

MWE基礎講座

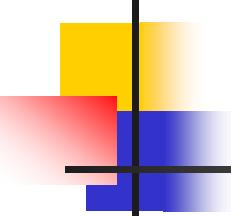
2010年12月2日

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

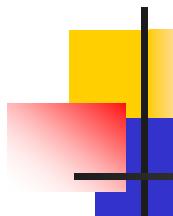
小林春夫

k_haruo@el.gunma-u.ac.jp



内 容

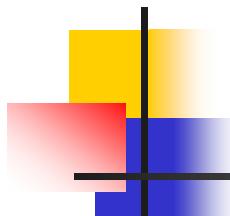
- セミナーの目的・目標
- デジタル回路の基礎
- スイッチレベル デジタルCMOS回路
- デジタルCMOS回路の性能(消費電力、スピード)
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1： SAR ADC+分散型積和演算回路
- 事例2： 自己校正機能をもったTDC回路
- 事例3： 冗長アルゴリズムを用いたSAR ADC
- 最後に



内 容

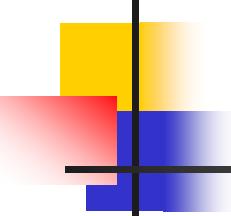
● セミナーの目的・目標

- 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



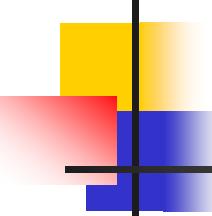
セミナーの目的・目標

- 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」

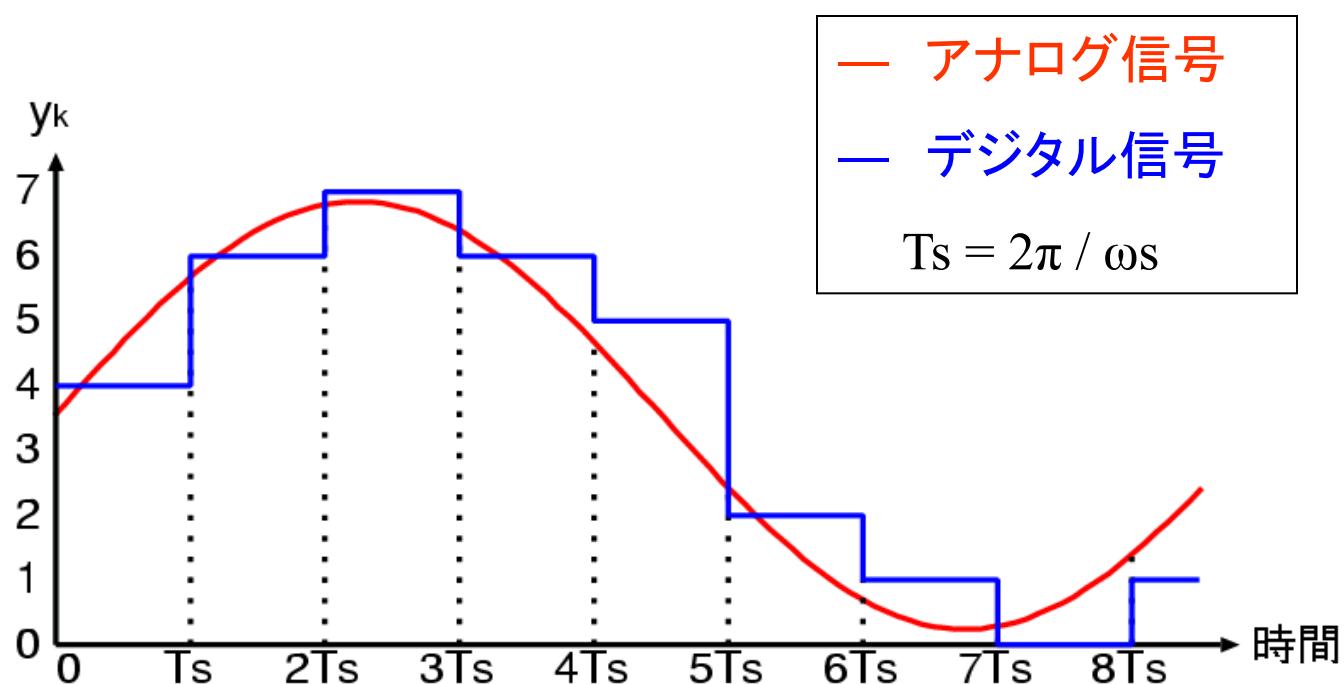
デジタル信号

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」

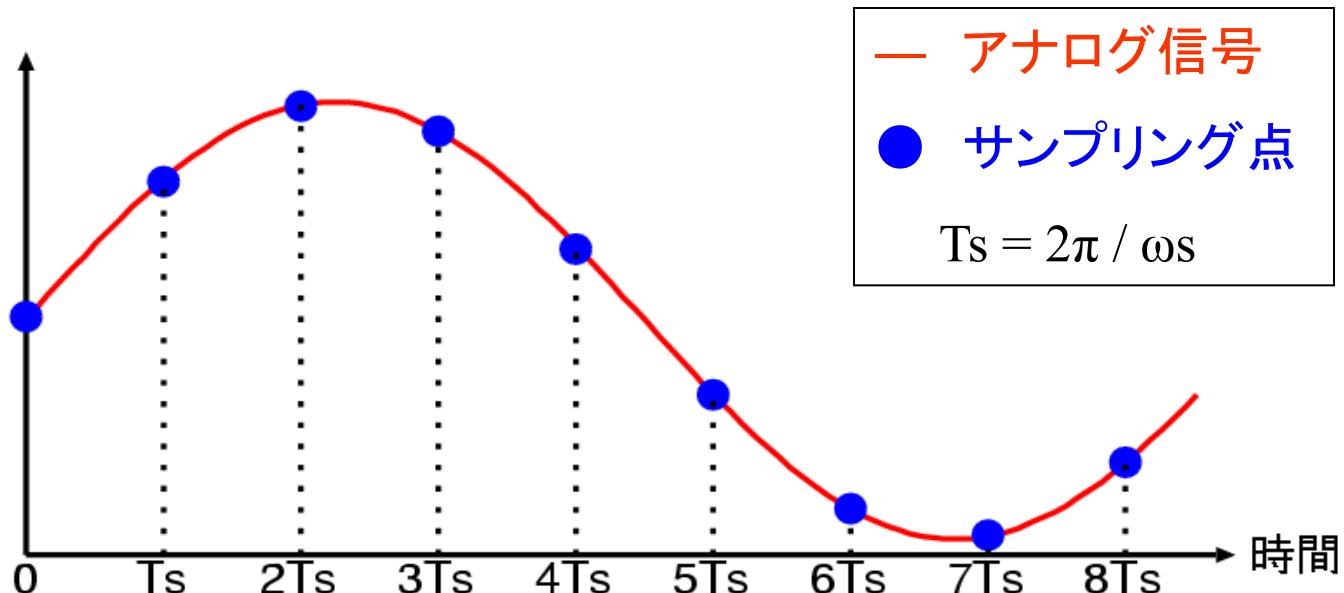
デジタル信号の特徴(1)

空間の量子化（信号レベルの数値化）

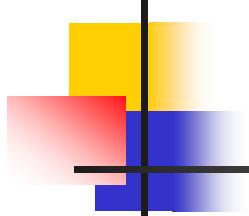


デジタル信号はアナログ信号レベルを
四捨五入(または切り捨て)

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

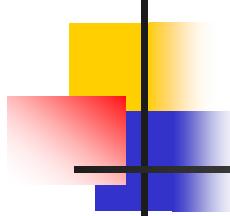
- 人間はなぜ10進数を使うか？
→ 手の指が10本あるから。
- デジタルではなぜ2進数を使うか？
→ 2つの状態は技術的に容易かつ安定して実現可能。

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 | 例 | 2進数 | 1011 | を |
|-----|------|-----|------|---|-----|------|---|
| 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
 $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 = 11$



2進数

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

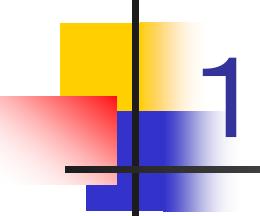
$$2^7 = 128$$

$$2^8 = 256$$

$$2^9 = 512$$

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

(参考 $1,000 = 1\text{k}$)



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

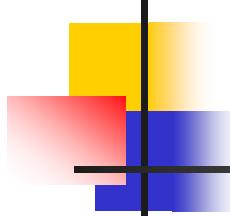
- 人間はなぜ10進数を使うか？

→ 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

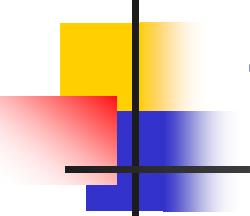
$$3 \times 8 \times 8 \times 8 + 7 \times 8 \times 8 + 2 \times 8 + 4$$

計算が必要

●2進に変換

011 111 010 100

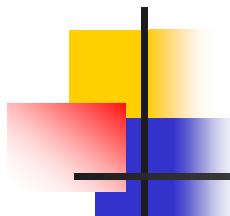
左表から機械的に得られる



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



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 | b ₃ b ₂ b ₁ b ₀ | ud | sd | b ₃ b ₂ b ₁ b ₀ |
| 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 |

負の数の表現 (2の補数)

+5 (2進表現 0101)から -5 の2進表現を得る

0101 のビット反転
1010 を得る。

それに 1 を加える

$$\begin{array}{r} 1010 \\ +) 0001 \\ \hline 1011 \end{array}$$

-5 の2進表現

2の補数表現から10進数へ

符号無 $ud = b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2 + b_0$

符号付 $sd = -b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2 + b_0$

Example: 2進表現 1 0 1 1

符号無 11

符号付 -5



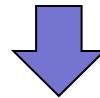
$$ud = 8 + 2 + 1 = 11$$

$$sd = -8 + 2 + 1 = -5$$

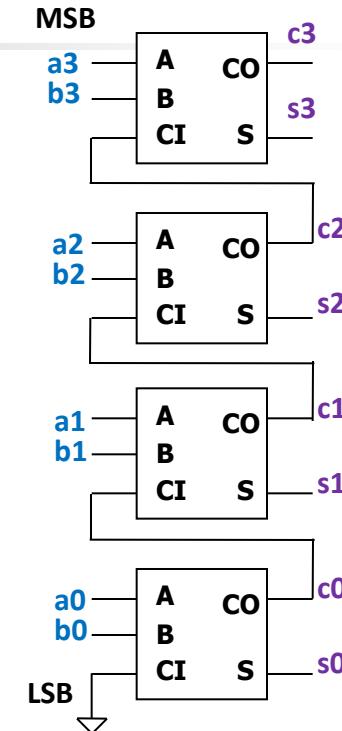
なぜ2の補数表現を用いるのか

2進演算 符号無 符号付

$$\begin{array}{r} 0011 \\ + 1010 \\ \hline 1101 \end{array} \quad \begin{array}{r} 3 \\ + 10 \\ \hline 13 \end{array} \quad \begin{array}{r} 3 \\ + -6 \\ \hline -3 \end{array}$$



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



$$(a_3 \ a_2 \ a_1 \ a_0) = (0 \ 0 \ 1 \ 1)$$
$$(b_3 \ b_2 \ b_1 \ b_0) = (1 \ 0 \ 1 \ 0)$$

$$(s_3 \ s_2 \ s_1 \ s_0) = (1 \ 1 \ 0 \ 1)$$
$$(c_3 \ c_2 \ c_1 \ c_0) = (0 \ 0 \ 1 \ 0)$$

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 符号付 2進

ud sd 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

符号無

ud b₄ b₃ b₂ b₁ b₀

8 01000

9 01001

10 01010

11 01011

12 01100

13 01101

14 01110

15 01111

符号付

sd b₄ b₃ b₂ b₁ b₀

-8 11000

-7 11001

-6 11010

-5 11011

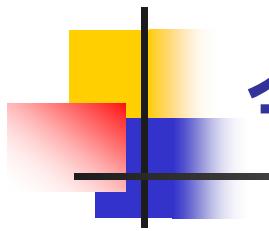
-4 11100

-3 11101

-2 11110

-1 11111

異なる !!



今から320年前、1692年のパリ

哲学者、數学者、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」

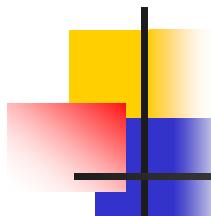
と語る。

(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。





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

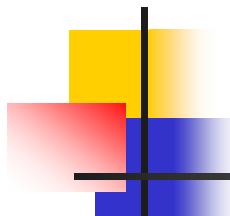
George Boole (1815-1864、英)

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」

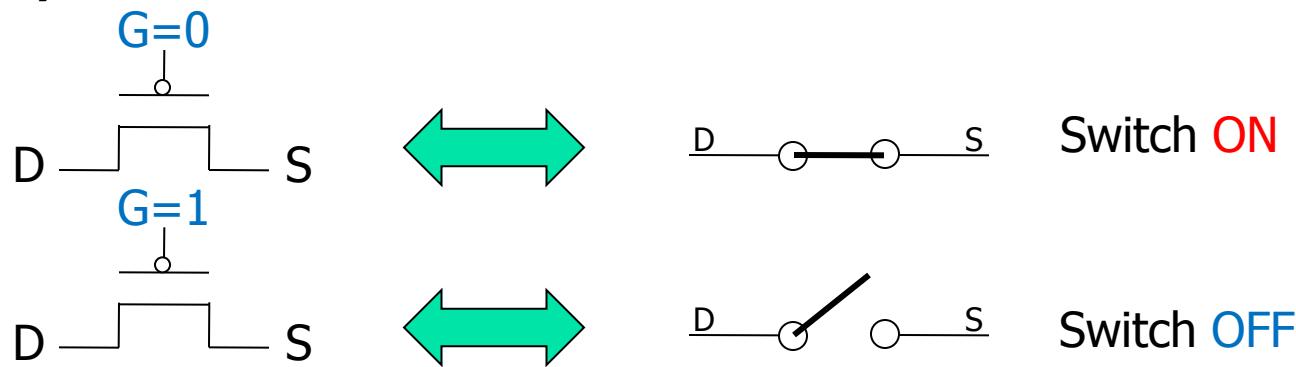


内 容

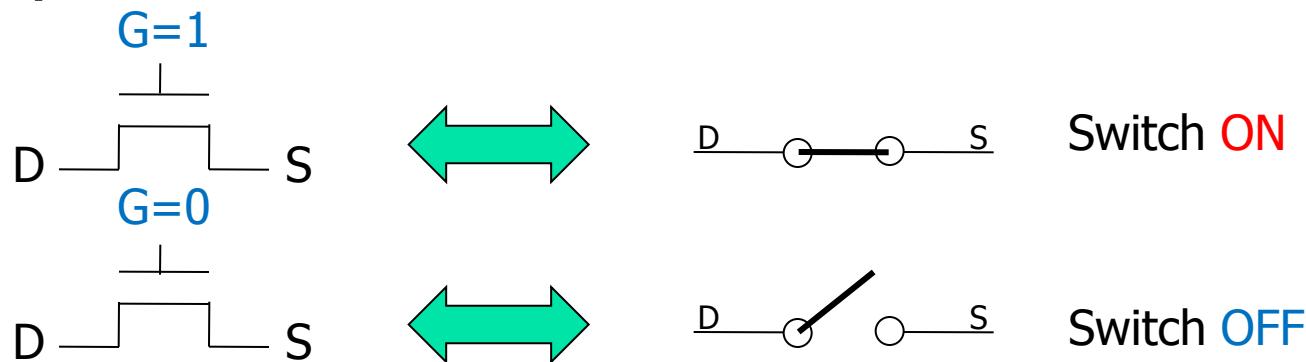
- セミナーの目的・目標
- デジタル回路の基礎
- **スイッチレベル デジタルCMOS回路**
- デジタルCMOS回路の性能(消費電力、スピード)
- 同期回路設計とカウンタ回路
- 加算器、ビットシフト、乗算器
- 事例1：SAR ADC+分散型積和演算回路
- 事例2：自己校正機能をもったTDC回路
- 事例3：冗長アルゴリズムを用いたSAR ADC
- 最後に

PMOS, NMOS スイッチ

(1) PMOS

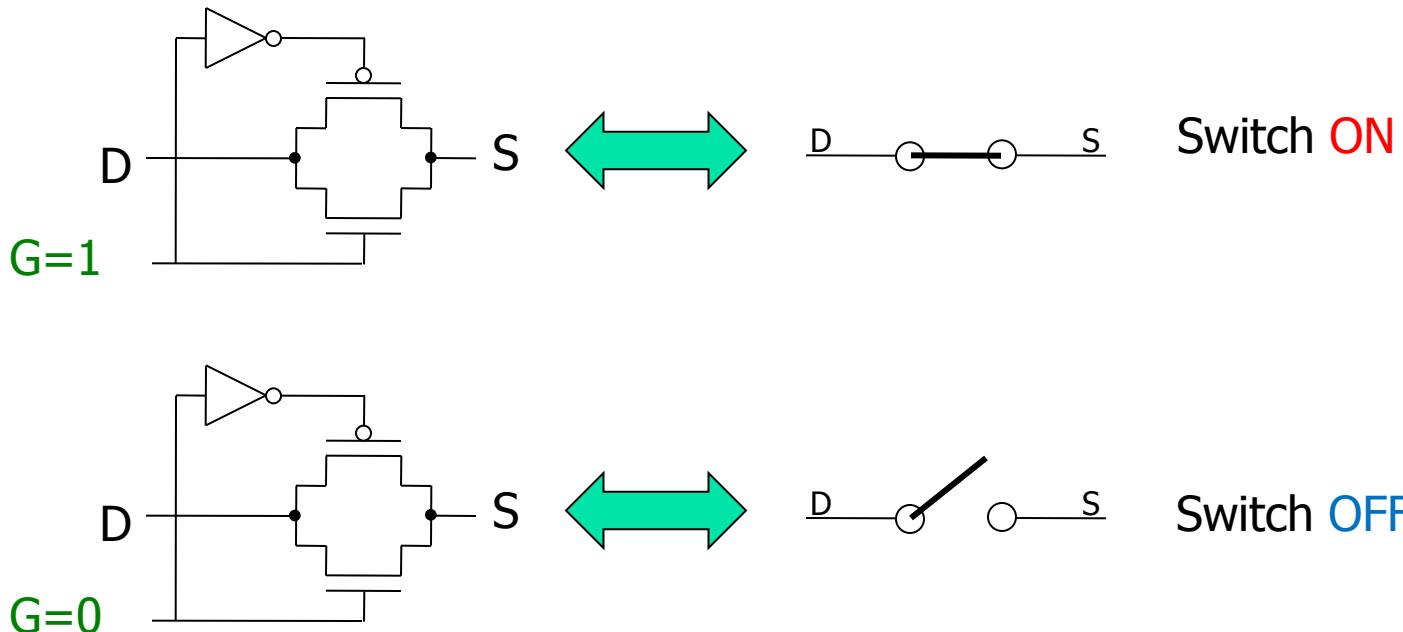


(2) NMOS



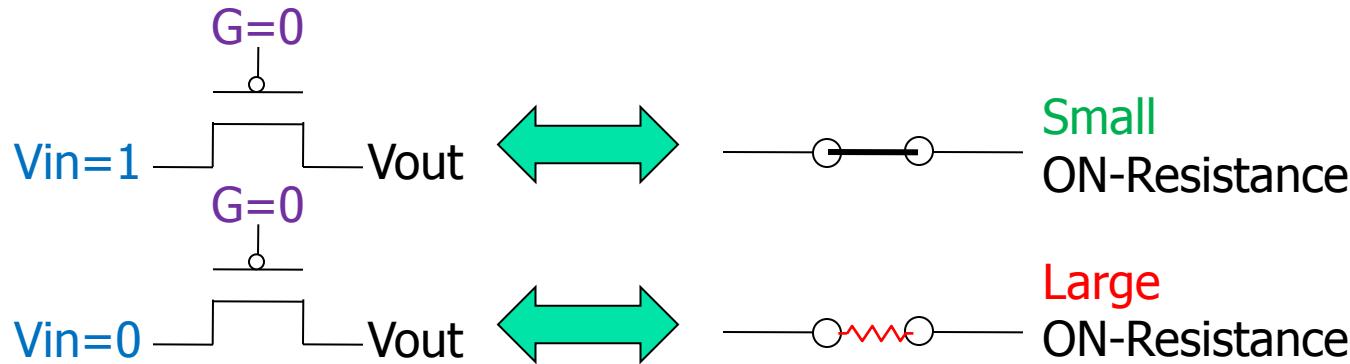
CMOSスイッチ

(3) CMOS



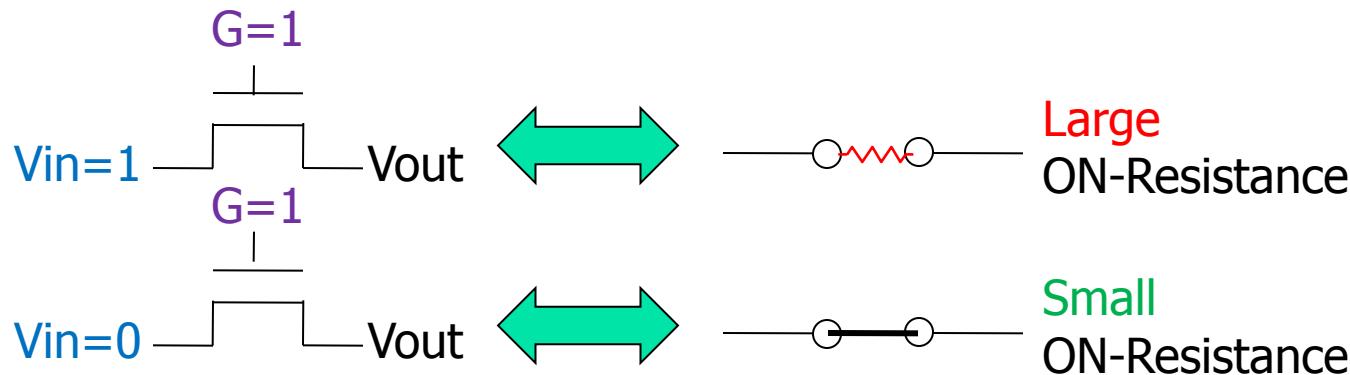
PMOS, NMOSスイッチの オン抵抗

(1) PMOS



PMOSは
正電源側で
用いる

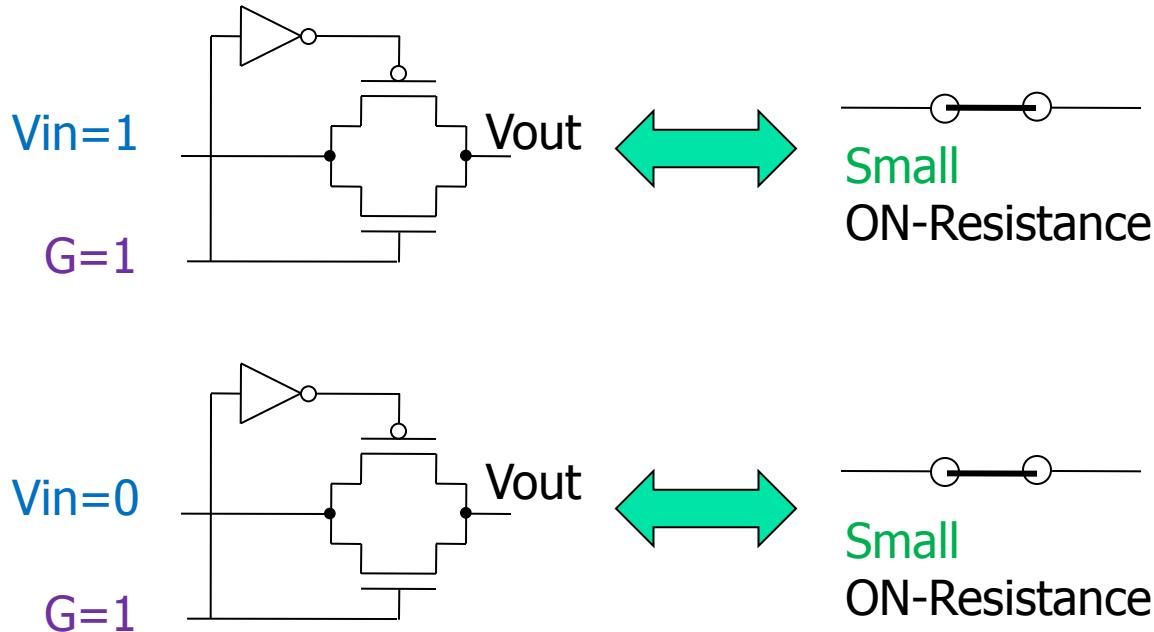
(2) NMOS



NMOSは
GND側で
用いる

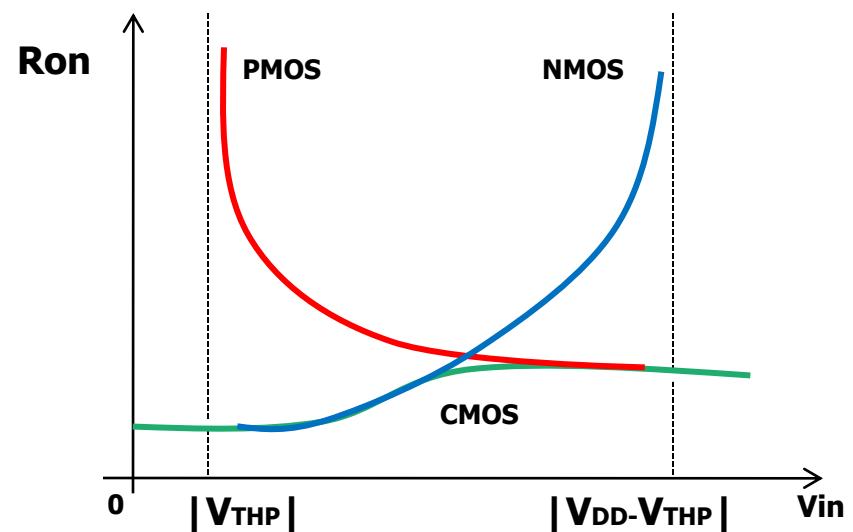
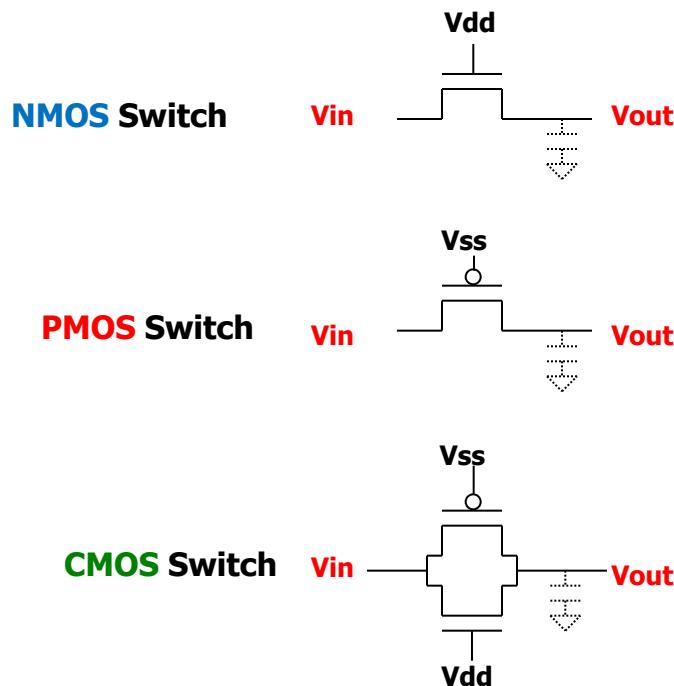
CMOSスイッチのオン抵抗

(3) CMOS



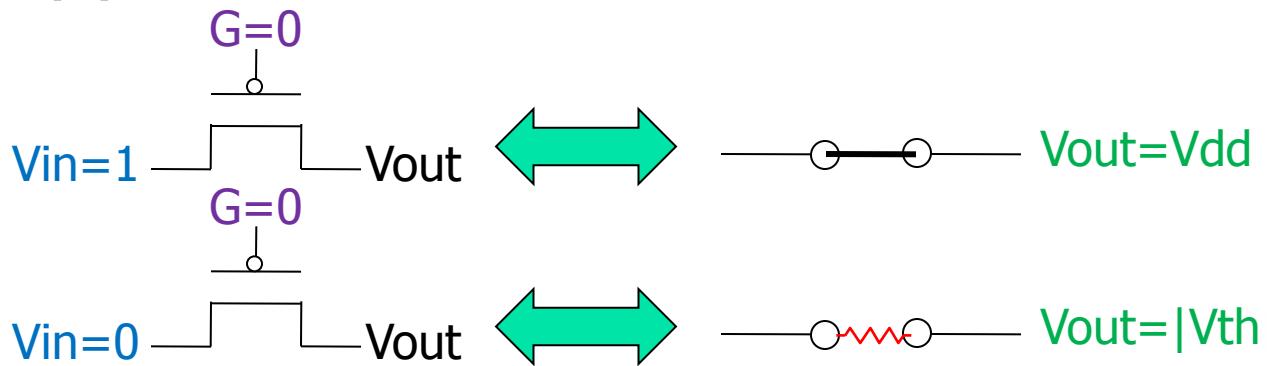
CMOSは
GND側でも
正電源側でも
オン抵抗が
小さいが、
トランジスタ数
が増える。

PMOS, NMOS, CMOSスイッチの 入力電圧に対するオン抵抗



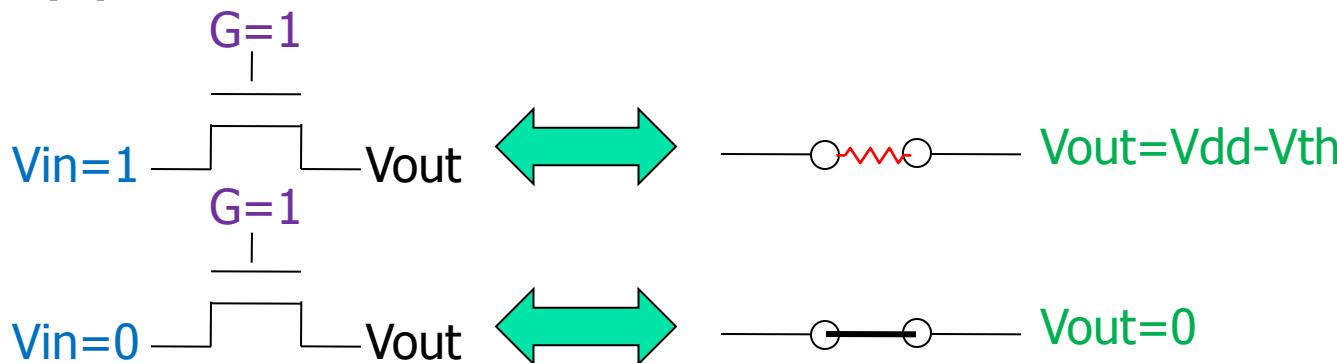
PMOS, NMOSスイッチの 出力電圧

(1) PMOS



PMOSは
 V_{out} は
GNDまで
下がらない。

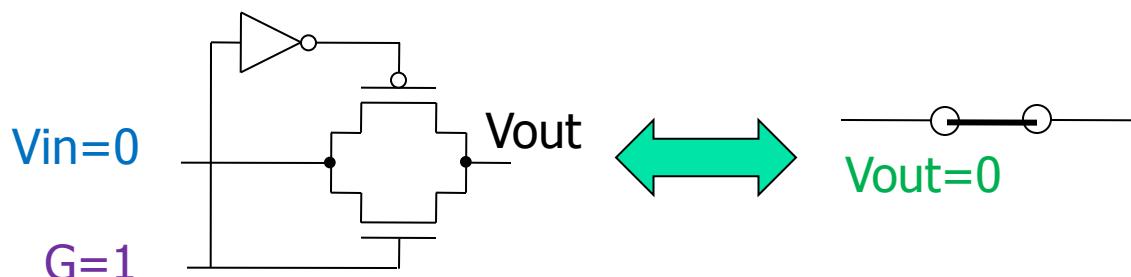
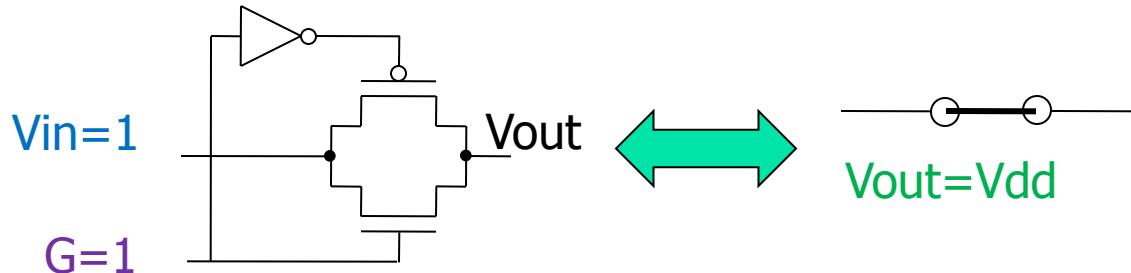
(2) NMOS



NMOSは
 V_{out} が
 V_{dd} 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 V_{out} が
 GND, V_{dd} 間を
フルスイング。

論理否定(NOT)

論理変数 A, Z

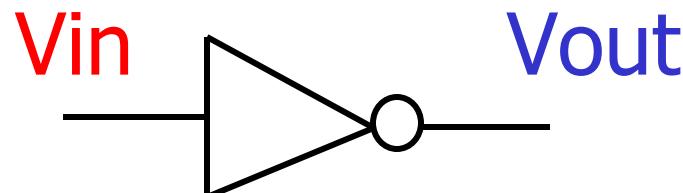
A : 入力, Z : 出力

$$Z = \overline{A}$$

真理値表

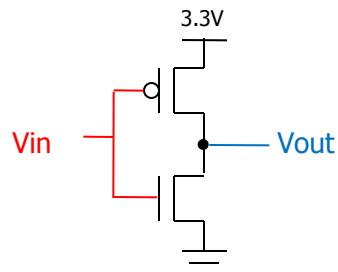
| A | Z |
|-----|-----|
| 0 | 1 |
| 1 | 0 |

NOT を実現する回路 \rightarrow インバータ回路

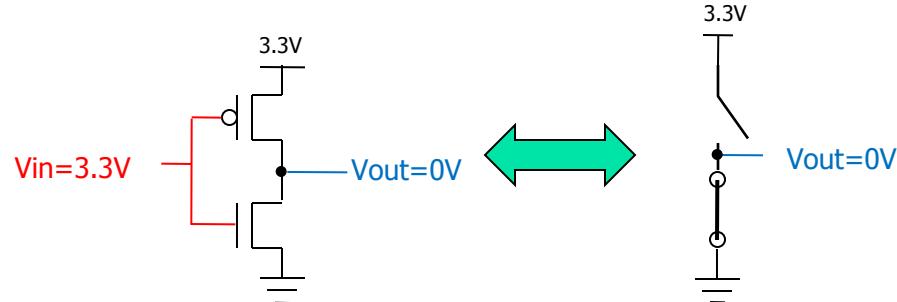


CMOSインバータ回路

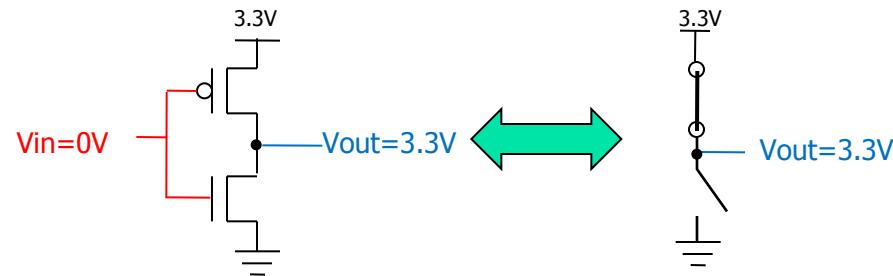
Inverter



a) When $V_{in}=1$ (3.3V)



b) When $V_{in}=0$



NAND

(NAND = AND + NOT)

