STABLE: A new QF-BV SMT Solver for hard Verification Problems combining Boolean Reasoning with Computer Algebra

Evgeny Pavlenko, Markus Wedler, Dominik Stoffel, Wolfgang Kunz
Dept. of Electrical and Computer Eng., University of Kaiserslautern, Germany {pavlenko, wedler, stoffel, kunz}@eit.uni-kl.de

Alexander Dreyer
Dept. of System Analysis, Prognosis and Control
Fraunhofer ITWM, Kaiserslautern, Germany
alexander.dreyer@itwm.fraunhofer.de

Frank Seelisch, Gert-Martin Greuel
Dept. of Mathematics,
University of Kaiserslautern, Germany
{seelisch,greuel}@mathematik.uni-kl.de

***Abstract*—This paper presents a new SMT solver, *STABLE*, for formulas of the quantifier-free logic over fixed-sized bit vectors (QF-BV). The heart of *STABLE* is a computer-algebra-based engine which provides algorithms for simplifying arithmetic problems of an SMT instance prior to bit-blasting.**

As the primary application domain for STABLE we target an SMT-based property checking flow for System-on-Chip (SoC) designs. When verifying industrial data path modules we frequently encounter custom-designed arithmetic components specified at the logic level of the hardware description language being used. This results in SMT problems where arithmetic parts may include non-arithmetic constraints. STABLE includes a new technique for extracting arithmetic bit-level information for these non-arithmetic constraints. Thus, our algebraic engine can solve sub-problems related to the entire arithmetic design component.

***STABLE* was successfully evaluated in comparison with other state-of-the-art SMT solvers on a large collection of SMT formulas describing verification problems of industrial data path designs that include multiplication. In contrast to the other solvers STABLE was able to solve instances with bit-widths of up to 64 bits.**

I. Introduction

Today's IP-based design styles for Systems-on-Chip (SoC) have become one of the main drivers for innovative verification methodologies. When verifying the functionality of the overall system a correctness by integration strategy is often pursued. This, however, requires modules of extremely high design quality that can only be achieved with formal methods. Property checking techniques based on Satisfiability solving (SAT) and SAT modulo theory solving (SMT) [1], [2], [3], [4] have proved quite successful in capturing the relevant aspects of design behavior and therefore have gained more and more significance in modern System-on-Chip (SoC) design flows.

SAT solving has been an intensive research area for several decades and plays a major role in almost all modern verification tools. BerkMin [5], MiniSat [6], PrecoSAT [7] are only some examples of efficient SAT solvers. For control-intensive modules of SoC designs SAT solvers have shown to be adequate proof engines when verifying their correctness. However, data paths including complex arithmetic blocks such as multiplication still remain a bottleneck for SAT-based property checkers. In industrial practice, simulation-based techniques usually prevail over SAT or other formal approaches as soon as arithmetic circuits are under verification.

978-3-9810801-7-9/DATE11/2011©EDAA

SMT [8] has recently gained a lot of attention since it promises to extend SAT-style property checking to a wider range of applications. The SMT languages [9] (version 1.2) and [10] (version 2.0) are more expressive than plain propositional logic. An SMT problem is usually described as a logic formula in combination with expressions from a first-order background-theory. As a result, an SMT instance can preserve a lot of information available only at a high level of abstraction which is lost as soon as a conversion into a CNF-based SAT-problem is applied ("bit-blasting"). Therefore, SMT solvers can handle many verification problems which are impossible to solve by standard SAT methods alone.

The SMT community has defined numerous theories that may be of interest for certain applications. For proving functional correctness of arithmetic data paths the theory of fixed-sized bit vectors (BV) becomes a natural choice. In the remainder of this paper we always consider the quantifier free logic with fixed-sized bit vectors (QF-BV) when referring to SMT-instances. Recently, different research groups developed SMT solvers for the QF-BV category, e. g., Yices [11], Z3 [12], Spear [13], Boolector [14], MathSAT [15], simplifyingSTP [16], [17]. These tools demonstrated significantly better performance in comparison to pure SAT. However, they still lack capacity to solve unsatisfiable formulas as they are derived from arithmetic data path verification in industry.

At higher levels of abstraction, techniques from symbolic computer algebra [18] have proven to be an attractive alternative when verifying arithmetic data-path designs. The approach in [18] requires, however, that both implementation and specification can be modeled with a single multivariate polynomial. Unfortunately, in SMT instances derived from high performance data paths this is usually not the case.

