2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA
Sept 25-26, 2014
Hotel Park Plaza
Bangalore, India

Model Extraction for designs based on switches for Formal Verification

Amar Patel, Lead Member Technical Staff, Mentor Graphics Corporation, Noida, India
(amar_patel@mentor.com)
Naman Jain, Senior Engineering Manager, Mentor Graphics Corporation, Noida, India
(naman_jain@mentor.com)

Gate-level designs often are described at the switch level, but these constructs typically are not handled well by formal tools. This paper focusses on extracting "formal-friendly" models for designs that use switches, particularly bidirectional ones. We take a basic example of a gate implemented using bidirectional switches and show how to extract its formal model. We detail strategies to optimize bidirectional switches in the design and showcase results we achieved with a customer's design. This work targets extracting a formal friendly model for a specific design. But this is a generic solution—for a customer's real designs. These "formal-friendly" extracted models can be used for both formal verification as well as equivalence checking.

Keywords— Verification, Designs, Formal Verification, Switches, Tran.

I. INTRODUCTION

Technology is advancing at an astounding pace. Designs inch towards multi-billion-gate sizes and verification tools and methodologies must continuously evolve to cater to these ever-increasing design sizes. Algorithms also must evolve to handle increasingly-complex designs. Now, formal analysis has a very important role in design verification. Its use is gaining momentum, not only for block-level designs but also for SoCs. Various sophisticated formal analysis solutions, such a security and connectivity formal property checks are performed from block-level design creation through block integration. These advances require formal verification not only of RTL blocks but also of designs having gate-level portions.

Switch-based designs—usually with gate-level module descriptions—exist in many design flows where formal verification would be helpful. Such designs get these gate-level models from pre-defined libraries (provided by fabricators or pertaining to particular technologies) and they also can be generated by design verification tools.

Traditionally, to verify such designs with formal analysis, the verification engineer somehow provides the tools with "formal-friendly" models for library cells. Major issues with such a methodology are: these models often are not available and if they are, they are designed specifically for formal verification. These "formal-friendly" models are not related to the real design logic, so the engineer verifies something that does not go into the actual chip.

In this paper, we present an automated model extraction capability used for formal verification and equivalence checking. Our modeling is "formal-friendly," ensuring optimal performance from formal algorithms. We have successfully implemented our model extraction capability in various situations using formal methods on real life designs. The analysis tool used for this process is the Questa® Formal verification platform from Mentor Graphics Corporation.

In section II and III, we introduce models extracted for MOS and bidirectional switches and we present strategies to optimize bidirectional switches in the design. In the subsequent sections we describe some attributes of extracted models—functional accuracy, formal engine "friendliness" and ease of debugging. We do not need to consider strength and delay specifications of switches while extracting their models. Models of resistive switches are equivalent to their non-resistive switches, so, we only address non-resistive switches in the following sections.

II. MODEL EXTRACTION OF MOS SWITCHES

Switches are SystemVerilog primitives which provide specific functionality to the design. This section will cover different types of MOS switches available in SystemVerilog LRM (see IEEE Std. 1800-2009, Chapter 28) and how they can be best modeled in continuous assignment model.

MOS switches are unidirectional. They act as buffers between their input and output pins. Input pin data flows to the output pin when the switch's control pin values assumes values that activates (i.e. turn ON) the switch. When the switch is off, the data signal is not buffered from input to output pin. There are 3 types of SystemVerilog non-resistive MOS switches – pmos, nmos and cmos. The pmos and nmos switches each have three terminals - data *output*, data *input* and *control*.

1) PMOS Switch:

Passes the signal from *input* pin to *output* pin when *control* pin is '*0*', '*x*' or '*z*'. For the pmos instance *pmos p1 (out, data, control)*, the equivalent continuous assignment model is:

$$out = (control === 1)\ ?\ \text{z} : \text{data} \tag{1}$$

2) NMOS Switch

Passes the signal from *input* pin to *output* pin when *control* pin is '*1*', '*x*' or '*z*'. For the nmos instance *nmos n1 (out, data, control)* the equivalent continuous assignment model is:

$$out = (control === 0)\ ?\ \text{z} : \text{data} \tag{2}$$

3) CMOS Switch

Combination of *pmos* and *nmos* switches, having four terminals: data *output*, data *input* and two *control* pins (one for the *n*-channel and the other for the *p*-channel). The *cmos* instance *cmos c1 (out, data, nControl, pControl)* is equivalent to the combination of *nmos* and *pmos* instances *nmos n1 (out, data, nControl)* and *pmos p1 (out, data, pControl)*. Its equivalent continuous assignment model is:

$$out = ((\text{nControl} === 0)\ \&\&\ (\text{pControl} === 1))\ ?\ \text{z} : \text{data} \tag{3}$$

