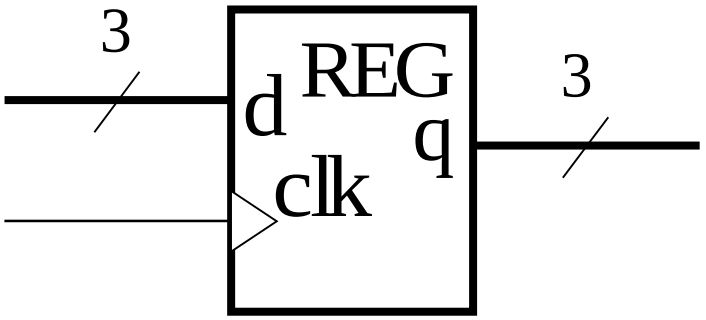
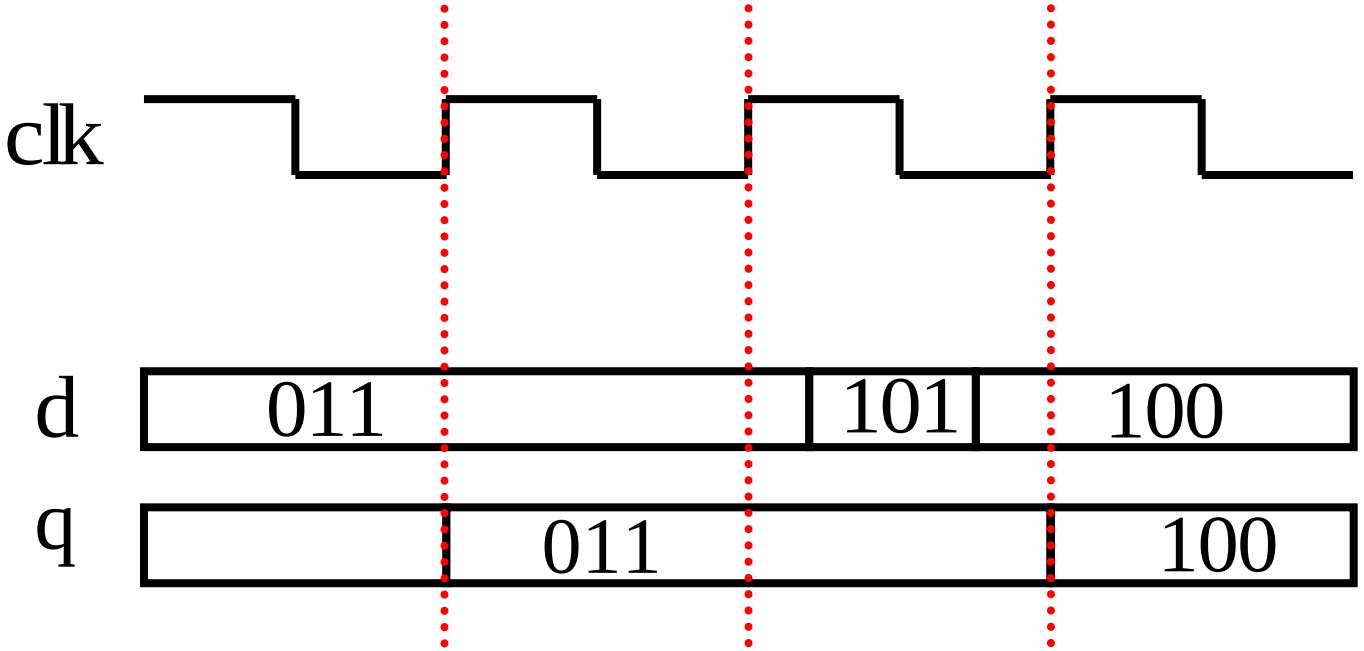
第3回 ハードウェア記述言語
～順序回路、ステートマシン～

中野 秀洋

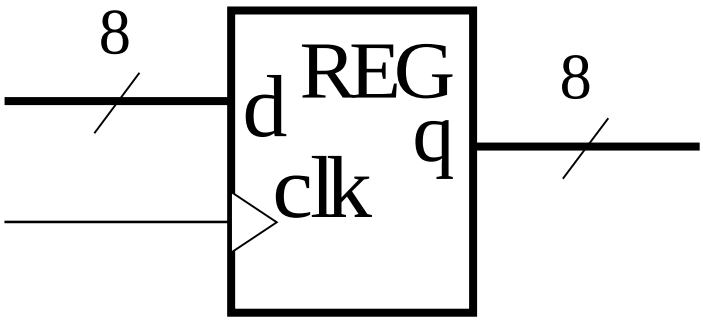
レジスタ



クロック信号が立ち上がるときに
入力値を書き込み



8bitレジスタ



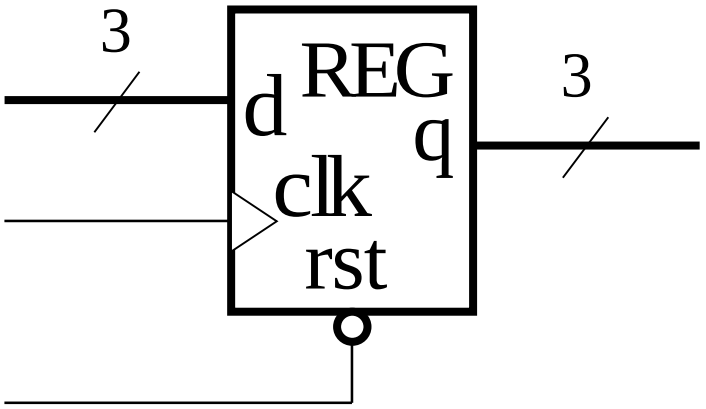

```
module reg8(clk, d, q);
    input         clk;
    input  [7:0] d;
    output [7:0] q;
    reg    [7:0] q;

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

posedge
　　信号の立ち上がり

negedge
　　信号の立ち下がり

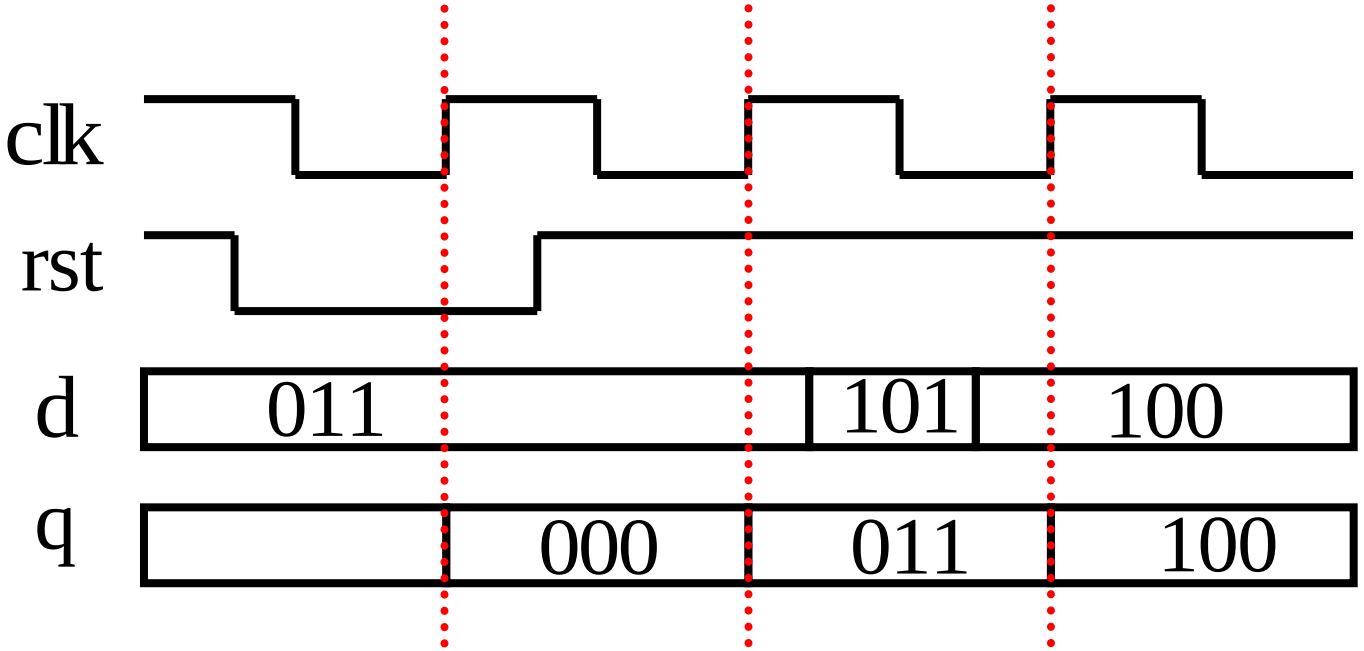
clk が立ち上がったときのみ always 文が評価される
（それ以外のときは q の値を保持）
q に d の値をセット

リセット付きレジスタ

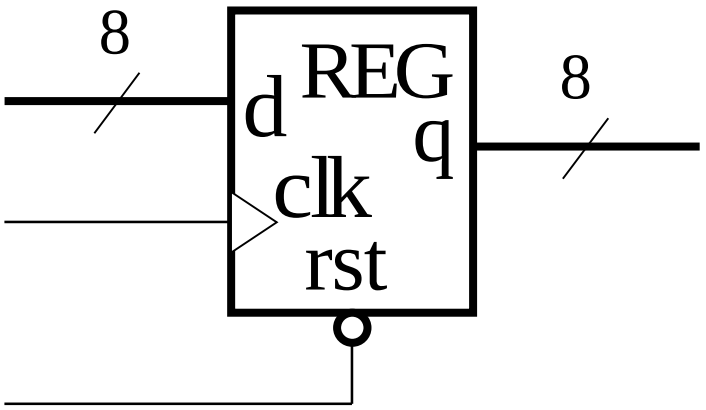


クロック信号が立ち上がるときに

リセット信号が立ち下がっていればリセット
そうでなければ入力値を書き込み



8bitリセット付きレジスタ



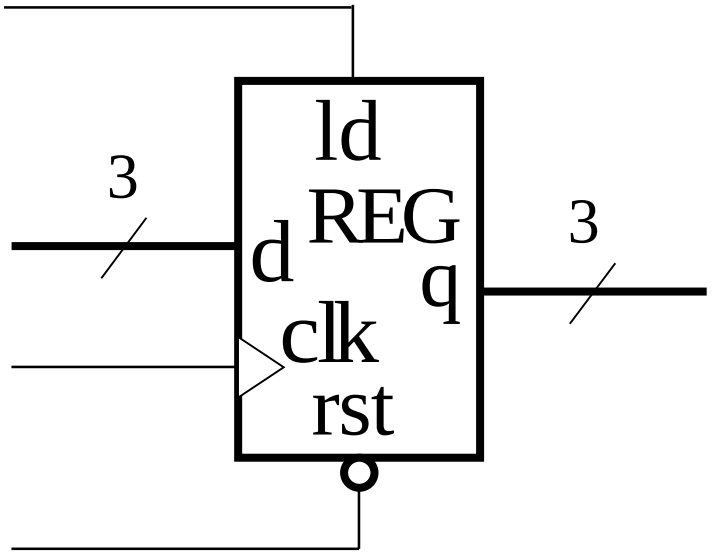

