Single-stage and cascade design of high order multiplierless linear phase FIR filters using genetic algorithm

Ye, Wen Bin; Yu, Ya Jun

2013

Ye, W. B., & Yu, Y. J. (2013). Single-Stage and Cascade Design of High Order Multiplierless Linear Phase FIR Filters Using Genetic Algorithm. IEEE Transactions on Circuits and Systems I-Regular Papers 60(11), 2987 - 2997.

https://hdl.handle.net/10356/103634

https://doi.org/10.1109/TCSI.2013.2256211

Single-Stage and Cascade Design of High Order Multiplierless linear phase FIR Filters Using Genetic Algorithm

Wen Bin Ye, *Student Member, IEEE* and Ya Jun Yu, *Senior Member, IEEE*

***Abstract*— In this work, a novel genetic algorithm (GA) is proposed for the design of multiplierless linear phase finite impulse response (FIR) filters. The filters under consideration are of high order and wide coefficient wordlength. Both the single-stage and cascade form are considered. In a practical filter design problem, when the filter specification is stringent, requiring high filter order and wide coefficient wordlength, GAs often fail to find feasible solutions, because the discrete search space thus constructed is huge and the majority of the solution candidates therein can not meet the specification. In the proposed GA, the discrete search space is partitioned into smaller ones. Each small space is constructed surrounding a base discrete coefficient set which is obtained by a proposed greedy algorithm. The partition of the search space increases the chances for the GA to find feasible solutions, but does not sacrifice the coverage of the search. The proposed GA applies to the design of single-stage filters. When a cascade form filter is designed, for each single-stage filter meeting the filter specification generated during the course of GA, an integer polynomial factorization is applied. Design examples show that the proposed GA significantly outperforms existing algorithms dealing with the similar problems in terms of design time, and the hardware cost is saved in most cases.**

***Index Terms*— finite impulse response (FIR), genetic algorithm, cascade form, low hardware cost.**

I. Introduction

Multiplierless linear phase finite impulse response (FIR) filters have been very popular for the past decades, since the coefficient multipliers are implemented by adders and hard-wired shifts, resulting in a low hardware cost. Many algorithms [1]–[17] have been proposed to minimize the number of adders used to synthesize the adder-shift network to minimize the hardware cost. In earlier time, such algorithms [1]–[11] are applied to a set of discrete coefficient values which already meet a given filter specification, to find a synthesis of the coefficient values using as few adders as possible. However, in filter design problems, the set of coefficient values meeting the given specification usually is not unique. It is possible to find some other sets of coefficient values which also meet the given specification and can be synthesized using less number of adders. Therefore, in recent years, many designers have proposed algorithms incorporating the synthesis of coefficient values into the search of discrete coefficients for a given filter specification [12]–[17].

While the algorithms in the later group definitely generate results with lower hardware cost, they have a common problem, i.e., requiring extremely long computation time. This problem becomes more serious when the filter order is high and/or the coefficient wordlength is wide, since the computation time of most of these algorithms increases exponentially along with the increasing of filter orders and coefficient wordlengthes. A recently developed algorithm [18] confines the search space of each coefficient to the vicinity, for example the 2 closest discrete values, of the middle values in the feasible range, and thus is capable of designing long filters.

In order to extend the search space and meanwhile restrict the search time to a tolerable level, in this paper, a novel GA is proposed for the design of long filters and filters requiring coefficients with wide wordlength. GAs are artificial intelligence techniques based on the principle of "survival of the fittest" [19]. As a stochastic algorithm, a GA can be used to solve complicated problem with a huge search space, and thus is suitable for the problem under consideration. However, a drawback of GAs is that they may fail to find a feasible solution if the majority of the solution candidates in the search space are infeasible. In the filter design problem, when the filter specification is stringent and the hardware restriction is high, existing GAs such as [20]–[22] often cannot find a feasible solution.

In order to overcome these limitations, in the proposed GA, the discrete search space is partitioned into smaller ones. Each small space is constructed surrounding a base discrete solution which is obtained by a proposed greedy algorithm with different passband gain. This increases the chances for the GA to find feasible solutions, but does not sacrifice the coverage of the search. In addition, the search in the multiple spaces can run in parallel, and thus the computation time for the design of filters under consideration reduces significantly. Although a mixed integer linear programming (MILP) based algorithm [23] has been developed for the parallel optimization of FIR filters with signed power-of-two coefficients, the programming and the partitioning of the problem are complicated, and therefore, no attempts have been made for the design of filters sharing adder-shift network among coefficients. Moreover, in the proposed GA, an encoding technique is tailored for the partitioned search spaces, and a new fitness function putting more weights on the minimization of hardware cost

Manuscript received on Sep. 20, 2012; revised on Dec. 07, 2012 and Feb. 16, 2013. W. B. Ye and Y. J. Yu are with the school of Electrical and Electronic Engineering, Nanyang Technological University Email:yewe0003@e.ntu.edu.sg; eleyuyj@pmail.ntu.edu.sg.

are proposed.

The preliminary results of the proposed technique have been presented in [24]. Compared with [24], there are three major contributions in this paper. First, a more efficient subspace partitioning technique is proposed. Second, adaptive crossover and mutation rates are introduced such that the search efficiency is improved and the fittest solutions are preserved with the highest probability during the generational process. The last and most important is that a novel integer polynomial factorization technique is proposed such that the proposed GA can be used to design cascade form FIR filters; this technique circumvents the difficulty in the determination of the filter orders and coefficient effective wordlength (EWL) of the subfilters. In this paper, the EWL of a coefficient refers to the wordlegnth excluding the sign bit and the leading zero bits of the coefficient value; and the EWL of a filter refers to the EWL of the coefficient with the maximum magnitude in the filter.

Design examples show that the proposed GA uses much less time in the design of long filters and filters with wide coefficient wordlength. The hardware costs of the filters designed by the proposed GA are less than that designed by the other techniques in most cases, for both the single-stage and cascade form filters.

It should be noted that the proposed algorithm only aims to the design of linear phase FIR filters. For linear phase FIR filters, once the filter order is determined, the phase shift for each particular frequency is fixed. In other words, linear phase FIR filter may not be used in applications where the phase margin is concerned.

The rest of the paper is organized as follows. In Section II, the proposed GA for the design of the single-stage filters under consideration is discussed in details. Section III presents three single-stage design examples. The computational complexity, the selection of parameters, and the superiority and limitation of the algorithm are also discussed. Section IV introduces how to extend the proposed GA to design cascaded FIR filters by the introduction of an integer polynomial factorization technique. Two sets of design examples are given in Section V to show the advantages of the GA in the design of cascade form filters. Finally, the paper is concluded in Section VI.

II. Design Single-Stage FIR Filters Using GA

In this section, first, the GA is briefly reviewed. A new partition technique is then introduced to divide the whole search space into several smaller spaces based on different passband gains. After that, adaptive crossover and mutation rates controlled by the fitness values are introduced. Finally, a new encoding form of chromosomes and a fitness function of the GA are proposed.

A. Overview of GA

A GA is an algorithm that mimics the process of natural evolutions. By such evolutions, the solution gradually evolves to the optimum one. Basically, in GAs, potential solutions of the unknowns are encoded to chromosomes and 3 genetic operations including selection, crossover and mutation are applied to these chromosomes. In detail, after initializing the first generation population of the chromosome pool, some individuals (individual chromosomes) in the existing population are selected according to a selection mechanism to breed a new generation. Those selected individuals are called "parents" and they produce "children" by crossover and mutation operators. The generational process containing selection, crossover and mutation is repeated until a prespecified termination criterion is reached.

B. Partitioning the search space

In the multiplierless filter design, if the coefficient space is nonlinear, such as the power-of-two space and subexpression space, floating the passband gain away from the unit may result in designs requiring less hardware. However, this enlarges the search space significantly.

Let N and B be the number of filter coefficients and the EWL of the filter, respectively. Apparently, after scaling the coefficient values to integers, the dynamic range of each coefficient is within $[-2^B\ 2^B-1]$. Therefore, for $\lfloor \frac{N+1}{2} \rfloor$ distinct coefficients in a linear phase FIR filter, the number of possible combinations of discrete solutions is as large as $2^{\lfloor \frac{N+1}{2} \rfloor (B+1)}$. For a moderate long and bit wide filter, for example N and B taken as 50 and 12, respectively, the number of possible discrete solutions in the whole search space will be 2^{325}. However, the majority of these solutions are infeasible, i.e., they do not meet the filter specification, when the specification is reasonably stringent. Therefore, in most cases, it is very difficult to find feasible solutions using conventional GAs. To circumvent this problem, we confine the search spaces to the neighbors of several discrete coefficient sets. The discrete coefficient sets are obtained using a successive reoptimization approach for the selected passband gains. The way to select the passband gains, G_m, for m =0,1,... , $M-1$ is discussed in the next subsection. For each G_m, the discrete coefficient set is found in the following two steps.

