Kobayashi Laboratory

# 2乗則を用いたデジタル乗算器アルゴリズム

群馬大学大学院　理工学府 電子情報部門
小林春夫
koba@gunma-u.ac.jp
http://www.el.gunma-u.ac.jp/~kobaweb/

群馬大学 GUNMA UNIVERSITY

# 10進数と2進数

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

例　2進数　1011　を
10進数に変換
1x2x2x2+0x2x2+1x2+1
=11

# 2進数 （2のべき乗）

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

$2^9 = 512$

$2^{10} = 1{,}024 = 1K$

(参考　1,000=1k)

# デジタル回路と2進数

- 人間はなぜ10進数を使うか？

  → 手の指が10本あるから。

- デジタルではなぜ2進数を使うか？

  → 2つの状態は技術的に容易かつ安定して実現可能。

例：　電圧の高いと低い

　　電流の流れる状態と流れない状態

　　パルスのあるとなし。

# 今から330年前、1692年のパリ

哲学者、数学者、科学者　ライプニッツ
（**Gottfried Wilhelm Leibniz**）
「全ての数を1と0によって表す驚くべき表記法」
を提案。
王立科学アカデミーに理解されず
学会誌にも掲載されなかった。

「誰も予想しなかった卓越した用途がありはずだ」
と語る。（慶応義塾大学　青山先生資料）

# ゴットフリート・ヴィルヘルム・ライプニッツ

（Gottfried Wilhelm Leibniz,
1646年 - 1716年）

ライプニッツは哲学者、数学者、科学者など幅広い分野で活躍した学者・思想家として知られているが、また政治家であり、外交官でもあった。17世紀の様々な学問（法学、政治学、歴史学、神学、哲学、数学、経済学、自然哲学（物理学）、論理学等）を統一し、体系化しようとした。その業績は法典改革、モナド論、微積分法、微積分記号の考案、論理計算の創始、ベルリン科学アカデミーの創設等、多岐にわたる。ライプニッツは稀代の知的巨人といえる。

# 微分　積分は逆演算

微分積分学の創始者：　ニュートン、ライプニッツ

微分積分学の基本定理

「微分と積分が互いに逆の操作・演算である」

関数を積分し微分するともとの関数になる

この定理が発見されるまでは、微分法と積分法は
なんの関連性も無い全く別の計算だと考えられていた。

微分積分学：　曲線の下の面積を求める問題（積分）と
動きを瞬間的に捉えるという問題（微分）を
考えてきた先人の成果の上に成立。

# デジタル加算

## 2進数の加算

```
  0011   (3)
+) 1011  (11)
-------------
  1110   (14)
```

## 10進数の加算

```
   437
+) 258
-------
   695
```

## 2入力2進加算

0+0=00

0+1=01

1+0=01

1+1=10

## 3入力2進加算

0+0+0=00

0+0+1=01

0+1+1=10

1+1+1=11

# デジタル加算器の実現(2)

## （全加算器； Full Adder）

### 3入力2進加算

A　入力1

B　入力2

+) Cin　下からの繰り上がり

Co S

$S = A \oplus B \oplus Cin$

$Co = B \cdot Cin + A \cdot Cin + A \cdot B$

(Co は A, B, Cin の多数決)

真理値表

| A | B | Cin | Co | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

# 全加算器 (Full Adder)の補足説明

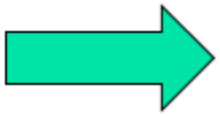
Cin: Carry in (下位の桁からの繰り上げ)

S: Sum（加算結果）

Cout: Carry out （上位の桁への繰り上げ）

$$\begin{array}{rcc} & 0 & 1 \\ +) & 1 & 1 \\ \hline 1 & 0 & 0 \end{array}$$

を考える。



全加算器の演算

$$\begin{array}{rcc} & 1 & \\ & 0 & 1 \\ +) & 1 & 1 \\ \hline 1 & 0 & 0 \end{array}$$

# 全加算器の補足説明(続き)

```
   0  1  0  1  (5)
+) 0  1  1  1  (7)
-------------------
 0 1  1  0  0  (12)
```

を考える。

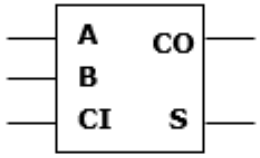
```
     1  1  1
   0  1  0  1
+) 0  1  1  1
-------------
 0 1  1  0  0
```

Carry（桁上げ）

Sum（加算結果）

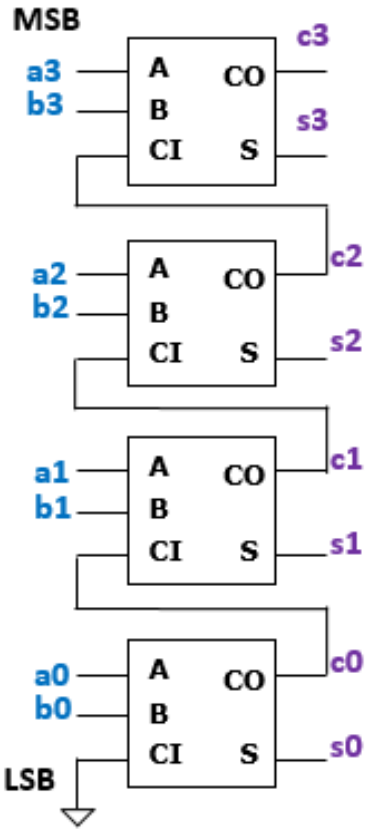
# 全加算器と桁上げ伝播

**Full Adder**




| A | B | CI | S | CO |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Adder & Carry Propagation**




Example :

$$\begin{array}{r} 0\,1\,1\,1 \\ +\ 0\,1\,0\,1 \\ \hline 1\,1\,0\,0 \end{array}$$

(a3 a2 a1 a0) =(0 1 1 1)
(b3 b2 b1 b0)=(0 1 0 1)

(s3 s2 s1 s0)=(1 1 0 0)
(c3 c2 c1 c0)=(0 1 1 1)

# ビットシフトと乗算

10進数で　x10, x100, ÷1000 等は簡単
同様に2進数で x2, x4、÷8 等は簡単

● 1 bit left shift ⟷ ×2

| b4 | b3 | b2 | b1 | b0 | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 1 | 1 | 0 | 3 ×2=6 |
| 0 | 1 | 1 | 0 | 0 | 6 ×2=12 |
| 1 | 1 | 1 | 0 | 1 | -3 |
| 1 | 1 | 0 | 1 | 0 | -3 ×2=-6 |
| 1 | 0 | 1 | 0 | 0 | -6 ×2=-12 |

●1 bit right shift ⟷ ÷ 2

| b4 | b3 | b2 | b1 | b0 | |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 8 |
| 0 | 0 | 1 | 0 | 0 | 8 / 2=4 |
| 0 | 0 | 0 | 1 | 0 | 4 / 2=2 |
| 1 | 1 | 0 | 0 | 0 | -8 |
| 1 | 1 | 1 | 0 | 0 | -8 / 2=-4 |
| 1 | 1 | 1 | 1 | 0 | -4 / 2=-2 |

| b4 | b3 | b2 | b1 | b0 | 十進数 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 0 | 1 | 0 | 1 | 5 |
| 0 | 0 | 1 | 1 | 0 | 6 |
| 0 | 0 | 1 | 1 | 1 | 7 |
| 0 | 1 | 0 | 0 | 0 | 8 |
| 0 | 1 | 0 | 0 | 1 | 9 |
| 0 | 1 | 0 | 1 | 0 | 10 |
| 0 | 1 | 0 | 1 | 1 | 11 |
| 0 | 1 | 1 | 0 | 0 | 12 |
| 0 | 1 | 1 | 0 | 1 | 13 |
| 0 | 1 | 1 | 1 | 0 | 14 |
| 0 | 1 | 1 | 1 | 1 | 15 |
| 1 | 0 | 0 | 0 | 0 | -16 |
| 1 | 0 | 0 | 0 | 1 | -15 |
| 1 | 0 | 0 | 1 | 0 | -14 |
| 1 | 0 | 0 | 1 | 1 | -13 |
| 1 | 0 | 1 | 0 | 0 | -12 |
| 1 | 0 | 1 | 0 | 1 | -11 |
| 1 | 0 | 1 | 1 | 0 | -10 |
| 1 | 0 | 1 | 1 | 1 | -9 |
| 1 | 1 | 0 | 0 | 0 | -8 |
| 1 | 1 | 0 | 0 | 1 | -7 |
| 1 | 1 | 0 | 1 | 0 | -6 |
| 1 | 1 | 0 | 1 | 1 | -5 |
| 1 | 1 | 1 | 0 | 0 | -4 |
| 1 | 1 | 1 | 0 | 1 | -3 |
| 1 | 1 | 1 | 1 | 0 | -2 |
| 1 | 1 | 1 | 1 | 1 | -1 |

