

2乗則を用いたデジタル乗算器 アルゴリズムとFPGA実装の検討

群馬大学大学院 理工学府

電子情報部門

佐々木秀 小林春夫



OUTLINE

- 研究背景
- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計とシミュレーション検証
- 各方式の比較
- 今後の検討回路
- まとめ

OUTLINE

● 研究背景

- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計とシミュレーション検証
- 各方式の比較
- 今後の検討回路
- まとめ

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
<hr/>	
64	結果



2進数での演算

011001	: 25 ₁₀	被加数
+100111	: 39 ₁₀	加数
<hr/>		
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

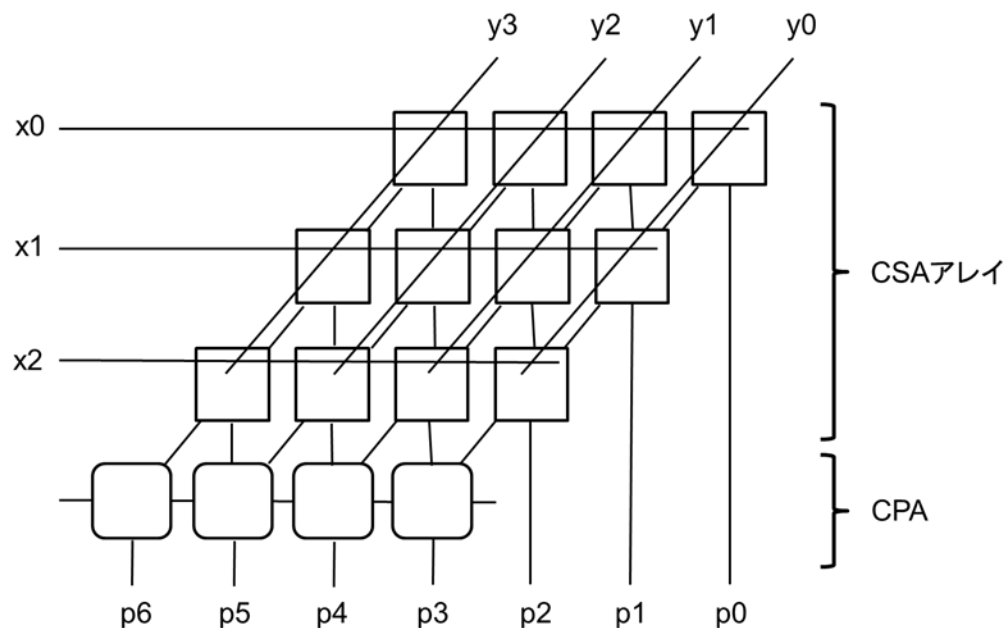
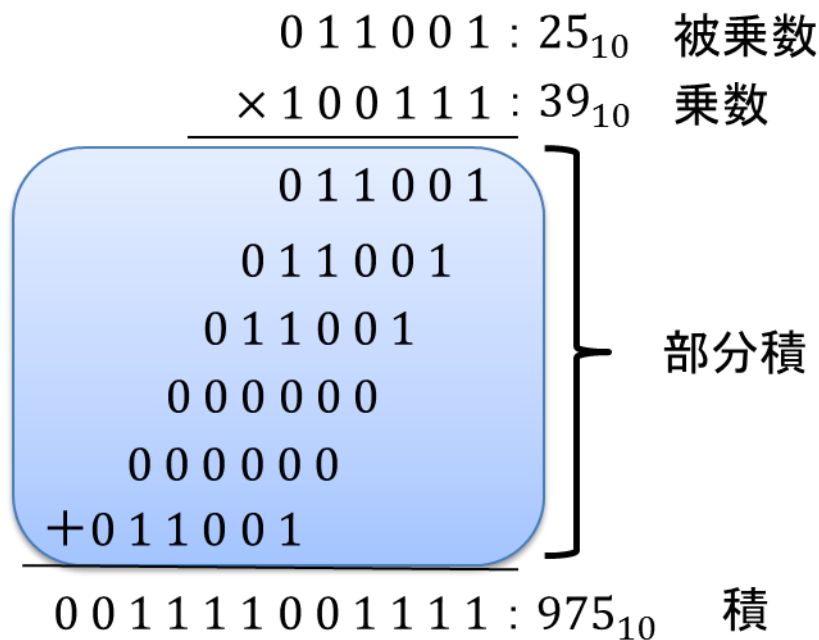
25	被乗数
×39	乗数
—	
45	} 部分積
18	
15	
+ 6	
—	
975	積

2進数での演算をベース

011001	: 25 ₁₀	被乗数
×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



部分積 & 加算のため全加算器の2次元配列

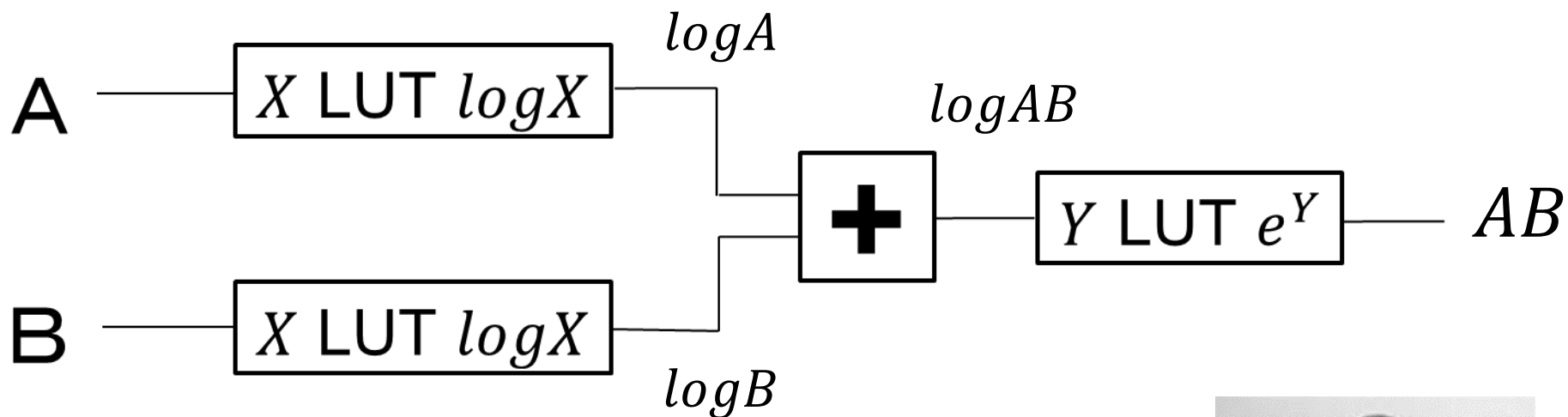
例: 8bit × 8bit の場合

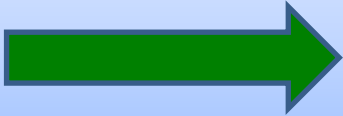
直接的構成では8 × 8 = 64個の全加算器が必要

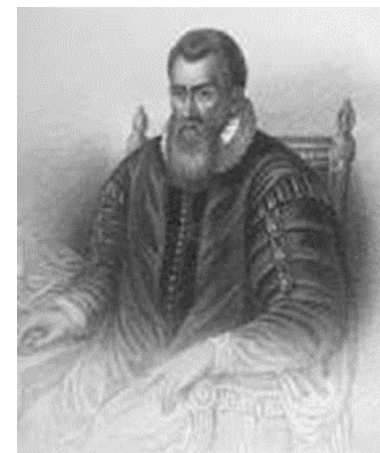
回路規模・消費電力・演算時間が大

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

OUTLINE

- 研究背景
- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計とシミュレーション検証
- 各方式の比較
- 今後の検討回路
- まとめ

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

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

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

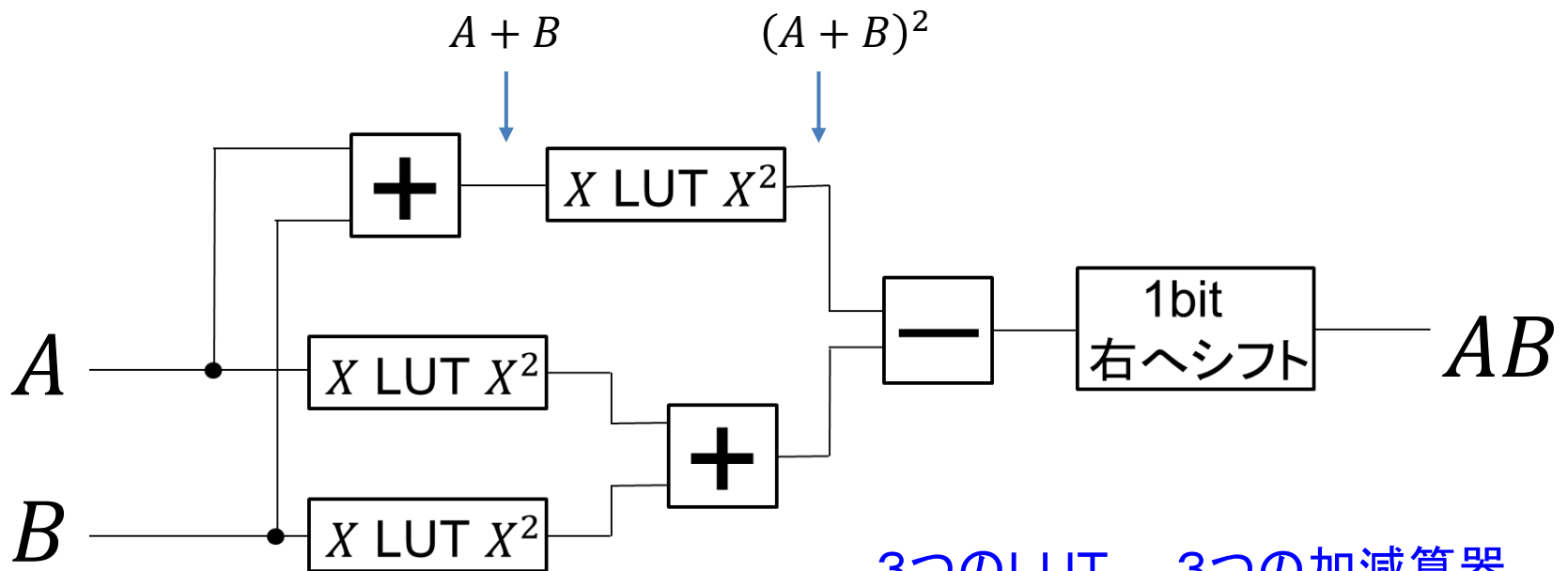
加算回数 : 1回
減算回数 : 2回
LUT参照 : 3回

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.....



