MWE基礎講座

2010年12月10日

2021年4月20日　追加

デジタル･アシスト･アナログ技術のための デジタルCMOS回路設計　再入門

群馬大学大学院 工学研究科 電気電子工学専攻

小林春夫

koba@gunma-u.ac.jp

内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例１：　SAR ADC+分散型積和演算回路
- 事例２：　自己校正機能をもったTDC回路
- 事例３：　冗長アルゴリズムを用いたSAR ADC
- 最後に

内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

セミナーの目的・目標

- アナログRF回路設計では
  デジタルアシスト・アナログ技術が重要
- 比較的小規模のCMOSデジタル回路
  設計の基本をレビューする
- 3つの事例を紹介する

内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

アナログ信号とデジタル信号

アナログ信号

連続的な信号

例：　自然界の信号（音声、電波）、アナログ時計

「坂道」

デジタル信号

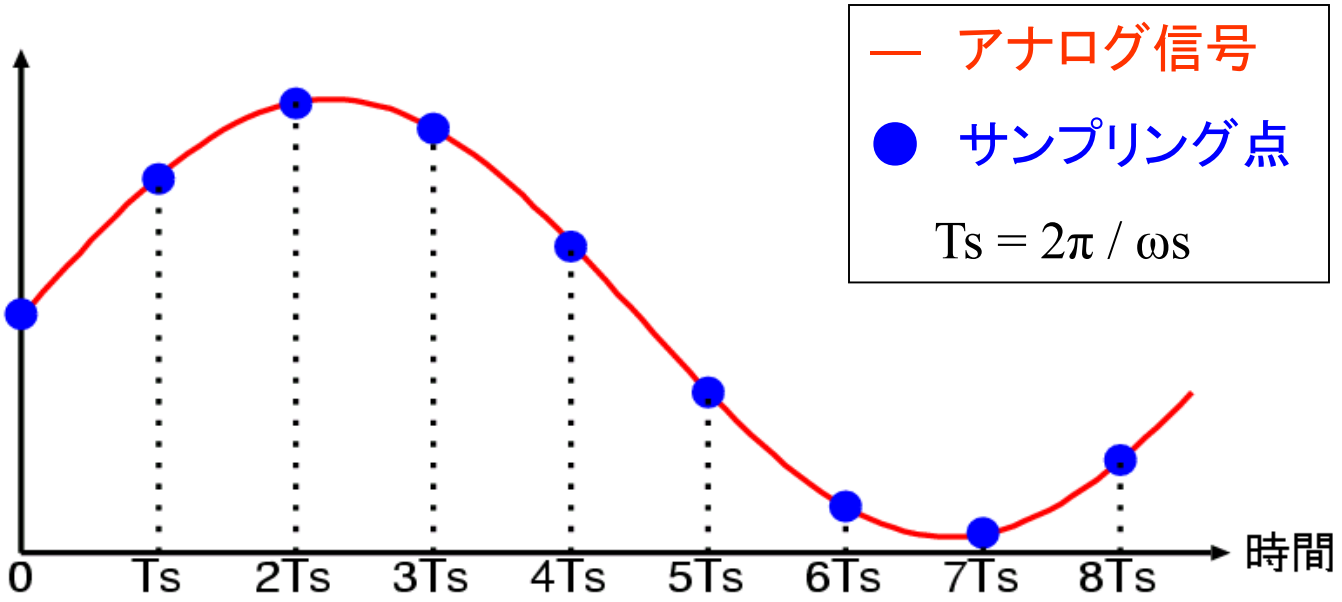
離散的・数値で表現された信号

例：コンピュータ内での２進数で表現された信号

デジタル時計

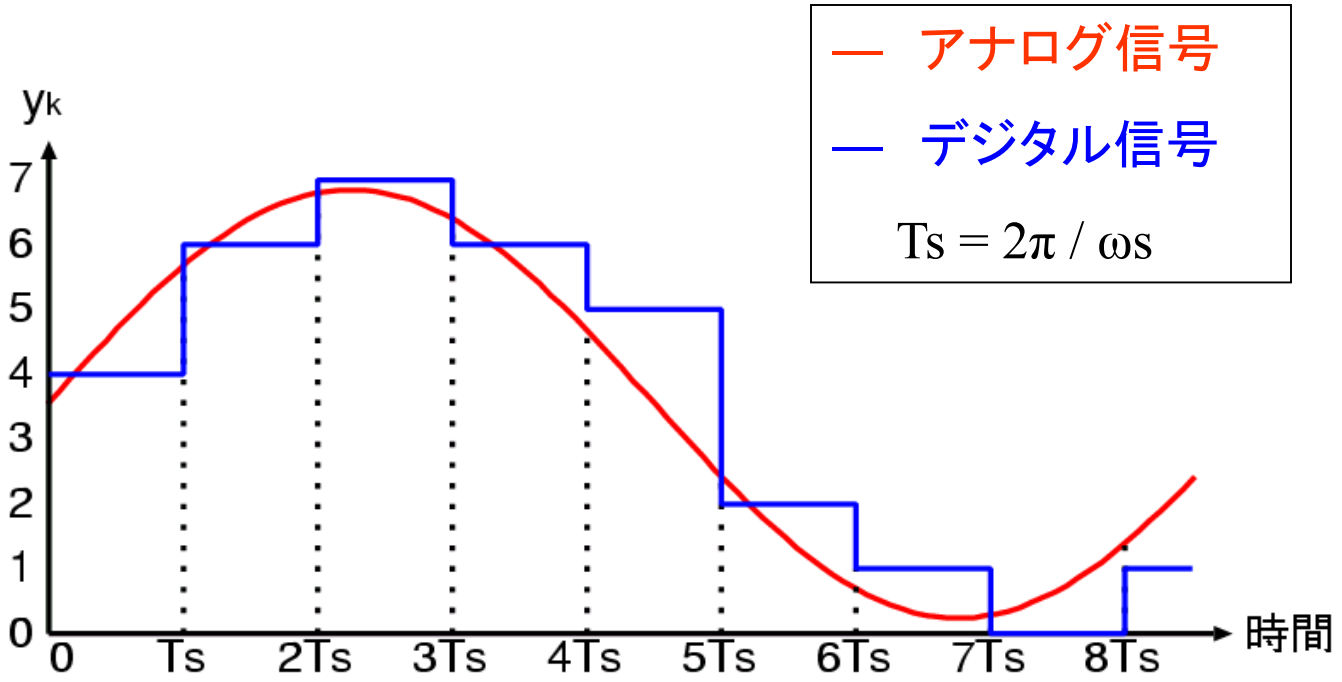
「階段」

デジタル信号の特徴（1）
時間の量子化（サンプリング）



一定時間間隔のデータを取り、間のデータは捨ててしまう。

デジタル信号の特徴(2)
空間の量子化（信号レベルの数値化）



デジタル信号はアナログ信号レベルを
四捨五入(または切り捨て)

デジタル回路と2進数

- 人間はなぜ10進数を使うか？

  → 手の指が10本あるから。

- デジタルではなぜ2進数を使うか？

  → 2つの状態は技術的に容易かつ安定して実現可能。

例：　電圧の高いと低い

　　電流の流れる状態と流れない状態

　　パルスのあるとなし。

１０進数と2進数

10進	2進	10進	2進
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

例　2進数　1011　を
　１０進数に変換
1x2x2x2+0x2x2+1x2+1
=11

2進数

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

$2^9 = 512$

$2^{10} = 1{,}024 = 1K$

(参考　1,000=1k)

１６進数、８進数とデジタル

10進	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8進	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24
16進	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14

- 人間はなぜ１０進数を使うか？

⇒ 手の指が１０本あるから。

- デジタルコンピュータは２進数が基本。

ではなぜ１６進数、８進数を使うか？

⇒ ２進数と１６進数、８進数は相性がよいから。

8進数と2進数の変換

8進	2進 (4 2 1)
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

例　8進4桁　3724

●10進に変換

3x8x8x8+7x8x8+2x8+4

計算が必要

●2進に変換

011 111 010 100

左表から機械的に得られる

１６進数と2進数の変換

16進	2進	16進	2進
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

例　16進で3桁
　A46
2進数に変換
1010 0100 0110
左表から機械的
に得られる

4ビットの場合

負の数の表現（2の補数）

符号無	符号付	2進	符号無	符号付	2進
ud	sd	b3 b2 b1 b0	ud	sd	b3 b2 b1 b0
0	0	0000	8	-8	1000
1	1	0001	9	-7	1001
2	2	0010	10	-6	1010
3	3	0011	11	-5	1011
4	4	0100	12	-4	1100
5	5	0101	13	-3	1101
6	6	0110	14	-2	1110
7	7	0111	15	-1	1111

4ビットの場合

負の数の表現（2の補数）

+5　(2進表現　0 1 0 1)から　-5 の2進表現を得る

0 1 0 1 のビット反転
1 0 1 0 を得る。

それに 1 を加える

```
  1 0 1 0
+) 0 0 0 1
-----------
  1 0 1 1   ←  -5 の2進表現
```

4ビットの場合

2の補数表現から10進数へ

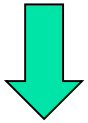
符号無　ud = b_3 x2x2x2 + b_2 x2x2 + b_1 x2 + b_0

符号付　sd = − b_3 x2x2x2 + b_2 x2x2 + b_1 x2 + b_0

Example:　2進表現　1 0 1 1

符号無　11

符号付　-5



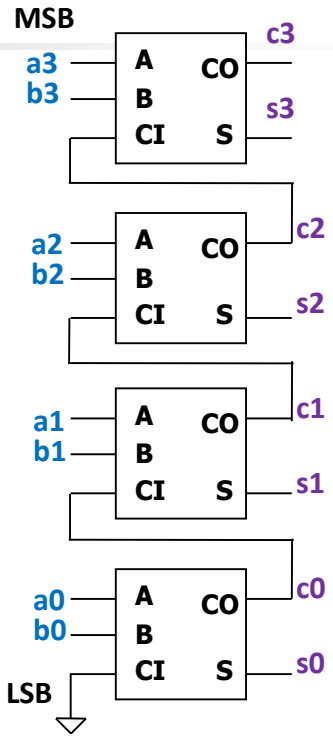
ud = 8 + 2+ 1 = 11

sd = -8 + 2 + 1 = -5

なぜ2の補数表現を用いるのか

2進演算	符号無	符号付
0 0 1 1	3	3
＋ 1 0 1 0	＋ 10	＋ -6
1 1 0 1	＝ 13	＝ -3

2の補数表現をすれば
符号付、符号無で
同じ2進演算アルゴリズム
同じハードウェアを
使用可能



(a3 a2 a1 a0) =(0 0 1 1)
(b3 b2 b1 b0)=(1 0 1 0)

(s3 s2 s1 s0) =(1 1 0 1)
(c3 c2 c1 c0) =(0 0 1 0)

負の数の表現（2の補数）

4,5ビットの場合

ビット数の拡張

4 ビット

符号無 ud	符号付 sd	2進 b3 b2 b1 b0
8	-8	1000
9	-7	1001
10	-6	1010
11	-5	1011
12	-4	1100
13	-3	1101
14	-2	1110
15	-1	1111

5 ビット

異なる !!

符号無 ud	b4 b3 b2 b1 b0	符号付 sd	b4 b3 b2 b1 b0
8	01000	-8	11000
9	01001	-7	11001
10	01010	-6	11010
11	01011	-5	11011
12	01100	-4	11100
13	01101	-3	11101
14	01110	-2	11110
15	01111	-1	11111

2進数　固定小数点表現

2進数　1011.011

10進数

1x2x2x2 +0x2x2+1x2+1

+0x(1/2)+1x[1/(2x2)]+1x[1/(2x2x2)]

= 8+2+1+(1/4)+(1/8)

今から320年前、1692年のパリ

哲学者、数学者、科学者　ライプニッツ

（**Gottfried Wilhelm Leibniz**）

「全ての数を1と0によって表す驚くべき表記法」

を提案。

王立科学アカデミーに理解されず

学会誌にも掲載されなかった。

「誰も予想しなかった卓越した用途がありはずだ」

と語る。

（慶應義塾大学　青山友紀先生資料より）

ゴットフリート・ヴィルヘルム・ライプニッツ（Gottfried Wilhelm Leibniz, 1646年 - 1716年）

ライプニッツは哲学者、数学者、科学者など幅広い分野で活躍した学者・思想家として知られているが、また政治家であり、外交官でもあった。17世紀の様々な学問（法学、政治学、歴史学、神学、哲学、数学、経済学、自然哲学（物理学）、論理学等）を統一し、体系化しようとした。その業績は法典改革、モナド論、微積分法、微積分記号の考案、論理計算の創始、ベルリン科学アカデミーの創設等、多岐にわたる。ライプニッツは稀代の知的巨人といえる。

60進法

- 60 多くの約数を持つ
  - ⇨ 2, 3, 4, 5, 6, 10, 12, 15, 20, 30

- １時間は60分

学会発表でのプログラム　１時間で

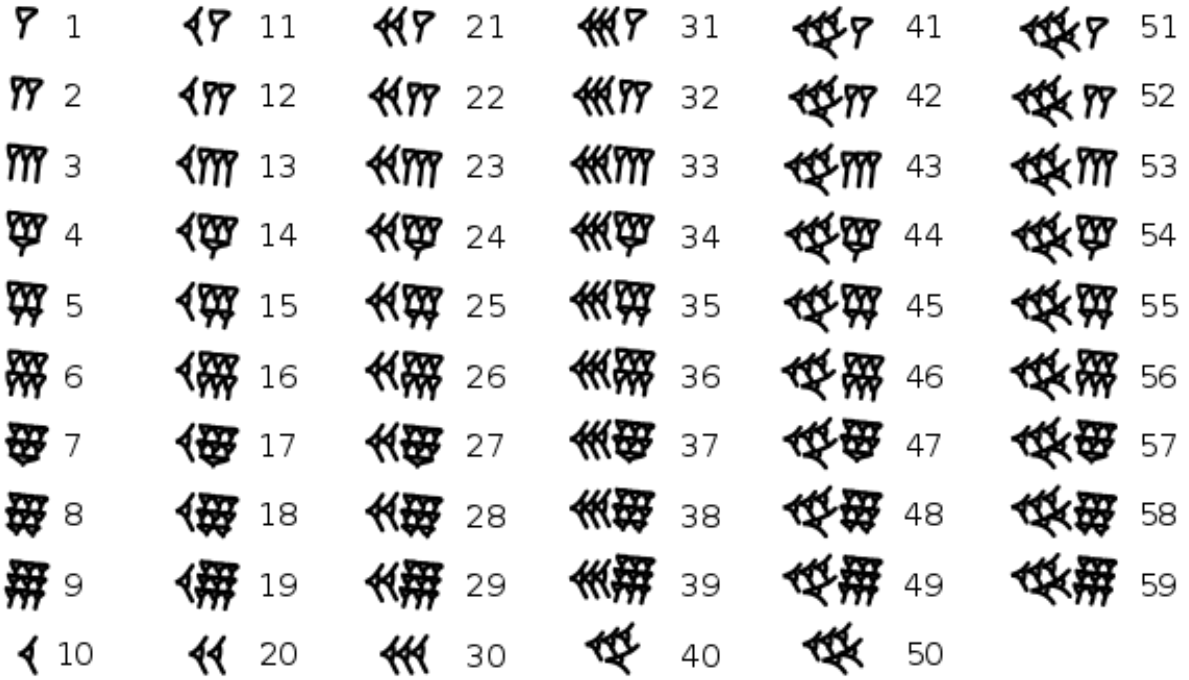
２人のプレゼン　一人当たり 30分

3人　20分

4人　15分

5人　12分

60進法を用いた古代バビロニア



バビロニア数字

ブール代数 （2進数のための数学）

- 論理変数　A, B, C, …, Z
- 値は 1 または 0 をとる。（2値変数）

正論理　　1:　真 (true)
　　0:　偽 (false)
負論理　　0:　真 (true)
　　1:　偽 (false)

真にGND線を用いたいとき等に負論理を使用。

キーワード:

論理否定、論理積、論理和、排他的論理和、
真理値表、ド・モルガンの法則

ブール代数の創始者 George Boole (1815-1864、英)

コンピュータを理論的に支えるブール代数を提唱。

デジタル回路の設計には必須の知識。
デジタル回路は、電圧の High, Lowのみで
情報を演算するため、組み合わせ回路は
ブール代数での論理式で書き表わせる。

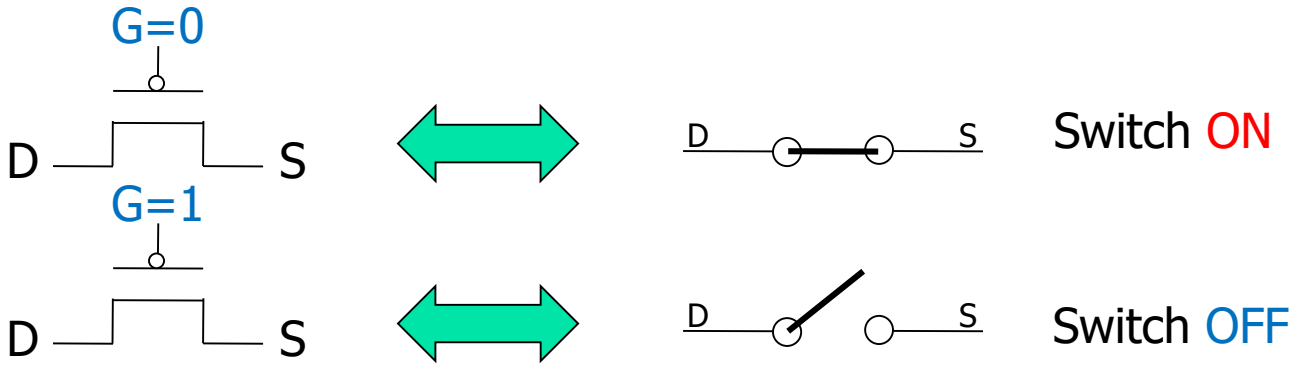
「記号の操作は計算の明白な要素として取り扱え
数量から切り離すことができる」

内　容

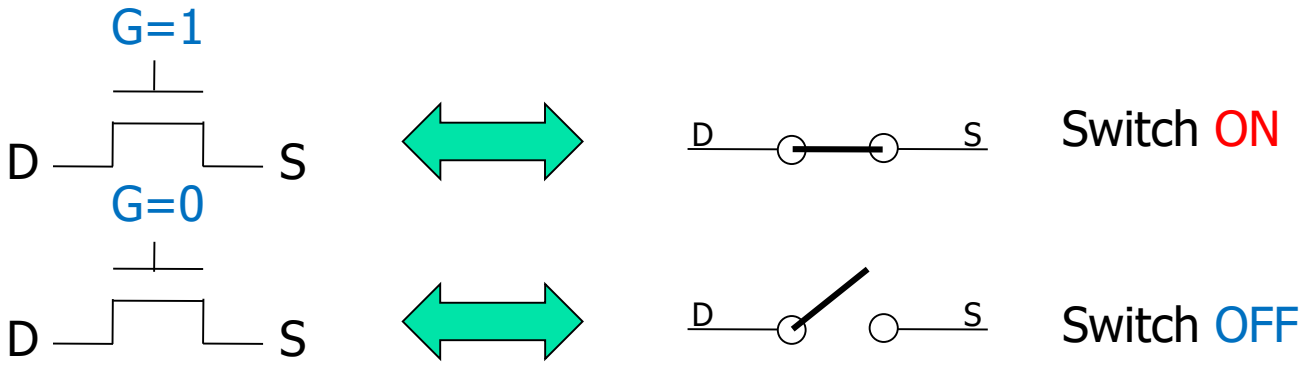
- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

PMOS, NMOS スイッチ

(1) PMOS

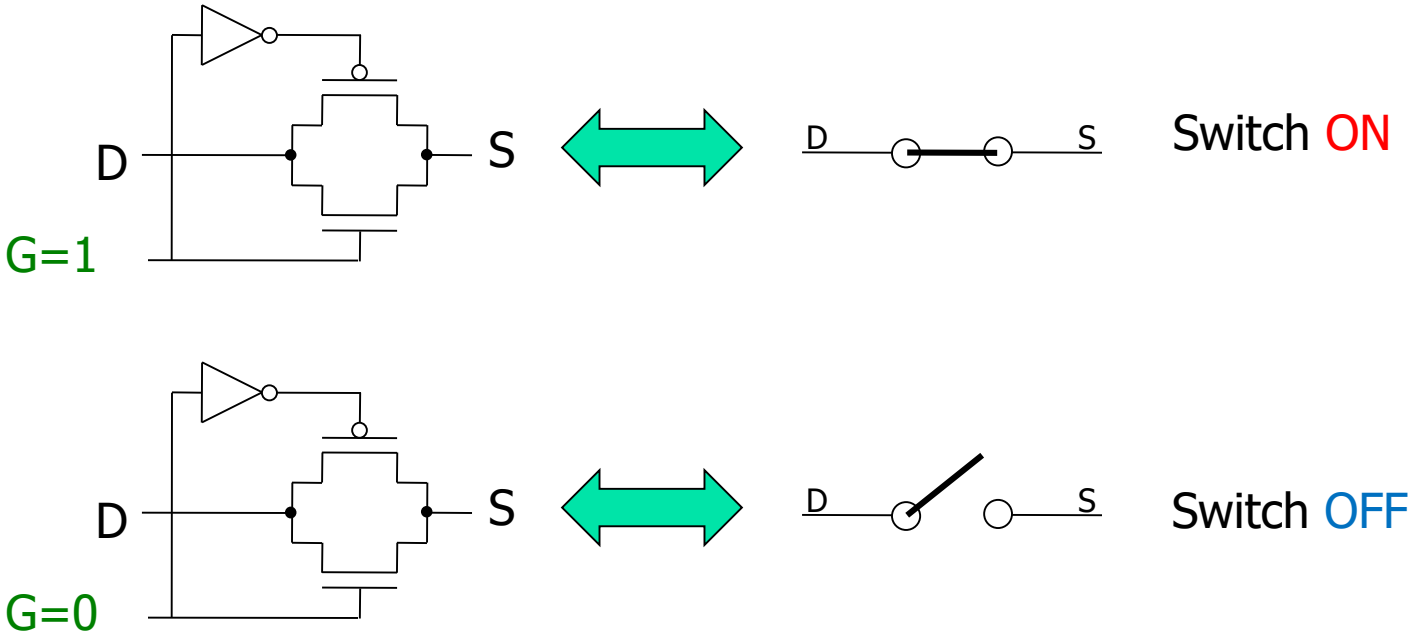


(2) NMOS



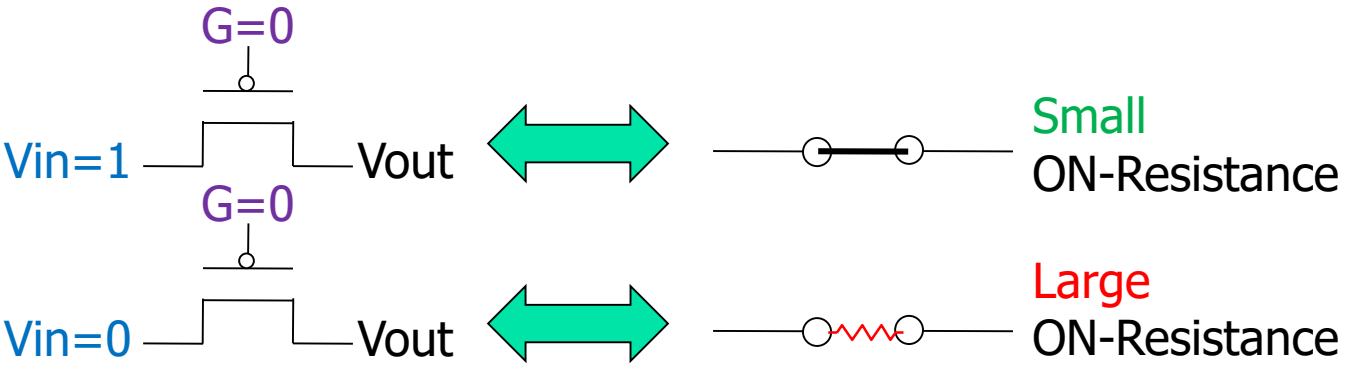
CMOSスイッチ

(3) CMOS



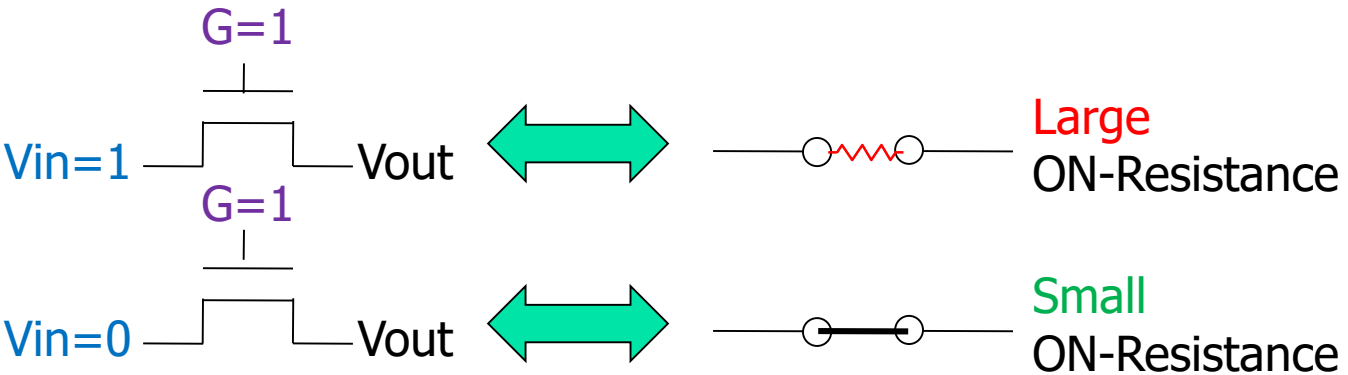
PMOS，NMOSスイッチのオン抵抗

(1) PMOS



PMOSは正電源側で用いる

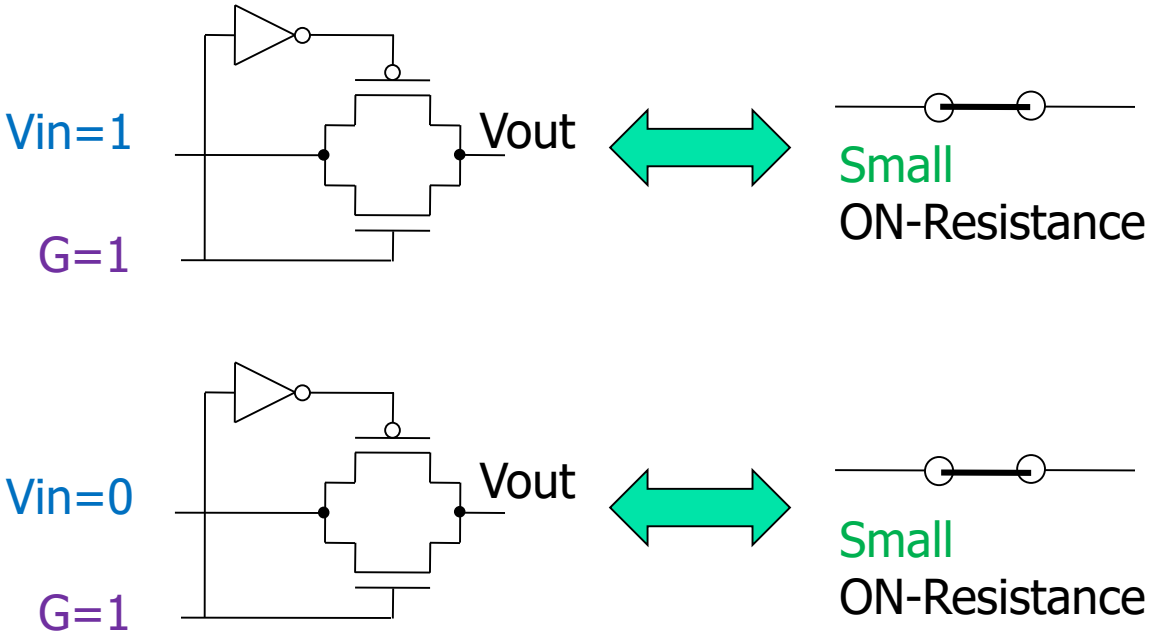
(2) NMOS



NMOSはGND側で用いる

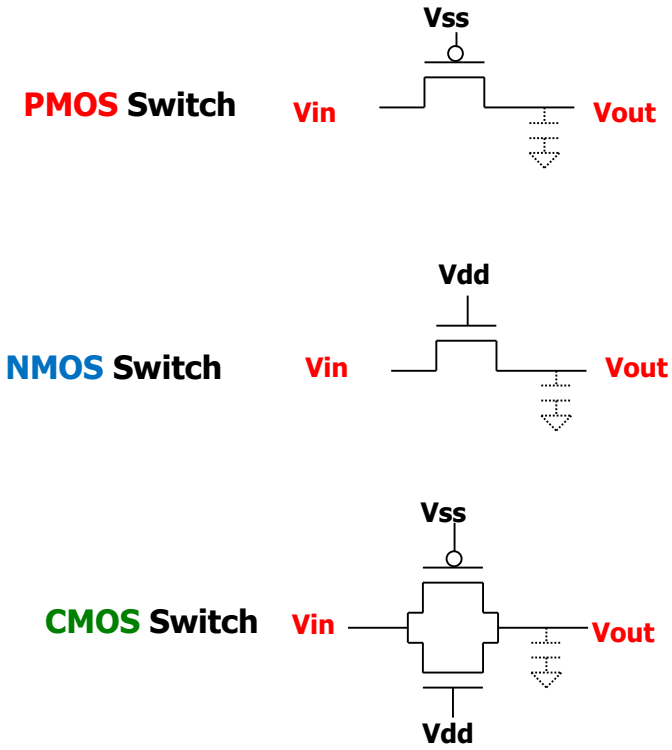
CMOSスイッチのオン抵抗

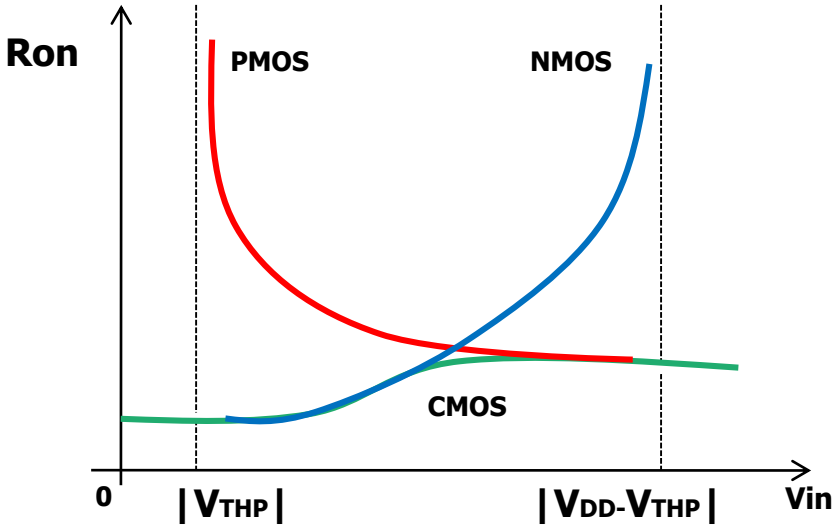
(3) CMOS



CMOSは
GND側でも
正電源側でも
オン抵抗が
小さいが、
トランジスタ数
が増える。

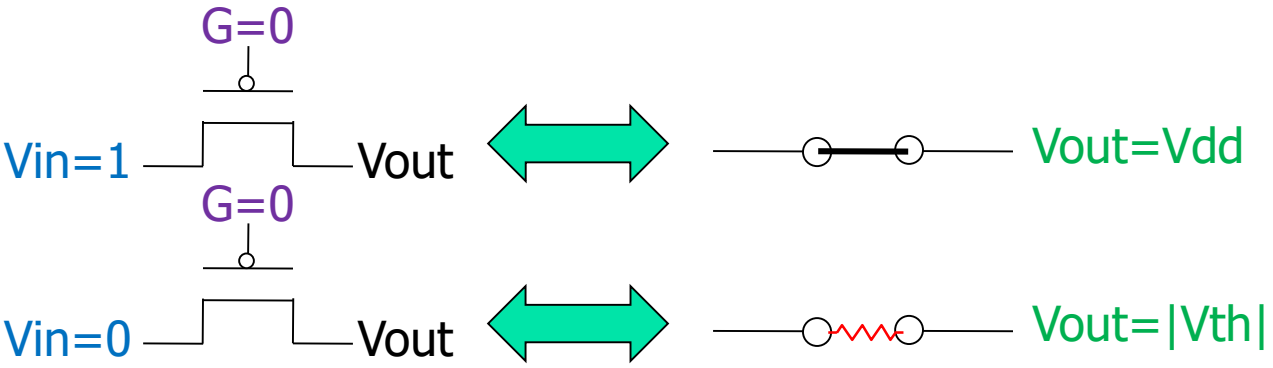
PMOS, NMOS, CMOSスイッチの入力電圧に対するオン抵抗





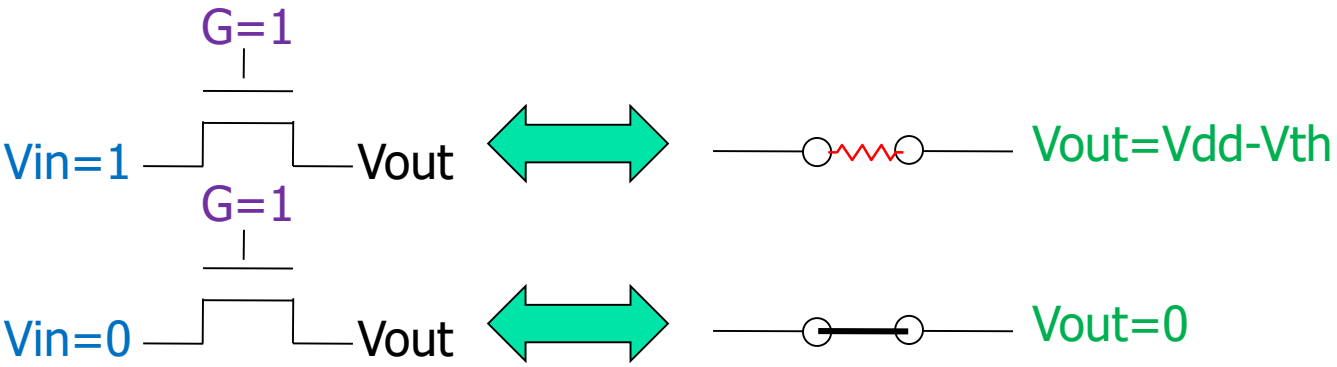
PMOS，NMOSスイッチの出力電圧

(1) PMOS



PMOSは
Vout は
GNDまで
下がらない。

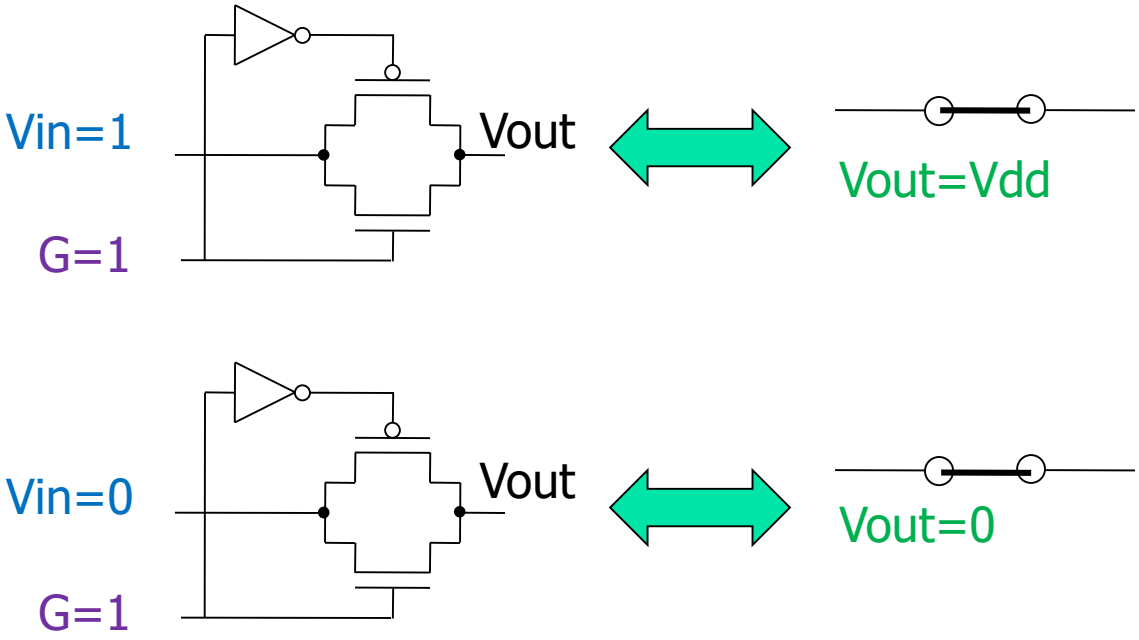
(2) NMOS



NMOSは
Voutが
Vdd まで
上がらない。

CMOSスイッチの出力電圧

(3) CMOS



CMOSでは
出力電圧Voutが
GND, Vdd 間を
フルスイング。

論理否定(NOT)

論理変数　A, Z

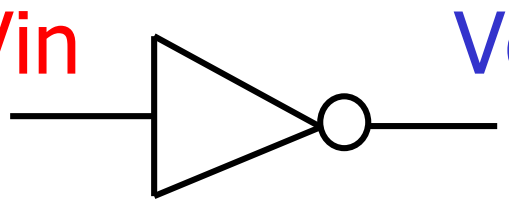
A: 入力,　Z: 出力

$Z = \overline{A}$

真理値表

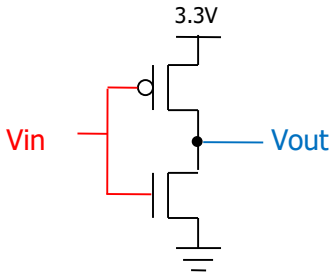
A	Z
0	1
1	0

NOT を実現する回路 ⇒ インバータ回路

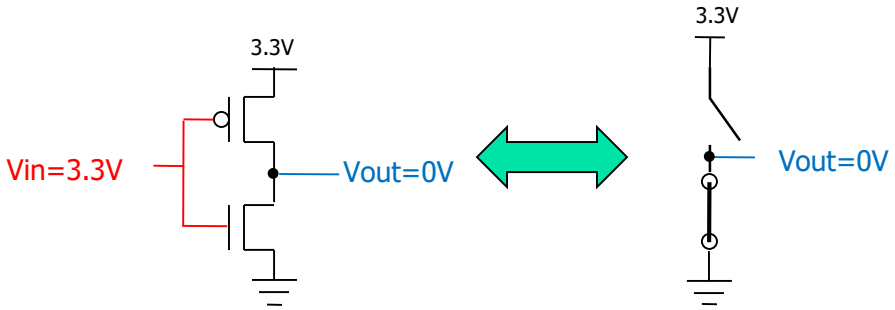


CMOSインバータ回路

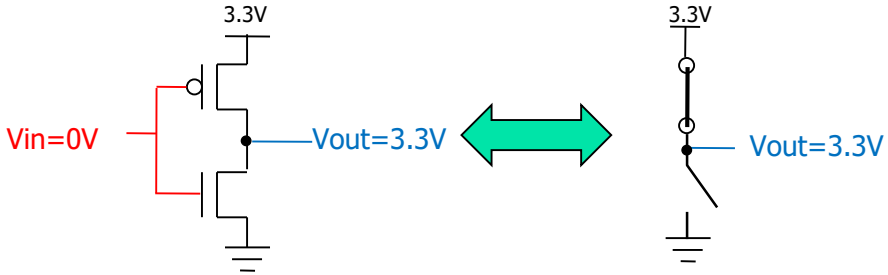
Inverter



a) When Vin=1 (3.3V)



b) When Vin=0



NAND　(NAND = AND + NOT)

