Verification of Arithmetic Datapaths using Polynomial Function Models and Congruence Solving

Neal Tew[1], Priyank Kalla[1], Namrata Shekhar[2], Sivaram Gopalakrishnan[1]
[1]ECE Dept., University of Utah, Salt Lake City UT
[2]Synopsys Inc., Marlborough MA
tew@cs.utah.edu, kalla@ece.utah.edu, Namrata.Shekhar@synopsys.com, sgopalak@ece.utah.edu

Abstract— **This paper addresses the problem of solving finite word-length (bit-vector) arithmetic with applications to equivalence verification of arithmetic datapaths. Arithmetic datapath designs perform a sequence of** ADD, MULT, SHIFT, COMPARE, CONCATENATE, EXTRACT**, etc., operations over bit-vectors. We show that such arithmetic operations can be modeled, as constraints, using a system of polynomial functions of the type $f : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$. This enables the use of modulo-arithmetic based decision procedures for solving such problems in one unified domain. We devise a decision procedure using Newton's p-adic iteration to solve such arithmetic with composite moduli, while properly accounting for the word-sizes of the operands. We describe our implementation and show how the basic p-adic approach can be improved upon. Experiments are performed over some communication and signal processing designs that perform non-linear and polynomial arithmetic over word-level inputs. Results demonstrate the potential and limitations of our approach, when compared against SAT-based approaches.**

I. Introduction

Many datapath designs perform a sequence of arithmetic operations such as ADD, MULT, SHIFT, COMPARE, DIVIDE, etc., over finite word-length operands (bit-vectors). In practice, operating on finite word-length data-types can introduce subtle, unforeseen errors such as overflow, causing incorrect function or even security vulnerabilities. With the widespread use of finite-precision arithmetic in multimedia DSP, communication centric ASICs, embedded control, etc., it is imperative to devise new techniques to efficiently model and verify such systems at higher levels of abstraction.

Traditionally, *bit-level* decision procedures such as SAT, BDD, ATPG, etc., have been used for verifying designs at the bit-level [1] [2]. Techniques that rely on "bit-blasting" to solve such problems at lower-levels of abstraction are computationally infeasible. Other techniques reason about these computations using arbitrary precision arithmetic over integers (Z) or reals (R) - which cannot always capture the nuances of finite-precision [3] [4]. For these reasons, recent works in this area try to integrate the above procedures [5] [6], solve a subset of bit-vector theories [7], or use abstraction-refinement based techniques that, at some point, rely on SAT engines for proofs [8]. These approaches have made some headway, but solving non-linear bit-vector arithmetic still remains a challenge.

Ideally, what is required is an *intermediate* level of representation that has the requisite power of abstraction, as well as the capability to model the effects of finite precision. A bit-vector of size m represents integer values in the range $\{0, 1, \ldots, 2^m - 1\}$, i.e. integer values reduced *modulo* 2^m. This motivates the use of number theory and commutative algebra for modeling and verification of bit-vector arithmetic.

This work is sponsored in part by the US National Science Foundation Faculty Early Career (CAREER) Development Award #CCF-546859.

When dealing with unsigned and two's complement *polynomial* arithmetic over m-bit-vectors, i.e. overflow arithmetic using only ADD, MULT operations, the computations can be modeled as *polynomial functions over finite integer rings of residue classes* Z_{2^m}. Subsequently, classical ring theory can be applied to solve the verification problems (see our prior work [9] [10] [11]). However, bit-vector arithmetic encompasses a larger gamut of word-level operations, such as ADD, MULT, SHIFT, COMPARE, CONCATENATE, SUB-WORD EXTRACT, DIVIDE, etc. In the presence of such computations, the classical polynomial function model "breaks down". For example, RIGHT-SHIFT cannot be modeled as a polynomial function over Z_{2^m}, as division by 2 is undefined in ring Z_{2^m}. Similarly, COMPARE is not a polynomial function; and so on.

While such computations cannot be represented by a single polynomial function, this paper shows that they can be modeled, as constraints, by a **system of polynomial functions** with composite moduli. This enables the direct use of modulo-arithmetic constraint solving to solve and verify bit-vector arithmetic in one unified domain – sacrificing neither word-level abstraction, nor the precision issues manifested by finite word-length operands.

Contributions: i) We address *equivalence verification* of arithmetic datapaths with finite word-length operands. The targeted applications are those that implement non-linear arithmetic commonly found in communication and multimedia DSP, embedded control, etc. ii) We model finite word-length bit-vector arithmetic, *as constraints, by a system of polynomial functions* of the type $f : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$ [12]; where $n_1, \ldots, n_d, m$ correspond to the bit-vector word-lengths. Following bit-vector operations are considered: ADD, MULT, LEFT SHIFT, RIGHT SHIFT, COMPARE, SUB-WORD EXTRACT, CONCATENATE, DIVIDE, MODULUS, MUX. Both unsigned as well as two's complement arithmetic are considered. iii) We devise a decision procedure based on modulo-arithmetic congruence solving to solve such arithmetic with composite moduli. We use Newton's p-adic iteration [13] for this purpose. In our case, $p = 2$. While the use of Newton's p-adic iteration was proposed in [14] for software verification, we show how the basic number-theoretic solving approach can be improved upon by integrating it with constraint satisfaction engines. iv) Using an efficient CAD implementation, we demonstrate its application to equivalence checking of arithmetic datapaths (RTL-to-RTL). Experiments conducted over some DSP and communication-centric designs demonstrate the potential (and limitations) of our approach, as compared against SAT-based approaches.

II. Previous Work

Straight-forward use of a SAT solver on the "bit-blasted" design has been employed in verification [1] [2], etc. Other techniques such as CVC [15], STP [6] and that of [16] use pre-

processing steps to simplify the generated constraints, and then use a SAT solver. SMT solvers such as [3] and [5] also employ bit-blasting.

There are procedures that target only a specific subset of bit-vector theories such as concatenation, extraction and linear equations [7] [17] [18]. These techniques are limited by a lack of generality.

The works of LPSAT [4], HSAT [19], [20] and that of Huang [21] use linear (modulo) arithmetic formulations to decide bit-vector formulas. A major limitation of all these works is their inability to resolve multiplication and other non-linear word-level operations.

