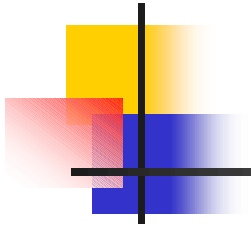


レースによる誤動作

複数の変数 ⇒ 同時変化，競合

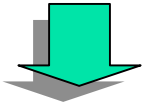
遅延のばらつき ⇒ 遷移先不定

クリティカルレース

クリティカルレースの除去

状態割り当ての工夫

（例） 状態⇒ ${}_nC_1$ 符号を割り当て



1ビットのみ 1
$(n\text{-}1)$ビット 0

クリティカルレースの除去

＜手順1a＞ 符号の割当て

δ

Q \ X	X_0	X_1	X_2	X_3
Q_0	Q_0	Q_1	Q_1	Q_0
Q_1	Q_2	Q_1	Q_1	Q_1
Q_2	Q_2	Q_3	Q_2	Q_0
Q_3	Q_0	Q_3	Q_2	Q_0

$_4C_1$ 符号

状態	q_1	q_2	q_3	q_4
Q_0	1	0	0	0
Q_1	0	1	0	0
Q_2	0	0	1	0
Q_3	0	0	0	1

クリティカルレースの除去

＜手順1b＞ 符号の割当て

δ

Q \ X	X_0	X_1	X_2	X_3
Q_0	Q_0	Q_1	Q_1	Q_0
Q_1	Q_2	Q_1	Q_1	Q_1
Q_2	Q_2	Q_3	Q_2	Q_0
Q_3	Q_0	Q_3	Q_2	Q_0

2値符号

入力	x_1	x_2
X_0	0	0
X_1	0	1
X_2	1	1
X_3	1	0

クリティカルレースの除去

＜手順２＞ 遷移の分割
（1変数のみ変化）

状態	q_1	q_2	q_3	q_4
Q_0	1	0	0	0
Q_1	0	1	0	0
Q_2	0	0	1	0
Q_3	0	0	0	1

$(0,0,1,0) \rightarrow (1,0,0,0)$

$(0,0,1,0) \rightarrow (1,0,1,0)$

$\rightarrow (1,0,0,0)$

クリティカルレースの除去

＜手順3a＞ 遷移後にq_i=1となる起点マス目に1記入

$$(x_1,\cdots,x_k,q_1,\cdots,q_n)$$

（例）q_1に対する状態遷移

$$\delta(X_3,Q_0)=Q_0 \quad \delta(1,0,1,0,0,0)=(1,0,0,0)$$

$$\delta(X_3,Q_2)=Q_0 \quad \delta(1,0,0,0,1,0)=(1,0,1,0)$$

$$\rightarrow \delta(1,0,1,0,1,0)=(1,0,0,0)$$

クリティカルレースの除去

＜手順3b＞ 遷移後にq_i=1となる起点マス目に1記入

$$(x_1,\cdots,x_k,q_1,\cdots,q_n)$$

（例）q_1に対する状態遷移

$$\delta(X_1,Q_0)=Q_1$$

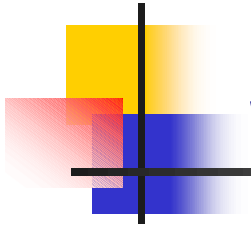
$$\delta(0,1,1,0,0,0)=(1,1,0,0)$$

$$\rightarrow \delta(0,1,1,1,0,0)=(0,1,0,0)$$

クリティカルレースの除去

＜手順４＞　論理回路の作成

q_iに慣性遅延 Δ_i 挿入



非同期動作の応用

- アービタ(Arbiter)

