Encoding Techniques, Craig Interpolants and Bounded Model Checking for Incomplete Designs

Christian Miller, Stefan Kupferschmid, Matthew Lewis, and Bernd Becker

Albert-Ludwigs-Universität, Freiburg, Germany
{millerc,skupfers,lewis,becker}@informatik.uni-freiburg.de

Abstract. This paper focuses on bounded invariant checking for partially specified circuits – designs containing so-called blackboxes – using the well known 01X- and QBF-encoding techniques. For detecting counterexamples, modeling the behavior of a blackbox using 01X-encoding is fast, but rather coarse as it limits what problems can be verified. We introduce the idea of 01X-hardness, mainly the classification of problems for which this encoding technique does not provide any useful information about the existence of a counterexample. Furthermore, we provide a proof for 01X-hardness based on Craig interpolation, and show how the information contained within the Craig interpolant or unsat-core can be used to determine heuristically which blackbox outputs to model in a more precise way. We then compare 01X, QBF and multiple hybrid modeling methods. Finally, our total workflow along with multiple state-of-the-art QBF-solvers are shown to perform well on a range of industrial blackbox circuit problems.

Keywords: BMC, blackbox, SAT, QBF, Craig interpolation, unsat-core.

1 Introduction

Recently, Bounded Model Checking (BMC) has become an important method for finding errors in sequential circuits [1,2]. BMC accomplishes this by iteratively unfolding a circuit k times for $k = 0, 1, \ldots$, adding the negated property, and then finally converting the BMC instance into a SAT formula so that a SAT-solver can be used. If the SAT-solver finds the k-th problem instance satisfiable, a path of length k violating the property has been found. In this paper we focus on BMC for *incomplete* designs, meaning that certain parts of the circuit (combined into a so-called blackbox) are not specified. The interest on verifying incomplete designs is becoming popular as larger system-on-chip (SoC) designs, that contain multiple blackbox IP cores, become more prevalent. Blackboxes can also add a layer of abstraction if a design is too large to verify in its entirety. Additionally, blackboxes allow us to start the verification process earlier in the design stages of a chip when certain components are only partially completed.

In these cases we want to answer the question of *unrealizability*, that is, is there a path of length k violating the property regardless of the implementation of the blackbox. If so, the property is unrealizable. For example, a processor with a blackbox covering the ALU is shown in Figure 1. Since the behavior of blackbox outputs is unknown we need to model them in an adequate way. One option is to use 01X-logic. This approach applies the value X to all blackbox outputs, and then encodes the circuit as done e.g.

O. Strichman and S. Szeider (Eds.): SAT 2010, LNCS 6175, pp. 194–208, 2010.

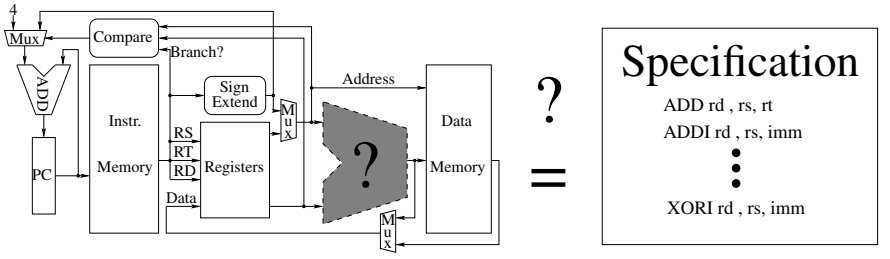

Fig. 1. Example MIPS type processor with blackboxes

in [8] by Jain et al. This again yields a propositional SAT formula, and a modern SAT-solver can be used. Counterexamples found by this approach are independent of the blackbox's behavior. However, using 01X-logic may be to coarse to prove the unrealizability of the property when the counterexamples depend on the blackbox's behavior. Therefore, we must sometimes model the blackbox outputs in a more precise way by universally quantifying them. This results in a quantified boolean formula (QBF).

QBF formulas are hard (PSPACE-Complete), and in this work we introduce improvements with respect to encoding a blackbox BMC problem using a combination of 01X-logic and QBF. Further, we present a method based on Craig interpolation to prove that using 01X-logic is too coarse to provide any information about the existence of a counterexample (problems of this type will be called *01X-hard*). For such problems, we show that it is usually not the case that *all* blackbox outputs of the incomplete design have to be encoded using QBF to obtain a counterexample. We then introduce two techniques for obtaining which blackboxes need to be modeled using QBF. The first is based on exploiting the computed Craig interpolants, and the second method uses the clauses from the unsatisfiable core to illuminate the problematic blackboxes. Our work, which incorporates all this into one tool, allows us to automatically combine the advantages of both 01X- and QBF-encodings so that we can verify more problems.

The paper is structured as follows. In Section 2 we introduce the concepts and related work for BMC of incomplete designs, Z-modeling using 01X-logic, Z_i-modeling using QBF as well as the combination of these modeling techniques. Section 3 then introduces our tool, optimizations, definitions for 01X-hard and 01X-easy, as well as new ideas for heuristically combining Z- and Z_i-modeling. Results and analysis of multiple industrial circuits are given in Section 4, and Section 5 concludes the paper.