In this paper, we present our new SMT solver *STABLE* for QF-BV instances. It integrates two recently developed techniques for solving hard arithmetic problems [19], [20]. When an arithmetic sub-problem of an SMT-instance is described at the arithmetic bit- or word-level we can directly apply the theory of Gröbner bases over finite rings. First, we convert the arithmetic parts of the decision problem to equivalent *variety subset problems* and then compute normal forms to solve them. If the normal form is a vanishing polynomial the corresponding proof goal is valid. In [20] it was suggested that the techniques of [18] could be applied for this check.

Contrary to [20] we now perform our computations in the quotient ring $Q := \mathbb{Z}/2^N[X]/\langle x^2 - x \;:\; x \in X\rangle$. We prove, that this allows us to omit an otherwise necessary and expensive zero function test for the normal form. This is one of the main contributions of this paper.

In *STABLE* we also employ the technique of [19]. It addresses instances where arithmetic sub-problems are specified using the complete set of bit-vector constraints including non-arithmetic Boolean constraints. Such instances are frequently derived from property checking of data paths with highly optimized custom-designed components. In order to apply the algebraic Gröbner approach we extract their arithmetic information.

This paper is organized as follows: Section II shows how arithmetic sub-problems of an SMT instance can be solved using computer algebra techniques. In this section, we assume the arithmetic problem parts to be specified at the arithmetic bit-level (ABL) or word-level. For cases where problem parts are specified below ABL we include an extraction technique described in Section III. Section IV illustrates our strategy for integrating the proposed techniques into STABLE. The paper ends with experimental results summarized in Section V and the conclusion in Section VI.

II. Using computer algebra to solve arithmetic subproblems of SMT instances

This section recalls the mathematical models required to solve arithmetic problem parts of an SMT decision problem in QF-BV with the algebraic techniques introduced in [20].

A. Mathematical background

In the following, we summarize some basic facts about Gröbner basis theory of polynomials over a finite ring $\mathbb{Z}/\langle 2^n\rangle$, cf. [21], [22]. We need 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 $\mathbb{Z}/\langle 2^n\rangle$. We can write any polynomial $f \neq 0$ as a finite sum of terms, $f = c_1m_1 + \cdots + c_rm_r$ with coefficients $c_i \neq 0$ and monomials $m_1 > m_2 > \cdots > m_r$. The largest term c_1m_1 plays a special role: we call $LM(f) := m_1$ the *leading monomial*, $LC(f) := c_1$ the *leading coefficient* and $LT(f) := c_1m_1$ the *leading term* of f.

Let X be a finite set of variables, $G \subset \mathbb{Z}/\langle 2^n\rangle[X]$ a finite polynomial set and $f \in \mathbb{Z}/\langle 2^n\rangle[X]$. If $LM(f)$ is a linear combination of leading terms of elements of G, i. e. , $LM(f) = \sum_{g\in G} p_g \cdot LM(g)$ for monomials p_g, then f is reducible to $f' := f - \sum_{g\in G} p_g \cdot g$ and hence we write $f \underset{G}{\longrightarrow} f'$. The transitive closure of the relation $\underset{G}{\longrightarrow}$ is denoted by $\overset{*}{\underset{G}{\longrightarrow}}$. If $f \overset{*}{\underset{G}{\longrightarrow}} g$ and if g is not reducible by G we call g a *normal form* of f with respect to G. There are effective algorithms for computing normal forms in polynomial rings of the form $\mathbb{Z}/\langle 2^n\rangle[X]$, cf. [21], [23].

The notion of a normal form is only useful if G is a Gröbner basis. Any finite set G of polynomials generates an ideal $I = \langle G\rangle := \{\sum_{g\in G} f_g g \mid f_g \in \mathbb{Z}/\langle 2^n\rangle[X]\}$. Such a set G is called a Gröbner basis of I, if for each $f \in \mathbb{Z}/\langle 2^n\rangle[X]$, $f \overset{*}{\underset{G}{\longrightarrow}} 0$ if and only if $f \in I$.

With this basic knowledge of Gröbner basis theory we can now study how to model arithmetic problem parts of an SMT instance by polynomials over a finite ring $\mathbb{Z}/\langle 2^n\rangle$.

