EQUIVALENCE VERIFICATION OF POLYNOMIAL DATAPATHS WITH FIXED-SIZE BIT-VECTORS USING FINITE RING ALGEBRA

Namrata Shekhar*, Priyank Kalla*, Florian Enescu† and Sivaram Gopalakrishnan*

*Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT-84112
{shekhar, kalla, sgopalak}@eng.utah.edu

†Department of Mathematics and Statistics
Georgia State University, Atlanta, GA 30302-4038
fenescu@mathstat.gsu.edu

***Abstract*— This paper addresses the problem of equivalence verification of RTL descriptions. The focus is on datapath-oriented designs that implement polynomial computations over fixed-size bit-vectors. When the size (m) of the entire datapath is kept constant, fixed-size bit-vector arithmetic manifests itself as polynomial algebra over finite integer rings of residue classes Z_{2^m}. The verification problem then reduces to that of checking equivalence of multi-variate polynomials over Z_{2^m}. This paper exploits the concepts of polynomial reducibility over Z_{2^m} and derives an algorithmic procedure to transform a given polynomial into a unique canonical form modulo 2^m. Equivalence testing is then carried out by coefficient matching. Experiments demonstrate the effectiveness of our approach over contemporary techniques.**

I. INTRODUCTION

RTL descriptions of integer datapaths that implement polynomial arithmetic are found in many practical designs, particularly in digital signal processing (DSP) for audio, video and multimedia applications. Such designs perform a sequence of ADD, MULT, SHIFT type of algebraic computations that can be modeled as *multi-variate polynomials of finite degree*. Initial algorithmic specifications of such systems involve data representation using floating-point formats. However, they are often implemented with fixed-point architectures in order to optimize the area, delay and power related costs of the implementations. In many cases, the design choice is that of a single, uniform system word-length for the computations. Such fixed-size datapath computations are generally implemented using: signal truncation, rounding or saturation arithmetic.

This paper addresses the problem of *RTL equivalence verification* of datapath descriptions that implement polynomial computations over fixed-size bit-vectors by way of *signal truncation*. In such designs, m-bit adders and multipliers produce an m-bit output; only the lower m-bits of the outputs are used and the higher-order bits are ignored. When the datapath size (m) over the entire design is kept constant, then fixed-size bit-vector arithmetic manifests itself as *polynomial algebra over finite integer rings* of residue classes Z_{2^m}; *i.e.* addition and multiplication is closed within the finite set of integers $\{0, \ldots, 2^m - 1\}$. In such cases, symbolically distinct polynomials (those with different degrees and coefficients) can become computationally equivalent. The equivalence verification problem then reduces to that of proving the *computational equivalence:* $f(x_1, \ldots, x_d)\%2^m \equiv g(x_1, \ldots, x_d)\%2^m$, where f, g are polynomials in d variables $x_1, \ldots, x_d$, and $m =$ datapath size; in other words, to prove $f(x_1, \ldots, x_d) \equiv g(x_1, \ldots, x_d)$ in $Z_{2^m}^d$.

Sponsored in part by a grant from NSF contract No. CCF-0514966.

A. Motivating the Verification Problem

Let us motivate the equivalence verification problem as it appears in the context of our work. Fig. 1 depicts a typical design flow for DSP applications. The floating-point MATLAB model is automatically converted to a fixed-point model; which is subsequently translated into RTL. Automatic translation utilities are available for this purpose [1]. The verification problem instance is that of checking the equivalence of the fixed-point design against the translated (and optimized) RTL models.

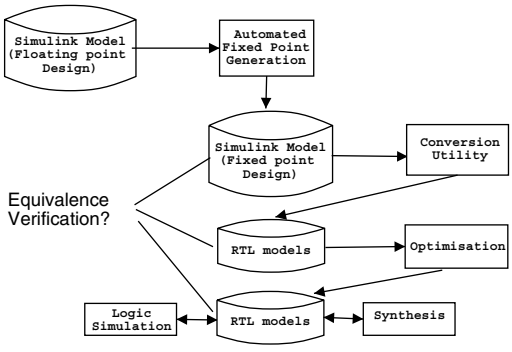

Fig. 1. The Equivalence Verification problem: Matlab to RTL flow.

