EECS 151 Disc 4

Rahul Kumar (session 1)
Yukio Miyasaka (session 2)

Berkeley
UNIVERSITY OF CALIFORNIA

Contents

State Machines

- A way to systematically design sequential circuits
- Moore: outputs depend only on current state
- Mealy: outputs depend on current state and input

Example

Design a state machine to determine whether an unsigned binary integer is divisible by 3.

Inputs:

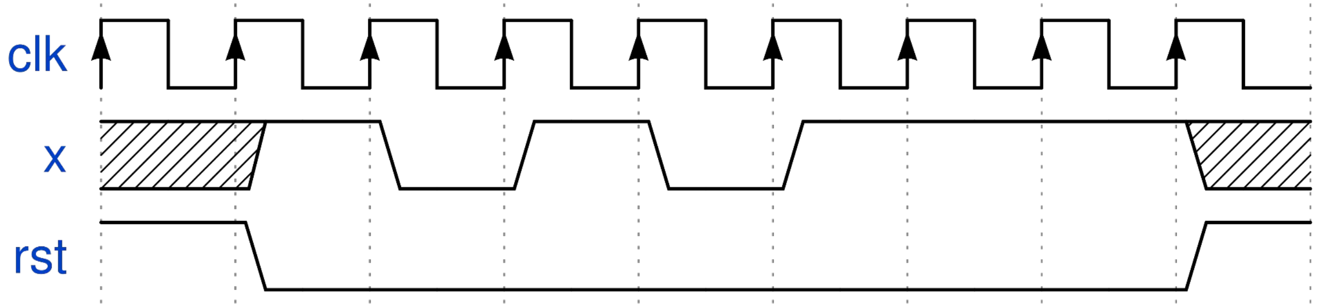
- x: the input integer. Given to FSM one bit at a time.
- rst: synchronous reset. At the next clock edge, clears any state associated with the FSM.

Output: one bit indicating if the value seen so far is divisible by 3.

Example

Input is supplied one bit at a time, MSB first.

Example: 1010111



Example

Draw a state transition diagram for this state machine.

Example

- Key idea: keep track of the number modulo 3.
- If 0, the number is divisible by 3.
- Can we track N % 3 without storing N?

Example

What happens when we get a new bit? In pseudocode:

```
next = 2*prev + bit
```

Same equation applies modulo 3!

```
next_residue = (2*prev_residue + bit) % 3
```

Example

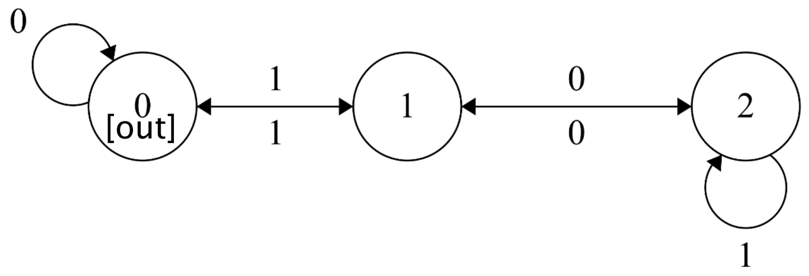
So keep 3 states, S0/S1/S2, corresponding to residues mod 3.

S0: if input is 1, go to S1. Else, stay at S0.

S1: if input is 1, go to S0. Else, go to S2.

