RTL Datapath Optimization Using System-level Transformations

Samaneh Ghandali1, Bijan Alizadeh1,2, Masahiro Fujita3, Zainalabedin Navabi1
1School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran
2School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
3VLSI Design and Education Center (VDEC), University of Tokyo, Tokyo, Japan
s.ghandali@ut.ac.ir, b.alizadeh@ut.ac.ir, fujita@ee.t.u-tokyo.ac.jp, navabi@ut.ac.ir

Abstract
This paper describe a system-level approach to improve the area and delay of datapath designs that perform polynomial computations over $\mathbb{Z}_2^m$, which are used in many applications such as computer graphics and digital signal processing domains. This approach optimizes the implementation of multivariate polynomial systems in terms of the number of arithmetic operations by performing optimization on a system level prior to high-level synthesis. Univariate functional decomposition of polynomial expressions and canonization form over $\mathbb{Z}_2^m$ are used in this method. We use GAUT high-level synthesis tool to generate RTL datapath architectures for the optimized polynomials. Experimental results on a set of benchmark applications with polynomial expressions show that this method outperforms conventional methods in terms of the area of the sequential datapath architectures in speed optimization mode with an average improvement of 25.81%, and the required clock cycles in two modes of speed optimization and area optimization, with an average improvement of 23.48% and 38.24%, respectively.

Keywords
High-level synthesis, system-level transformations, register transfer level (RTL), polynomial datapath, univariate functional decomposition, canonization form

1. Introduction
As the complexity and size of modern embedded application is continuously increasing, designing hardware at higher levels of abstraction for faster design adjustments and higher simulation speed is necessary. Conventional high level synthesis techniques are not efficient to eliminate redundancy and common sub-expression for polynomial datapaths over $\mathbb{Z}_2^m$. Such polynomial functions have been optimized manually to achieve efficient register-transfer-level (RTL) implementation. This process can be time consuming and error prone. Hence, developing high level synthesis and optimization techniques to automate the design of custom polynomial datapaths from a behavioral description is desirable.

The Horner form of a polynomial expression is a normal form representation using a nested format. This method transforms the expression into a sequence of nested additions and multiplications, which are suitable for univariate polynomials and for sequential machine evaluation using multiplier-accumulator units.

Another algebraic technique is based on kernel/co-kernel computation [6], in which first, lowest cost form of given polynomials from canonization, square-free factorization and original forms is taken into consideration. Then common coefficients and common cubes are extracted using the kernel/co-kernel extraction technique from [7]. Common sub-expressions are determined using algebraic division technique. This method is only applicable to those polynomials in which linear blocks exist explicitly.

In [7] and [8], a factoring method was proposed employing kernel/co-kernel extraction with common sub-expression elimination to reduce the size of implementation. The approximate factorization algorithm presented in [8] represents an arithmetic function $f$ as a product of sub-functions $f = f_1 f_2 \times \ldots \times f_n$ where $f_i$ is a multivariate polynomial. However, this algorithm is able to factorize square-free polynomials and cannot deal with a sub-function $f_i$ with a degree higher than one.

Another algebraic method has been proposed in [3] and then improved in [4]. The main idea is somehow similar to algebraic division techniques used in logic synthesis. This technique tries to decompose the original polynomial $poly$ as $poly = p_1 \times p_2 + p_3$ while $p_3$ should be minimized. For doing so, all possible initial values of $p_1$ and $p_2$ must be evaluated. Then for each initialization it is necessary to check whether other monomials in poly can be represented in the form $p_1 \times p_2$. Finally, the best initialization, which constitutes the lowest complexity $p_2$, is chosen. The algebraic technique in [2] improves the optimization heuristics in [3] and [4] to extract more common sub-expressions by considering single-variable and hidden monomials. This technique makes use of finite ring algebra and Modular Horner Expansion Diagram [5]. This method first reduces the original polynomials over $\mathbb{Z}_2^m$. Then common sub-expressions are extracted based on two heuristics. The main disadvantage of this technique is that decompositions are started from reduced polynomials while if the original polynomials are used more common sub-expressions would be extracted.

