operators alone. For example, 0_n represents the vector of n 0s; 1_n is the vector of n 1s; A_n is the vector of size n with alternating 1s and 0s; C_n is the $n \times 2^n$ counting sequence (which is the output of an n-bit counter), and D_n is the $n \times n$ diagonal sequence.

VSs provide a compact way of representing the input/output behavior of components in a circuit. A VS expression of the form $V = f(S, T)$ can be interpreted as describing the behavior of a functional block whose inputs are the VSs S and T, and whose output is the VS V; see figure 2(a). For example, if we select f to be the operator $\odot$, then the corresponding functional block performs a simple merger of two buses into one, as shown in figure 2(b). The output bus in this case is a juxtaposition of the two input buses. The element of figure 2(b), which is primarily an abstract notational device, is called a *merge element*, and provides a basic way of representing the construction of buses at higher levels of abstraction. The converse operation, viz, the splitting of a bus into two or more buses, can be performed by another class of abstract logic elements called *fanout elements* (figure 2(c)) which are examined further below. Merge and fanout elements allow representation of arbitrary changes in bus sizes in a simple manner, and play an important role in the construction of high-level models of general circuits.

We now present our bus-oriented circuit and fault modeling techniques. We consider two main levels of

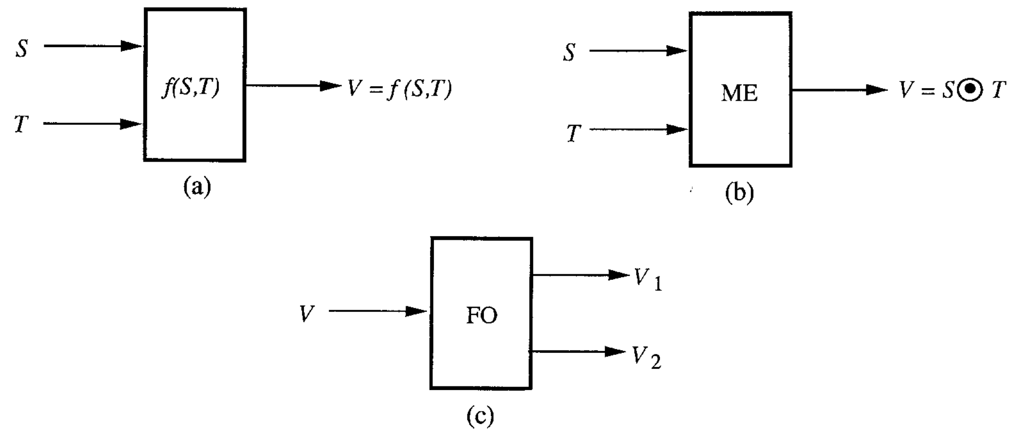


Fig. 2. Interpretation of VS operators as functional blocks: (a) general operator element f; (b) merge element; (c) fanout element.

complexity, dubbed high and low. The high-level circuit model, denoted M^H, is composed of components such as word-gates, multiplexers, adders, registers, etc., connected to one another using buses of appropriate sizes. This level corresponds approximately to the register or functional level of circuit design. When the bus size is reduced to one, a word gate becomes a single gate, and a bus is the same as a line; this results in the low-level circuit model M^G which corresponds to the classical gate level. This multilevel approach makes truly hierarchical modeling of circuits possible while minimizing the mixing of levels in the circuit models, and helps in significantly reducing the overall test generation effort.

The transition from the low to the high level allows buses to fan out and merge in new, and possibly complex, ways that are unique to the high-level models. A general fanout element is a component with one input bus X and n output buses $Z_1, \ldots, Z_n$ such that an input vector V applied to bus X produces output vectors $S_{\alpha_i}(V)$ on Z_i for $1 \leq i \leq n$, where S_{α_i} is the select operator. Hence, the behavior of complex fanout elements can be described simply by specifying the index sets α_i, $1 \leq i \leq n$, corresponding to the various output buses of the elements. Examples of fanout elements and other interconnection structures commonly used by our model are presented in figure 3. A fanout element is said to be *regular* if $S_{\alpha_i}(V) = V$ for $i = 1, \ldots, n$. This is a generalization of the simple types of fanout that occur in gate-level circuit models; note that all fanout elements in a gate-level model are regular with input and output buses of size one. The merge element introduced earlier can be similarly used to solve the problem of handling mergers of small buses to form bigger ones.

Our modeling process for a general combinational circuit consists of two major steps, viz, high-level model construction for the main types of subcircuits present in the circuit, and interconnecting these high-level models using appropriate buses and fanout elements. In the first step, we recognize the fact that different parts of a circuit have various degrees of regularity in their structures, and may need to be treated differently. The information about such regularity is typically available in a hierarchical CAD-based design environment, and is lost when the circuit is "flattened" to the gate-level. Thus, our modeling approach needs, and explicitly uses, hierarchical design information to create the high-level models, unlike other modeling approaches that construct the netlists needed for testing from flattened circuits.

We can identify three major classes of subcircuits based on their structures. For subcircuits that are regular or iterative in nature, we construct a new high-level representation called the MPS model, which is discussed in detail in the next section. Subcircuits that are not regular but have their input/output lines organized into suitable buses, are retained as primitive functional blocks. In contrast, for a subcircuit that is neither regular nor bus-structured (as is often true for control logic), we treat individual lines as buses of size one, which in effect makes the high-level and gate-level models identical. Once the high-level modules have been defined, they are interconnected by buses of appropriate size, and—where necessary—by fanout elements. In general, it is desirable to use buses that are as large as possible, consistent with the bus sizes of the high-level modules.

To illustrate the modeling process, let us apply it to some standard MSI circuits [17]. The gate-level model M^G of the 74157 4-bit 2-to-1 multiplexer is shown in figure 4(a). The corresponding high-level model M^H appears in figure 4(b). The subcircuits marked $M_{2,1}$ through $M_{2,4}$ of M^G form a simple iterative array, and are represented in M^H as a word-oriented subcircuit M_2^4. As shown in figure 4(b), a detailed model of the control logic consisting of three inverters is retained in M^H. Moreover, several fanout elements that change the size of the control buses from one to four need to be introduced in M^H. These elements marked FO_1, are needed to interface the control logic with buses of size one to the word-oriented high-level module M_2^4 with buses of size four. Obviously, these fanout elements are not regular in the sense defined above. On the other hand, the fanout elements marked RFO, in figure 4(b), are regular. As a second example, consider the standard 74181 ALU/function generator [17], whose M^G and M^H models appear in figure 5. The subcircuits M_1 through M_4 of M^G are again grouped into word-oriented high-level subcircuit (M^4) in M^H. Note that the fanout elements in M^4 marked RFO are regular. The subcircuit M_5 of M^G does not possess much regularity, but has bus-structured input/output connections, and is easily seen to be a modified carry-lookahead adder. Hence, this portion of the circuit is modeled as a primitive component MCLA with four input buses, two of size four and two of size one, and four output buses, as shown in figure 5(b). M^H for the 74181 ALU/function generator is completed by connecting M^4 and MCLA via two four-bit buses *B1* and *B2*; we do not need fanout elements in this final step.

