Regular paper | Journal of the KIECS. pp. 409-416, vol. 19, no. 2, Apr. 30. 2024, t. 124, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2024.19.2.409

FPGA를 위한 32비트 부동소수점 곱셈기 설계

Xuhao Zhang* · 김대익**

Design of 32-bit Floating Point Multiplier for FPGA

Xuhao Zhang* · Dae-Ik Kim**

요 약

빠른 고속 데이터 신호 처리 및 논리 연산을 위한 부동 소수점 연산 요구 사항이 확대됨에 따라 부동 소수점 연산 장치의 속도는 시스템 작동에 영향을 미치는 핵심 요소이다. 본 논문에서는 다양한 부동소수점 곱셈기 방식의 성능 특성을 연구하고, 캐리와 합의 형태로 부분 곱을 압축한 다음, 최종 결과를 얻기 위해 캐리 미리 보기 가산기를 사용한다. Intel Quartus II CAD 툴을 이용하여 Verilog HDL로 부동소수점 곱셈기를 기술하고 성능 평가를 하였다. 설계된 부동소수점 곱셈기는 면적, 속도 및 전력 소비에 대해 분석 및 비교하였다. 월러스 트리를 사용한 수정 부스 인코딩 방식의 FMAX는 33.96Mhz로 부스 인코딩보다 2.04배, 수정 부스 인코딩보다 1.62배, 월러스 트리를 사용한 부스 인코딩보다 1.04배 빠르다. 또한, 수정 부스 인코딩에 비해 월러스 트리를 이용한 수정 부스 인코딩 방식의 면적은 24.88% 감소하고, 전력소모도 2.5% 감소하였다.

ABSTRACT

With the expansion of floating-point operation requirements for fast high-speed data signal processing and logic operations, the speed of the floating-point operation unit is the key to affect system operation. This paper studies the performance characteristics of different floating-point multiplier schemes, completes partial product compression in the form of carry and sum, and then uses a carry look-ahead adder to obtain the result. Intel Quartus II CAD tool is used for describing Verilog HDL and evaluating performance results of the floating point multipliers. Floating point multipliers are analyzed and compared based on area, speed, and power consumption. The FMAX of modified Booth encoding with Wallace tree is 33.96 Mhz, which is 2.04 times faster than the booth encoding, 1.62 times faster than the modified booth encoding, 1.04 times faster than the booth encoding with wallace tree. Furthermore, compared to modified booth encoding, the area of modified booth encoding with wallace tree is reduced by 24.88%, and power consumption of that is reduced by 2.5%.

키워드

Floating Point Multiplier, Modified Booth Encoding, Wallace Tree, Carry Look-Ahead Adder, Fpga
부동 소수점 곱셈기, 수정 부스 인코딩, 월러스 트리, 캐리 미리 보기 가산기, FPGA

* 전남대학교 전자통신공학과
(zhangxuh123@sina.com)
** 교신저자 : 전남대학교 전자통신공학과
• 접 수 일 : 2024. 01. 16
• 수정완료일 : 2024. 02. 28
• 게재확정일 : 2024. 04. 12
• Received : Jan. 16, 2024, Revised : Feb. 28, 2024, Accepted : Feb. 12, 2024
• Corresponding Author : Dae-Ik Kim
Dept. of Electronic Communication Engineering, Chonnam National University
Email : daeik@jnu.ac.kr

Ⅰ. Introduction

In digital signal processors (DSP), multipliers are essential functional units [1-3]. Floating-point multipliers (FPM) have the characteristics of large area, long delay, and complex structure. In order to design a high-speed floating-point multiplier, based on the floating-point operation process, in order to reduce the number of partial products, speed up the sum of partial products, and then improve the operation speed of the multiplier, the delay, area, structural complexity, etc. In this regard, each process of the multiplication component has been systematically studied and fully studied and compared [4]. Parallel multipliers are mainly used in high-performance digital processors to complete high-speed data operations. In the process of in-depth research on multipliers, a series of optimization strategies have emerged: reducing the number of partial products through Booth algorithm and compressing tree structures such as Wallace Tree and Dadda Tree to achieve parallel processing of partial products; carry look-ahead adder (CLA,) and carry save adder (CSA) realize the fast sum of partial products of the final two rows. Among them, the partial product compression unit is a key factor that determines the speed, power, and area of the multiplier [5].

