EXPLOITING VANISHING POLYNOMIALS FOR EQUIVALENCE VERIFICATION OF FIXED-SIZE ARITHMETIC DATAPATHS

Namrata Shekhar[1], Priyank Kalla[1], Florian Enescu[2] & Sivaram Gopalakrishnan[1]
[1]Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT-84112
[2]Department of Mathematics and Statistics
Georgia State University, Atlanta, GA

Abstract

This paper addresses the problem of equivalence verification of high-level/RTL descriptions. The focus is on datapath-oriented designs that implement *univariate polynomial computations over fixed-size bit-vectors*. Such designs, found in many DSP applications, perform a sequence of ADD, MULT, SHIFT type of algebraic computations that can be modeled as univariate polynomials of finite degree. Often, the datapath size (m) of the entire design is kept constant - fixed according to the size of the bit-vector operands. Such fixed-size bit-vector arithmetic manifests itself as polynomial algebra over finite integer rings of residue classes Z_{2^m}. RTL verification problem then reduces to that of checking polynomial equivalence in Z_{2^m}: in other words, to prove $f(x)\%2^m \equiv g(x)\%2^m$.

This paper formulates the equivalence verification problem $f(x)\%2^m \equiv g(x)\%2^m$ by transforming it into proving $(f(x) - g(x))\%2^m \equiv 0$. Exploiting the theory of *vanishing polynomials* over finite integer rings, a systematic algorithmic procedure is derived to establish whether or not a given polynomial vanishes (always evaluates to 0) over Z_{2^m}. The algorithm has been implemented in MAPLE. Using our approach, symbolically distinct but computationally equivalent RTL descriptions of various multi-media and elementary function computations have been correctly verified; where BDD, BMD, SAT and MILP-based methods are found to be impractical.

I. INTRODUCTION

RTL descriptions of integer datapaths that implement polynomial arithmetic are found in many practical designs: such as in digital signal processing (DSP) for audio, video and multimedia applications, evaluation of elementary and transcendental functions, etc [1]. Such designs perform a sequence of ADD, MULT, SHIFT type of algebraic computations that can be modeled as *univariate polynomials of finite degree*. Such computations are often implemented with fixed-sized datapath architectures where the design choice is that of a single, uniform system word-length for the computations [2]. This allows to optimize the area, delay and power related costs of the implementations while operating within the desired signal-to-noise-ratio margins.

In many DSP applications, data is represented as a bit-vector (integer) where its bit-width (m) is determined by the desired precision. Subsequently, polynomial computations are performed over m-bit integers where the size of the entire datapath is kept constant 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. Usually, such computations require appropriate scaling of coefficients and/or signals such that overflow can be avoided/ignored and standard integer/two's complement arithmetic can be implemented.

When the datapath size (m) over the entire design is kept constant, fixed-size bit-vector arithmetic manifests itself as *polynomial algebra over finite integer rings* of residue classes Z_{2^m}; *i.e.* integer addition and multiplication is closed within the finite set of integers $\{0, \ldots, 2^m - 1\}$. This requires the modeling of modulo arithmetic operations on polynomials [3]. In such cases, symbolically distinct polynomials (those with different degrees and coefficients) can become computationally equivalent.

This paper addresses the problem of *RTL equivalence verification* of datapath descriptions that implement *univariate polynomial computations over fixed-size bit-vectors*. The specific verification problem corresponds to that of proving the *computational equivalence* of univariate polynomials in Z_{2^m}; in other words to prove: $f(x)\%2^m \equiv g(x)\%2^m$, where f, g correspond to polynomial computations over the bit-vector (variable) x and $m =$ datapath size. Our approach transforms the problem $f(x)\%2^m \equiv g(x)\%2^m$ into that of testing whether $(f(x) - g(x))\%2^m \equiv 0$; in other words to check whether $(f(x) - g(x))$ *vanishes* in Z_{2^m}. The theory of vanishing polynomials [4] [5] over finite rings is exploited to derive an algorithmic procedure to check for the same. The paper depicts how this theory can be applied to a CAD-based verification framework and demonstrates that our approach is fast, robust and scalable. It provides an efficient datapath verification solution where contemporary BDD, BMD, SAT and MILP based methods are found to be impractical.

