Equivalence Verification of Arithmetic Datapaths with Multiple Word-Length Operands*

Namrata Shekhar, Priyank Kalla
Electrical & Computer Engineering Department
University of Utah, Salt Lake City, UT-84112
{*shekhar, kalla*}*@eng.utah.edu*

Florian Enescu
Department of Mathematics & Statistics
Georgia State University, Atlanta, GA-30303
fenescu@mathstat.gsu.edu

Abstract: *This paper addresses the problem of equivalence verification of RTL descriptions that implement arithmetic computations (add, mult, shift) over bit-vectors that have differing bit-widths. Such designs are found in many DSP applications where the widths of input and output bit-vectors are dictated by the desired precision. A bit-vector of size n can represent integer values from* 0 *to* $2^n - 1$*; i.e. integers reduced modulo* 2^n*. Therefore, to verify bit-vector arithmetic over multiple word-length operands, we model the RTL datapath as a polynomial function from* $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}}$ *to* Z_{2^m}*. Subsequently, RTL equivalence* $f \equiv g$ *is solved by proving whether* $(f - g) \equiv 0$ *over such mappings. Exploiting concepts from number theory and commutative algebra, a systematic, complete algorithmic procedure is derived for this purpose. Experimentally, we demonstrate how this approach can be applied within a practical CAD setting. Using our approach, we verify a set of arithmetic datapaths at RTL where contemporary approaches prove to be infeasible.*

I. Introduction

Many practical Digital Signal Processing (DSP) applications implement integer arithmetic operations, such as ADD, MULT, SHIFT, etc., over multiple bit-vector variables. Examples of such designs abound in DSP for audio, video and multimedia applications. High-level or register-transfer-level (RTL) descriptions of such systems can be modeled as multi-variate polynomials of finite degree, for design, synthesis [1] and verification purposes [2]. For efficient and correct modeling of such systems, it is important to account for the effect of *bit-vector size* of the operands on the resulting computation. For example, the largest (unsigned) integer value that a bit-vector of size m can represent is $2^m - 1$; implying that the bit-vector represents integer values reduced *modulo* 2^m ($\%2^m$). This suggests that bit-vector arithmetic can be efficiently modeled as algebra over finite integer rings, where the bit-vector size dictates the cardinality of the ring.

In many DSP applications, the computations are generally performed over operands that have multiple word-lengths; *i.e.*, input and output bit-vectors may have differing bit-widths. For instance, a digital audio-video mixer may perform polynomial arithmetic over a 20-bit audio and a 32-bit video signal [3]. To analyze these designs efficiently, it is therefore required to derive efficient computational procedures to model and manipulate multiple operand bit-vector arithmetic.

This paper addresses the problem of equivalence verification of arithmetic datapath computations over bit-vectors where the input and output operands may have different bit-widths. The problem is addressed at the level of behavioural/RTL descriptions. The following sub-section motivates the verification problem as it appears in the context of our work and describes our approach to the problem.

A. Motivating the Verification Problem

Let us motivate the equivalence verification problem as it appears in the context of our work. Initial high-level (say, MATLAB) specifications of digital signal processing applications are usually in floating-point. Such designs can be converted to fixed-point models [4] and subsequently translated into RTL. Automatic translation utilities are available for this purpose [5]. The verification problem instance is that of checking the equivalence of the fixed-point design against the translated (and optimized) RTL models.

Consider the computation performed by a digital image rejection/separation unit that takes as input two signals: a 12-bit vector $A[11:0]$ and another 8-bit vector $B[7:0]$. These signals are outputs of a mixer wherein one signal emphasizes on the image signal and the other emphasizes on the desired signal. The design produces a 16-bit output Y_1. The computation performed by the design is described in RTL as shown in Eqn. 1. Note that because of the specified bit-vector sizes, the computation can be *equivalently* implemented as another polynomial Y_2, as shown in Eqn. 2.

`input` $A[11:0], B[7:0]$; `output` $Y_1[15:0], Y_2[15:0]$;

$$\begin{aligned} Y_1 &= 16384 * (A^4 + B^4) + 64767 * (A^2 - B^2) \\ &\quad + A - B + 57344 * A * B * (A - B) \qquad (1) \\ Y_2 &= 24576 * A^2 * B + 15615 * A^2 + 8192 * A * B^2 + \\ &\quad 32768 * A * B + A + 17153 * B^2 + 65535 * B \qquad (2) \end{aligned}$$

Note that symbolically, the polynomials are distinct ($Y_1 \neq Y_2$), as they have different degrees and coefficients. However, because of the specified word-lengths of the input and output operands, $Y_1[15:0] \equiv Y_2[15:0]$. So, how do we verify the equivalence of such multiple word-length bit-vector computations? An algorithmic solution to this problem is the subject of this paper.