```
module reg8(clk, rst, d, q);
    input         clk, rst;
    input  [7:0] d;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(rst == 1'b0) begin
            q <= 8'b00000000;
        end else begin
            q <= d;
        end
    end
endmodule
```

posedge
　信号の立ち上がり

negedge
　信号の立ち下がり

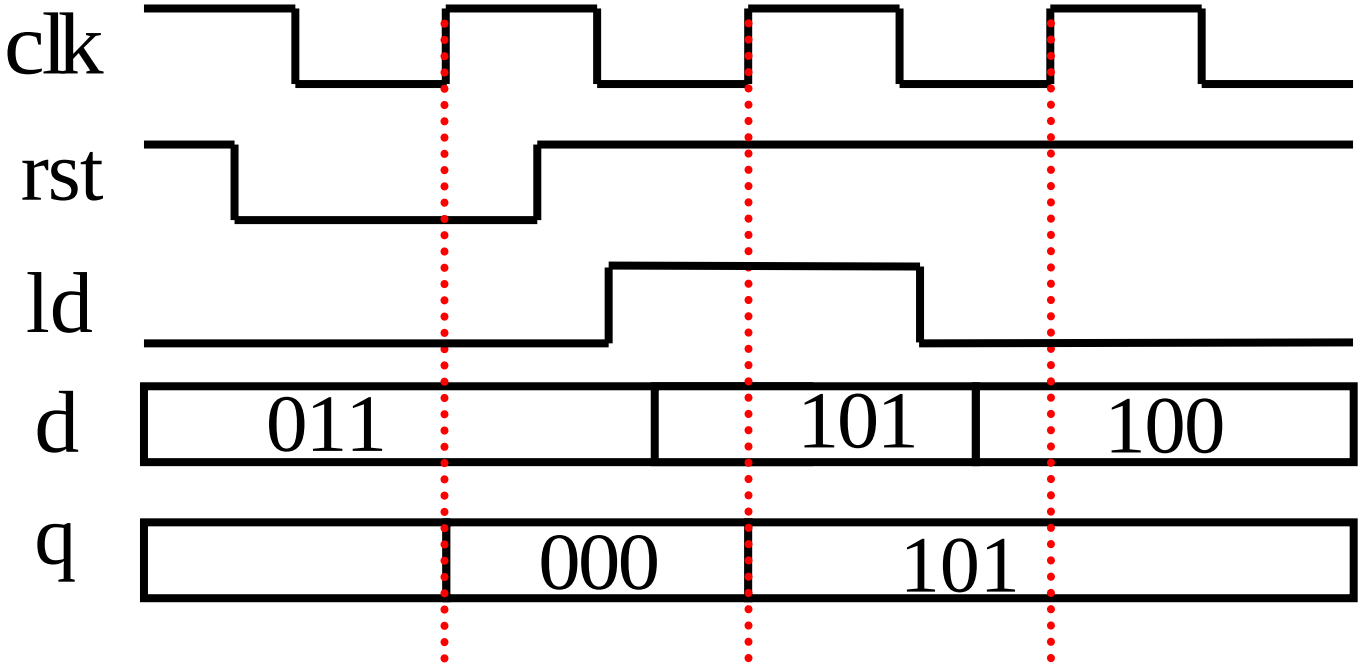
clk が立ち上がったときのみ always 文が評価される
（それ以外のときは q の値を保持）
rst が 0 なら q をリセット
そうでなければ q に d の値をセット

ロード付きレジスタ

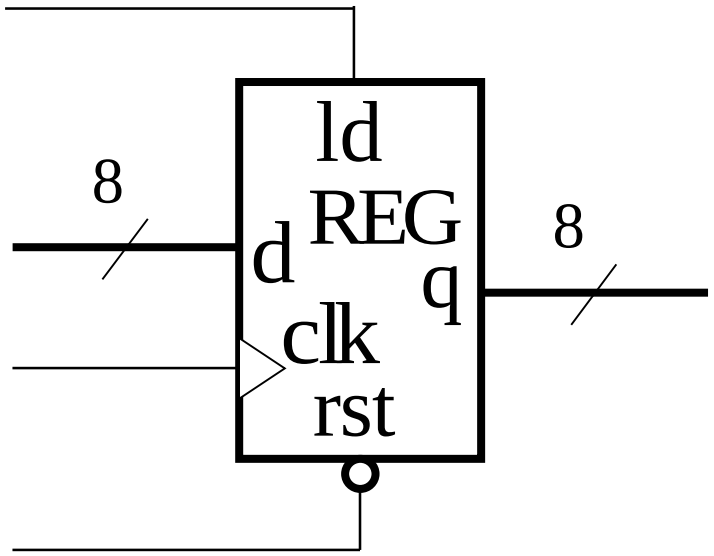


クロック信号が立ち上がるときに

リセット信号が立ち下がっていればリセット
ロード信号が立ち上がっていれば書き込み
そうでなければ値を保持



8bitロード付きレジスタ

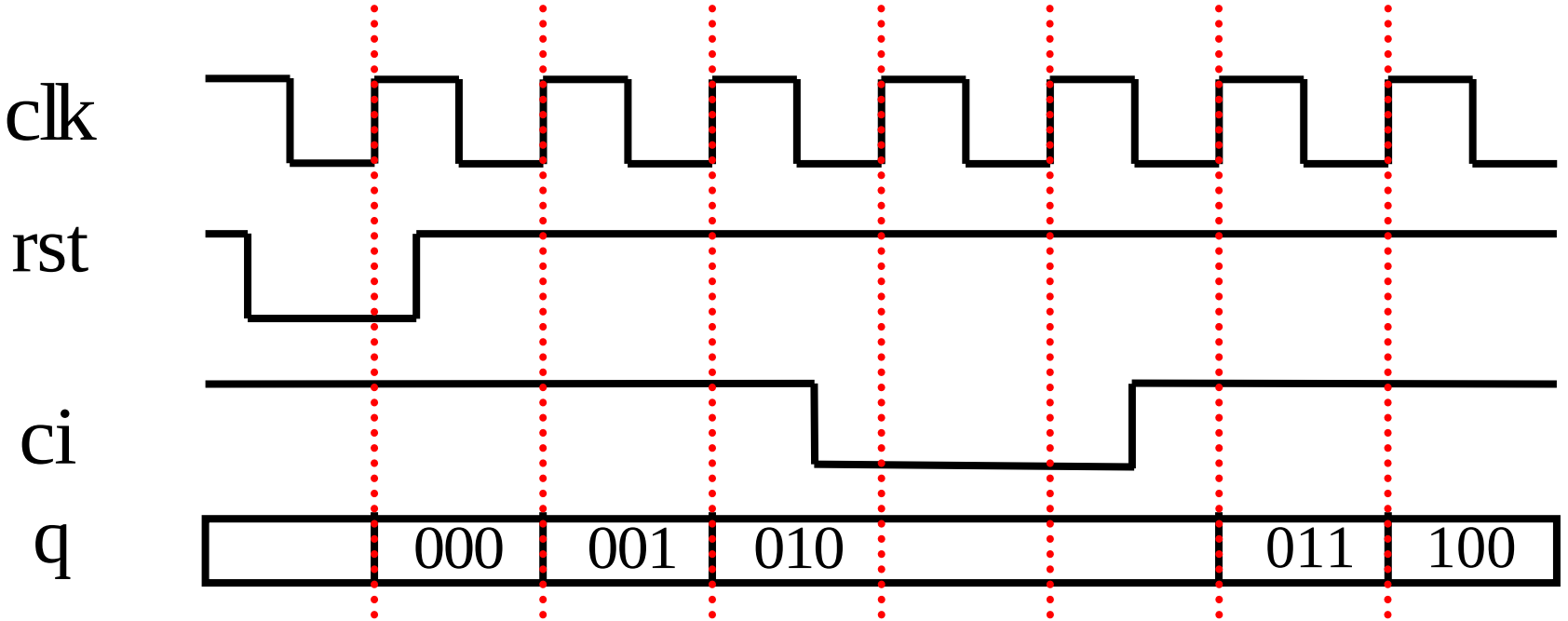



```
module reg8(clk, rst, ld, d, q);
    input         clk, rst, ld;
    input  [7:0] d;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(rst == 1'b0) begin
            q <= 8'b00000000;
        end else if(ld == 1'b1) begin
            q <= d;
        end
    end
endmodule
```

elseの記述を省略
どの条件も偽のとき
値は保持される

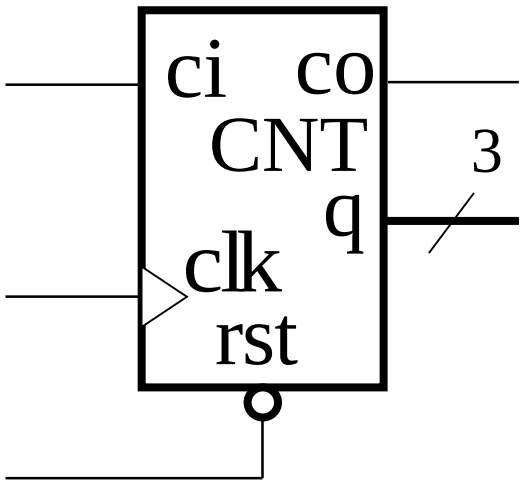
clk が立ち上がったときのみ always 文が評価される
（それ以外のときは q の値を保持）
rst が 0 なら q をリセット
ld が 1 なら q に d の値をセット
そうでなければ q の値を保持

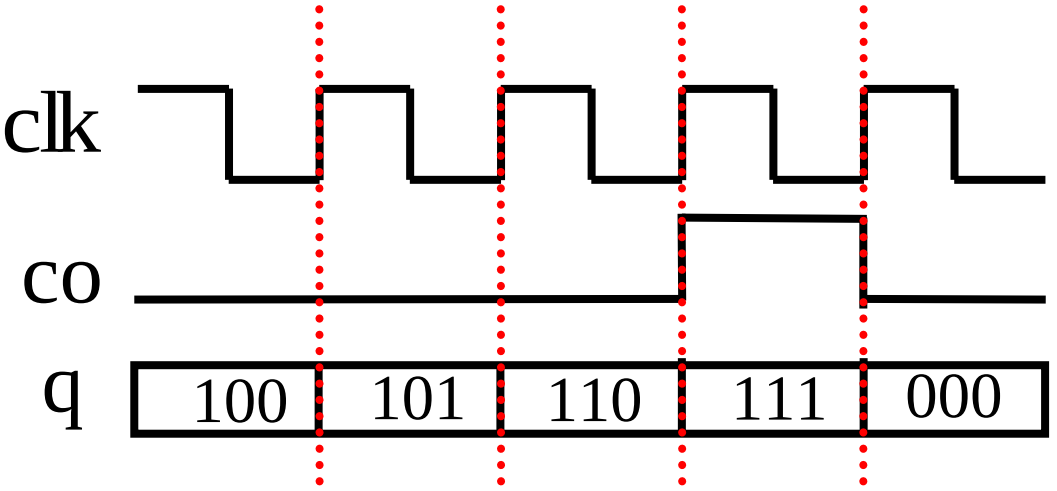
カウンタ



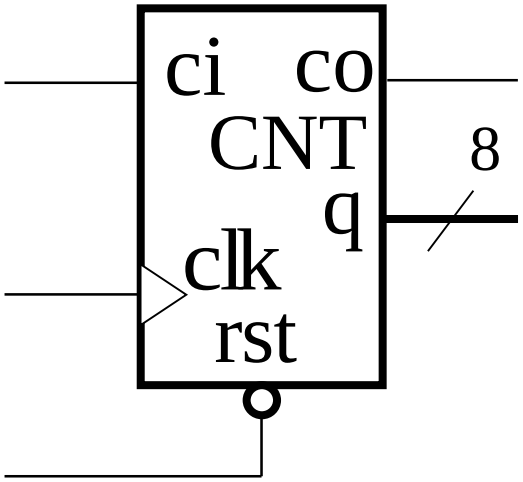
q の値に **ci の値を加算**（ci = 1ならインクリメント、ci = 0なら値を保持）

q の値が 0 に戻るとき **co = 1 を出力**





8bitカウンタ

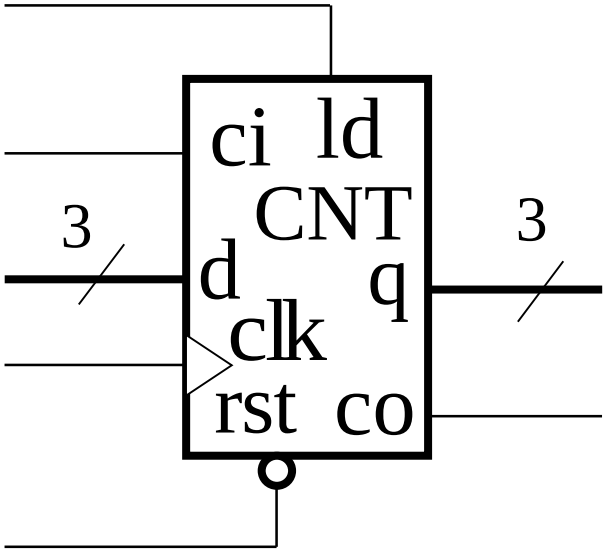


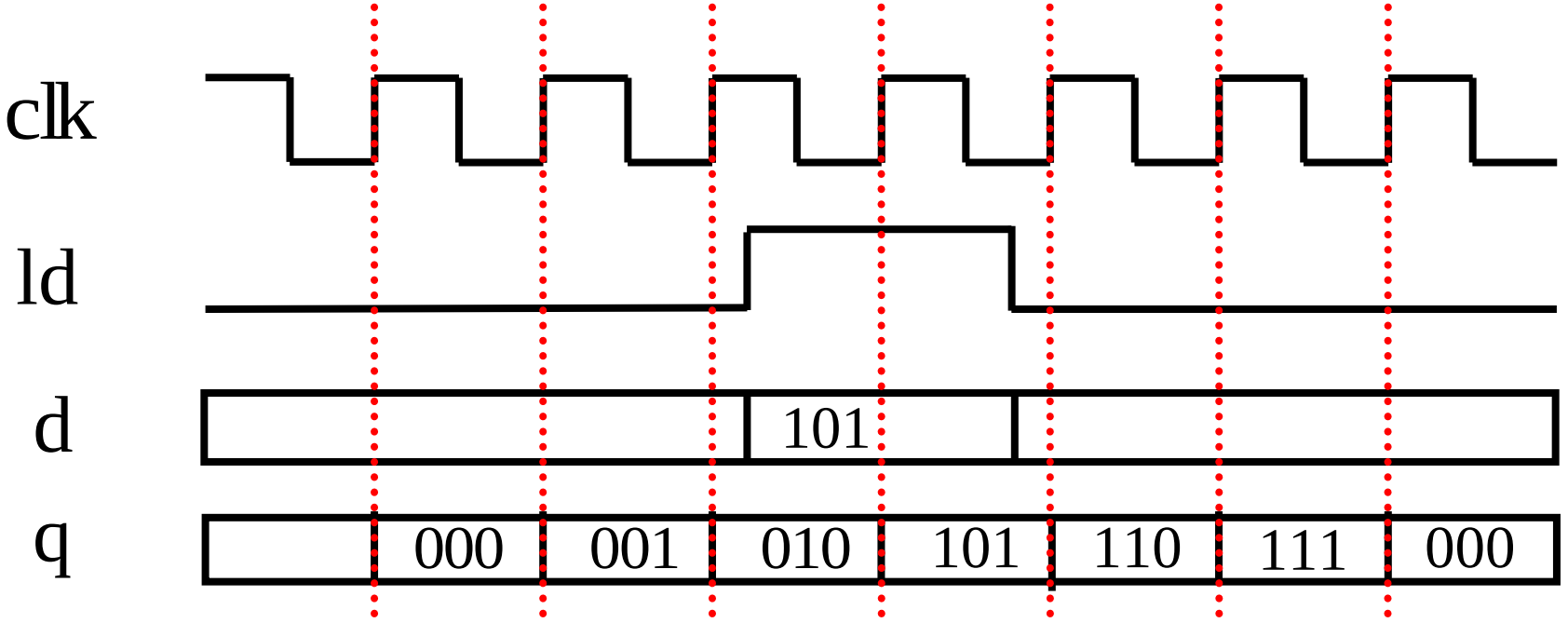

