

2乗則を用いた デジタル乗算アルゴリズムの検討

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
- 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
- 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



複雑な計算を要するデジタル通信システムの
チップ内に複数乗算器が必要

小規模回路、低消費電力、高速化
のため 様々なアルゴリズムが
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/>		
100000	: 64 ₁₀	結果

計算回数是不変

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

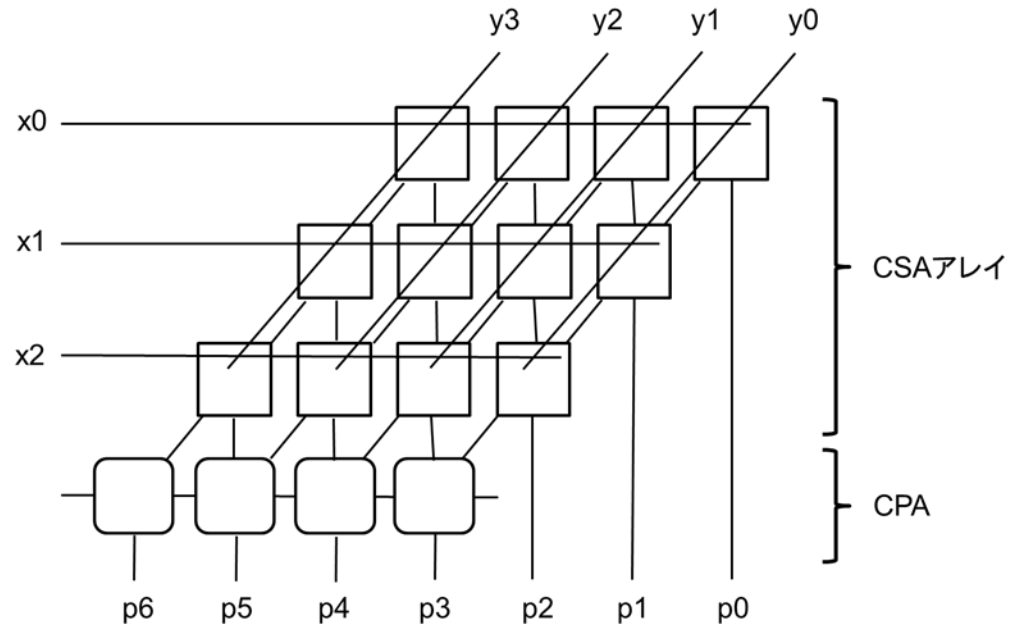
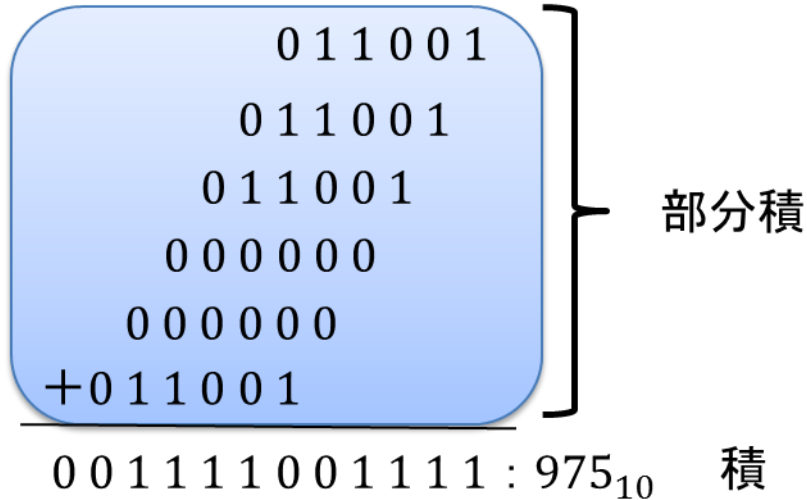
2進数での演算をベース

011001	: 25 ₁₀	被乗数
×100111	: 39 ₁₀	乗数
—		
011001	} 部分積	
011001		
011001		
000000		
000000		
+011001		
—		
001111001111	: 975 ₁₀	積

計算回数が多くなる

従来方式(デジタル乗算器)の問題点

0 1 1 0 0 1 : 25₁₀ 被乗数
× 0 1 1 0 0 1 : 39₁₀ 乗数



部分積 & 加算のため全加算器の2次元配列

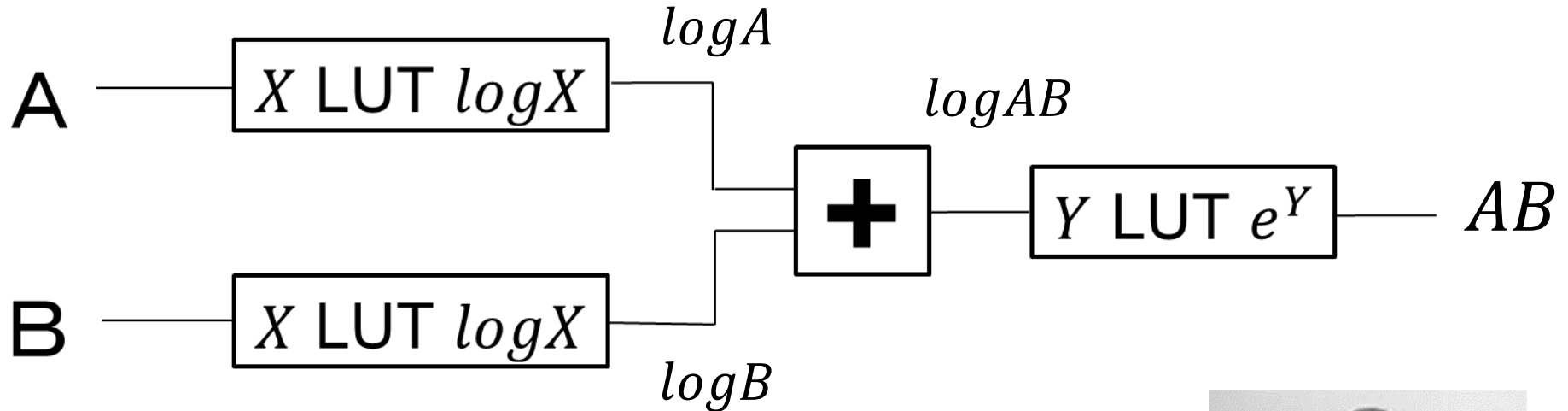
例: 8bit × 8bit の場合

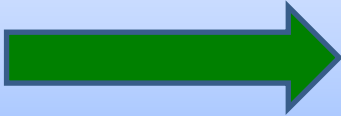
直接的構成では $8 \times 8 = 64$ 個の全加算器が必要

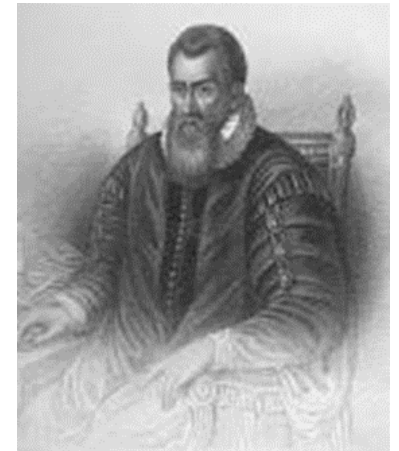
回路規模・消費電力・演算時間が大

従来方式(対数を用いた設計)の問題点

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
- 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