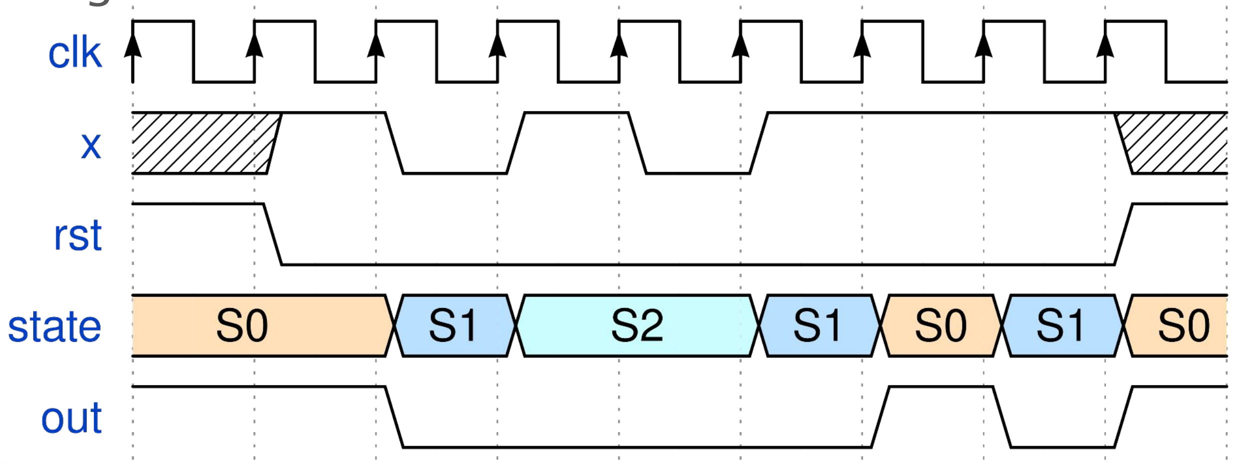
S2: if input is 1, stay at S2. Else, go to S1.

Example (Moore Machine)



Example

Implement this FSM as a Moore machine in Verilog, using a binary state encoding.

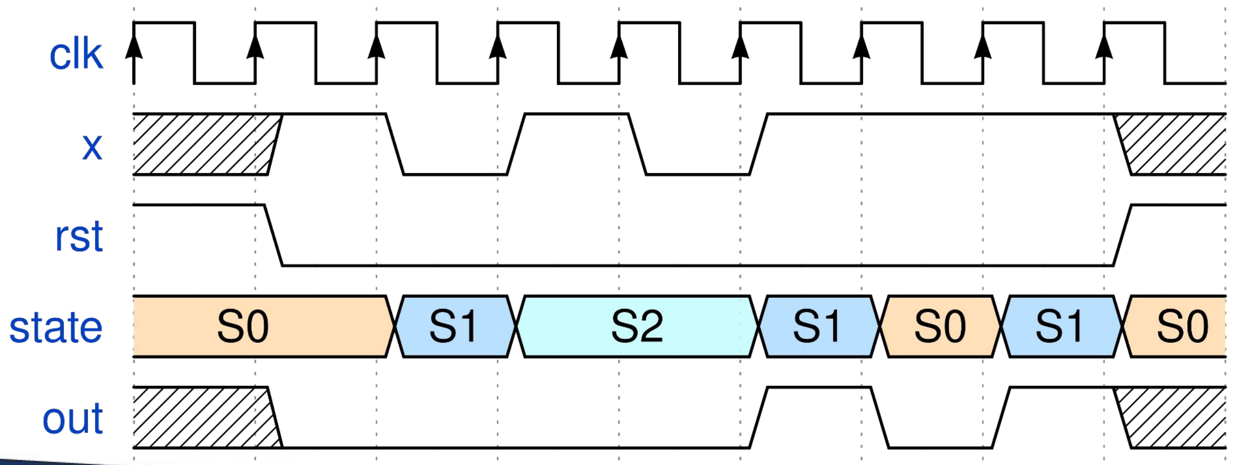


Example

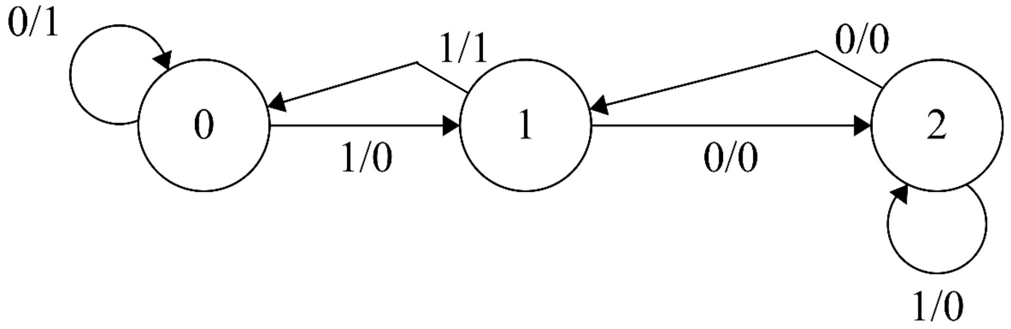
```
module div3_moore(
  input clk,
  input x,
  input rst,
  output out
);
  localparam S0 = 2'b00;
  localparam S1 = 2'b01;
  localparam S2 = 2'b10;
  reg [1:0] next_state;
  wire[1:0] state;
  REGISTER_R #(.N(2), .INIT(S0)) state_reg (.q(state), .d(next_state), .rst(rst), .clk(clk));

  assign out = state == S0;

  always @(*) begin
    case (state)
      S0: begin
        next_state = x ? S1 : S0;
      end
      S1: begin
        next_state = x ? S0 : S2;
      end
      S2: begin
        next_state = x ? S2 : S1;
      end
    endcase
  end
endmodule
```

Example

Now implement the same logic as a Mealy machine. The output should update as soon as the input updates.