```
module cnt8(
    clk, rst, ci, co, q);

    input         clk, rst;
    input         ci;
    output        co;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(rst == 1'b0) begin
            q <= 8'b00000000;
        end else begin
            q <= q + ci;
        end
    end
    assign co = &{q, ci};
endmodule
```

リダクション演算
全ビットの論理積

rst = 0 のとき q = 0 にリセット
そうでなければ q + ci をセット
　ci = 1 ならインクリメント、ci = 0 なら値を保持
q = 11111111, ci = 1 のとき co = 1 を出力

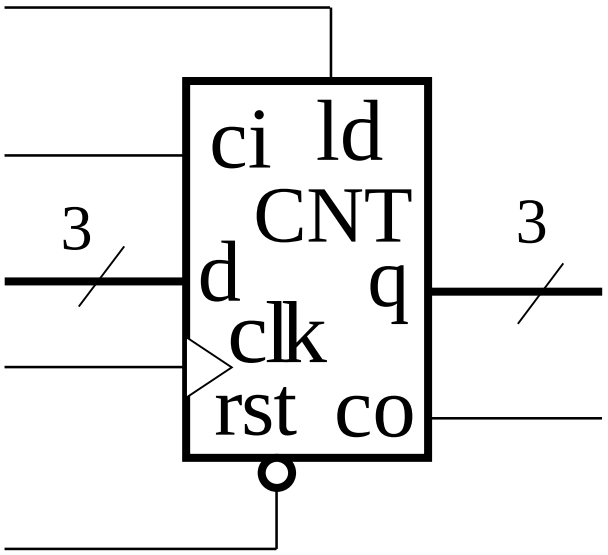
ロード付きカウンタ





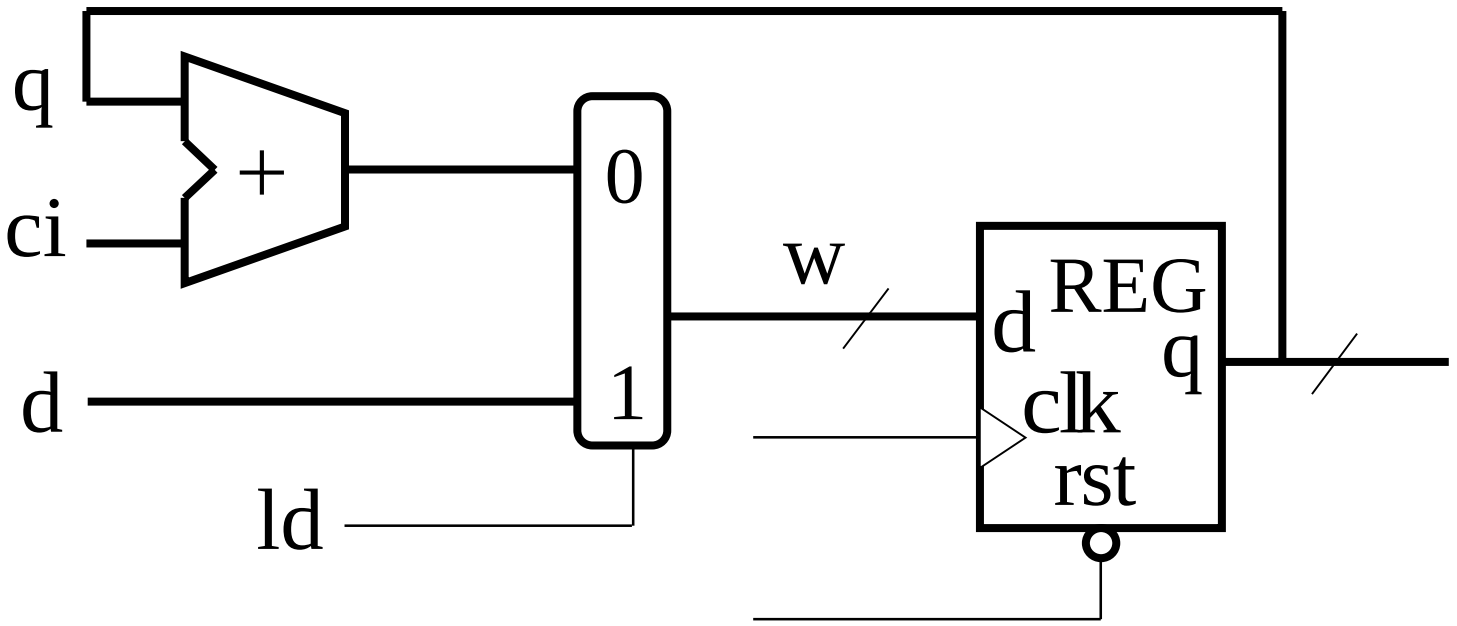
ld=1 なら d の値を q にセット

8bitロード付きカウンタ




```
module cnt8(
    clk, rst, ld, d, ci, co, q);

    input         clk, rst, ld;
    input  [7:0] d;
    input         ci;
    output        co;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(rst == 1'b0) begin
            q <= 8'b00000000;
        end else if(ld == 1’b1) begin
            q <= d;
        end else begin
            q <= q + ci;
        end
    end
    assign co = &{q, ci};
endmodule
```

レジスタとカウンタ（別の記法）




```
/* （リセット付き）レジスタ */
    assign w = d;
/* ロード付きレジスタ */
    assign w = (ld == 1’b1) ? d : q;
/* カウンタ */
    assign w = q + ci;
/* ロード付きカウンタ */
    assign w = (ld == 1’b1) ? d : q + ci;
```

```
assign w = ...;

always@(posedge clk) begin
    if(rst == 1'b0) begin
        q <= 8'b00000000;
    end else begin
        q <= w;
    end
end
```

10進カウンタ

```
module cnt10(
    clk, rst, ci, co, q);

    input         clk, rst;
    input         ci;
    output        co;
    output [3:0] q;
    reg    [3:0] q;

    always@(posedge clk) begin
        if(rst == 1'b0) begin
            q <= 4’b0000;
        end else if(co == 1’b1) begin
            q <= 4’b0000;
        end else begin
            q <= q + ci;
        end
    end

    assign co = (q == 4’b1001 && ci == 1’b1) ? 1’b1 : 1’b0;
endmodule
```

0から9までカウント

0, 1, 2, … , 8, 9, 0, 1, 2, …

1001(9)の次を0000(0)とするための条件

9からカウンタ値が増えるときcoは1

6進カウンタ

```
module cnt6(
    clk, rst, ci, co, q);

    input         clk, rst;
    input         ci;
    output        co;
    output [2:0] q;
    reg    [2:0] q;

    always@(posedge clk) begin
        if(rst == 1'b0) begin
            q <= 3’b000;
        end else if(co == 1’b1) begin
            q <= 3’b000;
        end else begin
            q <= q + ci;
        end
    end

    assign co = (q == 3’b101 && ci == 1’b1) ? 1’b1 : 1’b0;
endmodule
```

0から5までカウント

0, 1, 2, 3, 4, 5, 0, 1, 2, …

101(5)の次を000(0)とするための条件

5からカウンタ値が増えるときcoは1

60進カウンタ

```
module cnt60(
    clk, rst, ci, co, q);

    input         clk, rst;
    input         ci;
    output        co;
    output [6:0] q;
    wire          w0;

    cnt10 c0(.clk(clk), .rst(rst), .ci(ci), .co(w0), .q(q[3:0]));
    cnt6  c1(.clk(clk), .rst(rst), .ci(w0), .co(co), .q(q[6:4]));
endmodule
```

1の位:10進，4bit
10の位: 6進，3bit

1の位のcoを10の位のciへ

0から59までカウント

0, 1, 2, … , 58, 59, 0, 1, 2, …

この回路を時計の「分」の60進カウンタとして考えると・・・
ci は「秒」の60進カウンタからの入力
co は「時間」の24進カウンタへの出力

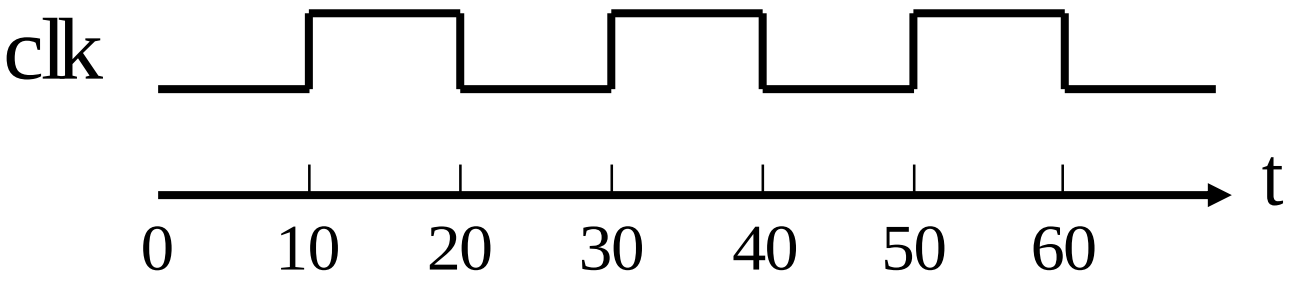
クロック信号の記述方法

テストベンチに以下を記述する

```
initial begin
   clk = 1'b0;
   forever #10 clk = ~clk;
end
```

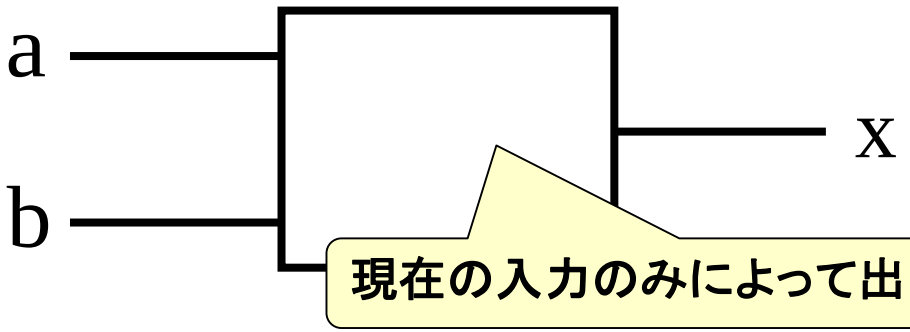
clk の初期値は0

繰り返し実行：時刻10経過したらclkを反転



組合せ回路と順序回路

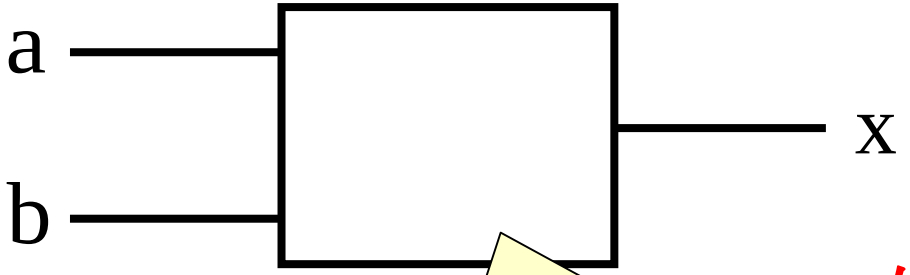
組合せ回路



入力が変化すると出力は必ず変化

```
always@( a or b ) begin
   x <= …;
end
```

順序回路



現在の入力と(記憶していた)過去の出力によって出力が決まる

aが変化したときのみ出力は変化
(それ以外のときは出力は保持)

```
always@( a ) begin
   x <= …;
end
```

aが立ち上がったときのみ出力は変化
(それ以外のときは出力は保持)

```
always@( posedge a ) begin
   x <= …;
end
```

組合せ回路を記述する際の注意

```
always@(a or b or c) begin
    case( c )
        2'b00: y <= a & b;
        2'b01: y <= a | b;
        2'b10: y <= a ^ b;
    endcase
end
```

```
always@(a or b or c) begin
    case( c )
        2'b00: y <= a & b;
        2'b01: y <= a | b;
        2'b10: y <= a ^ b;
        default: y <= 8'bxxxxxxxx;
    endcase
end
```

c=11の条件が無い
→ yの値を保持する回路が合成

default文の記述で回避可能

if-else文で組合せ回路を記述するときも同様の注意が必要

2種類の代入文

- ノンブロッキング代入
  - 全ての式の右辺を先に評価して左辺に代入
  - 以下の例では a と b の値が入れかわる

