Algebraic Techniques to Enhance Common Sub-expression Elimination for Polynomial System Synthesis

Sivaram Gopalakrishnan [1] and **Priyank Kalla**[2]
[1]Synopsys Inc, Hillsboro, OR
[2]Electrical & Computer Engineering, University of Utah, Salt Lake City, UT

Abstract: *Common sub-expression elimination (CSE) serves as a useful optimization technique in the synthesis of arithmetic datapaths described at RTL. However, CSE has a limited potential for optimization when many common sub-expressions are not exposed. Given a suitable transformation of the polynomial system representation, which exposes many common sub-expressions, subsequent CSE can offer a higher degree of optimization. The objective of this paper is to develop algebraic techniques that perform such a transformation, and present a methodology to integrate it with CSE to further enhance the potential for optimization. In our experiments, we show that this integrated approach outperforms conventional methods in deriving area-efficient hardware implementations of polynomial systems.*

I. Introduction

High-level descriptions of arithmetic datapaths that perform *polynomial computations* over bit-vectors are found in many practical applications, such as in Digital Signal Processing (DSP) for multi-media applications and embedded systems. These polynomial designs are initially specified using behavioural or Register-Transfer-Level (RTL) descriptions, which are subsequently synthesized into hardware using high-level/logic synthesis tools [1]. The growing market for such applications requires sophisticated CAD support for their design, optimization and synthesis.

The general area of high-level synthesis has seen extensive research over the years. Various algorithmic techniques have been devised, and CAD tools have been developed that are quite adept at capturing hardware description language (HDL) models and mapping them into control/data-flow graphs (CDFGs), performing scheduling, resource allocation and sharing, binding, retiming, etc., [2]. However, these tools lack the mathematical wherewithal to perform sophisticated algebraic manipulation for arithmetic datapath-intensive designs. Such designs implement a sequence of ADD, MULT type of algebraic computations over bit-vectors; they are generally modeled at RTL or behavioral-level as *systems of multi-variate polynomials of finite degree* [3] [4]. Hence, there has been increasing interest in exploring the use of algebraic manipulation of polynomial expressions, for RTL synthesis of arithmetic datapaths. Several techniques such as Horner decomposition, factoring with common sub-expression elimination [5], term-rewriting [6], etc., have been proposed. Symbolic computer algebra [3] [4] [7] has also been employed for polynomial datapath optimization. While these methods are useful as stand-alone techniques, they exhibit limited potential for optimization as explained below.

Typically, in a system of polynomials representing an arithmetic datapath, there are many common sub-expressions. In such systems, common sub-expression elimination (CSE) serves as a useful optimization technique, where isomorphic patterns in an arithmetic expression tree are identified, and merged. This prevents the cost of implementing multiple copies of the same expression. However, CSE has a limited potential for optimization if the common expressions are not exposed in the polynomial system representation. Hence, a transformation of the given representation, to expose more common sub-expressions, offers a higher potential for optimization by CSE. The objective of this paper is to develop algebraic techniques to perform this transformation, and present a methodology to integrate it with CSE to achieve a higher degree of optimization.

Motivation: Consider the various decompositions for a system of polynomials P_1, P_2 and P_3, implemented with variables x, y and z, as shown in table I. The direct implementation of this system will require 17 multipliers and 4 adders. To reduce the size of the implementation, a Horner-form decomposition may be used. This implementation requires the use of 15 multipliers and 4 adders. However, a more sophisticated factoring method employing kernel/co-kernel extraction with CSE [5] [8] can further reduce the size of the implementation, to use 12 multipliers and 4 adders. Now, consider the proposed decomposition of the system as shown in table I. This implementation requires only 8 multipliers and 1 adder. Clearly, this is an efficient implementation of the polynomial system. This decomposition achieves a high degree of optimization by analyzing common sub-expressions across multiple polynomials. This is not a trivial task, and is not achieved by any earlier manipulation techniques [5] [8]. Note that d_1 is a good building-block (common sub-expression) for these system of equations. Identifying and factoring out such building-blocks across multiple polynomial datapaths can yield area-efficient hardware implementations.

Contributions: In this paper, we develop techniques to transform the given system of polynomials by employing certain algebraic manipulations. This transformation, subsequently serves as a good candidate for common sub-expression elimination. Our expression manipulation is based on algebraic techniques such as:

[0]Sponsored by NSF grant CCF-0546859.

