FORMAL VERIFICATION OF FINITE STATE MACHINES

Unni Chandran[1] **D. Kuldeep**[1] **V. Sahula**[2]

Abstract

The classical methods of design verification of a digital design had been simulation and testing. Due to constraint of time, simulation based testing approaches use only a small sub set of input patterns whereas formal verification allows all input combinations to be taken into consideration. Model checking is one of the approaches of formal verification. The design description, modeled using the model verifier script, allows specification of some properties of the design using temporal logic. A synchronous sequential design can always be described as a finite state machine (FSM). The formal verification of an FSM is based on analysis of set of reachable states of the corresponding finite state automata. For large systems, state space explosion is main problem to deal with. Instead of describing large FSMs as flat description, we propose to describe FSM after abstracting non-essential group of states of the given FSM. This abstraction is a manual process and is guided by CTL formulae to be checked. We have used symbolic model verifier, public domain software, available from Cadence Berkeley Labs USA. We evaluate our proposal for some example circuist along with an 8-bit microprocessor.

1. Introduction

The classical methods of design verification of a digital design had been simulation and testing. Simulation based testing requires application of exhaustive input patterns to ensure functional correctness of the design. With the increasing demand for short time to market and larger complexity of the circuits, only sub set of input patterns is applied which is selected such that it ascertains a known level of fault coverage. This may lead to some errors in design remaining undetected and possible functional failure of the chip later. Formal verification allows all input combinations to be taken into consideration. Theorem proving and model checking are approaches popularly followed during formal verification. Theorem proving is less amenable to automation, whereas model checking allows high degree of automation for its algorithms.

In Section 2, we discuss briefly about various verification approaches for combinational and sequential circuits. A synchronous sequential circuit can always be described as a finite state machine (FSM). Section 3, contains discussion as how to describe and verify finite state machines. We propose to describe large FSMs as hierarchical description to a model checker, instead of a

[1] Undergraduate students, Department of ECE, Malaviya Regional Engineering College, Jaipur. E-mails: unni2805@rediffmail.com, mailtokuldy@yahoo.com

[2] Faculty member, Department of ECE, Malaviya Regional Engineering College, Jaipur. E-mail: sahula@ieee.org

flat description. We evaluate our proposal for 8-bit baby instruction set microprocessor design verification. We discuss results in Section 4 for hierarchical description of FSMs corresponding to various circuits, from small size circuits to microprocessor design. Conclusions are given in Section 5.

2. Verification approaches

There exists many approaches for formal verification of digital circuits both combinational (data path circuits) and sequential (controlling circuit) circuits. In the following section we provide brief discussion of some of the approaches.

2.1 Combinational equivalence

To check equivalence of two combinational circuits or Boolean functions, binary decision diagram (BDD) based approach is a convenient and natural choice. An ordered BDD (OBDD) for an output of a circuit or function, is always canonical for a given order of input variables, refer Bryant (1986). Hence, equivalence of OBDDs corresponding to two different circuits implies equivalence of the circuits. In practice, this approach is limited by the size of BDD generated for the circuits under consideration. The size of a BDD is highly dependent on the function being verified and the variables' order used.

2.2 Symbolic simulation

Symbolic simulation is a combination of BDD based verification approach and logic simulation. Alternately, logic simulator provides detailed timing analysis, hazards and oscillatory behavior simulation but only one simulation vector can be applied at a time. A 30 input circuit may require over a billion input vectors. Symbolic simulation provides two conveniences.

?? It is 3-valued simulation. Apart from logic value 0 and 1, a signal can take on logic value X that represent an unknown value. Resulting signal computation can be done as follows. For a 2-inout gate (AND/OR), if one of the input is X and the other a non-controlling value, the output would be X; whereas with the presence of controlling value on one of the input results in that controlling value appearing at output, irrespective of the value of the other input. Thus, setting an input to X gives effect of considering two values 0 and 1 simultaneously.