# デジタル乗算

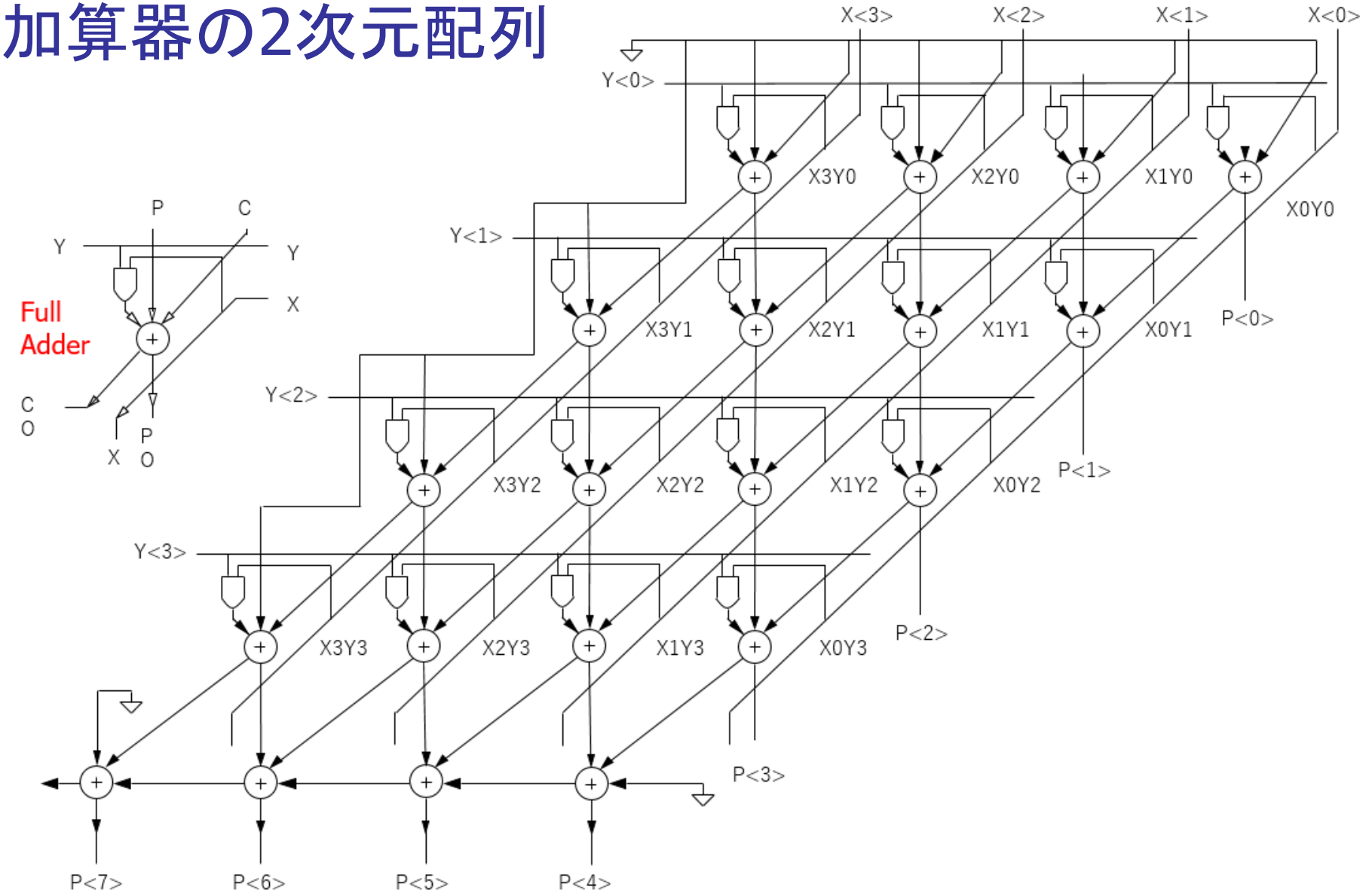
## 2進数の乗算

```
   0101   (5)
X) 1011   (11)
-------------
   0101
  0101
 0000
0101
-------------
0110111   (55)
```

## 10進数の乗算

```
    437
X)  258
-----------
   3496
  2185
   874
-----------
  112746
```

# 乗算器は
# 全加算器の2次元配列



# 研究の動機

対数　(logarithm)の発明と対数表の作成

John Napier（スコットランド 1550-1617)

安全な航海のため
天文学の計算（文字通り　天文学的数の掛け算）を
容易に行うため

log (AB) = log A + log B

# ジョン・ネイピア

*John Napier* 1550 - 1617

スコットランド

数学者、物理学者、天文学者、占星術師。

対数の発明者、対数表の作成。

かけ算を足し算に、割り算を引き算に。

天文学の膨大な計算を簡単に行える。

# 研究の動機 （続き）

が、数学の啓蒙書を読んでいて

２乗則

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

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

にNaipier が気が付いていたら　対数は必要なかった

の記述に出会う。

これに基づいて　デジタル乗算器を設計してみよう

# 研究背景

演算器
- 加算器
- 減算器
- 乗算器
- 除算器

デジタル計算機、DSPチップ、通信用SoCに広く使用

乗算器は加算器、減算器よりも頻繁に使う

複雑な計算を要するチップ内に複数乗算器が必要

乗算器は回路規模が大きいため低減が必要
小規模回路、低消費電力、高速化
のため様々な提案・実現されてきた

# デジタル加算の仕組み

小学校で習った方式（筆算）を2進数に変換する

10進数での演算

| | |
|---|---|
| 25 | 被加数 |
| ＋39 | 加数 |
| 64 | 結果 |

2進数での演算

| | |
|---|---|
| $011001 : 25_{10}$ | 被加数 |
| $＋100111 : 39_{10}$ | 加数 |
| $1000000 : 64_{10}$ | 結果 |

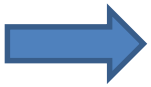
# デジタル乗算の仕組み

小学校で習った方式

乗数に応じた部分積を求め、
それらを足し合わせて積を得る

10進数での演算

| | |
|---|---|
| 25 | 被乗数 |
| ×39 | 乗数 |
| 45 | 部分積 |
| 18 | |
| 15 | |
| ＋ 6 | |
| 975 | 積 |



2進数での演算をベース

| | |
|---|---|
| $011001 : 25_{10}$ | 被乗数 |
| $\times 100111 : 39_{10}$ | 乗数 |
| 011001 | 部分積 |
| 011001 | |
| 011001 | |
| 000000 | |
| 000000 | |
| ＋011001 | |
| $001111001111 : 975_{10}$ | 積 |

