高性能計算基盤

第2回 CA0601:演算器

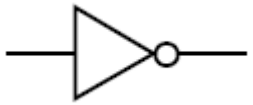
http://archlab.naist.jp/Lectures/ARCH/ca0601/ca0601j.pdf

ナレータ VOICEVOX:もち子(cv 明日葉よもぎ)

加減算回路

基本演算

表 1.2: NOT の真理値表



入力	NOT
0	1
1	0

表 1.3: AND の真理値表

(入力 1, 入力 2)	AND
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	1

表 1.4: OR の真理値表

(入力 1, 入力 2)	OR
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	1

表 1.5: NAND の真理値表

(入力 1, 入力 2)	NAND
(0, 0)	1
(0, 1)	1
(1, 0)	1
(1, 1)	0

表 1.6: NOR の真理値表

(入力 1, 入力 2)	NOR
(0, 0)	1
(0, 1)	0
(1, 0)	0
(1, 1)	0

表 1.7: XOR の真理値表

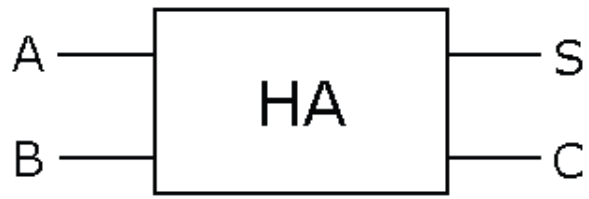
(入力 1, 入力 2)	XOR
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0

半加算器(1)

- 半加算器：1桁の2進数(A,B)の加算
  - S：2数の和
  - C：桁上げ

A		0		0		1	
B	+	0	+	1	+	0	
	0	0	0	1	0	1	
	C	S	C	S	C	S	

A		1					
B	+	1					
	1	0					
	C	S					



半加算器(Half Adder)の記号

半加算器(2)

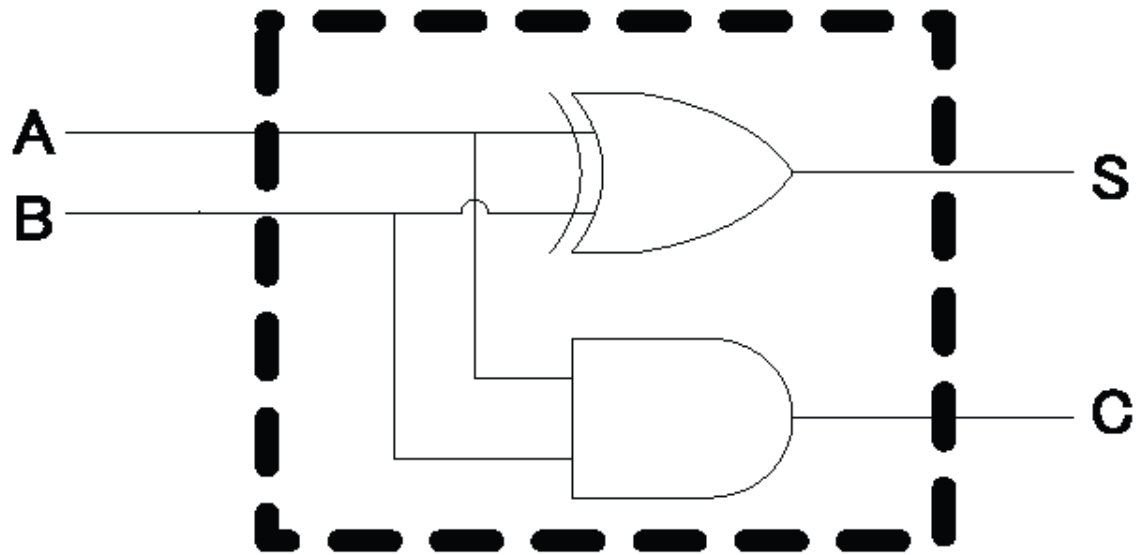
A		0		0		1	
B	+	0	+	1	+	0	+	1
	0	0	0	1	0	1	1	0
	C	S	C	S	C	S	C	S

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C = AB$$

半加算器(3)

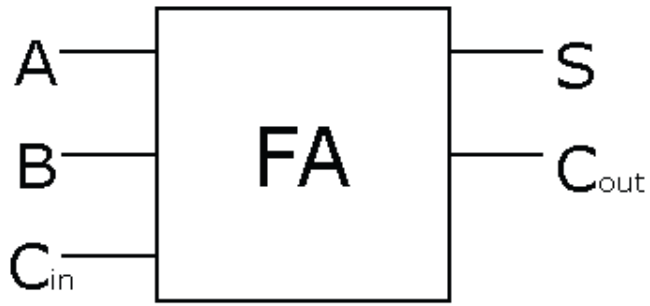


$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C = AB$$

全加算器(1)

- 全加算器：二つの2進数のある同じ桁の値(A,B)と下の桁からの桁上げ(C_{in})を入力として加算を行う
  - S：その桁の和の値
  - C_{out}：桁上げ



全加算器(Full Adder)の記号

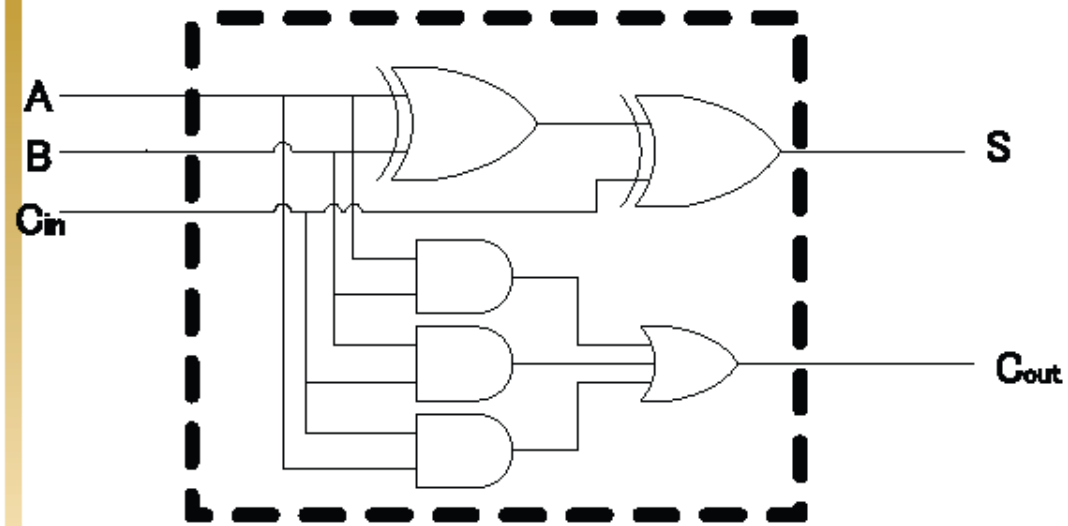
全加算器(2)

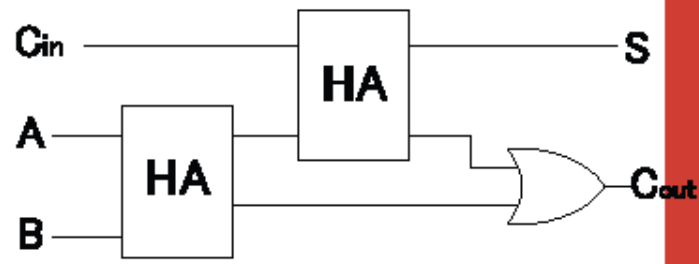
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

全加算器(3)



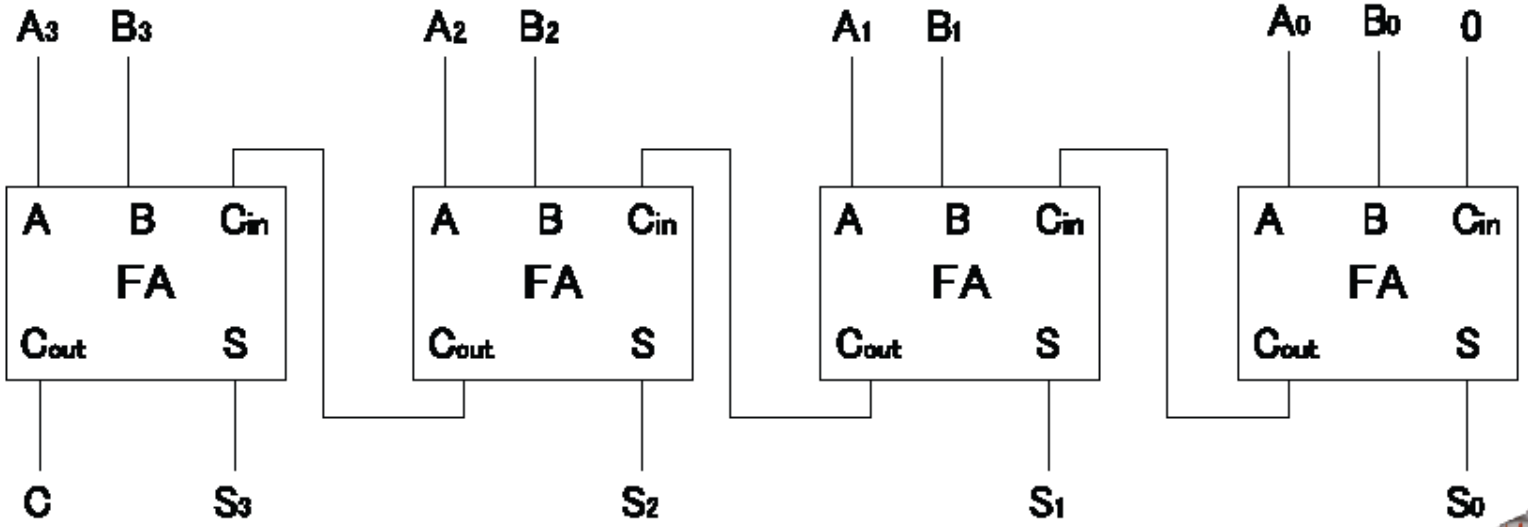


$$S = A \oplus B + C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

加算回路

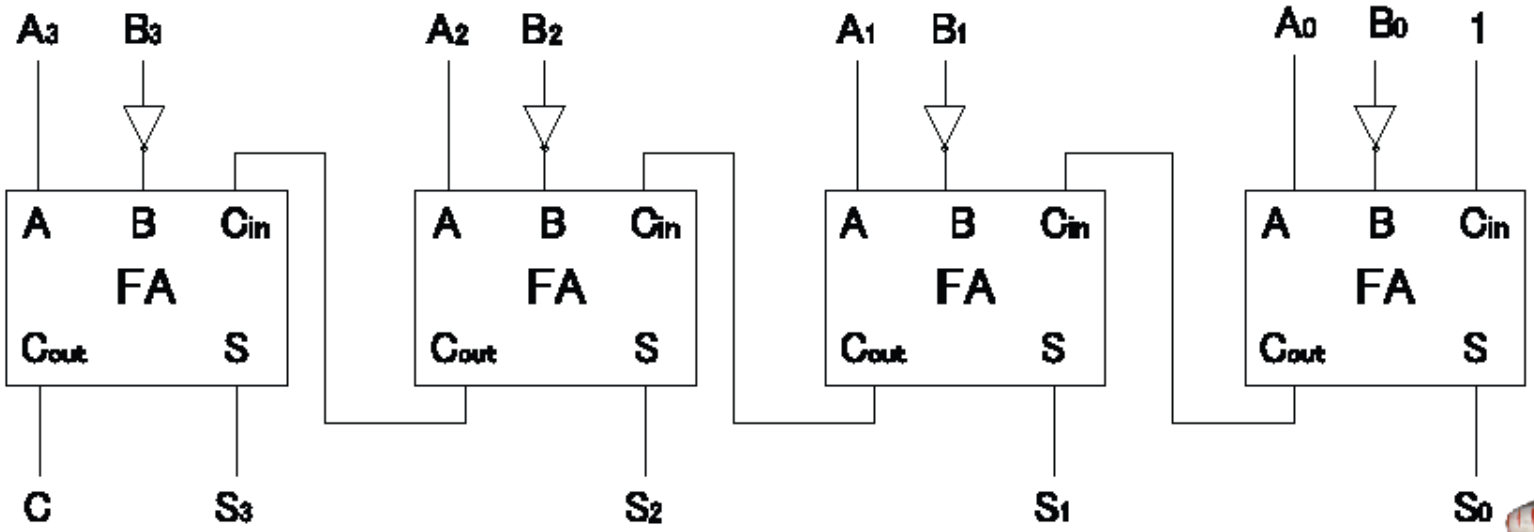
- 1ビット全加算器を縦列接続することで，nビット2進数の加算回路を構成できる



4桁の加算回路：
4桁の2進数（$A_3A_2A_1A_0$）と（$B_3B_2B_1B_0$）の和が（$S_3S_2S_1S_0$）となり，最上位桁の桁上がりがCとなる．

減算回路

- 加算回路を変形し2の補数を用いて減算回路が構成できる … $X\text{-}Y = X + (\overline{Y} + 1)$



4桁の減算回路

種々の論理回路
比較回路，一致回路

- 例題
  - 入力：2つの 2 bit の 2 進数（A，B）
  - 出力１：A＞Bのとき（比較回路）
  - 出力２：A＝Bのとき（一致回路）

実用的な演算器

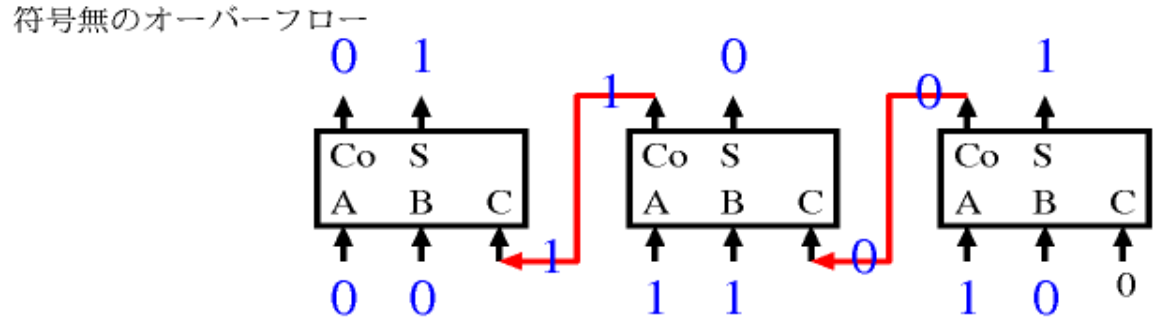
加算器の基本単位

全加算器

- ３つの１ビット信号（A,B,C）を加算し,２ビット（S,Co）を出力

```
A B C   Co S
0 0 0   0  0 (0)
0 0 1   0  1 (1)
0 1 0   0  1 (1)    Co = (B & C) | (A & C) | (A & B)
1 0 0   0  1 (1)    S  =  A ^ B ^ C
0 1 1   1  0 (2)
1 0 1   1  0 (2)
1 1 0   1  0 (2)
1 1 1   1  1 (3)
```

