A Performance-Driven QBF-Based Iterative Logic Array Representation with Applications to Verification, Debug and Test

Hratch Mangassarian[1] Andreas Veneris[1,2] Sean Safarpour[1] Marco Benedetti[3] Duncan Smith[1]

***Abstract*— Many CAD for VLSI techniques use time-frame expansion, also known as the Iterative Logic Array representation, to model the sequential behavior of a system. Replicating industrial-size designs for many time-frames may impose impractically excessive memory requirements. This work proposes a performance-driven, succinct and parametrizable Quantified Boolean Formula (QBF) satisfiability encoding and its hardware implementation for modeling sequential circuit behavior. This encoding is then applied to three notable CAD problems, namely Bounded Model Checking (BMC), sequential test generation and design debugging. Extensive experiments on industrial circuits confirm outstanding run-time and memory gains compared to state-of-the-art techniques, promoting the use of QBF in CAD for VLSI.**

I. Introduction

The tasks of verification, debugging and functional testing constitute major bottlenecks in the VLSI design cycle consuming up to 70% of the overall design effort [1]. When handling sequential circuits, some solutions explicitly unfold the circuit for a bounded number of time-frames specified by the problem. This is known as *time-frame expansion* or the *Iterative Logic Array (ILA)* representation of the circuit. Some popular applications using the ILA include sequential equivalence checking [2], Bounded Model Checking (BMC) [3], [4], trace compaction [5], sequential test generation (ATPG) [6] and design debugging [7], among others.

Despite the pre-eminence of the ILA, replicating the circuitry of a modern industrial-size design for a large number of cycles can lead to memory explosion issues. As such, more compact representations of sequential circuit behavior are required to ensure the scalability of verification and testing tools without sacrificing performance.

This work presents an original, performance-driven and parametrizable QBF-based ILA encoding supported by illustrative hardware constructions. This QBF formulation aims at not only reducing memory consumption, but also at achieving competitive run-times when compared to state-of-the-art Boolean satisfiability (SAT) solutions. Specifically, the major contributions of this paper are as follows:

- A novel QBF-based general-purpose ILA encoding using a single copy of the circuit, dramatically reducing memory requirements.
- A detailed hardware implementation for this encoding.
- An innovative and empirically vital extension of this encoding, titled *time-frame windowing*, generalizing the QBF-based ILA encoding, enabling further compression and boosting performance.
- Applications of this new framework to BMC, debugging and sequential ATPG, demonstrating its ease-of-use and practicality.

Alongside SAT-based initiatives aimed at reducing memory consumption when dealing with sequential behavior [8], recently the formalism of Quantified Boolean Formulas (QBF) was tested in BMC in an effort to bypass circuit replication using universal quantification [9], [10]. Experiments from [9], [10] suggest that memory compression comes at the cost of run-time performance. In contrast, an extensive suite of experiments on industrial circuits for the proposed formalism confirms the expected memory savings but also shows performance gains when compared to SAT. The memory footprint is reduced by an average of 89% and run-times are comparable and sometimes better by orders of magnitude. Due to the efficient memory management, the total number of solved instances is increased by 24%. Admittedly, the results encourage research in QBF-based encodings and solvers as platforms to efficiently tackle intractable CAD problems.

The paper is organized as follows. Section II contains preliminaries and Section III describes the general QBF-based ILA encoding. Sections IV, V and VI illustrate the QBF formulations for BMC, design debugging and sequential ATPG using the new ILA encoding. Section VII discusses the differences with previous work, Section VIII gives experimental results and Section IX concludes the paper.

[1]University of Toronto, ECE Department, Toronto, ON M5S 3G4 ({hratch, veneris, sean, duncan}@eecg.toronto.edu)

[2]University of Toronto, CS Department, Toronto, ON M5S 3G4

[3]LIFO, University of Orléans, BP 6759 – 45067 Orléans Cedex 2, France (Marco.Benedetti@univ-orleans.fr)

II. Preliminaries

A. SAT and QBF

Given a propositional logic formula Φ, SAT asks whether there exists a *satisfying assignment* to the variables $\mathcal{V}$ in Φ that evaluates it to 1 (Φ is SAT) or whether no such variable assignment exists (Φ is UNSAT). Formally, the SAT problem asks if $\exists \mathcal{V} \mid \Phi$. Traditionally, Φ is given in Conjunctive Normal Form (CNF) as a conjunction of *clauses* where each clause is a disjunction of literals. A literal is an instance of a variable or its negation. For example, the formula $\Phi = (a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (c)$ is SAT because $\{a = 1, b = 0, c = 1\}$ is a satisfying assignment. A logic circuit can be converted to CNF in linear time [11]. Today, industrial VLSI problems with millions of variables and clauses can be solved efficiently by modern SAT solvers [12], [13].

All variables in SAT are existentially quantified. QBF satisfiability is a generalization of SAT where variables can be universally and existentially quantified. A QBF formula in *prenex normal form* is written as:

$$Q_1\mathcal{V}_1, Q_2\mathcal{V}_2, \ldots, Q_r\mathcal{V}_r \mid \Phi$$

where

- the *prefix* $Q_1\mathcal{V}_1, Q_2\mathcal{V}_2, \ldots, Q_r\mathcal{V}_r$ consists of *quantifiers* $Q_i \in \{\forall, \exists\}$, such that $Q_i \neq Q_{i+1}$, and mutually disjoint variable sets $\mathcal{V}_i$ (also called *scopes*).
- the *matrix* Φ is a CNF formula on the variables in the prefix.