部分積の和の
計算回数が多くなる

# 全加算器の二次元配列を用いた乗算器

$011001 : 25_{10}$ 被乗数

$\times 100111 : 39_{10}$ 乗数

011001

011001

011001

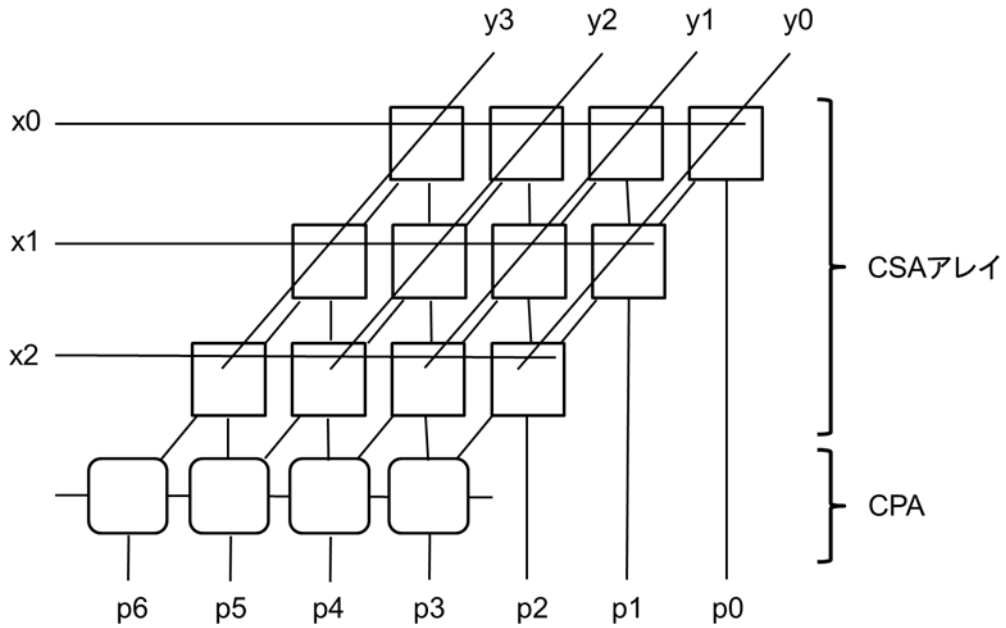
000000

000000

＋011001

部分積

$001111001111 : 975_{10}$ 積




部分積＆加算のため全加算器の２次元配列

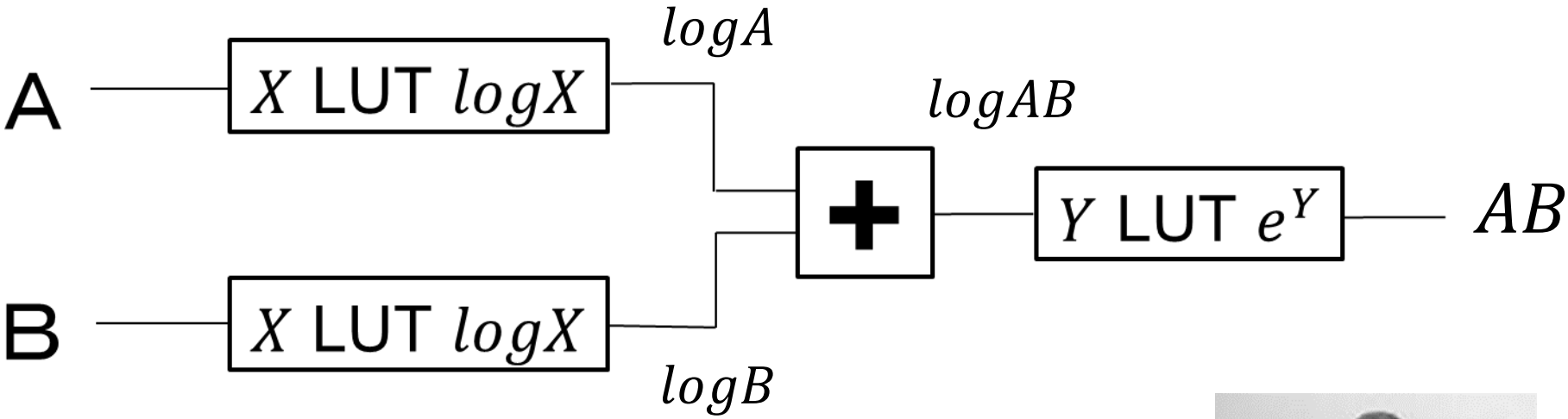
例：8bit×8bit の場合

直接的構成では8×8=64個の全加算器が必要

回路規模・消費電力・演算時間が大

# 対数を用いた乗算器

乗算を、対数を用いて計算する方式




高精度で対数、指数を得るためには
ビット数
LUTサイズ
大

対数の生みの親
ジョン・ネイピア

# 研究の目的

従来方式（加算器の二次元配列）
- 回路規模
- 消費電力
- 演算時間

→ 大

従来方式（対数）
- ビット数
- LUTサイズ

大

加算器の二次元配列、対数を使わない
デジタル乗算回路の設計を行う

↓

回路規模・消費電力・演算時間の縮小

# 検討する乗算アルゴリズム

## 2乗則に基づく乗算アルゴリズム

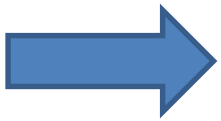
$$AB = \frac{1}{2}\left[(A+B)^2 - A^2 - B^2\right]$$

乗算を加算、減算、2乗、ビットシフトで実現

- 加算1回、減算2回
- 2乗はルックアップテーブル(LUT)で実現
- 1/2倍はシフト演算（実際は配線変更のみ）

直接的構成

乗算器１つ(24bit)
加算回数：23回



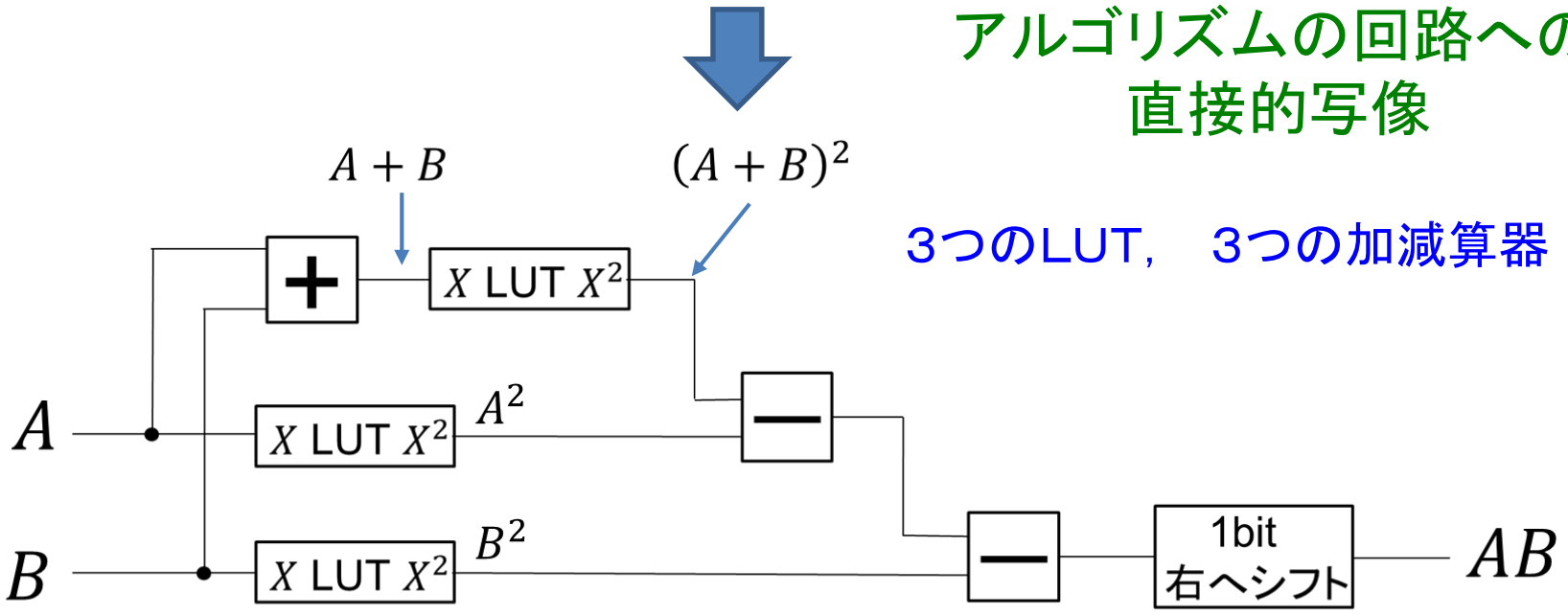
検討する構成

加算回数：1回
減算回数：2回
LUT参照：3回

# 検討アルゴリズムの実現回路

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

アルゴリズムの回路への
直接的写像




3つのLUT，　3つの加減算器

# ルックアップテーブル (LUT) とは

複雑な信号処理 → 計算時間がかかる

LUTを使うと・・・・・


計算せずに出力

メモリ (ROM, RAM)

