# Contention Resolution—A New Approach to Versatile Subexpressions Sharing in Multiple Constant Multiplications

Fei Xu, Chip-Hong Chang, *Senior Member, IEEE*, and Ching-Chuen Jong

***Abstract*—Multiple constant multiplications (MCM) have been a core operation in many digital signal processing applications. In this paper, an efficient generalized contention resolution algorithm (CRA) is proposed to eliminate three broad categories of reusable common subexpressions in MCM. The idea is to revert a precedential decision of suboptimal common subexpressions by a localized cost function evaluation when there is a conflict between two competitive subexpressions. The proposed derivatives of the basic CRA are versatile in that they are capable of satisfying search for both intra- and intercoefficient subexpressions, in any legitimate composition of horizontal, vertical and oblique subexpressions. As the algorithms expand the common subexpressions to higher-weight only when there is cost saving, the logic depth can be controlled by constraining the weights of the subexpressions. The variants of CRA follow an important tenet of good heuristic that significant improvement in the solution quality is attained with increased problem size but the computational time remains well bounded. Experimental results with both benchmark filters and randomly generated coefficient sets are analyzed and compared with a number of well known common subexpression elimination methods to demonstrate the effectiveness and efficiency of our proposed approach.**

***Index Terms*—Canonical signed digit (CSD), common subexpression elimination (CSE), finite-impulse response (FIR) filter, multiple constant multiplication (MCM).**

## I. Introduction

EFFICIENT multiplierless realization of digital filters with constant fixed point coefficients has been an area of pervasive research interest due to its widespread applications [1], [3]–[5], [7]–[9], [15]–[26]. The trend towards increasing sampling rates and resolutions of analog to digital converters has tightened the design constraints of digital frondend of wireless communication functions [6], making application-specific digital filters preferable to programmable filters implemented on digital signal processor core. One of the popular application-specific filter structures is the transposed direct form where the input is fed to all the coefficient multipliers in parallel and the regularity of the linear accumulation makes it highly amenable to implementation through a silicon compiler [2]. The core operation of the transposed direct form finite-impulse response (FIR) filter can be modeled as a multiple constant multiplications (MCM) problem [17], [19], and its area-time optimization has often been accomplished through the substitution of multiplications with a reduced number of shift-and-add operations.

A common denominator in methods used for reducing the number of adders (subtractors) of MCM block is the common subexpression elimination (CSE) [8], [17]–[19]. These algorithms search for the most frequently occurred common subexpressions and then maximize the reuse of the products of the input and common subexpressions. In the graph-dependence (GD)-based algorithms [1], [3]–[5], partial sums are symbolically encapsulated in the nodes of the graph and the shift amounts of the partial sums are annotated on the edges. GD algorithms involve the synthesis of a set of minimal cardinality connected graphs from the unity source to the sinks, which are the coefficients to be synthesized. The fundamentals in GD algorithms play the same role as the common subexpressions in CSE algorithms except that the redundancy detection and elimination are value based. Most CSE algorithms use canonical signed digit (CSD) [7], [8], [17], [18], [22], [23] representation of filter coefficients to detect the frequency of occurrences of bit patterns. On the contrary, GD algorithms use integer representation [1], [3]–[5] that makes no assumption on the number format. Hence, they may generate more possible compositions of the coefficients from the intermediate partial sums (also called fundamentals). However, as the search space enlarges, the already complex problem becomes daunting with the additional dimension of decision trade-offs. A common way of managing this level of complexity is to use pre-computed lookup table containing the best compositions of partial sums over a range of integer numbers. As the lookup table is designed based on minimal logic complexity and the table size has to be constrained, the MCM block optimized by conventional GD algorithms are likely to yield long critical path [24], unless some depth control is instilled in the process [3], [16].

By representing the set of CSD coefficients in a two-dimensional table [8], common subexpressions can be extracted by identifying horizontal (intra-coefficient), vertical and oblique (intercoefficient) groups of identical pattern cells. Due to the computational complexity and the symmetry of linear-phase FIR (LPFIR) filter, the search for redundant computations in MCM block is normally confined to horizontal common subexpressions [7], [8], [17]–[19], [22]. Inter-coefficient common subexpressions, which necessitate the insertion of

Manuscript received May 10, 2005; revised January 26, 2007 and May 16, 2007. This paper was recommended by Associate Editor Z. Wang.

F. Xu is with Xilinx Asia Pacific Pte. Ltd., Singapore (e-mail: fei.xu@pmail.ntu.edu.sg).

C.-H. Chang and C.-C. Jong are with School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: echchang@ntu.edu.sg; eccjong@ntu.edu.sg).

Digital Object Identifier 10.1109/TCSI.2007.913707

delay elements into the generator of common subexpressions, are seldom considered. Hartley [8] was one of the pioneers to exemplify the use of horizontal, vertical and oblique common subexpressions in digital filter design. However, due to the limited computing power of the machine at that time, he restricted the algorithm to seek for maximal sharings of the two most common subexpressions, 101 and $10\overline{1}$. Recently, Jang and Yang [11] showed by means of specific filter example that vertical common subexpressions transcend horizontal common subexpressions in LPFIR filters as adjacent coefficients have similar pattern in the most significant bit (MSB) portion. However, Vinod *et al.* [27] argued that elimination of vertical common subexpressions alone does not guarantee greater hardware savings over the conventional horizontal CSE methods in most practical LPFIR filters. They proposed the extraction of the four most common horizontal subexpressions of hamming weight-2 before the remaining nonzero digits are considered for vertical common subexpressions. None of the algorithms has simultaneously utilized all intra- and intercoefficient common subexpressions, including the oblique common subexpressions and the higher-weight common subexpressions of all sorts to reduce the adder costs of MCM block.

Optimal subexpression sharing algorithms for CSD coefficients based on integer linear programming (ILP) model [7], [25] have high computational complexity. Since exact optimal solution is intractable, many algorithms are heuristic in nature and they [8], [18], [24] use weight-2 subexpressions as the primitive elements for reuse and then search for higher-weight common subexpressions from existing lower-weight common subexpressions. However, this process tends to discriminate the higher-weight subexpressions from being selected as their nonzero digits have already been preoccupied by the lower-weight common subexpressions. On the contrary, some algorithms [17], [19] search for the highest-weight common subexpressions at the outset followed by splitting them into the lower-weight common subexpressions for further sharing. This is to insure that the higher-weight common subexpressions do not always give in to the lower-weight subexpressions when they are overlapped. We note from experimental results that admission of higher-weight common subexpressions does not always lead to the reduction of total adder cost. Some algorithms [8], [19] enhance the optimization by including more factors and enlarging the search scope, but the qualities of the solutions improve only marginally. This is because gradient decent approaches, though efficient, are sensitive to the order of execution. There is no feedback and back tracking to allow efficient resubstitution of lower cost common subexpressions in case of conflicts arising from the nullification of potential subexpression elimination caused by early decisions in the process.

The notion of contention resolution was first introduced by us in [22]. A preliminary contention resolution algorithm (designated as CRAH-2 in this paper) was developed to eliminate only weight-2 horizontal common subexpressions. In this paper, contentions between subexpressions of higher weights are introduced and the contention resolution algorithm (CRA) is generalized to handle horizontal, vertical and oblique common subexpressions of any weight and derivatives of CRA are proposed to trade goals of logic depth (LD) and logic complexity versus the algorithmic efficiency. The proposed algorithms take the CSD coefficients [2], [10], [12]–[14], [23] to reduce the logic complexity of MCM block without escalating its LD. The admissibility graph (AG) is devised to dynamically model the contentions arising in the pursuit of most profitable common subexpressions. The contentions are resolved through differential adder cost appraisal of localized pattern graphs to retain good subexpressions or replace inferior subexpressions in conflict. Weight unconstrained and weight constrained CRAs are proposed and all types of common subexpressions can be exploited simultaneously in the AG for optimization. The effects of sharing intercoefficient common subexpressions are also studied.

## II. Contention Resolutions on AG

To facilitate identification and resolution of contentions, a unique *AG* is proposed in [22]. For completeness we briefly review this data structure and introduce additional terminologies that are useful in explaining the extended contention resolutions presented in this paper.

*Definition 1:* Every vertex $v \in V$ in an AG, $G(V, P)$ respresents a nonzero digit. The vertex set $V$ is a collection of all the nonzero digits of the coefficient set and $V_i \subset V$ is the vertex set for the $i$th coefficient in the subgraph, $G_i$. A signed/unsigned vertex corresponds to the digit $\overline{1}/1$. Each vertex, $v$ is associated with a unique non-negative integer, $\mathrm{index}(v)$, determined by its position in the fixed point coefficient.

*Definition 2:* A weight-2 CSD subexpression, $\mathrm{I\hat{O}I}$, where $\mathrm{I} \in \{1, \overline{1}\}$ and $\hat{\mathrm{O}}$ is a string of zeros, is encoded by four parameters $(\mathrm{sign}, \mathrm{parity}, \mathrm{distance}, \mathrm{index})$. $\mathrm{sign} = 0$ if the leading I is "1" and $\mathrm{sign} = 1$ otherwise; $\mathrm{parity} = 0$ if the leading and trailing I are identical, and 1 if they are different; $\mathrm{distance}$ is the number of zeros in $\hat{\mathrm{O}}$; $\mathrm{index}$ is the position of the leading I in the CSD coefficient where the subexpression is found.

Subexpressions with the same $\mathrm{parity}$ and $\mathrm{distance}$ are common subexpressions and are assigned the same $\mathrm{order}$ number that ranks its frequency of occurrence in the coefficient set. $\mathrm{order} = 1$ for common subexpression with the highest frequency of occurrence.

*Definition 3:* An edge $e = (v_i, v_j), e \in E$ is a connection of two vertices $v_i$ and $v_j$ of its endpoints. An edge, $e = (v_i, v_j)$ forms a weight-2 subexpression with $\mathrm{distance} = |\mathrm{index}(v_i) - \mathrm{index}(v_j)| - 1$. The edge set, $E$ of an AG is a collection of all weight-2 subexpressions of the coefficient set. Two edges incident with a common vertex are said to be *adjacent*. The degree of a vertex $v$, denoted by $deg(v)$, is its number of incident edges.

