Some Techniques for Efficient Symbolic Simulation-based Verification

Prabhat Jain
Dept. of Computer Science
University of Utah
Salt Lake City, UT 84112

Ganesh Gopalakrishnan
Dept. of Computer Science
University of Utah
Salt Lake City, UT 84112

Abstract

We present some techniques to make symbolic simulation-based verification efficient in practice. The first technique is applied to the verification of non-regular designs. Minimally instantiated symbolic simulation vectors are first generated, and all these vectors are encoded into one vector using auxiliary (parametric) Boolean variables. The second technique also pertains to non-regular designs, and it offers a way to compactly encode input constraints during symbolic simulation. Two variations of this technique are explored. The third technique relates to the verification of regular arrays. It is shown that many regular arrays require input constraints to be obeyed, and that these constraints can be encoded using parametric Boolean variables. Another related technique (applicable to regular arrays where control-flow is data independent) does not encode the input constraints, but takes them into account after symbolic simulation. All these techniques are geared towards reducing the computational (and human) effort during simulation-based verification, and are supported by experimental results.

1 Introduction

As digital VLSI circuits are being used in an increasing number of real-life applications, it becomes important to verify them for correct operation. Conventionally, simulators are used to check that a circuit’s responses match the expected responses. However, simulation in the traditional sense is insufficient to establish correctness for large circuits because it is impractical to verify the circuit behavior for all possible states and inputs, which are usually very large in number. Several formal verification approaches have been suggested for the verification of digital circuits.

Formal verification at the high-level can provide useful information (e.g., circuit state invariants) for efficient symbolic simulation at the low-level, in addition to its other advantages. But, current formal hardware verification approaches cannot accurately model low-level circuit details. Since the simulators (e.g., switch-level) can model low-level circuit details accurately, an approach combining the capabilities of formal verification at the high-level and symbolic simulation at the low-level can derive the advantages of both the approaches.

Bryant has proposed symbolic switch-level simulation for formal hardware verification [1]. His verification approach has been applied to verify a static RAM, data paths, and pipelined circuits [2]. The combination of formal verification at the high-level and simulation-based verification at the low-level has been proposed in [5, 8].

The basic feature of a symbolic simulator is that it allows the use of Boolean variables in state and input vectors; each such use of a variable represents two values, thus halving the total number of vectors needed. One straightforward way to minimize the number of symbolic vectors is by loading each bit of the system’s state elements with distinct Boolean variables, and also using distinct Boolean variables at all the inputs of the circuit. This approach does not work in practice, due to several reasons. First of all, keeping states and inputs at their most general forms by using un-instantiated vectors is an attempt to prove that a circuit operates correctly for all possible states and inputs. Most realistic circuits do not operate as desired for all possible states and inputs.

Thus, one has to instantiate the symbolic state and input vectors to the right degree so that the circuits obey the state and input constraints that the designers have assumed for their correct operation. We refer to these vectors as minimally instantiated symbolic...
simulation vectors. Our verification approach for datapath and control circuits is based on a simple hardware specification formalism called HOP [4], a parallel composition algorithm called PARCOMP, and a switch-level simulator (COSMOS). Any other suitable specification formalism and a simulator can be used in our verification approach. In the past, we have studied the problem of generating minimally instantiated symbolic simulation vectors for non-regular designs, and also developed techniques to integrate the formal verification phase with the simulation phase. The details of this verification approach are discussed in [5].

We have also investigated a related technique based on HOP, PCA—a parallel composition algorithm for regular arrays, and COSMOS for the verification of regular array designs [6]. We have studied the problem of representing input constraints for regular arrays using parametric Boolean expressions. To illustrate this technique, consider a circuit with four inputs \( \text{in}_1, \text{in}_2, \text{in}_3, \) and \( \text{in}_4 \) and the input constraint that exactly one input be 1 at any time for its correct operation. This input constraint can be encoded using two parameter Boolean variables, say, \( b_1 \) and \( b_2 \), such that