2 Bounded Model Checking for Incomplete Designs

Standard BMC has been shown to be able to refute invariants on industrial sequential circuits [1,2]. Starting with the initial state of the circuit, BMC iteratively unfolds the system k times with $k = 0, 1, \ldots$ and checks in every iteration whether a counterexample for the given invariant exists or not. The algorithm stops, if a counterexample is found or a predefined unfolding limit is reached. Let I_0 characterize the initial state, $T_{i,i+1}$ the transition relation, which is *true*, if there is a transition from a state at time

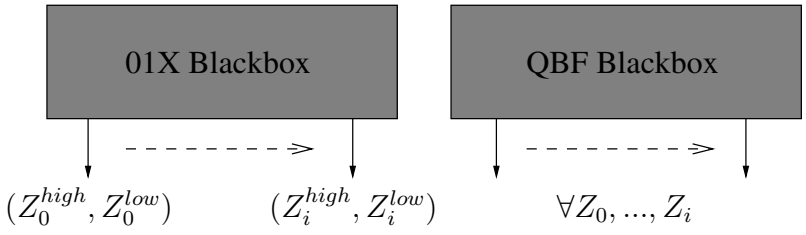

Fig. 2. Example modeling of blackbox outputs (01X vs. Z_i)

step i to a state at time step $i + 1$. Let P_k represent the invariant (property) at depth k. Then, the BMC formula (1) is satisfied iff there exists a counterexample of length k that violates the property.

$$I_0 \wedge T_{0,1} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k \tag{1}$$

In this paper we focus on bounded invariant checking of *incomplete* designs. An example of this is shown in Figure 1. In this pipelined processor, the ALU is not yet completed. To deal with this, we replace the ALU with a *blackbox*. The simplest way to model a blackbox is to assign *one* extra value denoting the unknown behavior to each blackbox output. With accordance to [18] we call this Z-modeling. For encoding the BMC problem this way we can make use of 01X-logic, an extension to propositional logic by a third logical value X, denoting an unknown state. This means that all signal lines in a circuit are now encoded using two bits instead of one. This allows each bus in the circuit to take on one of the three possible logic values (0,1, or X). For the blackbox outputs this is shown in the left part of Figure 2.

Encoding all blackbox outputs using 01X-logic is only sufficient for finding counterexamples that are independent of the blackbox's behavior. However, these problem can be encoded as a propositional SAT formula at the expense of additional variables and clauses. To encode all gates and buses using 01X-logic and Jain encoding [8], we use the following transformation functions:

$$\begin{aligned} AND_{01X}((g^{high}, g^{low}), (f^{high}, f^{low})) &:= (g^{high} \cdot f^{high}, g^{low} + f^{low}) \\ OR_{01X}((g^{high}, g^{low}), (f^{high}, f^{low})) &:= (g^{high} + f^{high}, g^{low} \cdot f^{low}) \\ NOT_{01X}((g^{high}, g^{low})) &:= (g^{low}, g^{high}) \end{aligned}$$

These transformation functions define the functional relationship of each gate using 01X-logic and our two bit encoding. This allows us to define the functionality of the AND_{01X} gate when one or both of its inputs are in the X (the unknown) state. For example, "0 AND_{01X} X" is always 0. However, "1 AND_{01X} X" results in X because the output of the AND_{01X} gate depends on the X input. Using this encoding, we can now convert the circuit and property into CNF form so that a high performance SAT-solver can be used.

Once a problem is encoded, the SAT-solver tries to find a counterexample that violates the property P_k, or prove that no such counterexample exists. Some counterexamples for an invariant can depend on the behavior of the blackbox. Unfortunately, these types of counterexample cannot be found using Z-modeling. The reason for this is that the X values assigned to the blackbox outputs propagate to the relevant signals checked by the invariant. Finding out if blackbox outputs cause this issue motivates our definition of 01X-hardness in Section 3.2.

To compute counterexamples of incomplete designs which depend on the blackbox's behavior, a more precise modeling method is needed. Similar to [18] we call this technique Z_i-modeling, and it is shown on the right side of Figure 2. This technique introduces one universally quantified variable for each blackbox output resulting in a QBF formula. QBF extends propositional formulas by allowing variables to be either universally ($\forall$) or existentially ($\exists$) quantified. Using the QBF formulation, there is no longer a need for three valued logic and the X state, as we can check all possible logic values of the blackbox outputs (0 and 1) using universal quantifiers.

Since we are interested in one input sequence, such that for every blackbox behavior the invariant is violated, we build the quantifier prefix stepwise as follows: let $x_0, \ldots, x_n$ be the primary inputs and $Z_0, \ldots, Z_m$ the blackbox outputs of the design. A second index denotes the unfolding depth of the variable (e.g. $Z_{3,2}$ is blackbox output Z_3 at unfolding depth 2). Further let H_i be the set of additional Tseitin-variables needed to encode the circuit at unfolding depth i. We end up in the following quantifier prefix (referred to as $pref_1$ and presented in [6]):

$$\underbrace{\exists x_{0,0}, \ldots, x_{n,0} \forall Z_{0,0}, \ldots, Z_{m,0} \exists H_0}_{\text{depth 0}} \ldots \underbrace{\exists x_{0,k}, \ldots, x_{n,k} \forall Z_{0,k}, \ldots, Z_{m,k} \exists H_k}_{\text{depth k}} \text{ CNF} \qquad (pref_1)$$

By combining both these methods we now can encode a combination of Z- and Z_i-modeled blackbox outputs. This allows use to reduce the number of universal variables in the problem with the aim of making the resulting QBF problems easier to solve. Concerning the combined Z/Z_i-modeling in [6], and unlike Figure 2, they used two variables for each Z_i-modeled blackbox output, one universally and one existentially quantified. This resulted in two quantifier alternations per Z_i-modeled blackbox output. This allowed them to keep transformation functions constant for both Z- and Z_i-modeled blackboxes. However, the resulting QBF formula was very complex, and could contain thousands of quantifier alternations. In Section 3 we introduce new transformation functions that remove these unneeded quantifier alternation and variables.

