

A High-Performance 8-Tap FIR Filter Using Logarithmic Number System

Yan Sun and Min Sik Kim

School of Electrical Engineering and Computer Science

Washington State University

Pullman, Washington 99164-2752, U.S.A.

Email: {ysun,msk}@eecs.wsu.edu

Abstract—This paper presents an approach to implement a high-performance 8-tap digital FIR (Finite Impulse Response) filter using the Logarithmic Number System. In the past, FIR filters were implemented by a conventional number system; their speed was limited because of the multiply-accumulate operations. We realize a fast FIR filter by utilizing the Logarithmic Number System, which allows a simple implementation of multiplication using a fixed-point adder. And the serious demerit of Logarithmic Number System’s algorithm, conversions to and from the conventional number representations, is effectively overcome by pipelining to reduce the delay and complexity of the filter. The critical path was reduced from a multiply-accumulate operation to an add operation. Our FIR filter can operate at 1.3 GHz under the condition of 1.2 V power supply using the SMIC 0.13 μm CMOS technology, and requires 27% less area than the original FIR filter.

I. INTRODUCTION

Finite impulse response (FIR) filters are important building blocks for various digital signal processing (DSP) applications. Recently, because of the increasing demand for video-signal processing and transmission, high-speed and high-order FIR filters have frequently been used to perform adaptive pulse shaping and signal equalization on the received data in real-time, e.g., ghost cancellation [1], [2], source coding, equalizer, Partial-Response Maximum Likelihood (PRML), and channel equalization [3]. Signal processing algorithms often require a substantial amount of floating-point (or fixed-point) computations to be performed at real-time or near real-time speeds. Hence, an efficient VLSI architecture for a high-speed FIR filter is crucial. A FIR filter is composed of multipliers and adders, and their performance adders determines the speed of FIR filter.

In the past, the conventional number system was typically used, and thus the speed could not reach 100 MHz for 10-bit input [2], [4], [5]. The speed was limited mainly by the delay in multipliers and adders. An effective solution is to exploit the unique feature of the Logarithmic Number System (LNS), which can convert multiplications into additions [6]. In such a solution, it would be critical to have an efficient implementation of an adder and the conversion between the conventional number system (CNS) and the Logarithmic Number System (LNS).

In this paper, we propose a novel FIR filter based on the Logarithmic Number System to achieve high performance. The

architecture is designed and optimized to take full advantage of LNS in the CMOS technology. In order to reduce the complexity of the filter further, different types of adders are implemented in different stages of the pipeline.

The rest of the paper is organized as follows. Section II introduces the LNS describes the details of conversions between CNS and LNS architectures. Section III presents the complete pipeline of the proposed filter, and Section IV describes the adders. Section V discusses the design, simulation, and fabrication of the proposed architecture, and Section VI concludes the paper.

II. CONVERSIONS FOR LOGARITHMIC NUMBER SYSTEM

A. Logarithmic Number System

The logarithmic number system is an attractive alternative to conventional number systems when data need to be manipulated at a very high rate over a wide data range. However, the major problem in LNS is deriving logarithms and antilogarithms quickly and accurately enough to allow fast conversions to and from the conventional number representations. The LNS number A has a value of

$$A = (-1)^{S_A} \cdot 2^{E_A} \quad (1)$$

where S_A is the sign bit and E_A is a fixed point number. The sign bit signifies the sign of the whole number. E_A is a 2’s complement fixed-point number, where a negative number represents $-1 < A < 1$. In this way, LNS numbers can represent both very large and very small numbers. A logarithmic number is stored in the format shown in Fig. 1.

