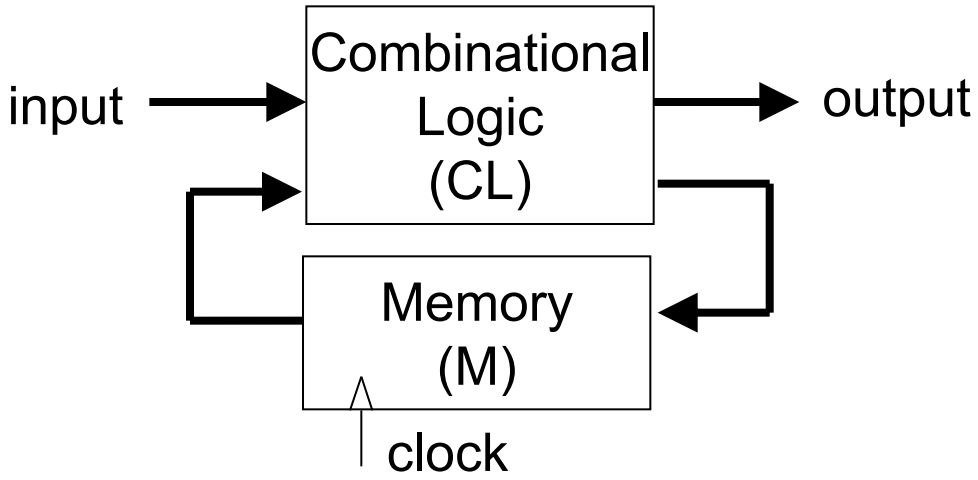
09 Finite State Machine (FSM) I

- What is a Finite State Machine (FSM)?
- States and their representations
- State Transition Diagram (STD)
- Number of Flip-Flops in System
- Separating Combinational Logic into 2
- State Encoding
- Put it all together into schematics
- Put it all together into Verilog

What is a Finite State Machine (FSM)?



Finite = less than infinity
Machine = circuit or system

State = set of (dynamic) information that is sufficient for the system to work

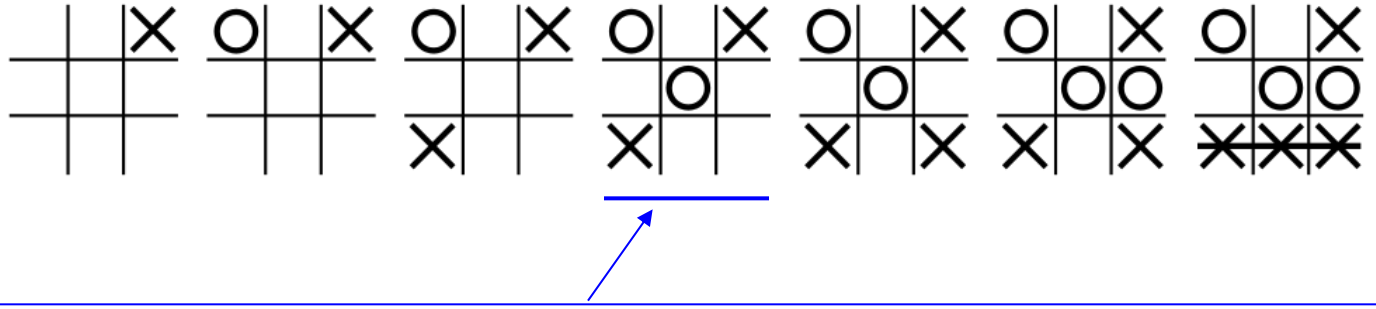
What is a State?

State = set of (dynamic) information that is sufficient for the system to work

Example:

- playing tic-tac-toe:

[WIKI]



Each board position is a state:

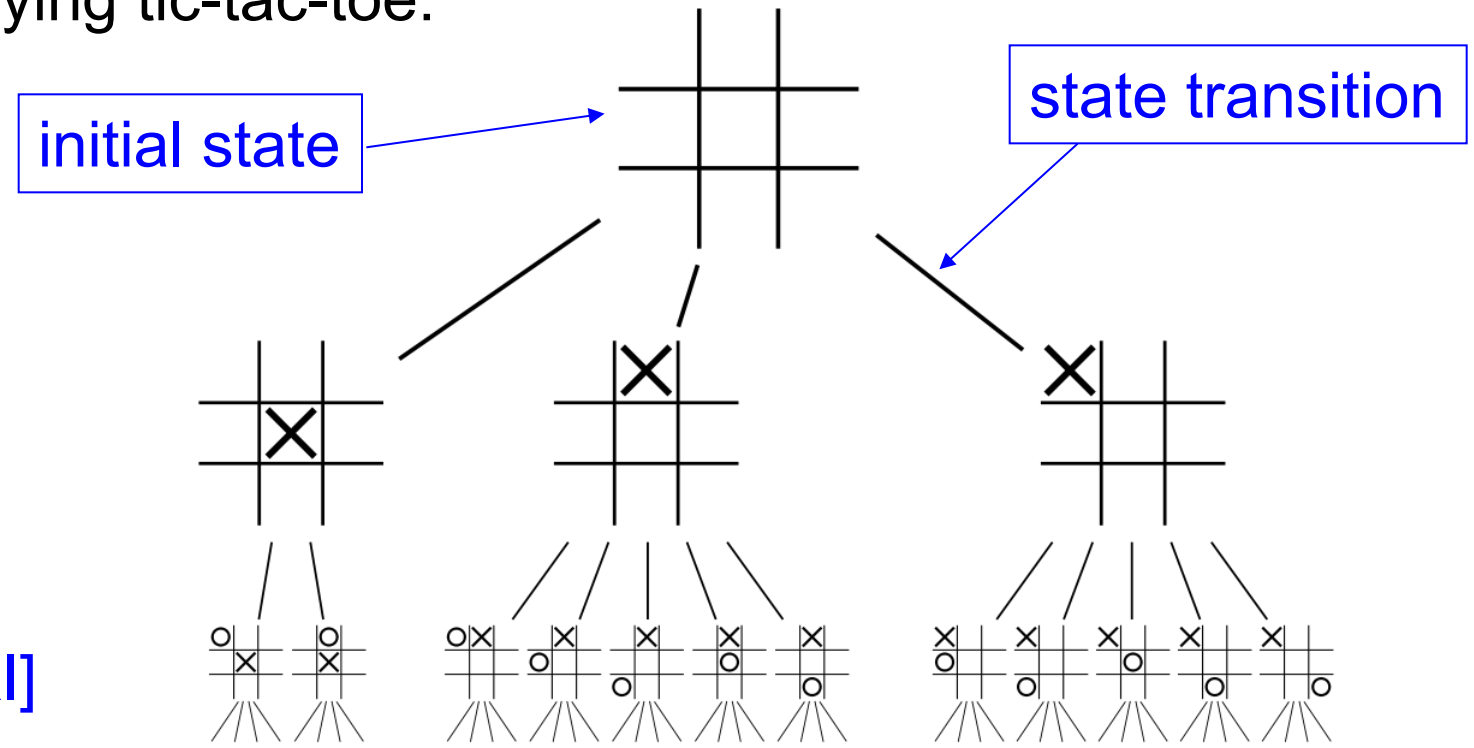
1. knows result (in progress, X wins, O wins, draw)
2. knows who goes next, X or O
3. can deduce the optimal next move

What is a State?

State	= set of (dynamic) information that is sufficient for the system to work

Example:

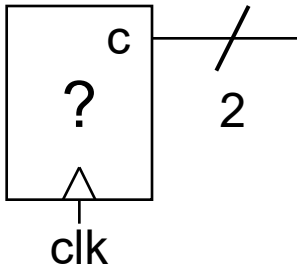
- playing tic-tac-toe:



[WIKI]

FSM example:

Example: - 0, 1, 2, 3 counter



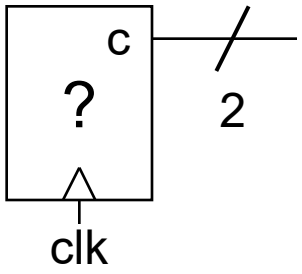
c: 0, 1, 2, 3, 0, 1, 2, 3, …

- every clock cycle, c changes in sequence

- Can this box have M (flip-flops) only? NO
- Can this box have CL only? NO

Box has both CL and M → It is an FSM.

FSM example:



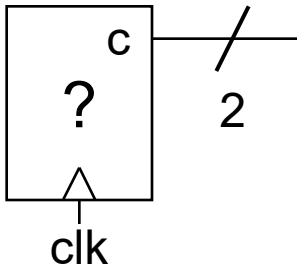
c: 0, 1, 2, 3, 0, 1, 2, 3, …

State = set of (dynamic) information that is sufficient for the system to work

What should the states be?

- current value of c → how many values? 4
- need to store current values of c
  → how many flip flops? 2

FSM example:



c: 0, 1, 2, 3, 0, 1, 2, 3, …

State = set of (dynamic) information that is sufficient for the system to work

- need to store states from one clock cycle to next

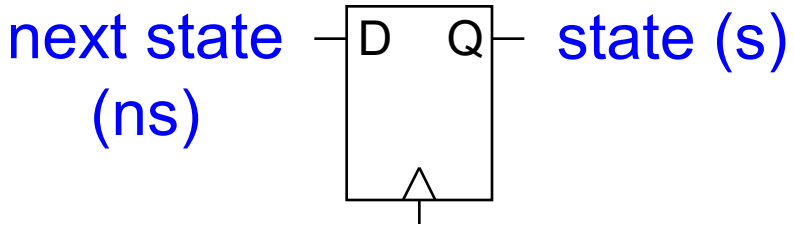
State = output of the flip-flops inside circuit

FSM example:

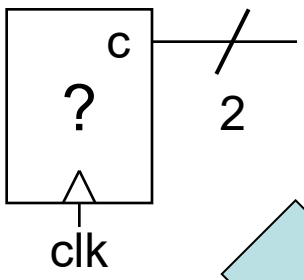
State = set of (dynamic) information that is sufficient for the system to work

State = output of the flip-flops inside circuit

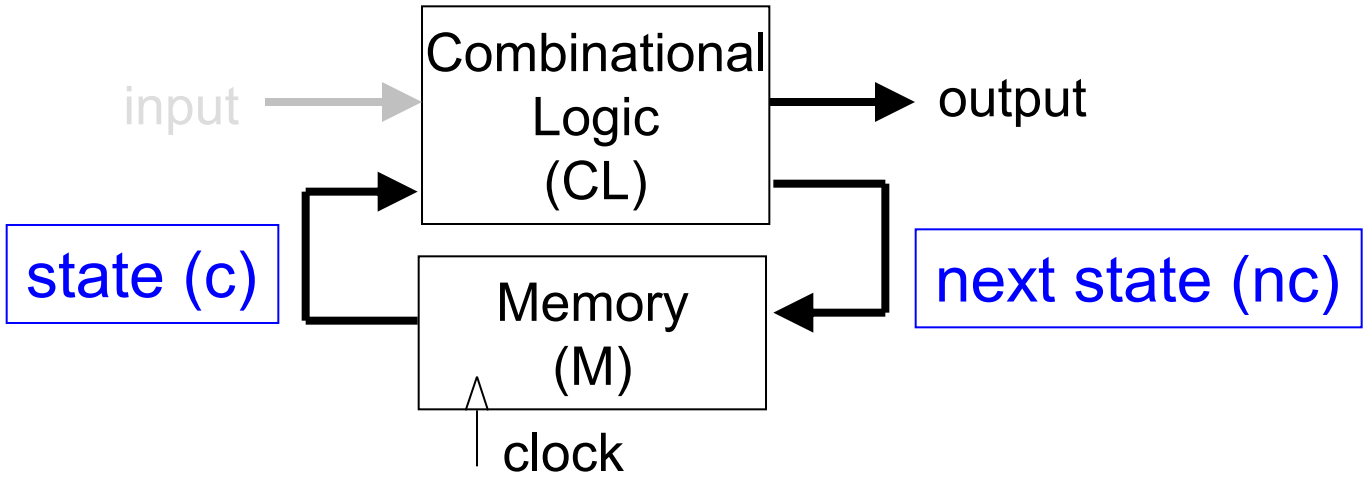
Next State = input of the flip-flops inside circuit



