High-Level Test Synthesis with Hierarchical Test Generation

Gert Jervan[1], Petru Eles[1], Zebo Peng[1], Jaan Raik[2], Raimund Ubar[2],

[1] Dept. of Computer and Information Science, Linköping University, Sweden
[2] Dept. of Computer Engineering, Tallinn Technical University, Estonia

Abstract: A novel full design and test generation system with combined High-Level Synthesis (HLS) and automated Hierarchical Test Pattern Generation (HTPG) was developed and experimented. The high-level synthesis is based on representation model called Extended Timed Petri Net (ETPN). In the test generator both register-transfer (RT) and gate-level descriptions are used, and Decision diagrams (DD) are exploited as a uniform model for describing systems at both levels. In addition to the synthesis and test generator tools, interfaces to behavioral and RT-level VHDL and EDIF netlist formats have been implemented. In the paper, an overview of the implementation is given and experimental data showing the viability of the approach and the efficiency of respective tools are provided.

1. Introduction

Recently, more-and-more high-level synthesis tools have become available. These tools are used by designers to automatically generate Register-Transfer Level (RTL) descriptions from a behavioral description. The HLS tools take into account several constraints, e.g. speed, area, or testability, and allow the designer to quickly compare the trade-offs between alternative RTL implementations. The usual disadvantage of existing HLS tools is that they miss automated testability analysis and test generation capabilities. The test synthesis system presented in this paper was implemented to address the above shortcoming.

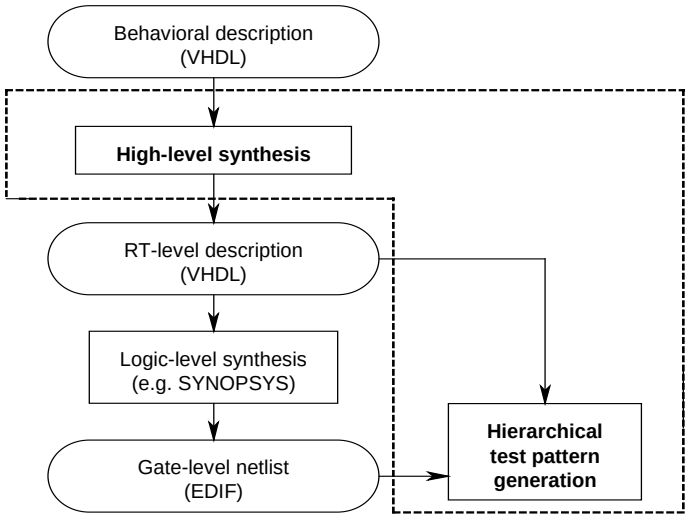

Fig. 1. Test synthesis flow

Fig. 1 presents the test synthesis flow used in the proposed approach. The system described in this paper is denoted by a surrounding dashed line. It includes the high-level synthesis tool CAMAD [1] (created at Linköping University), a hierarchical test pattern generation DECIDER [2] (implemented at Tallinn Technical University) and appropriate interfaces from VHDL and EDIF formats. In the following, different parts of the system are described in detail.

2. High-Level Synthesis with CAMAD

The CAMAD high-level synthesis system is built around an internal design representation, called ETPN (Extended Timed Petri Net), which has been developed to capture the intermediate results during the high-level synthesis process. The use of Petri nets provides a natural description of concurrency and synchronization of the operations and processes of a hardware system. It gives thus a natural platform to represent and manipulate concurrent processes of VHDL specifications.

ETPN is a *formal* representation model derived from Petri net theory and consisting of separate but related models for control and data path. The *datapath* is represented as a directed graph with nodes and arcs. The nodes are used to capture data manipulation units (data storages, arithmetic operators, etc.). The arcs represent the connections of the nodes. The *control part* of ETPN, on the other hand, is captured as a timed Petri net with restricted transition firing rules. These two parts are related by the control signals coming from the control part to the datapath, and the conditional signals traveling in the opposite direction.

