デジタルシステム設計
および演習

2020年度　第2回

先週の宿題

- 演習1:
  - 閾値が、2.0V±0.1V、1.8V±0.15V、2.3V±0.1V、2.2V±0.2V、1.6V±0.05V の5つ素子でディジタル回路を組みたい。
  - H（High）およびL（Low）の出力レベルをそれぞれ 5V, 0.2V とした時、H側およびL側のノイズ・マージンはそれぞれ幾らか？
- 解答1:

  各素子の閾値の範囲は、1.9V～2.1V、1.65V～1.95V、2.2V～2.4V、2.0V～2.4V、1.55V～1.65V となる。

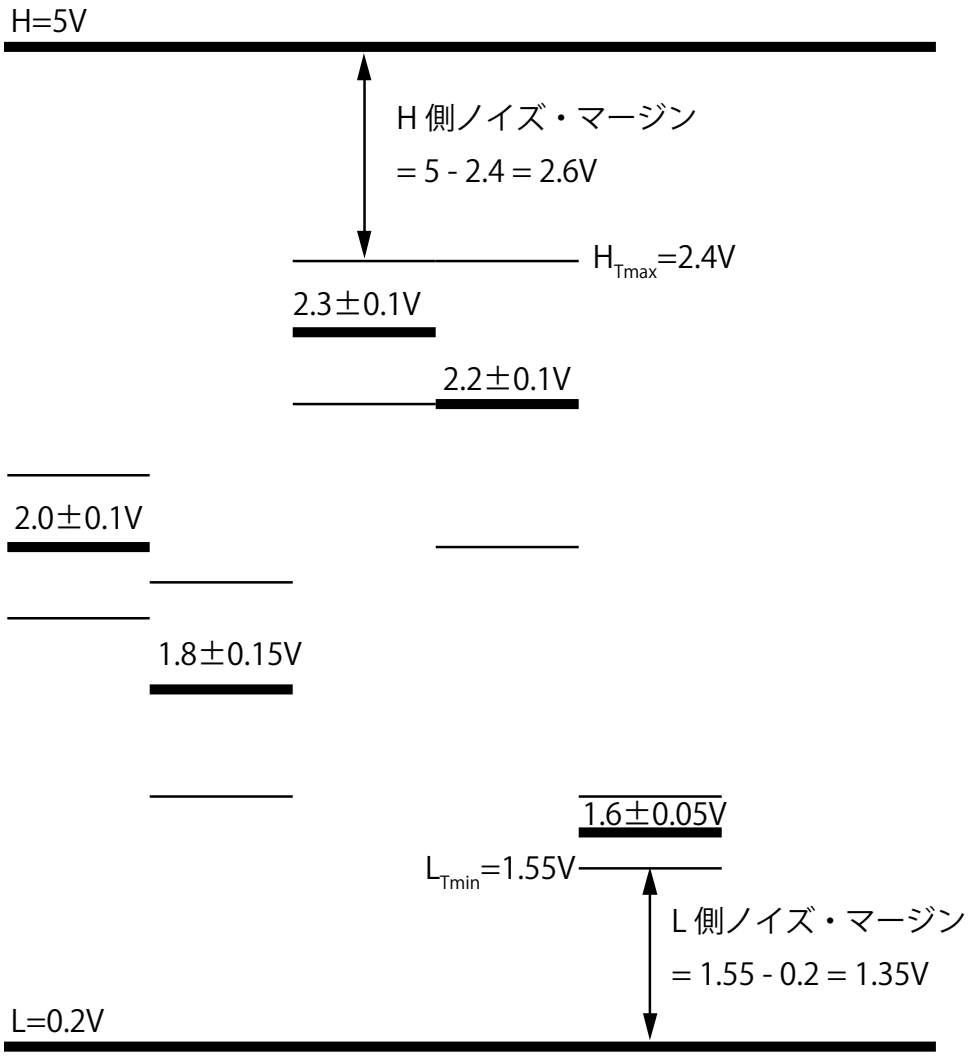
  H側の最大値は2.4V, L側の最小値は 1.55V。故に、

  H側のノイズ・マージンは　5V－2.4V＝2.6V

  L側のノイズ・マージンは 1.55V－0.2V＝1.35V

先週の宿題

- **演習1:**


H=5V
H 側ノイズ・マージン
= 5 - 2.4 = 2.6V
H_{Tmax}=2.4V
2.3±0.1V
2.2±0.1V
2.0±0.1V
1.8±0.15V
1.6±0.05V
L_{Tmin}=1.55V
L 側ノイズ・マージン
= 1.55 - 0.2 = 1.35V
L=0.2V

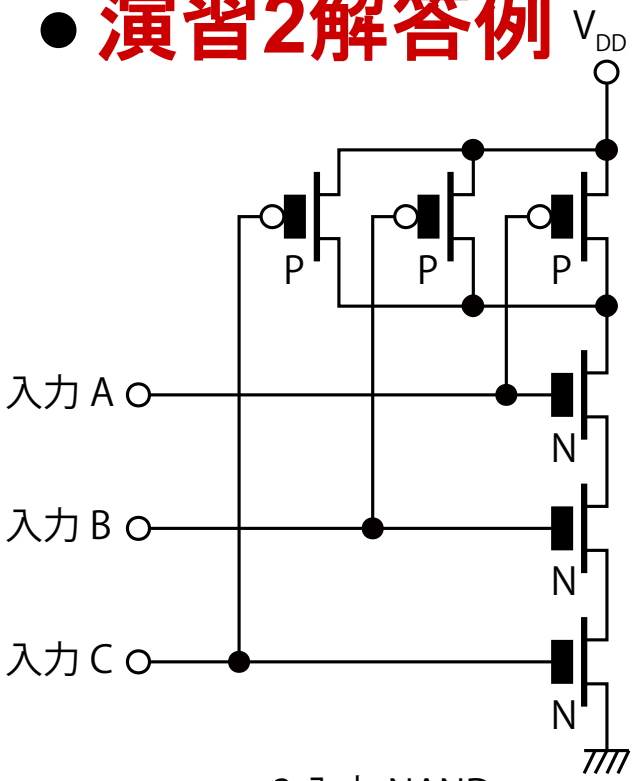
実際の素子の閾値の例

- TTL（駆動電圧V_{CC} = 5[V]固定）
  - $V_{TH} \approx 2.0$[V]
  - $V_{TL} \approx 0.8$[V]
- CMOS（駆動電圧V_{DD} = 5[V]の場合）
  - $V_{TH} \approx 0.5 \sim 0.7 \cdot V_{DD} = 2.5 \sim 3.5$[V]
  - $V_{TL} \approx 0.8 \sim 1.5$[V]

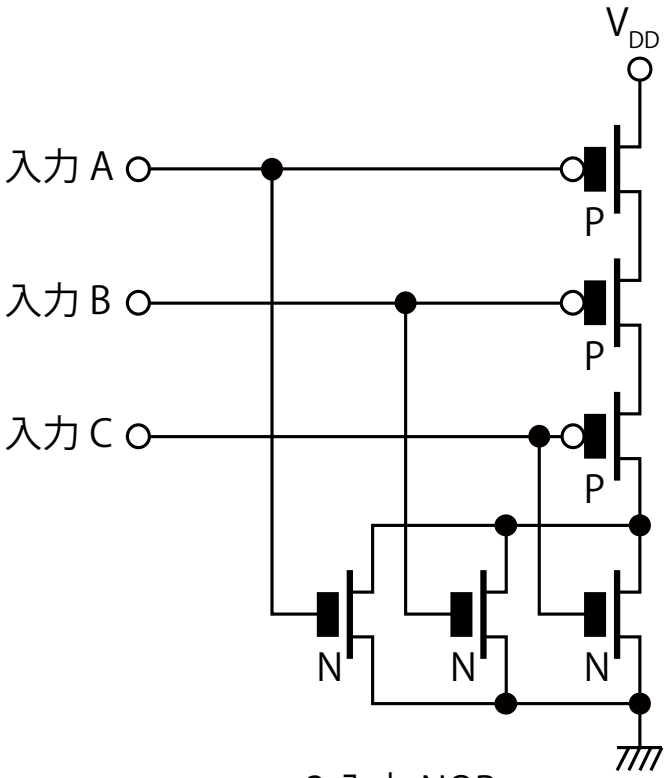
TTLとCMOSを混在させた場合の問題点を考察しなさい。また対策を述べなさい。

先週の宿題

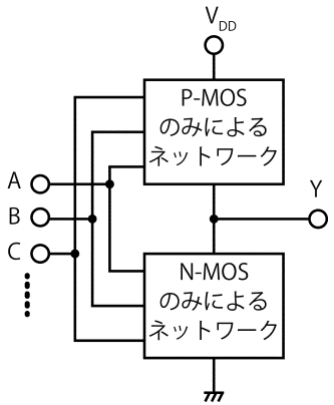
- 演習2解答例



3 入力 NAND

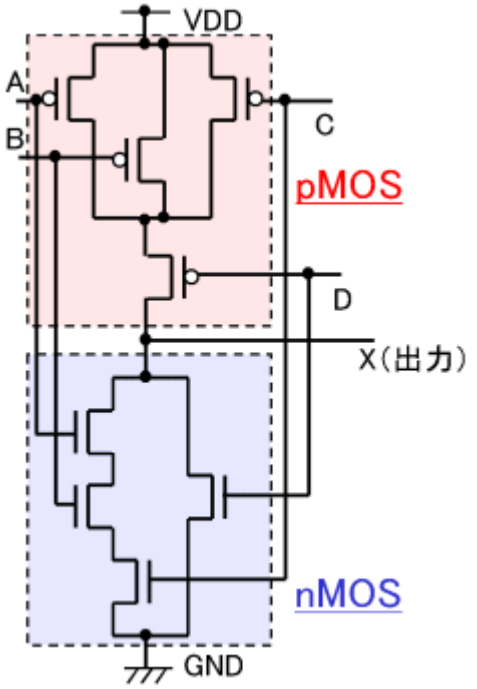


3 入力 NOR



CMOS回路の一般形

$$X = \overline{(A \cdot B \cdot C) + D}$$



実際の3入力NAND, NOR

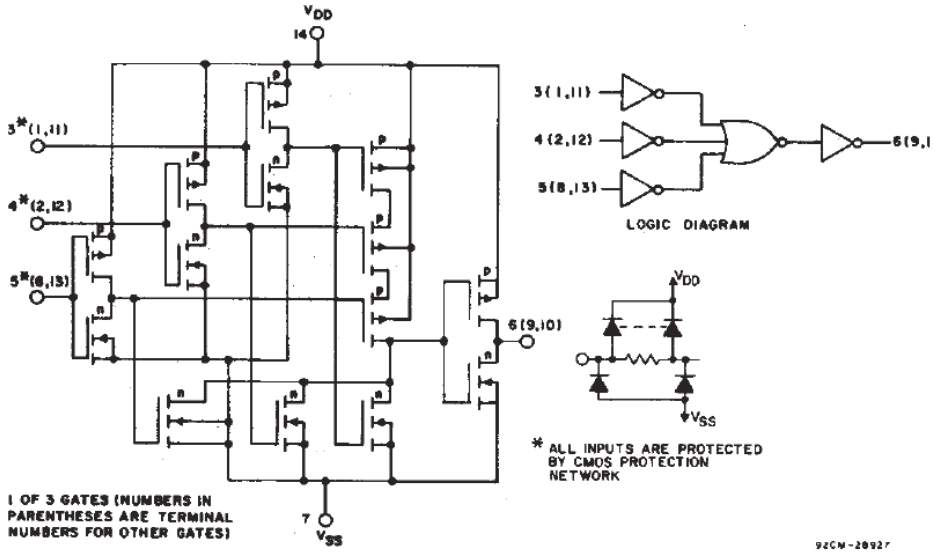


Fig. 9 – Schematic and logic diagrams for CD4023B.

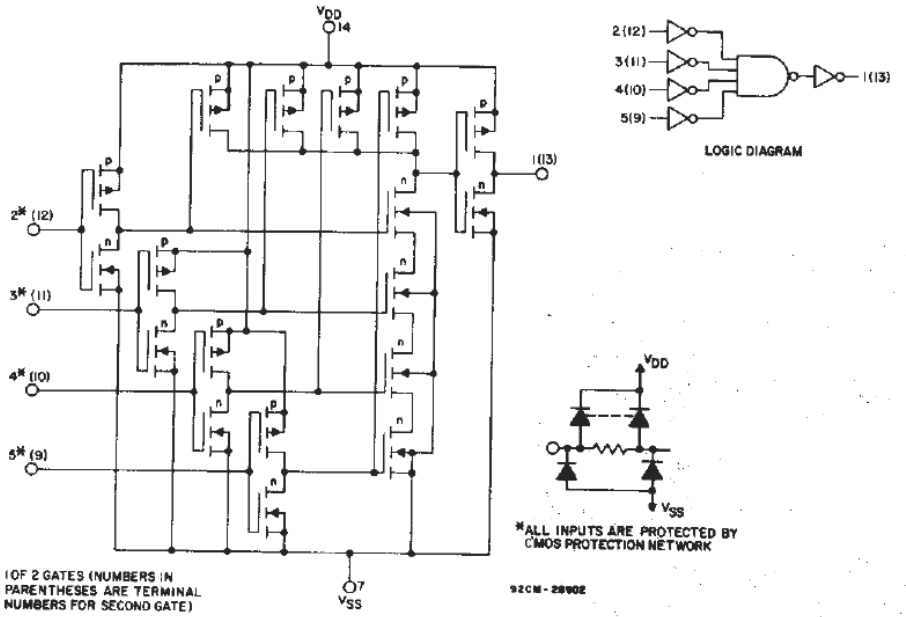


Fig. 6 – Schematic and logic diagrams for CD4002B.

実際の8入力NAND

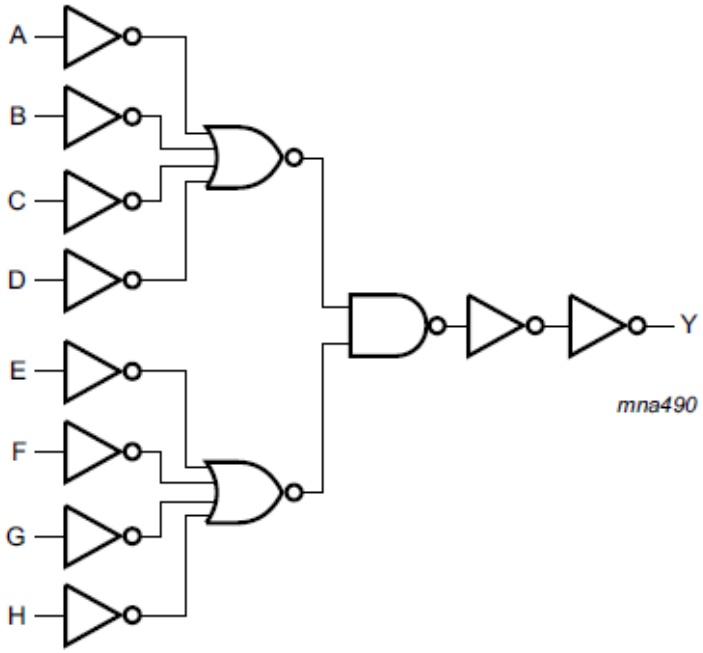


Fig 3. Logic diagram

先週の宿題

ダイヤモンド社書籍オンライン　http://diamond.jp/list/books
2012年9月11日「日の丸半導体」復活に向けての処方箋より

半導体産業の業態

業態名	特　徴	代表的な企業
IDM	設計から生産まで一貫して自社内で行なう	(日) ルネサス、エルピーダ、東芝 (米) インテル (韓) サムスン (欧) ST マイクロエレクトロニクス
ファウンドリ	設計は行わずもっぱら製造を請け負う	(台) TSMC、UMC (中) SMIC
ファブライト	製造装置を保持しつつ、一部を外部委託する	(日) 富士通マイクロエレクトロニクス (米) TI 社
ファブレス	設計に特化し、生産は100%外部委託する	(米) QUALCOM、AMD、NVIDIA、Broadcom (台) Media Tek