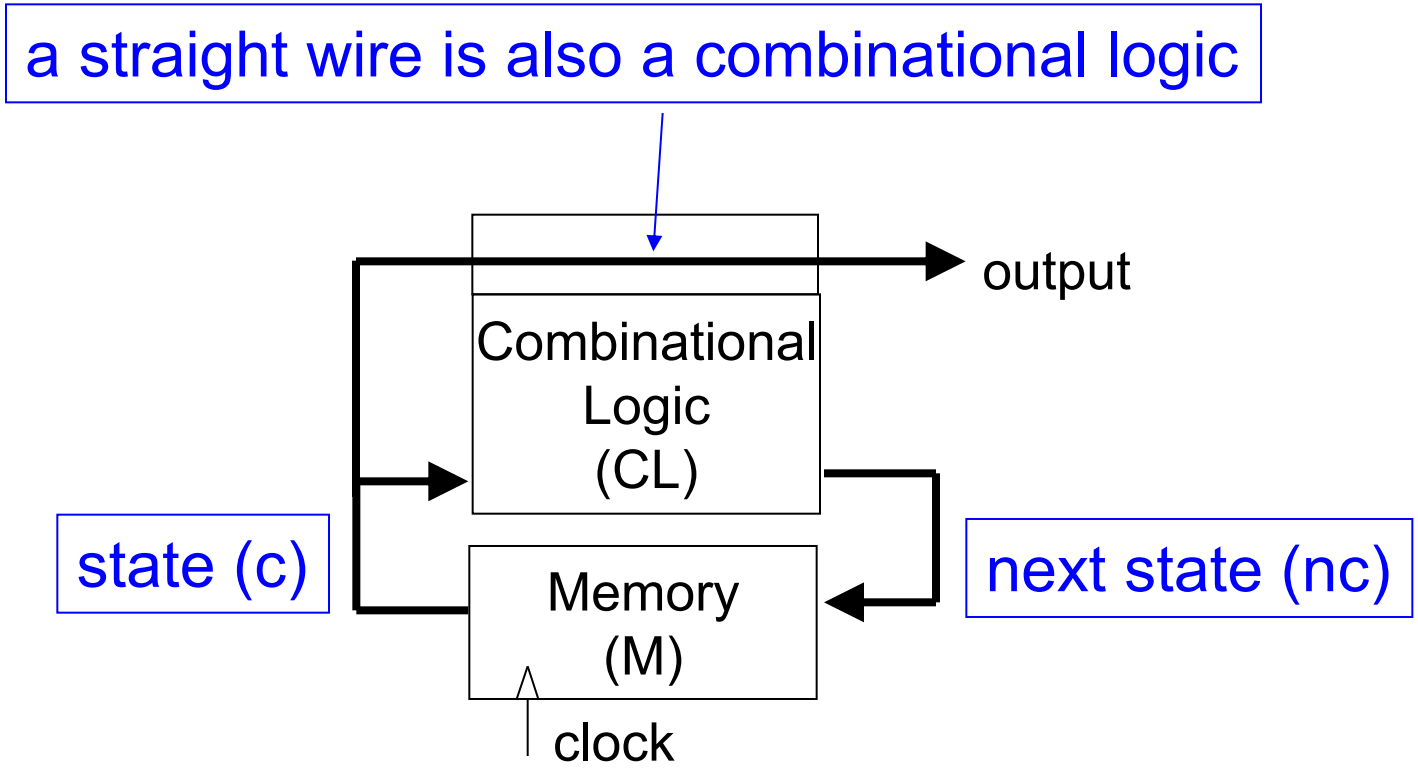
FSM example:



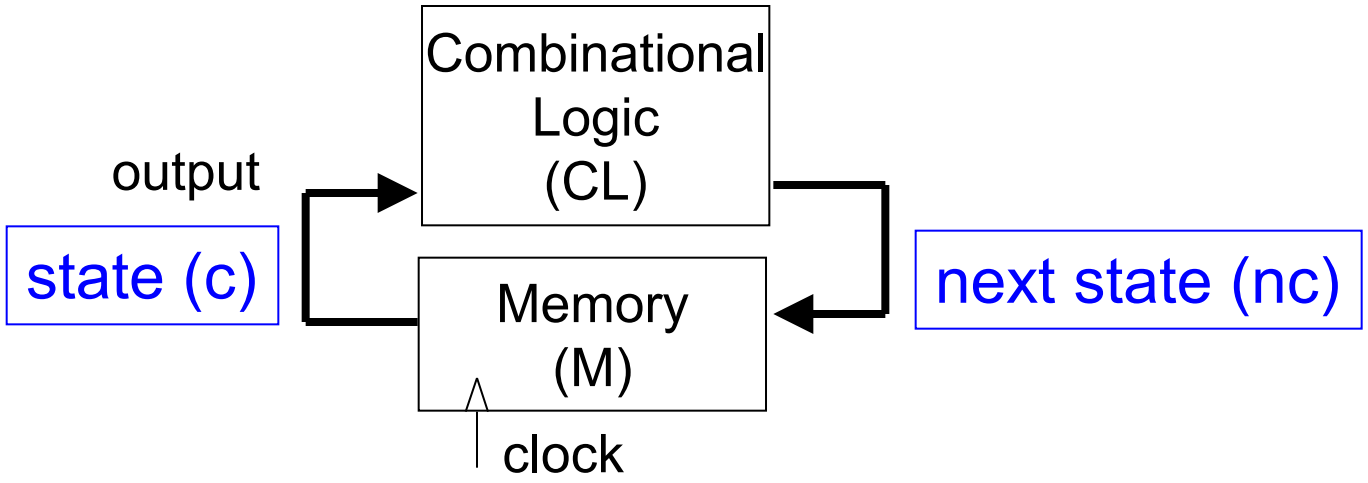
c: 0, 1, 2, 3, 0, 1, 2, 3, …



FSM example: c: 0, 1, 2, 3, 0, 1, 2, 3, …



FSM example: c: 0, 1, 2, 3, 0, 1, 2, 3, …



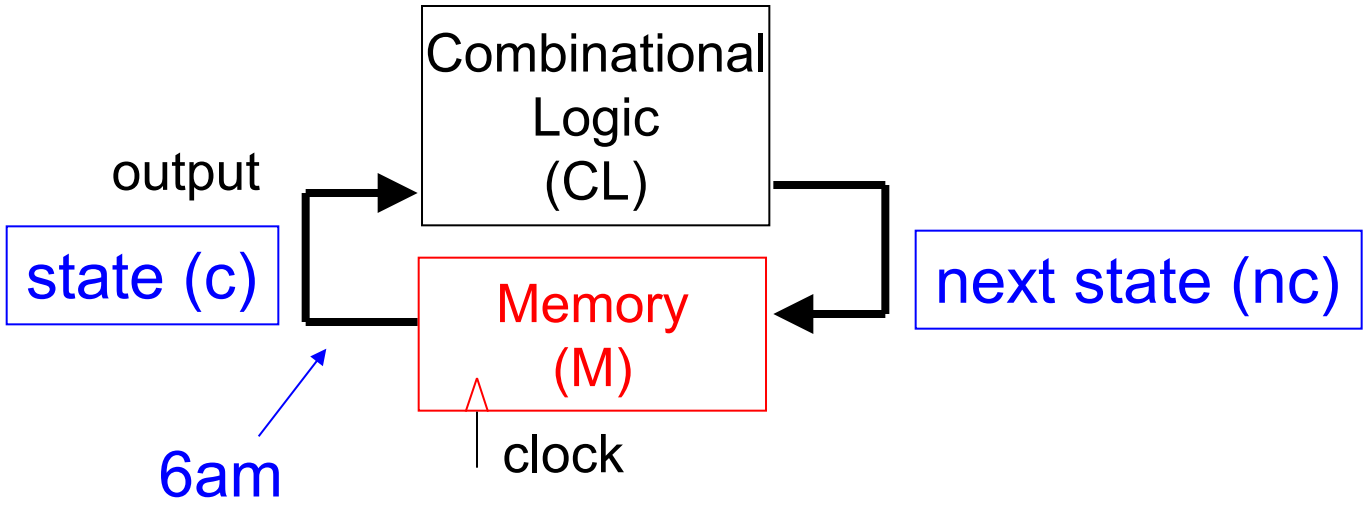
2 persons working in this factory

- CL (Claire): can calculate, but not remember
- M (Mike): can remember, cannot think

one clock period = one day

CLaire works 6am-6pm, Mike 6pm-6am

FSM example: c: 0, 1, 2, 3, 0, 1, 2, 3, …



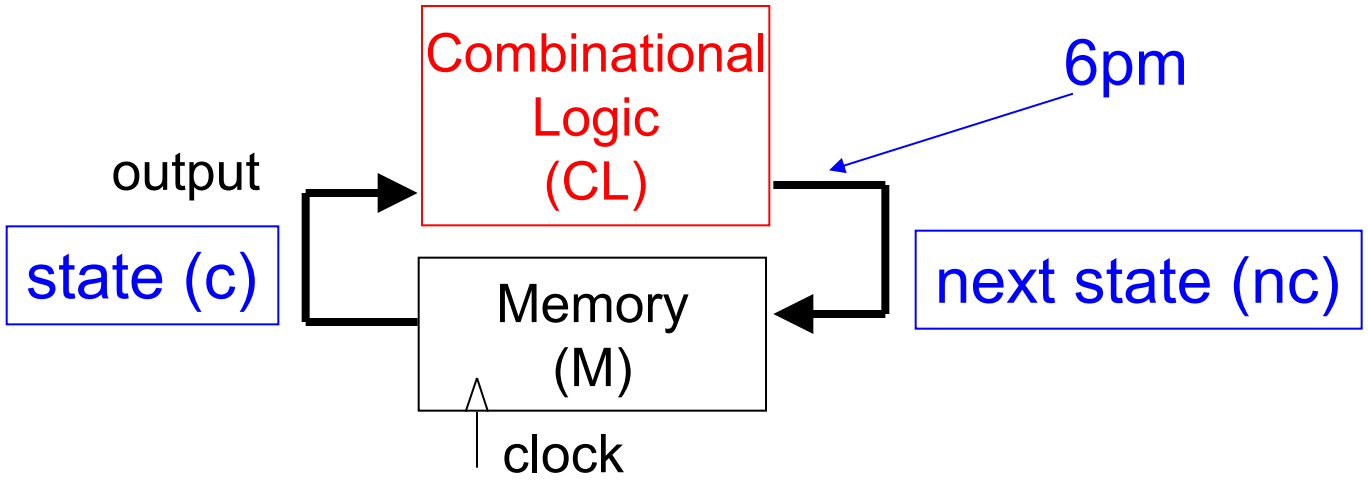
at 6am

- Mike gives what he remembers from last night to CLaire

(next c from yesterday becomes c today)

- This c is also shown as system output

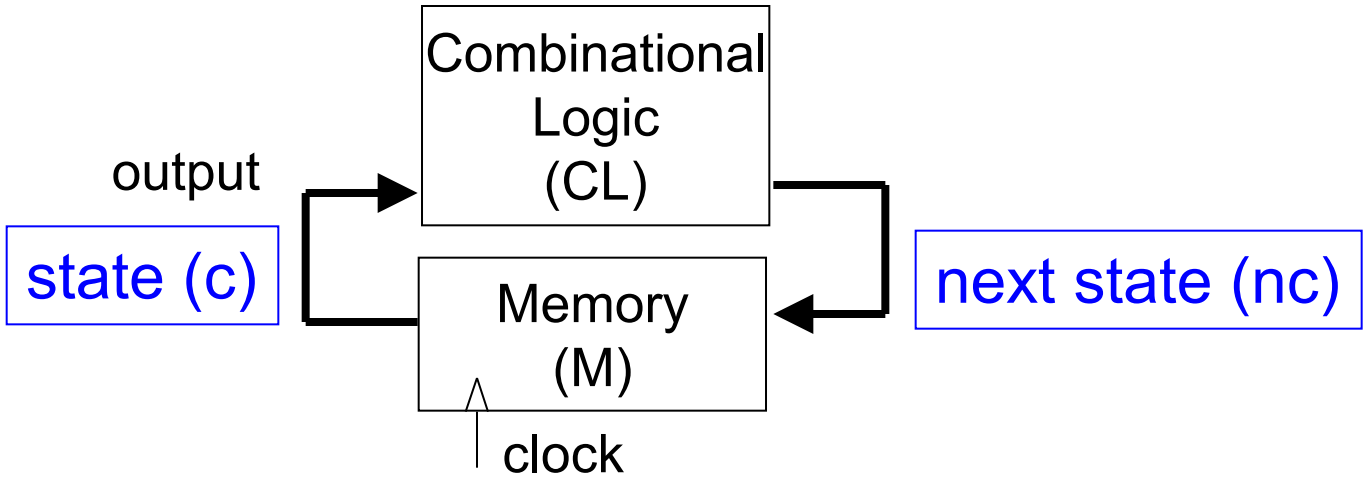
FSM example: c: 0, 1, 2, 3, 0, 1, 2, 3, …



