Robust QBF Encodings for Sequential Circuits with Applications to Verification, Debug and Test

Hratch Mangassarian, *Student Member, IEEE*, Andreas Veneris, *Senior Member, IEEE*, and Marco Benedetti

Abstract—Formal CAD tools operate on mathematical models describing the sequential behavior of a VLSI design. With the growing size and state-space of modern digital hardware designs, the conciseness of this mathematical model is of paramount importance in extending the scalability of those tools, provided that the compression does not come at the cost of reduced performance. Quantified Boolean Formula satisfiability (QBF) is a powerful generalization of Boolean satisfiability (SAT). It also belongs to the same complexity class as many CAD problems dealing with sequential circuits, which makes it a natural candidate for encoding such problems. This work proposes a succinct QBF encoding for modeling sequential circuit behavior. The encoding is parametrized and further compression is achieved using time-frame windowing. Comprehensive hardware constructions are used to illustrate the proposed encodings. Three notable CAD problems, namely bounded model checking, design debugging and sequential test pattern generation, are encoded as QBF instances to demonstrate the robustness and practicality of the proposed approach. Extensive experiments on OpenCore circuits show memory reductions in the order of 90% and demonstrate competitive run-times compared to state-of-the-art SAT techniques. Furthermore, the number of solved instances is increased by 16%. Admittedly, this work encourages further research in the use of QBF in CAD for VLSI.

Index Terms—SAT, QBF, BMC, k-induction, design debugging, sequential ATPG.

1 Introduction

The semiconductor industry has products pervading most commercial and consumer markets. Its growth is driven by a constant demand for electronic devices with continuously expanding functionalities and better performance. The design of such complex devices has been made possible by advances in Computer-Aided Design (CAD) tools. Today, major phases of the Very Large Scale Integration (VLSI) design flow, such as synthesis [1], placement/routing [2], verification [3] and test [4], have been fully or partially automated. Many chip design tasks deal with sequential circuits where procedures involving their analysis, optimization and verification have been shown to be PSPACE-complete (e.g., [5]–[8]). As the state-space of sequential designs continues to grow, their concise representation and efficient manipulation pose a real capacity challenge to modern CAD tools [9].

Existing CAD algorithms for sequential circuits can be classified into two categories. *Simulation-based* approaches are prevalent in the industry. This is due to their practicality and their scalability [9], [10]. These methods simulate the sequential design for a large number of input test vector sequences to prove its functionality, debug it, profile its power consumption, derive vectors for manufacturing test, etc. Due to the exponential number of possible input combinations, simulation is often non-exhaustive and it usually produces incomplete results, a fact that has become a cause for concern [9].

In the pursuit for viable alternatives to simulation, there has been a growing interest in *formal* CAD solutions. A formal CAD tool implicitly or explicitly explores the complete state-space of the design. It achieves this by operating on a mathematical model that encodes the sequential behavior of the design. A variety of such formal models have been proposed. Historically, the Finite State Machine (FSM) model has been used to explicitly explore the state-space of a sequential circuit. This approach leads to the state-space explosion problem [11]. Later, Binary Decision Diagrams (BDDs) [12] were introduced to traverse the design state-space symbolically, greatly improving the scalability of formal methods. However, BDDs can still lead to memory explosion problems as the size of modern designs grows [10], [13].

In the quest for scalable formal frameworks, Boolean satisfiability (SAT) has emerged as an effective platform for encoding many CAD for VLSI problems. This is primarily due to the dramatic performance improvements in SAT solvers [14]–[16]. A large number of NP-hard CAD problems have been translated into SAT instances which are solved by all-purpose SAT engines. Such SAT encodings for sequential VLSI problems often require an Iterative Logic Array (ILA) circuit representation, also known as time-frame expansion. An ILA represents a sequential design by replicating its combinational circuitry over a bounded number of cycles (time-frames).

- *H. Mangassarian is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, M5S 3G4 (hratch@eecg.toronto.edu).*
- *A. Veneris is with the Department of Electrical and Computer Engineering and with the Department of Computer Science, University of Toronto, Toronto, ON, Canada, M5S 3G4 (veneris@eecg.toronto.edu).*
- *M. Benedetti is with LIFO, University of Orléans, BP 6759 – 45067 Orléans Cedex 2, France (Marco.Benedetti@univ-orleans.fr).*

SAT encodings of problems in logic synthesis [17], model checking [13], [18], [19], Automatic Test Pattern Generation (ATPG) [20] and debugging [21], among others, use an ILA to allow the solver to reason on different time-frames of the design operation.

Although SAT solutions using an ILA model have further extended the scalability of formal tools, replicating the circuitry of a modern industrial-size design for a large number of cycles can exceed the available memory resources [18]. Evidently, more compact representations of sequential behavior are required using novel formalisms to ensure scalability for CAD tools without a sacrifice in performance.

Quantified Boolean Formula satisfiability (QBF) is a powerful generalization of SAT. It is PSPACE-complete, which makes it a natural candidate for encoding CAD problems dealing with sequential circuits. On-going developments in QBF solvers [22]–[25] have increased their performance dramatically. A robust QBF-based solution for VLSI CAD problems requires a succinct problem translation into a QBF instance, coupled with an effective solver. To this end, a few QBF encodings for formal verification problems have been investigated [26], [27] but experiments indicate that memory savings over SAT come at the expense of run-time performance.

The first aim of this paper is the development of a new, performance-driven mathematical model that encodes the sequential behavior of a design using QBF [28]. The encoding uses a single copy of the design and circumvents the memory-intensive circuit replication inherent in SAT-based representations using an ILA. To achieve this, it utilizes a novel hardware construction to represent the sequential circuit behavior.

Another contribution is the generation of a family of logically equivalent QBF encodings of the ILA, built around the original model. This is achieved using a technique termed time-frame windowing. This extension parametrizes the original solution to enable further compression and to boost the inference power of the QBF solver. It is shown that the resulting family of ILA encodings admits a non-trivial minimal-size member that proves to be empirically vital in the experiments.

The final goal of this work is the application of this theory to three notable CAD problems in sequential design: Bounded Model Checking (BMC), design debugging and sequential ATPG. This proves the usability and applicability of the encoding. Each of these problems is encoded in QBF using the new formalism and instances are solved with a general-purpose QBF engine.

Unlike previous QBF-based encodings for verification problems, the presented work provides a general-purpose QBF-based ILA representation for a multitude of CAD applications. It is designed to reduce memory requirements but also achieve competitive run-times when compared to state-of-the-art SAT. Indeed, an extensive suite of experiments on OpenCore designs confirms those achievements. Problem sizes are reduced by roughly 90% on average. Run-times are comparable to SAT and sometimes they outperform it by orders of magnitude. As an added advantage, due to the reduced memory footprint, the total number of solved instances is increased by 16%. The theoretical and empirical results of this paper encourage research in QBF-based encodings and QBF solvers as platforms to efficiently tackle intractable CAD problems.

The paper is organized as follows. Section 2 gives preliminaries in SAT, QBF and the ILA representation. Section 3 details the basic QBF encoding of time-frame expansion. Section 4 presents time-frame windowing. Sections 5, 6 and 7 illustrate the QBF formulations for BMC, design debugging and sequential ATPG, respectively, using the new formalism. Section 8 shows experimental results and Section 9 concludes the paper.

2 Preliminaries

2.1 Boolean Satisfiability

A propositional logic formula Φ over a set of Boolean variables $\{x_1, x_2, \ldots, x_n\}$ is said to be satisfiable or SAT if it has *a satisfying assignment*: a truth assignment to $x_1, x_2, \ldots, x_n$ that makes Φ evaluate to 1. Otherwise, Φ always evaluates to 0 and it is unsatisfiable or UNSAT.

A SAT solver determines whether a propositional formula Φ is SAT. Modern solvers take Φ in *Conjunctive Normal Form* (CNF) as a conjunction of *clauses* where each clause is a disjunction of *literals*. A literal is an occurrence of a variable x_i or its negation $\bar{x}_i$. For example, formula $\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_3)$ is SAT because $\{x_1 = 1, x_2 = 0, x_3 = 1\}$ is a satisfying assignment.

Given a logic circuit, a CNF formula expressing the circuit constraints can be constructed in linear time [29]. SAT-based solutions for CAD problems [13], [17], [18], [20], [21] translate a logic circuit into its equivalent CNF formula, which is given to the SAT solver. A circuit and its corresponding CNF formula are referred to interchangeably in this work.

Most SAT solvers are based on the search-based DPLL algorithm [30], first presented in 1962. After a number of pivotal advancements [14]–[16], SAT solvers today can handle industrial SAT instances with millions of clauses and variables.

2.2 Quantified Boolean Formulas

Formally, the problem of Boolean satisfiability asks whether $\exists x_1, x_2, \ldots, x_n \mid \Phi$. In formula Φ, all variables are existentially ($\exists$) quantified.

Quantified Boolean Formula satisfiability (QBF) is a generalization of SAT that also allows for universal ($\forall$) quantification of the variables. QBF is PSPACE-complete, which means that it can efficiently describe problems for which no polynomial-size SAT encodings are known.

A QBF formula in *prenex normal form* is written as:

$$Q_1\mathcal{V}_1 \quad Q_2\mathcal{V}_2 \quad \cdots \quad Q_r\mathcal{V}_r \quad \mid \quad \Phi$$

The *prefix* $Q_1\mathcal{V}_1 \; Q_2\mathcal{V}_2 \; \cdots \; Q_r\mathcal{V}_r$ consists of *quantifiers* $Q_i \in \{\forall, \exists\}$, such that $Q_i \neq Q_{i+1}$, and mutually disjoint

variable sets $\mathcal{V}_i$ called the *scopes*. The *matrix* Φ is a CNF formula on the variables in the prefix.

Q_r (Q_1) is referred to as the innermost (outermost) quantifier. Variable $v \in \mathcal{V}_i$ is labeled as an *existential* (*universal*) variable if $Q_i = \exists$ ($Q_i = \forall$). A scope $\mathcal{V}_i$ is said to *dominate* a scope $\mathcal{V}_j$ if $i < j$. If there exists a truth assignment to each existential variable as a function of its dominating universal variables, such that the matrix is satisfied for all universal variable assignments, the QBF problem is SAT, otherwise it is UNSAT. For example, the QBF problem: $\exists x_1\ \forall x_2\ \exists x_3 \mid (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1)$ is SAT because when $x_1 = 1$, for all values of x_2, there exists an assignment to x_3 ($x_3 = 1$ when $x_2 = 0$ and $x_3 = 0$ when $x_2 = 1$) that satisfies the matrix.

