"USE OF THE SUPSCRIPTED DALG IN SUBMODULE TESTING WITH APPLICATIONS IN CELLULAR ARRAYS"

M. Ladjadj, J. F. McDonald, D-H Ho and W. Murray Jr., Lt. USAF

Rensselaer Polytechnic Institute
Center for Integrated Electronics
Troy, New York 12181

Abstract

This paper will introduce the concept of the Transparency cube in the extraction and testing of an embedded logic submodule. The algorithm presented can establish control of a submodule in a very small amount of CPU time, and it was found to perform extremely well when tested on cellular array topologies such as those which occur in systolic architectures. The program described detects if parts of a circuit are untestable and notifies the user as to where additional logic is necessary to make the circuit more transparent or testable. It also introduces controllability numbers for flexible signals (or subscripted D's) and a method by which an Exclusive Or gate (EXOR) is handled as a single gate in the controllability calculations. This is useful since controllability numbers do not represent well the difficulties encountered with reconvergent fanout.

Introduction

The design community has had a need for sometime, for a method by which an individual subsection or package embedded in a circuit could be tested. With the program discussed here, the user has the ability to test an individual package rather than having to consider the entire circuit at once for each test pattern. Another advantage is that there is no enforcement of a particular fault model upon user (i.e. this paper refers to stuck-at faults as an example only, any failure type may be tested). It would be very wasteful to introduce control and observation paths into a circuit for the sole purpose of testing, and this program enables the user to test a submodule with little or no additional logic taking advantage of the naturally occurring transparent properties of the embedding logic.

The transparency cube program[1] for submodule testing establishes control of a user specified submodule by constructing control and observation paths to the submodule, which in effect make the surrounding logic transparent. The program outputs the "Controlling Test Set" which specifies the enabling signals to set up the control paths (or channels) from the circuit's primary inputs to the submodule's inputs and observation paths from the submodule's outputs to the circuit's outputs. The test pattern for the submodule is substituted into the controlling test set (into the respective control path positions) and transmitted through the control paths to test the submodule.

The idea of submodule testing was developed at RPI in 1979 by J. F. McDonald and A. N. Airapetian[2]. It was noted that a section of a circuit could be tested by grouping the gates into a submodule, or package, and constructing multiple, independent, simultaneously sensitized control and observation paths to the submodule. This submodule could be of any size (down to a single gate). It has now been demonstrated that this approach would be particularly useful in the testing of systolic or cellular array topologies (discussed later), where these arrays consist of only a few types of well characterized cells.

Control paths and Observation paths are discussed throughout the paper and should be defined here. Control paths refer to flexible, sensitized signal lines which connect directly from the circuit's primary inputs to a device's inputs and through which the device can be directly controlled. Observation paths refer to the flexible signal sensitized lines connecting the device's outputs to the circuit's primary outputs through which the device's output signal can be directly monitored. These paths enable the user to propagate unchanged signals to a device input and to watch the device outputs for the resulting output signals, forcing the surrounding logic to be essentially transparent. The independent nature for the paths demands a different subscripted D symbol to represent the signal on each of these paths. These paths cannot always be found, but they exist in a

0738-100X/86/0000/034t$01.00

sufficient number of circuits to make their exploitation worthwhile.

The techniques used for the D-propagation and Line Justification by this program stems from the work Cherif Benmehrez did in his thesis on the Subscripted D-Algorithm (AALG)[3] and its idea of multiple simultaneously sensitized paths. Further modifications to AALG have been carried out by RPI's Center for Integrated Electronics Automatic Test Pattern Generation Group. Our definition of "D" differs from Paul Roth's in that it refers to a variable signal, that is a signal that is not specified as either a zero or a one at the time it is propagated. This enables us to construct control paths (known as sensitized paths)[3] which can carry either a 0 or a 1 thus enabling us to test for a stuck-at-0 and stuck-at-1 without having to reconstruct the control paths and observation paths for the latter. The new program employs a new routing scheme for our propagation routines which will be covered.

This paper will detail how an embedded submodule is extracted from a circuit by the Transparency Cube Program and tested and compare our algorithm's performance on cellular arrays with that of other algorithms.