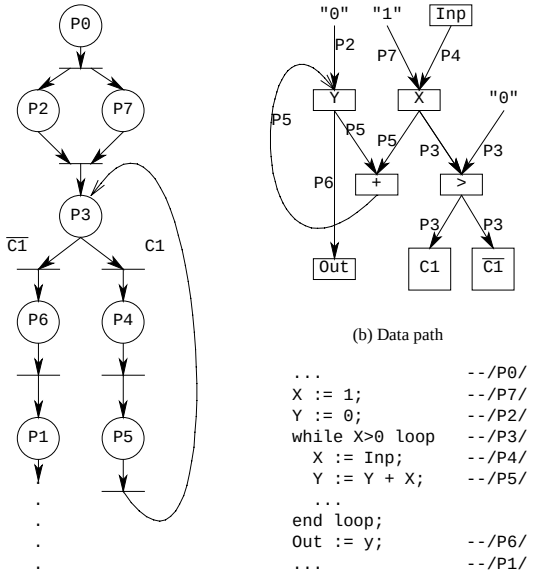

Fig. 2. An example of ETPN representation and its corresponding VHDL description

A simple example of ETPN is illustrated in Fig. 2. Fig. 2(b) shows the datapath where each datapath node is depicted as a rectangle with a label indicating the basic operation or storage identifier of the node. The arcs of the data path represent the data flow between the nodes. Flow of data from one node to another is controlled by the control signals coming from the control part. The control relation is indicated by using control place names to guard the arcs. When a control place in the Petri net holds a token, its associated arcs in the data path (arcs guarded by the corresponding label) will be open for data to flow. Fig. 2(a) depicts the Petri net which represents the control flow of the example. A control state is represented as a marking of the Petri net, i.e., the possession of tokens in a subset of the places of the Petri net which are depicted as circles. The changes of control states are carried out as firings of one or several transitions of the Petri net which are depicted as bars. To express that the control flow can be guarded by results of internal computations, we use conditional signals to guard the control flow. A transition may be guarded by one or several conditions produced from the datapath. A transition may be fired when it is enabled (all its input places have a token) and the guarding condition is true. If a transition has more than one guarding condition and at least one of them is true, the transition's guarding condition is true. The ETPN example is generated by CAMAD as the compilation output of the input VHDL specification given in Fig 2(c).

The main feature of the ETPN design representation is its ability to capture the intermediate result of a design explicitly so as to allow the design algorithm to make accurate design decisions. For example, if several operations are not data-dependent and can thus be executed concurrently, the situation can be captured precisely by giving their associated control places in the Petri net a potential to hold tokens simultaneously. That is, the set of Petri net places corresponding to the operations will not have any partial ordering relation between them. For example, in Fig. 2, P_2 and P_7 control the loading of data to register *X* and *Y* respectively, which are independent operations. Therefore, P_2 and P_7 can hold tokens simultaneously. When it is, however, discovered later during synthesis that the potential parallelism will not be able to be implemented (because of resource restrictions for example), additional partial ordering relations can be introduced by performing a place-stretch transformation.

ETPN is used as a *unified* design representation which captures the intermediate designs of the high-level synthesis process, and thus allows the synthesis algorithm to employ an iterative improvement approach to carry out the synthesis task. The basic idea is that once the VHDL specification is translated into the initial design representation, it can be viewed as a primitive implementation. Correctness-preserving *transformations* can then be used to successively transform the initial design into an efficient implementation. CAMAD *integrates* the operation scheduling, data path allocation, control allocation and, to some degree, module binding sub- tasks of high-level synthesis. This is achieved by developing a set of basic transformations of the design representation which deals *simultaneously* with partial scheduling and local data path/control allocation. An optimization algorithm is then used to analyze the (global) design and select transformations during each step of the iterative improvement process.

