論理回路入門（12）

順序回路３：交通信号機制御システム設計

李　亜民

2024 年 12 月 10 日 (火)

順序回路

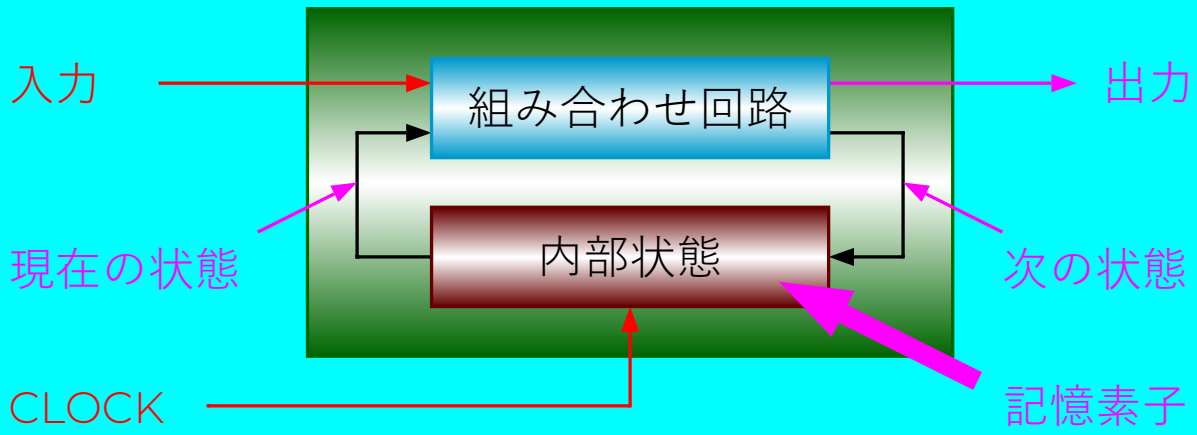
ポイント

- 順序回路の基本構造
- 順序回路の分析
- Mealy 型と Moore 型順序回路
- 順序回路設計の手順
- 状態遷移図
- 次の状態の真理値表、論理式、および回路
- 出力関数の真理値表、論理式、および回路
- 交通信号機制御システム

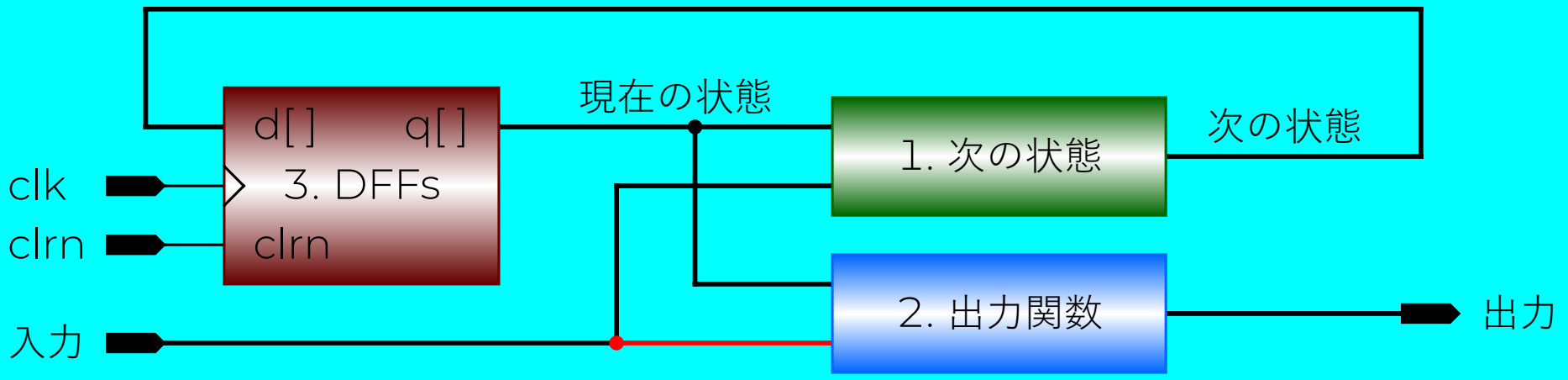
論理回路の種類

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。

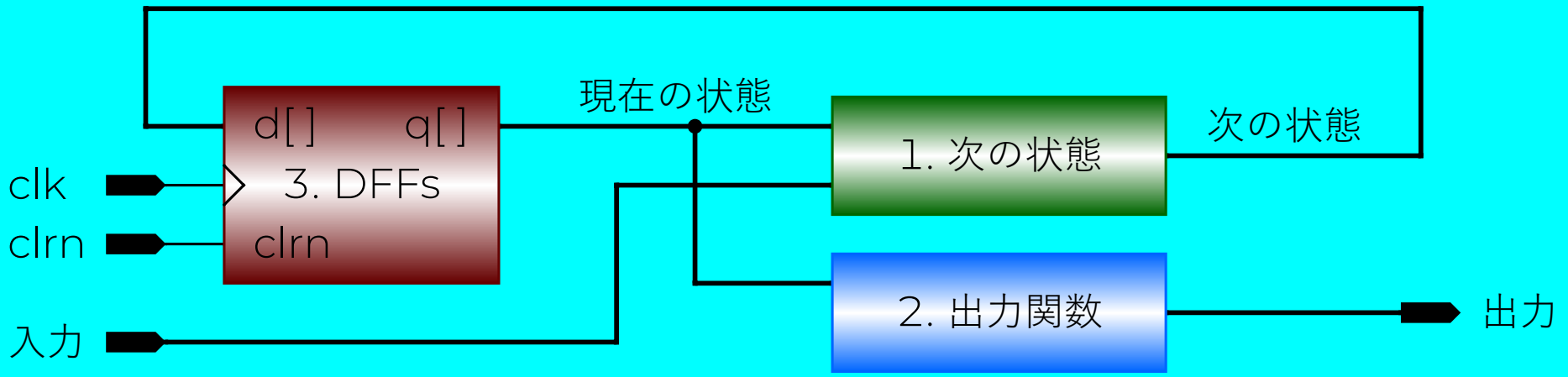
2. 順序回路 (Sequential Circuit): 内部状態と入力信号で出力が決まる回路である。有限状態機械 (Finite state machine — FSM) とも呼ばれる。



順序回路 — Mealy 型と Moore 型



(a) Mealy 型



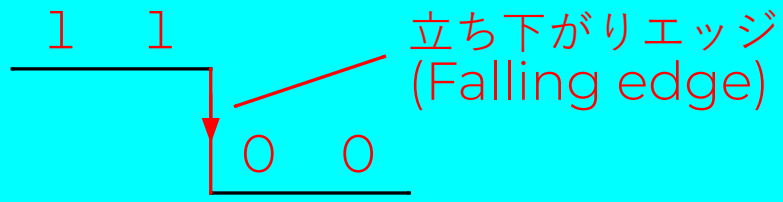
(b) Moore 型

順序回路設計の手順

1. 問題を理解する。
2. 状態遷移図をつくる。
3. FF の数を決める ($n = \lceil \log_2 N \rceil$、N は状態の数)。
4. 各状態に n ビットの番号を付け、真理値表をつくる。
   - 次の状態の真理値表をつくる。
   - 出力関数の真理値表をつくる。
5. 真理値表から論理式をつくる (どの条件で出力が 1 になるか)。
   - カルノー図を用いて論理式を簡単化する。
6. 論理式から回路をつくる。
7. テストベンチをつくる (シミュレーションするため)。
8. 回路をシミュレーションする (回路設計の正当性検証)。
9. 与えられた回路の動作を理解する (波形の説明)。

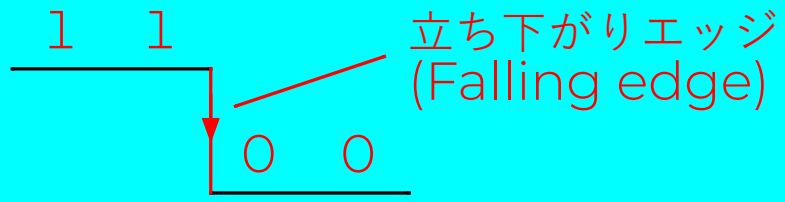
順序回路の設計（問題と状態遷移図）

問題：入力信号が１ビットで、入力列が“1100”の場合に１を出力する (立ち下がりエッジ検出) 順序回路を設計せよ。

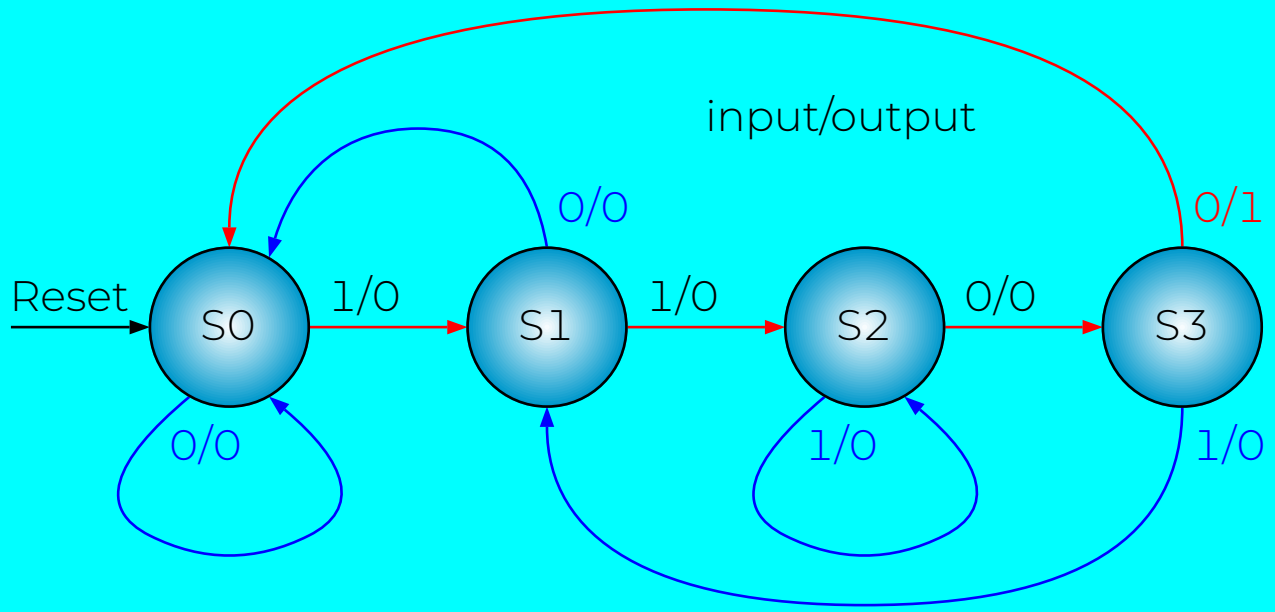


順序回路の設計（問題と状態遷移図）

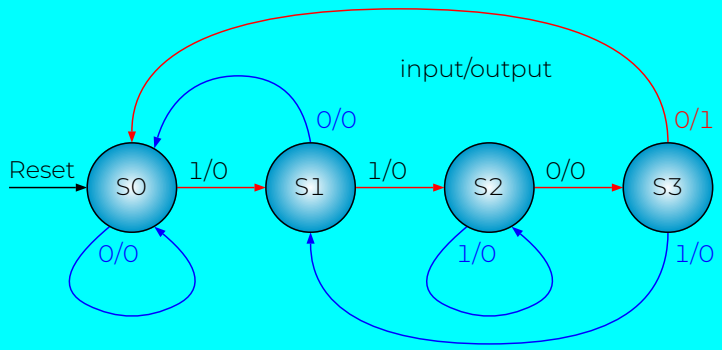
問題：入力信号が１ビットで、入力列が“1100”の場合に１を出力する (立ち下がりエッジ検出) 順序回路を設計せよ。



状態遷移図：

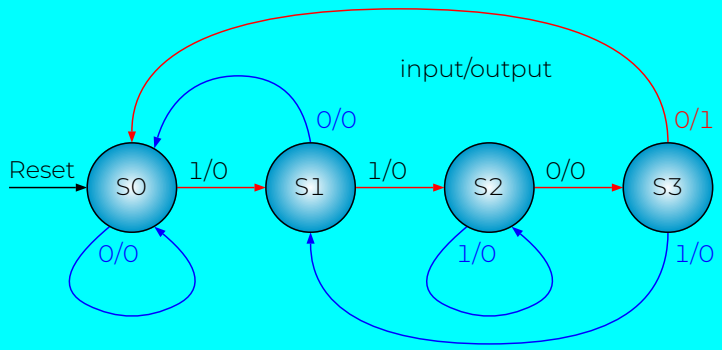


順序回路の設計（真理値表）



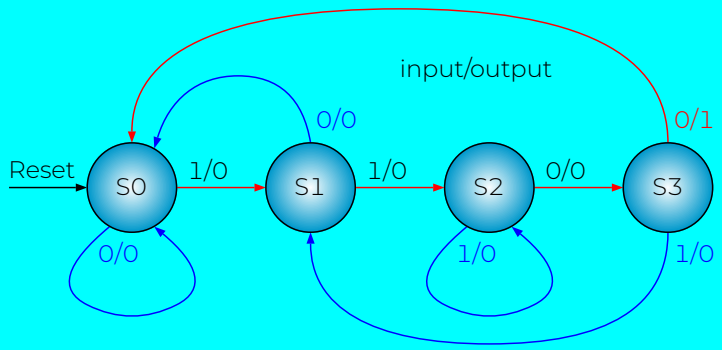
現在の状態			入力	次の状態		出力
	q[1]	q[0]	c	d[1]	d[0]	r
S0	0	0	0			

順序回路の設計（真理値表）



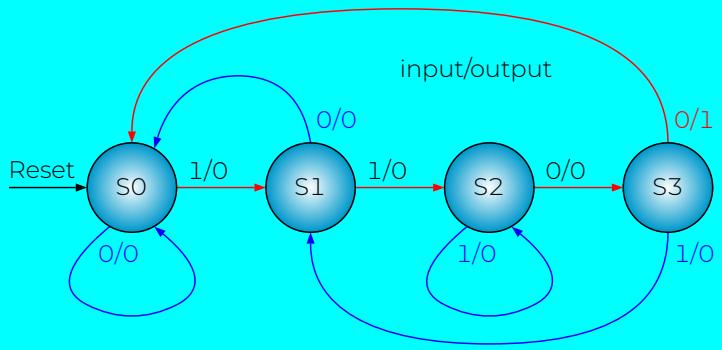
現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1				

