Design of efficient multiplierless FIR filters

D.L. Maskell

Abstract: An algorithm for reducing the hardware complexity of linear phase finite impulse response digital filters that minimise the adder depth in the multiplier block adders (MBAs) is presented. The algorithm starts by aggressively reducing both the coefficient wordlength and the number of non-zero bits in the filter coefficients. This reduces the number of adders (the adder depth) that are needed to construct the coefficient multiplier and results in an increased operating frequency. A modification to the representation of the filter coefficients such that the number of full adders (FAs) in our hardware implementation is proportional to the product of the input signal wordlength and the number of adders is proposed. That is, in general, the number of FAs is independent of the coefficient wordlength and the number of shifts between non-zero bits in the coefficient. Results show that the proposed technique achieves a 67 and 70% reduction in the number of MBAs and the number of multiplier block FAs, respectively. A software program has been implemented, which generates a Verilog HDL description of the digital filter. The proposed technique is not limited to filters with only a small number of taps and has been successfully applied to filters with up to 500 taps.

1 Introduction

Finite impulse response (FIR) digital filters are widely used in digital signal processing applications because of their stability, linear phase response and their simple regular structure. In today's consumer-driven embedded systems domain, constraints such as low power consumption and very high data throughput mean that general-purpose programmable filters are unsuitable. In these situations, the filter's constant coefficient multipliers are usually implemented in hardware using a sequence of shift and add operations. These structures are referred to as multiplierless implementations and present the filter designer with a number of conflicting design issues. The complexity of the design problem becomes evident when one considers the large design space and the need for optimising several incompatible and often competing objectives. These include area, power, and throughput as well as the objectives traditionally considered in multiplierless filter implementations, including the coefficient precision, the number of additions, the adder depth and so on.

Much of the research into efficient multiplierless filter implementations has been conducted in two independent sub-areas, namely discrete optimisation of the quantised filter coefficients [1–8]; and the reduction in the number of multiplier block adders (MBAs) [9–19]. Discrete coefficient optimisation techniques based on mixed integer linear programming using minimax or minimum normalised peak ripple magnitude have been proposed [4]; however, these techniques are extremely compute intensive and are restricted to filters with only a small number of taps. Other sub-optimal techniques have been proposed. These include random local search [5], tree search [2], simulated annealing [7] and genetic algorithm [8] methods.

Multiplier block reduction has attempted to minimise various cost functions such as adder depth, the number of adders and, more recently, multiplier block area. Most multiplier block minimisation techniques can be categorised as either graphical or subexpression elimination. Typical graphical techniques are discussed in [9–11]. Common subexpression (CSE) elimination techniques attempt to minimise the number of additions in the multiplier block by combining signed digit (SD) terms. These can be canonic SD (CSD) [12–16], minimal SD (MSD) [17] or all SD (ASD) [19]. The complexity of the adder implementations for various subexpression techniques was examined, with a significant reduction in the multiplier block full adder (FA) complexity when super-subexpressions (SSEs) were used [16].

Other authors have allowed for a relaxation of the original frequency specifications to achieve a reduction in the number of non-zero bits in the filter coefficients [20] or increased the filter order to achieve a reduction in the coefficient non-zero bits [20]. While there has been some attempt to combine both research areas, implementations that achieve a minimal hardware area satisfying a given frequency specification [21, 22] are only suitable for filters with a very small number of taps. In this paper, we present an integrated design technique which firstly attempts to aggressively reduce the coefficient wordsize and the number of non-zero bits in the filter coefficients while maintaining the original filter frequency specifications. Secondly, the minimum worst-case adder depth using simple horizontal CSE (HCSE) sharing is determined. Constrained by this minimum adder depth, the depth of the adder structure is relaxed to give a significantly reduced FA count and hence a reduced hardware complexity. Thirdly, the technique is applicable to filters with a much larger number of taps than existing techniques [21, 22].