Finally, other work similar to [6] examined the BMC problem of incomplete designs in the context of BDD based model checking [12,13]. However, most previous work used randomly placed blackboxes, and random selection of which type of model (Z or Z_i) each blackbox output should use. In this work, we place blackboxes for specific circuit components (e.g. adders, multiplier units, control units, ...), and not just random gates. Moreover, we introduce heuristics to automatically find which blackbox outputs must be modelled using Z_i and QBF, and which can remain using 01X-logic.

3 Workflow

Our blackbox BMC tool is called Bounce. Bounce supports multiple 01X, QBF, and hybrid encoding modes, as well as multiple QBF-solvers. The basic workflow of our tool is shown in Figure 3. This workflow consist of three major stages: a BMC problem encoder and a SAT-solver with Craig interpolation and unsat-core support; a heuristic search based component that in the case of 01X-hardness, finds the reasons for this; and thirdly, a hybrid Z/Z_i BMC problem encoder and QBF-solver.

In the first stage of the workflow, a behavioral level VHDL or Verilog circuit description is taken, and blackboxes are inserted for the components that are not fully specified. Then a small VHDL wrapper is added to the circuit so the reset (and/or other) functionality can be controlled. This allows us to initialize a circuit into a predefined state which is sometimes required. For instance, some circuits are only guaranteed to operate correctly if initialized properly. Once this completed, the circuit is then compiled with Synopsys Design Compiler (Version B-2008.09) and linked to a gate library containing only basic one and two input logic gates and storage elements.

The resulting gate level HDL code is then used to generate the BMC equation from Section 2. This is then sent to our multi-threaded SAT-solver MiraXT [9] which includes additional support for Craig interpolation and unsat-core production. If the solver for a specific unrolling returns *true*, the resulting variable assignment is our counterexample and the problem is classified as 01X-easy. Otherwise, if the solver continually returns *false* for each unrolling, then at some point a fixed-point should be reached. When a fixed-point is found, we say the problem is 01X-hard, and will show in Section 3.2 that this means no satisfiable solution using 01X-encoding will be possible.

For 01X-hard instances, we then can use the Craig interpolants (or alternatively the unsat-core of the problem at the fixed-point depth), to find the reasons why the instance is unsatisfiable. This is done by tracing the Craig interpolants backwards to the blackbox outputs, or scanning the unsat-core for blackbox related variables. The strategies Bounce uses for this are discussed in Sections 3.3 and 3.4. Note, however, that these

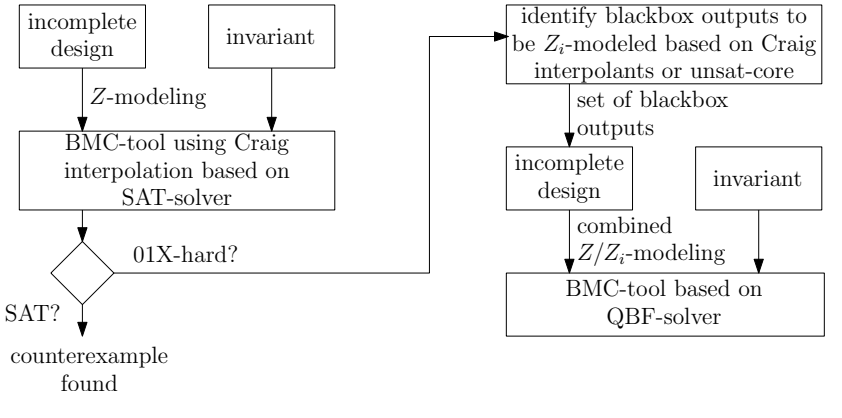

Fig. 3. Workflow

strategies are not guaranteed to find all necessary blackbox outputs (see Section 3.5) but on practical problems they seem to perform well enough.

Finally, once a list of blackbox outputs that need to be modeled precisely is identified, we can then use our optimized techniques (discussed in Section 3.1) to produce hybrid problems containing both Z- and Z_i-modeled blackboxes, hopefully resulting in an easier to solve QBF formula (due to less universally quantified variables). If the QBF-solver returns *true*, we know that the invariant will always be unsatisfied within the current design, irrelevant of the implementation of the blackbox (i.e. the circuit cannot be fixed by adding functionality to the blackbox). Otherwise, when a QBF-solver continually returns *false* for every unrolling of the problem, we cannot prove anything about the current invariant, unless we are able to prove the maximum depth of the circuit, which in practice can be infeasible.

3.1 Optimizing Blackbox Bounded Model Checking

The 01X transformation scheme maps the three logical values 0, 1 and X to the binary tuples (0,1), (1,0) and (0,0), respectively. However, in [5,6,7] all signals were encoded using 01X-logic. This resulted in a doubling of the number of variables in SAT formula that transformation functions would produce. We have extended this mapping so that signals not associated with the blackbox require only the regular one bit encoding. To combine 01X- with 01-logic, we introduce the new transformation functions:

$$AND_{01X}((g^{high}, g^{low}), f) := (g^{high} \cdot f, g^{low} + \neg f)$$
$$OR_{01X}((g^{high}, g^{low}), f) := (g^{high} + f, g^{low} \cdot \neg f)$$

Note, that in contrast to [6] we allow arbitrary sequential behavior of every blackbox (i.e. a blackbox output can produce different output values for the same input values at different time steps). Hence we do not care about the blackbox inputs. Furthermore, with respect to Z/Z_i-modeling in [6], and covered in Section 2, each Z_i-modeled blackbox output introduced two variables and quantifier alternations. However, using our new transformation functions this is no longer required, and we can encode all Z_i-modeled blackbox outputs with only one bit as shown on the right side of Figure 2.

