On the Implementation of Efficient Channel Filters for Wideband Receivers by Optimizing Common Subexpression Elimination Methods

A. P. Vinod, *Member, IEEE* and Edmund M-K.Lai, *Senior Member, IEEE*

***Abstract*—The most computationally intensive part of a wideband receiver is the channelizer, which extracts individual radio channels from the output of the ADC. The computational complexity of Linear Phase Finite Impulse Response (LPFIR) filters employed in the channelizer is dominated by the number of adders (subtractors) used in the implementation of the multipliers. Common Subexpression Elimination (CSE) has been proposed as an efficient method to minimize the number of adders in LPFIR filters. In this paper, two methods are proposed to efficiently implement the channel filters in a wideband receiver by optimizing CSE. We exploit the fact that significant amount of redundant multiplications exist in the filter bank channelizer as it extracts multiple narrowband channels from the wideband signal. By forming three and four nonzero-bit super-subexpression utilizing redundant identical shifts that exist between a two-nonzero-bit Common Subexpression (CS) and a third nonzero bit, or between two nonzero-bit CS, we show that the number of adders to implement the channel filters can be reduced considerably. Furthermore, the complexity of adders is analyzed and design examples of the channel filters employed in the Digital Advanced Mobile Phone System (D-AMPS) and the Personal Digital Cellular (PDC) channelizers show that the proposed methods offer considerable reduction in full adders when compared to conventional CSE methods.**

***Index Terms*— Adder complexity, Channelizer, Common subexpression elimination, Linear phase finite impulse response filters.**

I. Introduction

DIGITAL filters employed in the channelizer of a wideband receiver, which extracts several narrowband channels from a wideband signal, present a hardware design challenge [1]. LPFIR filters implemented with high-speed and low-power are required in channelizers. Although programmable filters based on digital signal processing cores offer the advantage of flexibility, they are not suitable for wideband receiver applications that demand high throughput and low-power consumption. Therefore, application specific digital filters are frequently adopted to meet the constraints of performance and power consumption in such applications. However, these filters employ a large number of multipliers that lead to excessive area and power consumption even if they are implemented in full custom integrated circuits. Therefore, the problem of implementing digital filters with small area and low-power consumption has received a great attention in the last decade. Early works have focused on replacing multiplications by decomposing them into simple operations such as addition, subtraction and shifts. Hence, the algorithms that minimize the complexity of multiplication in LPFIR filters focus on reducing the number of adders needed to implement the multipliers.

The number of adders required to implement a multiplier is proportional to the number of nonzero digits present in the filter coefficients. To reduce the complexity, the coefficients can be restricted to powers-of-two (binary) or expressed in Canonic Signed Digit (CSD) representation. On the average, the CSD representation offers a reduction of 33% of nonzero digits compared with the binary representation. Multiple Constant Multiplication (MCM) is a transformation closely related to the widely used substitution of multiplications with constants by shifts and additions. While the latter considers multiplication of only one constant at a time, the MCM considers multiplication of one variable with multiple constants. CSE tackles the MCM problem by minimizing the number of additions through extracting common parts among the constants represented in CSD [2]-[10]. In general, these methods eliminate redundant computations in multiplier blocks by employing the most common subexpressions consisting of two nonzero-bits. In this paper, we show that conventional HCSE and VCSE methods using two nonzero-bit CS can be optimized to form three and four nonzero-bit Super-Subexpression (SS) by exploiting redundant identical shifts among them. The proposed techniques offer considerable reduction in implementing LPFIR filters employed in the channelizer of a wideband receiver where SS among the coefficients of several filters are utilized.

The rest of this paper is organized as follows. In section 2, the HCSE algorithm used to implement MCM in LPFIR filters is reviewed and its application in channelizers is discussed. The complexity of implementation is analyzed in terms of full adders required for each adder. A Horizontal Super-Subexpression Elimination (HSSE) algorithm by optimizing the HCSE method is presented in section 3. In section 4, a Vertical Super-Subexpression Elimination (VSSE) is presented by optimizing the VCSE algorithm. We relate our

Manuscript received June 27, 2003.

The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (phone: 65-67906258; fax: 65-67926559; e-mail: asvinod@ntu.edu.sg; asmklai@ntu.edu.sg).

method to high level synthesis methods in section 5. The implementation of channel filters for the D-AMPS and the PDC cellular standards using proposed HSSE and VSSE techniques are illustrated in section 6. We also provide comparison of hardware reduction achieved employing the proposed methods with that in conventional CSE methods. Section 7 provides our conclusions.

II. Common Subexpression Elimination

A. The HCSE Approach

The idea of HCSE can be demonstrated on a LPFIR filter design example shown in Fig. 1. The function of the Multiplier Block (MB) shown in Fig. 1 is to compute the sum of partial products obtained when the input signal (x) is convolved with the filter coefficients (h_i).

Definition 1 (Multiplier block adders): The adders used in the MB to compute the sum of partial products formed when x is multiplied with h_i are called Multiplier Block Adders (MBA).

Definition 2 (Structural adders): The inter-tap adders used to compute the sum of convolved signals (shown between each delay stage) are called Structural Adders (SA). The number of structural adders in a filter structure is same as that of the number of distinct delay stages.

If T_{mba} and T_{sa} represent the numbers of MBA and SA respectively, the total number of adders required to implement the filter, (T_a), is

$$T_a = T_{mba} + T_{sa} \tag{1}$$

The coefficients are represented using CSD. The number $b_0.b_1b_2...b_{N-1}$ is said to be in CSD format if each b_i is 0, +1, or –1 and no two consecutive b_i are nonzero [3]. In conventional implementation using shifts and adds, the output of the filter can be expressed as:

$$y_{k-1} = x >> 6 + x >> 8, \text{ and}$$
$$y_k = x >> 1 + x >> 3 + x >> 6 + x >> 8 \tag{2}$$

where x is the input signal and '>>' represents shift right operation. (The output y_{k-1} is obtained from $x.(h_{k-1})$, i.e., $x.(2^{-6} + 2^{-8})$, which is represented as $x >> 6 + x >> 8$). In Fig. 1, the numbers adjacent to the data path represent the number of bit-wise right shifts. It requires four adders (MBA) to obtain the output expression (2) as shown in Fig. 1(b). The goal of CSE is to identify multiple occurrences of identical bit patterns that are present in the coefficient set. Since the computation of multiple identical expressions needs to be implemented only once, the resources necessary for these operations can be shared. The pattern [1 0 1] in the example in Fig. 1 is present thrice, which can be expressed as a Horizontal Common Subexpression (HCS),

$$x_1 = x + x >> 2 \tag{3}$$

Using the HCS (3), the output of the filter can be expressed as

$$y_{k-1} = x_1 >> 6 \text{ and } y_k = x_1 >> 1 + x_1 >> 6 \tag{4}$$

Hence, an optimized structure shown in Fig. 1(c) that requires two adders less than the original structure can be implemented. Thus, using HCSE, multiple occurrences of identical bit patterns are eliminated by forming HCS, and the number of adders required to implement the filter structure is minimized.

B. MCM in Filter Bank Channelizers

One of the objectives of this paper is to apply the CSE algorithm to filter bank channelizers (FBC). Channelization in wideband receivers involves the extraction of multiple narrowband channels from a wideband signal using several bandpass filters, called *channel filters.* As shown in Fig. 2, the output of each bandpass filter is followed by a mixer, decimator and a sample rate converter before the signal is fed to baseband processing.

The complexity of the FBC is dominated by the complexity of the channel filters since they operate at the highest sampling rate in the system. The channel filters must meet the high-speed/low-power requirements and need sufficiently large number of taps to meet the adjacent channel interference specifications. Efficient realization methods of channel filters are hardly discussed in the literature. We extend the conventional MCM problem proposed for individual LPFIR filters to FBC for multiplication of one variable (wideband signal) with multiple constants (coefficients) of a bank of bandpass filters. The idea is illustrated using Fig. 3. The most frequently occurring CS among the coefficients of *M* channel filters are identified to form a multiplier block. Further optimization of the multiplier block can be achieved using the proposed HSSE and VSSE methods. Exploiting the fact that the amount of CS grows linearly with the number of channels to be extracted, the CSE optimization techniques presented in sections 3 and 4 are applied to efficiently implement the channel filters.

C. Analysis of the HCSE Method

In this section, we discuss the implementation of a LPFIR filter using HCSE and provide an analysis of the issues related to its complexity. An 8-tap LPFIR filter whose coefficients in 16-bit CSD form in Fig. 4 is used as an example to illustrate the HCSE method.

