A New Hardware-Efficient Architecture for Programmable FIR Filters

Hwan-Rei Lee, Chein-Wei Jen, *Member, IEEE*, and Chi-Min Liu

Abstract—**Although much research has been done on efficient high-speed filter architectures, much of this work has focused on filters with fixed coefficients, such as Canonical Signed Digit coefficient filter architectures, multiplierless designs, or memory-based designs. In this paper, we focus on digit-serial, high-speed architectures with programmable coefficients. To achieve high performance goals, we consider both of algorithm level and architecture implementation level of FIR filters. In algorithm level, we reformulate the FIR formulation in bit-level and take the associative property of the addition in both the digit-serial multiplications and filter formulations. In architecture level, we considered issues to implement the reformulated results efficiently. The issues include addition implementation, data flow arrangements, and treatment of sign-extensions. Based on the above considerations, we can obtain a filter architecture with accumulation-free tap structure and properties of short latency, flexible pipelinability and high speed. Comparing the cost and performance with previous designs, we find that the proposed architecture reduces the hardware cost of a programmable FIR filter to only half that of previous designs without sacrificing performance.**

I. Introduction

INTENSIVE RESEARCH on digital signal processing (DSP) and advances in VLSI technologies have had a great impact on the application domains of electronics. Among the DSP applications, finite impulse response (FIR) filters are important building blocks. Recently, because of increasing demand for video signal processing and transmission, high-speed and high-order programmable FIR filters have frequently been applied for performing adaptive pulse shaping and signal equalization on the received data in real time [1]. Hence, an efficient VLSI architecture for high-speed programmable FIR filters is needed.

However, high-speed high-order programmable filters are difficult to be implemented efficiently because of the high implementation cost and the programmability requirements. In the literature, various kinds of filter architectures have been proposed for different applications. A programmable FIR filter allows us to modify the filter coefficients while the filter is operating. Hence, some low cost implementations that require special codings of filter coefficients, such as the canonical signed digit (CSD) representation [2], distributed arithmetics [3], and memory based approaches [4], are not suitable for programmable filters because coding of coefficients is difficult to accomplish in real time.

Manuscript received October 28, 1994; revised November 22, 1995. This work was supported by the National Science Council, Taiwan under Grant NSC 84-2215-E009-057. This paper was recommended by Associate Editor E. G. Friedman.

H.-R. Lee and C.-W. Jen are with the Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu 30050, Taiwan, R.O.C.

C.-M. Liu is with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan, R.O.C.

Publisher Item Identifier S 1057-7130(96)06485-3.

To implement programmable filters, one multiplier and one adder are needed for each tap. Hence, multiplier-and-accumulator (MAC) based architectures [5]–[7] are frequently adopted for programmable filters. However, the cost of multipliers is high and they are not suitable for high-order filters. There are also bit-level approaches such as the free-accumulation [8] and bit-plane [9] techniques. The former deconstructs the multiplications into bit-level additions and uses carry-save additions to speed up operations. However, because of the different weights of the addend, the addition wordlength grows fast. To reduce the required wordlength, bit-plane [9] approaches reorder the additions to solve the problem. However, they incur a long latency in accomplishing all the additions and they also spend high hardware cost on pipelining the additions to bit-level. Even though it is featured with high speed, however, long latency and high implementation cost make it unsuitable for adaptive or high filtering order applications.

In comparison, coding of input signals to reduce the multiplication complexity is attractive for programmable applications because the coder of each tap can be shared and no coding of coefficients is applied. Techniques for implementing the MAC of each tap by a Modified Booth multiplier has been presented in [6]. This approach reduces to the cost of multiplication to only that of a shared Modified Booth coder and accumulators in each tap. However, the cost of each tap is still high, because accumulators and latches are still needed in each tap to perform multiplications.

To further reduce the implementation cost of a programmable FIR filters, we should consider not only the implementation issues of multiplications and additions but also on the algorithm formulation issues. In this paper, we consider filter operations on both algorithm and architecture level to pursue a high-speed low-cost and low latency implementation. To achieve this goal, we should consider the issues to reduce the cost of computations of the filter formulation under the programmability requirements on algorithm level. Hence, instead of reducing computation costs by coding of filter coefficients at bit-level, we consider coding input signals at bit-level and reformulate the entire filtering operation to reduce the computation costs. Basing on the algorithm level reformulation, we have not only reduced the number of additions needed but also reduce the internal wordlength for the whole filtering operations based on a Modified Booth coding of input signals and the associative properties of additions. On the architecture level, we considered the issues to implement the filter architecture efficiently. The issues include the addition implementation, pipeline scheme, and sign-extension treatments. Based on the architecture considerations, we have the algorithm reformulated results implemented as a programmable, low-cost, accumulation-free tap structure that can be flexibly pipelined without incurring any longer latency. The cost of the filter is about half of the approach in [6].

The organization of this paper is as follows. In the next section, we discuss the Modified Booth multiplication algorithm, formulate the algorithm on bit-level and apply it to the formulation of FIR filters. In Section III, issues related to implementation of the architecture are discussed. In Section IV, the hardware cost of the proposed architecture and the architecture proposed in [6] are compared. Concluding remarks are given in the final section.

