# Finding Linear Building-Blocks for RTL Synthesis of Polynomial Datapaths with Fixed-Size Bit-Vectors

**Sivaram Gopalakrishnan**[1], **Priyank Kalla**[1], **M. Brandon Meredith**[2] and **Florian Enescu**[2]
[1]Electrical & Computer Engineering, University of Utah, Salt Lake City, UT-84112
[2]Mathematics & Statistics, Georgia State University, Atlanta, GA-30303
{sgopalak, kalla}@ece.utah.edu, mmeredith5@student.gsu.edu, fenescu@mathstat.gsu.edu

**Abstract:** Polynomial computations over fixed-size bit-vectors are found in many practical datapath designs. For efficient RTL synthesis, it is important to identify good decompositions of the polynomial into smaller/simpler units. Symbolic computer algebra algorithms and tools have been used for this purpose. However, fixed-size ($m$) bit-vector arithmetic is polynomial algebra over the finite integer ring $Z_{2^m}$, which is a *non-unique factorization domain* (non-UFD). While non-UFDs provide an extra freedom to search for decompositions, they complicate polynomial manipulation as traditional division-based algorithms are inapplicable.

This paper presents new mathematical concepts for polynomial decomposition over $Z_{2^m}$, for RTL synthesis over fixed-size $m$-bit vectors. Given a polynomial, we identify a specific set of linear expressions and compute the Gröbner bases of their ideal (over non-UFD $Z_{2^m}$) using syzygies. This basis serves as *good building-blocks* for the given computation. A decomposition is identified by subsequent Gröbner basis reduction. Experimental results demonstrate significant area savings due to our approach, as compared against contemporary datapath synthesis techniques.

## I. Introduction

RTL descriptions of integer datapaths that implement polynomial arithmetic are found in many practical applications, such as in Digital Signal Processing (DSP) for audio, video and multimedia applications [1] [2]. Arithmetic datapath intensive designs implement a sequence of ADD, MULT type of algebraic computations over bit-vectors; hence they are generally modeled at RTL or behavioral-level as *multivariate polynomials of finite degree* [2] [3]. Due to a large number of ADD, MULT operations in such designs, designers often employ ingenious strategies to control the datapath size. In many cases, the design choice is that of a *single, uniform system word-length* for the computations [4]. Such fixed-size datapath computations are generally implemented using: signal truncation, rounding or saturation arithmetic [5].

In this paper, we focus on datapath descriptions that implement polynomial computations over fixed-size bit-vectors by way of signal truncation. In such designs, $m$-bit adders and multipliers produce an $m$-bit output; only the lower $m$-bits of the outputs are used and the higher-order bits are ignored. Usually, such computations require appropriate scaling of coefficients and/or signals such that "overflow" can be avoided/ignored and standard fixed-point arithmetic can be implemented.

This paper addresses the problem of area optimization of polynomial datapaths implemented with fixed-size bit vectors. Fixed-size ($m$) bit-vectors represent integer values reduced modulo $2^m$ (mod $2^m$). Therefore arithmetic implemented with fixed-size ($m$) bit-vectors manifests itself as polynomial algebra over finite integer rings of residue classes $Z_{2^m}$. We exploit the number-theoretic properties of finite integer rings and develop algorithmic algebraic techniques to search for polynomial decompositions. We present new mathematical techniques that allow for such decompositions leading to an area-efficient hardware implementation.

### A. Motivation

Consider a polynomial $h = x^2 + 6 * x$. If $h$ is computed over $Z$ (integral domain), then $f$ can be uniquely factorized as $h = (x)*(x+6)$ because $Z$ is a unique factorization domain (UFD). However, if $h$ is a polynomial in $Z_{2^3}$ (computed over 3-bit vectors), it can be factorized as $h = (x) * (x + 6)$ or $h = (x+4)*(x+2)$ because $Z_{2^3}$ is a non-unique factorization domain (non-UFD). This clearly suggests that, a non-UFD ($Z_{2^m}$) offers the potential to search for more decompositions.

To demonstrate this potential further, let us consider the implementation of Savitzky-Golay (SG) filters. These filters are widely used in image processing applications. Typically, an image is represented in an $n \times n$ matrix, that consists of $n^2$ pixels. Each pixel represents a polynomial computation in two dimensions: $x$ and $y$. Given the size of the image matrix ($n$), and the order of computation, we can generate two-dimensional polynomial representations for each pixel in the matrix [6]. A third order polynomial $f$ representing a pixel in a $6 \times 6$ matrix is given in Eqn. 1. The polynomial $f$ is computed over a fixed-size 16-bit datapath ($Z_{2^{16}}$).

$$\begin{aligned} f = \quad & 7x^3 - 984y^3 - 76x^2y + 92xy^2 + 7x^2 \\ & -39xy - 46y^2 + 7x - 46y - 75 \end{aligned} \tag{1}$$

When synthesized using the Synopsys Design Compiler, $f$ occupies an area of 37506 sq. units.

