ELSEVIER

Journal of Systems Architecture 45 (1999) 1195–1207

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

A floating point multiplier performing IEEE rounding and addition in parallel

Woo-Chan Park [1], Tack-Don Han [2], Shin-Dug Kim [*], Sung-Bong Yang [3]

Parallel Processing Systems Laboratory, Department of Computer Science, Yonsei University, 134 Shinchondong Seodaemun-Ku, Seoul 120-749, South Korea

Received 15 January 1997; received in revised form 3 November 1997; accepted 19 February 1998

Abstract

In the conventional floating point multipliers, the rounding stage is usually constructed by using a high speed adder for the increment operation, increasing the overall execution time and occupying a large amount of chip area. Furthermore, it may accompany additional execution time and hardware components for renormalization which may occur by an overflow from the rounding operation. A floating-point multiplier performing addition and IEEE rounding in parallel is designed by optimizing the operational flow based on the characteristics of floating point multiplication operation. A hardware model for the floating point multiplier is proposed and its operational model is algebraically analyzed in this research. The floating point multiplier proposed does not require any additional execution time nor any high speed adder for rounding operation. In addition, the renormalization step is not required because the rounding step is performed prior to the normalization operation. Thus, performance improvement and cost-effective design can be achieved by this approach. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Floating point unit; Floating-point multiplier; IEEE rounding; Carry select adder; Computer arithmetic

1. Introduction

An FPU (floating point unit) is the principal component in graphics accelerators, DSPs (digital signal processor), and high performance computer systems. As the chip integration density increases due to advance in semiconductor technology, it has become possible for an FPU to be placed on a

[*] Corresponding author. E-mail: sdkim@kurene.yonsei.ac.kr
[1] E-mail: chan@kurene.yonsei.ac.kr
[2] E-mail: hantack@kurene.yonsei.ac.kr
[3] E-mail: yang@kurene.yonsei.ac.kr

single chip together with an integer unit, allowing the FPU to exceed its original supplementary function and becoming the principal element of the CPU (central processing unit) [1,2]. In this case, a compromise between the chip area available and the FPU functions included should be considered. In general, only some primary arithmetic units such as an adder/subtractor and a multiplier are integrated on a chip and additional software handling is required for further complete floating point operations. Therefore, the overall floating point operations are greatly affected by how the floating point multiplier and the adder/subtractor are designed.

In this research, an architectural model for the floating point multiplier is only considered and its operational characteristics are verified. In general, processing flow of the conventional floating-point multiplication consists of either multiplication, addition, normalization, and rounding stages [2–4], or multiplication, addition, rounding, and normalization stages [5–7]. To process a rounding operation, either a high speed incrementor or an adder is required. Furthermore, the former needs additional hardware components for renormalization when an overflow has occurred due to rounding, while the latter needs additional hardware components to process the rounding operation prior to the normalization.

In this approach, a floating point multiplier which performs rounding and addition in parallel is proposed and algebraically analyzed. The proposed floating point multiplier needs a few additional simple hardware components to execute both the rounding and the addition in parallel, and does not require any hardware for rounding and renormalization. And thus, it can execute a floating point multiplication in just three stages, resulting in fast execution time. Hence, performance improvement in floating point multiplication operations can be achieved and only reasonable chip area is required in this approach. Since the IEEE standard 754 [8] has been published, almost all the FPUs follow the IEEE standard 754. Rounding operation in this approach is considered to be evaluated for all four IEEE standard modes.

In [9], a structure of a floating point adder/subtractor performing rounding and addition/subtraction in parallel is presented. In [10–13], a floating point multiplier performing rounding and addition in parallel is discussed. However, [10] supports only round-to-zero mode, which is the easiest case to implement. [11] presents a hardware model for a special case of round-to-nearest mode, while [12] presents a general hardware model that can support all four IEEE rounding modes. The approach in [13], which is adapted in designing the Ultra Sparc RISC processor, implements the floating point multiplier supporting all four IEEE rounding modes and performing rounding and addition in parallel. However, [13] requires a complex processing algorithm comparing with that of our approach. This in turn requires additional hardware components that is to increase the length of the critical path on the pipeline stages. This causes one more pipeline delay in their approach [13] and requires additional hardware components comparing with our approach.

In this approach, an optimal hardware model is presented and implemented by modifying the general hardware model presented as in [12], and its operational model is algebraically analyzed. This analysis can be used as a basis to analyze and evaluate a floating point operational model for any given floating point architecture. Refs. [11,12] do not prove the correctness of their implementations, but the correctness of the proposed floating point multiplier is proved as illustrated in Section 3.5. Also, the proposed one has negligible delay for a *selector*, which selects a proper result after performing addition and rounding in parallel, by effectively choosing a *predictor*, which inserts the predicted value to the empty position before carry propagation addition. Logic components required for the selector and the predictor are determined via algebraic analysis.

Section 2 presents a brief overview of the IEEE rounding methods and illustrates the operational characteristics existing among the multiplication, addition, and rounding stages. Section 3 suggests a hardware model which can execute rounding and addition in parallel and its implementation with respect to those four rounding modes. In Section 4, conclusions are made.

2. Parallel execution of IEEE rounding and addition operations

In this section, IEEE rounding methods are discussed and the operational characteristics in performing rounding and addition in parallel are illustrated. The operational characteristics extracted will be utilized to design a hardware model in later section.

2.1. IEEE rounding methods

A normalized floating point number according to the IEEE's standard is expressed as $A = (-1)^{s} \times 1.f \times 2^{\text{e-bias}}$, where s denotes the sign bit for a fraction, f denotes the fraction expressed in the form of absolute values, and e is used to denote the biased exponent. Note that $1.f$ in the above equation is called the significand.

