RTL Datapath Optimization Using System-level Transformations

Samaneh Ghandali[1], Bijan Alizadeh[1,2], Masahiro Fujita[3], Zainalabedin Navabi[1]
[1]School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran
[2]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[3]VLSI 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 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 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 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 $f = f_1 \times f_2 \times \ldots \times f_n$ 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_3, 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 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 Z_{2^m}, which are equal to 0 over 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 $Z_{2^{n_1}} \times Z_{2^{n_2}} \times ... \times Z_{2^{n_d}}$ to Z_{2^m}[9]. Let $f_1(\bar{x}),\ldots, f_p(\bar{x})$ be p given polynomial functions over $Z_{2^{n_1}} \times Z_{2^{n_2}} \times ... \times Z_{2^{n_d}}$ to Z_{2^m} as the specification where $\bar{x} = < x_1,x_2,...,x_d >$ is a vector of d input variables and $n_1, n_2, ..., n_d$ denote size of the corresponding variables. Z_{2^n} represents the finite set of integers $\{0, 1, \ldots, 2^n\text{-}1\}$. m is the size of the output bit-vector f.

Theorem 1: Let f be a polynomial function from $Z_{2^{n_1}} \times ... \times Z_{2^{n_d}}$ 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,

$$Y_0(x)=1,\ \ Y_1(x) = x,$$

$$Y_2(x)= x\times(x-1),\ \ldots,\ Y_k(x)=Y_{k-1}(x)\times(x-k+1).$$

a_K is an integer such that $1 \le a_K < \frac{2^m}{gcd(2^m, \prod_{i=1}^{d} k_i!)}$, $K=<k_1, k_2, ..., k_d>$ for each $k_i= 1, 2, ..., \mu_i$, and $\mu_i = min\{2^{n_i}, SF(2^m)\}$. $SF(n)$ is the least $k \in \mathbb{N}$ such that n divides $k!$, and denotes *Smarandache* function [10]. $gcd(x,y)$ computes the greatest common divisor of x and y.

$$f = \sum_K a_K Y_K = \sum_K a_K \times Y_{k_1}(x_1) \times ... \times Y_{k_d}(x_d) \quad (1)$$

For example, let $f = 2x^5+x^4+x^2-2x$, the canonical form of f over Z_{2^3} is $2x^2$. Note that the canonical form of a polynomial over $Z_{2^{n_1}} \times Z_{2^{n_2}} \times ... \times Z_{2^{n_d}}$ to Z_{2^m} may be zero.

Definition 1: If g and h are univariate polynomials, then univariate polynomial $f(x) = g(x)\ o\ h(x)$ 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., $f(x)=g(x)\ o\ h(x) =g(h(x))$).

Example 1: Let $f(x)= x^4+x^2-3$, then $f(x)=g(x)\ o\ h(x) = (x^2+x-3)\ o\ x^2$ 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.

