TOBIAS ACHTERBERG, RAIK BRINKMANN, MARKUS WEDLER

Property Checking with Constraint Integer Programming

ZIB-Report 07-37 (November 2007)

Property Checking with Constraint Integer Programming

Tobias Achterberg[*], Raik Brinkmann[†], Markus Wedler[‡]

November 11, 2007

Abstract

We address the property checking problem for SoC design verification at the register transfer level (RTL) by integrating techniques from integer programming, constraint programming, and SAT solving. Specialized domain propagation and preprocessing algorithms for individual RTL operations extend a general constraint integer programming framework. Conflict clauses are learned by analyzing infeasible LPs and deductions, and by employing reverse propagation. Experimental results show that our approach outperforms SAT techniques for proving the validity of properties on circuits containing arithmetics.

Keywords: SoC design verification, property checking, constraint programming, integer programming

1 Introduction

Formal verification techniques have to cope with the ever growing complexity of new System-on-Chip (SoC) designs. One of the milestones in this chase is the establishment of bounded model checking (BMC) [6] in industrial design flows. The reduction of the property checking problem to a satisfiability (SAT) problem facilitates formal verification of industrial circuit designs far beyond the scope of classical model checking techniques. However, it is well known that SAT solvers have problems when dealing with instances derived from the verification of arithmetic circuits. Although SAT based property checking can often be applied successfully to the control part of a design, it typically fails on data paths with large arithmetic blocks. Verification engineers typically resort to incomplete techniques such as constraint random simulation or bit-slicing in order to find errors in arithmetic units. However, these methods cannot *prove* correctness of a logic design. Especially, errors in corner cases are likely to be missed.

[*]Konrad-Zuse-Zentrum für Informationstechnik Berlin, achterberg@zib.de
[†]OneSpin Solutions GmbH, raik.brinkmann@onespin-solutions.com
[‡]University Kaiserslautern, markus.wedler@eit.uni-kl.de

Therefore reasoning at higher levels of abstraction became an attractive field of research in the recent past. A survey on various abstraction techniques that can be applied at the RTL is given in [15]. These techniques are of great value when the validity of the property at hand does not rely upon the exact functionality of the datapath under verification. Unfortunately, this does not apply to the verification problems considered in this work. Therefore the algorithms presented in this paper can be considered as orthogonal to the abstraction techniques summarized in [15].

Given a word-level representation of a decision problem at hand solvers for integer programming (IP) [8, 11, 26] or constraint programming (CP) [25] have been studied as promising candidate decision procedures. However, current IP and CP solvers do not learn conflict clauses during the search like SAT solvers do. They usually perform poorly on the control part of a design. A combination of word level and Boolean solvers has to be developed. Two promising ways of integrating IP and SAT have been proposed in [9] and [5]. The first one uses pseudo-Boolean constraints (PBCs) as clauses in a DPLL-style solver, and the second one uses linear equations as propositions. However, PBC solvers perform the reasoning at the bit level and the benefit of the stronger search space pruning due to learned PBCs usually does not justify the overhead for handling these more complex constraints. On the other hand, using IP techniques at the leaves of a decision tree without learning from infeasibilities sacrifices pruning potential in the logic part of the circuit.

More recently, SAT-modulo-theory (SMT) solvers have gained significant attention. In this category we consider two approaches called DPLL(T) [12] and HDPLL [21, 22] as being most related to our work. The techniques presented in these references are based on integration of different theories into a unified DPLL-style decision procedure. DPLL(T) combines the theory of Boolean logic with the theory of uninterpreted functions with equality (EUF). In fact, there is no mechanism for learning across theories in DPLL(T). It can handle only comparisons with equality, which makes it currently unsuitable for RT level property checking. On the other hand, HDPLL combines the DPLL algorithm with techniques from CP and IP, namely domain propagation and Fourier-Motzkin elimination.

This paper presents a novel integration of techniques from SAT, IP and CP into a unified decision procedure that tackles the property checking problem at the RT level. For each RT operation, a specific domain propagation algorithm is applied, using both, bit- and word-level representations. In addition, we provide *reverse* propagation algorithms to support conflict analysis at the RT level.

In HDPLL the Fourier-Motzkin elimination step is only used as a last resort to check the feasibility on the data path after all binary variables have been fixed. In contrast, we solve a linear programming (LP) relaxation at *every* subproblem in the search tree. Using the dual simplex algorithm the LPs can be resolved efficiently after applying changes to the variables' bounds. Due to the "global" view of the LP, infeasibilities of a subproblem can be detected much higher in the search tree than with constraint programming or SAT techniques alone. By using dual information one can also derive conflict clauses out of infeasible LP relaxations [1]. In addition, a feasible LP solution can either yield a counter-example for the property, or can be used to control the next branching decision, thereby guiding the search.

