Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2007, Article ID 92523, 14 pages
doi:10.1155/2007/92523

Research Article

A Hardware-Efficient Programmable FIR Processor Using Input-Data and Tap Folding

Oscal T.-C. Chen and Li-Hsun Chen

Department of Electrical Engineering, Signal and Media Laboratories, National Chung Cheng University, Chia-Yi 621, Taiwan

Received 4 March 2006; Revised 1 August 2006; Accepted 24 November 2006

Recommended by Bernhard Wess

Advances in nanoelectronic fabrication have enabled integrated circuits to operate at a high frequency. The finite impulse response (FIR) filter needs only to meet real-time demand. Accordingly, increasing the FIR architecture's folding number can compensate the high-frequency operation and reduce the hardware complexity, while continuing to allow applications to operate in real time. In this work, the folding scheme with integrating input-data and tap folding is proposed to develop a hardware-efficient programmable FIR architecture. With the use of the radix-4 Booth algorithm, the 2-bit input subdata approach replaces the conventional 3-bit input subdata approach to reduce the number of latches required to store input subdata in the proposed FIR architecture. Additionally, the tree accumulation approach with simplified carry-in bit processing is developed to minimize the hardware complexity of the accumulation path. With folding in input data and taps, and reduction in hardware complexity of the input subdata latches and accumulation path, the proposed FIR architecture is demonstrated to have a low hardware complexity. By using the TSMC 0.18 μm CMOS technology, the proposed FIR processor with 10-bit input data and filter coefficient enables a 128-tap FIR filter to be performed, which takes an area of 0.45 mm^2, and yields a throughput rate of 20 M samples per second at 200 MHz. As compared to the conventional FIR processors, the proposed programmable FIR processor not only meets the throughput-rate demand but also has the lowest area occupied per tap.

1. INTRODUCTION

Finite impulse response (FIR) filter is regarded as one of the major operations in digital signal processing; specifically, the high-tap-number programmable FIR filter is commonly applied in ghost cancellation and channel equalization. The main operation of an FIR filter is convolution, which can be performed using addition and multiplication. The high computational complexity of such an operation makes the use of special hardware more suitable for enhancing the computational performance. This special hardware used to realize a high-tap-number programmable FIR filter is costly. Thus minimizing the hardware cost of this special hardware is an important issue.

With the regular computation of an architecture, a folding scheme that utilizes the same and small hardware component to repeatedly complete a set of computation is frequently used to reduce the hardware complexity of such architecture [1, 2]. Generally, the folding schemes of an FIR architecture can be classified into input-data folding, coefficient folding, and tap folding [3–11]. Additionally, while advances in nanoelectronic fabrication have enabled integrated circuits to operate at a high frequency, the throughput-rate demand of an FIR filter does not change significantly. Due to such phenomenon, the folding technique must be further improved to design a hardware-efficient FIR architecture. Figure 1 presents the relationship between the computational performance, hardware complexity, and circuit speed on different hardware platforms in realizing a high-tap-number FIR filter. With only one or few multipliers/adders, the programmable processors cannot be applied to realize a high-tap-number FIR filter in real time. On the other hand, the conventional FIR architectures using the application specific integrated circuit (ASIC) approach would have the fixed folding numbers to do so; but with the increase in circuit speed, the conventional architectures are only able to slightly decrease hardware complexities by reducing their pipelined latches. Therefore, in the advanced fabrications, conventional FIR architectures with fixed folding numbers cannot be used to realize a hardware-efficient FIR filter. Instead,

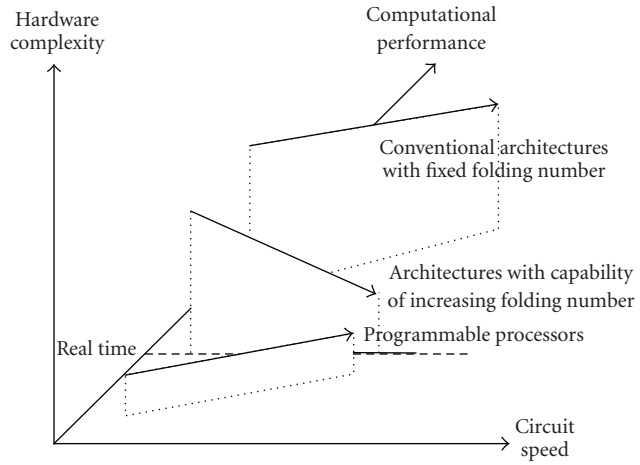

Figure 1: The relationship between computational performance, hardware complexity, and circuit speed on different hardware platforms to realize a high-tap-number FIR filter.

an FIR architecture that can increase its folding number would cost-effectively meet the real-time performance demand. With the use of high-speed circuitry, the folding number of such architecture is increased accordingly to effectively decrease the computation units required. In overall, this FIR architecture can fill the gap between fabrication migration and hardware platform development, in the design of an architecture that meets real-time demand with hardware efficiency.

In the FIR architecture design, the circuit required for a multiplication operation poses a major concern because it takes a hefty part of hardware complexity. The multiplication operation includes partial-product generation, partial-product shifting, and partial-product summation. Of which, partial-product shifting can be realized with hardwire so no additional hardware complexity is dedicated here. To avoid computation at large word lengths, the folding scheme can be applied to add the partial products at the same precision index from multiple multiplication operations, shift the added results, and then perform summation of these shifted results to complete an FIR filter operation. Based on the above arrangement, an FIR architecture employing input-data and tap folding is proposed in this work. With input-data folding, each input datum is partitioned into multiple input subdata with short word lengths. In each clock cycle, multiplication operations are performed on input subdata at the same precision index and the coefficients correlated to these subdata. Results are then added, and the shifting and accumulation operations of the multiplications are performed on the summed results accordingly to derive at an output datum. With the shifting operation performed after the tap summation, it would not incur an increase in the word length of the intermediate data thus saves the hardware cost of adders in the tap summation. However, with the use of only input-data folding, the architecture's folding number is limited by the input-data word length and cannot increase along with the use of high-speed circuitry. The proposed architecture then takes it further by integrating tap folding to partition an FIR filter into multiple sections, and completes each section chronologically. The folding number of the proposed architecture using the input-data folding and tap-folding schemes is the product of the folding numbers from input-data folding and tap folding. An increase in the folding number of the tap-folding scheme would also increase the folding number of the proposed FIR architecture to accommodate the use of high-speed circuit in effectively reducing the hardware complexity. In comparison to the conventional architectures under the same folding number, the proposed architecture clearly demonstrates a lower hardware complexity.

Based on the radix-4 Booth algorithm, two approaches to reduce the hardware complexity of the FIR architecture are proposed—one is a 2-bit input subdata approach and the other is a tree accumulation approach with simplified carry-in bit processing. In the 2-bit input subdata approach, other than the input subdata currently in-use, the Booth decoder could also rely on the prior input subdata and control signal to perform Booth decoding. Such flexibility would allow the proposed FIR architecture to reduce the latch amount required to store these input subdata. As for the tree accumulation approach, a full adder is fully utilized to perform the addition operations. The proposed FIR architecture can omit the use of half adders, and lives up to its appeal for a design with low hardware complexity. In this work, the cell library of the TSMC 0.18 μm CMOS technology is used to implement the proposed FIR processor equipped with 10-bit input data and coefficients to realize 128 taps. Other than satisfying the throughput-rate requirement, the proposed FIR processor is demonstrated to have the least hardware area per tap than the conventional ones.

2. CONVENTIONAL BOOTH-ALGORITHM FIR ARCHITECTURES USING FOLDING SCHEMES

The operation of an FIR filter can be written as

$$Y_n = \sum_{i=0}^{N-1} C_i \times X_{n-i}, \tag{1}$$

where X, C, and Y represent the input data, filter coefficients, and output data, respectively, and N is the number of taps. The Booth algorithm is typically used to implement the multiplication operations of a programmable FIR filter and thus effectively reduce computational time and hardware complexity [12, 13]. Comparing radix-2, radix-4, radix-8, and radix-16 Booth algorithms in terms of both computational performance and hardware complexity reveals that the radix-4 Booth algorithm strongly outperforms in terms of hardware efficiency [14]. Therefore, the radix-4 Booth algorithm was applied in the proposed FIR architecture.