$f_1(x) = x^4+2x^3+x^2+xy^3-3xy^2+2xy$

$f_2(x) = x^6+3x^5+3x^4+x^3+x^2+ x.$

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

$f_1(x) = x(x^3+2x^2+x+y^3-3y^2+2y)$

$f_2(x) = x(x+1)(x^4+2x^3 +x^2+ 1)$,

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,

$h(x) = x^2+x,\ g(x) = x^2$

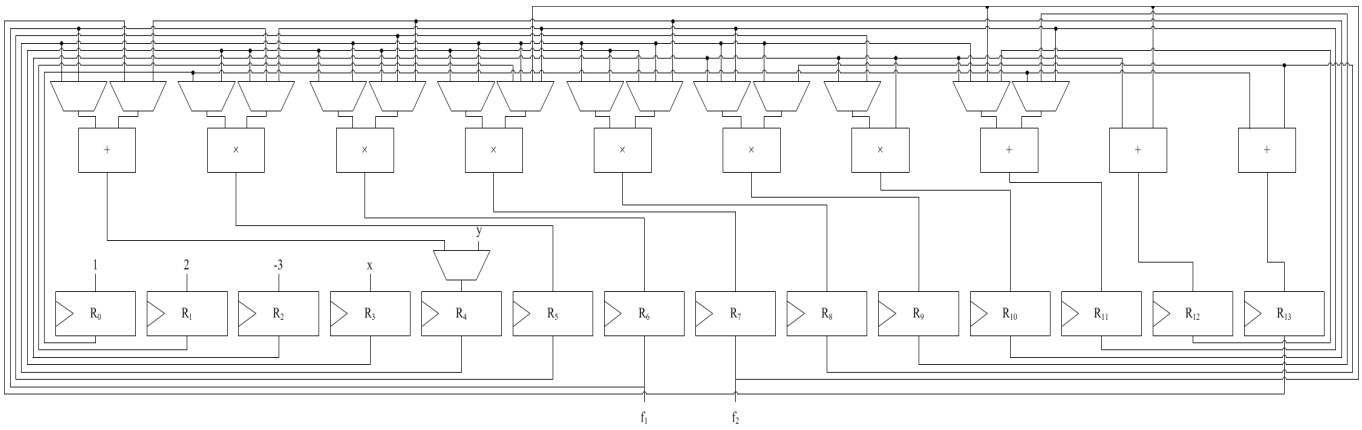
$f_1(x) = g\ o\ h + xy^3-3xy^2+2xy = x^2\ o\ (x^2+x) + xy^3-3xy^2+2xy,$

because canonical form of xy^3-3xy^2+2xy over Z_{2^2} is 0,

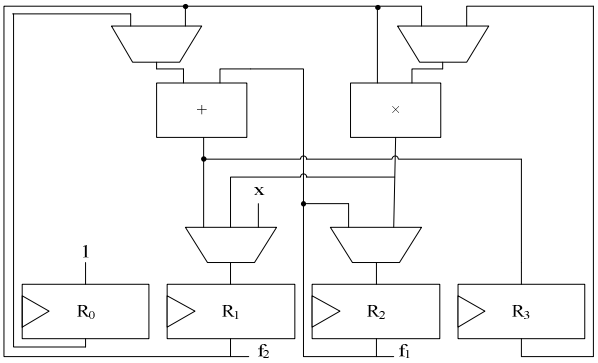
$f_1(x)= x^2\ o\ (x^2+x) = (x^2+x)^2 = h^2,$

and f_2 is converted as follows.

$h(x) = x^2 + x,\ g(x) =x^3 + x,$



(a)



(b)

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.

			Factorization	Proposed Method
Speed Optimization	**Cycles**		6	6
	Registers		14	4
	Muxes		160	32
	FU	+	4	1
		-	0	0
		×	6	1
	Area		530	91
Area Optimization	**Cycles**		22	6
	Registers		13	4
	Muxes		320	32
	FU	+	1	1
		-	0	0
		×	1	1
	Area		91	91

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 $f(x_1,\ldots, x_d)$ is transformed to several univariate polynomials by representing f based on each input variable x_i $(1 \le i \le d)$. Then each obtained univariate polynomial is decomposed through univariate functional decomposition algorithm explained in subsection 4.1 in order to obtain good building blocks. In the second phase, to extract common sub-expressions among the given polynomials, we make use of univariate functional decomposition algorithm unlike other works that utilize algebraic division technique [1][6][7]. Finally, among various forms of the polynomials in terms of the extracted common sub-expressions, the form with smallest number of the arithmetic operations is selected. These phases are 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 variables. Then each obtained univariate polynomial is decomposed through univariate functional decomposition algorithm in order to obtain good building blocks. This phase is explained in the following steps.