Since there are no obvious choices for special values to signify exceptions and also zero cannot be represented in LNS, we use flag bits to code for zero, $\pm\infty$, and NaN in a similar manner as Detrey [7]. Although using two flag bits would increase the storage requirements for logarithmic numbers,

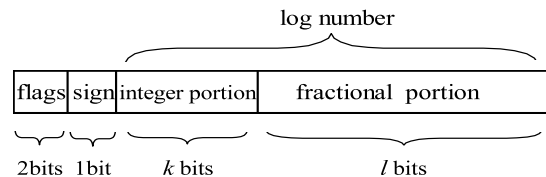


Fig. 1. Binary format of an LNS number

they give the greatest range available for a given bit width. Let ULP be the unit of least precision. Then, for $k = 8$ and $l = 23$, the range is $\pm 2^{-129}$ to $2^{128-ULP}$, which are approximately $\pm 1.5 \times 10^{-39}$ to 3.4×10^{38} .

CNS (such as the floating-point number system) multiplication is complex because of the need to multiply the mantissas and add the exponents. In contrast, multiplication becomes a simple computation in LNS [8]; the product is computed by adding the two fixed point logarithmic numbers using the logarithmic property:

$$\log xy = \log x + \log y. \quad (2)$$

Since the logarithmic numbers are 2's complement fixed-point numbers, addition is an exact operation unless there is overflow or underflow events. Overflow events occur when the two numbers being added sum up to a number too large to be represented in the word width while underflow results when the added sum is too small to be represented. The former results in $\pm\infty$ and the latter results in zero.

B. Conversion from FP to LNS

Floating-point (FP) is a widely-used and standard (IEEE-754) number system. Therefore, modules that covert FP numbers to logarithmic numbers and vice versa are essential in our system. The conversion from FP to LNS involves three steps that can be done in parallel. The first step is checking whether the FP number is one of the special values, and setting the two flag bits accordingly. The second step is to compute the integer portion of the logarithmic number, which is simply the exponent of the FP minus a bias. The last step, computing the fraction, involves evaluating a nonlinear function $\log_2 1.x_1x_2 \dots x_n$. Using a look-up for this computation is viable for smaller mantissas but becomes unreasonable for larger word sizes.

For larger word sizes up to single precision, an approximation developed by Wan [9] is used. This algorithm reduces the amount of memory required by factoring the FP mantissa as follows (for $m = \frac{n}{2}$):

$$\begin{aligned} & \log_2 1.x_1x_2 \dots x_n \\ &= \log_2(1.x_1x_2 \dots x_m)(1.0 \dots 0c_1c_2 \dots c_m) \\ &= \log_2 1.x_1x_2 \dots x_n + \log_2 1.0 \dots 0c_1c_2 \dots c_m \end{aligned}$$

where $.c_1c_2 \dots c_m = \frac{.x_{m+1}x_{m+2} \dots x_{2m}}{1+x_1x_2 \dots x_m}$. The result, $\log_2 1.x_1x_2 \dots x_m$ and $\log_2 1.0 \dots 0c_1c_2 \dots c_m$, can be stored in a $2^m \times n$ ROM.

In an attempt to avoid a slow division, we do the following: if $c = .c_1c_2 \dots c_m$, $b = .x_{m+1}x_{m+2} \dots x_{2m}$, and $a = .x_1x_2 \dots x_m$, then $c = \frac{b}{1+a}$. This can be rewritten as follows:

$$\begin{aligned} c &= \frac{b}{1+a} \\ &= \frac{1+b}{1+a} - \frac{1}{1+a} \\ &= 2^{\log_2(1+b) - \log_2(1+a)} - 2^{-\log_2(1+a)} \end{aligned}$$

where $\log_2(1+b)$ and $\log_2(1+a)$ can be looked up in the same ROM as the one for $\log_2 1.x_1x_2 \dots x_m$. All that remains is calculating 2^z . When $0 \leq z < 1$, this can be approximated as

$$2^z \approx 1 - \log[1 + (1-z)] + 2^{-12} + 2^{-13}. \quad (3)$$

Note that $\log[1+(1-z)]$ can also be evaluated using the same ROM as above. To reduce an error in the approximation, the difference,