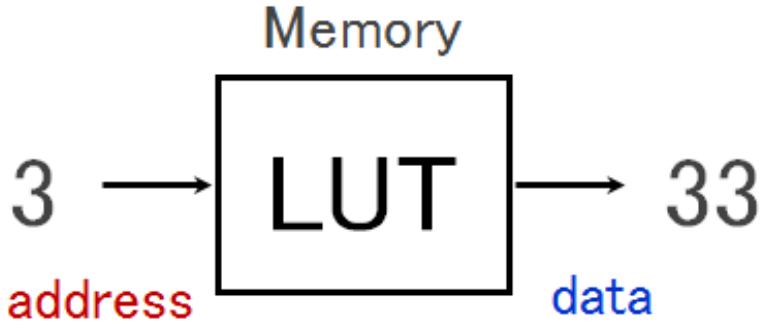
| アドレス | メモリ |
|---|---|
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| ・ | ・ |
| ・ | ・ |
| ・ | ・ |
| 100 | 10000 |

| データ |
|---|
| 1 |
| 4 |
| 9 |
| ・ |
| ・ |
| ・ |
| 10000 |

# Look Up Table (LUT)

Example 

| Cat Age | Human Age |
|---|---|
| 1 | 20 |
| 2 | 27 |
| 3 | 33 |
| 4 | 39 |
| 5 | 45 |
| 6 | 50 |
| 7 | 55 |
| 8 | 60 |



