

KYUSHU UNIVERSITY · SYS-104

デジタル回路の 基礎 (2)

順序回路とメモリ

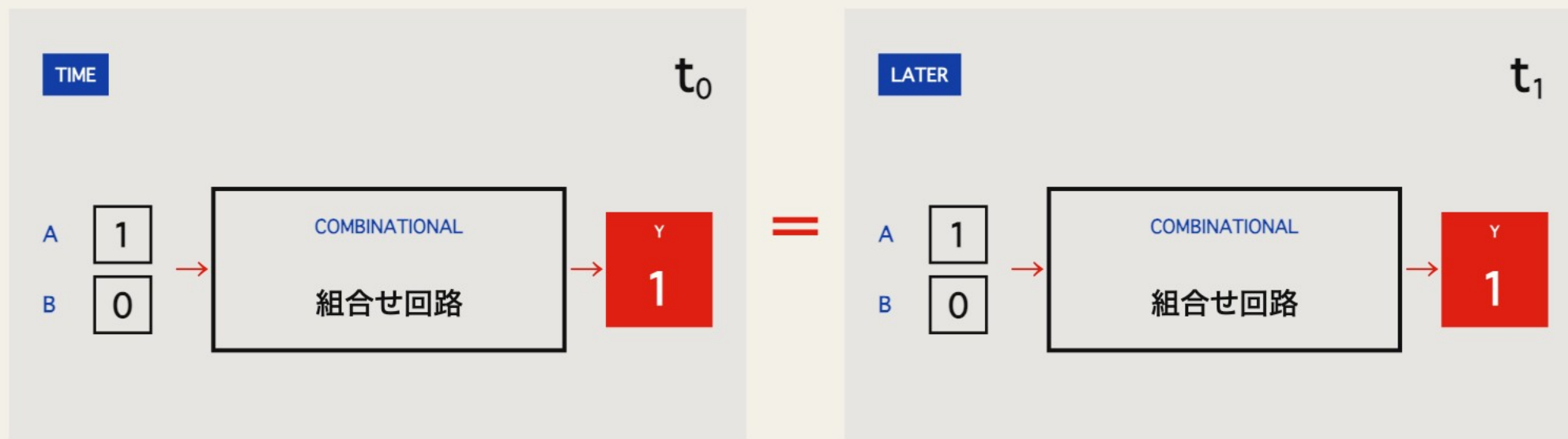
2026 年 9 月

Olivia Chen Toyama · C2LAB

LECTURE 4



組合せ回路の復習

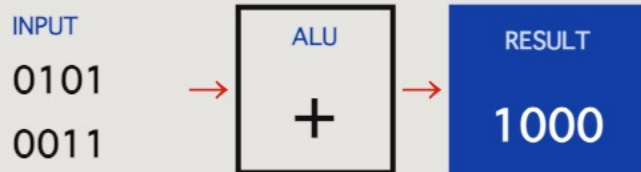


CURRENT INPUT

出力は「いま入っている値」だけで決まる

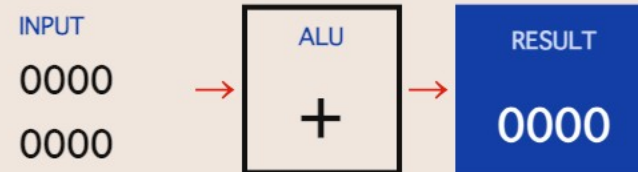
計算結果を残すには

01 計算中



入力が変わる

02 次の計算



NEEDED: MEMORY

前の結果 1000 は、組合せ回路の中には残らない

回路に「状態」を記憶させるには？



OUROBOROS / CLOSED LOOP

出力を、もう一度入力へ戻す



フィードバック

現在の出力を、次の出力を決める材料として使う
安定して保持できるかは、輪の中の回路による。

記憶回路の3つの機能

01

ONE BIT



1 ビットを保持する

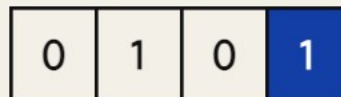
入力が変わっても、0 または 1
を残す

PART 1-2

フィードバック+ラッチ

02

REGISTER



→ 一斉に更新する

複数ビットを、同じ時刻に書き
換える

PART 3-4

クロック+レジスタ

03

MEMORY

00	0101
01	1110
10	0011

→ 読み書きする場所を選ぶ

アドレスで、必要なデータを指
定する

PART 5

メモリ

この3つの機能を、次の5段階で組み立てる

まず、出力を入力へ戻す



まずは、二つの NOT で 0 と 1 が安定する仕組みを調べる。