II. Algorithm Reformulation

Compared with merely finding an efficient way to implement multiplications, it is advantageous to consider the overall formulation of the algorithm for FIR filters, because both the additions in the filter formulation and the serial multiplications are associative. Hence the additions can be arranged efficiently to reduce the overall hardware cost.

In this section, we will first review and formulate the digit-serial Modified Booth multiplication scheme on bit level and then apply the formulated result to FIR filters with two possible formulations to show the advantages of the algorithm reformulation in Section II-2. One of the two formulations is the direct replacement of the multiplication-addition operations in FIR filters. The direct replacement approach has been adopted in [6] and [7]. The other is the proposed reformulated algorithm. A filter architecture based on the proposed reformulated algorithm will have an accumulation-free tap, shorter internal wordlength, and flexibly pipelinable architecture compared with the direct implementation approach.

2.1 The Modified Booth Algorithm

The Modified Booth algorithm [10], [11] is a digit-serial algorithm for performing 2's complement multiplications. For the multiplication of two numbers, taking c as the multiplicand and considering x as the multiplier at bit level, we can encode x according to the Modified Booth algorithm. Taking every two bits of x as a pair and another bit from the previous pair to form a triplet with the one bit overlapped and assuming x has w_x bits, we can represent the triplet $\boldsymbol{x_l}$ as

$$\boldsymbol{x_l} = \{x^{2l+1}, x^{2l}, x^{2l-1}\} \tag{1}$$

where $l = 0, 1, \cdots, w_x/2 - 1, x^i$ is the ith bit of $x, x^{-1} = 0$, and x^{2l-1} is the overlapped bit from the previous triplet. Then we can reformulate the 2's complement representation of x as follows:

$$\begin{aligned} x &= -x^{w_x-1} \cdot 2^{w_x-1} + \sum_{i=0}^{w_x-2} x^i \cdot 2^i \\ &= \sum_{l=0}^{w_x/2-1} (-2x^{2l+1} + x^{2l} + x^{2l-1}) \cdot 2^{2l} \end{aligned} \tag{2}$$

Based on the coding of the algorithm, the multiplication, $c \cdot x$, is expressed as

$$\begin{aligned} c \cdot x &= \sum_{l=0}^{w_x/2-1} (-2x^{2l+1} + x^{2l} + x^{2l-1}) \cdot c \cdot 2^{2l} \\ &= \sum_{l=0}^{w_x/2-1} B(\boldsymbol{x_l}, c) \cdot 2^{2l}, \end{aligned} \tag{3}$$

where w_x is the wordlength of x and the Modified Booth coding function $B(\cdot,\cdot)$ is:

$$B(\boldsymbol{x_l}, c) = \begin{cases} 0 & \text{if } \boldsymbol{x_l} = \{0,0,0\}, \{1,1,1\} \\ c & \text{if } \boldsymbol{x_l} = \{0,1,0\}, \{0,0,1\} \\ -c & \text{if } \boldsymbol{x_l} = \{1,1,0\}, \{1,0,1\} \\ 2c & \text{if } \boldsymbol{x_l} = \{0,1,1\} \\ -2c & \text{if } \boldsymbol{x_l} = \{1,0,0\}. \end{cases} \tag{4}$$

From (3), we need a $(w_x/2)$-operand adder to sum the $(w_x/2)$ terms in (3) for each multiplication. As for the Modified Booth coding function, a combinational logic block is needed to generate the selection of zero, negative, or double of the operand, c, to the adder.

Compared with multiplications without the coding scheme, the number of additions is reduced from (w_x) to $(w_x/2)$ at expense of extra Modified Booth codings. Moreover, no further processing of the sign bit of the operand x is needed. This corresponds to a more regular architecture or simpler control from an implementation point of view.

2.2 Application of the Algorithm Reformulation to FIR Filters

In this section, the Modified Booth multiplication formulated above is applied to FIR formulations. By taking advantage of associative property of the accumulations in both the FIR and the Modified Booth multiplications, we can greatly simplify the inner product computation needed for FIR filters.

Considering an N-tap FIR filter with the input sequence x_n, output sequence y_n, and impulse response c_i, we can express the basic FIR formulation as

$$y_n = \sum_{i=0}^{N-1} c_i \cdot x_{n-i}. \tag{5}$$

Replacing the multiplication of $c_i \cdot x_{n-i}$ by the Modified Booth multiplication, we have:

$$\begin{aligned} y_n &= \sum_{i=0}^{N-1} c_i \cdot x_{n-i} \\ &= \sum_{i=0}^{N-1} \left[\sum_{l=0}^{w_x/2-1} B(\boldsymbol{x_{n-i,l}}, c_i) \cdot 2^{2l} \right]. \end{aligned} \tag{6}$$

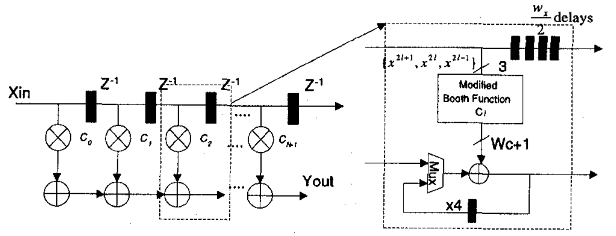


Fig. 1. Direct implementation of digit-serial multiplication and accumulation of N-tap FIR filters.

