國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

應用設計空間探索於有限脈衝響應濾波器之硬體最佳化

Design Space Exploration for Hardware-Efficient FIR Filter Design

研 究 生：楊創任

指導教授：周景揚 教授

中 華 民 國 一 〇 一 年 七 月

應用設計空間探索於有限脈衝響應濾波器之硬體最佳化

學生：楊創任　　　　　　　　　　　　　　　　　指導教授：周景揚 博士

國立交通大學
電子工程學系　電子研究所碩士班

摘　　要

在這篇論文中，我們提出一個演算法，針對線性相位的有限脈衝響應濾波器選擇一組符合規格的濾波器係數，此演算法的主要目的是最小化一個利用多重常數乘法器所實作的有限脈衝響應濾波器中的加法器個數。在傳統的設計中，實作於有限脈衝響應濾波器中的多重常數乘法器只會利用加法和左移(left-shift)這兩種運算，然而，我們的演算法允許使用右移(right-shift)運算來擴展設計空間。我們也發展一個啟發式的分支限界法(branch and bound method)，它可以使我們在擴展的設計空間中有效率的搜尋。實驗數據顯示我們的演算法相較於目前存在最好的方法，在加法器個數上最多可以改善30.6%且平均改善13.8%。

Design Space Exploration for Hardware-Efficient FIR Filter Design

Student：Chuang-Ren Yang　　　　Advisor：Dr. Jing-Yang Jou

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

In this thesis, we propose an algorithm to determine coefficients for a specified linear phase FIR filter design. The target of our algorithm is to minimize the adder cost as the FIR filter is implemented through multiple constant multiplication (MCM). Traditionally, an MCM block in an FIR filter design is implemented using addition and left-shift operations only. Nevertheless, our algorithm allows the use of right-shift operations to further expand the design space. We also develop a heuristic-based approximated branch and bound method to search in broader design space efficiently. Experimental results show that our method can reduce the adder cost by up to 30.6% and 13.8% on average as compared to an existing state-of-the-art technique.

Acknowledgements

I greatly appreciate my advisors, Dr. Jing-Yang Jou and Dr. Juinn-Dar Huang, for their patient guidance, valuable suggestion, and encouragement during these years. I would like to thank Bu-Ching Lin and Yung-Chun Lei for their discussion and help on my research. Specially thank to all member of EDA Lab for their friendship and company. Finally, I would like to express my sincerely acknowledgements to my family and my friends for their patient and support.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 FIR Filter

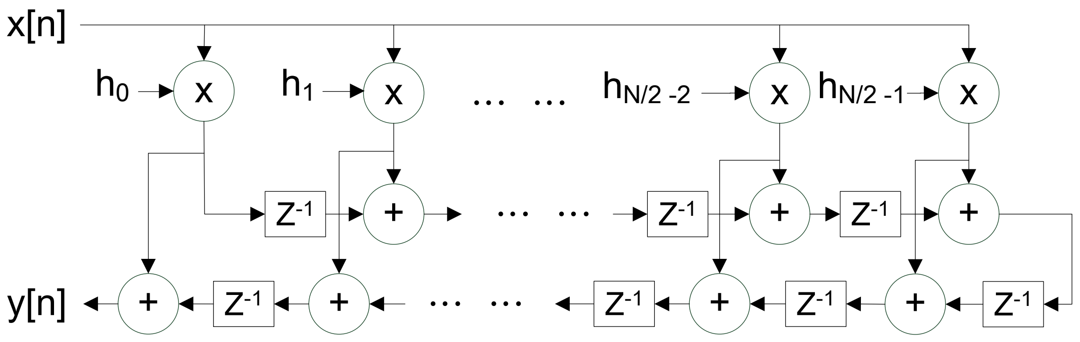
In signal processing system, the finite impulse response (FIR) filter is usually an important component because of its stability and linear phase properties. For an N-tap FIR filter, the input data $x[n]$ with different time scale are multiplied by corresponding constant coefficients h_k and then are summed up to output data $y[n]$. The equation is shown in the following.

$$y[n] = \sum_{k=0}^{N-1} h_k x[n-k] \tag{1.1}$$

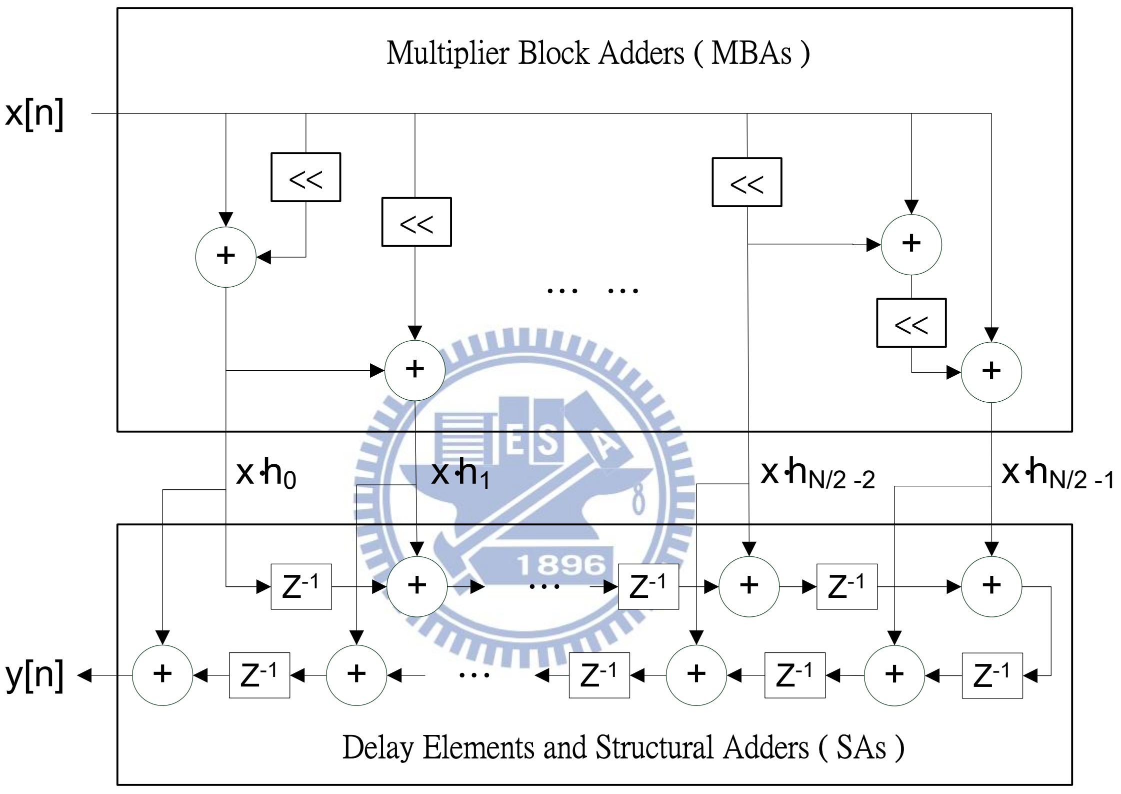
1.2 The Implementation of FIR Filter

Compared with its alternative – the infinite impulse response (IIR) filter, the FIR filter has the advantage of stability. However, the hardware cost of an FIR filter is generally much higher than that of an IIR filter. Therefore, it is an important design issue to minimize the hardware cost when implementing a FIR filter.

As indicated in (1.1), a design of FIR filter is composed of constant multipliers, adders and delay elements; and the constant multipliers occupy the most part of area. A trivial implementation is to adopt general multipliers for carrying out those constant multiplications. However, the general multiplier is an expensive functional unit in terms of hardware implementation cost. Since the coefficients are all constants in filter design, in order to reduce the cost, the constant multipliers are generally implemented by adders and shifters. For example, the constant multiplication $5 * x$ can be computed as $(x << 2) + x$. The constant multiplier can be replaced by a shifter and an adder. The cost of a constant multiplier is roughly proportional to the number of required adders because those constant shifters are accomplished by simply rewiring and thus are actually at no cost. That is, the cost of an FIR filter basically depends on the total number of adders. Fig. 1(a) shows an architecture of a transposed form N-tap FIR filter, which is functionally equivalent to (1.1). It is observed that input data are multiplied by a set of constant coefficients. Moreover, while implementing these constant multipliers all together at the same time, it has a very good chance to share adders among them for cost reduction. A structure which effectively implements a set of constant multipliers is also referred to as a multiple constant multiplication (MCM) block. Fig. 1(b) illustrates an N-tap FIR filter implemented by an MCM block. The adders in an MCM-based filter can be classified into structural adders (SAs) and multiplier block adders (MBAs). MBAs are adders residing in the MCM block, while SAs are those used to sum up the outputs of the MCM block.



(a)



Multiplier Block Adders (MBAs)
x[n]
<<
x·h0
x·h1
x·hN/2 -2
x·hN/2 -1
Z-1
y[n]
Delay Elements and Structural Adders (SAs)

(b)

Fig. 1 (a) An *N*-tap FIR filter architecture. (b) An *N*-tap FIR filter architecture with MCM.