978-3-9810801-5-5/DATE09 © 2009 EDAA

- Canonization
- Square-free factorization
- Common coefficient extraction
- Factoring with kernel/co-kernel computation
- Algebraic division

We show how the above mentioned algebraic methods are developed/employed. These methods form the foundation of an integrated CSE technique for determining area-efficient implementations of the polynomial system.

TABLE I
Various decompositions for a polynomial system

Original System	Horner-form decomposition
$P_1 = x^2 + 6xy + 9y^2$; $P_2 = 4xy^2 + 12y^3$; $P_3 = 2x^2z + 6xyz$;	$P_1 = x(x + 6y) + 9y^2$; $P_2 = 4xy^2 + 12y^3$; $P_3 = x(2xz + 6yz)$;
Factorization + CSE	**Proposed Decompsoition**
$P_1 = x(x + 6y) + 9y^2$; $P_2 = y^2(4x + 12y)$; $P_3 = xz(2x + 6y)$;	$d_1 = x + 3y$; $P_1 = {d_1}^2$; $P_2 = 4y^2 d_1$; $P_3 = 2xzd_1$;

II. Previous Work

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 algebraic manipulations to reduce the cost of the implementation. For this reason, there has been increasing interest in exploring the use of algebraic methods for RTL synthesis of arithmetic datapaths.

In [9] [10], the authors derive new polynomial models of complex computational blocks by the way of polynomial approximation for efficient synthesis. In [3], 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 [11] [12] [13] for Gröbner bases. Other algebraic transforms have also been explored for efficient hardware synthesis: factoring with common sub-expression elimination [5], exploiting the structure of arithmetic circuits [14], term re-writing [6], data-flow transformations using Taylor Expansion Diagrams (TEDs), etc. Similar algebraic transforms are also applied in the area of code optimization. These include: reducing the height of the operator trees [15], loop expansion, induction variable elimination, etc. A good review of these approaches can be found in [16].

Kernel/co-kernel extraction: Factoring using kernel/co-kernel extraction with common sub-expression elimination [5] is the only technique that integrates algebraic manipulations with CSE. However, this approach has its limitations. Let us understand the general methodology of this approach before describing its limitations. The following terminologies are mostly referred from [5].

Polynomial systems can be manipulated by extracting common expressions by using the kernel/Co-kernel factoring. A **literal** is a variable or a constant. A **cube** is a product of variables raised to a non-negative integer power, with an associated sign. For example, $+acb$, $-5cde$, $-7a^2bd^3$ etc., are cubes. A sum-of-product (SOP) is said to be cube-free if no cube (except "1") divides all the cubes of the SOP. For a polynomial P and a cube c, the expression P/c is a **kernel** if it is cube-free and has atleast two terms. For example, when $P = 4abc - 3a^2b^2c$, the expression $P/abc = 4 - 3ab$ is a **kernel**. The cube that is used to obtain the kernel is the **co-kernel** (abc). This approach has two major limitations:

Coefficient Factoring: Numeric coefficients are treated as literals, not numbers. For example, consider a polynomial $P = 5x^2 + 10y^3 + 15pq$. According to this approach, coefficients $\{5, 10, 15\}$ are also treated as literals like variables $\{x, y, p, q\}$. Since it does not use algebraic division, it cannot determine the following decomposition: $P = 5(x^2 + 2y^3 + 3pq)$.

Symbolic Methods: Polynomials are factored without regard to their algebraic properties. Consider a polynomial $P = x^2 + 2xy + y^2$, which can actually be transformed as $(x + y)^2$. Such a decomposition is also not identified by this kernel/co-kernel factoring approach. The reason for the inability to perform such a decomposition is due to the lack of symbolic computer algebra manipulation.

This paper develops certain algebraic techniques that address these limitations. These techniques, along with kernel/co-kernel factoring, can be seamlessly integrated with CSE to provide an additional degree of optimization. With this integration, we seek to extend the optimization potential offered by the conventional methods.

III. Preliminary Concepts

This section will review some fundamental concepts of factorization and polynomial function manipulation, mostly referred from [17] and [2].