Now, let us apply conventional polynomial decomposition techniques on $f$. When factorization and common subexpression elimination transforms [7] [8] [9] are applied to $f$, the representation of $f$ is as follows

$$\begin{aligned} & C = 7x - 46y; \\ f = \quad & 7x^3 - 984y^3 - 76x^2y + 92xy^2 + \\ & C(x + y + 1) - 75 \end{aligned} \tag{2}$$

The area occupied by this representation is 32846 sq. units.

On applying the horner form decomposition on $f$, we get the following representation (Eqn. 3).

$$\begin{aligned} f = \quad & -75 + (74 + (-46 - 984y)y)y + (74 \\ & +(-39 + 92y)y + (7 - 76y + 7x)x)x \end{aligned} \tag{3}$$

S. Gopalakrishnan and P. Kalla were supported by an NSF CAREER grant, award #CCF-0546859; M. Brandon Meredith and F. Enescu were supported by an NSF grant, award #CCF-0515010.

The implementation of this representation yields an area of 53840 sq. units.

Now, consider the following decomposition of $f$ in terms of a linear expression $g$ in Eqn. 4.

$$\begin{aligned} g &= (x + 18718y); \\ f &= g(g(7x + 16y + 7) + 53y \\ &\quad +7) - 75 \end{aligned} \tag{4}$$

The implementation area of this representation is 28407 sq. units. The linear expression $g$ aids in performing the polynomial decomposition leading to a cost-effective area implementation. In other words, $g$ serves as a good building block for $f$.

- How do we identify/generate such linear expressions that serve as good building blocks?
- More importantly, how do we perform polynomial decomposition using these linear expressions, resulting in an optimized implementation?

The answers to the above questions are the subject of this paper.

### B. Problem Modeling, Approach and Contributions

We model arithmetic datapaths over fixed-size bit-vectors as *polynomial functions over finite rings* of residue classes $Z_{2^m}$. Given a polynomial function, we derive a specific set of linear expressions, which we refer to as the reduced linear form (RLF) set. Using the RLF set, we derive the Gröbner basis of their ideal over non-UFD $Z_{2^m}$ [10]. The basis serves as good building-blocks. Note that Gröbner basis computation using the well-known Buchberger's algorithm is applicable over the fields of reals $R$, fractions $Q$, complex numbers $C$, etc, but not over non-UFDs. On the other hand, Möller proposed an algorithm [10] that can compute Gröbner bases of ideals over arbitrary commutative rings. A decomposition is identified by subsequent Gröbner basis reduction. We show how this decomposition leads to an area-efficient hardware implementation.

## II. Previous work and Limitations

Contemporary high-level synthesis tools are quite adept in extracting control/data-flow graphs (CDFGs) from the given RTL descriptions, and also in performing scheduling, resource-sharing, retiming, and control synthesis. However, they are limited in their capability to employ sophisticated manipulations to reduce the cost of the implementation. For this reason, there has been increasing interest in exploring the use of algebraic manipulation for RTL synthesis of arithmetic datapaths. The works of [11] [12] derive new polynomial models of complex computational blocks for efficient synthesis. In [2], Symbolic Computer Algebra tools are used to search for a decomposition of a given polynomial according to available components in a design library, using a Buchberger-variant algorithm [13] [14] [15] for Gröbner bases. However, the derived polynomial models represent the computations over fields $(R, Q)$, or over the integral domain $(Z)$ - (UFDs). This often results in a polynomial approximation [3], without properly accounting for the effect of bit-vector size on the resulting computation. Moreover, the approach of [2] is restrictive inasmuch as it can only search for an implementation based on the given library elements. While this works for elementary and trignometric functions such as $log(x)$, $sin(x)$, $cos(x)$, etc., it is quite inefficient for arbitrary polynomials. Other algebraic transforms have also been explored for efficient hardware synthesis: factorization and common sub-expression elimination [7] [16], exploiting the structure of arithmetic circuits [17], term re-writing [18], etc. However, these techniques overlook the effect of bit-vector size on the given computation. Gröbner bases has been used for factorization in [19], but for logic-level synthesis of efficient arithmetic-circuit architectures. But in contrast, we use the Gröbner bases computations for decomposition of polynomial datapaths at RTL.

