Circuits Syst Signal Process
DOI 10.1007/s00034-009-9114-7

Optimization of Linear Phase FIR Filters in Dynamically Expanding Subexpression Space

Ya Jun Yu · Yong Ching Lim

Received: 20 July 2008 / Revised: 9 October 2008

Abstract The most advanced techniques in the design of multiplierless finite impulse response (FIR) filters explore common subexpression sharing when the filter coefficients are optimized. Existing techniques, however, either suffer from a heavy computational overhead, or have no guarantees on the minimal hardware cost in terms of the number of adders. A recent technique capable of designing long filters optimizes filter coefficients in pre-specified subexpression spaces. The pre-specified subexpression spaces determine if a filter with fewer adders may be achieved. Unfortunately, there is no known technique that can find subexpression spaces that can guarantee the solution with the minimum number of adders in the implementation. In this paper, a tree search algorithm is proposed to update and expand the subexpression spaces dynamically, and thus, to achieve the maximum subexpression sharing during the optimization. Numerical examples show that the proposed algorithm generates filters using fewer adders than other non-optimum algorithms. On the other hand, as a consequence of its efficiency, our proposed technique is able to design longer filters than the global optimum algorithm.

Keywords FIR filters · Common subexpression sharing · Subexpression space · Tree search algorithm

This project is supported in part by Nanyang Technological University (NTU) Research Grant COE-SUG (1) and Temasek Laboratory @ NTU.

Y.J. Yu (✉) · Y.C. Lim
School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore
e-mail: eleyuyj@pmail.ntu.edu.sg

Y.C. Lim
e-mail: elelimyc@pmail.ntu.edu.sg

Published online: 05 June 2009

1 Introduction

It is well known that multiplierless finite impulse response (FIR) filters are very attractive in VLSI (very large-scale integration) implementation, since multipliers are the most expensive and power-hungry components. In multiplierless FIR filters, filter coefficients are optimized in a discrete space, such as the finite wordlength space [11] and signed power-of-two (SPT) space [9, 14, 15, 26, 28], so that the multiplication of a data sample by filter coefficient values can be replaced by a sequence of shifts and adds. The shifts for filter coefficients with fixed values can be realized by using hardwired shifters, and hence the complexity of multiplierless filters is usually measured by the number of adders.

The most recent development for multiplierless filters is sharing common subexpressions among coefficients to minimize the number of adders [1–4, 6–8, 10, 17, 18, 20–25, 27]. Existing techniques for the design of FIR filters by sharing subexpressions can be classified into two categories.

In the first category, the filters are designed in two stages. First, the FIR filter is designed in a discrete space such as a finite wordlength space or SPT space to meet a given specification. In the second stage, a common subexpression elimination [1, 7, 8, 18, 20, 23, 24], a graph-based algorithm [2, 3, 10, 21] or a difference algorithm [4, 17, 22] is applied on the discrete coefficients to find and share the common subexpressions. An obvious disadvantage of such a two-stage optimization is that the search space in the second stage is limited by the finite wordlength or SPT coefficients obtained in the first-stage optimization.

In the second category, the common subexpression sharing is taken into consideration when filter coefficients are optimized. A local search algorithm is proposed in [24]. By constraining the search range and search patterns, the resulting filters have a tradeoff between the hardware cost and computational complexity. In [25], the dynamic range of each coefficient is determined, and thus, the permissible discrete values for each coefficient are obtained. An exhaustive search by applying a simple branch and bound algorithm is then used to evaluate the filter performance and adder cost for each possible combination. In the evaluation of the adder cost for each coefficient combination, the algorithms in the first category are adopted. In this technique, the search range becomes impractically huge when the filter order is high and/or the coefficient wordlength is large. In addition, the algorithm in the first category cannot guarantee the minimum number of adders for an arbitrarily given discrete coefficient set. The optimum of the technique of [25] is, therefore, not guaranteed. In [6], a $0/1$ integer programming is used to minimize the number of adders subject to a given filter specification. This algorithm produces global optimum solutions in terms of number of adders, but this algorithm is computationally very demanding, and thus is only suitable for relatively short filters. A branch-and-bound (B&B) mixed-integer linear programming (MILP) is proposed in [27] to optimize FIR filters in subexpression spaces. In this algorithm, subexpression spaces are constructed from a given set of subexpression bases. Different coefficients may have different subexpression spaces, which are determined by the number of subexpression terms allocated to each coefficient. The B&B MILP, thus, only searches for the coefficient values in the specified subexpression spaces. This technique, when applied without sectioning, can be used to design filters with order up to 70; with sectioning, it has been used [27] to design a 121-tap filter. However, in this algorithm, the choice of the subexpression basis set and the allocation of the number of subexpression terms to each coefficient are critical to the success of the algorithm. Unfortunately, there are no guaranteed ways to find an optimum subexpression basis set and an optimum term allocation scheme that may minimize the number of adders used in the implementation of a filter.

To get around the difficulty of choosing the basis set and allocating the subexpression terms in the algorithm of [27], in this paper, a method that is capable of dynamically expanding the subexpression basis set is proposed. In this algorithm, the basis set is initialized to have order 0, i.e., it only has element 0, 1 and -1. The optimization starts from a design with continuous coefficients. Coefficients are fixed to the discrete value one at a time, and the remaining coefficients are re-optimized in the continuous space. If the fixed discrete coefficient value can be generated from the existing basis set without using any adders, the optimization proceeds to fix the second coefficient; otherwise, this discrete value is generated from the current subexpression bases using the minimum number of adders, and the basis set is updated and expanded by including this discrete value. To efficiently search in the discrete space and update the subexpression basis set, the depth-first width-recursive tree search algorithm [16] is used to traverse the search process.