B. Algebraic modeling of arithmetic decision problems

We may assume that the SMT decision problem is represented as an acyclic netlist of bit vector functions. The arithmetic components G_j of this netlist may have multiple output bit-widths that are denoted by n_j in the sequel. In [20] each of these components is modeled by n_j polynomials over the ring $R = \mathbb{Z}/2^N$ with appropriate N. Initially the size N of the ring is chosen heuristically to be

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

where n is the bit-width of the comparison constraint in the proof goal, m the component number in the netlist and n_j the bit-width of the j-th component. However, this initial value for N is just a heuristic choice that turned out to be sufficient for many practical problem instances. During computations our current implementation detects cases where a larger ring is required and automatically moves to a sufficiently larger ring with size $N' > N$.

For each arithmetic component in the cone of influence of a proof goal we generate n_j polynomials of the form

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

with $t = 1, \ldots, n_j$. The variables $a_i^{(j)}$ correspond to the inputs and the variables $r_i^{(j)}$ correspond to the outputs of the j-th component. The so called slack variables $s_t^{(j)}$ are newly introduced artificial variables that are used to model the modulo semantics of the arithmetic components in bit-vector netlists. The polynomials $f_j^{(t)}$ are defined as the polynomials with minimal coefficients representing the polynomial function $(\mathbb{Z}/2^t)^{m_j} \to \mathbb{Z}/2^t$ that specifies the lower most output bits $r_i^{(j)}, i = 0, \ldots, t-1$ of the arithmetic component G_j.

Since the netlist is acyclic we have $r_i^{(j)} \neq a_k^{(l)}$. We illustrate this model with an unsigned k-bit adder with input variables $a = (a_i \mid 0 \leq i < k)$ and $b = (b_i \mid 0 \leq i < k)$ and result $r = (r_i \mid 0 \leq i < k)$. This adder can be modeled by k polynomials $\tilde{G}_+^{(t)}$, $t = 1, \ldots, k$ of Equation (2).

$$\tilde{G}_+^{(t)} : \sum_{i=0}^{t-1} 2^i r_i - \sum_{i=0}^{t-1} 2^i (a_i + b_i) - 2^t s_t. \quad (2)$$

Such polynomials are sufficient to model word-level arithmetic components like signed-/unsigned multipliers or adders. Likewise bit-level arithmetic components such as full-adders, half-adders and bit-wise products can be modeled in the same

formalism. A full-adder with inputs x_0, x_1, x_2 and outputs c, s for carry and sum can, e. g., be modeled by the polynomials

$$\begin{array}{l} \tilde{G}_{fa}^{(2)} : 2c + s - (x_0 + x_1 + x_2) + 4s_2 \\ \tilde{G}_{fa}^{(1)} : s - (x_0 + x_1 + x_2) + 2s_1. \end{array} \quad (3)$$

In the polynomials $\tilde{G}_j^{(t)}$ some of the slack variables $s_t^{(j)}$ can be omitted if we know that $0 \leq f_j^{(t)} \leq 2^t - 1$ holds over $\mathbb{Z}$. For example the slack variable s_2 in Equation 3 can be omitted. As illustrated, the above algebraic model utilizes word-level information where available and is able to handle bit-level arithmetic information where necessary as well.

In order to analyze proof goals with algebraic methods we need the notion of a variety. The variety $V(F)$ of a polynomial set $F \subset R[x_1, \ldots, x_n]$, R a commutative ring with 1, is the set of tuples $(r_1, \ldots, r_n) \in R^n$ where all $f \in F$ simultaneously vanish.

Let X be the set of non slack variables in $\{\tilde{G}_j^{(t)}\}$. A proof goal is modeled by a polynomial g in $\{a_1, \ldots, a_t\} \subset X$. It vanishes if and only if the proof goal is fulfilled, i. e.,

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

holds for all tuples in the variety $V(\{\tilde{G}_j\})$. Note that n used for the modulo in the above equation depends on the bit width of the comparison constraint in the underlying problem instance. For example, an n-bit equality comparison of operands a and b is modeled by polynomial (4).

$$g = \sum_{i=0}^{n-1} 2^i (a_i - b_i)\,. \quad (4)$$

We prove that this vanishes modulo 2^n for all tuples in the variety $V(\{\tilde{G}_j\})$. This leads to the *variety subset problem*:

$$V(\{\tilde{G}_j\}) \subset V(2^{N-n} g)\,. \quad (5)$$

Following, we solve this for $R = \mathbb{Z}/2^N$; it replaces the well-known *ideal membership problem* of the field case.