II. MOTIVATION: VERIFICATION PROBLEM & APPLICATIONS

Let us motivate the equivalence verification problems as they appear in the context of our work. We begin with a simple example (for didactic purposes) that demonstrates the concept of a vanishing polynomial. Consider the RTL computations t_1 and t_2 shown in Fig. 1 below. The outputs t_1

and t_2 are symbolically distinct polynomials. However, because the datapath size is fixed to 4-bits, the computations $t_1\%2^4 \equiv t_2\%2^4$ are equivalent. Interpreting the computations as polynomials over Z (and not over Z_{2^4}) would erroneously prove non-equivalence. Computing their difference ($\%16$) results in $t_1[3:0] - t_2[3:0] = 8 * (X[3:0])^2 + 8 * X[3:0]$. While $8x^2 + 8x$ is symbolically a non-zero polynomial, operating in Z_{2^4}, $(8x^2 + 8x)\%16 \equiv 0, \forall x \in \{0, \ldots 15\}$. In other words, $8x^2 + 8x$ *vanishes* in Z_{2^4}. Therefore, by identifying whether a polynomial vanishes over a finite ring, one can prove or disprove the equivalence of fixed-size datapath computations.

```
module fixed_bit_width (X, t1, t2);
input [3:0] X; output [3:0] t1, t2;
/* t1 and t2 are equivalent in Z_2^4 but not in Z*/
assign t1[3:0] = 4 * (X[3:0])^2 + 12 * X[3:0] - 6;
assign t2[3:0] = 12 * (X[3:0])^2 + 4 * X[3:0] + 10;
```

Fig. 1. Description of fi xed size datapath: equivalent computations but different polynomial representations.

Now let us demonstrate some practical scenarios that motivated this work. Consider the anti-alias function of an MP3 decoder that computes [6]:

$$F = \frac{1}{2\sqrt{a^2 + b^2}}, \tag{1}$$

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 computation can then 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:

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

Now suppose that application of high-level/algebraic manipulation techniques (such as [6] [7] [8] [9] [10]) to the above may transform the RTL implementation to:

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

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 $F\%2^{16} \equiv G\%2^{16}$.

As another interesting example (from [10]), consider the 4^{th}-degree polynomial computation $F_1[15:0]$ expressed in Horner's form below. Equivalently, it can also be computed as a *cubic* polynomial $F_2[15:0]$, also shown below. F_1 and F_2 have both different degrees and different coefficients; yet $F_1\%2^{16} \equiv F_2\%2^{16}$.

$$\begin{aligned} F_1[15:0] \;=\; & 1277 + (16384 + (63371 + \\ & (20994 + 40960X[15:0]) \\ & X[15:0])X[15:0])X[15:0] \\ F_2[15:0] \;=\; & 4610(X[15:0])^3 + 6027(X[15:0])^2 + 1277 \end{aligned}$$

Another important application of this work is the verification of automatic translation of MATLAB/Simulink [11] computational models to VHDL descriptions (RTL). For example, many DSP computations are first simulated in MATLAB using the Simulink/Filter-design toolbox; often, using fixed-precision arithmetic. Subsequently, the obtained data-flow computations are then automatically translated into VHDL (for synthesis) using various translator toolboxes, such as [12]. The original behavioural computations can then be verified against the translated RTL.

This paper describes an algorithmic solution to such equivalence verification problems. Note that equivalence of multivariate polynomial datapaths, or multiple word-size datapaths, or fixed-size Boolean bit-vector computations is not the subject of this paper. The proposed techniques can perhaps be extended to address the above problems; however, that requires further research and is not the immediate focus of this paper.

III. Limitations of Contemporary Verification Approaches

It is evident that Binary Decision Diagrams (BDDs) [13], and their derivatives [14] [15], are ill-suited for our application; mostly due to the presence of a large number of multiply operations. On the other hand, BMDs [16] have been successful for multiplier verification [16] [17]. To represent $X_m * Y_m$, (m = bit-vector size) the size complexity of the resulting BMD is linear in the total number of bits. However for polynomial terms of the form X_m^k, their size complexity is $O(m^k)$. K*BMDs [18] reduce this complexity to $O(m^{k-1})$, but the time complexity to reduce/canonize the diagrams is still exponential. As a result, verification of high-degree polynomials using BMDs/K*BMDs may require long compute-times. TEDs [19] have been proposed as canonical DAG representations of polynomials. However, TEDs do not model modulo-arithmetic. Therefore, while they can verify symbolic equivalence over Z, they cannot prove computational equivalence of polynomials over finite rings.

