Towards A Verification Technique for Large Synchronous Circuits

Prabhat Jain, Prabhakar Kudva, and Ganesh Gopalakrishnan

Department of Computer Science,
University of Utah,
Salt Lake City, UT 84112

Abstract. We present a symbolic simulation based verification approach which can be applied to large synchronous circuits. A new technique to encode the state and input constraints as *parametric Boolean expressions* over the state and input variables is used to make our symbolic simulation based verification approach efficient. The constraints which are encoded through parametric Boolean expressions can involve the Boolean connectives $(\cdot, +, \neg)$, the relational operators $(<, \leq, >, \geq, \neq, =)$, and logical connectives $(\wedge, \vee)$. This technique of using parametric Boolean expressions vastly reduces the number of symbolic simulation vectors and the time for verification. Our verification approach can also be applied for efficient modular verification of large designs; the technique used is to verify each constituent sub-module separately, however in the context of the overall design. Since regular arrays are part of many large designs, we have developed an approach for the verification of regular arrays which combines formal verification at the high level and symbolic simulation at the low level(e.g., switch-level). We show the verification of a circuit called *Minmax*, a pipelined cache memory system, and an LRU array implementation of the least recently used block replacement policy, to illustrate our verification approach. The experimental results are obtained using the COSMOS symbolic simulator.

1 Introduction

Most digital VLSI circuits are checked for correct operation through *scalar valued simulation*. In this approach, *scalar* bit vectors—vectors over 0 and 1—are used as inputs to the circuit being simulated. As most real-world circuits require an impracticably large number of scalar vectors to check for all possible execution paths, scalar simulation alone is insufficient to verify a digital VLSI circuit.

Several formal verification approaches have been suggested for the verification of digital VLSI circuits. But, current formal hardware verification approaches cannot accurately model low-level circuit details (e.g., charge sharing). On the other hand, 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. 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 [4]. In [4, 1], it is shown that a symbolic simulator can be used to *verify* (check for all possible execution paths) many non-trivial circuits. His verification approach has been applied to verify a static RAM, data paths, and pipelined circuits [5, 6, 7]. Our verification approach for datapath and control circuits is based on a simple hardware specification formalism called HOP [9], a parallel composition algorithm called PARCOMP, and a switch-level simulator(COSMOS). 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 symbolic simulation phase. The combination of formal verification at the high-level and symbolic simulation based verification at the low-level has been proposed in [11, 14]. We have obtained encouraging results in this regard [11, 13, 12].

In order to reduce the symbolic simulation effort, a new technique to encode the state and input constraints as parametric Boolean expressions on the state and input variables is incorporated in our verification approach. This technique of using parametric Boolean expressions vastly reduces the number of symbolic simulation vectors and the time for verification, and thus makes our verification approach applicable to large synchronous circuits. Parametric forms have also been used in [2, 8] for the verification of finite state machines.

Our verification approach can be applied for efficient modular verification of large designs. Parametric Boolean expressions can be used to encode the input and state constraints of the sub-modules of the design. Each sub-module is individually verified. When verifying a sub-module, it is assumed that its context operates correctly, and so the inputs expected by the sub-module are derived directly from the input constraints of the sub-module. (The input constraints of each sub-module are typically known to the designer (*e.g.* a certain internal bus carries only unary values), and can be proved to be a consequence of the design, during high level verification.) The outputs of the sub-module being verified are not isolated from its context, and so the sub-module being verified is subject to the true electrical loadings.

Since regular arrays are part of many large designs, we have developed an approach for the verification of regular arrays which combines formal verification at the high level and symbolic simulation at the low level(e.g., switch-level). The verification approach is based on a simple hardware specification formalism called HOP, a parallel composition algorithm for regular arrays called PCA, and a switch-level symbolic simulator(COSMOS). We illustrate our verification approach on the Least Recently Used(LRU) page replacement policy implemented as a two-dimensional array of LRU cells in VLSI.

1.1 Outline of the Paper

In the following section, we present the basic idea of parametric Boolean expressions and the encoding of the state and input constraints as parametric Boolean expressions. In Section 3, 4, and 5 we present our symbolic simulation based verification approach and the use of parametric Boolean expressions through

examples. In Section 3, we show the verification of a circuit called *Minmax*. In Section 4, we show the verification of a pipelined cache memory system. In Section 5, we present our verification approach for regular arrays using an LRU array as an example. In Section 6, we summarize the results, report the ongoing effort, and outline the future work.