Q_r (Q_1) is called the innermost (outermost) quantifier. A 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 value assignment to each existential variable as a function of its dominating universal variables, by which every combination of assignments to the universal variables can be extended to satisfy the matrix, the QBF problem is said to be SAT, otherwise it is UNSAT. For example, $\exists a \forall b \exists c \mid (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (a)$ is SAT because when $a = 1$, for all values of b, there exists an assignment to c ($c = 1$ when $b = 0$ and $c = 0$ when $b = 1$) that satisfies the matrix. Today, state-of-the-art QBF solvers [14]–[17] use search [16], [17], resolution and expansion [14], skolemization [15], as well as BDD-based strategies to solve QBF instances that typically contain tens to hundreds of thousands of variables and clauses.

B. Notation

The following notation is used throughout the paper. x, y and s (x_i, y_i and s_i) are the Boolean vectors (bits) denoting the (i^{th} bit in the) primary inputs, primary outputs and state elements of a sequential circuit, respectively. Symbol b denotes the number of state elements (flip-flops). The behavior of a sequential circuit can be formally described by the predicate $T(s, s')$ expressing the transition relation of the system. It evaluates to 1 if and only if $s \rightsquigarrow s'$ is a valid state transition. A predicate $T(s, s', x, y)$ explicitly mentioning inputs and outputs can also describe the system behavior.

Building the ILA of a sequential circuit for a bound k consists of unfolding 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. This process shown in Fig. 1 is also referred to in the literature as *circuit unrolling* or *time-frame expansion*.

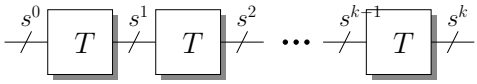


Fig. 1. Iterative Logic Array

Let s^{j-1} and s^j respectively denote the current-state and next-state variables of the j^{th} time-frame. Using this notation, the ILA shown in Fig. 1 can be formally encoded as:

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

Similarly, for each $z \in \{x, y\}$ and subsequently defined circuit variables, let z^j denote the corresponding variable in time-frame j, and $Z = \langle z^1, z^2, \ldots, z^k \rangle$ the sequence of all corresponding variables in the ILA.

III. ILA Encoding using QBF

An ILA representation is required in many CAD procedures for sequential circuits. Each of these problems contains a set of problem-specific constraints along with the common ILA component, which usually constitutes the bulk of the problem size.

What follows outlines a new succinct QBF encoding of an ILA. It replaces the replicated transition relation $\bigwedge_{i=1}^{k} T$ shown in Fig. 1 with a *single* copy of T using appropriate control logic. A novel hardware construction allows the QBF solver to operate on different time-frames using a single copy of T. As will be shown shortly, the described QBF formulation consists of three quantifier scopes ($\exists\forall\exists$) irrespective of k and $O(\lg k)$ universal variables. In subsection III-A, this scheme is generalized to include an arbitrary number of copies of T in the matrix in an effort to boost performance.

To generate the encoding, a set of universal *time-frame select* variables $t = \langle t_1, t_2, \ldots, t_{\lceil \lg k \rceil} \rangle$ are created to allow indexing the ILA time-frames. Fig. 2 shows the hardware construction consisting of two appropriately constrained multiplexers (MUXes), respectively connected to the current-state and next-state of the transition relation T. The time-frame select variables are the common select lines of the MUXes. For a given assignment of these variables corresponding to time-frame j, where $1 \leq j \leq k$, the two MUXes connect the current state s of T to state s^{j-1} of the ILA and the next state s' of T to state s^j. In effect, depending on the assignment given to vector t, the single copy of T in Fig. 2 "simulates" a different ILA time-frame.

The next step is to state Eq. 1 as a QBF instance using the described construction. In order to insure state contiguity, it is necessary that the set of ILA states $\{s^0, s^1, \ldots, s^k\}$ dominate the time-frame select variables. Informally, the QBF problem can be stated as follows:

> *Does there exist an assignment to states $s^0, s^1, \ldots, s^k$, such that for all time-frame select variables, T is satisfied?*

The binary description inherent to a QBF formulation forces the consideration of $2^{\lceil \lg k \rceil} = \Theta(k)$ time-frames. To simplify the notation and without loss of generality, the ceilings are dropped and it is assumed that k is a power of 2 in the remaining of the paper, unless noted otherwise. The left MUX in Fig. 2 is now formalized as follows:

$$\begin{aligned} &\mathrm{MUX}_{\lg k}(s, t, \langle s^0, s^1, \ldots, s^{k-1} \rangle) \quad \equiv \\ &\bigwedge_{j=0}^{k-1} [(\lambda(t) = j) \rightarrow (s = s^j)] \quad \equiv \quad (s = s^{\lambda(t)}) \end{aligned} \tag{2}$$

where $0 \leq \lambda(t) \leq k$ denotes the integer encoded as a binary number in the $\lg k$-bit time-select vector t (*e.g.*, $\lambda(\langle 0, 1, 0 \rangle) = 2$). Observe that the MUX given in Eq. 2 can be expressed using $2kb$ clauses of $\lg k + 2$ literals. The right MUX in Fig. 2 is similarly defined with a one-step shift. For a certain assignment to t, the two MUXes in Fig. 2 set the current-state s of T to $s^{\lambda(t)}$ and its next-state s' to $s^{\lambda(t)+1}$, thus making T simulate time-frame $\lambda(t) + 1$ in the ILA.

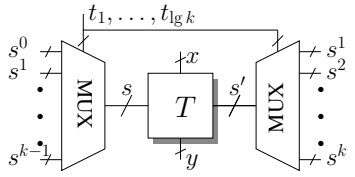


Fig. 2. Hardware construction for QBF bounded sequential model

The question above can now be formalized as:

$$\exists s^0, \ldots, s^k \; \forall t \; \exists s, s' \mid T(s, s') \wedge (s = s^{\lambda(t)}) \wedge (s' = s^{\lambda(t)+1}) \tag{3}$$

which can also be written explicitly mentioning primary inputs and outputs by using the predicate $T(s, s', x, y)$ instead of $T(s, s')$.