- 下位のCo出力を上位のCに入力すると,任意ビット長の加算器ができる
  ただし,64段の64ビット加算器は遅過ぎる（logN段が一般的）



0 1 1 + 0 1 0 = 1 0 1

$3_{(10)}$ + $2_{(10)}$ = $5_{(10)}$

どうすれば段数を減らせるか？

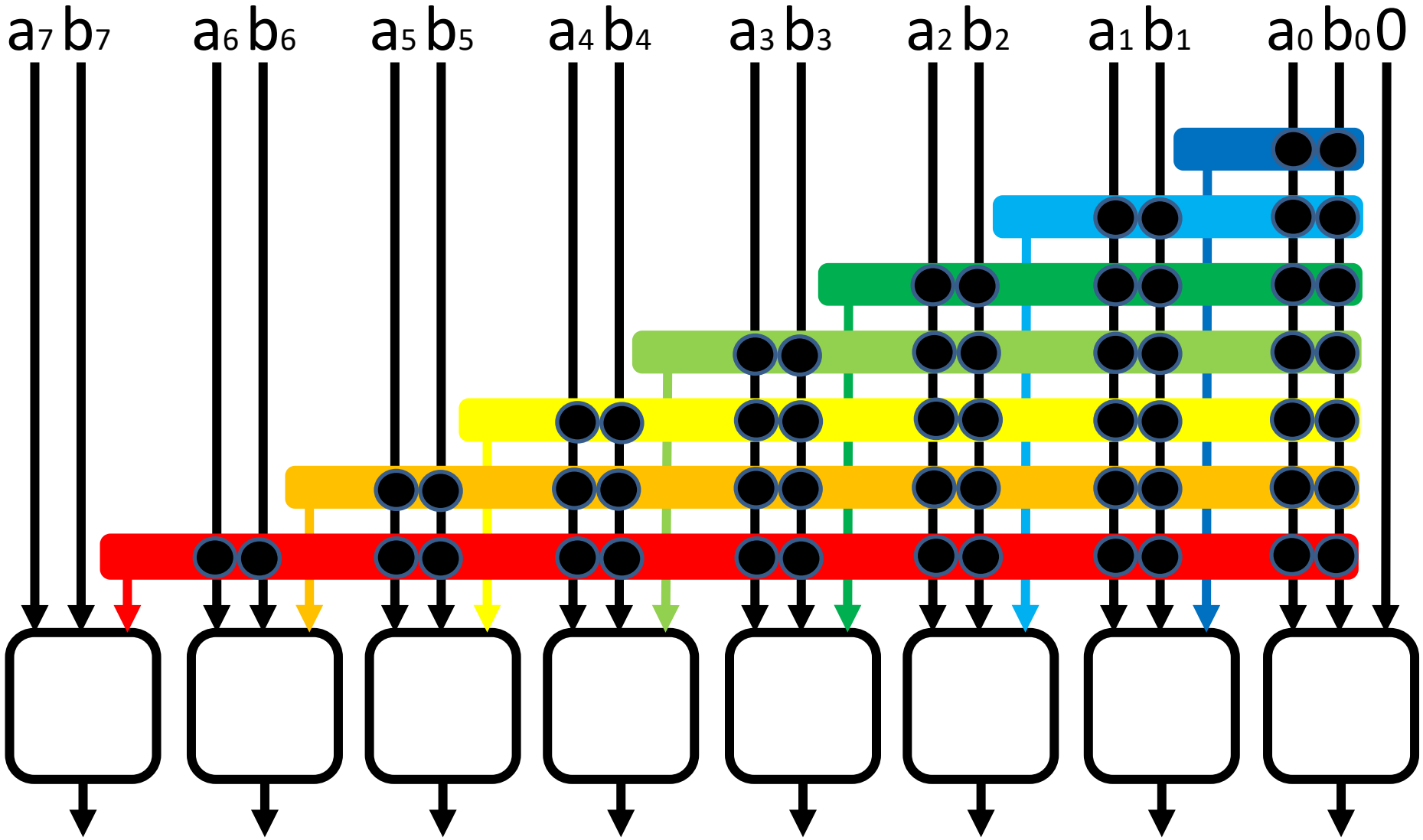
コンピュータシステムのあらゆる階層に
応用されている基本的な考え方

まず自分で考える！

思いついた人は素質有り

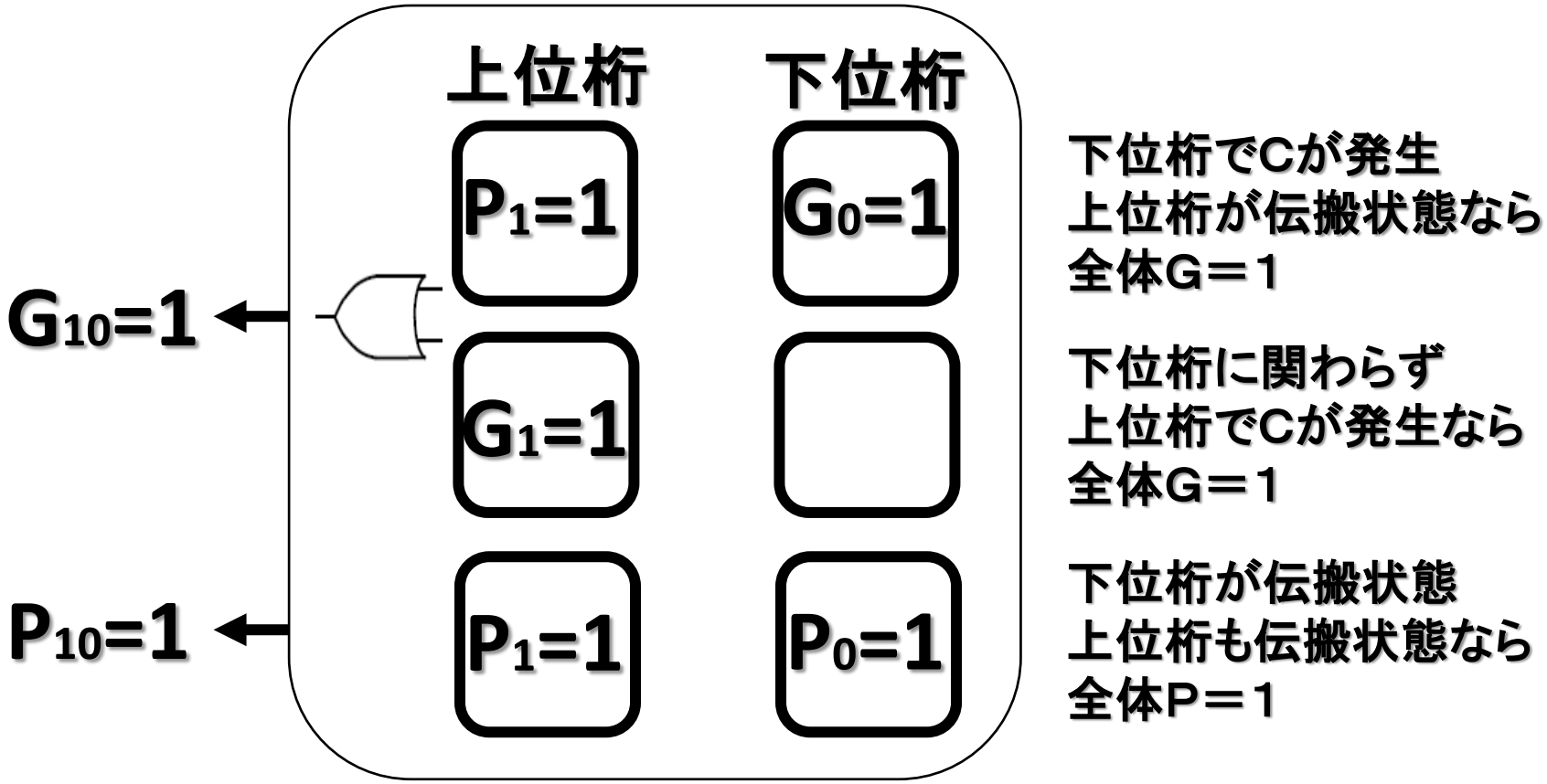
どうすれば段数を減らせるか？

理論上は、より下位ビットを全て使って計算可能だが



どうすれば段数を減らせるか？

キャリー伝搬信号（Propagate）と
キャリー生成信号(Generate）を束ねていく



どうすれば段数を減らせるか？

N段をLogN段に高速化する手法は様々な場所で使われる

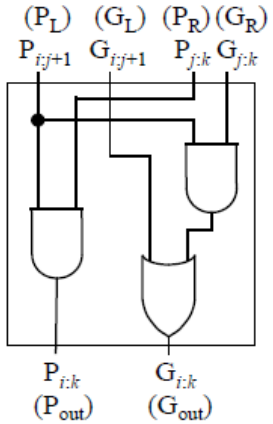
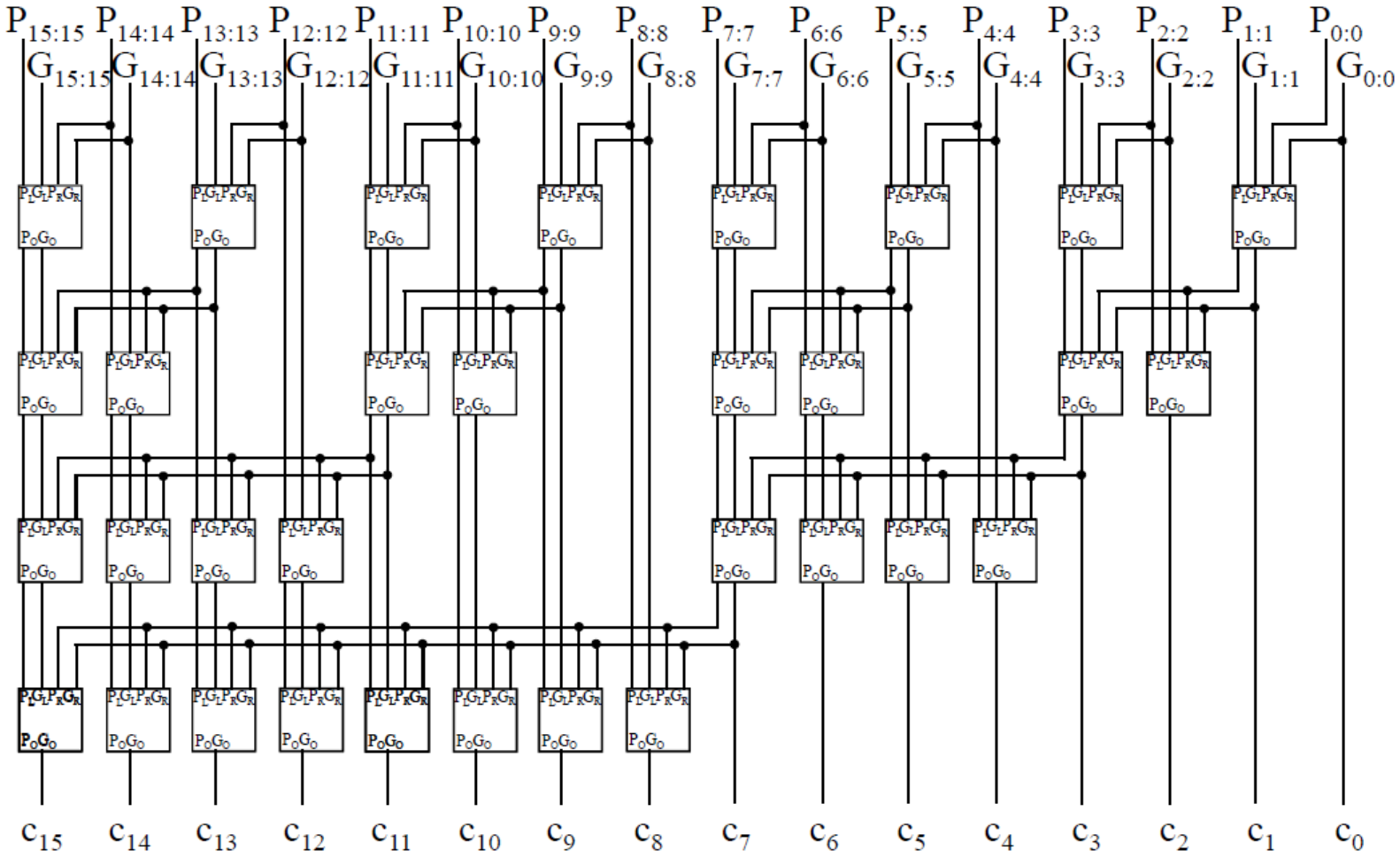


図 2.1: プレフィックス演算を行う回路



シフト回路

01000000（　64）
左1ビットシフト← 10000000（128）＊2と同じ
右1ビットシフト→ 01000000（　64）　/2と同じ
符号付なら　　10000000（-128）
右1ビットシフト→ 11000000（　-64）符号を複写

