LETTER

Design of the Floating-Point Adder Supporting the Format Conversion and the Rounding Operations with Simultaneous Rounding Scheme*

Woo-Chan PARK†, **Cheol-Ho JEONG**†, *and* **Tack-Don HAN**†, ***Nonmembers***

SUMMARY The format conversion operations between a floating-point number and an integer number and a round operation are the important standard floating-point operations. In most cases, these operations are implemented by adding additional hardware to the floating-point adder. The SR (simultaneous rounding) method, one of the techniques used to improve the performance of the floating-point adder, can perform addition and rounding operations at the same stage and is an efficient method with respect to the silicon area and its performance. In this paper, a hardware model to execute CRops (conversion and rounding operations) for the SR floating-point adder is presented and CRops are analyzed on the proposed hardware model. Implementation details are also discussed. The proposed scheme can maintain the advantages of the SR method and can perform each CRop with three pipeline stages.

key words: *computer arithmetic, floating-point unit, floating-point adder*

1. Introduction

CRops consist of three operations: *ftoi*, *itof*, and *rnd* operations [1]. The *ftoi* denotes conversion operation from a floating-point number to an integer number. The *itof* denotes conversion operation from an integer number to a floating-point number. The *rnd* is to round a floating-point number to an integral valued floating-point number in the same format.

In general the processing flow of the conventional floating-point addition operation consists of alignment, addition, normalization, and rounding stages [2]. Several techniques have been provided to improve the performance. The SR method, presented in [3]–[7], can perform addition and rounding operations at the same stage and neither requires 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. However, it is difficult to support CRops on the SR floating-point adder because the processing flows of CRops will not match well with that of SR floating-point adder.

In this paper, a hardware model to execute CRops for SR floating-point adder is presented. CRops are analyzed on the proposed hardware model. In addition, implementation details are discussed. Among SR methods the hardware model of [3] is adapted in this paper. The overall result of this paper can be easily adapted into other SR floating-point adders with minor modifications.

In this paper, only 32-bit single precision floating-point numbers and 32-bit integer numbers are considered. Other number formats, such as 64-bit double precision floating-point numbers and 64-bit integer numbers, can be manipulated by minor modification to the proposed scheme. The processing of CRops can be divided into the exponent part and the fractional part. Specifically only the fractional part is considered because the exponent part is very simple.

The next section presents a brief overview of the SR floating-point adder. In Sect. 3, a hardware model to execute CRops for the SR method is proposed, the operational flows of the CRops for the SR floating-point adder are analyzed, and implementation details are discussed. Finally, conclusions are presented in Sect. 4.

2. Background

2.1 The Operational Flows of the CRops

The brief processing flow of the *ftoi* operation is as follows. The significand value of the floating point number is shifted to the right by the shift amount calculated in exponent value. Then, the rounding operation is performed with LSB, G, R, Sy bits for each rounding mode. Finally, because the significand is represented as sign-magnitude, the rounded result is converted into 2s complement form in the case of negative number.

The brief processing flow of the *itof* operation is as follows. If input integer number is negative, it is converted into absolute number. Then, as the normalization step, the leading zero is calculated in the leading zero counter [9] for the absolute number. Finally, rounding is performed.

The *rnd* operation can be performed with the first two steps of the *ftoi* operation and the second step of

Manuscript received July 2, 2001.
Manuscript revised February 25, 2002.

†The authors are with the Department of Computer Science, Yonsei University, Seoul, Korea.

*This work was supported by the NRL-Fund from the Ministry of Science & Technology of Korea.

the *itof* operation. The processing flow is as follows. The first step performs alignment operation, which is identical with the first step of the *ftoi* operation. Then, rounding is performed, which equals to the second step of the *ftoi* operation. At last, the rounded result is shifted to left for normalization operation, which is identical with the second step of the *itof* operation.

2.2 The SR Floating Point Adder

Figure 1 shows the overall block diagram of the SR floating-point adder [3], [4]. An SR floating-point adder consists of three pipeline stages: alignment, addition and rounding, and normalization stages. A major point of difference compared with conventional floating-point adder is that addition and rounding operations can be performed in the same pipeline stage. The first stage and the last stage of Fig. 1 are similar to the alignment stage and the normalization stage of the conventional floating-point adder.

