An Automated Method for Test Model Generation from Switch Level Circuits

Tim McDougall
Atanas Parashkevov

Motorola Inc.
2 Second Avenue, Mawson Lakes,
South Australia, 5095.
Phone: +61 8 8168-3500
tmcdouga,aparashk@asc.corp.mot.com

Simon Jolly

Foursticks Pty. Ltd.
2/259 Glen Osmond Rd,
Frewville, South Australia, 5063
Phone: +61 8 8338-5500
sjolly@foursticks.com

Juhong Zhu, Jing Zeng,
Carol Pyron, Magdy Abadir

Motorola Inc.
7700 W. Parmer Lane
Austin, TX 78729, USA
Phone: +1 512 996-4906
juhong,zeng,cpyron,abadir@ibmoto.com

Abstract — **Custom VLSI design at the switch level is commonly applied when a chip is required to meet stringent operating requirements in terms of speed, power, or area. ATPG requires gate level models, which are verified for correctness against switch level models. Typically, test models are created manually from the switch level models—a tedious, error-prone process requiring experienced DFT engineers. This paper presents an automated flow for creating gate level test models from circuits at the switch level. The proposed flow utilizes Motorola's Switch Level Verification (SLV) tool, which employs detailed switch level analysis to model the behavior of MOS transistors and represent them at a higher level of abstraction. We present experimental results, which demonstrate that the automated flow is capable of producing gate models that meet the ATPG requirements and are comparable to manually created ones.**

I. Introduction

Contemporary chip design depends critically on the availability of appropriate methodologies and tools in order to keep up with the ever-increasing chip complexity. Typically, logic designers describe a circuit at the Register-Transfer Level (RTL), and synthesis tools are run to compile RTL designs into a technology-specific cell library. When a chip has to meet stringent operating requirements, e.g. speed, power and, less often, area, certain parts of the chip may be manually implemented with custom circuits carefully tuned at the switch level. This practice, commonly referred to as custom design, is especially popular in the design of high performance processors. The manual efforts by the designers made at the switch level make it challenging to test custom designs.

To establish a DFT methodology for custom designs or designs in general, we need to establish a set of criteria to measure the merits of generated gate models. This includes the level of automation and the associated turn-around time, the size of the resulting test models, the number and test coverage of test patterns, and the ability to support diagnosis.

One way of generating test models is to obtain them from RTL models directly without optimization. This is used for testability analysis early on during the design flow. However, RTL models do not contain the implementation details of the circuits, which makes diagnosis of silicon failure hard using this approach.

Another way of generating gate level models is to manually inspect switch level circuits and reason about their functionality. This process requires not only expertise, but also time to optimize the size of test models, test coverage, test vector length, correspondence between the fault locations in the test models and the defect locations in the switch level circuits. Manual test view generation for custom blocks is a very ad-hoc, tedious and error-prone activity. Therefore, it would be very beneficial to automate the test model creation process by capturing the test specific knowledge in a tool.

An internal Motorola transistor level analysis and extraction tool, SLV (Switch Level Verification) has been developed to handle the generation of gate level test models for custom designs. The highly automated test model generation flow, called the SLV/ATPG flow, utilizes SLV in conjunction with the Vgate synthesis tool to produce a gate level model. Unlike commercial synthesis tools, Vgate performs simple-minded synthesis that preserves the original structure of the RTL model without modification.

The SLV tool has been used in equivalence checking flows for designs at the switch level [8]. This paper describes the SLV/ATPG flow for automating the generation of test models from switch level custom circuits. Section II discusses relevant research and tools performing switch level modeling and abstraction. SLV/ATPG is discussed in Section III, which provides insight into the traditional flow employed by designers, key algorithms affecting the suitability of the flow, and a user's perspective of the suitability and usability of the flow. Section IV presents experimental results of the SLV/ATPG flow in use at the Somerset Design Center of Motorola, and compares it to the manual flow previously employed. A summary is provided in Section V.

II. Previous Work

In the domain of test model generation, there have been mainly two approaches. One approach, exemplified by Grog [1] and MatchMaker [2], is to perform extraction of an RTL/gate level representation via structural pattern matching using some sort of library of structural models or pattern-matching rules. Structural pattern matching abstracts away elements of the switch level circuit that match a certain structure and replaces it by an equivalent gate level representation. The operation and the correctness of such tools very much depends on the library used.

