Simulation Bounds for Equivalence Verification of Polynomial Datapaths Using Finite Ring Algebra*

Namrata Shekhar[1], Priyank Kalla[1], M. Brandon Meredith[2], Florian Enescu[2]

[1] Department of Electrical & Computer Engineering
University of Utah, Salt Lake City, UT-84112

[2] Department of Mathematics & Statistics
Georgia State University, Atlanta, GA-30303

Submitted For Review to the IEEE Transactions on VLSI -
Special Section on Design Verification and Validation: Theory and Techniques
Guest Editors: Dhiraj K. Pradhan and Ian G. Harris

Designated Contact Author:
Priyank Kalla
Email: kalla@ece.utah.edu
Ph: (801)-587-7617
Fax: (801)-581-5281

Abstract

Keywords: Simulation-based verification, Equivalence Checking, Bit-Vector Arithmetic, Finite Integer Rings, Polynomial Functions

This paper addresses simulation-based verification of high-level (algorithmic, behavioral or RTL) descriptions of arithmetic datapaths that perform polynomial computations over finite word-length operands. Such designs are typically found in digital signal processing (DSP) for audio/video and multi-media applications; where the word-lengths of input and output signals (bit-vectors) are predetermined and fixed according to the desired precision. Initial descriptions of such systems are usually specified as Matlab/C code. These are then automatically translated into behavioral/RTL descriptions (HDL) for subsequent hardware synthesis. In order to verify that the initial Matlab/C model is *bit-true equivalent* to the translated RTL, how many simulation vectors need to be applied?

This paper derives some important results that show that *exhaustive simulation is not necessary* to prove/disprove their equivalence. To derive these results, we model the datapath computations as polynomial functions over finite integer rings of the form Z_{2^m}; where m corresponds to the bit-vector word-length. Subsequently, by exploring some number-theoretic and algebraic properties of these rings, we derive an *upper bound on the number of simulation vectors* required to prove equivalence or to identify bugs. Moreover, these vectors cannot be arbitrarily generated. We identify exactly those vectors that need to be simulated. Experiments are performed within practical CAD settings to demonstrate the validity and applicability of these results. While our results cannot prove equivalence of an RTL description against its gate-level netlist, we show that their exists a large class of practical applications that can benefit from these results.

*This work is sponsored in part by: (i) NSF CAREER grant CCF-546859 and (ii) Georgia State University Research Initiation Grant.

I. INTRODUCTION

Increasing size and complexity of digital systems has resulted in a vast array of formal verification techniques which operate at different levels of abstraction. These include model checking, equivalence checking, theorem proving, among others. In spite of many advances in formal verification, simulation-based validation has remained an important method for ensuring functional correctness during various stages of the design cycle. This paper addresses the problem of simulation-based equivalence testing of arithmetic datapaths that perform polynomial computations over finite word-length operands. Such datapath oriented designs are commonly found in many digital signal processing applications, that perform ADD, MULT type of algebraic computations over bit-vectors of specified widths. Fig. 1 describes a typical design flow for such datapath intensive (signal-processing) applications, along with the context in which the equivalence verification problem appears.

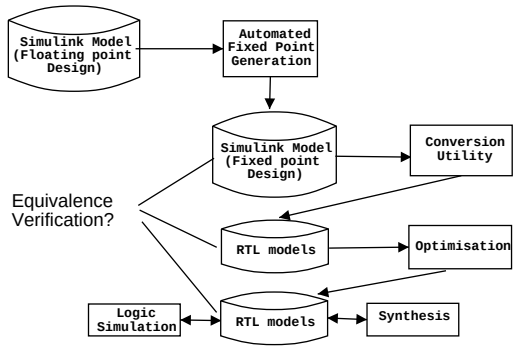

Fig. 1. The Equivalence Verification problem: Matlab to RTL design flow.

Initial algorithmic specifications (such as MATLAB models) of most signal processing applications involve data representation using floating-point formats. However, they are often implemented with fixed-point architectures, where the required precision dictates the bit-vector sizes of the variables. Various automated tools exist for this translation [1]. For synthesis and optimization purposes, these high-level descriptions may be subsequently converted to HDL using automatic utilities [2] [3]. Design optimization may be further achieved by applying high-level synthesis and restructuring operations on the translated RTL model [4] [5]. It is required to show that the translated and optimized RTL models are bit-true equivalent to the fixed-point specification.

Simulation is extensively used to validate the input-output behavior of the original model (at the MATLAB, C level); say to validate the pass-band of a filter. These vectors can then also be applied at the RT-level. Validation of the Matlab model is fast, even with a large number of test vectors, since it is compiled-code simulation. However, simulating the RTL model with the same set of vectors is generally slow. In general, equivalence proofs require exhaustive simulation, which makes simulation-based approaches impractical. In this regard, this paper derives an important result related to simulation-based verification of high-level descriptions of arithmetic datapaths. In particular, we show that:

1. Exhaustive simulation is not always necessary to verify equivalence or to find bugs;
2. An upper bound can be derived for the maximum number of test vectors required for this purpose; and
3. Which simulation vectors to choose for the equivalence proofs.

A. Bit-Vector Arithmetic and Finite Ring Algebra

Polynomial algebra provides a suitable platform for modeling such arithmetic intensive designs. However, for correct modeling of such systems, the bit-vector size needs to be accounted for in the polynomial model. 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* [6]. Therefore, we model the datapaths as *polynomial functions over the ring* Z_{2^m}, where the bit-vector word-lengths (m) dictate the cardinality of the ring. Subsequently, we exploit the number-theoretic and algebraic properties of these rings to systematically establish the claims mentioned above.

Let us motivate this issue using a few examples and put our contribution in perspective.

`module fixed_bit_width (`x, t_1, t_2`);`
`input` $[7:0]$ `x;`
`output` $[7:0]$ t_1, t_2`;`
`assign` $t_1[7:0] = 64 * x^2 + 192 * x - 169$`;`
`assign` $t_2[7:0] = 192 * x^2 + 64 * x + 87$`;`

Consider the RTL computations t_1 and t_2 shown above. The outputs t_1 and t_2 are symbolically distinct polynomials. However, because the datapath size is fixed to 8-bits, the computations t_1 and t_2 are bit-true equivalent. In other words, $t_1[7:0] \equiv t_2[7:0]$, or mathematically speaking, $t_1 \% 2^8 \equiv t_2 \% 2^8$. To prove the desired equivalence, exhaustive simulation would require that we compute and compare the values of t_1 and t_2 for $x = 0, 1, \ldots, 2^8 - 1$. This paper derives results which

prove that in the above case: i) if the two designs are not equivalent (bug), a maximum of 10 (specific) vectors are sufficient to capture the erroneous behavior; ii) if for these 10 vectors, no bug is detected, then the designs are indeed equivalent. Note that our approach presents a significant reduction in the number of required simulation vectors.

Now, consider a practical application: Given two degree-k polynomials $F_1(x)$ and $F_2(x)$, their coefficients can be represented as $(k+1)$-length vectors $A = (a_0, \ldots, a_k)$ and $B = (b_0, \ldots, b_k)$. Computing the *convolution* of such vectors results in another $k+1$ vector, according to:

$$c_i = \sum_{j=0}^{k} a_j b_{i-j} \quad 0 \leq i \leq k \tag{1}$$

where, c_i is the i^{th} component in the vector C. This procedure, however, has a complexity of $O(n^2)$. It is well-known [7] [8] that convolution can be effectively implemented in hardware by:
1. Computing the DFT of vectors A and B,
2. Calculating their pairwise product; and finally,
3. Taking the inverse DFT of the result. In other words, the result vector $(c'_0, c'_1, \ldots, c'_k)$ is computed as

$$C' = DFT^{-1}(DFT(A) \cdot DFT(B)) \tag{2}$$

This operation is shown in Fig. 2 for degree-3 polynomials. Such a 'divide-and-conquer' strategy results in a complexity of $O(n \cdot logn)$, and is a popular way of implementing the convolution of two vectors.

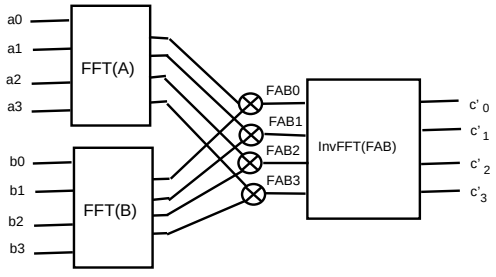

Fig. 2. Convolution of A and B.

