Simulation Based Verification of Register-Transfer Level Behavioral Synthesis Tools

R. Ernst
Institut f. Datenverarbeitungsanlagen
Technische Universität Braunschweig
D3300 Braunschweig, FRG*

S. Sutarwala
AT&T Bell Laboratories
Allentown, PA 18103

J.-Y. Jou
AT&T Bell Laboratories
Murray Hill, NJ 07974-2070

M. Tong
AT&T Bell Laboratories
Murray Hill, NJ 07974-2070

Abstract
We present a simulation based system for verification of register-transfer level behavioral synthesis tools. Applications are tool debugging and automatic regression test. Key feature is a transformation of sequential circuits for application of pseudo-random test patterns. The results show a high relevance of verification with pseudo-random patterns.

Introduction
Part of the CAD-work in the Bell Laboratories is the development of register-transfer level behavioral synthesis tools. Reliable correctness of the software is crucial as more and more designers use behavioral circuit descriptions as design entry level. We were looking for a tool for debugging and regression test of synthesis systems. The tool to be verified was BESTMAP [1]. BESTMAP accepts a register-transfer level description of a circuit in a subset of the language "C" as input and creates a gate structure as output which is implemented with standard cells.

The first step is to check, whether formal automatic verification is feasible. Automatic formal verification of gate level circuit descriptions became popular with the advent of efficient algorithms for symbolic manipulation of binary decision diagrams [2] (improvement through efficient variable ordering see [3]). Those algorithms are only effective, if the functions are not "black boxes", but can be broken up into small subfunctions, which are individually analyzed before the results are composed.

Unfortunately, the behavioral model is such a "black box". A possible way would be to map the behavioral description to a gate level reference structure and then to verify the equivalence of synthesized and reference structure. In [5] such an approach is used to verify the equivalence of "hardware flowcharts" and gate level descriptions. In our case, however, the mapping from behavior to circuit structure itself is a major task of the synthesis system including C-model parsing, word length calculation, don't care condition detection and multiple assignment resolution. Similar mapping algorithms would be used for verification and synthesis, and, consequently, we cannot expect that the verification software is significantly more reliable than the synthesis tool.

Other verification approaches use symbolic manipulation for a direct comparison of hardware descriptions at different levels of abstraction, in some cases supported by exhaustive simulation ("hybrid simulation", see [6]; this paper also contains a brief overview of verification systems). Similar to the first approach, it would be necessary to partition the behavioral description in subfunctions requiring a C-model analysis which leads to comparable problems as mentioned above.

It seems that, at the current state, there is no practical way of automatic formal verification. We decided, therefore, to choose a simulation approach using a standard in-house mixed level simulator.

We present the simulation based verification system TSG (Test System Generator), which has been developed for the verification of register-transfer level synthesis tools. The system provides an automatic functional comparison of the input description and the output structure.

In the next section, we will take a short look at the synthesis system BESTMAP. Then, the verification system TSG is presented. The system applies random stimuli patterns for verification. In a further section, we will see how both behavioral description and synthesized structure are transformed to improve the effectiveness of random patterns. Finally, experiences using TSG in tool development are summarized.

The Synthesis System BESTMAP
As input, BESTMAP takes a register-transfer level behavioral circuit descriptions in a subset of the programming language "C". Without modification, the circuit description can be used as circuit model for our in-house simulation tools [7] [8].

There are currently two major limitations for the C-description:
- all states (registers) in the model must be controlled by the same clock. The states may be loaded asynchronously.

* The work was done while the author was with AT&T Bell Laboratories in Allentown, PA 18103
- All variables must be of integer type.

**BESTMAP** translates the C-model to an intermediate, Value Trace-like form, thereby calculating the bitwidth of automatic variables and resolving multiple assignments. This intermediate form can directly be mapped to an initial structure. Structures for more complex operations, like comparison or addition, are supplied by a module binder. Finally, the structure is resynthesized by a multi-level logic optimization tool[10]. The logic optimization tool compares the function of the structures before and after optimization with built-in verification.

The output of **BESTMAP** is a structure model based on standard cells.

**Verification System**

**Verification Function**

The verification system configuration is shown in figure 1. Behavioral model and synthesized standard cell structure are compared with simulation using the simulator MOTIS[7]. A driver applies the same input stimuli and, in case of sequential circuits, a clock signal to both of the circuit models.

Two types of stimuli patterns can be chosen:
- uniformly distributed pseudo-random patterns
- user supplied patterns.