論理変数 A,B, Z

A,B: 入力, Z: 出力

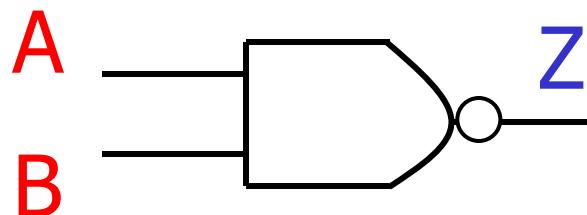
$$Z = \overline{A \cdot B}$$

| A | B | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

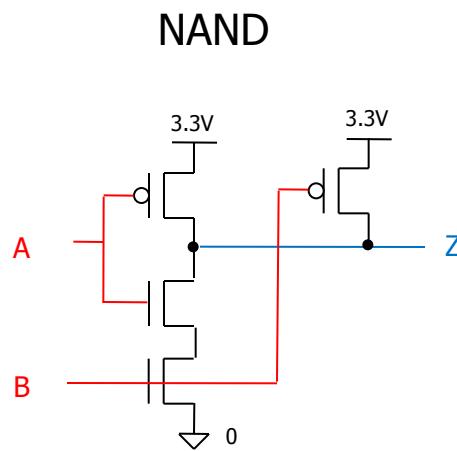
真理値表

NANDを実現する回路

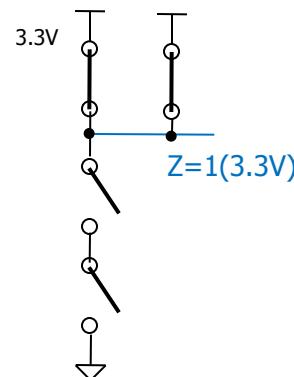
NAND回路



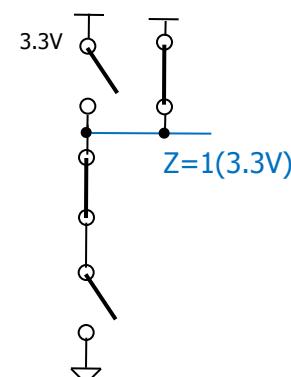
CMOS NAND回路



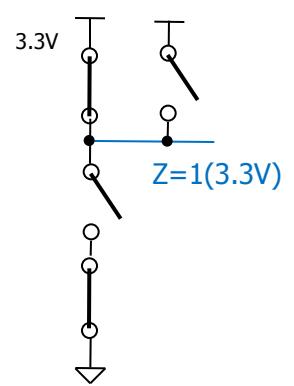
a) When $A=0, B=0$



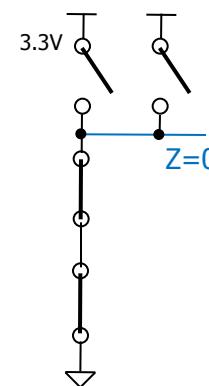
b) When $A=1, B=0$



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

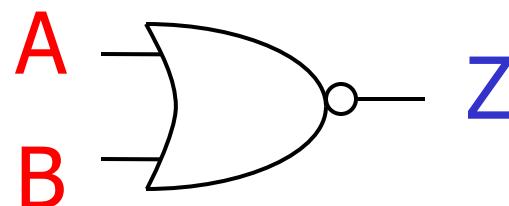
$$Z = \overline{A+B}$$

| A | B | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

真理値表

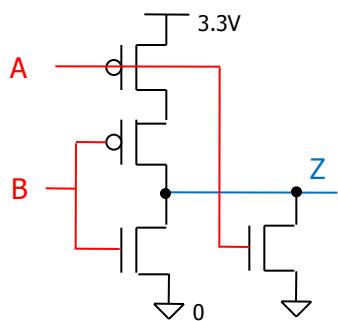
NORを実現する回路

NOR回路

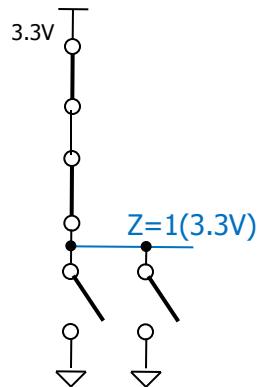


CMOS NOR回路

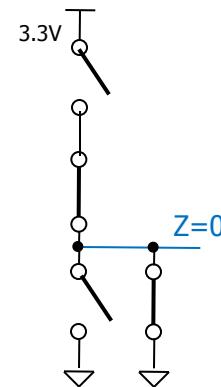
NOR回路



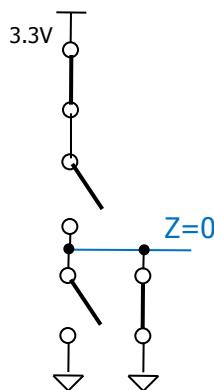
a) When $A=0, B=0$



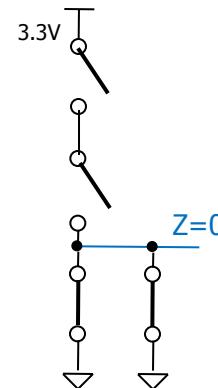
b) When $A=1, B=0$

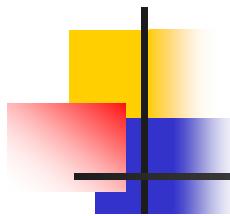


c) When $A=0, B=1$



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 | 真理値表 |
|---|---|---|------|
| 0 | 0 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 1 | |
| 1 | 1 | 1 | |

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。

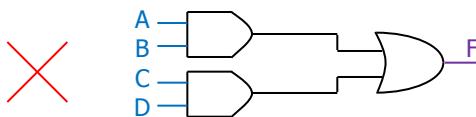
AND = NAND + Inverter

OR = NOR + Inverter

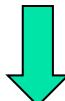
で構成（非効率）

例：

$F = A \bar{B} + C \bar{D}$ の回路実現

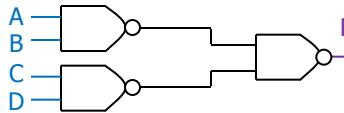
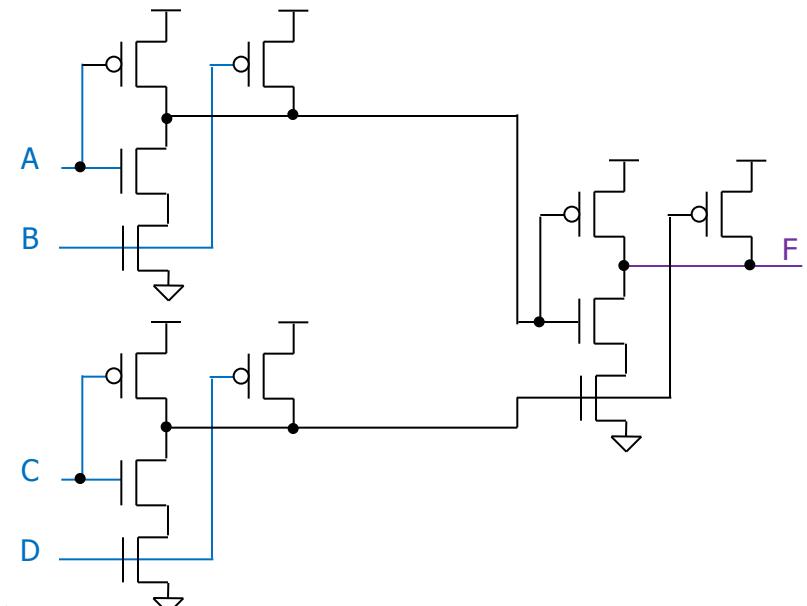
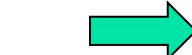
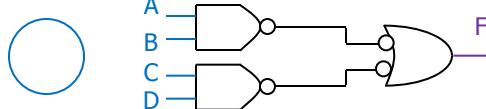


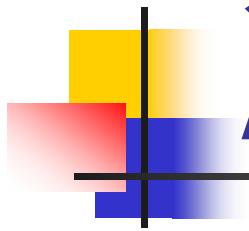
負論理



ド・モルガン
の法則

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





オーガスタス・ド・モルガン

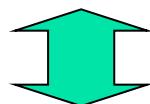
Augustus De Morgan, 1806-1817

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。

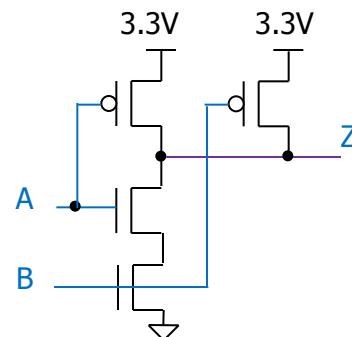
$$\overline{A \cdot B} = \overline{A} + \overline{B}, \quad \overline{A+B} = \overline{A} \cdot \overline{B}$$

「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



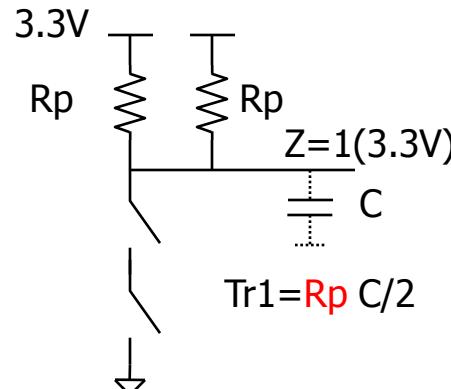
立上り時定数 $T_r = Rp C$
立下り時定数 $T_f = 2 Rn C$

$$Rp = 2 \sim 3 Rn$$

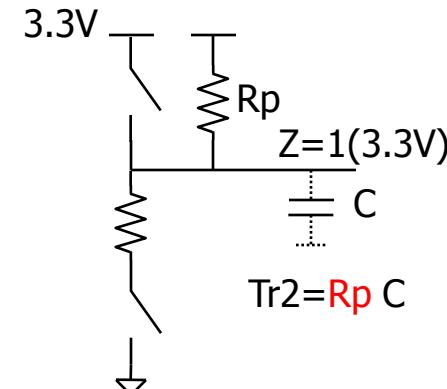


$$Tr = Tf$$

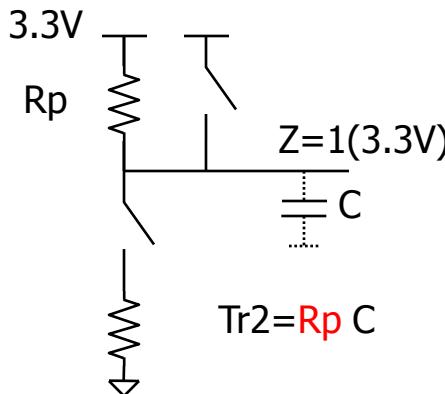
a) When A=0,B=0



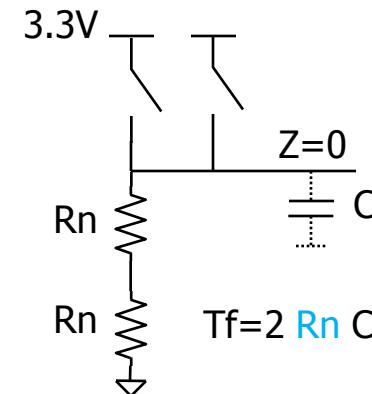
b) When A=1,B=0

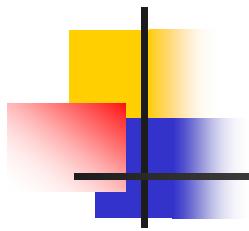


c) When A=0,B=1



d) When A=1,B=1

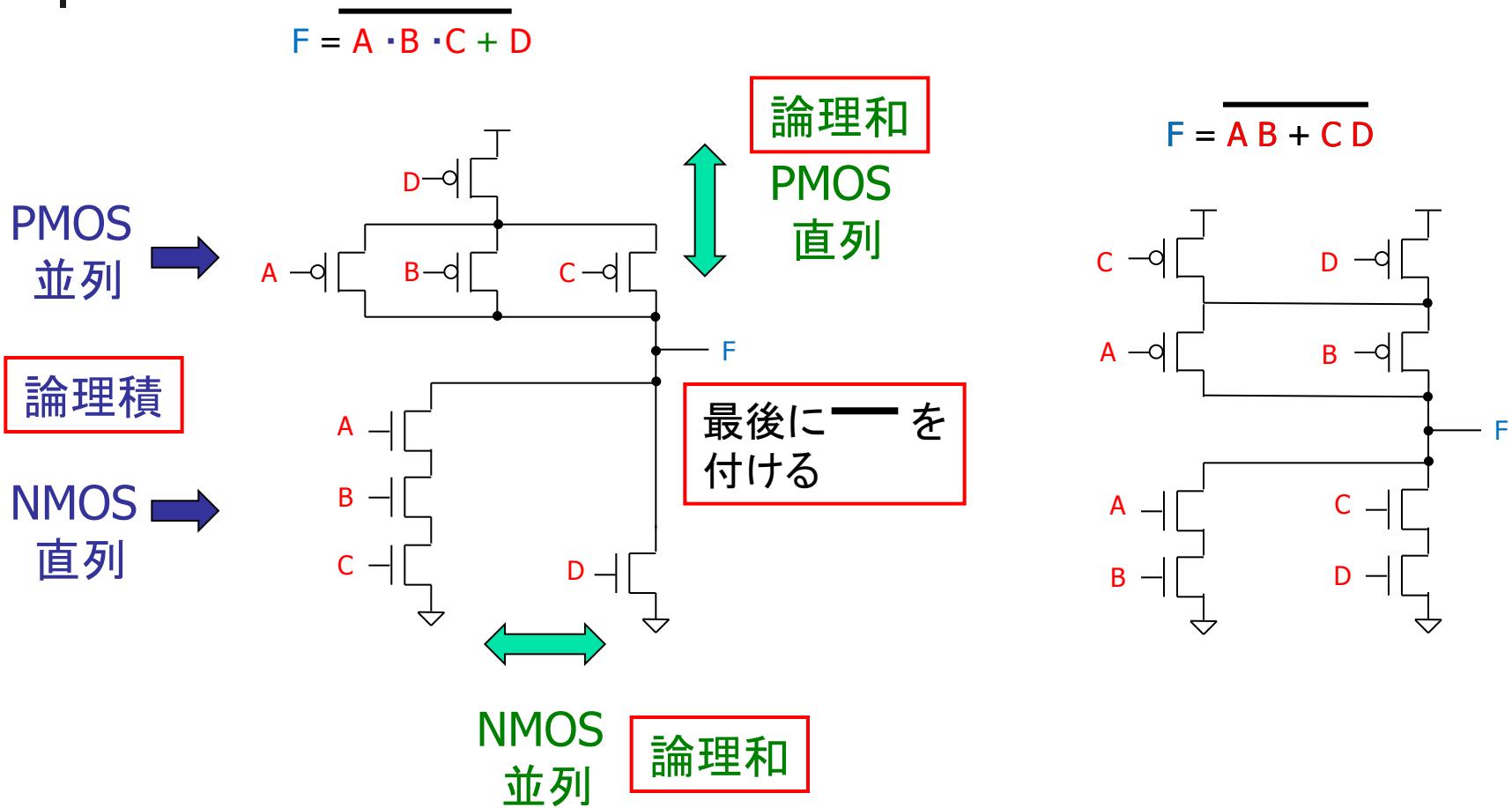




複合論理CMOS回路

| | | 論理積 | 論理和 |
|------|----|-----|-----|
| | | 並列 | 直列 |
| PMOS | 並列 | | |
| | 直列 | | 並列 |
| NMOS | 直列 | | |

複合論理CMOS回路 例

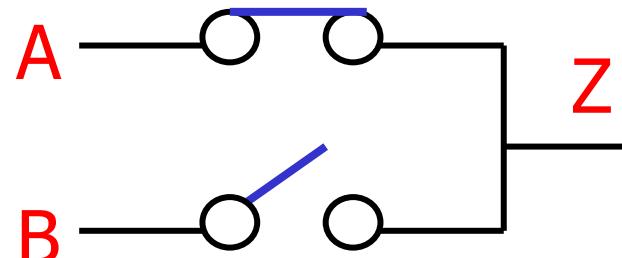


マルチプレクサ

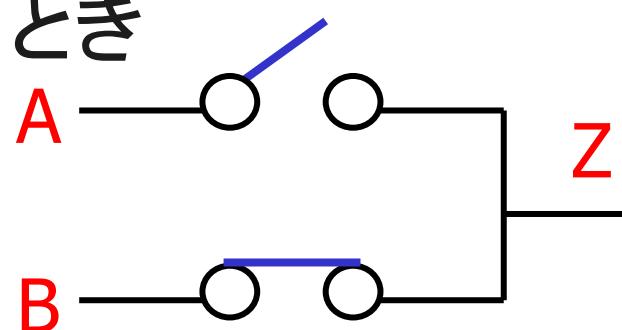
論理変数 A, B, S, Z

A, B, S : 入力, Z : 出力

$S=0$ のとき



$S=1$ のとき

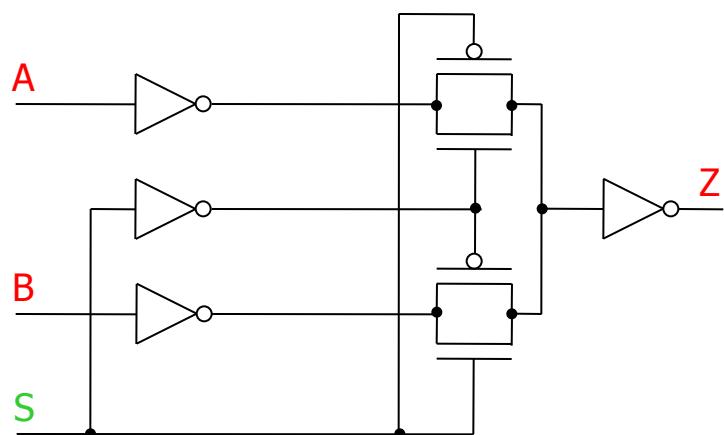


| S | Z |
|-----|-----|
| 0 | A |
| 1 | B |

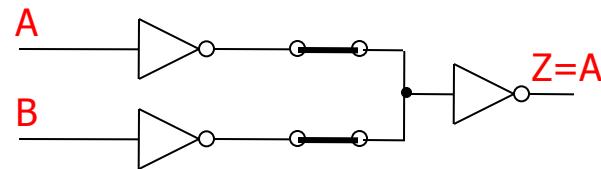
真理値表

CMOS マルチプレクサ回路

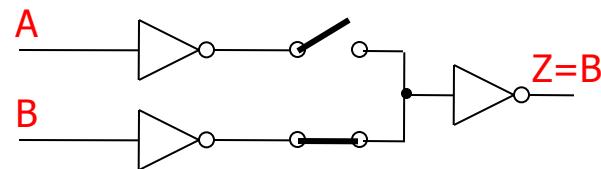
Multiplexer



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

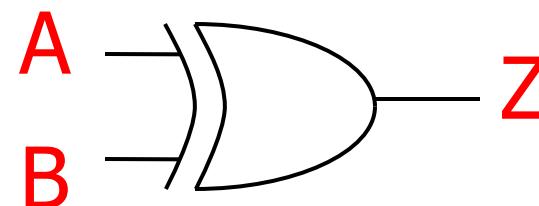
$$Z = A \oplus B$$

| A | B | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

真理値表

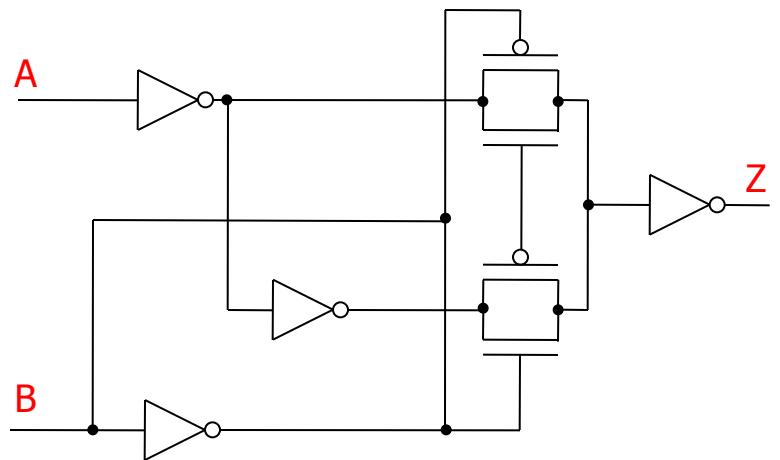
EXORを実現する回路

EXOR回路

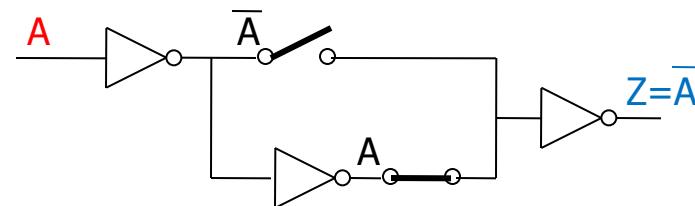


CMOS EXNOR回路

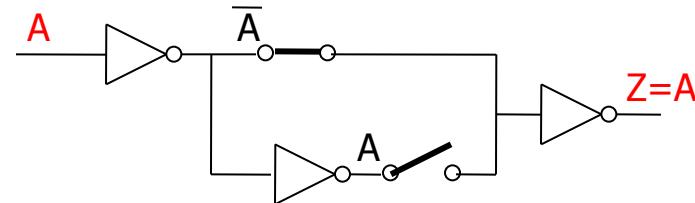
$$Z = A \bar{B} + \bar{A} \bar{B}$$



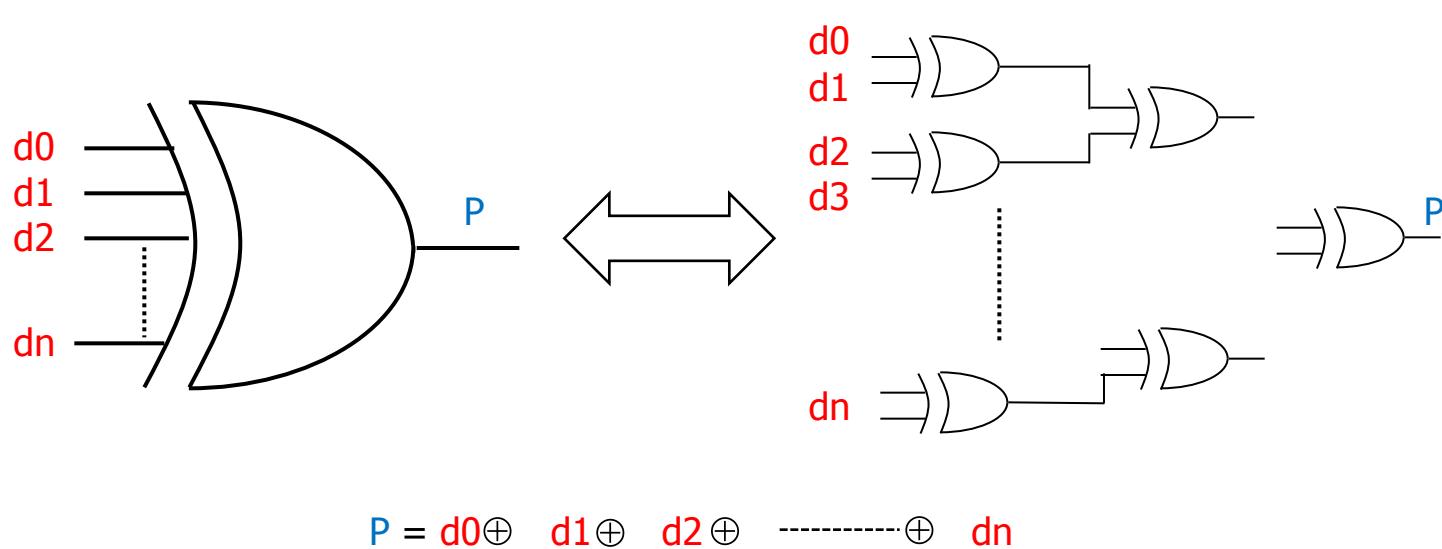
a) When $B = 0$



b) When $B = 1$



多入力EXOR 回路とパリティ



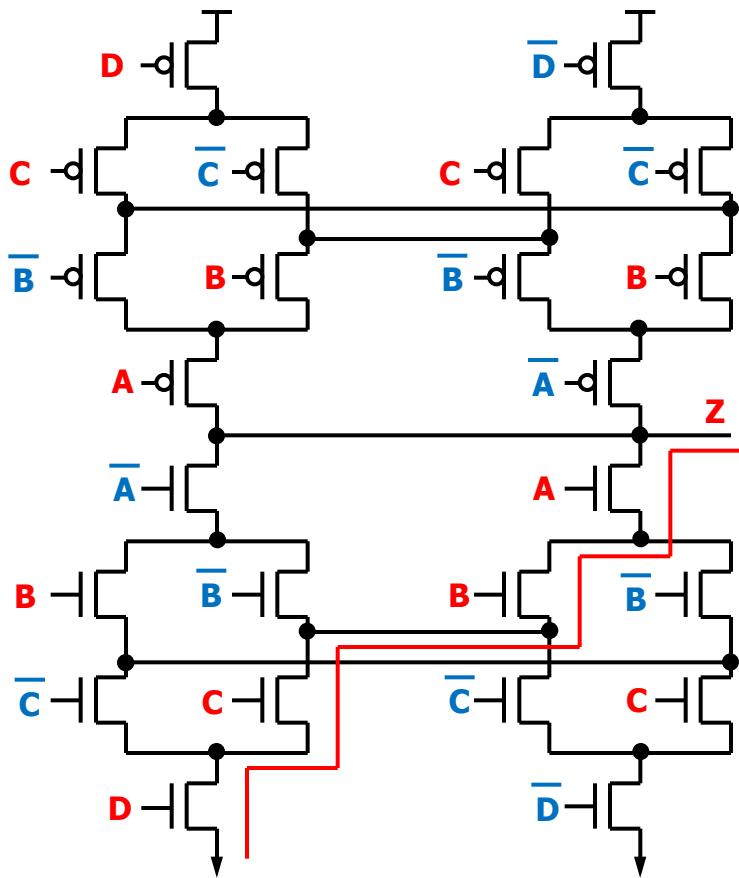
Parity

$d_0, d_1, \dots d_n$ の 1 の数が奇数個 $\rightarrow P=1$
偶数個 $\rightarrow P=0$

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 | 0 |
| 1 | 1 | 1 | 1 | 1 |

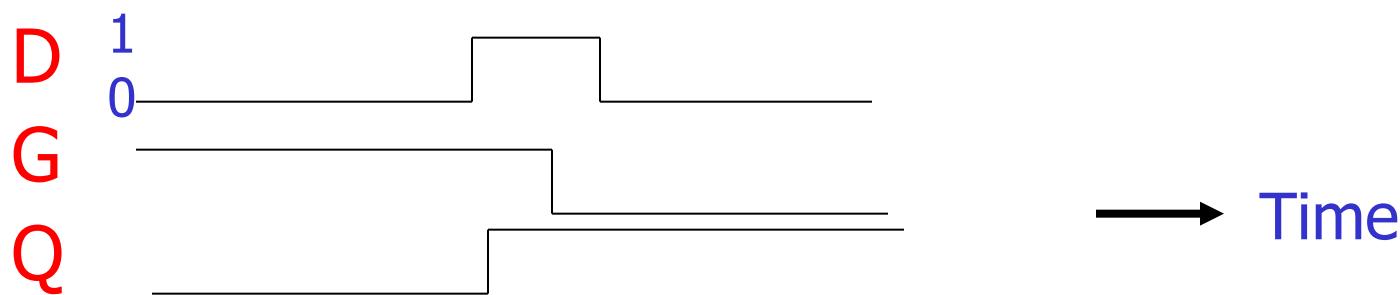
情報記憶素子(ラッチ)

論理変数 D, G, Q

D, G : 入力, Q : 出力

$G=1$ のとき $Q=D$

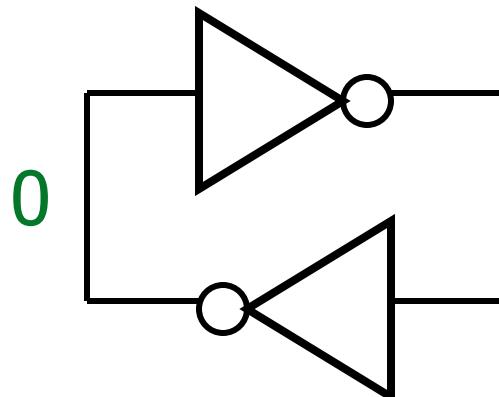
$G=0$ のとき Q は G が1から0になる瞬間の
 D の値(1 or 0)を保持(記憶)している。



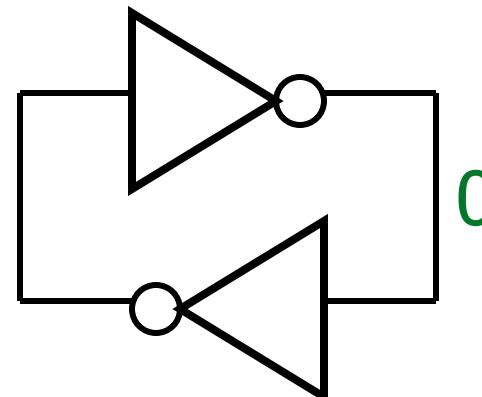
2つのインバータのリング接続 メモリ回路

2つの安定状態

データ“1”を記憶



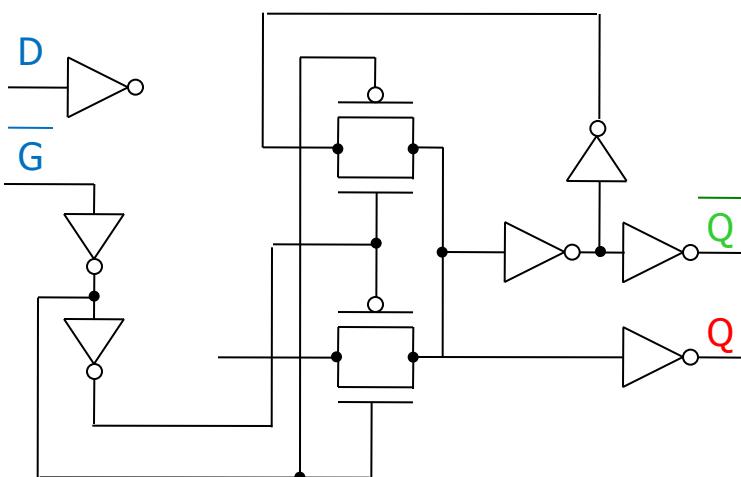
データ“0”を記憶



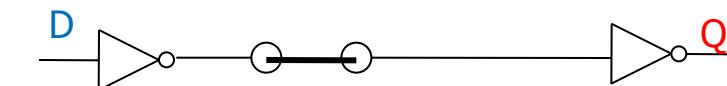
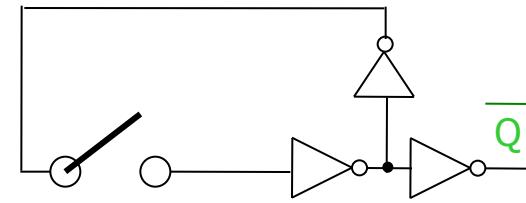
- SRAM (Static ランダム・アクセス・メモリ)
Latch, Flip-Flop 等のメモリ素子は
これを利用している。

CMOS ラッチ回路

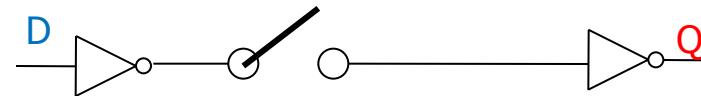
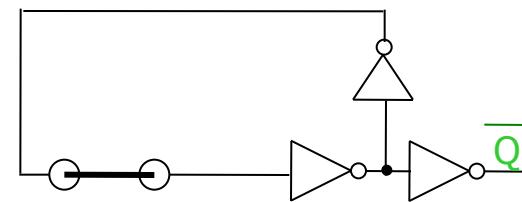
Latch回路
(メモリ素子)



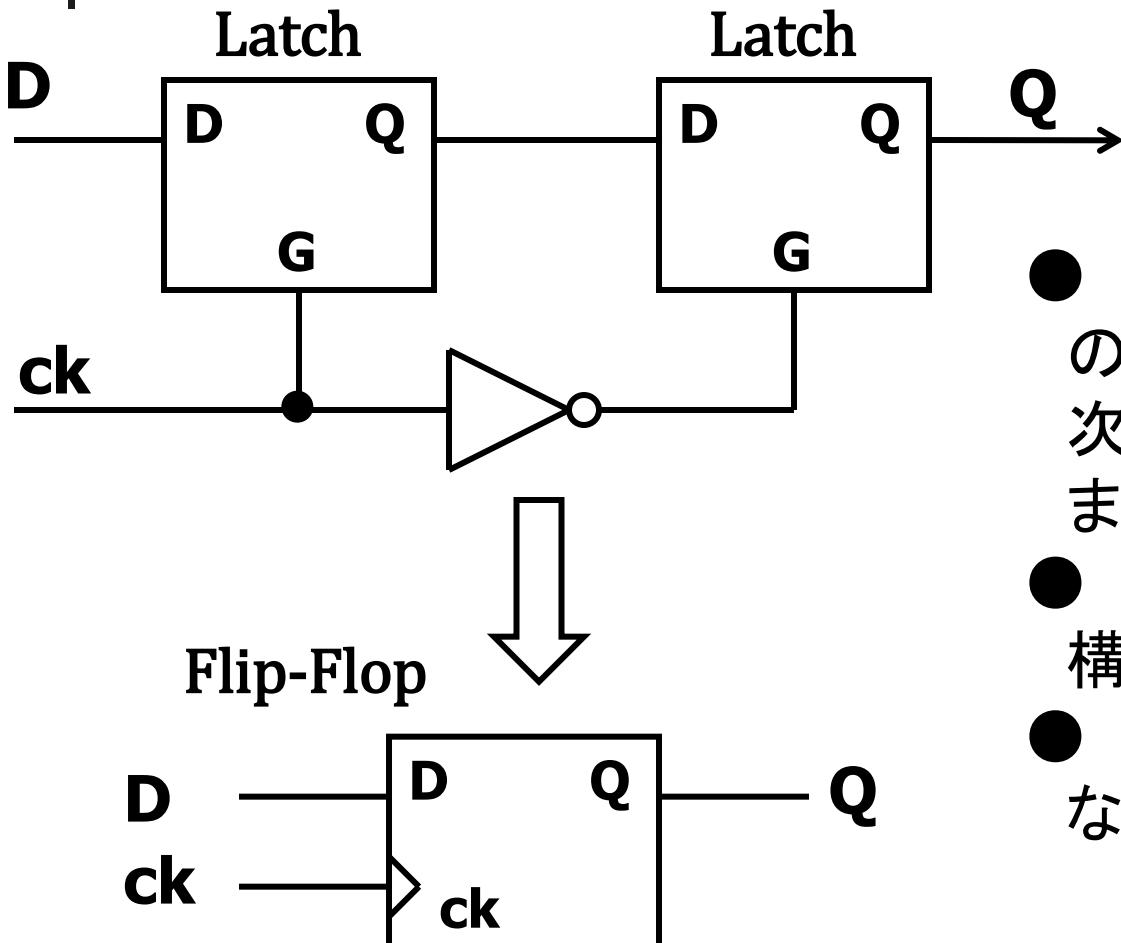
a) When $\bar{G} = 0$



b) When $\bar{G} = 1$

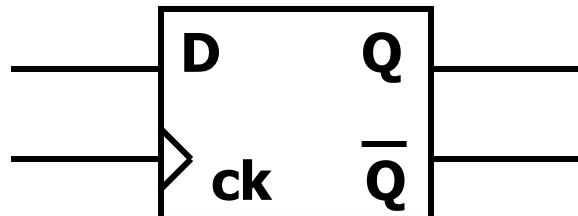


フリップ・フロップ回路

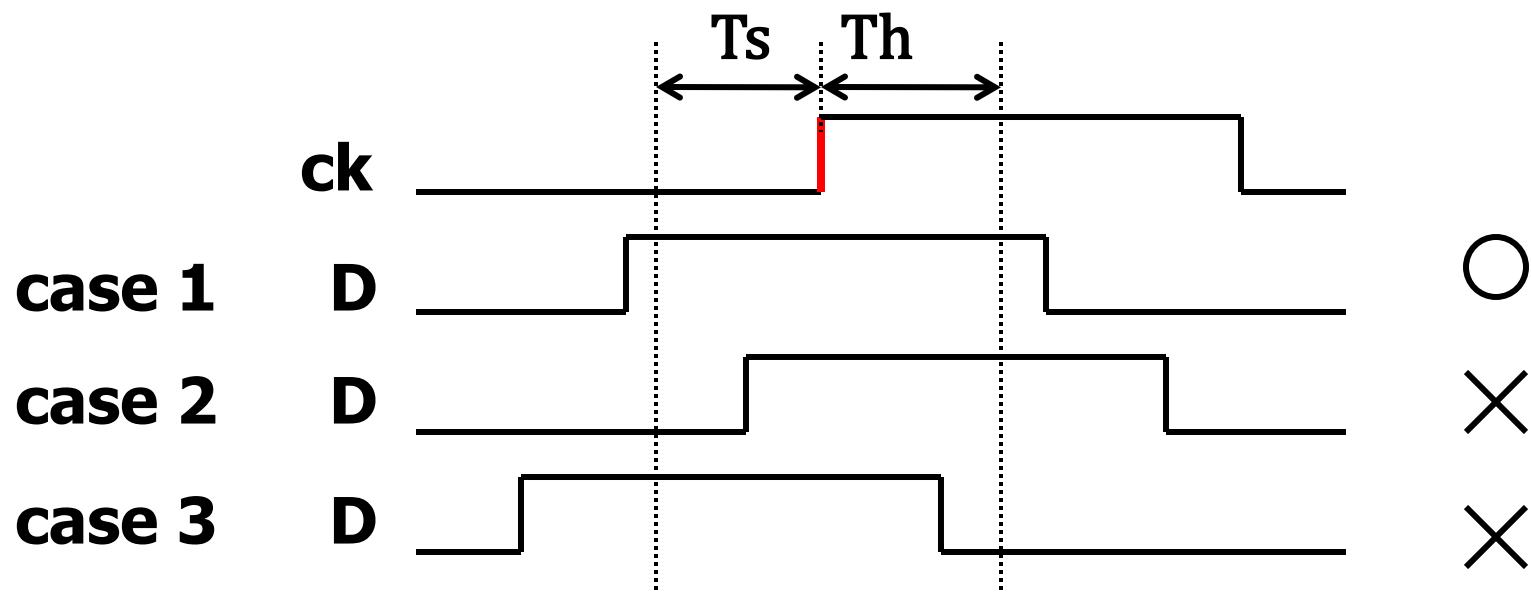


- クロックck の立ち上がりの瞬間のデータDを次のクロック立ち上がりまで(CK=1, 0でも)保持
- 2つのラッチ回路から構成
- 高速回路ではラッチではなくフリップフロップを使用