III. MODEL EXTRACTION OF BIDIRECTIONAL SWITCHES

SystemVerilog has three types of non-resistive bidirectional switches: *tran*, *tranif0* and *tranif1*. A tran switch has two inout pins and is always active. Each tranif0 and tranif1 switch has a control pin that determines whether or not the switch is active: tranif0 is active when control is not 1 and tranif1 is active when control is not 0.

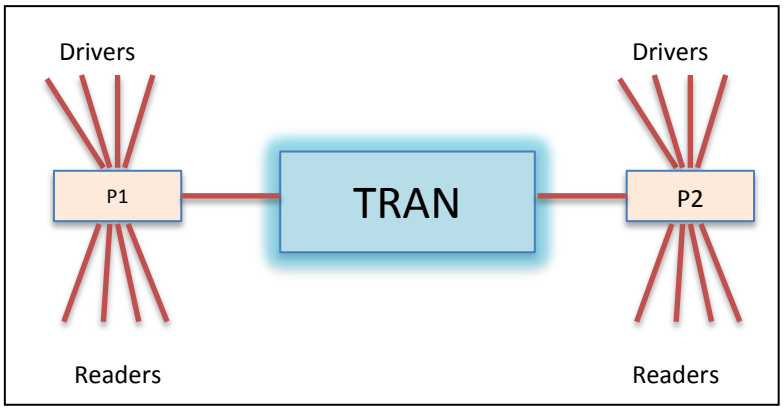

Figure 1 Bidirectional switch tran (p1, p2)

Unlike MOS switches (which have one input pin and one output pin), bidirectional switches have two inout pins. When a MOS switch is active, data is buffered from input pin to output pin. When bidirectional switch is

active, data can be conducted in either direction between the inout pins. When it is inactive (i.e., turned OFF), data is not conducted between the inout pins. Since no definite direction of data flow exists for bidirectional switches, they are difficult to model in a synthesizable manner. There is no continuous assignment equivalent to a bidirectional switch. Strength is not taken into consideration, so we assume both signals have same strength and data conduction direction depends only on whether or not the inout pin has a value assigned to it. To accurately model bidirectional switches for formal verification, we make the following assumptions:

- *Control* pin of a bidirectional switch behaves similar to MOS switches. (There is no truth table given for bidirectional switches in the SystemVerilog LRM.)
- Direction of data conduction is towards the *inout* pin that has no data assigned to it.
- When more than one signal is assigned to the inout pin signal of a bidirectional switch, the value of that signal is determined by multiple driver resolution function (MDRF) of the inout pin signal. (In the rest of the paper we will use MDRF for multiple driver resolution function.)

A. Bidirectional Switch Modeling

An initial model of bidirectional switches would be to use the same approach as for MOS switches. For example, the model for *tranif0 t (p1, p2, control)* is:

$$p1 = (control === 1) \ ? \ z : p2 \tag{4}$$

$$p2 = (control === 1) \ ? \ z : p1 \tag{5}$$

But this model creates a cyclic dependency of *p1→p2* and *p2→p1*. To model bidirectional switches properly, data cannot be assigned from one *inout* pin to the other *inout* pin (as was done for MOS switches). To break this cyclic dependency, the signal assigned to *inout* pin *p1* instead is assigned to *inout* pin *p2* and the signal assigned to *inout* pin *p2* instead is assigned to *inout* pin *p1*.