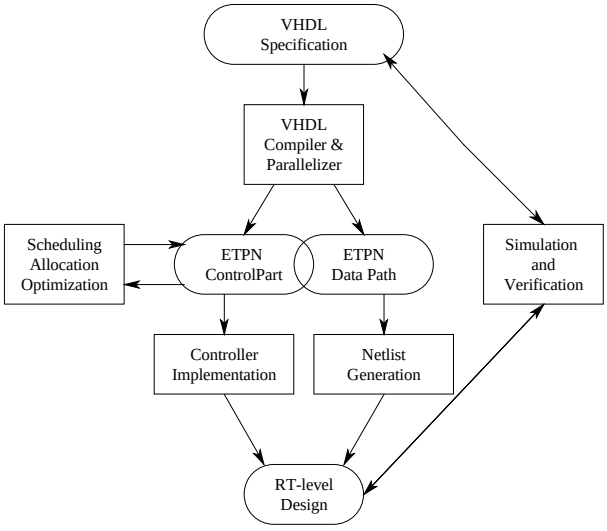

Fig. 3. Overview of CAMAD

Fig. 3 illustrates the basic structure of the CAMAD system. The first step of CAMAD is to map the VHDL specification into ETPN and to perform automatic parallelism extraction. After the transformation steps a RTL hardware implementation is generated which consists of a data path net-list and a controller specified in the form of a finite state machine. The final RTL implementation can be converted into structural VHDL which, as well as the input system specification, can be simulated for verification.

3. DECIDER: Automatic Test Pattern Generation for Sequential Circuits

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 means of Decision Diagram (DD) models [3]. At the RT-level, the design is partitioned into a datapath and a control part. At present, we assume that all the functional units are combinational and that single-clocked circuits are considered.

Datapath can be represented by a system of DDs, where for each primary output, fanout signal and register a DD corresponds. In addition, multiplexers that are connected to an input of an FU are represented by a separate DD. In datapath DD models, the non-terminal nodes correspond to control signals (e.g. multiplexer addresses and register enable signals) and terminal nodes represent operations. Register transfers and constant assignments are treated as special cases of operations. Values of the control signals labeling the nonterminal nodes of the DD activate the main path that determines, which operation is assigned to the variable whose value is calculated by that DD. Figure 4 shows an example of a DD representation for a datapath register.

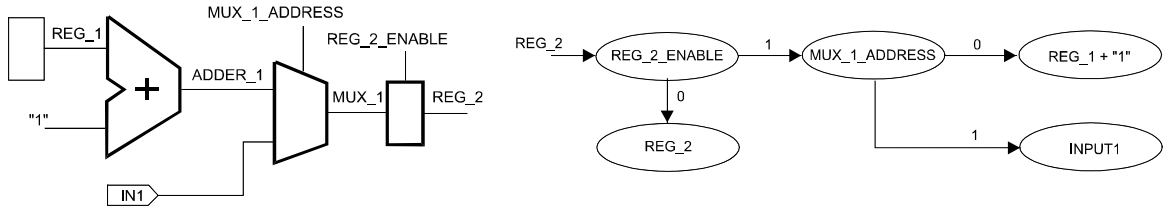

Fig. 4. DD model of a datapath fragment.

Control part of an RTL is represented by a DD that calculates the values for a vector consisting of the state variable and control signals. In the DD, 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 5 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 5 describes the behavior of the FSM when the current state is equal to s5.

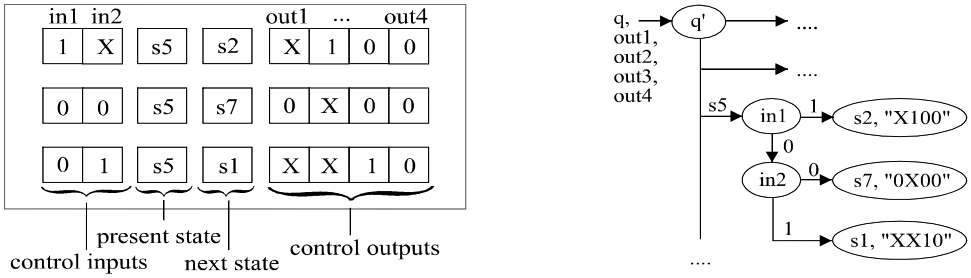

Fig. 5. Converting a state table to a DD.

