Performance Evaluation of Multi-operands Floating-Point Adder

Arvind Kumar, Sunil Kumar, Prateek Raj Gautam, Akshay Verma and Tarique Rashid

Abstract In this paper, an architecture is presented for a fused floating-point three operand adder unit. This adder executes two additions within a single unit. The purpose of this execution is to lessen total delay, die area, and power consumption in contrast with traditional addition method. Various optimization techniques including exponent comparison, alignment of significands, leading zero detection, addition, and rounding are used to diminish total delay, die area, and power consumption. In addition to this, the comparison is described of different blocks in term for die area, total delay, and power consumption. The proposed scheme is designed and implemented on Xilinx ISE Design 14.7 and synthesized on Synopsis.

Keywords Floating-point adder · Significand bits · Exponent bits · Total delay and Xilinx

1 Introduction

The use of floating-point arithmetic, which is according to IEEE-754 standard [1], is to make general-purpose application specific processor. Floating-point number contains three components: exponent bits, the sign bit, and significand bits that are

A. Kumar (✉) · S. Kumar · P. Raj Gautam · A. Verma · T. Rashid
Motilal Nehru National Institute of Technology Allahabad, Allahabad, India
e-mail: arvindk@mnnit.ac.in

S. Kumar
e-mail: rel1516@mnnit.ac.in

P. Raj Gautam
e-mail: prateekrajgautam@gmail.com

A. Verma
e-mail: rel1602@mnnit.ac.in

T. Rashid
e-mail: rel1404@mnnit.ac.in

A. Khare et al. (eds.), *Recent Trends in Communication, Computing, and Electronics*, Lecture Notes in Electrical Engineering 524,
https://doi.org/10.1007/978-981-13-2685-1_51

Sign (1-bit)	Exponent (8-bit)	Significand (23-bit)

Fig. 1 Representation of single precision floating-point number [1]

Sign (1-bit)	Exponent (11-bits)	Significand (52-bits)

Fig. 2 Representation of double precision floating- point number [1]

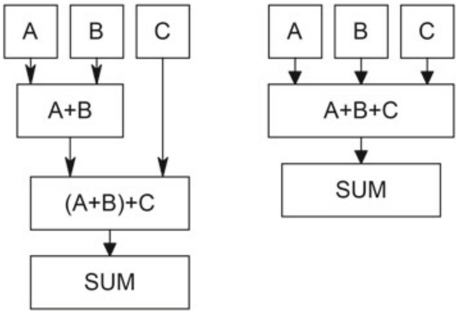

Fig. 3 Discrete versus fused floating-point adder [2]

shown in Fig. 1. There are two standard floating-point representations [1]: single precision and double precision representation. In single precision representation, there are one sign bit, eight exponent bit, and twenty-three significand bits. However, in double precision representation, there are one sign bit, eleven exponent bit, and fifty-two significand bits that are shown in Figs. 1 and 2.

An addition is more significance in arithmetic, and it is widely used operation in various applications. Discrete floating-point adder uses two operands at a time which is well optimized. In order to use multiple operands for the addition, we have to use the multiple traditional adder ones after other because it can use only two operands at a time. Discrete floating-point adders degrade accuracy owing to the multiple rounding one after in each addition. Due to this die area, total delay and power consumption become larger. To improve quality, the fused floating-point adder is used. It executes two additions in a single unit so that only single rounding is required which reduces die area and power consumption. The comparison between discrete and fused floating-point adder [2, 3] is shown in Fig. 3.

The proposed adder performed addition of three floating-point operands and executed additions as

$$S = A \pm B \pm C$$

There are many fused floating-point units that are presented: fused multiply-add (FMA), fused add subtract (FAS), fused dot product (FDP), and a fused three-term adder (FTA) [4, 5].