To improve the efficiency of floating point multipliers, many fast multiplication algorithms have been proposed and modeled. This article designs a 32-bit floating-point multiplier. When designing a floating-point multiplier, part of the product compression unit is a key factor in determining the speed, power, and area of the multiplier. Some fast multiplication schemes that may be considered for implementation are Booth's algorithm, Vedic multiplication algorithm and array multipliers, Vedic multipliers and Wallace tree multipliers for study and comparison. Rounding is also given due consideration when implementing the multiplier design. Mainly research on modified booth and wallace tree, and optimizing the design of the compression unit of the mantissa operation part. First, the system structure of the multiplier is introduced, then the structure and logic circuit of the compression unit are optimized and analyzed, and simulation analysis is performed, and finally a conclusion is drawn.

Ⅱ. Floating point multiplier

The multiplier and multiplicand input to the multiplier designed in this article are both 32-bit single-precision floating-point numbers: including 8-bit exponent bit, 1-bit sign bit, and 23-bit mantissa bit. The output floating point result is a 40-bit extended precision floating point number: including 8 exponent bits, 1 sign bit, and 31 mantissa bits. When performing a multiplication operation, first the multiplier and multiplicand mantissas are expanded to 25 bits: including 1 sign bit, 1 hidden bit, and 23 mantissa bits [6].

The mantissa digit of the product is obtained by multiplying the multiplier and the mantissa digit of the dividend and performing normalization and rounding, as shown in equation (1), where Normal is the exponent correction during the mantissa normalization process, and the value of Normal. It is equal to the value of the number of right shifts when normalizing the mantissa (if the mantissa is shifted right by 1 bit to normalize, then Norm a l = 1; if the mantissa is shifted right by 2 bits to normalize, then Normal=2, if the mantissa is already a normalized number, then Normal=0). The exponent c(exp) of the product is equal to the sum of the exponents of the two operands plus the value of the exponent correction bit, as shown in equation (2).

$$[c(man),Normal]=a(man)*b(man) \quad \cdots (1)$$

$$c(exp)=a(exp)+b(exp)+Normal \quad \cdots (2)$$

The mantissa operation part determines the speed of the entire multiplier. Booth coding is a common algorithm for reducing the number of partial products. Considering the delay, area and complexity of the circuit, the multiplier designed in this article adopts the modified Booth algorithm [7]. First, the mantissas of the multiplier and the multiplicand are modified by the Booth coding unit to generate 13 partial products, and then the Wallace tree composed of the compression units of the 3-2 and 4-2 compressors compresses these 13 partial products into Carry and Sum forms. , and finally the product is normalized and rounded during CLA summation, and the exponent of the product is adjusted, as shown in Fig. 1.

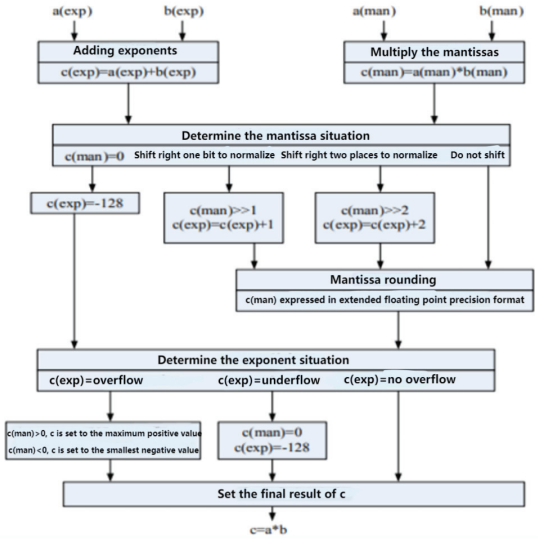


Fig. 1 Floating point multiplier operation flow chart

The flow chart of floating point operations is given in Fig. 1 (taking 32-bit floating point operations as an example).