We demonstrate the effectiveness of the proposed approach on industrial benchmarks obtained from verification projects conducted using OneSpin 360 [19].

2 Concepts

Bounded interval property checking based on an iterative circuit model has become an attractive alternative to classical simulation based verification techniques for block-level verification of RTL designs. The property checking problems encountered in interval property checking can be formulated as constraint satisfaction problem (CSP), which can be seen as a generalization of SAT to arbitrary constraints and variables with arbitrary domains, see [4]. We model the property checking CSP with variables $\varrho \in \{0, \ldots, 2^{w_\varrho - 1}\}$ of width w_ϱ and constraints $r^i = C_i(x^i, y^i, z^i)$, which resemble circuit operations with up to three input bit vectors x^i, y^i, z^i, and an output bit vector r^i. For each bit vector variable ϱ, we introduce single bit variables ϱ_b, $b = 0, \ldots, w_\varrho - 1$, with $\varrho_b \in \{0, 1\}$, for which linking constraints

$$\varrho = \sum_{b=0}^{w_\varrho - 1} 2^b \varrho_b \tag{1}$$

define their correlation. In addition, we consider the following circuit operations: ADD, AND, CONCAT, EQ, ITE, LT, MINUS, MULT, NOT, OR, READ, SHL, SHR, SIGNEXT, SLICE, SUB, UAND, UOR, UXOR, WRITE, XOR, ZEROEXT with the semantics as defined in [7].

We solve the property checking CSP with a branching algorithm, which successively divides the problem instance into subproblems by splitting the domains of the bit vector variables ϱ into two disjunctive parts: either by fixing a certain bit of a bit vector to $\varrho_b = 0$ and $\varrho_b = 1$, respectively, or by introducing local lower bounds $\varrho \geq l$ and local upper bounds $\varrho \leq u$ on the individual bit vectors. At each node in the resulting search tree, different methods from constraint programming (CP), integer programming (IP) and SAT solving are applied to tighten the local subproblem and to prune the search tree. The algorithm terminates if a solution has been found that satisfies all constraints, or if all nodes of the search tree have been pruned, thereby proving the infeasibility of the problem instance.

2.1 CP Techniques

The main engine of constraint programming is *domain propagation*: at each node in the search tree, logical deductions on the current set of the variables' domains are applied in order to exclude further values from the variables' local domains. Such a domain reduction can then trigger additional reductions. The propagation is stopped if no more domain reductions can be found.

For the bit linking constraints (1) and for each type of circuit operation we implemented a specific domain propagation algorithm that exploits the special structure of the constraint class. In addition to considering the current domains of the bit vectors ϱ and the bit variables ϱ_b, we exploit knowledge about the global equality or inequality of bit vectors or bits, which is obtained in the preprocessing stage of the algorithm. For example, if we know already that for certain bits the input vectors x and y in an equality constraint $r = \text{EQ}(x, y)$ are equal, i.e., $x_b = y_b$ for a few bits b, the value of the resultant r can already be decided after the remaining input bits have been fixed.

Some of the domain propagation algorithms are very complex. For example, the domain propagation of the MULT constraint uses term algebra techniques to recognize certain deductions inside its internal representation of a partial product and overflow addition network. Others, like the algorithms for SHL, SLICE, READ, and WRITE, involve reasoning that mixes bit- and word-level information.

2.2 IP Techniques

The core of current state-of-the-art IP solvers is the LP relaxation of the problem instance which is solved at every node in the search tree. Typically, an integer program contains an objective function. The LP relaxation yields a dual objective bound which can be used to prune the search tree by means of optimality considerations. Additionally, the LP relaxation can be strengthened by adding so-called *cutting planes* which exploit the integrality restrictions on the variables.

Because property checking is a pure feasibility problem, there is no natural objective function. However, the LP relaxation usually detects the infeasibility of the local subproblem much earlier than domain propagation. This is due to the fact that the LP has a "global view" taking all constraints except the integrality conditions into consideration at the same time, while domain propagation is applied successively on each individual constraint.

Table 1 shows the linearizations of the circuit operation constraints that are used in addition to the bit linking constraints (1) to construct the LP relaxation of the problem instance. Very large coefficients like 2^{w_r} in the ADD linearization can lead to numerical difficulties in the LP relaxation. Therefore, we split the bit vector variables into words of $W = 16$ bits and apply the linearization to the individual words. The linkage between the words is established in a proper fashion. For example, the overflow bit of a word in an addition is added to the right hand side of the next word's linearization. The relaxation of the MULT constraint involves additional variables y_n and r_n which are "nibbles" of y and r with $L = \frac{W}{2}$ bits.