代表的な製品とメーカーおよび業態の特徴

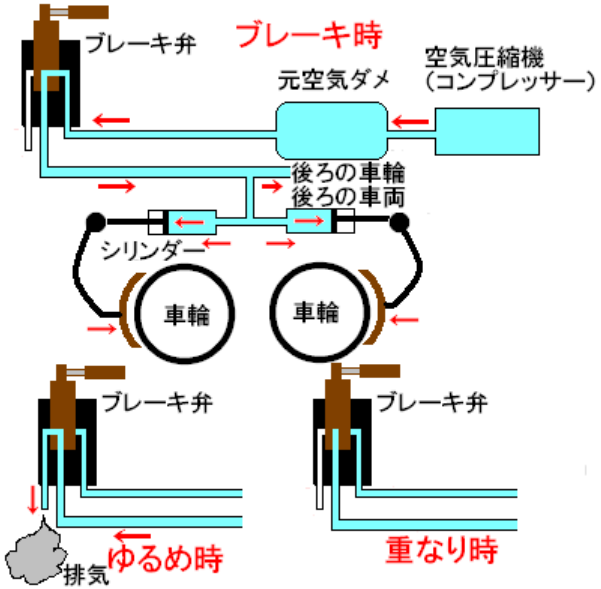
業態	主要IC	代表的なメーカー	付加価値の源泉
IDM	MPU	(米) インテル	機能 (What)
IDM	メモリ	(韓) サムスン	プロセス (how)
ファウンドリ	ロジック系	(台) TSMC	生産システム

本日の内容

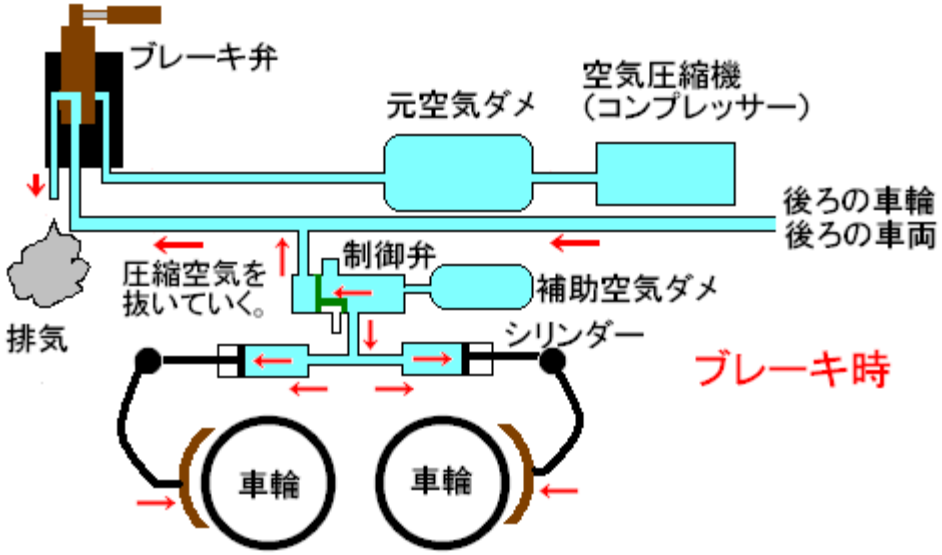
- 高信頼化設計（概要）
- 論理回路（概要）（赤字以外は「論理回路」の復習）
  - 基本ゲート、MIL記号
  - 種々の基本定理、ド・モルガンの定理
  - 組合せ回路の最適化、万能論理関数集合
  - デジタルIC
- 基本論理回路
  - マルチプレクサ、デコーダ、エンコーダ、加算器、…
- ソフトウェアによる論理設計
  - ゲートアレイ、PLA、FPGA等

高信頼化設計

- フェール・セーフ（Fail safe）
  - 失敗（故障、トラブル発生、操作ミス）した場合でも、安全側に倒れる（制御する）ようにしておくこと（装置やシステムは必ず故障することを前提）
  - 例：鉄道車両の自動空気ブレーキ



直通空気ブレーキ



自動空気ブレーキ

高信頼化設計

- **フォールト・トレラント（Fault tolerant）**

  フェールセーフの基本は「壊れたら止めるだが」、飛行中の航空機エンジンやコンピュータの基幹システムは止めることができない。

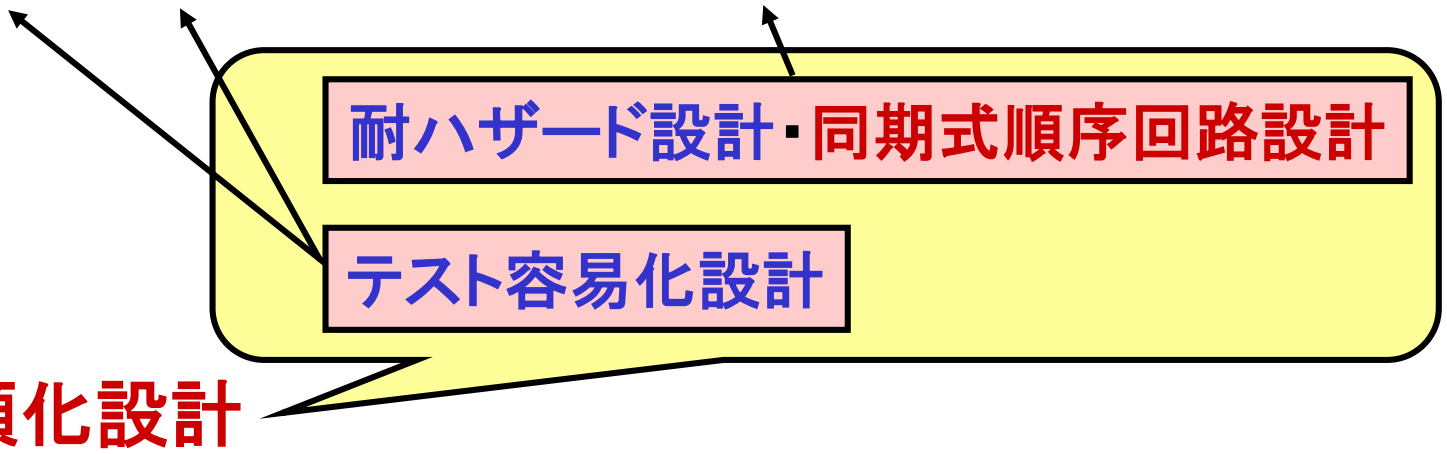
  - **システムの一部に障害が発生しても、最低限の機能を維持するようにしておくこと**
  - **例**
    - **卒研レポート**
      - ドラフト版を提出しておく
      - コピーを人に預けておく
    - **データのバックアップ**
      - 必ずしも最新のデータでは無くなるが
    - **航空機、自動車**
      - 双発エンジン、系統の多重化等
      - **ランフラットタイヤの装着**、応急タイヤの装備、

高信頼化設計

- イージー・メンテナンス（Easy maintenance）
- メンテナンス・フリー（Maintenance free）
  - 性能維持のためや、少々の故障をしても、わざわざ手間をかけた対応をしなくてもよいようにしておくこと
  - 例
    - インクジェット・プリンタ
      - 時々、自動ヘッドクリーニングが作動する
    - シールドバッテリ
      - 寿命まで補水が不要

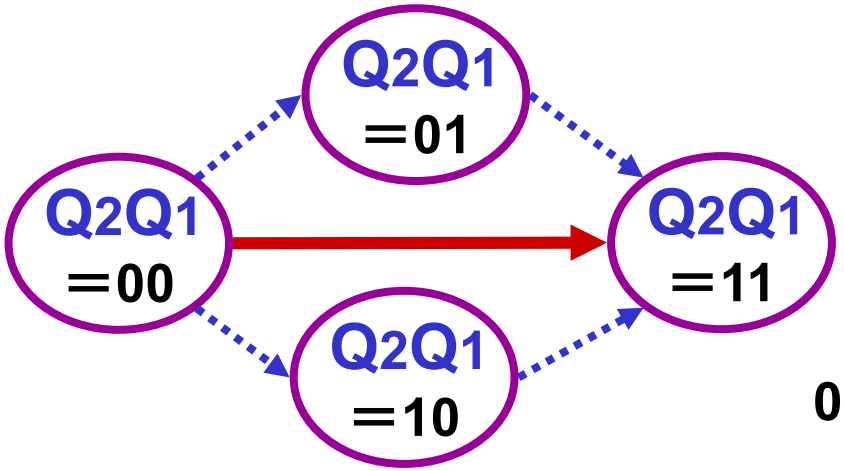
高信頼化設計

- 仕様のバグ
- 設計者のミスによるバグ
- ツールでのバグ
- **使用部品・製品自身の不具合（バグとは言わない）**
- 際どい設計が原因の場合もある
  - （初期）不良、故障、（外乱による）動作不良



高信頼化設計

- **競合／レーシング（racing）の例**
  - **非同期式順序回路**において、**2つ以上の状態変数**（メモリ）が**同時に変化**することを**競合（レーシング）**という
  - 更に、競合の際に「同時」という状態変化タイミングにおける微少な遅延のバラツキによって、**不正な**（＝状態遷移表で指定した遷移先と異なる）**遷移先を持つ可能性**がある場合を**際どい競合**、あるいは、**クリティカルレース（critical race）**という



通常、単に レーシング と言うと、
クリティカルレースの事を指す

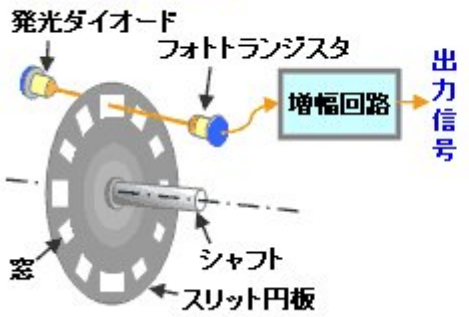
00→11 はレーシングである

二進数以外の符号化の例(1)

- グレイコードの応用例

通常のバイナリコードを用では、ハザードが発生すると全く異なる角度が検出されてしまう。

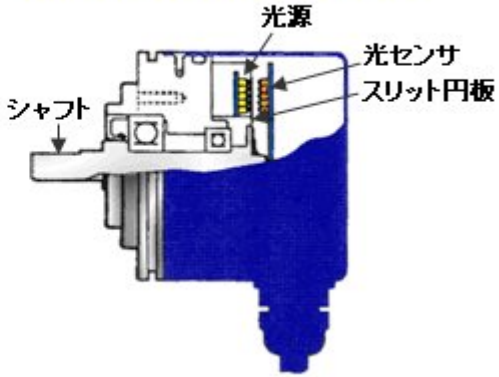
＜図5　光学式の構造＞



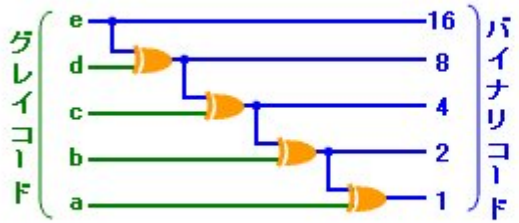
＜図6　受光素子の出力信号＞

＜図9　アブソリュート形のスリット円板＞

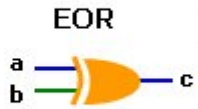
＜図10　スリット円板とセンサの配置＞



＜図11　グレイ－バイナリ変換回路＞



＜図12　EORの真理値表＞

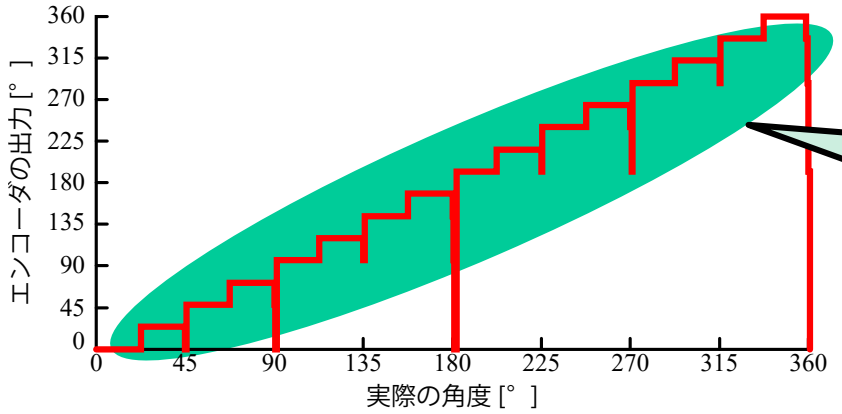


入力		出力
a	b	c
0	0	0
1	0	1
0	1	1
1	1	0

＜表1　グレイコード表＞

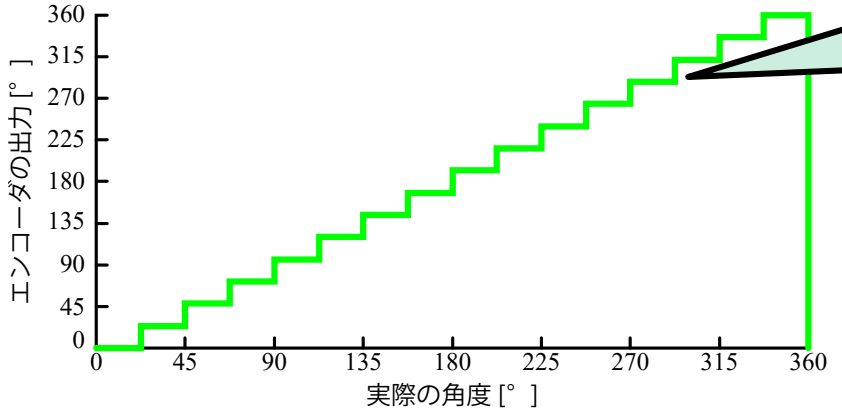
10進数	バイナリコード					グレイコード				
	16	8	4	2	1	e	d	c	b	a
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	1
3	0	0	0	1	1	0	0	0	1	0
4	0	0	1	0	0	0	0	1	1	0
5	0	0	1	0	1	0	0	1	1	1
6	0	0	1	1	0	0	0	1	0	1
7	0	0	1	1	1	0	0	1	0	0
8	0	1	0	0	0	0	1	1	0	0
9	0	1	0	0	1	0	1	1	0	1
10	0	1	0	1	0	0	1	1	1	1
11	0	1	0	1	1	0	1	1	1	0
12	0	1	1	0	0	0	1	0	1	0
13	0	1	1	0	1	0	1	0	1	1
14	0	1	1	1	0	0	1	0	0	1
15	0	1	1	1	1	0	1	0	0	0
16	1	0	0	0	0	1	1	0	0	0
17	1	0	0	0	1	1	1	0	0	1
18	1	0	0	1	0	1	1	0	1	1
19	1	0	0	1	1	1	1	0	1	0

