VERIS: An Efficient Model Checker for Synchronous VHDL Designs*

Yiping Fan, Jinsong Bei, Jinjian Bian, Hongxi Xue, Xianlong Hong, Jun Gu†
Dept. of Computer Science and Technology, Tsinghua University, Beijing 100084, China
Email: {fanyp | beijs | bianjm | xuexy | hongli}@tiger.cs.tsinghua.edu.cn
† Dept. of Computer Science, Hong Kong University of Science and Technology, Hong Kong, China
Email: gu@cu.ust.hk

Abstract
In this paper a solution for property verification of synchronous VHDL designs is introduced, and an efficient symbolic model checker is implemented. The model checker applies the feature of synchronous circuit design and the locality feature of property to reduce the state space of the internal finite state machine (FSM) model, thus speeding up the reachability analysis and property checking of circuits. A counterexample generation mechanism is also implemented. We have used the implemented model checker to verify several benchmark circuits; the experimental results contrast with another well-known model checker and demonstrate that our solution is more practicable.

Keywords
Formal Verification, Model Checker, VHDL, Synchronous Circuits, FSM

1 Introduction

Interest in formal verification techniques for hardware designs has been growing recently years. The most effective and successful formal technique is model checking [1,2], which explores the state space of designs to check if it meets some properties.

A lot of subtle research has been introduced to give formal semantics to VHDL and to apply formal verification techniques [3,10,11]. However, VHDL is a very complex language and the models that capture all the VHDL features are almost inherently not acceptable for design automation tools. So we must trade off the number of VHDL features modeled, and the practical usefulness of the semantics, for achieving more feasibility. D. Deharbe defined the operable semantics of a verification-oriented subset of VHDL, in terms of abstract machine [4]. In this model, state transition models a VHDL delta cycle. However, Deharbe’s method is still too complex, and the result of states is unreasonable when it is adopted to represent synchronous circuits. Most of the digital systems are synchronous circuits, so we should find an efficient verification method for them.

We have developed a verification system VERIS for synchronous VHDL designs. The verification system is based on symbolic model checking like Deharbe’s model. However, our approach exploits the features of synchronous circuits to propose a new FSM model, which results in reducing state spaces dramatically. By removing irrelevant portions of the circuit according to the property to be verified, we can reduce the state space and speed up the verification further. Our results show that the state machine is more reasonable, the size of state space is reduced several orders of magnitude and becomes readable, while unreadable when using Deharbe’s mixed modeling method.

To reduce the size of state model further more efficiently, we apply the locality feature of property to eliminate parts of the model that are not relevant to the specification. The model checker needs to consider only the transition functions of the variables in the cone of influence of the specification.

To make the model checker more practicable, we give counterexamples when it does not meet any properties.

In the next section, we give the outline of our model checker. In Section 3, we describe the model elaborator, which converts the synchronous VHDL design to an internal model. Then we describe the property verification in Section 4, including the elimination of irrelevant portion and counterexample generation. In Section 5, we show the experimental results, and finally, some conclusions are shown.

2 System structure

*This work was supported by Chinese National Key Fundamental Research and Development Plan (973) under Grant No. G1998030404.
3 Model elaborator

The role of the model elaborator is as the following: Given a synchronous sequential VHDL design \( D \), a finite state model \( M \) can be elaborated. The requirement put on \( M \) is that it has the same observable behavior as \( D \). The “Observable behavior” means that the response of \( M \) to the stimulus on its inputs should be the same as the response of the output ports of \( D \) to the same input values of its input ports. The behavior should be considered at the level of the clock cycle.

3.1 VHDL description for a synchronous sequential design

VHDL is a very complex language \([5]\). Some of its constructs generate infinite models and cannot be abstracted to a finite-state model, for example, an infinite loop. It is restricted to a subset of VHDL such that design descriptions can be mapped to finite state representations. That is, objects must be of a finite type (no access or file types, no unconstrained arrays, and no generics) and quantitative timing information is not accepted (any \textit{after} clauses in assignment statements, and any \textit{for} clauses in \textit{wait} statements).

We further restrict the circuit design type to synchronous circuits. A synchronous sequential circuit must have a system clock. The inputs are introduced into the circuit to process sequentially, and to generate outputs by the controlling of the system clock. That is, the external events are synchronized with the system clock. If a process or a concurrent statement is independent of the system clock, it must be a combination part. In that case, we can reduce the combination parts in the phase of elaborating the model.