The MINUS and SUB operations can be replaced by an equivalent ADD operation as shown in the table. SHR is replaced by the more general SLICE operator. The operations CONCAT, NOT, SIGNEXT, and ZEROEXT do not need an LP relaxation: their resultant bits can be aggregated with the corresponding operand bits such that the constraints can be deleted in the preprocessing stage of the algorithm. No linearization is generated for the SHL, SLICE, READ, and WRITE

Operation	Linearization	
r = AND (x,y)	$r_b \le x_b$, $r_b \le y_b$, $r_b \ge x_b + y_b - 1$	
r = OR (x,y)	$r_b \ge x_b$, $r_b \ge y_b$, $r_b \le x_b + y_b$	
r = XOR (x,y)	$x_b - y_b - r_b \le 0$, $-x_b + y_b - r_b \le 0$,	
	$-x_b - y_b + r_b \le 0$, $x_b + y_b + r_b \le 2$	
r = UAND (x)	$r \le x_b$, $r \ge \sum_{b=0}^{w_x-1} x_b - w_x + 1$	
r = UOR (x)	$r \ge x_b$, $r \le \sum_{b=0}^{w_x-1} x_b$	
r = UXOR (x)	$r + \sum_{b=0}^{w_x-1} x_b = 2s$,	$s \in \mathbb{Z}_{\ge 0}$
r = EQ (x,y)	$x - y = s - t$,	$s, t \in \mathbb{Z}_{\ge 0}$,
	$p \le s$, $s \le p(u_x - l_y)$,	$p \in \{0, 1\}$,
	$q \le t$, $t \le q(u_y - l_x)$,	$q \in \{0, 1\}$,
	$p + q + r = 1$	
r = LT (x,y)	$x - y = s - t$,	$s, t \in \mathbb{Z}_{\ge 0}$,
	$p \le s$, $s \le p(u_x - l_y)$,	$p \in \{0, 1\}$,
	$r \le t$, $t \le r(u_y - l_x)$,	
	$p + r \le 1$	
r = ITE (x,y,z)	$r - y \le (u_z - l_y)(1 - x)$	
	$r - y \ge (l_z - u_y)(1 - x)$	
	$r - z \le (u_y - l_z)\, x$	
	$r - z \ge (l_y - u_z)\, x$	
r = ADD (x,y)	$r + 2^{w_r} o = x + y$,	$o \in \{0, 1\}$
r = MULT (x,y)	$v_{bn} \le u_{y_n} x_b$, $v_{bn} \le y_n$,	$v_{bn} \in \mathbb{Z}_{\ge 0}$
	$v_{bn} \ge y_n - u_{y_n}(1 - x_b)$	
	$o_n + \sum_{i+j=n} \sum_{l=0}^{L-1} 2^l v_{iL+l,j}$	
	$\quad = 2^L o_{n+1} + r_n$,	$o_n \in \mathbb{Z}_{\ge 0}$
r = MINUS (x)	replaced by $0 = \text{ADD}(x, r)$	
r = SUB (x,y)	replaced by $x = \text{ADD}(y, r)$	
r = SHR (x,y)	replaced by $r = \text{SLICE}(x, y)$	
r = CONCAT (x,y)	removed in preprocessing	
r = NOT (x)		
r = SIGNEXT (x)		
r = ZEROEXT (x)		
r = SHL (x,y)	no linearization	
r = SLICE (x,y)		
r = READ (x,y)		
r = WRITE (x,y,z)		

Table 1. LP relaxation of circuit operations. l_ϱ and u_ϱ are the lower and upper bounds of a bit vector variable ϱ.

constraints. Their linearizations are very complex and would dramatically increase the size of the LP relaxation, thereby reducing the solvability of the LPs. For example, a straight-forward linearization of the SHL constraint on a 64-bit input vector x that uses internal ITE-blocks for the potential values of the shifting operand y already requires 30944 inequalities and 20929 auxiliary variables.

In addition to the infeasibility detection capabilities, the LP relaxation can be used to tighten the variables' domains, to find feasible solutions, and to select a branching variable, i.e., a variable whose domain is split into two parts in order to create the two subproblems of the current node. Domain reduction can be achieved by applying the so-called *reduced cost strengthening* mechanism [18] which uses the dual solution as a proof that a variable cannot exceed a certain value. Usually, only a very few variables have a fractional value in the solutions of the LPs. If all values are integral and if the solution satisfies the constraints that were not linearized, the LP solution is a feasible solution of the property checking problem. On the other hand, variables with fractional values are promising candidates to be used for branching.