The radix-4 Booth algorithm incorporates the multiplier X_{n-i} and the multiplicand C_i with word lengths of W and L, respectively. Each input datum X_{n-i} is partitioned into many 3-bit groups, each of which has one bit that overlaps with the previous group, which can be written as

$$X_{n-i,l} = \{x_{n-i}^{2l+1}, x_{n-i}^{2l}, x_{n-i}^{2l-1}\}, \tag{2}$$

where l is an integer between 0 and $(W/2)-1$; x_{n-i}^{j} is the jth digit of X_{n-i}, and x_{n-i}^{-1} is zero. x_{n-i}^{2l-1} overlaps the preceding group $X_{n-i,l-1}$. The 2's complement representation of X_{n-i} can be

$$\begin{aligned} X_{n-i} &= -x_{n-i}^{W-1} \times 2^{W-1} + \sum_{j=0}^{W-2} x_{n-i}^{j} \times 2^{j} \\ &= \sum_{l=0}^{(W/2)-1} \left(-2x_{n-i}^{2l+1} + x_{n-i}^{2l} + x_{n-i}^{2l-1}\right) \times 2^{2l} \end{aligned} \tag{3}$$

C_i is multiplied by X_{n-i}, and (3) is modified to

$$\begin{aligned} C_i \times X_{n-i} &= \sum_{l=0}^{(W/2)-1} \left(-2x_{n-i}^{2l+1} + x_{n-i}^{2l} + x_{n-i}^{2l-1}\right) \times C_i \times 2^{2l} \\ &= \sum_{l=0}^{(W/2)-1} B\left(X_{n-i,l}, C_i\right) \times 2^{2l}, \end{aligned} \tag{4}$$

where $B(X_{n-i,l}, C_i)$ is the output of Booth decoding that can take five values, 0, $\pm C_i$ and $\pm 2C_i$, according to $X_{n-i,l}$.

According to (1), an FIR architecture can fold itself based on input data, coefficients, and taps. First, in the input-data folding scheme, with the radix-4 Booth algorithm being used to perform the multiplication operations, each W-bit input datum is partitioned into $(W/2)$ 3-bit input subdata that then undergo Booth decoding in order. From (1) and (4), the operation of an FIR filter can be modified as

$$\begin{aligned} Y_n &= \sum_{i=0}^{N-1} \left[\sum_{l=0}^{(W/2)-1} B\left(X_{n-i,l}, C_i\right) \times 2^{2l} \right] \\ &= \sum_{l=0}^{(W/2)-1} \left[\sum_{i=0}^{N-1} B\left(X_{n-i,l}, C_i\right) \right] \times 2^{2l}. \end{aligned} \tag{5}$$

Like the input-data folding scheme, the coefficient-folding scheme can be employed to partition each L-bit coefficient into $(L/2)$ 3-bit sub-coefficients, and then Booth decoding is performed in a sequence. Equation (1) can be modified as

$$\begin{aligned} Y_n &= \sum_{i=0}^{N-1} \left[\sum_{l=0}^{(L/2)-1} B\left(C_{i,l}, X_{n-i}\right) \times 2^{2l} \right] \\ &= \sum_{l=0}^{(L/2)-1} \left[\sum_{i=0}^{N-1} B\left(C_{i,l}, X_{n-i}\right) \right] \times 2^{2l}, \end{aligned} \tag{6}$$

where $C_{i,l}$ is the lth 3-bit sub-coefficient of the coefficient, C_i, and $B(C_{i,l}, X_{n-i})$ can be one of the five values, 0, $\pm X_{n-i}$, and $\pm 2X_{n-i}$. In the tap-folding scheme, an FIR filter is partitioned into f parts to complete the operations accordingly. Such a scheme can be applied to modify the operation of an FIR filter from (1) as follows:

$$\begin{aligned} Y_n &= \sum_{i=0}^{(N/f)-1} \left[\sum_{k=0}^{f-1} X_{n-(if+k)} \times C_{if+k} \right] \\ &= \sum_{k=0}^{f-1} \left[\sum_{i=0}^{(N/f)-1} X_{n-(if+k)} \times C_{if+k} \right]. \end{aligned} \tag{7}$$

Equations (5), (6), and (7) reveal that the FIR architectures equipped with input-data folding, coefficient folding, and tap folding would result in folding numbers of $W/2$, $L/2$, and f, respectively.

The three folding schemes based on (5), (6), and (7) are applied in the design of the two FIR architectures that are commonly used, the direct form and the transposed direct form, to derive the six FIR architectures shown in Figure 2. Among them, the preprocessing units of architectures in Figures 2(a), 2(b), 2(c), and 2(d) can partition input data or coefficients into many 3-bit input subdata or 3-bit sub-coefficients, and perform predecoding on these input subdata or sub-coefficients to reduce the hardware complexities of Booth decoders [3–5, 11]. Input (sub)-data latches and (sub)-coefficient latches are used to store input (sub)-data and (sub)-coefficients, respectively. N Booth decoders are applied to perform Booth decoding, with the results being added in the accumulation path. Pipelined latches are then used to reduce the delay and to arrange the data flow in accumulation computation. Lastly, the post-processing unit performs summation and shifting on results from the accumulation path to realize the computation of (5) and (6). As for the architectures shown in Figures 2(e) and 2(f), N/f multipliers are assigned to perform the multiplication operations. Each multiplier is equipped with $W/2$ or $L/2$ Booth decoders to generate partial products. Partial products from N/f multipliers are summed together in the accumulation path. Finally, the results from the accumulation path are carried on to the post-processing unit to perform the summation operation, thus satisfies the computation in (7) [6–8].

An FIR architecture with the transposed direct form is able to use the pipelining in the accumulation path to reduce the number of input (sub)-data latches. But, for the transposed direct-form architectures using coefficient folding and tap folding, as shown in Figures 2(d) and 2(f), the operation frequencies of input data paths are lower than those of pipelined latches in the corresponding accumulation paths. Hence, the accumulation path has to use more pipelined latches to store the computation results from its adders, in order to generate the correct output of an FIR filter. Due to this fact, the two architectures in Figures 2(d) and 2(f) cannot achieve low hardware complexities, and thereby are not explored further.

To take a closer look at the architectures in Figures 2(a), 2(b), 2(c), and 2(e), the features of functional units of these four architectures are listed in Table 1. Under the same folding number, $W/2 = L/2 = f$, these four architectures all have the same amount of Booth decoders. However, with the pre-processing unit capable of performing predecoding on subdata and sub-coefficients to reduce the hardware complexity of the Booth decoders, hardware complexities of the Booth decoders in architectures shown in Figures 2(a), 2(b), and 2(c) are lower than that in Figure 2(e). Moreover, the partial-product shifting operations of Figures 2(a), 2(b), and 2(c) are processed in the post-processing unit, so their accumulation paths also have lower hardware complexities than the accumulation path in Figure 2(e). Furthermore, with the use of multiplexers to select input data and coefficients, the

