VLSI DESIGN SYNTHESIS WITH TESTABILITY

Catherine H. Gebotys and Mohamed I. Elmasry

Department of Electrical Engineering,
University of Waterloo, Waterloo, Ontario. N2L 3G1 Canada

ABSTRACT

A VLSI design synthesis approach with testability, area, and delay constraints is presented. This research differs from other synthesizers by implementing testability as part of the VLSI design solution. A binary tree data structure is used throughout the testable design search. Its bottom up and top down tree algorithms provide datapath allocation, constraint estimation, and feedback for design exploration. The partitioning and two dimensional characteristics of the binary tree structure provide VLSI design floorplans and global information for test incorporation. An elliptical wave filter example was used to illustrate the design synthesis with testability constraints methodology. Test methodologies such as multiple chain scan paths and BIST with different test schedules were explored. Design Scores comprised of area, delay, fault coverage, and test length were computed and graphed. Results show that the 'best' testable design solution is not always the same as that obtained from the 'best' design solution of an area and delay based synthesis search.

1. INTRODUCTION

Current commercial VLSI design automation tools used for the design of complex VLSI systems lack the capability to handle early architectural design exploration and test implementation. Both these problems cause longer VLSI design cycles. For example, architectural design exploration is usually limited in industry due to tight design time schedules. This lack of performance feedback in different architectures can cause problems at later stages of the design cycle such as exceeding expected chip areas or the inability to achieve required throughputs. Currently large efforts at later stages of the design cycle are required for test implementation. Testability analysis tools, such as SCOAP [1], which were designed to support testability incorporation during design stages actually provide poor predictions of testability [2] and do not suggest how and what test methodologies should be applied. Larger and faster design exploration that incorporates testability early into the design solutions would clearly be advantageous.

2. PROBLEM DESCRIPTION AND PREVIOUS RESEARCH

We propose a solution to the following problem. Given a general algorithmic description of a behavior with area, delay, and test constraints, perform a design synthesis by mapping the algorithm into a chip design which satisfies the given constraints. This will create a testable design using area, speed, and testability estimates to guide the search through the design space.

Most research on design synthesis has not included testability incorporation. Only estimates of area and delay have been examined to provide feedback into the design search. The DAA [3], FACET[4], and BUD[5] approaches provide design synthesis however no test incorporation is performed.

Although many researchers stress that testability should be considered during the early stages of design [6], most testability research has been done after a structural design solution is defined with no feedback to the original synthesis process for finding more testable designs. Testability incorporation in the Silc silicon compiler [7] utilizes testability measures at the functional, and structural levels to guide test incorporation however no feedback to the synthesis process is provided and no other test methodology for the data path is considered apart from the scan path inclusion. The testability design expert system [8] implements testability in a graph-based structural design however no feedback to a synthesis process is used and the test implementation is based on local structural enhancements with no global information. Built in exhaustive self test incorporation [9] in data paths provides speed estimates but no design synthesis or feedback is provided.

Our VLSI design synthesis methodology with testability constraints, CATREE (for computer aided trees), enhances the state-of-the-art in the area of VLSI design synthesis with testability constraints by including the following features.

1. Testability is implemented as part of the VLSI design solution with test cost, silicon area, and delay estimates used as feedback to guide the design synthesis search.
2. A two dimensional binary tree data structure [10] is used throughout complete synthesis and testability incorporation holding algorithmic and structural information. Design hierarchy, partitioning, and two-dimensionality naturally represented with the data structure are used to advantage for design solution searches, constraint estimation, and test metho-

dology incorporation.

3. This design and test methodology provides a larger, more complete, and flexible design search. By incorporating testability into the synthesis stage the overall design constraints can be better satisfied.

CATREE allows the exploration of the effects of different test methodologies on a specific design, and the effects of a specific test methodology on different design solutions. It is also vital to the acceptance of synthesis tools in industry by providing automatic feedback to the synthesis process when test overheads exceed design constraints. Otherwise the design problem is left unsolved with the emphasis on the user to determine what has been done by the synthesizer and how one may reinvoke it to provide a different solution.

The CATREE design synthesis with testability constraints approach uses a synthesis search based on area and delay constraints followed by a testability search based on area, delay, and test cost constraints that finds a solution or returns to the synthesis search for a more testable solution. This methodology is shown in figure 1.