A. Time-Frame Windowing

Eq. 3 gives a QBF encoding for an ILA using a single copy of the transition relation, as shown in Fig. 2. The given formulation can be generalized to include an arbitrary number of copies of T, forming a fixed-length *window* of explicitly unfolded time-frames, as shown in Fig. 3. This parameterization enhances the ability of the QBF solver to make direct inferences spanning a number of contiguous time-frames. As will be shown shortly, time-frame windowing also provides a means to further minimize the encoding size. Fig. 3 shows the hardware construction using a window of size τ time-frames. Notice that setting $\tau = 1$ reduces Fig. 3 to Fig. 2 with $(s, s') = (s^{0,1}, s^{1,1})$. On the other hand, when $\tau = k$, windowing degenerates the proposed QBF encoding to the original ILA in Fig. 1.

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, $\lceil \frac{k}{\tau} \rceil$ 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. Note that the total number of considered time-frames now becomes $\tau 2^{\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil} = \Theta(k)$. To simplify the notation, the ceilings are dropped by assuming that $k \bmod \tau \equiv 0$ and that $\frac{k}{\tau}$ is a power of 2. The formal QBF encoding whose matrix is equivalent to the hardware construction shown in Fig. 3 is given as:

$$\begin{aligned} &\exists s^0, s^\tau, s^{2\tau}, \ldots, s^k \quad \forall t \quad \exists s^{0,\tau}, s^{1,\tau}, \ldots, s^{\tau,\tau} \mid \\ &\bigwedge_{j=1}^{\tau} T(s^{j-1,\tau}, s^{j,\tau}) \wedge (s^{0,\tau} = s^{\lambda(t)\cdot\tau}) \wedge (s^{\tau,\tau} = s^{(\lambda(t)+1)\cdot\tau}) \end{aligned} \tag{4}$$

which can also be written in terms of $T(s^{j-1,\tau}, s^{j,\tau}, x^{j,\tau}, y^{j,\tau})$ for each circuit in the window.

Increasing the size of the window does not necessarily increase the size of the matrix. In fact, each MUX in Fig. 3 can now be expressed using $\frac{2k}{\tau} \cdot b$ clauses with $\lg \frac{k}{\tau} + 2$ literals in each clause. This corresponds to a $\Theta(\frac{\tau}{1 - \lg \tau / \lg k})$ reduction in the number of literals from the CNF representation of each MUX of Fig. 2. Therefore, relaxing all assumptions, the total number of literals in the CNF matrix of Eq. 4, which is a reliable figure of merit for the matrix size, is given by:

$$\tau \cdot lit(T) + 4b \cdot 2^{\lceil \lg \lceil \frac{k}{\tau} \rceil \rceil} \cdot (\lg \lceil \frac{k}{\tau} \rceil + 2) \tag{5}$$

where $lit(T)$ denotes the number of literals in the CNF of T. The window size τ^* that minimizes Eq. 5 can be found numerically.

In the following three sections, the presented ILA framework is adapted to three notable sequential CAD problems, namely BMC, design debugging and sequential ATPG.

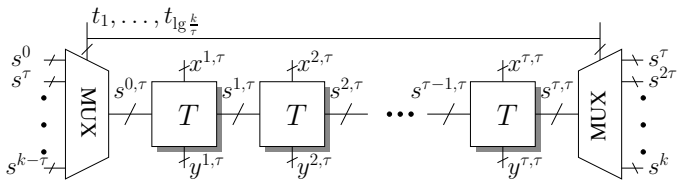


Fig. 3. Hardware construction using a time-frame window of size τ

IV. Bounded Model Checking

A. Background

Model Checking is an area of formal verification which is concerned with verifying (or falsifying) safety, liveness and other properties in a finite-state system. For checking safety properties, the idea behind BMC is to search within a bounded sequential limit for *counter-examples* violating a given property [3]. For instance, the following (safety) question:

> *Given a set of bad states $B(s)$, is a bad state reachable in exactly k time-frames starting from a valid initial state?*

can be formulated with the following SAT problem:

$$\exists s^0, s^1, \ldots, s^k \quad | \quad I(s^0) \wedge \bigwedge_{j=1}^{k} T(s^{j-1}, s^j) \wedge B(s^k) \tag{6}$$

where $I(s^0)$ denotes the predicate of valid initial states. To answer the question whether a bad state is reachable in *at most* k time-frames, it suffices to replace $B(s^k)$ in Eq. 6 by $\bigvee_{j=1}^{k} B(s^j)$.

Given Eq. 6, a SAT solver will either return a counter-example which consists of a sequence of states leading to a bad state (SAT), or it will prove that a bad state can not be reached in exactly k time-frames (UNSAT). In principle, BMC is complete for a large enough bound of k. However, as k increases, SAT-based BMC may require excessive memory due to the underlying ILA structure [3].