Example (Mealy Machine)



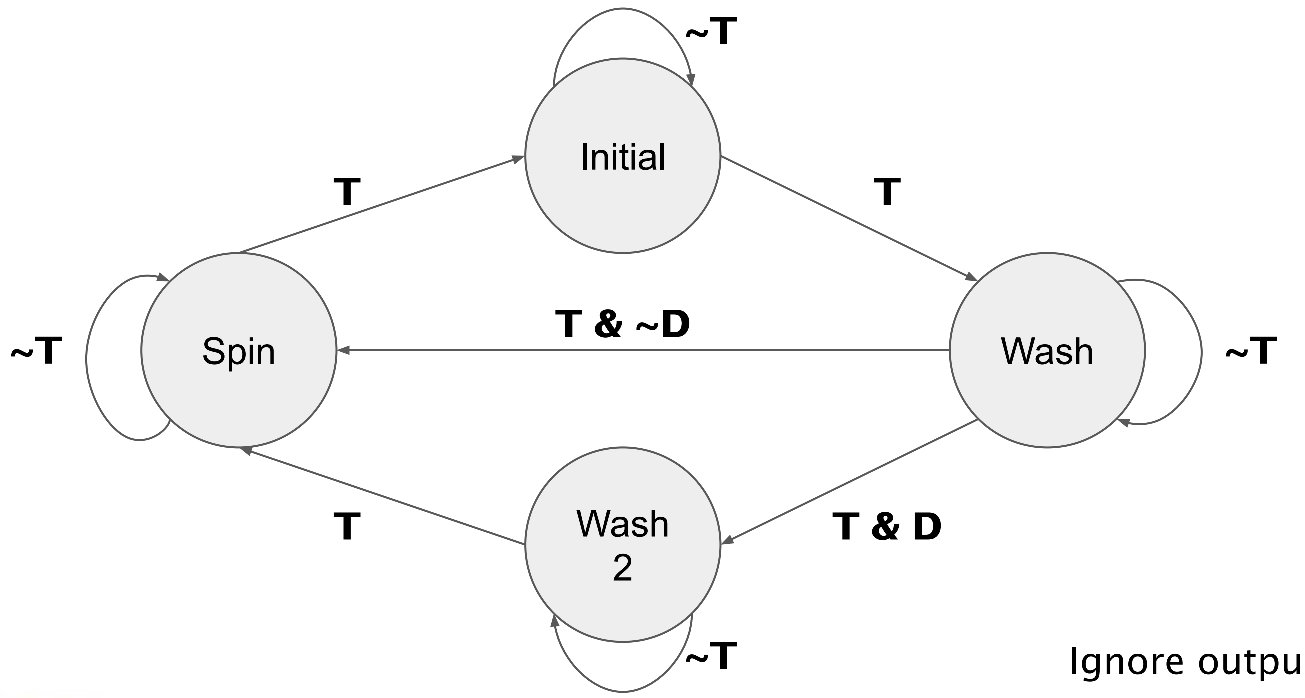
Example

```verilog
module div3_mealy(
  input clk,
  input x,
  input rst,
  output out
);
  localparam S0 = 2'b00;
  localparam S1 = 2'b01;
  localparam S2 = 2'b10;
  reg [1:0] next_state;
  wire[1:0] state;
  REGISTER_R #(.N(2), .INIT(S0)) state_reg (.q(state), .d(next_state), .rst(rst), .clk(clk));

  assign out = next_state == S0;

  always @(*) begin
    case (state)
      S0: begin
        next_state = x ? S1 : S0;
      end
      S1: begin
        next_state = x ? S0 : S2;
      end
      S2: begin
        next_state = x ? S2 : S1;
      end
    endcase
  end
endmodule
```