To explain, consider signals *p1* and *p2* on which *a1* and *a2* are assigned respectively. For instance *tran t (p1, p2)*, the modeling is:

$$p1 = \mathrm{a1\ MDRF\ a2} \tag{6}$$

$$p2 = \mathrm{a2\ MDRF\ a1} \tag{7}$$

Similarly, instance *tranif0 t0 (p1, p2, control)* is modeled as:

$$p1 = \mathrm{a1\ MDRF\ ((control\ } === \mathrm{\ 1)\ ?\ z\ :\ a2)} \tag{8}$$

$$p2 = \mathrm{a2\ MDRF\ ((control\ } === \mathrm{\ 1)\ ?\ z\ :\ a1)} \tag{9}$$

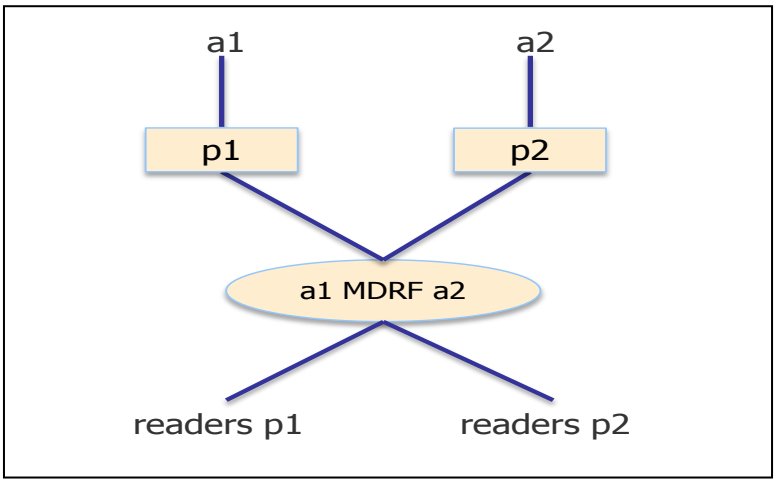

Figure 2 Bidirectional switch model

And, instance *tranif1 t1 (p1, p2, control)* is modeled as:

$$p1 = \text{a1 MDRF ((control } === \text{ 0) ? z : a2)} \tag{10}$$

$$p2 = \text{a2 MDRF ((control } === \text{ 0) ? z : a1)} \tag{11}$$

The model is extended for multiple assigns on a signal. Suppose *p1* is assigned *a*11, *a*12,…, *a*1*m* and *p2* is assigned *a*21, *a*22,…, *a*2*n* then the final value on *p1* and *p2* (respectively *a*1 and *a*2) would be the resolution of all values assigned to it:

$$a1 = \text{a11 MDRF a12 MDRF a13 ... MDRF a1m} \tag{12}$$

$$a2 = \text{a21 MDRF a22 MDRF a23 ... MDRF a2n} \tag{13}$$

When there are multiple drivers on *p1* and *p2* then in the above *tran*, *tranif0* and *tranif1* models (i.e., (6) to (11)) replace *a1* and *a2* by the corresponding MDRF value from (12) and (13) respectively.

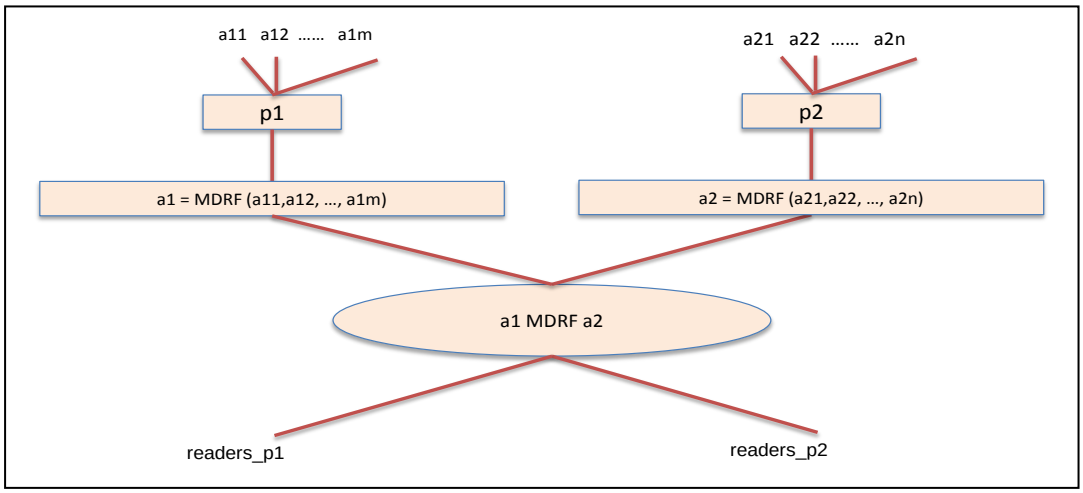

Figure 3 Bidirectional switch model with multiple drivers on p1 and p2

B. Bidirectional Switches Chain Modeling

Modeling bidirectional switches becomes complex when the signals are connected in a chain through bidirectional switches. Consider the following two bidirectional switches connected in a chain:

