Hierarchical Test Generation with Multi-Level Decision Diagram Models[1]

Gert Jervan Antti Markus Jaan Raik Raimund Ubar[2]

Tallinn Technical University, Estonia
Dept. of Computer Engineering
e-mail: jaan@pld.ttu.ee

Abstract

A hierarchical test generation approach to digital circuits that uses Register-Transfer (RT) and gate level information as input is presented. The proposed test generator implements a novel approach based on Decision Diagram (DD) models. Uniform modeling procedures are used on both levels, as well as for datapath and control parts. Experimental results showing the efficiency of the approach and comparison with other approaches are provided.

1 Introduction

While test generation for combinational circuits was considered to be a solved problem already by the end of 1980s [1, 2], test generation for sequential circuits still remains a major challenge. There have been many different approaches to sequential circuit test generation proposed over the years. They can be divided into different cathegories according to the abstraction level of the input (behavioral [3], architectural [4], gate-level [5]) and test generation strategy used (deterministic [5], simulation-based [6, 7]). Some of the approaches implement information from several (usually two) abstraction levels and are refered to as hierarchical test generators [8, 16]. Despite of the diversity of solutions offered to solve the problem, the achieved fault coverages tend to be low and test generation times long for more complex circuits.

In current paper, a novel hierarchical test pattern generation approach that uses architectural (RTL) and gate level descriptions as its input model is presented. Differently from known approaches, control and datapath parts, as well as different types of modules in the datapath, i.e. registers, Functional Units (FU) and multiplexers, are handled in a uniform way. This feature simplifies the structure of the hierarchical test generation algorithm, which would target the entire circuit under test. Current approach implements a top-down method, where data and control constraints are extracted at the higher level to be considered when deriving tests for components at the lower level. The path activation algorithm for the top-down approach is explained in Section 3.

The proposed test generator is based on using generalized decision diagrams (DD), called previously by the author Alternative Graphs (AG) [9, 10]. This model was

[1] This work has been supported by the Estonian Science Foundation grant G-1850 and German-Estonian project EST 008-96 funded by BMFT Germany.

[2] Currently on the leave at Université Joseph Fourier/TIMA Laboratory, Grenoble, France.

proposed for supporting mixed symbolic and topological approach [11] and for combining this approach with hierarchical technique. Moreover, DDs support uniform approach to test design at different system levels. The same DD-approach was used to generalize gate-level test generation algorithms for higher functional or behavioral levels [12, 10] and for hierarchical approaches [13].

Logic level AGs [14, 10], referred to also as Structurally Synthesized BDDs (SSBDD), represent inherently the topology of gate-level circuits, and therefore, differently from BDDs [17] that represent function only, they support test generation for gate-level structural faults without representing these faults explicitly. The worst case complexity for generating SSBDDs is linear in respect to the number of logic gates, while it is exponential for 'classical' BDDs. Due to the fact the model generation times can be neglected. Furthermore, it is necessary to generate DD models only once and for the fault-free circuit only.

Principal difference between combinational and sequential circuits lies in the fact that in sequential circuits feedback loops occur. If an algorithm is not capable of proper handling of the feedback loops, it can not cope with the complexity of highly sequential circuits. In current approach we used simple testability measures to reorder the nodes in high-level DD models so that the feedback loops were entered only in the case if all the other choices failed. This feature is absolutely necessary if we want to avoid the situation where a path activation procedure enters an eternal loop that was activated due to a consistent but false decision. In Section 4 this type of situation is explained by the example of fault effect propagation.

The paper is organized as follows. Section 2 gives a short overview of using DD models for describing digital systems. Section 3 describes the test generation algorithm based on multi-level decision diagram models. In Section 4, an example for explaining current test generation approach is presented. Finally, in section 5, experimental results are given. The experiments show that the uniform DD-based algorithm offers significantly better performance than the list of state-of-the-art test generation approaches it was compared to.

2 Decision Diagrams as a System Model

A Decision Diagram (DD) is defined as a non-cyclic directed graph whose nodes are labeled by variables (constants or algebraic expressions). For each value from a set of predefined values of a non-terminal node variable, there exists a corresponding output branch from the node. Consider a situation where all variables are fixed to some value. By these values, for each non-terminal node a certain output branch is chosen entering into a successor node. Let us call these connections between nodes - *activated branches* and the chains of them - *activated paths.* For each combination of values of variables, there exists a *main activated path* from the root node to some terminal node. This relation describes a mapping from a Cartesian product of the sets of values for variables in all nodes to the joint set of values for variables in terminal nodes. Therefore, by DDs it is possible to represent arbitrary functions $Y=F(x)$, where Y is the variable whose value will be calculated on the DD and x is the vector of all variables in nodes of the DD.