「現在の入力」だけでは決まらない回路

COMBINATIONAL

組合せ回路



同じ入力なら、いつでも同じ出力



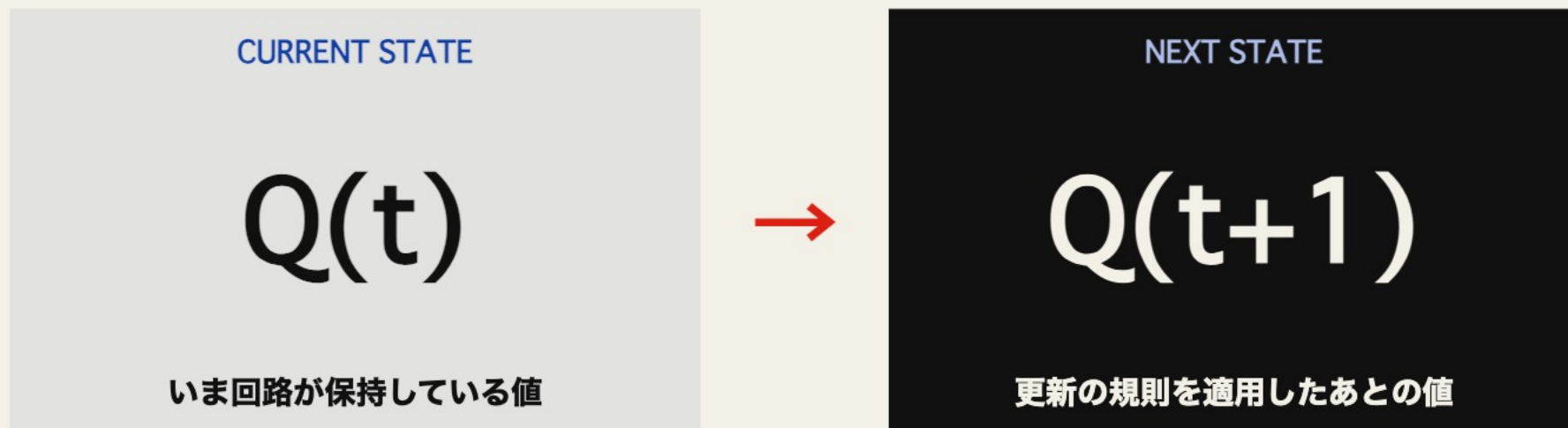
SEQUENTIAL

順序回路



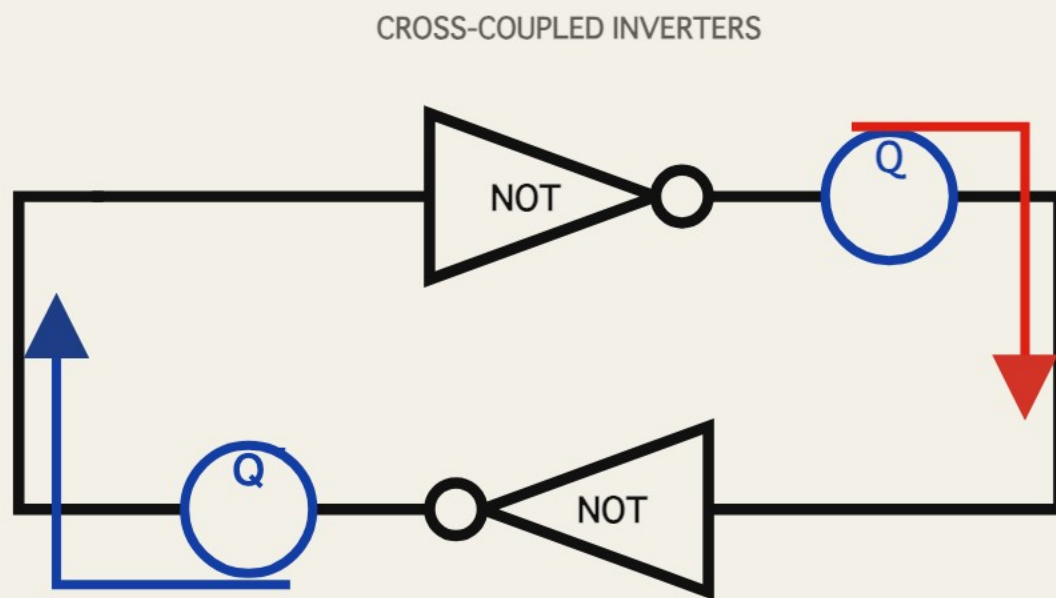
同じ入力でも、前の状態によって結果が変わる

今の状態と次の状態



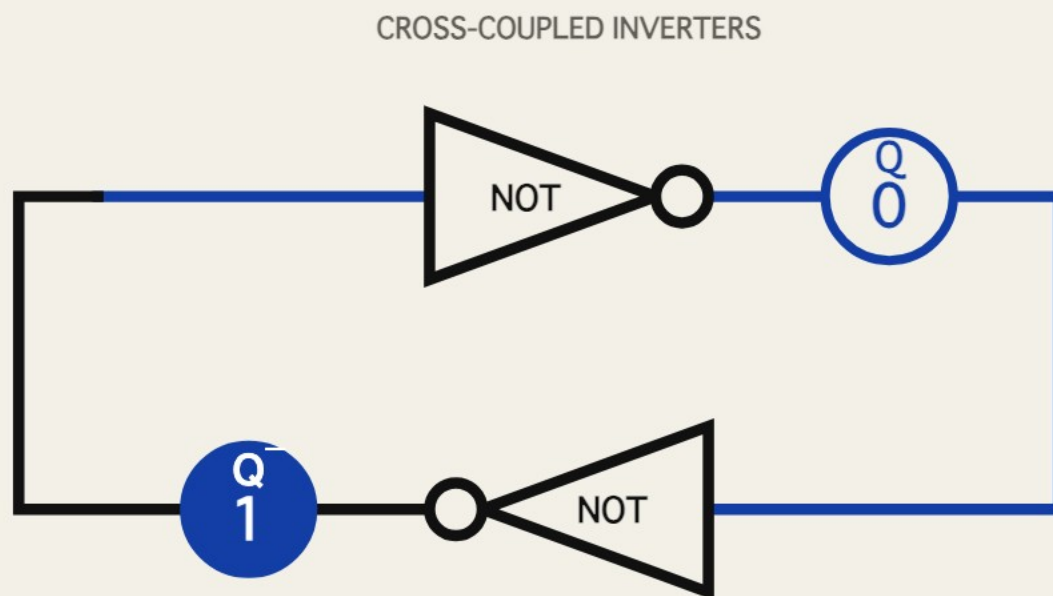
例：いま $Q(t)=0$ 。「1 を書く」と、次の $Q(t+1)=1$ 。

出力を、次の入力へ戻す



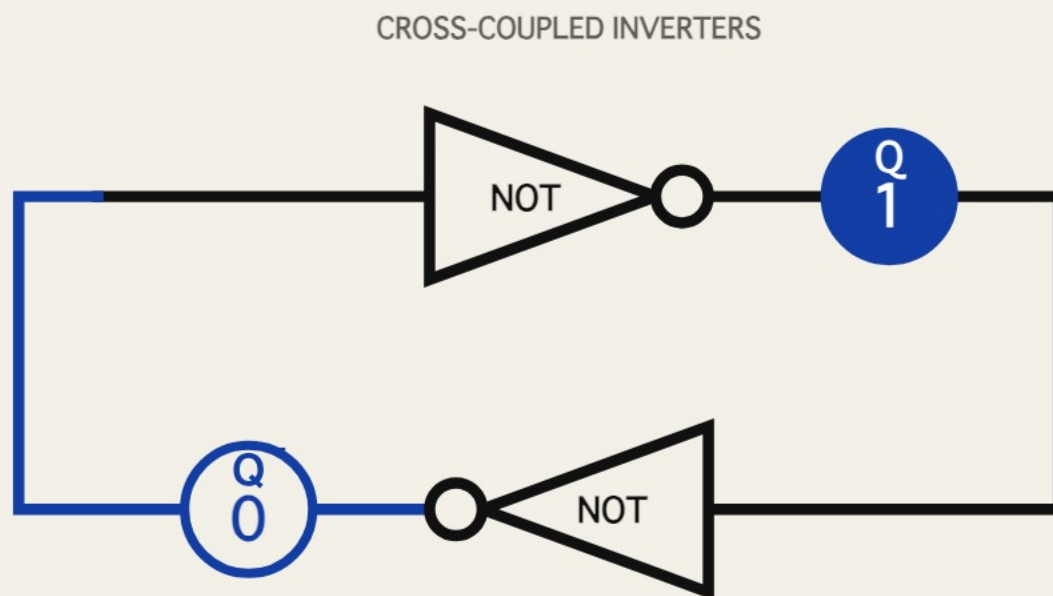
出力が、反対側の次の入力へ戻る

安定状態 0



NOT の関係を満たすため、この状態を自分で支え続ける

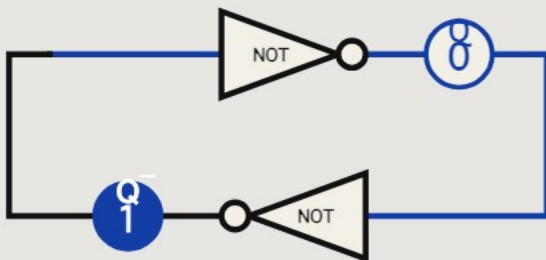
安定状態 1



NOT の関係を満たすため、この状態を自分で支え続ける

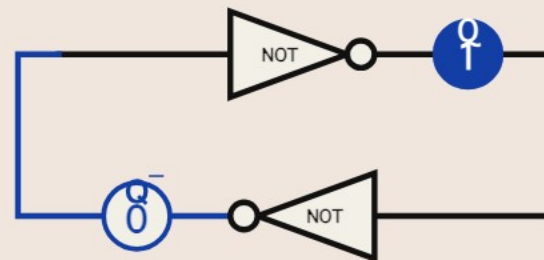
一つの回路が、0 と 1 を保持する

$Q = 0$



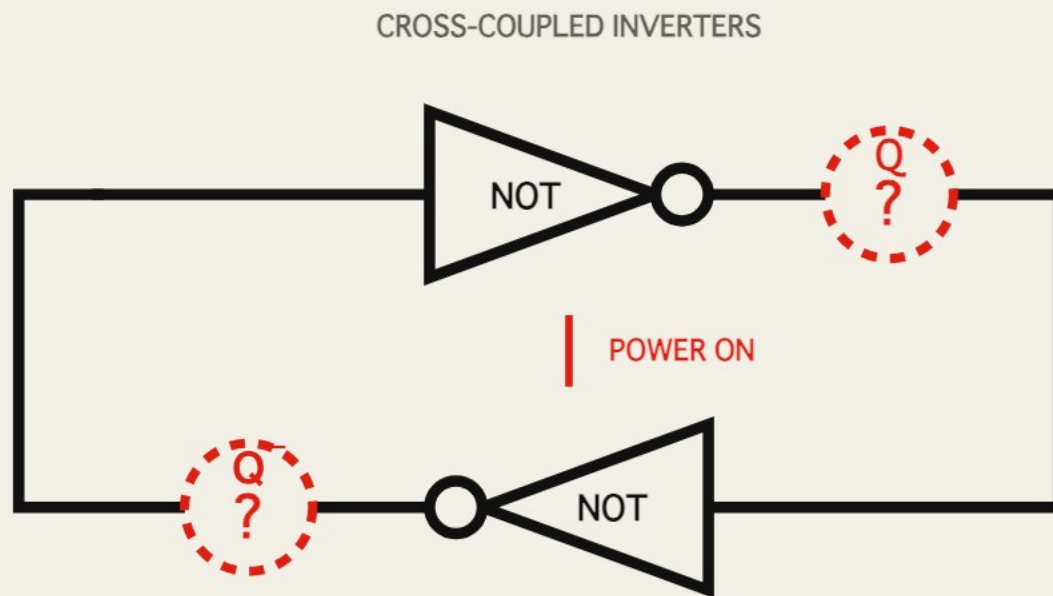
$Q^{\bar{}} = 1$ と組になって安定

$Q = 1$



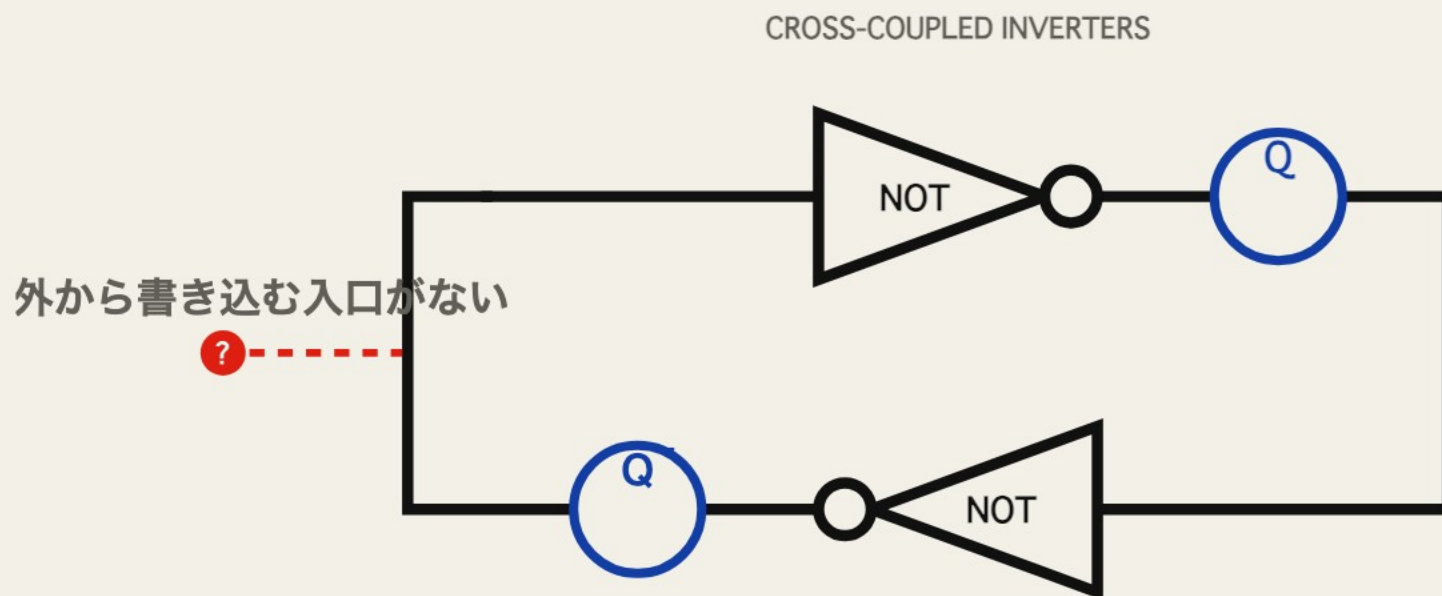
$Q^{\bar{}} = 0$ と組になって安定

電源投入直後は、どちらになる？



小さな差で、どちらの状態に落ち着くかが決まる

保持できても、書き込めない



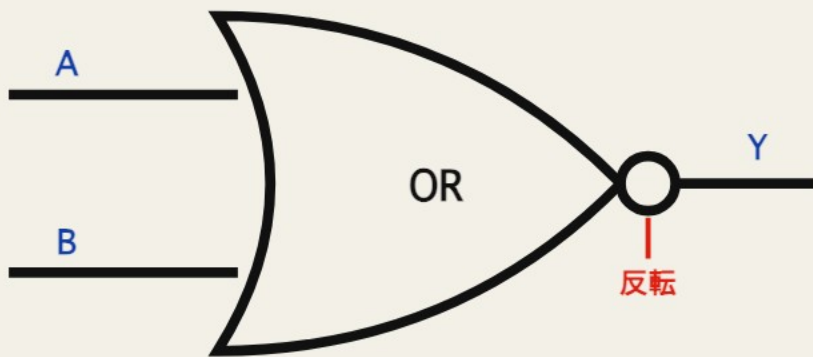
欲しい操作は二つ：**0 を書く** / **1 を書く**。書いた後は、その値を保つ。