（調停者，仲介者）

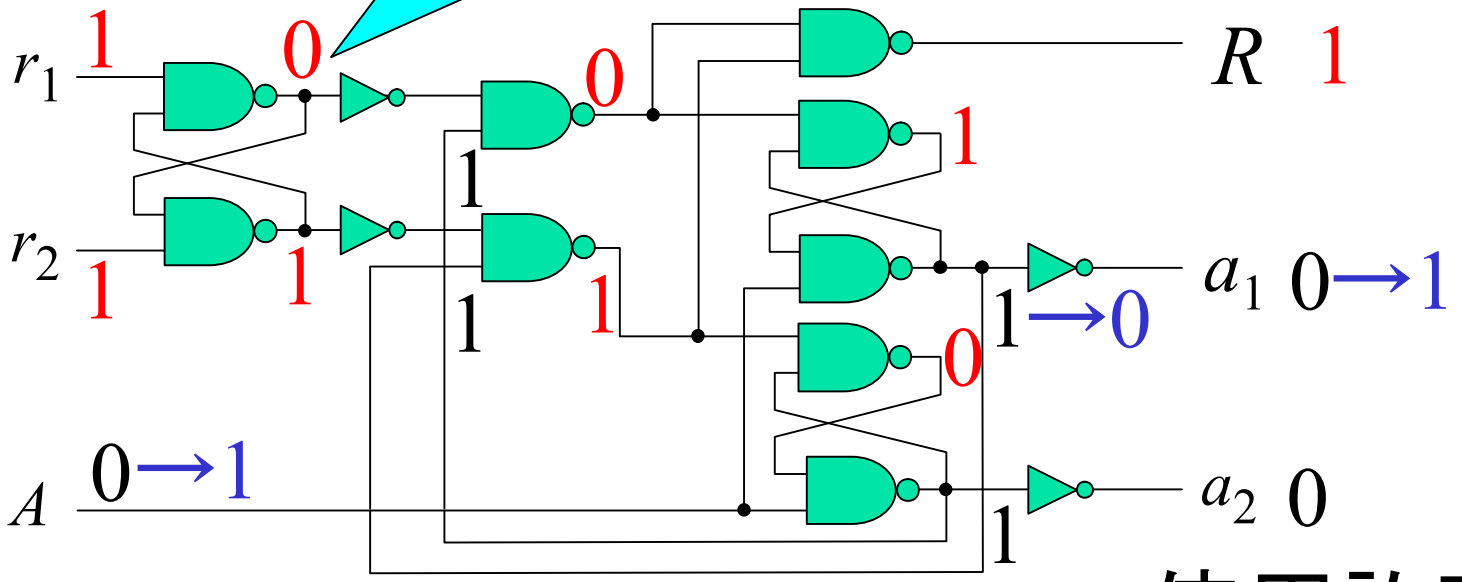
信号競合⇒優先順位

アービタ

プロセッサからの要求

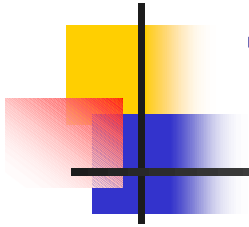
クリティカルレース

メモリへの要求



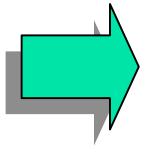
メモリからの使用許可

使用許可



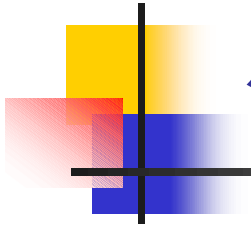
論理設計の目標

- 開発期間短縮
- コスト低減



設計の
自動化

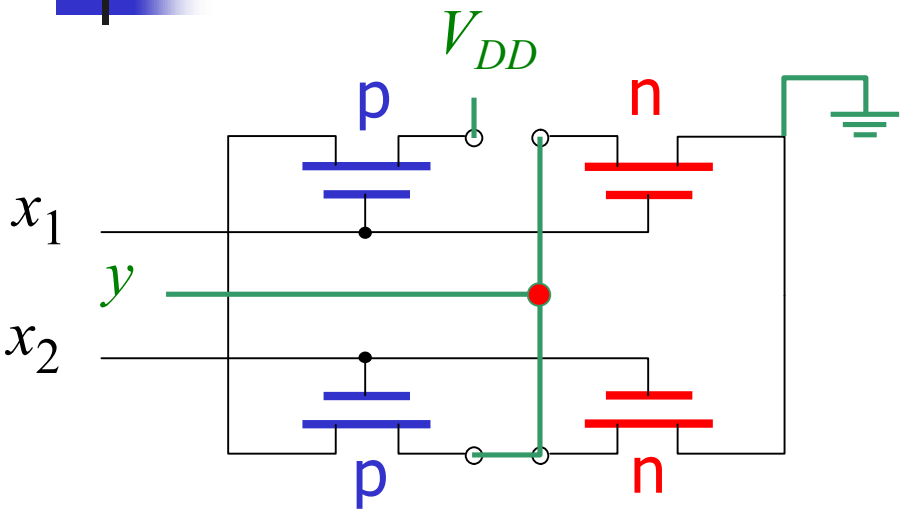
ゲートアレイ方式
スタンダードセル方式
ROM (Read Only Memory)
PLA (Programmable Logic Array)
シストリックアレイ