2.3 SAT Techniques

One of the key ingredients in modern SAT solvers is *conflict analysis* [16]: infeasible subproblems are analyzed in order to learn deduced clauses that can later be used to prune other nodes of the search tree. In addition, these conflict clauses enable the solver to perform so-called *non-chronological backtracking.* We briefly describe the generalization of conflict analysis to our integrated constraint integer programming (CIP) approach. Further details can be found in [1].

The following differences of SAT and CIP have to be considered. First, CIP involves non-binary variables. Like fixings of binary variables, changes in the lower and upper bounds of non-binary variables can be reason and consequence of a domain propagation. Therefore, we have to generalize the concept of the *conflict graph*, which represents the deductions that lead to the proof of infeasibility of the current subproblem. The nodes in the generalized conflict graph represent bound changes instead of fixings. Note that a single non-binary variable can now have multiple instead of a single appearance the conflict graph.

The analysis of the conflict graph consists of selecting a cut that separates the branching decisions from the conflict vertex. The vertices on the frontier of this cut yield the conflict clause. However, if one of these vertices is a bound change on a non-binary variable, the resulting constraint is no longer a clause (i.e., a disjunction of literals) but a disjunction of bound constraints.

The second difference is the fact that SAT solving only involves a single type of constraints, namely clauses, while CIP consists of different constraint classes, each of which is treated by a different domain propagation algorithm. Because conflict analysis needs to know the *reasons* for the deductions that lead to the conflict, we have to provide constraint specific *reverse propagation* algorithms. This leads to a much more involved bookkeeping system, in particular if bounds of non-binary variables are involved in the propagations.

While the first two differences between SAT and CIP are only technical obstacles for generalizing conflict analysis to CIP, there is also a principle issue. The conflict analysis of SAT relies on the fact that a single clause detected the conflict by observing that all its literals are fixed to 0. This *conflict detecting* *clause* yields the starting point from which the conflict graph can be constructed in a reverse fashion by applying *reverse propagation* on the literals that were fixed to 0. In contrast, most of the the infeasibilities in CIP subproblems are detected by the LP relaxation. In this case the LP as a whole is responsible for the conflict, which in particular includes *all* local bounds. Therefore, we have to identify a preferably small subset of the local bounds which—together with the constraints and global bounds—suffice to render the LP infeasible. This subset takes the role of the literals in the conflict detecting clause of SAT conflict analysis. After identifying this starting set of local bounds, the conflict graph can be generated and analyzed in a reverse fashion just like in SAT solving.

The problem of identifying a subset of the local bounds of *minimal cardinality* such that the LP stays infeasible if all other local bounds are removed is $\mathcal{NP}$-hard [1, 3]. Thus, we use the following heuristic to remove local bounds.

Consider the local LP relaxation

$$\max\{c^T x \mid Ax \le b,\ l \le x \le u,\ \tilde{l} \le x \le \tilde{u}\}$$

with l and u being the global bounds, and $\tilde{l}$ and $\tilde{u}$ being local bounds added by branching or domain propagation. First, we will only consider the case with $l = \tilde{l} = 0$. We further assume that each component of the global upper bounds u was tightened at most once to obtain the local upper bounds $\tilde{u} \le u$. Thus, the set of local bound changes consists of at most one bound change for each variable. The general case will be discussed below.

Suppose the local LP relaxation

$$\text{(P)}\quad \max\{c^T x \mid Ax \le b,\ 0 \le x \le \tilde{u}\}$$

is infeasible. Then its dual

$$\text{(D)}\quad \min\{b^T y + \tilde{u}^T r \mid A^T y + r \ge c,\ (y, r) \ge 0\}$$

has an unbounded ray, i.e., $(\bar{y}, \bar{r}) \ge 0$ with $A^T\bar{y} + \bar{r} \ge 0$ and $b^T\bar{y} + \tilde{u}^T\bar{r} < 0$. Note that the dual LP does not need to be feasible.

We can aggregate the rows and bounds of the primal LP with the non-negative weights $(\bar{y}, \bar{r})$ to get the following proof of infeasibility:

$$0 \ \le\ (\bar{y}^T A + \bar{r}^T)x \ \le\ \bar{y}^T b + \bar{r}^T \tilde{u} \ <\ 0. \qquad (2)$$

Now we try to relax the bounds as much as possible without loosing infeasibility. Note that the left hand side of (2) does not depend on $\tilde{u}$. Relaxing $\tilde{u}$ to some $\hat{u}$ with $\tilde{u} \le \hat{u} \le u$ increases the right hand side of (2), but as long as $\bar{y}^T b + \bar{r}^T \hat{u} < 0$, the relaxed LP

$$(\hat{P})\quad \min\{c^T x \mid Ax \le b,\ 0 \le x \le \hat{u}\}$$

is still infeasible with the same infeasibility proof $(\bar{y}, \bar{r})$. This leads to the following heuristic to produce a relaxed upper bound vector $\hat{u}$ with the corresponding LP still being infeasible, see [1].

