Exploiting symmetry when verifying transistor-level circuits by symbolic trajectory evaluation

Manish Pandey and Randal E. Bryant

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA-15213, USA.

Abstract. In this paper we describe the use of symmetry for verification of transistor-level circuits by symbolic trajectory evaluation. We show that exploiting symmetry can allow one to verify systems several orders of magnitude larger than otherwise possible. We classify symmetries in circuits as *structural symmetries*, arising from similarities in circuit structure, *data symmetries*, arising from similarities in the handling of data values, and *mixed* structural-data symmetries. We use graph isomorphism testing and symbolic simulation to verify the symmetries in the original circuit. Using *conservative approximations*, we partition a circuit to expose the symmetries in its components, and construct reduced system models which can be verified efficiently. We have verified Static Random Access Memory circuits with up to 1.5 Million transistors.

1 Introduction

In this paper we have focussed on exploiting symmetry in the verification of transistor-level circuits by symbolic trajectory evaluation (STE). Many high performance hardware designs are custom designed at the transistor-level to optimize their area and performance, and this makes it necessary to verify them directly at the transistor-level. Common examples of such hardware units include static random access memory (SRAM) arrays which are found in instruction and data caches of microprocessors, cache tags, and TLBs, to name a few. These circuits exhibit considerable symmetry. Past work on verification of such arrays using STE [9, 3] has verified large systems which contain over 10^5 transistors, and over 10^4 memory bits. However, these approaches do not scale up well for much larger systems. By exploiting symmetry with the use of STE, we show that it is possible to verify systems that are orders of magnitude larger than previously possible. We present empirical results for the verification of SRAM circuits of varying sizes, including one with over 1.5 million transistors. Furthermore, our results show that our techniques scale up linearly or sub-linearly with SRAM size, and one can verify circuits that are much larger than our benchmarks.

Our verification approach builds on the following three ideas — *circuit partitioning*, *structural analysis*, and *conservative modeling.* Many systems, viewed as a whole, do not possess symmetries that can easily be exploited, but they are made up of smaller components which can. One can exploit the symmetry in the components by partitioning the larger system, verifying the smaller components, and composing the verification results.

We describe two forms of symmetries. *Structural symmetries* arise from similarities in the structure of a system, e.g., by replication of system components. *Data symmetries* arise from similarities in handling of data values in the system. Most previous work exploiting symmetry in formal verification to date [5, 8, 6] focussed on aspects of structural rather than data or mixed structural-data symmetry. We have found these other forms of symmetry useful in verifying many common digital building blocks. For instance, a change in the value of each decoder input results in a symmetric exchange in the values at the different decoder outputs. Such instances of mixed symmetry cannot be expressed by the other approaches. Structural symmetries in a transistor-level circuit can be detected and verified through a purely *structural analysis* of the system by doing circuit graph isomorphism checks. Other forms of symmetries can be verified through symbolic simulation.

Symmetry in a system imposes a partitioning of the system state space, where permutations of the same state appear in the same partition class. Once symmetry has been checked, one need verify the system for only one representative from each partition class. We exploit this property by constructing a *conservative model* of the system which provides full functionality only for the representative case. This approximation results in large savings in the system excitation function representation size.

Our work is related in many ways to recent symmetry work by others, including that by Clarke [5], Ip [8], and Emerson [6]. In [5], Clarke et al. describe the symmetry of a system as a transition relation preserving state permutation. The paper describes the construction of the symmetry reduced transition relation, and symbolically model checking with it. In [8], Ip and Dill discuss the verification of systems, where the symmetry in the system is identified by a special scalar-set datatype in the system description language. They describe an on-the-fly construction of the reduced state transition graph, and the checking of safety properties with it. Our work contrasts with these in many ways. For example, we do not constrain the user to explicitly give symmetry in the system description because we can directly work with transistor netlists. We can construct conservative models directly by switch-level analysis of the transistor-level system description, which is more efficient than on-the-fly construction of the reduced state transition relation.

2 Background

Symbolic trajectory evaluation (STE) was originally formulated as a formal verification method using a symbolic ternary simulator as the verification "engine" [4]. With ternary simulation, each state variable may have value 0, 1, or X, where X indicates an unknown or indeterminate state. With symbolic simulation, the state values are encoded using BDDs, allowing one simulation run to effectively evaluate circuit operation under many possible operating conditions.

With STE, the three state values are partially ordered by their "information content" with $X < 0$ and $X < 1$. The simulator is used to verify assertions of the form $[A \Longrightarrow C]$, where A and C are formulas containing (possibly symbolic) values for state variables, conjunctions, and the temporal logic "next time" operator. Intuitively, antecedent A defines a stimulus for the circuit inputs and initial state, while consequent C defines the expected response for the circuit outputs and new state. The simulator then proves that for any initial state and input sequence satisfying stimulus A, the circuit will generate outputs and new state satisfying C.

In a later formulation of STE [11], the states and their ordering was generalized to any complete lattice, and a slightly more general class of assertions was allowed. In this presentation, we will take a middle ground, using a lattice-structured state set, but with these states closely matching the ternary values of the original formulation. This particular formulation is chosen to allow a clear expression of symmetry properties.

