# An FIR Processor with Programmable Dynamic Data Ranges

Oscal T.-C. Chen and Wei-Lung Liu

***Abstract*—This work developed a modified direct form based on the radix-4 Booth algorithm to realize a finite impulse response (FIR) architecture with programmable dynamic ranges of input data and filter coefficients. This architecture comprises a preprocessing unit, data latches, configurable connection units, double Booth decoders, coefficient registers, a path control unit, and a postprocessing unit. Programmable dynamic ranges of input data and filter coefficients can be any positive even numbers or multiple of a word length of coefficient registers, using configurable connection units or a path control unit, respectively. In particular, the proposed architecture employs only data-path controls to accomplish programmable operations, without changing word lengths and components of data latches and filter taps. A practical 8-bit and 16-bit FIR processor has also been implemented by using the TSMC 5 V 0.6 $\mu$m CMOS technology. It is suitable for operations of asymmetric, symmetric, and anti-symmetric filters at 64, 63, 32, 31, and 16 taps, and is well explored to optimize its functional units. The proposed processor has throughput rates of 50 M and 25 M samples/s for 8-bit and 16-bit input data of various filter applications, respectively.**

***Index Terms*—Arithmetic, configurable, digital-CMOS, digital-filter, VLSI.**

## I. Introduction

Finite impulse response (FIR) filters have found extensive use in many real-world applications such as communication processing and multimedia computing. The efficient implementation architectures of FIR filters, such as a memory-based approach [4], a canonical signed-digit (CSD) approach [3], and a Booth-algorithm approach [1], [2], [5], have been exhaustively explored in the recent decade. This work improves upon the conventional FIR architectures with programmable dynamic ranges of input data and filter coefficients. In memory-based FIR architecture design [4], the word length of input data and the precision of filter coefficients are usually fixed for one memory configuration. To achieve programmable dynamic ranges, memory cells have to be reconfigured and connections among taps have to be rearranged.

Due to the high cost of the original architecture for a large dynamic data range, the memory-based FIR architecture may not be a good candidate for programmable design. In CSD FIR architecture design, each filter coefficient is represented by many digits of $+1$, 0, and $-1$, where only $+1$ and $-1$ digits interpret addition and subtraction operations of input data, respectively. Khoo *et al.*presented a CSD FIR architecture that is easily programmable for different precision of filter coefficients, but in this architecture, most functional units of each tap require the maximum word-length design [3]. In addition, all CSD taps are directly addressed by every input datum, using a fixed word-length data bus. When considering a large dynamic range of input data, we need to partition input data into a subdatum sequence. In such a case, a complicated tap design is needed to support computation in this FIR architecture; therefore, CSD FIR filters with programmable dynamic ranges of input data may not be realized at a low cost. By employing the Booth algorithm in an FIR architecture design [1], [2], [5], input data are recoded into the bit-level format, so that different dynamic ranges of input data can be achieved by configuring connections between an input bit-level sequence and filter taps. In addition, the precision of filter coefficients can be programmed by configuring connections among filter taps, because these filter taps have regular structures. Therefore, the FIR architecture using the Booth algorithm can be a cost-effective design to achieve programmable dynamic data ranges.

Manuscript received November 18, 1998; revised July 19, 1999. This work was supported in part by the National Science Council, Taiwan, R.O.C., under Contract NSC 87-2213-E-194-024 and by the Computer and Communication Research Laboratories, ITRI, Taiwan, under Contract G4-86023-p.

The authors are with the Signal and Media Laboratories, Department of Electrical Engineering, National Chung Cheng University, Chia-Yi 621, Taiwan, R.O.C.

Publisher Item Identifier S 1063-8210(00)01029-5.

## II. FIR Computing Based on the Radix-4 Booth Algorithm

Generally, the speed of a multiplication-accumulation operation has a critical impact on the speed of computing in an FIR architecture. To reduce the complexity of multiplication [5], the radix-4 Booth algorithm is used to interpret input data of $\mathbf{X}$. Each datum of $X_i$ is partitioned into many 3-bit groups (triplets), each of which has one bit overlapped with the previous group. Hence, the 2's complement of $X_{n-i}$ with a word length of $W$ can be represented by

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

where $X_{n-i}^{(j)}$ is the $j$th digit of $X_{n-i}$ and $X_{n-i}^{(-1)}$ is equal to zero. When considering a filter coefficient $C_i$ multiplied by $X_{n-i}$, (1) is modified to

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

According to (2), $B(X_{n-i,l}, C_i)$ is the intermediate product that can be represented by five different values of $-2C_j, -C_j, 0, C_j$, and $2C_j$. In addition, the multiplication between an input datum and a filter coefficient requires $w/2$ summations.