ルックアップテーブル (LUT) とは

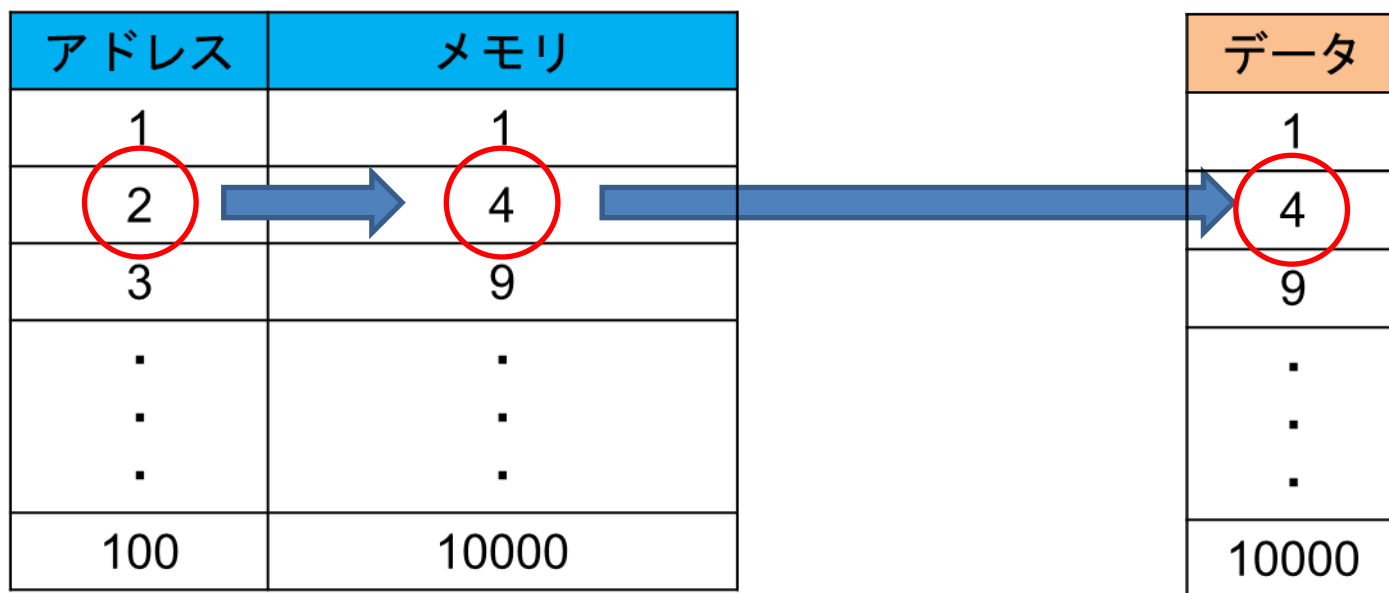
複雑な信号処理  計算時間がかかる

LUTを使うと……



$F(X)$
計算せずに出力

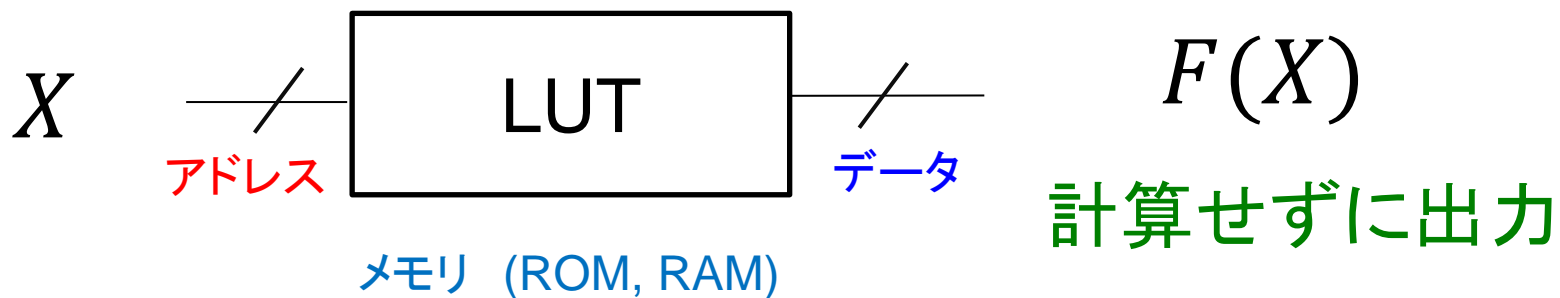
アドレス	メモリ	データ
1	1	1
2	4	4
3	9	9
⋮	⋮	⋮
100	10000	10000



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



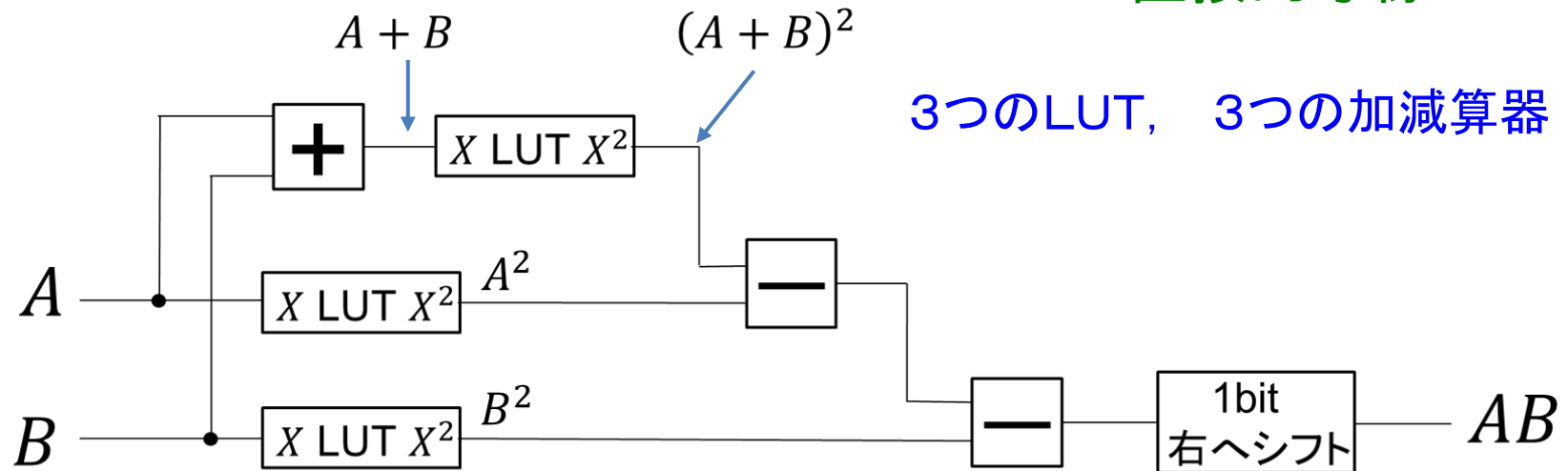
LUTを上位ビット、下位ビットに分割し
計算量・回路量を低減する方式
(Divide & Conquer)
を検討

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



※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:

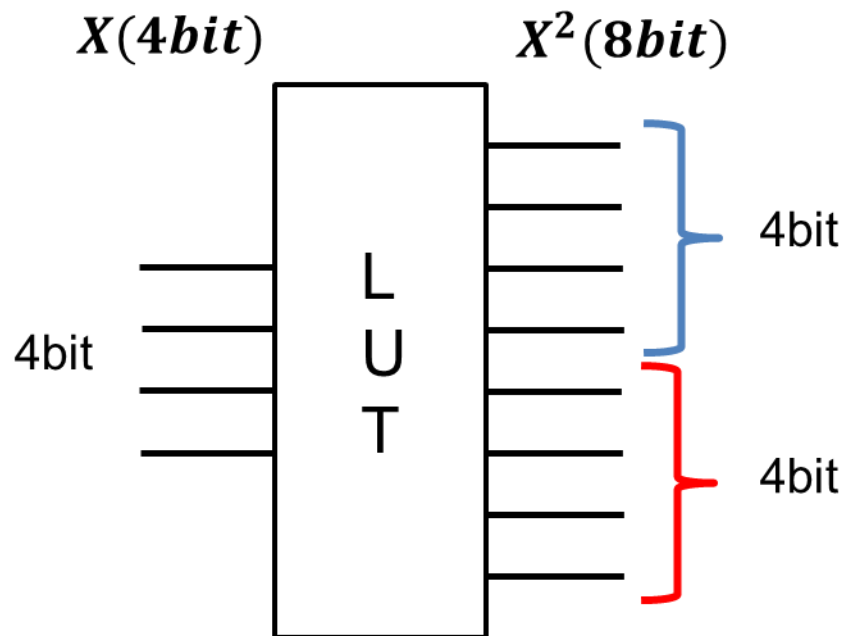
支配下の都市相互の連帯を禁じ、
都市毎に処遇に格差をつける

分割統治(divide & conquer)で
征服した都市の反乱を抑える。



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

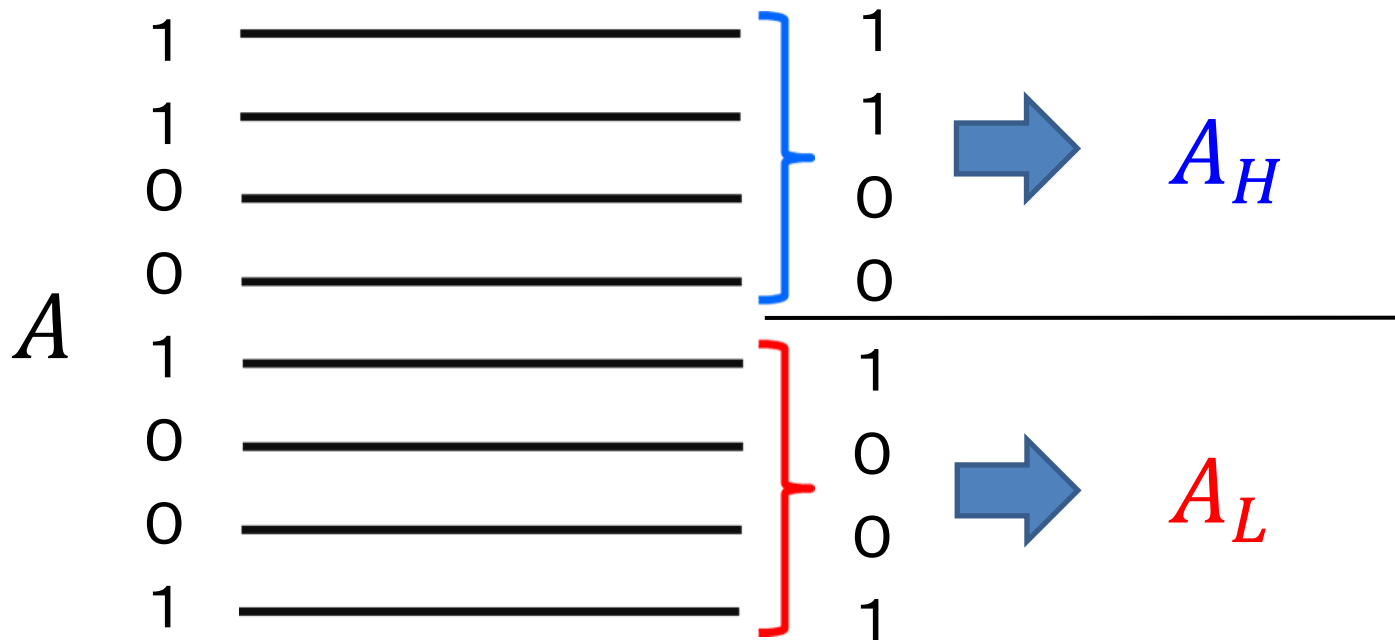
シフト演算で桁を揃える



演算で用意する
メモリを小さくできる

提案するDivide & Conquer法

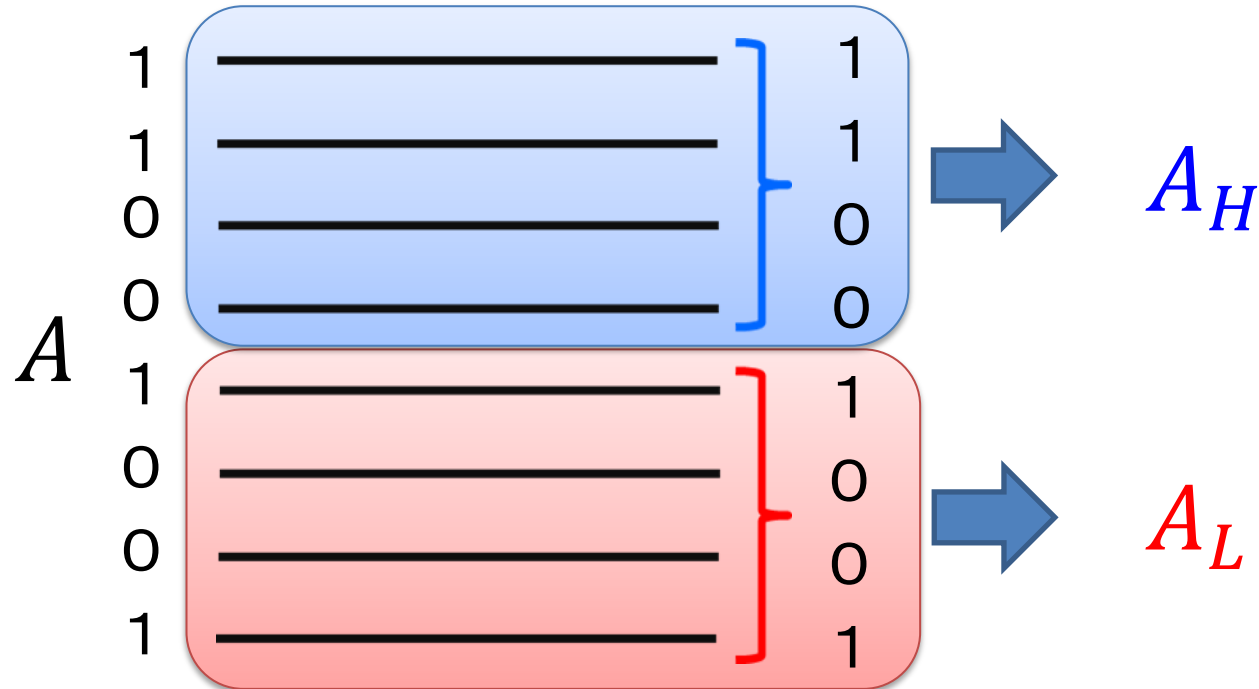
例: 8bit ($A=11001001$) について考える。



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

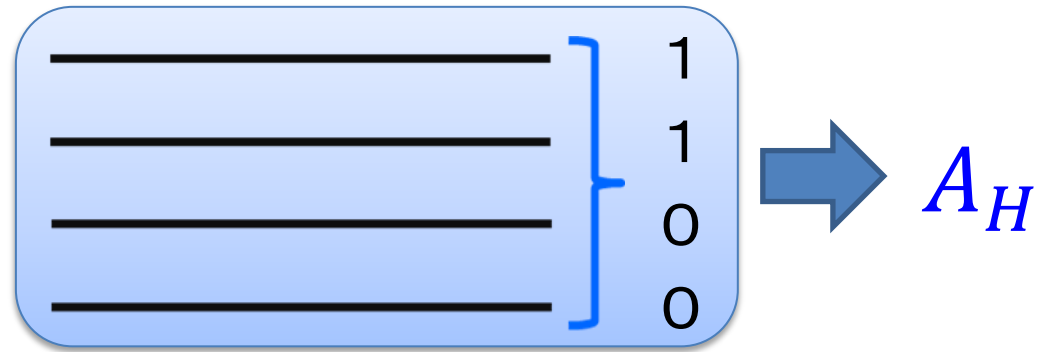
8bit x 8bit 乗算 $[A^2]$ を
4bit $[A_H]$, $[A_L]$ 毎の計算で求める。