符号化の比較



桁間のバラつきによるハザードのため、角度によっては大きな誤差が発生する

バイナリコード

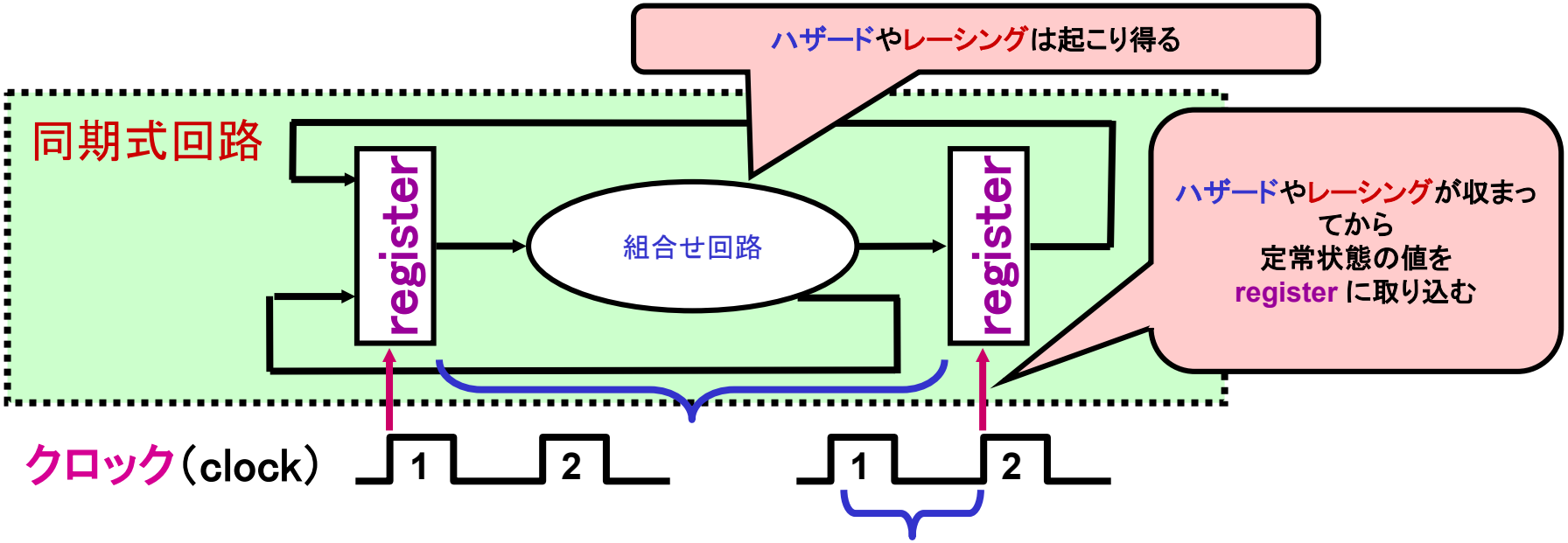


ハザードフリーのため、誤差は機械的精度で決まる

グレイコード

高信頼化設計

- 同期式（順序）回路のメリット
  - ハザードやレーシングが発生しても、消滅した後の定常状態の値を次状態としてラッチに保持
    ⇒ハザードフリー、レーシングフリーである



  - 全ての register（＝ラッチ）が同期して動作
    ⇒ 遅延時間の計算が簡単になる
    （register の出口から次の register の入口まで）

高信頼化設計

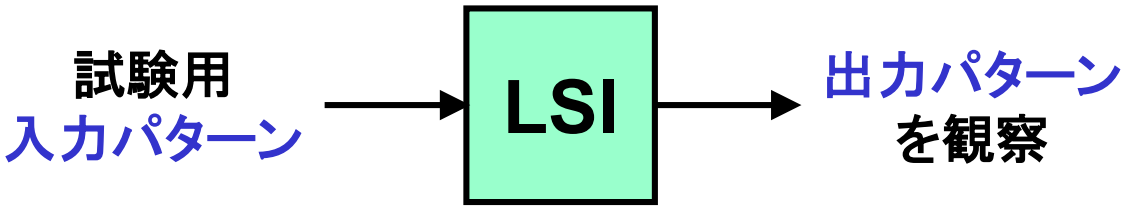
- 非同期式回路 vs. 同期式回路
  - 非同期式回路
    - 高速動作が期待できるが、ハザードやレーシングによる「トラブル」から逃れることは大変困難
  - 同期式回路
    - register から register までの組合せ回路のクリティカル・パス（最大遅延経路）を揃えることにより、ある程度の高速動作が可能
    - 組合せ回路部分のみの遅延計算なので、設計が簡単（見落としが少ない）で、CADにも適している
    - 即ち、大規模な回路に適している

⇒出来るだけ同期式順序回路（同期回路）で回路を組もう！

高信頼化設計

- テスト容易化設計
  1. LSIの中の故障(不具合)を見つけるために出荷試験を行いたい。
  2. 正常動作しない不良LSIが度々製造される。どこが悪いのか調べたい。

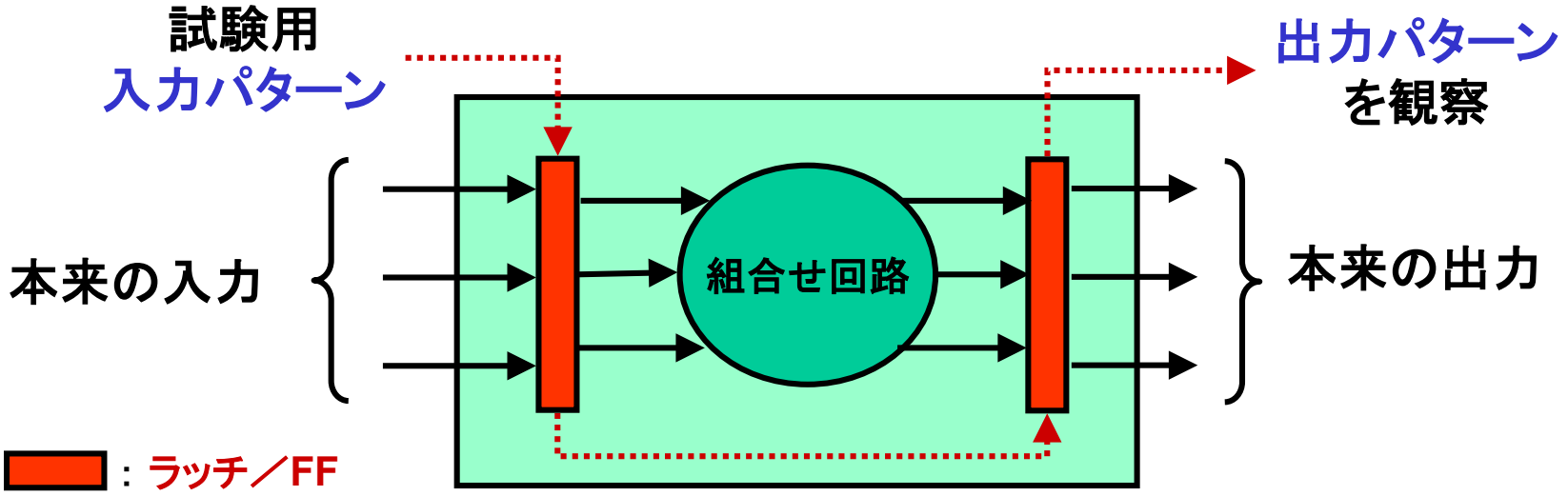
     ⇒ LSIに入力を与えて、出力を観察すれば良い。



1. しかし、LSIは内部に状態を持つため、あらゆる状況を網羅する試験は現実的ではない(時間・コスト)。
2. また、外部から観察される出力だけでは、回路のどの部分が不良なのか分からない。

高信頼化設計

- テスト容易化設計
  - 組合せ回路の場合は、比較的簡単
    入力を与え、出力と期待出力を比較（故障シミュレーション）
  - 順序回路の場合は、
    全てのラッチ／FFに任意の値をセット出来るようにする
    次に 1クロックだけ動作させる
    動作後の全てのラッチ／FFの値を観測出来るようにする

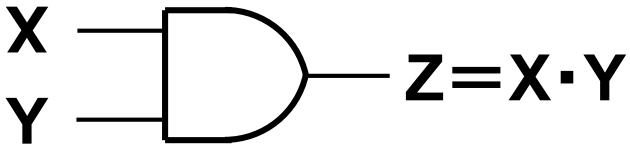


⇒ スキャン設計と称する

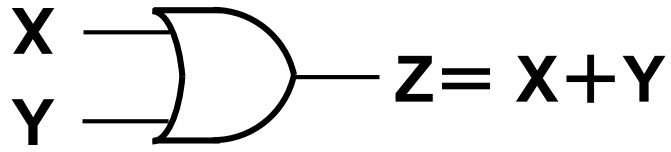
論理回路(概論)

- 3つの基本論理ゲート
  - NOTゲート: $X = \overline{Z}$ なる論理ゲート

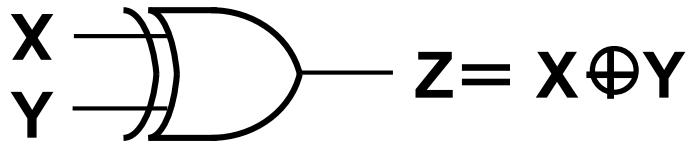
  - ANDゲート: $X \cdot Y = Z$ なる論理ゲート



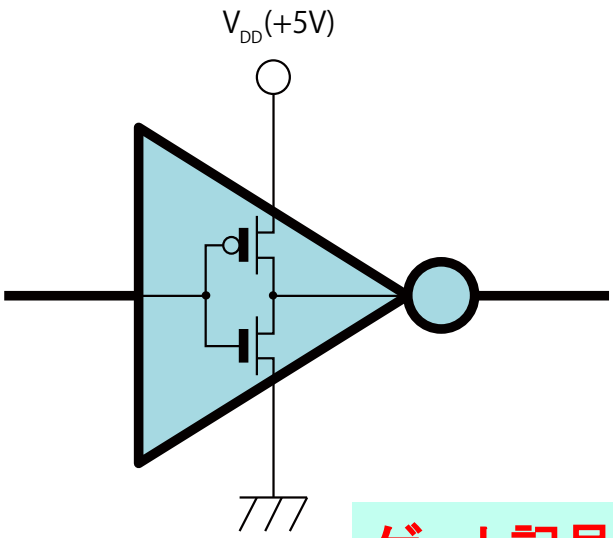
  - ORゲート: $X + Y = Z$ なる論理ゲート

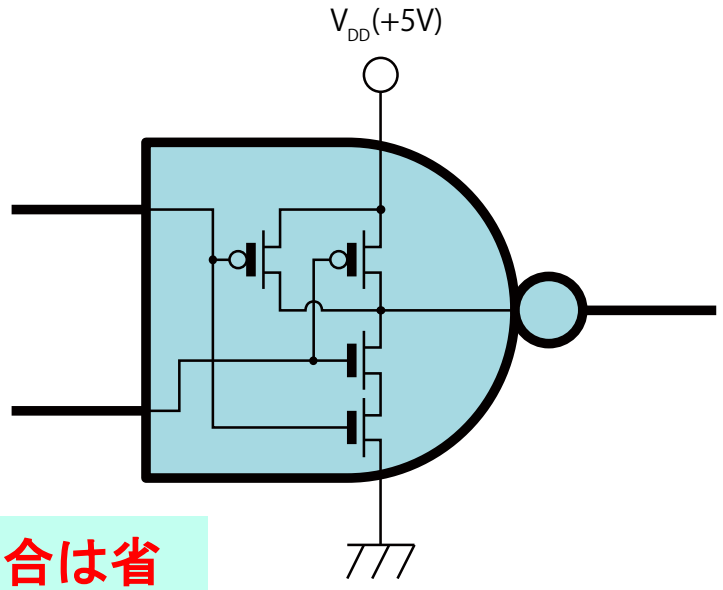


- 追加の論理ゲート
  - EXORゲート: $Z = X \oplus Y = \overline{X} \cdot Y + X \cdot \overline{Y}$ なる論理ゲート

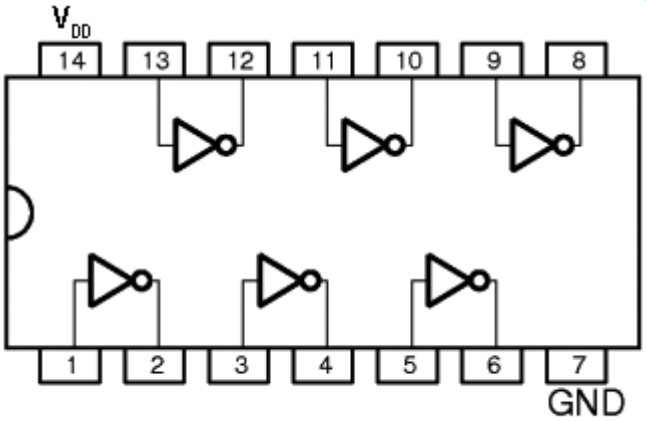


論理回路（概論）

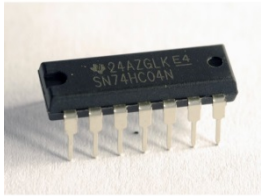


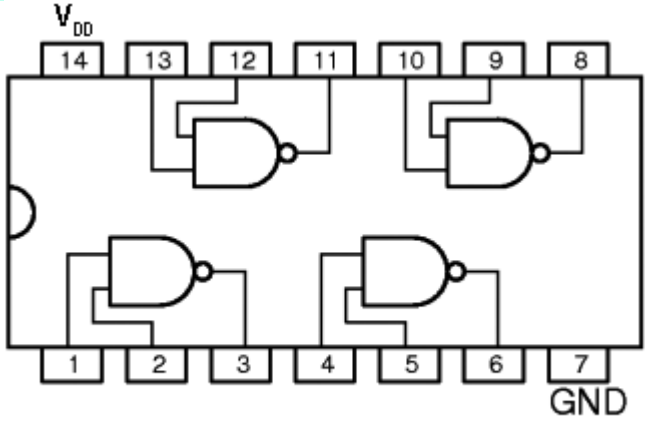


ゲート記号を描く場合は省略されるが、実際には電源が繋がっている



7404　Hex Inverters

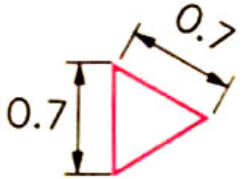




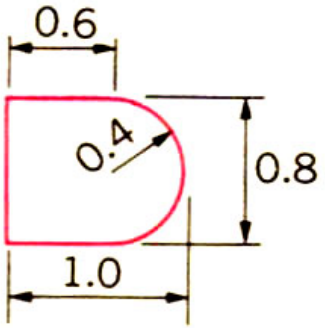
7400　Quad 2 Input NAND

論理回路(概論)

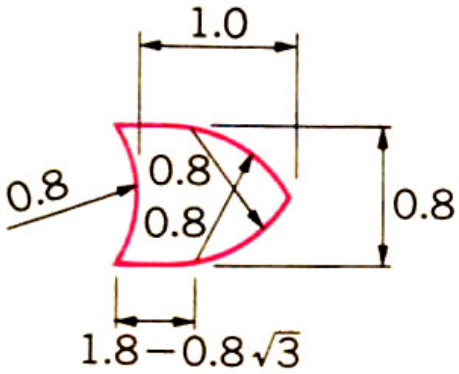
- MIL(Military Standard Specification)記号



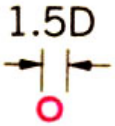
(a) アンプ



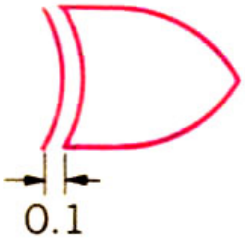
(b) AND



(c) OR



(d) 負論理



(e) Exclusive OR

(f) 機能ブロック

(g) 入力端子

(寸法は製図の際の比例定数)

論理回路(概論)... 種々の基本定理

- **定理1.8(交換則)**

  $X \cdot Y = Y \cdot X \quad \Leftrightarrow \quad X + Y = Y + X$

- **定理1.9(結合則)**

  $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) \quad \Leftrightarrow \quad (X + Y) + Z = X + (Y + Z)$

- **定理1.10(分配則)**

  $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$

  $\Leftrightarrow \quad X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$

- **定理1.11(吸収則)**

  $X + (X \cdot Y) = X \quad \Leftrightarrow \quad X \cdot (X + Y) = X$

  $X + (\overline{X} \cdot Y) = X + Y \quad \Leftrightarrow \quad X \cdot (\overline{X} + Y) = X \cdot Y$ **(系1.12)**

論理回路(概論)...ド・モルガンの定理

- 定理1.13(De Morganの定理)

$$\overline{X \cdot Y} = \overline{X} + \overline{Y} \quad \Leftrightarrow \quad \overline{X + Y} = \overline{X} \cdot \overline{Y}$$

$$\overline{(X_1 \cdot X_2 \cdots X_n)} = \overline{X_1} + \overline{X_2} + \cdots + \overline{X_n}$$

$$\Leftrightarrow \quad \overline{(X_1 + X_2 + \cdots + X_n)} = \overline{X_1} \cdot \overline{X_2} \cdots \overline{X_n}$$ (系1.14)

$$X \cdot Y = \overline{\overline{X} + \overline{Y}} \quad \Leftrightarrow \quad X + Y = \overline{\overline{X} \cdot \overline{Y}}$$ (系1.15)