$$\Delta z = 2^z - (1 - \log[1 + (1-z)] + 2^{-12} + 2^{-13}), \quad (4)$$

can be stored in another ROM. This ROM only has to be 2^m bits deep and less than m bits wide because Δz is small. This algorithm reduces the lookup table size from $2^n \times n$ to $2^m \times (m + 5n)$ (m from the ROM for Δz and $5n$ from the ROMs for $4 \log_2 1.x_1x_2 \dots x_m$ and $\log_2 1.0 \dots 0c_1c_2 \dots c_m$). For single precision (23-bit fraction), this reduces the memory from 192 MB to 32 KB. The memory requirement can be reduced further if the memories are time-multiplexed or dual-ported. In order to reduce the error caused by conversions, the output of the conversion circuit is designed 4-bits wider than the input in our FIR filter.

C. Conversion from LNS to FP

In order to convert from LNS to FP, we create an efficient converter via simple transformations. The conversion involves three steps. First, the exception flags must be checked and possibly translated into the FP exceptions. Second, the integer portion of the logarithmic number becomes the exponent of the FP number. A bias must be added for conversion, because the logarithmic integer is stored in 2's complement format. And finally, the logarithmic fraction is converted into the FP mantissa.

The last step involves evaluating a non-linear function 2^n . Since the logarithmic fraction is in the range $[0, 1)$, the conversion will be in the range $[1, 2)$, which maps directly to the mantissa without any need to change the exponent. Typically, this is done in a look-up table. For a reasonable word sizes, however, the amount of memory required would become prohibitive. To reduce the memory requirement, we used the property:

$$2^{x+y+z} = 2^x \cdot 2^y \cdot 2^z. \quad (5)$$

The fraction of a logarithmic number is broken up into k numbers in the following manner:

$$.r_1r_2 \dots r_n = x_1x_2 \dots x_{n/k}y_1y_2 \dots y_{n/k}z_1z_2 \dots z_{n/k} \dots \quad (6)$$

The values of $2^{-x_1x_2 \dots x_{n/k}}$, $2^{-y_1y_2 \dots y_{n/k}}$, etc. are stored in k ROMs of size $2^{n/k} \times n$. With this scheme, the memory requirement is $k \cdot (2^{n/k} \times n)$ instead of $2^n \times n$. The memory saving comes at the cost of $(k-1)$ multiplications. The value of k is selected to minimize the area usage for each word size, and we use $k = 4$ in our FIR filter.