$$\text{tranif1 t1(p1, p2, c1)} \tag{14}$$

$$\text{tranif1 t2 (p1, p3, c2)} \tag{15}$$

Here *p2* and *p3* are not directly connected, but they would impact each other when both *t1* and *t2* are active. Suppose the signals assigned to *p1*, *p2*, and *p3* are *a1*, *a2*, and *a3* respectively. Then the model is:

$$\text{p1 = a1 MDRF ((c1 } === \text{ 0)? z : a2)MDRF ((c2 } === \text{ 0) ? z : a3)} \tag{16}$$

$$\text{p2 = a2 MDRF ((c1 } === \text{ 0)? z : a1) MDRF (((c1 } === \text{ 0) || (c2 } === \text{ 0)) ? z : a3)} \tag{17}$$

$$\text{p3 = a3 MDRF ((c2 } === \text{ 0)? z : a1) MDRF (((c1 } === \text{ 0) || (c2 } === \text{ 0)) ? z : a2)} \tag{18}$$

Next consider the case where one more bidirectional switch is in the chain along with (14) and (15)

$$\text{tranif1 t3(p2, p4, c3)} \tag{19}$$

Here *p1*, *p2*, *p3*, *p4* all are directly or indirectly impacting each other: some pins are directly connected, some are indirectly connected. Here, if *t1* is active then *p2* assigns to *p1*, if *t2* is active then *p3* assigns to *p1* and if *t1* and *t3* are active then *p4* assigns to *p1*. The final value of *p1* is an MDRF of all the assignments on it. The model for *p1*, *p2*, *p3* and *p4* is:

$$p1 = a1\ \mathrm{MDRF}\ ((c1\ ===\ 0)?\ z : a2)\mathrm{MDRF}\ ((c2\ ===\ 0)?\ z : a3)\ MDRF\ (((c1\ ===\ 0)\ ||\ (c3\ ===\ 0))\ ?\ z : a4) \tag{20}$$

$$p2 = a2\ \mathrm{MDRF}\ ((c1\ ===\ 0)?\ z : a1)\ \mathrm{MDRF}\ (((c1\ ===\ 0)\ ||\ (c2\ ===\ 0))\ ?\ z : a3)\ \mathrm{MDRF}\ ((c3\ ===\ 0)\ ?\ z : a4) \tag{21}$$

$$p3 = a3\ \mathrm{MDRF}\ ((c2\ ===\ 0)?\ z : a1)\ \mathrm{MDRF}\ (((c1\ ===\ 0)\ ||\ (c2\ ===\ 0))\ ?\ z : a2)\ \mathrm{MDRF}\ (((c3\ ===\ 0)\ ||\ (c1\ ===\ 0)\ ||\ (c2\ ===\ 0))\ ?\ z : a4) \tag{22}$$

$$p4 = a4\ \mathrm{MDRF}\ ((c3\ ===\ 0)?\ z : a2)\ \mathrm{MDRF}\ (((c3\ ===\ 0)\ ||\ (c1\ ===\ 0))\ ?\ z : a1)\ \mathrm{MDRF}\ (((c3\ ===\ 0)\ ||\ (c1\ ===\ 0)\ ||\ (c2\ ===\ 0))\ ?\ z : a3) \tag{23}$$

Adding each new bidirectional switch in the chain increases the possible number of paths and in worst case can have exponential complexity. Each new bidirectional switch added in the chain should propagate its impact to all the components in the earlier chain. In order to correctly model a bidirectional switch, the impact from all the bidirectional switches connected directly and indirectly to it should be taken into account. This is achieved by doing a traversal over the chain of back to back connected bidirectional switches and evaluating the contribution for each signal connected to the chain in terms of the other signals in the chain. Fig. 4 shows pseudo code of a generic traversal on chain of bidirectional switches to extract model for each signal in the chain. The complexity of extracting model for chain of bidirectional switches depends on how the chain has been designed. The complexity is of order of number of paths in the chain of switches.

```
For a chain of bidirectional switches  -
Let N signals are connected in a chain of M switches.

For each bidirectional switch (A, B, control) in M.
{
 expr_A = Value(A)
 expr_B = Value(B)
 propagate(A, expr_B, control)
 propagate(B, expr_A, control)
}

propagate(target, source_expr, control)
{
 cache - Cache of nodes in current path
 Insert(cache, target)
 Assign(target, source_expr, control)
 propagate_recurse(target, source_expr, control)
}

For non-conditional bidirectional switches control can be
assumed as 1
```

```
propagate_recurse(node, source_expr, control)
{
  for each switch neighbor (neigh, node, ctrl) of "node"
     if (Exists(cache, neigh))  { continue }
     else {
       new_control = merge(control, ctrl)
       Assign(neigh, source_expr, new_control)
       Insert(cache, neigh)
       propagate_recurse(neigh, source_expr, new_control)
       Remove(cache, neigh)
     }
}


Assign(target, source_expr, control)
{
  target_value_expr       = Value(target)
  new_target_value_expr = target_value_expr
                                MDRF
                                (control === false) ? z : source_expr
}
```