提案するDivide & Conquer法

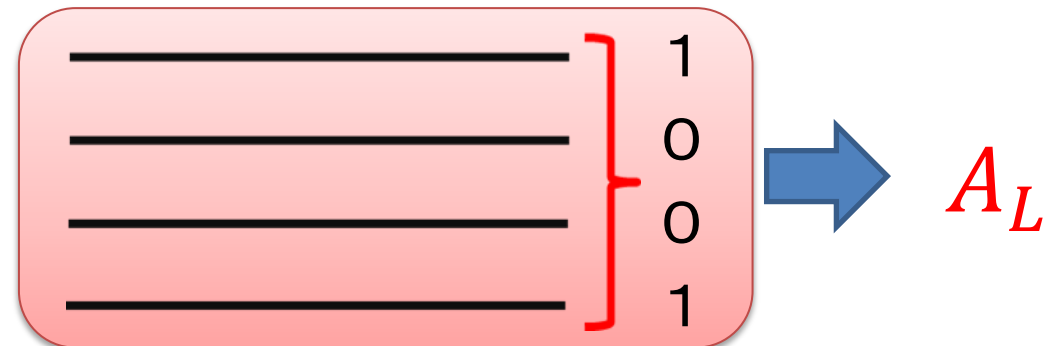


入力、出力された値を上下に分割 (Divide)

提案するDivide & Conquer法

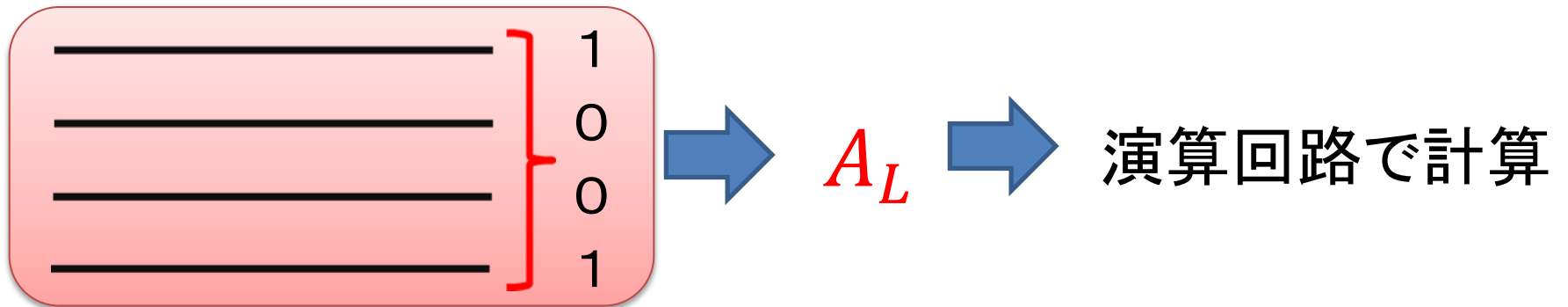
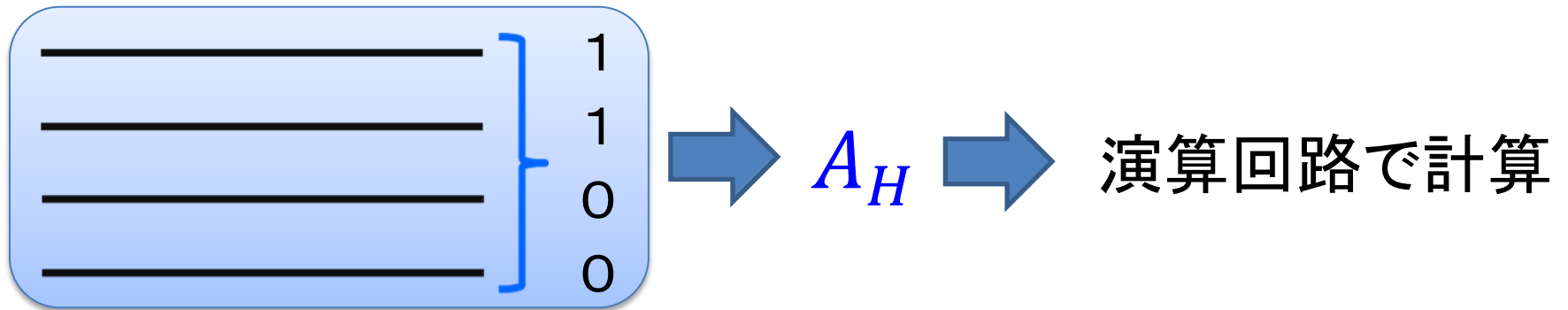


A



入力、出力された値を上下に分割 (Divide)

提案するDivide & Conquer法



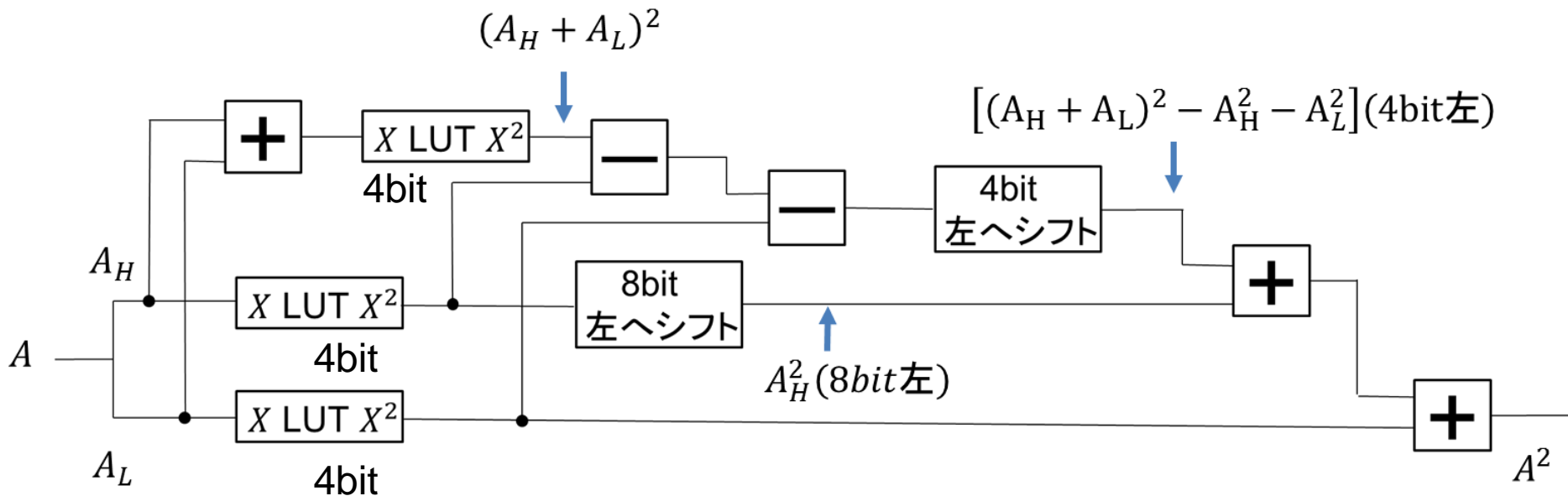
分割した値をそれぞれ計算：征服 (Conquer)

Divide & Conquer の計算回路(8bit)