Example: Washing machine

- Takes two steps: wash and spin
- Wash twice if a double wash switch **D** is on
  - wash -> wash -> spin
- Receives a signal **T** from a timer:
  - When it should start washing
  - At the end of each step

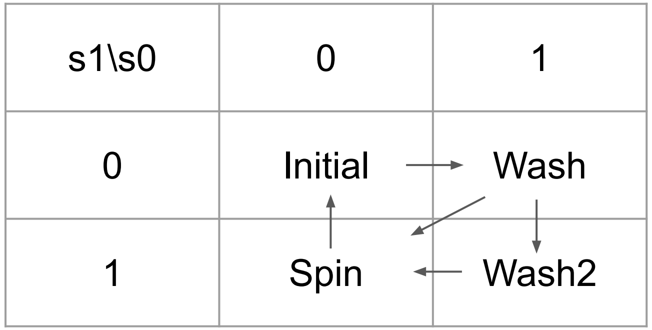
State Transition Diagram



Ignore output signal for now

State Assignment

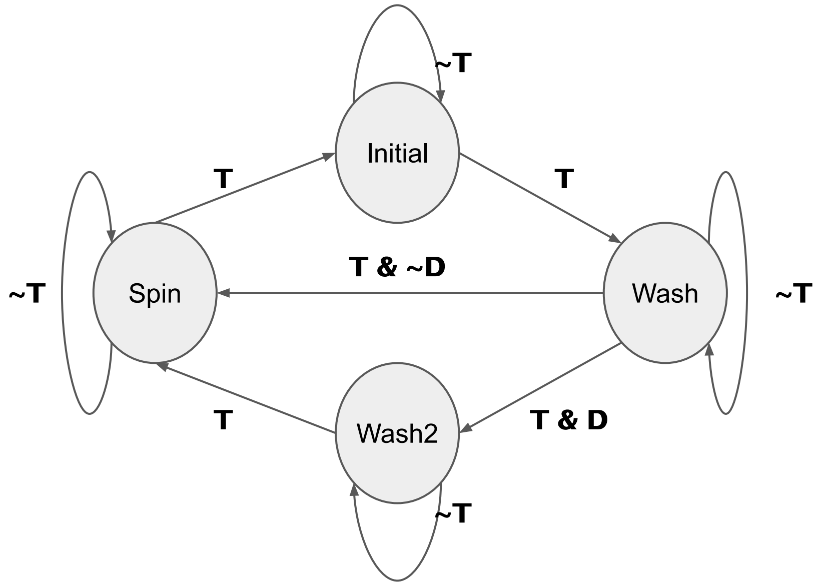
Initial	= state 0
Wash	= state 1
Wash2	= state 3
Spin	= state 2



s1\s0	0	1
0	Initial	Wash
1	Spin	Wash2

Transitions in binary encoding

Verilog with Binary Encoding



```
module washing_machine_binary(input clk, rst, T, D);
  localparam Initial = 2'b00;
  localparam Wash    = 2'b01;
  localparam Wash2   = 2'b11;
  localparam Spin    = 2'b10;

  reg [1:0] next_state;
  wire[1:0] state;
  REGISTER_R #(.N(2)) state_reg (.q(state),
.d(next_state), .rst(rst), .clk(clk));

  always @(*) begin
    next_state = state;
    case (state)
      Initial: if(T) next_state = Wash;
      Wash:    if(T) next_state = D? Wash2: Spin;
      Wash2:   if(T) next_state = Spin;
      Spin:    if(T) next_state = Initial;
    endcase
  end
endmodule
```

Berkeley
UNIVERSITY OF CALIFORNIA

Gate-Level Circuit with Binary Encoding

s1	s0	T	D	n1	n0
0	0	1	-	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	1	-	0	0
1	1	1	-	1	0
x	y	0	-	x	y

Gate-Level Circuit with Binary Encoding

s1	s0	T	D	n1	n0
0	0	1	-	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	1	-	0	0
1	1	1	-	1	0
x	y	0	-	x	y