*Sponsored by NSF grants CCF-0514966 & CCF-515010.

B. Problem Modeling and Approach

We model the multiple word-length bit-vector computations as follows. Let $x_1, x_2, \ldots, x_d$ denote the d-variables (bit-vectors) in the design. Let $n_1, n_2, \ldots, n_d$ denote the size of the corresponding bit-vectors. Therefore, $x_1 \in Z_{2^{n_1}}, x_2 \in Z_{2^{n_2}}, \ldots, x_d \in Z_{2^{n_d}}$. Note that Z_{2^n} corresponds to the finite set of integers $\{0, 1, \ldots, 2^n - 1\}$. Let m correspond to the size of the output bit-vector f; hence, $f \in Z_{2^m}$. Subsequently, we model the arithmetic datapath computation as a multi-variate polynomial over $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}}$ to Z_{2^m} [6]. Here $Z_a \times Z_b$ represents the *Cartesian product* of Z_a and Z_b. The equivalence problem then corresponds to checking the congruence of two polynomials: $f \equiv g \% 2^m$.

If two such polynomials f, g are indeed computationally equivalent, then it means that they correspond to the same underlying **polynomial function** (or polyfunction). Unfortunately, checking for the equality of such polyfunctions is an NP-hard problem [7]. Our approach transforms the equivalence problem $f \equiv g \% 2^m$ as one of proving $(f - g) \% 2^m \equiv 0$; known as the *zero-equivalence problem* [7]. In other words, we test whether or not $(f - g) \% 2^m$ corresponds to a **nil polyfunction**. For the example shown above, we can compute $(Y_1 - Y_2) \% 2^{16}$:

$$Y_1 - Y_2 = 16384(A^4 + B^4) + 32768 * A * B * (A+1) + 49152 * (A^2 + B^2) \quad (3)$$

Note that $Y_1 - Y_2$ is a polynomial with non-zero coefficients. However, $\forall A \in Z_{2^{12}}, B \in Z_{2^8}$, $(Y_1 - Y_2) \% 2^{16} = 0$; *i.e.*, $(Y_1 - Y_2)$ *vanishes* as a function from $Z_{2^{12}} \times Z_{2^8}$ to $Z_{2^{16}}$.

Chen [6] has analyzed properties of such (nil) polyfunctions from a number-theoretic and commutative algebra perspective. We exploit some results from [6] and derive a systematic, algorithmic procedure to test for vanishing polyfunctions. Moreover, we demonstrate the applicability of our procedure within a practical CAD setting - by verifying a set of polynomial datapath computations over bit-vectors of disparate lengths for which contemporary techniques prove to be infeasible.

II. Previous Work

It is evident that contemporary graph-based canonical representations (BDDs [8], BMDs [9], K*BMDs [10] and their derivatives) are ill-suited for our application. This is mostly because these are based on variants of binary (bit-level) decomposition principles and, as such, they do not have the power of abstraction to model high-degree polynomial computations over wide bit-vectors. Recent work of Galois field decomposition of Boolean functions (MODD [11]) and other arithmetic transforms of Boolean functions [12] are also unable scale w.r.t. the design size corresponding to our applications. TEDs [2] have been proposed as canonical DAG representations for multi-variate polynomials. However, TEDs do not model modulo-arithmetic. While they can prove polynomial equivalence over the integral domain (Z), they cannot canonically model polyfunctions over finite integer rings.

Modulo arithmetic concepts have been studied in the context of RTL verification for bit-vector arithmetic [13] [14], word-level ATPG [15] and MILP-based simulation vector generation [16]. However, these are mostly geared toward *solving linear congruences* under modulo arithmetic - a different application from *proving equivalence of polynomial functions*. The recent work of [17] uses an arithmetic bit-level normalization technique to simplify subsequent SAT instances for bounded model checking of arithmetic circuits. Techniques such as theorem proving (HOL), term-rewriting [18] etc., have been used for datapath verification. However, they have generally been successful when datapath size can be abstracted away, say using data-dependence, symmetry and other abstractions [19] [20].

