# An Algebraic Approach for Proving Data Correctness in Arithmetic Data Paths

Oliver Wienand, Markus Wedler, Dominik Stoffel, Wolfgang Kunz, and Gert-Martin Greuel

University of Kaiserslautern/Germany
wedler@eit.uni-kl.de

**Abstract.** This paper proposes a new approach for proving arithmetic correctness of data paths in System-on-Chip modules. It complements existing techniques which are, for reasons of complexity, restricted to verifying only the control behavior. The circuit is modeled at the arithmetic bit level (ABL) so that our approach is well adapted to current industrial design styles for high performance data paths. Normalization at the ABL is combined with the techniques of computer algebra. We compute normal forms with respect to Gröbner bases over rings $\mathbb{Z}/\langle 2^n \rangle$. Our approach proves tractable for industrial data path designs where standard property checking techniques fail.

## 1 Introduction

Property checking has become well-established in modern design flows for Systems-on-Chip (SoCs). Its main application domain is ensuring the correctness of the individual SoC blocks. 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. Therefore, in recent years, a lot of effort has been made to develop sophisticated methodologies and tools for formal module verification based on property checking. Today, formal property checking can handle almost all types of modules that can be found in today's SoCs. Nonetheless, a few pathological cases remain that sometimes limit the application of property checking in industrial practice. In particular, data paths are often a challenge for formal techniques, especially, if not only the correctness of the control flow but also correctness of the data is 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) or binary decision diagrams (BDDs) to handle arithmetic functions. Especially 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 motivated the research community to investigate alternative proof methods with focus on arithmetic.

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 473–486, 2008.

In case the validity of a property can be proven without consideration of the exact functionality of the data path, abstraction and refinement techniques have shown superiority over pure Boolean SAT techniques. A survey on these techniques can be found in [1]. However, for properties that depend on the exact functionality of the datapath a suitable abstraction is not likely to be found.

Another 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 for such a theory is the theory of equality with uninterpreted functions used in UCLID [2]. In case the problem at hand really depends on the exact functionality of a datapath, as is typically the case, most SMT solvers resort to bit blasting [1] for the corresponding problem parts. In this case SMT solvers show the same performance limitations as pure SAT solvers as soon as these datapaths include multiplication operations. The decision problems in RTL-property checking could be expressed as SAT problems for formulas of the quantifier free logic (QF-BV) and in principle be solved using solvers such as Yices [3], MathSat[4], Z3 [5] or Spear [6]. For sophisticated datapath implementations involving multiplication, however, our experience is that the problems are still beyond the capacity of such solvers.

Recently, techniques from symbolic computer algebra have entered the verification arena. The authors of [7] present a procedure to determine whether a multivariate polynomial with fixed word length operands is vanishing. By this means a comparison of polynomial representations for bit vector functions is feasible. This procedure is extended towards multiple word length operands in [8,9]. However, both approaches require a word-level representation of the datapaths under comparison. This limits their applicability in RTL property checking. Due to performance and area requirements RTL designers typically design specialized arithmetic components. These components are often designed using bit level arithmetic circuitry to build addition trees and partial products. The smallest entities in an addition tree can be described using half and full adders in general. An approach for verification of such bit level implementations using Gröbner basis theory over fields is reported in [10]. 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 CSA adder, fulfills its local specification, the polynomial representation is used to verify the block in the next level of the hierarchy. However, as 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 the case.

A heuristic approach to exploit the availability of arithmetic bit level (ABL) information in RTL designs has been reported in [11]. In this work a data structure called ABL description for representation of addition networks and bitwise multiplication is transformed into a reduced normal form. By canceling out common addends from addition networks in the fanin of a comparator the normalization approach relieves the SAT solver from reasoning in structurally different implementations for the same arithmetic function.

In order to overcome the limitations of [10] we use computer algebra algorithms for rings $\mathbb{Z}/\langle 2^N \rangle$ to solve decision problems at the arithmetic bit level. This extends the normalization approach of [11] with a clean and well-understood mathematical foundation. We show that an ABL description [11] can directly be transformed into a set of equivalent *variety subset problems*. We exploit the observation that 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 with special properties. This allows to solve the variety subset problem and hence decide problems at the arithmetic bit level.