In [8], authors use an abstraction-refinement based approach to solve bit-vector arithmetic. They iteratively compute under- and over-approximations of the (encoded) bit-vector formula, and use a SAT solver to ultimately find a proof. Their approach can handle arbitrary word-level operations, including multiplication and other non-linear operations. The efficiency of their approach is dependent upon finding small unsatisfiable cores from the formula. For equivalence checking instances, unfortunately, the UNSAT core size can be relatively large - in such cases their technique may not see the full benefit of abstraction.

In a technical report, *Babic et al.* [14] propose the use of p-adic approaches to solve a system of polynomial equations modulo 2^m, for software verification. However, their approach cannot model unsigned and two's complement COMPARE operations as polynomials. Instead, they reason about compare operations over unbounded integers; the overall system integrates a SAT solver, a p-adic solver and inequality reasoning in a Nelson-Oppen [22] type framework. Moreover, implementation details and experimental results are not reported, which makes it hard to gauge the effectiveness of their overall approach. Furthermore, division is not supported in their framework.

III. Preliminaries

Definition III.1: Integers x, y are called **congruent modulo** n, denoted $x \equiv y \bmod n$ (or $x \equiv y\%n$), if n is a divisor of their difference: $n|(x-y)$.

We denote by Z the ring of integers. The set $Z_n = \{0, 1, \ldots, n-1\}$, where $n \in N$, forms a commutative ring with unity. It is called the **residue class ring**, where addition and multiplication are defined (mod n). Note that for our applications, $n = 2^m$, where m is the bit-vector word-length.

The concept of congruence naturally extends to polynomials; e.g. $f(x_1, \ldots, x_d) \equiv 0 \bmod 2^m$ is a **polynomial congruence**. In this work, we investigate a **system** of such congruences (mod 2^m): $f_i(x_1, \ldots, x_d) \equiv 0 \bmod 2^m, 1 \leq i \leq n$. If such a system has solutions, then we call the system **satisfiable**; otherwise we call it **unsatisfiable**. The concept of finding the solutions to a system of polynomial congruences is addressed in Section V.

A. Polynomial Functions and Bit-Vector Arithmetic

Functions over finite integer rings that can be represented by polynomials are generally termed as polynomial functions (or **polyfunctions**). Early studies on polyfunctions $f : Z_n \rightarrow Z_n$ have been extended and generalized to those over $f : Z_n \rightarrow Z_m$ and further to $Z_{n_1} \times Z_{n_2} \times \cdots \times Z_{n_d} \rightarrow Z_m$ [12]. The following definition of such a polynomial function is taken from [12], and modified, for our application, to rings modulo an integer power of 2.

Definition III.2: A function f from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$ is said to be a polynomial function if it is represented by a polynomial $F \in Z[x_1, x_2, \ldots, x_d]$; i.e. $f(x_1, x_2, \ldots, x_d) \equiv F(x_1, x_2, \ldots, x_d)$ for all $x_i \in Z_{2^{n_i}}$, $i = 1, 2, \ldots, d$ and $\equiv$ denotes congruence (mod 2^m).

Example III.1: Let $f : Z_{2^1} \times Z_{2^2} \rightarrow Z_{2^3}$ be a function defined as: $f(0,0) = 1$, $f(0,1) = 3$, $f(0,2) = 5$, $f(0,3) = 7$, $f(1,0) = 1$, $f(1,1) = 4$, $f(1,2) = 1$, $f(1,3) = 0$. Then, f is a polyfunction representable by $F = 1 + 2y + xy^2$, since $f(x,y) \equiv F(x,y) \bmod 2^3$ for $x = 0, 1$ and $y = 0, 1, 2, 3$.

Note that in the above model, each variable has its own range, but the result F is computed *modulo* 2^m. In our context, $n_1, \ldots, n_d$ represent word-lengths of the input variables and m is the output word-length.

Not every function of the type $Z_{n_1} \times Z_{n_2} \times \cdots \times Z_{n_d} \rightarrow Z_m$ is a polynomial function. In [12], Chen derives the necessary and sufficient conditions for a function of his type to be a polynomial function. Moreover, if a function is a polynomial function, Chen shows how a (canonical) polynomial representation can be derived (interpolated) for that function.

Example III.2: For example, consider the following RTL code-snippet, where x is a 2-bit vector and y is 5-bits wide:

if $(x > 2)$ *then* $y = x^3$ *else* $y = x^2$*;*

By applying Chen's polynomial interpolation criteria, the 5-bit output y can be represented as a polyfunction from $Z_{2^2} \rightarrow Z_{2^5}$ as $Y = 3x^3 + 8x^2 + 22x \pmod{2^5}$.

It is practically infeasible to apply Chen's polynomial interpolation criteria to large designs, as it requires one to *analyze the entire function*. So, we investigated whether or not individual non-linear bit-vector arithmetic operators (such as RIGHT-SHIFT, COMPARE) can be represented as a polynomial function? If this were possible, then we could: i) traverse the data-flow graph (DFG) of the design; ii) for every bit-vector operator, derive its polynomial representation; iii) create a system of polynomial congruences and solve them for verification.

Unfortunately, our studies showed that most non-linear bit-vector operations (such as RIGHT SHIFT, COMPARE, DIVIDE, MODULUS) are not polynomial functions. In fact, we also found that designs with only ADD, MULT operations, but with arbitrary internal signal word-lengths may also cease to be polyfunctions, as shown below.

Consider the (didactic) equivalence checking example shown in Fig. 1. Fig 1 (a) implements $x^2 + x$ and Fig. 1 (b) implements $x(x+1)$. Over R and Z, these computations are equivalent. However, in the figure, the designs are implemented with specific bit-vector sizes and hence they are not equivalent. Note that for $x = 2, F = 2$, whereas $G = 6$.

In Fig. 1 (a), $t_1(x)$ is a polynomial function from $Z_{2^2} \rightarrow Z_{2^2}$, represented as $t_1 \equiv x^2 \pmod 4$. Also, $F(t_1, x)$ is a polynomial function over $Z_{2^2} \times Z_{2^2} \rightarrow Z_{2^3}$, represented as $F \equiv t_1 + x \pmod 8$. However, the whole design cannot be represented as a single polynomial function because the *composition* of t_1 and F is not a polynomial function. *Bhargava* [23] studied this problem and found that *over finite integer rings, the composition of two or more polynomial functions is not always a polynomial function*. In such cases, we cannot represent the entire design as a polynomial function – hence, we cannot employ classical ring theory to reason about the equivalence of such designs.