Unlike SAT where the DPLL algorithm is widely used, no particular paradigm has yet been shown superior in QBF solving. Search-based solvers [22], [25] extend the DPLL algorithm to deal with universal quantification. Quantifier elimination solvers [23] purge variables in the prefix. The solver in [23] uses Q-resolution to eliminate existential variables and expansion to eliminate universal ones. sKizzo [24] is a hybrid solver that uses skolemization to represent the QBF instance with BDDs. It then leverages search and resolution-based techniques to reason on these BDDs. State-of-the-art QBF solvers solve industrial QBF instances that typically contain tens to hundreds of thousands of variables and clauses.

2.3 ILA Representation of Sequential Circuits

The ILA representation of a sequential circuit for a bound k replicates its combinational component k times such that the next-state of each time-frame is connected to the current-state of the next time-frame. For example, to prove a property with BMC, the design might have to be unfolded for a number of time-frames equal to the diameter of the system, which can be exponential in the number of state variables [18]. In SAT-based debugging [21], the replication bound k is equal to the size of the counter-example that demonstrates a faulty behavior and it can be in the order of thousands of cycles. Similarly, in ATPG [20], k is the length of the input test vector sequence. Clearly, replicating the combinational circuitry for increasing values of k can become memory intensive in SAT-based applications.

The following notation is used in the paper. Variables x, y and s are Boolean vectors denoting the primary inputs, primary outputs and state elements of a sequential circuit. For each $z \in \{x, y, s\}$, z_i denotes the i^{th} bit in vector z. Symbol b denotes the number of state elements (DFFs). The behavior of a sequential circuit can be formally described by the predicate $T(s, s')$ expressing the transition relation of the system, which evaluates to 1 if and only if $s \rightarrow s'$ is a valid state transition. The predicate $T(s, s', x, y)$ explicitly mentioning primary inputs x and outputs y is also used to describe the system behavior.

The ILA of a sequential circuit for a bound k is shown in Figure 1. The variables s^{j-1} and s^j respectively denote the current-state and next-state of the j^{th} time-frame, for $1 \leq j \leq k$. Similarly, for each $z \in \{x, y\}$ and other circuit variables, z^j denotes the corresponding variable in time-frame j, and $Z = \langle z^1, z^2, \ldots, z^k \rangle$ denotes the sequence of all corresponding variables in the ILA.

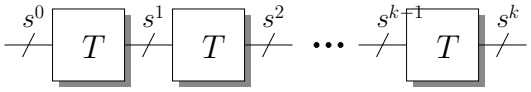


Fig. 1. Iterative Logic Array

The ILA in Figure 1 can be encoded as:

$$T^k(s^0, s^k) \quad = \quad \exists s^1, s^2, \ldots, s^{k-1} \mid \bigwedge_{j=1}^{k} T(s^{j-1}, s^j) \quad (1)$$

Predicate $T^k(s^0, s^k)$ evaluates to 1 if and only if there exists a valid path $p : s^0 \overset{p}{\rightsquigarrow} s^k$, with k state transitions.

3 Encoding ILAs using QBF

This section outlines a succinct QBF encoding of T^k which circumvents the memory-intensive circuit replication necessary in SAT encodings.

A *time-frame select* vector $t = \langle t_1, t_2, \ldots, t_{\lceil \lg k \rceil} \rangle$ of universal variables is created to allow indexing of the ILA time-frames in Figure 1. The aim is to associate each truth assignment to $\langle t_1, t_2, \ldots, t_{\lceil \lg k \rceil} \rangle$ with a different ILA time-frame. The construction in Figure 2 illustrates in hardware the matrix of the proposed QBF encoding. The time-frame select variables are the common select lines of two multiplexers (MUXes). The function of these MUXes is to connect the current-state s and next-state s' of the transition relation T to the current-state and next-state of a particular time-frame in the ILA, according to the truth assignment given to t. To achieve this, the inputs to the left MUX, $\langle s^0, s^1, \ldots, s^{k-1} \rangle$, are shifted by one time-frame from the inputs to the right MUX, $\langle s^1, s^2, \ldots, s^k \rangle$. In effect, depending on the assignment given to vector t, the single copy of T in Figure 2 "simulates" a different ILA time-frame.

In the prefix of the encoding, it is necessary that the states $s^0, s^1, \ldots, s^k$ dominate t in order to ensure state contiguity. Informally, the QBF encoding of T^k will express whether there exists an assignment to states

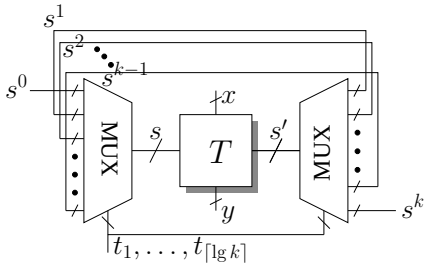
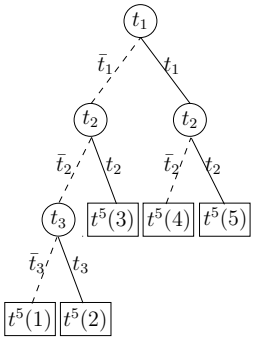


Fig. 2. QBF matrix of ILA encoding



(a) Tree for $\mathtt{MUX}_5$

$$\begin{array}{l}[(\bar{t}_1 \wedge \bar{t}_2 \wedge \bar{t}_3) \rightarrow (s = s^0)] \wedge \\ [(\bar{t}_1 \wedge \bar{t}_2 \wedge t_3) \rightarrow (s = s^1)] \wedge \\ [(\bar{t}_1 \wedge t_2) \rightarrow (s = s^2)] \wedge \\ [(t_1 \wedge \bar{t}_2) \rightarrow (s = s^3)] \wedge \\ [(t_1 \wedge t_2) \rightarrow (s = s^4)]\end{array}$$

(b) Left MUX when $k = 5$

Fig. 3. Encoding MUXes

$s^0, s^1, \ldots, s^k$, such that for all assignments to t, the construction in Figure 2 is satisfied.

To translate this statement into a QBF instance, we describe how to formally encode the two MUXes in Figure 2. A MUX with k inputs and $\lceil \lg k \rceil$ select lines, denoted $\mathtt{MUX}_k$, is associated with a binary tree with k leaves. The nodes at each tree level are labeled by a select line, and the outgoing node edges are labeled by the negative and positive literals of the select line associated with that node. For each truth assignment of the select lines, the literals along the edges of a unique path from the root to a leaf are satisfied, and that leaf is selected.

For example, the decision tree associated with a MUX of $k = 5$ inputs, select lines $\langle t_1, t_2, t_3 \rangle$, inputs $\langle s^0, s^1, s^2, s^3, s^4 \rangle$ and output s, is shown in Figure 3(a). For a given k and for $1 \leq j \leq k$, let $t^k(j)$ denote the conjunction of literals on the path from the root to the j^{th} leaf in the tree associated with a $\mathtt{MUX}_k$, where leaves are ordered from left to right. In Figure 3(a), if $t_1 = \mathtt{0}$ and $t_2 = \mathtt{1}$, then $t^5(3) = \bar{t}_1 \wedge t_2$ is satisfied irrespective of t_3, and hence $s = s^2$. The implication $t^5(3) \rightarrow (s = s^2)$ expresses this constraint. Figure 3(b) shows the set of constraints for $\mathtt{MUX}_5$.

The left and right MUXes in Figure 2 can now be respectively formalized as follows:

$$\begin{aligned} \mathrm{MUX}_k(s, t, \langle s^0, s^1, \ldots, s^{k-1} \rangle) &= \bigwedge_{j=1}^{k} [t^k(j) \rightarrow (s = s^{j-1})] \\ \mathrm{MUX}_k(s', t, \langle s^1, s^2, \ldots, s^k \rangle) &= \bigwedge_{j=1}^{k} [t^k(j) \rightarrow (s' = s^j)] \end{aligned}$$

Using these formalizations, T^k can be encoded in QBF as follows:

$$\begin{aligned} T^k(s^0, s^k) &= \exists s^1, \ldots, s^{k-1} \ \forall t_1, \ldots, t_{\lceil \lg k \rceil} \ \exists s, s' \ | \\ & T(s, s') \wedge \bigwedge_{j=1}^{k} \left\{ t^k(j) \rightarrow [(s = s^{j-1}) \wedge (s' = s^j)] \right\} \end{aligned} \quad (2)$$

3.1 Space Requirements

For a given k and for $1 \leq j \leq k$, let $l^k(j)$ denote the length of the path from the root to the j^{th} leaf in the tree associated with $\mathtt{MUX}_k$, where leaves are ordered from left to right. Each conjunct of the form $t^k(j) \rightarrow (s = s^{j-1})$ and $t^k(j) \rightarrow (s' = s^j)$ in the MUXes can be expressed using $2b$ clauses of $l^k(j) + 2$ literals. Hence, the number of literals in the encoding of a $\mathtt{MUX}_k$ is given by:

$$2b \cdot \left(\sum_{j=1}^{k} (l^k(j) + 2) \right) = 2b \cdot \left(2k + \sum_{j=1}^{k} l^k(j) \right)$$

The summation of the lengths of all paths to the leaves in the decision tree, $\sum_{j=1}^{k} l^k(j)$, can be expressed as the following closed-form formula:

$$\sum_{j=1}^{k} l^k(j) = k \cdot \lceil \lg k \rceil + k - 2^{\lceil \lg k \rceil}$$

by considering k paths of length $\lceil \lg k \rceil$, and subtracting the number of leaves that are not at the lowest tree level.

Adding the literals in the transition relation T and the two MUXes, the total number of literals $\mathcal{L}$ in the matrix of Equation 2 is given by:

$$\mathcal{L} \quad = \quad lit(T) + 4b \cdot \left[k \cdot (\lceil \lg k \rceil + 3) - 2^{\lceil \lg k \rceil} \right] \quad (3)$$

where $lit(T)$ denotes the number of literals in the CNF representation of T. This corresponds to $\Theta(lit(T) + b \cdot k \cdot \lg k)$ literals, as opposed to $\Theta(k \cdot lit(T))$ literals in a SAT encoding. Since $lit(T)$ is practically much larger than k and b, this leads to a significantly reduced problem size.

4 Time-Frame Windowing

Equation 2 gives a QBF encoding for an ILA using a single copy of the transition relation, as shown in Figure 2. The formulation can be generalized to include an arbitrary number of copies of T, forming a fixed-length *window* of explicitly unfolded time-frames. This is illustrated using the hardware construction in Figure 4.

This parameterization of the basic scheme generates a family of logically equivalent ILA encodings by balancing the role of explicit circuit unrolling and of universal quantification when reasoning on the sequential operation of a design. Considering the two corner cases of this parametrization, setting $\tau = 1$ reduces time-frame windowing to the basic formulation shown in Figure 2 with $(s, s') = (s^{0,1}, s^{1,1})$, while setting $\tau = k$ degenerates the proposed QBF encoding to the SAT encoding of the ILA shown in Figure 1. The inclusion of explicitly unrolled copies of T in the window enhances the ability of the QBF solver to make direct inferences spanning a number of contiguous time-frames. We show that a non-trivial property of this family of encodings is the existence of an optimum trade-off in terms of minimizing the size of the matrix of the encoding.