Canonization: Polynomials implemented over specific bit-vector sizes can be represented in a unique canonical form. According to [7], any polynomial representation F for a function f, from $Z_{2^{n_1}} \times Z_{2^{n_2}} \times \dots Z_{2^{n_d}}$ to Z_{2^m}, can be uniquely represented as

$$F = \Sigma_{\mathbf{k}} c_{\mathbf{k}} \mathbf{Y_k} \quad (1)$$

where,

- $\mathbf{k} =< k_1, \dots, k_d >$ for each $k_i = 0, 1, \dots, \mu_i - 1$;
- $c_{\mathbf{k}} \in Z$ such that $0 \le c_{\mathbf{k}} < \frac{2^m}{gcd(2^m, \prod_{i=1}^{d} k_i!)}$.

In Eqn.(1), $\mathbf{Y_k}$ is represented as

$$\begin{aligned} \mathbf{Y_k(x)} &= \prod_{i=1}^{d} Y_{k_i}(x_i) \\ &= Y_{k_1}(x_1) \cdot Y_{k_2}(x_2) \cdots Y_{k_d}(x_d) \quad (2) \end{aligned}$$

where $Y_k(x)$ is a falling factorial defined as follows:

Definition III.1: Falling factorials of degree k are defined according to:

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

  $\vdots$
- $Y_k(x) = (x - k + 1) \cdot Y_{k-1}(x)$

Intuitively, this suggests that while having a canonical form representation as in Eqn.(1), it is possible to find common $Y_{k_i}(x_i)$ terms.

For example, consider the following polynomials implemented over $Z_{2^{16}}$:

$$F = 4x^2y^2 - 4x^2y - 4xy^2 + 4xy + 5z^2x - 5zx \quad (3)$$

$$G = 7x^2z^2 - 7x^2z - 7xz^2 + 7zx + 3y^2x - 3yx \quad (4)$$

Using the canonical form representation, we get:

$$F = 4Y_2(x)Y_2(y) + 5Y_2(z)Y_1(x) \quad (5)$$

$$G = 7Y_2(x)Y_2(z) + 3Y_2(y)Y_1(x) \quad (6)$$

Such a representation exposes many common terms in $Y_{k_i}(x_i)$. These terms, subsequently serve as a good basis for common subexpression elimination.

Definition III.2: **Square-free polynomial:** Let F be a field or an integral domain Z. A polynomial u in $F[x]$ is a *square-free polynomial* if there is no polynomial v in $F[x]$ with $\deg(v, x) > 0$, such that $v^2 \mid u$.

Although the definition is expressed in terms of a squared factor, it implies that the polynomial does not have a factor of the form v^n with $n \geq 2$.

Example III.1: The polynomial $u_1 = x^2 + 3x + 2 = (x + 1)(x + 2)$ is square-free. However, $u_2 = x^4 + 7x^3 + 18x^2 + 20x + 8 = (x + 1)(x + 2)^2$ is *not* square-free, as v^2 (where $v = x + 2$) divides u_2.

Definition III.3: **Square-free factorization:** A polynomial u in $F[x]$ has a unique factorization

$$u = cs_1{s_2}^2 \cdots {s_m}^m \quad (7)$$

where c is in F and each s_i is monic and square-free with $gcd(s_i, s_j) = 1$ for $i \neq j$. This unique factorization in Eqn. 7 is called square-free factorization of u.

Example III.2: The polynomial $u = 2x^7 - 2x^6 + 24x^5 - 24x^4 + 96x^3 - 96x^2 + 128x - 128$ has a square-free factorization $2(x - 1)(x^2 + 4)^3$ where $c = 2$, $s_1 = x - 1$, $s_2 = 1$, and $s_3 = x^2 + 4$. Note that a square-free factorization may not contain all the powers given in Eqn. 7.

A square-free factorization only involves the square-free factors of a polynomial and leaves the deeper structure that involves the irreducible factors intact.

Example III.3: Using square-free factorization,

$$x^6 - 9x^4 + 24x^2 - 16 = (x^2 - 1)(x^2 - 4)^2, \quad (8)$$

both factors are reducible. This suggests that even after obtaining square-free polynomials, there is a potential for additional factorization. In other words, consider Eqn. 8, where $(x^2 - 1)$ can be further factored as $(x + 1)(x - 1)$, and $(x^2 - 4)^2$ can be factored as $((x + 2)(x - 2))^2$.

IV. Optimization Methods

The limitations of contemporary techniques come from their narrow approach to factorization, relying on single types of factorization, instead of the myriad of optimization techniques available. We propose an integrated approach, to polynomial optimization, to overcome these limitations. This section describes the various optimization techniques that are developed/employed in this paper.