\[
\begin{align*}
\text{in}_1 \land \neg \text{in}_2 \land \neg \text{in}_3 \land \neg \text{in}_4 \lor \\
\neg \text{in}_1 \land \text{in}_2 \land \neg \text{in}_3 \land \neg \text{in}_4 \lor \\
\neg \text{in}_1 \land \neg \text{in}_2 \land \text{in}_3 \land \neg \text{in}_4 \lor \\
\neg \text{in}_1 \land \neg \text{in}_2 \land \neg \text{in}_3 \land \text{in}_4
\end{align*}
\]

where \( F = G \) means that \( F \) and \( G \) are logically equivalent. Thus, the circuit can be simulated by applying the expression \( b_1 \land b_2 \) on input \( \text{in}_1, \text{in}_2 \) and \( \neg b_2 \) on \( \text{in}_3 \), and so on. We call the right-hand side a parametric Boolean form, and the variables \( b_1 \) and \( b_2 \) parametric Boolean variables. The use of parametric forms for the verification of finite-state machine descriptions has been described in [3]. In [7], we have reported a method to generate parametric Boolean expressions for the constraints expressed as a Boolean equation.

Organization of the Paper

In section 2, we take a simple example that was also studied in [5], called 'Minmaz' to illustrate the encoding technique for minimally instantiated vectors. In section 3, we apply our technique for handling input constraints on a non-regular design—a Huffman encoder. Two techniques are investigated in the context of the Huffman encoder circuit. In section 4, we study the problem of verifying regular array designs through symbolic simulation. Two different techniques have been studied in this connection. These techniques are illustrated on the regular array implementation of the least recently used (LRU) algorithm (this example was also studied in [6]). In section 5, we summarize our results and present ongoing research.

2 Composing Symbolic Simulation Vectors

This technique is illustrated on the Minmaz example. In section 2, we briefly summarize our results from [5] and show how minimally instantiated simulation vectors can be obtained for Minmaz. In section 2, we discuss how these vectors can be composed to obtain one symbolic simulation vector.

Minimally Instantiated Vectors for Minmaz

Minmaz has three registers, \( \text{MAXI}, \text{MINI}, \) and \( \text{LASTII} \). It implements five operations, \( \text{IClr} \_en, \text{IClr} \_dis, \text{Idis}, \text{Ireset}, \) and \( \text{Ien} \). The Ien operation considered here, reads the current input, updates \( \text{MAXI} \) and \( \text{MINI} \) with the (running) maximum value so far, and the minimum value so far, respectively. It also causes an output equal to the average of the max-so-far and min-so-far to be produced on the output port \( \text{OUT} \).

We first wrote the specification of the desired behavior of Minmaz in HOP. This specification is called \( \text{MINMAX} \). We then wrote the behavioral specifications for its submodules, and a structural description corresponding to the Minmaz circuit. We then submitted the structural description to PARCOMP to derive a behavioral description for Minmaz from its structural description. The derived specification is called \( \text{MM} \_\text{IBAS} \). \( \text{MM} \_\text{IBAS} \) was proved to be equivalent to \( \text{MINMAX} \) using algebraic/equational reasoning techniques (the proof was done manually). In the process, we discovered that the contents of the \( \text{MAXI} \) register will always be greater than or equal to that of the \( \text{MINI} \) register. Therefore, it is necessary that a simulation of the Minmaz circuit with state values violating this condition be never attempted.

We generated minimally instantiated simulation vectors for the Minmaz using the technique outlined in [5]. Some of the sixteen vectors generated, for the case where the circuit input \( \text{IN} \) is greater than \( \text{MAXI} \) (also taking the circuit invariant \( \text{MAXI} > \text{MINI} \) into
account) are now listed:

\[ MINI_0 = (0,0,MINI_1,MINI_0), IN_0 = (1,IN3,IN1,IN0) \]
\[ MAXI_0 = (0,1,MAXI_1,MAXI_0) \]
\[ MINI_1 = (0,MINI_0,MINI_1), IN_1 = (1,IN0,IN1,IN0) \]
\[ MAXI_1 = (0,MINI_1,MAXI_0) \]
\[ MINI_2 = (0,MINI_1,MINI_2), IN_2 = (1,IN0,IN1,IN0) \]
\[ MAXI_2 = (0,MINI_2,MAXI_3) \]

\[ \ldots \]
\[ MINI_{15} = (IN0,IN1,IN0), IN_{15} = (IN0,IN1,IN0) \]
\[ MAXI_{15} = (IN0,IN1,IN0) \]

Here, \( MINI_i \) represents the \( i \)th vector to be loaded into the register \( MINI \) (\( 0 \leq i \leq 15 \)). Verification time using this approach is listed in Figure 1 under the heading “Minimal Instantiation”.