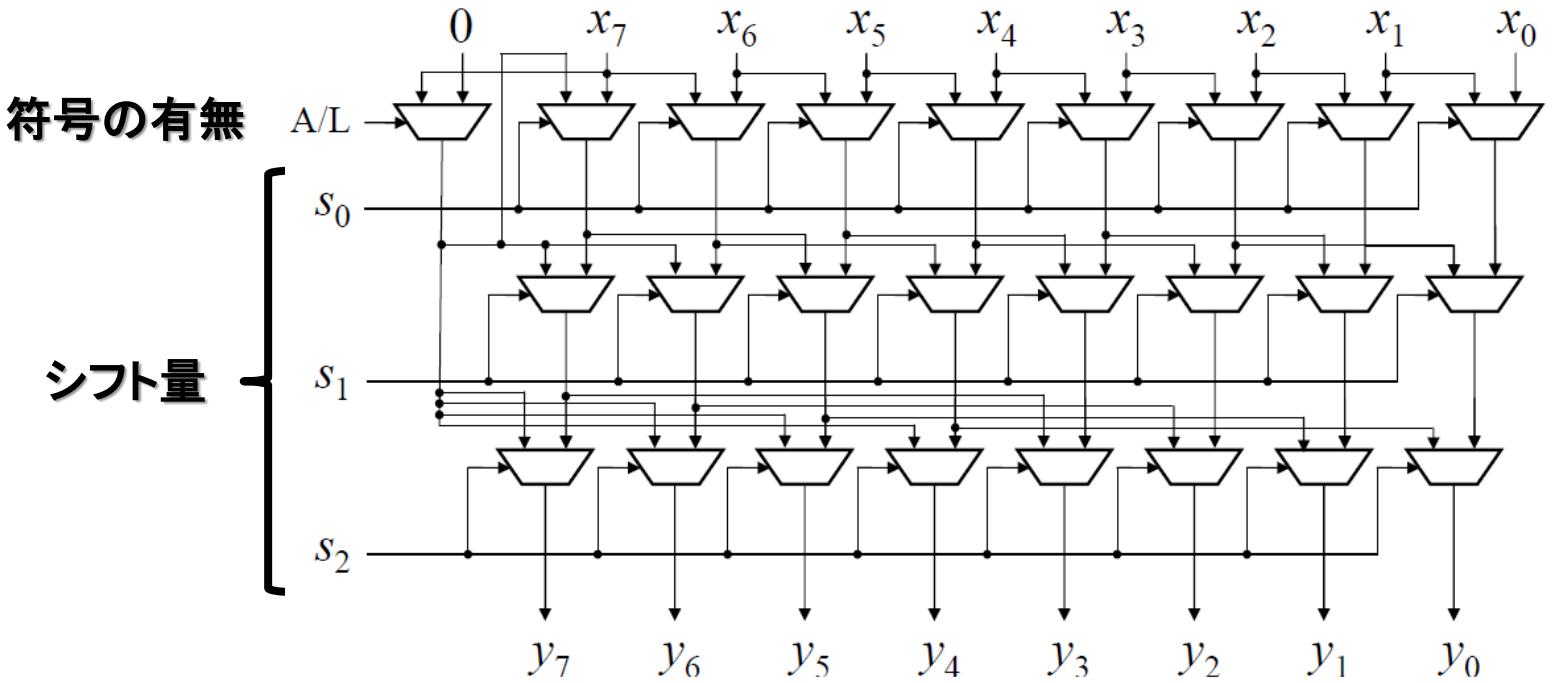
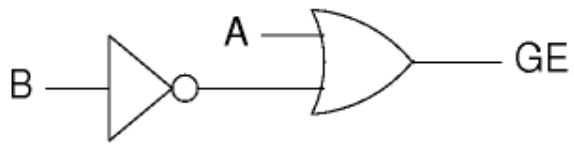


図 2.3: 8 ビットの右シフト回路

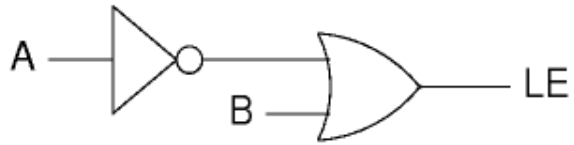
別の方法による加算器の構成

まずはAとB各1ビットの大小比較

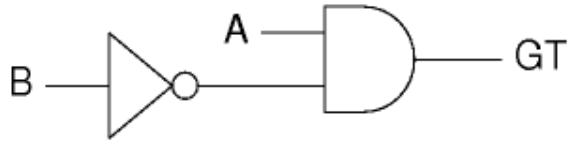
- A≧Bの場合のみ，GEが1
  AB=00:GE=1,AB=01:GE=0
  AB=10:GE=1,AB=11:GE=1



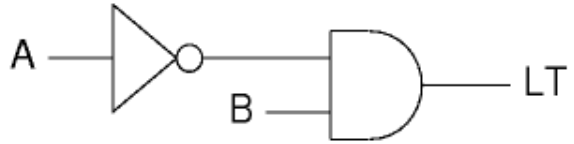
- A≦Bの場合のみ，LEが1
  AB=00:LE=1,AB=01:LE=1
  AB=10:LE=0,AB=11:LE=1



- A＞Bの場合のみ，GTが1
  AB=00:GT=0,AB=01:GT=0
  AB=10:GT=1,AB=11:GT=0



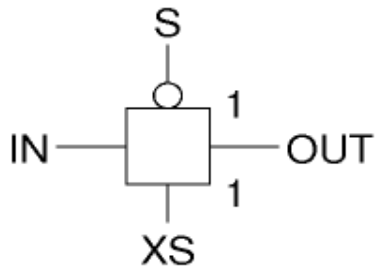
- A＜Bの場合のみ，LTが1
  AB=00:LT=0,AB=01:LT=1
  AB=10:LT=0,AB=11:LT=0