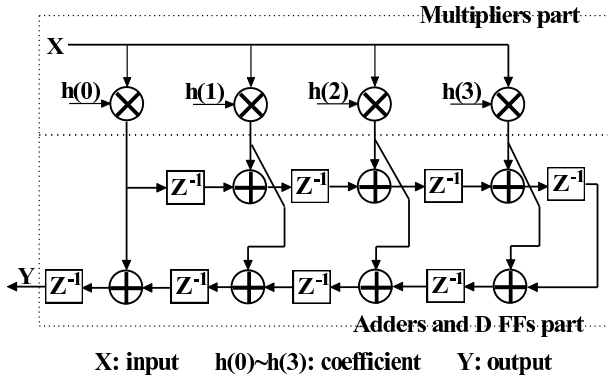


Fig. 2. Conventional FIR filter architecture

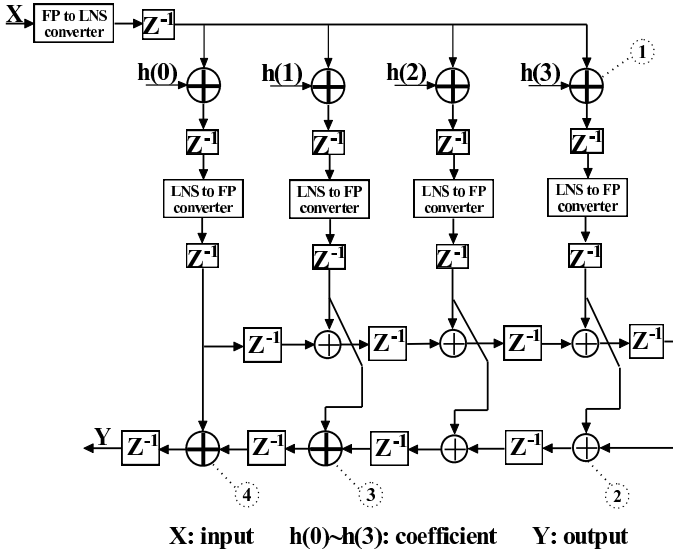


Fig. 3. Proposed FIR filter architecture

III. THE PIPELINED FIR

We design an 8-tap filter with 16-bit input/output and a 16-bit coefficient ($h(0)$ – $h(3)$). The conventional pipeline is presented in Fig. 2. This architecture has been widely adopted for the fast filter architecture because of its pipeline characteristic [10].

The proposed filter architecture is shown in Fig. 3. In our implementation of the architecture using Logarithmic Number System, FP-to-LNS and LNS-to-FP conversions' delays are effectively overcome by hiding them in the pipeline stages. Therefore, only the delay in an adder determines the pipeline clock frequency. The conversions may introduce latency by two clocks but will not cause significant performance degradation. Four kinds of specially-designed adders are marked with ①, ②, ③, and ④ in Fig. 3.

IV. ADDERS USED IN THE PROPOSED FIR

A. Conventional Carry Select Adder

Adders are critical components in the proposed FIR. Among various adders, we select the carry-select adder (CSA) because of its speed and area usage. It is used to calculate the sum

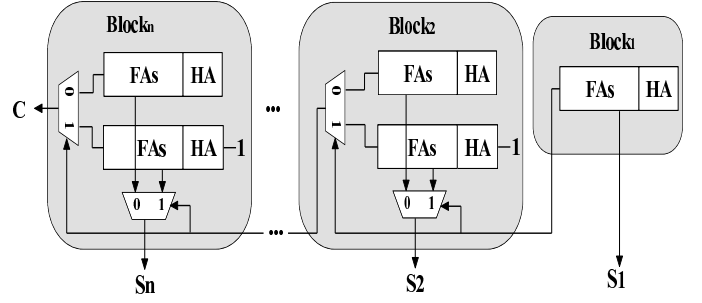


Fig. 4. Conventional CSA using dual RCAs

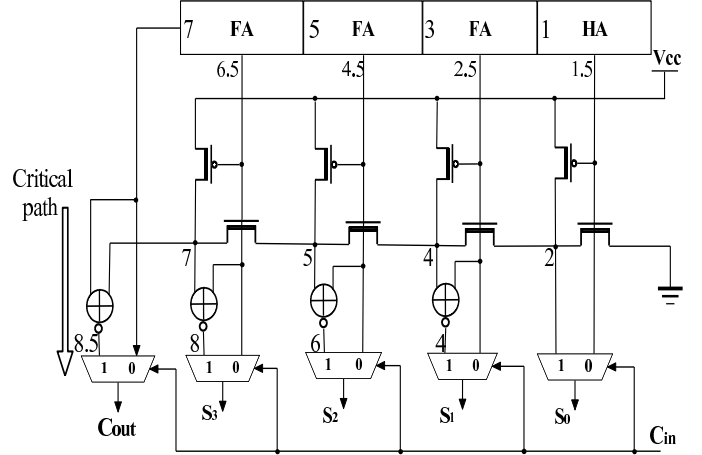


Fig. 5. Carry-select adder using add-one circuit

of logarithmic number and the coefficient. A CSA of a very large size can be constructed hierarchically by combining smaller “block” adders [11]. Fig. 4 shows the conventional CSA consisting of two ripple-carry adders (RCAs) in each block (except Block₁), one for $C_{in} = 0$ and the other for $C_{in} = 1$, where C_{in} is an input from the lower block. A lot of work has been done to reduce the large area of CSA without significant speed penalty [12]–[14]. In this paper, instead of using dual RCAs, we propose a high performance carry-select adder by replacing one RCA with a fast add-one circuit.

