論理回路入門（11）

ラッチとフリップフロップ

李　亜民

2023 年 12 月 5 日 (火)

ラッチとフリップフロップ

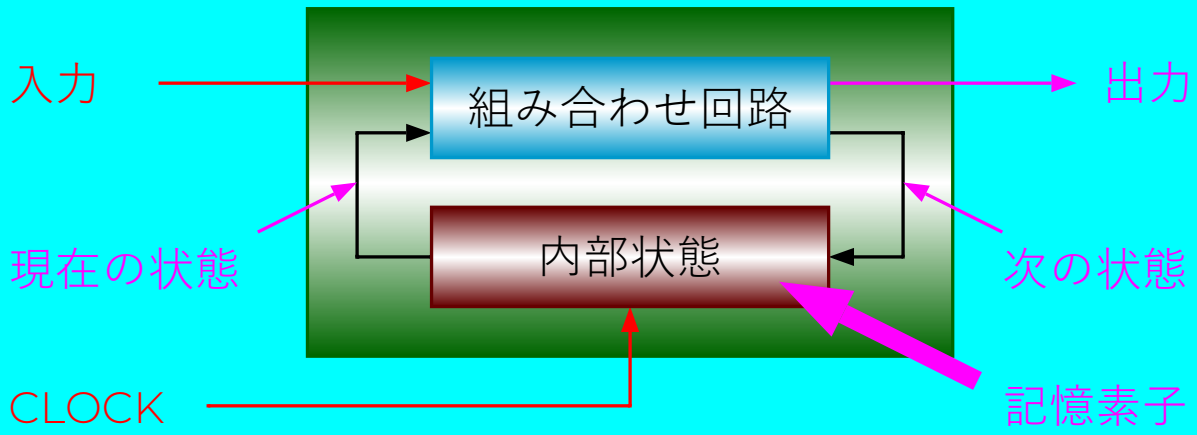
ポイント

- 順序回路
- RS ラッチ (RS Latch)
- D ラッチ (D Latch)
- D フリップフロップ (DFF, DFFE)
- T フリップフロップ (TFF, TFFE)
- JK フリップフロップ (JKFF, JKFFE)
- レジスタ (Register)
- レジスタファイル (Register File)

論理回路の種類

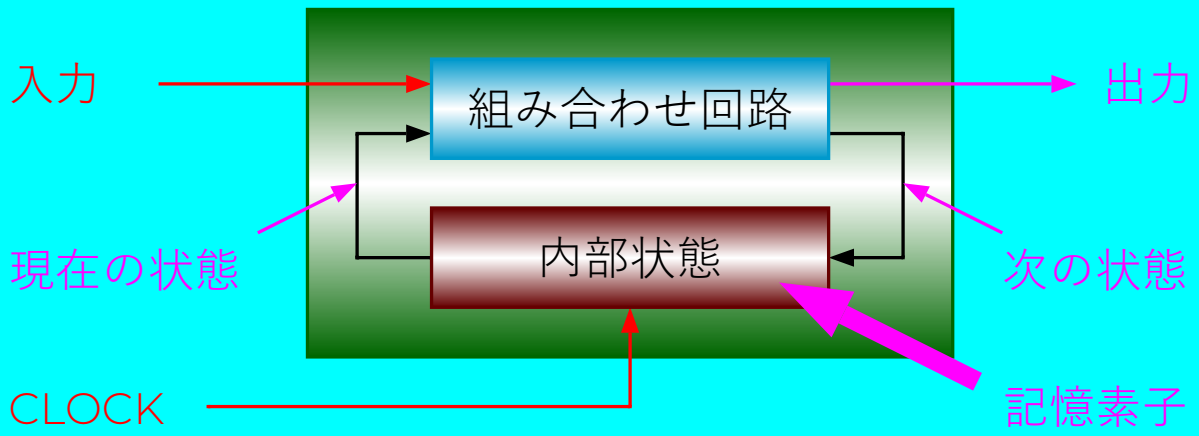
1. 組み合わせ回路 (Combinational Circuit): 現在の入力のみで出力が決まる回路である。

2. 順序回路 (Sequential Circuit): 内部状態と入力信号で出力が決まる回路である。有限状態機械 (Finite state machine — FSM) とも呼ばれる。



順序回路

- 順序回路 (Sequential Circuit): 内部状態と入力信号で出力が決まる回路である。有限状態機械 (Finite state machine — FSM) とも呼ばれる。



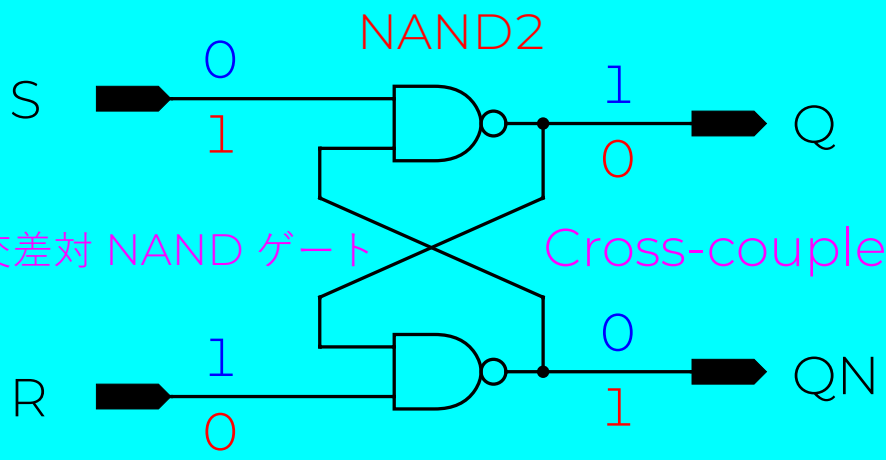
- 順序回路の例：自動販売機の制御回路

  どれだけのお金が入れられたかを記憶しなければならない。

記憶素子の種類

- ラッチ (Latch)
  - RS ラッチ (RS Latch)
  - D ラッチ (D Latch)
- フリップフロップ (Flip Flop)
  - D フリップフロップ (DFF, DFFE)
  - T フリップフロップ (TFF, TFFE)
  - JK フリップフロップ (JKFF, JKFFE)
- メモリ (Memory)
  - SRAM (Static Rondom Access Memory)
  - DRAM (Dynamic Rondom Access Memory)
  - ROM (Read Only Memory)
  - CAM (Content Addressable Memory) — 連想メモリ

RS ラッチ



NAND2 ゲートの真理値表

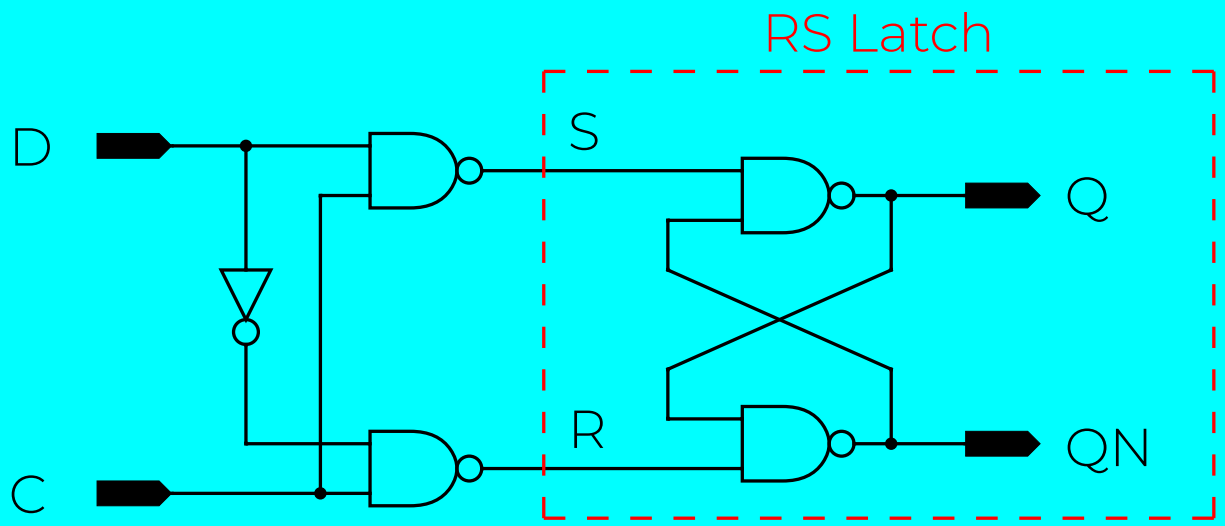
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

S: Set, active low (0 の時、セットする (Q = 1))

R: Reset, active low (0 の時、リセットする (Q = 0))

- S = 0, R = 1: Set (Q = 1) · · · · · · · · · · · · · · · · · セット
- S = 1, R = 0: Reset (Q = 0) · · · · · · · · · · · · · · · リセット
- S = 1, R = 1: No change · · · · · · · · · · · · · · · · · · 保持
- S = 0, R = 0: Not allowed · · · · · · · · · · · · · · · · · 禁止

D ラッチ



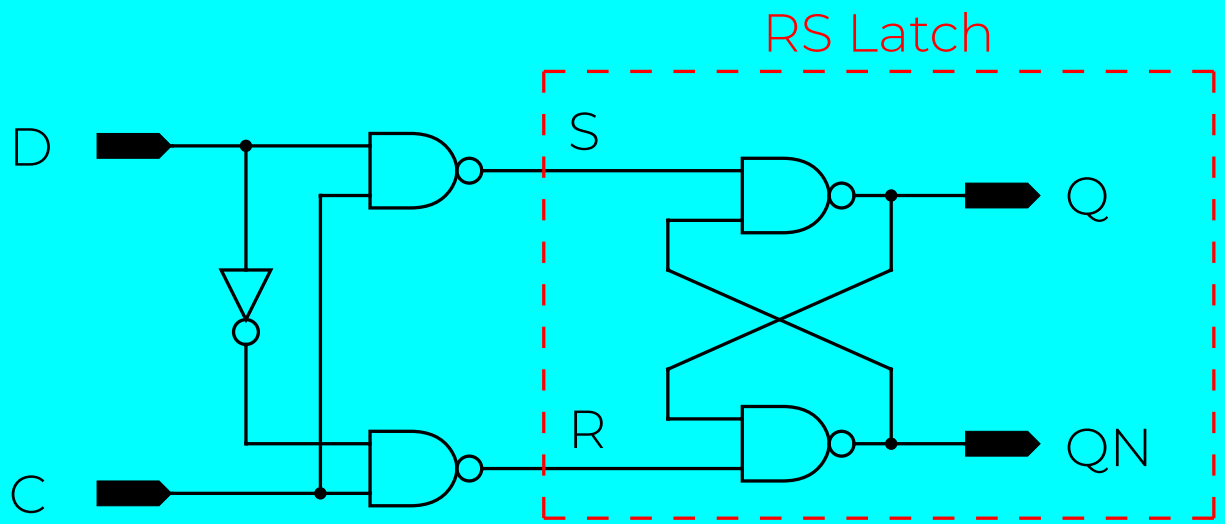
NAND2 ゲートの真理値表

A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

- C = 1, D = 0: S = 1, R = 0, Q = 0 = D
- C = 1, D = 1: S = 0, R = 1, Q = 1 = D
- C = 0, D = x: S = 1, R = 1, No change