Algorithm 2.1 Let $\max\{c^Tx \mid Ax \leq b,\ 0 \leq x \leq \tilde{u} \leq u\}$ be an infeasible LP with dual ray $(\bar{y}, \bar{r})$.

1. Set $\hat{u} := \tilde{u}$, and calculate the infeasibility measure $d := \bar{y}^Tb + \bar{r}^T\hat{u} < 0$.
2. Select a variable j with $\hat{u}_j < u_j$ and $d_j := d + \bar{r}_j(u_j - \tilde{u}_j) < 0$. If no such variable exists, stop.
3. Set $\hat{u}_j := u_j$, update $d := d_j$, and go to 2.

In the general case of multiple bound changes on a single variable, we have to process these bound changes step by step, always relaxing to the previously active bound. In the presence of non-zero lower bounds the reduced costs r may also be negative. In this case, we can split up the reduced cost values into $r = r^u - r^l$ with $r^u, r^l \geq 0$. It follows from the Farkas lemma that $r^u \cdot r^l = 0$. The infeasibility measure d of the dual ray has to be defined in Step 1 as $d := \bar{y}^Tb + (\bar{r}^u)^T\hat{u} + (\bar{r}^l)^T\hat{l}$. A local lower bound $\tilde{l}$ can be relaxed in the same way as an upper bound $\tilde{u}$, where u has to be replaced by l in the formulas of Steps 2 and 3.

As mentioned above, the analysis of an infeasible LP with Algorithm 2.1 yields a set of local bound changes that form the starting point of the conflict graph analysis. By applying reverse propagation, one or more conflict constraints can be extracted from the conflict graph. In our implementation, we use the *1-FUIP* [27] rule for generating conflict constraints. In addition to the *1-FUIP* conflict constraints we extract clauses from *reconvergence cuts* [27] in the conflict graph to support *non-chronological backtracking* [16].

2.4 Preprocessing

Before the actual branch-and-bound based search algorithm is applied we try to simplify the given problem instance by employing the following preprocessing techniques. The individual preprocessing algorithms are applied periodically until no more reductions can be found.

2.4.1 Probing

Probing denotes a very time-consuming preprocessing technique which evolved from the IP community [24]. It consists of successively fixing each binary variable to zero and one and evaluating the corresponding subproblems by domain propagation techniques. Let $x \in \{0, 1\}$ be a binary variable (e.g., a bit ϱ_b of a bit vector ϱ), and let $y \in \{l, \dots, u\}$ denote some other binary or non-binary variable. Let l_0 and u_0 be the lower and upper bounds of y that were deduced from $x = 0$. Let l_1 and u_1 be the corresponding bounds of y deduced from $x = 1$. Now, the following observations can be made:

- If one of the fixings of x leads to an infeasible subproblem, x can be permanently fixed to the opposite value and removed from the problem instance.

- If $l_0 = u_0$ and $l_1 = u_1$, y can be expressed as $y = l_0 + (l_1 - l_0)x$ and removed from the problem instance.
- $\bar{l} := \min\{l_0, l_1\}$ and $\bar{u} := \max\{u_0, u_1\}$ are valid global bounds of y.
- $x = 0 \rightarrow l_0 \leq y \leq u_0$ and $x = 1 \rightarrow l_1 \leq y \leq u_1$ are valid implications that can be stored in an implication graph and exploited during the solving process, e.g., in other preprocessing algorithms or in the branching variable selection.

2.4.2 Term Algebra Preprocessing

The signature defined by the circuit operations and strings of the constants 0 and 1 and the individual bits ϱ_b of bit vectors yields an extended term algebra. By regarding the valid equations of terms defined by the semantics of the circuit operations, we obtain a term replacement system.

Each constraint of the problem instance defines an additional equation between the resultant bit vector and the corresponding term including the operand bit vector as variables. By substituting the operands with their defining terms and by applying the semantical term replacement rules, we can extract equalities of terms or subterms and thereby equalities of the corresponding resultant bit vector or sub-vectors. Typical term replacement rules are, e.g., the commutativity, associativity and distributivity laws of addition and multiplication, and logical reasoning on AND, OR, XOR, and NOT constraints.

In the current version of our code, we only process ADD and MULT constraints in the term algebra preprocessing, and we do not exploit the distributivity law. However, due to the aggregations of bits ϱ_b found in other presolving algorithms, we implicitly include other constraints in the term replacement system, in particular CONCAT, NOT, SIGNEXT, ZEROEXT, and SLICE with a fixed offset operand. We expect further preprocessing improvements by incorporating other constraints into the term algebra and by exploiting replacement rules like the distributivity law which link different operations.