順序回路の設計（真理値表）



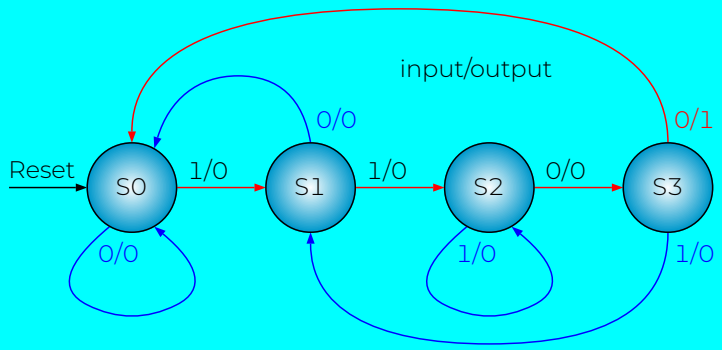
現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0				

順序回路の設計（真理値表）



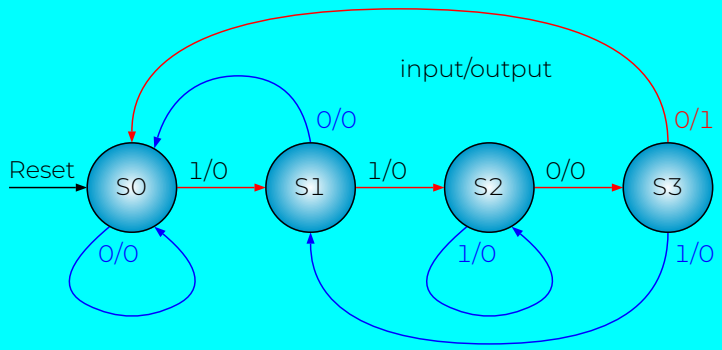
現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0	S0	0	0	0
			1				

順序回路の設計（真理値表）



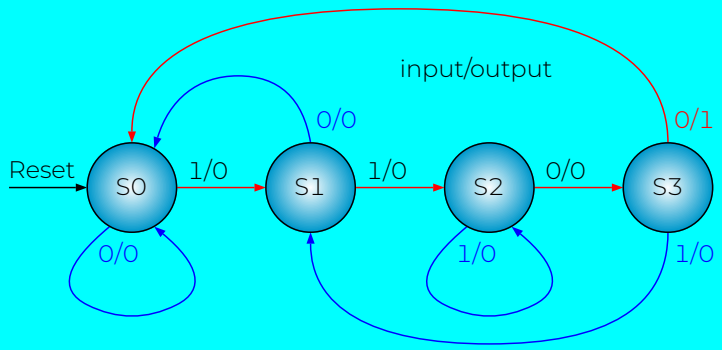
現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0	S0	0	0	0
			1	S2	1	0	0
S2	1	0	0				

順序回路の設計（真理値表）



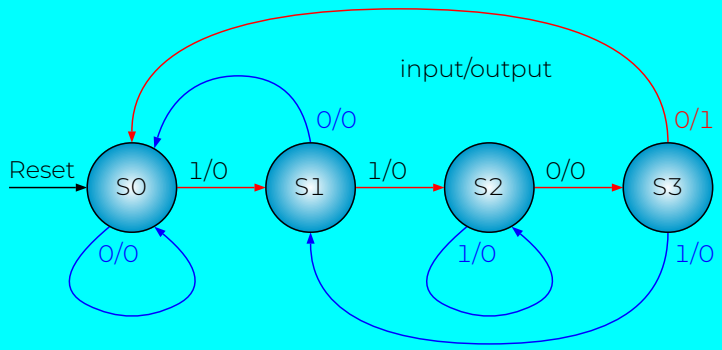
現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0	S0	0	0	0
			1	S2	1	0	0
S2	1	0	0	S3	1	1	0
			1				

順序回路の設計（真理値表）



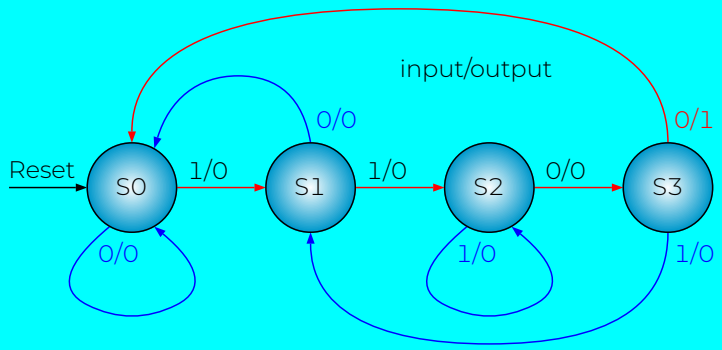
現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0	S0	0	0	0
			1	S2	1	0	0
S2	1	0	0	S3	1	1	0
			1	S2	1	0	0
S3	1	1	0				

順序回路の設計（真理値表）



現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0	S0	0	0	0
			1	S2	1	0	0
S2	1	0	0	S3	1	1	0
			1	S2	1	0	0
S3	1	1	0	S0	0	0	1
			1				

順序回路の設計（真理値表）



現在の状態			入力	次の状態			出力
	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0	S0	0	0	0
			1	S2	1	0	0
S2	1	0	0	S3	1	1	0
			1	S2	1	0	0
S3	1	1	0	S0	0	0	1
			1	S1	0	1	0

順序回路の設計（カルノー図）

d[1]

c \ q[1]q[0]	00	01	11	10
0				1
1		1		1

$\overline{q[1]} \cdot q[0] \cdot c \quad q[1] \cdot \overline{q[0]}$

d[0]

c \ q[1]q[0]	00	01	11	10
0				1
1	1		1	

$\overline{q[1]} \cdot \overline{q[0]} \cdot c \quad q[1] \cdot \overline{q[0]} \cdot \overline{c} \quad q[1] \cdot q[0] \cdot c$

r

c \ q[1]q[0]	00	01	11	10
0			1	
1				

$q[1] \cdot q[0] \cdot \overline{c}$

	q[1]	q[0]	c		d[1]	d[0]	r
S0	0	0	0	S0	0	0	0
			1	S1	0	1	0
S1	0	1	0	S0	0	0	0
			1	S2	1	0	0
S2	1	0	0	S3	1	1	0
			1	S2	1	0	0
S3	1	1	0	S0	0	0	1
			1	S1	0	1	0

順序回路の設計（カルノー図と論理式）

d[1]

q[1]:	0	0	1	1
q[0]:	0	1	1	0
c: 0				1
1		1		1

$\overline{q[1]} \cdot q[0] \cdot c$　　$q[1] \cdot \overline{q[0]}$

d[0]

q[1]:	0	0	1	1
q[0]:	0	1	1	0
c: 0				1
1	1		1	

$\overline{q[1]} \cdot \overline{q[0]} \cdot c$　　$q[1] \cdot \overline{q[0]} \cdot \overline{c}$

$q[1] \cdot q[0] \cdot c$

r

q[1]:	0	0	1	1
q[0]:	0	1	1	0
c: 0			1	
1				

$q[1] \cdot q[0] \cdot \overline{c}$

$$d[1] = \overline{q[1]} \cdot q[0] \cdot c + q[1] \cdot \overline{q[0]}$$

$$d[0] = \overline{q[1]} \cdot \overline{q[0]} \cdot c + q[1] \cdot q[0] \cdot c + q[1] \cdot \overline{q[0]} \cdot \overline{c}$$

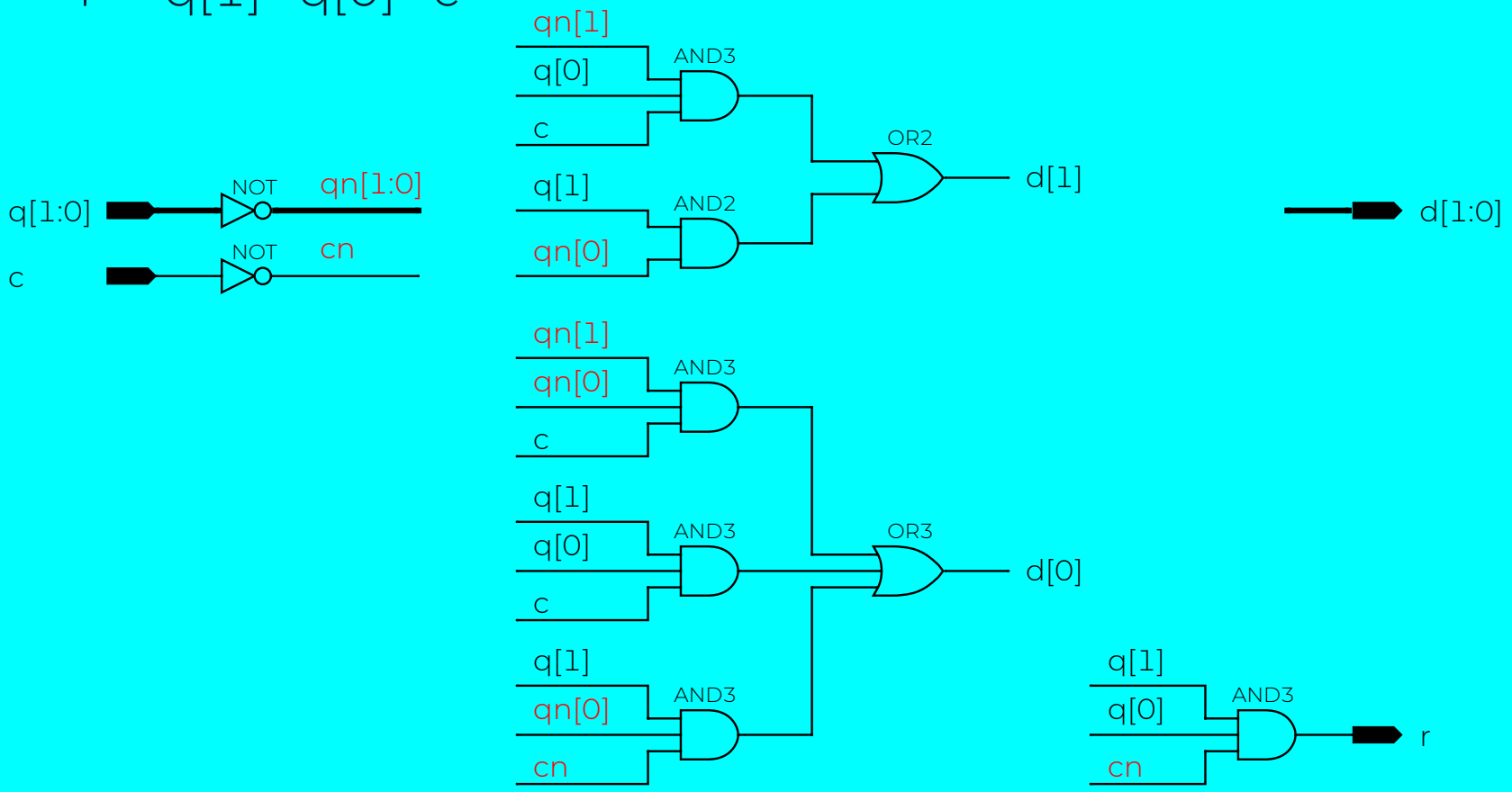
$$r = q[1] \cdot q[0] \cdot \overline{c}$$　　Mealy 型

順序回路の設計（論理式と回路図）

$$d[1] = \overline{q[1]} \cdot q[0] \cdot c + q[1] \cdot \overline{q[0]}$$

$$d[0] = \overline{q[1]} \cdot \overline{q[0]} \cdot c + q[1] \cdot q[0] \cdot c + q[1] \cdot \overline{q[0]} \cdot \overline{c}$$

$$r = q[1] \cdot q[0] \cdot \overline{c}$$



順序回路の設計（回路）

```
‘timescale 1ns/1ns
module detector (clk, clrn, c, r);
    input  clk, clrn, c;
    output r;

    reg  [1:0] q; // current state
    wire [1:0] d; // next state

    // 1. next state ------------------------------------ 1
    assign d[1] = ~q[1] &  q[0] &  c | q[1] & ~q[0];
    assign d[0] = ~q[1] & ~q[0] &  c | q[1] &  q[0] & c |
                   q[1] & ~q[0] & ~c;
    assign r    =  q[1] &  q[0] & ~c;

    // 2. outputs --------------------------------------- 2
    assign r = q[1] & q[0] & ~c; // S3 & ~c

    // 3. dffs ------------------------------------------ 3
    always @(posedge clk or negedge clrn) begin // dffs
        if (clrn == 0) begin
            q <= 0;
        end else begin
            q <= d;
        end
    end
endmodule
```

detector.v

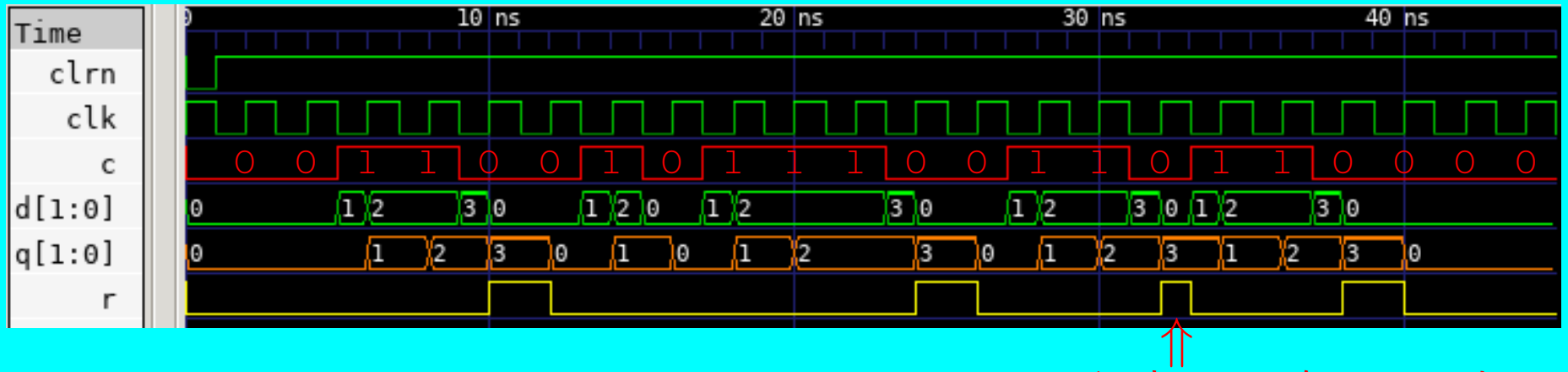
順序回路の設計（テストベンチ）

```
‘timescale 1ns/1ns
module detector_tb;
    reg clk, clrn, a, x;
    wire r;
    detector i0 (clk, clrn, x, r);
    initial begin
        #0  clk = 1; clrn = 0; x = 0; // S0
        #1  clrn = 1;
        #4  x = 1;
        #4  x = 0;
        #4  x = 1;
        #2  x = 0;
        #2  x = 1;
        #6  x = 0;
        #4  x = 1;
        #4  x = 0;
        #2  x = 1;
        #4  x = 0;
        #8 $finish;
    end
    always #1 clk = ~clk;
    initial begin
        $dumpfile ("detector.vcd");
        $dumpvars;
    end
endmodule
```

detector_tb.v

順序回路の設計（波形）

```
% iverilog -Wall -o detector detector_tb.v detector.v
% vvp detector
% gtkwave detector.vcd
```