2 Parametric Boolean Expressions

We explain the idea of parametric Boolean expressions with the help of an example. Suppose a circuit with four inputs $in1$, $in2$, $in3$, and $in4$ has to obey the constraint that exactly one of these inputs be a 1. This constraint is captured by the sum of products formula:

$$\begin{aligned}&(in1 \wedge \neg in2 \wedge \neg in3 \wedge \neg in4) \ \vee (\neg in1 \wedge in2 \wedge \neg in3 \wedge \neg in4) \ \vee \\ &(\neg in1 \wedge \neg in2 \wedge in3 \wedge \neg in4) \ \vee (\neg in1 \wedge \neg in2 \wedge \neg in3 \wedge in4) \end{aligned} \quad (1)$$

Alternatively, this constraint is also captured by the formula:

$$\begin{aligned}= \exists b1\ b2\ .\ ((in1 = b1 \wedge b2) \wedge (in2 = b1 \wedge \neg b2) \wedge \\ (in3 = \neg b1 \wedge b2) \wedge (in4 = \neg b1 \wedge \neg b2))\end{aligned} \quad (2)$$

Formula 2 is logically equivalent to formula 1. However, formula 2 has the following advantage: the inputs $in1$ through $in4$ are expressed directly in terms of *parametric expressions* over *parametric variables* $b1$ and $b2$. These parametric expressions cover all possible values of $in1$ through $in4$ which satisfy the required constraint. Each permissible combination of $in1$ through $in4$ is obtained by every value assignment to the parametric variables. By applying the parametric expressions directly through input ports $in1$ through $in4$ during symbolic simulation, one symbolic simulation vector is tantamount to simulating the desired combinations of $in1$ through $in4$ separately. Thus, the parametric form of input constraints can be seen to effect a trade-off between the number of symbolic simulation vectors (which goes down exponentially) and the number of extra variables in symbolic input vectors (which goes up only by a logarithmic value, at worst). We find that this trade-off can reduce the symbolic simulation time significantly.

Many automatic ways to obtain the parametric form are well known, e.g., Boole's and Löewenheim's procedures [3]. We have improved these standard procedures in a number of ways. Some details are provided in section 6. In the rest of the paper, we focus on the applications of the parametric form, and not on procedures for obtaining them *per se*.

Constraints on the State and Input Vectors

In many situations, it is convenient to express the constraints of a circuit as a Boolean expression on the state and/or input bit-vectors. For example, a set-associative cache would require all the tags (bit-vectors) in a set to be different

($\neq$) for its correct operation. Here we consider the constraints which may involve relational operators ($<, \leq, >, \geq, \neq, =$) and logical connectives ($\wedge, \vee$). These constraints on the bit-vectors can also be expressed as Boolean expressions containing individual bit-variables of the bit-vectors, and parametric Boolean expressions can be obtained for these individual bit-variables. However, the direct generation of parametric Boolean expressions for bit-vectors, taking advantage of the recursive nature of the relations on the bit-vectors (*e.g.* $\neq$ on an N bit vector can be expressed using the *xor* function and $\neq$ on an $N-1$ bit vector) would be computationally more efficient. Moreover, symbolic simulation is more efficient using the vectors generated by taking advantage of the recursive nature of the relations on the bit-vectors.

$$
\begin{array}{ll}
a_3 = x_{ab} \cdot y_{ab} + p_3 & b_3 = (\overline{x_{ab}} + \overline{y_{ab}}) \cdot p_3 \\
a_2 = x_{ab} \cdot \overline{y_{ab}} + p_2 & b_2 = x_{ab} \cdot y_{ab} \cdot q_2 + \overline{x_{ab}} \cdot p_2 \\
a_1 = \overline{x_{ab}} \cdot y_{ab} + p_1 & b_1 = x_{ab} \cdot q_1 + \overline{x_{ab}} \cdot \overline{y_{ab}} \cdot p_1 \\
a_0 = \overline{x_{ab}} \cdot \overline{y_{ab}} + p_0 & b_0 = (x_{ab} + y_{ab}) \cdot q_0
\end{array}
$$

Fig. 1. Parametric Boolean Expressions for $A : [a_3, a_2, a_1, a_0] > B : [b_3, b_2, b_1, b_0]$

To illustrate the generation of parametric Boolean expressions for the constraints involving bit-vectors, consider two 4-bit vectors $A : [a_3, a_2, a_1, a_0]$ and $B : [b_3, b_2, b_1, b_0]$ and the constraint $A > B$. The parametric Boolean expressions for the bit-variables of these two vectors are shown in Figure 1. The instantiations of these variables result in minimally instantiated symbolic A and B vectors which satisfy the constraint $A > B$. For example, with $x_{ab} = 0$ and $y_{ab} = 1$, we obtain $A : [p_3, p_2, 1, p_0]$ and $B : [p_3, p_2, 0, q_0]$. We call x_{ab} and y_{ab} *control parametric variables* and $p_i (0 \leq i \leq 3)$ and $q_j (0 \leq j \leq 2)$ parametric Boolean variables.

