Study on Digital Multiplier Architecture Using Square Law

群馬大学大学院　理工学府
電子情報・数理教育プログラム
孫　逸菲 (Sun Yifei, ソン・イッヒ)

群馬大学
GUNMA UNIVERSITY

OUTLINE

OUTLINE

Research Background

Digital arithmetic devices
- ・Adder ・Subtractor
- ・Multiplier ・Divider

DSPs, μ Processors use several digital multipliers on a chip.

↓ Requirements

Small scale
Low power
High speed

● Digital multiplier hardware implementation algorithm has been a research topic for 50 years.

● Decrease of the multiplier scale is still a research topic .

How Digital Multiplier Works

Decimal

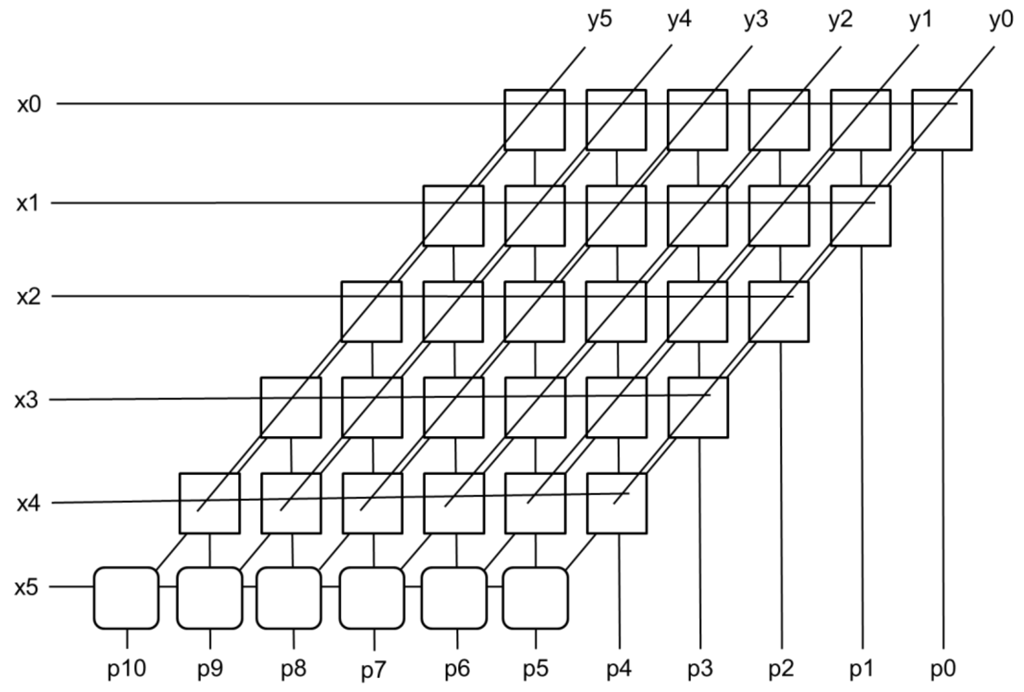
```
      2 5    multiplicand
    x 3 9    multiplicator
    -----
      4 5  ┐
    1 8    │
  1 5      ├  Partial products
+ 6        ┘
---------
  9 7 5        product
```

Binary

```
              0 1 1 0 0 1 : 25₁₀   multiplicand
            x 1 0 0 1 1 1 : 39₁₀   multiplicator
            ---------------------
              0 1 1 0 0 1  ┐
            0 1 1 0 0 1    │
          0 1 1 0 0 1      │  Partial products
        0 0 0 0 0 0        │
      0 0 0 0 0 0          │     AND gate
  + 0 1 1 0 0 1            ┘
  ---------------------------
  0 0 1 1 1 1 0 0 1 1 1 1 : 975₁₀   product
```

Calculation of the sum of partial products increases

Purpose of Study



Composition of array digital multiplier

Multiplier (Using 2D array of full adders)

- Circuit size
- Power
- Computation time

→ BIG

Ex: In 6bit ✕ 6bit situation

6 ✕ 6 = 64 full adders are needed

↓

Reduce

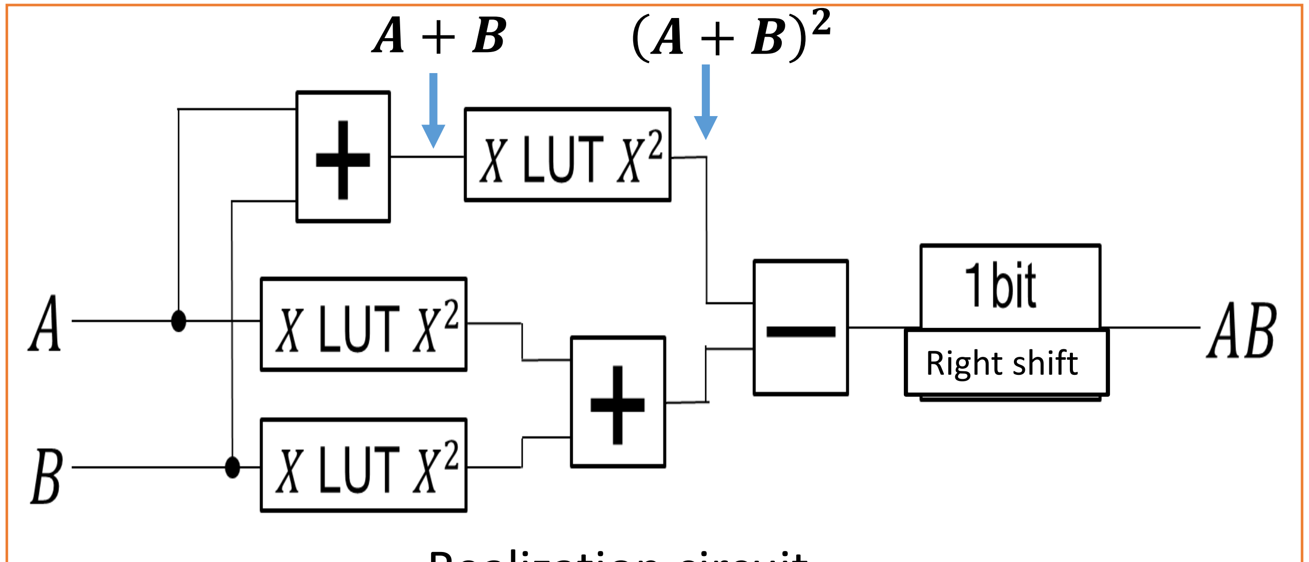
circuit size・power・computation time

OUTLINE

Investigated Multiplier Algorithm

Based on square law

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



Realization circuit

- Squaring 3 times
- Addition twice
- Subtraction once
- $\frac{1}{2}$ operation can be realized with a 1-bit left shift or just interconnection change

General Multiplier Algorithm

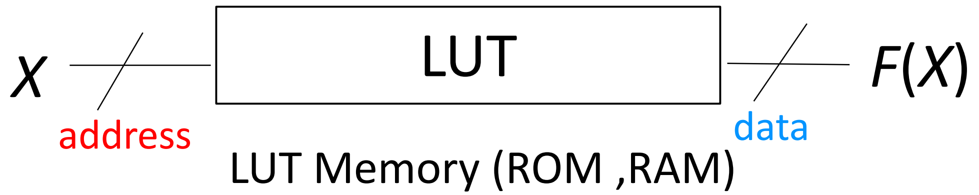
$$AB = A + A + \cdots + A$$

Multiplier A×B

Number of additions : B times

What is Look Up Table (LUT)