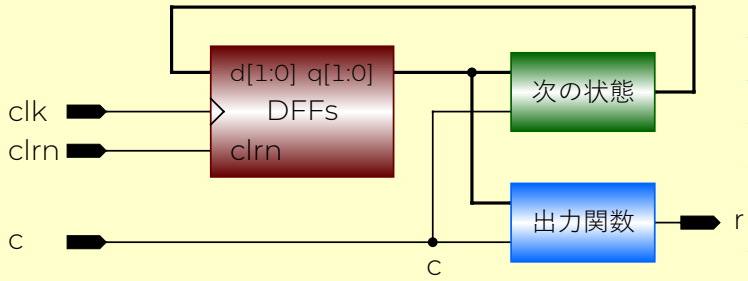
c = 0011001011100110110000 注意: 間違った波形

MAC 対応: ターミナルで gtkwave 起動後 SST の detector_tb の左側にある三角形のボタンを押し、表示された i0 を選択することで機械の中の信号も全て摘出できる。

Windows/ModelSim 対応: add wave -r /*

順序回路の設計（回路）

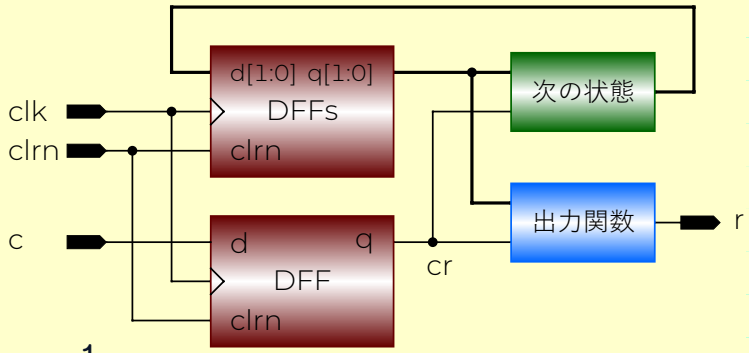
```
‘timescale 1ns/1ns
module detector (clk, clrn, c, r);
    input  clk, clrn, c;
    output r;

    reg  [1:0] q; // current state
    wire [1:0] d; // next state

    // 1. next state ------------------------------------ 1
    assign d[1] = ~q[1] &  q[0] &  c  | q[1] & ~q[0];
    assign d[0] = ~q[1] & ~q[0] &  c  | q[1] &  q[0] & c  |
                   q[1] & ~q[0] & ~c ;
    assign r    =  q[1] &  q[0] & ~c ;

    // 2. outputs --------------------------------------- 2
    assign r = q[1] & q[0] & ~c ; // S3 & ~c

    // 3. dffs ------------------------------------------ 3
    always @(posedge clk or negedge clrn) begin // dffs
        if (clrn == 0) begin
            q <= 0;
        end else begin
            q <= d;
        end
    end
endmodule
```



detector.v

順序回路の設計（入力 c を同期する）

```
‘timescale 1ns/1ns
module det_reg  (clk, clrn, c, r);
    input  clk, clrn, c;
    output r;
    reg    cr;    // registered c
    reg  [1:0] q; // current state
    wire [1:0] d; // next state

    // 1. next state ------------------------------------ 1
    assign d[1] = ~q[1] &  q[0] &  cr | q[1] & ~q[0];
    assign d[0] = ~q[1] & ~q[0] &  cr | q[1] &  q[0] & cr |
                   q[1] & ~q[0] & ~cr;
    assign r    =  q[1] &  q[0] & ~cr;

    // 2. outputs --------------------------------------- 2
    assign r = q[1] & q[0] & ~cr; // S3 & ~cr

    // 3. dffs ------------------------------------------ 3
    always @(posedge clk or negedge clrn) begin // dffs
        if (clrn == 0) begin
            q <= 0; cr <= 0;
        end else begin
            q <= d; cr <= c; // registering c
        end
    end
endmodule
```



det_reg.v

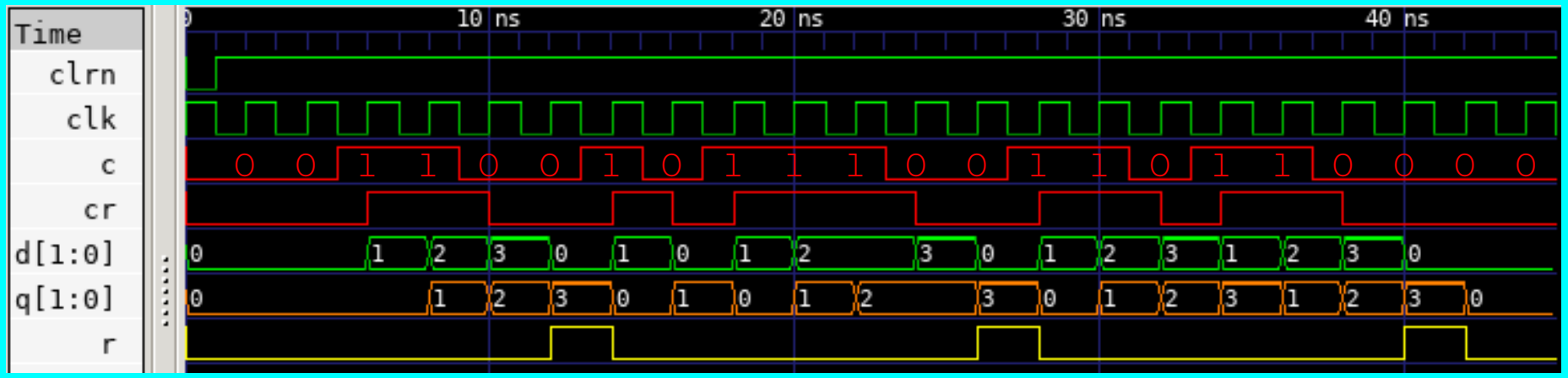
順序回路の設計（入力 c を同期する）

```
‘timescale 1ns/1ns
module det_reg_tb;
    reg clk, clrn, a, x;
    wire r;
    det_reg i0 (clk, clrn, x, r);
    initial begin
        #0  clk = 1; clrn = 0; x = 0; // S0
        #1  clrn = 1;
        #4  x = 1;
        #4  x = 0;
        #4  x = 1;
        #2  x = 0;
        #2  x = 1;
        #6  x = 0;
        #4  x = 1;
        #4  x = 0;
        #2  x = 1;
        #4  x = 0;
        #8 $finish;
    end
    always #1 clk = ~clk;
    initial begin
        $dumpfile ("det_reg.vcd");
        $dumpvars;
    end
endmodule
```

det_reg_tb.v

順序回路の設計（入力 c を同期する）

```
% iverilog -Wall -o det_reg det_reg_tb.v det_reg.v
% vvp det_reg
% gtkwave det_reg.vcd
```



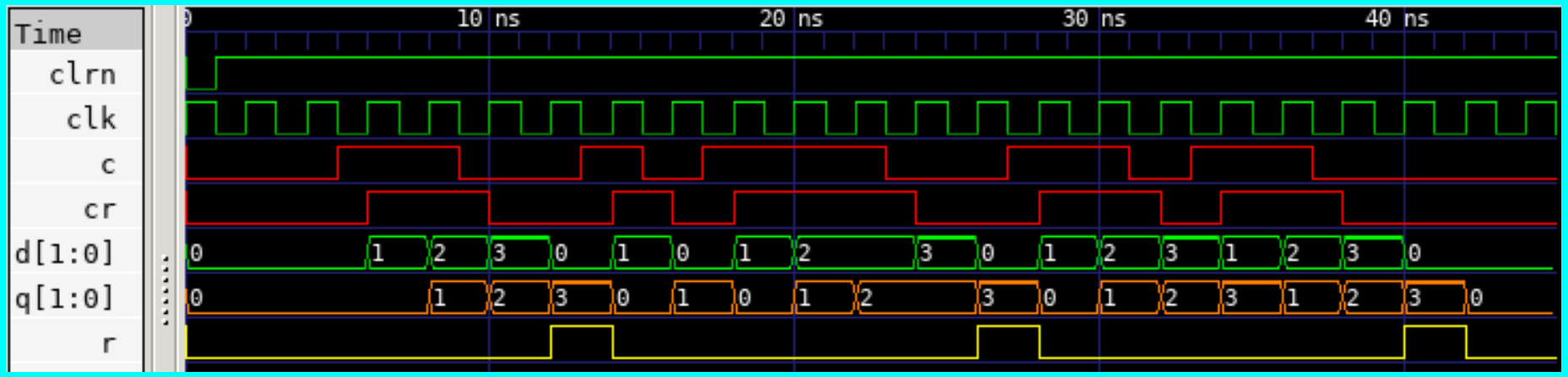
c = 0011001011100110110000

cr：同期された c

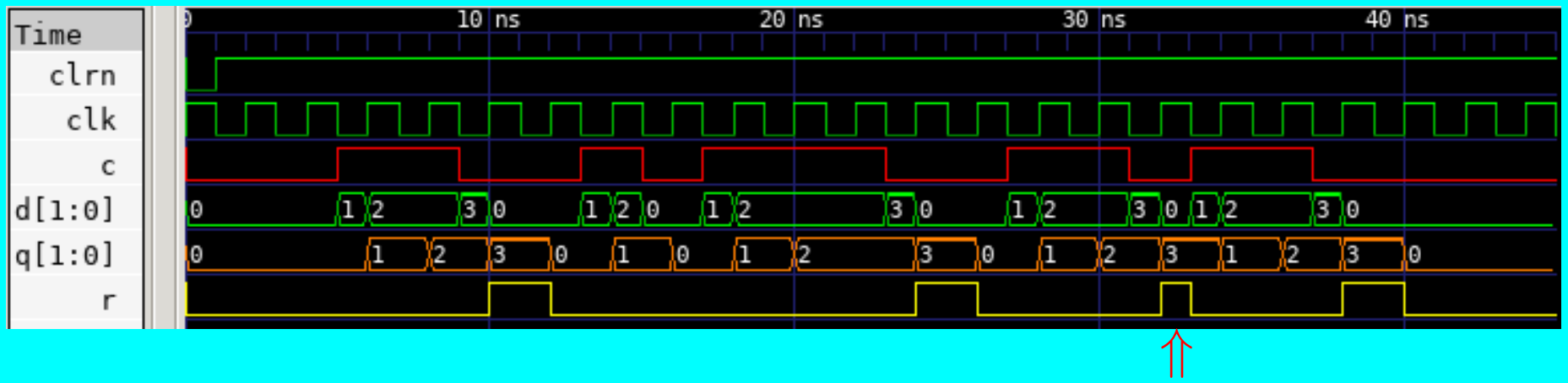
間違った波形が生成されてなかった（入力を同期した）

順序回路の設計（入力 c を同期する）

入力 c 同期波形（cr を使用）



入力 c 非同期波形（c を使用）

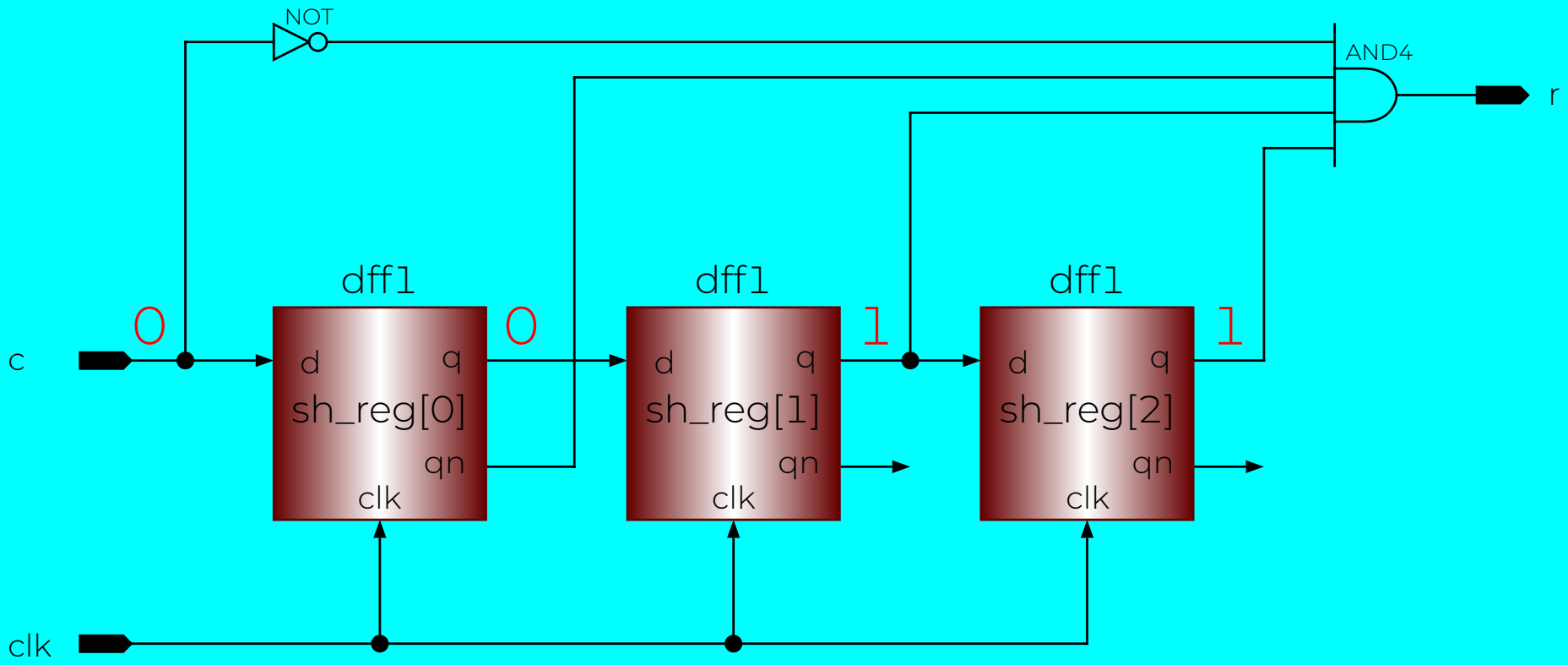


パターン検出の順序回路について

パターン検出は
シフトレジスタで
実現可能である

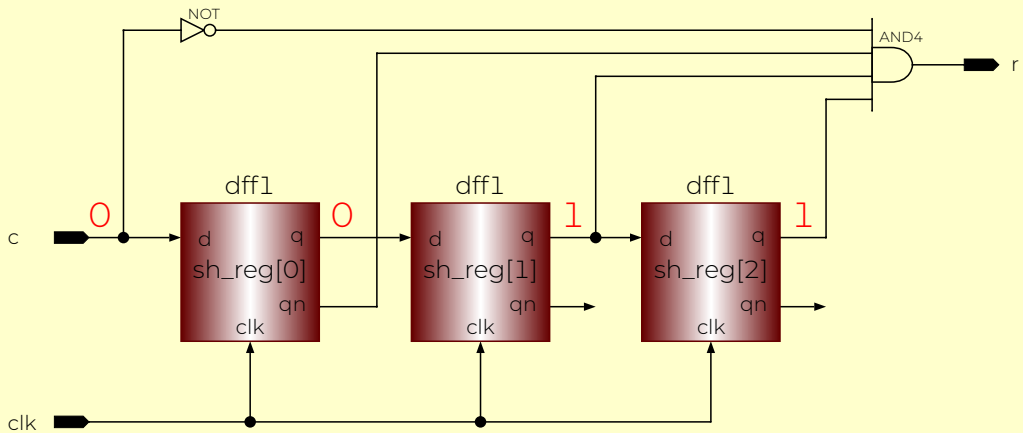
パターン検出の順序回路について

“1100” のパターンを検出する回路



パターン検出の順序回路について

```
‘timescale 1ns/1ns
module det_sh (clk, clrn, c, r);
    input  clk, clrn, c;
    output r;

    reg  [2:0] sh_reg; // 3-bit register

    // dffs
    always @(posedge clk or negedge clrn) begin // dffs
        if (clrn == 0) begin
            sh_reg <= 0;
        end else begin
            // shift 3-bit register
            sh_reg[2] <= sh_reg[1];
            sh_reg[1] <= sh_reg[0];
            sh_reg[0] <= c;
        end
    end

    // outputs
    assign r = sh_reg[2] & // 1
               sh_reg[1] & // 1
              ~sh_reg[0] & // 0
              ~c;          // 0

endmodule
```



