The Use of Hierarchical Information to Test Large Controllers

F.Fummi D.Sciuto
Dip. di Elettronica e Informazione
Politecnico di Milano, 20133 Milano, ITALY

Abstract – Gate-level test pattern generators require insertion of scan paths to handle the flat gate-level representation of a large sequential controller. In contrast, we present a testing methodology based on the hierarchical finite state machine model. Such a model is used to specify very complex control devices by means of a top-down design approach. Our approach allows the generation of compact test sets with very high stuck-at fault coverages, without any DfT logic.

I. Introduction

This paper describes a strategy for generating test sequences starting from a functional description of a sequential circuit without re-extraction of a STG from the implementation. In particular, we target descriptions realized either by writing the specifications directly in a Hardware Description Language (HDL), such as VHDL or Verilog, or by means of graphical tools that enable a user to enter hierarchical descriptions, such as Statecharts [6], that are then converted into HDL descriptions. Since functional information alone is not sufficient to achieve a complete fault coverage [1], we propose an integration with some structural information, obtained after logic synthesis. This test strategy mixes the accuracy of a gate-level test pattern generator with the efficiency of a functional one, by exploiting the available information on the design.

The fault list is generated for the actual gate-level implementation, considering the stuck-at fault model on the combinational part of the circuit, while the fault propagation and justification phases are performed on the hierarchical Finite State Machine model representing the specification.

The paper is organized as follows. In Section II testability relations are presented between the structural description of a sequential circuit and its functional representation described by a *specification* STG (SSTG). In Section III the hierarchical FSM model (HFSM) is summarized: it allows the description of highly complex controllers and the generation of the specification STG without explicitly enumerating all the states of an HFSM. Section IV describes the proposed testing methodology based on the identification of the set of transitions of the SSTG that are able to detect all stuck-at faults in the combinational part of the FSM implementation and the strategy to build the global test sequence by concatenating such transitions. Finally, Section V presents experimental results on academic and industrial benchmarks.

II. Functional/Structural Testability

A FSM can be represented by its state transition graph (STG), where nodes represent states and edges represent transitions. Let us discriminate between two kinds of state transition graphs: the implementation STG, named simply STG in the following, and the *specification* STG, named $SSTG$. The STG is the completely specified STG which represents the FSM implementation and includes all transition relations between circuit inputs and outputs. It is composed of completely specified transitions. In addition, we also consider the state transition graph representing the FSM specification, the SSTG. The SSTG does not include unspecified transitions for which both the next state function and the output function are not specified, and it can be composed of incompletely specified transitions. We assume that all states of the specification STG are reachable from the reset state, s_0, i.e., there is an input sequence, for each state s_i of the SSTG, that drives the machine into s_i when applied starting from s_0. We also assume that a fault-free reset input is available in the FSM implementation which sets the memory elements to the reset state s_0.

We adopt at the functional level a *single transition fault* model [4], so that, transitions to be tested are selected at the gate level while sequences able to distinguish the faulty next state from *all* other states are identified on the SSTG (see Section IV). Conversely, at the gate level we adopt the stuck-at fault model that is put in relation to the transition fault model as follows. Let us assume that a combinational test pattern generator has identified the set of all test vectors $TVS(f)$ for a stuck-at fault f in the combinational part of a sequential circuit, i.e., $TVS(f) = \{TV(f)\}$. Each test vector $TV(f) \in TVS(f)$ can be associated with a transition $t(i, s_i, s_j, o)$ of the STG. Such a transition is called a *test transition* ($tt(i, s_i, s_j, o)$) and it may not belong to the specification STG if the FSM is incompletely specified.

ASP-DAC '97
0-89791-851-7$5.00

Definition 1 *A stuck-at fault f on a line l of the combinational part of the FSM implementation is functionally redundant (FR) if for each $TV(f) \in TVS(f)$, the corresponding test transition $tt(i, s_i, s_j, o)$ does not belong to the SSTG or it belongs to the SSTG but it propagates the fault only on unspecified output bits.*

This means that *functionally redundant* faults do not alter the specified behavior of the FSM, thus, they can be inserted into (removed from) an FSM implementation without modifying the input/output behavior specified by its SSTG. Moreover, their removal removes all *combinationally redundant* faults, the majority of the *sequentially redundant* faults and an extra set of faults which are redundant with respect to the specification only [5].

