Unbounded Scalable Verification Based on Approximate Property-Directed Reachability and Datapath Abstraction

Suho Lee and Karem A. Sakallah

University of Michigan and Qatar Computing Research Institute

Abstract. This paper introduces the Averroes formal verification system which exploits the power of two complementary approaches: counterexample-guided abstraction and refinement (CEGAR) of the design's datapath and the recently-introduced IC3 and PDR approximate reachability algorithms. Averroes is particularly suited to the class of hardware designs consisting of wide datapaths and complex control logic, a class that covers a wide spectrum of design styles that range from general-purpose microprocessors to special-purpose embedded controllers and accelerators. In most of these designs, the number of datapath state variables is orders of magnitude larger than the number of control state variables. Thus, for purposes of verifying the correctness of the control logic (where most design errors typically reside), datapath abstraction is particularly effective at pruning away most of a design's state space leaving a much reduced "control space" that can be efficiently explored by the IC3 and PDR method. Preliminary experimental results on a suite of industrial benchmarks show that Averroes significantly outperforms verification at the bit level. To our knowledge, this is the first empirical demonstration of the possibility of automatic scalable unbounded sequential verification.

1 Introduction

This paper explores the possibility of scaling formal verification of complex hardware systems beyond what is possible today by exploiting the power of two complementary approaches: counterexample-guided *datapath abstraction and refinement* and the recently-introduced IC3 [1] and PDR [2] approximate reachability algorithms. Our prototype implementation of this verification framework, which we call the Averroes system for sequential verification, is premised on the conjecture that the complexity of sequential verification can be reduced significantly by a) abstracting away irrelevant datapath "state" that basically clutters reachability analysis without providing any useful guidance for its convergence, and b) performing approximate reachability on this abstracted state space. The approach can be viewed as a "layering" of two CEGAR loops: an inner loop that performs approximate reachability on the datapath-abstracted state space, and an outer datapath refinement loop that tightens the abstraction based on the spurious counterexamples generated by the inner loop. Initial empirical evaluation of this approach shows that it significantly outperforms bit-level verification on a set of industrial RTL benchmarks and suggests that the combination of

A. Biere and R. Bloem (Eds.): CAV 2014, LNCS 8559, pp. 849–865, 2014.

datapath abstraction and approximate reachability makes it possible to perform automatic unbounded scalable verification on real-world industrial benchmarks.

The rest of the paper is organized in 9 sections. Sections 2 and 3 briefly review previous work and cover preliminaries. We then provide a high-level description of the IC3/PDR approach in Section 4 followed by an example, in Section 5, to motivate datapath abstraction. Sections 6 and 7 provide an overview and a detailed description of the Averroes algorithm. Preliminary experimental evaluation is covered in Section 8, and Section 9 ends the paper with some conclusions.

2 Previous Work

The recently introduced IC3 algorithm [1] and its re-implementation in PDR [2] represent a major milestone in the decades' long quest for scalable model checking (MC). Both can be described as SAT-based induction methods and both share some features of the earlier attempts at using induction [3,4]. In particular, assuming that a given safety property P holds but is not inductive (i.e., is not closed under the transition relation), induction methods can be viewed as ways of performing *approximate reachability* with the goal of finding an assertion that strengthens (i.e., restricts) P so that it becomes an *inductive invariant* [5]. Alternatively, such methods can be seen as an application of counterexample-guided abstraction refinement (CEGAR) [6,7,8] whereby overapproximations of the reachable states are *refined* iteratively until enough unreachable states have been eliminated to prove that P does in fact hold or to produce a counterexample trace. Eliminating the need to compute exact reachability makes it possible for induction methods to converge in a number of iterations that can be much smaller than the sequential depth of the transition relation. Additionally, induction methods can be applied without having to unroll the transition relation which allows them to have better scalability than the earlier memory-intensive BDD [9,10] or BMC [11,12] approaches.

Several extensions of the IC3/PDR approach have already been proposed. In [13], the authors describe an extension to PDR that enables reasoning about nonlinear predicate transformers and linear real arithmetic. In [14], IC3-style reachability is generalized to handle transition systems described by first-order formulas, combined with control flow graph (CFG) analysis, and used to verify safety properties of software. The work that is closest to ours is [15] where Kurshan's *visible variable abstraction* [6] is layered on top of IC3 to significantly scale performance, over just IC3, on a set of large industrial benchmarks.

The datapath/control dichotomy has been addressed by many authors. In [16], properties are classified as control, data, and data/control, and various degrees of data sensitivity are introduced and analyzed. A formal model of systems that can be decomposed into an interconnection of datapath and controller modules is described in [17] and used to automatically generate an abstraction by datapath reduction. In [18], datapath abstraction is shown to yield significant savings in both runtime and memory in a symbolic verification system. The Reveal verification system [19] performs automatic datapath abstraction from Verilog RTL models and iteratively refines them in a standard CEGAR flow. It is important to note that all of these approaches were limited to bounded verification that involved unrolling a design's transition relation a fixed number of times. To our

knowledge, the approach described in this paper is the first to couple datapath abstraction and refinement with unbounded model checking.

3 Preliminaries

Our concern in this paper is to determine if a sequential hardware design satisfies a specified safety property. We assume that the design's behavior is encoded by a transition relation $T(X, X^+)$ where X and X^+ denote n-bit vectors of current- and next-state variables. In general, T is easily derivable from any suitable design description, e.g., a netlist or a model in a hardware description language such as Verilog. Furthermore, T may involve additional non-state variables including primary inputs and signals that model combinational blocks in the design. These extra variables are assumed to be part of the definition of T even when not explicitly listed. In the sequel, we will assume that T is available as a propositional formula in conjunctive normal form (CNF). We also assume the existence of two additional predicates (also available as CNF formulas) on the design's state variables: $I(X)$ denoting the design's initial (reset) state(s), and $P(X)$ denoting the set of states that satisfy the desired safety property. We will informally refer to the states that satisfy (resp. violate) $P(X)$ as good (resp. bad or error) states. Finally, we denote by $R(X)$ the design's set of reachable states, i.e., those states that can be reached from $I(X)$ in one or more transitions. A *trace* Π is a state sequence $\langle s_0(X), s_1(X), \cdots, s_{k-1}(X)\rangle$ such that each s_i is a set of states, $s_0(X) \in I(X)$, and $s_i(X) \wedge T(X, X^+) \rightarrow s_{i+1}(X^+)$ holds for $0 \leq i \leq k-2$. The length of a trace with k states is $k-1$. An empty trace is one whose state sequence (as a set) is empty; its length is undefined.