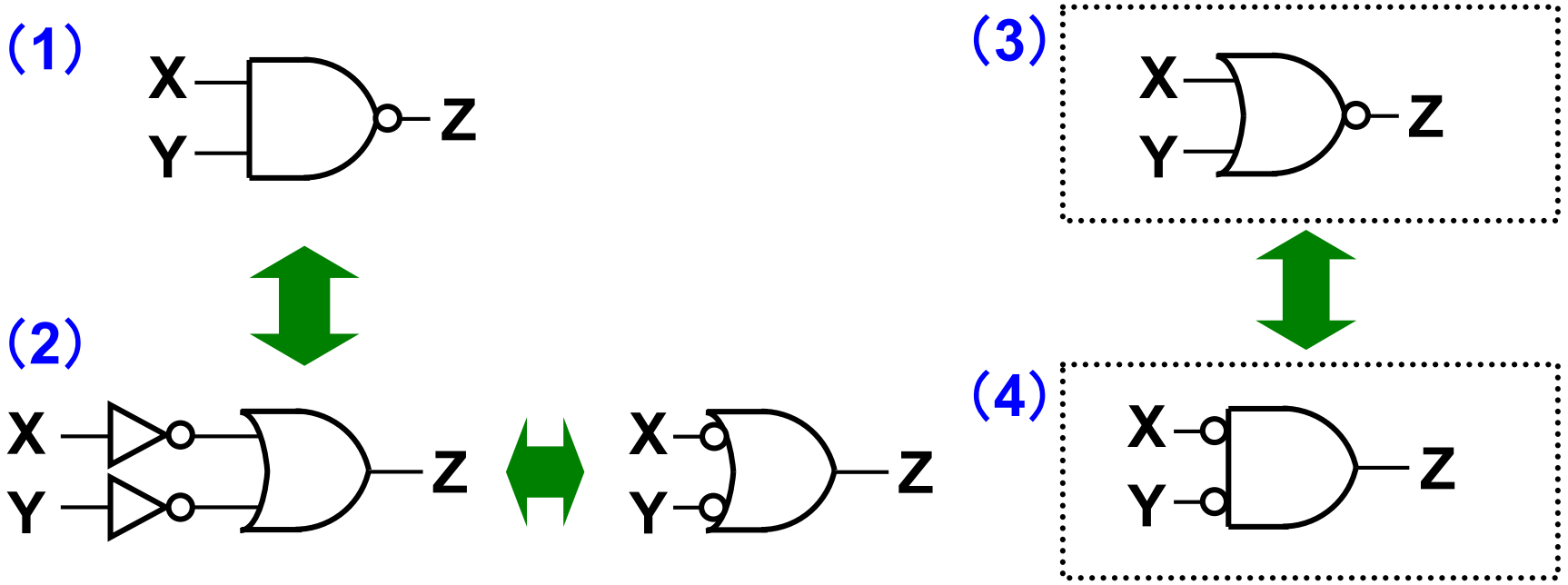
論理回路(概論)

●論理回路上でのド･モルガンの定理

- 定理1.13(De Morganの定理)

$$\overline{X \cdot Y} = \overline{X} + \overline{Y} \quad \Leftrightarrow \text{双対} \quad \overline{X + Y} = \overline{X} \cdot \overline{Y}$$

(1) (2) 双対 (3) (4)



論理回路(概論)

●論理回路上でのド･モルガンの定理

- 定理1.13(De Morganの定理)

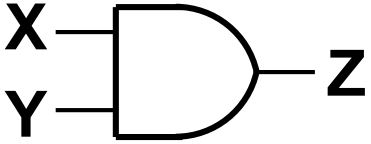
$$X \cdot Y = \overline{\overline{X} + \overline{Y}} \quad \Leftrightarrow \text{双対} \quad X + Y = \overline{\overline{X} \cdot \overline{Y}}$$

(1) (2) (3) (4)

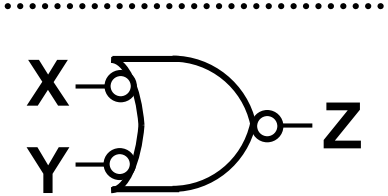
(系1.15)

●演習:点線の中を埋めよ

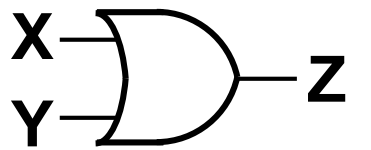
(1)



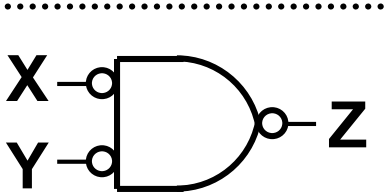
(2)



(3)



(4)



論理回路(概論)... 組合せ回路の最適化

- **最適化の目的**
  - 論理ゲートの個数が減り、配線が簡単になる

    ⇒論理回路の**実装コストが安く**なる

    ⇒空間的に**小さく**できる

    ⇒**高速**に動作する

    ⇒**故障し難く**なる

- **論理関数の簡単化 vs. 論理回路の最適化**
  - 論理演算の個数を減らす ⇒ 論理ゲートの個数を減らす

    論理関数の簡単化 ⇒ 論理回路の最適化

    論理関数の最小化 ⇒ 論理回路の最適化

論理回路(概論)... 組合せ回路の最適化

- **最適化設計の具体的目的**

  A. 空間サイズの削減

  「論理回路を構成するのに必要な**論理ゲートの総数**」を**最小**にする

  ⇒ **ファクタリング**(括り出し)等による**多段論理最小化**

  B. 時間サイズの削減

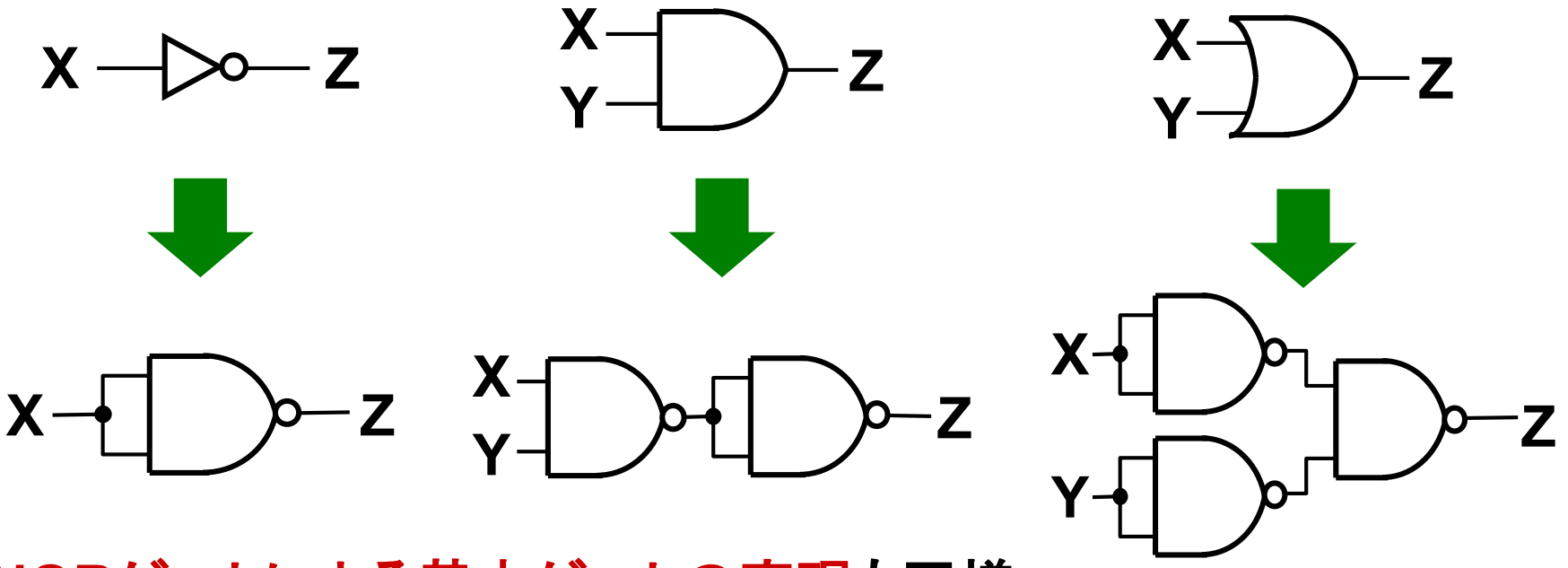
  「論理回路の入力端子から出力端子に至るまでに**通過する論理ゲートの段数**」を**最小**にする

  ⇒ **カルノー図**や**Q-M法**を用いて**最小積和型論理式**を求めることにより、どのような論理関数も**AND－OR回路**、すなわち、(否定1段＋)2段で実現できる

Aと**B**は両立するのか？　... **必ずしも両立しない！**

論理回路(概論)… 万能論理関数集合

- **NANDゲートによる基本ゲートの実現**
  - NOT: $\overline{X}=Z \rightarrow \overline{X \cdot X}=\overline{X}=Z$
  - AND: $X \cdot Y=Z \rightarrow \overline{\overline{(X \cdot Y)} \cdot \overline{(X \cdot Y)}}=\overline{X \cdot Y}=X \cdot Y=Z$
  - OR: $X+Y=Z \rightarrow \overline{\overline{(X \cdot X)} \cdot \overline{(Y \cdot Y)}}=\overline{X} \cdot \overline{Y}=X+Y=Z$



NORゲートによる基本ゲートの実現も同様

論理回路(概論)... 万能論理関数集合

- **定理2.4(NANDの万能性)**
  - 任意の論理関数は NAND だけで表せる. 従って、次の U_3 も万能論理関数集合である、

$$U_3 = \{ \text{NAND} \}$$

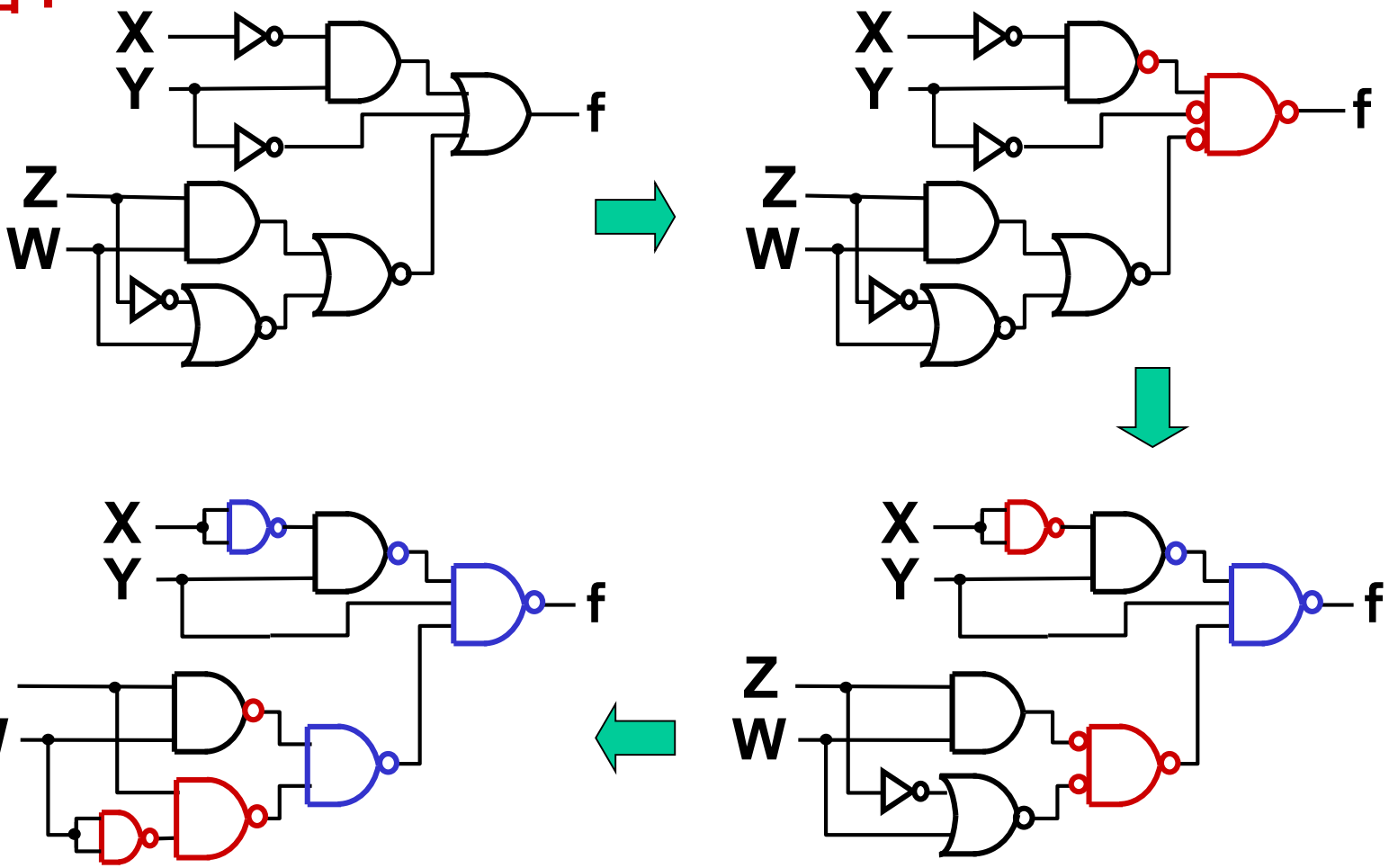
証明: NAND演算(“｜”)によって NOT,AND,OR が表せればよい

別証明: NAND演算(“｜”)によって NOT,ANDが表せればよい

- **定義2.19(NAND形式)**
  - U_3 によって表した論理関数(論理式)を NAND形式 という

論理回路(概論)... 万能論理関数集合

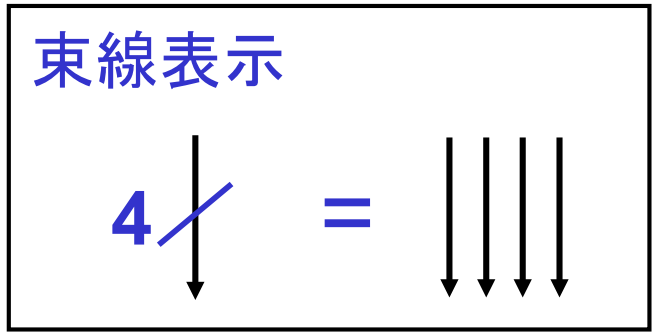
- 演習1: 下図のAND／OR回路を NAND回路に変換せよ
- 解答:

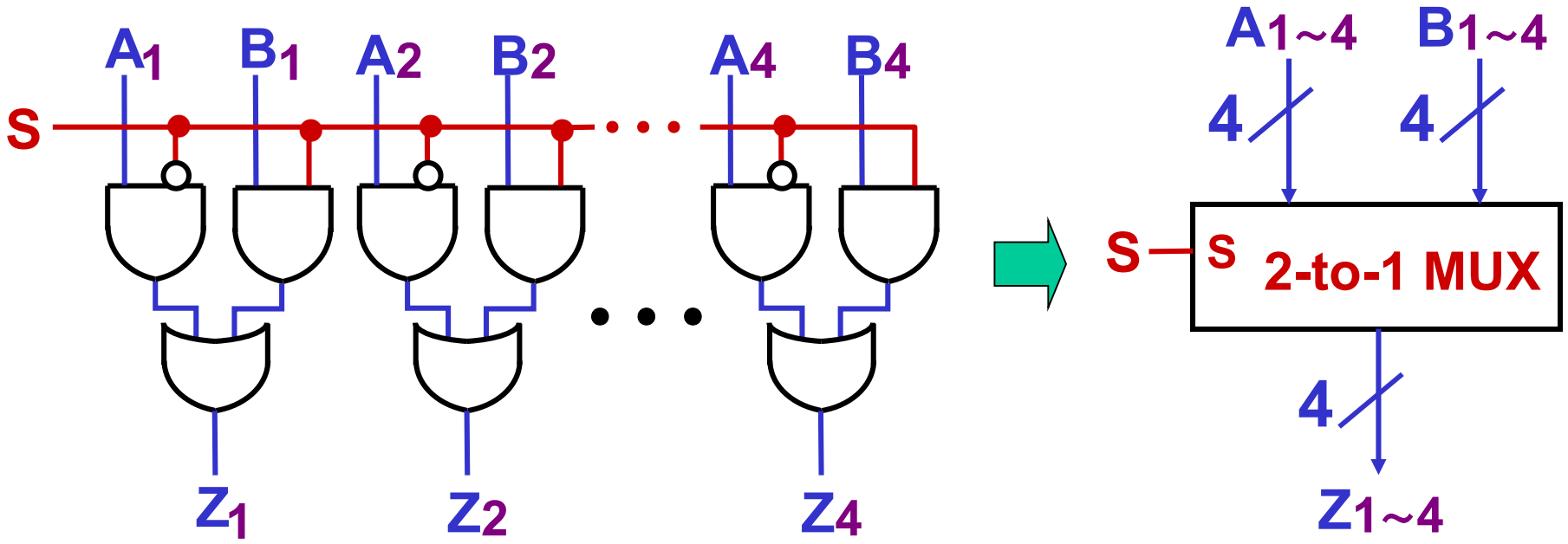


基本論理回路

- マルチプレクサ（multiplexer）
  - selector ともいう
  - 通常 2^k-to-1マルチプレクサ

- 例：2-to-1マルチプレクサ
  （2-to-1 MUX）





基本論理回路

- 例: 4-to-1マルチプレクサ（4-to-1 MUX）
- 演習2: 右表の機能を持つ4-to-1マルチプレクサ（1ビット）のAND／OR回路を設計せよ
- 解答: 論理関数は、