The main issue with the pattern-matching approach is the level of automation that can be reasonably achieved. Also, the extracted gate level representation may not accurately capture behavior caused by subtle switch level phenomena, such as charge sharing and sneak paths. For example, the behavior of a pass-transistor circuit may depend on its operating conditions and may be correct only for a subset of all possible input signals. Structural pattern matching tools are not capable of such detailed level of analysis.

An alternative approach is to employ symbolic switch level analysis based on a switch level model of MOS transistor behavior [3]. This involves performing appropriate symbolic computations to extract a functional representation that faithfully captures the switch level circuit behavior. The switch level models used are an abstraction of the actual circuit behavior and capture a number of attributes, such as transistor strengths and charge node size. The whole circuit is usually partitioned into a set of channel-connected components (CCCs), which may be analyzed independently.

There is a growing number of switch level analysis tools that are available. Anamos [3] and Tranalyze [4] create a gate level model that contains unit delay gates and has an extra simulation clock input, which makes them not applicable to testing custom circuits. LayBool [5] (now called LynxLB) is able to produce an RTL description of a switch level circuit. However, in our experience, it has trouble handling some common circuit structures, e.g. latches with asynchronous inputs and domino logic. OLCC [6] is an experimental tool that only handles combinational logic. GateMaker [7] is an IBM internal tool that produces gate level views for switch level models and is used at IBM for ATPG purposes. Therefore, GateMaker appears well tuned for the needs of testing custom blocks. However, this tool still requires a reasonable amount of manual effort, e.g. in annotating the input design whenever combinational or sequential structural loops are present.

The work presented in this paper utilizes a tool platform called SLV (Switch Level Verification) developed at Motorola Inc. Given a switch level circuit, SLV produces a functionally equivalent RTL model that preserves the hierarchy and overall structure of the input design. SLV builds on the state-of-the-art switch level analysis techniques discussed above and brings about novel algorithms for analyzing pre-charged and sequential logic, gated clocks, structural loops, and hierarchical designs [8]. These techniques significantly reduce the manual intervention required to translate a switch level design into an RTL model. In most cases, the user has to specify only top-level input (environment) constraints and clock signals.

III. The Automated Test Model Generation Flow

The need for automation and tool support for test view generation for custom blocks is addressed in the flow illustrated in Figure 1. A two step process is used to obtain a gate level test model from a switch level design. The first step applies the SLV tool to create an RTL model from the switch level input design. The RTL model has to be not only functionally equivalent but also closely resemble the structure and hierarchy of the original circuit to facilitate fault diagnosis. The second step utilizes Vgate to perform synthesis and obtain a gate level model that maintains the RTL model structure. This gate level model is suitable as input to an ATPG tool for test pattern generation. The breakdown of the process into two steps helps to compartmentalize the function of each tool and allows the flexibility of handling other applications. Also, the technical challenges that SLV and Vgate solve in the flow are distinctly different.

The SLV tool employs state-of-the-art algorithms that cover the wide range of custom MOS logic design styles and structures in use at Motorola. SLV has the ability to detect, resolve and generate an equivalent RTL model for combinational structural loops, sequential elements such as latches and flip-flops, and pre-charged logic such as pre-charge gates and domino logic pipelines. These algorithms, while important as they determine the range of circuits which our automated test model generation can handle, are beyond the scope of this paper. In the rest of this section, we discuss the main SLV-related aspects that affect the suitability of the test model produced by the automated flow.

Hierarchical Analysis. The SLV tool employs hierarchical switch level analysis, which reduces memory consumption and processing time by analyzing each logic block in the circuit once only, and in relative isolation to the rest of the circuit. As a result, the model generated has a high degree of structural similarity to the input design, which is useful for diagnosis purpose.

