Abstract. This paper describes our new satisfyability (SAT) modulo theory (SMT) solver STABLE for the quantifier-free logic over fixed-sized bit vectors. Our main application domain is formal verification of system-on-chip (SoC) modules designed for complex computational tasks, for example, in signal processing applications. Ensuring proper functional behavior for such modules, including arithmetic correctness of the data paths, is considered a very difficult problem.

We show how methods from computer algebra can be integrated into an SMT solver such that instances can be handled where the arithmetic problem parts are specified mixing various levels of abstraction from the plain gate level for small highly optimized components up to the pure word level used in high-level specifications. If the arithmetic problem parts include multiplications such mixed problem descriptions quickly drive current SMT solvers towards their capacity limits.

High performance data paths are often designed at a level of abstraction that we call the *arithmetic bit level* (ABL). We show how ABL information, if available in an SMT instance, can be used to transform the decision problem into an equivalent set of variety subset problems. These problems can be solved efficiently with techniques from computer algebra based on Gröbner basis theory over finite rings $\mathbb{Z}/\langle 2^n \rangle$. Sometimes, instances contain problem parts at a level below the ABL using gate-level operations. These problem parts, e.g., originate from custom-designed arithmetic components that are highly optimized using the gate-level constructs of a hardware description language (HDL). For such cases we integrate a local ABL extraction technique based on local Reed-Muller forms.

Dagstuhl Seminar Proceedings 09461
Algorithms and Applications for Next Generation SAT Solvers
http://drops.dagstuhl.de/opus/volltexte/2010/2509

Solving hard instances in QF-BV combining Boolean reasoning with computer algebra

Markus Wedler[1], Evgeny Pavlenko[1], Alexander Dreyer[3], Frank Seelisch[2], Dominik Stoffel[1], Wolfgang Kunz[1], Gert-Martin Greuel[2]

[1] Electronic Design Automation Group, Department of Electrical- and Computer-Engineering
University of Kaierslautern, Germany
{wedler,pavlenko,stoffel,kunz}@eit.uni-kl.de
[2] Computer Algebra Group, Department of Mathematics
University of Kaierslautern, Germany
{seelisch,greuel}@mathematik.uni-kl.de
[3] Abteilung Systemanalyse, Prognose und Regelung
Fraunhofer Institut für Techno- und Wirtschaftsmathematik (ITWM)
Fraunhofer-Platz 1, Kaiserslautern, Germany
alexander.dreyer@itwm.fraunhofer.de

1 Introduction

Modern design flows for Systems-on-Chip (SoCs) pursue a correctness-by-integration strategy when verifying the functionality of the overall system. This requires high quality designs for the individual modules that are supposed to be integrated into the SoC. Traditional simulation-based verification techniques are reaching their capacity limits rapidly. This motivates the application of highly automated property checking techniques. A promising approach is called *Interval Property Checking (IPC)* [1,2]. It is mostly based on satisfiability solving (SAT) and SAT modulo theory solving (SMT).

IPC has been used in industry for many years to ensure correctness of the individual SoC modules. This does not only lead to high quality IP (intellectual property) modules but also reduces the costs for system integration and chip-level simulation. Given IP modules of provably high quality, chip-level simulation may concentrate on true system-level aspects and is relieved from hunting bugs in local modules. IPC has the capacity to handle almost all types of modules that can be found in today's SoCs. Nonetheless, a few pathological cases remain that sometimes limit its application in industrial practice. In particular, data paths are often a challenge. This is true, especially, if not only the correctness of the control flow but also correctness of the computed data needs to be proved.

For complex arithmetic data paths simulation is therefore still prevailing in industrial verification environments. This is due to the inability of standard proving procedures based on satisfiability solving (SAT), SAT modulo theory solving (SMT), or binary decision diagrams (BDDs) to handle arithmetic functions. In particular, multiplication — as it is part of nearly all data paths for signal processing applications — has remained a severe problem for standard tools. This

deficiency has stimulated a lot of research on specialized proof methods focussing on arithmetic. In the following section, we sketch some of this research.

1.1 Related Work

Sometimes the validity of a property can be proven without considering the exact functionality of the data path. In such cases abstraction and refinement techniques have shown superior to pure Boolean SAT techniques. A survey on these techniques can be found in [3]. However, there are other cases where the properties depend on the exact functionality of the data path. In such situations, it is quite unlikely that a suitable abstraction can be found.

Another interesting direction of research investigates SAT-modulo-theory (SMT) solvers. These solvers combine a SAT solver with specialized solvers for certain well-selected theories. An example the latter is the theory of equality with uninterpreted functions used in UCLID [4]. The use of this logic is beneficial in cases where the arithmetic functionality is described in the design at the same level of abstraction as in the specification. More specifically, the logic of uninterpreted functions can reduce the complexity of a proof problem if the respective function symbols used in the specification and implementation can be mapped to each other without a semantic analysis. For high performance data paths, however, the level of abstraction used in register tranfer level (RTL) descriptions is often below the word level being used in properties. Therefore, the mapping is likely to fail.

The quantifier-free logic over fixed-sized bit vectors (QF-BV) is a logic that facilitates interpretation of bit vector functions with respect to their semantics. We consider this logic to be the natural choice to express the proof obligations that are generated by an interval property checker. SAT problems for QF-BV formulas can, in principle, be solved using solvers such as Yices [5], MathSAT [6], Z3 [7], Boolector [8] or Spear [9]. If the decision problem originates from RTL property checking we observed, however, that these solvers often show the same performance bottlenecks as when plain SAT is applied to a bit-blasted version of the problem. The reason for this behavior is again that large portions of the arithmetic circuitry are specified at the bit level in order to conduct manual optimizations and in order to customize the exact structure of this circuitry. In these bit level parts the solvers cannot exploit word level information and, thus, loose their advantages over SAT.

For equivalence checking of arithmetic RTL circuits, especially multipliers, a technique based on rewriting was proposed in [10]. A database of rewrite rules is provided to support a large number of widely used multiplier implementation schemes. However, for non-standard implementations the approach requires updating the database manually and is thus not fully automatic. Fully automatic techniques for equivalence checking and debugging of arithmetic data paths are provided in [11,12]. These techniques extract arithmetic bit level information from low level gate net lists. They consider the datapath to be clearly separated from control logic which in high performance RTL designs is often not the case.

This renders integration of the techniques into a general purpose SMT solver with a property checking application scenario exceedingly difficult.

Recently, techniques from symbolic computer algebra have entered the verification arena. In [13] a procedure is presented that determines whether a multivariate polynomial with fixed word length operands is vanishing. This makes it possible to compare polynomial representations for bit vector functions. The approach is extended towards multiple word length operands in [14,15]. Both approaches, however, require a clean word-level representation of the data paths. This limits their applicability in RTL property checking as has been discussed above.