Figure 4 Pseudo code for model extraction of chain of switches

C. *Optimizations for Bidirectional Switches*

We have shown that adding each new bidirectional switch in the chain of bidirectional switches makes the expressions more complex. If a bidirectional switch can be optimized such that it reduces the size of the chain then it will reduce the effort required in propagation its impact. For bidirectional switches, the following optimizations are done to determine the direction of conduction and also to reduce the chain of instances to simple equations.

1) When both the inout pins of bidirectional switch are connected to constants then the switch can be ignored as constants retain their value in design, see Fig. 5, Example 1.

2) When only one of the inout pins in bidirectional switch is constant then it can be optimized to an assignment of constant pin to non-constant pin, see Fig. 5, Example 2.

3) When one of the inout pins in bidirectional switch doesn't have any assignment on it then it can be optimized to an assignement from assigned pin to un-assigned pin, see Fig. 5, Example 3.

4) When there are multiple bidirectional switch instances on same set of signals then it can be optimized to one instance with control pin as function of control pins of actual instances, see Fig. 5, Example 4.

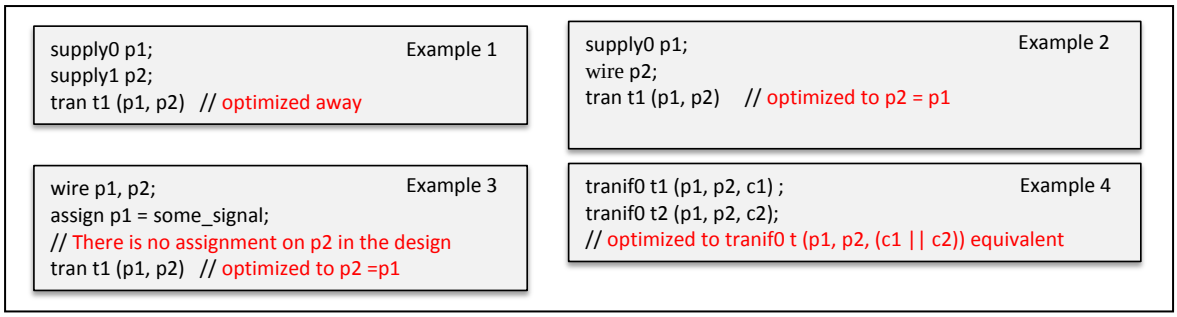

Figure 5 Optimization examples for bidirectional switches

5) Library cell modules are generated to implement basic gates using bi-directional instances only. Such cells have only one output port, other ports are of type input port and module definitions consist of bi-directional instances only. Those cells can be optimized by optimizing bidirectional switch instances invloving input and output ports of cell. Input ports can treated as read only and output ports as write only in bidirectional switches.

Fig. 6 show an example of 2-Input NAND gate implemented using a library cell of bi-directional switches and the optimized design after applying the above optimizations on this design. The 2-state truth table of this 2-Input NAND gate matches with 2-state truth table of optimized output signal, "out".