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



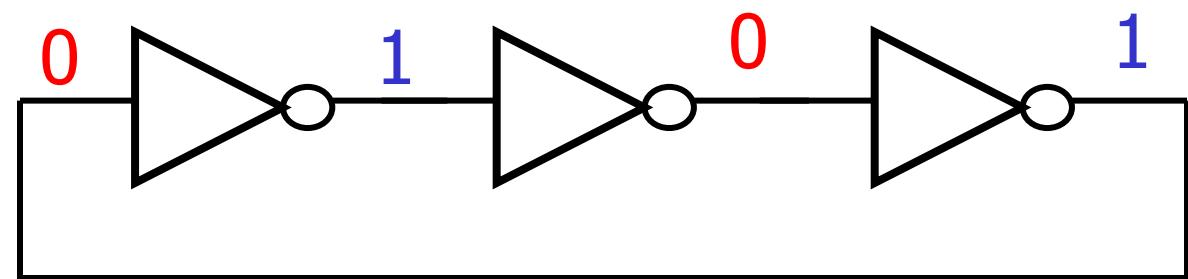
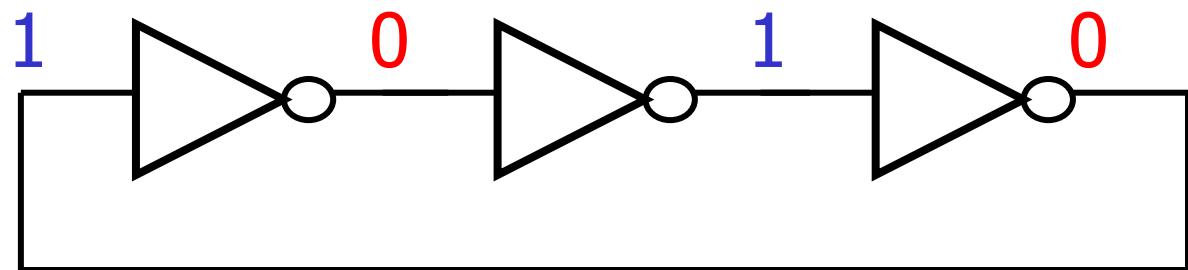
T_s : set-up time
 T_h : hold time



Set-up Time & Hold Time of Flip-Flop

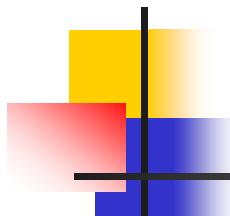
奇数個インバータのリング接続 リング発振器

安定状態
なし



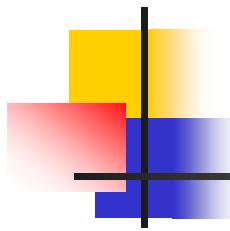
T: インバータ遅延、 $2N+1$ 個のインバタリング接続

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



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]

$$\text{Joule} = \text{Watt} \cdot s$$

電力は単位時間当たりに消費されるエネルギー

$$\text{電力} = \text{電圧} \cdot \text{電流} \quad P = V \cdot I$$

- 電流： 単位時間当たりに流れる電荷量

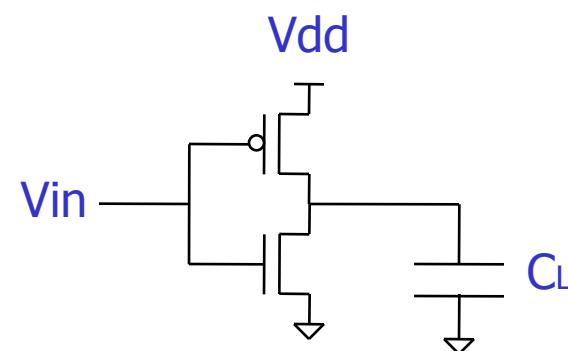
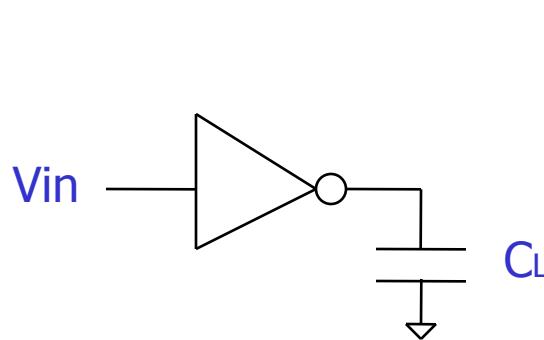
デジタルCMOS回路の電力消費

デジタルCMOS回路(インバータ)

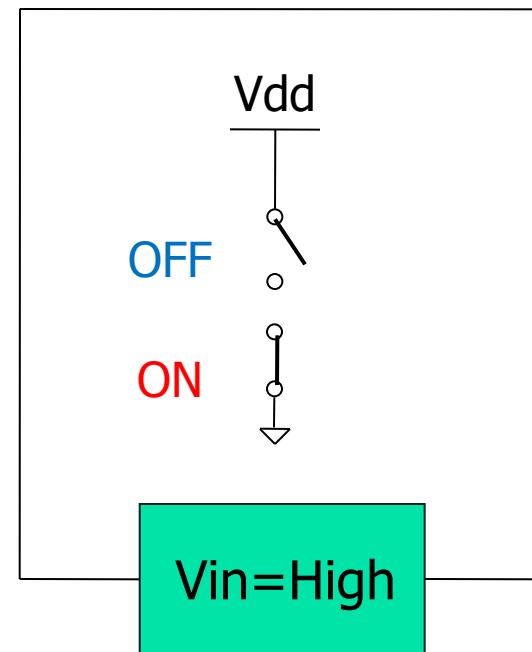
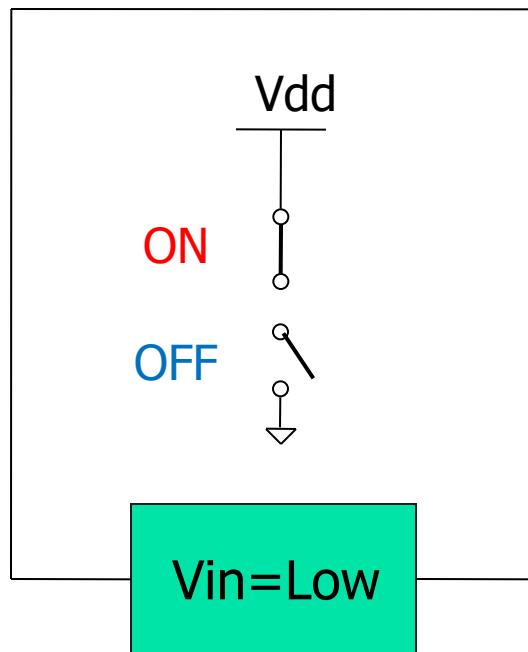
V_{dd} : 電源電圧

V_{in} : 入力、 V_{out} : 出力

C_L : 負荷容量

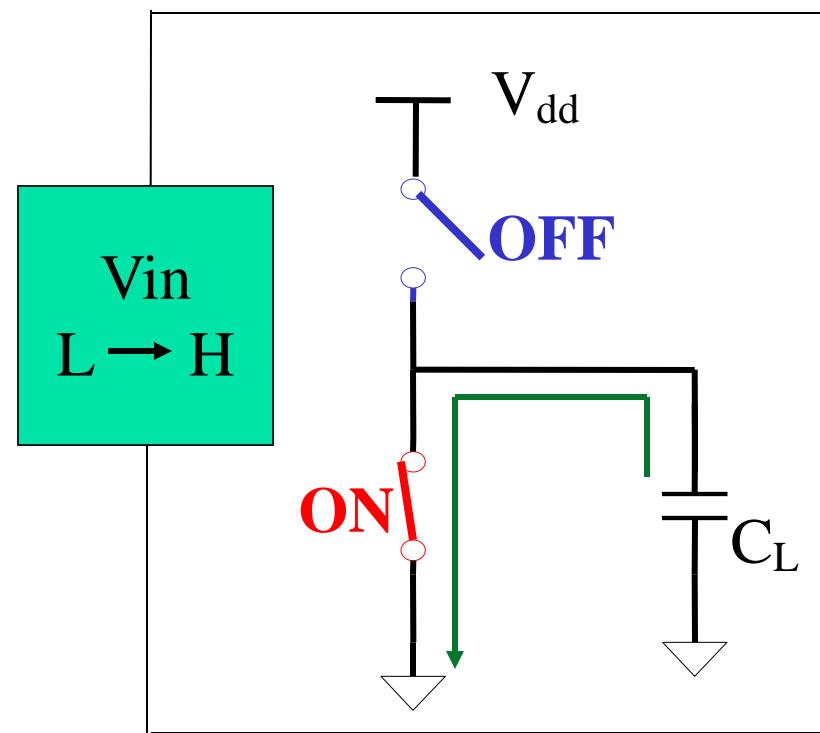
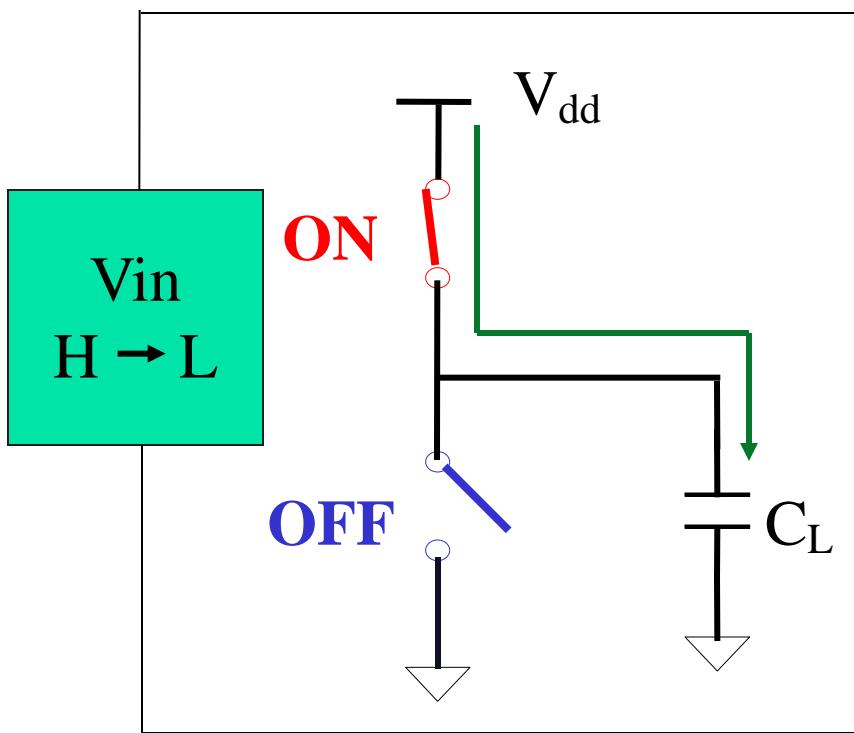


静的電力消費は小さい

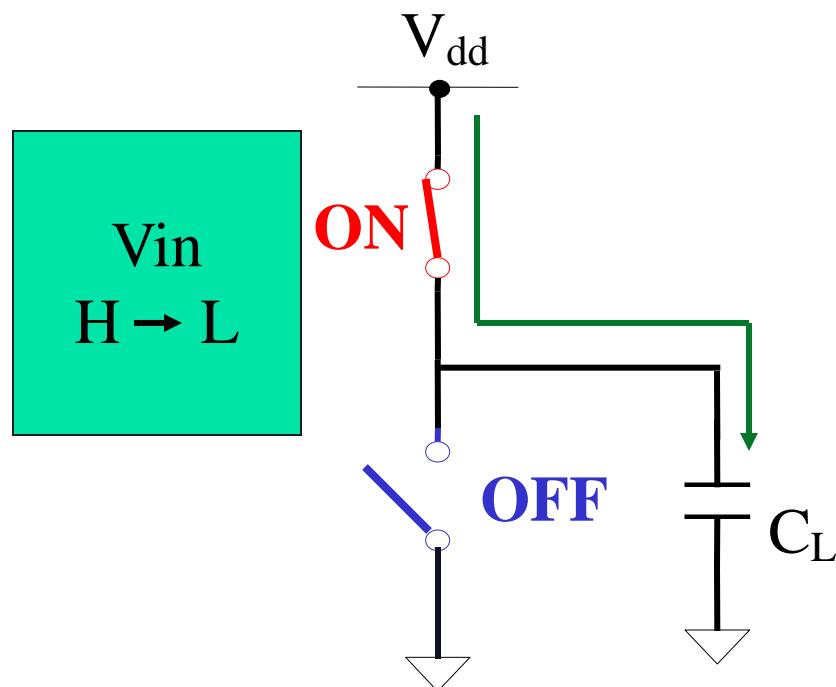


(注) 最近の微細CMOSデジタル回路では リーク電流
が大きくなり、静的電力消費の占める割合が増えてきている。

動的消費電力 (1)



動的消費電力 (2)



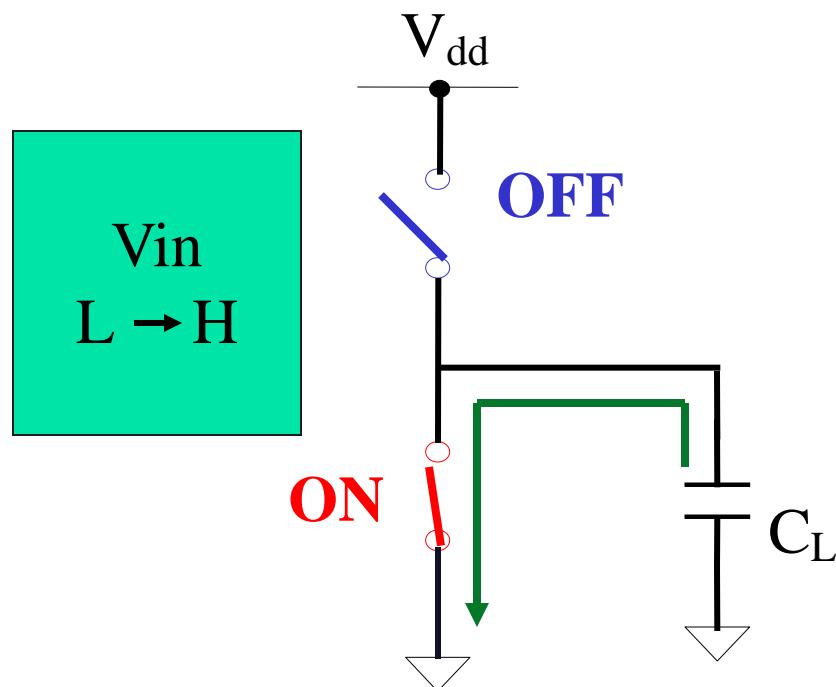
入力 V_{in}

High Low

蓄積電荷 Q

0 $C_L V_{dd}$

動的消費電力 (3)

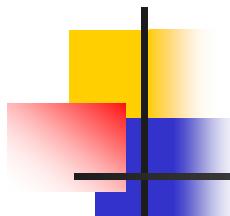


入力 V_{in}

Low $\xrightarrow{\hspace{1cm}}$ High

蓄積電荷 Q

$C_L V_{dd} \xrightarrow{\hspace{1cm}} 0$



動的消費電力 (4)

$V_{in} : H \rightarrow L \rightarrow H$ 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 output が f 回のトグルするとき

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

\therefore 消費電力

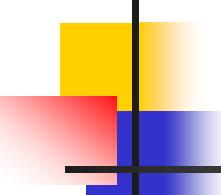
$$P = V_{dd} \cdot I$$

$$= V_{dd} (f \cdot C_L \cdot V_{dd})$$

$$= f \cdot C_L \cdot V_{dd}^2$$

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。

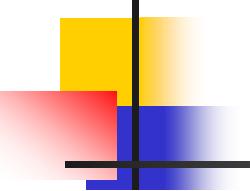
X

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

○



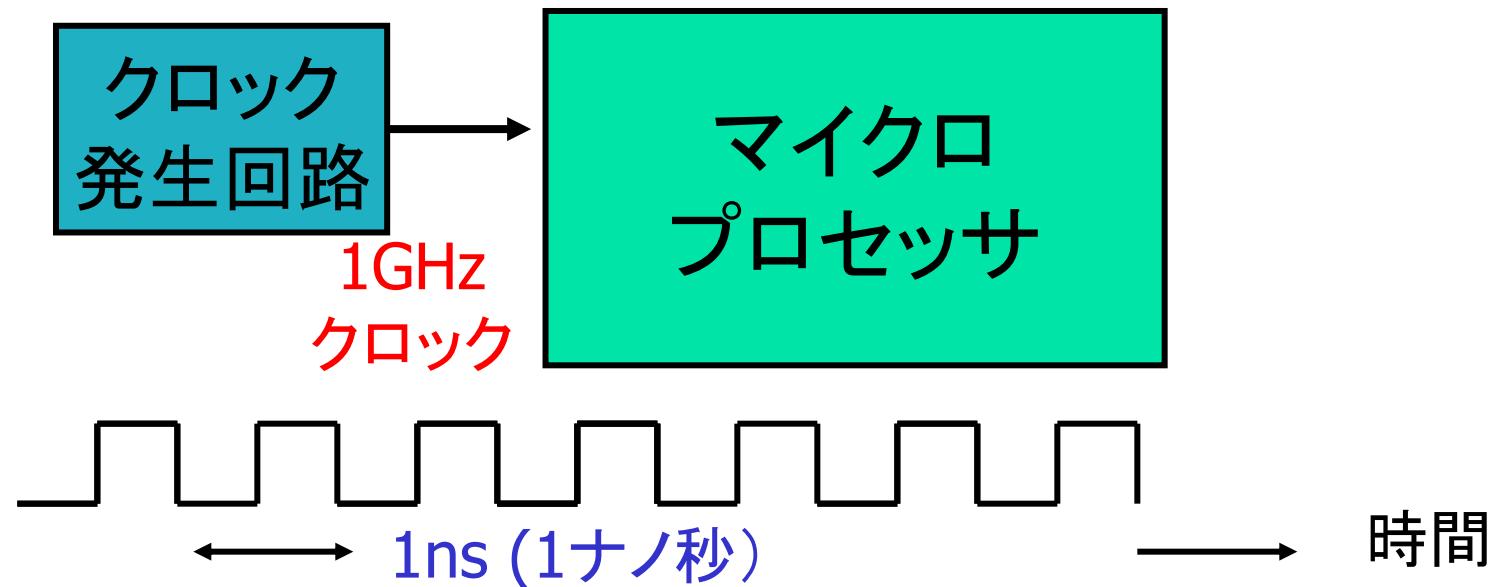
デジタルCMOS 回路の 低消費電力化技術 例

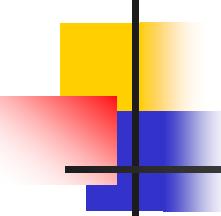
低消費電力化技術 → f , CL , V_{dd} を小さくする。

- 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 CL 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

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

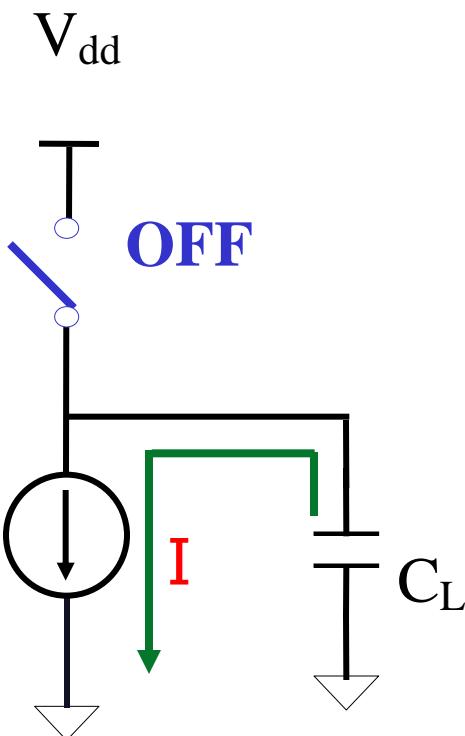
電源電圧 V_{dd} :

- 低消費電力化のため電源電圧を下げる
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

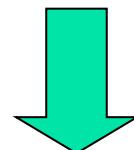
なぜ電源電圧を上げると デジタルCMOS回路は高速化するのか？



引き抜く電荷
 $Q = C V_{dd}$

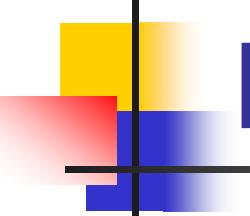
MOSの2乗則

$$I = K (V_{dd} - V_{th})^2$$
$$\doteq K V_{dd}^2$$



ゲート遅延

$$T = Q / I$$
$$= C / (K V_{dd})$$



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 V_{dd} , 1つのプロセッサ

電源電圧 V_{dd}

プロセッサ

$$\text{消費電力} = K (V_{dd})^2$$

$$\text{処理スピード} = L V_{dd}$$

ケース2：電源電圧 $V_{dd}/2$, 2つのプロセッサ

電源電圧 $V_{dd}/2$

プロセッサ

電源電圧 $V_{dd}/2$

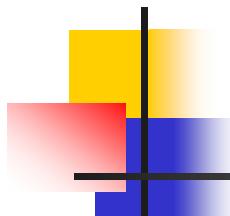
プロセッサ

$$\begin{aligned}\text{消費電力} &= K (V_{dd}/2)^2 + K (V_{dd}/2)^2 \\ &= (1/2) K (V_{dd})^2\end{aligned}$$

処理スピード

$$\begin{aligned}&= (1/2) L V_{dd} + (1/2) L V_{dd} \\ &= L V_{dd}\end{aligned}$$

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

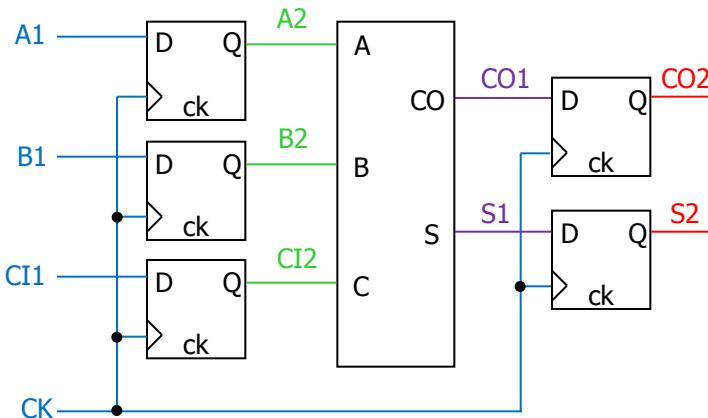


内 容

- 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

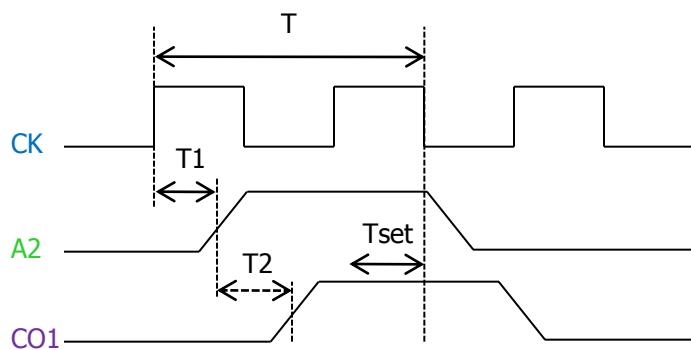
$$T > T_1 + T_2 + T_{set}$$

T : clock period

T1 : Flip-Flop Delay

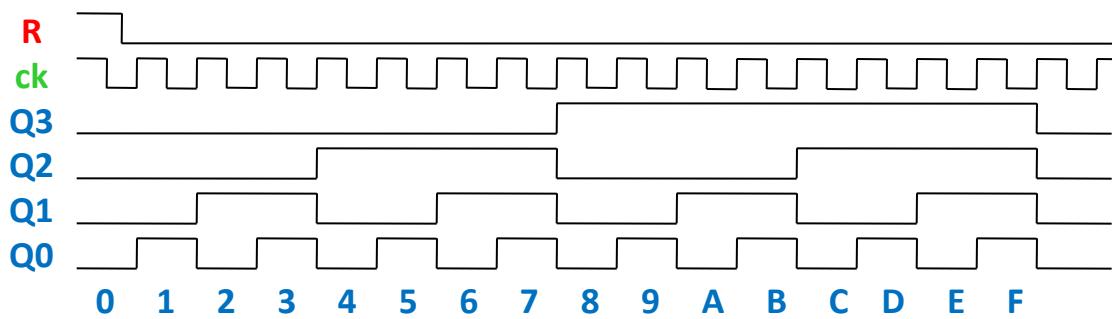
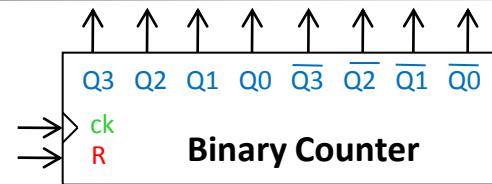
T2 : Full Adder Delay

Tset : Flip-Flop Setup Time

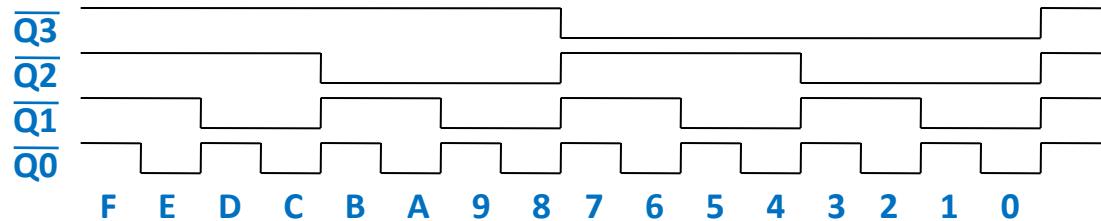


同期設計は
タイミング設計が
比較的容易

2進カウンタ (Binary Counter)



Up Counter

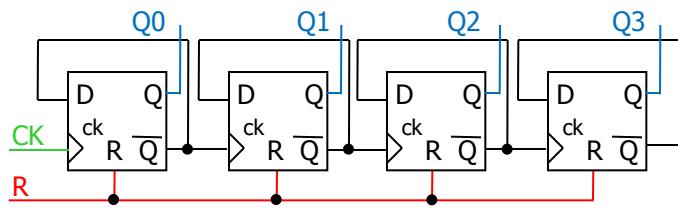


Down Counter

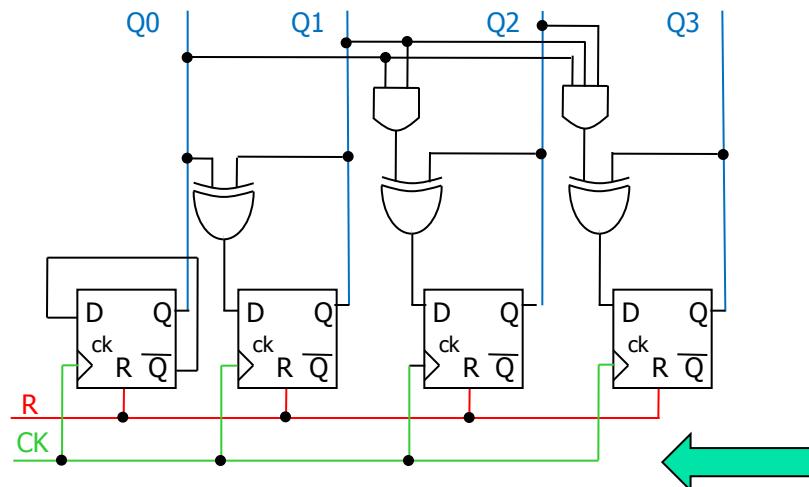
- クロック数を数える
- タイミング発生回路に使用される

非同期、同期カウンタ

非同期カウンタ



同期カウンタ

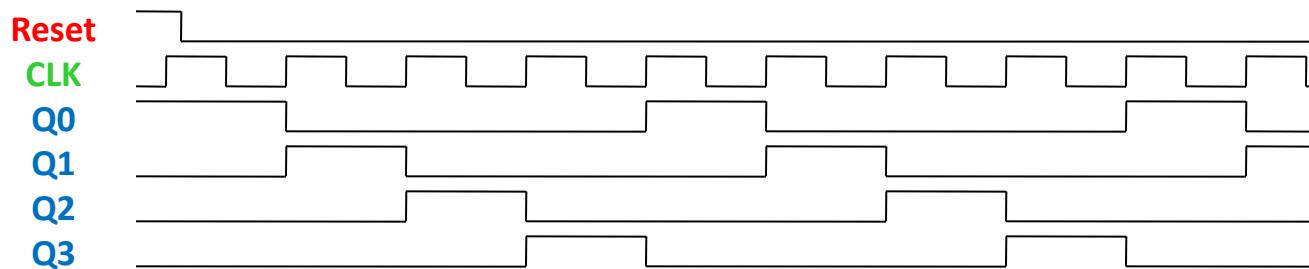
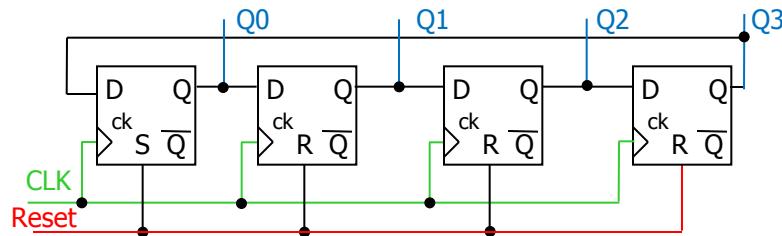


- 非同期回路は小規模になりえるが回路設計・変更・検証が難しい。

- デジタル回路は同期設計が基本

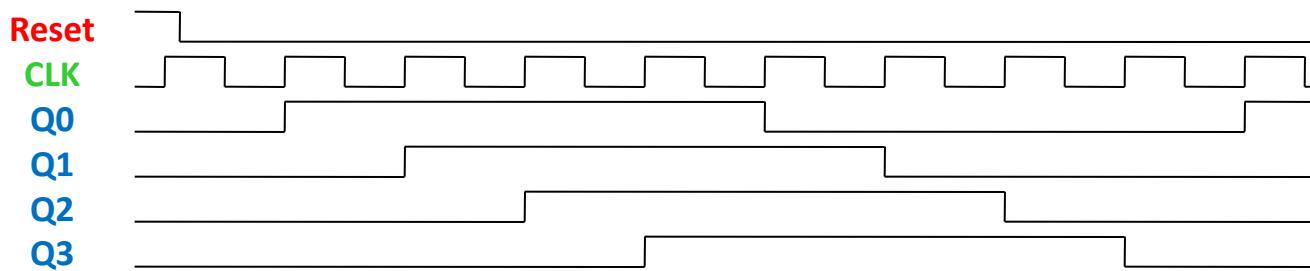
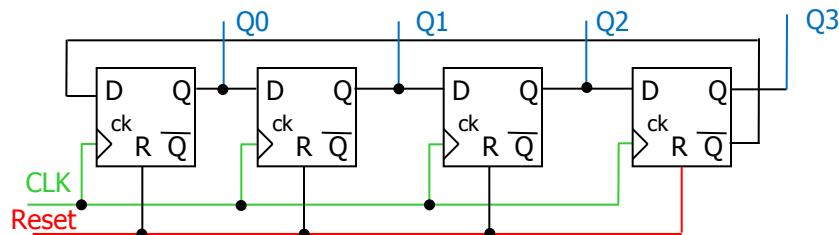
同期回路：
全てのフリップフロップのクロックが同一

リング・カウンタ (Ring Counter)



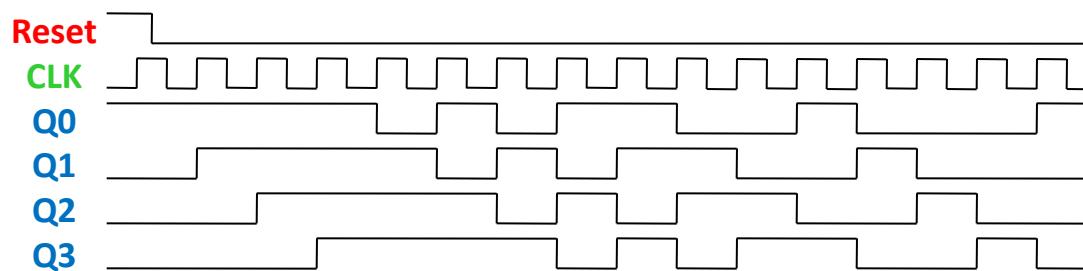
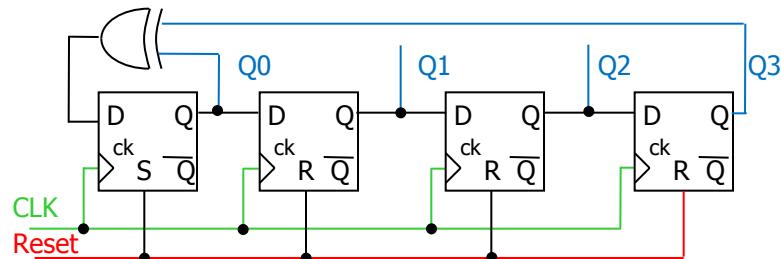
逐次比較近似ADCのタイミング発生回路等
に使用される。

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)

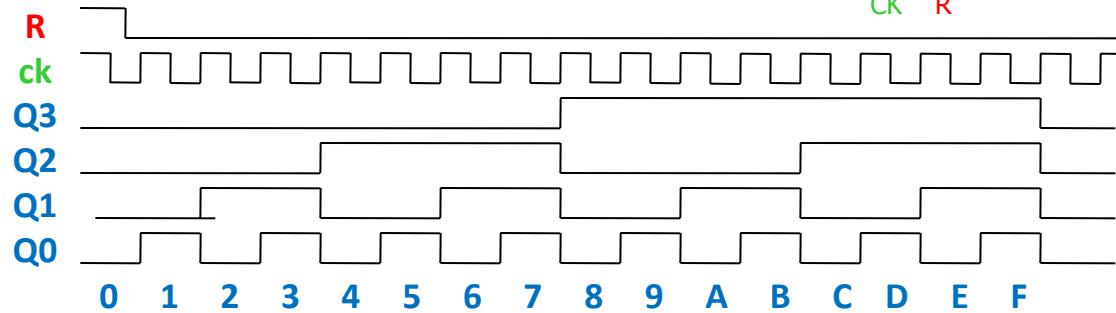
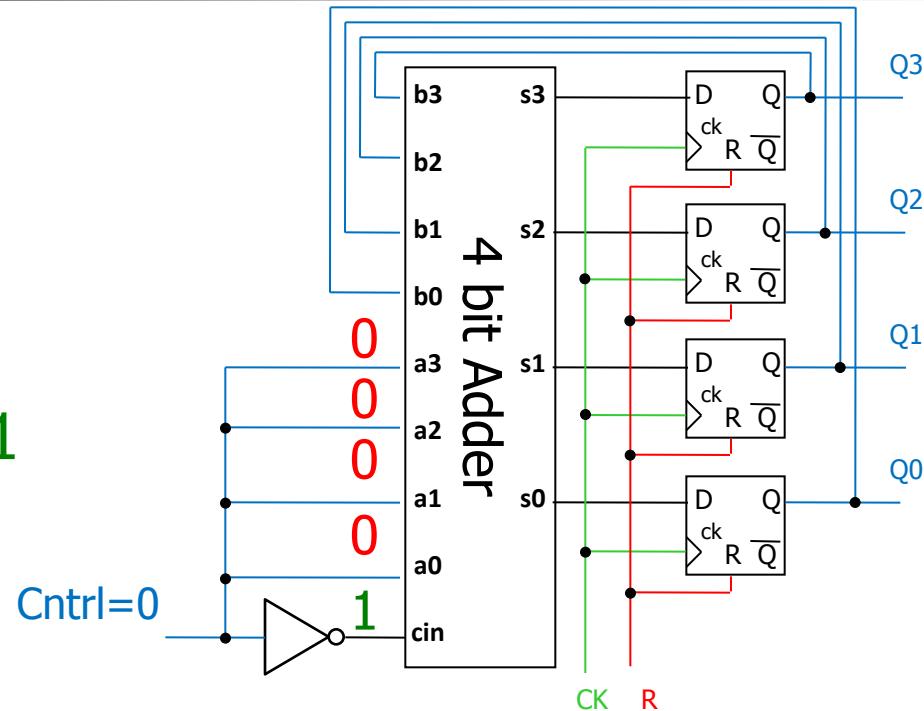


- $Q_0=Q_1=Q_2=Q_3=0$ 以外の15通りの信号を発生
- (再現性のある)疑似ランダム信号を発生
- フィードバックの取り方には決まりあり
- バックグラウンド自己校正等で疑似ランダム信号発生に使用される。