The proposed algorithm generates filters using fewer adders than the algorithms proposed in [24, 25, 27]. The optimization time for problems with similar size varies heavily, but as an estimation the proposed algorithm can be used with length up to 50 and bit width up to 11 bits, and the solving time ranges from a few hours to a few tens of hours on a 3 GHz PC. Since the computation time of the depth-first width-recursive algorithm increases exponentially with the filter length [16], it takes an impractically long time to optimize longer filters and with more bits. However, comparing with the global optimum algorithm reported in [6], the proposed algorithm is able to handle longer filters with wider bits; when the filter length and bit width are the same as those reported in [6], this newly proposed technique produces similar results at a reduced computation time.

The remainder of the paper is organized as follows. Section 2 reviews the subexpression space construction based on a given set of subexpression bases. The limitation of the B&B MILP algorithm in optimizing FIR filters in subexpression spaces is also discussed. Section 3 reviews the depth-first width-recursive tree search to optimize the filter coefficients in discrete spaces. Approaches to optimize the filter coefficients of each node of the tree are introduced in Sect. 4. Detailed techniques to dynamically expand the subexpression basis set and subexpression space are discussed in Sect. 5. Section 6 introduces the features to efficiently search the tree. Numerical examples are shown in Sect. 7 to illustrate the advantage of the proposed technique. In Sect. 8, the computational complexity of the tree search algorithm is analyzed.

2 Subexpression Spaces and B&B MILP Search in Subexpression Spaces

Since a finite wordlength discrete value can be converted into an integer value by multiplying a suitable integer power-of-two, in this paper, only integer values are considered.

A subexpression space is constructed based on a subexpression basis set [27]. The elements of the basis set are integer odd numbers and zero. It is stipulated that if an odd integer is in a basis set, its negative value must also be in the same set. A typical basis set, which includes the most common subexpressions [8], can be represented as $S_t = \{0, \pm 1, \pm 3, \pm 5\}$. The order of the basis set is defined as the number of adders required to generate the value of all elements in the set. Therefore, the order of the basis set S_t is 2.

Zero is a default basis for any basis set. In the following, 0 is omitted from the basis set for expository convenience. S_t thus is represented as $\{\pm 1, \pm 3, \pm 5\}$.

A subexpression space with wordlength Q is constructed by defining its element as

$$n = \sum_{i=0}^{K-1} y(i) 2^{q(i)}, \quad y(i) \in S, \tag{1}$$

where S is a subexpression basis set and $q(i)$ is an integer. $y(i)2^{q(i)}$ thus forms a subexpression term and K is the allowed number of subexpression terms. Since the subexpression space is of wordlength Q, it is obvious that $q(i) < Q - \log_2 |y(i)|$.

For a linear-phase FIR filter of length N, the zero-phase frequency response can be expressed as

$$H(\omega) = \sum_{n=0}^{\lfloor \frac{N-1}{2} \rfloor} h_n \operatorname{Trig}(\omega, n), \tag{2}$$

where the h_n's are the filter coefficients and $\operatorname{Trig}(\omega, n)$ is an appropriate trigonometric function depending on the parity of N and the symmetry of the impulse response [12].

When using B&B MILP to optimize the filter coefficients in subexpression space, a subexpression basis set is pre-specified, and the number of subexpression terms is pre-allocated to each coefficient. The normalized peak ripple magnitude (NPRM) [12] is minimized subject to the filter specifications such as band edges and band ripple ratios.

During the B&B optimization, filter coefficients are selected for branching, and the bounds imposed on the coefficients can only be the values in the subexpression spaces constructed by the pre-specified subexpression basis set with the pre-allocated numbers of subexpression terms. Therefore, the success of the B&B MILP heavily depends on the proper choice of the subexpression basis set and the number of subexpression terms for each coefficient. Unfortunately, there are no guaranteed ways to find an optimum subexpression basis set and an optimum term allocation scheme which may minimize the number of adders.

Furthermore, in B&B MILP, when a coefficient is allocated with more than one subexpression term, a new discrete value which is not in the pre-specified basis set may be generated. However, it is not possible to include this new discrete value as a subexpression basis to construct a denser subexpression space in the search due to the nature of the B&B MILP; this further limits B&B MILP to optimize the filter coefficients to achieve the minimum number of adders.

3 Depth-First Width-Recursive Search

To get around the limitations in the B&B MILP search algorithm, dynamically expanding the subexpression basis set is incorporated with a depth-first width-recursive search [16] to optimize the filter coefficients in subexpression space. In this section, the depth-first width-recursive tree search is reviewed.

The depth-first width-recursive search starts with the design of the continuous coefficient value minimax FIR filter. After the continuous optimum solution is obtained, a coefficient h_k is selected for quantization. As the optimum value for h_k may be at a considerable distance from the infinite precision solution, it is necessary to assign several discrete values (in the vicinity of its continuous optimum value) to h_k. For each discrete value assigned to h_k, the remaining unquantized coefficients are re-optimized in the continuous space to partially compensate for the frequency response deterioration due to the quantization of h_k. A tree search algorithm is then produced. Each set of coefficients, which may contain only continuous values, or only discrete values, or a mix of continuous and discrete values, is a node of the tree. The tree width determines the number of discrete values assigned to each coefficient.

The algorithm starts with the tree width of $L = 1$. The continuous solution is the root of the tree (i.e., node P_0 in Fig. 1). A new node P_1, which is a child of P_0, is produced when a coefficient (say h_2) is quantized and the remaining coefficients are re-optimized. P_0, by definition, is the parent of P_1. Nodes having the same parent are siblings. Any newly generated node will be either further explored by quantizing another continuous coefficient to discrete value or fathomed depending on whether further exploration of the node is beneficial or not. A fathomed node having no child is a leaf of the tree (i.e., P_3 in Fig. 1). If every node except the leaves in the tree has L children, it is said that the tree with width L has been completely traversed. In Fig. 1, the search along nodes P_0, P_1, P_2 to P_3 completes a width 1 search when P_3 is fathomed.