Routing of the Signal Paths

The transparency cube program makes extensive use of testability numbers[7,8] in the routing of the submodule's control and observation paths. Testability numbers heuristically seek to measure how easily a gate can be controlled to have a specific value and how easily the gate's output may be observed at the circuit's outputs. In this paper, we will consider only circuits made up of combinational standard gates (AND, NAND, OR, NOR and NOT). A "combinational node" is defined as either a primary input of the circuit or an output of a standard cell. The controllability of a given node N is represented in the three quantities:

CO(N) - control node N to have a zero
C1(N) - control node N to have a one, and
CD(N) - control node N to have a flexible signal Di called a subscripted D.

Lower controllability numberic values indicate higher controllability. These quantities are related to the minimum number of combinational node assignments required to justify a 0, 1, D, or subscribed Di respectively on node N. The controllability "CD" measure concept is explained in the next section.

The Observability of node N [O(N)] is related to both the number of combinational standard cells between the node N and a primary output of the circuit and the minimum number of node assignments required to propagate the logical value on node N to a primary output.

The controllability and observability for fixed signals as computed by Goldstein[6] are used in this paper. The controllability and observability for flexible signals are calculated in the next section. It is important to note that these numbers (controllability and observability) do not have much correlation with fault detection capability and fault coverage as shown by Savir et al[10]. Also, these numbers do not represent well the conflicts arising due to reconvergent fanout. These numbers give a good approximation of the difficulty of creating a sensitized path and are used strictly to ease the routing of the control paths from the submodule's inputs backwards to the circuit's inputs and observation paths from the submodule's outputs to the circuit's outputs.

Little or no effort will be spent on getting secondary faults on those sensitized paths as our primary goal in this phase is to make the logic surrounding the submodule "transparent" to the signals generated by the test for the submodule. Once the control and observation paths have been generated, the AALG algorithm[3-5] is used to completely test the submodule by itself. The existence of both control paths and observation paths for the submodule guarantees that the set of test patterns generated by the subscripted D-algorithm can be applied to the circuit's inputs and the result observed at the circuit's outputs.

After calculating the testability numbers, the nodes (i.e. gates and primary inputs) are sorted into arrays for referencing. During the propagation phases, the program considers a node's available outputs/inputs and searches the arrays (corresponding to control and observation path routing) and selects the most observable route for the preferred forward propagation path and the most controllable route for backward propagation accordingly (i.e. it looks at the arrays and selects which ever available node is ranked highest).

Controllability Calculations for a Flexible Signal

The controllability number "CD" measures how easily a node can be assigned a flexible signal or subscripted D. With the input node controllabilities defined to be one, the program calculates the controllability measures C1(N), CO(N) and CD(N) for each node progressing from the primary inputs to the

primary outputs of the circuit. In the calculation of the controllability of a standard cell (AND, NAND, OR, NOR and NOT) output node, all the possible input assignments that accomplish the desired output node justification are examined.

For example, the DJ controllability for an AND gate will be the minimum effort needed to have a DJ (flexible signal) on any input of the gate while all the other inputs are assigned enabling signals. Therefore, CD will be equal to the minimum of the input CDs (ICD) plus the sum of the one (1) controllability of the other inputs added to the cell (AND gate) depth. Fig. 1 lists the controllability calculations for the standard cells.

In Fig. 1 each gate (node) has M input gates and I indicates that the value (ICD, IjC1, IjCO) is that of an input gate while j and k represent the individual input gates. The cell depth is one.

Exclusive or Gate Controllability Modification

We modified the controllability calculation for Exclusive Or (EXOR) gates so that signals could be routed through them correctly. The EXOR gate consists of 4 NAND gates and normally would be treated as a 3 level deep gate during the testability calculations. The EXOR's output would thus appear difficult to control, and it would receive a high testability number (poor). Our program, however, considers it to be a one level deep single gate and its output is easily controlled. Without this modification the signal would be misrouted and unnecessary backtracking would result. As shown in Fig. 2, the DJ controllability of the EXOR gate can be measured by the DJ controllabilities of gate D and gate E. A DJ signal at either gate can be generated by a DJ signal routed through gate A, B, or C. This can occur by either of the four ways expressed by the expressions EXPA, EXPB, EXPC and EXPD where CO, C1 and CD represent the difficulty to have a O, 1 and a DJ signal respectively. Fig. 3 shows the results for both the incorrect and the modified testability calculations for the EXOR gate as well as the modified equations.