$A \rightarrow A^2$



Address	Memory
1	1
2	4
3	9
100	10000

No calculation

Data
1
4
9
10000

Using LUT → Memory reference processing

Efficient

Disadvantage

LUT processing ⇒handled large number of bits ⇒ Large circuit size

OUTLINE

Improvement Plan of Implementation Circuit

Come up with Divide & Conquer method

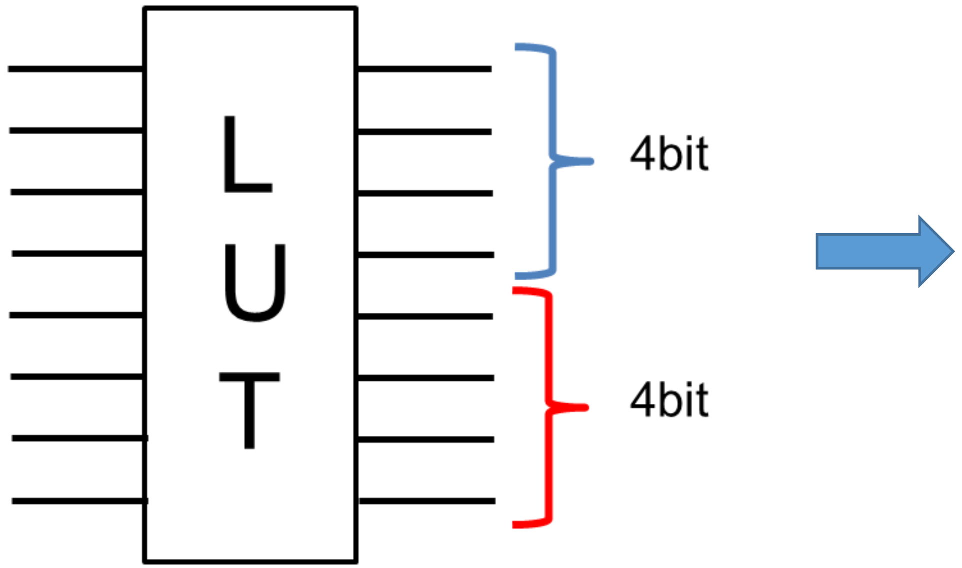
↓

$\boxed{\text{Cut LUT size}}$

$$AB = \frac{1}{2}\left[(A+B)^2 - (A^2 + B^2)\right]$$

A , *B* , *A+B* divide into upper bits and lower bits for calculation scale reduction

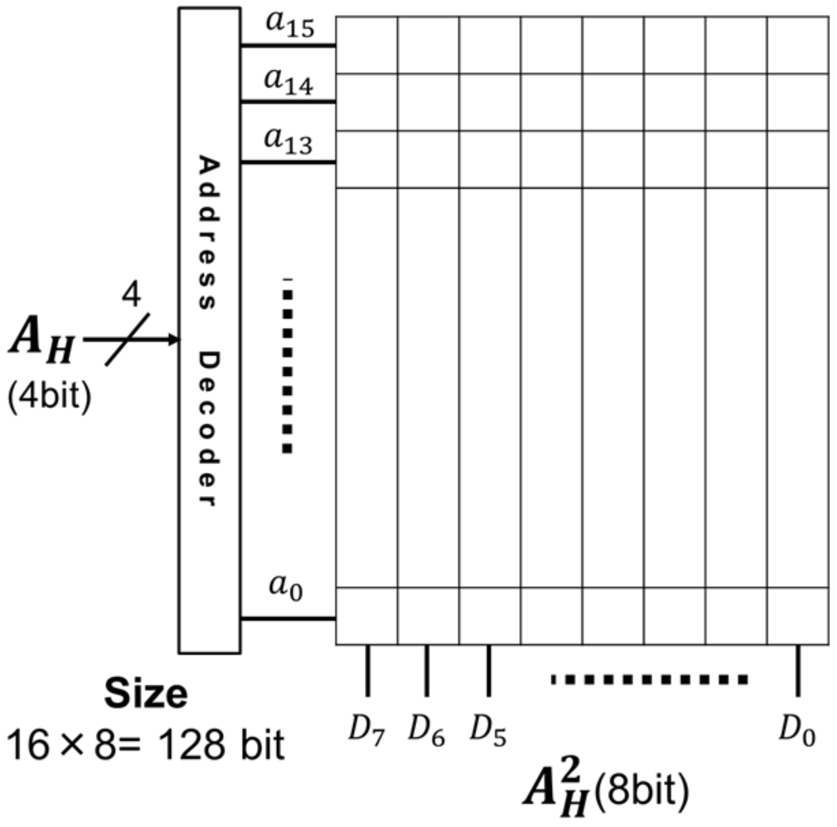
In 8 bit case : divide into
- upper 4 bits
- lower 4 bit