Common Coefficient Extraction: The presence of many coefficient multiplications in polynomial systems increases the area-cost of the hardware implementation. Moreover, existing coefficient factoring techniques [5] are inefficient in their algebraic manipulation capabilities. Therefore, it is our focus to develop a coefficient factoring technique that employs efficient algebraic manipulations and as a result, reduces the number of coefficient multiplications in the given system.

Consider the following polynomial $P_1 = 8x + 16y + 24z$. When coefficient extraction is performed over P_1, it results in three possible transformations, given as follows:

$$P_1 = 2(4x + 8y + 12z) \quad (9)$$

$$P_1 = 4(2x + 4y + 6z) \quad (10)$$

$$P_1 = 8(x + 2y + 3z) \quad (11)$$

From these three transformations, Eqn.(11) extracts the highest common term in P_1. This results in the best transformation (reduced set of operations). A method to determine the highest common coefficient is the GCD computation. Therefore, in this approach, GCD computations are employed to perform common coefficient extraction (CCE) for a system of polynomials. The pseudo-code to perform CCE is shown in Algorithm 1.

CCE($a_1, \cdots, a_n$)
/* ($a_1, \cdots, a_n$) = Coefficients of the given polynomial;*/
for every pair (a_i, a_j) in n **do**
Compute GCD(a_i, a_j);
Ignore GCDs = "1";
if GCD(a_i, a_j) $< a_i$ and GCD(a_i, a_j) $< a_j$ **then**
Ignore the GCDs;
end if
end for
Order the GCDs in decreasing order;
while GCD list is non-empty **do**
Perform the extraction using that order
Store the linear/non-linear blocks created as a result of extraction
Remove GCDs corresponding to extracted terms and update the GCD list
end while

Algorithm 1: Common Coefficient Extraction

Consider the polynomial P_1 computed as

$$P_1 = 8x + 16y + 24z + 15a + 30b + 11 \quad (12)$$

The input to CCE are the coefficients of the given polynomial that are involved in coefficient multiplications. In other words, if there is a coefficient addition in the polynomial, it is not considered while performing CCE. For the example in Eqn.(12), only the coefficients $\{8, 16, 24, 15, 30\}$ are considered and 11 is ignored. The reason is because there is no benefit in extracting this coefficient and a direct implementation is the cheapest in terms of area-cost.

The algorithm then begins by computing the GCDs for every pair-wise combination of the coefficients in the input set. Computing pair-wise GCDs,

$$\begin{aligned} GCD(8, 16) &= 8; \\ GCD(8, 24) &= 8; \\ &\vdots \\ GCD(15, 30) &= 15; \end{aligned} \quad (13)$$

we get the following set $\{8, 8, 1, 2, 8, 1, 2, 1, 6, 15\}$. However, only a subset is generated by ignoring "GCDs = 1" and "GCDs $(a_i, a_j) < (a_i, a_j)$". This subset is generated dynamically. The reason for ignoring these GCDs is that we only want to extract the highest common coefficients that subsequently results in a reduced cost. For example, the GCD$(24, 30) = 6$. However, extracting 6 does not reduce the cost of the sub-expression $24z + 30b$ in Eqn.(12). The entire GCD set resulting from Eqn.(13) is just shown for clarity. The resulting subset is $\{8, 15\}$. This set is then arranged in the decreasing order to get $\{15, 8\}$. The first element is "15". On performing the extraction using "15", the following decomposition is realized:

$$P_1 = 8x + 16y + 24z + 15(a + 2b) \quad (14)$$

This creates a smaller polynomial $(a + 2b)$. It should be noted that this is a linear polynomial. This polynomial is stored and the extraction continues until the GCD list is empty. After CCE, the polynomial decomposition is:

$$P_1 = 8(x + 2y + 3z) + 15(a + 2b) \quad (15)$$

Two linear blocks $(a+2b)$ and $(x+2y+3z)$, are finally obtained. The motivation behind storing these polynomials is that they can serve as potentially good building blocks in the subsequent optimization methods.

Common Cube Extraction: Common cubes need to be extracted that consist of variables from the given polynomial representation. The kernel/co-kernel extraction technique from [5] is quite efficient for this purpose. Therefore, this approach is employed to perform the common cube extraction (consisting of only variables). This technique can also extract coefficients. However, since CCE is a more efficient factoring technique for coefficients, we employ this technique for only extracting variables.