保持する状態を、外から選ぶ



保持できるようになった。次は、外から 0 / 1 を書き分けたい。

NOR ゲートの記号と規則



NOR OR の出力を NOT で反転する

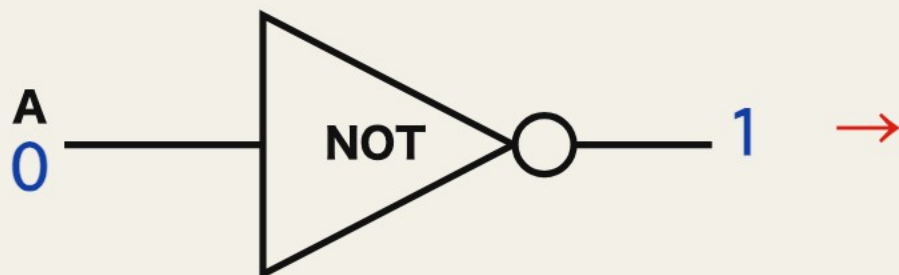
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOR にすると、書き込み用の入口を増やせる

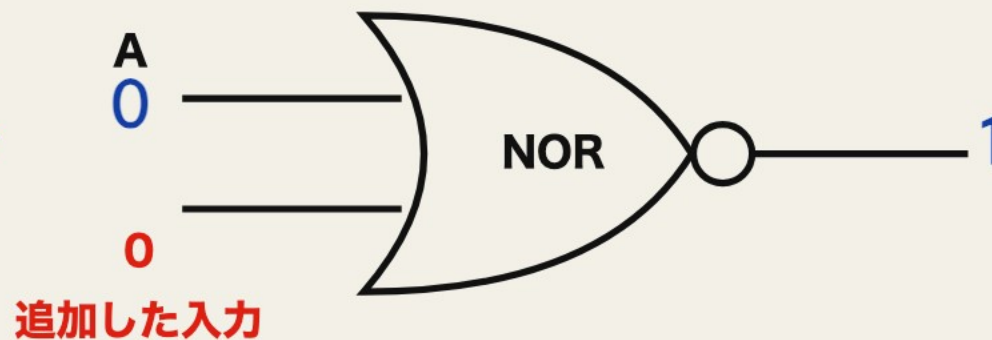
A を 1 にする

追加した入力を 1 にする

これまで：NOT



入力を増やす：NOR

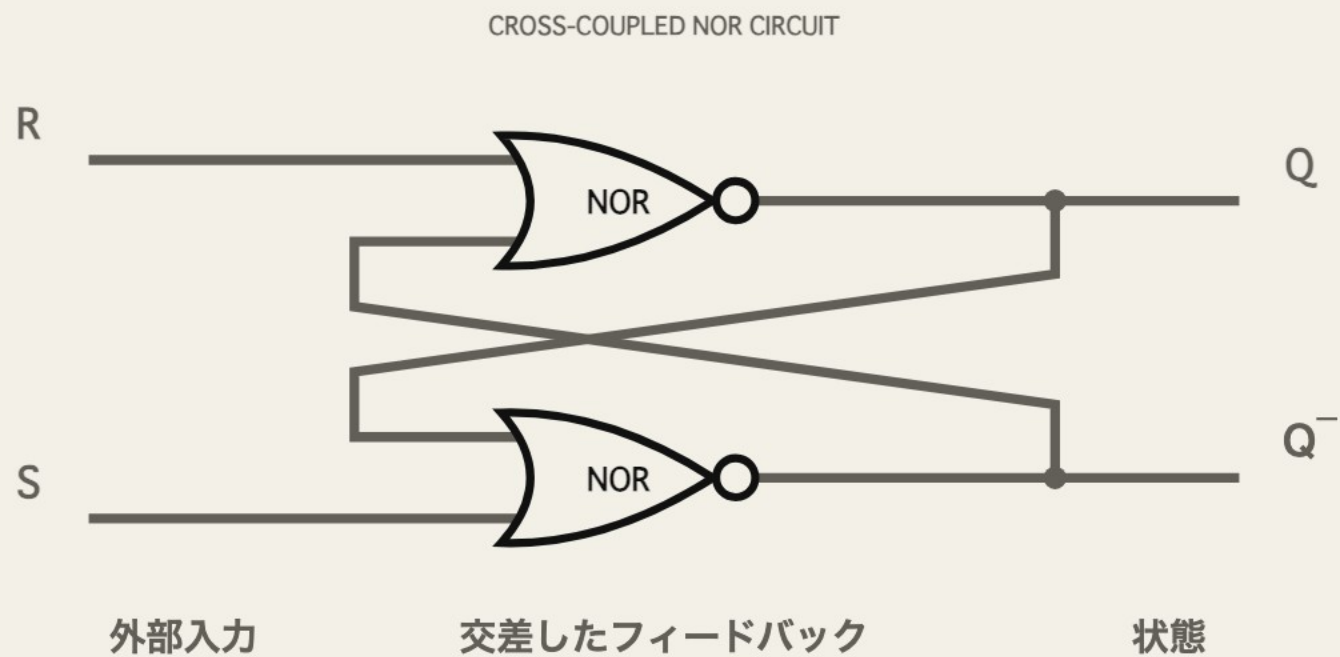


$$\text{NOR}(A, 0) = \text{NOT}(A)$$

追加した入力が 0 なら、今までと同じ反転回路になる

二つの NOT を NOR に置き換え、状態を選ぶ入口を加える

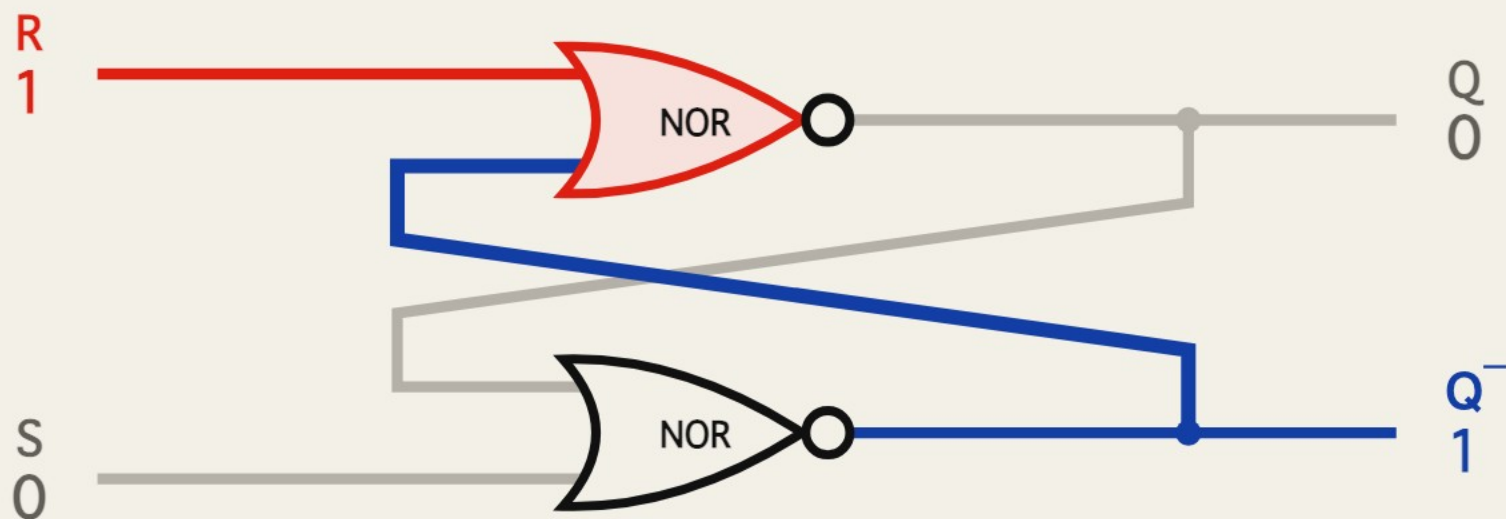
二つの NOR を交差接続する



S と R を変えて、記憶を確かめる

0 を書く → 保持する → 1 を書く → 保持する。同じ「00」の結果を比べる。

0 を書く S=0, R=1	保持する S=0, R=0	1 を書く S=1, R=0	禁止入力 S=1, R=1	最初から
-------------------	------------------	-------------------	------------------	------