det_sh.v

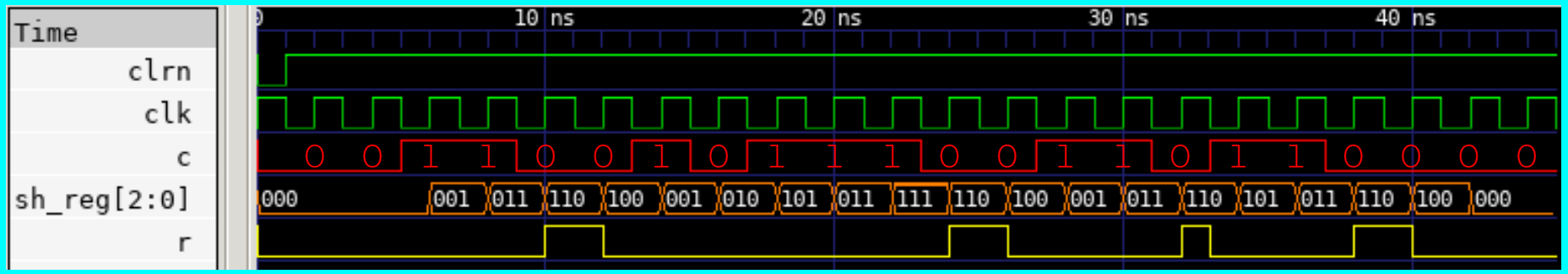
パターン検出の順序回路について

```
‘timescale 1ns/1ns
module det_sh_tb;
    reg clk, clrn, a, x;
    wire r;
    det_sh i0 (clk, clrn, x, r);
    initial begin
        #0  clk = 1; clrn = 0; x = 0; // S0
        #1  clrn = 1;
        #4  x = 1;
        #4  x = 0;
        #4  x = 1;
        #2  x = 0;
        #2  x = 1;
        #6  x = 0;
        #4  x = 1;
        #4  x = 0;
        #2  x = 1;
        #4  x = 0;
        #8 $finish;
    end
    always #1 clk = ~clk;
    initial begin
        $dumpfile ("det_sh.vcd");
        $dumpvars;
    end
endmodule
```

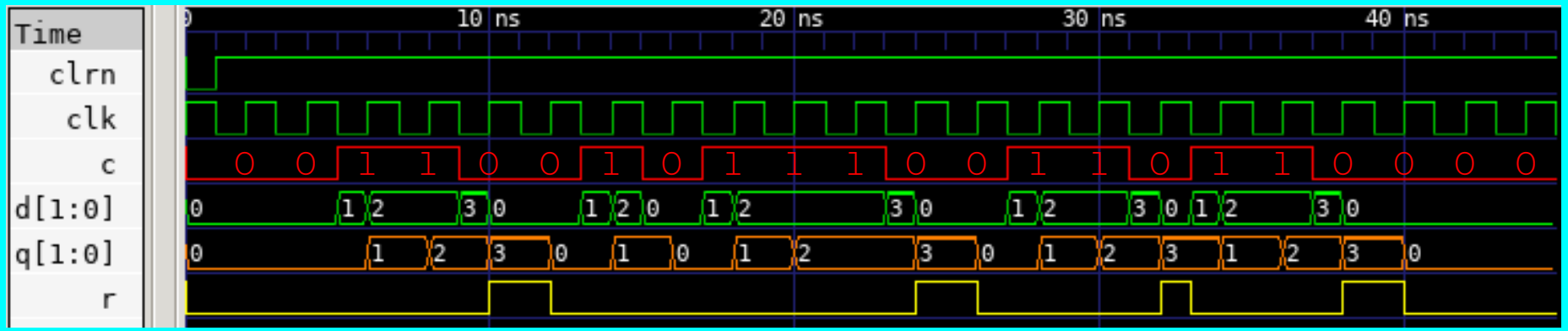
det_sh_tb.v

パターン検出の順序回路について

```
% iverilog -Wall -o det_sh det_sh_tb.v det_sh.v
% vvp det_sh
% gtkwave det_sh.vcd
```

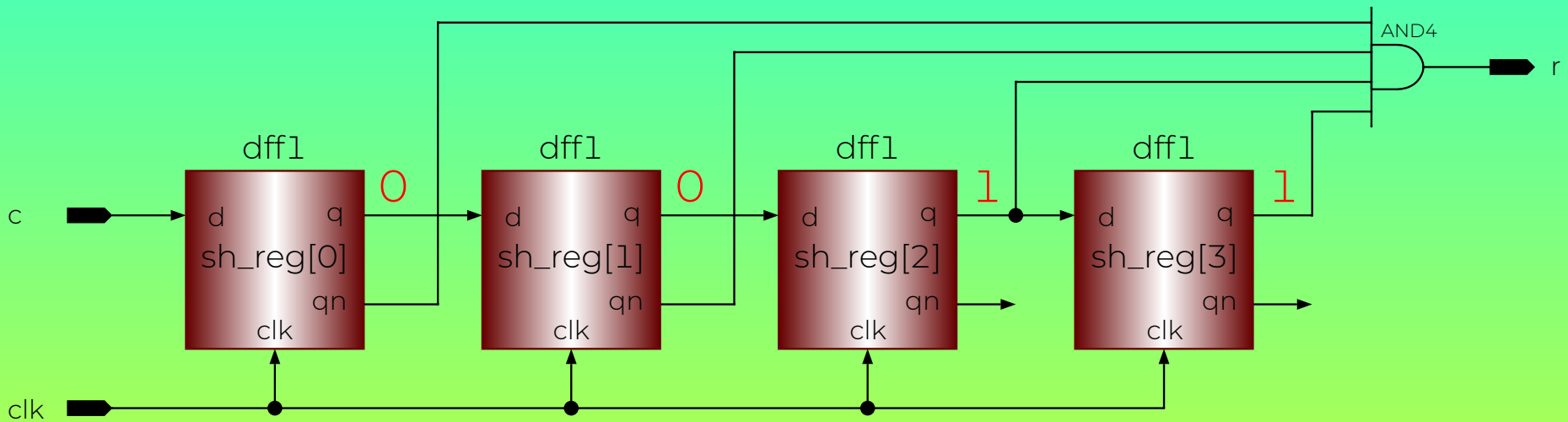


比較：detector 回路の波形



パターン検出の順序回路について

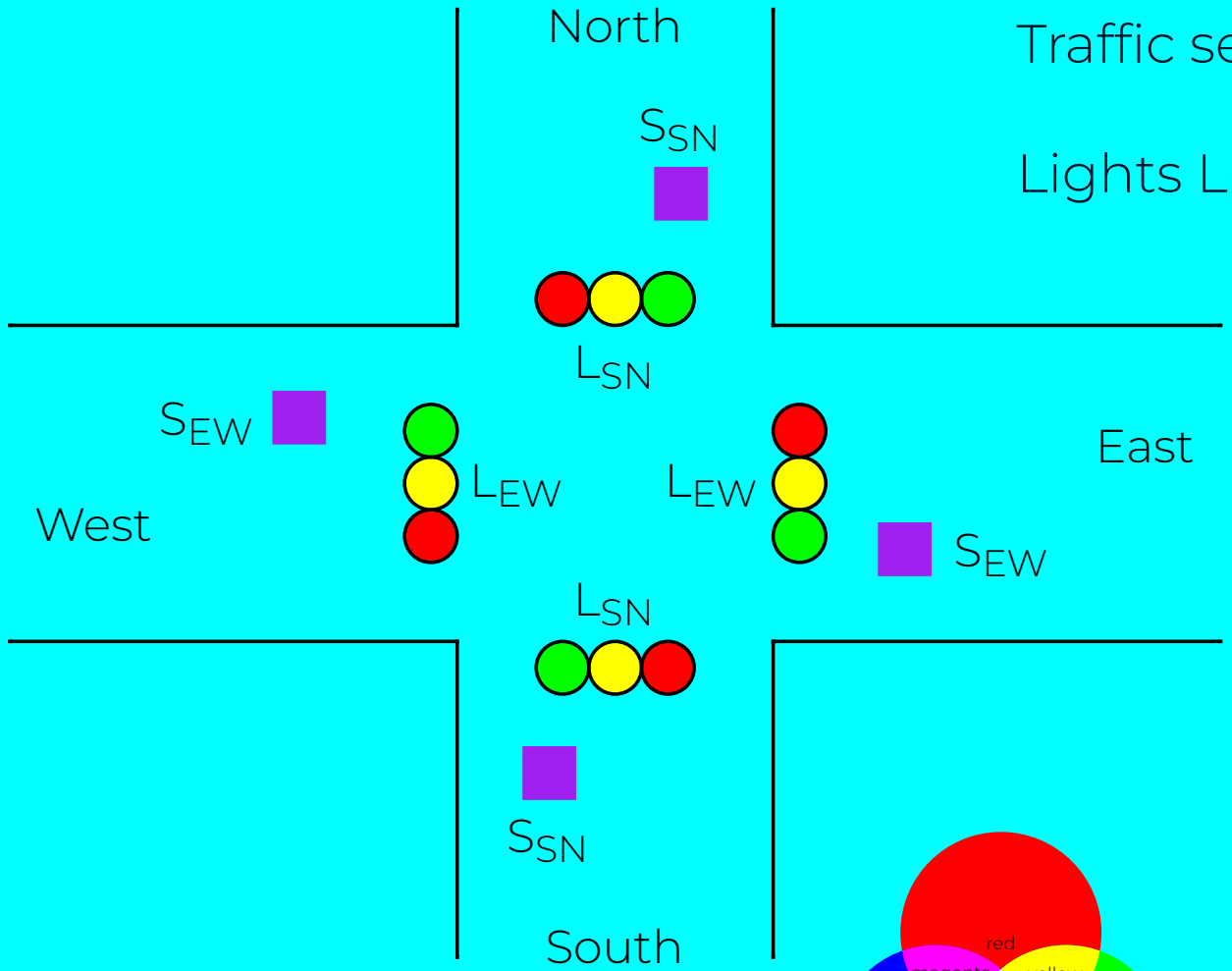
発展：4 ビット DFF を使用して“1100”パターンを検出する回路を設計してください（入力を同期する）。



立ち下がりエッジ検出については、PS2 キーボード インターフェース コントローラー回路 ps2_keyboard.v を参照してください。

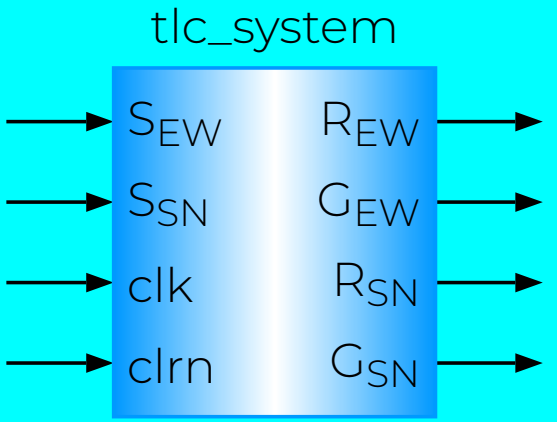
交通信号機
制御システム

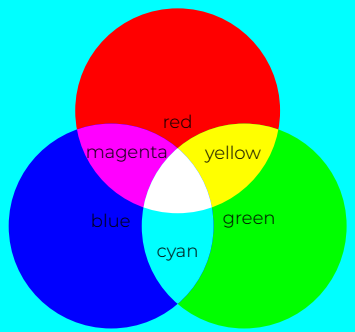
交通信号機制御システム（問題）



Traffic sensors S_{EW} and S_{SN}
(1 if there is traffic)
Lights L_{EW} and L_{SN}