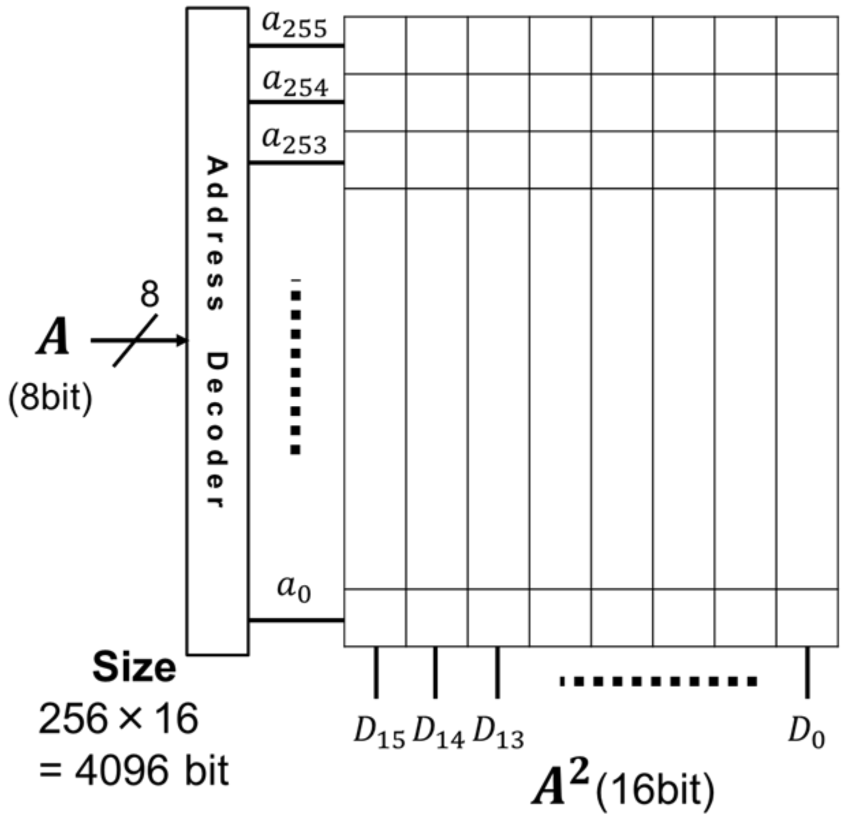


LUT size becoming smaller

Number of Bits Handled by LUT

Reduce required number of bits using Divide & Conquer





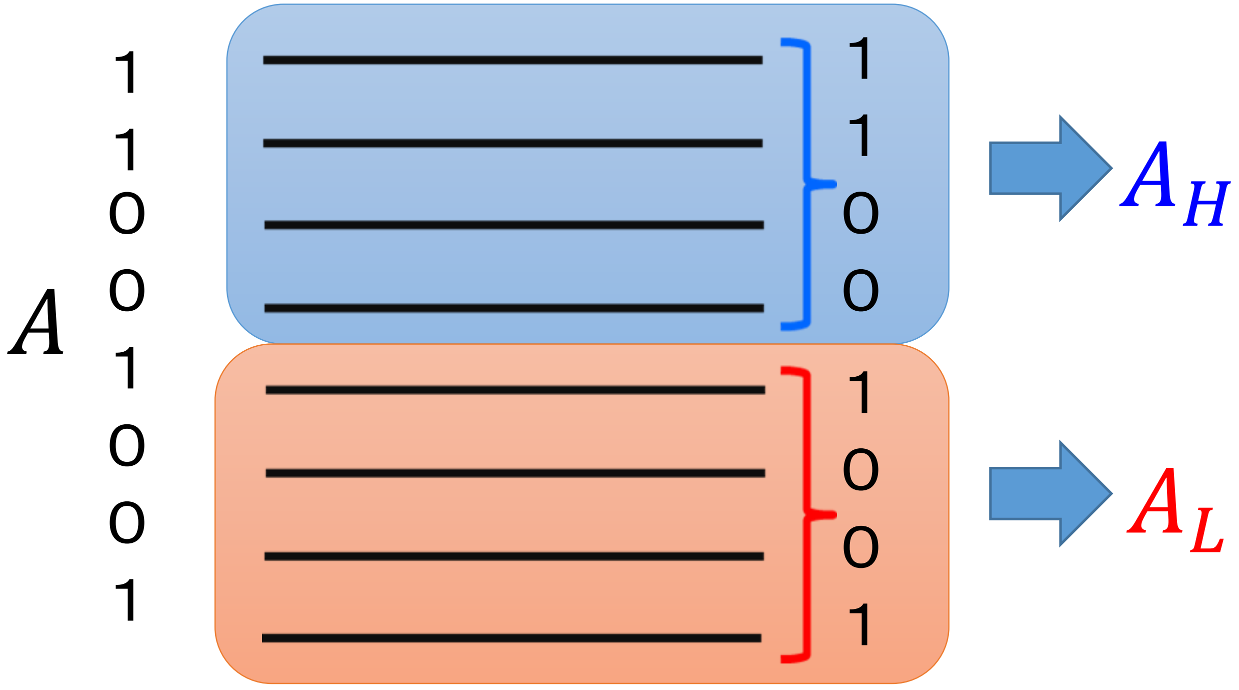
The number of input bits is reduced by half ➡ LUT size will be significantly reduced

Divide & Conquer Method Analysis

In 8 bit case（$A = 11001001 : 201_{10}$）

8bit x 8bit divide 4bit [A_H], [A_L]

Calculated by each [A_H], [A_L]



$$\boxed{A = 11001001}$$

Divided input, output values up and down

$$A = 11001001$$

$$A_H = 1100 : 12_{10}$$

$$A_L = 1001 : 9_{10}$$

Conquer

$$A_H^2 = 10010000 : 144_{10}$$

$$A_L^2 = 1010001 : 81_{10}$$

$$A_H A_L = 1101100 : 108_{10}$$

Divide & Conquer Method Analysis

$$A^2 = A_H^2(Nbit\ left\ shift) + A_H A_L\left((\frac{N}{2}+1)bit\ left\ shift\right) + A_L^2$$

N=8 bit situation

$$A^2 = A_H^2(8bit\ left\ shift) + A_H A_L(5bit\ left\ shift) + A_L^2$$

$A_H^2 = 10010000$

$A_H A_L = 1101100$

$A_L^2 = 1010001$

$A_H^2(8bit\ left\ shift) = 1001000000000000$

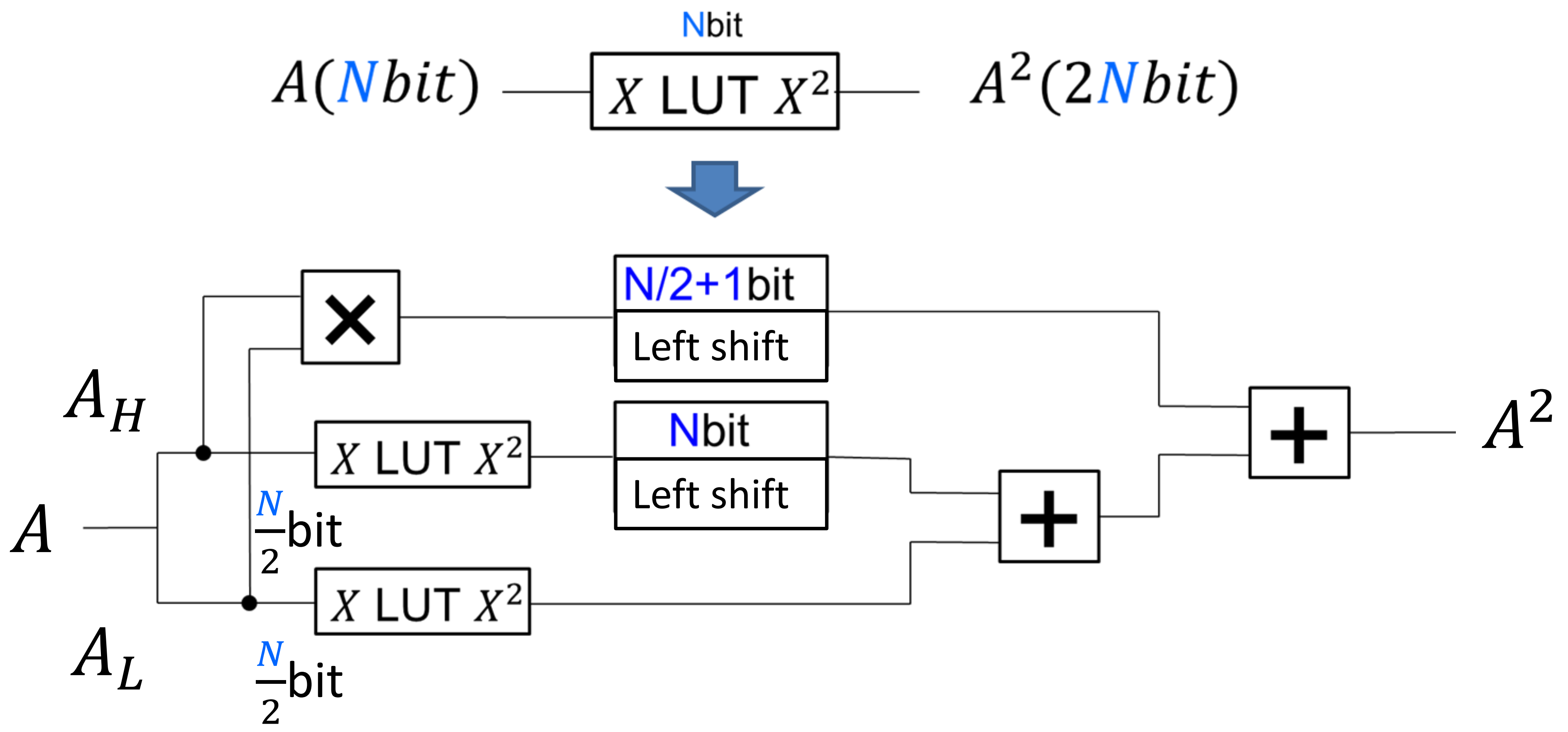
$A_H A_L(5bit\ left\ shift) = 110110000000$