As an example, consider the anti-alias function of an MP3 decoder that computes [2]: $F = \frac{1}{2\sqrt{a^2+b^2}}$ under the assumption that $a^2 + b^2 > 0$. Computing $x = a^2 + b^2$, the function can be implemented as $F = \frac{1}{2\sqrt{x}}$. The given equation can be approximated using the Taylor series expansion for a range of x based on the given application. Under the constraint that the datapath size is to be fixed to 16-bits, the computation

$F[15:0]$ (scaled appropriately) can be represented in RTL as:

$$\begin{aligned} F[15:0] &= 156(X[15:0])^6 + 62724(X[15:0])^5 \\ &\quad +17968(X[15:0])^4 + 18661(X[15:0])^3 \\ &\quad +43593(X[15:0])^2 + 40224(X[15:0]) \\ &\quad +13281 \end{aligned}$$

Application of high-level synthesis or symbolic algebra based manipulation techniques [2] [3] to the above may transform the RTL implementation to:

$$\begin{aligned} G[15:0] &= 156(X[15:0])^6 + 5380(X[15:0])^5 \\ &\quad +1584(X[15:0])^4 + 10469(X[15:0])^3 \\ &\quad +27209(X[15:0])^2 + 7456(X[15:0]) \\ &\quad +13281 \end{aligned}$$

Note that polynomially, $F \neq G$ because they have different coefficients; but because the datapath size is fixed to 16 bits $F[15:0] \equiv G[15:0]$, or in other words $F\%2^{16} \equiv G\%2^{16}$.

So how do we prove that the above computations are indeed equivalent? An algorithmic solution to this problem is the subject of this paper.

II. Limitations of Contemporary Approaches

It is evident that Binary Decision Diagrams (BDDs) [4], Binary Moment Diagrams (BMDs) [5], K*BMDs [6] and their derivatives are ill-suited for our application; mostly due to the presence of high-degree polynomial computations over wide bit-vectors. TEDs [7] have been proposed as canonical DAG representations for multi-variate polynomials. However, TEDs do not model modulo-arithmetic and thus cannot prove polynomial equivalence over finite integer rings.

Modulo arithmetic concepts have been studied in the context of RTL verification for bit-vector arithmetic [8] [9], word-level ATPG [10] and MILP-based simulation vector generation [11]. However, these are mostly geared toward *solving linear congruences* under modulo arithmetic - a different application from *proving polynomial equivalence* modulo 2^m. There exist various applications (such as FIR, IIR, Kalman, Elliptical wave filters, FFT, etc.) whose RTL computations have been verified using: co-operative decision procedures, theorem provers (HOL), term-rewriting, and congruence closure based techniques [12]. DSP implementations of the above applications are mostly linear and/or multi-linear forms - which are easy to verify. However, for polynomial equivalence in Z_{2^m}, such approaches are not very efficient.

Within the scope of Symbolic Computer Algebra, tools such as [13] [14] do provide algorithmic solutions to polynomial equivalence over a variety of rings. However, these solutions are available for fields (R, Q, C), prime rings Z_p, integral and Euclidean domains - collectively called the unique factorization domains (UFDs). Within UFDs, computer algebra systems solve the equivalence checking problem by *uniquely* factorizing an expression into *irreducible* terms and comparing the coefficients of the factored terms ordered lexicographically. Efficient algorithms for factorization have been developed [15] [16], which can be readily used for this purpose. However, in the case of our application, the finite integer ring formed by specific modulo value 2^m is a non-UFD, due to the presence of zero divisors (*e.g.*, $4 \neq 2 \neq 0, 4 \cdot 2 = 0$ in Z_8). Since Z_{2^m} is a non-UFD, any polynomial in $Z_{2^m}^d$ cannot be uniquely factorized into irreducible terms. For example, consider $f(x) = x^2 - x$ in the non-UFD Z_6; f factorizes in two (non-unique) irreducible forms: $(x)(x-1)$ and $(x-3)(x-4)$. On the same lines, techniques using the concepts of Grobner's bases [17] [2] find extensive application in UFDs. However, for the above reasons, they cannot be directly ported to solve the above problem in the non-UFD Z_{2^m}. The symbolic algebra libraries ZEN [18] and NTL [19] allow for polynomial manipulation (factorization, multiplication, primality testing etc.) over rings of the type $Z_n, n =$ integer, as well as over their polynomial extensions. However, to the best of our knowledge, a "ready-made" algorithmic procedure to test $f(x_1, \ldots, x_d)\%2^m \equiv g(x_1, \ldots, x_d)\%2^m$ is not available.

A. Related Work in Number Theory & Polynomial Algebra

The problem $f(x_1, \ldots, x_d)\%n \equiv g(x_1, \ldots, x_d)\%n$ is known to be NP-hard when $n \geq 2$ [20]. Researchers from the field of number theory and commutative algebra have analyzed properties of polynomials over arbitrary finite integer rings. Singmaster [21] presented the theory of *univariate vanishing polynomials* over $Z_n, n \in N, n > 1$; *i.e.* those polynomials f such that $f(x)\%n \equiv 0$. For example, $2x^2 + 2x \equiv 0\%4, \forall x \in Z_4$; hence $2x^2+2x$ is a vanishing polynomial in Z_4. He identified necessary and sufficient conditions for a univariate polynomial to vanish $\%n$. The equivalence test for $f(x) \equiv g(x)$ in Z_n can then be re-formulated as determining whether $(f(x) - g(x))\%n \equiv 0$. However, in digital design, we mostly encounter multi-variate polynomials. Hungerbuhler and Specker [22] extend the concepts from [21] and *derive a unique/canonical form representation of a multi-variate polynomial over finite integer rings of the form* Z_{p^m}, where p is any prime integer. Their result suits our application as in our case $p = 2$.

