Hardware Verification Using Symbolic State Transition Graphs

Pinhong Chen* Jyuo-Min Shyu+ Liang-Gee Chen*

*Dept. of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C

+Computer & Communication Research Labs.
Industrial Technology Research Institute
Hsinchu, Taiwan, R.O.C

Abstract

In this paper, a new approach for hardware verification using symbolic state transition graph (implemented in BDDs) is presented . We propose a novel idea, symbolic state transition graph (symbolic STG), which can represent FSM in terms of the relations between symbolic input variables, state variables, and output results rather than the exact input-output bit patterns. Compared to the conventional STG methods, the symbolic STG is more concise, higher-level, fewer states and easier to specify. Based on the transition relation method and an event-driven scheduling technique to compute the symbolic states, we propose two algorithms to verify the circuit implementation with respect to its symbolic STGs. The algorithms can be used to find out a necessary condition for the implementation to satisfy the specification, which can be used for the allocation of design errors.

1 Introduction

To design error-free digital circuits, VLSI designers usually have to perform simulation to verify the behavior of the implementation to detect as many design errors as possible before fabrication. However, exhaustive simulation is impractical for complex circuits and systems. Formal verification seems to provide a light on this.

Many researchers have proposed techniques to deal with hardware verification problems. In the comparison of two finite-state machines (FSMs), researchers such as [4] compute the reachable states to check whether the two machines have the same outputs. In temporal model checking methods [5], the circuit behavior is checked against some temporal properties. Another approach, symbolic simulation[6], assumes the circuit accepts symbolic inputs and produces symbolic outputs for the analysis of specified behavior [6].

In [2], the authors recognized the need of a method to verify a FSM with respect to its specification rather than to check the equivalence of two machines. For the correctness of a FSM, the design is correct if the implementation can satisfy the specification. In [2], the authors proposed a method to verify a FSM with respect to its specification, but the state transition table may be too tedious for a circuit with many registers, and it cannot specify the symbolic relations between inputs and outputs.

Unlike the comparison of two FSMs, we propose an approach, in which the specification is a state transition graph (STG) describing the state transition relations and the input-output relations with referenciable symbolic input sequence in terms of symbolic formulas represented and manipulated by BDD's. Instead of verifying one sequence of states each time in the symbolic logic simulator [6], our method can verify a symbolic FSM with any kind of transitions, as well as evaluate and check the behavior or properties specified in the specification.

In section 2, the basic idea of symbolic STG is discussed. Problem definition then follows in Section 3 along with notations and terminology. Section 4 derives a fixed-point computation method for the sets of states in symbolic STG and presents our verification algorithm. Section 5 shows our experimental results. Finally, we summarize our work.

2 Symbolic State Transition Graphs

In conventional STG methods, the behavior of a FSM is characterized by the states and the transitions between the states. However, since the STG is represented by the bit patterns of input-output variables, the number of states grows rapidly as the number of registers in the circuit increases. For instance, for a n-bit counter, we need 3×2^n transitions and 2^n states to completely specify the circuit. Furthermore, when the circuit has datapath components, it is almost impossible to completely specify with a STG. To deal with this problem, we propose to describe a STG symbolically. We specify the input-output relations in terms of boolean variables and symbolic input variables to describe what should be done in each symbolic state. Furthermore, by incorporating the concept of edge-valued BDDs [3], we can even specify the symbolic arithmetic relations in terms of boolean variables, making the symbolic STG indistinguishable from a high-level specification.

A symbolic state can represent a set of physical machine states; symbolic input sequence can be referenced (globally) by some other symbolic states, thus the declarative semantics can be achieved. As an example, consider the sequential parity checker in Figure 1(a). Suppose initially the register stores X_0 as its contents. Let the first input be X_1, the second input be X_2 and the third input be X_3, then the output is $X_0 \oplus X_1 \oplus X_2 \oplus X_3$ after the three inputs. By repeat-

1063-6404/93 $03.00 © 1993 IEEE

ing the process, we can obtain a specification of the circuit, as shown in Figure 1(b).

