アルゴリズムのハード化

工学研究科 電気工学専攻

久門 尚史

はじめに

- アルゴリズムのハード化とは?
- FPGA を用いたアルゴリズムのハード化
  - FPGA の構造
  - FPGA を用いた設計
- 符号化とアルゴリズム
  - アナログとディジタル: グレイコードによる算術演算
- 制約と局所性
  - 局所化と誤差伝搬: 逆ラプラス変換の高精度化
- 物理現象とアルゴリズムのハード化
  - 物理的特性を生かしたハード化: 可逆演算

アルゴリズムとは?

- アルゴリズムとは，計算の手順 (算法) のことを言う．
- 身近なアルゴリズム
  - 算術演算
  - ガウスの消去法
  - 高速フーリエ変換 (FFT)
  - クイックソート
- ポイント
  - 問題のもつ構造 (トポロジー)
  - 局所化 (Localization)

アルゴリズムとは? — 例: 乗算

- 乗算を筆算で解く

$$
\begin{array}{rrrrrrr}
 & & & 0. & 1 & 2 & 3 \\
 & & \times & 0. & 4 & 5 & 6 \\
\hline
 & & & & 7 & 3 & 8 \\
 & & & 6 & 1 & 5 & \\
 & & 4 & 9 & 2 & & \\
\hline
0. & 0 & 5 & 6 & 0 & 8 & 8
\end{array}
$$

- ポイント
  - 小さい問題に分割 (構造に基づく)
  - 局所的関係

アルゴリズムとは? — 例: LU 分解

- 連立一次方程式 $Ax = b$
- アルゴリズム
  - LU 分解:

$$Ax = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -3 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -1 & -2 \\ 0 & 0 & -4 \end{bmatrix} x = LUx = b$$

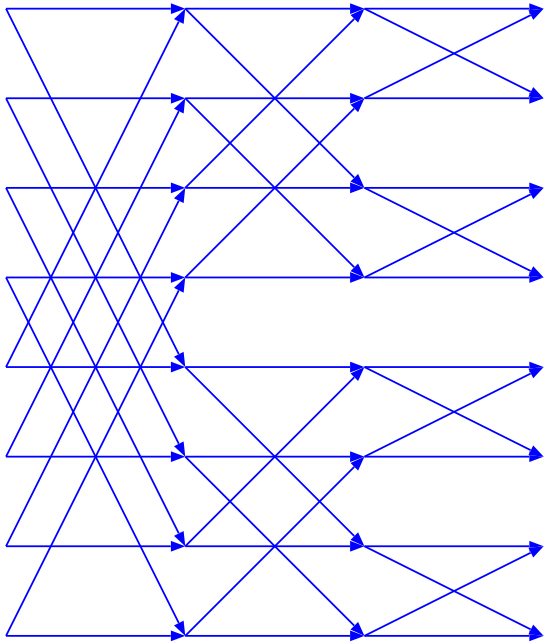
  - 後退代入: $y \equiv Ux$ とおき，$Ly = b$ を解いてから $Ux = y$ を解く.
- ポイント
  - 等価なものに変換
  - 大域的関係 (Matrix) を Local な関係 (後退代入) に

アルゴリズムとは? — 例: 高速フーリエ変換

- 離散フーリエ変換

$$F = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W & W^2 & \cdots & W^{N-1} \\ 1 & W^2 & W^4 & \cdots & W^{2N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W^{N-1} & W^{2N-2} & \cdots & W^{(N-1)(N-1)} \end{bmatrix}, \qquad W^n = 1$$

- バタフライ演算
  - 回転群の構造
  - 分解して結合
  - $O(n^2)$ から $O(n \log n)$ へ



ハードとは? — 回路に対する視点

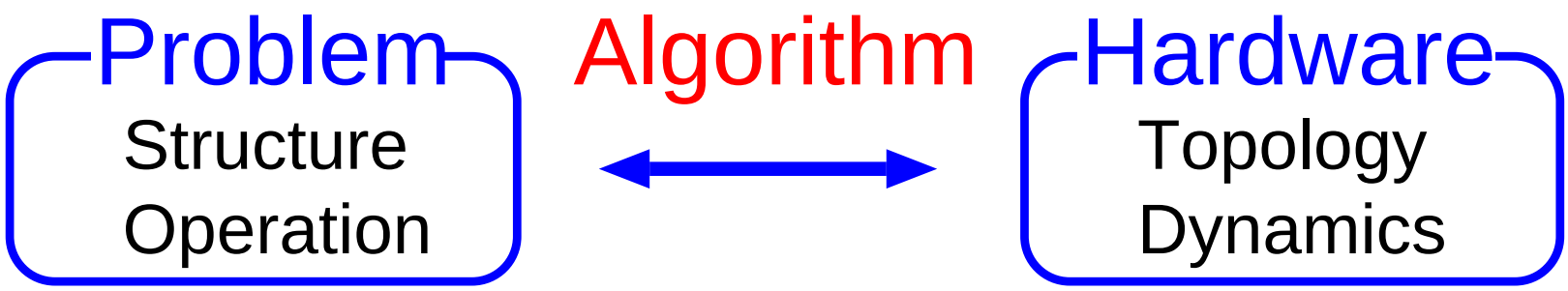
- アルゴリズムを回路のダイナミクスに対応させて実行.
- 回路の分類
  - 視点: 信号，エネルギー
  - 役割: 変換，伝送，保存
  - 例えば

	変換	伝送	保存
信号	CPU	通信線	メモリ
エネルギー	コンバータ	送電線	バッテリー

- 回路の要素
  - 素子: L, C, R，トランジスタ，OP アンプ，レジスタ,..
  - トポロジー: 素子のつながり方

ハードとは? — アルゴリズムとハードの関係

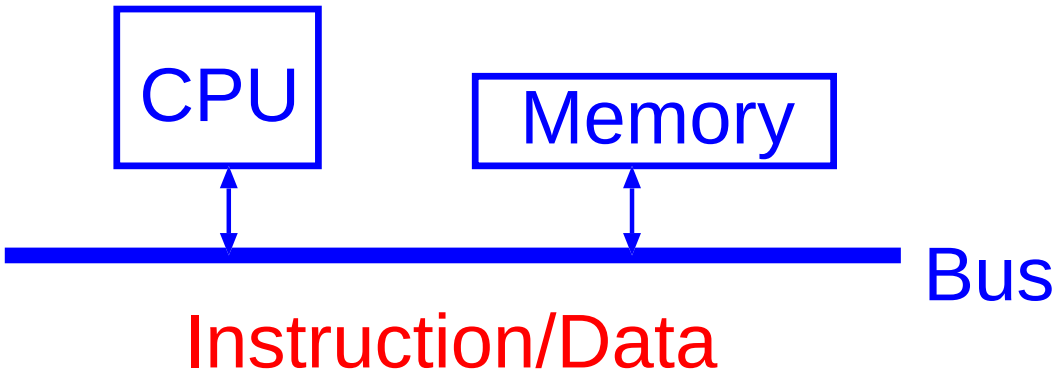
- 問題を解く



  - アルゴリズム: 問題とハードウェアの対応
- 一般の計算機: ソフトウェアによる解法
  - ノイマン型アーキテクチャ
  - 問題を解く過程を CPU の手続きに変換
  - 可変性: 手続きをメモリ上におくため，多様な問題に対応

ハードとは? — ノイマン型のボトルネック

- ノイマン型アーキテクチャ



  - 長所: ソフトウェアにより種々の問題に対応
  - 短所: バスによるボトルネック

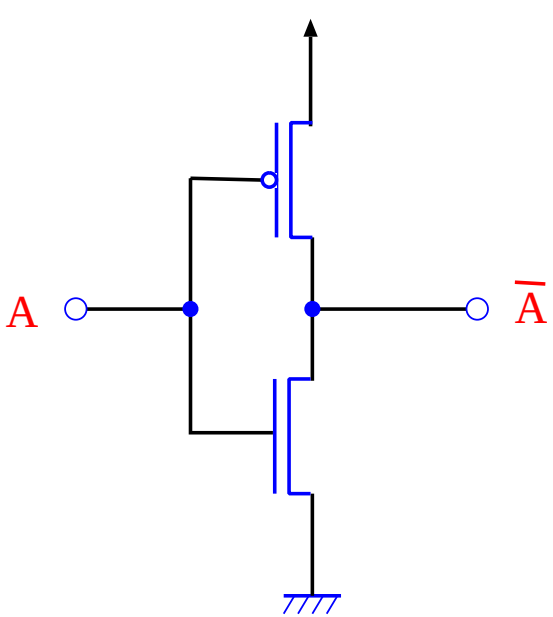
- 再構成可能な集積回路 (**F**ield **P**rogrammable **G**ate **A**rray)
  - ソフトウェアのような可変性
  - 問題に対応した構造のハードウェアが組める (並列性，高速性)

アルゴリズムのハード化と FPGA

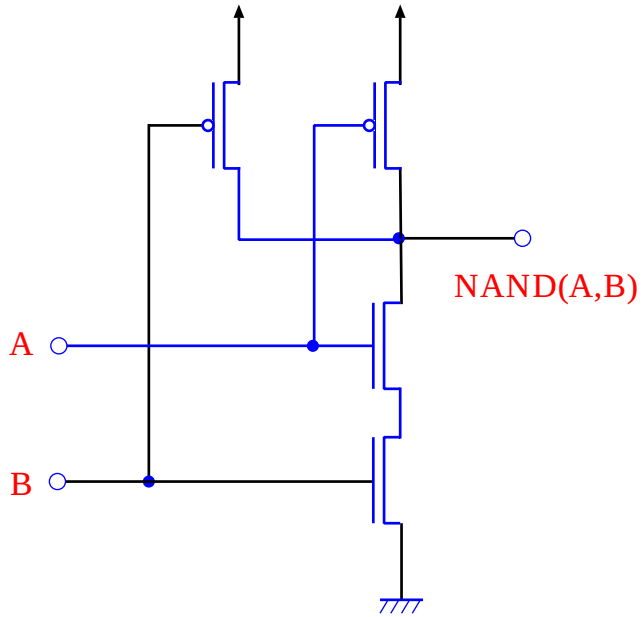
- アルゴリズムのハード化とは?
- FPGA を用いたアルゴリズムのハード化
  - FPGA の構造
  - FPGA を用いた設計
  - FPGA の発展
- 符号化とアルゴリズム
- アルゴリズムの局所化，可逆性
- 物理現象とアルゴリズムのハード化

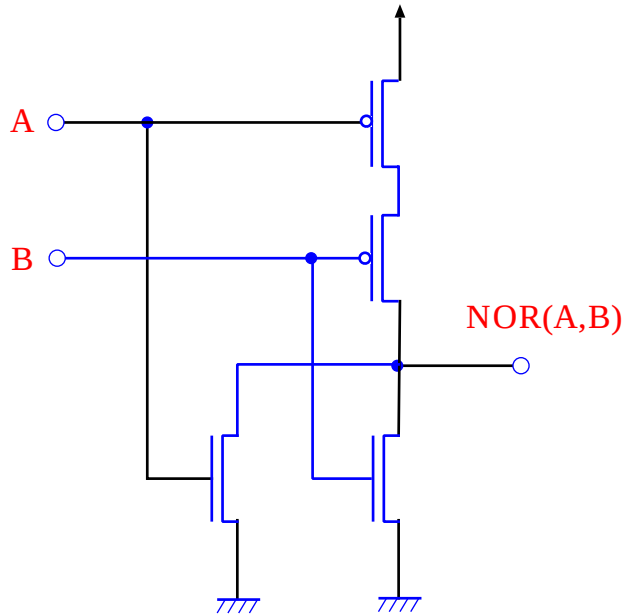
FPGA の構造 — 固いハード: CMOS 論理

- CMOS 構造
  - 相補性を利用したディジタル演算
  - 素子は MOS スイッチのみで回路のトポロジーを変化させる.
  - 論理を変更することは不可.