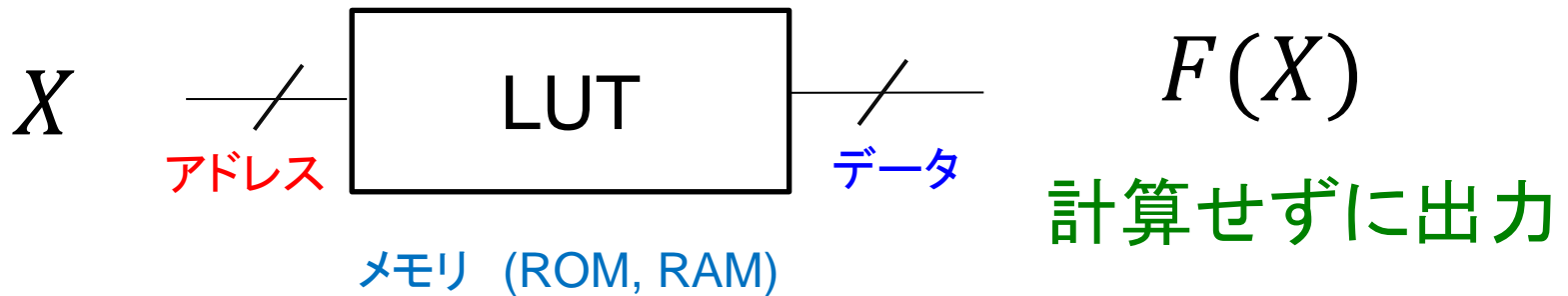
$F(X)$
計算せずに出力

アドレス	メモリ	データ
1	1	1
2	4	4
3	9	9
⋮	⋮	⋮
100	10000	10000

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……



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

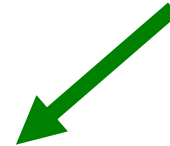
- ・ 簡単に2乗演算計算
- ・ メモリの参照処理のみ
- ・ 回路設計が容易
- ・ 計算速度の向上

欠点

- ・ 演算ビット数が多い



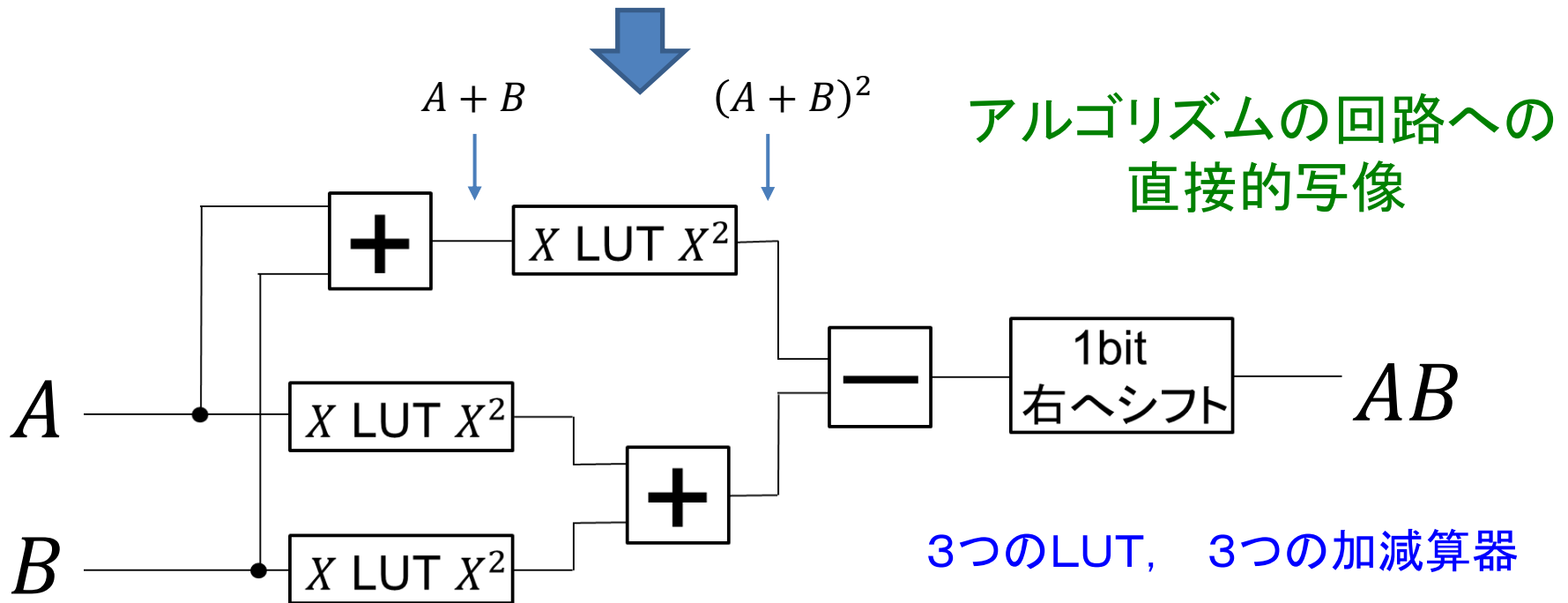
回路規模が大



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

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



※演算するbit数によってLUTのサイズが大きくなってしまふ

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:

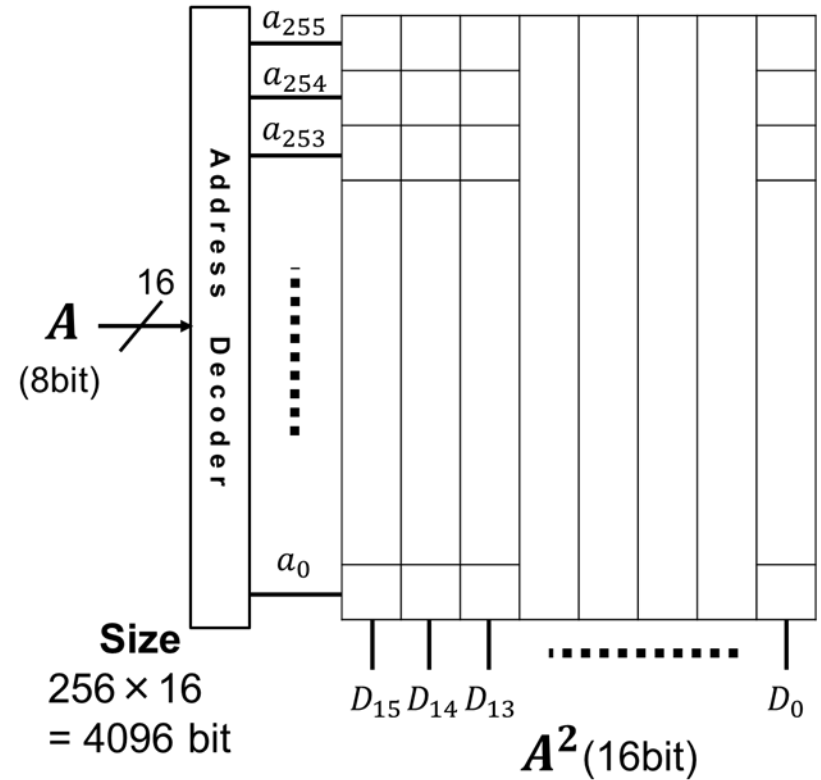
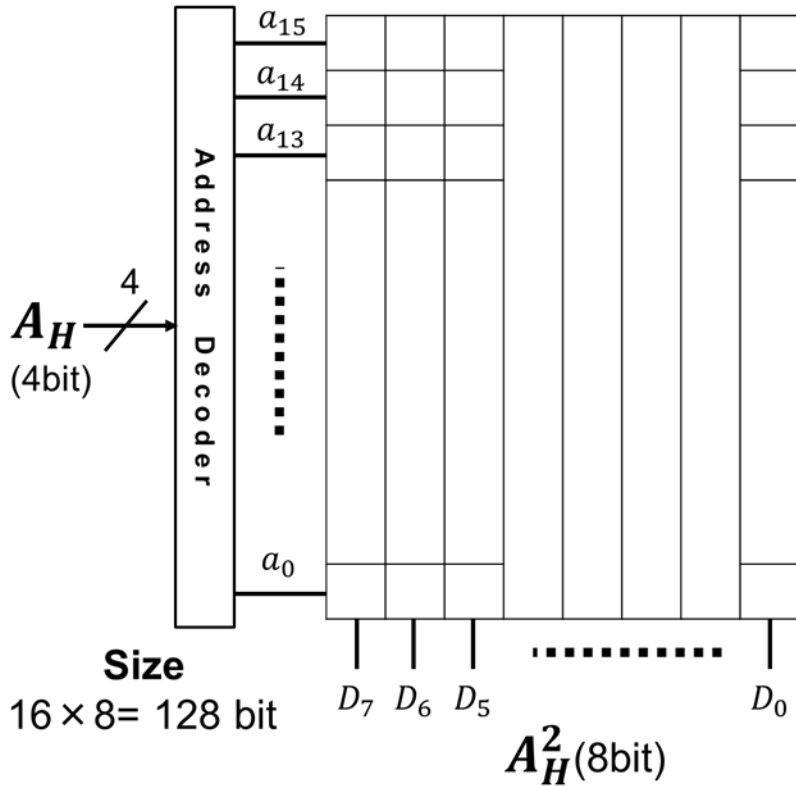
支配下の都市相互の連帯を禁じ、
都市毎に処遇に格差をつける