2.4.3 Irrelevance Detection

In order to prove the validity of a given property, often only a part of the circuit has to be considered. For example, if a property on an arithmetic logical unit (ALU) describes a certain aspect of the addition operation, the other operations of the ALU are irrelevant. Suppose that ITE constraints select the operation of the ALU by routing the output of the desired operation to the output register of the circuit. The calculated values of the other operations are linked to the discarded inputs of the ITE constraints and thereby do not contribute to the output of the circuit. These *irrelevant* constraints and the involved intermediate bit vectors have no influence on the validity of the property and can therefore be removed from the problem instance.

The detection of irrelevant parts of the circuit can also be applied to the local subproblems during the branch-and-bound search. In particular, irrelevant bit

Prop	Meth	register width 5	10	15	20	25	30	35	40
add_fail	SAT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	CIP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
sub_fail	SAT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	CIP	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1
muls	SAT	0.5	—	—	—	—	—	—	—
	CIP	0.0	0.0	0.0	0.1	0.1	0.1	0.2	0.3
neg_flag	SAT	0.1	100.0	—	—	—	—	—	—
	CIP	0.8	3.6	11.6	36.3	81.8	136.6	218.4	383.5
zero_flag	SAT	0.0	0.0	0.1	0.1	0.2	0.4	0.5	0.6
	CIP	2.3	0.6	1.6	4.0	6.2	10.7	15.6	379.7

Table 2. ALU invalid (top) and valid (bottom) properties.

vector variables need not to be considered as branching candidates. Disregarding locally irrelevant variables in the branching decision can be seen as replacement for the more indirect method of selecting the next branching variable under the literals involved in the recent conflict clauses, which is employed in state-of-the-art SAT solvers [13].

2.4.4 Domain Propagation

The constraint specific domain propagation algorithms, as described briefly in Section 2.1, are also applied during preprocessing. Whenever a new fixing, aggregation, domain reduction, or implication was found by one of the preprocessing algorithms, domain propagation is applied again to the affected constraints.

3 Experimental Results

In this section we examine the computational effectiveness of the described constraint integer programming techniques on industrial benchmarks obtained from verification projects conducted using OneSpin 360 [19]. All calculations were performed on a 3.8 GHz Pentium-4 workstation with 2 GB RAM. In all runs we used a time limit of 7200 seconds. The specific chip verification algorithms were incorporated into SCIP 0.90i, which is a framework for constraint integer programming [2]. The LP relaxations were solved using CPLEX 10.0.1 [14].

For reasons of comparison, we also solved the instances with SAT techniques on the gate level. Before the SAT solver is called, a preprocessing step is executed to simplify the instance at the gate level. We used MiniSat 2.0 [10] to solve the resulting SAT instances. We also tried MiniSat 1.14, Siege v4 [23] and zChaff 2004.11.15 [17], but MiniSat 2.0 turned out to perform best on most of the instances of our test set.

3.1 Test Set

We conducted experiments on the following sets of property checking instances:

Prop	Meth	register width							
		5	10	15	20	25	30	35	40
#1	SAT	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1
	CIP	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.4
#2	SAT	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1
	CIP	0.1	0.1	0.2	0.3	0.3	0.4	0.6	0.7
#3	SAT	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1
	CIP	0.1	0.1	0.2	0.2	0.2	0.5	0.4	0.7
#4	SAT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	CIP	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.3
#5	SAT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	CIP	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1
#8	SAT	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1
	CIP	0.0	0.1	0.1	0.1	0.2	0.2	0.2	0.2
#6	SAT	0.0	0.2	0.5	0.8	1.3	1.6	2.1	2.9
	CIP	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1
#7	SAT	0.0	0.6	1.3	1.9	2.2	1.7	22.5	16.4
	CIP	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.2
#9	SAT	0.1	8.9	21.3	764.0	5554.4	81.6	177.1	4087.1
	CIP	0.0	0.1	0.1	0.1	0.1	0.1	0.2	0.2
#10	SAT	0.0	0.2	0.5	0.8	1.3	1.6	2.1	2.9
	CIP	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1

Table 3. PipeAdder invalid (top) and valid (bottom) properties.

ALU: an arithmetical logical unit which performs ADD, SUB, SHL, SHR, and signed and unsigned MULT operations.

PipeAdder: adder with 4-stage pipeline to sum four values.

PipeMult: multiplier with a 4-stage pipeline.

Biquad: a DSP/IIR filter core obtained from [20] in different representations with some valid and invalid properties.