2 Problem definition

This section describes the problem scenario and provides an analysis of issues relating to the implementation of FIR filters in hardware. As in [16] and [23], we introduce some of the terms used through this paper.

© The Institution of Engineering and Technology 2007
doi:10.1049/iet-cds:20060201
Paper first received 14th June 2006 and in final revised form 6th February 2007
The author is with the Centre for High Performance Embedded Systems (CHiPES), School of Computer Engineering, Nanyang Technology University, 639798, Singapore
E-mail: asdouglas@ntu.edu.sg

IET Circuits Devices Syst., 2007, 1, (2), pp. 175–180
1. **CSD format**: A number $b_0b_1b_2 \cdots b_{M-1}$ is in a CSD format if each $b_i$ is of the set $[0, 1, -1]$ and there are no two consecutive non-zero digits. In this paper, the coefficient wordsize is $M$ bits, the number of non-zero bits is $l$ and we represent $-1$ as $\bar{1}$.

2. **Multiplier block adders**: The MBAs are the adders used to calculate the multiplierless products of the input signal ($x$) and the filter coefficient ($h_k$) in the transposed direct form FIR filter structure. For symmetric FIR filters, the coefficients are symmetrical, thus only $[N/2]$ equivalent multipliers are needed, where $N$ is the number of filter taps.

3. **Structural adders**: The structural adders are the adders between the delay stages of the FIR filter structure. The number of structural adders is one less than the number of filter taps ($N$).

4. **Adder depth**: The adder depth (AD) is the critical path (in terms of addition operations) of the product implementation for any filter coefficient. An AD is related to the logic depth (LD), as defined in [14], as $LD = AD + 1$. A multiplication can have different ADs depending on the implementation structure. For example a linear- or tree-structured implementation will have different ADs. We define the minimum AD ($D_{min}$) as the smallest AD, which is able to implement all of a particular filter’s coefficients. That is, $D_{min}$ is dependent on the filter coefficient with the greatest number of non-zero bits and is defined as

$$D_{min} = \max\{\log_2 l_{\text{max}}\} \quad (1)$$

where $l_{\text{max}}$ is the maximum number of non-zero bits in any of the filter coefficients.

5. **Minimal FA implementations**: The number of FAs to implement a multiplication by a filter coefficient has been addressed in [16]. The concept of adder span is introduced, such that if a subexpression is formed, say $x_2$, where $x_2 = x + x \gg 2$ corresponding to [101], then $x_2$ has a span of $W + 2$, where $W$ is the wordlength of the input signal ($x$). Consider the following example [16]: Let $h_k = 0.1001010101_2$ and $x$ in this example is quantised to 8-bits. The output can be expressed as

$$y_j = x \gg 1 + x_2 \gg 4 + x_2 \gg 8 + x \gg 12 \quad (2)$$

where $x_2 = x + x \gg 2$ and we are only considering subexpressions [101] and [101].

The coefficient multiplier can be implemented using a linear adder structure with an adder depth of 4 or as a binary tree structure with an adder depth of 3. The number of FAs is determined based upon the spans of the operands as

$$N_{\text{linear}} = 14 + 18 + 20 = 52 \text{ FAs}$$

$$N_{\text{tree}} = 14 + 2 \times 20 = 54 \text{ FAs} \quad (3)$$

The number of FAs needed to compute a result is underestimated [16], because the possibility of an overflow is not considered. That is, the product $x_2 = x + x \gg 2$ corresponding to $x[101]$ has a span $W + 2 + 1$, not $W + 2$ as reported in [16].