It is well known that LPFIR filters are symmetric since its impulse response (which resembles a sinc function) satisfies the condition:

$$h(n) = h(N-1-n) \tag{5}$$

where *N* is the number of taps (filter length). Thus, only extra $\lfloor N/2 \rfloor$ structural adders are required (floor value considered if *N* is odd) to obtain the filter output corresponding to the symmetric part. If N_b is the number of nonzero bits in the symmetric half coefficient set represented in CSD, it requires $N_b - 1$ adders to obtain $\sum_{i=0}^{\lfloor N/2 \rfloor - 1} x.h_i$. Therefore, the number of adders required to implement the filter is given by (6):

$$(N_b - 1) + \lfloor N/2 \rfloor \tag{6}$$

In this example, $N_b = 29$, and $N = 8$. Hence thirty-two adders would be required to implement the filter without using HCSE. The 2-bit HCS, [1 0 1] and [1 0 -1], shown inside the rectangles in Fig. 4 are given by:

$$x_2 = x_1 + x_1 >> 2 \text{ and } x_3 = x_1 - x_1 >> 2 \quad (7)$$

where x_1 is the input signal. If N_{hs} is the total number of 2-bit HCS in the symmetric half coefficient set and N_{as} is the number of adders required for distinct HCS, the reduction of adders achieved using HCSE is $N_{hs} - N_{as}$. Hence the number of adders required to implement the filter using HCSE can be obtained by modifying (6):

$$(N_b - 1) + \lfloor N/2 \rfloor - (N_{hs} - N_{as}) \quad (8)$$

In this case $N_{hs} = 13$, and $N_{as} = 2$. According to (8), twenty-one adders are required to implement the filter. This offers a reduction of 34% over direct implementation without HCSE.

D. Adder Complexity

All of the CSE techniques presented in literature discuss the reduction of hardware at the adder level to show the efficiency of those methods. However, the complexity of each adder is significant in practical implementations with high-speed/low-power requirements. In this section, we analyze the complexity of the adders, since it determines the actual cost of implementation. An adder that adds two *n*-bit numbers would require *n* full adders (FA) to compute the sum. We consider ripple carry adders (RCA) through out the paper on account of its low power consumption. Even if carry look-ahead adders (CLA) are considered on account of their improved speed, the full adder requirement of CLA is identical to that of RCA (the difference is that CLA will have an extra carry look-ahead logic to reduce the delay at the cost of more power consumption). The area, power, and speed of an adder depend on the value of *n*, which is called the adder width. Efforts to optimize these parameters should focus on minimizing the adder width, i.e., the number of FA. Firstly, we derive the expressions for analyzing the complexity of adders in HCSE optimized filters and then compute the number of FA's required to implement them.

Definition 3 (Nonzero terms): The subexpressions and the nonzero bits other than the subexpressions of a coefficient are termed as its nonzero terms. For example, the two nonzero terms of a coefficient represented in CSD, (0.1010001), are [1 0 1] (CS) and 1 (least significant bit).

Definition 4 (Operands): The input signal shifted corresponding to the positional weights of the nonzero terms of the coefficient form the operands of the adders. For instance, in the case of the coefficient, (0.1010001), the operands are $x_2 >> 1$ and $x_1 >> 7$, where x_1 is the input and $x_2 = x_1 + x_1 >> 2$ is the CS, [1 0 1]. Note that the number of nonzero terms and operands are identical. The number of adders required to compute the output for a coefficient is equal to one less than the number of operands.

Definition 5 (Span): The span is analogous to the wordlength, which is equal to the number of bits of an operand. Considering the above example, if x_1 is an 8-bit quantized signal, the span of the operand, $x_2 >> 1$, is eleven and that of $x_1 >> 7$ is fifteen.

Definition 6 (adder-step): One addition stage in a maximal path of decomposed multiplications is called the adder-step. A multiplication can have different adder-steps, depending on the structure of multiplication.

We employ the high-speed tree structure shown in Fig. 5 to implement the MB. Using the binary tree in Fig. 5, the number of adder-steps, (A_n), required to compute the sum of partial products of n operands (nonzero bits of the coefficient) is given by $2^{A_n} \geq n$. From this, we obtain

$$A_n = \left\lceil \frac{\log_{10}(n)}{\log_{10}(2)} \right\rceil \quad (9)$$

The A_n obtained in (9) is the lowest number of adder-steps (lower bound) possible to achieve in an addition structure since the tree structure considered in our method performs parallel addition. Therefore, our method always results in a minimum adder-step implementation and hence has the lowest delay.

Case I - Odd number of operands:

Consider the coefficient $h(n) = (0.1001010101)$. If x_1 represents the input signal, the output can be expressed as

$$y(n) = x_1 >> 1 + x_2 >> 4 + x_2 >> 8 \quad (10)$$

where $x_2 = x_1 + x_1 >> 2$ is the HCS corresponding to the bit pattern [1 0 1]. In this case, the number of operands is three (odd) and hence two adders are required to compute $y(n)$. If x_1 is represented using 8 bits, the minimum span (neglecting the carry part) of x_2 is 10 and those of the first, the second, and the third operands of (10) are 9, 14, and 18 respectively. For an adder whose operands have spans s_1 and s_2 such that $s_2 > s_1$, the adder width is s_2. There are two possible ways to implement (10) as shown in Fig. 6(a) and (b). In the implementation shown in Fig. 6(a), both adders, A_1 and A_2, have the maximum width of 18. In the case of Fig. 6(b), only A_2 has the maximum width, 18, while the width of A_1 is 14. Hence the implementation of Fig. 6(b) requires fewer FA's than that in Fig. 6(a). Note that the number of adder-steps are identical (two) in Fig. 6(a) and 6(b). Thus, using the minimum FA scheme of Fig. 6(b), addition of three operands would require $(s_2 + s_3)$ FA's, where s_1, s_2, and s_3 represent the spans of the operands in (10) such that $s_3 > s_2 > s_1$.

Filter coefficients in CSD form with wordlengths up to 24-bits are considered here. Since no adjacent bits in CSD are one's, a 24-bit CSD number can have a maximum of 12 nonzero bits and hence at the most twelve nonzero operands could occur in a multiplication. Consider the filter tap shown in Fig. 7 that has an odd number of operands (nine), whose spans are indicated as s_i. The s_i's shown adjacent to the adders represent the adder widths. The total number of FA's required to implement this filter tap is given by the sum of the widths of all adders, i.e., $s_2 + 2s_4 + s_6 + 3s_8 + s_9$.

By extending this minimum adder-step structure to 24-bit CSD coefficients, it can be shown that the number of FA's, N_0, required to compute the output corresponding to a coefficient with n operands can be determined using the expression:

$$N_o = s_2 + a_1 s_3 + 2s_4 + a_3 s_5 + s_6 + a_5 2s_7 + 3s_8 + a_7 s_9 + s_{10} + a_9 2s_{11} + 2s_{12} \quad (11)$$

where s_n is the span of the nth operand and a_i's are equal to zero except $a_{n-2} = 1$. For instance, if 7 operands are present, using (11) we get $N_0 = s_2 + 2s_4 + s_6 + 2s_7$. Expression (11) can be represented in matrix form for easier computation of FA's for any coefficient with n operands ($n \le 11$, since n is odd) as

$$N_0 = U_{H_o}(\lceil n/2 \rceil).S \quad (12)$$

where $U_{H_o}(\lceil k \rceil)$ represents the elements of the $\lceil k \rceil$-th row of the matrix and S is the span vector,

$$U_{H_o} = \begin{bmatrix} 1 & & & & & & & & & \\ 1 & 1 & & & & & & & & \\ 1 & 0 & 2 & 1 & & & & & & \\ 1 & 0 & 2 & 0 & 1 & 2 & & & & \\ 1 & 0 & 2 & 0 & 1 & 0 & 3 & 1 & & \\ 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 2 \end{bmatrix} \text{ and } S = \begin{bmatrix} s_2 \\ s_3 \\ . \\ . \\ . \\ s_n \end{bmatrix}.$$

Case II - Even number of operands:

Consider the coefficient, $h(n) = (0.100101010101)$. The output can be expressed as

$$y(n) = x_1 >> 1 + x_2 >> 4 + x_2 >> 8 + x_1 >> 12 \quad (13)$$

In this case, the number of operands is four (even) and hence three adders are required to compute $y(n)$. The possible addition sequences to obtain (13) are shown in Fig. 8(a) and (b). If the spans of the operands of (13) are s_1, s_2, s_3 and s_4 respectively, the implementation in Fig. 8(a) would require $s_2 + s_3 + s_4 = 14 + 18 + 20 = 52$ FA's. On the other hand, it would require $s_2 + 2s_4 = 14 + 2(20) = 54$ FA's to realize the scheme in Fig. 8(b), which is larger than the former. However, it should be noted that implementation in Fig. 8(b) requires one less adder-step than that of Fig. 8(a) and hence its critical path is shorter. On account of its minimum critical path, we use the structure in Fig. 8 (b) in our method, though its costs a few additional FA's. The addition scheme in Fig. 8(b) can be extended to 24-bit CSD to show that the number of FA's, N_e, required to compute the output corresponding to a coefficient with n operands ($n \le 12$) is given by:

$$N_e = s_2 + 2s_4 + c_0 s_6 + 3s_8 + c_1 s_{10} + 3s_{12} \quad (14)$$

where $c_0 \equiv \begin{cases} 2, \text{for } n = 6 \\ 1, \text{for } n \ne 6 \end{cases}$ and $c_1 \equiv \begin{cases} 2, \text{for } n = 10 \\ 1, \text{for } n \ne 10 \end{cases}$.