In more detail, instead of a current and a next-state (s, s') getting assigned to contiguous current and next-states in the ILA, for a window of size τ there are $\tau + 1$ states $s^{0,\tau}, s^{1,\tau}, \ldots, s^{\tau,\tau}$ getting assigned to sets of $\tau + 1$ contiguous ILA states at a time. Given a bound k, $\lceil \frac{k}{\tau} \rceil$

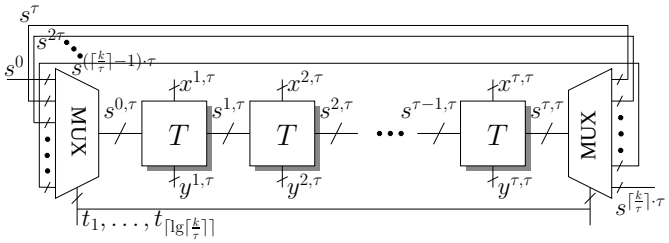


Fig. 4. A time-frame window of size τ

windows of size τ are needed to cover all time-frames. Therefore, a generalized $t = \langle t_1, t_2, \ldots, t_{\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil} \rangle$ for any $\tau \geq 1$ now denotes a *window select* vector which, for each assignment, selects a different window using a similar MUX-based scheme as before. The left and right MUXes in Figure 4 can be respectively formalized as follows:

$$\begin{aligned} &\mathrm{MUX}_{\lceil \frac{k}{\tau} \rceil}(s^{0,\tau}, t, \langle s^0, s^\tau, \ldots, s^{(\lceil \frac{k}{\tau} \rceil - 1)\cdot\tau} \rangle) \quad = \\ &\quad \bigwedge_{j=1}^{\lceil \frac{k}{\tau} \rceil} [t^{\lceil \frac{k}{\tau} \rceil}(j) \rightarrow (s^{0,\tau} = s^{(j-1)\cdot\tau})] \\ &\mathrm{MUX}_{\lceil \frac{k}{\tau} \rceil}(s^{\tau,\tau}, t, \langle s^\tau, s^{2\tau}, \ldots, s^{\lceil \frac{k}{\tau} \rceil \cdot \tau} \rangle) \quad = \\ &\quad \bigwedge_{j=1}^{\lceil \frac{k}{\tau} \rceil} [t^{\lceil \frac{k}{\tau} \rceil}(j) \rightarrow (s^{\tau,\tau} = s^{j\cdot\tau})] \end{aligned}$$

Note that the total number of considered time-frames is $\lceil \frac{k}{\tau} \rceil \cdot \tau = \Theta(k)$. The QBF encoding of the ILA using a window of size τ is given as:

$$\begin{aligned} &T^{\lceil \frac{k}{\tau} \rceil \cdot \tau}(s^0, s^{\lceil \frac{k}{\tau} \rceil \cdot \tau}) \quad = \quad \exists s^\tau, s^{2\tau}, \ldots, s^{(\lceil \frac{k}{\tau} \rceil - 1)\cdot\tau} \\ &\forall t_1, \ldots, t_{\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil} \ \exists s^{0,\tau}, s^{1,\tau}, \ldots, s^{\tau,\tau} \mid \bigwedge_{j=1}^{\tau} T(s^{j-1,\tau}, s^{j,\tau}) \\ &\wedge \bigwedge_{j=1}^{\lceil \frac{k}{\tau} \rceil} \left\{ t^{\lceil \frac{k}{\tau} \rceil}(j) \rightarrow [(s^{0,\tau} = s^{(j-1)\cdot\tau}) \wedge (s^{\tau,\tau} = s^{j\cdot\tau})] \right\} \quad (4) \end{aligned}$$

The matrix of this encoding is equivalent to the construction shown in Figure 4.

In Equation 4, the states $\{s^j | j \bmod \tau \not\equiv 0\}$ are not available in the outermost existential scope of the prefix. Consequently, if $j > 0$ is not a multiple of τ, it is not possible to express a constraint $C(s^j)$ by simply conjuncting it with the matrix. Instead, it can be observed that state s^j corresponds to state $s^{1+(j-1) \bmod \tau, \tau}$ in the $\lceil \frac{j}{\tau} \rceil^{th}$ window. Therefore, the implication:

$$t^{\lceil \frac{k}{\tau} \rceil}(\lceil \frac{j}{\tau} \rceil) \rightarrow C(s^{1+(j-1) \bmod \tau, \tau}) \quad (5)$$

expresses $C(s^j)$. In particular, if $k \bmod \tau \not\equiv 0$, in order to get $T^k(s^0, s^k)$, the implication $t^{\lceil \frac{k}{\tau} \rceil}(\lceil \frac{k}{\tau} \rceil) \rightarrow (s^k = s^{1+(k-1) \bmod \tau, \tau})$ which "extracts" s^k must be conjuncted to the matrix of Equation 4.

4.1 Space Requirements

Increasing the size of the window does not necessarily increase the size of the matrix. In fact, each MUX in Figure 4 can now be expressed using $2b \cdot \lceil \frac{k}{\tau} \rceil$ clauses with at most $\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil + 2$ literals in each clause because the number of inputs to each MUX is $\lceil \frac{k}{\tau} \rceil$. This corresponds to a reduction from $\Theta(k \cdot \lg k)$ to $\Theta(\frac{k}{\tau} \cdot \lg \frac{k}{\tau})$ in the number of literals in the CNF representation of a $\mathtt{MUX}_{\lceil \frac{k}{\tau} \rceil}$ compared to the CNF representation of each $\mathtt{MUX}_k$ in Figure 2.

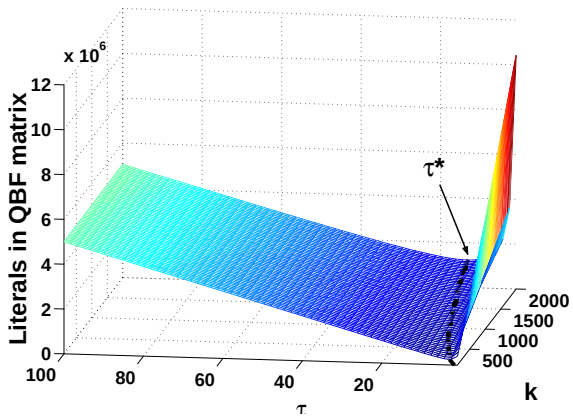


Fig. 5. Number of QBF matrix literals vs. k and τ when $lit(T) = 50\,000, b = 100$

Using a similar reasoning as before, the total number of literals of $\mathcal{L}$ in the CNF matrix of Equation 4, which we use as a figure of merit for the matrix size, is:

$$\mathcal{L} \quad = \quad \tau \cdot lit(T) + 4b \cdot \left[\lceil \frac{k}{\tau} \rceil \cdot (\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil + 3) - 2^{\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil} \right] \quad (6)$$

The window size $\tau^* \in \{1, 2, \ldots, k\}$ minimizing $\mathcal{L}$ can be found numerically. In the following, we want to show that τ^* can be non-trivial, i.e., it is possible to have $\tau^* \in \{2, 3, \ldots, k-1\}$. To that end, we first prove that $\mathcal{L}$ is convex with respect to τ, disregarding the ceilings:

$$\frac{\partial^2 \mathcal{L}}{\partial \tau^2} = 4b \cdot k \cdot \left(\frac{2}{\tau^3} \cdot \lg \frac{k}{\tau} + \frac{7}{\tau} \right) \ > \ 0 \quad \text{for} \quad \tau \leq k$$

Next, in order for $\tau^* \in \{2, 3, \ldots, k-1\}$, it is sufficient to have $\frac{\partial \mathcal{L}}{\partial \tau}|_{\tau=2} \leq 0$ and $\frac{\partial \mathcal{L}}{\partial \tau}|_{\tau=k-1} \geq 0$. This gives:

$$4b \cdot \frac{k}{(k-1)^2} \cdot \left(\lg \frac{k}{k-1} + 3 \right) \leq lit(\tau) \leq b \cdot k \cdot (\lg k + 2)$$

which is satisfied for typical values of b and $lit(\tau)$ and reasonably large k.

Figure 5 shows a three-dimensional plot of Equation 6 versus τ and k, using a transition relation with $lit(T) = 50\,000$ literals in its CNF representation and $b = 100$ state variables. This corresponds to a circuit with roughly $8\,000$ primitive gates. The values of τ^* minimizing the number of literals in the matrix are highlighted in the figure for each value of k. Figure 5 clearly demonstrates the matrix size reduction made possible by parametrizing τ, as opposed to using a default value of $\tau = 1$.

5 Bounded Model Checking

Model checking is concerned with verifying (or falsifying) safety, liveness and other properties in a finite-

state system. Bounded Model Checking (BMC) [13], [18] replicates the transition relation for a bounded number of cycles k, which can be incremented in search for *counter-examples* with k state transitions that violate a given property. This essentially constitutes an ILA construction conjuncted with additional property-specific constraints which are given to a SAT solver. In this work, we consider safety properties. Another model checking technique that leverages the ILA is k-induction [31]. In this section, we encode both BMC and k-induction in QBF using the proposed framework.

5.1 BMC using SAT

Given a set of bad states, determining whether a bad state is reachable in k time-frames starting from a valid initial state can be formulated as follows in SAT [18]:

$$\exists s^0, s^1, \ldots, s^k \quad | \quad I(s^0) \wedge \bigwedge_{j=1}^{k} T(s^{j-1}, s^j) \wedge B(s^k) \qquad (7)$$

where $I(s)$ denotes the predicate recognizing valid initial states and $B(s)$ is the predicate recognizing bad states. To determine whether a bad state is reachable in at most k time-frames, it suffices to replace $B(s^k)$ in Equation 7 by $\bigvee_{j=0}^{k} B(s^j)$.

Given Equation 7, a SAT solver either returns a counter-example of a sequence of states leading to a bad state (i.e., returns SAT), or it proves that a bad state can not be reached in k time-frames (UNSAT). In principle, BMC is complete as it can prove that no bad state can be reached if a large enough bound k is used. However, as k increases, SAT-based BMC may require excessive memory due to the underlying ILA size [18].