When encoding the problems as QBF, we have an additional optimization that can reduce quantifier alternations even further. In $pref_1$ equation from Section 2, the inputs in each step can 'react' to the values of the blackbox outputs from the previous steps. However, in total this quantifier prefix yields at most $2 \cdot (k+1)$ quantifier alternations when unfolding k times. This number can be lowered to 2 when restricted to uniform counterexamples [14], having one block of the existential quantified inputs of all unfolding depths followed by all universally quantified blackbox outputs and then all existentially quantified Tseitin-variables.

$$\underbrace{\exists x_{0,0}, \ldots, x_{n,k}}_{\text{primary inputs depth 0...k}} \quad \underbrace{\forall Z_{0,0}, \ldots, Z_{m,k}}_{\text{blackbox outputs depth 0...k}} \quad \underbrace{\exists H_0, \ldots, H_k}_{\text{Tseitin depth 0...k}} \text{ CNF} \qquad (pref_2)$$

This formulation implies the first one, since every uniform counterexample is also a counterexample resulting from the first formulation, but we avoid the number of

quantifier alternations increasing with the unfolding depth. However, this formulation is not as exact, and may not be able to verify as many problems or invariants.

3.2 Craig Interpolation and Proving 01X-Hardness

In this section we will introduce Craig interpolants [3] and how interpolation can lift a classical BMC procedure to a complete model checking technique to prove invariants [11]. Later we describe how we benefit from this procedure.

Theorem 1 (Craig). *Given two propositional formulas A and B with the property that $A \wedge B$ is unsatisfiable, then there exists a Craig interpolant C for A and B. This Craig interpolant has the following properties:*

- *C contains only variables which occur in A and B (*AB-common variables*).*
- $\models A \Rightarrow C$
- $\models C \Rightarrow \neg B$

Craig interpolants in BMC are used as an over-approximated forward image of reachable states in a transition system. If the computed over-approximated forward image reaches a fixed-point, that is no new states are reachable, and the given invariant still holds, no counterexample is possible for any unrolling depth. Let I_k be the initial state, P_k the invariant to disprove and $T_{i,i+1}$ the transition relation from a state at time step i to a state at time step $i+1$[1]. After showing that $I_0 \wedge \neg P_0$ is unsatisfiable (that is initially the property is not violated), the procedure first solves the BMC formula $\Phi = A \wedge B$, where $A := R_0 \wedge T_{0,1}$, $B := \neg P_1$ and initially $R_0 := I_0$. If Φ is unsatisfiable then a Craig interpolant C_1 for the formulas A and B is computed[2]. By $A \Rightarrow C_1$, the interpolant C_1 is an over-approximation of the states reachable in one step from R_0. If this over-approximation shifted to the zeroth instantiation of the variables (as described by C_0) is a subset of the so far reachable states, that is $C_0 \Rightarrow R_0$, then further transitions can only lead to states already characterized by R_0. As a consequence, the target states are unreachable and the verification procedure terminates. Otherwise, we expand the set of reachable states such that it also covers the reachable states given by the shifted interpolant, that is $R_0 := R_0 \vee C_0$. Then, the procedure is iterated until the above termination criterion holds. For a more detailed account, confer [11].

As we are focusing on BMC problems for hardware designs, the only variables occurring in both formulas A and B are the latch variables. Now we describe how this procedure can be used to classify incomplete designs. This not only prevents the 01X BMC tool from endlessly returning *false*, but also provides us with a proof that we must use a more precise, but harder to solve QBF formulation.

Proving 01X-Hardness. For a pure 01X-encoded BMC problem, unsatisfiability for every unfolding depth has two possible reasons. First, as demonstrated earlier, Z-modeling all blackbox outputs may be too coarse. Second, there may exist no counterexample. However, in both of these cases 01X-encoding is not suitable to get to a result. This motivates the following definition of *01X-hardness*.

[1] In the following a lower index denotes the timed instantiation of a boolean formula.

[2] Note, that C_1 only contain *AB*-common variables.

Definition 1 (01X-hardness). *An incomplete design combined with an invariant is 01X-hard iff the pure 01X-encoded BMC problem is unsatisfiable for all unfoldings.*

Example. Figure 4 shows a 01X-hard incomplete design with two storage elements q_0 and q_1, two primary inputs x_0 and x_1 and one blackbox with two outputs Z_0 and Z_1. For this design we want to disprove the invariant $AG(\neg q_0 \vee \neg q_1)$ stating that this circuit never reaches a state where both storage elements are *true* at the same time.

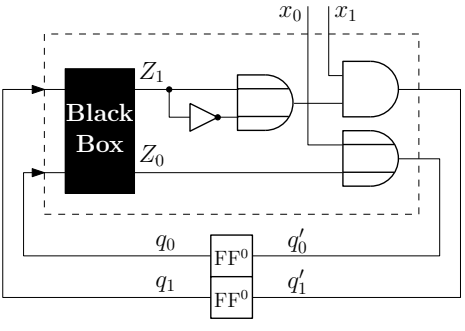

Fig. 4. 01X-hard incomplete design

Using Z-modeling for blackbox output Z_1, one can see that the assigned X value propagates to the storage element q_1 independent of the value x_1. Obviously this fact holds for every unroll depth.