We propose a different method for assigning FAs [24], which significantly reduces the hardware count compared to that presented in [16]. To illustrate this technique, consider a filter coefficient $h_k = 0.58203125$ (0.100101012), where $D_{\text{min}}$ has been previously calculated as having a depth of 3 (based on a different coefficient). Note that the product by $h_k$ could be implemented using a linear adder structure with a resulting adder depth of 3, or with a tree structure resulting in an adder depth of 2 as shown in Fig. 1a. As we have already calculated $D_{\text{min}}$ as 3, both of these implementations are satisfactory. Fig. 1a examines the necessary operations to perform the calculation $x^*h_k$, where $-1 \leq y < 1$. Here we choose $x$ as an 8-bit value ($W = 8$) such that $x = 0.9296875$ (0.11011112). The values in bold represent a necessary (hard wired) sign extension. The underlined section represents the actual addition process, and hence the number of FAs. Both solutions need three adders and produce the same result, $x^*h_k = 0.5411071780$ (0.1000101010000112). However, they use a different number of FAs. The linear implementation requires $3W + 1 = 27$ FAs, while the tree implementation requires 30 FAs. The implementation as in [16] requires 38 FAs (or 40 FAs if we include the sign extensions necessary to cater for any overflow). Our technique for implementing the filter coefficient represents a 32% saving in adder area compared to the technique in [16].

This very simple example using only positive SD numbers shows that it is possible to greatly reduce the number of FAs needed to implement a constant coefficient multiplier. When we consider the other CSE, [101], we can again produce a result using $W + 1$ FAs, as shown in Fig. 1b. For instance, $x[101]$, where $x = 0.9296875$ (0.11011112), gives $x^*h_k = x + (x + 2) = 0.348632813$ (1.1010011012). However, a problem arises when we try to generate $x[101]$, or any summation which has a negative sign associated with the right hand side (RHS) summand. For example, $x[101]$, where $x = 0.9296875$ (0.11011112), gives a result, $x^*h_k = x + 1 + x \ll 2 = 0.348632813$ (0.01011001012), as shown in Fig. 1c. This implementation requires $W + 1 + 2$ FAs.

To overcome this limitation, we carry to the left the (negative) sign of any addition involving a negative RHS summand. This procedure is illustrated in Fig. 2, and can result in the negative sign being carried all the way up to the structural adder associated with the coefficient (as shown in line 4 and line 6 of Fig. 2). This can easily be extended to a larger coefficient wordsize, and a larger number of non-zero bits, and ensures that the maximum number of FAs for a linear implementation of a constant coefficient is $(l - 1) * (W + 1)$. That is, the number of FAs is independent of the coefficient wordsize and the shift amount between non-zero digits. The number of FAs can be further reduced by subexpression sharing.

![Fig. 1 Number of full adders to calculate the multiplierless product $x^*h_k$](image)
3 Proposed method

Having developed a methodology that eliminates the dependency on the coefficient shift amount, we can now describe the problem to be solved as: Given a frequency/deviation specification, generate a filter implementation that attempts to minimise the latency (the AD) and the hardware area (the number of FAs).

One method of reducing the AD is to implement each coefficient \( (h_j) \) as a separate binary tree of adders. However, as seen earlier, this results in a greater number of adders and hence a larger area. Other approaches \([9, 11, 17]\) result in a smaller number of adders but at the expense of AD and an increased design time. An AD has a significant effect on the latency, and hence the filter throughput, which could be overcome to some extent by adding pipeline stages to the MBAs. However, while this would address the latency issues relating to an increased AD, the addition of pipeline latches would result in a significant increase in the filter area counteracting the initial reduction in the number of adders.

Our proposed algorithm is presented in Fig. 3, and is described below. It implements a number of techniques for reducing the adder depth and hence the filter latency and filter area.

3.1 Coefficient optimisation

As mentioned in Section 1, discrete coefficient optimisation techniques based on mixed integer linear programming are extremely compute intensive and are thus restricted to filters with a small to moderate number of taps. Thus, instead of attempting to find a global optimal solution, we attempt to find a much faster local solution that satisfies the original frequency specifications, while aggressively minimising both the number of non-zero bits and the coefficient wordlength. To achieve this we use a modification of the Hooke and Jeeves pattern search algorithm, which we have implemented in MATLAB. This modified algorithm looks for a solution based upon an initial coefficient word-size and an initial number of non-zero CSD bits. If a solution is not found, we incrementally increases the number of non-zero CSD bits followed by an increase in the coefficient wordlength until an acceptable solution is found. Aggressively minimising the number of non-zero bits results in a significant reduction in the AD, as AD is directly related to the number of bits in the coefficient.