?? Simulator also accepts symbolic values for inputs and signals. This means instead of setting an input value to a constant like 1, 0 or X; its value is set to a symbolic value.

To implement these proposals, a logic simulator if modified to use BDDs to represent the values on wires as a function of symbolic values, i.e. for each of the wire the value can be represented by a function and in turn by a BDD. Number of required simulation runs is halved when an input is set to a symbolic value, but it makes BDDs representing values on wires larger.

2.3 Sequential equivalence

Sequential circuits are often modeled as finite state machines (FSMs). A typical equivalence problem maps to a comparison whether two state machines have identical behavior. This is equivalent to finding a product FSM of the two given FSMs and computing set of reachable states. Let us consider two FSMs, each of them having single output. In each of the state of product FSM, the output is Ex-OR of outputs of individual machines. For equivalence of the two FSMs, the output in each of the states of the product machine must be zero. There are three

stages during reachability analysis using BDDs: (i) Representing set of states using BDDs, (ii) Image computation and (iii) Performing iterations to obtain set of reachable states. A Boolean variable is associated with each of the latch in a given sequential circuit. A BDD now can be used to represent set of states. This BDD can be thought of as representing a set of truth assignments (state assignment): if function represented by BDD is true for a truth assignment, that assignment is there in the set; otherwise, if function is false, that assignment is not in the set. Let us consider a circuit with 3 latches assigned three variables x_2, x_1, x_0. For this example, $f_1=x_2?x_1$ represents the set containing truth assignments {110,111} whereas $f_2=1$ represents all eight truth assignments {000,001,010,011,100,101,110,111}.

Given a BDD that represents a set of states of a state machine, the image BDD is a new BDD that represents the set of all possible states that machine could enter after exactly after a clock cycle. This image BDD can be build as follows: a BDD is built that represents the relationship between the present and next values of the latches. This BDD is called transition relation. For a sequential circuit having single input *i*, x_0, x_1, x_2 as outputs of latches and y_0, y_1, y_2 as inputs of latches, the transition relation will be a BDD representing $(y_0?f_1(x_2, x_1, x_0,i))\ ?\ (y_1?f_2(x_2, x_1, x_0, i))\ ?\ (y_2?f_3(x_2, x_1, x_0, i))$. This transition relation is is ANDed with the BDD whose image is to be computed; and then variables for the present state and inputs are existentially quantified out[?].

A procedure to iteratively find images is pursued till the set of reachable states is achieved. In very first iteration, we start with 'reset' state and then in successive iterations, compute images to get the set of reachable states in one clock cycle.

If, N_0 :? BDD for reset state, then it follows that

$$N_1 :?\ N_0\ ?\ image?N_0?$$
$$N_2 :?\ N_1\ ?\ image?N_1?$$
$$N_3 :?\ N_2\ ?\ image?N_2?$$

$$N_{i?1} :?\ N_i\ ?\ image?N_i?$$

This approach is limited by the size of BDDs generated.

2.4 Model checking of sequential circuits

Model checking is an alternate method of verifying state machine characteristics. Instead of pursuing evaluation of set of reachable states or product machine based equivalence checking of FSMs; model checking provides for checking of certain temporal properties. Temporal logic is a formal way of expressing properties that change with time. There are various kinds of temporal logic; Computational tree logic (CTL) is one of them, which is used for model checking. CTL is a prepositional logic of branching time, i.e. it is based on prepositional logic and uses a discrete model of time where at each instant, time

[?] $?x.f$ called existential quantification, is true when there is a value of *x* that makes *f* true. $?x.f\ ?\ f_x\ ?\ f_{?x}$, where $f_x\ ?\ f_{x?1}$ and $f_x\ ?\ f_{x?0}$.

may split into more than one possible future. Symbolic model checking uses BDDs in model checking algorithms. The reachability-based algorithms enumerated in Section 2.3 are used in symbolic model checking in their generalized form. Let P be a set of atomic properties. CTL formulas may then be defined recursively, as follows.