論理変数　A,B, Z

A,B: 入力,　Z: 出力

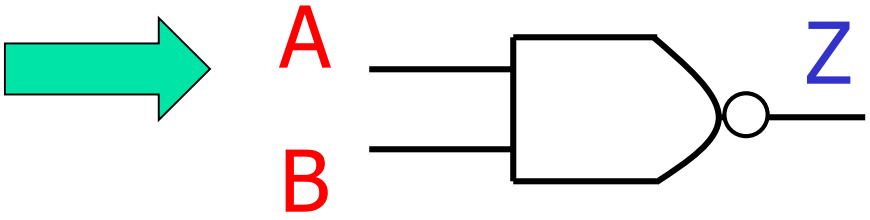
$$Z = \overline{A \cdot B}$$

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

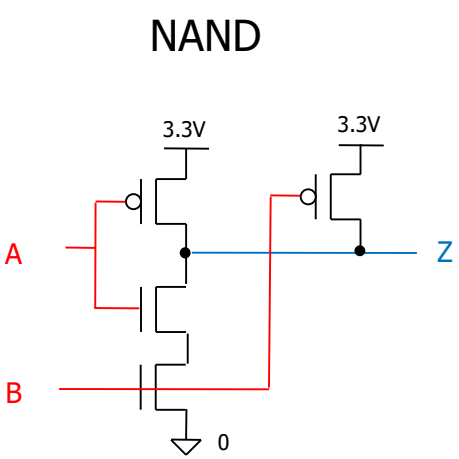
真理値表

NANDを実現する回路

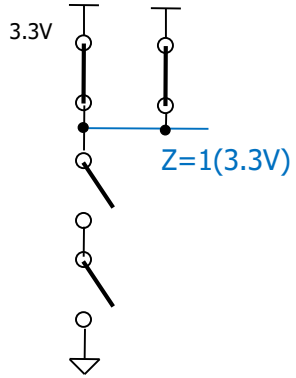
NAND回路



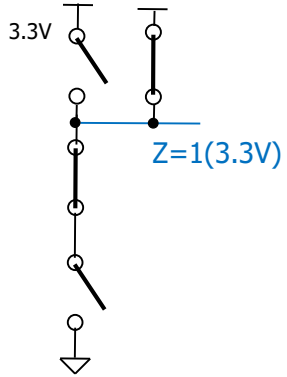
CMOS NAND回路



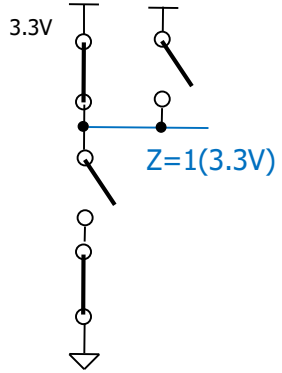
a) When A=0,B=0



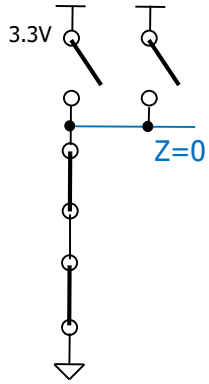
b) When A=1,B=0



c) When A=0,B=1



d) When A=1,B=1



NOR　（NOR = OR + NOT）

論理変数　A,B, Z

A,B: 入力,　Z: 出力

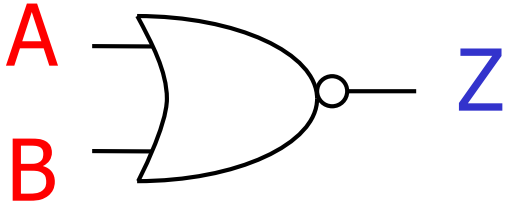
$Z = \overline{A+B}$

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

真理値表

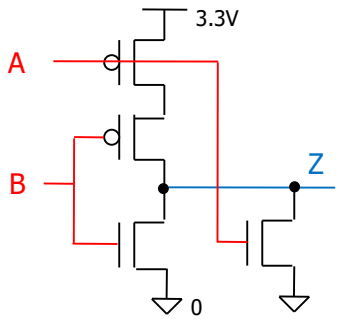
NORを実現する回路

NOR回路

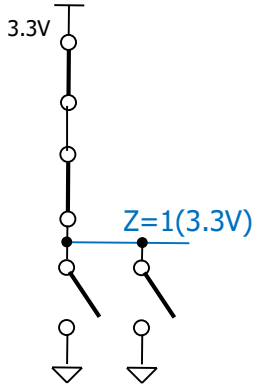


CMOS NOR回路

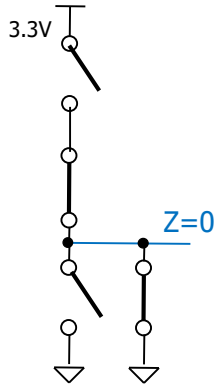
NOR回路



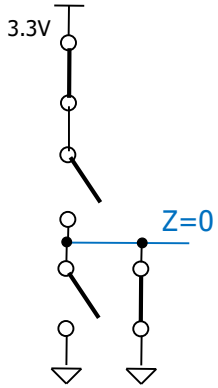
a) When A=0,B=0



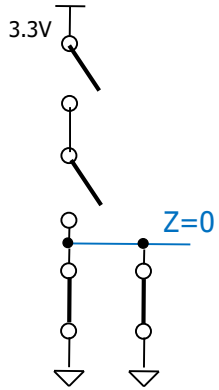
b) When A=1,B=0



c) When A=0,B=1



d) When A=1,B=1



論理和と2進数の加算

論理和　Z=　A＋B

右の真理値表は

論理和の定義

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

真理値表

論理和　　1+1=1

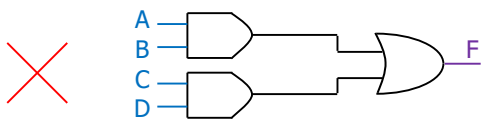
2進数の加算　1+1=10

同じ 記号 ＋ でも意味が異なることに注意。

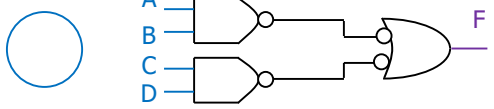
NAND、NOR回路を使用する。AND，OR回路の使用は避ける。

AND = NAND + Inverter
OR = NOR + Inverter
で構成 （非効率）

例：
F= A B + C D の回路実現

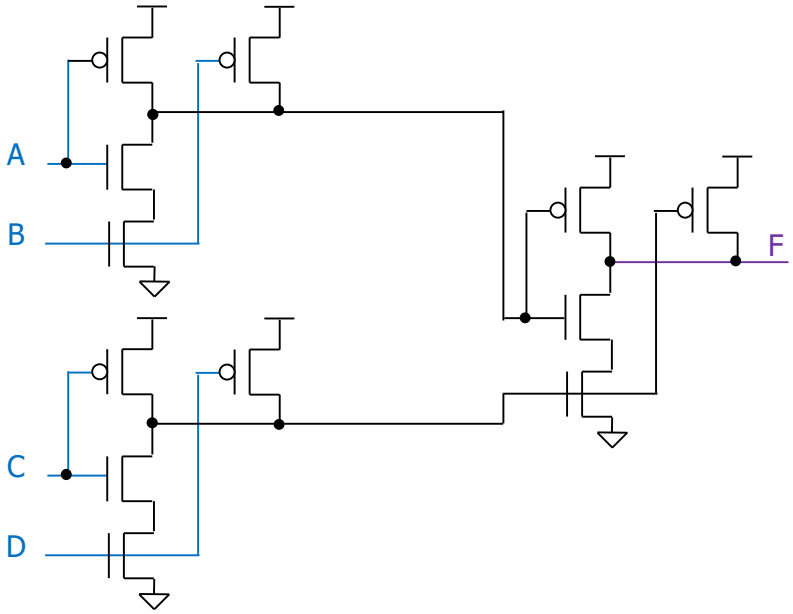


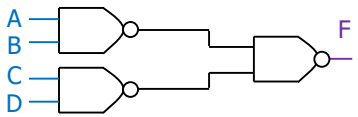
負論理



ド・モルガンの法則

$\overline{A}+\overline{B} = \overline{A \cdot B}$



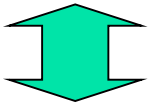


オーガスタス･ド･モルガン
Augustus De Morgan, 1806-1817

インド生まれのイギリスの数学者
ド･モルガンの法則を発案した。

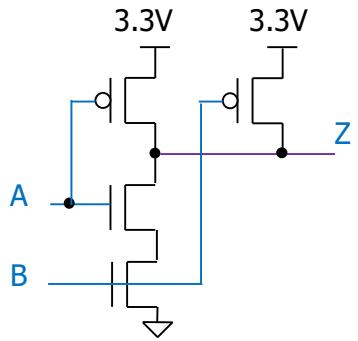
$$\overline{A \cdot B} = \overline{A} + \overline{B}, \quad \overline{A+B} = \overline{A} \cdot \overline{B}$$

「私の身長は 160 cm 以上であり、
かつ私の体重は 50 kg 以上である」ではない。



「私の身長は 160 cm 未満であるか、
または私の体重は 50 kg 未満である」

2入力NAND回路の
立上り、立下り時間はほぼ等しい
NAND回路は　NOR 回路より良い

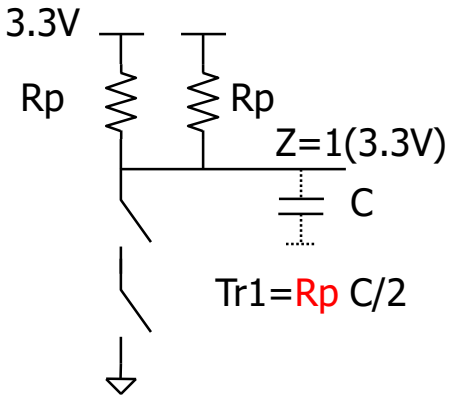


立上り時定数 Tr = Rp C
立下り時定数 Tf = 2 Rn C

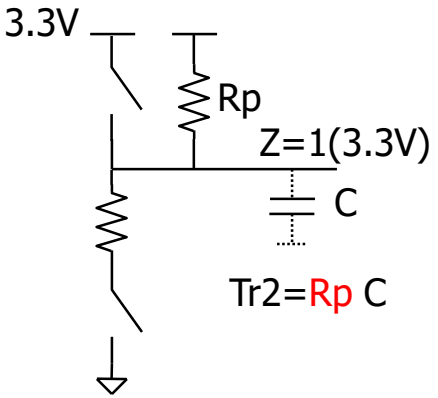
Rp = 2～3 Rn

Tr = Tf

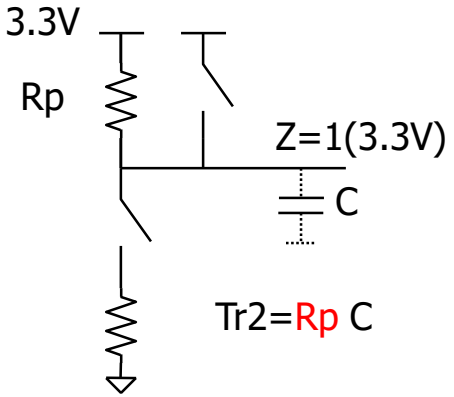
a) When A=0,B=0



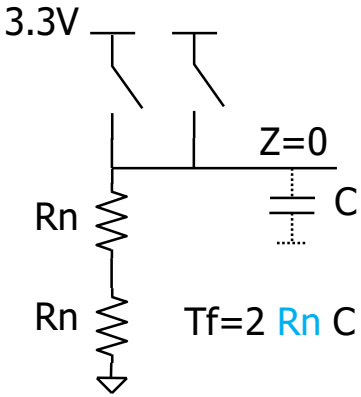
b) When A=1,B=0



C) When A=0,B=1



D) When A=1,B=1



複合論理CMOS回路

	論理積	論理和
PMOS	並列	直列
NMOS	直列	並列

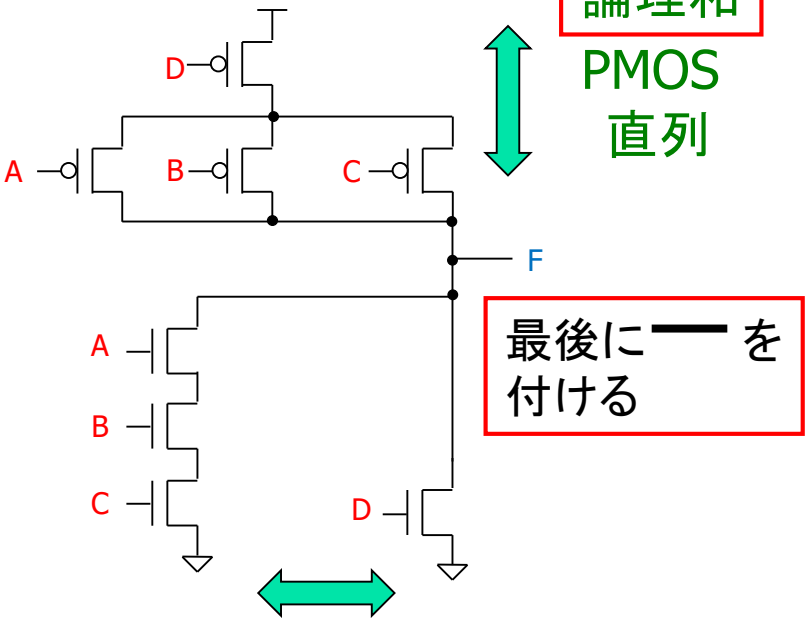
複合論理CMOS回路　例

$F = \overline{A \cdot B \cdot C + D}$

PMOS
並列

論理積

NMOS
直列



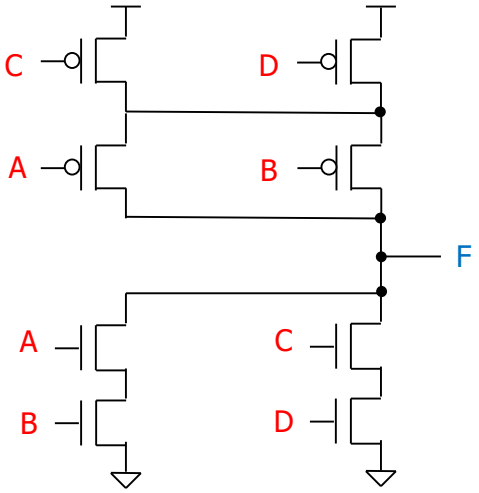
論理和
PMOS
直列

最後に ‾ を
付ける

NMOS
並列

論理和

$F = \overline{A\,B + C\,D}$



マルチプレクサ

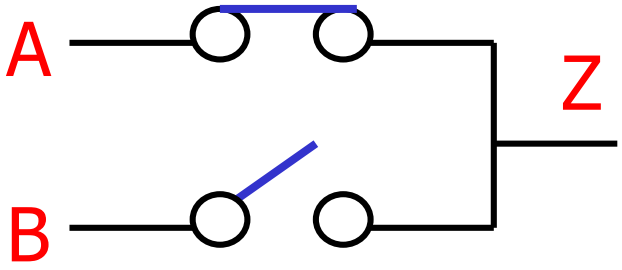
論理変数　A,B,S,Z

A,B,S:入力,　Z:出力

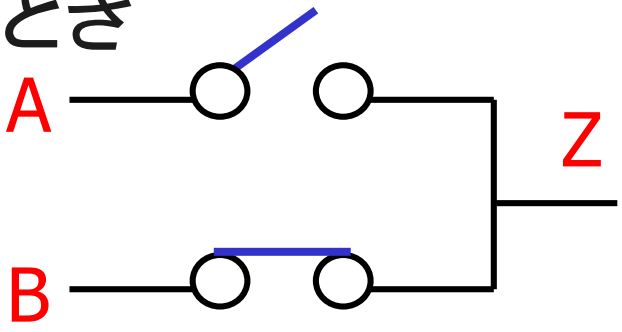
S	Z
0	A
1	B

真理値表

S=0 のとき

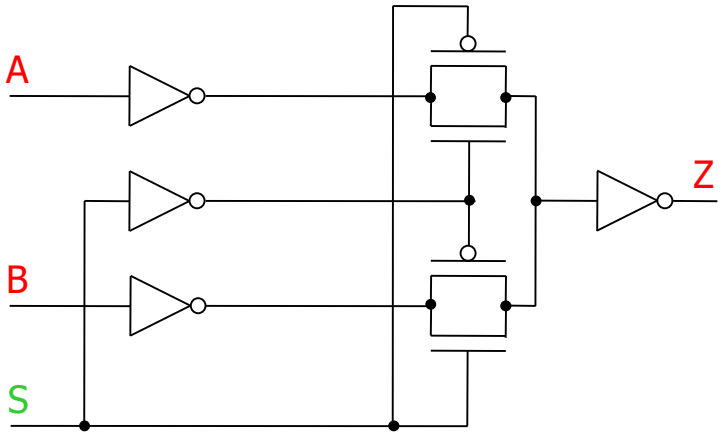


S=1 のとき

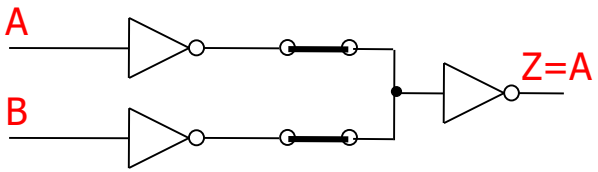


CMOS マルチプレクサ回路

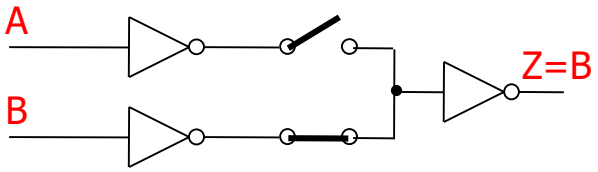
Multiplexer



a) When S=0



b) When S=1



排他的論理和(EXOR)

論理変数　A,B, Z

A,B:入力,　Z:出力

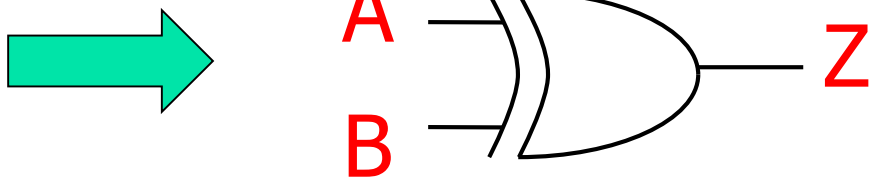
Z=　A⊕B

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

真理値表

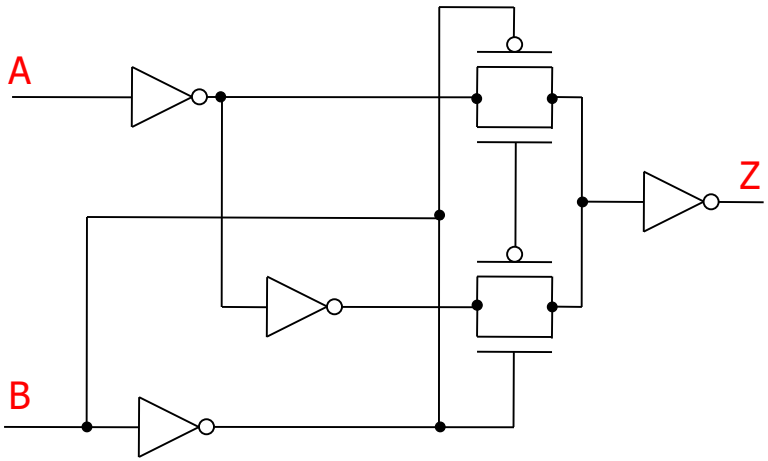
EXORを実現する回路

EXOR回路

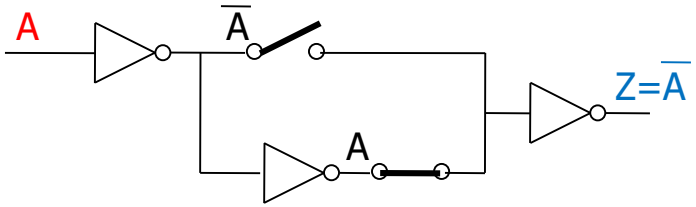


CMOS EXNOR回路

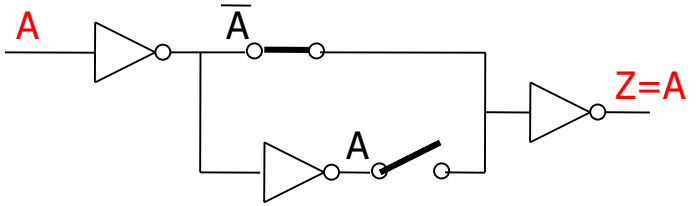
$Z = A\,B + \overline{A}\,\overline{B}$



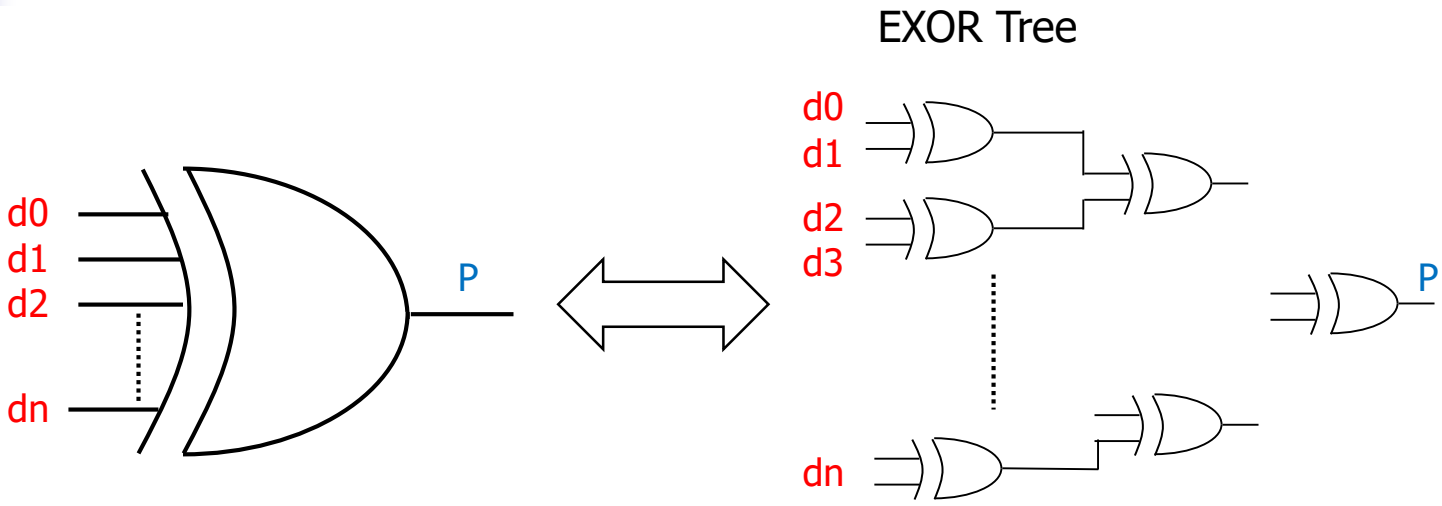
a) When B =0



b) When B =1



多入力EXOR 回路とパリティ



$P = d0 \oplus d1 \oplus d2 \oplus \cdots\cdots \oplus dn$

Parity

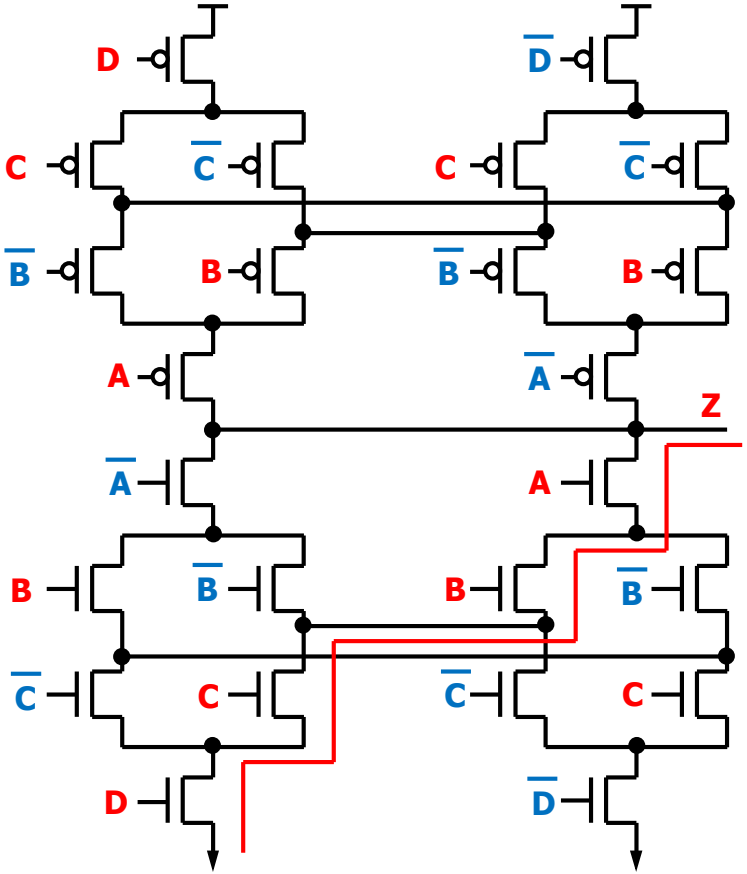
d0, d1, … dn の1の数が奇数個 ⟹ P=1

偶数個 ⟹ P=0

メモリ、通信のデータチェック等に使用

４入力EXORの実現回路

$Z = A \oplus B \oplus C \oplus D$

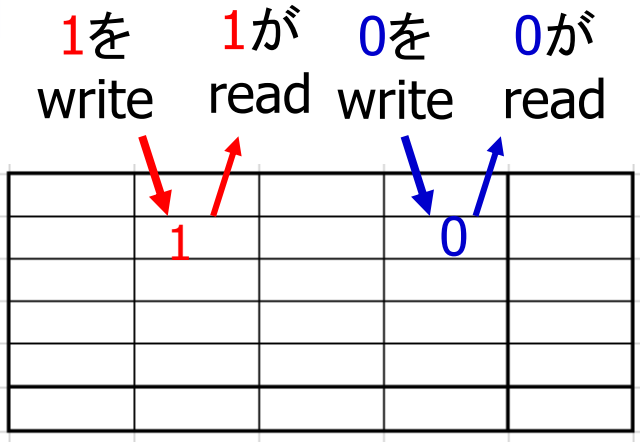


A	B	C	D	Z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

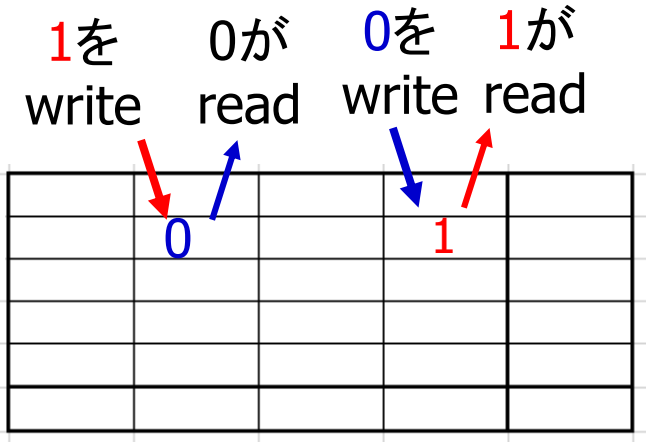
ソフトエラー

LSI内部回路が何らかの理由で
一時的な誤作動を起こし、
記録された内容が破壊されること

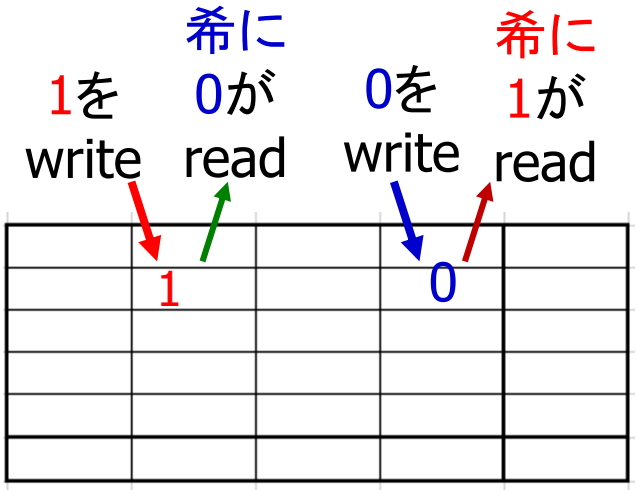
正常な
メモリ



故障



ソフト
エラー



LSI微細化・低電源電圧化

↓

ソフトエラー率　高

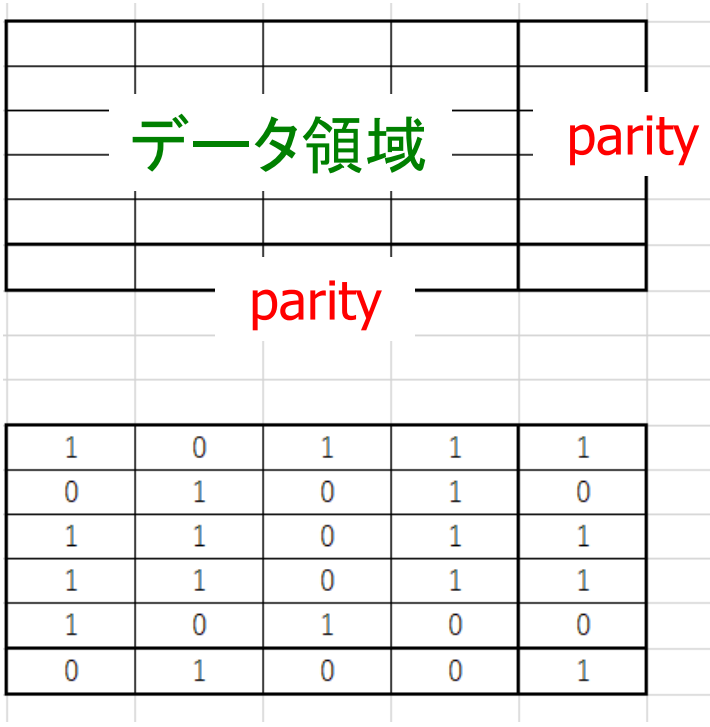
宇宙線もソフトエラーを誘発

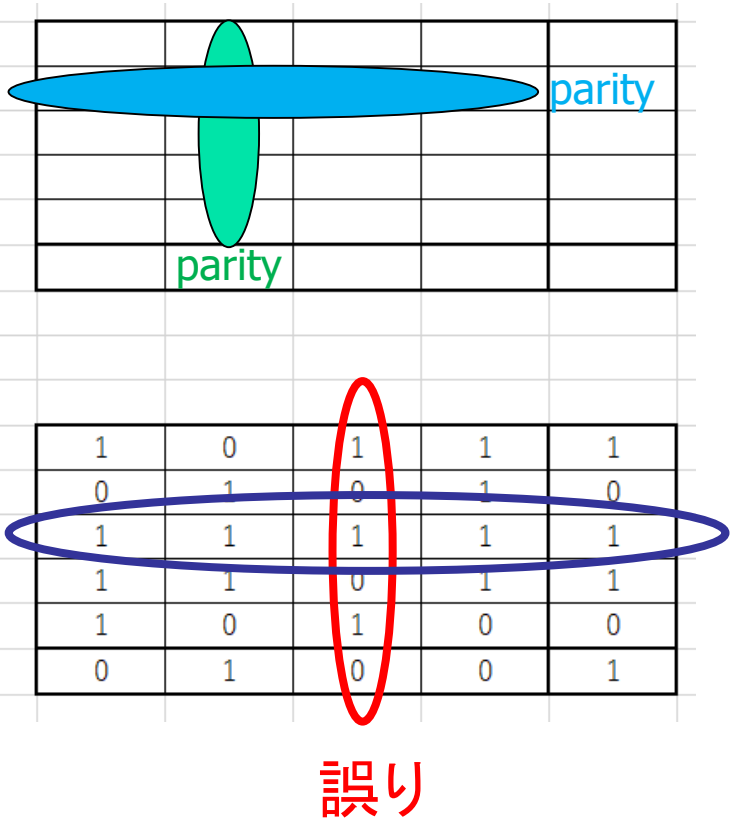
↓

宇宙航空用LSIはソフトエラー対策必須

誤り訂正符号 (Error Correction Code: ECC)

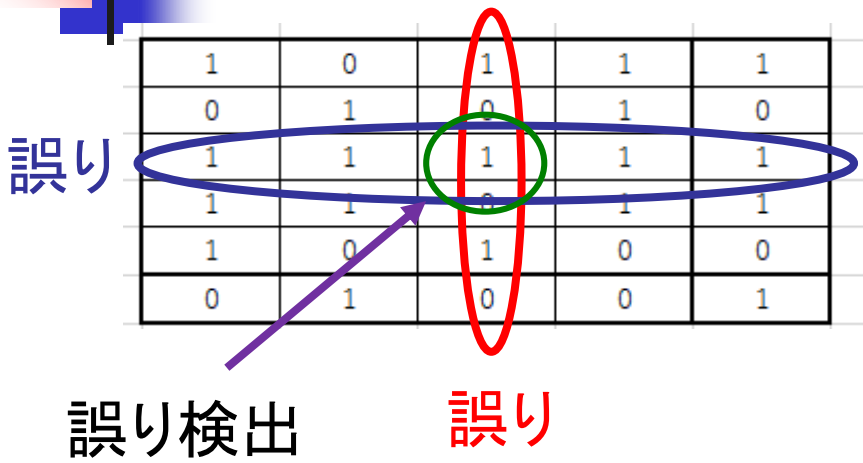
ECC付メモリ

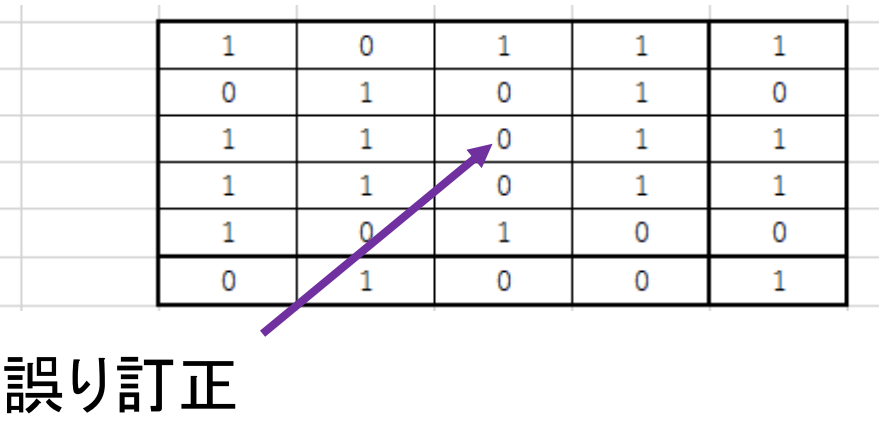




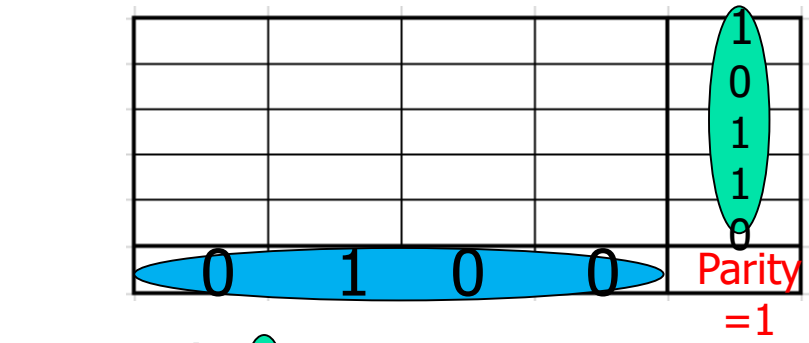
正しいデータの場合

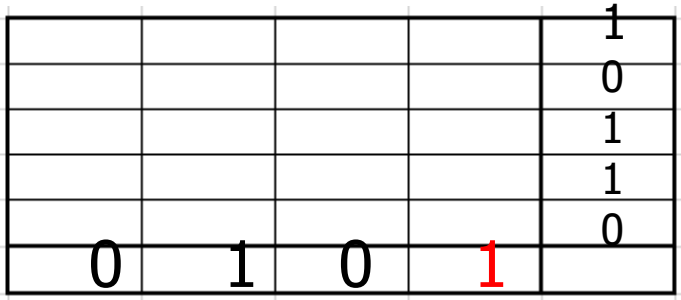
誤り訂正符号（続き）
(Error Correction Code: ECC)





Parity データは誤りがあることのみ検出可。訂正はできず。





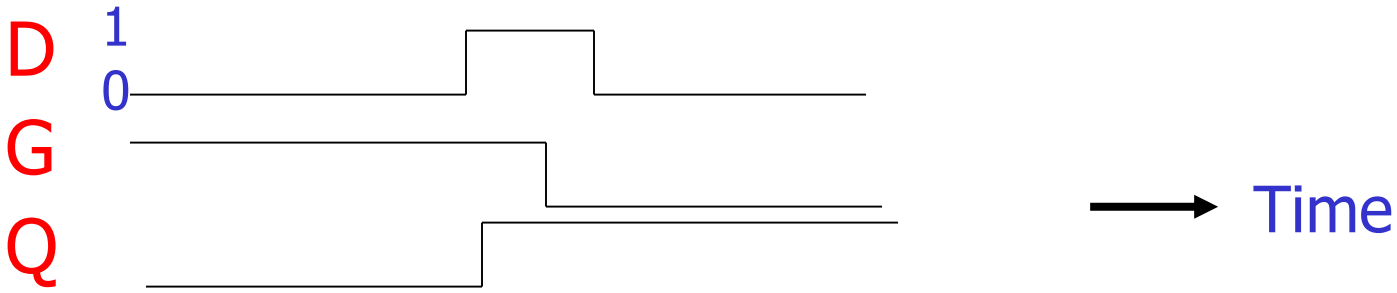
情報記憶素子(ラッチ)

論理変数　D, G, Q

　D, G:入力,　Q:出力

G=1 のとき　　Q=D

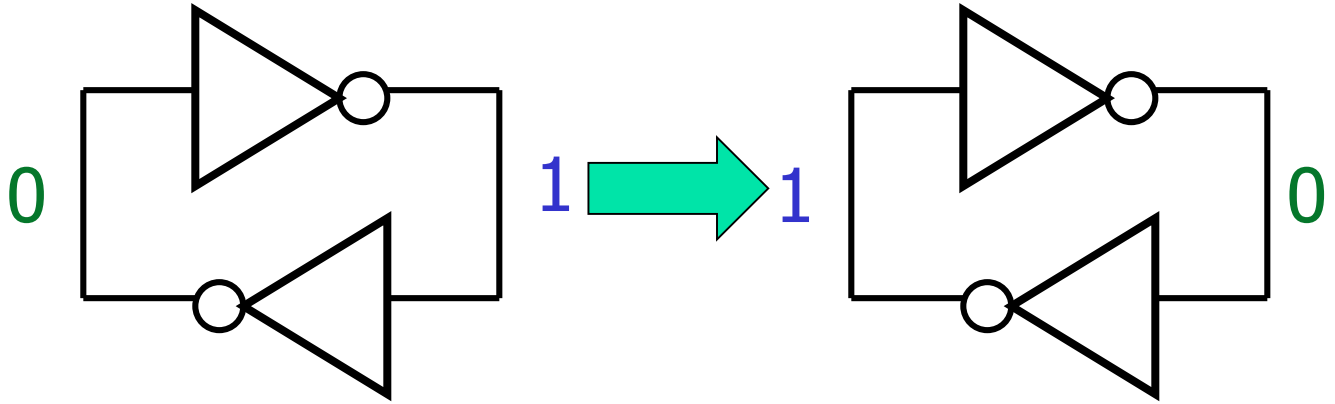
G=0 のとき　　Qは Gが1からOになる瞬間の
　　Dの値(1 or 0) を保持(記憶)している。



2つのインバータのリング接続 メモリ回路

2つの安定状態

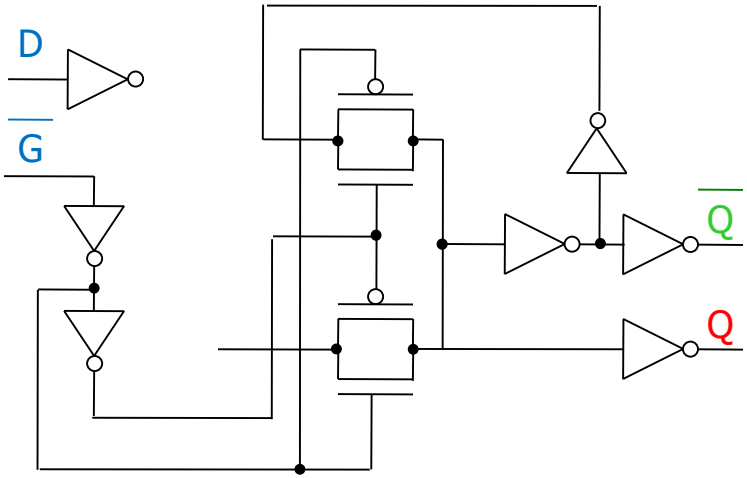
データ“1”を記憶　　データ“0”を記憶



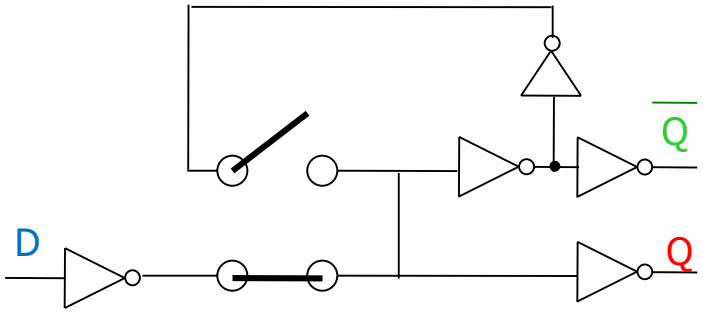
- SRAM (Static ランダム・アクセス・メモリ)
  Latch, Flip-Flop 等のメモリ素子は
  これを利用している。