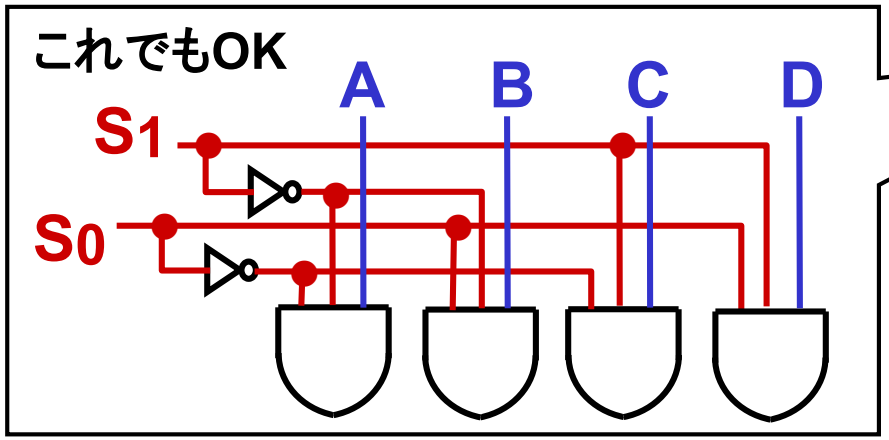
$$Z = \overline{S_1} \cdot \overline{S_0} \cdot A + \overline{S_1} \cdot S_0 \cdot B + S_1 \cdot \overline{S_0} \cdot C + S_1 \cdot S_0 \cdot D$$

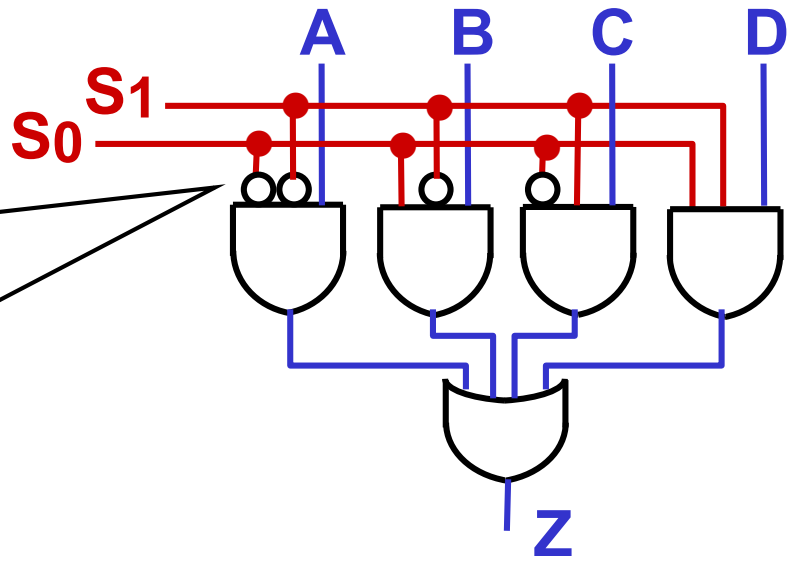
故に、右図となる

機能定義表

真理値表ではない！

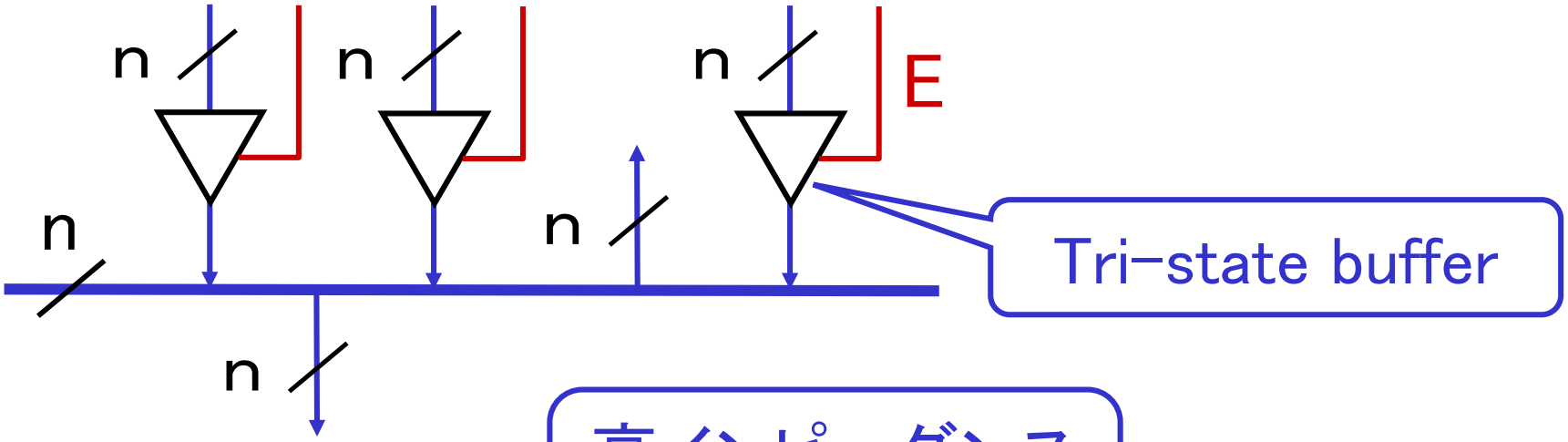
S_1	S_0	Z
0	0	A
0	1	B
1	0	C
1	1	D





基本論理回路

- bus (バス)



E	In	Out
1	0	0
1	1	1
0	X	Z

高インピーダンス
（断線と等価）

演習：

MUXとの違い（メリット）は？

基本論理回路

- **デコーダ（decoder）**
  - n本の信号を2進数列のコードと見なし、それに対応する単一出力（2^n本のデータ出力のうち1本だけ「1」とし、他を「0」とする）を生成するものを n×m（＝2^n）デコーダという

2×4デコーダの機能定義表

D_1	D_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

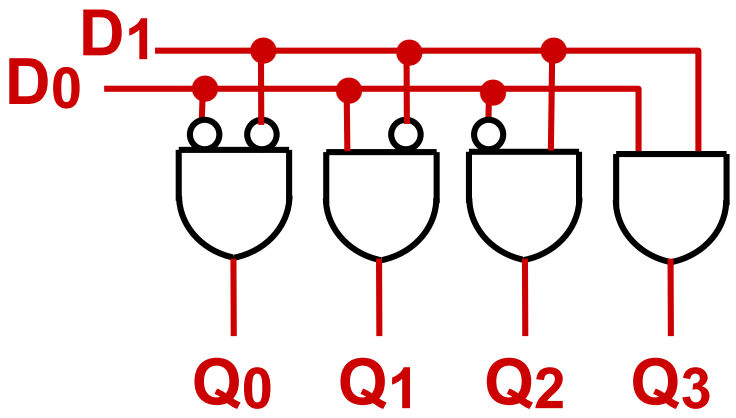
- **デマルチプレクサ（demultiplexer）**
  - n本の選択制御線によって、2^n本のデータ出力のうちから対応する1本を選んで、それだけに1本の入力データを分配（接続）するものを 1×m（＝2^n）デマルチプレクサという

1×4デマルチプレクサの機能定義表

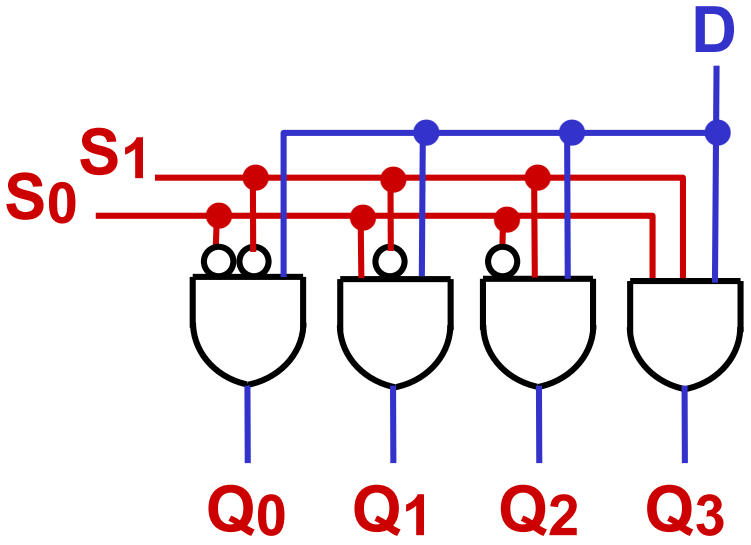
S_1	S_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

基本論理回路

- 演習3:
  - 前頁の機能定義表に対応する2×4デコーダを設計せよ
- 解答:



- 演習4:
  - 前頁の機能定義表に対応する1×4デマルチプレクサを設計せよ
- 解答:



基本論理回路

- エンコーダ（encoder）
  - m本の信号の内、1つだけ「1」となり、「1」となった入力に対応する「n本の出力の組（コード）」を生成する
  - m（＝2^n）×nエンコーダという

4×2エンコーダの機能定義表

D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
（上記以外）				—	—

- プライオリティエンコーダ（priority encoder）
  - m本の信号の内、優先度の高い信号の「1」に対応する「n本の出力の組（コード）」を生成する
  - 優先度付きエンコーダともいう

4×2優先度付きエンコーダの機能定義表

D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	—	0	1
0	1	—	—	1	0
1	—	—	—	1	1
0	0	0	0	—	—

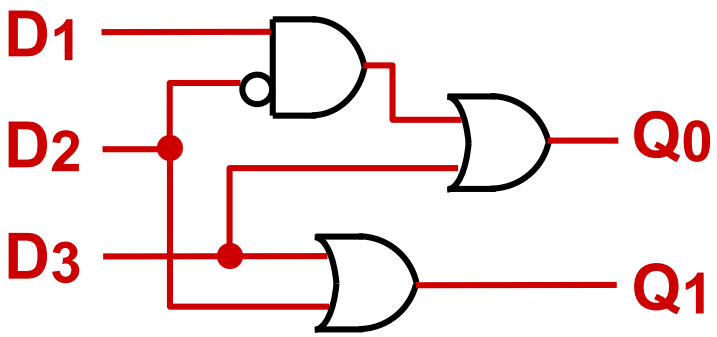
基本論理回路

- **演習：**
  - 前頁の**機能定義表**に対応する**2×4優先度付きエンコーダ**を設計せよ
- **解答：**カルノー図は右図となるので、論理関数は、

$$Q_1 = D_3 + D_2$$

$$Q_0 = D_3 + \overline{D_2} \cdot D_1$$

故に、論理回路は下図



Q1

D1D0 \ D3D2	00	01	11	10
00	—	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

Q0

D1D0 \ D3D2	00	01	11	10
00	—	0	1	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1

(用語解説)

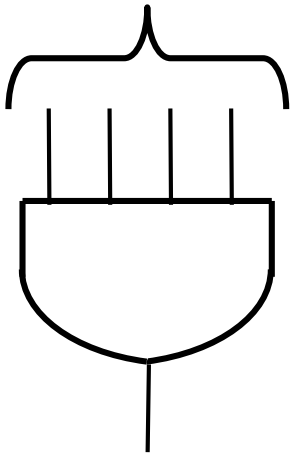
- **fan-in**
  - 1ゲートへの入力信号数
  - 通常、ゲートごとに上限あり

  または
  - ロジックICの入力が，そのロジックを駆動するロジックに与える負荷を単位ロジックの入力本数で表したもの

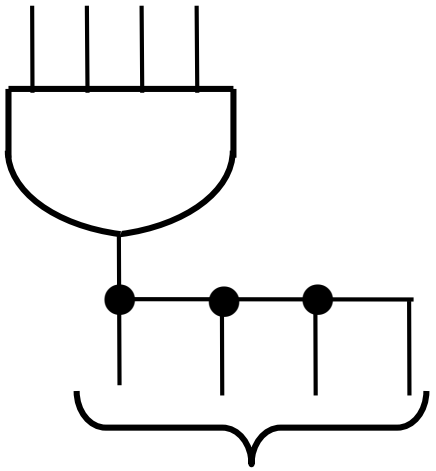
- **fan-out**
  - 1ゲートからの出力信号数
  - やはり、ゲートごとに上限あり

  または
  - ロジックICの出力が駆動できるロジック信号入力数を単位ロジックの入力本数で表したもの．
  - ファンアウトが大きいほど，駆動能力が大きく，したがって，より多くのロジック入力を接続できる．
  - 出力の駆動能力および周波数により異なる．

例： fan−in = 4



例：



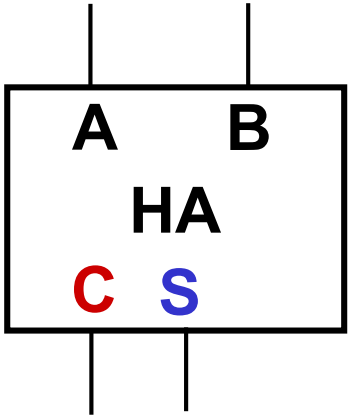
fan−out = 4

基本論理回路

- 半加算器（HA: Half Adder）

S：和（Sum）
C：桁上げ（Carry）

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

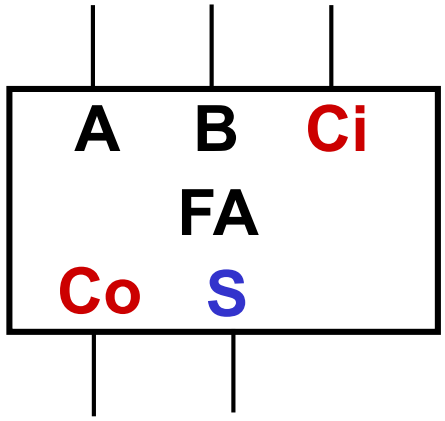


- 全加算器（FA: Full Adder）

Ci：Carry-in
S：和（Sum）
Co：Carry-out

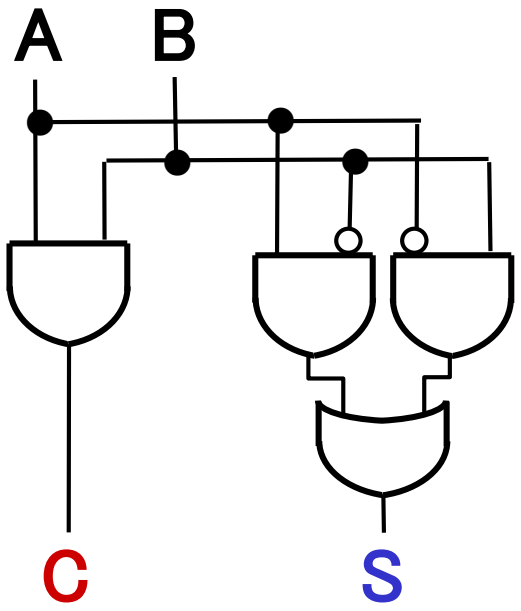
A	B	Ci	S	Co
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