The Algebraic method in [1] proposed for the first time a kind of polynomial optimization technique based on redundancy addition/removal. The main idea is somehow similar to logic optimization based on redundancy addition/removal which has been developed in logic synthesis area. In this method, first, kernels/co-kernels of given polynomials are extracted as good building blocks, then a large number of vanishing polynomials over $\mathbb{Z}_2^m$, which are equal to 0 over $\mathbb{Z}_2^m$, are generated as redundancy in order to transform the given polynomials in such a way that more common sub-expressions can be extracted. Finally, using algebraic division common sub-expressions are determined.

In the current paper, we introduce some system-level techniques for transformation of the given system of
polynomials, which offer more common sub-expressions. Our optimization method reduces the complexity of polynomial datapaths in terms of the number of arithmetic operations by performing optimization on a system-level prior to high-level synthesis. Furthermore, in order to generate RTL datapath architecture for the optimized polynomials, we use GAUT high-level synthesis tool [12] as a high-level synthesis tool, although any other high level synthesis tools can be utilized. Our optimization method reduces the area and the number of clock cycles at the RTL datapath architectures. In this method, we use mathematics concept of univariate functional decomposition of polynomial expressions in order to obtain good building blocks and hence extract more common sub-expressions.

In summary, our design flow in this paper consists of the following tasks:

- System-level transformations to optimize datapath designs that perform polynomial computations over \( Z_2^m \) using univariate functional decomposition and canonization form.
- Univariate functional decomposition of the given polynomials to obtain good building blocks and extract suitable common sub-expressions.
- High-level synthesis using GAUT [12] to generate datapath architectures for the optimized polynomials as sequential circuits.
- Evaluating the performance of the proposed method and showing its effectiveness by comparing it with the state-of-the-art polynomial optimization methods in the literature.

The remainder of this paper is organized as follows. Section 2 introduces some preliminaries which are used in the rest of the paper. A motivational example is presented in section 3. Section 4 explains, in detail, our proposed polynomial optimization method. Section 5 evaluates the performance of our algorithms and presents experimental results that demonstrate their effectiveness. Finally, section 6 provides our conclusion.

2. Preliminaries

This section introduces some preliminaries which are used in the rest of the paper. In this paper arithmetic data paths are modeled as polynomial functions over \( Z_2^{n_1} \times Z_2^{n_2} \times \ldots \times Z_2^{n_d} \) to \( Z_2^m \) [9]. Let \( f_1(x), \ldots , f_p(x) \) be \( p \) given polynomial functions over \( Z_2^{n_1} \times Z_2^{n_2} \times \ldots \times Z_2^{n_d} \) to \( Z_2^m \) as the specification where \( x = < x_1,x_2,\ldots,x_d > \) is a vector of \( d \) input variables and \( n_1, n_2, \ldots , n_d \) denote size of the corresponding variables. \( Z_2^n \) represents the finite set of integers \( \{0, 1, \ldots , 2^n-1\} \). \( m \) is the size of the output bit-vector \( f \).

**Theorem 1:** Let \( f \) be a polynomial function from \( Z_2^{n_1} \times \ldots \times Z_2^{n_d} \) to \( Z_2^m \). Then according to [9], \( f \) can be uniquely represented in a canonical form as \( (1) \), where \( Y_k \) is falling factorial of degree \( k \in Z \) (\( Z \) denotes the ring of integers) and is defined as follows, \( Y_k(x) = 1 \), \( Y_k(x) = x \).

\( a_k \) is an integer such that \( 1 \leq a_k < \frac{2^m}{\text{gcd}(m, k_1, k_2, \ldots , k_p)} \), for each \( k = 1, 2, \ldots , a_k, \) and \( \mu_k = \min(2^{n_i}, SF(2^m)) \). \( SF(n) \) is the least \( k \in N \) such that \( n \) divides \( k! \), and denotes Smarandache function [10]. \( \text{gcd}(x,y) \) computes the greatest common divisor of \( x \) and \( y \).

\[ f = \sum_k a_k Y_k = \sum_k a_k \times Y_{k_1}(x_1) \times \ldots \times Y_{k_d}(x_d) \quad (1) \]

For example, let \( f = 2x^4+3x^3+3x^2+3x \), the canonical form of \( f \) over \( Z_{23} \) is \( 2x^2 \). Note that the canonical form of a polynomial over \( Z_{2^m} \) may be zero.