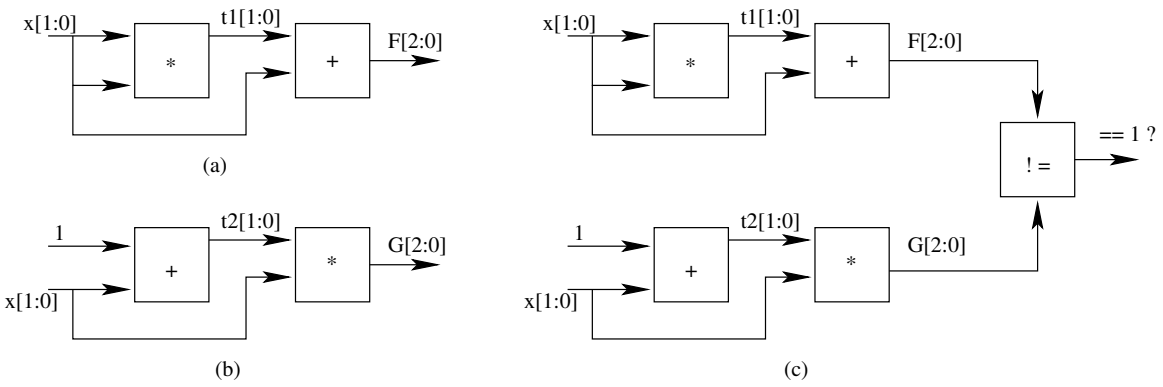

Fig. 1. Check if $x^2 + x \equiv x(x+1)$ with bit-vector truncation

Motivating our idea: Interpreting *Bhargava's* result conversely, we investigated whether or not we can take a non-polynomial function and *decompose it into a system of polynomial functions*. Intriguingly, our investigations show that non-polynomial bit-vector operations such as COMPARE, DIVIDE, SUB-WORD EXTRACT, RIGHT-SHIFT etc. can be modeled, as constraints, using a sequence of ADD/SUB, MULT, and SELECTION operations - each of which can be individually modeled as polynomial functions of Chen's type [12]. Now we can traverse the DFG of the design and model individual bit-vector arithmetic computations using a system of polynomial functions and solve the corresponding polynomial congruences.

Fig. 1 (c) depicts our equivalence checking setup using a "word-level miter". While the ADD, MULT units are polynomial functions themselves, the $\neq$ comparator is not. However, if this comparator can also be modeled, as constraints, using a system of polynomial functions over $f : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$, then we can use constraint satisfaction and polynomial algebra to solve this problem. In what follows, we describe the modeling of these non-linear bit-vector arithmetic operators using polynomial function constraints (with composite moduli), and then show how to solve them.

IV. Modeling Bit-Vector Arithmetic with Polynomial Function Constraints

Boolean operations: Let x, y, c be Boolean variables, i.e. in Z_2. Then $c = \bar{x}$ can be represented over Z_2 as $c = (1 + x)$. Similarly, $c = x \wedge y$ is $c = x \cdot y$, $c = x \vee y$ is $c = x + y + x \cdot y$ and $c = x \oplus y$ (XOR) is $c = x + y$, where all operations are performed modulo 2.

A. Unsigned/Two's Complement Bit-Vector Comparisons

We will now show how comparator operations can be modeled, as constraints, using polynomial congruences. Modern HDLs, such as Verilog, can perform comparisons over unsigned as well as two's complement inputs. The constraints that we derive can model both unsigned and two's complement comparisons[1]. For all the comparator operations, x and y are m-bit vectors, representing either both unsigned or both two's complement values and c denotes the Boolean result.

[1]This should be somewhat obvious as m-bit two's complement arithmetic is just an "interpretation" of arithmetic modulo 2^m.

1. **Greater than (>)**: Let $c = (x > y)$. Here, $c \in Z_2$ and $x, y \in Z_{2^m}$. In order to decide $x > y$, we consider whether or not $(x - y) > 0$. In other words, we first compute $x - y$, and based on whether the result is zero, positive or negative, we suitably *select* the value of c. Since x, y are m-bit vectors, and $(x - y)$ will produce an $(m+1)$-bit result, we introduce intermediate variables $t_1 \in Z_{2^{m+1}}$ and $t_2 \in Z_{2^m}$. Then the polynomial constraint set for $c = (x > y)$ is given by:

$$t_1 \equiv (x - y - 1) \pmod{2^{m+1}} \quad (1)$$
$$t_1 \equiv (c \cdot t_2 + (1 - c)(2^{m+1} - 1 - t_2)) \pmod{2^{m+1}} \quad (2)$$

The first constraint models t_1 as a function from $Z_{2^m} \times Z_{2^m} \rightarrow Z_{2^{m+1}}$, whereas the second constraint models t_1 as a function from $Z_2 \times Z_{2^m} \rightarrow Z_{2^{m+1}}$. The objective is to find assignments to the variables such that the above congruences are satisfied. Further, the constraints for $c = (x \leq y)$ can be obtained by replacing c with $(1 - c)$ in Eqn. (2).

Example IV.1: Consider the above operation for unsigned integers $x, y \in Z_{2^3}$ and $c = (x > y)$ where $c \in Z_2$. The intermediate variables are $t_1 \in Z_{2^4}$ and $t_2 \in Z_{2^3}$. The corresponding constraints are:

$$t_1 \equiv (x - y - 1) \bmod 2^4$$
$$t_1 \equiv (c \cdot t_2 + (1 - c) \cdot (15 - t_2)) \bmod 2^4$$

Now consider the following instances of x and y. In each case, the above model provides the correct solution for c.

- $x = 4,\ y = 5$:

$$t_1 \equiv (4 - 5 - 1) \bmod 2^4 = 14 \bmod 2^4$$
$$14 \equiv (c \cdot t_2 + (1 - c) \cdot (15 - t_2)) \bmod 2^4$$

The **only feasible solution** for these congruences is $t_2 = 1,\ c = 0$, which is correct as $c = (x > y)$ is false. If we put $c = 1$, t_2 cannot be equal to 14 as $t_2 \in Z_8$.

- $x = 5,\ y = 4$ gives $t_1 = 0, t_2 = 0, c = 1$ as the only solution.
- $x = 5,\ y = 5$ gives $t_1 = 15, t_2 = 0, c = 0$ as the only solution.