*Definition 4:* A path is a succession of adjacent edges. A path $\eta$ of weight-$w$ consists of $w$ vertices $(v_1, v_2, \ldots, v_w) \subseteq V$ and $w - 1$ edges $(e_1, e_2, e_3, \ldots, e_{w-1}) \subset E$. A path $\eta$, denoted by $v_1 v_2 \ldots v_w$, is an alternative sequence of vertices and edges, $v_1, e_1, v_2, e_2, \ldots v_{w-1}, e_{w-1}, v_w$ if the endpoints of the path, $\eta$ is $v_1$ and $v_w$, and successive vertices $v_i$ and $v_{i+1}$ are endpoints of the intermediate edge $e_i$.

*Definition 5:* A *precedence edge*, $e_p = (v_i, v_j)$ is used to link two vertices, $v_i, v_j \in V$ to admit a common subexpression and a *contention edge*, $e_c = (v_i, v_j)$ is used to indicate a

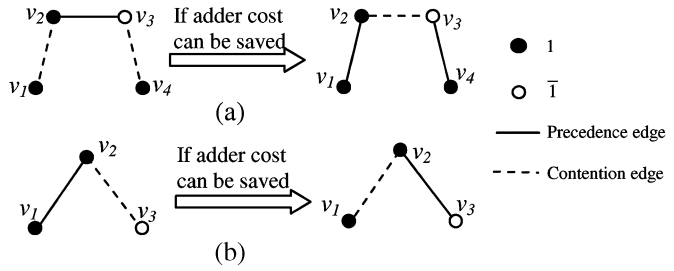



Fig. 1. Basic scenarios of contention resolution.

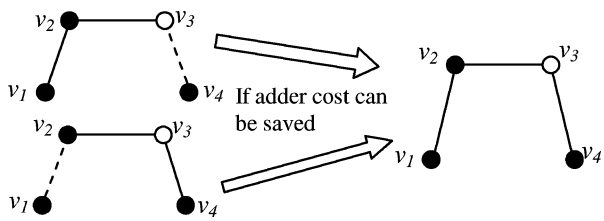



Fig. 2. Generation of weight-4 precedence path.

subexpression that is denied admission as a common subexpression by another subexpression that has already been admitted as common subexpression. The sets of precedence and contention edges are denoted by $E_p$ and $E_c$, respectively. Both precedence and contention edges are annotated with their order numbers upon labeling on the AG. A vertex is $free$ if it is not connected by a precedence edge (or path) and on the contrary, vertices connected by precedence edges (or paths) are *fixed*.

*Definition 6:* Two edges are said to be equivalent if their subexpressions have the same order. Two paths $\eta_1 = v_1 v_2 \dots v_w$ and $\eta_2 = v'_1 v'_2 \dots v'_w$ are said to be equivalent, denoted as $\eta_1 \equiv \eta_2$ if they have the same weight and all their corresponding edges are equivalent, i.e., $\mathrm{order}(e_i) = \mathrm{order}(e'_i) \forall i = 1, 2, \dots, w-1$.

*Definition 7:* A path, $\eta$ is a precedence (or contention) path if all its edges are precedence (or contention) edges. Otherwise $\eta$ is called a hybrid path. Precedence paths in the solution AG are higher weight common subexpressions.

A simple four-vertex AG [22, Fig. 1] has well illustrated some of the above terminologies. An AG is constructed by connecting vertices with precedence or contention edges. A contention is defined as a conflict between two or more different common subexpressions. It can be identified by the existence of a common vertex between two or more edges. When it occurs, only one edge can be a precedence edge and all other edges connected to the same common vertex have to be labeled as contention edges. This is to indicate that selection of one edge into the final common subexpression list will automatically nullify the admissibility of the others.

Two basic contentions are illustrated in Fig. 1. These two types of contention occur frequently. The first scenario occurs when both endpoints of a precedence edge are connected to contention edges, and the other endpoints of the adjacent contention edges are free. The second scenario happens whereby one endpoint of the precedence edge is connected to at least one contention edge.

These two types of contention are most fundamental and other complicated contentions may be reduced to these two types and resolved accordingly. The contention resolution decides if the pattern graph should remain as it is or to promote the contention edge(s) to precedence edge(s) and downgrade the precedence edge to contention edge. A cost metric NAS (acronym of number of adders saved) is defined to evaluate if the status of the precedence edge and contention edge(s) need to be swapped whenever a contention is encountered. NAS can be interpreted as an anticipated minimal number of adders saved due to swapping based on a localized model. Let $e_p = (v_2, v_3)$ be the precedence edge, and $e_{c1} = (v_1, v_2)$ and $e_{c2} = (v_3, v_4)$ be the two contention edges of the first type [see Fig. 1(a)]. Assume that a coefficient is represented by a simple pattern graph of only these four vertices, $v_1$, $v_2$, $v_3$ and $v_4$. If $e_p$ is chosen as the final common subexpression, two adders are required to realize the coefficient for the two free vertices $v_1$ and $v_4$. On the other hand, if $e_{c1}$ and $e_{c2}$ replace $e_p$ to be the final subexpression, there is no free vertex but an adder is required to sum the two subexpressions. Therefore, we set $\mathrm{NAS} = 1$ initially for this simplistic case and adjust the cost by considering other edges in $E_P$ and $E_C$. If there exists equivalent precedence edges of $e_{c1}$ and $e_{c2}$, i.e., $\exists e_{p1}, e_{p2} \in E_P$ with $\mathrm{order}(e_{p1}) = \mathrm{order}(e_{c1})$ and $\mathrm{order}(e_{p2}) = \mathrm{order}(e_{c2})$, then the adders required for the partial sums of $e_{c1}$ and $e_{c2}$ have already been allocated elsewhere. NAS remains unchanged at this stage as no additional adder is incurred. Otherwise, the number of equivalent precedence edges of $e_{c1}$, $e_{c2}$ and $e_p$ are independently examined. If there is no edge in $E_P$ equivalent to $e_{c1}$, then one adder is required to realize this subexpression and $\mathrm{NAS} = \mathrm{NAS} - 1$. The same evaluation is carried out for $e_{c2}$. If there is no other equivalent precedence edge of $e_p$, then $\mathrm{NAS} = \mathrm{NAS} + 1$ because we assume that there are at least one precedence edges equivalent to $e_p$ at the outset. If the final value of NAS is greater than zero, it will be profitable to change the status of the edges from the left pattern graph to the right pattern graph of Fig. 1.

As higher weight subexpressions evolve, the notion of contention resolution needs to be extended to include precedence path. The generation of a precedence path is by itself a consequence of contention resolution. A weight-$n$ precedence path will be generated from a weight-$(n-1)$ precedence path and an adjacent edge if it is profitable. Fig. 2 shows two possible generations of a weight-4 precedence path.

Any hybrid path that has potential to become a precedence path is called a candidate path. When it is decided that a candidate path be converted to a precedence path, some adders will be saved. In some situations, more complicated contention resolutions are needed when there are precedence edges emanating from the candidate path, and the resolution may require that the precedence edges be abandoned by changing them to contention edges to avoid conflict. This is illustrated by an example in Fig. 3. The precedence edge $v_3 v_4$ emanating from an endpoint $v_3$ of the candidate path $v_1 v_2 v_3$ is converted to a contention edge if there is adder saved in forming a precedence path $v_1 v_2 v_3$ of weight-3. The calculation of NAS for the candidate paths is more complicated because all equivalent paths corresponding to

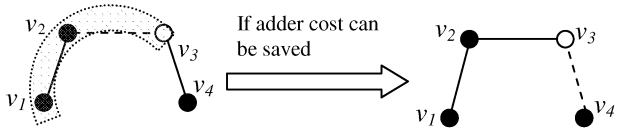



Fig. 3. Contention resolution in path generations.

TABLE I
CLASSIFICATION OF CRAS

| Types of subexpressions | weight-2 | weight-$W$ | weight-$X$ |
|---|---|---|---|
| Horizontal only | CRAH-2 | CRAH-$W$ | CRAH |
| Vertical only | CRAV-2 | CRAV-$W$ | CRAV |
| Horizontal and vertical | CRAHV-2 | CRAHV-$W$ | CRAHV |
| Horizontal, vertical and oblique | CRA-2 | CRA-$W$ | CRA |

higher weight common subexpressions in the entire AG need to be considered simultaneously.

The optimality of CSE algorithms relies on the identification of good common subexpressions. The frequency of occurrence is commonly adopted to appraise the quality of a subexpression but there exist different types of common subexpressions whose frequencies of occurrence are interdependent. Many heuristic CSE algorithms [1], [4], [8], [15], [17]–[19] suffer from a common problem that once a subexpression is identified as the most profitable common subexpression, the decision cannot be reverted. With contention resolution, the number of occurrences of every common subexpression needs to be numerated only once at the beginning and it is used to order the precedence of common subexpressions so that lower order edges corresponding to the most frequently occurred common subexpression are admitted into the AG first. The common subexpressions are orderly admitted as either precedence or contention edges or paths, and contentions are identified by shared (overlapping) vertices and resolved through differential adder cost evaluations of localized pattern graphs described above. The resolution of contention may downgrade certain precedence edges to contention edges depending on the gross effect of conflict. The frequencies of occurrence of the precedence edges or paths are updated dynamically as they are admitted and when they have been changed as a consequence of contention resolution. This will ensure that every admission of a common subexpression into an AG in construction will tend to improve the solution globally. Depending on the number of contentions and the frequency of downgradings, some common subexpressions may eventually end up with one or no precedence edge along the contention resolution process and are no longer common subexpressions. In what follows, we propose several algorithms that apply contention resolution to eliminate various types of common subexpressions based on the AG.

## III. DERIVATIVES OF CRA

By incorporating the considerations of different types and weight constraints of the common subexpressions, variants of CRA algorithms are classified with the abbreviations given in Table I. The weight-2 algorithms aim at mininizing the adder cost with the lowest LD. They only search for quality common subexpressions of weight-2. CRAH-2, proposed in [22], outperforms many CSE algorithms in terms of LD. Some results reported in [3] have revealed that reducing the LD of a circuit could potentially reduce the switching activities due to the shortening of paths via which glitches propagate. Therefore, reducing the LD has an added incentive of minimizing the power-delay product. Since higher-weight subexpressions are evolved from lower-weight subexpressions, the CRAs can be weight-constrained to control the LD. The weight-$W$ algorithms are designed to flexibly limit the weight of the common subexpressions to only $W$. The weight-$X$ ($X$ denotes don't care) algorithms impose no constraint on the weights of the common subexpressions. The algorithms in this class strive for the maximal adder cost reduction by sharing most profitable common subexpressions regardless of their weights. Including all types of subexpressions into consideration will generally but not always enhance the probability of getting the least cost solution. The competitive effects of the simultaneous elimination of all three types of common subexpressions are intricate. Oblique common subexpressions are generally not repeatable in the symmetrical structure, and their elimination will devastate the symmetry. For filters with high number of repeated coefficients, intercoefficient CSEs tend to inhibit the reuse of these coefficients. Different CRA algorithms are provided for different applications of MCM transformation.