There are algorithms for the MCM problem, which generate the MCM design for the given set of coefficients and minimize the number of adders. The MCM algorithms can simply be divided into two classes: graph-based algorithms [1][2] and common subexpression elimination (CSE) algorithms [3]. Graph-based algorithms construct the graph which can represent the structure of MCM and iteratively extend the graph by a heuristic. CSE algorithms find common subexpressions in a convenient representation of coefficients and share as many the subexpressions as possible. The subexpression means that some part of an expression. For example, 5**x* equals to (*x*<<1) + 3**x*, so x and 3**x* are subexpressions of 5**x*. Graph-based algorithms can usually get a better solution than CSE algorithms, because they are not restricted to a particular representation.

The characteristic of a filter is generally defined by the filter specification. Some of conventional FIR filter design flows first determine a set of coefficients which satisfies the given filter specification and then use the MCM algorithm to minimize the number of adders required for the corresponding MCM block. However, because there is no cost information available during the coefficient selection process, it is likely to find the other set of coefficients that is implemented by fewer adders and also satisfies the specification. That is, it can help minimize the adders cost if the cost can be properly estimated while selecting coefficients. Several works [4-6][9] have addressed this issue and provided fairly good outcomes. However, the solution space of this problem is simply too large so that it is impractical to perform an exhaustive search for the exhaustive optimal solution due to limited runtime.

In this thesis, we propose a new MCM-based FIR design methodology. Unlike previous techniques, besides addition and left-shift operations, right-shift operations are also allowed while construction the MCM block, which expands the design space. Moreover, we also develop a branch and bound (B&B) strategy to make the solution exploration in that expanded design space more efficiently and effectively. Experimental results show that the proposed methodology is capable of producing better solutions in acceptable runtime when compared with existing techniques.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we introduce the specification of the filter and the previous works. Chapter 3 explains the motivation of this work. In Chapter 4, the proposed method is demonstrated. The experimental results are shown in Chapter 5. Finally, Chapter 6 gives the conclusions and the future works.

Chapter 2
Background

2.1 The Specification of FIR Filter

In this thesis, we consider the linear phase FIR filter. The frequency response of a Type I linear phase FIR filter with *N* taps is written as

$$H(\omega) = h_M + 2\sum_{n=0}^{M-1} h_n \cos((M-n)\omega) \tag{2.1}$$

where

$$M = (N-1)/2 .$$

The frequency response equations of Type II, III and IV linear phase FIR filters are similar to this [14].

The frequency response of the filter can be classified into four types: low-pass, high-pass, band-pass and band-stop. For the sake of convenience, we just illustrate the low-pass filter in the following. Fig. 2 shows the specification of the low-pass filter. The parameters ω_p, ω_s, δ_p, δ_s are the end of the pass-band, the beginning of the stop-band, the maximum allowable pass-band ripple and the maximum allowable stop-band ripple, respectively. The specification means that the frequency response must be inside the region. Thus, it can be expressed as the formula in the following.

$$\begin{aligned} 1-\delta_p \le H(\omega)/\beta \le 1+\delta_p \quad & \text{for } \omega \in [0,\omega_p] \\ -\delta_s \le H(\omega)/\beta \le \delta_s \quad & \text{for } \omega \in [\omega_s,\pi] \end{aligned} \tag{2.2}$$

where

$$\beta = \frac{1}{2}\left[\max H(\omega) + \min H(\omega)\right] \text{ for } \omega \in [0,\omega_p]$$

is the average pass-band gain.

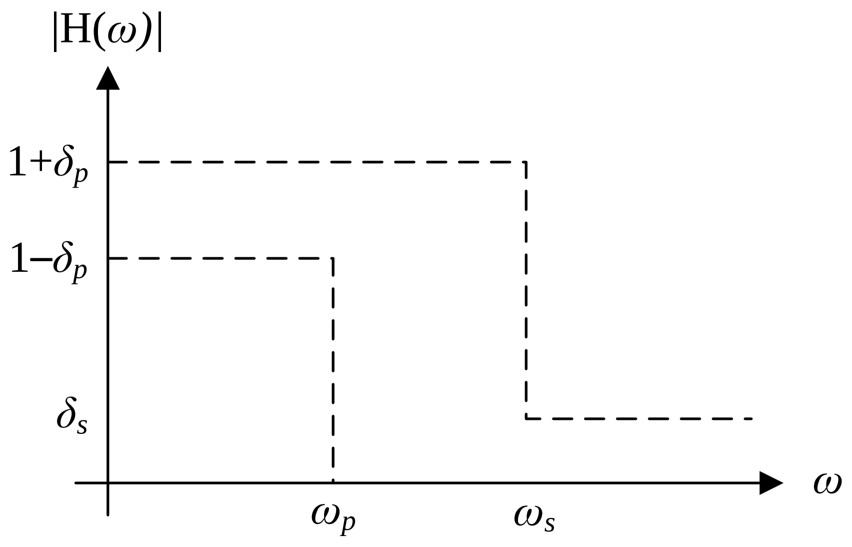


Fig. 2 The specification of a low-pass filter.

2.2 Previous Works

In the FIR filter design, in order to efficiently minimize the number of adders, we need to go back to the preceding process, that is, we must take account of the cost when determining a set of coefficients which satisfies the specification. Some previous works solve this problem [4-6][9], and they are briefly described here.

In [4], the work uses linear programming to derive the boundary of all coefficients which can meet the specification and searches coefficients within the boundary. The search method is the B&B that finds a better solution by first generating a look-up table containing all the possible subexpressions for a given wordlength and a given maximum number of adders per coefficient. It just considers the individual cost of each coefficient when generating the look-up table, so it possibly loses the better solutions.

In [5], the work formulates the problem as a 0-1 integer linear programming to minimize the number of adders. The formulation comprehensively considers the subexpression of each coefficient, but it needs a large number of variables to decide which coefficients and subexpressions are used, so it is very time consuming.

In [9], the work proposes a local search method and uses a common-subexpression-based method to account for the sharable adders. The canonical signed digit (CSD) representation is used. Although the representation can represent the coefficient with a minimum number of non-zero bits, it does not guarantee having the fewer number of adders than other representations.

In [6], the work uses the B&B search method in the boundary for each coefficient and proposes a cost estimation to minimize the number of adders. The cost estimation simply computes the required number of adders for generating a new coefficient by adding or shifting the integers in the subexpression basis set which is dynamically expanded during the search process. The experimental results show that the total number of adders in FIR design is fewer than other previous works under the same filter specification.

It is apparent that the scheme of first identifying a boundary and then performing a B&B search is widely adopted in coefficient decision as shown in [4][6]. The boundary computation can reduce the search space because we just need to search within the boundary of each coefficient. The B&B search strategy can eliminate invalid searches based on the filter specification and the total adder cost. We also adopt this B&B search and the boundary computation in our algorithm.

Chapter 3
Motivation

3.1 Right-Shifter in MCM

In the previous works [4-6][9], after the wordlength (WL) is decided, the value of every coefficient must be an integer ranging from -2^{WL} to $2^{WL}-1$ since an MCM block consists of adders and left-shifters only. For example, if the given wordlength is 10-bit, the value of each coefficient must be an integer between -1024 and 1023. However, we can reverse the sign of coefficients by replacing the structural adders by subtractors, so the range of coefficients is actually -1024~1024. If we consider the right-shifter, we can select the non-integers as the coefficients in certain range. For example, assume that the input data is x and 2.5 is the coefficient which we select, that is, the operation $2.5 * x$ is needed to be computed. This constant multiplication can be computed as $(5 * x) >> 1$. Applying the right-shift operation, we can select the non-integer 2.5 as the coefficient.

An example is given here. Assume the input data is x, and we select two coefficients, "3" and "20", for the given filter specification without right-shifter operation. The architecture is shown as Fig. 3(a), and it needs two adders. However, we can select the non-integer as the coefficient with the right-shifter operation. Assume that the coefficient "3" can be replaced by "2.5" and the set of coefficients still satisfies the filter specification. In this case, an adder can be replaced by a shifter, and the modified architecture is shown as Fig. 3(b). One adder can be saved to reduce the cost.

In another similar case, assume that there is no integer which satisfies the specification when determining the second coefficient. The previous works will return "no solution" in this case. However, with the right-shift operation, we can select "2.5" if it can satisfy the specification and thus a solution is available. In this way, applying right-shift can make it easier to find a feasible solution.