The main contributions of this paper are: i) We formulate the fixed-vector-size (m) RTL datapath verification problem as polynomial equivalence in Z_{2^m}; ii) From the concepts presented in [22], we *derive a systematic algorithmic procedure* that operates on the given polynomials in Z_{2^m} and reduces them to a unique canonical form; iii) We extract the data-flow graphs (DFG) corresponding to the given RTL descriptions and construct their polynomial representations by traversing the DFGs from inputs to outputs. The polynomials are then reduced to their canonical forms and the equivalence check is performed by coefficient-matching. iv) Experimentally, we demonstrate that the proposed approach is able to verify the equivalence of high-degree polynomial RTL datapaths (real-world benchmarks), where contemporary methods prove to be impractical.

In the rest of the paper, we use the notation $Z_{2^m}[x_1, \ldots, x_d]$ or $Z^d_{2^m}$, to denote the **ring of polynomials** $\%2^m$ over the d variables $x_1, \ldots, x_d$. Polynomial addition and multiplication is performed $\%n$ $(n = 2^m)$ according to the rules below.

$$(a+b)\%n = (a\%n + b\%n)\%n \quad (1)$$
$$(a \cdot b)\%n = (a\%n \cdot b\%n)\%n \quad (2)$$
$$(-a)\%n = (n - a\%n)\%n \quad (3)$$

III. Vanishing Polynomials over Finite Rings

It is a well-known result in number theory that for any $n \in N$, $n!$ divides the product of n consecutive numbers. For example, $4!$ divides $4 \times 3 \times 2 \times 1$. But this is also true of any n consecutive numbers: $4!$ also divides $99 \times 100 \times 101 \times 102$. Consequently, it is possible to find the **least** $k \in N$ such that $n|k!$. This value k corresponds to the *Smarandache function, SF(n)*[23]. In the ring of interest, Z_{2^m}, let $SF(2^m) = k$, such that $2^m|k!$. As an example, $SF(2^3) = 4$ as 8 divides $4! = 4 \times 3 \times 2 \times 1 = 2^3 \times 3$. Note that 8 does not divide $3!$, and hence the least $k = 4$.

This property can be utilized to treat the equivalence problem as a divisibility issue in Z_{2^m}, i.e. $f - g \equiv 0 \Rightarrow 2^m|(f-g)$. In Z_{2^3}, let $8|(f-g)$. But, $8|4!$ too. Therefore, if $(f-g)$ can be represented as a product of 4 consecutive numbers, then $(f-g)$ would vanish in Z_{2^3}. So, what is a natural example of such a polynomial? The answer is $(x+1)(x+2)(x+3)(x+4)$. In this regard, Singmaster [21] proposed a set of monic polynomials (with leading coefficient = 1), S_k, where each S_i represents (in polynomial form) a product of i consecutive numbers; $S_0(x) = 1$, $S_1(x) = x + 1$, …, $S_k(x) = (x + k) \cdot S_{k-1}(x)$. Any expression in $Z_{2^m}[x]$ that can be factored into at least S_k (where $k = SF(2^m)$), will be divisible by 2^m and vanish.

Example 3.1: Consider a polynomial computation p in variable x over Z_{2^8} (representing an 8-bit polynomial datapath).

$$p = x^{10} + 55x^9 + 40x^8 + 230x^7 + 77x^6 + 167x^5 + 98x^4 + 156x^3 + 168x^2 + 32x$$

In Z_{2^8}, $SF(2^8) = 10$. If p can be factorized into a product of 10 consecutive numbers (or $S_{10}(x)$), then p is a vanishing polynomial. Indeed, in Z_{2^8}, p can be written as $\prod_{i=1}^{10}(x+i) = \begin{pmatrix} x+10 \\ 10 \end{pmatrix} 10!$.

When a polynomial cannot be factored into such S_k expressions, can it still vanish? Consider the quadratic polynomial $4x^2+4x$ in Z_8. It can be written as $4(x+2)(x+1)$. However, $4x^2+4x$ cannot be factorized as $\prod_{i=1}^{4}(x+i) = (x+4)(x+3)(x+2)(x+1)$. The missing factors, $(x+4)(x+3)$ in this case, are compensated for by the multiplicative constant 4; therefore, $4x^2+4x \equiv 0\%8$. Singmaster identified the constraints on such multiplicative constants such that the polynomial in question would vanish. We state the following result.