In current approach, gate-level descriptions of Functional Units (FU) are transformed into Structurally Synthesized BDD (SSBDD) models [3, 4]. Differently from traditional BDDs [5], which represent function only, SSBDDs support test generation for gate-level structural faults in terms of signal paths without representing these faults explicitly. Furthermore, the worst case complexity and memory requirements for generating SSBDD models for FFRs are linear in respect to the number of logic gates in the circuit, while for traditional BDDs the total storage space exceeds 2^n bits for an n-input combinational circuit [6]. Hence, SSBDDs for an arbitrary realistic-sized digital circuit can be generated very rapidly.

In Fig. 6, the structure of the DECIDER test pattern generation system is presented. The test flow takes place as follows. From the CAMAD high-level synthesis, a VHDL RT-Level (RTL) description of the design is obtained. The VHDL description is converted to DD models that are used as a high-level input representation for the test generation. The RTL VHDL description and the library describing the behavior of the FUs is applied to logic synthesis. (At present, we use Design Compiler

from Synopsys Inc. for logic synthesis). As the result of the synthesis we obtain gate-level EDIF netlists of the entire flattened design as well as for each FU separately. Subsequently, these netlists are converted into Structurally Synthesized BDD (SSBDD) models by the EDIF interface of the DECIDER system. SSBDD models provide the logic-level input description for the hierarchical test pattern generator.

The Automatic Test Pattern Generator (ATPG) of the system uses the following approach. Tests are generated for each FU of the system separately. Symbolic path for propagating fault through the network of components is activated and corresponding constraints are extracted at the high level. These constraints are reduced to inputs, input values for the FU are calculated and gate-level fault simulation is applied to the FU using these values. Subsequently, final test patterns for the whole circuit are compiled.

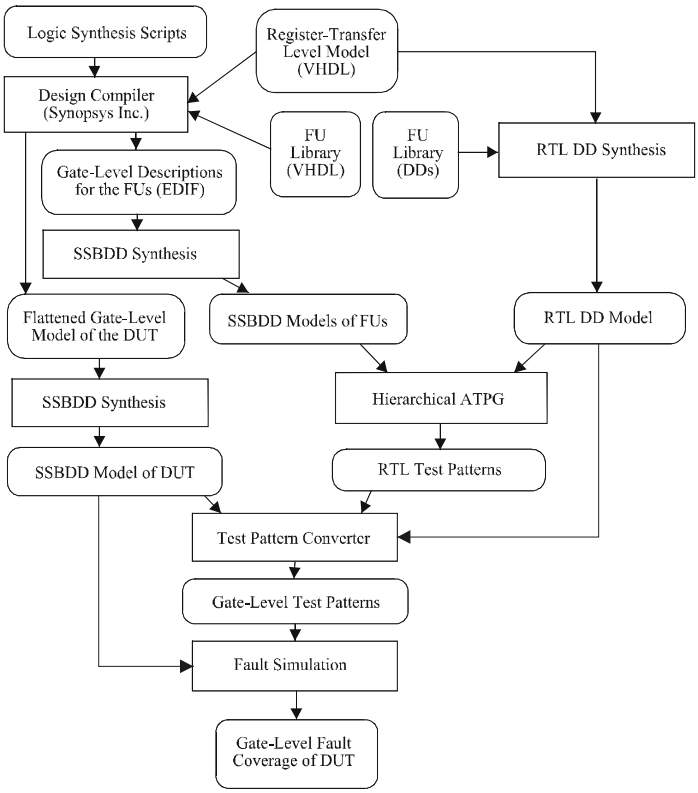

Fig. 6. Data flow of DECIDER

These test patterns are the input stimuli for the RTL design. However, as the test generation implements high-level fault models, we do not know the precise gate-level stuck-at fault coverage of the tests. Therefore, the test patterns have to be converted to correspond to the stimuli for the gate-level netlist of the entire design. This is required for gate-level stuck at fault simulation of the design in order to measure the quality of generated tests.