CMOS ラッチ回路

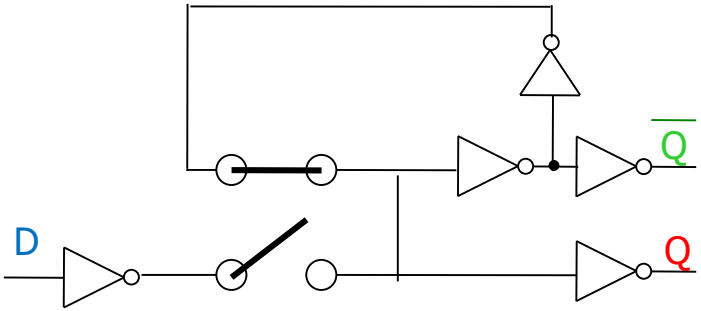
Latch回路
（メモリ素子）



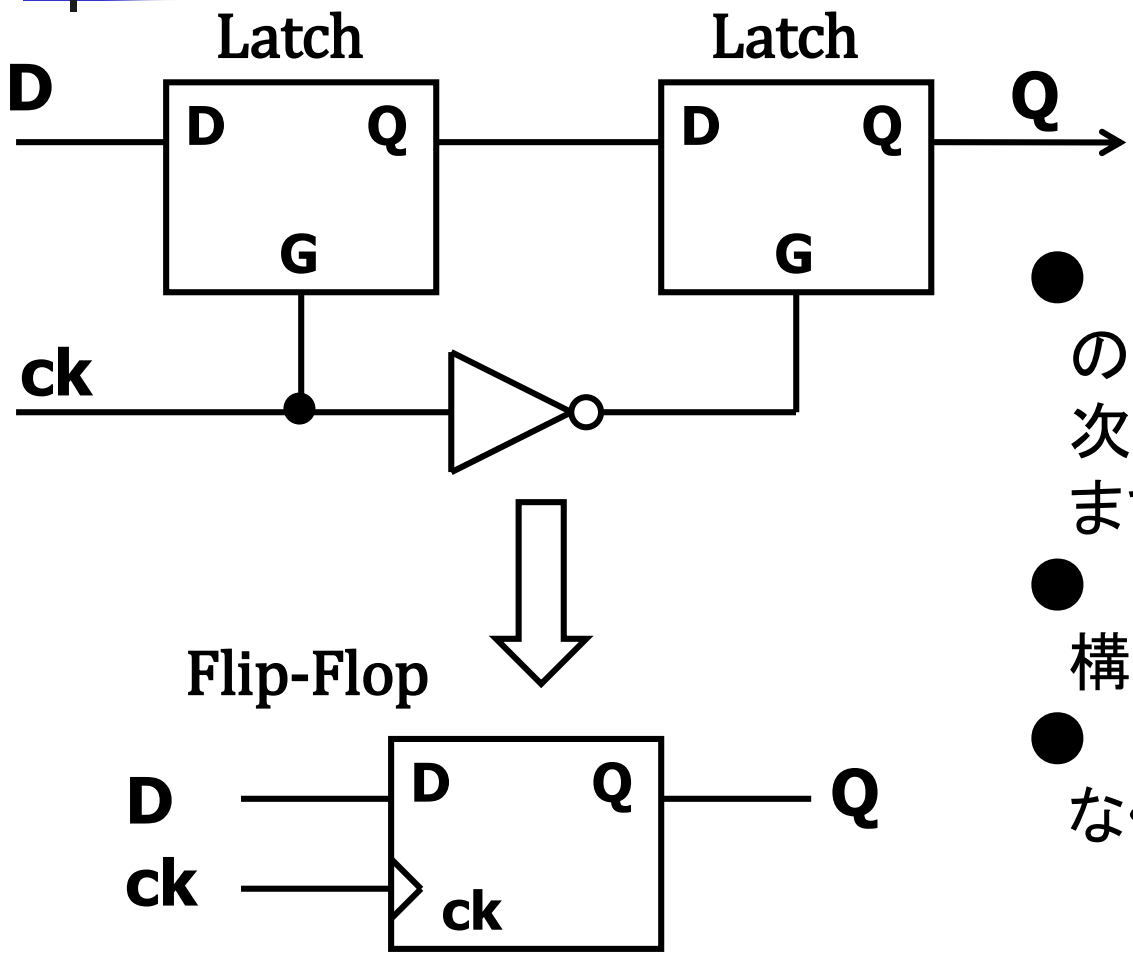
a) When $\overline{G}$ =0　トラック・モード



b)When $\overline{G}$ =1　ラッチ・モード

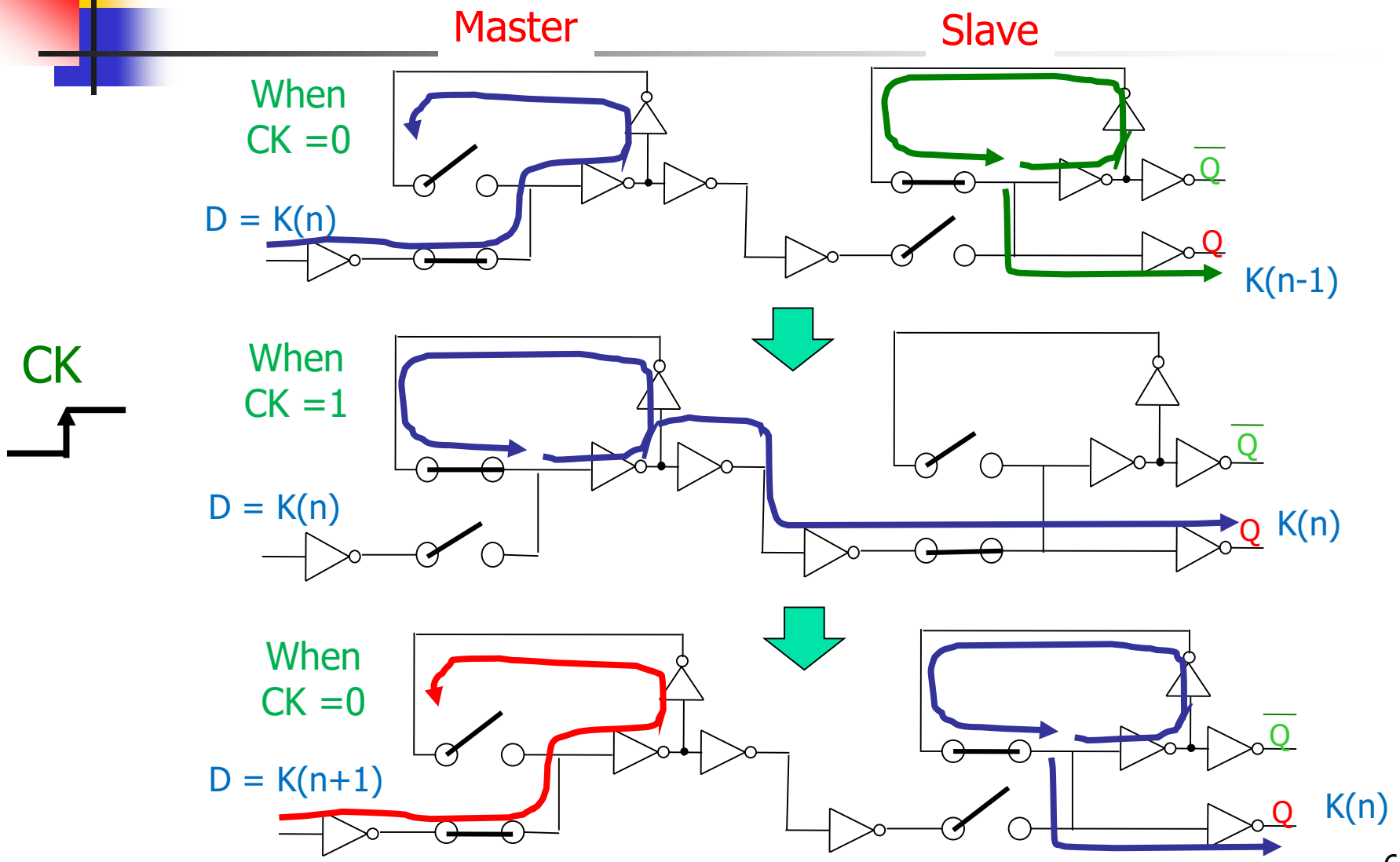


フリップ・フロップ回路

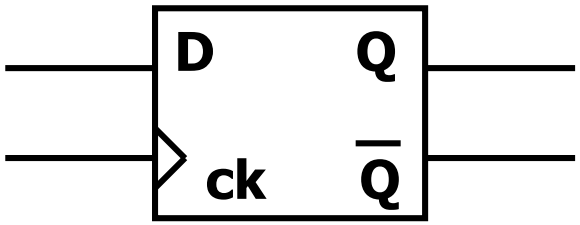


- クロックck の立ち上がりの瞬間のデータDを次のクロック立ち上がりまで（CK＝1，0でも）保持
- 2つのラッチ回路から構成
- 高速回路ではラッチではなくフリップフロップを使用

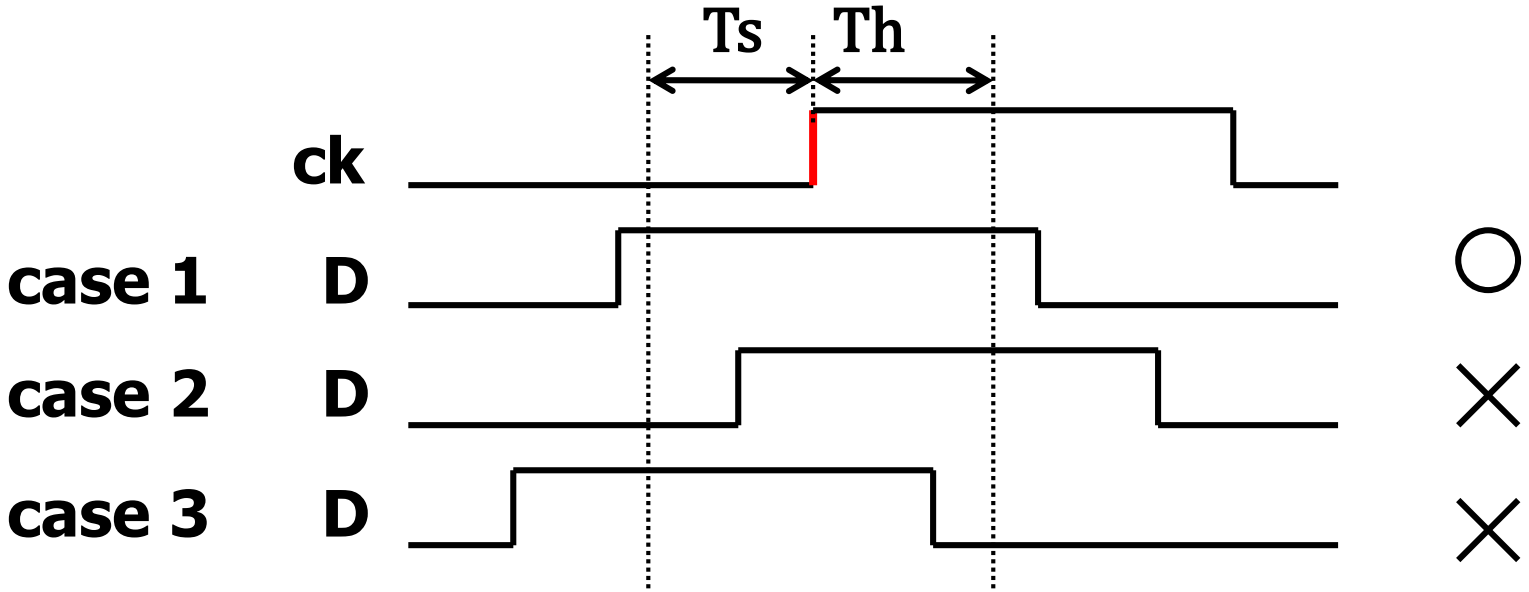
マスター・スレーブ・フリップフロップ回路



フリップ・フロップ回路と セットアップ時間、ホールド時間



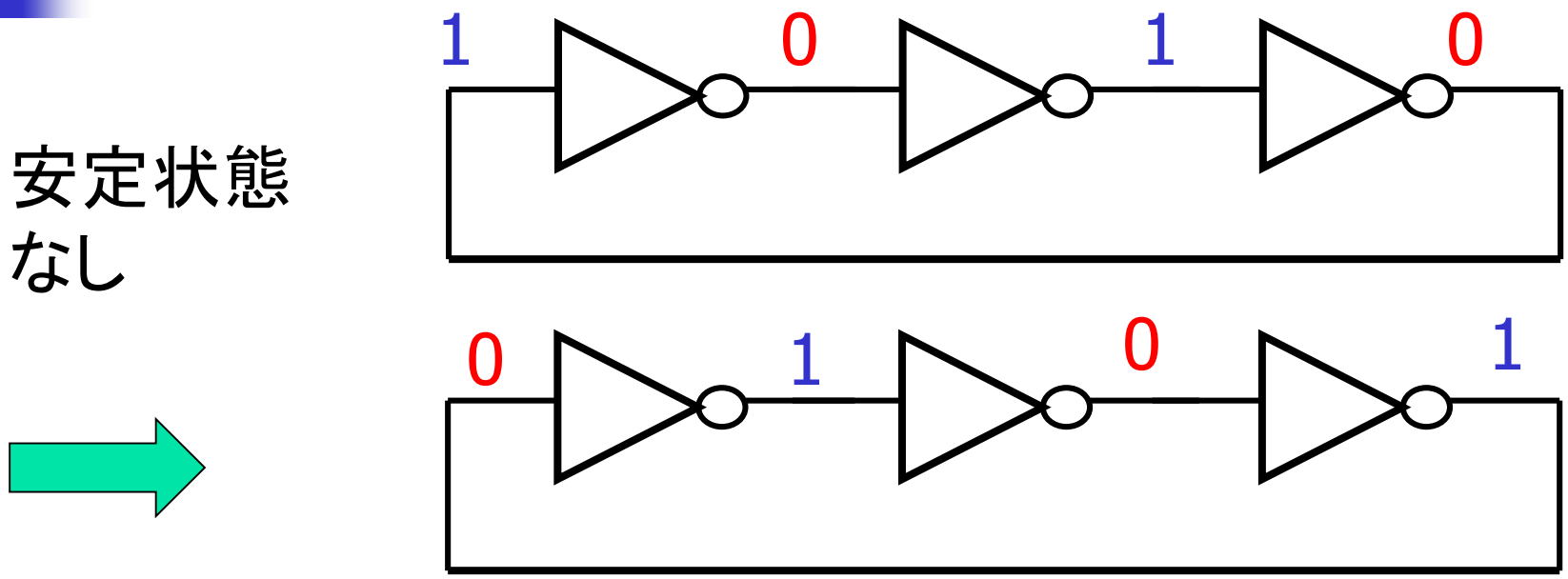
Ts : set-up time
Th : hold time



Set-up Time & Hold Time of Flip-Flop

奇数個インバータのリング接続
リング発振器

安定状態
なし



T: インバータ遅延、 2N+1 個のインバータリング接続

周波数 $f = \dfrac{1}{2\,(2N+1)\,T}$ で発振する。

内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- **デジタルCMOS回路の性能（消費電力、スピード）**
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

エネルギーとパワー

- エネルギー　[Joule]

  電力(パワー)　[Watt]

  Joule = Watt・s

  電力は単位時間当たりに消費されるエネルギー

  電力＝電圧･電流　P = V･I

- 電流:　単位時間当たりに流れる電荷量

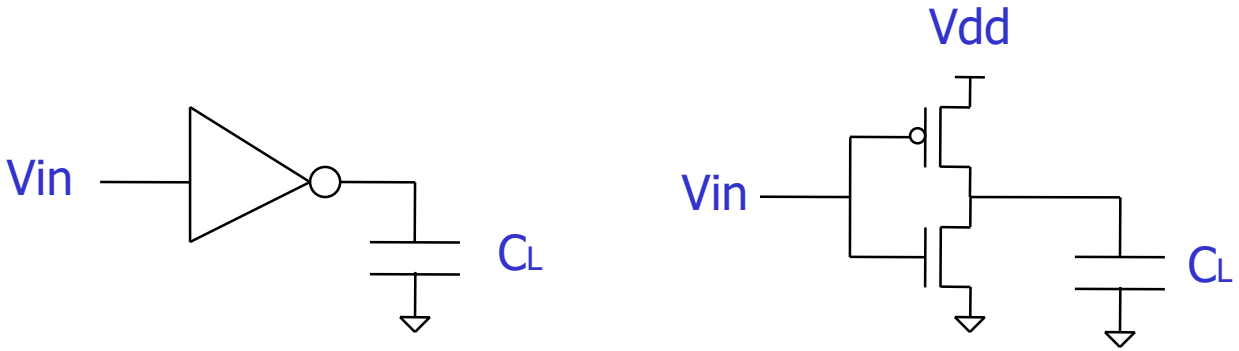
デジタルCMOS回路の電力消費

デジタルCMOS回路（インバータ）

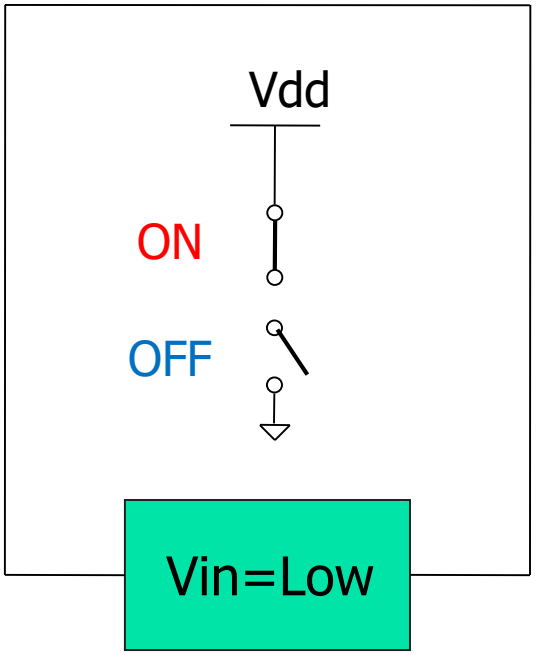
Vdd: 電源電圧

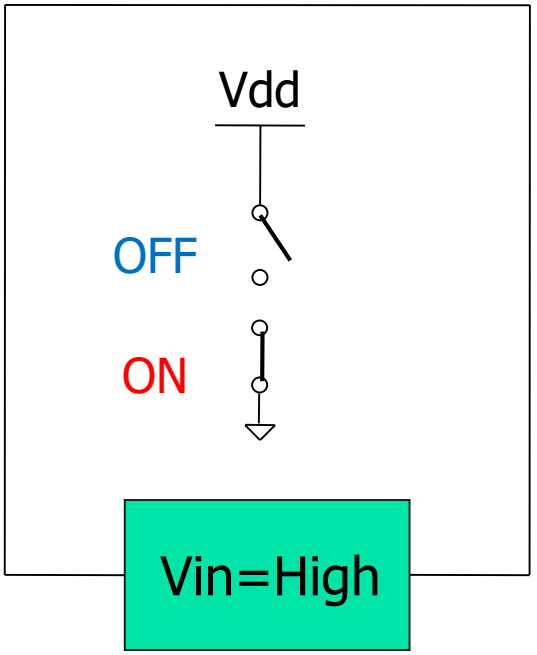
Vin: 入力、　**Vout**: 出力

C_L ： 負荷容量



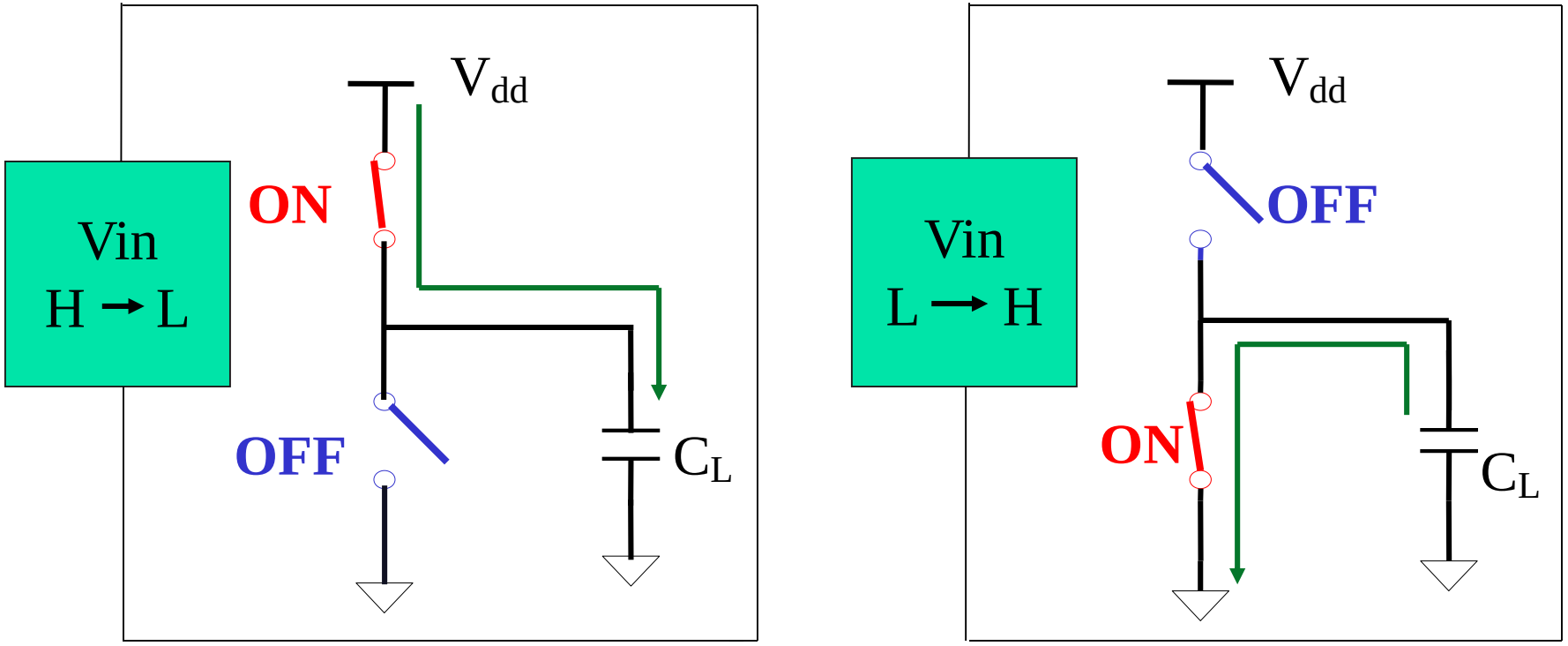
静的電力消費は小さい



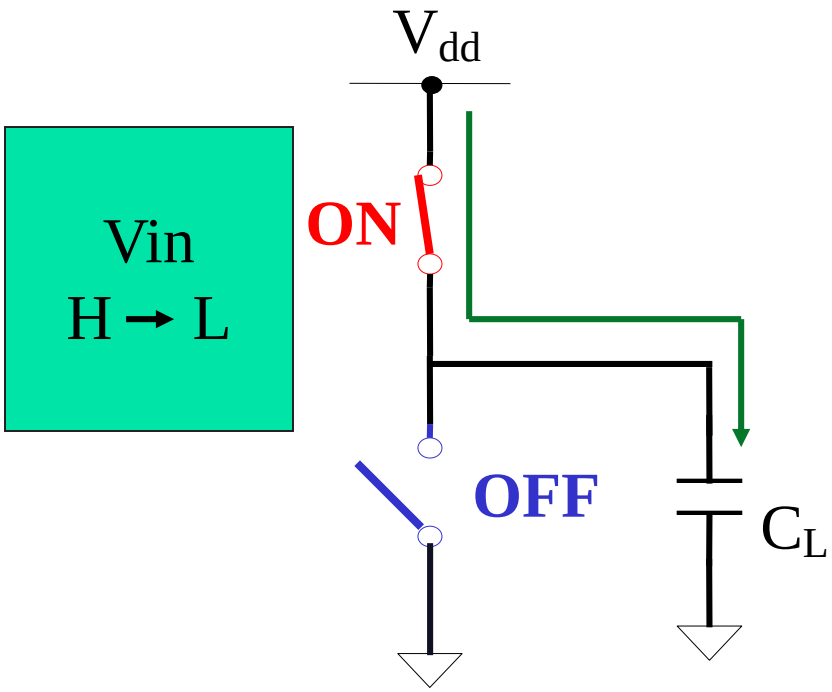


（注） 最近の微細CMOSデジタル回路では リーク電流が大きくなり、静的電力消費の占める割合が増えてきている。

動的消費電力 (1)



動的消費電力 (2)



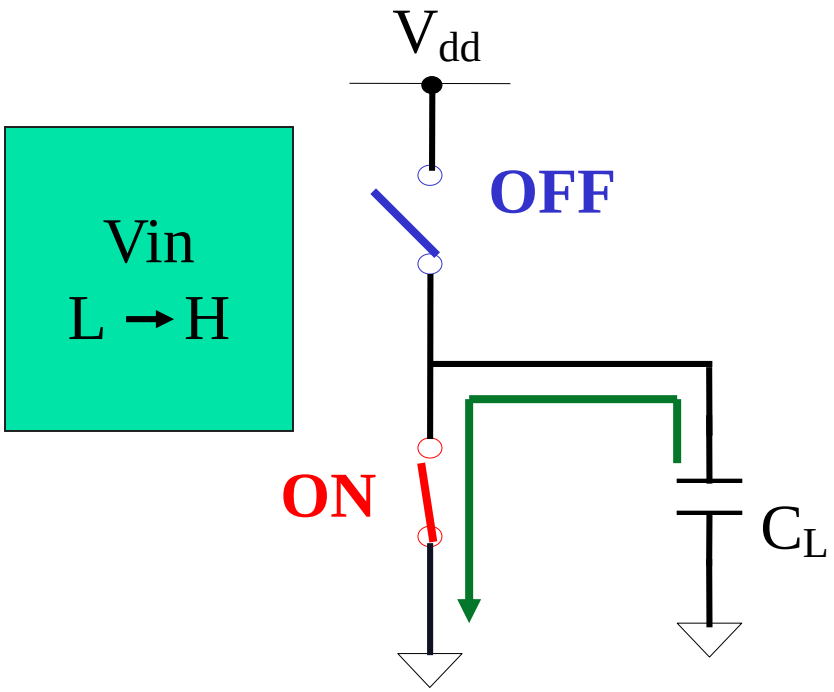
入力Vin

High → Low

蓄積電荷Q

0 → $C_L V_{dd}$

動的消費電力 (3)



入力Vin

Low ⇒ High

蓄積電荷Q

CL Vdd ⇒ 0

動的消費電力 (4)

V_{in} : H ⟶ L ⟶ H のとき

電荷 $Q = C_L V_{dd}$ が電源 V_{dd}から GND へ流れる。

一秒間に出力が f 回のトグルするとき

Vdd からGNDへ流れるトータルの電荷 $Q_{total} = f\, C_L\, V_{dd}$

∴ 消費電力

$$P = V_{dd} \cdot I$$
$$= V_{dd}(f \cdot C_L \cdot V_{dd})$$
$$= f \cdot C_L \cdot V_{dd}^{\,2}$$

f : 出力トグル周波数　　C_L : 負荷容量

V_{dd} : 電源電圧

デジタルCMOS VLSIの低消費電力化

低消費電力化は大きな技術的課題

例： 携帯電話 ⇒ バッテリーが長持ちさせる

低消費電力化技術 ⇒ f, CL, Vdd を小さくする。

技術のトレンド：

周波数 f ：マイクロプロセッサのクロック周波数はより高くなる。

X

寄生容量CL： 半導体の微細化により寄生容量は小さくなりつつある。 ○

電源電圧Vdd: より低くして用いる。

5V ⇒ 3.3V ⇒ 1.8V ⇒ 1V ○

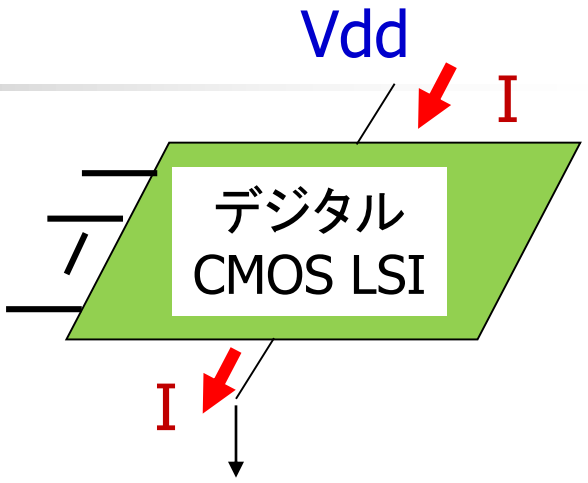
デジタルCMOS 回路の低消費電力化技術　例

低消費電力化技術 ⇒ f, C_L, Vdd を小さくする。

- 0, 1 のトグル回数の多いノード (f の高いノード)の負荷容量C_L を小さくする。
  - ⇒ そのノードの配線長を短くする (配線容量を小)
    - レイアウトが低消費電力化に貢献
  - Fan-out を小さくする
- ノードの0, 1 のトグル回数の少ないアルゴリズムを開発する
  - ⇒ 数学が低消費電力化に貢献

デジタルCMOS LSI に対する
IDDQ テスト

各入力ピンを
Vdd または
GNDに固定



デジタルCMOS LSI

テスト時に入力を固定（トグルしない）

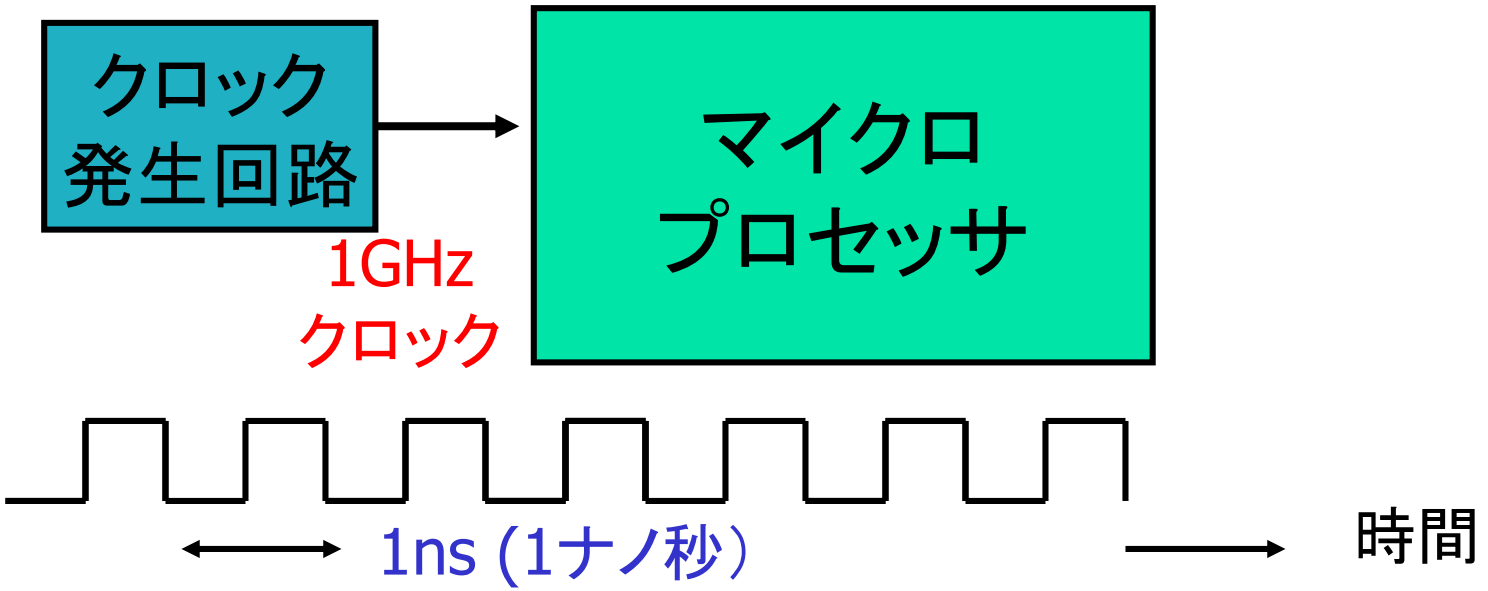
正常チップ：電源VddからGNDへの電流 I は微小

電流Iが大きければ「どこかに故障あり」と判定

http://mix.kumikomi.net/index.php/IDDQテスト

マイクロプロセッサのクロック

- クロックに同期して動作（同期回路）
  クロックの立ち上がりで論理回路はトグル。
- より高い周波数になってきている。



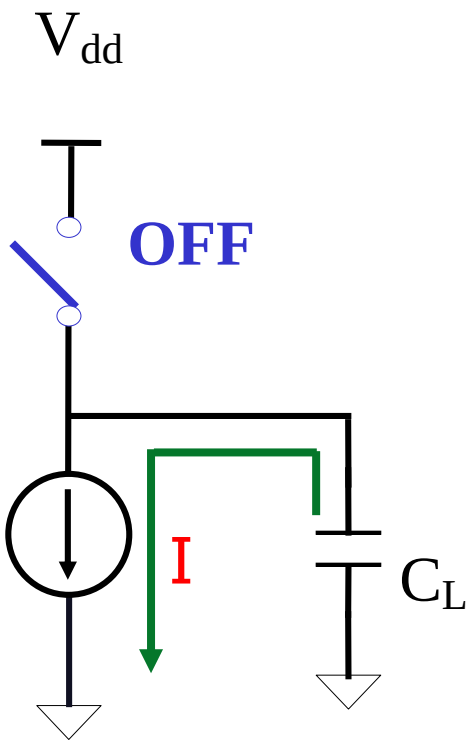
デジタルCMOS 回路のスピード

電源電圧 Vdd:

- 低消費電力化のため電源電圧を下げると
  スピードは遅くなる。
- スピードは電源電圧に比例
- 消費電力は電源電圧の２乗に比例

温度: スピードは温度にほぼ反比例。
電子、正孔の移動度が温度上昇とともに減少のため

なぜ電源電圧を上げるとデジタルCMOS回路は高速化するのか？



引き抜く電荷
Q=C Vdd

MOSの２乗則

$I = K (Vdd-Vth)^2$

$\fallingdotseq K\ Vdd^2$

ゲート遅延

$T = Q / I$

$= C / (K\ Vdd)$

デジタル回路の Figure of Merit (FOM)

FOM = スピード/消費エネルギー

「A」のエネルギーを消費し「B」のスピードの回路と、
「2A」のエネルギーを消費し「2B」のスピードの回路の
FOM は同じ。

工学設計：トレードオフ（Trade-off, 妥協）
の考え方が重要

デジタルCMOS回路：
電源電圧を小さくして使用するとFOMが良。

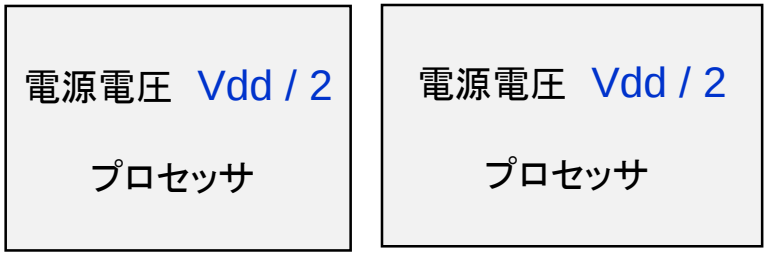
並列処理による低消費電力化

ケース1： 電源電圧 Vdd, 1つのプロセッサ

消費電力 $= K (Vdd)^2$

処理スピード $= L\ Vdd$

ケース2： 電源電圧 Vdd/2, 2つのプロセッサ



消費電力 $= K (Vdd/2)^2 + K (Vdd/2)^2$

$= (1/2)\ K (Vdd)^2$

処理スピード

$= (1/2)\ L\ Vdd + (1/2)\ L\ Vdd$

$= L\ Vdd$

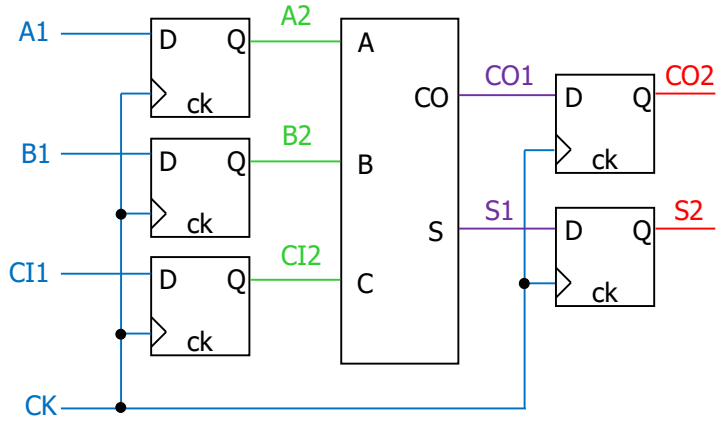
ケース2はケース1と処理スピードは同じであるが、消費電力は1/2 になる

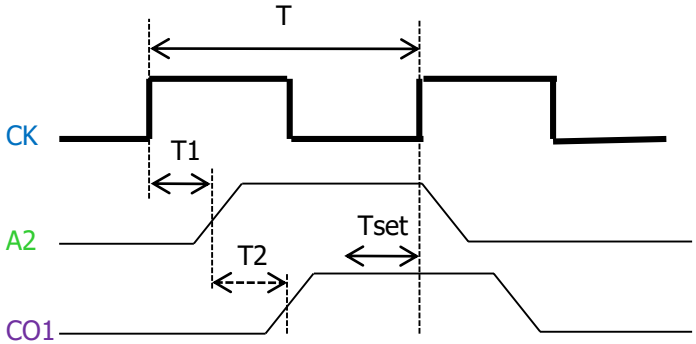
内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- **同期回路設計とカウンタ回路**
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

同期回路とタイミング設計

Synchronous Circuit Design



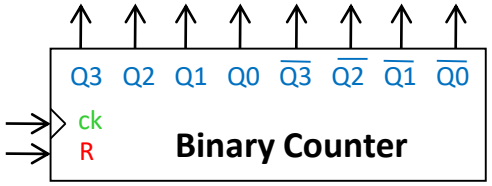


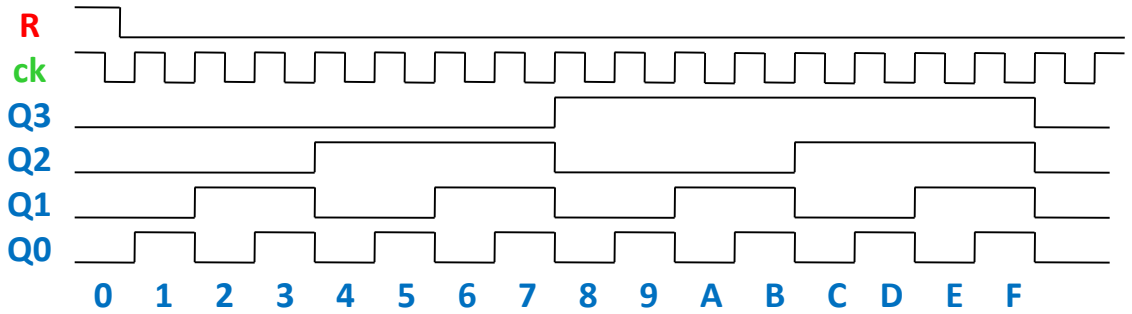
- Timing Requirement

T>T1+T2+Tset
T ： clock period
T1 ： Flip-Flop Delay
T2 ： Full Adder Delay
Tset ： Flip-Flop Setup Time

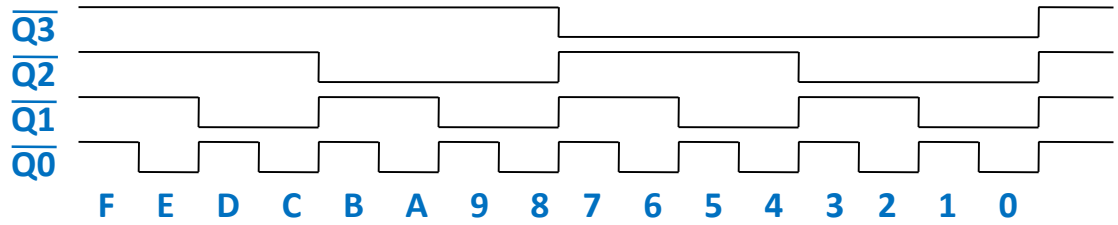
同期設計は
タイミング設計が
比較的容易

2進カウンタ (Binary Counter)





Up Counter

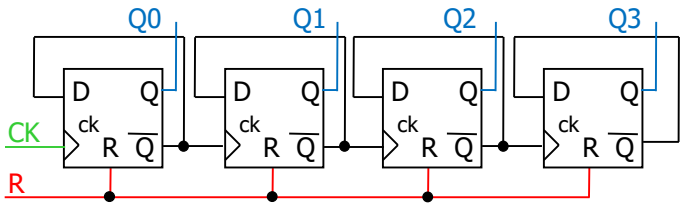


Down Counter

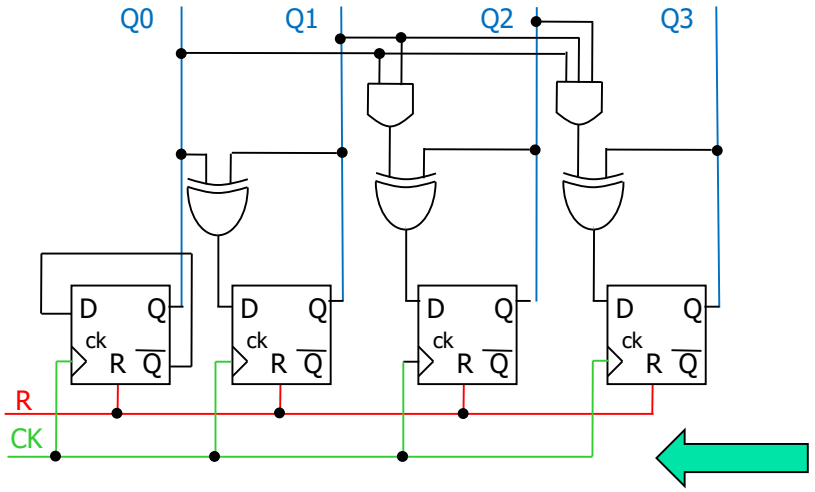
- クロック数を数える
- タイミング発生回路に使用される

非同期、同期カウンタ

非同期カウンタ



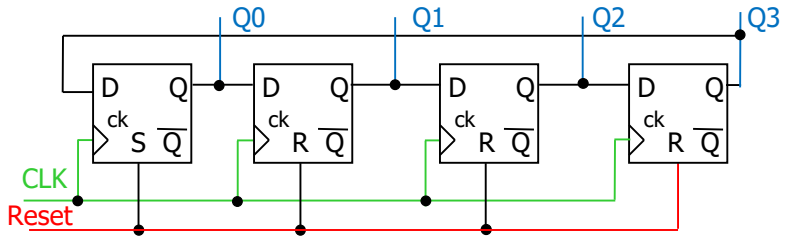
同期カウンタ

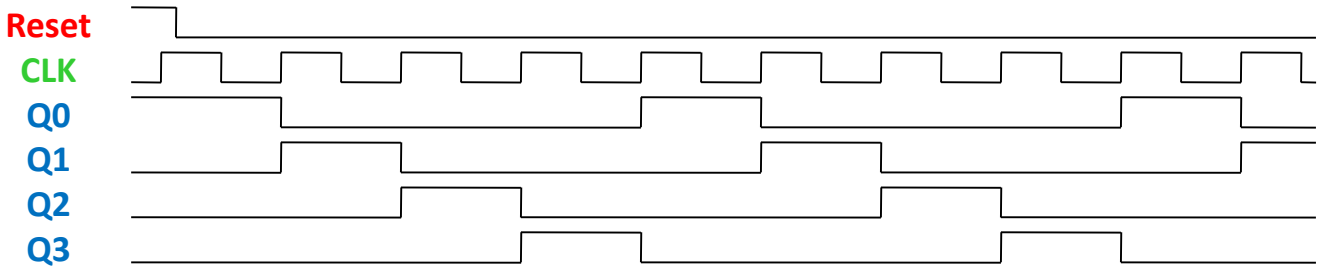


- 非同期回路は小規模になりえるが回路設計・変更・検証が難しい。
- デジタル回路は同期設計が基本

同期回路:
全てのフリップフロップのクロックが同一

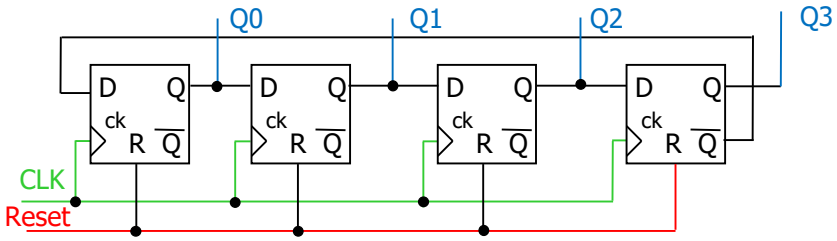
リング・カウンタ （Ring Counter)