When considering an N-tap FIR, the multiplication between $X_{n-i}$ and $C_i$ can be accomplished by the radix-4 Booth algorithm. Applying (2) leads to

$$\begin{aligned} Y_n &= \sum_{i=0}^{N-1} C_i \times X_{n-i} = \sum_{i=0}^{N-1} \left[ \sum_{l=0}^{(W/2)-1} B(X_{n-i,l}, C_i) \times 2^{2l} \right] \\ &= \sum_{l=0}^{(W/2)-1} \sum_{i=0}^{N-1} \left[B(X_{n-i,l}, C_i)\right] \times 2^{2l}. \end{aligned} \tag{3}$$

For some low-pass and high-pass filters with symmetric and anti-symmetric coefficients, respectively, input data of $X_{n-i}$ and $X_{n-N+1+i}$ are added or subtracted to generate a result; that result is then multiplied by a filter coefficient $C_i$. Using such an approach, the hardware needed

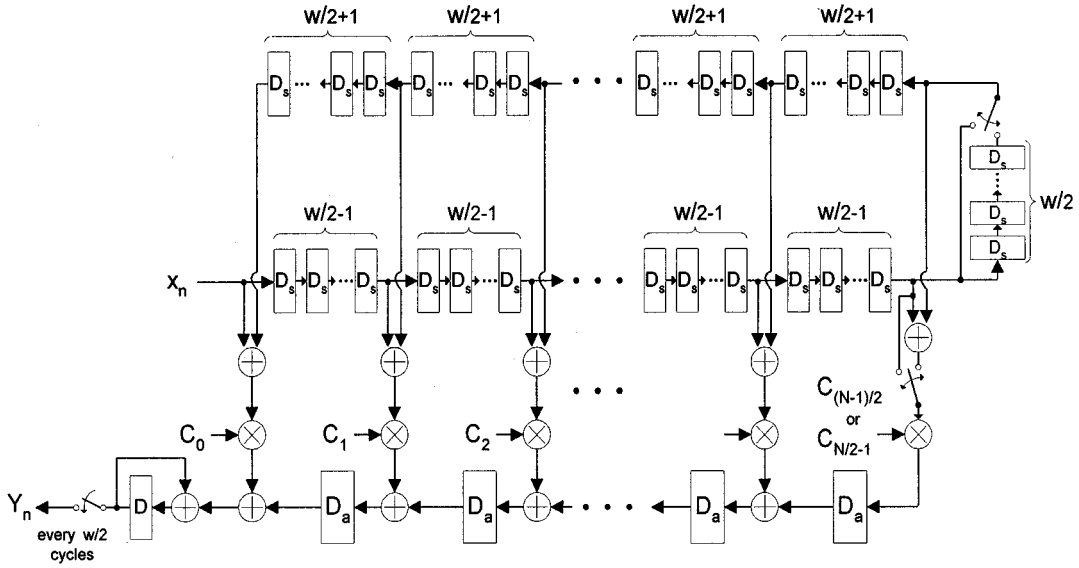


Fig. 1. (Anti)-symmetric FIR using the transposed direct-form architecture.