B. Proposed QBF-based Formulation

The BMC formulation given in Eq. 6 can be translated using the QBF-based ILA encoding of Eq. 3 as follows:

$$\exists s^0, s^1, \ldots, s^k \quad \forall t \quad \exists s, s' \quad | \\ I(s^0) \wedge T(s, s') \wedge (s = s^{\lambda(t)}) \wedge (s' = s^{\lambda(t)+1}) \wedge B(s^k) \tag{7}$$

The QBF-based BMC encoding for a window of size τ (using Eq. 4) is given as follows:

$$\exists s^0, s^\tau, s^{2\tau}, \ldots, s^k \; \forall t \; \exists s^{0,\tau}, \ldots, s^{\tau,\tau} \mid I(s^0) \wedge \bigwedge_{j=1}^{\tau} T(s^{j-1,\tau}, s^{j,\tau}) \\ \wedge (s^{0,\tau} = s^{\lambda(t)\cdot\tau}) \wedge (s^{\tau,\tau} = s^{(\lambda(t)+1)\cdot\tau}) \wedge B(s^k) \tag{8}$$

If the assumption that $k \bmod \tau \equiv 0$ is relaxed, s^k might not be available in the first scope of the prefix in Eq. 8. The reader can verify that state s^k in the ILA would then correspond to state $s^{(k-1) \bmod \tau + 1, \tau}$ in the $\lceil \frac{k}{\tau} \rceil^{th}$ window (*i.e.,* when $\lambda(t) = \lceil \frac{k}{\tau} \rceil - 1$). Therefore, the "bad state constraint" $B(s^k)$ in Eq. 8 would be replaced by $(\lambda(t) = \lceil \frac{k}{\tau} \rceil - 1) \rightarrow B(s^{(k-1) \bmod \tau + 1, \tau})$, which, for a given bad state in $B(s)$, can be expressed in b clauses of $\lg \lceil \frac{k}{\tau} \rceil + 1$ literals. For instance, given a single 2-bit bad state $\langle s_1, s_2 \rangle = \langle 0, 1 \rangle$, $k = 5$ and $\tau = 2$, the constraint becomes $(t_1 \vee \bar{t}_2 \vee t_3 \vee \bar{s}_1^{1,2}) \wedge (t_1 \vee \bar{t}_2 \vee t_3 \vee s_2^{1,2})$.

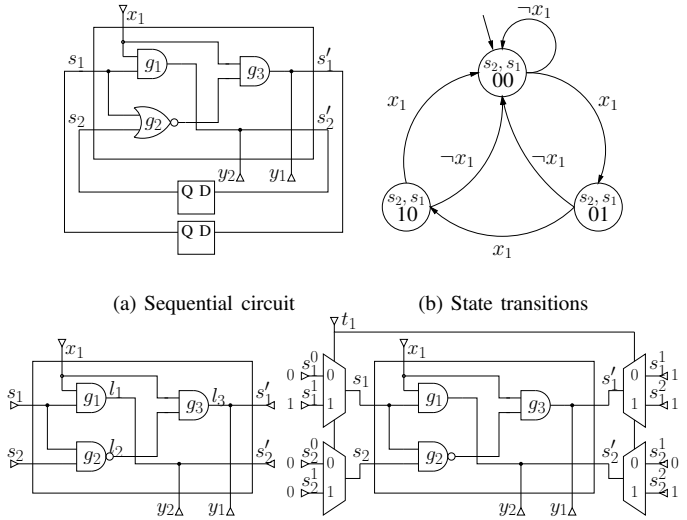
In the following example, the given QBF-based formulation is generated in bit-level detail for a specific BMC instance.

Example 1 *Fig. 4(a) depicts a modulo-3 incrementer where the output $y_2 y_1$ is a binary number incremented if and only if the input $x_1 = 1$. Fig. 4(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 Fig. 4(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 $\tau = 1$ and the hardware implementation are given by Eq. 9 and Fig. 4(d), respectively.*

$$\exists s_1^0, s_2^0, s_1^1, s_2^1, s_1^2, s_2^2 \; \forall t_1 \; \exists s_1, s_2, s_1', s_2' \mid \bar{s}_1^0 \wedge \bar{s}_2^0 \wedge T(\langle s_1, s_2 \rangle, \langle s_1', s_2' \rangle) \wedge \\ (\langle s_1, s_2 \rangle = \langle s_1^{\lambda(t_1)}, s_2^{\lambda(t_1)} \rangle) \wedge (\langle s_1', s_2' \rangle = \langle s_1^{\lambda(t_1)+1}, s_2^{\lambda(t_1)+1} \rangle) \wedge s_1^2 \wedge s_2^2 \tag{9}$$

When $t_1 = 0$, Fig. 4(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. As shown in Fig. 4(d), a counter-example reaching the bad state is the sequence of states $\langle s_1, s_2 \rangle$: $\langle 0, 0 \rangle \rightsquigarrow \langle 1, 0 \rangle \rightsquigarrow \langle 1, 1 \rangle$, corresponding to the input sequence $\langle x_1^1, x_1^2 \rangle = \langle \langle 1 \rangle, \langle 1 \rangle \rangle$.



(a) Sequential circuit (b) State transitions

(c) Erroneous circuit (d) BMC hardware construction

Fig. 4. BMC example

V. Design Debugging

A. Background

Given an erroneous design and a set of counter-examples, design debugging locates all possibly erroneous lines [7], [18]. SAT-based design debugging [7] encodes the problem as a SAT instance whose satisfying assignments correspond to the potential error locations in the circuit. For each counter-example, the encoding process consists of the following three steps:

i) The erroneous circuit is *enhanced* with extra hardware. A multiplexer with select line e_i is introduced at the output of every gate g_i, as done in Fig. 5(a) for the circuit in Fig. 4(c). An inactive multiplexer 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 primary input w_i. This can freely "fix" any potential error at the output of g_i. The enhanced transition relation is denoted as $T_{en}(s, s', \{x, w, e\}, y)$, where w and e respectively represent the unconstrained inputs and select lines of the multiplexers.

ii) The enhanced circuit is unfolded as an ILA. A set of constraints $\Phi_C(s^0, \langle x^1, \ldots, x^k \rangle, \langle y^1, \ldots, y^k \rangle)$ is added in order to ensure that the initial-state, primary inputs and primary outputs of the (enhanced) ILA match the correct behavior of the circuit given that counter-example. Added multiplexers on the same original gate share a common select line throughout the ILA.

iii) A final constraint $\Phi_N(e)$ setting the error cardinality (that is, the number of activated select lines) to N is added.