A	B	Ci	S	Co
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

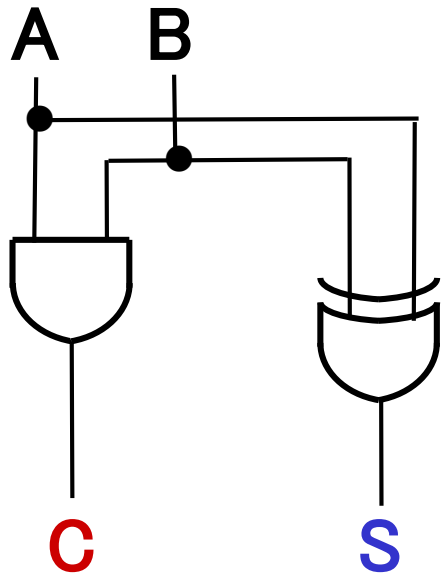


基本論理回路

- **演習：**
  - 半加算器の論理回路を**AND／OR**回路で設計せよ
- **解答：**

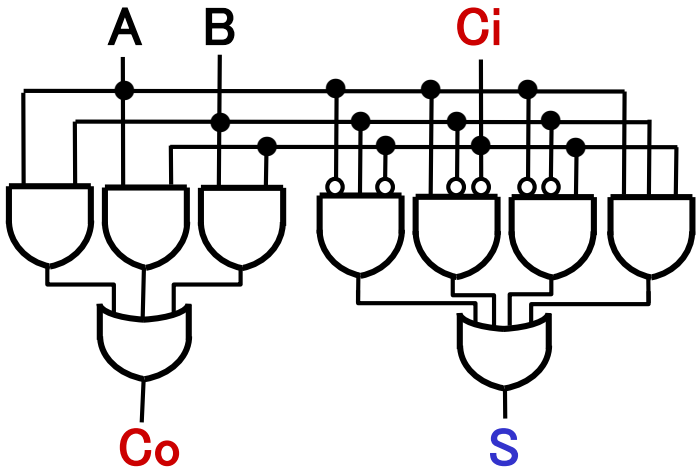


- **演習：**
  - 半加算器の論理回路を**AND, OR, NOT, XOR**ゲートで設計せよ
- **解答：**

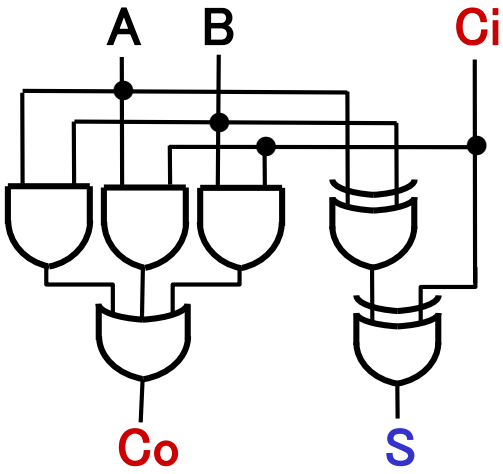


基本論理回路

- **演習：**
  - 全加算器の論理回路を**AND／OR回路**で設計せよ
- **解答：**



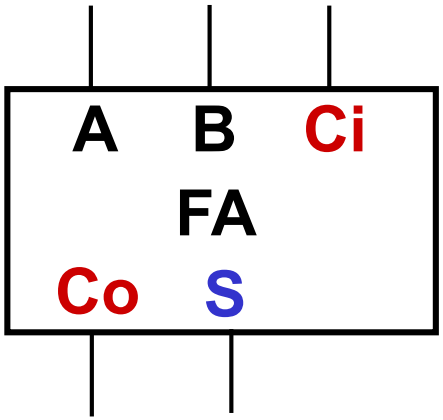
- **演習：**
  - 全加算器の論理回路を**AND, OR, NOT, XOR**ゲートで設計せよ
- **解答：**

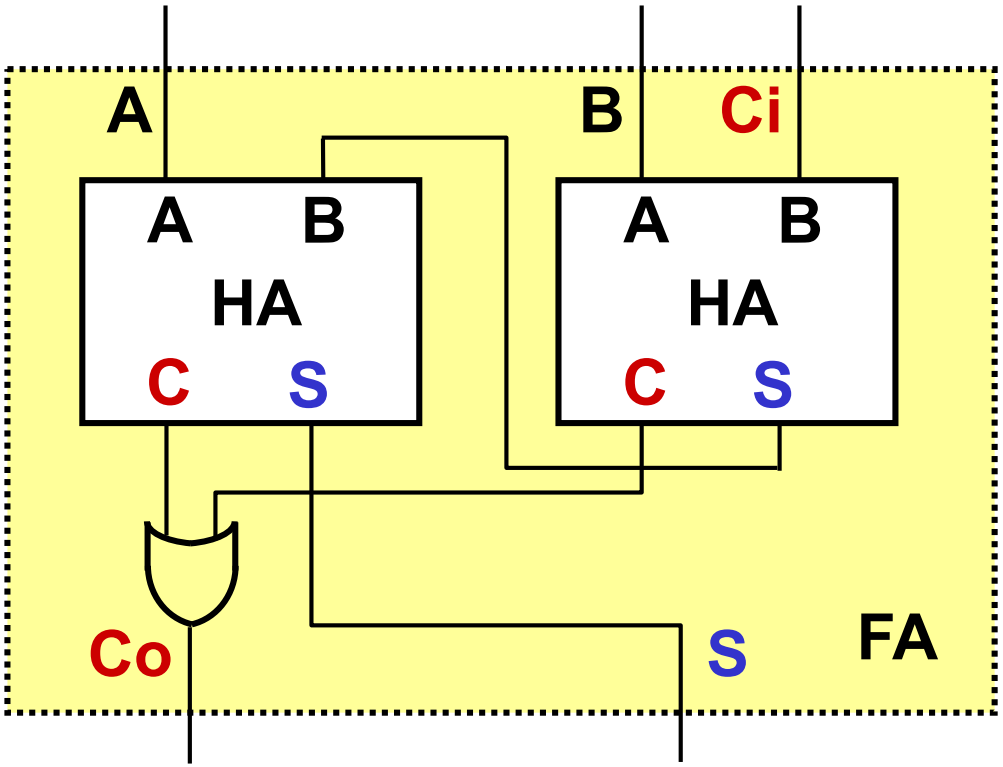


基本論理回路

- 2個の半加算器（と1個のORゲート）で構成した全加算器

Ci : Carry-in
S : 和（Sum）
Co : Carry-out





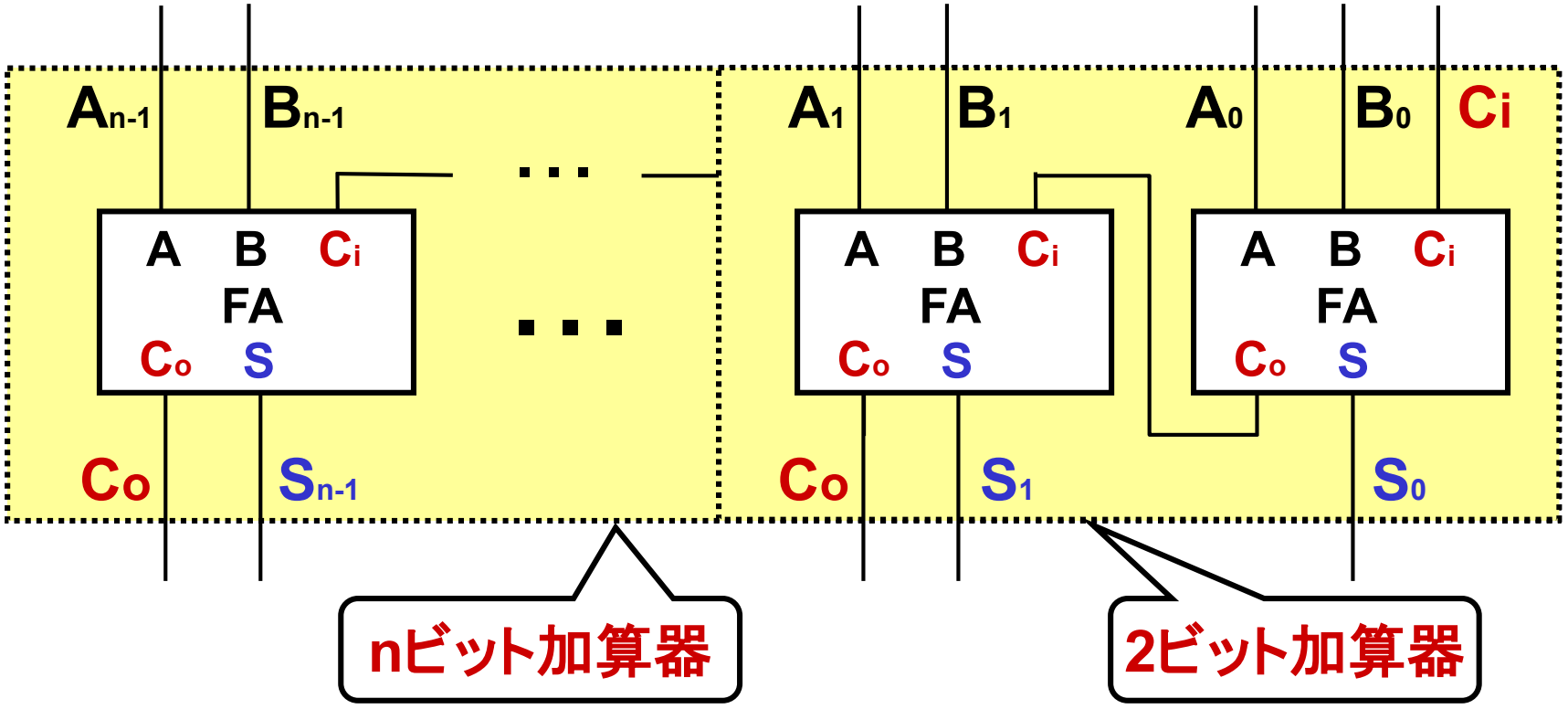
A	B	Ci	S	Co
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

A	B	Ci	S	Co
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

- 空間サイズ（S_T ＝4＋4＋1）も時間サイズ（4）も良くないので、実際には使われない

基本論理回路

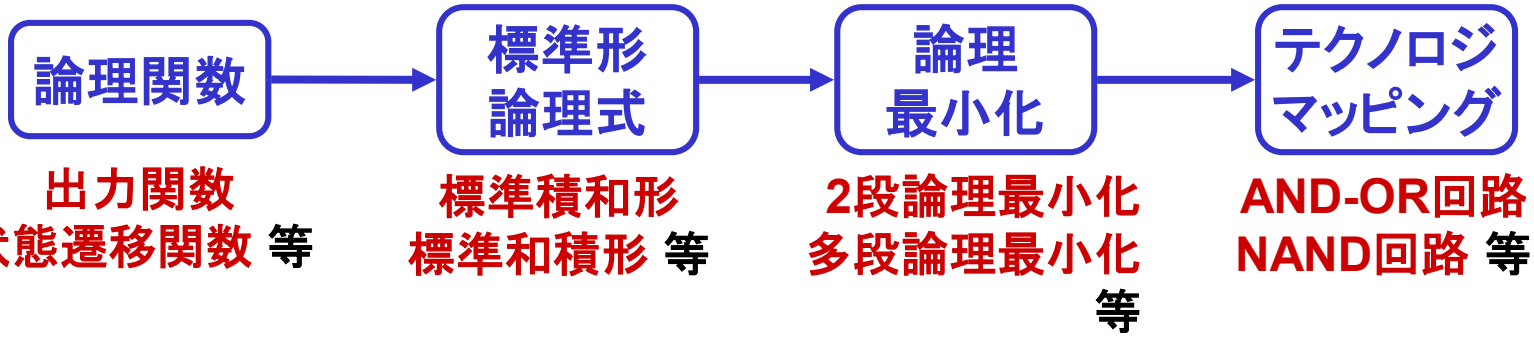
- **全加算器**（1ビット）をカスケード（＝縦つなぎ）接続して構成した **n ビット加算器**



- **空間サイズ**は良いが、**時間サイズ**はビット数に比例して悪くなる
  ⇒ **桁上げ予見回路**の付加（コンピュータアーキテクチャ）

ソフトウェアによる論理設計

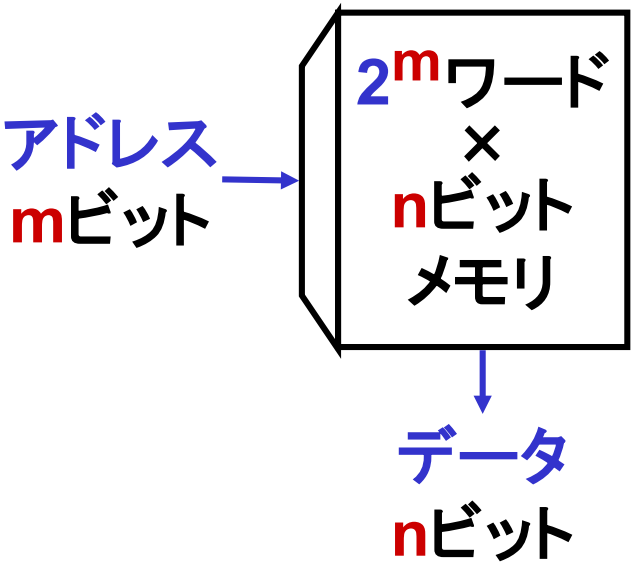
- **論理回路の標準形とテクノロジマッピング（復習）**
  - 論理関数から論理回路（組合せ論理回路／順序回路）へ



- **プログラム可能論理回路**
  - 一旦合成したものを修正できたり、何度も作り直せる論理回路を「プログラム可能（プログラマブル：programmable）論理回路」という
  - 論理回路の配線情報をプログラムとして持ち、これを書き換える事が可能になっている

ソフトウェアによる論理設計

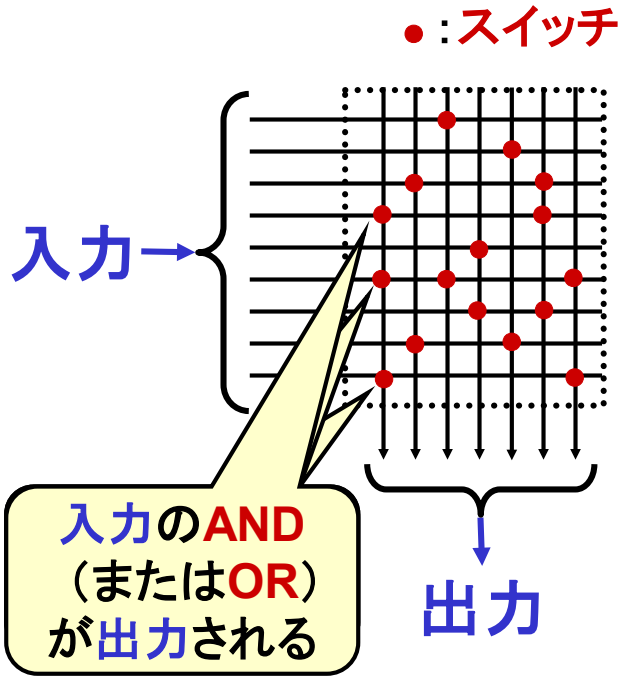
- プログラム可能回路（その1）
  - メモリによる組合せ回路の構成
    - 2^mワード×nビットのメモリは、「mビット入力／nビット出力」の組合せ回路と見なせる
    - メモリ内容を入れ替えることにより、任意の入力に対して、任意の出力を得ることが出来るため、プログラム可能論理回路として利用できる



    - メモリ内容は「mビット入力／nビット出力」の組合せ回路の真理値表そのものである
    - 論理が最適化されておらず、ドントケア等の不要な組合せに対してもメモリを使用するため、冗長性がある