ゲートアレイ方式

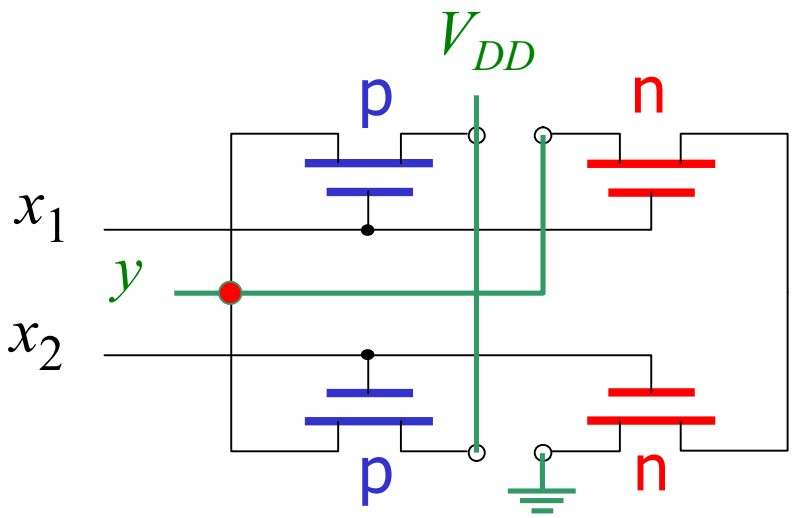
- ゲートアレイ
  - 2次元配置した論理ゲート
  - 事前に量産
- 配線
  - 機能実現

CMOSゲートアレイ



NOR

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0

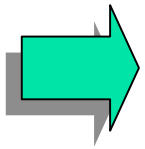


NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

スタンダードセル方式

- 基本セル
  - 論理ゲート，フリップフロップ，デコーダ，加算器，比較器，カウンタなど
- マクロセル
  - 基本セルの組合せ



ライブラリ豊富

スタンダードセル方式

＜長所＞

設計の自由度大

高密度集積化

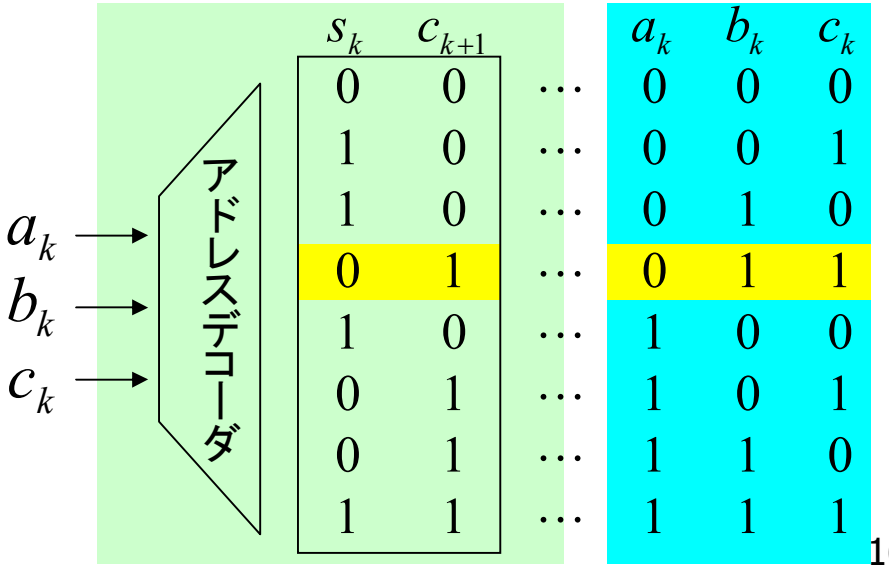
＜短所＞

ゲートアレイ方式よりも
開発期間長，コスト高

ROM (Read Only Memory)