```
always@(posedge clk) begin
   a <= b;
   b <= a;
end
```

式の記述の順序が
変わっても結果は同じ

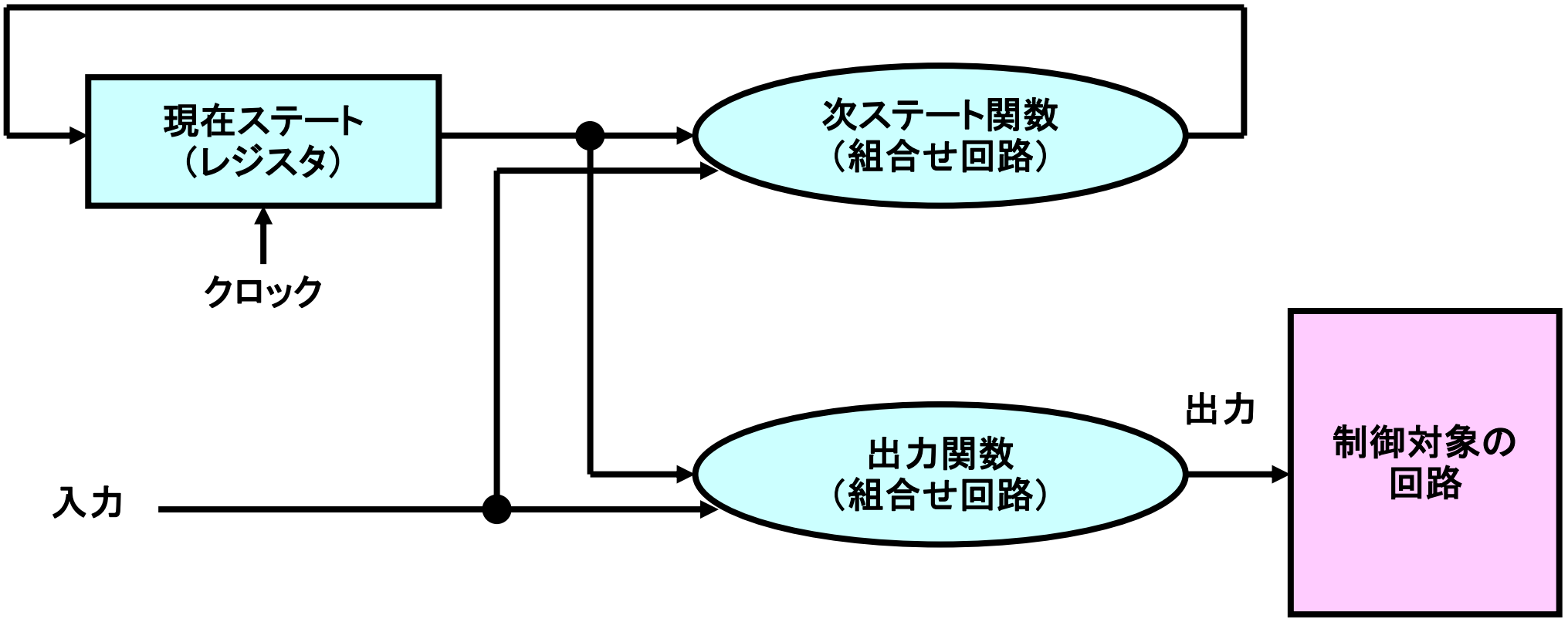
- ブロッキング代入
  - 上の式を評価してから下の式を評価
  - 以下の例では a と b の値は入れかわらない