Step 1: The optimum solution with continuous coefficient values meeting the filter specifications with a passband gain G_m is found using a linear programming formulated as [16]:

$$\begin{aligned} &\text{Minimize}: f=\delta \\ &\text{Subject to}: G_m-\delta \le H(\omega) \le G_m+\delta, \text{for} \quad \omega \in [0,\omega_p] \\ &-(\delta_s\delta)/\delta_p \le H(\omega) \le (\delta_s\delta)/\delta_p, \text{for} \quad \omega \in [\omega_s,\pi] \end{aligned} \tag{1}$$

where ω_p, ω_s are the passband and stopband edges, δ_p, δ_s are the passband and stopband ripple tolerances of the ideal passband and stopband gains away from 1 and 0, respectively, and $H(\omega)$ is the zero-phase frequency response of the filter to be optimized, given by

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

In (2), $h(n)$ is the unknown symmetric impulse response of the filter to be optimized and Trig(ω,n) is an appropriate trigonometric function depending on the parity of N and

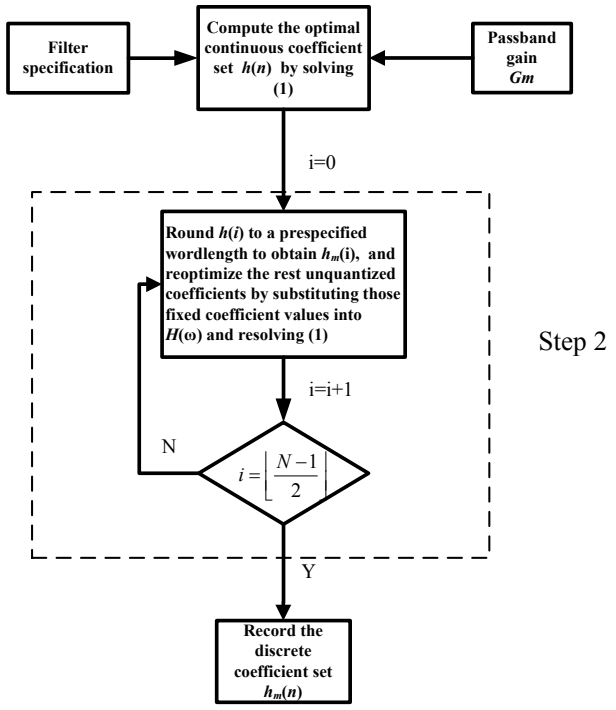

Fig. 1. The flow chart of the greedy algorithm in determining the base coefficient set

symmetry of the filter. By solving the problem formulated in (1), the obtained $h(n)$ is the impulse response of a filter whose frequency response has a passband gain G_m. Divided by G_m, the impulse response can be normalized to unit passband gain. Let an index i be zero.

Step 2: Round $h(i)$ to a prespecified wordlength. If $i = \lfloor \frac{N-1}{2} \rfloor$, meaning that every coefficient has been rounded to a discrete value, the program stops. Otherwise, reoptimize the rest unfixed coefficient by substituting those fixed coefficient values into $H(\omega)$ and re-solve (1); $i = i + 1$ and repeat Step 2.

The above method is a greedy successive reoptimization algorithm. The flow chart of the greedy algorithm to find the discrete coefficient sets is shown Fig. 1. In each iteration of Step 2, the algorithm rounds a coefficient into a discrete value and reoptimizes the rest continuous coefficients to compensate for the degradation of the frequency response. Using the above method, M sets of discrete coefficient values are obtained. Denoting the discrete coefficient set obtained using gain G_m as $h_m(n)$ for n=0,1,...$\lfloor \frac{N-1}{2} \rfloor$, to simplify the representation, $h_m(n)$ are scaled to integers and denoted as $h_{q_m}(n)$, i.e,

$$h_{q_m}(n) = h_m(n) \times 2^{S_m} \tag{3}$$

where S_m is an integer making $||\mathbf{h}_{q_m}||_\infty$ containing the pre-specified EWL; $\mathbf{h}_{q_m}$ represents the vector $[h_{q_m}(0), h_{q_m}(1),... h_{q_m}(\lfloor \frac{N-1}{2} \rfloor)]^T$ and $||\mathbf{h}_{q_m}||_\infty$ stands for the norm-∞ of the vector $\mathbf{h}_{q_m}$, i.e, the maximum magnitude of $\mathbf{h}_{q_m}$. $\mathbf{h}_{q_m}$ for m =0,1,2,...$M-1$ is named as the base discrete coefficient sets for the construction of subspaces. Last, for each passband gain G_m and the corresponding coefficient set, an $\lfloor \frac{N+1}{2} \rfloor$-dimensional search space is constructed as follows. The lower bound and upper bound, denoted as $h^l_{q_m}(n)$ and $h^u_{q_m}(n)$, of n-th dimension , for $n = 0, 1, ... \lfloor \frac{N-1}{2} \rfloor$, respectively, are defined as:

$$\begin{cases} h^u_{q_m}(n) = h_{q_m}(n) + 2^{\text{ceil}(B_m(n)/3)} - 1 \\ h^l_{q_m}(n) = h_{q_m}(n) - 2^{\text{ceil}(B_m(n)/3)} + 1 \end{cases} \tag{4}$$

where $B_m(n)$ is the EWL of the n-th coefficient. In such a way, the coefficient with a larger magnitude has a larger range for variation in the discrete coefficient space. For example, for a particular passband gain G_m, the corresponding base discrete coefficient set is $\{3, 9, 21\}$ and the corresponding EWL of each coefficient value is 2, 4 and 5, respectively. The lower bound and upper bound of each discrete coefficient are computed according to (4), and therefore, each coefficient value may vary in the range of [2,3,4], [6,7,8,9,10,11,12] and [18,19,...21...24], respectively, where the underlined values are $h_{q_m}(n)$.

C. Determining the Gains

For a given filter specification, when the passband gain is G_m or $2G_m$, the discrete coefficient sets obtained from the successive reoptimization algorithm are the same for a given EWL. Therefore, we only need to consider the passband gain varying in a range from α to 2α for any positive α. We choose α to be 0.7 as [17] and thus G_m may vary in [0.7 1.4]. First, we divide the variation range of the passband gain to M sections in [0.7 1.4] as that in [25]; in our case, M is set to 20, and thus G_m is confined to a range of Γ_m, given by [0.7+0.035m 0.7+0.035(m+1)] for m=0,1,2...$M-1$. Each range Γ_m is further discretized by a stepsize of 0.001, and thus G_m may take any value from the 35 possible values in Γ_m. Let the 35 possible values be $G_{m,j}$ for j=0,1,2...34. The discrete coefficient set for each $G_{m,j}$, denoted as $h_{qm,j}(n)$, obtained by the proposed greedy algorithm discussed in Section II-B, leads to a filter with normalized magnitude ripples $\delta_{m,j}$ computed by

$$\begin{cases} H(\omega) &= \sum_{n=0}^{\lfloor \frac{N-1}{2} \rfloor} h_{q_{m,j}}(n)\text{Trig}(\omega, n) \\ gain &= \dfrac{\max(H(\omega)) + \min(H(\omega))}{2}, \text{for} \quad \omega \in [0, \omega_p] \\ \widehat{\delta_p} &= \dfrac{\max(H(\omega)) - \min(H(\omega))}{2}, \text{for} \quad \omega \in [0, \omega_p] \\ \widehat{\delta_s} &= \max(\text{abs}(H(\omega))), \text{for} \quad \omega \in [\omega_s, \pi] \\ \delta_{m,j} &= \max(\frac{\widehat{\delta_p}}{gain}, \frac{\widehat{\delta_s}}{gain} * \frac{\delta_p}{\delta_s}) \end{cases} \tag{5}$$

Thus, G_m is selected to be $G_{m,i}$ whose corresponding $\delta_{m,i}$ is minimum among $\delta_{m,j}$ for all j=0,1,2...34. With the selected G_m for m=0,1,2...$M-1$, twenty spaces are constructed according to the method proposed in Subsection II-B, and the proposed GA is applied to each search space separately.

D. Adaptive Crossover and Mutation Rates

During the generational process, crossover and mutation are the two basic operations to generate the offsprings. The crossover and mutation rates, denoted as P_c and P_m, play a vital role in the GA algorithm. The higher P_c and P_m are, the more variations are introduced into the offsprings; this

prevents the premature convergence of the GA and increases the search efficiency. However, large P_c and P_m may lead to the solution unstable, i.e., fitter chromosomes have the same probability to be cast away during the generational process as unfitter chromosomes. Therefore, in the proposed GA, instead of keeping P_c and P_m unchanged during the whole process, P_c and P_m are adaptively adjusted according to the fitness values of the chromosomes, expressed as :

$$\begin{aligned} P_c &= \begin{cases} \lambda_1 \frac{F_{max}-F^L}{F_{max}-F_{mean}} + \lambda_0, & \text{for } F^L \geq F_{mean} \\ \lambda_1 + \lambda_0, & \text{otherwise} \end{cases} \\ P_m &= \begin{cases} \lambda_2 \frac{F_{max}-F}{F_{max}-F_{mean}} + \frac{\lambda_0}{100}, & \text{for } F \geq F_{mean} \\ \lambda_2 + \frac{\lambda_0}{100}, & \text{otherwise} \end{cases} \end{aligned} \tag{6}$$

where F_{max} and F_{mean} are the maximum and mean values of the overall fitness values in the current generation, respectively, F^L is the larger fitness value of the two parent chromosomes that are selected for crossover operation, and F is the fitness value of the chromosome that is to mutate. λ_1 and λ_2 are weighting parameters and λ_0 is a constant. According to (6), the chromosome with smaller fitness value has larger P_m and P_c; thus, their codons in the chromosomes are recombined more quickly. On the contrary, for the fittest chromosome in the current generation, their P_m and P_c are λ_0 and $\lambda_0/100$, the minimum mutation and crossover rates, so that the good codons are preserved. The selection of parameters in (6) is presented in section III-C.

E. Encoding and Initial Population

In the proposed algorithm, chromosomes are encoded to ternary strings of $\{\bar{1}, 0, 1\}$. Since the search space in the proposed GA has been confined to a small range around the base discrete coefficient set, each element in the population is a small deviation from the base discrete values. Therefore, to encode the discrete coefficient values in the population, instead of encoding the discrete coefficient values themselves, the deviations from the base discrete values are encoded to reduce the length of chromosomes. To illustrate this encoding method, the same example in Section II-B is used, where the base discrete coefficient values are $\{3, 9, 21\}$, and the lower bound and upper bound for each value are [2,4], [6,12] and [18,24], respectively. Thus, the possible largest deviations are 1, 3 and 3, respectively. Therefore, 5 digits are sufficient to encode all elements in the given search space. For example, an element of $\{4, 8, 20\}$ is encoded to be $10\bar{1}01$. If the coefficient values are directly encoded to a ternary string, 12 digits are needed.