In principle, it is possible to transform a circuit related SMT problem into an algebraic problem over $\mathbb{Z}/2$. Efficient computer algebra systems for this special case are available [16,17]. In [16,17] polynomials are represented by zero suppressed binary decision diagrams (ZDDs). As a result during the algebraic computation functional decision diagrams (FDDs) of the original circuitry are generated. FDDs, however, are known to grow exponentially for multipliers and are therefore unsuitable for the problems considered in this paper.

At the bit level arithmetic circuitry is typically specified using arithmetic entities such as half adders and full adders. If such entities can be identified within the design we call the resulting netlist an arithmetic bit level (ABL) description. An approach for verification of such bit level implementations using Gröbner basis theory over fields is reported in [18]. This approach requires polynomial specifications for every building block in the hierarchy of the arithmetic circuit design. After proving that a block, e. g., a carry-save adder, fulfills its local specification the polynomial representation is used to verify the block in the next level of the hierarchy. However, since the correctness proof includes a range check the intermediate results at the block boundary are required to have sufficient bit width to represent every possible result. For designs implementing integer arithmetic with fixed bit width this is often not available.

A heuristic approach to exploit arithmetic bit level (ABL) information in RTL designs has been reported in [19]. By local equivalence transformation of the arithmetic bit level description a reduced normal form is computed that is sufficient to prove the arithmetic problem parts of a property checking instance and relieves the SAT/SMT solver from reasoning in structurally different implementations for the same arithmetic function. This method is subsumed by the more general algebraic approach presented here which also provides a well-understood mathematical basis for ABL-based verification techniques.

1.2 Contribution

In this paper we present our new SMT solver STABLE for the quantifier-free logic over bit vectors. This solver integrates two recently developed techniques for solving hard arithmetic problems. These techniques allow for application of the Gröbner basis theory over finite rings.

At first, we convert the arithmetic problem parts of the decision problem into equivalent variety subset problems and show how these problems can be solved

efficiently using Gröbner basis theory over finite rings $\mathbb{Z}/\langle 2^n \rangle$. In order to solve decision problems at the arithmetic bit level we combine the technique of [19] with further computer algebra algorithms for the ring $\mathbb{Z}/\langle 2^n \rangle$. In particular, we exploit the fact that the ABL decision problems of [19] can be transformed into a set of *variety subset problems*. Under certain monomial orderings the set G of polynomials generated from the ABL components forms a Gröbner basis of the ideal $I = \langle G \rangle$ generated by these polynomials. This allows to efficiently solve the variety subset problem and decide problems at the arithmetic bit level.

The second technique integrated into STABLE is an ABL extraction technique for cases where some arithmetic parts of the problem are formulated below the ABL. In instances derived from property checking this, e.g., may originate from highly optimized custom-designed components that are instantiated within the datapath of the design and that are specified below the ABL using gate-level operations of the HDL.

The remainder of the paper is organized as follows: Section 2 shows how arithmetic sub-problems of an SMT instance can efficiently be solved using computer algebra techniques. In this section we assume the arithmetic problem parts to be specified at the arithmetic bit level or word level. For cases where problem parts are specified below the ABL we include an extraction technique based on local Reed-Muller forms presented in Section 3. Section 4 illustrates our strategy for integrating the proposed techniques into an SMT solver for the quantifier-free logic over fixed-sized bit vectors (QF-BV). The paper presents experimental results in Section 5 and concludes with Section 6.

2 Using computer algebra for arithmetic SMT problems

This section recalls the mathematical models required to solve arithmetic problem parts of an SMT decision problem for the quantifier-free logic over bit-vectors with algebraic techniques as introduced in [20].

2.1 Mathematical background

We first study how to model ABL components by polynomials over a unique coefficient ring. Due to the finite bit width used in QF-BV instances the ring $\mathbb{Z}/\langle 2^n \rangle$ turns out to be the natural choice. However, the mapping of a decision problem for QF-BV to an algebraic problem in terms of polynomials from $\mathbb{Z}/\langle 2^n \rangle [X]$ with variables $X = (x_1, \ldots, x_k)$ over such a ring is not trivial and will be detailed in the sequel. The key observation is that we can construct a set G of such polynomials modeling the arithmetic problem parts of the instance. The resulting polynomial set G is a Gröbner basis of the generated ideal $I = \langle G \rangle$. This makes the proposed approach computationally feasible.
See e.g. [21] for a summary on Gröbner basis theory, and [22] for the specifics over coefficient rings. All Gröbner bases arising from our practical approach will in fact be strong Gröbner bases in the sense of the definition given in [22], thus in the remainder of this paper we shall use the terms Gröbner basis and strong

Gröbner basis interchangeably. Moreover, the following notion of reduction has been adapted and simplified to the setting of strong Gröbner bases.

We require a *global monomial ordering* $<$, i. e., a well ordering on the set of monomials such that multiplication with a monomial respects the ordering. Here, a *monomial* is a power product of variables and a term is the product of a monomial with a coefficient, i. e., an element of the ring $\mathbb{Z}/\langle 2^n\rangle$. Any polynomial $f \neq 0$ can be written as a finite sum of terms, $f = c_1 m_1 + \cdots + c_r m_r$ with coefficients $c_i \neq 0$ and monomials m_i such that $m_1 > m_2 > \cdots > m_r$. The largest monomial m_1 plays a special role: we call $\mathrm{LM}(f) := m_1$ the *leading monomial*. Likewise, $\mathrm{LC}(f) := c_1$ is called *leading coefficient* and $\mathrm{LT}(f) := c_1 m_1$ refers to the so called *leading term* of f.

Let $G \subset \mathbb{Z}/\langle 2^n\rangle[X]$ be a finite set of polynomials and $f \in \mathbb{Z}/\langle 2^n\rangle[X]$. If $\mathrm{LT}(f) = c_1 m_1$ is divisible by the leading term of an element $h \in G$ we say that f is reducible to $f' := f - (\mathrm{LT}(f)/LT(h)) \cdot h$ and write $f \underset{h}{\rightarrow} f'$. The transitive closure of the relation $\underset{h}{\rightarrow}$ is denoted by $\overset{*}{\underset{G}{\rightarrow}}$. If $f \overset{*}{\underset{G}{\rightarrow}} g$ and if g is not reducible by any h of G we call g a *normal form* of f with respect to G. Algorithm 1 computes such a normal form which is even reduced, cf. [21].