**Definition 1:** If \( f \) and \( h \) are univariate polynomials, then univariate polynomial \( f(x) = g(x) \circ h(x) \) is their functional composition, and \( (g, h) \) is a univariate functional decomposition of \( f \), where \( g \) and \( h \) are polynomials with lower degree than \( f \) and are called left decomposition factor and right decomposition factor of \( f \), respectively. \( o \) is the composition operator via computing the output of \( g \) when it has an argument of \( h(x) \) instead of \( x \) (i.e., \( f(x) = g(x) \circ h(x) \)).

**Example 1:** Let \( f(x) = x^4+x^2-3 \), then \( f(x) = g(x) \circ h(x) = (x^2+x-3) \circ x^2 \) is a univariate functional decomposition of \( f \), where \( g(x) = x^2+x-3 \) and \( h(x) = x^2 \).

3. Motivational Example

In this section, we present an example to motivate the optimization technique to be presented. In order to demonstrate the effectiveness of the proposed method, let us consider the following polynomial system.

\[ f_1(x) = x^4+2x^2+x^2+y^3-3y^2+2xy \]
\[ f_2(x) = x^6+3x^2+3x^4+x^2+2x \]

This system needs 32 multiplications and 10 additions.

After applying the factorization technique using MATLAB [11] to these polynomials, \( f_1 \) and \( f_2 \) are transformed to the following forms

\[ f_1(x) = x(x^3+2x^2+x+y^3-3y^2+2xy) \]
\[ f_2(x) = x(x+1)(x^2+2x^3+2x^2+1) \]

which need 19 multiplications and 9 additions.

By applying our proposed optimization method over \( Z_{2^2} \) to the original polynomials, \( f_1 \) is converted to the following form,

\[ h(x) = x^2+x, \quad g(x) = x^4 \]
\[ f_1(x) = g \circ h = x^4 \circ h(x) = x^4 \circ (x^2+x) = x^6+3x^2+3x^4+x^2+2x \]

because canonical form of \( x^6+3x^2+3x^4+x^2+2x \) over \( Z_{2^2} \) is 0,

\[ f_1(x) = x^2 \circ (x^2+x) = (x^2+x)^2 = h^2 \]

and \( f_2 \) is converted as follows.

\[ h(x) = x^2+x, \quad g(x) = x^4 \]

\[ f_2(x) = x^6 \circ h(x) = x^6 \circ (x^2+x) = x^8+x^6+2x^4+2x^3+2x^2+2x+1 \]

which need 15 multiplications and 9 additions.
Figure 1: (a) Datapath architecture of the polynomials, implemented using factorization, (b) Datapath architecture of the polynomials, implemented using our proposed method

\[ f_2(x) = g \circ h = (x^3 + x) \circ (x^2 + x) = (x^2 + x)^3 + x^2 + x = h^3 + h = h(h^2 + 1) = h(f_1 + 1). \]

The optimized polynomial system requires only 3 multiplications and 2 additions. We have used GAUT as a high-level synthesis tool to generate datapath architectures for the polynomial systems. GAUT tool has been used in many academic projects, and its HLS algorithms for binding, allocation, and scheduling are well documented [12].

We have used GAUT to generate datapath architectures for two modes; speed optimization and area optimization in which only one functional unit is considered for each operation type existed in the design. The datapath architecture of the polynomials, implemented using factorization, in the speed optimization mode is shown in Fig. 1(a). The datapath architecture of the polynomials, implemented using our proposed method is shown in Fig. 1(b).

The results reported by GAUT for the polynomials, implemented using factorization and our proposed method are shown in Table 1. We have used “notch” library, provided by GAUT, and we have set clock cycle to 20. This table reports area and number of the clock cycles, registers, multiplexers, and functional units (adder, subtracter, multiplier) in the datapath architectures of the factored polynomials and optimized polynomials using our proposed method, in speed optimization and area optimization modes.

Table 1. Gaut report for the polynomials, implemented using factorization, and for the polynomials, implemented using our proposed method, in speed optimization and area optimization modes.