2.1 State Domain

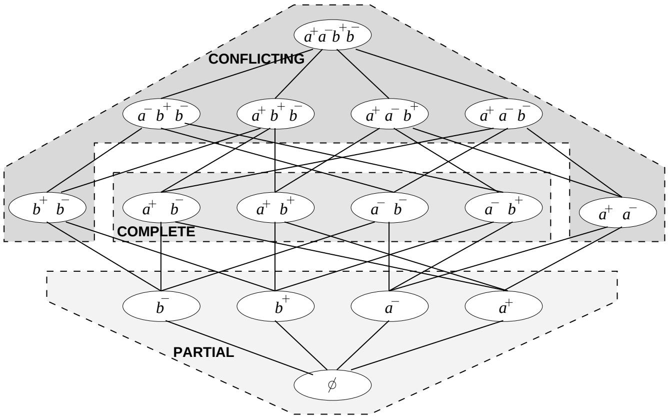


Fig. 1. Structure of State Lattice for Two Node Circuit

Let N denote the nodes (i.e., signal points) of a circuit. For each node n we define two *atoms*, written n^+ and n^-, indicating that node n has value 1 or 0, respectively. Let $\mathcal{A}$ denote the set of all atoms. We define a circuit state S to be any subset of $\mathcal{A}$, and $\mathcal{S}$ to be the set of all possible states, i.e., $\mathcal{S} = 2^{\mathcal{A}}$.

State set $\mathcal{S}$, together with the subset ordering $\subseteq$ forms a complete lattice, where states are ordered according to their "information content," i.e., how much they restrict the values of the circuit nodes. For example, the structure of the state domain for a circuit having nodes a and b is illustrated in Figure 1. In this diagram we indicate the set of atoms in each state, where $a^{\pm}$ indicates that both a^+ and a^- are present, and similarly for $b^{\pm}$. As the shaded regions indicate, states can be classified as being "partial", "complete", or "conflicting". In a partial state, some nodes have no corresponding atoms while others have at most one. In a complete state, there is exactly one atom for each node. In a conflicting state, there is some node n for which both atoms n^- an n^+ are present. Such a state is physically unrealizable—it requires a signal to be both 0 and 1 simultaneously. Conflicting states are added to the state domain only for mathematical convenience. They extend the semilattice derived from a ternary system model into a complete lattice. Our state lattice has the empty set $\emptyset$ as its least element and the set of all atoms $\mathcal{A}$ as its greatest element.

We view the operation of a circuit as an infinite sequence of states. A partial ordering $\sqsubseteq$ is defined over such sequences as the pointwise extension of the state ordering $\subseteq$. That is, for state sequences $S_0 = s_0^0 s_0^1 \ldots$, and $S_1 = s_1^0 s_1^1 \ldots$, $S_0 \sqsubseteq S_1$ iff $\forall i \geq 0 . s_0^i \subseteq s_1^i$.

2.2 Model Structure

The behavior of a circuit is defined by its *excitation function* $Y\colon \mathcal{S} \rightarrow \mathcal{S}$. This function serves a role similar to the transition relation or next-state functions of temporal logic model checkers. We require this function to be monotonic over the information ordering, i.e., if two states are ordered $s_1 \subseteq s_2$, then their excitations must also be ordered: $Y(s_1) \subseteq Y(s_2)$. Intuitively, we can view a state as defining a set of constraints on the signal values. We require the excitation function to remain consistent as more constraints are applied.

We will define a circuit model $\mathcal{M}$ to be the combination of a lattice-structured state set and a monotonic excitation function, i.e., $\mathcal{M} = \langle \mathcal{S}, Y \rangle$. The behavior of a circuit can be represented as an infinite sequence of states. We define a *circuit trajectory* to be any state sequence $\sigma = \sigma^0 \sigma^1 \ldots$ such that $Y(\sigma^i) \subseteq \sigma^{i+1}$ for all $i \geq 0$. That is, the state sequence obeys the constraints imposed by the circuit excitation function.

2.3 Representation of the Excitation Function

Our verifier computes the excitation function by evaluating logic expressions derived from the transistor circuit structure. These expressions are generated by the Anamos symbolic switch-level analyzer [2]. The analysis and the resulting excitation expressions capture a variety of low-level MOS circuit effects such as dynamic charge storage, different signal strengths, and bidirectional signal transmission.

As an example, consider the CMOS circuit shown in Figure 2, consisting of a NAND gate, (transistors T1, T2, T3 and T4), followed by pass transistor T5, and the pull-down transistor T6. This is followed by an inverter consisting of transistors T7 and T8. The first step in the symbolic switch-level analysis of the circuit is to partition it into *channel connected subnetworks* (CCSNs), each consisting of a set of storage nodes connected by the source-drain terminals of the transistors. In our circuit, this yields two subnetworks: CCSN1 containing nodes s and t, and CCSN2 containing node p. The analyzer derives the excitation expressions for each CCSN separately.

Figure 2 also shows the expressions describing the excitation for CCSN1 in the example circuit. These expressions are represented as a directed acyclic

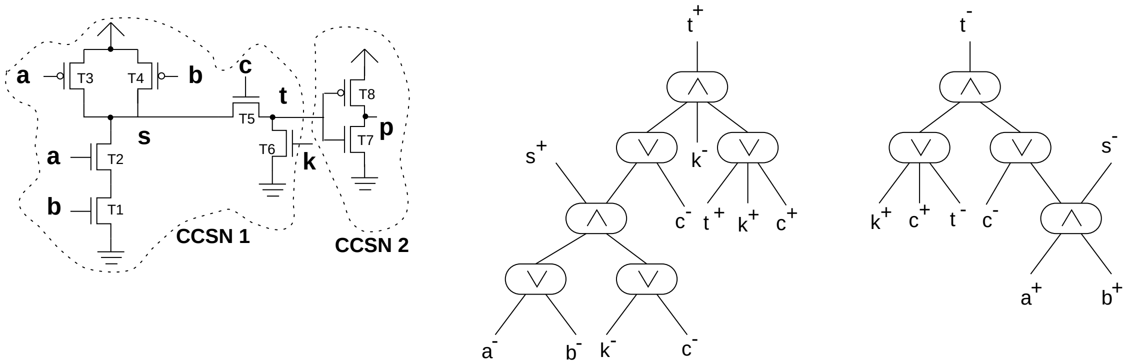


Fig. 2. Switch-level analysis of a circuit with pass logic and stored charge.