Suppose that A and B are the significands of any two floating point numbers without any sign bit and of length n bits. In the multiplication stage, two normalized significands A and B are multiplied by using Booth encoder and reduction structure such as array or Wallace tree [4]. Then, the $2n$-bit summation, $S = s_{2n-1}s_{2n-2}\cdots s_0$, and the $2n$-bit carry, $C = c_{2n-1}c_{2n-2}\cdots c_0$, are generated. And then, at the addition stage, $F = C + S = f_{2n-1}f_{2n-2}\cdots f_0$ is computed as the result of the multiplication. The result F is $2n$ bits wide and can be divided into the most significant $(n+1)$-bit field and the least significant $(n-1)$-bit field. Here, the least significant $(n-1)$ bits are used as the primary information for proper rounding operation and these $(n-1)$ bits are represented as the round bit R and sticky bit Sy [11,12]. The R becomes the MSB (most significant bit) of the least significant $(n-1)$ bits and Sy is the boolean ORed value of all the remaining least significant $(n-2)$ bits.

As previously mentioned, the significand number is represented as $1.f$ if the fraction part is f. But, to simplify the notation, the most significant $(n+1)$ bits are considered as the integer portion and the least significant $(n-1)$ bits, which are used to represent the R and Sy bits, are as the fraction portion. The integer portion is represented by using the subscript I and the fractional portion by using the subscript T. Fig. 1 shows that the most significant $(n+1)$ bits of F are the integer portion F_I, while the least significant $(n-1)$ bits are the fractional portion F_T. Then, F can be represented as follows:

$$\begin{aligned} F &= A \times B = C + S \\ &= (c_{2n-1}c_{2n-2}\ldots c_{n-1}.c_{n-2}\ldots c_0) \\ &\quad + (s_{2n-1}s_{2n-2}\ldots s_{n-1}.s_{n-2}\ldots s_0), \\ &= f_{2n-1}f_{2n-2}\cdots f_{n-1}.f_{n-2}\cdots f_0 \\ &= f_{2n-1}f_{2n-2}\ldots f_{n-1}.RSy. \end{aligned} \tag{1}$$

IEEE standard 754 stipulates four different rounding modes, which are the round-to-nearest, round-to-zero, round-to-positive-infinity, and the round-to-negative-infinity modes. These four rounding modes can be classified mainly into the round-to-nearest, round-to-zero, and the round-

$$F = \overbrace{f_{2n-1}f_{2n-2}\cdots f_{n-1}}^{F_I} \cdot \overbrace{f_{n-2}\cdots f_1 f_0}^{F_T}$$

Fig. 1. F_I and F_T definition.

to-infinity modes, because round-to-positive-infinity and round-to-negative-infinity modes can be considered as the round-to-zero and round-to-infinity modes according to the sign bit of a number. Following three algorithms are the results of rounding operation with LSB, R, and Sy which are generated after multiplication, addition, and normalization stages. The LSB is the least significant bit of the significand portion. Return "0" means the truncation, and return "1" means the incrementation as the result of any rounding operation.

```
Algorithm 1. $\mathrm{Round_{Nearest}}(\mathrm{LSB}, R, Sy)$
    if ($R = 0$) return 0
    else if ($Sy = 1$) return 1
     else if ($\mathrm{LSB} = 0$) return 0
       else return 1
```

```
Algorithm 2. $\mathrm{Round_{Zero}}(\mathrm{LSB}, R, Sy)$
    return 0
```

```
Algorithm 3. $\mathrm{Round_{Infinity}}(\mathrm{LSB}, R, Sy)$
    if (($R = 1$) or ($Sy = 1$)) return 1
    else return 0
```

2.2. Multiplication, addition, and rounding

As from Eq. (1), shifting operation for normalization depends on the most significant bit of F, i.e., the value of f_{2n-1}. If $f_{2n-1} = 0$, then any bit shifting operation for normalization is not required, but if $f_{2n-1} = 1$ then one bit shifting to the right should be performed in the normalization stage. The former case is denoted as NS (no shift) and the latter case is denoted as RS (right shift). X denotes the do not care condition. If any overflow is generated from the execution result of an arbitrary operation, e.g., Z, then *overflow*(Z) returns 1, otherwise it returns 0. The symbol "$\wedge$" denotes a boolean AND operation, "$\vee$" denotes a boolean OR operation, and "$\oplus$" represents a boolean XOR operation.

According to [11,12], the sticky bit Sy is equal to zero, i.e., $Sy = 0$ if the summation of the numbers of trailing-zeros for the significands of any two floating point numbers, e.g., A and B, is greater than or equal to $n-2$, while $Sy = 1$ if it is smaller than $n-2$. Thus, Sy can be determined in the multiplication stage because it can be computed in parallel with multiplication operation and the low order $(n-2)$ bits of F are used only to generate the sticky bit. Accordingly, the low order $(n-2)$ bits of C and S have no effect on the high order $(n+2)$ bits except a carry generated into the most significant $(n+2)$ bits. This carry signal is denoted as C^{in}_{n-2}. In general, C^{in}_{k} denotes the value of carry signal from the $(k-1)$th bit position into the kth bit position. Given this fact, Eq. (1) can be converted into Eq. (2).

$$\begin{aligned}
F &= C + S,\\
&= (c_{2n-1}c_{2n-2}\cdots c_{n-1}.c_{n-2})\\
&\quad + (s_{2n-1}s_{2n-2}\cdots s_{n-1}.s_{n-2}) + (0.0c_{n-3}\cdots c_0)\\
&\quad + (0.0s_{n-3}\cdots s_0)\\
&= (c_{2n-1}c_{2n-2}\cdots c_{n-1}.c_{n-2})\\
&\quad + (s_{2n-1}s_{2n-2}\cdots s_{n-1}.s_{n-2}) + 0.C^{\mathrm{in}}_{n-2} + 0.0Sy,\\
C^{\mathrm{in}}_{n-2} &= \mathrm{overflow}((c_{n-3}\cdots c_0)\\
&\qquad + (s_{n-3}\cdots s_0)).
\end{aligned} \tag{2}$$

Let D_I be the summation of C_I and S_I, i.e., $D_I = C_I + S_I = d_n d_{n-1}\cdots d_0$. Because F can be represented as integer and fractional parts, Eq. (2) can be converted into Eq. (3).