Boolean expressions on bit-vectors containing the logical connectives $\wedge$ and $\vee$ can be first simplified into a disjunction of conjunctive-forms (or "cubes"). Then, the parametric Boolean expressions for each conjunctive-form can be obtained and combined to get the parametric Boolean expression for the given Boolean expression. The parametric Boolean expressions can be obtained using a Boolean equation solving procedure [3, 8], but we are working on automating the generation of the parametric Boolean expressions for bit-vectors, taking advantage of the recursive nature of the relations on the bit-vectors.

3 Verification of *Minmax*

In this section, we take a simple example that was also studied in [11]. *Minmax* (Figure 2) [15] has three registers, `MAXI`, `MINI`, and `LASTIN`. It implements five operations, `Iclr_en`, `Iclr_dis`, `Idis`, `Ireset`, and `Ien`. Here, we consider `Ien`

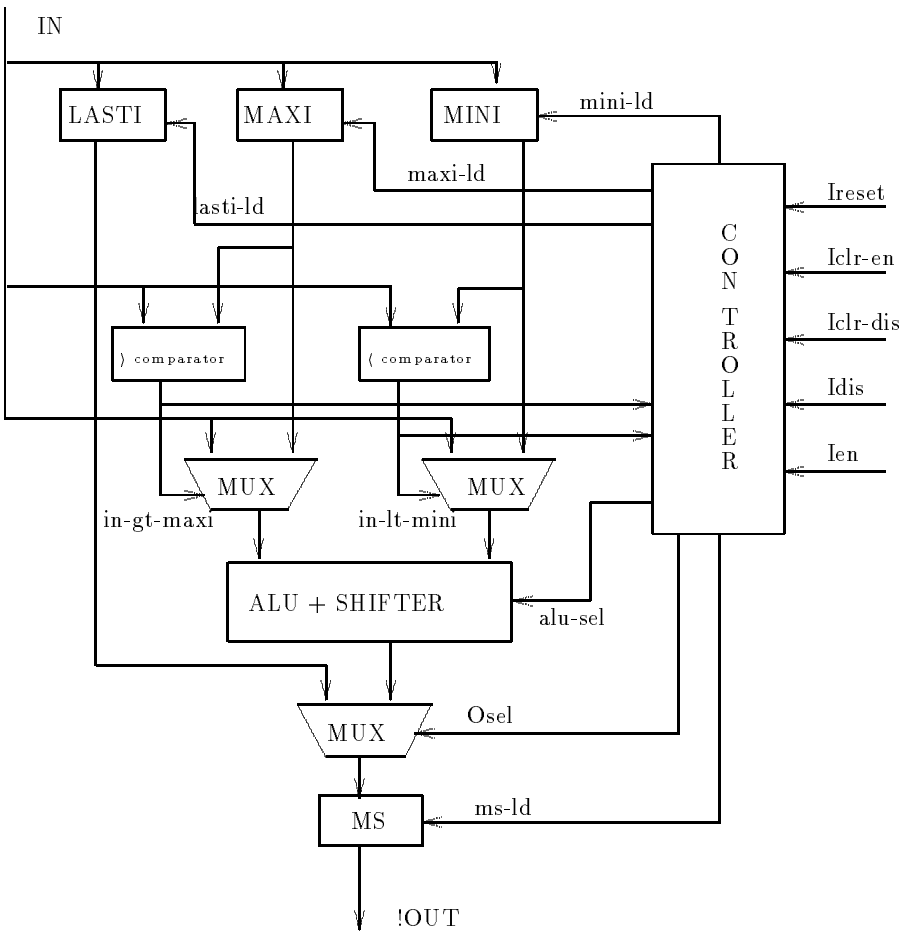

Fig. 2. Schematic of *Minmax*

operation which reads the current input, updates `MAXI` and `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 `!OUT`. A formal verification of *Minmax* was carried out using algebraic/equational reasoning techniques [10].