graph where the leaves indicate possible atoms in the set s and the roots indicate possible atoms in the set $Y(s)$. That is, for state s, a leaf labeled by atom a evaluates to true if $a \in s$ and to false otherwise. The Boolean operations indicated by the intermediate vertices are then evaluated. If a root labeled by atom a evaluates to true, then a is included in $Y(s)$. As this example shows, the excitation expressions use only the Boolean operations $\wedge$ and $\vee$, implying that the denoted excitation function obeys our monotonicity constraints.

To see how these excitation expressions capture the behavior of a transistor circuit, consider the expression with root labeled t^+ indicating when node t will be set to 1. This requires the conjunction of three conditions:

1. Either node s must be set to 1 (i.e., s^+ is included in the excitation state), or the transistor connecting nodes s and t must be disabled;
2. Node k must be set to 0, disabling the pulldown transistor;
3. and either t must be charged to state 1, or it is not storing charge dynamically.

As this example illustrates, the expressions are not particularly intuitive, e.g., one would not expect the atom k^+ to appear in the third term listed above. Nonetheless, they accurately capture the behavior of the circuit and obey our required monotonicity constraint.

For a large class of circuits, including the memory circuits we have verified, it can be shown that the DAGs describing the excitation functions have complexity linear in the number of circuit transistors. The constant factors can be significant, however. For example, a static RAM (SRAM) requires about 6 transistors for each memory bit. The DAGs generated by Anamos require around 73 vertices per memory bit. Moreover, the entire memory array consists of just two CCSNs. Hence the time and memory required to generate these DAGs limits the size of the memory circuit that can be analyzed. We will show how our symmetry reduction techniques can be used to verify the circuit using a reduced circuit model.

2.4 Trajectory Evaluation

In STE the system specification consists of a set of *trajectory assertions*, each having the form $[A \Longrightarrow C]$, where A and C are *trajectory formulas*. Antecedent A describes the stimulus to the circuit over time, while consequent C describes its expected response. Trajectory formulas (TFs) have the following recursive definition:

1. **Atoms**: For any node n, atoms n^+ and n^- are TFs.
2. **Conjunction**: $(F_1 \wedge F_2)$ is a TF if F_1 and F_2 are TFs.
3. **Domain restriction**: $(E \to F)$ is a TF if F is a TF and E is a Boolean expression.
4. **Next time**: $(\mathbf{X}F)$ is a TF if F is a TF.

The Boolean expressions occurring in domain restriction operators, having the form $E \to F$, give these formulas a symbolic character. They can be thought of as "guards," i.e., F must hold for the cases where E evaluates to true. For the theoretical development, however, it is convenient to first consider the form where E is restricted to be either 0 (false) or 1 (true). A *scalar* trajectory formula obeys this restriction throughout its recursive structure. The extension to the symbolic case then simply involves considering the valuation of the expressions for each variable assignment.

$\mathbf{X}$ is the *next time* temporal operator which causes advancement of time by one unit.

The truth of a scalar trajectory formula F is defined relative to a state sequence. Due to the restricted form of our temporal formulas, it can be shown that for every scalar formula F, there is unique *defining sequence* δ_F. Any sequence S satisfying F must satisfy the relation $\delta_F \sqsubseteq S$. Informally, the states in δ_F contain those atoms of F for which the guard expressions evaluate to true, positioned according to the nesting depth of the next-time operators.

We can combine a (scalar) trajectory formula and the circuit excitation function to generate a *defining trajectory* τ_F consisting of a sequence of states $\tau_F^0 \tau_F^1 \ldots$ given by:

$$\tau_F^i = \begin{cases} \delta_F^0 & \text{if } i = 0 \\ \delta_F^i \cup Y(\tau_F^{i-1}) & \text{otherwise} \end{cases}$$

It can be shown that τ_F is the unique minimum trajectory satisfying F. That is, τ_F satisfies F, and for any σ satisfying F, must obey the ordering $\tau_F \sqsubseteq \sigma$.

For an assertion $[A \Longrightarrow C]$ where both A and C are scalar formulas $\models_{\mathcal{M}}$ $[A \Longrightarrow C]$ if every trajectory of $\mathcal{M}$ satisfying A also satisfies C. Such an assertion can be verified by showing that $\delta_C \sqsubseteq \tau_A$. Essentially, this involves simulating the circuit applying the constraints given in antecedent A and checking the constraints given in the consequent C at the appropriate time points.

3 Symmetry

We express both circuit operation and the specifications in terms of sets of atoms. We can therefore express symmetries in a circuit and the corresponding transformations of the specification in terms of bijective mappings over atoms. A state transformation, σ, is a bijection over the set of atoms: $\sigma : \mathcal{A} \rightarrow \mathcal{A}$. We can extend σ to be a bijection over states by defining $\sigma(s)$ for state s as $\cup_{a \in s} \sigma(a)$.