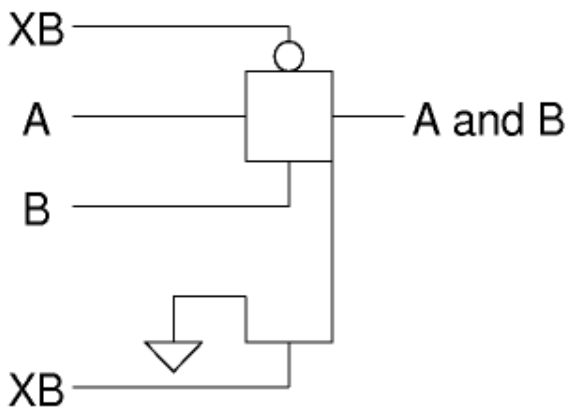
伝送ゲート（トランスミッションゲート）

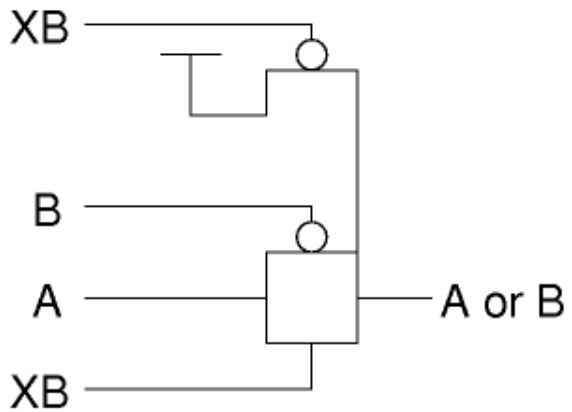
PCH/NCHの別の使い方

- Ｓが０，ＸＳが１の時，INがOUTに伝搬
- Ｓが１，ＸＳが０の時，INとOUTは遮断



- NAND/NORではなく，AND/ORが作れる



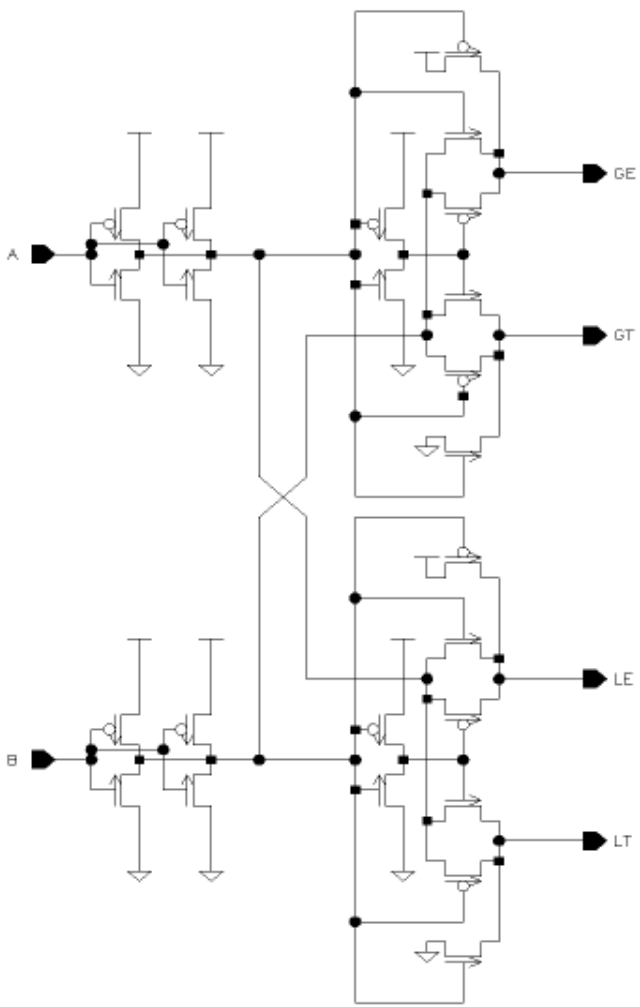


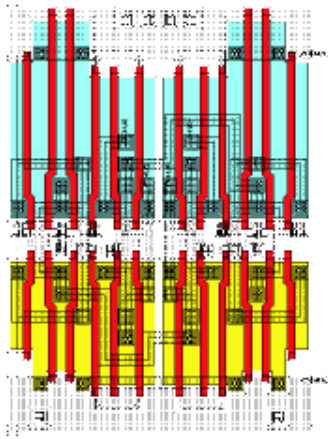
伝送ゲート（トランスミッションゲート）

前述のGE,GT,LE,LTを一度に求める大小比較回路 … Igen

- 伝送ゲートの駆動力は弱いので前段にインバータを配置

  前述の回路とは論理が異なるが考え方は同じ





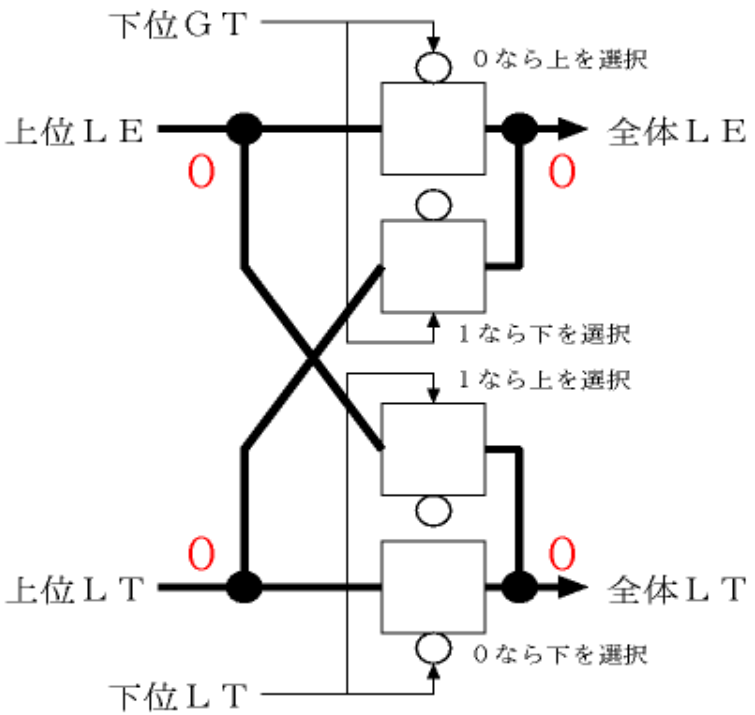
NAIST

伝送ゲート（トランスミッションゲート）

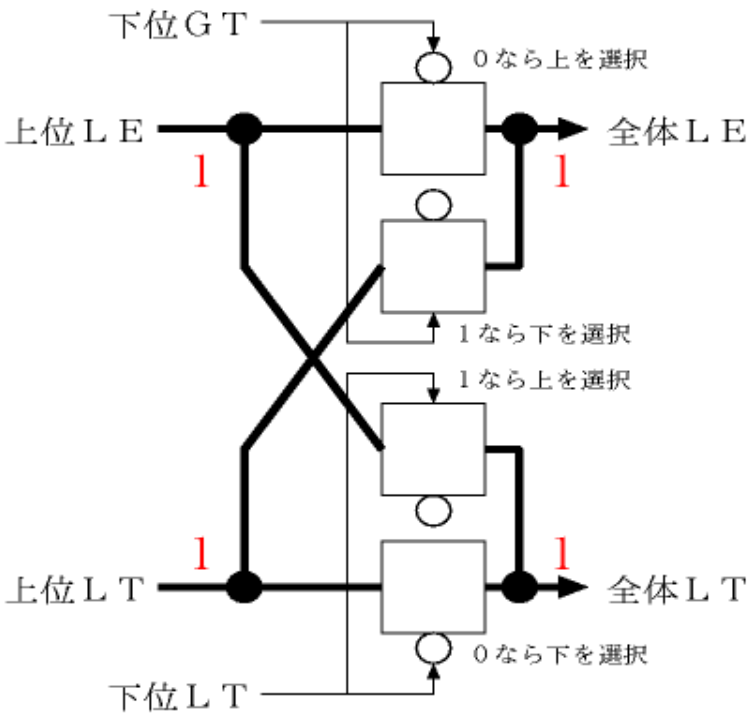
複数ビットの大小比較

- AとB各2ビットの大小比較では，上位ビットの関係が優先

- 上位ビットがA＞Bの場合
  無条件にA＞B



- 上位ビットがA＜Bの場合
  無条件にA＜B



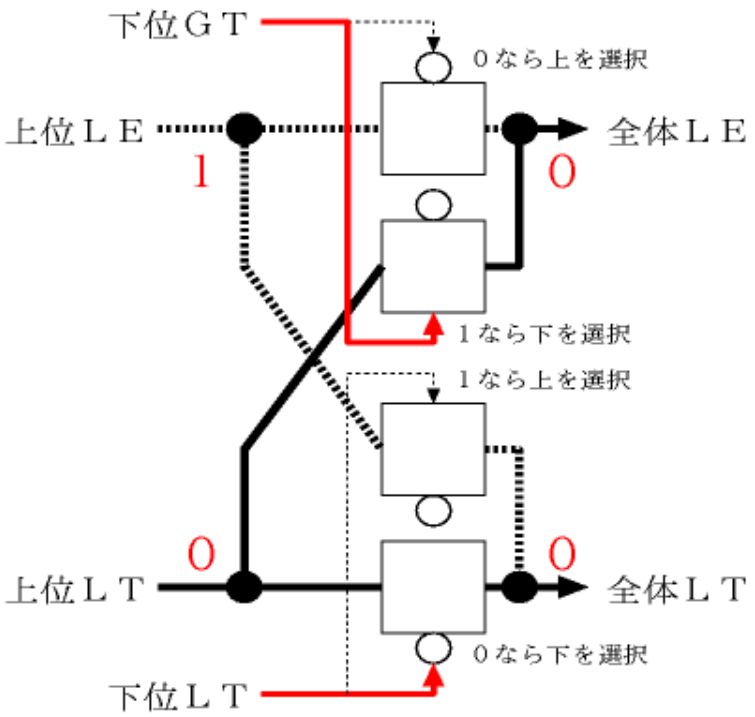
NAIST

伝送ゲート（トランスミッションゲート）

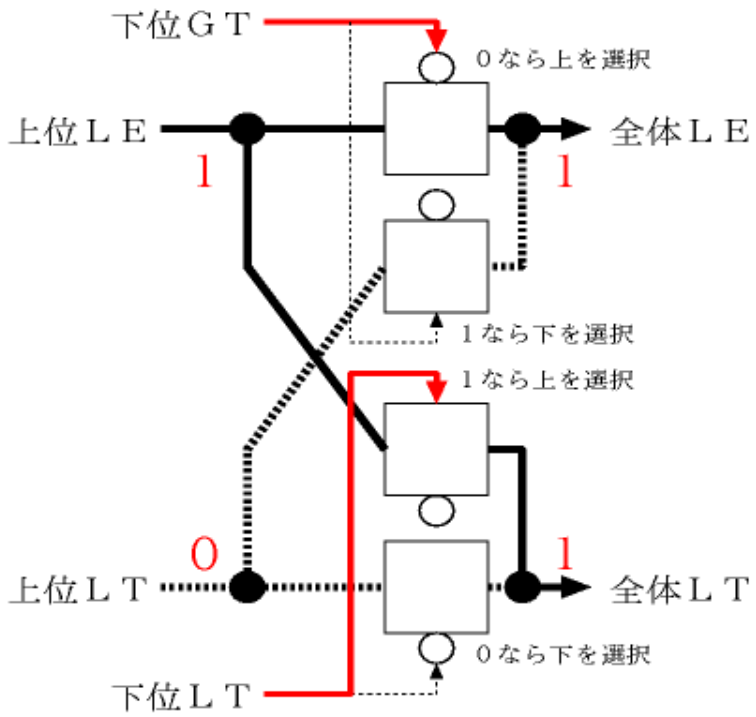
複数ビットの大小比較

- 上位ビットが同じ場合，下位ビットにより判定

- 下位ビットがA＞Bの場合
  全体もA＞B



- 下位ビットがA＜Bの場合
  全体もA＜B

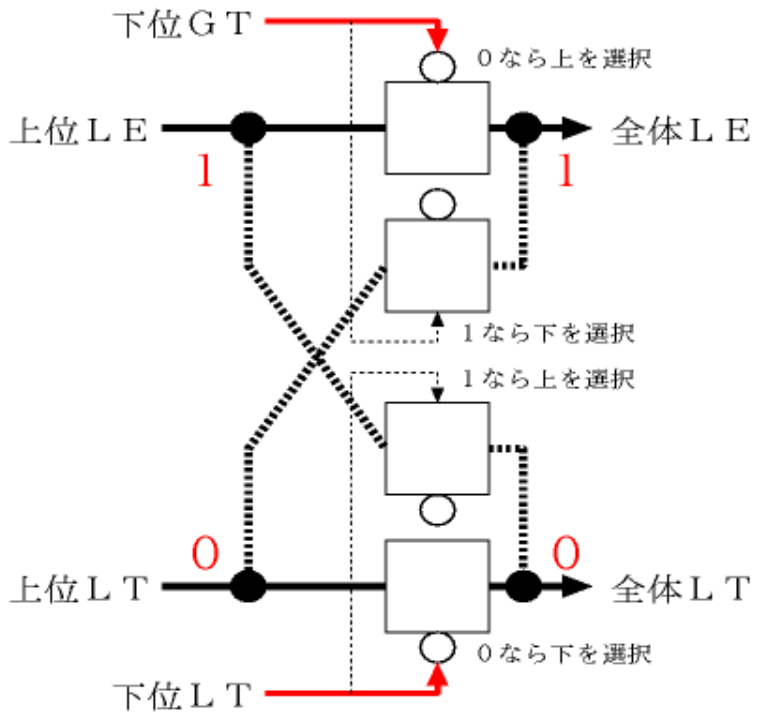


NAIST

伝送ゲート（トランスミッションゲート）

複数ビットの大小比較

- 下位ビットも同じ場合，全体としても同じ

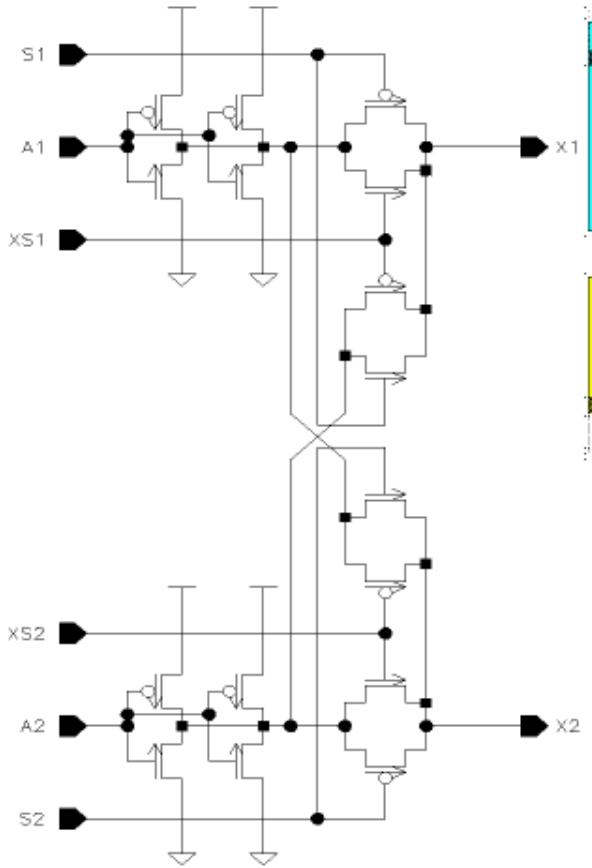


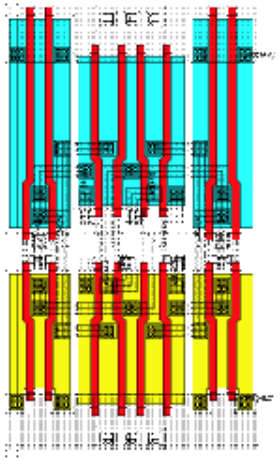
伝送ゲート（トランスミッションゲート）

前述の選択回路 … Isel2

- S1とXS1により，X1に至る入力を選択
- S2とXS2により，X2に至る入力を選択
- 同じく，伝送ゲートの駆動力は弱いので前段にインバータを配置

GTとLE，GEとLTに相補関係があるので，インバータの有無は配線の繋ぎ替えでＯＫ



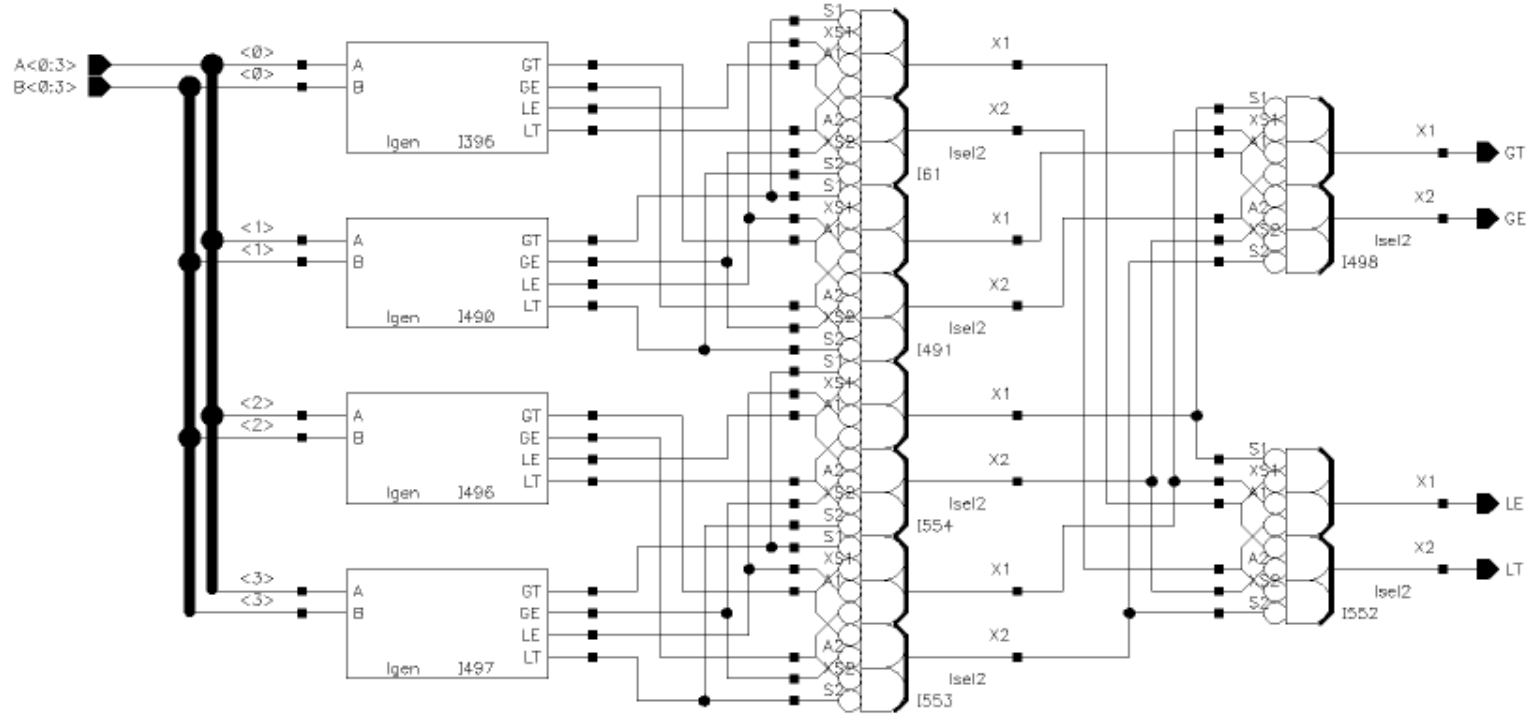


NAIST

大小比較器

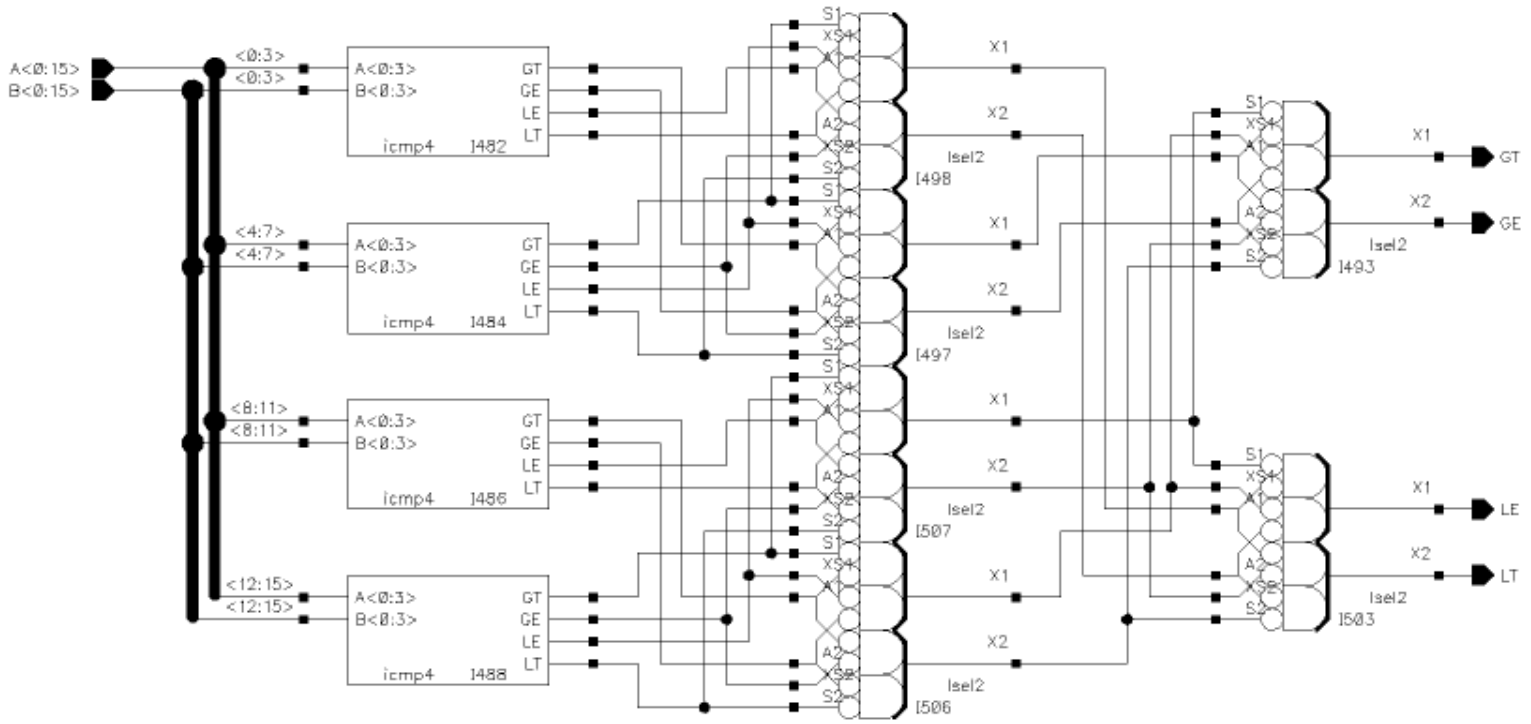
以上のリーフセルを用いた高速大小比較器

- 4個のIgenと，4個+2個のIsel2による，4bit大小比較器（icmp4）
- 1ビットの大小関係（GT,GE,LE,LT）2組から2ビットの関係を生成
- 2ビットの大小関係（GT,GE,LE,LT）2組から4ビットの関係を生成
  つまり，logN段で大小関係がわかる