Lemma 3.1: The expression $b \cdot S_k(x) \equiv 0$ in $Z_{2^m}[x]$ if and only if $\frac{2^m}{(k!,2^m)}|b$; where $(k!, 2^m)$ is the greatest common divisor (GCD) of $k!$ and 2^m, $b \in Z_{2^m}$ and $S_k(x)$ is as defined above.

Example 3.2: Let us explain the above concept with the help of the previous example. Let $p(x) = 4x^2 + 4x$ in Z_{2^3}. Note that $p(x) = 4(x+2)(x+1) = 4 \cdot S_2(x)$. Therefore, in this case, $b = 4, k = 2$; and $\frac{2^3}{(2!,2^3)}$ (=4) divides b (=4). Because the above condition is satisfied, $p(x)\%2^3 \equiv 0$. Note that, if b were replaced by 3, then $p(x) = 3(x+2)(x+1)$ would not be a vanishing polynomial as $\frac{2^3}{(2!,2^3)}$ does not divide 3.

Singmaster extended this result to develop a canonical representation of a univariate polynomial that vanishes over any finite integer ring. We have studied this work and applied it to verification of univariate polynomial datapaths in [24].

The above concept of vanishing polynomials leads to the concept of *reducibility*. For example, in Z_{2^3}, $4x^2+4x \equiv 0 \Rightarrow 4x^2 \equiv -4x\%8 \Rightarrow 4x^2 \equiv (8-4)x\%8 \Rightarrow 4x^2 \equiv 4x\%8$. In other words, $4x^2$ can be *reduced* to $4x$. Hungerbuhler and Specker [22] extend the concept of polynomial reduction to the multi-variate case. In the next section, we present the necessary theoretical foundation to perform the requisite reductions on multi-variate polynomials.

IV. Reducibility of Multi-variate Polynomials

We use the following multi-index notation in the rest of the paper [22]: $\mathbf{k} =< k_1, k_2, \ldots, k_d >$ are the degrees corresponding to the d variables $\mathbf{x} =< x_1, x_2, \ldots, x_d >$, respectively.

$$\mathbf{x}^{\mathbf{k}} = \prod_{i=1}^{d} x_i^{k_i}$$
$$\mathbf{k}! = \prod_{i=1}^{d} k_i!$$
$$\begin{pmatrix} \mathbf{x} \\ \mathbf{k} \end{pmatrix} = \prod_{i=1}^{d} \begin{pmatrix} x_i \\ k_i \end{pmatrix}$$

In the above notation, the monomial $4{x_1}^2 x_2$ can be represented in the above notation as $a\mathbf{x}^{\mathbf{k}} \equiv a{x_1}^{k_1}{x_2}^{k_2}$, where, $a = 4$, $k_1 = 2$, $k_2 = 1$, $\mathbf{k} =< 2, 1 >$ and $\mathbf{k}! = 2! \cdot 1! = 2$.

We consider the following results. The proofs are available in [22] and are not reproduced.

Lemma 4.1: Let a be an element of Z_{2^m}. If $2^m|a\mathbf{k}!$ then the polynomial $q(\mathbf{x}) = a\mathbf{k}! \begin{pmatrix} \mathbf{x+k} \\ \mathbf{k} \end{pmatrix}$ is a vanishing polynomial over Z_{2^m} and the term of maximal degree is $a\mathbf{x}^{\mathbf{k}}$.

Example 4.1: Consider the polynomial $f_1(x) = 4x^2+4x \in Z_{2^3}$ of degree 2. f can be represented as $4 \cdot 2! \cdot \begin{pmatrix} x+2 \\ 2 \end{pmatrix}$. Here, $a = 4$ and $\mathbf{k}! = k_x! = 2$ and clearly, $2^3|4 \cdot 2!$. Now consider another polynomial in two variables $f_2(x, y) = 4x^2y + 4xy + 4x^2 + 4x$ in Z_8 which is equivalently written as $4 \cdot 2! \cdot \begin{pmatrix} x+2 \\ 2 \end{pmatrix} \begin{pmatrix} y+1 \\ 1 \end{pmatrix}$. Here, $a = 4$ and $\mathbf{k}! = k_x! \cdot k_y! =$

$2! \cdot 1! = 2$, and $2^3 | 4 \cdot 2$. Based on Lemma 4.1, both f_1 and f_2 are vanishing polynomials.

Lemma 4.2: $a\mathbf{x^k}$ is reducible iff $2^m | a\mathbf{k}!$.