$$\begin{aligned}
F &= C_I + S_I + 0.c_{n-2} + 0.s_{n-2} + 0.C^{\mathrm{in}}_{n-2} + 0.0Sy,\\
&= C_I + S_I + 0.R\,Sy + C^{\mathrm{in}}_{n-1},\\
&= D_I + C^{\mathrm{in}}_{n-1} + 0.R\,Sy,\\
C^{\mathrm{in}}_{n-1} &= \mathrm{overflow}(C^{\mathrm{in}}_{n-2} + s_{n-2} + c_{n-2}),\\
R &= C^{\mathrm{in}}_{n-2} \oplus s_{n-2} \oplus c_{n-2}.
\end{aligned} \tag{3}$$

According to Eq. (3), F_I can be obtained as the summation of integer parts with a carry and can be represented as follows:

$$F_I = C_I + S_I + C_{n-1}^{\text{in}} = D_I + C_{n-1}^{\text{in}}. \tag{4}$$

Normalization should be accounted for two different cases, i.e., NS and RS cases. In the case of NS, the result value after normalization of F is denoted as P^{NS}. For the case of RS, the result value after normalization of F is denoted as P^{RS}. Then, P_I^{NS} and P_T^{NS} are shown as in Eq. (5).

$$\begin{aligned} P_I^{\text{NS}} &= f_{2n-2}f_{2n-3}\cdots f_{n-1}, \\ P_T^{\text{NS}} &= R\,Sy. \end{aligned} \tag{5}$$

Also, P_I^{RS} and P_T^{RS} are shown as in Eq. (6):

$$\begin{aligned} P_I^{\text{RS}} &= f_{2n-1}f_{2n-2}\cdots f_n, \\ P_T^{\text{RS}} &= f_{n-1}(R \vee Sy). \end{aligned} \tag{6}$$

Suppose that Q is the result value of any rounding operation that is performed prior to the normalization stage. Also, the notation, $\text{Round}_{\text{mode}}(f_{n-1}, R, Sy)$, is used to denote any rounding operation, where the subscript mode represents one of three different rounding modes. Then, in the case of NS, Q is represented by Q^{NS}. Thus, Q^{NS} can be represented as follows according to Eqs. (4) and (5):

$$\begin{aligned} Q^{\text{NS}} &= P_I^{\text{NS}} + \text{Round}_{\text{mode}}(f_{n-1}, R, Sy), \\ &= F_I + \text{Round}_{\text{mode}}(f_{n-1}, R, Sy), \\ &= D_I + C_{n-1}^{\text{in}} + \text{Round}_{\text{mode}}(f_{n-1}, R, Sy). \end{aligned} \tag{7}$$

Also, for the case of RS, Q is represented by Q^{RS} and Q^{RS} can be obtained as follows according to Eqs. (4) and (6). Two cases depending on the value of f_{n-1} should be considered.

If $f_{n-1} = 0$

$$\begin{aligned} Q^{\text{RS}} &= (P_I^{\text{RS}} + \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy)) \times 2, \\ &= ((f_{2n-1}f_{2n-2}\cdots f_n) \\ &\quad + \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy)) \times 2, \\ &= (f_{2n-1}f_{2n-2}\cdots f_n f_{n-1}) + 2 \\ &\quad \times \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy), \\ &= F_I + 2 \times \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy), \\ &= D_I + C_{n-1}^{\text{in}} + 2 \\ &\quad \times \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy). \end{aligned}$$

If $f_{n-1} = 1$

$$\begin{aligned} Q^{\text{RS}} &= (P_I^{\text{RS}} + \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy)) \times 2, \\ &= ((f_{2n-1}f_{2n-2}\ldots f_n) \\ &\quad + \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy)) \times 2, \\ &= (f_{2n-1}f_{2n-2}\ldots f_n f_{n-1}) + (1 \text{ or } 2) \\ &\quad \times \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy), \\ &= F_I + (1 \text{ or } 2) \times \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy), \\ &= D_I + C_{n-1}^{\text{in}} + (1 \text{ or } 2) \\ &\quad \times \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy). \end{aligned}$$

If $f_{n-1} = 0$ and $\text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy) = 1$, then $Q^{\text{RS}} = (f_{2n-1}f_{2n-2}\cdots f_{n-1}) + 2$. However, if $f_{n-1} = 1$ and $\text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy) = 1$, then $Q^{\text{RS}} = (f_{2n-1}f_{2n-2}\cdots f_{n-1}) + (1 \text{ or } 2)$ because $(f_{2n-1}f_{2n-2}\cdots f_{n-1}) + 1$ and $(f_{2n-1}f_{2n-2}\cdots f_{n-1}) + 2$ generate identical most significant $(n+1)$ bits of Q^{RS}, which represents the effective final significand value after normalization by one bit shifting to the right. Especially, in the round-to-nearest mode, f_{n-1} must be one so that rounding result is obtained by incrementation. But, in the round-to-infinity mode, if $R \vee Sy = 1$, then rounding result is obtained by incrementation regardless of the value of f_{n-1}. Thus, the above equation can be converted into Eq. (8) to be used in Section 3.4.

If $f_{n-1} = 1$

$$\begin{aligned} Q^{\text{RS}} &= F_I + \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy), \\ &= D_I + C_{n-1}^{\text{in}} + \text{Round}_{\text{mode}}(f_n, f_{n-1}, R \vee Sy). \end{aligned}$$

If $f_{n-1} = X$

$$Q^{\mathrm{RS}} = F_I + 2 \times \mathrm{Round}_{\mathrm{mode}}(f_n, f_{n-1}, R \vee Sy),$$
$$= D_I + C^{\mathrm{in}}_{n-1} + 2 \times \mathrm{Round}_{\mathrm{mode}}(f_n, f_{n-1}, R \vee Sy). \tag{8}$$

3. Hardware model to perform IEEE rounding and addition in parallel and its implementation

In this section, a hardware model is presented to execute IEEE rounding and addition operations in parallel. Its construction is implemented with respect to round-to-nearest, round-to-zero, and round-to-infinity modes.