The main steps involved in the basic CRA can be summarized as follows.

Step 1) A list of common subexpressions based on their recurrences in the target coefficient set is extracted. The common subexpressions are sorted in descending order of their frequencies of occurrence. The vertices of an unconnected AG corresponding to the nonzero digits of the coefficient set are generated.

Step 2) The first common subexpression is removed from the sorted list. All its equivalent edges are labeled as either precedence edges or contention edges onto the AG.

Step 3) If there is any contention, $\mathrm{NAS}$ is evaluated to resolve the contention.

Step 4) Steps 2 and 3 are repeated until the common subexpressions in the list have been exhausted.

Step 1 is the preparation step and is evoked only once. Let $L$ be the set of unique CSD coefficients. A list of all possible common subexpressions of weight-2 is generated from $L$. For every subexpression, its frequency of occurrence in $L$ is computed. In computing the frequency, we do not differentiate the sign of the subexpressions. Equivalent overlapping subexpressions (e.g., 101 of 10101) are counted only once. Let S denotes the ordered list of distinct subexpressions sorted in descending order of frequency. When there is a tie, the subexpressions are further sorted in the order of horizontal, vertical and oblique types, and for the same type, in ascending order of $\mathrm{distance}$. Subexpressions of unity frequency are excluded from $S$. A unique order number is assigned to each distinct subexpression according to their sorted order in the list. Subexpressions of $\mathrm{order} = 1$ have the highest precedence. In this way, no common subexpression will be neglected because of the consideration of overlapping subexpressions. Equivalent common subexpressions are uniquely identified by their order numbers

which remain intact throughout the process but their reuse statistics can be updated upon contention resolution. Once the ordered list $S$ is established, $G_i(V_i, E_i)$, which is the AG for each coefficient $C_i$, $i = 1, 2, \ldots, |L|$, can be constructed. The program goes into Step 2 and starts the contention resolution iteratively. This portion has its diversity according to the types and weight control of common subexpressions of Table I. The general procedure of weight-2 CRA has been described in [22] for CRAH-2. It can be extended to vertical, oblique and any combination of subexpressions by allowing precedence and contention edges to bridge across vertices of two different coefficients. The uniqueness of CRAs for the weight-$X$ and weight-$W$ categories will be further elaborated in the following subsections.

### A. CRAs for Weight-$X$ Subexpressions

Compared with weight-2 common subexpressions, elimination of higher-weight common subexpressions offers greater potential savings in adder cost. In weight-$X$ CRAs, although there is no weight limit imposed on the common subexpressions, precedence paths are only formed when there is reduction in adder cost. The weight-$X$ algorithms comprise two parts of contention resolutions. In the first part, contentions among edges are resolved and no new path is generated. In the second part, paths are extended from edges and contentions involving paths are resolved. Precedence paths of progressively increased weights are evolved when there is saving in adder cost. Fig. 4 gives the generic pseudocode of the weight-$X$ CRAs.

The lists $L$ and $S$ have the same meanings as before. $\max\_\text{order}$ is the number of distinct order edges. The precedence and contention edges for the construction of AG are stored in the lists, $E_P$ and $E_C$, respectively. In addition, a separate order list of equivalent candidate paths, $P_C$ and an order list of precedence paths, $P_P$ are similarly created. The variable $\max\_\text{order\_path}$ is the number of distinct order paths. $P_C$ and $P_P$ are initially empty and they are updated in the second part on every iteration (line 21). Hybrid paths are generated by construction. A candidate path is then identified by a contention edge adjoined with a precedence edge (or path). It has potential to form higher weight precedence path upon contention resolution. The frequencies of the equivalent paths are accounted each time a path is generated or an existing path is degraded. These paths are sorted dynamically by their weights and frequencies.

Edge contention identification and resolution, including concurrent edge contentions have been explained in [22]. Here the contention path resolution is elaborated. Let $S_{G1}$ and $S_{G2}$ be the subgraphs comprising the candidate path and its emanating paths before and after the conversion. NAS is equal to the difference in adder costs between $S_{G1}$ and $S_{G2}$. We need to consider only those precedence edges and paths emanating from the candidate path that have their attributes changed after the conversion. Each additional contention edge generated by the conversion induces an adder. Breaking an emanating precedence path creates a hybrid path and introduces an adder. Let $\Delta e_p$ be the number of precedence edges in $S_{G1}$ that have been changed to contention edges in $S_{G2}$. Let $n_{p\to h}$ be the number of emanating precedence paths in $S_{G1}$ that have been changed to hybrid paths

```
1: for (each C_i in L, i = 1 .. |L|)
2:     generate G_i from the non-zero digits of C_i;
3: E_P = ϕ; E_C = ϕ;
4: for (i = 1.. max_order) {
5:     if (v_1, v_2 ∈ S_j, are both free, with order(S_j) = i, S_j ∈ S ) {
6:         connect (v_1, v_2) with precedence edge, e_p;
7:         order(e_p) = order(S_j);
8:         E_P = E_P ∪ e_p; }
9:     else {
10:        connect (v_1, v_2) with contention edge, e_c;
11:        order(e_c) = order(S_j);
12:        E_C = E_C ∪ e_c; }
13:    if (contention) {        -- contention resolution required
14:        compute exchange cost, NAS;
15:        if (NAS > 0) {   -- potential saving by exchange
16:            let {e_p}∈E_C be the precedence edges involved;
17:            let {e_c}∈E_C be the contention edges involved;
18:            E_P = E_P ∪ {e_c} −{e_p};
19:            E_C = E_C ∪ {e_p} −{e_c};}}
20:    update P_C, and order for P_C;
21:    for (j = 1 .. max_order_path) {
22:        calculate τ;              --τ is given in (3)
23:        for (each η_i from P_C, with order(η_i) = j) compute NAS(η_i);
24:        sum = Σ_{NAS(η_i)>0} NAS(η_i);
25:        if (sum > τ) {  -- potential saving in path resolution
26:            for (all η_i, with NAS(η_i)> 0) {
27:                E_p' := the set of emanating precedence edges from η_i to be
                          converted;
28:                E_c' := the set of contention edges in η_i before change.
29:                E_P = (E_P ∪ E_c') − E_p';
30:                E_C = (E_C ∪ E_p' ) − E_c';
31:                Update P_P; }}}
32: return E_P, P_P;
```

Fig. 4. Pseudocode of weight-$X$ CRA.

in $S_{G2}$. If $n(e_c)$ is the number of contention edges of the candidate path $\eta$, we have

$$\text{NAS}(\boldsymbol{\eta}) = n(e_c) - \Delta e_p - n_{p\to h}. \tag{1}$$

Equivalent candidate paths with positive NAS cost metric will be converted to precedence paths by promoting their contention edges to precedence edges if their aggregate saving of adders is greater than the cost of generating the final precedence path, $\tau$, i.e.,

$$\sum_{i=1}^{|\eta_c|} \text{NAS}_i > \tau \tag{2}$$

where $\eta_c$ is the set of equivalent candidate paths that have positive NAS. The threshold $\tau$ for promoting the equivalent candidate paths, $\eta = e_1 e_2 \ldots e_{w-1}$ to precedence paths is given by

$$\tau = \sum_{j=1}^{w-1} \tau_j$$

and

$$\begin{cases} \tau_j = 0 & \exists e_p \in E_p, \text{order}(e_p) = \text{order}(e_j) \\ \tau_j = 1 & \forall e_p \in E_p, \text{order}(e_p) \neq \text{order}(e_j) \end{cases}. \tag{3}$$

At the end of the process described in Fig. 4, the precedence edges and paths in $E_P$ and $P_P$ form the common subexpressions. The adder cost can be determined from $E_P$, $P_P$ and $V_{\text{free}}$. The calculation can be divided into two parts: 1) Common subexpression implementation: Each unique common subexpression of weight-2 requires an adder. Let $n(E_P)$ denote the number of distinct common subexpressions in $E_P$, then the cost of realizing the weight-2 common subexpressions, $A_1 = n(E_P)$. The number of adders required to implement a path is not fixed since the weight-2 common subexpressions generated may be reused. Let $P_w$ be the set of distinct order precedence paths of *weight-w*. Further, let $n(\eta_w)$ denote the number of adders required to construct the precedence path, $\eta_w \in P_w$. A subexpression represented by a weight-$w$ path, $\eta_w$ may require from one to $w-1$ adders to construct depending on whether there exists other lower-weight paths equivalent to some segments of $\eta_w$. From experiment, we observe that majority of the precedence paths are of weight-3 and weight-4. For weight-3 subexpressions, originally, two adders are needed to form this type of subexpressions, one for forming an edge from two of the three connected vertices and one for adding this edge to the remaining vertex. There are only three possibilities of generating the edge from a weight-3 path. From the order numbers of the edges, it is easy to identify which formation actually contains a precedence edge that has already been implemented so that an adder can be saved. For weight-4 path, $e_1$ and $e_3$ will be checked to see if there exist precedence edges with the same orders. If so, only one adder is needed for the generation of this path and its LD remains minimal. Otherwise, a more general method is used. For higher-weight precedence paths $\eta_w$ with $w > 3$, we look for $\eta_{w-1}$ from two decompositions of $\eta_w = \eta_{w-1}(1) + v_1$ and $\eta_w = \eta_{w-1}(w) + v_w$. If $\eta_{w-1}(1)$ and $\eta_{w-1}(w)$ have not been generated, their decomposition to $\eta_{w-2}$ continue while $w > 2$. If no lower-weight paths are found, the number of possibilities to realize an adder tree for the required subexpression is formidable. Hence, only two different decompositions are considered, either all odd or all even indexed edges are evaluated for the first level of adders. For a weight-$w$ path $\eta = v_1 v_2 \ldots v_w$, the odd decomposition consists of a set of subexpressions corresponding to all edges, $e_i$ with odd index $i$ while the even decomposition contains edges of even $i$. These two decompositions are evaluated in conjunction with the existing partial sums. The set of edges with the lower adder cost is chosen. Let $n(P_P)$ denote the number of adders used to implement paths in $P_P$, then $A_2 = n(P_P)$. (2) Coefficient implementation: Nonzero digits of each coefficient that are not parts of the final common subexpressions need to be summed with the subexpressions to produce the final coefficient. This cost per coefficient is equal to the number of precedence edges (paths) plus the number of free vertices per subgraph less one. Let $|E_P|$ denote the number of precedence edges, $|P_P|$ denote the number of precedence paths for the coefficient set and $|V_{\text{free}}| = \sum_{v \in V, \deg(v)=0} 1$ is the number of free vertices. The total number of adders required for this part is $A_3 = |E_P| + |P_P| + |V_{\text{free}}| - 1$.