ソフトウェアによる論理設計

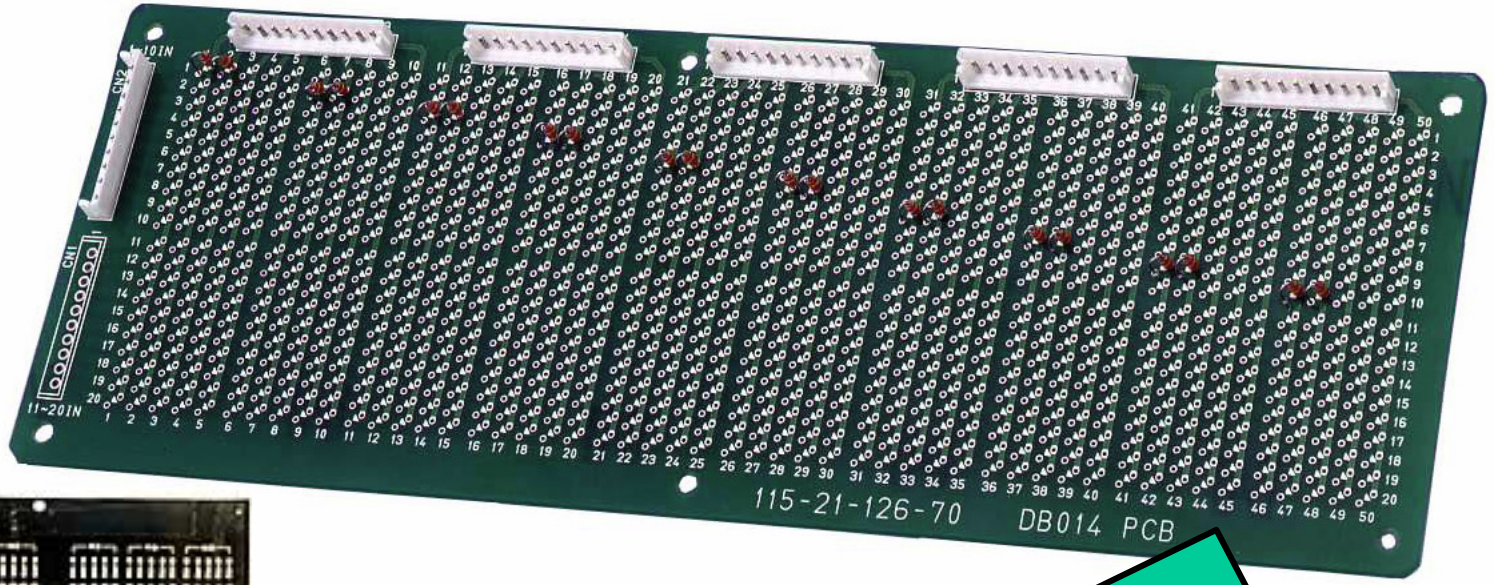
- プログラム可能回路（その2）
  - プログラム可能ANDアレイ
    - 選択された入力だけの論理積（AND）を出力するANDゲートの並び
  - プログラム可能ORアレイ
    - 選択された入力だけの論理和（OR）を出力するORゲートの並び

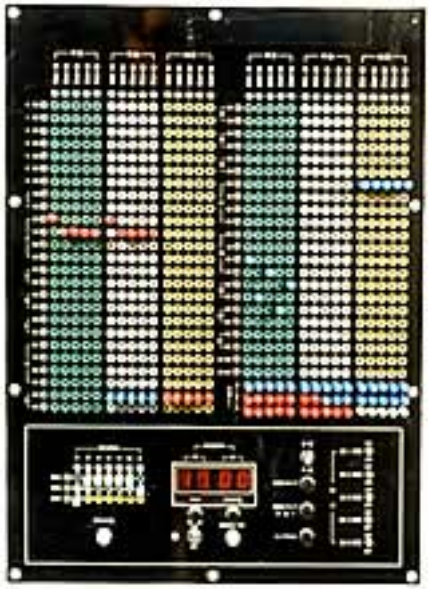


- スイッチが on の入力線同士の AND（またはOR）が出力される
- スイッチには、ヒューズ（fuse）などの機械式、電気式、紫外線などの光学式などがある

ソフトウェアによる論理設計

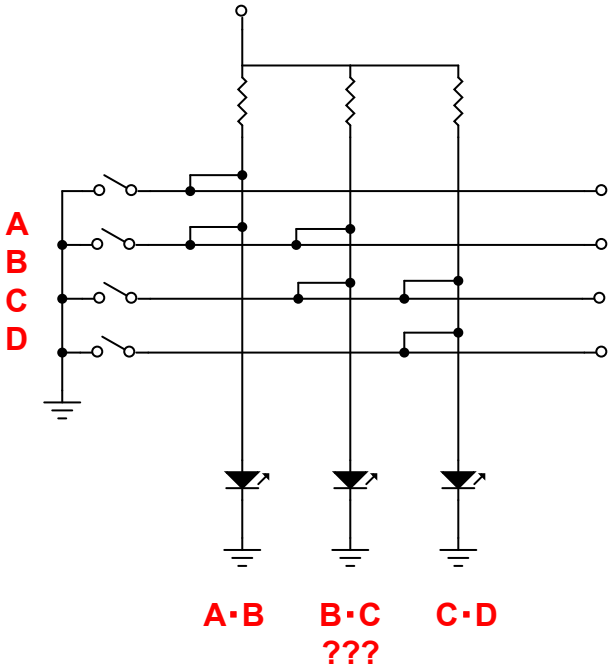
ダイオードマトリクス　[単なるスイッチ（短絡）では実現不可能]

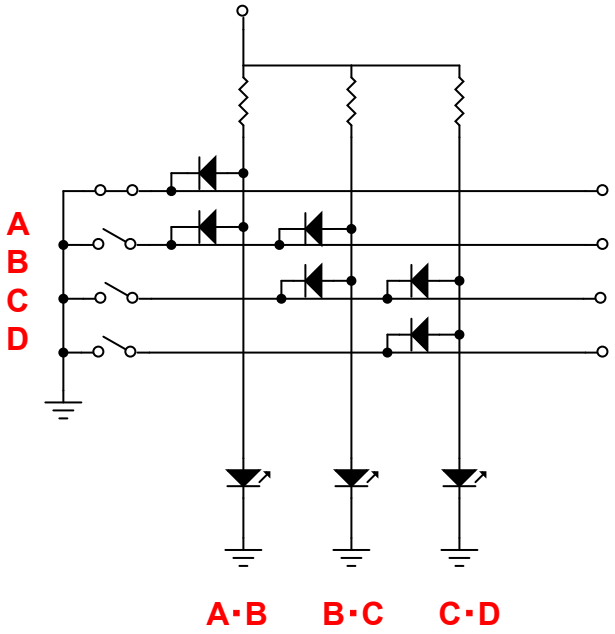




これで1000bit(=125Byte)
のROMとして機能する

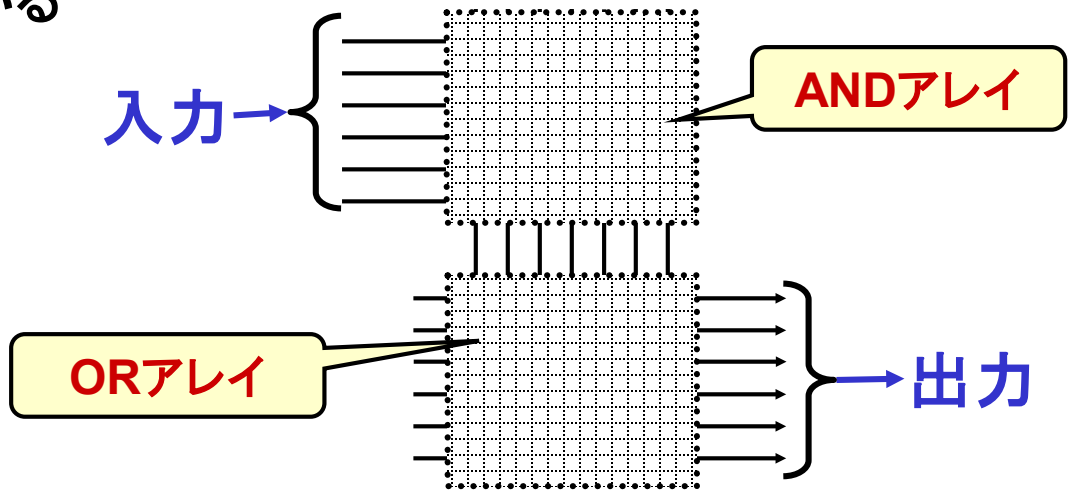
ダイオードマトリックスの原理





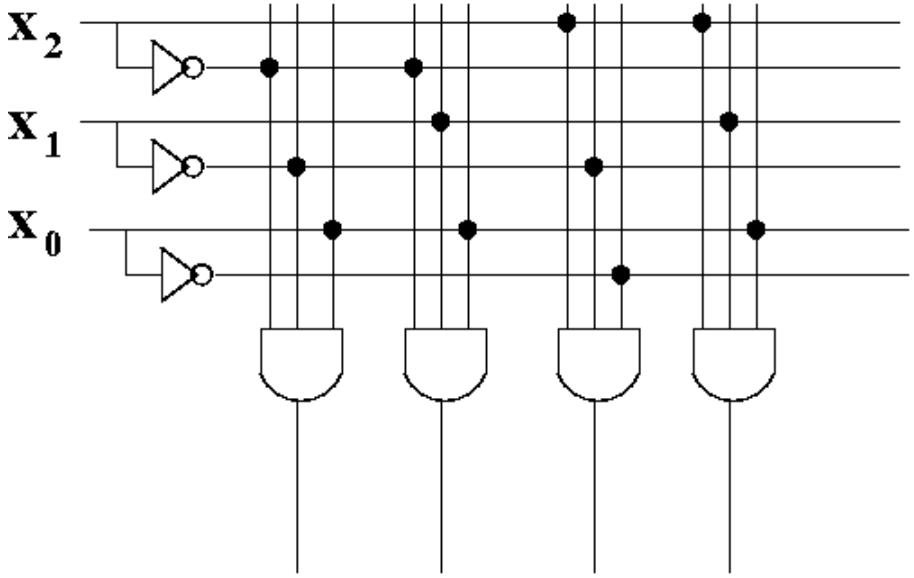
ソフトウェアによる論理設計

- プログラム可能回路（その3）
  - プログラム可能ロジックアレイ
    （PLA：Programmable Logic Array）
    - プログラム可能ANDアレイとプログラム可能ORアレイを組合せ、自由度を持たている

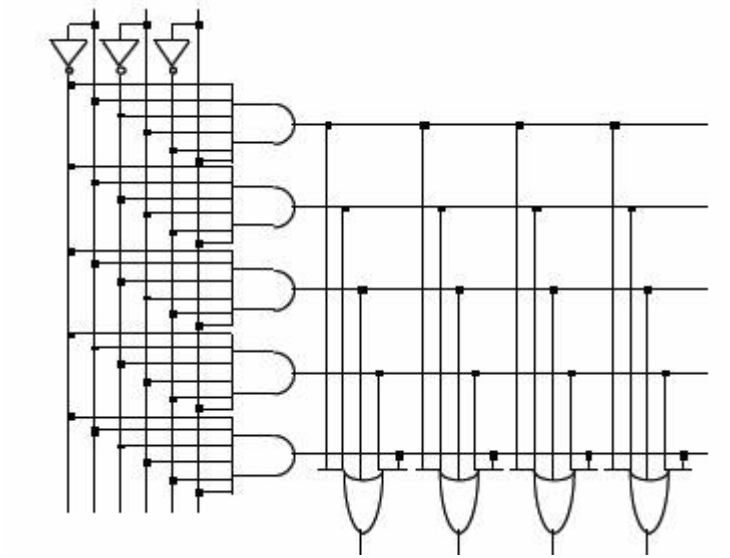


    - AND-OR回路（2段）を（ほぼ）自由に構成できる
  - 「使用者の手元でプログラム可能」なPLAをフィールドプログラム可能回路アレイ（FPLA：Field Programmable Logic Array）という

ソフトウェアによる論理設計



ANDアレイ

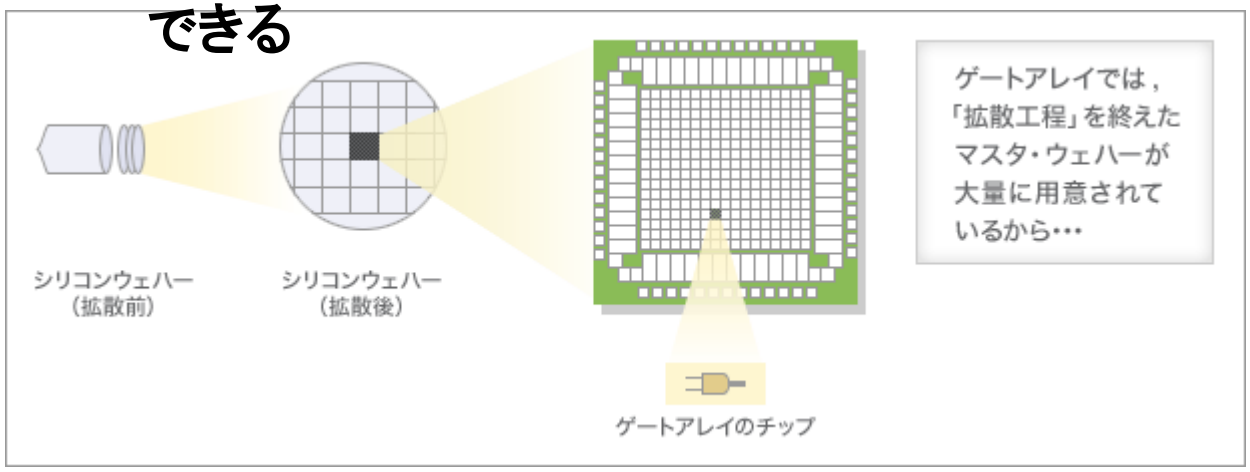


AND-ORアレイ

実際にはこの図のような回路になっている。
●部分がプログラム可能なスイッチ。

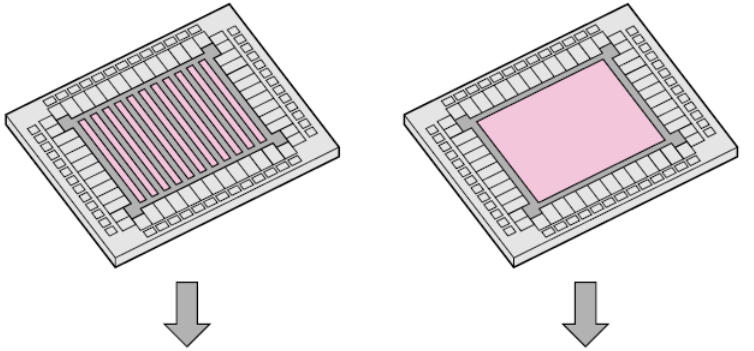
ソフトウェアによる論理設計

- プログラム可能回路（その4）
  - ゲートアレイ（gate array）
    - NANDゲートあるいはNORゲートのいずれかになり得る基本ハードウェア部品をICの中に規則正しく敷き詰めたものをゲートアレイという
    - 未配線のゲートアレイを予め大量生産しておき、配線情報は回路設計者が後で指定する
    - 注文生産と比較して短期間かつ安価に希望する論理回路を入手できる