In this work, we give the formal encoding of the design debugging problem for one counter-example of k time-frames. For a discussion on collapsing multiple counter-examples using QBF, refer to [18]. For brevity, for each $z \in \{x, y, w\}$, we let $Z^\tau = \langle z^{1,\tau}, z^{2,\tau}, \ldots, z^{\tau,\tau} \rangle$.

SAT-based design debugging finds N erroneous gates by activating the corresponding bits in e to match the expected behavior of the circuit. N is initialized to 1 and increased until Eq. 10 becomes SAT:

$$\exists e, s^0, s^1, \ldots, s^k, X, W, Y \mid \bigwedge_{j=1}^{k} T_{en}(s^{j-1}, s^j, \{x^j, w^j, e\}, y^j) \\ \wedge \Phi_C(s^0, X, Y) \wedge \Phi_N(e) \tag{10}$$

B. Proposed Debugging Formulation

The design debugging problem given in Eq. 10 can be encoded using the QBF-based ILA encoding of Eq. 3:

$$\exists e, s^0, s^1, \ldots, s^k, X, Y \;\; \forall t \;\; \exists s, s', x, w, y \mid T(s, s', \{x, w, e\}, y) \\ \wedge (s = s^{\lambda(t)}) \wedge (s' = s^{\lambda(t)+1}) \wedge (x = x^{\lambda(t)+1}) \wedge (y = y^{\lambda(t)+1}) \\ \wedge \Phi_C(s^0, X, Y) \wedge \Phi_N(e) \tag{11}$$

For a window of size τ, using Eq. 4, the formulation becomes:

$$\exists e, s^0, s^\tau, s^{2\tau}, \ldots, s^k, X, Y \;\; \forall t \;\; \exists s^{0,\tau}, \ldots, s^{\tau,\tau}, X^\tau, W^\tau, Y^\tau \;\; |$$
$$\bigwedge_{j=1}^{\tau} T(s^{j-1,\tau}, s^{j,\tau}, \{x^{j,\tau}, w^{j,\tau}, e\}, y^{j,\tau}) \wedge (s^{0,\tau} = s^{\lambda(t)})$$
$$\wedge\, (s^{\tau,\tau} = s^{\lambda(t)+1}) \wedge \bigwedge_{j=1}^{\tau} [(x^{j,\tau} = x^{\lambda(t)\cdot\tau+j}) \wedge (y^{j,\tau} = y^{\lambda(t)\cdot\tau+j})]$$
$$\wedge\, \Phi_C(s^0, X, Y) \wedge \Phi_N(e) \qquad (12)$$

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

Example 2 *Consider the correct and erroneous designs in Fig. 4(a) and Fig. 4(c). In Example 1, BMC yielded a* 2*-time-frame counter-example with initial state* $\langle s_1^0, s_2^0\rangle = \langle \mathtt{0}, \mathtt{0}\rangle$ *and the sequence of inputs* $\langle x_1^1, x_1^2\rangle = \langle\langle \mathtt{1}\rangle, \langle \mathtt{1}\rangle\rangle$*. According to the correct state transition diagram shown in Fig. 4(b) (the states and outputs have the same values), the* expected *output sequence corresponding to this counter-example is* $\langle\langle y_1^1, y_1^2\rangle, \langle y_2^1, y_2^2\rangle\rangle = \langle\langle \mathtt{1}, \mathtt{0}\rangle, \langle \mathtt{0}, \mathtt{1}\rangle\rangle$*. 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* $\tau = 1$ *and* $N = 1$ *and its corresponding hardware construction are shown respectively in Eq. 13 and Fig. 5(b).*

$$\exists e_1, e_2, e_3, s_1^0, s_2^0, s_1^1, s_2^1, s_1^2, s_2^2, x_1^1, x_1^2, y_1^1, y_2^1, y_1^2, y_2^2 \;\; \forall t_1$$
$$\exists s_1, s_2, s_1', s_2', x_1, w_1, w_2, w_3, y_1, y_2 \;\; |$$
$$T(\langle s_1, s_2\rangle, \langle s_1', s_2'\rangle, \{x_1, \langle w_1, w_2, w_3\rangle, \langle e_1, e_2, e_3\rangle\}, \langle y_1, y_2\rangle)$$
$$\wedge\, (\langle s_1, s_2\rangle = \langle s_1^{\lambda(t_1)}, s_2^{\lambda(t_1)}\rangle) \wedge (\langle s_1', s_2'\rangle = \langle s_1^{\lambda(t_1)+1}, s_2^{\lambda(t_1)+1}\rangle)$$
$$\wedge\, (x_1 \leftrightarrow x_1^{\lambda(t_1)+1}) \wedge (\langle y_1, y_2\rangle = \langle y_1^{\lambda(t_1)+1}, y_2^{\lambda(t_1)+1}\rangle)$$
$$\wedge\, \bar{s}_1^0 \wedge \bar{s}_2^0 \wedge x_1^1 \wedge x_1^2 \wedge y_1^1 \wedge \bar{y}_2^1 \wedge \bar{y}_1^2 \wedge y_2^2 \wedge (e_1 + e_2 + e_3 = 1) \qquad (13)$$

Fig. 5(b) also shows the $\Phi_C(s^0, X, Y)$ *constraints applied at the initial-state/inputs/outputs of the circuit. The variables of interest for the design debugging problem are the select lines* $e = \langle e_1, e_2, e_3\rangle$*. Eq. 13 is* SAT *and a QBF solver will return the assignment* $\langle e_1, e_2, e_3\rangle = \langle \mathtt{0}, \mathtt{1}, \mathtt{0}\rangle$*, which means that gate* g_2 *is erroneous.*

VI. Sequential Test Generation

A. Background

A circuit line is stuck-at-0 (stuck-at-1) if it always assumes a constant value of 0 (1). Sequential ATPG for stuck-at faults finds an input test sequence which deviates the primary output(s) of the sequential circuit containing the fault from its correct behavior. It is a mature, yet intractable problem that has been tackled using several approaches including SAT-based ones that use an ILA [6], [19].