Step 1: Multiply the mantissa bits of the operands, where the input data is 24 bits and the output result is 48 bits;

Step 2: The exponent part of the operands is added, and the result is c(exp);

Step 3: Determine the mantissa situation. If it is 0, perform step 7 to set the exponent position of the result to -128;

Steps 4 and 5: used to normalize the results;

If it is necessary to shift right by 1 bit for normalization, perform step 8, shift the mantissa to the right, and add 1 to the direct index. If it is necessary to shift right by two digits for normalization, perform step 9, shift the mantissa to the right by two digits, and add 2 to the exponent;

Step 10: Expand the mantissa result to extended-precision floating-point format.

Steps 6 to 11: Determine the exponent situation

If the exponent overflows, proceed to step 14. If the mantissa is greater than 0, set the exponent to the maximum positive number. If the mantissa is less than 0, set the exponent to the minimum negative number.

If the exponent underflows, execute step 15, set the exponent to -128, and the mantissa is 0, if the exponent is within the range. Then proceed to step 16 to obtain the final result.

For floating-point multipliers, the multiplication algorithm and structure are the basis of its hardware implementation. The main steps in floating-point multiplication are: partial product generation, partial product compression, carry propagation addition and rounding processing. Reduction and compression of partial products is the key to distinguishing various multiplication algorithms. And according to existing papers, most of the time, area, and power consumption are consumed in mantissa multiplication. Therefore, when designing, you need to choose an appropriate algorithm based on delay, power, speed, area, complexity and other requirements.

III. Design of Floating Point Multiplier

The mantissa operation part consumes the most resources and has the longest delay, and is the key to the design of floating-point multipliers. The mantissa operation includes three parts: Booth coding to generate partial products, Wallace tree and CLA. The design of this key component is introduced below.

3.1 Modified Booth encoding

Since Booth encoding does not improve the operation speed of the multiplier, the two bits of each overlap check multiplication in the Booth algorithm are extended to three bits of each overlap check, that is the modified Booth algorithm. The modified Booth algorithm encodes each time when checking 3 bits, 2 bits are from the current group and the 3rd bit is from the lowest bit of the higher group. Effectively, the lowest bit of each group is checked 2 times. This modified Booth algorithm can ensure that the partial product is reduced by half, thereby increasing the computing speed and reducing the hardware complexity [8]. This multiplier generates partial products based on the modified Booth algorithm. According to the above formula (3), the value of the partial product can be calculated. For the continuous three-digit data obtained at one time, the lowest bit and the middle bit represent 1, while the highest bit represents -2, and the result is three bits. The sum of the additions, so the possible results after encoding can only be: {0, X, -X, 2X, -2X}. Table 1 is the partial product generation coding table of the modified Booth algorithm :

$$Y = -Y_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} y_i \cdot 2^i + y_{-1} \quad \cdots (3)$$

$$= \sum_{i=0}^{\frac{n-2}{2}} (y_{2i-1} + y_{2i} - 2y_{2i+1}) 2^{2i}$$

Table 1. Modified Booth coding truth table

y_{i-1}	y_i	y_{i+1}	p_i
0	0	0	0
0	0	1	+X
0	1	0	+X
0	1	1	+2X
1	0	0	-2X
1	0	1	-X
1	1	0	-X
1	1	1	0

For the partial product based on the modified Booth coding 16-bit complement form, the dot matrix diagram is shown in Fig. 2. The number of partial products obtained using this encoding method is approximately (n +1)/2. Adding 1 is because it is necessary to ensure that the partial product generated by the highest bit is positive, otherwise it needs to be supplemented for correction. The advantage of the modified Booth algorithm is that it can compress the number of partial products PPi to about 1/2 of the original, and it has nothing to do with the value of the multiplier. Therefore, the number of summations required to complete the sum of partial products is approximately 1/2 of the original number. This can not only improve the operation speed in Floating Point Multiplier, but also reduce the number of adders required. However, it should be noted that the operands in the modified Booth algorithm are expressed in two's complement, and sign extension is required during addition and subtraction operations [9].