5.2 BMC using QBF

The BMC formulation in Equation 7 can be re-written in QBF using the ILA encoding of Equation 2 as follows:

$$\exists s^0, s^1, \ldots, s^k \quad \forall t \quad \exists s, s' \quad | \quad I(s^0) \wedge T(s, s') \wedge \bigwedge_{j=1}^{k} \left\{ t^k(j) \rightarrow [(s = s^{j-1}) \wedge (s' = s^j)] \right\} \wedge B(s^k) \qquad (8)$$

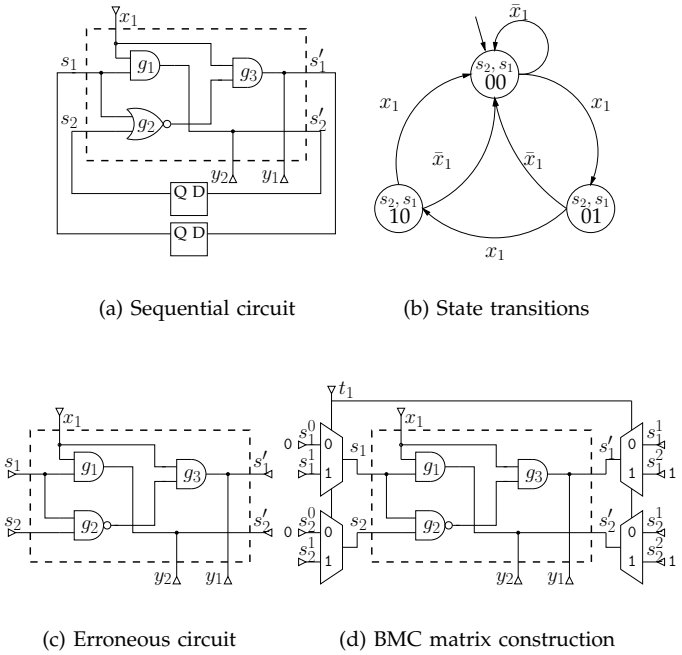
The above equation is obtained by constraining the initial and final states of the ILA to $I(s^0)$ and $B(s^k)$, respectively.

Using Equation 4 and assuming that $k \bmod \tau \equiv 0$, the QBF-based BMC encoding can be generalized for a window of size τ as follows:

$$\exists s^0, s^\tau, \ldots, s^k \, \forall t \, \exists s^{0,\tau}, \ldots, s^{\tau,\tau} \mid I(s^0) \wedge \bigwedge_{j=1}^{\tau} T(s^{j-1,\tau}, s^{j,\tau}) \wedge \bigwedge_{j=0}^{\frac{k}{\tau}} \left\{ t^{\frac{k}{\tau}}(j) \rightarrow [(s^{0,\tau} = s^{(j-1)\cdot\tau}) \wedge (s^{\tau,\tau} = s^{j\cdot\tau})] \right\} \wedge B(s^k) \qquad (9)$$

If k is not a multiple of τ, s^k will not be available in the first scope of the prefix in Equation 9. As described in Equation 5, $B(s^k)$ in Equation 9 would be replaced by $t^{\lceil \frac{k}{\tau} \rceil}(\lceil \frac{k}{\tau} \rceil) \rightarrow B(s^{1+(k-1) \bmod \tau, \tau})$. For each unit clause in $B(s^k)$, this implication can be expressed using a clause of at most $\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil + 1$ literals. For instance, given $B(s) = \bar{s}_1 \wedge s_2$, $k = 5$ and $\tau = 2$, $B(s^5)$ in Equation 9 becomes:

$$\begin{aligned} t^3(3) \rightarrow B(s^{1,2}) \quad &= \quad t_1 \rightarrow (\bar{s}_1^{1,2} \wedge s_2^{1,2}) \\ &= \quad (\bar{t}_1 \vee \bar{s}_1^{1,2}) \wedge (\bar{t}_1 \vee s_2^{1,2}) \end{aligned}$$



(a) Sequential circuit (b) State transitions

(c) Erroneous circuit (d) BMC matrix construction

Fig. 6. BMC example

In the following example, the given QBF-based formulation is generated in bit-level detail for a BMC instance.

Example 1 *Figure 6(a) depicts a modulo-3 incrementer where the output y_2y_1 is a binary number incremented if and only if the input $x_1 = 1$. Figure 6(b) shows the circuit's state transition diagram with initial state $\langle s_1, s_2 \rangle = \langle 0, 0 \rangle$. Note that state $\langle s_1, s_2 \rangle = \langle 1, 1 \rangle$ is unreachable.*

Given the incorrect implementation in Figure 6(c) (gate g_2 is NAND instead of NOR), a BMC problem can be formulated to ask whether the bad state $\langle s_1, s_2 \rangle = \langle 1, 1 \rangle$ is reachable for a given bound. $I(s) = \bar{s}_1 \wedge \bar{s}_2$ and $B(s) = s_1 \wedge s_2$ are given. The QBF-based BMC formulation for $k = 2$ (it is UNSAT for $k = 1$) using $\tau = 1$, along with the corresponding construction are given by Equation 10 and Figure 6(d), respectively.

$$\begin{aligned} &\exists s_1^0, s_2^0, s_1^1, s_2^1, s_1^2, s_2^2 \quad \forall t_1 \quad \exists s_1, s_2, s_1', s_2' \quad | \\ &\bar{s}_1^0 \wedge \bar{s}_2^0 \wedge T(\langle s_1, s_2 \rangle, \langle s_1', s_2' \rangle) \wedge s_1^2 \wedge s_2^2 \wedge \\ &\left\{ \bar{t}_1 \rightarrow [(s_1 = s_1^0) \wedge (s_2 = s_2^0) \wedge (s_1' = s_1^1) \wedge (s_2' = s_2^1)] \right\} \wedge \\ &\left\{ t_1 \rightarrow [(s_1 = s_1^1) \wedge (s_2 = s_2^1) \wedge (s_1' = s_1^2) \wedge (s_2' = s_2^2)] \right\} \end{aligned} \qquad (10)$$

When $t_1 = 0$, Figure 6(d) simulates the first ILA time-frame, where $\langle s_1^0, s_2^0 \rangle = \langle 0, 0 \rangle$ is the initial state, and $\langle s_1^1, s_2^1 \rangle$ represents the next-state. When $t_1 = 1$, the same $\langle s_1^1, s_2^1 \rangle$

constrains the current-state of the second time-frame, while $\langle s_1^2, s_2^2 \rangle = \langle 1, 1 \rangle$ *constrains the (bad) final state. A counter-example reaching the bad state is the sequence of states* $\langle s_1, s_2 \rangle$*:* $\langle 0, 0 \rangle \rightarrow \langle 1, 0 \rangle \rightarrow \langle 1, 1 \rangle$*, corresponding to the primary input sequence* $\langle x_1^1, x_1^2 \rangle = \langle 1, 1 \rangle$.

5.3 Modeling k-Induction

The method of k-induction [31] uses an inductive proof consisting of a base-case and an inductive-step to verify (or falsify) properties. In this subsection, we express this proof method using our QBF-based ILA encoding.

The base-case asks whether a bad state is reachable within $k-1$ transitions of the initial states. This can be expressed as a BMC instance, as described in Subsection 5.1. The inductive-step states that if the bad state $B(s)$ is not reached within any sequence of $k-1$ unique state transitions $s^0 \rightarrow \cdots \rightarrow s^{k-1}$, then it is unreachable in k unique state transitions. This can be expressed as the following SAT instance [31]:

$$\exists s^0, s^1, \ldots, s^k \quad | \quad \bigwedge_{j=1}^{k} T(s^{j-1}, s^j) \wedge \bigwedge_{j=0}^{k-1} \overline{B}(s^j) \wedge B(s^k) \wedge \bigwedge_{0 \leq i < j < k} s^i \neq s^j \qquad (11)$$

$B(s)$ is unreachable in *any* number of state transitions, if the base-case holds and Equation 11 is UNSAT. Otherwise, k is increased in both the base-case and the inductive-step until the property is verified or a counter-example is found in the base-case. A discussion on the implementation of the uniqueness constraints $\bigwedge_{0 \leq i < j < k} s^i \neq s^j$ can be found in [27].

The k-induction formulation for the inductive-step given in Equation 11 can be translated into QBF using the proposed ILA encoding of Equation 2 as follows:

$$\exists s^0, s^1, \ldots, s^k \ \forall t \ \exists s, s' \quad | \quad T(s, s') \wedge \bigwedge_{j=0}^{k-1} \overline{B}(s^j) \wedge B(s^k) \wedge \bigwedge_{j=1}^{k} \{ [t^k(j) \leftrightarrow (s = s^{j-1})] \wedge [t^k(j) \rightarrow (s' = s^j)] \} \qquad (12)$$

The equivalences in the terms $\bigwedge_{j=1}^{k} [t^k(j) \leftrightarrow (s = s^{j-1})]$, which enforce the uniqueness constraints, are based on a similar encoding given in [27].

5.4 Previous Work

In [26], a QBF-based BMC encoding is given, which introduces two universal state variables, and hence $\Theta(b)$ universal bits. A k-induction formulation using QBF is given in [27], which uses k universal bits to traverse the ILA time-frames using a forced one-hot encoding. Furthermore, the non-copying iterative squaring encoding studied in [27] defines $T^{2k}(s, s')$ recursively as a function of T^k. This corresponds to a BMC formulation with $\Theta(\lg k)$ universal variables and $\Theta(\lg k)$ quantifier scopes. The authors in [27] conclude that QBF solvers are not taking advantage of those compact encodings to improve performance.

Our modeling of BMC using QBF stems from the general-purpose ILA representation in Section 3. It differs from previous work due to the novel MUX-based implementation. Additionally, the proposed BMC formulation introduces at most $O(\lg k)$ universal variables, thus preserving the advantage of the non-copying iterative squaring method while using a constant number of quantifier scopes ($\exists\forall\exists$) and a linear representation of time. Another major contribution is the time-frame windowing technique, which allows further size compression and boosts the inference power of the QBF solver. These unique characteristics seem to have a significant impact on performance, as shown in Section 8.

Finally, our encoding is not limited to BMC. It provides a general platform for a flexible and performance-driven represention of sequential circuits in a multitude of CAD problems, as shown in the sections that follow.

6 Design Debugging

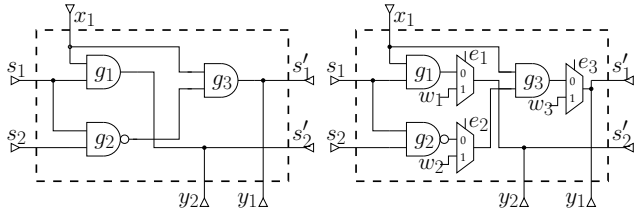
Design debugging starts after a functional verification engine has produced a counter-example indicating that the design is erroneous. Its objective is to locate all possibly erroneous lines in the netlist [21], [32].

6.1 Design Debugging using SAT

SAT-based design debugging [21] takes in the erroneous circuit T and the expected behavior of the counter-example where T fails. It encodes the debugging problem as a SAT instance whose satisfying assignments correspond to the potential error locations in the circuit.