C. Solving arithmetic decision problems by normal form computation

We recall how to solve the variety subset problem [20], [23]. The set $G = \{\tilde{G}_j\}$ is a Gröbner basis for such global monomial orderings, where $r_i^{(j)}$ is larger than every monomial in $a_k^{(j)}, s_t^{(j)}, r_l^{(j)}$ for all i, k, t, j and $l < i$. We solve our problem by computing a normal form $h = NF(2^{N-n} g, G)$ of our proof goal [24]. It holds if and only if h defines the zero function, since we may assume that h depends on inputs only.

In general, zero function tests over $\mathbb{Z}/2^N[X]$ are expensive [18]. To avoid them we reformulate the polynomials of Equation 1 using bit-valued variables in the quotient ring $Q := \mathbb{Z}/2^N[X]/\langle x^2 - x \,:\, x \in X\rangle$. Lemma 1 ensures that only the zero polynomial in Q defines the zero function on $Q^{|X|}$ and renders the zero function test superfluous here.

Lemma 1: Let $m, n \geq 1$ be natural numbers and $Q := \mathbb{Z}/\langle m\rangle[X]/\langle x^2 - x \,:\, x \in X\rangle$ with $X = \{x_1, \ldots, x_n\}$ be a polynomial quotient ring. Denote by $T := \{(t_1, t_2, \ldots, t_n) \mid t_i \in \{0, 1\}, 1 \leq i \leq n\}$ the set of all bit-valued inputs for polynomials in Q. If $f \in Q$ vanishes for all $t \in T$, then f is the zero polynomial.

Proof: We fix some $f \in Q$ which vanishes for all $t \in T$, and assume $f \not\equiv 0$ for a contradiction. Due to special structure of Q all terms of f are of the form $c \cdot x_{i_1} \cdots x_{i_k}$ with mutually distinct indices i_j in $\{1, 2, \ldots, n\}$. We pick a term s of f with k least, i. e. with the least number of variables. Now consider the tuple $t = (t_1, t_2, \ldots, t_n) \in T$ with $t_j = 1$ if x_j appears in s, and $t_j = 0$ otherwise. We claim $f(t) \neq 0$, yielding the desired contradiction.
Let s' denote any other term of f, i. e., any term of $f - s$. By construction, it is clear that s' mentions at least one variable which is not present in s. But then $s'(t) = 0$, hence $f(t) = s(t) + (f - s)(t) = s(t) = LC(s) \neq 0$. □

Note that for computations in Q, efficient DAG-based data structures are available.

III. Extracting arithmetic bit-level information

In this section we adopt an extraction technique developed in [19]. It derives compact polynomial representations for those parts of the algebraic proof goals that are defined using Boolean constraints of the bit-vector logic. In the sequel, we consider a connected sub-netlist of the global bit-vector netlist for the entire SMT instance. We assume that the considered sub-netlist is described by non-arithmetic bit-vector functions, i. e. in terms of simple Boolean constraints. By a topological analysis we may identify the variables used as inputs and variables r_i used as outputs of this sub-netlist.

This extraction technique consists of two phases:

- Derive a set of polynomials P based on the arithmetic transform for each Boolean constraint.
- Pre-normalize the polynomials with respect to the input variables of the considered sub-netlist.

We illustrate the extraction process using a custom-designed circuit component that implements a two-bit adder as our running example. The corresponding gate netlist is presented in Figure 1.

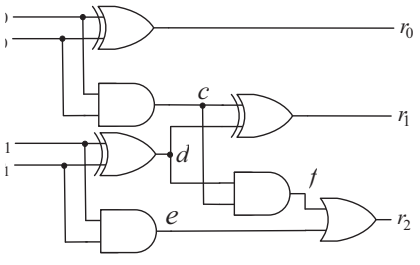

Fig. 1. Gate netlist of a two bit adder

A front-end of a standard property checker usually performs a one-to-one compilation of such a circuit structure into a collection of Boolean bit-level constraints of the bit vector logic.