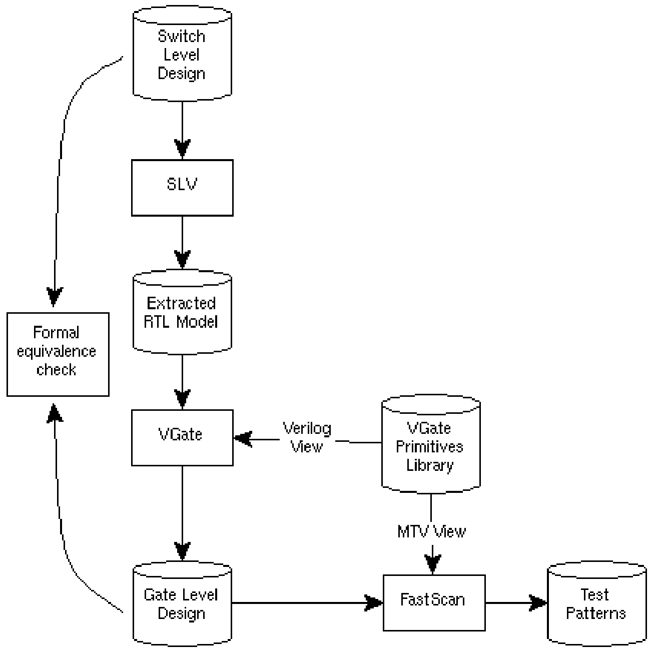

Fig. 1. The SLV/ATPG Automated Test Model Generation Flow.

The hierarchical analysis employed by SLV involves three steps:

1. Pre-processing of the design hierarchy and selective flattening of parts of the design hierarchy to ensure the correctness of the switch level analysis;
2. Evaluation of the design with each non-flattened hierarchical block being analyzed only once;
3. Creation of a hierarchical RTL model with instances of non-flattened blocks in the design hierarchy.

With this approach, SLV rarely requires any flattening to be performed during the pre-processing of the design when the input circuit is partitioned in a logical manner with accurate port directions declared in the input model.

Structure Based Output Generation. The generation of an RTL description for nets based on their Boolean functions provides functionally correct output. However, often this is not the preferred way of modeling the functionality or does not reflect the structure of the circuit being analyzed, which is required for an optimal test model. This is illustrated in the simple NAND gate example shown in Fig. 2.

The pull-up and pull-down functions of the net ‘out’ are given by:

```
out.pull-up = ~a | ~b
out.pull-down = a & b
```

As the functions of the net ‘out’ are Boolean, an example RTL description for net ‘out’ in the Verilog HDL would be:

```
assign out = ~a | ~b ;
```

The Verilog above is functionally correct but does not structurally model a NAND gate. A preferred, equivalent structural representation for net ‘out’ would be:

```
assign out = ~( a & b ) ;
```

The ability to produce a gate level test model structurally similar to the initial switch level design is the single most important factor that influences the suitability of the resulting test model. To address this, the SLV tool employs a technique that, in addition to using the resulting net functions, examines the structural relationships within CCCs in the circuit and creates a model closely modeling these.

The structural based output technique is employed on a net by net basis, and performs a depth first path traversal through the gates and transistors of the CCC containing the net. During the traversal the algorithm examines the functions of nets and the connected circuitry, and attempts to match some basic circuit design styles with the structure of the CCC being explored. If a match is found, then the RTL output is created based on the known structure, hence maintaining structural similarity. Otherwise, SLV reverts to BDD based output techniques.

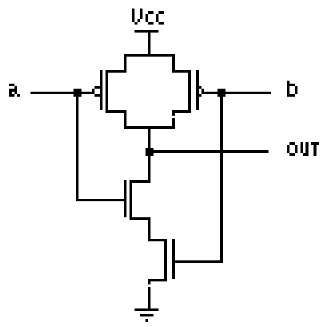

Fig. 2. A Simple Nand Gate.

The structural output technique will, in general, produce a single RTL statement for each path explored from the net where the traversal commenced, which is desirable for components such as multiplexers. The technique is quite effective at producing structurally similar output for the majority of designs, particularly for small leaf-cells.

Modeling of Weaker Strength Circuitry. The explicit path enumeration technique employed by SLV is complicated by the introduction of resistive transistor devices and different strength levels for nets in the analysis. SLV is capable of modeling resistive transistor devices and weak driving strength logic gates by introducing the discrete strength levels defined in Verilog.

The incorporation of these rules into the explicit path enumeration is a two step process involving:

1. The calculation of a driving strength for each conducting path, with separate pull-up and pull-down functions maintained for each driving strength;
2. The combination of all pull-up functions and all pull-down functions into a single pull-up and pull-down function in a way that captures the notion that a stronger driving strength dominates the effects of a weaker one.

As an example, consider the multiplexer design shown in Fig. 3. The pull-down weak transistor only affects the output if all the select signals are turned off.