Thus, the total number of adders of the final $AG$ is given by

$$A = A_1 + A_2 + A_3 = n(E_P) + n(P_P) + |E_P| + |P_P| + |V_{\text{free}}| - 1. \quad (4)$$

TABLE II
ORDER LIST OF COMMON SUBEXPRESSIONS OF WEIGHT TWO OF EXAMPLE 1

| Subexpressions | $S$ | Coef. | Frequency | Order |
|---|---|---|---|---|
| $10\bar{1}$ or $\bar{1}01$ | (1, 1, 1, 7), *(0, 1, 1, 5)* | 1 | 3 | 1 |
| | (0, 1, 1, 10) | 2 | | |
| | (0, 1, 1, 8) | 3 | | |
| $10000\bar{1}$ or $\bar{1}00001$ | (1, 1, 4, 10) | 1 | 3 | 2 |
| | (0, 1, 4, 5) | 2 | | |
| | (0, 1, 4, 8) | 3 | | |
| $101$ or $\bar{1}0\bar{1}$ | (1, 0, 1, 2) | 2 | 2 | 3 |
| | (0, 0, 1, 10) | 3 | | |
| $\bar{1}00\bar{1}$ | (1, 0, 2, 10) | 1 | 2 | 4 |
| | (1, 0, 2, 6) | 3 | | |
| $100\bar{1}$ or $\bar{1}001$ | (1, 1, 2, 3) | 1 | 2 | 5 |
| | (1, 1, 2, 8), *(0, 1, 2, 5)* | 2 | | |
| $100001$ | (0, 0, 4, 5) | 1 | 2 | 6 |
| | (0, 0, 4, 10) | 2 | | |
| $1000000\bar{1}$ or $\bar{1}0000001$ | (1, 1, 6, 7) | 1 | 2 | 7 |
| | (0, 1, 6, 10) | 3 | | |
| $1000000000\bar{1}$ or $\bar{1}0000000001$ | (1, 1, 9, 10) | 1 | 2 | 8 |
| | (0, 1, 9, 10) | 2 | | |

Since minimizing the LD under a fixed minimal number of adders offered by $E_P$, $P_P$ and $V_{\text{free}}$ is itself an optimization problem, especially for vertical and oblique subexpressions, it is not likely to provide a closed form estimation of the exact minimal LD until the implementation of the precedence paths is fixed. The following equation gives the upper bound of the LD from circuits synthesized by CRAH

$$D = \max\left(\max_{i=1}^{|L-1|} D_i + 1, D_L\right)$$
$$D_i = \left\lceil \log_2\left(\frac{V_{\text{free}}}{\max(\text{weight}(\eta_i) - 1)} + \sum_{\eta_i} 1\right)\right\rceil + \max(\text{weight}(\eta_i)) - 1. \quad (5)$$

*Example 1:* A small coefficient set is used to illustrate how CRAH works towards its solution. Consider the coefficient set, $L = \{\bar{1}00\bar{1}010\bar{1}001, 10\bar{1}00100\bar{1}0\bar{1}, 1010\bar{1}00\bar{1}000\}$, the coefficients in $L$ are decomposed into weight-2 subexpressions. The order list of subexpressions, $S$, encoded in (sign, parity, distance, index) is tabulated in Table II. The italic subexpressions are overlapping subexpressions of the same order, which will be labeled as contention edges in the AG, $G$.

The allocation of subexpressions is carried out on the subgraphs starting from order 1 subexpressions. In every iteration, all precedence and contention edges of current order number are labeled in the AG, $G$. There is no contention resolution with positive NAS until the fifth iteration when $\text{order} = 5$. For $\text{order} = 5$, there are two equivalent candidate paths, $v_{11}v_{12}v_{13} \equiv v_{23}v_{24}v_{25}$, as shown in Fig. 5(a).

At this time, the precedence edges of $G$ are recorded as $E_P(G) = \{v_{11}v_{12}, v_{13}v_{14}, v_{21}v_{23}, v_{24}v_{25}, v_{32}v_{33}\}$ while the contention edges as $E_C(G) = \{v_{12}v_{13}, v_{13}v_{15}, v_{14}v_{15}, v_{21}v_{22}, v_{22}v_{23}, v_{23}v_{24}, v_{31}v_{32}, v_{31}v_{33}, v_{33}v_{34}\}$. The candidate paths are $P_C = \{v_{11}v_{12}v_{13}, v_{23}v_{24}v_{25}\}$. If they are promoted to precedence paths, their adjacent precedence edges $v_{13}v_{14}$ and $v_{21}v_{23}$ will be broken into contention edges. Thus, the

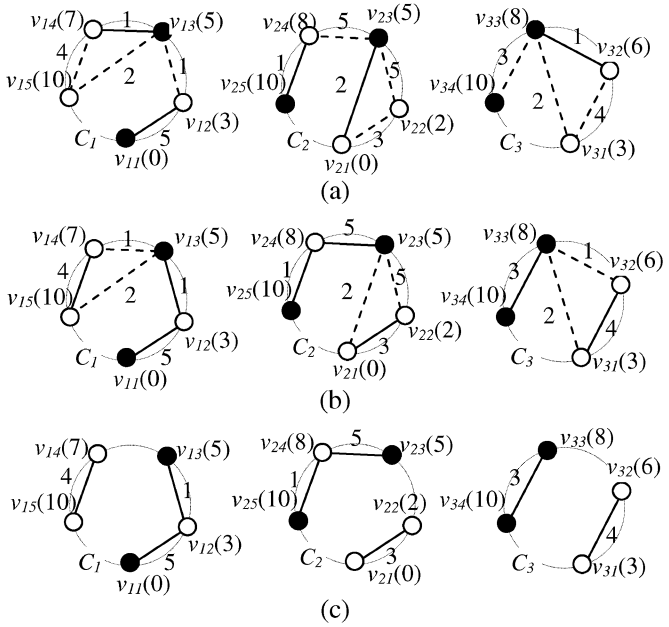



Fig. 5. Execution trace of CRAH for Example 1. (a) Iteration 5 of CRAH. (b) Results of promoting the candidate. (c) Final AG.

NAS evaluation accounts for the attribute changes of $v_{13}v_{14}$ and $v_{21}v_{23}$ as well. Since there is one contention edge in each candidate path, $n(e_c)$ is 1 for both NAS. Due to the conversion of $v_{13}v_{14}$ to contention edge, $\Delta e_p = 1$ for the candidate path in $C_1$. On the other hand, since $v_{21}v_{23}$ is the only order 2 precedence edge, its conversion to contention edge costs no additional adder. The adder required to add $v_{21}$ has already been accounted and $\Delta e_p = 0$ for $v_{23}v_{24}v_{25}$. Since no precedence path has been created yet, $n_{p\to h} = 0$ for both candidate paths. Using (1), $\mathrm{NAS}(v_{11}v_{12}v_{13}) = 1 - 1 - 0 = 0$ and $\mathrm{NAS}(v_{23}v_{24}v_{25}) = 1 - 0 - 0 = 1$. The promotion threshold $\tau = 0$ according to (3). Since $\mathrm{NAS}(v_{11}v_{12}v_{13}) + \mathrm{NAS}(v_{23}v_{24}v_{25}) > \tau$, $v_{11}v_{12}v_{13}$ and $v_{23}v_{24}v_{25}$ are promoted to precedence paths as shown in Fig. 5(b). The attribute changes of $v_{13}v_{14}$ and $v_{21}v_{23}$ make their transitive edges, $v_{14}v_{15}$ and $v_{21}v_{22}$ precedence. The successful path resolutions lead to positive NAS for successful contention resolution of $v_{31}v_{32}v_{33}v_{34}$ and resolution procedure from Line 22 of Fig. 4 is used to swap the status of the precedence edge, $v_{32}v_{33}$ with its adjacent contention edges, $v_{31}v_{32}$ and $v_{33}v_{34}$ in Fig. 5(b). All subexpressions have been labeled and no further contention resolution is required. The final AG is shown in Fig. 5(c).

The total number of adders used can be calculated from (4). Since $n(E_P) = 2$, $n(P_P) = 2$, $|E_P| = 4$, $|P_P| = 2$ and $|V_{\text{free}}| = 0$, $A = 9$ adders is required to implement this set of coefficients. The circuit implementation is shown in Fig. 6. It is noticed that Tap 2 is the critical path and the LD is 3.

It should be noted that due to the following considerations, the already formed precedence path will not be decomposed or broken into shorter path in subsequent iterations. First, from rigorous experimental simulation, it has been found that the probability of saving adder costs and the amount of savings are very low by breaking the established precedence paths. Second, the complexity for evaluating if a precedence path shall be decomposed in this case is high and because of the first reason, most of these extra computations end out to be useless.

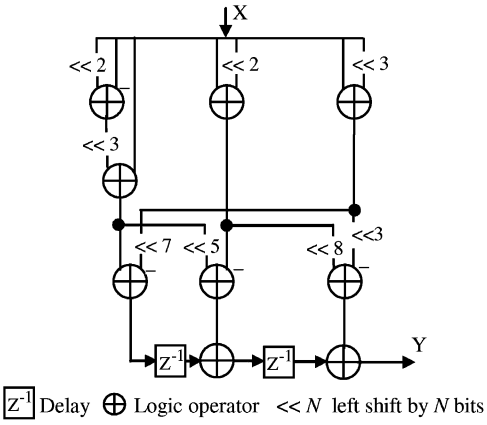



Fig. 6. Circuit implementation.