Two types of state transformations are particularly interesting. A *data* transformation involves swapping the two atoms for a single node. For node n, we write $n^{\pm}$ to denote the transformation consisting of the swapping of n^{+} with n^{-}. A *structural* transformation involves swapping the atoms for two different nodes. For nodes n_1 and n_2, we write $\mathsf{n}_1 \leftrightarrow \mathsf{n}_2$ to denote the transformation consisting of the swappings: n_1^{+} with n_2^{+} and n_1^{-} with n_2^{-}. By composing transformations of these two forms, we can express a variety of circuit transformations. We will denote more complex transformations as a list of elementary transformations.

A state transformation σ is a *symmetry property* of a circuit with excitation function Y when $\sigma(Y(s)) = Y(\sigma(s))$ for every state s. That is, the excitation of the circuit on the transformed state $\sigma(s)$ matches the transformation of the excitation of s. One can readily show that σ is a symmetry property if and only if its inverse σ^{-1} is a symmetry property. Furthermore, if σ_1 and σ_2 are symmetry properties, then so is their composition $\sigma_1 \sigma_2$.

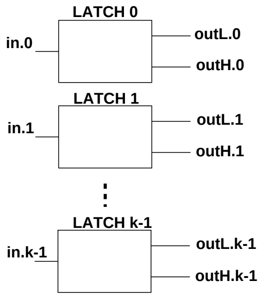


Fig. 3. Illustration of the symmetries of a circuit

If a *symmetry property* consists entirely of structural transformations, it is termed *structural symmetry,* and if it consists entirely of data transformations, it is termed *data symmetry.* A symmetry involving a combination of the two transformation types is called a *mixed symmetry.*

Consider, for example, the circuit shown in Figure 3. This circuit consists of k identical latches. In each latch outL is a complement of the input, and outH has the same value as the input. Since the latches are identical, this circuit has a structural symmetry corresponding to the swapping of any pair of latches i and j, such that $0 \leq i, j < k$:

$$[\mathsf{in}.i \leftrightarrow \mathsf{in}.j, \mathsf{outL}.i \leftrightarrow \mathsf{outL}.j, \mathsf{outH}.i \leftrightarrow \mathsf{outH}.j]\,. \qquad (1)$$

Each individual latch also stores data values 0 and 1 in a symmetric way, expressed for Latch 0 by the data symmetry:

$$\left[\mathsf{in.0}^{\pm}, \mathsf{outL.0}^{\pm}, \mathsf{outH.0}^{\pm}\right]. \qquad (2)$$

Finally, each latch can also be viewed as a one-bit decoder—it sets one of its outputs high based on its input data. Such behavior for Latch 0 is expressed by a mixed symmetry:

$$\left[\mathsf{in.0}^{\pm}, \mathsf{outL.0} \leftrightarrow \mathsf{outH.0}\right]. \tag{3}$$

We can extend state transformation σ to be a bijection over temporal formulas by defining $\sigma(F)$ to be the result of replacing every atom a in F by $\sigma(a)$. Similarly, we can extend σ to be a bijection over state sequences by applying σ to each state in the sequence. One can readily show that if temporal formula F has defining sequence δ_F, then its transformation $\sigma(F)$ will have defining sequence $\delta_{\sigma(F)} = \sigma(\delta_F)$. In addition, if σ is a symmetry property of a circuit model $\mathcal{M}$, then its defining trajectories for any temporal formula F will obey the symmetry: $\tau_{\sigma(F)} = \sigma(\tau_F)$. From this, one can conclude that for any assertion $[A \Longrightarrow C]$ and any symmetry property σ of model $\mathcal{M}$, $\models_{\mathcal{M}} [A \Longrightarrow C]$ if and only if $\models_{\mathcal{M}} [\sigma(A) \Longrightarrow \sigma(C)]$.

Thus, proving that σ is a symmetry property of a circuit allows us to infer the validity of a transformed assertion once we verify the original. For example, suppose we verify that Latch 0 in Figure 3 operates correctly for input value 1, and also prove that the transformations defined by Equations 1 and 2 are indeed symmetry transformations. Then we can infer from Equation 1 that for all j, Latch j operates correctly for input value 1, and from Equation 2 that Latch 0 operates correctly for input value 0. Furthermore, by composing these two transformations, we can infer that for all j, Latch j will operate correctly for input value 0.

The fact that symmetry properties may be composed makes it possible to prove the correctness of an entire set of assertions by simply verifying that each member of a set of "generators" for a group of transformations is a symmetry property. For example, Equation 1 represents a total of $k(k-1)/2$ symmetry transformations, corresponding to the pairwise exchange of any two latches. In general, one could argue that this circuit would remain invariant for any permutation π of the latches. Consider the transformation σ_π mapping the 6 atoms for each Latch i (two each for nodes $\mathsf{in}.i$, $\mathsf{outL}.i$ and $\mathsf{outH}.i$) to their counterparts in Latch $\pi(i)$. We could prove that each such transformation is a symmetry property, but this would require $k!$ tests. Instead, we can exploit the fact that any permutation π can be generated by composing a series of just two different permutation types. The "exchange" permutation swaps values 0 and 1, while the "rotate" permutation maps each value i to $i+1 \bmod k$. Thus, proving that the state transformations given by these two permutations are symmetry properties allows us to infer that σ_π is a symmetry property for an arbitrary permutation π.

We can verify structural symmetries in our circuit models by checking for isomorphisms in the switch-level network. Since Anamos derives its representation of the excitation function from the network, any isomorphisms in the network graph imply structural symmetries in the excitation function.