2 Methodology

The algorithm of three terms is given in Fig. 4 is represented as [6–8]

1. Unpacking each of the three floating-point numbers A, B, and C to obtained sign bit (1 bit), exponent (8 bits), and significand (23 bit +1 bit hidden).
2. In order to find the maximum exponent from the three exponents and calculate the exponent difference.
3. Arrange the significands right shift according to their respective exponent difference.
4. Sign logic determines the sign of A, B, and C according to op-codes op1 and op2.
5. Invert the significands according to their respective sign obtained from the sign logic.
6. Significand addition is performed by using 3:2 CSA (carry-save adder).
7. Leading zero detector is to compute leading zero of the output of CSA, and accordingly, significand is shifted by the same amount and exponent is also adjusted.
8. Rounding operation is performed to round off the resultant significand.
9. If output of CSA produces carry, then right shift the significand by 1, and accordingly, exponent will increment by 1.
10. Pack the resultant sign bit, exponent bits, and significand bits to produce the resultant floating-point number.

3 Proposed Design and Implementation

3.1 Exponent Comparison and Alignment of Significand

For floating-point addition [9, 10], that is essential to compute maximum exponent from the three exponents. Exponent difference is performed by subtracting the respective exponents from the maximum exponent. Significands are aligned by right shifting the significand by the amount of the respective exponent difference. All the arrangements of six subtractions of exponent differences ($\exp_a - \exp_b$, $\exp_b - \exp_a$, $\exp_b - \exp_c$, $\exp_c - \exp_b$, $\exp_a - \exp_c$, and $\exp_c - \exp_a$) are performed to calculate.

The differences of each pair, an absolute value is adopted based on the exponent comprising results that enables skipping the complementation after the subtractions.

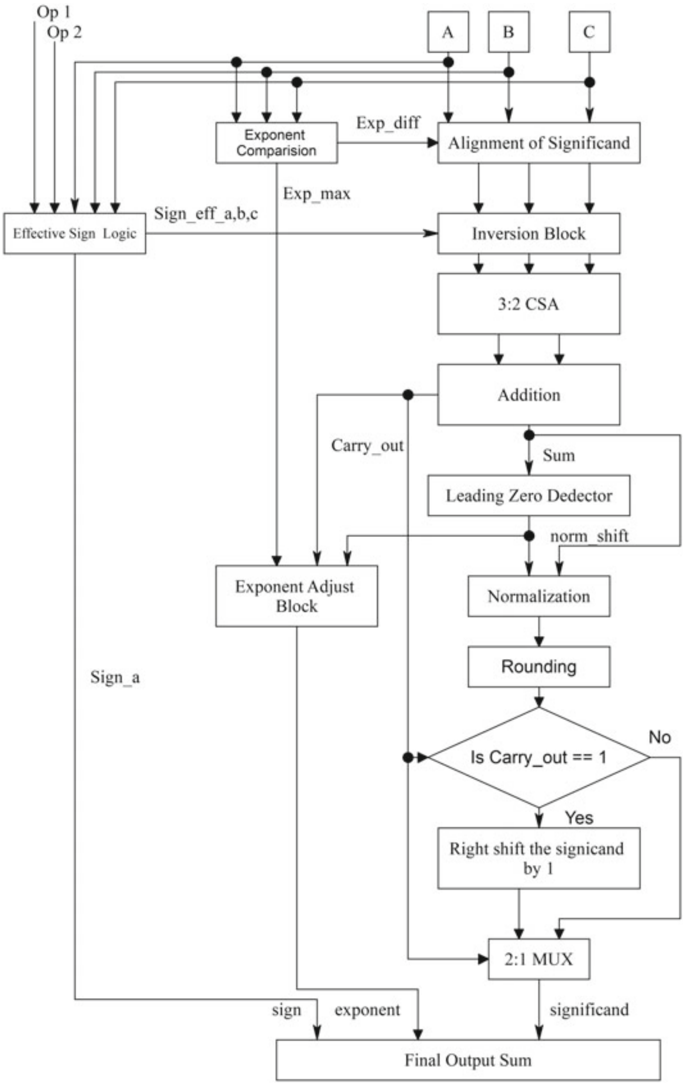

Fig. 4 Multi-operands floating point adder

An exponent comparison and significand arrangement logic is shown in Fig. 5.

The control logic estimates the largest exponent and arranged the significands based on the exponent comprising results as shown in Table 1. In order to guarantee the significand precision, the aligned significands become $2f + 6$ bits wide, including two overflow bits, round bits, guard bits and sticky bits. Where f is the significand bits can be seen in Fig. 6.