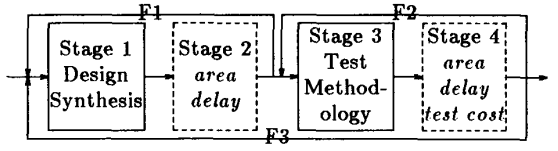

Figure 1. The CATREE VLSI design methodology with testability constraints.

Stages one through four corresponding to the design synthesis, area and delay estimation, test methodology implementation, and area, delay and test cost estimation stages respectively, will be discussed next.

3. DESIGN SYNTHESIS

The synthesis stage consists of parsing the input specification, scheduling operators, and allocating functional units, registers, and interconnect. Each will be discussed in more detail below.

The input specification, written by the user, provides the algorithmic code sequence, design constraint specifications, design library identification, initial schedule name, and a list of test methodologies to explore. Figure 2 gives an example of the current input specification. Currently only straight line code segments are synthesized. The user may specify extra arcs in the code sequence to examine tradeoffs between cycle time and the number of functional units. The library contains the list of operations each functional unit can perform, the number of cycles required by each operation, and parametric area, delay, and test data for each functional unit.

An initial, as soon (asap) or as late (alap) as possible, schedule is implemented upon entry into the synthesizer [11]. The code sequence is transformed into a directed acyclic graph of operators based upon data precedence and any user defined arcs. Extra dummy operators are used with multicycle operators, since each operator is assumed to require one cycle, and data precedence is used for proper scheduling. The graph is traversed top down or bottom up assigning firetimes for each operator and thus creating an asap or alap schedule respectively. Rescheduling may be invoked after stages two and four of figure 1 by specifying an operator and new firetime. Operators may be rescheduled as long as the minimum number of control steps required by the algorithm is maintained. For example an operator on the critical path cannot be rescheduled.

```
module      filter1( inout i2,i13,i33,i38,i39,i18,i26: byte;
                     in i1 : byte;
                     out o43 : byte );
var         a,b,c,d,e,f,g,x,w,a1,y,m : byte;
constraints area = 2000, delay = 500,
            fault_coverage = 95, test_length = 20000;
library     generic_PLmult;
schedule    asap;
test_method bist, scanpath;
begin       a := i1 + i2;
            b := a + i13;
            c := b + i26;
            d := i33 + i39;
            ...
            i18 := i18 + x;
            i13 := i18 + w
end.
```

Figure 2. Input specification partially shown for CATREE synthesis of a filter.

The binary tree data structure is formed by placing operators into leaf nodes of a tree according to their degree of shared variable connectivity [11]. The extra dummy operators of multicycle operators are not placed in the tree. The sum of the variable connectivities for each operator is used to order a list of operators from high to low connectivity. The two most connected operators are used to form the initial tree. Other operators are then selected from the ordered list and placed in a new leaf node close to existing operators in the tree to which they are most connected. Since nodes will be swapped or moved and merged during the design synthesis search, this simple tree formation algorithm appears to be sufficient. The tree algorithms attempt to decrease the complexity of the allocation algorithms by decreasing the wide range of cluster group choices and biasing it towards solutions with minimum interconnect without sacrificing quality.

A bottom up tree traversal algorithm [11] collects operators from the leaf nodes with nonconflicting firetimes and valid functional unit representation. In the tree functional units are thus represented by subtrees of their operators. The firetimes used for multicycle operators consists of all active cycle times, whereas firetimes of pipelined operators consist of the first cycle time only. Subtree reclustering of operator leaves can be performed to further minimize the number of functional units. The flexible data structure allows exploring different functional unit configurations for a particular schedule.

Register allocation [11] uses a bottom up tree traversal algorithm with variable cluster rules activated at each node of the tree. The output variable of each operator is placed in a list and propagated up from the leaf nodes. Generally, variables or clusters of variables are merged into one cluster if their lifetimes do not overlap. All left and right operands of the same functional unit are clustered separately. An example of variable clustering is shown in figure 3. At the root of the tree the clustered list is merged one last time with a list of input only variables. Each register, represented by a cluster of variables, is placed in a new leaf node closest to the functional units to which it is most connected. Constants are placed in tree nodes in the same manner. Variables passed between registers and functional units, called cross variables, are recorded at nodes in the tree whose two subtrees hold the register and functional unit. Examples have shown that this algorithm produces the minimum number of registers [10,11]. Registers also tend to be uniformly allocated over the groups of highly connected functional units. This provides reasonable results for the test incorporation described in section 5.