# ルックアップテーブル (LUT) とは

複雑な信号処理 → 計算時間がかかる

LUTを使うと・・・・・


$F(X)$

計算せずに出力

メモリの参照処理になり、効率化

※Sin, Cos, 2乗等任意演算がLUTで計算可能

# LUTの利点、欠点

**利点**

- 簡単に２乗演算計算
- メモリの参照処理のみ
- 回路設計が容易
- 計算速度の向上

**欠点**

- 演算ビット数が多い

↓

回路規模が大

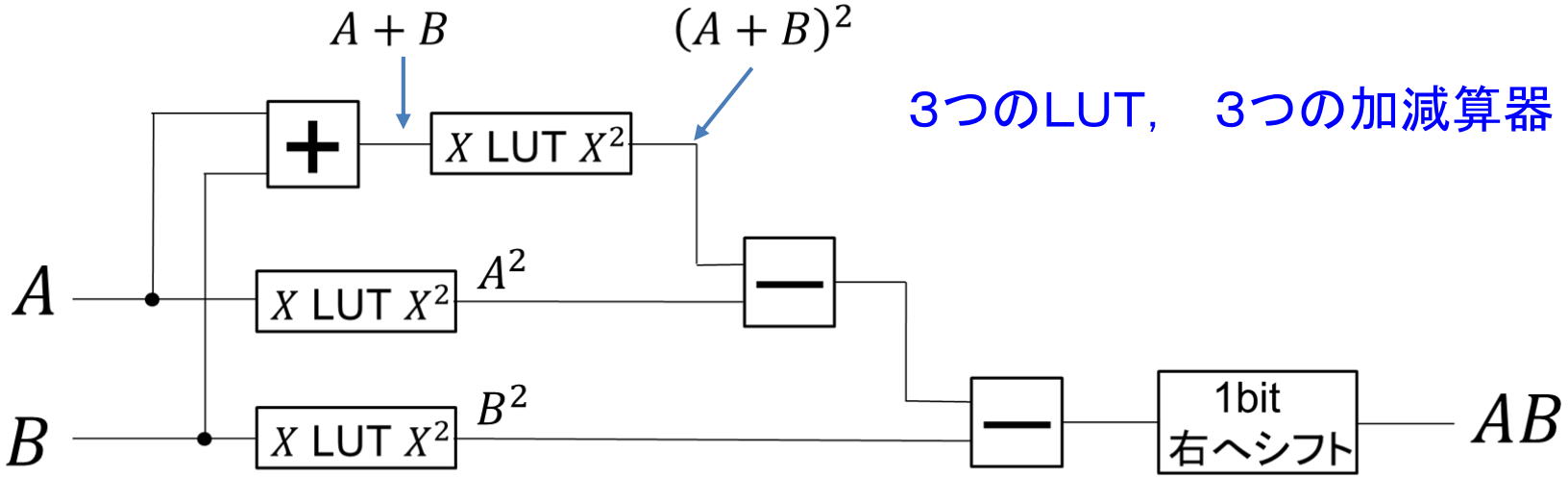
↙

**LUTを上位ビット、下位ビットに分割し**
**計算量・回路量を低減する方式**
**(Divide & Conquer)**
**を検討**

# 実現回路の改良案

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

アルゴリズムの回路への
直接的写像




3つのLUT，　3つの加減算器

※演算するbit数によってLUTのサイズが大きくなってしまう

**計算量・回路量を低減する方式(Divide & Conquer) を用いる**

# Divide & Conquer によるLUT規模削減

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

２乗演算：LUTで計算

LUTサイズ低減のため

A, B, A+B のそれぞれの上位ビット、下位ビットに分割し

計算量低減する方式

---

古代ローマ帝国：

支配下の都市相互の連帯を禁じ、

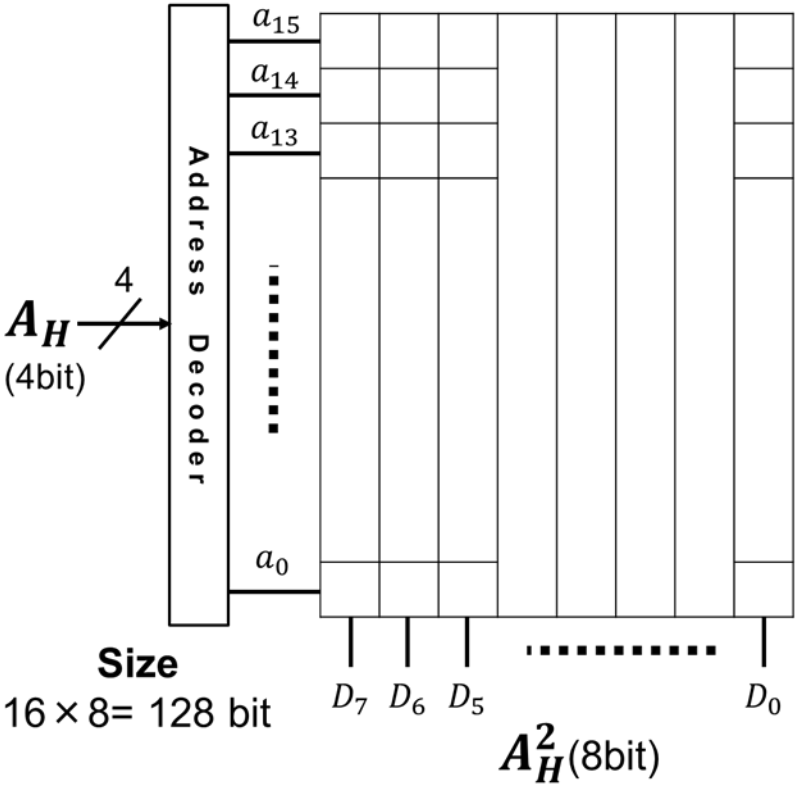
都市毎に処遇に格差をつける

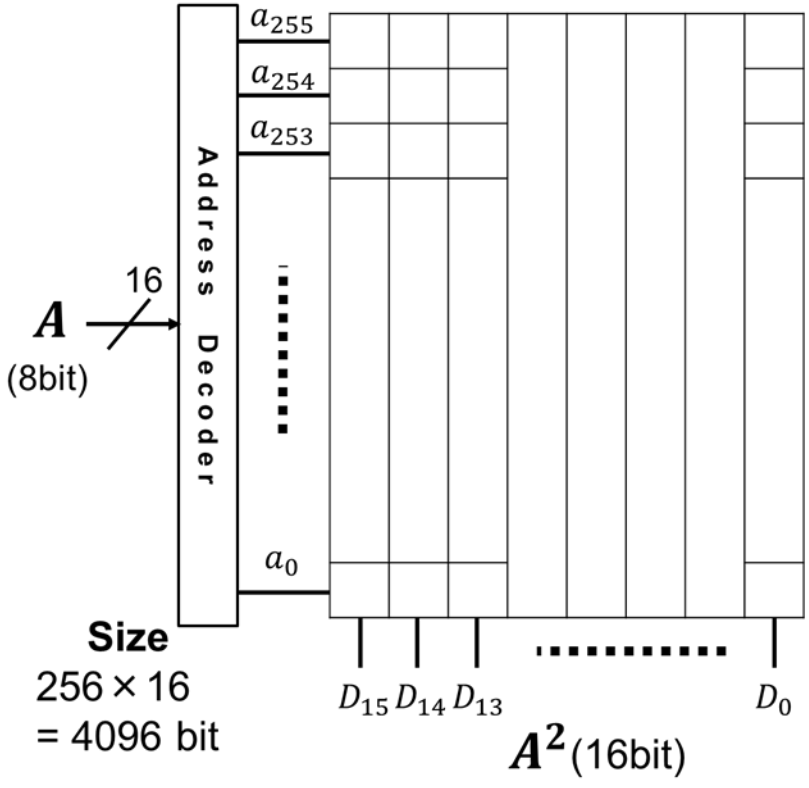
分割統治(divide & conquer)で

征服した都市の反乱を抑える。

# LUTで扱うビット数

LUTは扱うbit数が大きいほど、必要なbit数が多くなる。



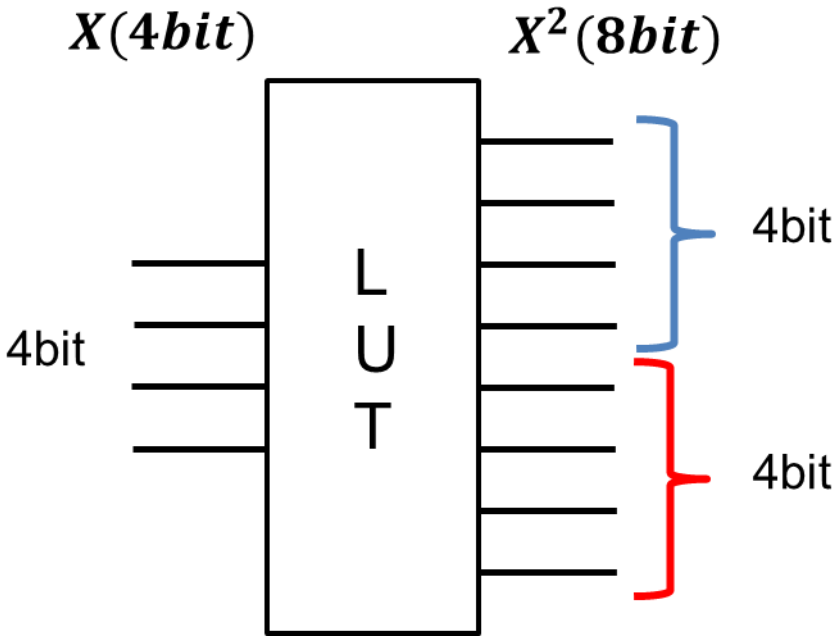

4bitでは128bit

8bitでは4096bit

Divide&Conquer法を用いることで必要なbit数を削減

# 検討するDivide & Conquer法

## 出力8bitの場合: 上下4bitずつ分割




bit数を減らして計算

シフト演算で桁を揃える

演算で用意する
メモリを小さくできる

「方法序説」
難問は、それを解くのに適切かつ必要なところまで分割せよ

ルネ・デカルト
*René Descartes* 1596 - 1650
フランス生まれの哲学者、数学者。合理主義哲学の祖

# ルネ・デカルト

## *René Descartes* 1596 - 1650

- フランス生まれの哲学者、数学者。合理主義哲学の祖 「方法序説」
- もっとも単純な要素から始め演繹していけば最も複雑なものに達しうる。
  1) 明証的に真であると認めたもの以外、決して受け入れないこと。（明証）
  2) 考える問題を出来るだけ小さい部分にわけること。（分析）
  3) 最も単純なものから始めて複雑なものに達すること。（総合）
  4) 何も見落とさなかったか、全てを見直すこと。（枚挙 / 吟味）
- 2つの実数によって平面上の点の位置を表すという方法（座標）を考案
  ➡ デカルト座標、デカルト平面
- 我思う、ゆえに我あり　I think, therefore I am.
- 難問は、それを解くのに適切かつ必要なところまで分割せよ
  Divide each difficulty into as many parts as is feasible and necessary to resolve it.

# フランシス・ベーコン

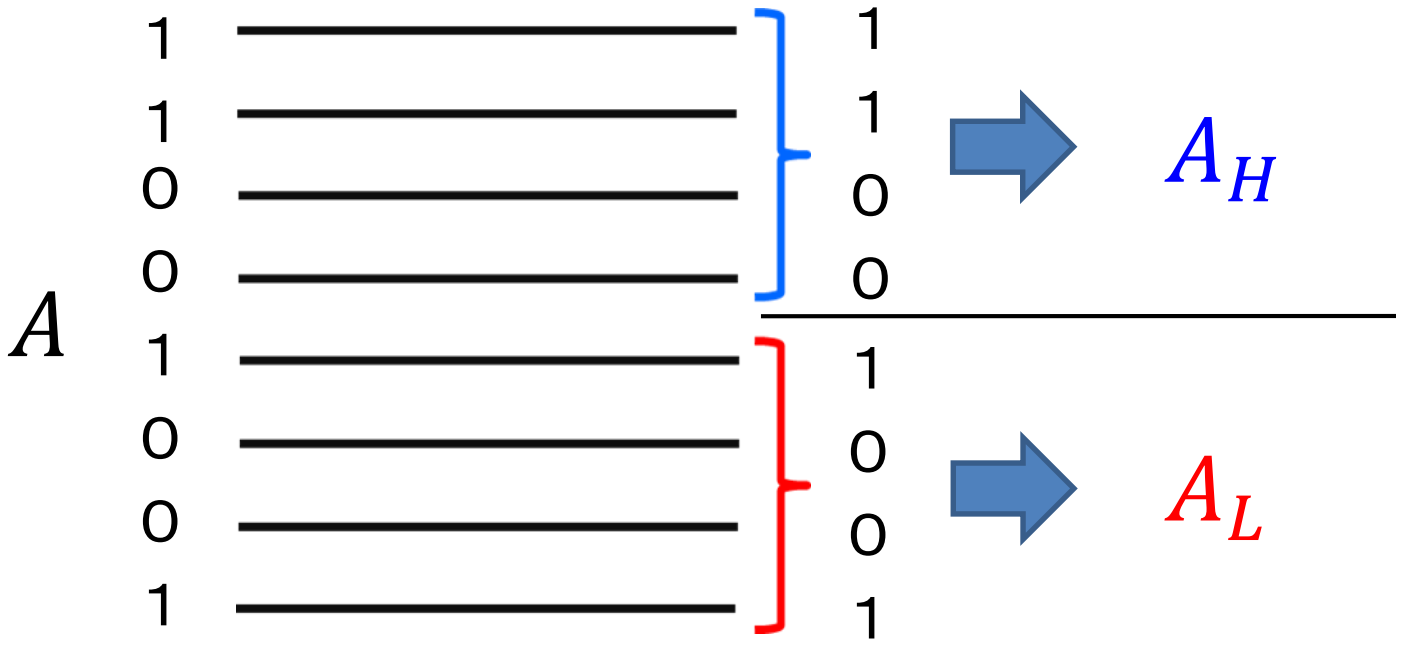
Francis Bacon　1561 - 1626

- イギリスの哲学者、神学者、法学者
- 近代的な帰納法の創始者
- 学問を確固たる実証の手続きで基礎付けようとした。
- 近代科学の精神を体現した最初の思想家
- イギリスではベーコン以後経験を重視する学問が栄える。

- 「知識は力なり」 Knowledge is power.
- 「最上の証明とは経験である」The best proof is experience.