As a hierarchical input to the test generator are descriptions where the architecture of the circuit is described at the RT-level and the low-level structure is given at the gate level. Both these levels can be described by DD models. In the RTL descriptions, designs are partitioned into datapath and control parts, where datapath is represented structurally by a netlist of interconnected blocks. The building blocks of datapath are registers, multiplexers and functional units (FU), where functions can be arbitrary arithmetic or logic operations.

Datapath can be represented by a system of DDs, where for each primary output and for each register, a DD corresponds. In the DD models, the non-terminal nodes correspond to control signals and terminal nodes represent operations. Register transfers and constant assignments are treated as special cases of operations. Activated branches between the nodes determine, which operation is assigned to the variable represented by DD with each value combination of the control signals. Figure 1 shows an example of a DD representation for a datapath register.

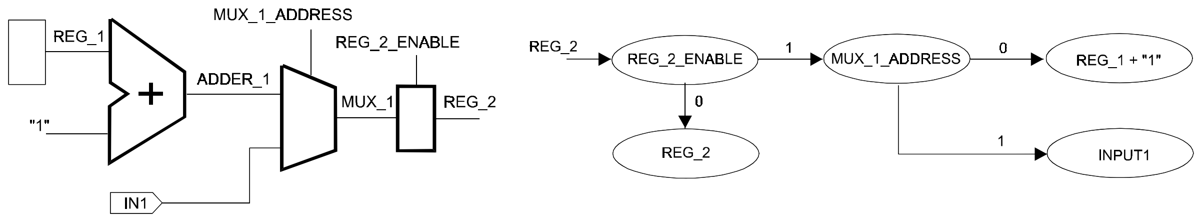


Figure 1. DD representation for a Datapath Fragment

The control part of an RTL description is described as a Finite State Machine (FSM) state table. Similar to datapath, the state table can be represented by a DD model. In that case, the non-terminal nodes correspond to current state and conditions (FSM inputs) and terminal nodes hold vectors with the values of next state and control signals (FSM outputs). Figure 2 shows an example of a fragment of an FSM state table and the corresponding DD representation. In the DD, *q* denotes the next state and *q'* denotes the current state value. Variables out1, out2, out3 and out4 are output signals of the FSM. The DD in Figure 2 describes the behavior of the FSM at the current state being equal to s5.

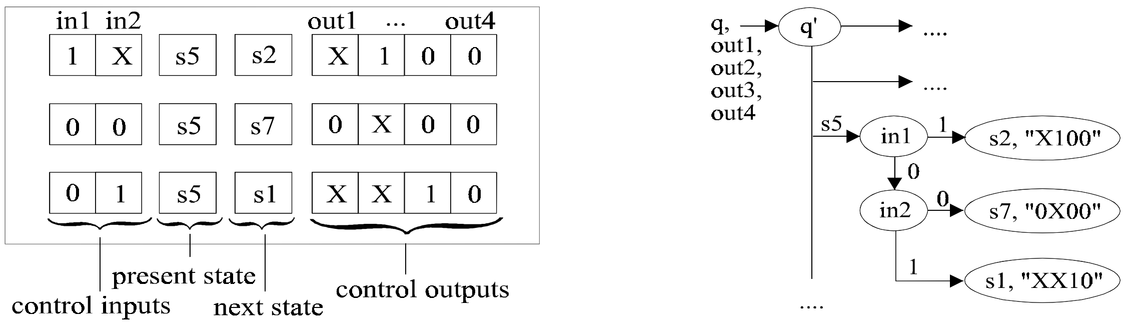


Figure 2. Representing an FSM State Table by a DD

In current hierarchical test generation approach, gate-level descriptions of the datapath modules are transformed into Structurally Synthesized BDD (SSBDD) models. Differently from BDDs, which represent function only, SSBDDs support test generation for gate-level structural faults without representing these faults explicitly. Furthermore, the worst case complexity for generating SSBDDs is linear in respect to the number of logic gates, while it is exponential for BDDs

SSBDD models for combinational circuits can be synthesized by a simple superposition procedure. We generate an SSBDD for a circuit output by starting from the output, substituting recursively all the gates by their respective elementary BDDs until primary inputs are reached. In order to avoid repetitive occurrences of subdiagrams in the model, the recursion can be terminated in fanout branches and SSBDDs can be synthesized for each primary output and fanout point separately. In that case the circuit will be described as a system where for each fanout-free region an SSBDD corresponds.