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。



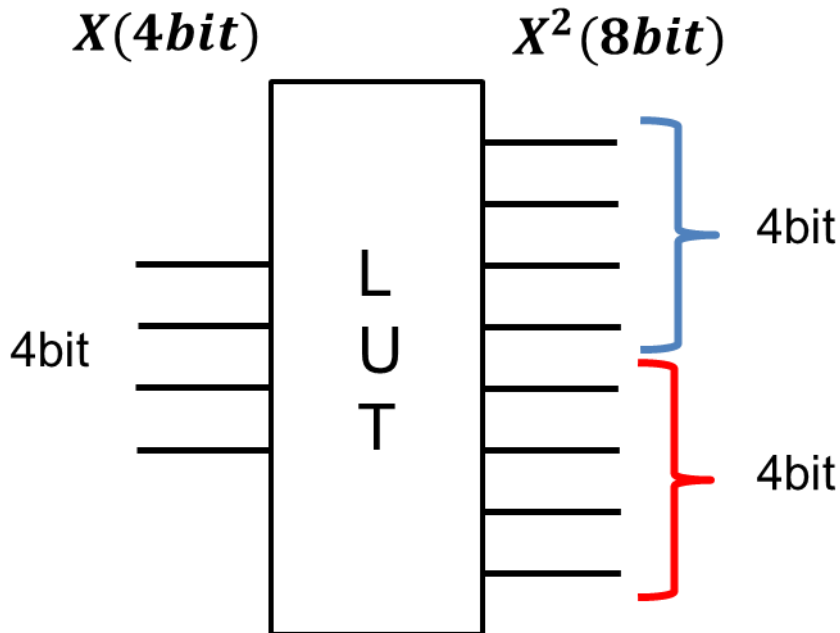
4bitでは**128bit**

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

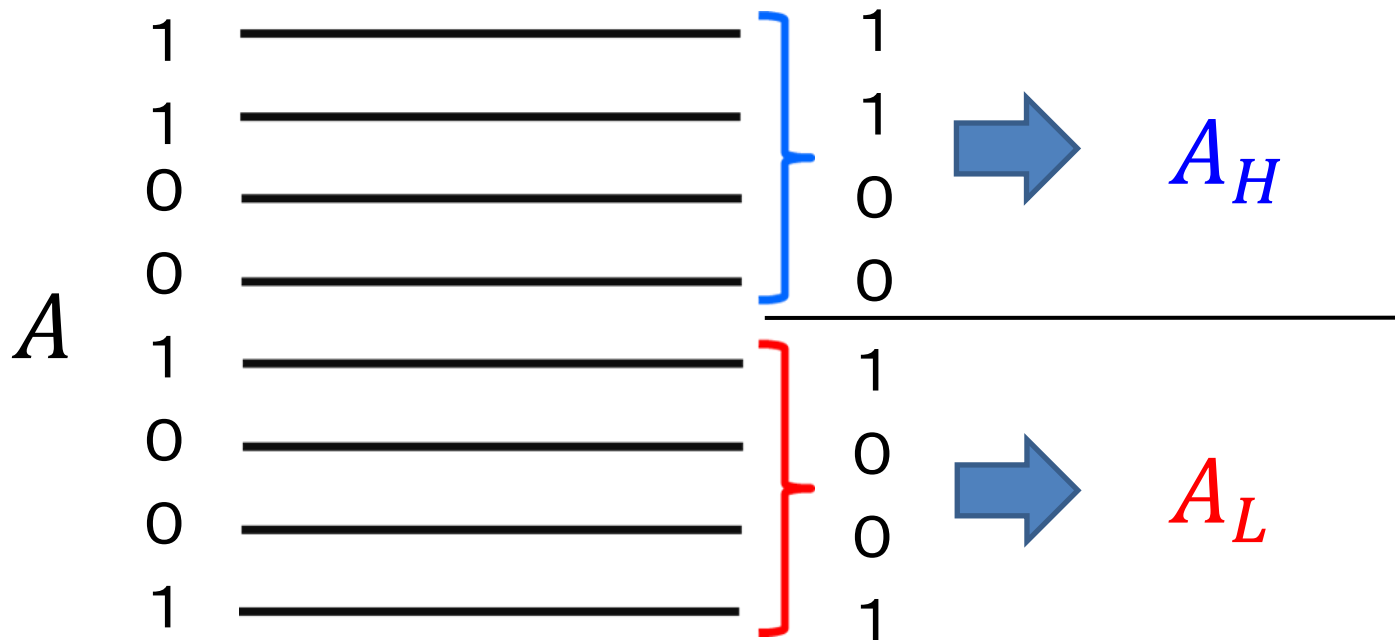
ルネ・デカルト

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

提案するDivide & Conquer法

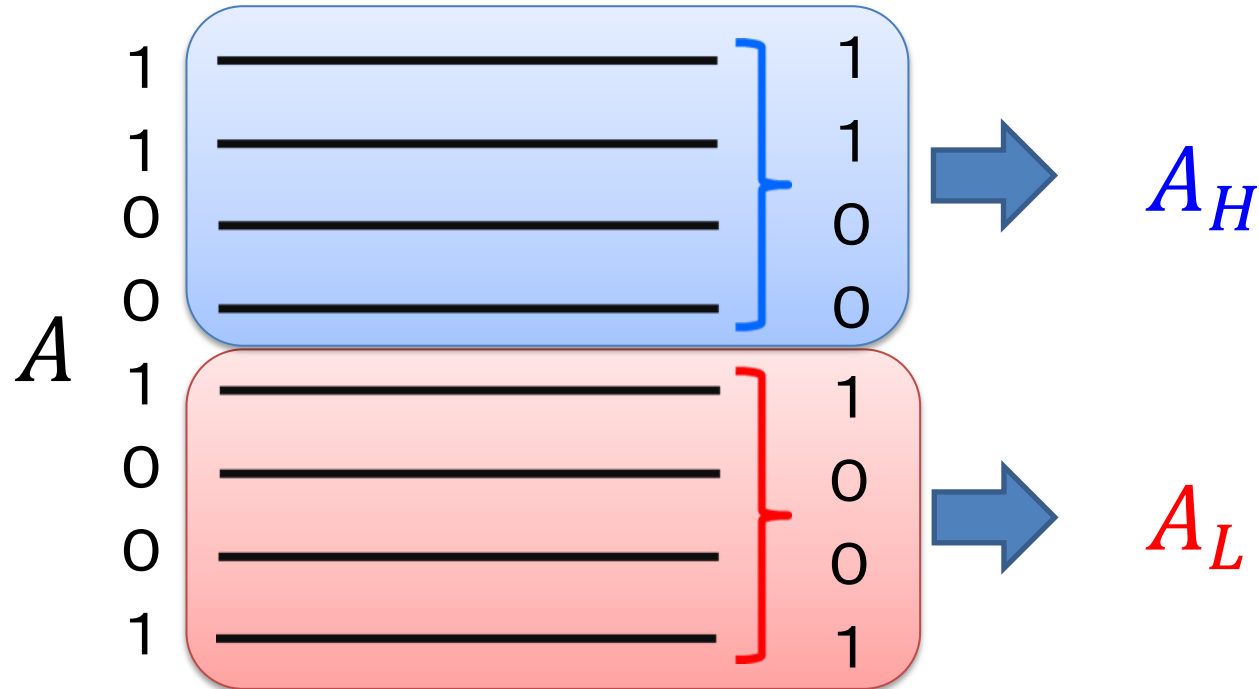
例: 8bit ($A=11001001$) について考える。



$$A^2 = A_H^2 (8bit \text{左シフト}) + A_H A_L (5bit \text{左シフト}) + A_L^2$$

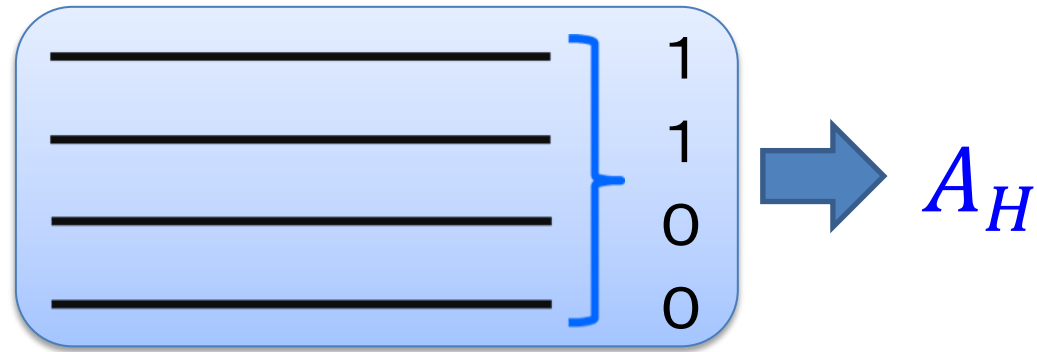
8bit x 8bit 乗算 $[A^2]$ を
4bit $[A_H]$, $[A_L]$ 毎の計算で求める。

提案するDivide & Conquer法

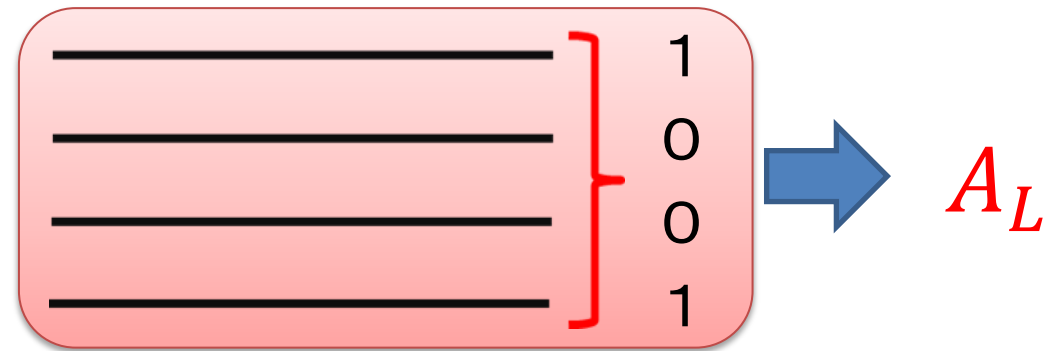


入力、出力された値を上下に分割 (Divide)