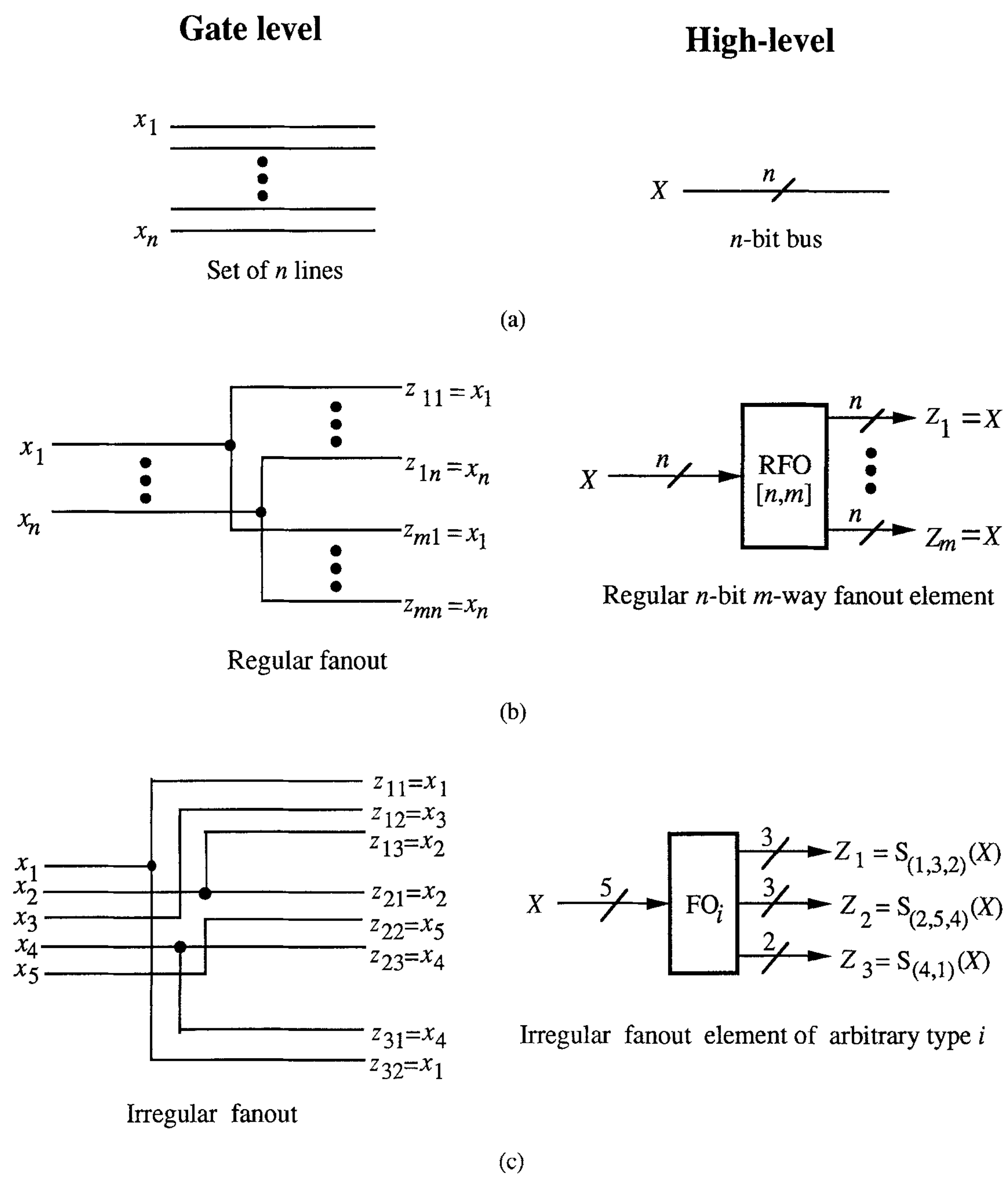
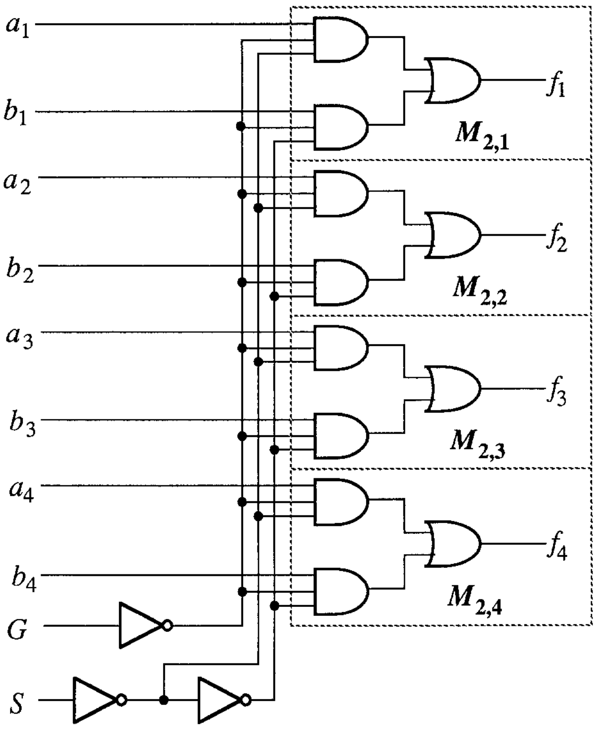


Fig. 3. Interconnection structures in gate-level and high-level circuit models: (a) simple connection (bus); (b) regular fanout; (c) irregular fanout.

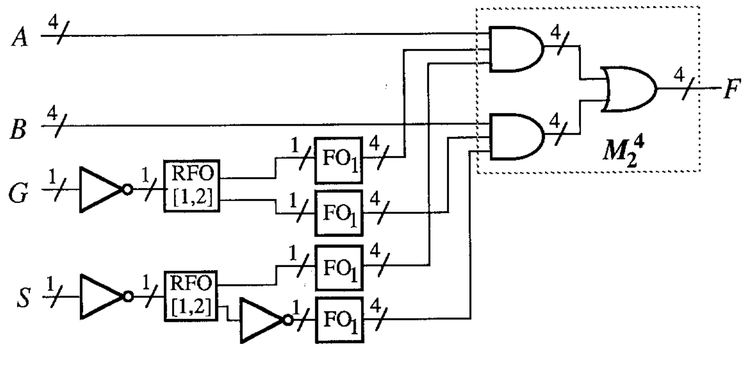
In order to make the test generation procedure hierarchical in the same sense as the circuit model, the fault model used also needs to be hierarchical. We now introduce such a fault model, which is a generalization of the SSL model from single lines to buses. This *bus fault* model assumes that buses instead of individual lines in the high-level models can be stuck at constant vector values. The most general kind of bus fault thus allows arbitrary subsets of lines in a bus to be stuck at 0, stuck at 1, or fault free. In the VS notation, fault F on bus B may be represented by a vector of the form

$$F = b_1 \odot b_2 \odot \ldots \odot b_n$$

where $b_i = 0$ (1) indicates that line i is s-a-0 (s-a-1), and $b_i = d$ represents a fault-free line. It is obvious that up to $3^n - 1$ such bus faults can be associated



(a)



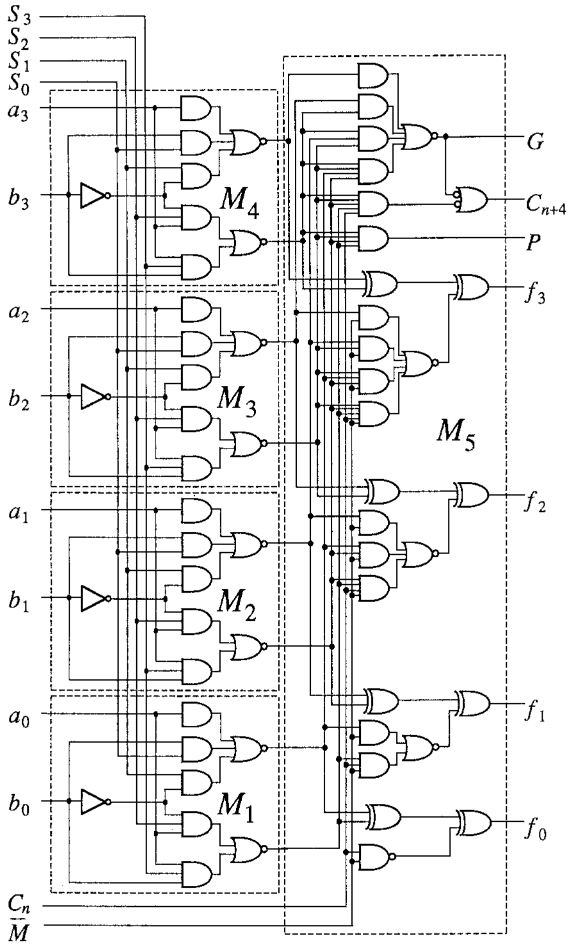
(b)

Fig. 4. Four-bit 2-to-1 multiplexer: (a) gate-level model M^G; (b) high-level model M^H.

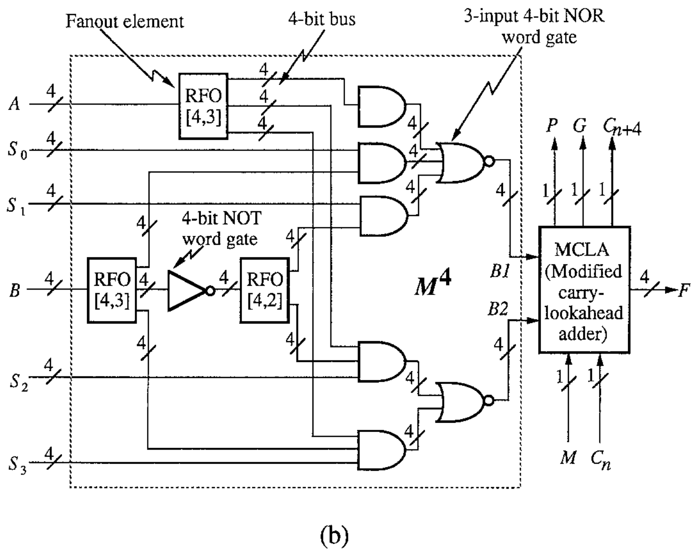
with a bus of size n. However, as we demonstrate below, we need only a small subset of the possible bus faults for test generation purposes.

The bus faults that appear to be most useful are classed as *total bus faults*, defined as follows. An n-bit bus B is *(totally) stuck-at-0* if all n lines in B are stuck at logic level 0; it is *stuck at 1* if all n lines are stuck at 1. Obviously, total bus faults reduce to SSL faults if the bus size n is reduced to one, and thus constitute a natural generalization of the SSL fault model. As shown later, for certain regular circuits, tests generated for total bus faults in their high-level models can be guaranteed to detect all SSL faults on individual lines of the corresponding buses. Furthermore, experimental results presented in section 5 show that for general circuits with some regularity in their structures, test sets generated for total bus faults provide moderate to good SSL fault coverage, while reducing the test set size as well as the test generation time substantially.

The primary advantage of high-level models is the simplification resulting from the presence of fewer (primitive) components, connections, and faults. The



(a)



(b)

Fig. 5. The 74181 ALU/function generator: (a) gate-level model M^G; (b) high-level model M^H.

simplifications introduced by our approach can be seen from comparing the two models of the 74181 ALU/function generator in figure 5. The gate-level model contains 201 lines, and consequently has 402 distinct SSL faults. The high-level model, on the other hand, contains only 26 buses and 52 total bus faults.