For example, if six operands are present (i.e., $n = 6$), it would require $(s_2 + 2s_4 + 2s_6)$ FA's. Using the matrix form, the number of FA's for computing the output of any given coefficient with n operands is given by:

$$N_e = U_{H_e}(n/2).S \quad (15)$$

where S is the span vector as in (12), and $U_{H_e}(k)$ represents the elements of the k-th row of the matrix,

$$U_{H_e} = \begin{bmatrix} 1 & & & & & & & & & \\ 1 & 0 & 2 & & & & & & & \\ 1 & 0 & 2 & 0 & 2 & & & & & \\ 1 & 0 & 2 & 0 & 1 & 0 & 3 & & & \\ 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 2 & \\ 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 0 & 3 \end{bmatrix}.$$

E. Full Adder Requirements in HCSE Method

The number of full adders (MBA) required to compute the partial products for the filter in Fig. 4 can be determined using (12) and (15) for odd and even number of operands respectively. We consider the first two coefficients in Fig. 4 for illustration of FA estimation.

1) Even number of operands: Consider the coefficient, $h(0)$. The expression of the output, $y(0) = x_1.h(0)$, is

$$y(0) = x_2 >> 1 - x_1 >> 5 + x_2 >> 8 + x_3 >> 13 \quad (16)$$

where x_2 and x_3 are given by (7) and x_1 is 16-bit input signal. Note that the spans of x_2 and x_3 are same, i.e., 18. In (16), there are four (even) operands and spans of second (s_2) and fourth operands (s_4) are 21 and 31 respectively. The number of FA's for computing $y(0)$ can be determined using (15):

$$N_e = U_{H_e}(n/2).S = U_{H_e}(4/2).S$$
$$= \begin{bmatrix} 1 & 0 & 2 \end{bmatrix}.\begin{bmatrix} s_2 \\ s_3 \\ s_4 \end{bmatrix} = s_2 + 2s_4 = 83 \quad (17)$$

2) Odd number of operands: Consider $h(1)$, where output $y(1) = x_1.h(1)$ is computed by ([-k] represents a delay of k):

$$x_1[-1] >> 2 - x_3[-1] >> 4 + x_2[-1] >> 8 + x_2[-1] >> 12 - x_1[-1] >> 16 \quad (18)$$

In this case, there is an odd number of operands (five) and $s_2, s_4,$ and s_5 are 22, 30, and 32 respectively. Using (12), the number of FA's for computing $y(1)$ is given by

$$N_o = U_{H_o}(\lceil n/2 \rceil).S = U_{H_o}(\lceil 5/2 \rceil).S$$
$$= \begin{bmatrix} 1 & 0 & 2 & 1 \end{bmatrix}.\begin{bmatrix} s_2 \\ s_3 \\ s_4 \\ s_5 \end{bmatrix} = s_2 + 2s_4 + s_5 = 114 \quad (19)$$

Using this method, the total number of FA's required to compute the partial products of the MBAs of the LPFIR filter in Fig. 4 is 376. In the next section, we present an optimization technique that minimizes the number of FA's.

III. Optimization of HCSE Method

We observe that several 3-bit and 4-bit Horizontal Super-Subexpressions (HSS) can be formed by exploiting identical shifts between an HCS and a nonzero bit or between two HCS, which will eliminate redundant computations of

HCS. While implementing multiplication using shifts and adds, if we could perform addition prior to shift, the adder width can be minimized. Note that in CSE implementations, the adders employed for CS have shorter widths since the shift operations for obtaining the final partial products are performed after the addition at the CS stage. In the proposed horizontal super-subexpression elimination (HSSE) method, shift operations are performed after additions at two stages - first at the HCS stage and then at the HSS stage. Therefore the adders at these two stages have shorter adder widths. Utilizing this fact, we shall now show that considerable reduction of FA's is possible by forming HSS from HCS using the HSSE method.

A. The HSSE Method

The HSSE procedure is as follows. First, the 2-bit HCS are extracted from the coefficient set represented in CSD. These HCS are then examined for multiple occurrences of identical shifts with a nonzero bit or with another HCS within the same coefficient to form 3-bit and 4-bit HSS respectively. Consider the example in Fig. 4, where HCS are given by (7). Note that following multiple bit patterns can be formed:

1) The HCS [101] and –1 with a shift of one unit between them (indicated by the connecting line, 'a') that occur within the coefficient $h(0)$ to form a 3-bit HSS, [1 0 1 0 -1]:

$$x_4 = x_2 - x_1 >> 4 \tag{20}$$

2) The HCS [101] and [10-1] with two shift units between them (indicated by the connecting line, 'b') that occur within the coefficient $h(0)$ to form a 4-bit HSS, [1 0 1 0 0 1 0 -1]:

$$x_5 = x_2 + x_3 >> 5 \tag{21}$$

3) The HCS [–101] and [101] with one shift unit between them (indicated by the connecting line, 'c') that occur within the coefficient $h(1)$ to form a 4-bit HSS, [-1 0 1 0 1 0 1]:

$$x_6 = -x_3 + x_2 >> 4 \tag{22}$$

It may be noted that several HSS in 'shifted and delayed' forms of (20), (21), and (22) occur in the coefficient set. For instance, consider the HSS, [1 0 1 0 0 1 0 -1], given by x_5 (21). The outputs corresponding to x_5 that occur in the coefficients, $h(0), h(2)$, and $h(3)$ are given by $x_5 >> 8$, $x_5[-2] >> 2$, and $x_5[-3] >> 9$ respectively. Thus the output expression can be obtained from x_5 by simple shift and delay operations. Note that no extra adders are required to compute this expression. However the HCSE method would require an extra adder for each subexpression.

We observe that several HSS exist in LPFIR filters, especially in the case where the number of taps is large and the bit precision of implementation is higher (16-bit and higher). We have investigated several examples of LPFIR filters with taps ranging from 100 to 1200 corresponding to different stop-band attenuation specifications. The infinite-precision filter coefficients were generated by the Parks-McClellan FIR filter design program provided by the MATLAB® "remez" function. Filter coefficients represented in CSD form for different wordlengths of 16-bits, 20-bits and 24-bits were considered. From the extensive examples we worked out, it has been observed that among the possible HSS, the 3-bit expressions [1 0 1 0 1], [1 0 1 0 –1], [-1 0 1 0 1], [-1 0 1 0 –1] and their negated versions are the most common HSS. Statistically, these 3-bit expressions form around 70% of all the possible HSS. Hence they account for the major reduction of adders in the proposed method. Employing the HSS (20), (21), and (22), the output of the filter whose coefficients given in Fig. 4 can be expressed as

$$\begin{aligned} &x_4 >> 1 + x_5 >> 8 + x_1[-1] >> 2 + x_6[-1] >> 4 + x_4[-1] >> 12 + \\ &x_5[-2] >> 2 + x_3[-2] >> 13 + x_6[-3] >> 1 + x_5[-3] >> 9 + \\ &x_6[-4] >> 1 + x_5[-4] >> 9 + x_5[-5] >> 2 + x_3[-5] >> 13 + \\ &x_1[-6] >> 2 + x_6[-6] >> 4 + x_4[-6] >> 12 + x_4[-7] >> 1 + \\ &x_5[-7] >> 8 \end{aligned} \tag{23}$$

Fig. 9 shows the filter structure using the HSSE method. Note that only seventeen adders (10 MBA and 7 SA) are required to implement the filter, two for HCS (7), three for HSS (20-22), and twelve for the filter output (23). It may be noted that though seventeen additions are present in (23), using symmetry of LPFIR filters, only twelve adders are sufficient to compute the sum as shown in Fig. 9. This is because, the outputs of adders, A_6, A_8, A_9, and A_{10} can be shared by respective symmetric filter tap pairs as shown in Fig. 9. Note that the sharing of symmetric parts is shown in Fig. 9 using the symbol '@'. Thus, by sharing, we can save one adder each corresponding to A_6, A_9, A_{10}, and two adders corresponding to A_7 and A_8 (sharing the output of A_8 results saving of two adders, A_7 and A_8). Thus the total saving due to sharing is five adders. Hence only twelve adders are required to obtain (23). Therefore, the HSSE implementation requires four adders less than the HCSE implementation. The adder-steps required to compute the partial products in the proposed method is four, which is the same as that of the HCSE method. Thus, both methods have identical critical paths of four adder-steps.