Let $T_a(s_a, s_a', x_a, y_a)$ ($T_b(s_b, s_b', x_b, y_b)$) denote the transition relation of the fault-free (faulty) circuit. The sequential ATPG problem can be formulated as follows:

$$\exists s_a^0, s_a^1, \ldots, s_a^k, s_b^0, s_b^1, \ldots, s_b^k, X, Y_a, Y_b \mid \bigwedge_{j=1}^{k} T_a(s_a^{j-1}, s_a^j, x^j, y_a^j)$$
$$\wedge\, (s_a^0 = s_b^0) \wedge \bigwedge_{j=1}^{k} T_b(s_b^{j-1}, s_b^j, x^j, y_b^j) \wedge \bigvee_{j=1}^{k} (y_a^j \neq y_b^j) \qquad (14)$$

where $Y_a = \langle y_a^1, \ldots, y_a^k\rangle$ ($Y_b = \langle y_b^1, \ldots, y_b^k\rangle$) denotes the output sequence of the fault-free (faulty) circuit.

This SAT-based sequential ATPG formulation essentially searches for the common sequence of inputs X feeding to both T_a and T_b, which causes at least one primary output in Y_a to be different from Y_b.

B. Proposed Sequential ATPG formulation

Using the QBF-based ILA formulation of Eq. 3, Eq. 14 can be encoded in QBF using two copies of the transition relation, T_a and T_b. However, it is possible to give the QBF-based sequential ATPG formulation using a single transition relation as follows. The circuit is enhanced by introducing a multiplexer at the stuck-at-fault location, which chooses between the correct and faulty line using a select line l. For the circuit in Fig. 4(a) and a stuck-at-1 at the output of gate g_2, the enhanced circuit T_{en} is given in Fig. 6(a). Now, the sequential ATPG problem corresponds to the following question:

> *Do there exist an initial state* s^0*, a common sequence of inputs* X *and different outputs* Y_a *and* Y_b*, 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_a^j *if* $l = \mathtt{0}$ *and* y_b^j *if* $l = \mathtt{1}$*?*

This question can be formalized as follows:

$$\exists s^0, X, Y_a, Y_b \;\; \forall l \;\; \exists s^1, \ldots, s^k, Y \;\; \forall t \;\; \exists s, s', x, y \mid T_{en}(s, s', \{x, l\}, y)$$
$$\wedge\, (s = s^{\lambda(t)}) \wedge (s' = s^{\lambda(t)+1}) \wedge (x = x^{\lambda(t)+1}) \wedge (y = y^{\lambda(t)+1})$$
$$\wedge \bigwedge_{j=1}^{k} [(l \rightarrow y^j = y_a^j) \wedge (\bar{l} \rightarrow y^j = y_b^j)] \wedge \bigvee_{j=1}^{k} (y_a^j \neq y_b^j) \qquad (15)$$

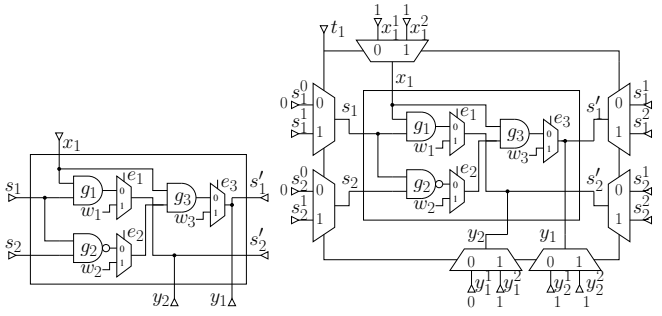
Figure 6(b) shows the hardware construction corresponding to the matrix of Eq. 15 for the circuit in Fig. 4(a). We do not show the formulation using time-frame windowing due to lack of space.

VII. Comparison to Previous Work

In [9], a QBF-based BMC encoding is given, which introduces two universal state variables, and hence $\Theta(b)$ universal bits, to traverse the ILA states. A related BMC formulation [10] introduces k universal bits to traverse all k ILA time-frames using a forced one-hot encoding. Additionally, the non-copying iterative squaring encoding in [10] recursively defines the reachability $T^{2k}(s, s')$ of state s' starting from state s in $2k$ time-frames 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 [10] conclude that QBF solvers are not taking advantage of those compact encodings to improve performance.

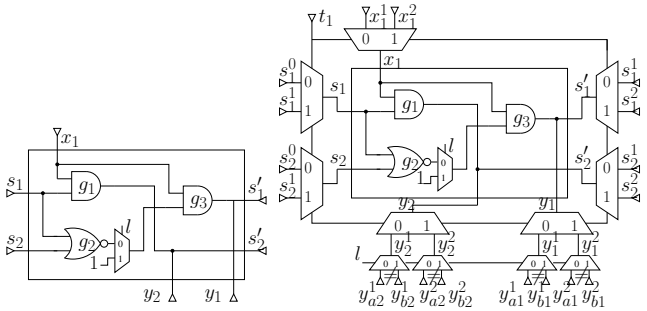
One of the subtle differences with the work here is the presentation of a general-purpose ILA encoding in Section III as opposed to the confinement to a particular application (BMC). The resulting BMC formulations are also different due to the underlying ILA encoding and our novel hardware multiplexer-based implementation. In fact, the proposed BMC formulation introduces at most $O(\lg k)$ universal variables, thus preserving the advantage of the non-copying iterative squaring method in the worst case, while using a constant number of quantifier scopes ($\exists\forall\exists$) and a linear representation of time. All these unique characteristics seem to have a significant impact on performance, as shown in Section VIII.