  S = 0, R = 0 の場合はない ☺

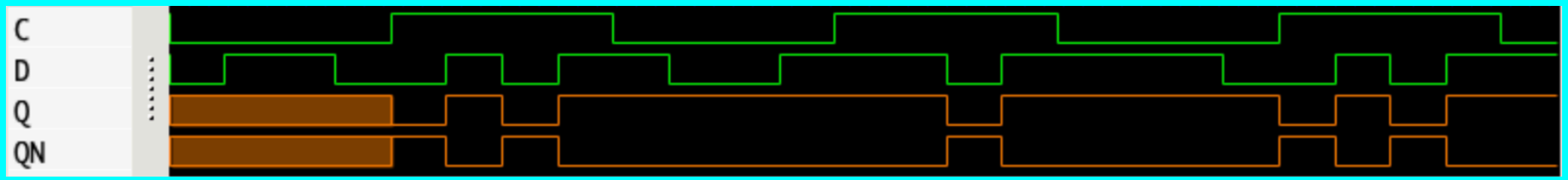
D ラッチシミュレーション



NAND2 ゲートの真理値表

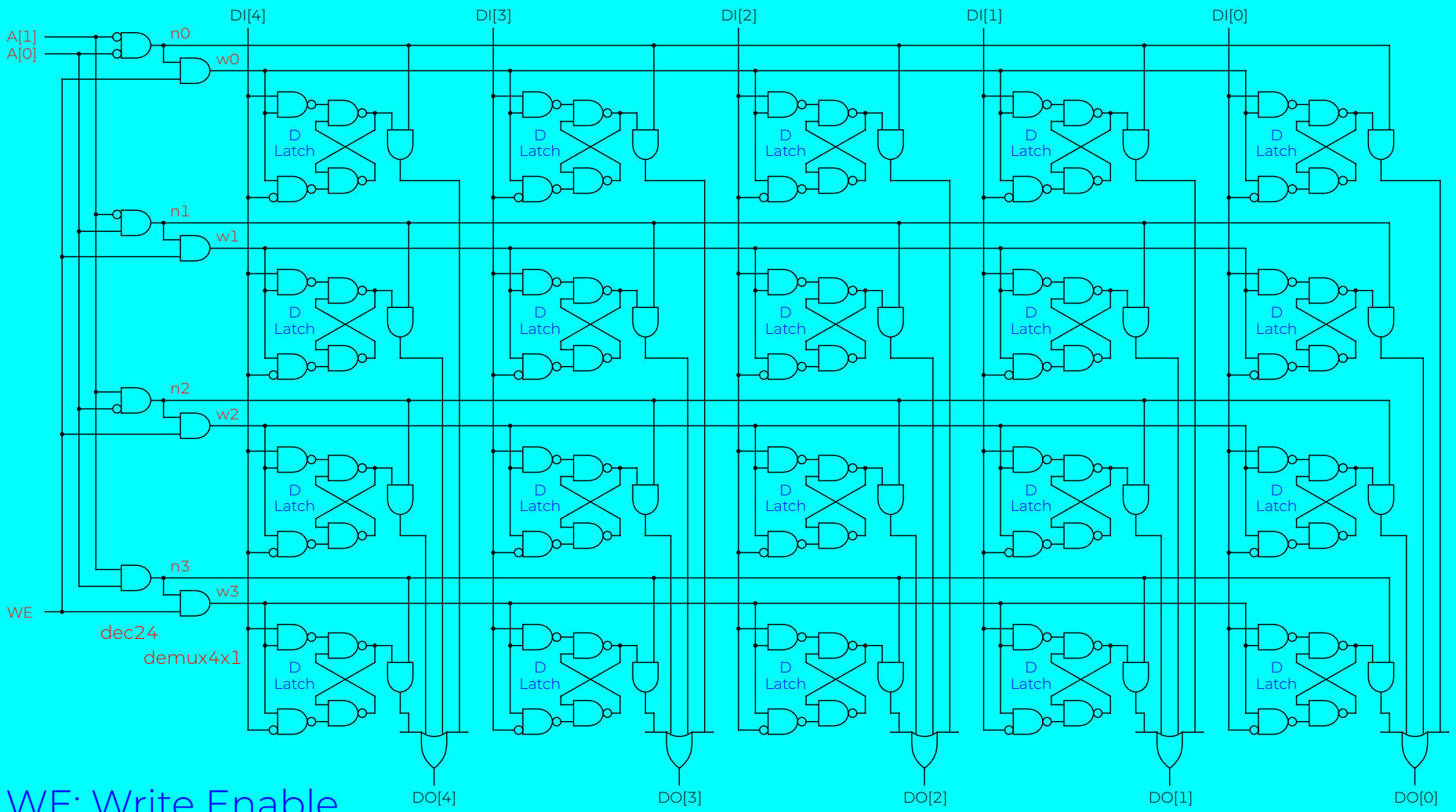
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

d_latch.v
d_latch_tb.v



入力 C = 1 の時、出力 Q = D（データ入力はデータ出力で伝わる）
入力 C = 0 の時、出力 Q は変化なし（以前のデータ出力を保持する）

D ラッチでSRAMの実装

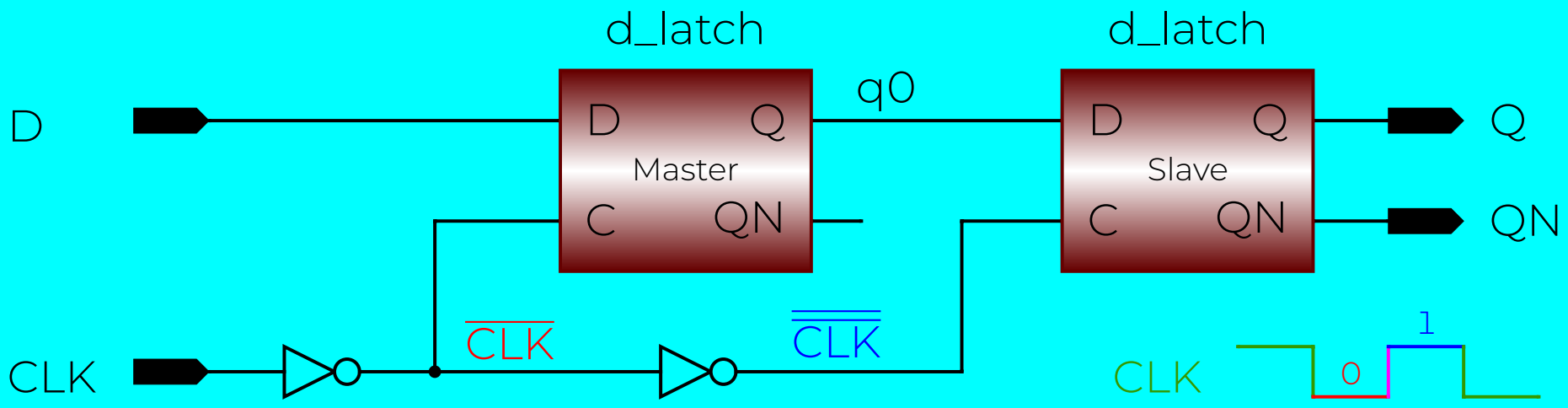


WE: Write Enable

4 words: 2-bit address A[1:0]: $2^2 = 4$

5 bits/word: data-in DI[4:0]; data-out DO[4:0]

D フリップフロップ (DFF)

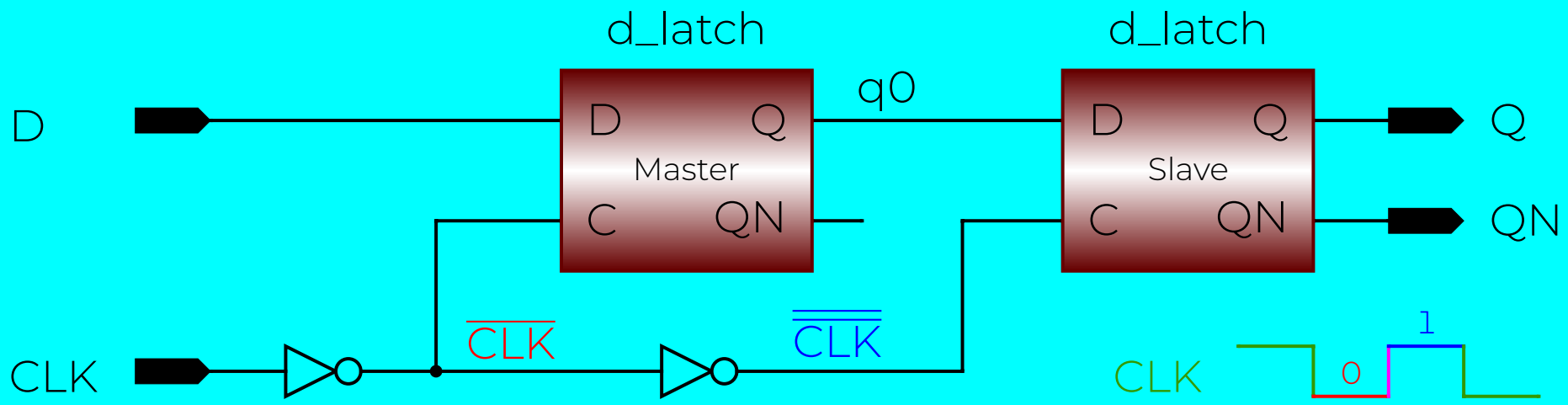


CLK = 0 の時、$\overline{CLK} = 1$、Master q0 = D (Slave 変化なし)