3.1. Hardware model

A hardware model capable of performing rounding and addition in parallel is designed as shown in Fig. 2. It is constructed as an n-bit half adder, a one-bit full adder (FA), a predictor, a carry select adder, a C^{in}_{n-2} generator, and a logic for q^{NS}_0. When C_I and S_I are added in the n-bit half adder and one-bit full adder, the predictor bit is provided to the full adder. Then, the n-bit carry and $(n+1)$-bit sum are generated. Here, the LSB of the sum is represented by L as shown in Fig. 2. These n-bit carry and the most significant n-bit sum are added by a single carry select adder [4] which is represented by the dotted box in Fig. 2

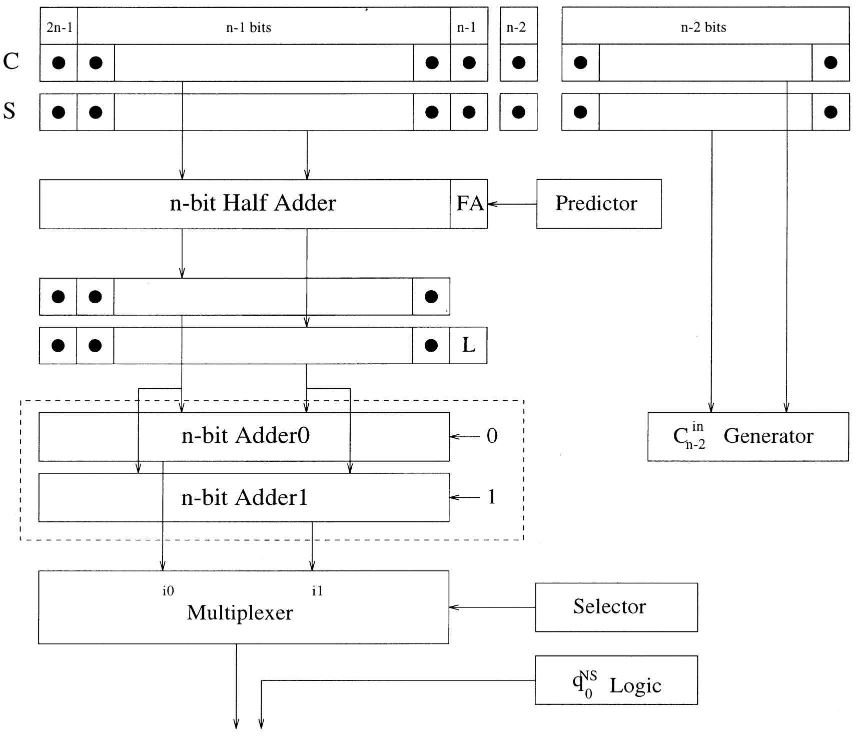

Fig. 2. Hardware model for performing IEEE rounding and addition in parallel.

and can be logically divided into Adder0 and Adder1. A carry select adder is not only adapted to the proposed floating point multiplier, but also has been used in most of the high performance floating point multipliers. The selector selects one of the result values after executing addition and rounding from the two inputs $i0$ and $i1$. If the value of the selector equals to zero, i.e., selector $= 0$, then $i0$ is selected and if selector $= 1$, then $i1$ is selected as the output value of the multiplexer. C^{in}_{n-2} generator is the logic for C^{in}_{n-2} in Eq. (2).

In Fig. 2, the multiplexer output may be one of two values, $D_I +$ predictor and $D_I +$ predictor $+ 2$, according to the value of selector. According to Eqs. (7) and (8), one of four possible cases, i.e., D_I, $D_I + 1$, $D_I + 2$, and $D_I + 3$ needs to be generated to perform rounding and addition in parallel. Therefore, if predictor and selector are properly selected, then the result value after performing addition and rounding in parallel, that is Q, can be generated. To configure the predictor, following two factors should be considered. First, input signals of the predictor must be generated before addition by the carry select adder. Second, the delay by the selector should be negligible enough to be ignored. In [11,12], the selector may be implemented by ROM (read only memory) because the configuration of the predictor makes the selector very complex. Thus, the delay from the selector may be non-negligible and affects the critical path. To design an optimal selector having negligible gate delay, the predictor must be selected very carefully. Selection of predictor and generation of the selector according to the predictor will be discussed for each rounding mode in the following subsections. Because the output of multiplexer is n bits wide, the LSB of Q^{NS} is not generated. Hence, q_0^{NS} is the logic to produce the LSB of Q^{NS}.

Fig. 3 shows the structure of the selector, predictor, and q_0^{NS}. As mentioned previously, four different IEEE rounding modes can be represented

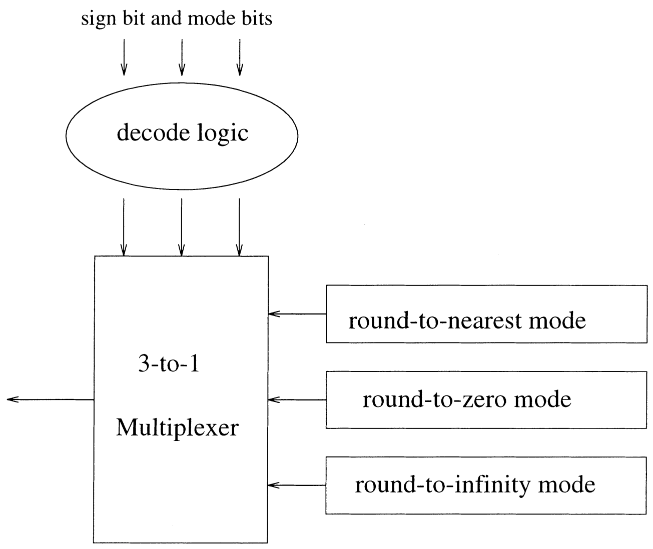

Fig. 3. The structure of the selector, predictor, and q_0^{NS}.

as round-to-nearest mode, round-to-zero mode, and round-to-infinity mode according to the sign of the result value and two mode bits. The multiplexer control inputs of the Fig. 3 is generated by decoding the sign of the result value and two mode bits, and select the result value of the current rounding mode from the result values of the three different rounding modes.