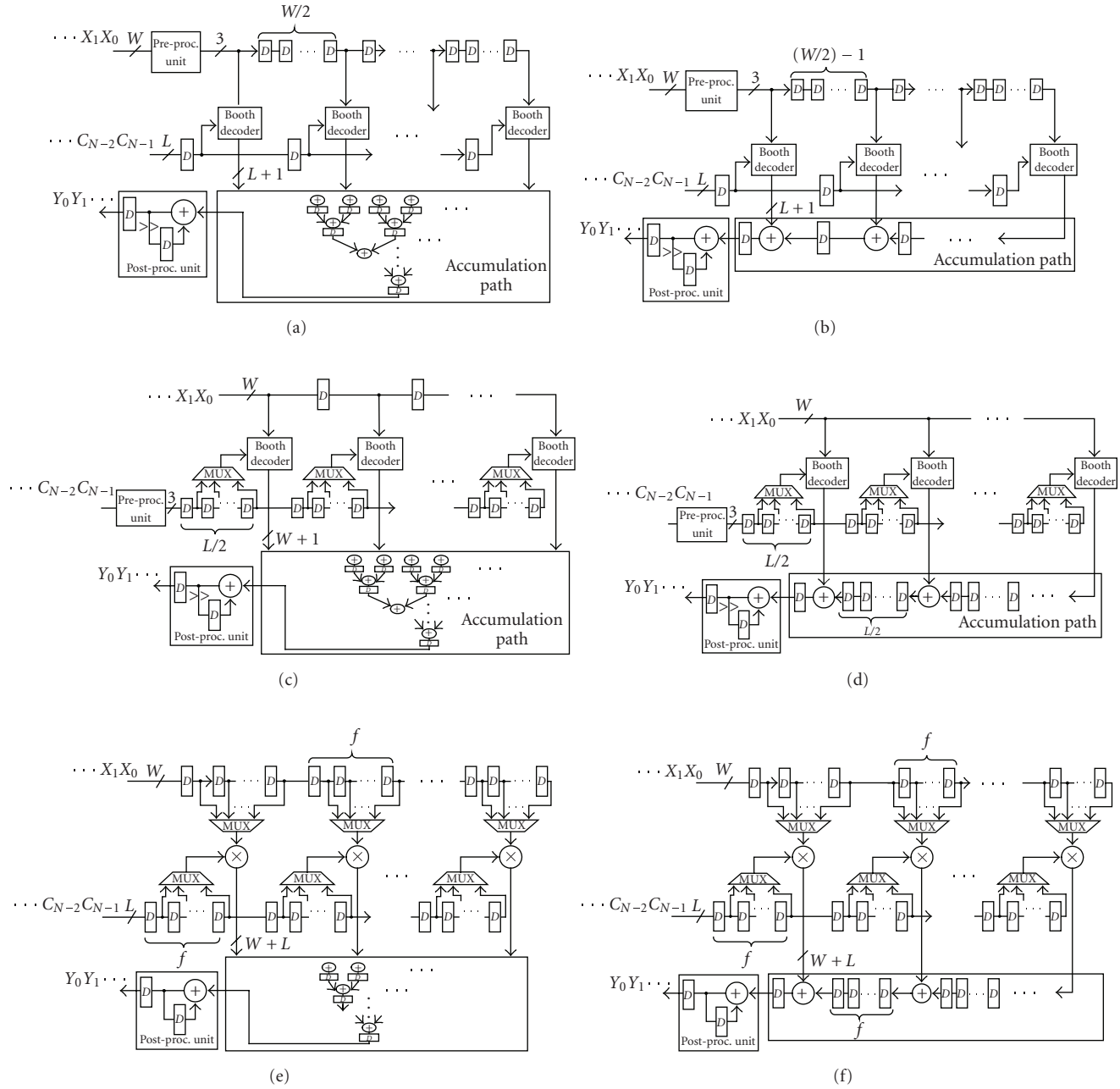

Figure 2: Six conventional FIR architectures. (a) Direct form using the input-data folding scheme. (b) Transposed direct form using the input-data folding scheme. (c) Direct form using the coefficient folding scheme. (d) Transposed direct form using the coefficient folding scheme. (e) Direct form using the tap-folding scheme. (f) Transposed direct form using the tap-folding scheme.

architecture in Figure 2(e) incurs a higher hardware complexity than the other three architectures. As illustrated in Table 1, when W equals L, architectures in Figures 2(a) and 2(c) have the same number of latches to store input (sub-)data and (sub-)coefficients. They both also have Booth decoders and accumulation paths with the same hardware complexities. However, with the architecture in Figure 2(c) requiring multiplexers to select the sub-coefficients, its hardware complexity would be slightly higher than the architecture in Figure 2(a). In comparing the architectures in Figures 2(a) and 2(b), the architecture in Figure 2(b) has fewer input subdata latches than those of Figure 2(a). But for the architecture in Figure 2(b), the linear accumulation structure causes the word lengths of the addition results to increase rapidly and thus raises the hardware complexities of the adders and latches in the accumulation path. Consequently, the hardware complexity of the architecture in Figure 2(a) is lower than that in Figure 2(b).

Comparing the other four architectures in Figures 2(a), 2(b), 2(c), and 2(e), under the same folding number, the

Table 1: Features of functional units of the architectures in Figures 2(a), 2(b), 2(c), and 2(e).

Features		Architectures			
		Figure 2(a)	Figure 2(b)	Figure 2(c)	Figure 2(e)
Hardware complexity	Preproc. unit	1	1	1	0
	Input (sub-)data latches	$N(W/2)$ of 3-bit latches	$N((W/2)-1)$ of 3-bit latches	N of W-bit latches	N of W-bit latches
	Input (sub-)data multiplexers	0	0	0	(N/f) of W-bit f-to-1 MUXes
	(Sub-)coeff. latches	N of L-bit latches	N of L-bit latches	$N(L/2)$ of 3-bit latches	N of L-bit latches
	(Sub-)coeff. multiplexers	0	0	N of 3-bit $(L/2)$-to-1 MUXes	(N/f) of L-bit f-to-1 MUXes
	Booth decoders	N of Booth decoders	N of Booth decoders	N of Booth decoders	$(N/f)\times(W/2)$ or $(N/f)\times(L/2)$ of Booth decoders
	Acc. path	Performing tree summation on N of $(L+1)$-bit partial products, and including $(N/2)$ of $(L+1)$-bit adders, $(N/4)$ of $(L+2)$-bit adders ... $(N/2^i)$ of $(L+i)$-bit adders ... 1 of $(L+\log_2 N)$-bit adder & $(N/2)$ of $(L+2)$-bit latches, $(N/4)$ of $(L+3)$-bit latches ... $(N/2^i)$ of $(L+i+1)$-bit latches ... 1 of $(L+\log_2 N+1)$-bit latch	Performing linear summation on N of $(L+1)$-bit partial products, and including 1 of $(L+1)$-bit adder, 2 of $(L+2)$-bit adders ... 2^{i-1} of $(L+i)$-bit adders ... $N/2$ of $(L+\log_2 N)$-bit adders & 1 of $(L+2)$-bit latch, 2 of $(L+3)$-bit latches ... 2^{i-1} of $(L+i+1)$-bit latches ... $N/2$ of $(L+\log_2 N+1)$-bit latches	Performing tree summation on N of $(W+1)$-bit partial products, and including $(N/2)$ of $(W+1)$-bit adders, $(N/4)$ of $(W+2)$-bit adders ... $(N/2^i)$ of $(W+i)$-bit adders ... 1 of $(W+\log_2 N)$-bit adder & $(N/2)$ of $(W+2)$-bit latches, $(N/4)$ of $(W+3)$-bit latches ... $(N/2^i)$ of $(W+i+1)$-bit latches ... 1 of $(W+\log_2 N+1)$-bit latch	Performing tree summation on $(N/f)\times(W/2)$ of $(L+1)$-bit partial products, each of which is shifted right by $2l$ bits $(l=0,1,2,\ldots,$ or $(W/2)-1)$ or $(N/f)\times(W/2)$ of $(W+1)$-bit partial products, each of which is shifted right by $2l$ bits $(l=0,1,2,\ldots,$ or $(L/2)-1)$
	Post-proc. unit	$(L+W+\log_2 N)$-bit adder and two $(L+W+\log_2 N)$-bit latches	$(L+W+\log_2 N)$-bit adder and two $(L+W+\log_2 N)$-bit latches	$(L+W+\log_2 N)$-bit adder and two $(L+W+\log_2 N)$-bit latches	$(L+W+\log_2 N)$-bit adder and two $(L+W+\log_2 N)$-bit latches
Folding number		$W/2$	$W/2$	$L/2$	f
Capability of increasing the folding number		No	No	No	Yes
Techniques to reduce hardware complexity at the use of high-speed circuitry		Reducing pipelined latches of the accumulation path	Reducing pipelined latches of the accumulation path	Reducing pipelined latches of the accumulation path	(1) Reducing pipelined latches of the accumulation path. (2) Increasing the folding number

architecture in Figure 2(a) displays the lowest hardware complexity but its folding number is limited by the input-data word length. When the high-speed circuitry is employed in this architecture, the only way to lower hardware complexity is to reduce the pipelined latches in the accumulation path. In contrast, the architecture in Figure 2(e) can increase its folding number to reduce the numbers of Booth decoders and adders, thus to effectively lower the hardware complexity. However, with the partial-product shifting operation performed prior to the accumulation path, the architecture in Figure 2(e) would have adders and pipelined latches with higher word lengths than those found in the accumulation paths of the architectures in Figures 2(a), 2(b), and 2(c). Hence, the integrated folding scheme combining input-data folding and tap folding is proposed in this work. Such integrated folding scheme can take advantages of the architectures in Figures 2(a) and 2(e) to have the accumulation path with a low hardware complexity and to have a capability of increasing the folding number to reduce hardware complexity.