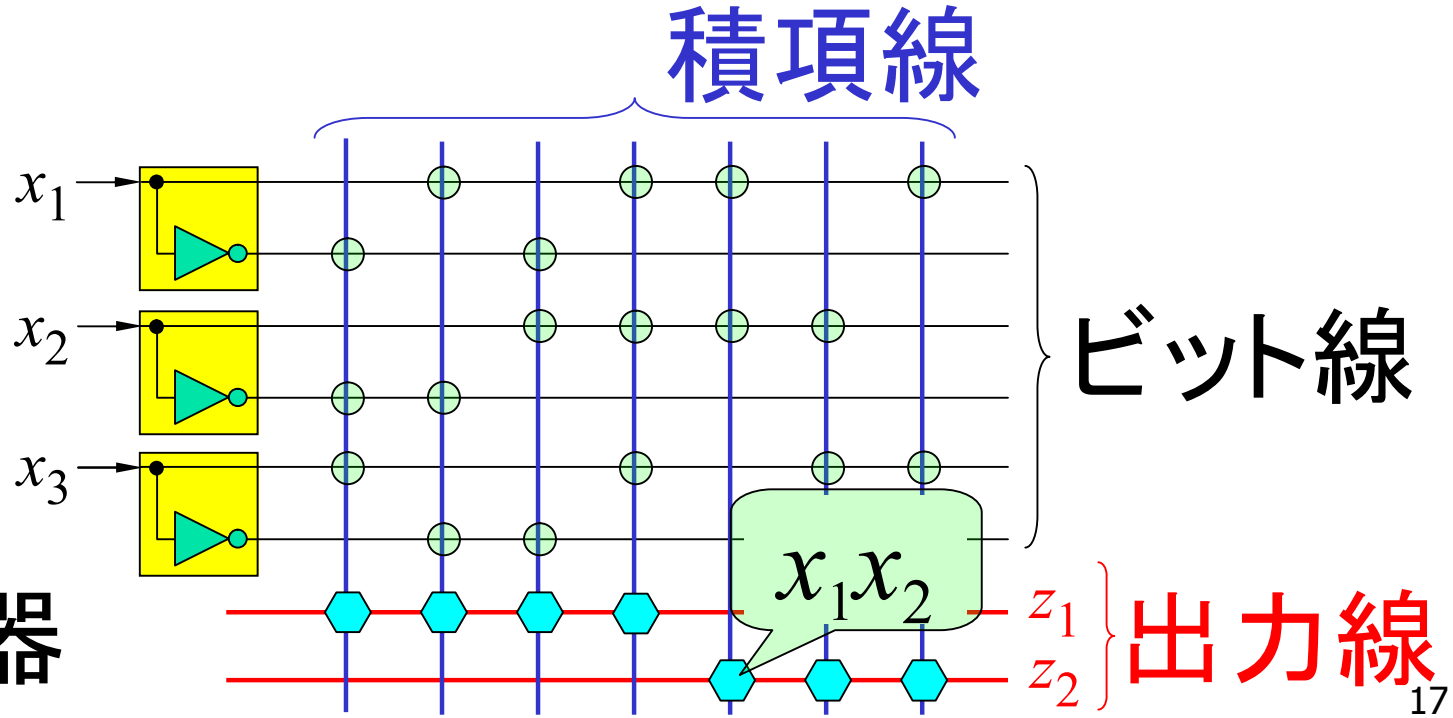
- 単純で規則的な構造
  - 超高密度集積化
- アドレスに関数値を記憶
  - 演算無し

（例）加算器



PLA (Programmable Logic Array)

