FSM Modeling of Synchronous VHDL Design for Symbolic Model Checking*

Jinsong Bei, Hongxing Li, Jinian Bian, Hongxi Xue and Xianlong Hong

Department of Computer Science and Technology, Tsinghua University, Beijing 100084 P.R. China

Abstract: *In this paper, we defined a new FSM model that based on the synchronous behavior and symbolic representation technique. The algorithm to elaborate the model from the VHDL description of synchronous circuits is presented. By eliminating the unnesessary transition function, our model has much less states than Deharbe's mixed model[1]. The exprimental results demonstrate the model and modeling method can make symbolic model checking more practical.*

Keywords: *Symbolic Model Checking, VHDL, Finite State Machine*

1 Introduction

Interest in formal verification techniques for hardware designs has been growing recently years. The most effective and successful formal techniques is model checking [2], which converts the circuit to a FSM model, then explores the state space of this model completely to check the circuit if it meets some property. The approach is highly automatic: the user can simply provide a description of the circuit implementation and the property to be checked, while the system does the rest.

Using binary decision diagrams (BDD)[3] and partitioned transition relations [4] to represent the FSM model's transition graph and sets of states, Burch et al. proposed a new model checking method called symbolic model checking, which can speed up the exploration of state space dramatically. It has been applied to check an example design with approximately 5×10^{120} states [5].

In order to apply the algorithm to practical circuit design, we need transform the circuit design to a FSM model. Deharbe defined operational semantics of a verification-oriented subset of VHDL, and developed a mixed FSM model to treat synchronous and asynchronous circuits in VHDL description. In this model, a state represents a point in the simulation where the current statement of all processes stops at a *wait* statement. In comparison with the method of [6], in which a state represents a point where the current statement of each process is any statement, Deharbe's model makes great improvement in memory size.

However, Deharbe's model is still too complex, especially to represent synchronous circuits. For example, though some *wait* statements have no relation to the 'global clock' and some concurrent assignment statements are also combinational, they are all denoted by state transitions in that model. It will result an incredible verification of synchronous circuits. Thus, a further improvement on the FSM model and modeling algorithm is still imperative.

In this paper, we defined a new FSM model for synchronous circuits, and proposed a new modeling algorithm based on clock cycle. Our improvement can be found in two aspects: First, we eliminated the transition function of variables and signals that don't relate to clock from the new model; Second, we removed the signal previous value, because it is no use in our model. The principle and process of our method is organized as follows. Section 2 introduces some preliminaries, including the definition of target model – FSM. After giving the problem formulation in Section 3, we present the elaboration and composition algorithm of the new model in Section 4. Section 5 gives the complexity analysis for the two models, and some experimental results. Finally we provide some conclusion of our approach.

2 Preliminaries

2.1 Synchronous Sequential Circuits

In a synchronous sequential circuit, it must have a system clock. The inputs are introduced into the circuit to processe sequentially, and to generate outputs by the controlling of the system clock. That is, the external events are synchronized with the internal clock.

2.2 The Finite State Machine Model

Definition 1: The finite state machine is modeled by means of atomic propositions, so that is possible to process it with Boolean operations($B = \{TRUE, FALSE\}$ denotes the usual Boolean domain). According to the synchronous behavior, we can define a model $M = (S, I, O, s_0, TF, OF)$ of a synchronized finite state machine, as follows:

S is a power of B, that represents the states of the machine, $S = B^{ns}$, and $s_1, s_2, ., s_{ns}$ are the corresponding state variables.

I is a power of B, that represents the inputs of the machine, $I = B^{ni}$, and $i_1, i_2, ., i_{ni}$ are the corresponding input variables.

O is a power of B, that represents the outputs of the machine, $O = B^{no}$, and $o_1, o_2, ., o_{no}$ are the corresponding output variables.

$s_0 \in S$, represents the initial state of the machine.

TF: $S \times I \rightarrow S$: $TF = \{tf_1, tf_2, ., tf_{ns}\}$, TF represents the next state function, and tf_i: $S \times I \rightarrow B$ is the transition function of the state variable s_i.

* This work was supported by Chinese National Key Project funds under the grants 96-738-01-01-07.

OF: $S \times I \rightarrow O$: $OF = \{of_1, of_2, ., .of_{ns}\}$, OF represents the output function, and of_i: $S \times I \rightarrow B$ is the output function associated to the variable o_i.