Testing Cellular Arrays

Very large cellular or systolic arrays as defined by Kung[9] can be difficult to test. This difficulty is due to the conflicts that arise when a signal is propagated through an array structure. We will employ an N by N Baugh Wooley Testable Multiplier (BW) (made up of identical adder cells) as our example (Fig. 5). During the propagation of the control and observation paths through such a multiplier, an enormous amount of backtracking will result if the algorithm attempts to propagate a signal down a horizontal line (excluding the original submodule horizontal output). The backtracking will be due to the conflicts that will arise if a horizontal line is chosen. Our program avoids these conflicts in the testability sorting section and thus backtracking is minimized.

The systolic array structure of the BW multiplier actually makes it easier to test. It should be recognized that the structure can be broken down into 4 identical submodules and these submodules, in turn, into 4 idential submodules repeatedly until the submodule consists of only one non-repetitious set of gates. Then it is obvious that there are only a few unique submodules (compared with the total number). Taking an 8 by 8 BW multiplier, we can break it down into 4 identical quadrants (submodule A(i), i = 1,4) each containing 16 cells (a 4 by 4 BW). The program runs on these four large submodules (A(i), i=1,4) and outputs the enabling signals. Next, the program takes one quadrant and divides it up into 4 quadrants (submodule B(i), i=1,4) each containing 4 cells and runs the algorithm on these 4 submodules (B(i), i=1,4). The program breaks up one of these 4 submodules (B(i) i=1,4) into its 4 component adders (submodule C(i), i=1,4) and calculates the enabling signals for those individual cells (C(i), i=1,4). Finally, the test for the individual cell (13 gates) is obtained either by hand or by running it through AALG (7 patterns).

The program has now generated all the "controlling" test sets for the unique submodules. Each of these test sets has positions corresponding to the inputs of the submodule that it controls. To test the entire multiplier, an iterative approach is taken to construct the test set. Beginning with the upper right hand corner of the BW multiplier, the test set for an individual cell (C(i), i=1,4) is substituted into the controlling test set for that cell (into the mentioned corresponding positions). These values are taken then substituted into the 4-cell controlling test set (test set for B(i), i=1,4). Then this test set is finally substituted into the controlling test set for the 16-cell upper right hand module (A(i), i=1,4). The program does the same for the other individual cells in that 4-cell submodule and repeats the process for the other 4-cell submodules. This process is in turn repeated for the other 16-cell submodules until the entire test set has been constructed.

The program requires one (and only one) pass for each unique submodule and since there were only 16 unique submodules in the 8 by 8 BW, our program required only 16 passes to generate the entire test. The Transparency Cube program generates the test set for an N by N BW multiplier (where N=2**k) in only 4 * Log2(N) passes through the program (28 * Log2(N)) test patterns. A 4 by 4 BW multiplier is not 100% testable by itself. Fig. 5(a) shows the 4 by 4 BW with the 6 untestable submodules noted. As we stated, the program tells the user where to place additional logic, and with the simple addition of 9 gates and one input (CSW), the circuit becomes 100% testable (Fig. 3(b)).

Performance Comparison

The transparency cube program generates the test set for the 4 by 4 testable BW multiplier (with its 8 unique submodules) in 120 CPU seconds on a VAX 11/780 covering 100% of the faults. Ohter ATPG algorithms were tested on the BW multiplier (AALG and PODEM). AALG was found to take 10 times, and PODEM 22 times, more CPU time than our algorithm with PODEM solving only 87% of the circuit's faults. These algorithms spent most of their time backtracking due to misrouting of the signals while the transparency cube algorithm did not due our implementation of testability numbers for flexible signals and our EXOR controllability calculation modification. For our program, the most difficult submodule required 57 seconds to create the transparency cube while the easiest took only 0.48 seconds.