3 Hierarchical Test Generation with DD Models

The high-level path activation, proposed in current paper is a complete algorithm, i.e. if transparent paths for fault effect propagation and value justification exist, they will be activated. The test generation algorithm has been implemented as a systematic search and therefore an inconsistency in any stage causes a backtrack and a return to the last decision. However, due to the NP-complete nature of the problem, in some cases, the search must be terminated after a certain maximal number of solutions have been tried. The hierarchical test generation algorithm consists of five main stages. These are fault manifestation, fault propagation, constraint justification, constraint satisfaction and low-level test, respectively. In the following, the different stages are explained..

Fault Manifestation. As it was mentioned above, there exist two types of nodes in DD models: terminal nodes and non-terminal nodes. Appropriate tests for the corresponding types have to be set up during the fault manifestation stage. The two types of tests are referred to as *scanning test* and *conformity test*. Scanning tests are applied to terminal nodes and their aim is to test the functional units (FU), registers and constants of the datapath. Conformity tests are set up for non-terminal nodes and they target the multiplexers of datapath as well as control signal decoders in the control part.

During the scanning test, a terminal node under test is selected. The path to the node is activated in respective DD. The symbolic fault-effect value is assigned to the variable corresponding to the DD, and symbolic local test pattern values are assigned to the arguments of the function labeling the node under test. These symbolic values are later treated as justification objectives. Figure 3 presents a simple example where scanning test is performed for the node *Op.1* in the DD *Register*. The blocks where faults are targeted by the scanning test are marked with gray areas in the figure.

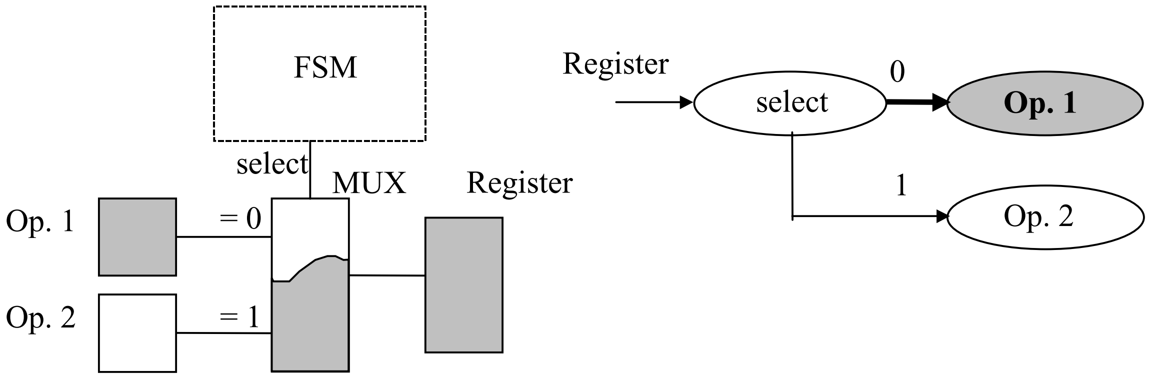


Figure 3. Scanning Test

Conformity test is similar to scanning test in the way that a path is activated to the node under test, and the fault effect value is assigned to the DD variable. In addition, the successor nodes of the node under test must be distinguished from each other. The symbolic values to be backtraced during the justification are assigned to the variables labeling corresponding terminal nodes of the DD. Due to the fact that conformity test for a node must be carried out with each value of the control signal under test, multiple conformity tests must be set up for each non-terminal node.

There could exist some additional, technology dependent, criteria included to the conformity tests. For example, in the case of AND-OR type multiplexers that most of the VLSI technologies are utilizing, the set of conformity tests for a node must cover zeros in all bit positions of its selected successor nodes. An example in Figure 4 illustrates the conformity test for the control signal *select*. The blocks where faults are targeted by the test are marked with gray areas.

Fault Effect Propagation. The aim of the propagation procedure is to determine the state sequence necessary to propagate the fault effect symbol to a primary output and to extract the logic conditions that must be satisfied at different time steps. Basing on the values assigned to control signals during the manifestation phase, a terminal node of the FSM DD is chosen, which provides the initial state and the control vector. Subsequently, a node is chosen from the set of nodes containing the variable to which the fault effect symbol has been assigned. A path is activated to the node in corresponding DD. According to that path, values are assigned to respective control signals. Again, basing on these values a consistent FSM DD terminal node providing current state and control vector is chosen. The procedure will end when the fault effect value reaches a primary output.