理由を見る

① 入力側の NOR

② 戻り先の NOR

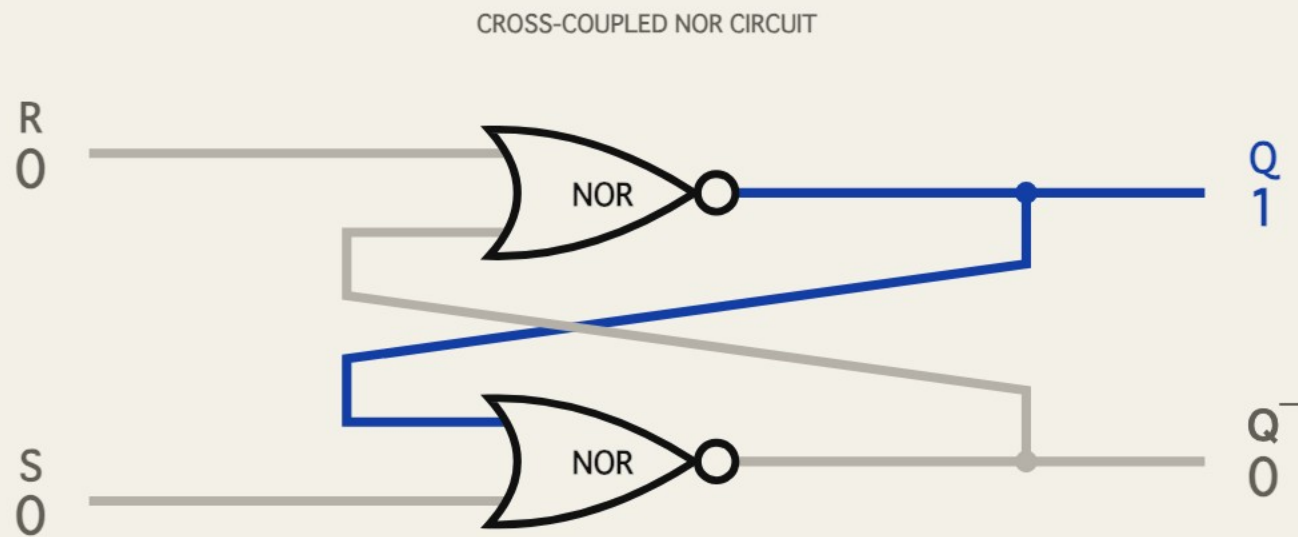
S	R	次の Q
0	0	—
0	1	0
1	0	—
1	1	—

初期値は Q=0。

図は入力安定した後の値を示す。

Q = 0 ① R=1 が入るので、上の NOR の出力は Q=0 になる。

最初の記憶回路：SR ラッチ

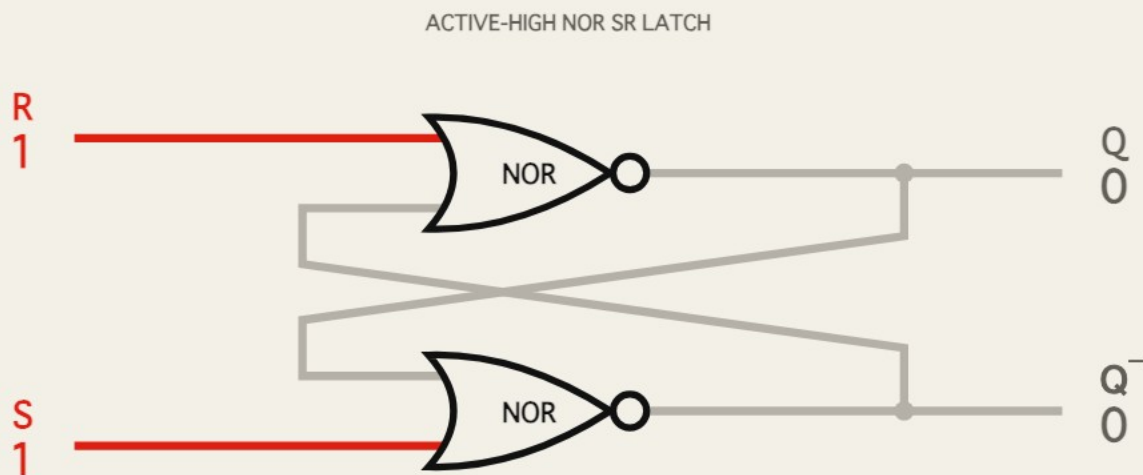
1 回路の名前**2 状態遷移表****3 外から使う記号**

Set で 1、Reset で 0 を書く。この保持回路を SR ラッチと呼ぶ。

禁止入力：二つの出力が同じになる

通常書き込み：S=1, R=0

禁止入力：S=1, R=1



二つの NOR の出力

$Q=0$

$\bar{Q}=0$

入力に 1 があるため、両方とも 0 になる。

禁止入力では、Q と $\bar{Q}$ が互いに反対という関係が崩れる。

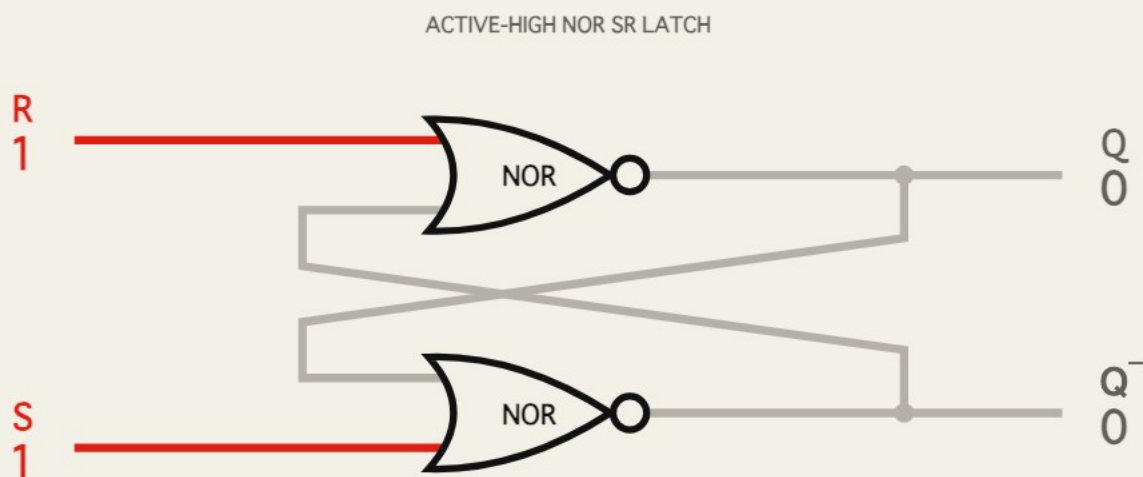
禁止入力を戻しても、結果は保証できない

解除前：S=R=1

解除：上が先に変化

解除：下が先に変化

解除：差がごく小さい



S と R を、両方 0 に戻すと

結果を予想する

二つの出力が 0 から変わろうとする。

禁止入力の解除後は、最終値も、安定するまでの時間も保証できない。

「？」は第三の論理値ではなく、まだ 0 / 1 として保証できない状態を表す。図は概念模型である。

禁止入力を作らない方法

SR LATCH

S と R を別々に操作

11?



