

論理回路入門 (11)

ラッチとフリップフロップ

李 亜民

2022 年 12 月 6 日 (火)

ラッチとフリップフロップ

ポイント

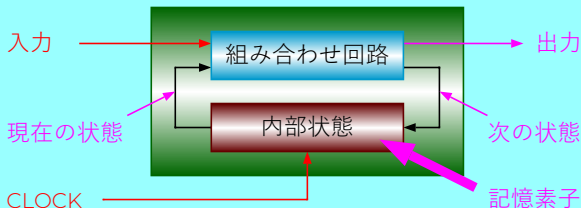
- 順序回路
- RS ラッチ (RS Latch)
- D ラッチ (D Latch)
- D フリップフロップ (DFF, DFFE)
- T フリップフロップ (TFF)
- JK フリップフロップ (JKFF)
- レジスタ (Register)
- レジスタファイル (Register File)

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。

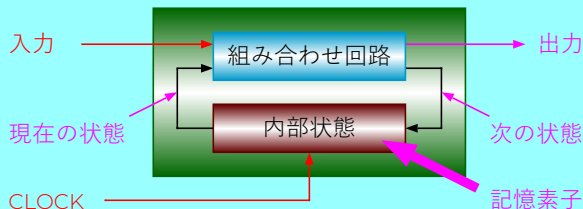


- ② 順序回路 (Sequential Circuit): 内部状態と入力信号で出力が決まる回路である。有限状態機械 (Finite state machine — FSM) とも呼ばれる。



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。



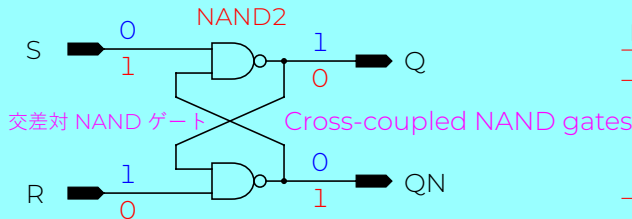
- 順序回路の例：自動販売機の制御回路

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)
 - ▶ JK フリップフロップ (JKFF)
- メモリ (Memory)
 - ▶ SRAM (Static Random Access Memory)
 - ▶ DRAM (Dynamic Random Access Memory)
 - ▶ ROM (Read Only Memory)
 - ▶ CAM (Content Addressable Memory) — 連想メモリ

RS ラッチ



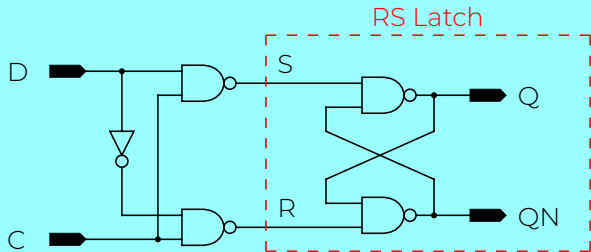
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

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

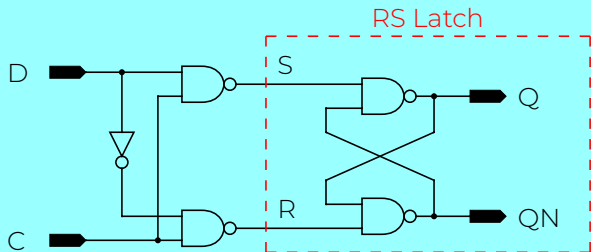


NAND2 ゲートの真理値表

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

- $C = 1, D = 0: S = 1, R = 0, Q = 0 = D$
- $C = 1, D = 1: S = 0, R = 1, Q = 1 = D$
- $C = 0, D = x: S = 1, R = 1, \text{No change}$
 $S = 0, R = 0$ の場合はない

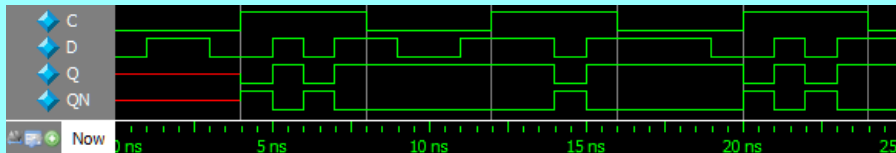
D ラッチシミュレーション



NAND2 ゲートの真理値表

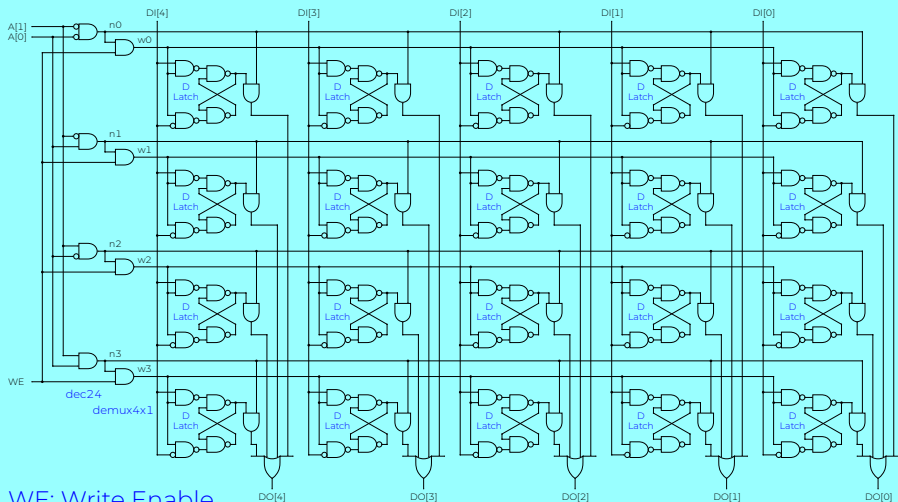
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