Modulo arithmetic concepts have been studied in the context of RTL verification for bit-vector arithmetic [20] [21], word-level ATPG [22] and MILP-based RTL verification [23]. However, these are mostly geared towards *solving linear congruences* under modulo arithmetic - a different application from *proving polynomial equivalence* modulo 2^m. Pradhan's recent work [24] can be applied to our problem; however, operating in $GF(2^m)(\neq Z_{2^m})$ their technique is also limited by its computational complexity.

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 [25] [17] [26] [27] [28] [29]. 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 [30] [31] [32] [33] 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 [34]. 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} cannot be uniquely factorized into irreducible terms[1]. On the same lines, techniques using the concepts of Grobner's bases [35] [36] [6] 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}. Similarly, the extensive work of Kaltofen *et. al.* [37] on verifying equivalence of polynomial computations represented by straight-line programs is also geared towards fields R and C, but not applicable in Z_{2^m}. The symbolic algebra library ZEN [38] allows 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 $f(x)\%2^m \equiv g(x)\%2^m$ is not available.

A. Related Work in Number Theory & Polynomial Algebra

The problem $f(x)\%n \equiv g(x)\%n$ (or equivalently, $(f - g)\%n \equiv 0$) is known to be NP-hard when $n \geq 2$ [39]. Researchers from the field of number theory and commutative algebra have analyzed the properties of vanishing polynomials. A vanishing polynomial over a finite ring is a member of the *Ideal* of that ring [3]. Therefore, transforming the problem $f\%n \equiv g\%n$ into $(f - g)\%n \equiv 0$ has its appeal because it then falls into the domain of *ideal membership testing problems* [40]. For example, over prime order rings $Z_p[x]$, vanishing polynomials necessarily and sufficiently have to be a multiple of $x^p - x$ (from Fermat's theorem [3]). However, in our case when $n = 2^m$ (non-UFD), the problem is not so straight-forward.

Singmaster [4] analyzed in detail the theory of *univariate vanishing polynomials* over $Z_n, n \in N, n > 1$; *i.e.* those polynomials f such that $f(x)\%n \equiv 0$. He identified necessary and sufficient conditions for a univariate polynomial to vanish $\%n$. Our fixed vector size RTL datapath verification problem can therefore be solved using Singmaster's results; as in our case $n = 2^m$.

[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 $(x-3)(x-4)$.

B. Our Approach & Contributions

- We formulate the fixed-vector-size (m) arithmetic datapath verification problem as the identification of vanishing polynomials in Z_{2^m}.
- From the concepts presented in [4], we *derive a systematic algorithmic procedure* that operates on the given polynomials in Z_{2^m} and tests whether they vanish. The algorithm is implemented within Maple [30].
- We extract the data-flow graphs (DFG) [41] corresponding to the given RTL descriptions and construct their polynomial representations by traversing the DFGs from inputs to outputs. The difference of these polynomials is computed and the vanishing test is performed to prove or disprove equivalence.
- Experimentally, we demonstrate that the proposed approach is able to verify the equivalence of high-degree polynomial RTL datapaths (real-world benchmarks), where none of the BDD [13], BMD [16], SAT [42] and MILP [23] based techniques provide an efficient solution.

IV. Preliminaries

Definition IV.1: An **Abelian group** is a set G and a binary operation " $+$ " satisfying: (i) *Closure:* For every $a, b \in G, a + b \in G$. (ii) *Associativity:* For every $a, b, c \in G, a + (b + c) = (a + b) + c$. (iii) *Commutativity:* For every $a, b \in G, a + b = b + a$. (iv) *Identity:* There is an identity element $0 \in G$ such that for all $a \in G; a + 0 = a$. (v) *Inverse:* If $a \in G$, then there is an element $a^{-1} \in G$ such that $a + a^{-1} = 0$.

