New Multiplier for a Double-Base Number System Linked to a Flash ADC

Minh Son Nguyen, Insoo Kim, Kyusun Choi, Jaehyun Lim, Wonho Choi, and Jongsoo Kim

The double-base number system has been used in digital signal processing systems for over a decade because of its fast inner product operation and low hardware complexity. This letter proposes an innovative multiplier architecture using hybrid operands. The multiplier can easily be linked to flash analog-to-digital converters or digital systems through a double-base number encoder (DBNE) for realtime signal processing. The design of the DBNE and the multiplier enable faster digital signal processing and require less hardware resources compared to the binary processing method.

Keywords: DBNS, double-base number encoder, flash ADC.

I. Introduction

There have been numerous efforts to design both fast digital signal processors (DSPs) and analog-to-digital converters (ADCs). Alternative number systems, including the double-base number system (DBNS), are often suggested to reduce the complexity of the arithmetic operations because multiplication, which is the dominant operation of DSP, is performed less efficiently in a binary number system. The effectiveness of the DBNS has already been demonstrated in the design of fast DSP systems [1]-[4]. When there is only a single DBNS term, the multiplication can be done simply by adding the exponents. However, there are usually many DBNS terms, so an algorithm was developed to find the canonic DBNS.

Meanwhile, the flash ADC is well known because it has the fastest conversion speed among existing ADCs. The flash ADC block consists of comparators, gain boosters, 0-1 generators, and an encoder [5]. However, the flash ADC is still unable to satisfy the realtime signal processing requirement using DBNS-based DSPs because of the conversion speed and binary data representation. Therefore, we previously proposed a double-base number encoder (DBNE) that converts the thermometer code of a flash ADC into the DBNS format to integrate a flash ADC and DBNS-based DSP on a single SoC architecture [6]. Figure 1 shows a conceptual diagram of the threshold inverter quantization ADC [7] connected to a DBNS finite impulse response (FIR) filter. The digitized signals in the DBNS are fed to the DBNS FIR filter.

However, the hardware implementation of the DBNS FIR filter is still complicated, even though it is the canonic DBNS. Thus, we propose an innovative fast multiplier that adopts hybrid data types: One operand uses the DBNS format supplied by the ADC, while the filter coefficients are represented by the IEEE 32-bit floating-point number system (FPNS) to maintain system compatibility, as shown in Fig. 1.

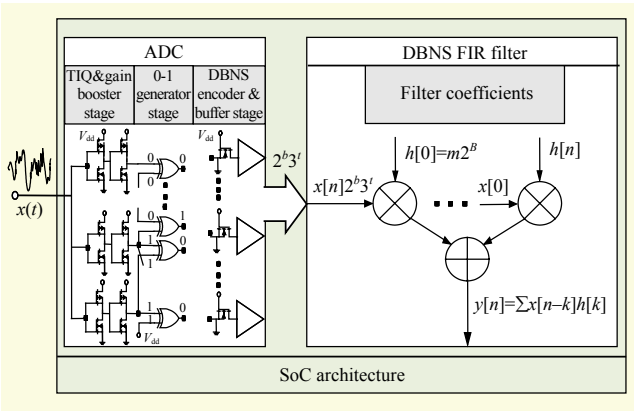

Fig. 1. Flash ADC and digital filter operation.

Manuscript received May 13, 2011; revised Aug. 3, 2011; accepted Aug. 17, 2011.
This research was supported by the University of Ulsan research grants in 2010.
Minh Son Nguyen (phone: +84 16 7998 2448, nmson@hcmiu.edu.vn) is with the Department of Computer Science and Engineering, Vietnam National University HCMC, Ho Chi Minh, Vietnam.
Insoo Kim (insoo@psu.edu) and Kyusun Choi (kyusun@cse.psu.edu) are with the Department of Computer Science and Engineering, Pennsylvania State University, PA, USA.
Jaehyun Lim (lim.jaehyun@gmail.com) is with Samsung Electronics, Rep. of Korea.
Wonho Choi (whchoi@ulsan.ac.kr) and Jongsoo Kim (corresponding author, jongsoo@ulsan.ac.kr) are with the Department of Electrical Engineering, University of Ulsan, Ulsan, Rep. of Korea.
http://dx.doi.org/10.4218/etrij.12.0211.0198