Require: *f a polynomial, G a finite set of polynomials, $>$ a monomial ordering*
Ensure: *A normal form of f*
  while $f \neq 0$ *and* $\emptyset \neq G' = \{g \in G \ : \ \mathrm{LT}(g) \mid \mathrm{LT}(f)\}$ **do**
    Select $g \in G'$ *with* $\mathrm{LT}(f) = m \cdot \mathrm{LT}(g)$
    $f := f - m \cdot g_i$
  end while
  compute f', the normal form of $f - \mathrm{LT}(f)$ w.r.t. G
  return $\mathrm{LT}(f) + f'$

Algorithm 1: Reduced normal form algorithm

The notion of a normal form is, however, only useful if G is a Gröbner basis. In order to define a Gröbner basis we need the ideal $I = \langle H\rangle := \{\sum_{h\in H} f_h h | f_h \in \mathbb{Z}/\langle 2^n\rangle[X]\}$ generated by an arbitrary finite set H of polynomials. Note that for the sets of solutions $V(I) := \{p \in (\mathbb{Z}/\langle 2^n\rangle)^k | f(p) = 0 \text{ for all } f \in I\}$ and $V(H)$ we have $V(I) = V(H)$ for any set of generators H. A set of generators G is called a Gröbner basis (of I) if and only if $f \overset{*}{\underset{G}{\rightarrow}} 0$ for all $f \in I$. Theory provides that $I = \langle G\rangle$ for any Gröbner basis G, and that the reduced normal form of any element $f \in \mathbb{Z}/\langle 2^n\rangle[X]$ w.r.t. G is essentially unique and equal to 0 if and only if $f \in \langle G\rangle = I$.

With this basic knowledge of Gröbner basis theory we continue with converting arithmetic decision problems into equivalent variety subset problems.

2.2 Algebraic modeling of arithmetic decision problems

Without loss of generality, we assume that the SMT decision problem is represented as an acyclic netlist of bit vector functions. The arithmetic components of this netlist can be converted into a set of equations $G_j, j = 1, \ldots, m$ which are of the form

$$G_j : \sum_{i=0}^{n_j-1} 2^i r_i^{(j)} = f_j\left(a_1^{(j)}, a_2^{(j)}, \ldots, a_{m_j}^{(j)}\right) \mod 2^{n_j}. \tag{1}$$

In these equations the $f_j \in \mathbb{Z}[X]$ are polynomials over $\mathbb{Z}$ and depend on a finite set of variables X. For the variables $r_i^{(j)}, a_k^{(l)} \in X$ used in this equation we assume $r_i^{(j)} \neq a_k^{(l)}$ for $1 \leq l \leq j$ and all i, k. The variables correspond to the inputs and outputs of the bit vector function and we call the $a_i^{(j)}$ *inputs* and $r_i^{(j)}$ *outputs* of G_j accordingly.

We give a few examples demonstrating that the equations G_j can be easily generated from the netlist representing the decision problem. Furthermore, the condition $r_i^{(j)} \neq a_k^{(l)}$ is fulfilled as the netlist is acyclic by definition.

Example 1 *The partial products of a standard $a \times b$-bit multiplier (not Booth-encoded) can be modeled by the polynomial equations*

$$G_{i,k} : p_{i,k} = a_i b_k \ mod\ 2, (k = 0, \ldots, a-1, i = 0, \ldots, b-1).$$

Example 2 *A full adder with inputs a_0, a_1, a_2 and outputs s and c for sum and carry is modeled by the equation*

$$G_{FA} : 2c + s = a_0 + a_1 + a_2 \ mod\ 4.$$

Example 3 *An unsigned k-bit adder with inputs $a = (a_i | 0 \leq i < k)$ and $b = (b_i | 0 \leq i < k)$ and result $r = (r_i | 0 \leq i < k)$ is modeled by*

$$G_{adder} : \sum_{i=0}^{k-1} 2^i r_i = \sum_{i=0}^{k-1} 2^i (a_i + b_i) \ mod\ 2^k.$$

Using the values 1, 2 and k for the modulo n_j, these examples also demonstrate that the equations G_j are sufficient to model both word and bit level arithmetic components of a mixed ABL/word-level decision problem within the same formalism. It allows for utilization of word-level information where available, and at the same time can handle bit level arithmetic information.

To complete the modeling of the arithmetic parts of the decision problem we need to consider the proof goals. For every proof goal we obtain an additional

polynomial g depending on a subset $\{c_1, \ldots, c_t\} \subset X$ of the variables used in the above equations. We need to check whether

$$g(c_1, \ldots, c_t) = 0 \mod 2^n$$

holds for all common solutions of the set of equations $\{G_j\}$. Note that the value n used in the above equation depends on the bit width of the comparison constraint in the underlying problem instance.

We illustrate the model generation for the proof goal by means of an example.

Example 4 *A n-bit equality comparison of operands a and b is modeled by the polynomial*

$$g = \sum_{i=0}^{n-1} 2^i (a_i - b_i)$$

In the remainder of this section we will show how to convert the set of equations $\{G_j\}$ and the proof goal g into a variety subset problem that is equivalent to our ABL decision problem.

The set of all solutions to $\{G_j\}$ is denoted as $V(\{G_j\})$. Analogously, let $V(g)$ be the set of all roots of g. Usually the equations G_j and the polynomial g are given $\mod 2^k$ for different k. We apply a number of transformations to create an equivalent *variety subset problem* $V(\{h_i\}) \subset V(g)$ where h_i and g are polynomials over a single ring $\mathbb{Z}/2^N$ with appropriate N. This is necessary in order to apply the methods of computer algebra. The problem can then be solved by constructing a Gröbner basis and using normal form computations with respect to this basis.

Instead of directly converting the equations G_j into a set of polynomials over a single ring we generate some additional equations. These equations are redundant in the sense that they can be derived from the original equations G_j. However, they will play an important role for the efficiency of the solution techniques described in Section 2.3. More precisely, these equations ensure that the polynomial system generated from them is a Gröbner basis of the corresponding ideal. This will be discussed later. For every G_j we generate n_j equations

$$G_j^{(t)} : \sum_{i=0}^{t-1} 2^i r_i^{(j)} = f_j^{(t)} \left(a_1^{(j)}, a_2^{(j)}, \ldots, a_{m_j}^{(j)} \right) \mod 2^t$$

with $t = 1, \ldots, n_j$ and with $f_j^{(t)} = f_j \mod 2^t$ being the minimal polynomial [23] representing the same polynomial function $\left(\mathbb{Z}/2^t\right)^{m_j} \to \mathbb{Z}/2^t$ as f_j. Obviously, every solution of the G_j is also a solution of the system $\{G_j^{(t)} \mid t = 1, \ldots, n_j\}$ and vice versa.

Let S be the set of variables (signals) occurring in g saturated with respect to the property that if $r_{t-1}^{(j)} \in S$ then all variables of $G_j^{(t)}$ are also in S. For the next steps, only the equations $G_j^{(t)}$ with $r_{t-1}^{(j)} \in S$ are relevant. The solution set for the variables in S does not change when omitting the other equations. Note that this corresponds to a cone-of-influence reduction on the netlist of a circuit.

