Sequential Circuit Test Generation Using Decision Diagram Models

Jaan Raik, Raimund Ubar
Department of Computer Engineering
Tallinn Technical University, Estonia

Abstract

A novel approach to testing sequential circuits that uses multi-level decision diagram representations is introduced. The proposed algorithm consists of a combination of scanning and conformity test generation procedures. Structural faults in both, datapath and control part are targeted. High-level simplified and fast symbolic path activation strategy is combined with random local test pattern generation for functional units. Current approach has achieved high fault coverages for known sequential circuit benchmarks in a very short time.

1. Introduction

Different techniques for solving the problem of generating tests for structural faults in sequential circuits have been proposed over the years. On the gate-level, deterministic [1] and simulation based [2,3] algorithms have been proposed. However, the execution times are extremely long and for medium and large circuits mostly rather low fault coverages have been achieved.

Recently, promising results based on software testing techniques combined with low level test have been published in [4]. The approach offers high fault coverages for medium sized benchmark circuits but the test generation still takes relatively much time. Furthermore, the authors have not developed any formalized method for generating the high-level test frames and the time needed to generate the frames has not been taken into account in the experiments. Trivial finite state machines containing only a single control state have been implemented for some of the larger example circuits.

At present, hierarchical test generation is the fastest and most effective means for sequential circuits testing [5]. Here, designs described on different abstraction levels, usually on architectural- and gate-level, are used. The method cannot be applied to designs that do not have appropriate modularity, or where gate-level implementation for the modules is not known. However, as a number of commercial high-level synthesis tools have emerged, the input description should not be a major issue.

Previous works in the area of hierarchical testing have the following main shortcomings:

1. Only the faults in the datapath Functional Units (FU) are targeted.

This usually results in low fault coverages for the control part as well as for multiplexers, registers and fanout buses of the datapath.

2. A complex set of symbols and constraints is used during high-level path activation.

This feature makes the symbolic path activation process very compute-intensive. For more complex circuits it can also cause high-level tests for many FUs to fail due to the strict conditions.

The aim of the approach proposed in current paper is to overcome the above mentioned shortcomings. Differently from known methods, both, control unit and datapath are handled in a uniform manner. A restricted set of symbols is used during the path activation. This allows to simplify the test generation algorithm while still maintaining a good correspondence between high-level assessments and gate-level fault coverage. The paper is organized as follows. Section 2 gives a short overview of representing circuit architecture by Decision Diagram (DD) models. Section 3 introduces the test generation algorithm. Finally, experimental results and conclusions are presented.

2. Decision Diagram representations

Consider a component (subnetwork) f of a digital sytem S as a function $y=f(x)$ where $y=(y_1,\dots y_n)$ and $x=(x_1,\dots x_m)$ are vector variables. The function f is defined on $X=X_1\times\dots\times X_m$ with values $y \in Y = Y_1\times\dots\times Y_n$, and both, the domain X and the range Y are finite sets of values. x_i, $i = 1,2,\dots m$, are input or state variables of the component f, whereas y_j , $j = 1,2,\dots n$, are output or next state variables. The values of variables may be Boolean, Boolean vectors, integers.

Definition 1. A Decision Diagram (DD) G is a directed noncyclic graph $G=(M,\Gamma,x)$. M is a set of nodes. Γ is a relation on M where $\Gamma(m) \subset M$ denotes the set of successor nodes of $m\in M$. The nodes $m\in M$ are marked by labels $x(m)$. The labels can be: variables x_i, algebraic expressions

of x_i, or constants. For nonterminal nodes m, where $\Gamma(m) \neq \varnothing$, an onto function exists between the values of $x(m)$ and the successors $m^e \in \Gamma(m)$ of m. By m^e we denote the successor of m for the value $x(m)=e$.

Definition 2. The edge (m, m^e) which connects nodes m and m^e is called activated iff there exists an assignment $x(m)=e$. Activated edges which connect m_i and m_j make up an activated path $l(m_i,m_j)$. An activated path $l(m^0,m^T)$ from the initial node m^0 to a terminal node m^T is called full activated path.

Definition 3. Decision Diagram $G_y = (M,\Gamma,x)$ represents a function $y = f(x)$ iff for each value of x, a full path in G_y to a terminal node m^T is activated, where $x(m^T) = y$ is valid.

As a hierarchical input to the test generator are descriptions where the architecture of the circuit is described at the Register-Transfer Level (RTL) and the low-level structure is given at the gate level. Both these levels can be described by DD models. Datapath can be represented by a system of DDs, where for each primary output and register, a DD corresponds. In addition, multiplexers that are connected to an input of an FU are represented by a separate DD. 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.