Finite rings of the type $Z_{2^m}$ are non-UFDs. Fundamental computer algebra results on Euclidean division and factorization cannot be applied over non-UFDs. As a result, contemporary (algebra-based) high-level synthesis frameworks are ineffective in this domain. However, in this paper we show how we can use the non-UFD $Z_{2^m}$ as a resource to offer further potential for optimization.

In the area of symbolic computer algebra, a large number of polynomial manipulation engines are available. However, they generally operate on UFDs. The recent computer algebra handbook is a good source of information on the manipulation capabilities of these tools [20].

Modulo arithmetic has been applied to the task of circuit/RTL verification [21]. The concept of polynomial functions over finite rings has also been applied to the equivalence verification and optimization of arithmetic datapaths in [22] [23]. This paper demonstrates its application to *polynomial decomposition* of arithmetic datapaths.

## III. Preliminaries

This section briefly reviews commutative and computational algebra concepts to put our polynomial decomposition problem in perspective. The material is mostly referred from [24].

*Definition III.1:* A **ring** is a set $R$ with two binary operations $'+'$ and $'\cdot'$ (addition and multiplication) satisfying additive and multiplicative associativity, additive commutativity, left and right distributivity, and existence of additive identity and inverse. A *commutative ring* also satisfies multiplicative commutativity.

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, $n = 2^m$.

$$(a + b)\%n = (a\%n + b\%n)\%n \tag{5}$$
$$(a \cdot b)\%n = (a\%n \cdot b\%n)\%n \tag{6}$$
$$(-a)\%n = (n - a\%n)\%n \tag{7}$$

*Definition III.2:* Let $R$ be a ring. A **polynomial** over $R$ in the indeterminate $x$ is an expression of the form:

$$a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0 = \sum_k a_i x^i \tag{8}$$

$\forall a_i \in R$. 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. When $R = Z_{2^m}$, the polynomials are evaluated $\%2^m$.

Note that since the element 1 always has an inverse (1 is the inverse of itself), division by monic polynomials can always be carried out in arbitrary rings. This concept will play an important role in this paper.

### A. Ideals and Ideal Membership

*Definition III.3:* Let $I$ be a subset of the ring $R$. Then $I$ is called an **ideal** of $R$ if

- $0 \in I$
- $I$ is closed under addition; $x, y \in I \Rightarrow x + y \in I$
- If $x \in R$ and $y \in I$, then $x \cdot y \in I$ as well as $y \cdot x \in I$.

*Definition III.4:* Let $f_1, f_2, \ldots, f_s$ be the given elements of the commutative ring $R$. Let $I$ be an ideal in $R$. If:

$$I = \{g_1 f_1 + g_2 f_2 + \ldots + g_s f_s : g_1, \ldots, g_s \in R\} \qquad (9)$$

then, $f_1, \ldots, f_s$ are called the **generators (or basis)** of the ideal $I$ and we denote this as $I = \langle f_1, f_2, \ldots, f_s \rangle$.

Every ideal in finite integer rings of the form $Z_n[x_1, \ldots x_d]$ , $n \in N$, has a finite set of generators. Moreover, a given ideal may have many different basis. However, it is possible to transform any given ideal basis into an especially useful type of basis, called a *Gröbner basis*. Buchberger [13] devised an algorithm to transform any ideal basis into a unique, minimal, canonical basis, which he called the Gröbner basis. This technique has since been known as the Buchberger's algorithm, which has now become "text-book knowledge" in computer algebra [25] [26]; and it has been applied in CAD applications in synthesis [2] [19] and SAT/verification [27] [28] [29].

Gröbner bases allow to test for membership in an ideal. For example, given an ideal $I = \langle f_1, \ldots, f_s \rangle$ and a polynomial $f$, we can determine whether or not $f \in I$ using Buchberger's algorithm, and also find a suitable decomposition of $f$ in terms of the ideal members. This concept relates to our work as follows: Given $f$, a polynomial for RTL synthesis, and given $\langle f_1, \ldots, f_s \rangle$ (the "good building blocks" for the datapath), find a decomposition of $f$ in terms of the $f_i$'s.

**Buchberger's Algorithm and non-UFDs:** It is important to note that Buchberger's algorithm is applicable over the fields of reals $R$, fractions $Q$, complex numbers $C$, Galois fields $GF(p^m), p =$ prime, etc.; i.e. over *unique factorization domains*, as it relies upon polynomial division. Over non-UFDs of the type $Z_{2^m}$, division is not well defined and hence the Buchberger's algorithm cannot be applied.