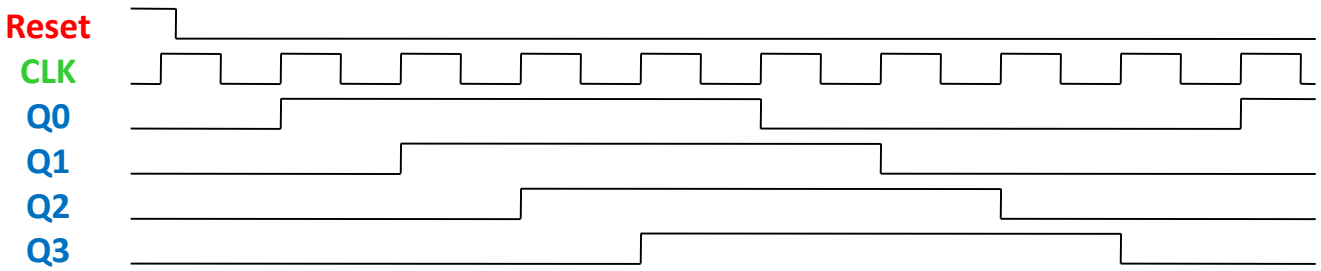




逐次比較近似ADCのタイミング発生回路等に使用される。

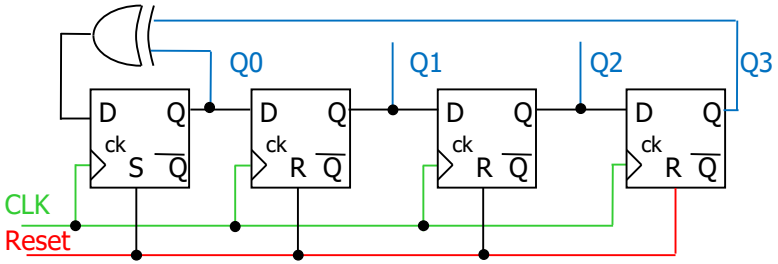
ジョンソン・カウンタ（Johnson Counter）

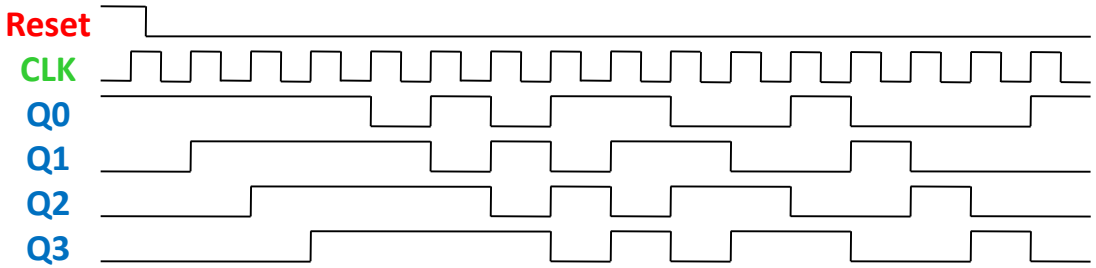




リングカウンタ回路と似ているが
出力信号は大きく異なる。
デジタル計算機のシーケンサ回路として使用された。

疑似ランダム信号発生回路 (Linear Feedback Shift Register)





- Q0=Q1=Q2=Q3=0 以外の15通りの信号を発生
- (再現性のある)疑似ランダム信号を発生 ⇨ M系列信号
- フィードバックの取り方には決まりあり

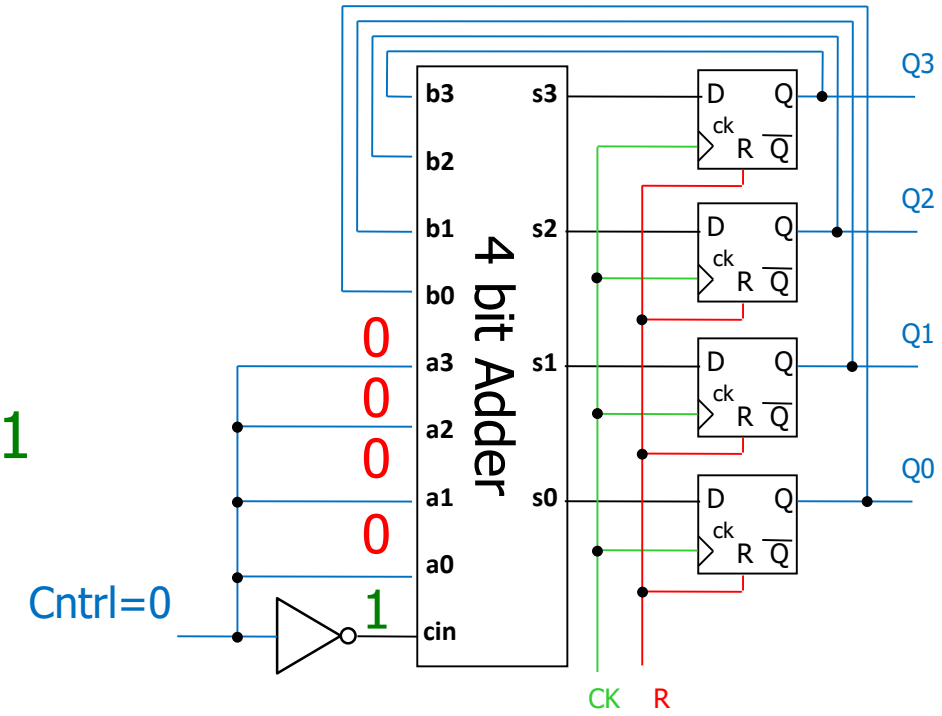
https://ja.wikipedia.org/wiki/線形帰還シフトレジスタ

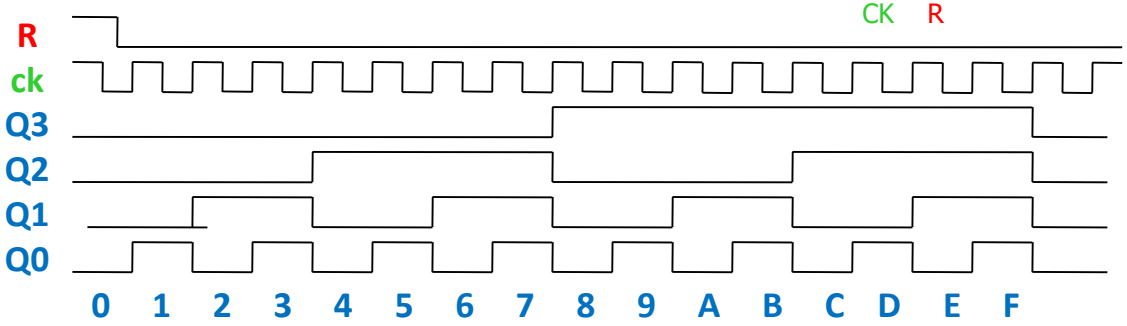
http://zakii.la.coocan.jp/signal/41_lfsr.htm

アップ・ダウン　2進カウンタ

制御信号
Cntrl=0 のとき
アップカウンタ

$$\overrightarrow{Q(n+1)} = \overrightarrow{Q(n)} + 1$$

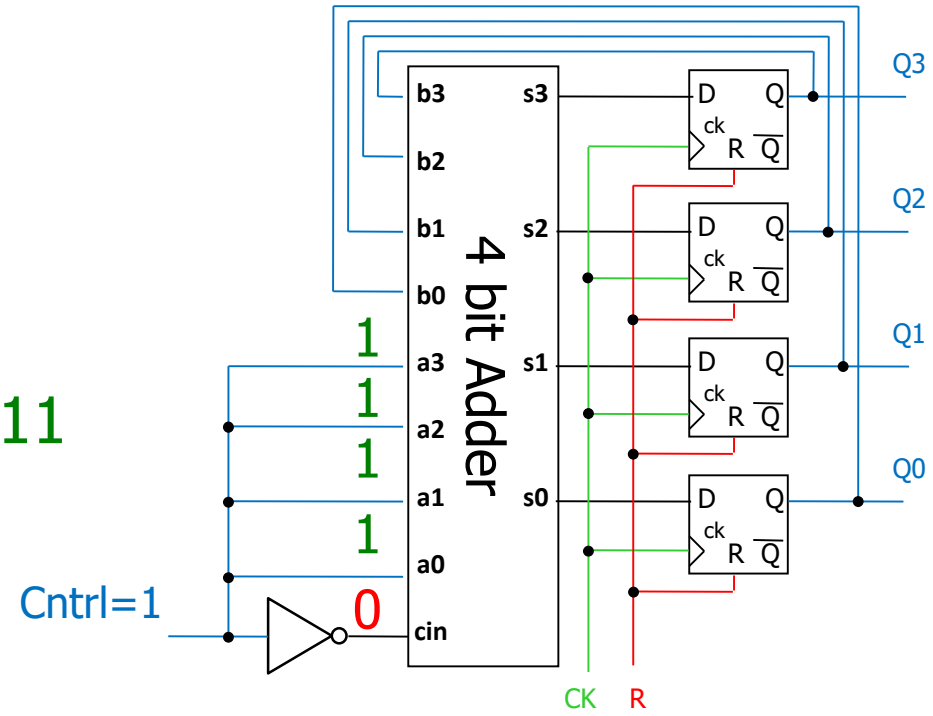


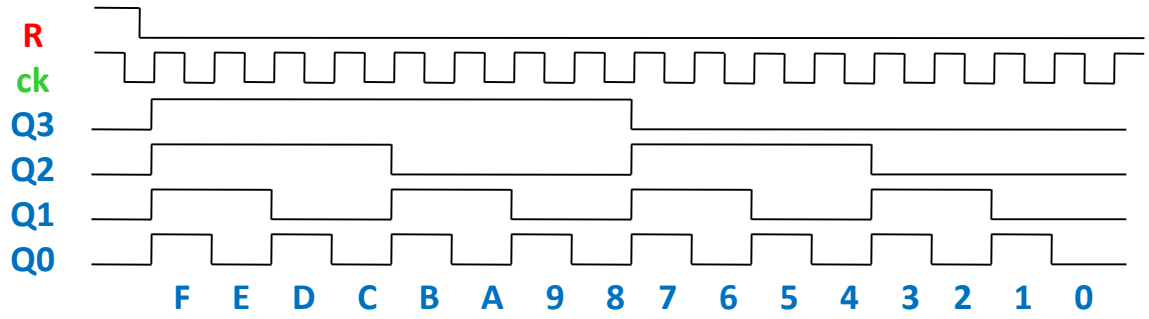


アップ・ダウン　2進カウンタ

制御信号
Cntrl=1 のとき
ダウンカウンタ

$$\overrightarrow{Q(n+1)} = \overrightarrow{Q(n)} + 1111$$
$$= \overrightarrow{Q(n)} - 1$$





内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- **加算器、ビットシフト、乗算器**
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

デジタル加算

2進数の加算

```
  0011   (3)
+) 1011  (11)
------------
  1110   (14)
```

10進数の加算

```
  437
+)258
------
  695
```

2入力2進加算

0+0=00

0+1=01

1+0=01

1+1=10

3入力2進加算

0+0+0=00

0+0+1=01

0+1+1=10

1+1+1=11

デジタル加算器の実現(2)
（全加算器； Full Adder）

3入力2進加算

A　入力1

B　入力2

+) Cin　下からの繰り上がり

Co S

$S = A \oplus B \oplus Cin$

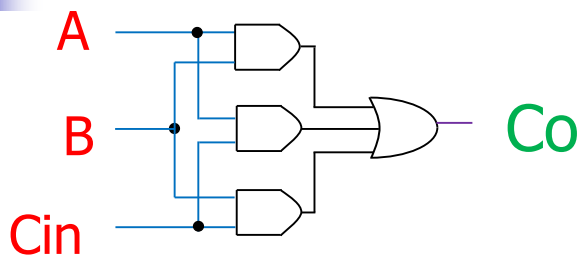
$Co = B \cdot Cin + A \cdot Cin + A \cdot B$

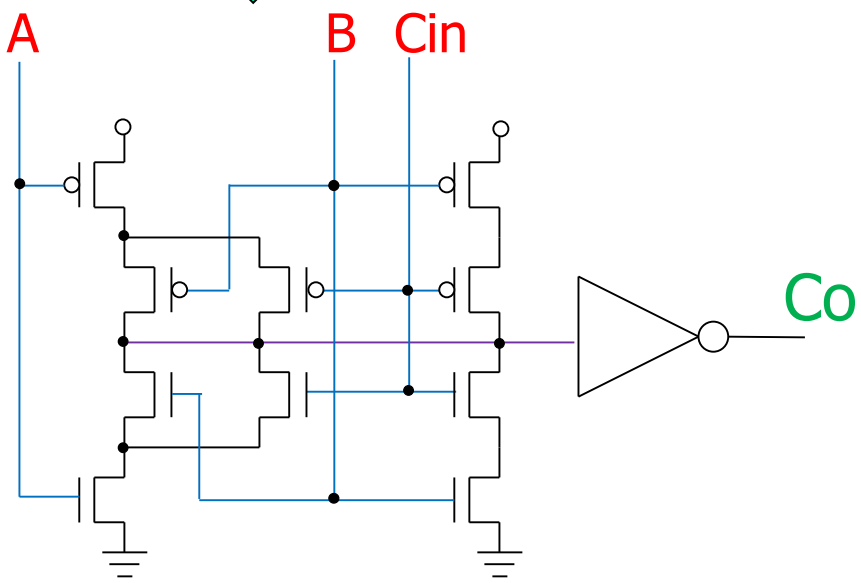
(Co は A, B, Cin の多数決）

真理値表

A	B	Cin	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

多数決回路 (Majority Circuit)





真理値表

A	B	Cin	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

全加算器 (Full Adder)の補足説明

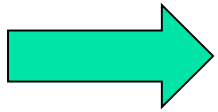
Cin: Carry in (下位の桁からの繰り上げ)

S: Sum（加算結果）

Cout: Carry out （上位の桁への繰り上げ）

```
   0   1
+) 1   1
-------
 1 0   0
```

を考える。



全加算器の演算

```
   1
   0   1
+) 1   1
-------
 1 0   0
```

全加算器の補足説明(続き)

```
    0  1  0  1  (5)
+)  0  1  1  1  (7)
-------------------
 0  1  1  0  0  (12)
```

を考える。

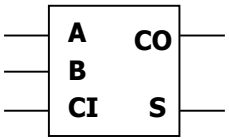
```
    1  1  1
    0  1  0  1
+)  0  1  1  1
-------------------
 0  1  1  0  0
```

Carry（桁上げ）

Sum（加算結果）

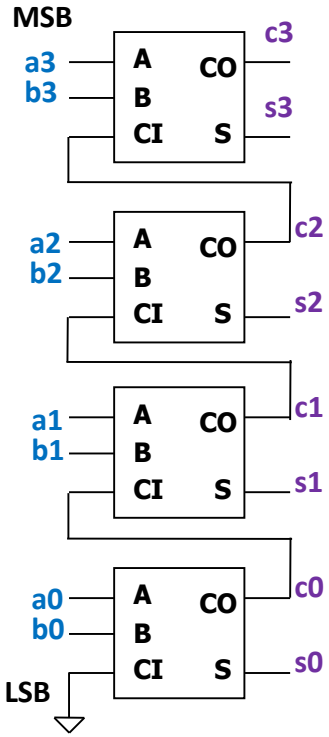
全加算器と桁上げ伝播

Full Adder



A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
0	1	1	0	1
1	1	0	0	1
1	0	1	0	1
1	1	1	1	1

Adder & Carry Propagation



Example :

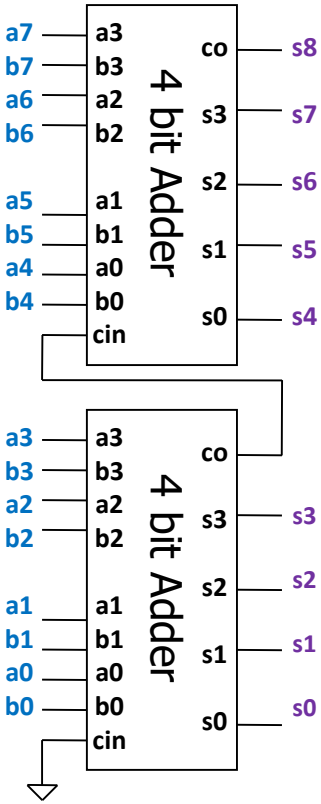
$$\begin{array}{r} 0\,1\,1\,1 \\ +\ 0\,1\,0\,1 \\ \hline 1\,1\,0\,0 \end{array}$$

(a3 a2 a1 a0) =(0 1 1 1)
(b3 b2 b1 b0)=(0 1 0 1)

(s3 s2 s1 s0)=(1 1 0 0)
(c3 c2 c1 c0)=(0 1 1 1)

桁上げ選択加算器 (Carry Select Adder) による高速化

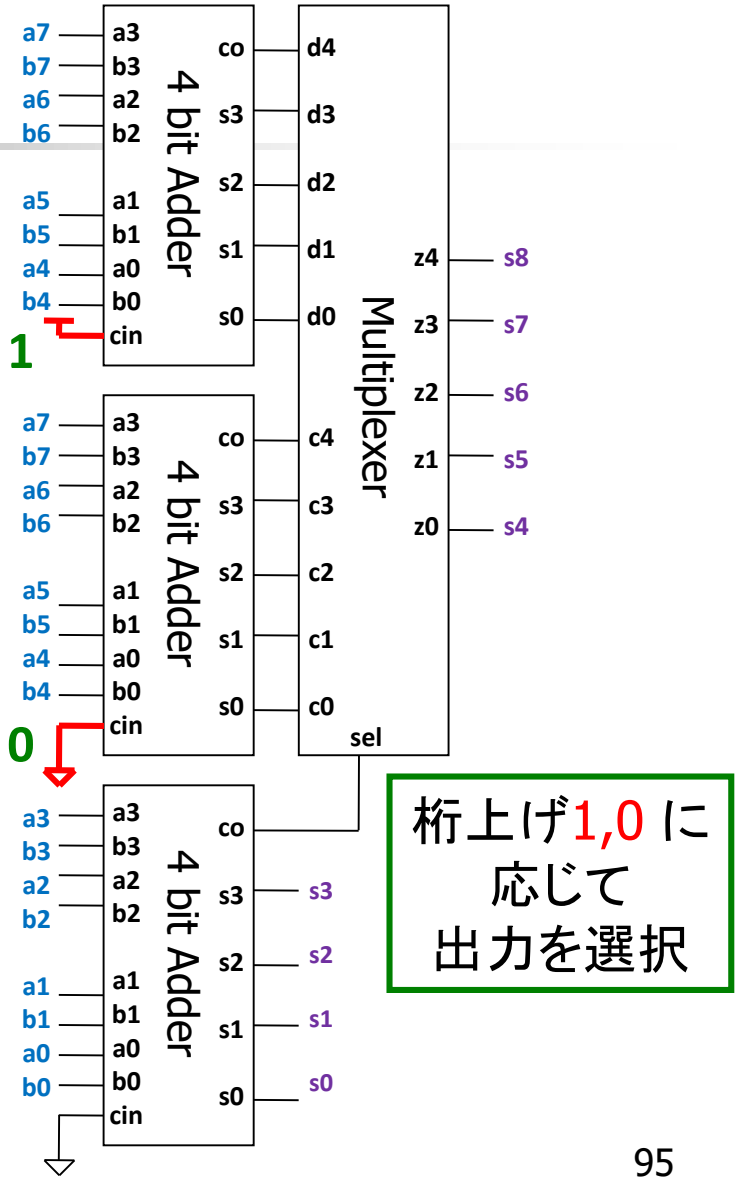
通常の
8bit 加算器



桁上げ1の場合を計算

桁上げ0の場合を計算

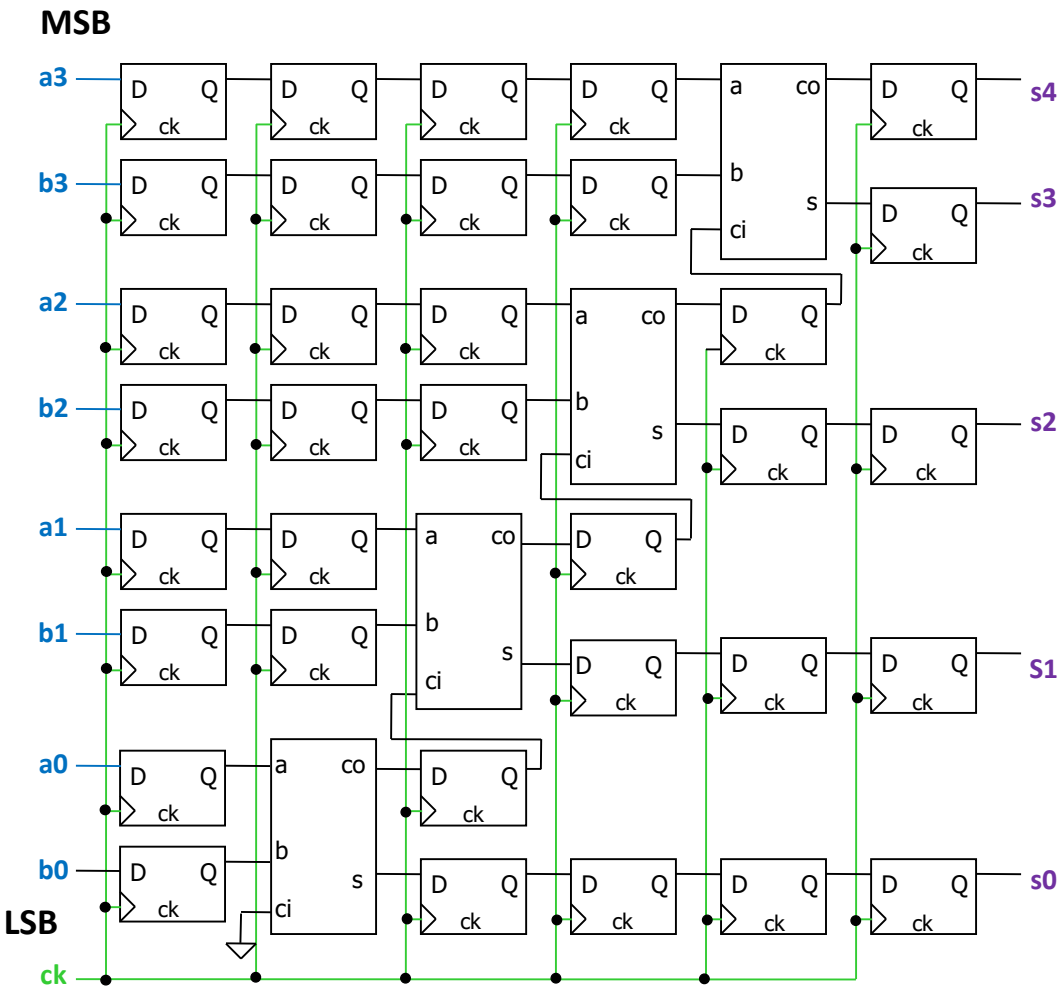
約1．5倍の回路規模で約2倍のスピード



桁上げ1,0 に応じて出力を選択

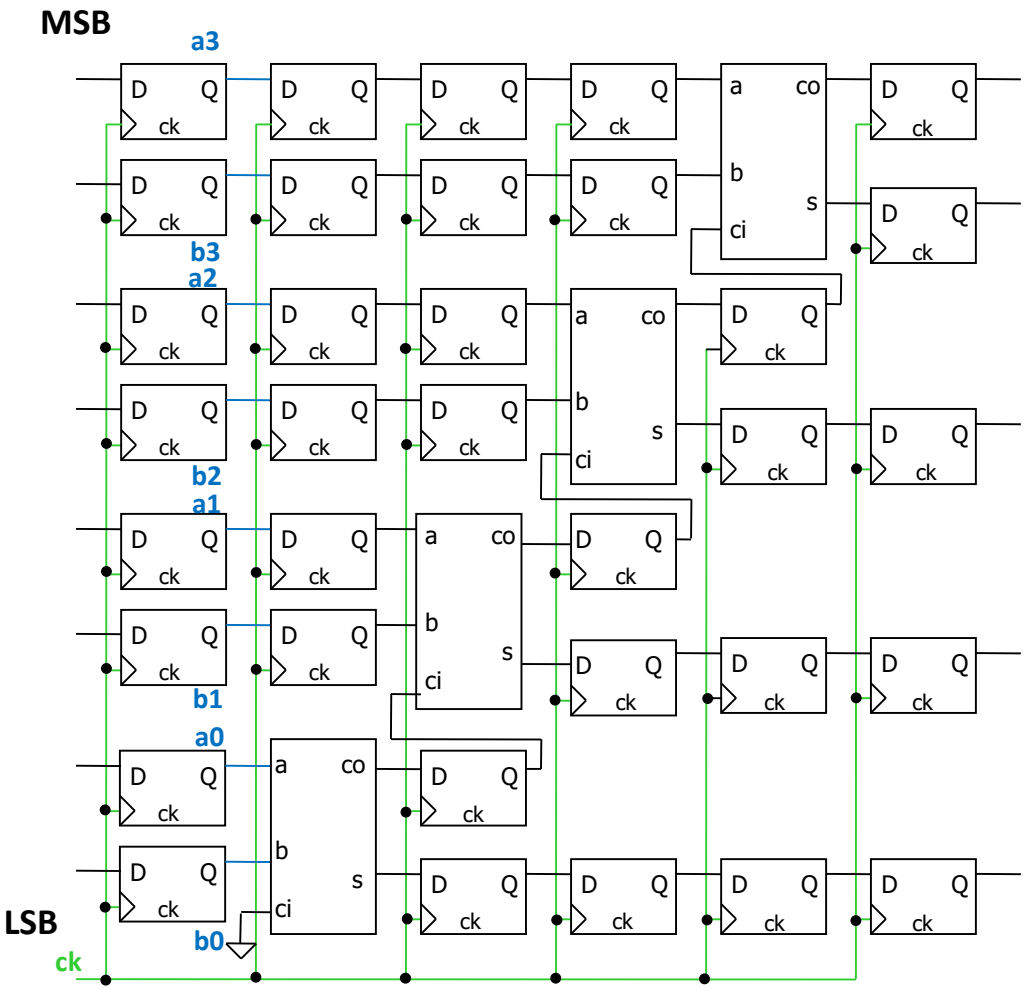
パイプライン加算器の構成

初期状態



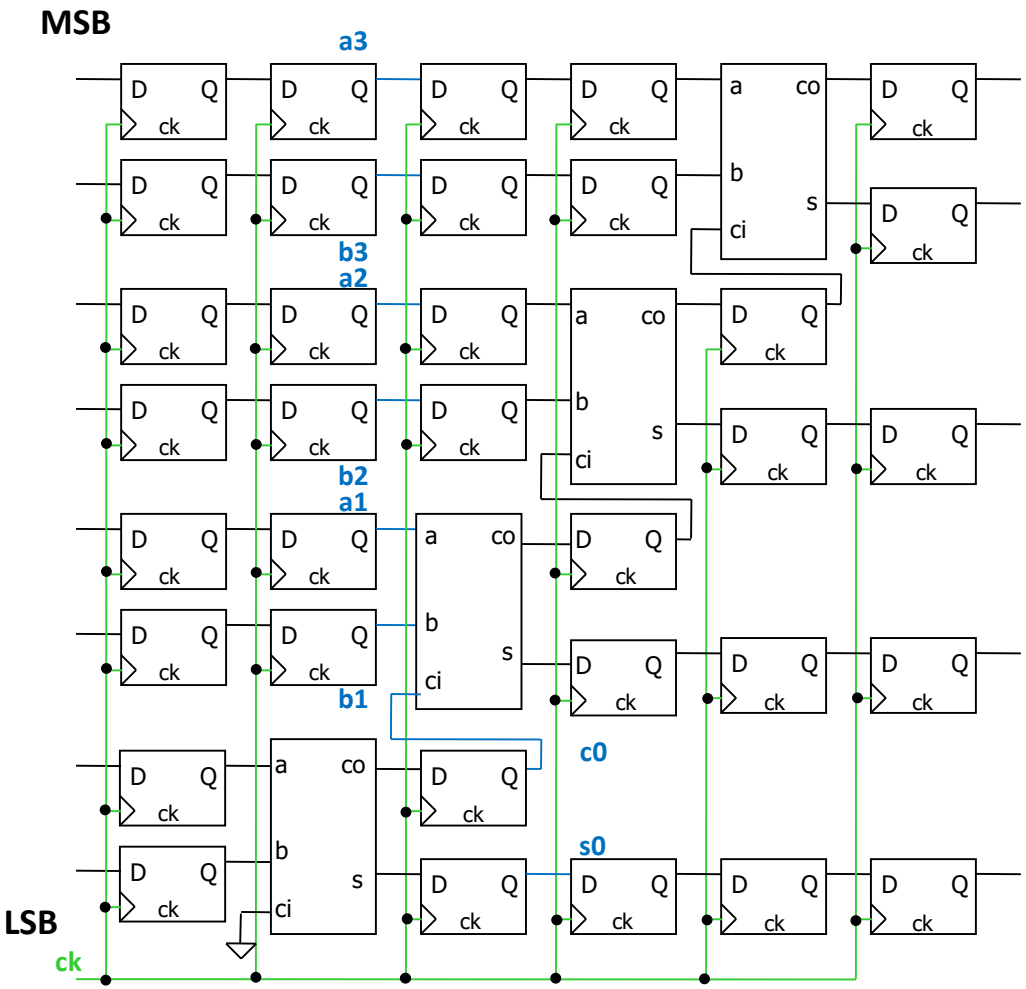
パイプライン加算器の動作

1クロック後



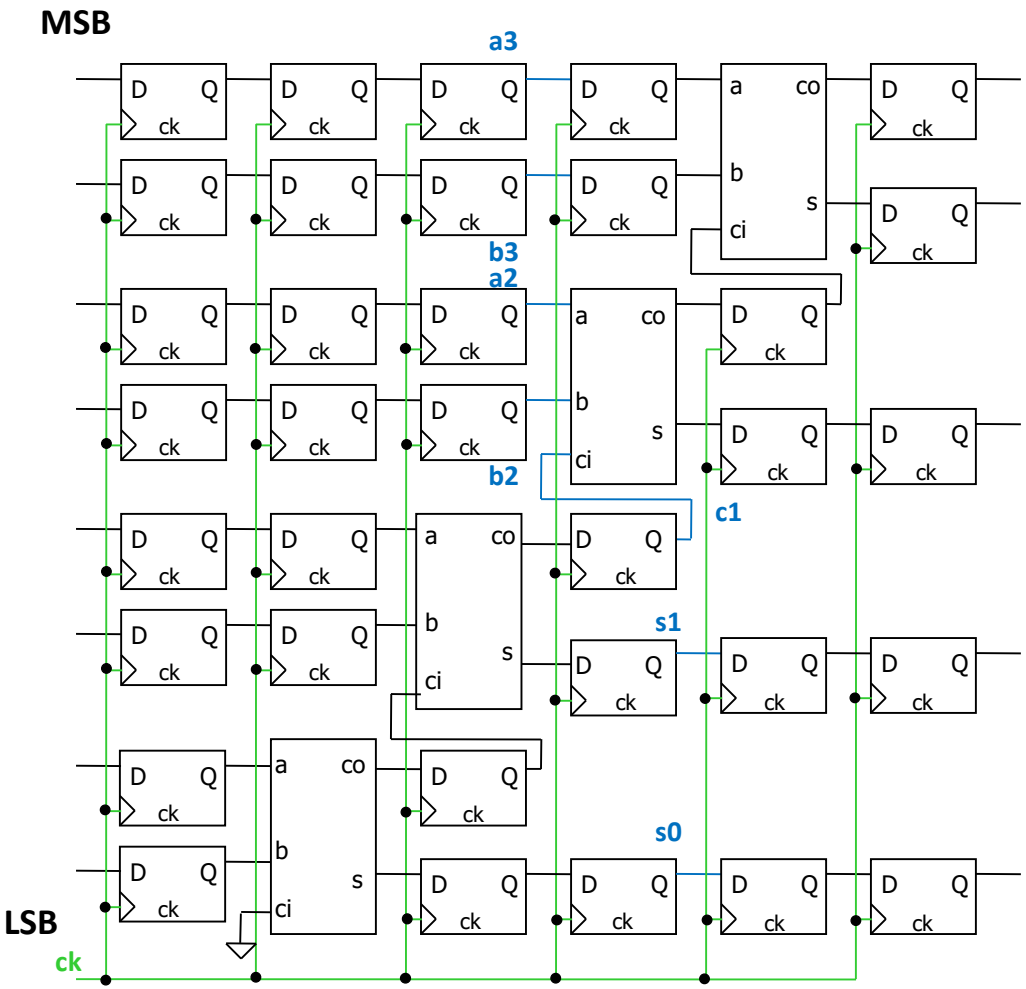
パイプライン加算器の動作

2クロック後



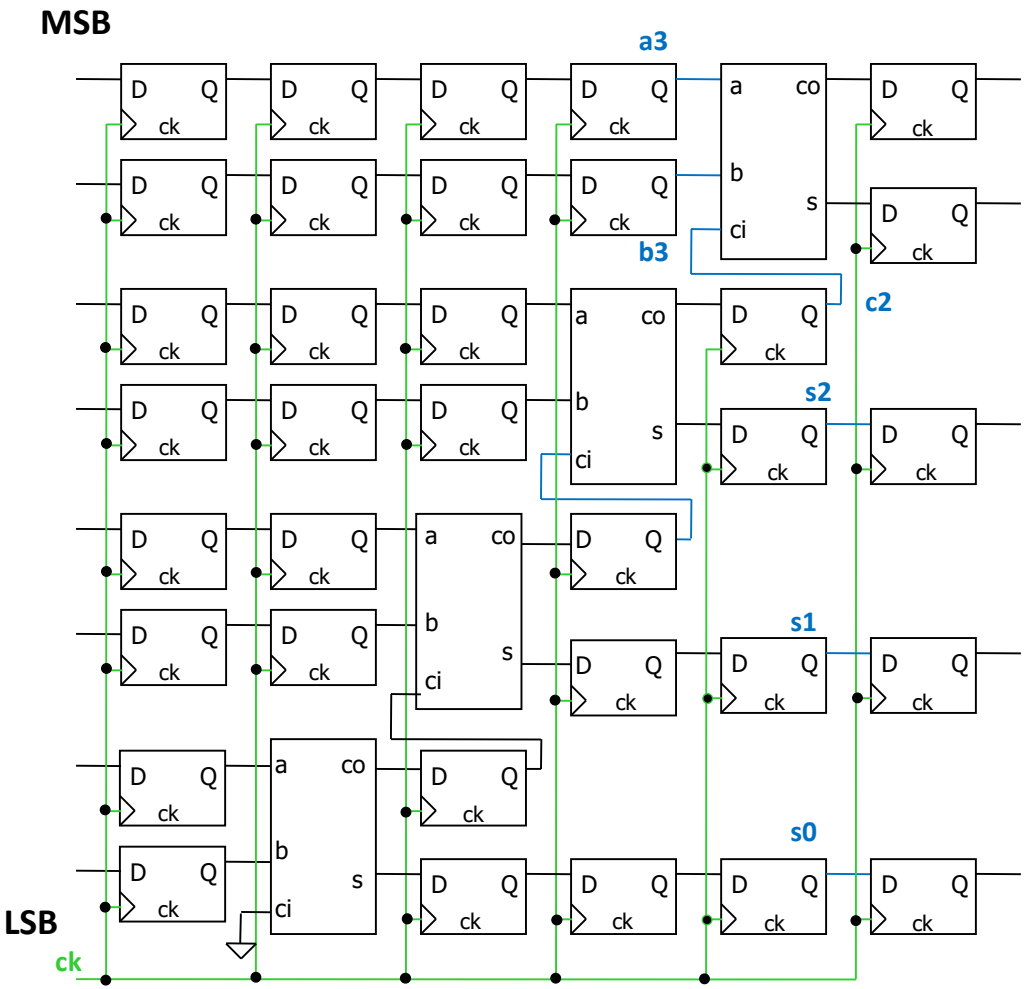
パイプライン加算器の動作

3クロック後



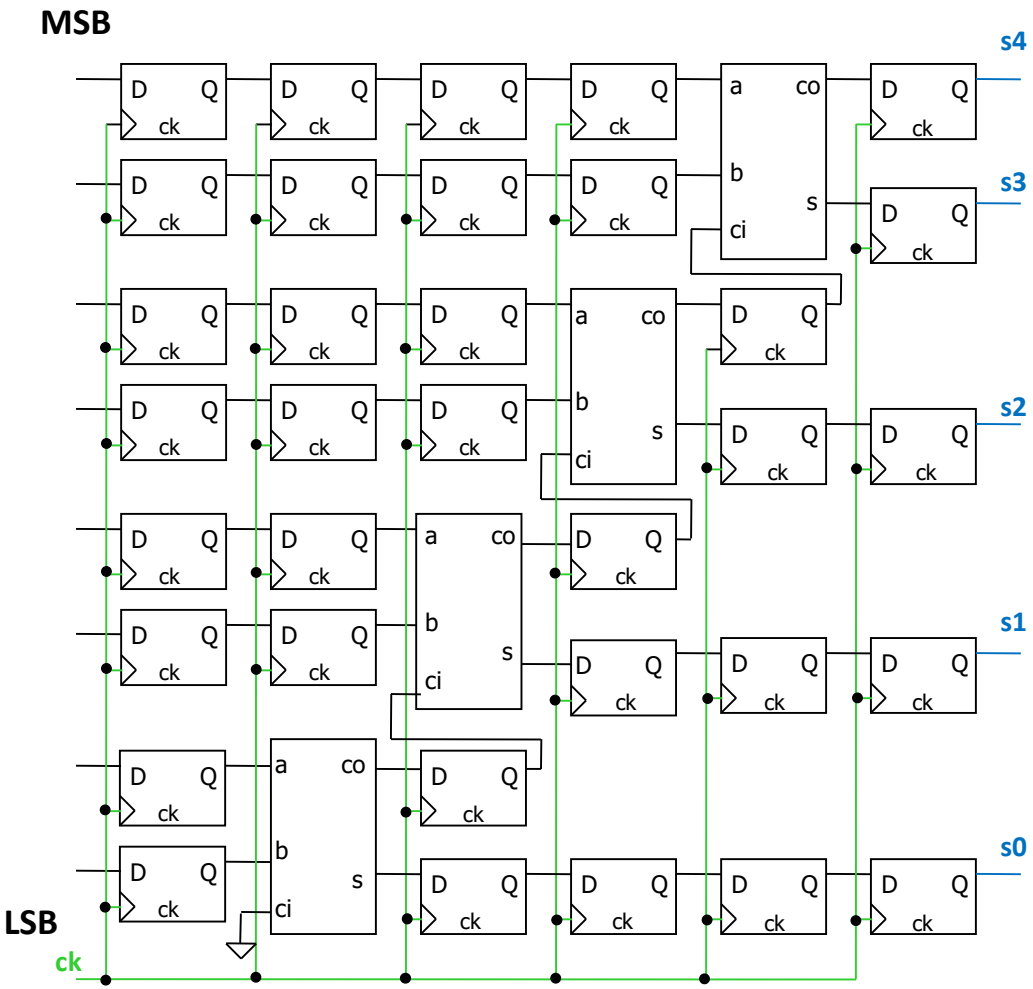
パイプライン加算器の動作

4クロック後



パイプライン加算器の動作

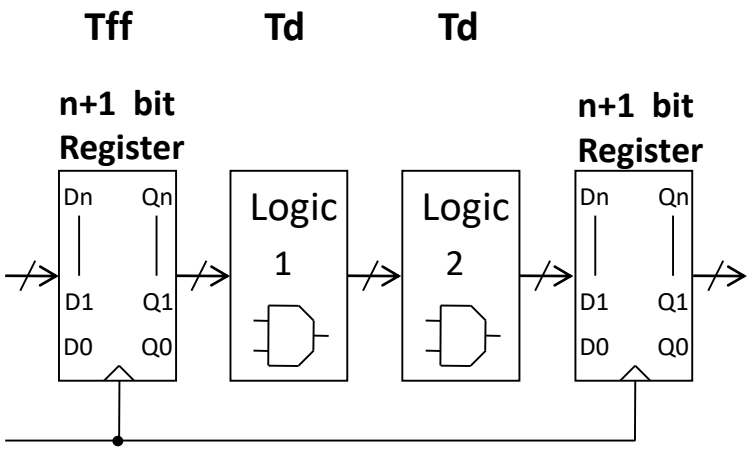
5クロック後



パイプライン構成による高速化

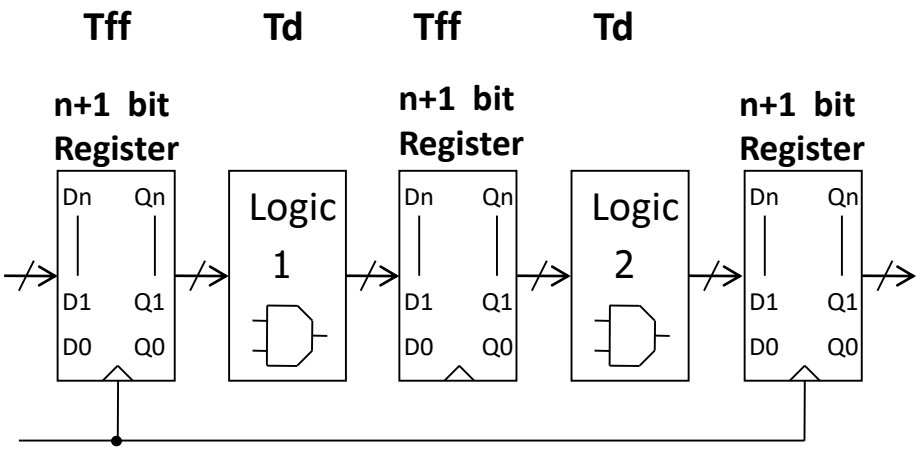
Pipeline Circuit

● **Circuit 1**



clock period T > Tff + 2Td + Tset

● **Circuit 2**



clock period T > Tff + Td + Tset

Td >> Tff + Tset ⇒ circuit 2 は circuit 1 の2倍高速

パイプライン構成の特徴

- バケツリレー　の原理
- レジスタを組み合せ回路の中間にいれればよい
- ２つの性能指標
  Throughput：出力が出てくるレート（１クロック）
  Latency：　ある入力の結果が出力されるまでの遅延　（５クロック）
- パイプライン構成では
  Throughput は良くなる
  Latency は数クロックかかる（良くない）
- フィードバック構成の使用は注意必要