Transparency Cubes

A transparency cube of a module is a generalization of the Propagation D Cube (PDC) in that it permits the simultaneous propagation of more than one subscritped D to pass over simultaneously sensitized paths through a logic block. It is, therefore, like a multiple propagation D-cube. The goal of the transparency cube algorithm is to create transparency cubes for the submodule under test. A "controlling test set" renders the surrounding logic transparent to the signals coming to the inputs of the submodule and the signal going from its outputs. To make a logic transparent to a given signal, the path on which the signal is propagated must be sensitized. A transparency cube is a collection of fixed and flexible signals that is formed when as many sensitized paths as possible are propagated from the inputs of the submodule to its outputs. A path is sensitized when there is a successful propagation of a flexible signal Dj on it. The ideal goal for the algorithm is to find a "master" transparency cube that simultaneiously sensitizes all the input and output paths of the submodule. This is not always possible. The algorithm attempts to find the minimum number of transparency cubes made of flexible and fixed signals. The minimum number of transparency cubes to completely control and observe a submodule is obtained when any combination of input and output signals to the submodule can be generated by assigning fixed value signals to the flexible Dj signals of one of the transparency cubes. The transparency cubes for a cell are shown in Fig. 8.

Example

The circuit on Fig. 3(b) is now tested using the transparency cube algorithm. The algorithm starts by partitioning the circuit into identical submodules. Each submodule is itself divided into 4 identical cells. At this point, the algorithm has identified all the submodules and cells that form the circuit. The transparency cubes of the submodules and cells are found before the complete test pattern for the unique cell is generated. The transparency cubes for the right most submodule are constructed trying to control and observe all its inputs and outputs. The submodule of interest is made of cells C00, C01, C10 and C11. As explained in the previous section, a "master" transparency cube is impossible to build since the signal on output H02 (entering cell C02) is not independent from the signal on output V12 (leaving cell C02). The signals are output and input respectively of the submodule (Fig. 3(b)). This forces the algorithm to search for multiple cubes. The cubes are created by sensitizing each input and output individually. After each successful sensitization, an attempt is made to find another sensitized input or output path using the unassigned inputs and outputs. When no more paths can be added, the combination of fixed and flexible signals on the paths to the inputs and outputs of the submodule form a transparency cube. The algorithm constructs the next transparency cubes by first sensitizing the paths that were not previously sensitized. The construction of transparency cubes ends when any combination of fixed input and fixed output signals can be found in the transparency cubes when flexible signals are assigned fixed value signals. The submodule is controlled foramt he primary inputs and observed at the primary outputs by a control cube. The control cube creates sensitized paths from the inputs and outputs of the submodule tot he primary inputs and outputs of the circuit. The two control cubes found for the right most submodule are:

INPUTS
B1 A1 B0 V01 A0 V00
D5 D6 D7 D8 D11 D10 D12 D13 D12

```
OUTPUTS
PO P1      P2 P3
D1 D2 D3 D4 I/O D3 D4
```

The signal value for HO2 is 1 in one of the control cubes and 0 in the other. The primary inputs and outputs are shown in bold letter. The other entries represent the rest of the inputs and outputs of the submodule. They will make it easier to explain the substitution of one transparency cube (for a cell) in the other (for the submoudule). As it can be seen from the transparency cubes (Fig. 8), any pattern of fixed value signals can be applied to the submodule from the primary inputs and the output observed at the primary outputs. The algorithm can then test any cell in the submodule by constructing sensitized paths to the inputs and outputs of the cell from the inputs and outputs of the submodule. This is done by finding the control cubes for the cells as shown in Fig. 8 for all the four cells. The control cube for cell C00 is written below:

```
INPUTS
B1 A1 V12 B0      A0 V00 H00
0  0   X     0    D5 D6  D3     0

OUTPUTS
H01 V11 P0 P1 V22 H13 H02 P2 P3
D1  D1  D2 D1  X   0   0  X  X
```