3. PROPOSED FIR ARCHITECTURE

By using input-data folding and tap folding, the FIR filter computation in (1) can be modified as

$$\begin{aligned} Y_n &= \sum_{l=0}^{(W/2)-1} \left[\sum_{i=0}^{(N/f)-1} \sum_{k=0}^{f-1} B\left(X_{n-(if+k),l}, C_{if+k}\right) \right] \times 2^{2l} \\ &= \sum_{l=0}^{(W/2)-1} \sum_{k=0}^{f-1} \left[\sum_{i=0}^{(N/f)-1} B\left(X_{n-(if+k),l}, C_{if+k}\right) \right] \times 2^{2l}, \end{aligned} \tag{8}$$

where f is the folding number of tap folding and $W/2$ is the folding number of input-data folding. $\sum_{i=0}^{(N/f)-1} B(X_{n-(if+k),l}, C_{if+k})$ is computed using N/f Booth decoders, and an accumulation path sums the outputs from the Booth decoders. $\sum_{l=0}^{(W/2)-1} \sum_{k=0}^{f-1}$ and $\times 2^{2l}$ are sequentially computed in the post-processing unit. According to (8), this integrated folding scheme can design an FIR architecture with a high folding number by increasing the folding number of tap folding. Moreover, unlike the conventional tap folding, its partial-product shifting operation is processed in the post-processing unit to reduce hardware complexity in the accumulation path. Based on (8), the proposed FIR architecture is presented in Figure 3. While the input-data and tap-folding schemes are employed in the proposed FIR architecture, the 2-bit input subdata approach and tree accumulation approach with simplified carry-in-bit processing are developed to further reduce the hardware complexity. The following subsections describe these two approaches.

3.1. 2-bit input subdata approach

According to (2), the least significant bit of each original 3-bit input subdatum is either zero or the most significant bit of the previous input subdatum [12, 13]. Consequently, 2-bit input subdata rather than 3-bit input subdata can be used to reduce the number of latches on the input data path. As shown in Figure 4, the preprocessing unit comprises an input latch, a multiplexer, and a 1-bit XOR gate. The input latch stores input data. The multiplexer that is addressed by the control unit selects a correct sequence of 3-bit input subdata. Meanwhile, the 1-bit XOR gate is used to predecode the 3-bit input subdata to generate new 2-bit input subdata that can slightly reduce the hardware complexities of Booth decoders.

Figure 3 shows that 2-bit input subdata generated by the preprocessing unit are pipelined to input subdata latches. Through multiplexers selecting data from input subdata and coefficients, each Booth decoder can obtain the appropriate input subdata and coefficient for Booth decoding. In the radix-4 Booth algorithm, possible results, $\pm j \times C_i$, from the Booth decoders are generated, where j is an integer between zero and two. However, in the 2-bit input subdata approach, a 2-bit input subdatum from the input subdata latches cannot represent five choices. The Booth decoder must use one bit from the neighboring input subdata latch ($b_{l-1,1}$) as well as two bits from its corresponding input subdata latches ($b_{l,1}$ and $b_{l,0}$), as shown in Figure 5. According to (2), when l in (8) equals zero, this one extra bit ($b_{l-1,1}$) must be set as zero. To realize the computation of (8), a control signal is used to control an AND gate so that $b_{l-1,1}$ can be reset to zero at every $f \times (W/2)$ clock cycles and be held at zero for f clock cycles. Accordingly, $b_{l,1}$, $b_{l,0}$, and $b_{l-1,1}$ with this control signal are employed to generate a partial product and a carry-in bit, which represent the output of 0, C_i, $-C_i$, $2C_i$, or $-2C_i$. In particular, an inverter is applied to invert the sign bit of the partial product, so when the outputs generated by the Booth decoders are summed in the accumulation path, the sign extension operation can be omitted and the hardware complexity of the accumulation path is reduced accordingly [5]. Although the proposed Booth decoder is little more complex than the conventional Booth decoder [11], such a design would allow 2-bit input subdata latches to be used instead of conventional 3-bit input subdata latches in the input data path.

3.2. Tree accumulation approach

In the FIR architecture, each Booth decoder generates a partial product and a carry-in bit. The accumulation path sums all of the partial products and carry-in bits. These summed results are then inputted to the post-processing unit to yield the final result. The carry-save addition technique is applied to minimize the carry propagation delay and increase the computational efficiency of the accumulation path. Its fundamental functions include full adders and half adders. The full adder processes three input bits at the same precision index and then generates two output bits at different precision indexes, whereas the half adder processes only a pair of input bits at the same precision index, producing two output bits at different precision indexes. The half adder cannot be used to reduce the bit number because the number of input bits is equal to that of output bits. Therefore, sufficient use of full adders and reduced use of half adders would further decrease the hardware complexity of the accumulation path.

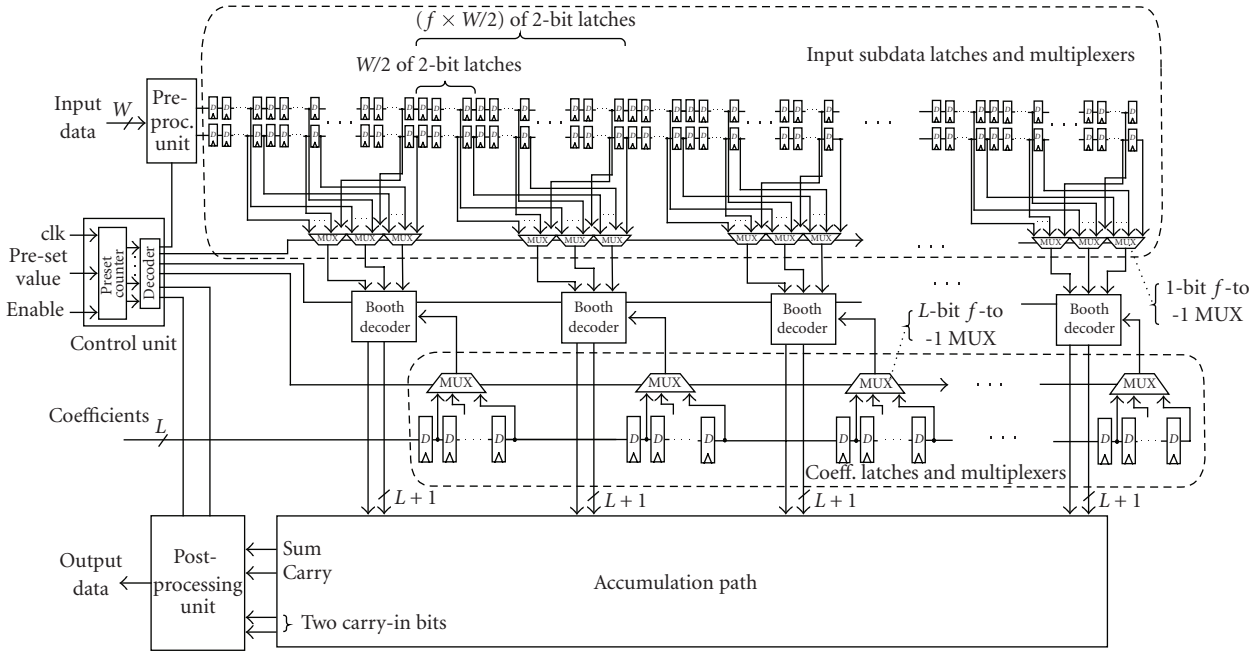

Figure 3: The proposed FIR architecture with input-data and tap folding.

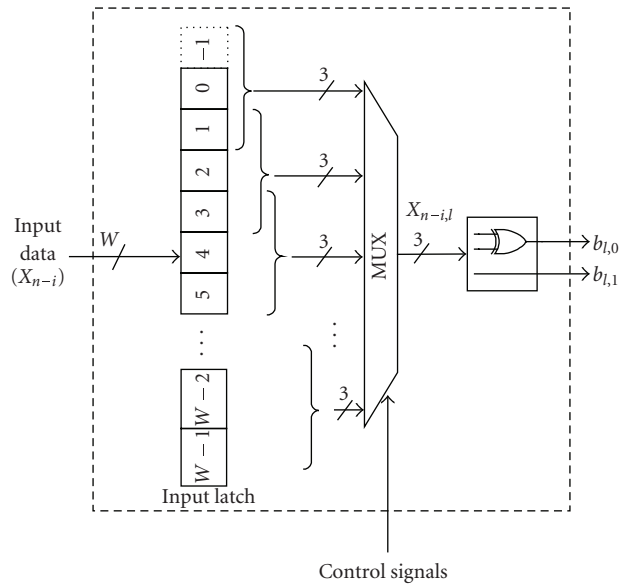

Figure 4: Preprocessing unit.