In current test generation approach, all the symbolic path activation procedures (manifestation, propagation, justification) are implemented as interactive choices between datapath and control part DDs. Activated paths in DDs make it possible to determine relevant variable assignments at each time step.

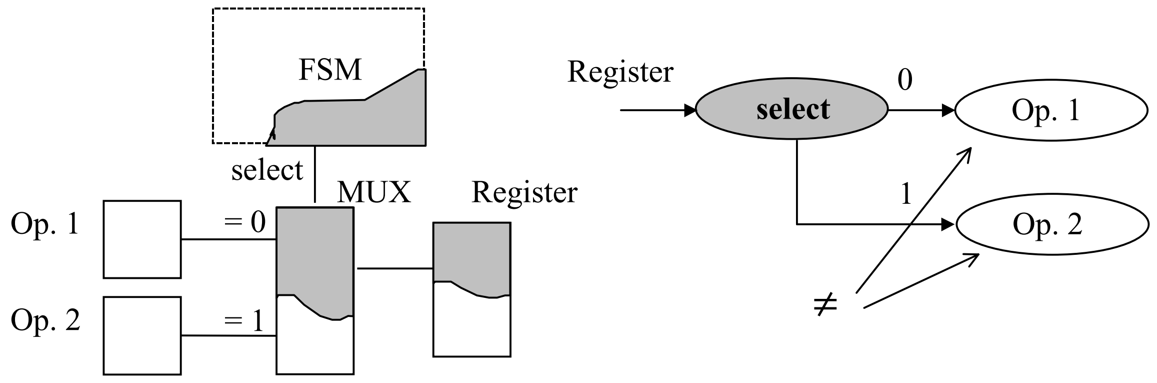


Figure 4. Conformity Test

Constraint Justification. The aim of constraint justification is to justify the variable values in the extracted constraints. Each time a backward step is made during the justification, the contents of the constraints will be updated. This is done according to the control vectors that are active at corresponding FSM states. Justification will end when all the variables in the constraints are primary inputs or constants.

The constraints can be divided into two categories: *path activation constraints* and *transformation constraints*. Path activation constraints correspond to the conditions that have to be satisfied in FSM in order to provide transparent paths through the circuit. Transformation constraints, in turn, reflect the value changes along the activated paths; They are extracted during the manifestation phase and represent the symbolic local test patterns for the module under test. Both types of constraints can be represented by common data structures and manipulated by common procedures for update, modeling and simulation.

Justification starts with traversing the propagation state sequence in the reverse order until the fault manifestation step is reached. During each time frame that is earlier than the manifestation step, the justification procedure selects a justification objective. In current implementation the objective is to backtrace the first unjustified variable in the transformation constraints. In the case when transformation constraints are justified, the objective will be to backtrace the first unjustified variable in the path activation constraints, respectively.

At every justification step, the constraints containing only constant variables will be simulated. This improvement to the test generation algorithm makes it possible to detect obvious inconsistencies at early stages of path activation and hence reduces the search space. At this point of the algorithm we have created the high-level symbolic *test frames*. In the following phases, actual values have to be calculated for the symbolic values of the frames.

Constraint Satisfaction. Subsequent to constraint justification, the constraints have to be solved. In order to achieve that, any known Constraint Satisfaction Problem (CSP) solving algorithm can be applied. In current implementation we use random generate-and-test technique. In the future, more advanced CSP methods have to be implemented to avoid possible loss of solutions while testing very large and complex circuits.

In some circuits, path activation constraints and transformation constraints can contain common variables. (For example, the GCD benchmark tested in current paper belongs to this class). This causes difficulties since local test patterns for the module under test (MUT) are dependent on path activation constraints. In these cases, a set of different solutions for the path activation constraints is generated. This set of solutions is applied to transformation constraints for simulation in order to derive local test patterns for MUT.

Low-Level Test. Only the path activation constraints are managed during constraint satisfaction while transformation constraints are considered in the low-level test. This step targets the gate-level structural faults in the modules under test (MUT). During the low-level test, random values are generated to the unassigned variables of transformation constraints. The constraints are simulated to obtain the transformed vectors at the inputs of MUT, which in turn are applied to the fault simulation for the module. If a fault is detected at the output of the module, it is assumed to be detected at the primary outputs of the whole device. This is true because the propagation of the fault effect symbol to primary outputs has been guaranteed by previous stages of the algorithm.