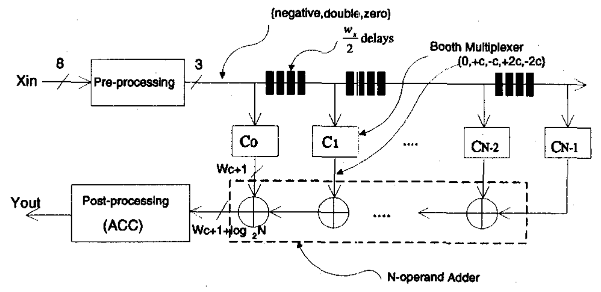


Fig. 3. A filter architecture that is free of accumulation in each tap.

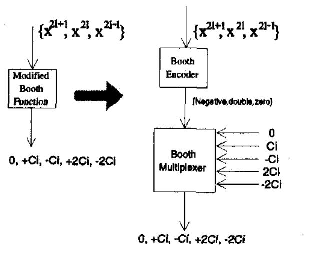


Fig. 2. Implementing the Modified Booth function in two steps.

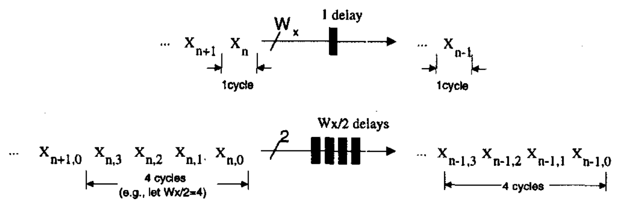


Fig. 4. Duplication of delay elements for digit serial sequences.

A direct implementation of (6) leads to the architecture shown in Fig. 1. In this figure, N digit-serial Modified Booth multipliers are used to implement the inner summation of (6) and each of the multiplications takes $w_x/2$ cycles to accumulate the results of $B(x_{n-i,l}, c_i)\cdot 2^{2l}, l = 0, 1, \cdots, w_x/2-1$. Then, for every $w_x/2$ cycles, the multiplication-accumulation results are passed to the next stage.

Let us consider the cost of the directly implemented architecture. N digit-serial multipliers and accumulators are needed, and thus N multiplexers are also needed to switch the computed data to the next stage every $w_x/2$ cycles. The internal wordlength needed to keep a full-precision output is $w_x + w_c + \log_2 N$ bits, where w_c is the wordlength of coefficients. Note the direct replacement approach has been adopted in [6] and [7] and the Modified Booth function $B(\cdot,\cdot)$ can be implemented in two steps (as shown in Fig. 2): the first step encodes each triplet of x to {*zero, negative, double*} and the second selects an appropriate value of $\{0, c_i, -c_i, 2c_i, -2c_i\}$ from the triplet encoded in the first step. Since the first step is common to all taps, it can be implemented in the preprocessing block shown in Fig. 1 and shared by all taps. In the preprocessing stage, the bit-parallel x_n input to the stage is converted to digit-serial bit triplets, encoded as {*zero, negative, double*}, and passed to the taps.

Compared with the direct implementation, if we rearrange the two summations in (6) according to the associative property of addition, (6) becomes:

$$y_n = \sum_{l=0}^{w_x/2-1} \left[\sum_{i=0}^{N-1} B(x_{n-i,l}, c_i)\right]\cdot 2^{2l}. \tag{7}$$

In (7), we need to perform two summations of the results of the Modified Booth function, $B(x_{n-i,l}, c_i)$. One is for index l and the other is for i. For the inner summation, index i, the N results are directly summed by an N operand adder. For the outer summation, index l, only one accumulator in the lower left block in Fig. 3, ACC, in Fig. 3 and $w_x/2$ cycles are needed to accumulate the results. Compared with the architecture in Fig. 1, the accumulations in each tap are moved to the accumulator in the ACC block so that no accumulations are needed in the taps.

In Fig. 3, the delays on the delay line to the taps are lumped every $w_x/2$ delay elements. Compared with a bit-parallel approach whose tapped delay duration is one clock cycle, the digit serial approach takes $w_x/2$ cycles to maintain one sample delay. This idea is depicted in Fig. 4. In the figure, a one-cycle delay for the bit-parallel approach should be duplicated $w_x/2$ times to ensure that the timing of the input matches that of the output.

Comparing the directly implemented tap structure in Fig. 1 with the reformulated structure in Fig. 3, we see that the reformulated one have at least the following advantages: First, there is no accumulation loop for the shift-and-add in each tap. Instead, the partial results from each Modified Booth multiplexer are summed with an N-operand addition and only one final accumulator is needed to perform the first summation $(\Sigma_{l=0}^{w_x/2-1})$ in (7).

Second, since every term, $B(x_{n-i,l}, c_i)$, of the second summation in (7) is of the same wordlength $(w_c + 1)$, the maximum possible wordlength for full precision output of the N terms is $(w_c+1+\log_2 N)$, which is shorter than $(w_c+w_x+\log_2 N)$, which should be adopted in the direct implemented taps. In the architecture implementation, a shorter maximum possible wordlength translates into a lower hardware cost if full precision output is implemented. On the other hand, if only a fixed output wordlength is required, a shorter maximum wordlength corresponds to a higher signal-to-noise ratio.

Third, since there are no accumulation loops in the taps, we do not need an extra control signal to reset these accumulators every $(w_x/2)$ clock cycles.