From 6am-6pm
- CLaire sees c from Mike
- CLaire calculates what next c should be
- at 6pm, CLaire gives next c to Mike

- CLaire does not need to remember next c

FSM example: c: 0, 1, 2, 3, 0, 1, 2, 3, …



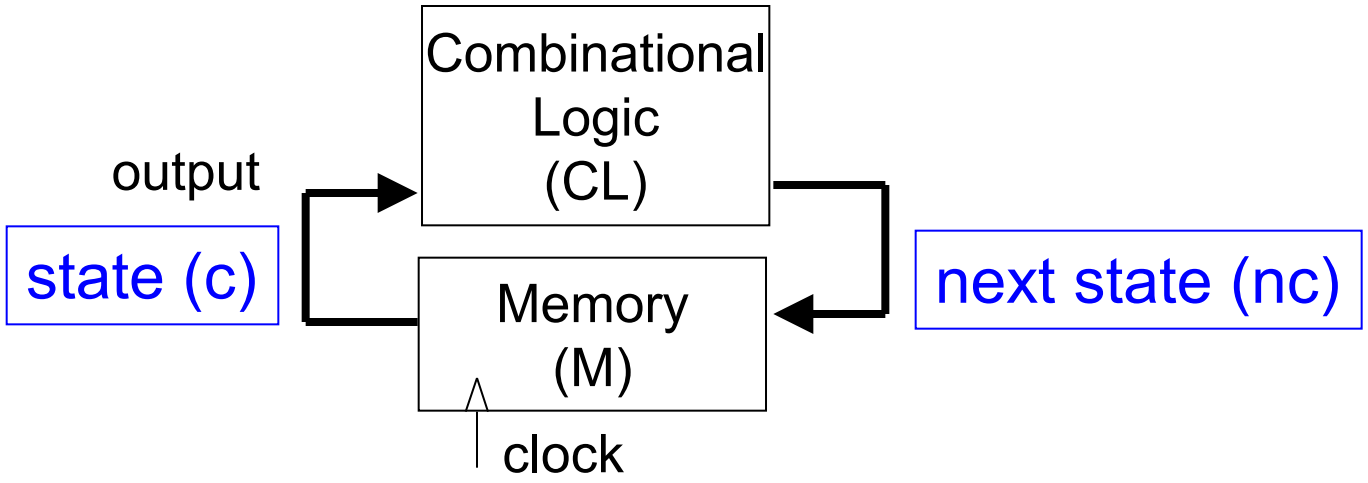
What does CLaire do?

- Look at c
  if c is 3, let nc be 0
  else nc is c+1

What does Mike do?

- store nc that CLaire gives at 6pm, and give it back to CLaire next morning, calling it c

FSM example: c: 0, 1, 2, 3, 0, 1, 2, 3, …



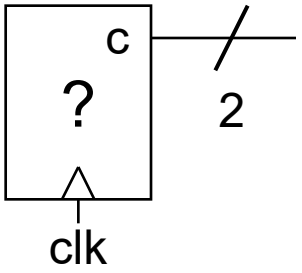
Outsides look at c and see 0, 1, 2, 3, 0, 1, 2, 3, …

Real life circuits:

- Mike takes a very short time to transfer nc to c
- Most of the time, CLaire is working to calculate nc from c

Representing FSM:State Transition Diagram

State = set of (dynamic) information that is sufficient for the system to work
= output of the flip-flops inside circuit



c: 0, 1, 2, 3, 0, 1, 2, 3, …

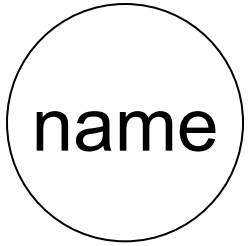
This system has

- 4 states (0, 1, 2, 3) (requires 2 flops)

State Transition Diagram

- graphical diagram
- shows relationships between current state (s) and next state (ns)
- does NOT represent circuits

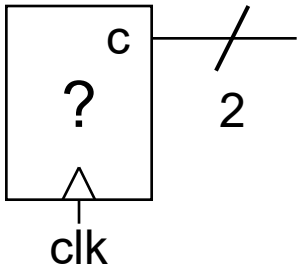
SYMBOLS USED:



circles = states
one circle per state
write name inside

arcs = arrows = transitions
comes out from current state
and points to next state

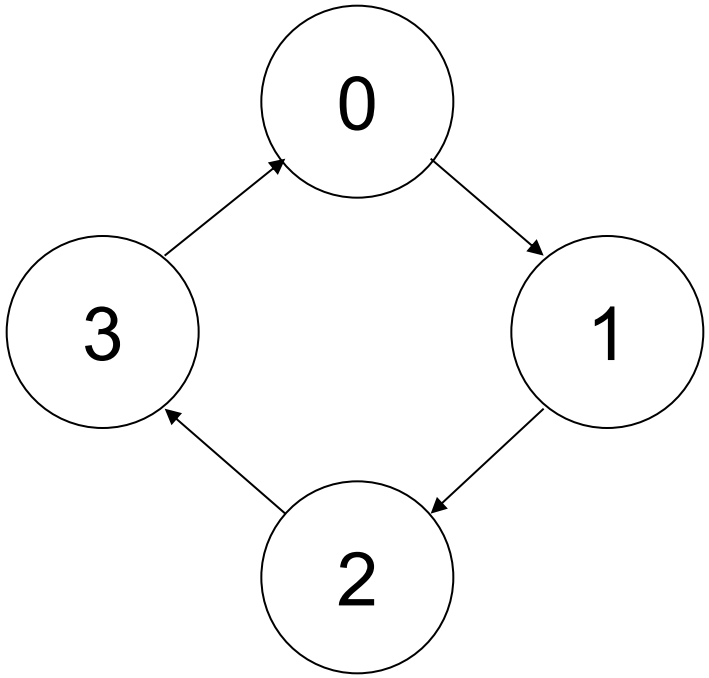
State Transition Diagram



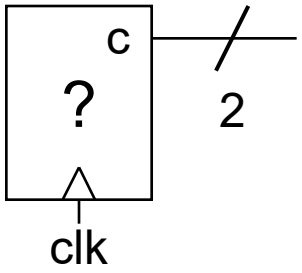
c: 0, 1, 2, 3, 0, 1, 2, 3, …

This system has 4 states (0, 1, 2, 3)

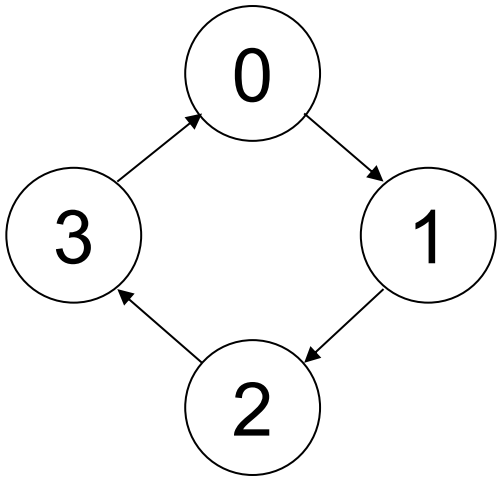
STD of system above:



Number of Flip-Flops



c: 0, 1, 2, 3, 0, 1, 2, 3, …



How many flip-flops inside box?

- state = output from flops
- 4 states
- output from flops must have at least 4 distinct values
- 2 bits of information = 4 distinct values

→ system requires (at least) 2 flops

Number of flops depends on number of states

Number of Flip-Flops

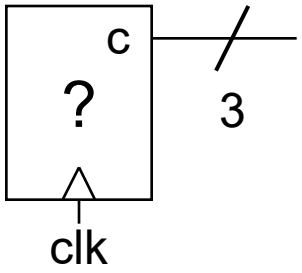
Number of flops depends on number of states

number of flops = ceiling(log2 (#of states))

examples:

3 states system needs at least 2 flops
4 states system needs at least 2 flops
8 states system needs at least 3 flops
5 states system needs at least 3 flops
9-16 states system needs 4 flops

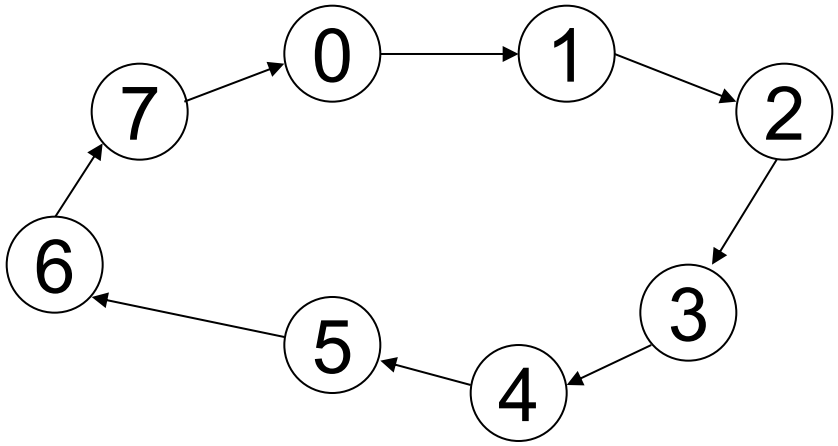
State Transition Diagram



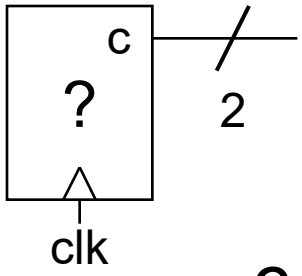
c: 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, …

This system has 8 states (0, …, 7)
requires at least 3 flip-flops

STD of system above:



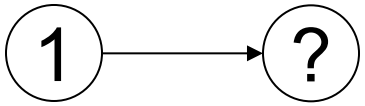
State Transition Diagram



c: 0, 1, 2, 3, 2 , 1, 0, 1, 2, 3, 2, 1, 0, 1, …

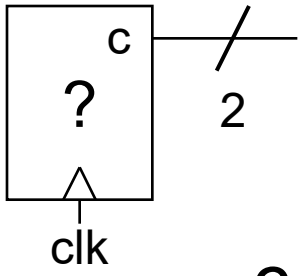
c output moves up and down in (0, 1, 2, 3)
What are the states?
This system has how many states?