Where X represents a "don't care" signal and the inputs and outputs of the cell are in bold letters. The other entries are included for ease of reference in the substitution of cubes. The test pattern from the primary inputs of the circuit are constructed by "connecting" the sensitized paths form the cell to those of the submodule, by replacing the cell's control cube into the submodule's control cube. Note the Dj represents a flexible signal that can be set to 0 or 1. The results of the substitution generate the following control cube:

```
INPUTS
B1 A1 V12 B0 V01 A0 V00 H00 H11
0  0  D7  D4  0   D5 D6  D3   0

OUTPUTS
H01 V11 P0 P1 V22 H13 H02 P2 P3
D1  D1  D2 D1 D8   0   0  D9 D1
```

The inputs and outputs necessary for the control and observation of the cell are in bold letters. It's obvious that any change in the cell is propagated to the primary outputs and the cell can be completely controlled. Hence, the complete test pattern for the cell can be applied and its results observed at the primary outputs. The test pattern set is obtained by assigning fixed signals to the Dj's. The first pattern is obtained as follows:

```
INPUTS
B1 A1 V12 B0 V01 A0 V00 H00 H11
0  0  D7  1   0   1   0   1   0

OUTPUTS
H01 V11 P0 P1 V22 H13 H02 P2 P3
 1   1  1  1  D8   0   0  D9 D10
```

Where D1, D2, D3, D4 and D5 are fixed to 1 and Ai, Bi, Hi, Vi, Ho and Vo are equivalent to A0, B0, H01, V00, H00 and P0 respectively.

This computation is done for all patterns for all the submodules and cells. It is obvious that an enormous amount of repetitious computations is avoided as the algorithm constructs smaller and smaller sensitized paths as it goes from submodule level to cell level (and further if more partitioning was possible). This uses the computations done for each subpart of the circuit as the sensitized paths of the submodule are exploited time and again by the test patterns generated for the different cells.

Fault Coverage

In summary, it is clear that once the submodule (or super-cell) is controlled and observed (Fig. 6) by its transparency cubes, all the gates inside submodules can be tested if test patterns for the individual, isolated submodules are known. The algorithm does not try immediately to test each gate, but rather, it finds the transparency cubes for the cells inside the submodule. The transparency cube for cell C10 is shown in Fig. 7. Note that the other 3 submodules and their respective cells are handled in the same way as the first submodule. Once the transparency cubes are generated, they are substituted into their "parent" transparency cube as explained in the previous sections.

The test pattern set for a cell is then found and a fault simulation for the stand alone cell is performed. The test pattern set for the cell embedded in the circuit's logic is found by replacing the test pattern in the transparency cube and assigning fixed values to the flexible signals of the transparency cube. The fault coverage for each pattern is found and stored. The faults covered by each pattern can be found by analyzing the sensitized paths created throughout the entire circuit by the application of that pattern to the primary inputs. The cells at the inputs of the circuit are tested first. The controllability of a given cell and the observability of its outputs are established by the intersection of the transparency cubes

for the embedding logic surround that cell. The test patterns for the cells at the input stage are applied to the inputs of these cells. The functional response of these cells at their outputs are found and the resultant input vector to the next stage of cells is checked against the test patterns of those cells. The matching patterns are tagged. The faults detected by the tagged patterns are also detected by the test pattern generated for the input stage cells provided sensitized paths exist downstream which make the appropriate cell outputs observable. Untagged test patterns in the minimal test set for the second level of cells are then generated by giving fixed values to the flexible signals for the transparency cubes for those cells.

Controllability and observability for this next stage is guaranteed by the existence of the transparency cubes for these cells. The functional responses at the outputs of the second stage cells are again found and checked against test patterns for the next level cells. The matched patterns are tagged. The procedure continues until the primary outputs are reached. This method was used by Jerdonek et al[9] for their test vector generation algorithm. The transparency cube algorithm, constructively matching test patterns for one cell to the others, generates a compacted test pattern set with an extremely high fault coverage. This fault coverage can, therefore, be attained without exhaustive fault simulation.

Conclusion

The submodule testing program enables the user to extract and control a specified section of a cricuit in a minimal amount of CPU time. With this control, the user can easily test the section by transmitting the test pattern down the control paths and observing the outputs at the observation path ends. Through the use of the subscripted D-Algorithm and its flexible signals, testability numbers for the signal routing, and the controllability calculation modification for the EXOR gate, our algorithm minimizes backtracking and the CPU time required to obtain the "controlling test set" for the specified submodule.