Symbolic STG can be just some partial specification, which can be viewed as a kind of model checking.

3 Basic Definitions

In this paper, a set is represented by boolean variables using BDDs, and we do not distinguish between the set of states represented by BDDs, the set, or the BDD's formula [4]. The set B denotes the binary set $\{0, 1\}$.

3.1 Finite State Machine

Definition A Mealy machine is a 5-tuple $< States, I, O, Next, Out >$, where $States$ is the finite set of states, I is the input alphabet, O is the output alphabet, $Next : States \times I \rightarrow States$ is the next state function, and $Out : States \times I \rightarrow O$ is the output function.

3.2 Symbolic State Transition Graph

Definition We associate a Mealy machine with a symbolic STG, in which each symbolic state denotes a set of FSM states (or a set of possible values' combination of boolean variables) represented by BDDs. Each transition is represented by input and output constraints (also represented by BDDs). We adopt the following notations:

- $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ is the vector of input variables or the input ports of a circuit.
- $\mathbf{v} = (v_1, v_2, \ldots, v_l)$ is the vector of symbolic input values, which are those symbols used to carry information across symbolic states. This kind of variables allows us to specify very useful semantics to FSMs, making possible the representation of the pre-conditions and post-conditions of a specification.
- $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ is the vector of state variables, and $\mathbf{s}^-, \mathbf{s}^+$ represent the previous and the next state vectors respectively.
- $S(\mathbf{s}, \mathbf{v}) : B^n \times B^l \rightarrow B$ is the BDD's formula to denote the on-set of the present state, and $S^-(\mathbf{s}^-, \mathbf{v}), S^+(\mathbf{s}^+, \mathbf{v})$ represent the on-sets of the previous and the next states respectively.
- $I_j(\mathbf{s}, \mathbf{x}, \mathbf{v}) : B^n \times B^m \times B^l \rightarrow B$ is the input condition of the transition j of the present state, where $1 \leq j \leq$ the number of transitions of the present state.
- $O_j(\mathbf{s}^+, \mathbf{s}, \mathbf{x}, \mathbf{v}) : B^n \times B^n \times B^m \times B^l \rightarrow B$ is the output constraint of the transition j of the present state, where $1 \leq j \leq$ the number of transitions of the present state.
- $N(\mathbf{s}^+, \mathbf{s}, \mathbf{x}) : B^n \times B^n \times B^m \rightarrow B$ is the transition relation: $N(\mathbf{s}^+, \mathbf{s}, \mathbf{x}) = \bigwedge_{i=1}^{n}(s_i^+ \Leftrightarrow Next_i(\mathbf{s}, \mathbf{x}))$

3.3 Quantification Operators

For a boolean function $f : B^m \rightarrow B$ and a vector $\mathbf{x} = (x_1, x_2, \cdots, x_m)$ over B^m, we adopt the following notations

$\exists x_i f(\mathbf{x}) \stackrel{\text{def}}{=} f(x_1, x_2, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_m) \vee f(x_1, x_2, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_m)$

$\exists \mathbf{x} f(\mathbf{x}) \stackrel{\text{def}}{=} \exists x_1 \exists x_2 \cdots \exists x_m f(\mathbf{x})$ and,

$\forall x_i f(\mathbf{x}) \stackrel{\text{def}}{=} f(x_1, x_2, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_m) \wedge f(x_1, x_2, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_m)$

$\forall \mathbf{x} f(\mathbf{x}) \stackrel{\text{def}}{=} \forall x_1 \forall x_2 \cdots \forall x_m f(\mathbf{x})$

3.4 Transition Relation Method

We use transition relation method [4, 5] for computing the next state set: $S^+(\mathbf{s}^+) = \exists \mathbf{x} \exists \mathbf{s} \{\bigwedge_{i=1}^{m}(s_i^+ \Leftrightarrow Next_i(\mathbf{s}, \mathbf{x})) \wedge S(\mathbf{s})\}$.