Inverter

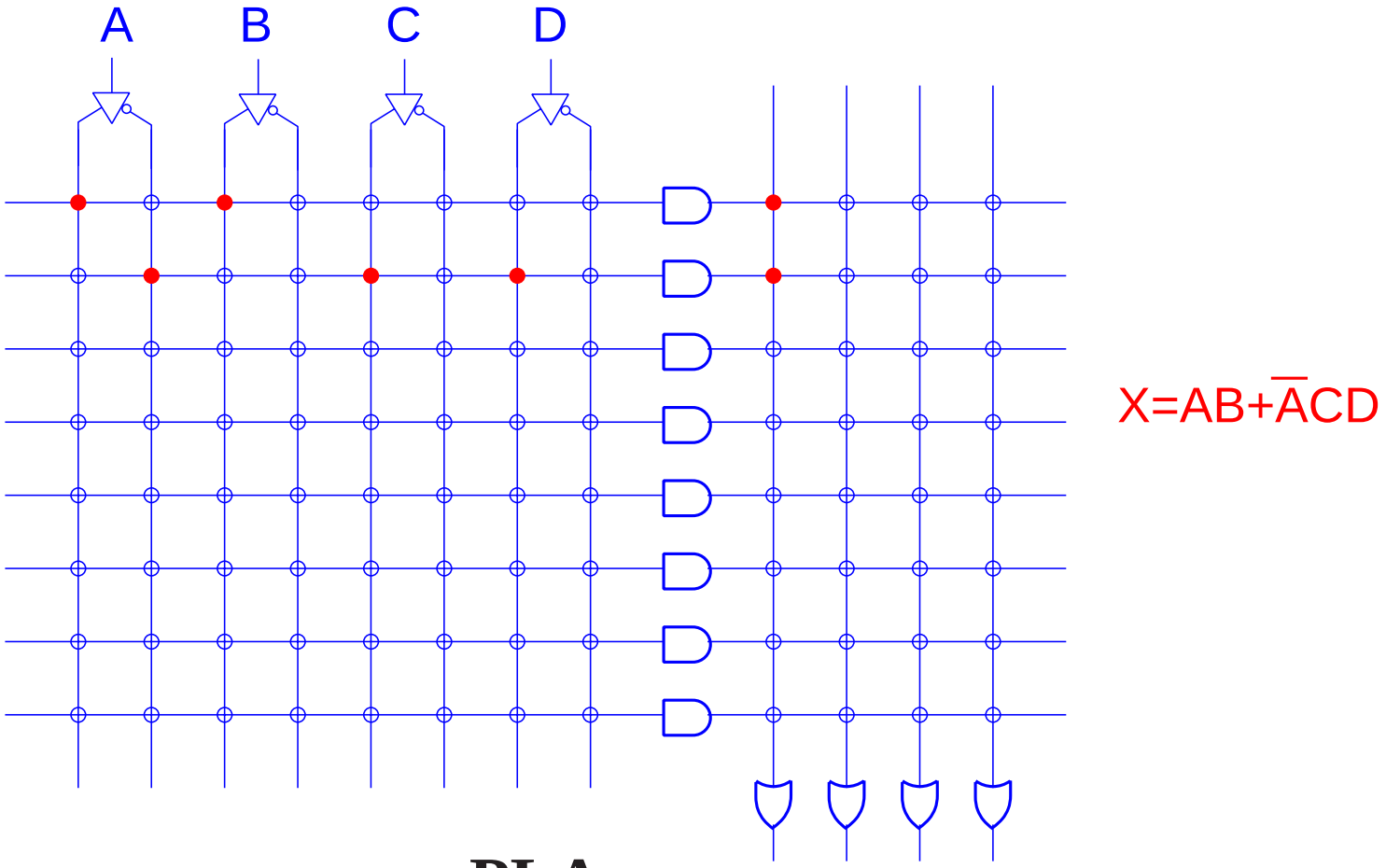




NAND と NOR

FPGA の構造 — トポロジーの可変性

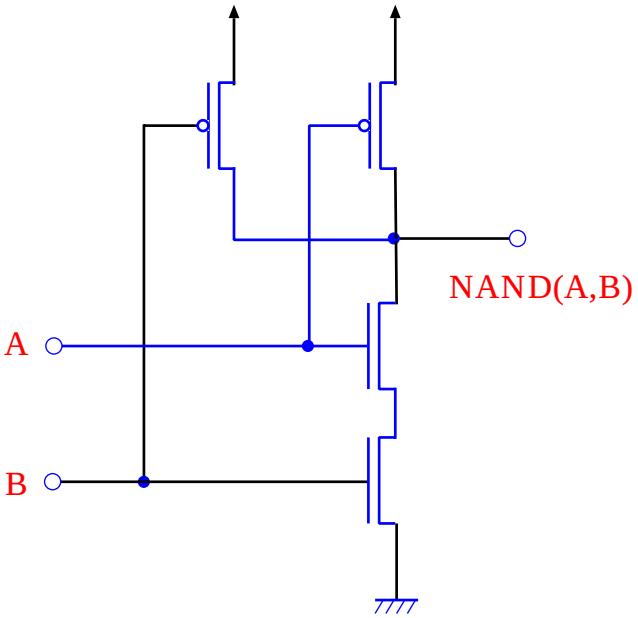
- PLA (Programmable Logic Arrays)
  - トポロジーを変化させて任意の積和論理を構成
  - 論理素子は AND と OR



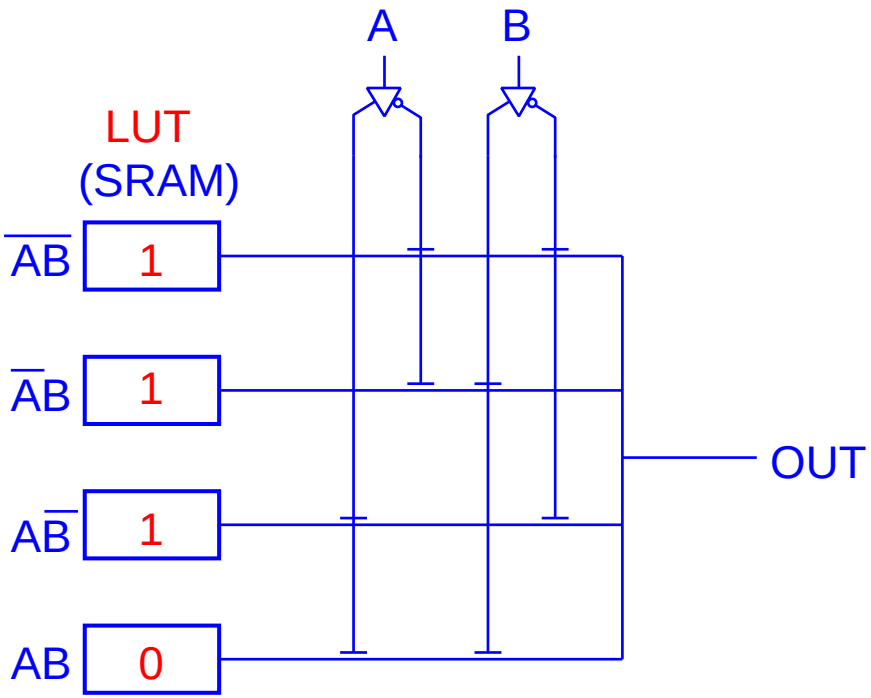
PLA

FPGA の構造 — FPGA の可変論理の仕組み

- LUT(Look Up Table) を利用して論理を可変に
  - メモリにより回路を表現
  - その場で計算するか，出力パターンをメモリに蓄えておくか．



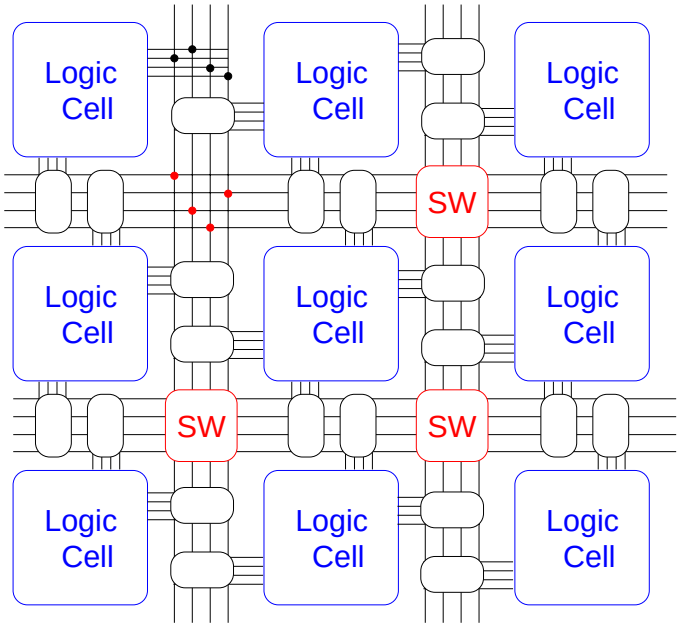
CMOS による NAND



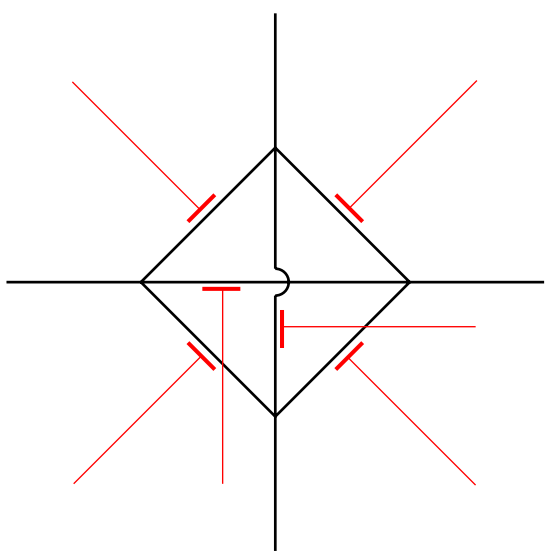
LUT による可変論理

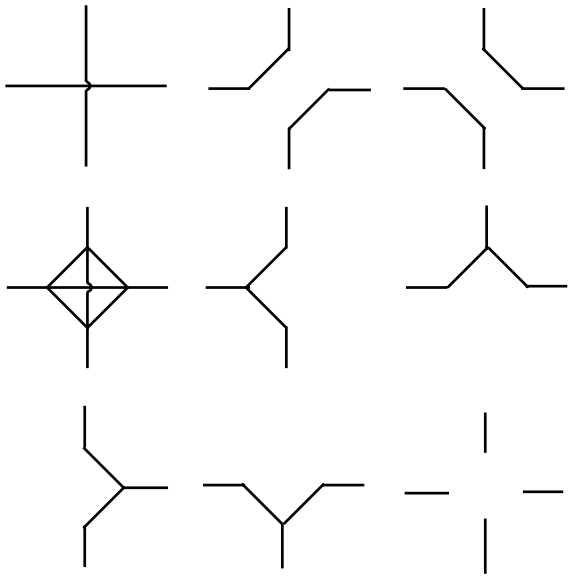
FPGA の構造 — トポロジーの可変性

- FPGA の内部構造
  - Logic Cell[LUT(論理), FF(メモリ)] が Matrix 状に配置
  - 各 Logic Cell をつなぐ配線
- 交差点でのスイッチによりトポロジー可変



Logic Cell の配置





SW 部分の構造

FPGA を用いた設計 — 設計の流れ

1. ハードウェア記述言語 (Hardware Description Language) で記述
2. 論理シミュレーション
   - 設計の確認
3. 論理合成
   - 簡単化された回路出力が得られる .
4. FPGA へのマッピング
   - 論理回路を FPGA へのせる形に変換
5. 回路情報を FPGA にダウンロード

FPGA を用いた設計 — HDL 記述

- ハードウェア記述言語 (Hardware Description Language)
  - 論理回路の動作を記述
  - 代表的な HDL: Verilog-HDL, VHDL, SFL 等

例:
4 ビットカウンタ
の Verilog 記述

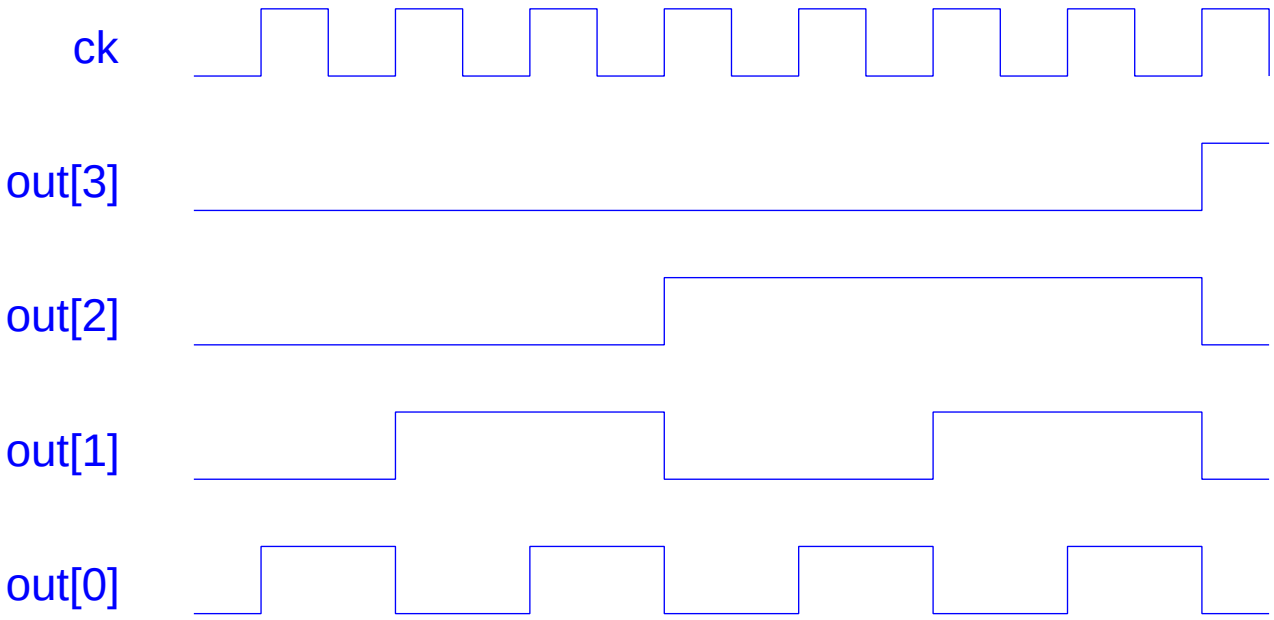
```
module count(out, ck);
 output [3:0]  out;
 input ck;
 reg [3:0] q;

 always @(posedge ck)
 begin
  q <= q+1;
 end
 assign out = q;
endmodule
```

FPGA を用いた設計 — シミュレーション

- 記述した HDL を検証する
  - テストするための記述
  - シミュレータによる検証 (VCS 等)

```
module testcount;
 wire [3:0] out;
 reg ck;
 initial
 begin
  ck <= 0;
  #200
  $finish;
 end
 always #10
 begin
  ck <=~ck;
 end
 count inst0 (out, ck);
endmodule
```