II. DBNS with Index Calculus

In [4], a new redundant DBNS used to represent any real number X within an error tolerance ε was proposed as

$$\left| X - \sum_{k} s_k 2^{b_k} 3^{t_k} \right| < \varepsilon, \quad (1)$$

where s_k represents the sign of the double-base number, while b_k and t_k are the exponents of the binary and ternary bases, respectively. For example, the integer 5 can be represented by $5=2^1 3^0+2^0 3^1$ or $5=2^0 3^0+2^2 3^0$. In general, k is larger than 1 to represent a given real number within a small error. Therefore, array architecture is required to support general DBNS arithmetic operations. However, if k is fixed, such as $k=1$, which indicates a single term representation, the bit sizes of b_k and t_k must be sufficiently large to represent the number X within a reasonable non-zero value of ε.

In any ADC, due to the quantization operation, there are always conversion errors that must be less than 0.5 LSB. Therefore, it is possible to match the value ε to the ADC's conversion error if the value ε is smaller than the conversion error. X can then be represented as a single term, $X \approx s 2^b 3^t$, with finite precision [7].

The typical FIR filter can be rewritten as

$$\begin{aligned} y(n) &= \sum_{k=0}^{N-1} h(k)x(n-k) = \sum_{k=1}^{N-1} h(k)x(n-k) + h(0)x(n) \\ &= \sum_{k=1}^{N-1} h(k)x(n-k) + y(0), \end{aligned} \quad (2)$$

where $x(n)$ and $h(n)$ represent the digital signal and filter coefficient, respectively [8]. This filter equation can be rewritten in DBNS notation as

$$y(n) = m2^B \cdot 2^b 3^t + y(n-1). \quad (3)$$

The 23-bit mantissa m can be represented by $m=1, f=1+f$, and the fraction f can be expressed by $f=0.f_2 f_{21}\ldots f_1 f_0 \in [0, 1)$, $f_i = \{0, 1\}$, while the 8-bit exponents B, b, and $t \in [-127, 127)$.

III. Proposed Multiplier Architecture

The multiplication of $T=m2^B \cdot 2^b 3^t$ in (3) is a critical operation to be processed in FIR filters. In the T expression, if the mantissa m of the filter coefficient and ternary 3^t of the digitized signal can be converted into a 2^E format, the multiplication becomes simple. The filter coefficients are in the range of [0, 1), and the mantissa m is in the range of [1, 2). The nonlinear value of 2^f is always less than the linear value of $m=1+f$, where $m=1, f=1+f$, and $f \in [0, 1)$, in the given ranges as shown in Fig. 2. Except for the $m=1$ and $m=2$ values, a filter

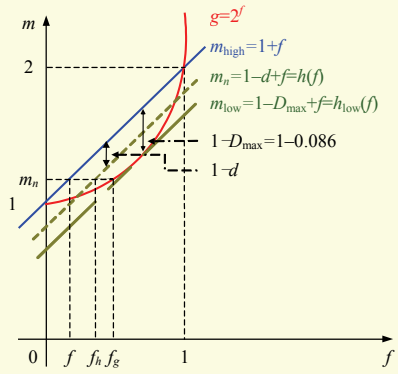

Fig. 2. Approximation based on nonlinear and linear functions.

coefficient can be converted into the exponential format by mapping the value m onto the graph g. However, the hardware implementation for finding f_g is impractically complicated. Therefore, a polynomial approximation method can be applied to find the corresponding exponent value that is dividend into the deviation constant d and error limitation value δ, if $m=1+f$ is almost equal to $2^{f+d+\delta}$ within an allowable error range. The maximum error D_{max} between the two functions is equal to 0.086 [9], [10]. A small value e_D is defined as the new maximum error for obtaining $m=1+f \approx (1+e_D)\, 2^{f+d+\delta}$. Thus, if N partitions are used for the polynomial approximation, the maximum error is defined to be $e_D=D_{max}/N$, and the multiplication in (3) can be rearranged as

$$T = m2^B \cdot 2^b 3^t \cong (1+e_D) 2^{f+d+\delta} 2^B \cdot 2^b 3^t, \text{ where } \delta << e_D. \quad (4)$$