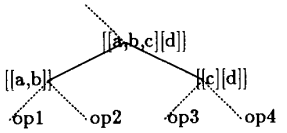

Figure 3. Bottom up variable clustering in the binary tree structure. Lifetimes of the variables (a,b,c,d) propagated from operator leaf nodes (op1,op2,op3,op4) determine register allocation performed.

A top down tree algorithm allocates interconnect for a random topology or bus implementation by using cross variable cluster rules activated at each tree node. Two top down tree traversals are done allocating interconnect from functional unit output to register input, and from register output to functional unit input. There are four cluster rules used for bus allocation and two of these are used for the random topology allocation. For example the final cluster rule for bus allocation states that two clusters can merge if the transfer times of different cross variables do not overlap. Bus allocation can also be done using one tree traversal where cross variables transferred to and from functional units at different times can be allocated to the same bus. Table 1 shows the random topology and bus design styles for the CATREE synthesis solutions of various code sequences [12,13]. A heuristic multiplexor minimization algorithm was also written. It swaps cross variables of commutative operators between clusters to minimize the number of multiplexor inputs. The algorithm also implements multiple level multiplexor allocation if a decrease in the number of multiplexor inputs is achieved. The top down interconnect algorithm attempts to minimize the number of interconnect which are most likely to be long since they are located at high levels in the tree.

Table 1. CATREE random and bus design styles.

Synthesis Example	#FUs	#Regs	Random #MuxIns	Bus #Buses
DE	5	9	13	6
Magg	3	3	13	4
f1	6	14	33	7

4. AREA AND DELAY ESTIMATES

Area and delay estimations [11] are obtained by creating a floorplan, performing a bottom up area estimate, and then extracting delays of circuit paths defined in the tree. The floorplan algorithm moves top down through the tree alternatively assigning X and Y split dimensions. Two son nodes which are split by the X dimension, are given low or high parameters by calculating each son node's connectivity to their X dimension neighbour node. The son with the higher connectivity is chosen as the low or high son which will lie closest to the neighbour node. An estimate of area can be then calculated using a bottom up tree traversal algorithm summing the minimum bounding boxes or areas of functional units, interconnect, multiplexors and registers. Delay estimates are calculated by outputting paths through the design, consisting of functional units, fanout, registers, multiplexors and interconnect lengths, and computing their delays.

If estimates of area and delay do not meet the constraints, resynthesis is invoked by rescheduling operations and reallocating functional units. When the area constraint is not met, a bottom up tree traversal algorithm is used to collect functional units which can be merged together by rescheduling their conflicting operators. Also the user may selectively choose which functional units are to be merged or split if for example delay is not met. The 'best' solutions are stored in case no further beneficial merging can occur.

5. TEST METHODOLOGY IMPLEMENTATION

If the area and delay estimates meet the design constraints, testability incorporation is explored. The test methodologies selected for implementation are given in the input specification. Variations of scan path and BIST methods can be implemented. Design partitioning, naturally represented in the binary tree data structure, is used to implement multiple scan chains, test scheduling, and nonuniform test incorporation of these methodologies. It is currently assumed for simplicity that the test methodologies are implemented to allow test patterns to be applied to, generated at, or observed at, the functional unit inputs or outputs.

Each test methodology will provide a different set of rules for identifying test registers. Single output and input registers of functional units are ranked highest in being transformed into test registers for that functional unit. The remaining assignments of registers to be transformed into test registers are then made. For simplicity it is assumed that constants may be transformed

into test registers. Due to conflicts between register use among functional units, the allocation of new multiplexor extensions, multiplexors, and test registers (which become new leaves in the tree) can be done.

The implementation of multiple scan chains or nonuniform test methodologies uses the same algorithm outlined above except it is applied to subtrees representing different partitions of the design. For example a double scan chain methodology applies the algorithm to each of the two subtrees located one level below the tree root.

The definition of single or multiple scan chains is obtained after test register assignment. A bottom up traversal of the tree or subtree structure is done listing test registers as they are encountered. Scan chains are thus formed from the lower left to the upper right corners of the floorplan. This algorithm can be changed to implement scan chains in different configurations through the chip floorplan. The tree is ideally suited for these computations due to its natural partitioning and two-dimensional characteristics.

Test scheduling can also be implemented using the tree data structure. Test scheduling refers to the schedule for testing groups of functional units sequentially in more than one phase. This algorithm works bottom up collecting subtrees of X or less functional units, where X represents the number of phases required to fully test the design. Each of the X functional units will be tested during separate phases. Reasonably low overheads are obtained since the functional units found within a subtree are highly connected to their local registers and therefore suited to sharing them.