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

$$f = \sum_{e_1,..,e_{i-1},e_{i+1},..,e_d \ge 0} f_{e_1,..,e_{i-1},e_{i+1},..,e_d} x_1^{e_1} \ldots x_{i-1}^{e_{i-1}} x_{i+1}^{e_{i+1}} \ldots x_d^{e_d} \quad (2)$$

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

After applying this transformation to all given polynomials f_j $(1 \le j \le p)$ where p is the number of given polynomials, all obtained univariate polynomials $f_{e_1,\ldots,e_{i-1},e_{i+1},\ldots,e_d}(x_i)$ $(1 \le i \le d)$ from all f_j are stored in a set named $AllSubPoly_{x_i}$ $(1 \le i \le d)$.

Example 2: Suppose $f_1(x,y) = x^6y^2+5x^6y+2x^5y^2+10x^5y-x^4y^2-5x^4y+x^3y^4-x^3y^2-5x^3y+2x^2y^4+2x^2y^2+10x^2y-xy^2-5xy$, and $f_2(x,y) = x^6y^3+x^4y^4-2x^4y^3+2x^4y^2-2x^4y+x^2y^3+xy^4+2xy^2-2xy$.

Then f_1 based on the variable y is represented as

$f_1 = x^6(y^2+5y)+x^5(2y^2+10y)+x^4(-y^2-5y)+x^3(y^4-y^2-5y)+x^2(2y^4+2y^2+10y)+x(-y^2-5y)$,

so $AllSubPoly_y$ = { $f_1(y)= -y^2-5y$, $f_2(y)=2y^4+2y^2+10y$, $f_3(y)=y^4-y^2-5y$, $f_4(y)=-y^2-5y$, $f_5(y)=2y^2+10y$, $f_6(y)=y^2+5y$}.

And f_1 based on the variable x is represented as

$f_1 = y^4(x^3+2x^2)+y^2(x^6+2x^5-x^4-x^3+2x^2-x)+y(5x^6+10x^5-5x^4-5x^3+10x^2-5x)$,

so $AllSubPoly_x$ ={$f_1(x)=5x^6+10x^5-5x^4-5x^3+10x^2-5x$, $f_2(x)=x^6+2x^5-x^4-x^3+2x^2-x$, $f_4(x)= x^3+2x^2$}.

f_2 based on the variable y is represented as

$f_2 = x^6(y^3)+x^4(y^4-2y^3+2y^2-2y)+x^2(y^3)+x(y^4+2y^2-2y)$,

so $AllSubPoly_y = AllSubPoly_y \cup \{y^3, y^4-2y^3+2y^2-2y, y^4+2y^2-2y\}$.

And f_2 based on the variable x is represented as

$f_2 = y^4(x^4+x)+y^3(x^6-2x^4+x^2)+y^2(2x^4+2x)+y(-2x^4-2x)$,

so $AllSubPoly_x = AllSubPoly_x \cup \{x^4+x,\ x^6-2x^4+x^2,\ 2x^4+2x,\ -2x^4-2x\}$.

Step 2: univariate functional decomposition is computed for each member of $AllSubPoly_{x_i}$ $(1 \le i \le d)$ (i.e., $f_{e_1,\ldots,e_{i-1},e_{i+1},\ldots,e_d}(x_i)$) 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 $f = g\ o\ h = g(h)$ 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 and 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.

$$f(x) = g(x) \circ h(x) + f_0 = g\big(h(x)\big) + f_0 \quad (3)$$

As explained in [14], f, g and h are in the following forms, $f = x^{rs}+a_{rs-1}x^{rs-1}+\ldots+a_0$, $h = x^s+c_{s-1}x^{s-1}+\ldots+c_1x$, $g = x^r+b_{r-1}x^{r-1}+\ \ldots\ +b_0$, respectively. In this algorithm, first, coefficients of h, i.e., $(c_1,\ldots,c_{s-1})$, are calculated from coefficients of f by *h_UniDec* procedure (lines 4-11 in Fig. 2). For this purpose, polynomial q_k is defined as follows:

$$q_k = x^s + c_{s-1}x^{s-1} + \ldots + c_{s-k}x^{s-k},\ 0 \le k \le s.$$

Then $q_0=x^s$, $q_s= q_{s-1}=h$, and $q_k = q_{k-1}+ c_{s-k}x^{s-k}$, $1 \le k \le s$.