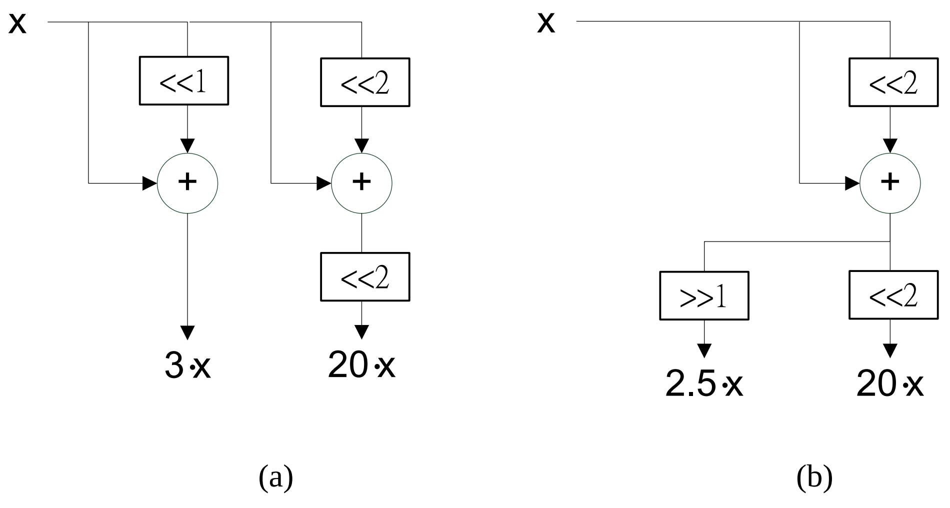


Fig. 3 Example of cost reduction

3.2 Heuristic Pruning Condition

After applying right-shift operation, the search space of coefficient sets is expanded and thus requires more time to find an exact solution from it. Similarly, the search space grows exponentially to the wordlength, thus in the previous works the wordlength can only be set to a value such that the run time is acceptable.

In this thesis, we introduce a heuristic pruning condition during the B&B search to reduce the run time. This heuristic pruning is based on the ripple of the frequency response, which implies the quality of current coefficient set. If the ripple is too large during B&B search, it hardly can find a feasible solution.

Applying the heuristic pruning may miss the best solution when searching in the design space. However, the run time is greatly reduced and thus it allows us to expand the search space. We found that in most cases searching heuristically in a larger design space is more effective than finding an exact solution in a smaller design space. Thus we apply this heuristic pruning in our algorithm.

3.3 Lower Bound Analysis

In order to illustrate that the right-shifter is beneficial for cost reduction, we compare the lower bounds of the number of adders between the design with and without right-shifters. The method for computing the lower bound will be explained in Section 4.6, and we just show the results here. Table I lists ten filters and their lower bound with and without right-shifters. The ten filters are randomly generated and their numbers of taps are lower, because the runtime of computing the lower bound is very long, and the higher-tap filter leads to longer runtime. Table I can show that the right-shifter is actually beneficial for cost reduction. However, the improvement is not large because the number of taps is small.

Table I The comparison for lower bound

Filter	Tap	LB without right-shifters	LB with right-shifters
M1	24	31	28
M2	23	28	26
M3	22	25	24
M4	24	28	27
M5	24	26	26
M6	23	31	30
M7	23	22	22
M8	22	26	24
M9	24	28	26
M10	24	26	25

3.4 Problem Formulation

In this thesis, we address the problem of the linear phase FIR filter design based on the MCM architecture. We are given:

- the wordlength of coefficients
- the specification of FIR filter: ω_p, ω_s, δ_p and δ_s

Our goal is to generate a set of coefficients and minimize the total number of structural adders (SA) and multiplier block adders (MBA) for the FIR filter design under the given filter specification constraint.

Chapter 4
Our Proposed Method

In this chapter, we propose an algorithm to determine coefficients for a specified linear phase FIR filter design. The target of our algorithm is to minimize the number of adders as the FIR filter is implemented through MCM. Besides, our algorithm allows the use of right shift operations in the MCM block to further expand the design space. Our method efficiently uses the B&B search to find a set of coefficients which has lower cost and satisfies the specification. The method uses the lower bound of MCM problem to estimate the cost and applies a heuristic bound condition in the B&B search.

4.1 Search of The Solutions

To find the set of coefficients which satisfy the specification and require fewer adders, we use the B&B algorithm same as the previous work [6]. In this previous work, they determine the coefficients from the smallest coefficient to the largest coefficient, that is, h_0, h_1, … , h_M in (2.1), because the larger coefficient can be composed of the smaller coefficients by adders and left-shifters. However, we use the right-shift operation, so we determine the coefficients in the reverse order. The reason is that the smaller coefficients can be derived by right-shifting of the larger coefficients. Fig. 4 shows the B&B tree of the 5-tap FIR filter. The coefficients are determined in the corresponding level, where each edge represents one decision of the coefficient, and each path represents one set of coefficients. For example, the Path 1 is a set of coefficients that contains $h_0 = 2$, $h_1 = 4$, and $h_2 = 7$. Of course, we have some pruning conditions to reduce the search time, and it is discussed in the following section.

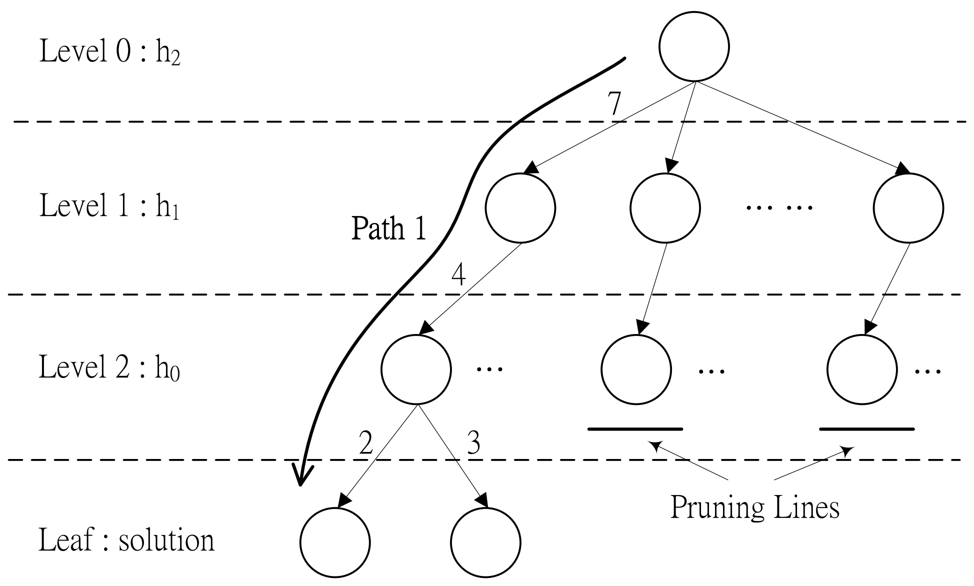


Fig. 4 The B&B tree of the Type I 5-tap FIR filter

4.2 Boundary Computation

Assume that the given wordlength of coefficients is *WL*, so the coefficients can be selected between – 2^{WL} and 2^{WL}. The search space of B&B is very large. In order to reduce the search space, it is needed to reduce the range of coefficients. In the FIR filter design, the set of coefficients is not unique for the same filter design specification, but we can compute the boundary for each coefficient according to the specification. The boundary means that the coefficients outside the boundary never satisfy the filter specification. By computing the boundary, we can reduce the search space and the runtime.

4.2.1 Linear Programming Formulation

To determine the boundary of the coefficient h_k, we formulate a linear programming (LP) model, and the formulation is written as

$$
\begin{aligned}
&\text{minimize: } h_k \\
&\text{subject to: } 1-\delta_p \le H(\omega)/\beta \le 1+\delta_p \quad \text{for } \omega \in [0,\omega_p] \\
&\qquad -\delta_s \le H(\omega)/\beta \le \delta_s \qquad \text{for } \omega \in [\omega_s,\pi] \\
&\qquad \beta_l \le \beta \le \beta_u \\
&\qquad -2^{WL} \le h_i \le 2^{WL} \qquad \text{for i=0,1,...,} M
\end{aligned}
\tag{4.1}
$$

where $H(\omega)$ are introduced in (2.1), the specification constraint is the same as (2.2), and β_l, β_u are two constants which specify the lower bound and the upper bound of β, respectively. Using (4.1), we can derive the lower bound of h_k. To derive the upper bound of h_k, replace **minimize** by **maximize** in (4.1). Using this LP model, we can derive the boundary of each coefficient. Eventually, we use LP solver, named gurobi [15] to solve this LP problem.

4.2.2 The Selection of β_l and β_u

Assume the largest coefficient is h_M, where M is the same as (2.1). First, set β_l and β_u as unity and compute the lower bound of h_M, denoted as $h_{M(low)}$. Secondly, set β_u as $2^{WL}/h_{M(low)}$ because the maximum of the coefficient is 2^{WL} in this design with the wordlength *WL*. For FIR filter design in binary arithmetic, the lower bound of β is unnecessary to be smaller than half of the upper bound [7], so β_l is set as $\beta_u/2$.