Definition IV.2: A **Commutative ring with unity** is a set R and two binary operations " $+$ " and " $\cdot$ ", as well as two distinguished elements $0, 1 \in$ R such that: (i) R is an Abelian group with respect to addition with additive identity element 0. (ii) *Multiplication Closure:* For every $a, b \in$ R, $a \cdot b \in$ R. (iii) *Multiplication Associativity:* For every $a, b, c \in$ R, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$. (iv) *Multiplication Commutativity:* For every $a, b \in$ R, $a \cdot b = b \cdot a$. (v) *Multiplication Identity:* There is an identity element $1 \in$ R such that for all $a \in$ R, $a \cdot 1 = a$. (vi) *Distributivity:* For every $a, b, c \in$ R, $a \cdot (b + c) = a \cdot b + a \cdot c$ holds for all $a, b, c \in$ R.

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 *modulo n* ($\%n$). For our application, $n = 2^m$.

Definition IV.3: Integers x, y are called **congruent modulo** n ($x \equiv y\ \%n$) if n is a divisor of their difference: $n|(x - y)$.

Definition IV.4: A **field** F is a commutative ring with unity, where every element in F except 0 has a multiplicative inverse, i.e. $\forall$ a $\in$ F $-$ 0, $\exists$â $\in$ F such that a$\cdot$â $= 1$.

The system Z_n forms a field if and only if n is prime. Hence, Z_{2^m} is not a field as not every element in Z_{2^m} has an inverse. Lack of inverses in Z_{2^m} makes RTL verification complicated, as it disallows the use of Euclidean algorithms for division and factorization.

Definition IV.5: Let R be a ring. A **polynomial** over R in the indeterminate x is an expression of the form $a_0+a_1x+a_2x^2+\cdots+a_kx^k=\sum a_ix^i, \forall a_i \in$ R. Elements a_i are coefficients, k is the degree. The element a_n is called the **leading coefficient**; when $a_n = 1$, the polynomial is monic.

The system consisting of the set of all polynomials in x over the ring R, with addition and multiplication defined accordingly, also forms a ring, called the **ring of polynomials** R[x].

In the sequel, we will assume that polynomial addition and multiplication is always performed $\%n(n = 2^m)$, and the coefficients are reduced according to standard modulo operations:

$$(a+b)\%n = (a\%n + b\%n)\%n \quad (2)$$

$$(a \cdot b)\%n = (a\%n \cdot b\%n)\%n \quad (3)$$

$$(-a)\%n = (n-a)\%n \quad (4)$$

With this basic background, we now present the results of [4] in the context of our work for identifying vanishing polynomials over Z_{2^m}.

V. On Univariate Vanishing Polynomials

Singmaster[4] showed that equivalence of univariate polynomials $f(x) \equiv g(x)$ over any finite integer ring Z_n ($n > 1$) can be solved as $(f(x) - g(x))\%n \equiv 0$. He derived necessary and sufficient conditions for a polynomial to vanish ($\equiv 0$) modulo n. We highlight some concepts from [4] as they relate to this paper. Note that the results from [4] are stated for rings specific to our application $n = 2^m$. Moreover, the results are stated without proof, as their proofs can be found in [4]. However, we do explain the significance of these results and demonstrate their application by means of examples.

It is a well-known result [5] 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)* [43], *i.e.* $k = SF(n)$. In the ring of interest, Z_{2^m}, let $SF(2^m)$ be equal to the least integer k, such that $2^m|k!$. As an example, $SF(2^3) = 4$ as 8 divides $4! = 4\times3\times2\times1 = 2^3\times3$. Note that 8 does not divide 3!, and hence the **least** k in question $= 4$.

The significance of the above concept can be explained as follows. Consider the ring Z_8. The least value k such that $8|k!$ is $k = 4$. Therefore, any integer that can be factored into a product of (at least) $k = 4$ consecutive numbers will vanish in Z_8. 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 $(f-g)$, evaluated at x, can be represented as the product of 4 consecutive numbers (depending on x), then $(f-g)$ would vanish in Z_{2^3}. So, what is a natural example of a polynomial with this property? The answer is: $(x+1)(x+2)(x+3)(x+4)$.