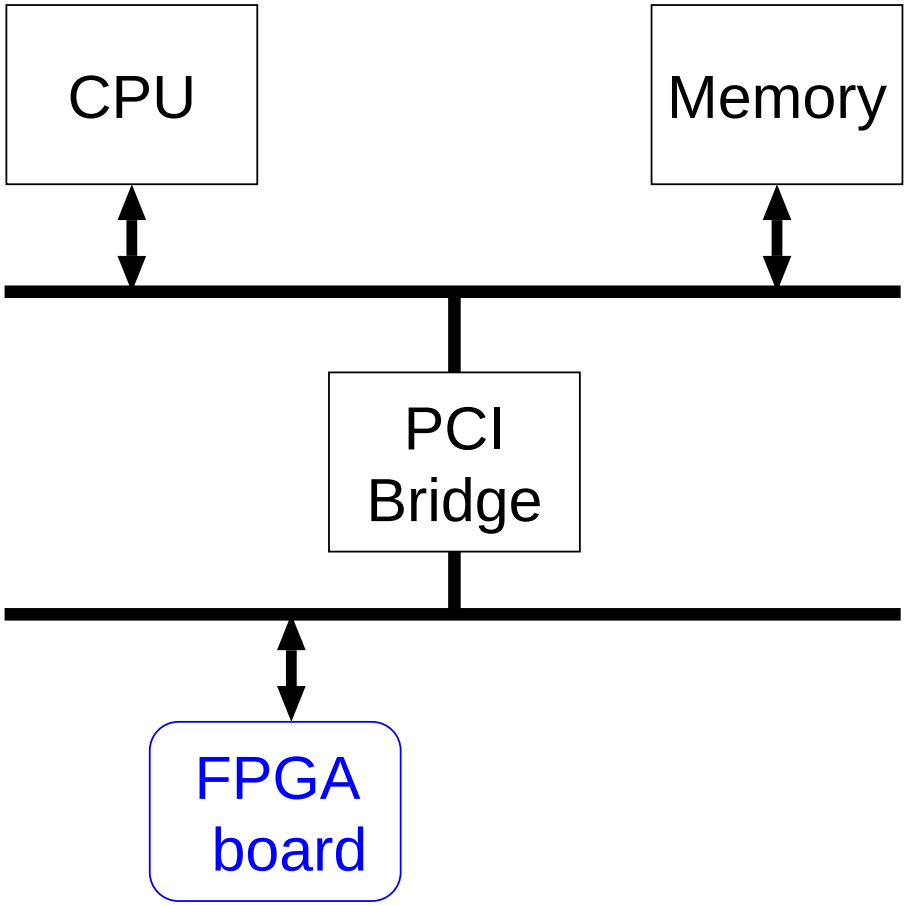
シミュレーション結果

FPGA を用いた設計 — 論理合成からのプロセス

- 論理合成 (FPGA Express 等)
  - 論理合成
  - 論理の簡単化
  - 論理回路が出力として得られる．
- FPGA へのマッピング
  - FPGA へダウンロードできる形のファイル．
- ダウンロード
  - JTAG，ROM を用いて回路情報をダウンロード

FPGA を用いた設計 — PCI バスへの実装

- パソコンの PCI バスへ FPGA ボードを実装
- デバイスドライバが必要

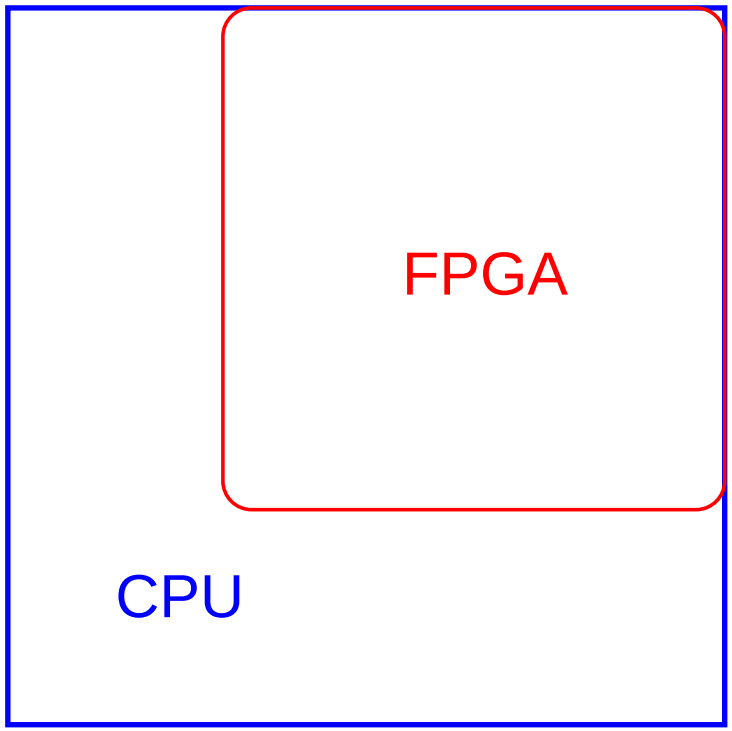


FPGA の発展 — 動的再構成可能な FPGA

- 動的再構成とは
  - 動作中に 1 クロックで瞬時に回路を再構成する機能
- 応用例
  - 低消費電力化
    - 必要なところだけ動作させる．
  - 時分割でチップを活用
    - チップ 1 個で複数チップの役割
  - 回路の自己生成 (自己増殖)
    - 必要な命令はチップ自身で構成

FPGA の発展 — 集積度の向上がもたらす影響

- 現在は 1 千万ゲート規模
- 将来の回路技術
  - 微細化の進展
  - 単電子トランジスタ
  - 分子回路
- チップの形態
  - CPU の一部に動的再構成可能部をもつチップ
  - キャッシュから FPGA へ



FPGA の発展 — 再構成可能チップの将来像

- すべてのチップが再構成可能部をもつ
- 時間スケールで見たチップの専用性
  - チップ作成時
  - アプリケーションの実行前
  - アプリケーションの実行中のモード変換
  - クロック単位の回路再構成
- 回路情報もソフトウェア
  - Web を通して回路情報のダウンロード
  - データ，命令，論理もメモリーベースに

符号化とアルゴリズム

- アルゴリズムのハード化とは?
- FPGA を用いたアルゴリズムのハード化
- 符号化とアルゴリズム
  - 符号化とは
  - グレイコード演算
- 制約と局所化
- 物理現象とアルゴリズムのハード化

符号化とは — アルゴリズムと符号化

- アルゴリズムのハード化を考えるときの枠組み
  - ディジタルかアナログか
  - どのように符号化するか
- グレイコードはアナログ的要素をもつディジタル表現

integers	binary codes	Gray codes
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

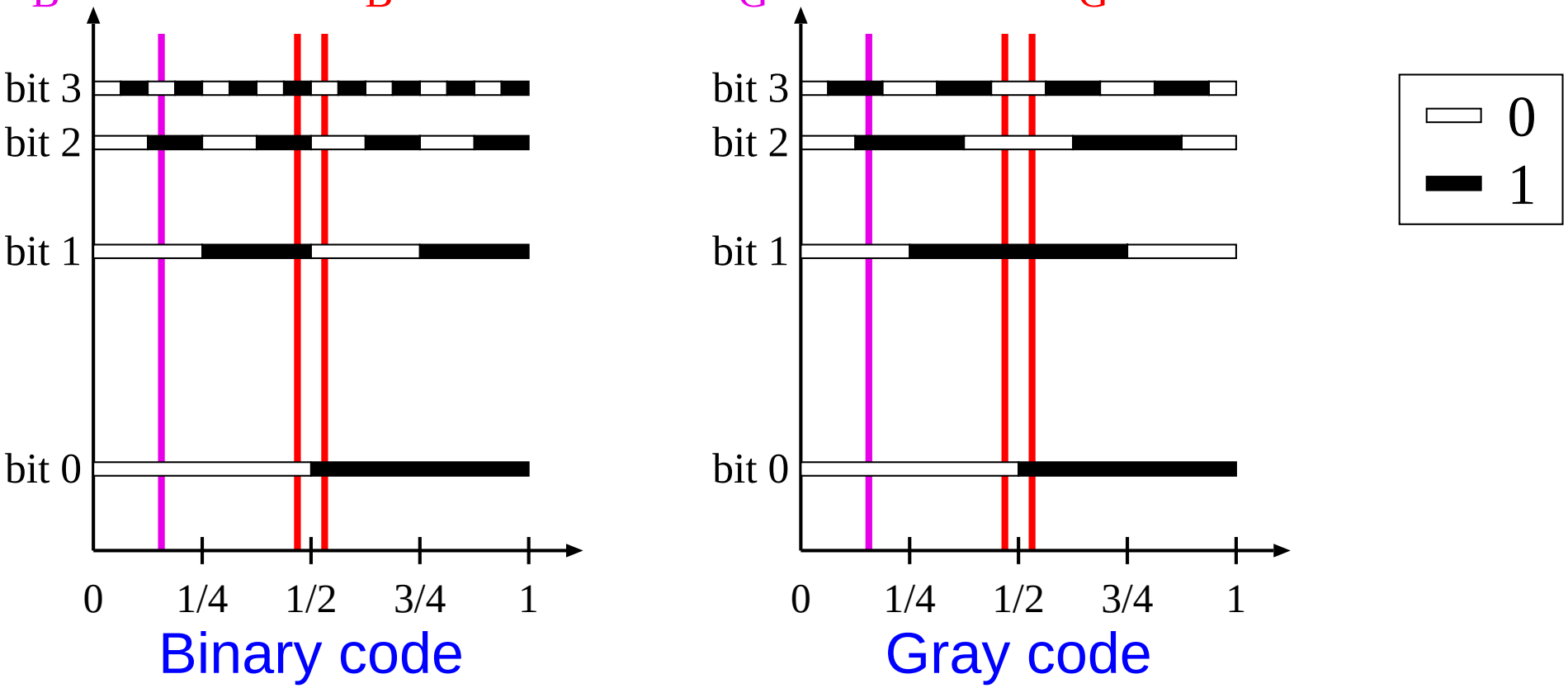
グレイコード演算 — グレイコードの性質

- グレイコードとは
  - 1 増減したときに 1 ビットのみ変化
  - シャフトエンコーダ，ディジタル回路設計に用いられる．
- グレイコードによる算術演算
  - これまでは 2 進コードに変換してから実行
  - 位相の性質により上位桁からの演算
    - 任意精度演算が可能
  - 連続性をもつ数値計算への応用

グレイコード演算 — 実数

- 実数の表現

x_B=0.0010⋯　y_B=0.⊥⊥⊥⊥⋯　x_G=0.0011⋯　y_G= 0.⊥100⋯



- 2 進コードによる演算
  - キャリーの伝搬
  - 下位桁からの計算
- グレイコードによる演算
  - キャリーの伝搬がない
  - 上位桁からの演算

グレイコード演算 — 2 進数の演算

- 2 進数の演算は下位桁からの演算
  - 加算の例

	1	1	0	0
+	0	0	1	1
0	1	1	1	1

	1	1	0	1
+	0	0	1	1
1	0	0	0	0

- 下位桁の変化 (微小変化) で上位桁が影響を受ける.
- 実数の位相構造とのちがい

グレイコード演算 — 上位桁からの演算

- 上位桁からの演算とは
  - 入力は上位桁から与えられる
  - 出力も上位桁から計算される

- 区間演算との対応
  - $0.1 \equiv [0.1, 0.2]$
  - 上位桁からの演算が進むことは，区間縮小に対応．

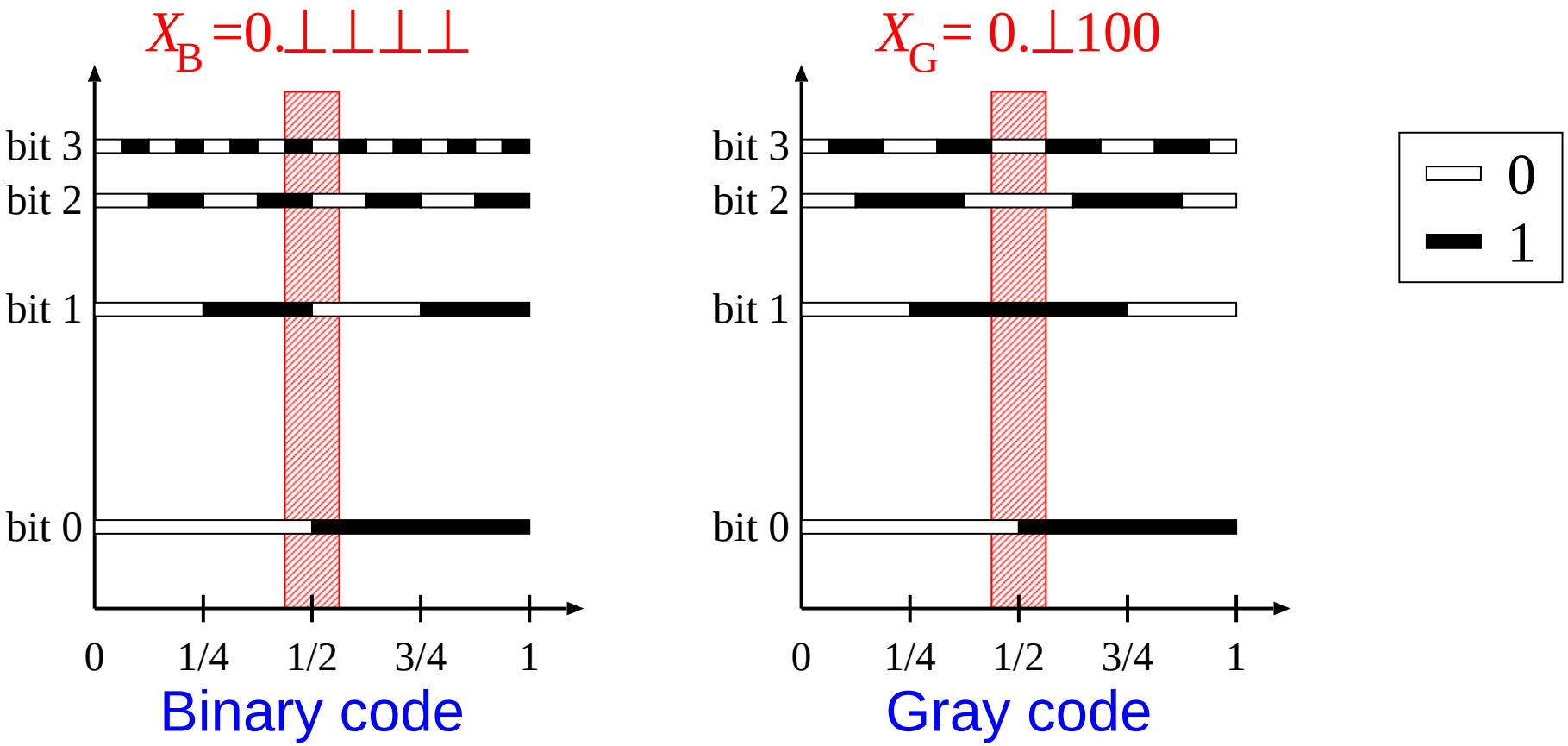
グレイコード演算 — 上位桁からの演算の特徴

- 上位桁からのアルゴリズムは
  - 上位桁から任意の桁の演算ができる
  - 任意精度演算が可能
- 出力の精度が十分でないとき
  - 通常の 2 進コードによる演算では
    - 精度をあげてはじめから再計算
  - グレイコードでは
    - 入力に 1 ビット入れるだけで良い．
- 冗長 2 進数とのちがい，
  - 表現に冗長性がない．

グレイコード演算 — Why Gray codes?