B. Full Adder Requirements

The number of full adders required to compute the partial products of the filter in Fig. 9 can be determined using (12) and (15) for odd and even number of operands respectively. Ten adders (A_1 to A_{10}) are required to compute the partial products. Note that the adders A_3, A_4, and A_5, that compute the HSS part have relatively short adder-width when compared to other adders in subsequent stages. This is because the use of HSS adders allows us to perform most of the 'right shift' operations after addition, which is more efficient than the usual 'shift and add' method. As a result, fewer FA's are required to compute the partial products. Thus, using the HSSE method, only 253 FA's are required to compute the partial products of the MBAs of the LPFIR filter in Fig. 4. This is a reduction of 32.7% over the HCSE method. Design examples of implementing channel filters of a wideband receiver using the HSSE method is discussed in section 6.

IV. Optimization of VCSE Method

In the VCSE method [8], the fact that many vertical common

subexpressions (VCS) exist in an LPFIR filter since the adjacent coefficients have similar patterns in the MSB part is utilized for reducing adders. In this section, we show that the SS technique used for optimizing the HCSE method can also be applied to the VCSE method.

Consider an 8-tap LPFIR filter whose coefficients in 16-bit CSD form are shown in Fig. 10. In this case, $N_b = 22$ (considering symmetric half coefficients) and $N = 8$. Using (6), twenty-five adders would be required to implement the filter if VCSE is not used. The VCS, [1 -1] and [1 1], encircled in Fig. 10 are given by:

$$x_2 = x_1 - x_1[-1] \text{ and } x_3 = x_1 + x_1[-1] \quad (24)$$

Using these VCS, eighteen adders (12 MBA and 6 SA) are required to implement the filter, two for the VCS (24) and sixteen for the output. This offers a reduction of 28% over the direct implementation without VCSE.

A. Proposed VSSE Method

The 2-bit VCS used in VCSE method can be extended to obtain several 3-bit and 4-bit Vertical Super-Subexpressions (VSS) by exploiting identical shifts between a VCS and a nonzero bit or between two VCS. Consider the example in Fig. 10, where VCS are given by (24). The following multiple bit patterns can be combined:

1) The VCS [1 –1] and [-1 1], with a shift between them (indicated by connecting line 'a') that occur across the coefficients, [$h(0)$, $h(1)$] to form a 4-bit VSS, $\begin{bmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \end{bmatrix}$:

$$x_4 = x_2 - x_2 >> 2 \quad (25)$$

2) The two VCS [1 1], with a shift between them (indicated by connecting line 'b') that occur across the coefficients, [$h(0)$, $h(1)$] to form a 4-bit VSS, $\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$:

$$x_5 = x_3 + x_3 >> 2 \quad (26)$$

Note that several VSS in 'shifted and delayed' forms of (25) and (26) occur in the coefficient set. Employing the VSS, the output of the filter in Fig. 10 can be expressed as

$$x_4 >> 5 + x_5 >> 14 + x_5[-1] >> 9 + x_4[-2] >> 4 + x_4[-2] >> 13$$
$$x_1[-3] >> 1 + x_1[-3] >> 11 + x_1[-4] >> 1 - x_4[-4] >> 4 +$$
$$x_1[-4] >> 11 - x_4[-4] >> 13 + x_5[-5] >> 9 - x_4[-6] >> 5 +$$
$$x_5[-6] >> 14 \quad (27)$$

The filter structure using the VSSE method is shown in Fig. 11. Only fifteen adders are required to implement this filter, two for VCS (24), one each for VSS (25) and (26), and eleven for the filter output (27) after using the symmetry of coefficients. Thus the VSSE method offers better reduction than the VCSE method. The adder-steps in both methods are identical (four) and hence their critical paths are the same. The reduction of FA's, FA_R, offered by the VSSE method over the VCSE method can be determined using the formula:

$$FA_R = \sum_{i=1}^{m} \sum_{j=1}^{n} SCS(i)_{(j)} - SSD(i) \quad (28)$$

where SCS is the span of a VSS, SSD is the span of the shift differential between the VCS of a VSS, m is the number of distinct VSS in the symmetric half coefficient set, and n is the total number of VSS for each distinct VSS set.

We illustrate this using the coefficients of the filter in Fig. 10. Consider the VSS, $\begin{bmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \end{bmatrix}$, and $\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$, across the coefficients, $h(0)$ and $h(1)$. If the wordlength of x_1 is 16 bits, then these VSS have spans $16 + 7 = 23$ and $16 + 16 = 32$ respectively. The spans of the VSS, x_5, across $h(1)$ and $h(2)$ is $16 + 11 = 27$ and that of x_4 across $h(2)$ and $h(3)$ are $16 + 6 = 22$ and $16 + 15 = 31$. Thus, the sum of spans is 135. The spans of the shift differentials *(SSD)* of the VSS, x_4 and x_5, are 18 each. Using (28), it can be found that the proposed VSSE method requires 99 FA's fewer than the VCSE method, which is a reduction of 31%.

B. Compatibility Issue in Vertical Subexpression Methods

An inherent drawback of the VCSE method is that the symmetry of LPFIR filter coefficients cannot be completely exploited for efficient implementation of the filter. In the case of HCSE method, since all the bits forming an HCS exist within the coefficient, its symmetric counter- part can be easily implemented using delays and structural adders. Thus extra adders (MBA) are not required to compute the symmetric half coefficients when HCSE method is used. However, the bits that form a VCS in VCSE method occur across the coefficients and hence the symmetry is destroyed when the bits are of opposite sign [4]. Hence in VCSE implementations, extra adders are required to obtain the symmetric part of the coefficients when more than one VCS with bits of opposite sign exist. Since the basic ideas of VCSE and VSSE methods are the same, the limitation inherent in the former exists in the latter also. Therefore, compatible VCS patterns have to be identified to form a VSS. Two VCS (4-bit VSS) or a VCS and a nonzero bit (3-bit VSS) are said to be compatible to form a VSS if its symmetry is not affected, i.e., no extra adders are required to compute the symmetrical part of the LPFIR filter. The signs of the bits in VCS determine the compatibility. We use the notation, $s(b)$, to represent 'sign of bit 'b' in defining compatibility.

Definition 7 (Compatible 4-bit VSS): Let $\begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$ and $\begin{bmatrix} b_2 \\ b_3 \end{bmatrix}$ represent the VCS that form a VSS. These VCS are compatible if one of the following conditions is satisfied:

(a) $s(b_0) = s(b_2)$ and $s(b_1) = s(b_3)$.

(b) $s(b_0) \neq s(b_2)$ and $s(b_1) \neq s(b_3)$.

Fig. 12(a) and Fig. 12(b) illustrate compatible 4-bit VSS corresponding to conditions (a) and (b) respectively. The negated versions of these bit patterns are also compatible. Note that VCS, whose bits are related with delays up to two units, i.e., $x_1 \pm x_1[-2]$, are considered to form VCS in this proposed method.

Definition 8 (Compatible 3-bit VSS): Consider the VCS, $\begin{bmatrix} b_0 \\ 0 \\ b_1 \end{bmatrix}$. A 3-bit VSS can be formed by combining this VCS and a third bit, b_2. In this case, the necessary conditions for compatibility are:

(a) b_2 must have a unity delay relation with respect to b_0 and b_1, and

(b) $s(b_0) = s(b_1)$.

Fig. 12(c) illustrates 3-bit compatible VSS patterns. Note that the bit '1' which is combined with the VCS 101 can be anywhere in the second row. The notation 'x' denotes "don't cares" since the bits in these locations will not affect the compatibility of VSS. Any VSS that does not satisfy the conditions mentioned above is *incompatible.*

We investigated the same examples of LPFIR filters designed using Parks-McClellan method discussed in previous section. CSD coefficients of wordlengths 8-bits, 12-bits, 14-bits and 16-bits were considered. It has been observed that the most common VSS are the 3-bit VSS that form around 60% of all the VSS and hence they account for the major reduction of adders in the VSSE method. Design examples of the HSSE and VSSE methods are provided in section 6.

V. Extension of CSE to High Level Syhthesis

A. High Level Synthesis Transformation

In high level synthesis, the primary goal of transformations has been to optimize the ASIC design to reduce cost metrics (area and power) while meeting throughput constraints [11]. The high level synthesis literature has an extensive coverage of CSE as a powerful transformation to reduce power consumption and area [2], [12]-[14]. Iqbal *et al.* [12] used CSE within their algebraic speed-up procedure for throughput improvement. The objective function in [12] was to reduce the critical path. The approaches in [2], [13] and [14] focused on a more apparent goal of reducing the number of operations, and therefore, area and power of designs. The significant advancement for the transformation using CSE was achieved by Potkonjak *et al.* in [13], [14]. They first formulated the MCM problem in high level synthesis by considering the multiplications of one variable with several constants at a time and also reduced the number of shifts and additions based on an iterative pairwise matching. Mehendale *et al.* [2] considered the CSE problem by examining the filter coefficient matrix and the iterative elimination of the most frequently occurring common subexpressions.