The remainder of the paper is organized as follows: Section 2 briefly reviews the notion of an ABL description and describes how such a description can be generated given a design under verification and a property. Section 3 details the mathematical modeling for decision problems at the ABL. The proposed techniques are evaluated by experiments summarized in Section 4. Finally, Section 5 concludes the paper.

## 2 ABL Description

Arithmetic bit level (ABL) descriptions as introduced in [11] have proven to be useful for modeling the arithmetic parts of a property checking instance. In this section we briefly review this notion as far as it is required for this paper. We use the following notations:

- For $a \in \mathbb{Z}$, $b > 0$ the remainder, $a \mod b$, of the integer division $a/b$ denotes the smallest $k \geq 0$ with $k = a - mb$ for some $m \in \mathbb{Z}$.
- For $n > 0$ and $a \in \mathbb{Z}$ the uniquely determined bit vector $(a_{n-1}, \ldots, a_0)$ with $a \mod 2^n = \sum_{i=0}^{n-1} 2^i a_i$ is denoted as $\langle a, n \rangle = (a_{n-1}, \ldots, a_0)$, i.e., $\langle a, n \rangle$ is the $n$-bit binary unsigned integer representation of $a$.
- $\mathbb{B} = \{0, 1\} \subset \mathbb{Z}$ denotes the Boolean space.

The combinatorial transition function of an RTL circuit design is usually modeled by a directed acyclic graph where the vertices are labeled with bit vector functions. It is common practice to translate verification problems for RTL circuits into such bit vector netlists with a single output indicating whether, e.g., a certain property holds for a design. For the arithmetic problem parts we extract an *ABL description* from this netlist. This description again is a directed acyclic graph where the vertices can be of type "partial product generator", "addition network" or "comparator". These vertex types are defined as follows:

**Definition 1.** *Let $n, m \in \mathbb{N}$, $w : \{0, \ldots, m\} \to \mathbb{Z}$ and $c \in \mathbb{Z}$. The bit vector function $r : \mathbb{B}^m \to \mathbb{B}^n$ with*

$$r(x_1, \ldots, x_m) = \langle (c + \sum_{i=0}^{m-1} w(i) \cdot x_i), n \rangle$$

*is called* addition network with addend set $A = \{x_1, \ldots, x_m\}$. *$n$ is called* result width, *$c$ is called* constant offset *of the network and $w(i)$ is called* weight of the addend $x_i$.

*The bit vector function* $pp : \mathbb{B}^n \times \mathbb{B}^m \to \mathbb{B}^{nm}$ *with*

$$pp(x_1, \ldots, x_n, y_1, \ldots, y_m) = (x_i \cdot y_j | i = 1, \ldots, n \text{ and } j = 1, \ldots, m)$$

*is called* partial product generator.

*Every bit vector function* $cmp : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}$ *with*

$$cmp(< x + k, n >, < y + k, n >) = cmp(< x, n >, < y, n >)$$

*for all* $k \in \mathbb{Z}$ *is called* comparator.

Partial product generators model bit-wise multiplication and comparators model comparison of bit vectors. Bit level addition units like *half adders (HA)* or *full adders (FA)* are modeled as addition networks. By construction, addition networks can be used to model any addition circuit ranging from *HAs* and *FAs* up to the entire addition scheme of a multiplier or a multiply-accumulate unit. This is true for both signed and unsigned arithmetic.

*Example 1.* An signed $2 \times 2$-bit multiplier can be modeled with the partial product generator

$$pp(x_0, x_1, y_0, y_1) = (x_0y_0, x_1y_0, x_0y_1, x_1y_1)$$

and the addition network

$$r(p_{0,0}, p_{1,0}, p_{0,1}, p_{1,1}) = \langle p_{0,0} - 2p_{1,0} - 2p_{0,1} + 4p_{1,1}, 4 \rangle$$

A simple bit-level implementation of this multiplier may implement the addition network using a fulladder and two halfadders. They can be modeled by the addition networks $fa(a, b, c) =< a + b + c, 2 >$ and $ha(a, b) =< a + b, 2 >$, respectively.