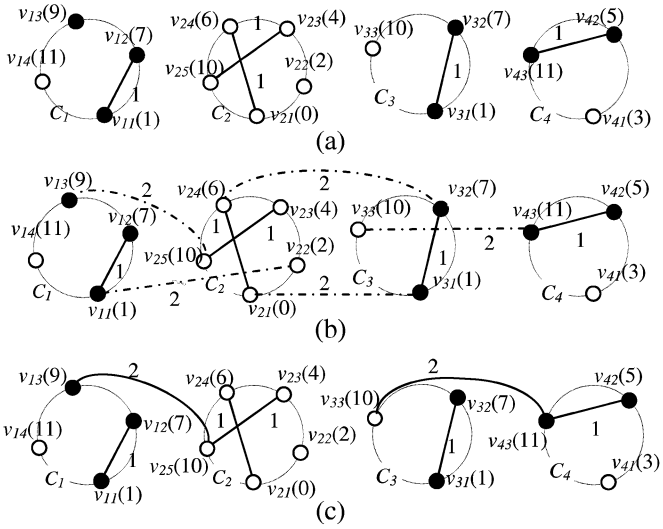



Fig. 7. Exceution traces of CRA for Example 2. (a) AG after labeling of order 1. (b) AG after labeling of order 2 subexpressions. (c) Final AG.

*Example 2:* When vertical and oblique subexpressions are taken into consideration, contention resolution becomes more complicated. The execution of CRA is illustrated on a small coefficient set, $L = \{\bar{1}01010000010, 0\bar{1}000\bar{1}0\bar{1}0\bar{1}0\bar{1}, 0\bar{1}0010000010, 10000010\bar{1}000\}$. First, the order list of subexpressions, $S$ is generated. There are two subexpressions appearing 5 times in the set, one horizontal and one oblique subexpressions given by $x_{hc} = x \ll 6 + x$ and $x_{oc} = x - x[-1] \ll 1$, where $x[-i]$ denotes the value of $x$ after $i$ sample delays and $\ll j$ the left shift of $j$ digits. As mentioned before, horizontal subexpressions have higher precedence. Order 1 edges are labeled in Fig. 7(a). There is no contention at this stage. The precedence edges of the AG, $G$ are recorded as $E_P = \{v_{11}v_{12}, v_{21}v_{24}, v_{23}v_{25}, v_{31}v_{32}, v_{42}v_{43}\}$ and $E_C = \phi$.

For order 2 subexpressions, Fig. 7(b) is obtained after labeling the 5 oblique edges. All five order 2 edges are contention edges but three of them are constituents of candidate paths. The candidate paths are highlighted in Fig. 7(b), $P_C = \{v_{13}v_{25}v_{23}, v_{24}v_{32}v_{31}, v_{33}v_{43}v_{42}\}$.

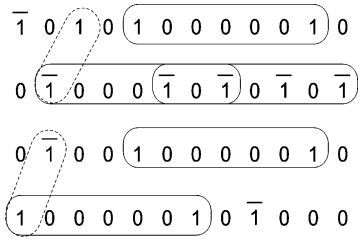


Fig. 8. Solution of CRA.

For the three candidate paths, the NAS is calculated using (1) $\mathrm{NAS}(v_{13}v_{25}v_{23}) = 1-0-0 = 1$, $\mathrm{NAS}(v_{24}v_{32}v_{31}) = 1-1-0 = 0$ and $\mathrm{NAS}(v_{33}v_{43}v_{42}) = 1-0-0 = 1$. The promotion threshold $\tau = 1$ according to (3). Since $\mathrm{NAS}(v_{13}v_{25}v_{23}) + \mathrm{NAS}(v_{24}v_{32}v_{31}) + \mathrm{NAS}(v_{23}v_{24}v_{25}) > \tau$, the candidate paths will be promoted to precedence paths except $v_{24}v_{32}v_{31}$ because no adder will be saved by converting the candidate path $v_{24}v_{32}v_{31}$ to precedence path. The newly generated path is $x_p = x \ll 5 - x[-1] \ll 6 - x[-1] = x \ll 5 - x_{hc}[-1]$. The same set of precedence edges and paths remains unchanged to the end of the process and the final AG is shown is Fig. 7(c).

There are two common subexpressions found in CRA $x_{hc} = x \ll 6 + x$ and $x_p = x \ll 5 - x_{hc}[-1]$. The total number of adders used can be calculated from (4). Since $n(E_P) = 1$, $n(P_P) = 1$, $|E_P| = 3$, $|P_P| = 2$ and $|V_{\mathrm{free}}| = 3$, $A = 9$ adders are required to implement this set of coefficients. The common subexpressions are circled in the coefficient set shown in Fig. 8.

Direct implementation of Example 2 from CSD uses 14 adders. In comparison, 5 adders are saved by CRA solution and the expression, $y = x_{hc} \ll 1 - x_{hc}[-1] + x_{hc}[-2] \ll 1 + x_p \ll 4 - x_p[-2] \ll 5 - x \ll 11 - x[-1] \ll 2 - x[-3] \ll 3$, is more succinct.

To map the AG of Fig. 7(c) to hardware architecture, we need to generate the weight-2 common subexpressions corresponding to the distinct precedence edges first. For precedence edges that bridge across two subgraphs, $G_i$ and $G_j$, $j - i$ unit delay elements are inserted before the adder. Higher-weight common subexpressions are generated with endeavor to use shorter subexpressions. Finally the free vertices are summed with the generated subexpressions to complete the MCM block. The adder tree of the subexpressions and the free vertices are balanced to have minimum depth. Fig. 9 shows the mapping of Fig. 7(c) to an adder tree following the above strategy.

CRAH and CRAHV produce the same solution as shown in Fig. 10(a) since there are not many vertical common subexpressions and they are inferior to the available horizontal common subexpressions in this example.

CRAs can also be applied on other representations such as binary and MSD since they are pattern matching based algorithms. For this example, $C_1$ has three MSD representations and $C_4$ has two MSD representations. Of the six different sets of MSD coefficients. $L_{\mathrm{MSD}} = \{0\bar{1}\bar{1}010000010, 0\bar{1}000\bar{1}0\bar{1}0\bar{1}0\bar{1}, 0\bar{1}00100000101, 10000010\bar{1}000\}$ produces a lower adder cost solution than the CSD coefficient set using CRAH, as shown in Fig. 10(b).

Although a diversity of representations can be explored, the number of combinations to represent a coefficient set with non-canonical representations grows exponentially with the length

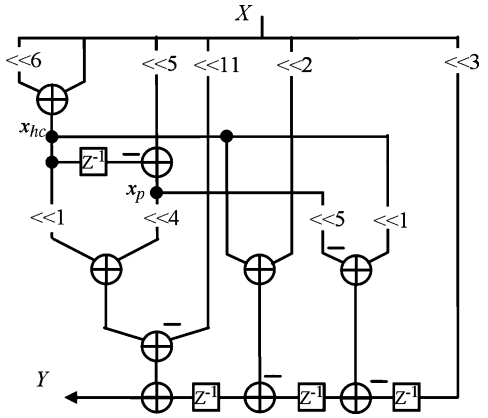



Fig. 9. Mapping of solution AG to adder tree.

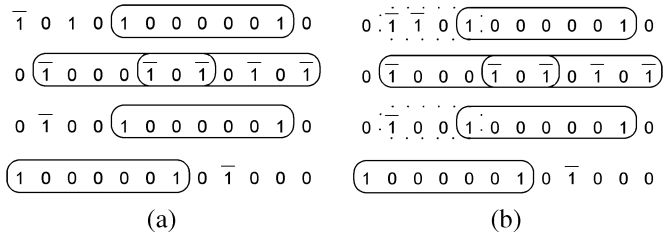


Fig. 10. Solutions obtained by: (a) CRAH and CRAHV and (b) CRAH with MSD representation.

of the filter and the coefficient wordlength. Any CSE algorithm that attempts to find the best solution from different representations is difficult to extricate from the curse of dimensionality. Hence, we will deal with the CSD coefficient set.

### *B. CRAs for Weight-$W$ Subexpressions*

With weight constraint, CRA algorithms for weight-$W$ subexpressions will check the weight of each candidate path and process it only if it is less than $W$. The procedures for weight-$W$ CRAs are similar to those for weight-$X$. These algorithms are useful for evaluating the tentative benefits of higher-weight subexpressions and the hypothetical relationship between MCM block characteristics, LD and different types of subexpressions. The maximum allowable LD, $D_{\max}$ can be used to calculate the corresponding weight constraint, $W_{\max,i}$ to be imposed on each coefficient $i$. The weight limit, $\{W_{\max}\}$ for the whole coefficient set can be used to constrain the subexpression weights of CRAs accordingly in order to limit the LD of their solutions. For horizontal common subexpressions, the LD of the MCM block can be constrained by limiting the maximum weight of the common subexpressions for each coefficient according to Table III.