(a) Every atomic proposition $p \in P$ is a CTL formula.

(b) If f_1 and f_2 are CTL formulae, so are $\neg f_1$, $f_1 \wedge f_2$, AXf_1, EXf_1, $A[f_1 U f_2]$ and $E[f_1 U f_2]$.

A and *E* are "universal" and "existential" quantifiers, respectively. F, X, and G refer to "eventuality", "next time step" and "at all times", respectively. Intuitively, AX means "all successors", EX means "there exists a successor". The formula $A[f_1 U f_2]$ means "always until" i.e. along all possible f_1 holds until f_2 is satisfied. Similarly, $E[f_1 U f_2]$ means "exists-until" i.e. there exists a path such that f_1 holds until f_2 is satisfied. Following are also true.

$$AG\, f \equiv \neg EF \neg f \; ; \; EG\, f \equiv \neg AF \neg f$$

The two important classes of properties, which are usually desired to be verified are- *safety* property, *liveness* property. Safety properties are properties, which intuitively assert, "bad things never happens". Properties in liveness class state that "good things happen eventually". They are also referred to as *eventuality* or *progress* properties.

3. Model checking FSMs

There have been a number of approaches for verifying sequential circuits by complete state space. State exploration based verification methods are more amenable to automation than theorem proving. State exploration methods can easily produce a counter example trace that helps in finding errors in the circuit. A theorem prover only indicates that proof cannot be through without mentioning whether this was because of circuit error or a weakness in theorem proof. Although state exploration methods are automatic, a user is required to do more than just providing a specification and a circuit description.

(i) When using BDD-based unification methods, a good variable ordering must be chosen in order to have reduced size BDDs.

(ii) Choosing one of the many techniques in literature such as reachability analysis, sequential equivalence and CTL model checking, user starts with an abstracted model of the circuit. A method is expected to work well on full circuit as well when it performs better on a smaller (abstracted) version of the circuit.

(iii) User must also provide a partitioning table of the transition relation. One of the ways is to have one partition for each state variable of the circuit. Knowing the hierarchical structure of the circuit, verification process speed up by combining some of the partitions.

(iv) As model checking does not support hierarchical finite state machine description, user has a definite role here. One of the ways, a hierarchical finite state machine description of a system is verified is to flatten out the description. But these results in state space blow up, as total number of states would be multiplication of number of states of at each level of hierarchy.

There are two alternatives- (I) Developing algorithms to enable model checker accept hierarchical finite state machine descriptions and then integrating it with model checking tool. (II) Manually model checking each level of hierarchy separately. This requires formulation of CTL formulae for each level of hierarchy. For a large system, when described as flat FSM behavior, we propose a complementary approach to (ii). Flat FSMs descriptions can be partitioned and one or more groups of atomic states can be abstracted to respective superstates and model checking performed on abstracted version of FSM. This partition, of course should be allowed by the CTL formula to be checked. Intuitively, this should result in reduced verification time. We evaluate the effect of abstraction of an FSM behavior on verification time.

4. Model checking hierarchical FSMs

We consider example description of flat finite state machine. After abstracting out certain groups of states into superstates, we evaluate verification time for flat and hierarchical descriptions. We check for various properties related to safety and liveness classes. We observe large reduction in time and number of BDD nodes used with hierarchical description. State transition graph corresponding to one of the circuits considered is shown in Figure 1. The FSM graph corresponds to a traffic light controller.