$A_L^2 = 1010001$

$$A^2 = 1001110111010001 : 40401_{10}$$

$$(A^2 = 201 \times 201 = 40401)$$

Divide & Conquer Method Circuit



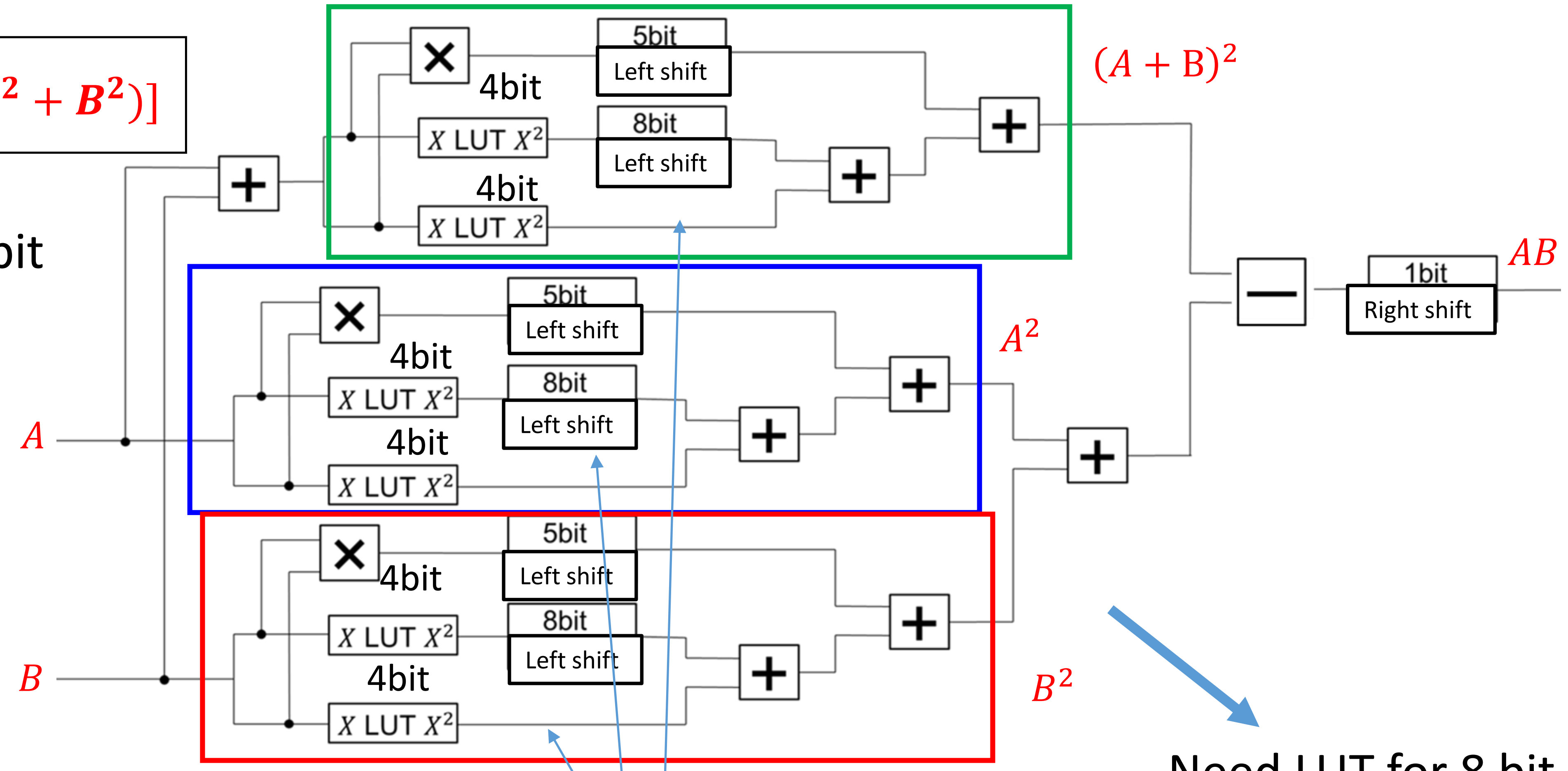
$$A^2 = A_H^2(Nbit\ left\ shift) + A_H A_L\left((\frac{N}{2}+1)bit\ left\ shift\right) + A_L^2$$

Using Divide & Conquer with X times , LUT size will decrease 2^Xtimes

Divide & Conquer Method Circuit (8 bit case)

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

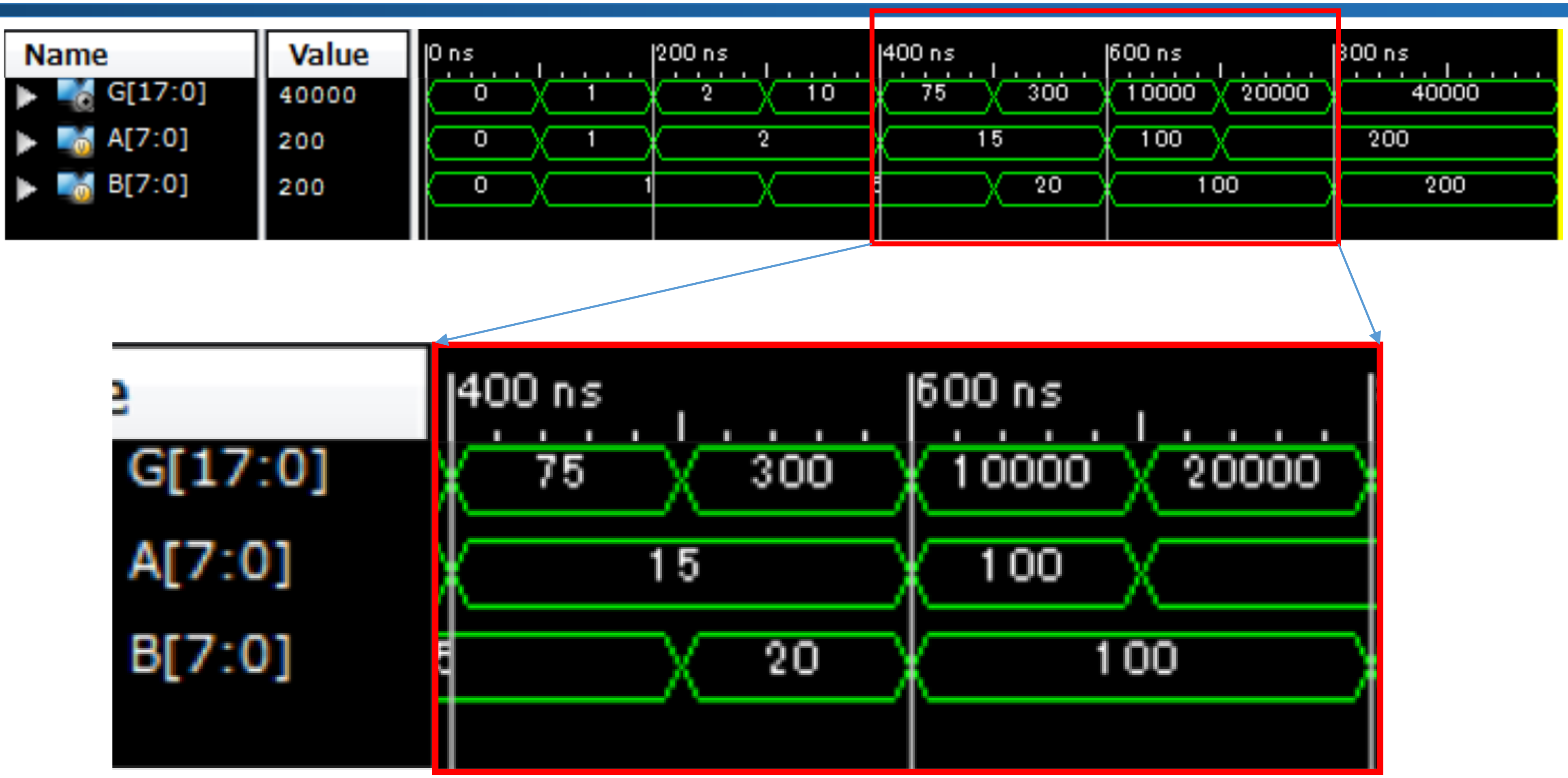
Need LUT for 16 bit



Need LUT for 8 bit

By dividing , the bit of LUT becoming smaller

RTL Simulation (8 bit × 8 bit)