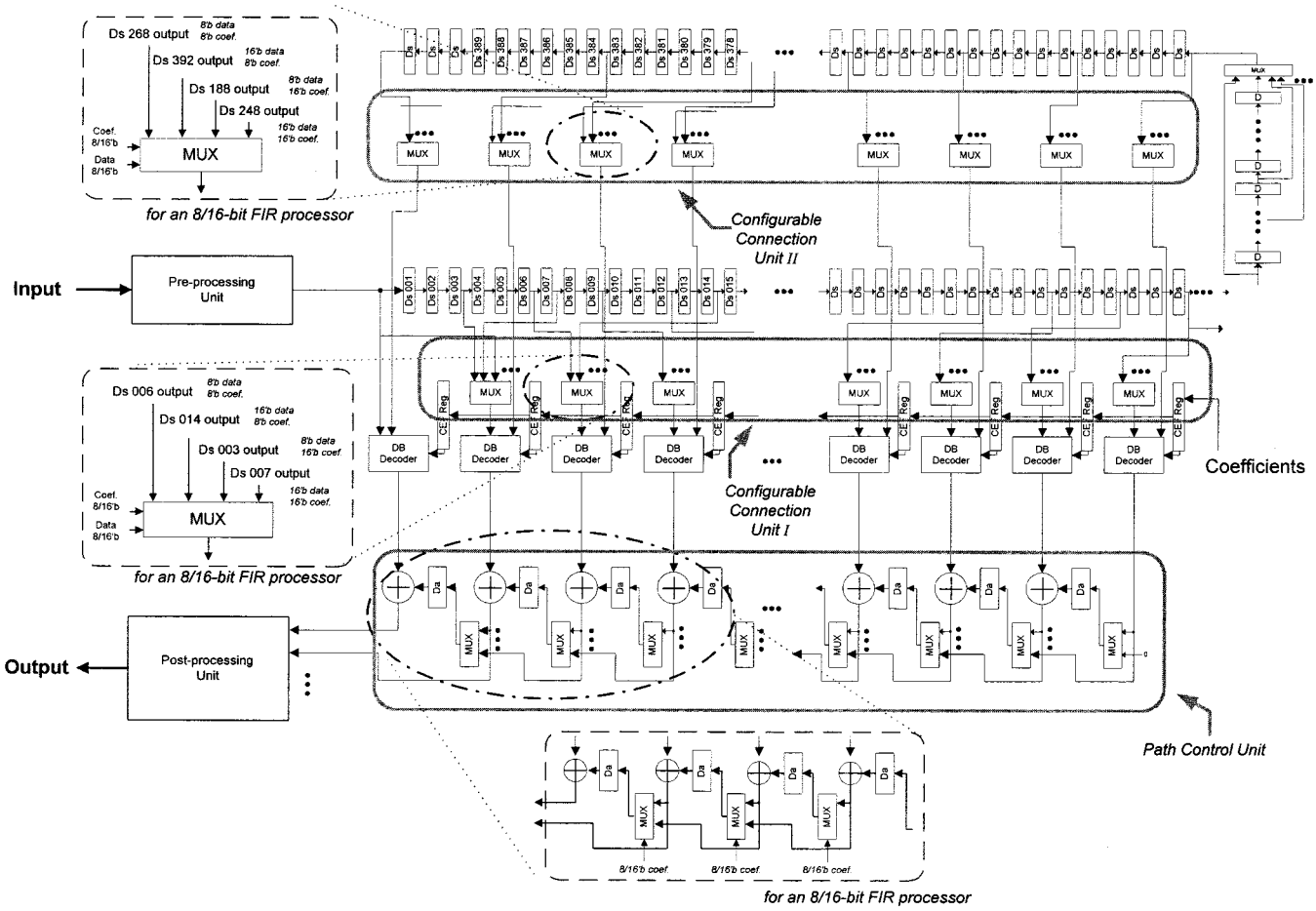


Fig. 2. Proposed FIR architecture.

for an (anti)-symmetric N-tap FIR processor is less complex than that of an asymmetric N-tap FIR processor used for (anti)-symmetric FIR computing. Hence, there is a need to incorporate (anti)-symmetric FIR computing in a general FIR processor design. By using the transposed direct-form architecture, the (anti)-symmetric FIR can be implemented as shown in Fig. 1. Here, we illustrate a symmetric FIR with an even

TABLE I
PROGRAMMABLE CAPABILITIES OF THE PROPOSED AND OTHER FIR ARCHITECTURES

| Programming capabilities / FIR architectures | Multiplication schemes | Programmable dynamic ranges of input data | Programmable dynamic ranges of filter coefficients | Programmable (anti)-symmetric and asymmetric | Programmable coefficients | Programmable tap number | On-chip coefficient learning function |
|---|---|---|---|---|---|---|---|
| The proposed architecture | Booth algorithm | Yes | Yes | Yes | Yes | Yes | No |
| Nicol et al.[1] | Booth algorithm | No | No | No | Yes | Yes | Yes |
| Choi et al.[2] | Booth algorithm | No | No | No | Yes | No | No |
| Khoo et al.[3] | CSD | No | Yes | No | Yes | Yes | No |
| Golla et al.[4] | Memory | No | No | Yes | Yes | Yes | No |

number of filter taps. Equation (3) can be modified to

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

where the double Booth decoder, $\tilde{B}(\cdot)$, has nine possible values: $-4C_i, -3C_i, -2C_i, -C_i, 0, C_i, 2C_i, 3C_i$, and $4C_i$. In order to achieve programmable dynamic ranges of input data and filter coefficients, (4) is modified by the following:

$$Y_n = \sum_{l=0}^{(W/2)-1} \left[ \sum_{j=0}^{P-1} \left( \sum_{i=0}^{(N/2)-1} \tilde{B}(X_{n-i,l}, X_{n-N+1+i,l}, C_{i,j}) \right) \times 2^{-(j \times k-1)} \right] \times 2^{2l} \tag{5}$$

where $C_{i,j}$ is the $j$th subprecision component of $C_i$, $P$ is the number of subcoefficients, and $k$ represents a word length of a subcoefficient. According to (5), programmable capabilities can be realized in the control of $W$ and $P$.

## III. PROPOSED FIR ARCHITECTURE

By using the radix-4 Booth algorithm for multiplication, the proposed programmable FIR architecture, as shown in Fig. 2, consists of a preprocessing unit, data latches, configurable connection units, double Booth decoders, filter coefficient registers, a path control unit, and a postprocessing unit. In particular, the proposed architecture employs only data-path controls to realize the programmable operations, which can make the FIR design more regular and low-cost. Table I lists the programmable capabilities of the proposed FIR architecture and compares them with those of FIR architectures proposed in other work [1]–[4]. The proposed FIR architecture has the most programming capabilities besides on-chip coefficient updating. Each unit of the proposed programmable FIR architecture is described as follows.

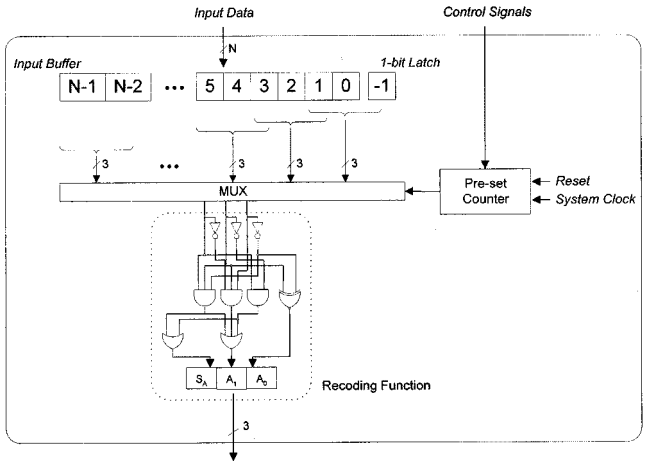


Fig. 3. Preprocessing unit.