The use of pseudo-random patterns has the advantage that the number of patterns generated can arbitrarily be extended to increase confidence in the correctness of the circuit. The pseudo-random pattern generator uses the same seed for subsequent simulation runs and, therefore, repeats the same pattern sequence. This is very useful for debugging as it assures the repeatability of error situations.

The effectiveness of comparing with pseudo-random patterns is improved by an optional model transformation for sequential circuits. This will be explained later in detail.

The output patterns of the circuit models are observed by a monitor. The monitor uses a built-in table to check, whether the output of the synthesized structure is correct. If, for example, an output signal of the behavioral circuit description is undefined (0 or 1) or is "don't care", the output of the synthesized circuit is correct for one of the values 0, 1 or "undefined", but incorrect, if it is floating (the simulator uses the same signal level for "undefined" and "don't care"). In case of discrepancies, the monitor generates an error list.

On user request, it creates output and command files for a graphic output function of the simulator which produces a logic analyzer-like signal output.

The user decides how many input patterns should be generated before the simulation stops and if the simulation should be interrupted in case the comparison detects errors in the synthesized structure.

Besides input stimuli, the driver generates a monitor trigger signal indicating when the output of the synthesized circuit is stable and should be compared with the output of the behavioral model.

Both driver and monitor are "C" language functions and are customized to the synthesized circuit under comparison. Parameters are:
- the model interface
- synchronous or asynchronous circuit
- clock control
- pseudo-random or user supplied patterns.

**Transformation of Sequential Circuits for Pseudo-Random Patterns**

In case of sequential circuits, comparison with random in-
put patterns is hardly relevant as the circuit behavior depends on the sequence of input patterns. A precise analysis of sequential circuit [11] test with random patterns showing poor performance even for small circuits, justifies this assumption.

The key function of TSG is a transformation of the two circuit models (behavioral description and synthesized structure) to models with fully controllable and observable states to improve effectiveness of random input patterns:

- In the synthesized circuit structure, the Flip-Flops are replaced by subcircuits allowing external loading (different subcircuits for different Flip-Flop types). For each state signal, an additional input and an output signal are added for external load and external observation.

- The behavioral description is enhanced by simple load statements to load and observe the state externally. Additional input and output signals are added analogous to the structure model.

- The input signals for corresponding states in behavioral description and synthesized structure are loaded with the same values.

- The driver generates "load data & state -> monitor -> clock -> monitor" cycles applying pseudo-random patterns to the input signals. The effect is, that in each cycle first data are externally loaded to the state signals, then the normal function is performed for one cycle.

The transformation is chosen such that the original models are functionally equivalent if the transformed models are.

The procedure is similar to scan techniques employed to improve circuit testability [12], with the difference that the response is additionally monitored before the clock is applied in order to check the asynchronous part of the circuit behavior in response to pseudo-random input and state patterns.

For Flip-Flop replacement, inverter swapping in logic optimization has to be regarded. If in logic optimization, a net is found which only connects an inverter output with a Flip-Flop data input, the inverter can be removed exchanging the signals at the Q and Q-bar outputs of the Flip-Flop.

Figure 2 shows an example of the Flip-Flop replacement rule for an edge triggered single data input Flip-Flop. The Flip-Flop itself is replaced by a Flip-Flop with asynchronous preset/clear of otherwise the same function. The "state load" signal enables a direct loading of the Flip-Flop from an additional primary input signal. To observe the state, a "direct state output" is added to the list of primary outputs of the synthesized structure model. In case of inverter swapping, the polarity of the state signal changes, which means that the polarity of direct loading and direct output have to change, too. Similar rules can be defined for level-triggered, preset/clear or multiple data input Flip-Flop types.

The original circuit may contain Flip-Flops with asynchronous preset/clear inputs (synthesized from asynchronously loaded states in the register-transfer level description). For those Flip-Flops, the replacement circuit contains an arbiter to prevent simultaneous preset/clear signals, which lead to an undefined Flip-Flop state. Simultaneous preset and clear signals are a possible result of random input patterns. Glitch filters are included to suppress Flip-Flop state changes caused by fake glitches in the unit delay simulation employed for comparison. In a more accurate timing simulation, these filters would be superfluous, but timing simulation needs much more run time.

The C-model transformation consists of the following steps:

1. The input signal of the C-model is extended to include signals for direct load of each of the state signals of the hardware. A signal "load state" is added to the list of primary inputs corresponding to the "model load signal" of the transformed structure model.