The vectors that detect previously undetected faults are compiled into final test vectors for the whole hierarchical circuit. This takes place by substituting the symbolic values in the high-level symbolic test frames by the actual values found during constraint satisfaction and low-level test.

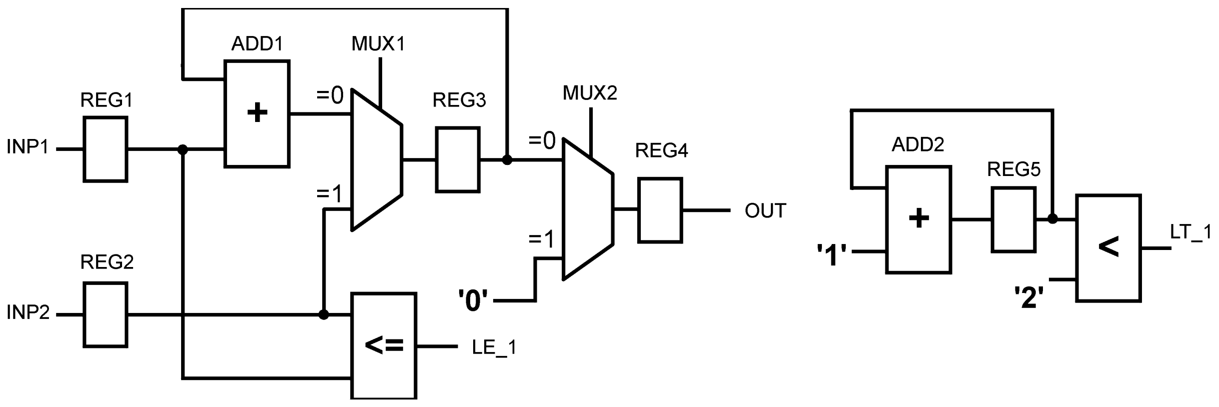
In the above algorithm description, transparency aspects were not included. Current test generator is open in terms of transparency rules and supports different levels of transparencies: I-paths [18], F-paths [15], etc. Required rules can be provided by the user. In the experiments presented in current paper, the tests were carried out by implementing transparencies based on the F-path method [15].

4 Test Generation Example

Let us consider the simple RTL example shown in Figure 5. It contains a small datapath of 5 registers and an FSM with 5 states. In the figure, the architectural description of the datapath and the state table of the FSM are given. Figure 6 presents the DD model synthesized from the RTL example. In the following we explain the DD-based test generation algorithm by performing a scanning test for the node REG1+REGR3 in DD REGR3 (Figure 6).

Fault Manifestation. By activating the path to the node under test in DD REGR3 we obtain the following values for control signals. (See the path marked with bold arrows in Figure 6).

```
REGR3_RESET := 0
REGR3_ENABLE := 1
MUX1_ADDRESS := 0
```



RESET	LE_1	LT_1	pres. state	next state	MUX1_ADDR.	MUX2_ADDR.	REG1_ENABLE	REG2_ENABLE	REGR3_EN.	REGR3_RESET	REG4_ENABLE	REGR5_EN.	REGR5_RESET
1	X	X	X	s0	X	1	1	1	0	1	1	0	1
0	0	X	s0	s1	X	X	0	0	0	0	0	1	0
0	1	X	s0	s2	X	X	0	0	0	0	0	1	0
0	X	X	s1	s3	1	X	0	0	1	0	0	0	0
0	X	X	s2	s3	0	X	0	0	1	0	0	0	0
0	X	0	s3	s4	X	X	0	0	0	0	0	0	0
0	X	1	s3	s0	X	X	0	0	0	0	0	0	0
0	X	X	s4	s0	X	0	0	0	0	0	1	0	0

Figure 5. An RTL Design Example

Subsequently we find the FSM DD terminal node whose control vector is consistent with these values. From the FSM DD in Figure 6 we see that the only consistent node is node 8. The vector labeling the node determines next state and current control signal values. By activating the path to node 8 in the FSM DD, we imply that current state is s2. In addition, we assign the fault effect value to REGR3 and symbolic local test pattern values to REG1 and REGR3.

Fault Effect Propagation. Propagation starts in state s3 (this is the next state value in FSM DD node 8, which was chosen in the manifestation step). Propagation objective is chosen from the set of nodes, which are labeled by the variable containing the fault effect. At present, the fault effect symbol is in variable REGR3, which labels the following nodes:

Node REGR3 in DD REG4,
node REGR3 in DD REGR3
and node REG1+REGR3 in DD REGR3.