According to [14], we can calculate the first $k+1$ coefficients of h^r from coefficients of q_k^r. The $k+1^{st}$ coefficient of q_k^r is the coefficient of x^{rs-k}, this agree with a_{rs-k}, i.e., the $k+1^{st}$ coefficient of f, $1 \le k \le s-1$. Thus if the earlier coefficients $c_{s-1},\ldots,c_{s-k+1}$ of h are known, then c_{s-k} can be determined by computing

$$c_{s-k}=\frac{a_{rs-k}-d_k}{r}, \quad 1 \le k \le s-1,$$

where, d_k is the coefficient of x^{rs-k} in q_{k-1}^r [14].

Second, from f and h, coefficients of g, i.e., $(b_0,\ldots,b_{r-1})$, are calculated by *g_UniDec* procedure (lines 12-16 in Fig. 2),

let $A[i,j]$ be the coefficient of x^{is} in h^j, $0 \le i, j \le r$. Then $b = (b_0,\ldots,b_{r-1})$, can be determined by solving the following equation:

$$Ab = a,$$

where $a=(a_0, a_s, \ldots, a_{rs})$ 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).

```
UniDec (f::polynom, r::integer)
1:  h:=h_UniDec(f, r);
2:  (g, f0):=g_UniDec(f, h, r);
3:  return each (h, g, f0);

h_UniDec(f::polynom , r::integer)
4:  q0^i=x^is, 0≤i≤r
5:  for k from 0 to (degree(f)/r)
6:     dk:= coef(q(k-1)^r, x^((degree(f)-k)) ) ;
7:     c((degree(f)/r)-k):=(a(degree(f)-k)- dk)/r ;
8:     qk+1^0:= qk^0;
9:     qk+1^j:= Σ0^j (j i) × c^i ((degree(f)/r)-k) × x^(i((degree(f)/r)-k))
                  × qk^(j-i); 1≤j≤r;
10: h:=Σ1^(degree(f)/r) ci×x^i;
11: return h;

g_UniDec(f::polynom , h::polynom , r::integer)
12: A(i+1,j+1):=coef(h^j, x^(i*(degree(f)/r))); 0 ≤i,j≤r ;
13: b := LinearSolve(A, a) ;
14: g :=Σ0^r bi×x^i ;
15: f0:= f- subs({x =h}, g);
16: return (g, f0);
```

Figure 2: Univariate functional decomposition algorithm

Example 3: Let us consider a member of $AllSubPoly_x$ 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.

$h = x^3 + c_2x^2 + c_1x$, $g = x^2 + b_1x + b_0$.

The steps of the univariate decomposition algorithm are as follows.

$q_0^0 = 1, q_0^1 = x^3, q_0^2 = x^6$

step 1: ($k = 1$)

$d_1 = coef(q_0^2, x^{6-1}) = 0, c_2 = \frac{(a_5 - d_1)}{2} = 1$

$q_1^0 = 1, q_1^1 = x^3 + x^2, q_1^2 = x^6 + 2x^5 + x^4$

step 2: (k=2)

$d_2 = coef(q_1^2, x^{6-2}) = 1, c_1 = \frac{(a_4 - d_2)}{2} = -1$

$q_1^0 = 1, q_1^1 = x^3 + x^2 - x,$

$q_1^2 = x^6 + 2x^5 - x^4 - 2x^3 + x^2$

So h is obtained as $h = x^3 + x^2 - x$.

Then by using the coefficients of f and h, coefficients of g are calculated as follows.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}, a = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

So g is obtained as $g = x^2 + x$.