Contemporary Symbolic Computer Algebra tools 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. However, in the case of our application, the finite integer rings of residue classes Z_{2^m} (formed by m-bit vectors) correspond to non-UFDs, due to the presence of zero divisors (*e.g.*, $4 \neq 2 \neq 0, 4 \cdot 2 = 0$ in Z_8). In non-UFDs, any polynomial cannot be uniquely factorized into irreducible terms[1]. On the same lines, techniques using the concepts of Grobner's bases [21] [1] find extensive application in UFDs. However, for the above reasons, they cannot be directly ported to solve the above problem in non-UFDs of the type Z_{2^m}. We have analyzed a large number of symbolic algebra packages (NTL, ZEN, Maple, Mathematica, CoCoA, Singular, Macaulay, Pari, Macsyma, among others [22]). To the best of our knowledge, none of the available packages provide a "ready-made" procedure that can solve the desired polyfunction equivalence.

The works that come closest to ours are those of [23] and [24]. In [23], the zero-equivalence problem is solved for *univariate* datapaths (those with just one bit-vector variable). While [24] extends the results of [23] to multivariate bit-vector computations, the focus is still restricted to fixed-length datapaths. Both techniques lack the mathematical wherewithal to model polynomial computations over bit-vectors of unequal word-lengths.

III. Preliminaries

In what follows, Z corresponds to the set of integers, Z^+ to the set of non-negative integers and Z_n to the finite set of integers $\{0, 1, \ldots, n-1\}$. The ring of residue classes modulo 2^m is denoted by Z_{2^m}; where addition and multiplication are closed over $\{0, 1, \ldots, 2^m - 1\}$. $Z_{2^m}[x]$ denotes the ring of univariate polynomials over the variable x, with coefficients from Z_{2^m}. Similarly, $Z_{2^m}[x_1, \ldots, x_d]$ corresponds to the ring of multivariate polynomials in d-

[1]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 $(x-3)(x-4)$

variables, also denoted as $Z_{2^m}^d$.

In the context of our work, $n_1, n_2, \ldots, n_d$ corresponds to the bit-vector sizes of the input variables $x_1, x_2, \ldots, x_d$ and m represents the output bit-vector size. Subsequently, we represent the RTL computations as polyfunctions from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}}$ to Z_{2^m}. Chen [6] defines the corresponding polyfunction as follows:

Definition III.1: A function f from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots \times Z_{2^{n_d}}$ to Z_{2^m} is said to be a polynomial function (or *polyfunction*) 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 $x_i = 0, 1, \ldots, 2^{n_i} - 1$; $i = 1, 2, \ldots, d$.

Example III.1: Let $f : Z_{2^1} \times Z_{2^2} \rightarrow Z_{2^3}$ be a polyfunction in two variables (x_1, x_2), 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 $F = 1 + 2x_2 + x_1{x_2}^2$, since $f(x_1, x_2) \equiv F(x_1, x_2)\%2^3$ for $x_1 = 0, 1$ and $x_2 = 0, 1, 2, 3$.

It is possible for a polynomial with non-zero coefficients to *vanish* on such mappings; in which case the polynomial represents a *nil polyfunction* and their corresponding polynomials are often called *vanishing polynomials*.

The following Section describes the concepts that can be used to identify such polynomials. In the sequel, polynomial addition and multiplication are performed $\%n$ $(n = 2^m)$ according to the rules below:

$$(a+b)\%n = (a\%n + b\%n)\%n \quad (4)$$

$$(a \cdot b)\%n = (a\%n \cdot b\%n)\%n \quad (5)$$

$$(-a)\%n = (n - a\%n)\%n \quad (6)$$

Also, we use the following multi-index notation: $\mathbf{k} =< k_1, k_2, \ldots, k_d >$ are the (non-negative) degrees corresponding to the d input variables $\mathbf{x} =< x_1, x_2, \ldots, x_d >$, respectively.

IV. Theory

We begin with the analysis of univariate polynomials that vanish on $Z_{2^m}[x]$ (for didactic purposes) and then extend the results to vanishing polynomials from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}}$ to Z_{2^m}.