In this regard, Singmaster [4] 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. More formally, the following definition:

Definition V.1:

$$\begin{aligned} S_0(x) &= 1 \\ S_1(x) &= x+1 \\ S_2(x) &= (x+2)(x+1) \\ S_3(x) &= (x+3)(x+2)(x+1) \\ &\vdots = \vdots \\ S_k(x) &= (x+k)\cdot S_{k-1}(x). \end{aligned}$$

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 V.1: Consider the polynomial p in Z_{2^8}, given by: $p = x^{10} + 55x^9 + 40x^8 + 230x^7 + 77x^6 + 167x^5 + 98x^4 + 156x^3 + 168x^2 + 32x$

Also, $SF(2^8) = 10$. 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}, $p = S_{10}(x) = \prod_{k=1}^{10}(x+k)$, and hence $p\%2^8 \equiv 0$.

When a polynomial cannot be factored into such S_k expressions, can it still vanish? For example, the quadratic vanishing polynomial $4x^2+4x$ in Z_8, written as $4(x+2)(x+1)$, cannot be factorized as $S_4(x) = \prod_{k=1}^{4}(x+k)$. The missing factors, $(x+4)(x+3)$ in this case, are compensated for by the multiplicative constant 4; therefore, $4x^2 + 4x \equiv 0$. Singmaster identified the constraint on such multiplicative constants such that the polynomial in question would vanish. We state the following result.

Lemma 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, $b \in Z_{2^m}$, k is the degree of the expression $b \cdot S_k(x)$ and $S_k(x)$ is as defined in Definition V.1.

Example V.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 if b were replaced by 3, then $p(x) = 3(x+2)(x+1)$ would not be a vanishing polynomial as $\frac{2^3}{(2!,2^3)}$ would not divide 3.

The above concepts lead to Singmaster's theorem (Theorem 6 [4]) that identifies the necessary and sufficient conditions for a polynomial to vanish over any finite integer ring. We re-state the result for Z_{2^m}.

Theorem 1: Let F be a polynomial in $Z_{2^m}[x]$. Then F vanishes $\%2^m$ if and only if:

$$F \equiv F_n \cdot S_n + \sum_{k=0}^{n-1} a_k \cdot b_k \cdot S_k, \quad (5)$$

where:

- $n = SF(2^m)$, *i.e.* the least n such that $2^m | n!$;
- F_n is an arbitrary polynomial in $Z_{2^m}[x]$;
- a_k is an arbitrary integer;
- $b_k = \frac{2^m}{(k!, 2^m)}$.

Let us explain the above result by means of a few examples.

Example V.3: Let $F = (x+4)(x+3)(x+2)(x+1)$ in $Z_{2^3}[x]$. Here $n = SF(2^3) = 4$. Therefore, $F(x)$ can be represented as $F(x) = F_4 \cdot S_4 + 0$, where $F_4 = 1$ and $S_4 = (x+4)(x+3)(x+2)(x+1)$. Hence, F vanishes in Z_{2^m}.

Example V.4: Now consider $F = 4x^2 + 4x = 4(x+2)(x+1)$ in $Z_{2^3}[x]$. Again, $n = SF(2^3) = 4$. However, F cannot be factored into S_4, therefore $F_4 = 0$. Now consider $k = n-1 = 4-1 = 3$. Since $4x^2 + 4x$ (quadratic) cannot be factored by $S_3(x)$ (cubic), $a_3 = 0$. However, F can be factored according to S_2, leading to $a_2 = 1$ and $b_2 = 4$. Therefore, F vanishes in Z_{2^3}.

Example V.5: Now let us consider a non-vanishing polynomial $F = 3x^3 + 2x^2 + x + 2 = 3(x+3)(x+2)(x+1)$ in Z_{2^3}. Again, $n = 4$, but F cannot be factored into S_4 and hence $F_4 = 0$.

However, F can be factored into S_3 as $F = 3 \cdot S_3$. Therefore, from the above theorem we need to check whether the constraints on a_3 and b_3 are satisfied. In this case $b_3 = \frac{2^3}{(3!, 2^3)} = 4$. This implies that $a_3 \cdot 4 = 3$. However, there is no such integer a_3 in Z_{2^3}. This violates the necessary condition and hence the polynomial cannot vanish.

