Proceedings of International Conference on Mechanical, Electrical and Medical Intelligent System 2017

Study on Digital Multiplier Architecture Using Square Law and Divide-Conquer Method

Yifei Sun[1,a], Shu Sasaki[1,b], Dan Yao[1,c], Nobukazu Tsukiji[1,d], Haruo Kobayashi[1,e]

[1] Division of Electronics and Informatics, Gunma University, 1-5-1 Tenjincho, Kiryu-shi, Gunma, 376-8515, Japan

[a]<t172d004@gunma-u.ac.jp>, [b]<t15804040@gunma-u.ac.jp>, [c]<yao_dan@outlook.com>, [d]<ntsukiji@gunma-u.ac.jp>, [e]<koba@gunma-u.ac.jp>

Keywords: digital multiplier, square law, divide and conquer method, digital circuit, FPGA

Abstract. In this paper, we study digital multiplier architecture using a square law for obtaining the product AB from the sum and square of the inputs A and B and a Divide & Conquer method for small circuit implementation. We have designed them at the register transfer level (RTL) to confirm its operation. We have investigated the squaring calculation circuit with look-up table (LUT) and also direct squaring calculation logic. We show that in case of the squaring law usage, the Divide & Conquer method can be utilized in both cases of squaring calculation circuits with LUT and direct logic, and it can reduce the circuit. The digital multiplier is widely used for digital computers and DSP chips. When it is realized directly, a two-dimensional array of full adders is required; as the number of bit increases, its circuit size and power become large and its computation time is also increased. The investigated architecture is expected to solve these problems.

1. Introduction

Digital multipliers are widely used for digital computers and DSP chips as well as MPU. Since the multiplication of binary numbers is performed by adding of binary numbers repeatedly, a large amount of calculation is required. If the digital multiplier is realized directly, it becomes a two-dimensional array of full adders [1] (Fig. 1, Fig. 2); there is a problem that the circuit size, power consumption and operation time become large [2]. Therefore, various algorithms and architectures have been proposed to solve these problems for many years. Based on these, digital multipliers have been designed and realized.

However, the digital multiplier architecture and algorithm are still important research areas even now. In digital communication systems, massive digital computation in real time is required; if we can realize small scale digital multipliers, many of them can be mounted and they can perform parallel operation.

Here we consider using the following two equations [3, 4] for calculating the product AB from the sum and square of the two digital inputs A and B.

$$AB = \frac{1}{4}\{(A+B)^2 - (A-B)^2\} \tag{1}$$

$$AB = \frac{1}{2}\{(A+B)^2 - A^2 - B^2\} \tag{2}$$

Then we show that for squaring operation, the Divide & Conquer method can be applied which reduces the circuit size. We consider that squaring and addition/subtraction with the Divide & Conquer method are simple, compared to the direct multiplication.

In this paper, we compare our investigated architectures and algorithms for digital multiplier with the direct implementation using a 2-dimensional array of full adders (Fig. 1, Fig. 2), because there are many architectures and algorithms such as Booth algorithm and Wallace tree configuration, and hence the direct implementation would be suitable as a reference.

In this paper, we will show the following: we investigate the architecture and algorithm in Eq. (1).

①If the squaring is implemented with logic circuit, the circuit size is comparable to the direct implementation.

②If the squaring is implemented with Look-up tables (LUTs), their sizes are large and speed may be slow for a large number of input data bits.

③However, if the Divide & Conquer method is applied, the LUT sizes reduce drastically. Eq. (2) plays an important role there.

④ If the Divide & Conquer method is applied for the dedicated logic implementation of squaring operation, the circuit size is reduced by 2/3. There, Eq. (2) plays an important role again.

If the Divide & Conquer method is applied repeatedly, the hardware can be reduced further. We have performed register transfer level (RTL) simulation and confirmed the validity of the investigated algorithms and architectures.

2. LUT AND MULTIPLIER

2.1 Look-up Table (LUT)

The LUT is a memory (RAM or ROM), and its input is memory "address", while its output is memory "data" (Fig.3). By storing the calculation data in the memory, a desired calculation result for the input specified by "address" can be obtained as its output provided by "data" [5].