The basic algorithm for calculating d and δ can be expressed as follows:

```
MAX_RES=a large integer constant
find d0 = Dmax / partition N
for i = 1 to N
   find intersection point fh of linear mn and nonlinear g
   for each fh
      find errors eDj while fh is in range of [fh–d0/2, fh+d0/2]
      by every resolution d0/MAX_RES to make mhigh ≈ g
/*    error limitation value δi is equal to resolution and
      j is the number of intersection points                  */
find maximum eD from the error set {eDj}.
```

Now, the mantissa m is converted into 2^E format with a tolerable error. The error e_D can be easily calculated using (4):

$$[(m/2^{f+d+\delta}) - 1] = e_D. \quad (5)$$

Next, convert the ternary 3^t into the binary format to simplify the multiplication. Since $3^t = 2^{t\log_2(3)}$, the exponent t of the ternary number 3^t can be rearranged such that

$$2^{t\cdot\log_2(3)} = 2^{t\cdot(2^0+2^{-1}+2^{-4}+2^{-6}+2^{-8}+2^{-9}+2^{-10}+2^{-20}+2^{-21}+2^{-23}+2^{-40}\cdots)}. \quad (6)$$

It is possible to sort (6) into the integer and fraction parts

depending on t as (7) and (8). When the series value in (6) is an integer, division is indicated. Thus, it can be processed by right shifting the exponent, while a fraction corresponds to a right shifting of the mantissa. In the 32-bit FPNS system, $t\in[-127, 127)$ and the mantissa are 23 bits. Therefore, the integer indices require only {0, 1, 4, 6}, and the fraction index requires up to the 23rd position.

$$I=t+int\left[t\cdot 2^{-1}\right]+int\left[t\cdot 2^{-4}\right]+int\left[t\cdot 2^{-6}\right]=\sum_{i=\{0,1,4,6\}} int\left[t\cdot 2^{-i}\right], \tag{7}$$

$$\begin{aligned} F=&frac\left[t\cdot 2^{-1}\right]+frac\left[t\cdot 2^{-4}\right]+frac\left[t\cdot 2^{-6}\right]+frac\left[t\cdot 2^{-8}\right]\\ &+frac\left[t\cdot 2^{-9}\right]+frac\left[t\cdot 2^{-10}\right]+frac\left[t\cdot 2^{-20}\right]\\ &+frac\left[t\cdot 2^{-21}\right]+frac\left[t\cdot 2^{-23}\right]\\ =&\sum_{i=\{1,4,6,8,9,10,20,21,23\}} frac\left[t\cdot 2^{-i}\right]. \end{aligned} \tag{8}$$

Now, the ternary base number can be converted as $3^t=2^I\cdot 2^F$, where $I\in[-127, 127)$ and $F\in[0, 1)$. The fraction and integer of $[t\cdot(2^{-i})]$ are simply obtained by shift operations. If one substitutes I and F into (4), then T_1 becomes

$$T_1=(1+e_D)2^{f+d+F+\delta}\cdot 2^{B+b+I}. \tag{9}$$

When applying the polynomial approximation with a sufficient number of N partitions to revert the nonlinear function $(1+e_D)2^{f+d+F+\delta}$ back to a linear function, T_2 can be obtained as

$$T_2=\left(1+f+F\right)\cdot 2^{B+b+I} \text{ and } (1+f+F)\geq 2^{(f+d+F+\delta)}. \tag{10}$$

When the partition number N is enough, T_1 and T_2 are almost equal, and e_D is approximated as

$$[1-(2^{f+d+F+\delta}/1+f+F)]=e_D. \tag{11}$$

T_2 is now in the new floating-point format $M2^E$, which completed the multiplication of different base number systems. The mantissa M is a binary addition of $(1+f+F)$, and the exponent E is a binary addition of $(B+b+I)$. Thus, the expressions I and F can be processed by binary adders and shifters.