Fourth, since there is no accumulation in each tap, the pipelining of the implemented architecture is more flexible.

That is, we can adjust the number of pipeline stages according to the speed specification. This will be discussed in Section III.

III. Architecture Design

Architecture design involves finding techniques to implement the algorithm formulated above efficiently—that is, with high speed, low cost and short latency. In the previous section, we designed an accumulation-free tap structure for filters. In this section, we will consider how to implement the filter efficiently. The issues we must treat include the adder implementation, the data flow arrangement, and the treatment to sign extensions of additions. Of these three issues, data flow arrangement reduces the cost for pipelining registers and increases the speed of the architecture with the pipeline registers with incurring long latency. The addition implementation uses carry save adders to eliminate the carry ripple in each tap, and the treatment of sign extensions reduces the cost of the addition implementation.

3.1 Addition Implementation and Data Flow Arrangement

As can be seen from the discussion in the previous section, the implementation of the N-operand adder forms the critical path in our design. This critical path includes the series of adders needed to implement the N-operand addition and the carry ripple in each of the adders. Since the N-operand addition is a multi-operand addition and the carry save addition scheme can free the addition from carry ripple until the final result is needed [12], [11], the ripples of the additions can be left to the final stage of our architecture, where the final addition is performed by a vector merging adder (VMA) [12]. Since the VMA is activated only once every $(w_x/2)$ cycles and only one VMA is needed regardless of the order of the filter, the VMA can be implemented with either a high-cost, high-speed adder or just a low-cost adder just fast enough to complete the addition ripple in $w_x/2$ cycles.

Another long signal path is the series of $(N-1)$ CSA's used to implement the N-operand addition. The length of this path can be reduced by pipelining. Considering Fig. 5, by the associative property of addition, we can arrange the accumulation in either the same direction or opposite to the signal flow of x, as shown in Fig. 5(a) and (b).

The pipeline arrangement in the two cases leads to different results. In the first case in Fig. 5(a), we need to insert $3 + 2(w_c + 1 + \log_2 N)$ extra delay registers for every cut-line applied. The first term, 3, corresponds to the pipelining adder applied to the flow of encoded input signals, $\{negative,\ double,\ zero\}$, and the other term corresponds to the wordlength of both carry and sum signals for the CSA's. Obviously, for every cutset, one extra latency cycle is introduced. That is, the number of clock cycles between one input sample and its corresponding output sample is increased by one for every cut-line applied.

Considering the second case shown in Fig. 5(b), in which the accumulation and the input signal flow in opposite directions, we may find that pipelining the architecture shown in Fig. 5(b) is more efficient. According to the delay transfer rules in [13], we can move the same number of delay elements

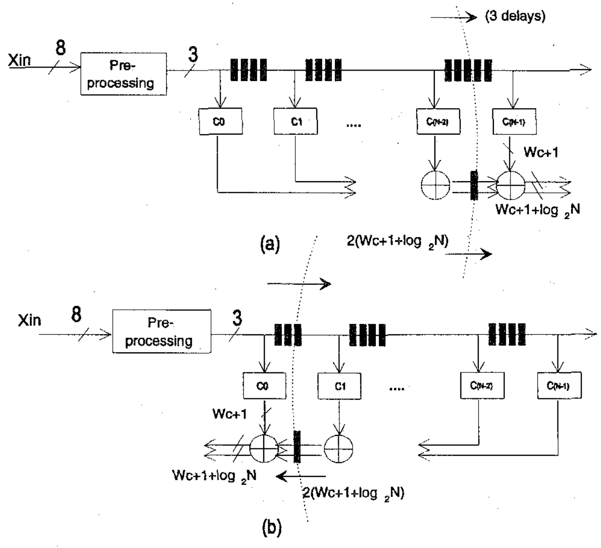


Fig. 5. The cut-set retiming for the signal flow arrangement of input and accumulation signals.

from all the inbound edges to the outbound edges of a cut-line without modifying the system's behavior. That is, the delay registers in the figure can be transferred from the signal path, $\{negative,\ double,\ zero\}$, to the CSA side. Thus, in Fig. 5(b), the extra number of registers introduced by the cutset retiming is equal to the difference between the number of registers on the CSA side and on signal path side, $(2(w_c+1+\log_2 N)-3)$. The second term, 3, corresponds to the delay transfer instead of the insertion as in the first case. Inspecting the resulting architecture, we find that the latency does not increase and fewer extra registers are needed. Thus, we adopt the contraflow scheme in our architecture.

3.2 Sign Extensions

To reduce the number of full adders used in our implementation, we can also consider the treatment of the sign extensions of each addition. Since the Modified Booth multiplier introduces a (w_c+1) bit number into the accumulation path in each tap and the wordlength of the accumulation is $(w_c+1+\log_2 N), \log_2 N$ b of sign extensions must be applied to the number before the addition. Thus, $(w_c+1+\log_2 N)$ full adders should be used in each tap. However, we can show that if a transformation similar to the idea used in Bough-Wooley multipliers [11, 14] is applied, the full adders for the $\log_2 N$ guard bits can be reduced to only half adders and no sign extensions are needed in each tap. This reduction of sign extensions also reduces the fanouts of the MSB (Most Significant Bit) of the Booth multiplexer outputs and thus helps to increase the operating speed.