Lemma 4.2 provides the necessary and sufficient conditions to reduce a given monomial ($a\mathbf{x^k}$) in d variables over $Z_{2^m}^d$. Using the two Lemmas, reduction is carried out by subtracting a vanishing polynomial of the same total degree. In other words, if $2^m | a\mathbf{k}!$, then $a\mathbf{x^k} = a\mathbf{x^k} - q(\mathbf{x})$. The resulting polynomial now has a lesser total degree (though it may have more terms).

Example 4.2: Consider the term $4x^2y$ in f_2 from Example 4.1. Reduction in Z_{2^3} is carried out by subtracting a vanishing polynomial of degree-2 in x and degree-1 in y, given by Lemma 4.1. Here, $\mathbf{x} =< x, y >$ and $\mathbf{k} =< 2, 1 >$.

$$\begin{aligned} 4x^2y &= 4x^2y - 4 \cdot 2! \cdot \begin{pmatrix} x+2 \\ 2 \end{pmatrix} \begin{pmatrix} y+1 \\ 1 \end{pmatrix} \\ &= 4x^2y - 4(x+2)(x+1)(y+1) \\ &= 4xy + 4x^2 + 4x \end{aligned}$$

In the above expression, $4x^2$ can be further reduced to $4x$ (as $4x^2 + 4x \equiv 0 \% 2^3$). Therefore, $4x^2y \equiv 4xy + 4x + 4x \equiv 4xy \% 2^3$.

A. Canonical representation of polynomials modulo 2^m

The above concepts of monomial reductions can be applied to a given polynomial, iteratively, and the polynomial can be reduced to a minimal, unique canonical form. For this purpose, we first define the following concept.

Definition 4.1: We define $\nu_2(\mathbf{k}!)$ as the maximum degree x such that 2^x divides $\mathbf{k}!$:

$$\nu_2(\mathbf{k}!) = max\{x \in N : 2^x | \mathbf{k}!\}$$

For example, $\nu_2(4!) = 3$. Note that $\nu_2(k!)$ gives *the number-of-factors-2* in $k!$.

The above results lead to the following canonical form for polynomials in $Z_{2^m}^d$ [22].

Theorem 1: Every polynomial f in $Z_{2^m}^d$ has a unique representation of the form

$$f = \Sigma_{\mathbf{k} \in N^d} \ \alpha_{\mathbf{k}} \mathbf{x^k} \tag{4}$$

where $\nu_2(\mathbf{k}!) < m$ and $\alpha_{\mathbf{k}} \in \{0, 1, \cdots, 2^{m - \nu_2(\mathbf{k!})} - 1\}$.

Proof: While the proof of this theorem is provided in [22], we highlight the key concepts as they allow us to derive a systematic algorithmic procedure to reduce a given polynomial to the above canonical form.

The representation exists: Let $a\mathbf{x^k}$ be the monomial of the highest total degree that appears in f. If 2^m divides $a\mathbf{k}!$, then from Lemma 4.2 it can be reduced and replaced with a polynomial of lower degree. Such a reduction can be applied to all monomials $a\mathbf{x^k}$ if $2^m | a\mathbf{k}!$.

Now, lets us consider those terms in f where 2^m does not divide $a\mathbf{k}!$. This guarantees that 2^m does not divide $\mathbf{k}!$ and therefore, $\nu_2(\mathbf{k}!) < m$. Now, let us analyze the coefficient a. Let $s = \nu_2(\mathbf{k}!)$. Since, 2^m does not divide $a\mathbf{k}!$, 2^{m-s} cannot divide a (otherwise, 2^m would divide $a\mathbf{k}!$). Therefore, the coefficient a can be divided by 2^{m-s} as follows:

$$a = q \cdot 2^{m-s} + r \tag{5}$$

where q is the quotient and r is the remainder. Moreover, $0 \leq r < 2^{m-s}$. So

$$a\mathbf{x^k} = q \cdot 2^{m-s} \cdot \mathbf{x^k} + r \cdot \mathbf{x^k} \tag{6}$$

Note that the term $q \cdot 2^{m-s} \cdot \mathbf{x^k}$ is again reducible, from Lemma 4.2. The second term, $r \cdot \mathbf{x^k}$, is already in reduced form since $r < 2^{m-s} = 2^{m - \nu_2(\mathbf{k!})}$.