<table>
<thead>
<tr>
<th></th>
<th>Factorization</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed Optimization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycles</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Registers</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>Muxes</td>
<td>160</td>
<td>32</td>
</tr>
<tr>
<td>FU +</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>FU -</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FU ×</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Area</td>
<td>530</td>
<td>91</td>
</tr>
<tr>
<td><strong>Area Optimization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycles</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>Registers</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>Muxes</td>
<td>320</td>
<td>32</td>
</tr>
<tr>
<td>FU +</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FU -</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FU ×</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Area</td>
<td>91</td>
<td>91</td>
</tr>
</tbody>
</table>

4. Proposed System-level Optimization Method

We introduce some system-level techniques for transformation of the given system of polynomials, which
offer more common sub-expressions. Our optimization method reduces the complexity of polynomial datapaths in terms of the number of arithmetic operations by performing optimization on a system-level prior to high-level synthesis. Furthermore, to generate datapath architecture for the optimized polynomials as sequential circuits, we use GAUT high-level synthesis tool. Our optimization method reduces area and number of clock cycles in the datapath architectures.

In the first phase of the proposed system-level optimization method, each given multivariate polynomial \( f(x_1, ..., x_d) \) is transformed to several univariate polynomials by representing \( f \) based on each input variable \( x_i \) (1 \( \leq i \leq d \)). Then each obtained univariate polynomial is decomposed through univariate functional decomposition algorithm explained in subsection 4.1 in order to obtain good building blocks. This phase is explained in the following steps.

Step 1: Each given multivariate polynomial \( f(x_1, ..., x_d) \) is rewritten based on each input variable \( x_i \) (1 \( \leq i \leq d \)) as (2).

\[
f = \sum_{e_1, e_2, ..., e_d \geq 0} f_{e_1, e_2, ..., e_d} x_1^{e_1} ... x_d^{e_d} \quad (2)
\]

where \( f_{e_1, e_2, ..., e_d}(x_i) \) is a univariate polynomial which represents the polynomial \( f \) based on the variable \( x_i \), and \( e_1, e_2, ..., e_d \) are degrees of \( d-1 \) variables \( x_1, ..., x_{i-1}, x_{i+1}, ..., x_d \) in polynomial \( f \).

After applying this transformation to all given polynomials \( f_j \) (1 \( \leq j \leq p \)) where \( p \) is the number of given polynomials, all obtained univariate polynomials \( f_{e_1, e_2, ..., e_d}(x_i) \) (1 \( \leq i \leq d \)) from all \( f_j \) are stored in a set named \( \text{AllSubPoly}_{x_i} \) (1 \( \leq i \leq d \)).

**Example 2:** Suppose \( f(x,y) = y^3 + 5y^2 + 10y + x^4 + 10y^2 + 2x^2y^2 + x^2y^2 + xy^3 + 2x^2y^3 + 2y^3 + 2x^2y^2 + xy^3 + 2x^2y^2 + 2y^3 - 2y^2 \).

Then \( f_i \) based on the variable \( x \) is represented as

\[
f_i = y^3(5y^2 + 10y^2) + x^2(2y^2 + 10y^2) + x^2y(3y^2 - 2y^2) + x^2y(3y^2 - 2y^2) + x^2y^2(2y^2 + 10y^2) + x^2y^3(2y^2 - 2y^2),
\]

so \( \text{AllSubPoly}_{y} = \{ f_1(y) = -y^2 - 2y^2 + 2y^2 + 10y^2, f_2(y) = 2y^2 + 2y^2 + 10y^2, f_3(y) = -y^2 - 2y^2 + 2y^2 + 10y^2, f_4(y) = 2y^2 + 10y^2, f_5(y) = y^2 + 5y^2 \} \).

And \( f_i \) based on the variable \( x \) is represented as

\[
f_i = y^3(5y^2 + 10y^2) + x^2(2y^2 + 10y^2) + x^2y(3y^2 - 2y^2) + x^2y(3y^2 - 2y^2) + x^2y^2(2y^2 + 10y^2) + x^2y^3(2y^2 - 2y^2),
\]

so \( \text{AllSubPoly}_{x} = \{ f_1(x) = 5x^6 + 10x^5 - 5x^4 + 10x^2 - 5x, f_2(x) = x^6 + 2x^6 - x^6 + 2x^2 - x^6 - x^6, f_3(x) = x^7 + 2x^2 \} \).