2. For each state variable in the C-model, a signal for observation of the state is added to the list of primary outputs.

```c
if (load_state) {
    load states through additional primary input signals
} else {
    original model body
}
write state signal values to additional primary output signals
```
3. A conditional load function is added to the C-model by surrounding the model body with an "if" clause as shown in figure 3.

Figure 4 shows the signal sequence which the driver function generates for comparison of the transformed two models. The figure assumes positive edge triggered Flip-Flops. With the inactive (here: falling) clock edge, the driver applies a new set of stimuli patterns to all inputs, except to the clock, including the inputs for a direct load of the states. The "load state" signal is set to "1" to load the Flip-Flops in the synthesized structure model and to load the state variables in the C-model. When the model load signal changes back to "0", the normal circuit operation is executed. To compare the asynchronous model functions, a monitor trigger signal is send before the rising (active) clock edge. After the rising clock edge, the comparison is repeated, now including the synchronous model functions.

The control sequence shown in the diagram is sufficient for single edge triggered circuits. The control sequence can be adapted to other clocking schemes.

A remaining problem is the identification of corresponding states in behavioral description and synthesized structure. In most cases, the state names of corresponding signals in both representations are related, allowing an unambiguous identification (this is a convention for the designer's convenience). In particular cases, however, signal names must be removed for logic optimization, e.g. when an output signal is directly connected to a state signal (our netlist language does not allow multiple names for the same net). For those cases, an alias list must be provided by the synthesis system showing the correspondence between state signals. Inverter swapping is recognized by checking the names of the signals connected to the positive and inverted outputs of a Flip-Flop and Flip-Flop substitution is performed accordingly.

Comparison System Generation and Execution

The verification system consists of:

- a comparison system generator
- a comparison system executable.

The comparison system generator reads the C-model, the synthesized structure and, if model transformation is required, the alias list generated by the synthesis system. After the optional model transformation, it creates the driver and monitor functions and builds the simulator executable. The comparison system generator is a push-button system. The user only specifies whether user supplied or pseudo-random patterns shall be applied and if the models shall be transformed. The comparison system executable runs the simulation and interacts with the user.

Results

TSG proved to be very helpful for debugging and testing of BESTMAP. To each of our about 100 test cases of different size and application, we applied 10,000 to 100,000 pseudo-random patterns. When an error was reported, the comparison was repeated with the same pseudo-random pattern sequence, this time generating a graphical signal display. The display showed input, output and state signals of C-model and synthesized structure such that the developer could see the condition which led to a circuit malfunction by looking at few signal patterns per error.

All the synthesis system errors which were found were already discovered with the first 300 patterns. This is a very encouraging result pointing out the relevance of pseudo-random patterns for this application. The result also means that we can use TSG for automatic regression test.

A possible explanation for the observed effectiveness of pseudo-random patterns is that errors in the BESTMAP synthesis process tend to have "distributed" effects affecting many cells and nets in the synthesized structure:

![Figure 4: Signal Sequence for Comparison of Transformed Models (rising edge triggered flip-flops)](image-url)

- parser and structure mapping (see figure 2) act predominantly on words rather than on bits.
- the module binder uses the same parametrized structure modules to implement functions in different places of the synthesized structure.
the logic optimizer, which among the components of BESTMAP is the only one likely to cause "single-spot" errors, uses a built-in formal verification function which compares the input structure with the resynthesized structure.

Whenever a synthesis error was found in the field test phase, it turned out that the model test set did not cover a particular synthesis feature. That means, at least in our case software test coverage is now the dominating problem. Similar experience was made with the verification of the register-transfer level synthesis capability of the system BRIDGE[13]. Formal verification would face the same problem, unless used to directly prove the correctness of the synthesis software rather than the correctness of a set of synthesized circuits. BESTMAP has meanwhile been used by many designers in the field, and an unusually high reliability for a first time tool release has been reported.

Conclusion

We presented the simulation based system TSG for verification of register-transfer level synthesis tools. The comparison approach of the system is general as it is not focused to a particular input language, it only requires that register-transfer level description and structure model can be simulated. The results show that the system delivers valid results for debugging and regression test. It was used to develop a synthesis tool, which was highly reliable from the first time it was delivered.

TSG is currently limited to single clock modules. As future work, we plan to extend the system to multiple clock designs.

REFERENCES