D LATCH

データ D

書き込み許可 E

D / E

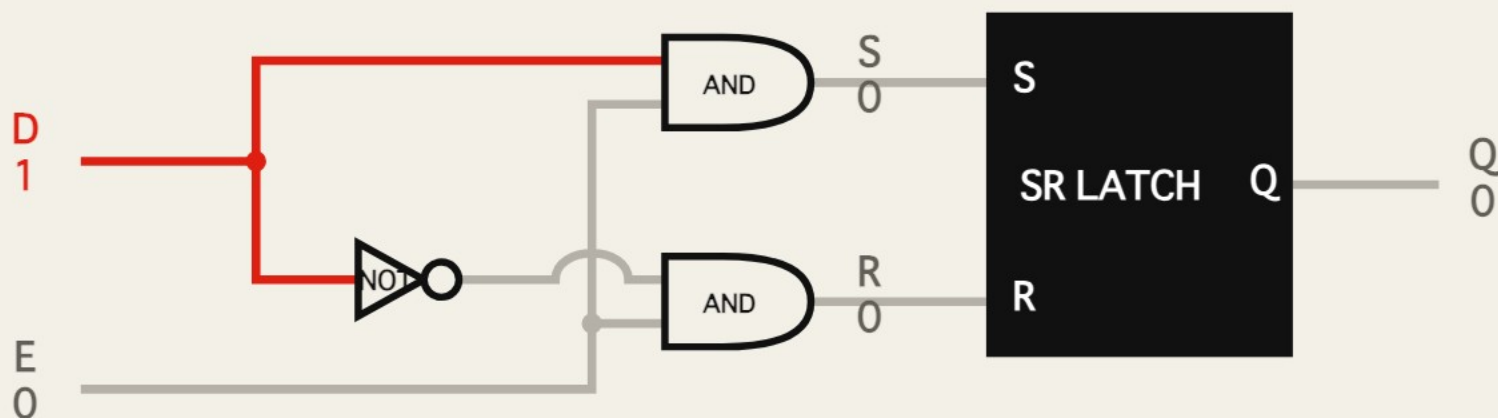
D と E を変えて、動きを確認める

D を 0 にする

E=1 にして開く

最初から

D=1 / E=0



入力から作る制御

$$S = E \cdot D$$

$$R = E \cdot \text{NOT}(D)$$

E	D	次の状態 Q(t+1)
0	任意	Q(t)
1	0	0
1	1	1

Q=0

E=0 : D を変えても、前の Q が残る
 $Q(t+1) = Q(t) = 0$

まず E=1 で 1 を書き、E=0 に戻してから D を変えてみる。初期値は Q=0。

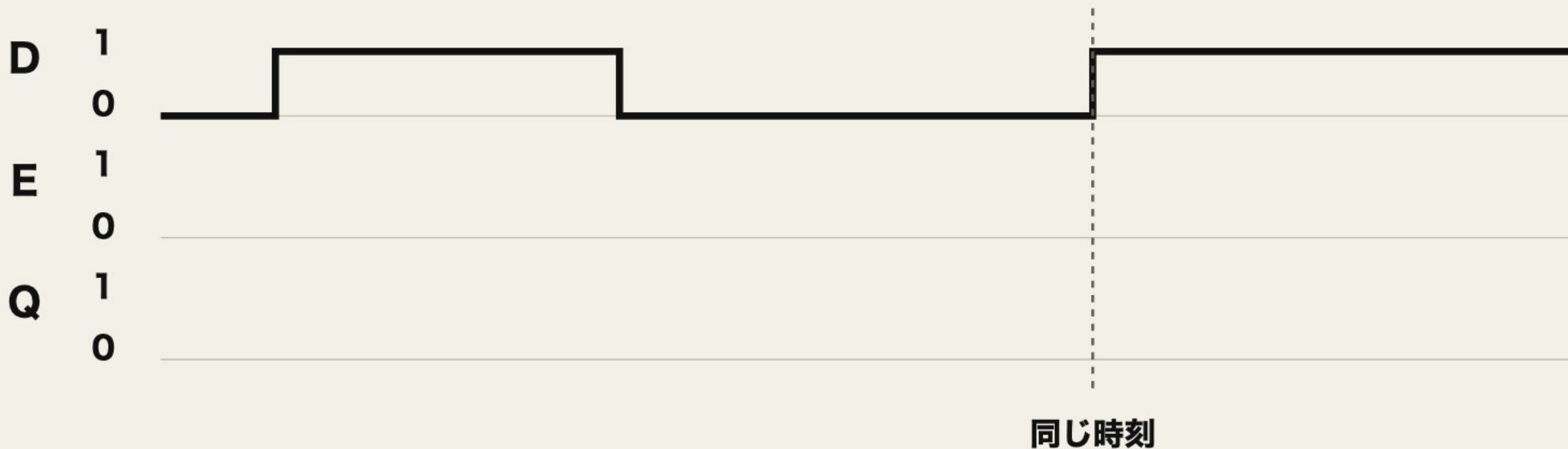
波形図で D・E・Q を順に読む

横は時間 → 上が 1、下が 0。同じ縦線は、同じ時刻。

1 D を読む

2 E の期間を重ねる

3 Q を確かめる

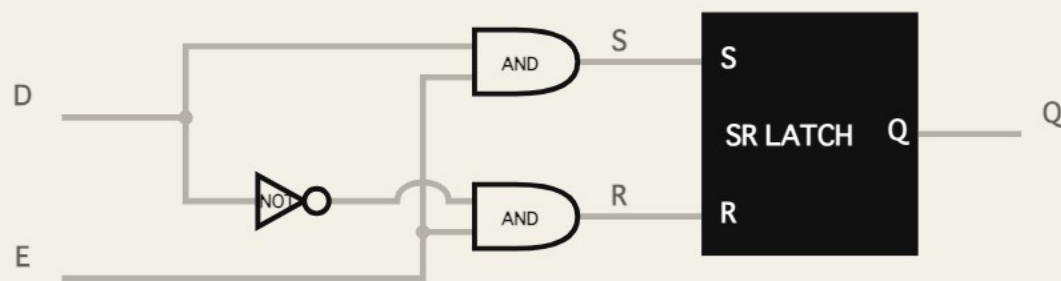


D の値が変わる時刻を、左から右へたどる

伝搬遅延を省略した論理的な波形。最初の Q は 0 とする。

この回路を D ラッチとして使う

INSIDE



D と E から、安全な S / R を作る

一つの記号へ



SYMBOL



E=1 Q は D を追う

E=0 Q を保持

開きっぱなしだと、1 が通り抜ける

E=1 にして開く

信号を一段進める

D を 0 にする

最初から



いまの状態 **000** E=0 にすると、その時点の値が残る

同じ E で全段を開くと、「一回で一段だけ」は保証できない

伝わる途中を一段ずつ止めた模式図。実際の伝搬時間を表すものではない。

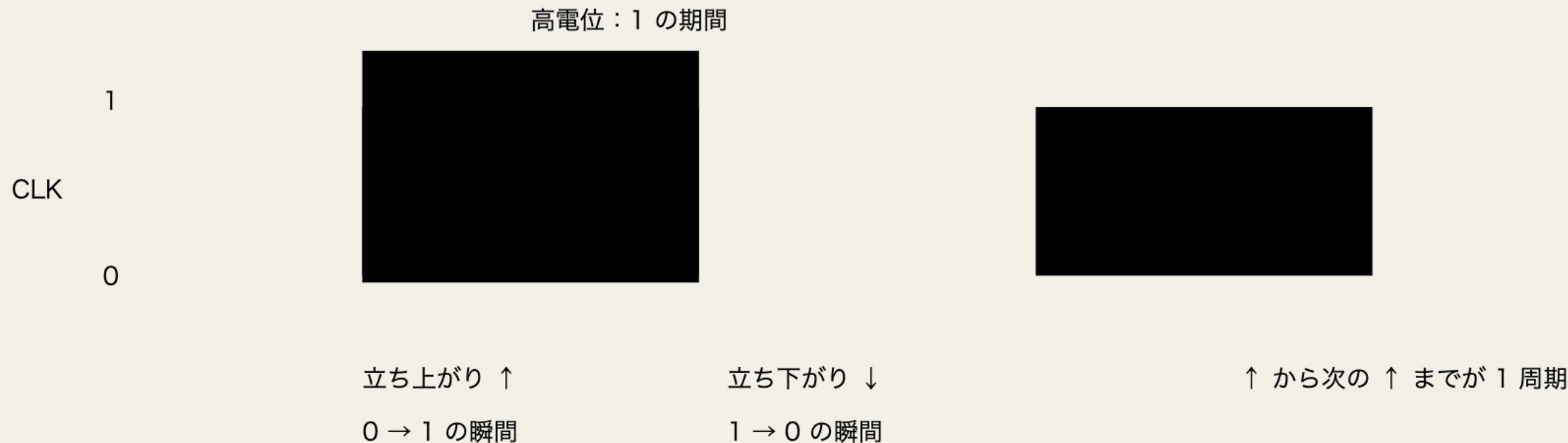
更新する時刻をそろえる



同時に開いたラッチは、値を通し続ける。更新する瞬間をそろえたい。

クロックは、0 と 1 を繰り返す信号

クロック (CLK) は、0 と 1 を周期的に繰り返す信号。



ラッチで使った E=1 開いている「期間」

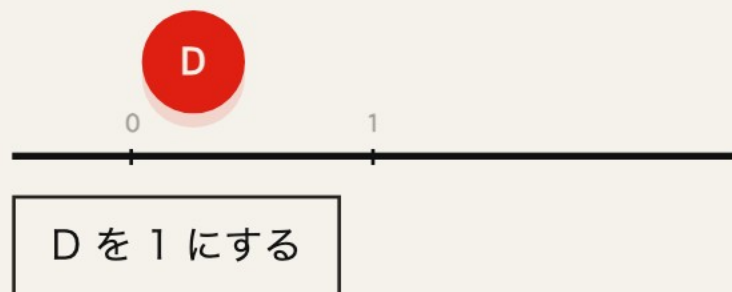
→

これから使う CLK ↑ 取り込む「瞬間」

立ち上がりを、記憶のシャッターにする

LIVE / COMBINATIONAL LOGIC

入力に応じて結果が変わる



この瞬間を撮る

PHOTO / REGISTER

撮った値を保持する



クロックは、計算する合図ではない

「今の結果を記憶せよ」という、回路全体に共通するシャッター

動く被写体 その時点の計算結果 D

シャッター 立ち上がりエッジ

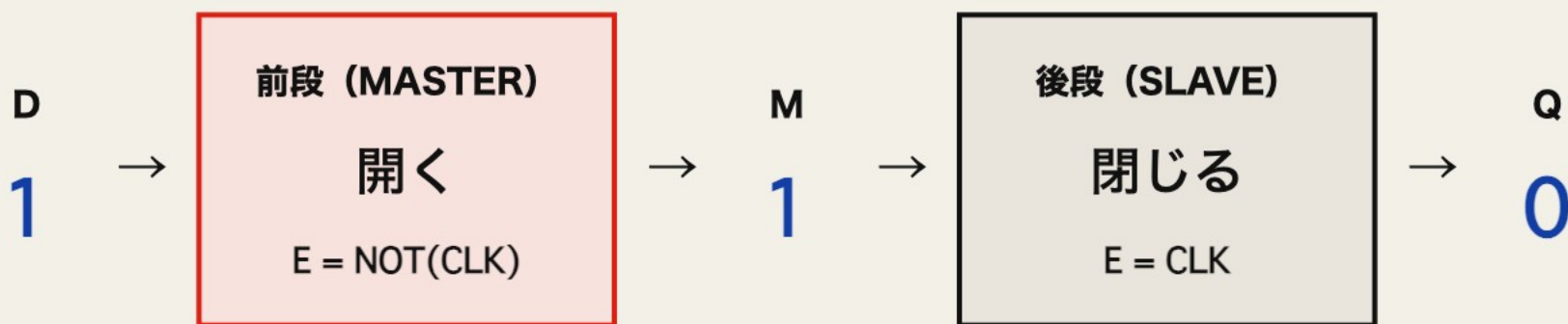
写真 保持された状態 Q

二つのラッチを、交互に開く

CLK を 0→1 にする

最初から

1 / 5



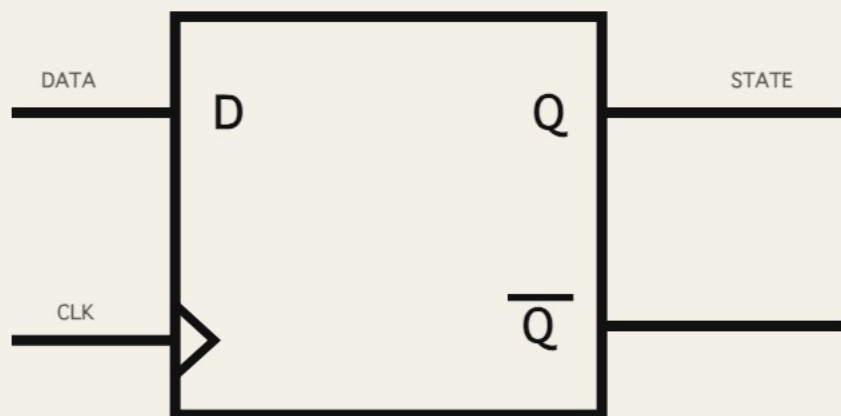
CLK=0 低電位では前段が D を受け取り、後段は Q を守る

操作	準備	CLK ↑	D を変更	CLK ↓	次の CLK ↑
D	1	—	—	—	—
CLK	0	—	—	—	—
M	1	—	—	—	—
Q	0	—	—	—	—