For reasons of space we omit the formal definition of *ABL descriptions* as a DAG. The interested reader is referred to [11]. Basically, the nodes of the graph are labelled with their vertex type and the edges describe the interconnections between them. Here, we explain this concept by continuing Example 1.

*Example 2.* The ABL description for the comparison of the bit level multiplier implementation discussed in Example 1 against its word level specification is depicted in Figure 1.

The vertices of this graph are labeled with the bit vector function defined in the previous example. The edges $(v, v')$ are labeled with bit vectors that propagate the result of $v$ to the inputs of $v'$. In other words, the variables are defined by the following equations:

- $(p_0, p_1, p_2, p_3) = pp(x_0, x_1, y_0, y_1) = (x_0y_0, x_1y_0, x_0y_1, x_1y_1)$
- $(z_0, z_1, z_2, z_3) = r(p_0, p_1, p_2, p_4) = \langle p_0 - 2p_1 - 2p_2 + 4p_3, 4 \rangle$

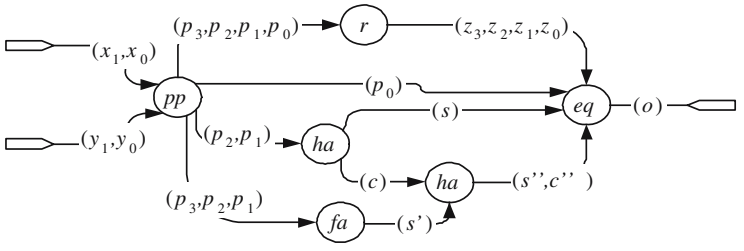


**Fig. 1.** ABL description for Example 1

- $(s, c) = ha(p_1, p_2) = \langle p_1 + p_2, 2\rangle$
- $(s', c') = fa(p_1, p_2, p_3) = \langle p_1 + p_2 + p_3, 2\rangle$
- $(s'', c'') = ha(c, s') = \langle c + s', 2\rangle$
- $(o) = eq((z_0, z_1, z_2, z_3), (p_0, s, s'', c'')) = ((z_0, z_1, z_2, z_3) == (p_0, s, s'', c''))$

This example illustrates that ABL descriptions may contain structurally dissimilar representations for one and the same arithmetic function. To simplify the comparison of such representations a heuristic ad-hoc algorithm called *ABL normalization* was proposed in [11]. This algorithm performs a series of local equivalence transformations on the ABL description that are based on the commutative and distributive laws.

However, in the next section we will describe how to obtain a variety subset problem that is equivalent to the decision problem resulting from the comparison of such ABL representations. This paves the way for the application of generic computer algebra algorithms for which efficient implementations are available.

## 3 Mathematical Background

Application of computer algebra techniques to ABL verification problems requires ABL components to be modeled by polynomials over a unique ring. Due to the operation mod used to specify ABL components, the ring $\mathbb{Z}/2^n$ seems to be the natural choice. However, the mapping of ABL descriptions on sets of polynomials $G \subset \mathbb{Z}/2^n[X]$ over such a ring is not trivial and will be detailed in this section. The key observation is that the constructed set $G$ is a Gröbner basis of the generated ideal $I = \langle G\rangle$. This makes the proposed approach computational feasible.

We start with a set of equations $G_j, j = 1, \ldots, m$ given by polynomials $f_j \in \mathbb{Z}[X]$, $X$ a finite set of variables, which are of the form

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

For the variables $r_i^{(j)}, a_k^{(l)} \in X$ in this equation we assume $r_i^{(j)} \neq a_k^{(l)}$ for $1 \leq l \leq j$ and all $i, k$. We call the variables $a_i^{(j)}$ *inputs* and $r_i^{(j)}$ *outputs* of $G_j$.

Note that the equations $G_j$ can be easily generated from the vertices of an ABL description and that the condition $r_i^{(j)} \neq a_k^{(l)}$ is fulfilled as the ABL description is acyclic by definition. For illustration we give a few examples.