This phase is explained in more details in the following subsections.

**4.1. Determining Building Blocks (Phase I)**

In this phase, each given multivariate polynomial \( f \) is transformed to several univariate polynomials by representing \( f \) based on each input variable \( x_i \) (1 \( \leq i \leq d \)). Then each obtained univariate polynomial is decomposed through univariate functional decomposition algorithm explained in subsection 4.1 in order to obtain good building blocks. This phase is explained in the following steps.

Step 2: univariate functional decomposition is computed for each member of \( \text{AllSubPoly}_{x_i} \) (1 \( \leq i \leq d \)) (i.e., \( f_{e_1, e_2, ..., e_d}(x_i) \)) by using the univariate functional decomposition algorithm explained in the follow.

**Univariate functional decomposition algorithm:** Let \( g \) and \( h \) be polynomials of degrees \( r \) and \( s \) over a field. Their functional composition \( f = g \circ h = g(h) \) has degree \( n = r \times s \). The univariate functional decomposition problem can be stated as follows: given \( f \) of degree \( n = r \times s \), determine whether such \( g \) and \( h \) exist, and in the affirmative case, compute them \([13]\).

The pseudo code of the univariate functional decomposition algorithm \([14]\), which is slightly modified in our method to also calculate indecomposable part of an input polynomial, is shown in Fig. 2. For every \( r \), \( s \) values for which \( r \times s = n \), UniDec procedure in Fig. 2 with \( f \) and \( r \) as inputs calculates a univariate functional decomposition for \( f \) as (3), where \( f_0 \) is indecomposable part of \( f \).

\[
f(x) = g(x) \circ h(x) + f_0 = g(h(x)) + f_0 \quad (3)
\]

As explained in \([14]\), \( f, g \) and \( h \) are in the following forms, \( f = x^r + a_{r-1}x^{r-1} + ... + a_0 \), \( h = x^s + c_{s-1}x^{s-1} + ... + c_0 \), \( g = x^t + b_{t-1}x^{t-1} + ... + b_0 \). Respectively. In this algorithm, first, coefficients of \( h \), i.e., \( c_{i, ..., s} \), are calculated from coefficients of \( f \) by \( h \text{UniDec} \) procedure (lines 4-11 in Fig. 2). For this purpose, polynomial \( q_k \) is defined as follows:

\[
q_k = x^k + c_{k}x^k + ... + c_{k+k}x^k, 0 \leq k \leq s.
\]

Then \( q_k = x^k \), \( q_k = a_{k}x^k \), and \( q_k = q_k + c_{k}x^k, 1 \leq k \leq s \).

According to \([14]\), we can calculate the first \( k+1 \) coefficients of \( h \) from coefficients of \( q_k \). The \( k+1 \)th coefficient of \( q_k \) is the coefficient of \( x^{k+k+1} \), this agree with \( q_k+k+1 \), i.e., the \( k+1 \)th coefficient of \( f, l \leq k \leq s-1 \). Thus if the earlier coefficients \( c_{i, ..., s} \) of \( h \) are known, then \( c_{i,k} \) can be determined by computing

\[
c_{i,k} = c_{i+k} - c_{i+k} \quad 1 \leq k \leq s - 1,
\]

where, \( d_i \) is the coefficient of \( x^{i+k} \) in \( q_k \) \([14]\).
let $A[i,j]$ be the coefficient of $x^i$ in $h^j$, $0 \leq i, j \leq r$. Then $b = (b_0, \ldots, b_s)$ can be determined by solving the following equation:

$$Ab = a,$$

where $a = (a_0, a_1, \ldots, a_r)$ are the coefficients of $f$.

Then, composition of $h$ and $g$ is computed by using the function `subs`, which is a function library of Maple [15] and computes the value of $g \circ h$. The difference between $f$ and $g \circ h$ is considered as indecomposable part of $f$ and refereed as $f_0$ (line 15 in Fig. 2).

**Example 3:** Let us consider a member of $\text{AllSubPoly}_x$ in example 2, $f = x^6 + 2x^5 - x^4 - x^3 + 2x^2 - x$. Because $n = 6$, one of the situation that $r$ and $s > 1$ are $r = 2$, $s = 3$. So $g$ and $h$ are in the following forms.

$$h = x^2 + c_2 x + c_1, g = x^3 + b_2 x + b_0.$$

The steps of the univariate decomposition algorithm are as follows.