4 Fixed-Point Computation Formulas and Verification Algorithm

From the semantics of symbolic STGs, three types of predicates can be formulated, from which the iteration formulas can be derived for our event-driven verification algorithm.

4.1 Semantics Predicates

According to the semantics of STG, the following 3 predicates are tautology:

Predicate (1):

$\forall j \forall \mathbf{x} \{S(\mathbf{s}, \mathbf{v}) \wedge I_j(\mathbf{s}, \mathbf{x}, \mathbf{v}) \wedge N(\mathbf{s}^+, \mathbf{s}, \mathbf{x}) \rightarrow O_j(\mathbf{s}^+, \mathbf{s}, \mathbf{x}, \mathbf{v})\}$,

where $1 \leq j \leq$ the number of transitions of the present state. This predicate means that the output constraint $O_j(\mathbf{s}^+, \mathbf{s}, \mathbf{x}, \mathbf{v})$ will be satisfied, when the symbolic STG is in the present state, and the input condition is satisfied.

Predicate (2):

$\forall j \forall \mathbf{x} \forall \mathbf{s}^+ \{S(\mathbf{s}, \mathbf{v}) \wedge I_j(\mathbf{s}, \mathbf{x}, \mathbf{v}) \wedge N(\mathbf{s}^+, \mathbf{s}, \mathbf{x}) \rightarrow S_j^+(\mathbf{s}^+, \mathbf{v})\}$.

This predicate means that the next state constraint $S_j^+(\mathbf{s}^+, \mathbf{v})$ of the transition j will be satisfied if the present state constraint $S(\mathbf{s}, \mathbf{v})$ is true and the input condition is satisfied.

Predicate (3):

$\forall k \forall \mathbf{x} \forall \mathbf{s}^- \{S_k^-(\mathbf{s}^-, \mathbf{v}) \wedge I_k^-(\mathbf{s}^-, \mathbf{x}, \mathbf{v}) \wedge N(\mathbf{s}, \mathbf{s}^-, \mathbf{x}) \rightarrow S(\mathbf{s}, \mathbf{v})\}$,

where $1 \leq k \leq$ the number of incoming transitions of the present state. This predicate means that the present state constraint $S(\mathbf{s}, \mathbf{v})$ will be satisfied if the previous state constraint $S_k^-(\mathbf{s}^-, \mathbf{v})$ is true and the input condition of the entering transition is satisfied.

4.2 Derivation of Iteration Formulas

Since it is not known what the set of states is for each symbolic state before verification, we cannot do tautology checking for the above predicates. Instead, we try to derive the constraint or the set of states for each symbolic state under every transition relation and output relation. Our method can be summerized as follows[9]:

First, since we represent true as 1 and false as 0, and with $\mathbf{s}, \mathbf{v}$ universely quantified, we rewrite Predicate (1) for each state as:

$S(\mathbf{s}, \mathbf{v}) \subseteq \neg \bigcup_j \exists \mathbf{x} \exists \mathbf{s}^+ \{I_j(\mathbf{s}, \mathbf{x}, \mathbf{v}) \wedge N(\mathbf{s}^+, \mathbf{s}, \mathbf{x}) \wedge \overline{O_j(\mathbf{s}^+, \mathbf{s}, \mathbf{x}, \mathbf{v})}\}$.....(1)

Second, for each state, we have, from Predicate (2)

$S(\mathbf{s}, \mathbf{v}) \subseteq \neg \bigcup_j \exists \mathbf{x} \exists \mathbf{s}^+ \{I_j(\mathbf{s}, \mathbf{x}, \mathbf{v}) \wedge N(\mathbf{s}^+, \mathbf{s}, \mathbf{x}) \wedge \overline{S_j^+(\mathbf{s}^+, \mathbf{v})}\}$.....(2)