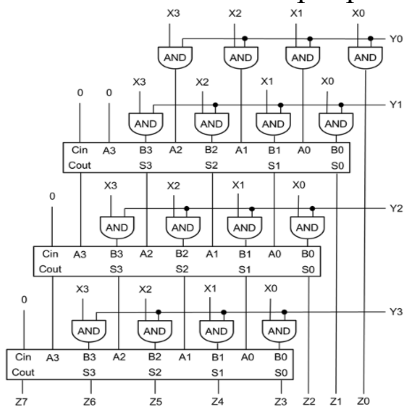


Fig. 1. 4-bit x 4-bit digital multiplier with a 2-dimensional array of full adders (direct implementation as a reference)

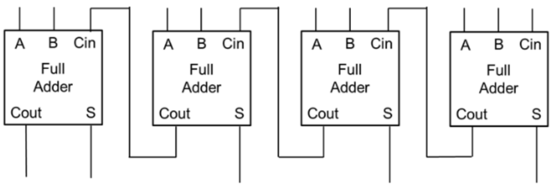


Fig. 2. 4-bit ripple carry adder used in Fig. 1 as a reference

X
address
LUT
data
F(X)

Fig. 3. Look-up table (LUT)

2.2 Multiplication Algorithm using Logarithm and Exponential Functions

We consider to compute the multiplication using logarithm and exponential LUTs in Fig. 4. If we calculate AB for the two data A and B, we will use an adder and LUTs as follows:

① Using logarithm data LUT to obtain logA and logB.

② Using adder to calculate logA+ logB (=logAB).

③ Using exponential data LUT to obtain AB from logAB.

However, in order to obtain logarithm and exponential data with high precision, the LUT needs large number of data bits and then its size becomes large and its operation becomes slow. Hence we exclude this algorithm here.

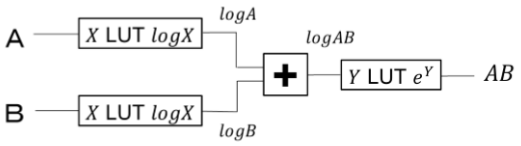


Fig. 4 Multiplier with logarithm and exponential LUTs

3. Multiplication Algorithm using square law

In this section, the square law of Eq. (1) and Eq. (2) was examined. Multiplication by 1/2 or 1/4 can be realized by one or two-bit right shift operation (actually only wiring change is enough). The square calculation uses LUT or logic circuit. Both of them can achieve the purpose for circuit size and power consumption reduction as well as high speed operation.

3.1 Multiplier Using Square Law and LUT

Fig. 5 shows the circuit configuration to realize Eq. (1), where two LUTs are used. Fig. 6 shows the circuit configuration to realize Eq. (2), where three LUTs are used.

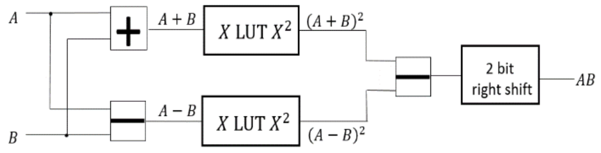


Fig. 5 Multiplier configuration for realizing square law equation (1) using LUTs

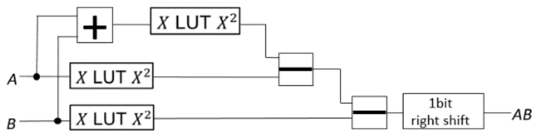


Fig. 6 Multiplier configuration for realizing square law equation (2) using LUTs

Considering the calculation time balance in each path, the circuit configuration in Fig. 7 also can be conceivable. Alternatively, one LUT can be used sequentially to perform calculations of A^2, B^2 and $(A+B)^2$ as shown in Fig. 8, and there although the computation time becomes about three times as large. Although the circuit amount can be reduced by one-third[5-7], but because of this architecture needs some registers or memory to store the previous LUT data, the circuit size still large.

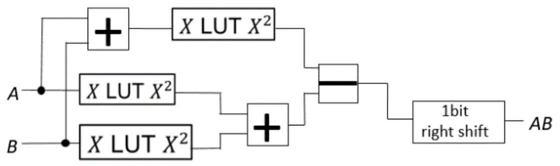


Fig. 7 Circuit that considering balance of calculation time

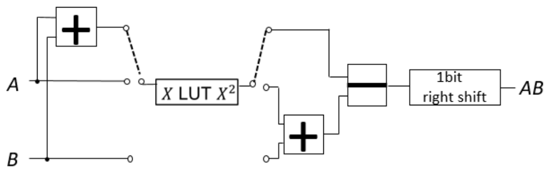


Fig. 8 Circuit that sequentially uses one LUT

For N-bit x N-bit multiplication LUT, its address is N-bit and its data is 2N-bit. Then the LUT size is 2^N x (2N). When N=8, the LUT size is 256 x 16=4096 bits (Fig. 9). When N=4, the LUT size is 16 x 8=128 bits (Fig. 10). Then we see that if N is reduced by a factor of 1/2, the LUT size is reduced by a factor of 1/32.