DFF の一つの実現方法。電源があり、必要なタイミング条件を満たす場合の論理モデル。

D フリップフロップ

STANDARD SYMBOL



NAME

D 型フリップフロップ

D Flip-Flop / DFF

記憶する量

1 bit

取り込む時刻

CLK ↑

それ以外

Q を保持

CLK ↑ $Q(t+1) = D$

それ以外は $Q(t+1) = Q(t)$

ラッチの「期間」と DFF の「瞬間」を比べる

→ 時間 上 1 下 0

D LATCH

開いている間、ずっと見る



まず、共通の入力 D の変化を読む

1 D を見る

2 時刻を加える

3 Q を比べる

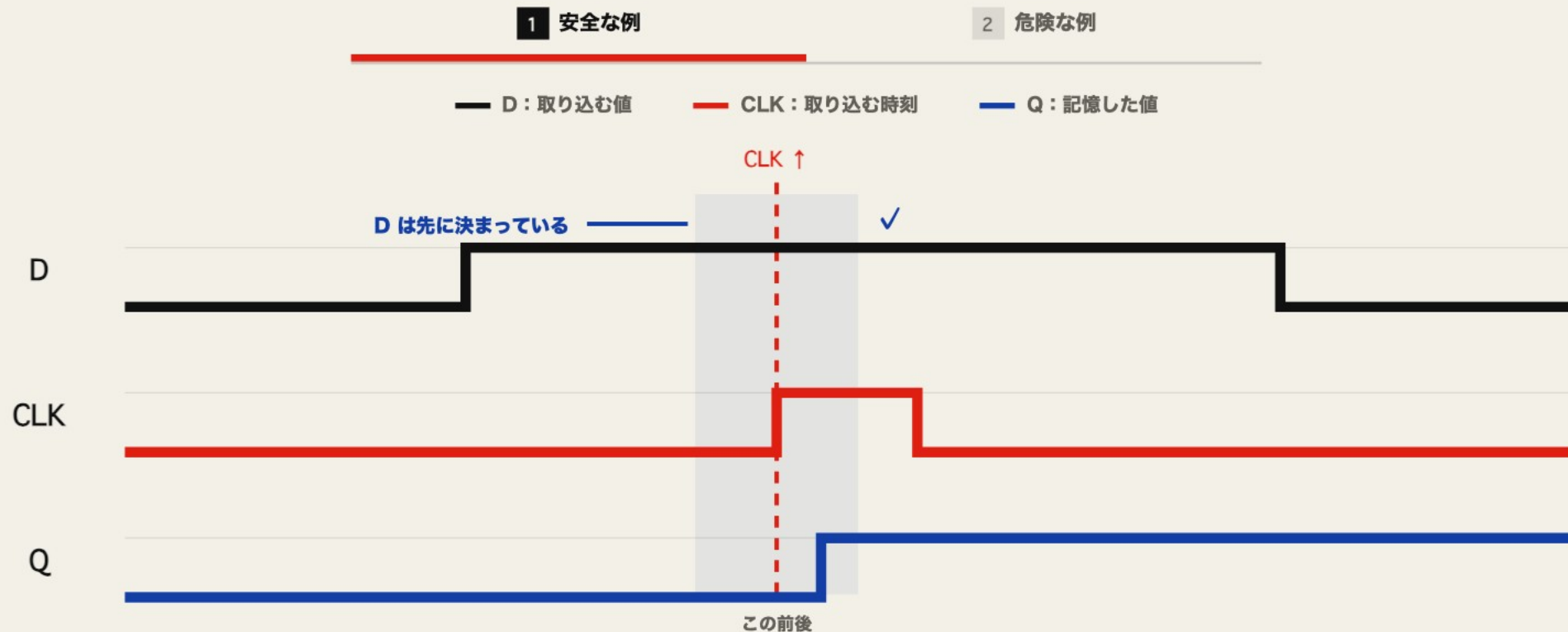
D FLIP-FLOP

決めた瞬間だけ見る



同じ D を、別の取り込み方で比べる

CLK ↑ の前後では、D を変えない



安全

D が先に安定していれば、Q は正しく更新される

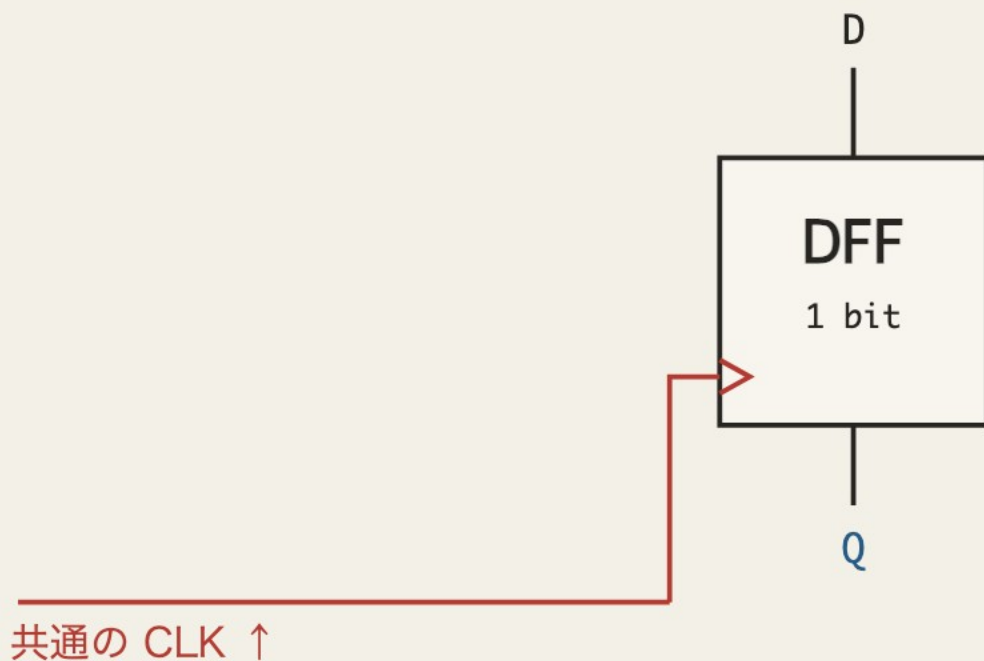
複数 bit をひとまとめにする



計算結果は一つの bit では足りない。複数 bit をまとめて残したい。

DFF を並べて、4 bit にする

同じ回路を、4 個並べる

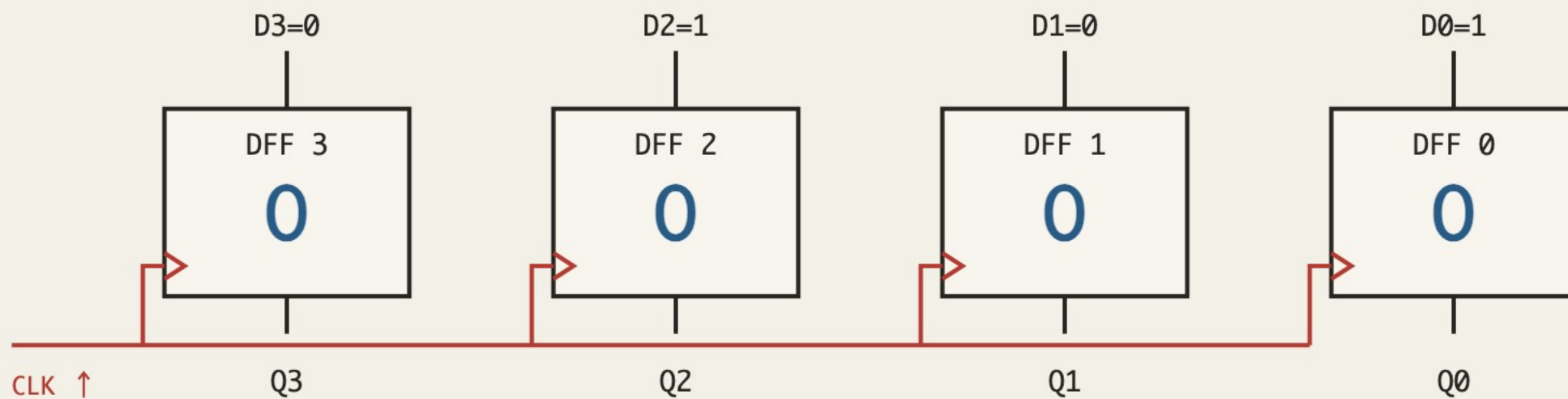


DFF 一つが、1 bit を保持する。

複数の bit をまとめて保持する回路を、レジスタと呼ぶ。

4 bit を同じ瞬間に取り込む

入力 D **0** **1** **0** **1** **CLK ↑ を送る** **最初から**

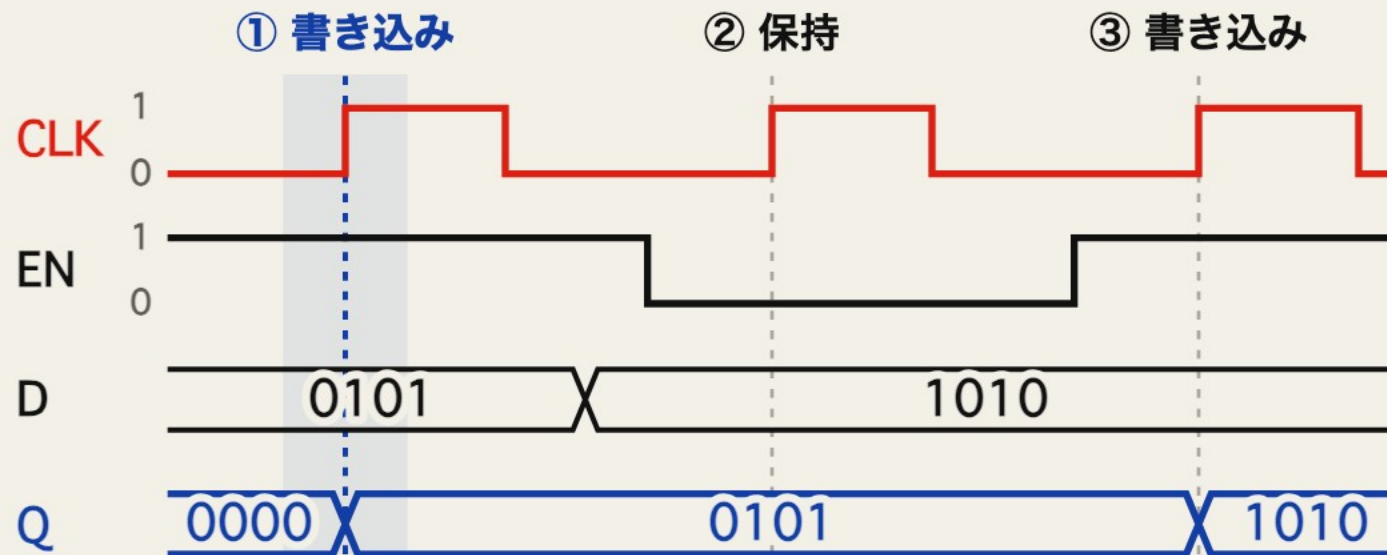


予想：入力 D を変えただけで、Q も変わるだろうか。

入力を変えても Q は保持される。次の CLK ↑ で、4 bit をまとめて取り込む。

EN=1 のときだけ、CLK ↑ で取り込む

横は時間、縦の点線は同じ瞬間。



① EN=1 : 書き込み

② EN=0 : 保持

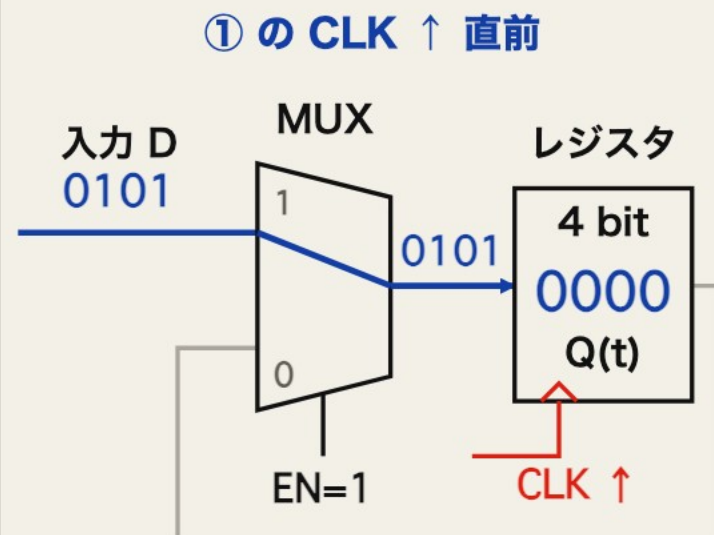
③ EN=1 : 再び書き込み

立ち上がり時: EN=1 D=0101

MUX が新しい D を選ぶ。立ち上がりで 0101 を取り込む。

EN は入力を選ぶ。取り込む瞬間を決めるのは CLK ↑。

D・Q は 4 bit をまとめて表示



Q 0000 → 0101

初期値 Q=0000 / 伝搬遅延は省略

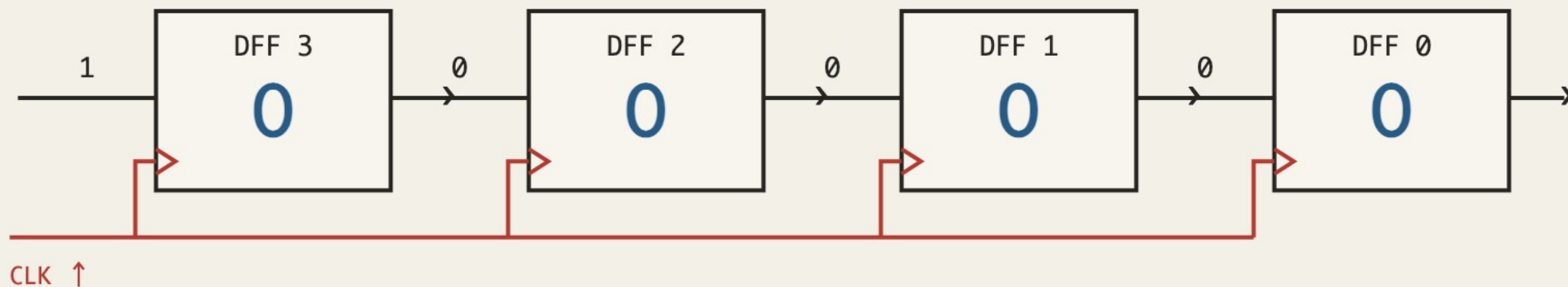
同じ CLK ↑ で、一段ずつ隣へ送る

先頭の D

1

CLK ↑ を送る

最初から