To allow space for the optimiser to work with, the number of filter taps is increased by a small amount. This approach is commonly used in filter coefficient optimisation \([2, 6, 20]\). A simple experiment, based on a lowpass filter specification, was conducted to examine the number of extra taps that are appropriate in this situation. The filter in this case (filter 1 of Table 1) had a passband ripple of 0.1 dB, a stopband attenuation of −50 dB with passband and stopband cutoff frequencies of 0.1\( \pi \) and 0.15\( \pi \) respectively. A number of filters were designed using MATLAB’s Remez function. The minimum tap filter that satisfies this specification has 103 taps. Filters with a 3% (106), 5% (108), 8% (111) and 10% (113) increase in the number of taps were designed. These filters were analysed using our proposed algorithm and the number of MB FAs, structural FAs, and the total number of FAs were compared for each implementation. These results are shown in Table 2. An input signal (\( x \)) wordlength of 16 bits is assumed.

Here, it is can be seen that the filter with the 5% increase in the number of taps gives the least amount of hardware (total FAs). The reason that the filter with the 3% increase in the number of taps has more MB FAs and SB FAs is that after optimisation, the required wordlength and number of non-zero bits to meet the filter specifications is

<table>
<thead>
<tr>
<th>Table 1: Test filter specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Number of full adders to implement filter 1</th>
</tr>
</thead>
<tbody>
<tr>
<td># Taps</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>106</td>
</tr>
<tr>
<td>108</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>113</td>
</tr>
</tbody>
</table>
larger than the filters with a larger number of taps. However, as the number of taps increases, the number of structural adders begins to increase, quickly eliminating any benefit from the coefficient reduction. This effect has also been reported in [26], where it was observed that the MBAs were less significant, in terms of adder cost, than the structural adders. Adder cost estimates [26] showed that increasing the filter order by \( \sim 4\% \) (with a minimum at \( \sim 5\% \)) had little effect on the complexity. The observation was also made that this large range significantly increases the design effort needed to find the optimal solution, and that the designer may choose instead to select a single design from the region knowing it to be near-optimal. On the basis of these results and observations, a 5% increase in the number of taps was used in all subsequent experiments.

3.2 Horizontal CSE elimination

CSE elimination [13], based on the four most common 2-bit subexpressions, is performed on the filter coefficients. These subexpressions are \([101]\), \([101]\), \([1001]\) and \([1001]\), and are allocated LSB first. The rationale for using the four most common subexpressions rather than the two most common subexpressions as in [13] or higher order combinations, such as 3-bit or 4-bit horizontal SSEs [13, 14, 16], is based on the following observations. The choice of signals to route within the multiplier block is a trade off between a reduced multiplier block area and an increased routing area. In addition, the available silicon area has increased significantly since Hartley [13] made the observation that the use of subexpression sharing will lead to an increase in the signal routing cost, which could be alleviated by choosing only the two most common subexpressions. SSEs are not considered, as aggressively minimising the number of non-zero digits results in very few 3-bit or 4-bit common subexpressions occurring in the filter coefficients, making it uneconomical to route these rarely used signals through the multiplier block. For example, in filter 1 of Table 3, the use of 3-bit SSEs reduces the adder complexity from 33 to 26 (a reduction of 21%), but requires the routing of an additional 6 signals (an increase of 150%). Note that there are no 4-bit SSEs in this filter.