Consider the following system of polynomials.

$$\begin{aligned} P_1 &= x^2y + xyz \\ P_2 &= ab^2c^3 + b^2c^2x \\ P_3 &= axz + x^2z^2b \end{aligned} \quad (16)$$

A kernel/co-kernel cube extraction results in the following representation. (Here, c_k is the co-kernel cube and k is the kernel).

$$\begin{aligned} P_1 &= (xy)_{ck}(x + z)_k \\ P_2 &= (b^2c^2)_{ck}(ac + x)_k \\ P_3 &= (xz)_{ck}(a + xzb)_k \end{aligned} \quad (17)$$

The simpler polynomials resulting from the extraction, are always stored. For the above example, these polynomials are simply the "kernels".

Algebraic Division: This method can potentially lead to a high degree of optimization. The problem essentially lies in identifying a good divisor, which can lead to an efficient decomposition. Given a polynomial $a(x)$, and a set of divisors $(b_i(x))$, $\forall i$ we can perform the division $a(x)/b_i(x)$ and determine if the resulting transformation is optimized for hardware implementation.

Using common coefficient extraction and cube extraction, a large number of linear blocks, that are simpler than the original polynomial, are exposed. These linear blocks can subsequently be used for performing algebraic division.

For example, using cube extraction the given system in Table I is transformed to:

$$\begin{aligned} P_1 &= x(x + 6y) + 9y^2; \quad P_1 = x^2 + y(6x + 9y) \\ P_2 &= 4y^2(x + 3y); \\ P_3 &= 2xz(x + 3y); \end{aligned} \quad (18)$$

The following linear blocks are now exposed: $\{(x + 6y), (6x+9y), (x+3y)\}$. Using these blocks as divisors, we divide P_1, P_2 and P_3. $(x + 3y)$ serves as a good building-block because it divides all the three polynomials as:

$$\begin{aligned} P_1 &= (x + 3y)^2; \\ P_2 &= 4y^2(x + 3y); \\ P_3 &= 2xz(x + 3y); \end{aligned} \quad (19)$$

Such a transformation to Eqn.(19) is possible only through algebraic division. None of the other expression manipulation techniques can identify this transformation. The motivation behind using the exposed "linear" blocks for division is that

- Linear blocks cannot be decomposed any further, implying that they have to be certainly implemented.
- They also serve as good building-blocks in terms of hardware implementation.

V. Integrated Approach

The overall approach to polynomial system synthesis is presented in this section. We show how we integrate the algebraic methods presented previously with common subexpression elimination. The pseudo-code for the overall integrated approach is presented in Algorithm 2. The algorithm operates as follows:

- The given system of polynomials is initially stored in a list of arrays. Each element in the list represents a polynomial. The elements in the array for each list represent the transformed representations of the polynomial. Figure 1 (a) shows the polynomial data-structure representing the system of four polynomials in its expanded form, canonical form (can), and square-free factored form (sqf).
- The algorithm begins by computing the canonical forms and the square free factored forms, for all the polynomials in the given system. At this stage, the polynomial data-structure looks like in Figure 1 (a).
- Then, the best cost implementation among these representations is chosen, and stored as $P_{iniital}$. The cost is stored as $C_{initial}$.
- Common coefficient extraction (CCE) and common cube extraction (Cub_Ex) are subsequently performed. The linear/non-linear polynomials obtained from these extractions are stored/updated. Also, the resulting transformations for each polynomial are updated in the polynomial data-structure. At this stage, the data-structure looks like in Figure 1 (b). To elaborate further, in this figure, $\{P_1, P_{1a}, P_{1b}, P_{1c}\}$ are various representations of P_1 (as a result of CCE and Cub_Ex), and so on.