4 Conservative Approximations

Let $\mathcal{M}'$ and $\mathcal{M}$ be circuit models over the same state set, having excitation functions Y' and Y, respectively. We say that $\mathcal{M}'$ is a *conservative approximation* of $\mathcal{M}$ if for every state s, $Y'(s) \subseteq Y(s)$. In such a case, one can readily show that for any temporal formula F, the defining trajectories for the two models, τ'_F and τ_F, must be ordered $\tau'_F \sqsubseteq \tau_F$. From this, we can infer that for any assertion $[A \Longrightarrow C]$, if $\models_{\mathcal{M}'} [A \Longrightarrow C]$, then $\models_{\mathcal{M}} [A \Longrightarrow C]$. Thus, proving an assertion for a conservative approximation to a circuit model allows us to infer that the assertion holds for the original circuit.

Conservative approximations provide a systematic way to reason about partitioned circuits, allowing us to verify the complete circuit by proving properties about each partition. This is particularly useful when the partitioning can expose highly symmetric regions of the circuit. In addition, if we can prove that a circuit has some structural symmetry, then we can create a "weakened" version of the circuit containing just enough circuitry to verify the behavior for one representative of the symmetry group.

Let N' be a subset of the set of circuit nodes N, and $\mathcal{A}'$ be the corresponding set of atoms. Then we can view the removal of those nodes not in N' as yielding a conservative approximation to the circuit with an excitation function Y' such that:

$$Y'(s) = Y(s \cap \mathcal{A}') \cap \mathcal{A}'. \qquad (4)$$

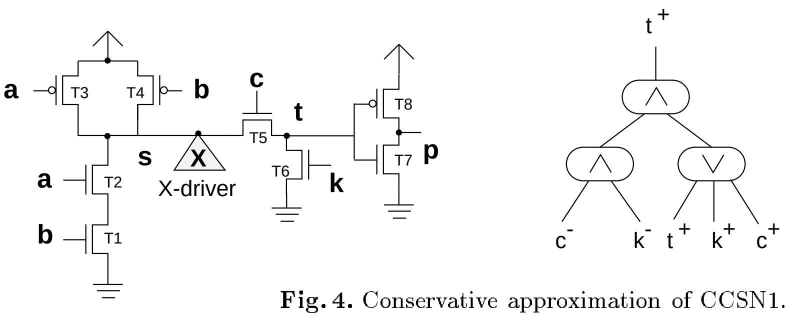


Fig. 4. Conservative approximation of CCSN1.

As an example, suppose we wish to create a reduced model for the circuit in Figure 2 by eliminating nodes a, b, and s. Then we could describe the remaining portions of CCSN1 by the excitation expressions shown in Figure 4. One can see that these expressions were obtained from those of Figure 2 by simplifying the result of setting the leaves for all eliminated atoms to false. This conservative approximation could be used to verify circuit operation for the cases where node c is set to 0. We have modified Anamos to generate these simplified expressions directly, avoiding the need to ever generate a complete model. In particular, we would replace node s in the example circuit by a "X-driver," consisting of an input node set to constant value X.

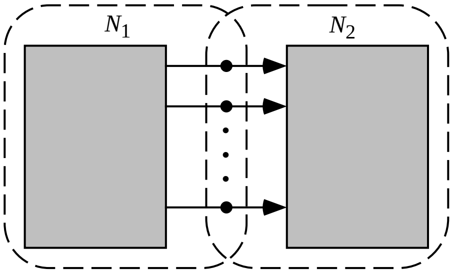


Fig. 5. Illustration of Circuit Partitioning.

We can view the partitioning of a circuit into different components as a process of creating multiple conservative approximations. For example, suppose we partition a circuit with nodes N into components having nodes N_1 and N_2, respectively, as illustrated in Figure 5. The set of nodes forming the interface between the components comprise the set $N_1 \cap N_2$. In this example, we assume the communication is purely unidirectional—N_1 generates signals for N_2. Suppose we wish to prove a property described by an assertion $[A \implies C]$, where the atoms of C are contained only in N_2. We could then create conservative models $\mathcal{M}_1$ and $\mathcal{M}_2$ using the subset construction given by Equation 4.

Taken individually, each of the two models is too weak to prove the assertion. Using a technique we term *waveform capture*, we can record the output values generated by model $\mathcal{M}_1$ and use them in verifying the assertion with model $\mathcal{M}_2$. In particular, let τ_A^1 be the defining trajectory generated by model $\mathcal{M}_1$ for antecedent A. Construct a temporal formula W describing the occurrence of the atoms corresponding to the nodes in $N_1 \cap N_2$ at the appropriate time points.[1] We have therefore proved that $\mathcal{M}_1$, and therefore $\mathcal{M}$, satisfies the assertion $[A \implies W]$. Using model $\mathcal{M}_2$, we then verify the assertion $[A \wedge W \implies C]$. Effectively, we "play back" the waveforms on the interface nodes. One can readily show that for any model $\mathcal{M}$ and any temporal formula F, if $\models_{\mathcal{M}} [A \implies F]$ and $\models_{\mathcal{M}} [A \wedge F \implies C]$, then $\models_{\mathcal{M}} [A \implies C]$, and therefore this pair of verifications is sufficient to prove the desired property.

For partitions in which the communication between partitions is bidirectional, this approach can be generalized to an iterative process, creating a series of waveforms $W_0, W_1, \ldots, W_k$ representing successively stronger approximations to the communication patterns between the two partitions.

5 Verification of a SRAM

Consider the 16-bit (1 bit/word) SRAM circuit shown in Figure 6. This circuit consists of the the following major components — row decoder, column address latches, column multiplexer (Mux) and the memory cell array core. To simplify the discussion here, many essential SRAM components like precharge column,

[1] Although the sequence τ_A^1 is infinite, we only need record the values up to the maximum depth of the next-time operators in C.