Table III is obtained by an exhaustive exploration of all possible adder tree topologies of CSD numbers with 2 to 8 nonzero digits. For a coefficient with two or three nonzero digits, the LD is fixed irrespective of how the coefficient is synthesized. Hence, the permissible weight of the precedence path is set equal to the hamming weight of the coefficient. As the hamming weight of the coefficient increases, adder trees of different depths can be used to synthesize the coefficient. If a coefficient is bounded by the LD given in the second column of Table III, the weights of

TABLE III
MAPPING BETWEEN LD AND PATH WEIGHT

| CSD Hamming weight | Logic depth bound | $W_{max}$ |
|---|---|---|
| 2 | 1 | 2 |
| 3 | 2 | 3 |
| 4 | 2 | 2 |
| | 3 | 4 |
| 5 | 3 | 3 |
| | 4 | 5 |
| 6 | 3 | 3 |
| | 4 | 4 |
| | 5 | 6 |
| 7 | 3 | 3 (×1 only) |
| | 4 | 4 |
| | 5 | 5 |
| | 6 | 7 |
| 8 | 3 | 2 |
| | 4 | 4 |
| | 5 | 5 |
| | 6 | 6 |
| | 7 | 8 |

the precedence paths synthesized for that coefficient will not exceed the corresponding numbers given in the last column. For example, according to Table III, if the LD is constrained to 4, coefficients of hamming weight 5 are allowed to have a precedence path of up to weight-5, and for higher hamming weight coefficients, only weight-4 or lower-weight precedence paths are allowed. Coefficients of hamming weights lower than 5 need not be considered as their LDs can never exceed 4. Fig. 11 shows three different adder tree topologies for a CSD coefficient with 7 nonzero digits. In Fig. 11(a), $v_1v_2v_5v_7$ represents a weight-4 precedence path, which is generated from a precedence path $v_1v_2v_5$. Fig. 11(b) shows a weight-4 precedence path, $v_1v_2v_3v_4$ generated from two weight-2 edges, $v_1v_2$ and $v_3v_4$ of the AGs. Although the coefficient of hamming weight 7 can also be synthesized at a lower LD of 3 as in Fig. 11(b), it will refrain the weight-3 precedence path from being reused if one exists and can be expanded to the desired weight-4 precedence path. Therefore, the maximum permissible path weights stipulated in Table III are stringent enough to guarantee the fulfillment of the LD constraints but some slacks are allowed for the reduction of adder costs. The topology of Fig. 11(b) can also be used to illustrate the only exceptional case of Table III corresponding to the synthesis of a weight-7 coefficient under a LD constraint of 3. In this case, only one weight-3 precedence path is allowed (as annotated by "$\times 1$" in Table III) even if there exist two weight-3 precedence paths. If there is no LD constraint, two weight-3 precedence paths, $v_1v_2v_5$ and $v_3v_4v_6$, can be fitted into either adder tree topologies of Fig. 11(a) or (c). Hence, by mapping the LD constraint to the maximum permissible weight of the precedence paths for each coefficient according to its hamming weight in Table III, CRAH optimization can be driven by the LD constraint.

## IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

The most popular application of MCM transformation is FIR filters. In this section, the performances of CRAs are evaluated and compared against practical filters and randomly generated

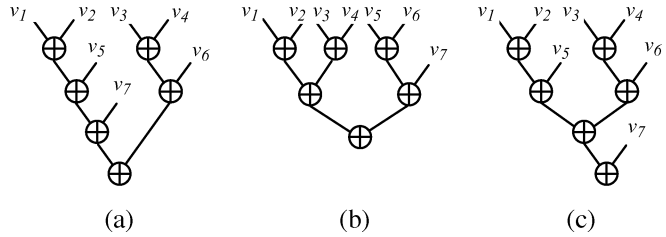



Fig. 11. Adder tree topologies for a weight-7 coefficient.

coefficient sets. Based on the experimental results, the pro and con of these CRAs are analyzed and discussed.

Nine filters of different orders and coefficient wordlengths are used to compare the performances of CRAs. FIR1 and FIR2 are the coefficient sets in Section III illustrating the operations of CRAH and CRA. FIR3 and FIR4 are used in [26] to describe the HCSE and VCSE algorithms. FIR5 [17], [20] is a 25-tap lowpass filter with normalized passband and stopband edge frequencies of 0.15 and 0.25 respectively, and unity ripple weighting factor. The passband and stopband ripples are 0.005 (or $-46.0$ dB). FIR6 [17], [20] is a 60-tap filter with normalized passband and stopband edge frequencies of 0.021 and 0.07, respectively. The passband ripple and stopband attenuation are 0.2 and 60 dB, respectively. FIR7 [13] is 121-tap highpasss filter with normalized stopband and passband edge frequencies of 0.37 and 0.4, passband peak-to-peak ripple of 0.080 dB and peak stopband ripple of $-80.3$ dB. FIR8 [26] is a LPFIR filter in the filter bank channelizer of the Digital Advanced Mobile Phone Systems (D-AMPS) with the sampling rate of 34.02 MHz chosen according to [28]. The channel filter extracts 30-kHz D-AMPS channel from the input signal after downsampling by a factor of 350. The passband and stopband edges are 30 and 30.5 kHz, respectively. The peak passband ripple is chosen to be 0.1 dB. The filter stopband attenuation is $-24$ dB and the number of taps is 200. FIR9 [26] is a channel filter generally designed for the personal digital cellular (PDC) standard. The sampling rate of the wide-band signal is 25.6 MHz, which covers 1024 channels of 25.5-kHz spacing. It has a stopband attenuation of $-30$ dB and the number of taps is 230.

The number of logic operator (LO) and logic depth (LD) of the multiplier blocks of the FIR filters obtained by our proposed CRAH algorithm are compared with the existing algorithms as shown in Table IV. $N$ is the total number of taps and $l$ is the wordlength of the coefficients. Harley, BHM, RAG-$n$, C1, Paško and NRSCSE in the table refer to the Harley's [8], Modified Bull and Horrock's [4], the $n$-dimensional reduced adder graph [4], C1 [3], Paško's [17] and nonrecursive signed CSE [18] algorithms, respectively. Those entries marked "$-$" are results unavailable due to unknown errors generated by executing the original programs from the authors.

Table IV shows that CRAH provides a good balance in the tradeoff of LO and LD. It exhibits the lowest adder cost for more than half of the filters. Comparing with the CSD-based algorithms, which are Hartley, NRSCSE and Pasko, CRAH provides the least LO for almost all the filters. Thanks to the guiding principle of CRAs that the LD is allowed to increase only when there is adder saving, CRAH's solutions achieve near optimal

TABLE IV
COMPARISON OF LO AND LO OF CRAH WITH OTHER METHODS

| Filter | $N$ | $l$ | CSD | | Hartley | | BHM | | RAG-$n$ | | C1 | | Paško | | NRSCSE | | CRAH | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | LO | LD | LO | LD | LO | LD | LO | LD | LO | LD | LO | LD | LO | LD | LO | LD |
| FIR1 | 3 | 11 | 11 | 3 | 8 | 3 | 7 | 4 | 6 | 3 | 6 | 3 | 7 | 3 | 8 | 3 | 7 | 3 |
| FIR2 | 4 | 12 | 11 | 3 | 9 | 3 | 8 | 4 | 7 | 3 | 6 | 3 | 7 | 3 | 7 | 3 | 7 | 3 |
| FIR3 | 8 | 16 | 25 | 3 | 13 | 3 | 12 | 7 | - | - | 12 | 6 | 11 | 6 | 13 | 3 | 11 | 3 |
| FIR4 | 8 | 16 | 18 | 3 | 10 | 3 | 10 | 4 | - | - | - | - | 11 | 4 | 10 | 3 | 10 | 3 |
| FIR5 | 25 | 8 | 11 | 2 | 9 | 2 | 6 | 2 | 6 | 2 | 6 | 2 | 6 | 2 | 6 | 2 | 6 | 2 |
| FIR6 | 60 | 13 | 57 | 2 | 40 | 2 | 26 | 4 | 25 | 3 | 29 | 2 | 32 | 2 | 35 | 2 | 32 | 2 |
| FIR7 | 121 | 14 | 140 | 3 | 84 | 3 | 52 | 7 | 51 | 3 | 54 | 4 | 59 | 4 | 68 | 3 | 56 | 3 |
| FIR8 | 200 | 12 | 145 | 2 | 93 | 3 | 31 | 5 | 31 | 3 | 31 | 2 | 32 | 3 | 38 | 2 | 29 | 3 |
| FIR9 | 230 | 12 | 141 | 3 | 94 | 3 | 27 | 4 | 27 | 3 | 27 | 3 | 27 | 4 | 35 | 3 | 26 | 3 |

LD. Even when it is compared with NRSCSE, which employs only weight-2 subexpressions and is expected to have the minimal LD, CRAH has the same LD in all filters except for FIR8, where it achieves 24% adder cost saving at the expense of one additional adder depth. CRAH has significantly lower adder cost than NRSCSE for long filters. Comparing with the GD algorithms like BHM and C1, CRAH has the shortest LD with comparable LO. CRAH and RAG-$n$ produce comparable LO and LD for the benchmark filters. It is not surprising that the GD algorithms outperform CSD-based algorithms in terms of LO in some cases due to its much larger search space. As opposed to CSD-based algorithms, GD algorithms are not restricted by the fixed number representations. In principle, they have an unlimited number of potential patterns from which better common subexpressions can be selected. The disadvantages are the long computation time and that the statistics of the common subexpressions for the entire coefficient set is not known in advance before the coefficients are fully composed from the fundamentals, which accounts for their poorer results in some cases. GD algorithms generally perform poorer in terms of LD. Even with control optimization on LD, as in C1, the partial sums grow with the composition of the coefficients. The side effect of LD constraint is it has somewhat limited the GD algorithms to achieve better results. Since LO and LD are two conflicting optimization factors, neither CRAs nor other heuristic algorithms can promise the solutions they synthesized have the least LD as well as the lowest LO. Nevertheless, the results obtained by CRAH are asymptotically close to the minimal achievable LO modeled and solved by the ILP algorithms [7], [25] for coefficients represented in CSD form. For example, for the three examples given in [7], CRAH-2 [22] gives exactly the same LO as [25] which uses only weight-2 common subexpressions, and CRAH produces the same results as [7] which considers higher weight common subexpression as well. In short, CRAs promise good overall performance in view of its achievable LD for solutions generated with commensurate optimality of logic complexity.

CRA exploiting intercoefficient common subexpressions is also compared in Table V. As mentioned in Section III, the adder cost of repeated coefficients will be aggravated by the intercoefficient subexpressions. The annihilation of the repetitive patterns by the intercoefficient common subexpressions is more likely to offset the benefits it gained. This effect can be mitigated by a restriction imposed on CRA that no intercoefficient common subexpressions are searched or evaluated in repeated coefficients. For the purpose of demonstrating the complexity reduction by the elimination of common subexpressions of all types and weights, only the adder costs of the half filters of D-AMPS and PDC including the tap accumulators are shown. To provide a thorough evaluation, four different transition bandwidths are chosen for each of D-AMPS and PDC so that $N$ varies from 200 to 940 for D-AMP, and from 230 to 800 for PDC. $l = 12$ and 16 bits are chosen based on popular analog-to-digital converter resolutions.