After the traverse of the tree with width 1 is completed, the width of the tree is incremented by one. In Fig. 1, the search is backtracked to P_2 and branched into P_4 by fixing h_1 to a second discrete value. (It has been fixed to a discrete value at node P_3.) P_4 is searched forward along P_5 until it is fathomed at P_6 and P_7, and

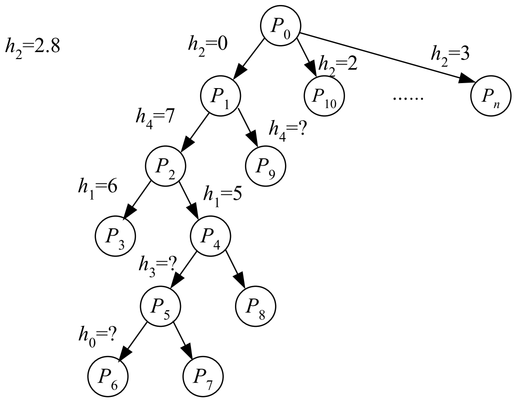

Fig. 1 An example of a depth-first width-recursive tree

the search then backtracks to P_4 and switches to search along P_8. The process of backtracking, switching, and searching forward is repeated until all nodes are either fathomed or have two children.

The width of the tree is increased recursively by one increment at a time until a predefined tree width, L, is reached.

4 Optimizing the Filter Coefficients to Generate the Tree Nodes

When using the depth-first width-recursive search to optimize FIR filters, each node is generated by optimizing all or some of the coefficient values in continuous value space. These optimization problems can be solved by linear programming. To optimize the continuous coefficients of P_0, the problem can be formulated as follows:

$$\begin{aligned} &\text{minimize} && \delta \\ &\text{subject to} && 1-\delta \le H(\omega) \le 1+\delta, \quad \text{for } \omega \in [0, \omega_p] \\ &&& -\frac{\delta_s \delta}{\delta_p} \le H(\omega) \le \frac{\delta_s \delta}{\delta_p}, \quad \text{for } \omega \in [\omega_s, \pi], \end{aligned} \tag{3}$$

where δ is the passband ripple to minimize, ω_p and ω_s are the passband and stopband edges, δ_p and δ_s are the relative passband and stopband ripple requirements, and $H(\omega)$ is the frequency response of the filter determined by (2). The variables to be optimized in the above linear programming problem are δ and h_n for $0 \le n \le \lfloor \frac{N-1}{2} \rfloor$.

For other nodes of the tree, some coefficients have been fixed to discrete values and the remaining filter coefficients are re-optimized in the continuous space. Let N_1 and N_2 be the coefficient index sets satisfying $N_1 \cap N_2 = \phi$ and $N_1 \cup N_2 = \{0, 1, \ldots, \lfloor \frac{N-1}{2} \rfloor\}$. Assume that the coefficient h_n for $n \in N_2$ have been fixed to h'_n, whereas the coefficients h_n for $n \in N_1$ are to be optimized in the continuous space. Thus, the problem to minimize the NPRM can be formulated as follows:

$$\begin{aligned} &\text{minimize} && \delta/\beta \\ &\text{subject to} && \beta-\delta-H_2(\omega) \le H_1(\omega) \le \beta+\delta-H_2(\omega), \quad \text{for } \omega \in [0, \omega_p] \\ &&& -\frac{\delta_s \delta}{\delta_p} - H_2(\omega) \le H_1(\omega) \le \frac{\delta_s \delta}{\delta_p} - H_2(\omega), \quad \text{for } \omega \in [\omega_s, \pi], \end{aligned} \tag{4}$$

where β is the passband gain. δ/β defines the NPRM. In (4)

$$H_1(\omega) = \sum_{n \in N_1} h_n \operatorname{Trig}(\omega, n), \tag{5}$$

and

$$H_2(\omega) = \sum_{n \in N_2} h'_n \operatorname{Trig}(\omega, n). \tag{6}$$

The variables to be optimized are δ, β and h_n for $n \in N_1$. This problem can be optimized by transforming it to a serial linear programming problem [12].

5 Dynamically Expanding Subexpression Basis Set and Subexpression Space

Since there are no guaranteed ways to find an optimum basis set before we obtain the filter coefficients, the basis set is initialized to the zeroth order set, i.e., $S_0 = \{\pm 1\}$. Basis set S_0 is associated with the root node P_0 in the tree shown in Fig. 1. Along with the generation of the nodes of the tree, each node is associated with a basis set, which is either inherited from its parent or expanded from its parent's basis set.

Assume that we have a node P_m associated with a basis set S_m. Before the children of P_m may be generated, the additional information necessary to be prepared for P_m is as follows. First, we need a coefficient to be quantized, say h_n, and second, a candidate list including L candidates, where L is a pre-specified tree width; these L candidates must satisfy two requirements: (1) they are discrete values in the subexpression space supported by the basis set S_m using not more than one adder; and (2) they are nearest to the continuous coefficient value h_n. The candidates are sorted by adder cost in ascending order; if the adder costs for multiple candidates are the same, they are sorted in ascending order by the distance between the candidates and h_n.

With the above information, L children of P_m may be generated, one at each time when the search traverses to the node P_m. Each child adopts one discrete value from the candidate list of P_m in order, and the remaining un-fixed coefficients are re-optimized in the continuous space, respectively. If the discrete value adopted by the child does not use any adder, the basis of P_m is inherited by this child; otherwise, the basis set of this child is expanded from that of P_m by including one more basis. The order of the basis set of the child in the latter case is increased by one from that of P_m.

We shall illustrate our approach by using an example. Suppose that P_0 has been obtained and a decision has been made to fix the value of h_2. See Fig. 1. Assume that the predefined tree width, L, is 6, and assume also that the continuous value of h_2 is 2.8. The closest L integers supported by S_0 using not more than 1 adder are $0, 2, 4, 1, 3$ and 5 after sorting. The corresponding adder costs to generate these candidates are -2, 0, 0, 0, 1, and 1, respectively.