With the above encoding technique, a chromosome with an all-zero string represents the base discrete coefficient set, and such chromosome is named as an "ancestor chromosome". Our design experiences show that for filters with length less than 35, the proposed GA is robust to find feasible solutions even with randomly produced population in most cases; however, when the filter length is longer, a good initial population is very important for the GA to find a feasible solution. In the proposed GA, half of the total population is produced by perturbing the "ancestor chromosome" and the other half is randomly produced in the respectively constructed space. The randomly produced population is to diversify the gene pool and thus helps to prevent the premature convergence.

F. Fitness Function

To minimize the hardware cost, the objective function f_g of the proposed GA is set to be the number of adders. After each chromosome is encoded, a multiple constant multiplication (MCM) algorithm is used to compute the number of adders used to synthesize the coefficient values. The *n*-dimensional reduced adder graph (RAGN) [5] algorithm is used for the reason that this algorithm usually outperforms other MCM algorithms in terms of number adders used to synthesize the discrete coefficient values (The total number of adder of the filters includes the adders to synthesize the coefficients and the structural adders used in the delay chain). However, as mentioned in Section II-B, the major elements in the search space cannot meet the given filter specification; for such chromosomes, even their corresponding coefficient sets can be synthesized using fewer adders, they should be cast away during the selection process gradually. A common way to cast away the infeasible elements is to add a penalty to the objective function when the filter specification is not satisfied, and the fitness values are thus significantly reduced for these cases. The proposed fitness function is given as,

$$F = \begin{cases} 1/f_g & \text{if } \delta_m \leq \delta_p \\ 1/(P_e \times \frac{\delta_m}{\delta_p}) & \text{if } \delta_m > \delta_p \end{cases} \tag{7}$$

Here, f_g is the total number of adders used to synthesize the filter using the coefficient values of the chromosome, δ_m is the ripple of the filter represented by the chromosome calculated as in (5), δ_p is the given ripple specification and P_e is a penalty value, given by

$$P_e = \sum_{n=0}^{\lfloor \frac{N-1}{2} \rfloor} \text{Maxaddercost}(h_{qm}(n)) \tag{8}$$

where $\text{Maxaddercost}(h_{qm}(n))$ is the maximum adder cost to synthesize the possible values of $h_{q_m}(n)$ in its range $[h^l_{q_m}(n)\ h^u_{q_m}(n)]$ given in (4). For example, if a coefficient value is in the range of [2,3,4], the corresponding adder costs to synthesize these discrete values are 0,1 and 0, respectively. So the maximum adder cost for this coefficient is 1. Computing P_e according to (8) guarantees that the fitness values of the infeasible individuals are smaller than that of the feasible individuals. Thus, during the generational process, the infeasible solutions will be gradually cast away. Meanwhile, for the coefficient sets not meeting the ripple specification, the coefficient sets with smaller ripples have bigger fitness values, ensuring that they have bigger chances to survive in the selection process. This helps the program to find out more coefficient sets meeting the given filter specification.

With the introduction of the penalty function, we do not need to apply the MCM algorithm to those coefficient sets which do not meet the given filter ripple requirement. Since it is time consuming to apply MCM algorithm to each coefficient

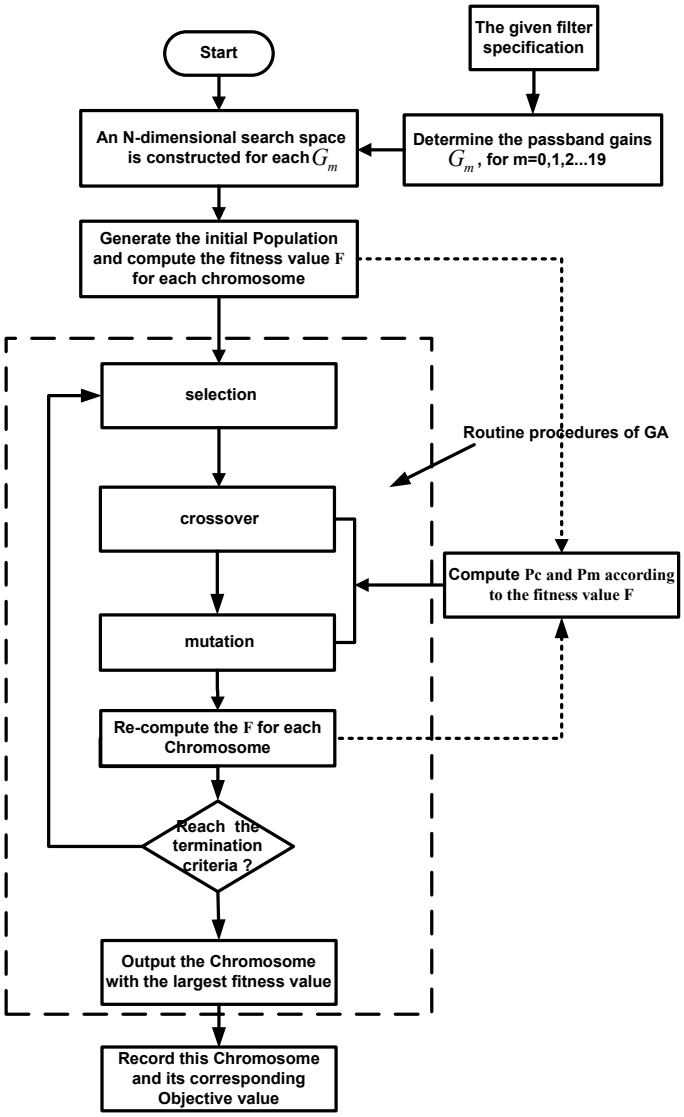

Fig. 2. The flow chart of the proposed GA

set, when the majority of the coefficient sets in the population are infeasible, the proposed fitness function can significantly reduce the runtime of the program.

G. Summary of the Proposed GA

The step-by-step design procedure is shown in Fig. 2.

1) Determine the passband gains $G_m(m$=0,1,2...19) using the method proposed in Section II-C.
2) For each G_m obtained in step 1, find the base discrete coefficient set and construct the corresponding search space. Run the GA using the fitness function given in (7) and the adaptive crossover and mutation rates given in (6). When the GA operation is terminated, the chromosome with the biggest fitness value is recorded. Since G_m has 20 different values, in total there are 20 sub-problems of the GA. Because the sub-problems of the GA are independent to each other, they can be run in parallel, resulting in a significant reduction of total runtime.
3) If the final optimum solution still does not meet the given filter specification, the filter length N or the EWL has to be increased, and the whole procedure is repeated.

TABLE I

THE FILTER SPECIFICATIONS FOR B, L1 AND C

Filters	Filter order	EWL	ω_p	ω_s	δ_p	δ_s
B (low pass)	104	8	0.2π	0.24π	0.01	0.01
L1 (high pass)	120	14	0.8π	0.74π	0.0057	0.0001
C (low pass)	324	10	0.125π	0.14π	0.005	0.005

TABLE II

RESULTS AND COMPARISON OF THE THREE BENCHMARK FILTERS, WHERE MBA AND SA ARE THE NUMBER OF MULTIPLIER BLOCK ADDERS AND STRUCTURE ADDERS

	Best published references / proposed GA		
Filters	B [18]	L1 [18]	C [18]
MBA	11 / 10	47 / 43	22 / 31
SA	100 / 98	120 / 120	306 / 306
Total adders	111 / 108	167 / 163	328 / 337

III. NUMERICAL EXAMPLE OF SINGLE-STAGE DESIGN

Three benchmark filters (B, L1 and C) are designed to show the superiority and limitation of the proposed algorithm in the design of high order and wide coefficient wordlength FIR filters. B, L1 and C are examples taken from [18], [26]. The specifications of the three filters are listed in Table I. The specifications show that the order of the three filters are all high and meanwhile filter L1 requires coefficient values with wide wordlength.

A. Simulations Results

In the proposed GA, the population size is set to 400 and the running of the GA is terminated when the generation reaches 100. The Roulette Wheel Selection [27] is used to select the chromosomes for crossover and mutation. The determination of the other GA parameters will be discussed in subsection III-C. The program is developed using Matlab. The 20 sub-problems of each example are casted to 20 computers, and each computer has two cores with CPU speed of 3.2G Hz.

The simulation results are given in Table II, where the number of the adders is given in terms of number of multiplier block adders (MBAs) and structure adders (SAs). It can be seen for the filters with length less than 150, the proposed GA generates designs using less number of adders. However, for the filter with length longer than 300, the proposed GA cannot beat the design obtained in [18] in terms of number of adders. The reasons for this is analyzed in subsection III-D. The coefficient values of the filters B and L1 obtained by the proposed GA are listed in Tables III and IV, respectively.