In [30], Möller proposed a new technique to compute Gröbner bases using *syzygies*[1]. This approach can be applied to compute Gröbner bases of ideals over arbitrary commutative rings. In fact, the textbook [10] has devoted an entire chapter (chapter 4) on computing Gröbner bases over rings using Möller's technique. In this project, we have used Möller's algorithm to compute Gröbner bases of a given ideal in $Z_{2^m}[x_1, \ldots, x_d]$. Note that the theory and procedure for: i) computing the Gröbner basis; and ii) reducing $f$ w.r.t. the Gröbner basis ($f \xrightarrow{G}_+ r$) is available in [10]. Due to lack of space we will not describe these procedures here.

In subsequent sections, we show the overall procedure to decompose a polynomial into linear building blocks. But first, let us introduce the problem of "polynomial simplification" which sheds the light on how to identify the "good linear building-blocks" from a given polynomial for synthesis.

## IV. The Polynomial Simplification Problem

Given a polynomial function $f$ in $d$ variables over $Z_{2^m}$, can one find a *linear change of variables* such that under this transformation $f$ is polynomial in $n$ variables where $n < d$? Let us illustrate this with two examples:

*Example IV.1:* Over $\mathbf{Z_8}$, let $f = x^2 + x + 4y + 1$. One can check that $f = u^2 + u + 1$ with $u = x + 4y$.

*Example IV.2:* Over $\mathbf{Z_{16}}$, let $f = 8xy^2 + 11y^3 + 9y^2 + 1$. This is $f = u^3 + u^2 + 1$ with $u = 8x + 3y$.

We will call the polynomials that can be reduced via linear transformations, *simplifiable polynomials*. Intuitively, if a polynomial is simplifiable, then the simplified form may provide impressive area savings when synthesized. This also suggests that the terms $u$ in the above examples are *potentially good building-blocks* for $f$. Recent work on simplifiable polynomials [31] obtained results that describe whether or not a polynomial is simplifiable, how many essential number of variables a polynomial has, and how to obtain its simplified form. However, this work is applicable over fields ($R, Q, C$, etc.) and cannot be directly ported over $Z_{2^m}$ (non-UFD).

We investigated this problem of simplifiable polynomials over finite integer rings. *The idea behind our approach is to describe the relationship between the simplifiable polynomial and the linear transformations that can be used to simplify it.* The fact is that there could be more than one way to simplify a given polynomial. Our approach is to show that there is a strong relationship between such linear transformations and the degree-one form of the polynomial $f$. This brings us to the following definition.

*Definition IV.1:* Let $f$ be an element of $Z_{2^m}[x_1, \ldots, x_d]$. Denote by $u$ its degree-one form (i.e. its linear part). The set $\{l : l \text{ monic in at least one of the } x_i; l \mid u\}$ is called the set of reduced linear forms of $f$. It is denoted by $RLF(u)$.

*Example IV.3:* Let $f \in Z_4[x, y]$, $f = y^2 + 2x$. Then

$$RLF(u) = \{x, x + 2y\}.$$

In the above example, the linear part of $f$ is $2x$; $x$ divides $2x$, and $(x+2y)$ also divides $2x$ modulo 4. Note that RLFs are defined to be monic in some variable so that division can be performed in $Z_{2^m}$. The following examples demonstrate how the RLFs can be used to test whether or not a polynomial is simplifiable.

*Example IV.4:* Let $f = 3x^2 + 4xy + 6xz + 4yz + 4y^2 + 4z^2 + x + 2y + 4z$ over $\mathbf{Z_8}$. We have that $RLF(f) = \{x + 2y + 4z\}$. Divide $f$ by $u = x + 2y + 4z$. We get $f = u(3x - 2y + 2z + 1) + 4z^2$. Divide the quotient $3x - 2y + 2z + 1$ further to $u$ and write $f$ as $f = u(3u - 2z + 1) + 4z^2 = 3u^2 - 2uz + u + 4z^2 = 3u^2 - uv + u + v^2$, after denoting $v = 2z$.

Although the approach looks promising, the specific problem formulation (of strictly reducing the number of variables) is too restrictive. In fact, in our preliminary experiments, we did not find any practical polynomial datapath designs to be

[1] To quote [26], "In astronomy, a syzygy indicates an alignment of planets; i.e. planets are yoked together. In a mathematical syzygy, it is polynomials that are yoked".

simplifiable. Moreover, we observed in many cases that there were more than one RLFs corresponding to a polynomial; and it is not possible to predict *a priori* their potential to simplify $f$, as shown below.

*Example IV.5:* Let $f = 2x + 2y + y^2 + xy + 2yz$ over $\mathbf{Z_4}$. Then $RLF(f) = \{x+y, x+y+2z\}$. If one chooses to divide $f$ by $u = x+y$, then $f = u(2+y)+yz$ and $f$ is not simplified. However, if we divide $f$ by $v = x+y+2z$ we get $f = v(2+y)$ and $f$ is simplifiable.