The following steps, relevant to the work here, are performed for a counter-example of k cycles [21]:

i) For each gate g_i in the circuit, an error modeling MUX with select line e_i is introduced. This is shown in Figure 7(b) for gates g_1, g_2 and g_3 of Figure 7(a). An inactive MUX select line ($e_i = 0$) does not modify the circuit, whereas an active select line ($e_i = 1$) disconnects g_i from its fanouts and replaces it with a new unconstrained input w_i, which can freely "fix" any potential error at the output of gate g_i. The transition relation of this enhanced circuit is denoted by $T_{en}(s, s', \langle x, w, e \rangle, y)$,



(a) Erroneous circuit T (b) Enhanced circuit T_{en}

Fig. 7. Enhancing T with error models

where w and e respectively denote the unconstrained inputs and select lines of the error modeling MUXes.

ii) The enhanced circuit T_{en} is replicated as an ILA of length k: $\bigwedge_{j=1}^{k} T_{en}(s^{j-1}, s^j, \langle x^j, w^j, e\rangle, y^j)$, such that the error modeling MUXes corresponding to the same gate share the same select line in all time-frames.

iii) A set of constraints $\Phi_C(s^0, \langle x^1, \ldots, x^k\rangle, \langle y^1, \ldots, y^k\rangle)$ is added to ensure that the initial-state, primary inputs and primary outputs of the ILA match the expected circuit behavior for the counter-example.

iv) An error cardinality constraint $\Phi_N(e)$ is added to ensure that the number of activated select lines is equal to the number of errors N.

SAT-based debugging returns N potentially erroneous gates by activating the corresponding bits in e to match the expected circuit behavior. N is initialized to 1 and incremented until Equation 13 becomes SAT:

$$\exists e, s^0, s^1, \ldots, s^k, X, W, Y \mid \bigwedge_{j=1}^{k} T_{en}(s^{j-1}, s^j, \langle x^j, w^j, e\rangle, y^j) \wedge \Phi_C(s^0, X, Y) \wedge \Phi_N(e) \qquad (13)$$

Note that an all-solution SAT solver can be used to return all satisfying assignments to e, or equivalently, all N-tuples of potentially erroneous gates.

6.2 Design Debugging using QBF

The design debugging formulation from Equation 13 can be translated into QBF using the ILA encoding in Equation 2 by adding two MUXes which help constrain the primary inputs and outputs at each time-frame:

$$\begin{aligned}&\exists e, s^0, s^1, \ldots, s^k, X, Y \ \forall t \ \exists s, s', x, w, y \mid \\ &\bigwedge_{j=1}^{k} t^k(j) \rightarrow [(s = s^{j-1}) \wedge (s' = s^j)] \wedge \\ &\bigwedge_{j=1}^{k} t^k(j) \rightarrow [(x = x^j) \wedge (y = y^j)] \wedge \\ &T_{en}(s, s', \langle x, w, e\rangle, y) \wedge \Phi_C(s^0, X, Y) \wedge \Phi_N(e) \end{aligned} \qquad (14)$$

Figure 8 gives the construction corresponding to Equation 14. The four MUXes assign the current-state, next-state, primary inputs and primary outputs of T_{en} according to the time-frame select variables t. Constraining the initial-state s^0, inputs X and outputs Y in Figure 8 according to $\Phi_C(s^0, X, Y)$, and adding the error cardinality constraint $\Phi_N(e)$ yields the matrix of the design debugging QBF formulation given by Equation 14.

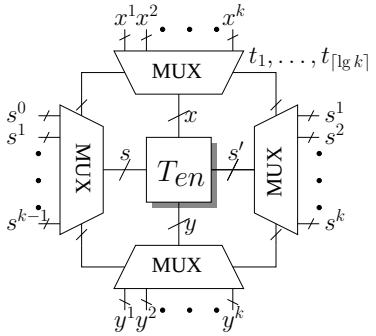
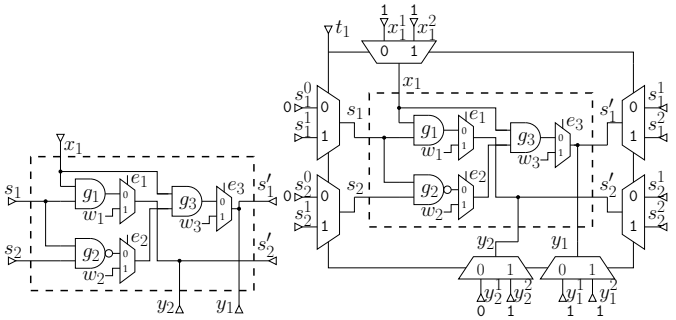


Fig. 8. Design debugging construction



(a) Enhanced circuit (b) Design debugging construction

Fig. 9. Design debugging example

The following example illustrates the above concepts.

Example 2 *Consider the correct and erroneous designs in Figure 6(a) and Figure 6(c). In Example 1, BMC yields a 2-time-frame counter-example with initial state* $\langle s_1^0, s_2^0\rangle = \langle 0, 0\rangle$ *and the sequence of inputs* $\langle x_1^1, x_1^2\rangle = \langle\langle 1\rangle, \langle 1\rangle\rangle$. *According to the correct state transition diagram shown in Figure 6(b) (the states and outputs have the same values), the* expected *output sequence corresponding to this counter-example is* $\langle\langle y_1^1, y_1^2\rangle, \langle y_1^2, y_2^2\rangle\rangle = \langle\langle 1, 0\rangle, \langle 0, 1\rangle\rangle$. *Therefore,* $\Phi_C(s^0, X, Y) = \bar{s}_1^0 \wedge \bar{s}_2^0 \wedge x_1^1 \wedge x_1^2 \wedge y_1^1 \wedge \bar{y}_2^1 \wedge \bar{y}_1^2 \wedge y_2^2$. *The QBF-based design debugging formulation with* $N = 1$, *along with its corresponding construction are shown respectively in Equation 15 and Figure 9(b).*

$$\begin{aligned}&\exists e_1, e_2, e_3, s_1^0, s_2^0, s_1^1, s_2^1, s_1^2, s_2^2, x_1^1, x_1^2, y_1^1, y_2^1, y_1^2, y_2^2 \ \forall t_1 \\ &\exists s_1, s_2, s_1', s_2', x_1, w_1, w_2, w_3, y_1, y_2 \mid \\ &T_{en}(\langle s_1, s_2\rangle, \langle s_1', s_2'\rangle, \langle x_1, \langle w_1, w_2, w_3\rangle, \langle e_1, e_2, e_3\rangle\rangle, \langle y_1, y_2\rangle) \\ &\{\bar{t}_1 \rightarrow [(s_1 = s_1^0) \wedge (s_2 = s_2^0) \wedge (s_1' = s_1^1) \wedge (s_2' = s_2^1)]\} \wedge \\ &\{\bar{t}_1 \rightarrow [(x_1 = x_1^1) \wedge (y_1 = y_1^1) \wedge (y_2 = y_2^1)]\} \wedge \\ &\{t_1 \rightarrow [(s_1 = s_1^1) \wedge (s_2 = s_2^1) \wedge (s_1' = s_1^2) \wedge (s_2' = s_2^2)]\} \wedge \\ &\{t_1 \rightarrow [(x_1 = x_1^2) \wedge (y_1 = y_1^2) \wedge (y_2 = y_2^2)]\} \wedge \\ &\bar{s}_1^0 \wedge \bar{s}_2^0 \wedge x_1^1 \wedge x_1^2 \wedge y_1^1 \wedge \bar{y}_2^1 \wedge \bar{y}_1^2 \wedge y_2^2 \wedge (e_1 + e_2 + e_3 = 1) \end{aligned} \qquad (15)$$

Figure 9(b) shows the $\Phi_C(s^0, X, Y)$ *constraints applied at the initial-state, inputs and outputs of the circuit. In Equation 15, the only satisfying assignment to the select lines is* $\langle e_1, e_2, e_3\rangle = \langle 0, 1, 0\rangle$, *indicating that gate* g_2 *is potentially erroneous.*

In [32], QBF is used in a debugging framework for a different end, namely handling multiple counter-examples using an ILA. In this sense, the work here and that of [32] are complementary and non-overlapping.

7 SEQUENTIAL ATPG

ATPG is the process of generating test vectors to detect faults in a logic circuit. Traditional ATPG engines generate tests for single stuck-at faults. A circuit line is stuck-at-0 (stuck-at-1) if it always assumes a constant value of 0 (1). A test vector that detects a stuck-at fault on some circuit line must produce different values at one or more outputs in the presence of that fault. Sequential ATPG has been tackled with several approaches including SAT-based ones that use the ILA representation [20].

7.1 Sequential ATPG using SAT

Let $T_c(s_c, s'_c, x_c, y_c)$ ($T_f(s_f, s'_f, x_f, y_f)$) denote the transition relation of the fault-free (faulty) circuit. The sequential ATPG problem can be formulated as the following SAT instance:

$$\exists s_c^0, s_c^1, \ldots, s_c^k, s_f^0, s_f^1, \ldots, s_f^k, X, Y_c, Y_f \quad | \quad (s_c^0 = s_f^0) \wedge \bigwedge_{j=1}^{k} [T_c(s_c^{j-1}, s_c^j, x^j, y_c^j) \wedge T_f(s_f^{j-1}, s_f^j, x^j, y_f^j)] \wedge \bigvee_{j=1}^{k} (y_c^j \neq y_f^j) \tag{16}$$

where $Y_c = \langle y_c^1, \ldots, y_c^k \rangle$ ($Y_f = \langle y_f^1, \ldots, y_f^k \rangle$) denotes the output sequence of the fault-free (faulty) circuit.

Equation 16 searches for the common sequence of inputs X feeding to both T_c and T_f, which causes at least one primary output in Y_c to be different from Y_f.

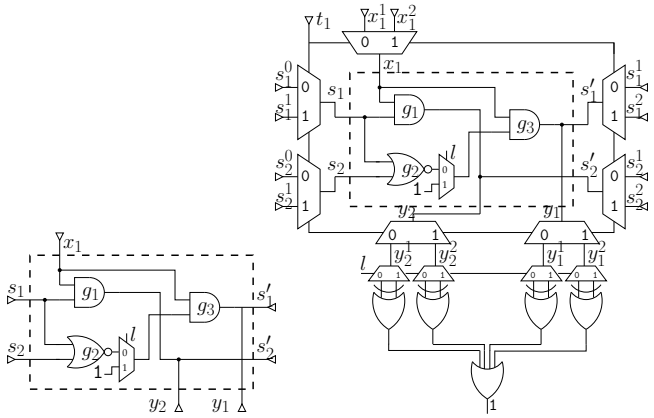
7.2 Sequential ATPG using QBF

Using the ILA formulation of Equation 2, Equation 16 can be encoded in QBF using $T = T_c \wedge T_f$. However, it is possible to further compress the encoding and use a single transition relation by taking advantage of universal quantification as follows.