To illustrate this idea, let us consider an addition of two 2's complement numbers, x_L and x_S, with different wordlengths, w_L and w_S, respectively, and assume that $w_L > w_S$. That is, x_L has a *longer* wordlength than x_S. To perform the addition of x_L and $x_S, (w_L - w_S)$ bits of sign extensions must be applied to x_S so that the wordlength of the two numbers are matched.

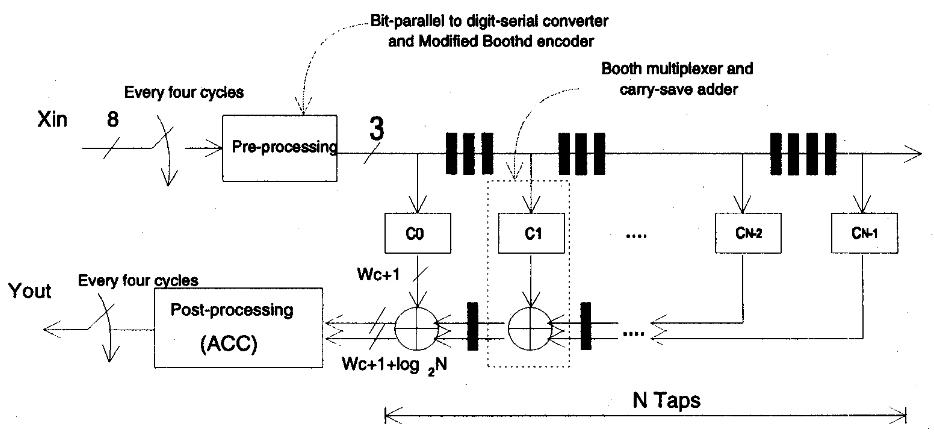


Fig. 6. The proposed architecture.

Considering the representation of x_S, where

$$\begin{aligned} x_S &= -x_S^{w_S-1} 2^{w_S-1} + \sum_{i=0}^{w_S-2} x_S^i \cdot 2^i \\ &= (x_S^{w_S-1}, x_S^{w_S-2}, \cdots, x_S^0) \end{aligned} \quad (8)$$

and applying sign extensions to x_S, we have

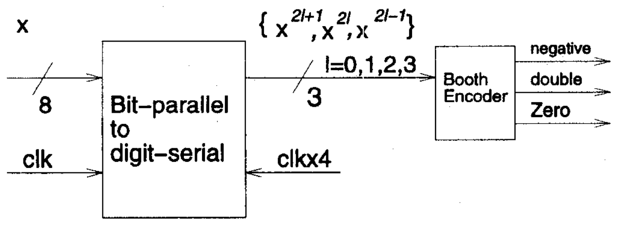
$$\begin{aligned} x_S &= -x_S^{w_S-1} 2^{w_L-1} + x_S^{w_S} \cdot \sum_{i=w_S-1}^{w_L-2} 2^i + \sum_{i=0}^{w_S-2} x_S^i 2^i \\ &= \overbrace{(\underbrace{x_S^{w_S-1}, \cdots, x_S^{w_S-1}}_{w_L-w_S\text{ b}}, x_S^{w_S-1}, x_S^{w_S-2}, \cdots, x_S^0)}^{w_L\text{ b}}. \end{aligned} \quad (9)$$

The first and second terms of (9) correspond to the sign extension bits of x_S. The terms could be a sequence of either 1's or 0's. To eliminate the sequence, we can add a constant 2^{w_S-1} to x_S to obtain

$$\begin{aligned} x_S + 2^{w_S-1} &= -x_S^{w_S-1} \cdot 2^{w_L} + \overline{x_S}^{w_S-1} \cdot 2^{w_S-1} \\ &\quad + \sum_{i=0}^{w_S-2} x_S^i \cdot 2^i \end{aligned} \quad (10)$$

$$= \overbrace{(x_S^{w_S-1}, 0, 0, \cdots, \underbrace{\overline{x_S}^{w_S-1}, x_S^{w_S-2}, \cdots, x_S^0}_{w_S\text{ b}})}^{(w_L+1)\text{ b}} \quad (11)$$

where $\overline{x_S}^{w_S-1}$ correspond to the inversion of $x_S^{w_x-1}$. Since x_S is sign-extended to a w_L-bit number, the first term in (10) can be ignored. This is because x_S is originally a w_S-bit number, and hence the addition of 2^{w_S-1} should always offset x_S to be positive. Thus, the (w_L+1)th bit of $(x_S + 2^{w_S-1})$ will always be zero. Thus, the addition of the two numbers can be implemented by only w_S full adders for the w_S nonzero bits in (11). Applying this idea to the FIR formulation in (7), we find that this transformation reduces the number of full adders in the CSA's of each tap. As for the compensation of the pre-added value in each addition, it can be compensated together



x
$\{x^{2l+1}, x^{2l}, x^{2l-1}\}$
l=0,1,2,3
8
Bit-parallel to digit-serial
3
clk
clkx4
Booth Encoder
negative
double
Zero

Fig. 7. The preprocessing stage.

by a constant value of

$$\sum_{l=0}^{w_x/2-1} (-N \cdot 2^{w_S-1}) \cdot 2^{2l}. \quad (12)$$

Since this compensation does not relate to the values of either the input sequence or the coefficients, it can be easily implemented by preloading the constant to the ACC block when a chip is fabricated.