3.3 Increasing the AD up to \( D_{\text{min}} \)

The minimum AD (\( D_{\text{min}} \)) is calculated. It should be observed that only a small number of the filter’s coefficients are constrained by \( D_{\text{min}} \) because \( D_{\text{min}} \) is calculated based upon the coefficient value with the most non-zero bits. We now take the remaining bits in the coefficients after CSE elimination and implement them (where possible) as a linear array of adders with a depth up to \( D_{\text{min}} \). As shown in a previous section, a linear array of adders requires a smaller number of FAs. Fig. 4 shows the implementation of the filter coefficient, \( h = 0.001000100010001010 \). The coefficient wordlength is 18 bits and the input signal \( (x_m) \) wordsize is 16 bits. The structural adder is 35 bits due to the possibility of an overflow in the calculation of the intermediate results. The number of (MB FAs) is 55 + 17 for the CSE.

4 Experimental results

Firstly, just the subexpression component of the algorithm was used to determine the number of MBAs for a known reference filter design [14]. This filter is a 121-tap highpass FIR filter with a normalised passband frequency of \( 0.8\pi \), a normalised stopband frequency of \( 0.74\pi \), a passband ripple of 0.1 dB and a stopband attenuation of \( -80\) dB that is based on the optimised version of Lim and Parker [2]. The number of MBAs and the adder depth using the four most common subexpressions (CSE-4) are compared with other techniques from the literature [18], and are presented in Table 4. These results show that our CSE algorithm compares favourably with the other minimum AD algorithms. For comparison purposes, a SSE implementation (determined by exhaustive search) gives the least MBAs at the minimum adder depth.

Next, our proposed algorithm is applied to several lowpass FIR filters with fixed frequency and attenuation characteristics. The filter specifications are summarised in Table 1, where \( f_p \) is the normalised passband frequency, \( f_s \)

\[
\begin{array}{l}
\text{Table 3: MBA and MB FA for the example filters} \\
\begin{array}{|c|c|c|c|c|}
\hline
\text{Filter} & \text{WLen} & \# \text{Steps} & \text{Simple} & \text{HCSE} & \text{MB FAs} \\
\hline
1 & 13^* & 3 & 91 & 44 (54) & 1740 (1024) \\
12 & 2 & 67 & 33 (41) & 586 (731) \\
12 & 3 & 66 & 32 (38) & 564 (670) \\
2 & 15^* & 3 & 208 & 98 (130) & 4303 (2698) \\
14 & 3 & 165 & 73 (98) & 1295 (1713) \\
3 & 17^* & 3 & 521 & 248 (303) & 11632 (6799) \\
16 & 3 & 367 & 148 (201) & 2683 (3431) \\
\hline
\end{array}
\end{array}
\]

*rounded coefficient filters

\[
\begin{array}{l}
\text{Table 4: Number of multiplier block adders and AD to implement the filter of [18]} \\
\begin{array}{|c|c|c|}
\hline
\text{Adder depth} & \text{MB adders} & \text{AD to implement the filter of [18]} \\
\hline
\text{simple CSD} & 145 & 3 \\
\text{CSE-2 [13]} & 84 & 3 \\
\text{RAG-n [9]} & 52 & 6 \\
\text{Pasko [14]} & 59 & 4 \\
\text{NCSE [15]} & 68 & 3 \\
\text{CRA-2 [18]} & 62 & 3 \\
\text{MRAG [10]} & 52 & 5 \\
\text{CSE-4} & 66 & 3 \\
\text{SSE} & 57 & 3 \\
\hline
\end{array}
\end{array}
\]

Highpass filter, 121 taps, 16-bit wordlength (\( f_p = 0.8\pi; f_s = 0.74\pi; A_p = 0.1 \) dB; \( A_s = -80 \) dB)
is the normalised stopband frequency, $A_p$ is the passband ripple, $A_s$ is the stopband attenuation and $\# Taps$ is the number of filter taps with a 5% increase in the number of taps included. The figures in brackets represent the minimum number of taps to satisfy the given filter specifications. The infinite precision coefficients were generated using the Remez function in MATLAB.