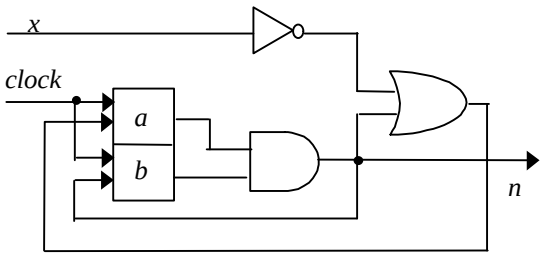
There is an implicit synchronous clock for each *tf* and *of*, that means state or output can only be changed when the clock pulses occur.

Definition 2: A machine state is represented by a unique valuation of the state variables of the model.

Any state of the machine is binary encoded into some valuation of the state variables of the model. The encoding from the set of machine states to the set of valuations is injective: two states are equal if they are represented by the same valuation.

Coudert et al. use a vector of *ns* + *no* functions of *ns* + *ni* Boolean variables, one for each different internal variable and output, this vector of functions represents both *TF* and *OF* (one function per state variable of the model) [8]. In this way, the state transition function and the output function of the finite state machine *M* was converted to Boolean function, thus we can present them with BDD.

Figure 1 shows a synchronous sequential circuit example and its FSM representation.



M = <*I*, *O*, *S*, s_0, *TF*, *OF*>, where
I = {*x*} O = {n} *S* = {*a*, *b*}
s_0 = *not a and not b*
a' = *not x or* (*a and b*) ---- *TF*
b' = *a and b* ---- *TF*
n = *a and b* ---- *OF*

Fig. 1. Synchronous circuit and its FSM representation

3 Problem Formulation

VHDL [9] is a very complex language, the models that capture all the features are almost inherently not applicable to produce design automation tools. Considering our target model, we restrict to 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 nor file types, no unconstrained arrays, no generics) and quantitative timing information is not accepted (no *after* clauses in assignment statements, no *for* clauses in *wait* statements).

In order to reduce the space size of the FSM model, we furtherly restrict the circuit design type to synchronous circuits. That is, the VHDL description must have a 'global clock' and if a process or concurrent statement has no relation to the 'global clock', it must be a combinational part. In that case, we can only care the sequential parts of the circiuts, and reduce the combinational parts in our process of elaborating the FSM model.

The meaning of modeling in this paper is as follows: Given a synchronous sequential VHDL design *D*, an FSM model *M* can be elaborated. The requirement put on *M* is that it has the same observable behavior as *D*. ′Observable behavior′ means that the response of outputs of *M* to stimuli on its inputs should be the same as the response of the output ports of *D* to values of its input ports. The behavior should be considered at the level of the clock cycle.

4 Modeling Algorithm

As we know, process statements are the atomic components of a design entity, and any VHDL concurrent statement that is not a block statement has a corresponding equivalent process statement. So we can decompose the transformation of any VHDL design unit to the FSM model by two steps:

1. Sub-FSM elaboration, that associate an FSM model to each VHDL process statement or equivalent concurrent statement within some declaration environment;

2. Sub-FSM composition, that covers the parallel composition of FSM, the declaration encapsulation on FSM, and the reduction of the intermediate signals. In the final model, one transition function presents a change of flip-flops' values in a circuit, which meas that a clock cycle has passed in the circuit. The function of all combinational parts are abstracted to Boolean expressions and denoted by the transition function, so it should be completed in the passed clock cycle time.

Section 4.1 and 4.2 gives the details of the sub-FSM elaboration and composition processes.

4.1 Sub-FSM elaboration

It is quite clear that the input, output, and state variables of the Sub-FSM that corresponds to a process statement should be elaborated from the signal read, the signals assigned, and the variables declared in the process statement. State transition and output functions should be elaborated from the assignments to variables and signlas. The initial state should correspond to the initial values of the process variables.

However, assignment statements in process are executed sequentially and thus are dependent of each other; the source expression of an assignment, as well as the condition of a conditional statement, depends on the previous variable assignment. While state transition functions and output functions occur simultaneously (parallelly) and are indeppendent in the FSM model.

The following gives the outline procedure of the sub-FSM elaboration.