Example 5 *Suppose the n-bit final adder of a multiply/accumulate unit is reused for computing an m-bit addition ($m < n$). In a property checking instance for this addition only the lower m bits of the adder influence on the arithmetic result. By the above construction we only instantiate the equations*

$$G^{(t)}_{adder} : \sum_{i=0}^{t-1} 2^i r_i = \sum_{i=0}^{t-1} 2^i (a_i + b_i) \ mod\ 2^t$$

for $t < m$.

So far the equations $G^{(t)}_j$ use the operation $\bmod 2^t$ for different t. In other words, we work in different rings $\mathbb{Z}/2^t$ and no ring is contained in any other (we have only surjections of rings $\mathbb{Z} \twoheadrightarrow \mathbb{Z}/2^{t'} \twoheadrightarrow \mathbb{Z}/2^t$ if $t' \geq t$).

In order to apply Gröbner basis techniques to our problem we need to generate a set of polynomials over a single ring. As a first step we determine a sufficient size $N \in \mathbb{N}$ of the ring. In our current implementation, we start with

$$N := n + \max\{n_j \mid j = 1, \ldots, m\}$$

with n_k, n, m as above.

However, this is a heuristic choice that turns out to be sufficient for many practical problem instances. Nonetheless it is possible to construct examples where a larger N would be required. Our current implementation detects these cases and automatically moves to a sufficiently larger ring with size $N' > N$.

As a second step we transform every equation into an element of the polynomial ring $\mathbb{Z}/2^N[X]$ over $\mathbb{Z}/2^N$. This is achieved by introducing new variables $s^{(j)}_t$ called *slack variables* and by considering the polynomials

$$\widetilde{G}^{(t)}_j := \sum_{i=0}^{t-1} 2^i r^{(j)}_i - f^{(t)}_j \left(a^{(j)}_1, a^{(j)}_2, \ldots, a^{(j)}_{m_j}\right) - 2^t s^{(j)}_t.$$

The set of common roots for the $\widetilde{G}^{(t)}_j$ projected on the variables in S corresponds to $V(\{G_j\})$. We can omit some of the extra variables $s^{(j)}_t$ if we know that $0 \leq f^{(t)}_j \leq 2^t - 1$ holds over $\mathbb{Z}$. If this condition cannot be guaranteed and we need to know the exact value of $s^{(j)}_t$ during the computation we can replace $s^{(j)}_t$ by a polynomial in the variables $a^{(j)}_1, a^{(j)}_2, \ldots, a^{(j)}_{m_j}$, i. e., a subset of the inputs of G_j. For example, in the polynomial $r_0 - a_0 - a_1 + 2s$ determining the lower result bit r_0 of a half adder the the slack variable s is can be substitued by the polynomial $a_0 a_1$. In general, the polynomials for the slack variables can be very large even for small polynomials $f^{(t)}_j$. In order to avoid these large polynomials it is often better to introduce the slack variables. This especially applies to cases where the slack variables are canceled out in the further course of computation. Let $G = \{\widetilde{G}^{(t)}_j \mid j = 1, \ldots, m \text{ and } t = 1, \ldots, n_j\}$ and $I = \langle G \rangle$ be the ideal generated by this set. Using the language of computer algebra our decision problem can be formulated by the following question regarding the variety $V(I)$:

Is $V(I) \subset V(\{2^{N-n}g\})$, where $V(P)$ denotes the set of all common roots of the polynomials in an arbitrary polynomial set P ?

In the next section we will detail how to efficiently solve this so called variety subset problem.

2.3 Solving arithmetic decision problems by normal form computation

The following proposition is key for an effective solution of the variety subset problem introduced in the previous section.

Proposition 1 *The set* $G = \{\widetilde{G}_j^{(t)}\}$ *is a Gröbner basis with respect to any monomial order refining the following partial order*

$$r_i^{(j)} > \textit{every monomial in the variables } a_k^{(j)}, s_t^{(j)}, r_l^{(j)}$$

for all i, k, t, j *and* $l < i$.

By Lemma 1 we prove that normal form computation can be used as an effective procedure for solving our problem.

Lemma 1 *Let* G *be a Gröbner basis of an ideal* $I \subset \mathbb{Z}/2^N[X]$, $X = (X', X'')$, *and* g *a polynomial such that the normal form* h *of* g *with respect to* G *is in* $\mathbb{Z}/2^N[X']$. *Assume that for all* X' *there exist* X'' *with* $f(X', X'') = 0$ *for all* $f \in G$. *Then,* h *defines the zero function if and only if* $V(G) \subset V(g)$.

Let $g \in \mathbb{Z}/2^n[X]$ and h be the normal form of $2^{N-n}g$ with respect to G which can be computed [24] by Algorithm 1. Since we are only interested in the function of h on $V(I)$ we can always replace portions of h by equivalent polynomials with respect to $V(I)$. In particular, we can replace every slack variable in the normal form by a polynomial expression in the inputs of the corresponding equation G_j. Therefore, we may assume that h does not contain any slack variables. Furthermore, the output variables of the equations G_j do not occur in h as otherwise h would be reducible by some of the generated sub-identities $G_j^{(t)}$, hence h satisfies the assumptions of Lemma 1.

This guarantees that the variables present in h are inputs to the bit vector netlist representing the ABL decision problem. Every valuation of these variables can be extended to a consistent valuation for the remaining signals of this netlist. Furthermore, we can effectively decide whether h defines the zero function for all rings $\mathbb{Z}/m$ (cf. [23]) and therefore decide the ABL problem by Lemma 1.

3 Extracting local ABL information

In this section we recall a local extraction technique [25] that generates ABL information for those problem parts of an SMT instance where local parts of

an arithmetic problem are specified at the gate-level. Representing gate-level circuitry by a polynomial equation G_j as introduced in Equation 1 of Section 2.2 is not straight forward. In particular, only a few bit vector functions $f : (\mathbb{Z}/\langle 2^n \rangle)^k \to \mathbb{Z}/\langle 2^n \rangle$ are polynomial functions [23]. This explains why it is not advisable to encode an entire SMT instance as a monolithic algebraic problem. However, for Boolean functions $f : (\mathbb{Z}/\langle 2 \rangle)^k \to \mathbb{Z}/\langle 2 \rangle$ a polynomial representation is always feasible. In principle, every Boolean function can be represented by the above mentioned polynomial equation G_j based on its Reed-Muller form. The positive Reed-Muller (positive Davio) decomposition for a Boolean function $f : \{0,1\}^n \to \{0,1\}$ with respect to a variable x_i is given by the following equation:

$$f(x_0, \ldots, x_m) = f|_{x_i=0} \oplus x_i \wedge (f|_{x_i=0} \oplus f|_{x_i=1}), \tag{2}$$

where $f|_{x_i=1} = f(x_0, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_m)$ and $f|_{x_i=0} = f(x_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_m)$ denote the positive and negative cofactors of f with respect to x_i. Recursive application of this decomposition results in the Reed-Muller form for a given Boolean function.