TD

s1s0

	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	1	1
10	1	1	0	0

K-map for n1

TD

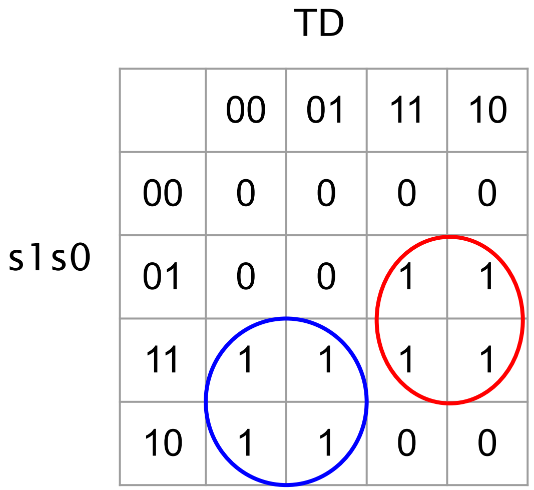
s1s0

	00	01	11	10
00	0	0	1	1
01	1	1	1	0
11	1	1	0	0
10	0	0	0	0

K-map for n0

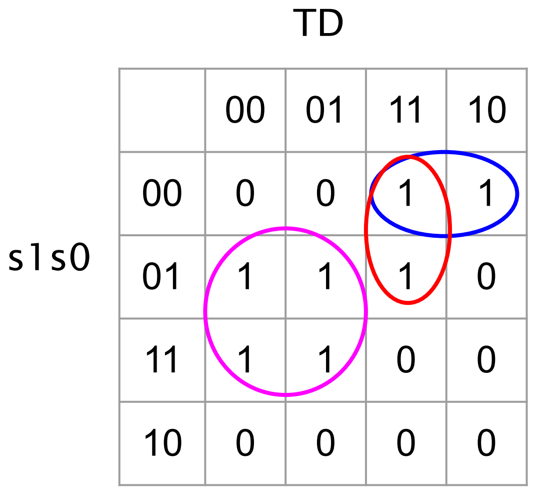
Gate-Level Circuit with Binary Encoding

s1	s0	T	D	n1	n0
0	0	1	-	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	1	-	0	0
1	1	1	-	1	0
x	y	0	-	x	y



s1s0 \ TD	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	1	1
10	1	1	0	0

K-map for n1



s1s0 \ TD	00	01	11	10
00	0	0	1	1
01	1	1	1	0
11	1	1	0	0
10	0	0	0	0

K-map for n0

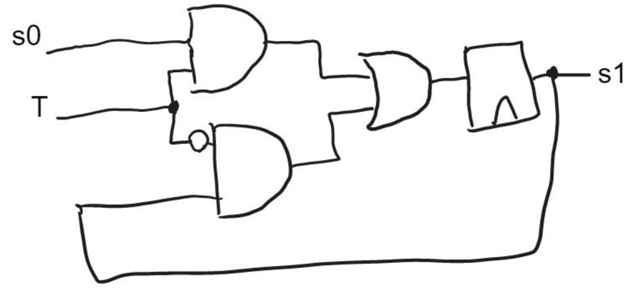
n1 = s0 T + s1 T'

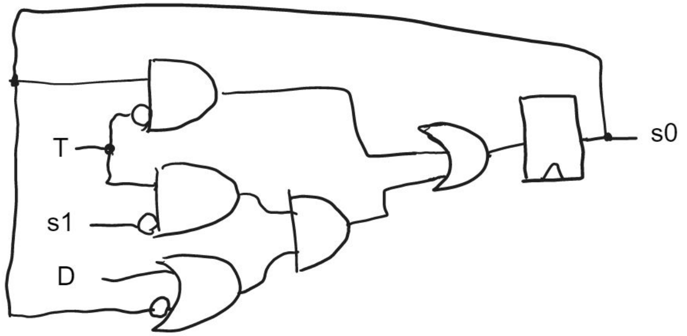
n0 = s0 T' + s1' T D + s0' s1' T

Gate-Level Circuit with Binary Encoding

n1 = s0 T + s1 T'

n0 = s0 T' + s1' T D + s0' s1' T
= s0 T' + s1' T (D + s0')