An enhanced circuit T_{en} is constructed by introducing a MUX at the stuck-at-fault location that chooses between the correct and faulty line using a select line l. T_{en} simulates T_c when $l = 0$ and T_f when $l = 1$. The sequential ATPG problem is concerned with the existence of an initial state s^0, a common sequence of inputs X and *different* outputs Y_c and Y_f, such that for both values of l, there exist states $s^0, s^1, \ldots, s^k$, such that for a given time-frame j, the primary outputs of T_{en} evaluate to y_c^j if $l = 0$ and y_f^j if $l = 1$.

This is formalized in the following QBF expression:

$$\exists s^0, X, Y_c, Y_f \quad \forall l \quad \exists s^1, \ldots, s^k, Y \quad \forall t \quad \exists s, s', x, y \quad | \quad T_{en}(s, s', \{x, l\}, y) \wedge \bigwedge_{j=1}^{k} [(l \rightarrow y^j = y_c^j) \wedge (\bar{l} \rightarrow y^j = y_f^j)] \wedge \bigwedge_{j=1}^{k} t^k(j) \rightarrow [(s = s^{j-1}) \wedge (s' = s^j)] \wedge \bigwedge_{j=1}^{k} t^k(j) \rightarrow [(x = x^j) \wedge (y = y^j)] \wedge \bigvee_{j=1}^{k} (y_c^j \neq y_f^j) \tag{17}$$



(a) Enhanced circuit (b) Sequential ATPG construction

Fig. 10. Sequential ATPG example

Example 3 *Consider the circuit in Figure 6(a) and a stuck-at-1 at the output of gate g_2. The corresponding enhanced circuit T_{en} is given in Figure 10(a). The QBF encoding of the sequential ATPG formulation for test sequences of $k = 2$ time-frames, along with its corresponding construction are shown respectively in Equation 18 and Figure 10(b).*

$$\begin{aligned}
&\exists s_1^0, s_2^0, x_1^1, x_1^2, y_{c1}^1, y_{c2}^1, y_{c1}^2, y_{c2}^2, y_{f1}^1, y_{f2}^1, y_{f1}^2, y_{f2}^2 \ \forall l \\
&\exists s_1^1, s_2^1, s_1^2, s_2^2, y_1^1, y_2^1, y_1^2, y_2^2 \ \forall t_1 \ \exists s_1, s_2, s'_1, s'_2, x_1, y_1, y_2 \ | \\
&T_{en}(\langle s_1, s_2 \rangle, \langle s'_1, s'_2 \rangle, \langle x_1, l_1 \rangle, \langle y_1, y_2 \rangle) \wedge \\
&\{\bar{t}_1 \rightarrow [(s_1 = s_1^0) \wedge (s_2 = s_2^0) \wedge (s'_1 = s_1^1) \wedge (s'_2 = s_2^1)]\} \wedge \\
&\{\bar{t}_1 \rightarrow [(x_1 = x_1^1) \wedge (y_1 = y_1^1) \wedge (y_2 = y_2^1)]\} \wedge \\
&\{t_1 \rightarrow [(s_1 = s_1^1) \wedge (s_2 = s_2^1) \wedge (s'_1 = s_1^2) \wedge (s'_2 = s_2^2)]\} \wedge \\
&\{t_1 \rightarrow [(x_1 = x_1^2) \wedge (y_1 = y_1^2) \wedge (y_2 = y_2^2)]\} \wedge \\
&\{l \rightarrow [(y_1^1 = y_{c1}^1) \wedge (y_2^1 = y_{c2}^1) \wedge (y_1^2 = y_{c1}^2) \wedge (y_2^2 = y_{c2}^2)]\} \wedge \\
&\{\bar{l} \rightarrow [(y_1^1 = y_{f1}^1) \wedge (y_2^1 = y_{f2}^1) \wedge (y_1^2 = y_{f1}^2) \wedge (y_2^2 = y_{f2}^2)]\} \wedge \\
&((y_{c1}^1 \neq y_{f1}^1) \vee (y_{c2}^1 \neq y_{f2}^1) \vee (y_{c1}^2 \neq y_{f1}^2) \vee (y_{c2}^2 \neq y_{f2}^2))
\end{aligned} \tag{18}$$

The variables of interest for the sequential ATPG problem are the initial state $\langle s_1^0, s_2^0 \rangle$ and the common input sequence $\langle x_1^1, x_1^2 \rangle$. A satisfying assignment for Equation 18 is the test sequence $\langle s_1^0, s_2^0, x_1^1, x_1^2 \rangle = \langle 0, 0, 1, 1 \rangle$, for which $y_{c1}^2 = 0$ differs from $y_{f1}^2 = 1$.

8 EXPERIMENTAL RESULTS

A C++ software module is implemented for each of the BMC, design debugging and sequential ATPG applications, that encodes problem instances in QBF as discussed in this work. The generated instances are solved using sKizzo [24], a state-of-the-art hybrid QBF solver based on symbolic skolemization. Table 2 shows the circuit characteristics of nine industrial designs from OpenCores.org [33] used to construct these instances. The columns in Table 2 show the name of the design, its gate-count, its number of DFFs (b), and the number of literals in its transition relation ($lit(T)$). All experiments

TABLE 1
BMC using QBF

Circuit Name	SAT # solved	SAT time (sec)	SAT mem (MB)	(QBF,$\tau=1$) # solved	(QBF,$\tau=1$) time (sec)	(QBF,$\tau=1$) mem (MB)	(QBF,$\tau=16$) # solved	(QBF,$\tau=16$) time (sec)	(QBF,$\tau=16$) mem (MB)	(QBF,τ^*) avg τ^*	(QBF,τ^*) # solved	(QBF,τ^*) time (sec)	(QBF,τ^*) mem (MB)
AC97	8/12	670.9	295.4	6/12	1 196.8	145.0	12/12	129.9	20.6	14.0	12/12	120.2	17.2
Divider16	8/12	772.7	108.4	8/12	853.4	33.6	10/12	771.9	6.4	12.0	8/12	809.9	4.9
ERP	10/12	347.2	45.3	12/12	67.3	29.9	10/12	385.5	3.5	16.7	10/12	382.3	3.0
ReacTimer	12/12	0.5	3.3	12/12	8.8	1.6	12/12	0.5	0.2	14.0	12/12	0.5	0.2
RSDecoder	2/12	1 666.8	244.1	2/12	1 667.6	52.0	2/12	1 668.7	13.6	8.3	2/12	1 667.4	9.0
SPI	11/12	249.3	34.5	12/12	36.6	7.7	12/12	3.5	2.0	9.0	12/12	3.1	1.4
Aqu	6/12	1 001.9	514.6	11/12	300.0	148.9	12/12	120.3	30.1	12.0	12/12	112.0	24.0
Fibonacci	12/12	0.6	11.0	12/12	24.4	2.6	12/12	1.0	0.6	12.0	12/12	0.9	0.5
AE18	4/12	1 358.3	48.9	6/12	1 194.3	10.0	4/12	1 429.3	2.7	9.0	4/12	1 412.0	1.9

are conducted on a Pentium IV 2.8 GHz Linux platform with 2 GB of memory and a time limit of 2000 seconds.

8.1 BMC and Time-Frame Windowing

BMC problems for safety properties of the form of Equation 7 are considered. For each circuit, six exponentially increasing bounds k of size 32, 64, 128, 256, 512 and 1024 are examined and two manually generated "bad" states are checked for each k: One that is reachable (SAT) and one that is not (UNSAT). The proposed QBF-based formulations are evaluated against a traditional SAT-based encoding solved by MINISAT V1.14 [16], a state-of-the-art SAT solver.

Table 1 compares the results with SAT for three QBF-based BMC encodings with different time-frame windowing schemes. The $(\text{QBF}, \tau = 1)$ encoding does not use time-frame windowing, $(\text{QBF}, \tau = 16)$ uses a fixed window of size $\tau = 16$ irrespective of k, while (QBF, τ^*) uses the window size τ^* which minimizes the number of literals in the formulation according to Equation 6. For each approach, columns # *solved*, *time* and *mem* respectively show the number of solved problem instances out of 12, the average run-time in seconds, and the average memory footprint of the files containing the problem instances in MBs. When averaging the run-times, an unsolved instance is counted as 2000 seconds, which is the time limit. Moreover, under (QBF, τ^*), column avg τ^* gives the average value of τ^* for each circuit.

SAT solves a total of 73 BMC instances, whereas the three different QBF-based windowing schemes $(\text{QBF}, \tau = 1)$, $(\text{QBF}, \tau = 16)$ and (QBF, τ^*) respectively solve 81, 86 and 84 instances out of 108. The most common aborting reason is running out of memory for the SAT approach, and timing-out for the QBF approach. The QBF encodings are respectively 65%, 94% and 95% smaller than the SAT-based formulations on average. Although all three QBF options outperform SAT in the number of solved instances, the effect of time-frame windowing is vital in terms of memory, run-time and the number of problem instances solved. In fact, $(\text{QBF}, \tau = 16)$ solves 5 more instances than $(\text{QBF}, \tau = 1)$, and (QBF, τ^*) uses 85% less memory compared to $(\text{QBF}, \tau = 1)$.

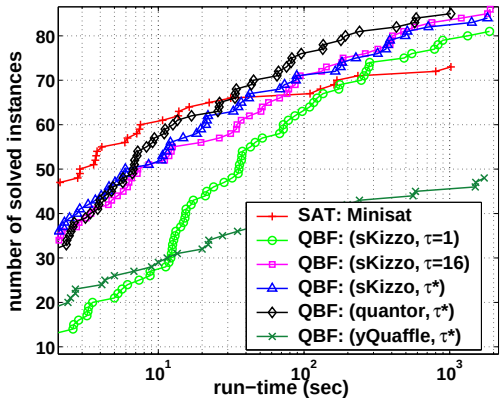


Fig. 11. BMC performance results

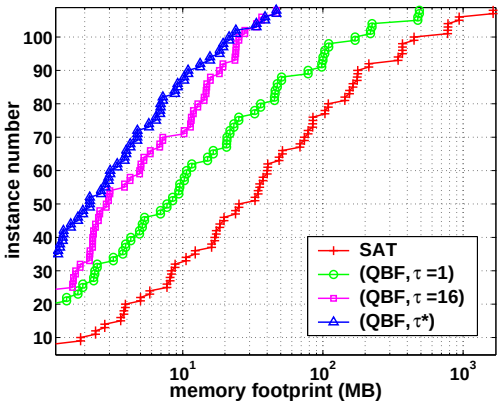


Fig. 12. BMC encoding sizes

TABLE 2
Circuit Characteristics

Circuit Name	Number of Gates	Number of DFFs	$lit(T)$
AC97	15 601	1 452	96 872
Divider16	5 248	388	37 505
ERP	2 449	347	16 248
ReacTimer	265	22	1 360
RSDecoder	12 041	521	74 952
SPI	2 012	90	12 454
Aqu	22 319	1 504	158 360
Fibonacci	652	36	4 062
AE18	2 520	116	17 328

Figure 11 plots the number of solved BMC instances using each encoding as a function of run-time given in a logarithmic scale. It clarifies the QBF versus SAT comparison and it highlights the positive influence of time-frame windowing. Focusing on the QBF solver `sKizzo` which is used to generate the numbers in Table 1, it can be seen that SAT has an initial advantage on smaller instances taking less than 30 seconds to solve. Both $(\texttt{sKizzo}, \tau = 16)$ and $(\texttt{sKizzo}, \tau^*)$ outperform SAT within 80 seconds, while $(\texttt{sKizzo}, \tau = 1)$ outperforms SAT within 300 seconds. All three QBF formulations take advantage of the declining slope of the SAT curve in Figure 11 as soon as the problem instances grow in complexity. Figure 12 compares the problem sizes generated by each encoding. As expected, all three QBF encodings require considerably less memory than SAT encodings, while $(\texttt{sKizzo}, \tau^*)$ achieves maximum compression.