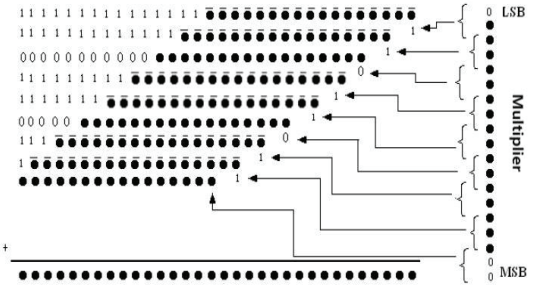


Fig. 2 Booth coded multiplier sign bit extension

3.2 Wallace tree structure

The first step of the Wallace tree structure is to group the partial multiplication integral of each column into a 3-bit group. In order to reduce the number of addends, each group uses a CSA component composed of a full adder; the second step, the results generated in the first step continue to be grouped by 3 bits, and the pseudo-sum and local carry signals of the same weight are processed through the CSA component, thereby reducing the number of addends again until there are only two outputs in the end; the third step, the final pseudo-sum and the local carry are added through the carry-pass adder to obtain the real result.

The number of operands in the above method is reduced by 1.5 times, and the intermediate results will be processed in this way until there are only two outputs in the end.

There are two main structures for fast compression of partial products: Wallace tree and Dadda tree. The Wallace tree compresses as many partial products as possible at each stage and is mainly used to implement high-speed parallel multiplier design [10]; the Dadda tree distributes partial products at each stage for compression thus balancing circuit delay and layout complexity[3]. The multiplier designed in this article uses the Wallace tree structure to implement partial product parallel compression. The depth of the compression tree is $\log_2 N$ levels, where N represents the number of partial product rows.

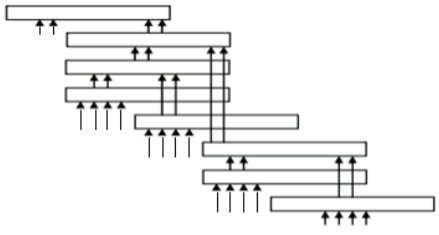


Fig. 3 Schematic diagram of Wallace tree connection structure

Fig. 3 is a schematic diagram of the Wallace tree connection structure, in which each rectangle represents a set of 4:2 compressors, and the line segments with arrows represent partial products and intermediate results to draw conclusions. The Wallace tree structure is theoretically the fastest adder tree for multiplication operations, but its complex connections make layout implementation difficult. Therefore, a 4:2 compressor is needed. Using a 4:2 reducer can reduce the wiring complexity and make the circuit structure regular [11]. The Wallace tree has a large area but small delay. In the Floating Point Multiplier design of this article, the Wallace tree multiplier structure is adopted in order to improve the operation speed.

To improve performance of the multipliers, 4:2 compressor logic is often designed. Fig. 4 is an improved structure [12]. In this structure, the delays from the four inputs to the output are equal, which is convenient for layout and consumes less power. This structure is used in the design, and its logical expression is:

$$S = P_1 \oplus P_2 \oplus P_3 \oplus P_4 \oplus CIN \quad \cdots (4)$$

$$C = \overline{P_1 \oplus P_2 \oplus P_3 \oplus P_4 + (P_1 \cdot P_2 + P_3 \cdot P_4)} \quad \cdots (5)$$