B. Computational Complexity

The runtime of the proposed GA for filters L1, B and C are 26m41s, 16m24s and 3h49m, respectively, when the partitioned problems are cast to 20 machines. When only one computer is used but the parallel toolbox in Matlab is exploited, the runtimes of the examples L1, B and C are 4h31m, 2h41m and 37h9m, respectively. However, the runtimes cannot accurately reflect the computational complexity of the algorithms due to various factors such as the efficiency of the

TABLE III

THE COEFFICIENT VALUES OF FILTER B

Filter B: $h(n) = h(104-n)$ for $0 \le n \le 52$; passband gain: 907.66, EWL: 8 normalized passband ripple: 0.00995; normalized stopband ripple : 0.00995																									
$h(0)$	=	-2	$h(8)$	=	-2	$h(16)$	=	-1	$h(24)$	=	4	$h(32)$	=	11	$h(40)$	=	20	$h(48)$	=	27					
$h(1)$	=	-1	$h(9)$	=	-2	$h(17)$	=	-4	$h(25)$	=	-1	$h(33)$	=	7	$h(41)$	=	24	$h(49)$	=	84					
$h(2)$	=	0	$h(10)$	=	-2	$h(18)$	=	-5	$h(26)$	=	-6	$h(34)$	=	-2	$h(42)$	=	16	$h(50)$	=	142					
$h(3)$	=	1	$h(11)$	=	0	$h(19)$	=	-4	$h(27)$	=	-9	$h(35)$	=	-11	$h(43)$	=	-2	$h(51)$	=	185					
$h(4)$	=	2	$h(12)$	=	2	$h(20)$	=	0	$h(28)$	=	-7	$h(36)$	=	-16	$h(44)$	=	-24	$h(52)$	=	200					
$h(5)$	=	2	$h(13)$	=	3	$h(21)$	=	3	$h(29)$	=	-2	$h(37)$	=	-14	$h(45)$	=	-40								
$h(6)$	=	1	$h(14)$	=	3	$h(22)$	=	6	$h(30)$	=	5	$h(38)$	=	-5	$h(46)$	=	-40								
$h(7)$	=	-1	$h(15)$	=	2	$h(23)$	=	6	$h(31)$	=	10	$h(39)$	=	9	$h(47)$	=	-18								

TABLE IV

THE COEFFICIENT VALUES OF FILTER L1

Filter L1: $h(n) = h(120-n)$ for $0 \le n \le 60$; passband gain:64608.72, EWL: 14 normalized passband ripple: 0.0055 ; normalized stopband ripple: 0.00009649																									
$h(0)$	=	5	$h(9)$	=	59	$h(18)$	=	-134	$h(27)$	=	274	$h(36)$	=	-543	$h(45)$	=	1068	$h(54)$	=	-2958					
$h(1)$	=	-13	$h(10)$	=	-30	$h(19)$	=	75	$h(28)$	=	-151	$h(37)$	=	291	$h(46)$	=	-536	$h(55)$	=	1500					
$h(2)$	=	22	$h(11)$	=	-16	$h(20)$	=	32	$h(29)$	=	-70	$h(38)$	=	150	$h(47)$	=	-384	$h(56)$	=	1620					
$h(3)$	=	-27	$h(12)$	=	60	$h(21)$	=	-139	$h(30)$	=	291	$h(39)$	=	-583	$h(48)$	=	1294	$h(57)$	=	-5833					
$h(4)$	=	23	$h(13)$	=	-80	$h(22)$	=	193	$h(31)$	=	-398	$h(40)$	=	785	$h(49)$	=	-1719	$h(58)$	=	10113					
$h(5)$	=	-7	$h(14)$	=	60	$h(23)$	=	-156	$h(32)$	=	317	$h(41)$	=	-614	$h(50)$	=	1326	$h(59)$	=	-13297					
$h(6)$	=	-18	$h(15)$	=	-2	$h(24)$	=	30	$h(33)$	=	-58	$h(42)$	=	95	$h(51)$	=	-119	$h(60)$	=	14473					
$h(7)$	=	46	$h(16)$	=	-73	$h(25)$	=	134	$h(34)$	=	-274	$h(43)$	=	563	$h(52)$	=	-1490								
$h(8)$	=	-63	$h(17)$	=	129	$h(26)$	=	-260	$h(35)$	=	522	$h(44)$	=	-1044	$h(53)$	=	2777								

computer languages, the speed of the computer and etc. In the following, the computational complexity of the algorithms are analyzed based on the number of linear programming to be solved.

In the proposed algorithm, the computation contains two parts: one is to find the base discrete coefficient set and the other is to search around the subspace constructed based on the base discrete coefficient set. Both of these two parts repeatly compute the filter ripple by solving linear programming problems. Let N, P and G be the length of the filter, the population size and the number of generations to be updated in the GA, respectively. The total number of linear programmings for the first part to find the base discrete coefficient set, expressed as LP_{N1} is given by

$$LP_{N1} = J\left\lceil \frac{N-1}{2} \right\rceil \tag{9}$$

where J is the number of passband gains in each section. In our case $J = 35$.

In the second part, the number of linear programming to be done, denoted as LP_{N2} , is determined by

$$LP_{N2} = PG \tag{10}$$

where $P = 400$ and $G = 100$ in our case. The computation time for the selection, crossover and mutations operations is much shorter than that for the linear programming and is ignored in our analysis. Let the average runtime of a linear programming be T. The overall computation time of the proposed algorithm is

$$ComputationTime = M\left(J\left\lceil \frac{N-1}{2} \right\rceil + PG\right)T \tag{11}$$

where M is the total number of partitioned subspaces.

In comparison, the total iterations needed to be run in the algorithm [18] is $L^{\lceil \frac{N-1}{2} \rceil}$, where L is the size of the refined value set. L is set to 2 in [18]. Each iteration requires two linear programmings to compute the lower bound and upper bound of the corresponding coefficient. The overall computation time of the algorithm in [18] can be expressed as

$$ComputationTime = 2\beta L^{\lceil \frac{N-1}{2} \rceil}T \tag{12}$$

where β is a factor influenced by many cutoff "mechanisms".

From (11) and (12), it can be seen that the computational complexity of the proposed algorithm increases linearly with the increasing of N, whereas that for the algorithm in [18] increases exponentially with the increasing of N. When N is large, the proposed algorithm has much lower computational complexity. The cutoff factor β may alleviate the computational complexity but it can not change the exponential relation by nature. For example, assuming that the cutoff factor β is 10^{-8} (which in practical is much larger), to design filter B (with order 104), the total computation time of the algorithm in [18] is $(1.8 \times 10^8)T$, whereas for the proposed GA it is $(8.364 \times 10^5)T$. The later one is much smaller than the former one. That is the reason why [18] need to be manually terminated after running 24h for filters B and C. If let the program finish the whole search, it may takes several days or even several weeks. Furthermore, the proposed GA can be easily programmed to be run in parallel, but it is much more complicated for algorithm in [18]. The reason that the running time to design filter C by a single computer using the proposed technique shows longer is because the proposed algorithm is developed in Matlab, which is an interpreted language and much slower than the complied language used in [18].

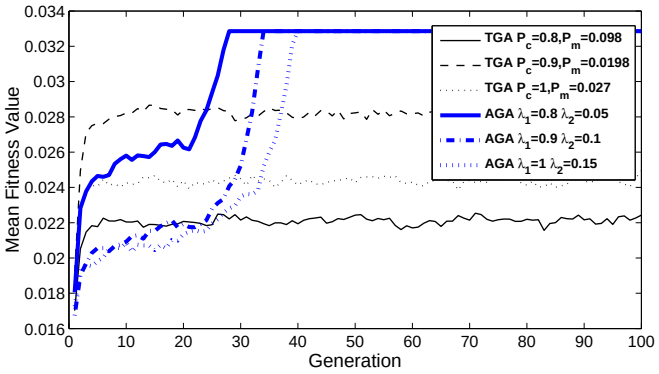

Fig. 3. The mean fitness values changes with the number of generation, AGA is adpative GA, TGA is traditional GA

C. Determining the Parameters λ_1,λ_2 and λ_0

For traditional GA, it is difficult to exactly determine an optimal P_m and P_c, because there are many random factors impacting on the search process. One of the goals of adaptive mutation and crossover is to ease the users' burden to specify P_m and P_c, since the performance of adaptive GA is not significantly influenced by the weighting parameters λ_1 and λ_2. The reason to introduce a constant λ_0 is because without it, the crossover and mutation rates for the best solution are 0. In such a case, the genes of the best solution in a population do not contribute to the next generation through crossover and mutation. By setting λ_0 to be a very small value (less than 0.01), on one hand the best solution is persevered and on the other hand the good "genes" have chances to be inherited by its children.

The general rule to choose λ_1 and λ_2 is that they are selected to be values relatively larger than the crossover and mutation rates in the traditional GA, respectively. This is to ensure that the probabilities of the individuals, whose fitness values are less than the average fitness value, to breed new individuals are reduced. The classical crossover and mutation rates for traditional GA are in the range from 0.8 to 0.95 and from 0.005 to 0.05 [19], respectively. In the proposed GA, λ_1 and λ_2 with values not smaller than 0.8 and 0.05, respectively, are tested. Fig. 3 shows the resultant mean fitness value vs. the generation for λ_1 and λ_2 to be (0.8, 0.05), (0.9, 0.1) and (1, 0.15) respectively, for the design of filter B with G_m for $m = 10$. For comparison, plots for the traditional GA with P_c and P_m to be (0.8, 0.0098), (0.9, 0.0198) and (1, 0.027), respectively, are also shown in Fig. 3. It shows that for the three cases in adaptive GA, the convergence speeds are slightly different, but they all converge to the same mean fitness value. However, the traditional GA tends to converge to a local optimal point.

In the design examples presented in this paper, λ_1, λ_2 and λ_0 are set to 0.9, 0.1 and 0.005, respectively.

D. Superiority and Limitation

The simulation results in Section III-A and the computational complexity analysis in Section III-B showed that the proposed algorithm designs filters with reduced hardware cost and reduced design time in most cases. However, when the filter order is as large as 300, such as the example filter C, the proposed GA cannot beat the design obtained in [18].

This is because the computational complexity of the algorithm is kept to be linearly proportional to the filter order as shown in (11), where the parameters M, J, P and G are all set to be constant independent of N. When N increases, the search space for each subproblem thus increases significantly. For example, when the filter order is 120, the size of the search space is around 3^{60}; when the filter order is increased to 300, the size of the search space is around 3^{150}. Thus the GA cannot obtained optimum solution when the population size (P) and the generation of GA (G) do not increase accordingly. In other words, we sacrifice the optimality to maintain the problem to be tractable, and the algorithm provides a promising approach to design long filters in tractable time.

In the end, a general speculation to the efficiency of the proposed GA is that the GA is more efficient in the design of filters with less stringent specifications. This is for the reason that the efficiency of the proposed GA depends on the size of search space and the amount of feasible solutions in the space. The larger search space and less feasible solutions, the lower efficiency of the algorithm. When the filters are of more stringent specifications, the search space does not change according to our algorithm, but the number of feasible solutions is reduced, such that the efficiency is also reduced. So this is opposite to MILP which in general is more efficient when the specifications are more stringent.