http://japan.renesas.com/products/soc/asic/gatearray/feature/index.jsp より

Gate Array（ゲートアレイ）LSI



固定の配線領域とゲート領域を持っており，搭載ゲートのほとんどを使用できるが，メモリなどを搭載するのは難しい構造になっている

（a）チャネル構造

大規模なシステムの1チップ化を可能にする．配線領域の幅やゲートの形を自由に決めることができ，メモリを効率良く搭載することもできる

（b）チャネルレス構造

図2　チャネル構造とチャネルレス構造
一般的に現在のゲート・アレイではチャネルレス構造が使用されている

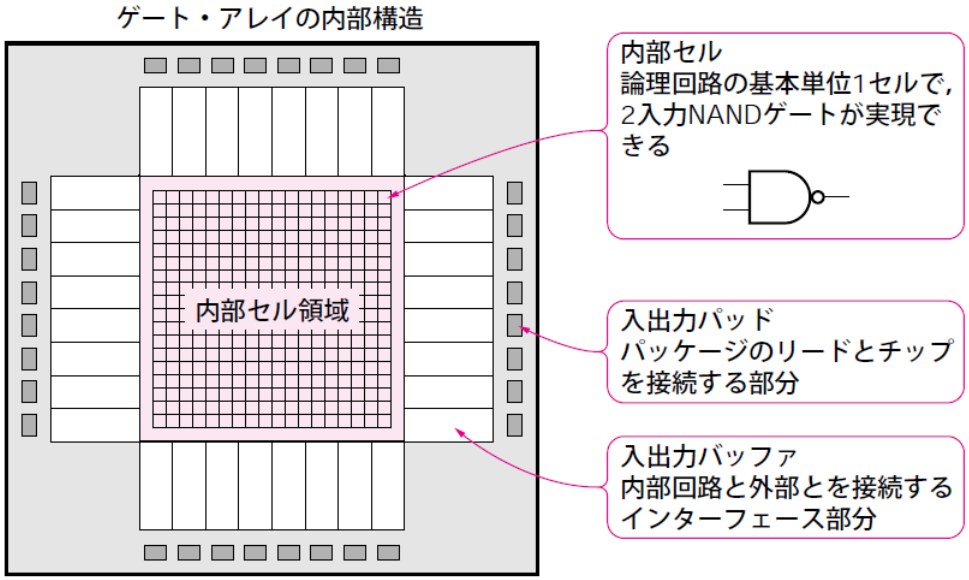
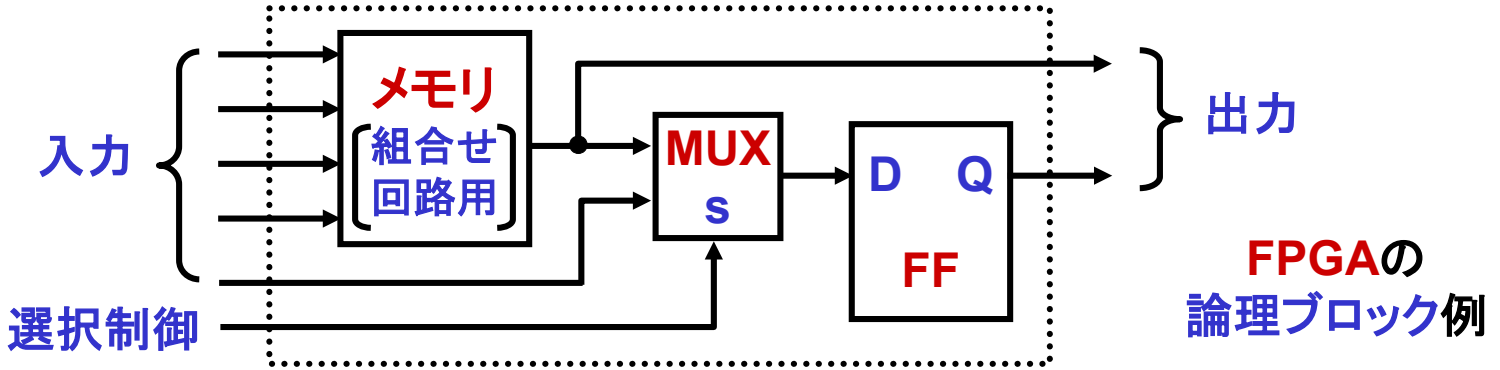


図3　ゲート・アレイの構造
内部セル領域，入出力バッファ領域，入出力パッド領域に分けられる

トランジスタ技術2007年10月号「ゲート・アレイ最新事情」より

ソフトウェアによる論理設計

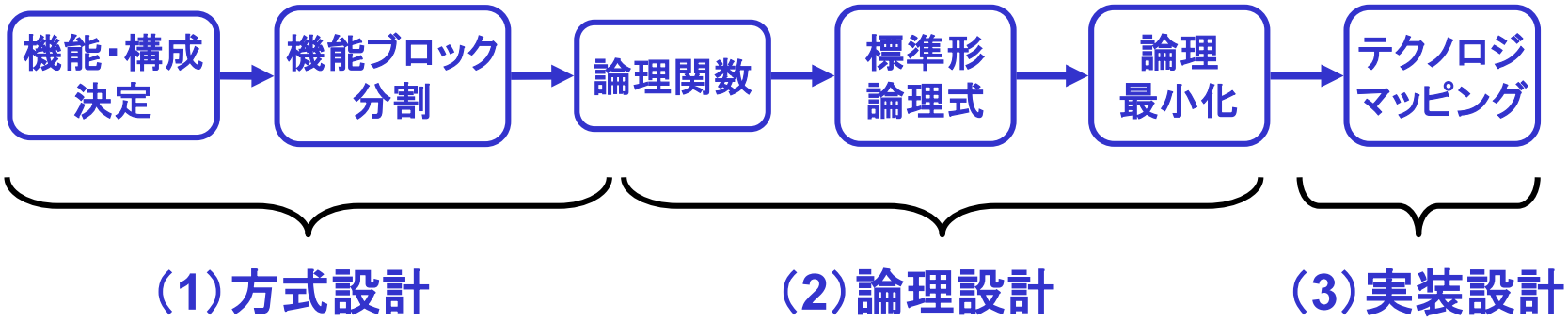
- プログラム可能回路（その5）
  - フィールドプログラム可能ゲートアレイ
    （FPGA：Field Programmable Gate Array）
    - ゲートアレイよりも機能の高い論理ブロックを規則正しくならべたもので、設計者の手元（フィールド＝IC工場出荷後）で所望の論理回路を実現できるものをフィールドプログラム可能ゲートアレイという



- 指定するのは、① 論理ブロックの機能と、② 論理ブロック間の配線
- 長所：フィールドで設計できる
- 短所：論理ブロック間の配線の制限が厳しく、論理ブロックの使用効率もゲートアレイに比べて低くなる

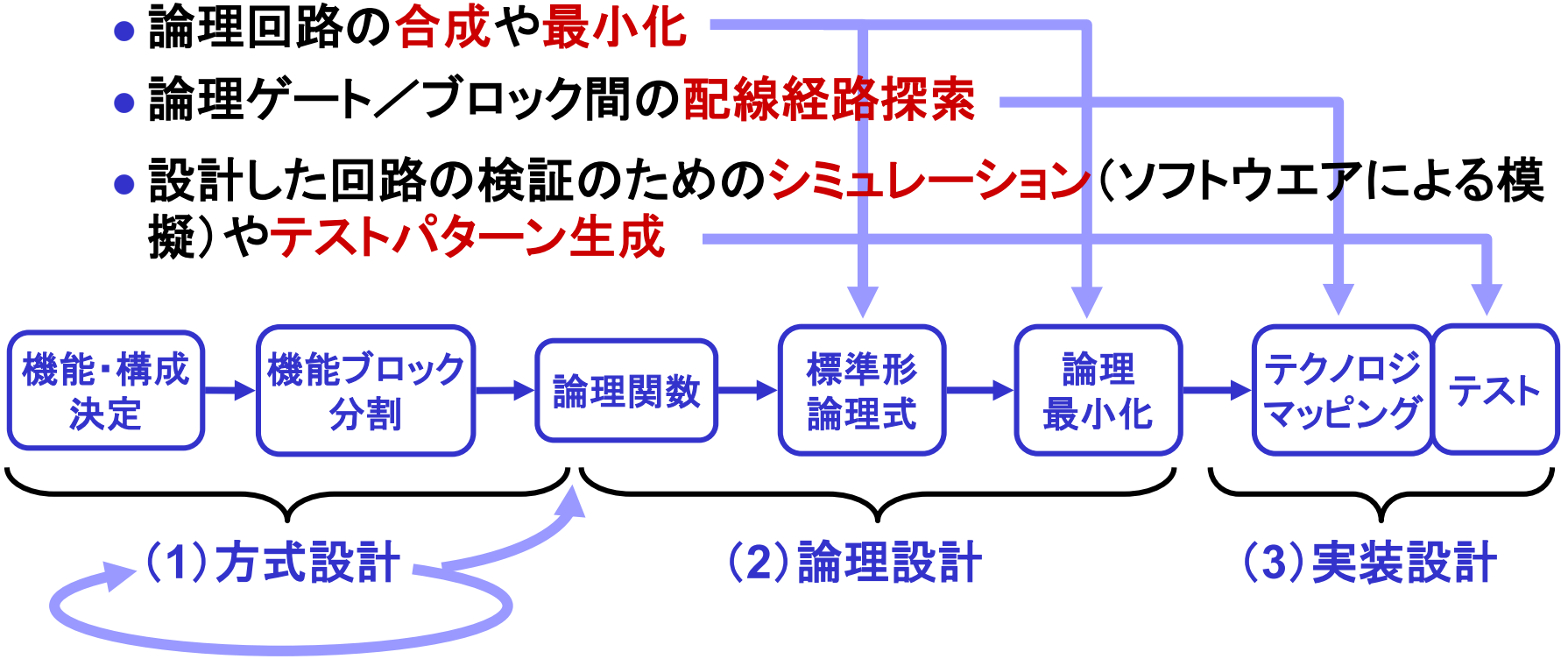
ソフトウェアによる論理設計

- **コンピュータハードウェア設計の流れ**
  - **(1)方式設計**
    - コンピュータの**ハードウェア構成の概略**(機能ブロック)**を決定**
    - ハードウェア機構とソフトウェア機能との**機能分担方式**(＝**コンピュータアーキテクチャ**)を設計することに当たる
  - **(2) 論理設計**
    - (1)で決めたハードウェア機構を**論理回路として合成**
  - **(3) 実装設計**
    - (2)で合成した論理回路の実装部品への**テクノロジマッピング**



ソフトウェアによる論理設計

- コンピュータによるハードウェア設計支援
  - CAD（Computer Aided Design）
    - 論理回路の合成や最小化
    - 論理ゲート／ブロック間の配線経路探索
    - 設計した回路の検証のためのシミュレーション（ソフトウエアによる模擬）やテストパターン生成



- 方式設計をプログラミングのように行えれば、方式設計の能率も向上するし、論理設計以降の下流にスムーズに繋げられる

ソフトウェアによる論理設計

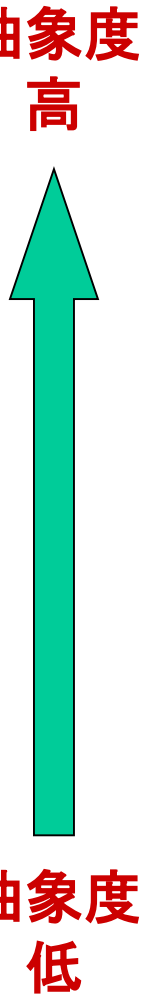
- ハードウェア記述言語
  - 論理回路やハードウェアの動作をプログラムとして記述するプログラミング言語をハードウェア記述言語（HDL：Hardware Description Language）という
  - 具体的には、論理回路やハードウェアの動作を回路図ではなく、方式設計した後のプログラムとして与えると、CADシステムが自動的にそれを最適化し、論理回路を合成してくれる機能である
  - 回路動作をビヘイビア（behavior）というので、HDLの事をビヘイビア記述言語とも言う

ソフトウェアによる論理設計

- ハードウェア記述のレベル
  1. ビヘイビアレベル
     - ハードウェアの動作だけを記述する

       （ハードウェア部品やそれらの接続、動作の細かいタイミン グは記述しない）
  2. レジスタトランスファレベル

     （RTL：Register Transfer Level）
     - 基本的な組合せ回路を論理ブロックとし、その間の接続関係と、その動作タイミングを規定する
  3. 論理ゲートレベル
     - 論理ゲートとそれらの組合せ（配線）を規定する
     - 回路図をプログラムとして書き直しただけのレベル



ソフトウェアによる論理設計

- HDLでの記述例（半加算器）

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
  port (
      a, b: in std_logic;
      s, co: out std_logic);
end half_adder;

architecture arch of half_addrer is
   signal w0, w1, w2: std_logic;
begin
   w0 <= a and b;
     ..........
```

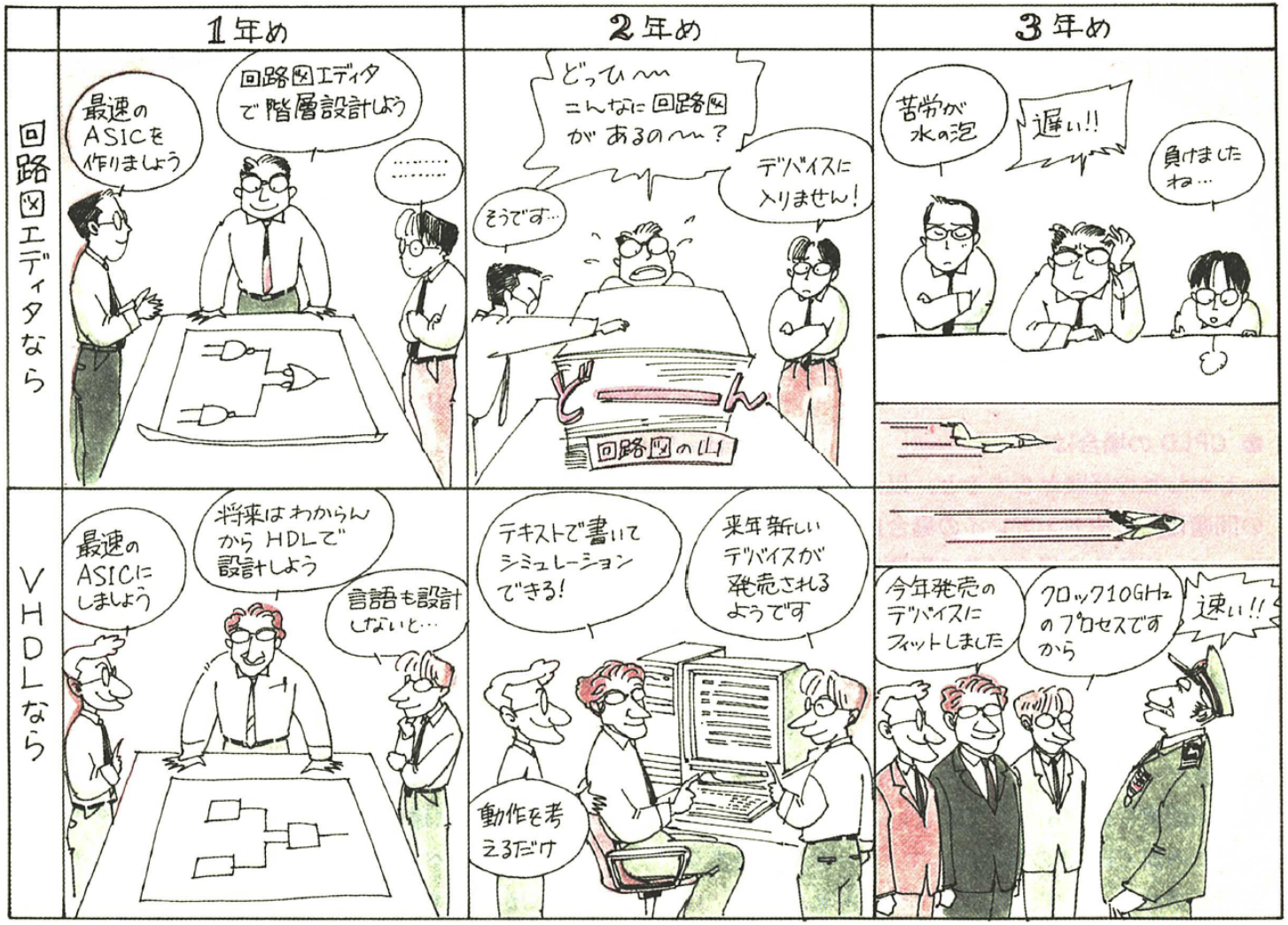
VerilogHDL

```
module half_adder (s, co, a, b);
    input a, b;
    output s, co;

   wire w0, w1, w2;
   assign w0 = a & b,
          w1 = w0,
          w2 = a | b,
          s = w1 & w2,
          co = w0;
endmodule
```

ソフトウェアによる論理設計

〈イラスト〉VHDL なら未来のデバイス・テクノロジが利用できる



トランジスタ技術1997年12月号より