The input value of $i0$ in the multiplexer is denoted by $E = e_n e_{n-1} \cdots e_1$. Because the LSB position of E corresponds to the nth bit position of C and S, the integer value of E is $E_I = E \times 2$. Thus, the $(n+1)$-bit integer field can be denoted as E_I^* and $E_I^* = E_I + L$. Then, E_I^* can be represented as follows:

$$E_I^* = E_I + L = C_I + S_I + \text{predictor} = e_n \ldots e_1 L. \tag{9}$$

Operation of this hardware model is clarified depending on each rounding mode in the following subsections. Also, E_I and E_I^* are used to analyze those functional mechanisms.

The parameters required for selector and q_0^{NS} for each rounding mode are L, e_1, c_{n-2}, s_{n-2}, Sy, e_n, and C_{n-2}^{in}. And, the parameters required for predictor on each rounding mode are c_{n-2} and s_{n-2}. Also, the logics for selector, q_0^{NS}, and predictor can be extracted and provided from Sections 3.2–3.5.

3.2. Round-to-nearest mode

In the round-to-nearest mode and RS case, the value of f_{n-1} must be one so that rounding result is obtained by incrementation. Thus, to perform rounding and addition in parallel, one of three possible cases, i.e., D_I, $D_I + 1$, and $D_I + 2$ needs to be generated according to Eqs. (7) and (8). Then, the predictor can be zero because the most significant n bits of D_I or $D_I + 2$, which are the multiplexer input values, are identical to the most significant n bits of $D_I + 1$ according to the $(n-1)$th bit value of $D_I + 1$. Therefore, when the predictor is zero, D_I, $D_I + 1$, and $D_I + 2$ can be generated. However, the addition of c_{n-2} and s_{n-2} should be considered to generate the value of selector. Thus, to design an efficient structure of the selector, the predictor can be configured as the carry signal from the addition of c_{n-2} and s_{n-2}, i.e., overflow$(c_{n-2} + s_{n-2})$. Therefore, the predictor is given in Eq. (10):

$$\text{predictor} = c_{n-2} \wedge s_{n-2}. \tag{10}$$

Because the predictor of Eq. (10) is identical to the result of overflow$(c_{n-2} + s_{n-2})$, Eq. (2) can be converted into Eq. (11) as follows:

$$\begin{aligned}
F &= C + S,\\
&= (c_{2n-1}c_{2n-2}\ldots c_{n-1}.c_{n-2})\\
&\quad + (s_{2n-1}s_{2n-2}\ldots s_{n-1}\cdot s_{n-2}) + (0.0c_{n-3}\ldots c_0)\\
&\quad + (0.0s_{n-3}\ldots s_0),\\
&= C_I + S_I + 0.c_{n-2} + 0.s_{n-2} + 0.C_{n-2}^{\text{in}} + 0.0Sy,\\
&= C_I + S_I + \text{overflow}(c_{n-2} + s_{n-2})\\
&\quad + 0.(c_{n-2} \oplus s_{n-2}) + 0.C_{n-2}^{\text{in}} + 0.0Sy,\\
&= C_I + S_I + \text{predictor} + 0.(c_{n-2} \oplus s_{n-2})\\
&\quad + 0.C_{n-2}^{\text{in}} + 0.0Sy,\\
&= E_I + L + 0.(c_{n-2} \oplus s_{n-2}) + 0.C_{n-2}^{\text{in}} + 0.0Sy,\\
&= E_I + L + C_{n-1}^{\text{in}} + 0.R + 0.0Sy,\\
&= E_I + 2 \times C_n^{\text{in}} + f_{n-1} + 0.R + 0.0Sy,\\
R &= c_{n-2} \oplus s_{n-2} \oplus C_{n-2}^{\text{in}},\\
C_{n-1}^{\text{in}} &= \text{overflow}((c_{n-2} \oplus s_{n-2}) + C_{n-2}^{\text{in}})\\
&= (c_{n-2} \oplus s_{n-2}) \wedge C_{n-2}^{\text{in}},\\
C_n^{\text{in}} &= \text{overflow}(L + C_{n-1}^{\text{in}}) = L \wedge C_{n-1}^{\text{in}},\\
f_{n-1} &= L \oplus C_{n-1}^{\text{in}},\\
f_n &= e_1 \oplus C_n^{\text{in}}.
\end{aligned} \tag{11}$$

According to Fig. 1, the $(n-2)$th bit position is the first bit position below the binary point, the $(n-1)$th bit position is the first bit position of the integer, and the nth bit position is the second bit position of the integer. Therefore, the real value of $C^{\rm in}_{n-2}$ can be aligned as $0.C^{\rm in}_{n-2}$, the value of $C^{\rm in}_{n-1}$ as $C^{\rm in}_{n-1}$, and the value of $C^{\rm in}_{n}$ as $2 \times C^{\rm in}_{n}$. These interpretations are applied in Eq. (11).