Suppose that the result for $C_{in} = 0$ is known to be S^0 . Then, the result for $C_{in} = 1$ (S^1) can be obtained by adding one to S^0 . Thus, an add-one circuit can replace the ripple-carry adder for $C_{in} = 1$ to reduce the area in a block. To design an efficient add-one circuit, the circuit to find the first zero is used in the implement scheme [15]. Adding one to the result for $C_{in} = 0$ (S^0) is conducted as follows. Assume that S_k^0 , the k th bit from the LSB (least significant bit) in S^0 , is the first zero from the LSB. Then S^1 is obtained by inverting each bit of S^0 starting from the LSB to S_k^0 inclusively, and leaving the other bits unchanged. The 4-bit add-one circuit architecture is showed in the Fig. 5.

B. Proposed Carry Select Adder

The carry-chain between blocks is the critical path in a CSA; it includes the delay of multiplexers and the wire delay. In addition, in every block, compared with the original FAs CSA

Proposed $(p+q)$ bits fast accumulation logic

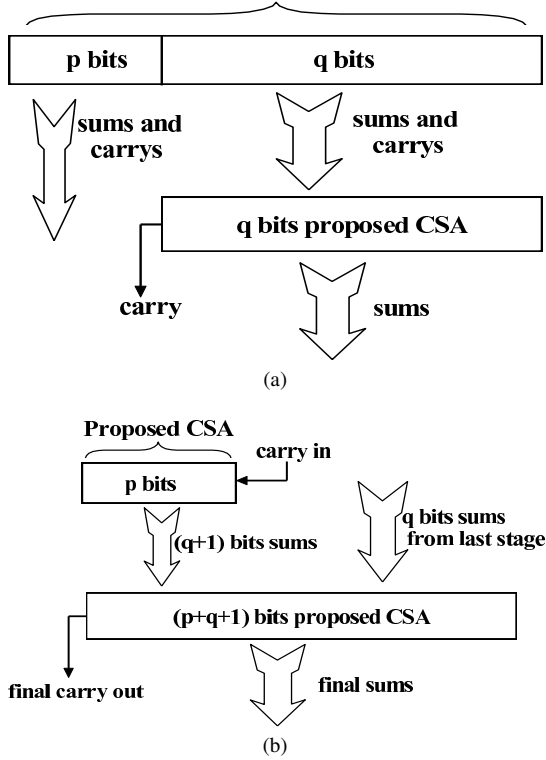


Fig. 8. Adders used in the last two stages

TABLE I
PROPOSED 20-BIT CARRY-SELECT ADDER WITH 5 BLOCKS

Block number	1	2	3	4	5	Total
Bit length of RCA	2	3	4	5	6	20
# of transistors	42	128	172	220	264	826
Delay	3	5.5	7.5	9.5	11.5	11.5

The final results unavoidably contain errors, because the exact mathematical result of an arithmetic operation often has more significant bits than can fit in its destination format. To deal with this, the result is rounded to fit the format, by choosing a nearest representable number in that format. If the result lies precisely halfway between two representable numbers, the one whose least significant bit is 0 is used. The all-one finding circuit in Fig. 6 is also used in the rounding circuit to accelerate the speed.

The last stage of the pipeline is the critical path, where its delay is 0.76 ns, including the 20-bit adder, the rounding circuit, clock skew and setup time. This means that the FIR filter can operate at 1.3 GHz without difficulties using the same CMOS technology. Our filter reduces the critical path from a multiply-accumulate operation to almost a single adder operation. Compared to the conventional FIR filter mentioned in Fig. 2, the proposed FIR filter requires a 26.8% smaller area under the same conditions.