In order to apply BMC with Craig interpolation to this example, the entire circuit is Jain encoded. Figure 5 illustrates the first unfolding $I_0 \wedge T_{0,1} \wedge \neg P_1$. The gates responsible for the unsatisfiability (unsat-core) are highlighted in gray. Applying the construction rules for Craig interpolants to this unsat-core yields the formula $C_1^0 = \neg({q'}_1^h)$. This formula represents an over-approximation of the reachable states after one transition step. Using this set of states as new initial states, the Craig interpolant C_1^1 derived from $C_0^0 \wedge T_{0,1} \wedge \neg P_1$ is equivalent to the one computed before and thus a fixed-point is reached and the 01X-hardness is proven.

Here, the Craig interpolant $C_1^0 = \neg({q'}_1^h)$ forces the storage element ${q'}_1 = ({q'}_1^h, {q'}_1^l)$ to be either $0_{01X} = (0, 1)$ or $X_{01X} = (0, 0)$. In 01X-logic, X represents both 0 and 1 simultaneously, meaning it is not possible to determine under which circumstances the values 0 and 1 appear. This can only be determined using Z_i-modeling. In our example, when Z_1 is Z_i-modeled, the output of the OR-gate is a constant 1 for all values of Z_1. The resulting QBF formula is then satisfiable after one transition step. Furthermore, this shows it is sufficient to model only the first blackbox output using QBF.

Now, we are in the advantageous situation to gather information about the reason for the unsatisfiability of every unrolling depth. This information is located in the Craig interpolant and also in the underlying unsat-core. How this information can be exploited to refine our encoding, will be discussed in the next sections.

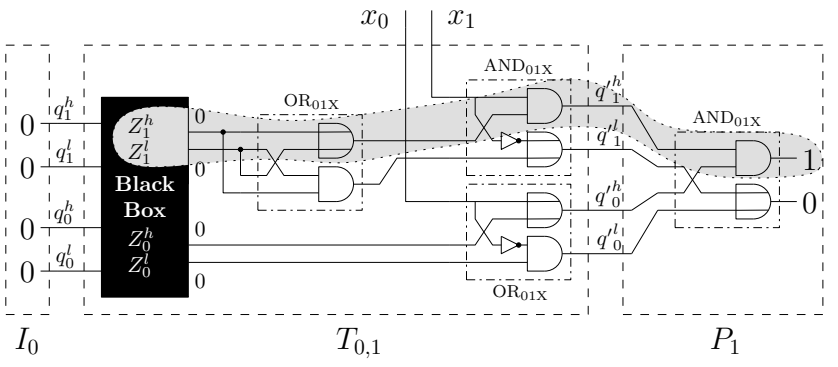

Fig. 5. Jain encoded incomplete design unfolded one time

3.3 Exploiting Craig Interpolants

For a given incomplete design and an invariant we apply the unbounded model checking procedure described in the previous section with all blackbox outputs Z-modeled and thus 01X-encoded. Once a fixed-point is reached the last Craig interpolant is analyzed.

This interpolant only contains latch variables[3], which are collected in a set L. All blackbox outputs affecting these latches are one reason for the 01X-hardness. Starting from each latch in L, a recursive backward traversal on the incomplete design is performed. If a blackbox output is reached, this blackbox output is marked to be Z_i-modeled. When we reach a latch which is not yet an element of L, we add it to L and continue the backward traversal until a fixed-point in L is reached. With this algorithm we identify at least all blackbox outputs having influence on the latches in L.

Coming back to our example in Figure 5, traversing backward from $L = \{q'^h_1\}$ only blackbox output Z_1 is marked to be Z_i-modeled, which is sufficient to prove the unrealizability applying the combined modeling technique presented in Section 3.1.

3.4 Utilizing Unsatisfiable Cores

Instead of exploiting the Craig interpolant as described above we also can make use of the unsat-core that the SAT-solver produces. After proving 01X-hardness we determine the unsat-core of the BMC problem of the unrolling depth where the fixed-point was found (it is clear that this problem must be unsat as the 01X-hardness was already proven). To determine the unsat-core, the solver's conflict analysis routine is modified so that each new conflict clause contains a link to the original problems clauses it was derived from. Simply put, this means the unsat-core contains all the clauses needed by the underlying SAT-solver to derive the empty clause. Among these clauses we detect every literal that represents a Jain-encoded blackbox output. All these outputs will now be marked to be Z_i-modeled.

In our example the clauses derived from the gray highlighted gates build the unsat-core and thus Z_1 is immediately detected to be Z_i-modeled. With this approach we

[3] Due to the 01X-encoding these latches are still Jain encoded.

only identify blackbox outputs which directly have an influence on the unsatisfiability, whereas the backward traversal of the method based on Craig interpolants also can identify blackbox outputs which may not effect any relevant latch.

3.5 Challenges and Limitations

For a 01X-hard system there is not necessarily only one single proof or unsat-core. This fact can result in situations, where not all necessary Z_i-modeled blackbox outputs are identified in order to find a counterexample. If we know the structure of the circuit in detail this problem can be avoided by decomposing the circuit and applying our method for each component. Since in general this is not a trivial task, a further approach is to collect the latch variables of *all* computed interpolants during the proof, not only those involved in the successful fixed-point-check. This is motivated by the heuristic search of the underlying SAT-solver computing the Craig interpolants. However, it will be shown in Section 4 that our unsat-cores seem to be sufficient on our industrial benchmarks to prove unrealizability.