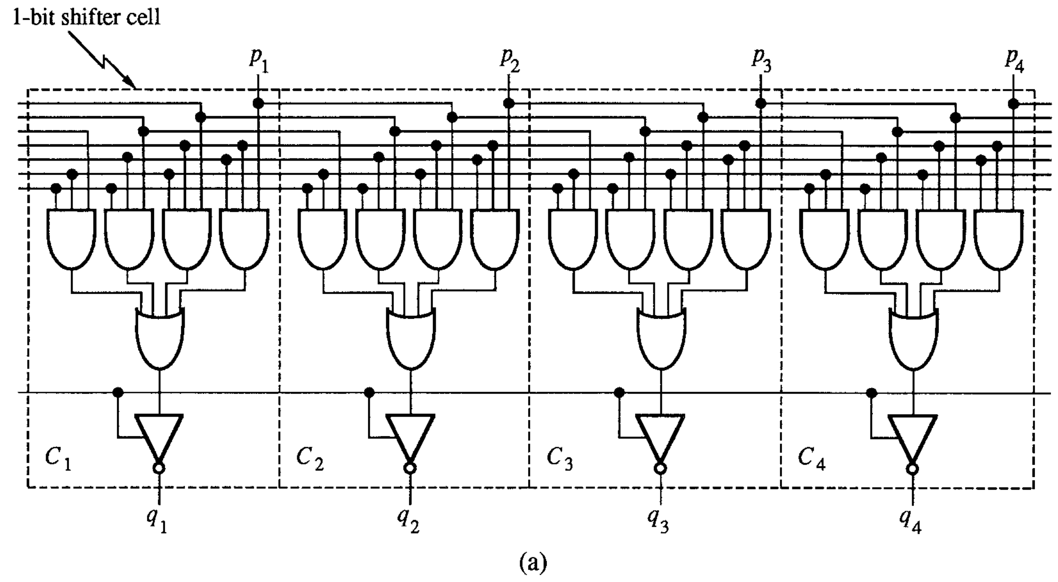
The simplicity of any high-level model is strongly related to the presence of repeated subcircuits in the gate-level model, and the regularity of their interconnections. Indeed, if there is little or no regularity of this kind, as in "random" combinational logic, then there may be no useful circuit model at any level higher than M^G. With the increasing integration of digital systems, there has been a trend toward building circuits from standard high-level modules interconnected in a regular manner. The modeling technique proposed here is particularly useful in generating tests for such circuits. In the next section, it is applied to a particular class of highly regular circuits to illustrate its advantages over traditional gate-level methods for test generation purposes. Note, however, that our general methodology can take full advantage of the more modest degree of regularity found in typical designs.

3. Regular Circuits

To illustrate the ability of our approach to exploit a circuit's regularity to simplify test generation, we first apply it to the class of k-regular circuits recently introduced by Y. You [19, 20]. The best-known examples of such circuits are bit-sliced ALUs and controllers, which are 1-regular, and have long been studied under the heading of cellular or *iterative logic arrays* (ILAs). The concept of k-regularity extends the notion of regularity from such completely regular circuits as ILAs to more general circuits which are not so obviously regular. Intuitively, a k-regular circuit is a one-dimensional array of repeated groups of modules, each group consisting of k distinct module types. Formally defined, a one-dimensional array (cascade) circuit $C_{1:n}$ of n modules $C_1, \ldots, C_n$ is said to be *k-regular* if $C_{(i-1)k:ik}$ is isomorphic to $C_{ik:(i+1)k}$, for all $0 < i \leq \lfloor n/k \rfloor$, where $C_{i:r}$ denotes the subcircuit comprising C_i, C_{i+1}, $\ldots$, C_r and all their interconnections. Specific examples of such circuits are provided in figure 6. Figure 6(a) shows a shifter circuit based on the 74350 IC [17] which is easily seen to be 1-regular. Figure 6(b) shows a parity checker circuit that checks for even parity, and is based on the 74280 IC [17]. Note that the model in figure 6(b) is actually an intermediate-level circuit in which two module types A and B have been recognized, and is an example of a 2-regular circuit. In order to construct M^H for k-regular circuits, we may sometimes start with such an intermediate-level model instead of a gate-level model because useful structural information is preserved in the former.

The high degree of repetitiveness in the k-regular circuits suggests that the proposed hierarchical modeling technique may be very useful in reducing their test generation complexity. In fact, for the special case of 1-regular circuits in which the horizontal output lines from a module are directly connected to the corresponding horizontal input lines to the module, a high-level model M^H of the circuit can be easily constructed such that a test for a total bus fault on bus B in M^H detects all SSL faults that can be associated with the bus B. The subcircuit consisting of modules $M_{2,1}$: $M_{2,4}$ in the 4-bit 2-to-1 multiplexer of figure 4(a) is an example of such a 1-regular array. A procedure to construct high-level circuit models for general k-regular circuits is briefly described next, and is then illustrated with the 2-regular parity checker.

Let the general k-regular circuit under consideration consist of n modules, such that $n = qk - r$, $0 \leq r \leq k - 1$. Following the notation of section 2, M^G and M^H denote the low-level and high-level models of the circuit, respectively. The superscripts G and H are also used to differentiate between components or buses of the low-level and the high-level models. Thus, M^G for a k-regular circuit consists of modules C_1^G through C_n^G connected to one another. M^H is obtained by grouping together the identical modules C_i^G, C_{i+k}^G, C_{i+2k}^G, $\ldots$ which are spaced periodically along the array, to form a word-oriented high-level module C_i^H. The different high-level modules are then connected together using buses of size q or $q - 1$ following a procedure called *pseudo-sequential contraction* (PSC) [3]. Briefly stated, this procedure forms buses from lines connected to the gate-level modules at intervals of k modules. These buses are then used to connect the high-level modules together, with a loop being introduced to represent the fact that identical modules occurring at intervals of k modules in the gate-level model are mapped onto the same high-level module. Four special fanout (FO) and merge (ME) components need to be introduced to perform this grouping of lines into buses in M^H. These are the fanout elements FO[MS, n] and FO[LS, n], and the merge elements ME[MS, n] and ME[LS, n] shown in figure 7. The parameter n indicates the size of the input bus in the case of the fanout elements, and the size of the output bus in the case of the merge elements. Note that one output bus of each fanout element is of



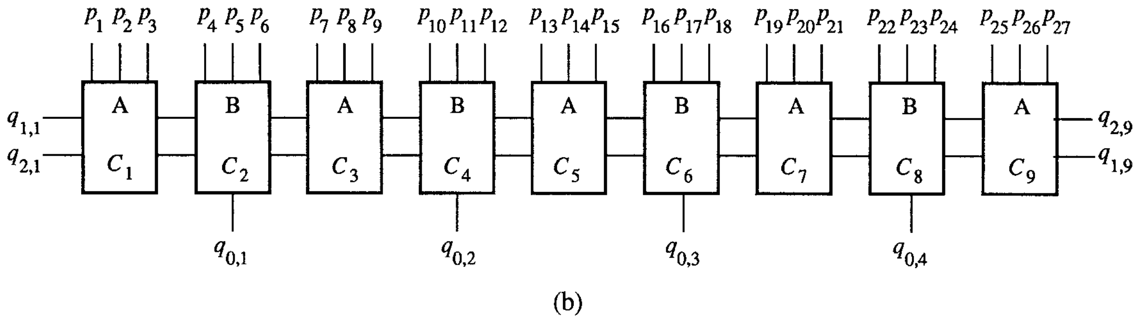


Fig. 6. Examples of *k*-regular circuits: (a) shifter (1-regular); (b) even parity checker (2-regular).

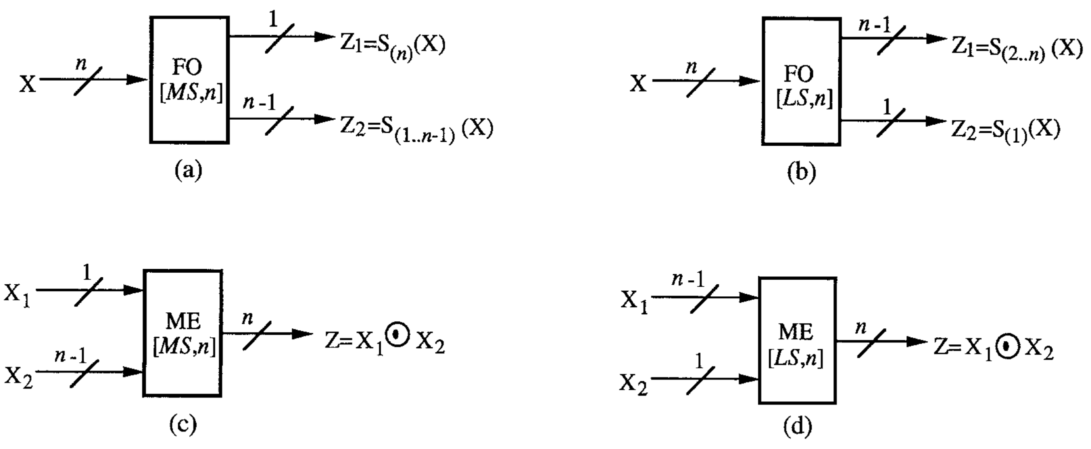
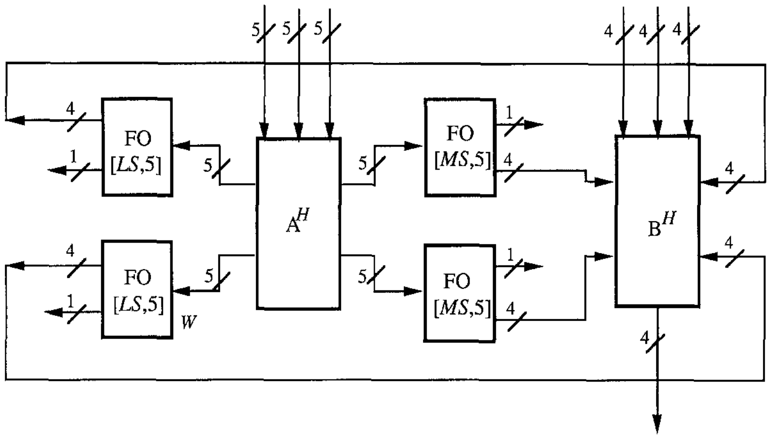