The verification task can now be stated as follows: prove that all states in R are good or derive a counterexample trace that starts in I and ends in $\neg P$. The algorithms we consider in this paper solve this task by induction. Using Bradley's terminology [5], these algorithms consist of two main steps:

- **Initiation**: prove that the initial states are good: $I \rightarrow P$.
- **Consecution**: derive a *strengthening* assertion $A(X)$ such that $A \wedge P \wedge T \rightarrow A^+ \wedge P^+$, where A^+ and P^+ are shorthand for $A(X^+)$ and $P(X^+)$.

What distinguishes these algorithms from earlier induction approaches is that the strengthening assertion A is derived incrementally rather than monolithically [1,2]. Furthermore, in contrast to methods that perform exact image computations (symbolically using BDDs or through SAT-based unrolling of the transition relation), these algorithms create and repeatedly tighten a sequence of approximate reachability frontiers without having to unroll the transition relation. Thus, they do not suffer from the memory explosion inherent in earlier approaches and are demonstrably more scalable. The first such algorithm in this category was Bradley's IC3 [1] which was subsequently re-implemented and enhanced by Een et al. [2] who dubbed it PDR. In the rest of this paper we will refer to this class of algorithms as IC3/PDR to emphasize their incremental inductive nature (the first two Is in IC3) and their property-directed slant (the PDR viewpoint).

```
1. trace Reach-CEGAR(T,I,P){
2.   F0 = I;
3.   if (F0 & !P)
4.     then return CE trace;// len(CEX)=0
5.   if (F0 & T & !P')
6.     then return CE trace;// len(CEX)=1
7.   k = 1;
8.   Fk = P;
9.   while (true){
10.    Fk+1 = P;
11.    while (Fk & T & !P')// CTI
12.      if Reachable(CTI,I)
13.        then return CEX trace;// len(CEX)≥k+1
14.        else Refine(1,k+1);
15.    if (Fi = Fi-1 for some 2≤i≤k+1)
16.      then return empty trace;// P holds
17.    k++;
18.  }
19.}
```

Fig. 1. High-Level Pseudo Code for CEGAR-Based Reachability

4 Reachability Approximation and Refinement

For our purposes we find it useful to view the IC3/PDR approach as a clever application of CEGAR whereby a series of reachability overapproximations are systematically refined based on counterexamples to induction (CTIs) [5] until either a) a feasible state sequence from the initial state to an error state (a counterexample trace) is found or b) the refinements become sufficient to render the property being checked inductive, i.e., an overapproximation of the reachable states that satisfies the property is found. A sketch of this approach, loosely mimicking IC3, is given in Fig. 1. The procedure, which we call **Reach-CEGAR**, takes as input T, I, and P, and returns a trace. An empty trace indicates that P holds; otherwise the returned trace represents a counterexample CEX demonstrating how P is violated.

Reach-CEGAR maintains an array of *frontiers* $F_0, F_1, \cdots, F_k, \cdots$ such that $F_0 = I$ and $F_j, j > 0$ is an overapproximation of what is reachable after j steps from I. After checking for 0- and 1-step counterexamples (lines 2 to 6), **Reach-CEGAR** enters its main loop (lines 9 to 18). At iteration $k > 0$, the goal is to check for the existence of CTIs that correspond to counterexample traces whose length is at least $k+1$. Each satisfying assignment to the current-state variables in the query on line 11 is a CTI that is checked to determine if it is reachable from I (line 12). If unreachable, the CTI is used to tighten the approximations of frontiers 1 to $k+1$ (line 14) by constraining them with appropriate *refinement clauses*. This process continues until either a reachable CTI is found (line 13) or all CTIs from the current frontier have been ruled out as unreachable. At that point **Reach-CEGAR** checks for convergence (line 15) which is indicated when two frontier approximations become equal. If converged, **Reach-CEGAR** returns an empty trace signaling that P is satisfied (line 16). Otherwise, it increments the iteration counter (line 17) and proceeds to check for the existence of CTIs that correspond to longer counterexample traces.

This sketch hides many details that are critical to the performance of the algorithm. Specifically, in IC3/PDR **Reachable** and **Refine** are not separate procedures. Instead, the reachability check implied by **Reachable** is decomposed into a collection of 1-step backward reachability checks that are queued and processed in some order. Each such check may spawn further checks and/or yield one or more refinements that are propagated backward and forward to tighten the frontier approximations. The checks and attendant refinements, which are performed through appropriate calls to an incremental SAT solver, are closely choreographed to improve the quality of the derived refinement clauses and speed up convergence. Different implementations will thus yield different refinements that can lead to drastically different performance.

There is, however, a critical detail in the implementation of **Reach-CEGAR** that deserves mention. Let ρ_j denote the CNF formula corresponding to the refinement clauses associated with frame j. With a slight abuse of notation, we will also view ρ_j as a set of clauses. At the beginning of each major iteration k, **Reach-CEGAR** insures that the sets of refinement clauses are distinct and subsumption-free, i.e., $\omega \nrightarrow \upsilon$ where $\omega \in \rho_j$ and $\upsilon \in \rho_i$ for $i \leqslant j$. The frontier overapproximations can now be expressed as:

$$F_j = P \wedge \bigwedge_{i=j}^{k+1} \rho_i, j \in [1, k+1]$$

which in turn implies that $F_1 \rightarrow F_2 \rightarrow \cdots \rightarrow F_{k+1}$, and reduces the convergence check on line 15 to checking that the set of refinement clauses at some frame j has become empty ($\rho_j = 1$). At that point, the refinement clauses at the last frontier serve as an inductive strengthening assertion [5] that helps prove the property: $\rho_{k+1} \wedge P \wedge T \rightarrow \rho^+_{k+1} \wedge P^+$

5 Motivating Example

Fig. 2 gives the Verilog description and corresponding state transition graph (STG) of an example sequential circuit that will serve to demonstrate the potential benefits of combining datapath abstraction with approximate reachability. The circuit clearly satisfies the specified property $P(X, Y) = (Y \leq X)$ since, as can be seen from the STG, the reachable states satisfy $R(X, Y) = (Y = X)$.