According to a fundamental result in number theory, 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 divides $k!$ (denoted $n|k!$). This value k corresponds to the *Smarandache function, SF(n)* [25]. In the ring Z_{2^m}, let $SF(2^m) = k$, such that $2^m|k!$. As an example, $SF(2^3) = 4$ as 8 divides 4! but 8 does not divide 3!; hence, least $k = 4$.

This property can be utilized to treat the equivalence problem as a divisibility issue in Z_{2^m}. When two polynomials $F(x)$ and $G(x)$ are equivalent in Z_{2^m} (*i.e.* $F(x)\%2^m \equiv G(x)\%2^m$), then it means that: $(F(x) - G(x))\%2^m \equiv 0 \Rightarrow 2^m|(F(x) - G(x))$. In $Z_{2^3}[x]$, let $8|(F(x) - G(x))$. But, 8|4! too. Therefore, if for all x, $(F - G)$ evaluated at x is a product of 4 consecutive numbers, then $(F - G)$ vanishes in Z_{2^3}. So, what is a natural example of such a polynomial? The answer is: $(x)(x-1)(x-2)(x-3)$. Such a product expression is referred to as a *falling factorial* and is formally defined below.

Definition IV.1: Falling factorials of degree $k \in Z$ are defined according to: $Y_0(x) = 1$, $Y_1(x) = x$, $Y_2(x) = x \cdot (x-1)$, ..., $Y_k(x) = x \cdot (x-1) \cdots (x-k+1)$

Example IV.1: Consider the following polynomial $F(x)$ over $Z_{2^8}[x]$:

$$\begin{aligned} F(x) &= x^{10} + 211x^9 + 102x^8 + 22x^7 + 41x^6 + 243x^5 \\ &\quad + 224x^4 + 36x^3 + 16x^2 + 128x \end{aligned}$$

In $Z_{2^8}[x]$, $SF(2^8) = 10$. Therefore, if for all x, $F(x)$ can be factored into a product of 10 consecutive numbers, (as in the case of $Y_{10}(x)$), then $F(x)$ is a vanishing polynomial in $Z_{2^8}[x]$. Indeed, $F(x) \equiv Y_{10}(x)\%2^8$; hence $F(x)\%2^8 \equiv 0$.

The above concept of falling factorials can be similarly defined for **multi-variate expressions** over $Z_{2^m}[x_1, \ldots, x_d]$:

$$\mathbf{Y_k} = \prod_{i=1}^{d} Y_{k_i}(x_i) = Y_{k_1}(x_1) \cdot Y_{k_2}(x_2) \cdots Y_{k_d}(x_d) \quad (7)$$

Extending the above concept, if a multivariate polynomial in $Z_{2^m}[x_1, \ldots, x_d]$ can be factorized into a product of $SF(2^m)$ consecutive numbers in **at least one of the variables** x_i, then it vanishes $\%2^m$. The following examples illustrates this idea.

Example IV.2: Consider the polynomial $F(x_1, x_2) = x_1^4 x_2 + 2x_1^3 x_2 + 3x_1^2 x_2 + 2x_1 x_2$ over $Z_{2^2}[x_1, x_2]$. Here, $SF(2^2) = 4$ and the highest degrees of x_1 and x_2 are $k_1 = 4$, and $k_2 = 1$, respectively. Note that $F\%4$ can be equivalently written as $F = Y_{<4,1>}(x_1, x_2)\%4 = Y_4(x_1) \cdot Y_1(x_2)\%4$. Since $F\%4$ can be represented as a product of 4 consecutive numbers in x_1, $2^2|F$ and $F \equiv 0$.

In the above example, both the input variables x_1, x_2, as well as the output F are in Z_{2^2}. We wish to extend the above concepts to analyze polynomials over $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots \times Z_{2^{n_d}}$ to Z_{2^m}. For this purpose, we define another quantity [6]:

$$\mu_i = min\{2^{n_i}, SF(2^m)\}; i = 1, 2, \ldots, d. \quad (8)$$

Now consider the following results [6]:

Lemma IV.1: Let $\mathbf{k} =< k_1, k_2, \ldots, k_d > \in (Z^+)^d$. Then, $\mathbf{Y_k} \equiv 0$ if and only if some $k_i \geq \mu_i$.