[d_latch_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 (回路)

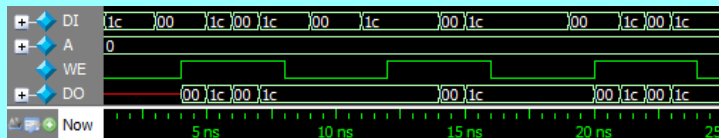


WE: Write Enable

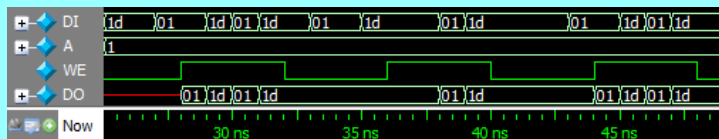
4 words: 2-bit address $A[1..0]$: $2^2 = 4$

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

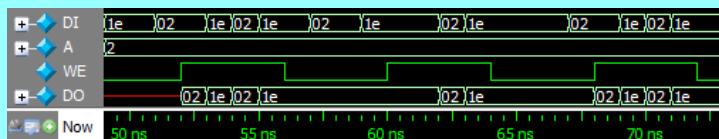
DラッチでSRAMの実装 (波形)



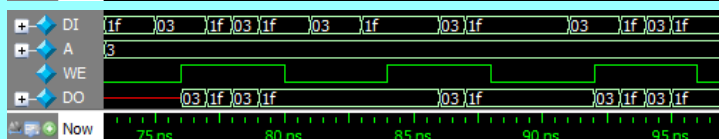
A = 0



A = 1

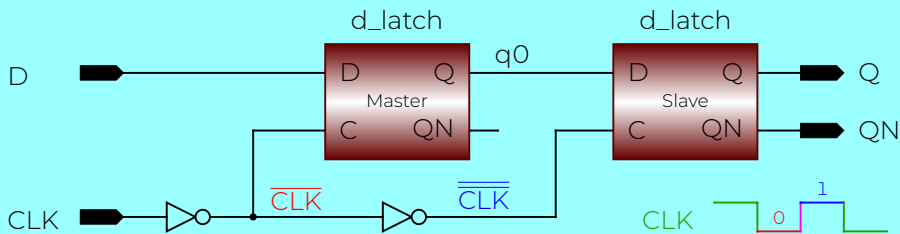


A = 2



A = 3

D フリップフロップ (DFF)

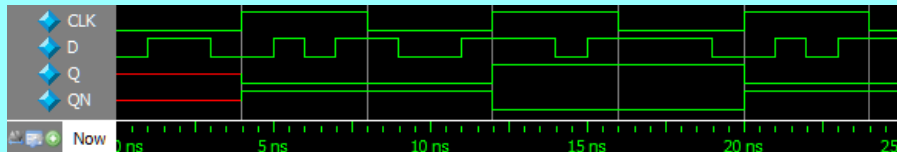
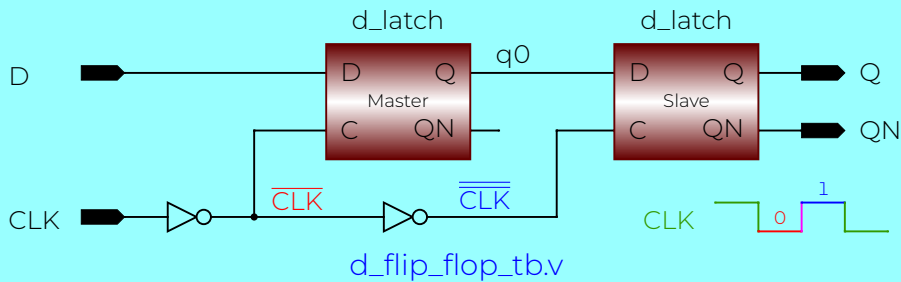


CLK = 0 の時、 $\overline{\text{CLK}} = 1$ 、Master $q0 = D$ (Slave 変化なし)

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

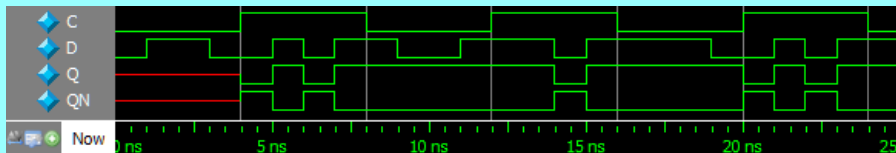
クロック信号 (CLK) の立ち上がり (0 から 1 への遷移) の直前のデータ入力 (D) の値が記憶され、出力 (Q) に出力される。