Multiplier: Gate level netlists for Booth and non-Booth encoded architectures of signed and unsigned multipliers, for which the correctness has to be proven. These instances are rather equivalence checking instances than property checking problems.

All test sets except the *Multiplier* set involve valid and invalid properties. The width of the input bit vectors in the *ALU*, *PipeAdder* and *PipeMult* sets range from 5 to 40 bits. For the *Multiplier* set they range from 6 to 14 bits.

3.2 Description of the Results

Tables 2–6 compare the results of MiniSat and our CIP approach. For each property listed in the "Prop" column the tables show the time in seconds of the two algorithms needed to solve instances of different input bit-widths. Results marked with '—' could not be solved within the time limit.

Prop	Meth	register width							
		5	10	15	20	25	30	35	40
#1	SAT	0.0	0.0	0.1	0.2	0.3	0.4	0.6	0.7
	CIP	0.3	0.6	0.7	1.6	2.3	4.7	7.7	10.5
#2	SAT	0.0	0.0	0.1	0.2	0.3	0.4	0.6	0.8
	CIP	0.2	2.2	1.2	2.8	3.5	4.6	7.7	9.4
#3	SAT	0.0	0.1	0.1	0.2	0.3	0.6	1.0	1.2
	CIP	0.7	3.0	2.6	9.3	13.5	20.4	21.8	28.8
#4	SAT	0.0	0.0	0.1	0.2	0.2	0.3	0.5	0.7
	CIP	0.5	2.8	6.2	11.1	23.1	33.0	6.3	7.3
#5	SAT	0.0	0.0	0.1	0.2	0.3	0.6	0.8	1.0
	CIP	1.1	5.4	15.0	32.5	52.0	92.0	55.1	125.2
#8	SAT	0.0	0.1	0.1	0.2	0.3	0.6	0.7	1.1
	CIP	1.2	8.1	19.8	43.1	45.5	89.9	118.9	91.4
#6	SAT	2.8	—	—	—	—	—	—	—
	CIP	0.0	0.1	0.1	0.2	0.3	0.4	0.6	0.7
#7	SAT	8.3	—	—	—	—	—	—	—
	CIP	0.1	0.2	0.5	1.0	1.8	2.9	4.8	6.8
#9	SAT	34.5	—	—	—	—	—	—	—
	CIP	0.1	0.5	1.6	3.5	7.1	11.6	19.8	27.9
#10	SAT	2.2	—	—	—	—	—	—	—
	CIP	0.1	0.1	0.2	0.4	0.6	0.8	1.1	1.5

Table 4. PipeMult invalid (top) and valid (bottom) properties.

The top parts of Tables 2 to 5 contain invalid properties (i.e., where the corresponding SAT and CIP instances are feasible). One can see that SAT clearly outperforms CIP to find counter-examples. This is due to the fact, that SAT processes the nodes much faster than CIP and is thereby able to investigate many more nodes in the same amount of time.

For most of the valid properties , as shown in the bottom parts of Tables 2 to 5, CIP dominates the SAT approach. Dramatic improvements can be observed on the *muls* and *neg_flag* properties of the ALU circuit (Table 2), on the *PipeMult* instances (Table 4), and on the *g2_checkg2* instance of the *Biquad* circuit (Table 5).

There seems to be no clear winner on the equivalence checking *Multiplier* examples (Table 6). SAT is faster on the smaller instances, but the running time for the CIP approach increases to a lesser extent with the bit-width of the inputs. The 10×10 multiplication circuits were verified with both techniques in roughly the same time. For larger bit widths, the CIP approach is superior to SAT.

In total, SAT failed on 58 out of 258 instances, while CIP was able to solve all instances within the time limit.

Prop	Meth	variant A	B	C
g3_checkreg1	SAT	0.0	0.0	0.0
	CIP	1.2	0.9	0.9
g3_xtoxmdelay	SAT	0.0	0.0	0.0
	CIP	0.5	0.1	0.1
g3_checkgfail	SAT	0.7	0.6	1.2
	CIP	290.0	29.2	50.2
g_checkgpre	SAT	22.2	57.6	29.1
	CIP	14.2	12.3	15.3
g2_checkg2	SAT	—	—	—
	CIP	213.9	204.8	257.6
g25_checkg25	SAT	0.0	2.4	2.5
	CIP	29.7	22.4	24.2
g3_negres	SAT	0.0	0.0	0.0
	CIP	0.7	0.0	0.0
gBIG_checkreg1	SAT	287.2	157.3	159.6
	CIP	170.0	7.0	8.6

Table 5. Biquad invalid (top) and valid (bottom) properties.