Example IV.3: Let $f : Z_{2^1} \times Z_{2^2} \rightarrow Z_{2^3}$ and its corresponding polynomial be $F = x_1^2 x_2 - x_1 x_2$. Here, $SF(2^3) = 4$, $k_1 = 2$ and $k_2 = 1$. Note that $\mu_1(2^1) = min\{2^1, 4\} = 2 = k_1$ (the condition in Lemma IV.1 is satisfied) and $\mu_2(2^2) = min\{2^2, 4\} = 4 > k_2$, and F can be written as:

$$\begin{aligned} x_1^2 x_2 - x_1 x_2 &\equiv x_1(x_1 - 1)x_2 \\ &\equiv Y_{<2,1>}(x_1, x_2) \\ &\equiv 0 \end{aligned}$$

When a polynomial cannot be factored into such Y_k expressions, can it still vanish? Consider the quadratic

polynomial $4x^2 - 4x$ in $Z_8[x]$. It can be written as $4(x)(x-1)$. While $4x^2 - 4x$ cannot be factorized as $(x)(x-1)(x-2)(x-3)$, it still vanishes in Z_8. The missing factors, $(x-2)(x-3)$ in this case, are compensated for by the *multiplicative constant* 4; therefore, $4x^2 - 4x \equiv 0\%8$. We now need to identify the constraints on such multiplicative constants such that the given polynomial would vanish. We state the following result [6]:

Lemma IV.2: The expression $c_{\mathbf{k}} \cdot \mathbf{Y_k} \equiv 0$ if and only if $\frac{2^m}{gcd(2^m, \prod_{i=1}^{d} k_i!)} | c_{\mathbf{k}}$; where:

- $c_{\mathbf{k}} \in Z$;
- $\mathbf{k} =< k_1, k_2, \ldots, k_d > \in Z^d$ such that $k_i < \mu_i, \forall i = 1, \ldots, d$; and
- $gcd(2^m, \prod_{i=1}^{d} k_i!)$ is the greatest common divisor of 2^m and $\prod_{i=1}^{d} k_i!$.

Example IV.4: Consider the polynomial $F = 4x_1x_2^2 + 4x_1x_2$ corresponding to $f(x_1, x_2) : Z_{2^1} \times Z_{2^2} \rightarrow Z_{2^3}$. We can use Lemma IV.2 to prove that f is a nil polyfunction. Here, $2^{n_1} = 2$, $2^{n_2} = 4$ and $2^m = 8$. $\mathbf{k} =< k_1, k_2 >=< 1, 2 >$ corresponds to the highest degrees of x_1, x_2. Moreover, $\prod_{i=1}^{2} k_i! = 1! \cdot 2! = 2$. Also, $SF(2^m = 8) = 4$; $\mu_1(2) = min\{2, 4\} = 2$, $\mu_2(4) = min\{4, 4\} = 4$.

$$\begin{aligned} F &\equiv 4x_1x_2^2 + 4x_1x_2 \\ &\equiv 4 \cdot x_1 \cdot x_2 \cdot (x_2 - 1) \\ &\equiv c_{<1,2>} \cdot Y_{<1,2>}(x_1, x_2) \\ &\equiv 0 \end{aligned}$$

because $\frac{8}{(8,1!\cdot 2!)} = 4$ which divides $c_{<1,2>} = 4$.

The above results can be extended to derive necessary and sufficient conditions for a polynomial to vanish as a function from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots Z_{2^{n_d}}$ to Z_{2^m}. We state the following theorem [6]:

Theorem 1: Let F be a polynomial representation for the function f from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots Z_{2^{n_d}}$ to Z_{2^m}. Then, F is a vanishing polynomial ($F \equiv 0$) if and only if it can be represented as:

$$F = Q_\mu \mathbf{Y}_\mu + \Sigma_{\mathbf{k}} a_{\mathbf{k}} b_{\mathbf{k}} \mathbf{Y_k} \tag{9}$$

where:

- $Q_\mu \in Z[x_1, \ldots, x_d]$ is an arbitrary polynomial;
- $\mathbf{Y_k}$ is the falling factorial defined in Eqn. 7;
- $\mathbf{Y}_\mu = \mathbf{Y_k}$ for some $k_i \geq \mu_i$;
- $\mathbf{k} =< k_1, \ldots, k_d >$ for each $k_i = 0, 1, \ldots, \mu_i - 1$;
- $a_{\mathbf{k}} \in Z$ is an arbitrary integer and
- $b_{\mathbf{k}} = \frac{2^m}{(2^m, \prod_{i=1}^{d} k_i!)}$.