The representation is unique: It suffices to argue that for any two non-unique equivalent polynomials f and g, $f - g = \Sigma_{\mathbf{k} \in N^d} \ \alpha_{\mathbf{k}} \mathbf{x^k} \simeq 0$ implies that all coefficients are zero. ■

Example 4.3: Let us re-visit Example 4.2: the monomial $4x^2y$ in Z_{2^3}. In this case, $\mathbf{k}! = 2$. Moreover, $\alpha_{\mathbf{k}} = 4$ and $\nu_2(\mathbf{k}!) = \nu_2(2!) = 1$. Clearly $\alpha_{\mathbf{k}} \notin \{0, 1, \cdots, 2^{3-1} - 1\} = \{0, 1, 2, 3\}$. Hence, it is not in canonical form and can be reduced by subtracting a vanishing polynomial, as shown in Example 4.2. The resulting monomial $4xy$ has $\alpha_{\mathbf{k}} = 4 \in \{0, 1, \cdots, 2^{3-0} - 1\} = \{0, 1, \cdots, 7\}$. This is the minimal unique form representation and further reduction is not possible.

Now consider the monomial $5x^2y$ in Z_{2^3}. Here, $\alpha_{\mathbf{k}} = 5$ and $\nu_2(2!) = 1$. Note that, 2^3 does not divide $5 \cdot 2!$. Moreover, $\alpha_{\mathbf{k}} = 5 \notin \{0, 1, 2, 3\}$. Therefore, we represent $5x^2y = 4x^2y + x^2y$. As shown above, $4x^2y$ can be reduced to a lower total degree and x^2y is already in reduced form.

Note that the proof of the above theorem allows us to derive an algorithmic procedure to reduce a given polynomial to its unique canonical form. The procedure operates as follows:

1) Order the terms in descending *term-order* [2] of their highest total degree.
2) For the highest degree term, if $2^m | a\mathbf{k}!$, reduce it.
3) Otherwise, check to see if the coefficient $\alpha_{\mathbf{k}} \in \{0, 1, \cdots, 2^{m - \nu_2(\mathbf{k!})} - 1\}$. If yes, then the term cannot be reduced further and is in its canonical form. Otherwise, the coefficient can be reduced as shown in Example 4.3.
4) Apply the above procedure repeatedly to all monomial terms.

Note that the procedure converges. Ordering the terms in the first step above, ensures that a monomial of particular total degree is reduced only once. The algorithm is given in Algorithm 1 where the number of monomial reductions is bound by $O(\mathbf{k}^d)$, where $\mathbf{k}$ is the highest degree and d is the number of variables.

Example 4.4: Let us consider the anti-alias function introduced in Sec. I. Two different implementations of this function were correctly verified by the algorithm. A sample verification run for the polynomial F is shown below:

$F = 156 * x^6 + 62724 * x^5 + 17968 * x^4 + 18661 * x^3 + 43593 * x^2 + 40224 * x + 13281$

Reducing $62724 * x^5$ ...

$F = 156 * x^6 + 5380 * x^5 + 9776 * x^4 + 59621 * x^3 + 51785 * x^2 + 56608 * x + 13281$
Reducing $9776 * x^4$ ...
$F = 156 * x^6 + 5380 * x^5 + 1584 * x^4 + 43237 * x^3 + 27209 * x^2 + 40224 * x + 13281$
Reducing $43237 * x^3$ ...
$F = 156 * x^6 + 5380 * x^5 + 1584 * x^4 + 10469 * x^3 + 27209 * x^2 + 7456 * x + 13281$
F is now in canonical form.

```
POLY_REDUCE(poly, d, m, var_list)
poly = Multi-variate polynomial of bit-width m;
var_list = list of d constituent variables in poly;
Order the monomials in poly in decreasing term-order;
for each monomial mon in poly do
  a = Coefficient (mon)
  for each variable v_i in var_list do
    k_i = Degree(v_i) in mon;
  end for

  k! = ∏ k_i!;
  ν_2(k!) = Σ ν_2(k_i!);
  α_k = 2^{m−ν_2(k!)} − 1;

  if (2^m | a k!) then
    /*Monomial is degree-reducible: Subtract vanishing polynomial*/
    mon = mon − a * ∏_{j=1}^{d} ∏_{i=1}^{k_d} (v_j + i);
  else
    /*Condition from Theorem 1*/;
    if (a > α_k) then
      /*Monomial is coefficient-reducible*/

      quo = quotient[a/(α_k+1)]; rem = remainder[a/(α_k+1)];
      /*Quotient is degree-reducible. Subtract vanishing polynomial*/
      reduce_quo = quo(α_k+1)(∏_{j=1}^{d} v_j^{k_j} − ∏_{j=1}^{d} ∏_{i=1}^{k_j} (v_j + i));
      /*Build reduced monomial*/
      mon = reduced_quo + rem ∏_{j=1}^{d} (v_j^{k_j});
    end if
  end if
  Update poly with the reduced mon, if required;
  if (poly == 0) then
    return 0;
  end if
end for

return poly;
```

Algorithm 1: ALGORITHM POLY_REDUCE: Reduction of a given polynomial.