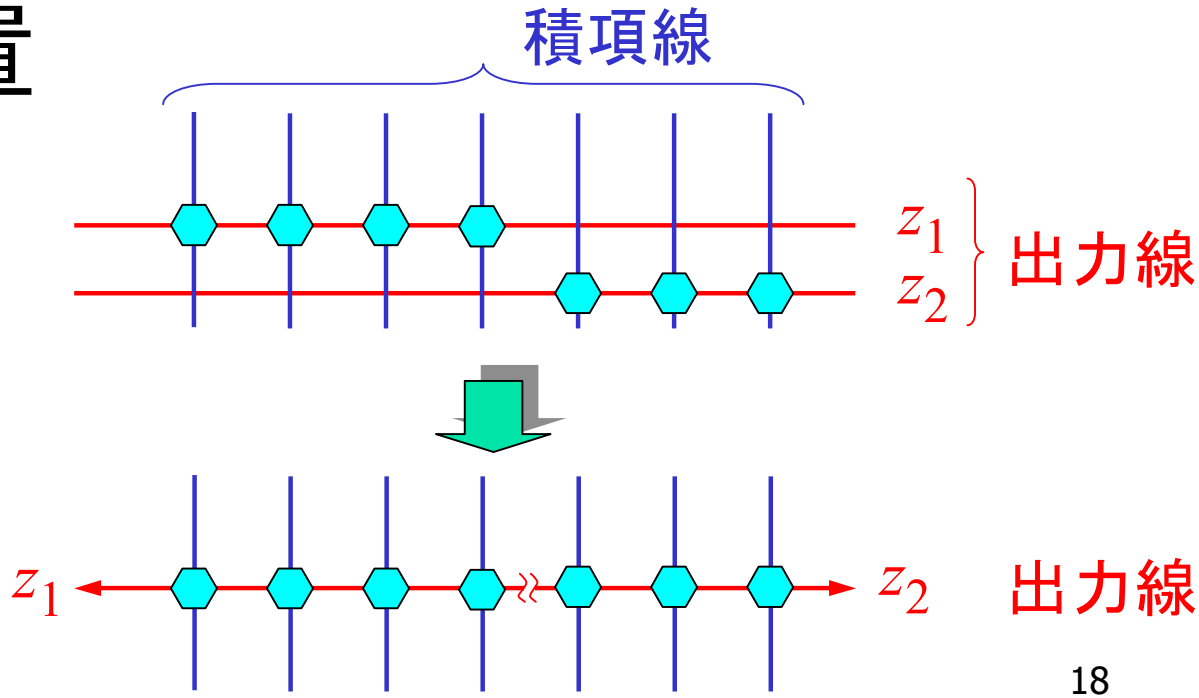
- AND-OR2段論理式を実現
- デコーダ，ANDアレイ，ORアレイ



（例）
加算器

PLAの最適設計

- 畳込み法
  - 出力線／ビット線(積項線非共有)対向配置

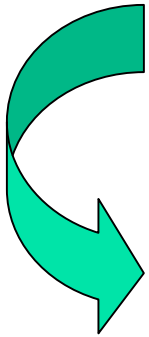


PLAの最適設計

- 出力位相割当て法
  - 出力の積項
    - できるだけ多く共通化

$$z_1 = \bar{x}_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + x_1x_2x_3$$