Fig. 7. The fanout and merge elements used in pseudo-sequential contraction: (a) FO[*MS*, *n*]; (b) FO[*LS*, *n*]; (c) ME[*MS*, *n*]; (d) ME[*LS*, *n*].

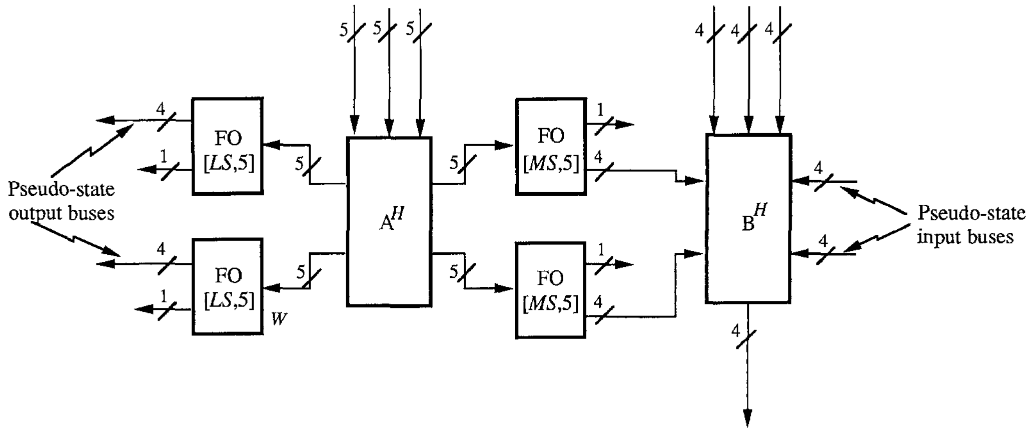
size one, while the other is of size $n - 1$. Parameters *LS* and *MS* indicate whether the output bus of size one is connected to the most significant or the least significant line of the input bus. Parameters *MS* and *LS* have a similar interpretation in the case of the merge elements, indicating whether the most or the least significant line of the output bus is connected to the input bus of size one.

The result of applying pseudo-sequential contraction to the parity-checker circuit of figure 6(b), is shown in figure 8(a). Since the circuit is 2-regular, only two high-level modules A^H and B^H corresponding to module types A and B, respectively, exist in M^H. The number of periods q is obviously five in this case. Merge (ME) elements are not needed, while special fanout elements of both types, viz, FO[*LS*, 5] and FO[*MS*, 5], are needed, as seen in figure 8. The interconnections between the high-level modules and the fanout elements, as generated by pseudo-sequential contraction, are shown in figure 8(a).

An important feature of M^H *for* k-regular circuits is the introduction of a loop, although no such loop exists in M^G. M^H is called *pseudo-sequential* (PS) due to the possible presence of loops; note, however, that it does not exhibit sequential behavior in the usual sense. It is easy to see that the pseudo-sequential high-level model contains a constant number of components whereas the number of components in the gate-level circuit model increases linearly with q. The bus size in M^H, on the other hand, increases with q. This trade-off of bus size for component count implies a significant reduction in the number of components in M^H compared to M^G, even for moderately large values of q. The only problem of using the pseudo-sequential



(a)



(b)

Fig. 8. (a) High-level (pseudo-sequential) model of parity checker of Fig. 6(b); (b) corresponding MPS model.

model directly for test generation purposes is the presence of loops in it, because conventional combinational test generation algorithms require acyclic circuit models. However, we can solve this problem in the standard way [4] by breaking the feedback loop in the PS model, thereby converting it to an acyclic *modified pseudo-sequential* (MPS) model. A pair of input/output buses is created for each loop that is broken, and these buses are referred to as the *pseudo-state input/output* (PSI/PSO) buses.

The advantages of our high-level modeling technique are illustrated by comparing M^G and M^H for the 27-bit parity checker circuit introduced in figure 6(b). The MPS model of the circuit is shown in figure 8(b); it has two pairs of pseudo-state input/output buses, denoted PSI_1/PSO_1 and PSI_2/PSO_2. To compare the number of components and buses in the MPS model to those in the gate-level model M^G, we need to expose internal details of the high-level modules A^H and B^H. Figure 9 shows these modules' internal structure. In this case, M^G contains 228 components which are connected using 359 single-bit lines. M^H for the same circuit has only 51 components interconnected via 91 buses. Thus, M^H has only one-fourth the number of components and buses of M^G. Moreover, the number of components and buses in the MPS model remain constant at 51 and 91, respectively, independent of the size of the array, while the number of components and lines in a gate-level model of the array are $47q$ and $81q$, respectively, if the array consists of q periods. The bus sizes,

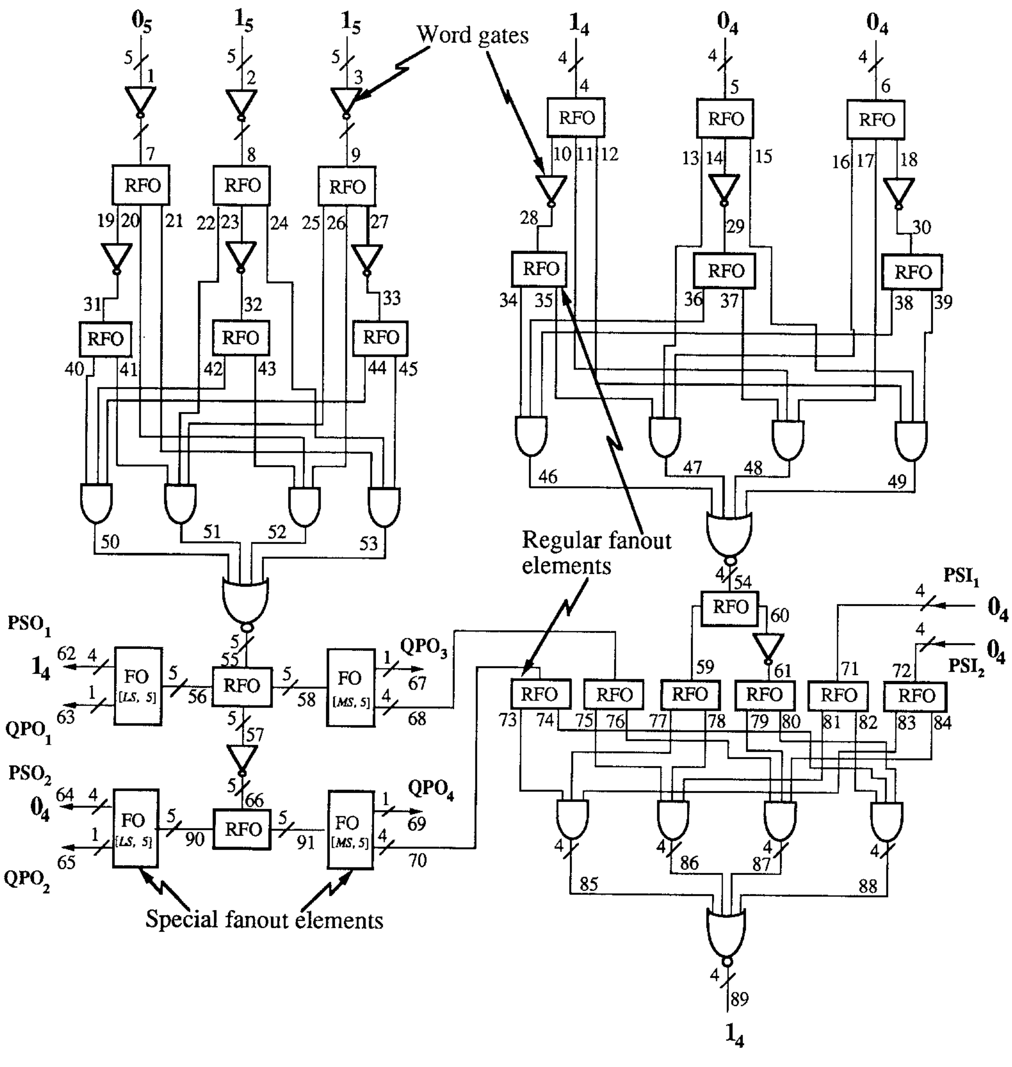


Fig. 9. Detailed MPS model of parity checker circuit of Fig. 6(b).

on the other hand, increase from 1 in the gate-level model to q in the MPS model. The usefulness of the MPS model will be demonstrated further in the next section.