4.3 Cost Function and Zero-Crossing-Coefficient

Fig. 1(b) shows an *N*-tap FIR filter architecture with MCM, and the adders can be classified into SAs and MBAs. The goal of our work is minimizing the total number of adders which include SAs and MBAs. Assume that the number of SAs and MBAs are N_{SA} and N_{MBA}, respectively. The cost function can be written as N_{SA} + N_{MBA}. The SAs are used to sum up the outputs of MCM, so N_{SA} is related to the number of coefficients, that is, $N_{SA} = Tap - 1$, where

Tap is the number of coefficients. However, if there is one coefficient whose value is zero, the output of corresponding multiplication must be zero no matter what the multiplicand is. Therefore, the corresponding adders can be removed. Besides, the linear phase FIR coefficients are symmetric, so we can save two adders when one coefficient is fixed to zero. Thus, the cost function can be further written as $N_{MBA} + Tap - 1 - 2*N_{zero}$, where N_{zero} is the number of the coefficients which equal to zero. Then, the cost function can be simplified as $N_{MBA} - 2*N_{zero}$, because $Tap - 1$ is constant.

In order to reduce the cost, N_{zero} should be as large as possible. Therefore, we determine the zero-crossing-coefficients (ZCC) at first, and then the B&B search will determine the remaining coefficients. The ZCC means that the boundary of the coefficient is crossing zero. Moreover, the set of ZCCs which has more number of zeros are searched at first. For example, assume that the feasible boundaries of h_0, h_1 and h_2 include zero. At first, three ZCCs are fixed to zero, that is, { h_0, h_1, h_2 }. Then, two ZCCs are fixed to zero, that is, { h_0, h_1 } or { h_1, h_2 } or { h_0, h_2 }. Then, one ZCC is fixed to zero, that is, { h_0 } or { h_1 } or { h_2 }. Finally, no coefficient is fixed to zero.

4.4 Algorithm Flow

Fig. 5 shows the algorithm flow. First, the algorithm computes the boundary of each coefficient according to the given specification. After this step, we can just search the coefficients within the corresponding boundaries. Secondly, the algorithm uses the B&B search to determine the coefficients. An important characteristic of the B&B search is that finding a good solution as soon as possible will result in earlier bound, and can reduce the runtime. Therefore, we create an iteration loop above the B&B search such that we can fix the ZCCs to zero first. In this iteration loop, we first set N_{zero} as the number of ZCCs and fix N_{zero}

ZCCs to zero. Then we use the B&B search to determine the remaining coefficients. Making more ZCCs to zero can save more SAs. However, it may cause the B&B search fail to find a feasible solution. If failed, we will reduce the N_{zero}. This loop continues until all combination of ZCCs will be tried or a feasible solution is obtained.

The main stage of this algorithm is the B&B search. In this stage, the algorithm determines the remaining coefficients by the B&B search. This thesis proposes a B&B search strategy, and it is introduced in Section 4.5.

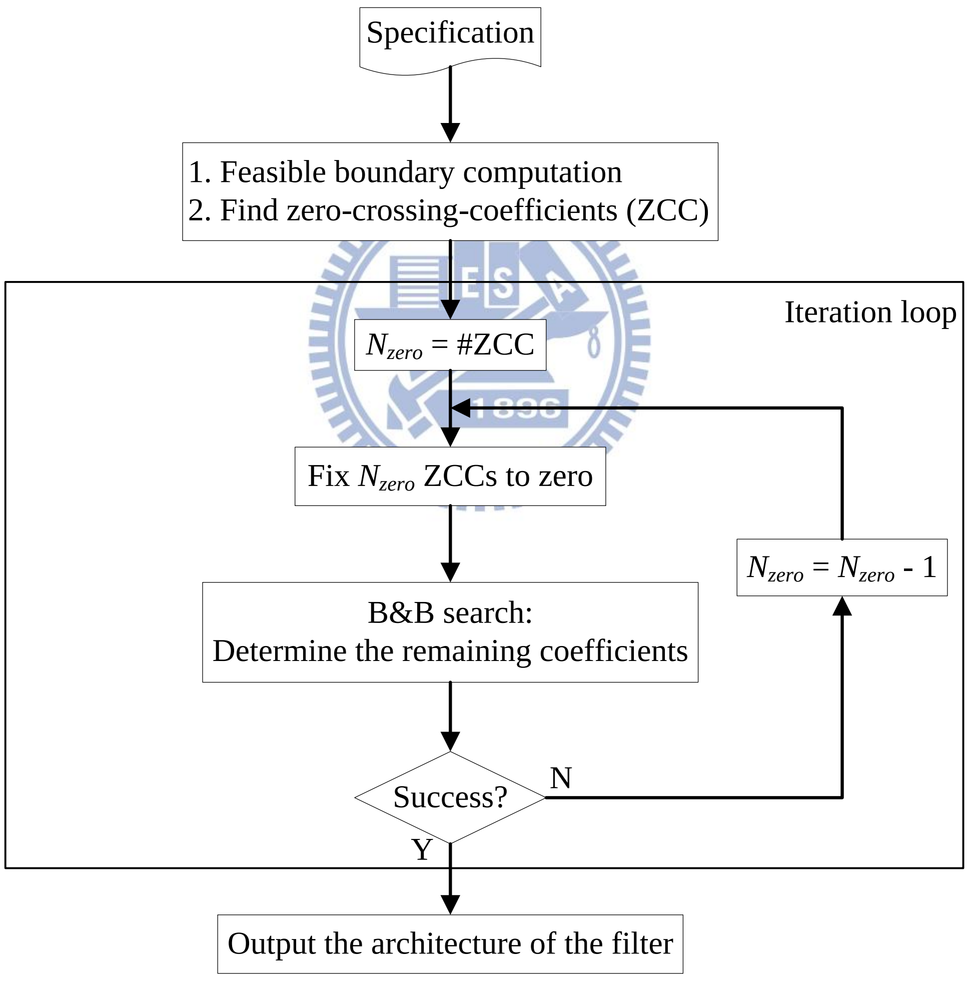

Specification
1. Feasible boundary computation
2. Find zero-crossing-coefficients (ZCC)
Iteration loop
N_{zero} = #ZCC
Fix N_{zero} ZCCs to zero
$N_{zero} = N_{zero} - 1$
B&B search:
Determine the remaining coefficients
Success?
N
Y
Output the architecture of the filter

Fig. 5　The algorithm flow

4.5 Branch and Bound Search

After fixing the ZCCs to zero, we will determine the remaining coefficients by the B&B search. In this section, we introduce the B&B search strategy to make the solution exploration in that expanded design space more efficiently and effectively.

4.5.1 Decision Flow

Applying the B&B search method, we need to do the coefficient decision in each node on the B&B tree. Fig. 6 shows the coefficient decision flow. Assume that the coefficient h_{k+1} is already determined, and the coefficient h_k will be determined this time. If k is equal to -1, the program already reaches the leaves of the B&B tree, so a satisfied set of coefficients is found. Then, we can record the result and go back to fix h_{k+1} to another candidate. If k is not equal to -1, the program will execute the following steps. Step 1, determine the candidate set, denoted as C, containing some values within the boundary of the coefficient h_k. Step 2, compute *LB* and *RIPPLE* for each candidate, which are used to determine the priority of search. Step 3, fix h_k to some value which belongs to C, and the priority is by ascending *LB*. When *LB*s are equivalent, the priority is by ascending *RIPPLE*. Step 4, check the pruning conditions. The path is pruned when matching the pruning conditions. The Step 5, if the pruning conditions are all not matched, the program goes to the decision of h_{k-1}. Else, go back to Step 4 to fix h_k to another candidate until try all candidates which belongs to C. When all candidates have been tried, if k does not equal to M, the program will go back to fix h_{k+1} to another candidate. If k equals to M, which means that the whole branch tree has been searched, the program is finished.