1. Resolving Wait Statements.

If a *wait* statement doesn't contain 'global clock', it must be combinational *wait* statement (not a strict sequential *wait* statement), this *wait* statement has no contribution to the behavior of synchronous circuits. Thus it can be neglected when we only focus our attention on the behavior of synchronous circuits.

2. Merging Irreducible *Wait* Statements.

After deleting all the reducible *wait* statements, the process will keep the *wait* statements that contains 'global clock', or will have no *wait* statement.

In the first case, as long as a variable or signal assignment appeared in this process is related to the *wait* statement, it will be related to the 'global clock' and contribute to the state space, otherwise, the assignment is combinational. If the number of the remnant *wait* statements is more than one, we merge all the irreducible *wait* statements to one called merged *wait* statement.

In the second case, all the variables or signals will be reduced in the following section because they don't contribute to the state space.

3. Resolving Dependencies.

Distinguishing the occurrences of variables in expressions that depend on some previous assignment executed in the same simulation cycle (called reducible occurrences) from the occurrences of variables whose current value has been assigned at a previous simulation cycle (called irreducible occurrences).

4. Deriving the Execution Tree.

Transform the graph of the process transition statement into an execution tree. The root of the execution tree is the initial vertex. Each vertex on a path is either an assignment or an *if* statement, and each path in this tree represents one possible execution of the statements in one zone.

5. Getting Decision Diagrams for Each Assigned Objects.

In the simplified execution tree, assignments to objects (signals or variables) are independent of each other. Thus, in order to determine their characteristic function, it is sufficient to create, for each assigned object O, a copy of this tree and to remove all vertices that contain assignments to other objects to get a simple decision diagram that gives the value assigned to O.

6. Generating FSM model.

- Every signal or variable of a process statement part, that has not been eliminated, elaborates a implicit BDD expression from its decision diagram.
- Initial values of the elaborated signals or variables elaborate the initial state.
- Decision diagrams associated to the elaborated variables elaborate the transition function.
- Decision diagrams associated to the elaborated outputs elaborate the output function.

4.2 Sub-FSM Composition and Reduction

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

1. Parallel composition of FSM models.

$M_1=<I_1, S_1, O_1, s_{01}, TF_1, OF_1>$ and $M_2=<I_2, S_2, O_2, s_{02}, TF_2, OF_2>$ being two FSM machines, the parallel compositional M_1 and M_2 is the FSM $M = < I_1 \cup I_2, O_1 \cup O_2, S_1 \cup S_2, s_{01} \circ s_{02}, TF_1 \times TF_2, OF_1 \times OF_2>$, where $\cup$ is the union operation, and $\circ$ is the concatenation, and $\times$ is the product operator.

2. Declaration encapsulation.

For the FSM that elaborated from the statement part of the architecture body, we should adjust the FSM model according to the ports and signals of the corresponding entity declaration. Let $M_e = < I_e, S_e, O_e, s_{0e}, TF_e, OF_e>$ be the result FSM model corresponding to the entity. Then,

- I_e are the effective values and the events on the entity ports of mode *in* and *inout*, which are read in the architecture,
- O_e are variables that represent the current values of the drivers of the entity ports of mode *out* and *inout*,
- S_e are the state variables that represent the effective values of all the signals declared locally,
- s_{0e}, TF_e and OF_e are the result that augmented with the I_e, S_e, and O_e.

3. Intermediate signals reduction.

Now, not all the state variables in the macro-FSM model are related to 'global clock'. We can divide all the state variables into two classes. One are all the signals and variables that relate to the 'global clock' and do have contribution to the state space of FSM model belong to class one; the other are the signals and variables that have no relation with the 'global clock' belong to class two.

The signals or variables of class two are combinational in synchronous circuits, so if they are not output signals, we can delete them by substituting them with their decision diagrams. Through this operation, we eliminate the combinational part of synchronous circuits and move them to the output functions and transition functions.

5 Model Analysis and Experimental Results

5.1 Model Analysis

FSM model serves the basis of model checking and its size determines whether or not the formal method can be used in a wider field. Therefore, a good measure to evaluate the complexity of the model is the Boolean variables needed to encode the state variables and output variables of the model.

A VHDL design unit with input ports $i_1, i_2, \ldots, i_m$; output ports $o_1, o_2, \ldots, o_n$, in which output ports $o'_1, o'_2, \ldots, o'_l$ have relation with 'global clock'; internal signals or variables $s_1, s_2, \ldots, s_p$, in which signals or variables $s'_1, s'_2, \ldots, s'_q$ have relation with 'global clock', elaborates a binary encoded FSM model such that:

The number of input variables $|I| = m$-1,

The number of output variables $|O| = n$,

The number of state variables $|S| = q$.