Figure 2 shows a detailed hardware model of the SR floating-point adder. In Fig. 2, because the normalization stage of the SR floating-point adder is identical to that of the conventional one, it is omitted. This hardware is obtained from the algebraic analysis to perform rounding and addition in parallel. In [3], a detailed analysis and logical gate level implementation are discussed in accordance with each rounding mode.

The significand of the smaller or equal exponent is denoted as M_1 and the significand of the larger or equal exponent is represented as M_2. Then, the shifted result of M_1 after alignment and the result value through bit-inverter are sent into the bit-half adder. The bit-half adder produces the carry part and the sum part. An empty slot is reserved at the LSB of the carry part by the half adder. The predictor bit is inserted into this empty slot. The predictor calculates predicted value for simultaneous rounding before carry propagation addition and is organized with very simple gates. Adder0 and adder1 are implemented by a single CSA (Carry Select Adder) [8], with C_{out} as an overflow signal of adder0. The ARD (addition and rounding decision) signal selects one of the result values, such as the two input values $i0$ and $i1$, after executing addition and rounding operations. If ARD = 0, then $i0$ is selected, otherwise $i1$ is selected as the output value of the multiplexer. The signals to generate the result value of ARD for each rounding mode are op, C_{out}, the MSB of the result value of adder1, the least significant two bits of the result of adder0, G, R, and Sy. The result value of the multiplexer is converted into a 1's complemented absolute value by the bit-inverter. In the case of $op = 1$, the LSB of the output value of the multiplexer may not be a correct result in some cases. The q_0^{LS} complements the LSB value in the case of $op = 1$.

It is possible to implement rounding and renormalization operations with other additional logic, instead

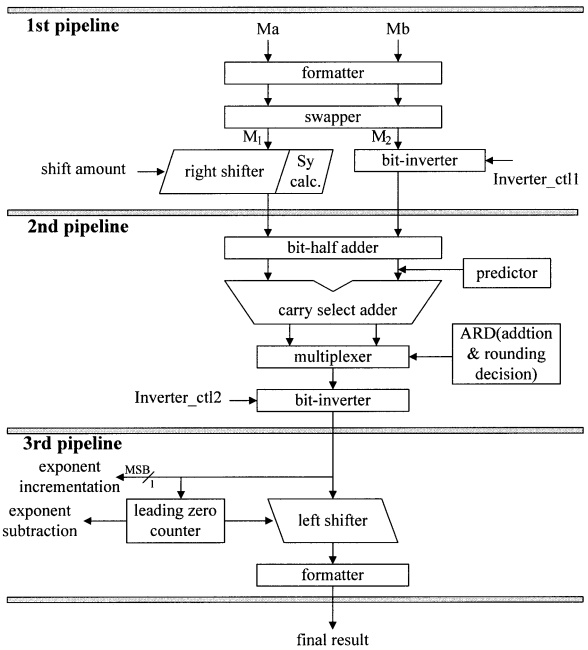

Fig. 1 Pipelines of the SR floating-point adder.

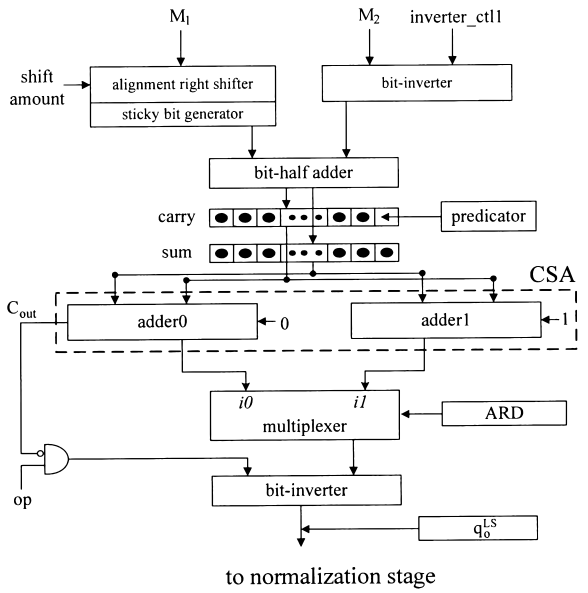

Fig. 2 Hardware model of the SR floating-point adder.

of an additional bit-half adder, a predictor, a selector, and a logic for q_0^{LS} as in the SR floating-point adder. This approach, however, would cause much longer execution time and use a large amount of chip area.