Note that for a large number of N, the LUT size is large and its speed may be slow; hence this implementation may not be efficient. However, for a small number of N, its size is reduced significantly and also its access speed may be much faster, and this implementation is efficient.

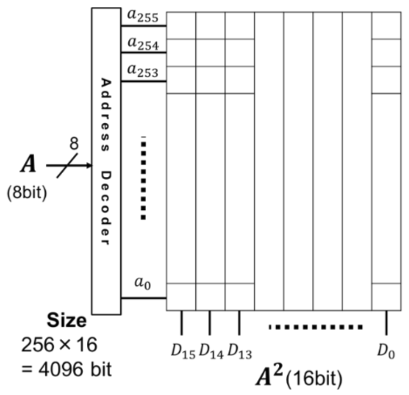


Fig. 9 LUT for 8-bit x 8-bit squaring calculation

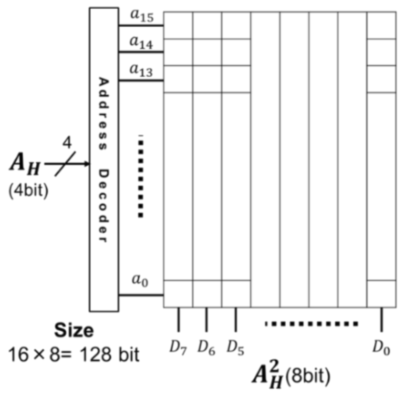
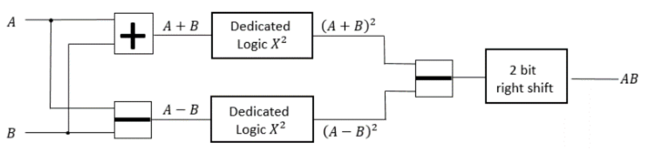


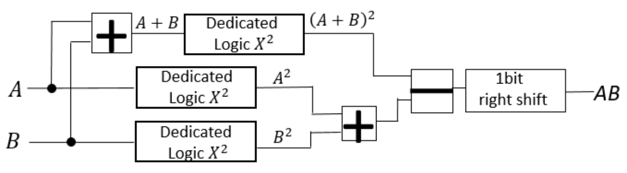
Fig. 10 LUT for 4-bit x 4-bit squaring calculation

3.2 Multiplier Using Square Law and Dedicated Logic

The squaring calculation circuit can be realized by the LUT. If the larger number of bits was handled, the memory size must be increased. For this reason, we have examined a dedicated circuit using the truth table of squaring. Fig. 11 shows its circuits based on Eq. (1), Eq.(2).



(a)



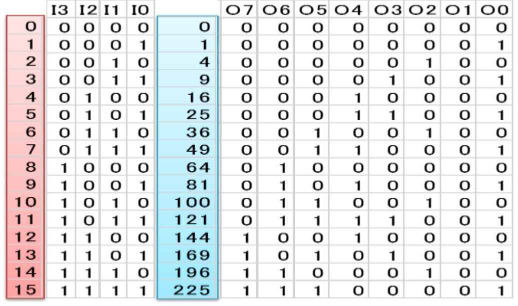
(b)

Fig. 11 Circuit using squaring operation logic circuit.(a) Based on Eq.(1). (b) Based on Eq.(2)

Here is the square operation logic ciruit, for example in 4-bit x 4-bit case, its output is 8-bit, the following equations are logic expressions obtained by the truth table in Table 1.

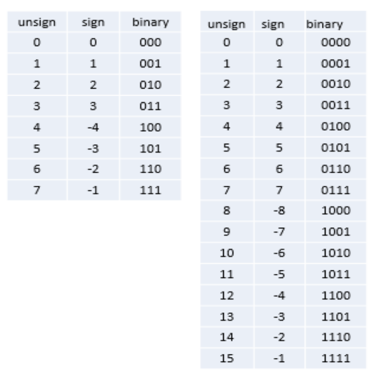
$$
\begin{gathered}
O0 = I0 \\
O1 = 0 \\
O2 = I1\overline{I0} \\
O3 = (I2 \oplus I1)I0 \\
O4 = \overline{I3}I2(\overline{I1} + I0) + I3\overline{I2}I0 + I3I2\overline{I1}\overline{I0} \\
O5 = (I3 \oplus I2)I1 + I3I2I0 \\
O6 = I3\overline{I2} + I3I2I1 \\
O7 = I3I2 \qquad (3)
\end{gathered}
$$