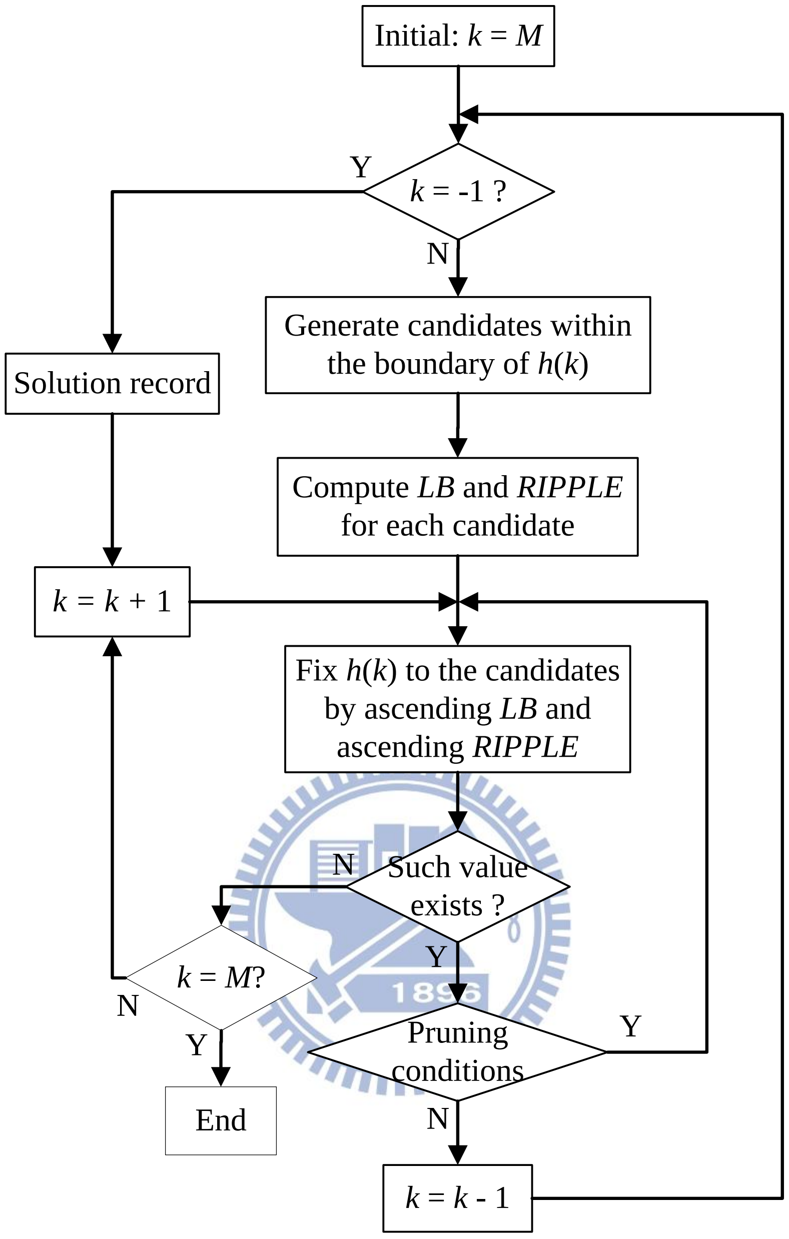

Initial: $k = M$
Y
$k = -1$?
N
Generate candidates within the boundary of $h(k)$
Solution record
Compute *LB* and *RIPPLE* for each candidate
$k = k + 1$
Fix $h(k)$ to the candidates by ascending *LB* and ascending *RIPPLE*
N
Such value exists ?
Y
$k = M$?
N
Y
Pruning conditions
Y
N
End
$k = k - 1$

Fig. 6 The decision flow

4.5.2 Candidate Selection

In this section, we explain about Step 2 in Fig. 6. The goal of Step 2 is determining the candidate set of the coefficient h_k. In Step 4, h_k will be fixed to each value in the candidate set in the certain order which was introduced in Section 4.5.1.

We will select values within the boundary of h_k as the candidates, because the values outside the boundary never satisfy the specification. However, the actual boundary of coefficient is much tighter than the initial boundary when more and more coefficients are fixed. Thus, if we want to derive the actual boundary, we must recompute the boundary by running the LP solver. In [6], a method is proposed to search the values without unnecessary LP runs, and we also adopt this method.

In order to reduce the number of running LP solver, we do not recompute the feasible boundaries of coefficients but compute them in the beginning just once when no coefficient is fixed. That is to say, use the LP model as (4.1) to compute the initial feasible boundaries of coefficients. The actual boundary of coefficient is much tighter than the initial boundary, so it is necessary to check whether a set of coefficients is satisfied. This problem can be solve by using a LP model as

$$\begin{aligned}
&\text{minimize: } \delta - \delta_p \cdot \beta \\
&\text{subject to: } \beta - \delta \le H(\omega) \le \beta + \delta && \text{for } \omega \in [0, \omega_p] \\
&\qquad -(\delta_s \cdot \delta)/\delta_p \le H(\omega) \le (\delta_s \cdot \delta)/\delta_p && \text{for } \omega \in [\omega_s, \pi] \\
&\qquad \beta_l \le \beta \le \beta_u \\
&\qquad -2^{WL} \le h_i \le 2^{WL} && \text{for each } h_i \text{ is unfixed} \\
&\qquad h_i = f_i && \text{for each } h_i \text{ is fixed}
\end{aligned} \qquad (4.3).$$

The model is proposed in [6]. δ is the pass-band ripple, and δ_p is the maximum allowable pass-band ripple. Therefore, if the objective function $\delta - \delta_p \cdot \beta$ is larger than 0, it means that no feasible solution satisfying the specification is available. Applying this LP model, we can check the satisfaction and avoid the unnecessary LP runs in the candidates selection.

The candidates of a coefficient consist of two types. The first type candidates are integer values within the coefficient boundary as in previous works. The second type candidates are non-integer values which can be derived from the former determined coefficients by the right-shift operation. Note that the right-shift operation is just applied at the output of the MCM block, because we derive non-integer values by right-shifting the existent coefficients. The right-shift operation may result in truncation error because of the non-integer property.

Extra fractional bits are required if no truncation error allowed. However, in the FIR filter design, the right-shift operation may be feasible, because the architecture of the FIR filter needs a series of adders to sum up the outputs of MCM that is shown as the SA of Fig. 1(b). The series of adders usually lead to the truncation error because the sizes of adders are not increased stage by stage in order to reduce hardware cost. Moreover, in fixed-point arithmetic, keeping all less-significant-bits after a multiplication is not necessary because of the quantization error already existed in input signals. Thus according to the output error requirement, a truncation procedure is often required to reduce area as shown in [13]. If such procedure is applied, the truncation error problem implied by right-shift operation can be tolerated or considered on the quantization problem of the FIR filter design.

The pseudo code of candidates selection (***CS***) is as follows.

```
CS ( $ub_k$, $lb_k$, $FC$, $x$ )
  1   $C_1 = \emptyset$ ;
  2   $C_2 = \emptyset$ ;
  3   for $v$ from $\lfloor x \rfloor$ to $lb_k$, $v$ is an integer
  4      $h_k = v$;
  5      if ( LP ($FC$) $\leq 0$ )
  6         add $v$ to $C_1$;
  7      else
  8         break;
  9   for $v$ from $\lceil x \rceil$ to $ub_k$, $v$ is an integer
 10      $h_k = v$;
 11      if ( LP ($FC$) $\leq 0$ )
 12         add $v$ to $C_1$;
 13      else
 14         break;
 15   foreach $v \in FC$
 16      while ( $v \geq lb_k$ )
 17         if ( $v \leq ub_k$ )
 18            $h_k = v$;
 19            if ( LP ($FC$) $< 0$ )
 20               add $v$ to $C_2$;
 21         $v = v / 2$;
 22   return $C_1 \cup C_2$
End
```

Note that ub_k, lb_k are the initial upper bound and the initial lower bound of h_k, respectively. And *FC* is the fixed coefficient set containing the coefficients which are already fixed. The x is a value which must satisfy the specification, and it can be derived when the LP runs for h_{k+1}. In line 1 and line 2, the integral candidate set C_1 and non-integral candidate set C_2 are empty initially. Two for loop in line 3 to line 14 add the integers which satisfy the specification to C_1. The first for loop in line 3 to line 8 searches the integers and checks the specification in one direction towards the initial lower bound or until unsatisfied; and the second for loop in line 9 to line 14 searches the integers and checks the specification in the other direction towards the initial upper bound or until unsatisfied. The third for loop in line 15 to line 21 searches the non-integers in the initial boundary. If the non-integer can satisfy the specification and be derived from the former determined coefficients by the right-shift operation, we add it to C_2. The last line 22 returns the union of C_1 and C_2 as the candidate set.

The search space is already very large even if the non-integer is not considered. In order to control the non-integral search space, we define a parameter L. We restrict the number of bits after binary point within L. Therefore, we can control the non-integral search space by modulating L. The experimental results show the performances with different L in Section 5.1.

4.5.3 *LB* and *RIPPLE* Computation

In this section, we explain the Step 2 in Fig. 6. The search space of B&B is very large. In order to speed up the search process, it is better to find a set of coefficients which satisfies the specification and has the low cost as soon as possible, so that the search could be early bounded. For this reason, we define two variables for each candidate in the candidate set, and we determine the search priority according to the two variables. The first one is *LB*, and it estimates the number of adders which is needed when fixing h_k to some candidate. The second one is *RIPPLE*, and it represents the quality of a candidate.

In order to reduce the time of cost computation, we use the lower bound of MCM design to estimate the number of adders. In [8], the lower bound of the number of adders for MCM design is proposed as

$$LB = \min_{i=0}^{N-1}\{\lceil \log_2 S(C_i) \rceil\} + N - 1 \tag{4.4}$$

where C_i are positive odd unique coefficients, N is the number of coefficients, and $S(C_i)$ is the minimal number of non-zero bits of C_i. We compute the *LB* for each candidate in the candidate set by (4.4), and the pseudo code is as follows.

```
LB_Compute(CS, FC)
    1. foreach $v \in CS$
    2.     $C = FC$;
    3.     add $v$ to $C$;
    4.     $lb$ = Lower_Bound($C$);
    5.     if ( $lb > BEST\_LB$ )
    6.         remove $v$ from $CS$;
    7.     else
    8.         $LB[v] = lb$
    9. return $LB$
End
```

Note that, *CS* is the candidate set, and *FC* is the fixed coefficient set. In line 1 to line 8, the program computes the lower bound for each candidate which is in the candidate set. In line 5 and line 6, the program removes the candidate whose *LB* are not smaller than *BEST_LB*, because it is very possible that the candidates result in worse cost than the best solution. In order to reduce the search space, we remove them from the candidate set. The *BEST_LB* is the minimal lower bound among the solutions which were already found.