Example IV.2: Now let us consider $c = (x > y)$ where x, y are 3-bit two's complement bit-vectors. Again, we have $c \in Z_2$ and $x, y \in Z_{2^3}$. The intermediate variables are $t_1 \in Z_{2^4}, t_2 \in Z_{2^3}$. Now consider the following cases:

- $x = -1, y = 2$:

$$t_1 \equiv (-1 - 2 - 1) \bmod 2^4 = 12 \bmod 2^4$$
$$12 \equiv (c \cdot t_2 + (1 - c) \cdot (15 - t_2)) \bmod 2^4$$

The only solution is $t_2 = 3,\ c = 0$.

- $x = 2, y = -2$ gives $t_1 = 3, t_2 = 3,\ c = 1$.
- $x = -4, y = -4$ gives $t_1 = 15, t_2 = 0, c = 0$.

Another way to understand why the same model works for both unsigned and two's complement compares is to consider that: i) in the two's complement case, we *sign-extend* x, y by 1-bit to normalize it to a $(m+1)$-bit computation; whereas in ii) the unsigned case we *zero-extend* x, y by 1-bit. Moreover, the second congruence models the selection of c based upon whether $x - y$ is zero, or between 1 and $2^m - 1$ (positive), or between 2^m and $2^{m+1} - 1$ (negative).

2. **Less than ($<$)**: Let $c = (x < y)$. Hence, $c \in Z_2$ and $x,\ y \in Z_{2^m}$. Let $t_1 \in Z_{2^{m+1}}$ and $t_2 \in Z_{2^m}$. Then, the polynomial constraints for $c = (x < y)$ are given by:

$$t_1 \equiv (x - y) \bmod 2^{m+1} \quad (3)$$

$$t_1 \equiv ((1 - c) \cdot t_2 + c \cdot (2^{m+1} - 1 - t_2)) \bmod 2^{m+1} \quad (4)$$

Again, the corresponding constraints for $c = (x \geq y)$ can be obtained by replacing c with $(1 - c)$ in Eqn. (4).

3. **Not Equal ($\neq$)**: Let $c = (x \neq y)$. Here, $c \in Z_2$ and $x,\ y \in Z_{2^m}$. We can rewrite this as

$$\begin{aligned} c &= (x > y) \vee (x < y) \\ &= c_1 \vee c_2 \end{aligned} \quad (5)$$

where $c_1, c_2 \in Z_2$, and use their corresponding models to represent $(x \neq y)$. The polynomial representation for $c = (x == y)$ can be similarly obtained by rewriting it as

$$c = (x \geq y) \wedge (x \leq y) \quad (6)$$

and using their corresponding representations. This way we can model all compare operations.

Another model for using the ($\neq$) compare as a miter: As shown in Fig. 1 (c), we need to "miter" the outputs of the two designs for equivalence checking. In principle, the $\neq$ comparator model described above can be used for this purpose. However, this model is inefficient as it introduces too many extra variables with different word-lengths. This complicates and slows-down the solving process. A better way to model the $x \neq y$ comparator as a miter is as follows: Let $x, y \in Z_{2^m}$, and let $t \in Z_{2^m}$ be a free variable. Then:

$$t \cdot (x - y) \equiv 2^{m-1} \pmod{2^m} \quad (7)$$

will have no solution for t when $x = y$. On the other hand, when x and y are different, $x - y$ is a non-zero integer, and there will always exist a $t \in Z_{2^m}$ that satisfies the above constraint.

B. Right Shift and Sub-Word Extraction

Left shift is simply multiplication by 2, so let us instead consider the right shift operation. Let x and F denote m-bit vectors and k denote a positive integer such that $F = (x \gg k)$. Here, $\gg k$ denotes x being shifted to the right by k bits. We represent this operation using an intermediate computation $t \in Z_{2^k}$ where $t = x \bmod 2^k$. The model for F over Z_{2^m} is :

$$2^k \cdot F \equiv (x - t) \bmod 2^m \quad (8)$$

Sub-Word Extraction: Let $x[m-1:0]$ denote a bit-vector, i, j denote positive integers, and $F = x[i:j]$. Here, $x \in Z_{2^m}$, $F \in Z_{2^{i-j+1}}$ and $i, j \in Z^+$. To model this computation as a polynomial, we introduce an intermediate computation $t \in Z_{2^m}$ such that $t \equiv (2^{m-i-1} \cdot x) \bmod 2^m$. Now, F can be modeled by

$$\begin{aligned} F &\equiv (t \gg m - i + j - 1) \bmod 2^{i-j+1} \\ (2^{m-i+j-1})F &\equiv (t - t \ \% \ 2^{m-i+j-1}) \bmod 2^{i-j+1} \end{aligned} \quad (9)$$

In similar fashion, concatenate and rotate operations can be modeled using a sequence of shifts, extracts and adds.

C. Division Operation

To model the unsigned division, let $Q = F/D$ be the quotient and $R = F\%D$ be the remainder of the division. Assume that $F, D, Q, R \in Z_{2^m}$. Then division is modeled by representing $F = Q \cdot D + R \pmod{2^m}$, and then selecting Q. However, we also have to add the following extra constraints: $R < D$; $Q \leq F$, and $Q \cdot D + R < 2^m$.

Example IV.3: Let F, D, Q, R be 4-bit vectors representing unsigned integers. Let $F = 5, D = 2$. Note that if we do not consider the constraints $R < D$ and $Q \leq F$, then $(Q, R) = (0, 5), (3, 15), (10, 1), \ldots$ (among others) can satisfy $F = Q \cdot D + R \pmod{2^4}$, which are all incorrect. However, when we add the extra constraints on the range of Q, D, R, we obtain the only correct solution $(Q, R) = (2, 1)$.