```
always@(posedge clk) begin
   a = b;
   b = a;
end
```

式の記述の順序が
変わると結果は異なる

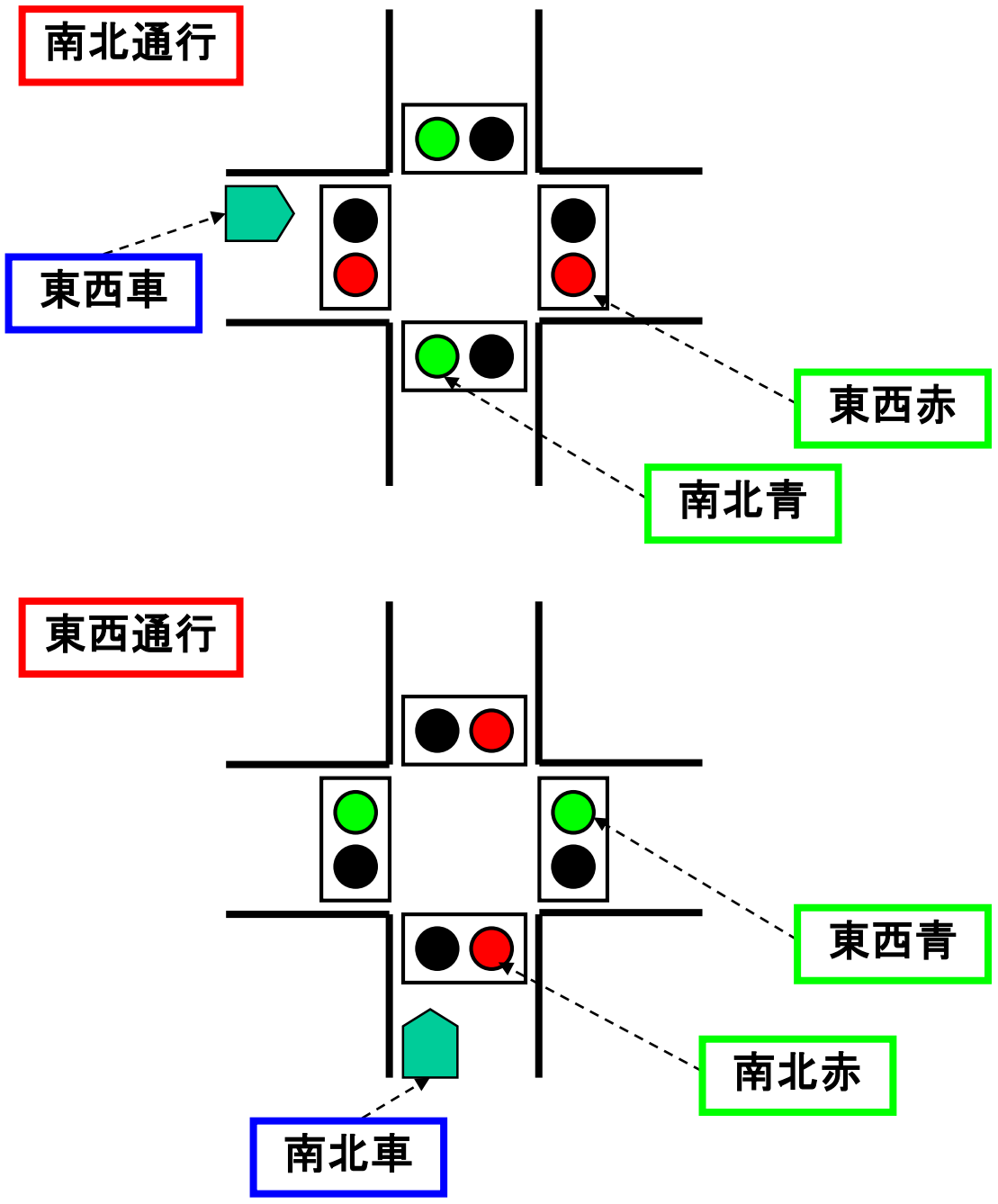
ステートマシン

- 制御回路などを実現する際に使用



- 入力＋現在ステート　→　次ステート決定
- 入力＋現在ステート　→　出力決定

例）信号機



入力:

南北車

東西車

ステート:

南北通行

東西通行

出力:

南北青

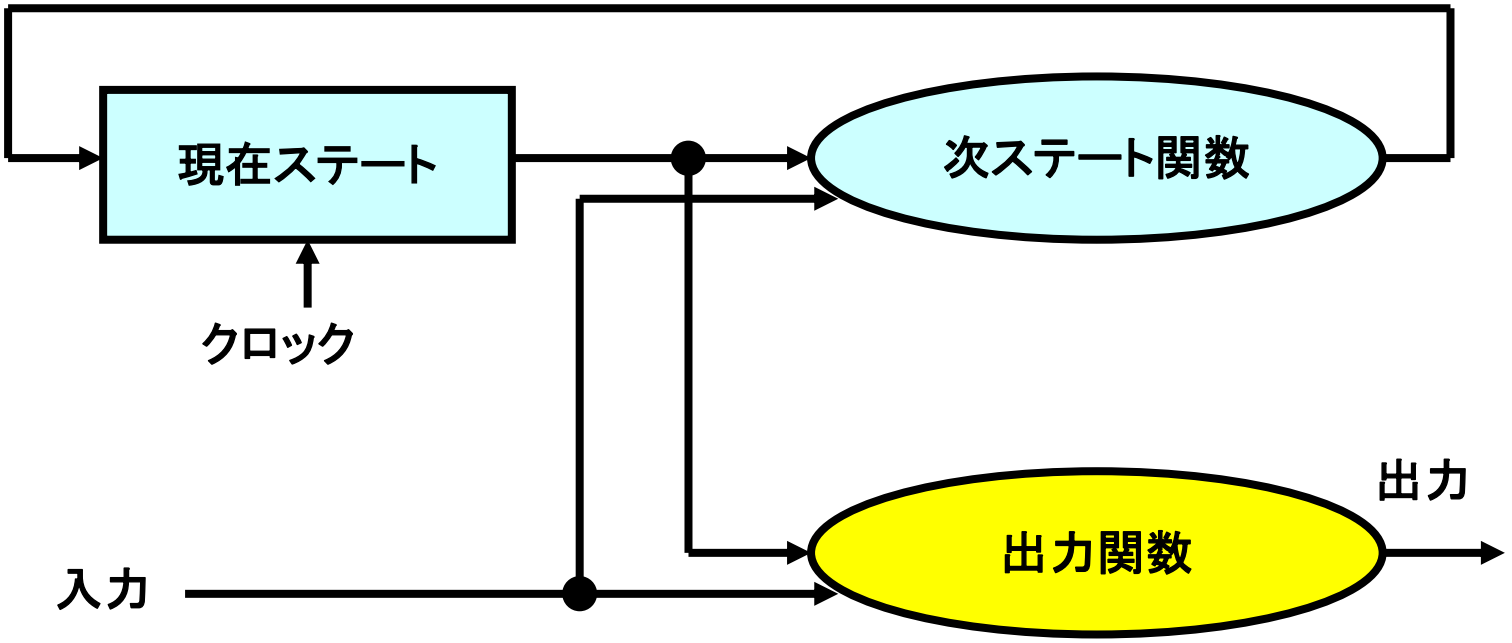
南北赤

東西青

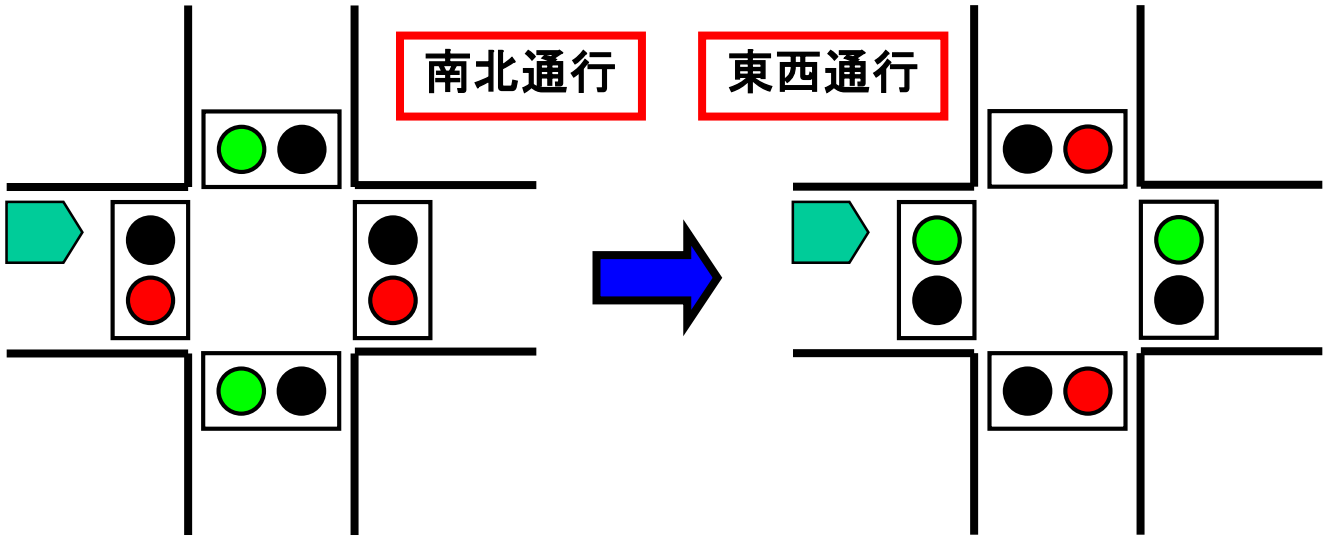
東西赤

出力関数

現在ステート	出力			
	南北青	南北赤	東西青	東西赤
南北通行	1	0	0	1
東西通行	0	1	1	0



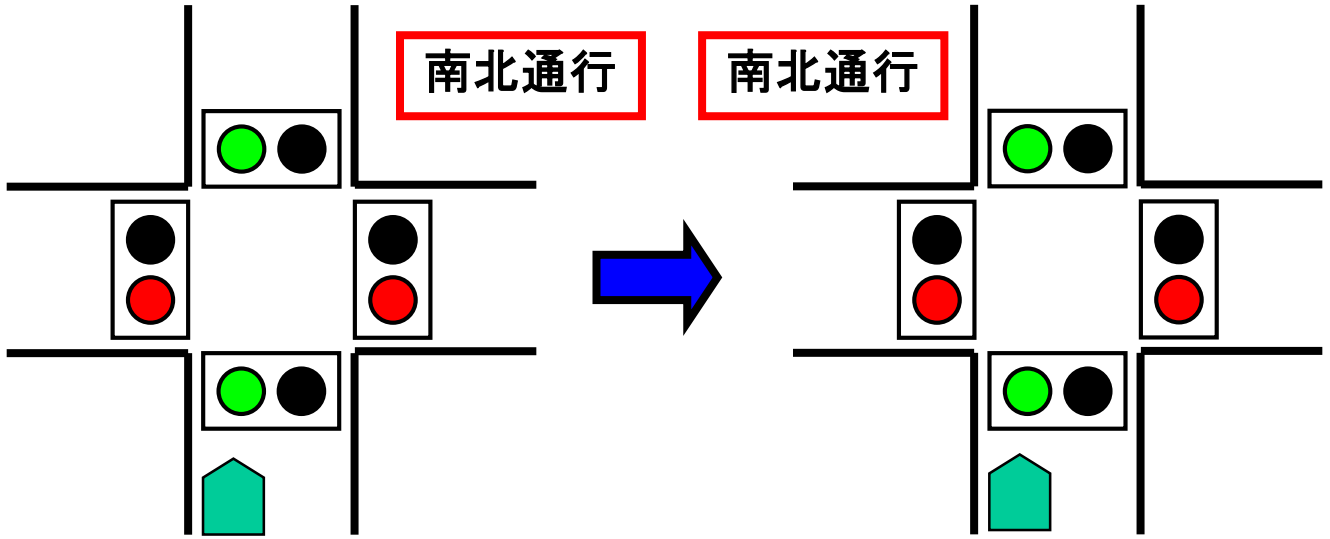
東西のみに車有り　⇒　東西を通行できるようにする



入力：東西車=1
南北車=0

現在ステート：
南北通行

次ステート：
東西通行



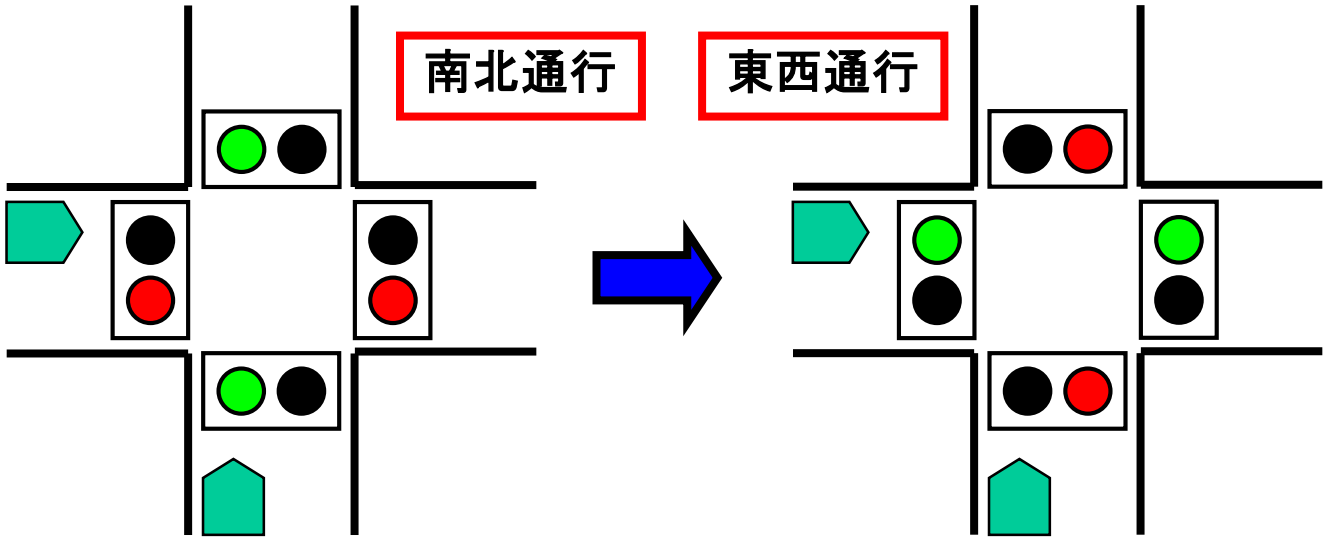
入力：東西車=0
南北車=1

現在ステート：
南北通行

次ステート：
南北通行

南北のみに車有り　⇒　南北を通行できるようにする

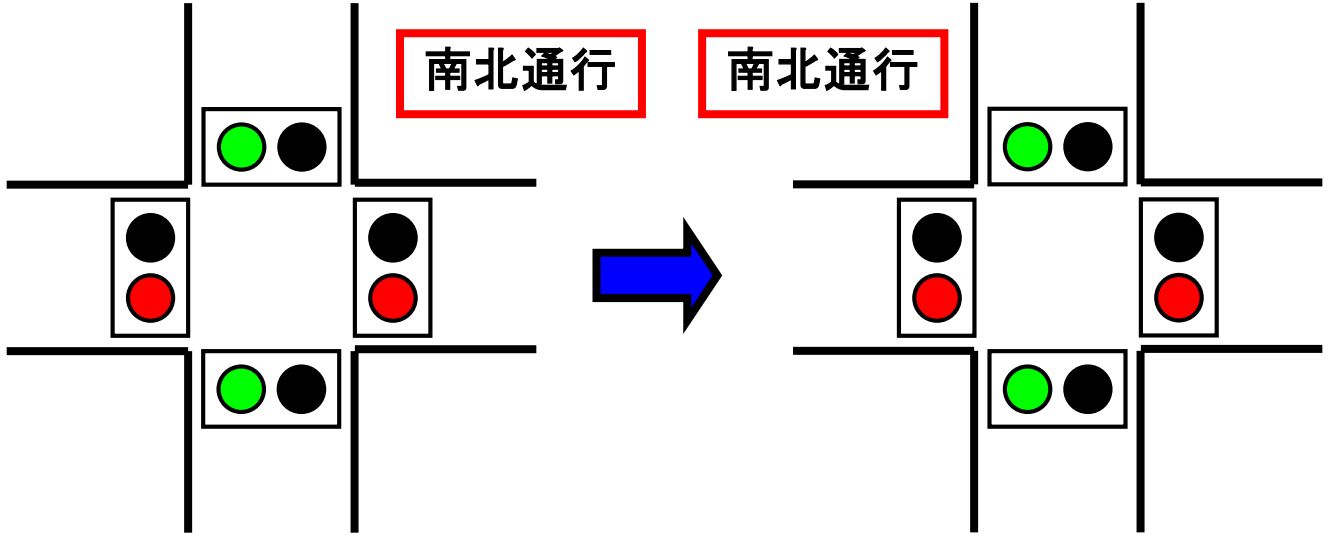
両方に車有り ⇒ 交互に通行できるようにする



入力：東西車=1
南北車=1

現在ステート：
南北通行

次ステート：
東西通行



入力：東西車=0
南北車=0

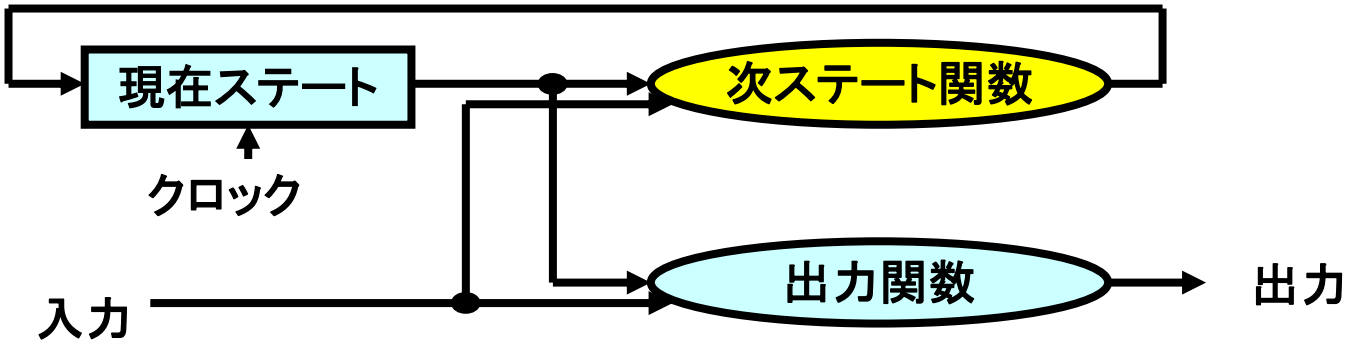
現在ステート：
南北通行

次ステート：
南北通行

どちらからも車無し ⇒ そのまま

次ステート関数

現在ステート	入力		次ステート
	南北車	東西車	
南北通行	0	0	南北通行
南北通行	0	1	東西通行
南北通行	1	0	南北通行
南北通行	1	1	東西通行
東西通行	0	0	東西通行
東西通行	0	1	東西通行
東西通行	1	0	南北通行
東西通行	1	1	南北通行



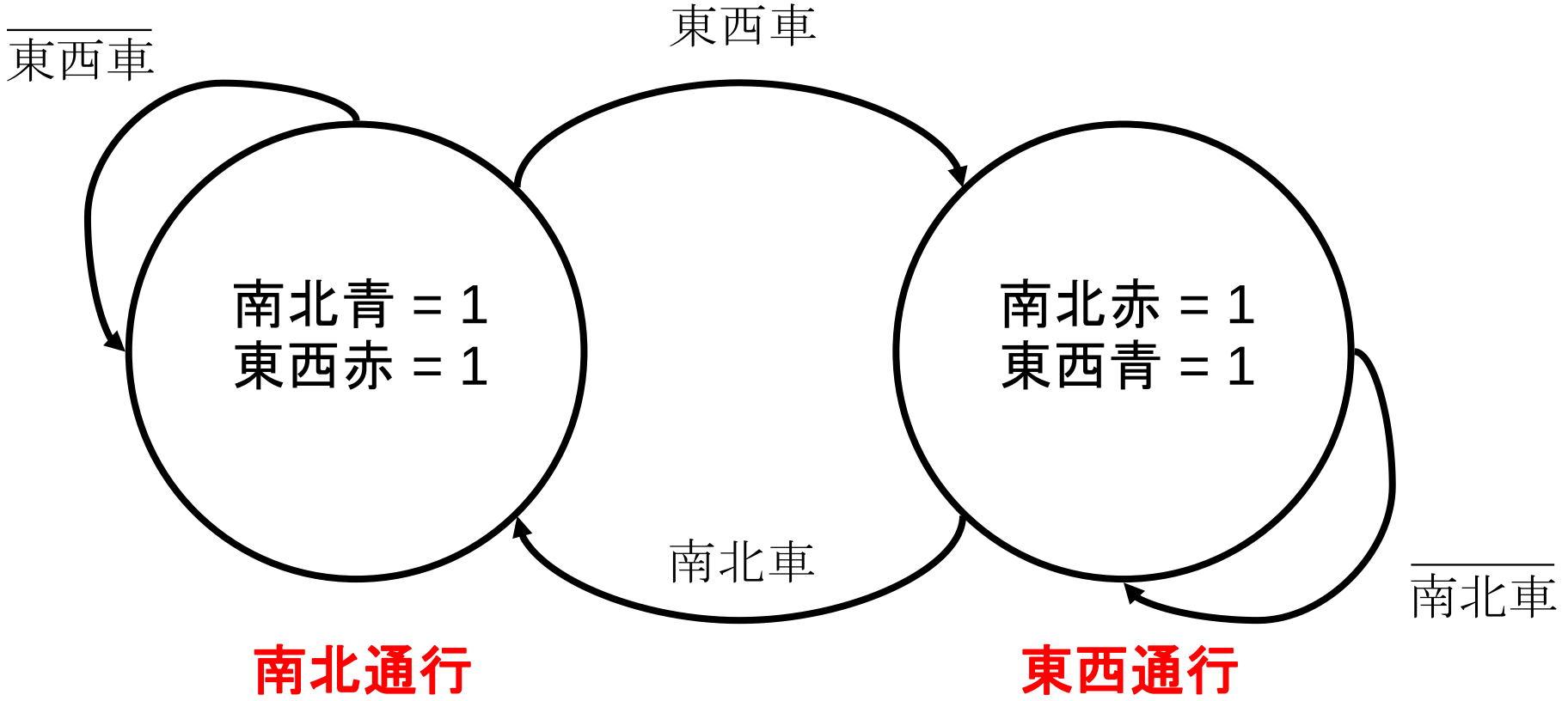
状態遷移図

状態（ステート） ・・・ 円で表現

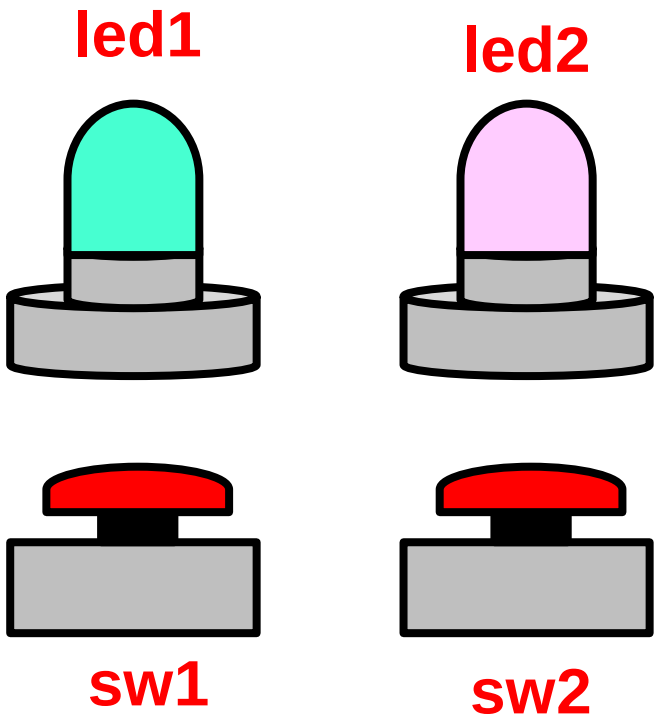
状態遷移 ・・・ 矢印で表現

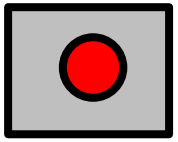
入力 ・・・ 状態遷移の矢印に記述

出力 ・・・ 状態の円の中に記述



課題(1)早押しスイッチの設計





sw1を押す ➔ led1点灯
sw2を押す ➔ led2点灯
早く押した方のみ点灯

その後に何を押しても変化しない
(点灯したまま)

sw1とsw2の同時押し ➔ 変化しない
(消灯したまま)

sw3押す ➔ 初期状態に戻る(消灯する)

入力:	sw1	出力:	led1
	sw2		led2
	sw3		

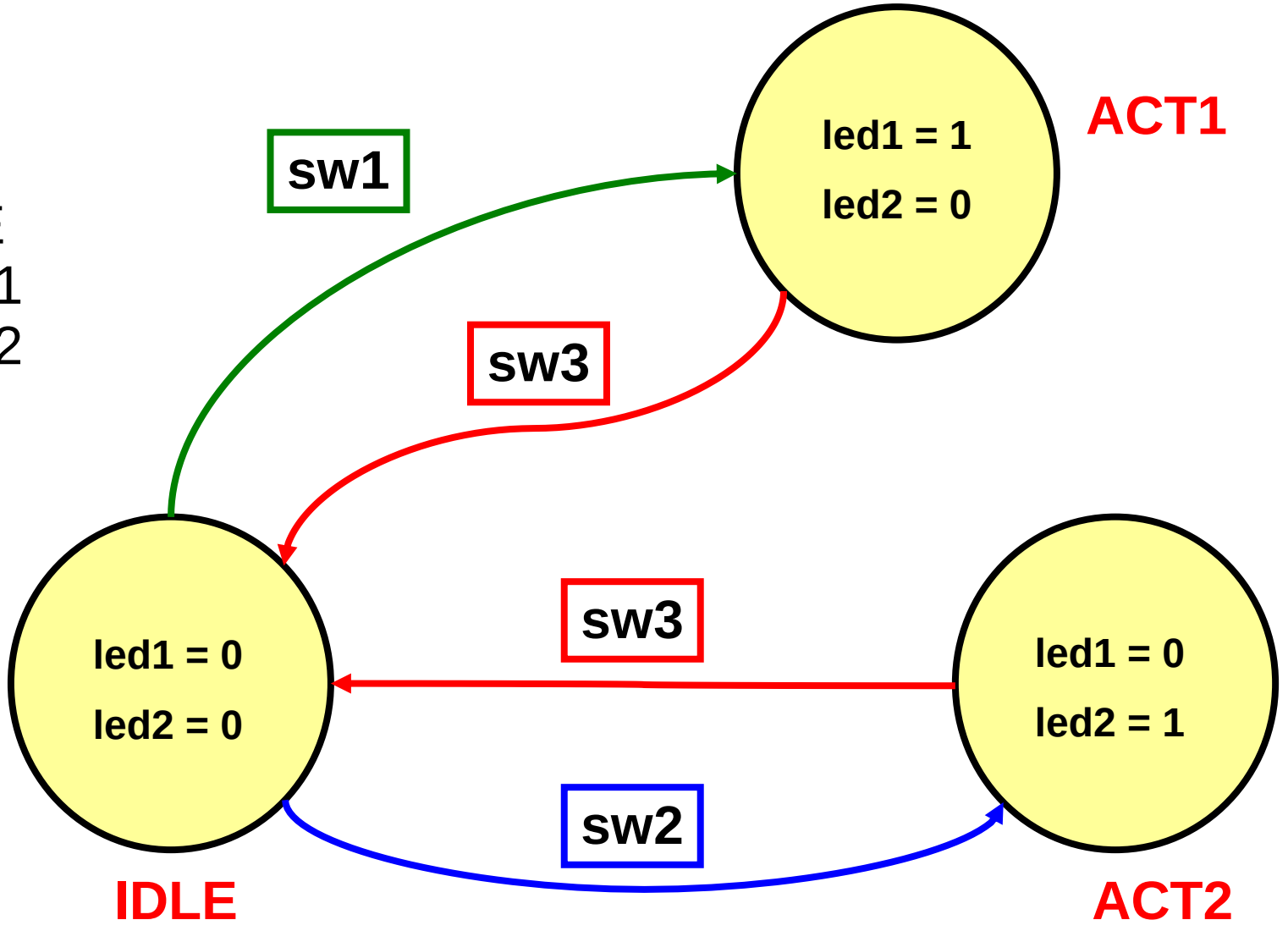
ステート: IDLE (消灯、初期状態)
ACT1 (led1点灯)
ACT2 (led2点灯)

早押しスイッチの状態遷移図

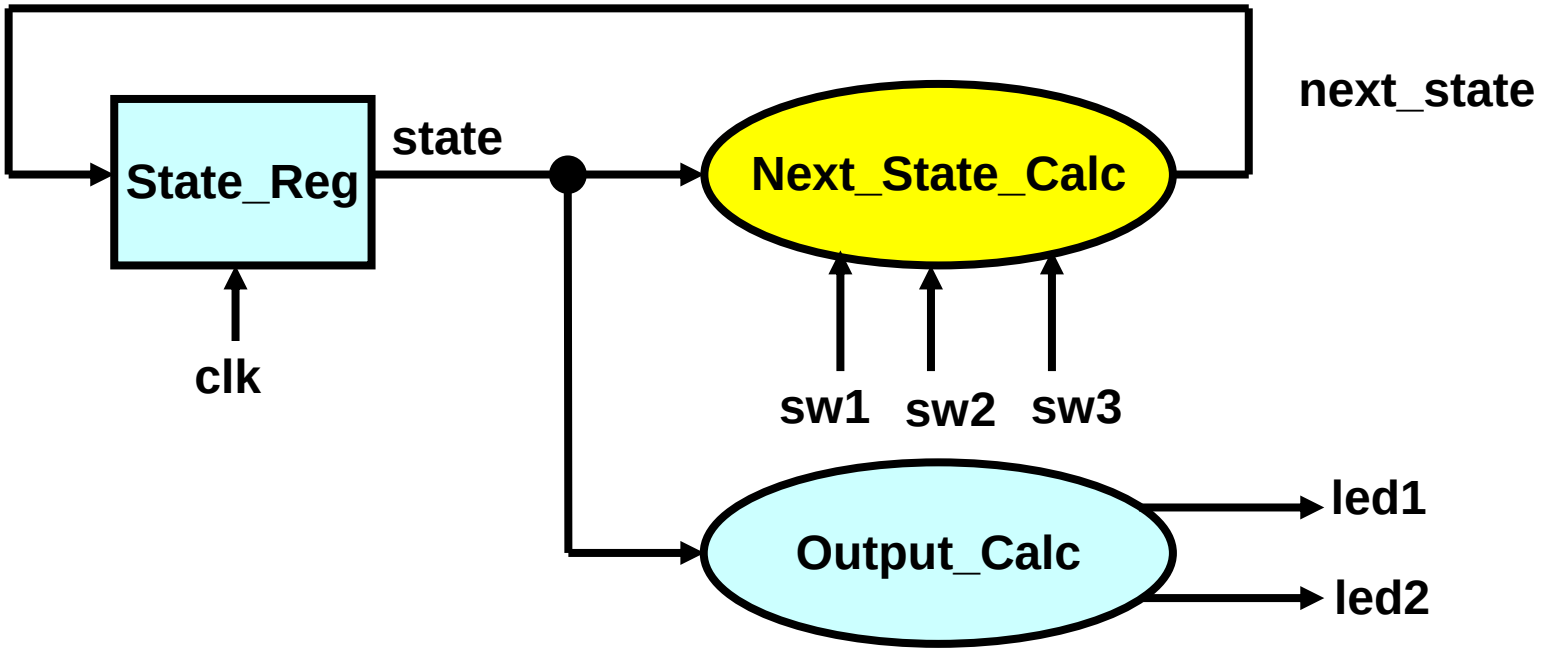
入力: sw1
sw2
sw3

ステート: IDLE
ACT1
ACT2

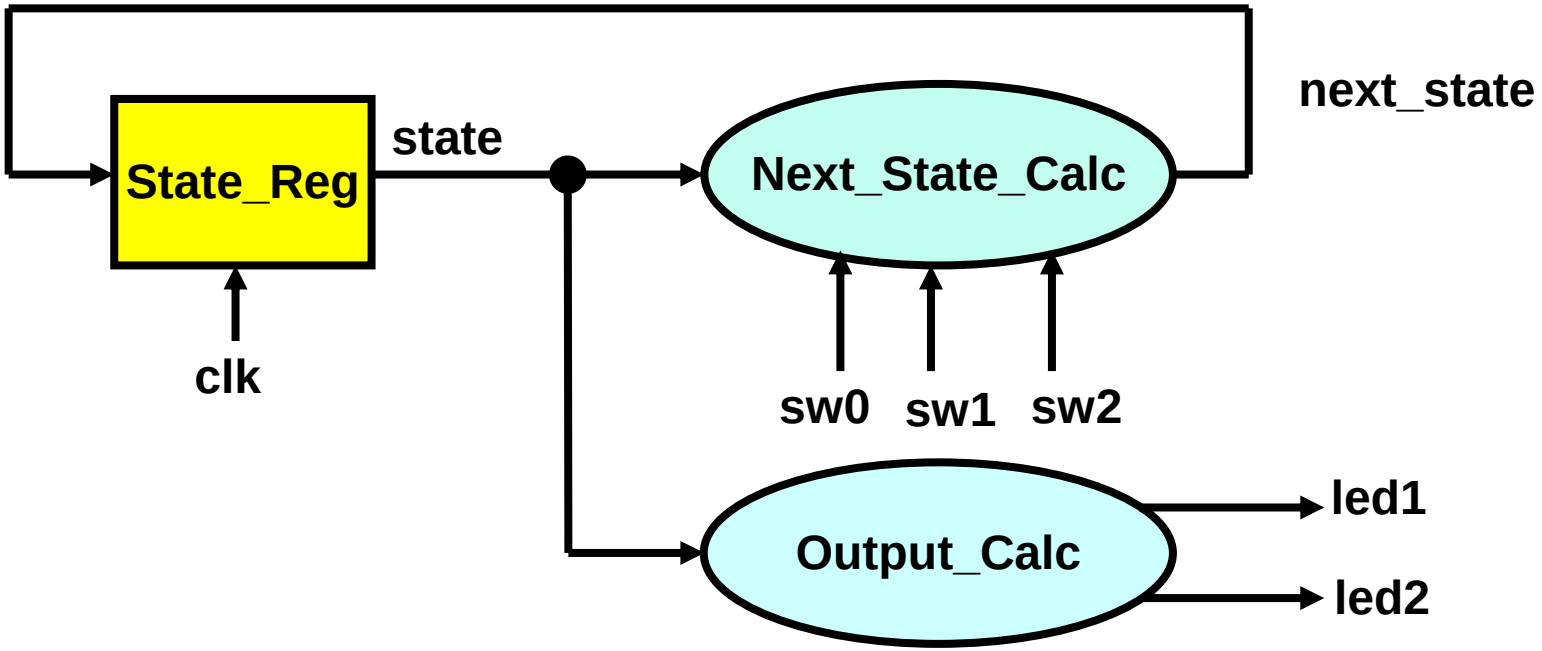
出力: led1
led2



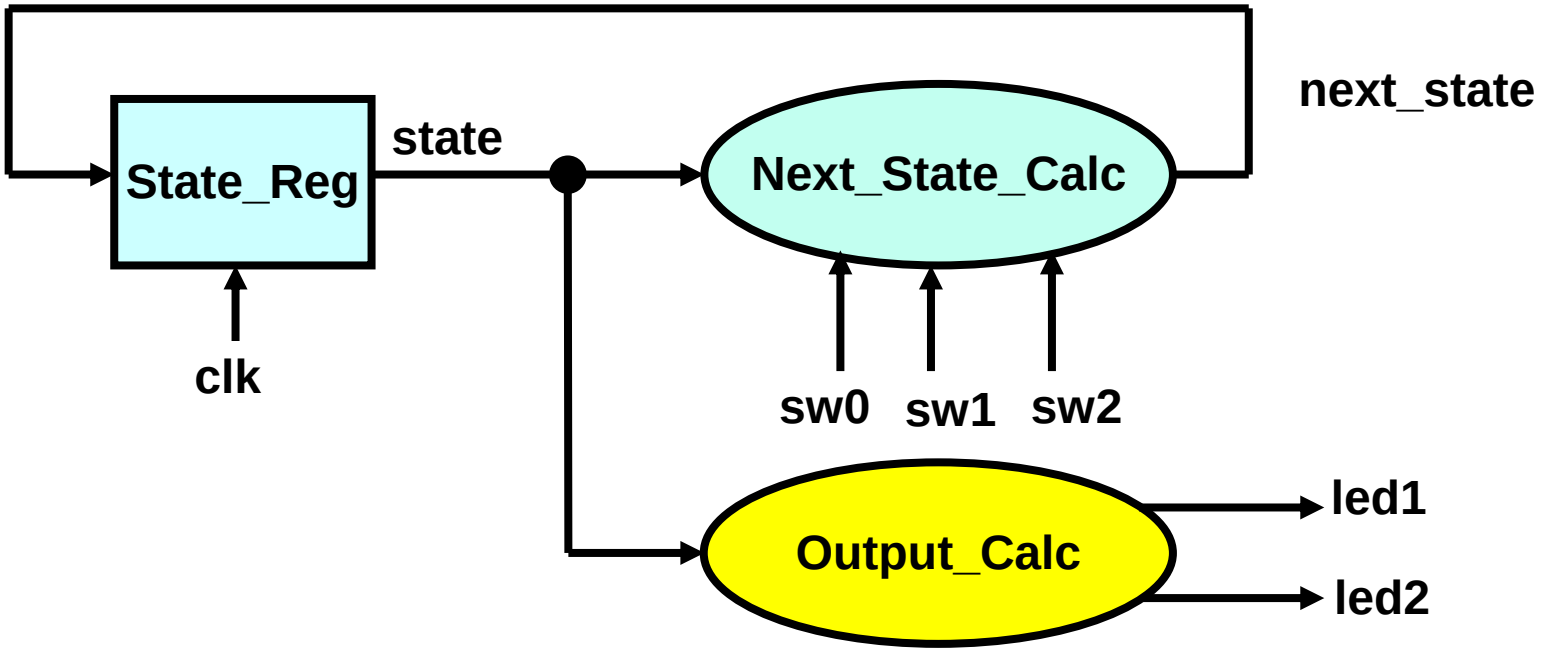
図に書いていない入力のパターンのとき状態は変化しない