IV. Design Cascaded FIR Filters using the Proposed GA

FIR filters in cascade structure are well-known for their low computational complexity, compared with the single-stage design [28]. Many researches [29]–[32] have attempted to design multiplierless FIR filters in cascade form. However, existing algorithms either can not design long filter [32] or fail to beat the state-of-the-art single-stage design techniques [17], [18].

While GAs have been successfully developed for the design of single-stage FIR filters [20]–[22], they do not show significant advantages in the design of subfilters in cascade form [29]. One of the difficulties in the design of cascade form FIR filter with discrete coefficients is the determination of the subfilter orders. GAs cannot provide an efficient technique to evolve automatically to an order combination of subfilters that can lead to a low hardware cost. In addition, in earlier algorithms for the design of cascade form multiplierless FIR filters [29]–[31], in order to obtain feasible cascade solutions, the EWL of each subfilter is not properly controlled. In these algorithms, although the EWL of the coefficients of each subfilter is shorter or equal to the minimum that can be achieved by the single-stage design, due to the cascade, the width of the SAs of the second subfilter significantly increases. Therefore, the cascade form filters designed using such approaches often have higher hardware cost than that of the best single-stage design.

In this section, we propose a GA for the design of cascade form filters. The GA search is performed on single-stage

filters, and an integer coefficient polynomial factorization technique which results in two discrete coefficient polynomials is proposed. In such a manner, we circumvent the above two difficulties in the design of cascade form filters, i.e. the GA need not to determine the orders of subfilters, and the EWLs of the subfilters are controlled by the single-stage design, so that the EWLs of the subfilters are kept low.

A. Integer Coefficient Polynomial Factorization

For a given integer coefficient $(N-1)$-th order polynomial, denoted as $H_d(z) = \sum_{n=0}^{N-1} h_d(n) z^{-n}$, where $h_d(n)$ are integers for n =0,1,2...$N-1$, there are $N-1$ roots (including real and complex conjugated roots). Denoting the j-th real root as R_j and the k-th complex root as $r_k e^{-i\theta_k}$, where r_k and θ_k are the magnitude and phase of the corresponding complex, respectively, the polynomial $H_d(z)$ can be factorized as:

$$H_d(z) = \beta \prod_{k=1}^{K} (z^{-1} + r_k e^{-i\theta_k}) \prod_{j=1}^{J} (z^{-1} + R_j) \qquad (13)$$

where $K + J = N - 1$, and β is a scaling constant.

If every pair of the factors that are complex conjugated roots are multiplied, $H_d(z)$ can be rewritten as:

$$H_d(z) = \beta \prod_{k=1}^{K/2} (z^{-2} + 2r_k \cos(\theta_k) z^{-1} + {r_k}^2) \prod_{j=1}^{J} (z^{-1} + R_j) \qquad (14)$$

which consists of J first-order factors and $K/2$ second-order factors, all containing real coefficients. Depending on if the coefficient values of a factor are all integers or not, these factors are classified into integer factors and non-integer factors. If we can cast the factors into several groups, and the product of each group is an integer coefficient polynomial, respectively, the original polynomial is successfully factorized. In this paper, we consider only the cascade of two subfilters, i.e. the polynomial is factorized to two integer coefficient polynomials. Therefore, the factorization problem is transferred to how these factors can be allocated to two "baskets", each corresponding to a sub-polynomial, such that the product of the factors in each basket, scaled by a proper constant, is an integer coefficient polynomial. Denoting the two baskets as B1 and B2 and their scalers as β_1 and β_2, respectively, $\beta_1\beta_2$ has to be equal to β.

When the polynomial factors are allocated to a basket, the resultant basket may be an integer basket, which contains only integer factors, or a non-integer basket, which contains only non-integer factors, or a mixed basket, which contains both integer and non-integer factors. Obviously, the product of the factors in an integer basket is an integer polynomial. On the other hand, the product of the factors in a non-integer basket and that in a mixed basket with some proper scalars, also have potentials to be an integer coefficient polynomial. If all the combinations of the allocation of factors into the two baskets B1 and B2 are tested, the problem is in combination complexity and therefore is intractable. For this reason, we restrict the factor allocation such that B1 can be allocated with integer factors only, and B2 can be allocated with either integer or non-integer factors, i.e, B1 is an integer basket taking scaling factor $\beta_1 = 1$ and B2 may be either a non-integer basket or a mixed basket taking a scaling factor $\beta_2 = \beta$.

Since the number of integer factors in the polynomial factorization is limited, the problem is significantly simplified. First, all the integer factors obtained in the factorization of the original integer coefficient polynomial are listed. Second, all the combinations of the integer factors which can be allocated to B1 are listed. For example, if there are 2 integer factors, B1 thus can be allocated with either of them or both of them, and in total, there are 3 combinations. For each combination in B1, the corresponding rest factors are allocated to B2. Finally, the product of the factors in B2 is checked; if the resultant polynomial contains only integer coefficients, the factorization of the integer coefficient polynomial to two integer coefficient polynomials is successful and the result is recorded.

However, in the factorization of (14), all factors are in the normalized form that the coefficient of the highest order term is unit. This excludes the factors such as $2z^{-1}+5$, $5z^{-1}+1$ or $2z^{-2}+z^{-1}+2$, from the integer factors. To secure such integer factors, every non-integer factor is tested to check if the factor can be converted to an integer factor by a converting constant.

Theoretically, the non-integer factors with rational number coefficients can be converted to integer factors if the coefficients are multiplied by a suitable integer, for example, a non-integer factor $z^{-1} + 0.1189$ can be converted to an integer factor if it is multiplied by a converting constant, 10000 in this example. However, when the converting constant is too large, although the resultant factor is an integer factor, the wordlength of the coefficient is too long to be implemented by using a limited number of adders. Therefore, we bound the converting constant, denoted as C, to $C < \max(h_d(n))/2$. If a non-integer factor is converted to an integer factor by converting constant C, the overall scaling factor β is adjusted to β/C.

With all the above considerations, for a given $(N-1)$-th order polynomial $H_d(z)$ with integer coefficients $h_d(n)$ for n =1,2,... $N-1$, the details of its factorization are summarized as follows:

1) Factorizing $H_d(z)$ into the form in (14) by finding all the roots of the polynomial. Integer factors and non-integer factors are classified.
2) Converting the non-integer factors into integer factors, if possible. If there are Q convertable non-integer factors, and the corresponding converting factors are C_i, for i =1,2 ...Q, adjusting β to $\beta / \prod_{i=1}^{Q} C_i$.
3) If even after conversion, there is no integer factors, the polynomial of the given coefficient set cannot be factorized into two integer sub-polynomials; otherwise, all the combinations of the integer factors are listed.
4) For each combination of the integer factors, checking whether the product of the rest factors scaled by $\beta_2 = \beta$ is a polynomial with integer coefficients. If yes, the factorization is successful.

TABLE V
AN EXAMPLE TO FACTORIZE A DISCRETE COEFFICIENT SET

B1	B2	Success
$2z^{-2}+z^{-1}+2$	$10(z^{-2}+0.0699z^{-1}+1)(z^{-2}+1.4301z^{-1}+1)(2z^{-2}+3z^{-1}+2)$	Y
$2z^{-2}+3z^{-1}+2$	$10(z^{-2}+0.0699z^{-1}+1)(z^{-2}+1.4301z^{-1}+1)(2z^{-2}+z^{-1}+2)$	Y
$(2z^{-2}+z^{-1}+2)(2z^{-2}+3z^{-1}+2)$	$10(z^{-2}+0.0699z^{-1}+1)(z^{-2}+1.4301z^{-1}+1))$	Y

B. An Example to Illustrate the Factorization Technique

A polynomial p with an integer coefficient set { 40,140,314,473,551,473,314,140,40} is given to illustrate the factorization technique in details. The roots of the polynomial p are computed by using Matlab function "roots", and they are $-0.25+0.9682i$, $-0.25-0.9682$, $-0.0350+0.9994i$, $-0.0350-0.9994i$, $-0.7150+0.6991i$, $-0.7150-0.6991i$, $-0.75+0.6614i$ and $-0.75-0.6614i$, respectively, with a scaling constant $\beta = 40$. By pairing the conjugated roots, we can get all the factors, they are $z^{-2}+0.5z^{-1}+1$, $z^{-2}+0.0699z^{-1}+1$, $z^{-2}+1.4301z^{-1}+1$ and $z^{-2}+1.5z^{-1}+1$. Originally, there are no integer factors. By conversion, the non-integer factors $z^{-2}+0.5z^{-1}+1$ and $z^{-2}+1.5z^{-1}+1$ are converted to integer factors $2z^{-2}+z^{-1}+2$ and $2z^{-2}+3z^{-1}+2$ when they are multiplied by an integer 2 for 2 is less than $473/2$. Thus the scaling constant β becomes 10. After conversion, there are two integer factors and two non-integer factors. According to the proposed factorization technique, B1 contains only integer factors. Therefore, there are in total 3 possible combinations of integer factors that can be allocated to B1 as listed in Table V. The rest factors, together with β, of each combination in B1 are allocated to B2. The coefficients of the products of the factors in B2 are checked; the resultant coefficients of B2 in this example are {20,60,107,123,107,60,20}, { 20,40,77,81,77,40,20} and {10,15,21,15,10} for the first, second and third factorization, respectively. Therefore, all of these factorizations are successful.