桁上げ計算節約加算器
Carry Save Adder

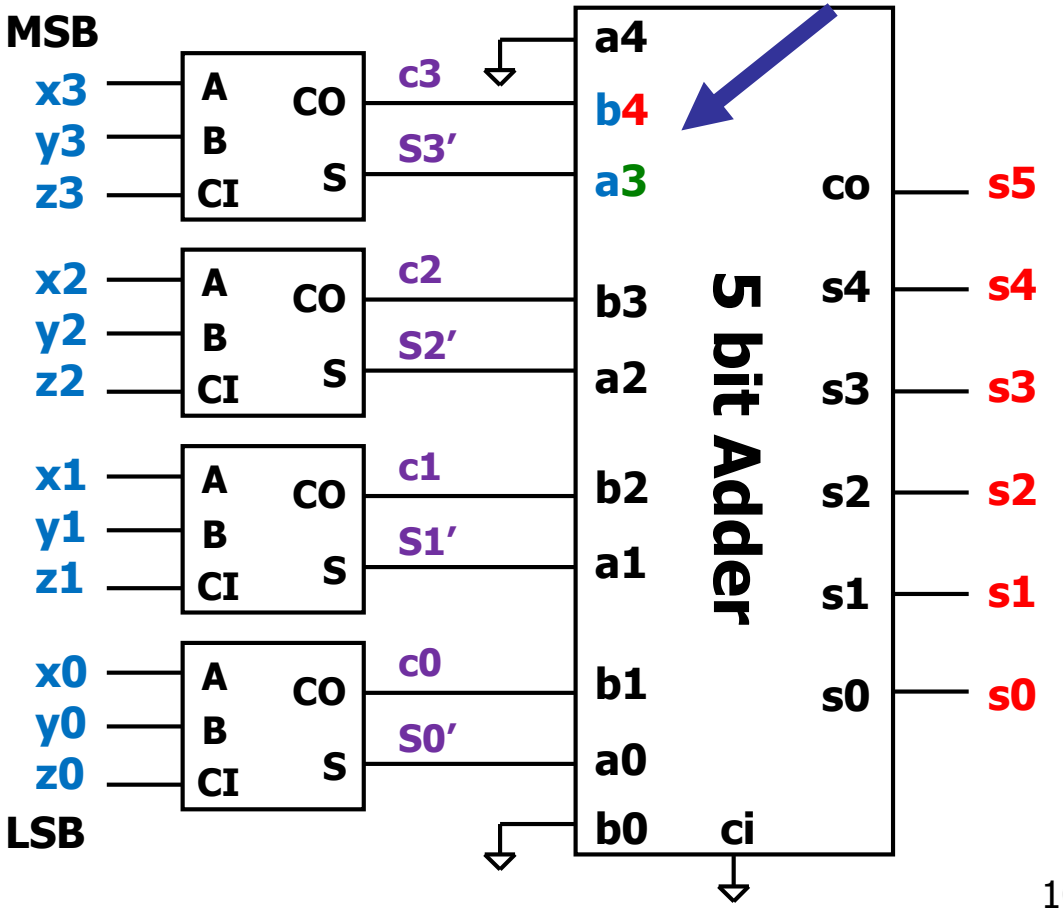
co は bn+1
s は an
に入力

3入力加算器　　S = X + Y + Z

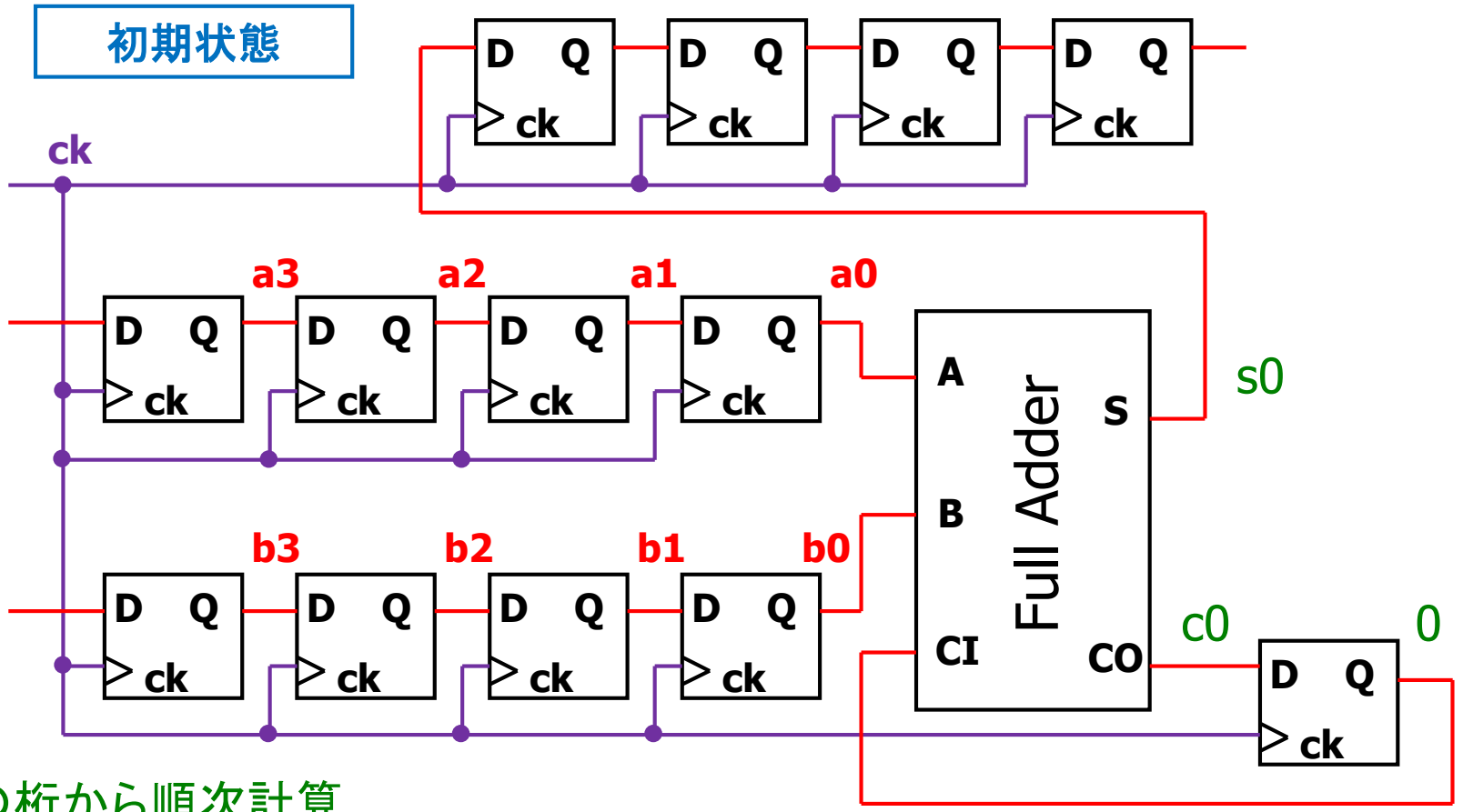
- 多入力加算
- 最後に1回だけ桁上げ計算

↓

- 高速
- 回路規模　小

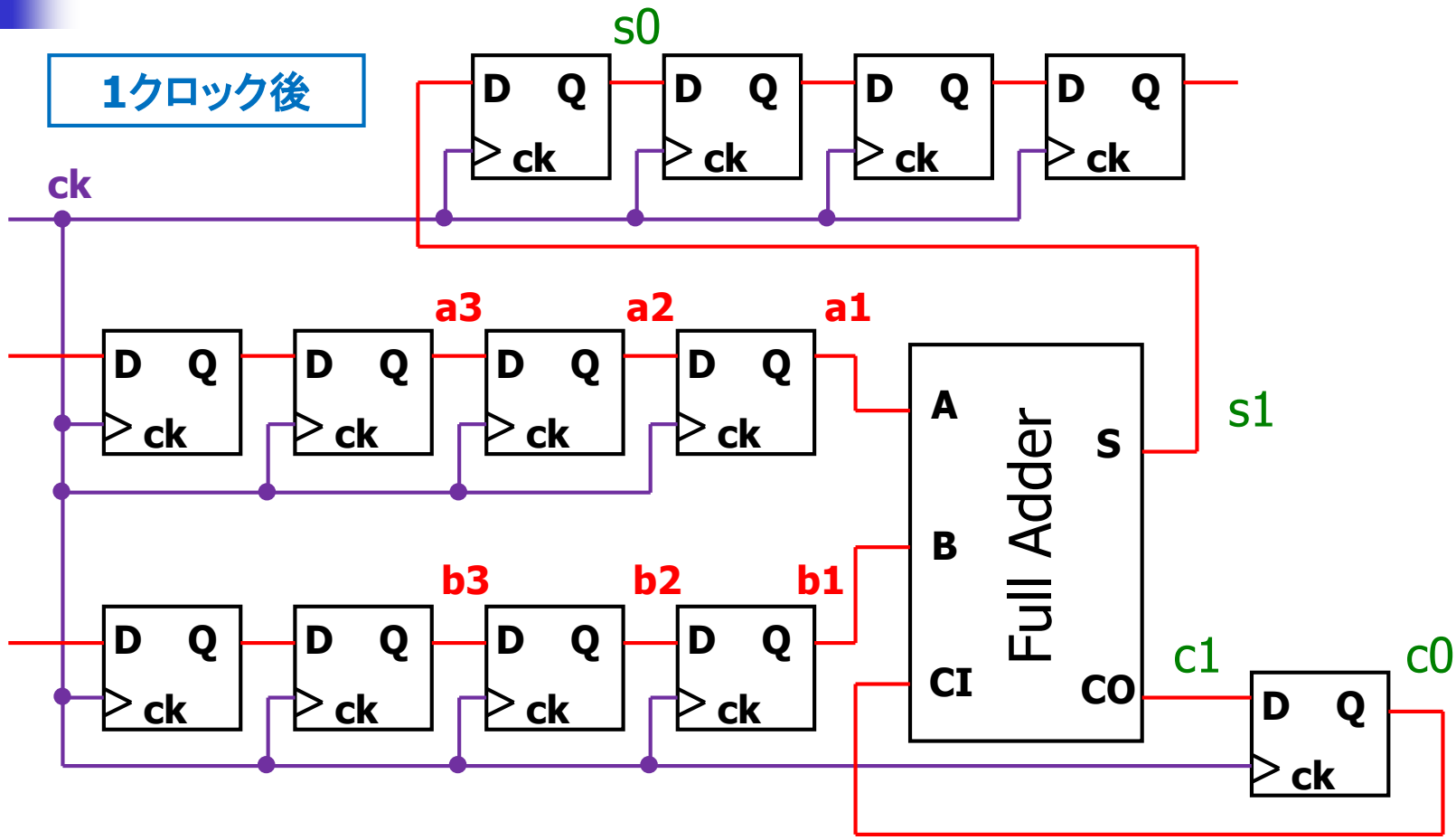


ビットシリアル加算器の構成
全加算器は1個でよい

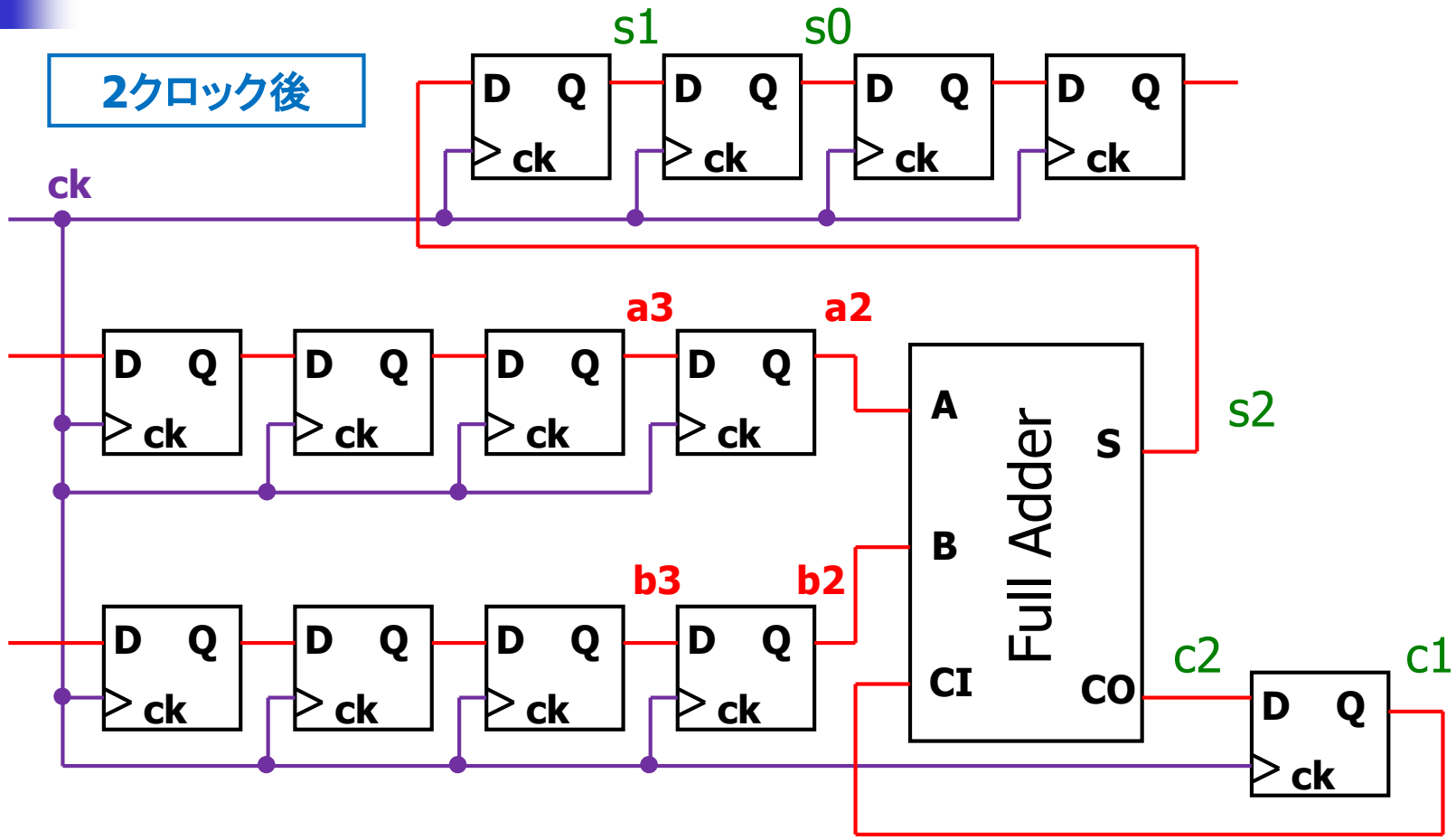


下の桁から順次計算
加算演算に桁数だけクロック数が必要

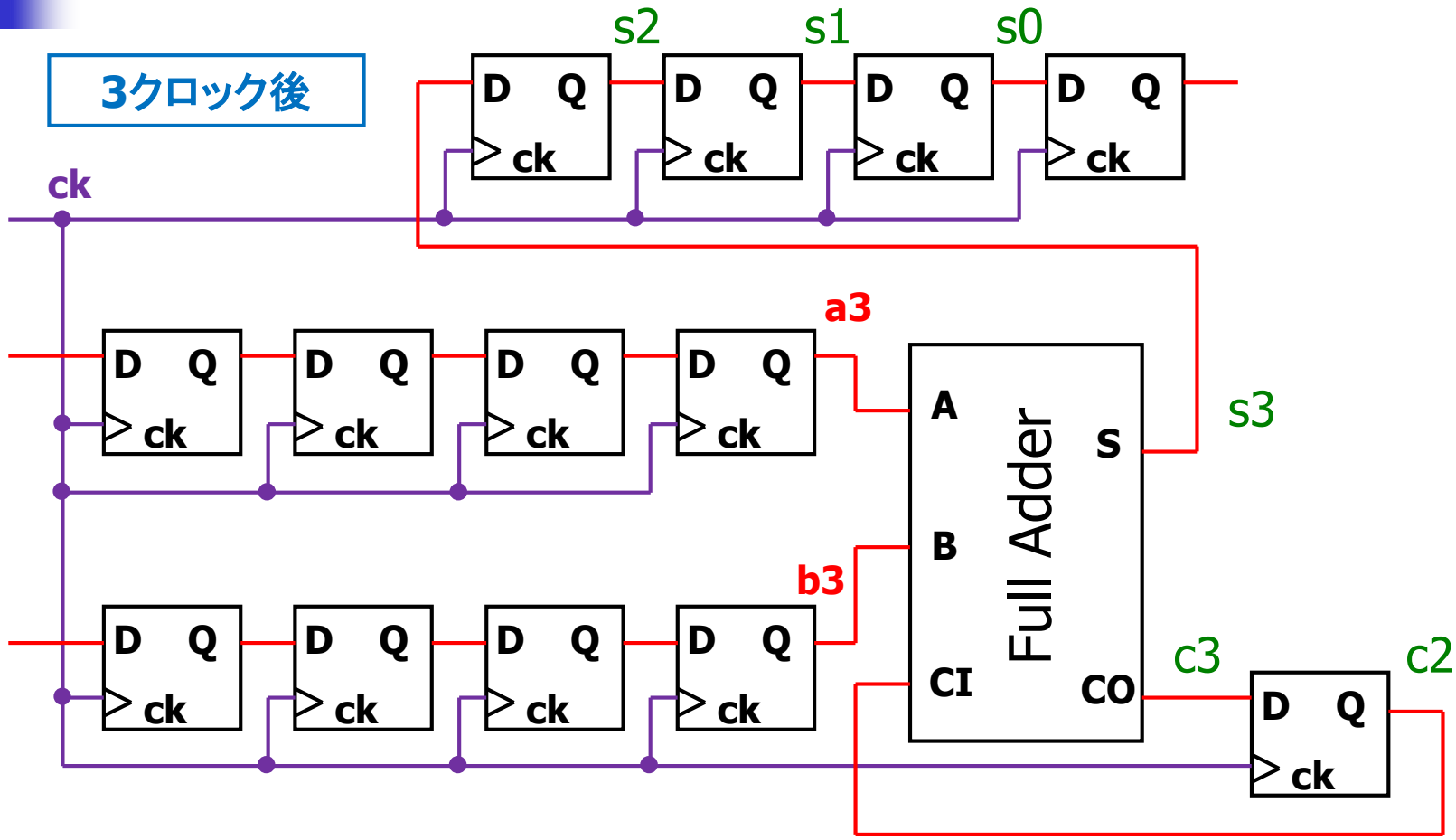
ビットシリアル加算器の動作



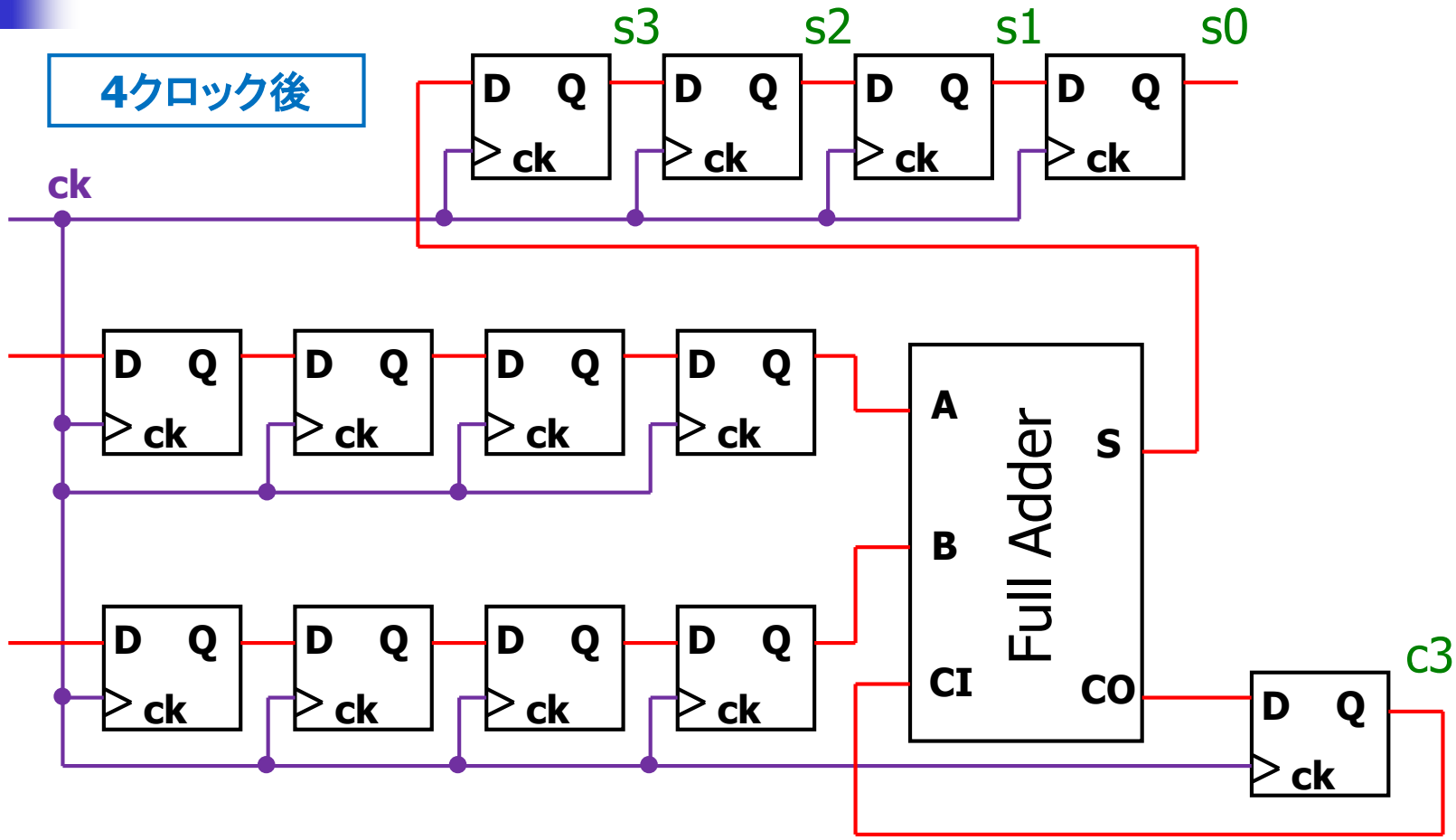
ビットシリアル加算器の動作



ビットシリアル加算器の動作



ビットシリアル加算器の動作



ビットシフトと乗算

10進数で　x10, x100, ÷1000 等は簡単
同様に2進数で x2, x4、÷8 等は簡単

● 1 bit left shift ⟷ ×2

b4 b3 b2 b1 b0

0 0 0 1 1 → 3

0 0 1 1 0 → 3 ×2=6

0 1 1 0 0 → 6 ×2=12

1 1 1 0 1 → -3

1 1 0 1 0 → -3 ×2=-6

1 0 1 0 0 → -6 ×2=-12

●1 bit right shift ⟷ ÷ 2

b4 b3 b2 b1 b0

0 1 0 0 0 → 8

0 0 1 0 0 → 8 / 2=4

0 0 0 1 0 → 4 / 2=2

1 1 0 0 0 → -8

1 1 1 0 0 → -8 / 2=-4

1 1 1 1 0 → -4 / 2=-2

b4	b3	b2	b1	b0	十進数
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	-16
1	0	0	0	1	-15
1	0	0	1	0	-14
1	0	0	1	1	-13
1	0	1	0	0	-12
1	0	1	0	1	-11
1	0	1	1	0	-10
1	0	1	1	1	-9
1	1	0	0	0	-8
1	1	0	0	1	-7
1	1	0	1	0	-6
1	1	0	1	1	-5
1	1	1	0	0	-4
1	1	1	0	1	-3
1	1	1	1	0	-2
1	1	1	1	1	-1

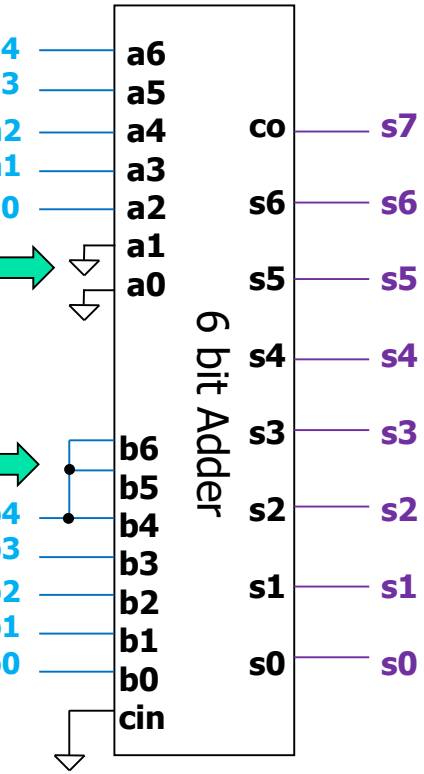
ビットシフトで2^n 倍乗算を実現

S= 4×A+B

乗算器を
使用せずに実現

2ビット
左シフトで
X4 を実現

2の補数表現で
符号ビットの拡張



デジタル乗算

2進数の乗算

```
   0101   (5)
X) 1011   (11)
-------------
   0101
  0101
 0000
0101
-------------
0110111   (55)
```

10進数の乗算

```
    437
X)  258
-------------
   3496
  2185
   874
-------------
  112746
```

2進デジタル乗算器

$$X = \sum_{i=0}^{m-1} X_i 2^i$$

$$Y = \sum_{j=0}^{n-1} Y_j 2^j$$

$$P = X \times Y = \sum_{i=0}^{m-1} X_i 2^i \bullet \sum_{j=0}^{n-1} Y_j 2^j$$

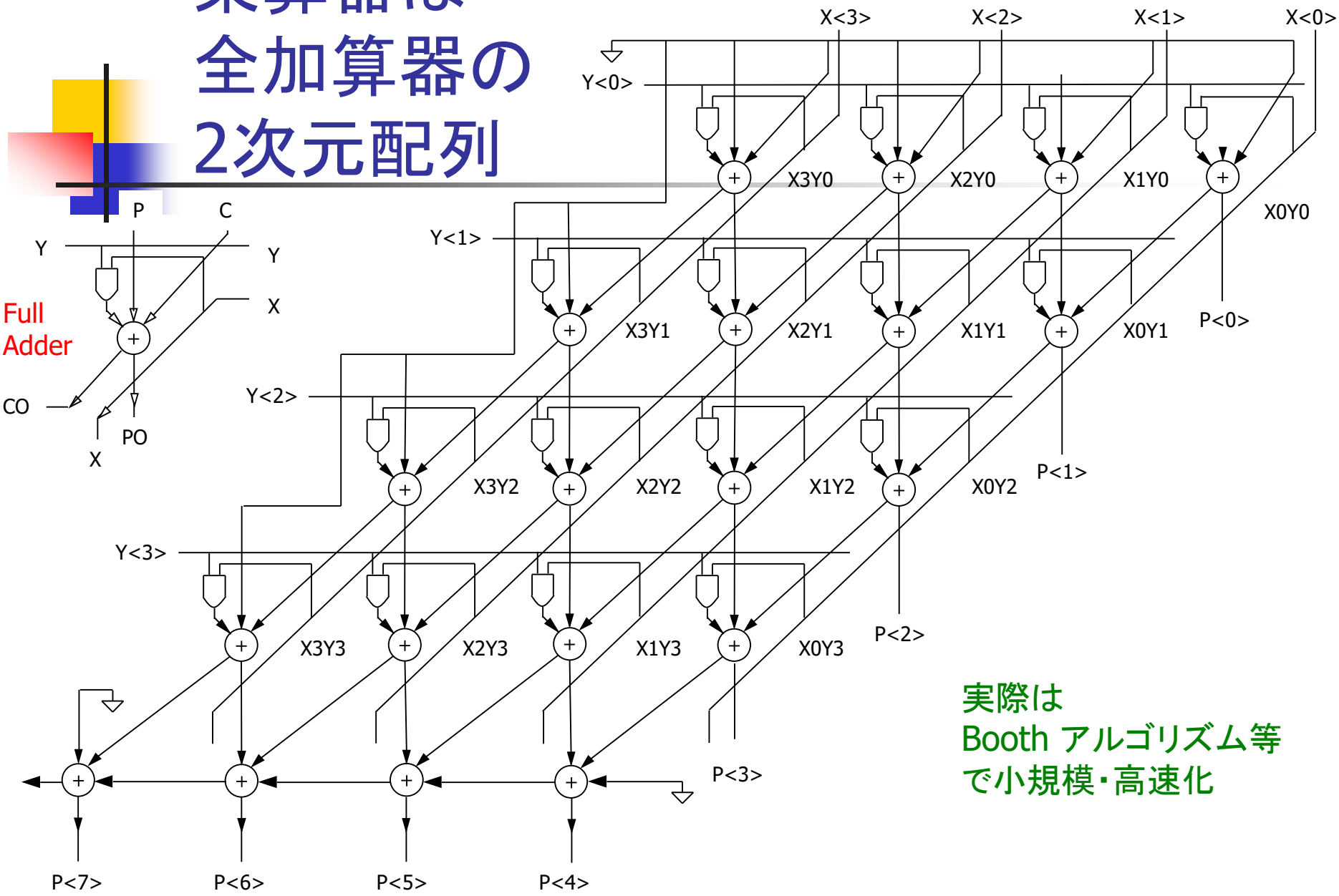
$$= \sum_{i=0}^{m-1} \bullet \sum_{j=0}^{n-1} Y_j 2^j \left(X_i Y_j\right) 2^{i+j}$$

$$= \sum_{k=0}^{m+n-1} P_k 2^k$$

4-bit Multiplier Partial Products

				X3	X2	X1	X0	Multiplicand
				Y3	Y2	Y1	Y0	Multiplier
				X3Y0	X2Y0	X1Y0	X0Y0	
			X3Y1	X2Y1	X1Y1	X0Y1		
		X3Y2	X2Y2	X1Y2	X0Y2			
	X3Y3	X2Y3	X1Y3	X0Y3				
P7	P6	P5	P4	P3	P2	P1	P0	Product

乗算器は全加算器の2次元配列



実際は
Booth アルゴリズム等
で小規模・高速化

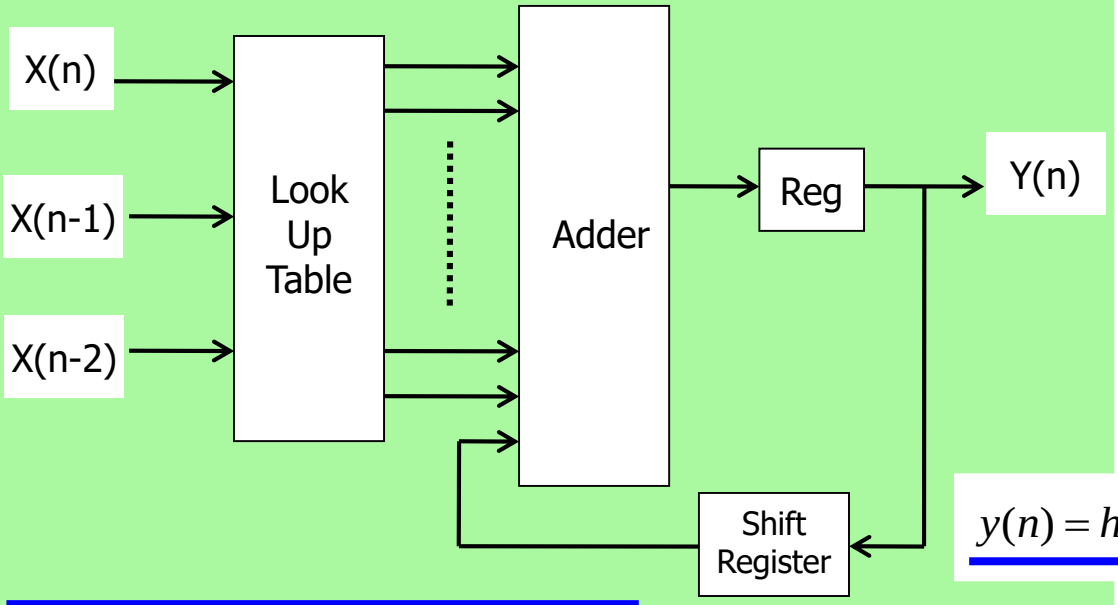
内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- **事例1：　SAR ADC+分散型積和演算回路**
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

乗算器を使用しない
定係数積和演算回路

定係数の内積演算をルックアップ・テーブル（ROM等）により実現する計算手法

分散型積和演算



分散型積和演算回路構成

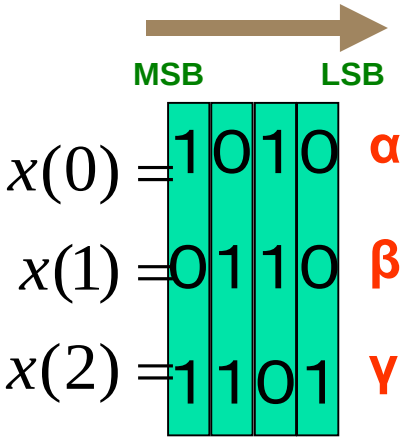
上位ビットからのシリアル演算

$$y(n) = h_0 x(n) + h_1 x(n-1) + h_0 x(n-2)$$

デジタル入力

分散型積和演算　動作原理

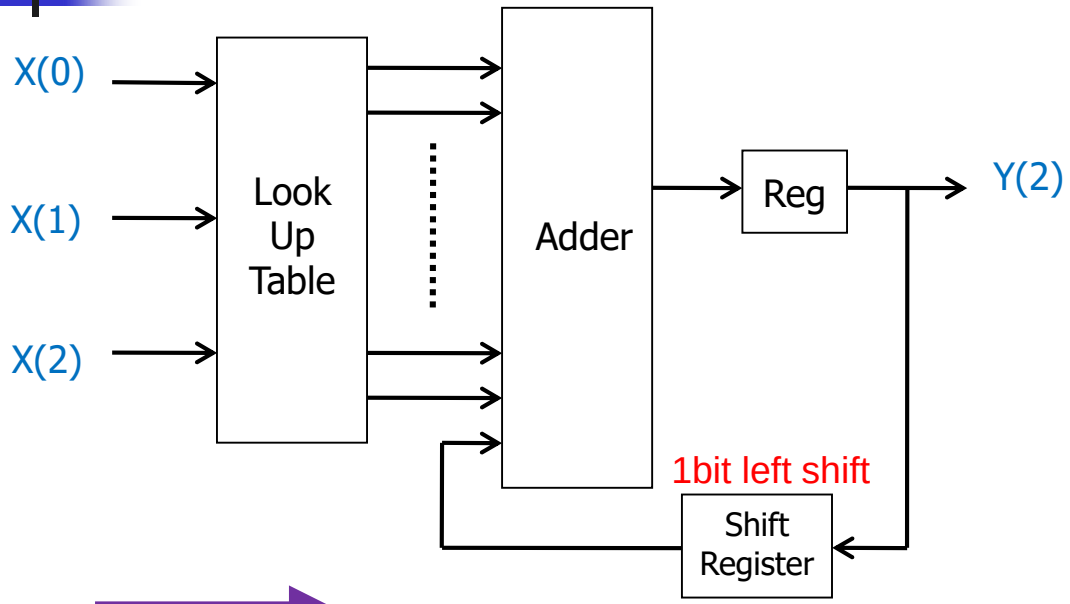
$$y(n) = h_0 x(n) + h_1 x(n-1) + h_0 x(n-2)$$



Look Up Table

γ	β	α	Y(n)
0	0	0	0
0	0	1	h0
0	1	0	h1
0	1	1	h0+h1
1	0	0	h0
1	0	1	2*h0
1	1	0	h1+h0
1	1	1	2*h0+h1

分散型積和演算　動作原理



MSB　LSB

x(0)=1 0 1 0

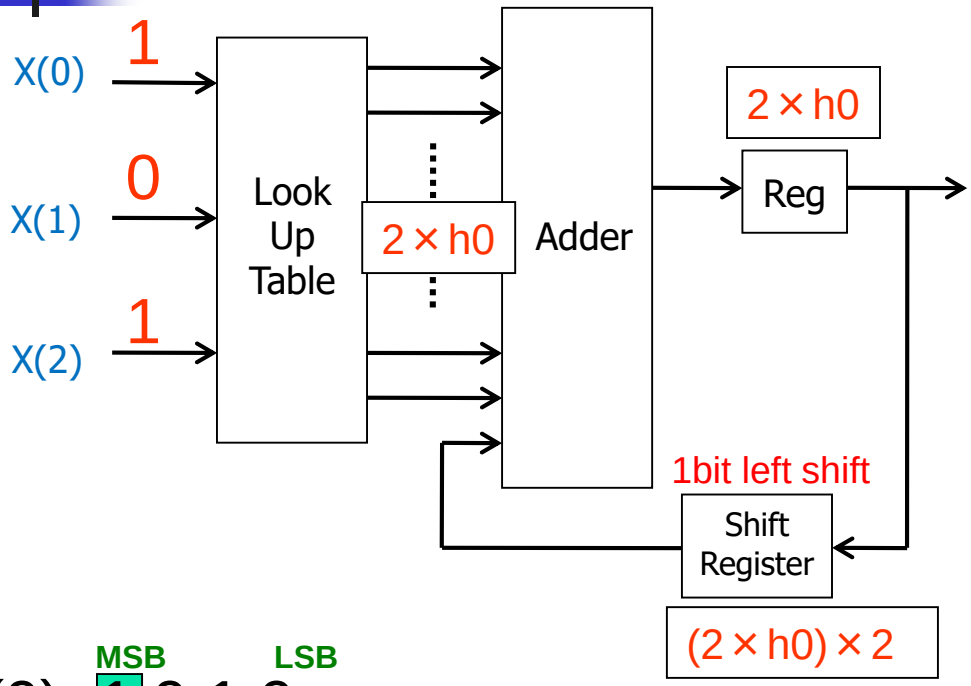
x(1)=0 1 1 0

x(2)=1 1 0 1

右シフトに処理

γ	β	α	Y(n)
0	0	0	0
0	0	1	h0
0	1	0	h1
0	1	1	h0+h1
1	0	0	h0
1	0	1	2*h0
1	1	0	h1+h0
1	1	1	2*h0+h1