DFF シミュレーション

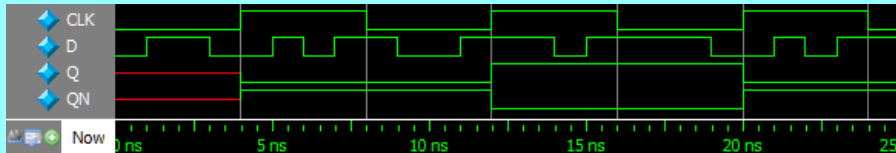


クロック信号 (CLK) の立ち上がり (0 から 1 への遷移) の直前のデータ入力 (D) の値が記憶され、出力 (Q) に出力される。

シミュレーション波形の比較

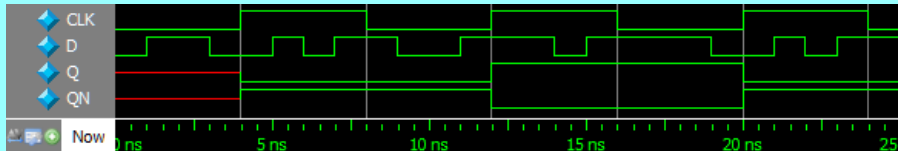


D ラッチ

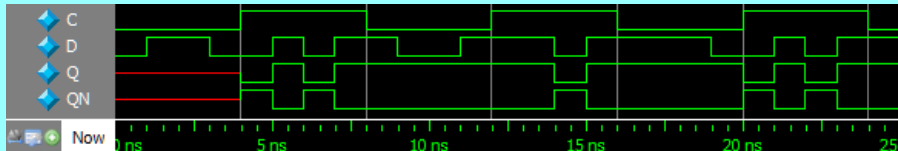


D フリップフロップ

シミュレーション波形の比較

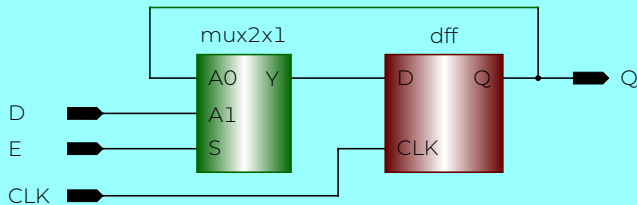


D フリップフロップ



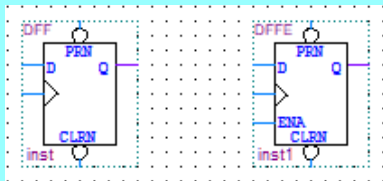
D ラッチ

イネーブル付き DFF (DFFE)

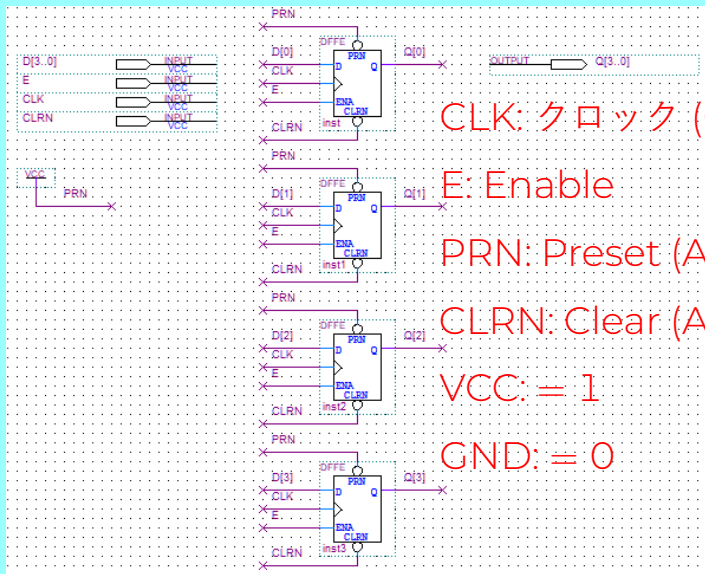


- $E = 1$: D を選択、DFF に保存
- $E = 0$: Q を選択、DFF に保存 (変化なし)

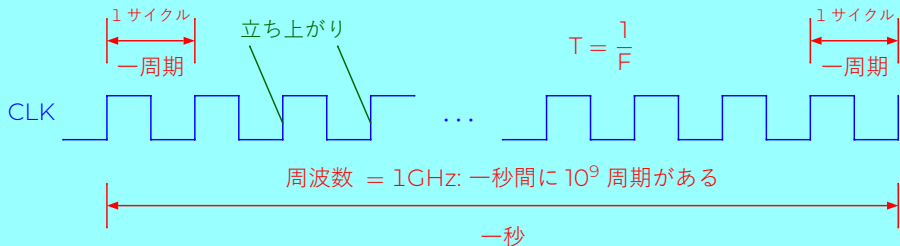
Quartus II では、DFF と DFFE は直接使用できる



4-Bit DFFE (レジスタ)