VI. CONCLUSION

In this paper, the implementation of the high-performance 8-tap FIR filter using the logarithmic number system has been

presented. The main advantage of the logarithmic number system is that multiplication is implemented by addition of the logarithms, which is an exact operation (provided no overflow occurs). We have designed special-purpose architectures specifically for an FIR filter by using specialized adders in different stages, conversion circuits, careful sequencing, and pipelining. The number of taps does not affect the delay of a filter, and the critical path is approximately equivalent to the delay of a single adder. This system can operate at a higher speed with a smaller area. In the simulation, the FIR filter is fabricated using the SMIC 0.13 μm CMOS technology, and operates at 1.3 GHz with 1.2 V power supply. Area savings surpass the overhead incurred by the conversion between the logarithmic and traditional number systems. This approach can be easily extended to filters with more taps.

REFERENCES

- [1] 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, Mar. 1993.
- [2] J. R. Choi, S. W. Jeong, L. H. Jang, and J. H. Choi, "Structured design of a 288-tap FIR filter by optimized partial product tree compression," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 3, pp. 468–476, Mar. 1997.
- [3] D. J. Pearson, S. K. Reynolds, A. C. Megdanis, S. Gowda, K. R. Wrenner, M. Immediato, R. L. Galbraith, and H. J. Shin, "Digital FIR filters for high speed PRML disk read channels," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 12, pp. 1517–1523, Dec. 1995.
- [4] W. L. Abbott, H. C. Nguyen, B. N. Kuo, K. M. Ovens, Y. Wong, and J. Casasanta, "A digital chip with adaptive equalizer for PRML detection in hard-disk drives," in *Digest of Technical Papers of the 41st IEEE International Solid-State Circuits Conference*, Feb. 1994.
- [5] 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, Nov. 1997.
- [6] G. B. Balaji, K. Balaji, H. Sundararaman, A. Naveen, and K. R. Santha, "Memory reduction techniques for logarithmic number system," in *Proceedings of International Conference on Signal Processing, Communications and Networking*, Feb. 2007.
- [7] J. Detrey and F. de Dinechin, "A VHDL library of LNS operators," in *Conference Record of the 37th Asilomar Conference on Signals, Systems and Computers*, Nov. 2003.
- [8] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed. A K Peters, 2002.
- [9] Y. Wan and C. L. Wey, "Efficient algorithms for binary logarithmic conversion and addition," *IEE Proceedings—Computer and Digital Techniques*, vol. 146, no. 3, pp. 168–172, May 1999.
- [10] S.-H. Baik, K.-N. Han, and E. Yoon, "A 230 MHz 8 tap programmable FIR filter using redundant binary number system," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Jul. 1999.
- [11] N. H. E. Weste and K. Eshraghian, *Principle of CMOS VLSI designs: A system perspective*, 2nd ed. Addison Wesley, 1994.
- [12] B. Amelifard, F. Fallah, and M. Pedram, "Closing the gap between carry select adder and ripple carry adder: A new class of low-power high-performance adders," in *Proceedings of the 6th International Symposium on Quality of Electronic Design*, Mar. 2005.
- [13] Y. He, C.-H. Chang, and J. Gu, "An area efficient 64-bit square root carry-select adder for low power applications," in *Proceedings of IEEE International Symposium on Circuits and Systems*, May 2005.
- [14] A. Nève, H. Schettler, T. Ludwig, and D. Flandre, "Power-delay product minimization in high-performance 64-bit carry-select adders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 3, pp. 235–244, Mar. 2004.
- [15] T.-Y. Chang and M.-J. Hsiao, "Carry-select adder using single ripple-carry adder," *Electronics Letters*, vol. 34, no. 22, pp. 2101–2103, Oct. 1998.
- [16] Y. Sun, X. Zhang, and X. Jin, "High-performance carry select adder using fast all-one finding logic," in *Proceedings of the 2nd Asia International Conference on Modeling & Simulation*, May 2008.