The first step of the extraction yields the polynomials illustrated in Equation 6 for our running example.

$$\begin{cases} g_1 = r_0 - (a_0 + b_0 - 2a_0b_0) \\ g_2 = c - (a_0b_0) \\ g_3 = d - (a_1 + b_1 - 2a_1b_1) \\ g_4 = r_1 - (c + d - 2cd) \\ g_5 = f - (cd) \\ g_6 = e - (a_1b_1) \\ g_7 = r_2 - (e + f - ef) \end{cases} \quad (6)$$

In theory these polynomials may immediately be used in our algebraic model generated in Section II-B. However, it is already apparent from the example that this fine grained modeling of the non-arithmetic constraints in the cone of influence of an algebraic proof goal will lead to fairly large problem instances if many such problem parts exist. For example, the two-bit adder of Figure 1 may be instantiated within a large multiplier several times. This unnecessary blow-up, both in terms of the numbers of variables as well as in the number of polynomials can considerably slow down the normal form computation. Fortunately, frequently custom-designed components used within such designs have a more compact polynomial representation that can be identified by our proposed extraction technique. It relies on a pre-normalization step that will be described in the remainder of this section.

We start with the polynomial set P derived in the first phase of the extraction process. We consider the subset $O \subset P$ of polynomials that depend on the output variables r_i. Due to the monomial ordering being compatible with the topological ordering of the netlist this is easily determined by checking whether the leading term is one of the r_i.

For each polynomial $h \in O$ we compute the reduced normal form of h with respect to P. The polynomials $g \in P$ used during this computation for the reduction of the tails of the polynomials h can be determined efficiently by backward tracing in the netlist. For our running example this results in the polynomials of Equation 7.

$$\begin{cases} g'_1 = r_0 - a_0 - b_0 + 2a_0b_0 \\ g'_4 = r_1 - (a_1 + b_1 + a_0b_0 - 2a_1b_1 - \\ \qquad 2a_0a_1b_0 - 2a_0b_0b_1 + 4a_0a_1b_0b_1) \\ g'_7 = r_2 - a_1b_1 - a_0a_1b_0 - a_0b_0b_1 + 2a_0a_1b_0b_1 \end{cases} \quad (7)$$

To conclude this section it should be noted that the weighted addition $4g'_7 + 2g'_4 + g'_1$ of the above polynomials yields the compact polynomial of Equation 8.

$$4r_2 + 2r_1 + r_0 - 2(a_1 + b_1) - (a_0 + b_0) \quad (8)$$

IV. STABLE: A NEW SMT SOLVER FOR QF-BV

The new QF-BV SMT solver *STABLE* integrates the techniques presented in Section II and Section III with a standard SAT-solver as backend. The basic flow of *STABLE* is illustrated in Figure 2.

The dashed boxes indicate the interface between the computer algebra library (*GBABL*), the extraction library (*ABL extractor*) and the solver infrastructure. The solver performs certain preprocessing steps after parsing the SMT instance. We represent the instance internally as a netlist of predefined bit-vector functions. For the SMT-LIB format [9], [10] this netlist needs to be generated from a collection of constraints that are implicitly conjoined by the use of common variables. Data path verification instances often have an implicative structure $a \rightarrow c$. The assumption a may cause a lot of propagations that further simplify the instance during preprocessing. For example, the assumption of a property frequently considers a specific scenario in which only a reasonable number of configurations for the data path need to be investigated. Propagation of the assumption drastically reduces the number of iterations in the main loop of the solver that analyzes individual configurations.

Before entering this loop we determine a set of Boolean branching variables V that influence arithmetic sub-problems in c. As branching variables we consider the select inputs s used as condition in if-then-else constraints in the cone of influence of c. For branching variables V we enumerate each possible assignment $val(V)$. Note that in case of an empty set V the loop is entered exactly once. After constant propagation of the assignment $val(V)$ we analyze the arithmetic proof goal f relevant under the respective configuration of the data path with a normal form computation. Note that an instance with several arithmetic proof goals may require an iteration of these two steps for each goal. For brevity, we omitted this extra loop in the flow of Figure 2.

In order to analyze a proof goal f *STABLE* generates a set G of polynomials $G \subset Q = \mathbb{Z}/2^N[X]/\langle x^2 - x \ : \ x \in X\rangle$ modeling the arithmetic constraints in the cone of influence of f. Next the *GBABL* engine computes the normal form (NF) of f with respect to G. If this normal form is the zero polynomial we learn the validity of the proof goal f under the current data path configuration $val(V)$ and proceed with the next iteration of the main loop.