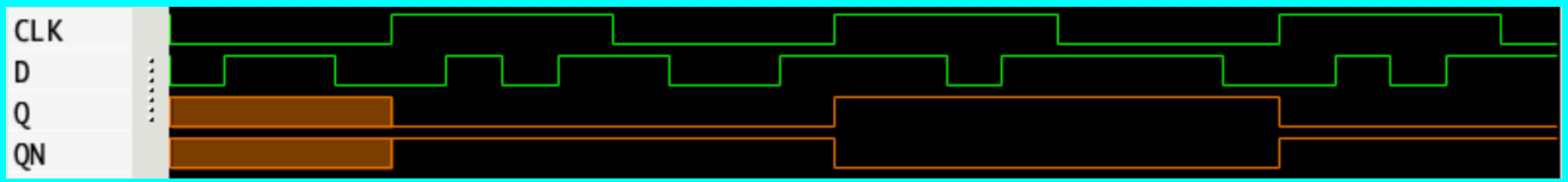
CLK = 1 の時、$\overline{CLK} = 0$、Slave Q = q0 (Master 変化なし)

クロック信号 (CLK) の立ち上がり (0 から 1 への遷移) の直前のデータ入力 (D) の値が記憶され、出力 (Q) に出力される。

DFF回路と波形

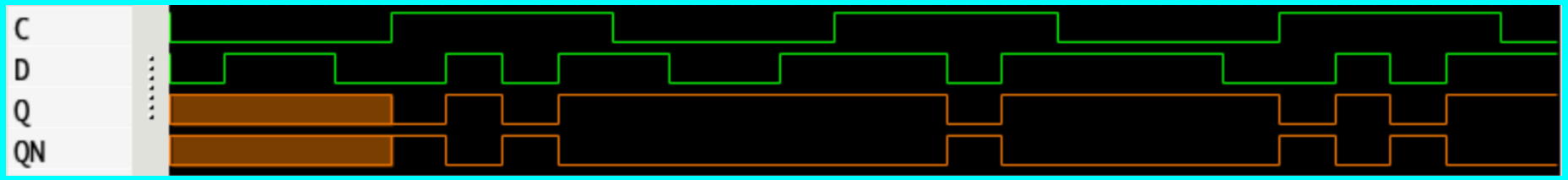


d_flip_flop.v　d_flip_flop_tb.v

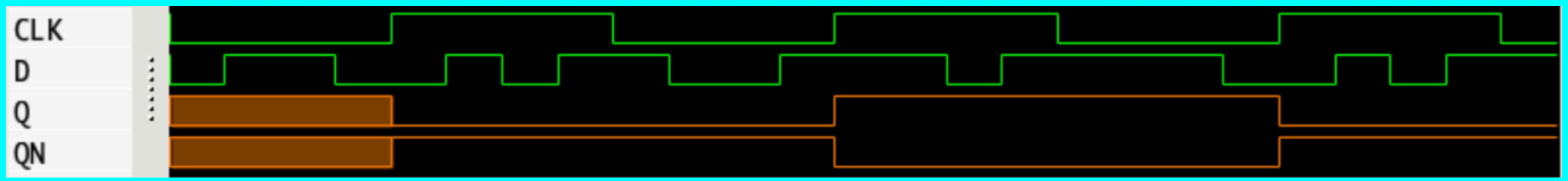


クロック信号 (CLK) の立ち上がり (0 から 1 への遷移) の直前のデータ入力 (D) の値が記憶され、出力 (Q) に出力される。

シミュレーション波形の比較

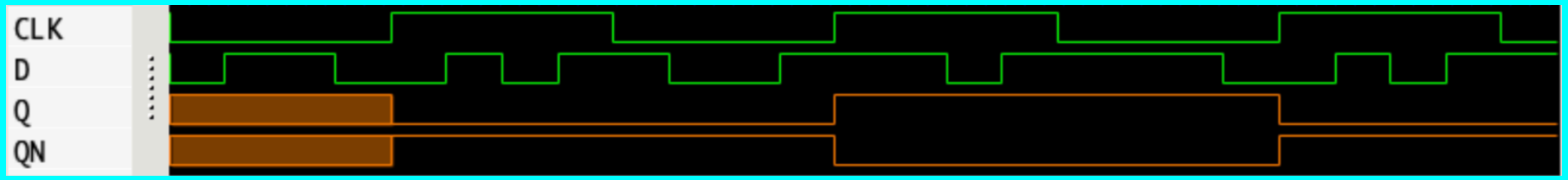


Ｄラッチ

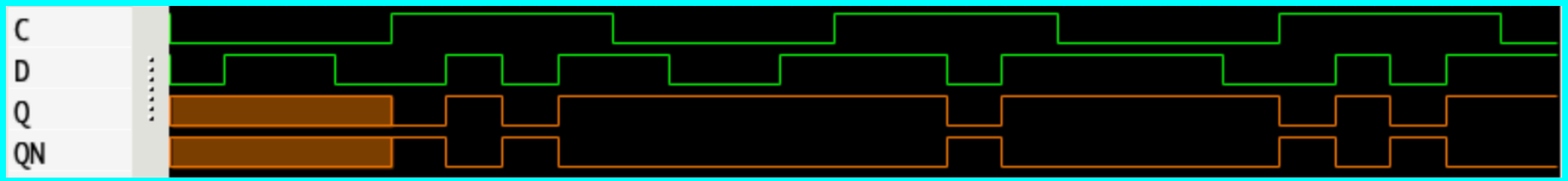


Ｄフリップフロップ

シミュレーション波形の比較

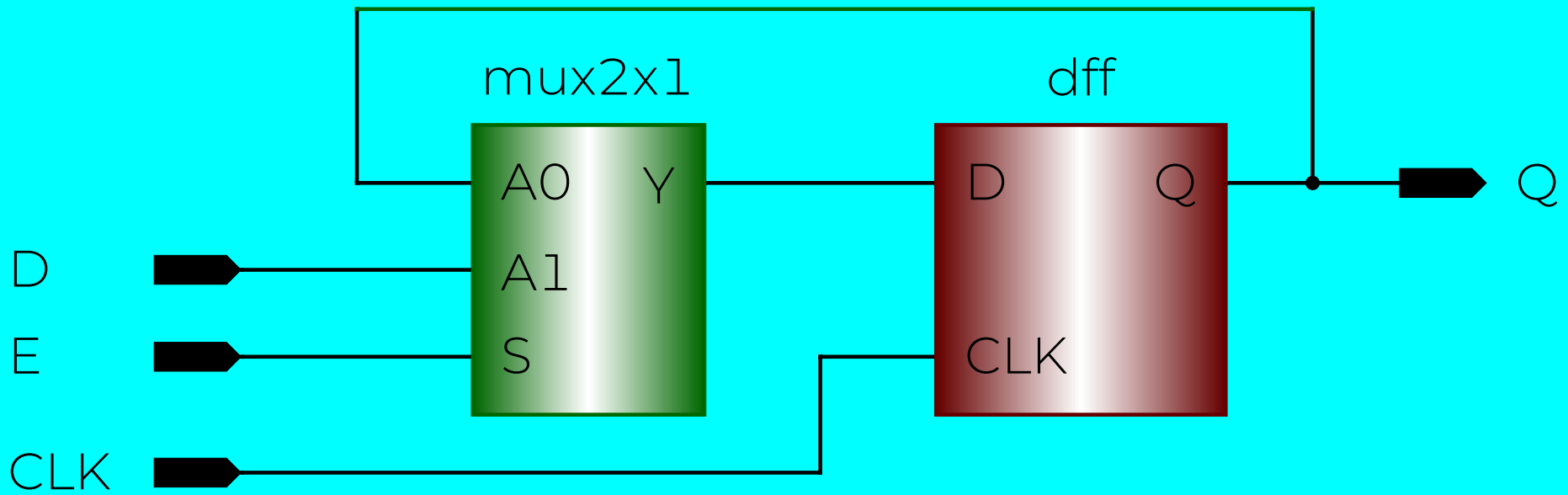


D フリップフロップ



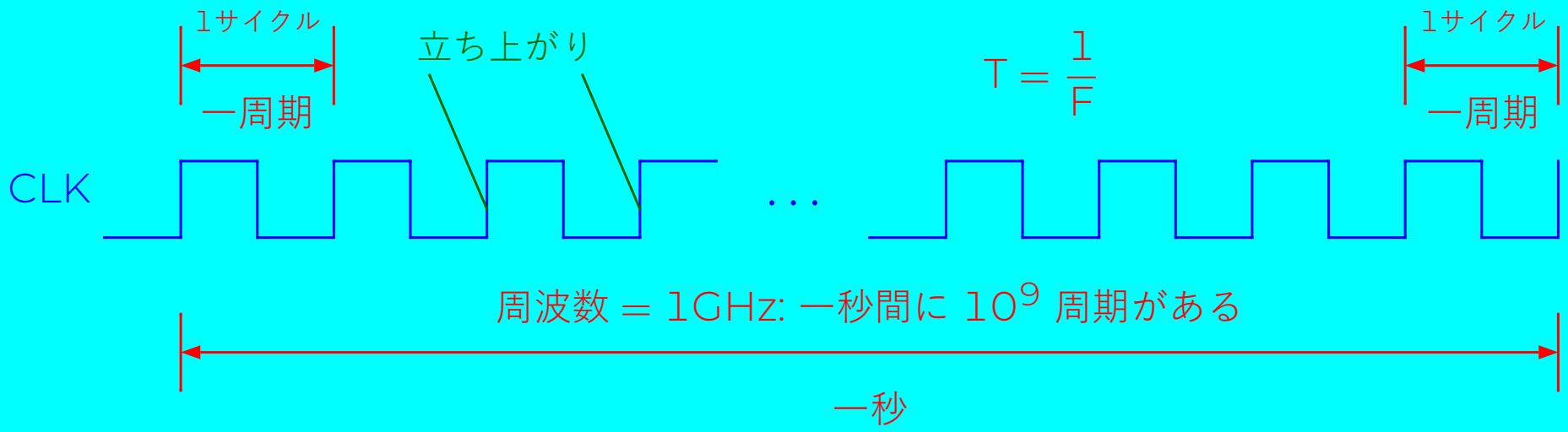
D ラッチ

イネーブル付き DFF (DFFE)



- E = 1: D を選択、DFF に保存
- E = 0: Q を選択、DFF に保存 (変化なし)