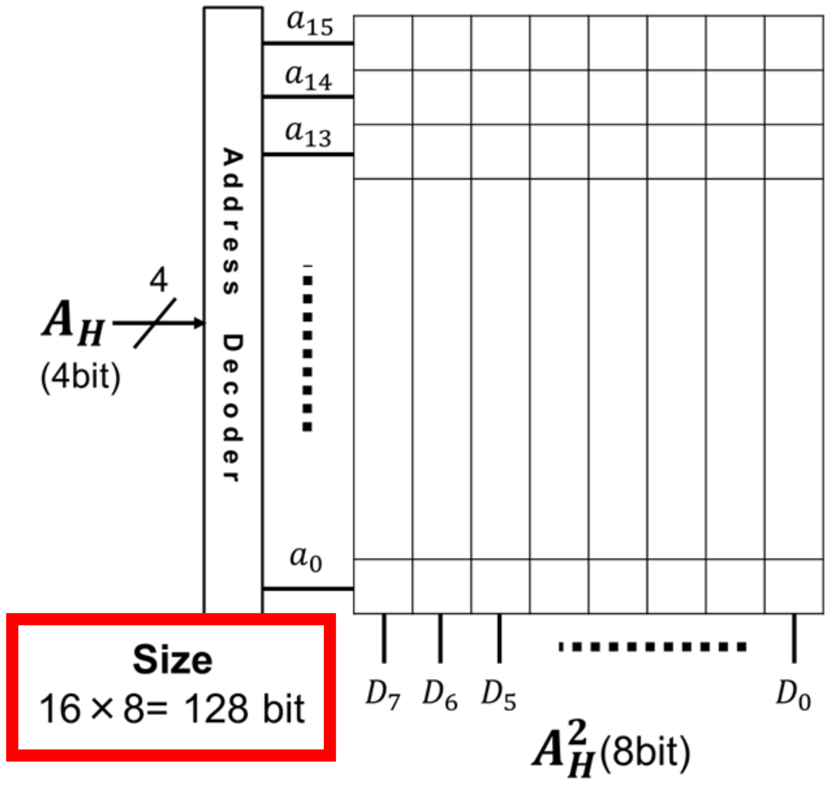


G=A × B

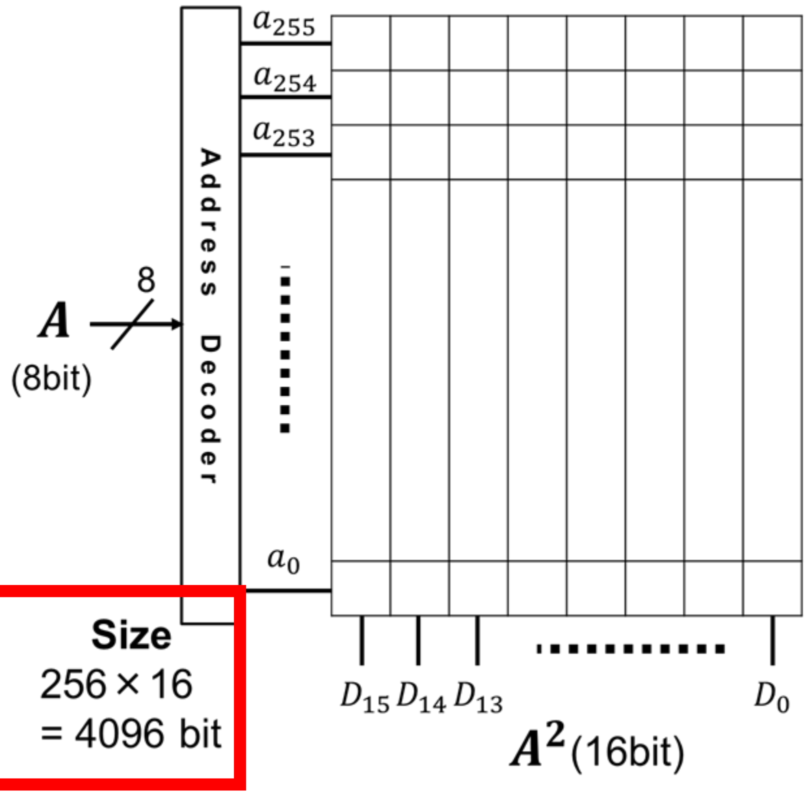
Input values A, B are changed every 100 ns and 200 ns.
A, B: input
G : output.