Combining the Minimally Instantiated Vectors

A better technique is reported in this paper: instead of simulating the sixteen vectors separately, they were encoded into one vector, using four (i.e., \( \log_2(Nvecs) \)) parametric Boolean variables, where \( Nvecs = 16 \) is the number of vectors obtained in the previous section. The encoding technique is the following (we show the technique on \( MINI \); the vectors for \( II \) and \( MAXI \) are similarly encoded). Let \( y_0, \ldots, y_{Nvecs-1} \) be the minterms over the parametric Boolean variables. Let \( MINI_{i,j} \) be the symbolic expression for vector \( MINI_i \), bit position \( j \) (big-endian), where \( 0 \leq j \leq 3 \). For example, \( MINI_{2,3} = 0 \);
\[ MINI_{2,2} = MINI_2; \]
\[ MINI_{15,1} = IN_1, \]
and so on.

We encode \( MINI_0 \) through \( MINI_{15} \) and obtain one vector \( MINI \):

\[ MINI = \sum_{i=0}^{15} MINI_{i,0} \lor \sum_{j=0}^{3} MINI_{i,j} \land y_j \]

where \( \Sigma \) denotes logical \( \lor \). Verification time using this approach is listed in Figure 1 under the heading “Composed Vectors”.

3 Handling Input Constraints

In this section, we illustrate our technique to handle input constraints of non-regular designs by the Huffman encoder circuit for 6 character inputs. The Huffman encoder circuit inputs \( in0, in1, in2, in3, in4 \), and \( in5 \) have the “1 out of \( n \)” constraint, i.e., only one of the character inputs is 1 at a time. The Huffman codes for the characters are based on the frequency of occurrence of these characters. Symbolic simulation and verification of the Huffman encoder circuit, for each character \( char \), takes \( length(code(char)) \) cycles.

One Symbolic Vector for Each Length Group

The overall nature of this approach is to partition the set of valid inputs based on the knowledge of the circuit implementation, and apply the input constraint encoding technique to each such partition of the valid inputs. More specifically, for Huffman encoder circuit, since each group of characters with same Huffman code lengths requires the same number of cycles to verify, we partitioned the set of valid inputs based on the length of the Huffman codes for the character inputs. Then, we applied the technique of encoding input constraints as parametric Boolean expressions to a group of character inputs having the same Huffman code lengths, and verified the Huffman encoder circuit using one symbolic simulation vector for each such group of character inputs. This allowed us to simulate \( length(code(char)) \) clock cycles for that group of character inputs, and verify the results obtained over these cycles. This technique was also applied to a Huffman encoder for 26 characters, and the results are shown in Figure 1.

One Symbolic Vector for All Valid Inputs

We also experimented with the combination of the input constraint encoding technique using parametric Boolean expressions, and the technique of partitioning the set of valid inputs based on the knowledge of the circuit implementation. This approach involves encoding the \( partition \) number (which is based on character code lengths) using parametric Boolean expressions. Then, for each character input, we used the conjunction of the parametric Boolean expression encoding the input constraint and the Boolean expression identifying the length category for that character. This technique requires us to symbolically simulate only one symbolic vector; this vector is simulated for the number of clock cycles equal to the maximum character code length (to cover all possible characters). The state and output expressions obtained as a result of simulating this symbolic vector were instantiated for each length category, and were verified against the reference state output expressions for each length category, over corresponding number of clock cycles. The improvement in the symbolic simulation and verification time, with the application of the above techniques, for Huffman encoder circuits, encoding 6 and 26 characters, is shown in Figure 1.
4 Verification of Regular Arrays