*1) Preprocessing Unit:* The preprocessing unit partitions input data as a triplet sequence according to the radix-4 Booth algorithm. This unit consists of an input buffer, a preset counter, a multiplexor, and a recoding function, as shown in Fig. 3. The input buffer with the maximum word length stores input data in various dynamic ranges. The preset counter generates control signals for the multiplexor to select 3-bit data in a correct sequence. According to (1), each triplet datum is recoded in the 2's complement representation using a recoding function, and is then pipelined into data latches.

*2) Data Latches:* Data latches realized by edge-triggered, true-single-phase flip-flops, store a triplet sequence from the preprocessing unit.

*3) Configurable Connection Units:* Configurable connection units select sequence units for convolution calculation. Simple multiplexors can be used to configure a connection topology for efficiently realizing programmable dynamic ranges of input data, as presented in Fig. 2. Each (anti)-symmetric filter tap connected to several input data latches is controlled by two multiplexors for different programmable ranges. Only two paths are enabled to link an (anti)-symmetric

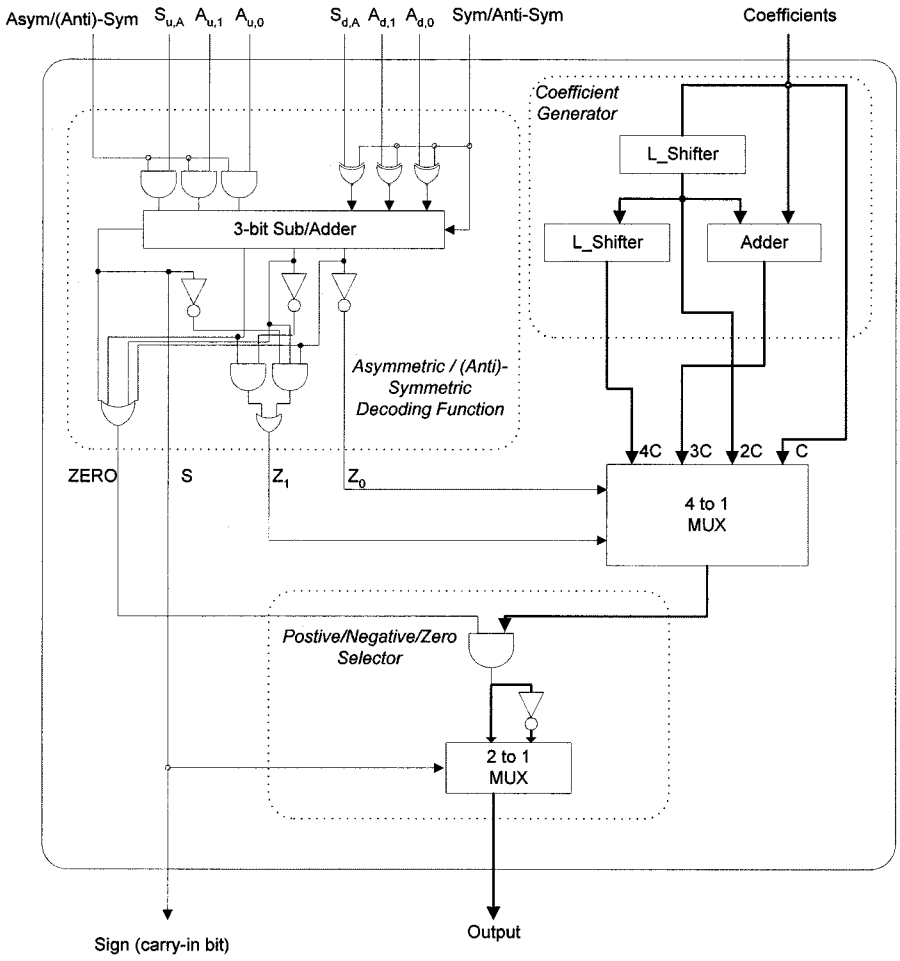


Fig. 4. Double Booth decoder.