3.3 The Proposed Architecture

Taking all the discussions above, we have the resulting architecture shown in Fig. 6. This figure shows the proposed architecture for a filter with input signal, x_{in}, of 8 b and it produces one output, y_{out}, every four cycles $(w_x/2 = 4)$.

Starting from the input of the architecture, we have the preprocessing block on the upper left on Fig. 6 to latch x_{in} and produce the Modified Booth coded result every clock cycle. The architecture of the preprocessing block is shown in Fig. 7. It basically contains a bit-parallel to digit-serial converter and part of a Modified Booth coder to produce the negative, double and zero signals for generating partial results of the tap coefficient in each tap. There are N taps on the right side of this figure.

The result of each tap, $B(x_{n-i,l}, c_i)$, is generated by a Booth multiplexer and accumulated along with the carry save adders. The carry save adders are simplified by the discussion of (8) to (12) so that the cost of each CSA is reduced from $w_c + \log_2 N - 1$ full adders to $(w_c + 1)$ full adders, $(\log_2 N - 1)$ half adders, and one XOR gate. The tap architecture is shown in Fig. 8.

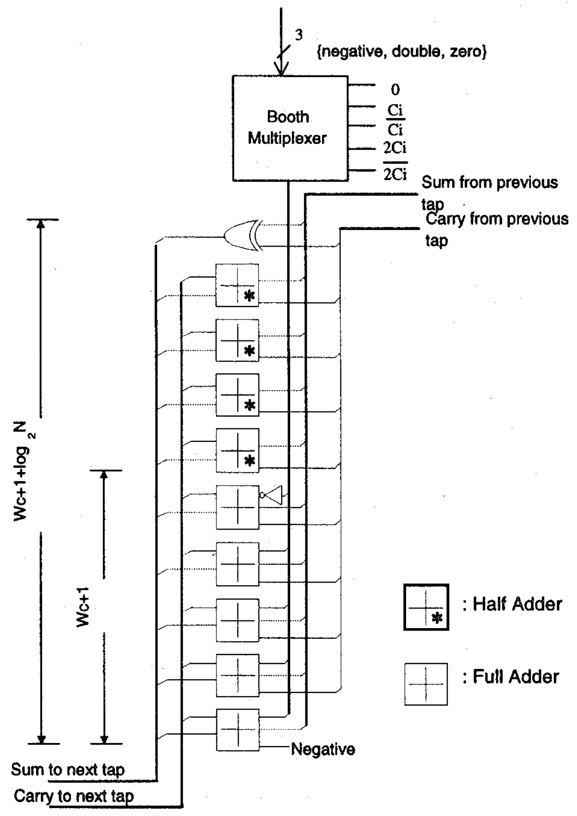


Fig. 8. The tap structure of the proposed architecture.

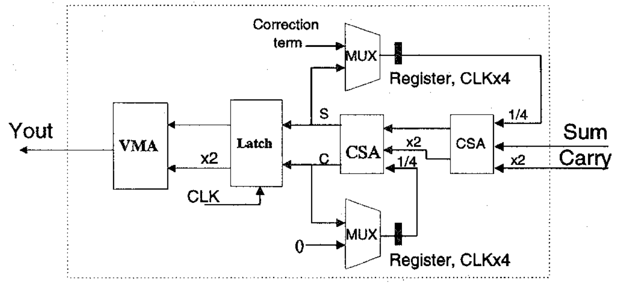


Fig. 9. The post-processing stage.

The results obtained from each tap are fed into the post-processing block. The post-processing block performs the function to accumulate the results and to compensate the terms for signed-extension treatments. The compensation term of (12) is preloaded to the post-processing block on the lower left of Fig. 6. The block produces one sample of y_{out} in terms of carry and sum from the CSA path and one final result is produced by VMA every four clock cycles. The architecture of this block is shown in Fig. 9. The post-processing stage uses two carry save adders (CSA's) to accumulate the carry and sum signals in every four cycles. During the last of the four cycles, the accumulated results are latched and passed to the VMA block for obtaining the final results. At the mean time, the multiplexers are switched to the correction term to put the term to the feedback registers for adding with the next four cycles.

The N taps on the right of Fig. 6 are pipelined with the contraflow arrangement of the input sequence and the accumulation. It can be seen that pipelining this architecture is more flexible than pipelining multiplier-based tap structures, because there are no loops of accumulation as in the direct replacement approach. Hence, if the speed requirement of our application is lower, fewer pipelining stages can be adopted. As shown in Fig. 8, the critical path of the fully pipelined structure is only the elapsed time of the Modified Booth multiplexer and one full adder. Thus, we can perform the cutset retiming every two or more taps to reduce the number of delay registers needed (and thus the hardware cost) according to the speed requirements of a particular application.

To show how the flexible pipelining applies to different applications, we have to verify the critical path and the number of reduced registers of the resulting pipelined architecture. Defining T_{FA}, T_{Mux} as the delay time of a full adder and that of a Booth multiplexer, we have the total elapse time of the critical path of a fully pipelined architecture as $T_{Mux} + T_{FA}$. On the other hand, if we perform cut-set on every k taps, we can inspect the critical path from Fig. 6 and verify that it becomes $kT_{FA} + T_{Mux}$. The number of pipeline registers are reduced from k sets to one for the k taps. Hence, adjusting the values of k, we will have architectures with different speed and cost. This is difficult for the direct replacement approach that contains loops in each tap.