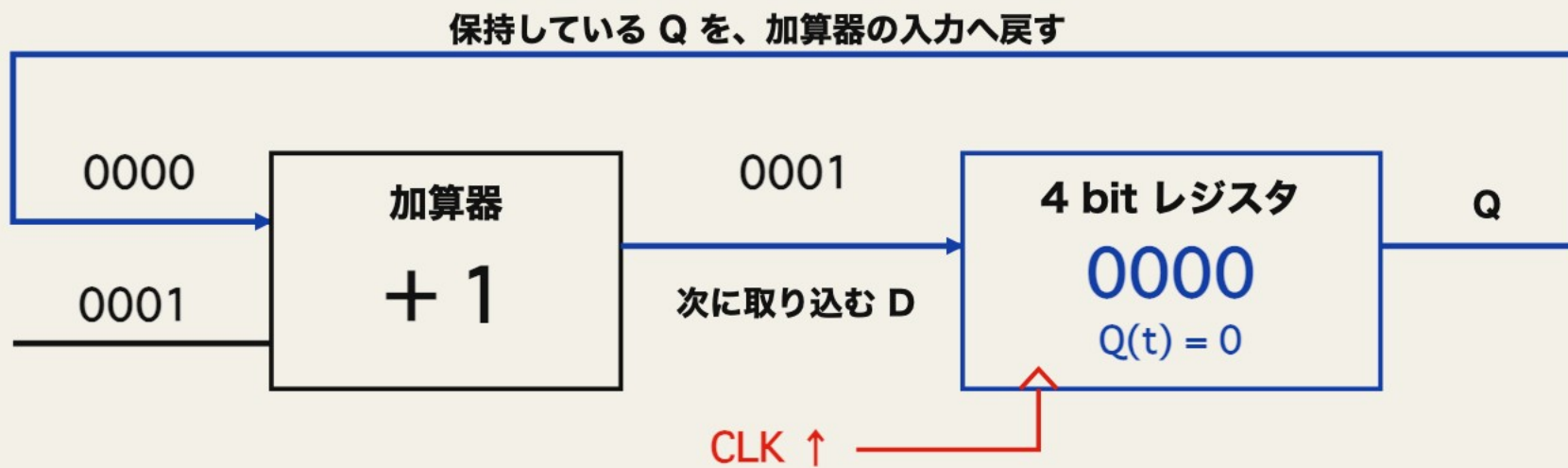


予想：次の CLK ↑ で、1 はどこまで進むだろうか。

全段が同じ時刻の「前の Q」を取り込むため、1 回の CLK ↑ で一段だけ進む。

足した結果を戻すと、数を数えられる

第3回の加算器 + 今回のレジスタ。Q に 1 を足し、次の CLK ↑ で取り込む。



CLK ↑ を送る

最初から

数えた値 (10進数) 0

CLK ↑ $Q(t+1) = Q(t) + 1$

次の立ち上がりまでは Q を保持

カウンタ 次回は、この仕組みで「次に読む命令の場所」を進める。

4 bit の例：15 の次は 0。加算器の結果のうち下位 4 bit を保存する。

カウンタを三つつなぐと、時計になる

1 秒ごとに同じ CLK ↑。一周する瞬間だけ、左のカウンタも進める。



1 秒進める

動かす

23:59:58 へ

最初から

停止中

START

12:34:58

1 秒ずつ進めて、59 → 00 のときの「分」を見る。

各カウンタ : EN=0 なら保持。EN=1 なら「上限で 0、それ以外は+1」。

4 bit を、一つのワードとして読む

4-BIT REGISTER

Q3	Q2	Q1	Q0
0	1	0	1

4 bit

この枠全体で 1 word

0101

四つを別々ではなく、まとめて読む

$4 + 1 = 5$

CPU のビット幅

4 bit 授業の小さな模型

→ 現実の CPU では、もっと広い値を扱う

32 bit

64 bit

基本的なレジスタの幅

64 bit

8 bit

8 bit

8 bit

8 bit

8 bit

8 bit

8 bit

8 bit

64 bit = 8 byte

REGISTER

64 bit の値を保持

ALU

64 bit を基本に計算

CPU ARCHITECTURE

64 bit が設計の基本幅

ここでは CPU の基本的なデータ幅を比べる。「word」という語が指す幅は、命令セットによって異なる。

アドレスで保存場所を選ぶ



一つのワードだけでは足りない。たくさんの保存場所から、一つを選びたい。

アドレスで、一つのワードを選ぶ



読みたい場所を、2 bit のアドレスで指定する

同じ場所を、読むか書き換える



10 のワード線 だけを開き、同じビット線 を使って読む／書く

レジスタを並べ、アドレスで選ぶ

二つのワードを読む

一つのワードを書く

レジスタファイル					この模型：1 行 = 4 DFF = 1 ワード				
	DFF	DFF	DFF	DFF		DFF	DFF	DFF	DFF
R0	0	0	0	0		0	0	0	0
R1	0	0	1	1		0	0	1	1
R2	0	1	0	1		0	1	0	1
R3	0	0	0	0		0	0	0	0