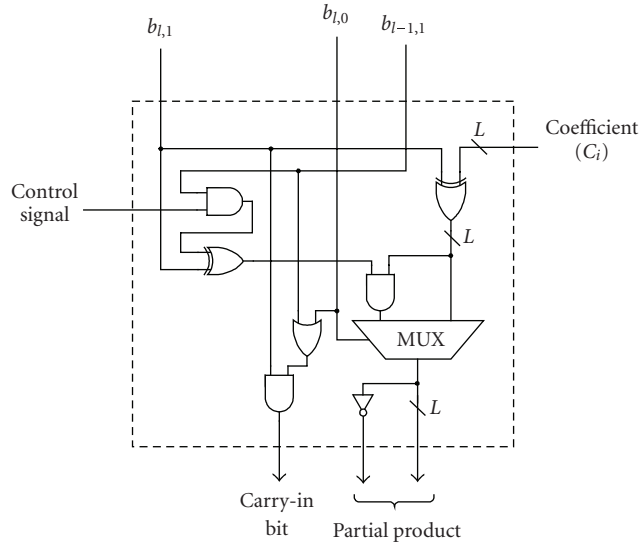

Figure 5: Booth decoder.

The conventional tree accumulation is divided into three parts to perform the additions in the accumulation path—the addition of the partial products, the addition of the carry-in bits, and the addition of the outputs of the two parts. The proposed tree accumulation approach hides the summation of the carry-in bits as part of the partial-product summation in the accumulation path, and also as part of the intermediate result summation in the post-processing unit. Eight 4-bit partial products and carry-in bits are used as an example in Figure 6, to demonstrate the proposed and conventional tree accumulation approaches using carry-save adders. Figure 6(a) depicts the conventional tree accumulation in which partial products and carry-in bits are summed individually, increasing the number of half adders required. Moreover, the summed partial products must be added to the summed carry-in bits in additional processing time. Herein, the conventional tree accumulation requires 28 full adders and five half adders. Figure 6(b) presents the proposed tree

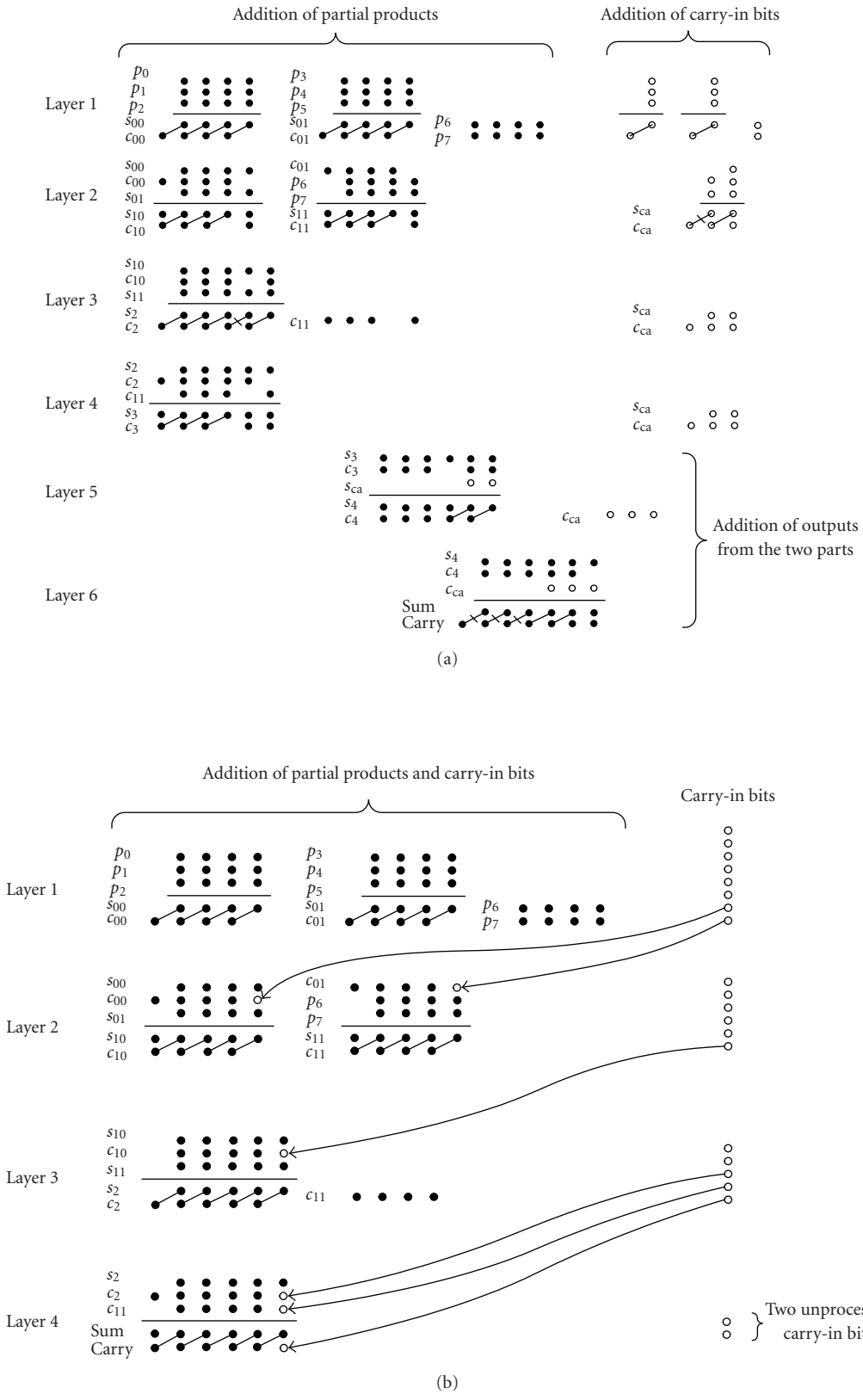

Figure 6: Operations of proposed and conventional tree accumulations. (a) Conventional tree accumulation. (b) Proposed tree accumulation.

accumulation in which the summation of the partial products and the carry-in bits are performed together. The proposed approach effectively exploits full adders to perform the addition of partial products and carry-in bits, and omits the use of half adders. Hence, only 26 full adders are required in the proposed tree accumulation.

An accumulation path can be partitioned into many pipelined stages to improve computational performance. When each pipelined stage needs the delay of one or two carry-save adders, 89 or 38 1-bit latches are required in the proposed tree accumulation, and 115 or 52 1-bit latches are required in the conventional tree accumulation. Thus, the proposed tree accumulation also has fewer latches than the conventional one. Also, as shown in Figure 6(b), a carry-in bit is regarded as the least significant bit of the carry value in each layer and is added with the other sum or carry value. But as Figure 6(b) points out too, the proposed accumulation only yields six carry values, which implies that it can only process the summation of eight partial products and six carry-in bits. The outputs of sum and carry and the two unprocessed carry-in bits would be moved to the post-processing unit to perform addition.

In the post-processing unit, carry and sum values generated from the accumulation path and two unprocessed carry-in bits are accumulated and shifted. Figure 7 shows the proposed post-processing unit. Two $(L+1+\log_2(N/f))$-bit carry-save adders are employed to perform sequential accumulation, and two $(L + W + \log_2 N)$-bit 2-to-1 multiplexers are applied in shifting. Notably, two $(L + W + \log_2 N)$-bit 2-to-1 multiplexers are used to select a zero value and a correction term in the first clock cycle. Adding the correction term is for compensating the omission of the sign extension operation from the accumulation path [3–5]. Additionally, the least significant bits of the two carry values generated by the carry-save adders in the post-processing unit are zero, so the unprocessed two carry-in bits can be considered to be the least significant bits of these two carry values, and their addition is performed in the two carry-save adders of the post-processing unit. Finally, the vector merge adder (VMA) is used to sum the carry and sum values to derive at a final result.

4. ANALYSES AND COMPARISONS OF PROPOSED AND CONVENTIONAL FIR ARCHITECTURES

In this section, the cell library of the TSMC 0.18 μm CMOS technology is applied to derive at the number of transistors required for each functional unit [15], and to use such numbers in the analyses and comparisons of hardware complexities between the proposed and conventional FIR architectures. First, three types of the FIR architectures employing input-data and tap folding, types I, II, and III, are defined to analyze the effectiveness of the proposed 2-bit input subdata approach and tree accumulation approach in reducing hardware complexity. All these three architectures have the same folding numbers, with the folding numbers of input-data folding and tap folding being $W/2$ and 2, respectively. The type-I FIR architecture uses both the proposed 2-bit input subdata approach and tree accumulation approach to lower its hardware complexity, while the type-II one only uses the 2-bit input subdata approach and the type-III one only adopts the proposed tree accumulation approach. The numbers of transistors required for these three architectures are shown in Figure 8.