提案するDivide & Conquer法

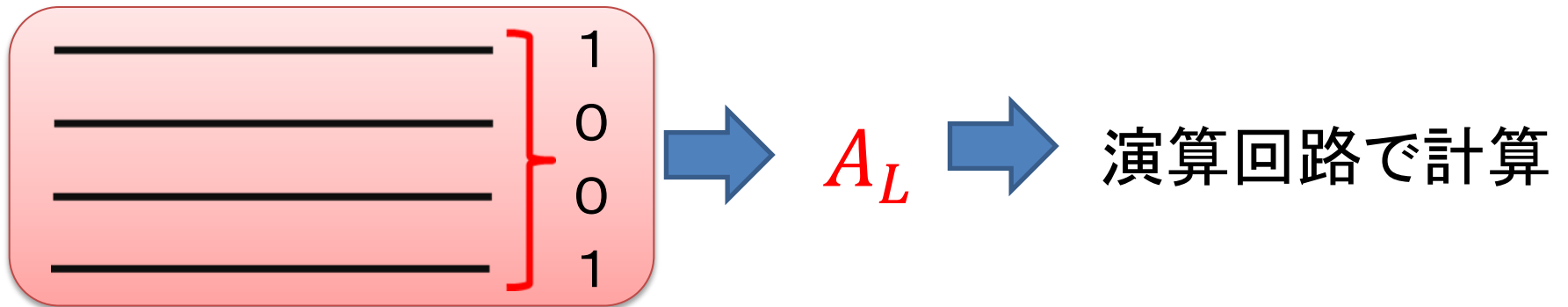
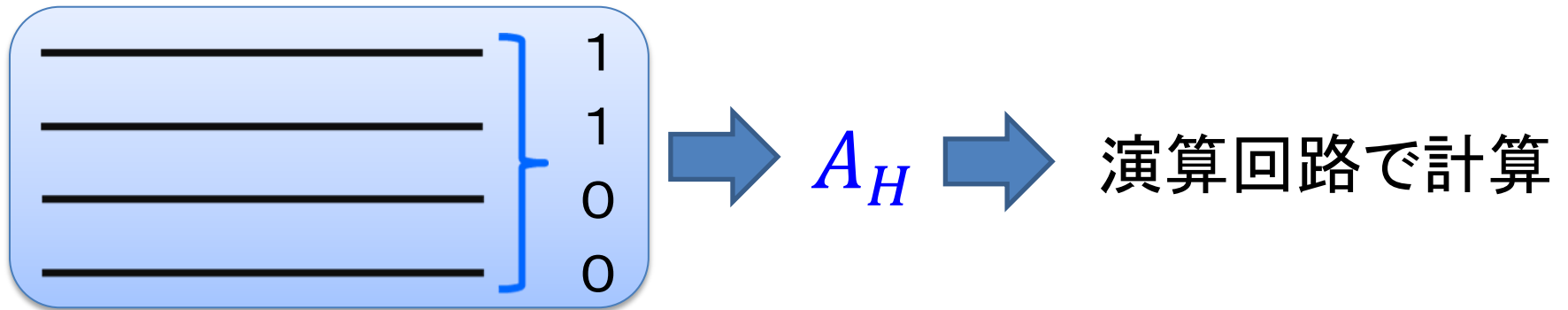


A



入力、出力された値を上下に分割 (Divide)

提案するDivide & Conquer法



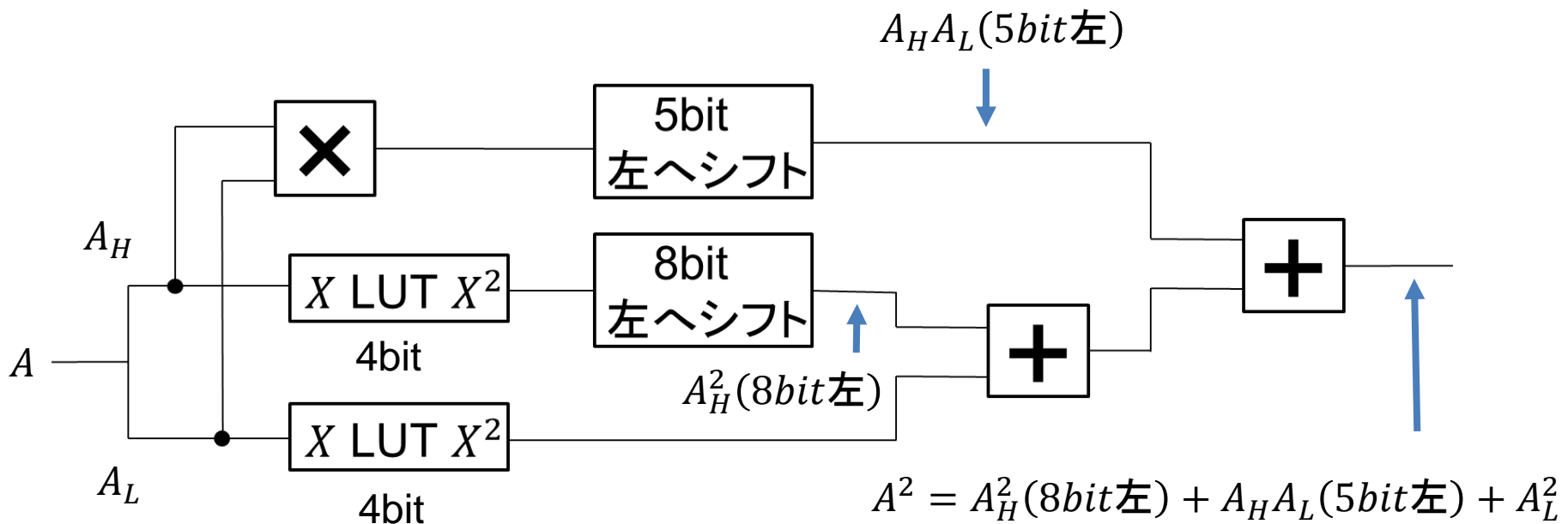
分割した値をそれぞれ計算：征服 (Conquer)

Divide & Conquer の計算回路(8bit)

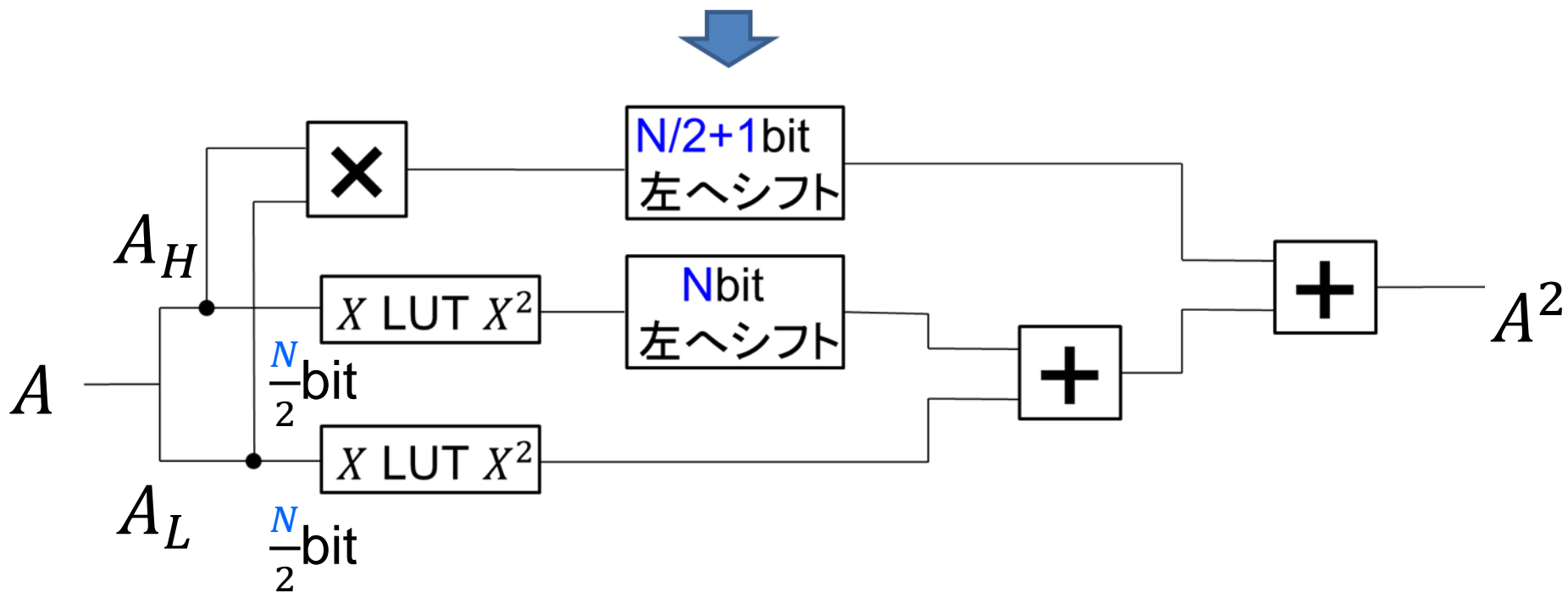
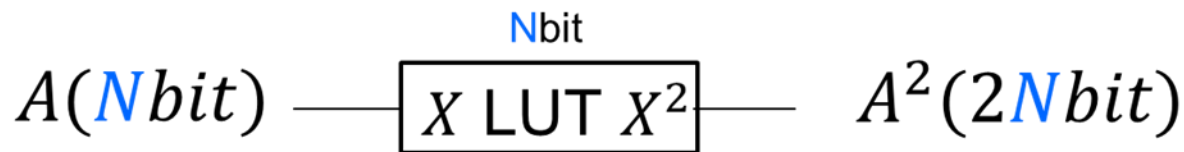
$$A^2 = A_H^2 (8bit \text{左シフト}) + A_H A_L (5bit \text{左シフト}) + A_L^2$$



この演算を実現する回路



検討アルゴリズム乗算回路(Nbit)