分散型積和演算　動作原理



MSB　LSB

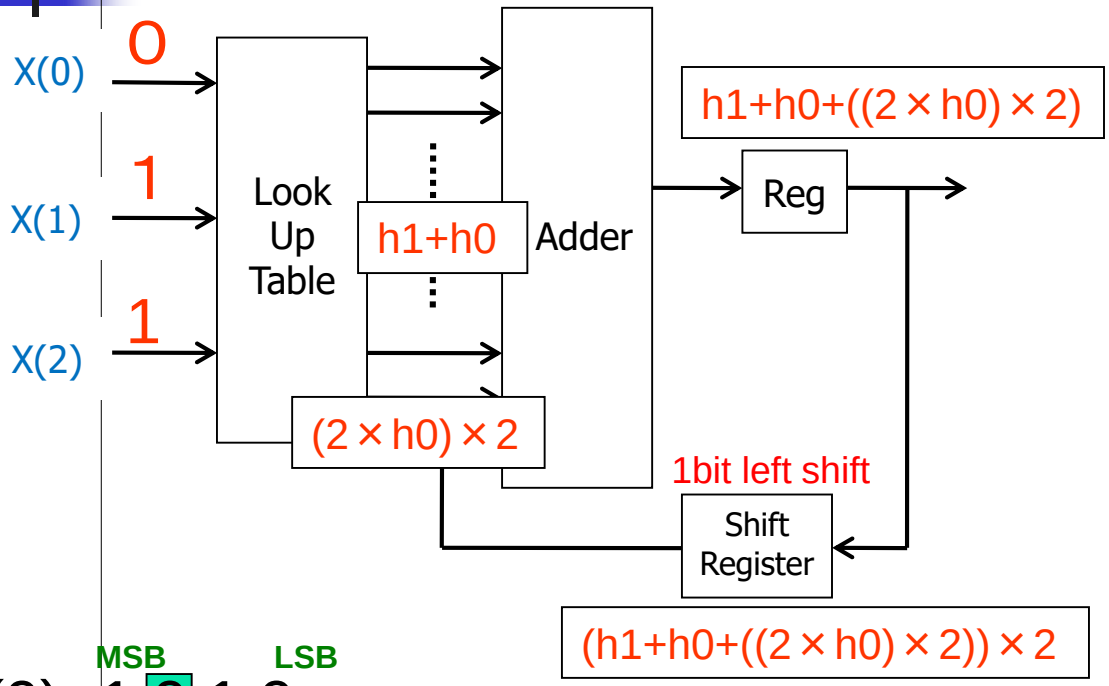
x(0)=1 0 1 0

x(1)=0 1 1 0

x(2)=1 1 0 1

γ	β	α	Y(n)
0	0	0	0
0	0	1	h0
0	1	0	h1
0	1	1	h0+h1
1	0	0	h0
1	0	1	2*h0
1	1	0	h1+h0
1	1	1	2*h0+h1

分散型積和演算　動作原理



γ	β	α	Y(n)
0	0	0	0
0	0	1	h0
0	1	0	h1
0	1	1	h0+h1
1	0	0	h0
1	0	1	2*h0
1	1	0	h1+h0
1	1	1	2*h0+h1

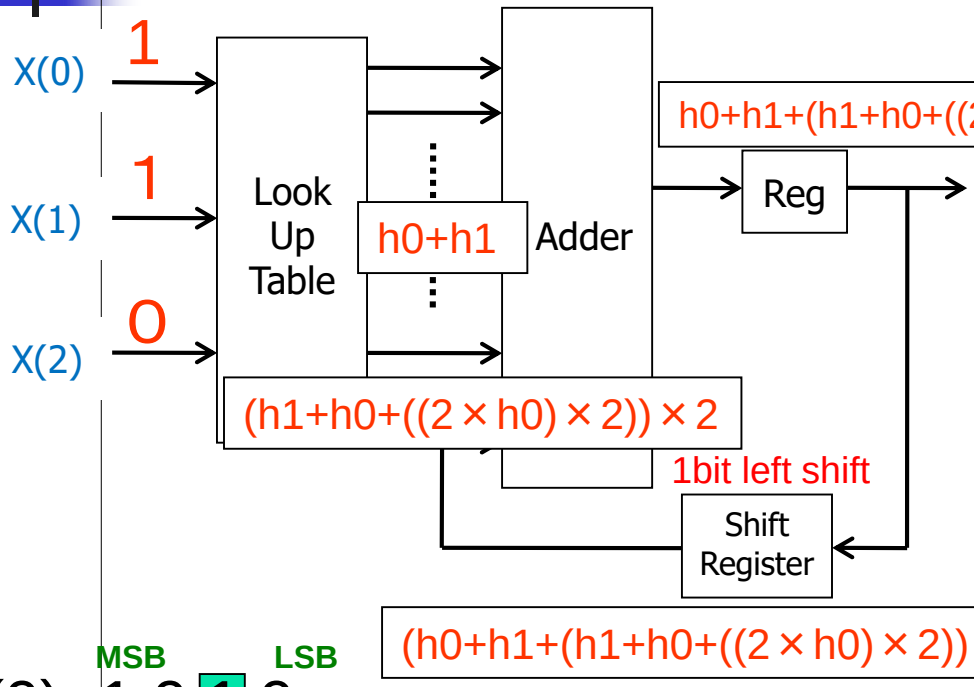
MSB　　LSB

x(0)=1 0 1 0

x(1)=0 1 1 0

x(2)=1 1 0 1

分散型積和演算　動作原理



γ	β	α	Y(n)
0	0	0	0
0	0	1	h0
0	1	0	h1
0	1	1	h0+h1
1	0	0	h0
1	0	1	2*h0
1	1	0	h1+h0
1	1	1	2*h0+h1

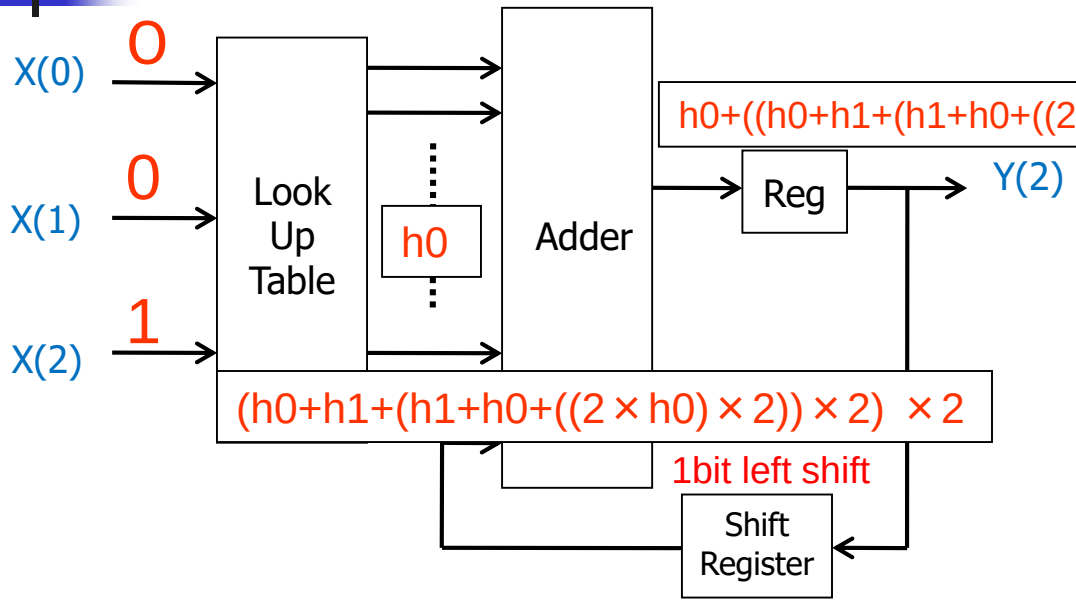
MSB　　LSB

x(0)=1 0 1 0

x(1)=0 1 1 0

x(2)=1 1 0 1

分散型積和演算　動作原理



γ	β	α	Y(n)
0	0	0	0
0	0	1	h0
0	1	0	h1
0	1	1	h0+h1
1	0	0	h0
1	0	1	2*h0
1	1	0	h1+h0
1	1	1	2*h0+h1

MSB　LSB

x(0)=1 0 1 0

x(1)=0 1 1 0

x(2)=1 1 0 1

分散型積和演算回路のメリット

- 乗算器を使用しない ⇨ 小規模回路・低消費電力
- 積和演算全体の計算時間

  複数の積和演算項を同時にビットシリアル演算

  ⇨ 乗算器を使用する構成と同等

  （積和演算項が多い場合は　同等以上）
- 逐次比較近似ADC　＋　積和演算

  (デジタル制御電源等)

  逐次比較近似ADCは上位ビットから出力される

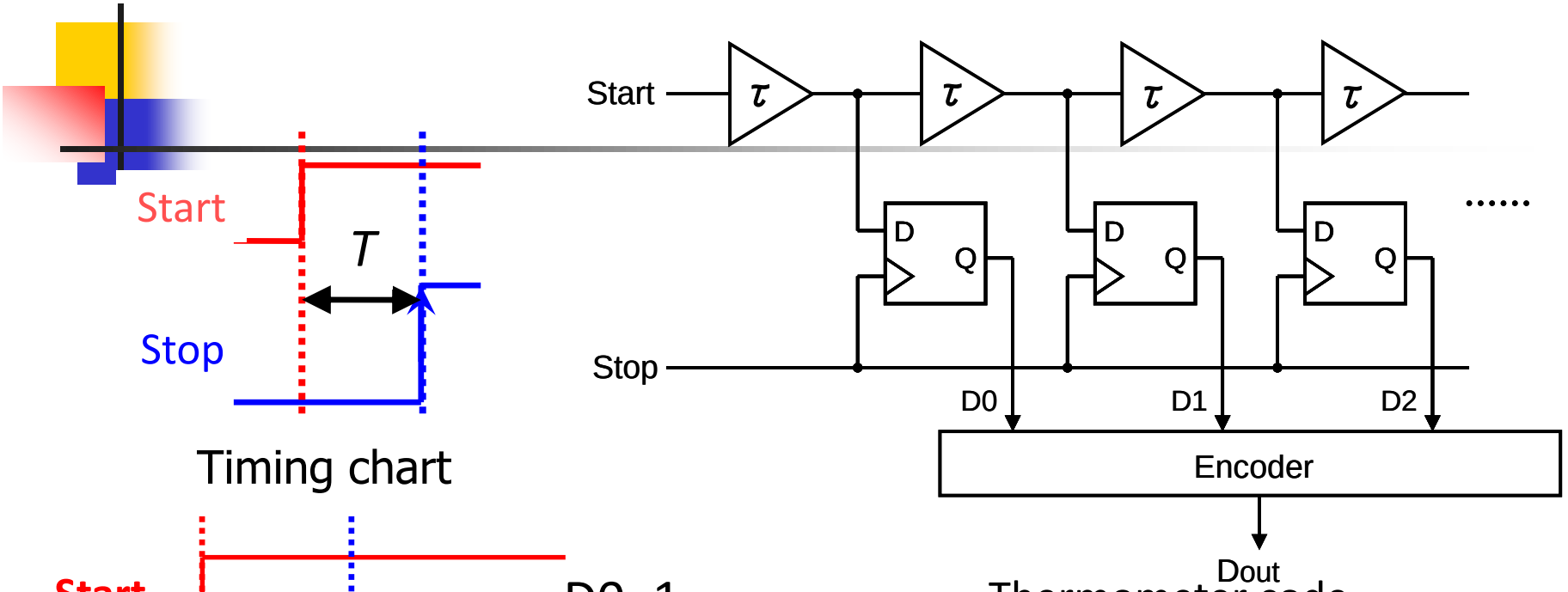
  ⇨ 積和演算をすぐ開始できる （latency が小）
- 分散型積和演算回路単体では MSB からではなく

  LSB から演算することも可能

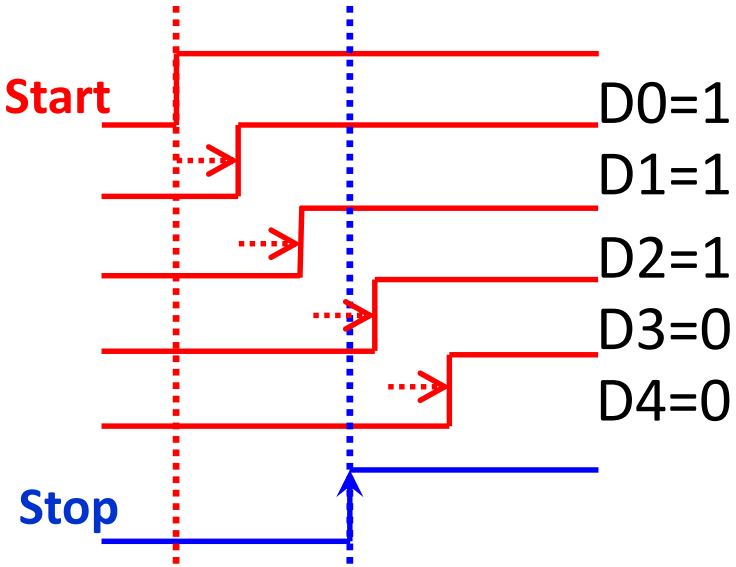
内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- **事例2：　自己校正機能をもったTDC回路**
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

基本TDC (Time-to-Digital Converter) 回路



Timing chart



Thermometer code

⇩

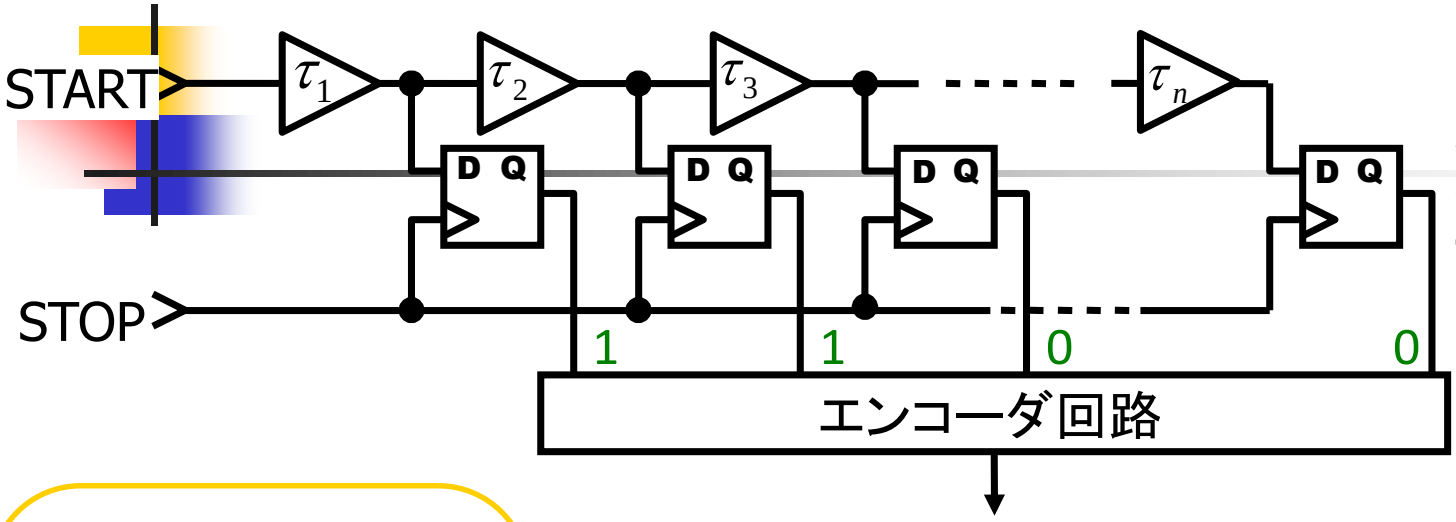
binary code

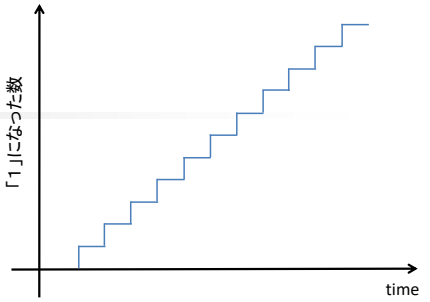
⇩

ディレイタップ何段に相当するかを測定

CMOS微細化とともに高性能化

TDC単調性の問題





TDC単調増加

Dout=2

各DFF出力Dout

0 0 0 0 0 0 0 0	0
1 0 0 0 0 0 0 0	1
1 1 0 0 0 0 0 0	2
1 1 1 0 0 0 0 0	3
1 1 1 1 0 0 0 0	4
1 1 1 1 1 0 0 0	5
1 1 1 1 1 1 0 0	6
1 1 1 1 1 1 1 0	7
1 1 1 1 1 1 1 1	8

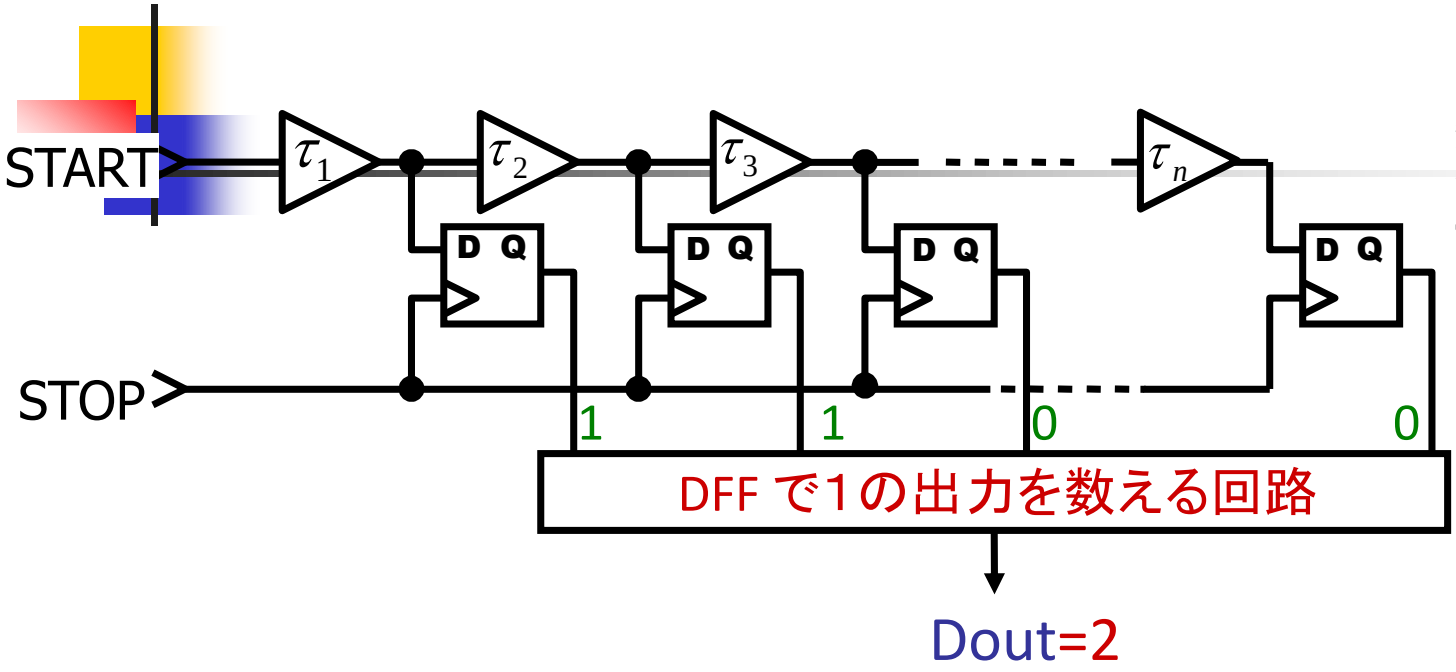
DFF出力のバブルエラー

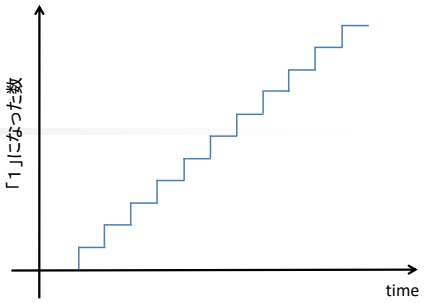
1 0 1 0 0 0 0 0	2
1 1 1 0 0 0 0 0	3
1 1 1 0 1 0 0 0	4
1 1 1 0 1 0 1 0	5
1 1 1 0 1 0 1 1	6

バッファの遅延ばらつき
DFFのオフセットばらつき

バブルエラーが発生

TDC単調性の確保





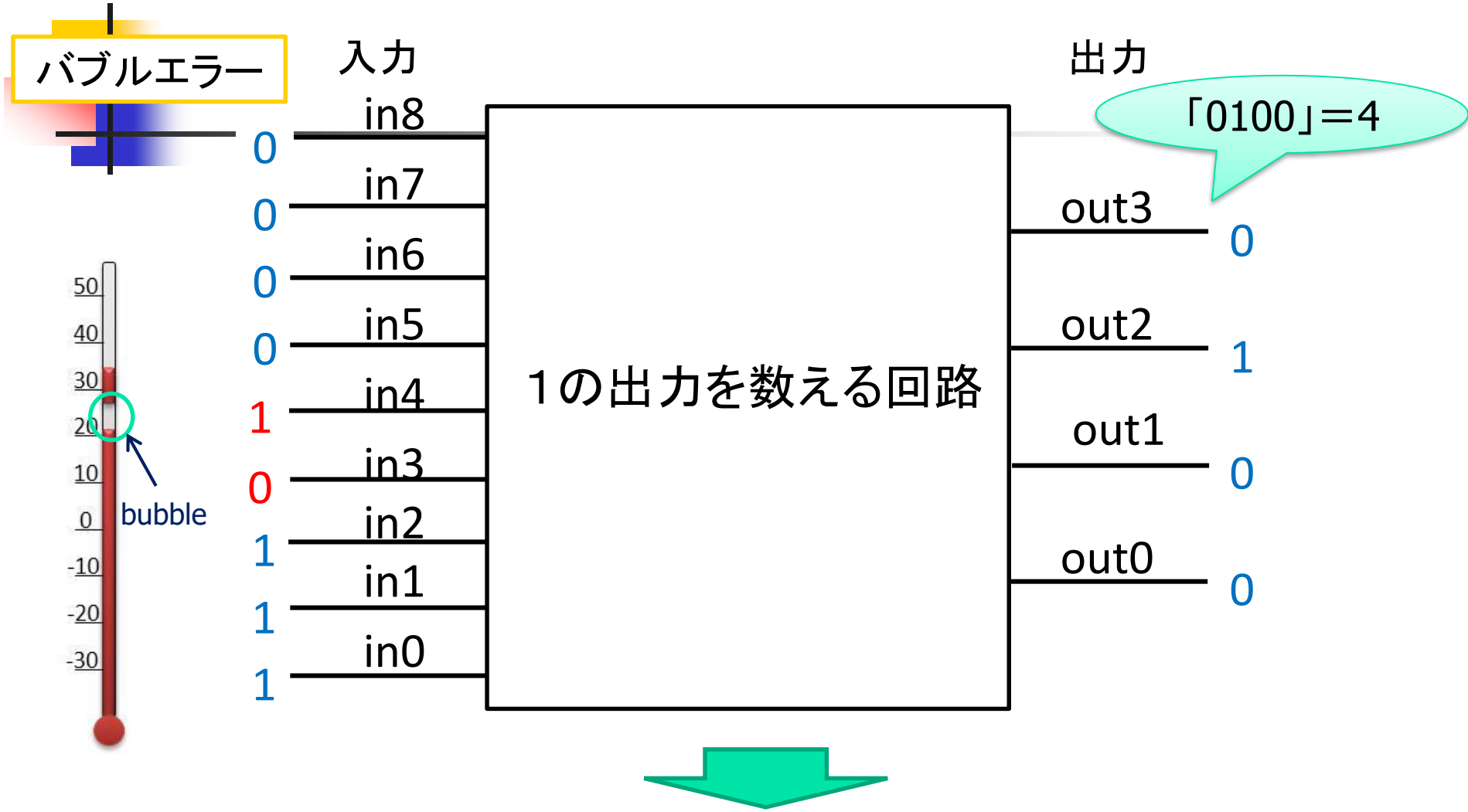
TDC単調増加

時間遅延τ(s)ごとに
DFFの出力「1」の数が
単調に増加

エンコーダ回路をDFFの出力が1のものを数える回路

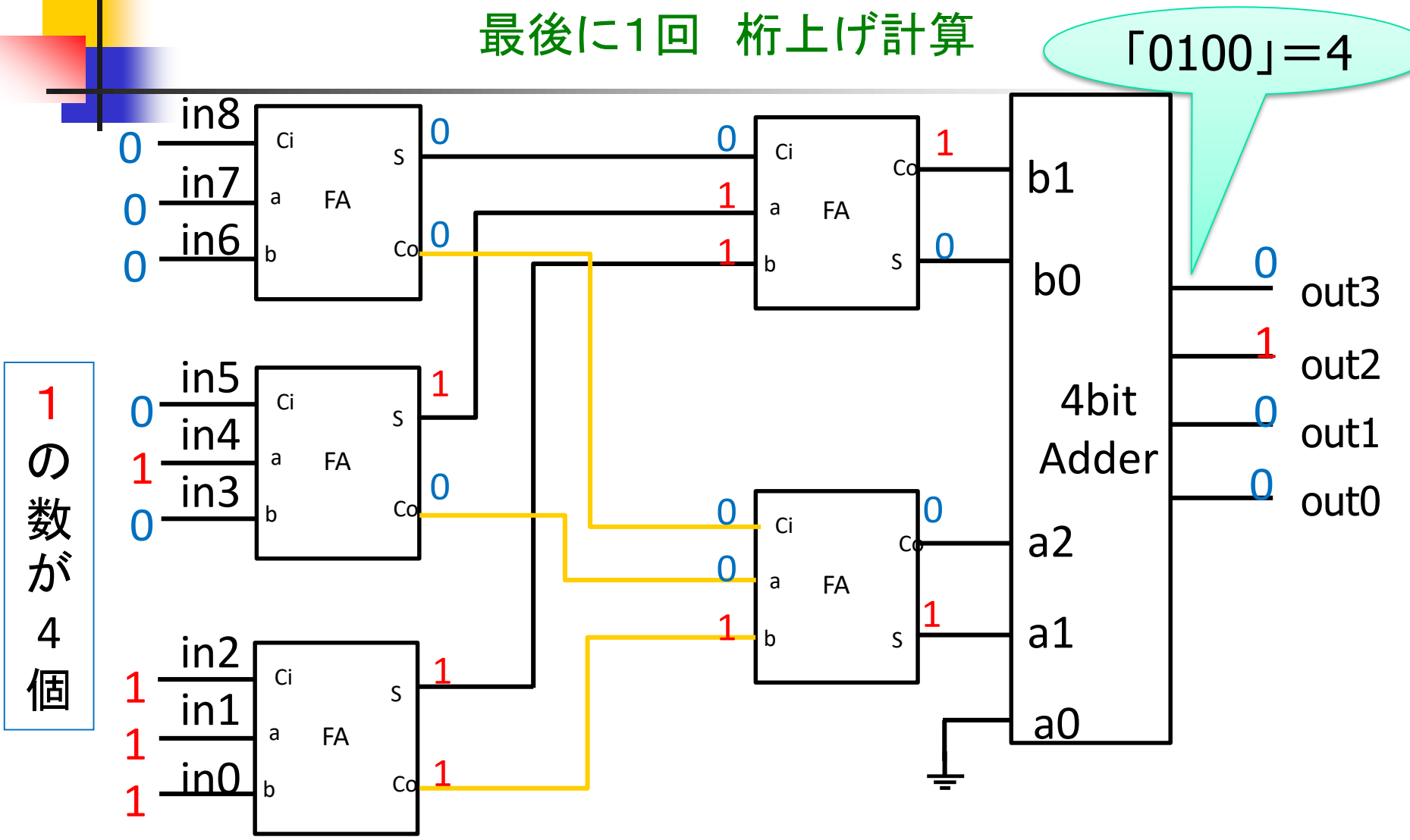
バブルエラーがある場合も単調増加の出力が可能

DFF で1の出力を数える回路



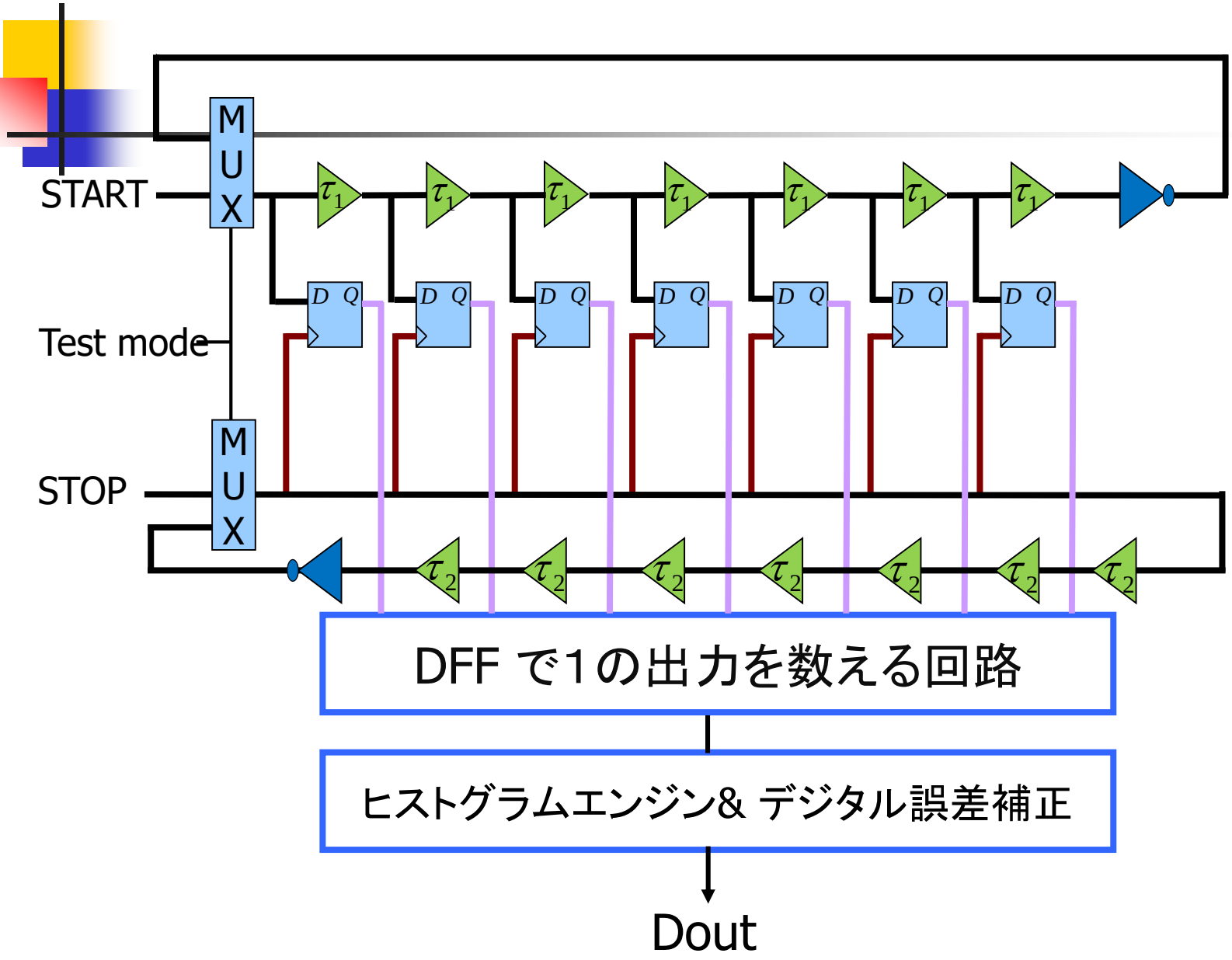
バブルエラーが起きても正しい出力が得られる

DFFの1の出力個数を数える回路

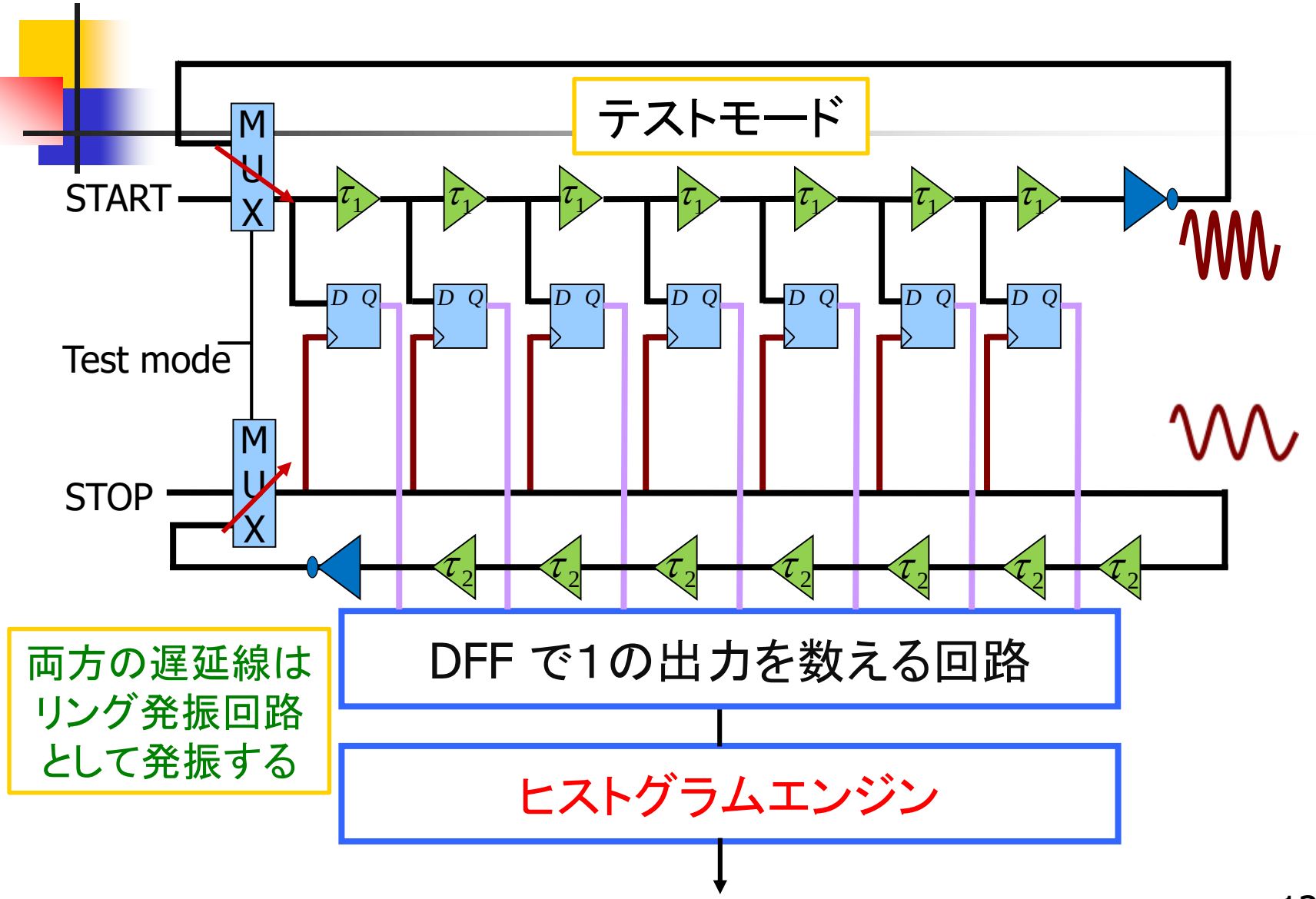


Carry Save Adder

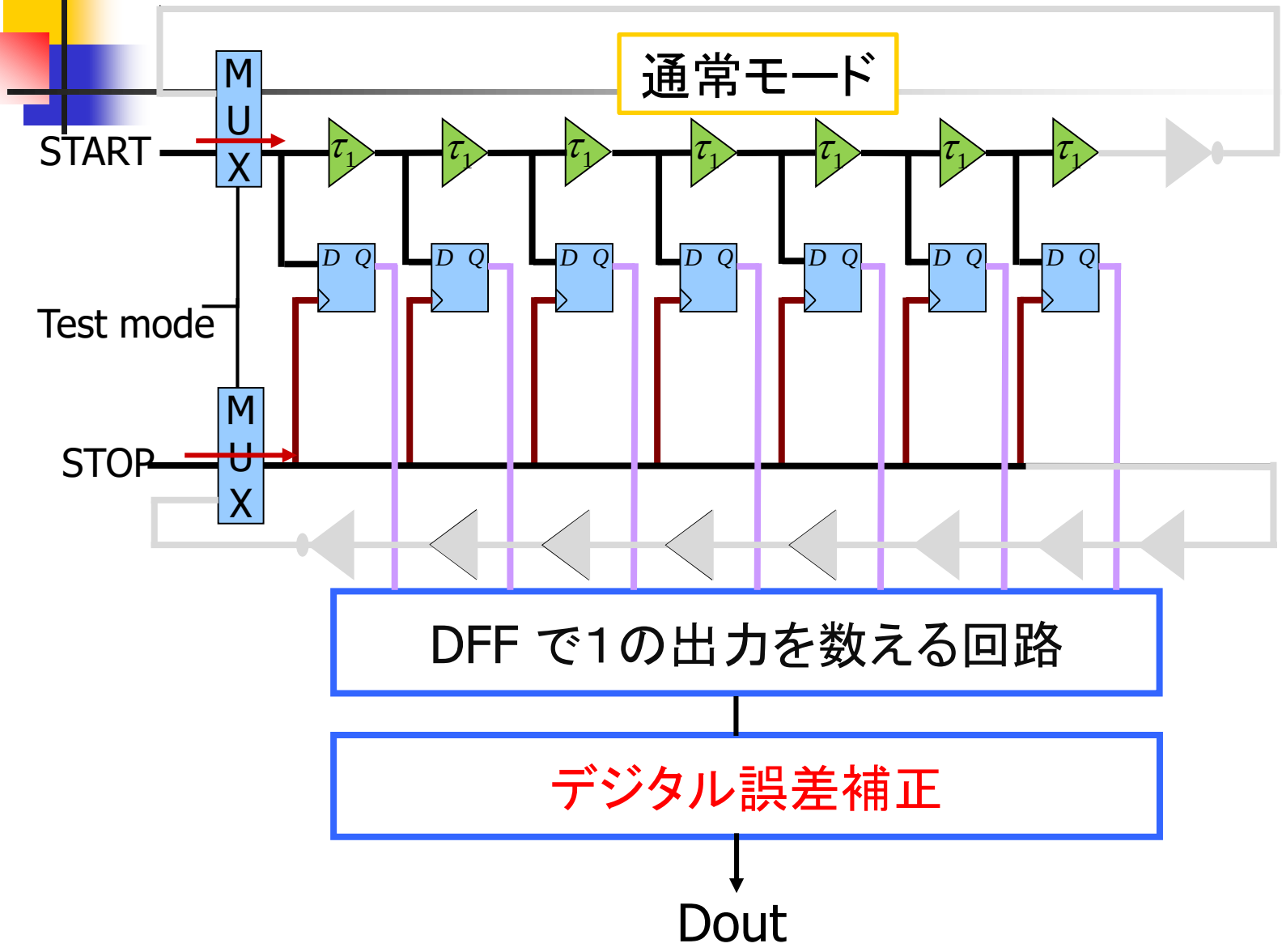
自己校正機能を備えたTDC回路の構成



自己校正機能を備えたTDC回路の構成



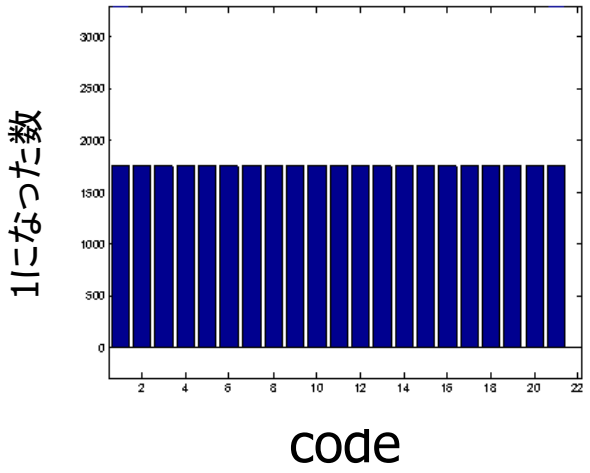
自己校正機能を備えたTDC回路の構成



自己校正の原理 （ヒストグラム法）

テストモード

両方のリング発振器は同期していない（無相関）



TDCが完全に線形

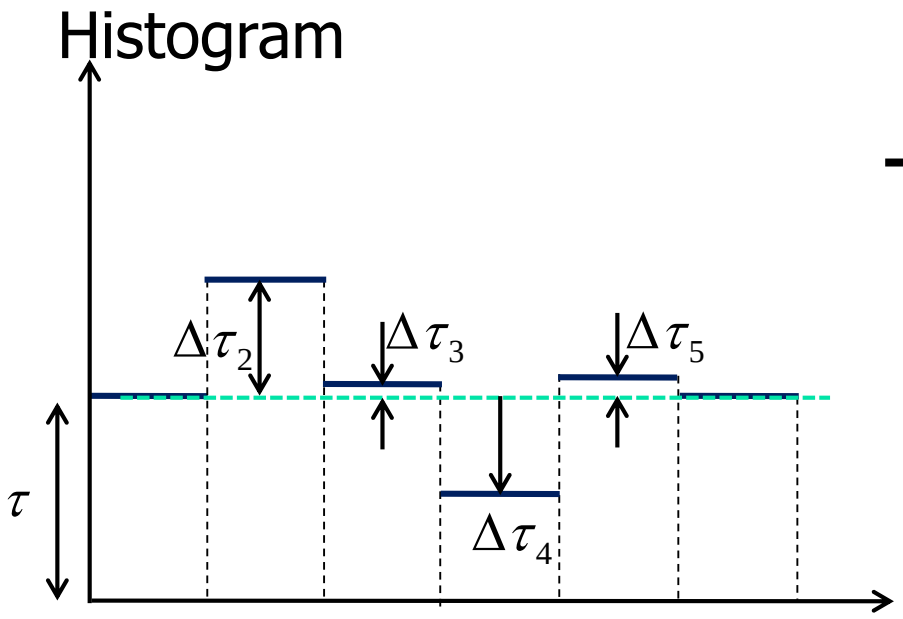
各出現コードの確率が等しい

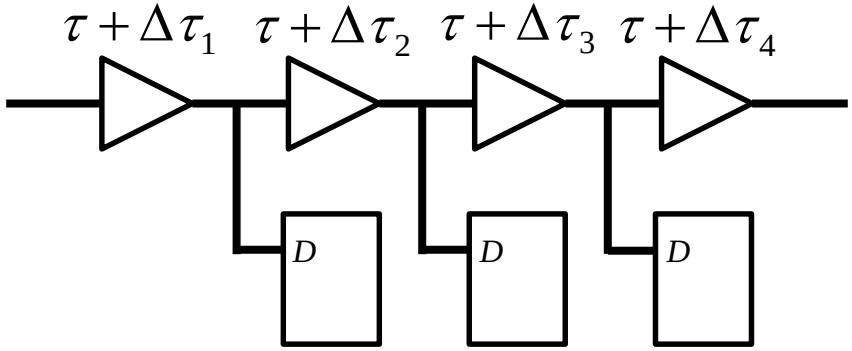
・充分多くの点数をとれば各デジタルコードの
ヒスト グラムは同一になる
・逆に、TDCのヒストグラムデータから
DNL, INL を計算