When IC3 is run on this example it proves the property after eliminating two CTIs and generating three refinement clauses. At exit the refinement clauses and corresponding frontier approximations are:

$$\begin{array}{ll} \rho_1 = \neg x_1 & F_1 = P \wedge \rho_1 \wedge \rho_2 \wedge \rho_3 \\ \rho_2 = 1 & F_2 = P \wedge \rho_2 \wedge \rho_3 \\ \rho_3 = (\neg x_0 \vee y_0) \wedge (\neg x_1 \vee y_1) & F_3 = P \wedge \rho_3 \end{array}$$

Note that the clause set for frontier 2 is empty ($\rho_2 = 1$) implying that $F_3 = F_2$.

In contrast, PDR proves the property by eliminating 6 CTIs and learning 7 refinement clauses. The difference between the two programs is due to their particular choices for the initial frontier approximations (PDR sets $F_k = 1$ instead of $F_k = P$) and the manner in which they perform backward reachability and refinement. The difference becomes more pronounced when the bit width of the

```
`define W 2
`define MAX `W'b11
module example(CLK);
  input wire CLK;
  reg [`W-1:0] X, Y;
  initial begin
    X = `W'd0;
    Y = `W'd0;
  end
  always @(posedge CLK) begin
    X <= (Y>X)? X :
         ((Y==X) || (X!=`MAX))? (X+`W'd1) : Y;
    Y <= (Y==X)? (Y+`W'd1) :
         ((Y>X) || (X!=`MAX))? Y : X;
  end
  wire P = (Y<=X);
endmodule
```

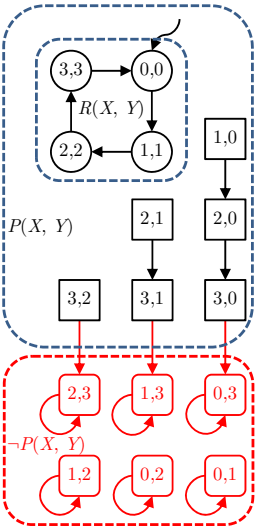


Fig. 2. Verilog description and corresponding STG of an example sequential circuit with a specified safety property. The state variables are 2-bit unsigned integers $X = x_1x_0$ and $Y = y_1y_0$ and their values are used to label the states (X followed by Y) in the STG. The good states are represented by circles (reachable states) and squares (unreachable states); squares with rounded corners correspond to bad states. Note that the circuit's sequential depth is exponential in the bit width W: $2^W = 2^2 = 4$.

state variables in the example is increased from 2 to 64. The results are shown in Table 1. For each bit width, the table compares five measures of performance for IC3 and PDR: runtime, number of frames, number of CTI checks, total and net number of refinement clauses, and total and net number of refinement literals. The number of net refinement clauses and literals reflects the effect of clause subsumption. With a time-out of 1500 seconds, both programs completed the verification up to a bit width of 8; neither program finished for larger bit widths. In most cases PDR outperformed IC3, carrying out many more CTI checks while learning fewer refinement clauses (after subsumption). However, for both programs the number of accumulated refinement clauses grows rapidly as the bit

Table 1. IC3 v. PDR on Example Circuit of Figure 2 for Different Bit Widths

Bit Width	Sequential Depth	Runtime, sec		Frames		CTI Checks		Refinement Clauses				Refinement Literals			
								Total		Net		Total		Net	
		IC3	PDR	IC3	PDR	IC3	PDR	IC3	PDR	IC3	PDR	IC3	PDR	IC3	PDR
2	4.00E+00	0.02	0.02	2	4	2	6	3	12	3	7	5	22	5	12
4	1.60E+01	0.07	0.05	15	15	16	71	84	114	34	32	258	328	87	69
8	2.56E+02	59.59	3.82	232	141	293	4782	31527	6503	740	195	178736	38364	3267	679
16	6.55E+04	T.O.	T.O.	311	1074	402	299511	179776	327581	1241	9576	2273502	4252418	13779	113497
32	4.29E+09	T.O.	T.O.	207	1080	200	313973	28018	327958	325	11431	724242	3932210	6786	126096
64	1.84E+19	T.O.	T.O.	200	923	241	244916	11470	259737	356	8922	636123	2857933	13744	97484

```
1. trace DP-CEGAR(T,I,P){
2.   (T̂, Î, P̂) = DP-Abstract(T,I,P);
3.   Δ = 1; // Initialize datapath lemmas
4.   while (true){
5.     ACEX = Reach-CEGAR(T̂, Î, P̂, Δ) ;
6.     if empty(ACEX)
7.       then return empty trace;// P holds
8.     CEX = DP-Concretize(ACEX);
9.     if Feasible(CEX)
10.      then return CEX trace;// P fails
11.      else Δ = Δ && DP-Refine(ACEX) ;
12.  }
13.}
```

Fig. 3. High-Level Pseudo Code for CEGAR-Based Datapath Abstraction

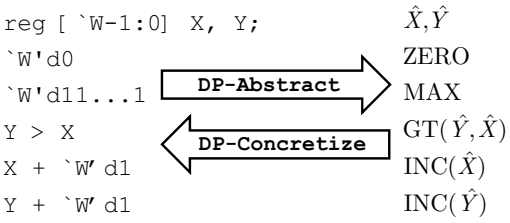
width increases. The large gap between the total and net number of refinement clauses also indicates that both programs learn *weak* clauses that end up being subsumed by stronger ones in later iterations. This suggests that the refinement process gets mired in irrelevant bit-level details that miss the big picture about the property being checked.