$$C_{OUT} = P_1 \cdot P_2 + P_3 \cdot P_4 \quad \cdots (6)$$

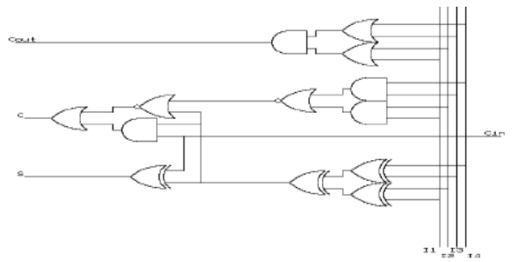


Fig. 4 4:2 Improved structure of compressor

Based on the above design, the circuit diagram of the 32-bit floating point multiplier finally designed in this article is shown in Fig. 5.

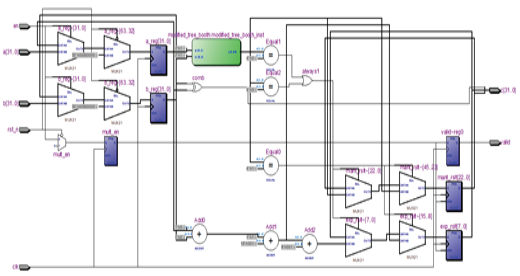


Fig. 5 The modified booth_tree circuit diagram designed in this article

3.3. Carry look-ahead adder

The performance of the adder can be analyzed from the aspects of delay, power consumption, area, etc. The basic idea of improving the speed of the adder is to speed up the generation and transmission time of the carry signal by improving the carry method [13]. The characteristic of the carry look-ahead adder is that carry signals at all levels are generated simultaneously. This parallel approach greatly shortens the time for carry generation. Fig. 6 is the logic diagram of a 4-bit carry look-ahead adder.

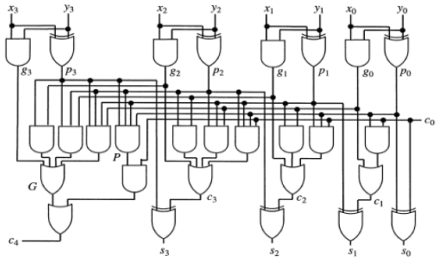


Fig. 6 4-bit carry look-ahead adder

IV. Simulation results and discussion

The floating point multiplier is designed by Verilog HDL and verified and synthesized on Intel Quartus II CAD tool. Te simulation results of the designed floating point multiplier are shown in Fig. 7.

The 32-bit floating point multiplier has been implemented in the Quartus II CAD tool as shown in Figure 7. For inputs a1 = 32'h3f67e1fb; b1 = 32'h3e0208d5; a2 = 32'h3f78e1fb; b2 = 32'h3e0299d5; and the output obtained is c1=32'h3deb9182; c2=32'h3dfdf09f;

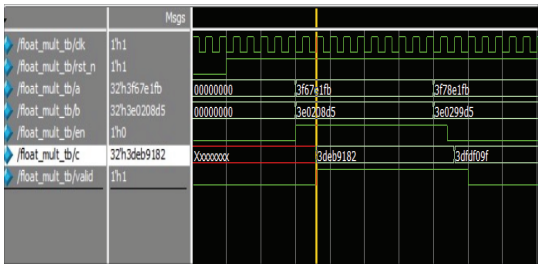
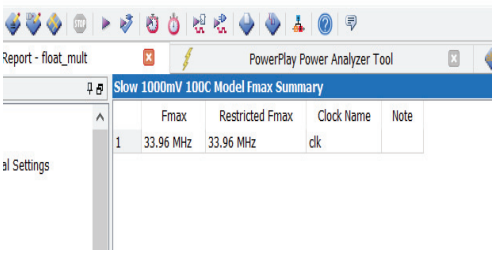
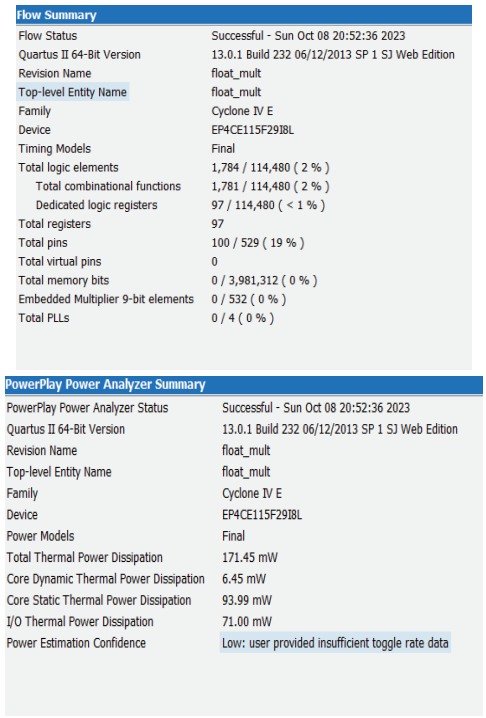