SLV produces the following structural based Verilog RTL description:

```
assign out = sel1 ? ~data1 : 1'bz ;
assign out = sel2 ? ~data2 : 1'bz ;
assign out = sel3 ? ~data3 : 1'bz ;
assign (pull0 , pull1) out = pulld ? 1'b1 : 1'bz ;
```

Input (Environmental) Constraints. In some designs there are implicit relationships between the behaviors of certain input signals, usually due to the manner in which the design is to be used. The SLV tool does not make any assumptions regarding these relationships without them being explicitly stated by the user – otherwise, the accuracy of generated models would be compromised without the user's awareness.

The example multiplexer in Fig. 3 is presented with no knowledge of its intended environment, and as such the Verilog RTL description produced contains a set of potentially clashing tri-state buffers. The user can provide information about the intended environment by using a constraint, e.g. a 'one-hot' constraint which implies that exactly one specified signal is high and all others are low at any given time. In a defect-free design the input 'pulld' only affects 'out' in a defective circuit when all selects are off. A 'one-hot' constraint could be used to inform SLV about the intended relationship between inputs 'sel1', 'sel2', 'sel3', and 'pulld'. When such a constraint is supplied, SLV produces the following Verilog RTL description:

```
assign out = sel3 & ~data3 | sel2 & ~data2 |
sel1 & ~data1 | ~sel1 & ~sel2 & ~sel3 & pulld ;
```

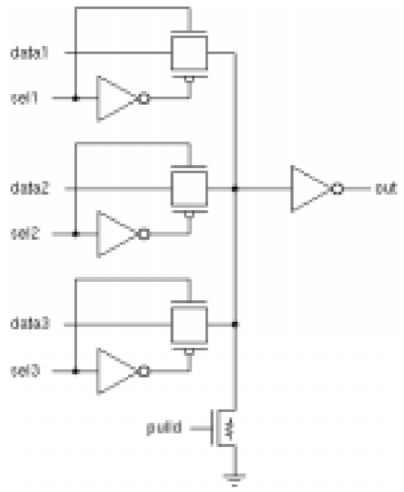

Fig. 3. An example multiplexer.

The above output has changed to a single combinational assignment to net 'out', and structurally represents a multiplexer without unnecessary reconvergent fanout structures. Input constraints provide a level of control regarding the structure of the RTL model produced. This clearly has positive implications for the suitability of the created test model. In addition, the performance and capacity of the SLV tool may also improve as a result of paths that do not conduct under the constraint conditions being eliminated.

Leaf Level Constraints. The main user provided guidance required by SLV is the identification of certain constraints of leaf cells. Since SLV cannot identify leaf level constraints systematically, the resulting model can be larger than necessary with poor test coverage. An example of such a leaf-cell implemented using pre-charge logic is shown in Fig. 4.

The nmos device with 'a2' as a gate signal is shared in forming the logic for the three output signals 'out1','out2' and 'out3'. Without recognizing the intended constraints between inputs 'a2', 'a1' and 'a3', and similarly between inputs 'b2', 'b1', and 'b3', a complete CCC analysis renders the logic functions as:

```
out1 = clk & ( a1 | b1 & ( a3 & b3 | a2 ) )
out2 = b2 & clk & ( a1 & b1 | a3 & b3 | a2 )
out3 = clk & ( a1 & b3 & b1 | a3 | b3 & a2 )
```

The term 'b2 & clk & a1 & b1' that the SLV tool determines for output 'out2' comes from the path 'p', indicated using a thicker solid line in Fig. 4. This path does not contribute to the output 'out2' if the 'b1' and 'b2' inputs are never high at the same time.

A 'one-hot' constraint specified between inputs 'a2', 'a1' and 'a3', and similarly between inputs 'b2', 'b1', and 'b3', results in the following logic functions capturing the original intent of the designer:

```
out1 = clk & ( a1 | a2 & b1 )
out2 = clk & b2 & a2
out3 = clk & ( b3 & a2 | a3 )
```

To assist in the identification of constraints, heuristics are employed to search for multi-sharing structures in leaf cells, which reduce the number of leaf cells that need to be inspected and potentially to be provided with constraints for test model creation. For example, only multi-output cells are examined. The SLV tool can also be used to assist in the detection of multi-sharing structures by detecting and reporting conducting paths that pass through two or more pre-charged nets in a single CCC. To prevent users from specifying constraints for cells that are not guaranteed by the instantiating environment, all leaf level constraints supplied to SLV for test model generations are first proved correct using a separate formal verification flow.

