A Low-Cost Fir Filter Design Based On Multiple Constant Multiplication/Accumulation Using Booth Multiplier

G. Srilakshmi¹, P.Ratna Kavya²


ABSTRACT: Low-cost finite impulse response (FIR) designs are presented using the concept of multipliers with the optimization of bit width and hardware resources without sacrificing the frequency response and output signal precision. Non-uniform coefficient quantization with proper filter order is proposed to minimize total area and cost. Multiple constant multiplication/accumulation in a direct FIR structure is implemented using an improved version of Booth multipliers. In this proposed method a booth multiplier is implemented. In Booth multiplier to multiply the signed numbers is an added advantage. Comparisons with previous FIR design approaches show that the proposed designs achieve the best area and power results.

KEYWORDS: Digital signal processing (DSP), faithful rounding, finite impulse response (FIR) filter, truncated multipliers, VLSI design

I. INTRODUCTION

Finite impulse response (FIR) digital filter is one of the fundamental components in many digital signal processing (DSP) and communication systems. It is also widely used in many portable applications with limited area and power budget. A general FIR filter of order $M$ can be expressed as

$$y[n] = \sum_{i=0}^{M-1} a_i x[n-i].$$

In case of linear phase, the coefficients are either symmetric or anti-symmetric with $a_i = a_{M-i}$ or $a_i = -a_{M-i}$.

There are two basic FIR structures, direct form and transposed form, the multiple constant multiplication (MCM)/accumulation (MCMA) module performs the concurrent multiplications of individual delayed signals and respective filter coefficients, followed by accumulation of all the products. Thus, the operands of the multipliers in MCMA are delayed input signals $x[n-i]$ and coefficients $a_i$. 
The operands of the multipliers in the MCM module are the current input signal $x[n]$ and coefficients. The results of individual constant multiplications go through structure adders (SAs) and delay elements. In order to avoid costly multipliers, most prior hardware implementations of digital FIR filters can be divided into two categories: multiplier based and memory based.

Multiplier-based designs realize MCM with shift-and-add operations and share the common sub operations using canonical signed digit (CSD) recoding and common sub-expression elimination (CSE) to minimize the adder cost of MCM. The more area savings are achieved by jointly considering the optimization of coefficient quantization and CSE. Most multiplier MCM-based FIR filter designs use the transposed structure to allow for cross-coefficient sharing and tend to be faster, particularly when the filter order is large. However, the area of delay elements is larger compared with that of the direct form due to the range expansion of the constant multiplications and the subsequent additions in the SAs. Blad and Gustafsson presented high-throughput (TP) FIR filter designs by pipelining the carry-save adder trees in the constant multiplications using integer linear programming to minimize the area cost of full adders (FAs), half adders (HAs), and registers (algorithmic and pipelined registers).

In this brief, we present low-cost implementations of FIR filters based on the direct structure with Booth multipliers. The MCMA module is realized by accumulating all the partial products (PPs) where unnecessary PP bits (PPBs) are removed without affecting the final precision of the outputs. The bit widths of all the filter coefficients are minimized.
using non-uniform quantization with unequal word lengths in order to reduce the hardware cost while still satisfying the specification of the frequency response.

II. COEFFICIENT QUANTIZATION AND OPTIMIZATION

A generic flow of FIR filter design and implementation can be divided into three stages: finding filter order and coefficients, coefficient quantization, and hardware optimization, in the first stage, the filter order and the corresponding coefficients of infinite precision are determined to satisfy the specification of the frequency response. Then, the coefficients are quantized to finite bit accuracy. Finally, various optimization approaches such as CSE are used to minimize the area cost of hardware implementations. Most prior FIR filter implementations focus on the hardware optimization stage.

![Fig 3 Proposed algorithms of coefficient quantization and fine tuning](image_url)

In this brief, we adopt the direct FIR structure with MCMA because the area cost of the flip-flops in the delay elements is smaller compared with that of the transposed form. Furthermore, we jointly consider the three design stages in order to achieve more efficient hardware design with faithfully rounded output signals.
After coefficient quantization, we perform recoding to minimize the number of nonzero digits. In this brief, we consider CSD recoding with digit set of \{0, 1, -1\} and radix-4 modified Booth recoding with digit set of \{0, 1, -1, 2, -2\} and select the one that results in smaller area cost.

While most FIR filter designs use minimum filter order, we observe that it is possible to minimize the total area by slightly increasing the filter order. Therefore, the total area of the FIR filter is estimated using the subroutine area_cost and estimate(). Indeed, the total number of PPBs in the MCMA is directly proportional to the number of FA cells required in the PPB compression because a FA reduces one PPB.