大小比較器

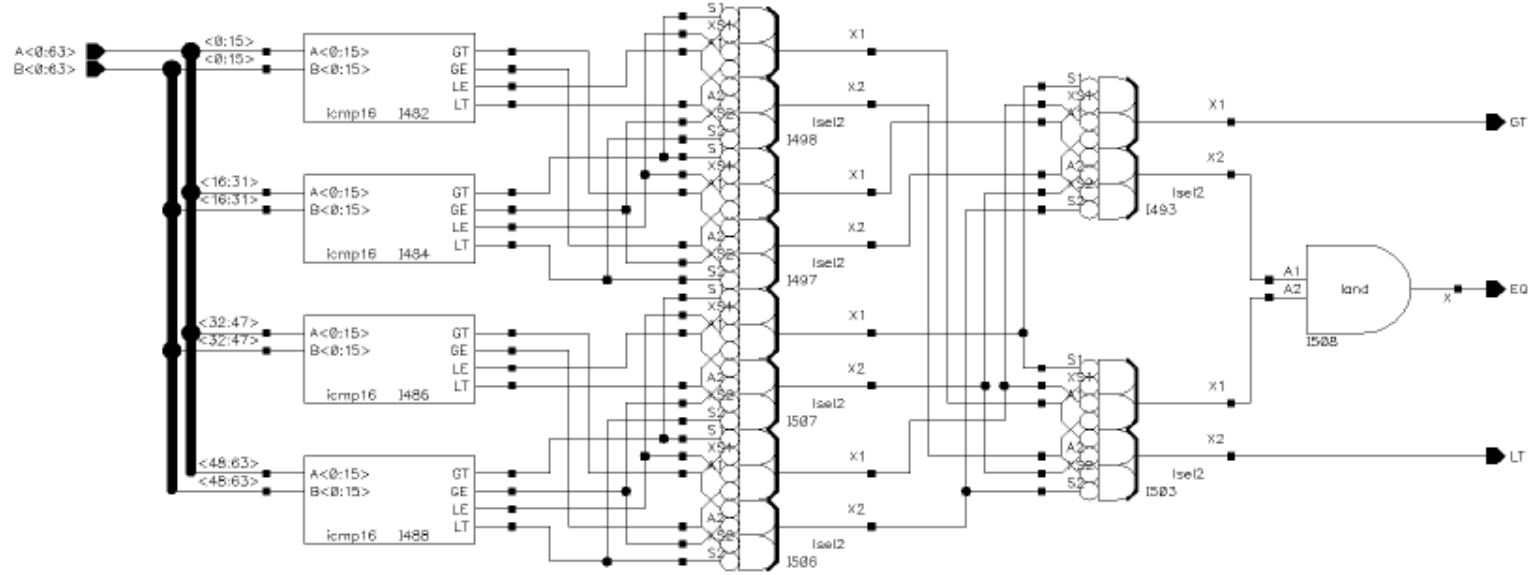
- 4個のicmp4と，4個+2個のIsel2による，16bit大小比較器（icmp16）

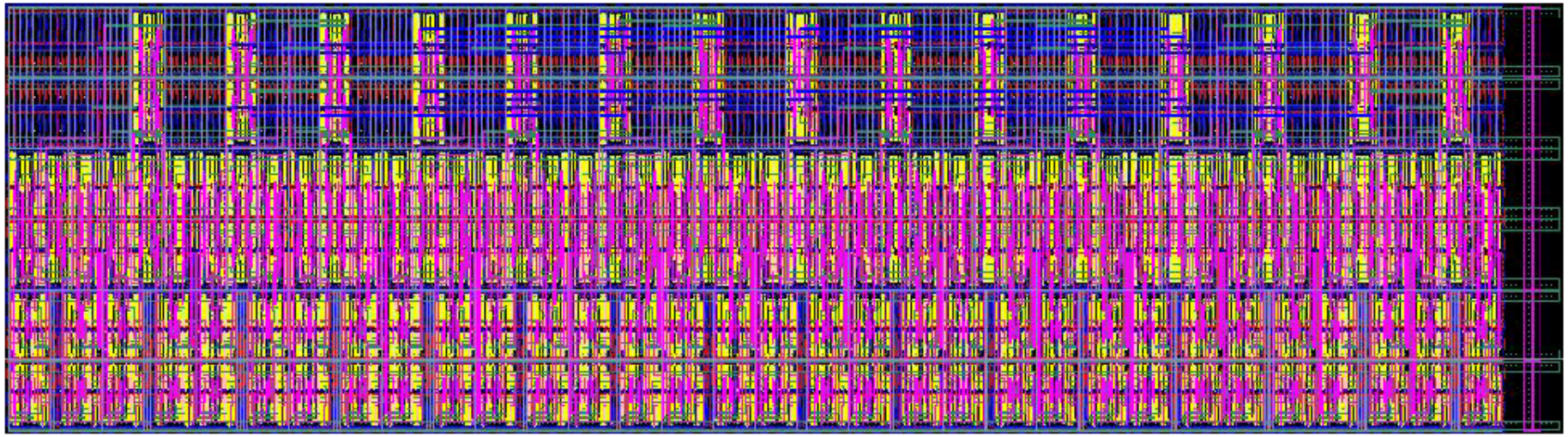


NAIST

大小比較器

- 4個のicmp16と，4個+2個のlsel2による，64bit大小比較器（icmp64）





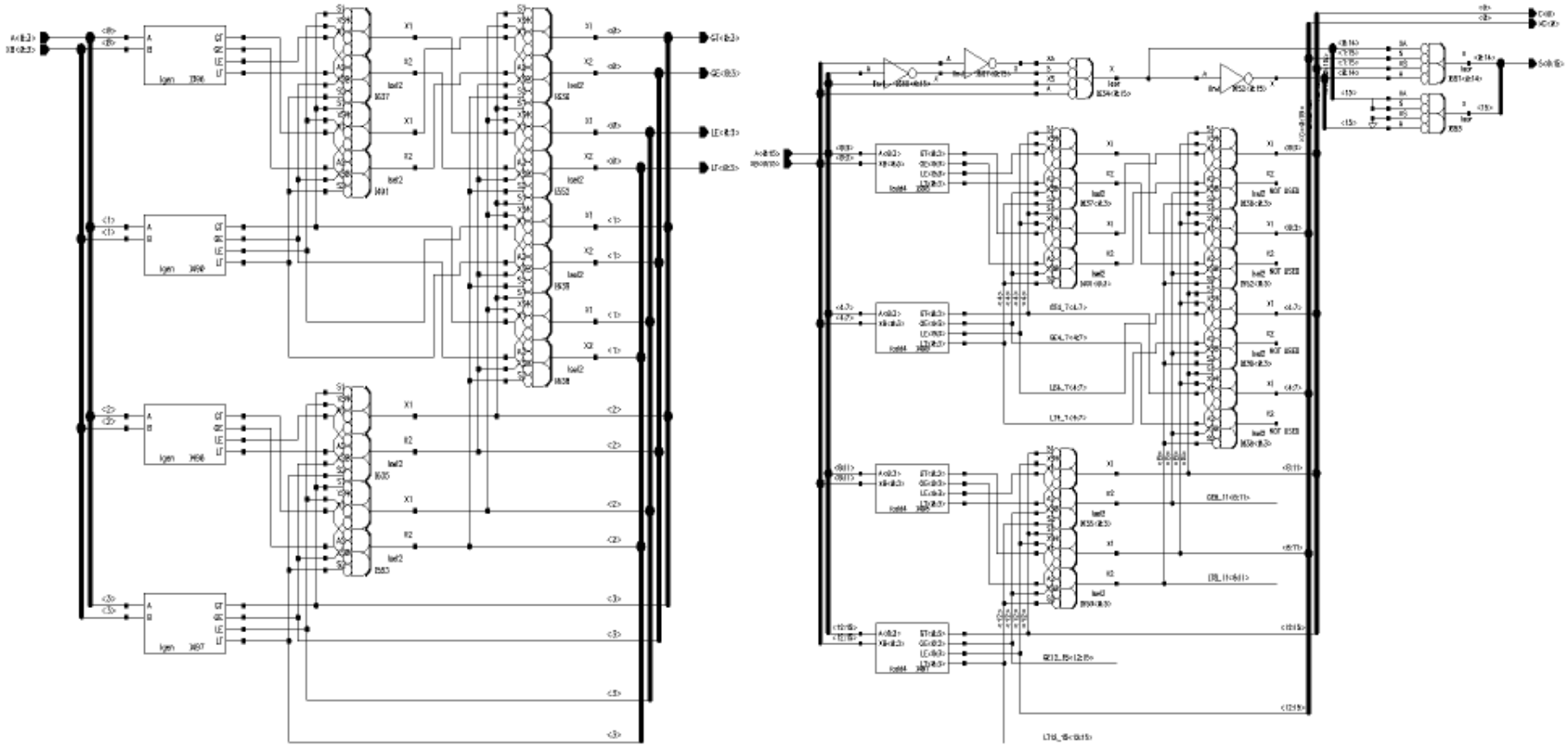
NAIST

加減算器

大小比較器は，実は減算器に必要なキャリー生成回路と同じ
Bを**2**の補数とすれば加算器に使える

- 4bitキャリー生成回路（iadd4）と，16bit加算器（iadd16）

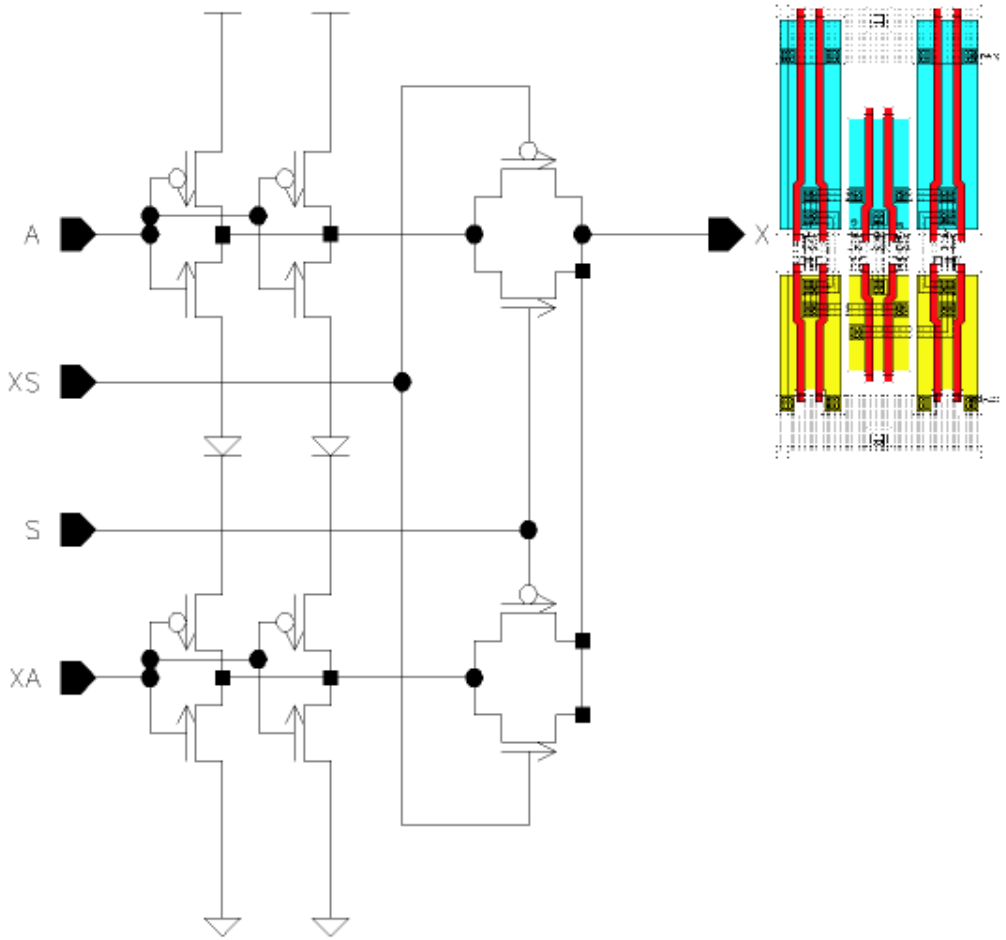
  加算器の最終段は，各ビットの部分和とキャリーの排他論理和（前述の全加算器と同じ機能）