6 Datapath Abstraction and Refinement

Our proposed procedure for integrating an IC3/PDR-style reachability computation within a datapath abstraction and refinement framework is summarized in Fig. 3. The initial datapath abstraction is performed by **DP-Abstract** which returns first-order logic (FOL) versions of the bit-level transition, initial, and property formulas (line 2) by, basically, replacing wide datapath signals with uninterpreted terms, and datapath operators and predicates with, respectively, uninterpreted functions and predicates. Single-bit control signals are not abstracted [19]. The abstract formulas are overapproximations of the bit-level versions and, thus, represent a sound abstraction. The procedure then initializes Δ (line 3) which serves as a database of derived datapath refinement lemmas. The reachability computation is carried out by calling a modified version of **Reach-CEGAR** (line 5) that operates on the abstract formulas. Note, in particular, that this version of **Reach-CEGAR** takes as a fourth argument a formula representing the learned datapath lemmas which it augments to all the queries it performs. If **Reach-CEGAR** returns an empty trace, **DP-CEGAR** terminates with the conclusion that the property holds (line 7). However, if **Reach-CEGAR** returns a non-empty abstract trace ACEX, a concrete bit-level version is constructed by **DP-Concretize** (line 8) and checked for feasibility (line 9). If found to be feasible, CEX is returned as a witness for the violation of the property (line 10). Otherwise, a datapath refinement procedure, similar to that in [20], is called to refute this spurious CEX by generating one or more datapath lemmas (line 11), and another round of abstract reachability is invoked. The hypothesis behind this architecture is that the approximate CEGAR-based reachability computation is now performed on an abstracted version of the

design that eliminates irrelevant bit-level details and, thus, is more scalable. More specifically, the abstract CEGAR-based reachability procedure is now operating on approximate reachability frontiers in an abstract approximate state space. The combination of these two orthogonal approximations can lead to drastic pruning by generating two types of refinement lemmas: *reachability* refinement lemmas, and *datapath* refinement lemmas. The latter are "universal" in that they are invariants that tighten the datapath abstraction by relating the uninterpreted terms, functions, and predicates. The former are derived during the approximate reachability computation, except they are now in terms of the abstract state variables. They are, thus, expected to be much stronger than the bit-level refinement clauses derived by IC3 and PDR.

To illustrate the potential of this approach, consider its application to the example sequential design from Section 5. Datapath abstraction creates the following uninterpreted variables, constants, predicates, and functions from the corresponding bit-level equivalents[1]:



When **Reach-CEGAR** is applied to the abstract transition relation it returns the 0-step counterexample

$$ACEX = (\hat{X} = \text{ZERO}) \wedge (\hat{Y} = \text{ZERO}) \wedge (\text{GT}(\hat{Y}, \hat{X}))$$

since it does not know the semantics of the abstract constant ZERO and the abstract predicate GT. However, upon concretization and bit-level feasibility checking, **DP-CEGAR** concludes that this counterexample is spurious and derives the following datapath lemma

$$\delta_1 = \neg\text{GT}(\text{ZERO}, \hat{X})$$

to rule it out. The second call to **Reach-CEGAR** returns a 1-step abstract counterexample which is also found to be infeasible and is refuted by the datapath lemma

$$\begin{aligned}\delta_2 = \neg[&(\hat{Y} = \hat{X}) \wedge (\hat{X}^+ = \text{INC}(\hat{X})) \wedge \\ &(\hat{Y}^+ = \text{INC}(\hat{Y})) \wedge (\text{GT}(\hat{Y}^+, \hat{X}^+))]\end{aligned}$$

This lemma is a constraint that relates the uninterpreted GT predicate and the uninterpreted INC function: in words, it states that applying INC to equal values

[1] Note that this abstraction is reversible; we just need to maintain the correspondence between the abstract entities and their bit-level counterparts.

cannot yield results in which one is greater than the other. The third, and final call, to **Reach-CEGAR** returns an empty trace after eliminating two CTIs and generating two abstract *single-literal* refinement clauses: $\neg \mathrm{GT}(\hat{X}, \hat{Y})$ and $(\hat{Y} = \hat{X})$. Thus, after eliminating 0- and 1-step counterexamples with two datapath lemmas, **DP-CEGAR** is able to prove the property in just one reachability iteration *regardless of the bit width of the state variables.*

7 The Averroes Algorithm

In this section we describe the **Averroes** program, a prototype implementation of **DP-CEGAR**. **Averroes**[2] is written in C++ and accepts design descriptions in a variety of formats including RTL Verilog. It calls **DP-Abstract** to create an initial abstraction of T, I, and P, similar to that described in [19], and passes it on to **Reach-CEGAR**, an IC3/PDR approximate reachability procedure to be described shortly. Abstract counterexample traces returned by **Reach-CEGAR** are bit blasted and checked for feasibility one transition at a time. Each infeasible transition in a counterexample triggers the generation of one or more datapath refinement lemmas using a simplified version of the minimal unsatisfiable subset (MUS) extractor in [20]. Feasibility checking is done using the bit vector (BV) theory in the Yices (version 1.0.35) SMT solver [21].

Fig. 4 highlights the major steps of the approximate reachability computation in **Reach-CEGAR** (lines 9–18 in Fig. 1). The formulas processed by **Reach-CEGAR** are all in the first-order logic of equality with uninterpreted functions (EUF) and all reasoning is done using the Yices SMT solver. Satisfying solutions returned from the SMT solver are converted to a conjunction of literals which take several forms:

- positive or negated bit-level variables
- positive or negated uninterpreted predicates
- equalities or disequalities between uninterpreted constants, terms, and functions

The procedure utilizes a queue Q of proof obligations each of which is a pair $(c(X), k)$ where $c(X)$ is a conjunction of literals (a cube) and k is a frame number. The following numbered list corresponds to the numbered boxes in Fig. 4:

1. At the start of major iteration k, frame k is overapproximated to P ($F_k = P$). The iteration then repeatedly checks for CTIs using the function 1-step which calls the SMT solver with the query: $F_k \wedge T \wedge \Delta \wedge \neg P^+$
2. A satisfying solution $s(X) \in F_k(X)$ to this query indicates a CTI that must now be checked for reachability from $I(X)$. Before proceeding with that check, however, the solution is "expanded" to remove irrelevant literals using a) cone of influence (COI) reduction, and b) finding MUSes, if any exist, of the formula[3] $s \wedge P \wedge T \wedge \Delta \wedge P^+$[22]. The enlarged cube $\widehat{s}$ is now added to

[2] The Averroes tool and some hand-crafted examples are available at `http://web.eecs.umich.edu/~suholee/AVERROES.html`

[3] PDR does this at the bit level using 3-valued simulation. In our case, this formula may be satisfiable and not yield an expansion of the cube!