3.1 Verification with Minimally Instantiated Symbolic Vectors

Since every circuit requires some state and/or input constraints to be obeyed for its correct operation, one needs to instantiate the symbolic state and input vectors to the right degree so that the state and/or input constraints of the circuit

are satisfied in the symbolic simulation based verification of that circuit. We refer to these vectors as *minimally instantiated symbolic simulation vectors*. In [11], we approached the verification of *Minmax* by enumerating minimally instantiated symbolic simulation vectors; we used Prolog to generate the minimally instantiated symbolic vectors for *Minmax*. We generated symbolic simulation vectors for each condition of a data dependent conditional branch, augmented with the circuit invariant $\texttt{MINI} \leq \texttt{MAXI}$. Some of the sixteen vectors generated, for the case $\texttt{IN} > \texttt{MAXI}$ (also taking the circuit invariant $\texttt{MINI} \leq \texttt{MAXI}$ into account) are now listed:

MINI`0 = [0,0,MINI1,MINI0], IN`0 = [1,IN2,IN1,IN0], MAXI`0 = [0,1,MAXI1,MAXI0]
MINI`1 = [0,MINI2,0,MINI0], IN`1 = [1,IN2,IN1,IN0], MAXI`1 = [0,MINI2,1,MAXI0]
MINI`2 = [0,MINI2,MINI1,0], IN`2 = [1,IN2,IN1,IN0], MAXI`2 = [0,MINI2,MINI1,1]
...
MINI`15 = [IN3,IN2,IN1,0], IN`15 = [IN3,IN2,IN1,1], MAXI`15 = [IN3,IN2,IN1,0]

Here, `MINI_i` represents the ith vector to be loaded into the register `MINI`, and so on for the other vectors. Verification time using this approach, for the cases $(\texttt{IN} > \texttt{MAXI})$ and $(\texttt{MINI} \leq \texttt{IN} \leq \texttt{MAXI})$, are listed in Figure 5 under the circuit name `Minmax4` and the column "minimal instantiation"(this does not include the time required to generate the minimally instantiated symbolic vectors).

3.2 Verification with Parametric Boolean Expressions

Verification of the *Minmax* circuit for the `Ien` operation required the verification of three transitions whose state and input constraints were: $\texttt{IN} < \texttt{MINI} \leq \texttt{MAXI}$, $\texttt{MINI} \leq \texttt{IN} \leq \texttt{MAXI}$, and $\texttt{MINI} \leq \texttt{MAXI} < \texttt{IN}$. We generated parametric Boolean expressions for the state and input vectors satisfying these three constraints to verify the three transitions for `Ien` operation of the *Minmax* circuit, using the technique outlined in Section 2. The use of parametric Boolean expressions for the verification of *Minmax* reduced the number of symbolic simulation vectors to 1 for each of the three constraints mentioned above and it also reduced the verification time significantly. The verification time for *Minmax* using this approach is listed in Figure 5 under the column "Parametric Expressions"(this does not include the time required to generate the parametric Boolean expressions).

4 Verification of A Pipelined Cache Memory System

In this section we consider the verification of a pipelined cache memory system to illustrate our technique to verify large designs.

4.1 A Pipelined Cache Memory System

The pipelined cache memory considered here has a 2-way set-associative cache with 4 sets in the cache. The size of a block in each set is one byte and the tag associated with each block is 3 bits. A set is selected by the two higher-order bits of the Read/Write address.