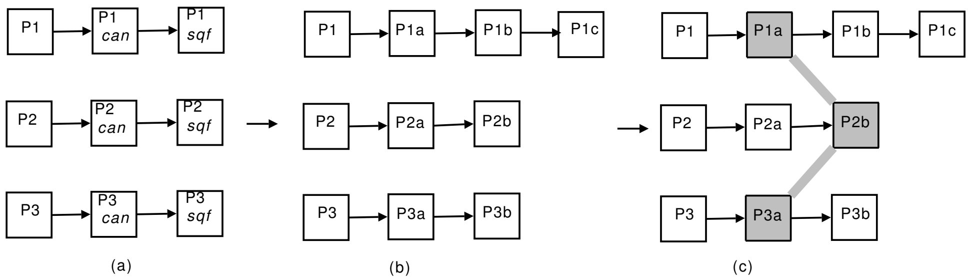

Fig. 1. Polynomial system representations

```
/*Given: (P_1, P_2, ···, P_n) = Polys (P_is) representing the system; Each P_i is a list to store multiple representations of P_i;*/
Poly_Synth(P_1, P_2, ···, P_n)
/*Initial set of Polynomials, P_orig*/
P_orig = ⟨P_1, ···, P_n⟩;
P_can = Canonize(P_orig);
P_sqf = Sqr_free(P_orig);
Initial_cost C_initial = min_cost(P_orig, P_can, P_sqf);
/*The polynomial with cost C_initial is P_initial*/
CCE(P_iniital); Update resulting linear/non-linear polynomials;
/*P_CCE = Polynomial representation after CCE();*/ Update P_is;
Cube_Ex(P_is); Update resulting linear/non-linear polynomials;
/*P_CCE_Cube = Polynomial representation after Cube_Ex();*/ Update P_is;
Linear polynomials are lin_poly =⟨l_1, ···, l_k⟩
for every l_j in lin_poly do
ALG_DIV(P_is);
Update P_is and l_js;
end for
for every combination of P_is (P_comb) representing P_orig do
Cost = CSE(P_comb);
if (Cost < C_initial) then
C_iniital = Cost;
P_final = P_comb;
end if
end for
return P_final;
```

Algorithm 2: Approach to Polynomial System Synthesis

• Using the linear blocks, algebraic division is performed and the polynomial data-structure is further populated, with multiple representations.

• The entire polynomial system can be represented using a list of polynomials, where each element in the list is some representation for each polynomial. For example, $\{P_1, P_{2a}, P_{3b}\}$ is one possible list that represents the entire system (refer Figure 1 (b)). The various lists that represent the entire system are given by:

$$\begin{gathered}\{(P_1,P_2,P_3),\quad (P_1,P_2,P_{3a}),\quad (P_1,P_2,P_{3b}),\\ \vdots\\ (P_{1c},P_{2b},P_3),\quad (P_{1c},P_{2b},P_{3a}),\quad (P_{1c},P_{2b},P_{3b})\}\end{gathered} \tag{20}$$

• From Figure 1 (c), it can be seen that the least-cost implementation of the system,

$$P_{final} = (P_{1a}, P_{2b}, P_{3a}) \tag{21}$$

The Algorithm 2 is explained with the polynomial system presented in table II. Initially, canonization and square-free factorization are performed. In this example, this technique does not result in any decomposition for square-free factorization. For P_3 and P_4, there is a low-cost canonical representation. We then compute the initial cost of the polynomial by using only CSE. In the original system, there are no common sub-expressions. The total cost of the original system is estimated as 51 MULTs and 21 ADDs. Then CCE is performed, resulting in the transformation, as shown in the table II. The linear polynomials obtained are $(x-y)$ and $(x+y)$. The non-linear polynomials are $(x^2+2xy+y^2)$ and $(x^2-2xy+y^2)$. After performing common cube extraction (Cube_Ex()), the additional linear blocks added are $(x+2y)$ and $(x-2y)$. Subsequently, algebraic division is applied using the linear blocks as divisors for all representations of the polynomial system. The final decomposition with CSE leads to an implementation where only the linear blocks $(x+y)$ and $(x-y)$ are used. The representation for the final implementation is shown in the final row of the table II. The total cost of the final implementation is 14 MULTs and 12 ADDs.

TABLE II
ILLUSTRATION OF ALGORITHM 2

Original System
$P_1 = 13x^2 + 26xy + 13y^2 + 7x - 7y + 11;$ $P_2 = 15x^2 - 30xy + 15y^2 + 11x + 11y + 9;$ $P_3 = 5x^3y^2 - 5x^3y - 15x^2y^2 + 15x^2y + 10xy^2 - 10xy + 3z^2;$ $P_4 = 3x^2y^2 - 3x^2y - 3xy^2 + 3xy + z + 1;$
After canonization and CCE
$P_1 = 13(x^2 + 2xy + y^2) + 7(x - y) + 11;$ $P_2 = 15(x^2 - 2xy + y^2) + 11(x + y) + 9;$ $P_3 = 5x(x - 1)(x - 2)y(y - 1) + 3z^2;$ $P_4 = 3x(x - 1)y(y - 1) + z + 1;$
After cube extraction
$P_1 = 13(x(x + 2y) + y^2) + 7(x - y) + 11;$ $P_2 = 15(x(x - 2y) + y^2) + 11(x + y) + 9;$ $P_3 = 5x(x - 1)(x - 2)y(y - 1) + 3z^2;$ $P_4 = 3x(x - 1)y(y - 1) + z + 1;$
Final Decomposition
$d_1 = x + y; d_2 = x - y; d_3 = x(x - 1)y(y - 1)$ $P_1 = 13(d_1{}^2) + 7d_2 + 11; P_2 = 15(d_2{}^2) + 11d_1 + 9;$ $P_3 = 5d_3(x - 2) + 3z^2; P_4 = 3d_3 + z + 1;$