Suppose that we intend to verify that both implementations compute the same values, say, for coefficients c_0 (obtained via direct convolution) and c'_0 (obtained via DFT-product-InvDFT). Further, as is often the case, assume that the entire datapath word-length in both cases is fixed to a certain width, say 4-bits. Each of the inputs (a_i and b_i, $0 \leq i \leq 3$) can take values between $\{0, \ldots, 2^4 - 1\}$, requiring a total of 2^{4*8} test vectors. We provide theoretical results which show that 6^8 vectors are sufficient to prove or disprove equivalence. Again, a method for generating these specific simulation vectors is also derived.

B. Problem Modeling

We model the arithmetic computations over bit-vectors 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 $Z_{2^{n_i}}$ corresponds to the finite set of integers $\{0, 1, \ldots, 2^{n_i} - 1\}$. 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 *polynomial function* (or polyfunction) from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}}$ to Z_{2^m} [9]. Here $Z_a \times Z_b$ represents the *Cartesian product* of Z_a and Z_b. In other words, the computation is modeled as a multi-variate polynomial $F(x_1, x_2, \ldots, x_d)$ $\%2^m$. The equivalence problem then corresponds to checking the congruence of two polynomials: $F(x_1, \ldots, x_d)\%2^m \equiv G(x_1, \ldots, x_d)\%2^m$.

The verification problem of Fig. 1 has seen a lot of interest recently in [10] [11] [12] [13] [14]. The works of [13] [14] use the same polynomial function model to derive a symbolic approach to prove/disprove equivalence of arithmetic datapaths. However, these works are restricted inasmuch as they can only provide a "yes/no" answer to the equivalence check. They cannot provide an error trace when bugs are detected. Moreover, our results also have implications on the applicability of the *fundamental theorem of algebra* which has been used in hardware design and verification [10] [11] [12], as described below.

C. Bit-Vector Arithmetic versus the Fundamental Theorem of Algebra

Lemma 1: Let $P(x)$ be a degree-k uni-variate polynomial. If $P(x) = 0$ for $(k+1)$ distinct values of x, then all the coefficients of $P(x)$ are zero.

The above lemma [15] is based on the *fundamental theorem of algebra* [16]. This theorem states that a degree-k univariate polynomial $P(x)$ has exactly k complex roots, unless all its coefficients are zero. Since integers are a special case of complex numbers, this theorem holds for the set Z as well. The work of [11] used this result for equivalence verification by modeling arithmetic datapaths as polynomials, and showed that $k+1$ vectors were sufficient to prove equivalence of any given degree-k polynomials. It was later extended in [15] to be appli-

cable to multi-variate polynomials as well, and was further applied to reduce the complexity of model-checking. Also, [10] used the same concepts for extracting polynomial representations from RTL descriptions for high-level synthesis purposes.

However, the above results are only relevant in *unique factorization domains* (UFDs), such as the set of real numbers (R), the set of integers (Z), finite fields ($Z_p, GF(p^n), p$ =prime) etc. In our context, the specific modulo value (2^m) does not correspond to unique factorization, due to the presence of *zero-divisors* (*e.g.*, $4 \neq 2 \neq 0, 4 \cdot 2 = 0 \% 8$), and correspondingly due to lack of multiplicative inverses. In other words, finite rings of residue classes Z_{2^m} do not form a field. As a result, factorization is not unique in such rings, as shown for the polynomial $F(x) = x^2 + 6x \in Z_8$ below.

$$\begin{aligned} x^2 + 6x &= x(x+6) \ \%2^3 \\ &= (x+4)(x+2) \ \%2^3 \end{aligned}$$

The polynomial $F(x)$ has a degree $k = 2$, but can be factorized in two non-unique ways; corresponding to four distinct roots. In such cases, Lemma 1 does not hold; simulating for $k+1 = 3$ values such as $x = 0, 2, 4$, does show $F = 0$ but that does not mean that all the coefficients of $F(x)$ are zero. Indeed, for $x = 1, F(x) = 7 \neq 0$. Therefore, simulating for only $k+1$ vectors is insufficient for verification.

Clearly, for bit-vector arithmetic, properties of this class of rings (of the type Z_{2^m}) need to be investigated further for simulation-based verification. We will now explore results for polynomial functions over such finite integer rings with applications in simulation-based verification of arithmetic datapaths.

D. Scope of the Paper

The approach presented in this paper has been applied to verify high-level descriptions of arithmetic datapaths, such as those in C and RTL (VERILOG/VHDL), some of which were automatically generated by MATLAB (Simulink and filter design toolboxes) [3]. Our technique is applicable to designs that implement unsigned and two's complement (overflow) arithmetic. In the DSP domain, however, rounding as well as saturation are also common modes of approximation. Modeling such architectures as polynomial functions over finite rings is significantly more involved and is not the subject of this paper. For the same reason, verification of (behavioral) RTL against its corresponding gate-level implementation (netlist) is also not dealt with in the paper. Even within this scope, we demonstrate that there exists a large class of applications that can benefit from our results.

E. Paper Organization

This paper is organized as follows: The next section reviews related work in simulation-based verification. Section III covers preliminary concepts and background material regarding polynomial functions and finite ring theory. Section IV describes the proposed results for univariate polynomials and provides the mathematical foundation for their support. These results are extended for multi-variate computations in Section V. Finally, Section VI describes the experimental setup and results, while Section VII concludes the paper.

II. Previous Work

Bryant, in [17] [18], used three-valued logic simulation to reduce the required number of simulation vectors for circuit verification. Subsequently, [19] incorporated some of these techniques into their framework, which provided a method for design verification at different levels of abstraction. Brand [20] proposed exploiting information from the design specification to significantly reduce the complexity of simulation. Clarke *et al.* further researched the problem of specifications and generators in [21]. However, BDDs were used to demonstrate a practical approach to this problem in the *SimGen* project [22]. Later on, Shimizu *et al.* [23] [24] automated this approach to verify large industrial designs. The above methods are focused towards generating the appropriate number of test vectors to ensure sufficient verification coverage. Our approach, on the other hand, applies polynomial methods that make exhaustive simulation unnecessary for arithmetic datapaths.

An algebraic approach to reducing the test vector set was proposed in [11], and later extended in [12]. These works model the given datapaths as polynomials and apply the fundamental theorem of algebra to verify the descriptions. Recently, [15] extended the theorem to be applicable to multivariate polynomials as well. However, as mentioned earlier, these results do not always hold over the proposed model, which accounts for bit-vector sizes of the input and output variables.

The works which come closest to ours have been presented in [13][14]. They model the given datapath as a polynomial function over a system of finite integer rings. [13] proves equivalence of fixed-size datapaths by using canonical representations of polynomials over finite rings. The concept of vanishing polynomials is used in [14] to derive a symbolic approach to test equivalence

of polynomial functions. Both approaches use some form of algebraic simplification, which suffers from the well-known intermediate-expression swell problem [25]. In addition, these techniques cannot provide an error trace whenever non-equivalence is detected.

This paper reinterprets the polynomial function model and extends the concepts presented in [9] to derive a novel solution for simulation-based verification of high-level descriptions of arithmetic datapaths. The next few sections cover some preliminary concepts, and then derive the theoretical contributions of this paper. Practical application of our work is subsequently demonstrated.

III. Preliminaries

This section briefly reviews basic commutative algebra concepts to put our polynomial equivalence problems in perspective. The material is mostly referred from [6].

Definition III.1: An **Abelian group** is a set G and a binary operation $'+'$ satisfying:

- *Closure:* For every $a, b \in G, a + b \in G$.
- *Associativity:* For every $a, b, c \in G, a + (b + c) = (a + b) + c$.
- *Commutativity:* For every $a, b \in G, a + b = b + a$.
- *Identity:* There is an identity element $0 \in G$ such that for all $a \in G; a + 0 = a$.
- *Inverse:* If $a \in G$, then there is an element $a^{-1} \in G$ such that $a + a^{-1} = 0$.

The set of integers Z, for instance, forms an abelian group under addition.

Definition III.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, R is an Abelian group with respect to addition with additive identity element 0, and the following properties are satisfied:

- *Multiplicative Closure:* For every $a, b \in$ R, $a \cdot b \in$ R.
- *Multiplicative Associativity:* For every $a, b, c \in$ R, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
- *Multiplicative Commutativity:* For every $a, b \in$ R, $a \cdot b = b \cdot a$.
- *Multiplicative Identity:* There is an identity element $1 \in$ R such that for all $a \in$ R, $a \cdot 1 = a$.
- *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* ($\%n$) according to the rules below. For our application, $n = 2^m$.

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

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

$$(-a)\%n = (n - a\%n)\%n \quad (5)$$

Definition III.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 III.4: A **zero divisor** is a non-zero element x of a ring R, for which $x \cdot y \equiv 0$, where y is some other non-zero element of R and the multiplication $x \cdot y$ is defined according to Eqn. (4).