クロック信号の周波数



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

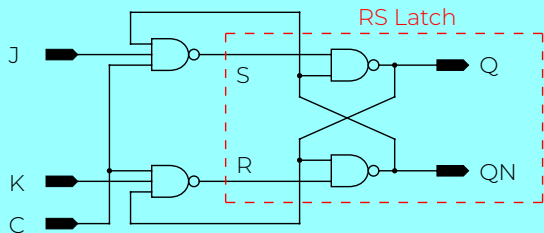
CPU の周波数: $4\text{GHz} = 4,000,000,000\text{Hz}$

ここには、 $G = 10^9 = 1,000,000,000$

メモリの容量: $16\text{GB} = 2^{34}\text{B} = 17,179,869,184$ 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 ($0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \dots$, 発振)
- $C = 0, J = x, K = x$: No change

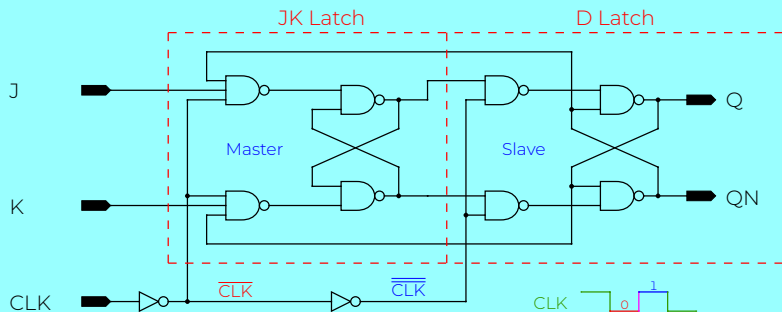
カルノー図からわかること： $Q_{\text{next}} = J\bar{Q} + \bar{K}Q$

J:	0	0	1	1
K:	0	1	1	0
Q:	0		1	1
1	1			1

$J\bar{Q}$
 $\bar{K}Q$

$$Q_{\text{next}} = J\bar{Q} + \bar{K}Q$$

JK フリップフロップ (JKFF)



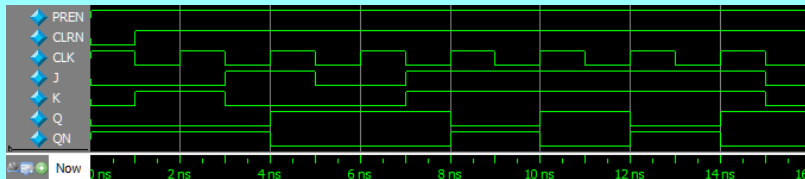
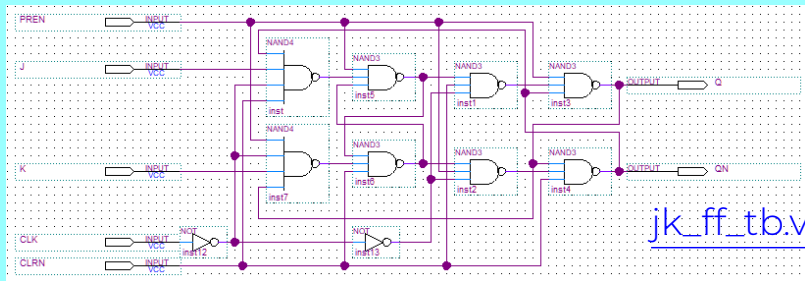
CLK = 0 の時、 = Master JK Latch (Slave D Latch 変化なし)

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

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

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。

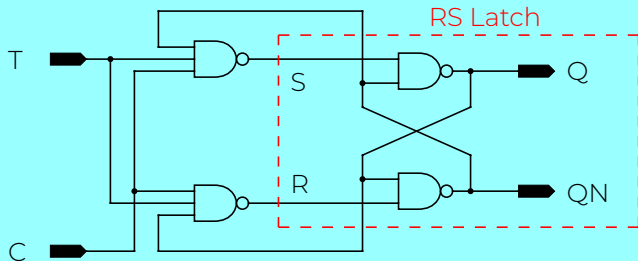
JKFF シミュレーション



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	Toggle	Toggle	Toggle	Toggle

change

Tラッチ



回路図からわかること：

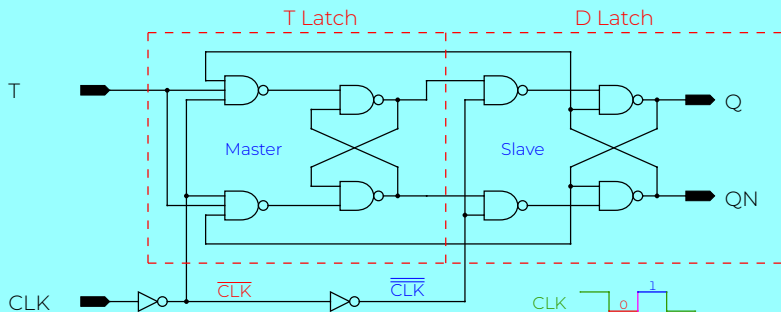
- $C = 1, T = 0$: No change \Rightarrow
- $C = 1, T = 1$: Toggle ($0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \dots$, 発振)
- $C = 0, T = x$: No change