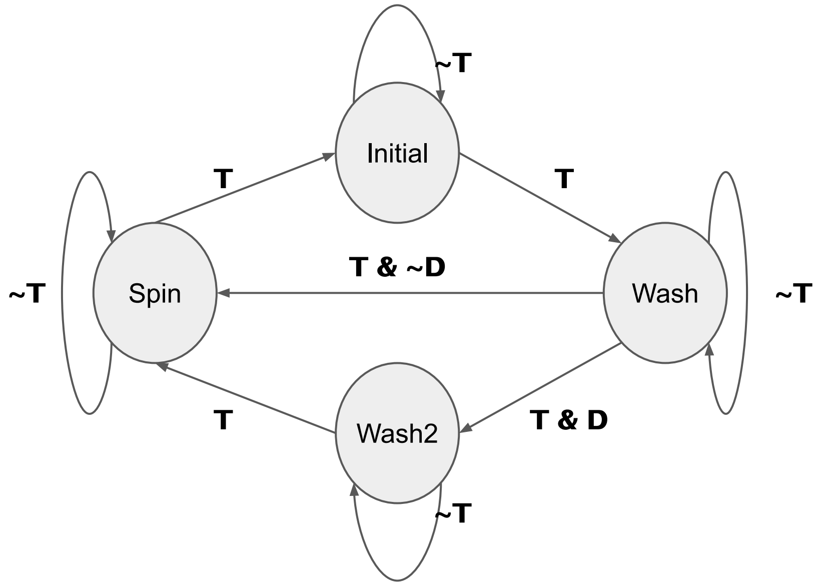


One-hot Encoding

- state 0 = 0001 (Initial)
- state 1 = 0010 (Wash)
- state 2 = 0100 (Spin)
- state 3 = 1000 (Wash2)

State assignment doesn't really matter in one-hot encoding

Verilog with One-hot Encoding

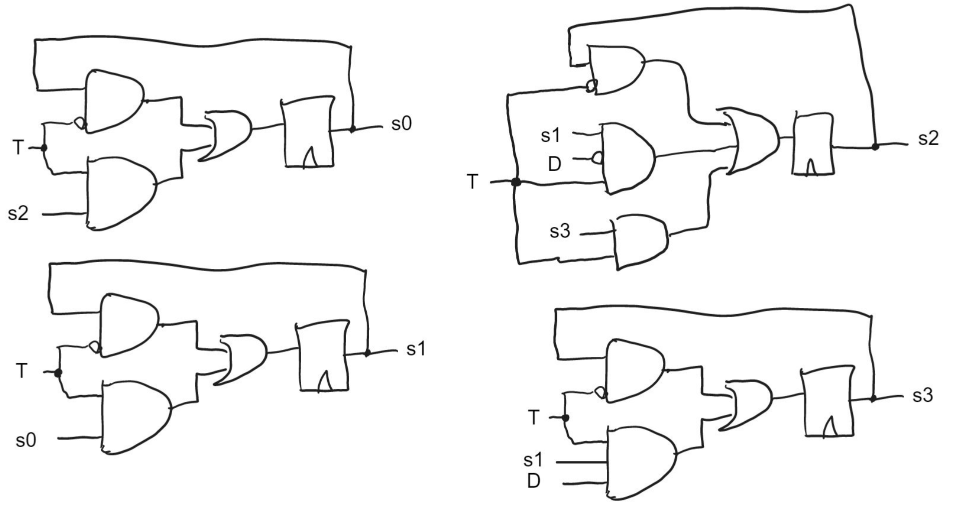


```
module washing_machine_onehot(input clk, rst, T, D);
  wire [3:0] next_state, state;
  REGISTER_R #(.N(4)) state_reg (.q(state),
.d(next_state), .rst(rst), .clk(clk));

  assign next_state[0] = (state[0] & ~T) |
                         (state[2] &  T);
  assign next_state[1] = (state[1] & ~T) |
                         (state[0] &  T);
  assign next_state[2] = (state[2] & ~T) |
                         (state[1] &  T & ~D) |
                         (state[3] &  T);
  assign next_state[3] = (state[3] & ~T) |
                         (state[1] &  T & D);
endmodule
```

Focus on incoming edges for each node

Gate-Level Circuit with One-hot Encoding

```
assign next_state[0] = (state[0] & ~T) |
                       (state[2] &  T);
assign next_state[1] = (state[1] & ~T) |
                       (state[0] &  T);
assign next_state[2] = (state[2] & ~T) |
                       (state[1] &  T & ~D) |
                       (state[3] &  T);
assign next_state[3] = (state[3] & ~T) |
                       (state[1] &  T & D);
```


T
s2
s0
T
s0
s1
T
s1
D
s3
s2
T
s1
D
s3