クロック信号の周波数

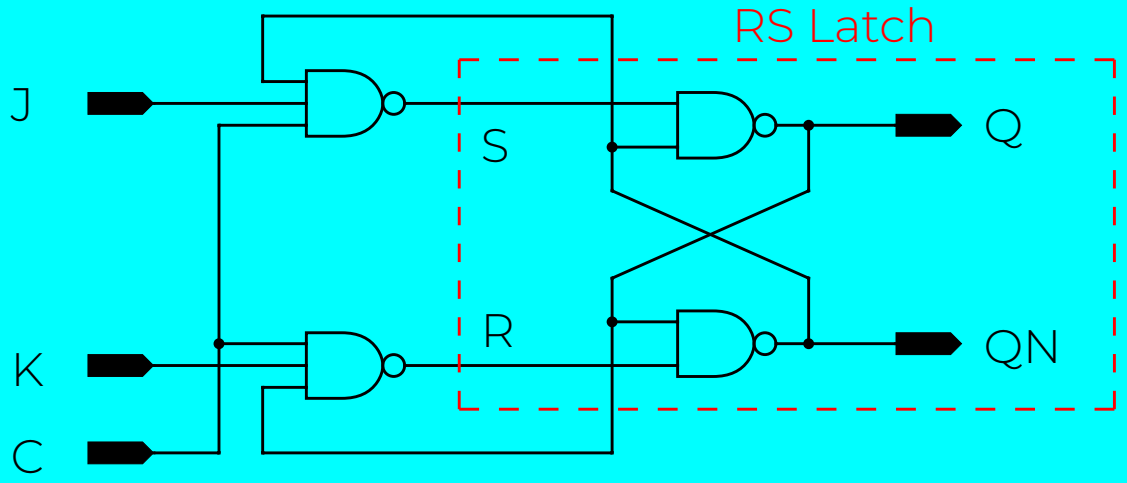


周波数 F = 1GHz: 一周期の時間 $T = \frac{1}{1 \times 10^9} = 10^{-9}\mathrm{s} = 1\mathrm{ns}$

CPU の周波数：$4\mathrm{GHz} = 4 \times 10^9\mathrm{Hz} = 4,000,000,000\mathrm{Hz}$
ここには、$G = 10^9 = 1,000,000,000$

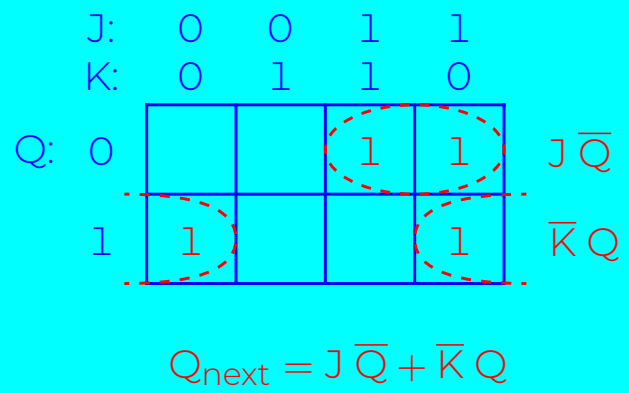
メモリの容量：$8\mathrm{GB} = 8 \times 2^{30}\mathrm{B} = 8,589,934,592$ バイト
ここには、$G = 2^{30} = 1,073,741,824 \neq 10^9$

JK ラッチ



回路図からわかること：

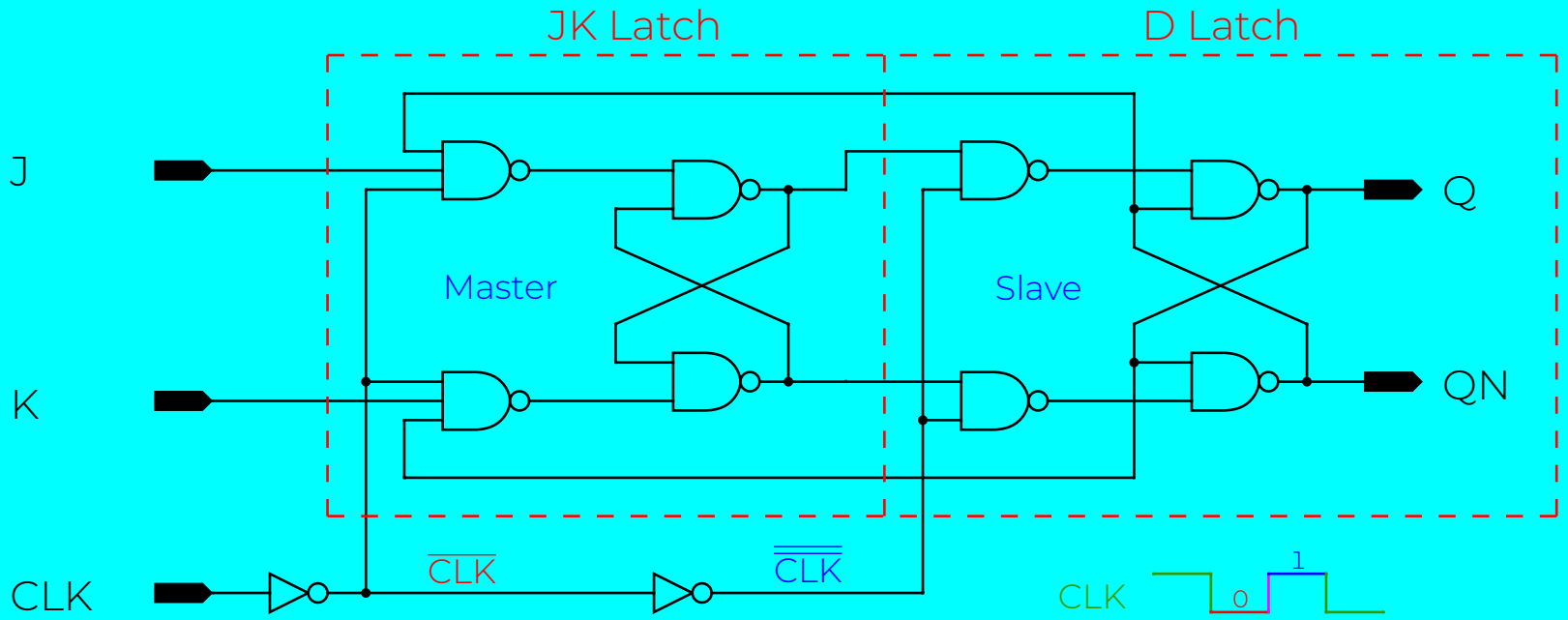
- $C = 1, J = 0, K = 0$: No change $\Rightarrow$
- $C = 1, J = 0, K = 1$: Reset
- $C = 1, J = 1, K = 0$: Set
- $C = 1, J = 1, K = 1$: Toggle ($0 \to 1 \to 0 \to 1 \cdots$, 発振)
- $C = 0, J = x, K = x$: No change



$Q_{next} = J\,\overline{Q} + \overline{K}\,Q$

カルノー図からわかること：$Q_{next} = J\,\overline{Q} + \overline{K}\,Q$

JK フリップフロップ (JKFF)



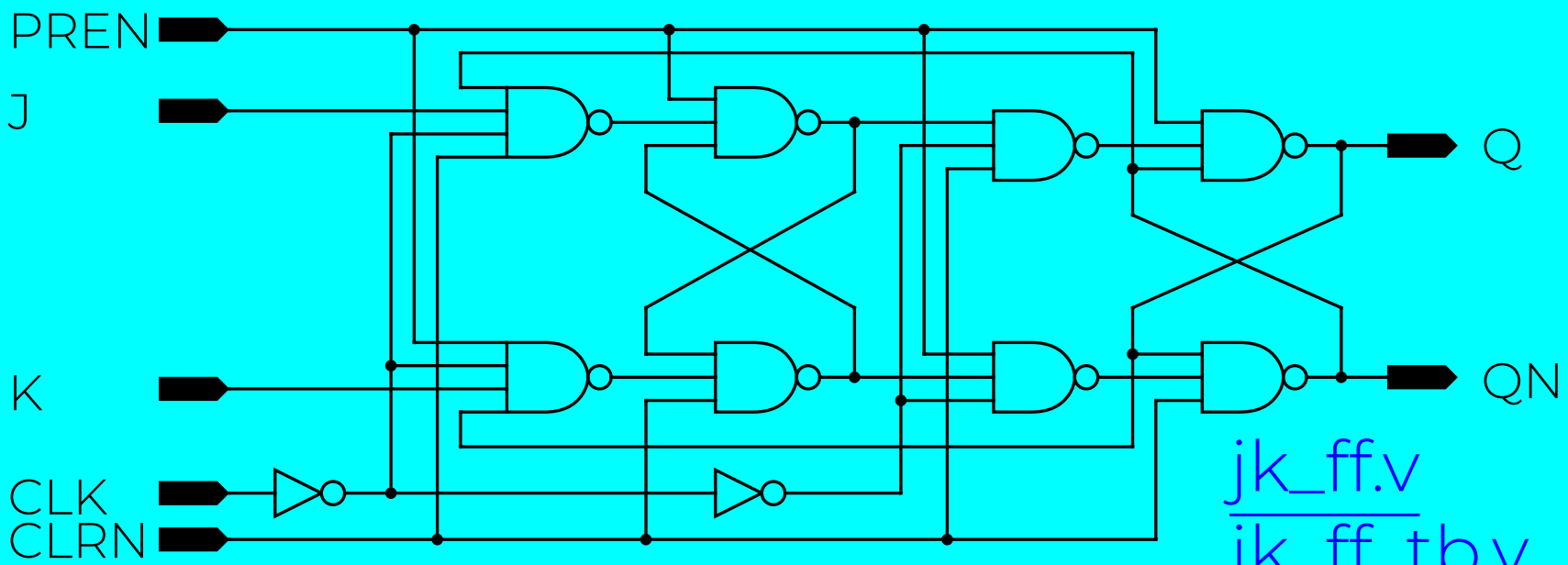
CLK = 0 の時、= Master JK Latch (Slave D Latch 変化なし)

CLK = 1 の時、= Slave D Latch (Master JK Latch 変化なし)