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

$Q_{\text{next}} = T\bar{Q} + \bar{T}Q$

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

T フリップフロップ (TFF)



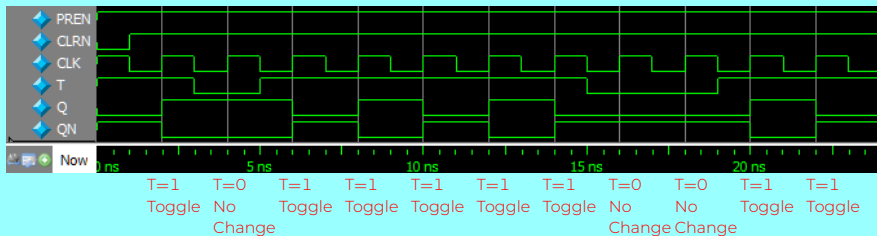
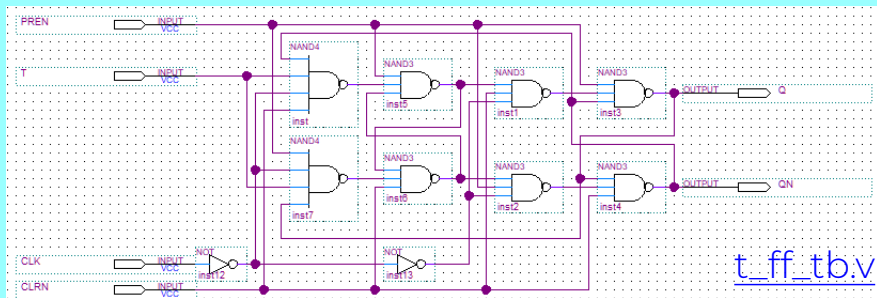
CLK = 0 の時、 = Master T Latch (Slave D Latch 変化なし)

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

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


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 シミュレーション

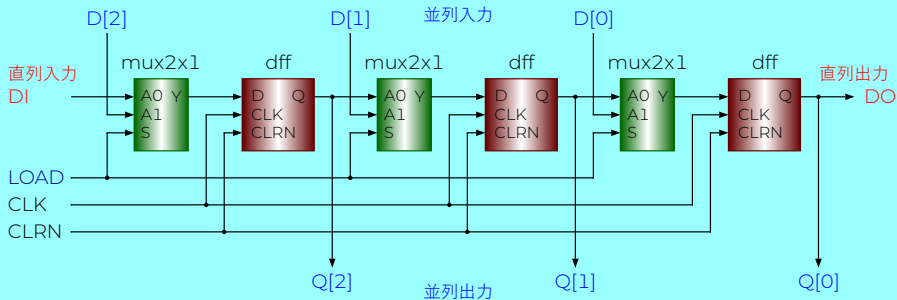


シフトレジスタ (Shift Register)

直列 to 並列, 並列 to 直列

D[2..0] 
DI 
LOAD 
CLK 
CLRn 

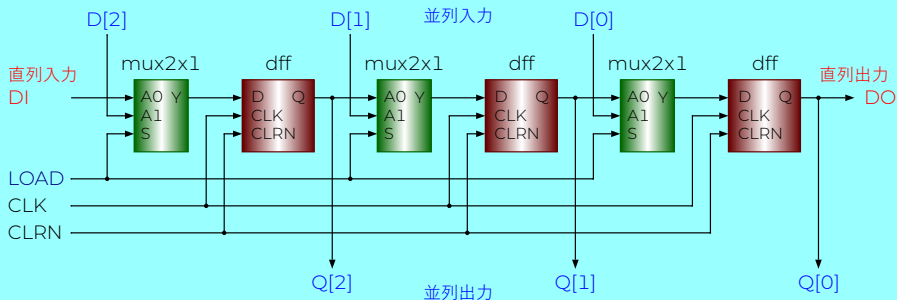
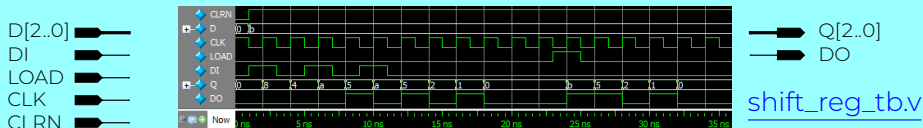
 Q[2..0]
 DO



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



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

LOAD = 0: Shift Right

注：実際の回路には 4 ビットの DFF がある。[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)

ALU

...