The submodule testing software performed exceptionally well on systolic array topologies (tested on multipliers in particular). Here the program constructed the entire test set for the circuit in a very small amount of CPU time when other algorithms took exceptinally prohibitive amounts of CPU time. The performance on submodules could not be compared to other algorithms since there are no similar programs to our knowledge. The program also aids the engineer in the construction of a testable circuit by indicating where additional logic is required in order for the circuit to become compeltely testable.

References

[1] Ding-Hwa Ho, "Submodule Testing Using the Simultaneous Control and Observation Paths of the Subscripted D-Algorithm", Master's Thesis, Rensselear Polytechnic Institute (RPI Technical Report FTCS84-1), September 1984.

[2] A. N. Airapetian and J. F. McDonald, "Improved Test Set Generation Algorithm for Combinational Circuit Control", Proceedings International Symposium on Fault-Tolerant Computing, pp. 133-137, June 1979.

[3] C. Benmehrez and J. F. McDonald, "The Subscripted D-Algorithm ATPG with Multiple Independent Control Paths", IEEE Automatic Test Program Generation Workshop, San Francisco, CA, March 15-16, 1983.

[4] C. Benmehrez and J. F. McDonald, "Measured Performance of a Programmed Implementaion of the Subscripted D-Algorithm", Proc. of the 20th Design Automation Conference, pp. 308-315, 1983.

[5] C. Benmehrez and J. F. McDonald, "Test Set Reduction Using the Subscripted D-Algorithm", Proc. of the International Test Conference, pp. 115-121, 1983.

[6] J. P. Roth, W. G. Bouricius and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circiuts", IEEE Transactions on Computers, vol. E-16, no. 5, October 1967.

[7] L. H. Goldstein, "Controllability Observability Analysis of Digital Circuits", IEEE Transactions on Circuits and Systems, vol CAS-26, pp. 285-293, September 1979.

[8] M. A. Breuer, "The Automatic Design of Testable Circuits", IEEE Transactions on Computers, pp. 3-6, August 1983.

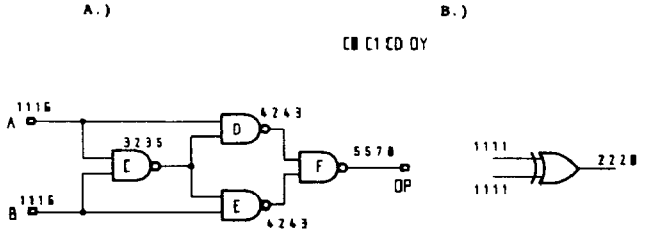
[9] H. T. Kung, "Let's Design Algorithms for VLSI Systems", Proceedings of the Caltech Conference on VLSI, January 1979.

[10] J. Savir, "Good Controllability and Observability Do Not Guarantee Good Testability", IEEE Transactions on Computers, vol. C32, no. 12, December 1983.

[11] S. E. Noujaim, R. T. Jerdonek and S. J. Hong, "A Structured Approach to Test Vector Generation", IEEE Transactions on Computers, 1984.

```
CO = Min (IjCO) + 1;    C1 = Sum (IjC1) + 1;
CD = Min (ICDj + 1);    ICDj = Sum (IjC1) + CDk;
For j=1 to M            For k=1 to M (k not=j).
NAND GATE
CO = Sum (IjC1) + 1;    C1 = Min (IjCO) + 1;
CD = Min (ICDj + 1);    ICDj = Sum (IkCO) + CDj;
For j=1 to M            For k=1 to M (k not=j).
OR GATE
CO = Sum (IjCO) +1;     C1 = Min (IjC1) + 1;
CD = Min (ICDj) +1;     ICDj = Sum (IkCO) + CDj;
For j=1 to M            For k=1 to M (k not=j).
NOR GATE
CO = Min (IjC1) + 1;    C1 = Sum (IjCO) + 1;
CD = Min (ICDj) + 1;    ICDj = Sum (IkCO) + CDj;
For j=1 to M            For k=1 to M (k not=j).
NOT GATE
CO = IC1 + 1;           C1 = ICO + 1;
CD = ICD + 1.
```

Figure - 1 These equations represent the output node controllability measures for the standard cells including controllability D.