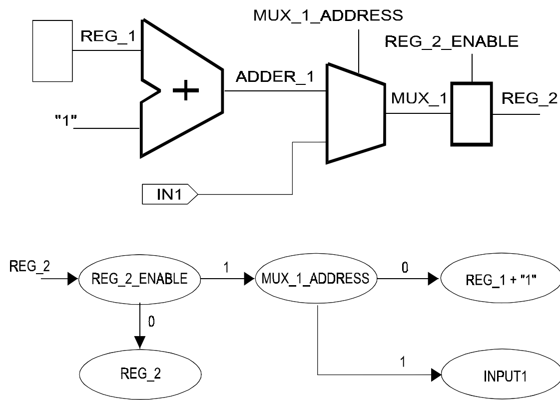

Figure 1. DD model of a datapath fragment

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.

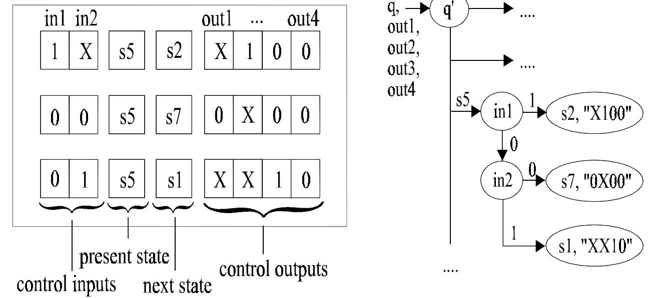

Figure 2. Converting a state table to a DD

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. More detailed information about SSBDDs can be found in [6].

3. Test generation algorithm

The high-level symbolic 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 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. For the sake of simplicity and speed, only three types of symbolic values are used during the path activation:

D	- line with the fault effect,
X	- line with unassigned value,
assigned	- line with a specified (integer) value.

The hierarchical test generation algorithm consists of five stages. These are fault manifestation, fault propagation, constraint justification, constraint satisfaction and low-level test, respectively. In the following, the different stages are explained more in detail.

3.1. Fault manifestation

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 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, the path to the node under test is activated in respective DD. The symbolic fault-effect value D is assigned to the variable corresponding to the DD, and new constraints are created from the arguments of the function labeling the node under test. These constraints are later treated as justification objectives. Figure 3 presents a simple example where scanning test is performed for the node FU1 in the DD MUX. The blocks where gate-level faults are targeted by the scanning test are marked with striped areas in the figure.

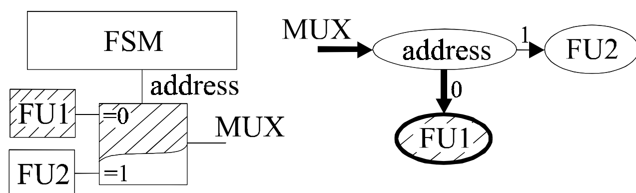

Figure 3. Example of scanning test

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, distinguishing of values of the variables labeling the terminal nodes is made. In current implementation pairwise distinguishing is used. Conformity test for a node must be carried out for each edge of the node under test activated and for each pair to be distinguished. Hence, there exist n(n-1) conformity tests for a non-terminal node with n successor nodes.

The distinguishing takes place as follows. In case the successor nodes are not terminals, paths are activated from the successors to terminal nodes. Constraints to be back-traced during the justification are created of the variables labeling corresponding terminal nodes of the DD. A simple example in Figure 4 illustrates the conformity test for the control signal address. The blocks where faults are targeted by the test are marked with striped areas.

3.2. Fault effect propagation

The aim of the propagation procedure is to determine the state sequence necessary to propagate the fault effect

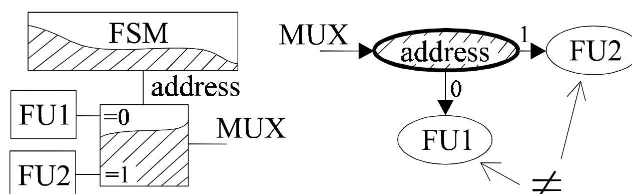

Figure 4. Example of conformity test

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 alternate choices at datapath and control part DDs. Activated paths in DDs make it possible to determine relevant variable assignments at each time step.

3.3. Constraint justification

During this phase we 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. It is done according to the control vectors that are active at corresponding FSM states. In addition, new constraints will be extracted if conditions are traversed in the FSM DD. 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 are necessary in order to calculate the 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.