In the candidate selection, we solve the LP model (4.3) to check the satisfaction, and the pass-band ripple δ actually can represent the quality of a set of coefficients. Because the smaller ripple means that it is more flexible to select the unfixed coefficients under the specification constraint. Therefore, we compute the *RIPPLE* for each candidate in the candidate set by deriving the pass-band ripple in (4.3). Because the β may be different for each candidate, we need to normalize the pass-band ripple as

$$RIPPLE = \delta / \beta \qquad (4.5).$$

The *RIPPLE* for each candidate can be recorded when the LP model is solved in the candidate selection, so no additional computation is needed.

4.5.4 Pruning Conditions

In this section, we introduce the pruning conditions of the B&B search. The pruning conditions are classified to two types: deterministic condition and heuristic condition. The deterministic condition means that it does not obstruct the obtainment of the best solution in the B&B search. On the contrary, the heuristic condition may lead to the loss of the performance, but it can reduce the search time. In our method, there are one deterministic condition: specification pruning condition, and two heuristic conditions: LB pruning condition and ripple pruning condition.

The specification pruning condition means that the set of fixed coefficients is unsatisfied even if we have not fixed all other coefficients yet. Since it is unsatisfied, this path is pruned. In fact, this condition check was done in the candidate selection, because we only add the values which satisfy the specification to the candidate set.

The LB pruning condition means that the lower bound of the number of adders for the fixed coefficients is not smaller than the minimal lower bound among the solutions which were already found. Actually, this condition check was done in the *LB* computation.

The ripple pruning condition means that the *RIPPLE* of the candidate is larger than *RIPPLE_Threshold* which is a dynamic value for each coefficient. We estimate that there is no satisfied solution after fixing a coefficient to a candidate which matches the ripple condition. The pseudo code of the B&B search is as follow, and it contains the ripple pruning.

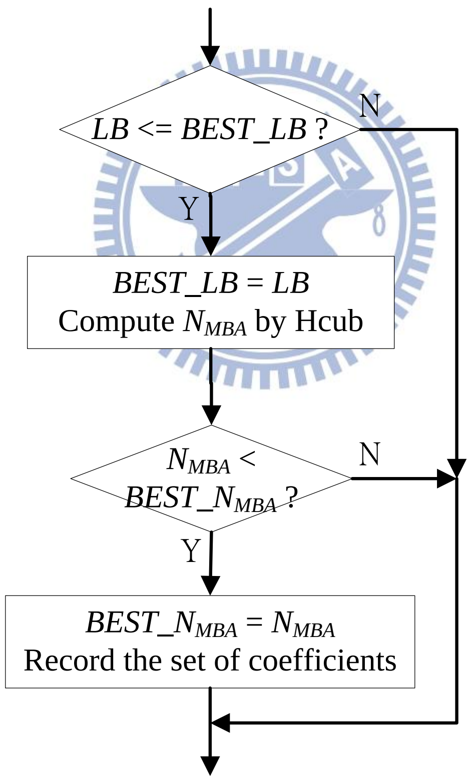
```
Initial: $RIPPLE\_Threshold$ [i] = 0, for $i = 0 \sim M$
BandB ( k, CB, RIPPLE_Threshold )
  1.  if ( $k == -1$ )
  2.      record the solution;
  3.      return TRUE;
  4.  $C$ = CS ($CB$);                          //candidates selection
  5.  {$LB$, $RIPPLE$} = LandR ();              //$LB$ and $RIPPLE$ computation
  6.  success = FALSE;
  7.  foreach $v \in C$ by ascending $LB$ and ascending $RIPPLE$
  8.      if ( $RIPPLE[v] < RIPPLE\_Threshold[k]$ )
  9.          success = TRUE;
  10.         $h_k = v$;
  11.         s = BandB($k$-1, $CB$, $RIPPLE\_Threshold$);
  12.         if ( s == FALSE)
  13.             $RIPPLE\_Threshold[k] = RIPPLE[v]$;
  14. return success;
End
```

The pseudo code is recursive, and the *RIPPLE_Threshold* for each h_i is initially set as infinity. Line 1 to line 3 is the terminal condition. Line 4 is the candidates selection, and line 5 is the *LB* and *RIPPLE* computation. Line 7 to line 13 fixes the coefficient h_k to each candidate and dynamically changes *RIPPLE_Threshold*. When determining a coefficient h_k, if a candidate of h_k failed to find any feasible solution, its *RIPPLE* will be recorded as the *RIPPLE_Threshold* of h_k. Later in the search, when the *RIPPLE* of any candidate of h_k is larger than the *RIPPLE_Threshold* of h_k, that branch will be pruned heuristically. The reason is that we estimate that there may be no satisfied solution if the *RIPPLE* of a candidate is larger than the *RIPPLE_Threshold*.

Applying the two heuristic pruning conditions may lose the best solution in the design space. However, the run time is greatly reduced and thus allowing us to expand the search space. We found that in most cases searching heuristically in a larger design space is more effective than finding an exact solution in a smaller design space. Thus we apply these heuristic pruning conditions in our method.

4.5.5 Solution Record

When k = -1 in Fig. 6, it means that the program has reached the leaves of the B&B tree. Therefore, we have a satisfied set of coefficients, and we can compute the real number of adders for this coefficient set. In the Section 4.3, we already explain that the cost function is $N_{MBA} - 2*N_{zero}$. In this step, we use Hcub [1], which is a graph-based MCM algorithm, to compute N_{MBA} for this set of coefficients. The best solution of coefficient sets is kept until the B&B tree is thoroughly searched and the filter architecture is obtained. The flow of solution record is shown in Fig. 7.



LB <= $BEST\_LB$?
N
Y
$BEST\_LB = LB$
Compute N_{MBA} by Hcub
N_{MBA} < $BEST\_N_{MBA}$?
N
Y
$BEST\_N_{MBA} = N_{MBA}$
Record the set of coefficients

Fig. 7 The flow of solution record

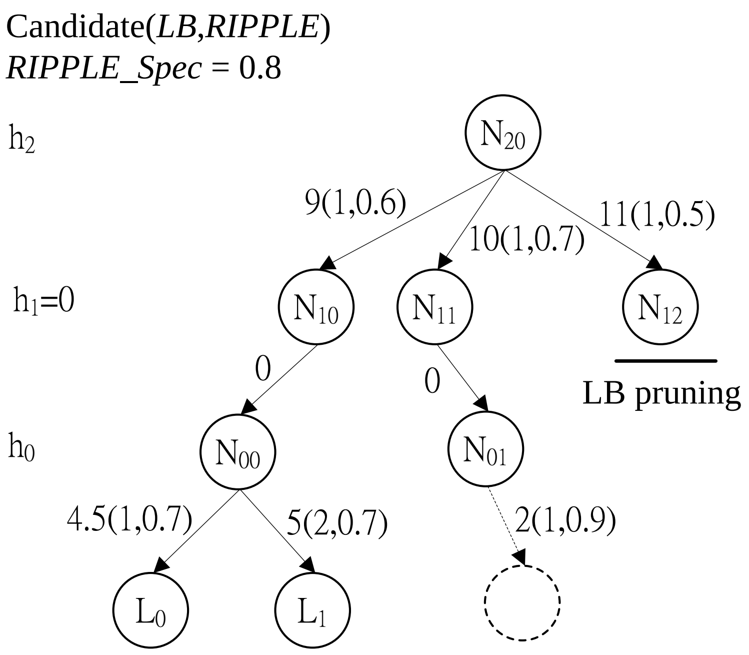


Fig. 8 The example of B&B search

4.5.6 The Example of B&B Search

In Fig. 8, we present an example showing how the B&B search works. This is a 5-tap linear phase filter, with 3 coefficients h_2, h_1 and h_0. The h_1 is a ZCC and we decide to fix it to zero in the iteration loop. Thus in the B&B search, h_1 will be locked to zero and only h_2 and h_0 will be determined.

Firstly, we consider h_2 because it is the largest coefficient. This is the first coefficient and only integer candidates are available. According to its boundary, the three available candidates are “9”, “10” and “11”. Then we will compute the *LB* and *RIPPLE* of them, the values are shown in Fig. 8. Note that the *RIPPLE* is computed with h_1 being fixed to zero. Then we will decide the priority of these candidates. According to the *LB* value, candidate “9” and “10” has higher priority. And then according to the *RIPPLE* we decide to branch on candidate “9” first.