1. $q_0^0 = 1, q_1^0 = x^3, q_0^1 = x^6$
2. $d_1 = \text{coef}(q_0^0, x^{6-1}) = 0, c_2 = \frac{(a_x - d_1)}{2} = 1$
3. $q_0^1 = 1, q_1^1 = x^3 + x^2, q_1^2 = x^6 + 2x^5 + x^2$

**Figure 2:** Univariate functional decomposition algorithm

**Example 4:** Let us consider example 2 again. By computing the univariate functional decomposition of $x_1$ and $\text{AllSubPoly}_y$, $h_{\text{set}}_x$ for variable $x$ and $h_{\text{set}}_y$ for variable $y$ are obtained as follows.

$$h_{\text{set}}_x = \{x^3 + x^2 - x, x^2 + x, x^3 + x, x^2\},$$

$$h_{\text{set}}_y = \{y^2, y^2 - 5y\}.$$

### 4.2. Common sub-expression extraction (Phase II)

The aim of the second phase of the proposed method is to extract common sub-expressions between all given multivariate polynomials, which is equal to extract common sub-expressions between their equivalent univariate polynomials which are stored in $\text{AllSubPoly}_x(1 \leq i \leq d)$.

To extract common sub-expressions we make use of univariate functional decomposition algorithm unlike other works that utilize algebraic division technique [1][6][7]. By considering members of $h_{\text{set}}_{x_i}$ as good building blocks, we try to re-decompose all members of $\text{AllSubPoly}_{x_i}$ by these building blocks and find common sub-expressions between them.

By using $g_{\text{UniDec}}$ procedure described in subsection 4.1, each $f_{e_{\gamma_1 \cdots \gamma_i \cdots \gamma_s}}(x) \in \text{AllSubPoly}_{x_i}$ is assessed whether a polynomial $g'$ can be calculated from this polynomial and each member of $h_{\text{set}}_{x_i}$ as shown in (5),

$$f_{e_{\gamma_1 \cdots \gamma_i \cdots \gamma_s}}(x) = g'(x_i) \circ h'(x_i) + f_0(x_i)$$

(5)
where \( g'(x_i) \) is a new right decomposition factor of \( f_{e_1,...,e_i-1,e_i+1,...,e_d} \). \( f_0(x_i) \) is a new indecomposable part of \( f_{e_1,...,e_i-1,e_i+1,...,e_d} \) and \( h'(x_i) \) is a member of \( h_{set_{x_i}} \). Please note that each member of \( h_{set_{x_i}} \) may belong to different members of AllSubPoly_{x_i}.

To reduce cost of the corresponding hardware implementation of each polynomial, we make use of canonical representation of \( f_0(x_i) \) over \( Z_{2^m} \) to \( Z_{2^m} \), which is explained in section 2, and then compare cost of the canonical form with the original form of \( f_0(x_i) \), and select the lower cost form.

### Example 5:
Let us consider two members of \( \text{AllSubPoly}_x \) in example 2: \( p_1 = x^6 + 2x^5 - x^4 - x^3 + 2x^2 - x \) and \( p_2 = (x^2 + 2x) o (x^3 + x^2 - x) - 3x^2 \).

**Form 1:**
\[
\text{canonical form}(f_0) = 2x^2, \quad \text{so } p_1 = (x^2 - x) o (x^3 - x) + 2x^2 \text{ over } Z_{2^3},
\]
\[
p_2 = x^2 o h_1 = x^2 o (x^3 - x).
\]

**Form 2:**
\[
p_1 = (x^2 + x) o h_2 = (x^2 - x) o (x^3 + x^2 - x),
\]
\[
p_2 = (x^2 + 2x) o h_1 + f_0 = (x^2 + 2x) o (x^3 + x^2 - x) - 2x^2 - x^4 - 2x^2 + 2x.
\]

**canonical form**(f_0) = -3x^2, so \( p_2 = (x^2 + 2x) o (x^3 + x^2 - x) - 3x^2 \text{ over } Z_{2^3} \).

Therefore result of the common sub-expression extraction phase is various decompositions of each \( f_{e_1,...,e_i-1,e_i+1,...,e_d} \) based on the different building blocks belonged to different members of AllSubPoly_{x_i}.