C. Incorporating the Factorization Technique into the Proposed GA

When the factorization technique is incorporated into the proposed GA, the GA search is conducted on single-stage filters. The 3 genetic operations including selection, crossover and mutation are applied as usual, but the fitness is computed in a different way. During the GA search, once a chromosome is obtained, an integer coefficient set of a single-stage filter is obtained. This discrete coefficient set is evaluated to check if the filter specification is satisfied, and the adder cost, f_g, is computed. The factorization technique proposed in Section IV-A is performed only when the filter specification is met. If the polynomial of the single-stage filter is successfully factorized into two integer polynomials, the adder cost to realize the two factorized integer coefficient sets, denoted as $f_{g_{cas}}$, is computed. If $f_{g_{cas}} \leq f_g$, the fitness value of the chromosome is set to $1/f_{g_{cas}}$; otherwise, to $1/f_g$. Note that 1) for an integer coefficient set (Chromosome), the factorization may not be unique, and thus $f_{g_{cas}}$ is set to the smallest adder cost and the corresponding factorization is recorded; 2) the factorization is not necessary leading to a smaller objective value (i.e., adder cost) than the single-stage realization, and the factorization is not recorded in this case.

A simple example to illustrate the fitness evaluation of the chromosomes is given as follows. If a chromosome is decoded to an integer coefficient set {5,29,52,62,52,29,5}, the adder cost of this integer set (including both MBAs and SAs) is $f_g = 10$. The proposed factorization technique is performed and the polynomial can be factorized into two subpolynomials with coefficient sets {1,4,1} and {5,9,11,9,5}, respectively. The adder cost to realize these two integer coefficient sets is $f_{g_{cas}} = 9$. Since $f_{g_{cas}} \leq f_g$, the fitness value of this chromosome is $1/f_{g_{cas}}$, i.e., $1/9$. The factorization result is recorded.

V. NUMERICAL EXAMPLES OF CASCADE FORM DESIGN

In this section, two sets of design examples are given to illustrate the superiority of the proposed algorithm.

In Example A, the long filters B, L1 and C are designed using the proposed algorithm in cascade form. Since there is no existing algorithm directly designing long filter in cascade form which aims to minimize the adder cost, the results are first compared with the best results of single-stage designs in [18]. We further re-pack the algorithms in [17], [32] in a manner such that they can be used to design long filters in the cascade form, and the results obtained are also compared.

In Example B, two moderately long filters A1 and S2 (with length around 60) taken from [32] are designed using the proposed GA in cascade form. The results are compared with the cascade design in [32]. Note that if enough time is given, the algorithm in [32] has high probability to find the cascade design with minimum number of adders. However, even for the moderately long filters, it may takes several weeks to finish the search; therefore in [32] the runtime for each design is limited to 24h.

A. Example A

The design results of B, L1 and C are listed in Table VI. The single-stage design obtained by [18] and by the proposed GA are also listed. The reason that we compare the results with the single-stage design in [18] is that no existing multiplierless technique can directly design long FIR filters in cascade form with lower hardware cost than the state-of-the-art single-stage design technique.

However, with existing cascade filter design techniques [29]–[32] and discrete coefficient filter design technique for moderately long filters [17], alternative approaches indirectly designing the long filters are possible. Here, filter B is designed using such a possible way to compare with our results.

The basic idea of the indirect design is to first separate the long filter into a cascade of several short filters with continuous coefficients, and each short filter is then designed by [17] or [32]. Two subfilters are considered in this example, denoted as F1 and F2, respectively. However, a difficulty is to set the specifications for each subfilter, such as filter orders, ripples and EWLs. In [30], two approaches are proposed.

The first method is named as direct approximation method, in which F1 is designed to continuous coefficient values directly using the frequency response specifications of the overall filter and by fixing F1, F2 is then optimized in discrete values. However, in the design of filter B, F1 designed in such a way cannot lead to an overall filter meeting the ripple requirement even if F2 is optimized with continuous coefficients.

The other way is to obtain subfilters using the interleaved zero approach. After obtaining the continuous coefficient values of F1 by using this method, F1 is fixed and F2 is optimized using algorithms [17] or [32] to obtain discrete coefficient values such that the overall filter meets the specification of filter B. Then, F2 is fixed and F1 is optimized using [17] or [32] to meet the overall specification. In the design of Filter B, the subfilter orders for both F1 and F2 are 52, according to the interleaved zero approach.

Using [17], the resultant hardware cost for F1 and F2 are 60 and 55 adders, with EWLs 8 and 5, respectively. Thus, the EWL of the overall filter is as larger as 15 due to the convolution of the two subfilters. Note that filter B designed using the proposed algorithm requires only 94 adders with overall EWL 8. In terms of the hardware cost, the filter obtained using the proposed GA is much lower, because of the small number of adders and small EWL. Using [32], each subfilter is further a cascade two short subfilters. However, in this case, in the step fixing F1 and optimizing F2, even if the EWL of F2 is increased to 6, no solution can be found. It is possible to further increase the EWL of F2 until a solution is found, but this, on one hand, increases the design time significantly, and, on the other hand, will increase the hardware cost significantly due to the large overall EWL of the filter.

This design examples show that such indirect optimization approaches either generate designs with much higher hardware cost than the proposed GA or cannot produce feasible cascade designs for a given search space. In addition, the runtimes of these algorithms are also much longer because each subproblem (to optimize the subfilter) may take several hours for a filter with orders around 50.

The reason that such indirectly techniques cannot generate better results is due to the difficulty in how to set the orders of the subfilters, and how to allocate zeros to each subfilter. Using the interleaved zero approach, for a given filter orders, the way to allocate zeros to the two subfilters are fixed. Certainly, there are other ways to allocate zeros that may result in feasible and better solutions, but obviously finding a good allocation is not straightforward, and significant additional research may be required.

Overall speaking, the proposed technique is the first one efficiently achieving lower hardware cost for the design of long filters in cascade form. The coefficient values of the cascade designs of B, L1 and C are listed in Table VII, VIII and IX, respectively.

TABLE X

THE FILTER SPECIFICATIONS FOR A1 AND S2

Filters	Filter order	ω_p	ω_s	δ_p	δ_s
A1 (low pass)	58	0.125π	0.225π	0.01	0.001
S2 (low pass)	59	0.042π	0.14π	0.012	0.001

TABLE XIV

DESIGN RESULTS OF FILTER B WITH DIFFERENT FILTER ORDERS. L_i MBA_i AND SA_i ARE THE LENGTH, NUMBER OF MULTIPLIER BLOCK ADDERS AND NUMBER OF STRUCTURAL ADDERS OF SUBFILTER i, RESPECTIVELY.

Order	EWL	L_1	L_2	MBA_1	MBA_2	SA_1	$SA2_2$	Total
105	8	2	105	0	10	1	90	101
106	8	3	105	0	10	2	88	100
107	8	2	107	0	8	1	92	101
108	8	No cascade design is found						
109	8	2	109	0	10	1	92	103

B. Example B

In this example, the proposed GA is used for the design of two moderately long filters A1 and S2 in cascade form and the results are compared with that obtained by MILP based algorithm [32]. The specifications of A1 and S2 are listed in Table X. Table XI shows that for the filters with moderately long length, which can be handled by deterministic algorithm such as [32], the propose heuristic GA use much shorter design time than that of the algorithm in [32], while the adder cost of the final designs is not worse than that in [32]. It has to be noted that, for filter S2, when the EWL and order are kept to be 10 and 59 as in [32], the proposed GA can not find the feasible solution. Therefore, as described in Section II-G, the filter order is increased by 1 and the optimization is re-done. The design times of the proposed GA for filter A1 and S2 are 9m48s and 21m18s using multiple computers and 1h36m and 3h11m using one computer. The design time are significantly less than that of the algorithm in [32], where both designs use more than 24 hours. The coefficient values of the cascade design of A1 and S2 are listed in Table XII, and XIII, respectively.

C. Robustness for the Design of Cascaded FIR Filters

Since the proposed GA for the design of cascaded FIR filter is based on the factorization of a given integer coefficient polynomial and the factorization is not guaranteed to be successful, it is possible that no cascade design is produced by the proposed GA. To test the robustness of the proposed GA for the design of cascaded FIR filters, filter B is designed with filter orders from 105 to 109, and the design results are listed in Table XIV. Table XIV shows that for each filter order except 108, cascade designs can be found.

An additional advantage of the proposed GA for the cascade design is its compatibility with the single-stage design, i.e., even if no cascade design is found, in the end of the optimization the proposed GA takes the best single-stage design as the final design, as discussed in Section IV-C.

TABLE VI

DESIGN RESULTS OF FILTERS B, L1 AND C. L_i MBA_i AND SA_i ARE THE LENGTH, NUMBER OF MULTIPLIER BLOCK ADDERS AND NUMBER OF STRUCTURAL ADDERS OF SUBFILTER i, RESPECTIVELY.