Further, it is not guaranteed that a fixed-point will be found. If so, we can not prove the 01X-hardness of the system, but we can continue our procedure by aborting at a given timeout or depth, and analyzing the Craig interpolants or unsat-cores computed so far. After several unrollings, this technique would hopefully allow us to still identify the required blackbox outputs. However, as will be shown in Section 4, both of these limitations did not present problems in any of the circuits that were tested.

4 Experimental Results

To evaluate our methods we used industrial opensource FPGA verified designs from opencores.org [15] and some instances of the VLIW ALU benchmark suite presented in [14]. This allows us to test our ideas on implementable circuits designs that are very similar to the ones used in industry. In our current design flow, the blackboxes were inserted into the behavioral VHDL source code of each design allowing us to select and replace specific entities (i.e. entire multipliers, shifters, dividers) from each circuit. The same method was used to insert errors into some circuits. A description of each benchmarks circuit is given below. To compile the behavioral VHDL source code of each design, Synopsys Design Compiler Version B-2008.09 was used along with a minimized gate library containing only one and two input basic logic gates and latches. All benchmarks were run on a dual AMD Opteron(tm) 250 processor machine (2.4GHz) with 4GB of RAM, running the 2.6.24 SMP enabled Linux kernel. Lastly, for all benchmarks a timeout (TO) value of 3600 seconds was used.

– Plasma - A simple opensource pipelined 32-bit RISC microprocessor supporting all MIPS Version I user mode instructions. The multiplier and shifter inside the ALU were replaced with blackboxes, and the boot loader was simplified. Proper functionality and reliability of the boot loader are being verified. The circuit contains 16,603 gates and 2,463 latches.

– FPU - An IEEE-754 compliant pipelined double precision floating point unit that supports four operations (+, -, *, /) and multiple rounding modes and exceptions. VHDL and gate level faults were inserted into the FPU. The multiplication and division units were replaced with blackboxes and some of the functionality of the remaining units was tested. The remaining circuit contains 21,280 gates and 2,701 latches.
– UART - A configurable, pipelined serial transmitter/receiver UART pair. Here, by defining the UART controller as a blackbox, we verify that all possible 8 bit inputs to the UART will be transmitted and received properly at different bit rates. The circuit contains 555 gates and 70 latches.
– QALU - A SoC design containing a configurable VLIW ALU and Timer. The VLIW ALU consists of four separate functional units (1xLogic, 2xArithmetic, 1xLoad/Store) and multiple working registers. The 12 bit Timer has set, reset, exception, and overflow functionality. Blackboxes replaced different arithmetic units in the two ALU sizes and some of the parts of the Timer. The functionality of the remaining Timer and ALU logic unit which contains gate level design errors are being verified. QALU_32 contains 31,866 gates, 538 latches, and QALU_64 contains 35,496 gates and 1,054 latches.

The results for the first stage of our tool are presented in Table 1. Here, the first column is the benchmark name, followed by the found counterexample or fixed-point depth and time (CE/FP Depth and Time). If the benchmark is 01X-hard, the number of detected blackbox outputs for both the Craig interpolant and unsat-core methods are given (# Detected Craig and Unsat). The first fact that is apparent from this table is that almost half of the properties we check are 01X-easy, meaning they can be solved using only a SAT-solver. Secondly, if problems are 01X-hard, only a small fraction of blackbox outputs are selected to be modeled using QBF. Overall, unsat-core method seems to outperform Craig interpolation. Finally, for all these benchmarks the first phase of our tools does not require a significant amount of time even though this circuits are quite

Table 1. Initial 01X results

Bench.	CE/FP Depth	CE/FP Time	01X Hard	# Detected Craig	# Detected Unsat
FPU-ec01	27	67.39	no	—	—
FPU-ec02	28	70.36	no	—	—
FPU-ee01	27	75.16	no	—	—
FPU-hc01	33	54.78	yes	3/141	6/141
FPU-hc02	33	58.87	yes	3/141	6/141
FPU-he01	32	57.14	yes	139/141	6/141
FPU-he02	49	433.52	yes	141/141	34/141
Plasma-ec01	15	8.86	no	—	—
Plasma-ee01	10	6.32	no	—	—
Plasma-hc01	24	22.82	yes	64/64	12/64
Plasma-hc03	46	55.09	yes	64/64	12/64
Plasma-he01	28	26.11	yes	64/64	10/64
UART-ec01	126	19.93	no	—	—
UART-ec02	245	74.44	no	—	—
UART-ec03	475	354.3	no	—	—
UART-hc01	8	0.29	yes	01/16	01/16
UART-hc02	47	3.93	yes	08/16	08/16
UART-hc03	4	0.12	yes	01/16	01/16
UART-hc04	133	39.88	yes	08/16	08/16
UART-hc05	15	0.76	yes	01/16	01/16
UART-hc06	121	23.07	yes	08/16	08/16
UART-hc07	4	0.14	yes	01/16	01/16
qualu32-e	4	1.37	no	—	—
qualu64-e	4	1.45	no	—	—
qualu32b-h	4	2.46	yes	2/66	2/66
qualu64b-h	2	1.93	yes	2/386	2/386