Fig. 7 32-bit floating point multiplier logic simulation results

Fig. 8 shows performance results of speed, power consumption and area of the designed floating point multiplier using modified Booth encoding with the Wallace tree. And the performance comparison results are shown in Table 2. We implement 4-types of floating point multipliers. And Booth means Booth encoding, Modified Booth means Modified Booth encoding, Booth Tree means Booth encoding using the Wallace tree, and Modified_booth_tree means Modified Booth encoding using the Wallace tree, respectively.





Flow Summary	
Flow Status	Successful - Sun Oct 08 20:52:36 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	float_mult
Top-level Entity Name	float_mult
Family	Cyclone IV E
Device	EP4CE115F29I8L
Timing Models	Final
Total logic elements	1,784 / 114,480 (2 %)
Total combinational functions	1,781 / 114,480 (2 %)
Dedicated logic registers	97 / 114,480 (< 1 %)
Total registers	97
Total pins	100 / 529 (19 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

PowerPlay Power Analyzer Summary	
PowerPlay Power Analyzer Status	Successful - Sun Oct 08 20:52:36 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	float_mult
Top-level Entity Name	float_mult
Family	Cyclone IV E
Device	EP4CE115F29I8L
Power Models	Final
Total Thermal Power Dissipation	171.45 mW
Core Dynamic Thermal Power Dissipation	6.45 mW
Core Static Thermal Power Dissipation	93.99 mW
I/O Thermal Power Dissipation	71.00 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Fig. 8 The results of area, speed, and power of the designed floating point multiplier

From Table 2, modified booth encoding with wallace tree is the best. It has a small area, low power consumption and the fastest computing speed. The FMAX of modified Booth encoding with Wallace tree is 33.96 Mhz, which is 2.04 times faster than the booth encoding, 1.62 times faster than the modified booth encoding, 1.04 times faster than the booth encoding with wallace tree. Furthermore, compared to modified booth encoding, the area of modified booth encoding with wallace tree is reduced by 24.88%, and power consumption of that is reduced by 2.5%.

Table 2. Comparison of implementation results of various floating point multipliers

Performance metrics	LE	REG	FMAX (Mhz)	Power (mW)
booth	2397	97	16.63	171.52
modified_booth	1736	97	21	170.02
booth_tree	2375	97	32.65	175.93
modified_booth _tree	1784	97	33.96	171.45

Ⅴ. Conclusion

In this paper, we design a 32-bit floating point multiplier, uses the modified Booth encoding with Wallace tree to complete partial product compression in the form of carry and sum, and then uses a carry look-ahead adder to obtain the final result. We compared different floating point multipliers: Booth encoding, modified Booth encoding, Booth encoding with Wallace tree, and modified Booth encoding with Wallace tree. Simulation results from the FPGA CAD tool show that the floating-point multiplier using modified Booth encoding with the Wallace tree is the fastest, and is expected to be widely used in high-speed DSP applications.