3. The Proposed Floating-Point Adder Supporting CRops

3.1 Proposed Hardware Model

Figure 3 shows the proposed hardware model supporting CRops on the SR floating-point adder. Comparing

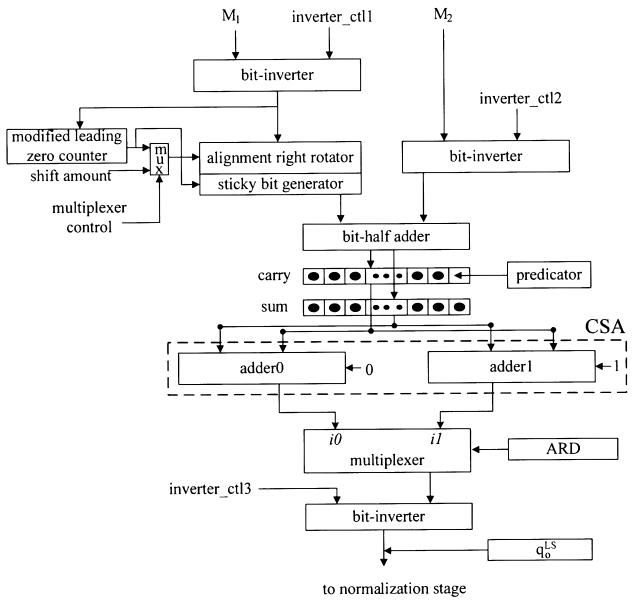

Fig. 3 Proposed hardware model supporting CRops.

with Fig. 2, a bit-inverter and a modified leading zero counter are added, the alignment right rotator is used instead of the right shifter, and the sticky bit generator is modified a little bit. Because a bit- inverter and a modified leading zero counter are executed in parallel with exponent subtraction that calculates shift amount for alignment, the execution delay for these logics is not critical.

The most important point for mapping the CRops operational steps into those of an SR floating-point adder is to sustain the features of the SR method, i.e., performing addition and rounding steps at the same pipeline stage. To keep the features of the SR method, results must be generated with the cases of the truncated result when ARD = 0 and the incremented result when ARD = 1.

3.2 The Operational Flow of the *ftoi* for the Proposed Model

It is assumed that A is the right shifted result from the right rotator of the first pipeline. In the case of a positive number, if $Round_{mode}(LSB, G, R, Sy)$ is an increment, then the RVAR (result value after rounding) is $A + 1$. Otherwise, i.e., if $Round_{mode}(LSB, G, R, Sy)$ is truncated, then the RVAR is A. $Round_{mode}(LSB, G, R, Sy)$ denotes the result of rounding operation according to each rounding mode. When all three inverter control signals are zero and M_2 is zero, the result values of adder0 and adder1 is A and $A + 1$ respectively, which is compliant for the features of the SR method.

In the case of a negative number, the RVAR can be generated by negating the result value after the rounding operation for A, as follows.

$$\begin{aligned} \mathrm{R}VAR &= -(A + Round_{mode}(LSB, G, R, Sy)) \\ &= -A - Round_{mode}(LSB, G, R, Sy) \\ &= \overline{A} + 1 - Round_{mode}(LSB, G, R, Sy) \\ &= \overline{A} + \overline{Round_{mode}(LSB, G, R, Sy)} \end{aligned}$$

Hence, if $Round_{mode}(LSB, G, R, Sy)$ is an increment, then the RVAR is $\overline{A}$, otherwise the RVAR is $\overline{A}+1$. The 'bar' over 'A' represents the bit inversion. These result values can be generated when inverter_ctl1 and M_2 are zero and inverter_ctl2 and inverter_ctl3 are one. Then the inverted value of M_2 can be evaluated as -1. In this case, the inverted values of adder0 and adder1 are $\overline{A-1} = \overline{A} + 1$ and $\overline{A-1+1} = \overline{A}$ respectively. Hence, the predictor and the ARD of the SR floating-point adder can be used without any modification.

3.3 The Operational Flow of the *itof* for the Proposed Model

In the case of floating point addition operation, an SR floating-point adder can catch some information for the normalization operation from two input exponents. However, in *itof* operation, because normalization operation must be performed prior to rounding operation, it is very difficult to implement the *itof* operation on the SR floating-point adder.