Table 2. $pref_1$ encoding results

Bench.	CE Depth	AIGsolve Craig	AIGsolve Unsat	AIGsolve All Z_i	QMiraXT Craig	QMiraXT Unsat	QMiraXT All Z_i	QuBE Craig	QuBE Unsat	QuBE All Z_i
FPU-hc01	27	33.41	74.44	52.88	14.10	8.00	7.93	44.5	44.88	42.13
FPU-hc02	28	54.99	35.77	64.79	13.79	11.12	10.69	54.61	53.98	49.41
FPU-he01	27	64.96	62.15	57.96	16.13	9.01	8.76	TO	143.83	TO
FPU-he02	27	TO	TO	TO	TO	74.03	TO	TO	TO	TO
Plasma-hc01	10	0.03	0.05	0.05	0.06	0.03	0.03	0.04	0.03	0.04
Plasma-hc03	15	0.09	0.05	0.07	0.08	0.07	0.09	0.05	0.04	0.03
Plasma-he01	10	0.06	0.04	0.04	0.09	0.03	0.05	0.05	0.04	0.05
UART-hc01	126	0.70	0.70	1.13	0.82	0.91	0.90	1.27	1.31	1.53
UART-hc02	126	2.12	2.21	2.12	0.78	0.86	0.88	1.80	1.78	1.76
UART-hc03	245	2.49	2.52	4.85	2.06	2.3	2.33	4.85	4.80	5.49
UART-hc04	245	13.42	13.38	13.38	2.1	2.29	2.42	5.78	6.05	6.30
UART-hc05	475	8.61	8.69	25.75	5.86	5.9	5.87	19.02	19.04	22.78
UART-hc06	475	177.98	178.07	177.74	6.18	6.12	6.17	23.58	23.57	23.93
UART-hc07	1,860	156.97	157.56	1,103.54	66.86	68.14	66.37	400.66	419.00	582.14
qualu32b-h	9	105.28	105.58	TO	TO	TO	TO	TO	TO	258.60
qualu64b-h	9	576.94	575.45	TO	TO	TO	TO	TO	TO	718.29
Total Solved		15	15	13	13	14	13	12	13	14
Total Time		1,198.05	1,216.66	1,504.3	128.91	188.81	112.49	556.21	718.35	1,712.48

complex. For example, a circuit with 30,000 gates could require well over 100,000 clauses for the transition relation $T_{i,i+1}$ described in Section 2 alone. In fact, some of the benchmarks contain over a million variables. Also, as an important side note, the inclusion of blackboxes in the FPU allows an extra level of abstraction that makes this verification possible. If the FPU contained the pipelined multiplier and divider, we are no longer able to verify the functionality of adder or subtractor as the entire circuit is too complex. This highlights another application of blackbox modeling and our tool.

For the remaining 01X-hard problems we ran the second phase of our tool using the two different QBF prefix introduced as $pref_1$ and $pref_2$ in Section 2 and 3. The results for each type of prefix are shown in Tables 2 and 3. In both tables, we test three different QBF-solvers, mainly: AIGsolve [16]; QMiraXT [10]; and QuBE [4]. QuBE was chosen as it was the only solver from the 2007 and 2008 QBF Competitions [17] that offered good performance on the blackbox benchmark set. AIGsolve and QMiraXT are newer solvers that have also been shown to perform well on blackbox benchmarks. For each of these solvers, we compared the performance using the Craig interpolation and unsat-core blackbox detection techniques, and the case where all blackboxes were modeled using Z_i. Finally the counterexample depth for each circuit is given (CE Depth).

In Table 2, when we are using the more complex, but also more accurate QBF prefix, AIGsolve seems to perform the best, followed by QMiraXT and QuBE. Note, that on the problems that QMiraXT was able to solve, it is generally the fastest. Also, with the exception of QuBE, the unsat-core technique performs best, and the *All* Z_i case performs the worst with respect to time and problems solved. The reason for the reverse performance trends with QuBE remain unclear, especially considering it as a DPLL based QBF-solver like QMiraXT. More interesting, is that this table shows that large

Table 3. $pref_2$ encoding results

Bench.	CE Depth	AIGsolve Craig	AIGsolve Unsat	AIGsolve All Z_i	QMiraXT Craig	QMiraXT Unsat	QMiraXT All Z_i	QuBE Craig	QuBE Unsat	QuBE All Z_i
FPU-hc01	27	75.91	34.04	54.48	8.26	8.61	7.88	47.27	47.19	42.41
FPU-hc02	28	58.81	35.74	65.06	11.49	11.15	27.07	55.68	58.41	50.87
FPU-he01	27	62.61	55.66	59.92	9.19	9.15	17.01	TO	119.65	TO
FPU-he02	27	TO	TO	TO	113.46	TO	116.30	TO	TO	TO
Plasma-hc01	10	0.05	0.03	0.04	0.06	0.04	0.07	0.04	0.03	0.06
Plasma-hc03	15	0.04	0.04	0.06	0.07	0.07	0.06	0.06	0.03	0.06
Plasma-he01	10	0.05	0.02	0.05	0.05	0.06	0.08	0.05	0.02	0.04
UART-hc01	126	0.68	0.67	1.15	0.95	0.81	1.00	1.30	1.37	1.52
UART-hc02	126	2.19	2.07	2.17	0.88	0.9	0.77	1.64	1.79	1.62
UART-hc03	245	2.57	2.43	4.81	2.7	2.48	2.53	5.04	4.89	5.51
UART-hc04	245	13.38	13.23	13.34	2.58	2.66	2.81	5.59	5.75	5.60
UART-hc05	475	8.76	8.84	25.60	7.09	6.92	7.45	18.54	18.64	22.13
UART-hc06	475	177.83	177.72	177.9	7.14	7.25	7.04	22.99	22.90	22.87
UART-hc07	1,860	161.98	163.47	1,078.84	TO	TO	TO	411.46	412.71	565.46
qualu32b-h	9	108.77	108.85	97.55	18.77	18.69	4.41	TO	TO	28.20
qualu64b-h	9	604.72	606.06	242.69	134.6	134.55	9.27	TO	TO	85.20
Total Solved		15	15	15	15	14	15	12	13	14
Total Time		1,278.35	1,208.87	1,823.66	317.29	203.34	203.75	569.66	693.38	831.55

benchmarks, as well as really deep (i.e. high depth) benchmarks can be solved with current QBF-solvers in a reasonable amount of time.