		Cascade Design by Proposed GA								Single Stage design by Proposed GA				Algorithm in [18]			
Filters	EWL	L_1	L_2	MBA_1	MBA_2	SA_1	$SA2_2$	Total	Runtime	MBA	SA	Total	Runtime	MBA	SA	Total	Runtime
B	8	2	104	0	8	1	85	94	22m19s	10	98	109	16m24s	11	100	111	>24h
L1	14	7	115	2	30	6	114	152	34m21s	43	120	163	26m41s	47	120	167	56m
C	10	3	323	0	20	2	278	298	4h3m	31	306	337	3h49m	22	306	328	>24h

TABLE VII

THE COEFFICIENT VALUES OF FILTER B IN CASCADE FORM

Filter B $h_1(n) = h_1(1-n)$ for $n = 0$, $h_2(n) = h_2(103-n)$ for $0 \le n \le 51$
$h(n) = h(104-n)$ for $0 \le n \le 52$, passband gain: 907.66 EWL: 8
normalized passband ripple: 0.00995; normalized stopband ripple: 0.00995

$h_1(0) =$	1	$h_2(7) =$	0	$h_2(15) =$	2	$h_2(23) =$	3	$h_2(31) =$	5	$h_2(39) =$	8	$h_2(47) =$	0
$h_2(0) =$	−2	$h_2(8) =$	−2	$h_2(16) =$	−3	$h_2(24) =$	1	$h_2(32) =$	6	$h_2(40) =$	12	$h_2(48) =$	27
$h_2(1) =$	1	$h_2(9) =$	0	$h_2(17) =$	−1	$h_2(25) =$	−2	$h_2(33) =$	1	$h_2(41) =$	12	$h_2(49) =$	57
$h_2(2) =$	−1	$h_2(10) =$	−2	$h_2(18) =$	−4	$h_2(26) =$	−4	$h_2(34) =$	−3	$h_2(42) =$	4	$h_2(50) =$	85
$h_2(3) =$	2	$h_2(11) =$	2	$h_2(19) =$	0	$h_2(27) =$	−5	$h_2(35) =$	−8	$h_2(43) =$	−6	$h_2(51) =$	100
$h_2(4) =$	0	$h_2(12) =$	0	$h_2(20) =$	0	$h_2(28) =$	−2	$h_2(36) =$	−8	$h_2(44) =$	−18		
$h_2(5) =$	2	$h_2(13) =$	3	$h_2(21) =$	3	$h_2(29) =$	0	$h_2(37) =$	−6	$h_2(45) =$	−22		
$h_2(6) =$	−1	$h_2(14) =$	0	$h_2(22) =$	3	$h_2(30) =$	5	$h_2(38) =$	1	$h_2(46) =$	−18		

TABLE VIII

THE COEFFICIENT VALUES OF FILTER L1 IN CASCADE FORM

Filter L1 $h_1(n) = h_1(6-n)$ for $0 \le n \le 3$, $h_2(n) = h_2(114-n)$ for $0 \le n \le 57$
$h(n) = h(120-n)$ for $0 \le n \le 60$, passband gain: 69265.174 EWL: 14
normalized passband ripple: 0.005572; normalized stopband ripple: 0.00009775

$h_1(0) =$	1	$h_2(5) =$	13	$h_2(14) =$	1	$h_2(23) =$	35	$h_2(32) =$	−18	$h_2(41) =$	106	$h_2(50) =$	−161
$h_1(1) =$	−3	$h_2(6) =$	4	$h_2(15) =$	20	$h_2(24) =$	−4	$h_2(33) =$	65	$h_2(42) =$	−59	$h_2(51) =$	259
$h_1(2) =$	5	$h_2(7) =$	9	$h_2(16) =$	6	$h_2(25) =$	33	$h_2(34) =$	−6	$h_2(43) =$	81	$h_2(52) =$	−147
$h_1(3) =$	−6	$h_2(8) =$	9	$h_2(17) =$	14	$h_2(26) =$	18	$h_2(35) =$	17	$h_2(44) =$	38	$h_2(53) =$	35
$h_2(0) =$	6	$h_2(9) =$	5	$h_2(18) =$	23	$h_2(27) =$	−5	$h_2(36) =$	66	$h_2(45) =$	−61	$h_2(54) =$	283
$h_2(1) =$	3	$h_2(10) =$	17	$h_2(19) =$	−3	$h_2(28) =$	50	$h_2(37) =$	−36	$h_2(46) =$	158	$h_2(55) =$	−486
$h_2(2) =$	4	$h_2(11) =$	5	$h_2(20) =$	27	$h_2(29) =$	−6	$h_2(38) =$	77	$h_2(47) =$	−90	$h_2(56) =$	745
$h_2(3) =$	3	$h_2(12) =$	15	$h_2(21) =$	10	$h_2(30) =$	31	$h_2(39) =$	5	$h_2(48) =$	72	$h_2(57) =$	−764
$h_2(4) =$	4	$h_2(13) =$	16	$h_2(22) =$	7	$h_2(31) =$	36	$h_2(40) =$	−9	$h_2(49) =$	109		

TABLE IX

THE COEFFICIENT VALUES OF FILTER C IN CASCADE FORM

Filter C: $h_1(n) = h_1(2-n)$ for $0 \le n \le 2$, $h_2(n) = h_2(322-n)$ for $0 \le n \le 161$
$h(n) = h(324-n)$ for $0 \le n \le 162$, passband gain: 5395.25, EWL: 10
normalized passband ripple: 0.00489; normalized stopband ripple:0.00489

$h_1(0) =$	1	$h_2(22) =$	1	$h_2(46) =$	0	$h_2(70) =$	3	$h_2(94) =$	6	$h_2(118) =$	−5	$h_2(142) =$	34
$h_1(1) =$	1	$h_2(23) =$	1	$h_2(47) =$	0	$h_2(71) =$	0	$h_2(95) =$	8	$h_2(119) =$	−12	$h_2(143) =$	31
$h_2(0) =$	−4	$h_2(24) =$	1	$h_2(48) =$	−3	$h_2(72) =$	−5	$h_2(96) =$	0	$h_2(120) =$	−19	$h_2(144) =$	23
$h_2(1) =$	0	$h_2(25) =$	−1	$h_2(49) =$	4	$h_2(73) =$	−3	$h_2(97) =$	12	$h_2(121) =$	−6	$h_2(145) =$	16
$h_2(2) =$	2	$h_2(26) =$	0	$h_2(50) =$	3	$h_2(74) =$	−2	$h_2(98) =$	10	$h_2(122) =$	−7	$h_2(146) =$	0
$h_2(3) =$	−2	$h_2(27) =$	0	$h_2(51) =$	−1	$h_2(75) =$	−8	$h_2(99) =$	−3	$h_2(123) =$	−7	$h_2(147) =$	−20
$h_2(4) =$	3	$h_2(28) =$	−3	$h_2(52) =$	6	$h_2(76) =$	−2	$h_2(100) =$	7	$h_2(124) =$	10	$h_2(148) =$	−33
$h_2(5) =$	1	$h_2(29) =$	−1	$h_2(53) =$	2	$h_2(77) =$	1	$h_2(101) =$	3	$h_2(125) =$	9	$h_2(149) =$	−48
$h_2(6) =$	−1	$h_2(30) =$	−1	$h_2(54) =$	−1	$h_2(78) =$	−5	$h_2(102) =$	−14	$h_2(126) =$	11	$h_2(150) =$	−55
$h_2(7) =$	2	$h_2(31) =$	−2	$h_2(55) =$	3	$h_2(79) =$	4	$h_2(103) =$	−2	$h_2(127) =$	22	$h_2(151) =$	−50
$h_2(8) =$	2	$h_2(32) =$	1	$h_2(56) =$	0	$h_2(80) =$	7	$h_2(104) =$	−4	$h_2(128) =$	14	$h_2(152) =$	−41
$h_2(9) =$	−1	$h_2(33) =$	−1	$h_2(57) =$	−5	$h_2(81) =$	0	$h_2(105) =$	−18	$h_2(129) =$	9	$h_2(153) =$	−17
$h_2(10) =$	0	$h_2(34) =$	0	$h_2(58) =$	−1	$h_2(82) =$	7	$h_2(106) =$	−3	$h_2(130) =$	12	$h_2(154) =$	18
$h_2(11) =$	1	$h_2(35) =$	4	$h_2(59) =$	−1	$h_2(83) =$	9	$h_2(107) =$	0	$h_2(131) =$	−4	$h_2(155) =$	54
$h_2(12) =$	−2	$h_2(36) =$	0	$h_2(60) =$	−7	$h_2(84) =$	−3	$h_2(108) =$	−9	$h_2(132) =$	−12	$h_2(156) =$	100
$h_2(13) =$	−1	$h_2(37) =$	2	$h_2(61) =$	0	$h_2(85) =$	4	$h_2(109) =$	8	$h_2(133) =$	−11	$h_2(157) =$	145
$h_2(14) =$	0	$h_2(38) =$	3	$h_2(62) =$	0	$h_2(86) =$	2	$h_2(110) =$	10	$h_2(134) =$	−23	$h_2(158) =$	182
$h_2(15) =$	−2	$h_2(39) =$	−1	$h_2(63) =$	−3	$h_2(87) =$	−8	$h_2(111) =$	2	$h_2(135) =$	−25	$h_2(159) =$	216
$h_2(16) =$	0	$h_2(40) =$	1	$h_2(64) =$	3	$h_2(88) =$	−4	$h_2(112) =$	16	$h_2(136) =$	−15	$h_2(160) =$	237
$h_2(17) =$	0	$h_2(41) =$	0	$h_2(65) =$	5	$h_2(89) =$	−2	$h_2(113) =$	12	$h_2(137) =$	−15	$h_2(161) =$	241
$h_2(18) =$	−1	$h_2(42) =$	−2	$h_2(66) =$	0	$h_2(90) =$	−12	$h_2(114) =$	1	$h_2(138) =$	−7		
$h_2(19) =$	1	$h_2(43) =$	−2	$h_2(67) =$	5	$h_2(91) =$	−2	$h_2(115) =$	8	$h_2(139) =$	11		
$h_2(20) =$	1	$h_2(44) =$	−1	$h_2(68) =$	6	$h_2(92) =$	0	$h_2(116) =$	1	$h_2(140) =$	16		
$h_2(21) =$	1	$h_2(45) =$	−4	$h_2(69) =$	−2	$h_2(93) =$	−6	$h_2(117) =$	−13	$h_2(141) =$	24		

TABLE XI

DESIGN RESULTS OF FILTER A1 AND S2. L_i MBA_i AND SA_i ARE THE LENGTH, NUMBER OF MULTIPLIER BLOCK ADDERS AND NUMBER OF STRUCTURAL ADDERS OF SUBFILTERS i, RESPECTIVELY.