Because the h_1 is fixed to zero, only one branch is allowed in the h_1-level. Then we go to the next level to determine h_0. Here we have two candidates, “5” is an integer and “4.5” is a non-integer by right-shifting “9”. These two candidates can generate two leaves L_0 and L_1.

The path reaching L_0 has smaller cost, so we record it as the currently best solution.

Now the branch of N_{10} is finished, so we go to the next candidate “10”. When go down to h_0 at N_{01}, we found that there is no candidate available. That means this branch is pruned by the specification pruning condition.

Later in the third candidate “11”, the *LB* of “11” is 2, which is larger the *LB* of the currently best solution and is pruned by the cost pruning condition. Now the whole B&B tree is searched, and the path reaching L_0 is the best solution, that is, “9”, “0”, and “4.5”.

4.6 Lower Bound Computation

In order to tabulate Table I for illustrating the benefit of right-shifters, we need to compute the lower bound of the number of adders for the filter design under the specification constraint. The method is almost the same as our method excluding the ripple condition and the cost computation by Hcub, because the ripple condition possibly leads to the loss of the real lower bound. Finally, the *BEST_LB* which is introduced in Section 4.5.5 is the lower bound of the number of adders for the filter design under the specification constraint. Without the ripple condition, the runtime will significantly increase, and this is the reason that the analysis restricts the low-tap filters.

Chapter 5
Experimental Results

In this chapter, three case studies are given to demonstrate the improvement of our algorithm. In Case Study I, the experimental results show the performances with different *L*. In Case Study II, eight filter cases in [6] are designed by using our algorithm, and the results are compared with the reported results of the best published work [6]. In Case Study III, we implement the algorithm in [6] and generate 8 filter specifications with different taps. The 8 filter cases are designed by using our algorithm, and the results are compared with the results by the algorithm in [6]. In this work, the algorithm is developed in C++/Linux environment, and the platform is built in Intel Xeon at 2.53GHz with 50GB of main memory.

Table II illustrates 8 filter specifications which are from [6]. In Table II, ω_p, ω_s, δ_p, δ_s are the end of the pass-band, the beginning of the stop-band, the maximum allowable pass-band ripple and the maximum allowable stop-band ripple, respectively.

Table II The specification of filter cases from [6]

Filter	Tap	ω_p	ω_s	δ_p	δ_s
X1	15	0.2π	0.8π	0.0001	0.0001
G1	16	0.2π	0.5π	0.001	0.001
S1	24	0.3π	0.5π	0.0157	0.0066
Y1	30	0.3π	0.5π	0.00316	0.00316
Y2	38	0.3π	0.5π	0.001	0.001
A1	59	0.125π	0.225π	0.01	0.001
S2	60	0.042π	0.14π	0.012	0.001
L2	63	0.2π	0.28π	0.028	0.001

5.1 Case Study I

In this case study, we compare the results with different *L*. Y2 and S2 are the test cases, and their specifications are listed in Table II. In Table III, the **FLB** is the lower bound of the number of adders for the filter design, and the **Total #adders** is the total number of adders for the filter design by our algorithm. The experimental results show that the large *L* can reduce the total number of adders and the runtime do not increase with *L*. That reason is that the larger *L* means the larger search space, so it is more possible to find a better solution earlier. Therefore, the search can be bounded earlier, and the runtime can be reduced.

From this case study, we can know the larger *L* is better, so we set *L* as same as the wordlength of the coefficients in the following case studies.

Table III The analysis for *L* in Y2 and S2

Y2				S2		
L	FLB	Total #adders	Runtime	*L*	Total #adders	Runtime
1	39	39	4m49s	1	74	18m27s
2	38	39	5m30s	2	74	22m50s
3	37	38	4m35s	3	74	27m34s
4	37	38	4m47s	4	73	27m56s
5	37	38	4m53s	5	73	28m21s
6	37	38	4m54s	6	73	29m30s
7	36	38	4m53s	7	72	26m31s
8	36	38	4m54s	8	72	26m14s
9	36	38	4m53s	9	72	26m36s
10	36	38	4m54s	10	72	26m40s

5.2 Case Study II

In this case study, there are 8 filters from [6], and their specifications are listed in Table II. In [6], the experimental results show that their results are better than the best published ones, so ours work is only compared with them. The design results of our algorithm and [6] are listed in Table IV for comparison. **Tap** is the number of coefficients, and **WL** is the wordlength of the coefficients excluding the sign bit. **MBA** is the number of multiplier block adders, **SA** is the number of structural adders, and **Total** is the total number of adders. In Table IV, the performances of ours and [6] are similar in the lower-tap filters. However, in the higher-tap filters, the performances of ours are better, and the runtime of ours are shorter.

Table IV The results and comparisons for cases from [6]

Filter	Tap	WL	[6]	Ours	Runtime
			MBA / SA / Total	MBA / SA / Total	[6] / Ours
X1	15	10	5 / 8 13	5 / 8 13	1s / 5s
G1	16	6	2 / 15 / 17	2 / 13 / 15	1s / 1s
		7	2 / 13 / 15	2 / 13 / 15	1s / 1s
S1	24	7	4 / 19 / 23	4 / 19 / 23	1s / 3s
Y1	30	9	7 / 23 / 30	6 / 23 / 29	6s / 1m20s
		10	6 / 23 / 29	6 / 23 / 29	5m9s / 4m2s
Y2	38	10	10 / 37 / 47	9 / 29 / 38	11s / 4m53s
		11	10 / 27 / 37	10 / 27 / 37	40m46s / 2h31m
A1	59	10	14 / 54 / 68	14 / 52 / 66	50h34m / 3h24m
S2	60	10	17 / 59 / 76	15 / 57 / 72	16h42m / 24m18s
L2	63	10	17 / 56 / 73	13 / 56 / 69	16h28m / 8h47m

5.3 Case Study III

In order to compare the results of the filters with different taps, we generate 8 filters, and the taps of these filters are from 39 to 89. Table V illustrates the 8 specifications of the filters. Similarly, ω_p, ω_s, δ_p and δ_s are the end of the pass-band, the beginning of the stop-band, the maximum allowable pass-band ripple and the maximum allowable stop-band ripple, respectively. The design results of our algorithm and [6] are listed in Table VI for comparison. In this case study, WL is always set as 10. In Table VI, the **Tap** is the minimal tap that a feasible solution can be found in the corresponding algorithm. The **Improvement** is derived by (**[6] - Ours**) / **ours**. The experimental results show that our improvement can be up to 30.6% and on average 13.8% in the number of adders. Note that the improvements are lower in the higher-tap filters. The reason is that there are more adders in SAs which dominate the total number of adders, and the number of adders in SAs can only be reduced by selecting the ZCCs.

Table V The specification of filter cases

Filter	ω_p	ω_s	δ_p	δ_s
T1	0.24π	0.43π	0.001	0.001
T2	0.32π	0.51π	0.001	0.001
T3	0.18π	0.3π	0.01	0.001
T4	0.24π	0.38π	0.0012	0.001
T5	0.28π	0.38π	0.01	0.001
T6	0.24π	0.34π	0.003	0.001
T7	0.08π	0.18π	0.001	0.0005
T8	0.04π	0.1π	0.02	0.001

Table VI The results and comparisons for the cases

Filter	Tap		[6]			Ours			Improvement (%)			Runtime	
	[6]	ours	MBA	SA	Total	MBA	SA	Total	MBA	SA	Total	[6]	ours
T1	40	39	10	39	49	8	26	34	20.0	33.3	30.6	3m9s	3m31s
T2	41	38	14	38	52	10	35	45	28.6	7.9	13.5	20s	1m38s
T3	48	48	18	47	65	13	45	58	27.8	4.3	10.8	4m36s	12m53s
T4	52	50	14	51	65	12	45	57	14.3	11.8	12.3	3m8s	10m24s
T5	56	56	18	55	73	12	51	63	33.3	7.3	13.7	18m28s	14m41s
T6	67	62	19	62	81	14	59	73	26.3	4.8	9.9	27m33s	29m12s
T7	78	75	23	77	100	17	72	89	26.0	6.5	11.0	4h35m	44m31s
T8	89	89	31	86	117	21	86	107	32.3	0.0	8.5	> 48h	3h3m
Avg.									26.1	9.5	13.8		

Fig. 9 shows the runtimes for the 8 filters with sorted order by the taps of filters. Note that the runtime of [6] in case T8 is more than 48 hours, and the best solution in the incomplete search is reported. It can obviously illustrate that the runtime of [6] significantly increases with the tap of the filter but ours do not.