The states cannot be c anymore



when c is 1,
sometimes next c is 0
sometimes next c is 2

State Transition Diagram



c: 0, 1, 2, 3, 2 , 1, 0, 1, 2, 3, 2, 1, 0, 1, …

c output moves up and down in (0, 1, 2, 3)

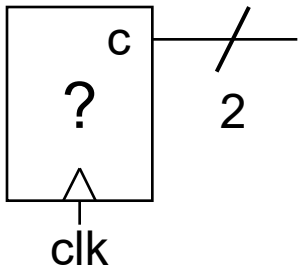
The states must contain BOTH c and DIRECTION (counting up or down)

State = set of (dynamic) information that is sufficient for the system to work

Knowing only c (output) is not sufficient

States are NOT the same as outputs

State Transition Diagram



c output moves up and down in (0, 1, 2, 3)

c: 0, 1, 2, 3, 2 , 1, 0, 1, 2, 3, 2, 1, 0, 1, …

state is 1up, output is 1

state is 1down, output is 1

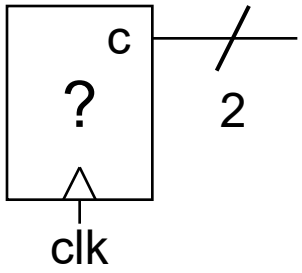
state is 3down, output is 3
(no 3up state)

state is 0up, output is 0
(no 0down state)

This system needs at least 6 states (3 flops)

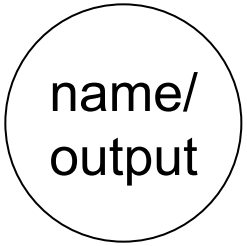
0up, 1up, 2up, 3down, 2down, 1down

State Transition Diagram

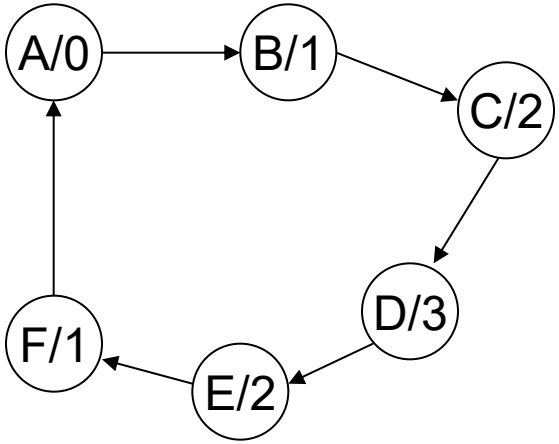


c: 0, 1, 2, 3, 2 , 1, 0, 1, 2, 3, 2, 1, 0, 1, …

Writing STD



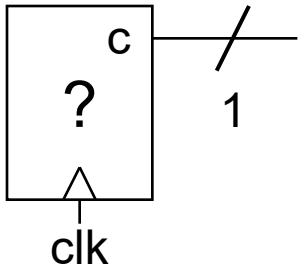
circles = states
one circle per state
write name and output inside



A = 0up, B = 1up, etc.

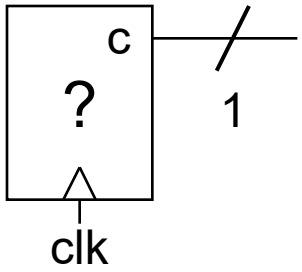
You can give states any name, but you MUST know what they mean

State Transition Diagram



c: 0, 0, 1, 1, 0, 0, 1, 1, …

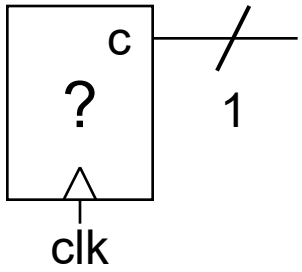
How many states? How many flops?



c: 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, …

1 2 3 4 1

How many states? How many flops?

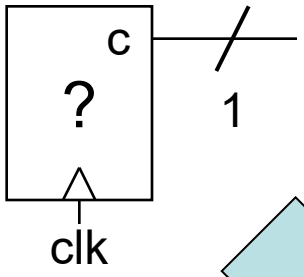


c: 0, 0, …, 0, 1, 0, 0, …, 0, 1, 0, 0, …, 0, 1

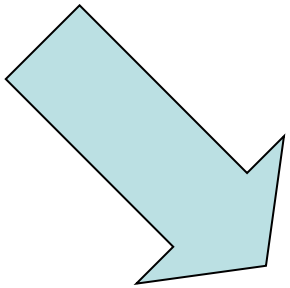
999,999 999,999 999,999

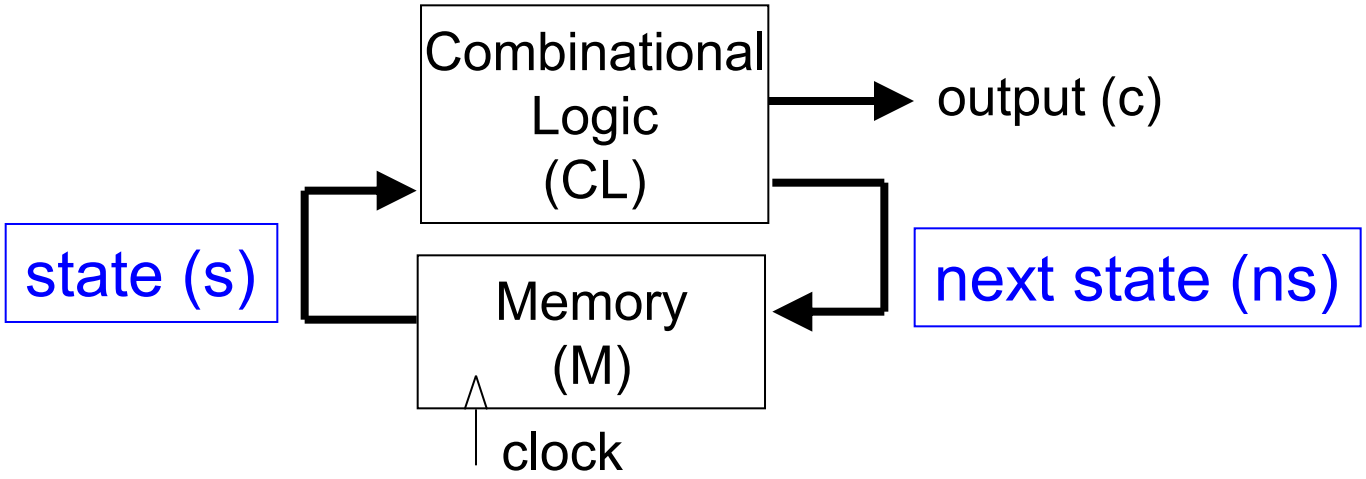
How many states? How many flops?

Back to Hardware:

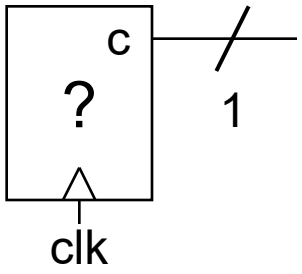


c: 0, 0, 1, 1, 0, 0, 1, 1, …





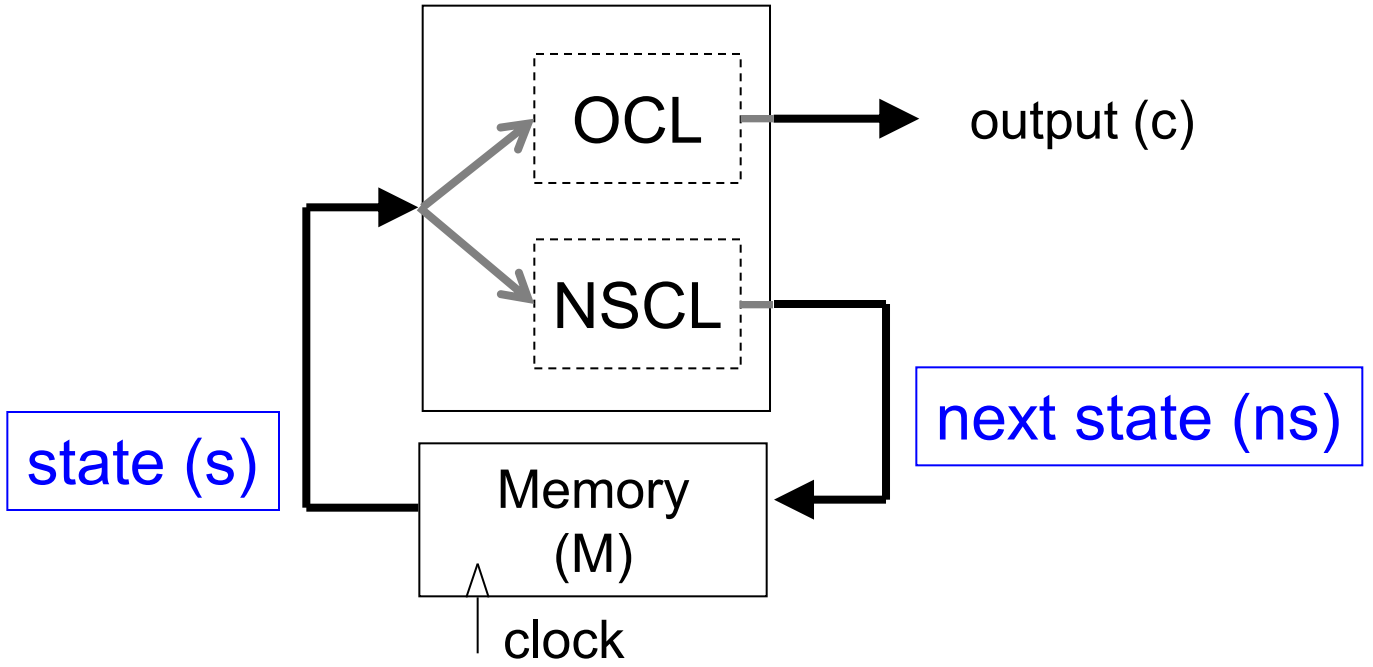
Back to Hardware:



c: 0, 0, 1, 1, 0, 0, 1, 1, …

---

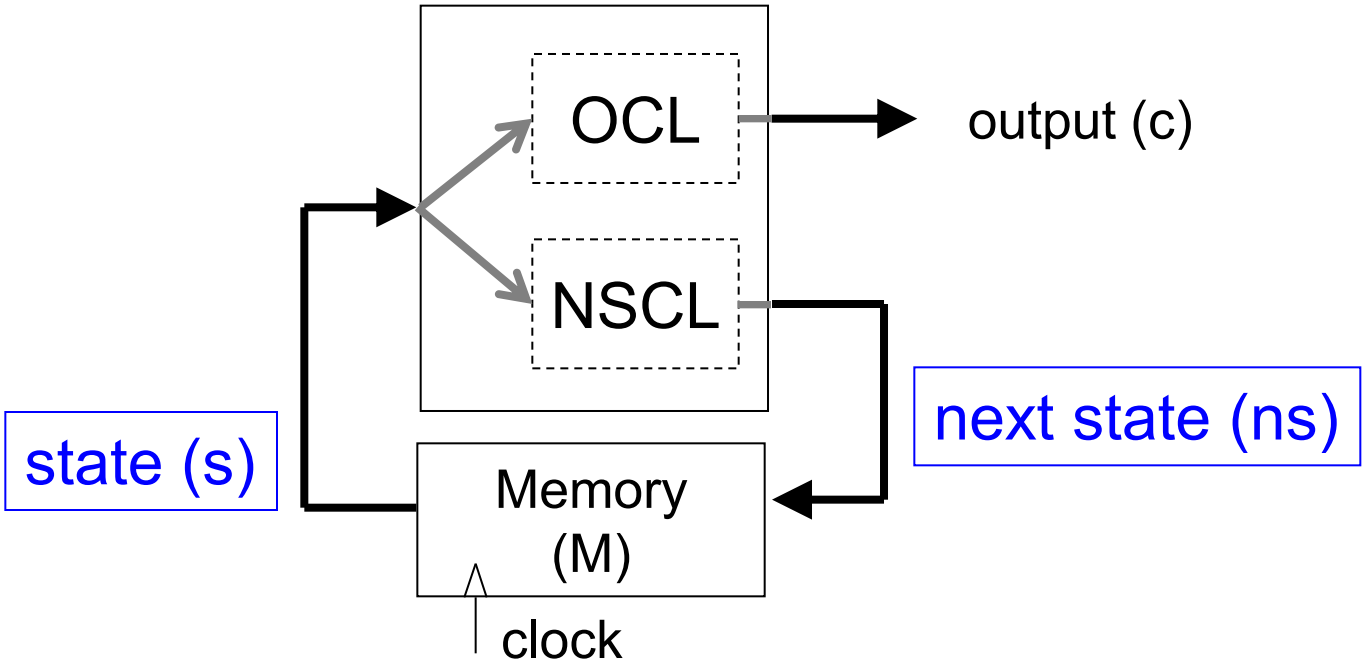
2 parts in Combinational Logic



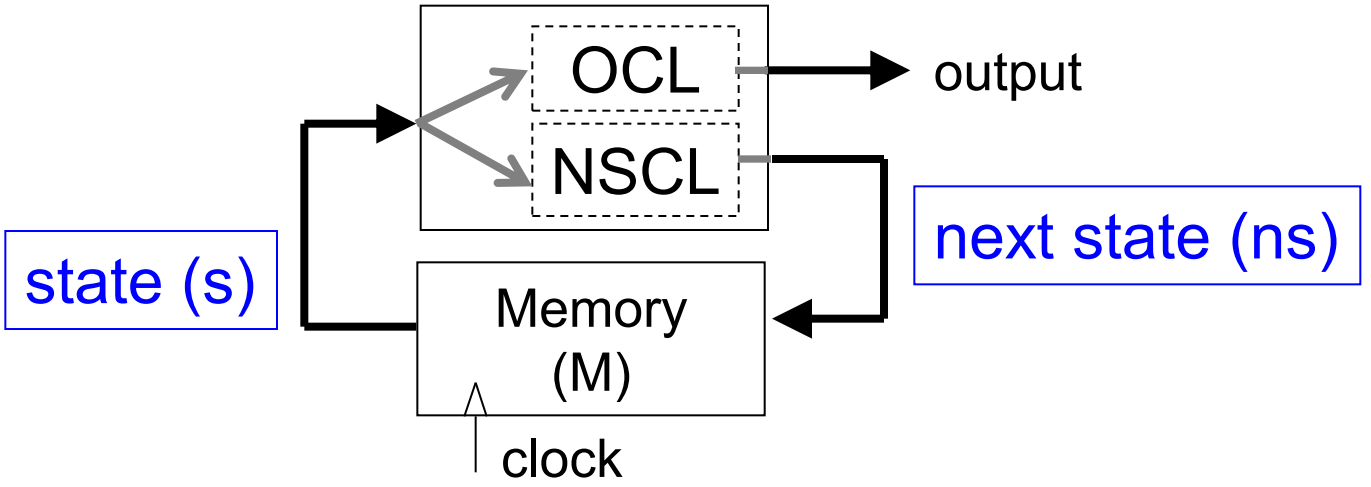
Back to Hardware:

2 parts in Combinational Logic

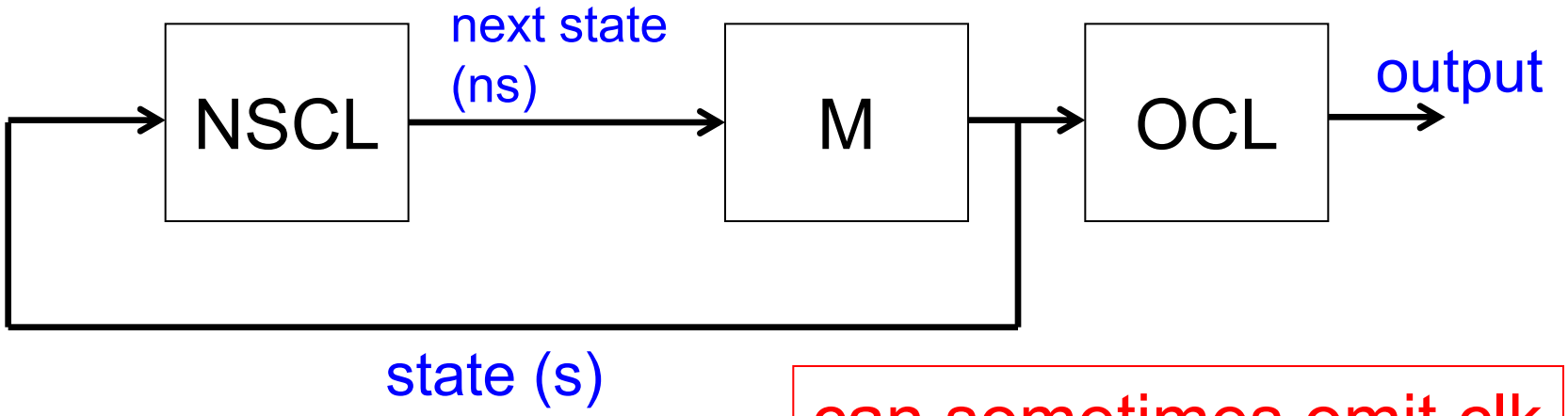
- NSCL (next state CL)
- OCL (output CL)



Back to Hardware:

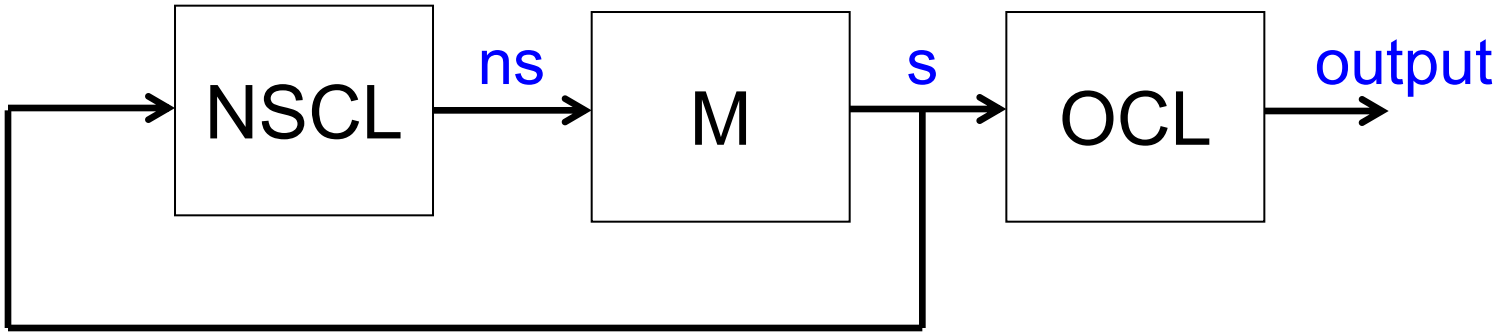


Rewrite



can sometimes omit clk

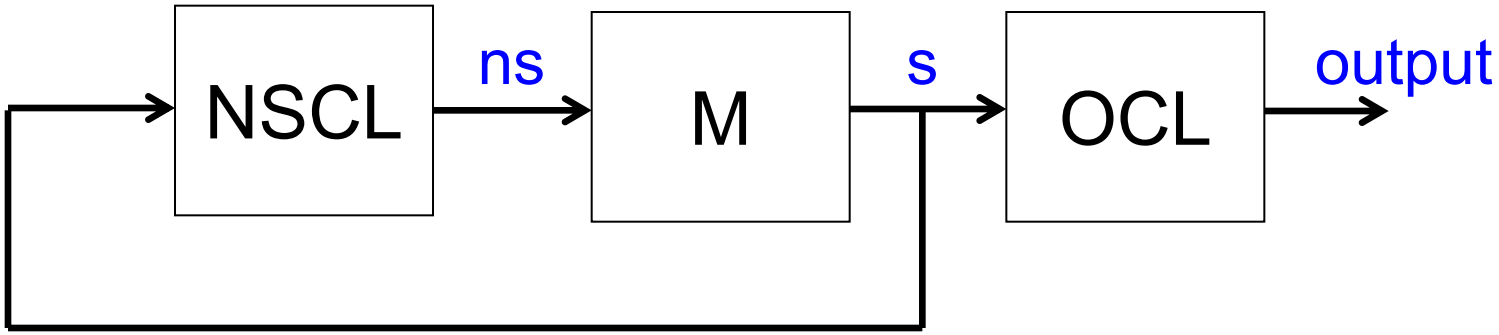
Back to Hardware:



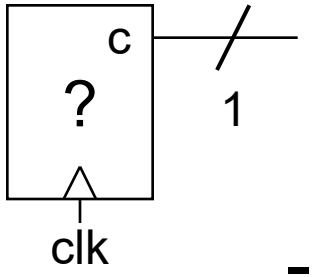
Hardware design:

- Have appropriate # of flops
- Design NSCL → ns = f(s)
- Design OCL → out = f(s)

Back to Hardware:



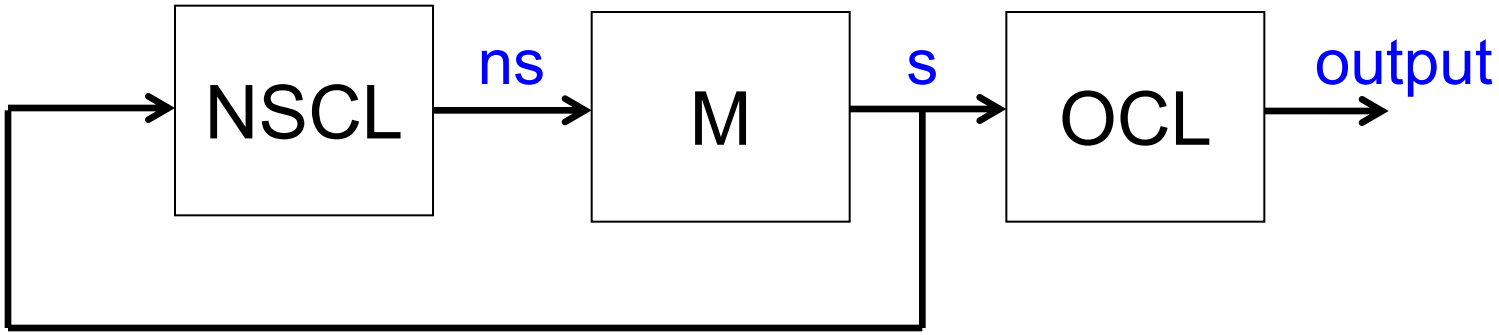
Design Example



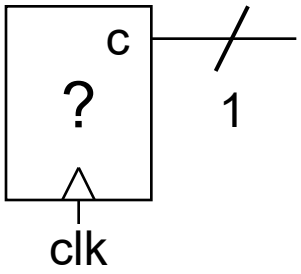
c: 0, 0, 0, 1, 0, 0, 0, 1, …

- What are the states?
- How many states?
- How many flip-flops?
- Draw the STD

Back to Hardware:

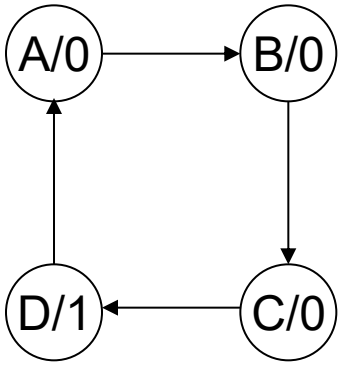


Design Example



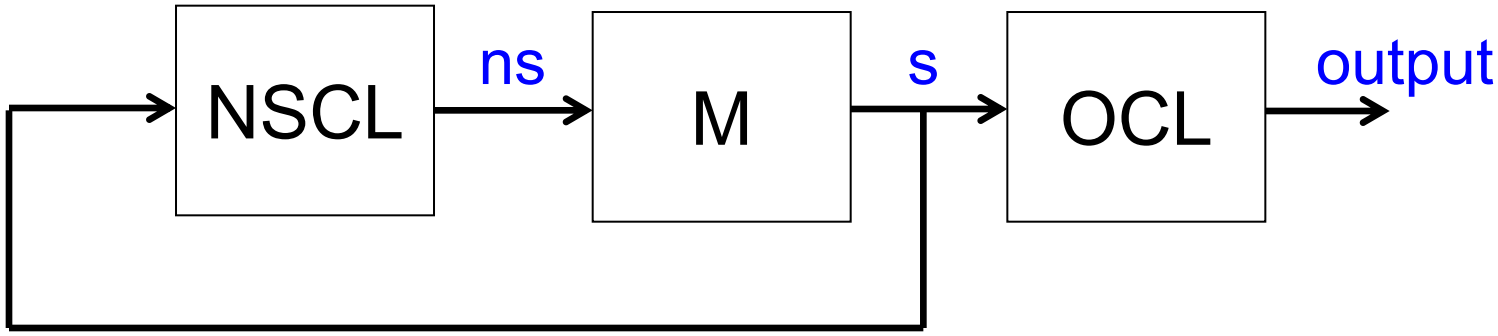
c: 0, 0, 0, 1, …

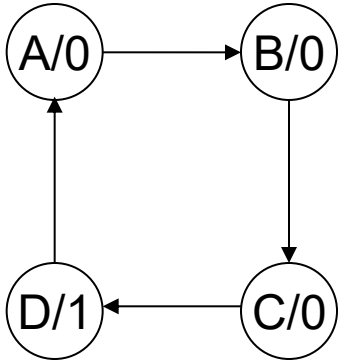
STD



- 4 states, 2 flip-flops

Back to Hardware:





Hardware has 0's & 1's

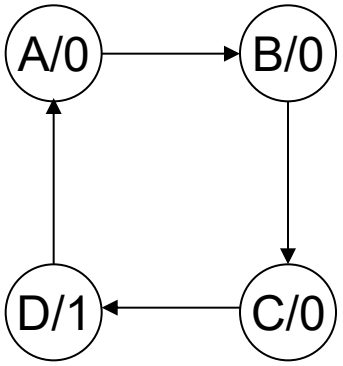
Hardware does not have A B C D (state names)

STATE ENCODING is Mapping states to 0's & 1's

Back to Hardware:

STATE ENCODING is mapping states to 0's & 1's

- minimal encoding = use as few bits as possible
  (4 states, 2 bits will be enough)

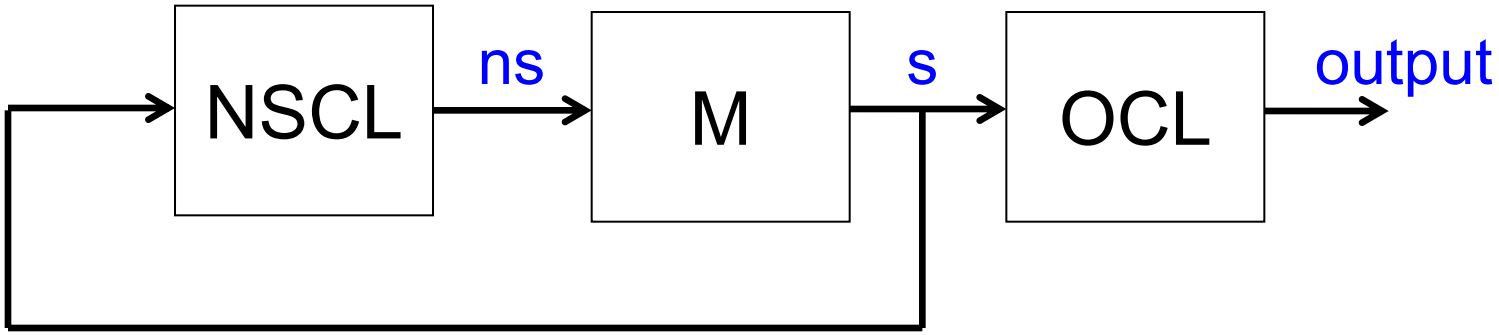


- one hot = use as many flops as states (4 states, 4 flops)

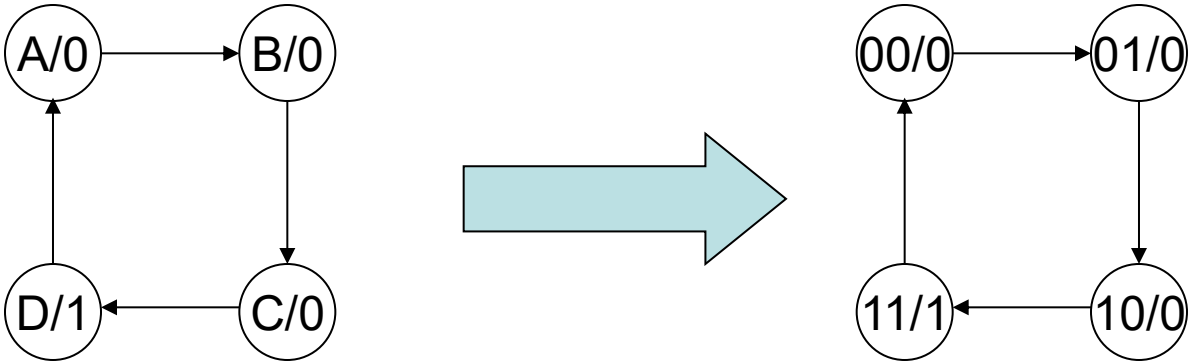
- other variations on one-hot

Using more flops sometimes make CL design faster
In this course, we will use minimal encoding

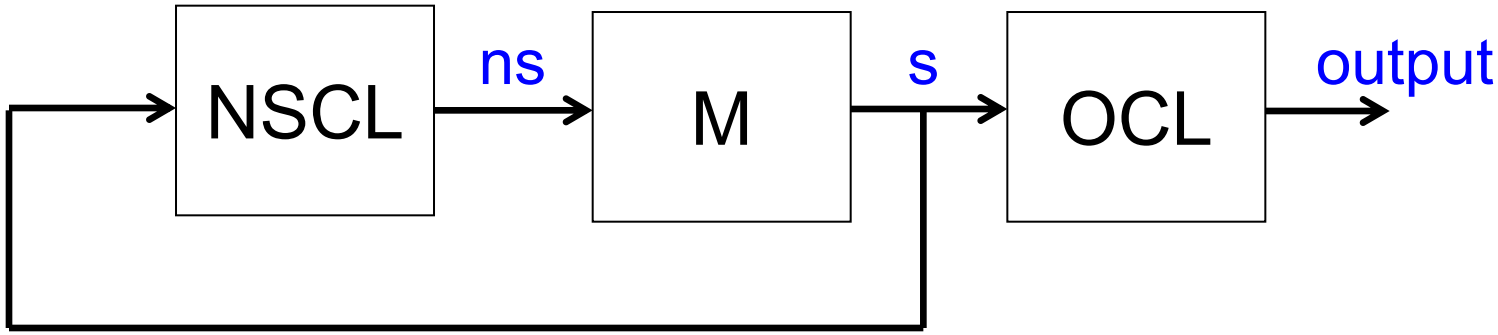
Back to Hardware:

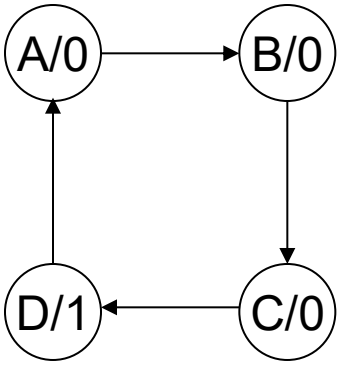


Minimal State Encoding
A = 00, B = 01, C = 10, D = 11



Back to Hardware:





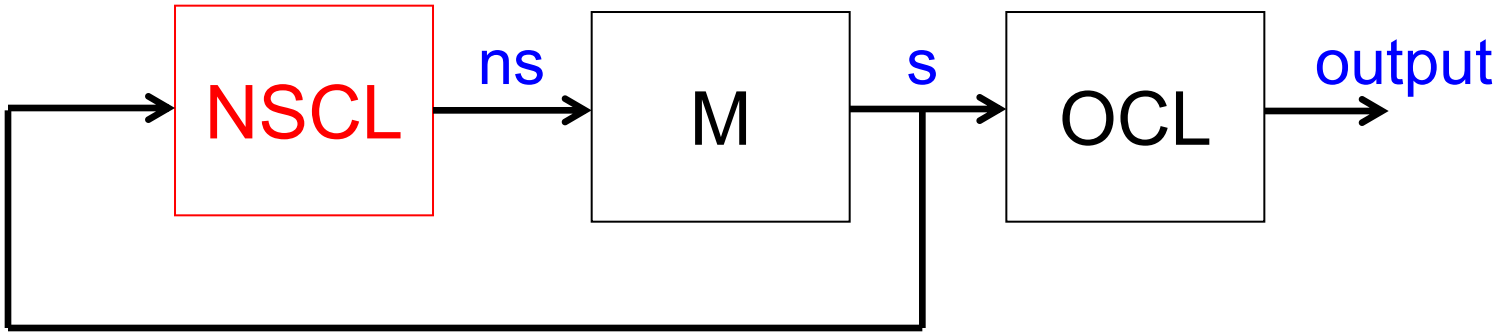
A = 00
B = 01
C = 10
D = 11

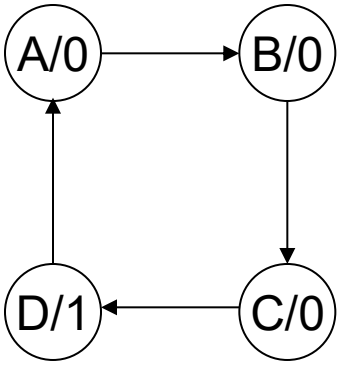
Back to design

ns = f(s)
out = f(s)

Map ns as a function of s
Map out as a function of s

Back to Hardware:



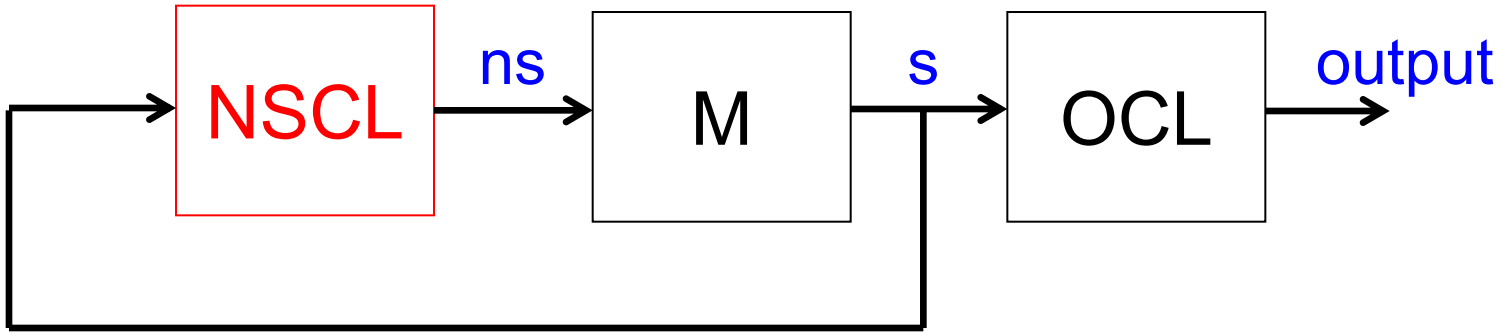


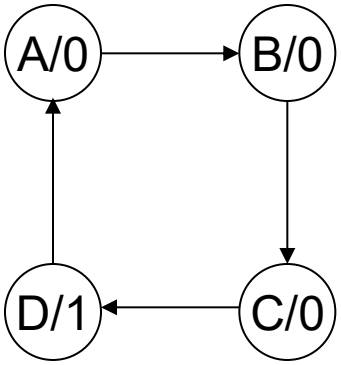
A = 00
B = 01
C = 10
D = 11

s	ns
A	B
B	C
C	D
D	A

s	ns
00	01
01	10
10	11
11	00

Back to Hardware:





A = 00
B = 01
C = 10
D = 11

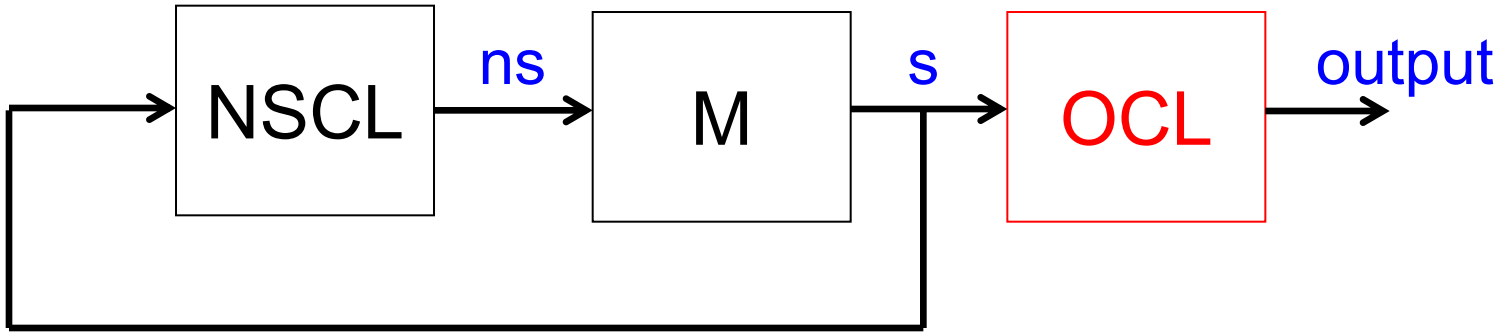
State Transition Table

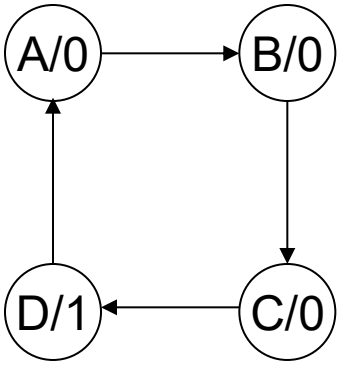
s	ns
A	B
B	C
C	D
D	A

s	ns
00	01
01	10
10	11
11	00

Remember ns = f(s)