The MPS models for k-regular circuits are truly hierarchical in that we can construct a sequence of high-level models $M^1 = M^H$, M^2, ..., $M^q = M^G$ for a given k-regular circuit, such that M^G and M^H represent the two extreme cases in this sequence of models. It follows from the definition of k-regularity that if a circuit with n modules, $n = qk_0 - r$, is k_0-regular, then it is also k_0s-regular for any integer $1 \leq s \leq q$. The sequence of high-level models of the given circuit can now be obtained using pseudo-sequential contraction by considering the given circuit to be k_0s-regular, and varying the parameter s within the permissible limits. Sizes of all buses other than pseudo-state buses decrease from q in M^1 to 1 in M^q. The pseudo-state input/output bus sizes go from $q - 1$ in M^1 to 0 in M^q, i.e., they vanish in the limiting case.

4. Test Generation Algorithm

We now present a test generation procedure which utilizes the hierarchical circuit and fault models developed in the previous sections. This algorithm allows the high-level circuit model to consist of arbitrary high-level components such as MCLA used in M^H of the 74181, as well as simple general-purpose components like word gates. In the case of k-regular circuits, as stated in section 2, we use an MPS model for M^H. Note also that in most cases, the natural high-level model M^H to use is the register-level model developed in the early stages of a typical CAD-based design process.

The input/output behavior of high-level components is described using the VS notation of section 2, combined with a straightforward generalization of the cubical representation found in [14]. The elements of the cubes thus obtained are vectors instead of scalars, and hence the cubes are referred to as *vector cubes*. Some examples of vector cubes for an n-input NOR gate are shown below:

$$\begin{aligned}&1_n \odot X_n \odot X_n \,/\, 0_n\\&X_n \odot 1_n \odot X_n \,/\, 0_n\\&0_n \odot 0_n \odot 0_n \,/\, 1_n\\&S_{(1\ldots n)}(1_{\lceil n/2 \rceil} \otimes X_{\lceil n/2 \rceil})\\&\odot S_{(1\ldots n)}(X_{\lceil n/2 \rceil} \otimes 1_{\lceil n/2 \rceil})\\&\odot X_n \,/\, 0_n\end{aligned} \tag{1}$$

where X_n represents the don't care vector of size n.

The concept of cube intersection has also been directly extended from the scalar to the vector case. Two vector cubes V_1 and V_2 have a nonempty *vector intersection* V if every element in cube V_1 can be intersected with its counterpart in V_2 using the rules

$$s \cap s = s, \qquad s \cap X = s$$

where $s \in \{0, 1\}$. For example, referring to the vector cubes of (1), we see that

$$1_n \odot X_n \odot X_n \,/\, 1_n \cap X_n \odot 1_n \odot X_n \,/\, 1_n = 1_n \odot 1_n \odot X_n \,/\, 1_n$$
$$1_n \odot X_n \odot X_n \,/\, 0_n \cap 0_n \odot 0_n \odot 0_n \,/\, 1_n = \phi$$

where $\cap$ now denotes vector intersection, and ϕ is the empty set.

The number of possible vector cubes of a component grows rapidly with increasing bus size. However, by adopting a hierarchical approach to test generation, and by restricting ourselves to total bus faults only at each level, we do not have to consider all possible vector cubes; it suffices to consider only those cubes relevant to test generation for total bus faults.

Before presenting the test generation algorithm, we briefly consider the SSL fault coverage obtained by generating tests for total bus faults in high-level models of circuits with varying degrees of regularity. This allows us to demonstrate the potential of the proposed approach, as well as to illustrate the major problems of achieving good fault coverage. Consider a class of extremely regular circuits whose M^H is composed of word gates and regular fanout elements only, interconnected by buses of fixed size. For such circuits, the following easily proven result holds:

Theorem 1: *A complete test set for all total bus faults in a high-level model* M^H *composed of word gates and regular fanout elements interconnected by buses of fixed size, is also a complete test set for all SSL faults in the corresponding gate-level model* M^G.

Although practical circuits that conform exactly to the specifications of theorem 1 are rare, many common circuits containing large repeated subcircuits also have similar characteristics. For instance, it is shown in [3] that a complete test set for total bus faults in M^H of the 4-bit 2-to-1 multiplexer of figure 4 is also a complete test set for all SSL faults in the corresponding gate-level model.

Some important issues in test generation for high-level circuits are illustrated by the MPS model M^H of the parity checker circuit shown in figure 9. As stated earlier, the buses marked PSI (PSO) in figure 9 represent the pseudo-state input (output) buses in the MPS model. Two problems in generating tests for an MPS model are that the PSI buses are not directly controllable and the PSO buses are not directly observable. The lack of observability implies that simply generating a test for a total bus fault in an MPS model does not guarantee detection of all SSL faults associated with the faulty bus, because the test generation process can lead to the error being propagated to an unobservable PSO bus. Drawing on the analogy between the sequential and pseudo-sequential circuit models, we see that more than one iteration (pass) through the MPS model may now be necessary to generate a test for a k-regular circuit. These iterations, however, differ significantly from those required for sequential circuits [4] in that all primary input bus assignments found in the first iteration are retained in all subsequent iteration steps. These *pseudo-iterations*, therefore, can only lead to successive refinements of the input assignment already found in the first iteration, unlike the corresponding iterations in the case of sequential circuits, which lead to a sequence of distinct input patterns.

The second problem of test generation for k-regular circuits, is the presence of uncontrollable PSI buses. We must now ensure that any vector assigned to a PSI can indeed be applied to the corresponding lines in the actual circuit. An example to the contrary is shown in figure 9 where the current input pattern applies a vector 0_4 to PSI_1, and generates an output of 1_4 at the corresponding PSO_1. However, since a corresponding pair of lines in PSI and PSO buses are derived from a single line in the gate-level model, as explained in section 3, they must be assigned the same signal value in the fault-free case; we designate this the *compatibility requirement*. Two vectors V_1 and V_2, defined on the signal set $\{0, 1, D, \bar{D}, X\}$, are said to be *compatible* with each other, if replacing the D and $\bar{D}$ elements in the two vectors by 1 and 0, respectively, results in vectors V_1' and V_2' such that $V_1' \cap V_2' \neq \phi$. Obviously, 0_4 is incompatible with 1_4, and hence the input assignment shown in figure 9 must be rejected as invalid.

We now turn to the generation of tests for arbitrary combinational circuits for which hierarchical descriptions on two or more levels exist. We have developed a hierarchical test generation algorithm VPODEM which works with both high-level and gate-level circuit and fault models. It is a substantially extended version of PODEM and recognizes two classes of circuits: combinational and modified pseudo-sequential; it also treats signals and faults as vectors. While we do not consider sequential circuits explicitly, they could be handled by VPODEM in the conventional manner by constructing iterative combinational models of the original circuits [4].

The choice of a conventional algorithm as the basis for our hierarchical test generator was motivated by the requirement that the algorithm should reduce to a standard test generation algorithm when a gate-level circuit and the SSL fault model are used; this allows us to guarantee 100 percent SSL fault coverage for general circuits. As discussed earlier, bus sizes may become nonuniform at the higher level of representation, and a combinational circuit may be transformed into a pseudo-sequential one. As a result, our test generation algorithm needs the capability of handling general high-level components, checking for compatibility between pseudo-state input/output bus assignments, and performing pseudo-iterations when needed. The basic test generation algorithm implemented by VPODEM, is summarized in figures 10 and 11. It consists of two main procedures: TESTGEN (figure 10) which is a redesigned version of the test pattern generator of conventional (or scalar) PODEM, and ITERATE (figure 11) which is an extension to conventional PODEM to handle MPS models. VPODEM's name is derived from the fact that it can assign vectors to buses in a high-level circuit model, in contrast with the conventional PODEM algorithm which can only assign scalar values to lines in a gate-level model.

The various steps performed by VPODEM are now illustrated by applying it to the MPS model of the parity checker circuit presented in figure 9, assuming the 5-bit bus labeled 55 to be totally stuck at 1. For this fault, VPODEM performs two pseudo-iteration steps; the resulting input assignments are shown in figures 12 and 13. At the beginning, all bus signals are assumed to be uninitialized, and so have value X_n assigned to them, where n is the bus size. In the first pseudo-iteration step, VPODEM goes through several cycles of initial objective selection and back-tracing [9] to find an input assignment of vectors 0_5, 1_5, and 1_5 to buses 1, 2, and 3 respectively, which propagates error vectors D_4 and $\bar{D}_4$ to PSO buses 62 and 64; see figure 12. Compatibility between various PSI and PSO buses is maintained because PSI buses 71 and 72 are still assigned X_4, which is, by definition, compatible with both D_4 and $\bar{D}_4$.