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

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

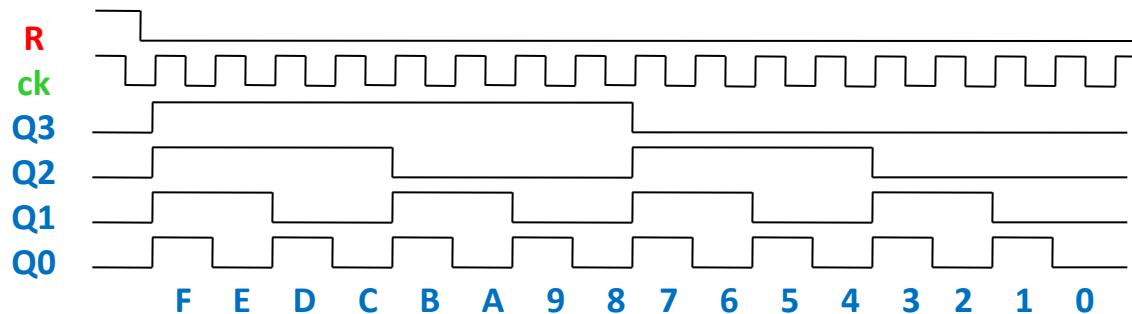
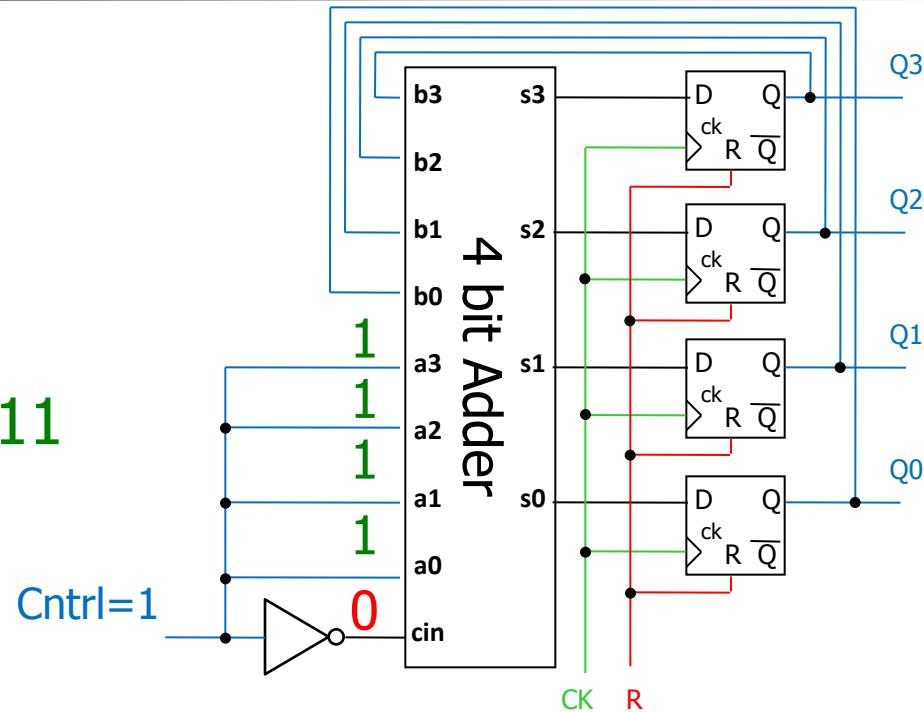


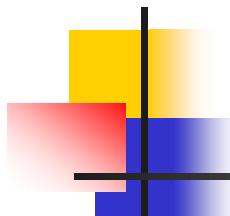
アップ・ダウン 2進カウンタ

制御信号

$Cntrl=1$ のとき
ダウンカウンタ

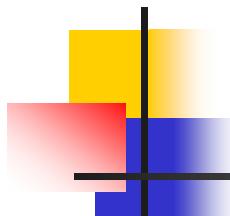
$$\begin{aligned} Q(n+1) &= \overrightarrow{Q(n)} + \overline{1111} \\ &= \overrightarrow{Q(n)} - 1 \end{aligned}$$





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

$$\begin{array}{r} 0011 \quad (3) \\ +) 1011 \quad (11) \\ \hline 1110 \quad (14) \end{array}$$

10進数の加算

$$\begin{array}{r} 437 \\ +) 258 \\ \hline 695 \end{array}$$

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)

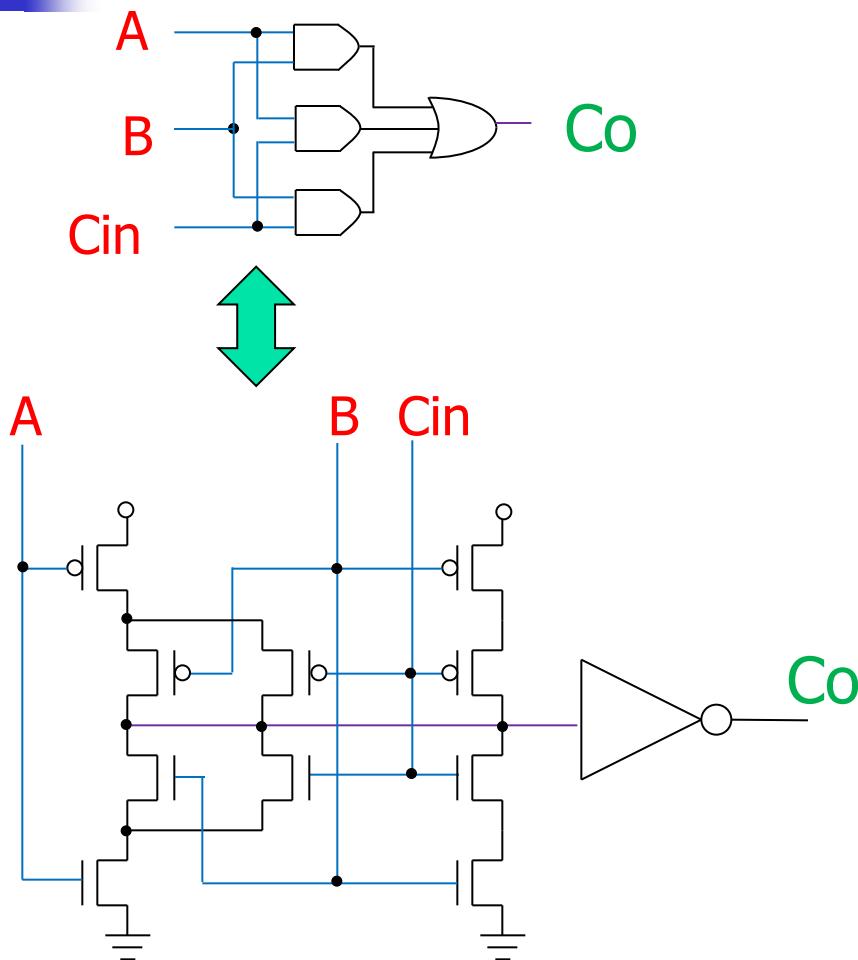
3入力2進加算

$$\begin{array}{r} A \quad \text{入力1} \\ B \quad \text{入力2} \\ +) \text{ Cin } \quad \text{下からの繰り上がり} \\ \hline \text{Co S} \\ S = A \oplus B \oplus \text{Cin} \\ \text{Co} = B \cdot \text{Cin} + A \cdot \text{Cin} + A \cdot B \\ (\text{Co} \text{ は } A, B, \text{Cin} \text{ の} \\ \text{多数決}) \end{array}$$

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)



真理值表

| 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)

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

を考える。

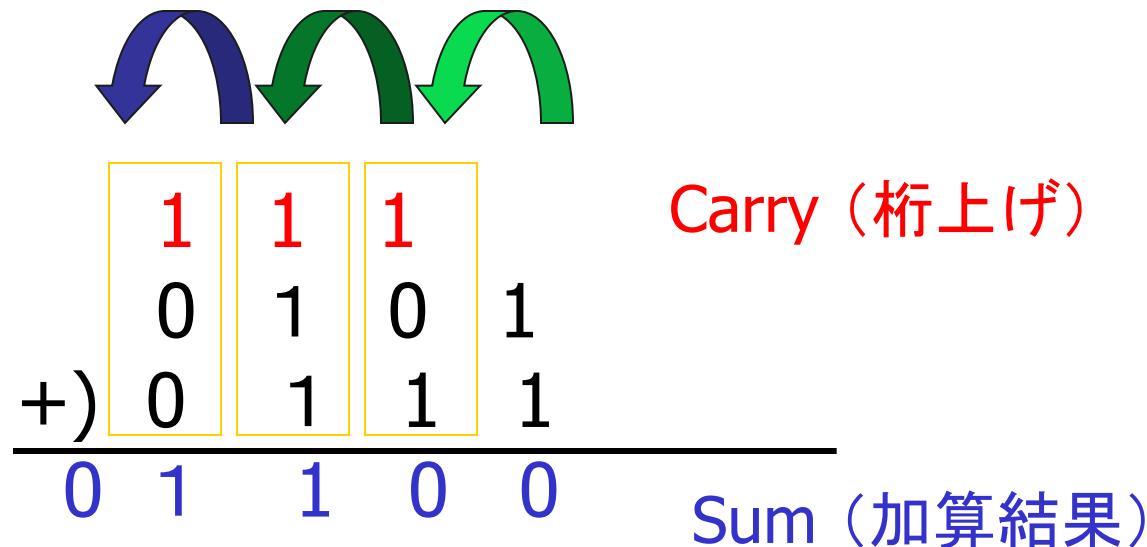


全加算器
の演算

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

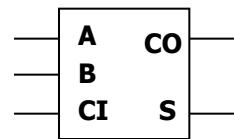
全加算器の補足説明(続き)

$$\begin{array}{r} 0 & 1 & 0 & 1 & (5) \\ +) 0 & 1 & 1 & 1 & (7) \\ \hline 0 & 1 & 1 & 0 & 0 & (12) \end{array} \quad \text{を考える。}$$



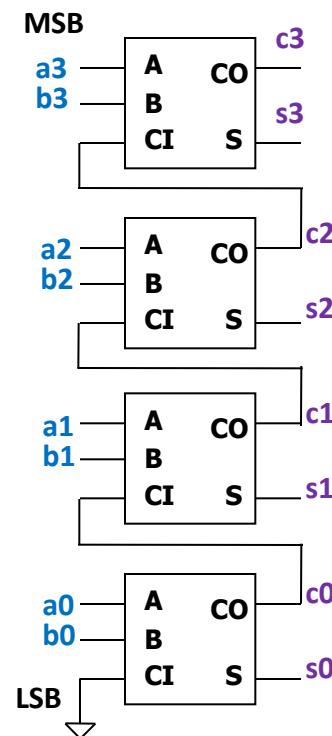
全加算器と桁上げ伝播

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 :

$$\begin{array}{r}
 0111 \\
 +0101 \\
 \hline
 1100
 \end{array}$$

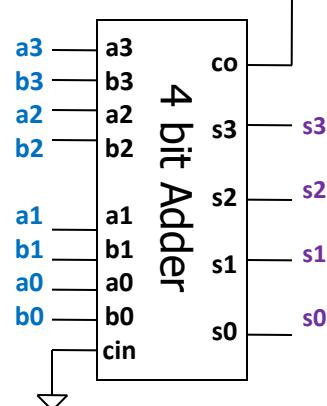
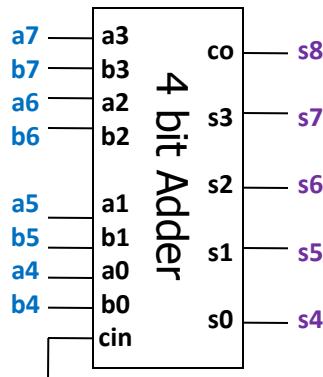


$$\begin{aligned}
 (a_3 \ a_2 \ a_1 \ a_0) &= (0 \ 1 \ 1 \ 1) \\
 (b_3 \ b_2 \ b_1 \ b_0) &= (0 \ 1 \ 0 \ 1)
 \end{aligned}$$

$$\begin{aligned}
 (s_3 \ s_2 \ s_1 \ s_0) &= (1 \ 1 \ 0 \ 0) \\
 (c_3 \ c_2 \ c_1 \ c_0) &= (0 \ 1 \ 1 \ 1)
 \end{aligned}$$

桁上げ選択加算器 (Carry Select Adder) による高速化

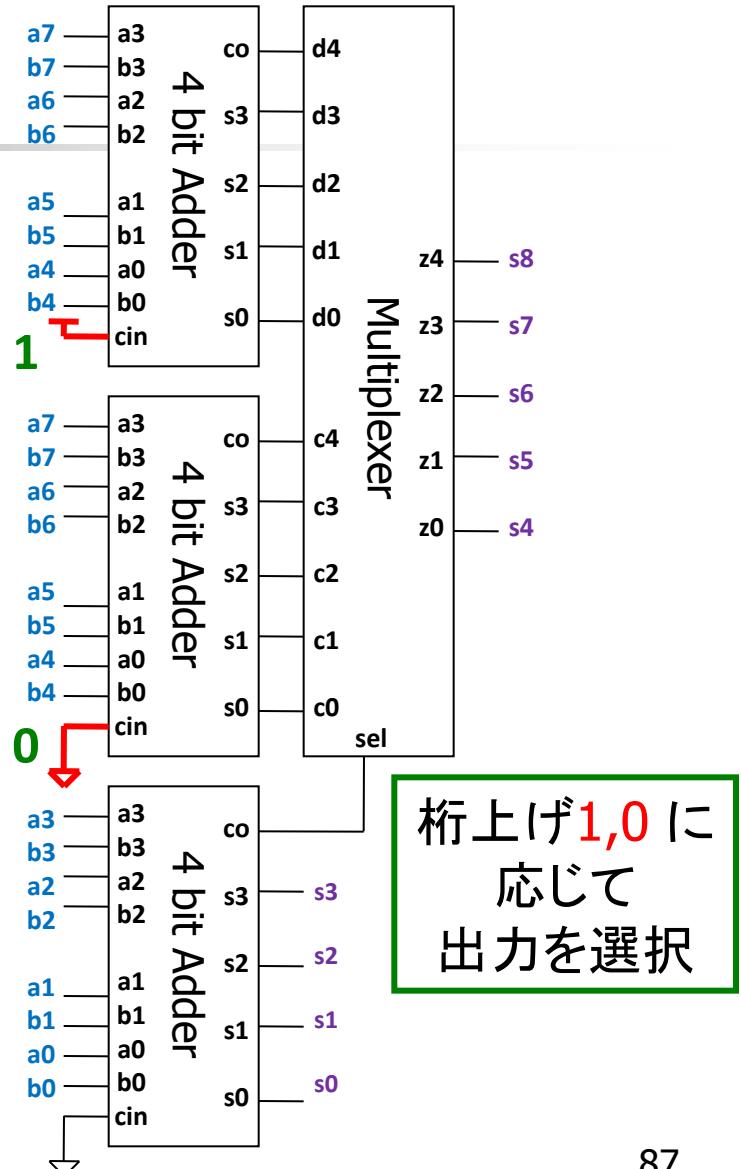
通常の
8bit 加算器



桁上げ1の場合を
計算

桁上げ0の場合を
計算

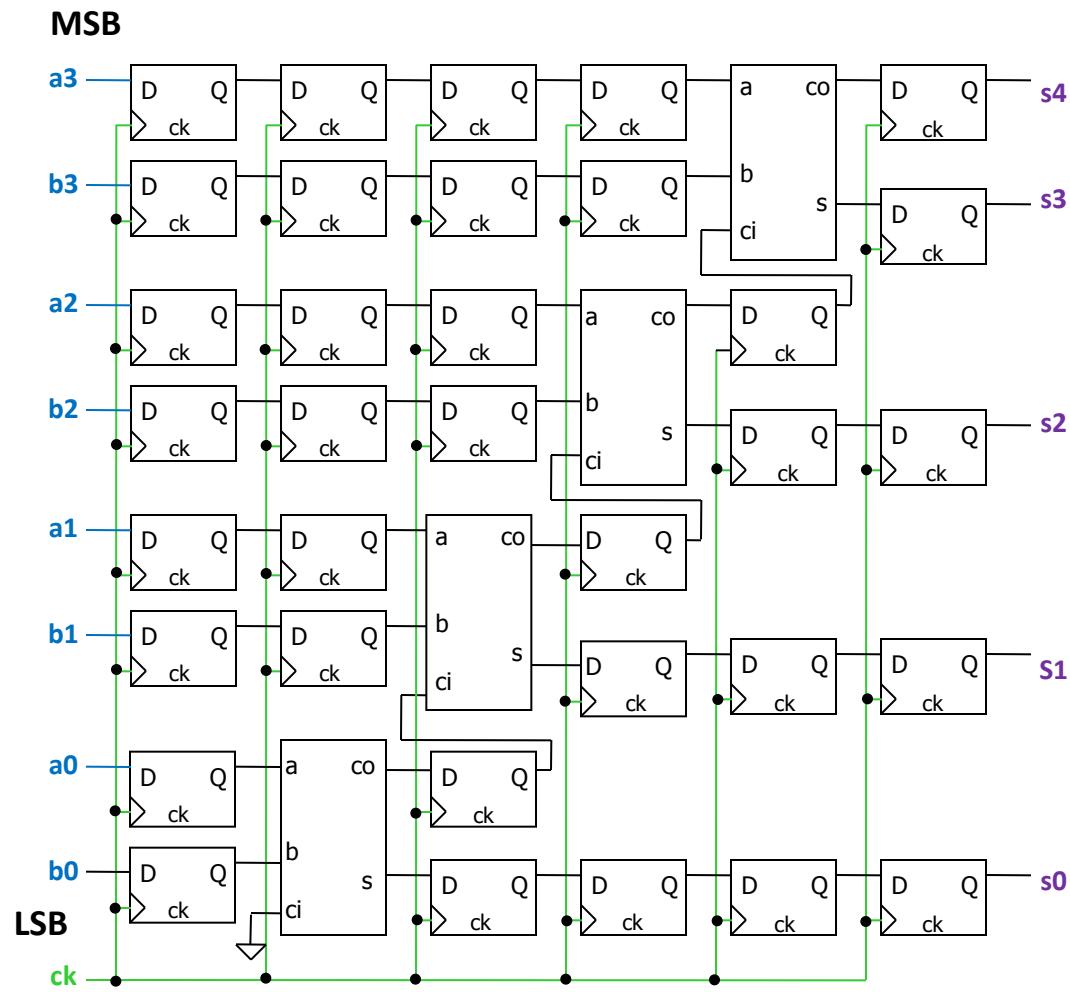
約1.5倍の
回路規模で
約2倍の
スピード



桁上げ1,0に
応じて
出力を選択

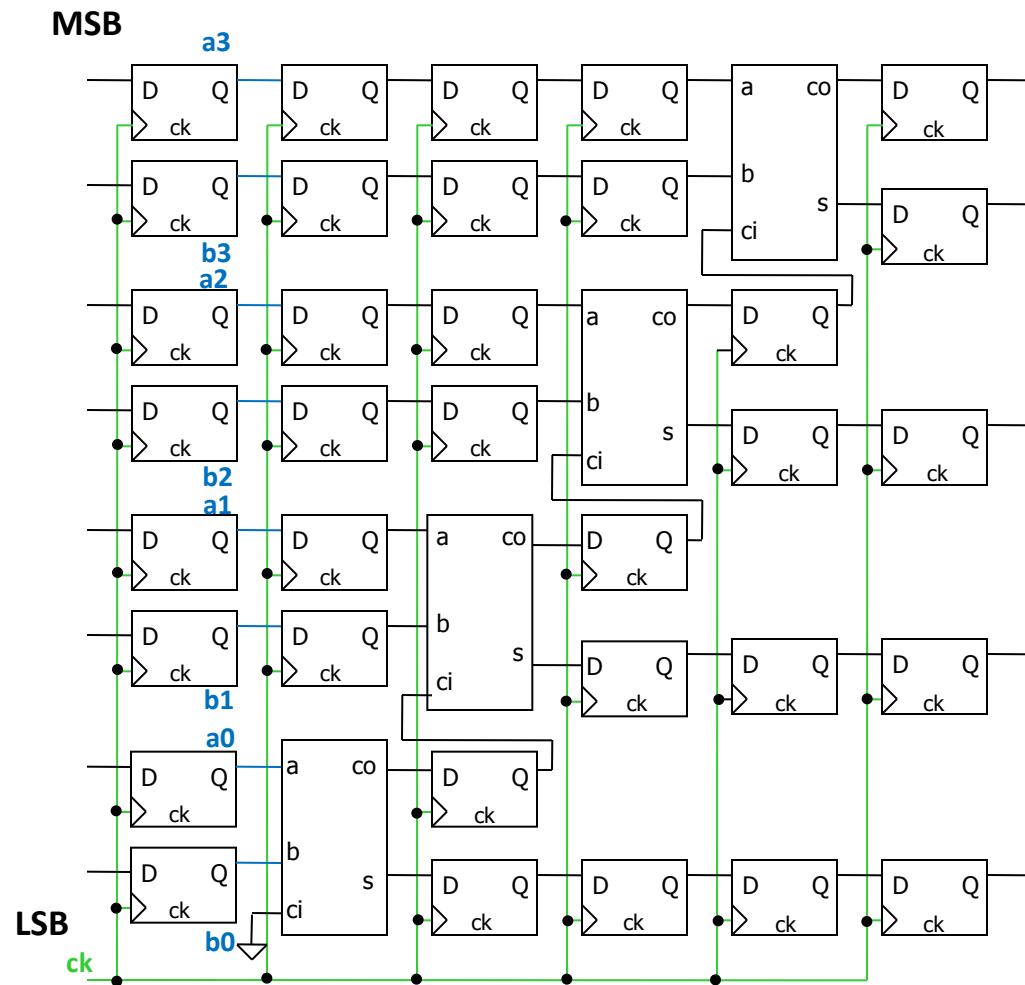
パイプライン加算器の構成

初期状態



パイプライン加算器の動作

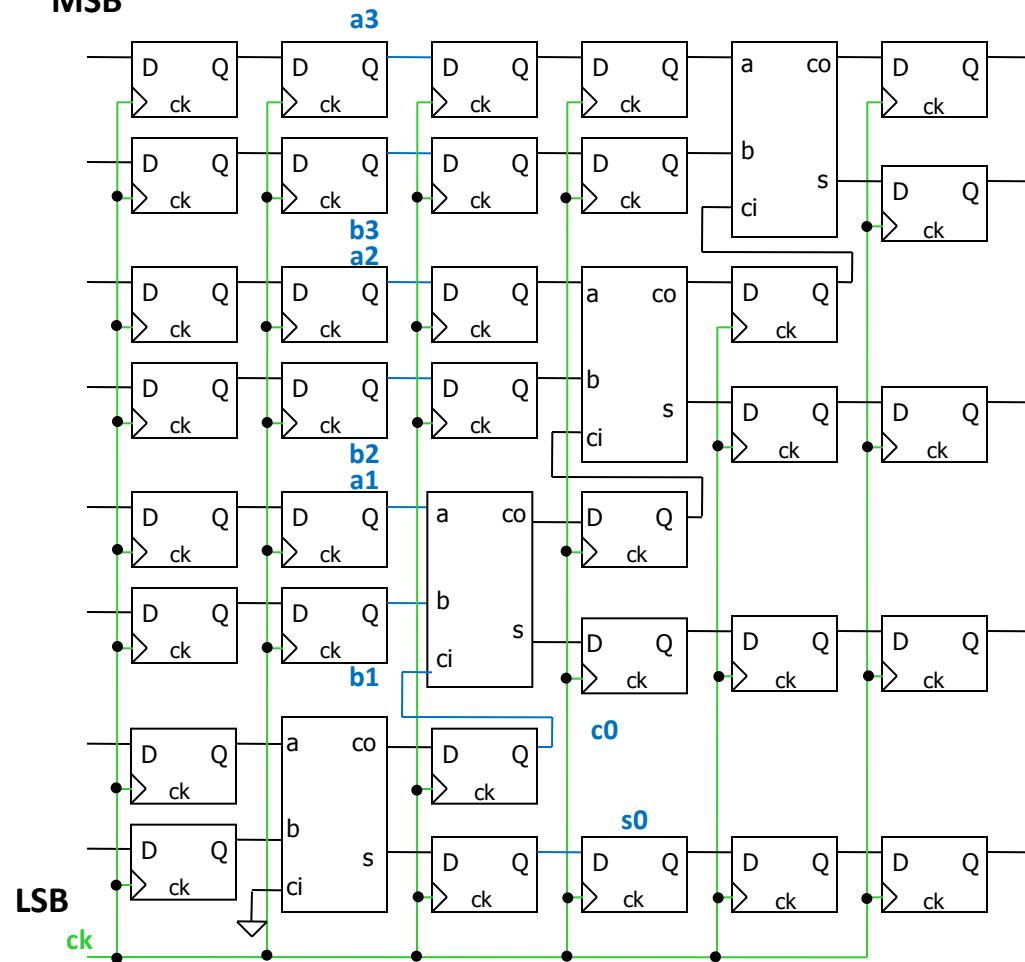
1クロック後



パイプライン加算器の動作

2クロック後

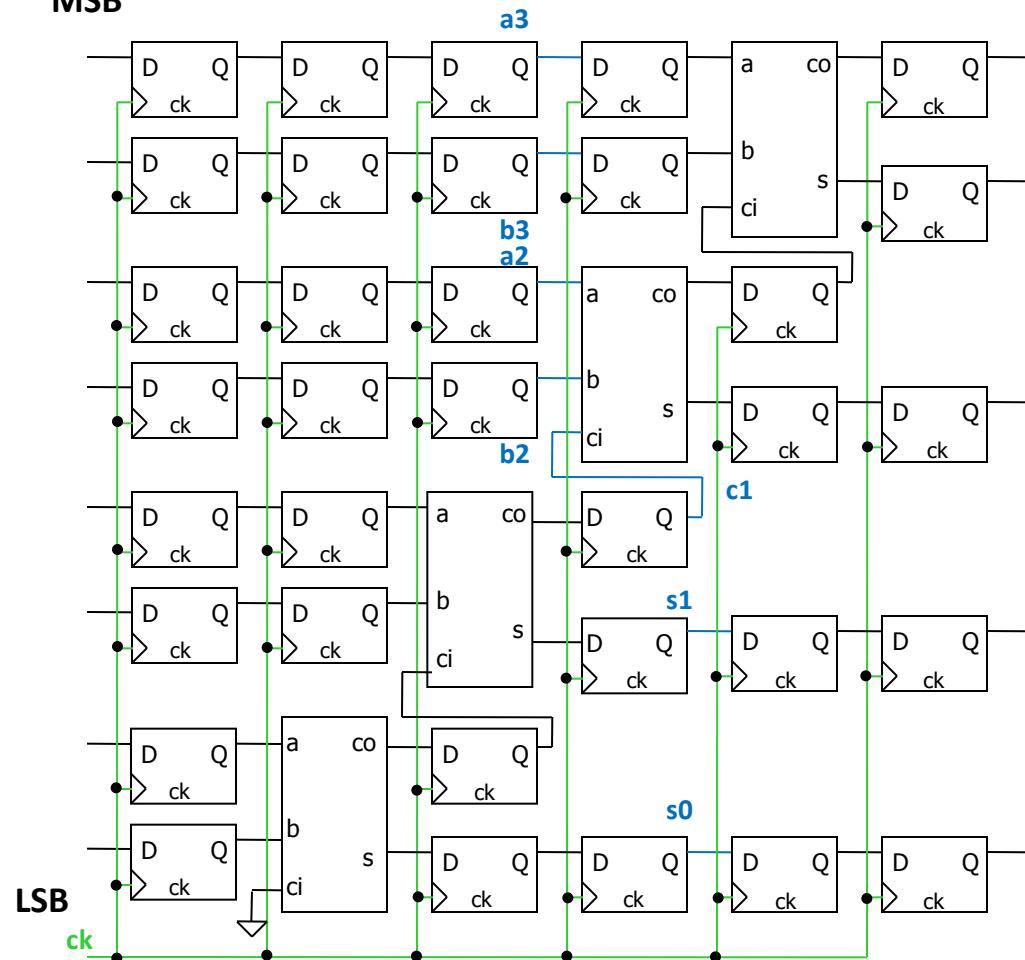
MSB



パイプライン加算器の動作

3クロック後

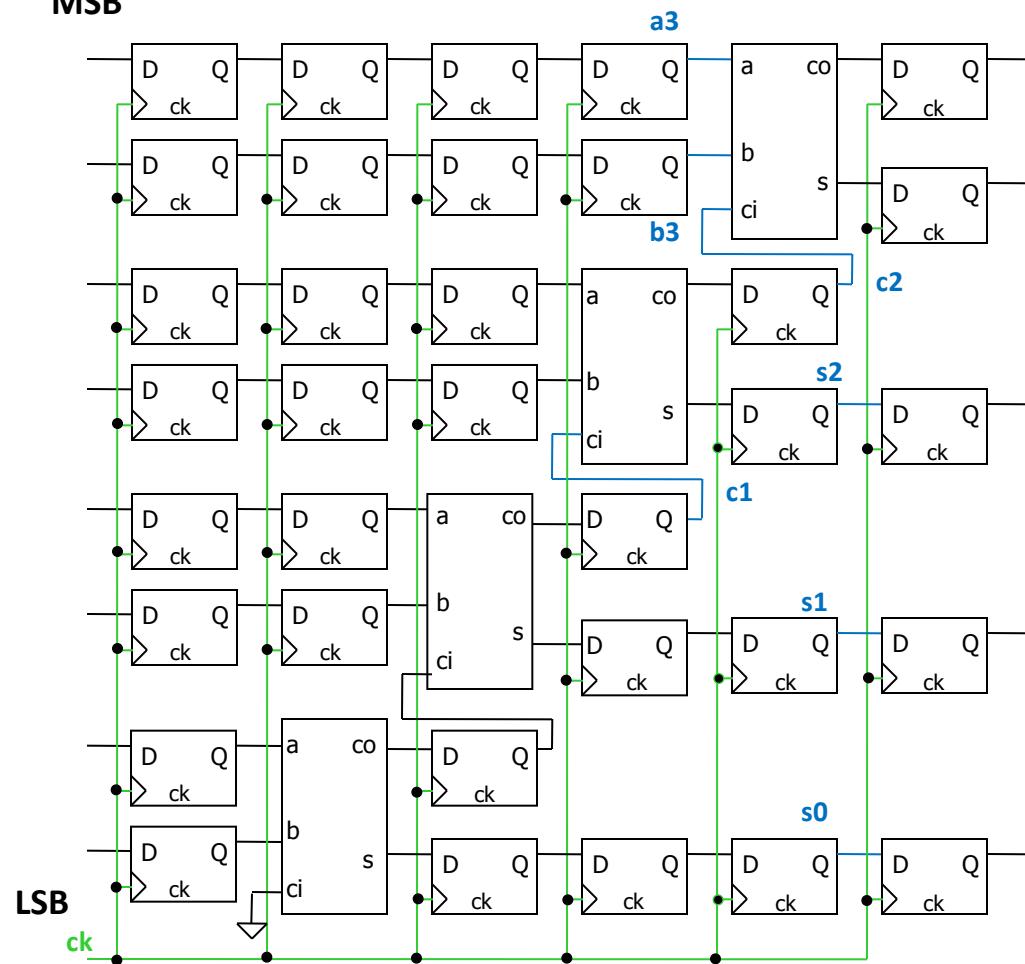
MSB



パイプライン加算器の動作

4クロック後

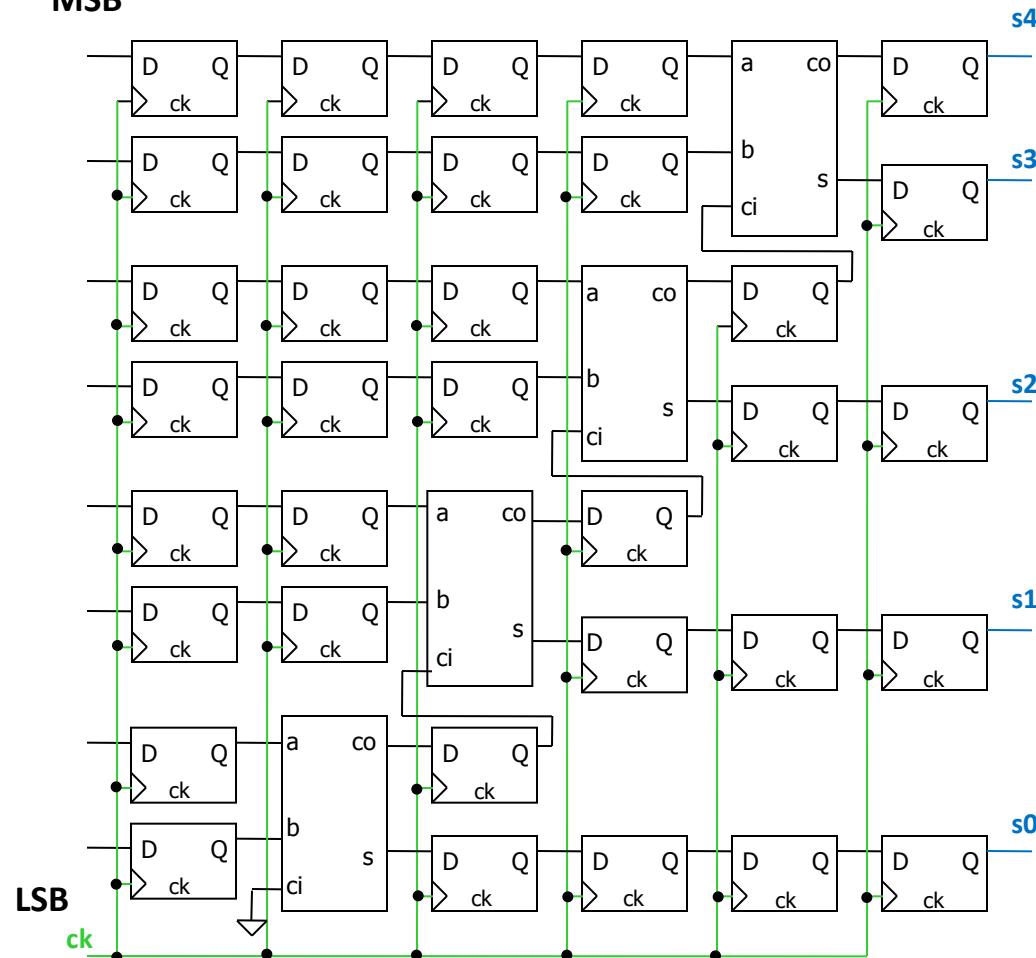
MSB



パイプライン加算器の動作

5クロック後

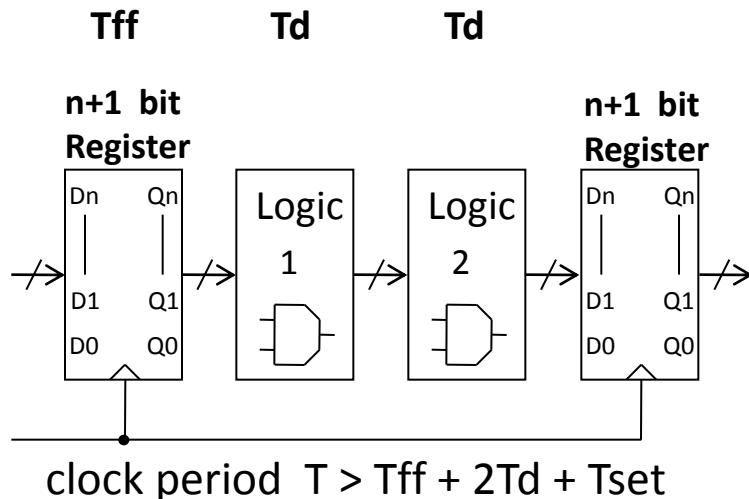
MSB



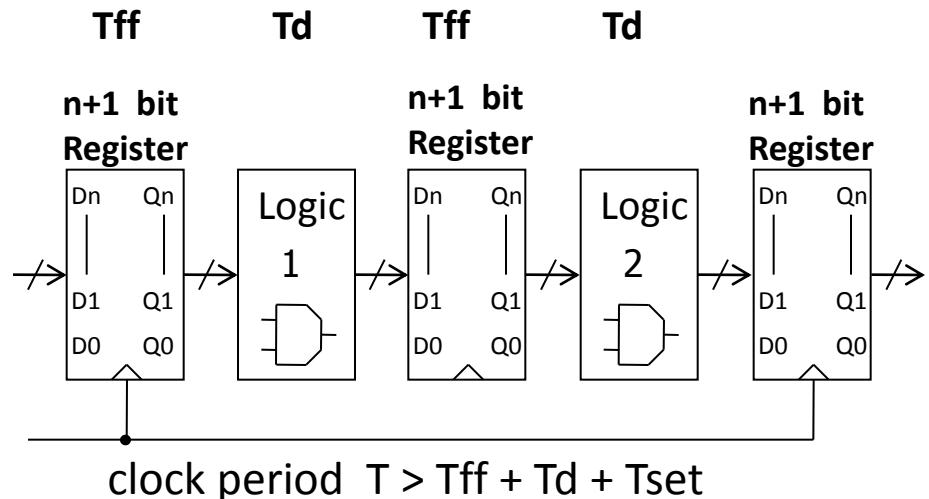
パイプライン構成による高速化

Pipeline Circuit

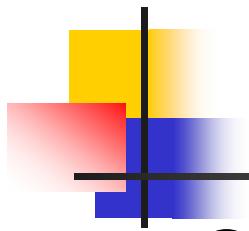
● Circuit 1



● Circuit 2



$T_d \gg T_{ff} + T_{set}$ → 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