If the value of $C^{\rm in}_{n}$ is equal to one, i.e., $C^{\rm in}_{n} = 1$, L and $C^{\rm in}_{n-1}$ should equal to one, $c_{n-2} \oplus s_{n-2} = 1$, and $C^{\rm in}_{n-2} = 1$. In this case, f_{n-1} and R are to be zero by Eq. (11) and the results of rounding for both NS and RS cases are obtained by truncation according to Algorithm 1. Therefore, the case where the value of $C^{\rm in}_{n}$ is one and the rounding result is obtained by incrementation, does not exist. Consequently, according to Eqs. (7) and (11), $Q^{\rm NS}$ is obtained as follows:

$$\begin{aligned} Q^{\rm NS} &= F_I + {\rm Round}_{\rm Nearest}(f_{n-1}, R, Sy), \\ &= E_I + L + C^{\rm in}_{n-1} \\ &\quad + {\rm Round}_{\rm Nearest}(f_{n-1}, R, Sy), \\ &= E_I + 2 \times C^{\rm in}_{n} + f_{n-1} \\ &\quad + {\rm Round}_{\rm Nearest}(f_{n-1}, R, Sy). \end{aligned} \tag{12}$$

In the case of NS, the selector and $q^{\rm NS}_0$ can be represented as follows according to Eq. (12):

$$\begin{aligned} {\rm selector} &= C^{\rm in}_{n} \vee (f_{n-1} \wedge {\rm Round}_{\rm Nearest}(f_{n-1}, R, Sy)) \\ q^{\rm NS}_0 &= f_{n-1} \oplus {\rm Round}_{\rm Nearest}(f_{n-1}, R, Sy). \end{aligned} \tag{13}$$

Also, if the result of ${\rm Round}_{\rm Nearest}(f_n, f_{n-1}, R \vee Sy)$ is equal to one, then the value of f_{n-1} becomes one. Therefore, $Q^{\rm RS}$ is as follows according to Eqs. (8) and (11):

$$\begin{aligned} Q^{\rm RS} &= F_I + {\rm Round}_{\rm Nearest}(f_n, f_{n-1}, R \vee Sy), \\ &= E_I + L + C^{\rm in}_{n-1} \\ &\quad + {\rm Round}_{\rm Nearest}(f_n, f_{n-1}, R \vee Sy), \\ &= E_I + 2 \times C^{\rm in}_{n} + f_{n-1} \\ &\quad + {\rm Round}_{\rm Nearest}(f_n, f_{n-1}, R \vee Sy). \end{aligned} \tag{14}$$

In the case of RS, the selector can be obtained as follows according to Eq. (14):

$${\rm selector} = C^{\rm in}_{n} \vee {\rm Round}_{\rm Nearest}(f_n, f_{n-1}, R \vee Sy). \tag{15}$$

3.3. Round-to-zero mode

It is considered that the predictor of the round-to-zero mode is identical to that of the round-to-nearest mode. Because ${\rm Round}_{\rm Zero}(X, X, X) = 0$ for both NS and RS cases, the selector and $q^{\rm NS}_0$ can be obtained by replacing ${\rm Round}_{\rm Nearest}(f_{n-1}, R, Sy)$ and ${\rm Round}_{\rm Nearest}(f_n, f_{n-1}, R \vee Sy)$ of Eqs. (13) and (15) with the value of zero. Therefore, the selector and $q^{\rm NS}_0$ are as follows:

$$\begin{aligned} {\rm selector} &= C^{\rm in}_{n}, \\ q^{\rm NS}_0 &= f_{n-1}. \end{aligned} \tag{16}$$

3.4. Round-to-infinity mode

For a specific case, such as $L = c_{n-2} \oplus s_{n-2} = C^{\rm in}_{n-2} = Sy = 1$, if the predictor of round-to-nearest mode is used, then $C^{\rm in}_{n} = 1$, $f_{n-1} = 0$, $R = 0$, and $Sy = 1$. Thus, rounding result in the round-to-nearest mode is obtained by truncation according to Algorithm 1. However, because the value of Sy is equals to one, i.e., $Sy = 1$, rounding result in the round-to-infinity mode is obtained by incrementation according to Algorithm 3. Eventually, in the round-to-infinity mode, the predictor of the round-to-nearest mode cannot be used. Thus, the predictor is given as in Eq. (17).

$${\rm predictor} = (c_{n-2} \wedge s_{n-2}) \vee Sy. \tag{17}$$

If Sy is equal to zero, the predictor of Eq. (17) is identical to that of the round-to-nearest mode and Sy does not cause any effect for the rounding operation. Given this case, the selector and $q^{\rm NS}_0$ can be obtained by replacing ${\rm Round}_{\rm Nearest}(f_{n-1}, R, Sy)$ and ${\rm Round}_{\rm Nearest}(f_n, f_{n-1}, R \vee Sy)$ of Eqs. (13) and (15) with ${\rm Round}_{\rm Infinity}(f_{n-1}, R, Sy)$ and

$\text{Round}_{\text{Infinity}}(f_n, f_{n-1}, R \vee Sy)$, respectively. If Sy is equal to one, rounding is obtained by incrementation and predictor is selected as one. Then, Q^{NS} is represented as follows according to Eqs. (3) and (7):

$$\begin{aligned} Q^{\text{NS}} &= F_I + \text{Round}_{\text{Infinity}}(f_{n-1}, R, Sy), \\ &= F_I + \text{predictor}, \\ &= C_I + S_I + \text{predictor} + \text{overflow}(s_{n-2} \\ &\quad + c_{n-2} + C^{\text{in}}_{n-2}), \\ &= E_I + L + \text{overflow}(s_{n-2} + c_{n-2} + C^{\text{in}}_{n-2}). \end{aligned} \tag{18}$$

Therefore, in the case of NS, the selector and q_0^{NS} are as follows:

$$\begin{aligned} &\text{selector} = L \wedge \text{overflow}(c_{n-2} + s_{n-2} + C^{\text{in}}_{n-2}), \\ &q_0^{\text{NS}} = L \oplus \text{overflow}(c_{n-2} + s_{n-2} + C^{\text{in}}_{n-2}). \end{aligned} \tag{19}$$

In the case of RS, if Sy is equal to one, rounding is obtained by incrementation. Then, $Q^{\text{RS}} = F_I + 2$ according to Eq. (8). Because one is added by the predictor, $Q^{\text{RS}} = Q^{\text{NS}} + 1$. Hence, Q^{RS} is obtained as given below.

$$Q^{\text{RS}} = E_I + L + 1 + \text{overflow}(s_{n-2} + c_{n-2} + C^{\text{in}}_{n-2}). \tag{20}$$

Thus, the selector can be obtained as

$$\text{selector} = L \vee \text{overflow}(c_{n-2} + s_{n-2} + C^{\text{in}}_{n-2}). \tag{21}$$

3.5. e_n Versus NS and RS

In this subsection, the relationships between f_{2n-1}, which is used to select a case of either NS or RS, and e_n are illustrated with respect to the round-to-nearest, round-to-zero, and round-to-infinity modes.