Divide & Conquer 方式

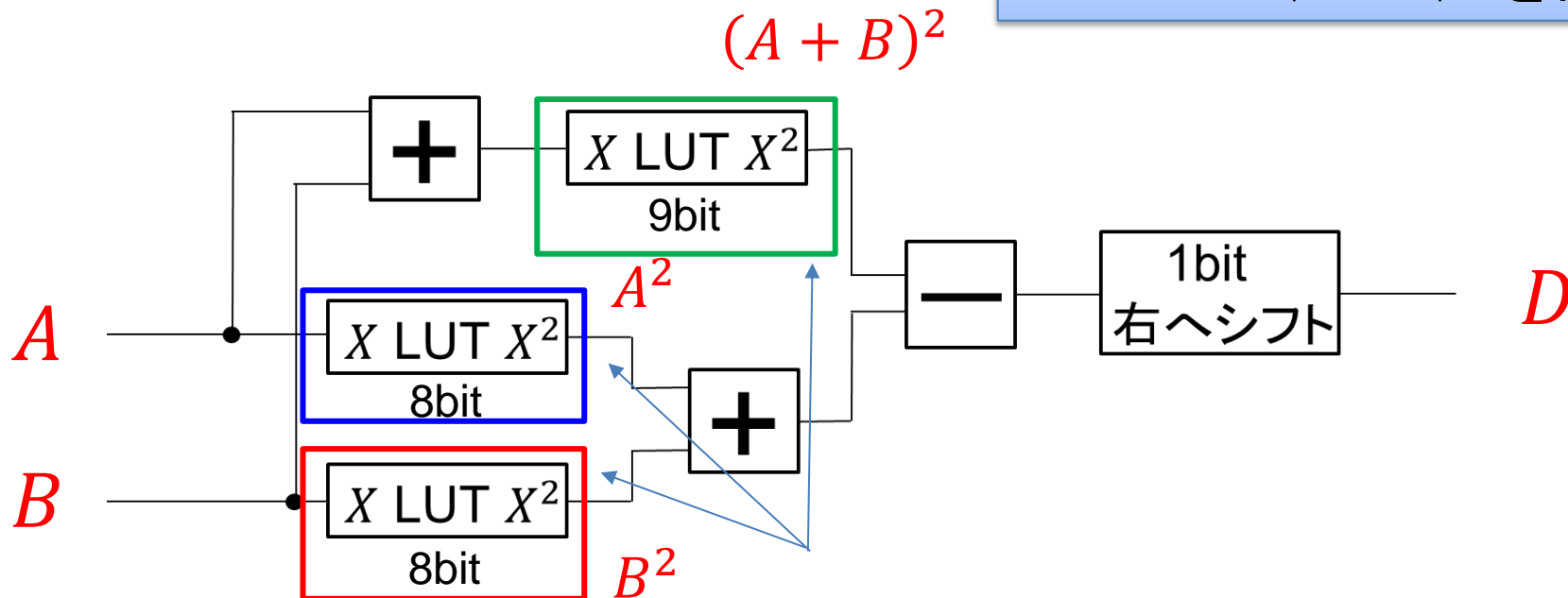
\times 回使用し、LUTサイズを $\left(\frac{1}{2}\right)^X$ にできる

OUTLINE

- 研究背景
- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計と
シミュレーション検証
- 各方式の比較
- 今後の検討回路
- まとめ

検討乗算回路(8bit) (Divide&Conquer 無)

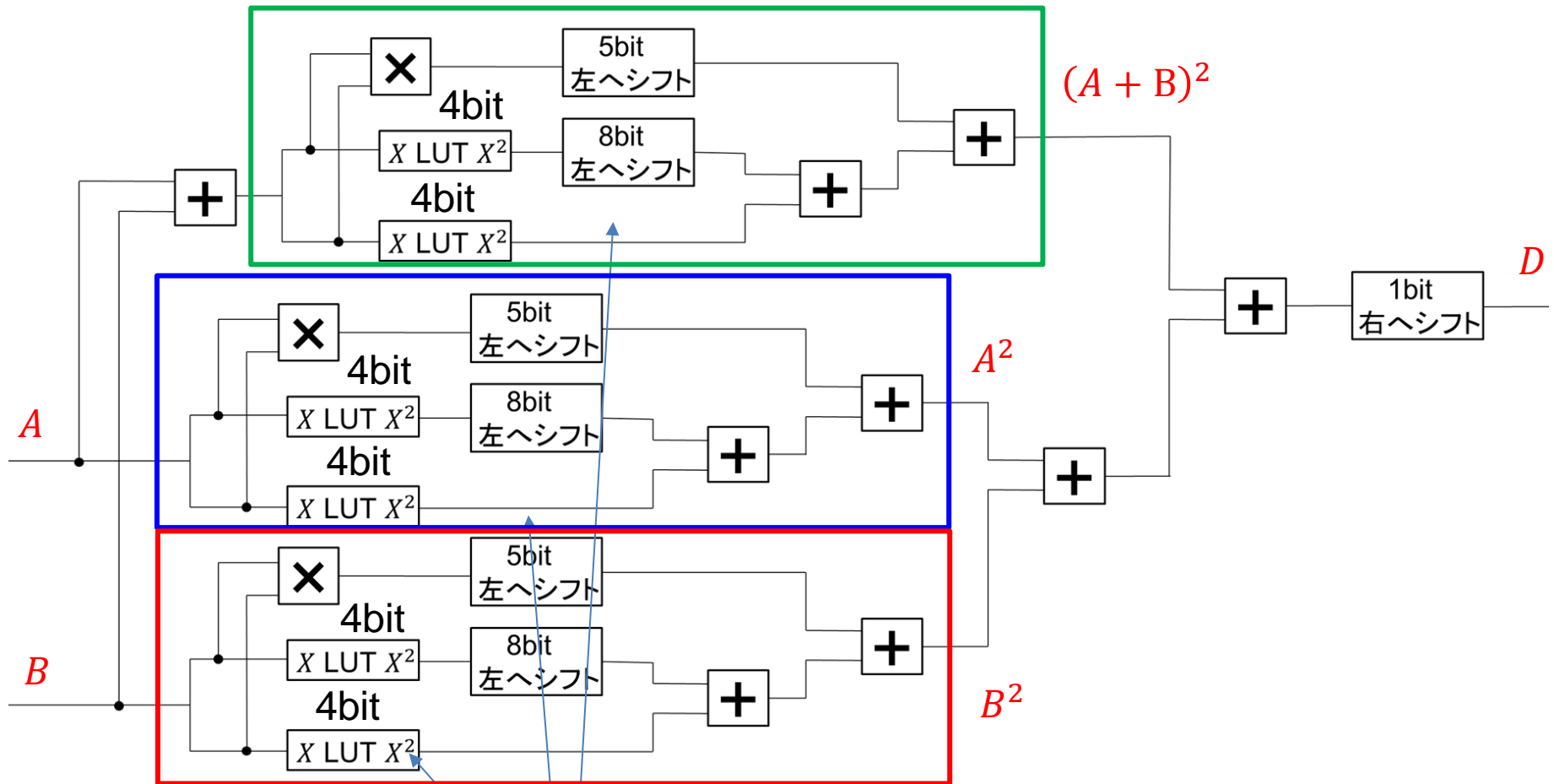
A,Bに8bitデータを入力
D = A×B (16bit) を出力



8bit用のLUTを用意するため
回路規模が大きくなる

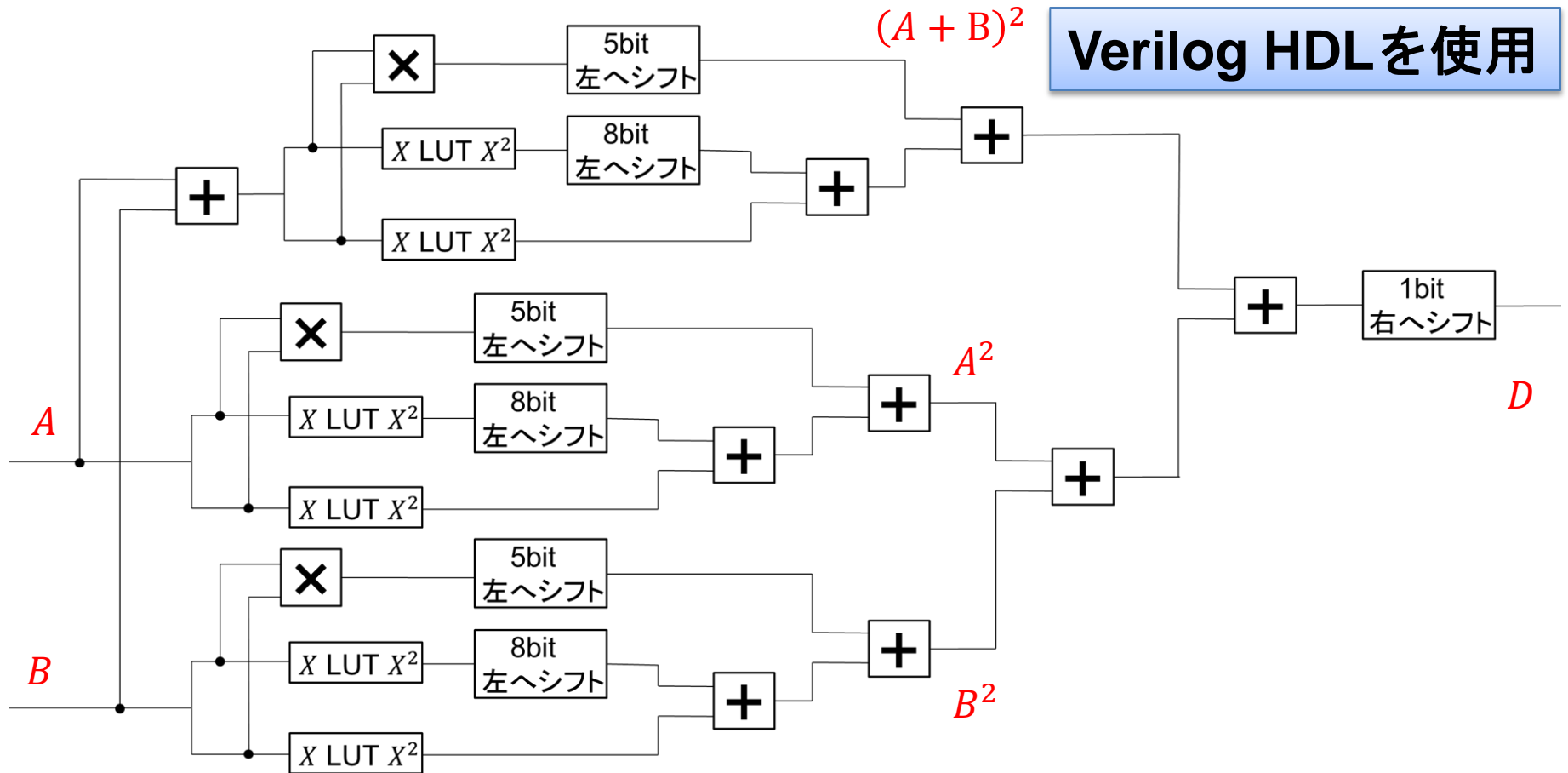
この回路をさらに変換する

検討乗算回路(8bit) (Divide&Conquer有)