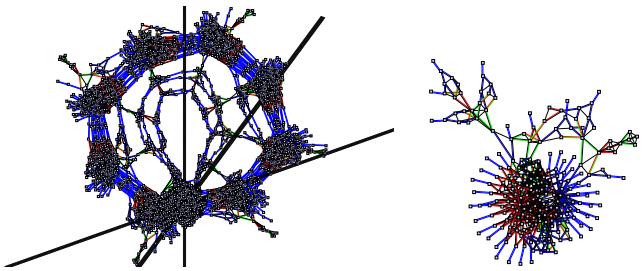
8.1.1 Impact on QBF Reasoning Strategies

In order to investigate the effect of the choice of the QBF solver in achieving these results, the (QBF, τ^*) instances are also run using two other contemporary QBF solvers with different reasoning strategies: `quantor` [23] which is based on Q-resolution and expansion, and `yQuaffle` [22] which is a search-based QBF solver. Table 3 shows the results of these experiments. The first two columns under each of $(\texttt{quantor}, \tau^*)$ and $(\texttt{yQuaffle}, \tau^*)$ show the numbers of solved `SAT` and `UNSAT` instances out of six using the respective QBF solver. The *time* column gives the average run-time for each circuit in seconds. `quantor` solves 85 instances out of 108, which is one more than `sKizzo` using a window of size τ^*, while `yQuaffle` solves only 48 out of 108, considerably less than the SAT approach. Figure 11 includes the plots of the number of solved instances by `quantor` and `yQuaffle`. The search-based QBF solver `yQuaffle` is dominated by both other QBF solvers, as well as the SAT solver. We also observed this trend with other search-based QBF solvers, such as `SQBF` [25].

We give two complementary explanations for this behavior. First, the reason why a resolution-based approach becomes so competitive may be based on the structure of the *interaction graphs* of the matrices for our QBF encodings. The interaction graph of a CNF formula is defined as a graph that contains a node for each variable and an edge connecting any two nodes whose corresponding variables appear in the same clause [34]. Our QBF encodings produce *sparse* interaction graphs, where each node is only connected to a very small percentage of the total set of nodes. This sparseness enables skolemization, resolution and expansion to proceed without immediate memory explosions.

TABLE 3
Search-based and resolution-based QBF for BMC

Circuit Name	(quantor,τ^*) # solved SAT	(quantor,τ^*) # solved UNSAT	(quantor,τ^*) time (sec)	(yQuaffle,τ^*) # solved SAT	(yQuaffle,τ^*) # solved UNSAT	(yQuaffle,τ^*) time (sec)
AC97	6/6	6/6	86.8	0/6	6/6	1 008.1
Divider16	3/6	6/6	536.9	0/6	0/6	–
ERP	4/6	6/6	347.9	0/6	6/6	1 000.1
ReacTimer	6/6	6/6	1.1	5/6	5/6	411.8
RSDecoder	1/6	1/6	1 667.4	0/6	0/6	–
SPI	6/6	6/6	25.8	2/6	6/6	826.9
Aqu	4/6	6/6	392.2	0/6	6/6	1 001.0
Fibonacci	6/6	6/6	1.9	6/6	6/6	326.6
AE18	3/6	3/6	1 143.7	0/6	0/6	–



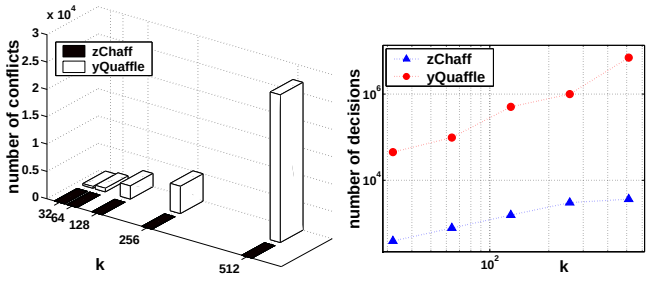
(a) QBF matrix for $\tau = 8, k = 64$ (b) Transition relation

Fig. 13. Interaction graphs for QBF-based BMC of benchmark ReacTimer

Figure 13(a) shows the interaction graph of the QBF matrix in the BMC encoding of ReacTimer with $k = 64$ and $\tau = 8$, drawn using `DPvis` [34]. The cluster of nodes at the bottom of Figure 13(a) corresponds to the two `MUX`es shown in Figure 4. The remaining 8 clusters correspond to the 8 copies of the transition relation T in the window. Each of these copies is only connected to the previous and next copy of the transition relation, and the `MUX` circuitry is only connected to the first and last copies of T, as shown in Figure 4. Furthermore, the interaction graph of each copy of the circuit itself (i.e., of each cluster in Figure 13(a)) is usually sparse because internal gates typically have a limited fan-out. For example, Figure 13(b) shows the interaction graph of one copy of the transition relation T for ReacTimer. Note that these interaction graphs represent the problem before the solving procedure begins.

On the other hand, the pessimistic results for `yQuaffle` and search-based QBF solvers in general can be attributed to the excessive trial-and-error in "guessing" correct ILA state transitions. This is caused by the restriction on the variable decision order to follow the prefix scope order, which forces a search-based QBF solver to first decide on all outermost state variables $\{s^i\}$ before decisions can be made on variables of inner scopes. This can become a recipe for conflicts and severely impede the search process when compared to SAT where solvers can branch on any variable at any given time.

To demonstrate this theoretical observation, we compare the number of conflicts and the number of decisions in `yQuaffle` to those of the `zChaff` SAT solver [15], for five BMC instances on ReacTimer with increasing bounds k. As shown in Figure 14(a), as k increases, `yQuaffle` produces significantly more conflicts due to



(a) Number of conflicts (b) Number of decisions

Fig. 14. Search-based SAT and QBF solver statistics for BMC of ReacTimer

the increasing number of outermost states. Meanwhile, `zChaff` produces almost no conflicts irrespective of k, by propagating the initial state constraints $I(s^0)$ forward and bad state constraints $B(s^k)$ backwards. Figure 14(b) shows the number of decisions in a logarithmic scale for each of `yQuaffle` and `zChaff` to solve these instances as k increases. Clearly, `yQuaffle` requires at least two orders of magnitude more decisions than `zChaff` to solve the same set of problems.

8.1.2 k-Induction

Instances of k-induction problems for safety properties of the form of Equation 11 are considered. For each circuit, bounds of size 32, 64, 128 and 256 are examined and bad states that are unreachable within those bounds (produced `UNSAT` using BMC) are used. The QBF-based approach is compared to a traditional SAT-based encoding solved by MINISAT V1.14.

Table 4 compares the QBF-based and SAT-based schemes. $\tau = 1$ is used in the QBF encodings. For each approach, columns # *solv.*, *time* and *mem* respectively show the number of solved problem instances out of 4, the average run-time in seconds, and the average problem size in MBs. Both approaches solve 24 out of 36 instances, while the QBF encodings are 97% smaller on average. Run-times are comparable when disregarding unsolved instances. The slight deterioration in the performance of the QBF-based method compared to BMC is likely due to the increased search-space of the inductive-step compared to the base-case.

TABLE 4
k-induction using QBF

	SAT			QBF		
Circuit Name	# solv.	time (sec)	mem (MB)	# solv.	time (sec)	mem (MB)
AC97	2/4	1 003.0	1 113.0	4/4	73.7	68.7
Divider16	3/4	581.5	621.5	0/4	–	16.2
ERP	3/4	500.1	599.0	4/4	3.7	14.2
ReacTimer	4/4	2.8	22.5	4/4	0.8	0.8
RSDecoder	1/4	1 502.3	1 048.2	0/4	–	25.0
SPI	4/4	2.9	109.7	4/4	1.3	3.7
Aqu	1/4	1 500.1	1 529.0	4/4	11.5	71.2
Fibonacci	4/4	1.8	40.7	4/4	1.6	1.3
AE18	2/4	1 201.0	538.0	0/4	–	4.8

8.2 Design Debugging

For the debugging problems, `sKizzo` has been purposely modified to an all-solution QBF solver that returns all the valid assignments to e in Equation 14, or equivalently, all possible error sites. The erroneous circuits are created by manually changing the functionality of certain modules to introduce an error. Counter-example sequences are obtained by pseudo-random simulation. For each circuit, six different design debugging problem instances with eight counter-examples for each instance are generated. All solutions are found using $N = 1$, i.e., there is a single erroneous module in each instance. Final results are averaged out over the number of instances. In order to deal with multiple counter-examples, the ideas in [32] are integrated in the approach. The results of the proposed QBF-based formulation using a unit-size window are compared to the SAT-based approach [21] that uses circuit replication with `zChaff` being the underlying SAT solver. Since `sKizzo` internally uses `zChaff` to solve propositional subproblems related to the QBF instance, this provides a fair comparison metric.

Table 5 presents the results of the proposed QBF formulation for design debugging. The second, third and fourth columns respectively show the average counter-example length, the maximum counter-example length and the average number of potentially erroneous modules in the circuit. For each formulation (SAT and QBF), columns # *solved*, *time* and *mem* respectively show the number of solved instances, the average run-time in seconds and the average memory usage of the problem formulation in MBs.