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



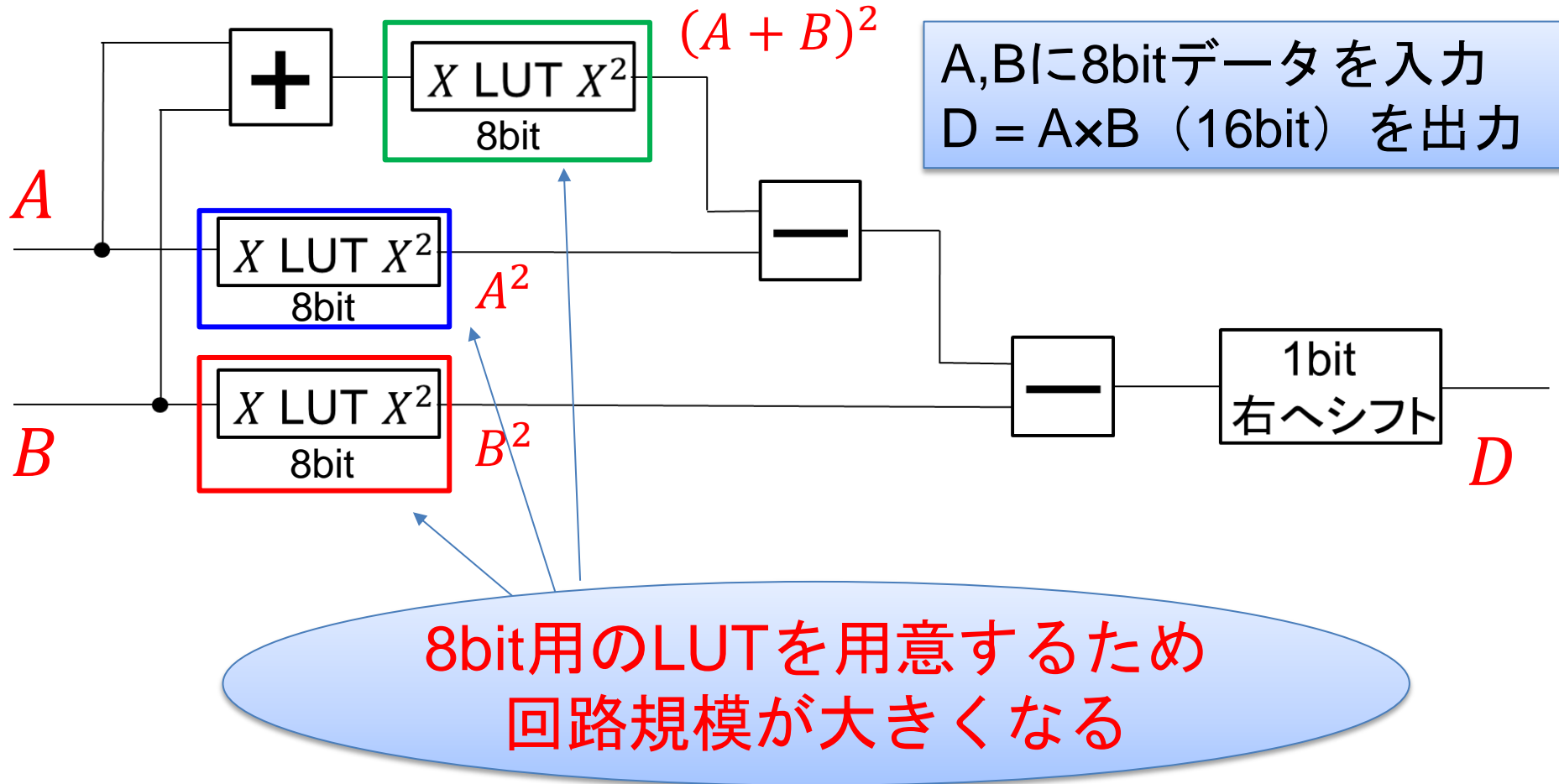
この演算を実現する回路



OUTLINE

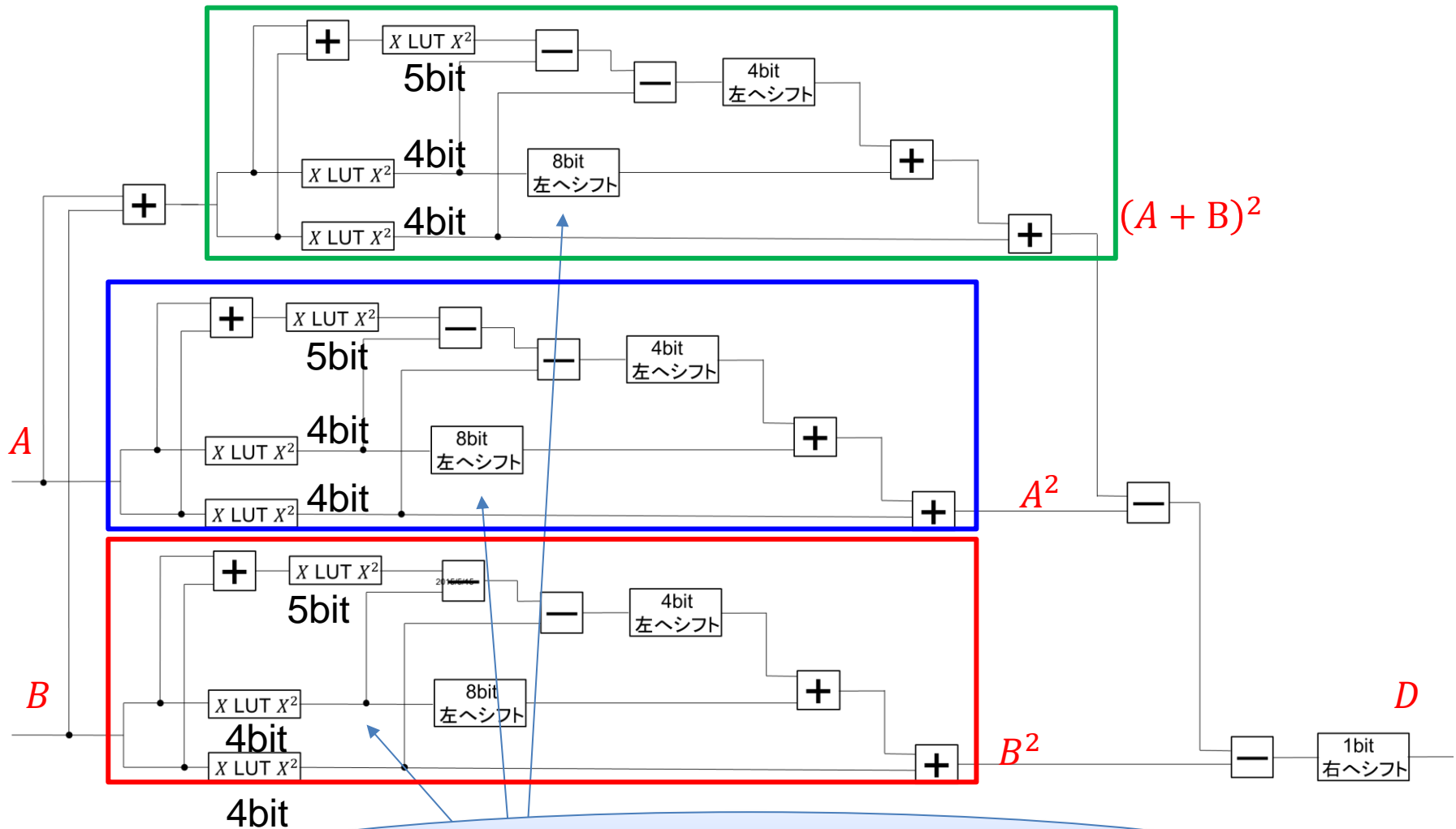
- 研究背景
- 検討乗算アルゴリズム
- 検討アルゴリズム乗算回路の設計と
シミュレーション検証
- 今後の検討回路
- まとめ

検討乗算回路(8bit) (Divide&Conquer 無)