V. EXPERIMENTS

We have implemented Algorithm 1 described in Sec. IV in Perl with calls to MAPLE 7 [13] for all the algebraic manipulations. The data-flow graph for the given RTL descriptions is extracted using GAUT [25]. Traversing the DFG from the inputs to the outputs, the polynomial representations are constructed. The datapath size (m) is also recorded. The algorithm is applied to reduce the two polynomials to their canonical forms. Equivalency check is carried out by coefficient-matching.

We have tested our algorithm with a number of designs collected from a variety of benchmark suites, as shown in Table I. The first two examples [2] are phase-shift keying and anti-aliasing functions, both used in digital communication. The degree-3 and degree-4 filter designs [26] are Volterra models of polynomial signal processing applications. Horner forms of polynomials, used for faster signal processing, are from [3]. MIBench is a 9^{th}-degree polynomial from [27]. The last example is a vanishing polynomial of degree 10, specifically created to validate our algorithm. The time reported is the total time for canonizing both polynomials and that required for subsequent coefficient matching. The two descriptions to be verified are symbolically different but computationally equivalent. The number of variables, the highest degree that a variable appears in a term, and the datapath size are shown in column 2. For example, Savitzky-Golay filter has 5 variables, highest degree of each being 3, and bit-width = 16.

We have performed equivalence checking of the given RTL designs using BDDs, BMDs, SAT and MILP based approaches. Since gate-level descriptions are required by both BDDs and SAT, we synthesized our designs using a commercially available logic synthesis tool. BDDs were used to verify the the resulting netlists using the VIS[28] package. It was found that though BDDs could solve the problem for some of the smaller benchmarks (especially for univariate polynomials), they failed for the rest of the designs.

From the gate-level netlists corresponding to the two designs, we generated miter circuits and converted them to CNF format. ZChaff [29] was used to prove equivalence via unsatisfiability testing. For all the designs, ZChaff could not solve the problem within the time-limit of 500s. For equivalence via MILP solving [11], linear inequalities were created from their data-flow graphs. Equivalency $f \equiv g$ was tested via proving UNSAT ($f \neq g$). The reason why MILP failed was because linearizing X_m^k requires expanding it into its constituent m bits. LPSOLVE was used as the resolution engine. Since BMD packages are not available in public domain, the TED[7] package was suitably modified to construct BMDs. Note that BMD decomposition is a special case of TEDs; when all variables are Boolean, the TED reduces to a BMD. We attempted to construct the BMDs from the synthesized gate-level netlists corresponding to the original RTL descriptions. Because of the presence of high-degree polynomial terms, the graph could be constructed only for the smaller benchmarks. The other benchmarks could not be verified within the time-out limit.

The last benchmark is a "vanishing polynomial" in 2 variables. We wanted to verify that the outputs always compute zero. Interestingly, (commercial) logic synthesis tools generated a redundant, non-empty circuit. To verify that the circuit was indeed redundant, we attempted to construct both BDDs and BMDs, but were unable to do so. While BDDs ran out of

TABLE I

COMPARISON OF TIME TAKEN BY VARIOUS APPROACHES

Benchmark	Specs	Our approach	BDDs-VIS	BMD	SAT-ZChaff	MILP
	Var/Deg/m	Time (s)	Nodes/Time(s)	Nodes/Time(s)	Vars/Clauses/Time(s)	Time(s)
PSK	2/4/16	13.48	NA/>500	NA/>500	52K/142K/>500	>500
Anti-alias function	1/6/16	6.81	1.2M/34.2	NA/>500	3.9K/107K/>500	>500
Cubic filter	3/3/32	8.27	NA/>500	NA/>500	141K/417K/>500	>500
Degree-4 filter	3/4/16	18.72	NA/>500	NA/>500	60K/166K/>500	>500
Savitzky-Golay filter	5/3/16	13.51	NA/>500	NA/>500	69K/200K/>500	>500
MIBENCH	2/9/16	28.13	0.1M/13	NA/>500	15K/42K/>500	>500
Horner Polynomial 1	3/4/16	3.56	NA/>500	2355 / 85	13K/36K/>500	>500
Horner Polynomial 2	3/4/16	3.55	NA/>500	1574 / 47	12K/34K/>500	>500
Horner Polynomial 3	2/4/16	4.53	NA/>500	6803 / 246	25K/75K/>500	>500
Polynomial Unopt.	1/4/16	2.79	0.5M/9.1	2705 / 69	10K/28K/>500	>500
Vanishing polynomial	2/10/16	15.19	NA/>500	NA/>500	10K/29K/>500	>500

memory, BMD-composition operations did not terminate.