It is interesting to note that the cost to generate integer 0 is -2. This happens because when a coefficient value is zero, the adder used to add the tap signal into the delay chain can be removed (see Fig. 2). For a symmetrical impulse response filter, coefficient values appear in pairs, and so two adders are saved for every zero value coefficient. The adders used to add the tap signal into the delay chain are called the structural adders, whereas the adders used to generate the coefficient values are called multiplier block (MB) adders. We assume that the number of structure adders is a fixed number for a given filter length. The saving on the structure adder due to zero coefficients can be counted as a negative number of adders for MB adders. That is the reason why we defined that the cost to generate 0 is -2.

Thus, the first child of P_0, node P_1, is generated by fixing h_2 to 0, the first candidate in the candidate list of P_0, and re-optimizing the remaining coefficients in the continuous space when the tree width is 1. Since there is no adder cost to generate the integer value 0, the basis set of the parent node is inherited by P_1, i.e., S_0 is assigned to S_1. P_1 then is ready to generate its own children if a coefficient is selected for quantization and a candidate list for that coefficient is generated.

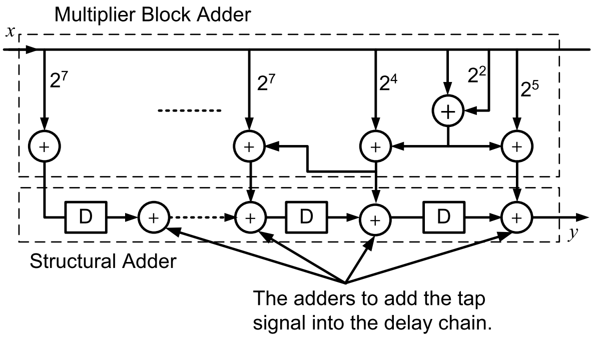

Fig. 2 The structural adders and multiplier block adders in an FIR filter

When the traverse of the tree with width 1 is complete, the width of the tree is increased to 2; if the search process backtracks to node P_0, and h_2 is now fixed to a second integer, the second candidate, 2, in the candidate list of P_0 is applied, and the remaining coefficients, again, are re-optimized. P_{10} (in Fig. 1) is generated. Just as in the case for node P_1, there is no cost to generate the integer value 2 based on the basis set S_0. Therefore, S_0 is inherited by P_{10}, i.e., $S_{10} = S_0$.

Accordingly, h_2 is fixed to integer values 4, 1, 3, and 5, respectively, when the tree width is increased to 3, 4, 5 and 6. When the integer value 3 in the candidate list of P_0 is assigned to h_2 and all remaining unfixed coefficients are re-optimized, assume that the generated node is P_n. It is noted that an adder is required to generate 3 from the basis set S_0. This generated value, 3, is included into the basis set associated with the node P_n. Thus, S_n, the basis set associated with P_n, is expanded to $\{\pm1, \pm3\}$. Note that only odd numbers can be elements for a basis set. In the case where the generated number is an even number, it is repeatedly divided by two, until it is an odd number.

Similarly, the node with fixed integer value of 5 will be associated with an expanded basis set $\{\pm1, \pm5\}$.

In the above example, the 6 nearest integers to h_2 automatically satisfy the requirements for the candidates. The following example shows a different case. Assume that the basis set of a node P_m is $S_m = \{\pm1, \pm3\}$, and the continuous coefficient selected for quantization is $h_k = 99.6$. Therefore, the sorted candidate list appears as 96, 100, 99, 98, 102 and 97 with adder cost of 0, 1, 1, 1, 1 and 1 respectively. 101 and 103 are not included in the list due to the adder cost of 2 to generate, although they are closer to h_k than some of the qualified candidates.

The restriction on the adder cost of candidates is to remove the heuristics during the generation of the candidate. Usually, there are more than one solution if an integer value requires 2 or more adders to generate from a given basis set. For example, to generate 11 based on basis set $\{\pm1\}$, a cost of 2 (adder) solution generates an intermediate value 3 or 5 (or 7, etc.) to produce 11 by $(2^3 + 3)$ or $(2^4 - 5)$. Therefore, a basis set of $\{\pm1, \pm3, \pm11\}$ or $\{\pm1, \pm5, \pm11\}$ will be generated and assigned to the corresponding node. However, it is not known which set is more beneficial when more coefficients are quantized; this causes the heuristics in the optimization of the coefficients with common subexpression sharing. Many of the algorithms regarding common subexpression sharing have dealt with this heuristic [3, 4, 21]. In our algorithm, each coefficient is generated using not more than one adder based on the existing basis set; this guarantees that once a set of discrete coefficients is obtained, its subexpression representation is optimum [5], i.e., the number of adders required to synthesize this set of coefficients is minimum.

The restriction on the adder cost of candidates may exclude some nearer candidates, but the resulting subexpression space is still large enough to get sufficient candidates. For example, in a 10-bit wordlength case, a basis set $\{\pm1, \pm3\}$ supports a 232-element space when not more than one adder is used [27].

6 Criteria for Efficient Search

To efficiently search the depth-first width-recursive tree, the coefficient selected for quantization at each node and the criteria to fathom a node must be determined.

Determine the Coefficient to be Quantized

Introduced in Sect. 4, the subexpression basis set is dynamically updated in such a way that the basis set of any node is either identical to its parent's basis set, or expanded from its parent's basis set by including only a new basis. This means that the coefficients are quantized in the subexpression space constructed from their respective parent's basis set using not more than $K = 2$ subexpression terms. Meanwhile, in the tree search, it is expected that the discrete values assigned to the coefficient which is selected for quantization are in the vicinity of the continuous coefficient value. If a coefficient has a large magnitude, but the basis set order is low, the quantization error of the fixed values will be large. For example, assume that the continuous coefficient value is 99.6, and the basis set is $\{\pm1\}$. The 6 qualified candidates are 96, 90, 112, 80, 124 and 126 (before sorting). The magnitude of the quantization errors are 3.6, 9.6, 12.4, 19.6, 24.4 and 26.4, respectively. In contrast, if the coefficient magnitude is small, say 2.8, the 6 qualified candidates supported by the same basis set are 3, 2, 4, 1, 5 and 0. The corresponding quantization errors are 0.2, 0.8, 1.2, 1.8, 2.2 and 2.8, respectively. They are much smaller than the quantization error to quantize the large magnitude coefficient. Therefore, to keep the quantization error small, the small magnitude coefficients are quantized earlier. During the course of quantizing the small magnitude coefficients, the basis sets are expanded as new bases are generated. Thus, the basis sets would have expanded to a fairly large size during the quantization of large magnitude coefficients. Hence, the quantization error to fix the large magnitude coefficient is small.