D. Dealing with Composite Moduli

Consider a system of n polynomials F_i $(1 \leq i \leq n)$ in variables $x_1, x_2, \ldots, x_d$, where each F_i represents a bit-vector computation. Let m_i $(1 \leq i \leq n)$ correspond to the output bit-widths of each of these polynomials, such that $F_i \in Z_{2^{m_i}}$. Then,

$$\begin{aligned} F_1(x_1, \ldots, x_d) &\equiv 0 \bmod 2^{m_1} \\ &\vdots \\ F_n(x_1, \ldots, x_d) &\equiv 0 \bmod 2^{m_n} \end{aligned} \quad (10)$$

Once these constraints are generated, they need to be solved. We are unaware of any approach that can directly solve such arithmetic with composite moduli. However, these constraints can be *scaled* to a uniform modulus and then solved using a p-adic approach. To perform the scaling, we first compute the least common multiple (LCM) of the moduli in the above system, *i.e.* $2^M = LCM(2^{m_1}, \ldots, 2^{m_n})$. Now, we rewrite the polynomials as:

$$\begin{aligned} \frac{2^M}{2^{m_1}} \cdot F_1(x_1, \ldots, x_d) &\equiv 0 \bmod 2^M \\ &\vdots \\ \frac{2^M}{2^{m_n}} \cdot F_n(x_1, \ldots, x_d) &\equiv 0 \bmod 2^M \end{aligned} \quad (11)$$

All the polynomials $F_1, \ldots, F_n$ are now computed mod 2^M. Once the solutions to these constraints are obtained for the scaled system of equations, they have to be *refined* to see if solution to each $x_i \in Z_{2^{n_i}}$. We explain our approach by means of an example.

Example IV.4: Let us re-visit the equivalence checking problem depicted in Fig. 1 (c). Here, the input is $x[1:0]$ and the outputs are $F[2:0]$ and $G[2:0]$. The problem instance requires to check whether there exists an assignment to $x[1:0]$ such that $F \neq G$. Since the internal signals $t_1, t_2 \in Z_4$ and $F, G \in Z_8$, the corresponding equations have to be scaled

to a common modulus $2^M = LCM(2^2, 2^2, 2^3, 2^3) = 2^3$. Consequently, the scaled system of equations is:

$$\begin{aligned} 2x^2 - 2t_1 &\equiv 0 \bmod 2^3 \\ t_1 + x - F &\equiv 0 \bmod 2^3 \\ 2x + 2 - 2t_2 &\equiv 0 \bmod 2^3 \\ t_2 \cdot x - G &\equiv 0 \bmod 2^3 \\ t \cdot (F - G) - 4 &\equiv 0 \bmod 2^3 \end{aligned}$$

Note that the last constraint $t \cdot (F - G) - 4 \equiv 0 \bmod 2^3$ models the miter at the output. The solutions are: i) $x = 2;\ F = 2; G = 6$; ii) $x = 3;\ F = 4; G = 0$; iii) $x = 5;\ F = 6; G = 2$ and iv) $x = 6; F = 6; G = 2$. Since $x \in Z_4$, solutions $(x = 5, 6)$ are out of bounds and need to be discarded. As $(x = 2, 3)$ are valid solutions, we deduce that $F \neq G$.

V. Solving Polynomial Congruences modulo 2^m

To solve the system of polynomial equations mod 2^m, we make use of Newton's iteration formula on the space of p-adic $(p = 2)$ expansion. We borrow the basic solving concept from the textbook [13] and from *Babic et al.* [14], and show how it can be improved upon for our specific problems.

To solve a univariate polynomial $f(x) = 0 \bmod p^m$, the basic concept is to first obtain solutions for $f(x) = 0 \bmod p$. Once all solutions are obtained (mod p), they are successively *lifted* modulo $p^2, p^3, \ldots, p^m$. To derive the recurrence formula for lifting, we begin with the Taylor's Expansion of a polynomial $f(x)$ at $x = r$:

$$\begin{aligned} f(x) &= f(r) + f'(r) \cdot (x - r) + \frac{f''(r)}{2!} \cdot (x - r)^2 + \\ &\quad \cdots + \frac{f^{(n)}(r)}{n!} \cdot (x - r)^n \quad (12) \\ &= f_0 + f_1 \cdot (x - r) + \cdots + f_n \cdot (x - r)^n \quad (13) \end{aligned}$$

We have to solve $f(x) \equiv 0 \bmod p^m$, for $p = 2$. Assume that a solution $r_{k-1} \in Z_{p^k}$, $k < m$ is known. Since $f(r_{k-1}) \equiv 0 \bmod p^k$, then it follows that $f(r_{k-1}) \equiv 0 \bmod p^{k-1}$. Therefore, we see that solutions can be given as $r_k = r_{k-1} + t \cdot p^{k-1}$, for $0 \leq t < p$. The p-adic expansion of the polynomial can then be written as:

$$\begin{aligned} f\left(r_{k-1} + t \cdot p^{k-1}\right) &\equiv f(r_{k-1}) + f'(r_{k-1}) \cdot t \cdot p^{k-1} + \cdots + \\ &\quad f^{(k-1)}(r_{k-1}) \left(t \cdot p^{k-1}\right)^n \pmod{p^k} \quad (14) \end{aligned}$$

Note that higher order terms $(p^{n \cdot (k-1)})$ are divisible by p^k for $n \geq 2$ and vanish. Eqn. (14) subsequently reduces to the one below, where we have to solve for $t, 0 \leq t < p$:

$$f(r_{k-1}) + f'(r_{k-1}) \cdot t \cdot p^{k-1} \equiv 0 \pmod{p^k} \quad (15)$$

The corresponding value of t is then computed as:

$$t \equiv -f'(r_{k-1})^{-1} \cdot \frac{f(r_{k-1})}{p^{k-1}} \pmod{p} \quad (16)$$

where f' denotes the derivative of f w.r.t. x, and f^{-1} denotes the multiplicative inverse of f mod p. Thus, if we have a root $r_{k-1} \pmod{p^{k-1}}$, the root $r_k \pmod{p^k}$ can be computed as as $r_k = r_{k-1} + t \cdot p^{k-1}$, with the value of t computed from Eqn. (16). Consolidating the above results, we arrive at the following lemma.

Lemma 1: (**Lifting Lemma**) Let f be a polynomial with integer coefficients, and r be a solution to $f(x) \equiv 0 \pmod{p^{k-1}}$, with $k \geq 2$ and prime p. Then:

1. If $f'(r) \neq 0 \bmod p$, then there exists a unique integer $t, 0 \leq t < p$ such that $f(r + t \cdot p^{k-1}) \equiv 0 \bmod p^k$ and is given by $t = -f'(r)^{-1} \frac{f(r)}{p^{k-1}} \bmod p$;
2. If $f'(r) \equiv 0 \bmod p$ and $f(r) \equiv 0 \bmod p^k$, then $f(r + t \cdot p^{k-1}) \equiv 0 \bmod p^k$ for all integers t;
3. If $f'(r) \equiv 0 \bmod p$ and $f(r) \neq 0 \bmod p^k$, then the current solution (r) does not lift.