For each state, Predicate (3) can be written as $S(\mathbf{s},\mathbf{v}) \supseteq \bigcup_k \exists \mathbf{x} \exists \mathbf{s}^- \{S_k^-(\mathbf{s}^-,\mathbf{v}) \wedge I_k^-(\mathbf{s}^-,\mathbf{x},\mathbf{v}) \wedge N(\mathbf{s},\mathbf{s}^-,\mathbf{x})\}$.....(3)

4.3 Algorithm for Computing Symbolic States

The algorithm finds in each symbolic state any possible combination of binary variables' values, which are represented by BDD's nodes. Our event-driven verification algorithm is based on the updated transitions to recompute each symbolic states, and converge if none of the symbolic state is changed.

Combining Equations (1) and (3), the forward method of event-driven algorithm can be described as follows:

foreach$(S(\mathbf{s},\mathbf{v}))\{$
  $I/O\_constraint(\mathbf{s},\mathbf{v}) \Leftarrow \neg \bigcup_j \exists \mathbf{x} \exists \mathbf{s}^+ \{$
    $\overline{I_j(\mathbf{s},\mathbf{x},\mathbf{v}) \wedge N(\mathbf{s}^+,\mathbf{s},\mathbf{x}) \wedge O_j(\mathbf{s}^+,\mathbf{s},\mathbf{x},\mathbf{v})}\}$
  $S(\mathbf{s},\mathbf{v}) \Leftarrow$ Initial sufficient conditions$\wedge$
    $I/O\_constraint(\mathbf{s},\mathbf{v})$
$\}$
Initialize E_list for event-driven computation.
$E\_list \Leftarrow$ all the transitions starting from the initial symbolic states.

foreach(transition $S^- \xrightarrow{I^-} S$ in E_list)$\{$
  $S(\mathbf{s},\mathbf{v}) \Leftarrow S(\mathbf{s},\mathbf{v}) \vee [\exists \mathbf{x} \exists \mathbf{s}^- \{S^-(\mathbf{s}^-,\mathbf{v}) \wedge I^-(\mathbf{s}^-,\mathbf{x},\mathbf{v}) \wedge N(\mathbf{s},\mathbf{s}^-,\mathbf{x})\} \wedge I/O\_constraint(\mathbf{s},\mathbf{v})]$
  if ($S(\mathbf{s},\mathbf{v})$ has been changed)
    Schedule all the transitions starting from S into E_list.
$\}$ **until** E_list is empty.
if (Any of $S(\mathbf{s},\mathbf{v})$ is empty) Report failure in verifying implementation.
else Report success with each symbolic state set $S(\mathbf{s},\mathbf{v})$.

The $S(\mathbf{s},\mathbf{v})$ obtained after the algorithm terminates is a necessary condition for the implementation to satisfy the specification. It represents the possible combination of the values of each state variable and each symbolic input in that symbolic state. This information can be analyzed to check whether the design is correct, or to allocate the logical errors in the design. If any symbolic state is found to be an empty set, the implementation fails to carry out the function specified in the symbolic STG.

Combining Equations (1) and (2), a backward method of event-driven algorithm can be obtained[9].

4.4 Localizing Datapath State Variables

When $S(\mathbf{s},\mathbf{v})$ is independent of a subset $\mathbf{s}_{reg} \subseteq \mathbf{s}$, we can apply the universal quantification on $\mathbf{s}_{reg}$ to the state constraint $S(\mathbf{s},\mathbf{v})$ [9].