Layout	Meth	register width 6	7	8	9	10	11	12	13	14
booth	SAT	0.4	3.3	21.0	135.4	935.1	—	—	—	—
signed	CIP	21.3	70.1	318.7	384.2	904.1	1756.2	2883.7	4995.9	3377.9
booth	SAT	0.5	2.5	17.9	102.9	879.0	4360.4	—	—	—
unsgnd	CIP	15.7	51.7	269.1	911.3	1047.6	2117.7	2295.1	4403.4	7116.8
nonbth	SAT	0.4	3.4	21.8	134.1	1344.1	—	—	—	—
signed	CIP	12.8	31.2	100.6	265.9	569.8	690.8	1873.0	1976.3	4308.9
nonbth	SAT	0.3	1.8	16.5	83.1	909.6	5621.5	—	—	—
unsgnd	CIP	3.6	22.4	111.2	214.0	335.4	1040.1	1507.5	2347.7	4500.2

Table 6. Multiplier properties (all valid).

4 Conclusions

We described an algorithm for the property checking problem that combines techniques from integer programming, constraint programming, and SAT solving. The algorithm includes solving the linear programming (LP) relaxation at every node of the search tree, applying constraint specific domain propagation algorithms using both, bit and word level representations, and learning conflict constraints from infeasible LPs as well as from bit and word level deductions. Experimental results on industrial benchmarks showed that our approach outperforms SAT techniques for proving the validity of properties on circuits containing arithmetics. However, due to the much more involved procedures employed in our algorithm it is inferior to SAT for finding counter-examples of an invalid property.

References

[1] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007. Special issue: Mixed Integer Programming.

[2] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007. `http://opus.kobv.de/tuberlin/volltexte/2007/1611/`.

[3] E. Amaldi, M. E. Pfetsch, and L. E. Trotter, Jr. On the maximum feasible subsystem problem, IISs, and IIS-hypergraphs. *Mathematical Programming*, 95(3):533–554, 2003.

[4] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[5] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT-based approach for solving formulas over boolean and linear mathematical propositions. In *Proc. Conference on Automated Deduction (CADE)*, pages 195–210, 2002.

[6] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. Intl. Design Automation Conference (DAC-99)*, pages 317–320, June 1999.

[7] R. Brinkmann. *Preprocessing for Property Checking of Sequential Circuit on the Register Transfer Level*. PhD thesis, University of Kaiserslautern, Kaiserslautern, Germany, 2003.

[8] R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In *Proc. Asia and South Pacific Design Automation Conference (ASPDAC-02)*, Bangalore, India, 2002.

[9] D. Chai and A. Kuehlmann. A fast pseudo-boolean constraint solver. In *Proc. Design Automation Conference (DAC-03)*, pages 830–835, 2003.

[10] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Proceedings of SAT 2003*, pages 502–518. Springer, 2003.

[11] F. Fallah, S. Devadas, and K. Keutzer. Functional vector generation for HDL models using linear programming and boolean satisfiability. *IEEE Transactions on CAD*, CAD-20(8):994–1002, August 2001.

[12] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Proc.Intl. Conf. Computer Aided Verification(CAV-04)*, pages 26–37, July 2004.

[13] E. Goldberg and Y. Novikov. Berkmin: A fast and robust SAT solver. In *Design Automation and Test in Europe (DATE)*, pages 142–149, 2002.

[14] ILOG. Cplex. `http://www.ilog.com/products/cplex`.

[15] D. Kroening and S. A. Seshia. Formal verification at higher levels of abstraction. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2007.

[16] J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions of Computers*, 48:506–521, 1999.

[17] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, July 2001.

[18] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[19] Onespin Solutions GmbH. OneSpin 360 MV. `http://www.onespin-solutions.com/360mv.php`.

[20] `http://www.opencores.org`.

[21] G. Parthasarathy, M. Iyer, K.-T. Cheng, and L. Wang. An efficient finite-domain constraint solver for RTL circuits. In *Proc. Intl. Design Automation Conference (DAC-04)*, June 2004.

[22] G. Parthasarathy, M. K. Iyer, K. T. Cheng, and F. Brewer. RTL SAT simplification by boolean and interval arithmetic reasoning. In *Proc. ntl. Conference on Computer-Aided Design (ICCAD-05)*, 2005.

[23] L. Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

[24] M. W. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[25] Z. Zeng, M. Ciesielski, and B. Rouzeyre. Functional test generation using constraint logic programming. In *Proc. IFIP International Conference on Very Large Scale Integration (IFIP VLSI-SOC 2001)*, Montpellier, France, 2001.

[26] Z. Zeng, P. Kalla, and M. Ciesielski. LPSAT: A unified approach to RTL satisfiability. In *Proc. Conference on Design, Automation and Test in Europe (DATE-01)*, Munich, Germany, March 2001.

[27] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.