In other words, when small coefficients are quantized earlier, there is a greater possibility that the nearest L integer values each requires no more than one adder to generate. When larger coefficients are quantized, the basis set has been expanded; the possibility that the nearest integers each requires not more than one adder to generate is still high. Thus, when coefficients are quantized in such an order, the adder cost restriction on the candidates enforced in Sect. 5 does not cause significant reduction in the permissible discrete coefficient values.

Criteria to Fathom a Node

A node is fathomed in 3 cases. First, a node is fathomed if all coefficients are fixed to a value of subexpression space, i.e., a discrete solution is generated.

Second, a new node is generated when a selected coefficient in its parent node is fixed to a discrete value in the candidate list and the remaining unquantized coefficients are re-optimized. If the resulting NPRM of the filter of the new node is larger than that of the given specifications, further exploration of the new node will not produce any node which can meet the specifications; such new nodes are fathomed.

Last, if a discrete solution using M adders (and meeting the filter specifications) has been generated during the tree search, with the exception of a special case, any node requires that not less than M adders may be fathomed. The reason is that, in general, more adders are required when more numbers are discretized. (The number of adders required by a node is equal to the order of the associated basis set, and therefore can be computed and tracked easily.) However, a special case occurs if a coefficient is fixed at 0. As we have discussed in Sect. 4, zero coefficient contributes -2 adders. Even if the number of adders used to generate a node is more than M adders, say $M+1$, it is still possible to generate a node using $M+1-2R$ adders if R more coefficients are fixed to 0 in the further search procedure. In practice, only a limited number of coefficients may be 0 concurrently, while the overall filter still meets the filter specifications.

Therefore, the last condition to fathom a node is when the number of adders used for one node is larger than $M+2Z-2T$, where M is the current minimum number of adders used to generate a discrete solution meeting the filter specifications, Z is the possible maximum number of concurrent zero coefficients, and T is the number of coefficients which has been fixed to zero at the current node. M can be initialized to a positive infinite number or any number known a priori, and it is updated when a better discrete solution is produced. T may be tracked and updated for each node during the search procedure. Z is determined with the help of linear programming before the search starts. First, each coefficient is determined individually if its value may be 0 but the overall filter still meets the specification; then Z may be determined by examining how many of these coefficients may be concurrently set to 0 but the overall filter still meets the specification. Generally, Z is small, say 2 or 3, if the filter length is not unreasonably longer than the required minimum length. The computation overhead to find the value Z is negligible compared with that of the tree search.

7 Numerical Examples

Several FIR filters are designed using the proposed optimization techniques. As shown in [27], the approaches of the second category subexpression optimization are superior to all existing techniques in the first category, this section compares the qualities of the results only with those obtained using techniques reported in [6, 24, 25, 27].

Table 1 Filter specifications for $G1$, $Y1$, $S1$ and $Y2$

Filters	ω_p	ω_s	δ_p	δ_s	EWL*
$G1$	0.2π	0.5π	0.01	0.01	8
$Y1$	0.3π	0.5π	0.00316	0.00316	10, 12
$S1$	0.3π	0.5π	0.0157	0.0066	9
$Y2$	0.4π	0.51π	0.01	0.001	11

*EWL: Effective WordLength (excluding sign bit)

Table 2 Number of adders used to implement the filters

	Proposed/Best in Literature				
	$G1$	$Y1$		$S1$	$L3$
Filter order	**15**/15 [6]	**29**/29 [25]	**27**/27 [24]	**23**/24 [27]	**35**/35 [27]
Structure adder	**13**/13 [6]	**23**/23 [25]	**21**/ 21 [24]	**19**/24 [27]	**31**/35 [27]
MB adder	**2**/2 [6]	**6**/7[25]	**8**/11 [24]	**4**/4[27]	**5**/3 [27]
Total	**15**/15 [6]	**29**/30 [25]	**29**/32 [24]	**23**/28 [27]	**36**/38 [27]

Example 1

Four filters, taken from the literature and labeled as $G1$ [6], $Y1$ [25], $S1$ [19, 27] and $L3$ [13], are designed. The filter specifications of $G1$, $Y1$ and $S1$ are tabulated in Table 1. The specifications of $L3$ refer to the example 3 in [13]. The number of adders used, including the structure adder and MB adders, obtained using the proposed algorithm, are listed in Table 2. The best results presented in the previous literature are also tabulated for comparison. Table 2 shows that the proposed algorithm produces results better than, or as good as, those obtained using techniques in [6, 24, 25, 27], in the sense of the number of adders.

The algorithm presented in [6] obtained optimum solution for filter $G1$ in terms of the number of adders. Our algorithm produces two results, of which one is the same as that reported in [6], and the other one uses the same number of adders but with a slightly better frequency response performance; the coefficient values are tabulated in Table 3. The computational time to complete the tree search with a width 20 is 526 seconds, while the best solution is generated at 104 seconds. For comparison, the general computational time to optimize a filter with 12 coefficients and 10 bit wordlength using the algorithm in [6] is within one day.

For $Y1$, two designs with different filter orders and coefficient wordlengths are optimized. In both cases, the results obtained using the proposed algorithm are better than the best published in the literature [24, 25], as shown in Table 3.

The filter coefficients obtained using the proposed algorithm for filters $Y1$, $S1$ and $L3$ are also listed in Table 3.

From Table 3, it can be seen that some coefficients for $S1$ and $L3$ are trivial; this reduces the number of structural adders. As a result, the total number of adders is reduced. This is the advantage of the proposed algorithm over the B&B MILP.