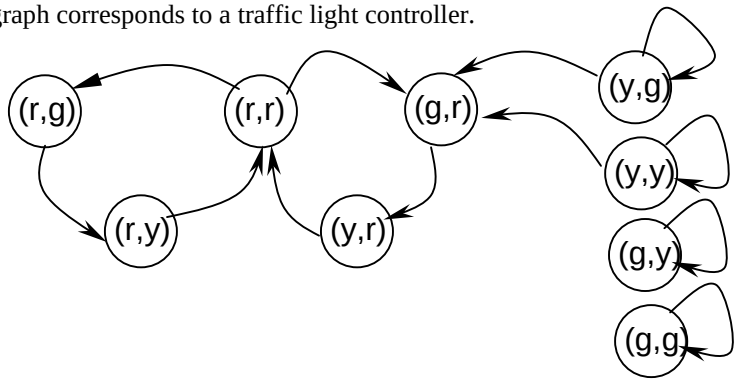

Figure 1 Transition graph (FSM) for traffic light controller

Table 1 Comparing hierarchical v/s flat description

Example Circuit	*Flat Description*		*Hierarchical*	
	# of BDD nodes	*Time*	*# of BDD nodes*	*Time*
Mod-5 counter	411	0.065	395	0.035
Sequence Detector	164	0.08	43	0.01

Table 2 Hierarchical FSM verification- Traffic light controller

Example Circuit	*Flat Description*		*Hierarchical-1*		*Hierarchical-2*	
	# of BDD nodes	*Time*	*# of BDD nodes*	*Time*	*# of BDD nodes*	*Time*
Traffic light controller	247	0.125	52	0.125	54	0.095

The results of verification on "symbolic model verifier" are presented in Table 1. Although these results are for some example circuits, nevertheless they convey the message that when the approach is scaled to large and complex FSM descriptions, significant reduction in verification time can be expected. The results of verifying traffic light controller FSM of are given in Table 2. Figure 2 compares results graphically for number of BDDs and time taken during verification of this circuit. The work has been extended to an 8-bit mini instruction set microprocessor where we describe the control specifications as a hierarchical FSM. Readers are referred to Alur et al (2000) for further reading on model checking of state machines.

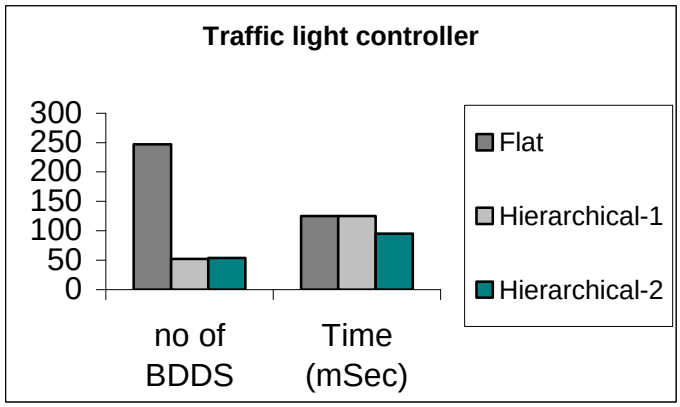

Figure 2 Verification of Flat vs hierarchical FSM for traffic controller

5. Conclusions

We have proposed that during verification of an FSM, it better be described as hierarchical FSM, abstraction of which is guided by the CTL formulae to be checked. For embedded systems, dominated by control logic, this would help in reducing verification time substantially.

References

Alur, R. and Yannakakis, M. (2000). Model checking of hierarchical state machines, *ACM Trans. on Programming Languages and Systems*, pp 1-31.

Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation, *IEEE Trans. On Computers*, Vol. C-35, No. 8.

Burch, J. R., Clarke, E. M., Long, D. E., McMillan, K. L., Dill, D. L., (1994). Symbolic model checking for sequential circuit verification", *IEEE Trans on Computer –Aided Design*, vol.13, no.4, pp.401-424, April 1994

Gupta, A. (1992). Formal Hardware Verification Methods: A Survey, *Formal Methods in System Design*, vol. 1, pp. 151-238.

Kern, C. and Greenstreet, M. R. (1999). Formal verification in hardware design: A survey, ACM Trans. on Design Automation of Electronic systems, vol. 4, April, pp. 123-193.

McFarland, M. C. (1993). Formal Verification of Sequential Hardware: A Tutorial, IEEE Trans on Computer Aided Design Of Integrated Circuits and Systems, vol. 12, No.5, pp.633-654.]