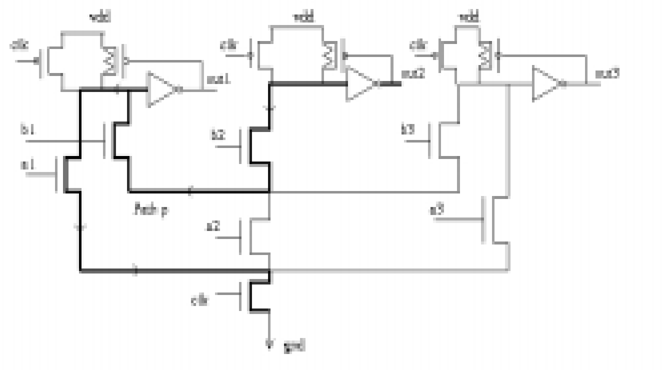

Fig. 4. Leaf Level Constraint Example.

Component Absorption. The SLV tool utilizes a user selectable technique called component absorption. The aim of component absorption is to reduce the number of statements in the RTL description without negatively impacting the structural similarities to the input design for an optimal test model. The component absorption technique allows the user to simplify the resulting RTL model produced by SLV.

Component absorption is applied on a net by net basis after the net functions have been determined using explicit path enumeration and prior to output generation. The local pull-up and pull-down functions of each output net of the circuit are examined to determine if they meet the selected compaction requirements, which can be:

- The functions implement a buffer or inverter; or
- The functions implement an arbitrary simple component and are smaller in size than a user definable threshold.

If the local functions of the net meet either of the requirements selected by the user, then the local functions of each input to the CCC containing the output net are absorbed into the output nets functions. This process is repeated until the local functions of the net do not meet either of the compaction requirements. The compaction algorithm traverses by examining the local functions of each net on which the functions of the output net depend, and so on until the input nets of the design are encountered.

While a more compact test model is generally created when the SLV component absorption feature is enabled, it is not always the case. This can be an issue for test model generation. Fig. 5 illustrates an example employing inverters. For the design the logic functions are determined by SLV to be:

```
net1 = F1(a,b)     net2 = F2(c,d)
x = ~net1  z = ~net2
y = net1 & net2
```

where Fn(s,t) implies that the logic functions are the relevant function Fn of inputs ‘s’ and ‘t’.

When compaction of buffers and inverters is enabled, the internal nets ‘net1’ and ‘net2’ are no longer in the RTL description and the logic expressions for ‘x’, ‘y’ and ‘z’ become:

```
x = ~F1(a,b)
y = F1(a,b) & F2(c,d)
z = ~F2(c,d)
```

This introduces unused gates into the test model when ‘x’ and ‘z’ may not be used when the circuit is instantiated. However, when the compaction of buffers and inverters is disabled, SLV preserves the original design intent. Since diagnostic analysis traces the back cone logic of fails, multiple instantiation of F1 will hinder the analysis.

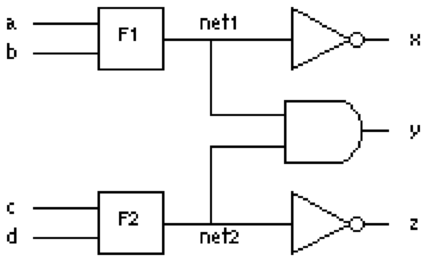

Fig. 5. The Effect of the Absorption Option of SLV.

A. The Application of the Vgate Tool to Automate the Generation of a Gate Level Model

The Vgate tool was developed at Motorola Inc, Austin, for the purpose of creating a gate level model from an RTL description. It employs simple synthesis to create the gate level model, the primary aim of which is to preserve the structure of the input RTL description. Vgate is utilized in the automated test model generation flow to produce the gate level test model from the RTL Verilog model obtained by SLV. The following sections present the main considerations of the Vgate tool with respect to producing an optimal test model.

Multiple Input Gates vs. Cascading Gates. A factor that can significantly increase the size of the gate level test model is the decomposition of devices with multiple inputs into groups of devices with fewer inputs. This results in a test model with cascading gates with few inputs each, thus introducing priorities on the input signals. Multiple input devices should be used in preference, thus simplifying the test model and representing the functionality without priorities.

The Vgate tool provides the ability to switch between these two output styles and the multiple input gates option is selected for the automated test model generation flow.

Modeling of Scan Latches. The issue of scan latch modeling for the test view requires a project specific decision. Vgate provides support for this by identifying and compiling specific latches in the RTL design as scan latches in the test model, e.g. a 2 port LSSD. To enable this, users of the SLV/ATPG flow need to manually identify which top-level input signals are to be regarded as scan clocks.