Table 1: Truth table of square (in 4-bit x 4-bit case)



	I3	I2	I1	I0		O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0	0	1
2	0	0	1	0	4	0	0	0	0	0	1	0	0
3	0	0	1	1	9	0	0	0	0	1	0	0	1
4	0	1	0	0	16	0	0	0	1	0	0	0	0
5	0	1	0	1	25	0	0	0	1	1	0	0	1
6	0	1	1	0	36	0	0	1	0	0	1	0	0
7	0	1	1	1	49	0	0	1	1	0	0	0	1
8	1	0	0	0	64	0	1	0	0	0	0	0	0
9	1	0	0	1	81	0	1	0	1	0	0	0	1
10	1	0	1	0	100	0	1	1	0	0	1	0	0
11	1	0	1	1	121	0	1	1	1	1	0	0	1
12	1	1	0	0	144	1	0	0	1	0	0	0	0
13	1	1	0	1	169	1	0	1	0	1	0	0	1
14	1	1	1	0	196	1	1	0	0	0	1	0	0
15	1	1	1	1	225	1	1	1	0	0	0	0	1

Table 2: Signed binary representation



A or B are 3 bits situation

unsign	sign	binary
0	0	000
1	1	001
2	2	010
3	3	011
4	-4	100
5	-3	101
6	-2	110
7	-1	111

A+B or A-B are 4 bits situation

unsign	sign	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	-8	1000
9	-7	1001
10	-6	1010
11	-5	1011
12	-4	1100
13	-3	1101
14	-2	1110
15	-1	1111

From Table 1, we can found the O1, i.e. the second bit, is always 0 which contribute the reduction of circuits. We also have investigated the comparison of the multiplier AB with the direct logic implementation (Fig. 1, Fig. 2) and the squaring circuit A^2 with the logic implementation quantitatively. We have found that the squaring circuit A^2 is almost half of the multiplier AB. See Appendix B. Hence the total size of the circuit based on Eq. (1) is almost the same as that of the reference multiplier in Fig. 1 if they are implemented directly with logic circuits. Then we need the Divide & Conquer method for the circuit size reduction, which will be discussed in the next section.

3.3 Usage of Absolute Value for Squaring Calculation

Let us consider to handle negative numbers as well as positive numbers and zero for the multiplier. Then we remark that first taking its absolute value and then calculating its squaring reduce the LUT and logic circuit size.

For example, the quarter square multiplication technique is easily demonstrated algebraically as

$$AB = \frac{1}{2}\{(A+B)^2 - A^2 - B^2\} \tag{2}$$

The number of addition and subtractions is 3. Consider the calculation in case of negative numbers. We convert negative numbers to their absolute values, and then calculate their squares. As shown in Table 2, the highest bit (the most significant bit) is the sign bit; if A or B are in 3-bit, $(A+B)$ are between -8 to 6, and $(A-B)$ are between -7 to 7. If $(A+B)$ or $(A-B)$ are negative, we reverse every bit, and then add one to it (i.e., we obtain its two complement). Then we have its absolute value and perform the squaring operation to it. If $(A+B)$ or $(A-B)$ are positive, we directly use squaring operation to it. Fig. 12 shows their circuit realization. This structure reduces the hardware whether it were implemented with LUTs or dedicated logic.

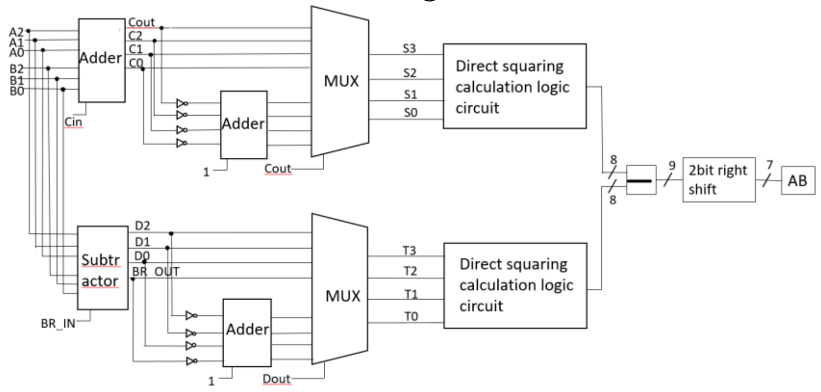


Fig. 12 Multiplier using quarter square law (3-bit x 3-bit)