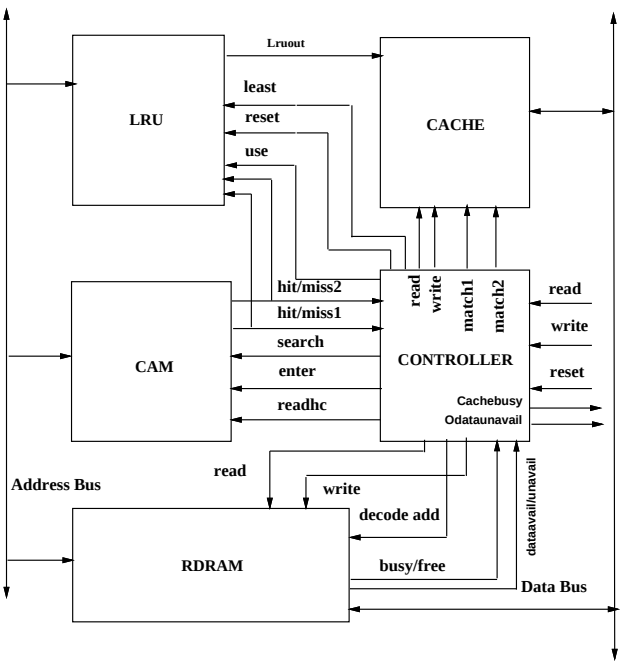

Fig. 3. The Pipelined Cache Memory System

The least recently used(LRU) block replacement policy is used for the cache miss on a Read or Write operation. Since each set has only two blocks, the LRU policy is implemented by one flip-flop for each set; output of the flip-flop indicates the least recently used block in the corresponding set. For higher set sizes, an LRU array would be used to implement the LRU block replacement policy. Verification of regular arrays, with LRU array as an example, is discussed in Section 5.1.

The main memory is updated using the write-through policy (i.e., for a Write operation, the data is written into the main memory and the cache at the same time). Since it takes more time to write the data into the main memory than into the cache for a Write operation, pipelining can be achieved by allowing more operations on the cache, while the data is being written into the main memory. In our pipelined cache system design, pipelining is achieved by allowing one or two Read operations (two Read operations, if the first Read operation following the Write operation results in a hit in the cache), while the data is being written into the main memory for a Write operation.

The block diagram of the pipelined cache memory system is shown in Figure 3. The pipelined cache system design consists of four main modules, as shown

in Figure 3. The CACHE module stores the data part of all the blocks in the cache. The LRU module contains the data storage and the logic necessary to implement the LRU block replacement policy. The CAM module stores the tag part of the addresses currently in the cache. It also contains the logic necessary to implement set selection and parallel search for the tag part of the address of a Read/Write operation. The CONTROLLER module controls the operation of the pipelined cache memory system. This pipelined cache memory system was implemented on a Tiny Chip (about 5,700 transistors) and the simulation files necessary for switch-level symbolic simulation in COSMOS were derived from the NET description of the design.

4.2 Verification Using Parametric Boolean Expressions

Symbolic simulation cannot be naively applied to verify the entire cache memory system. For example, if symbolic vectors are applied as the address inputs and the memory is asked to Read, all the locations covered by the symbolic address are "simultaneously read"; this can cause conflicting drives of values on the data output. Therefore, we resort to the technique of separately verifying the sub-modules of the cache memory. Specifically, the following sub-modules have to be separately verified: (a) the CACHE; (b) the DRAM; and (c) all remaining units treated as the third submodule. Notice that the DRAM and the CACHE modules of the pipelined cache memory system can be separately verified using the switch-level verification techniques outlined in [5].

To verify the pipelined cache memory system, we wrote the behavioral and structural description for the design in HOP. The inferred behavior of the design from the structural description by PARCOMP was used to determine the Read/Write operation sequences necessary to verify the pipelined cache memory system. Since our example cache memory system is pipelined, it is necessary to verify its operation over the sequences of Reads and Writes listed in the middle of Figure 5. Verification is separately carried out for each of these Read/Write sequences. For a particular sequence, the tags in the CAM are initialized to symbolic expressions that satisfy the CAM *invariant* (i.e., no two tags in a set have the same value). The Read/Write addresses are then set to symbolic expressions that cause the particular scenario (*e.g.* "Write Miss $\rightarrow$ Read Hit $\rightarrow$ Read Miss") to manifest.

In our first attempt, we used Prolog to encode the constraint among the tags of the CAM (captured by the CAM invariant) and the constraints on Read/Write addresses required to make each scenario manifest, and ran the Prolog description to generate minimally instantiated symbolic values that satisfied the constraints. An impracticably large number of symbolic vectors were obtained (e.g., the operation sequence Write Miss $\rightarrow$ Read Hit $\rightarrow$ Read Hit resulted in 191232 symbolic vectors).

We then explored the idea of using parametric Boolean expressions by generating the tags in the CAM and the Read/Write addresses satisfying the constraints as described above. The constraints involved the $\neq$ relation and the logical connective $\wedge$. The use of parametric Boolean expressions reduced the number

of symbolic vectors required for verification to *one* for all the Read/Write operation sequences beginning with a Write Hit operation and to *eight* for rest of the Read/Write operation sequences beginning with a Write Miss operation. The reason why eight symbolic vectors were required for each Read/Write operation sequence beginning with a Write Miss operation is the following: since a Write Miss operation would write the address tag in the CAM and the data in the CACHE, the set part of the Write address and the LRU value for the corresponding set were required to be instantiated to scalar values; there are four possible sets, and for each set, there are two possible LRU values. The symbolic simulation and verification times required for all the Read/Write operation sequences are shown in Figure 5.