In comparing the type-I and type-II architectures, the type-I architecture would require less transistors than the type-II one because the type-I architecture can simplify the processing of $N/2$ carry-in bits to reduce its hardware complexity. With an increase in the number of tap number (N), the number of carry-in bits that can be simplified in processing is also increased to allow the type-I architecture to further reduce the number of transistors required. Additionally, the difference in the numbers of transistors required between the type-I and type-II architectures is not significant with the changes found in input-data word length (W) or coefficient word length (L). In comparison to the type-III architecture, the type-I architecture can take the 2-bit input subdata approach to reduce $N \times (W/2)$ 1-bit latches, $(3 \times N \times (W/2))$-$(2 \times N \times (W/2))$. The Booth decoder in the type-I architecture demands slightly more logic gates than that of the type-III architecture, but it still requires less transistors than the type-III. With an increase in the input-data word length (W) and tap number (N), the type-I architecture can demonstrate that it requires less transistors than the type-III one.

As stated in Section 2, under the same folding number, the architecture in Figure 2(a) would have lower hardware complexity than the other architectures in Figure 2. But in comparison to the fixed folding number of the architecture in Figure 2(a), the folding number of the architecture in Figure 2(e) can be increased to lower hardware complexity. Due to this understanding, we compare the hardware complexities of the proposed architecture and the architectures in Figures 2(a) and 2(e). To fairly compare them, these three architectures must operate at the same throughput rate. According to [13], the throughput rate can be represented by n_s/T_{clk} where T_{clk} is a period of a clock cycle and n_s is the number of outputs produced in a clock cycle. Additionally, T_{clk} is equivalent to the critical delay. As for a folded FIR architecture, the folding number is the number of clock cycles required to generate an output. Accordingly, the throughput rate can be denoted as follows [13]:

$$\text{Throughput rate} = \frac{1}{\text{critical delay} \times \text{folding number}}. \tag{9}$$

With T_{FA} representing the delay of the full adder, and the throughput rate fixed at $1/(2 \times T_{\text{FA}} \times W)$, the numbers of transistors required for the above-mentioned three architectures in comparison are presented in Figure 9 where the word length of input data is equal to that of coefficients. In the proposed architecture, the folding numbers for input-data and tap folding are $W/2$ and 2, respectively; hence the folding number of the proposed architecture is W. According to (9), the proposed architecture has a critical delay of $2T_{\text{FA}}$, which indicates that the delay for each pipelined stage should be less than or equal to $2T_{\text{FA}}$. Looking at the

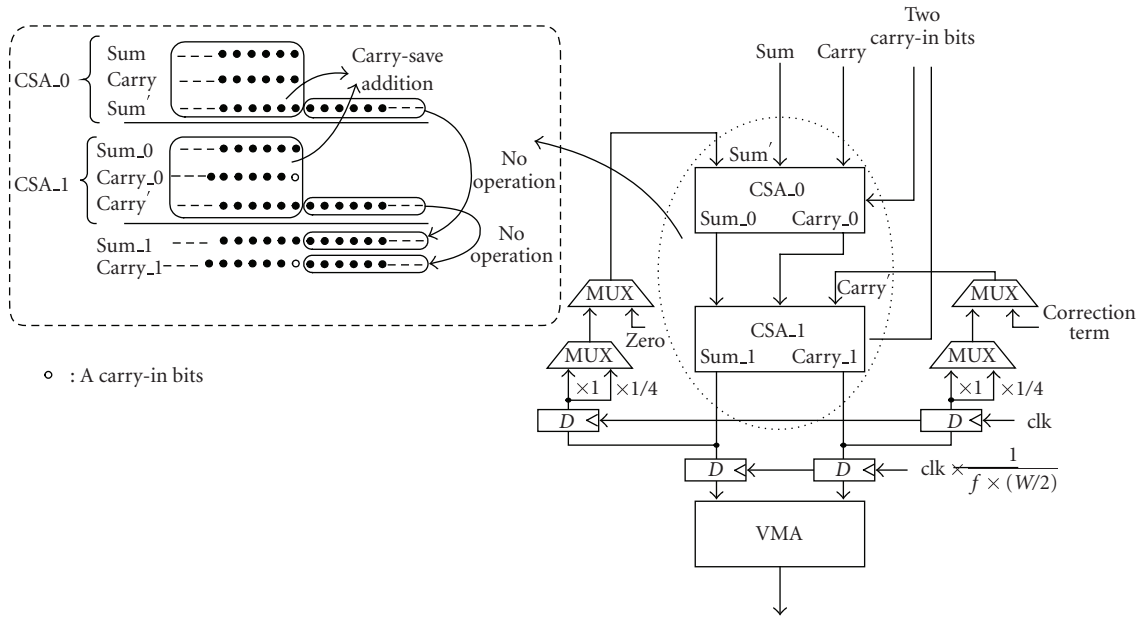

Figure 7: Post-processing unit.

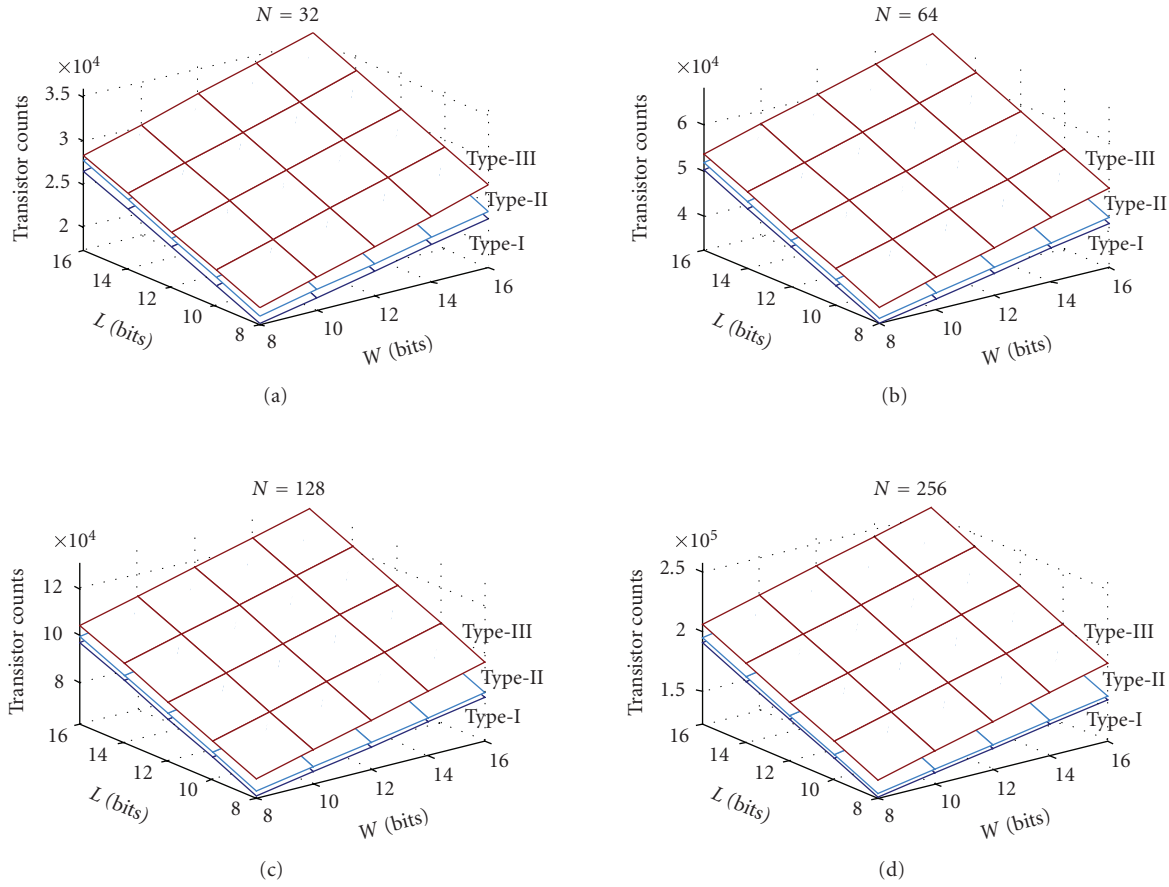

Figure 8: Transistor counts of three types of the FIR architectures using input-data and tap folding.