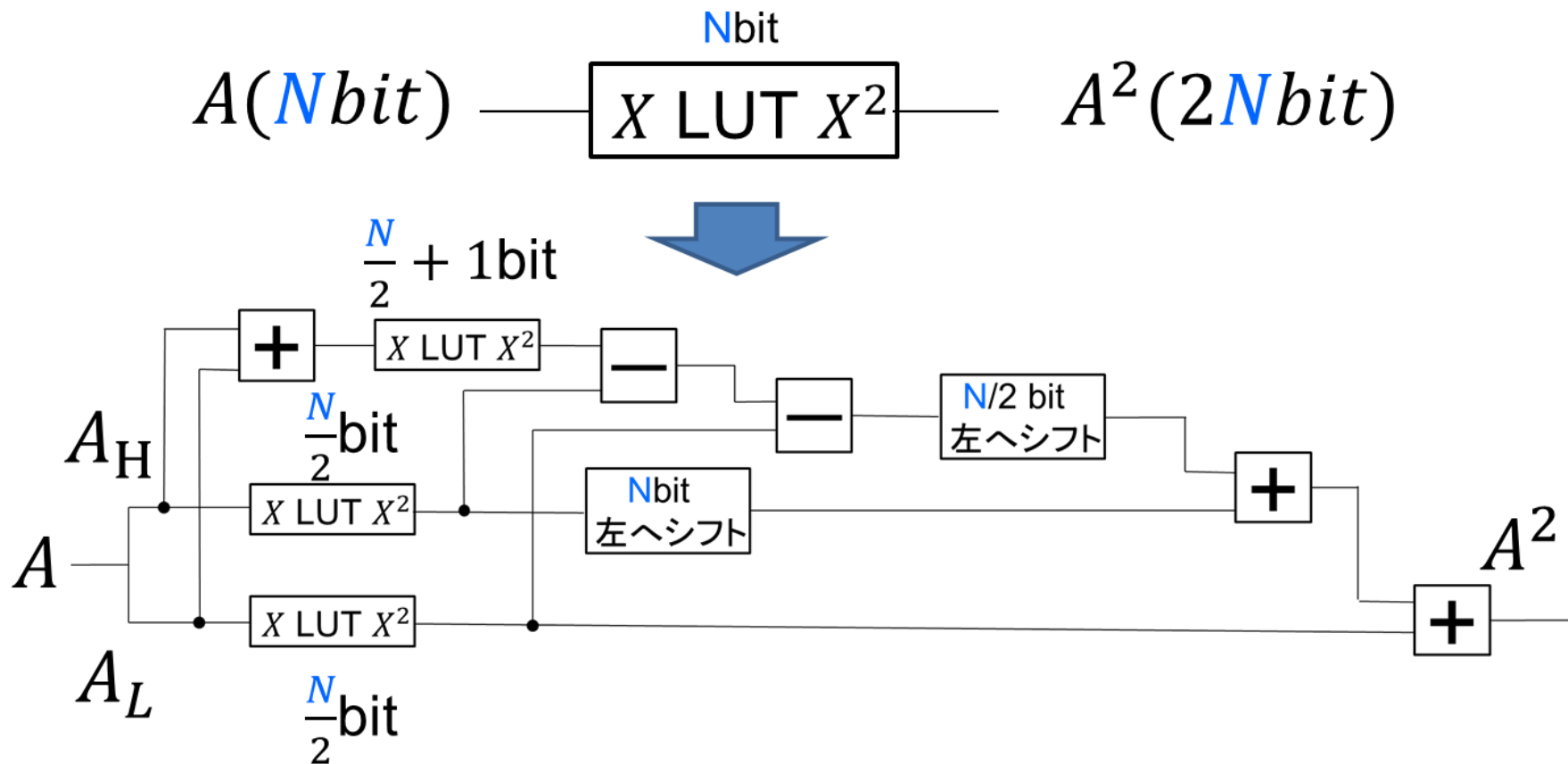
この回路をさらに変換する

検討乗算回路(8bit) (Divide&Conquer有)



分割でLUTのbit数が小さくなる

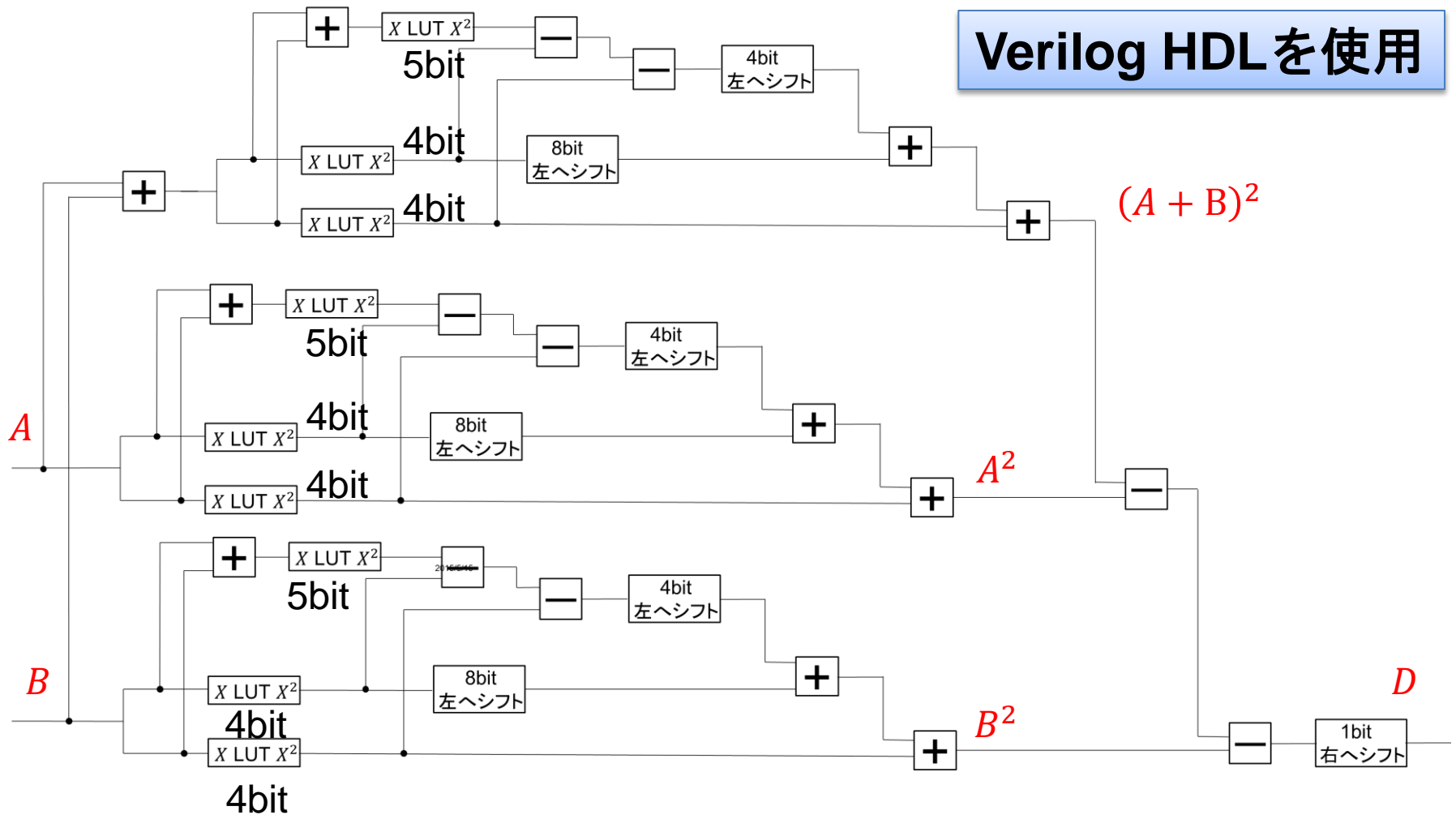
検討アルゴリズム乗算回路(Nbit)