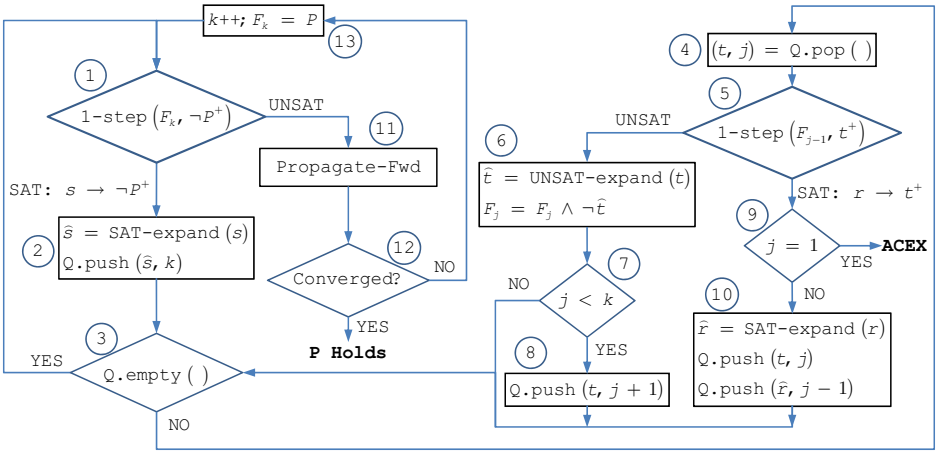


Fig. 4. Implementation of **Reach-CEGAR** in the Averroes Verifier. **Reach-CEGAR** performs approximate reachability computation on an EUF abstraction of the bit-level transition relation.

the Q as a proof obligation in frame k, meaning "can $\widehat{s}$ be eliminated from F_k by showing that it is unreachable from I along paths whose length is at least k?"

3. An empty queue signifies that the current CTI has been successfully eliminated and the algorithm proceeds to check for the existence of another CTI from the current frame.
4. The reachability computation starts here by retrieving a proof obligation (t, j) from the queue.
5. The 1-step function checks the formula $F_{j-1} \wedge T \wedge \Delta \wedge t^+ \wedge P^+$ to determine if t can be reached in one step from frame $j-1$.
6. If t is not reachable from frame $j-1$, it is enlarged to $\widehat{t}$ by extracting one or more MUSes from the UNSAT formula in step 5. The negation of $\widehat{t}$ is now added as a refinement clause to frame j (which means that all frames $1 \leq i \leq j$ are tightened as a result of the unreachability of t in frame j).
7. The processing of cube t terminates if we reach the last frontier k.
8. Otherwise, t is added as a proof obligation in frame $j+1$. This step is optional but, as pointed out in [2], it helps to improve performance and to find counterexample traces that are longer than $k+1$.
9. If the current proof obligation is $(t, 1)$ and t is found to be reachable from frame 0, then we have found an abstract counterexample trace ACEX and the procedure terminates.
10. If t in frame j is found to be reachable (in one step) from frame $j-1$, the satisfying solution r to the query in step 5 is enlarged similarly to how s was enlarged in step 2. Specifically, irrelevant literals are removed from r by COI reduction and MUS extraction, if any exist, from $r \wedge P \wedge T \wedge \Delta \wedge \neg t^+ \wedge P^+$. Processing continues by re-inserting (t, j) into the queue and adding $(\widehat{r}, j-1)$ as a new proof obligation.

Table 2. Statistics of the Large Industrial Benchmarks

Benchmark	Regs	FFs	State Bits	%Regs	%Reg Bits	AIG Size
mult_hold_1	6	2	258	75	99	24452
mult_hold_2	6	2	514	75	100	98052
mult_hold_3	6	2	1026	75	100	392708
mult_hold_4	6	10	266	38	96	24638
mult_viol_1	7	2	268	78	99	25008
mult_viol_2	7	2	524	78	100	99119
mult_viol_3	7	2	1036	78	100	394797
mult_viol_4	7	10	279	41	96	25193
mult_viol_5	7	10	279	41	96	25193
mult_viol_6	7	10	279	41	96	25190
mult_viol_7	7	10	279	41	96	25190
fifo_hold_2	28	10	474	74	98	6848
fifo_hold_3	44	10	866	81	99	17968
fifo_hold_4	76	10	1642	88	99	53904
M0+_hold	56	26	1306	68	98	41630

11. When all CTIs from the current frontier k have been eliminated, refinement clauses from earlier iterations are checked to see if they can be moved forward to tighten later frontiers. A refinement clause $\omega \in F_j$, $1 \leq j \leq k$ that causes the query $F_j \wedge T \wedge \Delta \wedge \neg\omega^+ \wedge P^+$ to be UNSAT indicates that cube $\neg\omega$ in frame $j+1$ is unreachable in one step from frame j and can thus be eliminated from frame $j+1$. This is accomplished by propagating clause ω forward: $F_{j+1} = F_{j+1} \wedge \omega$.
12. The procedure terminates proving that P holds if two successive frames become equal, i.e., if $F_j = F_{j+1}$ for some $1 \leqslant j \leqslant k$. This check is equivalent to finding the clause set associated with frame j has become empty.
13. Otherwise, a new frame is created and initialized to P and the procedure continues to check for CTIs corresponding to longer counterexample traces.