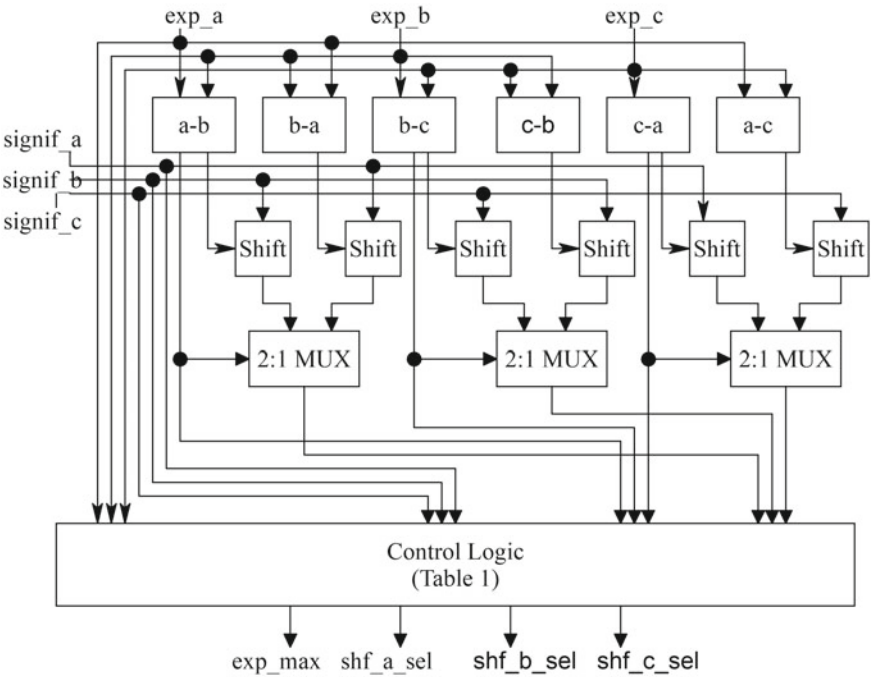

Fig. 5 Exponent comparison and significand arrangement logic

Table 1 Exponent comparison control logic [2]

$A \geq B$	$B \geq C$	$C \geq A$	exp_max	shf_a_sel	shf_b_sel	shf_c_sel
0	0	0	NA	NA	NA	NA
0	0	1	exp_c	shf_ca	shf_bc	signif_c
0	1	0	exp_b	shf_ab	signif_b	shf_bc
0	1	1	exp_b	shf_ab	signif_b	shf_bc
1	0	0	exp_a	signif_a	shf_ab	shf_ca
1	0	1	exp_c	shf_ca	shf_bc	signif_c
1	1	0	exp_a	signif_a	shf_ab	shf_ca
1	1	1	any	signif_a	signif_b	signif_c

3.2 *Effective Sign Logic*

Sign logic determines the three effective sign bits ($sign\_eff\_a$, $sign\_eff\_b$ and $sign\_eff\_c$) on the basis of the three sign bits and two op-codes as

$$sign\_eff\_a = sign\_a$$

$$sign\_eff\_b = sign\_a \oplus (sign\_b \oplus op1)$$

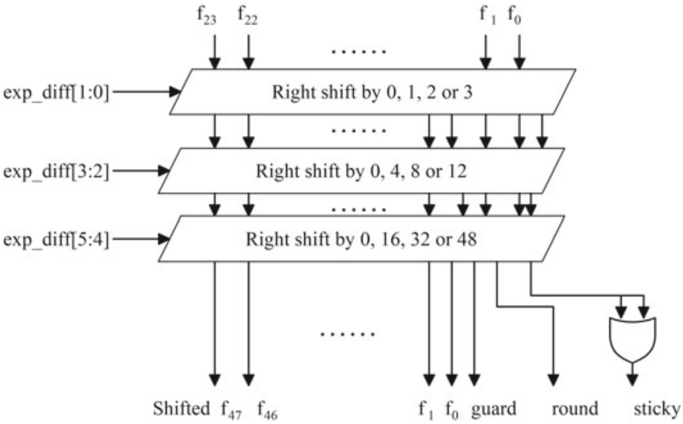

Fig. 6 Significand shifter is shown for single precision [2]

$$sign\_eff\_c = sign\_a \oplus (sign\_c \oplus op2)$$

where $\oplus$ is the sign of exclusive-OR operation.