As an example, consider the non-zero integers 2 and 4 in the ring Z_8. Since, $2 \cdot 4 \equiv 0\ \%\ 8$, 2 and 4 are zero-divisors of each other. A commutative ring that has no *zero divisors* is known as an **integral domain**. The set of integers, Z, is an example.

Definition III.5: A **field** F is a commutative ring with unity, where every element in F, except 0, has a multiplicative inverse, i.e. $\forall\ a \in F - \{0\}, \exists\ \hat{a} \in F$ such that $a \cdot \hat{a} = 1$.

The system Z_n forms a field if and only if n is prime. Hence, Z_{2^m} (for $m > 1$) 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 since Euclidean algorithms for division and factorization are no longer applicable.

Definition III.6: Let R be a ring. A **polynomial** over R in the indeterminate x is an expression of the form:

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k = \sum_k a_i x^i \quad (6)$$

$\forall a_i \in R$. Elements a_i are coefficients, k is the degree. The element a_k is called the **leading coefficient**; when $a_k = 1$, 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** $R[x]$. Similarly, $R[x_1, \ldots x_d]$ denotes a ring of multi-variate polynomials in d variables. Note that, when $R = Z_{2^m}$, the corresponding polynomials are evaluated $\%2^m$.

Functions over finite rings that can be represented by polynomials are generally termed as **polynomial functions** (or *polyfunctions*). Singmaster [26] initially studied various properties of polynomial functions of the type

$f : Z_n \rightarrow Z_n$. Subsequently, Chen extended the study to analyze polyfunctions of the type $f : Z_n \rightarrow Z_m$ [27] and further to those of the type $f : Z_{n_1} \times Z_{n_2} \times \cdots \times Z_{n_d} \rightarrow Z_m$ [9]. The following definition of such a polynomial function is taken from [9], and modified, for our application, to rings modulo an integer power of 2.

Definition III.7: A function f from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$ is said to be a polynomial function (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 $x_i \in Z_{2^{n_i}}$, $i = 1, 2, \ldots, d$ and $\equiv$ denotes congruence (mod 2^m).

Example III.1: Let $f : Z_2 \times Z_{2^2} \rightarrow Z_{2^3}$ 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 polynomials are called **vanishing polynomials** ($F \equiv 0$) and their underlying functions correspond to *nil polyfunctions*.

Example III.2: Consider the function $f(x_1, x_2) : Z_2 \times Z_{2^2} \rightarrow Z_{2^3}$ represented by the polynomial $F = 4x_1x_2^2 + 4x_1x_2$. While F has non-zero coefficients, $F \ \% \ 2^3 \equiv 0, \forall x_1 \in Z_2, \forall x_2 \in Z_4$.

Properties of such polynomials have been extensively studied in number theory and commutative algebra [28] [26] [27] [9]. We will later review some of these results in the context of deriving simulation bounds for equivalence of polyfunctions. Examples are used to demonstrate relevant results; their corresponding proofs can be found in [27] [9] and are therefore not reproduced here.

Definition III.8: Two polynomials F_1 and F_2 are equivalent $\% \ 2^m$, if their associated polyfunctions f_1 and f_2 from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$ are equal. We denote equivalence as $F_1 \ \% \ 2^m \equiv F_2 \ \% \ 2^m$.

IV. Univariate Polynomials

Given a polynomial function $f(x)$ and its representative polynomial $F(x)$ over Z_{2^m}, we need to reinterpret F in a way that would be more suitable for our purposes. In this context, we first define the forward difference operator (Δ) [29], which is a discrete analog to the derivative of a polynomial function.

Definition IV.1: Let $F(x)$ be a degree-p polynomial over Z.

$$\begin{aligned}
(\Delta F)(x) &= F(x+1) - F(x) \\
(\Delta^2 F)(x) &= (\Delta F)(x+1) - (\Delta F)(x) \\
&\vdots \\
(\Delta^p F)(x) &= \sum_{k=0}^{p} (-1)^k \binom{p}{k} F(p-k+x) \qquad (7)
\end{aligned}$$

Let us now demonstrate the application of the forward difference operator on a degree-2 polynomial in x.

Example IV.1: Let $F(x) = 4x^2 + 3x$ be a polynomial in Z. Applying the Δ operator described in Def. IV.1, we get

$$\begin{aligned}
(\Delta F)(x) &= F(x+1) - F(x) \\
&= 4(x+1)^2 + 3(x+1) - (4x^2 + 3x) \\
&= 8x + 7 \\
(\Delta^2 F)(x) &= (\Delta F)(x+1) - (\Delta F)(x) \\
&= 8(x+1) + 7 - (8x + 7) \\
&= 8 \\
(\Delta^3 F)(x) &= (\Delta^2 F)(x+1) - (\Delta^2 F)(x) \\
&= 0
\end{aligned}$$

We now state Newton's interpolation formula based on the above definition. The proof for this formula is widely available in literature [30] and is not reproduced here.

Proposition 1: If $F(x)$ is a polynomial of degree p with integral coefficients, then it can be written as

$$F(x) = \sum_{k=0}^{p} (\Delta^k F)(0) \binom{x}{k} \qquad (8)$$

Note that the binomial term in Eqn. (8) can be expanded according to,

$$\binom{x}{k} = \frac{x(x-1)\cdots(x-k+1)}{k!} \qquad (9)$$

The numerator of this term represents (in polynomial form) a product of i consecutive numbers in x. More formally, we have the following definition [27]:

Definition IV.2: Falling factorials of degree k are defined as:

- $Y_0(x) = 1$
- $Y_1(x) = x$
- $Y_2(x) = x(x-1)$

  $\vdots$
- $Y_k(x) = x \cdot (x-1) \cdots (x-k+1)$

Eqn. (8) can now be written as:

$$\begin{aligned} F(x) &= \sum_{k=0}^{p} (\Delta^k F)(0) \frac{x(x-1)\cdots(x-k+1)}{k!} \\ &= \sum_{k=0}^{p} \frac{(\Delta^k F)(0)}{k!} Y_k(x) \\ &= \sum_{k=0}^{p} c_k \cdot Y_k(x) \end{aligned} \quad (10)$$

where $c_k = \frac{(\Delta^k F)(0)}{k!}$. Since $F(x)$ has integral coefficients, c_k is always an integer for $0 \leq k \leq p$. Note that $F(x)$ has a maximum of $(p+1)$ terms in the sum. This is illustrated in the following example:

Example IV.2: Consider the polynomial $F(x) = 4x^2 + 3x$ in Z. Here, the degree of $F(x)$ is $p = 2$ and F can be expanded into $p + 1 = 3$ terms. Using Eqn. (10) and the $(\Delta^k F)(x)$ values computed in Example IV.1, this polynomial can be expressed as,

$$\begin{aligned} F(x) &= \sum_{k=0}^{2} c_k \cdot Y_k(x) \\ &= c_0 \cdot Y_0(x) + c_1 \cdot Y_1(x) + c_2 \cdot Y_2(x) \\ &= \frac{(\Delta^0 F)(0)}{0!} \cdot 1 + \frac{(\Delta^1 F)(0)}{1!} \cdot x \\ &\quad + \frac{(\Delta^2 F)(0)}{2!} \cdot x(x-1) \\ &= 7x + 4x(x-1) \end{aligned}$$

The above interpretation is useful in deriving the results proposed in this paper. However, before we proceed to the relevant theorems, we briefly review the concept of vanishing polynomials.

A. Vanishing Polynomials over Z_{2^m}

According to a fundamental result in number theory, for any $n \in N$, it is possible to find the *least* value $\lambda \in N$ such that $n|\lambda!$. We denote this value as $SF(n)$ [1] *i.e.* $\lambda = SF(n)$ [33]. Consider the ring Z_{2^m}. Here, $SF(2^m)$ corresponds to the least value such that $2^m|\lambda!$.

The significance of the above concept can be explained as follows. In the ring Z_{2^3}, let $SF(2^3) = 4$ as 8 divides 4! ($= 4 \times 3 \times 2 \times 1 = 8 \times 3$). Note that 8 does not divide 3!, and hence the *least* λ in question is 4. Consequently, any integer that can be factored into a product of (at least) 4 consecutive numbers will be divisible by 2^3. In other words, an integer that can be factored into a product of λ consecutive numbers will be $\equiv 0 \% 2^3$ and vanish in Z_{2^3}. Now consider a polynomial $F(x)$ in the ring Z_{2^3}, such that $2^3|F(x)$. Therefore, if F, evaluated at x, can be represented as the product of 4 consecutive numbers (depending on x), then F would vanish in Z_{2^3}. Falling factorials are natural examples of such polynomials. Indeed, if $F(x) = Y_4(x) = x(x-1)(x-2)(x-3)$, then $F(x) \equiv 0 \ \% \ 2^3$. More formally, we have the following result:

Lemma 2: Any polynomial in $Z_{2^m}[x]$ that can be expressed as a multiple of $Y_\lambda(x)$ will be divisible by 2^m and vanish.