Table 3 Subexpression coefficients of filters of $G1$, $Y1$, $S1$ and $L3$

Filter $G1$, $h(n) = h(15-n)$ for $8 \leq n \leq 15$

Passband gain: 634.67, Impulse Response $\times 512$

$h(7) = 27 \times 2^3$	$h(5) = 5 \times 2^2$	$h(3) = -27 \times 2^0$	$h(1) = 5 \times 2^1$
$h(6) = 1 \times 2^7$	$h(4) = -1 \times 2^5$	$h(2) = 0$	$h(0) = 5 \times 2^0$

Filter $Y1$, $h(n) = h(29-n)$ for $15 \leq n \leq 29$

Passband gain: 1400.27, Impulse Response $\times 1024$

$h(14) = (2^9 + 11) \times 2^0$	$h(10) = -3 \times 2^4$	$h(6) = -3 \times 2^3$	$h(2) = 0$
$h(13) = (9 \times 2^5 - 11) \times 2^0$	$h(9) = 9 \times 2^2$	$h(5) = -11 \times 2^0$	$h(1) = -1 \times 2^2$
$h(12) = 0$	$h(8) = 11 \times 2^2$	$h(4) = 1 \times 2^3$	$h(0) = -1 \times 2^0$
$h(11) = -(9 \times 2^1 + 9) \times 2^2$	$h(7) = 0$	$h(3) = 9 \times 2^0$	

Filter $Y1$, $h(n) = h(27-n)$ for $14 \leq n \leq 27$

Passband gain: 5180.94, Impulse Response $\times 4096$

$h(13) = (15 \times 2^7 + 15) \times 2^0$	$h(9) = -11 \times 2^4$	$h(5) = -45 \times 2^1$	$h(1) = 0$
$h(12) = 1 \times 2^{10}$	$h(8) = (45 \times 2^1 + 45) \times 2^0$	$h(4) = -5 \times 2^3$	$h(0) = -15 \times 2^0$
$h(11) = 0$	$h(7) = 81 \times 2^1$	$h(3) = 15 \times 2^1$	
$h(10) = -(15 \times 2^5 - 81) \times 2^0$	$h(6) = 0$	$h(2) = 1 \times 2^5$	

Filter $S1$, $h(n) = h(23-n)$ for $12 \leq n \leq 23$

Passband gain: 686.07, Impulse Response $\times 512$

$h(11) = 1 \times 2^8$	$h(8) = -(3 \times 2^4 + 5) \times 2^0$	$h(5) = 5 \times 2^2$	$h(2) = -3 \times 2^1$
$h(10) = 17 \times 2^3$	$h(7) = -3 \times 2^3$	$h(4) = 0$	$h(1) = 1 \times 2^1$
$h(9) = 0$	$h(6) = 1 \times 2^4$	$h(3) = -3 \times 2^2$	$h(0) = 3 \times 2^0$

Filter $L3$, $h(n) = h(35-n)$ for $18 \leq n \leq 35$

Passband gain: 1003.3, Impulse Response $\times 1024$

$h(17) = 15 \times 2^4$	$h(12) = -3 \times 2^4$	$h(7) = 3 \times 2^3$	$h(2) = -1 \times 2^3$
$h(16) = 3 \times 2^6$	$h(11) = -19 \times 2^1$	$h(6) = 1 \times 2^3$	$h(1) = 0$
$h(15) = (15 \times 2^3 - 1) \times 2^0$	$h(10) = -19 \times 2^0$	$h(5) = 0$	$h(0) = 11 \times 2^0$
$h(14) = 19 \times 2^1$	$h(9) = 1 \times 2^3$	$h(4) = -15 \times 2^0$	
$h(13) = -3 \times 2^3$	$h(8) = 3 \times 2^3$	$h(3) = -3 \times 2^2$	

In the proposed algorithm, small value coefficients may be quantized to 0 during the search procedure, whereas in the B&B MILP, it is very hard to know which coefficient should be set to 0 before the algorithm starts.

This example shows that the proposed algorithm generates better results than other non-optimum algorithms. Compared with the global optimum algorithm in the design of filters with lower orders and small coefficient wordlength, the proposed technique produces results that are not worse and with a reduced computational time.

Table 4 Subexpression coefficients of filter of $Y2$

Filter $Y2$, $h(n) = h(49-n)$ for $25 \leq n \leq 49$

Passband gain: 2596.25, Impulse Response $\times 2048$

$h(24) = (1 \times 2^9 + 21) \times 2^1$	$h(16) = -23 \times 2^1$	$h(7) = -5 \times 2^1$
$h(23) = (223 \times 2^1 + 23) \times 2^0$	$h(15) = 3 \times 2^4$	$h(6) = 3 \times 2^2$
$h(22) = -(3 \times 2^5 + 23) \times 2^0$	$h(14) = 3 \times 2^4$	$h(5) = 3 \times 2^2$
$h(21) = -(1 \times 2^8 - 33) \times 2^0$	$h(13) = -5 \times 2^2$	$h(4) = -3 \times 2^0$
$= -223 \times 2^0$	$h(12) = -5 \times 2^3$	$h(3) = -1 \times 2^3$
$h(20) = 1 \times 2^3$	$h(11) = 3 \times 2^0$	$h(2) = 0$
$h(19) = 69 \times 2^1$	$h(10) = 15 \times 2^1$	$h(1) = 3 \times 2^1$
$h(18) = 33 \times 2^0$	$h(9) = 3 \times 2^1$	$h(0) = 1 \times 2^2$
$h(17) = -21 \times 2^2$	$h(8) = -5 \times 2^2$	

Example 2

The second example is designed to show the capability of the proposed algorithm for optimizing filters with relatively higher order and wider bit width. The specification of the second example, labeled as $Y2$, is listed in Table 1. The minimum length of an FIR filter meeting the specification is 49. To allow some margin for coefficient quantization, a filter of length 50 is optimized in subexpression space. The coefficient effective wordlength is restricted to 11 bits (excluding sign bit).