However, we also need to consider the case where the normal form of the proof goal is a non-zero polynomial. This may have several reasons. The proof goal may indeed be invalid, it may be necessary to extract certain portions of the arithmetic problem by the techniques of Section III, or Boolean propagation may be to weak to completely eliminate the control logic in the cone of influence of the proof goal.

We approach these issues as follows. First, we check if some of the variables in the computed normal form $\mathrm{NF}(f)$ are defined by Boolean logic in the SMT instance. If this is the case we extract additional polynomials using the technique of Section III.

If the extraction process cannot identify further polynomials we conduct a resource limited SAT-check whether the normal form vanishes under the constraints of the surrounding SMT-instance. In the current configuration of STABLE the CPU-time spent on each of these checks is limited by 2 minutes. Note that the resource limitation is crucial. Some intermediate proof goals may be extremely hard for SAT before other proof goals have been proven by the normal form engine.

In case we succeed in proving that the normal form is zero

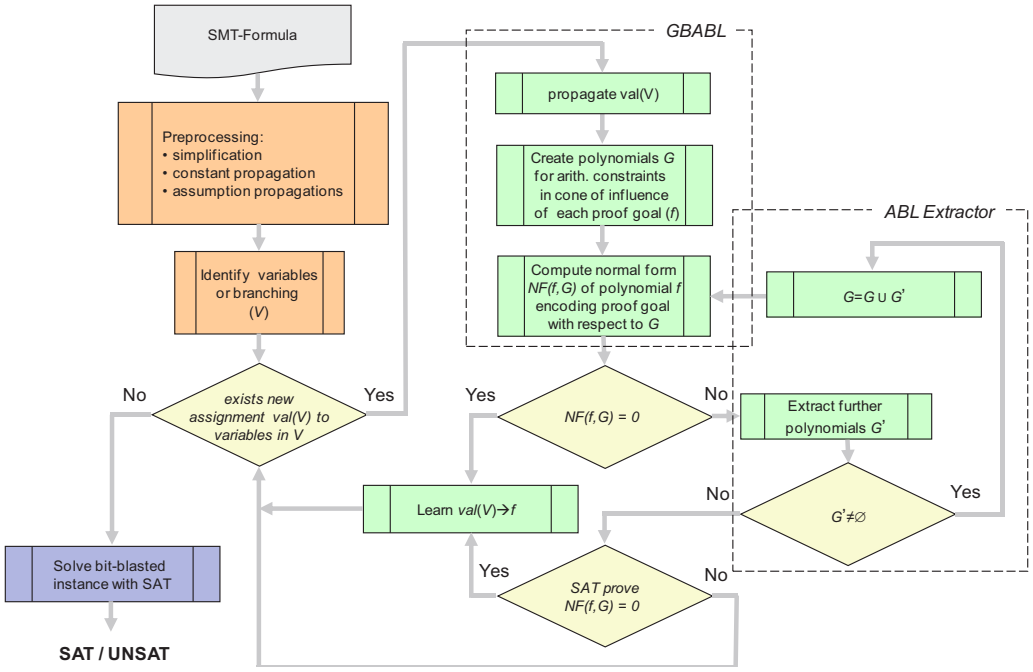

Fig. 2. Flow of *STABLE* with *GBABL (Gröbner basis-based ABL) engine* and *ABL extractor*

we may learn the constraint $val(V) \rightarrow f$. In the frequent special case that V is empty, i. e., we only need to consider a single configuration of the data path, this implication only consists of the unit clause f. This results in additional propagation steps that may considerably simplify other proof goals as well.

When all configurations of the data path that are consistent with the assumptions of the overall SMT instance have been analyzed, i. e., all valuations of V have been enumerated, we newly bit-blast the entire instance including the learned constraints and the corresponding propagation results. A final SAT-check is then conducted to prove non-arithmetic proof goals as well as those where our resource limited checks inside the main loop have been aborted. As a result of this SAT-check we obtain a satisfying assignment or the formula is proven to be unsatisfiable. In our current implementation we use PrecoSAT [7] as backend Solver.

V. Experiments

In this section, we evaluate *STABLE (ST)* and compare its performance against four state-of-the-art solvers, *Spear-2-7(Sp)*, *Boolector 1.4(Bo)*, *MathSat v. 4.3-smtcomp(MS)* and *simplifyingSTP(s.STP)*. These or their predecessors have proven to be highly effective for the logic QV-BV, and have been winners of the SMT competitions 2007-2010 in this category.