B. User Flow for ATPG Gate Level model creation

To handle test model creation for complex custom blocks, the automated flow was established using the SLV/ATPG technology. The detailed flow of the application of the SLV/ATPG automated test model generation process is shown in Fig. 6. There can be many leaf-cells in a custom circuit, and each can be contained at the top-level of the hierarchy or at some intermediate level.

SLV/ATPG is executed twice in the applied semi-automated flow. In the first pass, SLV/ATPG is executed on the top-level custom circuit with the top-level constraints specified for the design. If leaf-cell constraints have been identified and verified, SLV/ATPG may have to be run a second time on these leaf cells with the identified constraints, with the goal of obtaining a more compact test model for these leaf-cells. The more compact test models for the leaf-cells are used to replace the corresponding ones from the first run (illustrated using dashed lines). The second pass of the SLV/ATPG flow is performed with several combinations of settings of SLV/ATPG switches that affect the generation of test models. This allows the gate level test model to be finely tuned.

IV. Experiments and Results

Automatic Test Model Generation. The SLV/ATPG automated test model generation flow has been evaluated extensively at the Somerset Design Center of Motorola. It is considered a proven technology and is being used and improved for on-going projects. In Table 1, we compare SLV/ATPG generated test models with the corresponding test models created manually by test engineers as indicated by the subscripts ‘t’ and ‘m’ in the table. The designs created using the SLV/ATPG flow were generated directly by executing SLV at the top-level in the design hierarchy with constraints in the single pass I. We use a commercial ATPG tool to measure test quality of all test models.

The design name abbreviations in Table 1 follow these conventions: *dm*: dynamic multiplexer; *dmn*: dynamic multiplexer with n select signals; *dc*: dynamic comparator; *dmxy*: dynamic x:y multiplexer; *dr*: dynamic rotator; *drn*: n-bit dynamic rotator; *sh*: shifter; *a*: adder; *an*: n-bit adder.

The categories in the first row of the table are defined as follows:

TCov:	test coverage: percentage of the detected faults over the testable faults.
FCov:	fault coverage: percentage of the detected faults in relation to the total faults.
Patt:	the number of test patterns.
Gates:	total number of gates in the test model.
RE:	the number of redundant faults.
CFU:	collapsed fault list (see FU below).
FU:	‘full’ or complete faults

As shown in Table 1, for the comparison categories listed above, the data from the manual and automatic flows are comparable with the following observations:

- Gate size: the SLV/ATPG generated test models are consistently slightly larger than the manually created test models, with the difference being the largest for the 50-bit adder design ‘a50’, and no difference for the shifter design ‘sh’ which is the second largest design of all;
- Test coverage: the manual and automated flows give the same results for the majority of the designs. The exceptions are the dynamic comparator design ‘dc’, for which the manually created test model produces a slightly better result, and the 50-bit adder design ‘a50’, for which the automatically created test model gives a slightly better coverage. To achieve a given test coverage target, the two flows require approximately the same number of test patterns;
- Fault coverage: depending on whether redundancies exist in the resulting models, the fault coverages between the test models vary. The manually created models for the 8/16-bit dynamic rotators contain more redundancies than those created automatically, hence their fault coverages are lower than those of the automatically created models. On the other hand, the automatically created model for the dynamic multiplexer design has a higher redundancy count with a corresponding lower fault coverage compared to the manually created model.

Semi-Automatic Test Model Generation. The semi-automated test generation flow (using both pass I and pass II) was performed on the remaining designs, with constraints for leaf-cells identified and SLV/ATPG being executed with different combinations of switch settings in pass II. The combinations of switch settings identified as affecting the automated test models (named as ‘t1’, ‘t2’, ‘t3’, and ’t4’) are shown in Table 2.

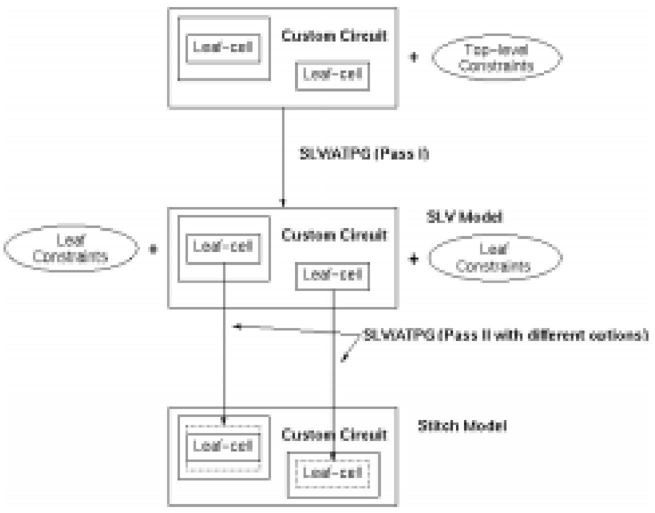

Fig. 6. Semi-Automated Test Model Generation.