4. Experimental Data

Table 1 shows comparison of DECIDER with two other state-of-the-art ATPG tools. These are GATEST [7] and HITEC [8], respectively. GATEST is a genetic algorithm based test generator. HITEC is a deterministic gate-level ATPG. Both of the tools have been developed at the University of Illinois at Urbana-Champaign. The experiments were run on a 300 MHz SUN UltraSPARC 10 workstation with 128 MB RAM under SOLARIS 2.6 operating system. Actual stuck-at fault coverages of the test patterns generated by all the three tools were measured by the fault simulator from TURBO TESTER tools [9]. As it can be seen from Table 1, HITEC offers the poorest performance, both, in terms of fault coverage and test generation time on all the three example circuits. Fault coverages achieved by GATEST and DECIDER are almost equal. GATEST reaches 1.2 % higher fault coverage for the *gcd* circuit and only 0.2 % higher coverage for *diffeq*. DECIDER in turn has a 2.1 % advantage in the case of *mult8x8* example. However, the test generation times of the two tools differ greatly. DECIDER spends 26 - 615 times less CPU time for test pattern generation process than GATEST.

Table 1. Test generation results.

	DECIDER		**GATEST [7]**		**HITEC [8]**	
	Fault cover, %	**Time, s**	**Fault cover, %**	**Time, s**	**Fault cover, %**	**Time, s**
gcd	91.0	**3.4**	**92.2**	89.8	89.3	195.6
mult8x8	**79.4**	**13.6**	77.3	1585	63.5	1793
diffeq	95.8	**15.8**	**96.0**	9,720	95.1	N. A.

5. Conclusions

A novel high-level synthesis system with integrated hierarchical ATPG is presented. The high-level synthesis system CAMAD is built around the ETPN representation. The use of Petri nets provides a natural description of concurrency and synchronization of the operations and processes of a hardware system. It provides a natural platform to represent and manipulate concurrent processes of VHDL specifications. In the ATPG, differently from known methods of test generation, both higher and lower abstraction levels, and both control and datapath parts are handled by a uniform approach. This joint formal basis allowed to adopt and generalize gate-level simulation and test generation methods and tools to higher level ones. Appropriate interfaces to behavioral and RT-level VHDL and EDIF netlist formats have been implemented. In the paper, an overview of the implementation is given and experimental data showing the viability of the approach and the efficiency of test generation are provided.

References

1. Z. Peng and K. Kuchcinski: Automated Transformation of Algorithms into Register-Transfer Level Implementations, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, 150-166, 1994.
2. J.Raik, R.Ubar, "Sequential Circuit Test Generation Using Decision Diagram Models", Proc. of DATE Conf., 1999.
3. R. Ubar, "Test Synthesis with Alternative Graphs", IEEE Design & Test of Comp., Spring 1996.
4. R. Ubar, "Test Generation for Digital Circuits Using Alternative Graphs", Proc. of Tallinn Technical University, Estonia, No. 409, pp. 75-81 (in Russian), 1976.
5. S.B.Akers, "Binary Decision Diagrams", IEEE Trans. Computers, Vol. 27, pp.509-516, June 1978.
6. H.-T. Liaw, C.-S. Lin, "On the OBDD-Representation of General Boolean Functions", IEEE Trans. Computers, Vol. 41, pp. 661-664, June 1992.
7. T. M. Niermann, J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits", Proc. European Conf. Design Automation, pp. 214-218, 1991.
8. E. M. Rudnick, J. H. Patel, G. S. Greenstein, T. M. Niermann, "Sequential Circuit Test Generation in a Genetic Algorithm framework", Proc. Design Automation Conf., pp. 698-704, 1994.
9. G.Jervan et al., "Turbo Tester: A CAD System for Teaching Digital Test", Microelectronics Education, Kluwer Academic Publishers, Dordrecht/Boston/London, pp.287-290, 1998.