3.2 Synchronized finite state model

According to the synchronous behavior, we can define a model \( M = (S, I, O, s_i, TF, OF, clk) \), called \textit{synchronized finite state model} \([10]\), where,

- \( S \) is a power of \( B \), which represents the states of the machine, \( S = B^{\text{ss}} \), and \( s_1, s_2, \ldots, s_{\text{ss}} \) are the corresponding state variables.
- \( I \) is a power of \( B \), which represents the inputs of the machine, \( I = B^{\text{si}} \), and \( i_1, i_2, \ldots, i_{\text{si}} \) are the corresponding input variables.
- \( O \) is a power of \( B \), which represents the outputs of the machine, \( O = B^{\text{so}} \), and \( o_1, o_2, \ldots, o_{\text{so}} \) are the corresponding output variables.
$s_0 \in S$, represents the initial state of the machine.

**TF:** $S \times I \rightarrow S$: $TF = \{t_{f1}, t_{f2}, \ldots, t_{fn}\}$, $TF$ represents the next state function, and $t_{fi}: S \times I \rightarrow B$ is the transition function of the state variable $s_i$.

**OF:** $S \times I \rightarrow O$: $OF = \{of_{f1}, of_{f2}, \ldots, of_{fn}\}$, $OF$ represents the output function, and $of_{fi}: S \times I \rightarrow B$ is the output function associated to the variable $o_i$.

clk is the global clock of the synchronized finite state machine.

$B$ denotes the usual Boolean domain ($B = \{TRUE, FALSE\}$).

There is an implicit synchronous clock (clk) for each $t_{fi}$ and $of_{fi}$, which means states or outputs can only be changed when the events of clk occurs.

### 3.3 Modeling algorithm

In VHDL manual, process statements are the atomic components of a design entity, and any VHDL concurrent statement except a block statement has a corresponding equivalent process statement. So we can decompose the transformation of any VHDL design unit to the synchronized model by two steps:

First, we need elaborate a sub-model for each VHDL process statement or equivalent concurrent statements in certain declaration environment. The following is the elaboration steps:

a. Determine the input variable and output variable of the sub-model.

b. Analysis every wait statements and assigned objects. Mark each signal or variable whether the system clock controls it or not.

c. Resolve data dependencies.

d. Get assignment decision diagram for every assigned object by deriving an execution tree.

e. Generate sub-models represented by BDD.

After elaborated sub-models from all process statements or equivalent concurrent statements, the next work is to compose all the sub-models to obtain the final model of the VHDL design entity. The following gives the composition mechanism:

a. Compose all of sub-models parallelly.

b. Declaration encapsulation, which means adjusting the input variables and output variables according to the ports and signals of corresponding entity declarations.

c. Reduce intermediate signals or variables. Reduce signals or variables independent of the system clock by substituting them with their assignment decision diagrams.

These steps can be explained as an example illustrated in figure 2. In the left of figure 2, the signal $b_1$ is assigned by $a_1$ and controlled by the system clock also. So we consider $b_1$ as a sequential variable and model it as an output of some transition function of FSM. While, in the right of figure 2, the output signal $b_2$ is assigned by $a_2$ and is independent of clock. So it must be a combinatorial variable and be reduced by replacing it with $a_2$.

So in the final model, all the state transitions represent the changes of state variables relating to the system clock cycle. All combination parts are abstracted to Boolean expressions and denoted by the transition functions, which allows our models to use fewer variables and fewer bits to represent one VHDL designs. So the final model yielded by above algorithm is dramatically smaller in size than that yielded by Debarre’s method \[\text{[4]}\], which observably speed up the whole model checking.

### 4 Property verification

The property verifier accepts the internal model and the property specification as inputs, and traverses the state space of the model to determine whether the property is satisfied.

#### 4.1 Temporal logic specification

The language used for specifying the expected behavior of a VHDL design is essentially the temporal logic CTL (Computation Tree Logic) \[\text{[6]}\]. A specification is composed of a set of properties about a VHDL description. It is the role of the model checker to verify whether the VHDL description satisfies the properties.

Properties are represented in computational tree logic CTL. A CTL formula is composed of a path quantifier $A$ (universal quantifier) or $E$ (existential quantifier) followed by a linear temporal formula with temporal operators $X$, $G$, $F$, $U$ and $W$. Quantifier $A$ selects all of state paths, while quantifier $E$ selects at least one state path. The linear temporal operator $X$, $G$, $F$, $U$ and $W$ selects the states of next clock cycles, all cycles, some cycles, until some cycles, and unless some cycles respectively. A formula is true for a
description if it holds in all initial states. For instance, the CTL formula $AGf$ represents that formula $f$ must hold at all states reachable from the initial states in all possible paths.