Example IV.3: Consider the polynomial $F(x)$ corresponding to to the function $f : Z_{2^4} \rightarrow Z_{2^4}$:

$$F(x) = x^6 + x^5 + 5x^4 + 15x^3 + 2x^2 + 8x \quad (11)$$

Here, $\lambda = SF(2^4) = 6$. Therefore, if $F(x)$ can be factored into a product of 6 consecutive numbers in x, then $F(x)$ is a vanishing polynomial in Z_{2^4}. In fact, $F(x) \equiv x(x-1)(x-2)(x-3)(x-4)(x-5) \ \% \ 2^4 \equiv Y_6(x) \ \% \ 2^4$; hence $F(x) \equiv 0 \ \% \ 2^4$.

When a polynomial cannot be factorized according to Lemma 2, it can still vanish. In this regard, Chen [27] identified the constraints on the coefficients which would determine whether the polynomial in question would vanish. We state the following result.

Lemma 3: The expression $c_k \cdot Y_k(x) \equiv 0$ in $Z_{2^m}[x]$ if and only if $\frac{2^m}{(k!, 2^m)} | c_k$;
where,

- c_k is an arbitrary integer in Z,
- $Y_k(x)$ is as defined above,
- $(k!, 2^m)$ is the greatest common divisor (GCD) of $k!$ and 2^m and
- $k \in Z$, such that $0 \leq k < \lambda$.

In other words, if $c_k \equiv 0 \ \% \ \frac{2^m}{(k!, 2^m)}$ then $c_k \cdot Y_k(x) \equiv 0$.

Example IV.4: Let $F(x) = 4x^2 - 4x$ over $Z_{2^3}[x]$. Note that $F(x) = 4(x)(x-1) = 4 \cdot Y_2(x)$. Here, $\lambda = 4$, degree $k = 2 < \lambda$ and coefficient $c_2 = 4$. Therefore, $F(x)$ cannot be written as a factor of $Y_4(x)$, according to Lemma 2. However, $c_2 = 4 \equiv 0 \ \% \ \frac{2^3}{(2!, 2^3)}$. Because the condition in Lemma 3 is satisfied, $F(x) \ \% \ 2^3 \equiv 0$. If c_2 were replaced by 3, then $F(x) = 3(x)(x-1) \ \% \ 2^3$ would not be a vanishing polynomial as $c_2 = 3 \neq 0 \ \% \ \frac{2^3}{(2!, 2^3)}$.

[1] This function has been extensively studied in number theory. It was initially studied by Lucas [31] and Kempner [32] and was recently re-visited upon by Smarandache [33]. In recent literature it is often referred to as the *Smarandache function* and hence we refer to it as $SF(n)$.

B. Application to Equivalence Verification

The properties of vanishing expressions can be applied to reduce any given polynomial over a finite integer ring. Let $F(x)$ be any polynomial corresponding to a polyfunction $f : Z_{2^m} \rightarrow Z_{2^m}$. Now, $F(x)$ can be written as [27],

$$F(x) = Q_\lambda(x)Y_\lambda(x) + R(x) \tag{12}$$

where,

- $\lambda = SF(2^m)$,
- $Q_\lambda(x)$ is an arbitrary polynomial in $Z[x]$,
- $Y_\lambda(x)$ represents the product of λ consecutive numbers in x and
- $R(x)$ is an arbitrary polynomial in $Z[x]$, such that $degree\ (R(x)) < \lambda$.

To obtain the above representation, we divide $F(x)$ by $Y_\lambda(x)$, resulting in the quotient $Q_\lambda(x)$ and remainder $R(x)$. Note that $Q_\lambda(x) \cdot Y_\lambda(x)$ represents a multiple of λ consecutive numbers and is $\equiv\ 0\ \%2^m$, according to Lemma 2. Therefore,

$$\begin{aligned} F(x) &= Q_\lambda(x)Y_\lambda(x) + R(x) \\ &= 0 + R(x) \\ &= R(x) \end{aligned} \tag{13}$$

The degree of $F(x)$ has now been reduced to $< \lambda$. Let us explain this representation with an example.

Example IV.5: Consider the polynomial $F(x) = x^4 + 3x^3 + 7x^2 + 6x$ over Z_{2^3}. The degree of $F(x)$ is 4. We compute $\lambda = SF(2^3) = 4$, and divide $F(x)$ by Y_4 to represent it as:

$$F(x) = Y_4(x) + x^3 + 4x^2 + 4x$$

Here, $Q_4 = 1$ and $R(x) = x^3 + 4x^2 + 4x$. Now, $Y_4(x)$ represents a product of λ consecutive numbers in Z_{2^3}, and evaluates to $0\ \%\ 2^3$. Thus, $F(x) = x^3 + 4x^2 + 4x$, where the degree is now $3 < \lambda$.

We now state the following results:

Theorem 1: Let $F(x) \in Z[x]$ be a polynomial and let $f : Z_{2^m} \rightarrow Z_{2^m}$ be its associated polyfunction. To prove that $F(x)$ vanishes in Z_{2^m}, it is sufficient to show that $F(x) \equiv 0\ \%\ 2^m$ for **any** λ **consecutive** integer values of x. Here, λ is the least integer such that $2^m | \lambda!$.

Proof:

Given a polynomial $F(x)$ corresponding to the polyfunction $f : Z_{2^m} \rightarrow Z_{2^m}$, we can use Eqn. (13) to reduce it to:

$$\begin{aligned} F(x) &= Q_\lambda(x)Y_\lambda(x) + R(x) \\ &= R(x) \end{aligned}$$

From Newton's interpolation formula, we know that any function with integral coefficients can be represented according to Eqn. (10). Note that Eqns. (7) - (10) also hold over Z_{2^m}, since the coefficients remain integral. We can now rewrite $F(x)$ as

$$\begin{aligned} F(x) = R(x) &= \sum_{k=0}^{\lambda-1} \frac{(\Delta^k R)(0)}{k!} Y_k(x) \\ &= \sum_{k=0}^{\lambda-1} c_k Y_k(x) \end{aligned} \tag{14}$$

since $deg(R(x)) < \lambda$. According to Lemma 3, if all the coefficients c_k of this expression reduce to 0 when computed $\%\frac{2^m}{(2^m, k!)}$, then $F(x)$ vanishes in Z_{2^m}. Thus, we need to verify this condition for each value c_k, where $0 \leq k < \lambda$. Further, each such computation requires evaluating $F(x)$ according to Eqn. (14). This implies that $F(x)$ is evaluated a maximum of λ *consecutive* times. Thus, the above theorem holds for integers from 0 to $\lambda - 1$. Since $F(x) \equiv 0 \forall x \Leftrightarrow F(x+a) \equiv 0 \forall x$, where $a \in Z$, the Theorem 1 is also true for *any* λ consecutive numbers. ■

Example IV.6: Let us explain this concept using the polynomial $F(x)\ \%\ 2^3$ from Example IV.5. Since $\lambda = 4$, we need to compute c_k for $0 \leq k < 4$.

- $k = 0$: $c_0 = \frac{(\Delta^0 F)(0)}{0!}\ \%\ \frac{2^3}{(2^3, 0!)} = 0$
- $k = 1$: $c_1 = \frac{(\Delta^1 F)(0)}{1!}\ \%\ \frac{2^3}{(2^3, 1!)} = 1$

Since $c_1 \neq 0\ \%\ 2^3$, $F(x)$ is not a vanishing polynomial.

Corollary 1: Let $F_1(x)$ and $F_2(x)$ be two polynomials with coefficients in Z. To prove $F_1(x)\ \%\ 2^m = F_2(x)\ \%\ 2^m$, it is sufficient to show that $F_1(x) \equiv F_2(x)$ in Z_{2^m} for **any** λ **consecutive** integer values of x.