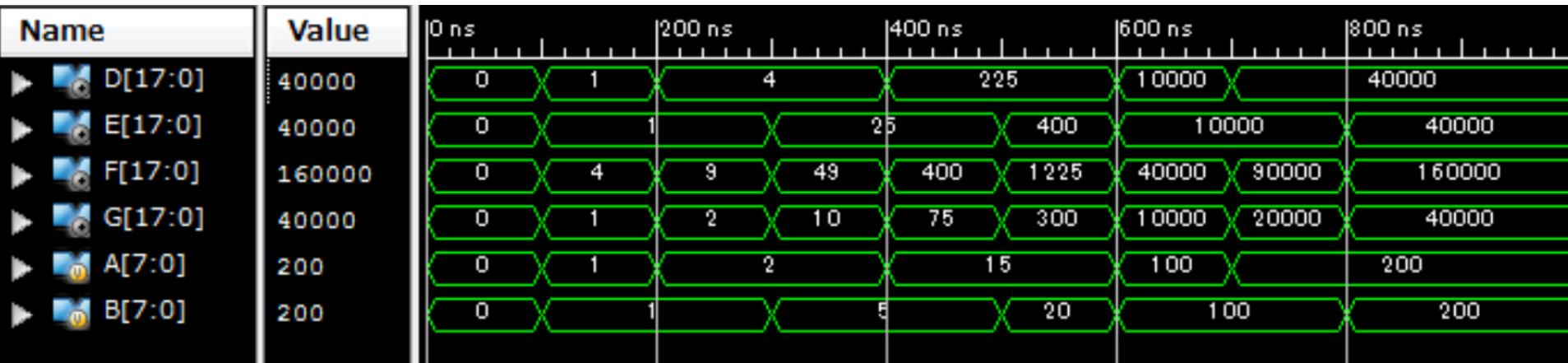
分割でLUTのbit数が小さくなる

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

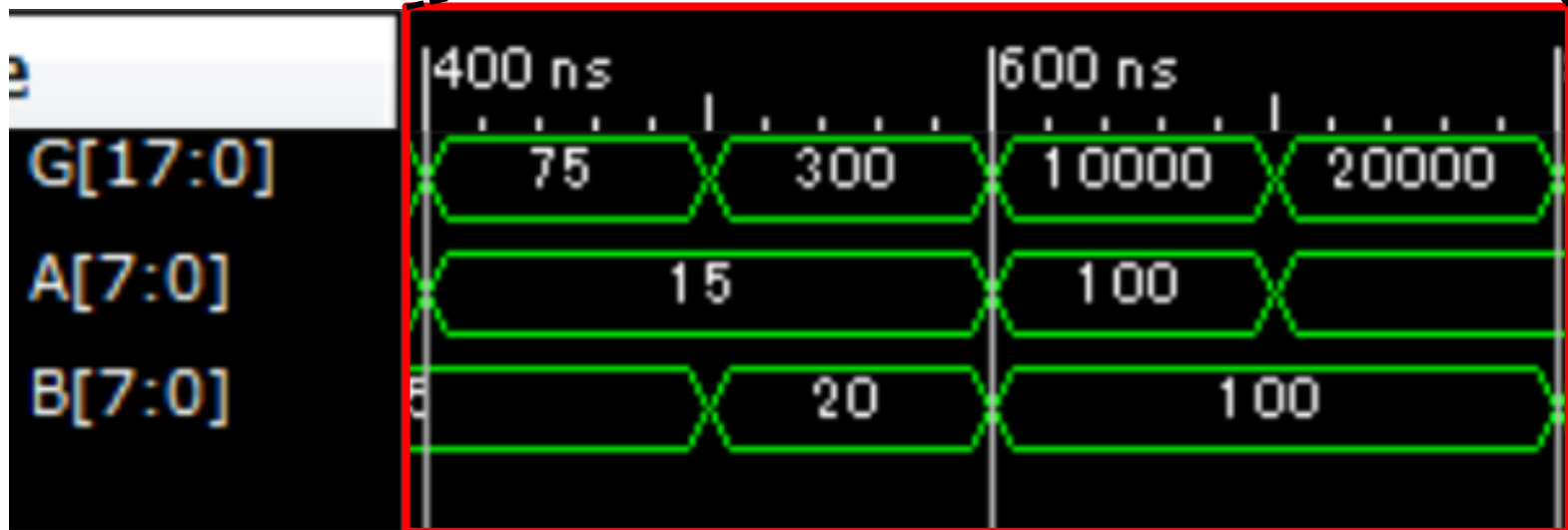
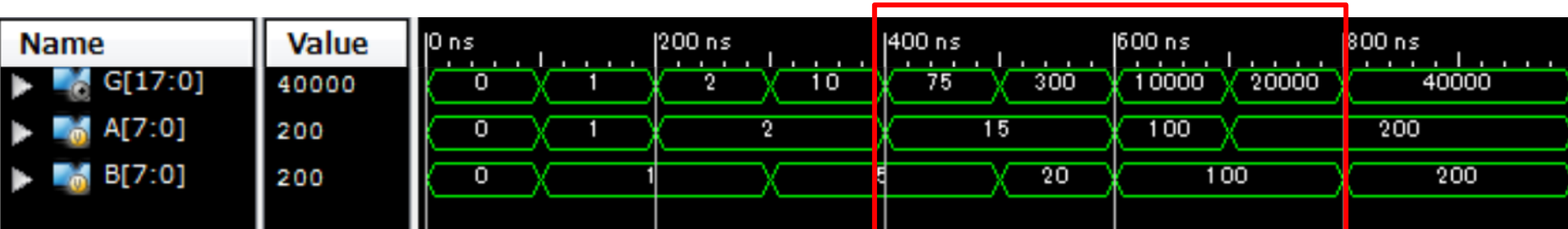
$$D(18\text{bit}) = A^2$$

$$E(18\text{bit}) = B^2$$

$$F(18\text{bit}) = (A + B)^2,$$

$$G(18\text{bit}) = \frac{1}{2} \{ (A + B)^2 - A^2 - B^2 \}$$

RTLシミュレーション (8bit × 8bit)



入力 A, B に対し $G = A \times B$ が出力

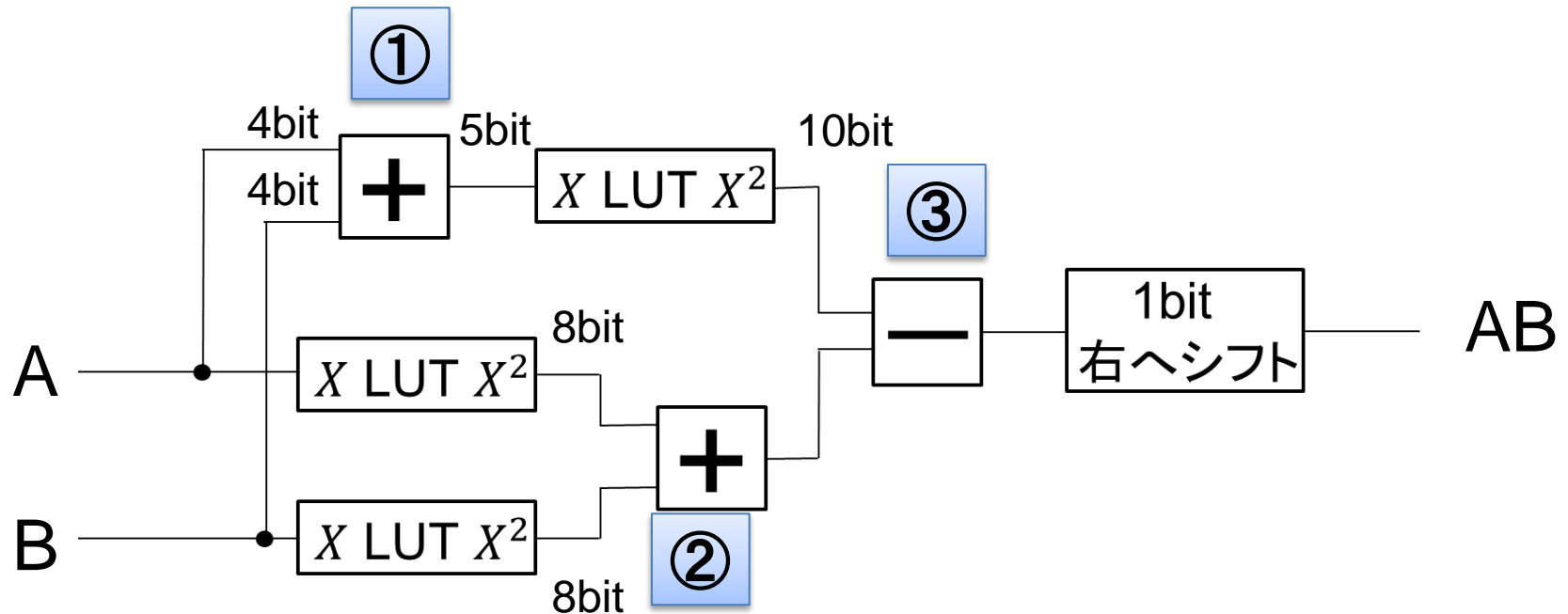
RTLシミュレーション

- 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

OUTLINE

- 研究背景
- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計とシミュレーション検証
- 各方式の比較
- 今後の検討回路
- まとめ