In general, the high level synthesis tasks of the methods in [2], [13] and [14] are based on elimination of 2 nonzero-bit common subexpressions as shown in Fig. 13 (a). The operands, *a, b, c,* and *d* in Fig. 13 (a) represent the input signal of the filter and its shifted versions. The sums, *e* and *f,* are the common subexpressions that are shared for minimizing adders and s_1, s_2, s_3, s_4 represent the shifts. Note that four adders are required to obtain the final expressions, *h* and *i.* Fig. 13 (b) illustrates our super-subexpression method, where the super-subexpression *g* is shared for further reduction of adders to obtain *h* and *i* using appropriate shifts. Note that only three adders are required using our method. Thus, by employing the new transformation (super-subexpression), our method improves the efficiency of CSE in high level synthesis and offers a more power efficient solution by reducing the number of operations (additions).

B. Area and Power Reduction

In CMOS technology, there are three sources of power dissipation arising from: switching (dynamic) currents, short-circuit currents, and leakage currents. Among these parameters, the switching component, which is a function of the effective capacitance, plays the most significant role [15]. It is possible to reduce power by employing transformations such as reductions in critical path, number of operations and average transition activity. These transformations result in architectures that minimize the effective capacitance of the circuit [15]. The basic motivation behind critical path reduction is that the supply voltage can be lowered while keeping the throughput fixed. It can be noted from the design examples of previous sections that the tree-structured (parallel) addition (shown in Fig. 5) adopted in our method results in considerable reduction of critical path. Moreover, when compared with a chain (serial) implementation, the signal paths are more balanced in a tree implementation and the amount of extra transitions is reduced. For example, the capacitance switched for a chained implementation is a factor of 1.5 larger than the tree implementation for a four input addition [15]. Thus, the filter structure used in our method is efficient in terms of critical path length and transition activity. Having optimized these two parameters, the most obvious approach for capacitance reduction is to reduce the number of operations (and hence the number of switching events) in the data control flow graph. The super-subexpression elimination methods proposed in this paper is an efficient transformation that directly reduces the number of operations through the reduction of FA's required for each adder.

We illustrate the area reduction achieved in our method using the example of the 8-tap filter coefficients in Fig. 4. The area is computed in terms of full adder area, based on Synopsys 0.35-micron library. In terms of gate equivalents, 1 full adder = 7 gates (Note that the basic gate is NAND gate). Note that 376 FA's are required to implement the MB using HCSE method. As 1 full adder area = 14.46 units, it requires $376 \times 14.46 = 5436.96$ units (area) to implement the MB of the filter coefficients in Fig. 4 using HCSE. On the other hand, our HSSE method needs only 253 FA's, which is equivalent to 3658.38 units for the MB. The higher reduction of FA's achieved using our method in the case of channel filters (that possess large number of taps) employed in wideband receivers significantly reduces the cost metrics, area and power.

VI. Design Examples

The channel filters of a wideband receiver that operate in the intermediate frequency (IF) have large number of taps due to their narrow transition band and high sampling frequency requirements. Therefore, the CSE optimization methods proposed in this paper offers considerable complexity reduction when used to implement the channel filters. We present examples of implementing channel filters for the D-AMPS and the PDC cellular standards using the HSSE and VSSE methods. The proposed optimization methods are compared with conventional 2-bit CSE techniques and reductions of FA's are determined. Based on the simulation results obtained for filters with different wordlengths, certain guidelines on the choice of HSSE and VSSE methods are also drawn.

Example 1: We consider the LPFIR filters employed in the filter bank channelizer of the Digital Advanced Mobile Phone System (D-AMPS) in [16]. The filter bank structure in Fig. 2 is used. Note that decimation by N is moved to the left of the bandpass filters using the noble identity and the sampling rate chosen is 34.02 MHz as in [16]. The channel filters extract 30 kHz D-AMPS channels from the input signal after downsampling by a factor of 350. The pass-band and stop-band edges are 30 kHz and 30.5 kHz respectively. The peak pass-band ripple specification is chosen as 0.1 dB. The filter stop-band specifications at different frequencies are chosen as in the D-AMPS standard [17]. The lengths of the LPFIR filters required to meet these specifications are determined using [18]

$$N = \frac{-10\log_{10}\delta_1\delta_2 - 13}{14.6\Delta f} + 1 \qquad (29)$$

where δ_1 and δ_2 represent the passband and stopband ripples, and Δf is the normalized width of the transition region. We applied the proposed HSSE and VSSE methods to implement the filters using 12-bit and 16-bit CSD coefficients. The 3-bit and the 4-bit SS formed from the 2-bit CS are utilized for optimization. Table I shows a comparison of the number of adders for computing the Partial Products (PP adders) required for implementing the filters using conventional HCSE and VCSE methods and the proposed methods.

We compare the reduction rates of HCSE, VCSE, HSSE, and VSSE methods with respect to conventional CSD implementation without using any CS methods. The comparison of reduction rates of adders achieved using proposed methods and that of CSE methods for 12-bit and 16-bit wordlengths are shown in Tables II and III respectively.

It can be observed that the VSSE method offers a better reduction rate over the HSSE method when the bit precision of implementation is lower (12-bit). The VSSE technique offers an average reduction of 39.9% for the 12-bit implementation whereas an average reduction of 43.7% is achieved using the HSSE technique for the 16-bit implementation. Note that both methods require fewer PP adders than the 2-nonzero-bit CSE methods. The number of FA's required for implementation is shown in Table IV. The reduction rates of FA's achieved using VSSE and HSSE methods over 12-bit and 16-bit CSD implementations without using subexpressions are shown in Tables V and VI respectively. Both HSSE and VSSE methods results in significant reduction of FA's when compared to HCSE and VCSE methods. In the case of implementation using 12-bit, the VSSE method offers the best reduction (47.2%), whereas the reduction offered by the HSSE method is the best (54.2%) for the 16-bit case.

The reduction achieved when the proposed methods are used to employ the D-AMPS channelizer where extraction of each channel requires a separate narrowband filter is examined. The wideband signal considered for channelization consists of 1134 D-AMPS channels, each of 30 kHz spacing. We analyzed the requirement of PP adders to implement the filters for extracting 70, 141, 283, 567, and 1134 channels. The number of filter taps chosen is 1180 and the coefficient wordlength considered is 16 bits to meet the requirement of attenuating blockers that can be potentially 96 dB stronger than the wanted signal. Simulation results shown in Fig. 14 depict the adder reductions achieved using our proposed methods as a function of the number of extracted channels. The percent reductions are shown with respect to conventional implementation without using any CSE methods. Both the HSSE and VSSE methods offer considerable hardware reduction and also result in *better rate of change in hardware reduction* as the number of channels increases compared to the CSE methods.

Example 2: In this example, we consider the channel filters employed in receivers for the PDC standard. The sampling rate of the wideband signal is 25.6 MHz, which covers 1024 channels of 25 kHz spacing. We fix the filter length as 1000 to meet the maximum attenuation requirement of –90 dB and 24-bit coefficients are considered. The number of PP adders required to implement the filter is shown in Table VII. The requirement of FA's are also shown in Table VII, which shows that the proposed HSSE and VSSE methods offer a minimum reduction of 20% over the CSE methods.

Based on the simulation results, the following guidelines for choosing the best implementation method can be formulated:

1) As in the case of VCSE method, the coefficient symmetry of LPFIR filters cannot be completely exploited in VSSE method. Hence the proposed VSSE method offers better hardware reduction over the HSSE method only when the bit precision of implementation is lower. For larger wordlength implementations, the spans of the operands are also larger and hence the HSSE method results in better reduction of adders.

2) For the 12-bit implementation, the VSSE method results in an average FA reduction of 20% over the VCSE method, whereas for wordlengths of 16-bit and higher, the HSSE method offers an average reduction of 25% over the HCSE method.