- 出力が区間 $1/2$ を含むとき

$$X = \left[\frac{1}{2} - \varepsilon_1, \frac{1}{2} + \varepsilon_2\right], \quad \varepsilon_1, \varepsilon_2 > 0$$



- すべてのビットが不定
- 1 ビットさえも無視できない

- 1 ビットのみ不定
- 下位桁は無視しても良い．

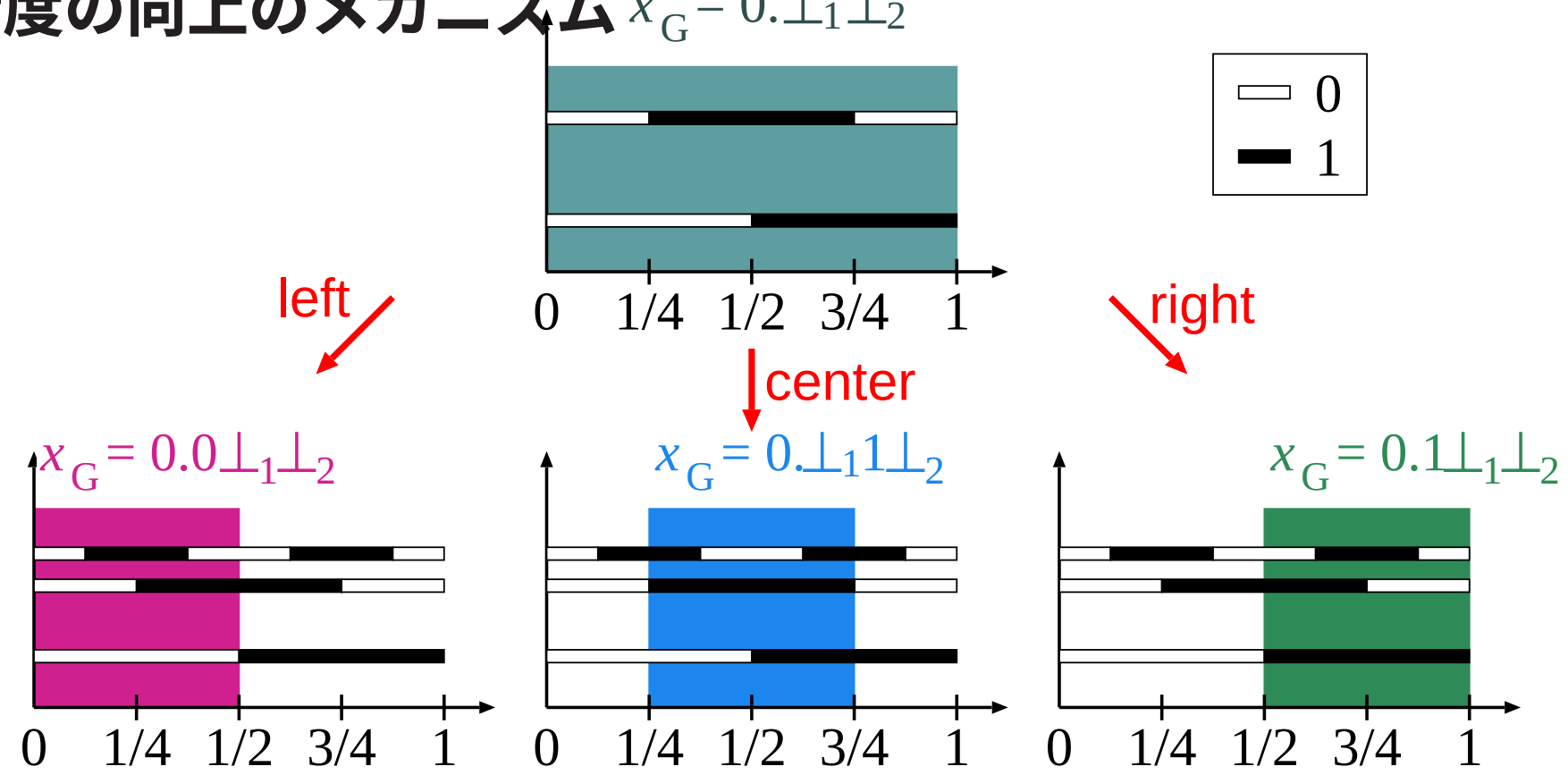
グレイコード演算 — グレイコードと区間の対応

- グレイコード x_G に対応する区間

$$x_G \longleftrightarrow X = [\underline{X}, \overline{X}] = [x-1, x+1] \times 2^{-n_x}$$

x : 中点 (integer),　n_x : スケール指数 (integer)

- 精度の向上のメカニズム

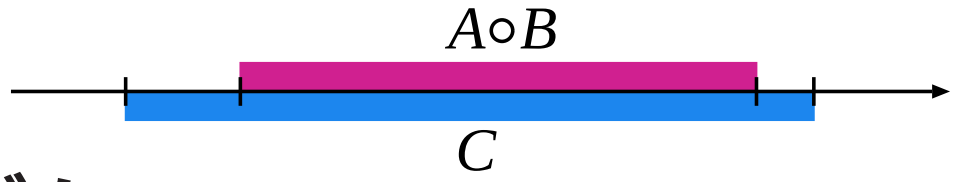


グレイコード演算 — 定義

- 区間演算 $\circ$ のグレイコードへの拡張
- $a_{\mathrm{G}} \circ b_{\mathrm{G}} = c_{\mathrm{G}}$ の定義
  - 入力グレイコードと対応する区間

$$a_{\mathrm{G}} \longleftrightarrow A, \quad b_{\mathrm{G}} \longleftrightarrow B$$

  - 出力グレイコードは $c_{\mathrm{G}} \longleftrightarrow C \supset A \circ B$ で定義される
    区間 C は $A \circ B$ を含む**最も小さい** 区間.

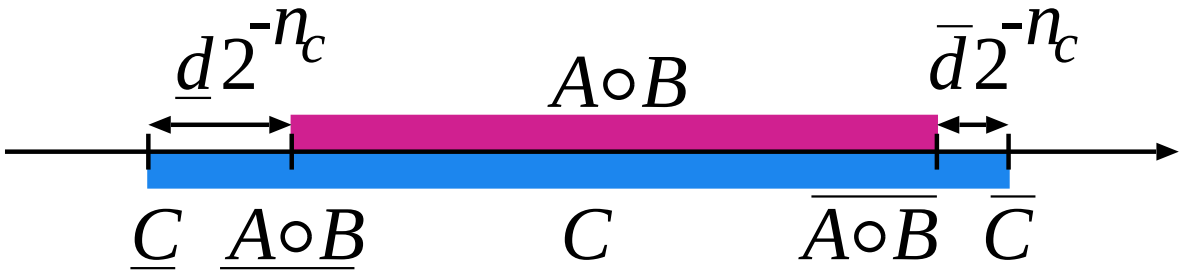


- 基本アルゴリズム
  - $a_{\mathrm{G}}, b_{\mathrm{G}}$ が入力される $\Longrightarrow$ A, B が区間縮小 $\Longrightarrow$ $A \circ B$ の区間縮小
  - C が区間縮小 $\Longrightarrow$ c_{G} が出力される.

グレイコード演算 — アルゴリズム

- 差を表現する $\underline{d}, \overline{d}$

$$\underline{d} = 2^{n_c}\left(\underline{A \circ B} - \underline{C}\right)$$
$$\overline{d} = 2^{n_c}\left(\overline{C} - \overline{A \circ B}\right)$$



  - 計算コストを下げるため $A \circ B, C$ の代わりとして使う

- 出力条件と $\underline{d}, \overline{d}$ の更新規則

conditions	c_{G}	$\underline{d}$	$\overline{d}$
$\overline{d} \geq 1$, **left**	$\perp_1 := P_c$	$\underline{d} := 2\underline{d}$	$\overline{d} := 2(\overline{d} - 1)$
$\underline{d} \geq 1$, **right**	$\perp_1 := \overline{P_c}$	$\underline{d} := 2(\underline{d} - 1)$	$\overline{d} := 2\overline{d}$
$\underline{d} \geq 1/2 \wedge \overline{d} \geq 1/2$, **center**	$\perp_2 := 1, 0$	$\underline{d} := 2(\underline{d} - 1/2)$	$\overline{d} := 2(\overline{d} - 1/2)$

- パリティ　$P_c = \bigoplus_{j=-\infty}^{i_{\mathrm{b}}-1} c_j$　($i_{\mathrm{b}} = \perp_1$ の位置)

制約と局所化

- アルゴリズムのハード化とは?
- FPGA を用いたアルゴリズムのハード化
- 符号化とアルゴリズム
- 制約と局所化
  - 局所化とは
  - 誤差伝搬について
- 物理現象とアルゴリズムのハード化

局所化とは — 制約論理の考え方

- 制約論理とは
  - 関係制約から解を導く: 関係の双方向性
    - 推論関係: 波平 → サザエ → タラオ
    - 偏微分方程式の境界値問題
  - 表現法
    - $xy = 1$ か $y = 1/x$ か
  - グレブナ基底: 代数関係の制約解消
    - 連立代数方程式，

$$x^2 + y^2 - 2 = 0, \quad xy - 1 = 0$$

を制約解消すると，

$$-x^4 + 2x^2 - 1 = 0, \quad x^3 - 2x + y = 0$$

局所化とは — 線形方程式の場合

- 線形連立方程式は $Ax = y$ はグローバルな関係．
  - $x \to y$ は容易だが，$y \to x$ は複雑．

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- LU 分解して L^{-1} を左からかけると，

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & a_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} l'_{11} & 0 & \cdots & 0 \\ l'_{21} & l'_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l'_{n1} & l'_{n2} & \cdots & l'_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- 関係が局所化され，$x \to y$ と $y \to x$ が対等になる．

局所化とは — 畳込みの局所化

- 線形システムの表現

$$y(t) = \int_0^t g(t-\tau)x(\tau)\mathrm{d}\tau$$

  - $x(t) \to y(t)$ は容易．$y(t) \to x(t)$ は複雑．

- ラプラス変換での線形システムの表現

$$Y(s) = G(s)X(s), \quad G(s)^{-1}Y(s) = X(s)$$

  - 関係が局所化されて，$X(s) \to Y(s)$ と $Y(s) \to X(s)$ は対等．

- ラプラス変換部分は局所化されているか?

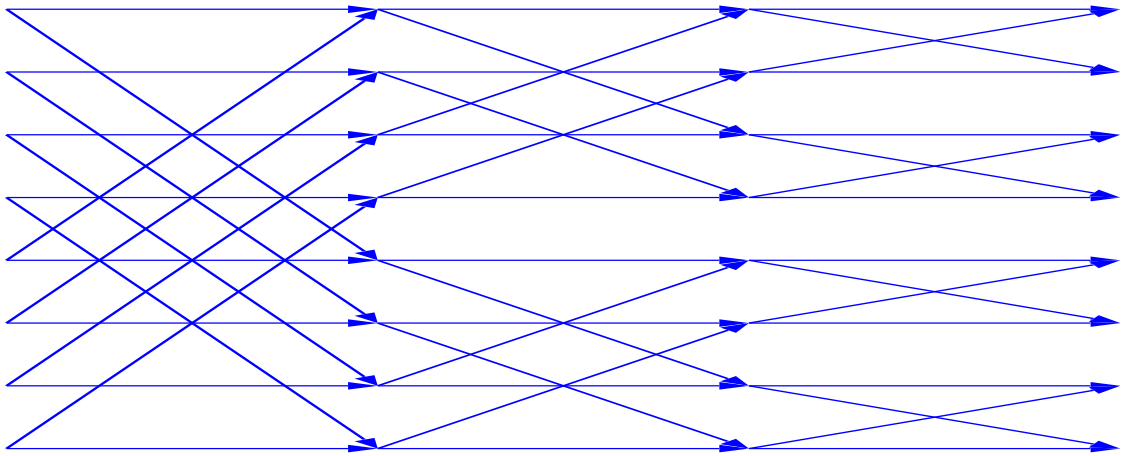
局所化とは — ラプラス変換と局所化

- ラプラス変換

$$F(s) = \int_0^\infty f(t)\mathrm{e}^{-st}\mathrm{d}t \quad (s = a + \mathrm{j}\omega)$$

$$= \int_0^\infty f(t)\mathrm{e}^{-at}\mathrm{e}^{-\mathrm{j}\omega t}\mathrm{d}t$$

- $f(t)$ に e^{-at} をかけてフーリエ変換．
- バタフライ演算は可逆: 順逆ラプラス変換は容易?