CLK の立ち上がりで、$Q_{next} = J\overline{Q} + \overline{K}Q$

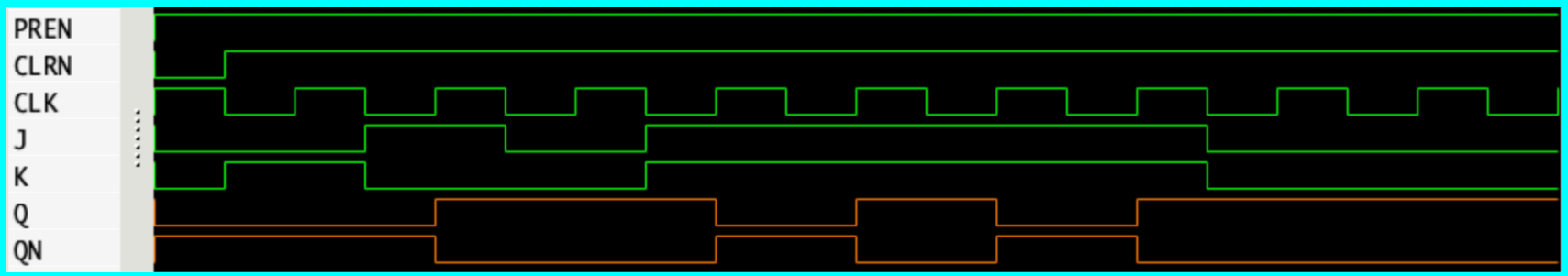
注意：RS ラッチの初期状態がわからないので、Clear などの信号が必要である。

JKFF回路と波形



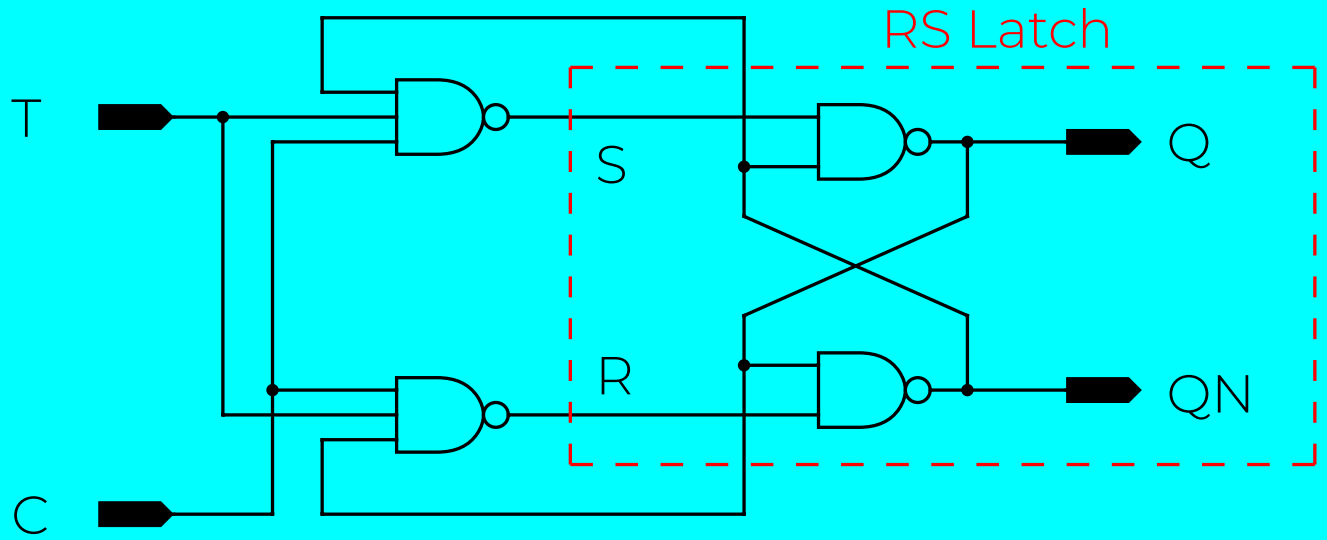
jk_ff.v

jk_ff_tb.v



J=0	J=1	J=0	J=1	J=1	J=1	J=1
K=1	K=0	K=0	K=1	K=1	K=1	K=1
Reset	Set	No change	Toggle	Toggle	Toggle	Toggle

T ラッチ



回路図からわかること：

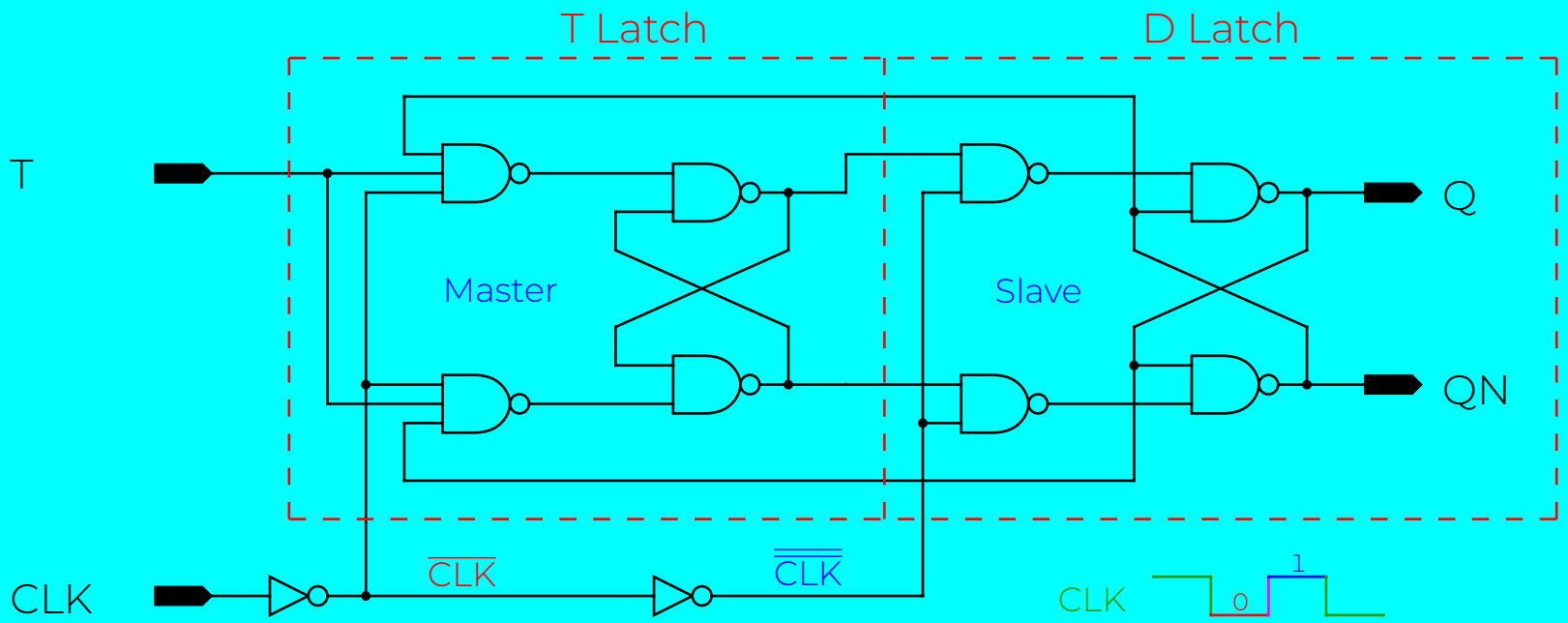
- $C = 1, T = 0$: No change $\Rightarrow$
- $C = 1, T = 1$: Toggle ($0 \to 1 \to 0 \to 1 \cdots$, 発振)
- $C = 0, T = x$: No change

T:	0	1	
Q: 0		1	$T\overline{Q}$
1	1		$\overline{T}Q$

$Q_{next} = T\overline{Q} + \overline{T}Q$

カルノー図からわかること：$Q_{next} = T\overline{Q} + \overline{T}Q$

T フリップフロップ (TFF)



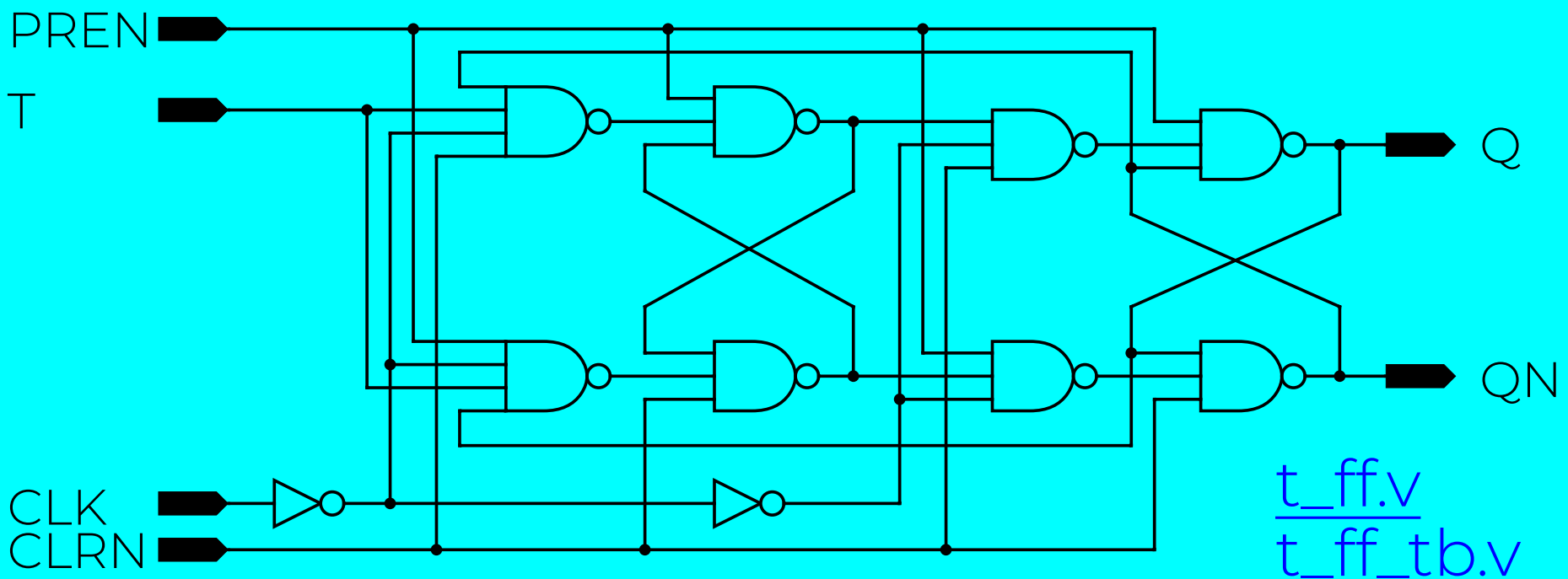
CLK = 0 の時、= Master T Latch (Slave D Latch 変化なし)