Despite the fact that $u = x + y$ does not simplify $f$ above, it still provides a decomposition $f = u(2+y)+yz$. Moreover, if there are more than one RLFs, we should be able to explore whether $f$ can be composed of any combination of these RLFs. These observations made us realize the *Gröbner basis flavor* of the problem: "Can $f$ be composed of the RLF set?" Formulating the problem in an algorithmic algebra framework: "Is $f$ a member of the ideal spanned by the reduced linear forms?" To answer this question, we can compute the Gröbner basis of the RLF set and *reduce* $f$ according to this Gröbner basis!

Applying this concept to the previous example, $f = 2x + 2y+y^2+xy+2yz \in Z_4$, and given RLF-set $= \{x+y, x+y+2z\}$, the Gröbner basis for the RLF-set is $\{x + y, 2z\}$. And this gives the decomposition $f = (x+y)(2+y)+(2z)(y)$, and hence $f$ is a member of the ideal spanned by the given RLF-set.

Now we have to devise an efficient way to compute the set of all reduced linear forms. In what follows, we show an Integer Linear Program (ILP) formulation to construct the RLF set.

### A. Generating RLF-set using ILP

For didactic purposes, let us consider a 3-variable polynomial $f(x, y, z) \in Z_{2^m}[x, y, z]$; the approach works for any number of variables. Let $LP(f) = a_1x + b_1y + c_1z$, be the linear part of $f$. According to our definitions, the RLF is monic in at least one variable and it divides LP(f). Let $h = x + iy + jz$ be the RLF monic in $x$, where $i, j$ are $\in Z_{2^m}$. We have to identify suitable $i, j$ such that,

$$Rem(LP(f), h = x + iy + jz) = 0\%2^m \tag{10}$$

where Rem represents the remainder of the division of $LP(f)$ by $h$. Note that we can compute $a_1 \cdot h - LP(f)$ to eliminate the $x$ term, and that results in $(a_1i-b_1)y+(a_1j-c_1)z$. Since $Rem(LP(f), x + iy + jz) = 0\%2^m$, then:

$$(a_1i - b_1) \equiv 0\%2^m \tag{11}$$

$$(a_1j - c_1) \equiv 0\%2^m \tag{12}$$

Similarly constraints can be derived for RLFs monic in the remaining variables $(y, z)$. Now we already know the values of $a_1, b_1, c_1, m$ and need to to find $i, j$ that satisfy the above constraints. The above linear congruences can be trivially represented as ILP constraints: $(a_1i - b_1) \equiv 0\%2^m$ implies that

$$i >= 0; \tag{13}$$

$$i < 2^m \tag{14}$$

$$a_1i - b_1 - 2^m k = 0; \tag{15}$$

$$integer \quad i, k \tag{16}$$

where $k$ is an arbitrary integer. Once the value of $i = i_0$ is found, we can further constrain the ILP as $i \neq i_0$ to find the coefficients for other RLFs. The search terminates when any ILP instance becomes infeasible. In this fashion, iterative ILP runs can generate the set of all RLFs.

## V. Overall Approach

Our overall polynomial decomposition approach is as follows. We take the given polynomial $f$ and the operating bit-vector size $(m)$, and generate the complete RLF-set. This is accomplished by using repeated calls to the ILP solver (as shown above). Then, using Möller's algorithm, we generate a Gröbner basis $G$ of this RLF-set in $Z_{2^m}[x_1, \ldots, x_d]$. Elements of this Gröbner basis $G$ serve as "good building blocks" for the polynomial $f$. Now we have to derive a decomposition for $f$ in terms of the computed Gröbner basis. This decomposition can be identified by *testing whether $f$ is a member of the ideal spanned by the Gröbner basis of the RLF-set.* To perform this ideal membership test we *reduce* the given polynomial $f$ w.r.t. $G$ (denoted $f \xrightarrow{G}_+ r$).

Let the computed Gröbner basis $G = \{g_1, \ldots, g_t\}$. Then the reduction results in:

$$f = h_1 \cdot g_1 + \cdots + h_t \cdot g_t + r \tag{17}$$

Note that if $f$ is a member of the ideal spanned by the RLF-set, then $r = 0$; otherwise, $r \neq 0$ is a *minimal* expression that cannot be reduced any further (w.r.t. $G$). Moreover, the reduction process automatically identifies $(h_1, \ldots, h_t)$. Thus we have a decomposition for $f$ in terms of $h_i$'s, $g_i$'s $(1 \leq i \leq t)$ and $r$. The datapath is then synthesized according to Eqn. 17.