```
always@ ( sw1 or sw2 or sw3 or state ) begin
    case ( state )
        `IDLE : begin
           if(sw1 == 1'b1 && sw2 == 1'b1)
              next_state <= `IDLE;
           ...
        end
        ...
    endcase
end
```

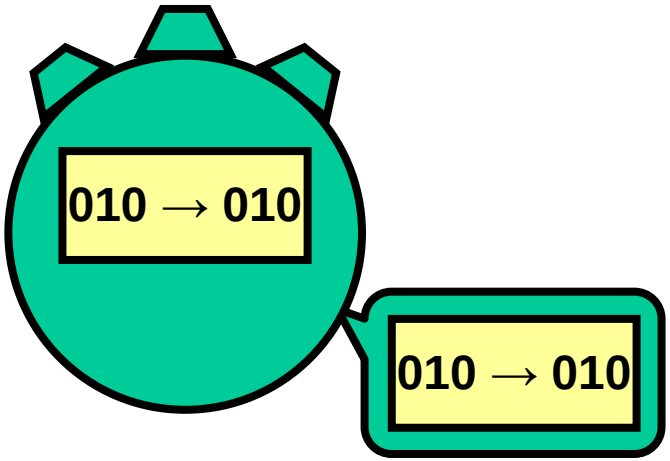