4. Divide & Conquer Method

4.1 Two Divide & Conquer Algorithms

Let us consider the case that A is 8-bit, and its higher 4-bit is denoted as A_H, where its lower 4-bit is denoted as A_L (Fig.13). Then A^2 were expressed by the following:

$$A^2 = A_H^2(8bit\ \ left\ \ shift) + 2A_HA_L(4bit\ \ left\ \ shift) + A_L^2 \tag{4}$$

Also we have the following from Eq. (2):

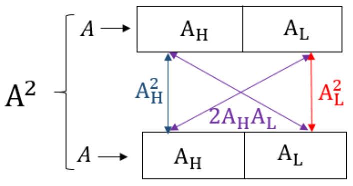
$$2A_HA_L = (A_H + A_L)^2 - A_H^2 - A_L^2 \tag{5}$$

Then it follows from Eq. (4), Eq. (5) that

$$A^2 = (A_H)^2(8bit\ \ left\ \ shift) + \{(A_H + A_L)^2 - A_H^2 - A_L^2\}(4bit\ \ left\ \ shift) + (A_L)^2 \tag{6}$$

The first method use equation (4), and the second method uses equation (6).

Then Fig. 14 (a), (b) show the squaring calculation circuit (A(8bit) $\rightarrow$ A^2(16bit)) based on the first and second methods respectively. 8-bit A is divided into higher 4-bit and lower 4-bit, and each is calculated and shifted appropriately and then all were added. Here bit shifts were realized only with proper interconnection arrangement (no hardware overhead).



$$A^2 = (A_H)^2(8bit\ left\ shift) + 2A_HA_L(4bit\ left\ shift) + (A_L)^2$$

Fig.13 Data (A) division into higher bits (A_H) and lower bits (A_L)

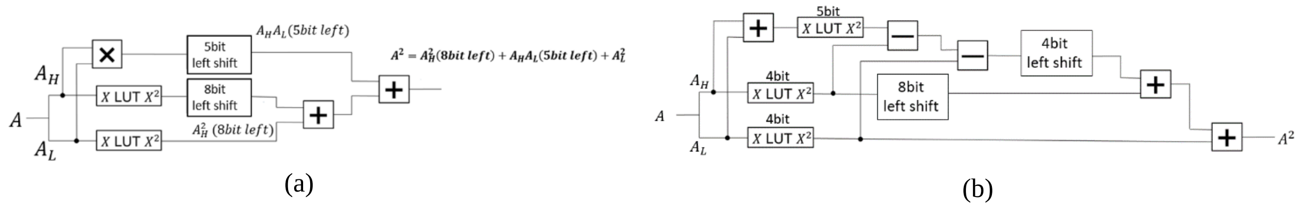


Fig. 14 Squaring calculation with the divide & conquer method. (a) First method. (b) Second method

Now let us consider 8bit data, $A = 11001001 = (201)_{10}$. Divide A into higher 4-bit (A_H) and lower 4-bit (A_L).

$A_H = 1100 \quad (12)_{10}$

$A_L = 1001 \quad (9)_{10}$

Then we proceed the calculation.

$A_H^2 = 10010000 \quad (144)_{10}$

$A_H^2\ (8bit\ left\ shift) = 1001000000000000 \quad (36864)_{10}$

$A_L^2 = 1010001 \quad (81)_{10}$

$A_H + A_L = 00010101 \quad (21)_{10}$

$(A_H + A_L)^2 = 110111001 \quad (441)_{10}$

$\{(A_H + A_L)^2 - A_H^2 - A_L^2\}\ (4bit\ left\ shift)$

$= 110110000000 \quad (3456)_{10}$

$(A_H)^2\ (8bit\ left\ shift) + \{(A_H + A_L)^2 - A_H^2 - A_L^2\}\ (4bit\ left\ shift) + (A_L)^2$

$= 1001000000000000 \quad (36864)_{10} + 110110000000 \quad (3456)_{10} + 1010001 \quad (81)_{10}$

$= 1001110111010001 \quad (40401)_{10} = A^2$

Then we see that the value obtained by the Divide & Conquer method and the direct calculated value of A^2 are the same, and the validity of the Divide & Conquer is shown in the above.

These divided bit streams can be divided further, and the Divide and Conquer can be applied repeatedly.

4.2 Effectiveness of Divide & Conquer Method for Squaring with LUTs

As Fig. 9, Fig. 10 shows, the LUT size for 8-bit A requires 4096 bits, whereas that for 4-bit is 128-bit, which is 1/32 of 8-bit case. In case of the Divide & Conquer second method in Fig. 14 (b), 3 LUTs are used and the size of each LUT is reduced by 1/32. Then the total LUT size is 3/32 compared to the LUT size without the Divide & Conquer method. Also note that the speed of the small sized LUT access time is much faster.