3.3 Inversion Block

Inversion block complements the significand on the basis of their respective effective sign.

Up to two significands are complimented with the help of three operand subtraction (e.g., $\text{A} - \text{B} - \text{C} = \text{A} + \text{B}' + 1 + \text{C}' + 1 = \text{A} + \text{B}' + \text{C}' + 2$). Increments are avoided after inverters and 2 bits are extended to the LSB of the significands as shown in Table 2.

Table 2 2-bit extended LSBs for complementation [2]

s_eff_a	s_eff_b	s_eff_c	$a_{-1}a_{-2}$	b_1b_{-2}	$c_{-1}c_{-2}$	sum0
0	0	0	00	00	00	0
0	0	1	10	00	10	1
0	1	0	00	10	10	1
0	1	1	10	11	11	2
1	0	0	10	10	00	1
1	0	1	11	10	11	2
1	1	0	11	11	10	2
1	1	1	00	00	00	0

3.4 Carry-Save Adder (CSA)

Each significand is passed to the 3:2 reduction tree. Carry save-adder (CSA) is used to perform the reduction that reduces the three significands with respect to two and then performed the addition. The advantage of using CSA is that it does not propagate carry. It saves the carry which minimizes the total delay in performing addition operation as compared to carry propagate adder.

3.5 Leading Zero Detector and Normalization

This block determines a position of the leading zero from the MSB of the output of the CSA. Significand becomes normalized significand based on the amount of left shift obtained from the leading zero detectors. An exponent is also adjusted by the amount obtained from leading zero detector block. Significand addition with normalization is the highest bottleneck of fused floating-point adder. To diminish the overhead, normalization is used.

3.6 Exponent Adjust Block

The largest exponent (exp max) determined by the exponent compare logic is adjusted by subtracting the shift amount from LZA and adding the carry out of the significand addition as shown in Fig. 7

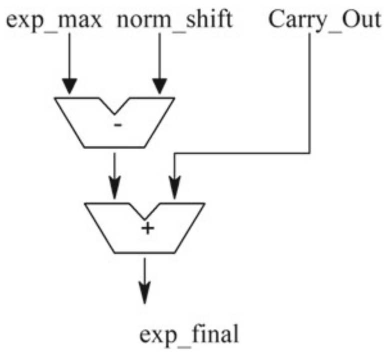

Fig. 7 Exponent adjust block

Table 3 Table for rounding operation

G	R	S	Operation performed
0	0	X	No changes in the LSB
0	1	X	No changes in the LSB
1	1	X	Add 1 to LSB Bit
1	0	0	Round to nearest even
1	0	1	Add 1 to the LSB

Significand bits(23 bits)	G	R	S

Fig. 8 Figure shows the position of significand and guard, round, and sticky bit

3.7 Rounding

In order to truncate the significand, we have to perform the rounding operation with the help floating-point multiplier; significand is round off based on guard bit (G), round bit (R), and sticky bit (S) as shown in Fig. 7 and Table 3. Rounding is determine to rounded floating the value of carry, guard, LSB, round, and sticky bits (Fig. 8).

Here, least significand bit (LSB) bit is just left of the guard bit as shown above.

4 Result Analysis

In this section, modules of proposed architecture are designed in Xilinx 14.7 and synthesis on synopsis tool. Their corresponding results are shown, respectively.

The result of the addition of three floating-point numbers is shown in Fig. 9.