VII. Conclusions

In this paper, we have proposed two techniques for optimizing the CSE methods to efficiently implement low-complexity LPFIR filters. They are based on the extension of conventional 2-nonzero-bit HCS to form 3-nonzero-bit and 4-nonzero-bit HSS by exploiting identical shifts between an HCS and a nonzero bit, or between two HCS. These HSS eliminate redundant computations of HCS and hence reduce the number of adders. We have also applied the optimization

technique to the VCSE method and formulated the VSSE algorithm. Furthermore, the complexities of adders are analyzed and expressions for determining the number of FA's required for each adder in a filter are derived. The experimental results show that considerable reduction of FA's can be achieved using proposed methods. Certain guidelines on the choice of HSSE and VSSE methods are also provided. We have applied the proposed methods to filter bank channelizers, where common CS that occur among a bank of filters are utilized. The design examples of channelizers based on D-AMPS and PDC cellular standards show that the optimization techniques presented in this paper offers an average reduction rate of 50% over conventional channel filter implementations.

ACKNOWLEDGMENT

The authors would like to thank Dr.A.B.Premkumar, School of Computer Engineering, Nanyang Technological University, Singapore, for the valuable discussions throughout the work.

REFERENCES

[1] J. Mitola, *Software Radio Architecture.* New York: Wiley, 2000.

[2] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, "Synthesis of multiplierless FIR filters with minimum number of additions," in *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design,* Los Alamitos, CA: IEEE Computer Society Press, 1995, pp. 668-671.

[3] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II,* vol. 43, pp. 677-688, Oct. 1996.

[4] M. Yagyu, A. Nishihara, and N. Fujii, "Fast FIR digital filter structures using minimal number of adders and its application to filter design," *ICICE Trans. Fundam. Electron. Commun. Comput. Sci.,* no. 8, pp. 1120-1129, E79-A, 1996.

[5] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A new algorithm for elimination of common subexpressions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Syst.,* vol. 18, no. 1, pp. 58-68, January 1999.

[6] R. Pasko, P. Schaumont, V. Derudder, and D. Durackova, "Optimization method for broadband modem FIR filter design using common subexpression elimination," in *Proceedings on the 10th International Symposium on System Synthesis,* pp. 100-106, 1997.

[7] M. M. Peiro, E. I. Boemo, and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," *IEEE Trans. Circuits Syst. II,* vol. 49, no. 3, pp. 196-203, March 2002.

[8] Y. Jang and S. Yang, "Low-power CSD linear phase FIR filter structure using vertical common sub-expression," *Electronics Letters,* vol. 38, no. 15, pp. 777-779, July 2002.

[9] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *IEE Proceedings – G,* vol. 138, no. 3, pp. 401-412, June 1991.

[10] [10] A. G. Dempster and M. D. Mcleod, "Use of minimum adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II,* vol. 42, pp. 569-577, Sept. 1995.

[11] [11] R. A. Walker and D. E. Thomas, *A Survey of High-Level Synthesis Systems.* Boston, MA:Kluwer Academic 1991.

[12] [12] Z. Iqbal, M. Potkonjak, S. Dey, and A. Parker, "Critical path minimization using retiming and algebraic speed-up," in *ACM/IEEE Design Automation Conf.,* pp. 573-577, 1993.

[13] [13] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Efficient substitution of multiple common multiplications by shifts and additions using iterative pairwise matching," in *ACM/IEEE Design Automation conf.,* pp. 189-194, 1994.

[14] [14] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. On CAD,* vol. 15, no. 2, pp. 151-165, Feb. 1996.

[15] [15] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen, "Optimizing power using transformations," *IEEE Trans. On CAD,* vol. 14, no. 1, pp. 12-31, Jan. 1995.

[16] [16] K. C. Zangi and R. D. Koilpillai, "Software radio issues in cellular base stations," *IEEE Journal on Selected Areas in Communication,* vol. 17, no. 4, pp. 561-573, April 1999.

[17] [17] N. Spencer, "An overview of digital telephony standards," *IEE Colloquium on the Design of Digital Cellular Handsets,* pp. 1/1-1/7, March 1998.

[18] [18] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing Principles, Algorithms,* and Applications. Prentice Hall, 1998.

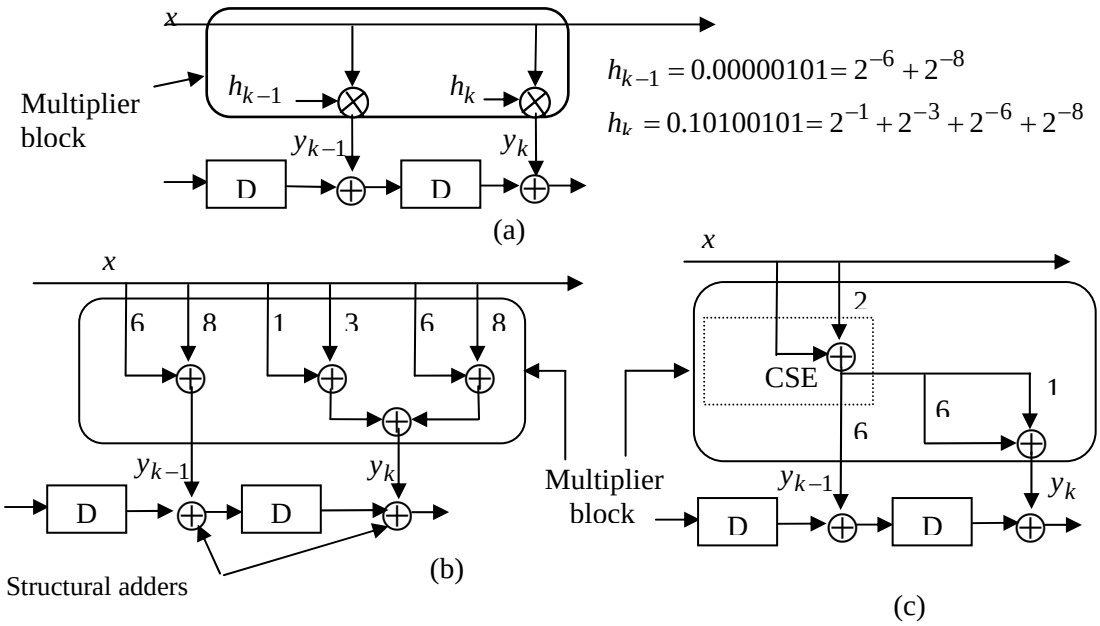


Fig. 1. FIR filter implementation using HCSE.

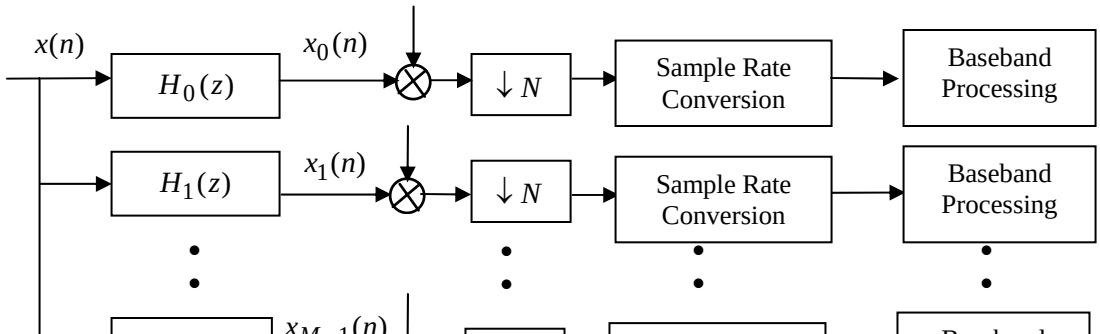


Fig. 2. Filter bank channelizer of an SDR receiver.

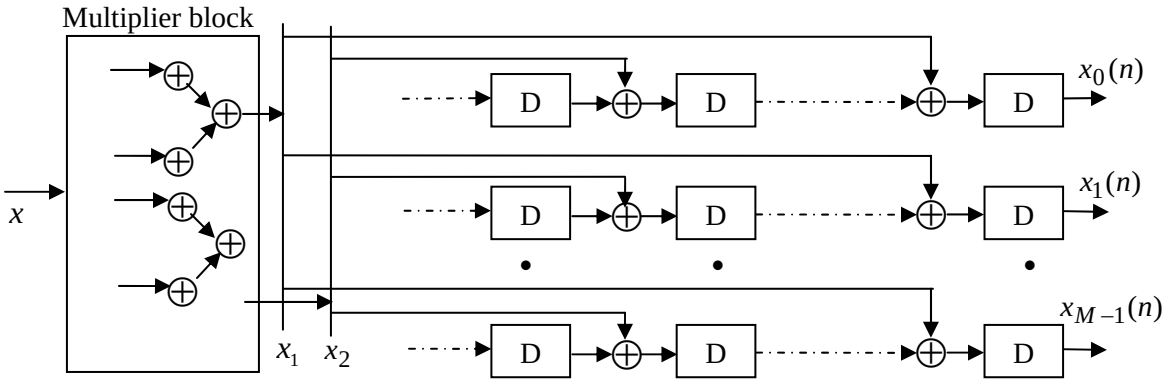


Fig. 3. CSE implementation of channel filters in a filter bank channelizer.