VI. Experiments

The datapath computations are provided as a polynomials system, operating over specific input/output bit-vector sizes. All algebraic manipulations are implemented in Maple [18]; however, factorization routines are available in MATLAB [19]. For common sub-expression elimination, we use the JuanCSE tool available at [8]. Based on

TABLE III

COMPARISON OF PROPOSED METHOD WITH FACTORIZATION/CSE

Systems	Var/Deg/m	# polys	Factorization/CSE		Proposed method		Improvement	
			Area	Delay	Area	Delay	Area %	Delay %
SG_3X2	2/2/16	9	204805	186.6	102386	146.8	50	21.3
SG_4X2	2/2/16	16	449063	211.7	197599	262.8	55.9	-24.1
SG_4X3	2/3/16	16	690208	282.3	557252	328.5	19.2	-16.3
SG_5X2	2/2/16	25	570384	205.6	271729	234.2	52.3	-13.9
SG_5X3	2/3/16	25	1365774	238.1	614955	287.4	54.9	-20.7
Quad	2/2/16	2	36405	118.4	30556	129.7	16	-9.5
Mibench	3/2/8	2	20359	64.8	8433	67.2	58.6	-3.7
MVCS	2/3/16	1	31040	119.1	22214	157.8	28.4	-32

the given decomposition (for each polynomial in the system), the individual blocks are generated using the Synopsys Design Compiler [1]. These units are subsequently used to implement the entire system.

The experiments are performed on a variety of DSP benchmarks. The results are presented in Table III. The first column lists the polynomial systems used for the experiments. The first five benchmarks are Savitzky-Golay filters. These filters are widely used in image-processing applications. The next benchmark is a polynomial system implementing quadratic filters from [20]. The next benchmark is from [21], used in automotive applications. The final benchmark is a multi-variate cosine wavelet used in graphics application from [5]. In the second column, we list the design characteristics: number of variables (bit-vectors), the order (highest degree) and the output bit-vector size (m). Column 3 lists the number of polynomials representing the entire system. Columns 4 and 5 list the implementation area and delay of the polynomial system, implemented using Factorization + common sub-expression elimination, respectively. Columns 6 and 7 list the implementation area and delay of the polynomial system, implemented using our proposed method, respectively. Columns 8 and 9 list the improvement in the implementation area and delay using our polynomial decomposition technique, respectively. Considering all the benchmarks, we show an average improvement in the actual implementation area of approximately **42%**.

VII. CONCLUSIONS

This paper presents a synthesis approach for arithmetic datapaths implemented using a system of polynomials. We develop algebraic techniques that efficiently factor coefficients and cubes from the polynomial system resulting in the generation of linear blocks. Using these blocks as divisors, we perform algebraic division, resulting in a decomposition of the polynomial system. Our decomposition exposes more common terms which can be identified by CSE, leading to a more efficient implementation. Experimental results demonstrate significant area savings using our approach as compared against contemporary datapath synthesis techniques.

REFERENCES

[1] "Synopsys Design Compiler and DesignWare library", Available at htpp://www.synopsys.com.

[2] G. DeMicheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 94.

[3] 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.

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

[5] 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.

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

[7] Sivaram Gopalakrishnan and Priyank Kalla, "Optimization of polynomial datapaths using finite ring algebra", *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, pp. 49, 2007.

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

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

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

[11] 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.

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

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

[14] 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.

[15] A. Nicolau and R. Potasman, "Incremental tree-height reduction for high-level synthesis", *in Proc. DAC*, 1991.

[16] G. DeMicheli and M. Sami, *Hardware/Software Co-Design*, Kluwer Academic Publishers, 96.

[17] J. Cohen, *Computer Algebra and Symbolic Computation*, A. K. Peters, 2003.

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

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

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

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