二つの読み出し口

読出 A アドレス R1

0011

読出 B アドレス R2

0101

ALU $3 + 5 = 8$

1000

レジスタファイルは、演算に使う少数のワードを、アドレスで選ぶ記憶回路

図は DFF を並べた学習用の構成。実際のレジスタファイルには専用の記憶セルを使う設計もある。

大量に保存するなら、1 bit を小さくしたい

さきほどの模型

4 bit × 4 行

16 個の DFF で、16 bit を保持

容量を増やす

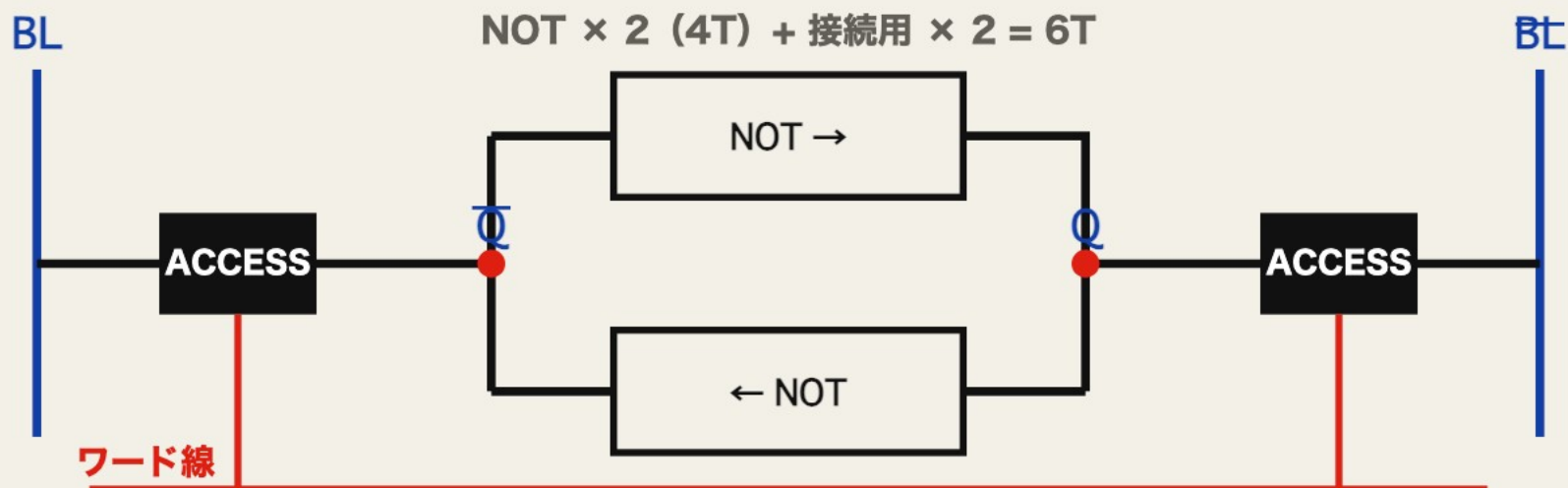
→ **1,000,000 bit**

同じ作り方なら、100 万個の DFF が必要

すべての bit に、同じ大きな回路が必要だろうか？

場所をアドレスで選ぶ仕組みはそのまま使い、
中の記憶セルを、より小さな構成に変える。

SRAM：相互フィードバックで保つ



保持

相互フィードバック

互いの出力が、安定した状態を支える

読み書き

接続用の 2T

WL でセルをビット線へつなぐ

主な用途

キャッシュ

電源中はリフレッシュ不要

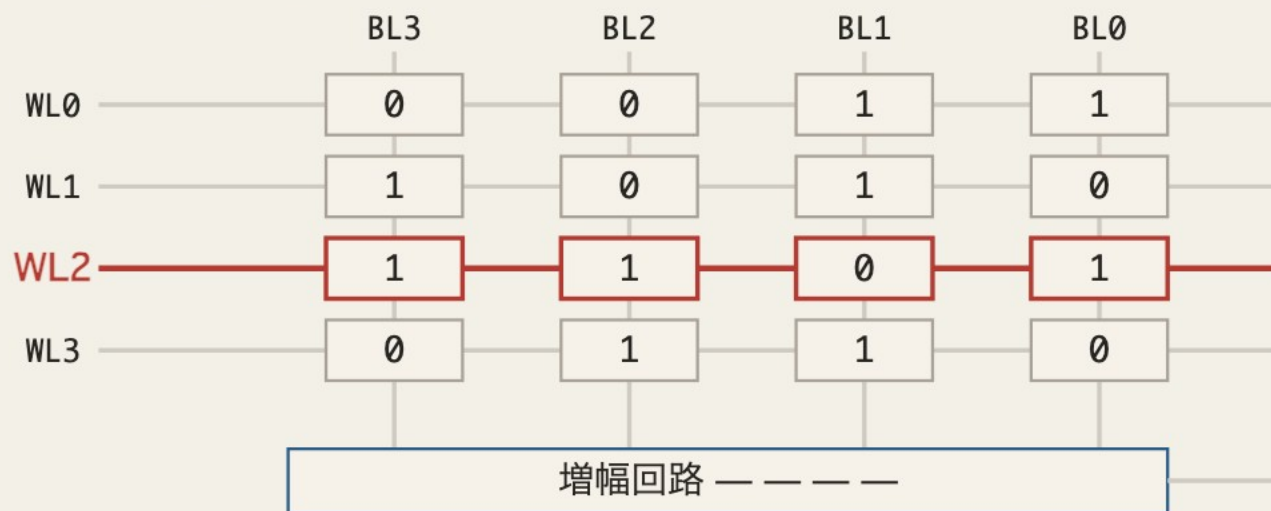
SRAM：ワード線で、セルを外につなぐ

1 行を選ぶ

2 セルとビット線を接続

3 差を増幅して読む

行アドレス 00 01 **10** 11 デコーダが WL2 を選ぶ



1 セル = 1 bit

典型例：6T

WL：行を開く線

BL：値を伝える線

読み出す値

WL が、どの行を外につなぐか決める。

接続関係の模式図。BL と反転側の BL を一本にまとめ、列選択を省略している。

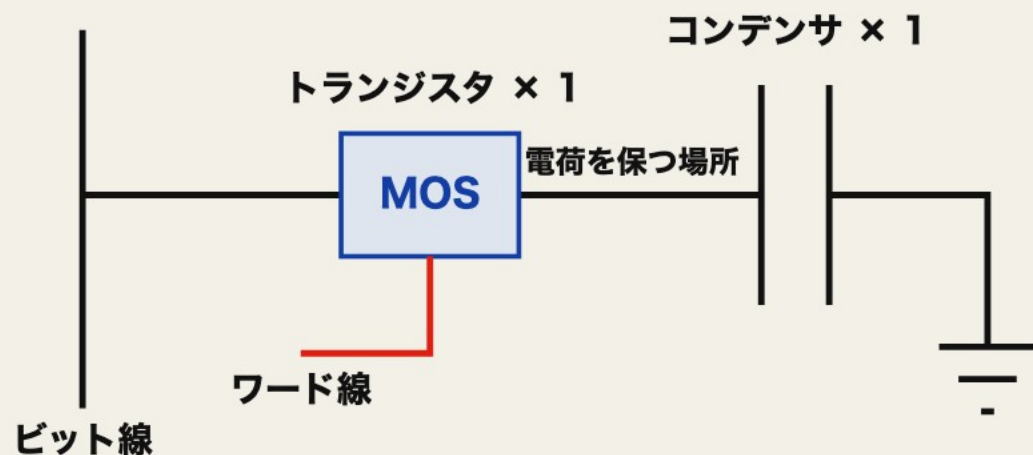
DRAM：コンデンサの電荷で保つ

0 を書く

1 を書く

時間を進める

リフレッシュ



読み取れる値

1

0 も 1 も、その値のまま保つ

読み分ける余裕（模式）

100%

1 を書いた。時間がたつと、読み分ける余裕が減る。

リフレッシュ 失う前に読む → 同じ値を書き戻す

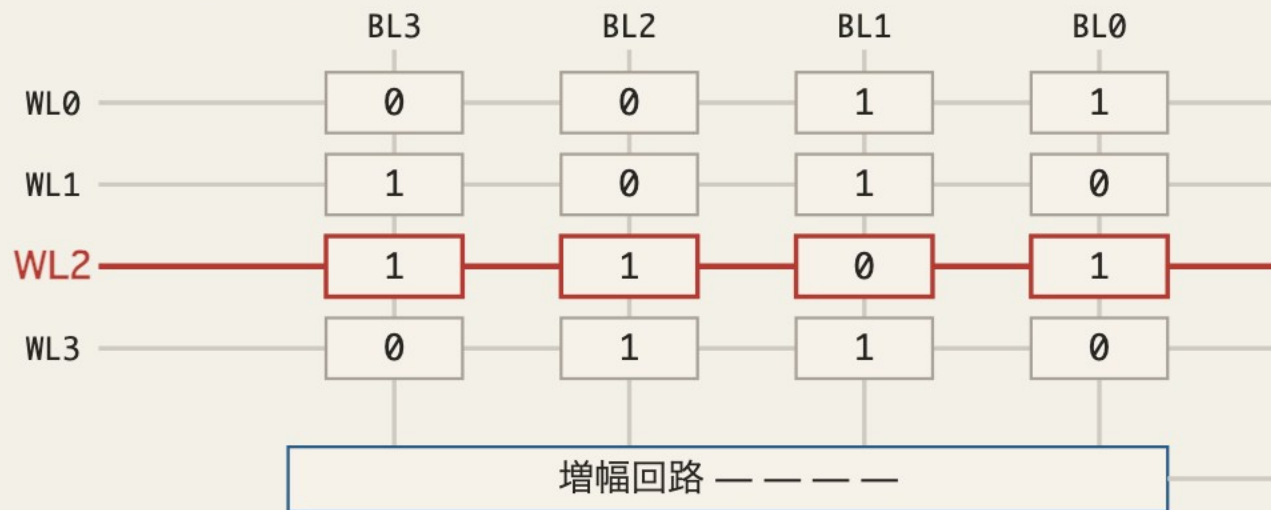
割合は読み分ける余裕の概念模型。電荷量や実際の時間・しきい値を示すものではない。

DRAM：行を読み、書き戻してから列を選ぶ

1 行を開く

2 増幅して書き戻す

3 列を選んで取り出す

 行アドレス 00 01 **10** 11 デコーダが WL2 を選ぶ

 1 セル = 1 bit
 典型例：1T + 1C

 WL：行を開く線
 BL：値を伝える線

 列アドレス 00 01 **10** 11 まず、行全体を開く

行全体を開き、各セルの電荷を BL と分け合う。

接続関係の模式図。読んだ際に弱まる電荷を補う書き戻しであり、すでに失われたデータを復元する処理ではない。

速さと容量に合わせて、記憶回路を使い分ける

	レジスタ	SRAM	DRAM
1 bit の保持	DFF など	二つの NOT の 相互フィードバック	コンデンサの電荷
回路の大きさ	大きい	典型例：6T	典型例：1T + 1C
定期的な補い直し	不要	不要	リフレッシュが必要
主な役割	演算のすぐそばで 少量の値を保持	キャッシュで 高速に供給	主記憶で 大量に保存

← 短い待ち時間を重視

容量・1 bit あたりの安さを重視 →

速く取り出したい値と、大量に置きたい値があるため、組み合わせて使う。

典型的な構成と傾向の比較。速度や面積の比率は、製造技術・回路・配置によって異なる。いずれも揮発性である。

残した計算結果を、次の計算に使える

CLK ↑ で結果を保存

最初から 0 / 3

A 0101

B 0011



組合せ回路：ALU

1000 $5 + 3 = 8$

CLK ↑



取り込む

記憶回路：レジスタ

0000

まだ取り込んでいない

↑ ————— 保存した 1000 を、ALU の入力 A へ戻す ————— ↵

冒頭と同じ計算：0101 + 0011 = 1000。この結果を残せるだろうか。

この間、レジスタへ次の書き込みは行わない。長く大量に保存したい値は、アドレスを指定してメモリへ書く。

何を加えると、何ができるようになったか

状態を保つ

フィードバック

→

書く値を決める

D ラッチ

→

更新時刻をそろえる

DFF

→

bit を束ねる

レジスタ

アドレスで場所を選ぶ

保持するセルの方式は、一つではない。

SRAM

相互フィードバックで保つ

DRAM

電荷で保ち、周期的に補い直す

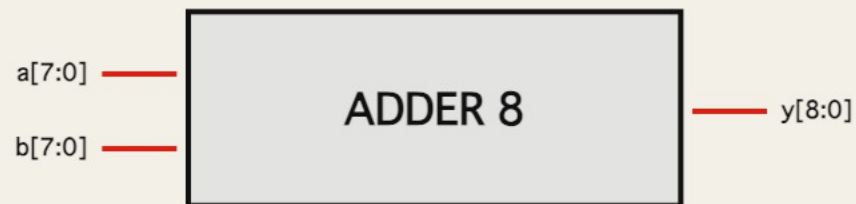
確認：DFF の $Q=1$ 。D を 0 にしたが、CLK $\uparrow$ はまだ来ていない。Q はどうなる？

理由と答えを確かめる

回路図を、コードでも表せる

```
module adder8(  
    input  [7:0] a, b,  
    output [8:0] y  
);  
    assign y = a + b;  
endmodule
```

THE SAME 8-BIT ADDER



module	回路という「箱」
input / output	箱の入出力端子
assign	組合せ回路の接続と計算

VERILOG 入門 learn.c2lab.org/computer-systems/2026/interactive/circuit/verilog.html ↗

出力が変わる条件を、コードで書き分ける

COMBINATIONAL / 組合せ回路

入力が変われば、出力も変わる

```
assign y = a + b;
```

INPUT CHANGE



y UPDATE

クロックなし・前の値を記憶しない

SEQUENTIAL / 順序回路

CLK ↑ の瞬間だけ、状態を更新する

```
always @(posedge clk)  
  q <= d;
```

CLK ↑



q UPDATE

次の CLK ↑ まで、前の値を保持する

演習2 A-D: 組合せ回路 → E: 組合せ回路+順序回路 入門を開く →

CPU の中で、記憶した値をどう使うか

レジスタ

今使う値を保持する



ALU

入力から結果を作る



メモリ

アドレスで値を保存する

どの値を読み、何を計算し、どこへ書くか。
この順序を決める仕組みが、次のテーマである。

次回：カウンタで命令の場所を進め、部品をつないで CPU を作る