Proof: We need to prove that:

$$\begin{aligned} F_1(x)\ \%\ 2^m &\equiv F_2(x)\ \%\ 2^m \\ \Rightarrow F_1(x) - F_2(x) &\equiv 0\ \%\ 2^m \end{aligned}$$

From Theorem 1, we know that any λ consecutive values of x are sufficient to prove that $F_1(x) - F_2(x)$ vanishes in Z_{2^m}. Corollary 1 directly follows from this result. ■

Example IV.7: Consider the two symbolically distinct polynomials $F_1(x) = 2x^5 + 13x^4 + 15x^2 + 2x + 8$ and $F_2(x) = x^4 + 10x^3 + 3x^2 + 2x + 8$. To prove that $F_1\ \%\ 2^4 \equiv F_2\ \%\ 2^4$, we need to compare the values of $F_1(x)$ and $F_2(x)$ for *any* $\lambda = SF(2^4) = 6$ consecutive values of x (instead of all 2^4 values). Therefore,

$$F_1(0) = 8; \quad F_2(0) = 8$$

$$F_1(1) = 8; \quad F_2(1) = 8$$
$$F_1(2) = 8; \quad F_2(2) = 8$$
$$F_1(3) = 8; \quad F_2(3) = 8$$
$$F_1(4) = 0; \quad F_2(4) = 0$$
$$F_1(5) = 0; \quad F_2(5) = 0$$

Since $F_1(x)$ and $F_2(x)$ are equivalent for all 6 values, Corollary 1 implies that the two polynomials are equivalent for all x in Z_{2^4}.

Example IV.8: Now consider the polynomials $F_1(x) = x^5 + 15x^4 + 5x^3 + x^2 + 2x + 8$ and $F_2(x) = x^4 + 10x^3 + 3x^2 + 2x + 8$ over Z_{2^4}. These polynomials are not equivalent $\% \ 2^4$. Let us apply the above result to determine their points of difference.

$$F_1(2) = 8; \quad F_2(2) = 8$$
$$F_1(3) = 0; \quad F_2(3) = 8$$
$$F_1(4) = 0; \quad F_2(4) = 0$$
$$F_1(5) = 0; \quad F_2(5) = 0$$
$$F_1(6) = 0; \quad F_2(6) = 0$$
$$F_1(7) = 0; \quad F_2(7) = 0$$

Within $\lambda = 6$ values, we were able to find that $F_1(3) \neq F_2(3) \ \% \ 2^4$.

V. Equivalence of Multi-variate Polynomials

In Section I, we had shown how multiple-word-length bit-vector computations can be modeled as polynomial functions from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \cdots \times Z_{2^{n_d}}$ to Z_{2^m}. Moreover, we had noted that a fixed-size datapath with multiple variables is a special case of the above, where $n_1 = \cdots = n_d = m$. Furthermore, in the previous section, we had sought to exploit the concept of vanishing polynomials to determine the required number of simulation vectors over $Z_{2^m}[x]$. We will now extend these concepts to address multi-variate polynomials.

In what follows, we use the multi-index notation: $\mathbf{k} = \langle k_1, k_2, \ldots, k_d \rangle$ are the (non-negative) degrees corresponding to the d input variables $\mathbf{x} = \langle x_1, x_2, \ldots, x_d \rangle$, respectively. Here, $\mathbf{k}$ and $\mathbf{x}$ are d-tuples, where $k_i \in Z^+$ and $x_i \in Z_{2^{n_i}}$, for $i = 1, 2, \ldots, d$. Z^+ denotes the set of non-negative integers

Definition V.1: Let $\mathbf{k} = \langle k_1, k_2, \ldots, k_d \rangle \in (Z^+)^d$. We define,

$$\mathbf{Y_k}(\mathbf{x}) = \prod_{i=1}^{d} Y_{k_i}(x_i) = Y_{k_1}(x_1) \cdot Y_{k_2}(x_2) \cdots Y_{k_d}(x_d), \tag{15}$$

where $Y_{k_i}(x_i)$ is the falling factorial of degree k_i in variable x_i, as defined in Definition IV.2.

We now extend Newton's interpolation formula for d variables.

Definition V.2: Let F be a polynomial in d variables $x_1, x_2, \ldots, x_d$ with degrees $\mathbf{p} = \langle p_1, p_2, \ldots, p_d \rangle$. We define the multi-variate $\mathbf{\Delta}$ operator as:

$$(\mathbf{\Delta^p} F)(\mathbf{x}) = (\Delta_1^{p_1} F)(\mathbf{x}) \circ \cdots \circ (\Delta_d^{p_d} F)(\mathbf{x}) \tag{16}$$

Here, $\circ$ denotes the successive application of the Δ operator for each of the d variables.

Example V.1: Let us now apply the $\mathbf{\Delta}$ operation to the polynomial $F(x_1, x_2) = 4x_1x_2$. The degrees of $\langle x_1, x_2 \rangle$ are $\langle 1, 1 \rangle$.

$$\begin{aligned}
(\mathbf{\Delta}^{\langle 0,1 \rangle} F)(x_1, x_2) &= 4x_1(x_2 + 1) - 4x_1x_2 \\
&= 4x_1 \\
(\mathbf{\Delta}^{\langle 1,0 \rangle} F)(x_1, x_2) &= 4(x_1 + 1)x_2 - 4x_1x_2 \\
&= 4x_2 \\
(\mathbf{\Delta}^{\langle 1,1 \rangle} F)(x_1, x_2) &= 4(x_2 + 1) - 4x_2 \\
&= 4
\end{aligned}$$

Proposition 2: Let $F(x_1, x_2, \ldots, x_d)$ be a polynomial with degree $\mathbf{p}$. Then, Newton's formula can be written in multi-index notation as,

$$\begin{aligned}
F(\mathbf{x}) &= \sum_{\mathbf{k} \leq \mathbf{p}} (\mathbf{\Delta^k} F)(\mathbf{0}) \binom{\mathbf{x}}{\mathbf{k}} \\
&= \sum_{\mathbf{k} \leq \mathbf{p}} \frac{(\mathbf{\Delta^k} F)(\mathbf{0})}{k_1! \ldots k_d!} \prod_{i=1}^{d} Y_{k_i}(x_i) \\
&= \sum_{\mathbf{k} \leq \mathbf{p}} \frac{(\mathbf{\Delta^k} F)(\mathbf{0})}{\mathbf{k}!} \mathbf{Y_k}(\mathbf{x}) \\
&= \sum_{\mathbf{k} \leq \mathbf{p}} c_{\mathbf{k}} \mathbf{Y_k}(\mathbf{x})
\end{aligned} \tag{17}$$

where $c_{\mathbf{k}} = \frac{(\mathbf{\Delta^k} F)(\mathbf{0})}{\mathbf{k}!}$.

In the above, $\mathbf{k} \leq \mathbf{p}$ implies that $k_1 \leq p_1$, $k_2 \leq p_2$, $\ldots$, $k_d \leq p_d$. The coefficients $c_{\mathbf{k}}$ are computed for all vectors $\mathbf{k} = \langle k_1, \ldots, k_d \rangle$, where each $k_i = 0, \ldots, p_i - 1$. This corresponds to a maximum of $\prod_{i=1}^{d} (p_i + 1)$ coefficients.

Example V.2: Let us represent the polynomial $F = 3x_2^2 + 4x_1x_2$ according to Proposition 2. Here, the degrees of x_1 and x_2 can be represented as $\mathbf{p} = \langle 1, 2 \rangle$.

$$F = \sum_{\langle k_1, k_2 \rangle \leq \langle 1,2 \rangle} c_{\langle k_1, k_2 \rangle} Y_{\langle k_1, k_2 \rangle}(x_1, x_2)$$

$$
\begin{aligned}
&= \frac{(\mathbf{\Delta^k} F)(0,0)}{0!\cdot 0!} Y_{\langle 0,0\rangle}(x_1,x_2) + \cdots \\
&\quad + \frac{(\mathbf{\Delta^k} F)(0,0)}{1!\cdot 2!} Y_{\langle 1,2\rangle}(x_1,x_2) \\
&= 3\cdot x_2 + 3\cdot x_2(x_2-1) + 4\cdot x_1\cdot x_2 \\
&= 3\cdot Y_{\langle 0,1\rangle}(x_1,x_2) + 3\cdot Y_{\langle 0,2\rangle}(x_1,x_2) \\
&\quad + 4\cdot Y_{\langle 1,1\rangle}(x_1,x_2)
\end{aligned}
$$

A. Multi-variate Vanishing Polynomials

As in the univariate case, we review results related to nil polyfunctions. Lemma 2 can be extended as follows: If a polynomial in d variables can be factorized into a product of λ consecutive numbers in *at least one of the variables* x_i, then it vanishes $\%\ 2^m$. The following example illustrates this idea.

Example V.3: Consider the polynomial $F(x_1,x_2) = x_1^4x_2 + 2x_1^3x_2 + 3x_1^2x_2 + 2x_1x_2$ over Z_{2^2}. Here, $\lambda = 4$ and the highest degrees of x_1 and x_2 are $k_1 = 4$, and $k_2 = 1$, respectively. Now $F \,\%\, 2^2$ can be equivalently written as $F = Y_4(x_1)\cdot Y_1(x_2)\,\%\, 2^2 = Y_{<4,1>}(x_1,x_2)\,\%\, 2^2$. Since $F \,\%\, 2^2$ can be represented as a product of 4 consecutive numbers in x_1, $2^2|F$ and $F \equiv 0$ in Z_{2^2}.