# 提案するDivide & Conquer法

例：8bit（A=11001001）について考える。

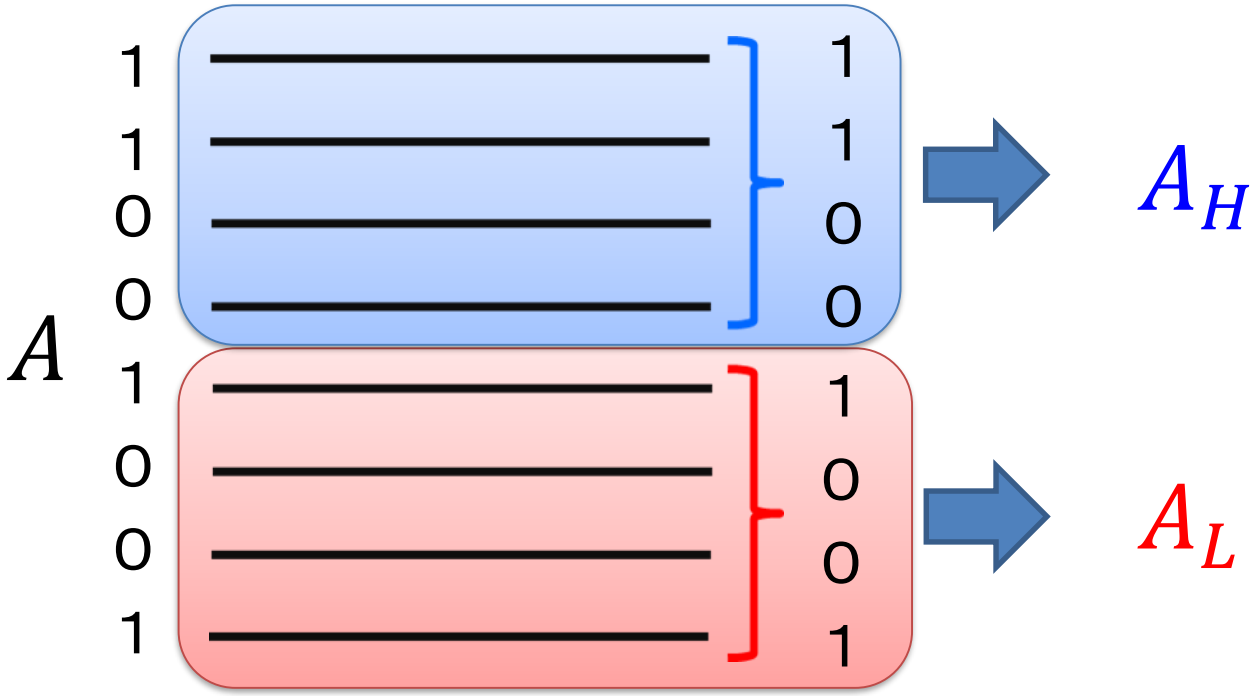



$$A^2 = A_H^2(8bit左シフト) + 2A_H A_L(4bit左シフト) + A_L^2$$

8bit x 8bit 乗算 $[A^2]$ を

4bit $[A_H]$, $[A_L]$ 毎の計算で求める。

# 提案するDivide & Conquer法




入力、出力された値を上下に分割（Divide）

# 提案するDivide & Conquer法


$A_H$

$A$


$A_L$

入力、出力された値を上下に分割（Divide）

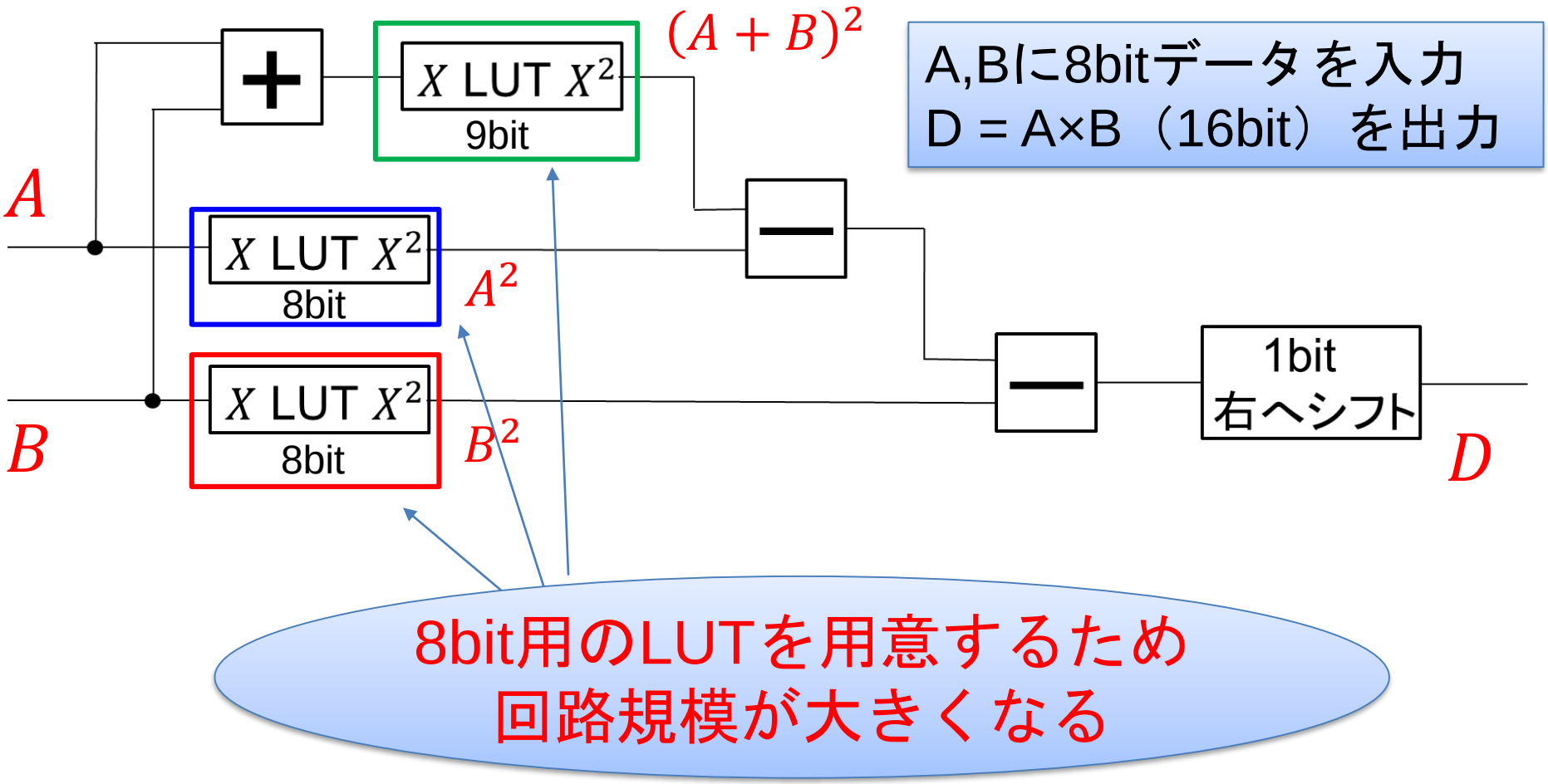
# 提案するDivide & Conquer法


$A_H$ → 演算回路で計算


$A_L$ → 演算回路で計算

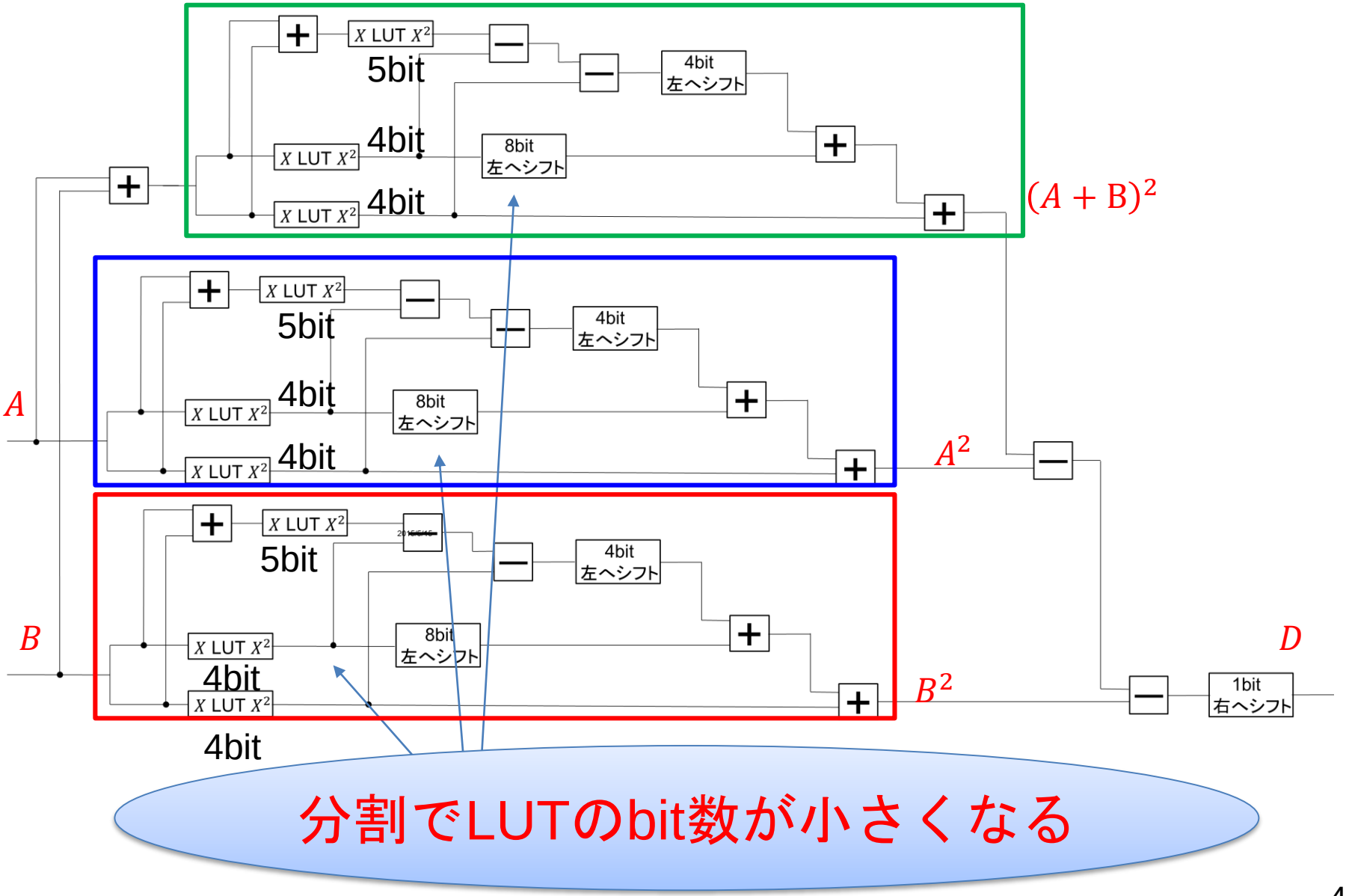
分割した値をそれぞれ計算：征服（Conquer）