8 Experimental Evaluation

Anecdotally, abstracting a design's datapath is commonly believed to yield scalable verification of its control logic. However, unlike verification at the bit-level which enjoys a large corpus of benchmarks and published results, there is little documentation in the open literature of the effectiveness of datapath abstraction on a diverse set of word-level benchmarks. The dearth of publicly-available RTL benchmarks that preserve the word-level semantics of a design was one of the main challenges we faced when evaluating the effectiveness of Averroes. Realizing that reporting on hand-crafted synthetic benchmarks would not be convincing, we opted instead to evaluate performance on a set of 139 industrial Verilog benchmarks that we obtained under non-disclosure agreements.[4]

[4] Companies understandably want to protect the IP of their, or their customers', RTL designs. However, to spur further research in this space, it is important to find a way to make such RTL designs publicly available without compromising their owners' IP rights.

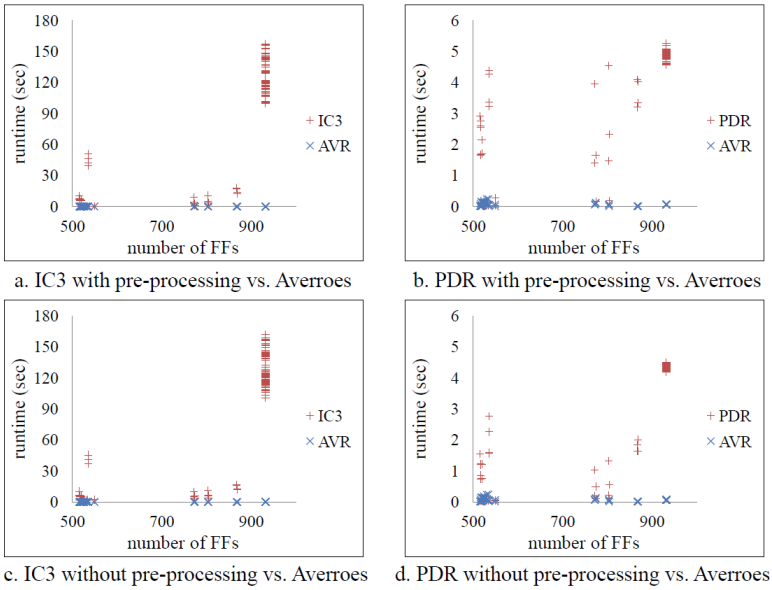


Fig. 5. Verification Results of the Generic Industrial Benchmarks

Of these, 124 were medium-sized "generic" benchmarks that were used for initial calibration. Their code sizes ranged between 298 and 805 lines; in terms of flip-flops, the smallest had 514 and the largest had 931. The remaining 15 benchmarks included 11 large multipliers, 3 FIFO designs, and the ARM Cortex-M0+ core [23]. The code sizes for these ranged from 116 to 10,226 lines. Table 2 lists additional statistics including the number of multi-bit registers (Regs), the number of single-bit flip-flops (FFs), the total number of state bits (FFs + the number of bits in the registers), the percentage of registers and register bits in the benchmark, and the number of AND nodes in the AIG representation [24] of its synthesized bit-level netlist. The multiplier benchmarks involved checking the sequential equivalence before and after clock gating optimizations; in four of these the property holds, and in the remaining seven it fails. The FIFO benchmarks check a "read-after-write" property for different FIFO depths. Finally, the M0+ experiment involved checking self-equivalence under partial initialization (i.e., when only a subset of the state bits are initialized on reset); this is sometimes referred to as self-equivalence with don't-cares or SEQX. In all cases, the verification involved an *unbounded check* to determine if the given safety property holds, on all, or is violated, by some, reachable states.

We compared the performance of Averroes to that of IC3 and PDR *with and without pre-processing*. In their default modes, IC3 and PDR simplify the input design before they start the approximate reachability loop: IC3 applies AIG sweeping [25]; PDR invokes the ABC dprove command [26]. Such pre-processing can greatly reduce the size of the input circuit which helps with the subsequent reachability computation. All experiments were run on a 3.2GHz Xeon desktop computer with a 16 GB memory. A time-out of 10,000 seconds was used for

Table 3. Verification Results of the Large Industrial Benchmarks

Benchmark	Runtime, sec			Frames			CTI Checks			Refinement Clauses			Solver Calls		
	IC3	PDR	AVR	IC3	PDR	AVR	IC3	PDR	AVR	IC3	PDR	AVR	IC3	PDR	AVR
mult_hold_1	T.O.	T.O.	0.02	1	3	1	41595	3	4	256	131720	3	776723	6331714	22
mult_hold_2	T.O.	T.O.	0.02	1	3	1	9559	3	4	512	13008	3	159978	1032718	22
mult_hold_3	ERR	T.O.	0.02	N/A	3	1	N/A	3	4	N/A	5986	3	N/A	546718	22
mult_hold_4	ERR	T.O.	0.04	N/A	2	2	N/A	2	8	N/A	1	8	N/A	10	56
mult_viol_1	ERR	116.15	0.05	N/A	2	1	N/A	2	5	N/A	3	3	N/A	21	15
mult_viol_2	ERR	256.32	0.18	N/A	2	1	N/A	2	5	N/A	2	3	N/A	16	15
mult_viol_3	ERR	1483.92	0.75	N/A	2	1	N/A	2	5	N/A	2	3	N/A	16	15
mult_viol_4	ERR	T.O.	0.54	N/A	2	7	N/A	2	30	N/A	262365	29	N/A	8177493	335
mult_viol_5	ERR	T.O.	11.88	N/A	2	22	N/A	2	47	N/A	252987	69	N/A	7961852	3040
mult_viol_6	ERR	T.O.	299.23	N/A	2	115	N/A	2	120	N/A	247035	275	N/A	8102702	55251
mult_viol_7	ERR	T.O.	1884.52	N/A	2	451	N/A	2	536	N/A	239809	754	N/A	7826919	425892
fifo_hold_2	13.79	14.87	1.35	9	12	8	1599	12	115	1691	4030	115	30060	94230	1574
fifo_hold_3	249.94	201.58	12.88	16	20	16	6455	20	355	6759	17772	317	306131	612147	9711
fifo_hold_4	5322.7	746.94	264.85	33	31	28	29898	36984	1804	25700	24609	1403	2602365	1611008	103590
M0+_hold	ERR	T.O.	917.76	N/A	8	17	N/A	5315	1154	N/A	3783	911	N/A	75898	45755

a. IC3 and PDR were run *with* pre-processing.