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

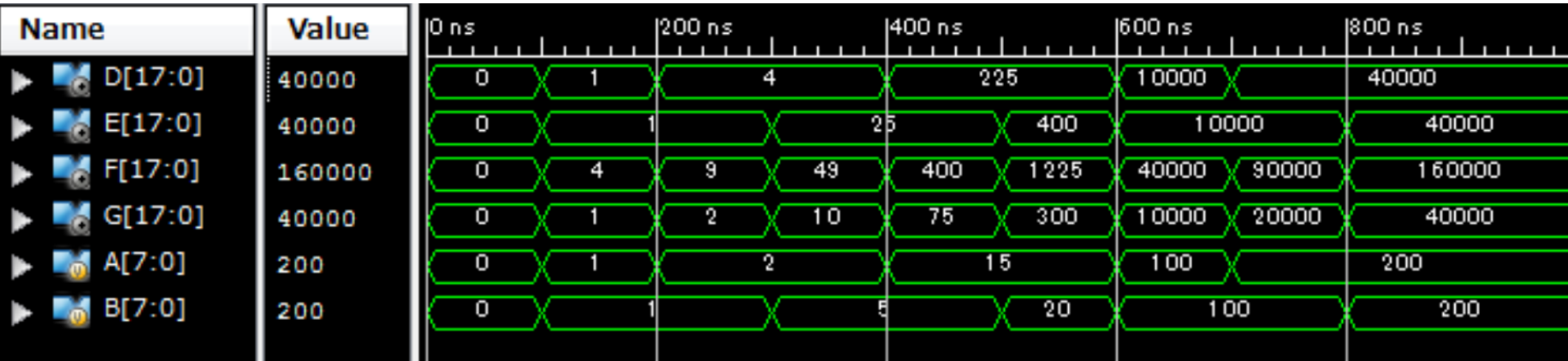
HDLシミュレーションによる検証

Verilog HDLを使用



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

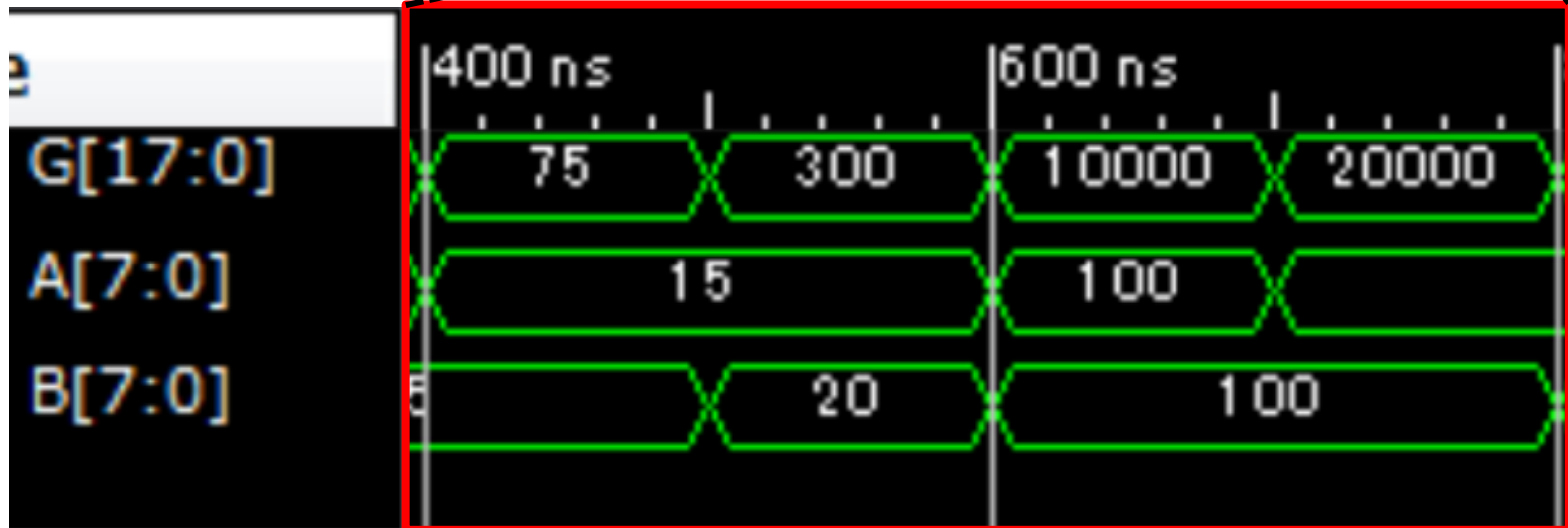
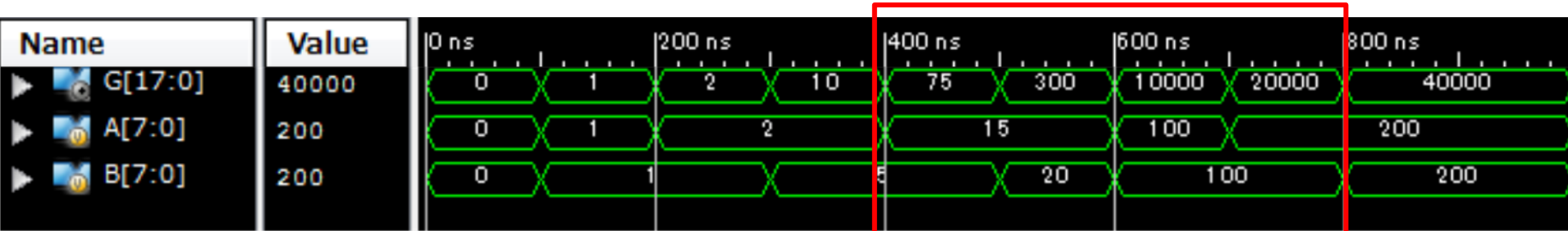
$$D(18\text{bit}) = A^2$$