Predicate (1) can be rewritten as:
$\forall j \, \forall \mathbf{x} \, \forall \mathbf{s}^+ \, \forall \mathbf{s}_{reg} \{S(\mathbf{s},\mathbf{v}) \wedge I_j(\mathbf{s},\mathbf{x},\mathbf{v}) \wedge N(\mathbf{s}^+,\mathbf{s},\mathbf{x}) \rightarrow O_j(\mathbf{s}^+,\mathbf{s},\mathbf{x},\mathbf{v})\}$
And, by defining $\mathbf{s}_{ctrl} \equiv \mathbf{s} - \mathbf{s}_{reg}$, we can obtain : $S(\mathbf{s},\mathbf{v}) = S(\mathbf{s}_{ctrl},\mathbf{v}) \subseteq \bigcap_j \neg \exists \mathbf{x} \, \exists \mathbf{s}^+ \, \exists \mathbf{s}_{reg} \{I_j(\mathbf{s},\mathbf{x},\mathbf{v}) \wedge N(\mathbf{s}^+,\mathbf{s},\mathbf{x}) \wedge \overline{O_j(\mathbf{s}^+,\mathbf{s},\mathbf{x},\mathbf{v})}\}$. The subsequent computation of transition relation excludes those variables in $\mathbf{s}_{reg}$. Therefore, $N(\mathbf{s}^+,\mathbf{s},\mathbf{x})$ can be replaced by $N(\mathbf{s}^+,\mathbf{s},\mathbf{x}) = N(\mathbf{s}^+_{ctrl},\mathbf{s},\mathbf{x}) = \bigwedge_{\mathbf{s}_i \in \mathbf{s}_{ctrl}} (\mathbf{s}_i^+ \Leftrightarrow Next_i(\mathbf{s},\mathbf{x}))$

4.5 Some Useful Heuristics

To improve the efficiency of verification using symbolic STG, we have proposed some techniques in our implementation [9]. For instance, the scheduling techniques can reduce the iterations. In general, the scheduled transitions which will cause the same $S(\mathbf{s},\mathbf{v})$ to update are scheduled successively. This technique will avoid the algorithms to skip between two symbolic states, and the algorithms can then terminate in the fewer iterations.

One crucial factor for the performance of BDD's package is the variable ordering [1, 4]. We interleave the support of variables and the next state variables $\mathbf{s}^+$ as [4, 5]. We note that, in practice, the transition relation method will result in an intractable size of BDDs even for small circuits [7, 5, 4]. The partitioned transition relation method in [5] and the balanced and-tree product method in [4] emphasized the sequence of the conjunction of each product term, and to smooth variables as soon as possible. In the domain partitioning and co-domain partitioning methods recursive algorithms are implemented to compute transition relation [7]. These techniques are also applied to our approach for each iteration formula.

5 Experimental Results

We have implemented our algorithm using the SIS package of U.C. Berkeley [8] and tested on many circuits successfully on Sun Sparc 2 workstation.

We show two cases here for discussion purposes. For other cases and more examples, see [9].

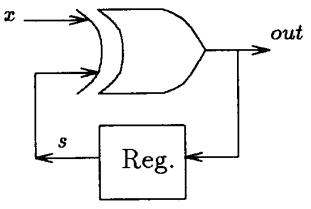
5.1 Simple Sequential Parity Checker

This parity checker has an exclusive-or gate and a register as shown in Figure 1(a), with a specification shown in Figure 1(b). The specification can be written in text form as:

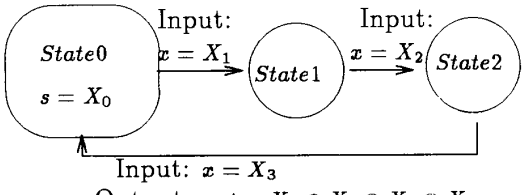
```
1:  state S0 ( s = X0 )
2:   if( x = X1 )goto S1
3:  state S1
4:   if( x = X2 )goto S2
5:  state S2
6:   if( x = X3 ){
7:       output out = X0 ⊕ X1 ⊕ X2 ⊕ X3
8:       redef(X0, X1, X2, X3)
9:       goto S0 }
```

where $\mathbf{s} = X_0$ in line 1 represents the initial sufficient condition for state $S0$, and the redef function in line 8 specifies the variables to be reassigned symbolic values. Our algorithm obtains the following sets of states for each symbolic state:
$S0$: $\mathbf{s} \Leftrightarrow X_0$
$S1$: $\mathbf{s} \Leftrightarrow X_0 \oplus X_1$
$S2$: $\mathbf{s} \Leftrightarrow X_0 \oplus X_1 \oplus X_2$
The result shows that the state variable s carries the information we want in each state.