Moreover, if $f(r) \equiv 0 \pmod p$ and $f'(r) \neq 0 \pmod p$, then there is a unique solution r_k modulo p^k for $k = 2, 3, \ldots$, given by the recurrence: $r_k = r_{k-1} - f(r_{k-1}) \cdot f'(r)^{-1}$.

Example V.1: For example, suppose we have to solve $f(x) = x^2 + x + 47 \equiv 0 \bmod 7^3$. First we solve $f(x) \equiv 0 \bmod 7$ and get $r = 1, 5$ as the solutions. We have to see if each of these solutions lifts to a solution modulo 7^2 and then to 7^3. Here $f'(x) = 2x + 1$.

For $r = 1$ as a solution mod 7, we see that $f'(r = 1) \equiv 3 \neq 0 \bmod 7$. Therefore, $r = 1$ will lift successively to a solution modulo 7^3. We set $r_1 = 1$. Moreover, $f'(r = 1)^{-1} = 5 \pmod 7$. Therefore, we have:

$$\begin{aligned} r_2 &= r_1 - f(r_1) \cdot f'(r = 1)^{-1} = 1 - f(1)(5) = 1 \bmod 7^2 \\ r_3 &= r_2 - f(r_2) \cdot f'(r = 1)^{-1} = 1 - f(1)(5) = 99 \bmod 7^3 \end{aligned}$$

This way, we have lifted solution $x = 1 \pmod 7$ to a solution $x = 99 \pmod{7^3}$. Similarly, we can check that the solution $r = 5$ also lifts to a solution $x = 243 \bmod 7^3$.

The above approach extends analogously to the case of *multivariate* polynomials. Let us denote by $\overline{f} = [f_1, \ldots, f_n]$ a (transposed) vector of polynomials and by $\overline{x} = [x_1, \ldots, x_n]$, the input variables. Also, let $\overline{t} = [t_1, \ldots, t_n]$, where $0 \leq t_i < p$. Since we are now operating over a vector of multivariate polynomials, we need to compute the (partial) derivatives of all polynomials w.r.t. all input variables. Therefore, we need to consider the *Jacobian* matrix J, which is a square matrix of partial derivatives, given as:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (17)$$

Analogous to Eqn. (15), we have to solve the following system for $\overline{t} = [t_1, \ldots, t_n]$, where $0 \leq t_i < p$:

$$\overline{f}(\overline{x}_{k-1}) + J(\overline{x}_{k-1}) \cdot \overline{t} \cdot p^{k-1} \equiv 0 \pmod{p^k} \quad (18)$$

Then, $\overline{t}$ is computed as:

$$\overline{t} = -J(\overline{x}_{k-1})^{-1} \cdot \left(\frac{\overline{f}(\overline{x}_{k-1})}{p^{k-1}}\right) \pmod p \quad (19)$$

and finally, the recurrence $\overline{x}_k \equiv \overline{x}_{k-1} + \overline{t} \cdot p^{k-1}$ is given by:

$$\overline{x}_k \equiv \overline{x}_{k-1} + [-J(\overline{x}_{k-1})^{-1} \cdot \left(\frac{\overline{f}(\overline{x}_{k-1})}{p^{k-1}}\right) \bmod p] \cdot p^{k-1} \pmod{p^k} \quad (20)$$

It is required to compute the inverse of the Jacobian for all the points mod p. However, every solution in Z_{p^k} is also a solution in $Z_{p^{k-1}}$. Therefore, $\overline{x}_k \equiv \overline{x}_1 \bmod p$ and hence $J(\overline{x}_{k-1}) \equiv J(\overline{x}_1) \bmod p$. In other words, the inverse of the Jacobian needs to be computed only once for the initial points $\overline{x}_1$. Depending upon whether or not the inverse of the Jacobian exists, we have to analyze three different lifting conditions analogous to those in Lemma 1, as given below:

Lifting conditions for multivariate polynomials:

- When the inverse of the Jacobian exists (mod p), then $\overline{x}_{k-1}$ lifts uniquely as $\overline{x}_{k-1} + \overline{t} \cdot p^{k-1}, 0 \leq t_i < p$, where $\overline{t}$ is computed as in Eqn. (19).
- When the inverse of the Jacobian does not exist (mod p), and $\overline{f}(\overline{x}_{k-1}) \equiv 0 \bmod p^k$, then $\overline{x}_{k-1}$ lifts (non-uniquely) to $\overline{x}_{k-1} + \overline{t} \cdot p^{k-1}$ for all values of $\overline{t} = [t_1, \ldots, t_n], 0 \leq t_i < p$.
- Otherwise, x_{k-1} does not lift.

This approach requires that we somehow compute the solutions $\overline{x}_1 \pmod 2$ as starting points, and then apply the lifting

lemma. To compute $\overline{x}_1$, we have to solve multi-variate non-linear congruences in Z_2, which is NP-complete. However, since $x^p = x \bmod p$ (Fermat's little theorem), the problem reduces to that of solving multi-linear equations in Z_2. Eqn. (20) provides all solutions to the given set of equations, which can be checked to see if each x_i is in $Z_{2^{n_i}}$, i.e. within the range of the given bit-vector.

A. Implementation issues: Solution Explosion

As a first attempt, we implemented the above approach completely within the symbolic algebra engine of MATHEMATICA, and conducted some experiments. We observed that in many practical instances, the inverse of the Jacobian J does not exist (mod 2). This happens because the generated problem instances are often an *incomplete system of congruences* – i.e. the number of polynomial congruences are not equal to the number of variables (refer to Example IV.4, where we have 6 variables and 5 congruences). In such cases, the Jacobian is not a square matrix – and a non-square matrix does not have an inverse over a commutative ring. As a result, every solution $\overline{x}_{k-1}$ lifts (non-uniquely) to a large number of solutions $\overline{x}_k$. Much of these are bad starting points for the next iteration (bogus lifts) and have to be discarded. This causes a **solution explosion problem**, as shown in the example below.