TABLE II
REQUIRED NUMBERS OF FUNCTIONAL UNITS FOR 8-BIT OR 16-BIT INPUT DATA AND FILTER COEFFICIENTS

| Functional units / Cases | Filter coef. (bits) | Input data (bits) | Number of data latches (Ds) | | | | Number of double Booth decoders | Number of 8-bit filter coefficient registers | Maximum tap number |
|---|---|---|---|---|---|---|---|---|---|
| | | | A | B | C | A+B+C | | | |
| (1) | 8 | 8 | 93 | 4 | 155 | 252 | 32 | 32 | 64 (anti)-symmetric |
| (2) | 8 | 8 | 93 | 0 | 155 | 248 | 32 | 32 | 63 (anti)-symmetric |
| (3) | 8 | 8 | 93 | 0 | 0 | 93 | 32 | 32 | 32 asymmetric |
| (4) | 8 | 16 | 217 | 8 | 279 | 504 | 32 | 32 | 64 (anti)-symmetric |
| (5) | 8 | 16 | 217 | 0 | 279 | 496 | 32 | 32 | 63 asymmetric |
| (6) | 8 | 16 | 217 | 0 | 0 | 217 | 32 | 32 | 32 asymmetric |
| (7) | 16 | 8 | 45 | 4 | 75 | 124 | 32 | 32 | 32 (anti)-symmetric |
| (8) | 16 | 8 | 45 | 0 | 75 | 120 | 32 | 32 | 31 (anti)-symmetric |
| (9) | 16 | 8 | 45 | 0 | 0 | 45 | 32 | 32 | 16 asymmetric |
| (10) | 16 | 16 | 105 | 8 | 135 | 248 | 32 | 32 | 32 (anti)-symmetric |
| (11) | 16 | 16 | 105 | 0 | 135 | 240 | 32 | 32 | 31 (anti)-symmetric |
| (12) | 16 | 16 | 105 | 0 | 0 | 105 | 32 | 32 | 16 asymmetric |
| (13) | 8/16 | 8/16 | 105 | 8 | 155 | 268 | 32 | 32 | Table 3 |

filter tap with two latches of input data. One path comes from the configurable connection unit I, and the other from the configurable connection unit II.

*4) Double Booth Decoders:* The double Booth decoder, as shown in Fig. 4, receives two selected triplets whose values are added or subtracted according to the operation of a symmetric or anti-symmetric

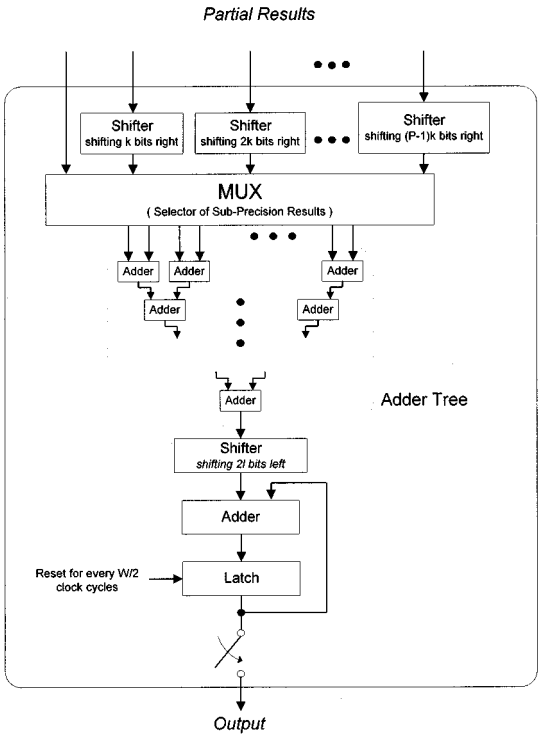


Fig. 5. Postprocessing unit.

TABLE III
OPERATION MODES OF THE PROPOSED 8-BIT AND 16-BIT FIR PROCESSOR

| Filter Coefficients \ Input Data | | | 8 bits | 16 bits |
|---|---|---|---|---|
| 8 bits | **Asymmetric** | | 32 taps | 32 taps |
| | (Anti)-Symmetric | even | 64 taps | NA |
| | | odd | 63 taps | NA |
| 16 bits | **Asymmetric** | | 16 taps | 16 taps |
| | (Anti)-Symmetric | even | 32 taps | 32 taps |
| | | odd | 31 taps | 31 taps |

filter, using a 3-bit carry-lookahead adder. In addition, one input of the double Booth decoder is controlled by AND gates to determine an asymmetric or (anti)-symmetric operation. The added or subtracted result is encoded into four control signals: $S, Z_0, Z_1$, and $Z_2$. Here, $Z_1$ and $Z_2$ control a 4-to-1 multiplexor to generate an output result of $C_i, 2C_i, 3C_i$ or $4C_i$, where $3C_i$ is produced by summing $2C_i$ and $C_i$ in a half adder. An AND function using $Z_0$ is performed on the intermediate result from the 4-to-1 multiplexor, where $Z_0$ interprets whether the output result is zero. The signal $S$ determines whether the final result is positive or negative. If $S$ is equal to logic 1, the result from the AND function is inverted and the following accumulation adder is fed with a carry-in bit of logic 1. Otherwise, the result from the AND function is directly added by the following accumulation adder with a carry-in bit of logic 0.

*5) Filter Coefficient Registers:* The filter coefficient registers store the values of the filter coefficients for operations of the double Booth decoders. They are realized by edge-triggered, true-single-phase flip-flops, with improving speed and power saving that can support a high-speed and power-efficient pipeline operation for programming filter coefficients [6].