Furthermore, definition 1 highlights that a functional test pattern generator based on the SSTG description of a circuit can generate test patterns covering at-most the set of *functionally irredundant* faults. But if *functionally redundant* faults are removed from the FSM (or HFSM) implementation [5], this guarantees that the full stuck-at fault coverage can be achieved as shown in Section V.

III. Hierarchical FSM Model

The *hierarchical FSM* (HFSM) model has been introduced in [5]. It is a subset of the *Statecharts* [6] representation which deals exclusively with the control part of a device. Let us define an HFSM as a 4-tuple $HFSM =< F, I, O, R >$ where F is a set of finite state machines modeled as Mealy machines, I is the set of inputs, O is the set of outputs, and R is the set of *dependency relations* described below.

Each FSM_i, belonging to F, is represented by a 6-tuple $FSM_i =< S_i, I_i, O_i, \delta_i, \lambda_i, s_{i0} >$ where I_i is a finite input alphabet, O_i is a finite output alphabet, S_i is the finite set of states, δ_i is the state transition function, λ_i is the output function and s_{i0} is the reset state. We say that FSM_i *executes* the transition $t(i_i, s_{ii}, s_{ij}, o_i)$ whenever, in the presence of input i of FSM_i (i_i), there is a transition in FSM_i from state s_{ii} to state $s_{ij} = \delta_i(s_{ii}, i_{ij})$ generating output $o_i = \lambda_i(s_{ii}, i_i)$.

A *dependency relation* $\Delta(FSM_i, DI, s_{jk}, FSM_j) \in R$ exists if FSM_j must be in state s_{jk} to allow FSM_i to execute each transition with input $i_{il} \in DI$. $DI \subset I$ is a subset of the inputs of FSM_j corresponding to unspecified transitions. In fact, the *dependency relation* constrains the behavior of the $HFSM$, for all inputs belonging to DI, to be described by the transitions of FSM_i. We say that FSM_i *depends* on FSM_j through s_{jk} and that FSM_j is in *dependency relation* with FSM_i (through s_{jk}). All states s_{jk} are called *hierarchical states* since they allow the execution of transitions of the FSMs which depend on them. A particular state (ID_i), called the *idle* state, is added to the original specification to allow the synthesis tools (e.g. the VHDL synthesizer) to correctly implement the entire architecture. In fact, each FSM_i of a *dependency relation* $\Delta(FSM_i, DI, s_{jk}, FSM_j)$, must be put in an *idle* state whenever it is not allowed to execute any transition, that is, whenever FSM_j is not in the *hierarchical* state s_{jk}.

A. Hierarchical SSTG construction

The HFSM model is well suited for testing purposes since it allows the analysis of one FSM at a time for test generation or testable synthesis as proved in the following. In fact, the synthesis of an HFSM starting from its representation in a hardware description language (e.g. VHDL) produces an implementation where memory elements are in a direct relation with the states of each FSM. This depends on the inability of VHDL synthesis tools to make inter-process or inter-procedure optimizations since each FSM is modeled as a process (set of processes), possibly calling procedures. (The HFSM is modeled as a pair of processes and each FSM as a pair of procedures.) For this reason, the set of states of each FSM_i is implemented through an independent state register[1]. On the contrary, combinational logic is shared among the FSMs implementations due to the minimization performed by logic synthesis algorithms. A global reset signal is inserted by the synthesis process to simultaneously put all FSMs into their idle states.