	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
h(0)	**1**	**0**	**1**	**a**	**-1**			**1**	**0**	**1**	**b**		**1**	**0**	**-1**	
h(1)		**1**		**-1**	**0**	**1**	**c**	**1**	**0**	**1**		**1**	**0**	**1**	**a**	**-1**
h(2)		**1**	**0**	**1**	**b**		**1**	**0**	**-1**				**1**	**0**	**-1**	
h(3)	**-1**	**0**	**1**	**c**	**1**	**0**	**1**		**1**	**0**	**1**	**b**		**1**	**0**	**-1**
h(4)	**-1**	**0**	**1**	**c**	**1**	**0**	**1**		**1**	**0**	**1**	**b**		**1**	**0**	**-1**
h(5)		**1**	**0**	**1**	**b**		**1**	**0**	**-1**				**1**	**0**	**-1**	
h(6)		**1**		**-1**	**0**	**1**	**c**	**1**	**0**	**1**		**1**	**0**	**1**	**a**	**-1**
h(7)	**1**	**0**	**1**	**a**	**-1**			**1**	**0**	**1**	**b**		**1**	**0**	**-1**	

Fig. 4. HCS and HSS in an 8-tap linear phase FIR filter coefficients.

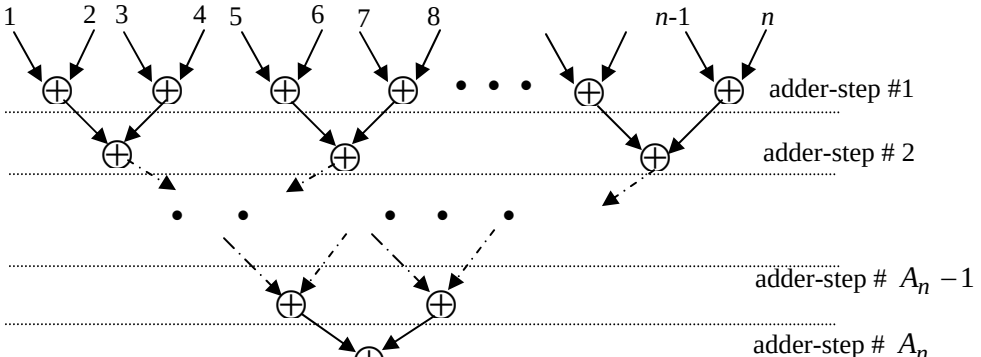
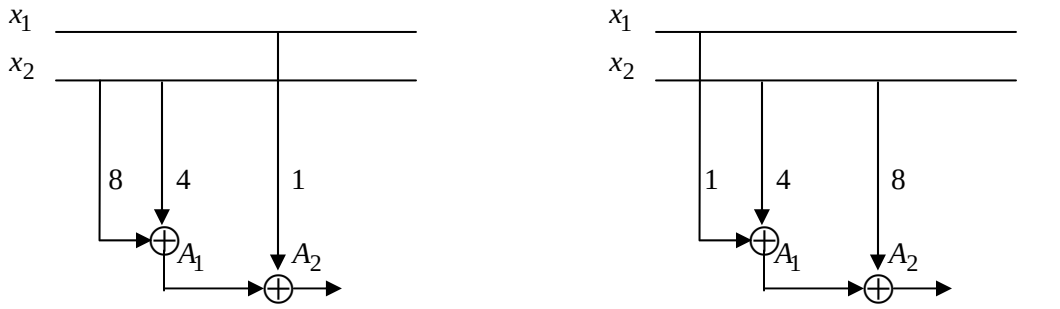


Fig. 5. Tree structure used for addition of partial products in the multiplier block.



(a) Implementation without full adder optimization. (b) Minimum full adder implementation.

Fig. 6. Optimization of addition sequences (Odd number of operands).

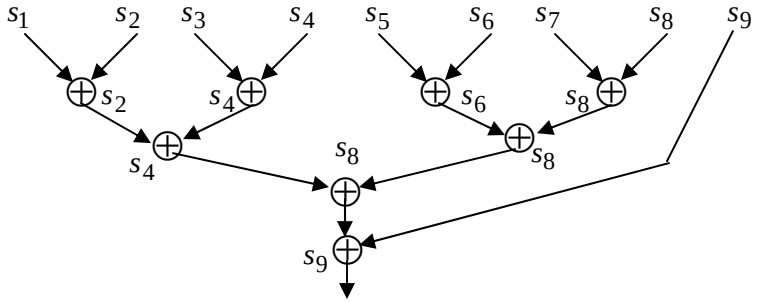
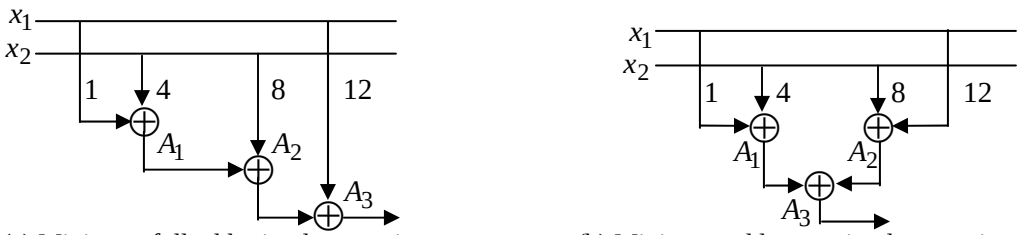


Fig. 7. Implementation of filter tap for odd number of operands.



(a) Minimum full adder implementation resulting in maximum adder-step. (b) Minimum adder-step implementation requiring additional full adders.

Fig. 8. Optimization of addition sequences (Even number of operands).

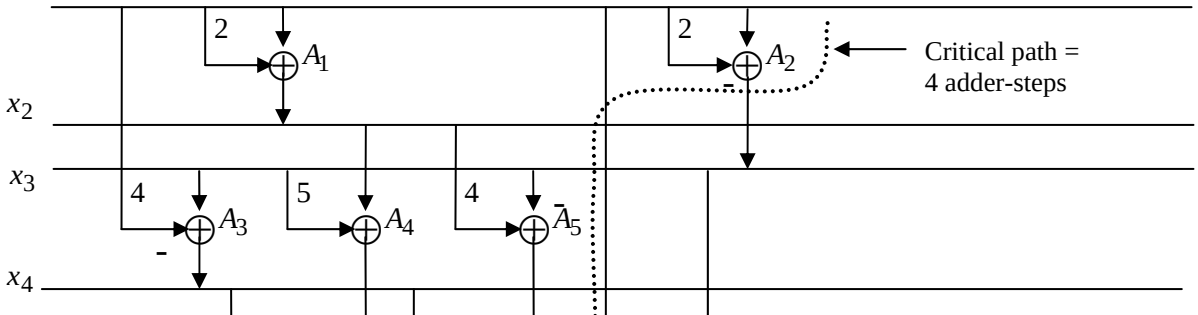


Fig. 9. Proposed filter structure using horizontal super-subexpressions of Fig. 4.

	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
$h(0)$					**1**	**0** a	**-1**							**1**	**0** b	**1**
$h(1)$					**-1**	**0**	**1**		**1**	**0** b	**1**			**1**	**0**	**1**
$h(2)$				**1**	**0** a	**-1**			**1**	**0**	**1**		**1**	**0** a	**-1**	
$h(3)$	**1**			**-1**	**0**	**1**					**1**		**-1**	**0**	**1**	
$h(4)$	**1**			**-1**	**0** a	**1**					**1**		**-1**	**0** a	**1**	
$h(5)$				**1**	**0**	**-1**			**1**	**0** b	**1**		**1**	**0**	**-1**	
$h(6)$					**-1**	**0** a	**1**		**1**	**0**	**1**			**1**	**0** b	**1**
$h(7)$					**1**	**0**	**-1**							**1**	**0**	**1**

Fig. 10. VCS and VSS in an 8-tap linear phase FIR filter coefficients.