IV. Cost Comparisons

In this section, we will compare the cost of two architectures: the first is an architecture obtained by the direct replacement of multipliers by digit-serial Modified Booth multipliers [6] and the second is the proposed architecture. For a fair comparison, the proposed architecture is fully pipelined to a carry-save adder level such that both architecture have the same elapse time of critical path.

For purposes of comparison, a number of components that are common to the two implementations are ignored, including the coefficient loading multiplexers, coefficient registers, preprocessing stage, and VMA. Table I shows the number of components used in both of the architectures. As shown in Table I, the proposed approach requires fewer full adders, multiplexers, and delay registers. First, because of the difference in addition wordlength, the proposed approach needs $N \times (w_x - 1)$ fewer full adders, and because of the treatment of sign extensions, the other $\log_2 N$ full adders are reduced to half adders. Second, since there are no multiplexers in the taps, the number of multiplexers is reduced to only those needed in the final accumulator block, ACC. Third, the number of delay registers is also reduced by the reduction in the wordlength and by the contraflow arrangement.

Let us now calculate the cost of filters of different orders. Table II shows the estimated gate counts for full adders, multiplexers, and delay registers for filters with input wordlength $w_x = 8$, coefficient wordlength $w_c = 8$ and full precision output wordlength for y_{out}. That is, no truncations are made in either architecture. Different filter orders, N, are listed. The total estimated gate count is the weighted sum of FA's

TABLE I
COMPARISON OF THE COST OF DIRECT MULTIPLIER REPLACEMENT AND THE PROPOSED ARCHITECTURE

	Full adders	Multiplexers	Delay registers
Direct Replacement [6]	$N(w_c + w_x + \log_2 N)$	$2N(w_c + w_x + \log_2 N)$	$N \times \frac{w_x}{2} \times 3$ $+2N(w_c + w_x + \log_2 N)$
New Approach	$N(w_c + \frac{\log_2 N}{2} + 1)$	$2 \times (w_c + w_x + \log_2 N)$	$N \times (\frac{w_x}{2} - 1) \times 3$ $+2N(w_c + \log_2 N + 1)$ $+2(w_c + w_x + \log_2 N)$

TABLE II
NUMBER OF THE ESTIMATED GATE COUNTS AND THE RATIO BETWEEN THE DIRECT REPLACEMENT (1) AND THE PROPOSED DESIGN (2)

N	FA1	FA2	MUX1	MUX2	Delay1	Delay2	total1	total2	ratio
4	72	40	144	36	192	160	1824	1148	1.58885
8	152	84	304	38	400	302	3824	2128	1.79699
16	320	176	640	40	832	600	8000	4176	1.91571
32	672	368	1344	42	1728	1226	16704	8464	1.97353
64	1408	768	2816	44	3584	2540	34816	17440	1.99633
128	2944	1600	5888	46	7424	5294	72448	36208	2.00088
256	6144	3328	12288	48	15360	11056	150528	75392	1.9966

FA: full adders. The cost of half adder is estimated as half of the cost of an FA
The cost of an FA is weighted by 6.
MUX: multiplexers. The cost of a MUX is weighted by 3.
Delay: delay registers. The cost of a Delay is weighted by 5.

TABLE III
SAVINGS RATIO FOR THE CASE WHERE $N = 64$

Techniques applied	FA×6	MUX×3	Delays×5	Total
Wordlength reduction	2688(7.7%)	0	4260(12.3%)	6948(20.0%)
Accumulation-free	0	8316(23.9%)	0	8316(23.9%)
Sign-extension	1152(3.3%)	0	0	1152(3.3%)
Data flow arrangement	0	0	960(2.8%)	960(2.8%)
Total Savings	3840(11.0%)	8316(23.9%)	5220(15.1%)	17376(50%)

(full adders), MUX's (multiplexers), and Delays according to a standard cell library [15]. The weights for FA's, MUX's and Delays are 6, 3, and 5, respectively. The ratios show that the cost of these components in our architecture is only half that in the direct replacement approach.

More precisely, taking the case of a filter of order $N = 64$ as an example, we can see from Table III how various techniques reduce the hardware cost. As shown in the table, that the greatest saving achieved by the reformulated FIR algorithm is due to the reduction in the required internal wordlength and the accumulation-free tap structure. Although the architecture techniques do not produce as significant a hardware savings as the algorithm reformulation, the techniques are still needed to implement the reformulated results efficiently. Fig. 4 shows a plot of the estimated gate counts for filters of various orders. The two curves correspond to the direct replacement approach and the proposed architecture, respectively. The small discontinuities on the curve correspond to increases in the wordlength for guard bits. That is, the guard bits for order $2^k + 1$ are 1 b greater than for order 2^k. The proposed architecture greatly reduces the hardware cost for the implementation of an FIR filter without sacrificing any performance.