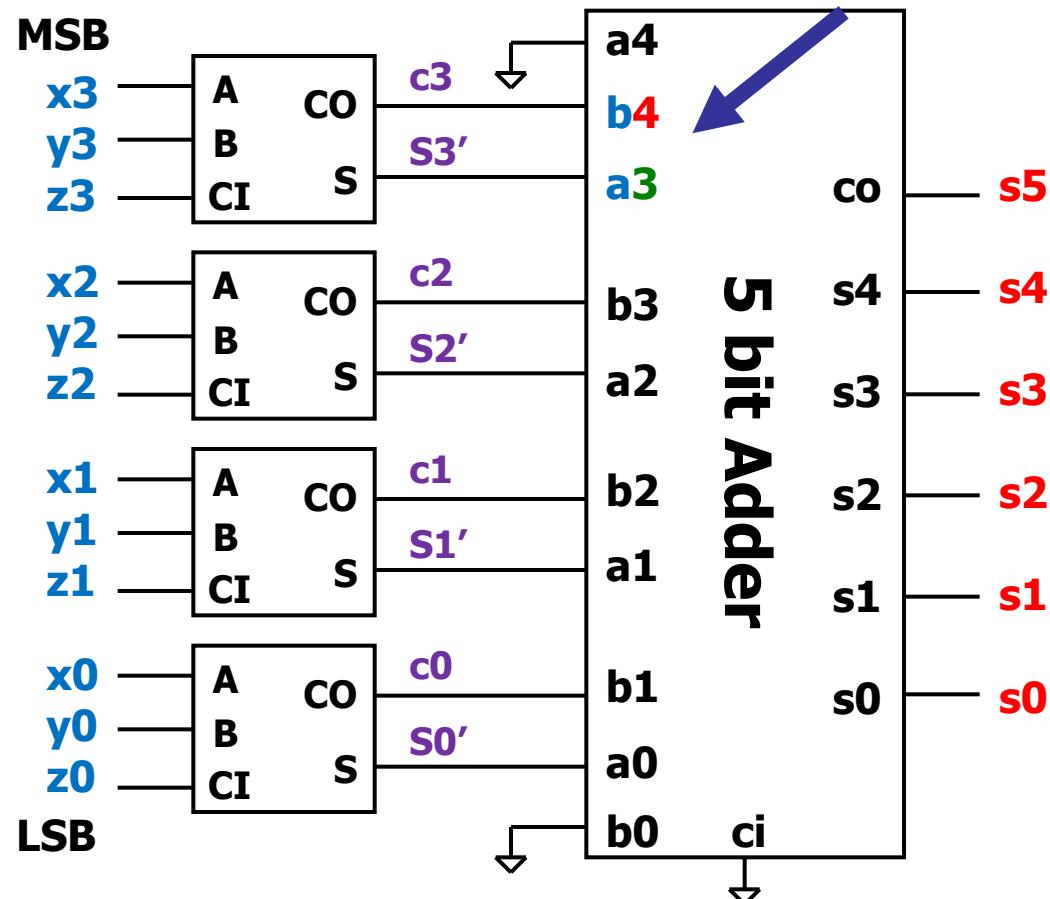
Carry Save Adder

3入力加算器

- 多入力加算
- 最後に1回だけ
桁上げ計算
- 高速
- 回路規模 小

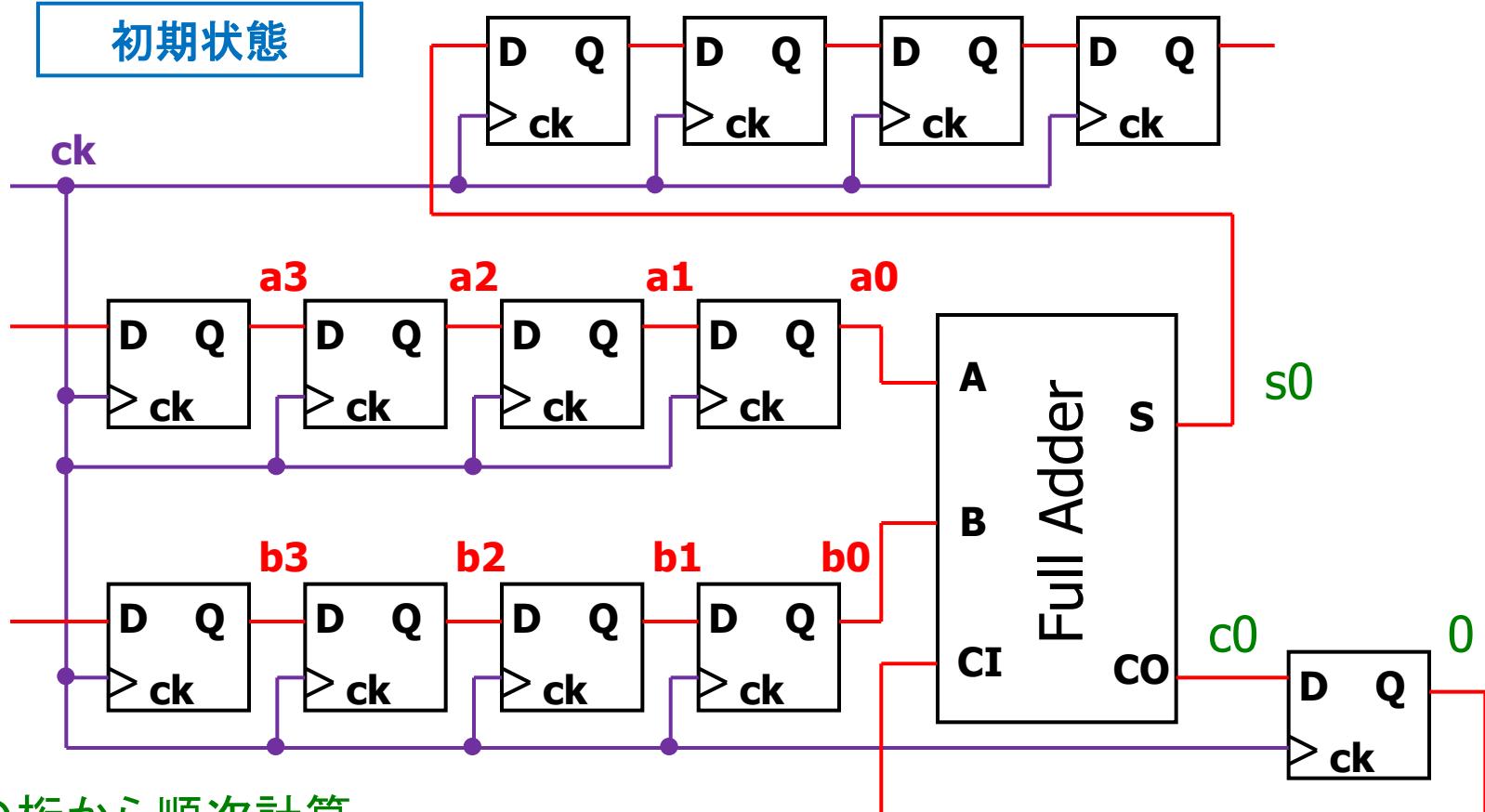
$$S = X + Y + Z$$

co は b_{n+1}
s は a_n
に入力



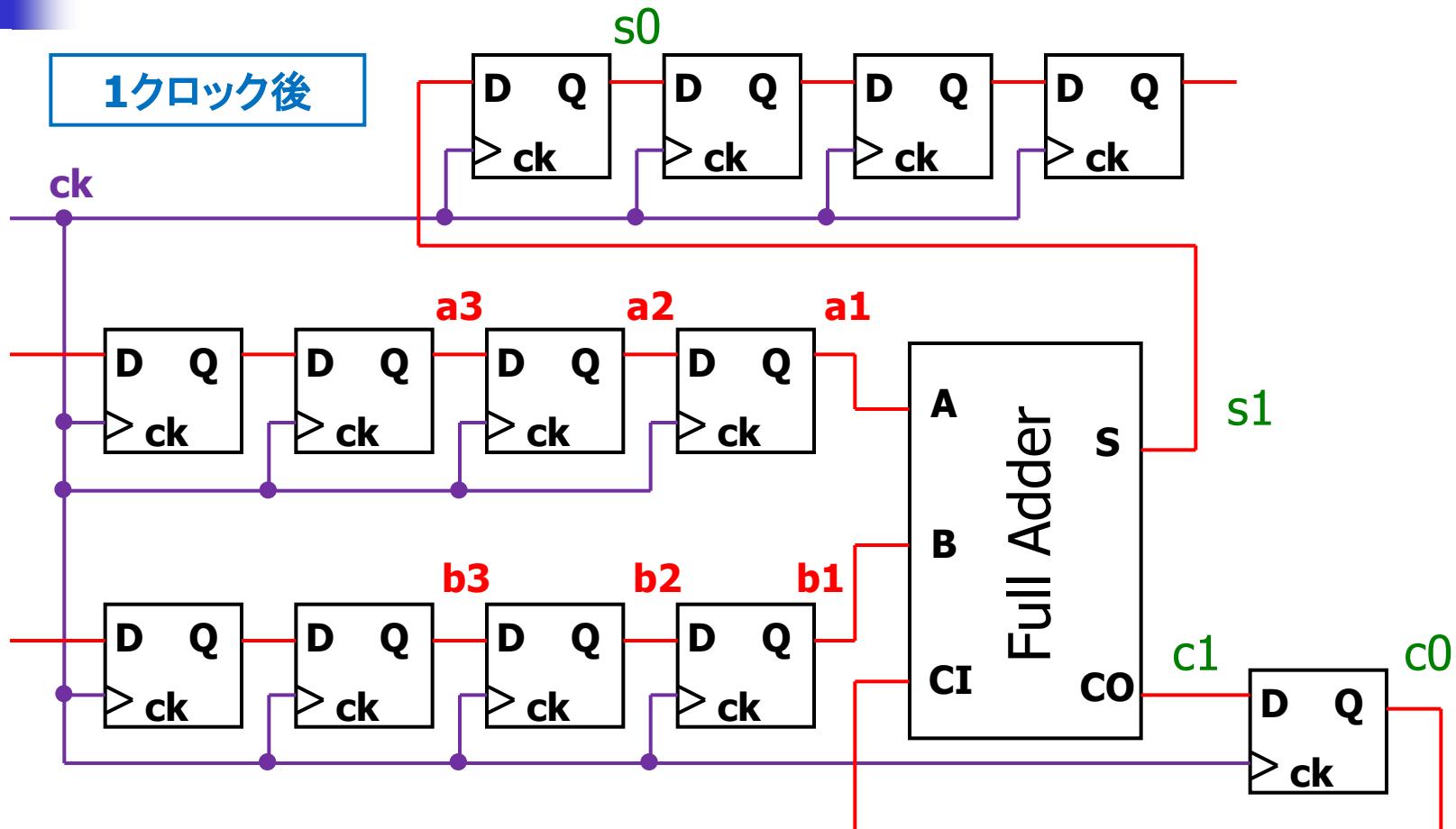
ビットシリアル加算器の構成

全加算器は1個でよい

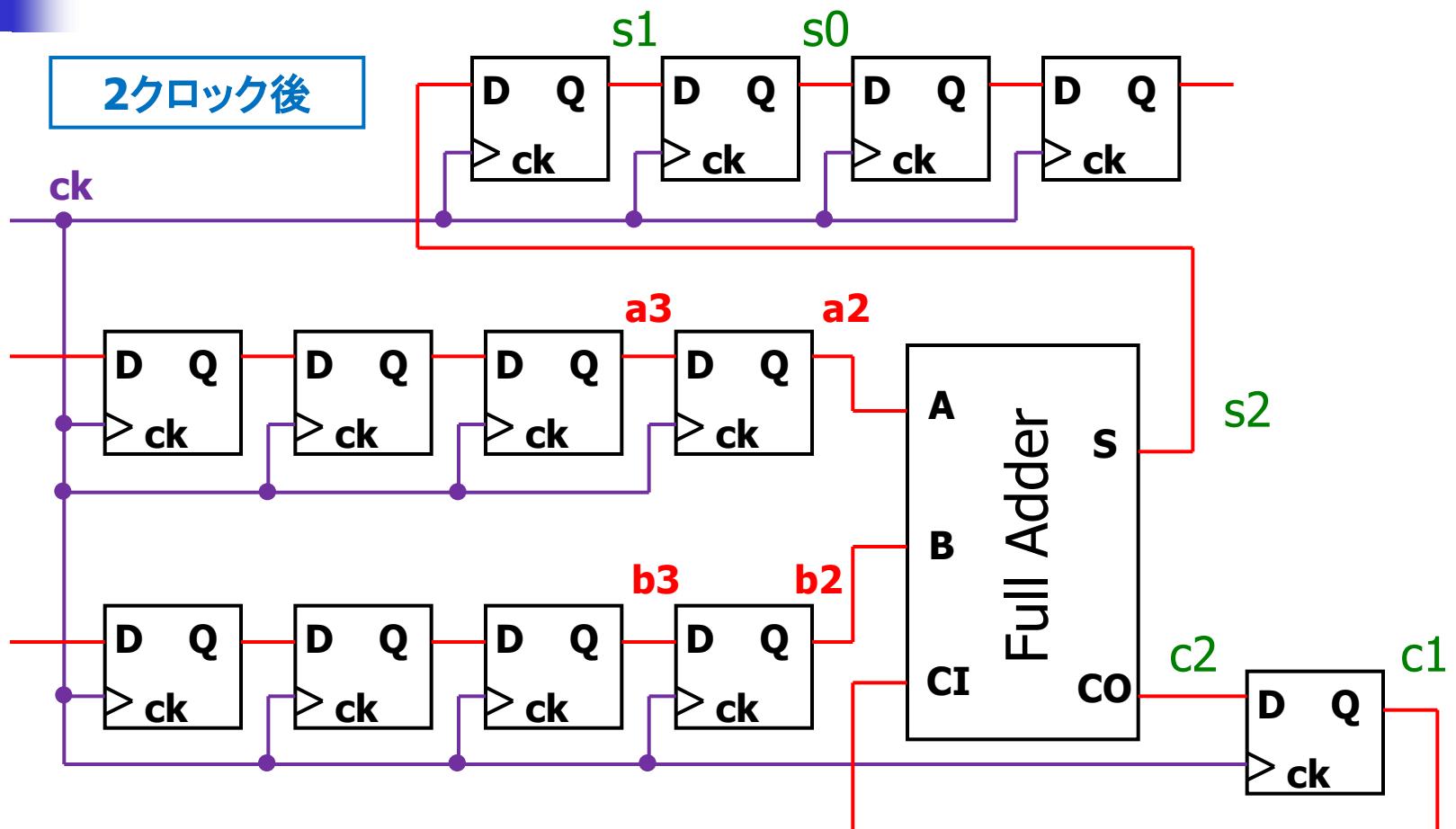


下の桁から順次計算
加算演算に桁数だけクロック数が必要

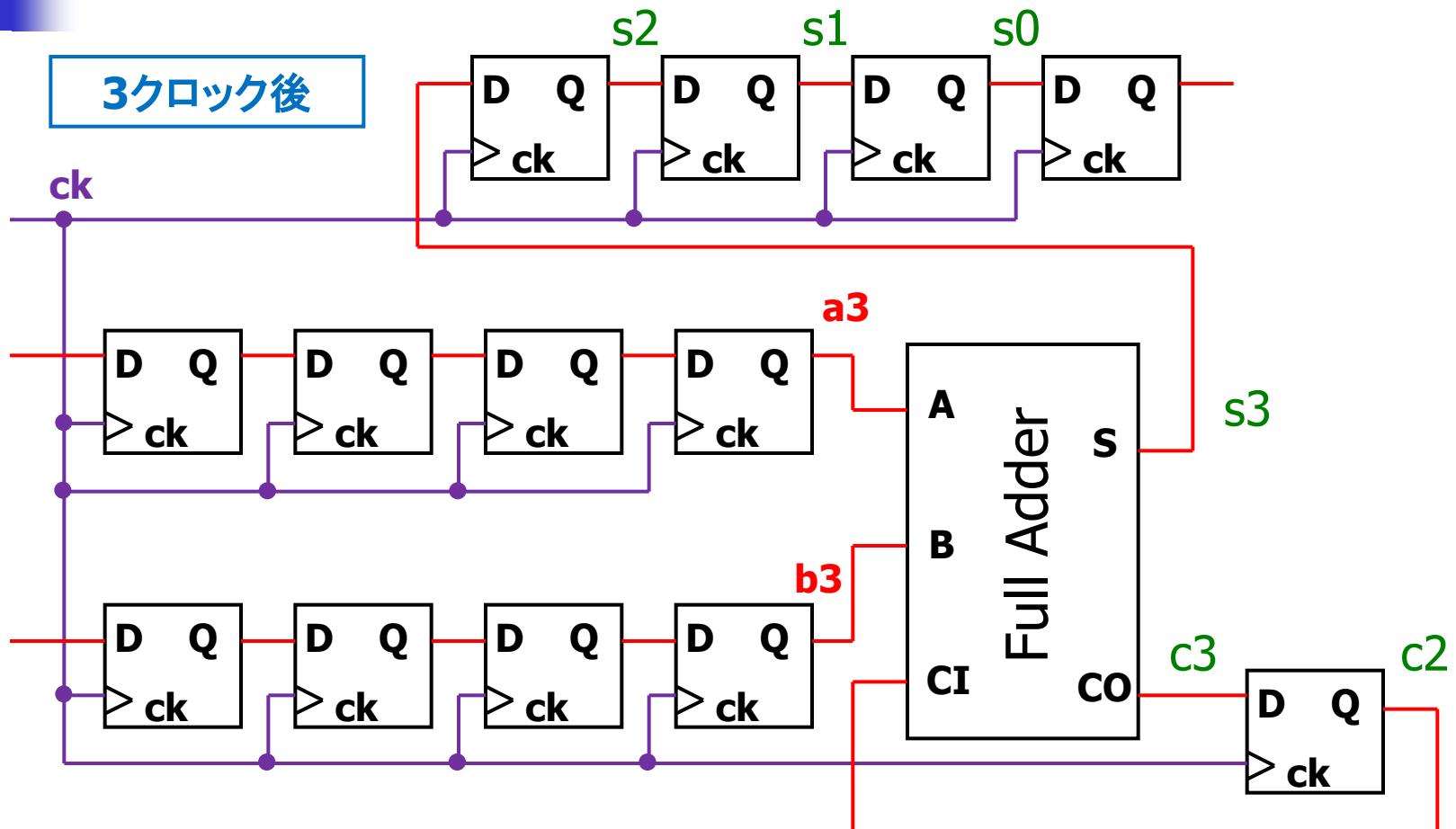
ビットシリアル加算器の動作



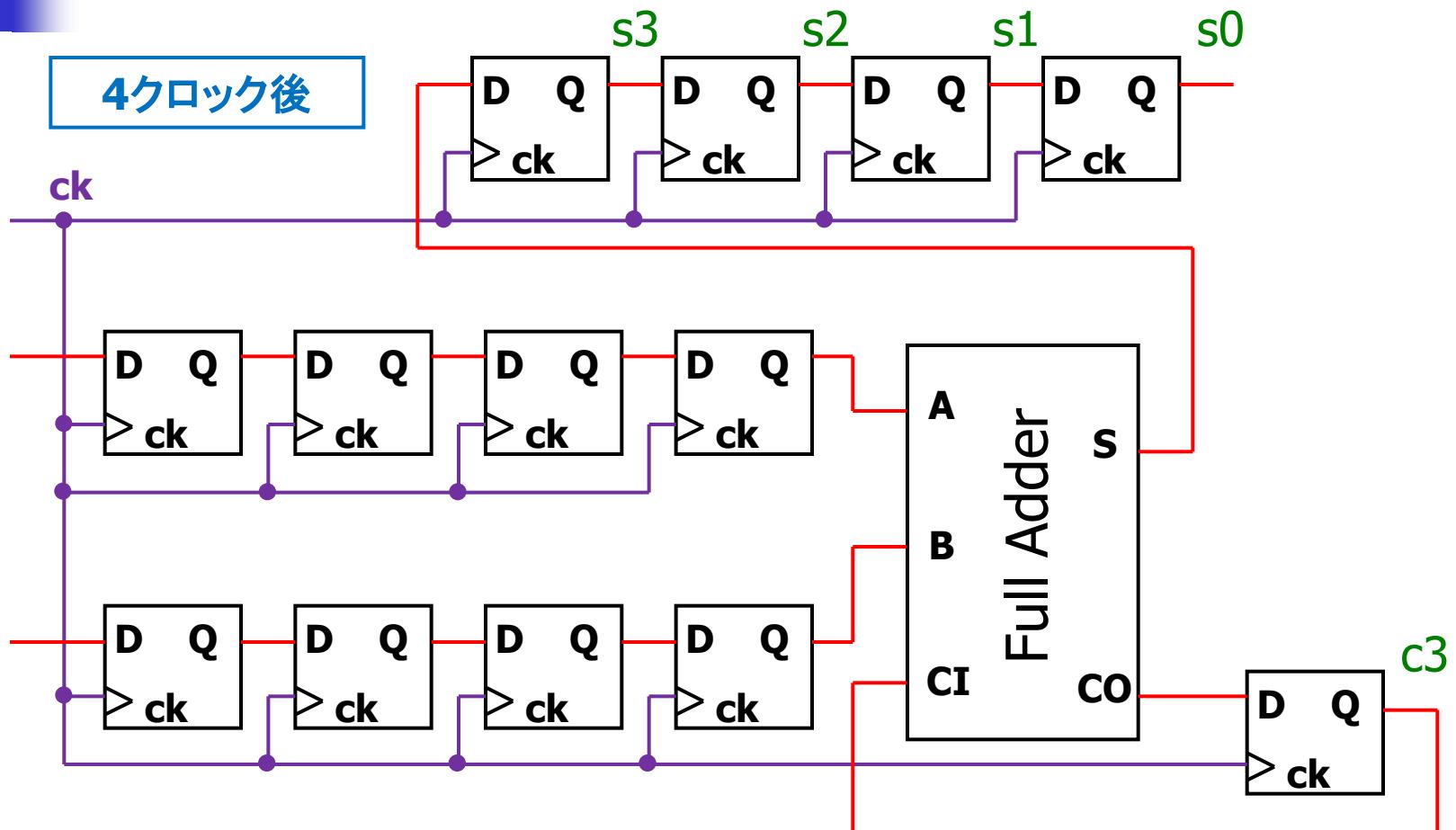
ビットシリアル加算器の動作



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 $\times 10, \times 100, \div 1000$ 等は簡単
同様に2進数で $\times 2, \times 4, \div 8$ 等は簡単

| ● 1 bit left shift | \leftrightarrow | $\times 2$ | b4 b3 b2 b1 b0 | 十進数 |
|---------------------|-------------------|---------------------|----------------|-----|
| b4 b3 b2 b1 b0 | | | 0 0 0 0 0 | 0 |
| 0 0 0 1 1 | | 3 | 0 0 0 0 1 | 1 |
| ↓ ↓ ↓ ↓ ↓ | | | 0 0 0 1 0 | 2 |
| 0 0 1 1 0 | | $3 \times 2 = 6$ | 0 0 1 0 0 | 4 |
| ↓ ↓ ↓ ↓ ↓ | | | 0 0 1 0 1 | 5 |
| 0 1 1 0 0 | | $6 \times 2 = 12$ | 0 0 1 1 0 | 6 |
| ↓ ↓ ↓ ↓ ↓ | | | 0 0 1 1 1 | 7 |
| 1 1 1 0 1 | | -3 | 0 1 0 0 0 | 8 |
| ↓ ↓ ↓ ↓ ↓ | | | 0 1 0 0 1 | 9 |
| 1 1 0 1 0 | | $-3 \times 2 = -6$ | 0 1 0 1 0 | 10 |
| ↓ ↓ ↓ ↓ ↓ | | | 0 1 0 1 1 | 11 |
| 1 0 1 0 0 | | $-6 \times 2 = -12$ | 0 1 1 0 0 | 12 |
| ↓ ↓ ↓ ↓ ↓ | | | 0 1 1 0 1 | 13 |
| 1 0 1 0 0 | | | 0 1 1 1 0 | 14 |
| ● 1 bit right shift | \leftrightarrow | $\div 2$ | b4 b3 b2 b1 b0 | 十進数 |
| b4 b3 b2 b1 b0 | | | 0 1 1 1 1 | 15 |
| 0 1 0 0 0 | | 8 | 1 0 0 0 0 | -16 |
| ↓ ↓ ↓ ↓ ↓ | | | 1 0 0 0 1 | -15 |
| 0 0 1 0 0 | | $8 / 2 = 4$ | 1 0 0 1 0 | -14 |
| ↓ ↓ ↓ ↓ ↓ | | | 1 0 0 1 1 | -13 |
| 0 0 0 1 0 | | $4 / 2 = 2$ | 1 0 1 0 0 | -12 |
| 1 1 0 0 0 | | | 1 0 1 0 1 | -11 |
| ↓ ↓ ↓ ↓ ↓ | | | 1 0 1 1 0 | -10 |
| 1 1 0 0 0 | | -8 | 1 0 1 1 1 | -9 |
| ↓ ↓ ↓ ↓ ↓ | | | 1 1 0 0 0 | -8 |
| 1 1 1 0 0 | | $-8 / 2 = -4$ | 1 1 0 0 1 | -7 |
| ↓ ↓ ↓ ↓ ↓ | | | 1 1 0 1 0 | -6 |
| 1 1 1 1 0 | | $-4 / 2 = -2$ | 1 1 0 1 1 | -5 |
| 1 1 1 1 0 | | | 1 1 1 0 0 | -4 |
| 1 1 1 1 0 | | | 1 1 1 0 1 | -3 |
| 1 1 1 1 0 | | | 1 1 1 1 0 | -2 |
| 1 1 1 1 0 | | | 1 1 1 1 1 | -1 |

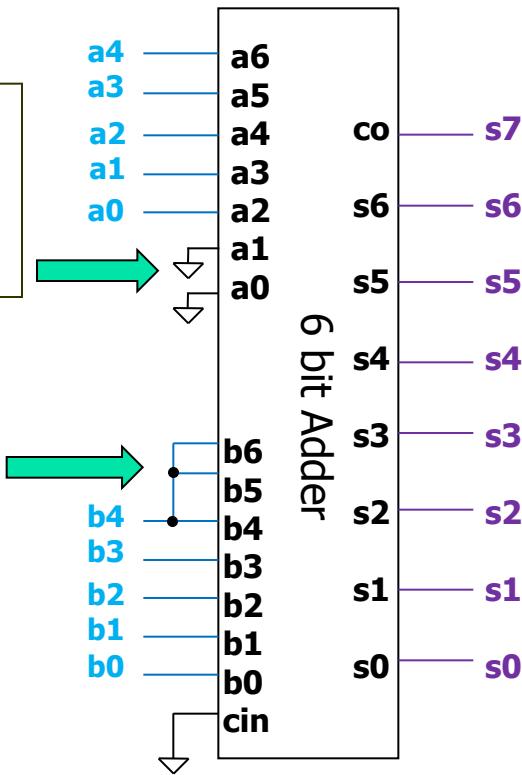
ビットシフトで 2^n 倍乗算を実現

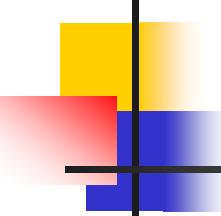
$$S = 4 \times A + B$$

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

$$\begin{array}{r} 0101 \quad (5) \\ \times 1011 \quad (11) \\ \hline 0101 \\ 0101 \\ 0000 \\ 0101 \\ \hline 0110111 \quad (55) \end{array}$$

10進数の乗算

$$\begin{array}{r} 437 \\ \times 258 \\ \hline 3496 \\ 2185 \\ 874 \\ \hline 112746 \end{array}$$

2進デジタル乗算器

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

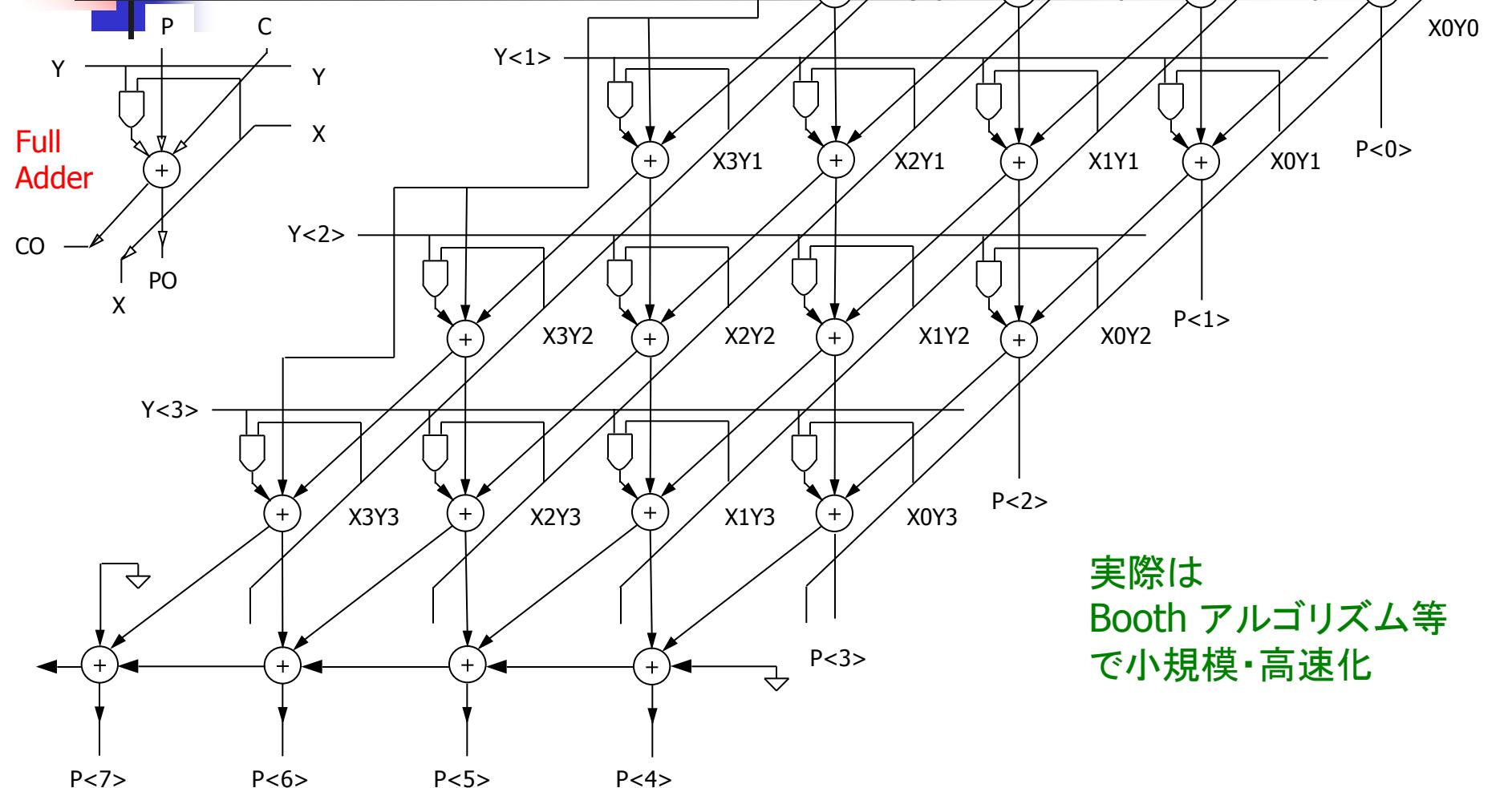
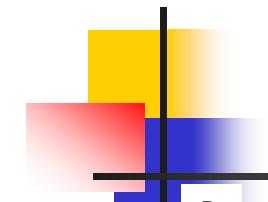
$$Y = \sum_{j=0}^{n-1} Y_j 2^j$$

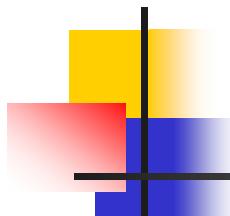
$$\begin{aligned} P &= X \times Y = \sum_{i=0}^{m-1} X_i 2^i \bullet \sum_{j=0}^{n-1} Y_j 2^j \\ &= \sum_{i=0}^{m-1} \bullet \sum_{j=0}^{n-1} Y_j 2^j (X_i Y_j) 2^{i+j} \\ &= \sum_{k=0}^{m+n-1} P_k 2^k \end{aligned}$$

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 | Product |

乗算器は全加算器の2次元配列



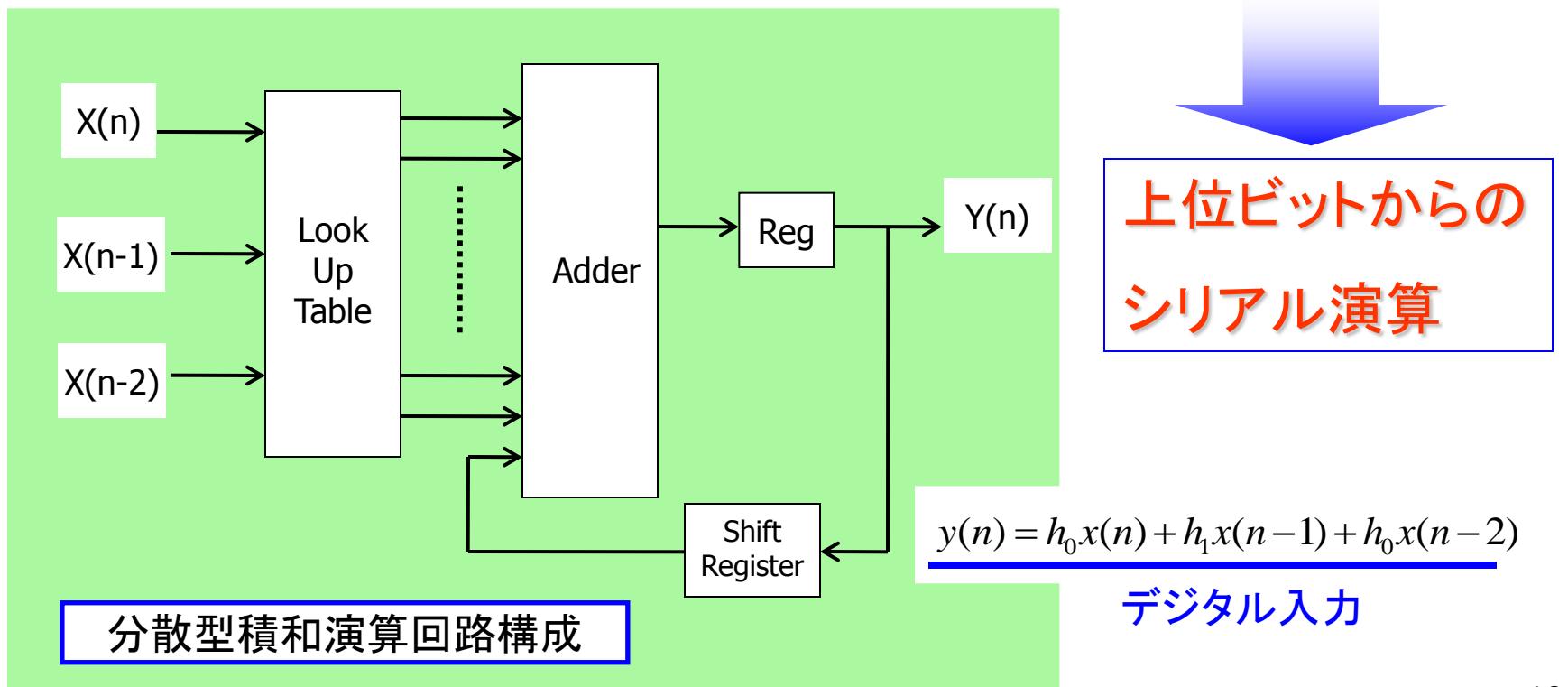


内 容

- 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
分散型積和演算 (ROM等)により実現する計算手法



分散型積和演算 動作原理

$$y(n) = h_0 \underline{x(n)} + h_1 \underline{x(n-1)} + h_0 \underline{x(n-2)}$$