At this point, one might question the motivation for using the Gröbner basis of the RLF-set as the "good building blocks", as opposed to using the RLF-set itself for the decomposition of $f$. It has been generally observed that the size of the Gröbner basis (number of elements) is no larger than that of the RLF-set. This was true in all our experiments too. Moreover, if $f$ is a member of the ideal spanned by the RLF-set then $r = 0$ and the decomposition is likely to be favorable. We demonstrate this with an example:

*Example V.1:* Consider $f = 2x + 2y + y^2 + xy + 2yz$ over $Z_4$. The RLF-set for $f$ is found to be $RLF(f) = \{f_1 = x + y + 2z;\ f_2 = x + 3y + 2z;\ f_3 = 3x + y + 2z;\ f_4 = x + y;\ f_5 = x + 3y;\ f_6 = 3x + y\}$. Using the reduction procedure from [10] we find that $f$ can be decomposed in terms of $f = (h_1 \cdot f_1+h_2 \cdot f_2+h_3 \cdot f_3+h_4 \cdot f_4+h_5 \cdot f_5+h_6 \cdot f_6)+r$ where $h_1 = 1;\ h_2 = y + 1;\ h_3 = y + 1;\ h_4 = y + 1;\ h_5 = y + 1;\ h_6 = y + 1$ and $r = 2yz + 2z$ (all computations done %4).

Computation of the Gröbner basis of the RLF-set results in $G = \{g_1 = 2z;\ g_2 = x+y+2z;\ g_3 = 3x+y;\ g_4 = 2y\}$; i.e. *only 4 elements as compared to 6* in the previous case. Moreover, reducing $f$ w.r.t. $G$ results in $f = h_1 \cdot g_1+h_2 \cdot g_2+h_3 \cdot g_3+h_4 \cdot g_4$. In this case, $h_1 = 1;\ h_2 = 3; h_3 = y + 1; h_4 = x + z + 1$. Moreover, here $r = 0$ as $f$ is a member of the ideal spanned by the RLF-set. Clearly, the decomposition in terms of the Gröbner basis is more favorable than using the RLF-set itself.

## VI. Experiments

The polynomial representing the datapath computation and the operating bit-vector size $(m)$ were given as the inputs. ILOG-CPLEX [32] was used as the integer linear program solver to generate the RLF set. The Gröbner bases computation and the polynomial reduction algorithm (using the

TABLE I

COMPARISON OF PROPOSED METHOD WITH OTHER DECOMPOSITIONS

| Benchmarks | Var/Deg/m | Horner Decomposition | | Factorization/CSE | | Proposed method | | Improvement | |
|---|---|---|---|---|---|---|---|---|---|
| | | Area | Delay | Area | Delay | Area | Delay | Area % | Delay % |
| MVCS | 2/3/16 | 39699 | 135.4 | 31040 | 119.1 | 22214 | 157.8 | 28.4 | -32 |
| SG 1 | 2/3/16 | 53840 | 131.1 | 32846 | 154.6 | 28407 | 179.7 | 13.6 | -37.2 |
| SG 2 | 2/3/16 | 44451 | 139.8 | 36659 | 121.9 | 31259 | 177.6 | 14.7 | -45 |
| SG 3 | 2/2/16 | 19744 | 107.4 | 16604 | 90.1 | 11311 | 130.2 | 31.8 | -44 |
| Quad 1 | 2/2/16 | 26120 | 115.6 | 19519 | 105.3 | 16480 | 142.4 | 15.6 | -35.2 |
| Quad 2 | 2/2/16 | 21853 | 112.9 | 16886 | 118.4 | 14076 | 129.7 | 16.7 | -14.8 |
| Mibench 1 | 3/2/8 | 9246 | 60.9 | 10808 | 63.8 | 5151 | 67.2 | 44.3 | -10 |
| Mibench 2 | 3/2/8 | 8085 | 61.6 | 9551 | 64.8 | 3282 | 68.1 | 59.4 | -10.5 |

Gröbner bases) were implemented in Maple [33]. For horner form decomposition and factorization, we used the routines available in MATLAB. For common sub-expression elimination, we used the JuanCSE tool available at [9].

The polynomial has to be synthesized with fixed-size ($m$) adders and multipliers. The version of DesignWare Library available to us does not have these units as pre-designed library modules. Hence we used the Synopsys Design Compiler to generate these designs. These fixed-size $m$-bit ADD, MULT units were used, subsequently, to implement the decomposed polynomial.