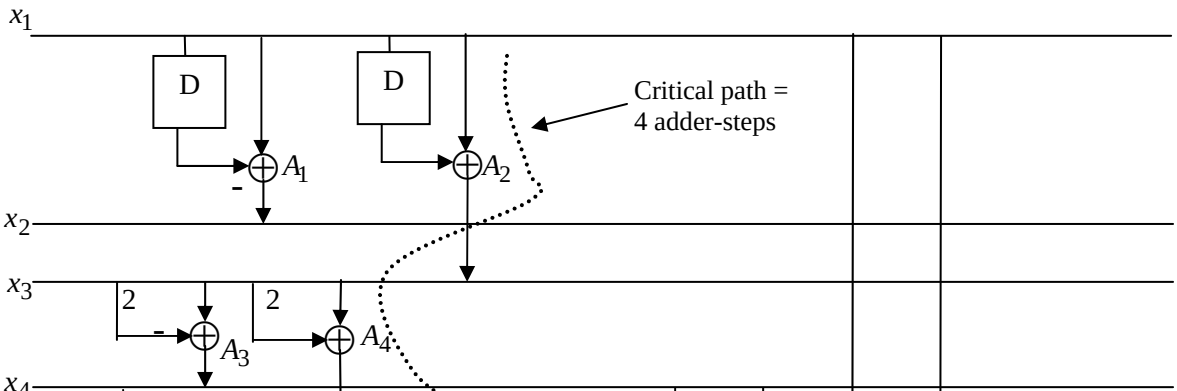
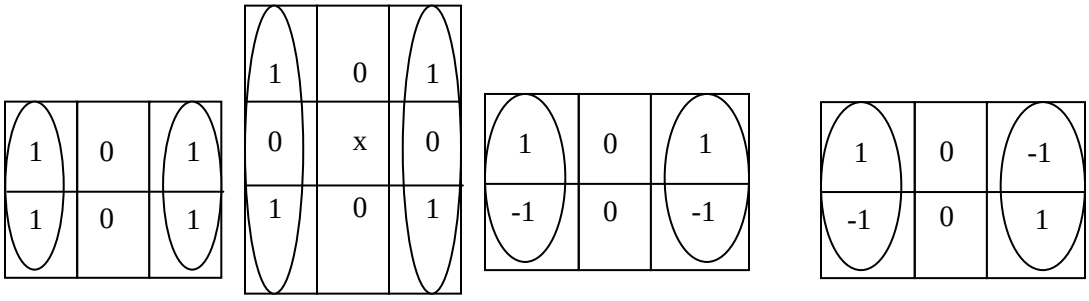
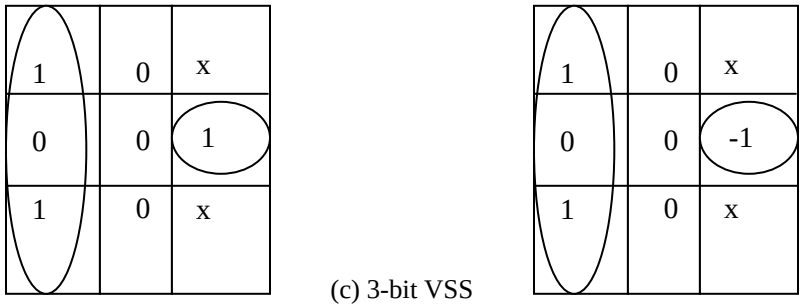


Fig. 11. Proposed filter structure using vertical super-subexpressions of Fig. 10.

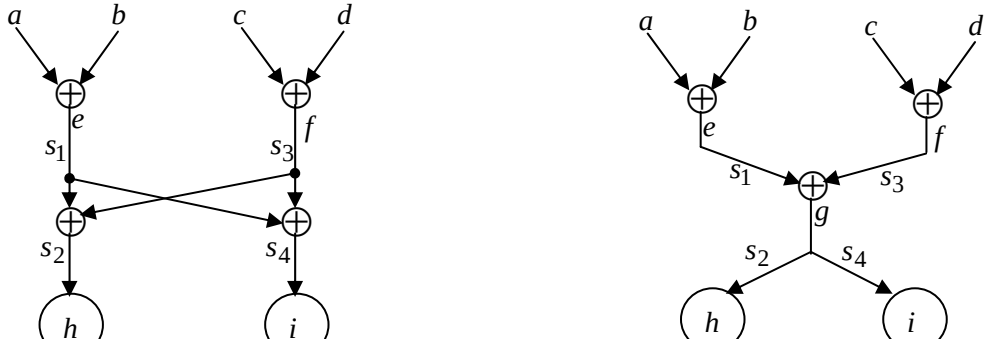


(a) 4-bit VSS: case a (b) 4-bit VSS: case b



(c) 3-bit VSS

Fig. 12. Compatible vertical super-subexpressions.



(a) Conventional 2-bit CSE. (b) Proposed 3-bit/4-bit SSE.

Fig. 13. Subexpression sharing as a high level synthesis transformation.

Table I Number of PP adders required to implement the filter in example 1

PSR (dB)	Filter length (N)	CSD		VCSE [8]		Proposed VSSE		HCSE [3]		Proposed HSSE	
		12 bit	16 bit	12 bit	16 bit	12 bit	16 bit	12 bit	16 bit	12 bit	16 bit
-48	260	318	404	226	310	204	267	240	290	220	258
-65	610	740	856	520	640	466	556	560	586	507	512
-85	940	1010	1280	675	910	590	772	745	818	664	679
-96	1180	1138	1480	736	1040	622	847	820	896	712	714

Table II Reduction of PP adders in example 1 over 12-bit CSD implementation.

Filter length (N)	Reduction of adders over conventional 12-bit CSD implementation (%)			
	VCSE [8]	Proposed VSSE	HCSE [3]	Proposed HSSE
260	28.9	35.8	24.5	30.8
610	29.7	37	25	31.5
940	33.2	41.6	26.2	34.3
1180	35.3	45.3	27.9	37.4
Average reduction	31.8	39.9	25.9	33.5

Table III Reduction of PP adders in example 1 over 16-bit CSD implementation.

Filter length (N)	Reduction of adders over conventional 16-bit CSD implementation (%)			
	VCSE [8]	Proposed VSSE	HCSE [3]	Proposed HSSE
260	23.3	33.9	28.2	36.1
610	25.2	35	31.5	40
940	28.9	39.6	36.1	46.9
1180	29.7	42.8	39.4	51.8
Average reduction	26.8	37.8	33.8	43.7

Table IV Number of full adders required to implement the filter in example 1.

PSR (dB)	Filter length (N)	VCSE [8]	Proposed VSSE	HCSE [3]	Proposed HSSE

		12 bit	16 bit	12 bit	16 bit	12 bit	16 bit	12 bit	16 bit
-48	260	3729	6200	3170	5140	3840	5560	3380	4710

Filter length (N)	Reduction of full adders using horizontal subexpression techniques over CSD implementation (%)			
	12-bit		16-bit	
	HCSE [3]	Proposed HSSE	HCSE [3]	Proposed HSSE
260	22.8	34.7	26.9	42.2
610	24.9	37.8	29.8	54.2
940	26.2	41.2	33.2	58.2
1180	28.1	46	35.6	62.1
Average reduction	25.5	39.9	31.4	54.2

-65	610	8730	12928	7250	10600	9072	11376	7892	8600
-85	940	10125	18746	8106	14809	12218	16121	10385	12090
-96	1180	11040	21632	8680	16660	13776	17740	11298	13042

Table V Reduction of FA's in example 1 achieved using vertical subexpression techniques

Filter length (N)	Reduction of full adders using vertical subexpression techniques over CSD implementation (%)			
	12-bit		16-bit	
	VCSE [8]	Proposed VSSE	VCSE [8]	Proposed VSSE
260	26.1	41.1	27.7	44.8
610	28	44.9	29.2	47.2
940	30.1	50	31.1	52.1
1180	31.4	52.8	33.8	56.7
Average reduction	28.9	47.2	30.45	50.2

Table VI Reduction of FA's in example 1 achieved using horizontal subexpression techniques

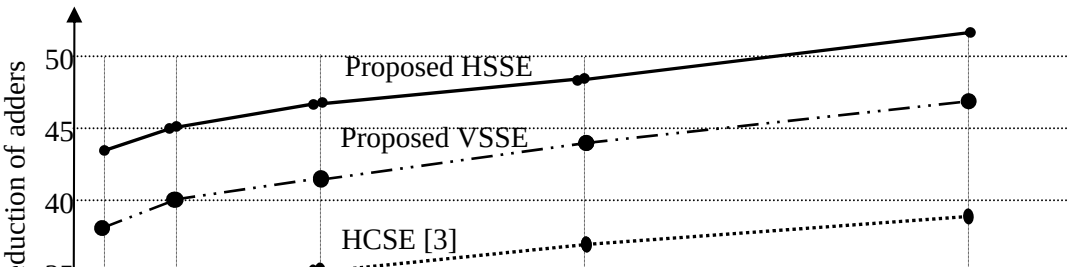


Fig. 14. Reduction of adders to implement the D-AMPS channel filters in design example 1 for different number of channels extracted.

Table VII Number of PP adders required to implement the channel filter for the PDC standard in example 2

	Vertical subexpression (12-bit)		Horizontal subexpression (16-bit)	
	VCSE [8]	Proposed VSSE	HCSE [3]	Proposed HSSE
Adders	696	610	852	746
Adder reduction (%)	30.6	39.1	33.8	42
Full adders	10600	7488	17260	11180
Full adder reduction (%)	30.7	50.9	35.2	58