The error ratio E_{mul} of the proposed multiplication can be determined by

$$E_{mul}=(T/T_2). \tag{12}$$

From (4) and (12),

$$E_{mul}=\left(\frac{T}{T_1}\cdot\frac{T_1}{T_2}\right)=\left(\frac{m}{2^{f+d+\delta}}\cdot\frac{2^{f+d+F+\delta}}{1+f+F}\right)=(1-e_D^2).$$

Thus,

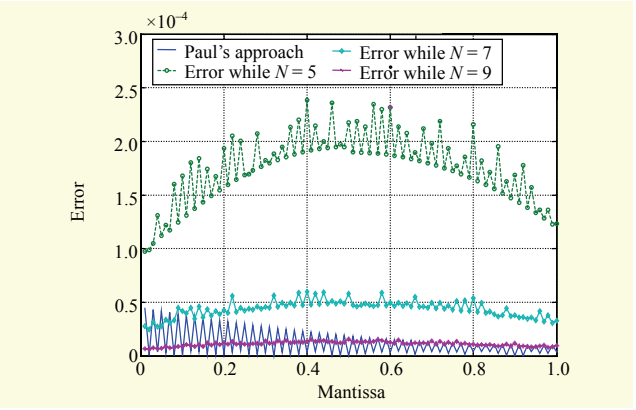

Fig. 3. Relationship between error and partition number N.

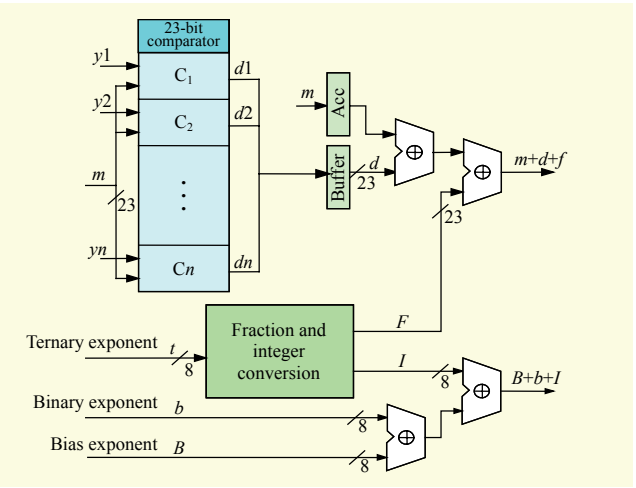

Fig. 4. Multiplication and conversion to floating point.

$$E_{mul}=\left(1-0.007396/N^2\right). \tag{13}$$

Equation (13) shows that the error ratio of the proposed multiplication depends on the partition value N. The error of the proposed circuit is shown in Fig. 3. The error of the proposed method with N=7 is closest to the results in [9].

Figure 4 shows the logic circuits of the proposed multiplier. The 23-bit comparators compare the value of the mantissa $m=1+f$ with a known constant Y, which is calculated from the linear and nonlinear functions. The "fraction and integer conversion" block performs (7) and (8), which only consists of shifters/adders. The block requires eight binary right shifts and two addition stages to generate a fraction part. Three shifters and two addition stages are required for an integer part. The delay time of the proposed multiplier is one stage of the parallel shifters and four stages of 23-bit addition only.

IV. Simulation and Experiment Results

Table 1 shows a comparison of the performances of the

Table 1. Layout results of 6-bit encoder in ADC.

	DBNE	Binary ROM encoder	Fat tree encoder
Technology	CMOS 0.18 μm	CMOS 0.18 μm	CMOS 0.18 μm
Max. speed	3.3 GHz	1.11 GHz	2.00 GHz
Avg. power	11.12 mW	32.64 mW	22.70 mW
Max power	17.21 mW	54.41 mW	38.65 mW
Area	0.0161 mm^2	0.0094 mm^2	0.0074 mm^2

Table 2. Comparison of DBNS and FPNS multiplication.

	FPNS multiplication	DBNS multiplication
Number of gates	3,965	2,134
Delay time	34.93 ns	15.27 ns

DBNE, ROM, and FAT tree encoders. Although the area of the DBNE is 2.2 times larger than that of the smallest encoder, the DBNE is faster and consumes less power than the other encoder types. The speed improvement results from the small load capacitance effect. The chip area of the proposed multiplier is 0.1408 mm^2, and its critical delay time is 9.5 ns.