For a general N-bit A case, the total LUT size is 2^N x $(2N)$ without the divide and conquer method, whereas that is $2^{\frac{N}{2}} \times (2 \times \frac{N}{2}) \times 3$. Then the reduction of $\left[2^{\frac{N}{2}} \times \left(2 \times \frac{N}{2}\right) \times 3\right] \div [2^N \times 2N] = \frac{3}{2} \times 2^{-\frac{N}{2}}$ is obtained.

We see the Divide & Conquer method is very effective.

4.3 Effectiveness of Divide & Conquer Method for Squaring with Dedicated Logic

Let us consider to calculate the right terms with direct calculation or dedicated logic.

$$AB = \frac{1}{2}\{(A+B)^2 - A^2 - B^2\} \qquad (2)$$

The numbers of the full adders are almost the same, because the square calculation $(\mathrm{A}+\mathrm{B})^2$ or $(\mathrm{A}-\mathrm{B})^2$ needs a half of the direct multiplication AB and Eq. (2) requires two square calculations $(\mathrm{A}+\mathrm{B})^2$ and $(\mathrm{A}-\mathrm{B})^2$.

Now let us consider to use the Divide & Conquer second method. Let

$$C = A + B$$

For each square calculation of the following requires 1/4 of direct calculation C^2.

$$(C_H)^2 \quad , \ (C_L)^2, \ (C_H + \ C_L)^2$$

Then using Eq. (6) from the above 3 terms, we have C^2 with 3/4 of the direct calculation.

5. RTL Design and Simulation

To verify the algorithm and validity of the circuit configuration, Verilog HDL circuit simulation was carried out. Specifically, we have realized the circuit configuration on simulation software, changed the two input values and calculated the output results. Then we checked whether the result was correct or not.

We have used the second Divide & Conquer method, i.e. the following equation (7).

$$A^2 = (A_H)^2(8bit\ \ left\ \ shift) + \{(A_H + A_L)^2 - A_H^2 - A_L^2\}(4bit\ \ left\ \ shift) + (A_L)^2 \qquad (7)$$

If the inputs A, B are 4-bit x 4-bit and the output AB is 8-bit, there are 16 x 16 (=256) combinations. If the inputs are 8-bit x 8-bit and the output is 16-bit, there are 256 x 256 (=65536) combinations. If the inputs are 16-bit x 16-bit and the output is 32-bit, there are 65536 x 65536 (=4294967296) combinations. In all these numerical values, the proposed algorithm was confirmed that the multiplication was correct.

In dedicated circuit using the truth table of squaring situation, implement the circuit configuration shown in Fig. 11(b) on the simulation software. The inputs are 4-bit x 4-bit and the output is 8-bit. We changed two input values, calculated and outputted the result. Then we checked whether the result is correct or not; the result proved its correctness.

In case of using absolute value for squaring calculation, the hardware was implemented with dedicated logic circuit. In the situation of inputs 3-bit x 3-bit, 6-bit x 6-bit and 8-bit x 8-bit, the results were also proved to be correct.

Simulation results are shown in Appendix A. With this program, the proposed algorithm can be implemented on FPGA. This time, we implemented 4-bit x 4-bit circuit (second method of Eq. (2)), 4-bit x 4-bit circuit (dedicated logic of Eq. (2)) and 3bit x 3bit circuit (absolute value of Eq. (1)) by using Spartan 3E FPGA and confirmed the operation.

6.Conclusion

We have investigated the square law algorithms with the Divide & Conquer methods to realize digital multipliers. We propose two Divide & Conquer methods, and show that one of them was very effective. If the squaring was implemented with LUTs, their size were reduced significantly and its access time becomes faster. If the squaring was implemented with dedicated logic, the size was reduced by 3/4. If the Divide & Conquer method were applied repeatedly, the hardware is expected to reduce further.

We have examined its hardware implementation and confirmed its operation by RTL simulation for FPGA implementation.

We will focus on the following as future works:

① Quantitative evaluation of the proposed circuit amount.

② Clarification of calculation precision, arithmetic unit and number of bits in LUT.

③ Clarification of implementation FPGA operation clock frequency and calculation speed.

④ Bit division for Eq. (1).

All digital multipliers are expressed in binary number, when there is minus situation, it expresses minus by two's complement. We have considered how to deal with minus number, although consideration is necessary for bit division, we will discuss it in the future.