We verified the pipelined cache memory system by supplying (using the `freeze` command in COSMOS symbolic simulator) the expected inputs from the DRAM and the CACHE module during the symbolic simulation of the Read/Write operation sequences, assuming that the DRAM and CACHE operate correctly.

4.3 Verification of Large Cache Sizes

We believe that the technique of using parametric Boolean expressions can be applied for the verification of large cache sizes. If the number of symbolic variables which can be used in the COSMOS symbolic simulator is a limitation for the verification of large cache sizes, the technique of using parametric Boolean expressions can be applied in the following way. The set part of the Write operation's address in an operation sequence can be instantiated to the scalar value and the tags of CAM for the sets in which addresses of the Read/Write operation sequence map to can be initialized to contain the parametric Boolean expressions satisfying the required constraints; the tags in all the other sets can be kept to the unknown value `X`. This would reduce the number of symbolic variables required in the verification of an operation sequence, but would increase the number of symbolic vectors required in the verification of the operation sequence. The number of symbolic vectors required would be proportional to the number of sets in the cache.

5 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 state and input constraints for the verification of regular arrays, because many regular arrays are designed to operate under input constraints (*e.g.*, "inputs must be unary"). In this section, we show our verification approach for regular arrays and show the application of parametric Boolean expressions in the verification of regular arrays. The hardware implementation of LRU page replacement policy

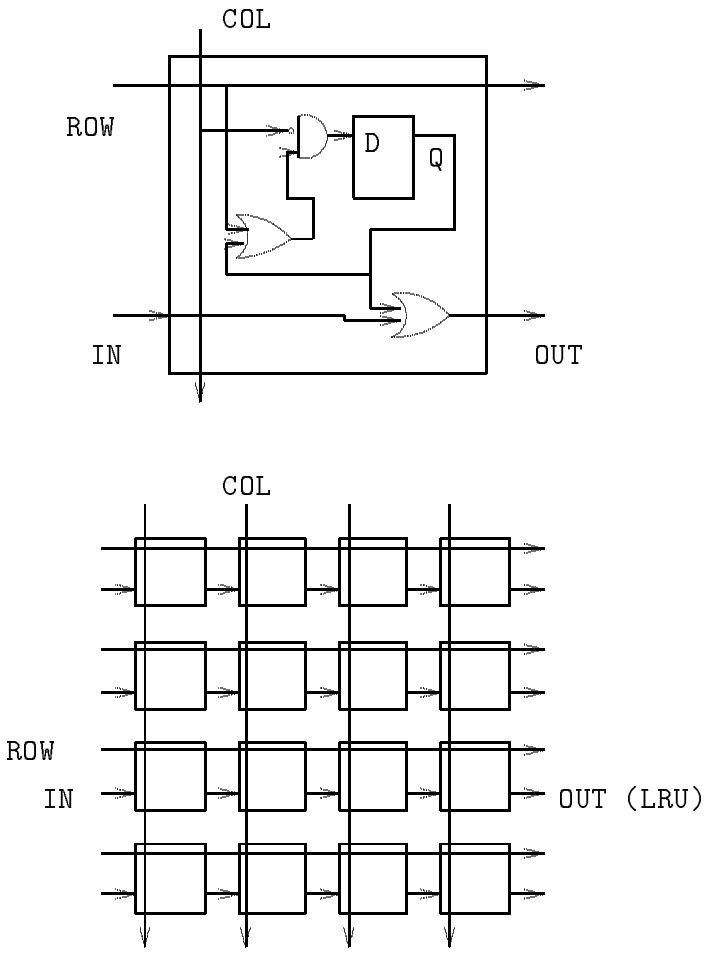

Fig. 4. LRU Cell and its HOP state diagram; LRU Array

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 the row k to 1 and sets all the bits of the 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.

5.1 The LRU Array

The LRU array is realized as a two-dimensional regular array of LRU cells. Each LRU cell of the regular array consists of a state bit which can be set to 1 by

keeping the ROW input to 1 and COL input to 0; the state bit can be set to 0 by keeping the COL input to 1. On rising edge of the clock, the state bit of the LRU cell is set to 0 or 1 depending upon ROW and COL inputs. On falling edge of the clock, the output OUT is computed as logical OR of IN input of the cell (which is OUT output of the previous cell) and the state bit of the LRU cell. The output of each row is logical OR of the state bits of the LRU cells in the row.