In the second pseudo-iteration step, the input assignment from the first step is saved, the vectors D_4 and $\bar{D}_4$ from buses 62 and 64 are assigned to buses 71 and 72 respectively, and the total bus fault is neglected. Our goal here is to generate a test for a bus fault that will detect any SSL fault on corresponding lines of the bus.

```
procedure TESTGEN (ENTRY,PSEUDO__ITERATION__FLAG)
 do until ((error is propagated to an output bus)
         OR (no alternative input assignment possible))
   call INITOBJ(ENTRY,PSEUDO__ITERATION__FLAG)
/*initobj sets the initial objective if parameter ENTRY is set to 1. If
PSEUDO__ITERATION__FLAG is set to FALSE, then it checks for
uninitialized condition on the faulty bus. Otherwise, it assumes error
signal to be present on some input, and tries to set an initial objective to
propagate this error signal to some output bus*/
   if ((failure in initobj) OR (ENTRY = 2)) then
/*ENTRY = 2 implies that execution should start with a backtracking
step (alternative assignment to primary input in decision-tree handler)*/
     call DECISION-TREE HANDLER (failure)
     if (no more alterantives exist) then
       EXIT in failure
     end /*if*/
     if (ENTRY = 2) then
       ENTRY := 1
     end /*if*/
   end /*if*/
   if (initial objective is selected successfully by initobj) then
     call BACKTRACE
     call ASSIGN PRIMARY INPUT
     call DECISION-TREE HANDLER (success)
   end /*if*/
   call IMPLICATION
   if (ENTRY = 3) then
     ENTRY := 1
   end /*if*/
/*If ENTRY = 3 to begin with, then initobj does nothing which means
all steps before implication are skipped. Setting ENTRY to 1 signals
initobj to set objective levels in successive cycles*/
   if pseudo-state input and output assignments are not compatible then
     ENTRY := 2
   end /*if*/
 end /*do until*/
 if (error is propagated to an output bus) then
   EXIT in success
 else
   EXIT in failure
 end /*if*/
end /*TESTGEN*/
```

Fig. 10. Pseudo-code of procedure TESTGEN in VPODEM.

However, an SSL fault can actually occur in only one repetitive array position or *period* of the gate-level model, say period s. Error signals in subsequent periods $s + r$, $r \geq 1$, of the array will only be the result of error signals generated on the horizontal output lines from period s. Hence, it is necessary to neglect the total bus fault in the second, and subsequent pseudo-iteration steps. VPODEM now assigns vectors 1_4 to the input bus 4, and 0_4 to buses 5 and 6 respectively, which causes error vector $\bar{D}_4$ to be propagated to primary output bus 89; see figure 13. It is easily seen that compatibility is still maintained between corresponding PSI and PSO bus assignments. Hence, the test input pattern consists of values assigned to the primary input buses 1, 2, 3, 4, 5, and 6. If the error vector can still not be propagated to any primary output bus, but can be

```
procedure ITERATE
  PSEUDO__ITERATION__COUNT=0
  PSEUDO__ITERATION__FLAG=FALSE
  ENTRY=1
  do while (PSEUDO__ITERATION__COUNT <MAX__COUNT)
    call TESTGEN (ENTRY,PSEUDO__ITERATION__FLAG))
    if (test is not found) then
      if (PSEUDO__ITERATION__COUNT = 0) then
        EXIT in failure
      else
        PSEUDO__ITERATION__COUNT := PSEUDO__ITERATION__COUNT-1
        pop input bus assignment off pseudo-iteration stack
        ENTRY := 2
        if (PSEUDO__ITERATION__COUNT = 0) then
          PSEUDO__ITERATION__FLAG=FALSE
        end /*if*/
      end /*if*/
    else
      if (PSI/PSO buses are absent) then
        EXIT in success
      else
        justify pseudo-state input/output bus assignments
        if (justification is not possible) then
          pop input bus assignment off pseudo-iteration stack
          ENTRY := 2
        else
          push input bus assignment into pseudo-iteration stack
          if (error vector appears on primary output bus) then
            EXIT in success
          else
            PSEUDO__ITERATION__COUNT := PSEUDO__ITERATION__COUNT+1
            assign previous pseudo-state output values to pseudo-state inputs
            assign primary input values from previous iteration
            ENTRY := 3
            PSEUDO__ITERATION__FLAG := TRUE
          end /*if*/
        end /*if*/
      end /*if*/
    end /*if*/
  end /*do while*/
  if (PSEUDO__ITERATION__COUNT = MAX__COUNT) then
    EXIT in success
  end /*if*/
end /*ITERATE*/
```

Fig. 11. Pseudo-code of procedure ITERATE in VPODEM.

propagated to some PSO bus while meeting the compatibility requirement, then further pseudo-iteration steps are required.

As noted earlier, a major difference between pseudo-iterations in VPODEM, and iterations occurring during test generation for sequential circuits [4], is that the former merely generates successive refinements of the input assignment found in the first step. Another major difference is that the number of pseudo-iteration steps possible for the MPS model of a given k-regular circuit is bounded by the number of periods in M^G. This number is reasonably small for most practical circuits;

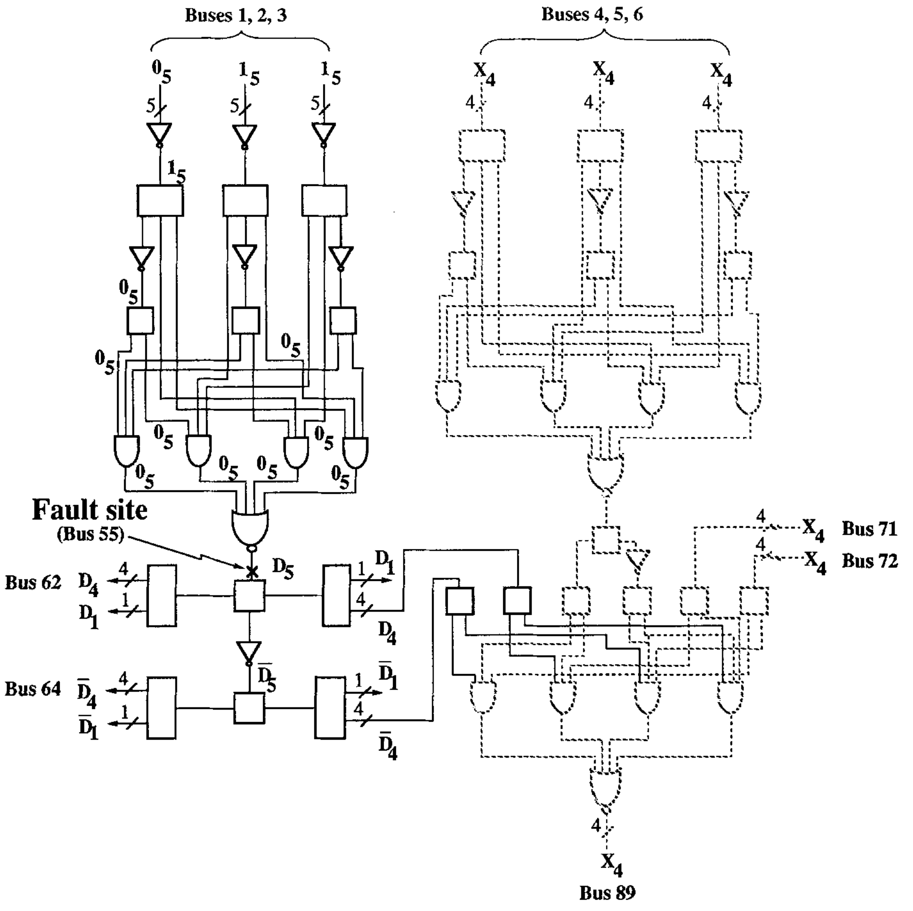


Fig. 12. ITERATE and TESTGEN illustrated: after assignment of primary inputs 1, 2, and 3 in first pseudo-iteration step.

similar bounds for sequential circuits are exponential in the number of states of the circuit, and are of little practical use. Also note that multiple pseudo-iteration steps are automatically avoided by VPODEM if the circuit model has no PSI/PSO buses. In such cases, VPODEM goes through only a single pseudo-iteration step similar to a PODEM iteration, the main difference being that VPODEM assigns vectors to buses of size greater than one, while PODEM assigns scalar values to single lines. A more detailed discussion of the VPODEM algorithm can be found in [3].