3.4. 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 large and complex circuits.

3.5. 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.

4. Experimental results

The proposed test generation algorithm has been implemented as a part of the DECIDER (DECision Diagram based test genERation) system [8]. At present, the system contains gate-level EDIF interface which is capable of reading designs of CAD systems like SYNOPSYS, CADENCE, MENTOR GRAPHICS, VIEWLOGIC, etc. In addition, an RT-level VHDL interface to a commercial high-level synthesis tool is under consideration.

Table 1 presents the main characteristics of the benchmark circuits used in the experiments. The two designs are well-known benchmarks from the *HLSynth* family. In Table 2 the results which were obtained on a SUN ULTRASPARC 2 workstation under Solaris 2.5 operating system are given. Fault coverages were determined by applying gate-level fault simulation to the generated patterns, i.e. actual stuck-at fault coverages are reported in the table.

For comparison, experimental data of [4] has been included. Table 3 describes the circuits used for the experiments in [4]. These experiments were run on an HP 9000 J200 256MB computer and the results are given in Table 4.

Though the circuits in Table 1 and Table 3 represent the same functionality, instead of trivial control units containing only a single control state, in current paper, control units with multiple states are implemented. This leads to a more complex class of devices with high sequential depth and global feedback loops over control and datapath parts. However, in our experiments a 4-bit version of the gcd circuit was used while in [4] the bit-width was 16.

Compared to the results obtained in [1,3,4] test generation times achieved by current method were significantly shorter. The fault coverage for *diffeq* was slightly lower which can be explained by the higher sequential depth of the implementation. The number of generated test sequences in our approach can be further minimized if high-level fault collapsing would be included.

5. Conclusions

Current paper describes a novel hierarchical test generation approach based on using decision diagram models. Differently from known methods, both, higher and lower design abstraction levels, and both, control and data parts are handled in a uniform manner. Joint formal basis for gate- and higher level descriptions allowed to adopt and generalize gate-level methods to high-level ones. The methods were improved by exploiting higher bitwidth of data variables. The new path activation technique avoids complex symbolic algebra while maintaining still a good correspondence between high-level fault coverage assessments and actual gate-level fault coverage. As a result, high fault coverages are very quickly achieved for known sequential circuit benchmarks.

Authors consider several additional improvements that could be made to the proposed algorithm. The present implementation does not include fault effect propagation

Table 1. Benchmark circuits

Circuit	Gates	Faults	PIs	POs	Flip-flops	Control states
diffeq	4195	15,836	81	48	115	6
gcd	227	844	9	4	15	8

Table 2. Test generation results

Circuit	Fault coverage, %	Test sequences	Time, s
diffeq	95.4	353	20.4
gcd	91.0	67	5.6

Table 3. Benchmarks used in [4]

Circuit	Gates	Faults	PIs	POs	Flip-flops	Control states
diffeq	9340	18,216	81	48	129	1
gcd	1191	2199	33	16	49	1

Table 4. Test generation results in [4]

Tool	Circuit	Coverage, %	Sequences	Time, s
High+gate [4]	diffeq	98.2	100	6480
	Gcd	90.4	90	1068
HITEC [1]	Diffeq	97.3	N. A.	84960
	Gcd	74.3	N. A.	49320
GATEST [3]	Diffeq	98.9	N. A.	27756
	Gcd	62.6	N. A.	636

simultaneously along multiple paths. In addition, the constraint satisfaction procedure could be enhanced by implementing more sophisticated methods. This remains the subject of our future research.

Acknowledgement

This work was supported by Estonian Science Foundation Grant G-1850 and by German-Estonian bilateral project EST-008-96 funded by BMFT Germany.

References

[1] T. M. Niermann, J. H. Patel, "HITEC: A test generation package for sequential circuits", *Proc. of the EDAC*, pp.214-218, 1991.

[2] 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 Tran. CAD*, vol.15, no.8, pp.991-1000, Aug. 1996.

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

[4] 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 DATE*, 1998.

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

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

[7] R.Ubar, J.Raik, "Hierarchical test generation for digital systems based on combining bottom-up and top-down approaches", *Proc. of SCI/ISAS'98*, pp.374-381, Orlando, July 1998.

[8] G.Jervan, A.Markus, J.Raik, R.Ubar, "DECIDER: A Decision Diagram based Hierarchical Test Generation System", Proc. of the DDECS'98 Conference, pp. 269-273, Szczyrk, Poland, September 2-4, 1998.