The functionality of an LRU cell is shown in Figure 4(a). A 4×4 LRU array is shown in Figure 4(c). The operation of the LRU array relies on the input constraint that only the ith ($0 \leq i \leq 3$) ROW bit and the ith COL bit are 1, when page i is referenced.

The LRU array implementation of the LRU policy is verified at two levels. At the first level, the LRU regular array behavior determined by PCA (a parallel composition algorithm for regular arrays) is verified against the abstract specification of the LRU array algorithm. The formal verification at this level is based on the homomorphism relation between states of the inferred behavior and the states of the abstract specification. We are skipping the details of this proof in this paper.

At the second level, the transistor level implementation of the LRU array corresponding to the structural description in HOP is verified against the behavior inferred by PCA. However, the PCA-inferred behavior cannot directly be used as the reference specification because PCA does not take into account the input constraints. Therefore, we first obtain the PCA-inferred behavior and then substitute into it the input and initial state values applied during the transistor level symbolic simulation; *this* forms the reference specification.

5.2 Verification with Parametric Boolean Expressions at the Inputs

The LRU array was verified for all combinations of row and column input values, which satisfied 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. (this is possible as the LRU array does not have any non-trivial circuit invariants.) We illustrate our technique to handle the input constraint on the 4×4 LRU array, and report the results for higher sizes. We first used *scalar values* on the row and column inputs, satisfying the input constraint, and verified the resulting new state and output values against the expected values. It required four symbolic simulation vectors to verify the 4×4 LRU array.

Then, we encoded the input constraint as parametric Boolean expressions on the row and column inputs, with two parameter Boolean variables $b1$ and $b2$ as described in in Section 2. With the use of this technique, the number of symbolic simulation vectors reduced 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 array verification, this technique reduces the number of symbolic simulation vectors to *one*, independent of the size n of the LRU array. Symbolic simulation and verification times for various sizes of the LRU array are shown in Figure 5 under "parametric expressions as inputs".

Circuit Name	No. of Transistors	IN > MAXI				MINI ≤ IN ≤ MAXI			
		Minimal Instantiation		Parametric Expressions		Minimal Instantiation		Parametric Expressions	
		No. of Vectors	Total time	No. of Vectors	Total time	No. of Vectors	Total time	No. of Vectors	Total time
Minmax4	1232	16	4.83	1	2.42	21	6.13	1	3.07

Operation Sequence	No. of Vectors	Total time
Write Hit → Read Miss	1	12.40
Write Hit → Read Hit	1	9.58
Write Hit → Read Hit → Read Hit	1	15.0
Write Hit → Read Hit → Read Miss	1	17.90
Write Miss → Read Miss	8	70.65
Write Miss → Read Hit	8	70.0
Write Miss → Read Hit → Read Hit	8	185.75
Write Miss → Read Hit → Read Miss	8	222.03

Circuit Name	No. of Transistors	Scalar Input Values		Parametric Expressions as Inputs	
		No. of Vectors	Total time	No. of Vectors	Total time
LRU 4×4	448	4	0.63	1	0.27
LRU 8×8	1792	8	6.93	1	2.29
LRU 16×16	7168	16	134.63	1	34.68

Fig. 5. Experimental Results[1] for Minmax, LRU array, and Pipelined Cache Memory System

The improvement in the symbolic simulation and verification time, with the use of parametric Boolean expressions, is significant for the large LRU array sizes. We find that the use of parametric Boolean expressions can lead to significant reduction in the number of symbolic vectors and the verification time in the symbolic simulation based verification of regular arrays.

6 Conclusions and Future Work

Symbolic simulation based verification is a powerful approach for the verification of hardware designs, which can complement formal verification using theorem provers. There is considerable incentive to make symbolic simulation based verification scale up to large circuits, as this would provide digital system designers with a familiar tool (a simulator) to verify the designs almost automatically. Results reported in this paper indicate that the symbolic simulation based verification approach can scale up to large circuit sizes in many cases. The main

[1] Total user time is shown in seconds.

motivation of our work has been to discover techniques that would help expand the class of circuits, and the circuit sizes that can be verified by the symbolic simulation based verification approach. One of the main observations is that the parametric Boolean expressions can be used in variety of ways for efficient symbolic simulation based verification of large synchronous circuits. Even though the generation of the parametric Boolean expressions can involve some computational effort, the parametric Boolean expressions, once generated, can be re-used during the debugging of the circuit being verified. In all the circuits we have verified, the use of parametric Boolean expressions enhanced the speed of the symbolic simulation process, mainly through a favorable tradeoff between the the number of simulation vectors (which is very much reduced) and the average number of symbolic variables per vector (which goes up only by a small amount).