The partitioning and two-dimensionality of the binary tree data structure provides global information which aids in test incorporation, test scheduling, and scan chain definition.

6. AREA, DELAY, AND TEST ESTIMATIONS

New area, speed, and test cost estimates are obtained after test incorporation. The area and delay estimates use the same algorithms outlined earlier in section 4 on the current binary tree with test incorporation. For illustration purposes the test cost includes measures of the fault coverage and test time of the VLSI design. Furthermore it is assumed that each functional unit is characterized in the VLSI data base with a measure of total number of faults, and for each test technique fault coverage and test time measures. Thus the total fault coverage estimation is computed from total and detected faults of the functional units, multiplexors, and registers. Different test methodologies are applied in an attempt to satisfy the area, delay, and test cost constraints.

If the test cost constraint has been met, the strategies outlined in section 4 for feedback after stage two are used. This creates feedback path F3. However if the test cost constraint is not met another test methodology or variation is reimplemented using feedback path F2. When all test methodologies are exhausted and the constraints are still not met feedback path F3 is used to resynthesize the design. This provides wide design exploration for testable designs.

7. EXPERIMENTAL RESULTS

An elliptical wave filter (EWF) example presented for synthesis in [12] was used to illustrate the VLSI synthesis with testability constraints. This example has been chosen as a benchmark for high level design synthesizers [12]. A basic block code sequence for the EWF, partially shown in figure 1, of 34 operators was derived from the precedence graph in [14] with correction by [12]. In figure 1, feedback path F1 representing the design synthesis search with area and delay constraints has been analyzed in several papers [3,5]. We will concentrate on showing results for feedback paths F2 and F3 of figure 1. Feedback path F2 illustrates the test methodology search of one design driven by area, delay and test cost whereas feedback path F3 shows the exploration of testable design solutions driven by the area, delay and test cost constraints.

To illustrate the effects of these feedback paths on the testable synthesized design search, four parameters: fault coverage, test length, area, and delay values; were recorded, normalized, weighed, and summed [15] to obtain design scores for each solution. The assumptions made to calculate these four parameters are given in [11]. Thus the highest score represents the best design solution for those weights.

Figures 4 through 6 show the CATREE synthesized design solutions with test incorporation for the EWF. The multipliers were assumed to be two cycle pipelined multipliers and an initial asap schedule was specified. Testable design solution details for the EWF can be found in [11]. Table 2 shows the number of cycles required by each design solution to implement the algorithm. The algorithm pipelining in this table done for f3, f4, and f5 was implemented by changing the code sequence. The delay parameter used in figures 5 and 6 refers to the maximum delay path for one cycle and does not take the overall number of cycles to implement the algorithm into consideration. The last design solution, f5, was created by adding two arc constraints to the pipelined code sequence of f4, in order to reduce the number of multipliers to one.

Table 2 shows the explicit design solutions in comparison with previously presented results [12]. The CATREE synthesized solutions perform fairly well but tend to have a larger number of multiplexors than the HAL solutions. The cpu time was about 20s for one completely synthesized solution with test incorporation.

Table 2. HAL[12] and CATREE synthesized EWF solutions.

	HAL		CATREE				
Design			f1	f2	f3	f4	f5
#Cyc	17	19	17	17	16	16	17
#Mult	2	1	2	2	2	2	1
#Add	3	2	4	3	4	3	3
#Regs	12	12	12	12	13	13	12
#Mxin	31	26	33	38	36	35	44

The schedules for the referenced results were not available and would most likely be different than those of CATREE.

Figure 4 shows only the functional units in the binary tree and the corresponding floorplan created for the filter solution f1. The scan chain definition for a single chain scan path is given.

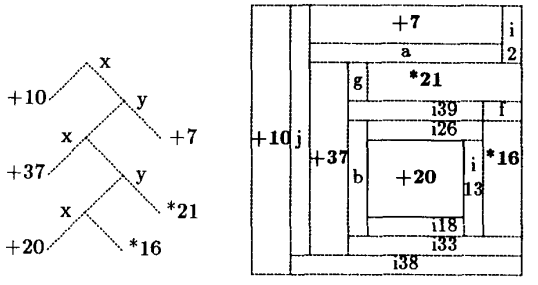

Figure 4. Floorplan and corresponding binary tree showing functional units only for f1 filter solution. Scan chain for single chain scan path is [i1,j,i33,b,i13,i26,i39,f,2,g,a,i2].