自己校正の原理 (非線形性の同定)

$\tau + \Delta\tau$ TDCが非線形

TDCの非線形性は遅延ばらつき $\Delta\tau$ によって生じる
INLをヒストグラムより求め逆関数を計算



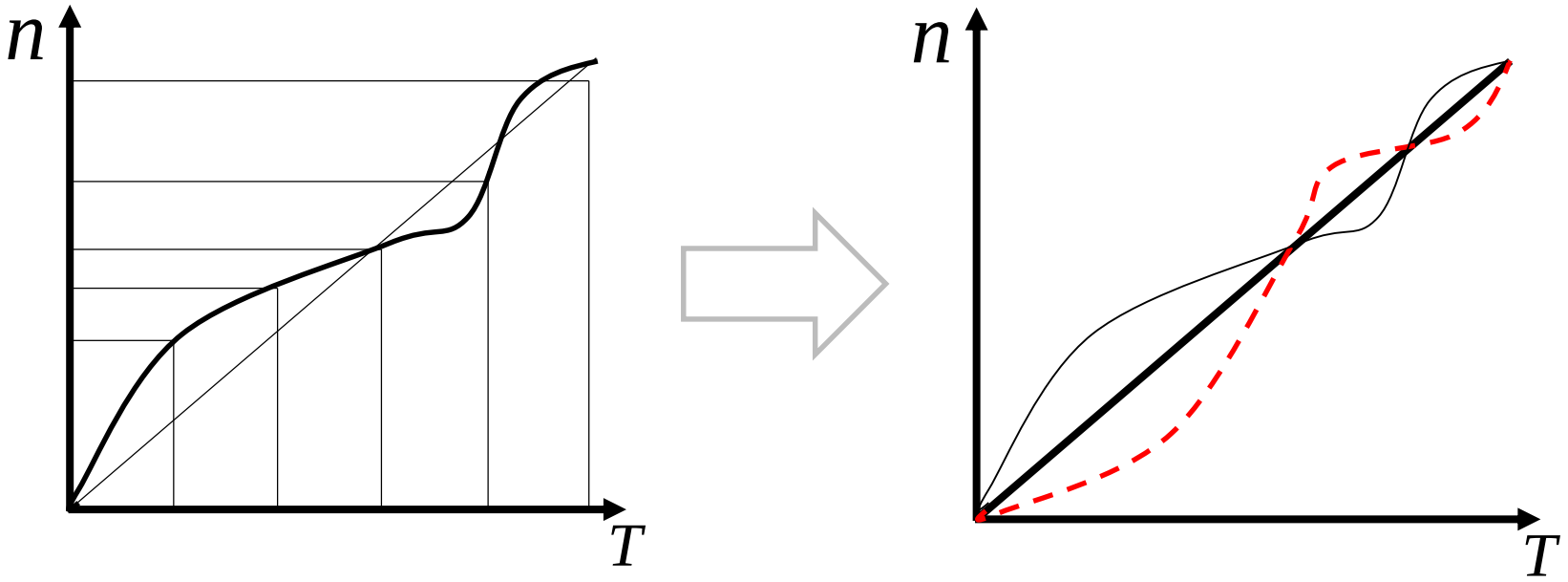


自己校正の原理 (非線形性の補正)

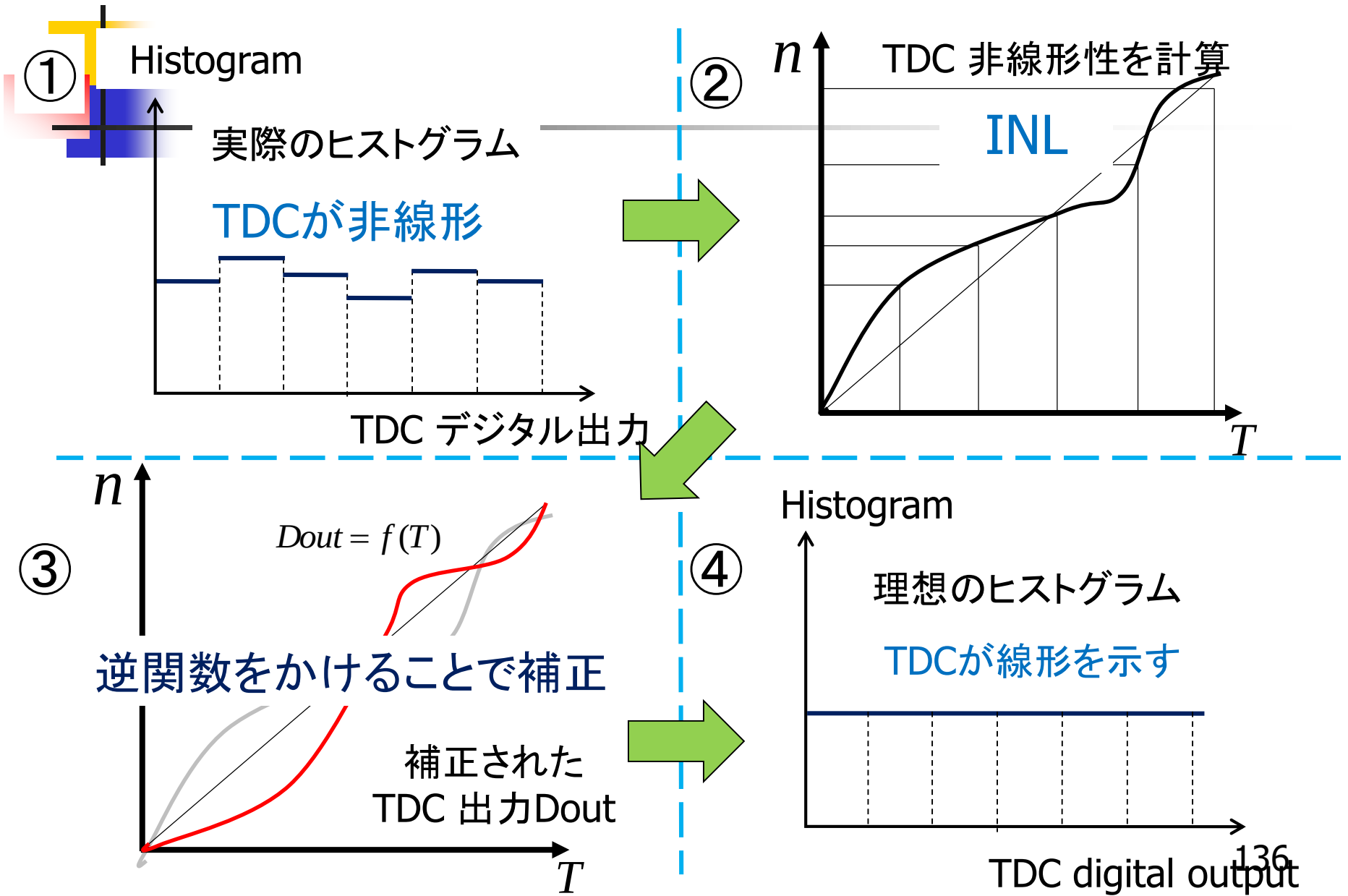
通常モード

非線形性の逆関数をデジタル的にかける

線形性が得られる



非線形性の自己校正

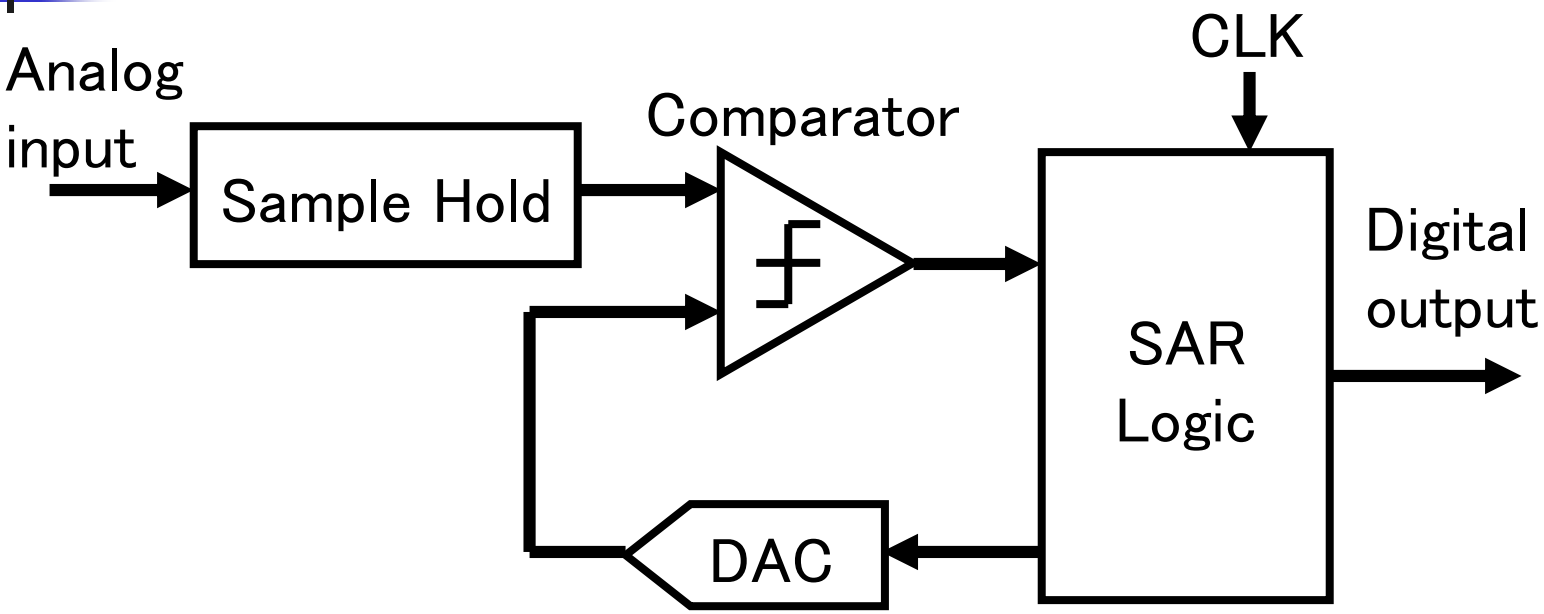

①
Histogram
実際のヒストグラム
TDCが非線形
TDC デジタル出力
②
n
TDC 非線形性を計算
INL
T
③
n
Dout = f(T)
逆関数をかけることで補正
補正された
TDC 出力Dout
T
④
Histogram
理想のヒストグラム
TDCが線形を示す
TDC digital output

内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- **事例3：　冗長アルゴリズムを用いたSAR ADC**
- 最後に

逐次比較近似ADC

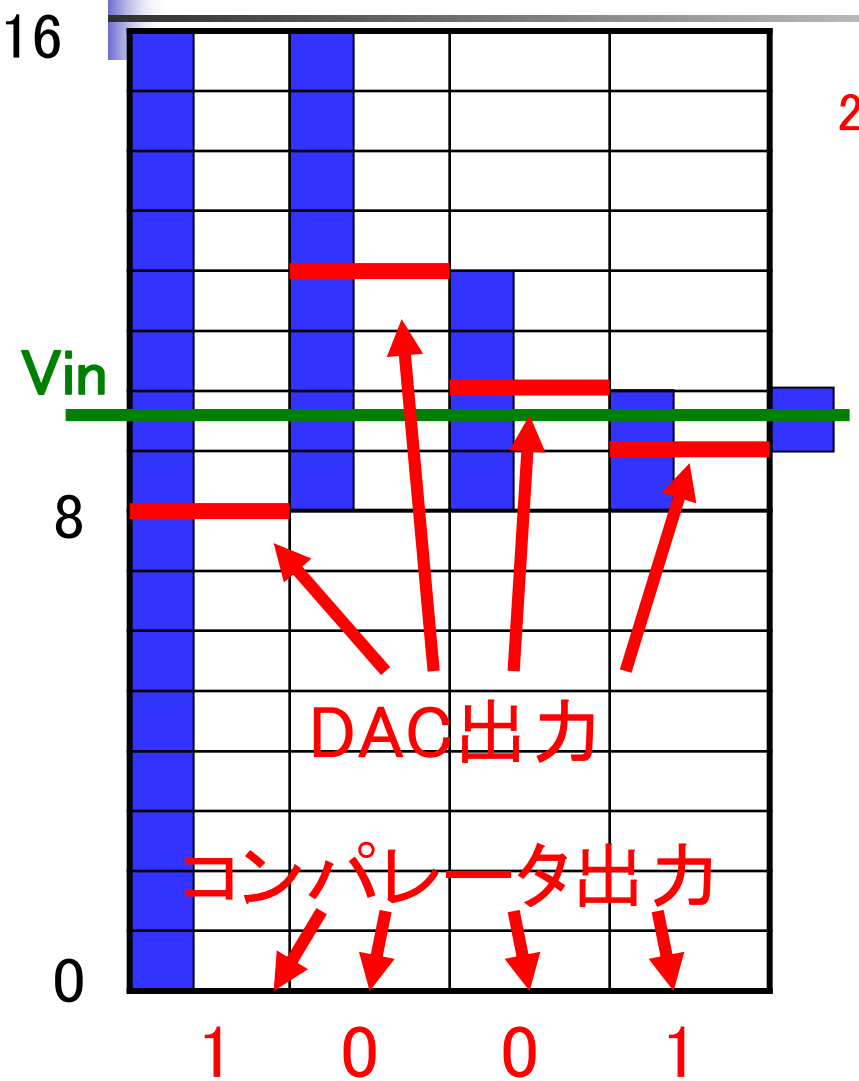
(Successive Approximation Register ADC: SAR ADC)



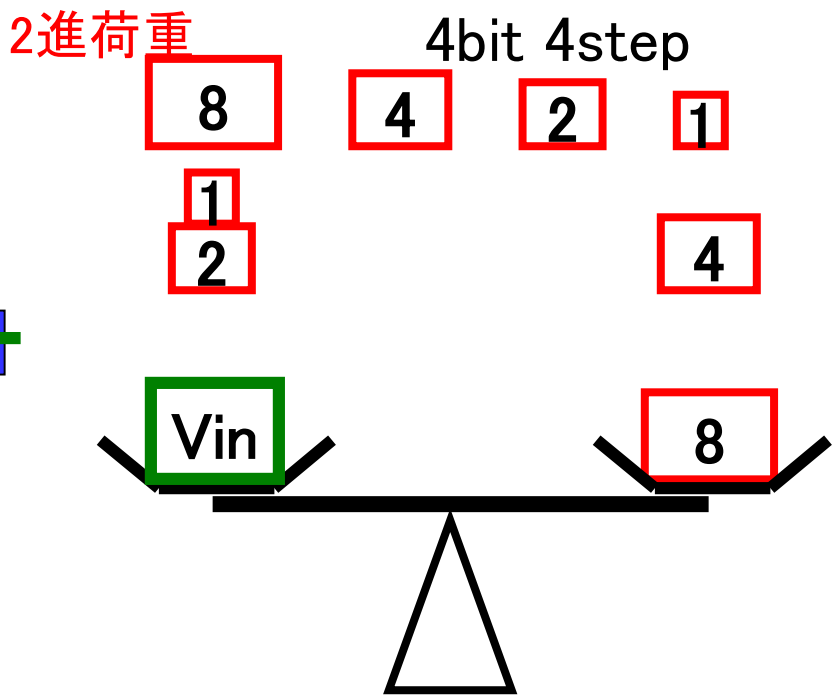
デジタル回路中心，オペアンプ不要.

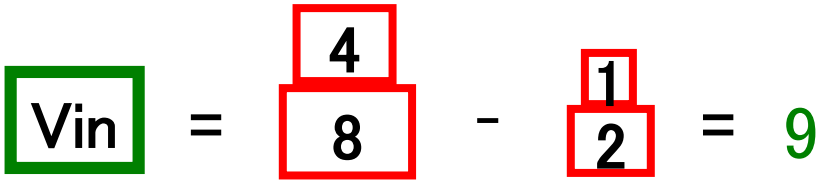
→ 微細CMOSに適している.

2進探索アルゴリズム

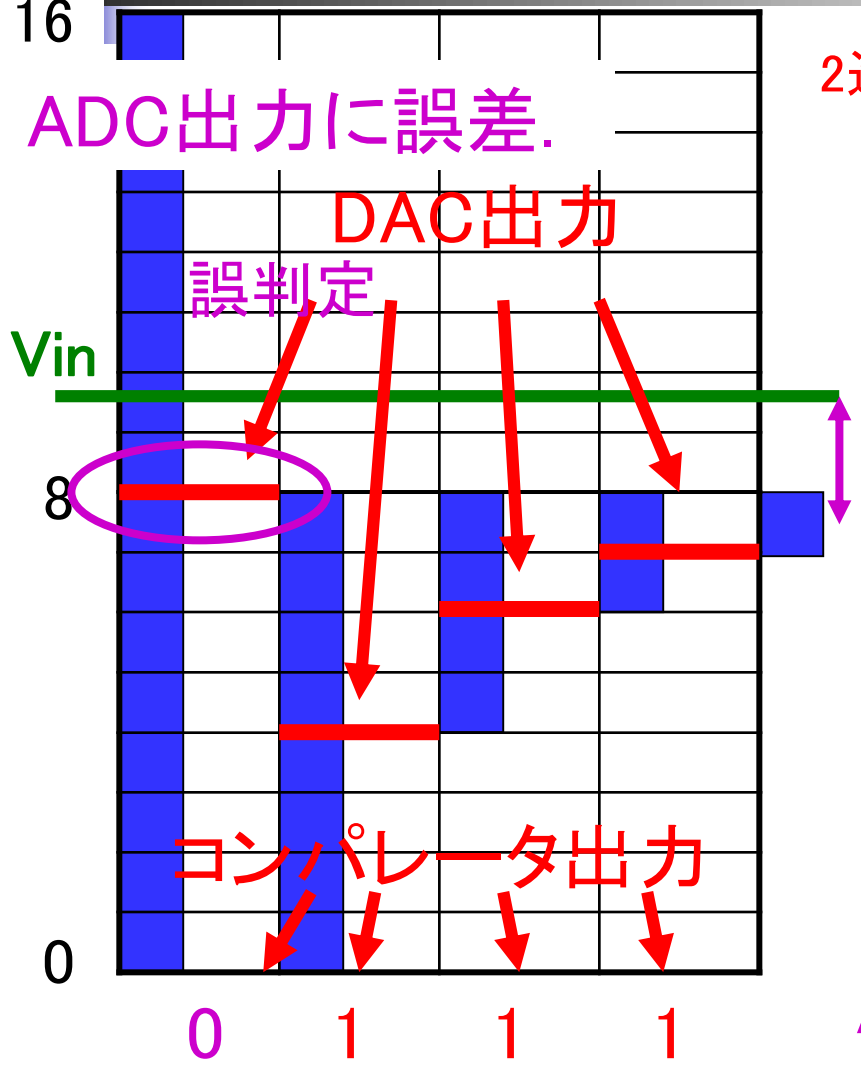


“天秤の原理”

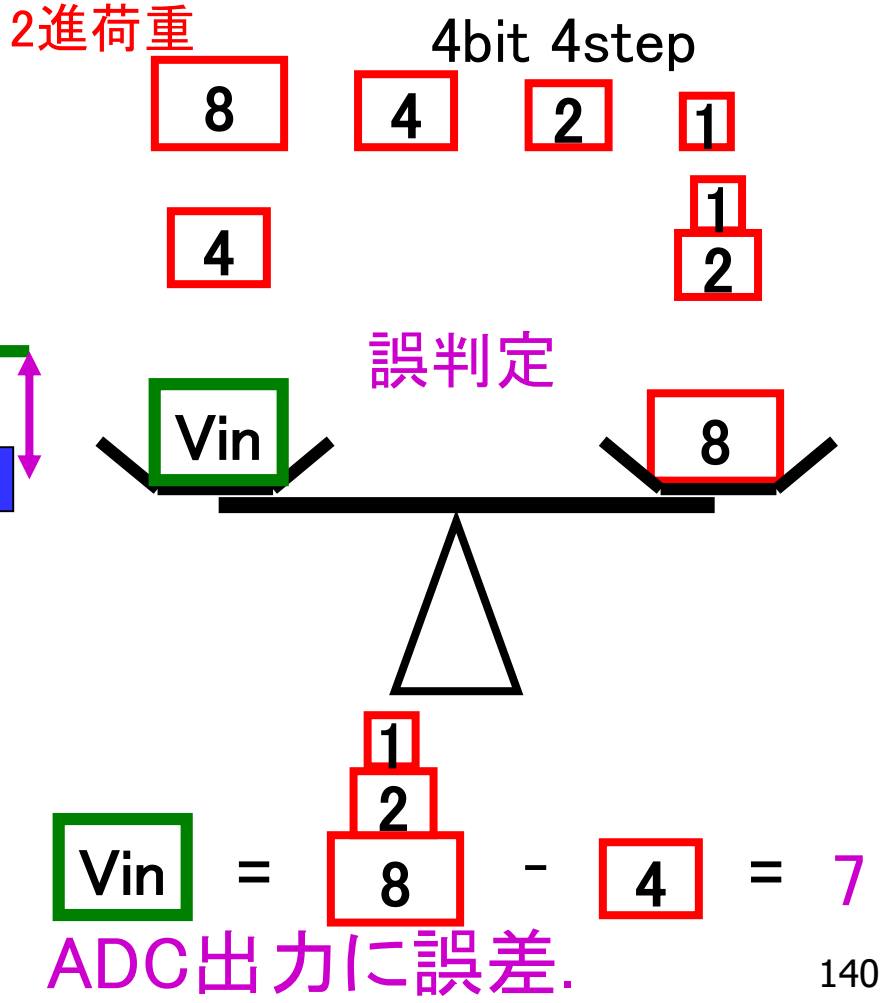




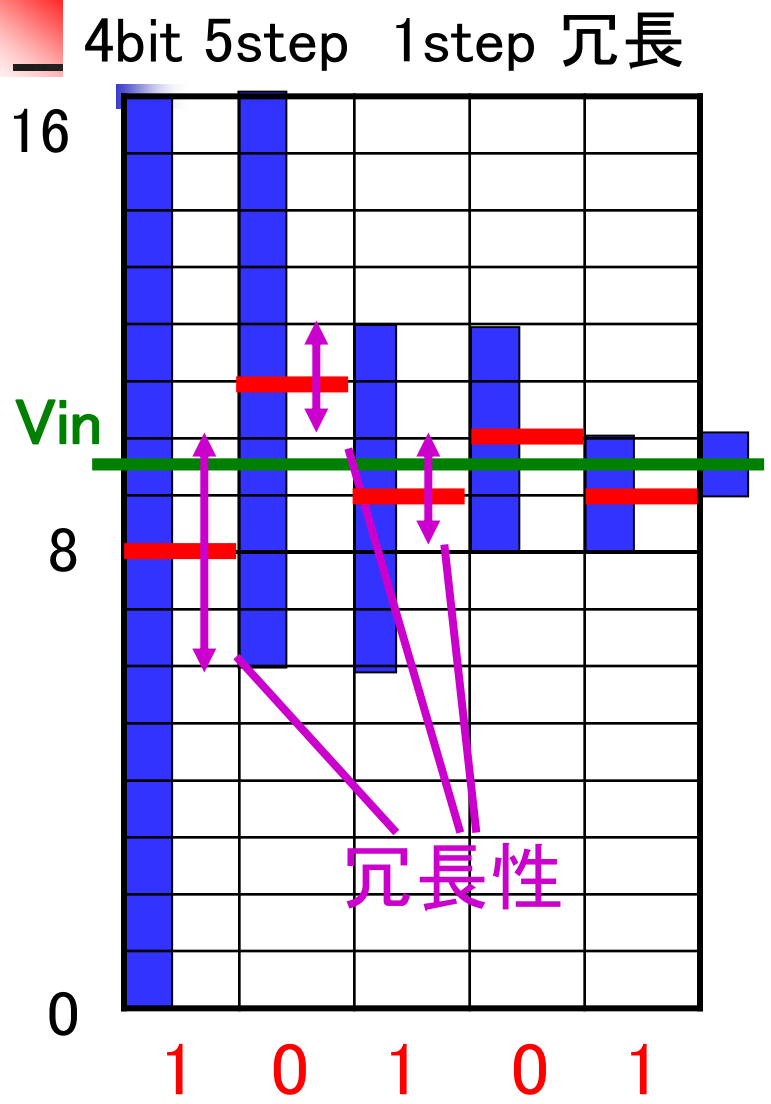
2進探索アルゴリズムの問題点

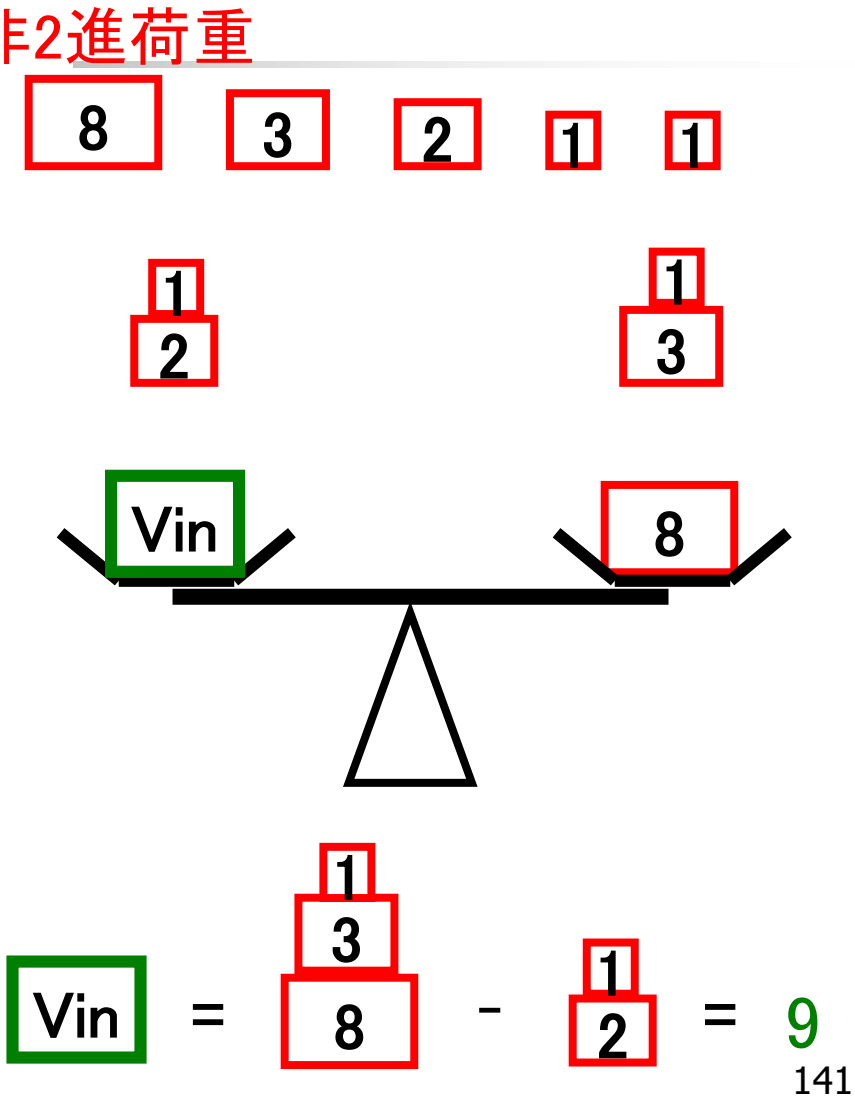


“天秤の原理”

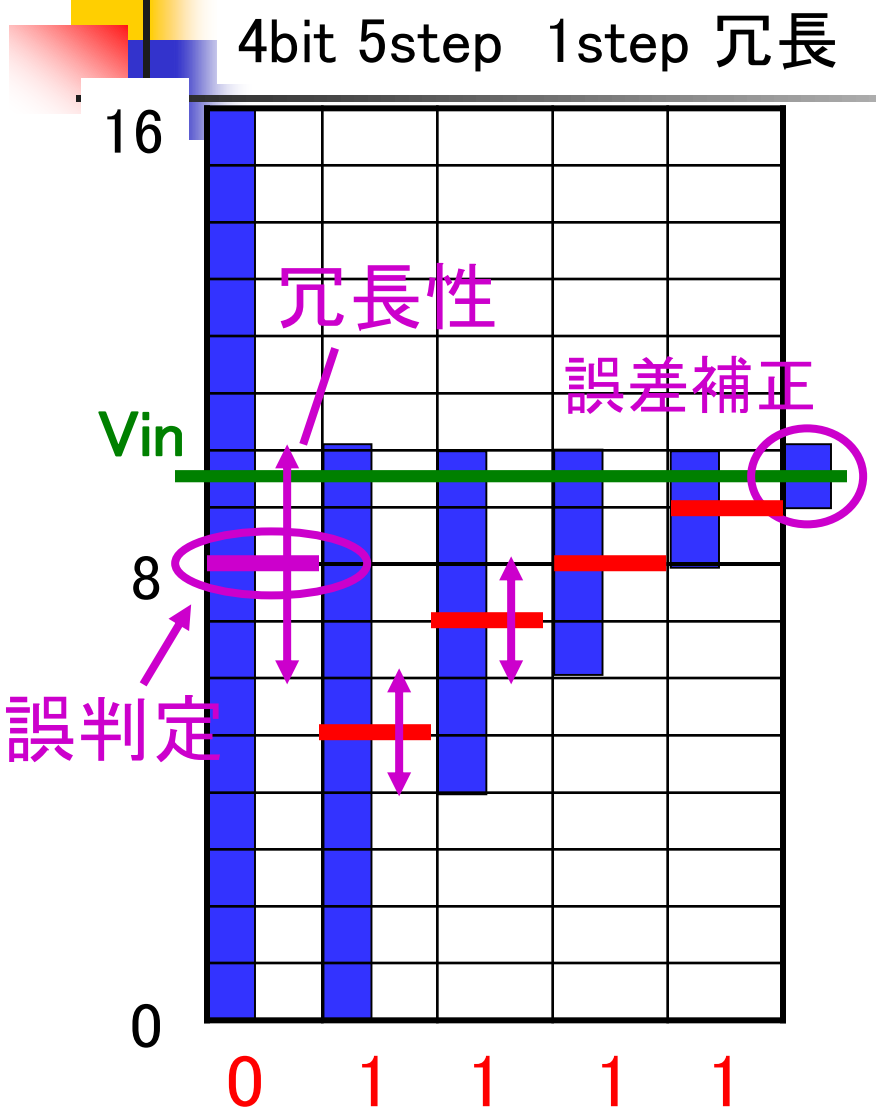


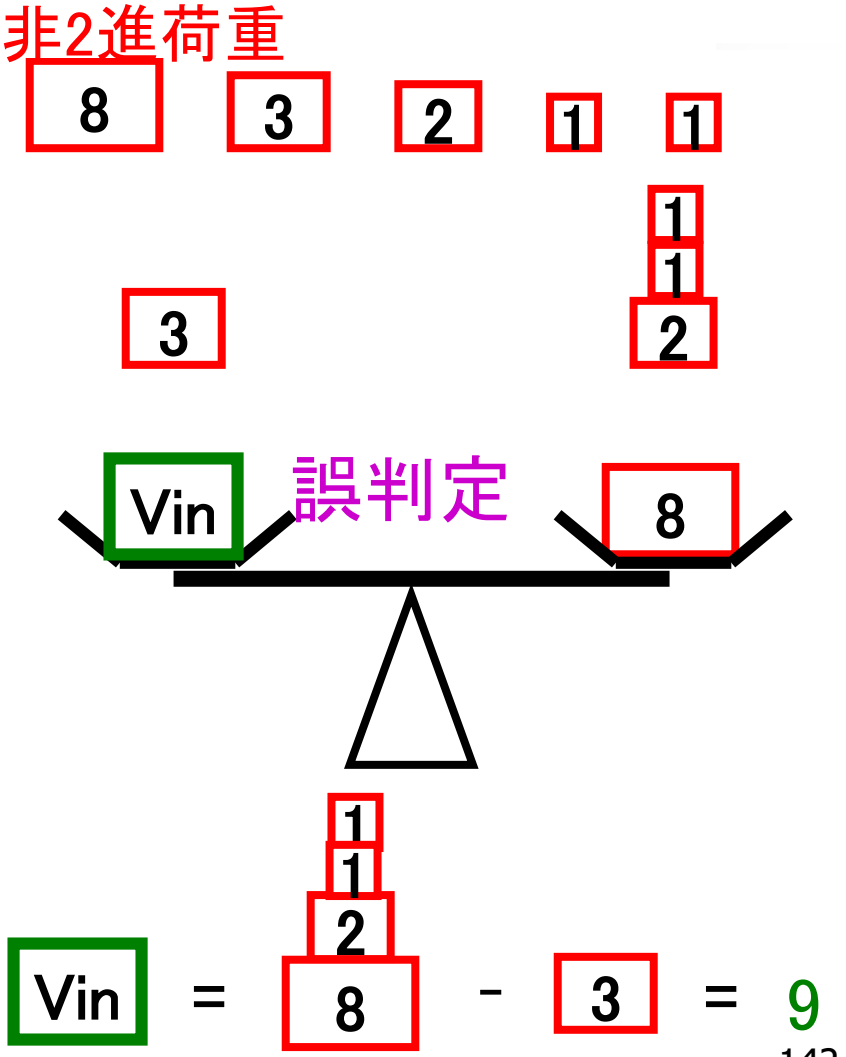
非2進探索アルゴリズム





非2進探索アルゴリズム





誤差補正原理　デジタル出力“9”の場合

2進探索アルゴリズム

誤差補正不可

コンパレータ出力: 1 0 0 1 ← 1パターン

Dout = 8 + 4 − 2 − 1 + 0.5 − 0.5 = 9

非2進探索アルゴリズム

誤差補正可能

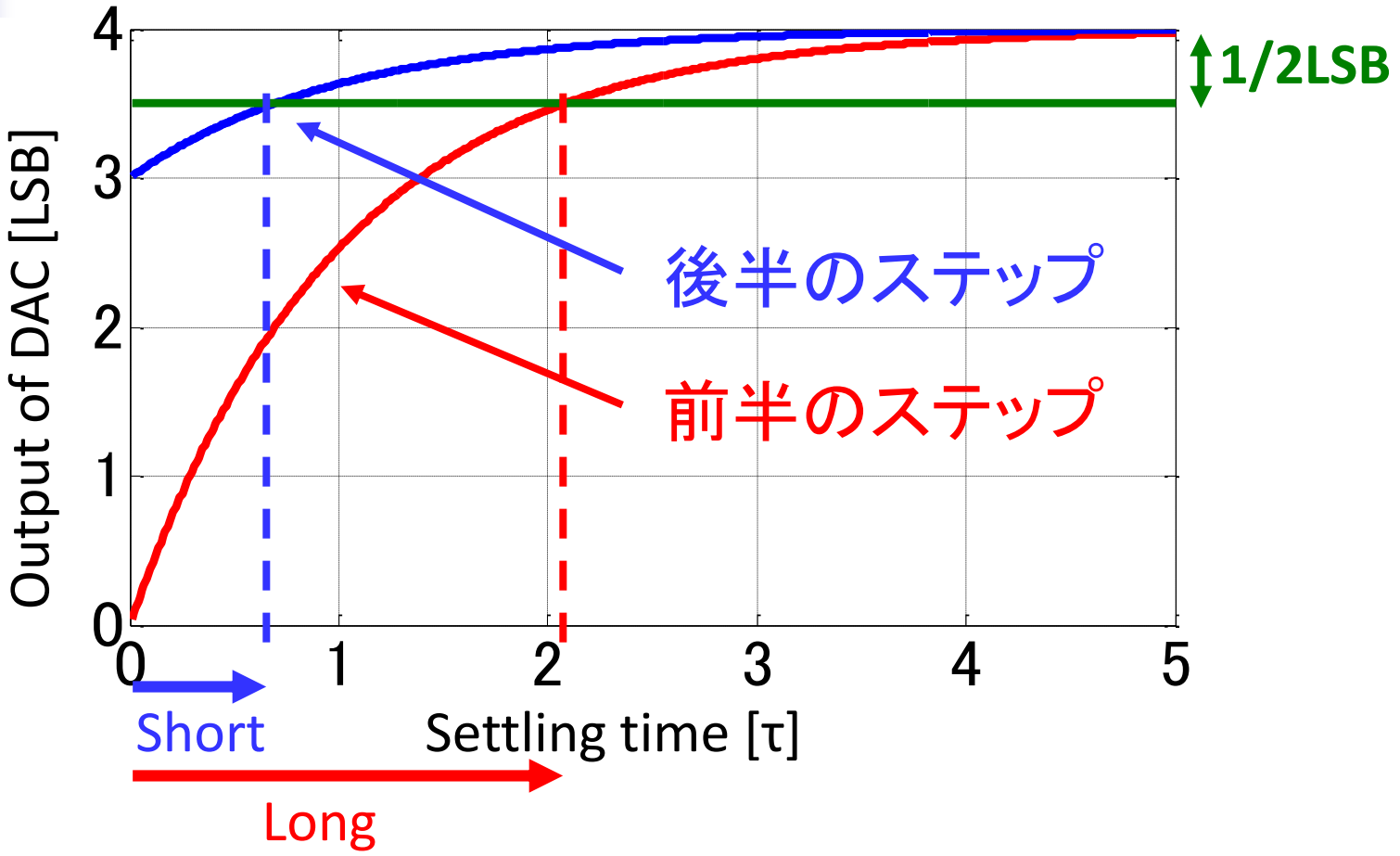
コンパレータ出力: 1 0 1 0 1 ← 複数パターン

Dout = 8 + 3 − 2 + 1 − 1 + 0.5 − 0.5 = 9

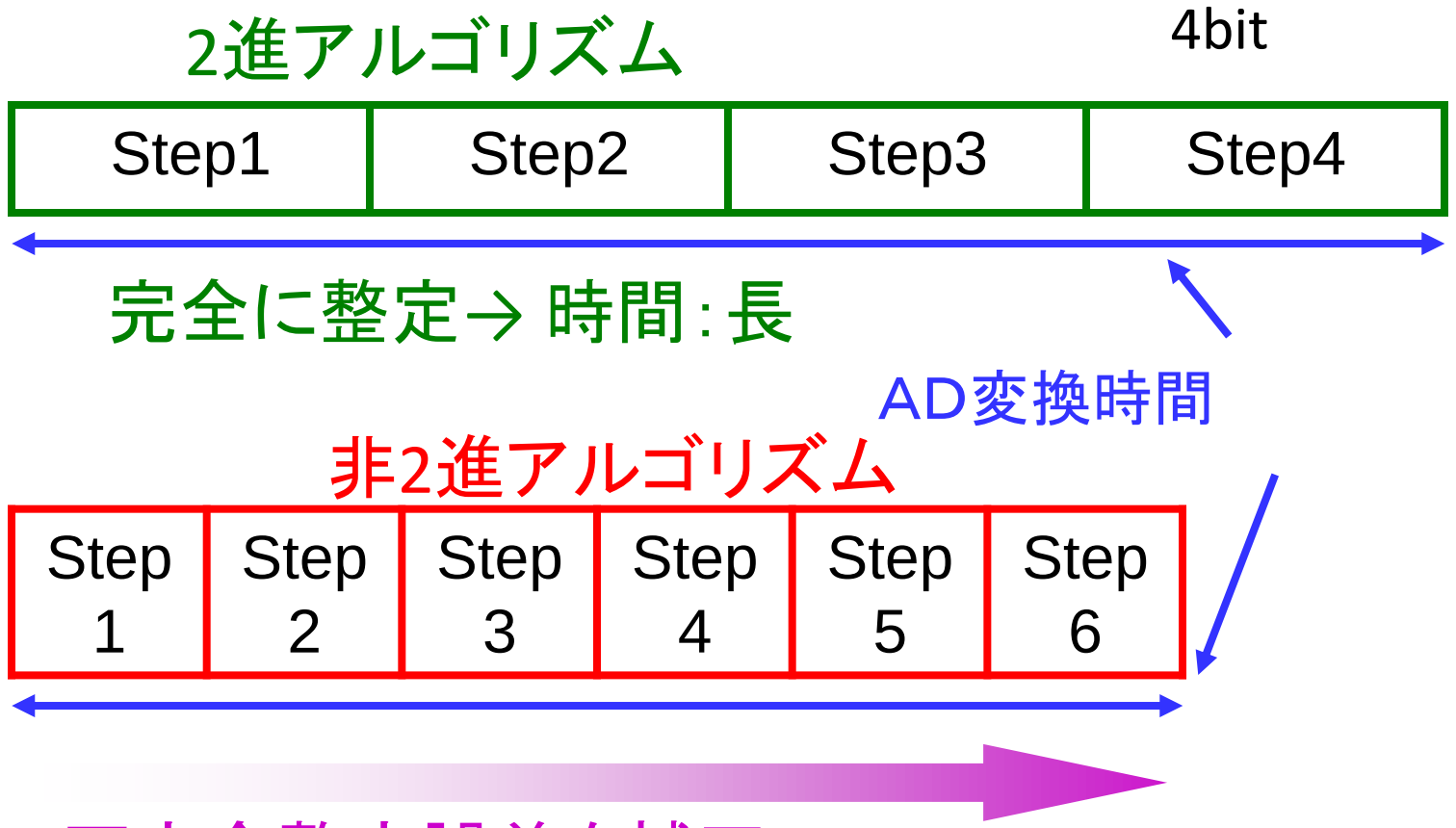
コンパレータ出力: 0 1 1 1 1

Dout = 8 − 3 + 2 + 1 + 1 + 0.5 − 0.5 = 9

DAC出力の整定


1/2LSB
後半のステップ
前半のステップ
Output of DAC [LSB]
Settling time [τ]
Short
Long
0
1
2
3
4
5