The general form resulting by applying the univariate functional decomposition algorithm to each member of $AllSubPoly_{x_i}$ $(1 \le i \le d)$ is shown in (4).

$$f_{e_1,..,e_{i-1},e_{i+1},..,e_d}(x_i) = g(x_i) \circ h(x_i) + f_0(x_i),$$
$$f_{e_1,..,e_{i-1},e_{i+1},..,e_d}(x_i) \in AllSubPoly_{x_i}, \quad (1 \le i \le d) \tag{4}$$

Step 3: In the third step of the first phase of the proposed method, all obtained right decomposition factors $h(x_i)$ of all members of $AllSubPoly_{x_i}$ are stored in a set named $h\_set_{x_i}$ as good building blocks.

Example 4: Let us consider example 2 again. By computing the univariate functional decomposition of all members of $AllSubPoly_x$ and $AllSubPoly_y$, h_set_x for variable x and h_set_y for variable y are obtained as follows.

$h\_set_x = \{x^3+x^2-x, x^2+x, x^3-x, x^2\}$,
$h\_set_y = \{y^2, y^2-5y\}$.

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 $AllSubPoly_{x_i}$ $(1 \le i \le d)$.

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 $h\_set_{x_i}$ as good building blocks, we try to re-decompose all members of $AllSubPoly_{x_i}$ by these building blocks and find common sub-expressions between them.

By using *g_UniDec* procedure described in subsection 4.1, each $f_{e_1,..,e_{i-1},e_{i+1},..,e_d}(x_i) \in AllSubPoly_{x_i}$ is assessed whether a polynomial g' can be calculated from this polynomial and each member of $h\_set_{x_i}$ as shown in (5),

$$f_{e_1,..,e_{i-1},e_{i+1},..,e_d}(x_i) = g'(x_i) \circ h'(x_i) + f_0'(x_i)$$
$$h'(x_i) \in h\_set_{x_i} (1 \le i \le d) \tag{5}$$

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 be belonged 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 $Z_{2^{n_i}}$ 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 $AllSubPoly_x$ in example 2; $p_1= x^6+2x^5-x^4-x^3+2x^2-x$ belonged to f_1, and $p_2= x^6-2x^4+x^2$ belonged to f_2. Two members of h_set_x in example 4 are $h_1=x^3+x^2-x$, and $h_2=x^3-x$ which are a right decomposition factor of p_1 and p_2 respectively. By applying the common sub-expression phase to p_1 and p_2 over Z_{2^3}, two obtained forms of these polynomials are as follows.

Form 1:

$p_1=(x^2-x)\ o\ h_2 + f_0 = (x^2-x)\ o\ (x^3-x) +2x^5+x^4+x^2-2x$,

$canonical\_form(f_0) = 2x^2$, so $p_1=(x^2-x)\ o\ (x^3-x) +2x^2$ over Z_{2^3},

$p_2= x^2\ o\ h_2= x^2\ o\ (x^3-x)$.

Form 2:

$p_1=(x^2+x)\ o\ h_1=(x^2-x)\ o\ (x^3+x^2-x)$,

$p_2=(x^2+2x)\ o\ h_1 + f_0= (x^2+2x)\ o\ (x^3+x^2-x) -2x^5-x^4-2x^2+2x$.