mult_hold_1	ERR	T.O.	0.02	N/A	2	1	N/A	134	4	N/A	217	3	N/A	1307	22
mult_hold_2	ERR	T.O.	0.02	N/A	2	1	N/A	257	4	N/A	512	3	N/A	3147	22
mult_hold_3	ERR	T.O.	0.02	N/A	2	1	N/A	521	4	N/A	612	3	N/A	2915	22
mult_hold_4	ERR	T.O.	0.04	N/A	2	2	N/A	189	8	N/A	250	8	N/A	1611	56
mult_viol_1	ERR	0.62	0.05	N/A	2	1	N/A	256	5	N/A	383	3	N/A	1439	15
mult_viol_2	ERR	10.86	0.18	N/A	2	1	N/A	386	5	N/A	532	3	N/A	2129	15
mult_viol_3	ERR	219.93	0.75	N/A	2	1	N/A	537	5	N/A	798	3	N/A	3396	15
mult_viol_4	ERR	T.O.	0.54	N/A	2	7	N/A	181	30	N/A	284	29	N/A	1821	335
mult_viol_5	ERR	T.O.	11.88	N/A	2	22	N/A	191	47	N/A	252	69	N/A	1596	3040
mult_viol_6	ERR	T.O.	299.23	N/A	2	115	N/A	177	120	N/A	273	275	N/A	1751	55251
mult_viol_7	ERR	T.O.	1884.52	N/A	2	451	N/A	179	536	N/A	260	754	N/A	1660	425892
fifo_hold_2	19.67	21.12	1.35	9	16	8	1944	7191	115	2090	5544	115	38000	192592	1574
fifo_hold_3	259.1	1252.89	12.88	17	29	16	6468	28402	355	8000	39937	317	273378	2525164	9711
fifo_hold_4	4715.67	10454.09	264.85	32	32	28	29359	98122	1804	42802	153114	1403	2453754	7618089	103590
M0+_hold	ERR	T.O.	917.76	N/A	8	17	N/A	5532	1154	N/A	4363	911	N/A	77808	45755

b. IC3 and PDR were run *without* pre-processing.

each verification run. Each of the 124 generic benchmarks was provided with a single specified safety property and were meant to calibrate the performance of Averroes against that of IC3 and PDR. Fig. 5 compares the runtime of Averroes against that of IC3 and PDR as a function of the number of flip-flops in these benchmarks. In almost all cases, Averroes is the fastest verifier and, unlike IC3 and PDR, its performance is largely independent of the number of flip-flops. This validates the hoped-for benefit of datapath abstraction. Oddly, the performance of IC3 and PDR with pre-processing was worse than without! This seems to be due to the fact that there was not much structural reduction due to pre-processing causing pre-processing overhead to outweigh its benefit.

Table 3 shows the results of our experiments on the 15 large benchmarks; time-outs are indicated as T.O., and ERR indicates that IC3 reported an error and was unable to process the benchmark[5]. As with the generic benchmarks, Averroes was the fastest verifier across this entire set of 15 benchmarks. IC3 and PDR had particular difficulty with the multiplier benchmarks. IC3 either

[5] We traced this error to an incorrect time-out exit that occurred before the specified time-out value!

Table 4. Runtimes (in Seconds) of FIFO on Various Depths

depth	State Bits	IC3	PDR	AVR	AVR_MA	AVR_MAA
2^2	474	13.79	14.87	1.35	1.8	8.28
2^3	866	249.94	201.58	12.88	10.92	20.57
2^4	1642	5322.7	746.94	264.85	120.51	21.93
2^5	3186	T.O.	T.O.	T.O.	2538.49	23.31
2^6	6266	T.O.	T.O.	T.O.	T.O.	19.17
2^7	12418	T.O.	T.O.	T.O.	T.O.	24.02
2^8	24714	T.O.	T.O.	T.O.	T.O.	20.58
2^9	49298	T.O.	T.O.	T.O.	T.O.	21.15
2^{10}	98458	T.O.	T.O.	T.O.	T.O.	27.9
2^{11}	196770	T.O.	T.O.	T.O.	T.O.	29.02
2^{12}	393386	T.O.	T.O.	T.O.	T.O.	23.57
2^{13}	786610	T.O.	T.O.	T.O.	T.O.	33.79
2^{14}	1573050	T.O.	T.O.	T.O.	T.O.	46.85
2^{15}	3145922	T.O.	T.O.	T.O.	T.O.	57.04
2^{16}	6291658	T.O.	T.O.	T.O.	T.O.	79.19

timed out or had an error exit. PDR timed out on eight out of the eleven cases. A possible explanation for this behavior is that the combinational logic in the multiplier benchmarks, which involves wide (32- to 256-bit) datapath signals, led to bit-level formulas that were too large and complicated for IC3 and PDR to handle effectively. An examination of the runtime per solver call for mult_hold_2 and fifo_hold_2 confirms this. These two benchmarks have similar sizes in terms of state bits, but mult_hold_2 leads to an AIG whose size is more than 14 times larger than that of fifo_hold_2. PDR made 3,147 solver calls in 10,000 seconds for the multiplier benchmark, averaging about 3.18 seconds per call. The corresponding data for the FIFO benchmark were 192,592 calls in 21.12 seconds, an average of 110 micro seconds per call which is more than four orders of magnitude faster. Additionally, the peculiarly low number of solver calls for mult_hold_4 in Table 3-a seemed too suspicious; on closer examination we found out that the first 10 calls were very quick, but the solver timed out on the 11th. This again suggests a difficult formula that thwarted the solver.

In contrast to PDR's performance, Averroes was able to solve all 11 cases, most in fractions of a second. Other performance metrics, such as the number of net refinement clauses and number of solver calls, are significantly less than those for PDR suggesting that datapath abstraction was effective in reducing the "size" of the reachability search space and that the abstract refinement clauses were much stronger than their bit-level counterparts in pruning the space. The cases requiring longer runtimes, about 30 minutes for mult_viol_7, were due to extremely long counterexample traces that require the traversal of many frames which, in turn, translate into many solver calls. For instance, the counterexample trace for mult_viol_7 consisted of 1002 transitions which required the traversal of 451 frames and making 425,892 solver calls.