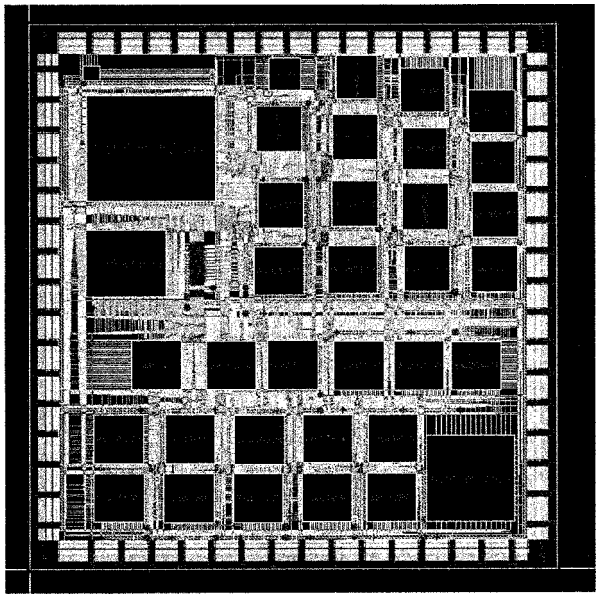

Fig. 6. Layout of the proposed FIR processor.

TABLE IV
SPECIFICATIONS OF THE PROPOSED FIR PROCESSOR

| Chip | A Programmable FIR Processor |
|---|---|
| Architecture | Transposed direct-form architecture |
| Multiplier-less operation | Radix-4 Booth algorithm |
| Programmable scheme | Efficient path controls |
| Input data | 8 or 16 bits |
| Filter coefficients | 8 or 16 bits |
| Number of taps | 16, 31, 32, 63, and 64 |
| Technology | TSMC 0.6 $\mu$ m CMOS |
| Design scheme | COMPASS standard cell library |
| Supply voltage | 5V |
| Clock frequency | 200MHz |
| Power consumption | 3.8W@200MHz |
| Die size | 6246 $\mu$ m $\times$ 6319 $\mu$ m |
| Number of pads | 63 |
| Package | 64CQFP |
| Throughput rate | 50MHz or 25MHz samples per second |

*6) Path Control Unit:* A path control unit selects an accumulation path for the accumulation of intermediate products, as shown in Fig. 2. Since the word length of registers used for storing filter coefficients is fixed, a filter coefficient might need to be stored in several registers. According to (5), each filter coefficient is partitioned into several nonoverlapped subcoefficients that are individually utilized in the decoding function of the Booth algorithm. In designing a path control unit, the main idea is to arrange the accumulation relationship among the product values of subcoefficients and input data. A latch is needed in every two neighboring taps at the accumulation flow to effectively realize the path control unit.

*7) Postprocessing Unit:* The postprocessing unit performs the final accumulation of the intermediate results from the convolution calculation between the selected sequence units and the filter coefficients. First, the selector of subprecision results, which is implemented by multiplexors, is used to determine the effective data according to the operation mode of the coefficient precision. According to (5), the $P$ effective results with sign extension are summed using an adder tree.

TABLE V
PERFORMANCE COMPARISONS BETWEEN THE PROPOSED FIR PROCESSOR AND PROCESSORS PRESENTED IN OTHER WORK

| Specifications / FIR architectures | Supply voltage | Clock frequency | Throughput rate | Tech-nology | Area | Power | Hardware components for multiplication | (ii)† A(mult) | (i)† P(mult) |
|---|---|---|---|---|---|---|---|---|---|
| The proposed processor | 5V | 200MHz | 50M samples/sec for 8-bit input data<br>25M samples/sec for 16-bit input data | 0.6μm | 39.7 $mm^2$ | 3.8W | (typical cases)<br>16-tap asym. coef. 16×16 multiplication<br>32-tap sym. coef. 16×16 multiplication | 0.81 $mm^2$<br>0.40 $mm^2$ | 10.1mW<br>5.1mW |
| Nicol et al.[1] | 3.3V | 50MHz | 50M samples/sec | 0.5μm | 21 $mm^2$ | 0.162W | 32-tap asymmetric coef. 10 ×12 multiplication | 0.66 $mm^2$ | 5.1mW |
| Choi et al.[2] | 3.3V | 60MHz | 15M samples/sec | 0.6μm | 63 $mm^2$ | 3.5W | 72-tap asymmetric coef. 10×10 multiplication | 0.73 $mm^2$ | 40.5mW |
| Khoo et al.[3] | 5V | 180MHz | (1)45M samples/sec for 8 non-zero digits<br>(2)90M samples/sec for 4 non-zero digits<br>(3)180M samples/sec for 2 non-zero digits<br>(in 16-bit coefficients) | 1.0μm | 11.8 $mm^2$ | 1.3W | (iii)†<br>(1)8-tap asym. coef. 16×15.1 multiplication<br>(2)16-tap asym. coef. 16×13.9 multiplication<br>(3)32-tap asym. coef. 16×8.8 multiplication | 0.18 $mm^2$<br>0.10 $mm^2$<br>0.08 $mm^2$ | 4.9mW<br>2.7mW<br>2.1mW |
| Golla et al.[4] | 5V | 30MHz | 30M samples/sec | 1.2μm | 49.6 $mm^2$ | 1.1W | 16-tap asym. coef. 8×9 multiplication<br>32-tap sym. coef. 8×9 multiplication | 0.90 $mm^2$<br>0.45 $mm^2$ | 34.7mW<br>17.3mW |

Note:

(i) $$P(mult) = \frac{Total\ power}{\#\ mults} \times \frac{12}{\#coef.\ bits} \times \frac{10}{\#sample\ bits} \times \left(\frac{3.3}{Vdd}\right)^2 \times \frac{0.5}{Tech.} \times \frac{50}{clock\ freq.}$$ [3]

(ii) $$A(mult) = \frac{Total\ area}{\#\ mults} \times \frac{12}{\#coef.\ bits} \times \frac{10}{\#sample\ bits} \times \left(\frac{0.5}{Tech.}\right)^2$$

(iii)The numbers of values interpreted by 2, 4, and 8 non-zero digits in 16-bit coefficients are $2^{8.8}$, $2^{13.9}$, and $2^{15.1}$, respectively, where the values of 8.8, 13.9, and 15.1 are equivalently viewed as word lengths of filter coefficients.

Then, output results are accumulated $w/2$ times for a final convolution value where a shifter before this accumulator is required to shift $2l$ bits left, according to (5). Fig. 5 shows the postprocessing unit that supports various dynamic ranges of input data and filter coefficients; the major components of the post processing unit are adders, shifters, latches, and multiplexors.

## IV. 8-BIT AND 16-BIT FIR PROCESSOR

The proposed FIR processor is designed to support applications of speech, audio, and image filter processing. Generally, word lengths of speech, audio, and image signals are 8 or 16 bits. A better filtering performance can be achieved by adequately rounding off filter coefficients on finite word-length FIR architectures. Hence, this study considered a general FIR computing core with 8 and 16 bits of input data and filter coefficients. The proposed FIR processor, based on 32 8-bit filter coefficient registers and 32 double Booth decoders, was explored with the aim of reducing the number of functional units. Due to the recoding of the radix-4 Booth algorithm, the numbers of data latches required for 8-bit and 16-bit input data are quite different. Table II lists the required numbers of functional units for the maximum tap number at 8-bit or 16-bit input data and filter coefficients. Herein, the data latches can be classified into three groups: A, B, and C. Since the proposed architecture is the transposed direct-form structure, the pipelining directions of the data latches in groups A and C are inverse to and the same as that of the accumulation flow, respectively. The data latches in the group B are used in the (anti)-symmetric FIR with an even number of filter taps. According to Table II, cases (4) and (5) require more than twice the number of data latches used in the other cases. To compromise flexibility and hardware cost, cases (4) and (5) of 64-tap and 63-tap 8-bit (anti)-symmetric filter coefficients at 16-bit input data are not supported. Case (13) is the configuration of the proposed FIR processor, which in turn is the result from integrating the configurations of cases (1) to (3) and (6) to (12). Especially in case (6), there are 217 data latches required in the group A: These data latches are partitioned into 105 data latches in the group A and 112 data latches in the group C. Using such an approach, case (6) can be realized in the proposed case (13). Table III lists ten programmable operations supported by this 8-bit and 16-bit FIR processor.

The proposed FIR processor has ten different operation modes realized by the path control unit and configuration connection units. The path control unit controls the accumulation paths of the 8-bit and 16-bit filter coefficients, where 2-to-1 multiplexors are used. The configurable connection units I and II are also realized using multiplexors that select adequate data latches for each double Booth decoder. In the proposed design, data latches are partitioned into three groups. In the group A, the last effective data latch is connected to the data latch in the group B or group C. Since this last effective latch may not be the last hardware component, the unused data latches in the group A for different dynamic ranges of input data and filter coefficients have to be bypassed. In such an approach, topologies of configurable connection units of cases (1) to (3), (7) to (9) or (10) to (12) are the same. Instead of ten operation modes, only four cases are addressed by the 4-to-1 multiplexors of configurable connection units I and II, thereby reducing the units' hardware complexities.

The proposed FIR processor was designed using the COMPASS standard cell library in the TSMC 5 V 0.6 $\mu$m single-poly-three-metal

CMOS technology. Fig. 6 shows the layout of this FIR processor with a die size of $6.3 \times 6.3$ mm$^2$. The power consumption is around 3.8 W at a clock frequency of 200 MHz, where the power and functionality of the proposed processor were analyzed using the PowerMill and TimeMill tools. The throughput rate is 50 MHz for 8-bit input data and 25 MHz for 16-bit input data. There are 241 742 transistors in the design, and the circuit density is around 164 $\mu$m$^2$ per transistor. When considering 8-bit input data and (anti)-symmetric filter coefficients, the proposed FIR processor can have a maximum of 64 taps with a computational power of 12.8 billion multiplication-accumulation operations/s. The specifications of the proposed FIR processor are listed in Table IV. Additionally, performance comparisons between the proposed FIR processor and processors presented in other work are listed in Table V, where the factors of area per $10 \times 12$-bit multiplier A(mult) and power per $10 \times 12$-bit multiplier P(mult) are defined using a 0.5 $\mu$m CMOS technology, a 3.3 V supply voltage, and a 50-MHz clock rate [1]–[4]. The memory-based FIR processor with the 8-bit input data and 9-bit filter coefficients dissipates a higher power [4]. The CSD FIR processor has the lowest power consumption and the smallest area because values of filter coefficients are rounded off by a few nonzero digits [3]. As compared to the other Booth-algorithm FIR processors [1], [2], the proposed processor is highly flexible, and has a good throughput rate, a fair die size and a reasonable power consumption in a 5 V standard cell implementation point of view.