CLK = 1 の時、= Slave D Latch (Master T Latch 変化なし)

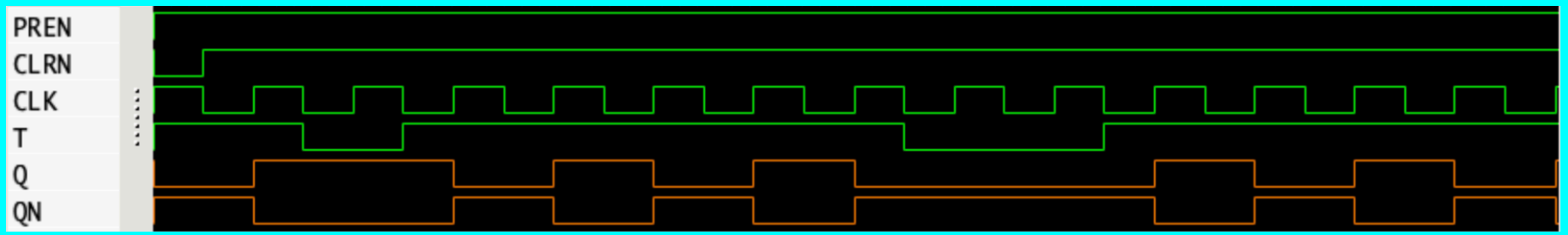
CLK の立ち上がりで、$Q_{next} = T\overline{Q} + \overline{T}Q$

注意：RS ラッチの初期状態がわからないので、Clear などの信号が必要である。

TFF回路と波形



t_ff.v
t_ff_tb.v



T=1 Toggle
T=0 No Change
T=1 Toggle
T=1 Toggle
T=1 Toggle
T=1 Toggle
T=1 Toggle
T=0 No Change
T=0 No Change
T=1 Toggle
T=1 Toggle

D フリップフロップ (1-bit)

```
‘timescale 1ns/1ns
module dffe1  (CLK, CLRN, E, D, Q);
    input         D;
    input         CLK, CLRN, E;
    output        Q;
    reg           Q;
    always @(posedge CLK or negedge CLRN) begin
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            if (E) begin
                Q <= D;
            end
        end
    end
endmodule
```

dffe1.v

D フリップフロップ (4-bit)

```
‘timescale 1ns/1ns
module dffe4  (CLK, CLRN, E, D, Q);
    input  [3:0] D;
    input        CLK, CLRN, E;
    output [3:0] Q;
    reg    [3:0] Q;
    always @(posedge CLK or negedge CLRN) begin
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            if (E) begin
                Q <= D;
            end
        end
    end
endmodule
```

dffe4.v

JK フリップフロップ (1-bit)

```
‘timescale 1ns/1ns
module jkffe1 (CLK, CLRN, E, J, K, Q);
    input         J, K;
    input         CLK, CLRN, E;
    output        Q;
    reg           Q;
    always @(posedge CLK or negedge CLRN) begin
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            if (E) begin
                Q <= J & ~Q | ~K & Q;
            end
        end
    end
endmodule
```

jkffe1.v

JK フリップフロップ (4-bit)

```
‘timescale 1ns/1ns
module jkffe4 (CLK, CLRN, E, J, K, Q);
    input  [3:0] J, K;
    input        CLK, CLRN, E;
    output [3:0] Q;
    reg    [3:0] Q;
    always @(posedge CLK or negedge CLRN) begin
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            if (E) begin
                Q <= J & ~Q | ~K & Q;
            end
        end
    end
endmodule
```

jkffe4.v

T フリップフロップ (1-bit)

```
‘timescale 1ns/1ns
module tffe1  (CLK, CLRN, E, T, Q);
    input         T;
    input         CLK, CLRN, E;
    output        Q;
    reg           Q;
    always @(posedge CLK or negedge CLRN) begin
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            if (E) begin
                Q <= T & ~Q | ~T & Q;
            end
        end
    end
endmodule
```

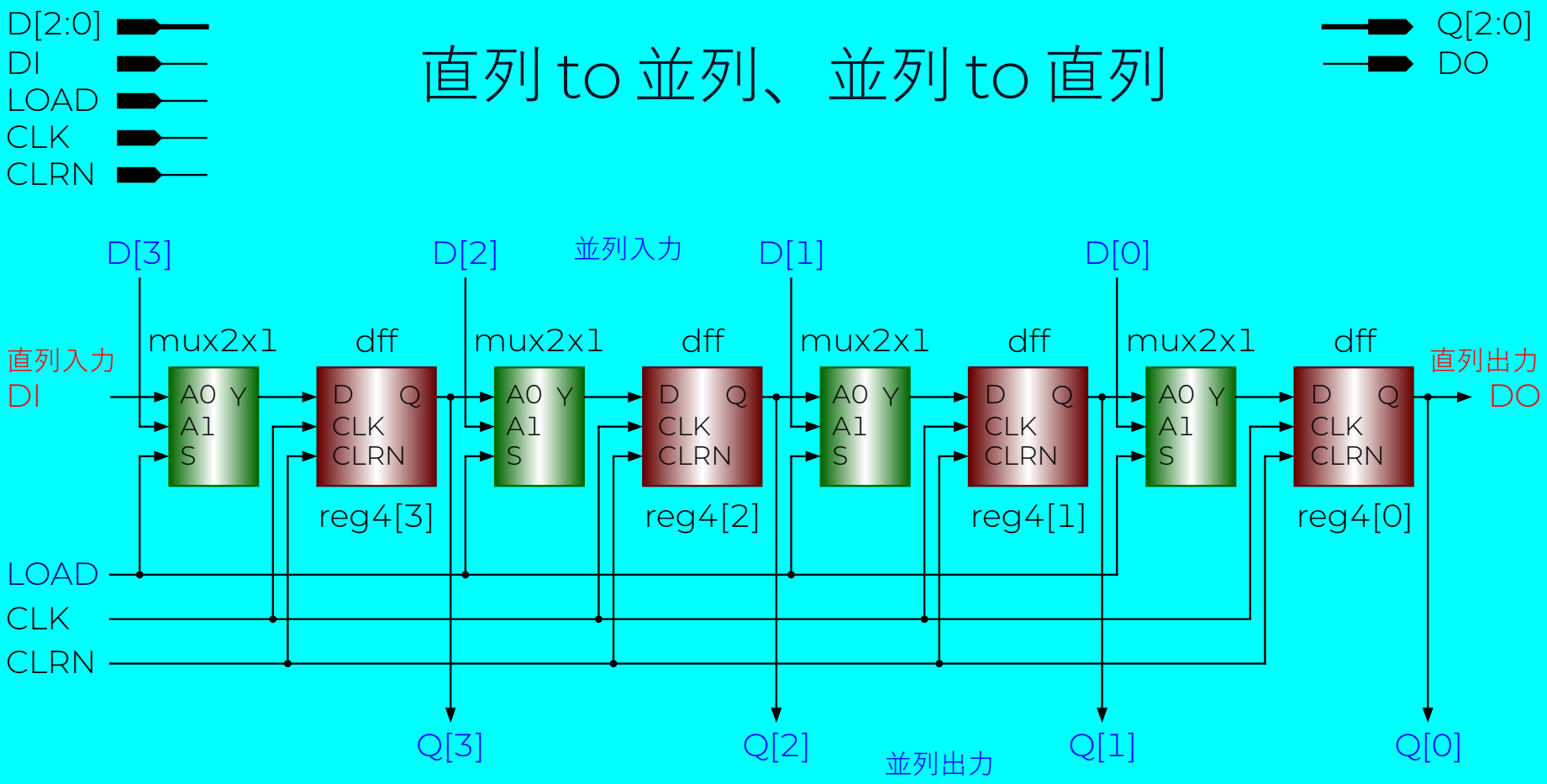
tffe1.v

T フリップフロップ (4-bit)

```
‘timescale 1ns/1ns
module tffe4  (CLK, CLRN, E, T, Q);
    input  [3:0] T;
    input        CLK, CLRN, E;
    output [3:0] Q;
    reg    [3:0] Q;
    always @(posedge CLK or negedge CLRN) begin
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            if (E) begin
                Q <= T & ~Q | ~T & Q;
            end
        end
    end
endmodule
```

tffe4.v

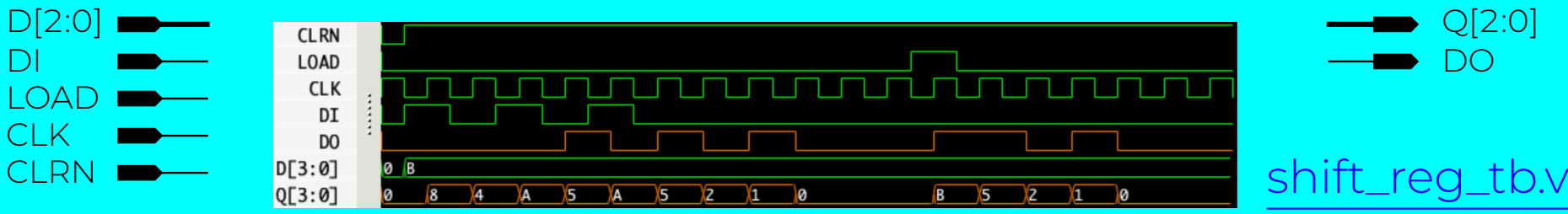
シフトレジスタ (Shift Register)