Notice that the Reed-Muller form $f(x_0, \ldots, x_m)$ for a signal y is essentially a Boolean polynomial, i. e., a polynomial with coefficients and exponents in $\{0,1\}$ resulting in the following equation:

$$y = f(x_0, \ldots, x_m)\%2. \tag{3}$$

In this equation and throughout the remainder of this paper the binary operation $x\%y$ on integers $x, y \in \mathbb{Z}$ shall compute the remainder of the integer division of x by y, i. e., the smallest integer $k \geq 0$ such that there exists an integer $l \in \mathbb{Z}$ with $x = ly + k$.

Unless the Reed-Muller form only consists of a single product term, this equation requires a new slack variable to be introduced if we follow the transformation steps presented in Section 2.2. Moreover, the slack variable is very likely to remain in the final normal form h of the proof goal. In the following subsection we study how to transform the Reed-Muller form of a Boolean function into a polynomial equation that is suitable for our verification approach based on algebraic normal form computation.

We calculate a polynomial $f' \in \mathbb{Z}/\langle 2^n \rangle[x_0, \ldots, x_m]$ such that

$$f'(x_0, \ldots, x_m) = y = f(x_0, \ldots, x_m)\%2$$

for all Boolean valuations of the x_i.

Proposition 2 *Let $f = \sum_{i=0}^{k} t_i \in \mathbb{Z}[X]$ be a Boolean polynomial, i. e., the coefficients c_i of the terms t_i are restricted to $c_i = 1$. Likewise, all variables have exponent 1. For every $n \in \mathbb{N}$ a polynomial $f' \in \mathbb{Z}/\langle 2^n \rangle [X]$ exists such that the following condition holds for all Boolean valuations of the variables in X:*

$$f'(X)\%2^n = f(X)\%2.$$

By means of this proposition we can generate ABL information for Boolean functions that is suitable for proving properties by normal form computation. The following example illustrates this transformation.

Example 6 *We consider the Boolean function* $r(a,b,c) = a \oplus b \oplus c$ *and polynomials over the ring* $\mathbb{Z}/\langle 8\rangle\,[a,b,c]$. *For corresponing polynomial* $f = a+b+c$ *for* r *we want to determine a polynomial* f' *with* $f'\%8 = f\%2$ *for all Boolean valuations of* a, b *and* c. *For these valuations, the polynomial function defined by* f *can take values in* $\{0,1,2,3\}$. *Encoding this result by three bits* (r_2, r_1, r_0) *results in the Boolean functions* $r_1(a,b,c) = ab \oplus ac \oplus bc$ *and* $r_2(a,b,c) = 0$ *with the corresponding polynomials* $f_1 = ab + ac + bc$ *and* $f_2 = 0$. $f_2' = f_2$ *fulfills the proposition. However,* f_1 *can take values* $\{0,1,2,3\}$. *Again, we only need to consider the second bit of this result that can be expressed by the Boolean function* $r_3(a,b,c) = abc$. *The corresponding polynomial* $f_3' = f_3 = abc$ *fulfills the condition of the proposition. Obviously, the polynomial*

$$f_1' = f_1 - 2f_3' = ab + ac + bc - 2abc$$

fulfills the condition $f_1'\%8 = f_1\%2$ *of the proposition. This also applies to the polynomial*

$$f' = f - 2f_1' = a + b + c - 2ab - 2ac - 2bc + 4abc$$

.

A circuit for evaluation of this polynomial for Boolean values of the variables can be implemented by the half-adder/full-adder netlist depicted in Figure 1.

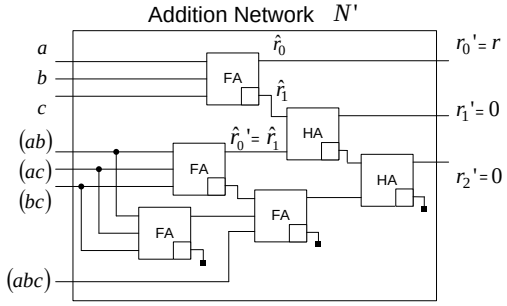

Fig. 1. Synthesized ABL model for the Reed-Muller form

Our overall procedure for extraction of polynomial for local gate-level parts of an SMT instance has two phases:

- Transform the local gate-level descriptions into Reed-Muller forms.
- Transform the Reed-Muller forms into equivalent polynomial equations as needed during normal form computation.

These algorithms are invoked on demand whenever the normal form computation terminates with a non-zero polynomial such that a gate level representation for the leading monomial of this polynomial exists. After converting this representation into a polynomial equation we continue with the normal form computation

We conclude this section by illustrating the extraction process by means of a small example. Suppose it is required to verify the design of a 2x2 unsigned multiplier with (radix-4) Booth-encoded partial products. We further assume that the partial products of the design are implemented at the gate level. The partial products provided for the addition tree of the design are listed in the third column of Table 1.

Table 1. Partial products and extracted polynomial equation for Booth-encoded unsigned multiplier

column	*result bit*	*Booth-encoded (radix 4) partial products*	*extracted polynomials*
0	r_0	$cpl_0 = \{b_1\}$	$cpl_0 = b_1 \% 16$
		$p'_0[0] = \{a_0b_0 \oplus b_1\}$	$p'_0[0] = a_0b_0 + b_1 - 2a_0b_0b_1 \% 16$
1	r_1	$p'_0[1] = \{b_1 \oplus a_1b_0 \oplus a_0b_1 \oplus a_0b_0b_1\}$	$p'_0[1] = b_1 + a_1b_0 - a_0b_1$ $+a_0b_0b_1 - 2a_1b_0b_1 \% 8$
2	r_2	$p'_1[2] = \{a_0b_1\}$	$p'_1[2] = a_0b_1 \% 8,$
		$cpl_1 = \{0\}$	$cpl_1 = 0 \% 4$
		$p'_0[2] = \{b_1 \oplus a_1b_1 \oplus a_1b_0b_1\}$	$p'_0[2] = b_1 - a_1b_1 + a_1b_0b_1 \% 4$
3	r_3	$signext = \{1\}$	$signext = 1 \% 2$
		$p'_1[3] = \{a_1b_1\}$	$p'_1[3] = a_1b_1 \% 2$
		$\overline{p'_0[3]} = \overline{cpl_0} = \{b_1 \oplus 1\}$	$\overline{p'_0[3]} = 1 - b_1 \% 2$