```
always@ ( posedge clk ) begin
    state <= next_state;
end
```

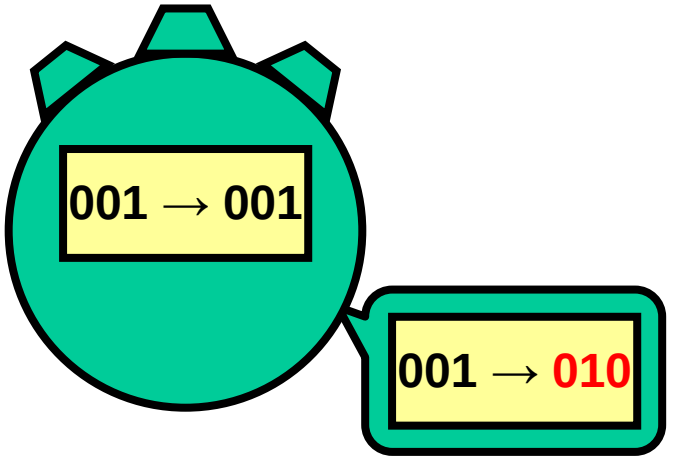



```
always@ ( state ) begin
    case ( state )
        `IDLE : begin
                led1 <= 1'b0;
                led2 <= 1'b0;
            end
            ...
        end
        ...
    endcase