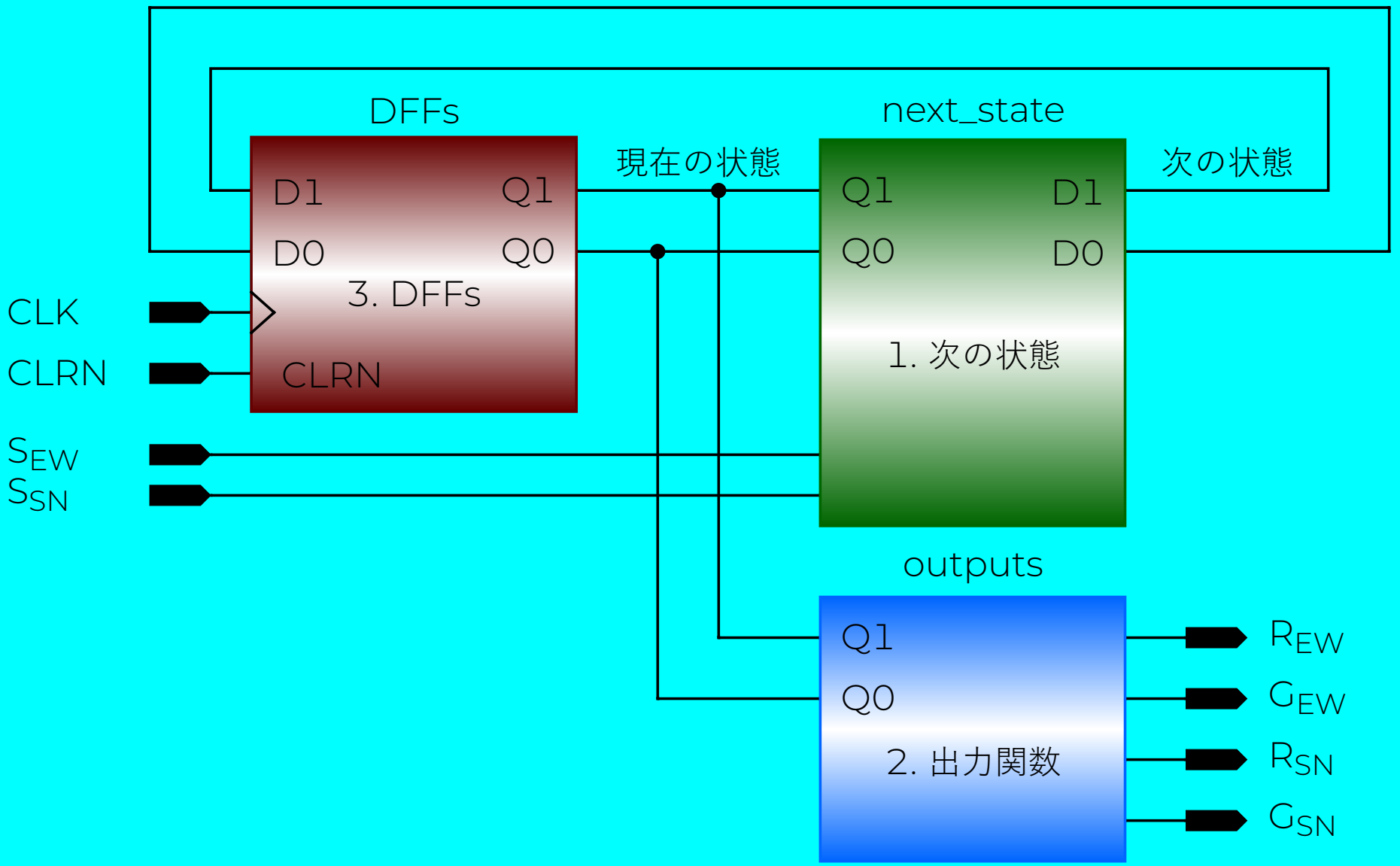
Rxx	Gxx	Color
0	1:	Green
1	0:	Red
1	1:	Yellow



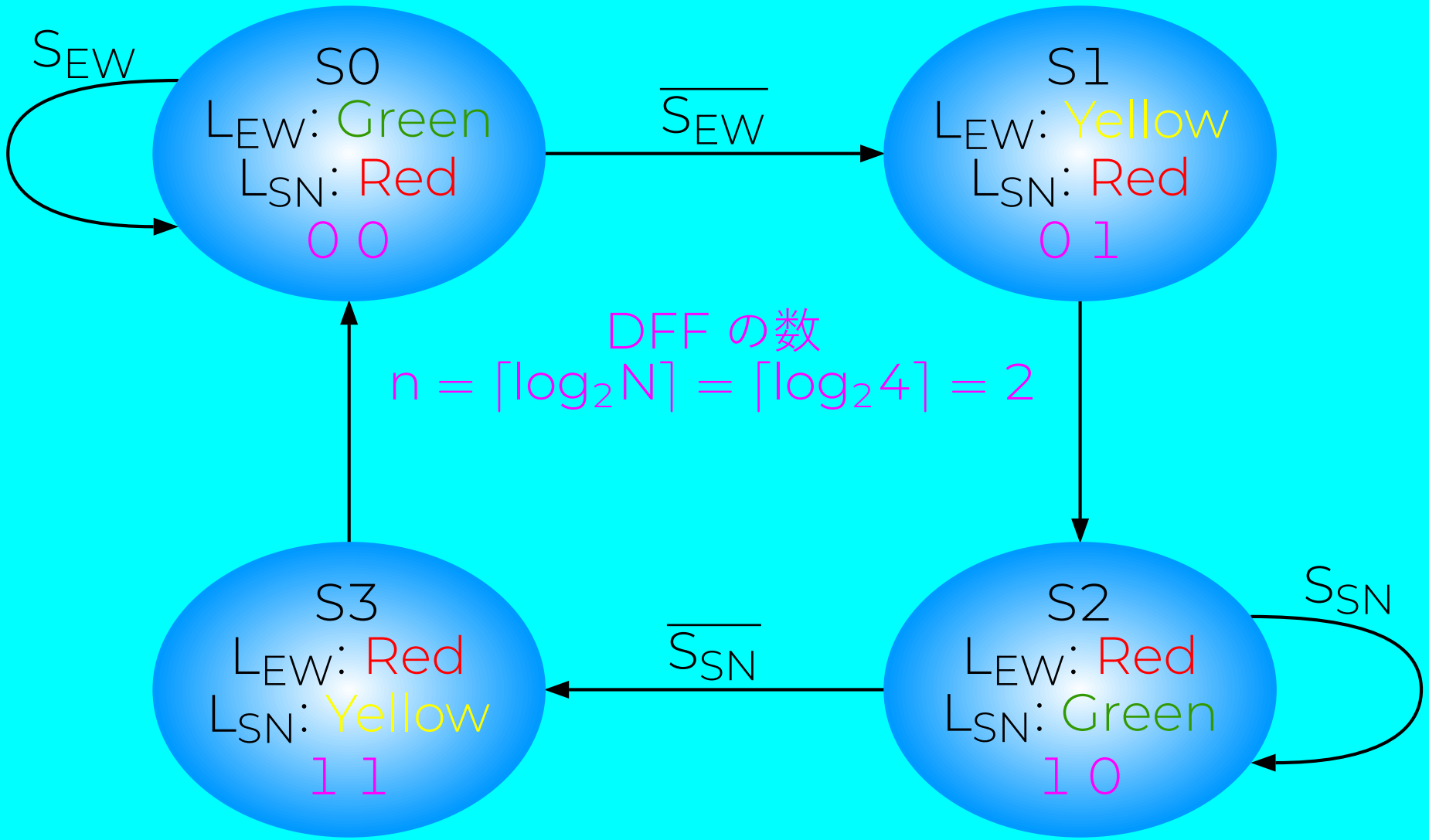


EW: East and West (東西)
SN: South and North (南北)

交通信号機制御（回路）

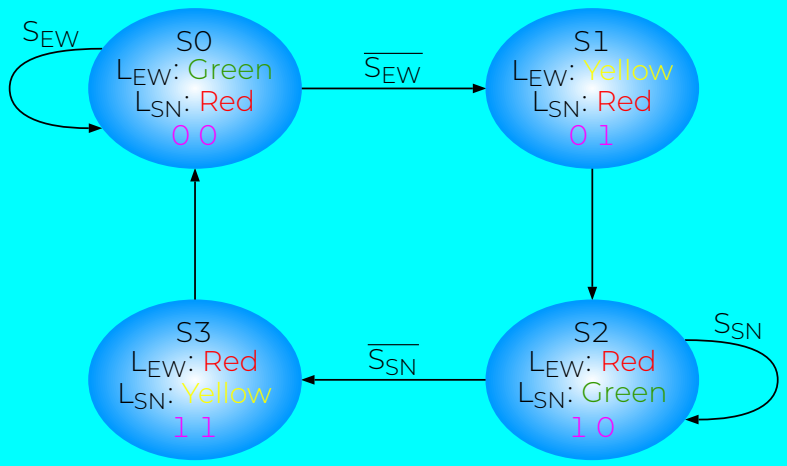

DFFs
D1
Q1
D0
Q0
3. DFFs
CLRN
CLK
CLRN
現在の状態
next_state
Q1
D1
Q0
D0
1. 次の状態
次の状態
SEW
SSN
outputs
Q1
Q0
2. 出力関数
REW
GEW
RSN
GSN

交通信号機制御（状態遷移図）



交通信号機制御（次の状態の真理値表）

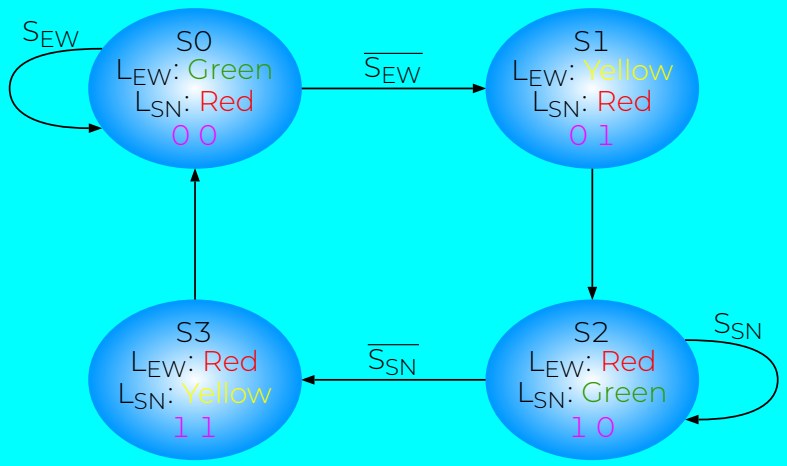
次の状態の真理値表
(状態遷移表)



現在の状態			入力		次の状態	
	Q1	Q0	S_{EW}	S_{SN}	D1	D0
S0	0	0	0	X		

交通信号機制御（次の状態の真理値表）

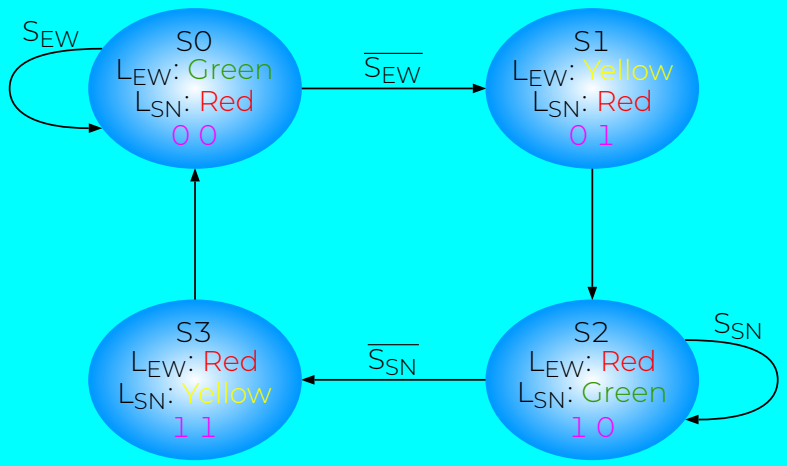
次の状態の真理値表
(状態遷移表)



現在の状態			入力		次の状態		
	Q1	Q0	S_{EW}	S_{SN}		D1	D0
S0	0	0	0	X	S1	0	1
			1	X			

交通信号機制御（次の状態の真理値表）

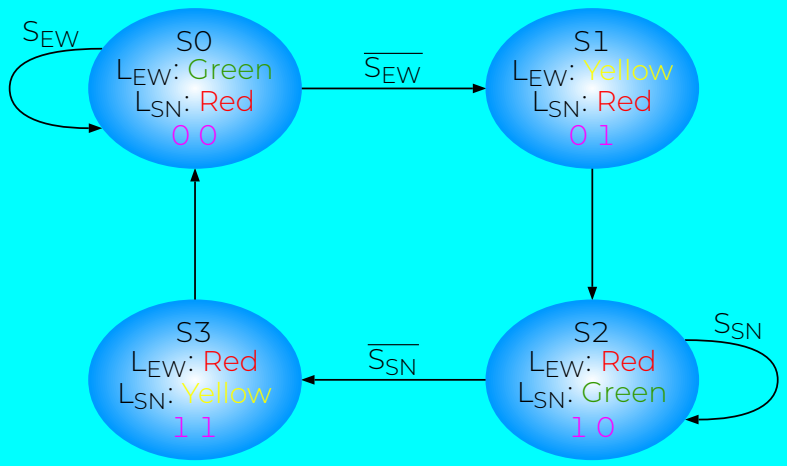
次の状態の真理値表
(状態遷移表)



現在の状態			入力		次の状態		
	Q1	Q0	S_{EW}	S_{SN}		D1	D0
S0	0	0	0	X	S1	0	1
			1	X	S0	0	0
S1	0	1	X	X			

交通信号機制御（次の状態の真理値表）

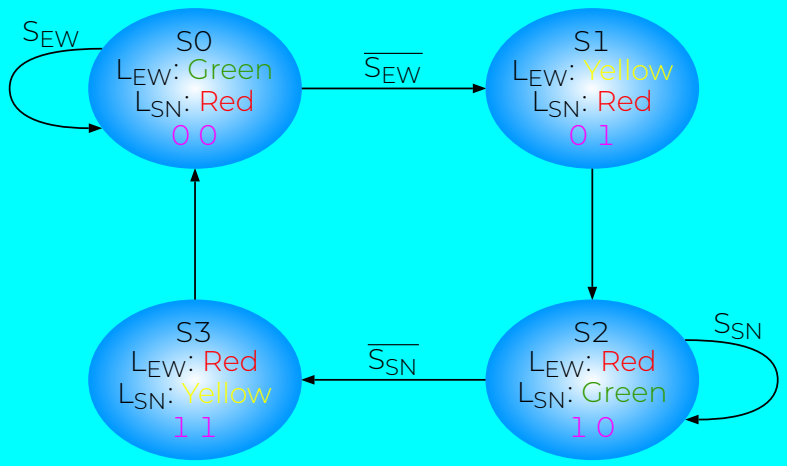
次の状態の真理値表
(状態遷移表)



現在の状態			入力		次の状態		
	Q1	Q0	S_{EW}	S_{SN}		D1	D0
S0	0	0	0	X	S1	0	1
			1	X	S0	0	0
S1	0	1	X	X	S2	1	0
S2	1	0	X	0			