		The propose GA								Algorithm in [32]							
Filters	EWL	L_1	L_2	MBA_1	MBA_2	SA_1	$SA2_2$	Total	Runtime	L_1	L_2	MBA_1	MBA_2	SA_1	$SA2_2$	Total	Runtime
A1	10	5	55	0	8	4	48	60	9m48s	2	58	0	9	2	52	63	24h
S2	10	3	59	0	13	2	54	69	21m09s	2	59	0	14	1	54	69	24h

TABLE XII

THE COEFFICIENT VALUES OF FILTER A1 IN CASCADE FORM

Filter A1: $h_1(n) = h_1(4-n)$ for $0 \le n \le 2$, $h_2(n) = h_2(54-n)$ for $0 \le n \le 27$ $h(n) = h(58-n)$ for $0 \le n \le 29$, passband gain: 4894.15, EWL: 10 normalized passband ripple: 0.009908; normalized stop ripple: 0.0009907																	
$h_1(0)$ =	1	$h_2(2)$ =	2	$h_2(7)$ =	−9	$h_2(12)$ =	18	$h_2(17)$ =	−27	$h_2(22)$ =	27	$h_2(27)$ =	97				
$h_1(1)$ =	2	$h_2(3)$ =	−2	$h_2(8)$ =	3	$h_2(13)$ =	0	$h_2(18)$ =	−9	$h_2(23)$ =	26						
$h_1(2)$ =	2	$h_2(4)$ =	2	$h_2(9)$ =	−8	$h_2(14)$ =	18	$h_2(19)$ =	−36	$h_2(24)$ =	80						
$h_2(0)$ =	2	$h_2(5)$ =	−6	$h_2(10)$ =	9	$h_2(15)$ =	−8	$h_2(20)$ =	−6	$h_2(25)$ =	74						
$h_2(1)$ =	0	$h_2(6)$ =	0	$h_2(11)$ =	−1	$h_2(16)$ =	6	$h_2(21)$ =	−20	$h_2(26)$ =	120						

TABLE XIII

THE COEFFICIENT VALUES OF FILTER S2 IN CASCADE FORM

Filter S2: $h_1(n) = h_1(2-n)$ for $0 \le n \le 1$, $h_2(n) = h_2(58-n)$ for $0 \le n \le 29$ $h(n) = h(60-n)$ for $0 \le n \le 30$, passband gain: 7263.98, EWL: 10 normalized passband ripple:0.011922;normalized passband ripple:0.0009935																	
$h_1(0)$ =	1	$h_2(3)$ =	2	$h_2(8)$ =	−11	$h_2(13)$ =	−22	$h_2(18)$ =	14	$h_2(23)$ =	116	$h_2(28)$ =	202				
$h_1(1)$ =	1	$h_2(4)$ =	0	$h_2(9)$ =	−11	$h_2(14)$ =	−24	$h_2(19)$ =	31	$h_2(24)$ =	143	$h_2(29)$ =	202				
$h_2(0)$ =	2	$h_2(5)$ =	−3	$h_2(10)$ =	−16	$h_2(15)$ =	−16	$h_2(20)$ =	47	$h_2(25)$ =	163						
$h_2(1)$ =	1	$h_2(6)$ =	−2	$h_2(11)$ =	−22	$h_2(16)$ =	−10	$h_2(21)$ =	73	$h_2(26)$ =	177						
$h_2(2)$ =	0	$h_2(7)$ =	−6	$h_2(12)$ =	−20	$h_2(17)$ =	−3	$h_2(22)$ =	96	$h_2(27)$ =	195						

VI. CONCLUSION

In this paper, a novel GA is proposed for the design of low hardware cost multiplierless FIR filters both in single-stage and cascade forms. In order to make the GA capable of the design of high order and wide coefficient wordlength filters, the discrete search space is partitioned into multiple spaces according to the passband gains. Moreover, the search efficiency of the GA is improved by adaptively adjusting the crossover and mutation rates. In addition, instead of minimizing the filter ripple as in conventional GAs, the proposed GA uses the adder cost of the filter as the objective function, and penalties are applied when the ripple requirement is not met, such that the hardware cost and filter ripples are balanced in the fitness values. The cascade design is incorporated into the GA search for the single-stage designs. Every single-stage design obtained during the GA operation, if meeting the filter specification, is applied with an integer coefficient polynomial factorization. Successful factorizations generate the cascade designs if the hardware cost are lower. Design examples show that the proposed GA uses much less computational time and the hardware cost of the resulting designs, for both single-stage and cascade designs, is reduced in most cases.

REFERENCES

[1] L. Aksoy, E. Gunes, and P. Flores, "An exact breadth-first search algorithm for the multiple constant multiplications problem," Tallinn, Nov. 2008, pp. 41–46.

[2] M. Faust and C. H. Chang, "Minimal logic depth adder tree optimization for multiple constant multiplication," in *Proc. IEEE Int. Symp. Circuits Syst.*, Paris, May 2010, pp. 457–460.

[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] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *IEE Proc. G*, vol. 138, pp. 401–412, June 1991.

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

[6] R. Pasko, P. Schaumont, V. Derudder, *et al.*, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 56–68, Jan. 1999.

[7] 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. Circuits and Systems*, vol. 15, pp. 151–165, Feb. 1996.

[8] A. G. Dempster, S. S. Demirsoy, and I. Kale, "Designing multiplier blocks with low logic depth," in *Proc. IEEE International Symposium on Circuits and Systems*, Scottsdale, Arizona, May 2002, pp. 773–776.

[9] O. Gustafsson and L. Wanhammar, "A novel approach to multiple constant multiplication using minimum spanning trees," in *Proc. IEEE Midwest Symp. Circuits Syst*, Tulsa, OK, Aug. 2002, pp. 652–655.

[10] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Improved multiple constant multiplication using minimum spanning trees," in *Proc. IEEE Asilomar Conf. Signals Syst. Comp.*, Pacific Grove, CA, Nov. 2004, pp. 63–66.

[11] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in *Proc. IEEE Int. Symp. Circuits Syst.*, New Orleans, LA, May 2006, pp. 1097–1100.

[12] O. Gustafsson and L. Wanhammar, "ILP modeling of the common subexpression sharing problem," in *Proc. IEEE ICECS'02*, Dubrovnic, Croatia, 2002, pp. 1171–1174.

[13] J. Yli-Kaakinen and T. Saramäki, "A systematic algorithm for the design of multiplierless FIR filters," in *Proc. IEEE Int. Symp. Circuits Syst*, Sydney, Australia, 2001, pp. 185–188.

[14] Y. J. Yu and Y. C. Lim, "Design of linear phase FIR filters in subexpression space using mixed integer linear programming," *IEEE Trans. Circuits Syst. I*, vol. 54, pp. 2330–2338, Oct. 2007.

[15] Y. J. Yu, D. Shi, and Y. C. Lim, "Design of extrapolated impulse response FIR filters with residual compensation in subexpression space," *IEEE Trans. Circuits Syst. I*, vol. 56, pp. 2621–2633, Dec. 2009.

[16] Y. J. Yu and Y. C. Lim, "Optimization of linear phase FIR filters in dynamically expanding subexpression space," *Circuits, Systems, and Signal Processing*, vol. 29, pp. 65–80, June 2009.
[17] D. Shi and Y. J. Yu, "Design of linear phase FIR filters with high probability of achieving minimum number of adders," *IEEE Trans. Circuits Syst. I*, vol. 58, pp. 126–136, Jan. 2011.
[18] M. Aktan, A. Yurdakul, and G. DÍźnda, "An algorithm for the design of low-power hardware-efficient FIR filters,," *IEEE Trans. Circuits and Systems*, vol. 55, pp. 1536–1545, July 2008.
[19] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.
[20] P. Gentili, F. Piazza, and A. Uncini, "Efficient genetic algorithm design for power-of-two FIR filters," in *Proc. International Conference on Acoustics, Speech, and Signal Processing*, Detroit, Michigan, USA, May 1995, pp. 9–12.
[21] L. Cen and Y. Lian, "A modified micro-genetic algorithm for the design of multiplierless digital FIR filters," in *Proc. IEEE Region 10 Conference*, Chiang Mai, Thailand, Nov. 2004, pp. 52–55.
[22] L. Cen, "A hybrid genetic algorithm for the design of FIR filters with SPoT coefficients," *Signal Processing*, vol. 87, pp. 528–540, 2007.
[23] Y. C. Lim, Y. Sun, and Y. J. Yu, "Design of discrete-coefficient FIR filters on loosely connected parallel machines," *IEEE Transactions on Signal Processing*, vol. 50, pp. 1409–1416, 2002.
[24] W. B. Ye and Y. J. Yu, "Design of high order and wide coefficient wordlength multiplierless FIR filters with low hardware cost using genetic algorithm," in *Proc. IEEE Int. Symp. Circuits Syst*, Seoul, Korea, May 2012, pp. 29–32.
[25] Y. C. Lim, "Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 1480–1486, Dec. 1990.
[26] Y. C. Lim and S. R. Parker, "Discrete coefficient FIR digital filter design based upon an LMS criteria," *IEEE Trans. Circuits Syst.*, vol. 30, pp. 723–739, Oct. 1983.
[27] D. E. Goldberg, *Genetic algorithm in search and optimization*. Addison-Wesley, 1989.
[28] J. W. Adams and J. A. N. Willson, "A new approach to FIR digital filters with fewer multipliers and reduced sensitivity," *IEEE Trans Circuits Systems*, vol. 30, pp. 277–283, May 1983.
[29] S. U. Ahmad and A. Antoniou, "Cascade-form multiplierless FIR filter design using orthogonal genetic algorithm," in *Proc. IEEE Int. Symp. Signal Process.Inf. Technol.*, Aug 2006, pp. 932–937.
[30] Y. C. Lim and B. Liu, "Design of cascade form FIR filters with discrete valued coefficients," *IEEE Transactions on Signal Processing*, vol. 36, pp. 1735–1939, Nov. 1988.
[31] Y. C. Lim, R. Yang, and B. Liu, "The design of cascaded FIR filters," in *Proc. IEEE Int. Symp. Circuits Syst*, May 1996, pp. 181–184.
[32] D. Shi and Y. J. Yu, "Design of discrete-valued linear phase FIR filters in cascade form," *IEEE Trans. Circuits Syst. I*, vol. 58, pp. 1627–1636, 2011.