$$z_2 = x_1x_2 + x_2x_3 + x_1x_3$$

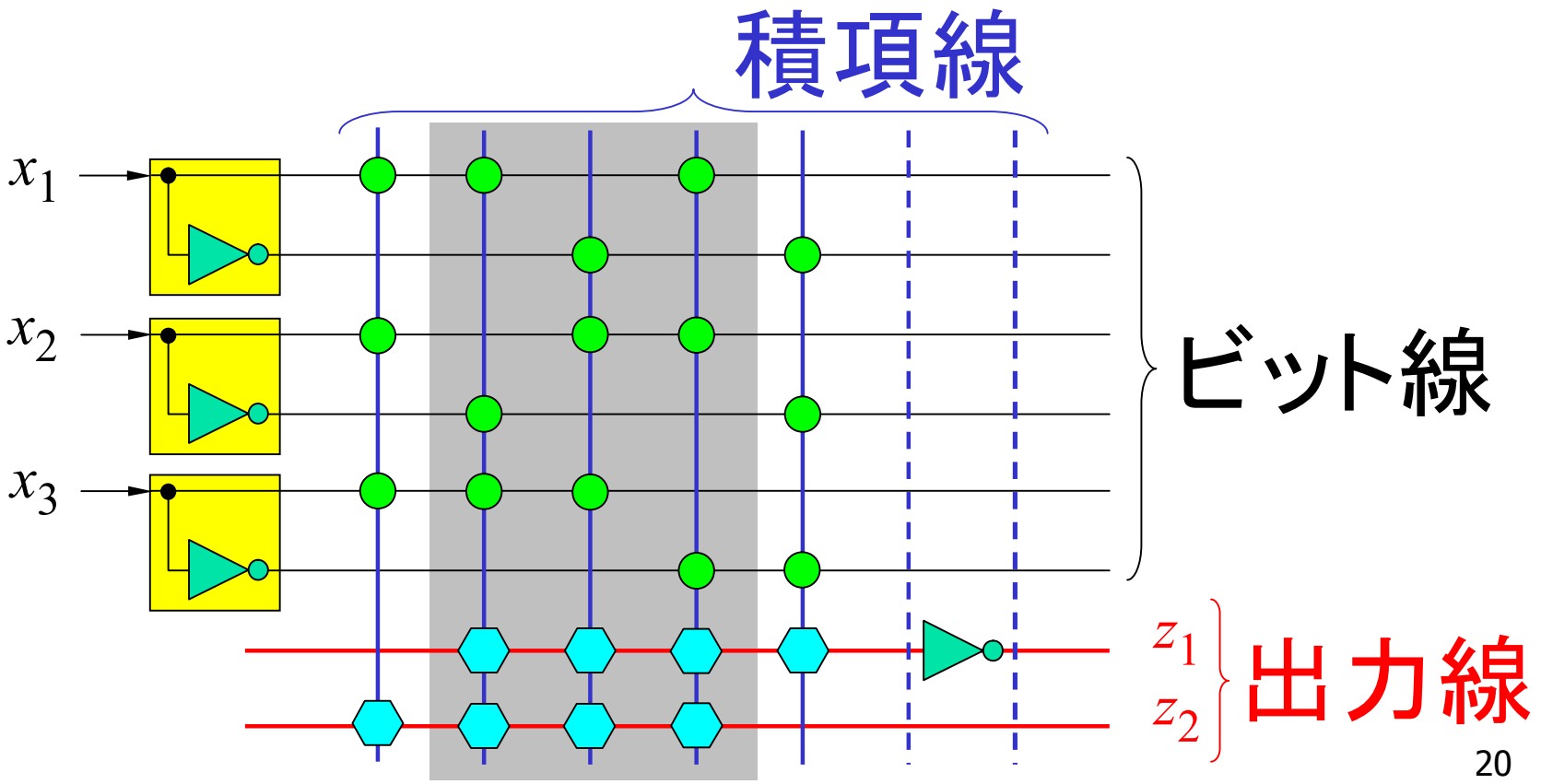


$$\bar{z}_1 = x_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + \bar{x}_1\bar{x}_2\bar{x}_3$$

$$z_2 = x_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2x_3$$

PLAの最適設計

- 出力位相割当て法



PLAの最適設計

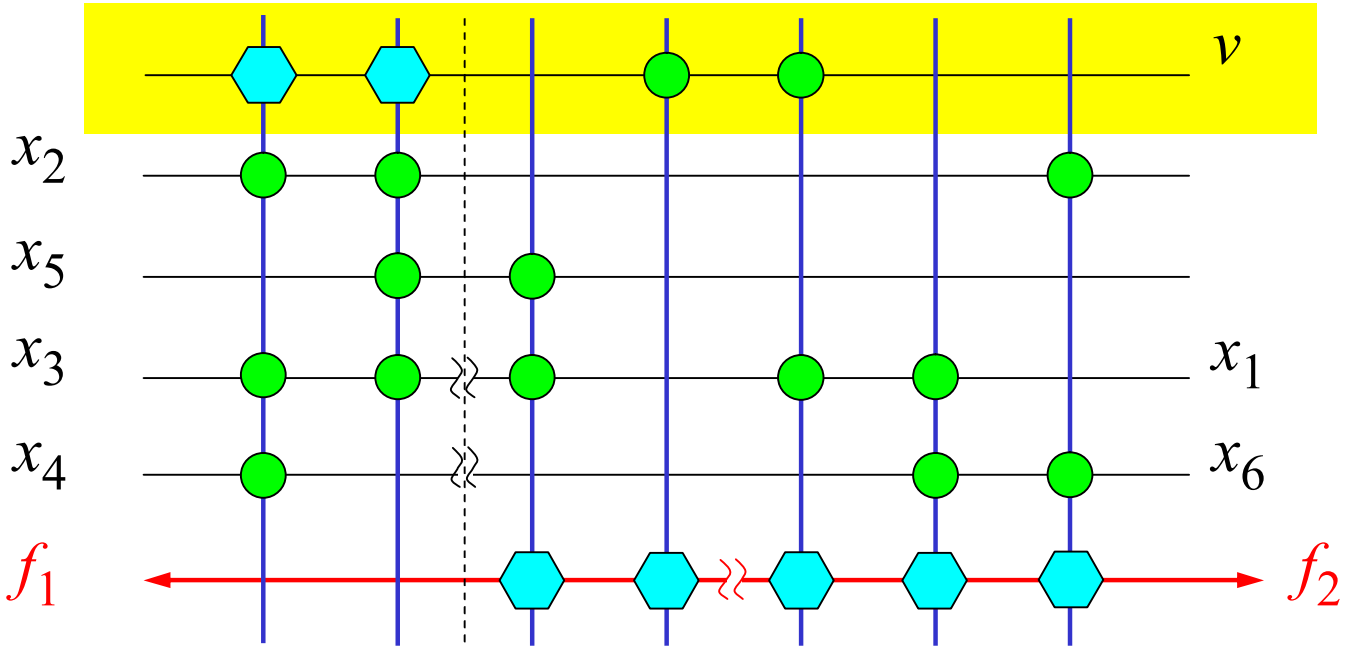
- 分割法
  - 共通な部分式をPLAで実現

$$f_1 = x_1x_5 + x_2x_3x_4 + x_2x_3x_5$$
$$f_2 = x_1x_2x_3x_4 + x_1x_2x_3x_5 + x_1x_6 + x_2x_6$$