交通信号機制御（次の状態の真理値表）

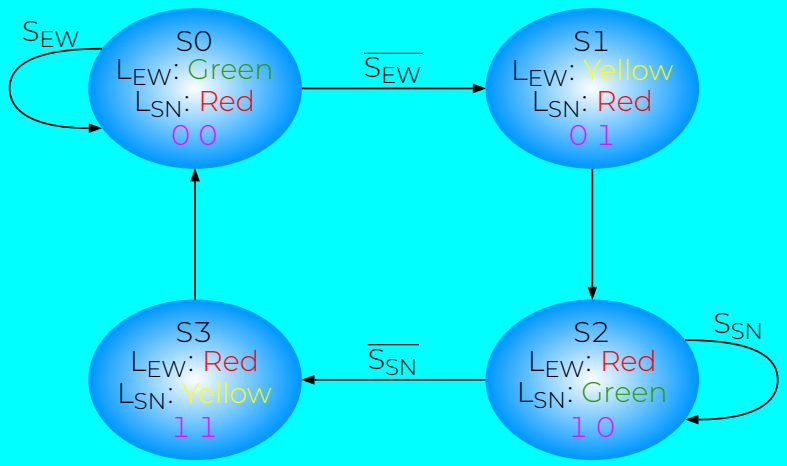
次の状態の真理値表
(状態遷移表)



現在の状態			入力		次の状態		
	Q1	Q0	S_{EW}	S_{SN}		D1	D0
S0	0	0	0	X	S1	0	1
			1	X	S0	0	0
S1	0	1	X	X	S2	1	0
S2	1	0	X	0	S3	1	1
			X	1			

交通信号機制御（次の状態の真理値表）

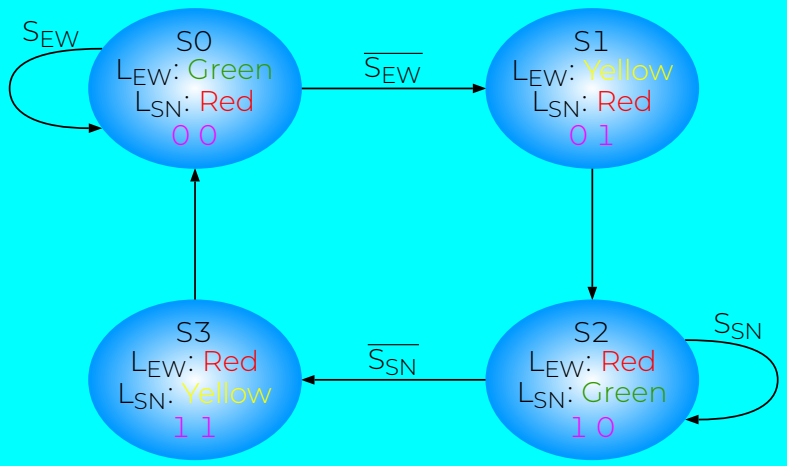
次の状態の真理値表 (状態遷移表)



現在の状態			入力		次の状態		
	Q1	Q0	S_{EW}	S_{SN}		D1	D0
S0	0	0	0	X	S1	0	1
			1	X	S0	0	0
S1	0	1	X	X	S2	1	0
S2	1	0	X	0	S3	1	1
			X	1	S2	1	0
S3	1	1	X	X			

交通信号機制御（次の状態の真理値表）

次の状態の真理値表
(状態遷移表)



現在の状態			入力		次の状態		
	Q1	Q0	S_{EW}	S_{SN}		D1	D0
S0	0	0	0	X	S1	0	1
			1	X	S0	0	0
S1	0	1	X	X	S2	1	0
S2	1	0	X	0	S3	1	1
			X	1	S2	1	0
S3	1	1	X	X	S0	0	0

交通信号機制御（次の状態の論理式）

現在の状態			入力		次の状態		
	Q1	Q0	S_{EW}	S_{SN}		D1	D0
S0	0	0	0	X	S1	0	1
			1	X	S0	0	0
S1	0	1	X	X	S2	1	0
S2	1	0	X	0	S3	1	1
			X	1	S2	1	0
S3	1	1	X	X	S0	0	0

次の状態の論理式

$$D1 = \overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0}$$

$$D0 = \overline{Q1} \cdot \overline{Q0} \cdot \overline{S_{EW}} + Q1 \cdot \overline{Q0} \cdot \overline{S_{SN}}$$

D1

Q1 Q0	0 0	0 1	1 1	1 0
S_{EW} S_{SN} 0 0		1		1
0 1		1		1
1 1		1		1
1 0		1		1

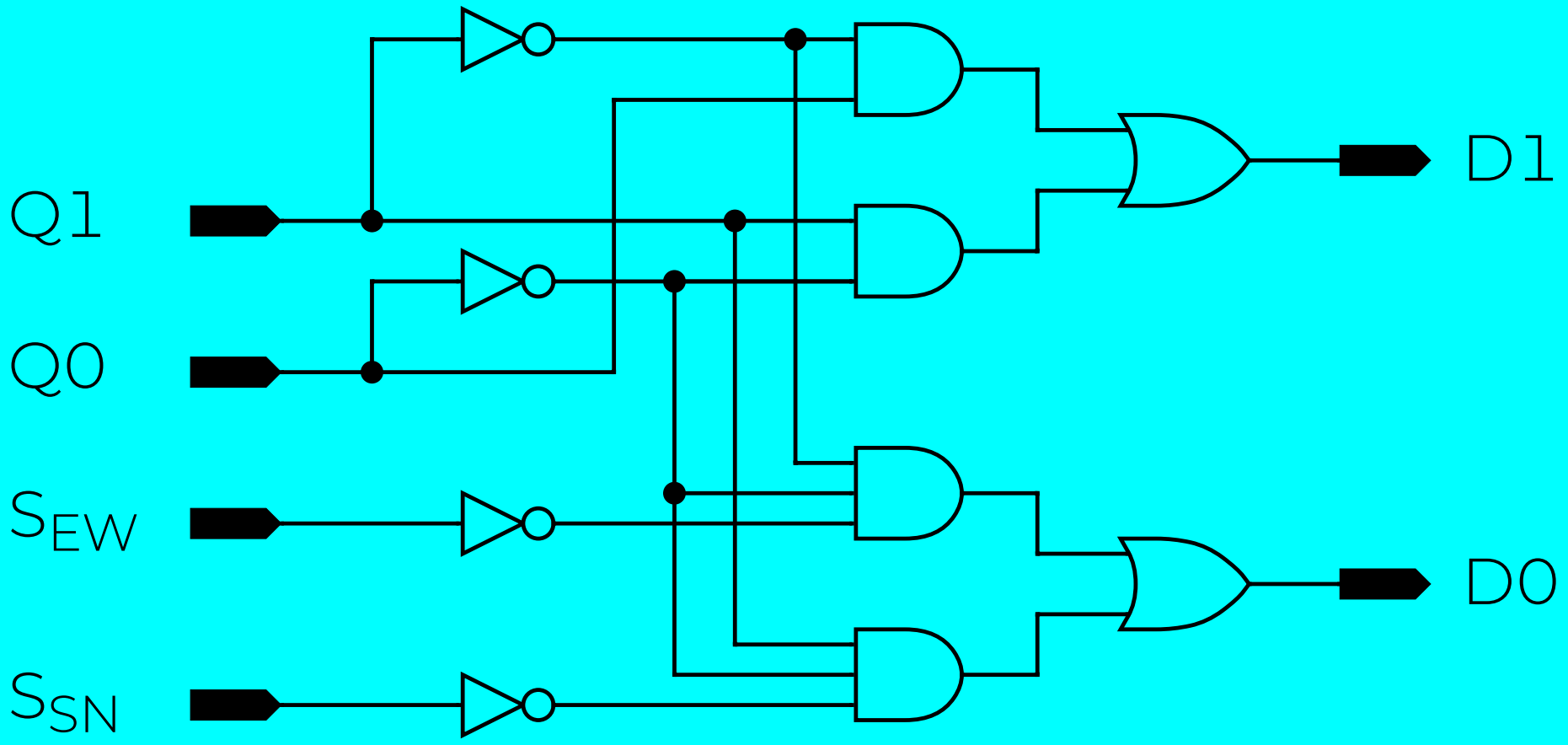
D0

Q1 Q0	0 0	0 1	1 1	1 0
S_{EW} S_{SN} 0 0	1			1
0 1	1			
1 1				
1 0				1

交通信号機制御（次の状態の回路）

$$D1 = \overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0}$$

$$D0 = \overline{Q1} \cdot \overline{Q0} \cdot \overline{S_{EW}} + Q1 \cdot \overline{Q0} \cdot \overline{S_{SN}}$$



交通信号機制御（出力関数の真理値表）

出力関数の真理値表

S0: L_{EW}: Green; L_{SN}: Red
S1: L_{EW}: Yellow; L_{SN}: Red
S2: L_{EW}: Red; L_{SN}: Green
S3: L_{EW}: Red; L_{SN}: Yellow

Color encoding:

R	G	Color
0	1	Green
1	0	Red
1	1	Yellow

現在の状態			出力			
	Q1	Q0	R_{EW}	G_{EW}	R_{SN}	G_{SN}
S0	0	0	0	1	1	0
S1	0	1	1	1	1	0
S2	1	0	1	0	0	1
S3	1	1	1	0	1	1

交通信号機制御（出力関数の論理式）

現在の状態			出力			
	Q1	Q0	R_{EW}	G_{EW}	R_{SN}	G_{SN}
S0	0	0	0	1	1	0
S1	0	1	1	1	1	0
S2	1	0	1	0	0	1
S3	1	1	1	0	1	1

出力関数の論理式

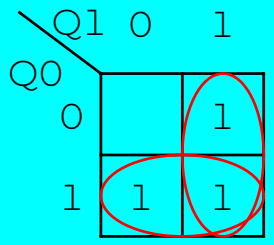
$$R_{EW} = Q1 + Q0$$

$$G_{EW} = \overline{Q1}$$

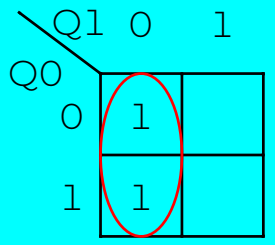
$$R_{SN} = \overline{Q1} + Q0$$

$$G_{SN} = Q1$$

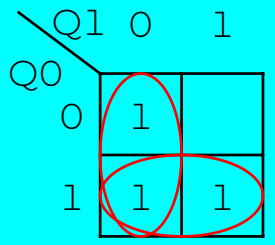
R_{EW}



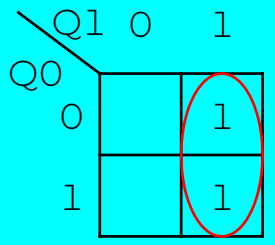
G_{EW}



R_{SN}



G_{SN}



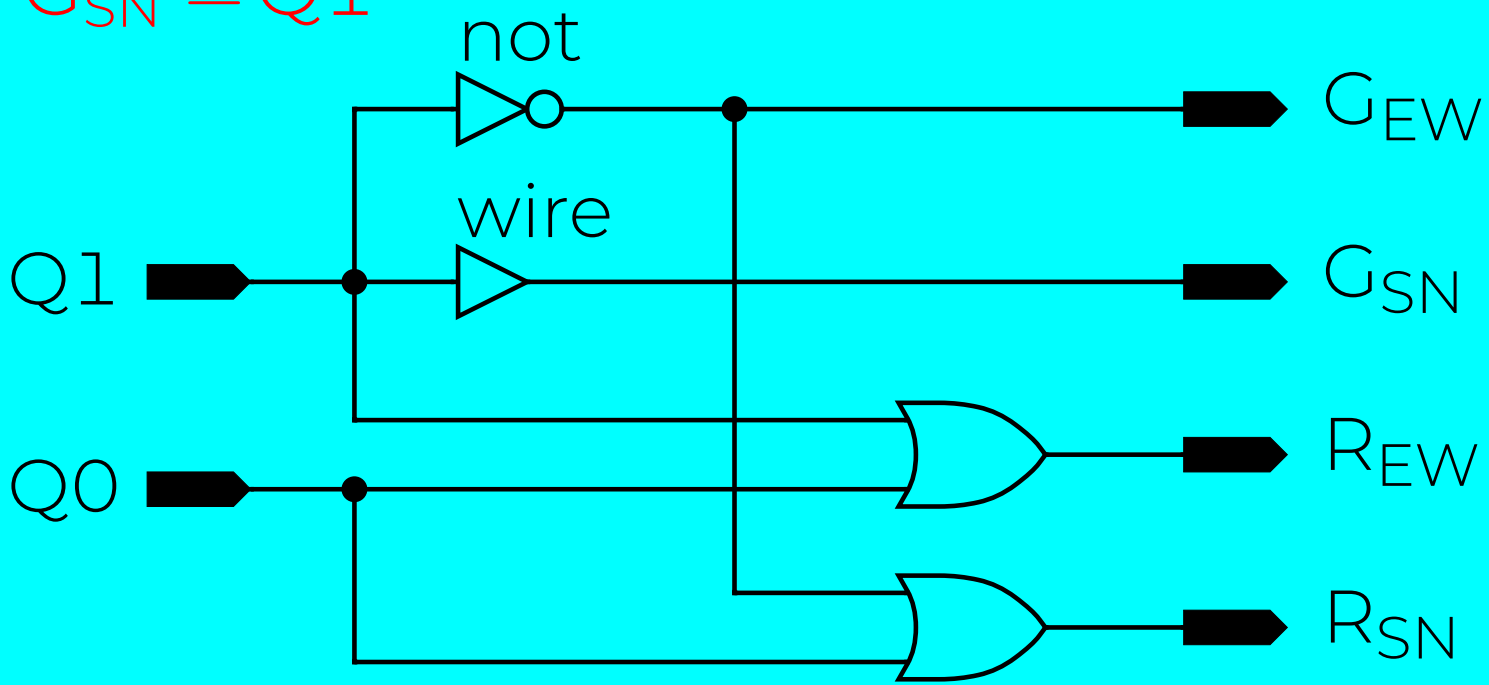
交通信号機制御（出力関数の回路）

$$R_{EW} = Q1 + Q0$$

$$G_{EW} = \overline{Q1}$$

$$R_{SN} = \overline{Q1} + Q0$$

$$G_{SN} = Q1$$



交通信号機制御（回路）

```
‘timescale 1ns/1ns
module tlc_system (CLK, CLRN, SEW, SSN, REW, GEW, RSN, GSN);
    input  CLK, CLRN, SEW, SSN;
    output REW, GEW, RSN, GSN; // outputs

    reg    [1:0] Q; // current state
    wire   [1:0] D; // next state
    // 1. next state ------------------------------------ 1
    assign D[1] = ~Q[1] &  Q[0] | Q[1] & ~Q[0];
    assign D[0] = ~Q[1] & ~Q[0] & ~SEW | Q[1] & ~Q[0] & ~SSN;
    // 2. outputs --------------------------------------- 2
    assign REW =  Q[1] | Q[0];
    assign GEW = ~Q[1];
    assign RSN = ~Q[1] | Q[0];
    assign GSN =  Q[1];
    // 3. DFFs ------------------------------------------ 3
    always @(posedge CLK or negedge CLRN) begin // DFFs
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            Q <= D;
        end
    end
endmodule
```