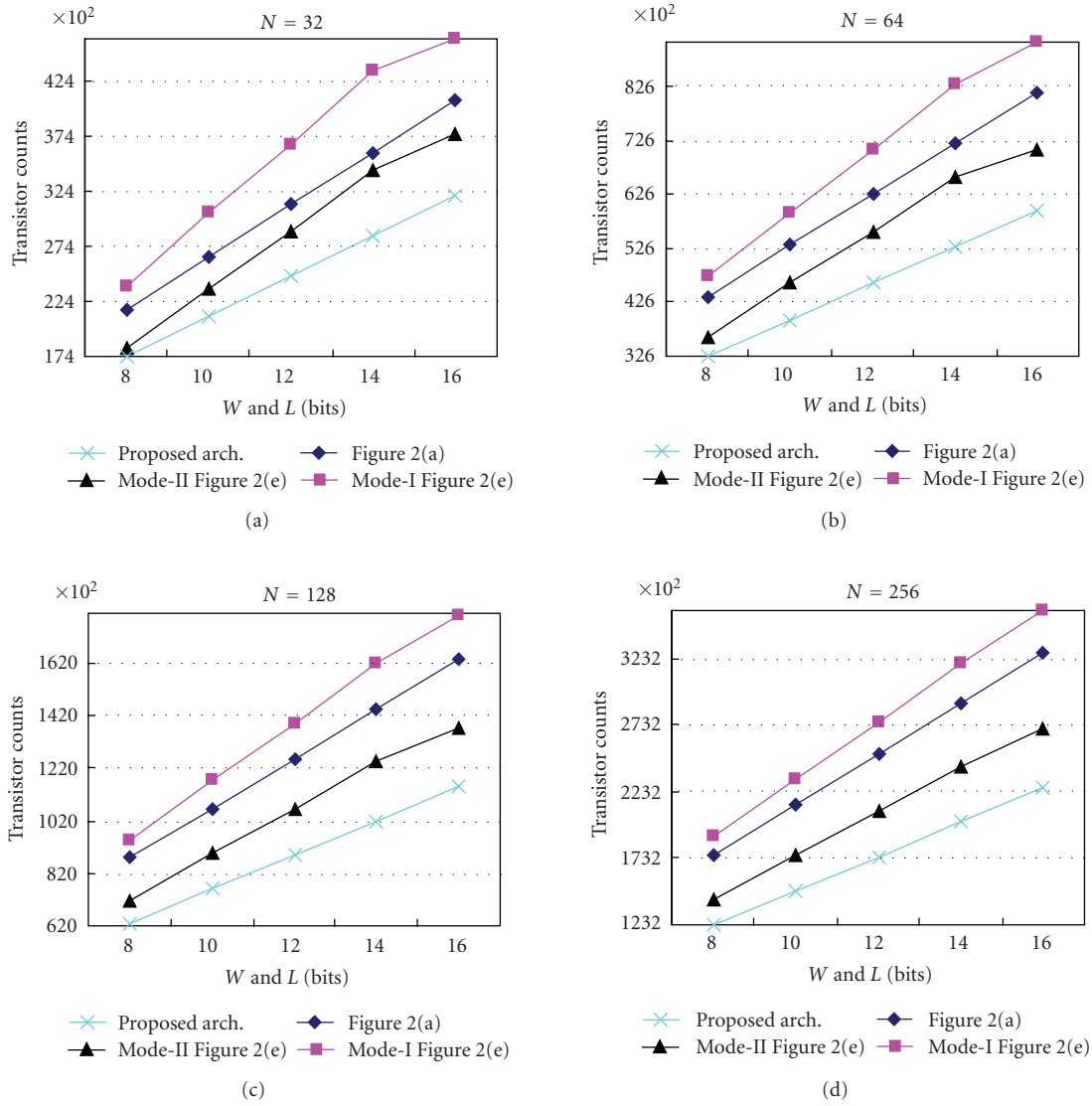

FIGURE 9: Transistor counts of the proposed architecture and conventional FIR architectures in Figures 2(a) and 2(e).

architecture in Figure 2(a), the folding number is $W/2$, which would derive at a critical delay of $4T_{FA}$ according to (9). In comparison to the proposed architecture, the pipelined stage in Figure 2(a) architecture would have a much longer delay. As for the architecture in Figure 2(e), since the folding number and critical delay in this architecture are both changeable, two modes of Figure 2(e) are considered for our comparison purposes. The architectures in the modes I and II of Figure 2(e) have different folding numbers and critical delays, but both still operate at the same throughput rate. The folding number and critical delay of the architecture in the mode-I of Figure 2(e) are $W/2$ and $4T_{FA}$, respectively, and the folding number and critical delay of the architecture in mode-II of Figure 2(e) are W and $2T_{FA}$, respectively.

As illustrated in Figure 9, the architecture in the mode-I of Figure 2(e) would require the most number of transistors due to having an accumulation path with a high hardware complexity and a low folding number. In contrast, the architecture in the mode-II of Figure 2(e), also using only tap folding, has a high folding number and a low critical delay, but a lower hardware complexity than those of the architectures in Figure 2(a) and mode-I of Figure 2(e). This phenomenon explains that under the same throughput rate, increasing the folding number instead of reducing pipelined latches could cut down more hardware complexity. On the other hand, the integrated input-data and tap folding in the proposed architecture can make adders of the accumulation path having small word lengths and the FIR architecture having a high folding number to reduce the hardware complexity. Additionally, the proposed 2-bit input subdata approach and tree accumulation approach can further lower hardware complexity. As shown in Figure 9, the comparison results reveal the proposed architecture to request the least transistor number than the other conventional architectures in realizing an FIR filter.

Table 2: Specification of the proposed programmable FIR processor.

Architecture	Direct form
Multiplier-less operation	Radix-4 Booth algorithm
Folding scheme	Integrating input-data folding and tap folding
Process	Standard cell library of the TSMC 0.18 μm 1P6M CMOS technology
Supply voltage	1.8 V
Tap number	128
Input-data word length $\times$ coefficient word length	10 bits $\times$ 10 bits
Clock frequency	200 MHz
Throughput rate	20 M samples per second
Folding number	10
Core area	675 μm $\times$ 662 μm
Power consumption	46 mW @ 200 MHz, 1.8 V

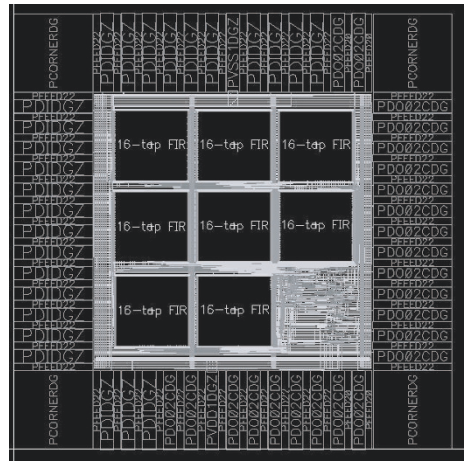

Figure 10: Layout of the proposed 128-tap programmable FIR processor.

5. PROPOSED 128-TAP FIR PROCESSOR

Based on the proposed architecture, the TSMC 0.18 μm single-poly-six-metal CMOS standard cells are employed to realize a 128-tap programmable FIR processor [15]. The Cadance tool is used to generate the layout of the proposed FIR processor, and then extract the netlist. Under such netlist, the Nanosim tool is employed to verify the functionality and power consumption using a uniform-distribution input sequence. This processor's specifications are detailed in Table 2 where input-data and coefficient word lengths are both 10 bits. The folding numbers for input-data and tap folding are 5 (10/2) and 2, respectively, so that the folding number of the proposed processor is 10 (5×2). With the clock frequency operated at 200 MHz, the throughput rate is 20 M samples per second (200 M/10), the core area is 0.45 mm^2, and the layout for the proposed processor is displayed in Figure 10. Table 3 compares the proposed processor with the other programmable FIR processors that use conventional folding schemes. From Table 3, the throughput rate of the proposed processor is larger than those of the conventional processors, indicating that the proposed processor meets the computational performance demands of the conventional processors.

Differences in fabrications and specifications are such that the following normalization must be completed before the areas are compared [16],

$$A = \frac{\text{core area}}{\text{tap number}} \times \left(\frac{0.18}{\text{tech.}}\right)^2 \times \frac{10}{\#\text{ bits coeff.}} \times \frac{10}{\#\text{ bits data}}. \tag{10}$$

Table 3 lists that the tap-folding processors are designed by Wang et al. and Meier and Schobinger. With employing the memory module to store data, Wang et al.'s processor has less hardware cost than Meier and Schobinger's one. However, in comparison to the proposed processor, given that Wang et al.'s processor using tap folding would generate multiplication outputs at full word length, its hardware complexity remains higher than the proposed processor. As for Edwards et al.'s processor, input-data folding is adopted to lower hardware complexity. Yet, the input-data folding inevitably restricts the folding number of this architecture to be limited by the input-data word length and cannot be increased to lower hardware complexity. Lastly, Pao et al. proposes a processor using the half bit-sequential multiplier structure so that the folding number is correlated with input-data and coefficient word lengths. Though this processor has a very high folding number, a full word-length multiplication output is still generated in each tap. The multiplication results from the taps are then summed together. Consequently, the addition in Pao et al.'s processor is performed on product results at a high word length, which then incurs high hardware cost for its adders. With hardware-complexity reduction from the integrated input-data and tap folding, and the approaches using 2-bit input subdata latches and the tree accumulation with simplified carry-in bit processing, the proposed FIR processor is demonstrated to have the least hardware area per tap than the conventional ones.