Our algorithm, like PODEM, is complete in the sense that it can generate tests for all detectable SSL faults in any well-formed combinational circuit. Test generation using both high-level and gate-level models may be necessary to obtain such complete SSL fault coverage. However, for circuits containing regular sub-circuits like k-regular circuits, a large percentage of SSL faults are expected to be detected by generating tests for the high-level model as suggested by the following result [3]:

Theorem 2: *A test for a total bus fault* $F_i = i_n$, $i \in \{0, 1\}$ *on a bus* B *in the MPS model of a k-regular circuit generated by VPODEM detects all SSL faults* $f_i = i_1$ *on individual lines of* B *in a gate-level model of the circuit.*

The advantages of the proposed algorithm over conventional test generation algorithms are twofold. First, VPODEM is invariant with respect to the representation level of the circuit and fault models used. We have achieved this invariance by adopting a hierarchical circuit modeling technique, and identifying an important class of faults suitable for the high-level circuit models, viz, total bus faults. Conventional test generation algorithms are, in most cases, tied to the gate-level circuit and fault models. The second, and perhaps more

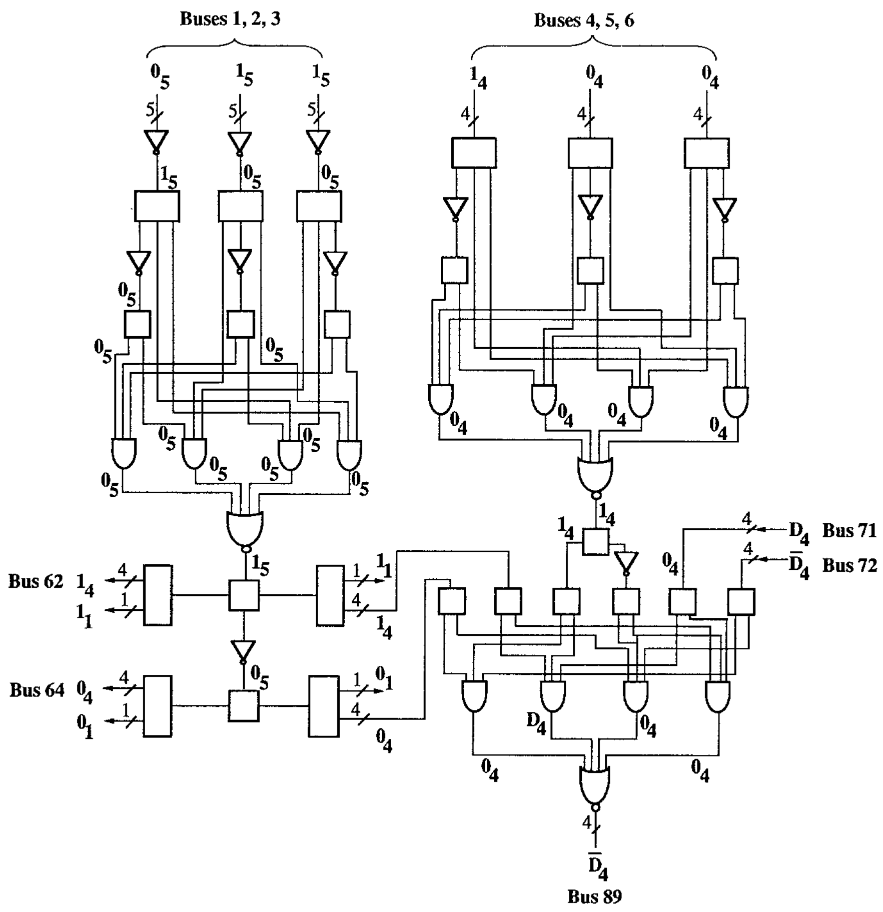


Fig. 13. ITERATE and TESTGEN illustrated: final assignments to primary inputs at termination of ITERATE.

important, advantage stems from the hierarchical nature of the proposed test generation technique. Tests can first be generated for total bus faults in M^H of the circuit. Due to the grouping of many lines into a single bus at this level, the total number of target faults is significantly less than in a gate-level model of the circuit, leading to considerable reduction in the overall test set size and test generation effort.

5. Experimental Results

This section outlines a computer program implementing VPODEM, and presents experimental results showing the advantages of the proposed algorithm over conventional techniques. The program is written in FORTRAN, the choice of language being made from considerations of portability, optimization, and possible vectorization.

The input to VPODEM is a text file which provides the circuit description and various user commands to control program execution and output. The input data is assumed to be organized into three blocks: component interconnection information in the form of a net list; the sizes and types of buses in the circuit; and user options controlling the built-in fault simulator. The circuit model, either low-level or high-level, is conceptually stored in the form of a set of circularly linked lists. However, due to the lack of dynamic data structures in FORTRAN, these lists, and other pertinent information like the status of the decision tree [9], vector cubes for general high-level components other than word gates, and various stacks are all simulated using arrays grouped into COMMON blocks.

VPODEM is implemented by a main program, two test generation procedures, a simple fault simulator, and various other subroutines that are used by these four procedures. The main program is responsible for creation of the various data structures, and controlling interaction between the test generation procedure and the fault simulator. The test generation procedures are straightforward implementations of ITERATE and TESTGEN described earlier. The built-in fault simulator can be used after each successful test generation cycle to eliminate all detected faults from the fault list, and thus reduce computation. Further implementation details of our program are available in [3].

The program has been used to generate tests for a set of eight representative circuits, mostly based on MSI circuits of the 7400 series of ICs, for which high-level and the gate-level models are readily available [17]. Benchmark circuits like the ISCAS set [6] cannot be used here since the required structural information about these circuits are unavailable. Circuit CUT1 is the 74630 parity generator with its primary fanout removed; CUT2 is the 74157 4-bit 2-to-1 multiplexer of figure 4; CUT3 is an 8-bit ripple-carry adder using NAND gates only; CUT4 is an 8-bit shifter based on the 74350; CUT5 is the 74181 ALU whose gate-level and high-level models have already been presented in figure 5; CUT6 is a modified 74381 obtained from [20]; CUT7 is the parity checker circuit of figure 9; and CUT8 is a 1/256 decoder tree circuit constructed using modified 1/16 decoders based on the 74154 IC [3]. The results of applying our test generation program to these eight circuits are tabulated in tables 1 through 3. Table 1 compares the complexity of the gate-level and high-level models using the component count and the number of buses as measures of complexity.

Table 1. Number of components and buses in gate-level and high-level models of circuits CUT1–8.

	Gate-level model M^G		High-level model M^H	
	Number of components	Number of lines	Number of components	Number of buses
CUT1	78	144	13	24
CUT2	20	43	12	18
CUT3	120	201	17	30
CUT4	128	227	26	47
CUT5	101	201	12	26
CUT6	240	441	81	164
CUT7	208	359	51	91
CUT8	808	2220	38	47

Table 2 compares the number of tests generated for total bus faults in the high-level model M^H to the number of tests generated for SSL faults in the gate-level model M^G, as well as the time spent (in seconds on a SUN 4/110 workstation) in test generation at the two levels. It also provides the SSL fault coverage of the tests generated for total bus faults in M^H, and the number of extra tests, if any, needed to obtain 100 percent SSL fault coverage. Note that test generation time represents the total time spent in the test generation routines only, and does not include the time spent in the fault simulator. (In our implementation, we use a simple but inefficient simulator leading to simulation times that are not easily compared.)

From table 1, it is clear that the high-level modeling leads to a substantial reduction in the circuit complexity. For example, CUT5 has 101 components and 201 lines at the gate level but only 12 components and 26 lines at the high level. Similarly, as illustrated in table 1, M^H of CUT7 has approximately one-fourth the number of components of its gate-level model. This complexity reduction is perhaps best illustrated by the modified decoder circuit CUT8. In this case, M^H contains less than one-twentieth the number of components and less than one-fortieth the number of buses in M^G. Although the size of the buses in M^H is usually greater than one, we show next that bus size plays a smaller role in the overall test generation effort than does the number of components and buses used.

Table 2 shows that the SSL fault coverage obtained using tests generated for total bus faults in M^H is quite high (70 to 100 percent) for all the sample circuits. However, the number of tests required to obtain this fault coverage is much smaller than the number of tests obtained using M^G alone. For example, in the case of CUT7, 60 tests were generated using M^G, while only

Table 2. Test set sizes, test generation times, and SSL fault coverage obtained using high- and gate-level models of CUT1–8.

	Gate-level model M^G		High-level model M^H			Number of extra tests for 100% SSL fault coverage	Total test generation time using two-level approach (sec.)
	Number of tests	Test generation time (sec.)	Number of tests	Test generation time (sec.)	SSL fault coverage		
CUT1	72	9.51	12	2.04	100%	0	2.04
CUT2	16	1.10	8	0.28	100%	0	0.28
CUT3	26	7.97	5	0.79	100%	0	0.79
CUT4	50	37.14	12	10.20	70%	14	20.47
CUT5	54	29.52	16	2.77	78%	16	12.11
CUT6	63	51.32	15	16.18	77%	21	33.79
CUT7	60	45.44	15	7.16	94%	16	16.24
CUT8	483	1738.88	12	2.71	98%	16	66.39

15 tests with 94 percent fault coverage were generated using M^H. A set of 16 additional tests were found to cover all SSL faults not covered by the high-level tests. Thus, for the circuits considered, the hierarchical approach to test generation leads to only half as many tests as generated by the gate-level technique alone. The program performs very well for highly regular circuits with small fanout like CUT1, CUT2, and CUT3, or for circuits like CUT8 which have been designed to facilitate high-level test generation. In these cases, test generation at the high level provides complete, or very nearly complete, SSL fault coverage with significantly smaller test sets than were generated using gate-level models only. On the other hand, the presence of complex fanout, as in CUT4 through CUT7, tends to reduce the SSL fault coverage. However, the combined use of both the high-level and the gate-level models is seen to reduce the total number of tests by 50 percent or more, in most cases. The corresponding speedup of test generation, though not exactly proportional to the reduction in test size or circuit complexity, is significant in all cases. For example, the speedup for CUT7 is approximately 3, that for CUT4 is approximately 2, while that for CUT8 is approximately 26.