$|\ |$ presents the cardinality of the set.

While in the mixed FSM model, these three complexity are: $|I| = m$, $|O| = n$, $|S| = p + prev(s)$, where the predicate *prev(s)* presents the previous value of the signal (internal signal or input signal) if it appears in a *wait* statement, otherwise *prev(s)* is null.

From the model complexity analysis, we can see that q is less p and l is less than n, and the miexed model has a item *prev(s)*, so the new model has great improvement in the number of state variable.

5.2 Experimental Results

We have applied the improvement model and its algorithm described above in our model checker for synchronous sequential VHDL design. The following table gives some experimental results, and compare with the results of synchronous and asynchronous mixed FSM model. S-Model is the FSM model of synchronous circuits, while M-Model is the mixed FSM model of synchronous and asynchronous circuits. The first column gives the circuit name. Traffic Light Controller is obtained from [10], daisy arbiter comes from [11], and priority arbiter is obtained by removing the token ring in the daisy arbiter. The fourth column shows the CPU time to compute the set of reachable states (reachability analysis). The fifth and sixth columns show the number of possible states and the number of reachable states from the initial states.

6 Conclusions

In this paper, we restrict our FSM model to synchronous circuits and utilize features of synchronous circuits to reduce the state space and BDD node number of our model. In many application examples, formal methods fail because the size of state space of FSM model runs out of memory. So although our model can only be used in the formal method of strict synchronous circuits, it decreases greatly in state space and can be used in more large-scale circuits.

Reference

[1] D. Deharbe and D. Borrione, "Semantics of verification-oriented subset of VHDL," Technical Report, School of Computer Science, CMU, 1996.

[2] R. P. Kurshan, “Formal Verification In a Commercial Setting”, In Proc. 34th ACM/IEEE DAC, 1997.

[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. On Computers C-35, 8, pp.677-691, 1986.

[4] J. R. Burch, E. M. Clarke, and D. E. Long, "Symbolic model checking with partitioned transition relations," In Proc. International Conference of VLSI, Aug. 1991.

[5] J. R. Burch, E. M. Clarke, D. E. Long, et al, "Symbolic model checking for sequential circuit verification," in IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, NO. 4, pp. 401-424, April 1994.

[6] W. Damm, B. Josko, and R. Schlor, "Specification and Validation methods for Programming Languages and Systems", chapter Specification and Verification of VHDL-based System-Level Hardware Designs, pp. 331-410. Oxford University Press, 1995.

[7] S.Bose and A.L.Fisher, "Automatic verification of synchronous circuits using symbolic logic simulation and temporal logic," In International Workshop on Applied Formal Methods for Correct VLSI Design, volume VLSI Design Methods-II, pages 151--158, 1990.

[8] O. Caudate, C. Berthed, and J. C. Madder, "Verification of sequential machines using functional vectors," In International Workshop on Applied Formal Methods for Correct VLSI Design, volume VLSI Design Methods-II, pages 179--196, 1990.

[9] IEEE, "IEEE Standard VHDL Language Reference Manual," Std 1076-1993, 1993.

[10] R. Lipsett, C. F. Schaefer, C. Ussery, "VHDL: Hardware Description and Design", Kluwer Academic Publishers, 1989.

[11] K. L. McMillan, "The SMV System," Technical Report, School of Computer Science, CMU, 1992.

Table 1 Experimental Results

Circuit Example	Model Type	Time of Modeling(s)	Time of Reachability analysis(s)	State space	Number of Reachable states
Traffic Light Controller	S-Model	0.04	0,05	8,192	37
	M-Model	0.06	0.19	2.62E+05	241
16-bit counter	S-Model	0.07	314.99	65536	65,536
	M-Model	3.54	6,080.9	9.22E+18	5.90E+05
4-cell daisy arbiter	S-Model	0.07	0.31	65536	1,536
	M-Model	0.09	2,040.7	2.15E+09	4.18E+06
64-cell priority arbiter	S-Model	1.62	17.510	3.40E+38	1.20E+21
	M-Model	381.5	-	-	-

- means data not available