### 4.2 Irrelevant cone elimination

The final model uses a Boolean functional vector to represent the transition functions [7]. This considerably limits the explosion of the size of the transition representation for large systems. Further more, the representation also makes it easy to eliminate parts of the model that are not relevant to the specification. The model checker needs to consider only the transition functions of the variables in the cone of influence of the specification, as figure 3 shows. In figure 3, the output $o_1$ of the FMS is influenced by inputs $i_1, i_2,$ and $i_3$, but is irrelevant to the other inputs. So if a property relates only with the output $o_2$, we can only consider the cone area $(o_1, i_1, i_2, i_3)$ of the FSM state space and eliminate the rest area, thus reduce the number of states for model checking and speed up the process of verification further. And it’s the same about the case of cone $(o_m, i_m)$. A cone of influence is simply constructed from the true support set of the transition functions.

### 4.3 Property checking

Property checking is a procedure finding the set of states in the internal model where a given CTL formula is satisfied. Since the model is represented as a Boolean functional vector, the complexity of the algorithm is linear in the size of the vector and in the length of the formula. The algorithm is quite fast in practice.

First, it will traverse the model to compute the reachable states of the model, which called reachability analysis. We can use the result to further reduce the size of BDDs representing transitions.

Then it is time to check the property in the specification file by using symbolic model checking techniques. If the property is not hold, the property verifier will call the counterexample generation module [8] to produce a counterexample as a VHDL testbench. Designers can use this information to debug the design.

### 4.4 Counterexample generation

One of the most important features of CTL model checking algorithms is the ability to find counterexamples. When this feature is enabled and the model checker determines that a formula with a universal path quantifier is false, it will find a computation path, which demonstrates that the negation of the formula is true.

Noting that the counterexample for a universally quantified formula (prefix by $A$) is the witness for the dual existentially quantified formula (prefix by $E$). Our generation algorithm focus on finding witnesses for the three basic CTL operators $EX$, $EU$, and $EG$ [8].

### 5 Experimental results

We have implemented VERIS in C language on SUN-SPARC workstation. The Table 1 gives some experimental results when running in the SUN-SPARC20 with 128M memory. The first column gives the circuit name. Traffic Light Controller is obtained from [9], daisy arbiter comes from [2], and priority arbiter is obtained by removing the token ring in the daisy arbiter. The second column is the model checker system we will compare each other. CV, the model checker from CMU, applies mixed abstract machine model for both synchronous and asynchronous circuits (Deharbe method) [4]. The third column shows the CPU time for modeling. The fourth column shows the CPU time to compute the set of reachable states (reachability analysis). The fifth column shows the memory usage of every example running. The sixth and seventh columns show the number of possible states and the number of reachable states from the initial states.

From the experimental results and model analysis, we can see that the VERIS requires quite less state space and memory size than the CV system. In CV, the size of state space is unreadable. Whereas in VERIS, it is agreement with our prediction. For
example, 8-biter counter apparently has only 256 states which equal the size of state space yielded in VERIS, but in CV, the size of state space is intangible 2.15E+09.

One of the interesting examples is the 16-bit counter. Before we do irrelevant reduction, computing reachable states takes 314 seconds, and it allocated 169K memory and 1,410 BDD nodes to store the computing results. If the property to be verified just care the behavior of the lower 8 bits, it will only takes 0.36 seconds to compute reachable states, and spend 72K memory and 114 BDD nodes by using the irrelevant reduction optimization.

6 Conclusion

In this paper, we introduced a model checker for synchronous VHDL designs. The model checker applies the feature of synchronous design and the locality feature of property to reduce the state space of the internal model, thus speeding the reachability analysis and property checking of real circuits. And a counterexample generation module will invoke while some property is not hold.

In the future, we will focus on the further performance improvement. Also we will extend the VHDL subset to increase its usability.

Reference


Yiping Fan is a graduate student of Department of Computer Science and Technology, Tsinghua University, Beijing, China. He received B.S. degree from Tsinghua University in 1998 and is reading for Ph.D. as a graduate student. He is researching in the technology of formal verification, especially model checking.

Jinsong Bei graduated from Department of Computer Science and Technology, Tsinghua University, Beijing, China in 1995. Then he got Ph.D. from Tsinghua University in 1999. In course of his graduate study, his research area is formal verification of model checking and BDDs. Now he is an IT practitioner.