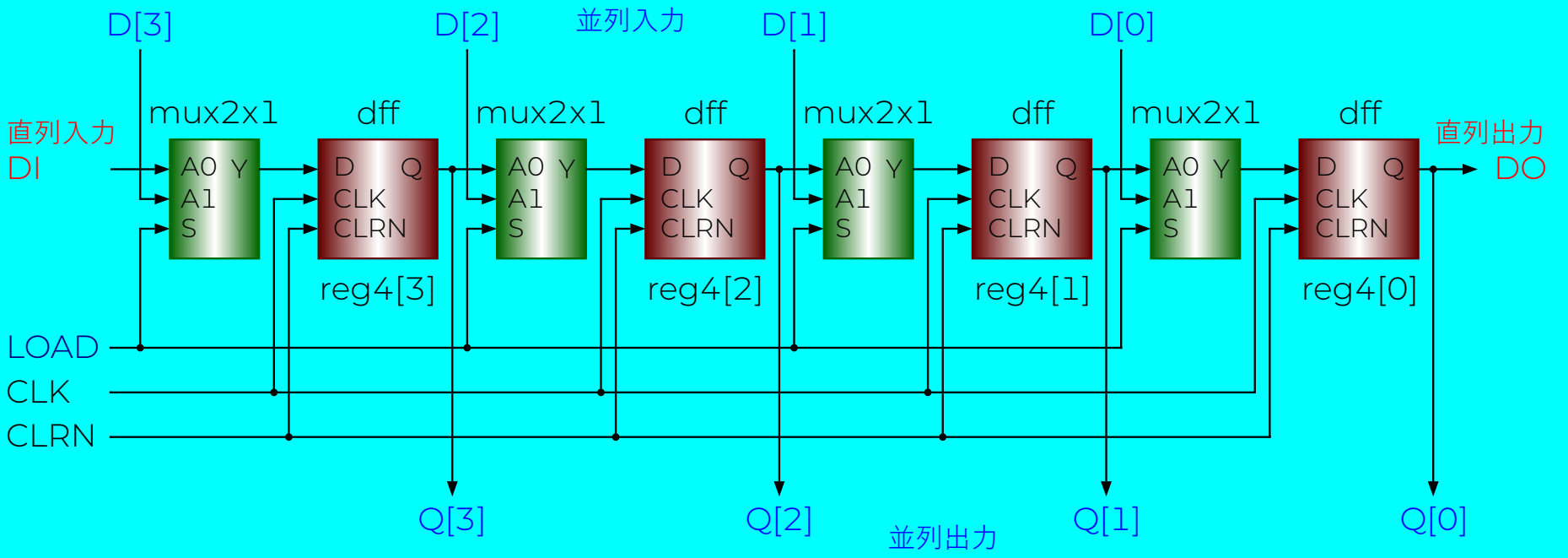
LOAD = 1: Load D[2:0] to DFFs

LOAD = 0: Shift Right

シフトレジスタのシミュレーション



shift_reg_tb.v



LOAD = 1: Load D[2:0] to DFFs

LOAD = 0: Shift Right

注：実際の回路は shift_reg.v を参照してください。

レジスタファイル (Register File)

コンピューターシステム

1. コンピューター

(1) メモリ

(2) 入出力インターフェース

(3) CPU (プロセッサー)

ALU

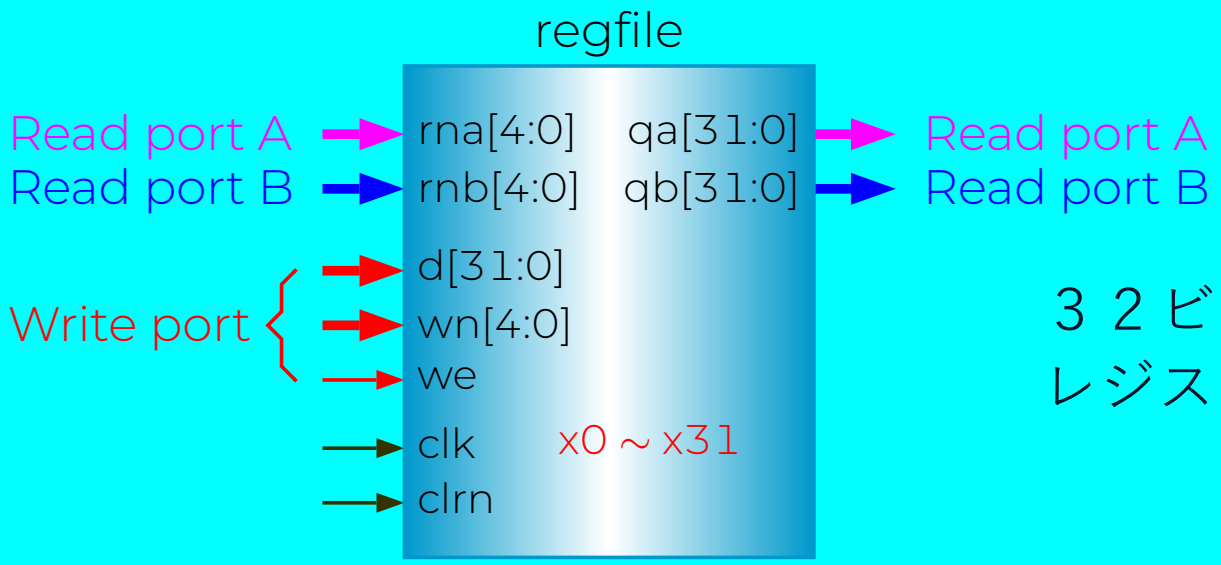
· · ·

レジスタファイル

2. ソフトウェア
(OS やコンパイラなど)

3. 入出力デバイス
(キーボードやディスプレイなど)

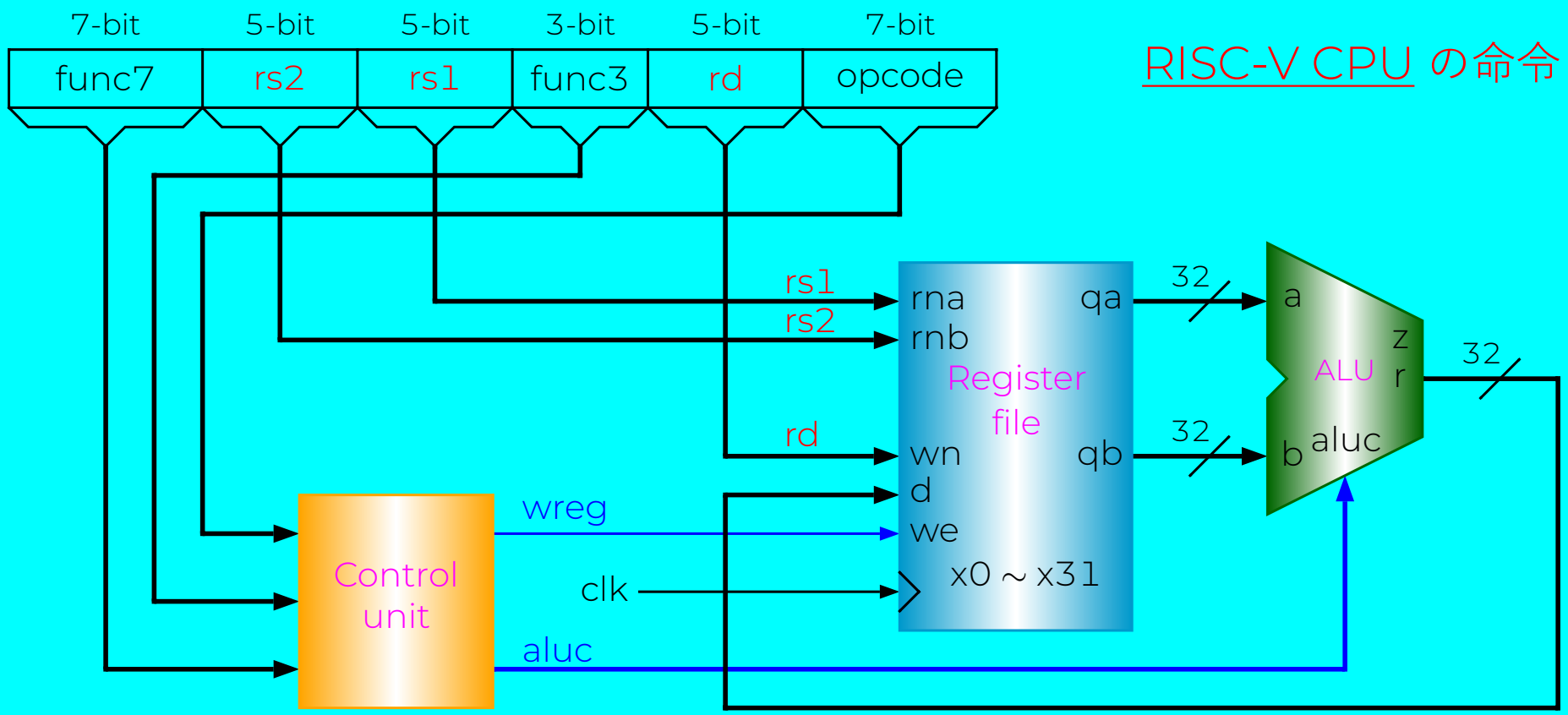
レジスタファイル (シンボル)



３２ビットの
レジスタ３２本 (x0 ～ x31)

rna[4:0] — Register number of read port A (5 bits, $2^5 = 32$)
rnb[4:0] — Register number of read port B (5 bits, $2^5 = 32$)
qa[31:0] — Data output of read port A (32 bits)
qb[31:0] — Data output of read port B (32 bits)
wn[4:0] — Register number of write port (5 bits, $2^5 = 32$)
d[31:0] — Data input of write port (32 bits)
we — Write enable (1 bit)

レジスタファイルの応用

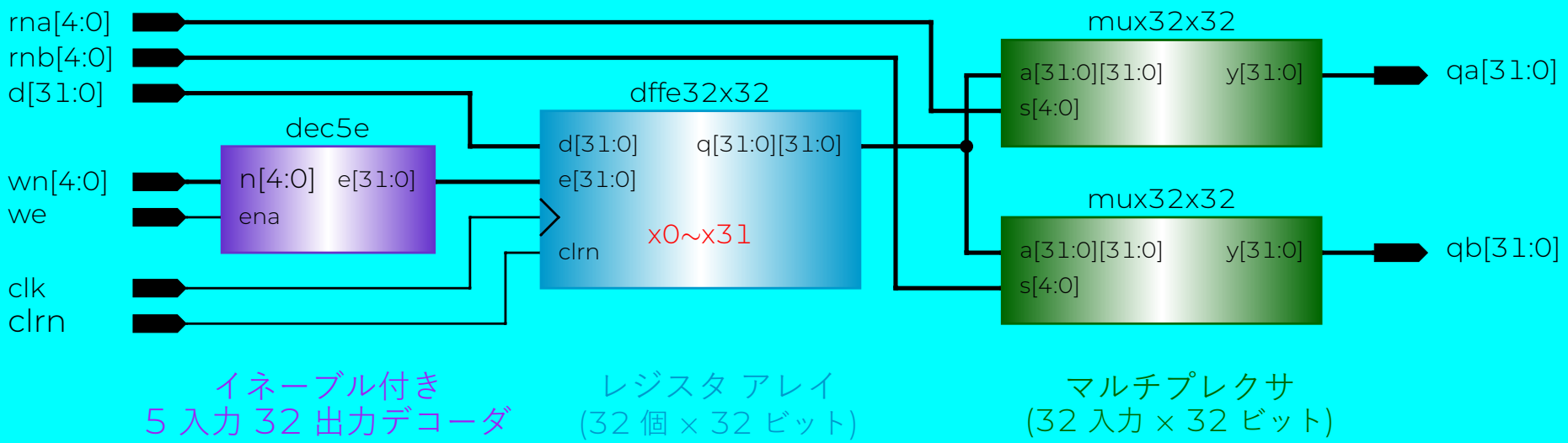


RISC-V 命令の例

```
add  rd, rs1, rs2    # reg[rd] = reg[rs1] + reg[rs2]
```

命令の意味: レジスタ rs1 に格納されている値とレジスタ rs2 に格納されている値を加算して，その結果をレジスタ rd に格納する。

レジスタファイルの回路



rna[4:0] — Register number of read port A (5 bits, $2^5 = 32$)
rnb[4:0] — Register number of read port B (5 bits, $2^5 = 32$)
qa[31:0] — Data output of read port A (32 bits)
qb[31:0] — Data output of read port B (32 bits)
wn[4:0] — Register number of write port (5 bits, $2^5 = 32$)
d[31:0] — Data input of write port (32 bits)
we — Write enable (1 bit)