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 generalize these results to analyze any arbitrary polynomial 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, μ_i [9]:

Definition V.3:

$$\mu_i = min\{2^{n_i}, \lambda\}; i = 1,2,\ldots,d \tag{18}$$

where $\lambda = SF(2^m)$.

We now present the following results from [9]:

Lemma 4: Let $\mathbf{k} = \langle k_1,k_2,\ldots,k_d\rangle \in (Z^+)^d$, where Z^+ denotes the set of non-negative integers. Then, $\mathbf{Y_k}(\mathbf{x}) \equiv 0$ if and only if $k_i \geq \mu_i$, for some i.

Example V.4: Let $F = x_1^2x_2 - x_1x_2$ be a polynomial corresponding to $f : Z_2 \times Z_{2^2} \rightarrow Z_{2^3}$. We show that F is a vanishing polynomial as F can be written according to:

$$
\begin{aligned}
x_1^2x_2 - x_1x_2 &\equiv x_1(x_1-1)x_2 \\
&\equiv Y_{\langle 2,1\rangle}(x_1,x_2) \\
&\equiv 0
\end{aligned}
$$

Here, $\lambda = 4$ and the degrees of x_1 and x_2 are $k_1 = 2$ and $k_2 = 1$. Now $\mu_1 = min\{2,4\} = 2 = k_1$ and $\mu_2 = min\{2^2,4\} = 4 > k_2$. Since the condition in Lemma 4 is satisfied ($\mu_1 = k_1$), $F \equiv 0 \,\%\, 2^3$.

Lemma 5: The expression $c_{\mathbf{k}} \cdot \mathbf{Y_k}(\mathbf{x}) \equiv 0$ if and only if $\frac{2^m}{(2^m, \prod_{i=1}^d k_i!)} | c_{\mathbf{k}}$; where:

- $\mathbf{k} = \langle k_1,k_2,\ldots,k_d\rangle$ represents the degrees of the variables $\mathbf{x} = \langle x_1,x_2,\ldots,x_d\rangle$ and $k_i < \mu_i$ for all $i = 1,\ldots,d$,
- $c_{\mathbf{k}}$ is an arbitrary integer,
- $\mathbf{Y_k}(\mathbf{x})$ is as defined above and
- $(2^m, \prod_{i=1}^d k_i!)$ is the greatest common divisor (GCD) of 2^m and $\prod_{i=1}^d k_i!$.

Alternately, $c_{\mathbf{k}} \equiv 0 \,\%\, \frac{2^m}{(2^m, \prod_{i=1}^d k_i!)}$ implies that $c_{\mathbf{k}} \cdot \mathbf{Y_k}(\mathbf{x}) \equiv 0$.

Example V.5: Consider the polynomial $F = 4x_1x_2^2 + 4x_1x_2$ corresponding to $f(x_1,x_2) : Z_2 \times Z_{2^2} \rightarrow Z_{2^3}$. We can use Lemma 5 to prove that f is a nil polyfunction. Here, $2^{n_1} = 2$, $2^{n_2} = 4$ and $2^m = 8$. Also, $\lambda = 4$; $\mu_1 = min\{2,4\} = 2$, $\mu_2 = min\{4,4\} = 4$.

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

because $c_{\langle 1,2\rangle} = 4 \equiv 0 \,\%\, \frac{8}{(8,1!\cdot 2!)}$.

B. Application to Equivalence Verification

The results of Sec. IV can be easily extended to multivariate polynomials as well. Given any polynomial $F(\mathbf{x})$ corresponding to the polyfunction $f : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots Z_{2^{n_d}} \rightarrow Z_{2^m}$, we can represent it as

$$F(\mathbf{x}) = \sum_{i=1}^{d} Q_i(\mathbf{x}) Y_{\mu(\mathbf{i})}(\mathbf{x}) + R(\mathbf{x}) \tag{19}$$

where,

- $\mu(\mathbf{i}) = \langle 0,...,\mu_i,...,0\rangle$ is a d-tuple, where μ_i is in position i and μ_i is defined according to Eqn.(18),
- $Q_i(\mathbf{x}) \in Z[x_1,\ldots,x_d]$ are arbitrary polynomials, possibly zero,
- $Y_{\mu(\mathbf{i})}(\mathbf{x})$ is the falling factorial of degree μ_i in variable x_i,
- $R(\mathbf{x})$ is an arbitrary polynomial in $\langle x_1,x_2,\ldots,x_d\rangle$, such that $degree\ (x_i) < \mu_i$, for all $i = 1,2,\ldots,d$.

From Lemma 4, $\sum_{i=1}^d Q_i(\mathbf{x}) Y_{\mu(\mathbf{i})}(\mathbf{x}) \equiv 0 \,\%\, 2^m$. This results in reducing the given polynomial to $R(\mathbf{x})$, where the degree k_i of each x_i is $< \mu_i$. This is illustrated in the example below.

Example V.6: Let $F(\mathbf{x}) = x_1^2 + 3x_2^2 + 4x_1x_2 + 7x_1$ over $Z_2 \times Z_{2^2} \rightarrow Z_{2^3}$. Here, $\lambda = 4$, and $\mu_1 =$

$min\{2,\lambda\} = 2$ and $\mu_2 = min\{4,\lambda\} = 4$. We represent the polynomial as,

$$F(x_1,x_2) = Y_{<2,0>}(x_1,x_2) + 3x_2^2 + 4x_1x_2 \qquad (20)$$

In this case, $Y_{<2,0>}(x_1,x_2)$ is a product of $\mu_1 = 2$ consecutive numbers in x_1 and vanishes $\% \ 2^3$. Hence, $F(\mathbf{x}) = R(\mathbf{x}) = 3x_2^2 + 4x_1x_2$, where the maximum degrees of x_1 and x_2 are respectively, $k_1 = 1 < \mu_1$ and $k_2 = 2 < \mu_2$.

We now state the following theorem:

Theorem 2: Let $F(\mathbf{x}) \in Z[x_1,\ldots,x_d]$ and let $f : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots Z_{2^{n_d}} \rightarrow Z_{2^m}$ be its associated polyfunction. To prove that $F(\mathbf{x})$ vanishes, it is sufficient to show that $F(\mathbf{x}) \equiv 0 \ \% \ 2^m$ for any $\prod_{i=1}^{d} \mu_i$ values of $\mathbf{x}$. Here μ_i is defined as the $min\{2^{n_i},\lambda\}$. Note that every such 'value' for $\mathbf{x} = \langle x_1,\ldots,x_d\rangle$ is a d-tuple, such that each x_i can take **any** μ_i **consecutive** values.

Proof: The proof is based on the corresponding procedure for univariate polynomials, which is extended and reproduced below.

Given any polynomial $F(\mathbf{x})$ corresponding to the polyfunction $f : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots Z_{2^{n_d}} \rightarrow Z_{2^m}$, we can reduce it according to Eqn. (19):

$$\begin{aligned} F(\mathbf{x}) &= \sum_{i=1}^{d} Q_i(\mathbf{x}) Y_{\mu(\mathbf{i})}(\mathbf{x}) + R(\mathbf{x}) \\ &= 0 + R(\mathbf{x}) \\ &= R(\mathbf{x}) \end{aligned} \qquad (21)$$

From Newton's formula in Proposition 2, we can now reinterpret the polynomial as:

$$\begin{aligned} F(\mathbf{x}) = R(\mathbf{x}) &= \sum_{\mathbf{k}<\mu} \frac{(\mathbf{\Delta^k} R)(\mathbf{0})}{\mathbf{k!}} Y_{\mathbf{k}}(\mathbf{x}) \\ &= \sum_{\mathbf{k}<\mu} c_{\mathbf{k}} Y_{\mathbf{k}}(\mathbf{x}) \end{aligned} \qquad (22)$$

We know that if all the $c_{\mathbf{k}}$ coefficients are $\equiv 0 \ \% \ \frac{2^m}{(2^m, \prod_{i=1}^{d} k_i!)}$, then the polynomial vanishes. Thus, we need to compute and compare the $c_{\mathbf{k}}$ values for the tuples $\mathbf{k} = \langle k_1, k_2, \ldots, k_d\rangle$, where each k_i can take any μ_i consecutive values. This requires evaluation of $F(\mathbf{x})$ for a maximum of $\prod_{i=1}^{d} \mu_i$ times.

Theorem 2 follows from the above results. ■

Example V.7: Consider the reduced polynomial from Example V.6. To determine if $F(x_1,x_2)$ vanishes, we need to compute $c_{\mathbf{k}}$ for $\prod_{i=1}^{2} \mu_i = 8$ values, where $0 \le k_1 < \mu_1 = 2$ and $0 \le k_2 < \mu_2 = 4$.