VI. Algorithmic Procedure to Identify Vanishing Polynomials

From the above results, we have been able to derive a systematic, complete, algorithmic procedure to identify whether or not a given polynomial vanishes over rings of the form Z_{2^m}, where m corresponds to the size of the datapath RTL that we wish to verify. The algorithm is given in Algorithm 1.

The algorithm takes as input the given uni-variate polynomial, $poly$, in variable x which is m-bits wide. The main procedure is outlined below:

1. Compute the Smarandache function value $n = SF(2^m)$; computational procedures are outlined in [43] [44].
2. Iteratively divide the polynomial ($poly$) by the S_k expressions, for $k = n$ downto 0.
3. If $k = n$ and $S_n | poly$, then it is a vanishing polynomial.
4. If the degree of $poly$ is less than that of S_k, then $F_n = 0$ and/or $a_k = 0$. Continue the division procedure with S_{k-1}.
5. Otherwise check for the constraints on a_k, b_k. If the constraints are not satisfied, then it is not a vanishing polynomial. If the constraints are satisfied, then the procedure iterates over the remaining terms.

```
ZERO_IDENTIFY(poly, m, x)
poly = Uni-variate polynomial in x;
m = Bit-width of poly;

/*Compute the Smarandache Function*/
n = SF(2^m);
isZero = 0; /*0 = continue; 1= non-zero polynomial*/

/*Compute values for b_k*/
for k = 0 to n − 1 do
   b_k = 2^m / gcd(k!, 2^m);
end for

/* Generate S_k polynomials */
S_0 = 1;
for k = 1 to n do
   S_k = (x + k) · S_{k−1};
end for

/* Iteratively Divide poly from S_n downto S_0*/
for k = n to 0 do
   /*Compute the quotient and remainder*/
   quo = quotient of poly divided by S_k, with coeffi cients reduced %2^m;
   rem = remainder of poly divided by S_k, coeffi cients reduced %2^m;
   if (quo == 0) then
      /* S_k is of higher degree than poly */
      poly = rem;
      continue;
   end if
   if ( (k == n) & (rem = 0) ) then
      /* S_n|poly and hence poly = 0 */
      break;
   end if
   if (quo % b_k ≠ 0) then
      /*If quo ≠ a_k · b_k, not a zero-polynomial*/
      isZero = 1;
      break;
   else
      /* constraint on a_k, b_k satisfi ed */
      if (rem == 0) then
         /* no more terms to test */
         break;
      else
         poly = rem;
      end if
   end if
end for
if (isZero == 1) then
   poly ≠ 0 in Z_{2^m}
else
   poly ≡ 0 in Z_{2^m}
end if
```

Algorithm 1: ZERO_IDENTIFY - Deduce whether a polynomial vanishes in $Z_{2^m}[x]$

6. The procedure converges with the correct answer on whether or not $poly \equiv 0$ in $Z_{2^m}[x]$.

Example VI.1: Let us demonstrate the operation of the algorithm on the example $poly = 4x^2 + 4x + 4$ in Z_{2^3}. In this case, $n = 4$. The algorithm proceeds as follows:

1. $k = 4$, degree of S_4 greater than that of $poly$, so $F_4 = 0$, continue.
2. $k = 3$, degree of S_3 greater than that of $poly$, so $a_3 = 0$, continue.
3. $k = 2$, $quo = 4, rem = 4$. Here $b_2 = 4, a_2 = 1$ so the conditions are satisfied. Now, $poly = rem = 4$ and continue.

4. $k = 1$, degree of S_1 greater than that of $poly$, so $a_1 = 0$, continue.
5. $k = 0$, $quo = 4, rem = 0$. Here $b_0 = 8$ and there does not exist an integer a_0 such that $4 = 8 \cdot a_0$. Therefore, in this situation, the conditions are violated and hence $4x^2 + 4x + 4$ is not a vanishing polynomial.

In the worst case, $n+1$ divisions by S_k expressions are performed on $poly$, where $n = SF(2^m)$. In a sense, the computational complexity depends upon the bit-vector size of the RTL datapath. The algorithm has been programmed using MAPLE [30], and its native division-related library calls are used for the process.