*Example 3.* The partial products of a non-Booth-encoded $n \times m$ multiplier can be modeled by the polynomial equations

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

*Example 4.* 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

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

*Example 5.* A $k$-bit adder with inputs $a = (a_0, \ldots, a_{k-1})$ and $b = (b_i)$ and result $r = (r_i)$ is modeled by

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

For every proof goal, we obtain an additional polynomial $g$ depending on a subset of variables $\{a_1, \ldots, a_t\} \subset X$ and need to check whether

$$g(a_1, \ldots, a_t) = 0 \mod 2^n$$

for all solutions of the set of equations $\{G_j\}$.

*Example 6.* A $k$-bit comparator of operands $a$ and $b$ is modeled by the polynomial

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

Denote the set of all solutions to $\{G_j\}$ 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 $N$, which is necessary in order to apply computer algebra. To solve the problem we construct a Gröbner basis and then use normal form computations with respect to this basis.

For the reader's convenience we recall some basic facts about Gröbner basis theory (cf. [12,13]). We need a *monomial ordering* $<$, i.e., a well ordering on the set of monomials s.t. 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 $\mathbb{Z}/2^N$. 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 $c_i$ coefficients $\neq 0$ and $m_i$ monomials s.t. $m_1 > m_2 > \cdots > m_r$. The largest term plays a special role and we call $\mathrm{LM}(f) := m_1$ resp. $\mathrm{LC}(f) := c_1$ resp. $\mathrm{LT}(f) := c_1 m_1$ the *leading monomial* resp. the *leading coefficient* resp. the *leading term* of $f$.

Let $G \subset \mathbb{Z}/2^N[X]$ be a finite set of polynomials and $f \in \mathbb{Z}/2^N[X]$. If $cm$ is any (non-zero) term of $f$ and if $cm$ is divisible by the leading term of an element $h \in G$ we say that $f$ is reducible to $f' := f - (cm/LT(h)) \cdot h$ and write $f \underset{h}{\rightarrow} f'$. The transitive and reflexive 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 $f$ w.r.t. $G$. This notion is, however, only useful if $G$ is a Gröbner basis. In order to define a Gröbner basis we need the ideal $I = \langle G \rangle := \{\sum_{h \in G} f_h h | f_h \in \mathbb{Z}/2^N[X]\}$ generated by an arbitrary set $G$ of polynomials. Note that for the set of solutions we have $V(I) = V(G)$ for any set of generators $G$. A set of generators $G$ is called a (strong) Gröbner basis (of $I$) if $f \overset{*}{\underset{G}{\rightarrow}} 0$ for all $f \in I$. If $G$ is a Gröbner basis then the normal form of any element $g \in \mathbb{Z}/2^N[X]$ is essentially unique and equal to 0 if and only if $f \in \langle G \rangle$.

### 3.1 Problem Formulation over a Single Ring

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 3.2. 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 [14] representing the same polynomial function $(\mathbb{Z}/2^t)^{m_j} \rightarrow \mathbb{Z}/2^t$ 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 further course of action 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 7.* Suppose the $n$ bit final adder of a multiply/accumulate unit is reused for computation of an $m$-bit addition ($m < n$). In a property checking instance for this addition only the lowermost $m$ bits of the adder take influence on the arithmetic result. By the above construction we only instantiate the equations

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

for $t < m$.

So far the equations $G_j^{(t)}$ use the operation $\bmod 2^t$ for different $t$, that is, we work over different rings $\mathbb{Z}/2^t$ and none is contained in the 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.

Let $N := n + \max\{n_j \mid j = 1, \ldots, m\}$ with $n_k, n, m$ as above. We want to transform every equation into an element of the polynomial ring over $\mathbb{Z}/2^N$. To achieve this, we introduce new variables $s_t^{(j)}$ (called *slack variables*) and consider the polynomials

$$\widetilde{G}_j^{(t)} := \sum_{i=0}^{t-1} 2^i r_i^{(j)} - f_j^{(t)} \left(a_1^{(j)}, a_2^{(j)}, \ldots, a_{m_j}^{(j)}\right) - 2^t s_t^{(j)}.$$

The set of common roots for the $\widetilde{G}_j^{(t)}$ projected on the variables in $S$ corresponds to $V(\{G_j\})$. We can omit some of the extra variables $s_t^{(j)}$ if we know that $0 \leq f_j^{(t)} \leq 2^t - 1$ holds over $\mathbb{Z}$. If this condition cannot be guaranteed and we need to know the exact value of $s_t^{(j)}$ during the computation we can replace $s_t^{(j)}$ by a polynomial in the variables $a_1^{(j)}, a_2^{(j)}, \ldots, a_{m_j}^{(j)}$, i.e., a subset of the inputs of $G_j$. For example, the polynomial modeling a half adder $r_0 - a_0 - a_1 + 2s$ results in the polynomial $s = a_0 a_1$ for the slack variable. However, often it is better to introduce the slack variables because, in general, the polynomials for the slack variables will be very large even for small polynomials $f_j^{(t)}$.