### 4.3. Complete system-level optimization algorithm
The complete proposed system-level optimization algorithm is explained in this subsection. The pseudo-code of the proposed method is shown in Fig. 3. Lines 3-10 describe the first phase in the proposed method in which each input multivariate polynomial is transformed to several univariate polynomials which are stored in AllSubPoly_{x_i} (1 \leq i \leq d) (lines 3-6). Then univariate functional decomposition algorithm in Fig. 2 is applied to these univariate polynomials and then all obtained right decomposition factors are stored in \( h_{set_{x_i}} \) as good building blocks (lines 7-10). Lines 11-16 describe the second phase in which each member of AllSubPoly_{x_i} is re-decomposed by members of \( h_{set_{x_i}} \) using \( g_{UniDec} \) procedure in Fig. 2 (lines 11-14), and common sub-expressions are determined. The canonical form over \( Z_{2^m} \) to \( Z_{2^m} \) is calculated for \( f_0(x) \) in order to reduce cost of the implementation (lines 15-16). Finally, to select the form with the smallest number of arithmetic operations, every new generated form of all members of AllSubPoly_{x_i} is considered to evaluate related hardware implementation by computing \( \text{cost_func} \) function. This function determines the number of arithmetic operations such as additions and multiplications needed for implementation of the given polynomials (lines 17-22).

![Figure 3: System-level optimization algorithm](image)

**Example 6:** Let us consider the polynomial system in example 2. This polynomial system originally needs 115 multiplications and 23 additions. By applying our proposed method, we get an implementation with only 16 multiplications and 10 additions as shown below.

\[
t_1 = x^2, \quad t_2 = x(t_1 - 1), \quad t_3 = t_1 + t_5, \quad t_4 = y^2, \quad f_1 = t_4 t_1 (x + 2) + y (y + 5) t_1 (t_3 + 1),
\]
\[
f_2 = (t_1^2 + x)(t_2(2 + t_4) - 2y) + yt_1 t_3^2.
\]

The results reported by GAUT for the datapath architectures of the original and optimized polynomials as sequential circuits, in speed optimization mode, are shown in Table 2.

![Table 2. GAUT report for the original and optimized polynomials](image)

### 5. Experimental Results
In order to show the effectiveness of our proposed optimization method, we have employed different polynomials extracted from real embedded systems. Various combinations of multivariate cosine wavelet (MVCS) for graphic applications [7], Savitzky-Golay (SG) filters [16] and digital image rejection unit (DIRU) for image processing...
applications, Quadratic filters (Quad) for DSP applications [17], Phase-Shift Keying (PSK) for digital communication [18] have been taken into account as multi-output polynomials.

We have implemented the proposed method along with the methods in [1] and [6] in Maple [15], and then we have used GAUT as a high-level synthesis tool [12] to automatically generate datapath architectures for obtained polynomials.

To generate datapath architectures based on optimized polynomials, we considered two modes provided by GAUT: speed optimization and area optimization. Table 3 illustrates the results obtained using our proposed method, Horner form, and methods in [1] and [6] for speed optimization mode. This table reports area and number of the clock cycles, required clock cycles are reduced by an average of 38.11%, 20.10%, and 12.24% in comparison with the Horner form, the method in [6] and the method in [1] across all benchmarks. This reduction indicates that our goal of reducing critical path delay has been achieved. Furthermore, area is improved in our method with an average improvement of 25.81% in comparison with other works.

In the area optimization mode, only one functional unit is considered for each operation type existed in the design (i.e., all operations with the same type should be bound to a same functional unit). Table 4 illustrates the results obtained using our proposed method, Horner form, and methods in [1] and [6] for area optimization mode. The results reported in the table indicate that our proposed method provides fewer required clock cycles in comparison with the Horner form, the method in [6] and the method in [1] with an average of 64.73%, 49.97%, and 0.01%, respectively, across all benchmarks. Such improvement in the required clock cycles with a fixed number of functional units indicates that by using our method the number of operations would be fewer than those of other works. In other words, our proposed method can efficiently determine common sub-expressions.