$$D1 = \overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0}$$
$$D0 = \overline{Q1} \cdot \overline{Q0} \cdot S_{EW} + Q1 \cdot \overline{Q0} \cdot \overline{S_{SN}}$$

$$R_{EW} = Q1 + Q0$$
$$G_{EW} = \overline{Q1}$$
$$R_{SN} = \overline{Q1} + Q0$$
$$G_{SN} = Q1$$

tlc_system.v

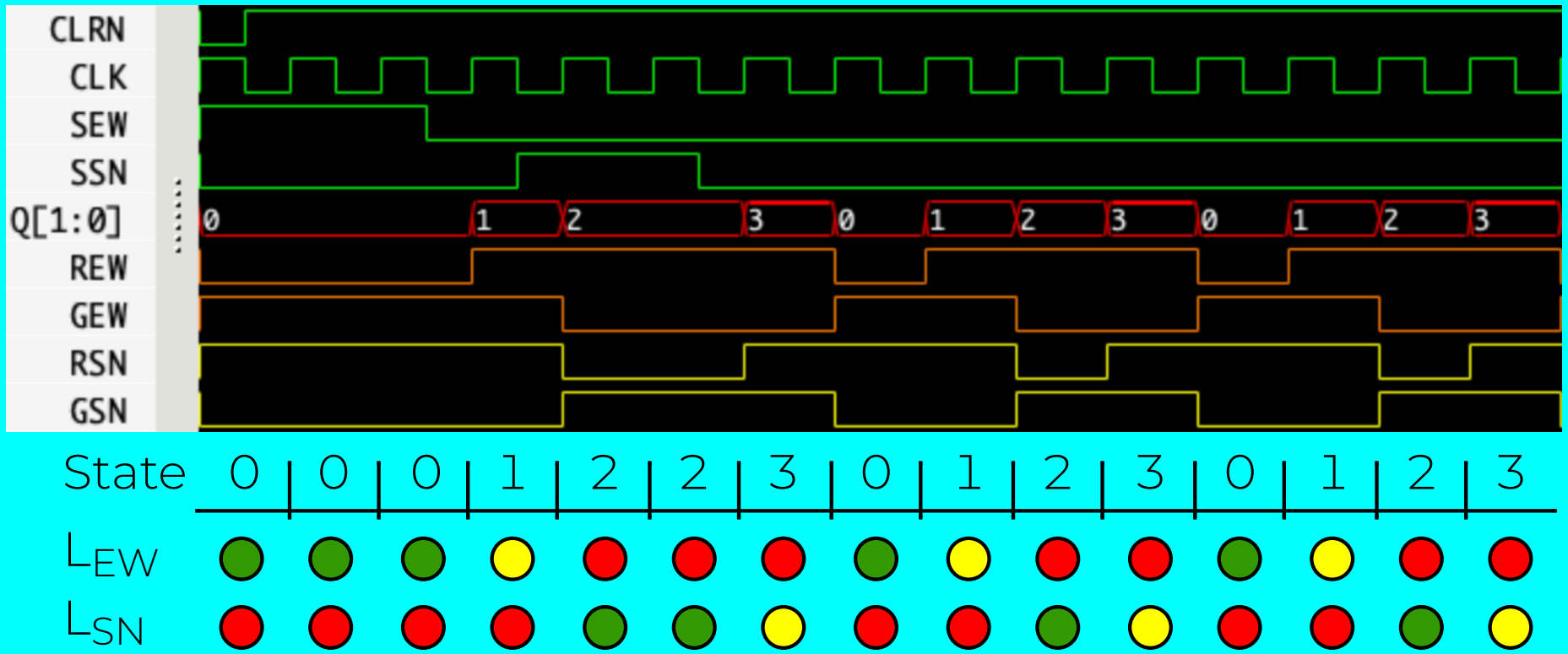
交通信号機制御（テストベンチ）

```
‘timescale 1ns/1ns
module tlc_system_tb;
    reg  CLK, CLRN, SEW, SSN;
    wire REW, GEW, RSN, GSN;

    tlc_system i0 (CLK, CLRN, SEW, SSN, REW, GEW, RSN, GSN);

    initial begin
        #0 CLK  = 1; CLRN = 0; SEW = 1; SSN = 0;
        #1 CLRN = 1;
        #4 SEW  = 0;
        #2 SSN  = 1;
        #4 SSN  = 0;
        #22 $finish;
    end

    always #1 CLK = ~CLK;

    initial begin
        $dumpfile ("tlc_system.vcd");
        $dumpvars;
    end

endmodule
```

tlc_system_tb.v

交通信号機制御（波形）

```
% iverilog -Wall -o tlc_system tlc_system_tb.v tlc_system.v
% vvp tlc_system
% gtkwave tlc_system.vcd
```

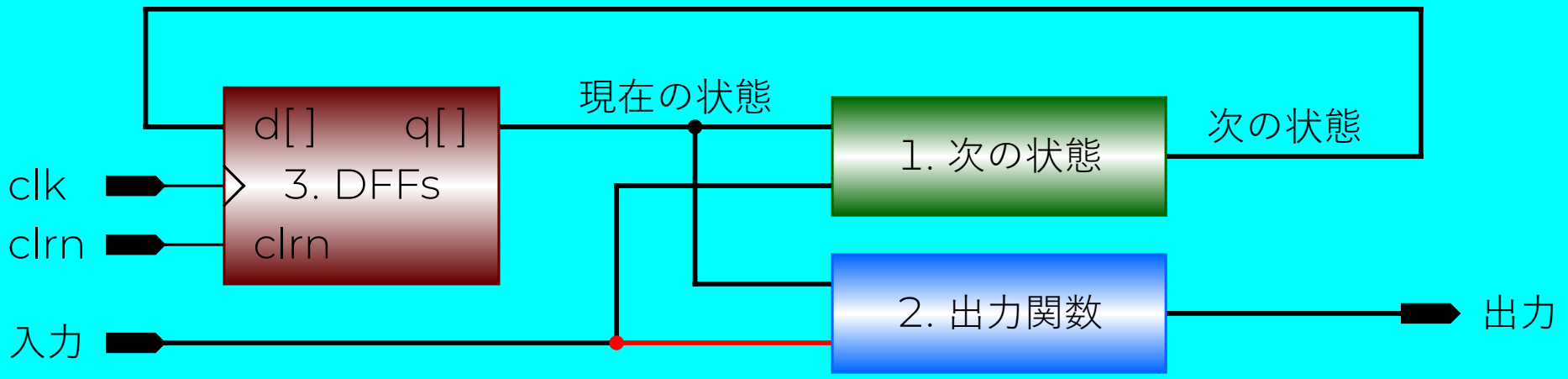


この回路では、どの方向からも車が来ないとき、信号機の色が変わり続けたり、一方から車が来続けると、たとえもう一方に多くの車があってもずっと信号が変わらなかったりするという問題があった。

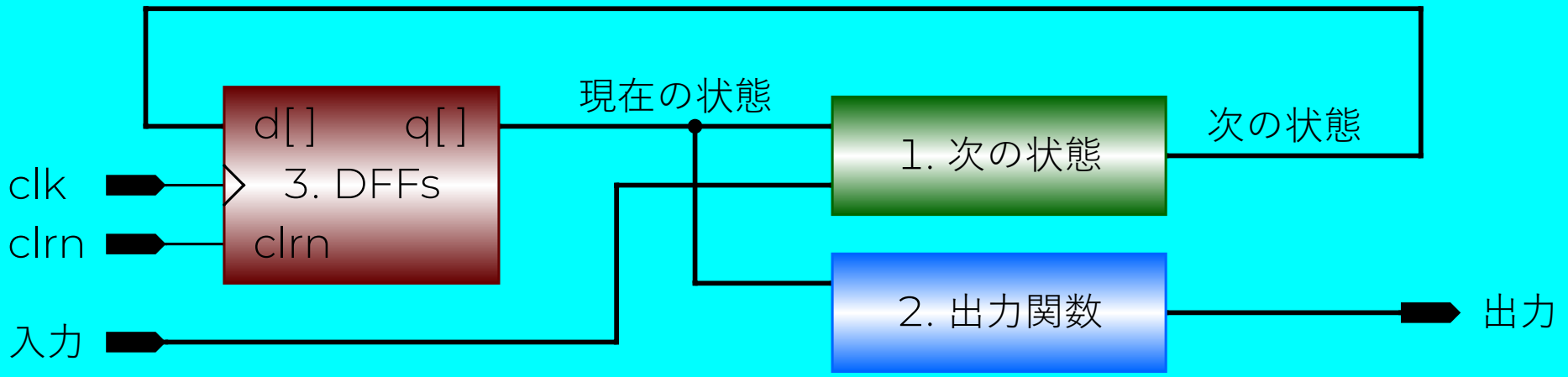
順序回路設計の手順

1. 問題を理解する。
2. 状態遷移図をつくる。
3. FF の数を決める ($n = \lceil \log_2 N \rceil$、N は状態の数)。
4. 各状態に n ビットの番号を付け、真理値表をつくる。
   - 次の状態の真理値表をつくる。
   - 出力関数の真理値表をつくる。
5. 真理値表から論理式をつくる (どの条件で出力が 1 になるか)。
   - カルノー図を用いて論理式を簡単化する。
6. 論理式から回路をつくる。
7. テストベンチをつくる (シミュレーションするため)。
8. 回路をシミュレーションする (回路設計の正当性検証)。
9. 与えられた回路の動作を理解する (波形の説明)。

順序回路 — Mealy 型と Moore 型



(a) Mealy 型



(b) Moore 型

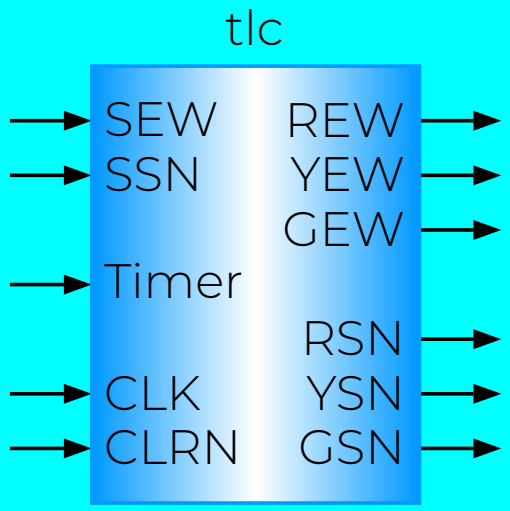
順序回路のまとめ

まとめ

- 順序回路の基本構造
- 順序回路の分析
- Mealy 型と Moore 型順序回路
- 順序回路設計の手順
- 状態遷移図
- 次の状態の真理値表、論理式、および回路
- 出力関数の真理値表、論理式、および回路
- 交通信号機制御システム

課題 XII (200 点)

交通信号機制御システムを設計し動作検証シミュレーションして下さい。



S: センサー
EW: 東西
SN: 南北

R: 赤信号 (1: 点灯; 0: 消灯)
Y: 黄信号 (1: 点灯; 0: 消灯)
G: 緑信号 (1: 点灯; 0: 消灯)

Timer = 1: SEW = SSN = 1 (東西と南北車がある)
⇒ 状態遷移を有効にする。

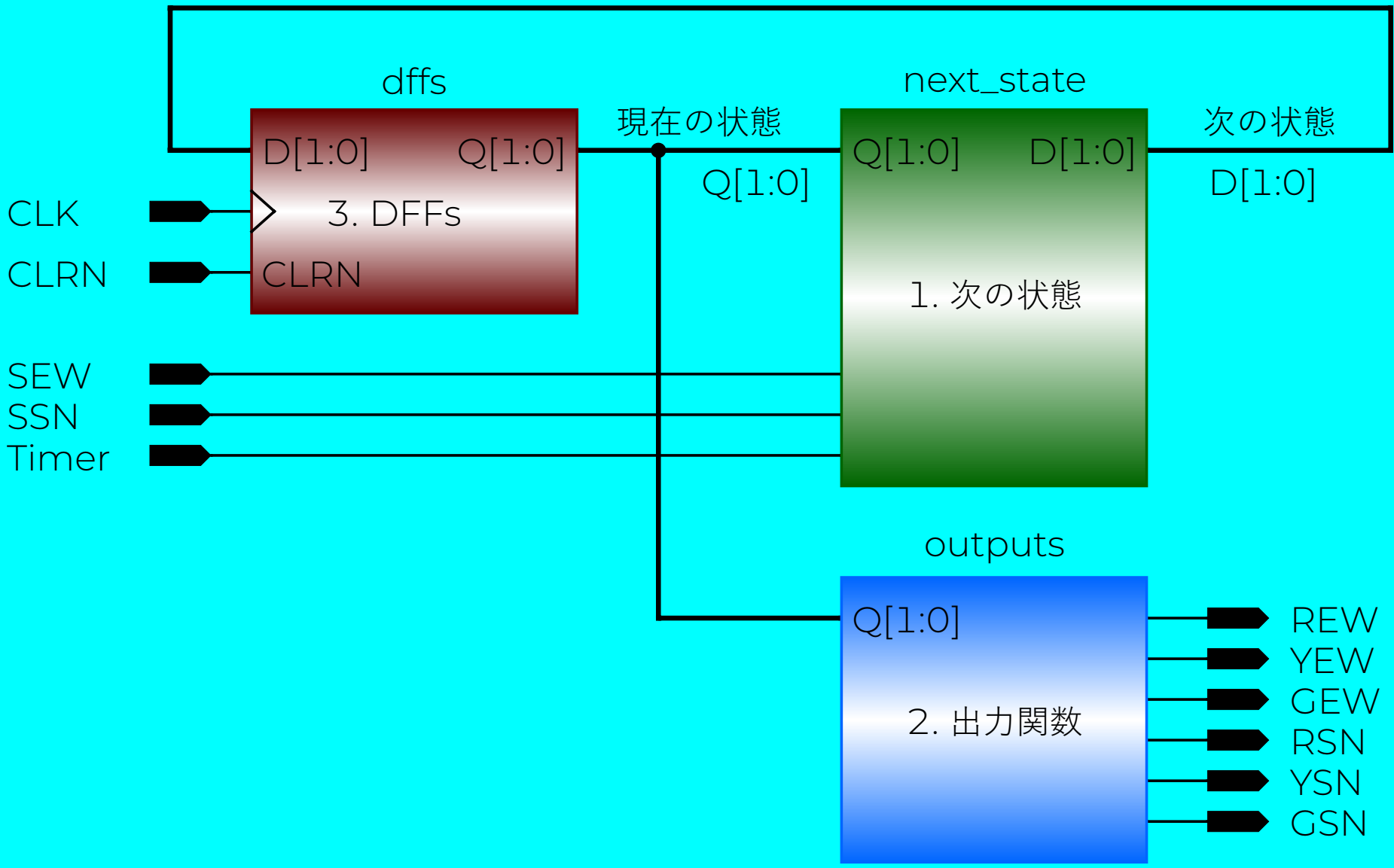
Timer = 0: SEW = SSN = 1 (東西と南北車がある)
⇒ 状態遷移を無効にする (維持する)。