レジスタファイル

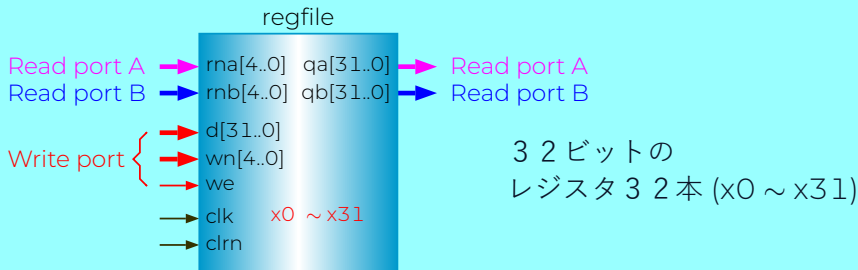
2. ソフトウェア

(OS やコンパイラなど)

3. 入出力デバイス

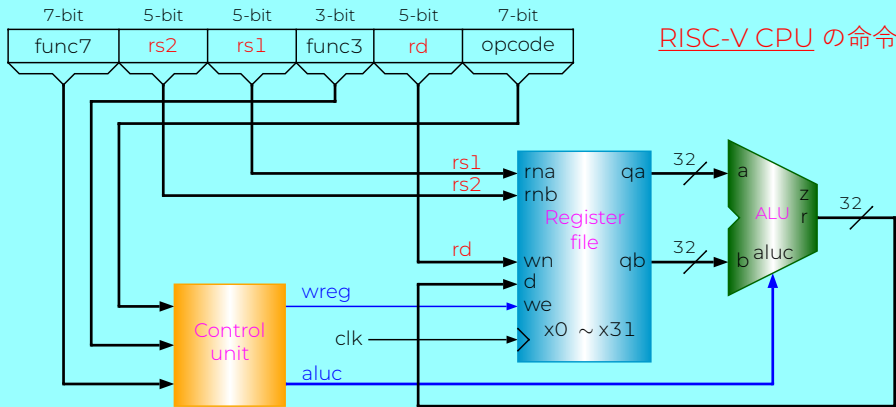
(キーボードやディスプレイなど)

レジスタファイル (シンボル)



- 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



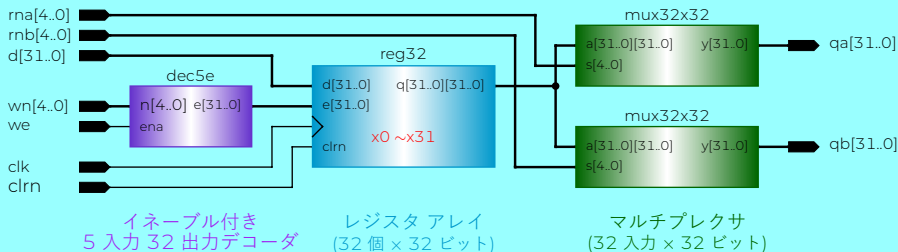
RISC-V CPU の命令

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。

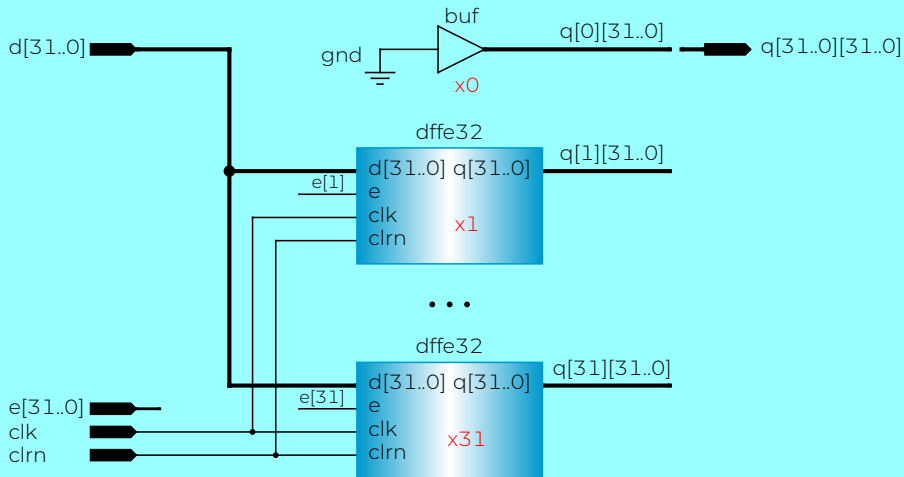
レジスタファイルの回路



- 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)