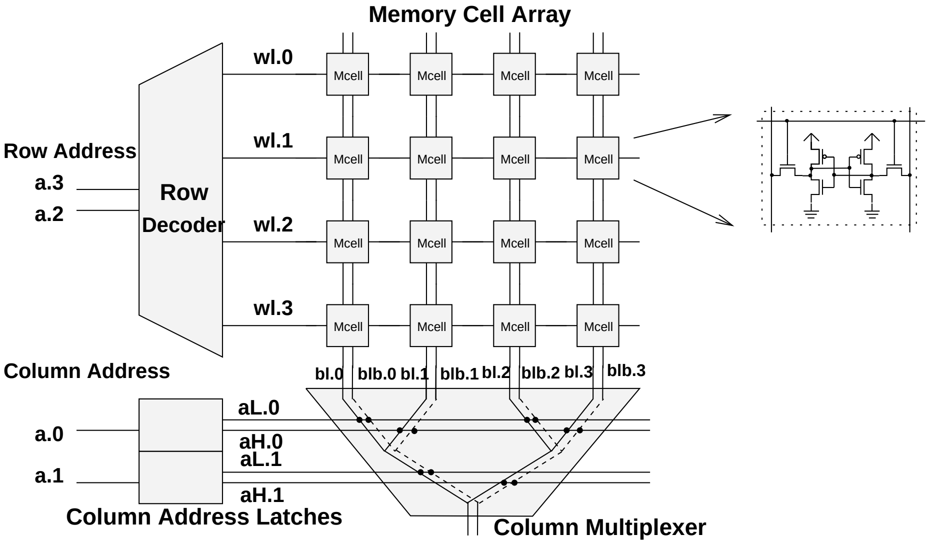


Fig. 6. SRAM circuit

write-drivers etc. have not been shown in the figure. This is a standard organization followed in many larger industrial SRAM arrays [7]. In order to verify this circuit we must show that the read and write operations work correctly. For example, if a memory location is addressed and read from, then the correct value must appear at the output. Similarly, if a write operation is done, the addressed memory location should be updated correctly. Such properties can be expressed with STE assertions.

The machinery we have built in the previous sections allows us to verify the read or write operation for only one location and, from the symmetry in the SRAM circuit, conclude that the operation works for every location. We expand on this below, starting with a discussion of SRAM symmetries.

5.1 Symmetries of a SRAM

Consider the decoder in Figure 7. For any memory operation, the value of the row address assigned to nodes a.2 and a.3, causes one of the word lines wl.0, wl.1, wl.2 and wl.3 to be active. The figure shows that wl.0 is active for row address 00. The same waveform occurs on the active word line regardless of the address. This mixed symmetry of the decoder is expressed by the group of transformations generated by transformations σ_0 and σ_1:

$$\sigma_0 = [\mathsf{a.2}^{\pm}, \mathsf{wl.0} \leftrightarrow \mathsf{wl.1}, \mathsf{wl.2} \leftrightarrow \mathsf{wl.3}]$$
$$\sigma_1 = [\mathsf{a.3}^{\pm}, \mathsf{wl.0} \leftrightarrow \mathsf{wl.2}, \mathsf{wl.1} \leftrightarrow \mathsf{wl.3}]$$

Transformation σ_i indicates that complementing bit i of the row address causes an exchange of signal waveforms for each pair of word lines j and k such that the binary representations of j and k differ at bit position i. The column address latches obey the "decoder" symmetry expressed by Equation 3.

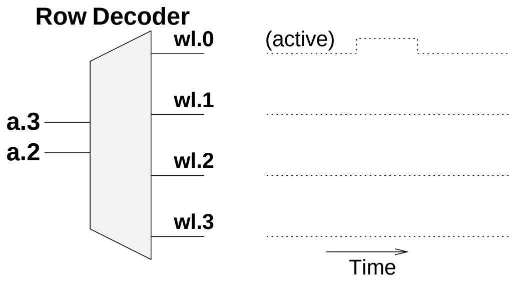


Fig. 7. Row decoder and signal waveforms on word lines for row address 00.

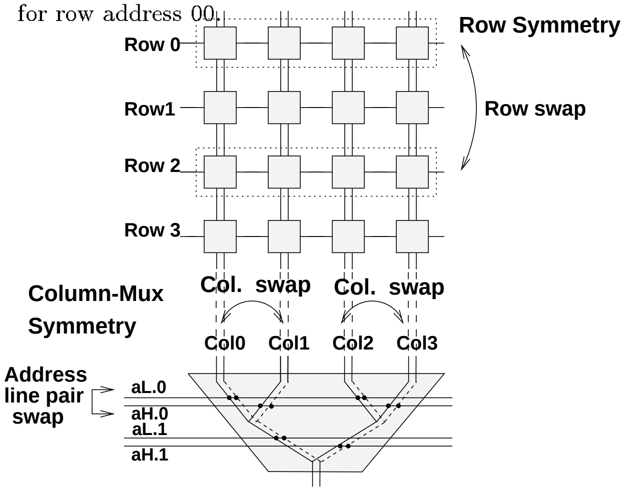


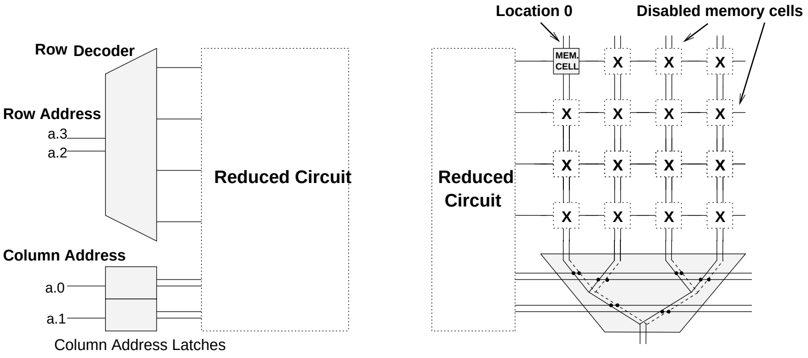
Fig. 8. Structural symmetries of the SRAM core.