Another major novelty of the proposed formulation given in Eq. 8, is the time-frame windowing technique, which will be shown to have a positive impact on the results. This explicit unrolling of the transition



(a) Enhanced circuit (b) Debugging hardware construction

Fig. 5. Design debugging example



(a) Enhanced circuit (b) ATPG hardware construction

Fig. 6. Sequential ATPG example

TABLE I
QBF-BASED VERSUS SAT-BASED BMC

Circuit Info			SAT			(QBF,$\tau = 1$)			(QBF,$\tau = 16$)			(QBF,τ^*)			
circuit name	# gates	# DFFs	# solved	time (sec)	mem (MB)	# solved	time (sec)	mem (MB)	# solved	time (sec)	mem (MB)	avg τ^*	# solved	time (sec)	mem (MB)
AC97	15,601	1,452	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	5,248	388	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	2,449	347	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	265	22	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	12,041	521	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	2,012	90	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	22,319	1,504	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	652	36	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	2,520	116	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

relation not only allows further size compression (Eq. 5), but also boosts the inference power of the QBF solver over several time-frames.

In [18], a QBF-based design debugging formulation is given to handle multiple counter-examples. In other words, [18] does not propose a QBF-based ILA encoding but describes a technique for collapsing multiple ILAs for different test vectors into a single ILA. In this sense, the work here and that of [18] are complementary and non-overlapping. For the debugging experiments, following next, we integrate both those orthogonal formulations into the final approach.

VIII. EXPERIMENTS

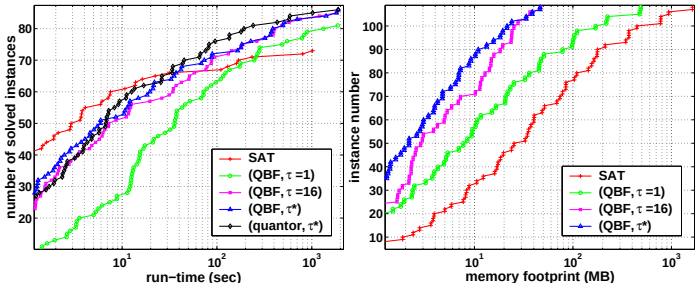
For each of BMC, design debugging and sequential ATPG, a C++ module is implemented which produces the respective QBF formulation from Sections IV, V and VI. The generated QBF instances are solved using `sKizzo` [15], a state-of-the-art QBF solver based on symbolic skolemization. Nine industrial circuits from OpenCores.org [20] are used to construct the problem instances. All experiments are conducted on a Pentium IV 2.8 GHz Linux platform with 2 GB of memory and a timeout of 2000 seconds.

A. Bounded Model Checking

Safety BMC problems of the form of Eq. 6 are considered. For each circuit, six exponentially increasing bounds of size 32, 64, 128, 256, 512 and 1024 are examined and two "bad" states are checked for each bound: One that is reachable (`SAT`) and one that is not (`UNSAT`). The proposed QBF-based formulations are evaluated against traditional SAT-based encodings solved by MINISAT V1.14 [13], a state-of-the-art SAT solver.

As shown in Table I, results for three QBF-based BMC encodings with different time-frame windowing schemes are compared to the SAT results. (QBF, $\tau = 1$) does not use time-frame windowing, (QBF, $\tau = 16$) uses a fixed window of size $\tau = 16$ and (QBF, τ^*) uses the window size τ^* which minimizes the number of literals in the formulation according to Eq. 5. The first three columns of Table I respectively show the circuit name, its number of gates and its number of state elements (`DFFs`). Next, 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-out. 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 (QBF, $\tau = 1$), (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, (QBF, $\tau = 16$) solves 5 more instances than (QBF, $\tau = 1$), and (QBF, τ^*) uses 85% less memory compared to (QBF, $\tau = 1$).



(a) Run-time performance (b) Memory footprint

Fig. 7. BMC Comparison Results

TABLE II
BMC USING `quantor`

	(`quantor`,τ^*)					
circuit name	# solved		largest k		time (sec)	mem MB
	`SAT`	`UNSAT`	`SAT`	`UNSAT`		
AC97	6/6	6/6	1024	1024	86.8	17.2
Divider16	3/6	6/6	128	1024	536.9	4.9
ERP	4/6	6/6	256	1024	347.9	3.0
ReacTimer	6/6	6/6	1024	1024	1.1	0.2
RSDecoder	1/6	1/6	32	32	1667.4	9.0
SPI	6/6	6/6	1024	1024	25.8	1.4
Aqu	4/6	6/6	256	1024	392.2	24.0
Fibonacci	6/6	6/6	1024	1024	1.9	0.5
AE18	3/6	3/6	128	128	1143.7	1.9

In order to demonstrate the robustness of the proposed BMC encodings, the (QBF, τ^*) experiments are also run using `quantor` [14], which is another state-of-the-art QBF solver based on resolution and expansion. These results are shown in Table II. For each circuit, the second and third columns respectively show the numbers of solved `SAT` and `UNSAT` instances out of six. The fourth and fifth columns respectively show the largest solved `SAT` and `UNSAT` bounds. The sixth column gives the average run-time and the last column gives the average memory footprint of the QBF instance file, which is the same as before. `quantor` solves 85 instances out of 108, which is one more than `sKizzo` using a window of size τ^*. Also, `UNSAT` instances are solved more easily than `SAT` ones, a fact that could be attributed to the resolution-based strategy used by `quantor`.