When M_1 is a positive number, normalization operation should be executed before rounding step according to Sect. 2.1. Therefore, normalization step must be performed in alignment stage to keep the three pipelines fully running. To achieve this, an additional MLZC (modified leading zero counter) and a bi-directional shifter, instead of the right shifter, are adapted in the alignment stage. The silicon area and the execution delay of the MLZC are not critical. However, because the silicon area overhead of the bi-directional shifter is almost twice compared with the right shifter, a right rotator is adapted in this paper. By the addition of some techniques, the left shift operation can be performed with the right rotator.

In Fig. 4, k-bit left shifting with an n-bit right rotator in the case of a positive number is shown. Where $a_{n-1}a_{n-2}\cdots a_{n-k}a_{n-k-1}\cdots a_1a_0$ is denoted as an n-bit number input into the right rotator, i.e., M_1 of Fig. 4. First, the $n-k$ bit right rotation operation is performed. In fact the value of $n-k$ can be generated directly by the MLZC. Then, $a_{n-k-1}a_{n-k-2}\cdots a_0a_{n-1}a_{n-2}\cdots a_{n-k}$ is generated as a result value. The second step is the generation of the n-bit mask of which the most significant $n-k$ bits are all one and the least significant k bits are all zero. In the third step, $a_{n-k-1}a_{n-k-2}\cdots a_0 0\cdots 0$, is the n-bit left shifted result, is generated by performing a logical-AND operation for each bit position with the results of the first step and the second step. Because the results of the first step and the second step can be generated

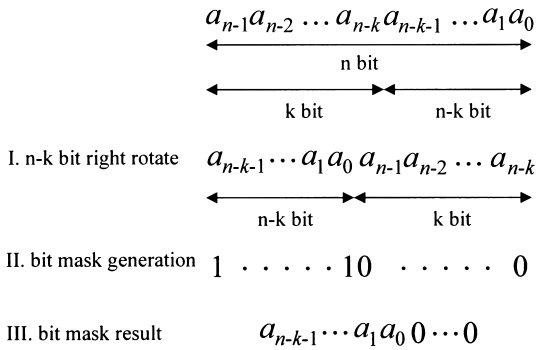

Fig. 4 k-bit left shifting with n-bit right rotator in the case of a positive number.

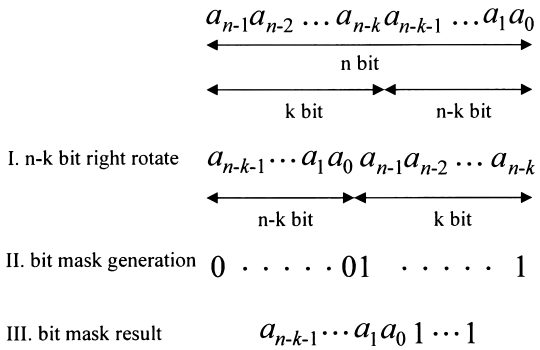

Fig. 5 k-bit left shifting with n-bit right rotator in the case of a negative number.

in parallel, the bit mask generation step will not affect the critical path delay.

When M_1 is a positive number, the value of the inverter_ctl1 is zero and M_1 is transmitted into the MLZC. The right rotator performs the left shift operations by a shift amount calculated from the MLZC and generates G, R, and Sy bits. If a 32-bit M_2 is zero and inverter_ctl2 and inverter_ctl3 are all zero, then the predictor and the ARD of an SR floating-point adder can be used without any modification because the $i0$ is selected in the case of truncation and $i1$ is in the case of increment.

When M_1 is a negative number, normalization operation should be performed after converting the negative number into an absolute one. To achieve this, a high-speed incrementer may be required. However, a high-speed incrementer requires a considerable amount of silicon area and gate delay, this scheme is not an effective approach. In this paper, by adjusting the value of M_2, this problem has been solved. Figure 5 shows the k-bit left shifting with an n-bit right rotator in the case of a negative number. $a_{n-1}a_{n-2}\cdots a_{n-k}a_{n-k-1}\cdots a_1a_0$ in Fig. 5 is the inverted result of M_1 and is input into the right rotator. Initially, an $n-k$ bit right rotation operation is executed. Then, $a_{n-k-1}a_{n-k-2}\cdots a_0a_{n-1}a_{n-2}\cdots a_{n-k}$ is generated. The second step is the generation of the bit mask of which the most significant $n-k$ bits are all zero and the least significant k bits are all one. In the third stage, $a_{n-k-1}a_{n-k-2}\cdots a_01\cdots1$ is generated by performing a logical-OR operation for each bit position with the results of the first step and the second step. Then, the inverter_ctl2 is zero and M_2 is one, i.e., $000\cdots01$. This adjustment for M_2 can be performed in the formatter of the first pipeline in Fig. 1. Because the two input values of the half adder are $a_{n-k-1}a_{n-k-2}\cdots a_01\cdots1$ and $000\cdots01$, the 2's complement absolute value of M_1 can be calculated in the addition stage of the second pipeline. Hence, the predictor and the ARD of the SR floating-point adder can be used without any modification. Then, to achieve this goal, G, R, and Sy bits should be generated at the first pipeline stage.