TABLE V
HALF FILTER ADDER COST COMPARISON OF CRA WITH OTHER METHODS

| Filter | $N$ | $l$ | Hartley | BHM | RAG-$n$ | C1 | Paško | NRSCSE | CRA |
|---|---|---|---|---|---|---|---|---|---|
| D-AMP | 200 | 12 | 146 | 129 | 129 | 129 | 130 | 136 | 128 |
| | | 16 | 257 | 181 | 180 | 181 | 188 | 219 | 181 |
| | 460 | 12 | 258 | 240 | 240 | 240 | 241 | 249 | 206 |
| | | 16 | 431 | 327 | 325 | 327 | 328 | 384 | 327 |
| | 610 | 12 | 301 | 285 | 285 | 285 | 286 | 294 | 241 |
| | | 16 | 510 | 401 | 399 | 401 | 402 | 459 | 396 |
| | 940 | 12 | 342 | 325 | 325 | 325 | 326 | 334 | 263 |
| | | 16 | 665 | 542 | 540 | 542 | 544 | 609 | 497 |
| PDC | 230 | 12 | 157 | 140 | 140 | 140 | 140 | 148 | 135 |
| | | 16 | 270 | 200 | 198 | 200 | 204 | 235 | 197 |
| | 450 | 12 | 242 | 227 | 227 | 227 | 227 | 236 | 188 |
| | | 16 | 425 | 318 | 317 | 318 | 320 | 370 | 312 |
| | 650 | 12 | 274 | 259 | 259 | 259 | 259 | 266 | 208 |
| | | 16 | 525 | 411 | 408 | 411 | 412 | 467 | 388 |
| | 800 | 12 | 289 | 274 | 274 | 274 | 274 | 281 | 200 |
| | | 16 | 573 | 468 | 465 | 468 | 469 | 518 | 433 |

It is clear from Table V that CRA yields the least LO. The savings over other algorithms are prominent as the coefficient wordlengths increases. For some cases, CRA shows great ascendency even with $l = 12$ attributable to the fewer number of nonzero digits in each coefficient and some of them are located in the least significant bits. Horizontal common subexpressions cannot be formed from these weight-1 coefficients, but intercoefficient common subexpressions can be formed by making them a part of the vertical or oblique subexpressions.

For more general comparisons, the performances of CRAH-2, CRAH, CRA-2, CRA, NRSCSE , and Pasko algorithms were tested on randomly generated data. Every data set used in this test was taken from 50 sets of randomly generated coefficients. The word length of the coefficients is fixed at 11. The adder costs normalized by the cost of CSD direct implementation shown in Fig. 12 give an insight into the relative hardware savings achievable by different algorithms. The results show that CRA is capable of generating better solutions with lower number of LOs especially for large filters. The advantage of CRA over CRAH is its effectiveness in harnessing the sharing of intercoefficient common subexpressions. The random simulation results are still indicative of the benefits of sharing intercoefficient common subexpressions for general MCM problems that can be solved by CRA.

Due to the nature of the algorithms, the LD can be indirectly constrained on CRAs if desired. CRA-$W$ will only generate common subexpressions of weight $w \leq W$. To the best of our knowledge, there is only a few LD constrained CSE algorithms [8], [16]. Unlike the LD constrained CSE algorithm of [16], our CRA-$W$ incurs no overhead to the general CRA. Thus, it is of interest to analyze how the adder cost changes as the maximal

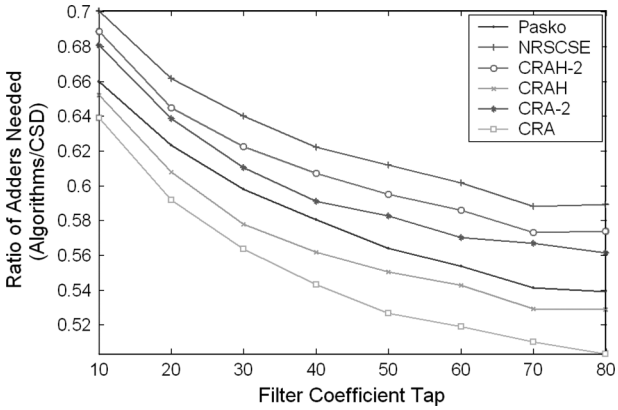



Fig. 12. Comparison of average adder cost.

TABLE VI
SIMULATION RESULTS OF CRAH-$W$ WITH VARYING $W_{\max}$

| $l$ | 11 | | | | 21 | | | | 31 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $N$ | 20 | 40 | 60 | 80 | 20 | 40 | 60 | 80 | 20 | 40 | 60 | 80 |
| $W_{max}=2$ | 14% | 2% | | | | | | | | | | |
| $W_{max}=3$ | 64% | 58% | 42% | 36% | 28% | 18% | 10% | 24% | 28% | 14% | 6% | 18% |
| $W_{max}=4$ | 22% | 40% | 58% | 64% | 70% | 58% | 56% | 46% | 52% | 56% | 40% | 38% |
| $W_{max}=5$ | | | | | 2% | 24% | 32% | 30% | 16% | 30% | 52% | 44% |
| $W_{max}=6$ | | | | | | | 2% | | | | | |
| $W_{max}=7$ | | | | | | | | | 4% | | | |
| %CDP | | 4% | 4% | 2% | 12% | 6% | 18% | 18% | 14% | 10% | 10% | 24% |

permissible weight $(W_{\max})$ of the common subexpressions increases. The results are tabulated in Table VI. 50 randomly generated coefficient sets for each combination of $N$ and $l$ were tested. The number of coefficient sets that have the minimum LO from each $W_{\max}$ is recorded. The percentage of the least adder cost solutions obtained from each $W_{\max}$ is listed for every filter combination. For example, for the coefficient sets of $N = 40$ and $l = 11$, only 2% of the 50 randomly generated coefficient sets have the least adder cost when the maximum LD is constrained to two. This means that for this particular coefficient set, no better result can be achieved by allowing the LD to increase. There exist filter sets that exhibit a critical turning point at some $W_{\max}$ value beyond which the average adder cost will increase by relaxing the LD. This is an interesting phenomenon contrary to conventional belief and the experimental results shown in [17]. The number of coefficient sets that display this critical descend phenomenon (CDP) for each combination of $N$ and $l$ are accounted and its percentage is recorded in the last row of Table VI labeled $\%CDP$. The simulation results of Table VI show the following general trends. For the same word length, the larger the coefficient set, the higher the weight limit to obtain the least adder cost solution whereas for the same number of taps, the longer the word length, the higher the weight limit needed to achieve the best cost solution. The percentage CDP also increases as the order and precision of the filter increases. In Table VI, blank entry denotes zero precentage.

It should be noted that CRAH-$W$ uses the same strategy when building higher-weight subexpressions within the constraint of $W$. That is, higher-weight subexpressions are only created when it leads to a reduction in adder cost. From the %CDP of Table VI, we conjecture that higher-weight subexpressions may not always foster adder cost reduction, as one would expect. The problem may occur when the elimination of higher-weight common subexpressions prevent the more beneficial elimination of some lower-weight common subexpressions whose number of occurrences may be outweighing. Although the ways to obtain higher-weight subexpressions vary from algorithm to algorithm, this local minimum exists, especially for CSE algorithm that starts the search from the highest-weight subexpressions.

TABLE VII
PERCENTAGE OF VCS, OCS AND HCS IN CRA-2

| Order | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 |
|---|---|---|---|---|---|---|---|---|
| *HCS* | 7.74 | 14.3 | 19.72 | 34.08 | 41.82 | 53.64 | 63.76 | 74.58 |
| *VCS+OCS* | 7.46 | 18.6 | 29.34 | 30.44 | 37.88 | 41.9 | 46.84 | 50.12 |
| $\frac{VCS+OCS}{CS=HCS+VCS+OCS}\times 100\%$ | 49.1 | 56.5 | 59.8 | 47.2 | 47.5 | 43.86 | 42.35 | 40.16 |

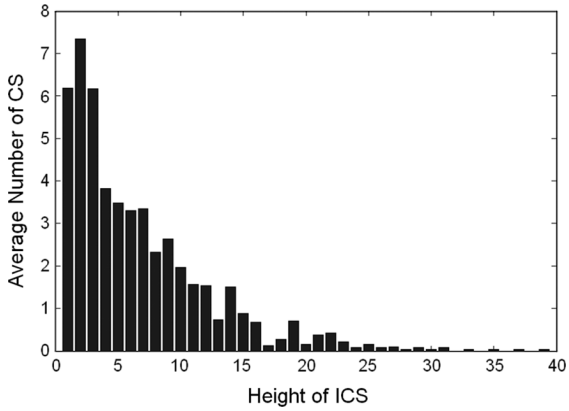



Fig. 13. Distribution of the heights of ICS.

To demonstrate the extent of redundancy that can be exploited from intercoefficient common subexpressions (ICS) in comparison with intra-coefficient common subexpressions, statistic data are collected from randomly generated coefficient set of word length 11 and filter orders ranging from 10 to 80. The statistical results obtained from CRA-2 are shown in Table VII. The total number of vertical and oblique common subexpressions (VCS and OCS) generated by CRA-2 is around half the number of overall common subexpressions.

Fig. 13 presents the distribution of the heights of ICS for coefficient sets of 80 taps and word length 11. The height $i$ and length $j$ defines an ICS, $x[-i] \ll j$ in a 2-D matrix representation. It is evident that most of the selected ICS have height below 20. Although not plotted, it is also found that the distribution of ICS's lengths has a narrow spread centered at 0, most of the ICSs have their lengths fall within the interval $[-2, 2]$. For coefficient sets of word length 11, the longest ICS is 8 bits. This information provides useful hints on how to lower the computation complexity of CRA.

The MATLAB programs of CRAH-2, CRAH, NRSCSE and Pasko algorithms were run on a Pentium IV 1.9-GHz personal computer with 256 MBytes of system memory. Their average computation time in seconds for randomly generated filters with varying number of taps, and fixed word length $(= 11)$ are compared in Fig. 14. CRAH-2 and CRAH are chosen for this comparison because the two algorithms with which they are compared consider only horizontal common subexpressions.

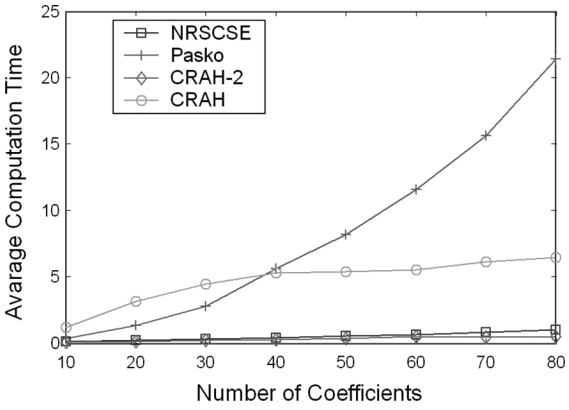



Fig. 14. Comparison on computation times.