The resulting filter meeting the filter specification has 48 non-zero coefficients and 11 adders are required to generate these coefficients in subexpression space; this corresponds to $47 + 11 = 58$ adders in total to implement the filter. The filter coefficients are listed in Table 4.

The optimization time for problems with similar size varies heavily, but as an estimation the proposed algorithm can be used with up to 11 coefficient bits and 25 distinct coefficients, and the solving time is from a few hours to a few tens of hours.

Existing techniques capable of designing the above filter include the B&B MILP algorithm. To use the B&B MILP algorithm to design the same filter, a 7th order continuous basis set [27] is used. A result using 63 adders (in total) is generated after the number of subexpression terms allocated to each coefficient is carefully adjusted.

8 Computational Complexity

It has been shown in [16] that the computational complexity of the depth-first width-recursive tree search algorithm is exponentially proportional to the filter length. In this section, we examine the computational complexity with respect to the wordlength of the coefficient.

The most time-consuming task in the tree search algorithm is to solve the linear programming problem (LPP) at each node of the tree. Assuming that the computation time to solve an LPP in our algorithm is a constant, the computational complexity is measured by the number of LPPs to be solved in the search procedure.

Consider the design of a filter with N' distinct coefficients and with a predefined tree width of L. If the tree is traversed completely, i.e., each node has L children and the node is fathomed only if all coefficients have been quantized, the number of LPPs to solve is approximately $(L-1)L^{N'-1}$; this number is independent of the specified coefficient wordlength Q and the initial value of the total number of allowed adders to use, M. However, the actual number of LPPs to be solved is much smaller than this because of the techniques introduced in Sect. 6.

The reduction of the number of problems to solve is not significantly affected by the initial value of M, because the quantized values for each coefficient are sorted by the adder costs in the candidate list. Therefore, the algorithm always tries to find a discrete solution as early as possible to use as few adders as possible. Once a discrete solution is obtained, M is updated. In contrast, the reduction of the number of LPPs to solve is significantly affected by the coefficient wordlength. For example, in the design of $G1$ where $N' = 8$, if the tree width is set to 20, the complete traverse of the tree will produce 1.6×10^{10} nodes, where each node corresponds to an LPP. When the wordlength is specified as 9 bits, the actual number of nodes produced is 30316. However, when the wordlength is increased by 1 bit to 10 bits, the actual number of nodes produced increases to 1.7×10^5. This happens because when the wordlength is increased by 1 bit, more of the nodes generated hold feasible solutions. The newly produced nodes will further produce more feasible children. In the above example, when the wordlength is increased by 1 bit, the number of nodes is increased by more than 5 times.

In the above comparison, the tree width L is unchanged when the wordlength is increased; this actually reduces the search range, because the quantization step decreases with increasing wordlength. To maintain the search range, L must be increased for each bit increment. Thus, the increment in the number of nodes is even greater.

To maintain the search range, the predefined tree width is doubled for every increment in wordlength. For wordlengths of 8, 9, 10 and 11 bits in the design of $G1$, the predefined tree widths are set to be 10, 20, 40 and 80, respectively. The plots of the numbers of newly generated nodes of each width for the 4 designs are shown in Fig. 3.

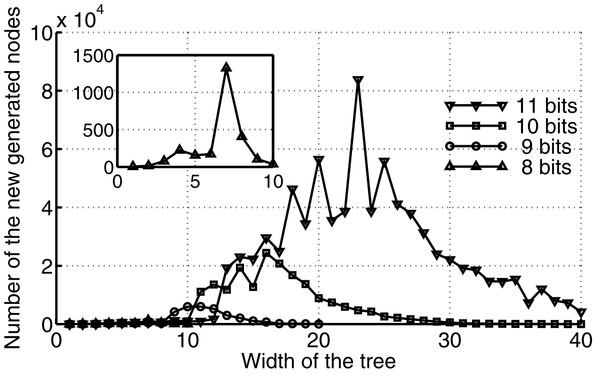

Fig. 3 The plots of the width versus the number of new generated nodes

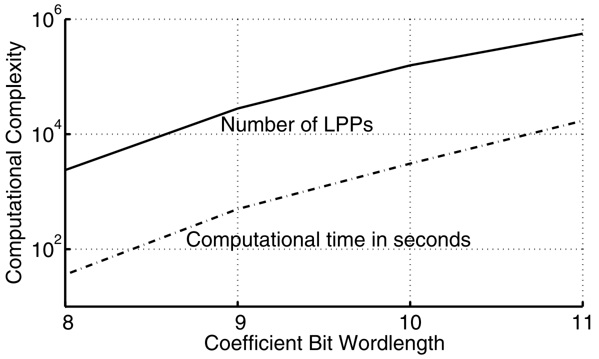

Fig. 4 The plot of the wordlength versus the computational complexity

From our experience, the necessary width of the tree to be searched can be determined by monitoring the number of new nodes produced in completing the search of each width of the tree. Generally, the number of new nodes generated increases with increasing tree width at the beginning. Thereafter, the number maintains relatively constant. After that, the number drops quickly with increasing width. If the number of newly generated nodes drops to less than half of the maximum number, from our experience, it is unlikely that better results can be produced, and the search may be terminated. From Fig. 3, it can been seen that the search may be stopped when the search for a width 8, 14, 20 and 26 is completed for a wordlength of 8, 9, 10 and 11 bits, respectively. In general, searching beyond these tree widths does not produce any better results. Therefore, the total numbers of nodes generated up to width 8, 14, 20 and 26 represent the computational complexity for wordlength 8, 9, 10 and 11 bit designs. The plot of the wordlength versus the total number of LPPs is shown in Fig. 4. The corresponding time to complete the optimization of these LLPs is also shown in Fig. 4 in seconds.

From Fig. 4, it can be seen that the increase of the computational complexity with increasing wordlength is slightly less than exponential.