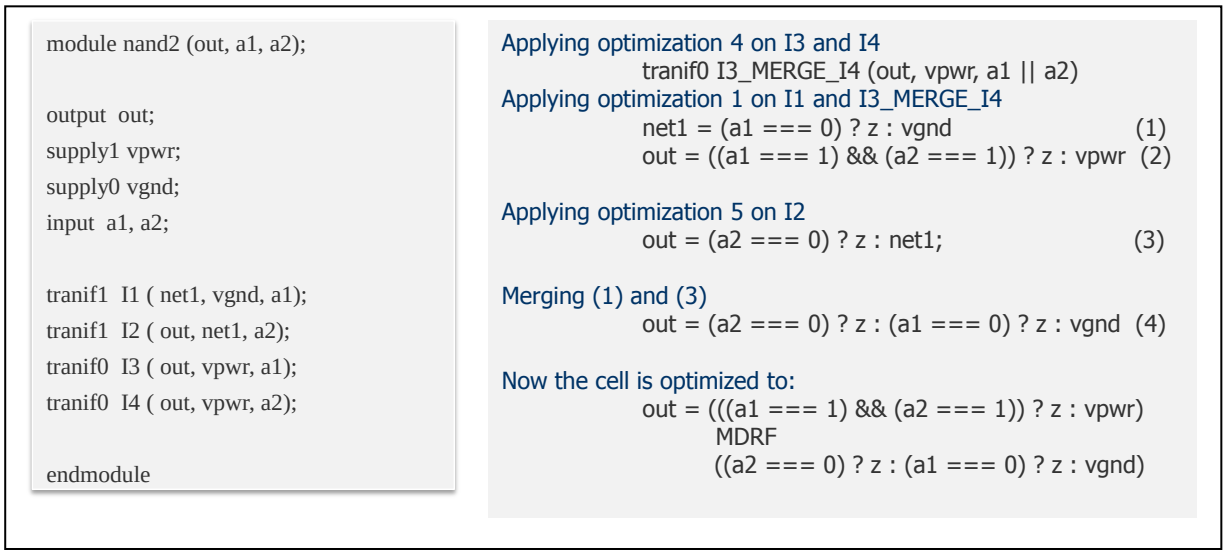

Figure 6 Optimization examples of cell implemented using bidirectional switches

IV. FUNCTIONAL ACCURACY OF EXTRACTED MODELS

The models extracted for switches need to be functionally accurate so as to get the desired behavior during formal verification. We will do a functional equivalence check of both MOS and bidirectional switches with the models extracted for them for formal verification.

A. *MOS Switches*

Functional accuracy check for MOS switches is done by comparing the truth table of the extracted model with the truth table of MOS switches specified in LRM (IEEE Std. 1800-2009, Chapter 28). Fig. 8 shows truth table of continuous assignment model generated for a pmos instance, "*pmos p1 (out, data, control)*". For our implementation L and H are considered as '0' and '1' respectively. This truth table of the continuous assignments modeled for pmos is 4-state ('0', '1', 'x' and 'z') accurate with the LRM (IEEE Std. 1800-2009, table 28.6) truth table, Fig. 7. Similar comparisons were done for nmos and cmos switches to check their equivalence with continuous assignment extracted models (2) and (3) respectively.

pmos rpmos	CONTROL				
		0	1	x	z
D	0	0	z	L	L
A	1	1	z	H	H
T	x	x	z	x	x
A	z	z	z	z	z

Symbol L represents a result that has value 0 or z
Symbol H represents a result that has value 1 or z

Figure 7 1800-2009 IEEE Std. pmos truth table

	CONTROL				
		0	1	x	z
D	0	0	z	0	0
A	1	1	z	1	1
T	x	x	z	x	x
A	z	z	z	z	z

$out = (control === 1)$? `z : data

Figure 8 pmos continuous assignment model truth table

B. *Bidirectional Pass Switches*

The functional accuracy check of bidirectional pass switches is done by comparing simulation results of design with the formal results obtained from implementation of model extraction of switches. Properties were generated for inout pins of bi-directional switches. These properties are run with the test bench created to provide all possible values to the pins of bidirectional switches. The same designs are being run with formal tool having implementation of model extraction of switches. Simulation results are matched with the formal results to check whether the extracted model is functionally correct or not.

V. FORMAL FRIENDLY MODELING

The model which we have created for switches is formal verification engine friendly. By formal verification engine friendly we mean that there is no special handling required for switches in verification engine after the switches are modeled. We have modeled switches to functionally equivalent expressions which formal verification engine already handles.

VI. DEBUG FRIENDLY MODELING

The continuous assignment model extraction of switches is done by preserving the source of bidirectional switch instances and maintaining a mapping between the extracted models with the bidirectional switch instances in the source files. This mapping helps in back annotation of models with the actual source in design and provides full visibility of the designs and debug utility for switches in the design. Fig. 8 is a snippet of Questa Static Formal schematic showing bidirectional switch, tranif0 used in design.