誤差伝搬について — 数値逆ラプラス変換とは

- 逆ラプラス変換

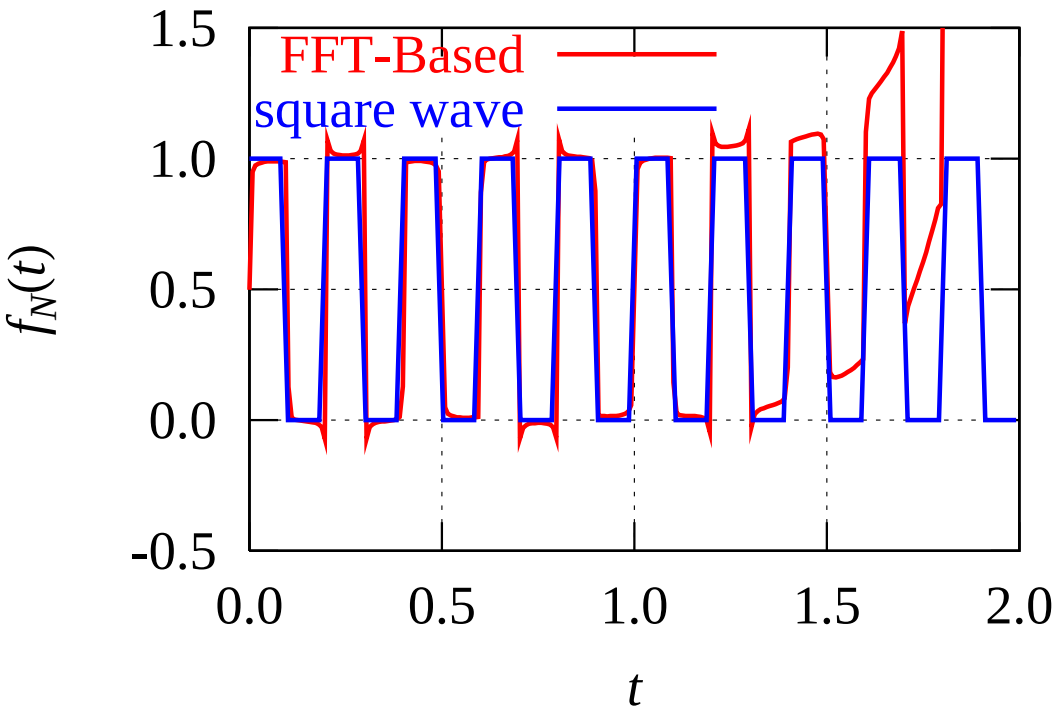
$$f(t) = \mathcal{L}^{-1}[F(s)](t) = \frac{1}{2\pi \mathrm{j}} \int_{a-\mathrm{j}\infty}^{a+\mathrm{j}\infty} F(s)\mathrm{e}^{st}\mathrm{d}s$$

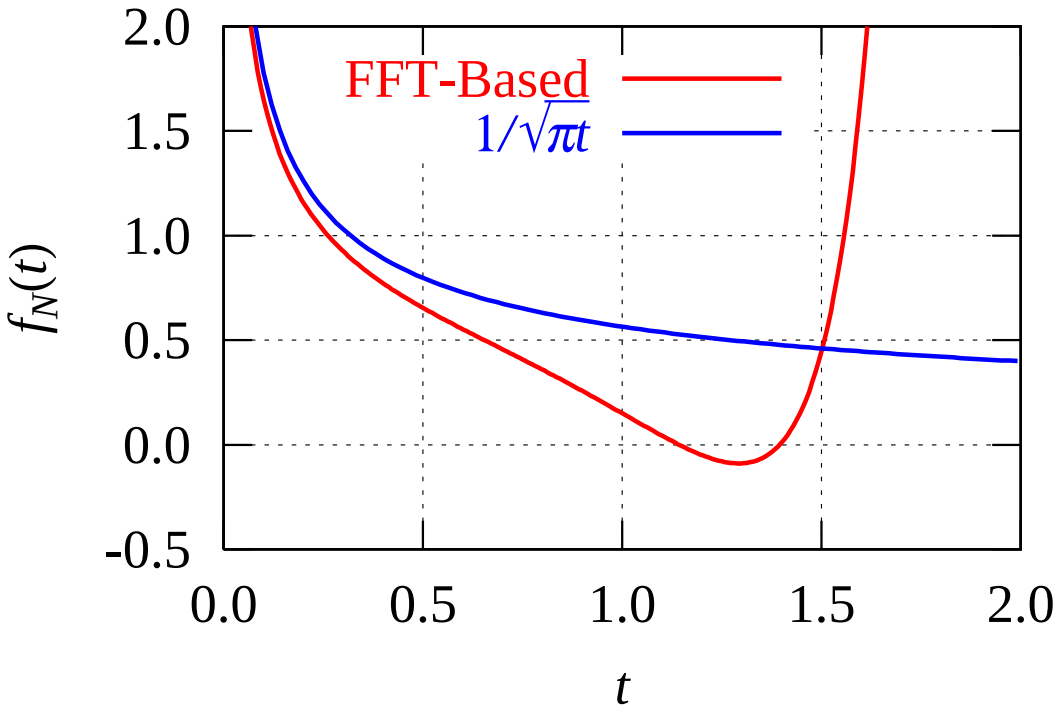
s: 複素周波数, $f(t)$: 原関数, $F(s)$: 像関数

- **数値逆ラプラス変換** = 数値的に Bromwich 積分を評価
  - これまでの提案手法
    - 確率密度関数、双線型変換、チェビシェフ関数、ラゲール関数、ルジャンドル関数、ガウス積分、フーリエ級数展開、積分路変更、オイラー変換、ε アルゴリズム、$\cdots$
  - FFT 型 — 対応する順変換が存在・誤差が大きい

誤差伝搬について — 計算例

- 方形波と $1/\sqrt{s}$





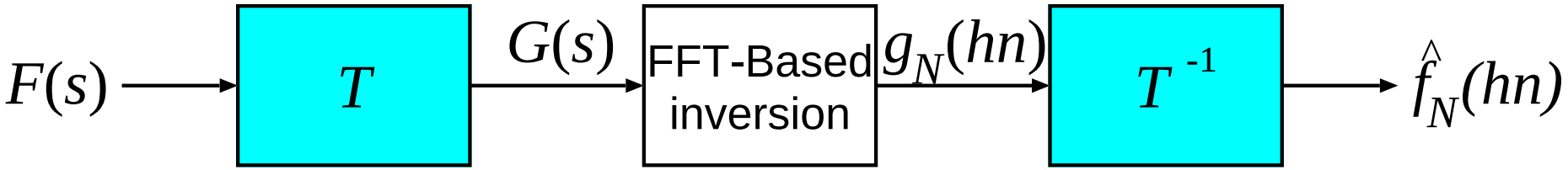
方形波 $F(s) = \dfrac{1}{s(1+\mathrm{e}^{-0.1s})}$

$F(s) = \dfrac{1}{\sqrt{s}}$

- ギブスの現象
- 逆変換できていない

誤差伝搬について — 誤差改善のアイディア

- 誤差改善のアイディア
  - $F(s)$ を誤差の少ないラプラス変換 $G(s)$ に変換
  - 逆変換後に t 領域で元に戻す



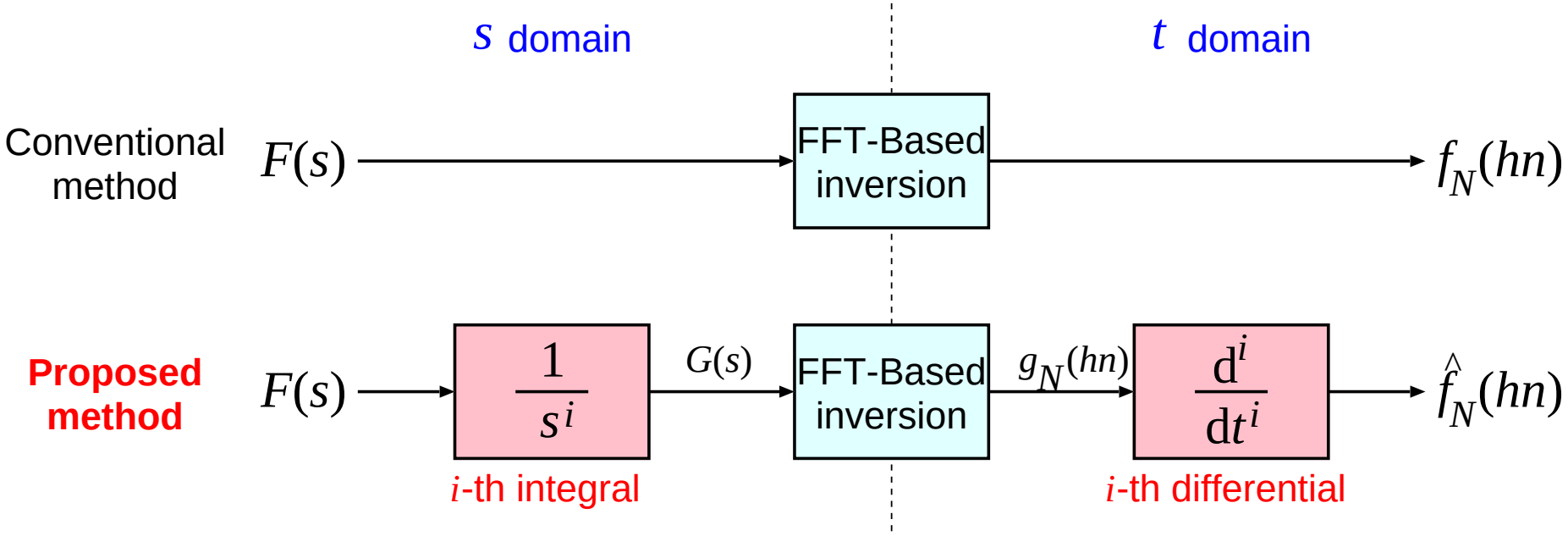
  - T は s 領域でかける
  - T^{-1} は t 領域でかける

誤差伝搬について — 提案手法

- 複素周波数 s は時間領域で微分演算子に対応

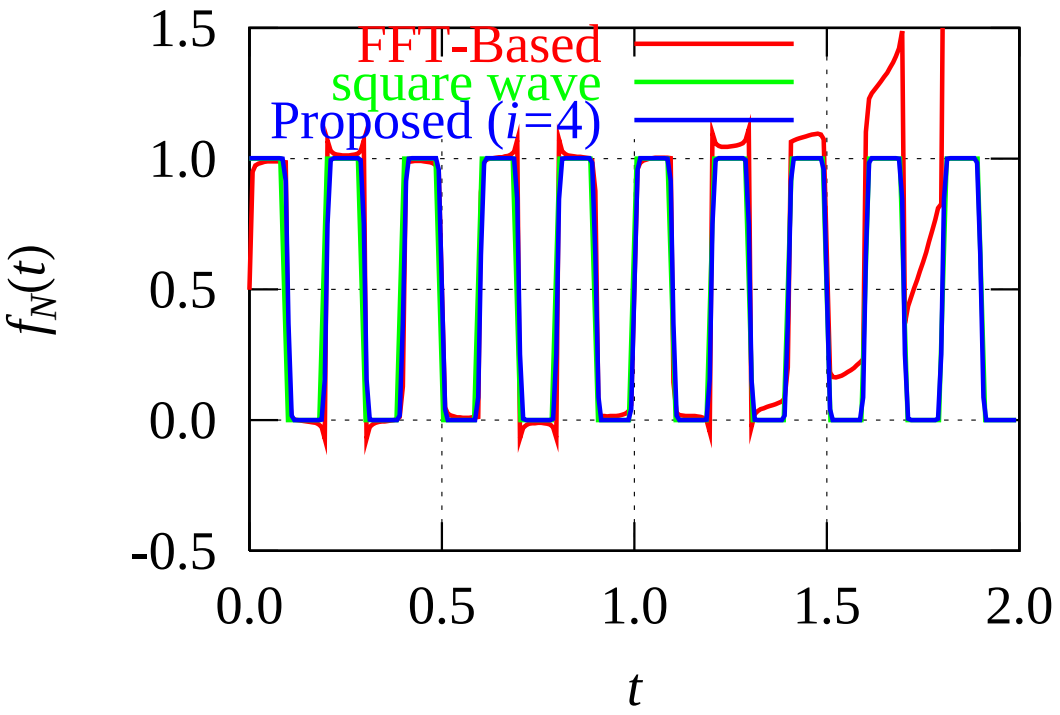
$$f(t) = \mathcal{L}^{-1}[F(s)] = \mathcal{L}^{-1}\left[s^i \cdot \frac{F(s)}{s^i}\right] = \frac{\mathrm{d}^i}{\mathrm{d}t^i}\mathcal{L}^{-1}\left[\frac{F(s)}{s^i}\right]$$