The algorithm selects the first node (REGR3 in REG4) and activates the path to it in corresponding DD. From the activated path, the following control signal values are implied:

REG4_ENABLE := 1 and MUX2_ADDRESS := 0.

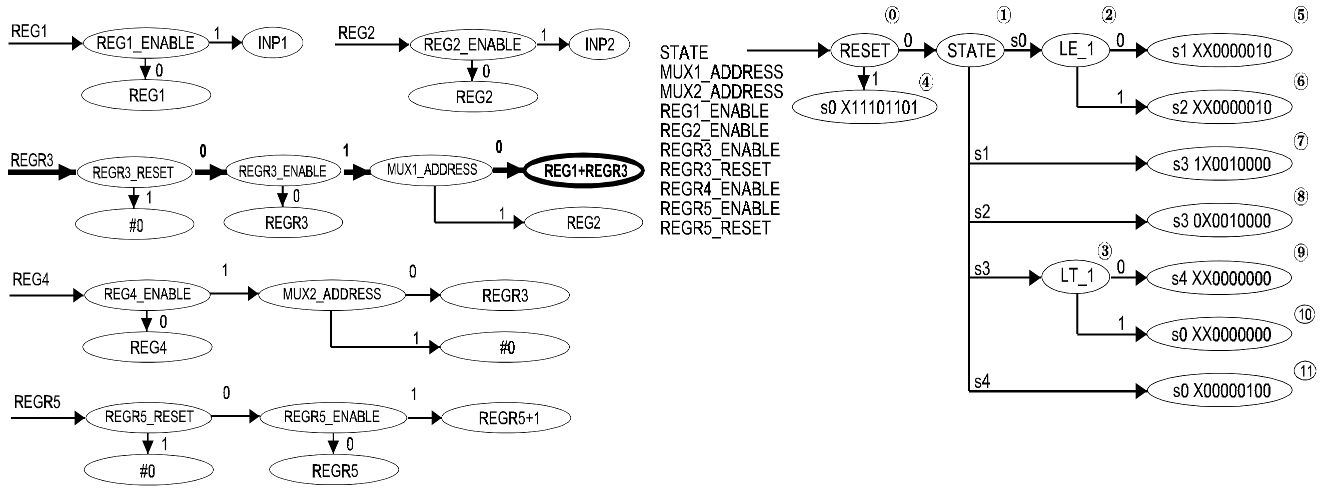


Figure 6. DD Models of the RTL Example

However, these values are inconsistent with the control vectors in FSM DD nodes 9 and 10, which are the terminal nodes that can be reached in the DD, when current state is equal to s3. Therefore, a backtrack occurs and a new propagation objective is selected, which is node REGR3 in DD REGR3. When we activate the path to the chosen node in DD REGR3, we will get:

REGR3_RESET := 0 and REGR3_ENABLE := 0.

These values are consistent with both, node 9 and node 10 in the FSM DD and we have to make a choice. Note that this is the situation, where node reordering can avoid the search algorithm from entering an eternal loop. Since selecting node 10 would create a loop in the FSM DD, the DD is ordered so that node 10 comes after 9 in the model. Hence, we make a choice and select the first one, i.e. node 9. By activating the path to the node in FSM DD we extract a constraint LT_1 = 0, which means that REG5 ≥ 2 (See Figure 5). This constraint is left to be justified by the subsequent justification procedure.

Let us move to the next clock cycle. Current state is now s4 (the next state value in node 9 of FSM DD). There is only one FSM DD terminal node (node 11) that can be reached from state s4. The control vector labeling this node sets REG4_ENABLE := 1 and MUX2_ADDRESS := 0. When we look at the DD for REG4, we will see that these values activate a path to REGR3, which means that symbolic fault effect propagates now to REG4 that is directly connected to a primary output. Thus, propagation is completed.

Constraint Justification. Justification starts with traversing the propagation state sequence in a reversed order until the manifestation step is reached. No decisions are made; only the contents of extracted constraints are updated, if necessary.

Subsequent to traversing the reversed propagation state sequence s4->s3->s2 the following constraints are extracted:

Transformation constraints:

REG1 := test_input1

REG2 := test_input2

(test_input1 and test_input2 represent symbolic local test patterns for the 1st and 2nd inputs of MUT, respectively).