4 × 4bitの乗算器比較説明



全加算器の数は加数、被加数どちらか高いほうを参照にするので

① : 4個

② : 8個

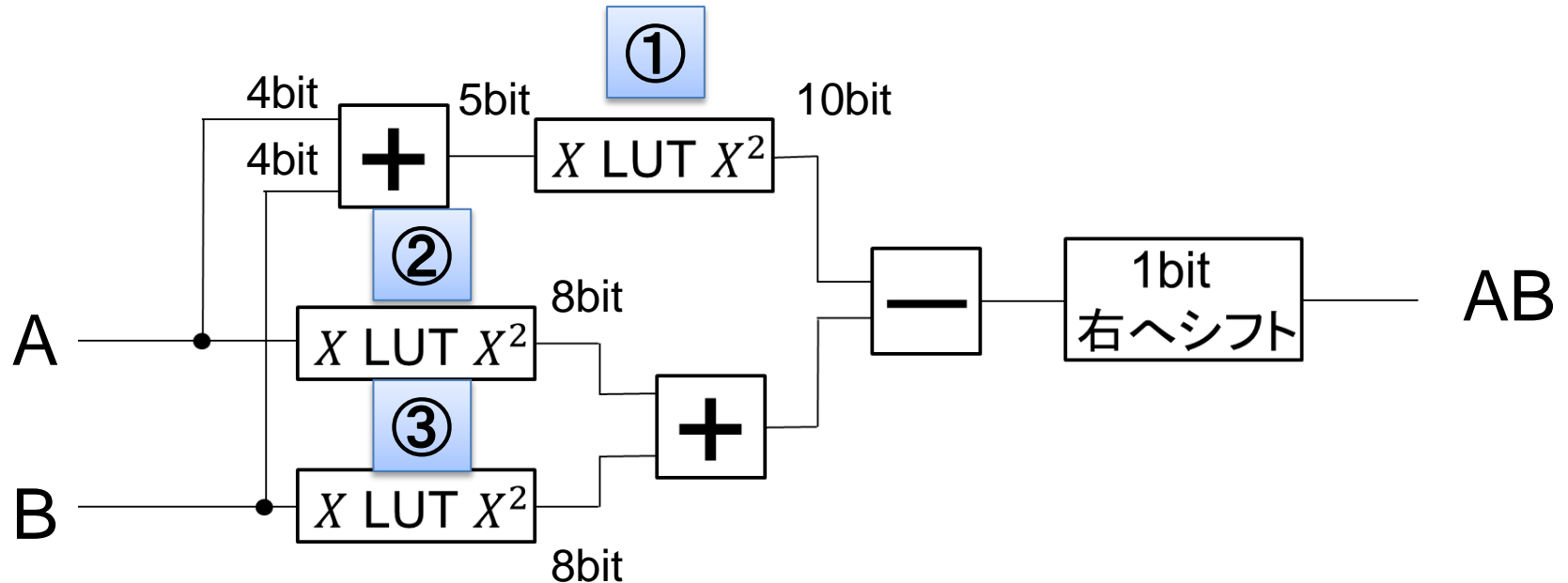
③ : 10個

合計22個

加算器1個、減算器2個あるので
加算(減算)回数は合計3回

LUTは3個

4 × 4bitの乗算器比較説明



各LUTの入カアドレスの数と、出力bitの数を掛け合わせると

① : 入カアドレス : 5bit($2^5 = 32$) 出力bit=10bit,

$32 \times 10 = 320\text{bit}$

② : 入カアドレス : 4bit($2^4 = 16$) 出力bit=8bit,

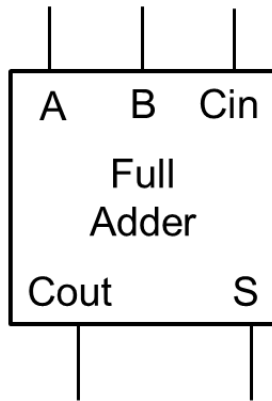
$16 \times 8 = 128\text{bit}$

③ : 1入カアドレス : 4bit($2^4 = 16$) 出力bit=8bit,

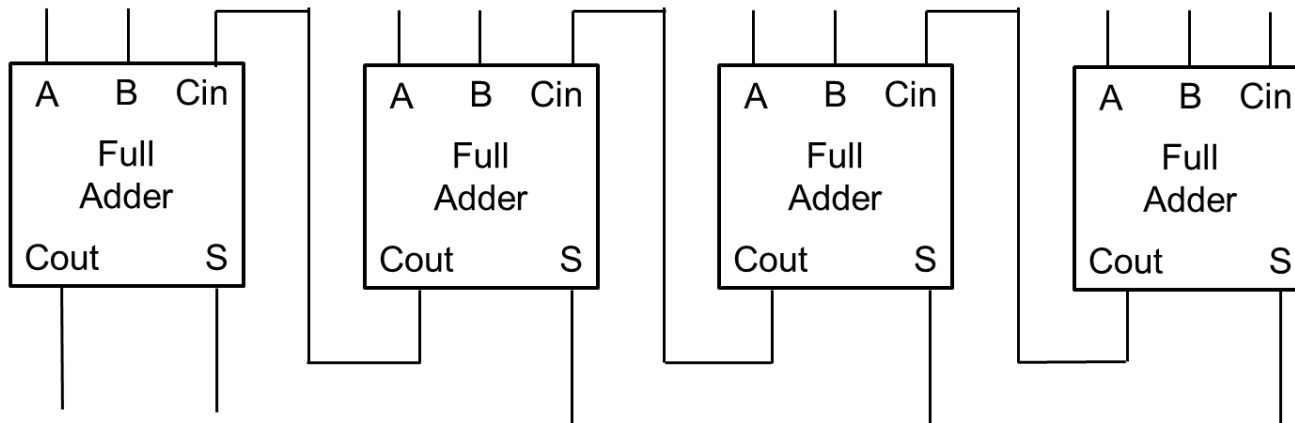
$16 \times 8 = 128\text{bit}$

合計 : $320 + 128 + 128 = 576\text{bit}$

演算時間の定義付け



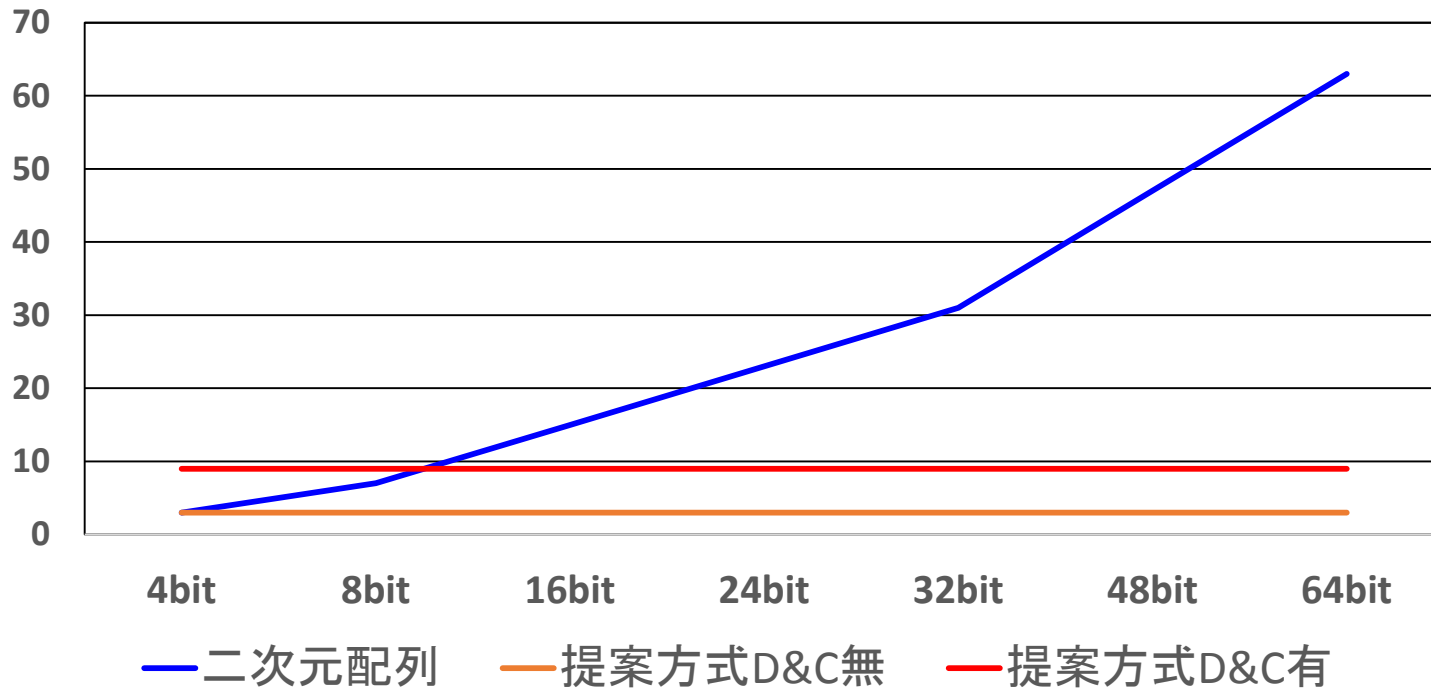
全加算器 1 つにかかる時間を T とする



4bit + 4bitの計算では加算器は 4 つ必要なため、必要時間は $4T$

各乗算器の比較表

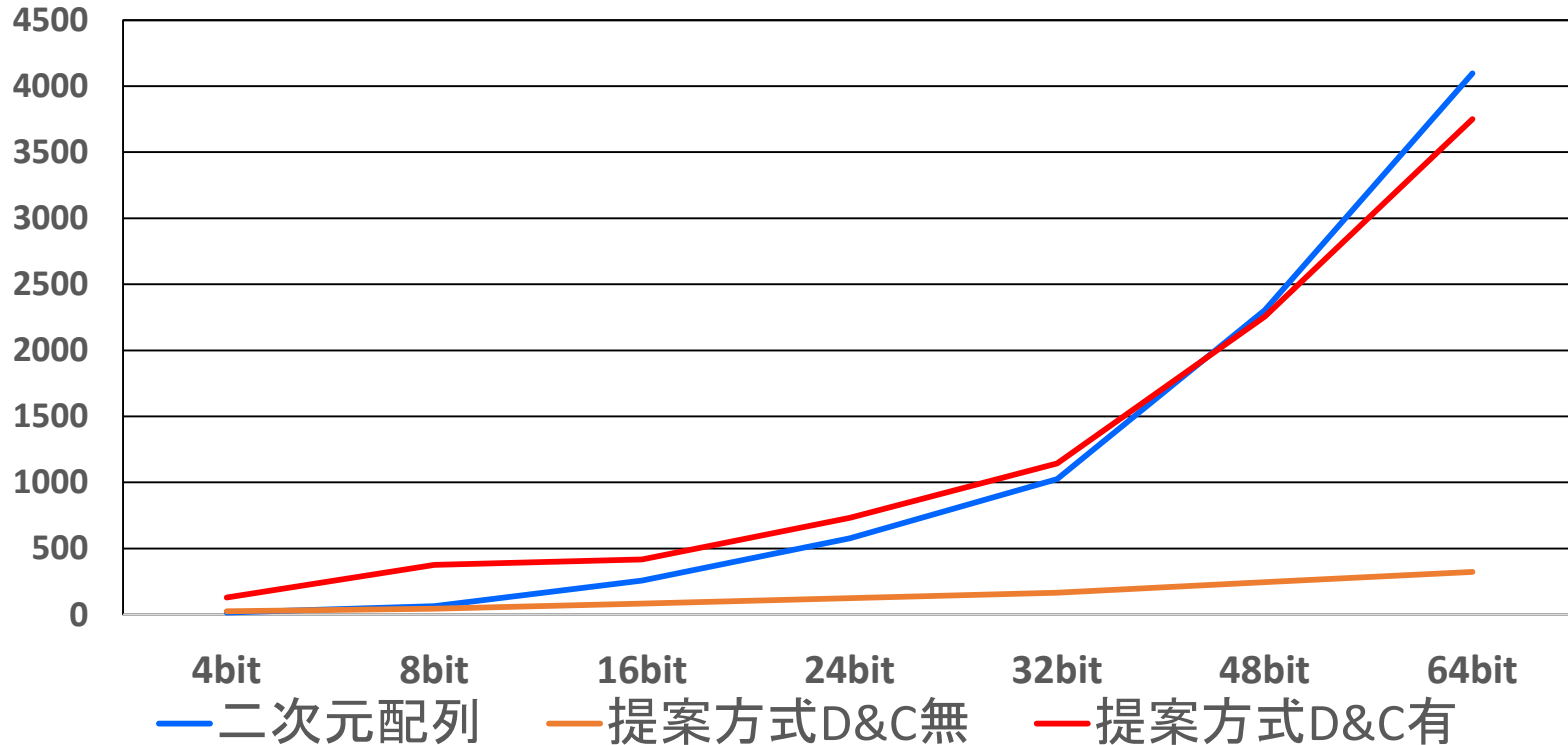
加算回数



従来方式よりも提案方式のほうが
加算回数が少なくなっている。

各乗算器の比較表

全加算器の合計演算時間



従来方式よりも提案方式のほうが
加算器にかける時間が少なくなっている。

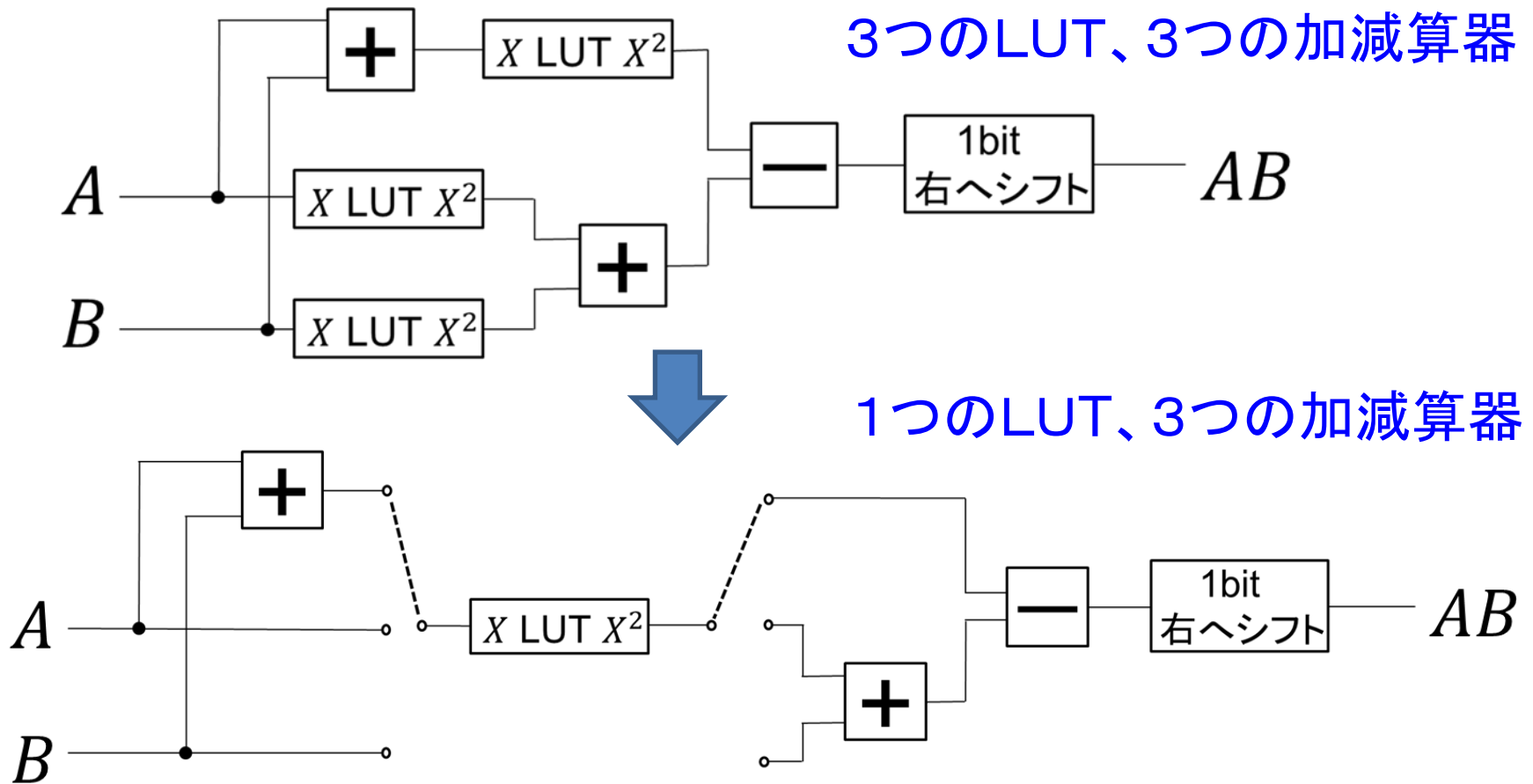
OUTLINE

- 研究背景
- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計とシミュレーション検証
- 各方式の比較
- 今後の検討回路
- まとめ

今後の検討回路

- 演算器を逐次的に使用した回路設計
- 別のアルゴリズムを用いた回路設計

乗算器を逐次的に使用した回路設計

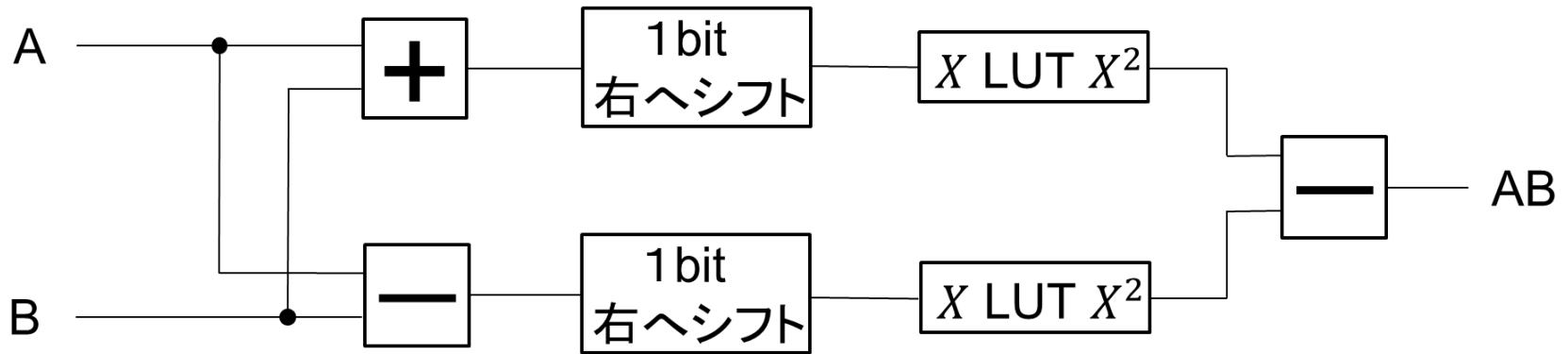


- 1つのLUTを $A^2, B^2, (A + B)^2$ の計算に逐次的に使用