Proof: The proof follows straight-forwardly from Lemma IV.1 (for the computation $Q_\mu \mathbf{Y}_\mu$) and from Lemma IV.2 (for the computation $\Sigma_{\mathbf{k}} a_{\mathbf{k}} b_{\mathbf{k}} \mathbf{Y_k}$). ■

The following example illustrates the above concept.

Example IV.5: Consider a polynomial $F = x_1^2 + 7x_1 + 4x_1x_2^2 + 4x_1x_2$ for $f : Z_2 \times Z_{2^2} \rightarrow Z_{2^3}$. Here, $\mu_1(2) = min\{2, SF(8)\} = 2$; $\mu_2(2^2) = min\{2^2, SF(8)\} = 4$. F can be written as follows:

$$\begin{aligned} x_1^2 + 7x_1 + 4x_1x_2^2 + 4x_1x_2 &\equiv Y_{<2,0>}(x_1, x_2) + \\ &\quad a_{<1,2>} b_{<1,2>} Y_{<1,2>}(x_1, x_2) \\ &\equiv 0 \end{aligned}$$

Here, $a_{<1,2>} = 1$ and $b_{<1,2>} = 8/(8, 1! \cdot 2!) = 4$. F can be written in the form given by Theorem 1, and is thus a vanishing polynomial.

V. Algorithm: Zero Equivalence

Using the concepts presented in the previous section, we have derived a systematic algorithmic procedure that tests whether a given polynomial vanishes as a function from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots Z_{2^{n_d}}$ to Z_{2^m}. Algorithm 1 depicts the procedure.

```
ZERO_EQ(F1, F2, d, x, m, n)
F1, F2 = Polynomials in x;
d = Number of variables;
x[1...d] = List of input variables;
m = Bit-width of F1 and F2;
n[1...d] = List of bit-widths of input variables, x;

poly = F1 - F2;
Compute SF(2^m);

/*Compute values for μ_i*/
for i = 1 to d do
  μ[i] = min{SF(2^m), 2^n[i]};
  k[i] = Max. degree of x[i] in poly;
end for

/*Check if Y_μ divides poly*/
for i = 1 to d do
  /*Lemma IV.1*/
  if (k[i] ≥ μ[i]) then
    quo, rem = poly / Y_<0,...,k[i],...,0>(x1,...,xd);
    if (rem == 0) /* rem = remainder */ then
      return true; /*poly = Q_μ Y_μ; a vanishing polynomial*/
    else
      poly = rem;
      break;
    end if
  end if
end for

/*Iterate over all possible degrees*/
for j = ∏_{l=1}^{d} (μ_l) to 1 do
  /*Update degrees*/
  for i = 1 to d do
    k[i] = Update degree of x[i] in current poly;
  end for
  quo, rem = poly / Y_<k[0],...,k[d]>(x1,...,xd);
  b_<k[0],...,k[d]> = 2^m / (2^m, ∏_{i=1}^{d} k[i]!);
  /*Lemma IV.2*/
  if (b_<k[0],...,k[d]> | quo) then
    if (rem == 0) then
      return true;
    else
      poly = rem;
    end if
  else
    return false;
  end if
end for
```

Algorithm 1: ALGORITHM ZERO_EQ: Zero testing a given polynomial.

The algorithm takes as input the two polynomials F_1 and F_2 in variables $x_1, \ldots, x_d$ with corresponding input bit-widths $n_1, \ldots, n_d$, and output bit-width m. The output is *true* if $F_1 \equiv F_2$. The algorithm operates as follows:

1. Find the difference of the two polynomials, *poly*. This is the expression which should vanish to prove equiva-

lence.
2. Compute the Smarandache function value for 2^m; an *O(n/log n)* computational procedure, given in [26], has been implemented. Subsequently, $SF(2^m)$ value is used to obtain the μ_i values.
3. Note the maximum degree (k_i) of each variable x_i in *poly*.
4. Divide the polynomial by the falling factorial expressions Y_μ in each of the d variables.
5. If the remainder is zero, $F_1 \equiv F_2$; because $F = Q_\mu \mathbf{Y}_\mu$.
6. Else, use the remainder as the new *poly*. Update the degrees (k_i) and continue dividing from $Y_{\mu-1}$ to Y_0 for each variable.
7. After each division, check for the following conditions:
- If the quotient can be written as $a_{\mathbf{k}} \cdot b_{\mathbf{k}}$ (where $b_{\mathbf{k}}$ is defined according to Theorem 1), and the remainder is zero, return *true*. It is a vanishing polynomial.
- If the quotient can be written as $a_{\mathbf{k}} \cdot b_{\mathbf{k}}$, and the remainder is non-zero, continue to the next iteration.
- If the quotient cannot be written as $a_{\mathbf{k}} \cdot b_{\mathbf{k}}$, return *false*. $poly \neq 0 \Rightarrow F_1 \neq F_2$.