- $c_{\langle 0,0\rangle} = \frac{(\Delta^{\mathbf{k}} F)(0,0)}{0!\cdot 0!} \ \% \ \frac{2^3}{(2^3, 0!0!)} = 0$
- $c_{\langle 0,1\rangle} = \frac{(\Delta^{\mathbf{k}} F)(0,0)}{0!\cdot 1!} \ \% \ \frac{2^3}{(2^3, 0!1!)} = 3$

Since $c_{\langle 0,1\rangle} = 3 \ \% \ 2^3$, $F(x_1,x_2)$ is not a vanishing polynomial.

Corollary 2: Let $F_1(\mathbf{x})$ and $F_2(\mathbf{x})$ be two polynomials with coefficients in Z. Let $f_1, f_2 : Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots Z_{2^{n_d}} \rightarrow Z_{2^m}$ be their associated polyfunctions. To prove that $F_1(\mathbf{x})$ is equivalent to $F_2(\mathbf{x})$, it is sufficient to show that $F_1(\mathbf{x}) \ \% \ 2^m \equiv F_2(\mathbf{x}) \ \% \ 2^m$ for any $\prod_{i=1}^{d} \mu_i$ values of $\mathbf{x}$. Here μ_i is defined as the $min\{2^{n_i},\lambda\}$. Each x_i in the d-tuple $\langle x_1,\ldots,x_d\rangle$ can take **any** μ_i **consecutive** values.

Proof: It is required to show that:

$$\begin{aligned} F_1(\mathbf{x}) \ \% \ 2^m &\equiv F_2(\mathbf{x}) \ \% \ 2^m \\ \Rightarrow F_1(\mathbf{x}) - F_2(\mathbf{x}) &\equiv 0 \ \% \ 2^m \end{aligned} \qquad (23)$$

Since $\prod_{i=1}^{d} \mu_i$ values are sufficient to show that $F_1(\mathbf{x}) - F_2(\mathbf{x})$ vanishes, it follows that these values are sufficient to prove $F_1(\mathbf{x}) \ \% \ 2^m \equiv F_2(\mathbf{x}) \ \% \ 2^m$. ■

Example V.8: Let us now consider the polynomials $F(x_1,x_2) = x_1x_2^3 + 5x_1x_2^2 + 2x_1x_2$ and $F_2 = x_1^4x_2 + 2x_1^3x_2 + 3x_1^2x_2 + x_1x_2^3 + 5x_1x_2^2 + 4x_1x_2$ over $Z_2 \times Z_{2^2} \rightarrow Z_{2^3}$. Here, $\mu_1 = 2$ and $\mu_2 = 4$. We need to check if $F_1 \ \% \ 2^3 \equiv F_2 \ \% \ 2^3$. Therefore, we evaluate both polynomials for a maximum of $\mu_1 \cdot \mu_2 = 8$ tuples. Here, x_1 can take any $\mu_1 = 2$ consecutive values and x_2 can take any $\mu_2 = 4$ consecutive values. Evaluating for $x_1 = 0, 1$ and $x_2 = 0, 1, 2, 3$ for Z_{2^3}, we get

$$\begin{aligned} F_1(x_1 = 0) = 0 \ &; \ F_2(x_1 = 0) = 0 \\ F_1(x_1 = 1; x_2 = 0) = 0 \ &; \ F_2(x_1 = 1; x_2 = 0) = 0 \\ F_1(x_1 = 1; x_2 = 1) = 0 \ &; \ F_2(x_1 = 1; x_2 = 1) = 0 \\ F_1(x_1 = 1; x_2 = 2) = 0 \ &; \ F_2(x_1 = 1; x_2 = 2) = 0 \\ F_1(x_1 = 1; x_2 = 3) = 6 \ &; \ F_2(x_1 = 1; x_2 = 3) = 6 \end{aligned}$$

Since $F_1(\mathbf{x}) \equiv F_2(\mathbf{x})$ for all 8 tuples, the two polynomials are equivalent.

VI. Application to Verification

Using the results from Corollaries 1 and 2 from the previous sections, we have been able to perform simulations over a number of designs collected from a variety of benchmark suites. The results are presented in Tables I and II.

The first set of examples are datapaths with a single input bit-vector variable, which are modeled as univariate polynomials. The anti-aliasing function is from [4]. The

TABLE I
SIMULATION RESULTS FOR PROVING EQUIVALENCE

Benchmark	**Specs**	**Exhaustive Simulation**	**Our Approach**
	Var/Deg/$< n_1, \ldots, n_d >/m$	**Test Vectors**	**Simulation Bound/Time (s)**
Univariate			
Anti-alias function	1/6/$< 11 >$/16	2^{11}	$18/< 1$
Poly_unopt	1/4/$< 16 >$/16	2^{16}	$18/< 1$
$cos\ x$	1/6/$< 32 >$/32	2^{32}	$34/< 1$
$cot^{-1}x$	1/9/$< 32 >$/32	2^{32}	$34/< 1$
$erf\ x$	1/7/$< 32 >$/32	2^{32}	$34/< 1$
$ln(\frac{1+x}{1-x})$	1/7/$< 32 >$/32	2^{32}	$34/< 1$
Multivariate			
IRR	2/4/$< 12, 8 >$/16	2^{20}	$18^2/< 1$
PSK	2/4/$< 11, 14 >$/16	2^{25}	$18^2/< 1$
Cubic filter	3/3/$< 12, 14, 10 >$/16	2^{36}	$18^3/< 1$
Degree-4 filter 2	3/4/$< 15, 11, 13 >$/16	2^{39}	$18^3/< 1$
4^{th} Order IIR	2/4/$< 24, 29 >$/32	2^{53}	$34^2/< 1$
MIBENCH	2/9/$< 16, 12 >$/16	2^{28}	$18^2/< 1$
Horner Polynomial	3/4/$< 14, 14, 16 >$/16	2^{44}	$18^3/< 1$
Vanishing polynomial	2/10/$< 12, 12 >$/16	2^{24}	$18^2/< 1$

TABLE II
SIMULATION RESULTS FOR DETECTING NON-EQUIVALENCE

Benchmark	**Random Simulation**	**Our Approach**	**Our Approach**
	Test Vectors	**Simulation Bound**	**Required Test Vectors/Time(s)**
Anti-alias function	12	18	$18/< 1$
PSK	14	18^2	$308/< 1$
Horner Polynomial	4	18^3	$4335/< 1$
4^{th} Order IIR	9	34^2	$416/< 1$
Vanishing polynomial	8	18^2	$103/< 1$

second example is a polynomial expression from [34]. The other univariate examples are implementations of elementary function computations. The first benchmark in the set of multivariate datapath instances represents an image rejection computation (IRR). The phase-shift keying (PSK) function is from [4] and is used in digital communication. The polynomial filters are Volterra models of polynomial signal processing applications taken from [35]. MIBENCH is a 9^{th}-degree polynomial from a set of automotive applications in [36]. Horner polynomials are borrowed from [34]. Polynomial computations commonly used in DSP are often implemented in Horner's form using multiply-add-accumulate (MAC) units. In [4], it was shown how computations by these MAC units can be extracted as polynomials in Horner's form. The vanishing polynomial example was specifically created to validate our concepts.

Some of these designs were available as RTL code. The others were available as high-level specifications in MATLAB or C code. RTL code for these reference designs was automatically generated using the MATLAB Simulink and Filter Design toolboxes (particularly for the digital filter designs) [3]. Once the reference RTL descriptions were obtained, they were further optimized using techniques from [4] and [5]. In [4], application of high-level restructuring and symbolic algebra-based transformations was presented for high-level synthesis. These include factorization and expansion, tree-height reduction, etc. The recent work of [5] has derived a sequence of polynomial algebra based transformations to reduce the area-cost of the implementation. This is achieved by modulating and segmenting the coefficients and subsequently removing algebraic redundancy (vanishing polynomials). These transformations were applied to the original RTL description to obtain functionally equivalent implementations.

Subsequently, the data-flow graphs for the original and optimized RTL descriptions were extracted using GAUT [37]. 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. We then used the algorithm given in the Appendix to compute the appropriate λ and μ values. Based on these parameters, the proposed results were applied to generate the maximum number of required test vectors. The descriptions were then simulated with these vectors to verify equivalence. The results are given in Table I.