In the case of a positive number, if the least significant six bits of the bit mask result in Fig. 4 are all zero then the resultant value of the Sy bit is zero, otherwise the Sy bit is one. Hence, only a six-input logical-OR gate is required to generate the Sy bit. In the case of a negative number, Sy is calculated by the bitwise logical-ORed value for the least significant six bits after adding $a_{n-k-1}a_{n-k-2}\cdots a_01\cdots1$ and $000\cdots01$. Therefore, in this case, if the six least significant bits of the bit mask result in Fig. 5 are all one then the resultant value of the Sy bit is zero, otherwise the Sy bit is one. To generate the Sy bit, in the negative number case, only a six-input logical NAND gate is required. Also, G and R bits can be provided easily by using the additional simple gates.

3.4 The Operational Flow of the *rnd* for the Proposed Model

In the conventional floating-point adder, because the normalization stage is followed by the rounding stage, rounding operation should be performed in the second pipeline stage to simplify pipeline flow. However, in an SR floating point adder, because addition and rounding operations are performed at the second pipeline stage, the pipelines flow of the SR method is similar to that of *rnd* operation. Thus, the *rnd* operation is easy to implement with an SR floating-point adder.

4. Conclusion

In this paper, a hardware model to execute CRops for the SR floating-point adder is presented and its implementation details are discussed. The proposed technique can maintain the advantages of the SR method and also perform the CRops with three pipeline stages. Floating-point adder supporting CRops of this paper is designed and evaluated by a Verilog HDL, implemented with $0.25\,\mu$ CMOS standard cell library and some hard-macro blocks, and adapted into a 32-bit CalmRISC FPU from Samsung Electronics [10]. We cannot devote

our full efforts to optimization process due to insufficient design time. Current working frequency is estimated about 80 MHz. It seems that we may achieve at least 150 MHz clock frequency after the optimization process.

References

[1] IEEE Std 754-1985, IEEE standard for binary floating-point arithmetic, IEEE, 1985.
[2] D. Goldberg, "Computer arithmetic," in Computer Architecture: A Quantitative Approach, Appendix A, ed. J.L. Hennessy and D.A. Patterson, Morgan Kaufmann Publishers, 1996.
[3] W.C. Park, S.W. Lee, O.Y. Kown, T.D. Han, and S.D. Kim, "Floating point adder/subtractor performing IEEE rounding and addition/subtraction in parallel, " IEICE Trans. Inf. & Syst., vol.E79-D, no.4, pp.297–305, April 1996.
[4] W.C. Park, T.D. Han, and S.D. Kim, "Efficient simultaneous rounding method removing sticky-bit from critical path for floating point addition," Proc. IEEE Asia Pacific Conference on ASICs, pp.223–226, Aug. 2000.
[5] A.B. Smith, N. Burgess, S. Lefrere, and C.C. Lim, "Reduced latency IEEE floating-point standard adder architectures," Proc. IEEE 14th Symposium on Computer Arithmetic, pp.35–42, April 1999.
[6] N. Quach and M. Flynn, "Design and implementation of the snap floating-point adder," Technical Report CSL-TR-91-501, Stanford University, Dec. 1991.
[7] P.M. Seidel and G. Even, "How many logic levels does floating point addition required ?" Proc. IEEE International Conference on Computer Design, pp.142–149, Oct. 1998.
[8] A. Tyagi, "A reduced-area scheme for carry-select adders," IEEE Trans. Comput., vol.42, no.10, pp.1163–1170, Oct. 1993.
[9] V.G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," IEEE Trans. VLSI Systems, vol.2, no.1, pp 124–128, March 1994.
[10] C.H. Jeong, W.C. Park, S.W. Kim, T.D. Han, and M.K. Lee, "In order issue our-of-order execution floating-point coprocessor for CalmRISC32," Proc. IEEE 15th Symposium on Computer Arithmetic, pp.195–120, June 2001.