end
```

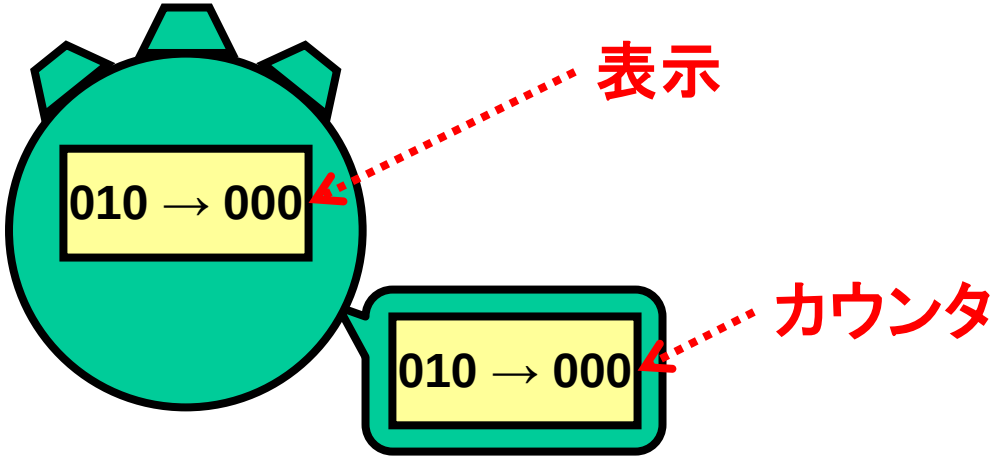
課題（2）ストップウォッチの設計



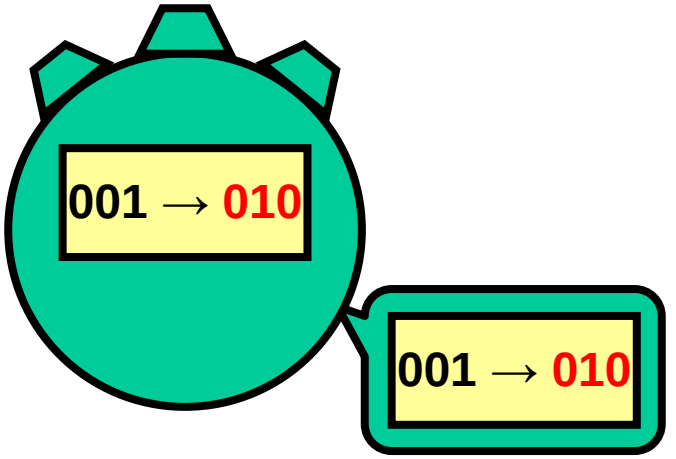
STOP：表示は動作、カウンタは停止



LAP：表示は停止、カウンタは動作

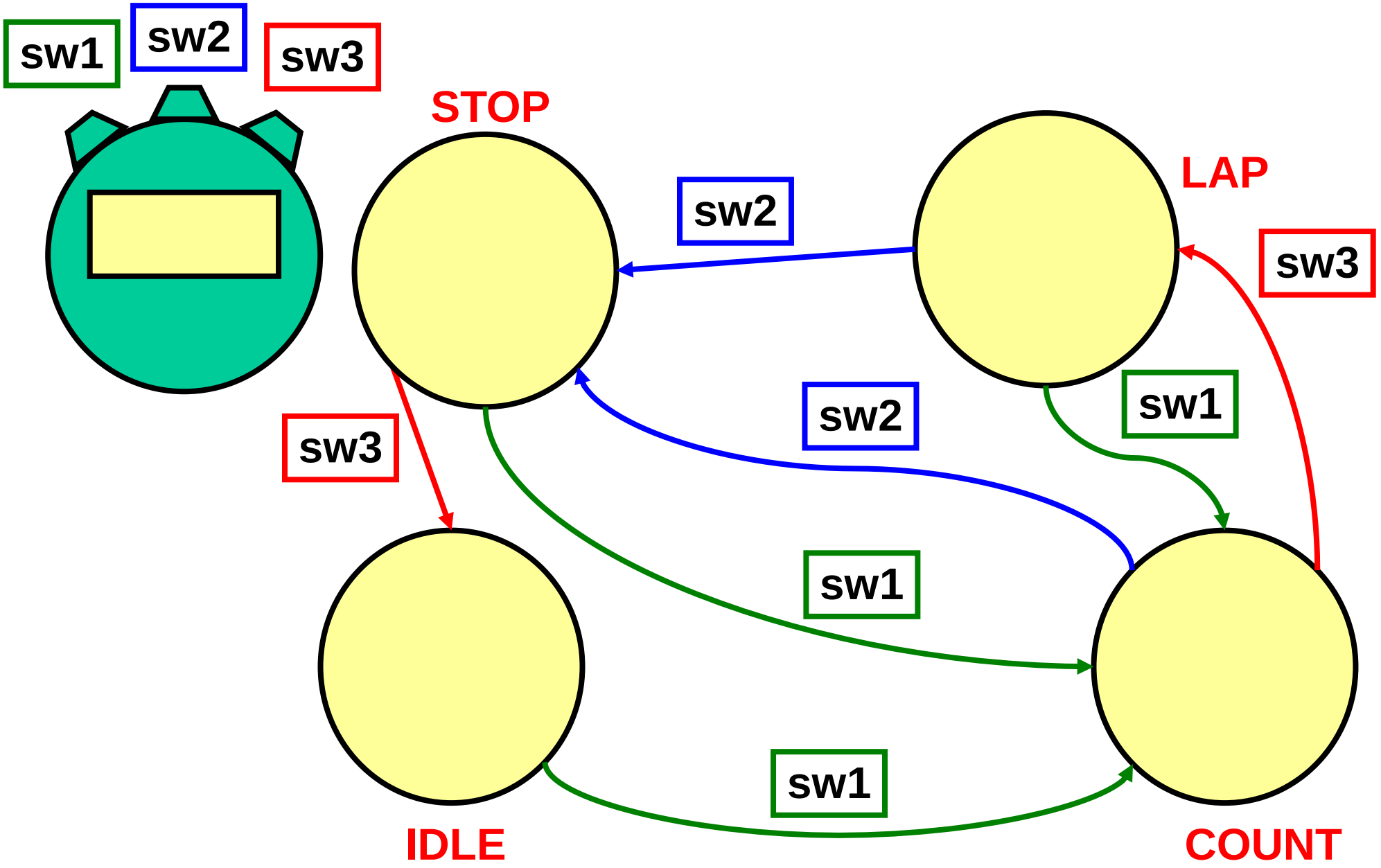


IDLE：値をリセットして
表示もカウンタも停止



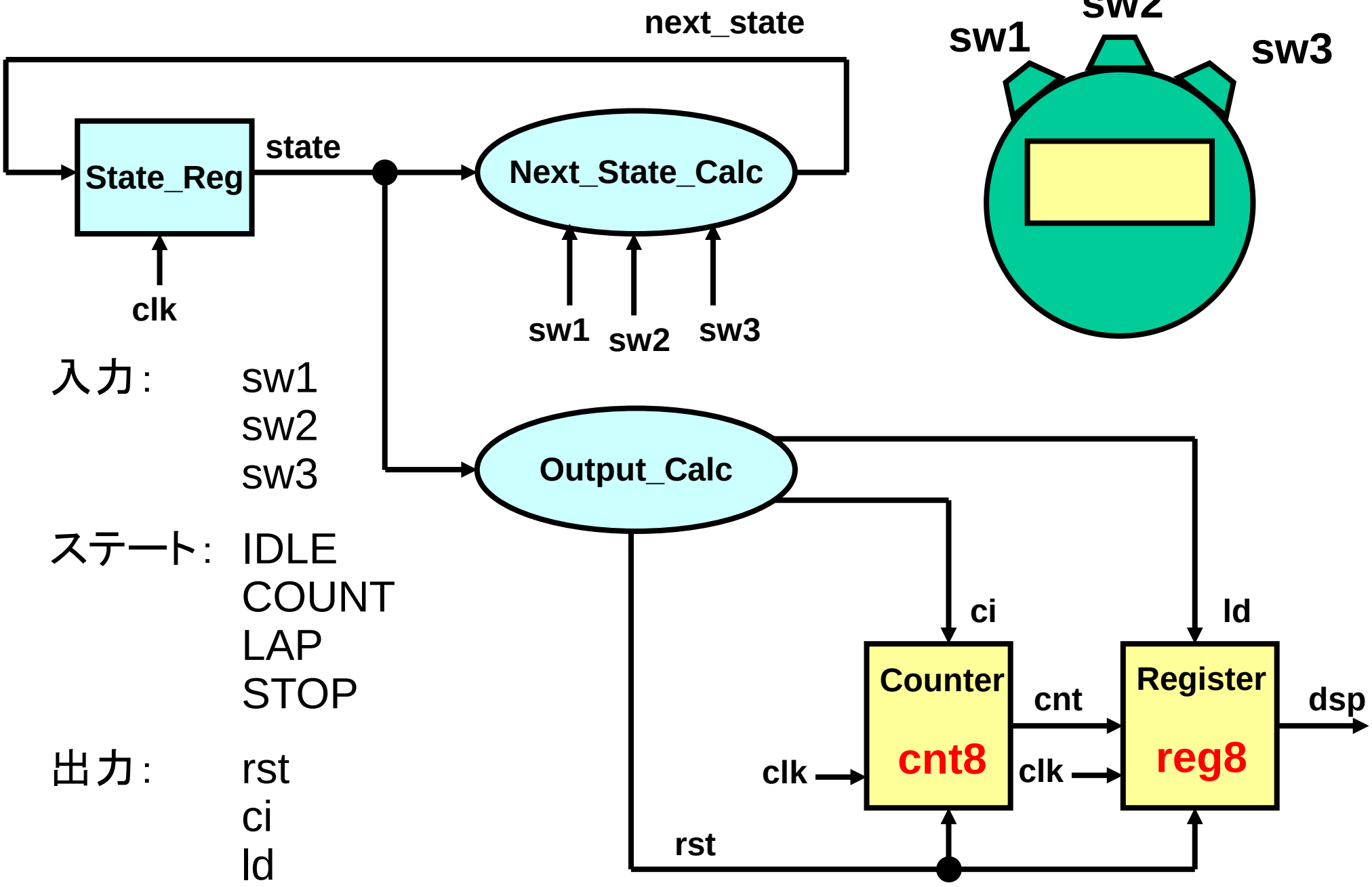
COUNT：表示もカウンタも動作

ストップウォッチの状態遷移図



※ 同時押しの場合はどれを優先にしてもよい

ストップウォッチの回路図

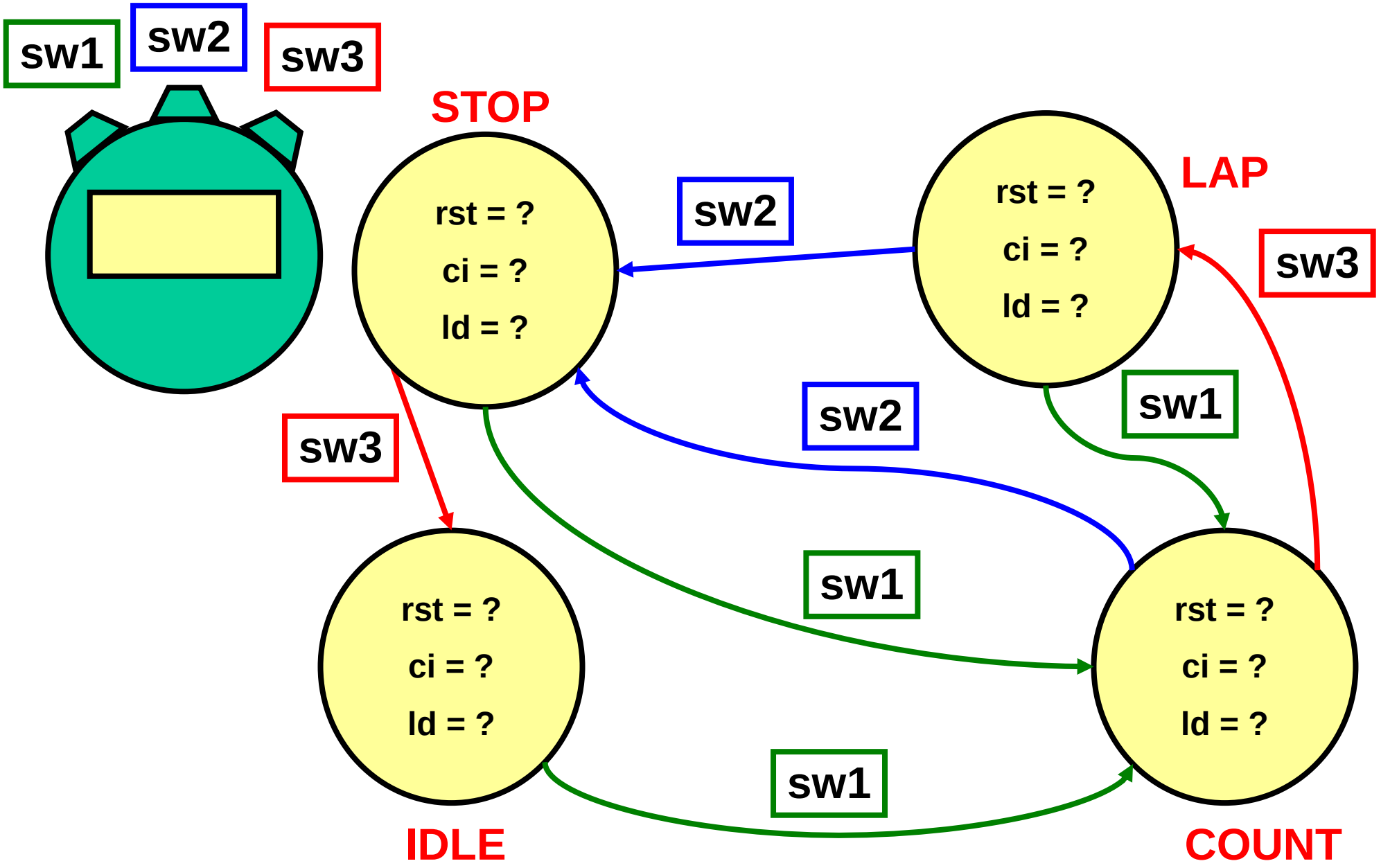


入力：　sw1
　　sw2
　　sw3

ステート：　IDLE
　　COUNT
　　LAP
　　STOP

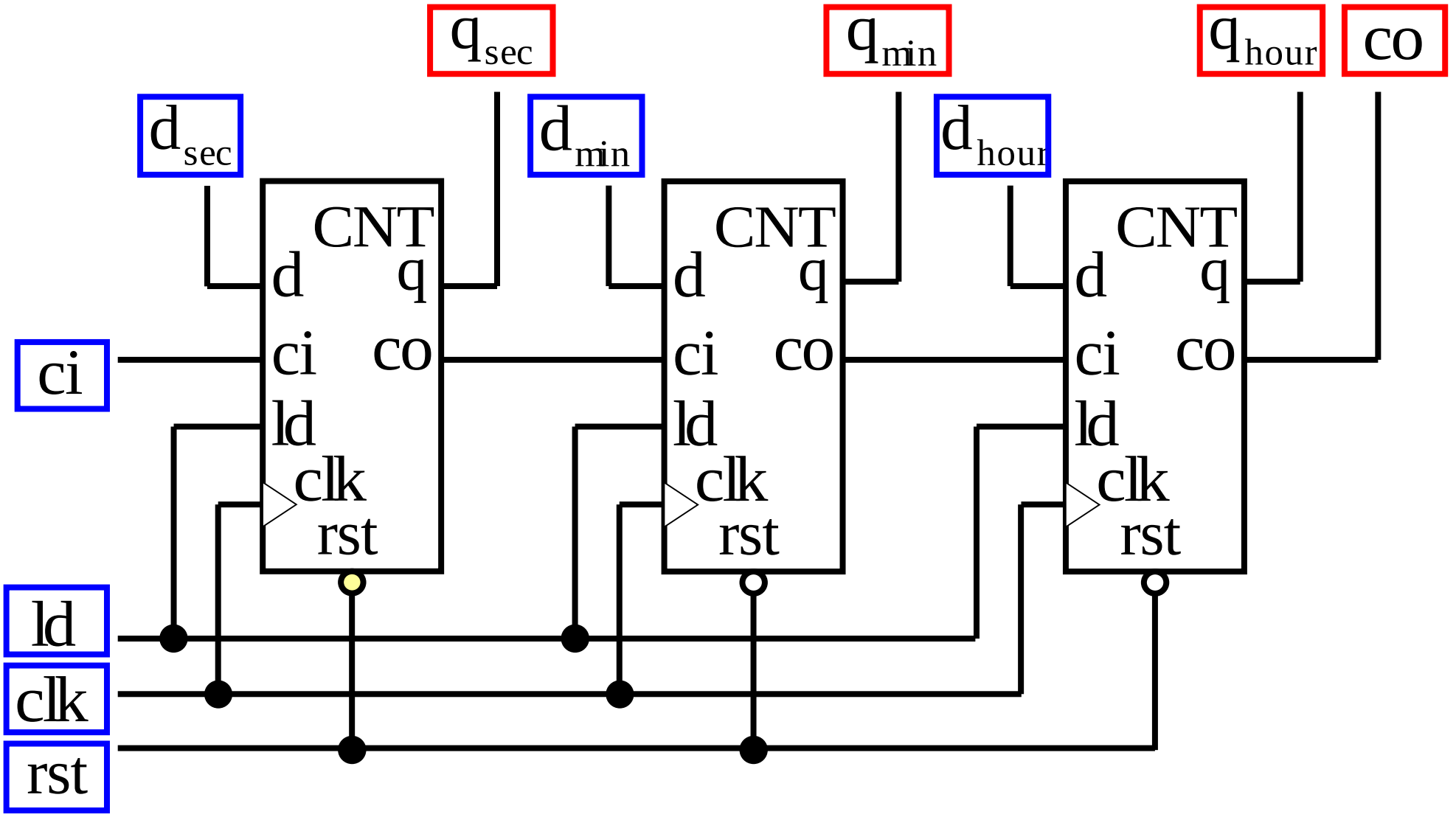
出力：　rst
　　ci
　　ld

出力関数(各自で考えよ)



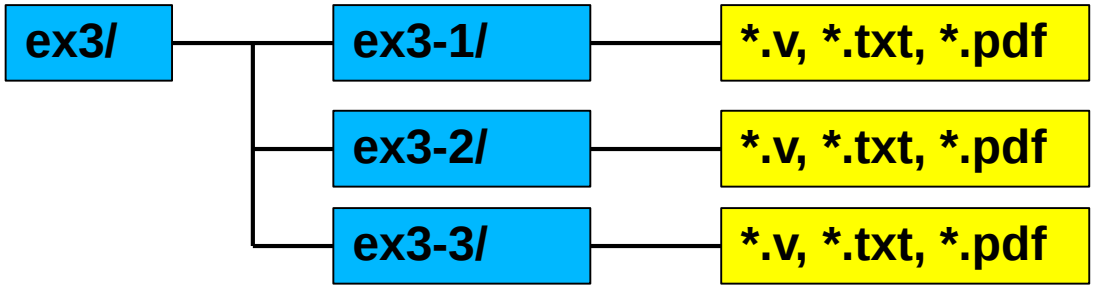
※ 同時押しの場合はどれを優先にしてもよい

課題（3）時計


q_{sec}
q_{min}
q_{hour}
co
d_{sec}
d_{min}
d_{hour}
CNT
d
q
ci
co
ld
clk
rst
ci
ld
clk
rst
60進カウンタ
60進カウンタ
24進カウンタ

提出すべきファイル

- 課題の回路のHDLソースファイル（テストベンチを含む）
  各設計方法（1）～（3）に対してディレクトリ（ex3-?）を作成すること
- 課題の回路のシミュレーション実行結果
  （ iverilog の実行結果をリダイレクトして txt ファイルを作成）
- 課題の回路のシミュレーション波形
  （gtkwaveから pdf ファイルを作成）
- 設計方法ごとに圧縮アーカイブファイル（ex3-?.tar.gz）を作成して
  Web上から提出せよ




```
cd ex3
tar zcvf ex3-1.tar.gz ex3-1/
tar zcvf ex3-2.tar.gz ex3-2/
tar zcvf ex3-3.tar.gz ex3-3/
```