Jinian Bian is a professor of Department of Computer Science and Technology, Tsinghua University, Beijing, China. He has been a teacher of the department since he graduated form Tsinghua University in 1970. His research area is electronic design automation (EDA), including logic and high-level synthesis, simulation and formal verification. His current interests also include design environment of System-On-a-Chip (SOC).

Hongxi Xue is a professor of Department of Computer Science and Technology, Tsinghua University, Beijing China. He graduated form Tsinghua University in 1962 and worked here as a teacher until now. His research area is electronic design automation (EDA), including synthesis, simulation and formal verification. Now he
interests to developing a design environment of System-On-a-Chip (SOC).

Xianlong Hong graduated from Tsinghua University, Beijing, China in 1964. Since 1988, he has been a professor in the Department of Computer Science & Technology, Tsinghua University. His research interests include VLSI layout algorithms and DA systems. He is the Senior Member of IEEE and Chinese Electronics Association.

Jun Gu received B.S. degree from Chinese Science & Technology University in 1982 and Ph.D. degree from Utah University in 1989. He was the professor of Calgary University in Canada and currently he is the professor of the Department of Computer Science in The Hong Kong University of Science & technology. His research interests include the optimization algorithms, local search and global optimization, and their application in VLSI CAD, system engineering, communication and multi-media fields. He is the chief scientist of the 973 Key Foundation Research & Development Project “The Application Theories and High Performance Software in Information Technology” in China.

Table 1, Experimental Results

<table>
<thead>
<tr>
<th>Circuit Example</th>
<th>Model Checker</th>
<th>Time of Modeling(s)</th>
<th>Reachability analysis(s)</th>
<th>Memory Usage (KB)</th>
<th>State space</th>
<th>Reachable states</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic Light Controller</td>
<td>VERIS</td>
<td>0.04</td>
<td>0.05</td>
<td>117</td>
<td>8,192 (2¹¹)</td>
<td>37</td>
</tr>
<tr>
<td>CV</td>
<td>0.06</td>
<td>0.19</td>
<td>263</td>
<td>2.62E+05</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>8-bit counter</td>
<td>VERIS</td>
<td>0.03</td>
<td>0.36</td>
<td>230</td>
<td>256 (2¹¹)</td>
<td>256</td>
</tr>
<tr>
<td>CV</td>
<td>0.91</td>
<td>18.50</td>
<td>368</td>
<td>2.15E+09</td>
<td>2.241</td>
<td></td>
</tr>
<tr>
<td>16-bit counter</td>
<td>VERIS</td>
<td>0.07</td>
<td>314.99</td>
<td>3,383</td>
<td>65,536 (2¹⁶)</td>
<td>65,536</td>
</tr>
<tr>
<td>CV</td>
<td>3.54</td>
<td>6,080.90</td>
<td>86,265</td>
<td>9.22E+18</td>
<td>5.90E+05</td>
<td></td>
</tr>
<tr>
<td>4-cell daisy arbiter</td>
<td>VERIS</td>
<td>0.07</td>
<td>0.31</td>
<td>307</td>
<td>65,536 (2¹⁶)</td>
<td>1,536</td>
</tr>
<tr>
<td>CV</td>
<td>0.09</td>
<td>2,040.70</td>
<td>4,374</td>
<td>2.15E+09</td>
<td>4.18E+06</td>
<td></td>
</tr>
<tr>
<td>16-cell daisy arbiter</td>
<td>VERIS</td>
<td>2.14</td>
<td>22,870.00</td>
<td>454</td>
<td>1.85E+19 (2¹⁶)</td>
<td>8.03E+10</td>
</tr>
<tr>
<td>CV</td>
<td>7.32</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>8-cell priority arbiter</td>
<td>VERIS</td>
<td>0.05</td>
<td>0.04</td>
<td>93</td>
<td>65,536 (2¹⁶)</td>
<td>2,304</td>
</tr>
<tr>
<td>CV</td>
<td>0.13</td>
<td>8,812.20</td>
<td>558</td>
<td>8.59E+09</td>
<td>5.10E+08</td>
<td></td>
</tr>
<tr>
<td>64-cell priority arbiter</td>
<td>VERIS</td>
<td>1.62</td>
<td>17.51</td>
<td>488</td>
<td>3.40E+38 (2¹²²)</td>
<td>1.20E+21</td>
</tr>
<tr>
<td>CV</td>
<td>381.50</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

- Means data not available