As clearly seen in Table 5, the design debugging results are even more favorable to QBF when compared to SAT. Along with an average of 89% reduction in the memory footprint of the formulation, the run-time performance is

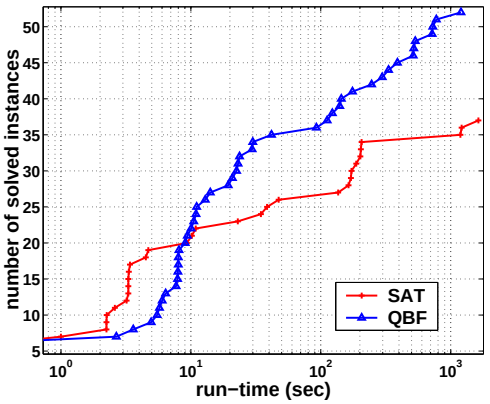


Fig. 15. Design debugging performance results

TABLE 5
QBF-based Design Debugging

Design Debugging Info				SAT			QBF		
Circuit Name	avg k	max k	# locations	# solved	time (sec)	mem (MB)	# solved	time (sec)	mem (MB)
AC97	32.3	60	2.0	4/6	702.9	272.7	6/6	111.4	30.1
Divider16	27.5	111	2.8	0/6	–	198.8	5/6	618.3	14.6
ERP	89.3	449	2.3	3/6	1001.4	350.6	5/6	517.5	64.7
ReacTimer	365.1	931	3.0	6/6	757.6	125.0	6/6	196.8	22.2
RSDecoder	2.5	8	3.5	6/6	5.1	27.0	6/6	5.0	2.1
SPI	21.5	56	2.0	6/6	145.4	56.3	6/6	13.7	3.2
Aqu	2.0	2	3.0	6/6	3.3	41.6	6/6	8.0	3.1
Fibonacci	3.0	4	2.0	6/6	0.2	1.1	6/6	0.1	0.1
AE18	156.1	504	5.0	0/6	–	580.1	6/6	464.7	68.6

improved by 39% on average. The QBF-based approach solves a total of 52 instances, while the SAT-based one solves 37. This amounts to a 41% increase in the number of solved instances with QBF.

Figure 15 plots the number of solved design debugging instances as a function of run-time for SAT-based and QBF-based formulations. Clearly, QBF has a run-time advantage over SAT. In fact, after less than ten seconds, the performance of the QBF solver remains invariably superior and SAT begins to plateau after 200 seconds because of excessive memory problems.

8.3 Sequential Test Generation

To generate sequential ATPG instances, three random stuck-at-faults are introduced in each circuit, for bounds of 10, 100 and 500 time-frames. The SAT and QBF formulations are given as described in Equations 16 and 17.

The results are shown in Table 6. `zChaff` is used to evaluate the SAT instances. For each approach, the columns # *solv.*, *time* and *mem* respectively show the number of solved problem instances out of three, the average run-time in seconds, and the average memory footprint of the files containing the problem instances in MBs. The QBF-based sequential ATPG approach solves 20 out of 27 instances, while the SAT approach solves 19. Furthermore, the QBF problem sizes are 84% smaller than their SAT counterparts. It should be noted that most of the solved instances returned `UNSAT`, which means that the introduced faults could not be detected using a test sequence within the given bounds. Finally, as shown, run-times are comparable between the two techniques.

TABLE 6
QBF-based Sequential ATPG

	SAT			QBF		
Circuit Name	# solv.	time (sec)	mem (MB)	# solv.	time (sec)	mem (MB)
AC97	2/3	673.3	739.3	3/3	67.3	133.2
Divider16	2/3	668.7	304.2	3/3	59.1	60.8
ERP	3/3	10.3	79.1	3/3	30.5	12.7
ReacTimer	3/3	2.3	10.2	3/3	22.8	1.5
RSDecoder	1/3	1369.1	613.1	1/3	1376.7	113.0
SPI	3/3	7.6	70.9	3/3	25.2	9.6
Aqu	1/3	1652.2	1 236.0	1/3	1716.0	205.9
Fibonacci	3/3	4.2	29.7	3/3	10.9	5.0
AE18	1/3	1420.8	148.0	0/3	–	18.8

9 Conclusion

This work presents a QBF-based ILA encoding and a robust hardware implementation for it to model the sequential behavior of a circuit. The encoding is parametrized using time-frame windowing, and the resulting family of logically equivalent encodings is shown to contain a non-trivial minimal-size member. A set of applications are encoded using the proposed formalism, namely BMC, design debugging and sequential test generation, to demonstrate its robustness and practicality. An extensive suite of experiments on publicly available industrial circuits confirms the expected memory gains and demonstrates the run-time competitiveness of the proposed techniques compared to state-of-the-art SAT-based approaches in all cases.

Admittedly, the theory and results of this paper emphasize the need for further research in QBF solvers and QBF-based CAD for VLSI solutions. Since the first complete QBF solver was presented decades after the first complete engine to solve SAT, research in this field remains at its infancy. This lures us into the fundamental research opportunities and multi-disciplinary contributions that still lie ahead.

References

[1] T. Sasao, *Logic Synthesis and Optimization*. Kluwer Academic Publisher, 1993.

[2] C. Sechen, *VLSI Placement and Global Routing using Simulated Annealing*. Kluwer Academic Publisher, 1988.

[3] R. Drechsler, *Formal Verification of Circuits*. Kluwer Academic Publisher, 2000.

[4] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[5] A. Sistla and E. Clarke, "The Complexity of Propositional Linear Temporal Logics," *Journal of the ACM*, vol. 32, no. 3, pp. 733–749, 1985.

[6] J. Jiang and R. Brayton, "Retiming and Resynthesis: A Complexity Perspective," *IEEE Trans. on CAD*, vol. 25, pp. 2674–2686, 2006.

[7] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen, "Scalable Sequential Equivalence Checking across Arbitrary Design Transformations," in *Int'l Conf. on Comp. Design*, 2006.

[8] A. Freitas, H. Neto, and A. Oliveira, "On the Complexity of Power Estimation Problems, " in *Int'l Workshop on Logic and Synthesis*, 2000, pp. 239–244.

[9] *International Technology Roadmap for Semiconductors*, 2007.

[10] M. Ganai and A. Gupta, *SAT-Based Scalable Formal Verification Solutions (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., 2007.

[11] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. The MIT Press, 2000.

[12] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation, " *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 667–691, 1986.

[13] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs, " in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 1579. Springer Verlag, 1999, pp. 193–207.

[14] J. Marques-Silva and K. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability, " *IEEE Trans. on Comp.*, vol. 48, no. 5, pp. 506–521, 1999.

[15] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver, " in *Design Automation Conf.*, 2001, pp. 530–535.

[16] N. Eén and N. Sörensson, "An Extensible SAT-Solver, " in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2003, pp. 502–518.

[17] A. Mishchenko, M. Case, R. Brayton, and S. Jang, "Scalable and Scalably-Verifiable Sequential Synthesis, " in *Int'l Workshop on Logic and Synthesis*, 2008.

[18] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, "Bounded Model Checking, " ser. Advances in Computers, no. 60, 2003.

[19] N. Amla, X. Du, A. Kuehlmann, R. Kurshan, and K. McMillan, "An Analysis of SAT-Based Model Checking Techniques in an Industrial Environment, " in *CHARME*, 2005, pp. 254–268.

[20] H. Konuk and T. Larrabee, "Explorations of Sequential ATPG using Boolean Satisfiability, " in *IEEE VLSI Test Symp.*, 1993, pp. 85–90.

[21] A. Smith, A. Veneris. M. Ali, and A. Viglas, "Fault Diagnosis and Logic Debugging using Boolean Satisfiability, " *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[22] F. Lu and S. Malik, "Conflict Driven Learning in a Quantified Boolean Satisfiability Solver, " in *Int'l Conf. on CAD*, 2002, pp. 442–449.

[23] A. Biere, "Resolve and Expand, " in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2004, pp. 238–246.

[24] M. Benedetti, "sKizzo: A Suite to Evaluate and Certify QBFs, " in *Int'l Conf. on Automated Deduction*, 2005, pp. 369–376.

[25] H. Samulowitz and F. Bacchus, "Using SAT in QBF, " in *Int'l Conf. on Principles and Practive of Constraint Programming*, 2005, pp. 578–592.

[26] N. Dershowitz, Z. Hanna, and J. Katz, "Bounded Model Checking with QBF, " in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2005, pp. 408–414.

[27] T. Jussila and A. Biere, "Compressing BMC Encodings with QBF, " in *Int'l Workshop on Bounded Model Checking*, 2006, pp. 1–14.

[28] H. Mangassarian, A. Veneris, S. Safarpour, M. Benedetti, and D. Smith, "A Performance-Driven QBF-Based Iterative Logic Array Representation with Applications to Verification, Debug and Test, " in *Int'l Conf. on CAD*, 2007, pp. 240–245.

[29] G. Tseitin, "On the Complexity of Derivations in the Propositional Calculus, " in *Studies in Constructive Mathematics and Mathematical Logic*, 1968, pp. 115–125.

[30] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving, " *Comm. of the ACM*, vol. 5, pp. 394–397, 1962.

[31] M. Sheeran, S. Singh, and G. Stålmarck, "Checking Safety Properties using Induction and a SAT-Solver, " in *Formal Methods in Computer-Aided Design*, 2000, pp. 127–144.

[32] M. Ali, S. Safarpour, A. Veneris, M. Abadir, and R. Drechsler,, "Post-Verification Debugging of Hierarchical Designs, " in *Int'l Conf. on CAD*, 2005, pp. 871–876.

[33] OpenCores.org, "http://www.opencores.org, " 2006.

[34] C. Sinz and E. Dieringer, "DPvis - A Tool to Visualize Structured SAT Instances, " in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2005.

Hratch Mangassarian received the B.E. degree in Computer and Communications Engineering with high distinction from the American University of Beirut, Lebanon, in 2005, and the M.A.Sc. degree in Electrical and Computer Engineering from the University of Toronto, ON, Canada, in 2008, where he is currently working toward the Ph.D. degree. His research is on formal verification and automated design debugging of digital designs, as well as QBF and its applications to CAD.

Andreas Veneris received the Diploma in Computer Engineering and Informatics from the University of Patras in 1991, the M.S. degree in Computer Science from the University of Southern California, Los Angeles in 1992 and the Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 1998. In 1998-99 he was a visiting faculty at the University of Illinois. Currently, he is an Associate Professor cross-appointed with the Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Toronto. His research interests include algorithms and CAD tools for debug, verification, synthesis and test of digital systems and circuits as well as combinatorics. He is co-recipient of a best paper award, co-author of two books and he holds multiple patents. He is a member of IEEE, ACM, AAAS, Technical Chamber of Greece, Professional Engineers of Ontario and The Planetary Society.

Marco Benedetti received his PhD in 2002 from DIS (Department of Informatics and Systems Science, Univ. La Sapienza, Rome, Italy) with a thesis on Propositional Satisfiability algorithms. He then moved to ITC-Irst (Institute for Scientific and Technological Research, Trento, Italy) to work on SAT-based techniques for Model Checking. At present, Dr. Benedetti is a Research Associate at LIFO (Laboratoire d'Informatique Fondamentale d'Orlans, France), and his research focuses on automated deduction for formal verification, planning, model checking, and related domains. He has developed several algorithms and techniques for evaluating Quantified Boolean Formulas (QBFs) and Quantified Constraint Satisfaction Problems (QCSPs), all of which are implemented in the publicly available, state-of-the-art solvers "sKizzo" and "QeCode". He has been recently applying such techniques to automated design debugging of digital designs.