Figure 5 shows the design scores for five test methodologies applied to one EWF design solution. The scan path with two chains is the best testable design solution over all different weights. The second best solutions vary according to which element of the four measures is most important. The single chain scan path is desirable when area and delay are most important while the two phased BIST solution is preferable when test cost is more important.

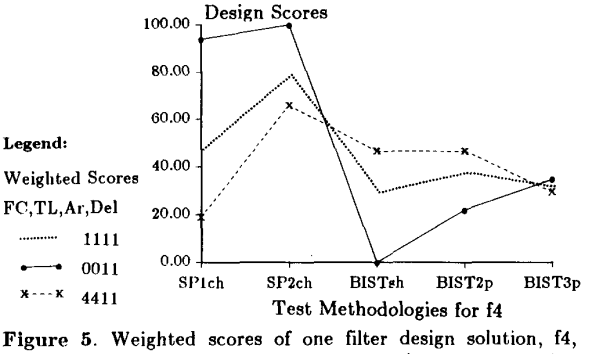

Figure 5. Weighted scores of one filter design solution, f4, with five different test methodologies (single and double chain scan path, and shared, 2 phased, and 3 phased BIST).

Figure 6 shows design scores for the four EWF design solutions based upon area and delay before test implementation, and area, delay, fault coverage, and test length after test incorporation. The design scores are based upon equally weighted parameters. f4 is the best solution when only area and delay are considered before test incorporation. The best test methodology over all filter design solutions is the double chain scan path. The second best is the 2 phased BIST methodology. For all test methodologies, solution f4 is the best testable design solution except in the case of the 3 phased BIST implementation. The best testable design solution for this later test methodology is f3. This is due to a higher delay and lower test score associated with the 3 phased BIST test implementation of design f4 as compared to f3.

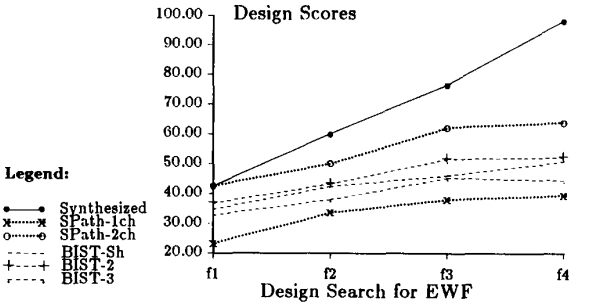

Figure 6. Design Scores for equally weighted fault coverage, test length, area, and delay measures for the filter synthesized testable design search. Synthesized solid line refers to the design before test incorporation with equally weighted area and delay parameters.

8. DISCUSSION AND CONCLUSIONS

Observations of the EWF example are also consistent with CATREE exploration of another synthesis example, the differential equation (DE) [11]. Generally, when equally weighted parameters are used the scan path methodology appears to be the test methodology to explore. Another observation is that weighing the test cost higher (4411) produces 'best' testable design solutions using the scan path methodology in EWF or BIST methodology in DE. In both cases the implementation details, such as number of chains or test schedule, will vary, hence test implementation of a particular methodology should be fully explored. In the EWF example the 'best' design solution before test incorporation is f4. The 'best' testable design solution is obtained from either f3 or f4 depending upon the test methodology implemented.

In general we have seen that the best solution for the area-delay based synthesis alone does not always lead to the best testable design solution for all test methodologies. Thus feedback after test incorporation to the synthesis process, feedback path F3, is important for finding good testable design solutions. Secondly, the best synthesized design solution from which the testable design is obtained will often vary for different test methodologies. Hence all test methodologies must be considered, feedback path F2, for all possible design synthesized solutions in a search for a testable design satisfying area, delay and test cost constraints.

The actual weights assigned to the four parameters will directly effect which design becomes the best solution. In practise the weight values must be assigned according to which parameters are the most critical, which in turn would depend upon the application.

If a test-characterized data base library were not available then the test cost parameters, fault coverage and test length, would have to be determined from test

software such as an automatic test pattern generator or fault simulator. Test cost could also be extended to include test confidence and input/output pin counts. Note as in section 5, the assumption concerning implementation of the test methodology, may cause test overheads to be significant such that all constraints cannot be satisfied. In these cases interface to test software, which for example selects a minimum number of nodes for the scan path, would be necessary. Another current limitation of this research occurs when the test cost constraint cannot be met after exhaustive test implementations. A possible solution would be to use a more testable functional unit version from the library or again to interface to test software for fault coverage improvement or for decreasing the test length.