# 検討乗算回路(8bit)（Divide&Conquer 無）

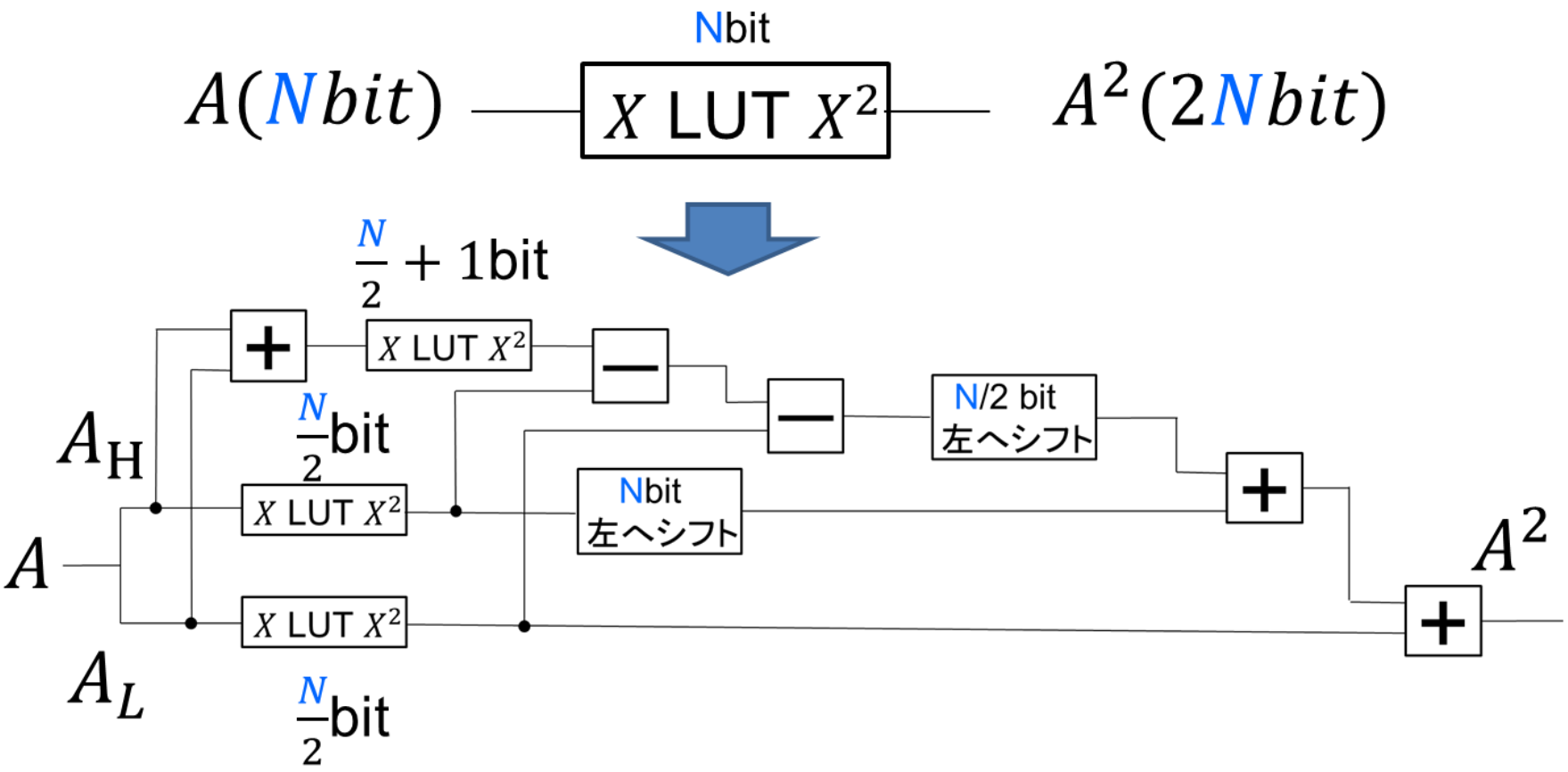



この回路をさらに変換する

# 検討乗算回路(8bit)（Divide&Conquer有）



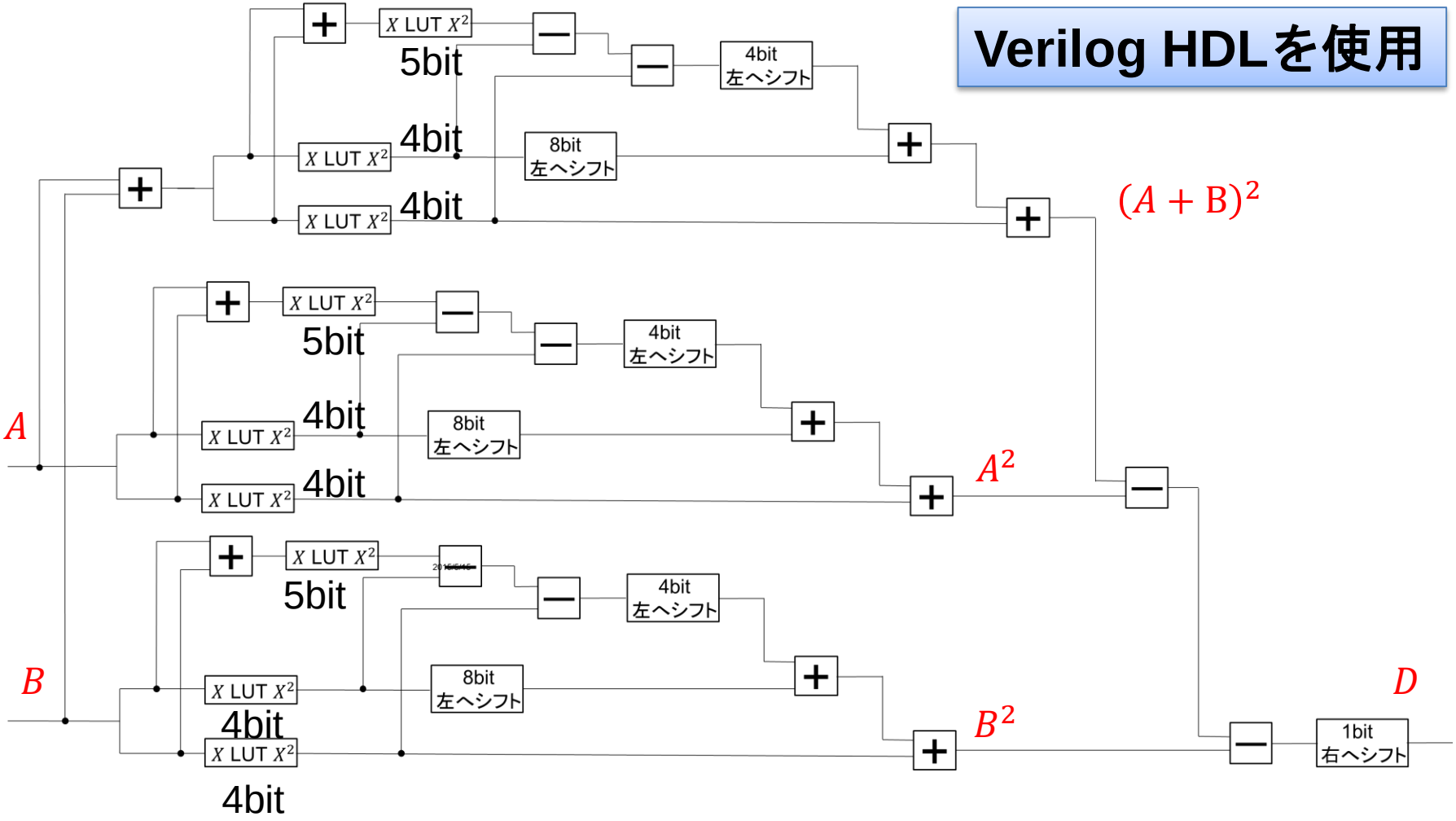
# 検討アルゴリズム乗算回路(Nbit)




Divide & Conquer 方式

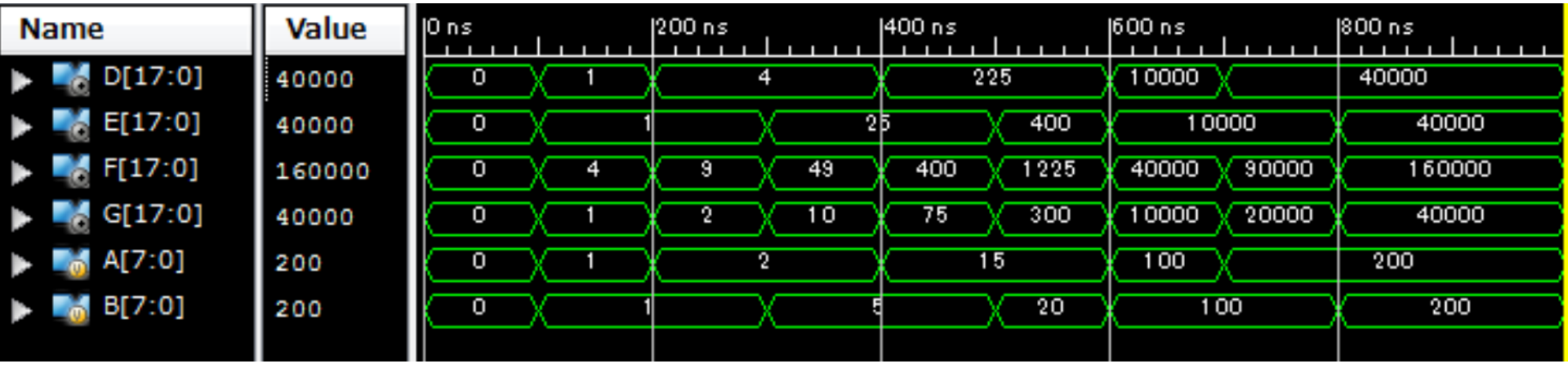
X回使用し、LUTサイズを$\left(\frac{1}{2}\right)^X$にできる

# HDLシミュレーションによる検証




8bit×8bit回路構成で正しい乗算結果が得られることを検証

# RTLシミュレーション（8bit × 8bit)




入力値A , Bの値を100ns、200ns毎に変化させ、
その区間における演算結果を波形上に表示

$$\mathrm{D(18bit)} = \mathrm{A}^2$$
$$\mathrm{E(18bit)} = \mathrm{B}^2$$
$$\mathrm{F(18bit)} = (\mathrm{A} + \mathrm{B})^2,$$
$$\mathrm{G(18bit)} = \frac{1}{2}\{(\mathrm{A}+\mathrm{B})^2 - \mathrm{A}^2 - \mathrm{B}^2\}$$

# RTLシミュレーション

- 同様に8bit×8bitのRTLシミュレーションを行い、
  256×256通りの結果を確認した

- 16bit×16bitのRTLシミュレーションを行い、
  65536×65536通りの結果を確認した

# まとめ

- ２乗則による乗算アルゴリズム・回路を検討
- Divide & Conquer 法により回路量削減を検討
- 検討アルゴリズムで乗算器をRTLレベル設計
  シミュレーションによる検証
- 従来方式と提案方式の比較をおこなった。

今後の課題

- 回路規模低減、計算スピード向上のための回路改良
- 従来法との回路量、計算スピードの比較・評価
- 他方式での乗算器検討
- Divide & Conquer方式を用いた回路との比較

# レポート課題

この講義の内容に関係したことを調べ
その内容について　A4レポート用紙2枚程度に
まとめよ。
できるだけ手書きでなくコンピュータを用いよ。

提出：　12月22日（金）の講義開始時