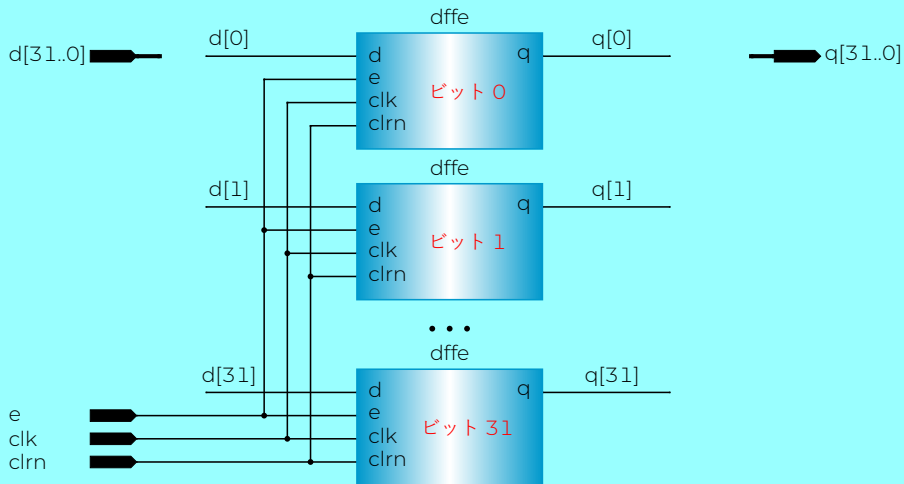
レジスタアレイの回路

32 個 32 ビットレジスタ reg32:



レジスタの回路

32 ビットレジスタ dffe32:



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)
- JK フリップフロップ (JKFF)
- レジスタ (Register)
- レジスタファイル (Register File)

課題 XI (100 点)

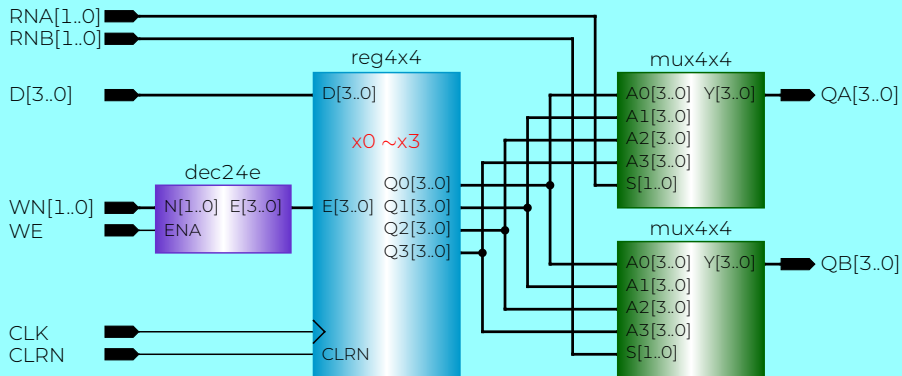
4 × 4 レジスタ・ファイルを回路図で設計し動作検証シミュレーションして下さい。

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

プロジェクト名は `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) 回路図



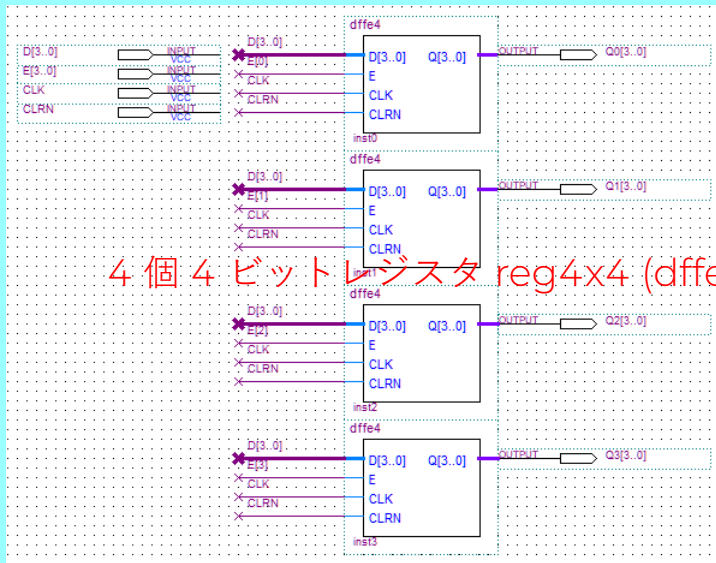
イネーブル付き レジスタ アレイ
2 入力 4 出力デコーダ (4 個 × 4 ビット)

マルチプレクサ
(4 入力 × 4 ビット)