Regular arrays form an important class of VLSI circuit designs, and with regular array designs being employed in numerous applications, the verification of regular arrays becomes an important step in their design and implementation as VLSI circuits. Also, it is important to develop efficient ways to handle input constraints for the verification of regular arrays, because many regular arrays are designed to be operated under input constraints (e.g., "1 out of n"). In this section, we show two techniques of handling input constraints of regular arrays, to reduce the symbolic simulation and verification effort. We use the Least Recently Used (LRU) priority algorithm, implemented as a two-dimensional array of LRU cells in VLSI, as an example to illustrate our techniques to handle input constraints. One hardware implementation of LRU algorithm which we consider here maintains an array of \( n \times n \) bits, initially all zeros, for a machine with \( n \) page frames. Whenever page \( k \) is referenced, the hardware sets all the bits of row \( k \) to 1 and sets all the bits of column \( k \) to 0. At any instant, the row with all bits set to 0 indicates the least recently used row, hence the least recently used page frame.

The operation of the LRU array relies on the input constraint that only the \( i \)th (\( 0 \leq i \leq n - 1 \)) row \( \text{col} \) bit and the \( i \)th \( \text{col} \) bit are 1, when page \( i \) is referenced.

Parametric Boolean Expressions at the Inputs

The LRU array was to be verified for all combinations of row and column input values which satisfy the input constraint for the LRU array. Each cell in the LRU array was initialized to a distinct symbolic variable, to verify the LRU array for all possible state values. We illustrate our technique for handling input constraints on the \( 4 \times 4 \) LRU array, and report the results for higher sizes.

In one alternative, we used scalar values, satisfying the input constraint, on the row and column inputs, and verified the resulting new state and output values against the expected values. It required four symbolic simulation vectors to verify the \( 4 \times 4 \) LRU array.

In the other alternative, we encoded the input constraint as parametric Boolean expressions on the row and column inputs, with two parameter Boolean variables \( b1 \) and \( b2 \). This technique reduced the number of symbolic simulation vectors from four to one. In general, \( \log_2 n \) parametric Boolean variables are required to encode the input constraint of an \( n \times n \) LRU array. In the LRU verification, this technique reduces the number of symbolic simulation vectors required to one, independent of the size \( n \) of the LRU array.

Simulation and verification times for various sizes of the LRU array are shown in Figure 1. We find that the improvement in the symbolic simulation and verification time, with the use of the encoding technique, is significant for large LRU array sizes.

Parametric Boolean Variables at the Inputs

For regular arrays, whose control flow does not depend on the data input values, we consider an alternative way of using the parametric Boolean expressions to encode the input constraints. Instead of using parametric Boolean expressions at the inputs, we use distinct symbolic variables at the inputs, which encode all possible input values, and substitute these symbolic input variables in the resulting state and output expressions by their corresponding parametric Boolean expressions encoding the input constraint, and verify the state and output expressions obtained by such substitution against the expected values. This technique of deferring the handling of input constraint until the verification time, potentially result in the reduced symbolic simulation effort, because the symbolic simulation is done with symbolic Boolean variables at the inputs, instead of parametric Boolean expressions at the inputs. For regular arrays with control flow dependent on input data values, the use of distinct symbolic variables at the inputs may not improve the symbolic simulation and verification time, as some of the control decisions in the circuit cannot be made without the inputs satisfying the required input constraint. This may result in more symbolic simulation effort, and can also fail to go through.

The symbolic simulation and verification times, using this technique, for various sizes of the LRU array are shown in Figure 1.

5 Results and Conclusions

Simulation-based verification is a powerful approach for the verification of hardware designs, which can complement formal verification using theorem provers. The exact boundary between these techniques has not been drawn yet, as the former technique is still in its infancy. There is considerable incentive to make simulation-based verification scale up to large circuits as this would provide digital system designers with a familiar tool (a simulator) that verifies designs almost automatically. Results reported in this paper indicate that simulation-based verification can scale
up to large circuit sizes in many cases. By studying more examples, we hope to get further insight into the technique(s) that would work best for a given example. This, and the implementation of a unified simulation/verification framework would constitute our future work. We are currently implementing the algorithm to generate parametric Boolean expressions for a constraint expressed as a Boolean equation.

References


Figure 1: Experimental Results\(^1\) for Minmax, Huffman Encoder, and LRU array\(^2\)