$$f_1 = x_1x_5 + v$$
$$f_2 = x_1v + x_1x_6 + x_2x_6$$
$$v = x_2x_3(x_4 + x_5)$$

PLAの最適設計

- 分割法
  - 共通な部分式をPLAで実現

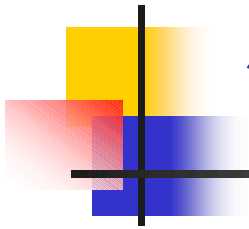


VLSI向き論理設計

- VLSIに適したアルゴリズム

アル・フワリズミー
（780-850頃，アラビアの数学者）

  - 繰り返し構造，規則的配列
  - 並列処理，パイプライン処理
  - データと制御のフロー
    - 簡単，規則的

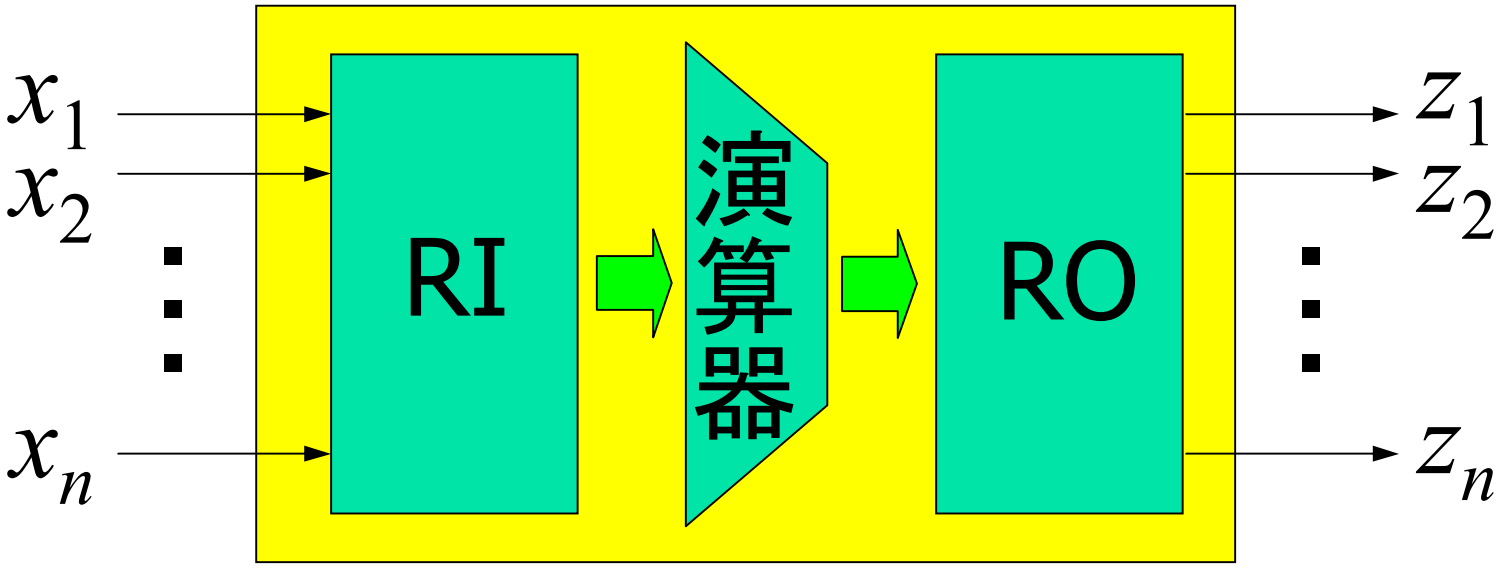


シストリックアレイ

- 単純な機能をもつプロセッサ
  - 1次元または2次元に配置
- データ:クロックに同期
  - 循環する間に計算
  - 複数のデータ
    - 複数の方向に流れる