Fig. 7(a) plots the number of solved BMC instances 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. It can be seen that SAT has an initial advantage on smaller instances taking less than 30 seconds to solve. Both (QBF, $\tau = 16$) and (QBF, τ^*) outperform SAT given 80 seconds, while (QBF, $\tau = 1$) outperforms SAT given 300 seconds. All three QBF formulations take advantage of the declining slope of the SAT curve in Fig. 7(a) as soon as the problem instances grow in complexity. Fig. 7(b) compares the memory footprints of the different methods. As expected, all three QBF encodings require considerably less memory than SAT encodings, while (QBF, τ^*) achieves maximum compression.

It should be noted that search-based QBF solvers such as [16], [17] performed poorly compared to `sKizzo` and `quantor`. The outstanding performance of `sKizzo` and `quantor` can be contributed to the linear and highly non-random structure of the interaction graphs of the problem matrices, which enables skolemization and resolution to proceed without immediate memory explosions. On the other hand, search-based QBF solvers seem to take a lot of time to "guess" correct state transitions in the ILA.

TABLE III
QBF-BASED VERSUS SAT-BASED DESIGN DEBUGGING

Design Debugging Info				SAT			QBF		
circuit name	avg k	max k	# loc.	# solv.	time (sec)	mem (MB)	# solv.	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

TABLE IV
QBF-BASED VERSUS SAT-BASED SEQUENTIAL ATPG

	SAT			QBF		
circuit name	# solved	time (sec)	mem (MB)	# solved	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

B. Design Debugging

For the debugging problems, `sKizzo` has been purposely modified to return not just one but all the valid assignments to e that identify possible error sites when the instance is `SAT`. The erroneous circuits are created by manually changing the functionality of certain modules to introduce the 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. For each circuit, the results are averaged out over the number of instances. In order to deal with multiple counter-examples, the ideas in [18] 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 [7] that uses circuit replication with `zChaff` [12] being the underlying SAT solver.

Table III 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 *# solv.*, *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 III, 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 improved by 39% on average. The QBF-based approach solves a total of 52 instances, while the SAT-based one solves 37, which amounts to a 41% increase in the number of solved instances using QBF.

Fig. 8 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.

C. Sequential Test Generation

Six circuits are used for the sequential test generation experiments. For each circuit, three random stuck-at-faults are introduced for three bounds of 10, 100 and 500 time-frames. The results for the SAT and QBF-based formulations described in Section VI are shown in Table IV. `zChaff` is used to evaluate the SAT instances. For each approach, the columns *# solved*, *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 formulation consumes 83% less memory than its SAT-based counterpart. 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.

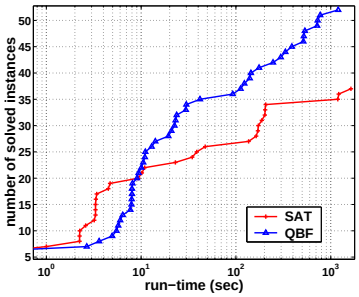


Fig. 8. Design debugging performance results

IX. CONCLUSION

This work presents a novel, performance-driven and parametrizable QBF-based ILA encoding and its implementation for modeling the sequential behavior of a circuit. Moreover, applications of this ILA encoding are presented for three problems, namely BMC, design debugging and sequential test generation. Extensive experiments confirm the major memory gains of the proposed approach and demonstrate its run-time competitiveness over state-of-the-art SAT-based approaches. The theory and results of this paper emphasize the need for further research in QBF solvers and QBF-based CAD solutions as complementary processes to current procedures.

REFERENCES

[1] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-chip Verification: Methodology and Techniques*. Kluwer Academic Publisher, 2000.
[2] D. Stoffel, M. Wedler, P. Warkentin, and W. Kunz, "Structural FSM traversal," *IEEE Trans. on CAD*, vol. 23, no. 5, pp. 598–619, 2004.
[3] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," ser. Advances In Computers, no. 60, 2003.
[4] 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.
[5] K. Chang, V. Bertacco, and I. Markov, "Simulation-based bug trace minimization with BMC-based refinement," in *Int'l Conf. on CAD*, 2005, pp. 1045–1051.
[6] M. R. Prasad, M. S. Hsiao, and J. Jain, "Can SAT be used to improve sequential ATPG methods?" in *VLSI Design*, 2004, pp. 585–591.
[7] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.
[8] L. Zhang, M. Iyer, G. Parthasarathy, and C. K. Cheng, "An efficient sequential SAT solver with improved search strategies," in *Design, Automation and Test in Europe*, 2005, pp. 1102–1107.
[9] 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.
[10] T. Jussila and A. Biere, "Compressing BMC encodings with QBF," in *Intl. Workshop on Bounded Model Checking*, 2006, pp. 1–14.
[11] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. on CAD*, vol. 11, pp. 4–15, 1992.
[12] 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.
[13] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2003, pp. 502–518.
[14] A. Biere, "Resolve and expand," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2004, pp. 238–246.
[15] M. Benedetti, "sKizzo: a suite to evaluate and certify QBFs," in *Int'l Conf. on Automated Deduction*, 2005, pp. 369–376.
[16] H. Samulowitz and F. Bacchus, "Binary clause reasoning," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2006, pp. 353–367.
[17] F. Lu and S. Malik, "Conflict driven learning in a quantified Boolean satisfiability solver," in *Int'l Conf. on CAD*, 2002, pp. 442–449.
[18] M. F. 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.
[19] H. Konuk and T. Larrabee, "Explorations of sequential ATPG using Boolean satisfiability," in *IEEE VLSI Test Symp.*, 1993, pp. 85–90.
[20] OpenCores.org, "http://www.opencores.org," 2006.