OUTLINE

Do NOT use LUT to Implement Square Law



Size
16 × 8= 128 bit

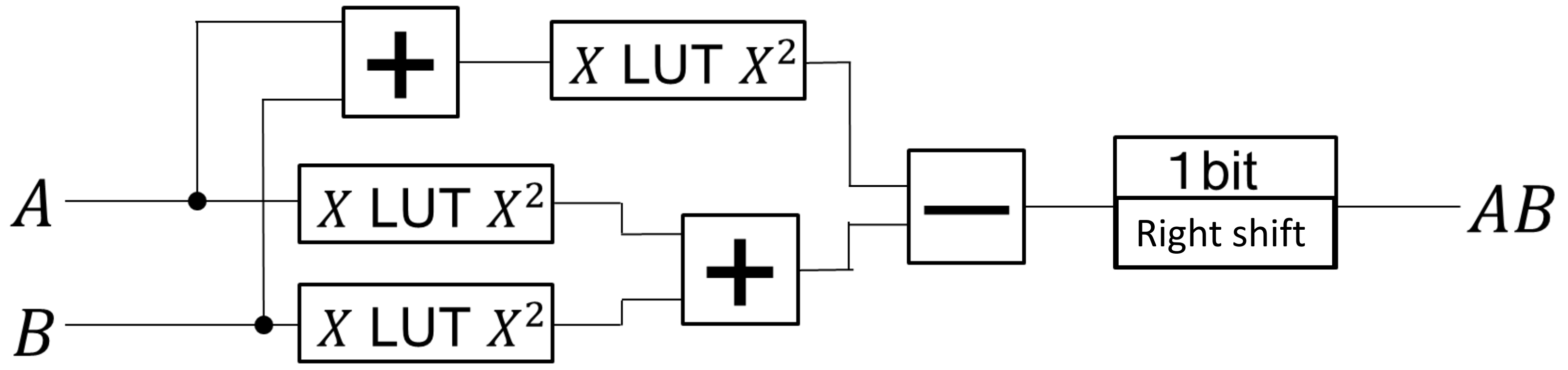


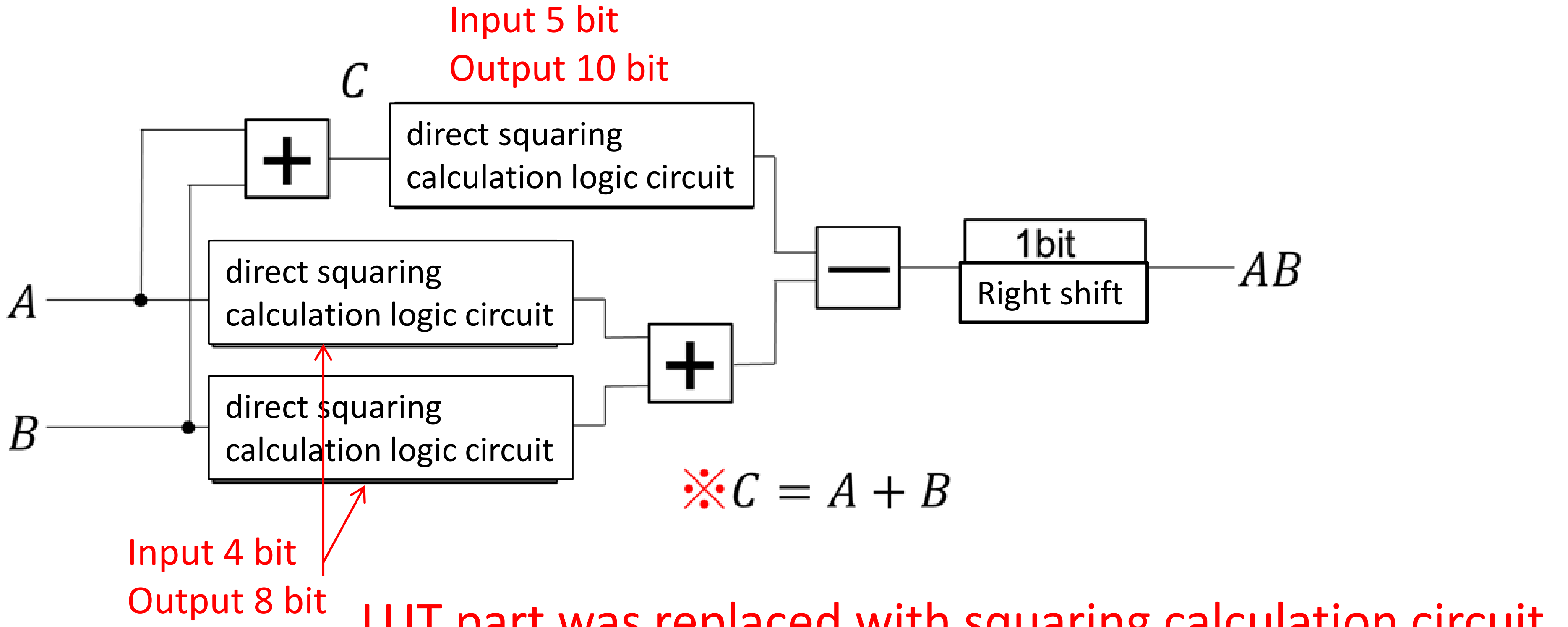
Size
256 × 16
= 4096 bit

Large size of LUT increases the overall circuit area

We have found that direct logic circuit Implementation of squaring can be simple.

Direct Squaring Calculation Logic Circuit




Input 5 bit
Output 10 bit
C
+
direct squaring calculation logic circuit
A
B
direct squaring calculation logic circuit
direct squaring calculation logic circuit
+
−
1bit
Right shift
AB
※$C = A + B$
Input 4 bit
Output 8 bit

LUT part was replaced with squaring calculation circuit.

Truth Table and Logic Expression

	Input				Output								
	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

O3 In the situation of output equal to 1 , the input are 0011 , 0101 , 1011 , 1101

$$\mathrm{O3} = \mathrm{I3I2I1I0}$$
$$\mathrm{O3} = \overline{\mathrm{I3}}\mathrm{I2}\overline{\mathrm{I1}}\mathrm{I0}$$
$$\mathrm{O3} = \mathrm{I3}\overline{\mathrm{I2}}\mathrm{I1I0}$$
$$\mathrm{O3} = \mathrm{I3I2}\overline{\mathrm{I1}}\mathrm{I0}$$

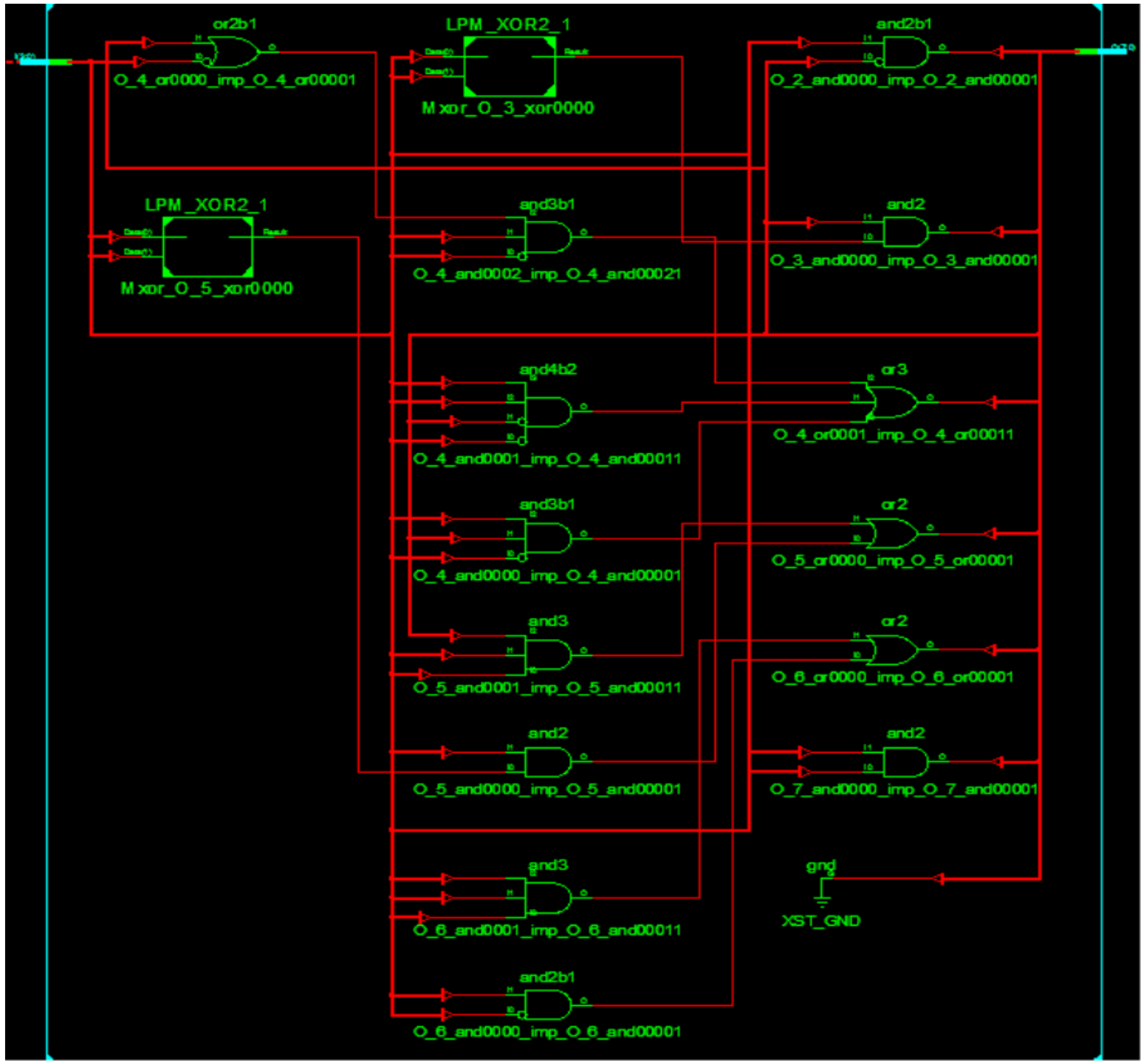
Write in a theoretical simplification

$$O3 = (I2 \oplus I1)I0$$

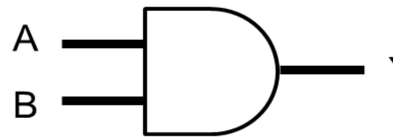
EXOR

Calculate the logic expression O0~O7

The Layout of Direct Squaring Calculation Logic Circuit

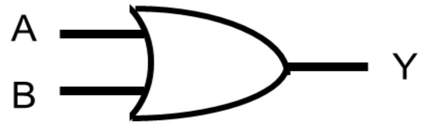

or2b1
O_4_or0000_imp_O_4_or00001
LPM_XOR2_1
Mxor_O_3_xor0000
and2b1
O_2_and0000_imp_O_2_and00001
LPM_XOR2_1
Mxor_O_5_xor0000
and3b1
O_4_and0002_imp_O_4_and00021
and2
O_3_and0000_imp_O_3_and00001
and4b2
O_4_and0001_imp_O_4_and00011
or3
O_4_or0001_imp_O_4_or00011
and3b1
O_4_and0000_imp_O_4_and00001
or2
O_5_or0000_imp_O_5_or00001
and3
O_5_and0001_imp_O_5_and00011
or2
O_6_or0000_imp_O_6_or00001
and2
O_5_and0000_imp_O_5_and00001
and2
O_7_and0000_imp_O_7_and00001
and3
O_6_and0001_imp_O_6_and00011
gnd
XST_GND
and2b1
O_6_and0000_imp_O_6_and00001