Let $G = \{\widetilde{G}_j^{(t)} \mid j = 1, \ldots, m \text{ and } t = 1, \ldots, n_j\}$ 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:

Is $V(I) \subset V(2^{N-n} g)$, where $V(I)$ and $V(f)$ denote the set of all common roots (in $(\mathbb{Z}/2^N)^k$, where $k$ is the number of variables) of the polynomials in $I$ and the set of roots of the polynomial $2^{N-n} g$, respectively?

In the next section we will detail how to efficiently solve this problem.

### 3.2 Solving Decision Problems at the ABL

The following proposition turns out to be the key for an effective solution of the presented problem.

**Proposition 2.** *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 $l < i$.*

*Proof.* Let $<$ be a monomial order as required in the statement. We need to show that it is not possible to generate a polynomial from the polynomials in $G$ with a leading term that is not divisible by any leading term of the polynomials in $G$. It is sufficient to show (cf. [15], Theorem 30)

(1) For any two polynomials $f, g \in G$ the normal form of

$$\frac{\operatorname{lcm}(\operatorname{LT}(f), \operatorname{LT}(g))}{\operatorname{LT}(f)} f - \frac{\operatorname{lcm}(\operatorname{LT}(f), \operatorname{LT}(g))}{\operatorname{LT}(g)} g$$

with respect to $G$ is zero.

(2) For any $f \in G$ the normal form of $\frac{2^N}{\operatorname{LC}(f)} f$ is zero.

A slight generalization of the product criterion (cf. [15], Lemma 35) states that (1) is fulfilled, as our polynomials have different variables in their leading terms and these variables do not occur in any other term of the corresponding polynomials. Now let $f = G_j^{(t)}$. We obtain

$$\begin{aligned}
\operatorname{LT}\left(\frac{2^N}{\operatorname{LC}(f)} f\right) &= \operatorname{LT}\Big( \underbrace{2^{N-t+1} G_j^{(t)}}_{\|} \Big) \\
&= \operatorname{LT}\Big( \overbrace{2^{N-t+1} G_j^{(t-1)}} \Big) \\
&= 2^{N-t+1} \operatorname{LT}\left(G_j^{(t-1)}\right)
\end{aligned}$$

as $2^{N-t+1} \cdot 2^{t-1} r_{t-1}^{(j)} = 0 = 2^{N-t+1} \cdot 2^{t-1} s_{t-1}^{(j)}$ and $2^{N-t+1} \cdot 2^{t-2} r_{t-2}^{(j)} \neq 0$, and since the polynomials $f_j^{(t)}$ appearing in $\widetilde{G}_j^{(t)}$ are chosen minimal. In the first step of the normal form algorithm we will select $G_j^{(t-1)}$ and this reduces $2^{N-t+1} G_j^{(t)}$ to zero. This shows (2).

By Lemma 3 we prove that normal form computation can be used as an effective solution procedure for our problem at hand.

**Lemma 3.** *Let $G$ be a Gröbner basis of an ideal $I \subset \mathbb{Z}/2^N[\mathbf{x}]$, $\mathbf{x} = (\mathbf{x}', \mathbf{x}'')$, and $g$ a polynomial such that $h$, the normal form of $g$ w.r.t. $G$, is in $\mathbb{Z}/2^N[\mathbf{x}']$. Assume that for all $\mathbf{x}'$ there exist $\mathbf{x}''$ with $f(\mathbf{x}', \mathbf{x}'') = 0$ for all $f \in G$. Then $h$ defines the zero function if and only if $V(G) \subset V(g)$.*

*Proof.* If $h$ defines a constant zero function the set $V(h) = V(g)$ contains all points and therefore $V(G) \subset V(g)$ is trivial. Assume that for the variables $\mathbf{x}'$ of $h$ a valuation exists such that $h$ is not zero. By assumption we can extend this valuation to a valuation on all variables such that $g(\mathbf{x}) = 0, g \in G$. It follows $V(G) \not\subset V(f)$.