The experiments were performed on a variety of DSP benchmarks and the results are presented in Table I. Column 1 lists the benchmarks used for the experiments. The first benchmark is a multi-variate cosine wavelet used in a graphics application from [8]. The next three benchmarks are Savitzky-Golay filters used in image processing applications, generated from MATLAB [6]. Quad[1-2] are quadratic filters from [1]. The last two benchmarks from [34] are used in automotive applications. Column 2 lists the design characteristics: number of variables (bit-vectors), their highest degree and the bit-vector size ($m$). Columns 3 and 4 list the implementation area and delay of the polynomial, implemented using the Horner form decomposition, respectively. Columns 5 and 6 list the implementation area and delay of the polynomial, implemented using Factorization + common sub-expression elimination, respectively. Columns 7 and 8 list the implementation area and delay of the polynomial, implemented using our method, respectively. The final column lists the improvement in the implementation area using our polynomial decomposition technique. The improvement is measured against the better decomposition (Horner Vs Factorization+CSE). In other words, for the first six benchmarks, we report the improvement w.r.t. the implementation area of the Factorization+CSE method. For the remaining two benchmarks, we report the improvement w.r.t. the implementation area of the Horner decomposition method. Considering all the benchmarks, the average improvement in the actual implementation area was approximately **28%**. It was also observed that there was an average increase of approximately 28.5% in delay.

**Example:** Let us consider the Mibench 2 filter example to illustrate the decompositions. The polynomial computation $f$ implemented over a fixed-size 8-bit datapath is given by

$$f = x^2 + 2xy + 6xz + y^2 + 6yz + 9z^2 + 43x + 43y + z; \quad (18)$$

Horner form decomposition of $f$ results in $f_{horner}$.

$$f_{horner} = (1+9z)z+(6z+43+y)y+(6z+2y+43+x)x; \quad (19)$$

While performing factorization, we initially split the polynomial $f$ in terms of its degree as follows

$$\begin{aligned} f_0 = 0 \quad ; f_1 = \quad & 43x + 43y + z; \\ f_2 \quad = \quad & x^2 + 2xy + 6xz + y^2 + 6yz + 9z^2; \\ f \quad = \quad & f_0 + f_1 + f_2; \end{aligned} \quad (20)$$

$f_2$ can be factored as $(x + y + 3z)(x + y + 3z)$. $f$ can then be represented as $f_{factor}$ given by

$$\begin{aligned} C \quad &= \quad (x + y + 3z)(x + y + 3z); \\ f_{factor} \quad &= \quad C + 43x + 43y + z; \end{aligned} \quad (21)$$

We determine the RLF set and compute the Gröbner bases of their ideal to be $(x + y + 131z)$. When our decomposition is applied, $f$ is represented as $f_{decomp}$ given by

$$\begin{aligned} A \quad &= \quad (x + y + 131z); \\ f_{decomp} \quad &= \quad A(A + 43); \end{aligned} \quad (22)$$

The polynomials $f_{horner}$, $f_{factor}$ and $f_{decomp}$ were written in Verilog as shown in the previous equations, and implemented using the Synopsys Design Compiler. The polynomial $f_{decomp}$ occupied a lesser area (3282 sq. units) when compared to the area of $f_{factor}$ (9551 sq. units) and $f_{horner}$ (8085 sq. units).

Many high-level synthesis techniques such as scheduling, resource sharing, retiming, etc., can be employed to further optimize the given implementation. Typically, given a polynomial computation, we can perform the decomposition and then apply these techniques. Our decomposition can be used as a pre-processing step to these techniques, thus providing an additional scope for optimization.

## VII. LIMITATIONS AND FUTURE WORK

While the proposed decomposition technique looks promising, it has certain limitations. The fixed-size datapath problem is somewhat restrictive. Often, datapaths contain multiple word-length operands. For example, the computation performed by a digital image rejection/separation unit takes two input signals: a 12-bit vector and an 8-bit vector, resulting in an output signal represented by a 16-bit vector. It is therefore important to also account for the input bit-vector sizes while

performing the decomposition. We are currently investigating the extension of the proposed decomposition technique to datapaths implemented with multiple word-length operands.

- Consider the example of a 5th degree Chebyshev polynomial ($P_5$) used in computer graphics applications.

$$P_5 = 16x^5 - 20x^3 + 5x \tag{23}$$

Since this is a univariate polynomial in $x$, the RLF generated for this expression has only one linear term: $x$. When our decomposition is applied, it simply reduces to that of a Horner form decomposition.