CRAH-2 uses the least CPU time in all the test cases. CRAH requires more computation time than Pasko algorithm on small coefficient set. This is because for smaller coefficient sets, more time is spent on the contention path resolution than the search for the most common subexpressions at each iteration. As the coefficient set increases, the computation time of Pasko exacerbates while that of other algorithms remain relatively constant. Pasko algorithm transforms the MCM block into a multiplier free linear transformation of dimension $m \times n$. More potential subexpressions are present with increasing values of $m$ and $n$ due to the increase in filter taps, but the increase in the value of $m$ also causes a drastic increase in computation effort. For CRAs, the number of iterations is dependent on the maximum order number of the subexpression list, which is limited by the word length. As the word length is fixed, the iteration time is relatively stable. That explains why the computational time increases only marginally as the coefficient set grows. CRAs generate common subexpression list only once at the very beginning, which saves a lot of computational effort. This comparison shows that CRAs possess the attribute of good heuristic that achieves quality solutions for large problem size with well-bounded computational time.

## V. Conclusion

In this paper, several new CRAs based on an AG representation of the coefficient set are proposed. The benefit of the variants of CRA is derived from their ability to appraise and substitute the chosen subexpressions when better alternatives emerge. With the core technique of contention resolution, derivatives have been devised to permit sharing of composite types of common subexpression for different MCM characteristics. Among the proposed algorithms, CRAH-2 yields the shortest LD architecture with substantially reduced number of LOs. The most versatile variant of CRA has no constraint on the weights of the common subexpressions and it searches for all horizontal, vertical and oblique subexpressions. It leads to a significant reduction in the number of arithmetic operations required to implement the MCM block. Evidence of the profitability of intercoefficient common subexpression extraction has been demonstrated. The LD driven synthesis option is important as lowering the LD improves the throughput rate and minimizes spurious switching activities through the critical path. Moreover, our algorithms are also computationally efficient. The runtime of CRAH-2 is comparatively lower than NRSCSE and the computation efficiency of CRAH is significantly higher than Pasko algorithm as the problem dimension grows. In short, logic complexity, LD and computational efficiency are conflicting goals to be optimized simultaneously. For any single factor, at least one variant of CRA stand up. The overall performances of CRAH-2 and CRAH are promising in view of its achievable LD for solutions generated with commensurate optimality of logic complexity. CRAs can be extended to deal with other constant representations like MSD and binary coefficients. MSD is promising for its polymorphism but an exhaustive enumeration of all possible MSD coefficient sets to find the best solution is formidable. Our future research will look into the search space reduction for MSD-based CRAs.

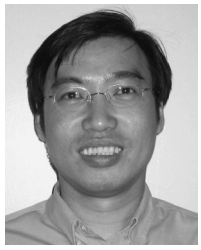
## References

[1] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *Proc. IEEE-G*, vol. 138, no. 3, pp. 401–412, Jun. 1991.
[2] C. L. Chen and A. N. Willson, Jr., "A trellis search algorithm for the design of FIR filters with signed-powers-of-two coefficients," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 1, pp. 29–39, Jan. 1999.
[3] A. G. Dempster, S. S. Demirsoy, and I. Kale, "Designing multiplier blocks with low logic depth," in *Proc. IEEE Int. Symp. Circuits Syst.*, Phoenix, AZ, May 2002, vol. 5, pp. 773–776.
[4] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 9, pp. 569–577, Sep. 1995.
[5] A. G. Dempster and N. P. Murphy, "Efficient interpolators and filter banks using multiplier blocks," *IEEE Trans. Signal Process.*, vol. 48, no. 1, pp. 257–261, Jan. 2000.
[6] J. E. Gunn, K. S. Baron, and W. Ruczczyk, "A low-power DSP core-based software radio architecture," *IEEE J. Select. Areas Commun.*, vol. 17, no. 4, pp. 574–590, Apr. 1999.
[7] O. Gustafsson and L. Wanhammar, "ILP modeling of the common subexpression sharing problem," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, Dubrovnik, Croatia, Sep. 2002, vol. 3, pp. 1171–1174.
[8] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 10, pp. 677–688, Oct. 1996.
[9] R. A. Hawley, B. C. Wong, T. J. Lin, J. Laskowski, and H. Samueli, "Design techniques for silicon compiler implementations of high-speed FIR digital filters," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 656–666, May 1996.
[10] R. M. Hewlitt and E. S. Swartzlantler, "Canonical signed digit representation for FIR digital filters," in *Proc. IEEE Workshop Signal Process. Syst.*, 2000, pp. 416–426.
[11] Y. H. Jang and S. J. Yang, "Low-power CSD linear-phase FIR filter structure using vertical common sub-expression," *Electron. Lett.*, vol. 38, no. 15, pp. 777–779, Jul. 2002.
[12] Y. C. Lim, "Design of discrete-coefficient-value linear-phase FIR filters with optimum normalized peak ripple magnitude," *IEEE Trans. Circuits Syst.*, vol. 37, no. 12, pp. 1480–1486, Dec. 1990.
[13] Y. C. Lim and S. R. Parker, "Discrete coefficient FIR digital filter design based upon an LMS criteria," *IEEE Trans. Circuits Syst.*, vol. CAS-30, pp. 723–739, Oct. 1983.
[14] M. Liu, C. Chen, D. Shin, C. Lin, and S. Jou, "Low-power multiplierless FIR filter synthesizer based on CSD code," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2001, vol. 4, pp. 666–669.
[15] I. C. Park and H. J. Kang, "Digital filter synthesis based on minimal signed digit representation," in *Proc. IEEE Design Automation Conf.*, Las Vegas, NV, Jun. 2001, pp. 468–473.
[16] H. J. Kang and I. C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 8, pp. 770–777, Aug. 2001.
[17] R. Paško, P. Schaumont, V. Derudder, S. Vernalde, and D. Ďuračkoý, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Comput.-Aided Des.*, vol. 18, no. 1, pp. 58–68, Jan. 1999.

[18] M. M. Peiró, E. I. Boemo, and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 49, no. 3, pp. 196–203, Mar. 2002.
[19] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplicaitons: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Comput.-Aided Des.*, vol. 15, no. 2, pp. 151–165, Feb. 1996.
[20] H. Samueli, "An improved search algorithm for the design of multiplier-less FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. 36, no. 7, pp. 1044–1047, Jul. 1989.
[21] H. Samueli, "The design of multiplierless digital data transmission filters with powers-of-two coefficients," in *Proc. IEEE Int. Telecommun. Symp.*, Rio de Janeiro, Brazil, Sep. 1990, pp. 425–429.
[22] F. Xu, C. H. Chang, and C. C. Jong, "Contention resolution algorithm for common subexpression elimination in digital filter design," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 52, no. 10, pp. 695–700, Oct. 2005.
[23] F. Xu, C. H. Chang, and C. C. Jong, "Design of low-complexity FIR filters based on signed-powers-of-two coefficients with reusable common subexpressions," *IEEE Trans. Comput.-Aided Des.*, vol. 26, no. 10, pp. 1898–1907, Oct. 2007.
[24] C. Y. Yao, H. H. Chen, T. F. Lin, C. J. Chien, and C. T. Hsu, "A novel common-subexpression-elimination method for synthesizing fixed-point FIR filters," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 11, pp. 2215–2221, Nov. 2004.
[25] A. Yurdakul and G. Dundar, "Multiplierless realization of linear DSP transforms by using common two-term expressions," *J. VLSI Signal Process.*, vol. 22, pp. 163–172, Sep. 1999.
[26] A. P. Vinod and E. M. Lai, "On the implementation of efficient channel filters forwideband receivers by optimizing common subexpression elimination method," *IEEE Trans. Comput.-Aided Des.*, vol. 24, no. 2, pp. 295–304, Feb. 2005.
[27] A. P. Vinod, E. M. Lai, A. B. Premkuntar, and C. T. Lau, "FIR filter implementation by efficient sharing of horizontal and vertical common subexpressions," *Electron. Lett.*, vol. 39, no. 2, pp. 251–253, Jan. 2003.
[28] K. C. Zangi and R. D. Koilpillai, "Software radio issues in cellular base stations," *IEEE J. Selected Areas Commun.*, vol. 17, no. 4, pp. 561–573, Apr. 1999.

**Fei Xu** received the B.Eng. degree in electronic engineering from Zhe Jiang University, Zhe Jiang, China, in 2002, and the Ph.D. degree from the School of Electrical and Electronic Engineering of Nanyang Technological University, Singapore, in 2007.

She is currently working as a Test Software Engineer at Xilinx Asia Pacific Pte. Ltd, Singapore. Her current research interest includes digital filter design, and computer-aided design for integrated circuit designs.



**Chip-Hong Chang** (S'92–M'98–SM'03) received the B.Eng. (Hons) degree from the National University of Singapore, Singapore, in 1989, and the M.Eng. and Ph.D. degrees from Nanyang Technological University (NTU), Singapore, in 1993 and 1998, respectively.

He served as Technical Consultant in industry prior to joining the School of Electrical and Electronic Engineering, NTU, in 1999, where he is now an Associate Professor. He holds joint appointments at the university as Deputy Director of the Centre for High Performance Embedded Systems (CHiPES) since 2000, and Program Director of the Centre for Integrated Circuits and Systems (CICS) since 2003. His current research interests include low power arithmetic circuits, constrained driven digital signal processing and digital watermarking for IP protection. He has published three book chapters and more than 120 research papers in international refereed journals and conferences.

Dr. Chang is a Fellow of IET, U.K.

**Ching-Chuen Jong** received the B.Sc. (Eng.) degree in electronics with computer science and the Ph.D. degree in electronic engineering from Queen Mary London, U.K., in 1983 and 1988, respectively.

From July 1987 to October 1990, he worked in the area of high-level synthesis of digital systems, first in academic, then in industrial in U.K. In 1991, he joined the Nanyang Technological University, Singapore, as a Faculty Member. He is currently an Associate Professor in the Division of Circuits and Systems, School of Electrical and Electronic Engineering. His technical interests include high-level synthesis, application-specific design and fast-prototyping of digital designs.

Dr. Jong is a Chartered Engineer in the U.K., a member of IET, U.K., and of BCS.