Circuit creates individual logic expressions by the number of bits of input



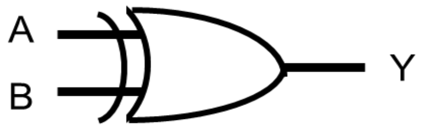
AND gate

Inout		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



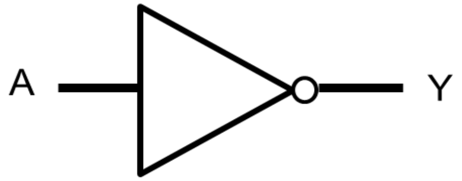
OR gate

Inout		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



EXOR gate

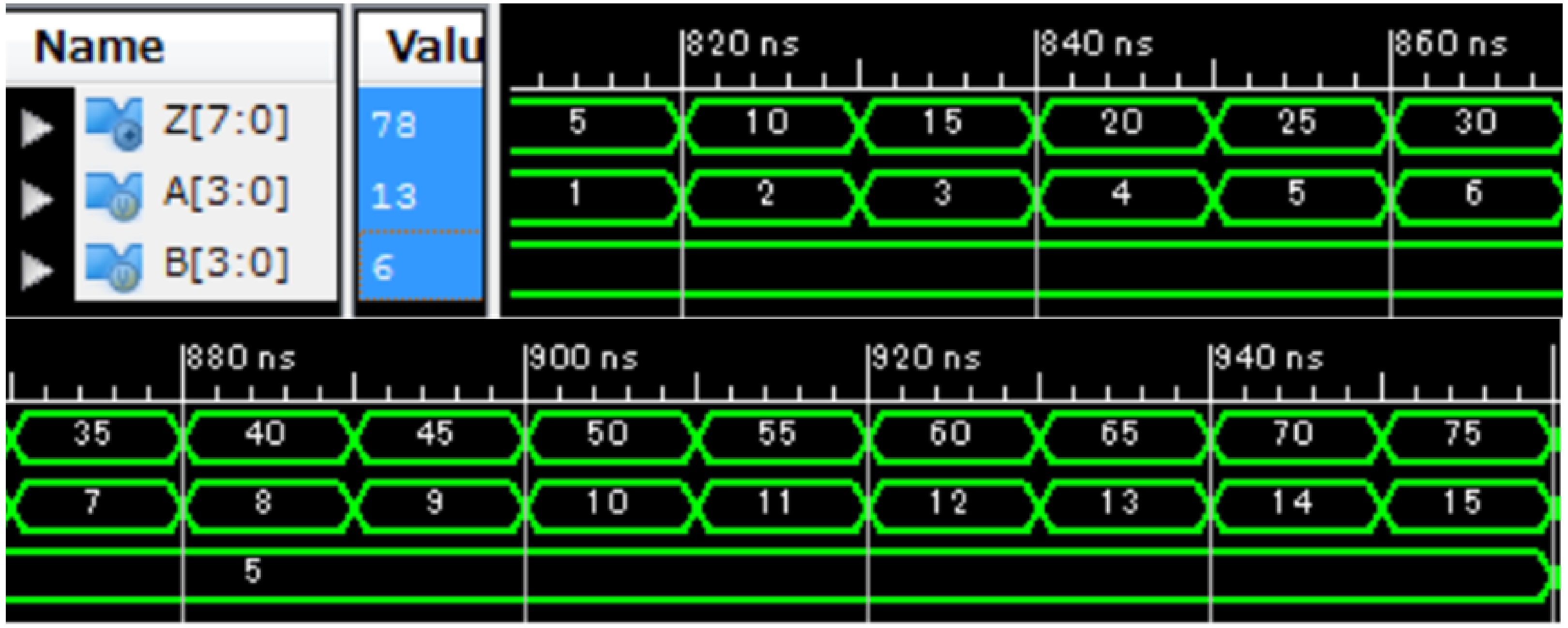
Inout		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