The mixed symmetries of the decoder and the column address latches can be verified by symbolic simulation, where a single run of the simulator with n symbolic Boolean values at the circuit inputs is equivalent to 2^n runs of a conventional simulator with 0-1 values. For example, to verify that σ_0 is a symmetry of the decoder, we symbolically simulate the decoder with symbolic values s_0 and s_1 at the decoder inputs a.2, and a.3 in Figure 7. As the simulation proceeds, we check that a substitution of $\overline{s_0}$ for s_0 in the symbolic waveform for wl.0 (resp., wl.2) matches the symbolic waveform on wl.1 (resp., wl.3).

Figure 8 illustrates the two structural symmetries of the SRAM core and column Mux combination. The *row symmetry* arises from the invariance of the core-mux circuit structure under permutations of the rows of the core. The *column-mux symmetry* arises from the invariance of the circuit structure under a swap of column address latch output pairs accompanied by a corresponding exchange of columns. For example, in Figure 8, a swap of aH.0 and aL.0 accompanied by a swap of column 0 with 1, and a swap of column 2 with 3 is a symmetry of the circuit.

We verify the core-Mux symmetries in two parts. First we verify that arbitrary row and column permutations are symmetries of the core. Verification that the exchange and rotate permutation generators for rows and columns are symmetries suffices for this. This gives a total of 4 symmetry checks for the core. Next we verify the column-mux symmetry for the Mux. In the figure, the generators of the four different column address line pair permutations are the two permutations associated with each column address latch output pair. Therefore, two symmetry checks verify the column-mux symmetry. In general n symmetry checks must be done for the Mux in a SRAM with n column address line pairs.

5.2 Verification steps



Conservative approximation 1

Conservative approximation 2

Fig. 9. Conservative approximations of the SRAM.

In order to verify the SRAM circuit we go through the following sequence of steps.

1. **Circuit partitioning** — We partition the SRAM circuit into two parts. The first part consists of the decoder with the column address latches. The second part consists of the memory core and the column Mux.
2. **Symmetry verification** — Using symbolic simulation we verify the symmetries of the decoder and column latches. Using circuit graph isomorphism checks we verify the symmetries of the other part.
3. **Conservative approximations** — We create two conservative approximations of the SRAM (Figure 9). In the first one, the memory core and the column Mux are disabled. In the second model, the decoder, the column address latches are disabled, and all the memory cells except that for location 0 are disabled.
4. **Waveform capture** — Given the assertion $[A \Longrightarrow C]$ specifying an operation for memory location 0, we use the antecedent A to symbolically simulate conservative approximation 1. During the process of symbolic simulation we record the signal waveforms on the outputs of the decoder and the column address latches. We construct a trajectory formula W, which captures the signal values on the outputs recorded above. As discussed earlier, it can be shown that $[A \Longrightarrow W]$ is true.

5. **Verification of SRAM core** — Finally, with conservative approximation 2, we show that given the waveform W, and the antecedent A, the consequent C is true, i.e., $[A \wedge W \implies C]$. From the earlier discussion in section 4, if $[A \implies W]$ and $[A \wedge W \implies C]$ are both true, then we can conclude that $[A \implies C]$ is true, i.e., the memory operation is verified for location 0. Given the symmetries of the circuit we can then conclude that the operation works correctly for every memory location.

6 Experiments and Results

All the time and memory figures in this paper have been measured on a Sun SparcStation-20. We used the Anamos switch-level analyzer to generate switch-level models [2]. We modified Anamos to make it possible to attach *X-drivers* to circuit nodes to generate reduced models (conservative approximations) of switch-level circuits. Table 1 shows the results of model generation for SRAM circuits of varying sizes. For circuits larger than 16K, it was not possible to generate the full circuit model within reasonable time or memory bounds (empty table entries). Conservative approximations of SRAM circuits, on the other hand, can be generated for much larger circuits for a miniscule fraction of the cost of the full model. The reduced model size grows proportional to the square root of the SRAM size, and its generation time and memory is proportional to the SRAM size.

SRAM size (bits)	No. of Transistors	Model Size (Bool. ops)		Anamos Time (CPU Secs.)		Anamos Memory (MB)	
		Full	Reduced	Full	Reduced	Full	Reduced
1K	6690	79951	2781	120	4.1	9.6	0.9
4K	25676	307555	5462	863	14.1	36.8	2.1
16K	100566	1205239	10895	7066	43.2	144.2	6.0
64K	397642	—	21960	—	170.7	—	22.0
256K	1581494	—	44545	—	732.7	—	80.0

Table 1. Generation of SRAM model: Full vs. Reduced model.