全体として3分の1の回路量、計算時間は3倍

別のアルゴリズムを用いた回路設計

次の計算アルゴリズムも有力である。

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



加算回数: 1回

減算回数: 2回

LUT参照回数: 2回

OUTLINE

- 研究背景
- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計とシミュレーション検証
- 各方式の比較
- 今後の検討回路
- まとめ

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

Q & A (LSI合同ゼミ) 1

Q1:LUTを用いて4bit、8bitの2乗処理をしたが、4bit、8bitの乗算器を使うのとどちらが回路規模が大きいのか知りたい。

A:まだどちらが大きいかわからないので、今後の検討課題とします。

Q2:加算器の演算時間を示したが、パラレル処理かシリアル処理のどちらの演算時間を示したのか。パラレル処理ならば同時処理に費やせる時間はカットして示せるので、改めてきちんと検討しなおした方がいい。

A:シリアル処理で測っていたので、パラレル処理についてはまだ検討段階です。

Q3:D & Cを使う際に×(掛け算)を使っているが、もっとbit数が多い場合はどうするのか？

A:さらにD & Cを使って分割して×(掛け算)を使おうと考えています。

Q & A (LSI合同ゼミ) 2

意見1:加算器だけでも色々ある。加算器1つを高速で回すようなものも存在するのでもっと色々調べてから比較してほしい

意見2:LUTの参照時間も考えて、演算時間の比較をしてほしい。

意見3:LUTを使うようだが、FPGA内のメモリはそんなに大きくないので別の方法を考えた方がいいかもしれない

意見4:LUTの必要性→2乗専用回路みたいなものがあればいいのでは？

意見5:FPGAで実装とのことだが、DSPブロックを使わないメリットは何か？

意見6:D&Cよりもまず提案方式そのものの特徴を見つける。リソースを用意して並列処理をしたり、資源を使いまわす発想が必要。

意見7:FAをリソースとして、時間との対比を行う。

意見8:京大の高木先生という人の論文が色々あるので見てみたら？

意見9:FPGAを実装対象ではなく、スタンダードセルにしたらいいかもしれない。自動的に回路の構成が出来て時間なども分かる。詳しくはVDECのサイトで。

意見10:発想はとても面白い。慣れないデジタルかもしれないが頑張ってもらいたい。