C.)

CO(OP) = MIN{[CO(A) + CO(B)],[C1(A) + C1(B)]} + 1

C1(OP) = MIN{[CO(A) + C1(B)],[C1(A) + CO(B)]} + 1

CD(OP) = MIN{ EXPA, EXPB, EXPC, EXPD } + 1

where

EXPA = CD(A) + CO(B)

EXPB = CD(A) + C1(B)

EXPC = CD(B) + CO(A)

EXPD = CD(B) + C1(A)

Figure - 2 Testability numbers CO, C1, CD and observability number OY for A) the miscalculated 3-level deep EXOR gate, B) the modified 1-level deep EXOR gate, and C) the modified EXOR testability equations.

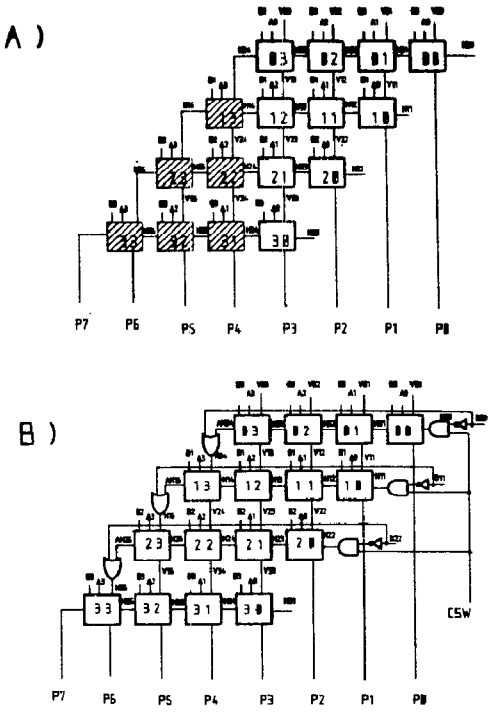


Figure - 3 A.) The BW multiplier - the 6 crosshatched cells are not totally controllable or observable, B.) Modified 100% testable BW multiplier (made using the program's placement suggestions).

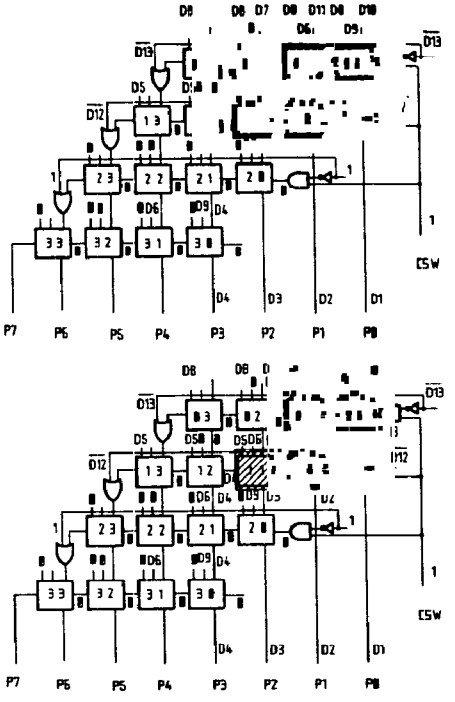


Figure - 4 Controlling test set for the 4x4 BW multiplier.

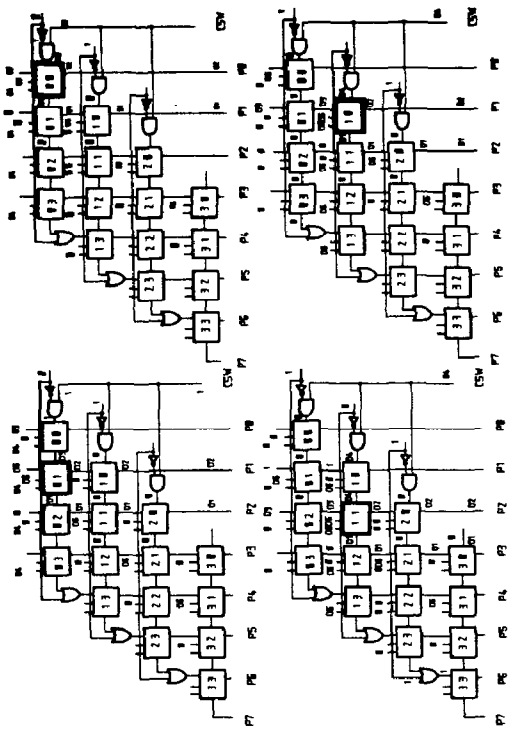


Figure - 5 Controlling test set for the 4 cells in one submodule.