Complexity: In Algorithm 1, the number of multivariate divisions is bound by $O(\prod_d \mu_i)$, where μ_i is as defined previously and d is the total number of variables.

VI. Experimental Results

Algorithm 1 was implemented in Perl with calls to Maple 7 [27] for all the algebraic manipulations. Using our algorithm, we have been able to perform verification runs over a number of designs collected from a variety of benchmark suites.The results are presented in Table I.

The first example is the from Sec. I, and represents the image rejection computation. The next two examples [1] are phase-shift keying and anti-aliasing functions, both used in digital communication. The polynomial filters [3] are Volterra models of polynomial signal processing applications. Horner polynomials [28] are commonly used in DSP - often implemented using multiply-add-accumulate units. In [1], it was shown how computations by these MAC units can be extracted as polynomials in Horner's form. MIBench is an automotive application from [29]. The last example is a vanishing polynomial of degree 10, specifically created to validate our algorithm. The second column in the table describes the characteristcs of these benchmarks: number of variables (var), highest degree of the polynomial (Deg), and input/output word-lengths (n_i, m).

Our experimental setup is as follows: High-level restructuring and symbolic algebra-based transformations - such as: modulating and segmenting the coefficients, factorization and expansion, addition and removal of algebraic redundancy (vanishing polynomials), etc. - were applied to the original RTL descriptions to obtain symbolically different but functionally equivalent implementations. Subsequently, the data-flow graphs for the given RTL descriptions were extracted using Gaut [30]. Traversing the DFGs from the inputs to the outputs, the polynomial representations were constructed. The datapath sizes of both inputs and outputs ($n_1, \ldots, n_d$ and m) were also recorded. The algorithm was invoked to find the difference between the two polynomials and subsequently verify that it computes zero, to prove equivalency. We were able to solve all problems in < 25 seconds.

We have performed equivalence checking of the given RTL designs using BDDs, BMDs and SAT 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 resulting netlists using the VIS [31] 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 [32] was used to prove equivalence via unsatisfiability testing. For all the designs, ZChaff could not solve the problem within the time-limit of 1000s. We also 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 (degree ≤ 4). The other benchmarks could not be verified within the time-out limit.

A. Limitations of our approach

Many DSP systems implement some form of computation approximation, by incorporating various rounding schemes. Our approach is currently restricted inasmuch as it cannot verify those datapaths where intermediate signals have varying precision (due to rounding). Similarly, saturation arithmetic architectures can also not be verified using our technique. Analysis of such designs requires substantially more work, and is the subject of our future investigations.

VII. Conclusions

We have presented a framework for equivalence verification of arithmetic datapaths with multiple word-length operands. Our approach models the design as a polyfunction from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$. The concept of vanishing (nil) polyfunctions is exploited to prove equivalence between two symbolically distinct (but computationally equivalent) polynomials. The concepts from number theory and commutative algebra have been applied to derive a complete algorithmic procedure for this purpose. Using our algorithm, a variety of benchmarks have been verified. Our approach was able to solve the problem in all cases, where contemporary verification approaches were infeasible. As part of future work, we are investigating applications of the proposed concepts to datapath computations that implement rounding.

Acknowledgements

The authors wish to acknowledge the help of Sivaram Gopalakrishnan of the University of Utah for conducting equivalence checking experiments with BDDs and the VIS package.