All experiments were carried out on an Intel Xeon CPU E5420 2,5 Ghz 32 GB RAM running Linux with a time-out limit of 1000 sec. and a memory limit of 8 GB per instance. We conducted experiments with four large benchmark sets:

- The benchmarks of the 2009 SMT competition [25].
- Formal verification instances of multipliers from a commercial module generator. They include custom-designed components for Booth-encoded partial products and parts of the addition network. The bit-width for each of the inputs is individually scaled from 4 up to 64 bits, i. e., it ranges from 4×4 up to 64×64 bits.
- Formal verification problems for Infineon's TriCore Processor. Each instance proves that a specific instruction variant with multiplication is properly executed by the data-path.
- Satisfiable instances from formal data-path verification, where typical bugs such as missing or wrong connections, wrong operators etc. have been introduced into the design.

We provide tables that report the quota of solved benchmarks for each of the above suites and solvers within various CPU-time limits and a memory limit of 8 GB. Moreover, we report the quota of instances that were aborted due to the time-out limit (T), the memory limit (M) or where the solver aborted with an unknown result(U).

CPU time sec.	Sp.	Bo.	MS	s.STP	ST.
< 10	82,3%	91,4%	97%	88,9%	83,9%
< 100	89,4%	97,5%	98,5%	97%	95%
< 250	89,9%	98,5%	99%	98,5%	96,5%
< 500	90,4%	98,5%	99%	98,5%	96,5%
< 1000	90,9%	98,5%	99%	99%	97%
T/M/U	9,1%	1,5%	1%	1%	3%

TABLE I
QUOTA FOR SMT-COMP09 INSTANCES

As expected *STABLE* is slightly outperformed on the competition benchmarks of 2009 since almost none of these instances includes hard arithmetic problems with multiplication. For the module generator instances *STABLE*'s competitors only solved instances with input bit-widths of less than 11

CPU time sec.	Sp.	Bo.	MS	s.STP	ST.
< 10	10,8%	13,7%	14,4%	13,4%	72,7%
< 100	19,9%	25,1%	25,9%	21,6%	98,4%
< 250	23,6%	29,3%	29,5%	23,8%	98,4%
< 500	24,3%	32,1%	32,3%	26,5%	99%
< 1000	25,3%	33,2%	36,4%	29,5%	100%
T/M/U	74,7%	66,8%	63,6%	70,5%	0%

TABLE II
QUOTA FOR MODULE GENERATOR INSTANCES

CPU time sec.	Sp.	Bo.	MS	s.STP	ST.
< 10	0,5%	12%	0,3%	0%	18,4%
< 100	0,5%	12%	0,5%	0%	96,4%
< 250	0,5%	15,8%	0,5%	0%	100%
< 500	0,5%	17,4%	0,5%	0%	100%
< 1000	0,5%	17,7%	0,5%	0%	100%
T/M/U	99,5%	82,3%	99,5%	100%	0%

TABLE III
QUOTA FOR TRICORE INSTANCES

CPU time sec.	Sp.	Bo.	MS.	s.STP	ST.
< 10	66,7%	88,5%	71,8%	53,9%	84%
< 100	71,2%	100%	84%	76,3%	98,8%
< 250	71,2%	100%	86,5%	96,8%	99,4%
< 500	71,2%	100%	88,5%	96,8%	99,4%
< 1000	71,2%	100%	88,5%	96,8%	100%
T/M/U	28,8%	0%	11,5%	3,2%	0%

TABLE IV
QUOTA FOR SATISFIABLE INSTANCES

bits. Surprisingly, *Boolector* solved a few instances from the TriCore suite. It turns out that these instances form instructions where one factor is constant. In this case the multiplier degenerates to a cascade of adders that *Boolector* can handle. However, for proper multiplication our competitors give up. We conclude with satisfiable instances. The results confirm our intuition that most satisfiable instances in data-path verification are easy to solve for SAT/SMT-solvers. Consequently, for such examples the overhead of our techniques may not pay off.

VI. CONCLUSION

We introduced the new QF-BV SMT solver *STABLE* featuring a computer algebra library for arithmetic decision problems. As a result, hard industrial verification problems for data path designs become tractable. Experimental results prove the superiority of *STABLE* in this domain over other generic state-of-the-art QF-BV SMT solvers.

ACKNOWLEDGMENT

The *Deutsche Forschungsgemeinschaft* and the *Stiftung Rheinland-Pfalz für Innovation* supported part of this work.