Path activation constraints:

REG5 ≥ 2

Looking at the FSM DD we see that the state s2 can only be reached from FSM terminal node 6. Therefore we move to the previous clock cycle and activate the path to this node. By doing that, a new constraint LE_1 = 1 (i.e. REG2 ≤ REG1) is extracted and current state is implied to be s0. The FSM DD node 6 sets REGR5_ENABLE to the value 1, which means that the updated path activation constraints are now:

REG5 + 1 ≥ 2

REG2 ≤ REG1

Current state s0 can be reached from FSM DD terminal nodes number 4, 10 and 11. In current implementation the justification objective is to backtrace the first unjustified variable in the transformation constraints, i.e. REG1 in our example. Looking at the REG1 DD in Figure 6, it is easy to see that in order to propagate REG1 towards primary inputs, value '1' must be assigned to REG1_ENABLE. Only the control vector in the node 4 in FSM DD is consistent with this assignment. When we select FSM node 4, the updated constraints will be the following:

Transformation constraints:

INP1 := test_input1

INP2 := test_input2

Path activation constraints:

0 + 1 ≥ 2

INP2 ≤ INP1

However, the first of the path activation constraints is inconsistent and therefore a backtrack occurs. Without going to more details we say that the justification algorithm will subsequently activate the state sequence s0 (FSM DD node 10) -> s3 (node 8) -> s2 (node 6) -> s0 (node 4). The final justified constraints will be:

Transformation constraints:

INP1 := test_input1

INP2 := test_input2

Path activation constraints:

0 + 1 + 1 ≥ 2

INP2 ≤ INP1

0 + 1 < 2

INP2 ≤ INP1

Constraint Satisfaction and Low-Level Test. According to this path, the local test patterns to the adder ADD1 (Figure 5) are constrained to the ones where the values for INP1 must be less or equal than the values of INP2. If the fault coverage achieved by these patterns is not 100 %, the algorithm will try to activate additional symbolic paths.

5 Experimental Results

Experiments were carried out on two highly sequential circuits, a Greatest Common Divisor (GCD), which belongs to the HLSynth92 benchmark suite, and an 8-bit multiplier example. The synthesized RTL version of the GCD circuit contains a datapath with 5 registers and an FSM with 12 states. The circuit was chosen as it reveals a number of difficult test generation problems. It contains a global data dependent loop and only one of the registers is directly observable. The multiplier mult8x8 has a complex 16-bit datapath containing several feedback loops.

Test generation results were the following. For the GCD benchmark, all of the 9 scanning tests were successful. Two of the 10 conformity tests failed. For the multiplier example, all of the 12 conformity tests were successful and one out of eight scanning tests failed. Actual quality of the generated test sequences was measured by applying gate-level fault simulation to the circuits and by neglecting a set of obviously untestable faults (e.g. lines tied to constants). The achieved fault coverages for datapath parts were 95.1 % for the GCD circuit and 95.9 % for mult8x8, respectively. Although control part faults were not explicitly targeted, fault coverages measured for control parts were high. Achieved test generation times were short. Both of the circuits were tested in less than 20 seconds. The number of test sequences was less than 100. Due to the fact that sequential circuits were considered, each test sequence consisted of multiple clock cycles. Table 1 presents the experimental results, which were run on a 233 MHz Pentium II computer with 64 MB RAM under Windows 95 operating system.

Circuit	gcd	mult8x8
Number of gate-level faults	1066	4432
Gate-level fault coverage DP (%)	95.1	95.9
Fault coverage CP (%)	89.4	92.1
Total gate-level fault coverage (%)	91.8	94.4
Test generation time (s)	14.6	17.7
Number of generated test sequences	53	93
Total test length (number of clock cycles)	627	2797

Table 1. Test Generation Results

In Table 2, comparative results with a gate-level sequential test pattern generator HITEC [5], a genetic test pattern generator GATEST [7] and a novel hierarchical test pattern generation approach published in [8] are given. The comparison is carried out on the example of the GCD circuit, which is the only circuit common with the experiments in [8]. As we can see from the table, the proposed DD-based technique outperforms the other test generation tools in all categories. It achieves a higher fault coverage in a much shorter time and generates less test sequences than [8]. The number of test sequences for [5] and [7] is not known.

	DD	Hier. [8]	GATEST [7]	HITEC [5]
fault coverage, %	91.8	90.4	62.6	74.4
time, s	14.6	1068	636	49320
test sequences	53	60	N.A.	N.A.