TABLE I

COMPARISON OF TIME TAKEN BY VARIOUS APPROACHES

Benchmark	**Specs**	**Our approach**	**BDDs-VIS**	**BMD**	**SAT-ZChaff**
	Var/Deg/$< n_1, \ldots, n_d >/m$	**Time (s)**	**Nodes/Time(s)**	**Nodes/Time(s)**	**Vars/Clauses/Time(s)**
IRR	$2/4/< 12, 8 >/16$	14.4	NA/1000	NA/>1000	10K/30K/>1000
PSK	$2/4/< 11, 14 >/16$	12.9	NA/>1000	NA/>1000	61K/183K/>1000
Anti-alias function	$1/6/< 11 >/16$	4.69	53M/38.3	NA/>1000	47K/142K/>1000
Cubic filter	$3/3/< 24, 28, 31 >/32$	9.47	NA/>1000	NA/>1000	120K/366K/>1000
Degree-4 filter 1	$3/4/< 15, 11, 13 >/16$	16.4	NA/>1000	NA/>1000	69K/205K/>1000
Degree-4 filter 2	$1/4/< 12 >/16$	4.95	30M/21.3	NA/>1000	25K/76K/>1000
Savitzky-Golay filter	$5/3/< 16, 16, 14, 12, 8 >/16$	14.25	NA/>1000	NA/>1000	64K/190K/>1000
4^{th} Order IIR	$2/4/< 24, 29 >/32$	11.12	NA/>1000	NA/>1000	214K/647K/>1000
MIBENCH	$2/9/< 16, 12 >/16$	22.9	26M/16.7	NA/>1000	24K/69K/>1000
Horner Polynomial 1	$3/4/< 14, 14, 16 >/16$	4.46	NA/>1000	2355 / 85	14K/44K/>1000
Horner Polynomial 2	$3/4/< 10, 8, 16 >/16$	4.71	NA/>1000	1574 / 47	12K/37K/>1000
Horner Polynomial 3	$2/4/< 13, 13, 16 >/16$	4.97	NA/>1000	6803 / 246	23K/70K/>1000
Vanishing polynomial	$2/10/< 12, 12 >/16$	2.19	NA/>1000	NA/>1000	10K/29K/>1000

REFERENCES

[1] 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.

[2] 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.

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

[4] D. Menard, D Chillet, F Charot, and O. Sentieys, "Automatic Floating-point to Fixed-point Conversion for DSP Code Generation", *in Intl. Conf. Compiler, Architecture, Synthesis Embedded Sys., CASES*, 2002.

[5] I. A. Groute and K. Keane, "M(VH)DL: A MATLAB to VHDL Conversion Toolbox for Digital Control", *in IFAC Symp. on Computer-Aided Control System Design*, Sept. 2000.

[6] 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.

[7] 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.

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

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

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

[11] A. Jabir and Pradhan D., "MODD: A New Decision Diagram and Representation for Multiple Output Binary Functions", *in Design, Automation and Test in Europe, DATE*, 2004.

[12] K. Radecka and Z. Zilic, "Specifying and verifying imprecise sequential datapaths by arithmetic transforms", *in Intl. Conf. Computer-Aided Design*, 2002.

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

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

[15] 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.

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

[17] M. Wedler, D. Stoffel, and W. Kunz, "Normalization at the Arithmetic Bit-Level", *in Design Automation Conf.*, 2005.

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

[19] V. Paruthi, N. Mansouri, and R. Vemuri, "Automatic datapath abstraction for verification of large scale designs", *in Intl. Conf. on Computer Design*, 1998.

[20] T. Raudvere, A. K. Singh, I. Sander, and Jantsch. A, "System level verification of digital signal processing applications based on the polynomial abstraction technique", *in Intl. Conf. on Computer Aided Design*, Nov 2005.

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

[22] J. Grabmeier, E. Kaltofen, and V. Weispfenning, *Computer Algebra Handbook*, Springer-Verlag, 2003.

[23] N. Shekhar, P. Kalla, F. Enescu, and S. Gopalakrishnan, "Exploiting Vanishing Polynomials for Equivalence Verification of Fixed-Size Arithmetic Datapaths", *in Intl. Conf. Computer Design*, 2005.

[24] N. Shekhar, P. Kalla, F. Enescu, and S. Gopalakrishnan, "Equivalence Verification of Polynomial Datapaths with Fixed-Size Bit-Vectors using Finite Ring Algebra", *in Intl. Conf. on Computer-Aided Design, ICCAD*, 2005.

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

[26] D. Power, S. Tabirca, and T. Tabirca, "Java Concurrent Program for the Smarandache Function", *Smarandache Notions Journal*, vol. 13, pp. 72–84, 2002.

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.