排他論理和

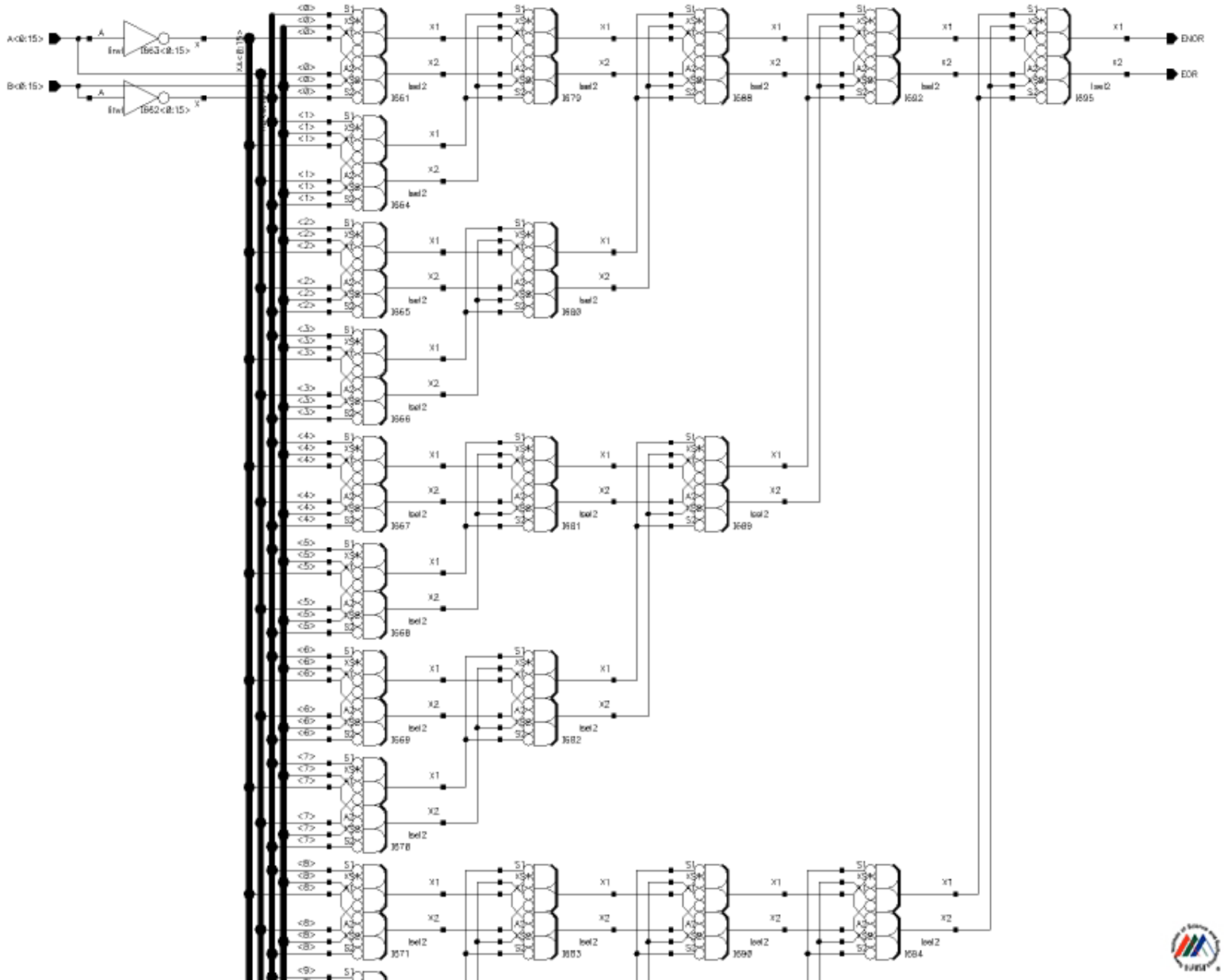
加算器の最終段に使用する排他論理和 … leor

- AとSの排他論理和を出力



排他論理和

- 単独で組み合わせると，多入力排他論理和（1が偶数個なら0を出力）

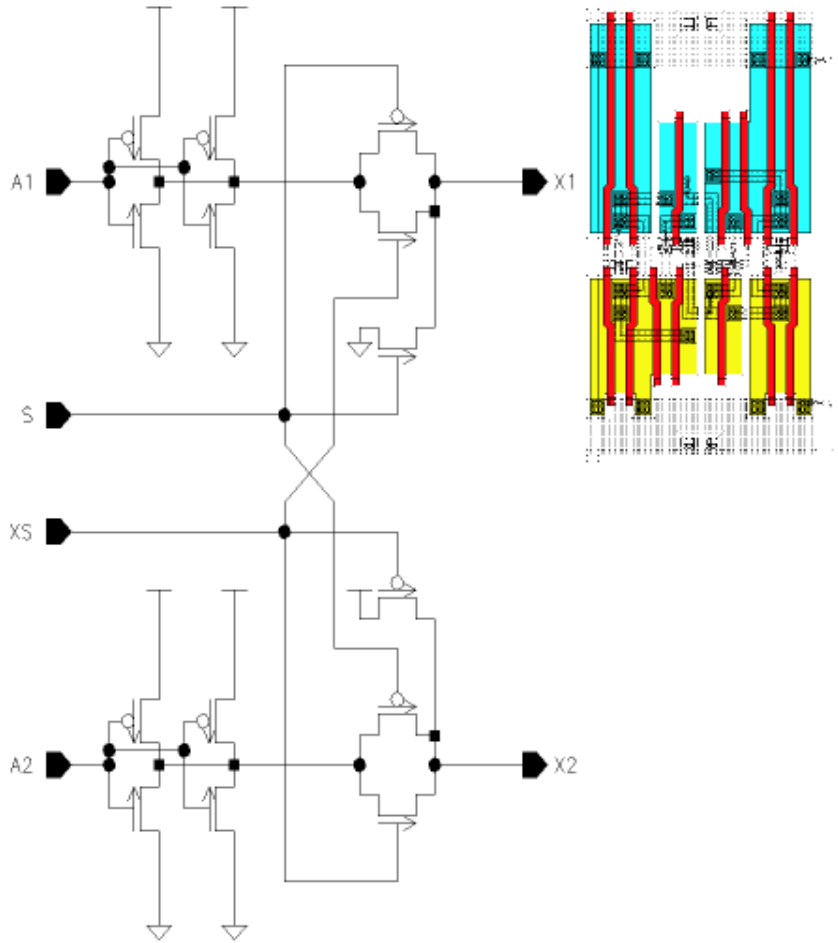


NAIST

伝送ゲート（トランスミッションゲート）

XBを11..1に固定すると，加減算器の各リーフセルは次の部品に簡単化

- NORとNANDを同時に求める回路 … Inorand
  - A1とSのNORをX1に出力
  - A2とXSのNANDをX2に出力

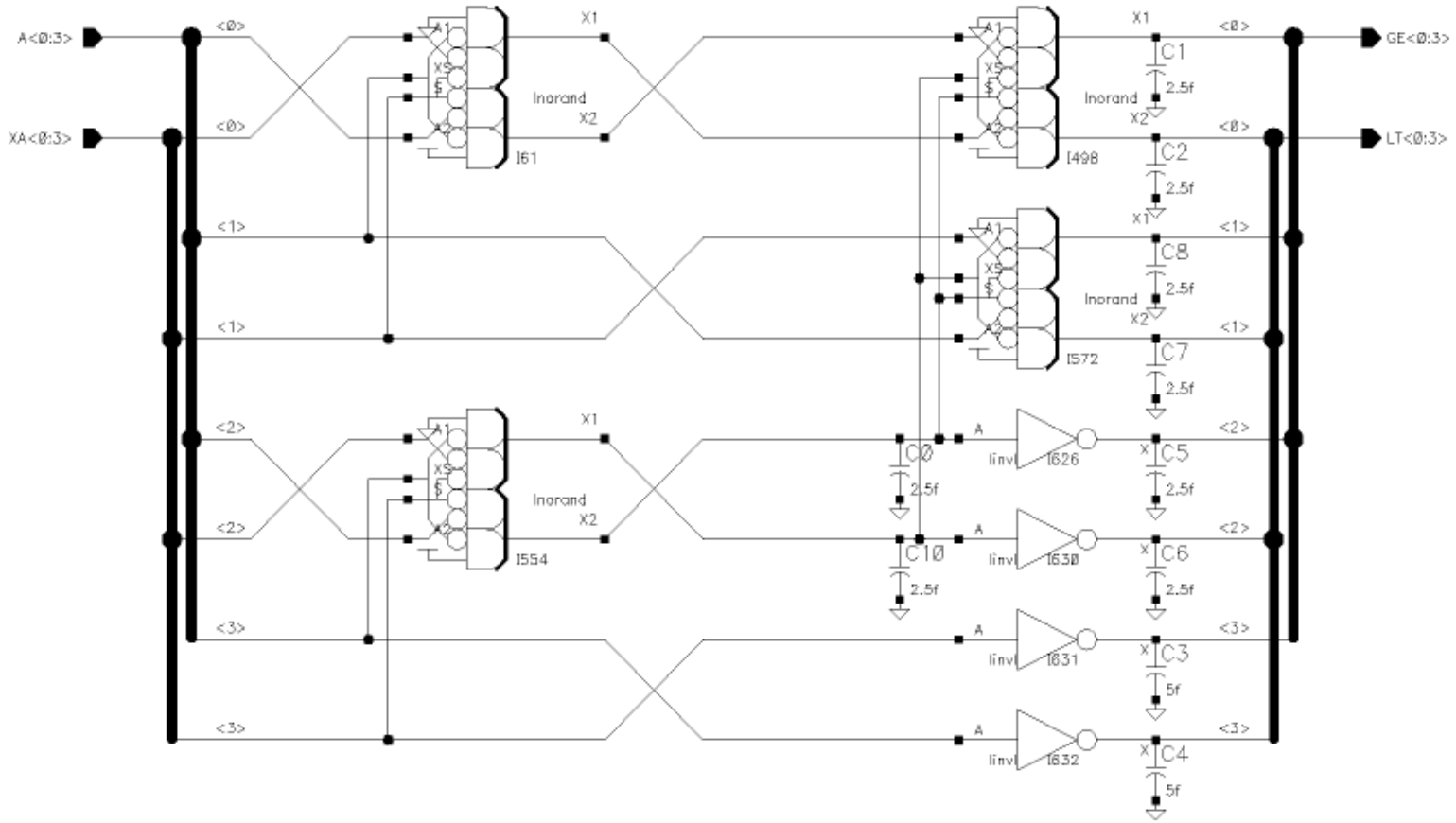


NAIST

インクリメンタ

XB（11..1）の加算は，B（00..1）の加算すなわちインクリメントに等しい

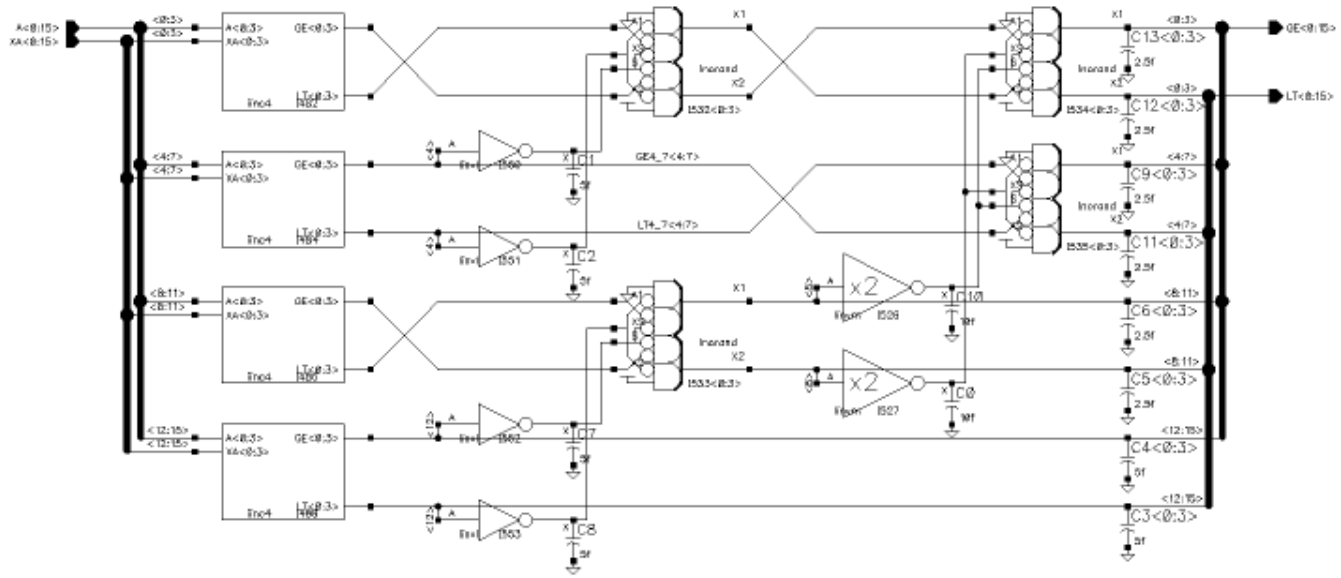
- インクリメンタ用4bitキャリー生成回路（iinc4）
- GTとLEは不要になる（各自確かめてみること）
- 結果的に，多入力ANDと反転論理（多入力OR）をlogN段で実現