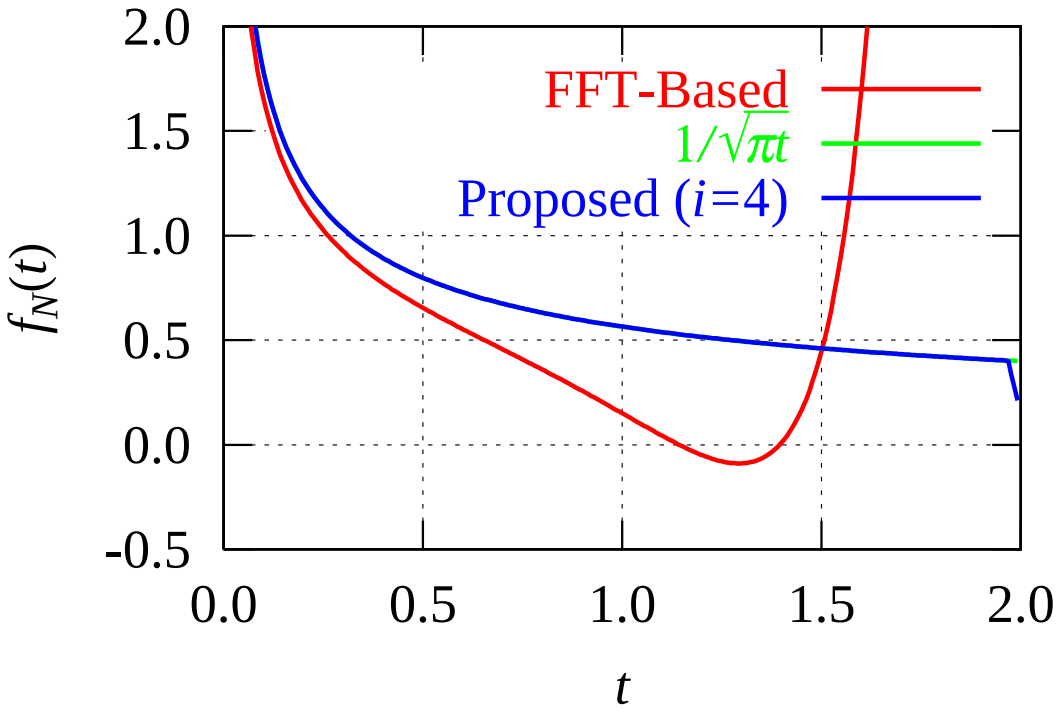
- 提案手法= s 領域で i 階積分 ＋ t 領域で i 階微分



誤差伝搬について — 改善された結果

- 方形波と $1/\sqrt{s}$





方形波 $F(s) = \dfrac{1}{s(1+\mathrm{e}^{-0.1s})}$

$F(s) = \dfrac{1}{\sqrt{s}}$

- 積分によって不連続点を除去、ギブスの現象を回避
- 端点以外は誤差小

物理現象とアルゴリズムのハード化

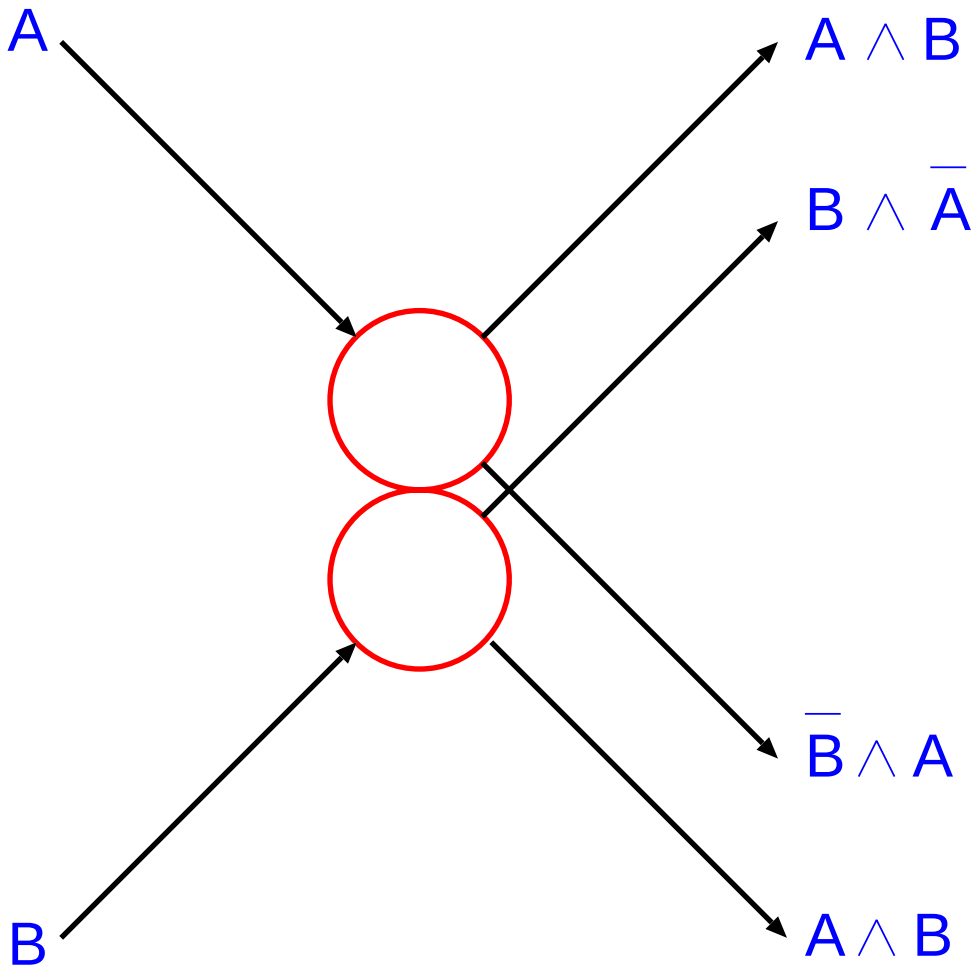
- アルゴリズムのハード化とは?
- FPGA によるルゴリズムのハード化
- 符号化とアルゴリズム
- 制約と局所化
- 物理現象とアルゴリズムのハード化
  - 計算とは
  - 電気回路の特性
  - 可逆回路

計算とは — 種々の物理現象を利用した計算

- アナログかディジタルか
- 多様な計算の仕組み
  - ビリヤードボールコンピュータ
  - 神経細胞の演算
  - 量子計算
  - 超伝導単一磁束量子論理 (SFQ) 回路
  - 光論理素子
  - 分子回路
  - 化学的計算
- 物理現象があれば計算ができる

計算とは — ビリヤードボールによる論理演算

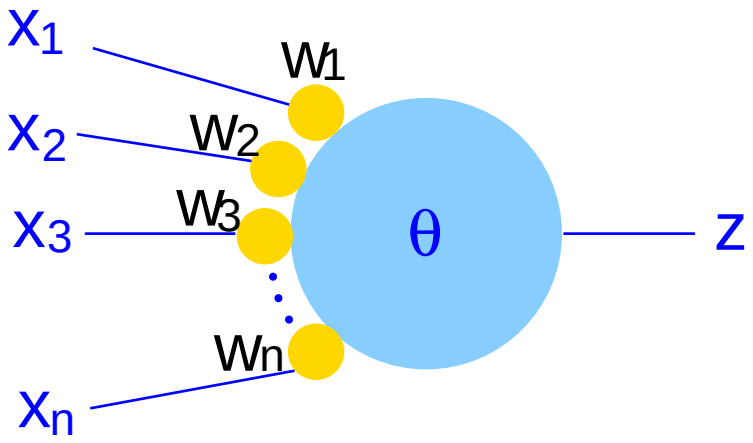
- ビリヤードボールによる演算例

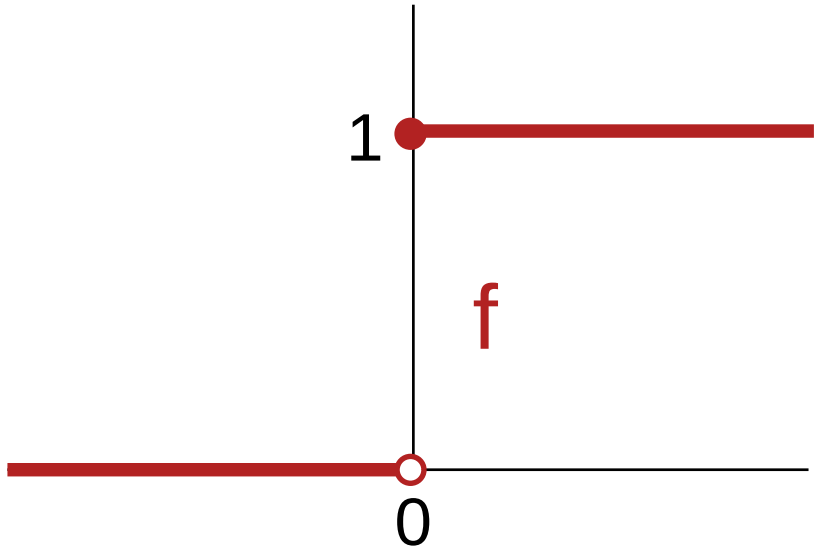


計算とは — 神経回路網による計算

- 神経回路の簡易モデル: しきい値論理

$$z = f(w_1 x_1 + w_2 x_2 + \cdots w_n x_n - \theta)$$



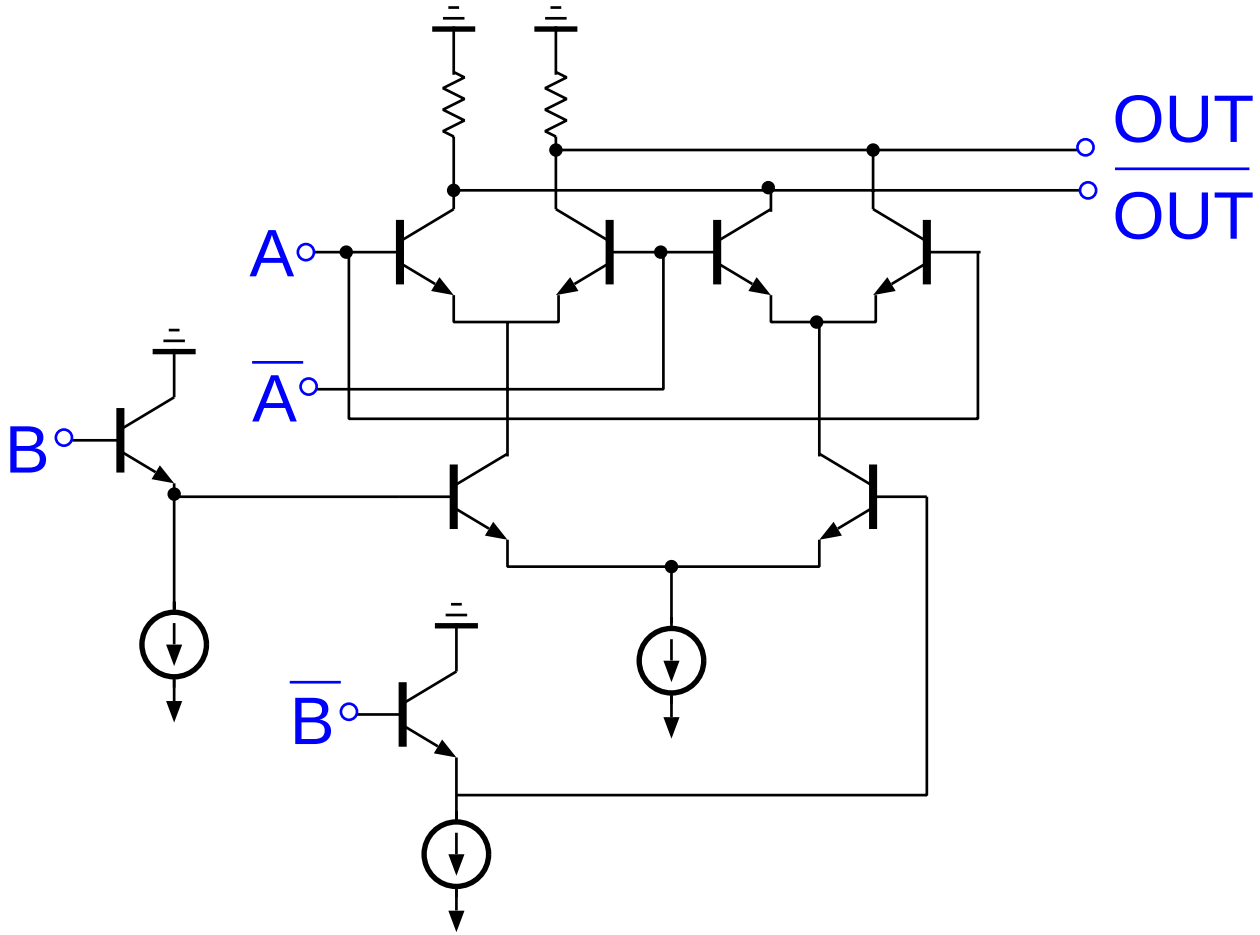


電気回路の特性 — 回路を特徴づけるもの

- 電気回路の要素
  - 電流，電圧，電力，周波数，モード
  - トポロジー
    - キルヒホッフの電圧則
    - キルヒホッフの電流則
    - テレゲンの定理
  - 素子
    - 静的素子，動的素子
    - 線形素子，非線形素子
    - 受動素子，能動素子
    - 相反素子，非相反素子
    - 可逆素子，非可逆素子

電気回路の特性 — 電流に情報をのせる

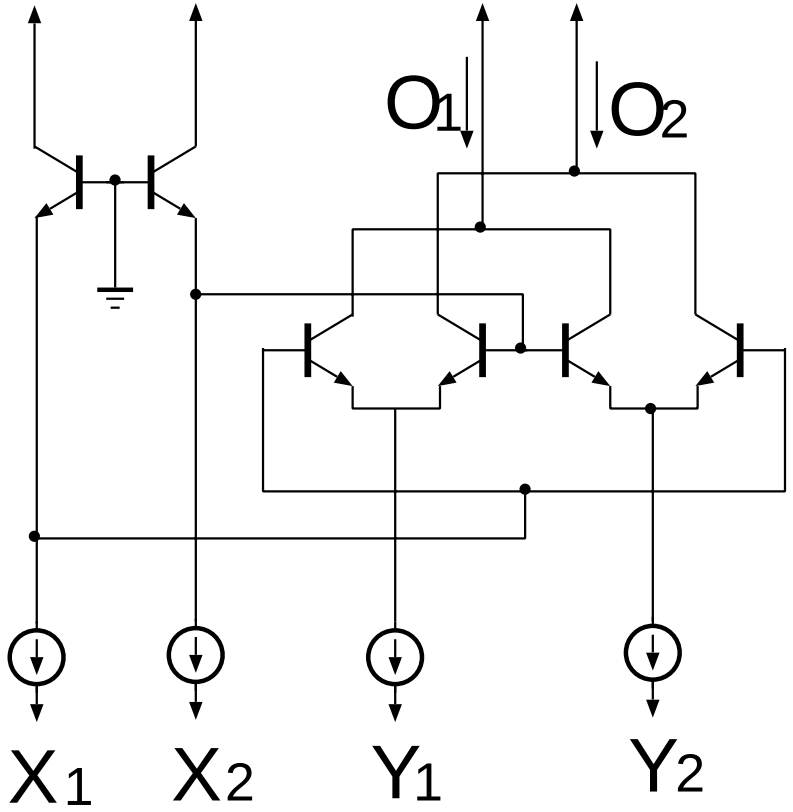
- 電流モード論理 (CML)
  - 電流によって信号を表現した論理回路



CML による ExOR 回路

電気回路の特性 — 非線形性の利用

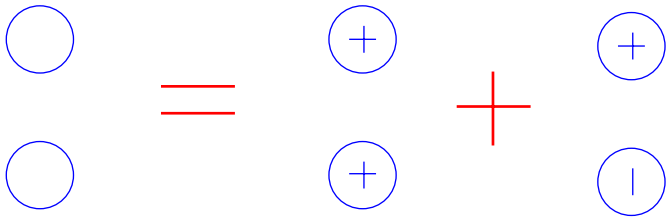
- トランスリニア回路
  - トランジスタの非線形性 (Exp 関数) を利用した回路