NOT gate

Inout	Output
A	Y
0	1
1	0

Simulation Result

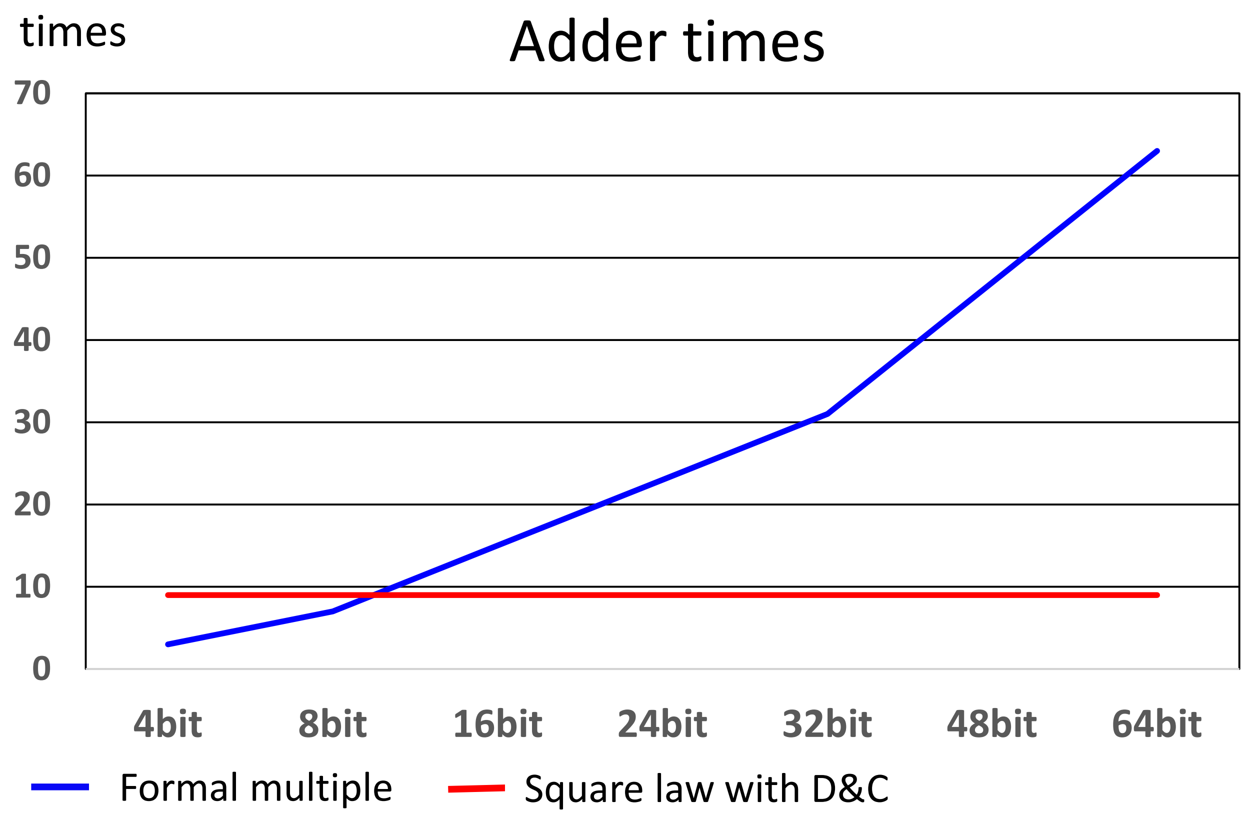


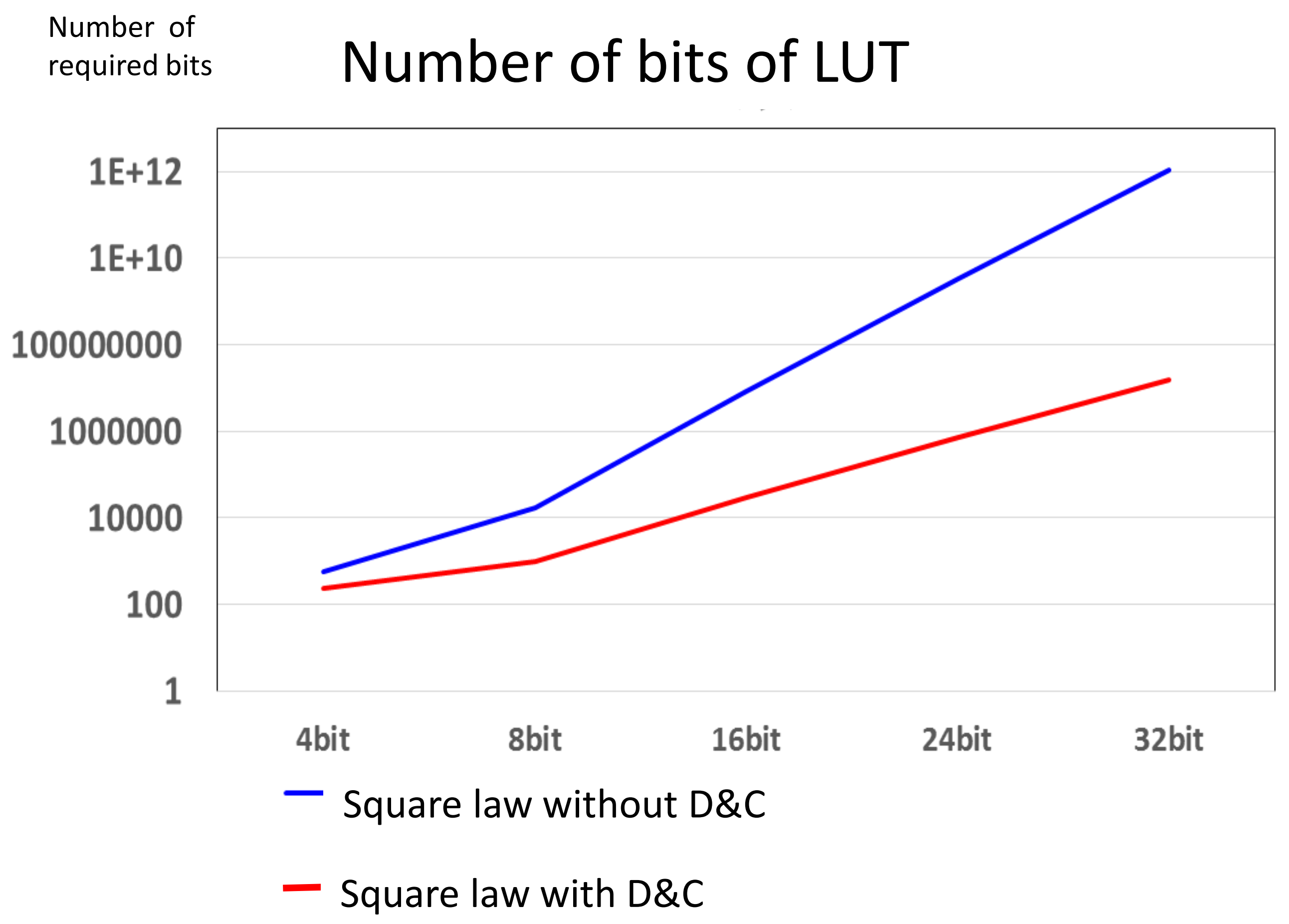
Input 4 bit × 4bit circuit

Using direct squaring calculation logic circuit was validated.

OUTLINE

Comparison of Various Algorithms





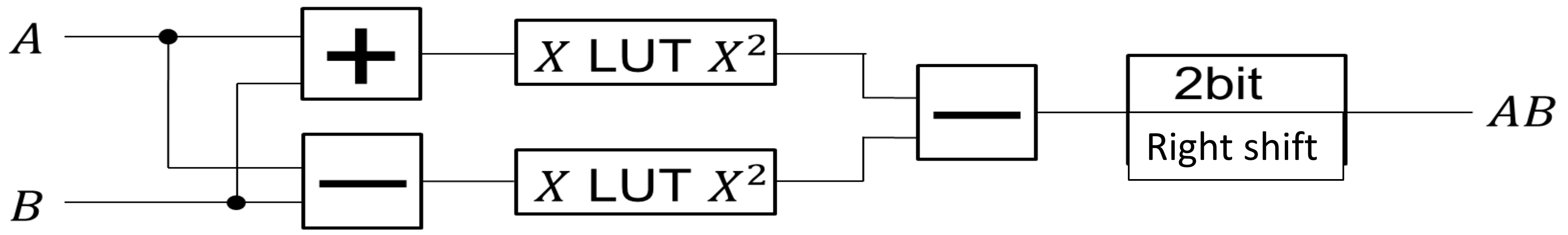
Square law method $\Rightarrow$ faster computation

Using D&C reduces LUT size

OUTLINE

Future Work

- Consideration of multipliers in other methods
  Using $AB = \frac{1}{4}\{(A+B)^2 - (A-B)^2\}$ multiplier algorithm to realize circuit design



LUT 2 times
Addition once
Subtraction twice

- Create squaring calculation logic circuit of upper bit

- Perform FPGA implementation and confirm operation

OUTLINE

- Research Background
- Digital Multiplier Algorithm
- Design of Multiply Circuit and Simulation Verification
- Circuit Design Using Squaring Calculation Logic and Simulation Verification
- Compared with Each methods
- Future Work
- Conclusion

Conclusion

- Discussion the multiplication algorithm based on square law

- Propose Divide & Conquer method to reduce the LUT size in RTL level validation by simulation

  ➡ reduce computation
  circuit area reduction

- Consider reduction of multiplication using squaring calculation logic in RTL level validation by simulation

  ➡ create dedicated circuit to calculate square simple

Thanks for your listening

Q&A

Q:This is the operation used in multiplication, have you considered how to make the other operation（such as sin、cos operation ）simple?

A:To use LUT also can realize sin and cos operation, but the capacity of LUT also large. I will consider it in the future. (After publication, I find an paper named *Application of LUT Cascades to Numerical Function Generators* can let the sin 、cos operation simple.)

Q:To explain the figure that comparison of various algorithms.

A:In left picture, the horizontal axis represent the number of bits that need to be multiplied, the vertical axis represent the add times. With the increase in the number of bit, the square law method has less computation adder times.

In right picture, the horizontal axis represent the number of bits that need to be multiplied, the vertical axis represent the number of required bit in LUT. With the increase in the number of bit, using Divide&Conquer method can reduce the size of LUT.