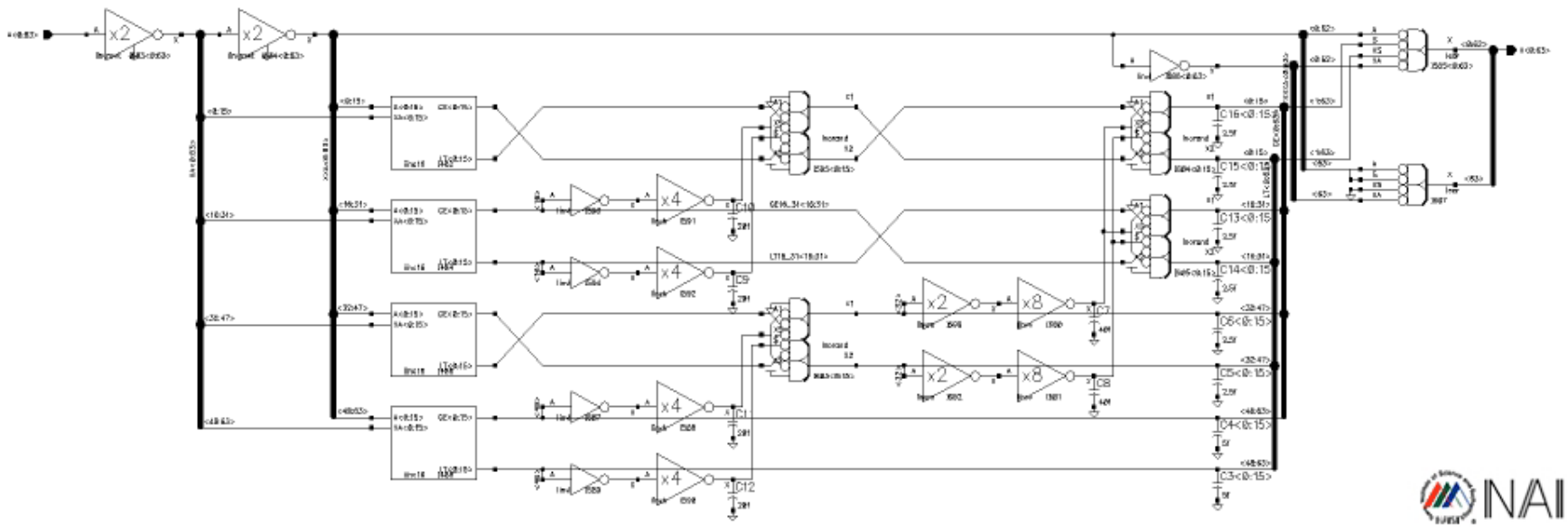
NAIST

インクリメンタ

- インクリメンタ用16bitキャリー生成回路（iinc16）



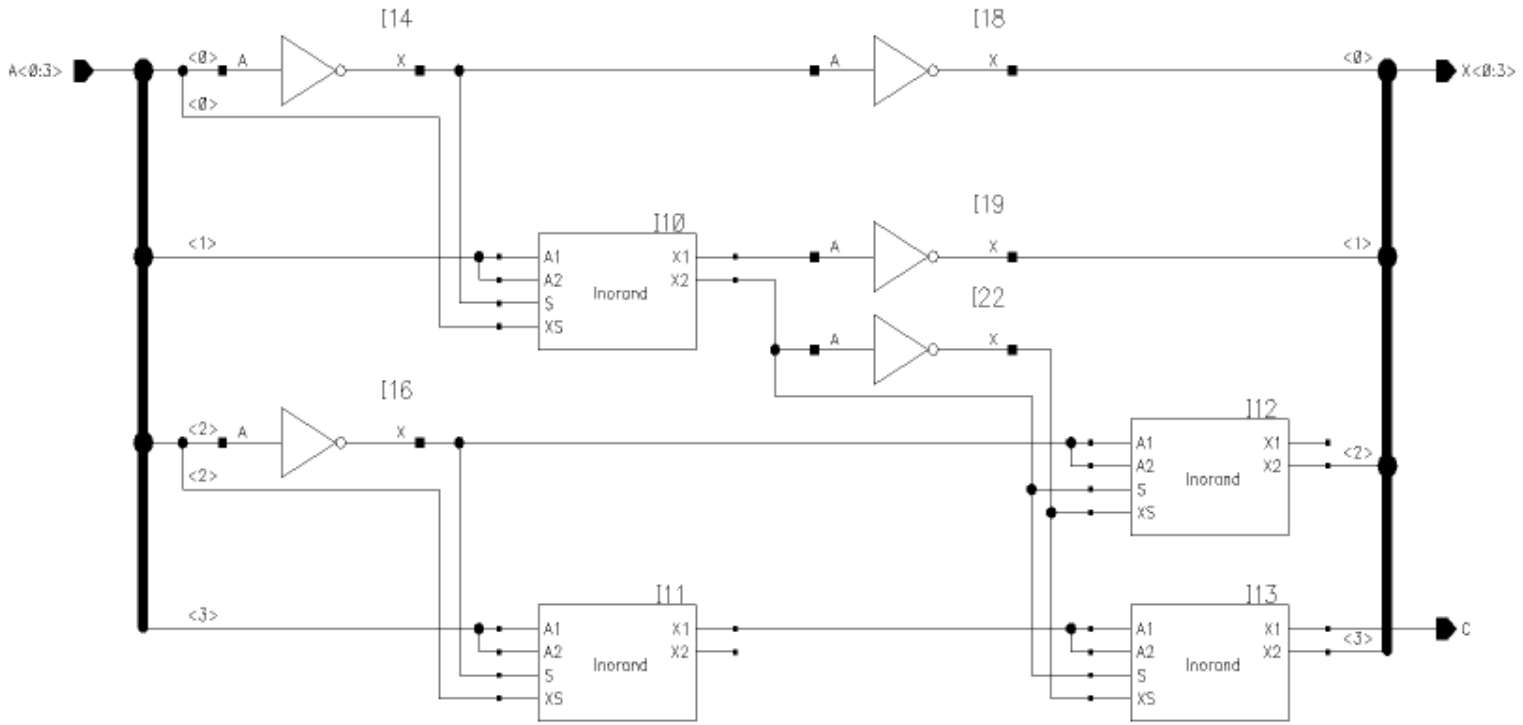
- 最終段に排他論理和を追加すると64bitインクリメンタ（iinc64）



プライオリティエンコーダ（エンコード手前まで）

さらに，Inorandから，プライオリティエンコーダが作れる

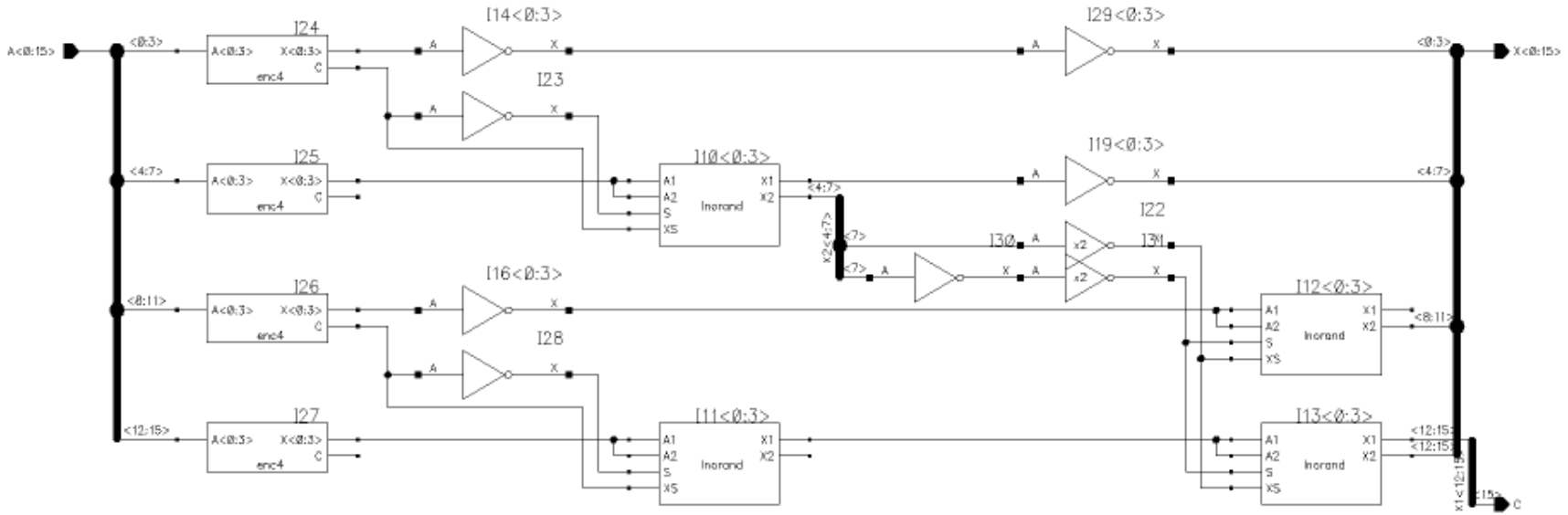
- インクリメンタは，言い替えると，下位の連続1を検出している
- 上位の連続0を検出して，先頭1を検出するプライオリティエンコーダ