## V. Conclusion

Based on the radix-4 Booth algorithm, this work has successfully developed an FIR architecture with programmable dynamic ranges of input data and filter coefficients. Notably, the proposed architecture employs only data-path controls to accomplish programmable operations. A practical FIR processor with 8-bit and 16-bit dynamic ranges of input data and filter coefficients was also implemented by using the TSMC 5 V 0.6 $\mu$m CMOS technology. Moreover, the processor is optimized for odd or even (anti)-symmetric and asymmetric filter coefficients in ten operation modes. This programmable FIR processor can be operated at a clock frequency of 200 MHz to produce throughput rates of 50 M and 25 M samples/s for 8-bit and 16-bit input data of various industrial applications, respectively.

## Acknowledgment

The authors would like to thank the reviewers for their valuable comments and suggestions.

## References

[1] C. Nicol, P. Larsson, K. Azadet, and J. O'Neill, "A low-power 128-tap digital adaptive equalizer for broadband modems," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1777–1789, Nov. 1997.

[2] J. Choi, S. Jeong, L. Jang, and J. Choi, "Structured design of a 288-tap FIR filter by optimized partial product tree compression," in *Proc. IEEE CICC*, May 1996, pp. 79–82.

[3] K. Khoo, A. Kwentus, and A. Willson, Jr., "A programmable FIR digital filter using CSD coefficients," *IEEE J. Solid-State Circuits*, vol. 31, pp. 869–874, June 1996.

[4] C. Golla, F. Nava, F. Cavallotti, A. Cremonesi, P. Piacentini, and G. Casagrande, "A 30M samples/s programmable filter processor," *Digest IEEE Int. Solid-State Circuits Conf.*, pp. 116–117, Feb. 1990.

[5] W.-L. Liu and O. T.-C. Chen, "A highly-scaleable symmetric/asymmetric FIR processor," in *Proc. IEEE Int. Conf. Acoustic, Speech, Signal Processing*, Mar. 1999, pp. 1917–1920.

[6] J. Yuan and C. Svensson, "New single-clock CMOS latches and flipflops with improved speed and power savings," *IEEE J. Solid-State Circuits*, vol. 32, pp. 62–69, Jan. 1997.

# On-Line Test for Fault-Secure Fault Identification

Samuel N. Hamilton and Alex Orailoğlu

***Abstract*—In an increasing number of applications, reliability is essential. On-line resistance to permanent faults is a difficult and important aspect of providing reliability. Particularly vexing is the problem of fault identification. Current methods are either domain specific or expensive. We have developed a fault-secure methodology for permanent fault identification through algorithmic duplication without necessitating complete functional unit replication. Fault identification is achieved through a unique binding methodology during high-level synthesis based on an extension of parity-like error correction equations in the domain of functional units. The result is an automated chip-level approach with extremely low area and cost overhead.**

***Index Terms*—Faults, reliability, reliable, test, ULSI, VLSI.**

## I. Introduction

As the sophistication and complexity of modern electronics increases, so does our reliance on it. Utilization in critical areas such as medicine, navigation, and transportation is already high and continues to rise. More and more pilots, surgical patients, and even everyday motorists bet their lives daily on the reliability of their electronics. Thus, the need for reliable computational equipment is both immediate and increasing.

Due to the dearth of efficient approaches to fault identification, designers frequently resort to complete duplication of a set of functional units in order to detect errors. Replication allows two units to compare duplicate calculations for consistency. If the calculations are run on disjoint hardware, the design is *fault secure* at the single fault level, as no single fault can corrupt output without detection. When an error does occur, it still must be determined which of the two calculations is correct. The consequent hardware required for fault identification entails additional complexity.

To avoid the cost associated with complete hardware duplication, there has been an increasing focus in the fault-tolerance literature on high-level synthesis [1]. Tailoring high-level synthesis routines for compatibility with efficient hardware solutions facilitates the production of more compact, reliable designs.

In this paper, we propose an algorithmic approach to fault identification that introduces error encoding schemes into high-level synthesis. By endowing scheduling and binding routines with the capacity to embed generic error correction codes, we enable efficient implementation of calculation duplication via load balancing while avoiding hardware complexity associated with traditional fault identification techniques. By adopting algorithmic duplication, we also introduce the capacity for fine grain reliability/cost tradeoffs similar to performance/cost tradeoffs inherent in high-level synthesis while simultaneously guaranteeing fault security for single faults. In addition, unlike approaches based on more restricted fault models, a voting or function unit fault cannot result in erroneous behavior. The proposed technique represents the first fault-secure methodology for fault identification to forgo triplication in favor of the significantly lower cost duplication entails.

Manuscript received November 23, 1998; revised February 12, 1999.

The authors are with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114 (e-mail: hamilton@cs.ucsd.edu; alex@cs.ucsd.edu).

Publisher Item Identifier S 1063-8210(00)01036-2.