Furthermore, we annotate in braces the corresponding Reed-Muller form in terms of the multiplier inputs a_k and b_i. We assume the addition tree of the implementation to be specified at the ABL. After normal form computation for the proof goal

$$8r_3 + 4r_2 + 2r_1 + r_0 - 4a_1b_1 - 2a_0b_1 - 2a_1b_0 - a_0b_0$$

with respect to the polynomials generated from this addition tree we obtain the following polynomial:

$$8(signext + p'_1[3] + \overline{p'_0[3]}) + 4(p'_1[2] + p'_0[2] + cpl_1)$$
$$+2p'_0[1] + cpl_0 + p'_0[0] - 4a_1b_1 - 2a_0b_1 - 2a_1b_0 - a_0b_0.$$

It is obvious that normal form computation alone cannot establish equivalence between reference and implementation, as the partial products a_0b_0 and a_1b_0 do not have an equivalent counterpart in the implementation. In order to continue with the normal form computation we convert the Reed-Muller forms for the partial products of the implementation into the corresponding polynomial equations. The results of this computation step are summarized in the

fourth column of Table 1. Based on these polynomials we conduct the normal form computation, i.e., a series of reductions, that is illustrated in Figure 2. The computation of Figure 2 reduces the prove goal to the zero polynomial and this proves that our implementation is correct.

Fig. 2. Normal form computation of the proof goal with respect to extracted polynomials

$$
\begin{aligned}
& 8(signext + p_1'[3] + \overline{p_0'[3]}) + 4(p_1'[2] + p_0'[2] + cpl_1) + 2p_0'[1] + cpl_0 + p_0'[0] \\
& \qquad -4a_1b_1 - 2a_0b_1 - 2a_1b_0 - a_0b_0 \\
\xrightarrow[signext]{} \; & 8(p_1'[3] + \overline{p_0'[3]}) + 4(p_1'[2] + p_0'[2] + cpl_1) + 2p_0'[1] + cpl_0 + p_0'[0] \\
& \qquad -4a_1b_1 - 2a_0b_1 - 2a_1b_0 - a_0b_0 + 8 \\
\xrightarrow[p_1'[3]]{} \; & 8(\overline{p_0'[3]}) + 4(p_1'[2] + p_0'[2] + cpl_1) + 2p_0'[1] + cpl_0 + p_0'[0] \\
& \qquad +4a_1b_1 - 2a_0b_1 - 2a_1b_0 - a_0b_0 + 8 \\
\xrightarrow[\overline{p_0'[3]}]{} \; & +4(p_1'[2] + p_0'[2] + cpl_1) + 2p_0'[1] + cpl_0 + p_0'[0] + 4a_1b_1 - 2a_0b_1 - 2a_1b_0 - a_0b_0 - 8b_1 \\
\xrightarrow[p_1'[2]]{} \; & +4(p_0'[2] + cpl_1) + 2p_0'[1] + cpl_0 + p_0'[0] + 4a_1b_1 + 2a_0b_1 - 2a_1b_0 - a_0b_0 - 8b_1 \\
\xrightarrow[p_0'[2]]{} \; & +4(cpl_1) + 2p_0'[1] + cpl_0 + p_0'[0] + 2a_0b_1 - 2a_1b_0 - a_0b_0 - 4b_1 + 4a_1b_0b_1 \\
\xrightarrow[cpl_1]{} \; & +2p_0'[1] + cpl_0 + p_0'[0] + 2a_0b_1 - 2a_1b_0 - a_0b_0 - 4b_1 + 4a_1b_0b_1 \\
\xrightarrow[p_0'[1]]{} \; & +cpl_0 + p_0'[0] - a_0b_0 - 2b_1 + 2a_0b_0b_1 \\
\xrightarrow[cpl_0]{} \; & +p_0'[0] - a_0b_0 - b_1 + 2a_0b_0b_1 \xrightarrow[+p_0'[0]]{} 0
\end{aligned}
$$

4 The SMT solver STABLE

In this section we will study how to integrate the techniques presented in Sections 2 and 3 into a generic solver for the quantifier-free logic over fixed-sized bit vectors (QF-BV). We present our SMT solver for this logic called STABLE following the flow chart illustrated in Figure 3.

The solver supports multiple input formats like the SMT-LIB format and the Spear format. Moreover, we support an intermediate format of the industrial interval property checker Onespin 360 MW [2]. In a preprocessing step we start with identifying control variables for the datapath as branching variables. Recall that our primary application domain is interval property checking. Interval properties usually focus on individual operations of a design. These operations may require a specific behavior of the environment of design as well as a specific

mode of operation for the design. In order to restrict environment and design with respect to the scenario covered by the property a verification engineer will specify so called assumptions. The intended behavior of the design is then specified as a commitment that is only valid for the current scenario. This allows for a compact and easy-to-review representation of the properties. The resulting SMT instances have an implicative overall structure $a \rightarrow c$. The sub formulas a and c correspond to assumption and commitment of the property and may share subexpressions. For example, if both, commitment and assumption of a property, refer to the same design signal, the cone of influence of this signal will become part of the instance and will be shared by both expressions. The assumption a typically restricts the valid valuations for control variables in the cone of influence of the commitment, i.e., these variables may only take a subset of all possible value assignments. We identify the constraints in the SMT instance that encode the assumption and assert that the assumption is fulfilled. In an outer loop of the solving algorithm we determine every possible valuation $v = V_{branch}$ of the control values v that leads to a new configuration of the datapath. We propagate these assignments in order to remove the control logic between the arithmetic components in the instance. Given a specific configuration for the control part of the instance, we determine the arithmetic constraints in the cone of influence of the proof goal. For these constraints polynomials are generated as described in section 2.2. Recall that the generated set of polynomials G forms a Gröbner basis for the generated ideal $I = \langle G \rangle$.

In the next step of the algorithm, we decide whether the arithmetic proof goal is valid under the current configuration of the datapath indicated by the current valuation V_{branch} of the branching variables v. We encode the proof goal by an appropriate polynomial f and compute its normal form with respect to the Gröbner basis G. If the normal form $NF(f, G)$ is a vanishing polynomial, the proof goal is valid under the current configuration of the control logic and we learn the constraint $v \neq V_{branch}$. Otherwise, we analyze whether the remaining variables in the normal form of the proof goal are defined by non arithmetic constraints. If this is the case, we try to extract further polynomials with the technique described in section 3 and start the normal form computation again.

This inner loop is repeated until either the normal form vanishes or no new polynomials can be extracted. In the latter case we learn the constraint $(v = V_{branch} \wedge NF(f, G) = 0) \rightarrow var(f)$. The constraint refers to the variable $var(f)$ in the cone of influence of c that encodes the proof goal. In our implementation the above mentioned constraint is only learned, when it is considerably simpler than the original problem. The decision whether the constraint is simple enough is based on the number of variables used in $NF(f, G)$.