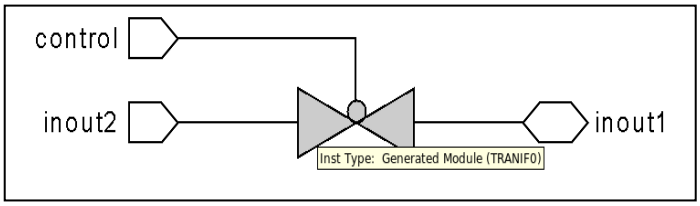

Figure 9 Snippet of tranif0 instance in QFormal Schematic

VII. APPLICATION OF MODELING

This model extraction done for designs based on switches for formal verification can be applied to applications based on formal verification, some of which are:

A. RTL vs Gate verification,

B. RTL level model checking when parts of the design are gate-level or IP blocks and

C. SoC level verification which uses formal based application for various verifications paradigms like

1) Connectivity checks
2) Security checks
3) X checks
4) Power checks
5) DFT checks

VIII. RESULTS

We internally validate our extracted models for functional correctness against simulation results of the original description. We have been able to run various real life designs through this flow and successfully apply formal techniques for verification. Table I, Design1 is an example of how modeling affects formal verification results; There were earlier incorrect firings in the design which after modeling are correctly proven by formal verification. We have been able to run a design, Table II, Design2 with around 3500 bidirectional switches in RTL and 5 million bidirectional switches after design is flattened. There was back to back chain of bidirectional switches involving around 50 signals. The reason behind such a huge use of bidirectional switches was because primary gates were implemented using switches. Table III, Design3 is an example of time taken in modeling of design with and without optimizations.

Table I. Results for design with and without model extraction of switches

Design		**Without Modeling**			**With Modeling**		
Name	***No. of bidirectional switches***	***No. of black-boxed modules***	***Properties***		***No. of black-boxed modules***	***Properties***	
			Proven	***Fired***		***Proven***	***Fired***
Design1	3	1	0	3	0	3	0

Table II. Results for design with and without model extraction of switches

Design		**Without Modeling**		**With Modeling**	
Name	***No. of bidirectional switches***	***No. of black-boxed modules***	***Coverage Exclusions***	***No. of black-boxed modules***	***Coverage Exclusions***
Design2	3529	679	154	52	1237

Table III. Time taken in modeling with and without optimizations

Design		**Time taken in modeling (seconds)**	
Name	***No. of bidirectional switches***	***Without Optimizations***	***With Optimizations***
Design3	38	60	2

IX. SUMMARY AND FUTURE WORK

In this paper, we propose a methodology to extract “formal-friendly” model for switch based circuit descriptions. We perform model extraction for various switches - tran, pmos, nmos and cmos. We describe some context based situations that lead to optimized models. As an example, we showed a complex network of bidirectional switches representing a NAND gate that is eventually extracted to a continuous assignment model representing the same functionality. Our models are efficient for formal usage and preserve the debug information

to back-annotate the results to original description. This allows verification of switch-based designs through our formal tool natively, without any specific setup requirements at the designers end.

In the future, various challenges can be addressed. Bidirectional switches make a design more complex when the switch signals are connected to pullp-pulldown sources. How can it be determined whether to propagate the first pullup-pulldown value to the other signal via the switch or to first switch the value and then propagate the source to the signal? Also, recall that in our continuous assignment modeling approach, our focus was primarily functionality where strength and delay specifications of switches were not taken into consideration. It would be interesting to study whether these specifications can impact the functionality of design.

X. ACKNOWLEDGEMENTS

We thank Arindam Chakrabarty and Yogesh Badaya, our colleagues at Mentor Graphics Corporation, India for their guidance and thoughts in discussions on extracting functionally-accurate models for switches.

XI. REFERENCES

[1] 1800-2009 - IEEE Standard for SystemVerilog -- Unified Hardware Design, Specification, and Verification Language, http://standards.ieee.org/findstds/standard/1800-2009.html

[2] 1364.1-2002 – IEEE Standard for Verilog Register Transfer Level Synthesis, http://standards.ieee.org/findstds/standard/1364.1-2002.html

[3] J. Moondanos et al., “VERTEX: Verification of transistor-level circuits based on model extraction”, Design Automation, 1993, with the European Event in ASIC Design.