Table 2. Comparative Results

6 Conclusions and Future Work

Current paper describes a novel hierarchical test generation approach based on decision diagram models. Differently from known methods, both, higher and lower design abstraction levels, and both, control and data paths are handled here by a uniform approach. Joint formal basis for gate- and higher level descriptions allows to adopt and generalize gate-level simulation and ATPG methods and tools to higher level ones. In addition, different types of datapath components (registers, FUs, multiplexers) are handled in a uniform manner. Simple testability measures are included to the DD representations in order to avoid the algorithm from being stuck to eternal loops.

Comparison with the known state-of-the-art test generators for sequential circuits show the advantages of the DD-based technique. Experiments on the Greatest Common

Divisor circuit indicates that the proposed approach achieves a significantly higher fault coverage with a speed 44 – 3400 times higher than in [5, 7, and 8].

The ATPG in the present implementation does not include fault effect propagation along multiple paths simultaneously. In addition, the constraint satisfaction procedure should be improved by implementing more sophisticated methods. We intend to overcome these shortcomings in the near future.

References

[1] M.H.Schulz, E.Auth, "Improved Deterministic Test Pattern Generation with Application to Redundancy Identification", *IEEE Trans. on CAD*, Vol. 8, No. 7, pp. 811-816, 1989.

[2] J. A. Waicukauski et al., "ATPG for Ultra-Large Structured Designs", *Proc. of the International Test Conference*, pp. 44-51, 1990.

[3] C. Cho, J. Armstrong, "B-algorithm: A Behavioral Test Generation Algorithm", *IEEE International Test Conference,* pp.968-979, 1997.

[4] J. Lee and J. H. Patel, "Architectural level test generation for microprocessors", *IEEE Trans. Computer-Aided Design*, vol. 13, no. 10, pp. 1288-1300, Oct. 1994.

[5] T. M. Niermann, J. H. Patel, "HITEC: A test generation package for sequential circuits", *Proc. of the European Conf. Design Automation (EDAC)*, pp.214-218, 1991.

[6] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits", *IEEE Trans. Computer-Aided Design*, vol. 15, no. 8, pp. 991-1000, Aug. 1996.

[7] E. M. Rudnick, J. H. Patel, G. S. Greenstein, T. M. Niermann, "Sequential circuit test generation in a genetic algorithm framework", *Proc. of the Design Automation Conf.*, pp. 698-704, 1994.

[8] E. M. Rudnick, R. Vietti, A. Ellis, F. Corno, P.Prinetto, M. Sonza Reorda, "Fast sequential circuit test generation using high-level and gate-level techniques", *Proc. of the DATE Conf.*, 1998.

[9] R. Ubar, "Test Generation for Digital Circuits Using Alternative Graphs", *Proc. of Tallinn Technical University, Estonia*, No. 409, pp. 75-81 (in Russian), 1976.

[10]R. Ubar, "Test Synthesis with Alternative Graphs", *IEEE Design & Test of Computers*, pp. 48-57, Spring 1996.

[11]F. Corno, U. Gläser, P. Prinetto, M. Sonza Reorda,H. T. Vierhaus, "Improving Topological ATPG with Symbolic Techniques", *IEEE VLSI Test Symposium*, Princeton, USA, pp.338-343, April 1995.

[12]R. Ubar, "Test pattern generation for digital systems on the vector AG-model", *13th International symposium on Fault Tolerant Computing,* Milano, Italy, pp.347-351, 1983.

[13]R. Ubar, M. Brik, "Multi-Level Test Generation and Fault Diagnosis for Finite State Machines", *Lecture Notes in Computer Science 1150, Dependable Computing - EDCC-2.* Springer Verlag, pp.264-281, 1996.

[14]R. Ubar, "Beschreibung digitaler Einrichtungen mit AG für die Fehlerdiagnose", Nachrichtentechnik / Elektronik, 30, H.3, pp. 96-102, 1980.

[15]S. Freeman, "Test Generation for Data Path Logic: The F-Path Method", *IEEE J. of Solid-State Circuits*, Vol.23, pp.421-427, April 1988.

[16]J. Lee, J. H. Patel, "Hierarchical test generation under intensive global functional constraints", *Proc.29th ACM/IEEE Design Automation Conf.*, pp. 261-266, June 1992.

[17]S. B. Akers, "Binary Decision Diagrams", *IEEE Trans. on Comp.*, Vol.27, pp.509-516, 1978.

[18]M. S. Abadir, M. A. Breuer, "A Knowledge-Based System for Designing Testable VLSI Chips", *IEEE Design & Test*, pp.56-68, August 1985.