In order to verify a structural symmetry, we take the original circuit, swap the circuit nodes and swap the circuit nodes specified in the symmetry. Then we verify if the new circuit is symmetric to the original circuit by verifying that the circuit graphs for the two circuits are isomorphic. The isomorphism check routines are based on a graph vertex coloring technique [1]. We have modified the isomorphism checking code from Anamos for our purpose. Essentially the coloring-based isomorphism check technique converts a circuit graph into a "canonical" representation, and two isomorphic circuits have the same canonical form. Table 2 reports the running time and memory taken for converting one instance of the memory core or column mux permutation into a canonical circuit, and the total time to do all the isomorphism checks. The total time and memory requirements scale linearly with the SRAM size. Table 3 reports the resources required to check the decoder and column address latch symmetries by symbolic simulation.

SRAM Size	Memory Core			Column Multiplexer			Total Isomorph.
(bits)	CPU Time (Secs.)	Memory (MB)	No. of checks	CPU Time (Secs.)	Memory (MB)	No. of checks	Check Time (Secs.)
1K	2.6	1.6	4	0.3	0.19	5	11.9
4K	11.1	6.5	4	0.5	0.38	6	47.4
16K	51.2	26.0	4	1.3	0.74	7	214.1
64K	232.1	104.0	4	3.0	1.44	8	952.4
256K	1135.6	416.0	4	6.6	3.50	9	4601.8

Table 2. Symmetry checks for memory core and column multiplexer.

We used the Voss verification system [10] to verify the reduced SRAM circuit. Table 4 shows the running time and the memory required for verifying the write operation for location 0. In addition, we must verify two other properties — that the read operation reads the value stored at the specified cell, and that operations at other addresses do not change the data in a given cell. The time and memory required to verify these other operations is similar to that of the write. The time and memory requirements grow roughly proportional to the square root of the memory size.

The total verification time for a SRAM circuit is the sum of the times in tables 1, 2, 3 and 4. For example, to verify a 64K SRAM, 170.7 secs. are required to generate the reduced circuit model, a total of 952.4 + 3.2 secs. are required to verify the circuit symmetries, and an additional 6.0 + 6.6 + 6.1 secs. are required to verify the reduced model for all the operations (time for other ops. not reported here). This gives a total verification time of 1145.0 secs. It is interesting to note that symmetry checks dominate much of this time. In the verification process, the only time we ever work with the complete circuit is the symmetry check phase. This partially explains the reason for the relatively large time and memory requirements of this phase. However, the circuit isomorphism code we have used is a simple modification of that in Anamos. There is considerable scope for reducing time and memory by developing a specialized circuit isomorphism checker.

SRAM Size	Time	Memory
(bits)	(CPU Secs.)	(MB)
1K	1.7	0.69
4K	2.1	0.74
16K	2.5	0.88
64K	3.2	1.10
256K	4.2	1.52

Table 3. Decoder and col. latch symmetry checks.

SRAM Size	Verif. Time	Verif. Memory
(bits)	(CPU Secs.)	(MB)
1K	1.5	0.79
4K	2.0	1.05
16K	3.0	1.80
64K	6.0	2.84
256K	18.5	4.26

Table 4. Verification of reduced SRAM writes.

7 Conclusion

We believe that with our work the problem of SRAM verification is solved. With more computational resources, and some fine-tuning of our programs, the results of our experiments indicate that we can verify multi-megabit SRAM circuits. The techniques we have presented can be used in a rather straightforward manner to exploit symmetries in other hardware units like set associative cache tags, where every set is identical in structure. One direction for future work in the short run would be to extend these ideas to verify content addressable memories. In the longer run, it would be interesting to apply these ideas to verify hardware units other than memory arrays. Candidates for such an application include a processor datapath, where one can find the presence of structural symmetries because of bit-slice repetition, and data symmetries arising from the datapath operations.

References

1. Derek L. Beatty and Randal E. Bryant. Fast incremental circuit analysis using extracted hierarchy. In *25th ACM/IEEE Design Automation Conference*, pages 495–500, June 1988.
2. Randal E. Bryant. Boolean analysis of MOS circuits. *IEEE Transactions on Computer-Aided Design*, CAD-6(4):634–649, July 1987.
3. Randal E. Bryant. Formal verification of memory circuits by switch-level simulation. *IEEE Transactions on Computer-Aided Design*, CAD-10(1):94–102, January 1991.

4. Randal E. Bryant and Carl-Johan H. Seger. Formal verification of digital circuits using symbolic ternary system models. In Robert P. Kurshan, editor, *Computer Aided Verification*, pages 121–146, 1990.
5. Edmund M. Clarke, Robert Enders, Thomas Filkorn, and Somesh Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9:77–104, 1996.
6. E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9:105–131, 1996.
7. Stephen T. Flannagan, Perry H. Pelley, Norman Herr, Bruce E. Engles, Taisheng Feng, Scott G. Nogle, John W. Eagan, Robert J. Dunnigan, Lawrence J. Day, and Robert I. Kung. 8-ns CMOS 64k × 4 and 256k × 1 SRAMs. *IEEE Journal of Solid-State Circuits*, pages 1049–1054, October 1990.
8. C. Norris Ip and David L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9:41–75, 1996.
9. Manish Pandey, Richard Raimi, Derek L. Beatty, and Randal E. Bryant. Formal verification of PowerPC(TM) arrays using symbolic trajectory evaluation. In *33rd ACM/IEEE Design Automation Conference*, pages 649–654, June 1996.
10. Carl-Johan H. Seger. Voss—a formal hardware verification system: User's guide. Technical Report 93-45, Department of Computer Science, University of British Columbia, 1986.
11. Carl-Johan H. Seger and Randal E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design*, 6:147–189, 1995.

This article was processed using the LaTeX macro package with LLNCS style