V. CONCLUSION

In this paper, we have proposed a new programmable digital FIR filter architecture. In this architecture, we combined both the algorithm and architecture level considerations. The comparison results showed that the cost of the architecture is about half compared with the previous approach. *On the algorithm level*, we combined and reformulated the Modified Booth multiplication algorithm and the filter operation in bit-level. The reduction of internal wordlength and the accumulation-free tap structure are obtained through this level of considerations. *On the architecture level*, carry-save addition

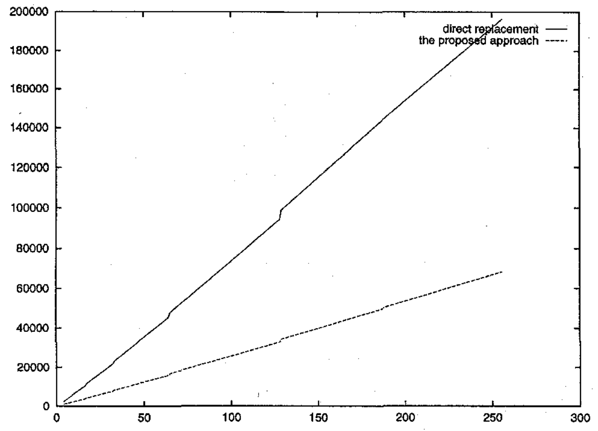


Fig. 10. Estimated gate counts for the direct replacement and the proposed approach with different filter orders.

scheme and pipelining through cut-set retiming results in efficient architecture design. As for the signed extension treatments, it further reduces the cost of carry save adders by removing the additions needed for sign-extension signals. The proposed architecture can also be used for other inner-product based DSP applications such as discrete cosine transform, discrete Fourier transforms (DFT), etc.

References

[1] S. Herman, *The Complete Ghost Canceler.* ICCE'93 Educational Session, June 1993.

[2] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst.,* vol. 36, pp. 1044–1047, July 1989.

[3] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," in *IEEE Asia-Pacific Conf. Circuits Syst.,* July 1989, pp. 4–18.

[4] H.-R. Lee, C.-W. Jen, and C.-M. Liu, "On the design automation of the memory-based VLSI architectures for FIR filters," *IEEE Trans. Consumer Electron.,* pp. 619–630, Aug. 1993.

[5] M. Sid-Ahmed, "A systolic realization for 2-D digital filters," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 37, pp. 560–565, Apr. 1989.

[6] B. Edwards, A. Corry, N. Weste, and C. Greenberg, "A single chip ghost canceller," *IEEE J. Solid-State Circuits,* vol. 28, pp. 379–383, Mar. 1993.

[7] B. Edwards, A. Corry, and C. Greenberg, "A single chip video ghost canceller," in *Proc. IEEE Custom Integrated Circuits Conf.,* May 1992, pp. 26.5.1–26.5.4.

[8] P. R. Cappello, "toward an FIR filter tissue," in *Int. Conf. Acoust., Speech, Signal Processing,* 1985, pp. 276–279.

[9] D. Reuver and H. Klar, "A configurable convolution chip with programmable coefficients," *IEEE J. Solid-State Circuits,* vol. 27, pp. 1121–1123, July 1992.

[10] L. P. Rubinfield, "A proof of the modified booth's algorithm for multiplication," *IEEE Trans. Comput.,* pp. 1014–1015, Oct. 1975.

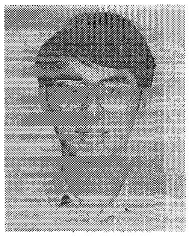
[11] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design.* New York: Wiley, 1979, ch. 4.2, p. 98.

[12] T. Noll, "Carry-save arithmetic for high-speed digital signal processing," *J. VLSI Signal Processing,* vol. 3, pp. 121–140, 1991.

[13] S. Y. Kung, *VLSI Array Processors.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

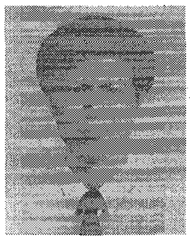
[14] O. Salomon, J.-M. Green, and H. Klar, "General algorithm for a simplified addition of 2's complement numbers in multipliers," in *Euro. Solid-State Circuit Conf.,* 1994, pp. 208–211.

[15] Texas Instruments, *TSC700 Series, 1.0-μ CMOS STANDARD CELLS,* 1992.



Hwan-Rei Lee was born in Taipei, Taiwan, R.O.C. in 1966. He received the B.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1989.

He is currently working on the Ph.D. degree in electronics engineering at National Chiao Tung University. His research interests include VLSI architecture design, parallel processing and computer architecture.



Chein-Wei Jen (S'78–M'84) received the B.S. degree from National Chiao Tung University in 1970, the M.S. degree from Stanford University, Stanford, CA, in 1977, and the Ph.D. degree from National Chiao Tung University in 1983.

He is currently with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, as a Professor. During 1985–1986, he was with the University of Southern California, Los Angeles, as a Visiting Researcher. His current research interests include VLSI design, digital signal processing, processor architecture, and design automation.

Dr. Jen is a member of Phi Tau Phi.

Chi-Min Liu received the B.S. degree in electrical engineering from Tatung Institute of Technology, Taiwan, R.O.C. in 1985, and the M.S. degree and Ph.D. degree in electronics from National Chiao Tung University, Hsinchu, Taiwan, in 1987 and 1991, respectively.

He is currently an Associate Professor with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan. His research interests include video/audio compression, speech recognition, radar processing, and application-specific VLSI architecture design.