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

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

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

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

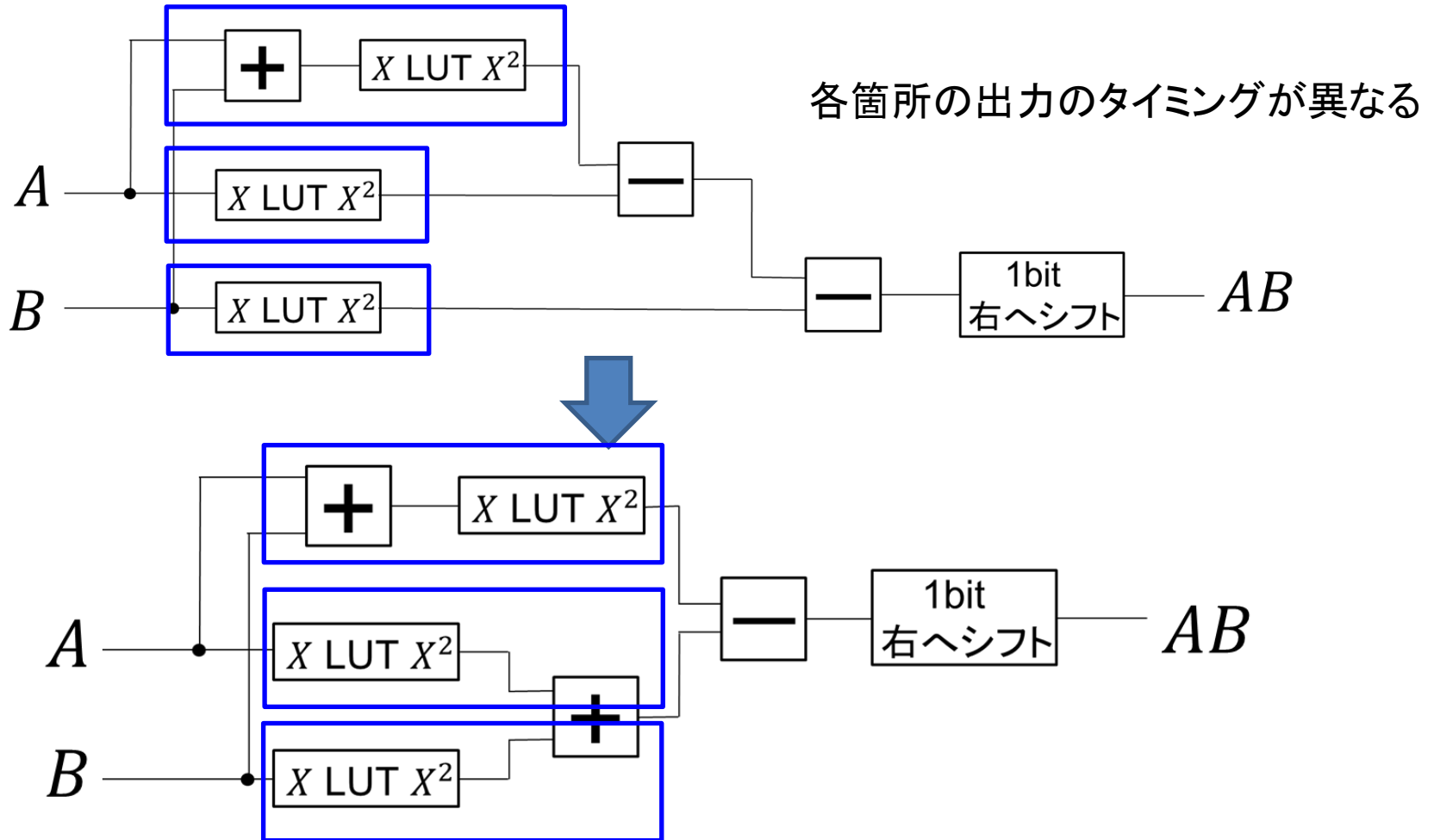
OUTLINE

- 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
- 今後の検討回路
- まとめ

今後の検討回路

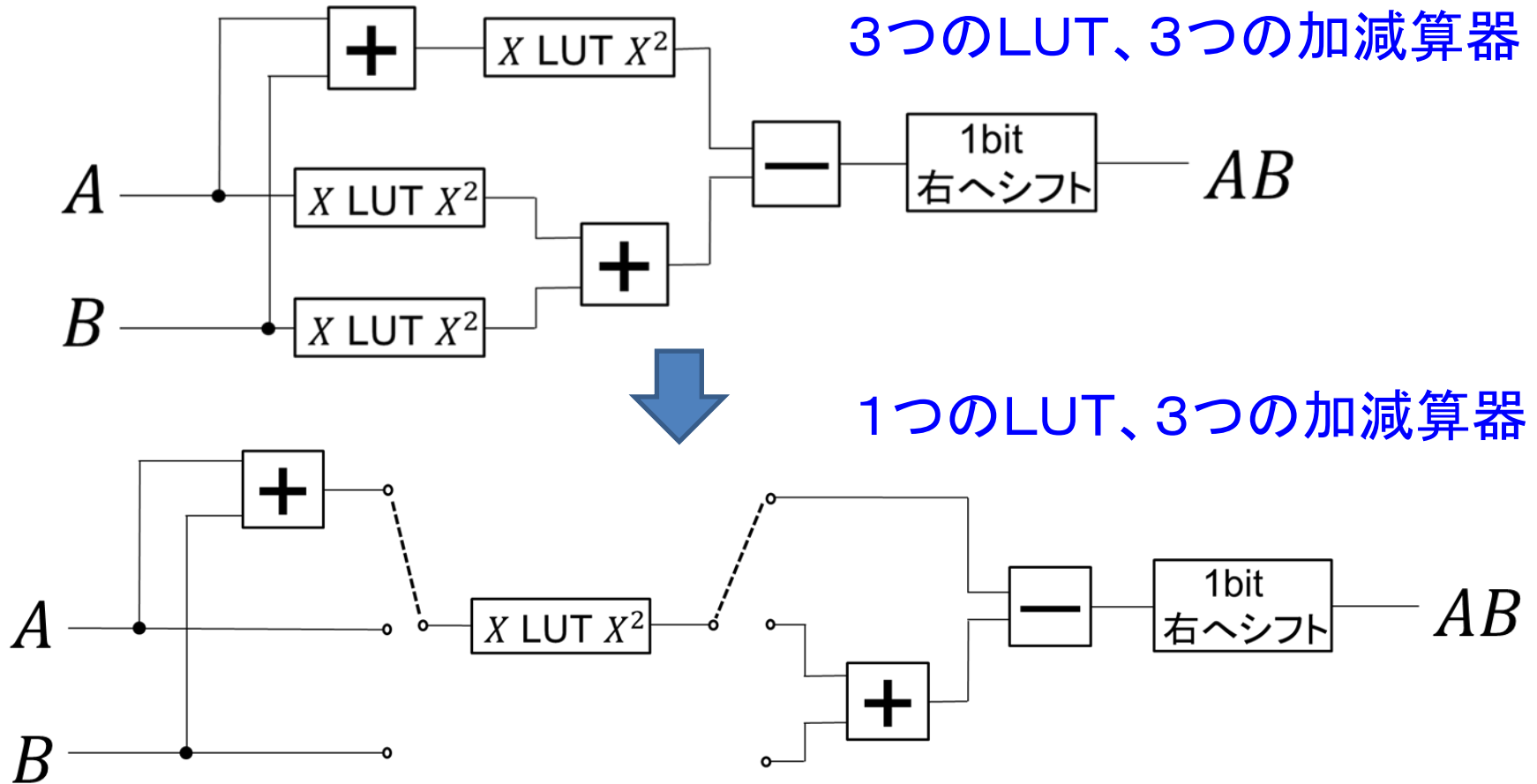
- 演算時間を考慮した回路設計
- 演算器を逐次的に使用した回路設計
- 別のアルゴリズムを用いた回路設計

演算時間を考慮した回路設計



どの経路も加算器とLUTを通るので出力タイミングが等しい

乗算器を逐次的に使用した回路設計

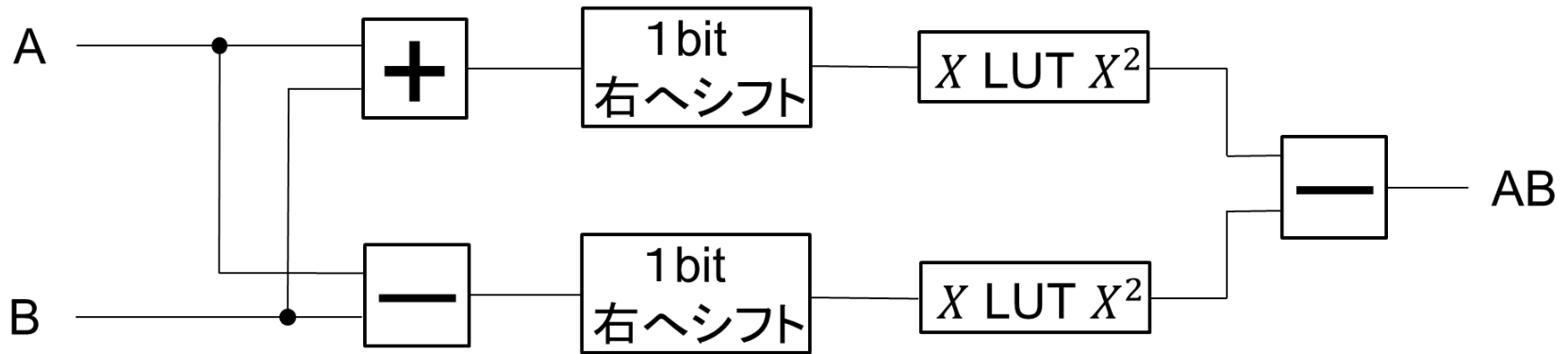


- 1つのLUTを $A^2, B^2, (A + B)^2$ の計算に逐次的に使用
全体として3分の1の回路量、計算時間は3倍

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
- 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

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

Q & A

Q. なぜ $AB = \frac{1}{4}\{(A + B)^2 - (A - B)^2\}$ の方は検証できていないのか？

A. $(A - B)$ の部分が負の数になると補数で数値が変化してしまい、正確な値が出力されないという問題がある

Q. FPGA内にDSPブロックという演算用の回路が存在するので、そこと検討回路を比較してどうなるか知りたい

A. これから検証して、速度や回路規模などを比較したいと考えています

Q & A

Q. FPGAには変数と定数を分けて計算する分割演算法というものが存在し、それと似ている。

それと何が違うのか？⇒この研究の新規性は何か？

A. 分割演算法が変数と定数を分けて計算するのに対して Divide&Conquerは変数と変数に分けて計算しているため、そこが違う⇒新規性に繋がる。

Q & A

Q. LUTを分割すると演算器が増えている。単純に分割するのではなく、増えていく演算器も考慮して実装をすべき闇雲に分割しても結局変わらない。

A. ありがとうございます。これから検討します。