Table 3 then shows the results for our more compact encoding style. For all these benchmarks the number of quantifier alternations is restricted to 2, unlike the results in Table 2 where some benchmarks contain thousands of alternations. The new simpler prefix seems to be more effective overall. This is shown by the fact that in Table 3 there are only 16 unsolved instances, where as in Table 2 there are 22. This is especially true for QMiraXT and AIGsolve as in almost all cases $pref_2$ performs better. QuBE benefits from this as well, but only in the *All* Z_i case where the total solve time is cut in half. Finally, even though $pref_2$ is less accurate than $pref_1$, it did not effect the results as in all cases $pref_2$ was still exact enough to solve all the benchmarks.

Lastly, when considering the larger picture, the use of blackbox verification techniques is now feasible. In Table 1 we showed that using 01X-logic alone is good enough to verify many properties. Furthermore, this table and the results in Tables 2 and 3 show that our blackbox detection procedures perform well, with our optimized encoding in Table 3 performing better. Finally, we showed that modern QBF-solvers perform well on a range of benchmarks, showing that they are finally ready for industrial uses.

5 Conclusion

In this work, we presented our tool Bounce that can automatically and efficiently verify a wide array of blackbox BMC problems. We introduced a novel and efficient encoding for the BMC problem of incomplete designs when combining Z- and Z_i-modeling

techniques. We also provided a procedure for proving the 01X-hardness of an incomplete design with the help of Craig interpolation. Furthermore, we showed how the Craig interpolants and unsat-cores of a problem can be used for heuristically determining which blackbox outputs should be Z_i-modeled in order to find a counterexample. Moreover, we showed that all these techniques and our tool perform well on a large set of industrial problems. Lastly, we compared our tools performance when connected to three different state-of-the-art QBF-solvers.

Our results show that blackbox verification is becoming feasible, and that it can be used for the testing of designs with incomplete information. These include early stage prototypes where every component is not yet developed, or SoC designs where the exact functionality of a component is unknown. Furthermore, the ability of blackbox methods to verify designs that are to complex if the entire design is considered also poses multiple interesting applications. Additionally, we are currently looking at ways to make the blackboxes gray by introducing the ability to give each blackbox certain properties, opening up many more opportunities for our tool in the future.

References

1. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
2. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded Model Checking Using Satisfiability Solving. Formal Methods in System Design 19, 7–34 (2001)
3. Craig, W.: Linear Reasoning: A New Form of the Herbrand-Gentzen Theorem. Journal of Symbolic Logic 22(3), 250–268 (1957)
4. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas. Journal of Artificial Intelligence Research (JAIR) 26, 371–416 (2006)
5. Herbstritt, M., Becker, B.: On SAT-based Bounded Invariant Checking of Blackbox Designs. In: Microprocessor Test and Verification Workshop (MTV), pp. 23–28 (2005)
6. Herbstritt, M., Becker, B.: On Combining 01X-Logic and QBF. In: Moreno Díaz, R., Pichler, F., Quesada Arencibia, A. (eds.) EUROCAST 2007. LNCS, vol. 4739, pp. 531–538. Springer, Heidelberg (2007)
7. Herbstritt, M., Becker, B., Scholl, C.: Advanced SAT-Techniques for Bounded Model Checking of Blackbox Designs. In: Microprocessor Test and Verification (MTV), pp. 37–44 (2006)
8. Jain, A., Boppana, V., Mukherjee, R., Jain, J., Fujita, M., Hsiao, M.: Testing, Verification, and Diagnosis in the Presence of Unknowns. In: IEEE VLSI Test Symposium (VTS), pp. 263–269 (2000)
9. Lewis, M., Schubert, T., Becker, B.: Multithreaded SAT Solving. In: 12th Asia and South Pacific Design Automation Conference, pp. 926–931 (2007)
10. Lewis, M., Schubert, T., Becker, B.: QMiraXT – A Multithreaded QBF Solver. In: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (2009)
11. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)

12. Nopper, T., Scholl, C.: Approximate Symbolic Model Checking for Incomplete Designs. In: Formal Methods in Computer-Aided Design, pp. 290–305 (2004)
13. Nopper, T., Scholl, C.: Flexible Modeling of Unknowns in Model Checking for Incomplete Designs. In: 8. GI/ITG/GMM Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (2005)
14. Nopper, T., Scholl, C., Becker, B.: Computation of Minimal Counterexamples by Using Black Box Techniques and Symbolic Methods. In: IEEE Int'l Conf. on Computer-Aided Design, pp. 273–280 (2007)
15. OpenCores, `http://www.opencores.org`
16. Pigorsch, F., Scholl, C.: Exploiting Structure in an AIG Based QBF Solver. In: Conf. on Design, Automation and Test in Europe (DATE), April 2009, pp. 1596–1601 (2009)
17. QBF Solver Evaluation, `http://www.qbflib.org/index_eval.php`
18. Scholl, C., Becker, B.: Checking Equivalence for Partial Implementations. In: Design Automation Conf., pp. 238–243 (2000)