Let $g \in \mathbb{Z}/2^n[\mathbf{x}]$ and $h$ be the normal form of $2^{N-n} g$ with respect to $G$, which can be computed [15] 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 3.

This guarantees that the variables present in $h$ are inputs to the ABL description. Every valuation of these variables can be extended to a consistent valuation for the signals of the ABL. Further we can effectively decide whether $h$ defines the zero function for all rings $\mathbb{Z}/m$ (cf. [14]) and therefore decide the ABL problem by Lemma 3.

As already noted in Section 3.1 it is not always efficient to replace all remaining slack variables by polynomial expressions in terms of the input variables of the corresponding equations. Therefore we use special procedures for the practical computations, which we do not detail here.

**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* $g \in G'$
*Let* $\mathrm{LT}(f) = m \cdot \mathrm{LT}(g)$ *with* $m \cdot \mathrm{LC}(g) \neq 0$
$f := f - m \cdot g_i$
**end while**
**return** $f$

**Algorithm 1.** Normal form algorithm

## 4 Experimental Results

In order to evaluate the techniques presented in the previous sections we conducted a series of experiments. Except for one experiment explicitly indicated in the sequel, all experiments were carried on a machine running Suse Linux 10.3 on a Intel Core 2 Duo E6400 with 8 GB RAM.

The algorithms presented in Section 3 have been implemented within the framework of the general purpose computer algebra system Singular [16]. We used the industrial formal property checker Onespin 360 MV [17] to generate bit vector netlists for the considered verification problems. From these bit-vector netlists we extracted an arithmetic bit level description for the arithmetic parts of the decision problem and dumped out the resulting ABL description. The resulting problem file is used to generate the variety subset problem that is handed over to Singular in order to find a solution.

As a first step of the evaluation we used a number of parameterized benchmarks to evaluate the scalability of the proposed approach with respect to the

bit-width of the datapath under verification. The benchmark suite consists of two instances (distrib and commute) for word-level implementations of the functions $ab + ac$ and $(ab)c$ where commutative and distributive laws have been applied to the word level operands, a bit level implementation of an unsigned multiplier with Booth-encoded partial products (mult_ub) and a sequential implementation for the multiplication of four values with a single multiplier (shared).

We compare the performance in terms of run-time of our solution based on Singular against the normalization approach of [11], a SAT-based decision procedure based on bit blasting, and the SMT solver Spear v.2.0 for the theory of fixed-size bit-vector functions (QF-BV). Note that an earlier version of Spear showed the best performance in this category on the 2007 SMT competition.

**Table 1.** CPU-times(s) of scalability experiments

| Instance | Bit-Width | Normalizer | SAT | SMT | Singular |
|---|---|---|---|---|---|
| distrib | 4 | 0,01 | 0,28 | 0,40 | 0,69 |
| distrib | 8 | 0,03 | > 3600s | > 3600s | 0,66 |
| distrib | 16 | 0,10 | > 3600s | > 3600s | 0,97 |
| distrib | 32 | 0,81 | > 3600s | > 3600s | 2,19 |
| distrib | 64 | 14,33 | > 3600s | > 3600s | 11,30 |
| commute | 4 | 0,02 | 0,55 | 1,01 | 0,69 |
| commute | 8 | 0,08 | > 3600s | > 3600s | 0,67 |
| commute | 16 | 1,40 | > 3600s | > 3600s | 1,09 |
| commute | 32 | 57,17 | > 3600s | > 3600s | 3,56 |
| commute | 64 | 2794,67 | > 3600s | > 3600s | 26,03 |
| mult_ub | 4 | 0,02 | 0,02 | 0,15 | 0,66 |
| mult_ub | 8 | 0,13 | 41,53 | > 3600s | 0,96 |
| mult_ub | 16 | 2,21 | > 3600s | > 3600s | 3,87 |
| mult_ub | 32 | 53,55 | > 3600s | > 3600s | 79,04 |
| mult_ub | 64 | 1136,14 | > 3600s | > 3600s | > 8 GB |
| shared | 4 | 0,04 | 2,83 | 16,78 | 0,97 |
| shared | 8 | 0,46 | > 3600s | > 3600s | 0,64 |
| shared | 16 | 39,79 | > 3600s | > 3600s | 1,09 |
| shared | 32 | 2707,72 | > 3600s | > 3600s | 20,25 |