canonical_form(f_0) = $-3x^2$, so $p_2= (x^2+2x)\ o\ (x^3+x^2-x) -3x^2$ 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 of the proposed method in which each input multivariate polynomial is transformed to several univariate polynomials which are stored in $AllSubPoly_{x_i}$ $(1 \le i \le d)$ (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 $Z_{2^{n_i}}$ to Z_{2^m} is calculated for $f_0'(x_i)$ 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 *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).

```
    Optimization Algorithm (LP, d)
1:  LP := List of input polynomials (f1, f2, ..., fp)
2:  d := Number of variables
3:  for j from 1 to p
4:     for i from 1 to d
5:        consider fj as fj:=Σ_{e1,...,ei-1,ei+1,...,ed≥0} f_{e1,...,ei-1,ei+1,...,ed} × x1^e1 ×
                          ...× x_{i-1}^{e_{i-1}} × x_{i+1}^{e_{i+1}} ×...× xd^ed;
6:        Add every f_{e1,...,ei-1,ei+1,...,ed}(xi) to AllSubPoly_{xi};
7:  for i = 1 to d do
8:     for k from 1 to size(AllSubPoly_{xi})
9:        (h, g, f0):=UniDec(AllSubPoly_{xi},r);
10:       Add h to h_set_{xi};
11: for i = 1 to d do
12:    for j from 1 to size(AllSubPoly_{xi})
13:       for each h' of h_set_{xi};
14:       (g', f'0):=g_UniDec(AllSubPoly_{xi}, h',r);
15:       if(cost_func (canonical_form(f'0)) < cost_func (f'0))
16:       f'0:= canonical_form(f'0);
17: for j = 1 to p do
18:    f'j:= reconstruct f'j from AllSubPoly_{xi};
19:  for every combination of f'j (transformation of the original fj)
20:    if(cost_func(current combination)<min_cost) then
21:       minimum cost combination = current combination
22:  return minimum cost combination
```

Figure 3: System-level optimization algorithm

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.

$t_1= x^2$, $t_2= x(t_1-1)$, $t_3= t_1+t_2$, $t_4= y^2$,
$f_1= t_4^2 t_1(x+2) + y(y+5)t_3(t_3+1)$,
$f_2= (t_1^2+x)(t_4(2+t_4)-2y)+ yt_4t_2^2$.

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.

	Cycles	Registers	Muxes	FU			Area
				+	-	×	
Original	15	21	320	3	1	12	1028
Optimized	8	12	224	2	1	4	356

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.

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

			DIRU PSK Quad	DIRU PSK SG2	DIRU Quad SG2	DIRU MVCS SG2	%Δ
Horner	Cycles		17	17	16	16	0.00
	Registers		22	24	20	20	
	Muxes		384	352	324	336	
	FU	+	2	2	3	4	
		-	1	1	0	0	
		×	11	13	7	7	
	Area		937	1103	605	613	
Technique in [6]	Cycles		12	15	17	13	13.42
	Registers		19	37	21	24	
	Muxes		272	368	306	304	
	FU	+	3	3	3	4	
		-	1	1	0	0	
		×	6	21	7	7	
	Area		530	1775	605	613	
Technique in [1]	Cycles		15	15	11	13	18.32
	Registers		16	15	20	26	
	Muxes		243	256	304	288	
	FU	+	2	2	4	4	
		-	1	1	0	0	
		×	5	5	7	15	
	Area		439	439	613	1277	
Our approach	Cycles		13	13	11	11	27.39
	Registers		17	19	20	22	
	Muxes		320	336	288	272	
	FU	+	2	2	2	3	
		-	1	1	0	0	
		×	4	5	7	7	
	Area		356	439	597	605	

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, registers, multiplexers, and functional units (adder, subtracter, multiplier) in the obtained datapath architectures. %Δ indicates the percent of improvement in the number of required clock cycles in all methods compared to the Horner form. The results in the table indicate that in our method 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.

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

			DIRU PSK Quad	DIRU PSK SG2	DIRU Quad SG2	DIRU MVCS SG2	%Δ
Horner	Cycles		72	86	68	86	0.00
	Registers		13	15	13	16	
	Muxes		592	656	560	672	
	FU	+	1	1	1	1	
		-	1	1	0	0	
		×	1	1	1	1	
	Area		99	99	91	91	
Technique in [6]	Cycles		52	91	72	71	8.38
	Registers		16	27	17	22	
	Muxes		672	1056	800	832	
	FU	+	1	1	1	1	
		-	1	1	0	0	
		×	1	1	1	1	
	Area		99	99	91	91	
Technique in [1]	Cycles		36	36	55	65	37.92
	Registers		12	11	18	19	
	Muxes		411	432	752	832	
	FU	+	1	1	1	1	
		-	1	1	0	0	
		×	1	1	1	1	
	Area		99	99	91	91	
Our approach	Cycles		41	48	49	52	38.68
	Registers		16	17	18	18	
	Muxes		624	704	720	752	
	FU	+	1	1	1	1	
		-	1	1	1	1	
		×	1	1	0	0	
	Area		99	99	91	91	