This completes the description of the analysis of the arithmetic problem parts. When the outer loop of the algorithm terminates we bit-blast the instance, including the learned constraints and hand it over to a standard SAT solver. Note, that we also bit-blast the arithmetic problem parts. The learned constraints guide the solver in searching a counterexamples for satisfiable instances and speed up the proof of unsatisfiability otherwise. In the next section

we present some preliminary experimental results that demonstrate the effectiveness of this strategy.

5 Experimental Results

In order to evaluate our solver STABLE we conducted a series of experiments. At first, we examined the applicability of the algebraic proof techniques in cases where ABL information is available in the arithmetic problem parts of the SMT instance. In order to demonstrate scalability of the approach we ran the solver on a set of scalable benchmarks with both word and bit-level arithmetic components.

In a second series of experiments, we evaluated the interplay of our ABL extraction techniques with the algebraic engine. These experiments used instances originating from interval property checking. The properties check generated HDL code of a commercial module generator for functional correctness. The instances contain arithmetic problem parts that are partially specified below the arithmetic bit level using the boolean constraints of the logic.

5.1 Evaluation of the algebraic techniques

We present preliminary results comparing our most recent of STABLE with current releases of Spear [9], Boolector [8] and MathSAT [6]. Note that earlier versions of these solvers won the SMT competitions of 2007, 2008 and 2009 in the category QF-BV. Finally, we also ran bit-blasted versions of the instances on the SAT solver PrecoSAT [26] that we use as our base SAT solver.

The benchmark suite consists of a combinational multiplier with Booth-encoded partial products (mult_ub) where the Booth encoder is specified at the gate level and the addition network is specified at the ABL. Furthermore, a sequential implementation for the multiplication of four values with a single shared word-level multiplier (shared_mult) is included.

Table 2 summarizes the results of these experiments. The table is organized as follows. Columns one contains the name of the instance. The remaining columns show the CPU times required by the respective solver to complete the proof. All experiments were carried out on an Intel Xeon CPU E5420 2,50 GHz 32 GB RAM running Linux with a time-out limit (TO) of 1000 s.

The presented results of the experiments show that the proposed models and algorithms are adequate to solve verification problems of industrial size.

5.2 Evaluation of the ABL extraction technique

In this section we present data on the experimental evaluation of the extraction technique for ABL information presented in Section 3. The instances considered in this experiment are property checking problems that check the HDL output of a commercial module generator for functional correctness. We compare against the same solvers as in subsection 5.1. The benchmark set consists of

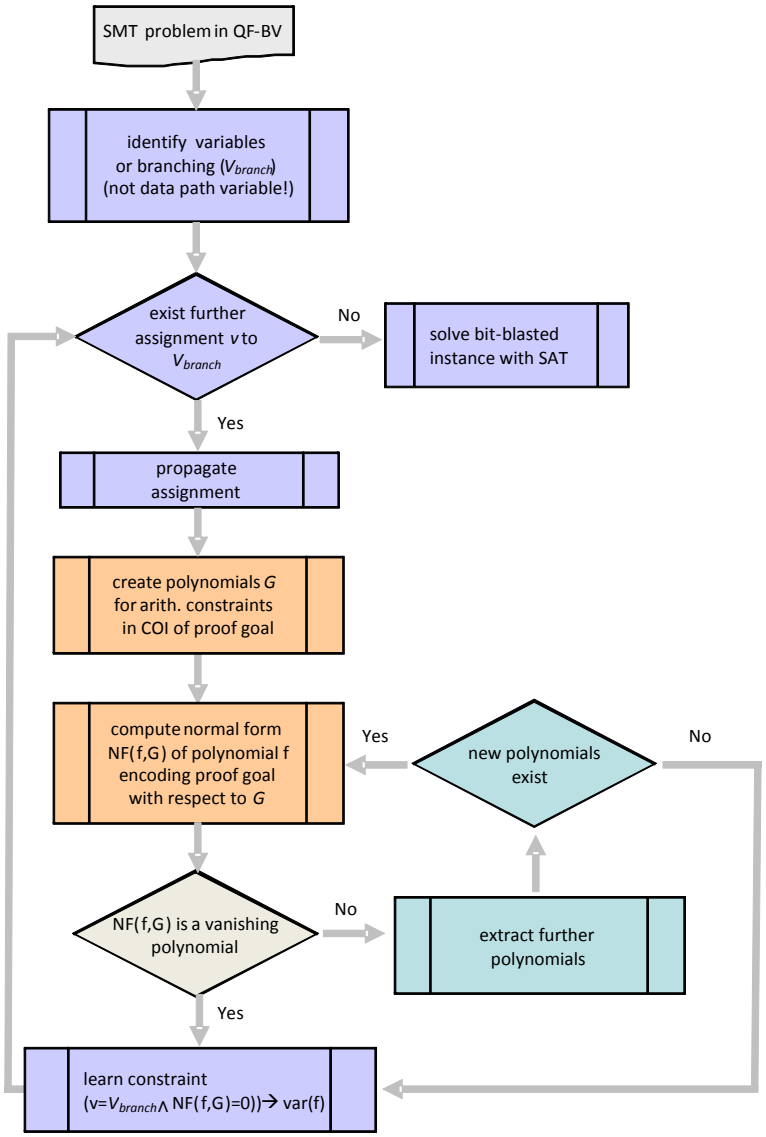

Fig. 3. High-level flow chart of the solver STABLE

Table 2. CPU-times(s) of scalability experiments

	PrecoSAT 236	Spear	Boolector 1,2	MathSAT 4,3	STABLE
shared_4	0,1	0,2	0,1	0,1	0,2
shared_8	TO	64,8	TO	TO	1,2
shared_12	TO	TO	TO	TO	5,6
shared_16	TO	TO	TO	TO	22
shared_20	TO	TO	TO	TO	81
shared_23	TO	TO	TO	TO	TO
mult_ub_4x4	0	0,02	0	0,01	0,1
mult_ub_8x8	17,8	21,47	12	31	0,3
mult_ub_16x16	TO	TO	TO	TO	1
mult_ub_32x32	TO	TO	TO	TO	6
mult_ub_64x64	TO	TO	TO	TO	52

about 1040 instances. Every instance includes a Booth-encoded multiplier with custom-designed components. These components are specified at the gate level. Table 3 presents a representative subset of the benchmark set and is organized in the same manner as Table 2. The names of the instances indicate the bit-width of the operands and whether signed or unsigned arithmetic operations are performed.