Table 1 summarizes the results of these experiments. The table is organized as follows. Columns one and two contain instance and operand bit-width of the datapath. The remaining columns show the CPU times required by the particular tool to prove the instance. In case the memory limit or timeout limit was reached this is indicated by "> 8 GB" and "> 3600", respectively.

In order to evaluate the performance of Singular with respect to other computer algebra systems we also report results for solving the generated variety subset problems with the industrial computer algebra tool Magma [18]. However, due to license restrictions, these results were obtained using another machine, namely an AMD Dual Opteron 2.2 GHz with 16 GB RAM running Linux. We re-ran the Singular problems on this machine in order to allow for comparison of

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

| Instance | Bit-Width | Singular | Magma |
|---|---|---|---|
| distrib | 16 | 1,08 | 2,33 |
| distrib | 32 | 2,70 | 15,61 |
| distrib | 64 | 14,53 | > 16 GB |
| commute | 16 | 1,35 | 5,53 |
| commute | 32 | 4,71 | 46,07 |
| commute | 64 | 38,96 | > 16 GB |
| mult_ub | 4 | 0,56 | > 3600s |
| mult_ub | 16 | 4,07 | > 3600s |
| mult_ub | 32 | 85,77 | > 3600s |
| shared | 4 | 0,46 | 1,08 |
| shared | 8 | 0,66 | 1,35 |
| shared | 16 | 1,37 | 3,09 |
| shared | 32 | 32,27 | 108,01 |

the run times. For the comparison we also increased the memory limit to 16GB. Table 2 summarizes the results for this comparison.

The presented results of the scalability experiments indicate that the proposed modeling and the proposed algorithms are adequate to solve verification problems with industrial impact. To demonstrate this we investigated a property suite originating from the verification of Infineon's Tricore 2 processor. The processor has advanced DSP features including a sophisticated integer pipeline that provides a large variety of multiply and multiply/accumulate instructions. The properties in the investigated property suite verify that every variant of these instructions causes the integer pipeline of the processor to deliver the expected arithmetic result according to the architectural manual. In order to obtain a high degree of resource sharing large portions of the datapath have been implemented at the arithmetic bit level and sophisticated control logic is used for configuration according to the executed instructions.

We used the techniques of [11] to generate the decision problems at the arithmetic bit level. All the resulting decision problems could be solved with Singular when modeled by polynomials as presented in this paper. Table 3 shows the results for a representative subset of the problem instances derived from the Tricore 2 property suite. It is organized as follows. The first column shows the commitment of the property specifying the arithmetic result of the integer instruction under verification. Columns two and three show the run-time of the normalization approach and the corresponding Singular run-time. Unless explicitly indicated all operations are considered as signed operations on the specified bit-vectors.

In essence, all our experiments show that the presented approach outperforms the ad-hoc normalization approach in terms of CPU time. Moreover, algorithms and modeling rely on a well-understood mathematical foundation which opens ample opportunities for further extensions of this framework.

However, the use of a generic computer algebra system as Singular for solving the normalization problems is paid with a price in terms of memory consumption.