This preliminary research, CATREE, is aimed at providing a framework for integration of synthesis and testability. Previously research in these two areas have remained separate. This integration has implications for work in both industry and research. In industry, valuable design cycle time would be saved by providing wider exploration of design solutions with area, delay and test cost constraints satisfied earlier in the VLSI design cycle. In research, it has been shown that testability should be considered an important part of the design synthesis search along with area and delay. Thus feedback path F3, along with F1 and F2, should be used when synthesizing designs for an algorithmic description.

In summary the CATREE design synthesis provides a useful approach towards incorporating testability within the synthesis process using a two dimensional binary tree data structure. The tree data structure provides allocation algorithms of reasonable complexity without sacrificing quality [11]. Integration of the VLSI data base is achieved through a common binary tree data structure used in complete synthesis with testability unlike a combination of separate synthesis and testability tools such as [5,4] with [8]. The binary tree data structure also supports automated feedback to the synthesis process after test exploration. Finally the two dimensionality naturally represented within the binary tree data structure supports testability incorporation and area-delay estimation, unlike other structures lacking this property [8,4].

Ongoing research involves extending CATREE into a single feedback simultaneous design and test synthesis methodology. This would be applicable to system design where one fully specified test methodology is given in the input specification at the beginning of the design cycle. Future research would include extending the synthesis to fully programmable algorithmic input using trace scheduling techniques to optimize hardware implementation for high probability traces. Also extending automatic feedback strategies, and automating testability measures earlier in the synthesis process, ie. evaluating test measures for chained operations would be investigated in the future. CATREE has been implemented in Quintus Prolog. Special thanks to Dr. R.J.Gebotys for his helpful comments. This work was supported in part by research grants from NSERC, BNR, and GE.

REFERENCES

[1] L.H.Goldstein and E.L.Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program", *Design Automation Conference*, 1980.

[2] I.M.Ratiu, A.Sangiovanni-Vincentelli, and D.O.Pederson, "VICTOR: A Fast VLSI Testability Analysis Program", *IEEE Test Conference*, 1982.

[3] D.E.Thomas, C.Y.Hitchcock, T.J.Kowalski, J.V.Rajan, and R.A.Walker, "Automatic Data Path Synthesis", *IEEE Computer*, December 1983.

[4] C.Tseng and D.P.Siewiorek, "Automated Synthesis of Data Paths in Digital Systems", *IEEE Transactions on Computer-Aided Design*, July 1986.

[5] M.C.McFarland, "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions", *Design Automation Conference*, 1986.

[6] T.W.Williams and K.P.Parker, "Design for Testability - A Survey", *Proceedings of the IEEE*, January 1983.

[7] H.S.Fung and S.Hirschhorn, "An Automatic DFT System for the Silc Silicon Compiler", *IEEE Design and Test*, February 1986.

[8] M.S.Abadir and M.A.Breuer, "A Knowledge-Based System For Designing Testable VLSI Chips", *IEEE Design and Test*, August 1985.

[9] A.Krasniewski and A.Albicki, "Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules", *International Test Conference*, 1985.

[10] C.H.Gebotys and M.I.Elmasry, "A VLSI Design Methodology with Testability Constraints", *Canadian Conference on VLSI*, Manitoba, Canada, Oct 1987.

[11] C.H.Gebotys and M.I.Elmasry, "VLSI Design Synthesis Exploration With Testability Constraints", Tech. Rep., UW/ICR 87-14, Department of Electrical Engineering, University of Waterloo, Waterloo, Ontario, Canada, December 1987.

[12] P.G.Paulin and J.P.Knight, "Scheduling and Allocation For Behavioral Synthesis of Pipelined ASICs", *Canadian Conference on VLSI*, Manitoba, Canada, October 1987.

[13] H.Trickey, "Flamel: A High-Level Hardware Compiler", *IEEE Transactions on Computer Aided Design*, March 1987.

[14] S.Y.Kung, H.J.Whitehouse and T.Kailath, "VLSI and Modern Signal Processing", Prentice-Hall Information and Systems Sciences Series, 1985, Pg 257-264.

[15] M.A.Breuer and Xi-an Zhu, "A Knowledge based System for selecting a test methodology for a PLA", *Design Automation Conference*, 1985.