If we mistook the specification, for example, so that line 7 were changed to output $out = X_0 \oplus X_1 \oplus X_3$, we



(a) Circuit Implementation



Output: $out = X_0 \oplus X_1 \oplus X_2 \oplus X_3$

(b)Symbolic states specification

Figure 1: Simple sequential parity checker

could obtain the state constraint for $S2(s, X_0, X_1, X_2)$ as $\overline{X_2}(s \Leftrightarrow X_0 \oplus X_1)$. It shows that if X_2 is equal to 0, the implementation still can satisfy the specification.

5.2 Greatest Common Divider Circuit

The greatest common divider(GCD) computation circuit consists of a datapath and a control unit. We will show how to localize register elements and the efficiency it can achieve. The GCD circuit is implemented by a successive subtraction algorithm. We use gcd$n$ to represent a gcd circuit with two n-bit registers: one is x and the other is y, and so are other circuits. We assign these two registers to be local state variables. The remaining state variables are control registers from $s0$ to $s5$.

The following table summaries the case for several GCD circuits *with* assigning local state variables x and y.

circuit	# of variables			CPU time	BBD
	state	reg.	PI	(seconds)	nodes
gcd8	6	16	9	3	4819
gcd10	6	20	11	5	6633
gcd12	6	24	13	8	7941
gcd14	6	28	15	23	9154
gcd16	6	32	17	110	12329

The following table is for the case *without* assigning local state variables.

circuit	# of variables		CPU time	BBD
	state	PI	(seconds)	nodes
gcd8	22	9	763	17071
gcd10	26	11	902	23433
gcd12	30	13	1121	60025
gcd14	34	15	NA	NA
gcd16	38	17	NA	NA

6 Conclusion

In this paper, we describe a novel idea, the symbolic STG, which utilizes the BDD's symbolic expressive power to efficiently represent a FSM. Also, we present the related algorithms to verify a circuit implementation with a symbolic STG. Experimental results have shown that it has great advantage over the conventional methods.

References

[1] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, C-35, No. 8, pp. 677-691, 1986.

[2] S. H. Hwang and A. R. Newton "An Efficient Verifier for Finite State Machines," *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 3, pp. 326-334, 1991.

[3] Y. T. Lai and S. Sastry, "Edge-Valued Binary Decision Diagrams for Multi-Level Hierarchical Verification," *Proc. of 29th Design Automation Conf.*, pp. 608-613, 1992.

[4] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton and A. Sangiovanni-Vincentelli, "Implicit state Enumeration of Finite State Machine using BDD's," *Proc. of Int. Conf. on Computer Aided Design*, pp. 130-133, 1990.

[5] J. R. Burch, E. M. Clarke and D. E. Long, "Representing Circuits More Efficiently in Symbolic Model Checking," *Proc. of 28th Design Automation Conf.*, pp. 403-407, 1991.

[6] R. E. Bryant and C. H. Seger, "Formal Hardware Verification by Symbolic Ternary Trajectory Evaluation," *Proc. of 28th Design Automation Conf.*, pp. 397-402, 1991.

[7] O. Coudert and J. C. Madre, "A Unified Framework for the Formal Verification of Sequential Circuits," *Proc. of Int. Conf. on Computer Aided Design*, pp. 126-129, 1990.

[8] E. M. Stentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton and A. Sangiovanni-Vincentelli ,"SIS: A System for Sequential Circuit Synthesis," *Electronics Research Laboratory Memorandum*, No. UCB/ERL M92/41, 1992.

[9] P. Chen, "Hardware Verification Using Symbolic State Transition Graphs," *Master Thesis, Dept. of Electrical Engineering, National Taiwan University, R.O.C.*, June, 1993.