In the round-to-nearest mode, F_I and E_I^* are represented as follows according to Eqs. (4), (9) and (10). Note that $D_I = C_I + S_I = d_n d_{n-1} \cdots d_0$ as mentioned in Section 2.2.

$$\begin{aligned} &F_I = C_I + S_I + C^{\text{in}}_{n-1} = D_I + C^{\text{in}}_{n-1}, \\ &E_I^* = C_I + S_I + \text{predictor} = D_I + \text{predictor}, \\ &C^{\text{in}}_{n-1} = \text{overflow}(c_{n-2} + s_{n-2} + C^{\text{in}}_{n-2}), \\ &\text{predictor} = \text{overflow}(c_{n-2} + s_{n-2}). \end{aligned} \tag{22}$$

Here, if $e_n = \text{overflow}(E_I^*) = 1$, then $f_{2n-1} = 1$ because F_I is greater than or equal to E_I^*, i.e., $F_I \geqslant E_I^*$. Therefore, when the value of e_n equals to one, it is a case of RS about E_I^*.

When $e_n = 0$, following three cases may be generated. First, if $\text{predictor} = 1$, then $C^{\text{in}}_{n-1} = 1$ according to Eq. (22). Because F_I equals to E_I^*, $f_{2n-1} = 0$. Second, if $\text{predictor} = 0$ and $C^{\text{in}}_{n-1} = 0$, then $F_I = E_I^*$ according to Eq. (22) and thus $f_{2n-1} = 0$. Third, if $\text{predictor} = 0$ and $C^{\text{in}}_{n-1} = 1$, then $C^{\text{in}}_{n-2} = 1$ and $c_{n-2} \oplus s_{n-2} = 1$ according to Eq. (22). In this case, $f_{2n-1} = 1$ can occur. That is, for all k, if $d_k = 1$ $(0 \leqslant k \leqslant n-1)$ and $d_n = 0$, then $F_I = 1000\cdots000$ and $E_I^* = 0111\cdots111$ according to Eq. (22), and $f_{n-1} = R = 0$ according to Eq. (11). But, because rounding is obtained by truncation, only propagation from C^{in}_{n-2} can affect the value of Q. Therefore, though this can be a case of RS about F since $f_{2n-1} = 1$, it is considered as a case of NS about E_I^*. Consequently, if $e_n = 0$ then it is a case of NS about E_I^*, and if $e_n = 1$ then it is a case of RS about E_I^*. In the round-to-zero mode, rounding is always obtained by truncation. Therefore, if $e_n = 0$ then it is a case of NS about E_I^*, and if $e_n = 1$ then it is a case of RS about E_I^*.

In the round-to-infinity mode, if $Sy = 0$, then the predictor of the round-to-infinity mode is identical to that of the round-to-nearest mode. And, in the third case of $e_n = 0$ for the round-to-nearest mode, the result of rounding in the round-to-infinity mode is obtained by truncation. Therefore, if $e_n = 0$, then it is a case of NS about E_I^*, and if $e_n = 1$ then it is a case of RS about E_I^*. If $Sy = 1$, then rounding result is obtained by incrementation. Also, F_I and E_I^* can be represented as follows:

$$
\begin{aligned}
F_I &= C_I + S_I + C_{n-1}^{\text{in}} = D_I + C_{n-1}^{\text{in}},\\
E_I^* &= C_I + S_I + \text{predictor} = D_I + 1,\\
C_{n-1}^{\text{in}} &= \text{overflow}(c_{n-2} + s_{n-2} + C_{n-2}^{\text{in}}).
\end{aligned}
\tag{23}
$$

When the result of e_n is equal to zero, the value of f_{2n-1} becomes zero because F_I is less than or equal to E_I^*, i. e., $F_I \leqslant E_I^*$. Therefore, if $e_n = 0$, then it is a case of NS about E_I^*. When $e_n = 1$, following two cases may be generated. First, if $C_{n-1}^{\text{in}} = 1$, then $F_I = E_I^*$ according to Eq. (23) and thus $f_{2n-1} = 1$. Second, if $C_{n-1}^{\text{in}} = 0$, $F_I = E_I^* - 1$ according to Eq. (23). Eventually, $f_{2n-1} = 0$ can occur. That is, for all k, if $d_k = 1$ $(0 \leqslant k \leqslant n-1)$ and $d_n = 0$, then $F_I = 011\cdots111$ and $E_I^* = 100\cdots000$. In this case, rounding is obtained by incrementation and according to Eq. (7) the most significant n bits of the result value after rounding F are $Q^F = 100\cdots000$. Also, according to Eq. (21) the most significant n bits of the result value after rounding E_I^* are $Q^E = 100\cdots000$, resulting in $Q^F = Q^E$. Therefore, though this can be a case of NS about F since $f_{2n-1} = 0$, it is a case of RS about E_I^*. Consequently, if $e_n = 0$, it is a case of NS about E_I^*, and if $e_n = 1$, it is a case of RS about E_I^*.

Conclusively, with regard to each rounding mode, if $e_n = 0$ then it is a case of NS about E_I^*, and if $e_n = 1$, then it is a case of RS about E_I^*. Thus, rounding and addition operations are analytically proved to be performed parallel for all four IEEE rounding modes.

4. Conclusion

In this research, a floating point multiplier capable of performing IEEE rounding and addition in parallel is presented. Processing flow of the proposed floating point multiplier consists of three stages, i.e., multiplication, addition and rounding, and normalization.