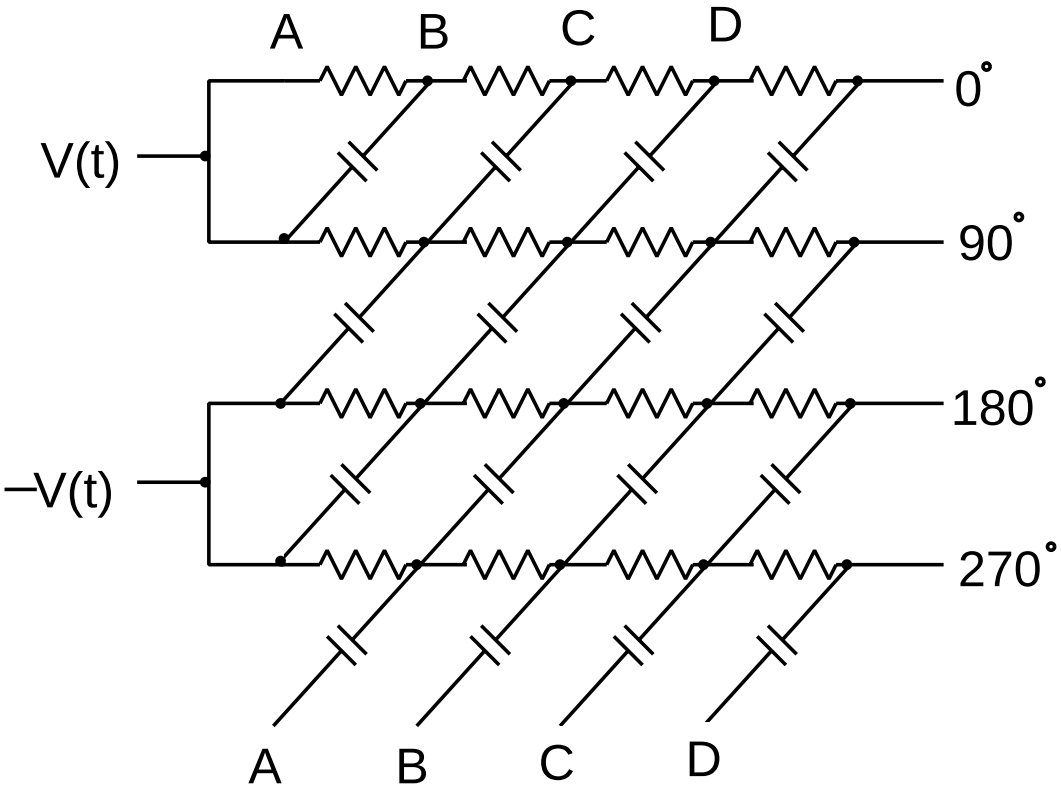
トランスリニア回路による乗算器

電気回路の特性 — モードの利用

- モード: 対称性
- モードにより分割 (TDM, FDM, CDM, SDM, ...)
  - 周波数分割: 各周波数に対応して設計
  - FPGA は時分割
  - 差動モード，コモンモード
  - 位相回転モード



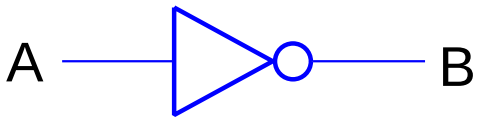
差動モードとコモンモード



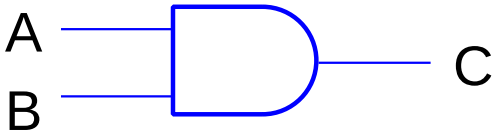
右回りモードと左回りモード

可逆性とは? — 可逆論理

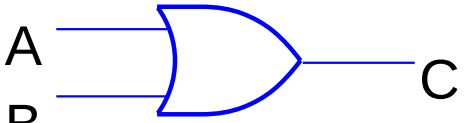
- **論理**の可逆性：出力から入力が分かるか?



NOT



AND



OR

NOT の真理値表

A	B
0	1
1	0

可逆

AND の真理値表

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

非可逆

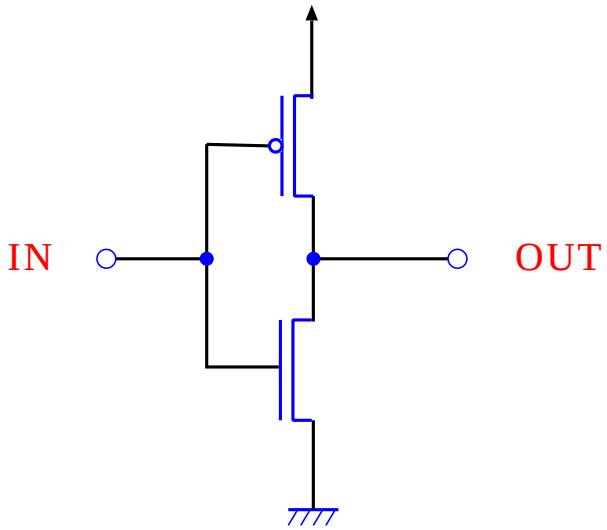
OR の真理値表

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

非可逆

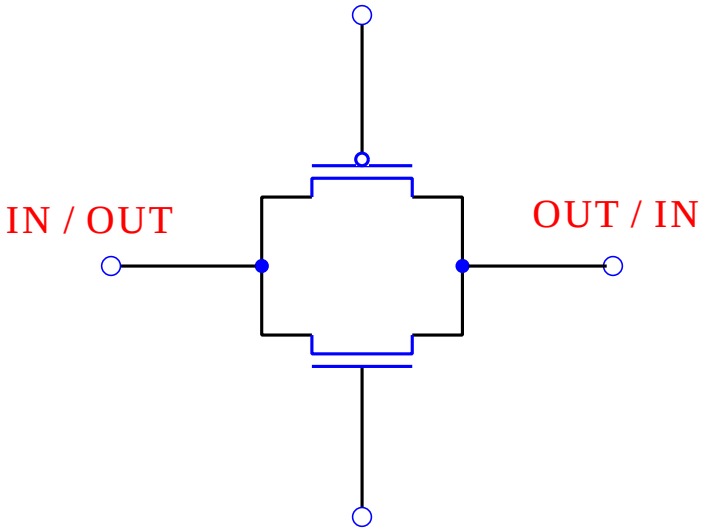
可逆性とは? — 回路の可逆性

- **回路**: 信号の向きを逆にできるか?
  - CMOS インバータ: 非可逆
  - パストランジスタ : 可逆



CMOS inverter

非可逆



Pass-transistor

可逆

可逆性とは? — 算術演算への応用

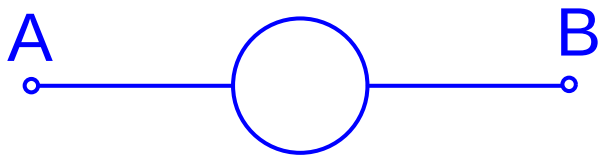
- 加算について

$$a + b = c.$$

  - a と b を与えると, 加算により $c = a + b$ を得る.
  - b と c を与えると, 減算により $a = c - b$ を得る.
  - 加算と減算は可逆演算.
- 乗算と除算も可逆演算
- 加算器，乗算器を設計すれば，減算器，除算器として使える.

可逆論理 — 単純な例

- 1 入力 1 出力

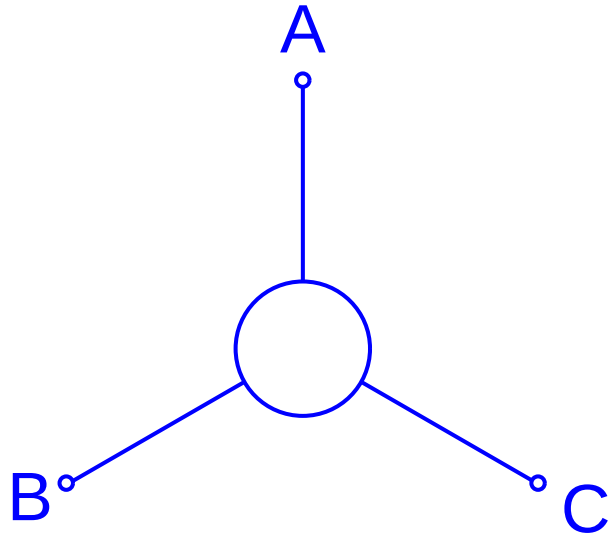


A	B
0	0
1	1

A	B
0	1
1	0

  - 1 入力 1 出力で可能な可逆論理は 2 種類のみ

- 2 入力 1 出力



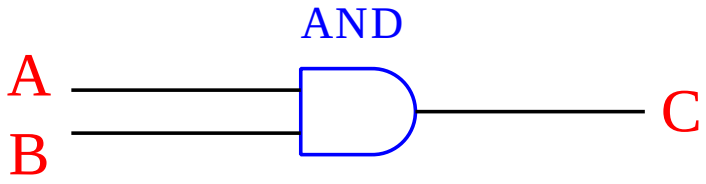
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

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

  - 2 入力 1 出力で可能な可逆な論理は 2 種類のみ

可逆論理 — 可逆性の拡張

- **AND 論理の双方化**



- **真理値表に不定状態 “Z” を導入して双方向化．**

(a)

Input A	Input B	Output C
Z	Z	Z
Z	0	0
Z	1	Z
0	Z	0
0	0	0
0	1	0
1	Z	Z
1	0	0
1	1	1

(b)

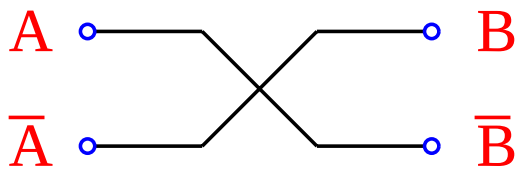
Input A	Input C	Output B
Z	Z	Z
Z	0	Z
Z	1	1
0	Z	Z
0	0	Z
0	1	-
1	Z	Z
1	0	0
1	1	1

(c)

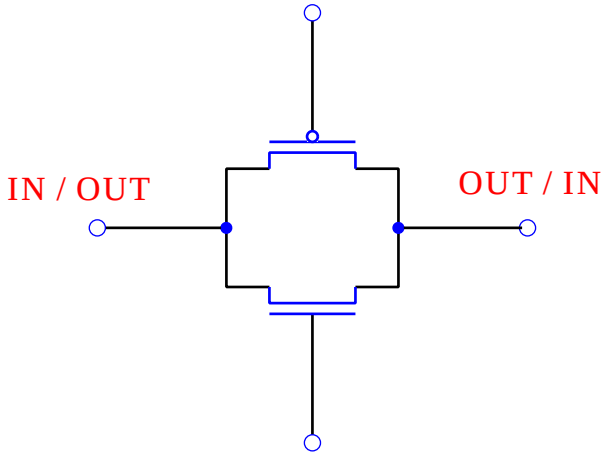
Input B	Input C	Output A
Z	Z	Z
Z	0	Z
Z	1	1
0	Z	Z
0	0	Z
0	1	-
1	Z	Z
1	0	0
1	1	1

可逆回路の設計 — 設計手法

- **2 線式論理:**
  X=1 と X=0 は，$(X,\overline{X})=(1,0)$ と $(X,\overline{X})=(0,1)$ で表現.
- **単調回路:**
  初期値は $(X,\overline{X}) = (0,0)$.
- **パストランジスタ論理:**
  双方向可能なスイッチ



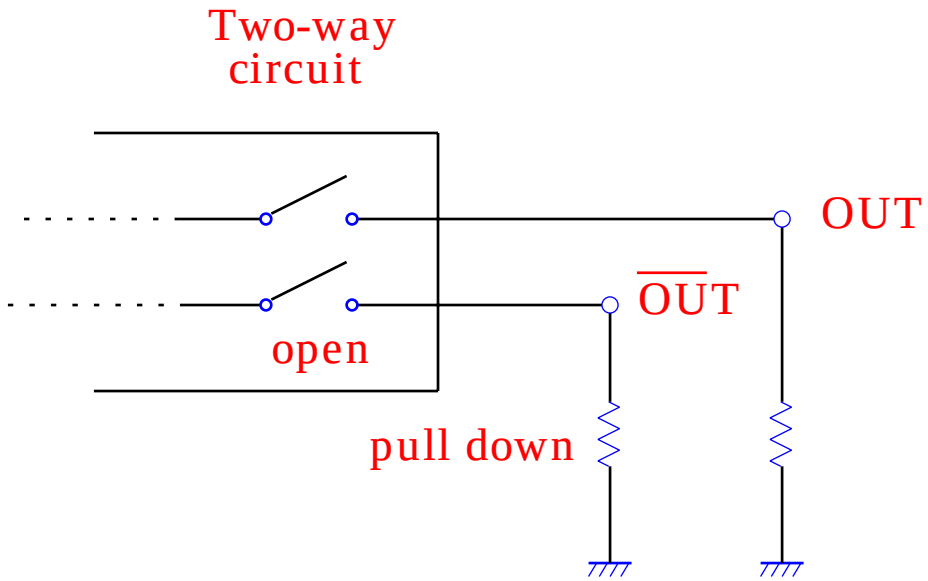
インバータの例



パストランジスタ

可逆回路の設計 — Expression of the unfixed bit

- 不定ビット を ハイインピーダンスで表現.
- プルダウンでハイインピーダンスを検出.