MSB LSB

$x(0) = 1010 \quad \alpha$

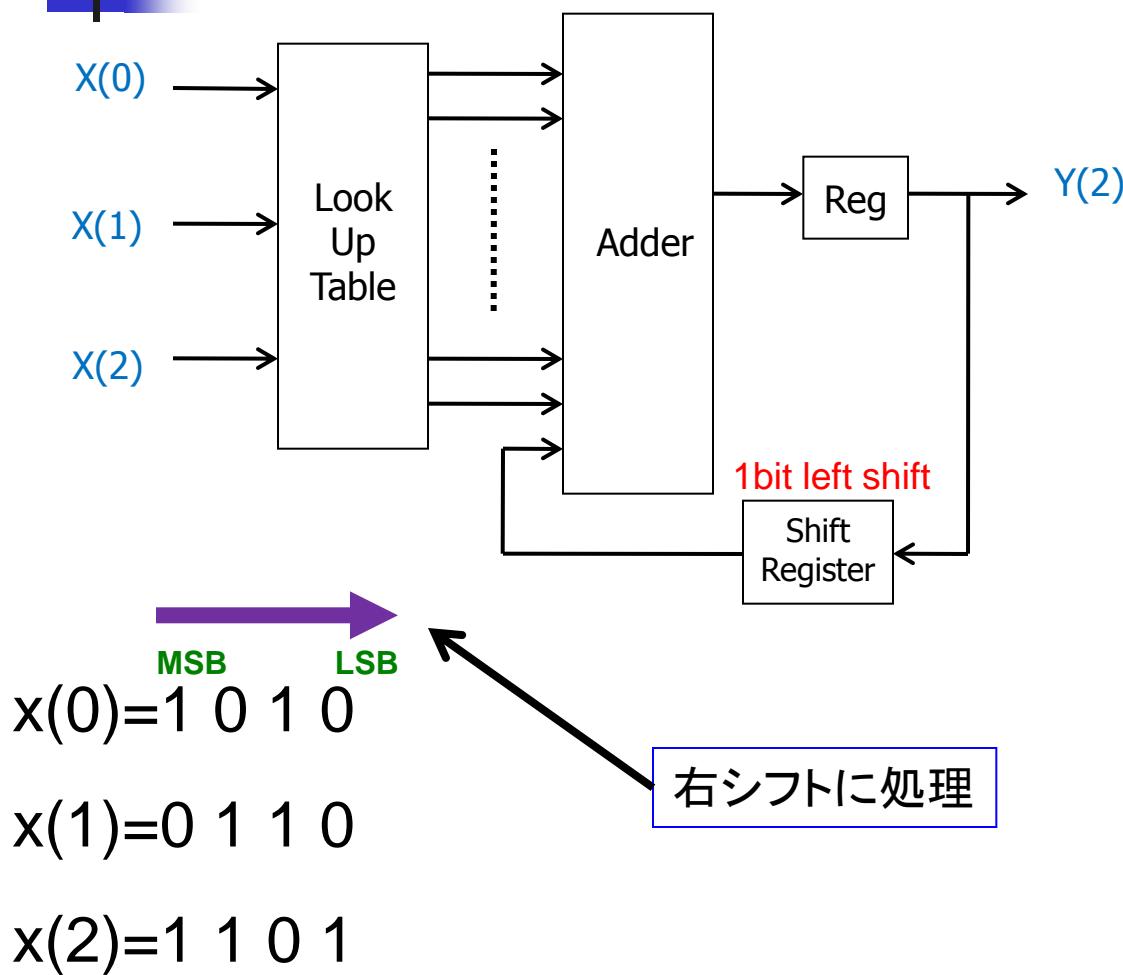
$x(1) = 0110 \quad \beta$

$x(2) = 1101 \quad \gamma$

Look Up Table

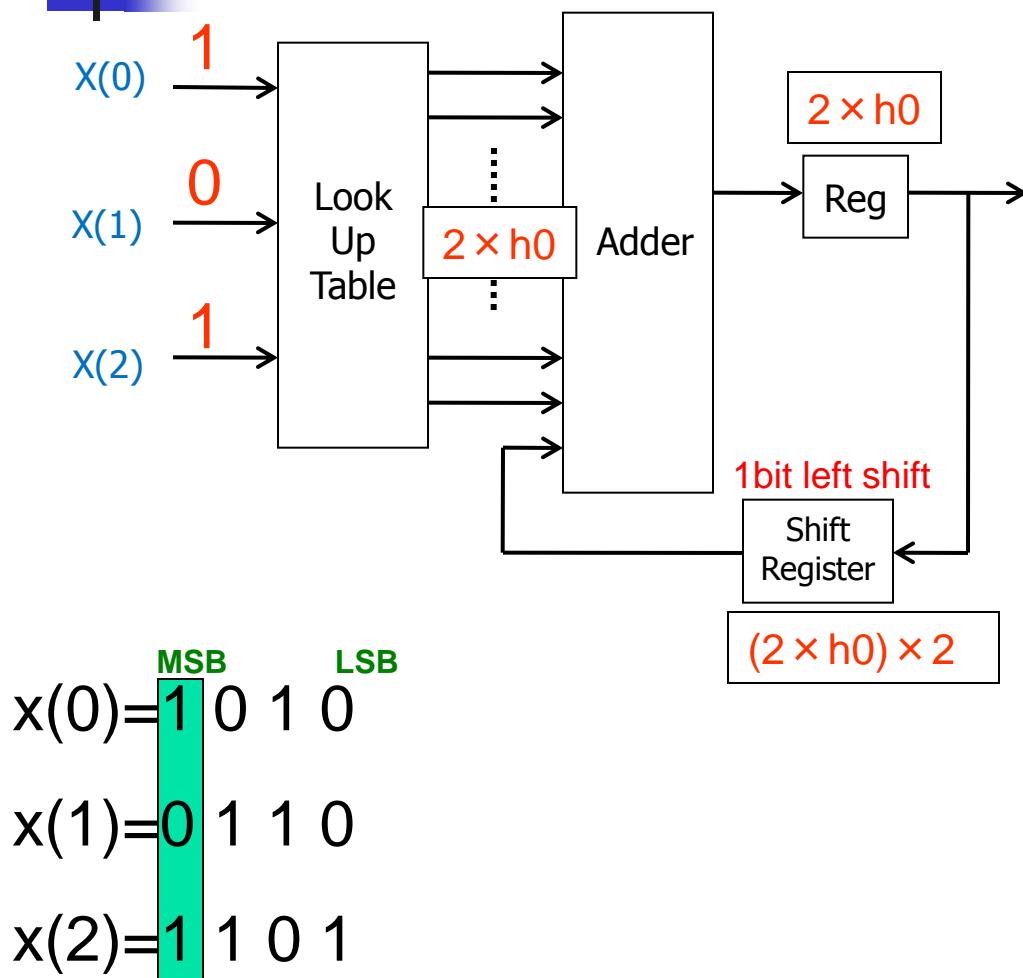
| γ | β | α | $Y(n)$ |
|----------|---------|----------|-------------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | h_0 |
| 0 | 1 | 0 | h_1 |
| 0 | 1 | 1 | h_0+h_1 |
| 1 | 0 | 0 | h_0 |
| 1 | 0 | 1 | $2*h_0$ |
| 1 | 1 | 0 | h_1+h_0 |
| 1 | 1 | 1 | $2*h_0+h_1$ |

分散型積和演算 動作原理



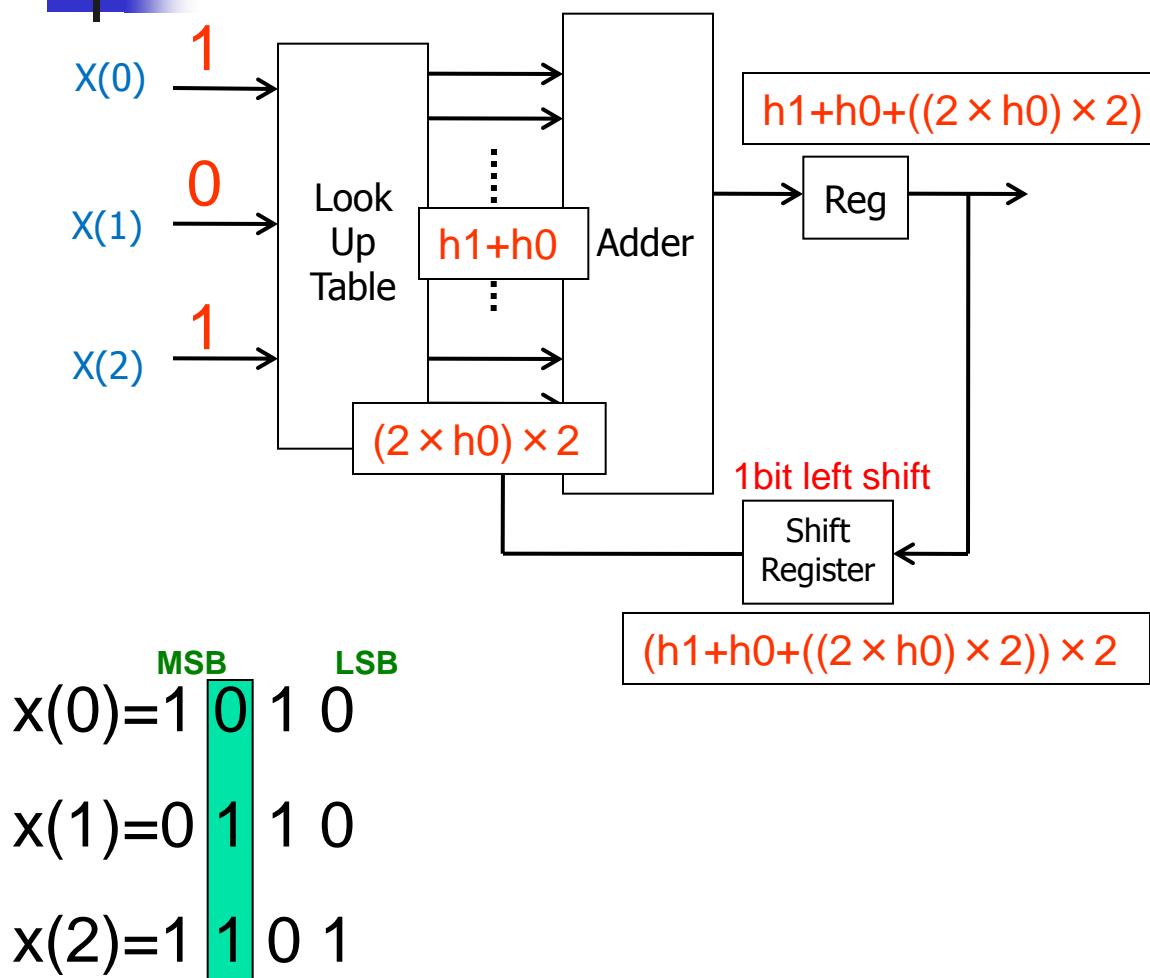
| γ | β | α | $Y(n)$ |
|----------|---------|----------|-----------------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | h_0 |
| 0 | 1 | 0 | h_1 |
| 0 | 1 | 1 | $h_0 + h_1$ |
| 1 | 0 | 0 | h_0 |
| 1 | 0 | 1 | $2 * h_0$ |
| 1 | 1 | 0 | $h_1 + h_0$ |
| 1 | 1 | 1 | $2 * h_0 + h_1$ |

分散型積和演算 動作原理



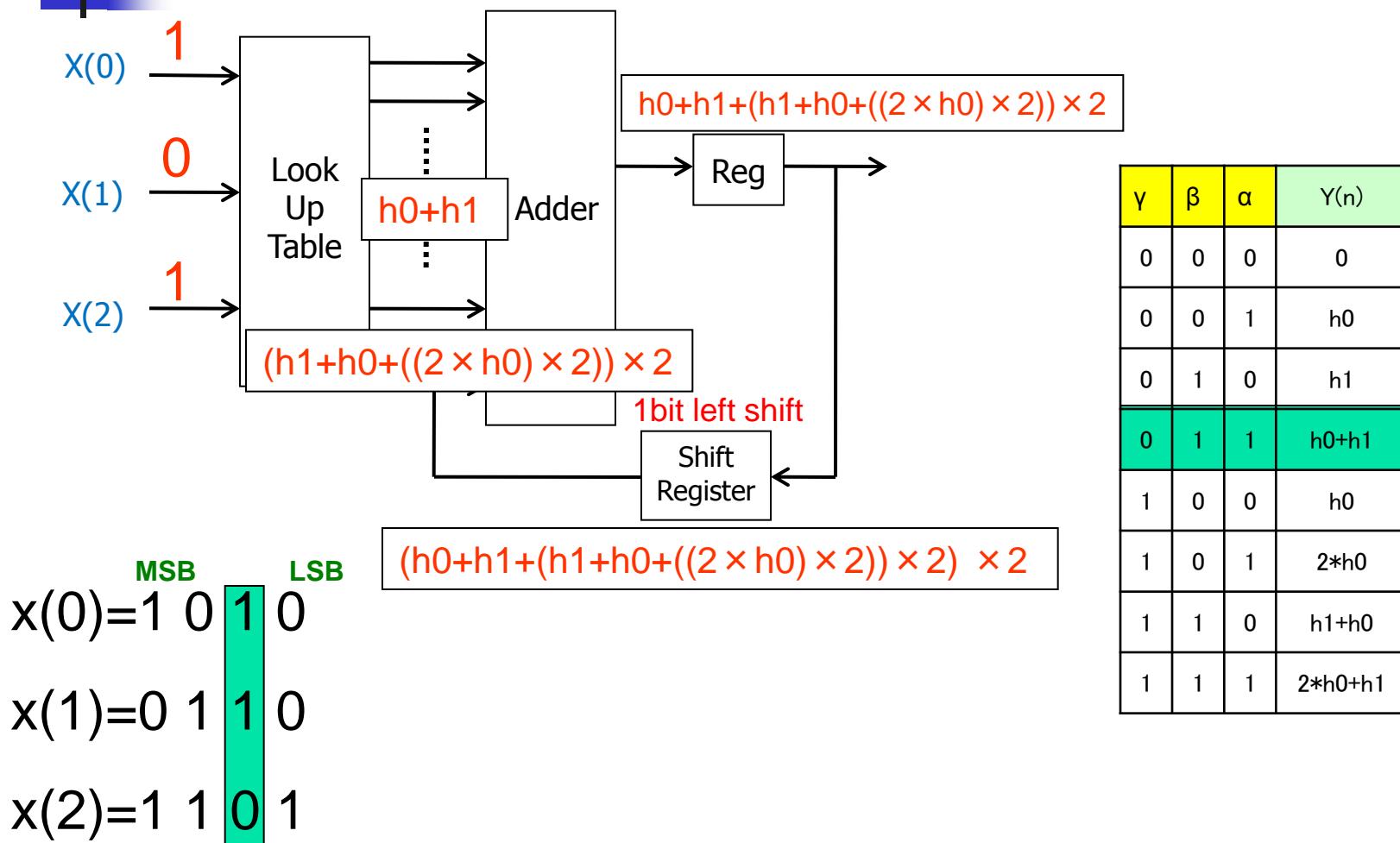
| γ | β | α | $Y(n)$ |
|----------|---------|----------|-----------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | $h0$ |
| 0 | 1 | 0 | $h1$ |
| 0 | 1 | 1 | $h0+h1$ |
| 1 | 0 | 0 | $h0$ |
| 1 | 0 | 1 | $2*h0$ |
| 1 | 1 | 0 | $h1+h0$ |
| 1 | 1 | 1 | $2*h0+h1$ |

分散型積和演算 動作原理

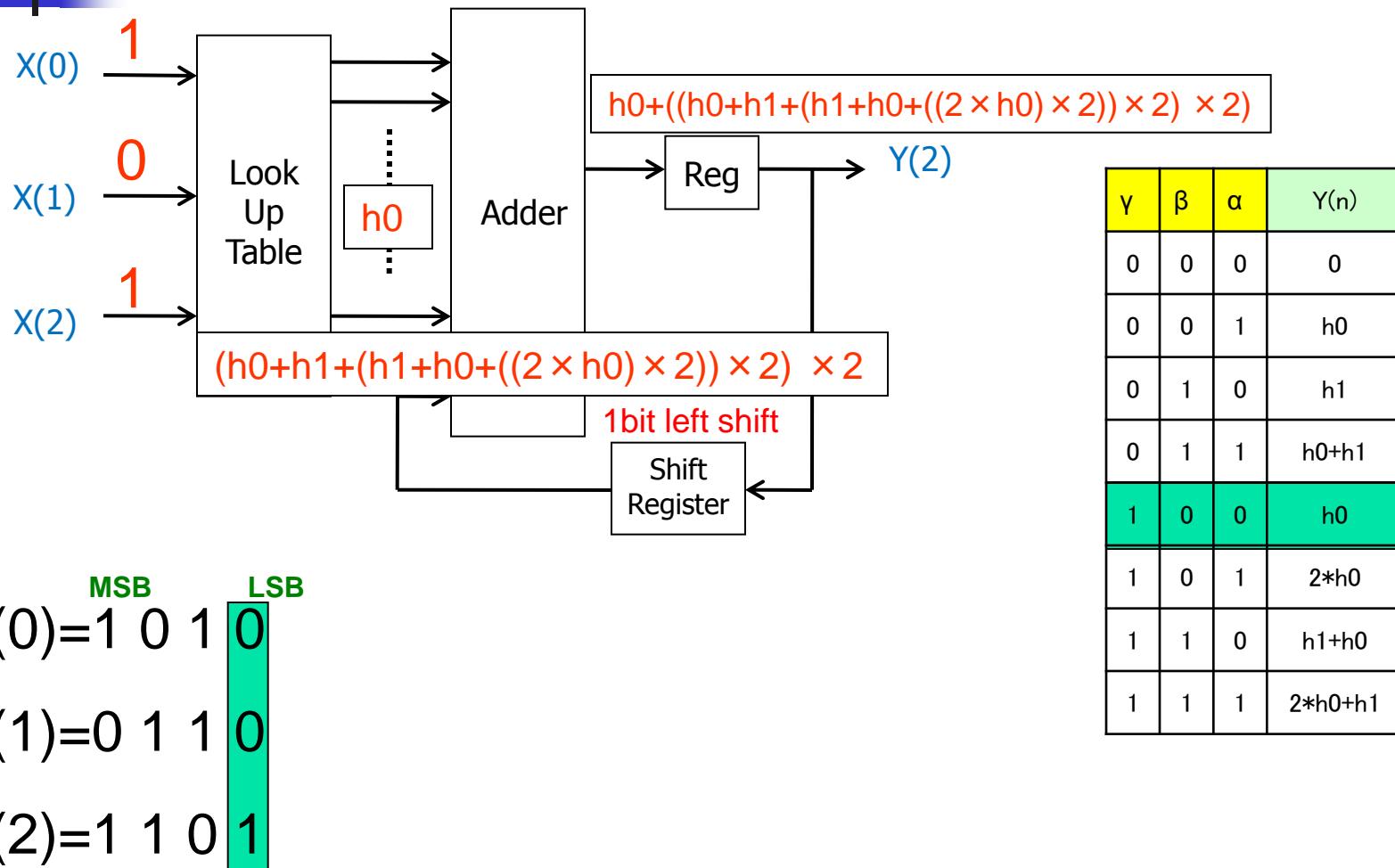


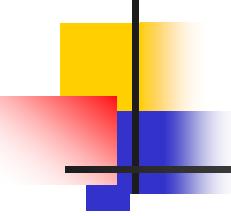
| γ | β | α | $Y(n)$ |
|----------|---------|----------|-------------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | h_0 |
| 0 | 1 | 0 | h_1 |
| 0 | 1 | 1 | h_0+h_1 |
| 1 | 0 | 0 | h_0 |
| 1 | 0 | 1 | $2*h_0$ |
| 1 | 1 | 0 | h_1+h_0 |
| 1 | 1 | 1 | $2*h_0+h_1$ |

分散型積和演算 動作原理



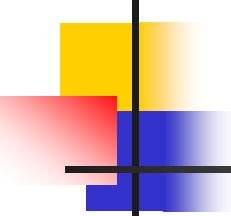
分散型積和演算 動作原理





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

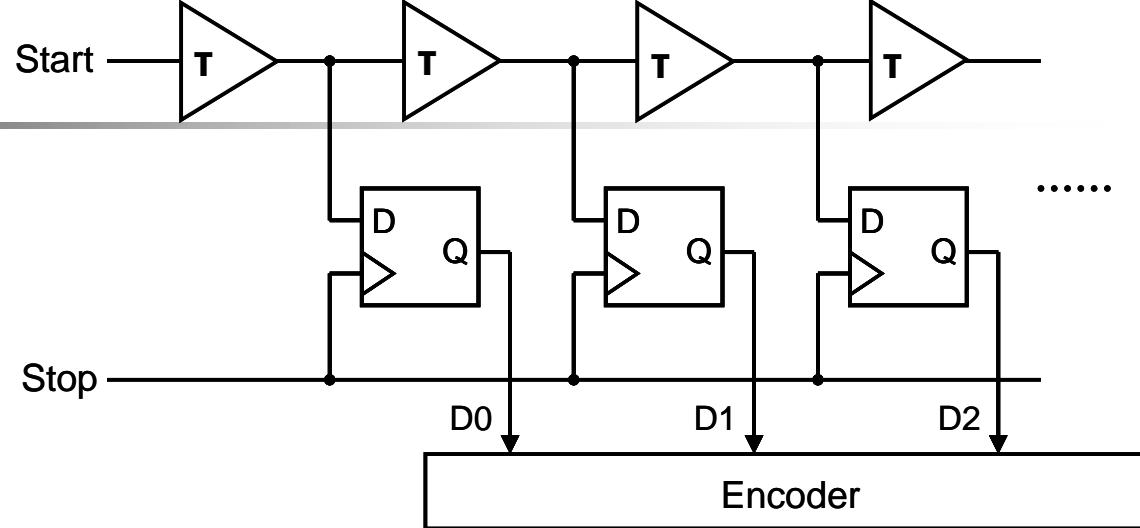
基本TDC (Time-to-Digital Converter) 回路

Start

T

Stop

Timing chart



Start

D0=1

D1=1

D2=1

D3=0

D4=0

Stop

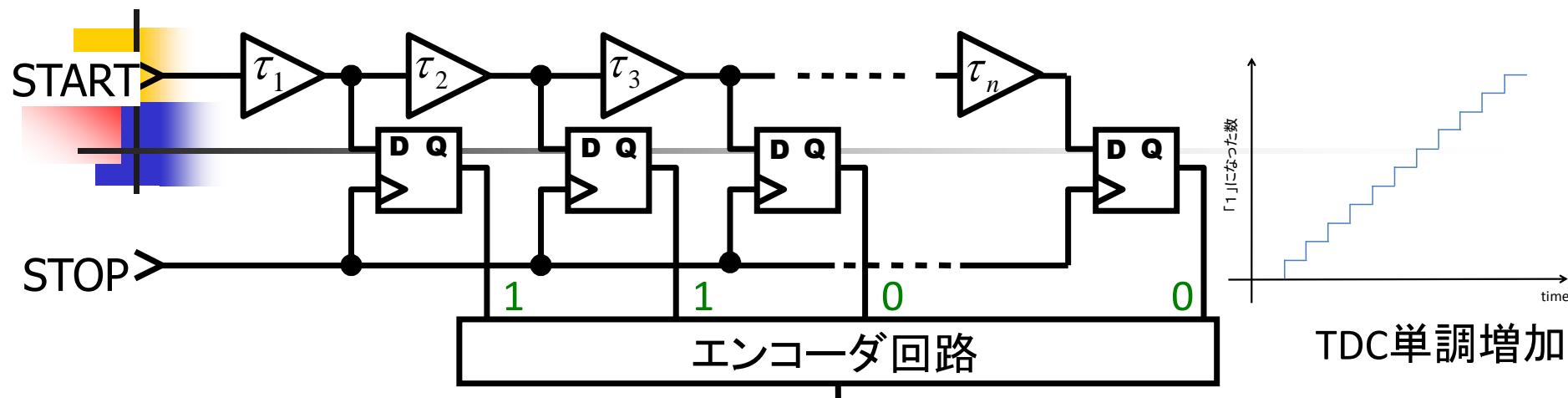
Thermometer code
 D_{out}

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



各DFF出力Dout

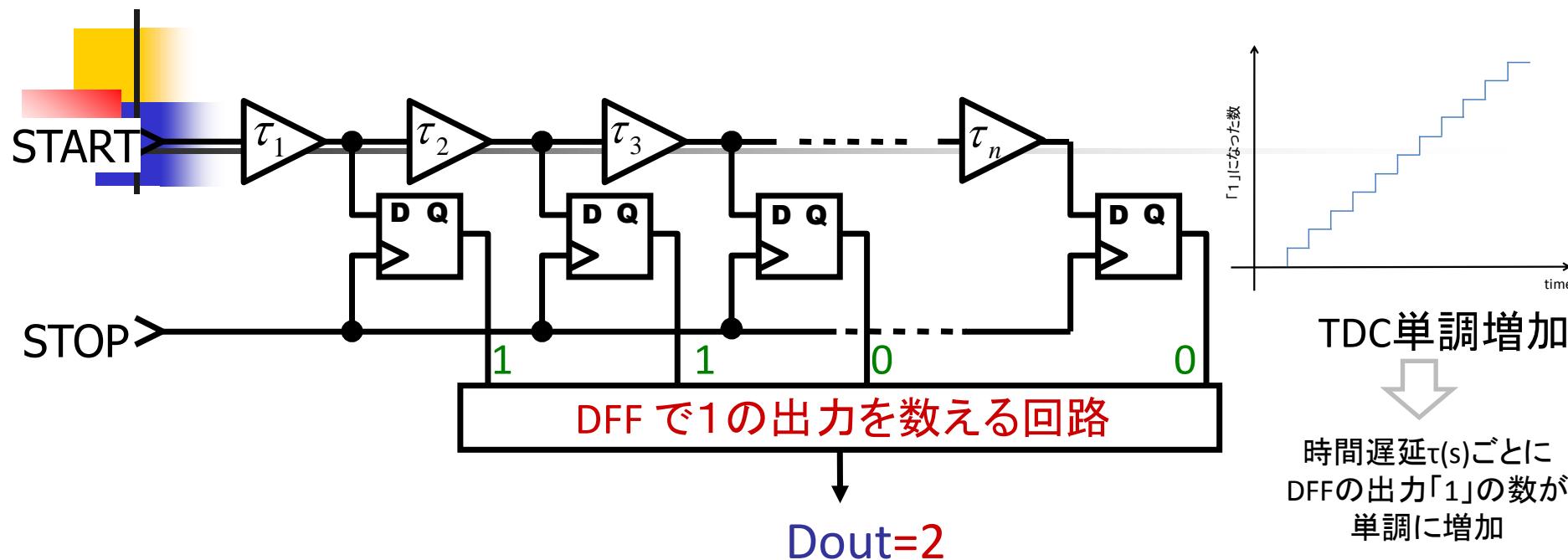
| | |
|-----------------|---|
| 0 0 0 0 0 0 0 0 | 0 |
| 1 0 0 0 0 0 0 0 | 1 |
| 1 1 0 0 0 0 0 0 | 2 |
| 1 1 1 0 0 0 0 0 | 3 |
| 1 1 1 1 0 0 0 0 | 4 |
| 1 1 1 1 1 0 0 0 | 5 |
| 1 1 1 1 1 1 0 0 | 6 |
| 1 1 1 1 1 1 1 0 | 7 |
| 1 1 1 1 1 1 1 1 | 8 |

Dout=2

| DFF出力のバブルエラー | | バッファの遅延ばらつき |
|-----------------|---|---------------|
| 1 0 1 0 0 0 0 0 | 2 | DFFのオフセットばらつき |
| 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 | |