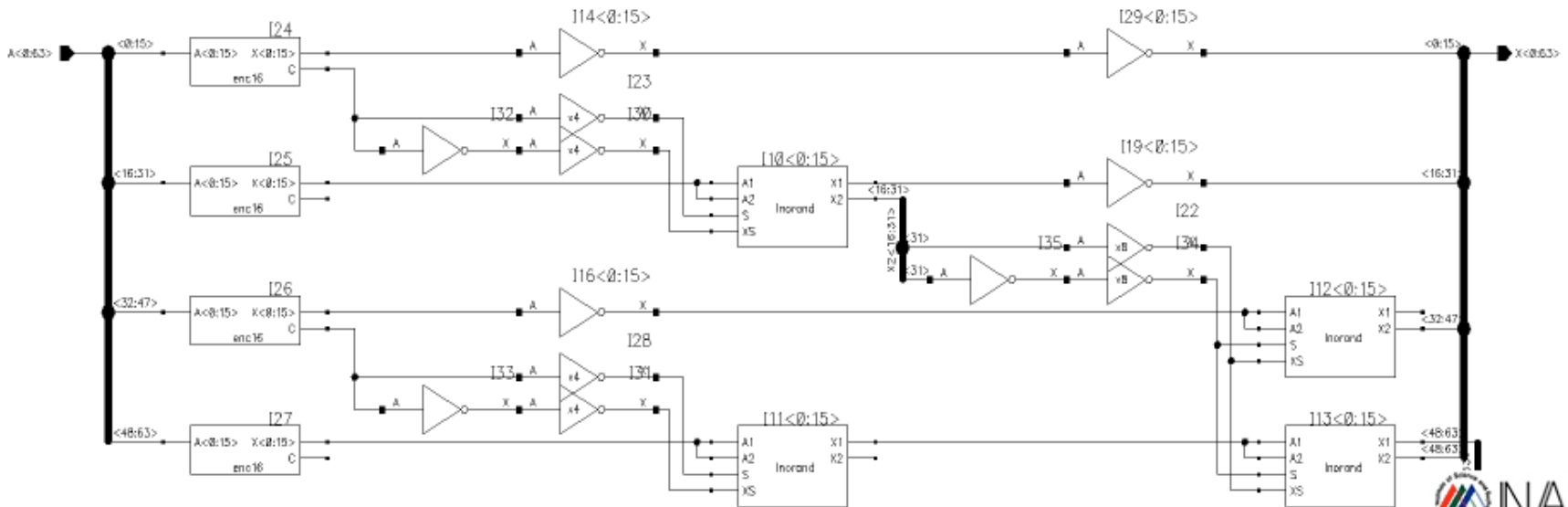
NAIST

プライオリティエンコーダ（エンコード手前まで）

- 16bitプライオリティエンコーダ（enc16）



- 64bitプライオリティエンコーダ（enc64）

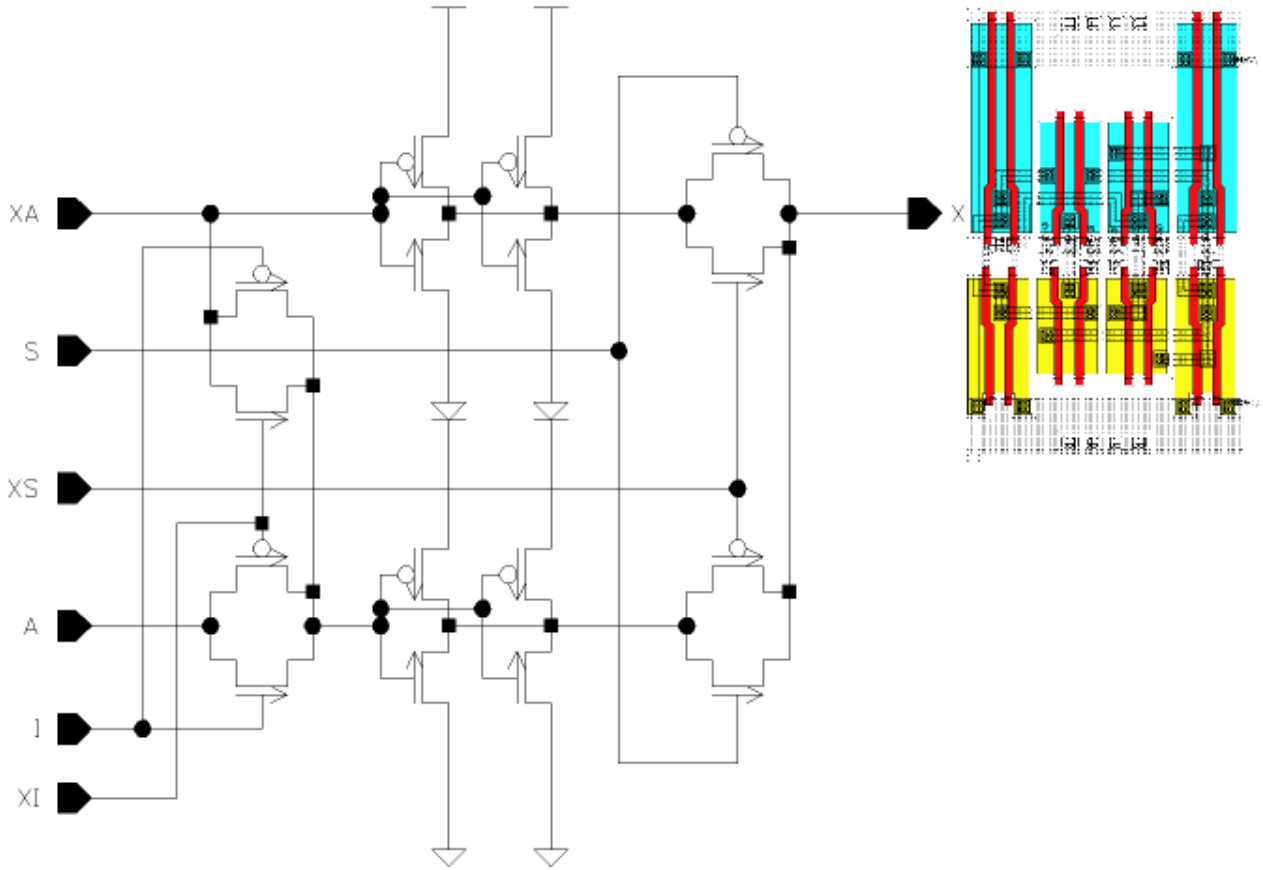


NAIST

ビット長の異なる加減算器用リーフセル

入力選択付き排他論理和 … lseor

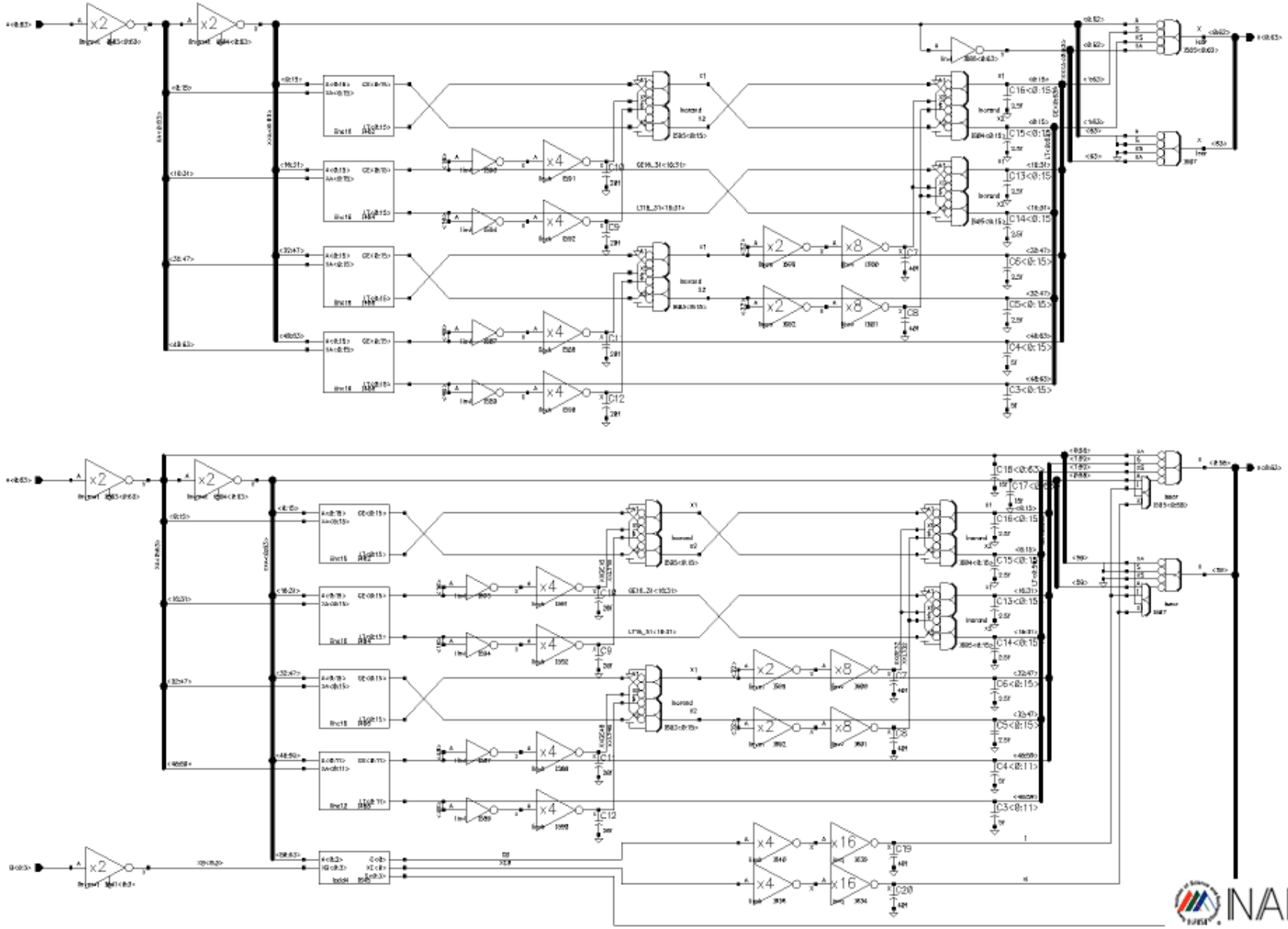
- Iが1の時，XAとSの排他論理和を出力
- Iが0の時，Aを出力
- インクリメンタと加減算器の組み合わせによる加減算器が作れる



NAIST

ビット長の異なる加減算器用リーフセル

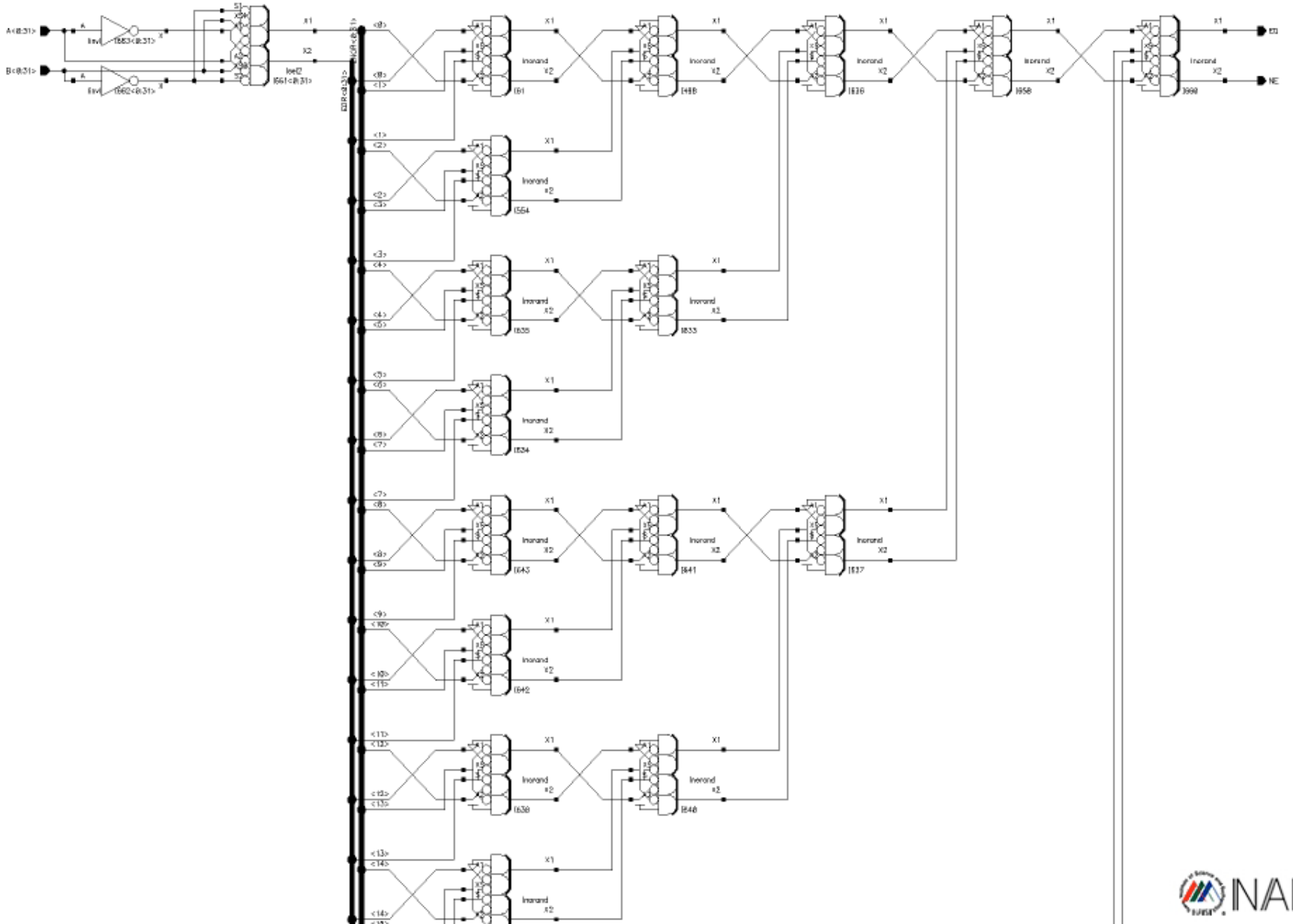
上は，前述の**64+1bit**加算器（インクリメンタ），下は**64+4bit**加算器



NAIST

Lnorandの応用

初段に**Isel2**を使うと，**Inorand**ツリーにより多ビット一致比較器が作れる



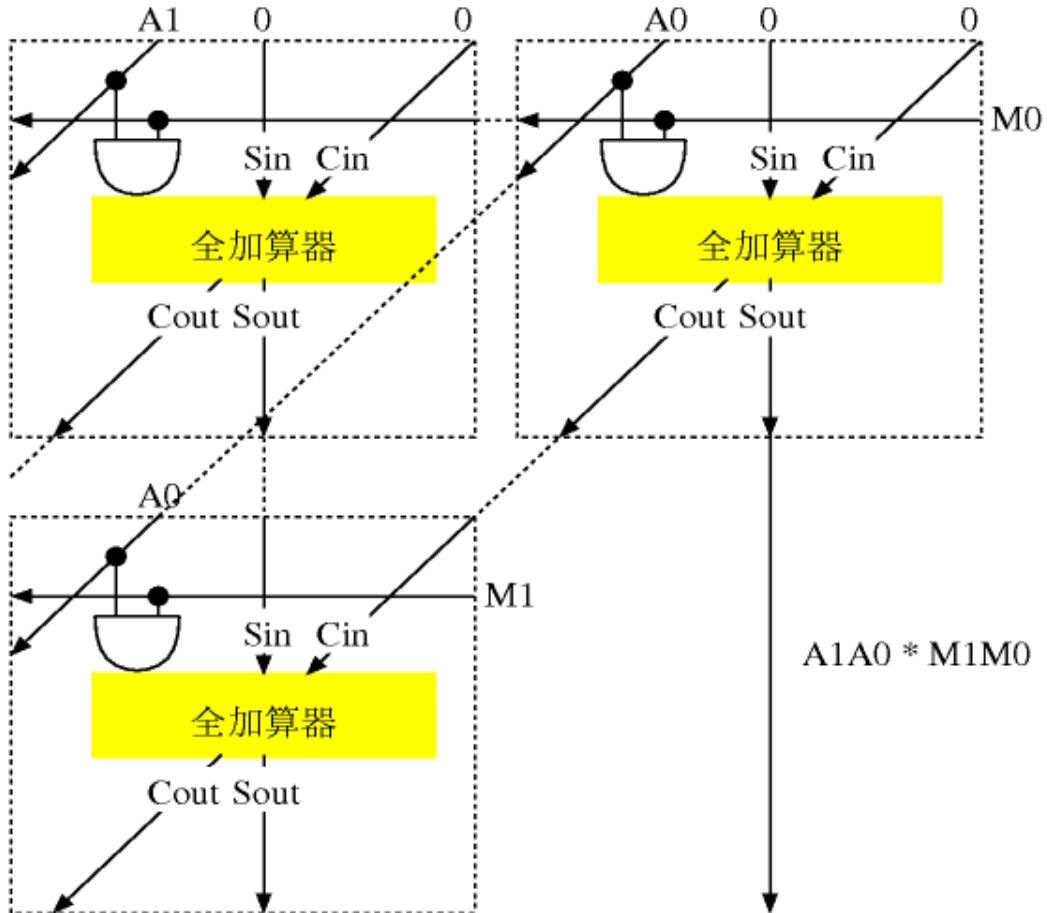
NAIST

乗算器と除算器

今までの知識を元に,以上のリーフセルを使った乗算器を考えてみること

ヒント：Ａを被乗数,Ｍを乗数,乗算器の前段を**CSA(**全加算器**)**木とすると,

- Sout = (A&M) ^ (Sin ^ Cin)
- Cout = ((A&M) & (Sin ^ Cin))|(Cin & ~(Sin ^ Cin))



NAIST

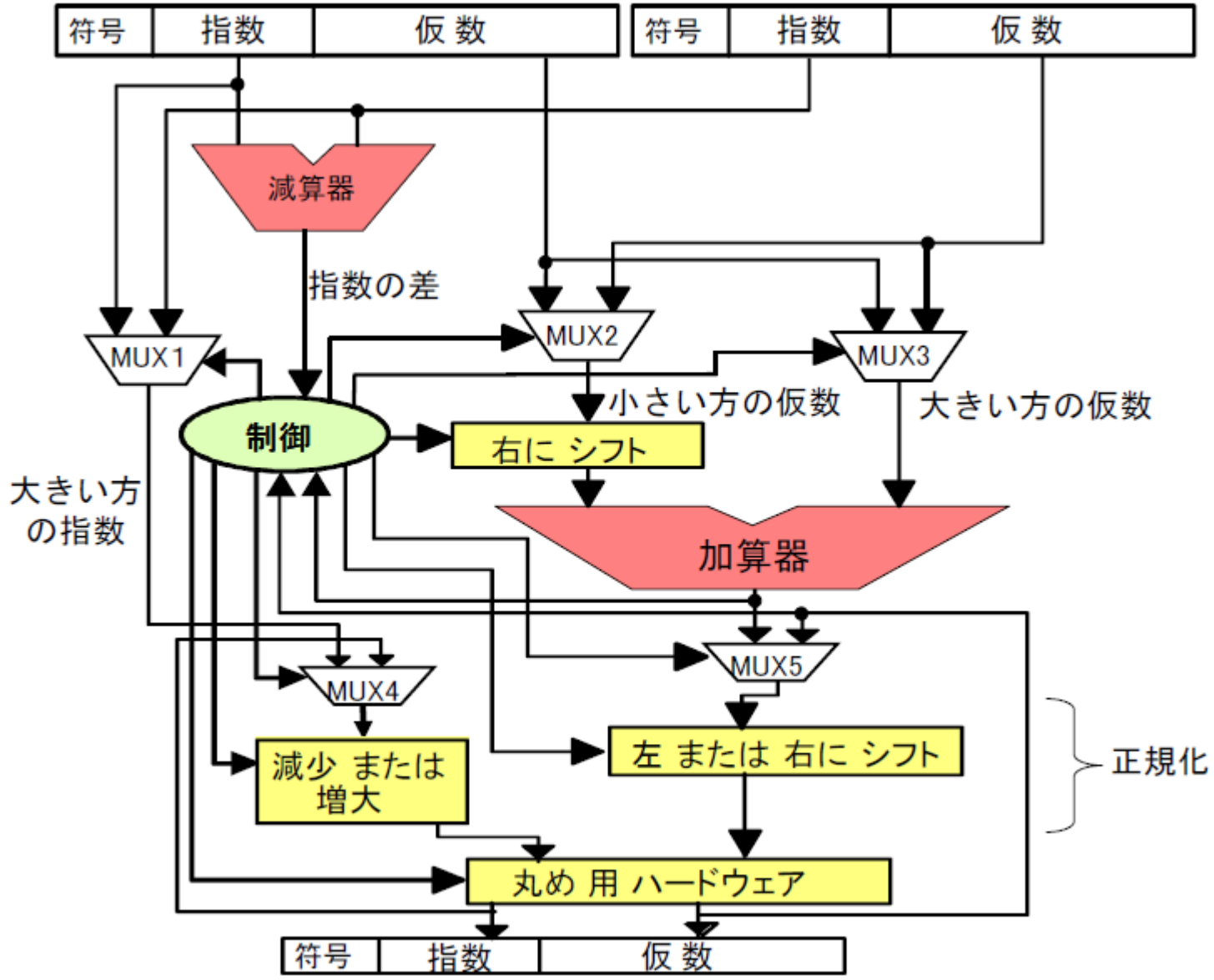
除算

引き戻し法のアルゴリズム

iを1からnまで1ずつ増やしながら以下の処理を繰り返す.
・現在の A の上位 i ビットの部分のみを i ビットの 2 進数と考え，それを A’とする.
  – A’≥B ならば，商の最上位から i 桁目を 1 とし，A から B の 2^(n−i) 倍 を引く.
  – A’＜B ならば，商の最上位から i 桁目を 0 とし，A はそのままにする.

i＝n までの処理が終わると商の全ての n ビットが求まり，その時の A の値が剰余となる.

浮動小数点演算とパイプライン化



今日はここまで