- Consider an example for an implementation of a Quartic Spline polynomial ($q$).

$$q = zu^4 + 4avu^3 + 6bu^2v^2 + 4uv^3w + qv^4 \tag{24}$$

In this polynomial implementation, there are no linear terms. For such cases, our algorithm cannot produce an RLF set, thus resulting in no decomposition. We can either "guess" a good linear expression and proceed with the decomposition, or create linear expressions using other decompositions and then, subsequently apply our technique.

The above issues are currently under investigation.

## VIII. Conclusions

This paper has presented a polynomial decomposition technique for arithmetic datapaths implemented with fixed-size ($m$) bit-vectors. Fixed-size bit-vector arithmetic is modeled as algebra over the finite integer ring $Z_{2^m}$. For a given polynomial computation, we generate a specific set of linear expressions using an integer linear program solver. For the generated set of linear expressions we compute the Gröbner bases of their ideal (over non-UFD $Z_{2^m}$) using syzygies. A Gröbner bases reduction algorithm is then applied to derive an efficient implementation of the given polynomial. Experimental results demonstrate significant area savings using our approach as compared against contemporary datapath synthesis techniques.

## Acknowledgment

The authors would like to thank Prof. David Cox of Amherst College for informing us about the Gröbner bases computation over a non-UFD $Z_{2^m}$ using the Möller's algorithm [10]. We would also like to acknowledge the help of Namrata Shekhar from the University of Utah for implementing the Möller's algorithm.

## References

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

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

[3] J. Smith and G. DeMicheli, "Polynomial circuit models for component matching in high-level synthesis", *IEEE Trans. VLSI*, vol. 9, 2001.

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

[5] G. Constantinides, P. Cheung, and W. Luk, "Synthesis of saturation arithmetic architectures", *ACM Trans. Des. Auto*, pp. 334–354, July 2003.

[6] J. Krumm, "Savitzky-Golay Filters for 2D Images", Available at http://homepages.inf.ed.ac.uk/rbf/ CVonline/LOCAL_COPIES/KRUMM1/SavGol.htm.

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

[8] A. Hosangadi, F. Fallah, and R. Kastner, "Optimizing polynomial expressions by algebraic factorization and common subexpression elimination", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 2012–2022, Oct 2006.

[9] JuanCSE, "Extensible, programmable and reconfigurable embedded systems group", Available at http://express.ece.ucsb.edu/suif/cse.html.

[10] W. Adams and P. Loustaunau, *An Introduction to Gröbner Bases*, American Mathematical Society, 1994.

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

[12] J. Smith and G. DeMicheli, "Polynomial methods for allocating complex components", *in Proc. DATE*, 1999.

[13] 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.

[14] B. Buchberger, "A Theoretical Basis for Reduction of Polynomials to Canonical Forms", *ACM SIG-SAM Bulletin*, vol. 10, pp. 19–29, 1976.

[15] B. Buchberger, "Some Properties of Grobner Bases for Polynomial Ideals", *ACM SIG-SAM Bulletin*, 1976.

[16] A. Hosangadi, F. Fallah, and R. Kastner, "Energy Efficient Hardware Synthesis of Polynomial Expressions", *in Int'l. Conf. on VLSI Design*, pp. pp. 653–658, 2005.

[17] 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.

[18] Arvind and X. Shen, "Using term rewriting systems to design and verify processors", *IEEE Micro*, vol. 19, pp. 36–46, 1998.

[19] A. K. Verma and P. Ienne, "Towards the Automatic Exploration of Arithmetic-Circuit Architectures", *in Design Automation Conf.*, pp. 445–450, 2006.

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

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

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

[23] S. Gopalakrishnan, P. Kalla, and F. Enescu, "Optimization of arithmetic datapaths with finite word-length operands", *in Proc. Asia-South Pacific Design Automation Conf.*, pp. 511–516, 2007.

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

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

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

[27] M. Clegg, J. Edmonds, and R. Impagliazzo, "Using the Grobner basis algorithm to find proofs of unsatisfiability", *in ACM Symp. on Theory of Computation*, pp. 174–183, 1996.

[28] C. Condrat and P. Kalla, "A Groebner Basis Approach to CNF formulae Preprocessing", *in TACAS*, 2007.

[29] M. Y. Vardi and Q. Tran, "Groebners Bases Computation in Boolean Rings for Symbolic Model Checking", *in IASTED*, 2007.

[30] H. M. Moller, "On the construction of Gröbner bases using syzygies", *J. Symb. Comp.*, vol. 6, pp. 345–359, 1988.

[31] E. Carlini, "Reducing the number of variables of a polynomial", *in Algebraic Geometry and Geometric Modeling*, 2004.

[32] "CPLEX Reference Manual, ILOG", 1999.

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

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