REFERENCES

[1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. International Design Automation Conference (DAC)*, June 1999, pp. 317–320.

[2] H. Jain, D. Kroening, N. Sharygina, and E. M. Clarke, "Word-level predicate-abstraction and refinement techniques for verifying RTL Verilog," *IEEE Transactions on Computer-Aided Design*, vol. 27, no. 2, pp. 366–379, 2008.

[3] Onespin Solutions GmbH, Germany. OneSpin 360MV. [Online]. Available: www.onespin-solutions.com

[4] R. Sebastiani, E. Singerman, S. Tonetta, and M. Y. Vardi, "GSTE is partitioned model checking." in *Proc. International Conference on Computer-Aided Verification (CAV)*, 2004.

[5] E. Goldberg and Y. Novikov, "Berkmin: A fast and robust SAT solver," in *Proc. International Conference on Design, Automation and Test in Europe (DATE)*, 2002, pp. 142–149.

[6] N. Een and N. Soerensson, "An extensible SAT-solver," in *Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT)*, May 2003.

[7] "PrecoSAT 236," http://fmv.jku.at/precosat/. [Online]. Available: http://fmv.jku.at/precosat/

[8] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, *Satisfiability Modulo Theories*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, February 2009, vol. 185, ch. 26, pp. 825–885.

[9] S. Ranise and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," www.SMT-LIB.org, 2006.

[10] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB Standard: Version 2.0," in *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, A. Gupta and D. Kroening, Eds., 2010.

[11] B. Dutertre and L. de Moura, "A Fast Linear-Arithmetic Solver for DPLL(T)," in *Proc. International Conference Computer Aided Verification (CAV)*, ser. LNCS, vol. 4144. Springer-Verlag, 2006, pp. 81–94. [Online]. Available: http://www.csl.sri.com/users/demoura/papers/CAV06/cav06.pdf

[12] L. M. de Moura and N. Bjoerner, "Efficient e-matching for smt solvers." in *CADE*, ser. Lecture Notes in Computer Science, F. Pfenning, Ed., vol. 4603. Springer, 2007, pp. 183–198.

[13] D. Babić and F. Hutter, "Spear Theorem Prover," in *Proc. of the SAT 2008 Race*, 2008.

[14] R. Brummayer and A. Biere, "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.

[15] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani, "A lazy and layered smt($\{BV\}$) solver for hard industrial verification problems," in *CAV*, 2007, pp. 547–560.

[16] V. Ganesh and D. L. Dill, "A decision procedure for bit-vectors and arrays," in *In Proceedings of the Computer Aided Verification Conference*. Springer, 2007, pp. 524–536.

[17] STP, http://sites.google.com/site/stpfastprover/STP-Fast-Prover.

[18] N. Shekhar, P. Kalla, and F. Enescu, "Equivalence verification of polynomial datapaths using ideal membership testing," *IEEE Transactions on Computer-Aided Design*, vol. 26, no. 7, pp. 1320–1330, July 2007.

[19] E. Pavlenko, M. Wedler, D. Stoffel, O. Wienand, E. Karibaev, and W. Kunz:, "Modeling of custom-designed arithmetic components in ABL normalization," in *Proc. Forum on Specification & Design Languages(FDL)*, Stuttgart, Germany, September 2008.

[20] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G.-M. Greuel, "An algebraic approach for proving data correctness in arithmetic data paths," in *Proc. International Conference Computer Aided Verification (CAV)*, Princeton, NJ, USA, July 2008, pp. 473–486.

[21] W. Adams and P. Loustaunau, *An introduction to Gröbner bases*. (Graduate studies in mathematics) AMS, 2003.

[22] G.-M. Greuel and G. Pfister, *A SINGULAR Introduction to Commutative Algebra*, 2nd ed. Berlin, Heidelberg, New York: Springer Verlag, 2007, 705 pages.

[23] O. Wienand, "Standard bases over rings and applications," 2010, manuscript, Kaiserslautern.

[24] M. Brickenstein, A. Dreyer, G.-M. Greuel, M. Wedler, and O. Wienand, "New developments in the theory of Gröbner bases and applications to formal verification," *Journal of Pure and Applied Algebra*, vol. 213, pp. 1612–1635, 2009.

[25] SMT-COMP 2009, http://www.smtcomp.org/2009/.