ラッチとフリップフロップ

まとめ

- 順序回路
- RS ラッチ (RS Latch)
- D ラッチ (D Latch)
- D フリップフロップ (DFF, DFFE)
- T フリップフロップ (TFF, TFFE)
- JK フリップフロップ (JKFF, JKFFE)
- レジスタ (Register)
- レジスタファイル (Register File)

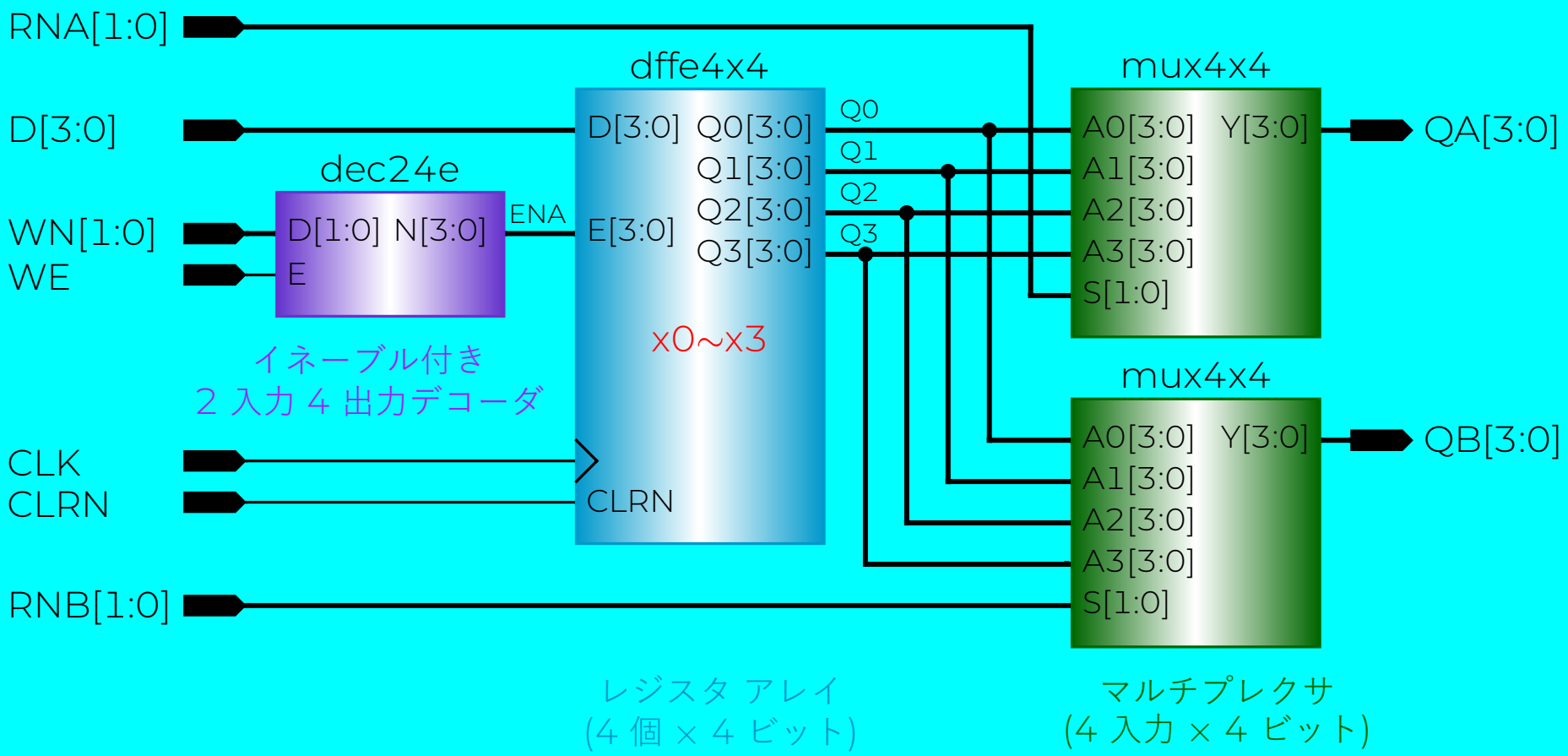
課題 XI (100 点)

４ × ４レジスタ・ファイルを設計し動作検証シミュレーションして下さい。

入力信号: D[3:0] . . . . . 4-bit data
入力信号: RNA[1:0] . . . . . 2-bit port A number
入力信号: RNB[1:0] . . . . . 2-bit port B number
入力信号: WN[1:0] . . . . . 2-bit write number
入力信号: WE . . . . . 1-bit write enable
入力信号: CLK . . . . . 1-bit clock
入力信号: CLRN . . . . . 1-bit clear. 0: clear
出力信号: QA[3:0] . . . . . 4-bit port A data
出力信号: QB[3:0] . . . . . 4-bit port B data

モジュール名は regfile4x4 にすること。
テストベンチ regfile4x4_tb.v を使って下さい。

課題 XI (100 点) 回路



dec24e ・・・・・・・・・・・・・・・・・第 10 回資料 P9 の dec38e を参照

mux4x4 ・・・・・・・・・・・・・・・・・・第 9 回資料課題 IX 問題 1 を参照

課題 XI (100 点) 回路

```
‘timescale 1ns/1ns
module regfile4x4 (CLK, CLRN, WE, D, WN, RNA, RNB, QA, QB);
    input  [3:0] D;
    input  [1:0] RNA, RNB, WN;
    input        CLK, CLRN, WE;
    output [3:0] QA, QB;

    wire   [3:0] ENA;
    wire   [3:0] Q0, Q1, Q2, Q3;

    // dec24e  (D,  E,  N);
    dec24e  i0 (  ,   , ENA); // 2-4 decoder with enable

    // dffe4x4 (CLK, CLRN, E,   D, Q0, Q1, Q2, Q3);
    dffe4x4 i1 (CLK, CLRN, ENA, D, Q0, Q1, Q2, Q3);

    // mux4x4  (A0, A1, A2, A3, S,   Y);
    mux4x4  i2 (  ,   ,   ,   ,    , QA); // QA
    mux4x4  i3 (  ,   ,   ,   ,    , QB); // QB
endmodule
```

regfile4x4.v

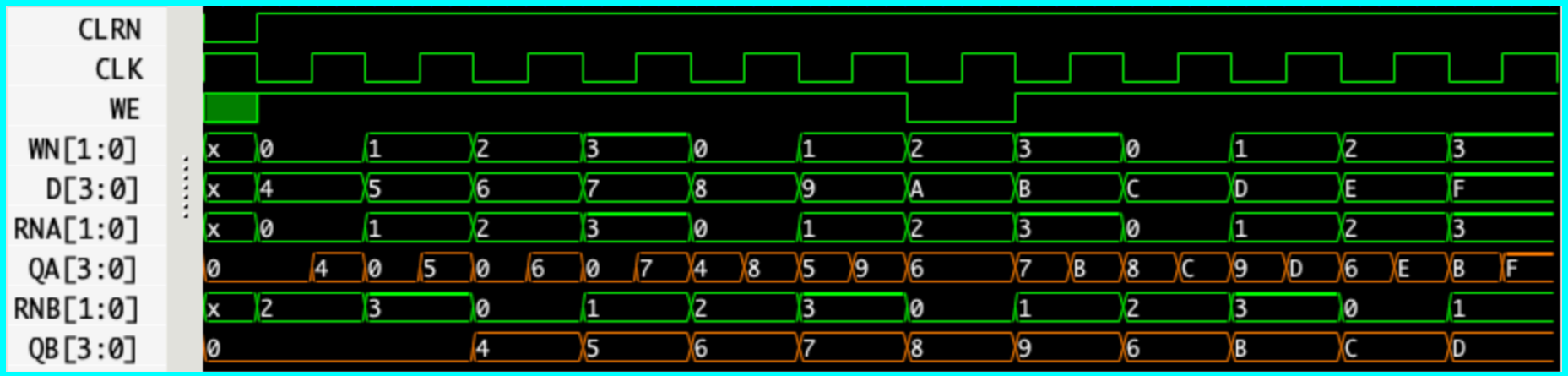
課題 XI (100 点) 回路

```
‘timescale 1ns/1ns
module dffe4x4 (CLK, CLRN, E, D, Q0, Q1, Q2, Q3);
    input  [3:0] E, D;
    input        CLK, CLRN;
    output [3:0] Q0, Q1, Q2, Q3;

    // dffe4 (CLK, CLRN, E,    D, Q);
    dffe4 i0 (CLK, CLRN, E[ ], D, Q0); // reg 0
    dffe4 i1 (CLK, CLRN, E[ ], D, Q1); // reg 1
    dffe4 i2 (CLK, CLRN, E[ ], D, Q2); // reg 2
    dffe4 i3 (CLK, CLRN, E[ ], D, Q3); // reg 3

endmodule
```

dffe4x4.v

課題 XI (100 点) 波形

```
% iverilog -Wall -o regfile4x4 regfile4x4_tb.v regfile4x4.v \
  dffe4x4.v dffe4.v mux4x4.v mux4x1.v dec24e.v
% ./regfile4x4
% gtkwave regfile4x4.vcd
```



シミュレーション波形の考察: CLK の立ち上がりで適当な値を各レジスタに保存して、そのあと、各レジスタに保存している値を読み出し確認して下さい。