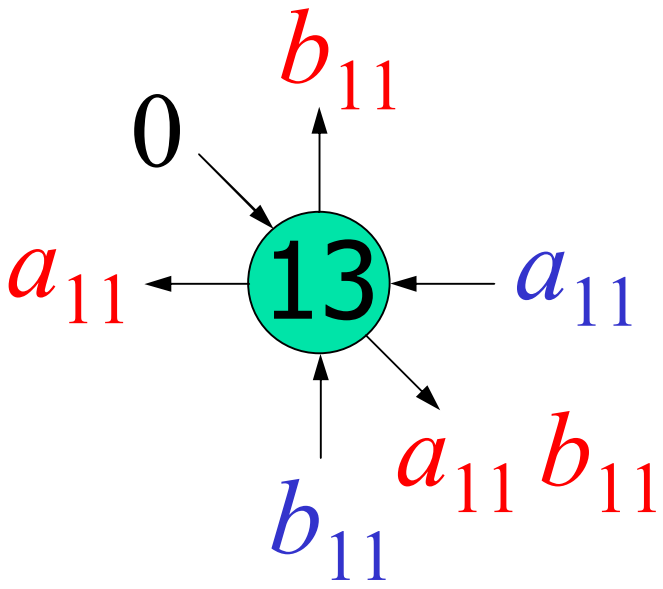
シストリックアレイ

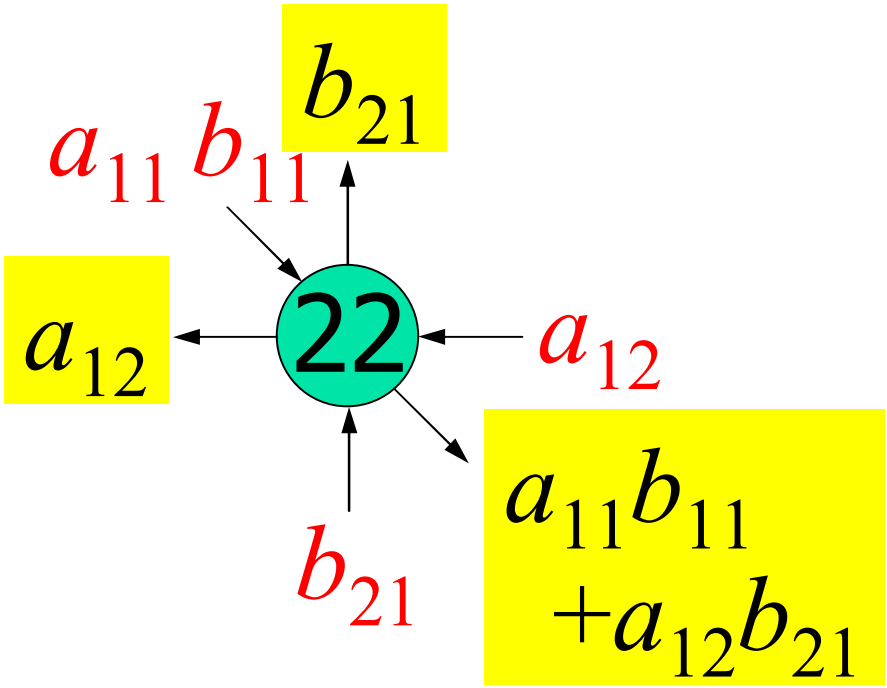
- プロセッサ



シストリックアレイ

演算サイクル $k=5 \Rightarrow k=6 \Rightarrow k=7$



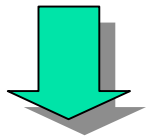


テスト容易化技術

- スキャンパス法

回路内のフリップフロップ

鎖状に連結 ⇒シフトレジスタ



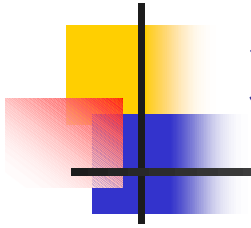
フリップフロップのテスト

テスト容易化技術

- 組み込み自己テスト法

チップ内

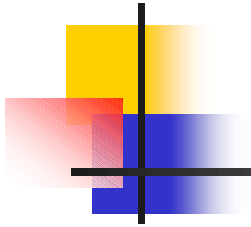
入力発生部
出力判定部 ⇒ 動作テスト



符号化による誤り検出

機能モジュールの出力
⇒符号化

符号語の有無を監視
⇒故障を自己検出



多重化による誤りマスク

故障時でも
正しい出力を出す

同一の論理信号を多重化
⇒多数決回路を利用