バブルエラーが発生

TDC単調性の確保



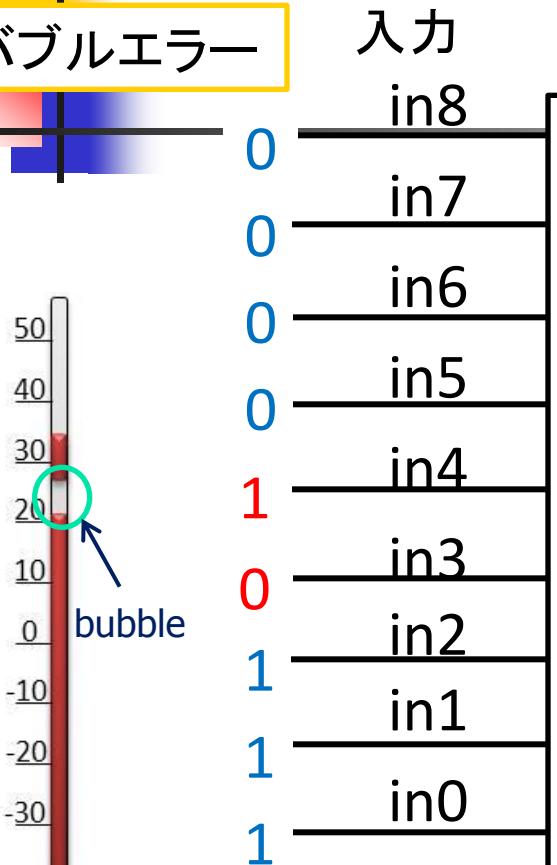
エンコーダ回路をDFFの出力が1のものを数える回路



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



1の出力を数える回路

出力

「0100」=4

| | |
|------|---|
| out3 | 0 |
| out2 | 1 |
| out1 | 0 |
| out0 | 0 |

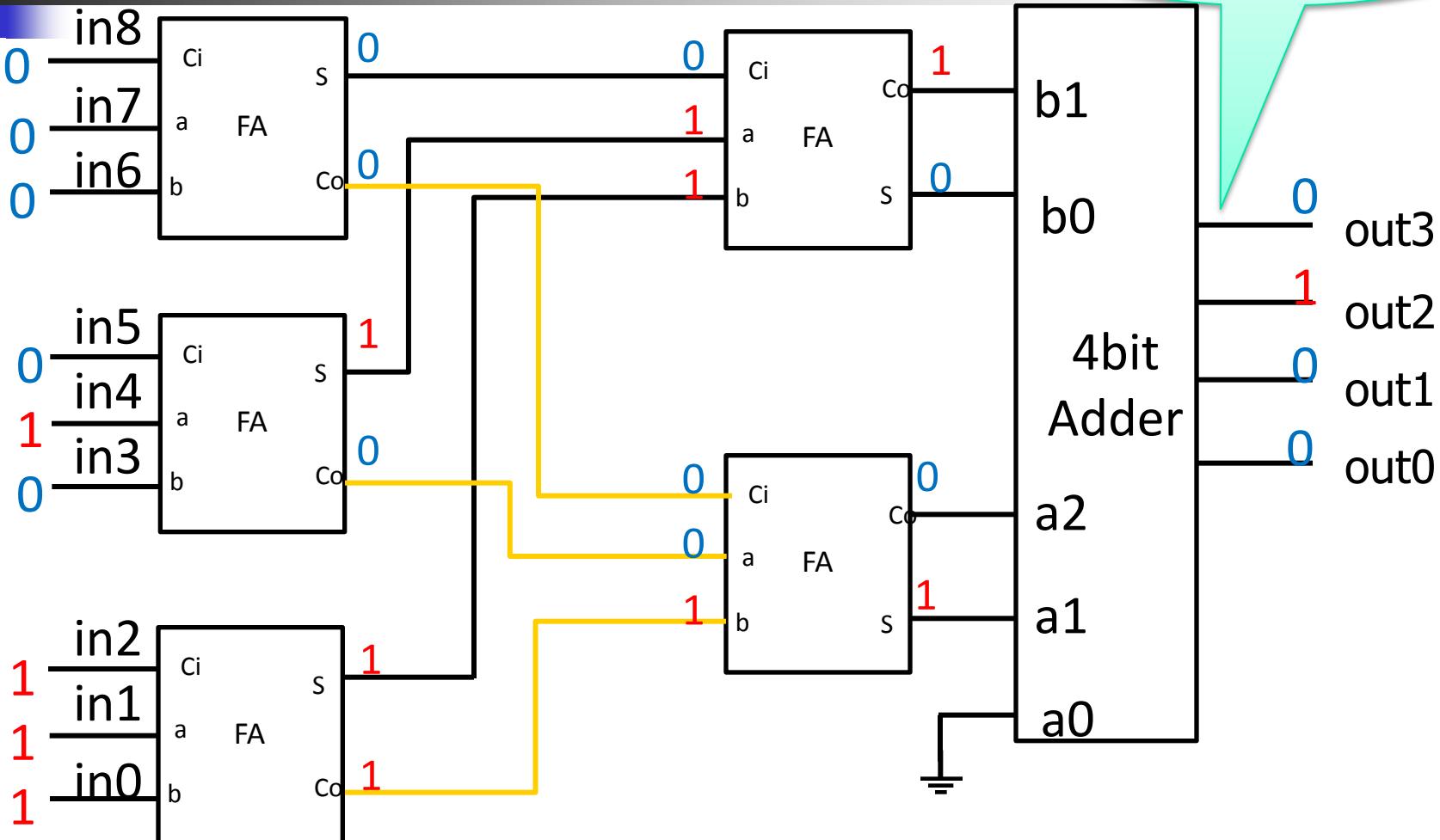


バブルエラーが起きても正しい出力が得られる

DFFの1の出力個数を数える回路

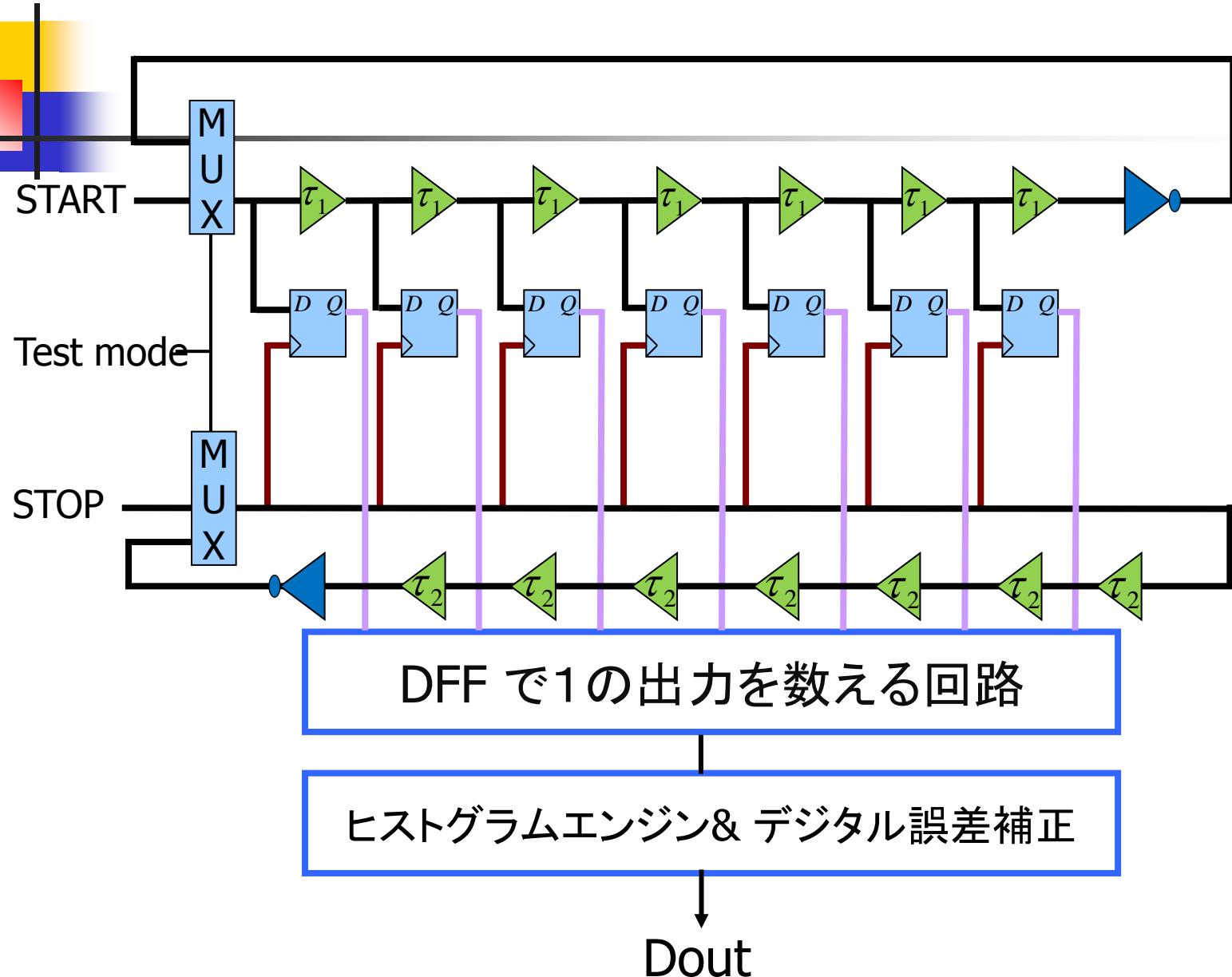
最後に1回 桁上げ計算

「0100」=4

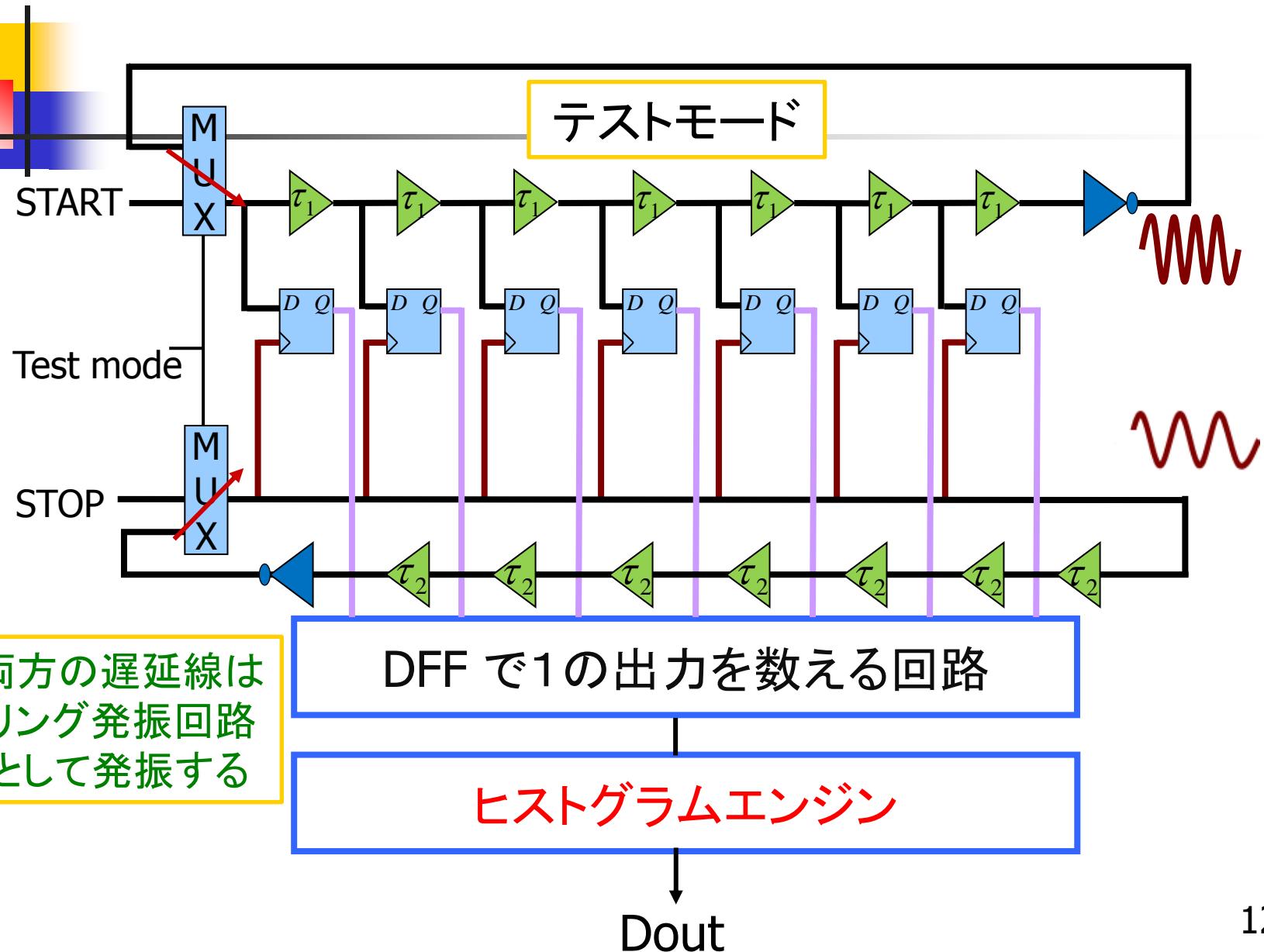


Carry Save Adder

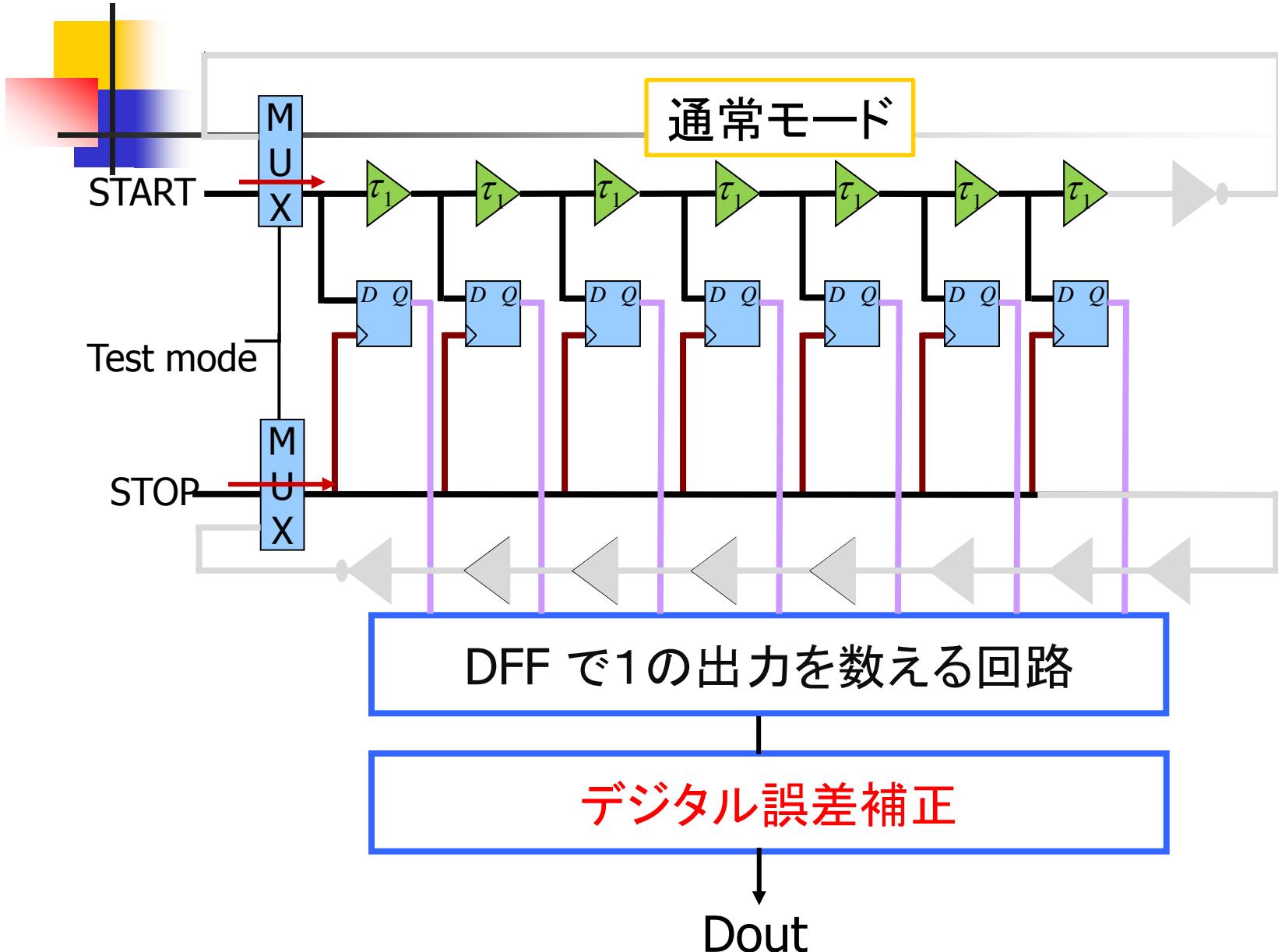
自己校正機能を備えたTDC回路の構成



自己校正機能を備えたTDC回路の構成



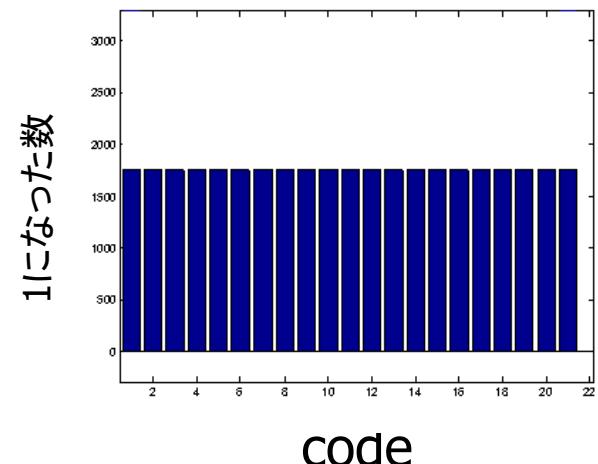
自己校正機能を備えたTDC回路の構成



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

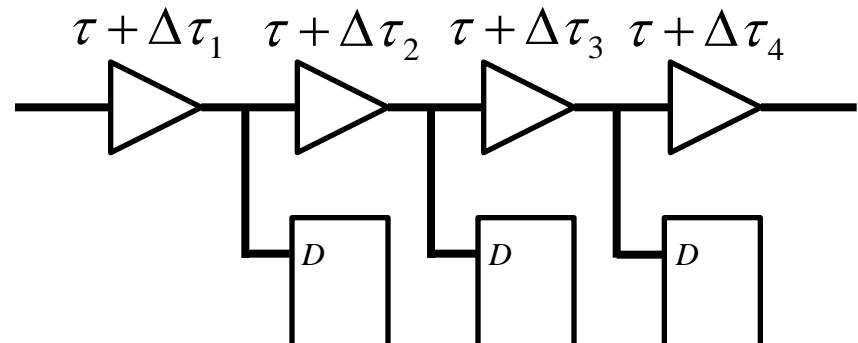
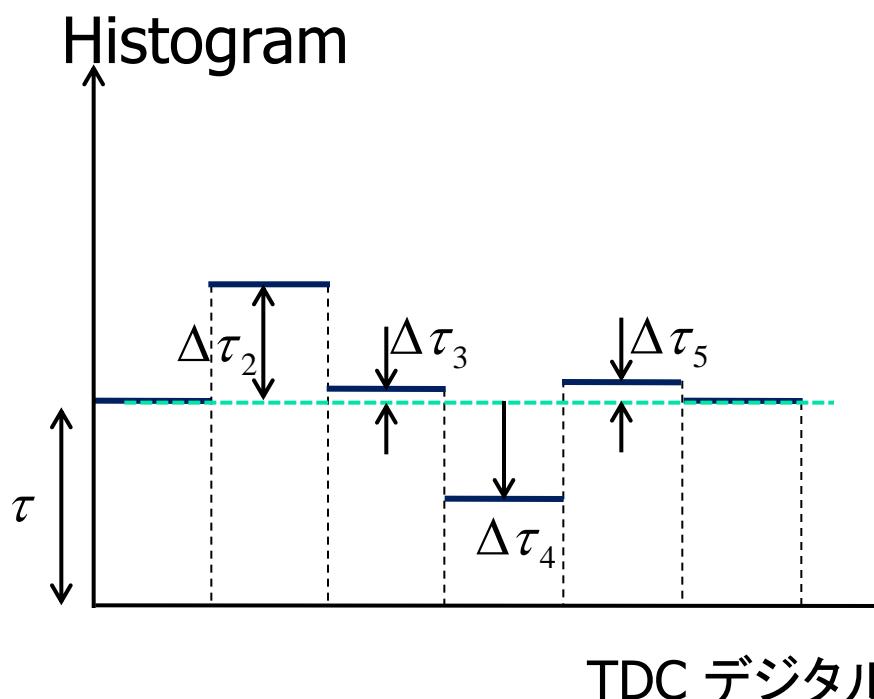
- ・ 充分多くの点数をとれば各デジタルコードのヒストグラムは同一になる
 - ・ 逆に、TDCのヒストグラムデータからDNL, INL を計算

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



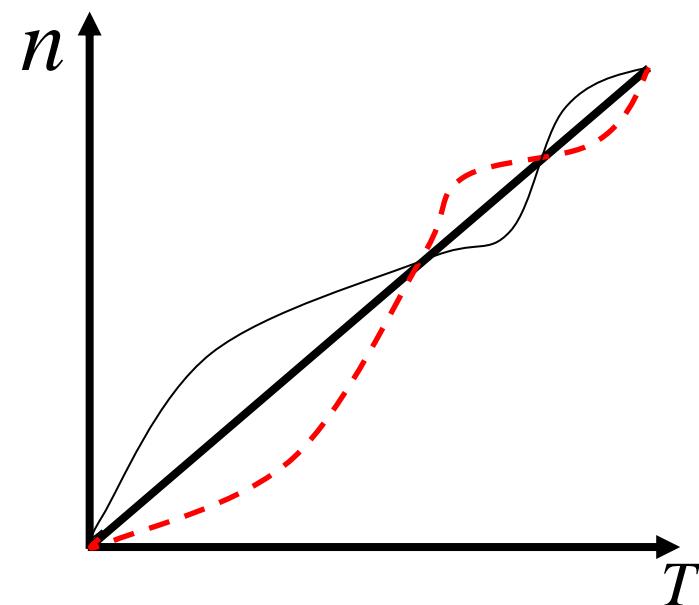
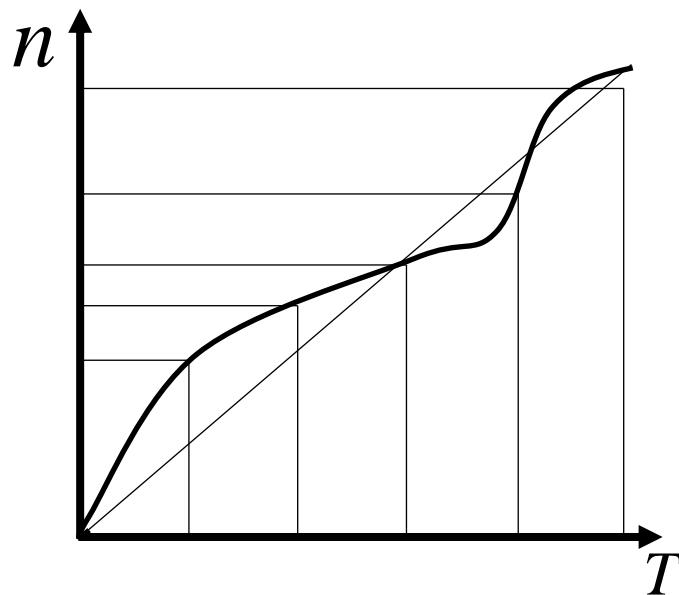
自己校正の原理（非線形性の補正）

通常モード

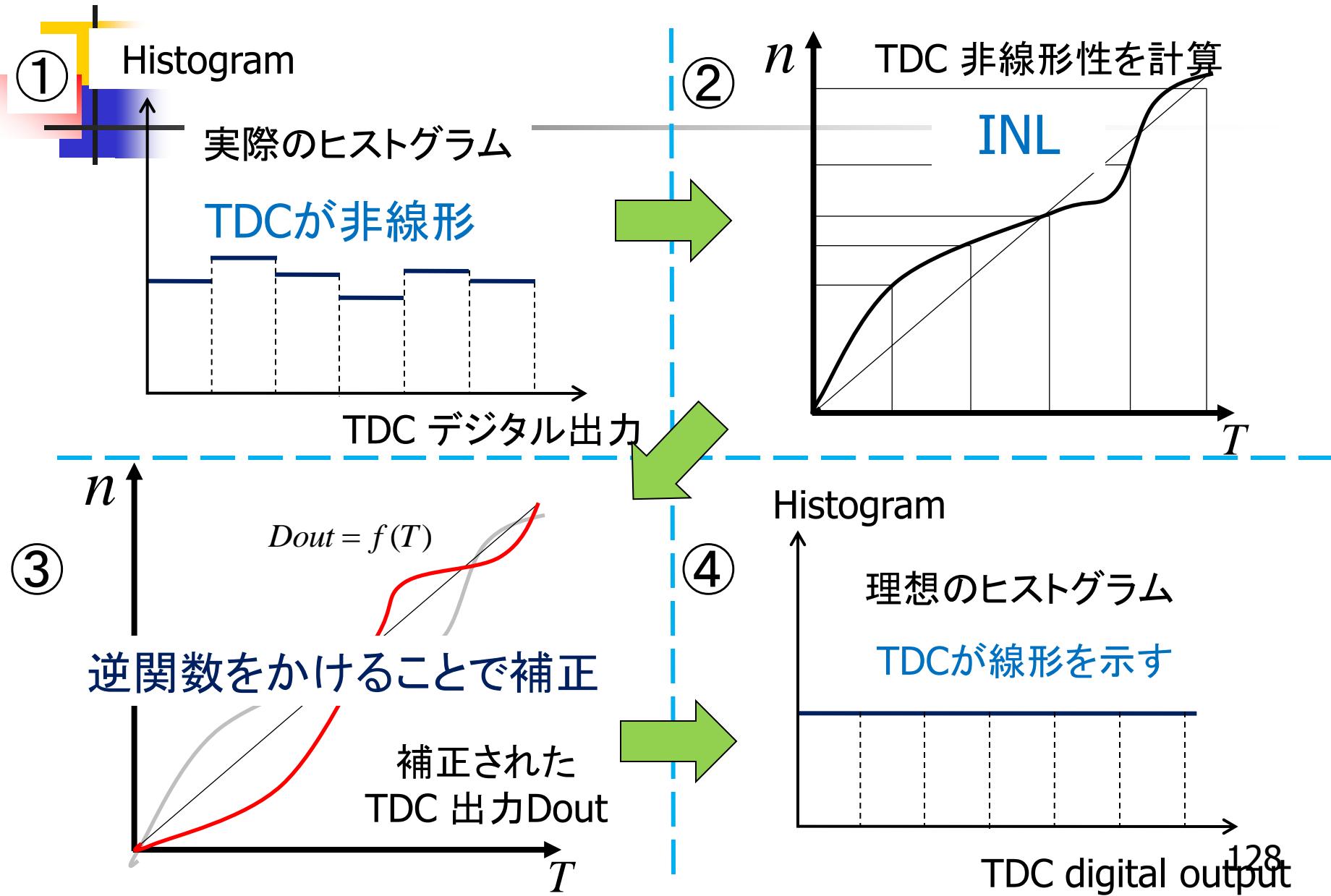
非線形性の逆関数をデジタル的にかける

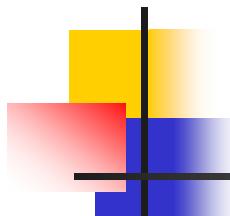


線形性が得られる



非線形性の自己校正



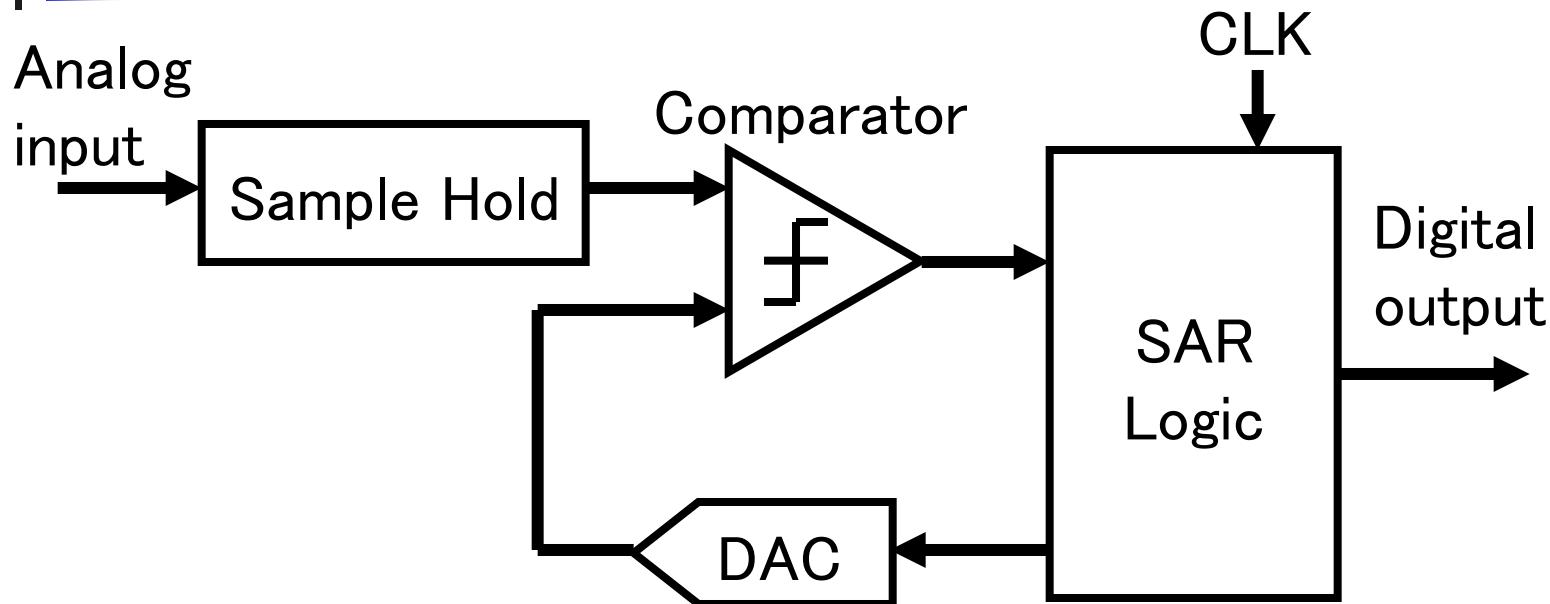


内 容

- 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

16

V_{in}

8

DAC出力

0

1

0

0

1

2進荷重

8

1
2

4bit 4step

4

2

1

4

8

V_{in}

8

4

8

1

2

= 9

“天秤の原理”



V_{in}

=

4
8

-

1
2

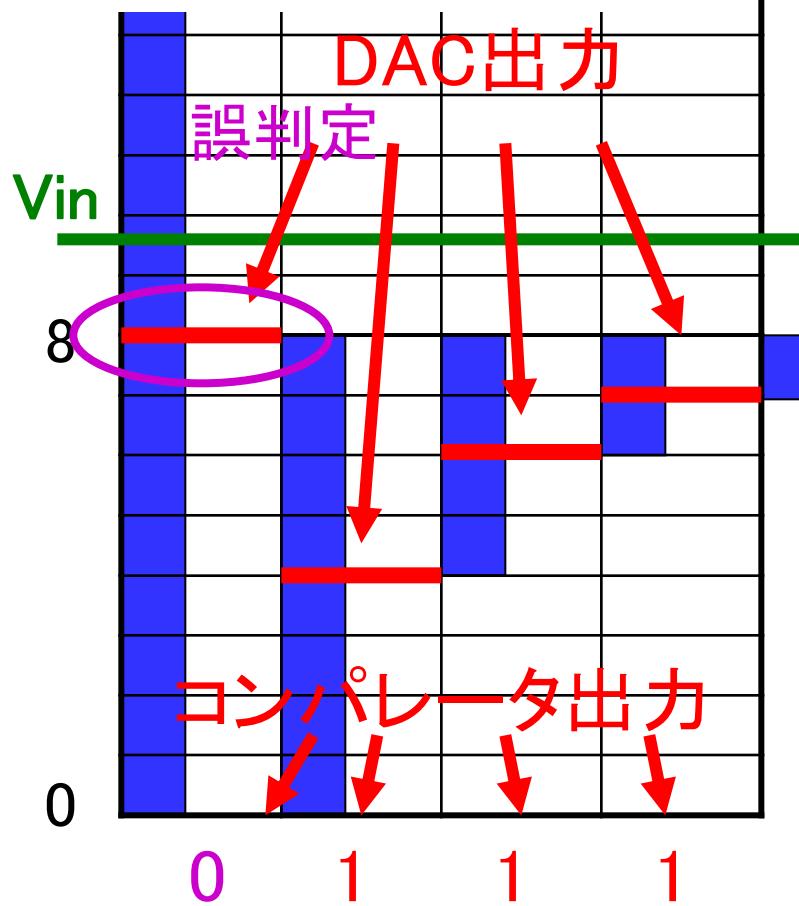
= 9

131

2進探索アルゴリズムの問題点

16

ADC出力に誤差.



2進荷重

8

4

4

2

2

1

1
2

4bit 4step

誤判定

8

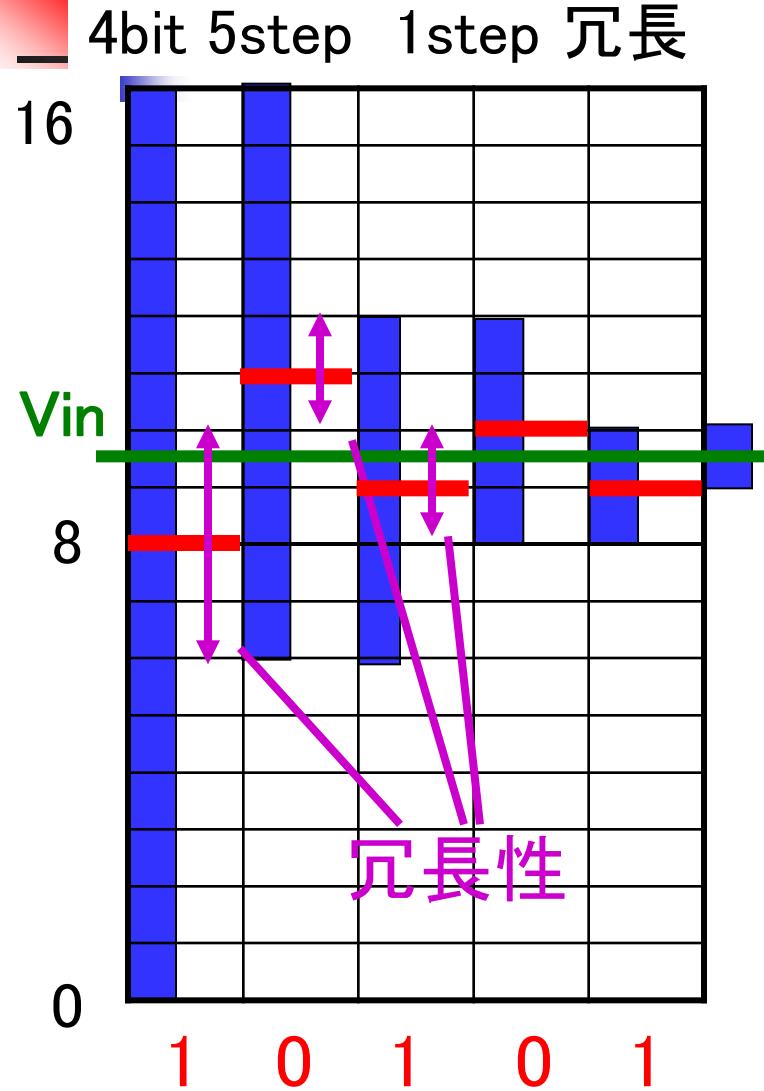
"天秤の原理"

The diagram shows a balance scale with a green square labeled V_{in} on the left pan and a red square labeled 8 on the right pan. A pink arrow points from the 8 square to the scale. Below the scale, a green square labeled V_{in} is equated to the difference between the 8 square and the 4 square: $V_{in} = 8 - 4 = 4$. The text "ADC出力に誤差" (ADC output error) is also present.

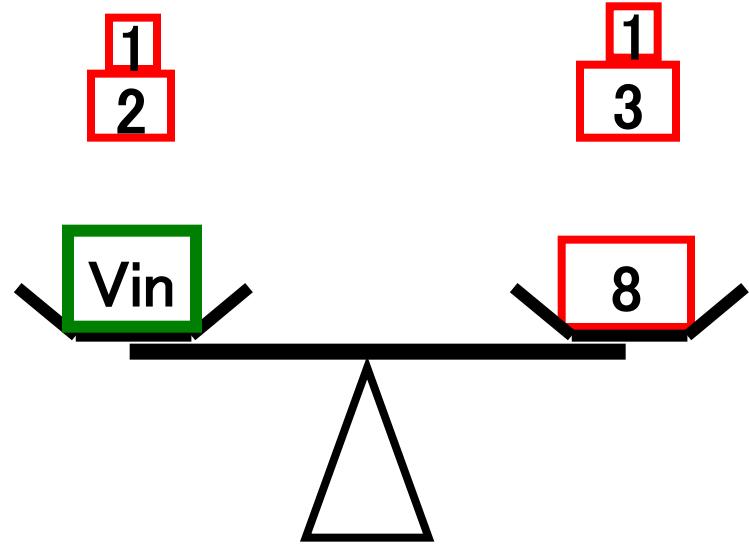
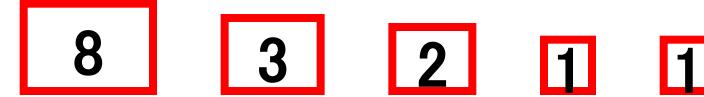
$$V_{in} = 8 - 4 = 4$$

ADC出力に誤差.

非2進探索アルゴリズム



非2進荷重



$$\boxed{\text{Vin}} = \boxed{8} - \boxed{2} = 9$$

133

非2進探索アルゴリズム

4bit 5step 1step 冗長

16

冗長性

Vin

8

誤判定

0

0

1

1

1

1

非2進荷重

8

3

2

1

1

1

1

8

3

誤判定

Vin

$$Vin = 8 - 3 = 9$$

誤差補正原理

デジタル出力“9”的場合

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

$$D_{out} = 8 + 4 - 2 - 1 + 0.5 - 0.5 = 9$$

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

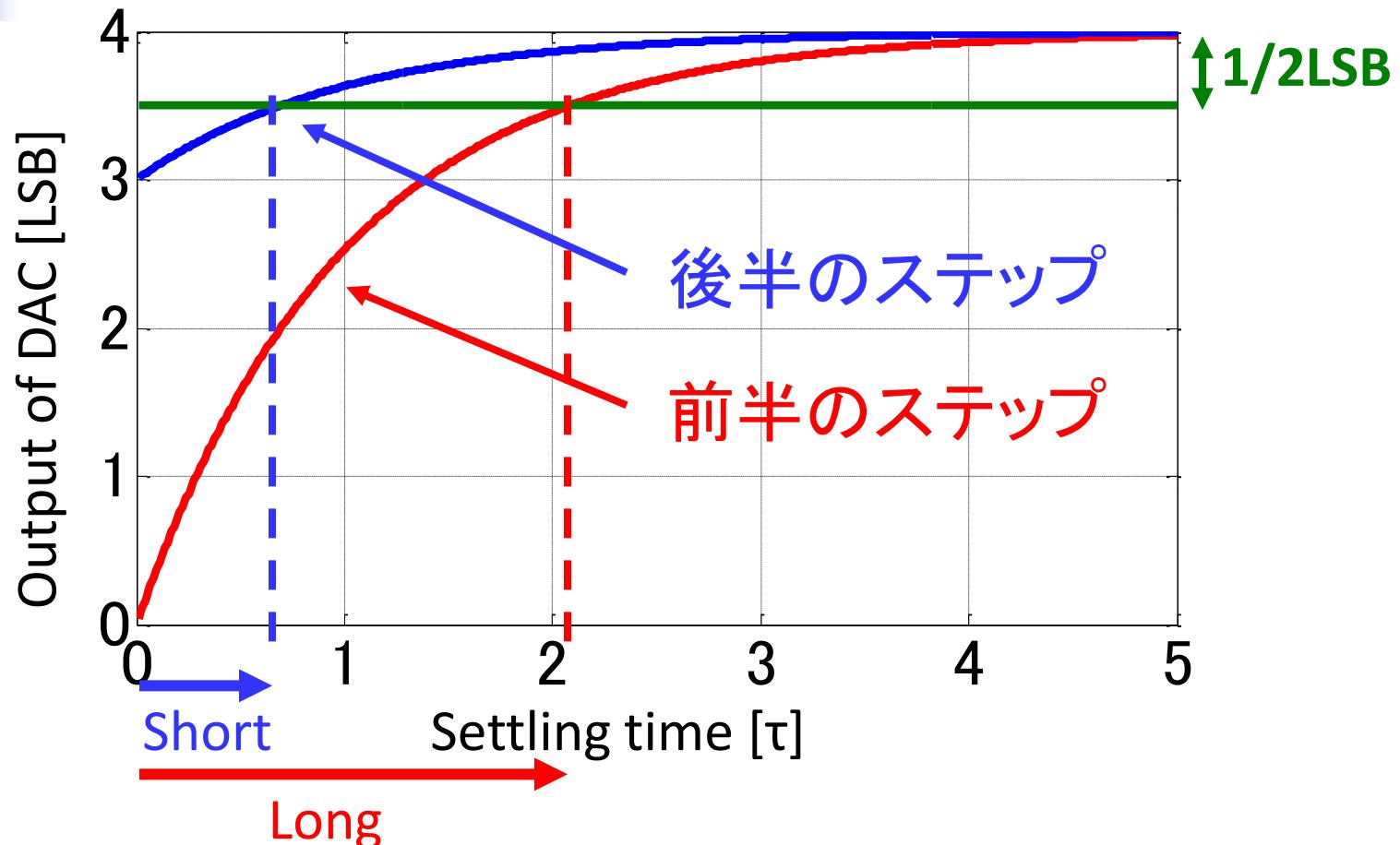
$$D_{out} = 8 + 3 - 2 + 1 - 1 + 0.5 - 0.5 = 9$$