We have automated our method for the generation of parametric Boolean expressions for the state and input constraints. We have also implemented two known methods, namely, Boole's and Löwenheim's method, to generate parametric Boolean expressions for comparison with our method. The comparison of our method with these methods shows that our method generates smaller parametric Boolean expressions which result in more efficient symbolic simulation. By studying more examples, we hope to get further insight into the technique(s) that would work best for a given example.

References

1. Derek L. Beatty, Randal E. Bryant, and Carl-Johan H.Seger. Synchronous circuit verification by symbolic simulation: An illustration. In *Sixth MIT Conference on Advanced Research in VLSI, 1990.* MIT Press, 1990.
2. Christian Berthet, Olivier Coudert, and Jean-Christophe Madre. New ideas on symbolic manipulations of finite state machines. In *Proceedings of the ICCD, 1990*, pages 224–227, 1990.
3. F. M. Brown. *Boolean Reasoning.* Kluwer Academic Publishers, 1990.
4. Randal E. Bryant. A methodology for hardware verification based on logic simulation. Technical Report CMU-CS-90-122, Computer Science, Carnegie Mellon University, March 1990. *Accepted for publication in the JACM.*
5. Randal E. Bryant. Formal verification of memory circuits by switch-level simulation. *IEEE Transactions on Computer-Aided Design*, 10(1):94–102, January 1991.
6. Randal E. Bryant, Derek L. Beatty, and Carl-Johan H. Seger. Formal hardware verification by symbolic ternary trajectory evaluation. In *Proc. ACM/IEEE 28th Design Automation Conference*, pages 397–402, June 1991.
7. Randal E. Bryant and Carl-Johan H. Seger. Formal verification of digital circuits using ternary system models. Technical Report CMU-CS-90-131, School of Computer Science, Carnegie Mellon University, May 1990. *Also in the Proceedings of the Workshop on Computer-Aided Verification*, Rutgers University, June, 1990.
8. Olivier Coudert, Christian Berthet, and Jean-Christophe Madre. Verification of sequential machines using boolean functional vectors. In *Proceedings of the IMEC-IFIP Workshop on Applied Formal Methods for Correct VLSI Design, Leuven, Belgium*, pages 179–196, November 1989.

9. Ganesh Gopalakrishnan. Hop: A formal model for synchronous circuits using communicating fundamental mode symbolic automata. Technical Report UUCS-TR-92-006, Dept. of Computer Science, University of Utah, Salt Lake City, UT 84112, 1992. Submitted to "Formal Methods in System Design".
10. Ganesh Gopalakrishnan and Prabhat Jain. A practical approach to synchronous hardware verification. In *Proc. VLSI Design '91: The Fourth CSI/IEEE International Symposium on VLSI Design, New Delhi, India*, January 1991.
11. Ganesh Gopalakrishnan, Prabhat Jain, Venkatesh Akella, Luli Josephson, and Wen-Yan Kuo. Combining verification and simulation. In Carlo Sequin, editor, *Advanced Research in VLSI : Proceedings of the 1991 University of California Santa Cruz Conference*. The MIT Press, 1991. *ISBN 0-262-19308-6*.
12. Prabhat Jain and Ganesh Gopalakrishnan. Some techniques for efficient symbolic simulation based verification. Technical Report UUCS-TR-91-023, University of Utah, Department of Computer Science, October 1991.
13. Prabhat Jain, Ganesh Gopalakrishnan, and Prabhakar Kudva. Verification of regular arrays by symbolic simulation. Technical Report UUCS-TR-91-022, University of Utah, Department of Computer Science, October 1991.
14. Carl-Johan H. Seger and Jeffrey Joyce. A two-level formal verification methodology using HOL and COSMOS. Technical Report 91-10, Dept. of Computer Science, University of British Columbia, Vancouver, B.C., June 1991.
15. D. Verkest and L. Claesen. The minmax system benchmark, November 1989.