dec24e 第 10 回資料 P8 の dec38e を参照

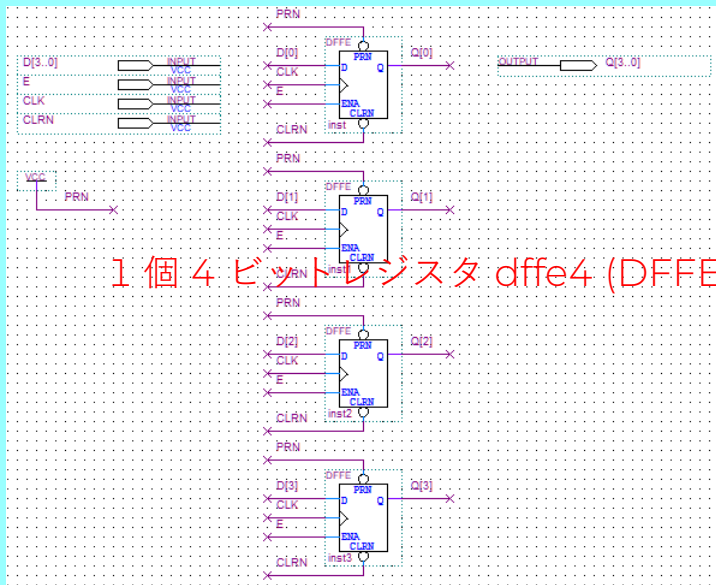
mux4x4 第 9 回資料課題 IX 問題 1 を参照

課題 XI (100 点) reg4x4.bdf



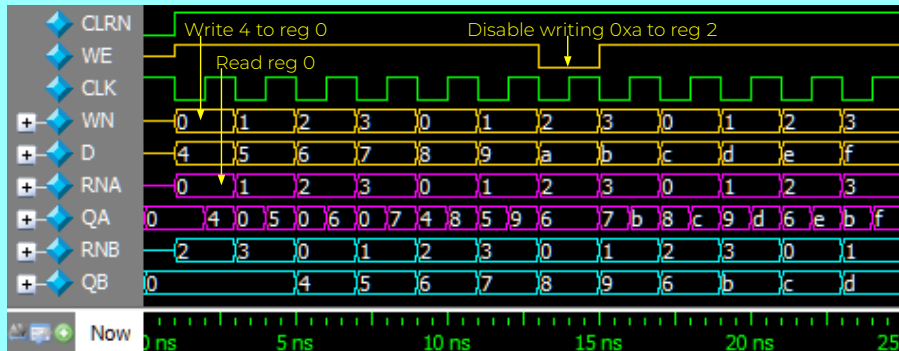
4個4ビットレジスタ reg4x4 (dffe4 を使う)

課題 XI (100 点) dffe4.bdf



1 個 4 ビットレジスタ dffe4 (DFFE を使う)

課題 XI (100 点) 波形



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。特に、0ns ~ 7ns と 13ns ~ 19ns の波形について、説明して下さい。

発展：自由練習

Verilog HDL による課題の実装 1 ビット DFF の設計

```
module dff1b (CLK, CLRN, D, Q);
    input      D;
    input      CLK, CLRN;
    output     Q;
    reg        Q;
    always @(posedge CLK or negedge CLRN) begin
        if (!CLRN) begin
            Q <= 0;
        end else begin

            Q <= D;

        end
    end
end
endmodule
```

発展：自由練習

Verilog HDL による課題の実装 1 ビット DFFE の設計

```
module dffe1b (CLK, CLRN, E, D, Q);
    input      D;
    input      CLK, CLRN, E;
    output     Q;
    reg        Q;
    always @(posedge CLK or negedge CLRN) begin
        if (!CLRN) begin
            Q <= 0;
        end else begin
            if (E) begin
                Q <= D;
            end
        end
    end
end
endmodule
```

発展：自由練習

Verilog HDL による課題の実装 4ビット DFFE の設計

```
module dffe4b (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) begin
      Q <= 0;
    end else begin
      if (E) begin
        Q <= D;
      end
    end
  end
end
endmodule
```


発展：自由練習

Verilog HDL による課題の実装

4 個 4 ビットレジスタの設計 (P31 を参照)

```
module dffe4bx4 (CLK, CLRN, E, D, Q0, Q1, Q2, Q3);
  input  [3:0] D;
  input          CLK, CLRN;
  input  [3:0] E;
  output [3:0] Q0, Q1, Q2, Q3;
  dffe4b i0 (CLK, CLRN, E[0], D, Q0); // reg 0
  dffe4b i1 (          ); // reg 1
  dffe4b i2 (          ); // reg 2
  dffe4b i3 (          ); // reg 3
endmodule
```

上記コードを完成しシミュレーションして下さい。

発展：自由練習

Verilog HDL による課題の実装

4個4ビットレジスタ・ファイルの設計（P30を参照）

```
module regfile4x4b (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;
    dec24ena i0 (WN, WE, ENA); // 2-4 decoder with enable
    dffe4bx4 i1 (CLK, CLRN, ENA, D, Q0, Q1, Q2, Q3);
    mux4x4b i2 (
                                ); // output QA
    mux4x4b i3 (
                                ); // output QB
endmodule
```

上記コードを完成しシミュレーションして下さい。

発展：自由練習

32 個 32 ビット RISC-V レジスタ・ファイルの設計

```
module regfile32x32 (rs1, rs2, d, rd, we, clk, rst_n, q1, q2);
  input  [31:0] d;                // data to be written
  input   [4:0] rs1, rs2, rd;    // RISC-V register numbers
  input                clk, rst_n, we; // we: write enable
  output [31:0] q1, q2;         // two outputs
  reg    [31:0] x [1:31];       // 31 registers, x[0] = 0
  assign q1 = (rs1 == 0) ? 0 : x[rs1]; // read port 1
  assign q2 = (rs2 == 0) ? 0 : x[rs2]; // read port 2
  integer i;
  always @(posedge clk or negedge rst_n)
    if (!rst_n) // active low reset
      for (i = 1; i < 32; i = i + 1) // in parallel
        x[i] <= 0; // clear all registers
    else
      if ((rd != 0) && we)
        x[rd] <= d; // write port
endmodule
```