Finally, we compared our high-level approach to random test generation, which is commonly used as an alternative to algorithmic test generation since randomly chosen test patterns are very easy to generate. For each circuit, the number of random patterns generated was made equal to the number of patterns generated using our high-level algorithm, and a standard random number generation routine was used to generate the random input patterns. Three random sequences with different initial seeds were used, and the average SSL fault coverage of the three resultant random pattern sets was evaluated using the fault simulator in our implementation of VPODEM.

The results of this comparison are shown in table 3, which indicates that the SSL fault coverage of the random tests is, in almost all cases, significantly smaller than that obtained by applying VPODEM to the high-level model alone. In fact, only in the case of CUT4, does the scheme of the random test pattern generation provide fault coverage comparable to that obtained by the high-level scheme, for the same small number of tests. For circuits like multiplexers (CUT2) or decoders (CUT8), the performance of the random test generation scheme was found to be far inferior to the proposed high-level scheme. Furthermore, random patterns must usually be supplemented with deterministic tests to achieve 100 percent SSL fault coverage with realistic sizes of test sets [1]. In contrast, our hierarchical approach is guaranteed to provide complete SSL fault coverage for arbitrary circuits. The data in tables 1 and 2 also show that the test sets from VPODEM are, in general, significantly smaller than the test sets obtained using conventional gate-level test generation algorithms.

Table 3. Comparison of the SSL fault coverage of random test generation and VPODEM.

		SSL fault coverage	
	Number of tests	Tests generated randomly	Tests generated by VPODEM
CUT1	12	88%	100%
CUT2	8	61%	100%
CUT3	5	71%	100%
CUT4	12	69%	70%
CUT5	16	66%	78%
CUT6	15	63%	77%
CUT7	15	81%	94%
CUT8	12	16%	98%

6. Conclusion

We have presented a novel methodology to model large digital circuits for test generation purposes. Circuit behavior is described by vector sequences of spatial (bus) size n, where n is a measure of complexity level. While $n = 1$ corresponds to the classical gate level, larger values of n correspond to the register level. We have introduced a new class of (total) bus faults, also parametrized by n, which generalize the classical SSL fault model to higher levels. The resulting circuit and fault models are truly hierarchical, and avoid the confusion arising out of mixing concepts from different levels. We have shown that in the special case of k-regular array circuits, useful trade-offs can be made between the number of periods in the array q, and the bus size n, using a new class of models called modified pseudo-sequential or MPS circuits. A high-level MPS circuit with a fixed number of components can model a k-regular array of arbitrary length. The use of such high level models with bus size n leads to approximately an n-fold reduction in the test set size due to the fact that tests for the faults associated with the individual lines of a multiline bus are effectively generated in parallel.

To evaluate the feasibility of hierarchical test generation using these concepts, we have defined and implemented the VPODEM algorithm. Our experiments with VPODEM show that for circuits of moderate to high regularity, complete test sets for total bus faults in M^H

provide quite good SSL fault coverage in M^G. This suggests that, even for moderately irregular circuits, there may be no need to consider the many nontotal faults possible in an n-bit bus; the two total bus faults may suffice. Moreover, the high-level test sets produced by VPODEM are smaller, and can be more rapidly generated, than the corresponding test sets derived from M^G alone. This suggests that, in many practical instances, all test generation can be done at the high level, thus dispensing completely with the less efficient classical circuit and fault models. Furthermore, we can use VPODEM to generate tests directly for SSL faults (provided, of course, a suitable gate-level model is available), allowing us to adopt a hierarchical approach for general circuits. Our experiments show that this hierarchical approach results in significantly smaller test sets providing 100 percent SSL coverage, compared to gate-level approaches that use standard deterministic methods either alone, or in combination with random test generation.

Acknowledgments

The authors wish to thank the anonymous referees, and Dr. V.D. Agrawal of AT&T Bell Laboratories, for valuable suggestions and comments.

This research was supported by the National Science Foundation under Grant No. MIP-8805517.

References

1. R.G. Bennetts, *Design of Testable Logic Circuits*, Addison-Wesley, Reading, MA, 1984.
2. D. Bhattacharya and J.P. Hayes, "High-level test generation using bus faults," *Proc. 15th Fault-Tolerant Comput. Symp.*, pp. 65–70, June 1985.
3. D. Bhattacharya and J.P. Hayes, *Hierarchical Modeling for VLSI Circuit Testing*, Kluwer Academic Publishers, Boston, MA, 1990.
4. M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Rockville, MD, 1976.
5. M.A. Breuer and A.D. Friedman, "Functional level primitives in test generation," *IEEE Trans. Computers* C-29 (3): 223–235, 1980.
6. F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," *Proc. IEEE Intern. Symp. on Circuits and Systems*, pp. 663–698, Kyoto, June 1985.
7. W.-T. Cheng and J.H. Patel, "Testing in two-dimensional iterative logic arrays," *Proc. 16th Fault-Tolerant Comput. Symp.*, Vienna, pp. 76–83, July 1986.
8. A.D. Friedman, "Easily testable iterative systems," *IEEE Trans. Computers* C-22 (12): 1061–1064, 1973.
9. P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Computers* C-30 (3): 215–222, 1981.
10. J.P. Hayes, "A calculus for testing complex digital systems," *Proc. 10th Fault-Tolerant Comput. Symp.*, pp. 115–120, October 1980.
11. S.C. Lee, "Vector boolean algebra and calculus," *IEEE Trans. Computers* C-25 (9): 865–874, 1976.
12. Y.H. Levendel and P.R. Menon, "Test generation algorithms for computer hardware description languages," *IEEE Trans. Computers* C-31 (7): 577–587, 1982.
13. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
14. J.P. Roth, "Diagnosis of automata failures: a calculus and a method," *IBM J. Res. Develop.* 10 (10): 278–281, 1966.
15. F. Somenzi, S. Gai, M. Mezzalama, and P. Prinetto, "Testing strategy and technique for macro-based circuits," *IEEE Trans. Computers* C-34 (1): 85–89, 1985.
16. T. Sridhar and J.P. Hayes, "Design of easily testable bit-sliced systems," *IEEE Trans. Computers* C-30 (11): 842–854, 1981.
17. Texas Instruments Inc., *The TTL Data Book*, vol. 2, Dallas, TX, 1985.
18. S.M. Thatte and J.A. Abraham, "A methodology for functional level testing of microprocessors," *Proc. 8th Fault-Tolerant Comput. Symp.*, pp. 90–95, June 1978.
19. Y. You, *Self-testing VLSI Circuits*, Ph.D. Dissertation, Dept. of Electrical Engineering and Computer Science, The University of Michigan, 1986.
20. Y. You and J.P. Hayes, "Implementation of VLSI self-testing by regularization," *IEEE Trans. Computer-Aided Design* 7, pp. 1261–1271, December 1988.

Debashis Bhattacharya received the B. Tech. (Honors) degree in 1983 from the Indian Institute of Technology, Kharagpur, India. He received the M.S. and Ph.D. degrees in Computer, Information and Control Engineering (CICE) program from the University of Michigan at Ann Arbor in 1985 and 1988, respectively.

From 1983 to 1988 he was a research assistant at the University of Michigan, working on high-level test generation techniques for digital circuits. He was awarded the IBM fellowship in 1986 and 1987, and the Outstanding Student Award in CICE in 1988. In 1988 he joined the Electrical Engineering Department at Yale University as an assistant professor. His current research interests include digital testing and simulation, VLSI design, and parallel processing using systolic arrays. He has coauthored a book titled *Hierarchical Modeling for VLSI Circuit Testing* (Kluwer Academic Publishers, to be published).

Dr. Bhattacharya is a member of the IEEE Computer Society, and Tau Beta Pi.

John P. Hayes received the B.E. degree from the National University of Ireland, Dublin, in 1965, and the M.S. and Ph.D. degrees from the University of Illinois, Urbana, in 1967 and 1970, respectively, all in electrical engineering.

While at the University of Illinois he participated in the design of the ILLIAC III computer, and carried out research in the area of fault diagnosis of digital systems. In 1970 he joined the Operations Research Group at the Shell Benelux Computing Center of the Royal Dutch Shell Company in The Hague, The Netherlands, where he was involved in mathematical programming and software development. From 1972 to 1982 Dr. Hayes was a faculty member of the

Departments of Electrical Engineering and Computer Science of the University of Southern California, Los Angeles. He is currently a Professor in the Electrical Engineering and Computer Science Department of the University of Michigan, Ann Arbor. He was Technical Program Chairman of the 1977 International Conference on Fault-Tolerant Computing. Dr. Hayes is the author of over a hundred technical papers and several books, including *Digital System Design and Microprocessors* (McGraw-Hill, 1984) and *Computer Architecture and Organization, 2nd ed.* (McGraw-Hill, 1988). He served as editor of the Computer Architecture and Systems Department of *Communications of the ACM* from 1978 to 1981, and was Guest Editor of the June 1984 Special Issue of *IEEE Transactions on Computers*. He was the founding Director of the Advanced Computer Architecture Laboratory at the University of Michigan from 1985 to 1988.

Dr. Hayes's current research interests include computer architecture; parallel processing; fault tolerance and reliability; and computer-aided design and testing of VLSI systems. He is a fellow of IEEE, and a member of the Association for Computing Machinery and Sigma Xi.