VII. Experimental Results and Analysis

Using the presented algorithm, we have been able to verify the equivalence of a variety of polynomial data-path computations. 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. The results are presented in Table I. The first example is an implementation of an anti-aliasing function [6]. The next four examples, from [10], are Horner form implementations of polynomial computations, commonly used in DSP. The remaining examples are implementations of elementary function computations. The last example is that of a "vanishing polynomial", specifically created to validate our concepts.

The data-flow graph for the above RTLs was extracted using GAUT [41]. Traversing the DFG from the inputs to the outputs, their polynomial representations $(f(x), g(x))$ were constructed. The datapath size (m) was also recorded. Subsequently, the difference $poly = f(x) - g(x)$ was computed, and the algorithm was invoked to test whether $poly$ vanishes in Z_{2^m}. Our approach was able to prove equivalence within a matter of seconds.

Experiments were also conducted with other verification techniques (BMD, SAT and MILP). Due to a large number of multiply operations, BDDs are clearly ill-suited for verification of such applications. BMDs have been successful in the verification of multipliers. Therefore, we used BMDs to perform the equivalency check. The designs were synthesized, corresponding netlists generated, and BMDs were constructed for the m-bit output words. Using BMDs, we were able to verify only four designs. It was observed that BMDs for expressions beyond degree-4 could not be constructed; the composition procedures did not complete.

Boolean Satisfiability solvers were also used for equivalency check. From the synthesized, gate-level netlists corresponding to the two designs, we generated miter circuits and converted them to CNF format. ZChaff [42] was used to prove equivalence via unsatisfiability testing. ZChaff was able to verify only a few cases within the time-out limit of 500s. Even those that it did verify, took substantially more time than our approach. For equivalence via MILP solving, linear inequalities (as those proposed in [23] for fixed-size bit-vectors) were created from their data-flow graphs. Equivalency $f \equiv g$ was tested via proving infeasibility of $(f - g) \neq 0$. The reason why MILP failed was because for computations of the form X_m^k, the vector needs to be expanded into its constituent m bits. LPSOLVE was used as the resolution engine.

The last benchmark, "vanishing polynomial", is the same expression as that given in Example V.1. Because it is a vanishing polynomial, the outputs always compute zero; which is what we wanted to verify. 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 memory, BMD-composition operations did not terminate.

VIII. Conclusions and Future Work

This paper has presented a technique to verify the equivalence of univariate polynomial RTL computations implemented with fixed-size bit-vectors. It has been shown that fixed-size bit-vector arithmetic manifests itself as polynomial algebra over finite integer rings of the form Z_{2^m}. Our approach formulates the equivalence verification problem $f(x)\%2^m \equiv g(x)\%2^m$ by transforming it into proving $(f(x) - g(x))\%2^m \equiv 0$. The concept of vanishing polynomials is exploited to derive an algorithm for this purpose. Results demonstrate that the proposed approach presents an efficient solution to verify the equivalence of high-level/RTL computations.

We are in the process of extending our technique to identify multi-variate vanishing polynomials. Unfortunately, such extensions are not straightforward and to the best of our knowledge solutions to the identification of multi-variate vanishing polynomials in Z_{2^m} are not available in literature. We would also like to extend our technique to verify multiple word-length architectures.

References

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

[2] K. Kum and W. Sung, "Word-length optimization for high-level synthesis of DSP systems", *in Intl. workshop Signal Processing Systems, SIPS*, 1998.

[3] R. J. B. T. Allenby, *Rings, Fields, and Groups: An Introduction to Abstract Algebra.*, E. J. Arnold, 1983.

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

[5] I. Niven and J. L. Warren, "A Generalization of Fermat's Theorem", *Proceedings of American Mathematical Society*, vol. 8, pp. 306–313, 1957.

[6] 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.

[7] M. Willems, H. Keding, T. Grotket, and H. Meyr, "Fridge: An Interactive Fixed-point Code Generation Environment for HW/SW Co-design", *in Intl. Conf. Acoustics, Speech, Signal Proc.*, 1997.

TABLE I

EQUIVALENCE CHECKING EXPERIMENTS: PARAMETERS AND RUN-TIME STATS. DEG = DEGREE OF POLYNOMIAL, m = DATAPATH SIZE.