### 6. CONCLUSION

In this paper we have proposed a system-level optimization method for the data paths implemented using a system of polynomials. Our method optimizes polynomials to reduce the complexity of polynomial datapaths in terms of the number of arithmetic operations over \( Z_{2m} \). In the proposed method first all given multivariate polynomials are transformed to several univariate polynomials. Then univariate functional decompositions are calculated for them to obtain good building blocks. To extract common sub-expressions we make use of univariate functional

### Table 3. Comparison of the datapath architectures in the speed optimization mode.

<table>
<thead>
<tr>
<th></th>
<th>DIRU PSK Quad</th>
<th>DIRU PSK SG2</th>
<th>DIRU Quad SG2</th>
<th>DIRU MVCS SG2</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>17</td>
<td>17</td>
<td>16</td>
<td>16</td>
<td>0.00</td>
</tr>
<tr>
<td>Registers</td>
<td>22</td>
<td>24</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Muxes</td>
<td>384</td>
<td>352</td>
<td>324</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>11</td>
<td>13</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>937</td>
<td>1103</td>
<td>605</td>
<td>613</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. Comparison of the datapath architectures in the area optimization mode.

<table>
<thead>
<tr>
<th></th>
<th>DIRU PSK Quad</th>
<th>DIRU PSK SG2</th>
<th>DIRU Quad SG2</th>
<th>DIRU MVCS SG2</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>72</td>
<td>86</td>
<td>68</td>
<td>86</td>
<td>0.00</td>
</tr>
<tr>
<td>Registers</td>
<td>13</td>
<td>15</td>
<td>13</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Muxes</td>
<td>592</td>
<td>656</td>
<td>560</td>
<td>672</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>99</td>
<td>99</td>
<td>91</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>DIRU PSK Quad</th>
<th>DIRU PSK SG2</th>
<th>DIRU Quad SG2</th>
<th>DIRU MVCS SG2</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>52</td>
<td>91</td>
<td>72</td>
<td>71</td>
<td>8.38</td>
</tr>
<tr>
<td>Registers</td>
<td>16</td>
<td>27</td>
<td>17</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Muxes</td>
<td>672</td>
<td>1056</td>
<td>800</td>
<td>832</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>99</td>
<td>99</td>
<td>91</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>DIRU PSK Quad</th>
<th>DIRU PSK SG2</th>
<th>DIRU Quad SG2</th>
<th>DIRU MVCS SG2</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>36</td>
<td>36</td>
<td>55</td>
<td>65</td>
<td>37.92</td>
</tr>
<tr>
<td>Registers</td>
<td>12</td>
<td>11</td>
<td>18</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Muxes</td>
<td>411</td>
<td>432</td>
<td>752</td>
<td>832</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>99</td>
<td>99</td>
<td>91</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>DIRU PSK Quad</th>
<th>DIRU PSK SG2</th>
<th>DIRU Quad SG2</th>
<th>DIRU MVCS SG2</th>
<th>%Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>41</td>
<td>48</td>
<td>49</td>
<td>52</td>
<td>38.68</td>
</tr>
<tr>
<td>Registers</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Muxes</td>
<td>624</td>
<td>704</td>
<td>720</td>
<td>752</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>99</td>
<td>99</td>
<td>91</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>

In this paper we have proposed a system-level optimization method for the data paths implemented using a system of polynomials. Our method optimizes polynomials to reduce the complexity of polynomial datapaths in terms of the number of arithmetic operations over \( Z_{2m} \). In the proposed method first all given multivariate polynomials are transformed to several univariate polynomials. Then univariate functional decompositions are calculated for them to obtain good building blocks. To extract common sub-expressions we make use of univariate functional
decomposition algorithm. We have used GAUT high-level synthesis tool to generate RTL datapath architectures for the optimized polynomials as sequential circuits. Experimental results show superiority of our approach in the area and delay savings in contrast with the other related works. As a future work, we are going to utilize multivariate functional decomposition algorithm to extract better building blocks.

References


[9] Z. CHEN,"On polynomial functions from $\mathbb{Z}_m^1 \times \mathbb{Z}_m^2 \times \cdots \times \mathbb{Z}_m^n$ to $\mathbb{Z}_m$: Discrete Math., Vol. 162, No. 1–3, pp. 67–76, 1996.