コンパレータ出力: 0 1 1 1 1

$$D_{out} = 8 - 3 + 2 + 1 + 1 + 0.5 - 0.5 = 9$$

冗長アルゴリズムによる高速化

DAC出力の整定



AD変換時間の比較



変換時間のMATLABシミュレーション

2進アルゴリズム

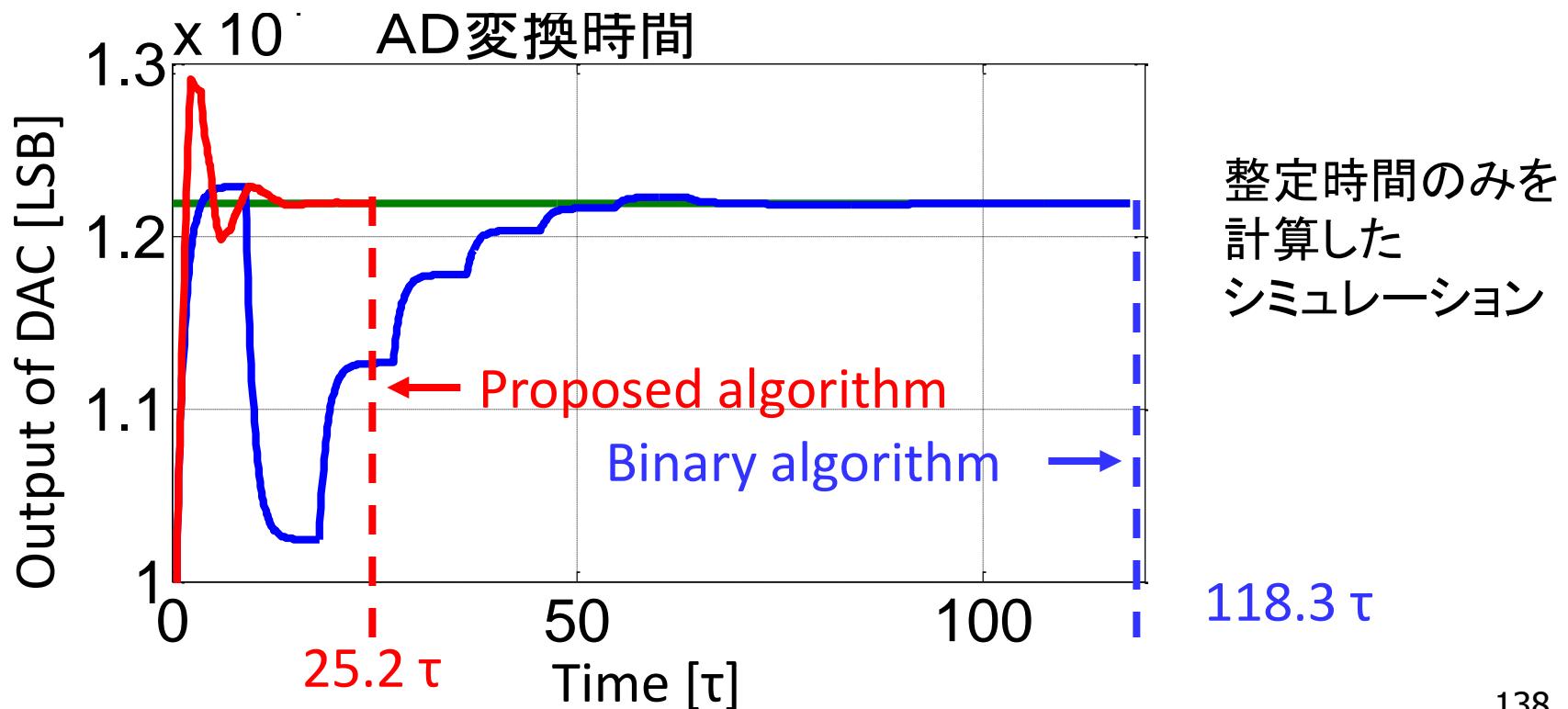
14-bit, 14-step

各ステップでの整定時間: 9.1τ

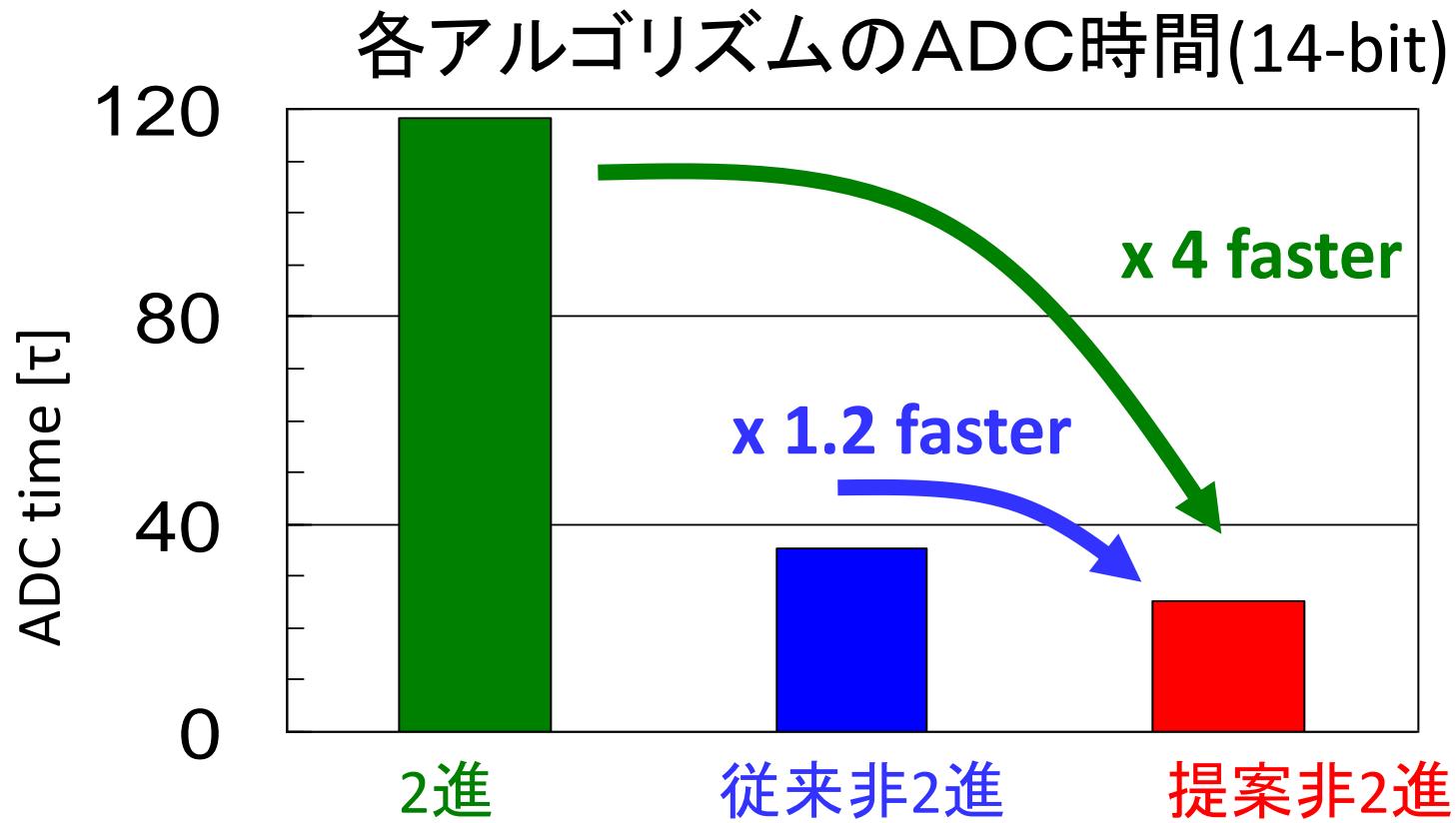
提案アルゴリズム

14-bit, 22-step

各ステップでの整定時間: 1.2τ



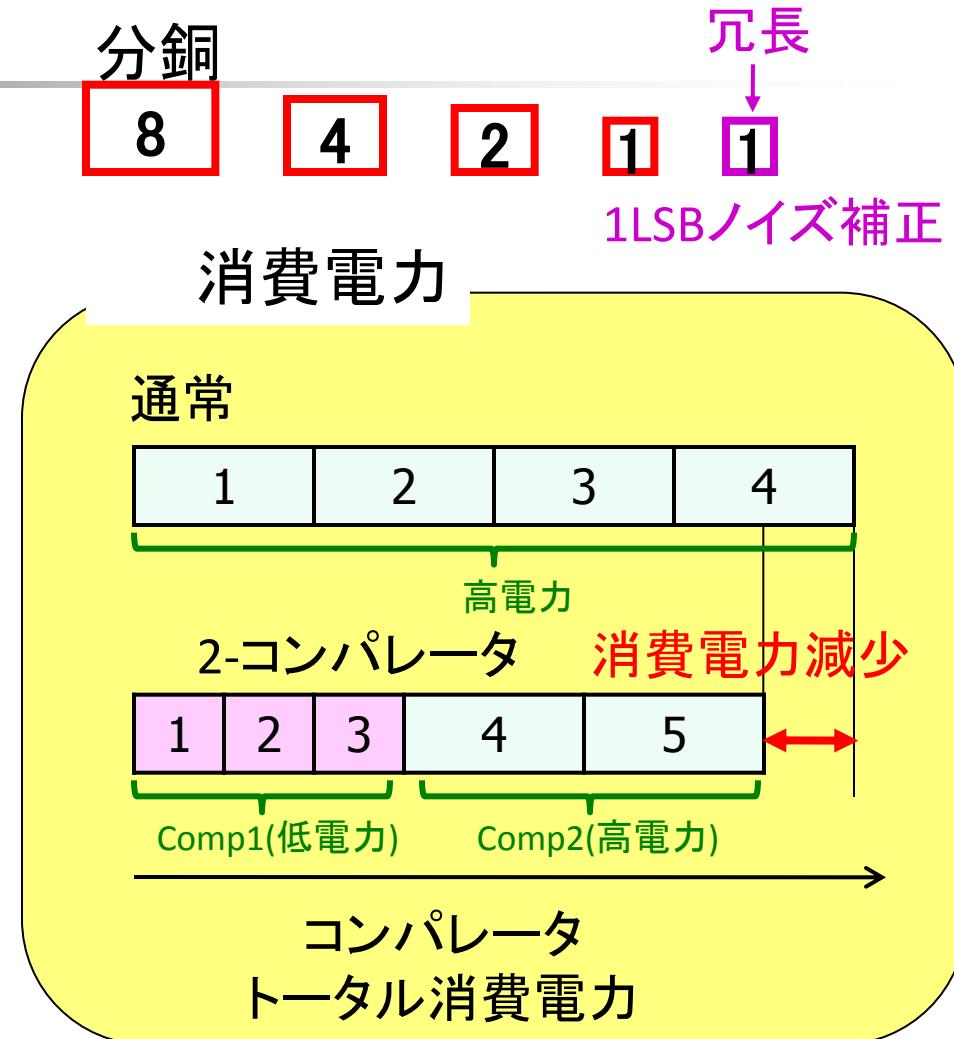
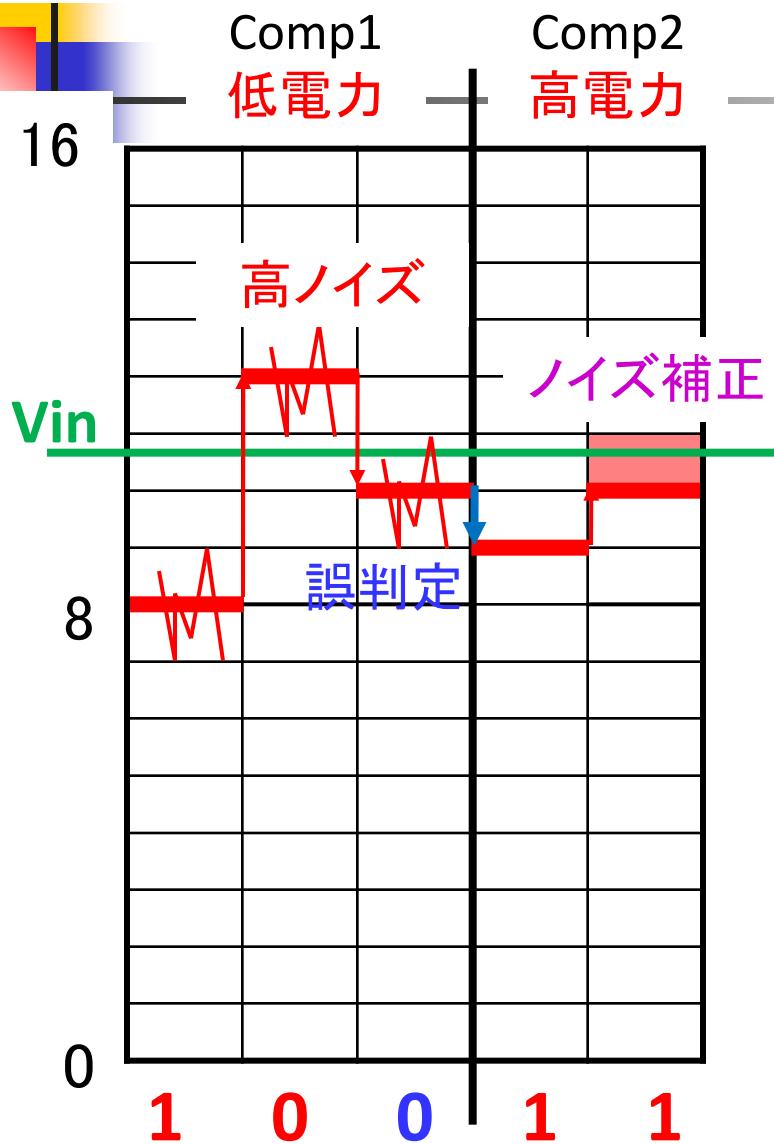
AD変換時間の比較



提案アルゴリズム → 4倍速い

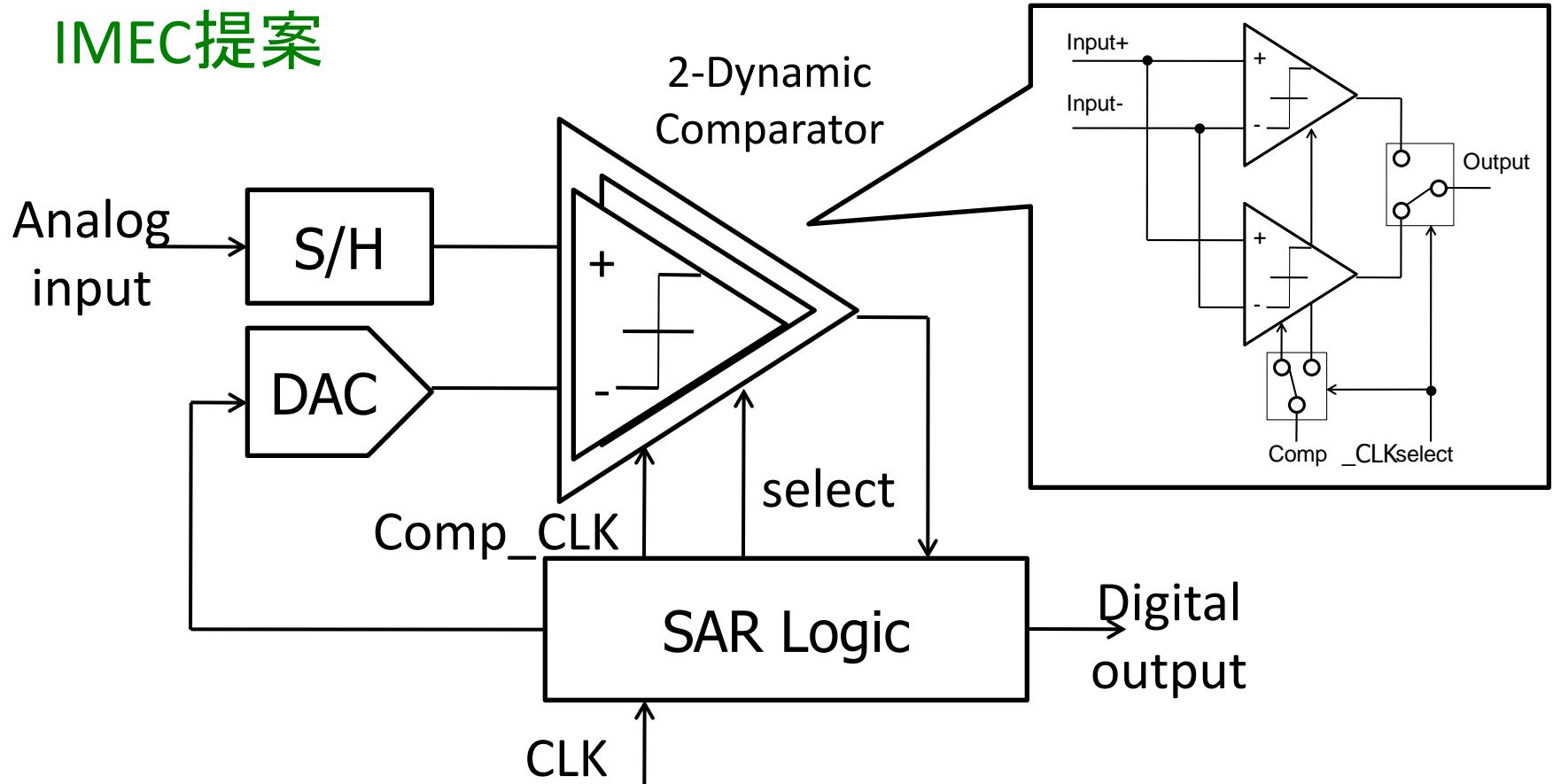
冗長アルゴリズムによる低消費電力化

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

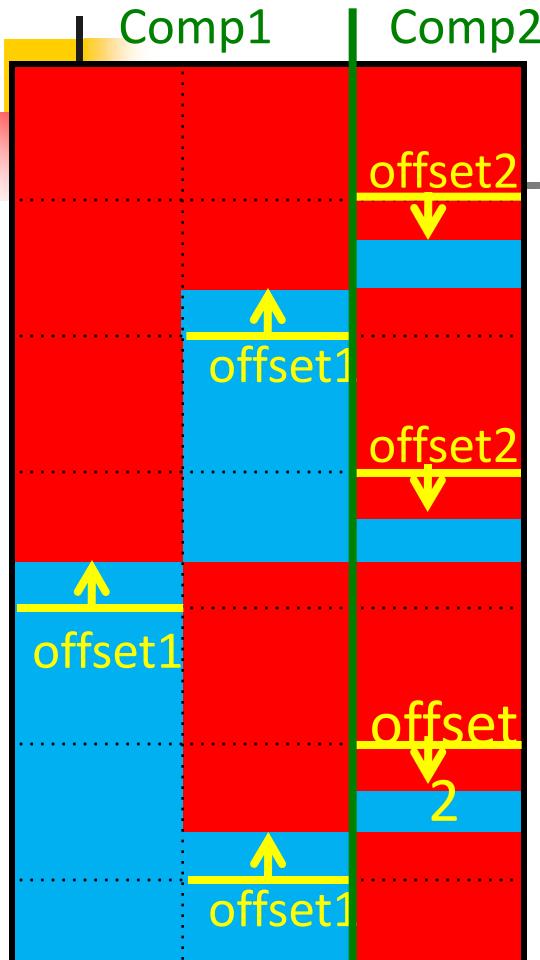


文献

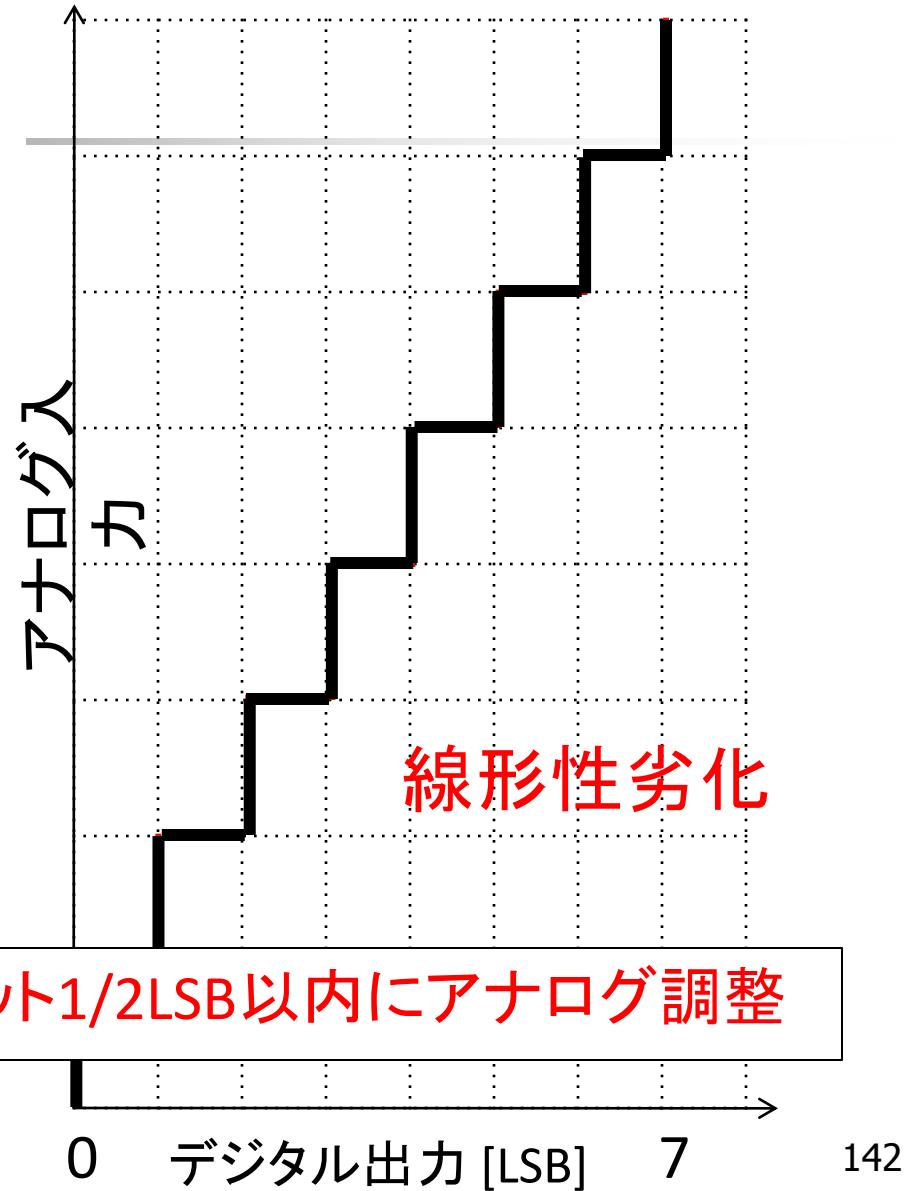
V.Giannini, P.Nuzzo, V.Chironi,A.Baschirotto, G.V.Plas, J.Craninckx

“An 820 μ W 9b 40MS/s Noise-Tolerant Dynamic-SARADC in 90nm Digital CMOS” ISSCC (Feb.2008). 141

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

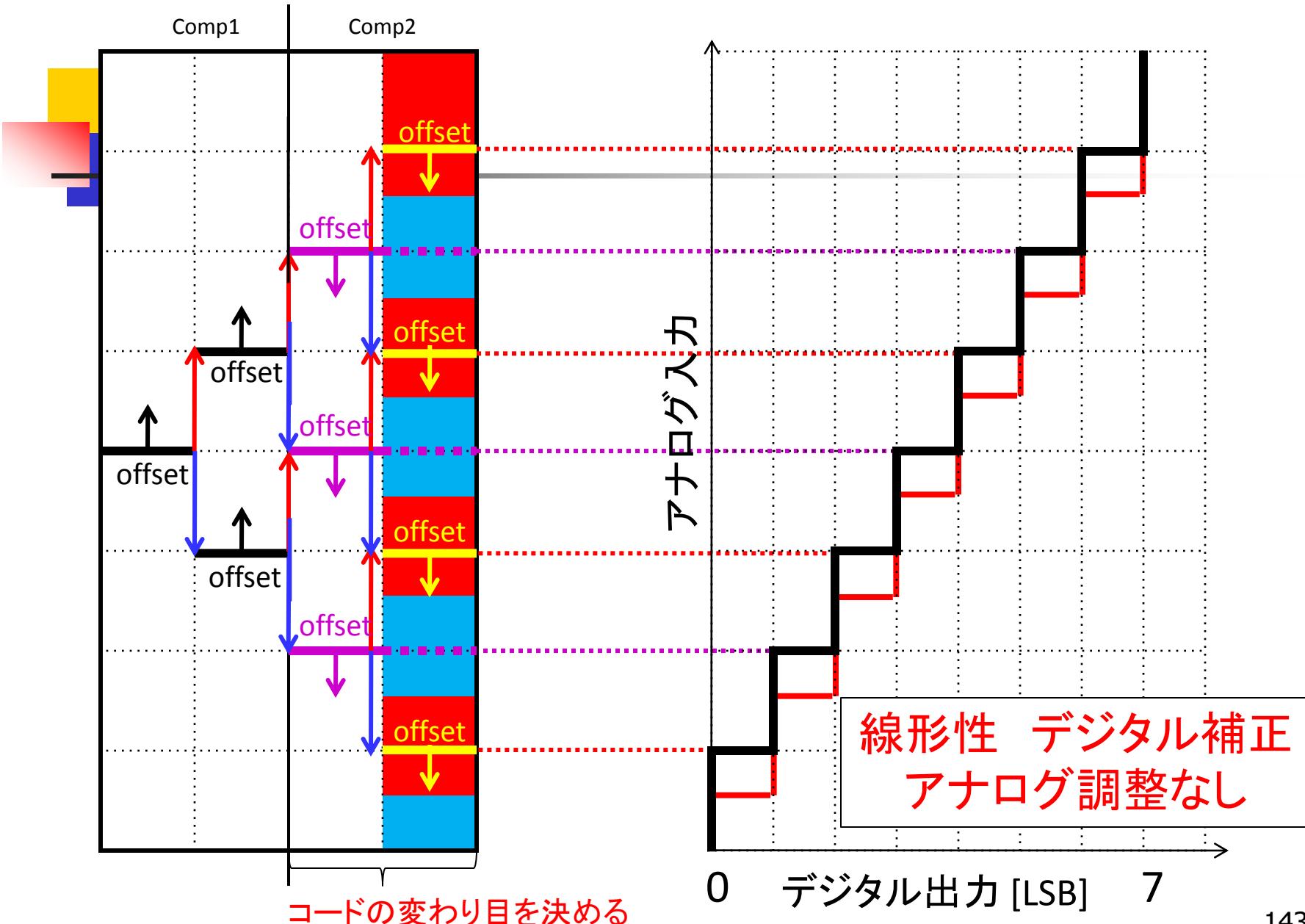


■ :1判定
■ :0判定



IMEC コンパレータオフセット1/2 LSB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



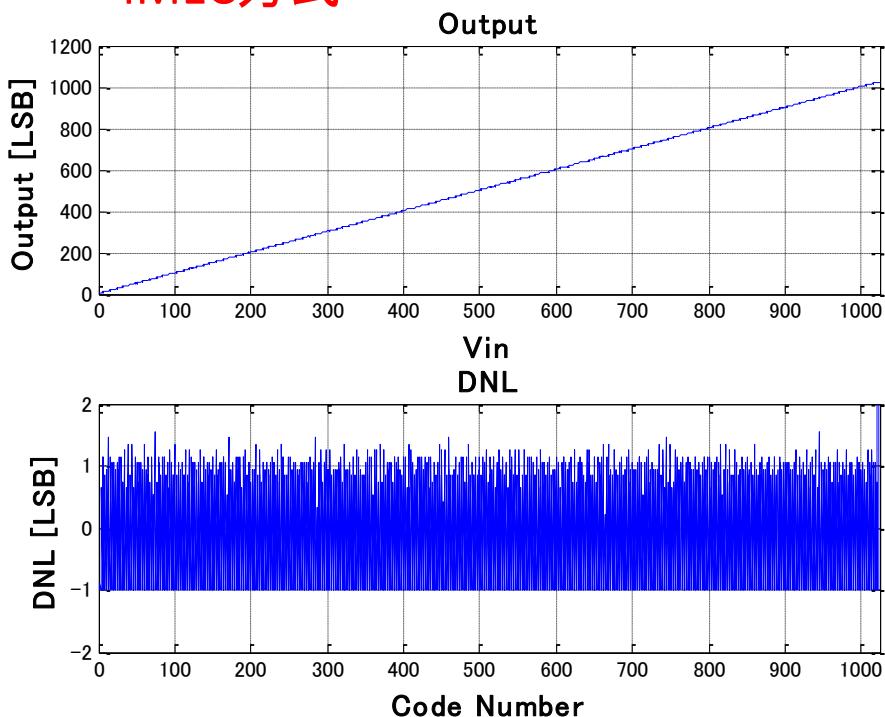
MATLABシミュレーション(ランプ波)

Comp1(低電力) オフセット: +4.0 LSB、 ノイズ: 1.0 LSB

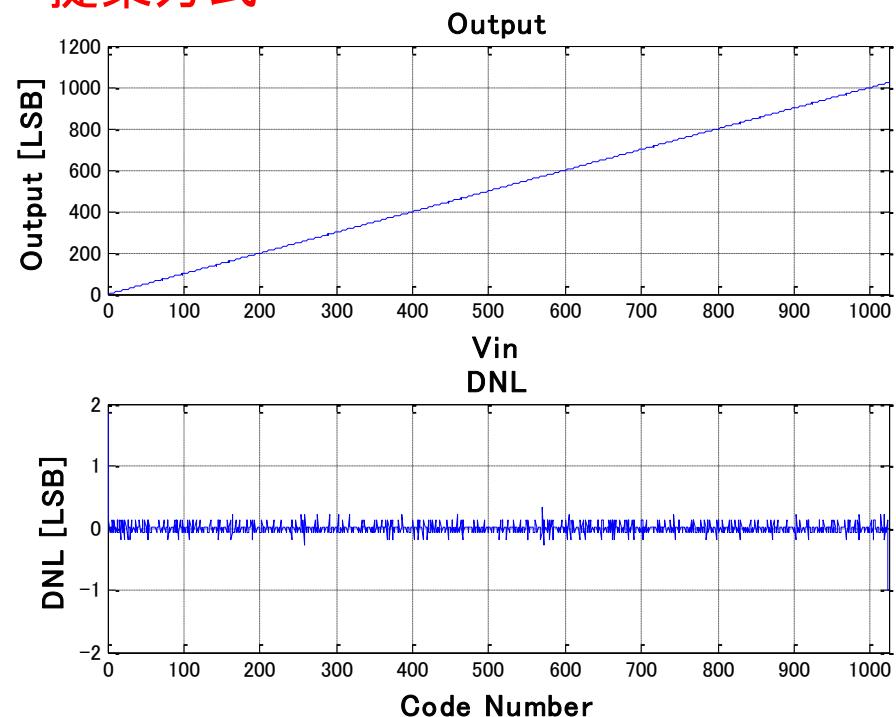
Comp2(高電力) オフセット: -2.0 LSB、 ノイズ: 0.2 LSB

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

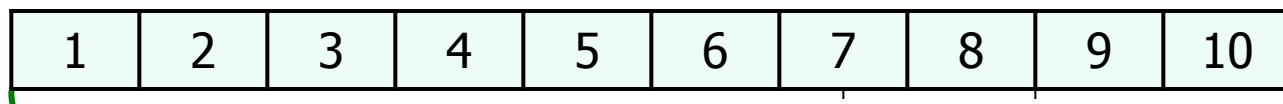


消費電力とコンパレータミスマッチ許容の トレードオフ

トレードオフ

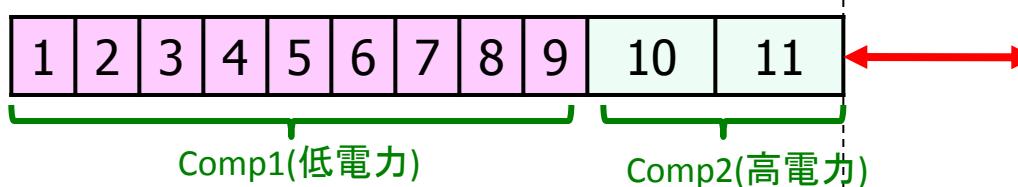
低消費電力化 ⇌ コンパレータのミスマッチ許容

通常 1-コンパレータ



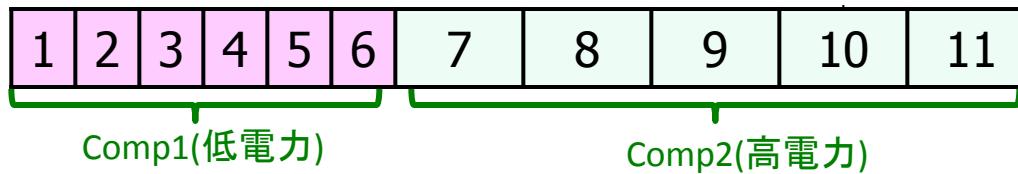
高電力

IMEC方式 2-コンパレータ(コンパレータミスマッチ許容:小)



低消費電力化の 効果が下がる

提案 2-コンパレータ(コンパレータミスマッチ許容:大)



コンパレータトータル消費電力

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.

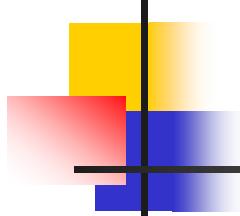
さらに効率が良くなる.

- ・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

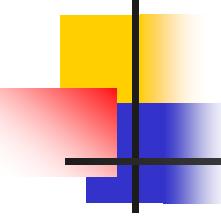
→各構成要素, 動作への要求緩和

(「無用の用」老子: 東洋哲学)



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

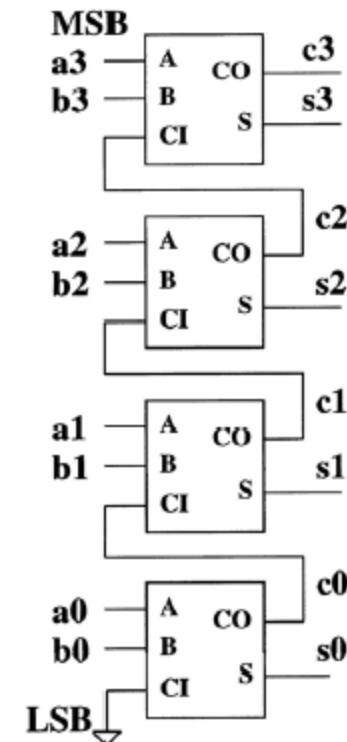
リップルキャリー加算器
下位ビットの桁上げが
次の上位ビットの入力に接続



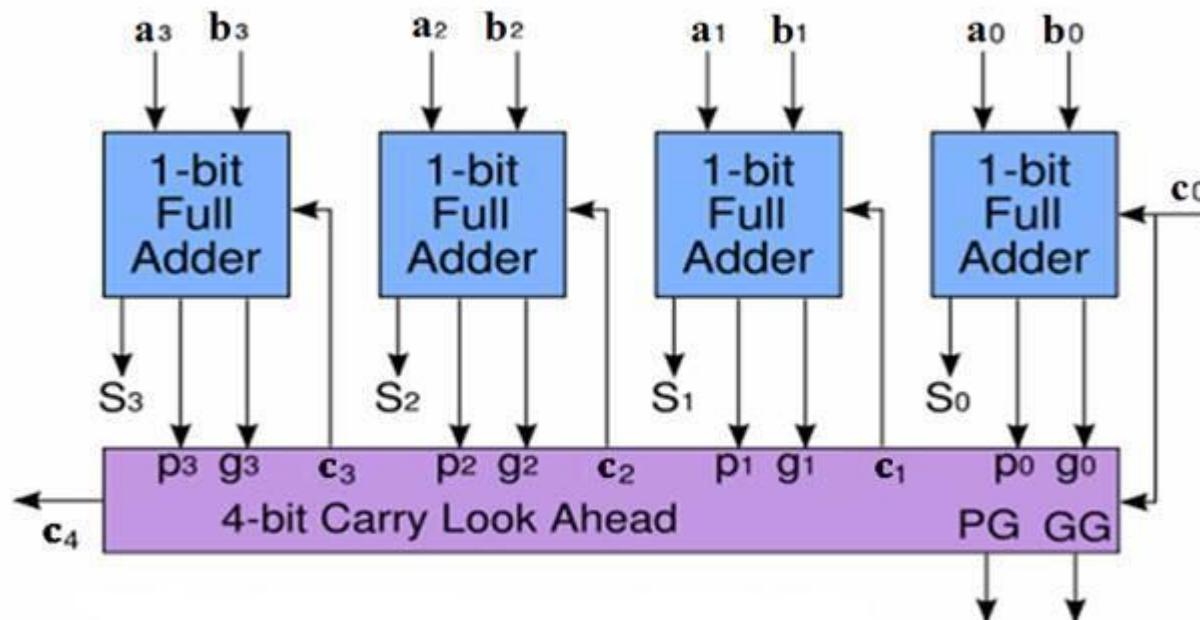
桁数分のキャリーの伝播が生じるので
全体の加算演算の時間がかかる



桁上げ信号の部分
別に早く計算し高速化



Carry Look Ahead Adderとは

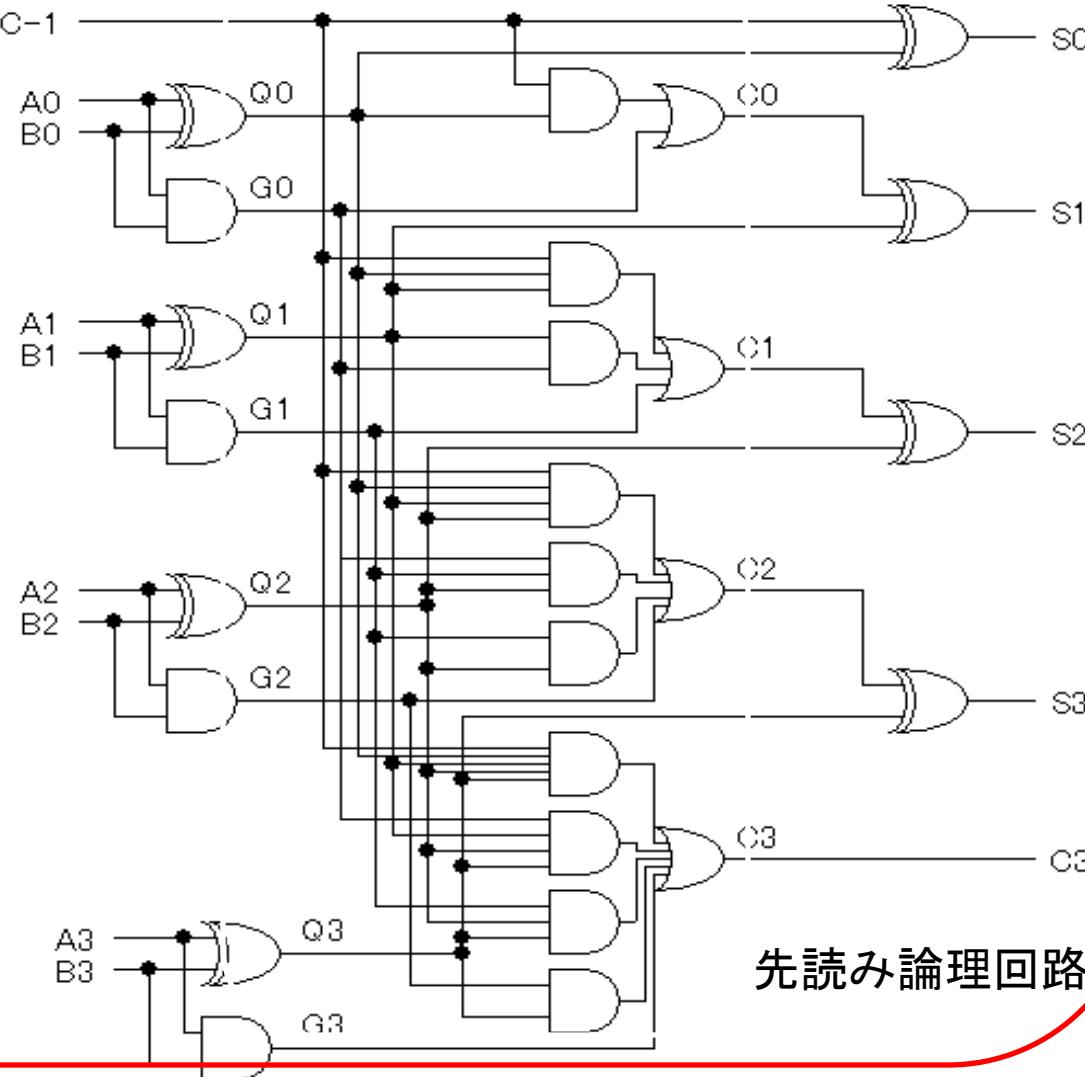


桁上げ(Carry)を別に先に計算する



Carry Look Ahead Adder 回路

4bit Carry Look Ahead Adder



Carry Look Ahead 論理式

キャリー生成項 と キャリー伝播項

- C_n :nビット目のキャリー

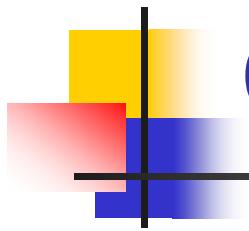
$$\begin{aligned} C_n &= A_n B_n + (A_n + B_n) \cdot C_{n-1} \\ &= G_n + P_n \cdot C_{n-1} \end{aligned}$$

- G_n :キャリー発生関数

$$G_n = A_n \cdot B_n$$

- P_n :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

$$C_0 = G_0 + P_0 \cdot C_{-1}$$

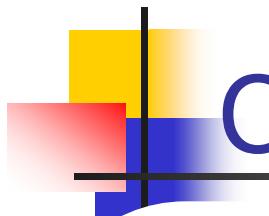
これを、 C_1 に代入

$$C_1 = G_1 + P_1 \cdot C_0$$

$$= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_{-1})$$

$$= A_1 \cdot B_1 + (A_1+B_1)\{A_0 \cdot B_0 + (A_0+B_0) \cdot C_{-1}\}$$

C_0 が決まればすぐに C_1 が決まる



Carry Look Ahead 論理式 (続き)

$$\begin{aligned}C_2 &= G_2 + P_2 \cdot C_1 \\&= G_2 + P_2 \cdot (G_1 + P_1 \cdot C_0) \\&= G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_{-1})) \\&= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}\end{aligned}$$

C₂もC₀が決まればすぐに決まる

$$\begin{aligned}C_3 &= G_3 + P_3 \cdot C_2 \\&= G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot (G_0 + P_0 \cdot C_{-1}))) \\&= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}\end{aligned}$$