The filter coefficients were quantised using two techniques for comparison purposes. Firstly, the coefficients were rounded to the smallest bit precision that still satisfies the original frequency specifications and, secondly, the coefficients were determined using our proposed algorithm. This second process involves optimisation of the CSD filter coefficients, HCSE and increasing the adder depth for coefficients less than the minimum adder depth ($D_{\text{max}}$). These results are presented in Table 3. The rounded coefficient filters correspond to the rows indicated by an * in the wordlength (WLen) column. The remaining rows in Table 3 correspond to filters designed using our proposed algorithm. The values in the HCSE column represent the MB adders using the four most common subexpressions, with the values in brackets representing the MB adders using the two most common subexpressions. In the MB FAs column, the values for the rows indicated by an * represent the number of FAs using the conventional method, while the values in brackets are the number of FAs using the method based upon the adder span [16] with the two most common subexpressions. The other rows in the MB FA column represent the number of full adders calculated using our proposed method. The values in brackets represent the number of FAs using the two most common subexpressions, while the others represent the four most common subexpressions. Again, an input signal ($x$) wordlength of 16 bits is assumed.

From Table 3 we see that there is a significant reduction in both the number of MB adders and the number of FAs using our proposed algorithm. There is an average of 67% reduction in the number of MB adders for our algorithm when compared to the simple conventional implementation. This can be compared to the 25–44% reduction in the number of MB adders presented in [16], the average reduction of 43% using SLRAGn with the minimum AD [23], and the average reduction of 61% (with an increase in the adder depth) in [17]. It should be noted that there are a number of algorithms that are able to reduce the number of adders at the expense of an increase in the AD [9, 17]. However, one of our initial design criteria was that the minimum AD not be exceeded and thus the comparison with [23] is the most appropriate.

When the number of FAs is compared, there is a 70% reduction in the number of FAs for our algorithm when compared to the simple conventional implementation. For implementations using the two most common subexpressions, there is an average 64% reduction in the number of FAs. These results can be compared to the average 40% saving using the adder span method [16] as calculated from Table 3, and the 25%–54% reduction in the number of FAs as presented in [16].

The filters implemented using the above algorithm were described in the Verilog hardware description language and were synthesised to FPGA using Xilinx’s Project Navigator V6.3. The target FPGA was a Spartan 3s1500fg320-5. The critical path delay for each of the filters is presented in Table 5. Again, as in Table 3, the rows indicated by an * represent the rounded coefficient filters. The critical path delays show that the filters designed using the algorithm proposed are faster by ~10%, or better. Our existing tools (which can be freely downloaded from: http://www.ntu.edu.sg/home/asdouglas/FIR.html) currently only target FPGAs, and as a result only generate carry propagate adder structures due to the existence of fast carry logic in modern FPGA structures negating the need for carry save adders. We are currently modifying our multiplier generator tool to allow for the generation of carry save adder structures to better target other hardware technologies [23, 25].

5 Conclusions

In this paper we have proposed an algorithm for reducing the hardware complexity of linear phase FIR digital filters without resorting to increasing the adder depth in the MBAs. In fact, we aggressively attempt to reduce the number of non-zero bits in the filter coefficients so that the AD can be reduced. We also proposed a modification to the representation of the filter coefficients such that the number of FAs in our hardware implementation is proportional to only the product of the signal wordlength and the number of adders. Results show that there is a 67 and 70% reduction in the number of MBAs and the number of MB FAs, respectively. These results are significantly better than other results presented in the literature.

6 Acknowledgment

The author would like to thank A.P. Vinod of the School of Computer Engineering, Nanyang Technological University, Singapore for the useful discussions related to this work.

7 References


Table 5: Critical path delay for the example filters

<table>
<thead>
<tr>
<th>Filter</th>
<th>Coefficient wordlength</th>
<th>Adder depth</th>
<th>Critical path delay, ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13*</td>
<td>3</td>
<td>13.949</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>10.565</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>12.575</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15*</td>
<td>3</td>
<td>13.909</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>12.752</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17*</td>
<td>3</td>
<td>14.508</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>13.182</td>
<td></td>
</tr>
</tbody>
</table>

*rounded coefficient filters