The presented floating point multiplier does not require any additional hardware for rounding and renormalization, but it is constructed by using an additional n-bit half adder, a full adder, selector, predictor, and q_0^{LS}. However, any high speed adder for rounding, which is needed in the conventional floating point multiplier, accompanies much longer execution time and uses a larger amount of chip area than the additional hardware used in this approach. Therefore, the presented floating point multiplier provides effectiveness in the point of chip area and improved execution time.

The parameters required for the selector and q_0^{NS} are L, e_1, c_{n-2}, s_{n-2}, Sy, e_n, and C_{n-2}^{in}. Among them, c_{n-2}, s_{n-2}, and Sy are generated before addition, L and e_1 are generated at the initial stage of addition, and e_n and C_{n-2}^{in} are generated after addition. To generate the results of the selector and q_0^{NS}, delay time may be caused because the value of the selector and q_0^{NS} can be determined after addition stage. But, if the logics of the selector and q_0^{NS} are implemented to minimize delay, then the delay time can be negligible enough to be ignored. Thus, performance improvement and cost effective design for the floating point multiplication can be achieved by this approach.

Acknowledgements

This work was one of ASIC basis technology project mainly managed by IDEC and supported by the ministry of trade, industry and energy and the ministry of science and technology of Korea.

References

[1] D. Alpert, D. Avnon, Architecture of the pentium microprocessor, IEEE Micro 13 (3) (1993) 11–21.
[2] C.R. Moore, M.C. Becker, M.S. Allen, D.P. Tuttle, The PowerPC 601 microprocessor, IEEE Micro 13 (5) (1993) 54–67.

[3] N. lde, H. Fukuhisa, Y. Kondo, T. Yoshida, M. Nagamatsu, J. Mori, I. Yamazaki, K. Ueno, A 320-MFLOPS CMOS floating-point processing unit for superscalar processors, IEEE J. Solid-state Circuits 28 (3) (1993) 352–360.

[4] D. Goldberg, Computer arithmetic, in: J.L. Hennessy, D.A. Patterson (Eds.), Computer Architecture: A Quantitative Approach, Appendix A, Morgan Kaufmann, Los Altos, CA, 1990.

[5] M. Awaga, H. Takahashi, The μVP 64-bit vector coprocessor: A new implementation of high-performance numerical computation, IEEE Micro 13 (5) (1993) 24–36.

[6] M. Darley, B. Kronlage, D. Bural, B. Churchill, D. Pulling, P. Wang, R. Iwamoto, L. Yang, The TMS390C602A floating-point coprocessor for sparc systems, IEEE Micro 10 (3) (1990) 36–47.

[7] M. Birman, A. Samuels, G. Chu, T. Chuk, L. Hu, J. McLeod, J. Barnes, Developing the WTL3170/3171 sparc floating-point coprocessors, IEEE Micro 10 (1) (1990) 55–64.

[8] IEEE Std 754-1985, IEEE standard for binary floating-point arithmetic, IEEE, 1985.

[9] W.C. Park, S.W. Lee, O.Y. Kown, T.D. Han, S.D. Kim, Floating point adder/subtractor performing IEEE rounding and addition/subraction in parallel, IEICE Trans. Information and Systems e79-d (4) (1996) 297–305.

[10] H. Fujii, C. Hori, T. Takada, N. Hatanaka, T. Demura, G. Ootomo, A floating-point cell library and a 100-MFLOPS image signal processor, IEEE J. Solid-state Circuits 28 (7) (1992) 1080–1088.

[11] M.R. Santoro, G. Bewick, M.A. Horowitz, Rounding algorithms for IEEE multiplier, in: Proceedings of the Ninth IEEE Symposium on Computer Arithmetic, 1989, pp. 176–183.

[12] N. Quach, N. Takagi, M. Flynn, On fast IEEE rounding, Technical Report CSL-TR-91-459, Stanford University, 1991.

[13] J.A. Prabhu, G.B. Zyner, 167 MHZ Radix-4 Floating Point Multiplier, in: Proceedings of the IEEE Symposium on Computer Arithmetic, July 1995, pp. 149–154.

Woo-Chan Park received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, Korea, in 1993 and 1995, respectively. He is currently pursuing Ph.D. degree at Yonsei University. His research interests include 3D computer graphics accelerator, computer arithmetic, high performance computer architecture, and VLSI design.

Tack-Don Han received the B.S. degree in electronic engineering from Yonsei University, Seoul, Korea, in 1978, and the M.S. degree in computer engineering from Wayne State University in 1983. In 1987, he received the Ph.D. degree in computer engineering from the University of Massachusetts at Amherst. He had been an assistant professor of electrical engineering at Cleveland State University from 1987 to 1989. In 1996, he was a visiting associate professor of Computer Systems Laboratory at Stanford University. He is currently an associate professor of computer science at Yonsei University, Seoul, Korea. His research interests include HCI(Human Computer Interface), VLSI design, processor and memory integration, advanced computer architecture, and parallel algorithms/architecture.

Shin-Dug Kim received the B.S. degree in electronic engineering from Yonsei University, Seoul, Korea, in 1982, and the M.S. degree in electrical engineering from the University of Oklahoma in 1987. In 1991, he received the Ph.D. degree at Purdue University in electrical engineering. He had been an associate professor of computer engineering at the KwangWoon University, Seoul, Korea from 1992 to 1994. He is currently an assistant professor of computer science at Yonsei University, Seoul, Korea. His research interests include parallel system architectures, parallel algorithm design, 3D computer graphics, and web computing.

Sung-Bong Yang received the B.S. degree in ceramic engineering from Yonsei University, Seoul, Korea, in 1981, and his M.S. and Ph.D. degrees from the University of Oklahoma in 1986 and 1992 respectively. He was an adjunct assistant professor of the School of Computer Science at the University of Oklahoma for the Fall semester of 1992. He is currently an assistant professor of the computer science at the Yonsei University, Seoul, Korea. His research interests include parallel and distributed algorithms, GIS, and high performance computing.