Table 3. CPU-times(s) on industrial benchmarks

	PrecoSAT 236	Spear	Boolector 1,2	MathSAT 4,3	STABLE
unsigned_4x4	0	0,02	0	0,07	0,08
unsigned_8x8	15,6	64,8	17,9	17,8	0,36
unsigned_16x16	TO	TO	TO	TO	2,2
unsigned_23x23	TO	TO	TO	TO	5,7
unsigned_32x32	TO	TO	TO	TO	10,2
unsigned_64x64	TO	TO	TO	TO	167
signed_4x4	0	0,02	0	0,06	0,1
signed_8x8	17,4	64,1	15,6	25,2	0,3
signed_16x16	TO	TO	TO	TO	2,1
signed_23x23	TO	TO	TO	TO	5,7
signed_32x32	TO	TO	TO	TO	9,4
signed_64x64	TO	TO	TO	TO	163

The results again indicate that STABLE is the only solver that can handle problems of relevant size for industrial applications. STABLE proved all 1040 instances successfully in 48,5 min.

6 Conclusion

In this paper, we describe recently developed SMT solver technology for formal arithmetic circuit verification. In particular, we introduce our SMT solver STABLE for the quantifier-free logic over bit-vectors (QF-BV). Problem parts specified at the arithmetic bit level are modeled using polynomials over finite rings $\mathbb{Z}/\langle 2^n \rangle$. We obtain variety subset problems that are equivalent to the original decision problem. The generated sets of polynomials, by construction, form a Gröbner basis of their generated ideal with respect to monomial orderings derived from the topological ordering of design signals.

This is the key for an efficient solution of the variety subset problems by normal form computation for the polynomials encoding the proof goals.

Custom-designed components with hand-crafted optimizations conducted at the pure gate level may result in incomplete arithmetic bit level descriptions for the data paths under verification. We provide an extraction technique based on local Reed-Muller forms that smoothly integrates with the normal form computation approach used in our arithmetic prover.

We demonstrate the effectiveness of the proposed approaches for industrial problems originating from the formal verification of arithmetic designs. These problems are beyond the capacity limits of today's formal property checkers.

References

1. Nguyen, M.D., Thalmaier, M., Wedler, M., Bormann, J., Stoffel, D., Kunz, W.: Unbounded protocol compliance verification using interval property checking with invariants. IEEE Transactions on Computer-Aided Design **27** (2008) 2068–2082
2. Onespin Solutions GmbH: (OneSpin 360MV. `www.onespin-solutions.com`)
3. Jain, H., Kroening, D., Sharygina, N., Clarke, E.M.: Word-level predicate-abstraction and refinement techniques for verifying RTL Verilog. IEEE Transactions on Computer-Aided Design **27** (2008) 366–379
4. Seshia, S.A., Lahiri, S.K., Bryant, R.E.: A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In: Proc. International Design Automation Conference. (2003)
5. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Proc. International Conference Computer Aided Verification (CAV). Volume 4144 of LNCS., Springer-Verlag (2006) 81–94
6. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Hanna, Z., Nadel, A., Palti, A., Sebastiani, R.: A lazy and layered smt($\{BV\}$) solver for hard industrial verification problems. In: CAV. (2007) 547–560
7. de Moura, L.M., Bjoerner, N.: Efficient e-matching for smt solvers. In Pfenning, F., ed.: CADE. Volume 4603 of Lecture Notes in Computer Science., Springer (2007) 183–198
8. Brummayer, R., Biere, A.: Boolector: An efficient SMT solver for bit-vectors and arrays. In: Proc. Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). (2009)
9. Babić, D., Hutter, F.: Spear Theorem Prover. In: Proc. of the SAT 2008 Race. (2008)

10. Vasudevan, S., Viswanath, V., Sumners, R.W., A.Abraham, J.: Automatic verification of arithmetic circuits in rtl using stepwise refinement of term rewriting systems. IEEE Transactions on Computers **56** (2007) 1401–1414
11. Stoffel, D., Kunz, W.: Equivalence checking of arithmetic circuits on the arithmetic bit level. IEEE Transactions on Computer-Aided Design **23** (2004) 586–597
12. Sarbishei, O., Tabandeh, M., Alizadeh, B., Fujita, M.: A formal approach for debugging arithmetic circuits. IEEE Transactions on Computer-Aided Design **28** (2009) 742–754
13. Shekhar, N., Kalla, P., Enescu, F., Gopalakrishnan, S.: Equivalence verification of polynomial datapaths with fixed-size bit-vectors using finite ring algebra. In: Proc. International Conference on Computer-Aided Design (ICCAD). (2005)
14. Shekhar, N., Kalla, P., Enescu, F.: Equivalence verification of arithmetic datapath with multiple word-length operands. In: Proc. International Conference on Design, Automation and Test in Europe (DATE). (2006)
15. Shekhar, N., Kalla, P., Enescu, F.: Equivalence verification of polynomial datapaths using ideal membership testing. IEEE Transactions on Computer-Aided Design **26** (2007) 1320–1330
16. Brickenstein, M., Dreyer, A.: PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. Journal of Symbolic Computation **44** (2009) 1326–1345 Effective Methods in Algebraic Geometry.
17. Brickenstein, M.: Boolean Gröbner bases – Theory, Algorithms and Applications. PhD thesis, University of Kaiserslautern, Germany (2010) to appear.
18. Watanabe, Y., Homma, N., Aoki, T., Higuchi, T.: Application of symbolic computer algebra to arithmetic circuit verification. In: Proc. International Conference on Computer Design (ICCD). (2007) 25 – 32
19. Wedler, M., Stoffel, D., Brinkmann, R., Kunz, W.: A normalization method for arithmetic data-path verification. IEEE Transactions on Computer-Aided Design **26** (2007) 1909–1922
20. Wienand, O., Wedler, M., Greuel, G.M., Stoffel, D., Kunz, W.: An algebraic approach for proving data correctness in arithmetic data paths. In: Proc. International Conference Computer Aided Verification (CAV), Princeton, NJ, USA (2008) 473–486
21. Greuel, G.M., Pfister, G.: A SINGULAR Introduction to Commutative Algebra. 2nd edn. Springer Verlag, Berlin, Heidelberg, New York (2007) 705 pages.
22. Adams, W., Loustaunau, P.: An introduction to Gröbner bases. (Graduate studies in mathematics) AMS (2003)
23. Greuel, G.M., Seelisch, F., Wienand, O.: The groebner basis of the ideal of vanishing polynomials. Journal of Symbolic Computation (To be published 2010) 14
24. Brickenstein, M., Dreyer, A., Greuel, G.M., Wedler, M., Wienand, O.: New developments in the theory of Gröbner bases and applications to formal verification. Journal of Pure and Applied Algebra **213** (2009) 1612–1635
25. Pavlenko, E., Wedler, M., Stoffel, D., Wienand, O., Karibaev, E., Kunz:, W.: Modeling of custom-designed arithmetic components in ABL normalization. In: Proc. Forum on Specification & Design Languages(FDL), Stuttgart, Germany (2008)
26. Biere, A.: PrecoSAT 236 (2009)