**Table 3.** Results for selected Tricore 2 properties

| Datapath result | Normalizer | Singular |
|---|---|---|
| res[31:0]=op3[31:0]+(op1[31:0]*op2[31:0]); | 49,94 | 4,42 |
| res[31:0]=op3[31:0]+(op1[15:0]*op2[31:16]<<1);<br>res[63:32]=op3[63:32]+(op1[31:16]*op2[15:0]<<1); | 39,72 | 2,28 |
| res[31:0]=op3[31:0]+(op1[31:16]*op2[31:16]<<1); | 18,39 | 2,47 |
| res[15:0]=rnd16(op3[31:0]+(7FFFFFFF));<br>res[31:16]=rnd16(op3[63:32]<br>+(op1[31:16]*op2[15:0]<<1)); | 19,90 | 2,46 |
| res[63:0]=op3[63:0]+(op1[31:16]*op2[15:0]<<16)<br>-(op1[31:16]*op2[31:16]<<16) | 31,04 | 8,77 |
| res[63:0]=op3[31:0]-(op1[31:0]*op2[31:0]); | 55,19 | 20,01 |
| res[63:0]=op3[63:0]-(op1[31:16]*op2[15:0]<<16)<br>-(op1[15:0]*op2[15:0]<<16) | 27,18 | 9,64 |
| res[63:0]=op1[31:0]*op2[31:0]; (unsigned) | 57,33 | 14,73 |
| res[31:16]=rnd16(op1[31:16]*op2[31:16]); | 17,41 | 2,21 |

Except for some of the problems where the ABL description is generated from word-level problems Singular typically requires 3–8 GB of memory. This is caused by the data structures used inside Singular to represent polynomials.

These data structures are not optimized with respect to the characteristics of the problems considered here. Compared to problems typically considered in computer algebra, we consider a large number of variables, and many polynomials. On the other hand the individual polynomials have low degree and only use a small fraction of the variables. With application-specific implementations of the employed algorithms such as the normal form computation a great improvement of the memory efficiency can be obtained easily.

## 5 Conclusion and Future Work

Decision problems at the arithmetic bit level have been modeled using polynomials over rings $\mathbb{Z}/\langle 2^n \rangle$. It has been proven that the generated sets of polynomials form a Gröbner basis with respect to certain monomial orderings that can easily be determined using the topological ordering of design signals. This allows for utilization of the normal form algorithm to efficiently solve a variety subset problem that is equivalent to the original decision problem.

By this means we provide a solid mathematical foundation to the ad-hoc technique of arithmetic bit level normalization. The developed techniques have proven to be applicable to verification problems of industrial size.

As the datastructures for polynomial representation of in-the-box computer algebra systems do not exploit the typical characteristics of the generated polynomial sets, we are working on a specialized implementation of the employed algorithms that will dramatically reduce the memory consumption.

## References


1. Kroening, D., Seshia, S.A.: Formal verification at higher levels of abstraction. In: Proc. International Conference on Computer-Aided Design (ICCAD) (2007)
2. 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)
3. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
4. 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: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 547–560. Springer, Heidelberg (2007)
5. de Moura, L.M., Bjoerner, N.: Efficient e-matching for smt solvers. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 183–198. Springer, Heidelberg (2007)
6. Spear, `http://www.cs.ubc.ca/~babic/index.htm`
7. 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)
8. 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)
9. Shekhar, N., Kalla, P., Enescu, F.: Equivalence verification of polynomial datapaths using ideal membership testing. IEEE Transactions on Computer-Aided Design 26(7), 1320–1330 (2007)
10. 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), pp. 25–32 (October 2007)
11. Wedler, M., Stoffel, D., Brinkmann, R., Kunz, W.: A normalization method for arithmetic data-path verification. IEEE Transactions on Computer-Aided Design 26(11), 1909–1922 (2007)
12. Adams, W., Loustaunau, P.: An introduction to Gröbner bases. (Graduate studies in mathematics) AMS (2003)
13. Greuel, G.M., Pfister, G.: A SINGULAR Introduction to Commutative Algebra, 2nd edn., 705 pages. Springer, Heidelberg (2007)
14. Wienand, O.: The Gröbner basis of the ideal of vanishing polynomials (2007) arXiv: arXiv:0709.2978v1 [math.AC]
15. 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 (accepted for publication)
16. Greuel, G.M., Pfister, G., Schönemann, H.: Singular 3.0.4. - A Computer Algebra System for Polynomial Computations. Centre for Computer Algebra, University of Kaiserslautern (2007), `http://www.singular.uni-kl.de`
17. Onespin Solutions GmbH Munich, Germany, `www.onespin.com`
18. Bosma, W., Cannon, J., Playoust, C.: The MAGMA algebra system I: the user language. J. Symb. Comput. 24(3-4), 235–265 (1997)