状態の割り当ては P46 の波形を参照してください。注意事項:

(1) もし SEW = SSN = 0, 状態変化なし (Timer: ドントケア)
(2) もし SEW ≠ SSN, 状態遷移を有効にする (Timer: ドントケア)
(3) もし SEW = SSN = 1 かつ Timer = 1, 状態遷移を有効にする
(4) もし SEW = SSN = 1 かつ Timer = 0, 状態遷移を無効にする

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

課題 XII (200 点)


dffs
D[1:0]
Q[1:0]
3. DFFs
CLRN
CLK
CLRN
現在の状態
Q[1:0]
next_state
Q[1:0]
D[1:0]
1. 次の状態
次の状態
D[1:0]
SEW
SSN
Timer
outputs
Q[1:0]
2. 出力関数
REW
YEW
GEW
RSN
YSN
GSN

課題 XII (200 点)

レポートの必須項目:

(1) 状態遷移図

(2) 次の状態 next_state の真理値表

(3) 次の状態 next_state 各出力信号のカルノー図

(4) 次の状態 next_state の論理式

D[1] =

D[0] =

(5) 次の状態 next_state の回路図

課題 XII (200 点)

(6) 出力関数 outputs の真理値表

(7) 出力関数 outputs 各出力信号のカルノー図

(8) 出力関数 outputs の論理式

REW =

YEW =

GEW =

RSN =

YSN =

GSN =

(9) 出力関数 outputs の回路図

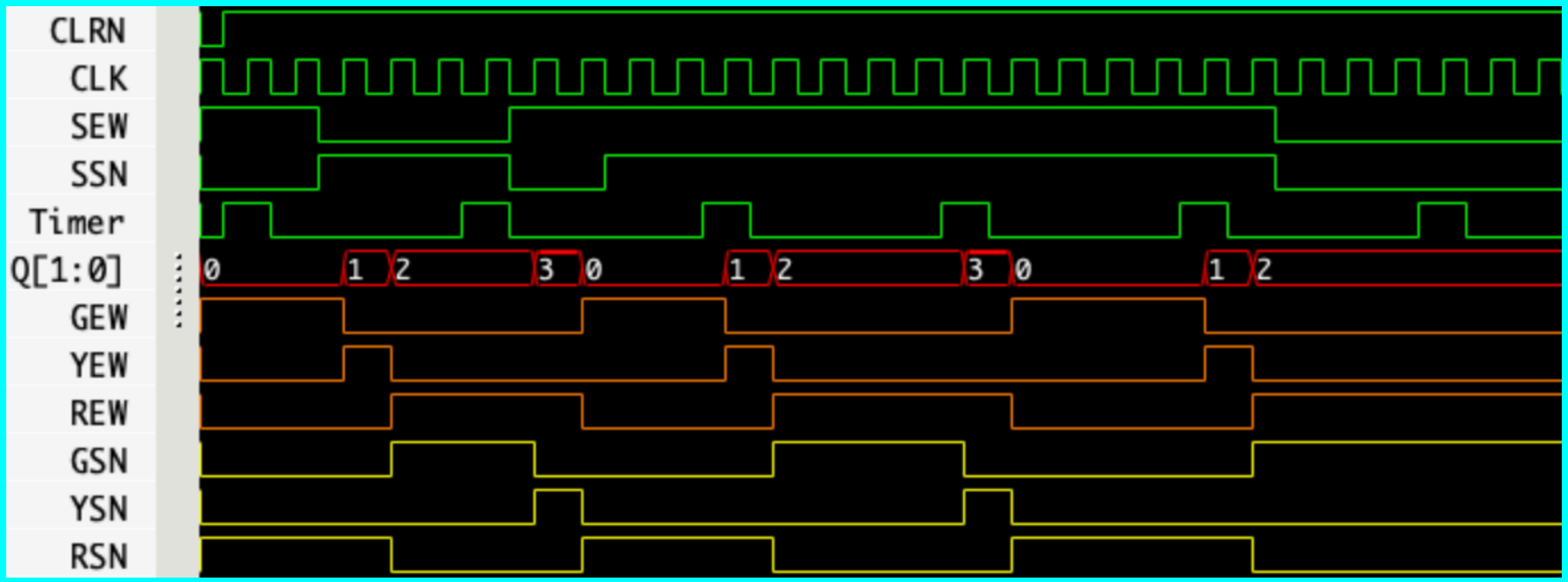
課題 XII (200 点)

```
‘timescale 1ns/1ns
module tlc (CLK,CLRN,SEW,SSN,Timer,REW,YEW,GEW,RSN,YSN,GSN);
    input  CLK,CLRN,SEW,SSN,Timer;
    output REW,YEW,GEW,RSN,YSN,GSN;
    reg    [1:0] Q; // current state
    wire   [1:0] D; // next state
    // 1. next state ------------------------------------ 1
    assign D[1] = ;
    assign D[0] = ;
    // 2. outputs --------------------------------------- 2
    assign REW = ;
    assign YEW = ;
    assign GEW = ;
    assign RSN = ;
    assign YSN = ;
    assign GSN = ;
    // 3. DFFs ------------------------------------------ 3
    always @(posedge CLK or negedge CLRN) begin // DFFs
        if (CLRN == 0) begin
            Q <= 0;
        end else begin
            Q <= D;
        end
    end
endmodule
```

tlc.v

課題 XII (200 点)

```
% iverilog -Wall -o tlc tlc_tb.v tlc.v
% vvp tlc
% gtkwave tlc.vcd
```



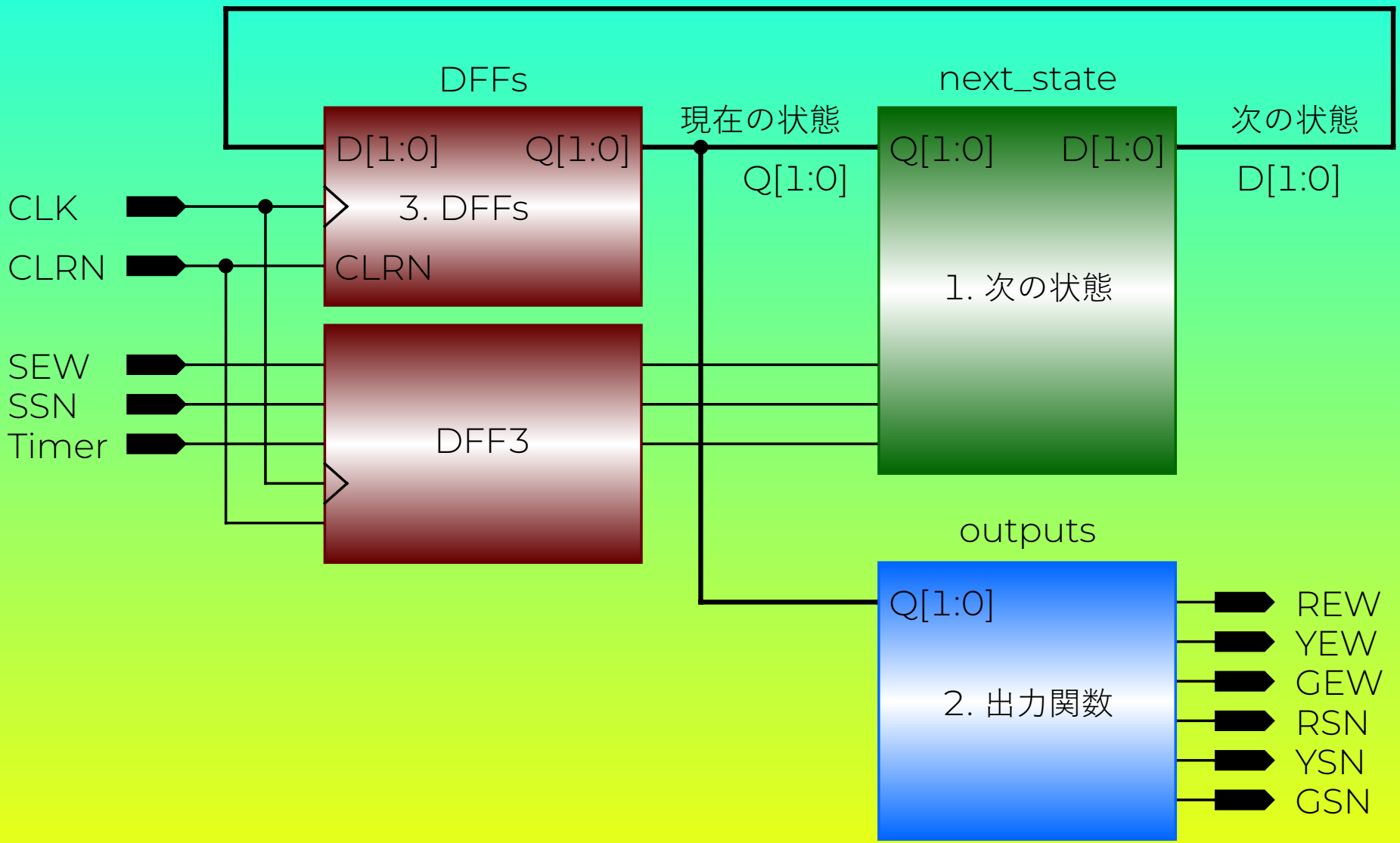
波形を説明してください。

Timer = 0, 維持する

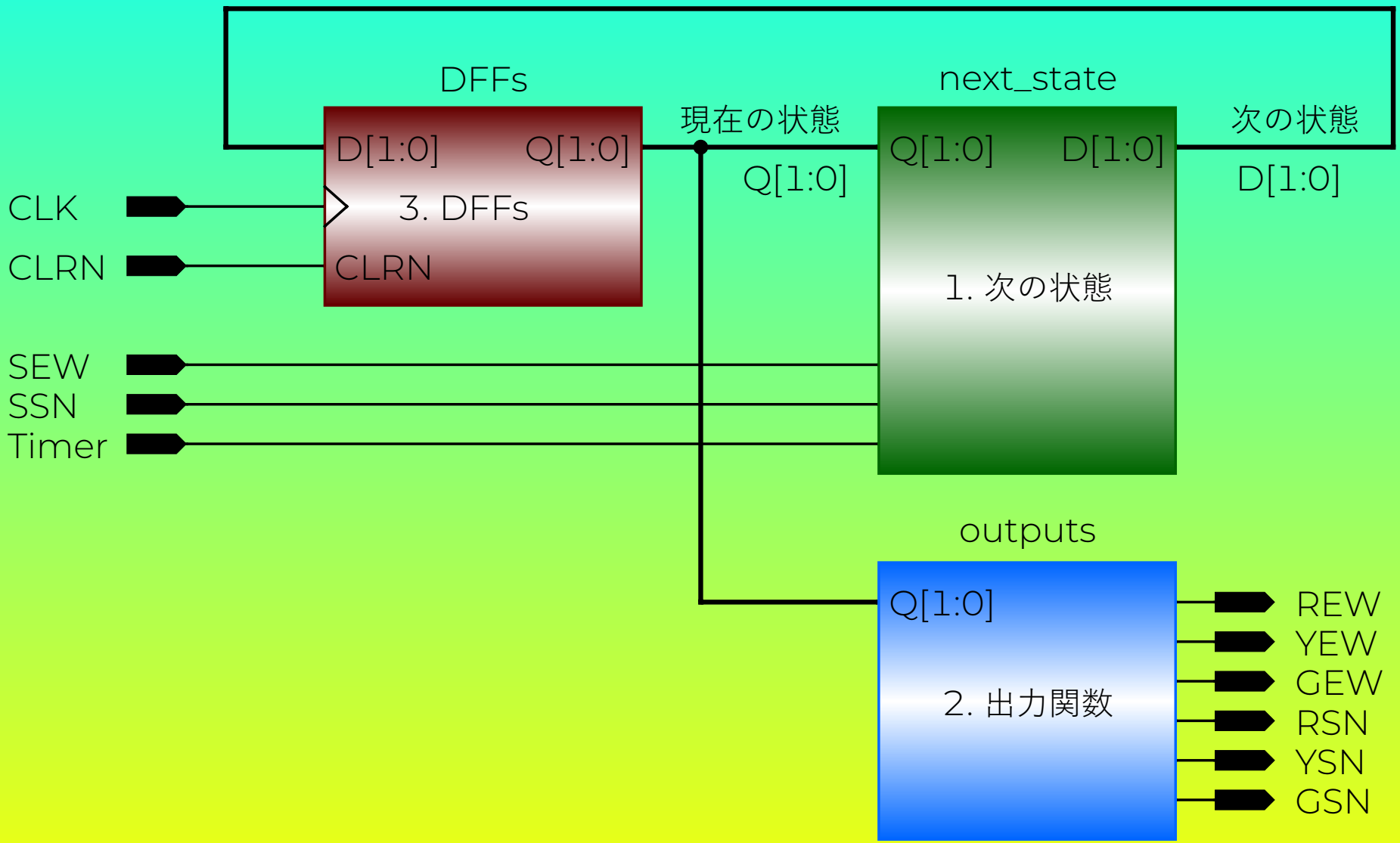
tlc_tb.v

MAC 対応: ターミナルで gtkwave 起動後 SST の tlc_tb の左側にある三角形のボタンを押し、表示された i0 を選択することで機械の中の信号も全て摘出できる。
Windows/ModelSim 対応: add wave -r /*

発展：入力を同期する


DFFs
D[1:0]
Q[1:0]
3. DFFs
CLRN
CLK
CLRN
SEW
SSN
Timer
DFF3
現在の状態
Q[1:0]
next_state
Q[1:0]
D[1:0]
1. 次の状態
次の状態
D[1:0]
outputs
Q[1:0]
2. 出力関数
REW
YEW
GEW
RSN
YSN
GSN

入力を同期しない


DFFs
D[1:0]
Q[1:0]
3. DFFs
CLRN
CLK
CLRN
現在の状態
Q[1:0]
next_state
Q[1:0]
D[1:0]
1. 次の状態
次の状態
D[1:0]
SEW
SSN
Timer
outputs
Q[1:0]
2. 出力関数
REW
YEW
GEW
RSN
YSN
GSN