VI. CONCLUSIONS AND FUTURE WORK

This paper has presented a polynomial algebra based framework for equivalence verification of arithmetic datapaths. The targeted applications are polynomial computations implemented with fixed-size bit-vectors. The concept of polynomial reducibility over finite rings, Z_{2^m}, is exploited to transform a given polynomial to a unique canonical form. The equivalence checking is carried out using coefficient matching. A variety of benchmarks were verified using the proposed method. Our algorithm was able to solve the problem in all cases, where established techniques failed.

As part of future work, we are currently looking to extend the concepts presented in this paper to verify fixed-size datapaths that implement rounding schemes by ignoring the lower order bits. We would also like to explore verification of datapaths with multiple word-lengths.

REFERENCES

[1] I. A. Groute and Keane. K, "M(vh)dl: A Matlab to Vhdl Conversion Toolbox for Digital Control", *in IFAC Symp. on Computer-Aided Control System Design*, Sept., 2000.

[2] A. Peymandoust and G. DeMicheli, "Application of Symbolic Computer Algebra in High-Level Data-Flow Synthesis", *IEEE Trans. CAD*, vol. 22, pp. 1154–11656, 2003.

[3] 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.

[4] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, August 1986.

[5] R. E. Bryant and Y-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams", *in DAC*, 95.

[6] R. Dreschler, B. Becker, and S. Ruppertz, "The K*BMD: A Verification Data Structure", *IEEE Design & Test*, pp. 51–59, 1997.

[7] M. Ciesielski, P. Kalla, Z. Zheng, and B. Rouzyere, "Taylor Expansion Diagrams: A Compact Canonical Representation with Applications to Symbolic Verification", *in Proc. Design Automation and Test in Europe, DATE'02*, Mar 2002.

[8] D. Cyrluk, O. Moller, and H. Ruess, "An Efficient Procedure for the Theory of Fixed-Size Bitvectors", *in LCNS, CAV*, vol. 1254, 1997.

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

[10] 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.

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

[12] Z. Zhou and W. Burleson, "Equivalence Checking of Datapaths Based on Canonical Arithmetic Expressions", *in DAC*, 95.

[13] Maple, ", http://www.maplesoft.com.

[14] G. M. Greuel, G. Pfister, and H. Schonemann, "SINGULAR 2.0", A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, 2001, `http://www.singular.uni-kl.de`.

[15] E. Kaltofen, "Polynomial Factorization 1987-1991", *in Proc. LATIN'92 of Lect. Notes Comput. Sci.*, pp. 294–313, 1992.

[16] E. Kaltofen, "Polynomial Factorization 1982-1986", *in Computers in Mathematics of Lect. Notes in Pure and Applied Mathematics*, pp. 285–309, 1990.

[17] T. Becker and V. Weispfenning, *Grobner Bases: A Computational Approach to Commutative Algebra*, Springer-Verlag, 1993.

[18] F. Chabaud and R. Larcier, "A toolbox for fast computation in finite extension over finite rings", http://zenfact.sourceforge.net/.

[19] V. Shoup, "Ntl: A library for doing number theory", http://shoup.net/ntl/.

[20] O. H. Ibarra and S. Moran, "Probabilistic Algorithms for Deciding Equivalence of Straight-Line Programs", *Journal of the Association for Computing Machinery*, vol. 30, pp. 217–228, Jan. 1983.

[21] D. Singmaster, "On Polynomial Functions (mod m)", *J. Number Theory*, vol. 6, pp. 345–352, 1974.

[22] N. Hungerbuhler and E. Specker, "A Generalization of the Smarandache Function to Several Variables", *To appear in: Smarandache Notions Journal*, vol. 15.

[23] F. Smarandache, "A function in number theory", *Analele Univ. Timisoara, Fascicle 1*, vol. XVII, pp. 79–88, 1980.

[24] N. Shekhar, P. Kalla, F. Enescu, and S. Gopalakrishnan, "Exploiting vanishing polynomials for equivalence verification of fixed-size arithmetic datapaths", *in (to appear) ICCD*, 2005.

[25] Universite' de Bretagne Sud LESTER, "Gaut, Architectural Synthesis Tool", *http://lester.univ-ubs.fr:8080*, vol. , 2004.

[26] V. J. Mathews and G. L. Sicuranza, *Polynomial Signal Processing*, Wiley-Interscience, 2000.

[27] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A Free, Commercially Representative Embedded Benchmark Suite", *in IEEE 4th Annual Workshop on Workload Characterization*, Dec 2001.

[28] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vencentelli, F. Somenzi, A. Aziz, S-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, G. Shiple, S. Swamy, and T. Villa, "VIS: A System for Verification and Synthesis", *in Computer aided Verification*, 1996.

[29] M. Moskewicz, C. Madigan, L. Zhao, and S. Malik, "CHAFF: Engineering and Efficient SAT Solver", *in In Proc. Design Automation Conference*, pp. 530–535, June 2001.