Acknowledgements

The authors would like to thank Prof. Shugang Wei and Prof. Hiroyuki Makino, Prof. Yasushi Yuminaka, Mr. Junshan Wang, Dr. Congbing Li, Mr. Shohei Shibuya and Mr. Takuya Arafune for valuable suggestions.

Appendix A

RTL simulation of digital multiplier using the investigated method is shown.

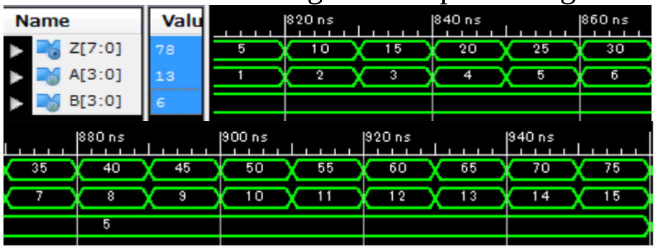


Fig. A1 4-bit x 4-bit simulation (using the second Divide & Conquer method)

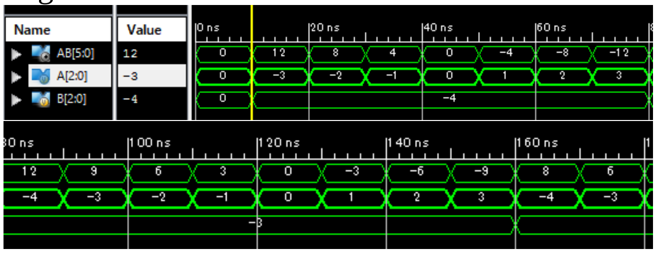


Fig. A3 Quarter square multiplication circuit (3-bit x 3-bit) simulation (equation (1))

The input values A and B were changed every 10ns and every 160ns, and the calculation result in that section was displayed on the waveform. In Fig. A1, the value of the cursor position in the simulation result were displayed. Here A=13 B=6 C=78. All these calculations were done in binary numbers. For the sake of clarity, the results were displayed in decimal.

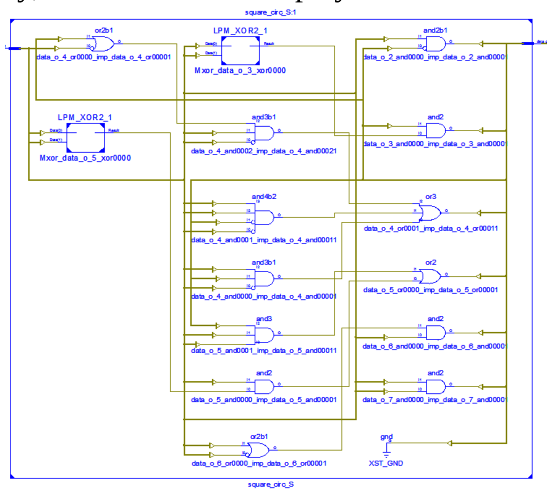
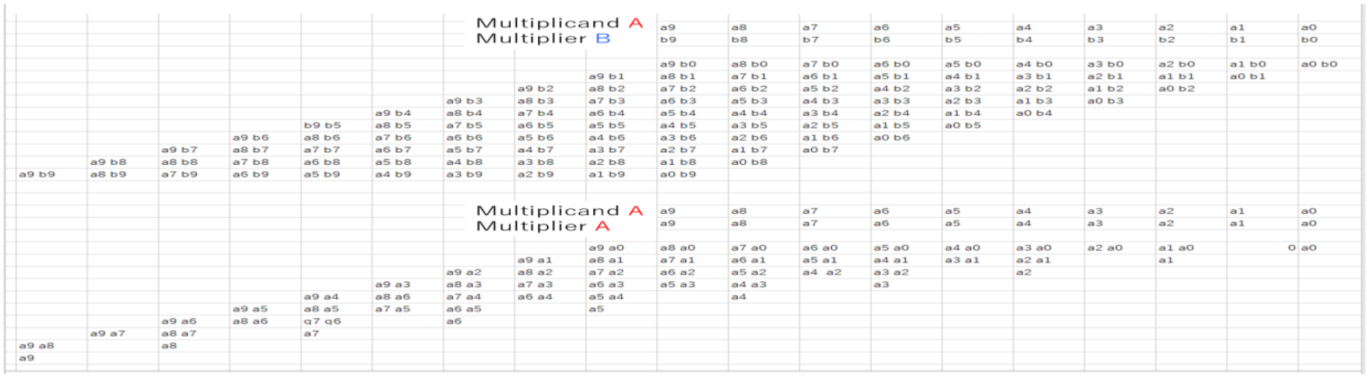


Fig. A2 Square calculation logic circuit (4-bit x 4-bit) simulation (equation (2))

As showing in Fig. A3, the input circuit program is for 3-bit x 3-bit. The input values A and B were changed every 10ns and every 70ns. The calculation results were displayed on the waveform. Here A= -3 B= -4 AB=12. The calculations were done in binary numbers. It was shown that the algorithm studied by this can be reflected on the circuit.