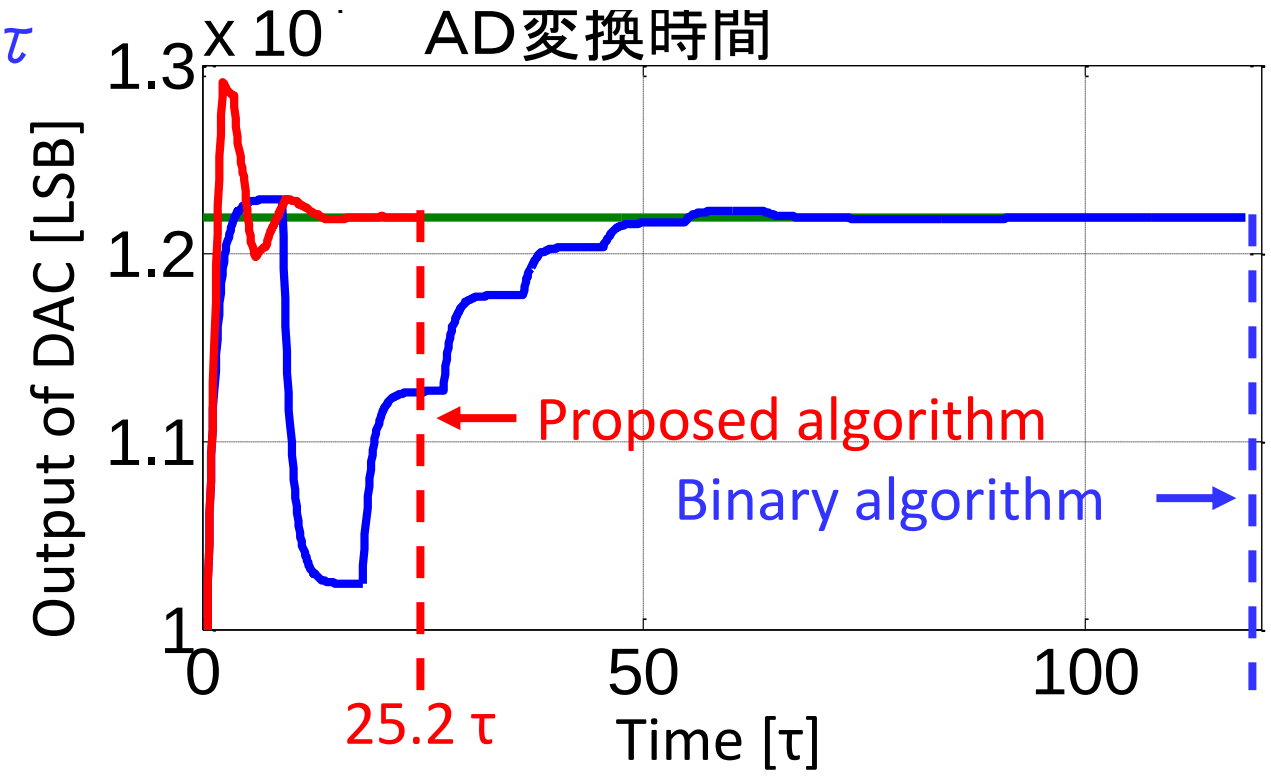
AD変換時間の比較


2進アルゴリズム
4bit
Step1
Step2
Step3
Step4
完全に整定→ 時間：長
AD変換時間
非2進アルゴリズム
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
不完全整定誤差を補正
不完全整定→ 時間：短

変換時間のMATLABシミュレーション

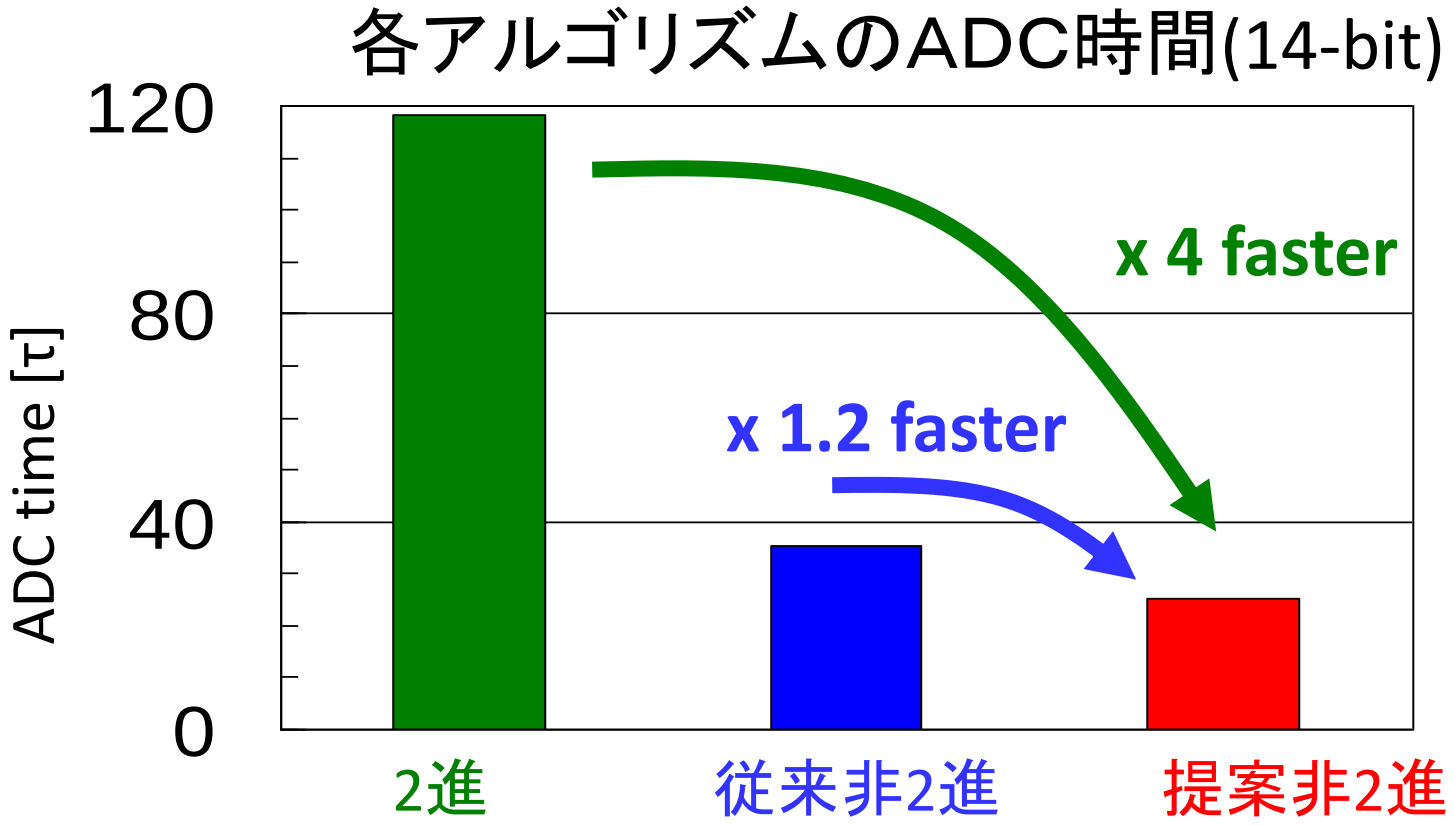
2進アルゴリズム
14-bit,14-step
各ステップでの整定時間: 9.1 τ

提案アルゴリズム
14-bit, 22-step
各ステップでの整定時間: 1.2 τ



整定時間のみを
計算した
シミュレーション

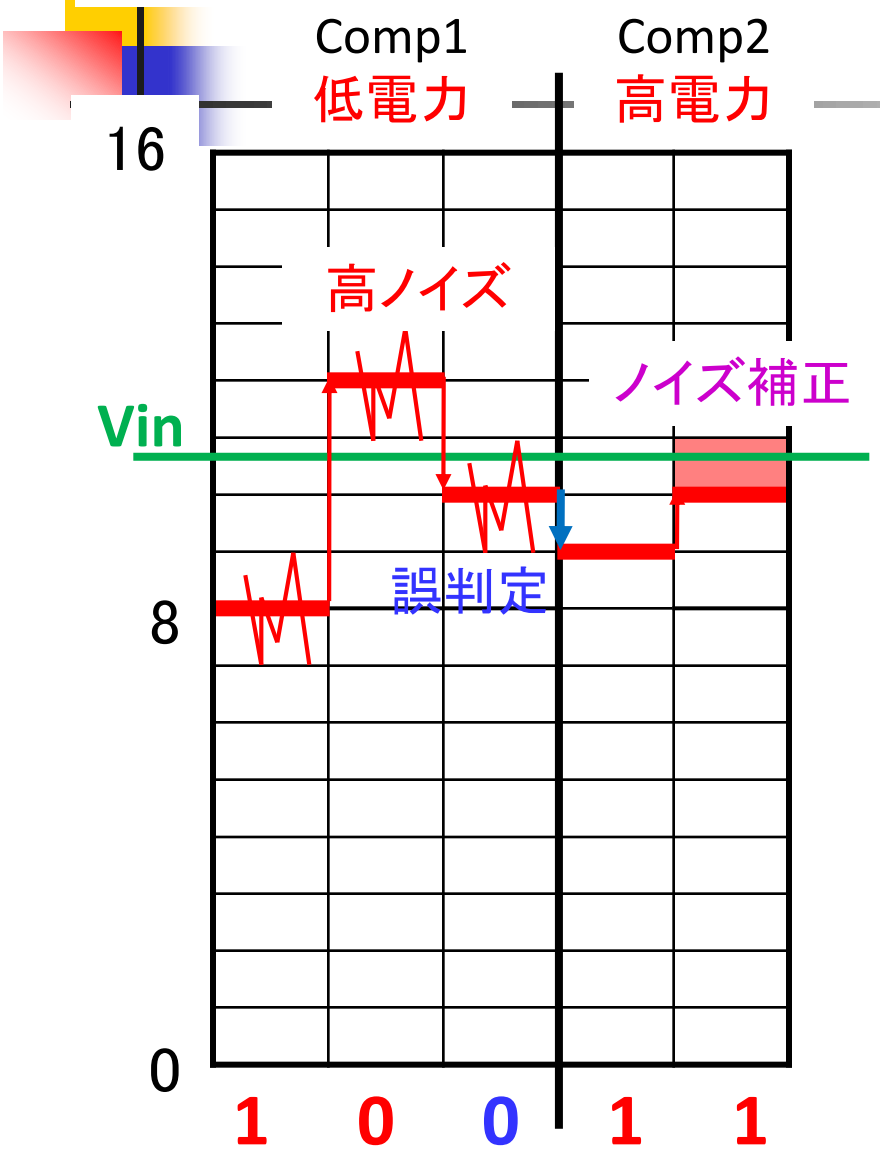
AD変換時間の比較

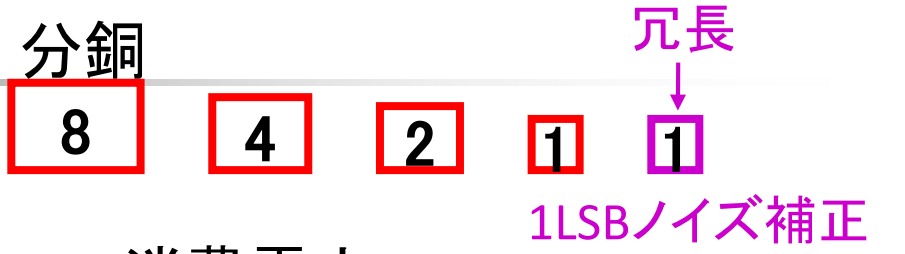


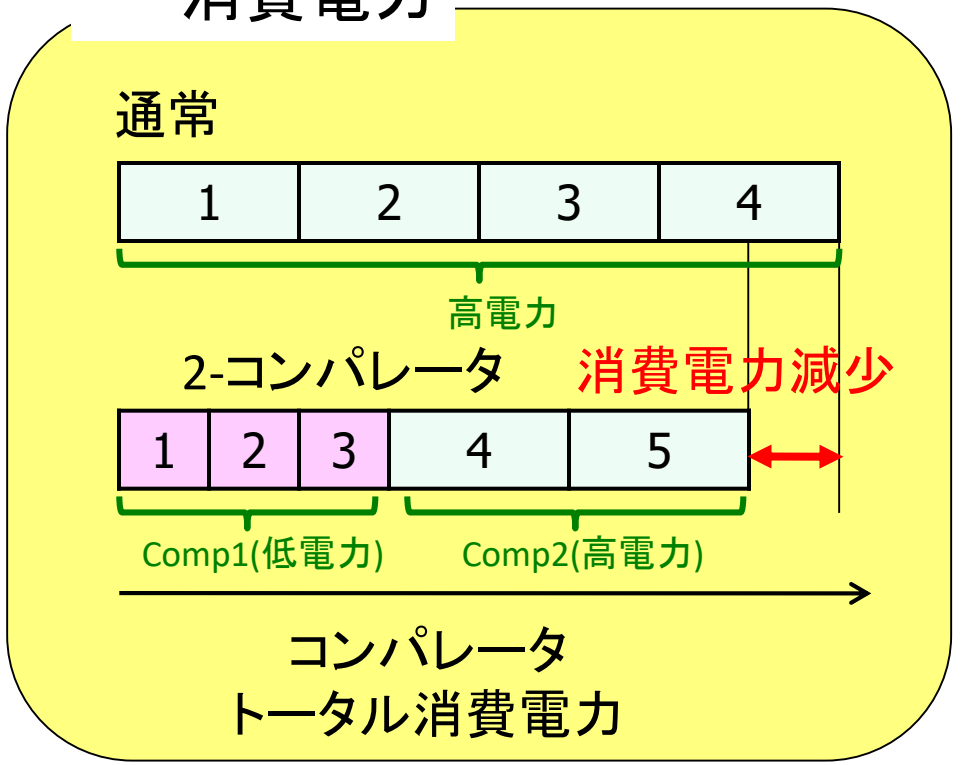
提案アルゴリズム ⟹ ４倍速い

冗長アルゴリズムによる低消費電力化

2つのコンパレータ SAR ADC（IMEC提案）

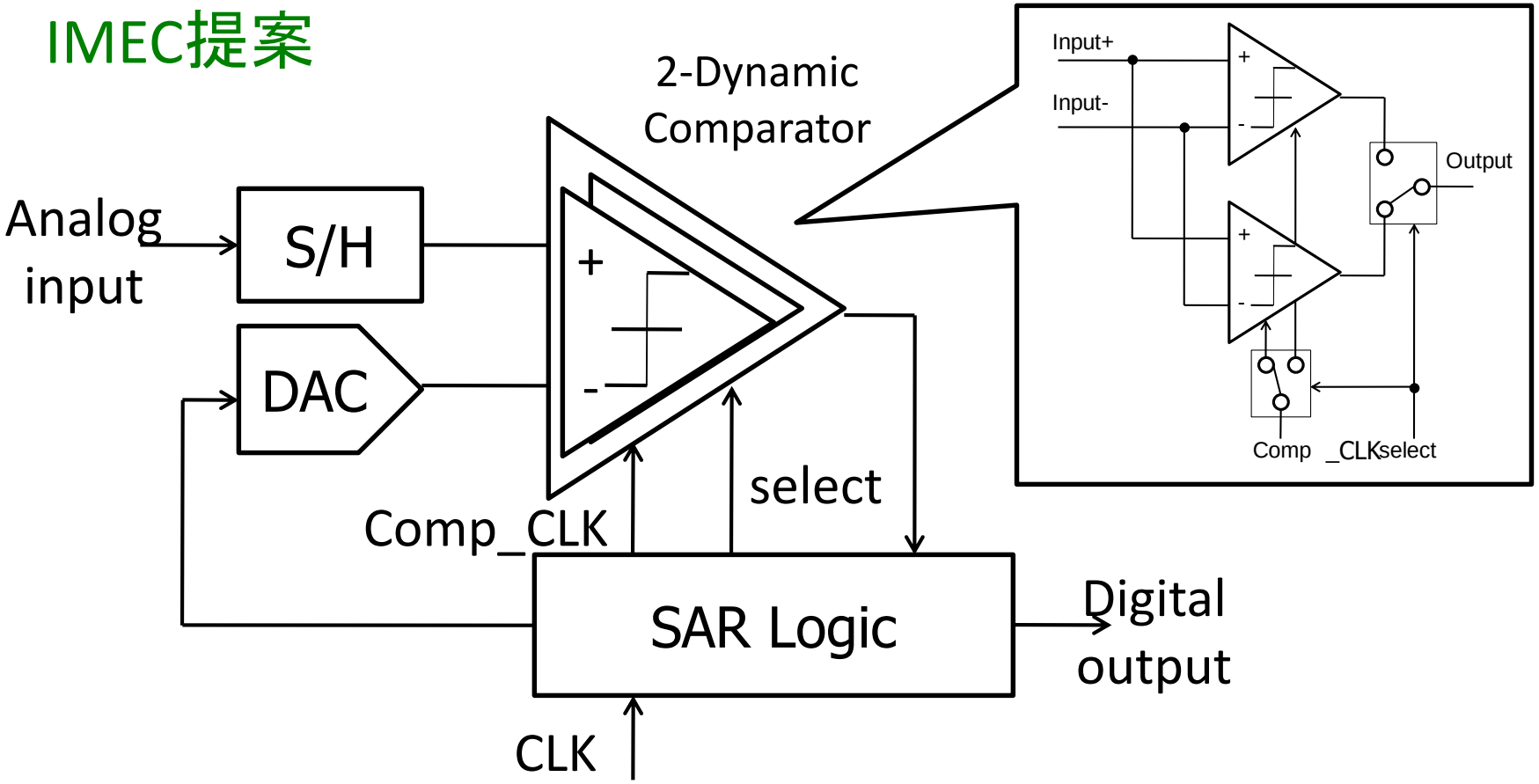






2-コンパレータ SAR ADC構成

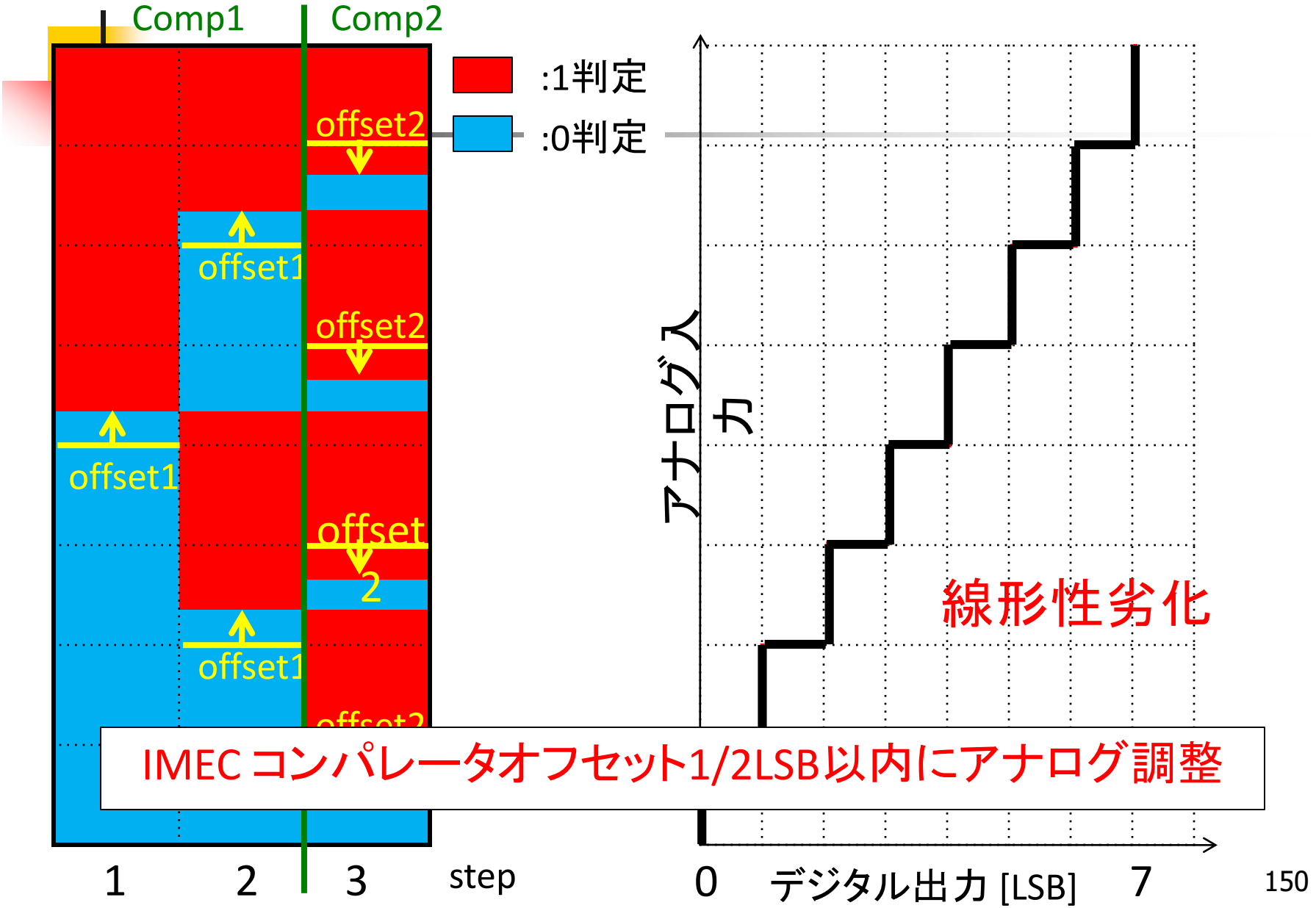
IMEC提案



文献

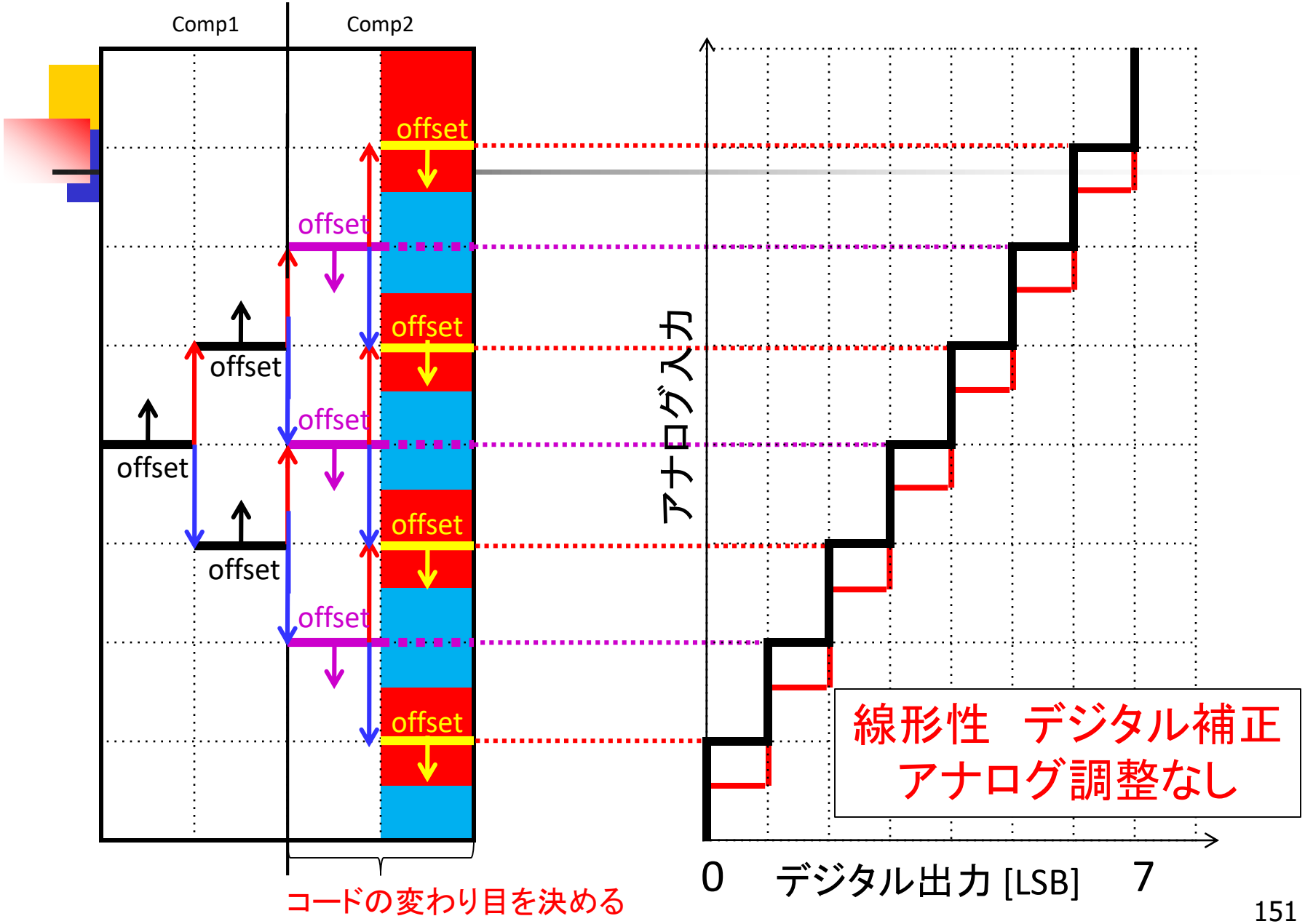
V.Giannini, P.Nuzzo, V.Chironi,A.Baschirotto, G.V.Plas, J.Craninckx
“ An 820 μ W 9b 40MS/s Noise-Tolerant Dynamic-SARADC in 90nm Digital CMOS ” ISSCC (Feb.2008).

2つのコンパレータを用いた技術
コンパレータオフセットミスマッチの影響



IMEC コンパレータオフセット1/2LSB以内にアナログ調整

提案方式　冗長アルゴリズムによるデジタル補正

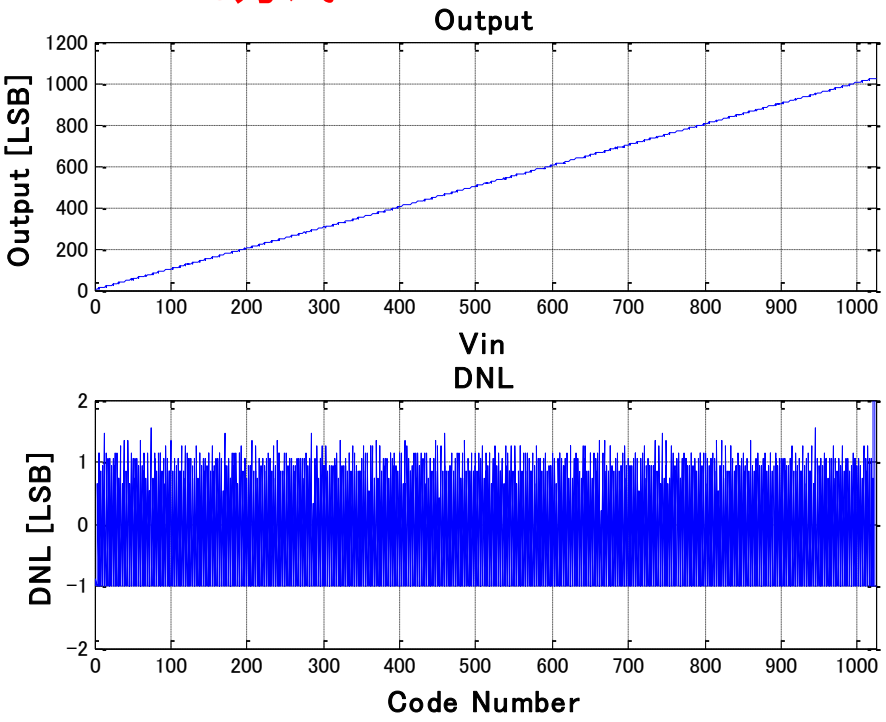


MATLABシミュレーション（ランプ波）

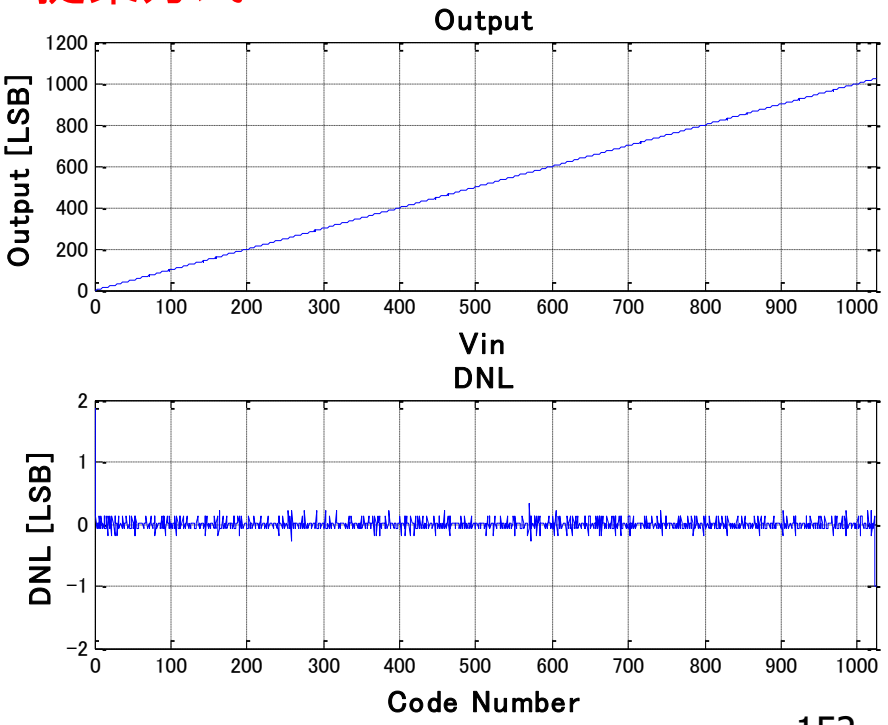
Comp1（低電力） オフセット：＋4.0 LSB、 ノイズ：1.0 LSB
Comp2（高電力） オフセット：－2.0 LSB、 ノイズ：0.2 LSB

コンパレータのアナログ・キャリブレーションなしの場合

IMEC方式



提案方式



消費電力とコンパレータミスマッチ許容のトレードオフ

非2進SAR ADC の特長

2進SAR ADC　　（IMEC: 西洋哲学）

構成，動作　無駄なし　→　効率的

冗長性（無駄）を入れる．

さらに効率が良くなる．

・スピード（DAC不完全整定）

・低消費電力化

冗長性により誤動作を許容

→各構成要素，動作への要求緩和

（「無用の用」老子: 東洋哲学）

154

内　容

- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル　デジタルCMOS回路
- デジタルCMOS回路の性能（消費電力、スピード）
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：　SAR ADC+分散型積和演算回路
- 事例2：　自己校正機能をもったTDC回路
- 事例3：　冗長アルゴリズムを用いたSAR ADC
- 最後に

デジタル技術の発展は産業・社会を変えた

- アナログ: 連続信号 「坂道」
  デジタル: 0, 1 「階段」
- デジタルは　産業的に
  技術のコピーを容易化
  ⇒ キャッチアップ早い
  インターフェースを容易化
  ⇒ エレクトロニクス産業の
  水平分業化 （産業構造が変わる）
- デジタルにより　社会的に
  人は数値で管理されるようになった

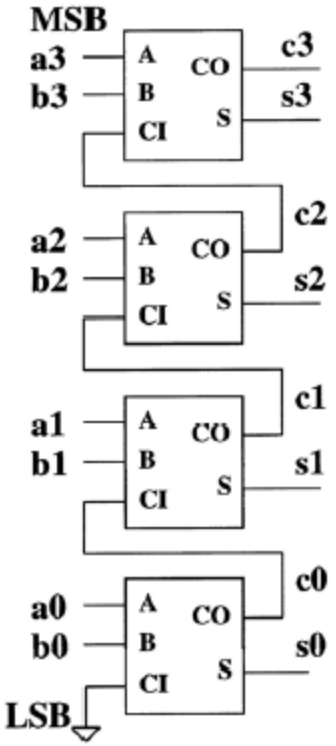
付録

Carry Look Ahead Adderとは

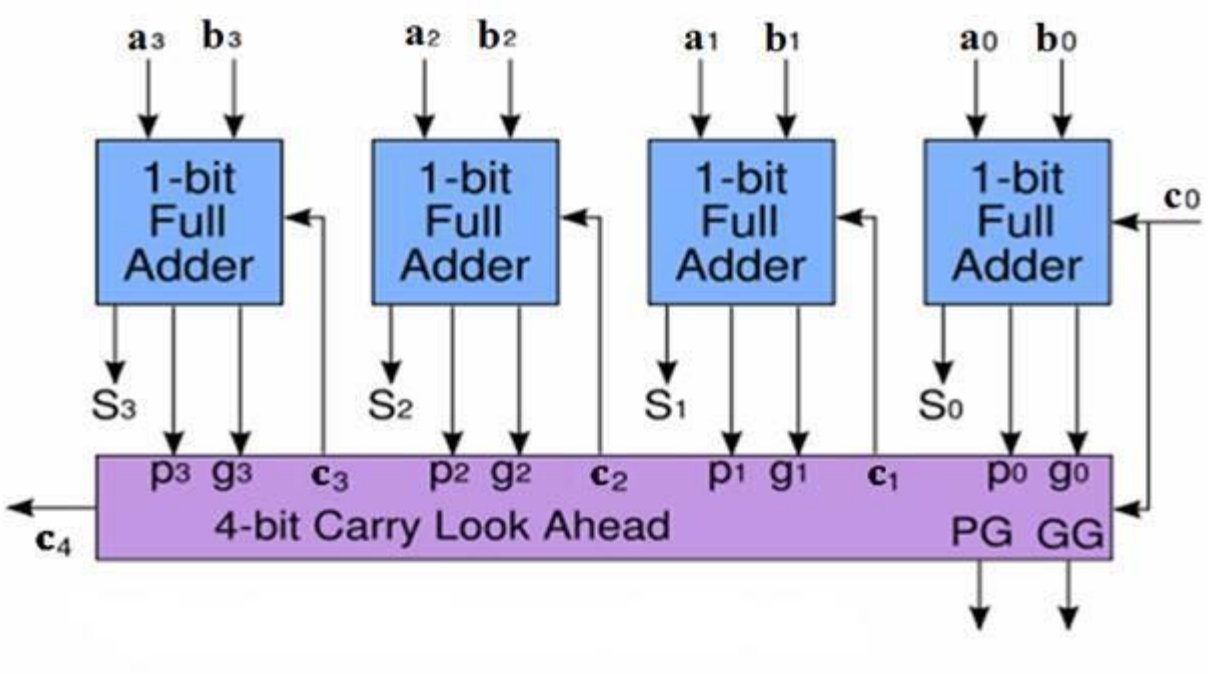
リップルキャリー加算器
下位ビットの桁上げが
次の上位ビットの入力に接続

桁数分のキャリーの伝播が生じるので
全体の加算演算の時間がかかる

桁上げ信号の部分
別に早く計算し高速化



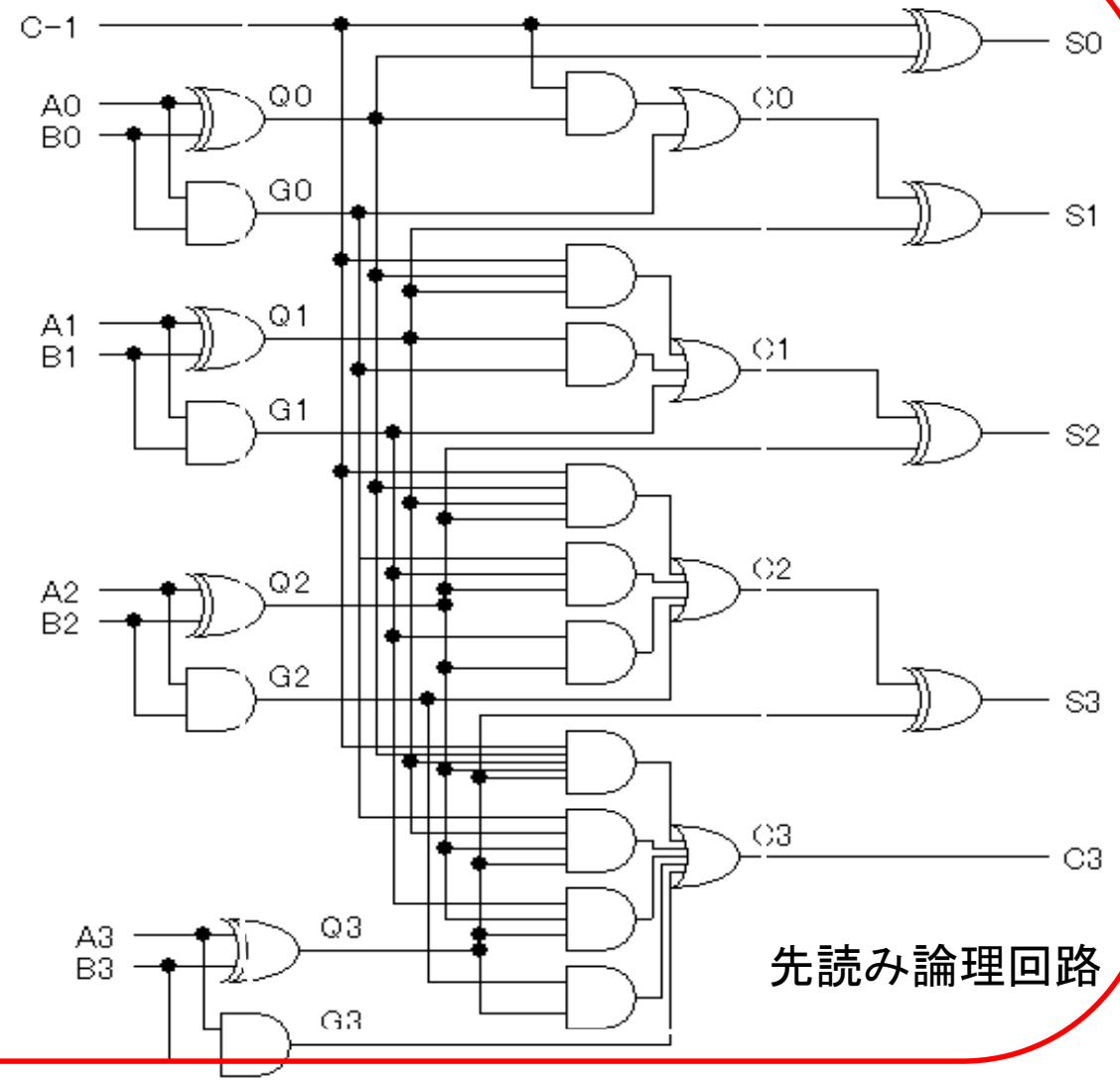
Carry Look Ahead Adderとは



桁上げ(Carry)を別に先に計算する

Carry Look Ahead Adder 回路

4bit Carry Look Ahead Adder



先読み論理回路

Carry Look Ahead 論理式

キャリー生成項 と キャリー伝播項

- Cn:nビット目のキャリー

$$C_n = A_n B_n + (A_n + B_n) \cdot C_{n-1}$$

$$= G_n + P_n \cdot C_{n-1}$$

- Gn:キャリー発生関数

$$G_n = A_n \cdot B_n$$

- Pn:キャリー伝播関数

$$P_n = A_n + B_n$$

Carry Look Ahead 論理式 (続き)

キャリーの計算

$C0 = G0 + P0 \cdot C\text{-}1$

これを、C1に代入

$C1 = G1 + P1 \cdot C0$

$= G1 + P1 \cdot (G0 + P0 \cdot C\text{-}1)$

$= A1 \cdot B1 + (A1+B1)\{A0 \cdot B0 + (A0+B0) \cdot C\text{-}1\}$

C0が決まればすぐにC1が決まる

Carry Look Ahead 論理式 (続き)

C2 = G2 + P2･C1
= G2 + P2･(G1 + P1･C0)
= G2 + P2･(G1 + P1･(G0 + P0･C-1))
= G2 + P2･G1 + P2･P1･G0 + P2･P1･P0･C-1

C2もC0が決まればすぐに決まる

C3 = G3 + P3･C2
= G3 + P3･(G2 + P2･(G1 + P1･(G0 + P0･C-1)))
= G3 + P3･G2 + P3･P2･G1 + P3･P2･P1･G0 + P3･P2･P1･P0･C-1