The comparison of the semi-automated generated test models and the manually created test models again indicated by the subscript ‘t’ and ‘m’ after the design name is presented in Table 3.

Name	TCov	Fcov	Patt	Gates	RE	CFU	FU
$dm2_m$	100	100	9	15	0	26	62
$dm2_t$	100	100	6	17	0	26	66
$dm4_m$	100	100	18	25	0	46	114
$dm4_t$	100	100	10	27	0	42	110
dc_m	96.58	96.58	26	51	0	88	234
dc_t	95.29	95.29	26	59	0	103	276
$dm41_m$	100	100	20	69	0	150	338
$dm41_t$	100	96.17	22	81	16	178	418
$dr16_m$	100	92.83	43	304	118	784	1814
$dr16_t$	100	96.19	37	325	63	768	1864
$dr32_m$	100	97.72	79	820	99	1860	4684
$dr32_t$	100	97.22	83	854	131	1956	5004
$dr8_m$	100	85.06	41	176	116	482	1098
$dr8_t$	100	92.46	37	196	52	450	1114
sh_m	100	100	222	1699	0	3438	9706
sh_t	100	100	222	1699	0	3438	9706
$a50_m$	99.88	96.35	257	1720	323	3450	9146
$a50_t$	99.90	96.96	270	2180	323	3613	10986

Table 1. Automatic Test Model Generation.

The design names were abbreviated in Table 3 and follow the same convention as in Table I while “lza” is a 50-bit lza adder. Table 3 shows that all designs have at least one combination of SLV switch settings that result in the SLV/ATPG generated test models possessing improved test and fault coverages than those corresponding to the manually created models. The sole exception is the 36-bit adder design ‘a36’ for which the manually created test model has test and fault coverages marginally better than those of the SLV/ATPG created test model. The sizes of the test models (in terms of the number of gates) are comparable between the manually flow and the SLV/ATPG flow, and are considered acceptable by test engineers. The selection of exactly which test models created by SLV/ATPG are used is left to the test engineers.

Name	-absorb_buf_inv	-absorb_components	-optimize_output
t1	on	on	on
t2	off	on	on
t3	on	off	on
t4	off	off	on

Table 2. SLV Command-line Switch Settings.

Name	TCov	Fcov	Patt	Gates	RE	CFU	FU
$a52_m$	89.96	83.16	168	2431	708	6176	14212
$a52_{t1}$	98.66	91.42	194	3319	664	6280	17462
$a52_{t2}$	99.07	91.54	169	3555	815	6203	18260
$a52_{t3}$	98.66	91.42	194	3319	664	6280	17462
$a52_{t4}$	99.05	91.66	171	3746	815	6203	19024
$a32_m$	88.40	80.42	160	2045	630	4606	11606
$a32_{t1}$	98.05	90.99	175	2190	446	4261	11614
$a32_{t2}$	98.77	91.47	162	2333	449	4269	12150
$a32_{t3}$	94.38	80.27	196	2670	1541	5461	14782
$a32_{t4}$	94.31	80.22	198	2670	1539	5461	14782
$a36_m$	98.16	94.31	136	3010	555	6650	16606
$a36_{t1}$	97.56	93.41	155	3323	628	7082	17886
$a36_{t2}$	92.25	87.09	145	4156	1042	8607	23248
$a36_{t3}$	97.52	93.36	171	3323	629	7074	17870
$a36_{t4}$	93.06	87.98	145	4617	1134	8319	25016
lza_m	84.63	80.90	117	2770	584	6112	15014
lza_{t1}	97.95	93.39	146	2661	502	5510	13762
lza_{t2}	97.75	93.95	138	2826	480	5526	14338
lza_{t3}	97.95	93.39	146	2661	502	5510	13762
lza_{t4}	97.96	94.50	136	3185	480	5526	15774
$a10_m$	83.85	71.67	52	440	440	1017	2506
$a10_{t1}$	90.79	75.32	51	369	369	915	2160
$a10_{t2}$	93.31	80.68	65	403	403	917	2298
$a10_{t3}$	90.79	75.32	51	369	369	915	2160
$a10_{t4}$	93.49	81.14	59	417	417	917	2354

Table 3. Semi-Automated Test Model Generation.