Benchmark	Specs	Our approach	BMD	SAT-ZChaff	MILP
	Deg / m	Time (s)	Nodes / Time(s)	Vars / Clauses / Time(s)	Time(s)
Anti-alias function	6 / 16	6.81	NA / >500	3.9K / 107K / >500	>500
Horner Polynomial 1	4 / 16	3.56	2355 / 85	13K / 36K / >500	>500
Horner Polynomial 2	4 / 16	3.55	1574 / 47	12K / 34K / >500	>500
Horner Polynomial 3	4 / 16	4.53	6803 / 246	25K / 75K / >500	>500
Polynomial Unoptimized	4 / 16	2.79	2705 / 69	10K / 28K / >500	>500
$cos\ x$	6 / 32	4.5	NA / >500	60K / 173K / 31.15	>500
$cot^{-1}x$	9 / 32	5.37	NA / >500	140K / 406K / 76.18	>500
$erf\ x$	7 / 32	4.99	NA / >500	88K / 255K / 49.34	>500
$ln(\frac{1+x}{1-x})$	7 / 32	6.04	NA / >500	86K / 247K / 39.9	>500
Vanishing polynomial	10 / 8	1.8	NA / >500	-NA-	-NA-

[8] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement", *in DATE*, 1999.

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

[10] 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.

[11] MATLAB/Simulink, ", http://www.mathworks.com/products/simulink.

[12] 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.

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

[14] I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and their Applications", *in ICCAD*, pp. 188–191, Nov. 93.

[15] U. Kebschull, E. Schubert, and W. Rosentiel, "Multilevel Logic Synthesis based on Functional Decision Diagrams", *in EDAC*, pp. 43–47, 92.

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

[17] L. Arditi, "*BMDs can Delay the use of Theorem Proving for Verifying Arithmetic Assembly Instructions", in Srivas, editor, *In Proc. Formal methods in CAD*. Springer-Verlag, 1996.

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

[19] 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.

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

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

[22] 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.

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

[24] T.L. Rajaprabhu, A.K. Singh, A.M. Jabir, and D.K. Pradhan, "Modd for CF: A Compact Representation for Multiple Output Function", *in IEEE International High Level Design Validation and Test Workshop*, 2004.

[25] Aaron Stump, Clark W. Barrett, and David L. Dill, "CVC: A Cooperating Validity Checker", in Ed Brinksma and Kim Guldstrand Larsen, editors, *14th International Conference on Computer Aided Verification (CAV)*, vol. 2404 of *Lecture Notes in Computer Science*, pp. 500–504. Springer-Verlag, 2002, Copenhagen, Denmark.

[26] R. Bryant, S. Lahiri, and S. Seshia, "Modeling and Verifying Systems using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions", *CAV*, vol. , 2002.

[27] B. Akbarpour and S. Tahar, "Verification of FFT using HOL Theorem Proving", Technical report, Concordia Univ., 2004.

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

[29] L. Bachmair, A. Tiwari, and L. Vigneron, "Abstract congruence closure", *Journal of Automated Reasoning*, 2002.

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

[31] Mathematica, ", http://www.wri.com.

[32] 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 .

[33] D. Bayer and M Stillman, "MACAULAY 3.1", A system for computation in algebraic geometry and commutative algebra, Cornell University, Columbia University, 2000, http://www.math.columbia.edu/ bayer/Macaulay/ .

[34] A. Heck, *Introduction to Maple*, Springer-Verlag, 1993.

[35] B. Buchberger, *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, PhD thesis, Philosophiesche Fakultat an der Leopold-Franzens-Universitat, Austria, 1965.

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

[37] T. S. Freeman, G. M. Imirzian, E. Kaltofen, and L Yagati, "DAGWOOD A System for Manipulating Polynomials Given by Straight-Line Programs", *Journal of the ACM Transactions on Mathematical Software*, vol. 14, pp. 218–240, Sept. 1988.

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

[39] 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.

[40] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties and Algorithms*, Springer-Verlag, 1997.

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

[42] 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.

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

[44] Smarandache Function Computation, ", http://mathworld.wolfram.com/SmarandacheFunction.html.