Example V.2: Consider the following system of polynomial congruences (from [14]) to be solved:

$$f_1 : 3x^2y + 7x \equiv 0 \bmod 16$$
$$f_2 : 2xy + 13y^2 + 3 \equiv 0 \bmod 16$$

First we solve $f_1, f_2 \equiv 0 \pmod 2$ and obtain solutions $\overline{x}_1 = \{[0,1],[1,1]\}$. The corresponding Jacobian has no inverse (mod 2). Therefore, both solutions lift (non-uniquely) as $\overline{x}_2 = \overline{x}_1 + 2 \cdot \overline{t}$, where $\overline{t} = [t_1, t_2], 0 \leq t_1, t_2 < 2$. We see that $\overline{x}_1$ lifts to $\{[0,1],[0,3],[2,1],[2,3],[1,1],[1,3],[3,1],[3,3]\}$ as potential solutions (mod 4). However, out of these only $\overline{x}_2 = \{[0,1],[0,3]\}$ are valid solutions (mod 4). For example, the (lifted) point $[x = 2, y = 1]$ is a solution to $f_2 \equiv 0 \pmod 4$, but not a solution to $f_1 \equiv 0 \pmod 4$. Similarly, $[2,3]$ is also not a solution to $f_1 \equiv 0 \pmod 4$. Therefore, every such (non-unique) lift has to be checked for validity, and all bogus lifts have to be discarded.

The above example shows the potential for solution explosion. To overcome the solution explosion problem, and to compute only the valid solutions (lifts) at every iteration, we do not apply the (brute-force) recurrence formula of Eqn. (20). Instead, we directly solve Eqn. (18) – i.e. "given $\overline{x}_{k-1}, \overline{f}(\overline{x}_{k-1})$, and $J(\overline{x}_{k-1})$, find $\overline{t}$ such that Eqn. (18) is satisfied." Since $\overline{x}_{k-1}$ is known, $\overline{f}(\overline{x}_{k-1})$ and $J(\overline{x}_{k-1})$ are a vector and a matrix of constants, respectively. This implies that Eqn. (18) is a system of **linear congruences** (mod 2^k); efficient algorithmic solutions for which are known in literature [24].

For our implementation, we formulate the problem as shown in Eqn. (18), using the commercially available constraint-programming engine of ILOG (**ILOG CP Optimizer**). We obtain only those solutions to $\overline{t}$ that lift $\overline{x}_{k-1}$ to $\overline{x}_k$ correctly; thus avoiding solution explosion – to an extent.

Lifting with Composite Moduli: We encounter variables in Z_2 (Booleans), in $Z_{2^m}, Z_{2^{m+1}}$ and so on. Our approach scales the Boolean variable from (mod 2) to (mod 2^m). If we were to then apply the recurrence formula of Eqn. (20), the scaled Boolean variable would lead to a large number of bogus lifts. In contrast, use of the constraint-programming engine allows us to restrict the lifting of variables according to their respective ranges. For example, let $x_i \in Z_{2^{n_i}}$, and assume that we have lifted the solutions to (mod 2^{n_i}). For the subsequent iterations, we can set the corresponding value of $t_i = 0$ in Eqn. (18), so that the value of x_i does not go out of range.

VI. Experiments

We have implemented the Newton's lifting based approach to solve a system of polynomial equations mod 2^m by integrating the symbolic algebra engine of MATHEMATICA together with ILOG's constraint-programming engine library. We have also implemented a parser in JavaCC that: i) parses the word-level arithmetic design descriptions in Verilog; ii) generates a data-flow graph and records the bit-vector computations and word-lengths; and iii) translates them into corresponding polynomial function constraints over $f : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$. For equivalence checking, we add the $\neq$ constraint as derived in Eqns. (5) or (7). These polynomial equations are subsequently scaled to a uniform modulus 2^M, and are ready to be solved. A proof of unsatisfiability verifies the equivalence, whereas a solution corresponds to a bug. Solutions (mod 2) are generated using the constraint-programming engine, the Jacobian and its inverse is computed using MATHEMATICA and then we apply our lifting approach.

We have tested our approach on a number of designs collected from a variety of benchmark suites, as shown in Table I. The first example is an anti-aliasing function used in MP3 decoders that computes $F = \frac{1}{2\sqrt{a^2+b^2}}$. The implementation uses a degree-6 polynomial approximation implemented in 16-bit fixed-point arithmetic. PolyUnopt design is borrowed from [25]; it is a degree-4 polynomial, factored as a Horner form to be mapped into a multiply-add-accumulate (MAC) unit. The RGB designs are image processing applications, Quad and 4^{th} order filters are Volterra models of polynomial signal processing applications. MiBench example is from [26] and is used in automotive applications.

For equivalence checking, the two RTL descriptions to be verified are symbolically different but computationally equivalent. The original designs were first modified by adding vanishing polynomials [9], and were then processed for common-subexpression elimination technique of [27]. The number of input bit-vector variables, number of adders and multipliers, and the output bit-vector size are shown in column 2. In column 3, under the "Miter-2" heading, we use the constraint of Eqn. (7) to model the miter at the output; whereas in column 4, we report the result using Eqn. (5) to model the miter. We have also performed experiments with the zChaff, MiniSAT, PicoSAT SAT solvers. We synthesized the design using the Synopsys design compiler, generated a netlist and translated the CNF-constraints. The designs were mitered and then given to the SAT solver. The RGB applications are only 10-bit datapaths with degree-2 polynomials, and the SAT solvers were able to prove equivalence quickly. For the other larger designs, our approach is faster. For the Quad filter, we found that the system of congruences had too many solutions – which created the solution explosion problem and hence required a large amount of time to lift every solution.

TABLE I

Comparison of time taken by various approaches

Benchmark	Specs Var/+/*/m	Miter-2 Time (s)	Miter-1 Time(s)	zChaff Time	MiniSat Time	PicoSAT Time(s)
Anti-alias	1/4/5/16	11	8	>1000s	>1000s	32s
Poly_Unopt	1/3/7/16	13	9	>1000s	966	23s
RGB 1	3/3/4/10	15	7	6.3	10.62	2s
RGB 2	3/4/6/11	19.047	13	11	14	14s
RGB 3	3/3/6/10	10	8	7	10	9
Quad Filter	2/4/8/16	2103	1294	N/A	N/A	N/A
4^{th} Order	1/4/9/16	567	10	>1000	>1000	25
MiBench	2/8/7/8	321	N/A	N/A	N/A	N/A
Quad Filter (bug)	2/4/8/16	597/1335	419/2104	1s	1s	<1s
Anti-alias (bug)	1/4/5/16	1s	3s	1s	1s	<1s

We then changed one of the adders in one design to a "saturation adder" and found the bug – e.g. using the Miter-2 model, the first bug was found in 597s, whereas all the bugs (42 solutions) were enumerated in 1335s. Overall, for these class of designs that have a large number of multiply and other non-linear operations, our approach appears to be competitive against the SAT-based approach.