The effort spent on identifying leaf cell constraints is very reasonable. The heuristics were automated to search for leaf cells that may require constraints, and they resulted in three to four leaf cells requiring manual intervention for most designs, except seven cells required intervention for the 36-bit adder.

It was noticed that the original switch level circuits for the 8-bit and 16-bit dynamic rotators have the dynamic clock signal connected to the final transistor in the footing of the pre-charge gate. The test models created manually for these cells contain the dynamic clock signal being conjuncted with many intermediate logic signals, while the model created by the SLV/ATPG flow correctly reflects the switch level design by containing the dynamic clock signal conjuncted with the entire resulting logic functions. Although there was no functional difference, the manually created test models in these cases lacked consistency with the original switch level model, compared to the SLV/ATPG created test models.

Chip Level Test model generation comparison. The SLV/ATPG flow is used to automate test generation for custom circuits in a chip. To compare the quality of test model generated by manual efforts and the SLV/ATPG flow at the chip level, the following experiment is conducted. Two chip test models are compiled: one using SLV/ATPG generated test models for all the custom circuits; the other using manually created gate models. The ATPG data is shown in Table 4.

Method	TCov	Fcov	Patt	Gates	RE	CFU	FU
MAN	93.00	84.45	2229	1630458	117932	2674300	6525582
SLV	93.19	84.61	1971	1640403	118472	2683230	6568870

Table 4. Chip Level Test Model.

In general, it takes many weeks to generate manual test models. The SLV/ATPG flow requires a couple of hours to identify the constraints for leaf-cells. The remainder of the flow is automatic including the verification efforts, and can be finished within a day. A good example is the 50-bit adder ‘a50’ that is a very complex design for which the test model can now be created automatically.

V. Conclusions

In this paper, we presented a highly automated flow generating test models for custom designs from switch level circuits. The main algorithmic challenges that impact the suitability of created test models have been discussed. We demonstrated experimental data to compare directly a set of test criteria between the manually created models and those generated using SLV/ATPG flow. The evaluation shows that SLV/ATPG flow is capable of producing gate level test models with good structural similarity and comparable test coverage and model sizes to those of the manually created ones. The SLV/ATPG flow has been widely used and adapted at the Somerset Design Center of Motorola.

VI. Acknowledgments

The authors would like to thank Bruce Jilek and George Joos from the Somerset Design Center for their contributions to the development and beta testing of the SLV/ATPG flow in a real design environment. Thanks also to Carl Pixley, Michael Burns, and Kurt Shultz from the Design Verification Group of Motorola in Austin, for their cooperation and support, and the development of the Vgate tool. The authors would like to thank Neil Boroky, and Craig Biggs from Motorola in Australia for their notable contributions at various stages in the development of the SLV tool.

VII. References

[1] Groupe Bull, *“Generalized Recognition of Gates: User's Guide”*, Version 06, April 4 1996.

[2] Motorola Semiconductor Israel Limited, *“The MatchMaker Tool User Guide”*, Motorola Internal Document, June 1997.

[3] R.E. Bryant, *“Boolean Analysis of MOS Circuits”*, IEEE Transactions on CAD, vol. 6, no. 4, pp. 634-649, July 1987.

[4] R.E. Bryant, *“Extraction of Gate-Level Models from Transistor Circuits by Four-Valued Symbolic Analysis”*, IEEE ICCAD'91.

[5] COMPASS Design Automation, *“Laybool User Guide”*, Beta Test Edition, April 1997.

[6] A. Gavrilov, S. Gavrilov, D. Blaauw, G. Vijayan, S. Pullela, and S. Moore, *“Resynthesis of Static CMOS Circuits for Low Power”*, Motorola Internal Document, 1997.

[7] S. Kundu, *“GateMaker: A Transistor to Gate Level Model Extractor for Simulation, Automatic Test Pattern Generation and Verification”*, IEEE International Test Conference, pp. 372-381, 1998.

[8] S. Jolly, A. Parashkevov, T. McDougall, *“SLV: A Tool for Equivalence Checking of Custom Circuits at the Switch Level”*, IEEE Design Automation Conference, pp. 299-304, June 2002