After Step 1 of uniform quantization and filter order optimization, the non-uniform quantization in Step 2 gradually reduces the bit width of each coefficient until the frequency response is no longer satisfied. Finally, we fine-tune the non-uniformly quantized coefficients by adding or subtracting the weighting of LSB of each coefficient and check if further bit width reduction is possible. We can find the filter order $M$ and the non-uniformly quantized coefficients that lead to minimized area cost in the FIR filter implementation.

III. BOOTH MULTIPLIER

It is a powerful algorithm for signed-number multiplication, which treats both positive and negative numbers uniformly.

For the standard add-shift operation, each multiplier bit generates one multiple of the multiplicand to be added to the partial product. If the multiplier is very large, then a large number of multiplicands have to be added. In this case the delay of multiplier is determined mainly by the number of additions to be performed. If there is a way to reduce the number of the additions, the performance will get better.

Booth algorithm is a method that will reduce the number of multiplicand multiples. For a given range of numbers to be represented, a higher representation radix leads to fewer digits. Since a k-bit binary number can be interpreted as K/2-digit radix-4 number, a K/3-digit radix-8 number, and so on, it can deal with more than one bit of the multiplier in each cycle by using high radix multiplication. This is shown for Radix-4 in the example below.

![Fig 5 Radix-4 multiplication in dot notation](image-url)
As shown in the figure above, if multiplication is done in radix 4, in each step, the partial product term \((B_{i+1}B_i)_2\) A needs to be formed and added to the cumulative partial product. Whereas in radix-2 multiplication, each row of dots in the partial products matrix represents 0 or a shifted version of A must be included and added.

Table 1 below is used to convert a binary number to radix-4 number. Initially, a “0” is placed to the right most bit of the multiplier. Then 3 bits of the multiplicand is recoded according to table below or according to the following equation:

\[
Z_i = -2x_{i+1} + x_i + x_i-1
\]

Example:
Multiplier is equal to 0 1 0 1 1 1 0
then a 0 is placed to the right most bit which gives 0 1 0 1 1 1 0 0
the 3 digits are selected at a time with overlapping left most bit as follows:

<table>
<thead>
<tr>
<th>(x_{i+1})</th>
<th>(x)</th>
<th>(x_i)</th>
<th>(Z_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1

For example, an unsigned number can be converted into a signed-digit number radix 4:
\((10 01 11 10 10 11 10)_2 = (-2 2 -1 2 -1 0 -2)_4\)
The Multiplier bit-pair recoding is shown in Table 2

<table>
<thead>
<tr>
<th>(x_{i+1})</th>
<th>(x)</th>
<th>(x_i)</th>
<th>(+2 \times \text{multiplicand})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+1 \times \text{multiplicand}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>+1 \times \text{multiplicand}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+2 \times \text{multiplicand}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-2 \times \text{multiplicand}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1 \times \text{multiplicand}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1 \times \text{multiplicand}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-2 \times \text{multiplicand}</td>
</tr>
</tbody>
</table>

Table 2

Here \(-2 \times \text{multiplicand}\) is actually the 2s complement of the multiplicand with an equivalent left shift of one bit position. Also, \(+2 \times \text{multiplicand}\) is the multiplicand shifted left one bit position which is equivalent to multiplying by 2.
To enter ±2*multiplicand into the adder, an (n+1)-bit adder is required. In this case, the multiplicand is offset one bit to the left to enter into the adder while for the low-order multiplicand position a 0 is added. Each time the partial product is shifted two bit positions to the right and the sign is extended to the left.

During each add-shift cycle, different versions of the multiplicand are added to the new partial product depends on the equation derived from the bit-pair recoding table above.

Let’s see some examples:

Example 1:

\[
\begin{array}{c}
000011 & (+3) \\
\times & 011101 & (+29) \\
\hline
+2-1+1 \\
000000000011 \\
111111101 \\
00000110 \\
1 \leftrightarrow 000001010111 & (+87)
\end{array}
\]

Example 2:

\[
\begin{array}{c}
111101 & (-3) \\
\times & 011101 & (+29) \\
\hline
+2-1+1 \\
111111111101 \\
0000000011 \\
11111010 \\
111110101001 & (-87)
\end{array}
\]

Example 3:

\[
\begin{array}{c}
111101 & (-3) \\
\times & 100011 & (-29) \\
\hline
-2+1-1 \\
000000000011 \\
1111111101 \\
00000110 \\
1 \leftrightarrow 000001010111 & (+87)
\end{array}
\]

IV. EXPERIMENTAL RESULTS

Block diagram

Copyright to IJAREEIE

www.ijareeie.com
RTL schematic

Technology schematic

Design summary
V. CONCLUSION

This brief has presented low-cost FIR filter designs by jointly considering the optimization of coefficient bit width and hardware resources in implementations. In this method a Booth multiplier is implemented. By using Booth multiplier to multiply the signed numbers also. Although most prior designs are based on the transposed form, gives information about the direct FIR structure with booth multiplier leads to the smallest area cost and power consumption.

REFERENCES