Let us now concentrate on the exploitation of such characteristics with respect to test pattern generation. The key idea is to consider the possibility of building a global SSTG (in the following HSSTG) for an HFSM specification, without having to enumerate all states generated by the product of all FSMs composing the HFSM. The procedure for building the HSSTG converts each transition $t(i_i, s_{ii}, s_{ij}, o_i)$ of the SSTG of each FSM_i into a transition of the HSSTG. The HSSTG is generated in a time linear with the total number of transitions of the entire HFSM. Note also that the final number of states is equal to the sum of the number of states of each FSM. The detailed procedure is described in [4].

B. HSSTG and testability

Since the HSSTG is composed of all transitions of each SSTG composing the HFSM, it is possible to partition the transitions of an HSSTG into sets related to each original SSTG. From Definition 1, it follows that the identification of *functionally irredundant* faults is based on the mapping of all test vectors $TV(f) \in TVS(f)$, for a target fault f, to test transitions $tt(i, s_i, s_j, o)$ belonging to the SSTG. The definition of the HSSTG allows the proposed testing methodology to be indifferently applied in the same manner to FSMs or HFSMs. In fact, the comparison of the test transitions, obtained from the test vectors, can be performed in the case of an HSSTG by considering each *separated* SSTG that composes it. Thus, the problem of testing *functionally irredundant* faults of an HFSM implementation is reduced to the same problem for a single

[1] Retiming synthesis strategies [3] are not taken into account.

FSM. This assertion is true since the following theorem can be proved [4].

Theorem 1 *For a stuck-at fault f of an HFSM implementation and the corresponding HSSTG, each test vector $TV(f) \in TVS(f)$ corresponds to a test transition $tt(i_j, s_{ji}, s_{jj}, o_j)$ belonging to a single SSTG of those composing the HSSTG.*

IV. Functional/Structural TPG

The proposed test generation algorithm is based on the *FSMTest* algorithm [1] that will be briefly summarized here. The main difference concerns the selection of the set of transitions to be tested, performed at the gate level in our proposed algorithm, while it was performed completely at the functional level in FSMTest.

The algorithm start by identifying a set of transitions of the *SSTG* able to test all stuck-at faults of the corresponding implementation. A test subsequence is then built for each currently analyzed test transition by concatenating the appropriate bridge sequence (BS) and distinguishing sequence (EUIO). The bridge sequence is necessary to drive the FSM to the activation state of the transition under test starting from the arrival state of the previous test subsequence; sometimes it can be null. The distinguishing sequence guarantees, in the majority of the cases, the correct propagation of the fault effects to an output variable. *EUIOs* have the advantage of reducing the total test length, and are easily concatenated and overlapped using the criteria shown in [1]. Moreover, *EUIOs* are not likely to be masked by a fault even if they are built by exploring the fault-free SSTG [4]. Finally, *FSMTest* concatenates and overlaps the identified test subsequence to generate a global test sequence [1].

A. HFSM extension

The following three modifications must be applied to the testing algorithm when an HFSM is considered.

Test vectors. The only difference from the case of a single SSTG concerns the completion of the *don't care* bits of the test vectors before their fault simulation. *Don't care* bits corresponding to input or state variables which are *don't care (-)* in the transitions of the HSSTG can be set to any value. However, state variables which are set to *unknown (U)* in the transitions of the HSSTG must remain unset, i.e., in fact *don't care*. This guarantees that test vectors can be translated in test transitions belonging to a single SSTG of the HSSTG (Theorem 1). Note that, the presence of *don't care* bits into the test vectors implies the use of a three-valued fault simulator to reduce the fault list.

Hierarchical bridge sequences (HBSs). They are necessary to allow the HFSM to execute the transitions of each FSM_i not *top* of the hierarchy. Moreover, HBSs are necessary whenever a transition outgoing from a hierarchical state must be tested. In fact, a test transition $tt(i_j, s_{ji}, s_{jj}, o_j)$ outgoing from the hierarchical state s_{ji} of FSM_j, can impose that all FSM_i which depend on FSM_j must be into a precise state. This depends on the algorithm that randomly completes the *don't care* bits of the test vectors in the case they actually correspond to *don't care* input and state variables of transitions of the HSSTG. Thus, an HBS must thus be used before execution of the test transition in order to apply the corresponding test vector to the HFSM implementation starting from the correct activation state.