Back to Hardware:





A = 00
B = 01
C = 10
D = 11

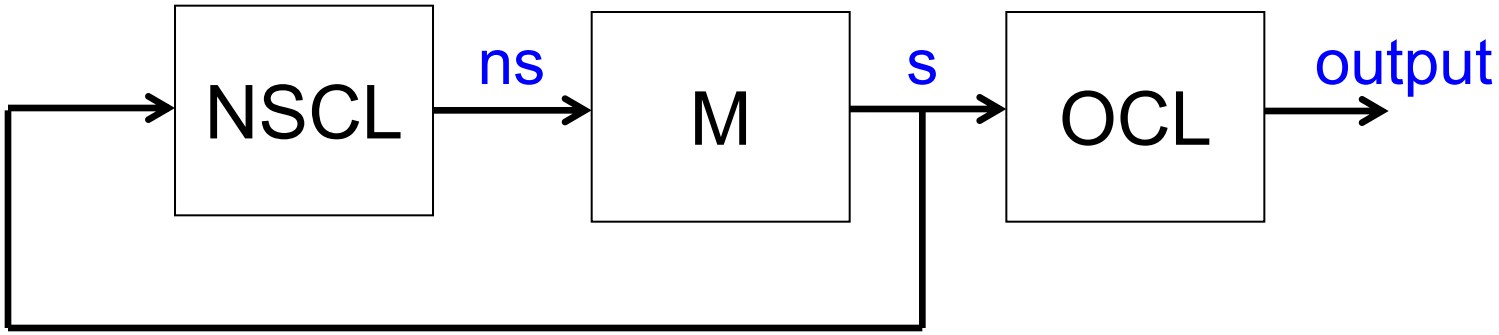
State Output Table

s	out
A	0
B	0
C	0
D	1

s	out
00	0
01	0
10	0
11	1

Remember out = f(s)

Back to Hardware:



State Transition

s	ns
00	01
01	10
10	11
11	00

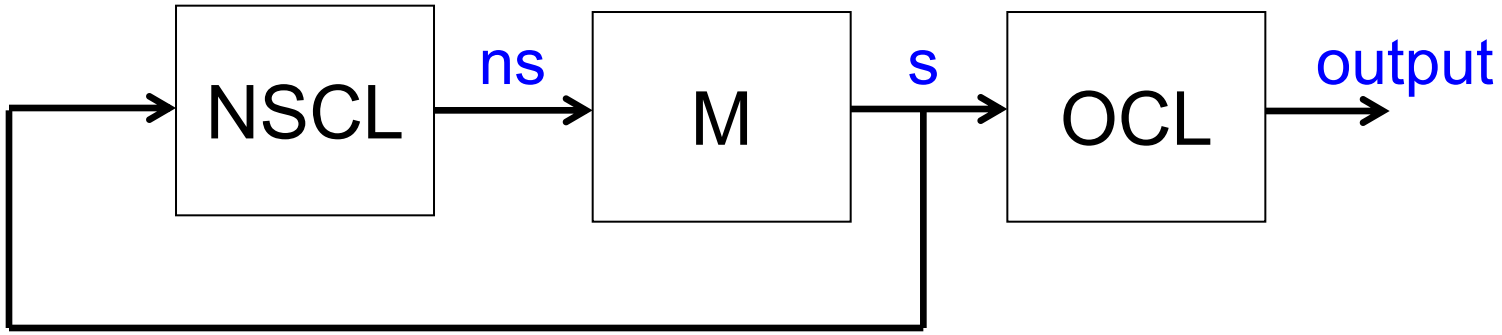
ns = f(s)

State Output

s	out
00	0
01	0
10	0
11	1

out = f(s)

Back to Hardware:



State Transition

s	ns
00	01
01	10
10	11
11	00

s_1s_0	ns_1ns_0
00	01
01	10
10	11
11	00

ns = f(s)

ns1 = f(s1, s0)

ns0 = f(s1, s0)

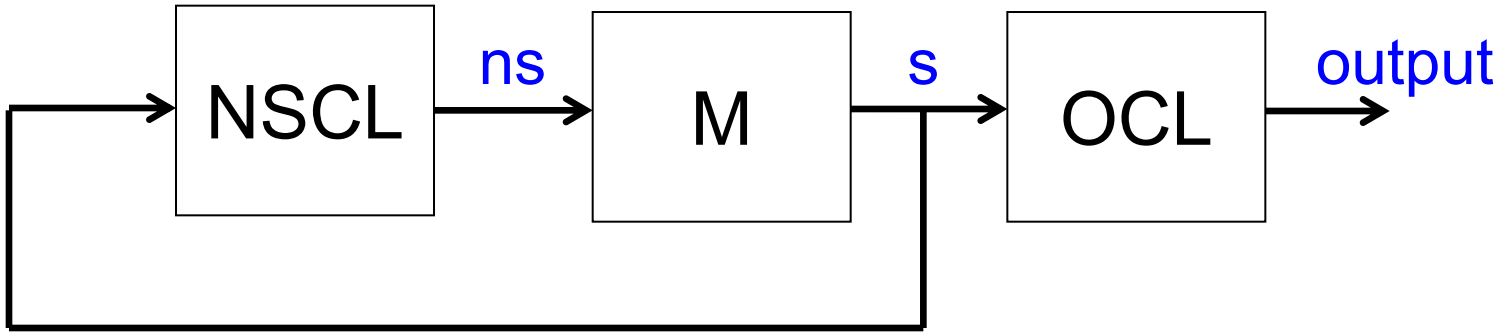
ns1

s_1 \ s_0	0	1
0	0	1
1	1	0

ns0

s_1 \	0	1
0	1	0
1	1	0

Back to Hardware:



State Transition

ns1

s_1 \ s_0	0	1
0	0	1
1	1	0

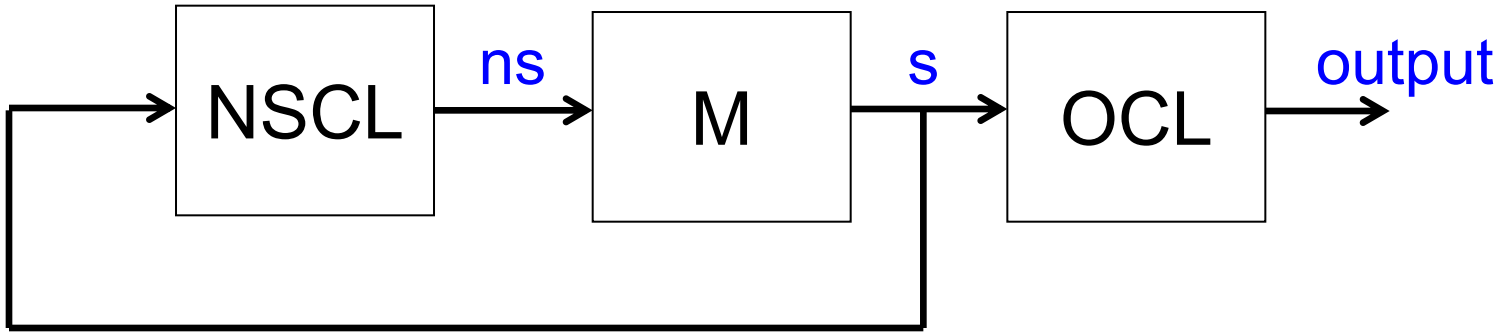
← ns1 = s0 ^ s1

ns0

s_1 \ s_0	0	1
0	1	0
1	1	0

← ns0 = s0’

Back to Hardware:



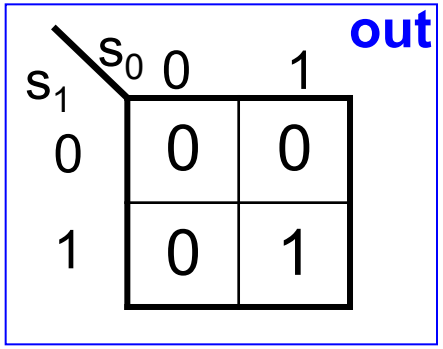
State Output

s	out
00	0
01	0
10	0
11	1

out = f(s)

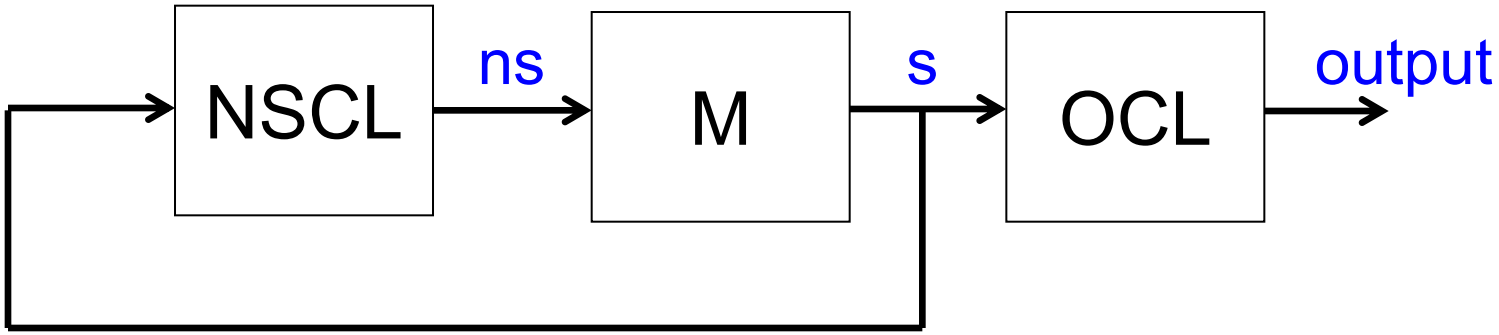
s1s0	out
00	0
01	0
10	0
11	1

out = f(s1,s0)



out = s1 s0

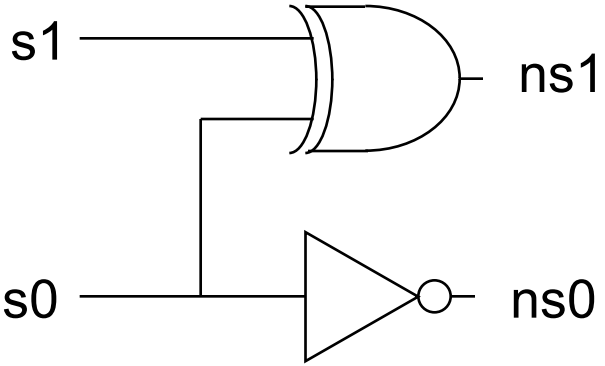
Back to Hardware:



State Transition

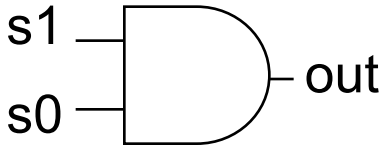
ns1 = s0 ^ s1

ns0 = s0’