References

[1] S. Kim, H. Seo, S. Kim, and D. Kim, "Approximate Multiplier With Efficient 4-2 Compressor and Compensation Characteristic," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 17, no. 1, 2022, pp. 173-180.

[2] H. Seo and D. Kim, "Approximate multiplier with high density, low power and high speed using efficient partial product reduction," *J. of*

the Korea Institute of Electronic Communication Sciences, vol. 17, no. 4, 2022, pp. 671-678.

[3] J. Kim and S. Lee, "High-Performance Multiplier Using Modified Gate-Diffusion Input (m-GDI) Compresso," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 18, no. 2, 2023, pp. 285-290.

[4] A. Akkas and M. J. Schulte, "A quadruple precision and dual double precision floating-point multiplier," In *Proc. Euromicro. Symp. on Digital System Design,* Belek-Antalya, Turkey, Sept. 2003, pp. 76-81.

[5] S. V. Siddamal, R. M. Banakar, and B. C. Jinaga, "Design of High-Speed Floating Point Multiplier," In *Proc. 4th IEEE Int. Symp. on Electronic Design, Test and Applications (delta 2008),* Hong Kong, China, Jan. 2008, pp. 285-289.

[6] IEEE Std. 754-1985, *IEEE Standard for Binary Floating-point Arithmetic*. IEEE, New York, NY, 1985.

[7] T. Krishnan and S. Saravanan, "Design of Low-Area and High Speed Pipelined Single Precision Floating Point Multiplier," In *Proc. 2020 6th Int. Conf. (ICACCS). Communications,* Coimbatore, India, Mar. 2020, pp. 1259-1264.

[8] Y. Guo, H. Sun, and S. Kimura, "Small-Area and Low-Power FPGA-Based Multipliers using Approximate Elementary Modules," In *Proc. 2020 25th Asia and South Pacific Design Automation Conf. (ASP-DAC). Communications,* Beijing, China, 2020, pp. 599-604.

[9] X. Jiang, P. Xiao, M. Qiu, and G. Wang, "Performance effects of pipeline architecture on an FPGA-based binary 32 floating point multiplier," *J. of Microprocessors and Microsystems*, vol. 37, no. 8, 2013, pp. 1183-1191.

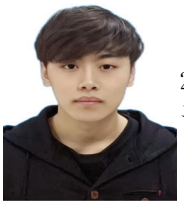
[10] M. J. Rao and S. Dubey, "A high speed and area efficient Booth recoded Wallace tree multiplier for fast arithmetic circuits," In *Proc. 2012 Asia Pacific Conf. on Postgraduate Research in Microelectronics and Electronics, Communications*, Hyderabad, India, Dec. 2012, pp. 220-223.

[11] Z. Zhao and Z. Line, "Design of an improved Wallace tree multiplier," *J. of Electronic Design Applications*, no. 8, 2006, pp. 113-116.

[12] S. Yuan and C. Zhang, "A Design of High-Speed 4-2 Compressor for Fast Multiplier," *J. of Microelectronics & Computer*, vol. 19, no. 4, 2002, pp. 53-56.

[13] K. Thiruvenkadam, J. Ramesh, and A. S. Pillai, "Area-efficient dual-mode fused floating-point three-term adder," *J. of Circuits, Systems, and Signal Processing*, vol. 38, no. 1, 2019, pp.173-190.

저자 소개



Xuhao Zhang

2023년 전남대학교 전자통신공학과 졸업 (공학석사)

※ 관심분야 : 저전력/고집적/고속 회로설계

김대익(Dae-Ik Kim)

1991년 전북대학교 전자공학과 졸업(공학사)
1993년 전북대학교 대학원 전자공학과 졸업(공학석사)
1996년 전북대학교 대학원 전자공학과 졸업(공학박사)
2002년~현재 전남대학교 전자통신공학과 교수
※ 관심분야 : VLSI 설계, 저전력 회로설계