Hierarchical EUIOs. In the case of a single FSM, *EUIOs* distinguish the next state of each test transition from all other states of the SSTG; this guarantees the propagation to the primary outputs of faults which are observable on the next state variables only. In the case of test transitions concerning hierarchical states, this is no longer sufficient. First of all, if a test transition $tt(i_j, s_{ji}, s_{jj}, o_j)$ of FSM_j goes out from a hierarchical state (s_{ji}), the *EUIO* propagates to the primary outputs those faults which generate a next state different from the correct one (s_{jj}). But, the application of $tt(i_j, s_{ji}, s_{jj}, o_j)$ also puts all $FSM_i \in children(FSM_j)$ into their *idle* states; thus an *EUIO* would be necessary to verify the correctness of these arrival states. But such *EUIOs* have been proved by experiments to be unnecessary, since *idle* states are the only states that avoid the interference between the outputs generated by transitions of each FSM_i and the outputs of transitions of FSM_j. Thus, if a fault avoids the transition of a FSM_i to its *idle* state, then it is very unlikely that the following transitions of the test sequence cannot identify the fault, i.e., do not generate a wrong output. The second case concerns a test transition $tt(i_j, s_{ji}, s_{jj}, o_j)$ incoming to the hierarchical state s_{jj}. Since it puts all FSM_i, which depend on s_{jj}, into their reset states, an *EUIO* is necessary for the reset state of each FSM_i. Such *EUIOs* are inserted into the test sequence before the actual *EUIO* for s_{jj}.

V. Experimental results

The proposed test generation algorithm has been implemented in C. The set of benchmarks reported in Table I is used to perform all experiments. They represent HFSM descriptions of hierarchical controllers produced with SPeeDCHART [9] (second part of Table I). Such a tool automatically generates a VHDL description of a HFSM which has been synthesized and translated to SIS for technology mapping. All implementations have been mapped onto a subset of the MCNC library gates composed of simple 2-inputs gates (*and, nand, or, nor, xor, xnor, not*). All *functionally redundant* faults have been removed from all mapped circuits by applying the minimization procedure described in [5], thus resulting in *functionally irredundant* circuits. The same methodology also identifies the set of transitions to be tested.

The proposed methodology is compared with HITEC [7] and Veritas [2]. To take into consideration

TABLE I
COMPARISON WITH GATE-LEVEL TPGS.

Benchmarks characteristics									HITEC			atpg (SIS)			Proposed		
name	#S	#I	#O	#M	#F	#L	#G	#Ft	FC%	CPU	TL	FC%	CPU	TL	FC%	CPU	TL
bench	23	4	4	6	14	290	163	324	100.0	14.8s	185	100.0	3.9s	207	100.0	1.5s	93
giga3	96	16	17	3	16	2785	1432	2540	99.7	244.7m	2624	99.8	880.2s	2519	99.8	71.7s	1300
giga2	109	16	17	3	16	3205	1644	2882	99.4	328.9m	2305	99.9	540.6s	1869	99.8	109.1s	1046
giga6	169	16	24	5	26	5036	2579	4444	48.3	-	1166	100.0	879.8s	2022	100.0	217.4s	1144
giga	169	16	24	5	26	6191	3163	5483	99.6	514.0m	2394	99.9	29.2m	2181	99.9	153.6s	2890
giga6r	169	16	24	5	26	4927	2527	4383	99.4	363.0m	3626	51.8	-	-	99.9	208.8s	2853
gigar	169	16	24	5	26	6364	3252	5637	80.4	-	2530	54.7	-	-	99.9	217.2s	3268
giga7	244	30	24	9	46	6631	3413	6028	87.3	-	7646	43.3	-	-	99.4	305.5s	3360
giga4r	309	30	24	12	60	7860	4044	7500	63.3	-	2146	31.0	-	-	99.1	620.1s	4839
giga4	309	30	24	12	60	7957	4093	7580	59.4	-	1215	57.2	-	-	99.0	808.8s	5513
giga5	309	30	24	12	60	7884	4063	7605	59.8	-	1526	35.2	-	-	99.0	721.6s	5539