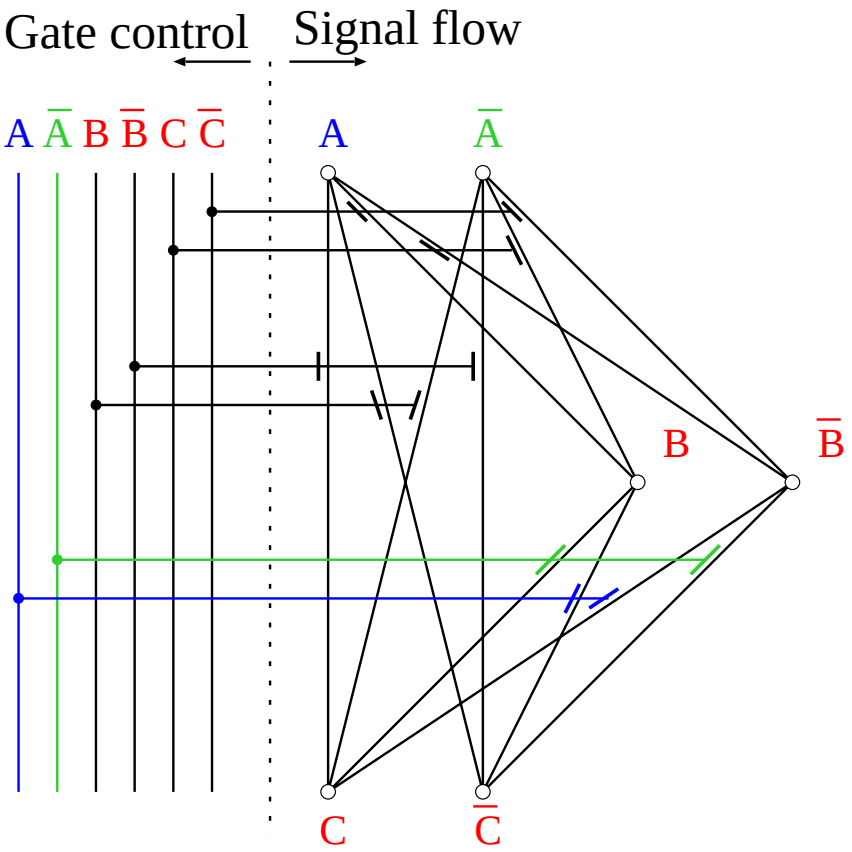
ハイインピーダンスの検出

状態 0, 1, Z に対応する電圧

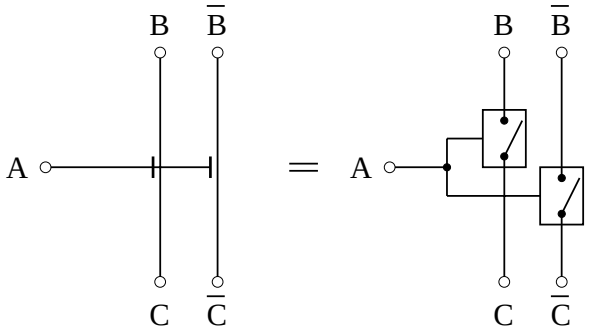
状態	出力電圧 (V)	
	OUT	$\overline{OUT}$
0	0	5
1	5	0
Z	0	0

可逆回路の設計 — 可逆 ExOR 回路

- A =0 のとき，B と C が接続，A=1 のとき B と $\overline{C}$ が接続
- 端子 A, B and C に対して対称な構造．



可逆 ExOR 回路

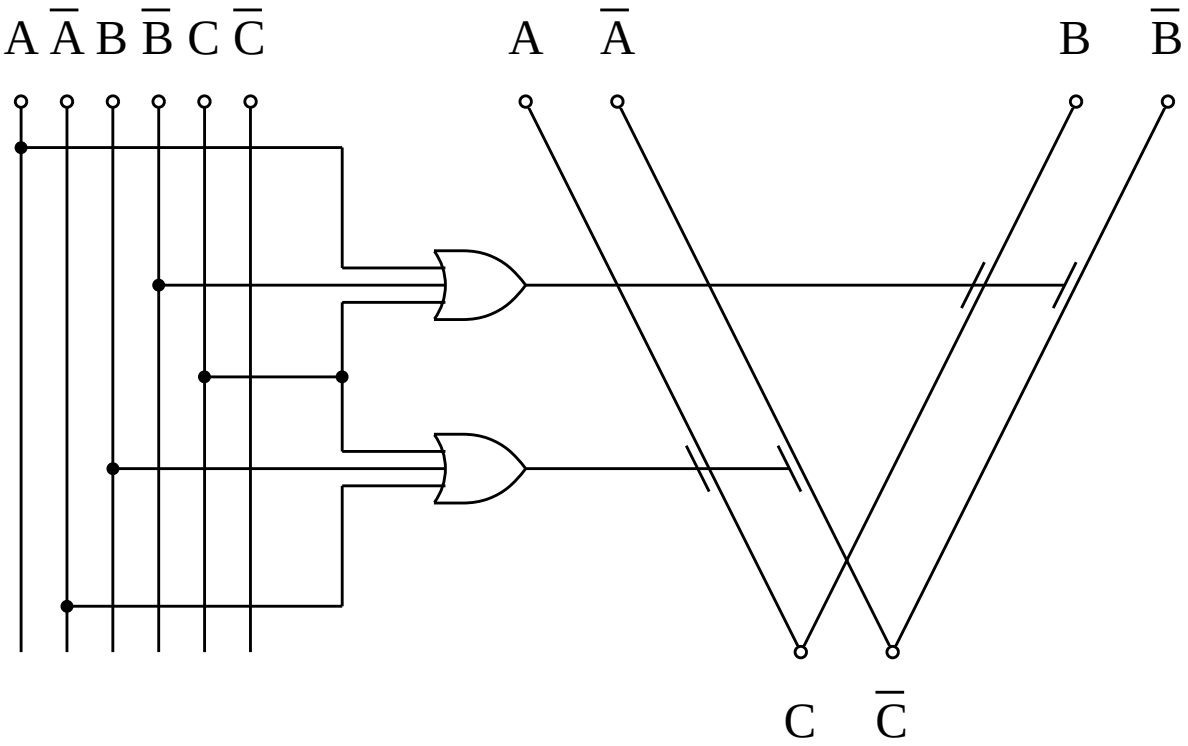


(a) (b)

Input		Input		Output	
A	$\overline{A}$	B	$\overline{B}$	C	$\overline{C}$
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1

可逆回路の設計 — 双方向 AND 回路

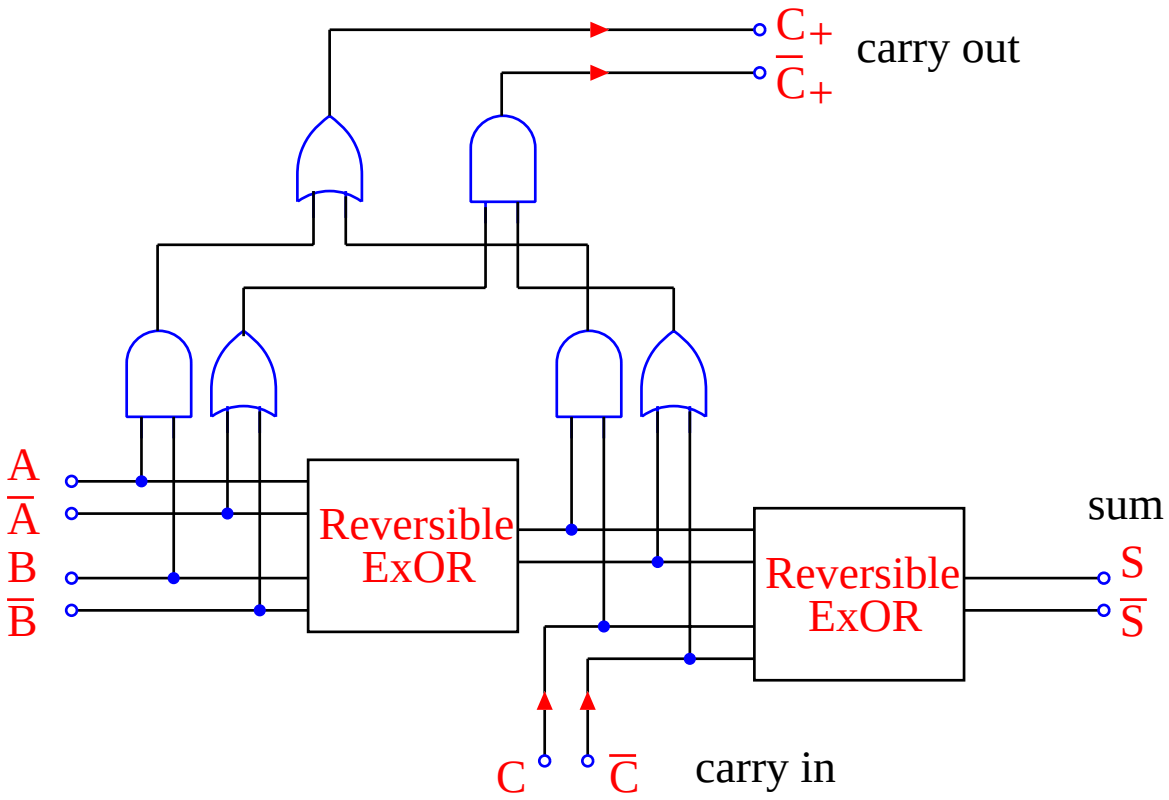
- A = 1 のとき， B と C が接続 A = 0 のとき A と C が接続 C=1, のとき すべての端子が接続
- 端子 A, B に対して対称な構造



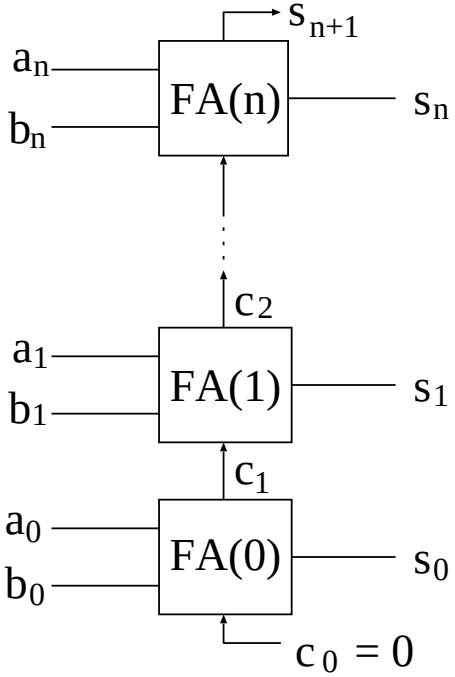
双方向 AND 回路

可逆回路の設計 — 加算器

- 可逆全加算器の動作
  - A, B, C が入力の時, 出力は S and C_+.
  - A, C, S が入力の時, 出力は B.



可逆全加算器

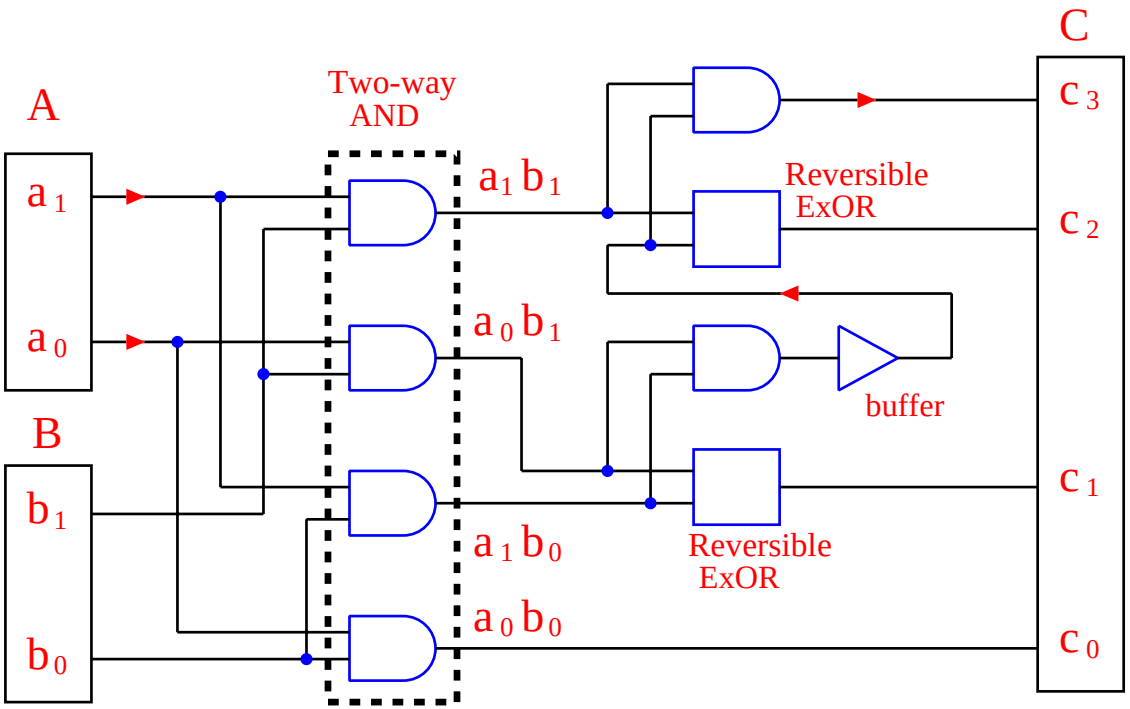


n-bit 可逆加算器

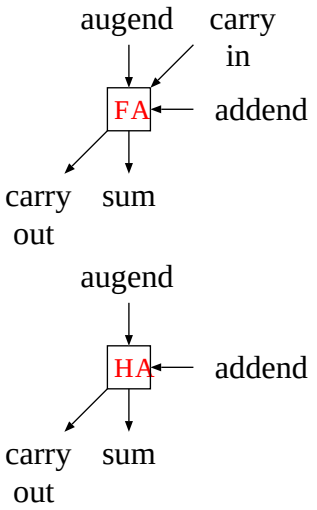
可逆回路の設計 — 可逆乗算器

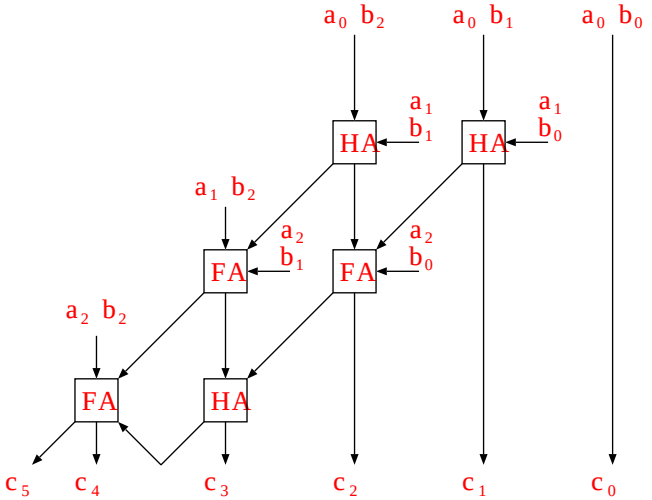
2×2bit 可逆乗算器

$$c_0 = a_0 b_0, c_1 = a_1 b_0 \oplus a_0 b_1, c_2 = a_1 b_1 \oplus (a_1 b_0 \cdot a_0 b_1), c_3 = a_1 b_1 \cdot a_1 b_0 \cdot a_0 b_1.$$



2×2 bit 乗算器





3×3 bit 乗算器

正論理のみ表示

まとめと課題

- まとめ
  - アルゴリズムのハード化は，問題からアルゴリズムを考え，ハードウェアにおいて実行するための枠組みである．
  - 広い枠組みで考えることにより，アルゴリズムをハード化する上で多様なアプローチができる．
- 課題
  - 今回の講義内容と自分の専門分野を比較し，講義内容との関係，発想や方法論の共通点及び相違点を論じよ．