Let us explain our approach using the polynomial benchmark *Poly_unopt*. Fig. 3 depicts the dataflow graph for the original expression. We have used the optimization procedure presented in [5] to apply a series of algebraic reductions yielding an equivalent representation with the minimum estimated cost (in terms of area). The corresponding graph is depicted in Fig. 4. It is now required to show that the reduced cost implementation is the same as the original representation. Hence, we first compute the required number of simulation vectors. From Corollary 1, we know that $\lambda = SF(2^{16}) = 18$ consecutive test vectors are required. We then extract the polynomials corresponding to both designs and perform the simulation run. It was found that both polynomials evaluate to the same values $\% \, 2^{16}$, implying that the corresponding designs are equivalent.

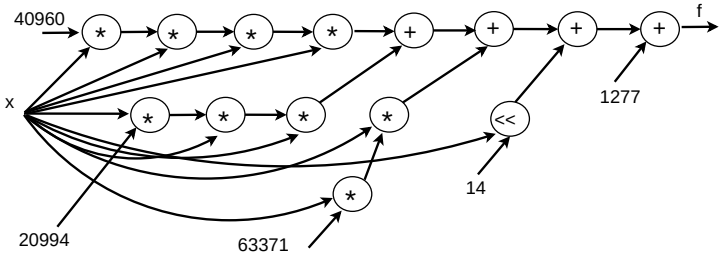

Fig. 3. Original expression of *Poly_unopt*

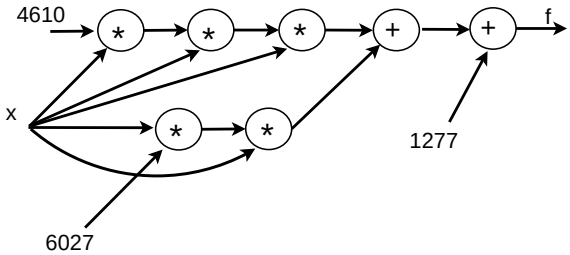

Fig. 4. Reduced expression of *Poly_unopt*

We also wanted to analyze the performance of our approach in the presence of bugs. To verify that we can detect non-equivalence of designs with the proposed simulation bound, we experimented with some designs by arbitrarily changing one or more of the coefficients. In all cases, we were able to detect the erroneous values within the required number of simulations. Table II presents the results of these experiments. Consider, for instance, the benchmark *PSK*. We were able to determine the error in 308 consecutive test vectors, well within the bound of 324 vectors.

VII. Conclusions

We have presented a method to determine simulation-bounds for equivalence verification of high-level descriptions of arithmetic datapaths. Our approach models the design as a polyfunction from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \ldots \times Z_{2^{n_d}} \rightarrow Z_{2^m}$. Established concepts from number theory and commutative algebra have been analyzed to derive an upper bound on the number of simulation vectors required for proving equivalence or identifying bugs. The set of all such vectors was also identified. Practical application of these results was demonstrated using several signal-processing designs. We were able to verify equivalence for a number of circuits by simulating the high-level descriptions. Also, the bugs in the design were detected within the proposed bound. As part of future work, we are investigating applications of the proposed results to various other datapath computations such as those that implement rounding and saturation arithmetic.

References

[1] 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.

[2] 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.

[3] MATLAB/Simulink; Matlab to RTL Translator, ", http://www.mathworks.com/products/simulink.

[4] 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.

[5] S. Gopalakrishnan, P. Kalla, and F. Enescu, "Optimizing Fixed-Size Bit-Vector Arithmetic using Finite-Ring Algebra", *in IWLS*, 2006.

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

[7] D. E. Knuth, *The Art of Computer Programming, Vol. II, Seminumerical Algorithms*, Addison Wesley, 1998.

[8] Crandall R. and Pomerance C., *Prime Numbers: a Computational Perspective*, Springer, 2000.

[9] 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.

[10] J. Smith and G. DeMicheli, "Polynomial methods for component maching and verification", *in In Proc. ICCAD*, 1998.

[11] P. Sanchez and S. Dey, "Simulation-Based System-Level Verification using Polynomials", *in High-Level Design Validation & Test Workshop, HLDVT*, 1999.

[12] I. Ugarte and P. Sanchez, "Formal Meaning of Coverage Metrics in Simulation-Based Hardware Design Verification", *in High-Level Design Validation & Test Workshop, HLDVT*, 2005.

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

[14] N. Shekhar, P. Kalla, and Enescu F., "Equivalence Verification Arithmetic Datapaths with Multiple Word-length Operands", *in Design Automation and Test in Europe (DATE)*, 2006.

[15] T. Raudvere, A. K. Singh, I. Sander, and A. Jantsch, "System level verification of digital signal processing applications based on the polynomial abstraction technique", *in Proc. of ICCAD*, pp. 285–290, 2005.

[16] J. E. Eaton, "The Fundamental Theorem of Algebra", *American Mathematical Monthly*, vol. 67, pp. 578–579, 1960.

[17] R. E. Bryant, "A Methodology for Hardware Logic Simulation", *Journal of the ACM*, vol. 38, pp. 299–328, 1991.

[18] R. E. Bryant, "Formal Verification of Memory Circuits by Switch-Level Simulation", *IEEE Transactions on CAD*, vol. 10, pp. 94–102, Jan, 1991.

[19] R. Ernst and J. Bhasker, "Simulation- Based Verification for High-Level Synthesis", *IEEE Design and Test*, vol. 8, pp. 14–20, Jan, 1991.

[20] D. Brand, "Exhaustive Simulation Need Not Require an Exponential Number of Tests", *in Proc. ICCAD*, pp. 98–101, Nov 1992.

[21] E. Clarke, S. German, Y. Lu, H. Veith, and D. Wang, "Executable Protocol Specification in ESL", *in Proc. FMCAD*, Nov 2000.

[22] J. Yuan, K. Shultz, C. Pixley, H. Miller, and A. Aziz, "Modeling design constraints and biasing in simulation using bdds", *in Proc. ICCAD*, Nov 1999.

[23] K. Shimizu and D. L. Dill, "Deriving a Simulation Input Generator and a Coverage Metric From a Formal Specification", *in Proc. DAC*, pp. 801–806, 2002.

[24] K. Shimizu and D. L. Dill, "Using Formal Specifications for Functional Validation of Hardware Designs", *IEEE Design & Test of Computers*, vol. 19, pp. 96–106, 2002.

[25] J. Moses, "Algebraic simplification: A guide for the perplexed", *Comm. ACM*, vol. 14, pp. 548–560, 1971.

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

[27] Z. Chen, "On polynomial functions from z_n to z_m", *Discrete Math.*, vol. 137, pp. 137–145, 1995.

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

[29] N. J. A. Sloane and S. Plouffe, *The Encyclopedia of Integer Sequences*, Academic Press, 1995.

[30] C. Jordan, *Calculus of Finite Differences*, New York: Chelsea, 1965.

[31] E. Lucas, "Question nr. 288", *Mathesis*, vol. 3, pp. 232, 1883.

[32] A. J. Kempner, "Miscellanea.", *Amer. Math. Monthly*, vol. 25, pp. 201–210, 1918.

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

[34] 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.

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

[36] 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.

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

VIII. Appendix

COMPUTE_SF(2^m)
2^m = Input for which λ needs to be computed
/* For $m = 1$ *or* $m = 2$ */
if $(m \leq 2)$ **then**
  return $(2 \cdot m)$
end if
/* For $m > 2$ */
$\nu = \lfloor log_2(1+m) \rfloor$;
$rem = m$;
$n = 0$;
while $(rem \neq 0)$ **do**
  $a_\nu = 2^\nu - 1$;
  $n = n + \lfloor rem/a_\nu \rfloor$
  $rem = rem \ \% \ a_\nu$
  $\nu = \nu - 1$
end while
return $(m + n)$

Algorithm 1: $SF(2^m)$ computation

The function $SF(n)$ was first considered by Lucas [31], though the algorithm for its computation was

outlined by Kempner [32]. It was revisited in [33] and it is defined as the smallest value λ for a given n at which $n|\lambda!$. We have adapted the algorithm from [32] for $n = 2^m$, and used the procedure to generate the simulation bounds for equivalence checking.

Example VIII.1: Let us compute the value of $SF(2^3)$. In this case, $m = 3$. The algorithm proceeds as follows:

1. Since $m > 2$, we compute the value of ν as $\lfloor log_2(1+3) \rfloor = 2$.
2. Now, initialize $rem = 3$ and $n = 0$. Continue to the *while* loop since $rem > 0$.
   - Compute $a_2 = 2^\nu - 1 = 3$ and $n = 0 + \lfloor rem/a_2 \rfloor = 1$
   - Update $rem = rem\%a_2 = 0$ and $\nu = \nu - 1 = 1$
3. Since $rem = 0$, exit the *while* loop. The computed value of λ is $(m+n) = 4$. From Sec. IV, we can verify that this is indeed the correct value.

Complexity: The worst case complexity of the algorithm is $O(m/log(m))$, where m corresponds to the word-length of the output variable in the datapath.