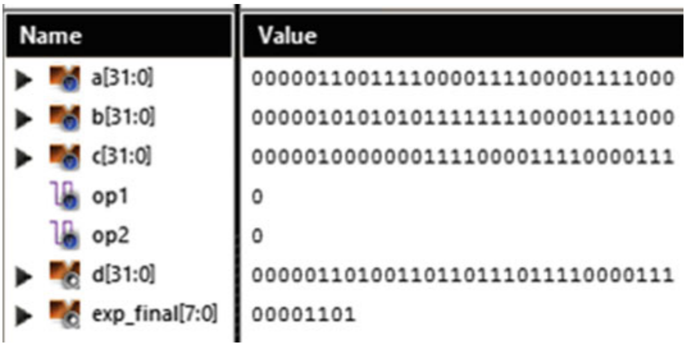

Fig. 9 Simulation result

Table 4 Comparison between a proposed paper with implementation of fused floating-point three-term adder unit [3]

Modules	Implementation of fused floating-point three-term adder unit		Performance evaluation of multi-operands floating-point adder	
	Number of Slices LUT used	Delay (ns)	Number of Slices LUT used	Delay (ns)
Exponent comparison and alignment of significand	519	5.965	1035	4.086
Carry save adder (CSA)			2	0.893
Effective sign logic	2	3.696	2	0.889
Inversion Logic			318	2.942
Leading zero detector and normalization	28	8.524	1052	12.999
Rounding	4	3.809	48	2.788
Control logic	3	3.809	165	0.751
Overall output			2316	18.993

Comparison between execution of fused floating-point three-term adder unit [3] and performance evaluation of multi-operands floating-point adder on the basis numbers of slices LUT used and delay are shown in Table 3. The fundamental difference between proposed and conventional design is alignment of significand bits and rounding. The proposed design executes the lesser significand bits addition compared to conventional designs. Further, the proposed design executes the significand bits addition and rounding at the same time so that the delay is diminished significantly.

The synthesis result obtained from synopsis tool is shown in Table 4.

5 Conclusion

In this paper, we have introduced an improved architecture for three-term adder with a fused floating point which is used to diminish die area, total delay, and power consumption in i with the discrete floating point adder. Further, this paper also compares the different performance of proposed architecture for Implementation of three-term adder unit with fused floating point in terms of delay and number of slices LUT used. In addition, die area and power consumption of different optimized blocks are provided by synthesis result. The optimization blocks are exponent comparison, alignment of significand, CSA, effective sign logic, inversion logic, leading zero detector, normalization, rounding, and control logic. In future, we will design architecture in order to obtain high-speed adder (Table 5).

Table 5 Synthesis result analysis on synopsis

Modules	Power (mW)	Area (μm^2)
Exponent comparison and alignment of significand	1.6009	14411.488
Carry-save adder (CSA)	0.6439	2472.736
Effective sign logic	0.00589	31.36
Inversion logic	0.398	3395.50472
Leading zero detector and normalization	1.0558	5357.856
Rounding	0.006759	700.1120
Control logic	0.5263	3206.5601
Overall output	6.6984	26683.440262

References

1. Zuras, D., Cowlishaw, M., Aiken, A., Applegate, M., Bailey, D., Bass, S., et al. (2008). IEEE standard for floating-point arithmetic. *IEEE Standards*, *754–2008*, 1–70.
2. Sohn, J., & Swartzlander, E. E. (2014). A fused floating-point three-term adder. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *61*(10), 2842–2850.
3. Popalghat, M., & Palsodkar, P. (2016). Implementation of fused floating point three term adder unit. In *2016 International Conference on Communication and Signal Processing (ICCSP)* (pp. 1343–1346). IEEE.
4. Drusya, P., & Jacob, V. (2016). Area efficient fused floating point three term adder. In International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) (pp. 1621–1625). IEEE.
5. Sohn, J., & Swartzlander, E. E. (2012). Improved architectures for a fused floating-point add-subtract unit. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *59*(10), 2285–2291.
6. Tenca, A.F. (2009). Multi-operand floating-point addition. In *2009 19th IEEE Symposium on Computer Arithmetic, ARITH 2009*. (pp. 161–168). IEEE
7. Seidel, P. M., & Even, G. (2004). Delay-optimized implementation of IEEE floating-point addition. *IEEE Transactions on Computers*, *53*(2), 97–113.
8. Tao, Y., Deyuan, G., Xiaoya, F., & Xianglong, R. (2012). Three-operand floating-point adder. In *2012 IEEE 12th International Conference on Computer and Information Technology (CIT)* (pp. 192–196). IEEE.
9. Underwood, K. (2004). Fpgas vs. cpus: trends in peak floating-point performance. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays* (pp. 171–180). ACM.
10. Monniaux, D. (2008). The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *30*(3), 12.