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_{2^m}. 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

[1] S. Ghandali, B.Alizadeh, Z. Navabi and M. Fujita, "Polynomial Datapath Synthesis and Optimization Based on Vanishing Polynomial over Z_{2^m} and Algebraic Techniques," 10th ACM-IEEE conference on Formal Methods and Models for Co-Design (MEMOCODE), 2012, pp.65-74.

[2] B. Alizadeh and M. Fujita, "Modular Datapath Optimization and Verification Based on Modular-HED," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, pp. 1422-1435, 2010.

[3] B. Alizadeh and M. Fujita, "Improved heuristics for finite word-length polynomial datapath optimization," *ACM- IEEE International Conference on Computer-Aided Design - Digest of Technical Papers (ICCAD)*, 2009, pp. 739-744.

[4] O. Sarbishei, B. Alizadeh and M. Fujita, "Polynomial datapath optimization using partitioning and compensation heuristics," *Design Automation Conference (DAC)*, 2009, pp. 931-936.

[5] B. Alizadeh and M. Fujita, "Modular-HED: A Canonical Decision Diagram for Modular Equivalence Verification of Polynomial Functions," *fifth Workshop on Constraints in Formal Verification (CFV)*, 2008, pp. 22-40.

[6] S. Gopalakrishnan and P. Kalla,"algebraic techniques to enhance common sub-expression elimination for polynomial system synthesis," *Design, Automation & Test in Europe (DATE) Conference*, 2009, pp. 1452 - 1457.

[7] 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(TCAD)*, vol. 25, pp. 2012–2022, 2006.

[8] A. Hosangadi, F. Fallah, and R. Kastner, "Factoring and eliminating commonsub expressions in polynomial expressions," *in Proc., ACM- IEEE International Conference on Computer-Aided Design (ICCAD)*, 2004, pp. 169-174.

[9] Z. CHEN,"On polynomial functions from $Z_{n1}\times Z_{n2}\times\cdots\times Z_{nr}$ to Z_m," *Discrete Math.*, Vol. 162, No. 1–3, pp. 67–76, 1996.

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

[11] MATLAB version 8.2, (2013), (computer software), The MathWorks Inc., Natick, Massachusetts.

[12] P. Coussy, et al., "GAUT: A High-Level Synthesis Tool for DSP Applications," *High-Level Synthesis: From Algorithm to Digital Circuits,* Springer, Berlin, Germany, 2008.

[13] J. von zur Gathen, J. Gutierrez, R. Rubio, "Multivariate Polynomial Decomposition," *Applicable Algebra in Engineering, Communication and Computing*, Vol. 14, No. 1 , pp. 11-31, 2003.

[14] D. Kozen and S. Landau, "Polynomial decomposition algorithms," *J. Symbolic Computation*, Vol. 7, No. 5, pp. 445–456, 1989.

[15] Maple, 2013, http://www.maplesoft.com.

[16] J. Krumm, "Savitzky-Golay Filters for 2D Images," http://homepages.inf.ed.ac.uk/rf/CVonline/LOCAL_COPIES/KRUMMI/SavGol.htm.

[17] V. J. Mathews and G. L. Sicuranza, "Polynomial Signal processing," *Wiley-Interscience*, 2000.

[18] A. Peymandoust and G. DeMicheli, "Application of symbolic computer algebra in high-level data-flow synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 22, pp. 1154–1165, Sep. 2003.