In this letter, we used the Xilinx ISE9.1 and Verilog hardware description language to verify the functionality of the proposed design with N=7 and FPNS multiplication. The FPNS multiplication needs extra control logic circuits to generate a final output excluding the arithmetic array unit. Also, the performance depends on the structure and implementation technology. Alternatively, the proposed multiplier requires data conversion overhead in an existing digital system. However, there is no such conversion overhead in an ADC application because the DBNE is faster than other binary encoders as shown in Table 1. The two multipliers' architectures are completely different. Thus, a performance comparison at the transistor level between the two multipliers was not done.

Table 2 shows the emulation results of the two multipliers with the Xilinx Spartan-3 FPGA chip. The proposed multiplier requires 45% fewer gates and is 44% faster due to the simple structure. The speed improvement in the Spartan-3 FPGA chip will decrease when the systolic array multiplier, which is designed with carry-lookahead adders and a fast parallel structure, is used in the FPNS multiplication.

V. Conclusion

This letter proposed new multiplication architecture. The multiplier circuit consists only of comparators, adders, and shifters. The proposed design reduces the required hardware resources by 45% and improves the speed by 44% compared to the FPNS method in a Spartan-3 FPGA board. Even though the proposed DBNS coding suffers from accuracy issues due to the logarithm characteristics, it can be successfully used with the ADC output because quantization conversion errors always arise. Unlike the ROM type architecture in [9], the proposed multiplier algorithm can easily be designed by using basic logic gates. Furthermore, the partition number N does not affect the delay time due to its simple structure; the delay time only depends on the buffer conditions. The 64-bit version of the IEEE FPNS employs an 11-bit exponent and a 53-bit mantissa. There is no exponent operation change except for a slight increment in the fraction. The order of complexity is less than O(log(n)). Therefore, the proposed multiplier's delay time and chip area both minimally increase in IEEE 64-bit FPNS. Future works will include a power and speed comparison and an error analysis.

References

[1] R. Kacem et al., "Low Power Implementation of Digital Filters Using DBNS Representation and Sub-expression Sharing," *2nd Int. Conf. Signals, Circuits, Syst.*, Nov. 2008, pp. 1-6.

[2] S.-C. Huang and L.-G. Chen, "A 32-Bit Logarithmic Number System Processor," *J. VLSI Signal Process.*, vol. 14, no. 3, 1996, pp. 311-319.

[3] V.S. Dimitrov, G.A. Jullien, and W.C. Miller, "Theory and Applications of the Double-Base Number System," *IEEE Trans. Comput.*, vol. 48, no. 10, 1999, pp. 1098-1106.

[4] V.S. Dimitrov and G.A. Jullien, "Loading the Bases: A New Number Representation with Applications," *IEEE Circuits Syst. Mag.*, vol. 3, no. 2, 2003, pp. 6-23.

[5] I. Kim et al., "Highly Efficient Comparator Design Automation for TIO Flash A/D Converter," *IEICE Trans. Fundamentals of Electron. Commun. Comput. Sci.*, vol. E91-A, no. 12, Dec. 2008.

[6] J. Lim et al., "Low Power Flash A/D Converter with TIQ Comparators for Multi-standard Mobile Applications," *IREE*, vol. 4, Dec. 2009, pp. 1447-1452.

[7] M.S. Nguyen et al., "Algorithm and Design of Double-Base Log Encoder for Flash A/D Converter," *KISPS*, vol. 10, no. 4, Oct. 2009, pp. 289-293.

[8] H.-J. Kang and I.-C. Park, "FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders," *IEEE Trans. Circuits Syst.*, vol. 48, Aug. 2001, pp. 770-778.

[9] S. Paul, N. Jayakumar, and S.P. Khatri, "A Fast Hardware Approach for Approximate, Efficient Logarithm and Antilogarithm Computations," *IEEE Trans. VLSI Syst.*, vol. 17, no. 2, Feb. 2009, pp. 269-277.

[10] M. Gok, "A Novel IEEE Rounding Algorithm for High-Speed Floating-Point Multipliers," *VLSI J.*, vol. 40, no. 4, 2007, pp. 549-560.