To fairly compare power consumption of the proposed and conventional FIR processors, the following normalization equation is applied [16, 17]:

$$P = \frac{\text{power}}{\text{tap number}} \times \left(\frac{1.8}{\text{vdd}}\right)^2 \times \frac{0.18}{\text{tech.}} \times \frac{10}{\#\text{ bits coeff.}} \times \frac{10}{\#\text{ bits input data}} \times \frac{20}{\text{throughput rate}}. \tag{11}$$

According to Table 3, the proposed FIR processor can have the least power consumption than the conventional ones owing to its low-complexity hardware design. When considering the product of hardware area and power consumption, the proposed processor still yields the best performance.

Table 3: Comparison of the proposed and conventional programmable FIR processors.

Processors	Features								
	Specifications	Folding schemes	Folding numbers	Throughput rates (samples per second)	Power (mW)	Core areas (mm^2)	**Normalized power per tap (P, mW)**	**Normalized area per tap ($A \times 10^{-3}$ mm^2)**	$P \times A$
Proposed FIR processor	0.18 μm, 1.8 V, 128 taps, 10×10 bits, 200 MHz	Input-data folding + tap folding	10	20 M	46	0.45	**0.36**	**3.51**	**1.26**
Edwards et al.'s processor [5]	1.0 μm, 5 V, 180 taps, 8×8 bits, 57.28 MHz	Input-data folding	4	14.35 M	2500	56.25	**0.71**	**15.82**	**11.23**
Wang et al.'s processor [6]	0.18 μm, 1.8 V, 73 taps, 16×16 bits, 10 MHz	Tap folding	10	1 M	10.69	0.74	**1.14**	**3.96**	**4.51**
Meier and Schobinger's processor [7]	0.5 μm, 16 taps, 8×10 bits, 70 MHz	Tap folding	10	7 M	N/A	1.00	**N/A**	**10.13**	**N/A**
Pao et al.'s processor [9]	0.8 μm, 5 V, 64 taps, 8×10 bits, 324 MHz	Input-data folding + coefficient folding	18	18 M	2300	8.50	**1.46**	**8.40**	**12.26**

6. CONCLUSION

Following advances in fabrication technology, circuits can now operate at a high frequency, while the FIR filter performance needs only to meet the real-time demand. Increasing the architecture's folding number can effectively reduce the hardware complexity, without violating the conditions demanded by the applications. Hence, a hardware-efficient FIR architecture with a high folding number is developed by integrating input-data folding and tap folding. Additionally, the 2-bit input subdata approach and tree accumulation approach with simplified carry-in bit processing are proposed to reduce the hardware complexities of input subdata latches and accumulation path, respectively. Based on the proposed architecture, the TSMC 0.18 μm CMOS technology is applied to realize a 128-tap programmable FIR processor with 10-bit input data and coefficients. Operating at 200 MHz frequency, the processor has a core area of 0.45 mm^2 and yields a throughput rate of 20 M samples per second. In comparison to conventional FIR processors, the proposed processor is able to achieve hardware efficiency owing to its low-complexity architecture design.

ACKNOWLEDGMENT

This project was partially supported by National Science Council, Taiwan, under Contract no. NSC 93-2215-E-194-002.

REFERENCES

[1] H. Li and C. N. Zhang, "Low-complexity versatile finite field multiplier in normal basis," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 9, pp. 954–960, 2002.

[2] A. Bigdeli, M. Biglari-Abhari, Z. Salcic, and Y. T. Lai, "A new pipelined systolic array-based architecture for matrix inversion in FPGAs with Kalman filter case study," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 89186, 12 pages, 2006.

[3] L.-H. Chen and O. T.-C. Chen, "A low-complexity and high-speed Booth-algorithm FIR architecture," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '01)*, vol. 4, pp. 338–341, Sydney, NSW, Australia, May 2001.

[4] L.-H. Chen and O. T.-C. Chen, "A hardware-efficient FIR architecture with input-data and tap folding," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 1, pp. 544–547, Kobe, Japan, May 2005.

[5] B. Edwards, A. Corry, N. Weste, and C. Greenberg, "A single-chip video ghost canceller," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 3, pp. 379–383, 1993.

[6] C. H. Wang, A. T. Erdogan, and T. Arslan, "High throughput and low power FIR filtering IP cores," in *Proceedings of IEEE International SOC Conference*, pp. 127–130, Santa Clara, Calif, USA, September 2004.

[7] S. R. Meier and M. Schobinger, "Time-sharing architectures for FIR filter structures," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*

(ICASSP '00), vol. 6, pp. 3307–3310, Istanbul, Turkey, June 2000.

[8] S. Saponara, L. Fanucci, and P. Terreni, "Design of a low-power VLSI macrocell for nonlinear adaptive video noise reduction," *EURASIP Journal on Applied Signal Processing*, vol. 2004, no. 12, pp. 1921–1930, 2004.

[9] S. Pao, K.-Y. Khoo, and A. N. Willson Jr., "A programmable FIR filter for TV ghost cancellation," in *Proceedings of the 39th IEEE Midwest Symposium on Circuits and Systems (MWSCAS '96)*, vol. 1, pp. 133–136, Ames, Iowa, USA, August 1996.

[10] O. T.-C. Chen and W.-L. Liu, "An FIR processor with programmable dynamic data ranges," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 4, pp. 440–446, 2000.

[11] D. S. Dawoud, "Realization of pipelined multiplier-free FIR digital filter," in *Proceedings of the 5th IEEE AFRICON Conference on Electrotechnological Services for Africa (AFRICON '99)*, vol. 1, pp. 335–338, Cape Town, South Africa, September-October 1999.

[12] I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1993.

[13] P. Pirsch, *Architectures for Digital Signal Processing*, John Wiley & Sons, New York, NY, USA, 1998.

[14] L.-H. Chen, W.-L. Liu, and O. T.-C. Chen, "Determination of radix numbers of the Booth algorithm for the optimized programmable FIR architecture," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '00)*, vol. 2, pp. 345–348, Geneva, Switzerland, May 2000.

[15] "TSMC 0.18μm Process 1.8 Volt SAGE-X^{TM} Standard Cell Library Databook," Artisan components, September 2003.

[16] K.-S. Kim and K. Lee, "Low-power and area-efficient FIR filter implementation suitable for multiple taps," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 1, pp. 150–153, 2003.

[17] C. J. Nicol, P. Larsson, K. Azadet, and J. H. O'Neill, "A low-power 128-tap digital adaptive equalizer for broadband modems," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 11, pp. 1777–1789, 1997.

Oscal T.-C. Chen was born in Taiwan in 1965. He received the B.S. degree in electrical engineering from National Taiwan University in 1987, M.S. and Ph.D. degrees in electrical engineering from University of Southern California, Los Angeles, USA, in 1990 and 1994, respectively. He worked in Computer Processor Architecture Department of Computer Communication & Research Labs. (CCL), Industrial Technology Research Institute (ITRI), for serving a system design engineer, project leader, and section chief from 1994 to 1995. He was an Associate Professor in Department of Electrical Engineering, National Chung Cheng University (NCCU), Chia-yi, Taiwan, from September 1995 to August 2003. After August 2003, he became a Professor in Department of Electrical Engineering, NCCU. In the technical society, he was an Associate Editor of IEEE Circuits & Devices Magazine from August 2003, and a founding member of the multimedia systems and applications technical committee of IEEE Circuits and Systems Society. He participates in the Technical Program Committee of many IEEE international conferences and symposiums. He was the co-recipient of the Best Paper Award of IEEE Transactions on VLSI Systems in 1995. His research interests include video/audio processing, DSP processors, VLSI systems, RF IC, microsensors, and communication systems.

Li-Hsun Chen was born in Taiwan, in 1976. He received the B.S. degree in electrical engineering from National Chung Cheng University in 1998. Currently, he is working toward the Ph.D. degree in electrical engineering. His research interests include digital filter design, reconfigurable architectures, DSP processors, and VLSI systems.