9 Conclusion

In this paper, a depth-first width-recursive tree search algorithm is proposed to optimize FIR filters in subexpression space. The subexpression basis set is updated and expanded if necessary during the optimization procedure, while initially the basis set is a zeroth order set. Thus, the newly produced bases can be used to generate the coefficient values quantized later. This algorithm is able to optimize filters with length up to 50 and with bit width up to 11 bits. Although it is not guaranteed that the minimum number of adders will be achieved, the result obtained using the proposed algorithm is better than those obtained using other non-optimum algorithms in terms of the total number of adders required. On the other hand, for any discrete coefficient set obtained using the proposed algorithm, the number of adders required to synthesize this set of coefficient is guaranteed to be minimum. Compared with the global optimum algorithm, the proposed algorithm is able to cope with longer filters.

References

1. N. Boullis, T. Tisserand, Some optimizations of hardware multiplication by constant matrices. IEEE Trans. Comput. **54**, 1271–1282 (2005)
2. D.R. Bull, D.H. Horrocks, Primitive operator digital filters. IEE Proc. G **138**, 401–412 (1991)
3. A.G. Dempster, M.D. Macleod, Use of minimum-adder multiplier blocks in FIR digital filters. IEEE Trans. Circuits Syst. II **42**, 569–577 (1995)
4. O. Gustafsson, A difference based adder graph heuristic for multiple constant multiplication problems, in *Proc. IEEE ISCAS'07, New Orleans*, LA, 2007, pp. 1097–1100
5. O. Gustafsson, Lower bounds for constant multiplication problems. IEEE Trans. Circuits Syst. II **54**, 974–978 (2007)
6. O. Gustafsson, L. Wanhammar, Design of linear-phase FIR filters combining subexpression sharing with MILP, in *Proc. MWSCAS'02*, 2002, pp. 9–12
7. O. Gustafsson, L. Wanhammar, ILP modeling of the common subexpression sharing problem, in *Proc. IEEE ICECS'02*, Dubrovnic, Croatia, 2002, pp. 1171–1174
8. R.I. Hartley, Subexpression sharing in filters using canonic signed digit multipliers. IEEE Trans. Circuits Syst. II **43**, 677–688 (1996)
9. B.-R. Horng, H. Samueli, Willson, Jr., A.N., The design of two-channel lattice structure perfect-reconstruction filter banks using power-of-two coefficients. IEEE Trans. Circuits Syst. I **40**, 497–499 (1993)
10. H.-J. Kang, I.-C. Park, FIR filter synthesis algorithms for minimizing the delay and the number of adders. IEEE Trans. Circuits Syst. II **48**, 770–777 (2001)
11. D.M. Kodek, Design of optimal finite wordlength FIR digital filters using integer programming techniques. IEEE Trans. Acoust. Speech Signal Process. **ASSP-28**, 304–308 (1980)
12. Y.C. Lim, Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude. IEEE Trans. Circuits Syst. **37**, 1480–1486 (1990)
13. Y.C. Lim, S.R. Parker, Discrete coefficient FIR digital filter design based upon an LMS criteria. IEEE Trans. Circuits Syst. **30**, 723–739 (1983)
14. Y.C. Lim, S.R. Parker, FIR filter design over a discrete power-of-two coefficient space. IEEE Trans. Acoust. Speech Signal Process. **ASSP-31**, 583–591 (1983)
15. Y.C. Lim, R. Yang, D.N. Li, J.J. Song, Signed power-of-two term allocation scheme for the design of digital filters. IEEE Trans. Circuits Syst. II **46**, 577–584 (1999)
16. Y.C. Lim, Y.J. Yu, A width-recursive depth-first tree search approach for the design of discrete coefficient perfect reconstruction lattice filter bank. IEEE Trans. Circuits Syst. II **50**, 257–266 (2003)
17. K. Nakayama, Permuted difference coefficient realization of FIR digital filters. IEEE Trans. Acoust. Speech Signal Process. **ASSP-30**, 269–278 (1982)
18. M. Potkonjak, M.B. Shrivasta, A.P. Chandrakasan, Multiple constant multiplication: Efficient and versatile framework and algorithms for exploring common subexpression elimination. IEEE Trans. Comput. Aided Des. **15**, 151–161 (1996)
19. H. Samueli, An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients. IEEE Trans. Circuits Syst. **36**, 1044–1047 (1989)
20. A.P. Vinod, E.M.-K. Lai, A.B. Premkuntar, C.T. Lau, FIR filter implementation by efficient sharing of horizontal and vertical common subexpressions. Electron. Lett. **39**, 251–253 (2003)
21. Y. Voronenko, M. Püschel, Multiplierless multiple constant multiplication. ACM Trans. Algorithms **3**, 1–38 (2007)
22. Y. Wang, K. Roy, CSDC: A new complexity reduction technique for multiplierless implementation of digital FIR filters. IEEE Trans. Circuits Syst. I **52**, 1845–1853 (2005)
23. F. Xu, C.-H Chang, C.-C. Jong, Contention resolution algorithm for common subexpression elimination in digital filter design. IEEE Trans. Circuits Syst. II **52**, 695–700 (2005)
24. F. Xu, C.-H Chang, 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. **26**, 1898–1907 (2007)
25. J. Yli-Kaakinen, T. Saramäki, A systematic algorithm for the design of multiplierless FIR filters, in *Proc. IEEE ISCAS'01*, Sydney, Australia, 2001, pp. 185–188
26. Y.J. Yu, Y.C. Lim, A novel genetic algorithm for the design of a signed power-of-two coefficient quadrature mirror filter lattice filter bank. Circuit Syst. Signal Process. **21**, 263–276 (2002)
27. Y.J. Yu, Y.C. Lim, Design of linear phase FIR filters in subexpression space using mixed integer linear programming. IEEE Trans. Circuits Syst. I **54**, 2330–2338 (2007)
28. Y.J. Yu, Y.C. Lim, Roundoff noise analysis of signals represented using signed power-of-two terms. IEEE Trans. Signal Processing **55**, 2122–2135 (2007)