Appendix B

Multiplication AB and square A^2 calculations in 10-bit case is shown in Fig. B. We see that the number of full adders for Square A^2 is about a half of that for multiplication AB.



						Multiplicand A			a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
						Multiplier B			b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
									a9 b0	a8 b0	a7 b0	a6 b0	a5 b0	a4 b0	a3 b0	a2 b0	a1 b0	a0 b0
								a9 b1	a8 b1	a7 b1	a6 b1	a5 b1	a4 b1	a3 b1	a2 b1	a1 b1	a0 b1	
							a9 b2	a8 b2	a7 b2	a6 b2	a5 b2	a4 b2	a3 b2	a2 b2	a1 b2	a0 b2		
						a9 b3	a8 b3	a7 b3	a6 b3	a5 b3	a4 b3	a3 b3	a2 b3	a1 b3	a0 b3			
					a9 b4	a8 b4	a7 b4	a6 b4	a5 b4	a4 b4	a3 b4	a2 b4	a1 b4	a0 b4				
				b9 b5	a8 b5	a7 b5	a6 b5	a5 b5	a4 b5	a3 b5	a2 b5	a1 b5	a0 b5					
			a9 b6	a8 b6	a7 b6	a6 b6	a5 b6	a4 b6	a3 b6	a2 b6	a1 b6	a0 b6						
		a9 b7	a8 b7	a7 b7	a6 b7	a5 b7	a4 b7	a3 b7	a2 b7	a1 b7	a0 b7							
	a9 b8	a8 b8	a7 b8	a6 b8	a5 b8	a4 b8	a3 b8	a2 b8	a1 b8	a0 b8								
a9 b9	a8 b9	a7 b9	a6 b9	a5 b9	a4 b9	a3 b9	a2 b9	a1 b9	a0 b9									
						Multiplicand A			a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
						Multiplier A			a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
								a9 a0	a8 a0	a7 a0	a6 a0	a5 a0	a4 a0	a3 a0	a2 a0	a1 a0	0	a0
							a9 a1	a8 a1	a7 a1	a6 a1	a5 a1	a4 a1	a3 a1	a2 a1		a1		
						a9 a2	a8 a2	a7 a2	a6 a2	a5 a2	a4 a2	a3 a2		a2				
					a9 a3	a8 a3	a7 a3	a6 a3	a5 a3	a4 a3		a3						
				a9 a4	a8 a6	a7 a4	a6 a4	a5 a4		a4								
			a9 a5	a8 a5	a7 a5	a6 a5		a5										
		a9 a6	a8 a6	q7 q6		a6												
	a9 a7	a8 a7		a7														
a9 a8		a8																
a9																		

Fig. B Multiplication AB and SquareA^2calculations in 10-bit

References

[1] A. V. Oppenheim, R.W. Shafer, Digital Signal Processing, Printice-Hall, Englewood Cliffs, NJ, 1975, pp. 56.

[2] K. Gentile, and R Cushing, A Technical Tutorial on Digital Signal Synthesis, Analog Devices, Inc. 1999, pp.78.

[3] N. Weste, D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Addison Wesley, 2010. Pp.125-126.

[4] E. L. Johnson, "A Digital Quarter Square Multiplier, " ***IEEE Trans. on Computers,*** Vol. C-29, No. 3, pp.258-261, March 1980.

[5] S. Sasaki, H. Kobayashi, "Study of Computation Architecture for Short-Time Spectrum Analysis, " The 5th Technical Meeting of IEEJ Tochigi Gunma Branch, Utsunomiya, March 2015.

[6] S. Sasaki, H. Kobayashi, "Study of Digital Multiplier Algorithm Using Addition and Square Formula, " The 38th Mul-valued Logic Forum, Sapporo, Japan, Sept. 2015.

[7] S. Sasaki, H. Kobayashi, "Study of Digital Multiplier Algorithms Using a Square Law and Its FPGA Implementation, " IEICE Signal Processing Workshop, Chiba, Japan, Aug. 2016.