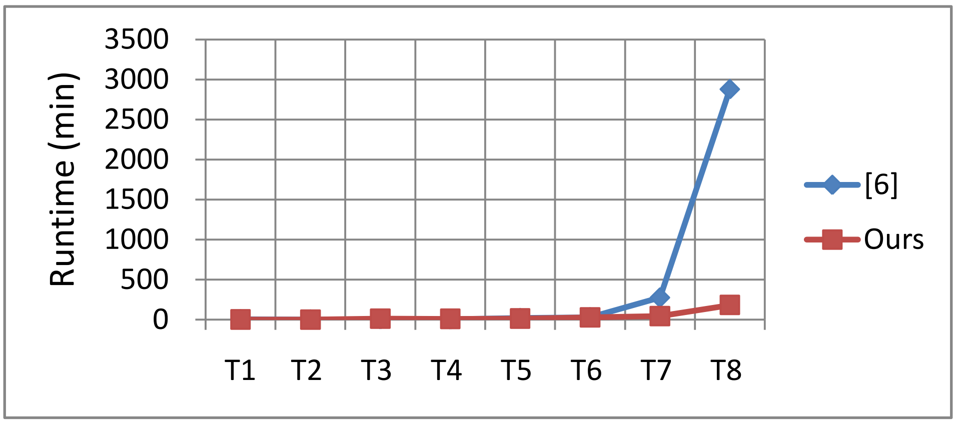


Fig. 9 Runtime for the filters with different tap

5.4 Design Example

In Table VII and Table VIII, we list the results of L2 and T5 by applying our algorithm. These results include the coefficients and the implementations of coefficients. First, we use adders, subtractors and left-shifters to design the basic subexpressions. Then, we implement the coefficients by left-shifting or right-shifting the basic subexpressions. Note that, we only construct the absolute value of coefficients in the MCM block. The negative coefficients are implemented by replacing the corresponding SAs by subtractors, so it is unnecessary to use negations.

Table VII The results of L2

Filter: L2 $h(n) = h(62 – n)$ for $0 \le n \le 30$ Pass-band gain: 4463.44			
$h(0) = 2.078125$	133 >> 6	$h(16) = – 64$	– (1 << 6)
$h(1) = 4.75$	19 >> 2	$h(17) = – 48$	– (3 << 4)
$h(2) = 6.71875$	215 >> 5	$h(18) = 0$	
$h(3) = 5.625$	45 >> 3	$h(19) = 61$	61
$h(4) = 0$		$h(20) = 103$	103
$h(5) = – 9.5$	– (19 >> 1)	$h(21) = 96$	3 << 5
$h(6) = – 19$	– 19	$h(22) = 29.1875$	467 >> 4
$h(7) = – 23$	– 23	$h(23) = – 79$	– 79
$h(8) = – 16.625$	– (133 >> 3)	$h(24) = – 180$	– (45 << 2)
$h(9) = 0$		$h(25) = – 215$	– 215
$h(10) = 22.5$	– (45 >> 1)	$h(26) = – 133$	– 133
$h(11) = 41$	41	$h(27) = 76$	19 << 2
$h(12) = 43.375$	347 >> 3	$h(28) = 377$	377
$h(13) = 24$	3 << 3	$h(29) = 694$	347 << 1
$h(14) = – 10.84375$	– (347 >> 5)	$h(30) = 934$	467 << 1
$h(15) = – 46$	– (23 << 2)	$h(31) = 1024$	1 << 10
Basic subexpressions			
3 = 1<<2 – 1	61 = 1<<6 – 3	133 = 19<<3 – 19	377 = 467 – 45<<1
19 = 1<<4 + 3	41 = 1<<6 – 23	215 = 3<<6 + 23	
23 = 3<<3 – 1	79 = 3 + 19<<2	467 = 1<<9 – 45	
45 = 3<<4 – 3	103 = 61<<1 – 19	347 = 19 + 41<<3	

Table VIII The results of T5

Filter: T5 $h(n) = h(55 - n)$ for $0 \leq n \leq 27$ Pass-band gain: 3310.29			
$h(0) = 2$	1 << 1	$h(14) = 45$	45
$h(1) = 2.875$	23 >> 3	$h(15) = 6.5$	13 >> 1
$h(2) = 0$		$h(16) = -52$	13 << 2
$h(3) = -7.1875$	– (115 >> 4)	$h(17) = -72$	– (9 << 3)
$h(4) = -13$	– 13	$h(18) = -18$	– (9 << 1)
$h(5) = -9.25$	– (37 >> 2)	$h(19) = 76$	19 << 2
$h(6) = 2.875$	23 >> 3	$h(20) = 120$	15 << 3
$h(7) = 16$	16 = 1 << 4	$h(21) = 46$	23 << 1
$h(8) = 15$	15	$h(22) = -115$	– 115
$h(9) = -3.65625$	– (117 >> 5)	$h(23) = -221$	– 221
$h(10) = -26$	– (13 << 2)	$h(24) = -117$	– 117
$h(11) = -27.625$	– (221 >> 3)	$h(25) = 236$	59 << 2
$h(12) = 0$		$h(26) = 698$	349 << 1
$h(13) = 37$	37	$h(27) = 1024$	1 << 10
Basic subexpressions			
9 = 1<<3 + 1 15 = 1<<4 – 1 13 = 1<<2 + 9	19 = 1 + 9<<1 23 = 1<<5 – 9 37 = 1 + 9<<2	45 = 9<<2 + 9 59 = 15<<2 – 1 115 = 1<<7 – 13	117 = 59<<1 – 1 221 = 13<<4 + 13 349 = 23<<4 – 19

Chapter 6
Conclusions & Future Works

In this thesis, a design method of the FIR filter is proposed to minimize the total number of adders on the architecture of the FIR filter. This method is based on the B&B search and finds a set of coefficients which requires fewer adders to implement. Besides, this method utilizes the right-shifter in the MCM design. The utilizing of the right-shifter expands the search space to find a better solution. We also propose a heuristic pruning condition to make the solution search in larger design space. The heuristic pruning condition is based on the ripple of the frequency response and it can be used to effectively reduce the runtime and still maintain the better performance.

Compared to the best published work [6], our method can reduce up to 30.6% and on average 13.8% in the number of adders. Besides, the runtime of our method is not significantly increased with the tap of the filter as that of method proposed in [6].

In this thesis, we explained that the right-shifter may cause the truncation error. In the future, we can consider the truncation error from the right-shifter onto the quantization problem of the FIR filter design.

References

[1] Y. Voronenko and M. Pu̇schel, “Multiplierless multiple constant multiplication,” in *ACM Transactions on Algorithms*, 2007, vol. 3, pp. 11.

[2] A. G. Dempster and M. D. Macleod, “Use of minimum-adder multiplier blocks in FIR digital filters,” in *IEEE Transactions on Circuits and Systems I*, 1995, vol. 42, pp. 569-577.

[3] A. G. Dempster and M. D. Macleod, “Using all signed-digit representations to design single integer multipliers using subexpression elimination,” in *International Symposium on Circuits and Systems*, 2004, vol. 3, pp. III-165-8.

[4] J. Yli-Kaakinen and T. Saramaki, “A systematic algorithm for the design of multiplierless FIR filters,” in *International Symposium on Circuits and Systems*, 2001, vol. 2, pp. 185-188.

[5] O. Gustafsson and L. Wanhammar, “Design of linear-phase FIR filters combining subexpression sharing with MILP,” in *Midwest Symposium on Circuits and Systems*, 2002, vol. 3, pp. III-9-III-12.

[6] S. Dong and Y. Y. Jun, “Design of linear phase FIR filters with high probability of achieving minimum number of adders,” in *IEEE Transactions on Circuits and Systems I,* 2011, vol. 58, pp. 126-136.

[7] Y. C. Lim, “Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude,” in *IEEE Transactions on Circuits and Systems*, 1990, vol. 37, pp. 1480-1486.

[8] O. Gustafsson, “Lower bounds for constant multiplication problems,” in *IEEE Transactions on Circuits and Systems II*, 2007, vol. 54, pp. 974-978.

[9] X. Fei, C. C. Hong, and J. C. Chuen, “Design of low-complexity FIR filters based on signed-powers-of-two coefficients with reusable common subexpressions,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2007, vol. 26, pp. 1898-1907.

[10] Y. Y. Jun and L. Y. Ching, “Design of linear phase FIR filters in subexpression space using mixed integer linear programming,” in *IEEE Transactions on Circuits and Systems I*, 2007, vol. 54, pp. 2330-2338.

[11] Y. Yu and Y. Lim, “Optimization of linear phase FIR filters in dynamically expanding subexpression space,” in *Circuits, Systems, and Signal Processing*, 2010, vol. 29, pp. 65-80.

[12] M. Aktan, A. Yurdakul, and G. Dundar, “An algorithm for the design of low-power hardware-efficient FIR filters,” in *IEEE Transactions on Circuits and Systems I*, 2008, vol.55, pp. 1536-1545.

[13] R. Guo, L. S. DeBrunner, and K. Johansson, “Truncated MCM using pattern modification for FIR filter implementation,” in *International Symposium on Circuits and Systems*, 2010, pp. 3881-3884.

[14] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, “Discrete-time signal processing,” 2 ed. New Jersey: Prentice-Hall, 1989, pp. 297-300.

[15] Gurobi Optimization, Inc. “Gurobi Optimizer.” Internet: http://www.gurobi.com, 2012.