Limitations: The main limitation that we have observed is the problem of solution explosion. While in our implementation, we have attempted to mitigate this problem to an extent, the problem still remains, particularly for the equivalence checking setup. Due to this problem, we have not been able to verify designs containing larger than 16-bit vectors.

VII. Conclusions and Future Work

In this paper, we have presented an approach to solving bit-vector arithmetic using concepts from number-theory and commutative algebra. We show that bit-vector arithmetic can be modeled, as constraints, by a system of polynomial functions with composite moduli. We show how such non-linear modulo-arithmetic congruences can be solved using Newton's p-adic iterations. We have implemented our approach within an integration of Mathematica and ilog cp-optimizer, and demonstrated how it can be applied to verify arithmetic datapath computations. Results show the power and the limitations of our approach against those of contemporary SAT solvers. As part of future work, we are investigating the use of Grobner bases to speed-up polynomial constraint solving.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Yhu, "Symbolic Model Checking without BDDs", *in TACAS*, pp. 193–207, 1999.

[2] Y. Xie and A. Aiken, "Scalable error detection using boolean satisfiability", *in ACM Symp. Prog. Lang.*, pp. 351–363, 2005.

[3] B. Dutertre and L. De Moura, "The YICES SMT Solver", http://yices.csl.sri.com/tool-paper.pdf, 2006.

[4] Z. Zeng, P. Kalla, and M. J. Ciesielski, "LPSAT: A Unified Approach to RTL Satisfiability", *in Proc. DATE*, 2001.

[5] R. Bruttomesso, A. Cimmatti, and *et al.*, "A Lazy and Layered SMT(BV) Solver for Hard Industrial Verification Problems", *in Proc. CAV*, LNCS, pp. 247–260. Springer-Verlag, 2007.

[6] V. Ganesh and D. Dill, "A Decision Procedure for Bit-Vectors and Arrays", *in Proc. (CAV)*, LNCS. Springer-Verlag, 2007.

[7] D. Cyrluk, O. Moller, and H. Ruess, "An Efficient Procedure for the Theory of Fixed-Size Bitvectors", *in Proc. Computer-Aided Verification*, vol. 1254, 1997.

[8] R. Brant, D. Kroening, and *et al*, "Deciding Bit-Vector Arithmetic with Abstraction", *in Proc. TACAS*, pp. 358–372, 2007.

[9] N. Shekhar, P. Kalla, and F. Enescu, "Equivalence Verification of Polynomial Datapaths using Ideal Membership Testing", *IEEE Trans. CAD*, vol. 26, pp. 1320–1330, July 2007.

[10] N. Shekhar, P. Kalla, M. B. Meredith, and F. Enescu, "Simulation Bounds for Equivalence Verification of Polynomial Datapaths using Finite Ring Algebra", *IEEE Trans. VLSI*, vol. 16, pp. 376–387, 2008.

[11] N. Shekhar, P. Kalla, F. Enescu, and S. Gopalakrishnan, "Equivalence Verification of Polynomial Datapaths with Fixed-Size Bit-Vectors using Finite Ring Algebra", *in International Conference on Computer-Aided Design, ICCAD*, pp. 291–296, November 2005.

[12] Z. Chen, "On polynomial functions from $Z_{n_1} \times Z_{n_2} \times \cdots \times Z_{n_r}$ to Z_m", *Discrete Math.*, vol. 162, pp. 67–76, 1996.

[13] R. Zippel, *Effective Polynomial Computation*, Kluwer Acad. Pub., 1993.

[14] D. Babic and M. Musuvathi, "Modular Arithmetic Decision Procedure", Technical Report TR-2005-114, Microsoft Research, 2005.

[15] Aaron Stump, Clark W. Barrett, and David L. Dill, "CVC: A Cooperating Validity Checker", *in Proc. CAV*, vol. 2404, pp. 500–504, 2002.

[16] M. Wedler, D. Stoffel, and W. Kunz, "Normalization at the arithmetic bit level", *in Proc. Design Auto. Conf.*, pp. 457–462, 2005.

[17] C. W. Barlett, D. L. Dill, and J. R. Levitt, "A Decision Procedure for bit-Vector Arithmetic", *in DAC*, June 1998.

[18] M.O. Moller and H. Ruess, "Solving Bit-Vector Equations", *in Proc. FMCAD'98*, pp. 36–48, 1998.

[19] F. Fallah, S. Devadas, and K. Keutzer, "Functional Vector Generation for HDL models using Linear Programming and Boolean Satisfiability", *in Proc. DAC*, '98.

[20] R. Brinkmann and R. Drechsler, "RTL-Datapath Verification using Integer Linear Programming", *in Proc. ASP-DAC*, 2002.

[21] C.-Y. Huang and K.-T. Cheng, "Using Word-Level ATPG and Modular Arithmetic Constraint Solving Techniques for Assertion Property Checking", *IEEE Trans. CAD*, vol. 20, pp. 381–391, 2001.

[22] G. Nelson and D. Oppen, "Simplification by cooperating decision procedures", *ACM Trans. on Programming Languages and Systems*, vol. 1, pp. 245–257, 1979.

[23] M. Bhargava, "Congruence Preservation and Polynomial Functions from z_n to z_m", *Discrete Mathematics*, vol. 173, pp. 15–21, 1997.

[24] M. Müller-Olm and H. Seidl, "Analysis of modular arithmetic", *ACM Trans. Program. Lang. Syst.*, vol. 29, pp. 29, 2007.

[25] A. K. Verma and P. Ienne, "Improved use of the Carry-save Representation for the Synthesis of Complex Arithmetic Circuits", *in Proceedings of the International Conference on Computer Aided Design*, 2004.

[26] M. R. Guthaus and et al., "Mibench: A Free, Commercially Representative Embedded Benchmark Suite", *in IEEE 4th Annual Workshop on Workload Characterization*, Dec 2001.

[27] A. Hosangadi, F. Fallah, and R. Kastner, "Factoring and eliminating common subexpressions in polynomial expressions", *in ICCAD*, pp. 169–174, 2004.