The three FIFO benchmarks involved checking a "read-after-write" property for the FIFO entries. The FIFO depths (number of entries) ranged from 4 (for fifo_hold_2) to 16 (for fifo_hold_4) and each benchmark had two FIFOs whose width is 32 bits and two FIFOs whose width is 16 bits. Again, Averroes outperforms both IC3 and PDR on these benchmarks, on average being about 20 times

faster. This is another indication of the effectiveness of datapath abstraction. To dramatize this, we carried out a parametric experiment by increasing the width of the FIFO entries. As expected, the runtime of Averroes did not change, whereas the runtimes of IC3 and PDR exhibited exponential behavior. However, all three verifiers exhibited exponential behavior as FIFO depths were increased! Upon reflection, this too should have been expected since FIFOs are basically "small" memories and datapath abstraction alone is insufficient to handle them. While memory abstraction is beyond the scope of this paper, we present in Table 4 data showing the performance of Averroes when it is augmented with the structural memory abstraction described in [27]. This type of abstraction can be layered on top of any model checking verifier and can certainly be added to IC3 and PDR. But as the column labeled AVR_MA in this table shows, memory abstraction scales the performance of Averroes only to a FIFO depth of 32. Further scaling requires integrating memory abstraction with datapath abstraction of the memory addresses. This is shown in column AVR_MAA. Clearly the combination of memory abstraction and memory *address* abstraction yields a verification flow that is largely independent of memory size. The linear increase in the runtime of Averroes is due to the bit-level feasibility checks on wider memory addresses as memory size increases.

The last benchmark in Table 3 is the SEQX instance of the Cortex-M0+. The verification goal here was to show that the M0+ core is self-equivalent when 41 of its state bits are left uninitialized on reset (i.e., their initial value is X or don't care). Specifically, SEQX holds when none of these don't-care values propagate to observable outputs. Effectively, the verifier is establishing the state equivalence of 2^{41} possible initial states. We should note that SEQX becomes quite trivial if the number of uninitialized state bits is small. In fact, bit-level verifiers can quickly solve such problems using structural hashing techniques. However, as the number of uninitialized state bits increases, structural hashing ceases to be effective (not very many equivalent signals to merge) and bit-level verifiers fail. This is clearly shown in Table 3: neither IC3 nor PDR was able to prove self-equivalence; Averroes required about 15 minutes to show that SEQX holds for M0+.

9 Conclusion

Many complex computational problems can be scalably handled by judicious elimination of irrelevant details. The Averroes verifier described in this paper integrates two orthogonal abstractions that, together, yield a scalable system for the verification of control-centric properties in hardware designs containing wide datapaths and complex control logic. To our knowledge, this is the first public demonstration of an automated verification flow for unbounded model checking of safety properties in industrial benchmarks. To be sure, there are many other abstraction approaches that have been shown to work well in different domains. However, for the particular control logic bugs targeted by our approach, datapath abstraction seems to provide the most scalability. Specifically, our preliminary evidence strongly suggests that scalability is quite achievable by augmenting bit-level reasoning with RTL word-level abstractions.

Acknowledgment. This work was funded in part by ARM, Ltd. Additional support was provided by the Qatar Computing Research Institute.

References

1. Bradley, A.R.: Sat-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011)
2. Een, N., Mishchenko, A., Brayton, R.: Efficient implementation of property directed reachability. In: FMCAD 2011, pp. 125–134. IEEE (2011)
3. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a sat-solver. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 108–125. Springer, Heidelberg (2000)
4. Bjesse, P., Claessen, K.: Sat-based verification without state space traversal. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 372–389. Springer, Heidelberg (2000)
5. Bradley, A.R., Manna, Z.: Checking safety by inductive generalization of counterexamples to induction. In: FMCAD 2007, pp. 173–180. IEEE (2007)
6. Kurshan, R.P.: Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach. Princeton University Press (1994)
7. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
8. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
9. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
10. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: A new symbolic model checker. STTT 2(4), 410–425 (2000)
11. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: Cleaveland, W.R. (ed.) TACAS/ETA 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
12. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1), 7–34 (2001)
13. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 157–171. Springer, Heidelberg (2012)
14. Cimatti, A., Griggio, A.: Software model checking via ic3. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 277–293. Springer, Heidelberg (2012)
15. Vizel, Y., Grumberg, O., Shoham, S.: Lazy abstraction and sat-based reachability in hardware model checking. In: FMCAD 2012, pp. 173–181 (2012)
16. Hojati, R., Brayton, R.K.: Automatic datapath abstraction in hardware systems. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 98–113. Springer, Heidelberg (1995)
17. Macii, E., Plessier, B., Somenzi, F.: Formal verification of digital systems by automatic reduction of data paths. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 16(10), 1136–1156 (1997)
18. Paruthi, V., Mansouri, N., Vemuri, R.: Automatic data path abstraction for verification of large scale designs. In: ICCD 1998, pp. 192–194. IEEE (1998)
19. Andraus, Z.S., Sakallah, K.A.: Automatic abstraction and verification of verilog models. In: DAC 2004, pp. 218–223 (2004)
20. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning 40(1), 1–33 (2008)
21. Dutertre, B., De Moura, L.: The Yices SMT solver (2006) Tool paper at, `http://yices.csl.sri.com/tool-paper.pdf`

22. Chockler, H., Ivrii, A., Matsliah, A., Moran, S., Nevo, Z.: Incremental formal verification of hardware. In: FMCAD 2011, pp. 135–143 (2011)
23. ARM Cortex-M0+ Processor, http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php
24. Biere, A.: The aiger and-inverter graph (aig) format (2007), fmv.jku.at/aiger
25. Een, N.: Cut sweeping. Cadence Design Systems, Tech. Rep. (2007)
26. Mishchenko, A., Case, M., Brayton, R., Jang, S.: Scalable and scalably-verifiable sequential synthesis. In: ICCAD 2008, pp. 234–241. IEEE (2008)
27. Bjesse, P.: Word-level sequential memory abstraction for model checking. In: FMCAD 2008, pp. 1–9 (November 2008)