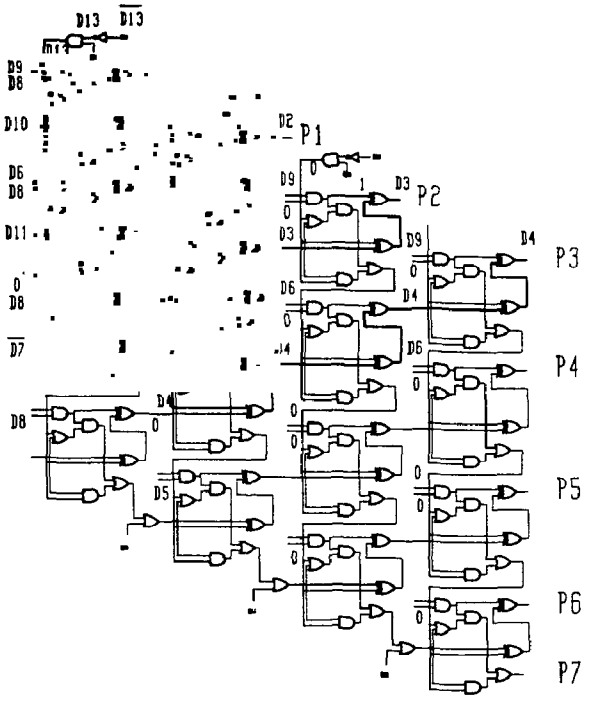


Figure - 6 Controlling test set for the submodule showing the sensitized control and observation paths (bold lines).

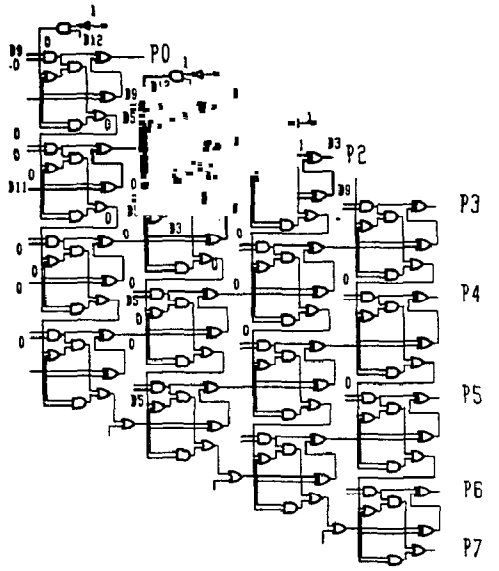


Figure - 7 Controlling test set for the cell C10 in the previous submodule. Sensitized paths are shown in bold lines.

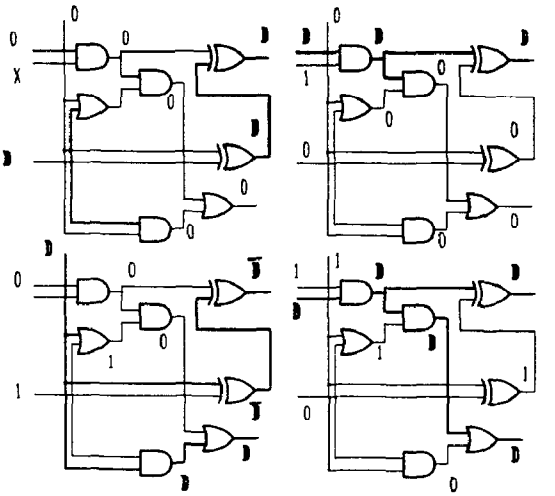


Figure - 8 Transparency cube for a cell. Sensitized paths are shown in bold lines.