$\#S$ = # of states. $\#I$ = # of inputs. $\#O$ = # of outputs.
$\#M$ = # of FSMs. $\#F$ = # of memory elements. $\#L$ = # of literals in factored form.
$\#G$ = # of gates. $\#Ft$ = # of equivalent stuck-at faults. $FC\%$ = percentage of fault coverage.
CPU = CPU time in seconds (s) or minutes (m). TL = test length in # of test vectors.

the evolution of the BDD technology we do not use the original Veritas program but its re-implementation into the SIS environment (*atpg* command) compiled with the CUDD package [8]. It is thus possible to apply dynamic reordering techniques and disregard the initial variables ordering. In any case, experiments with the original Veritas did not produce better results.

Results of this comparison are reported on the right of Table I. All programs have been run on the same Sun Sparcstation 20/125 with 128 MB RAM. To perform such experiments, limits have been set on the CPU time and memory usage to 10 hours and 128 Mbytes, respectively. Results produced for the smallest benchmarks show that we achieved the maximum fault-coverage in a fraction of the time required by SIS and HITEC by generating shorter test sequences. Furthermore, our approach can produce good results in terms of fault coverage and test length for HFSM implementations which cannot be analyzed by gate-level TPGs. In fact, HITEC achieves an insufficient low fault coverage due to the high sequential complexity of such circuits. For the last two benchmarks, after 10 hours of CPU time it achieves a fault-coverage less than 60%, while the proposed approach reaches 99% fault coverage. Conversely, Veritas cannot manage BDD relations of such complexity. For instance, *atpg* does not succeed to traverse the largest 6 benchmarks due to memory or time limits and the reported fault-coverage (column 5) is achieved by the random phase only.

VI. Concluding remarks

A testing approach dealing with the design of large sequential circuits using a top-down strategy has been described in this paper. Such an approach uses the functional information, described at the specification level, in conjunction with the gate-level structure to speed-up the test generation process and to enable management of large sequential circuits, otherwise untreatable. From the hierarchical specification of a large controller, the underlying Hierarchical Specification STG is derived, while the gate-level implementation is represented by its combinational part. The gate-level structure is used by the test pattern generator to extract the list of stuck-at faults for which a test sequence must be generated. Then, for each stuck-at fault, the corresponding transition in the $HSSTG$ is identified. Given the test transition, the fault propagation and justification sequences are identified in the $HSSTG$ and concatenated. Redundant faults are also identified and removed.

References

[1] G. Buonanno, F. Fummi, D. Sciuto, and F. Lombardi. FsmTest: Functional test generation for sequential circuits. *INTEGRATION: the VLSI Journal*, 20:303–325, 1996.

[2] H. Cho, S. Jeong, F. Somenzi, and C. Pixley. Synchronizing sequences and symbolic traversal techniques in test generation. *Journal od Electronic testing: Theory and Application*, 10(4):19–31, 1993.

[3] G. De Micheli. Synthesis and optimization of digital circuits. *McGraw-Hill Series in Electrical and Computer Engineering*, 1994.

[4] F. Fummi. Design for testability problems for highly complex circuits. *Ph.D. Thesis, Dept. of Electronics and Information, Politecnico di Milano*, 1995.

[5] F. Fummi, D. Sciuto, and M. Serra. Synthesis for testability of large complexity controllers. *Proc. IEEE ICCD*, pages 180–185, 1995.

[6] D. Harel. STATECHARTS: A visual formalism for complex systems. *Science of Computer Programming, North Holland*, 8:231–274, 1987.

[7] T. Niermann and J.H. Patel. HITEC: a test generation package for sequential circuits. *Proc. European Design Automation Conference*, pages 214–218, 1991.

[8] F. Somenzi. CUDD: CU decision diagram package. *Department of Electrical and Computer Engineering, University of Colorado at Boulder*, 1995.

[9] SPeeDCHART project designer user's manual. *Speed S.A.*, 1994.