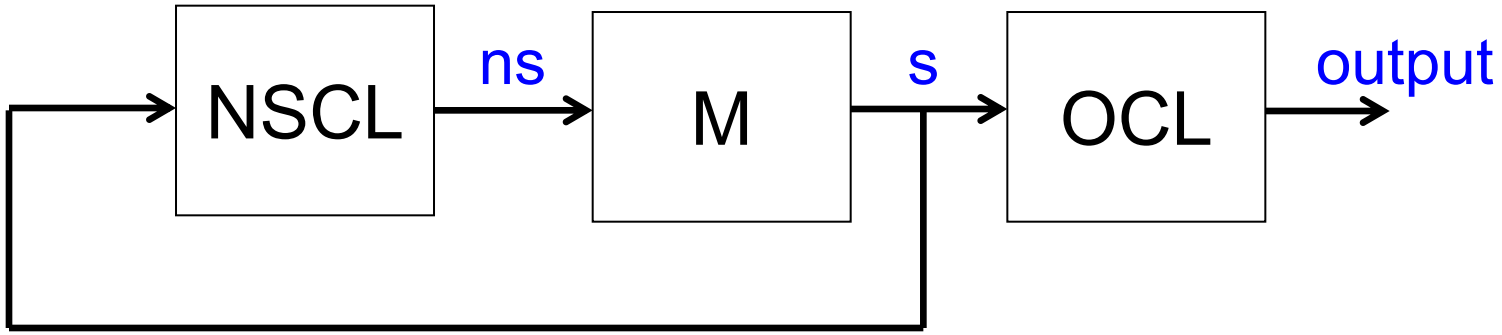


State Output

out = s0 s1



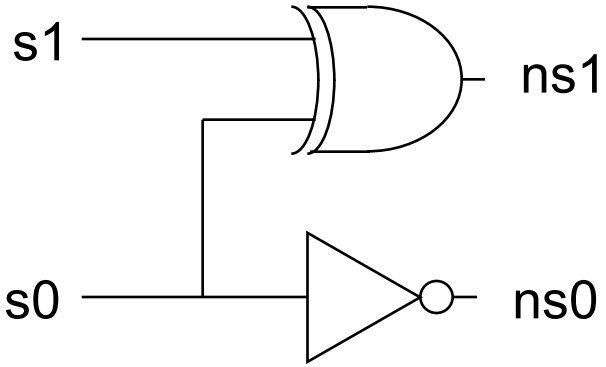
Back to Hardware:



State Transition

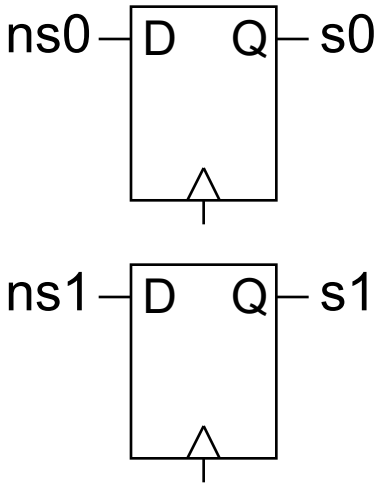
ns1 = s0 ^ s1

ns0 = s0’



NSCL

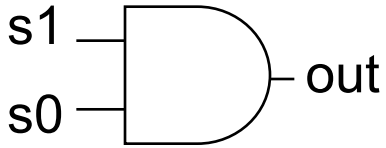
Memory



M

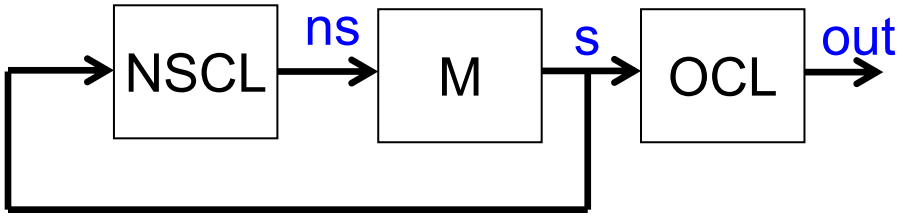
State Output

out = s0 s1



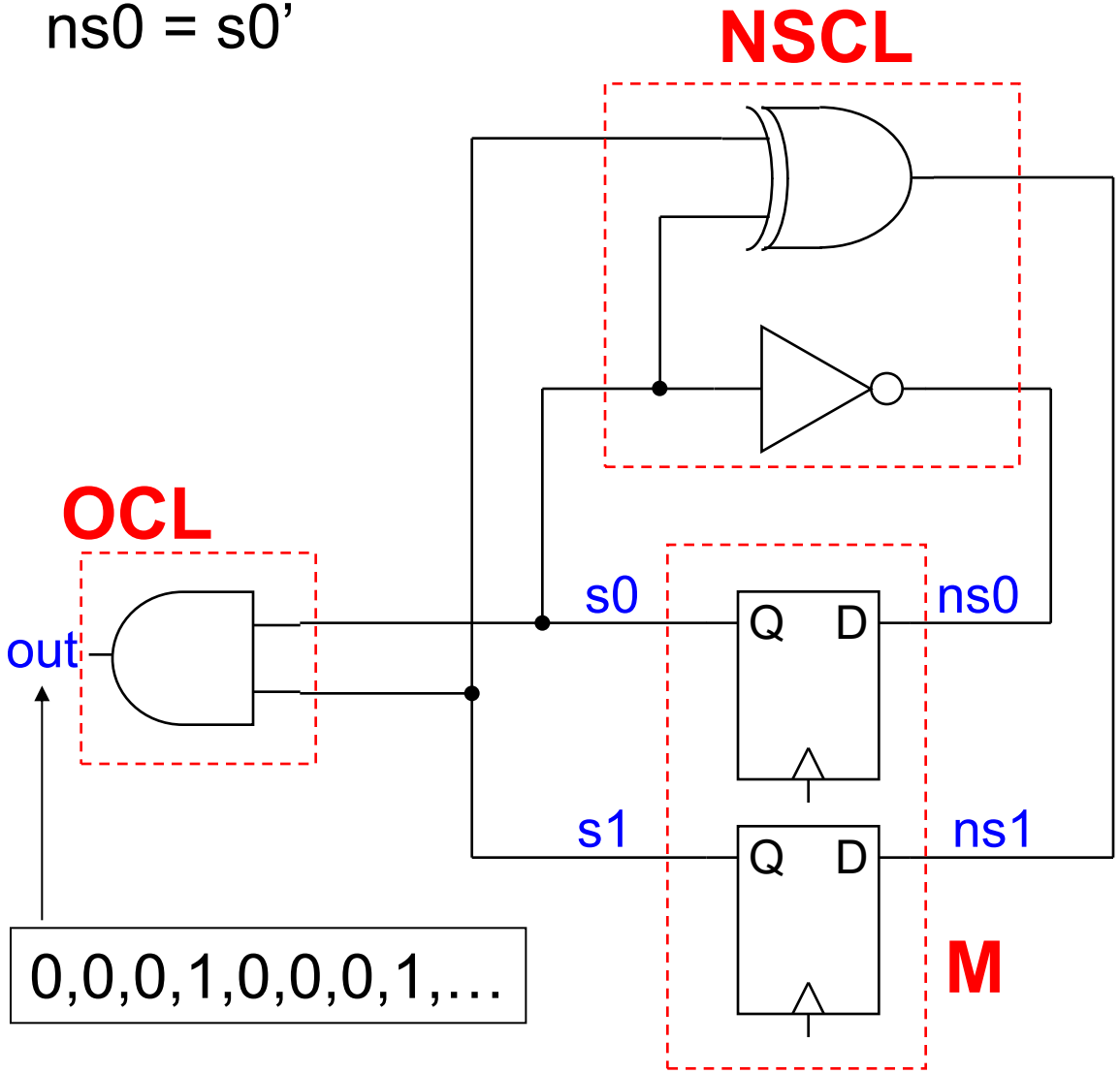
OCL

Putting it all together:



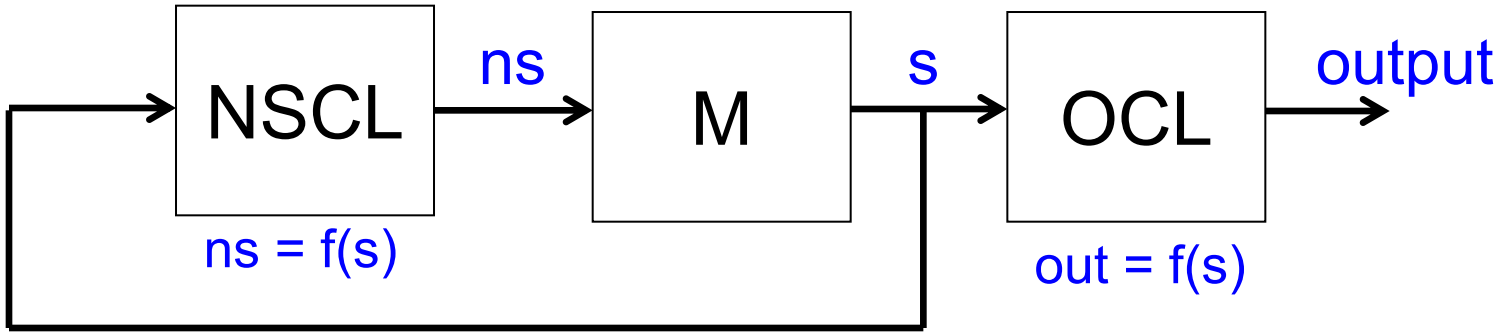
ns1 = s0 ^ s1 out = s0 s1

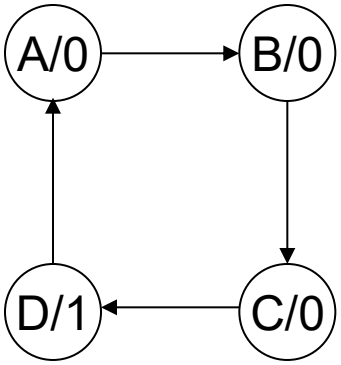
ns0 = s0'



for synchronous design (all flops running off the same clock), you can omit drawing clock signal

Back to Hardware (Verilog version):

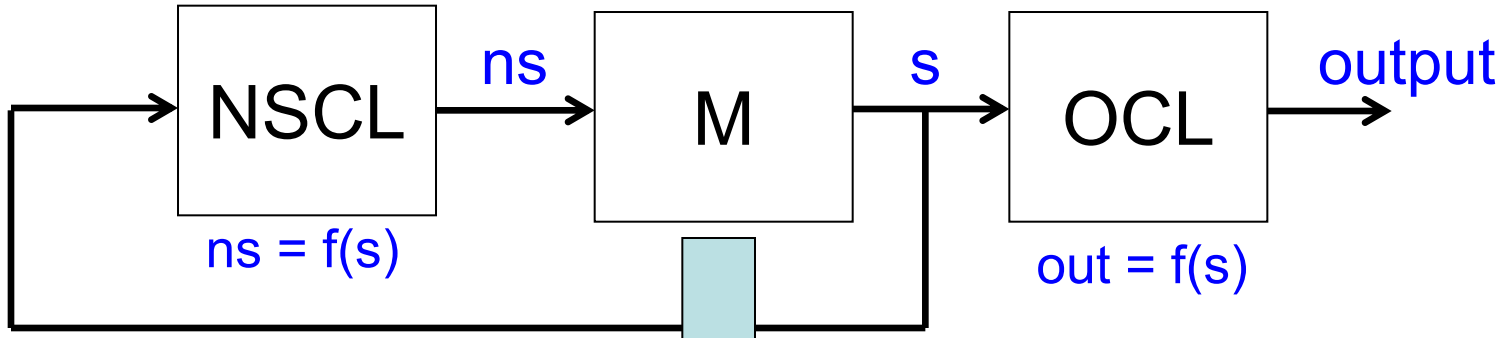


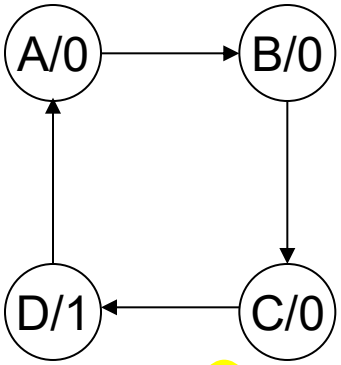


A = 00
B = 01
C = 10
D = 11

```
module fsm(clk, out);
     input clk;
     output [1:0] out;



     //
     // NSCL, M, and OCL design
     //

endmodule
```

Back to Hardware (Verilog version):



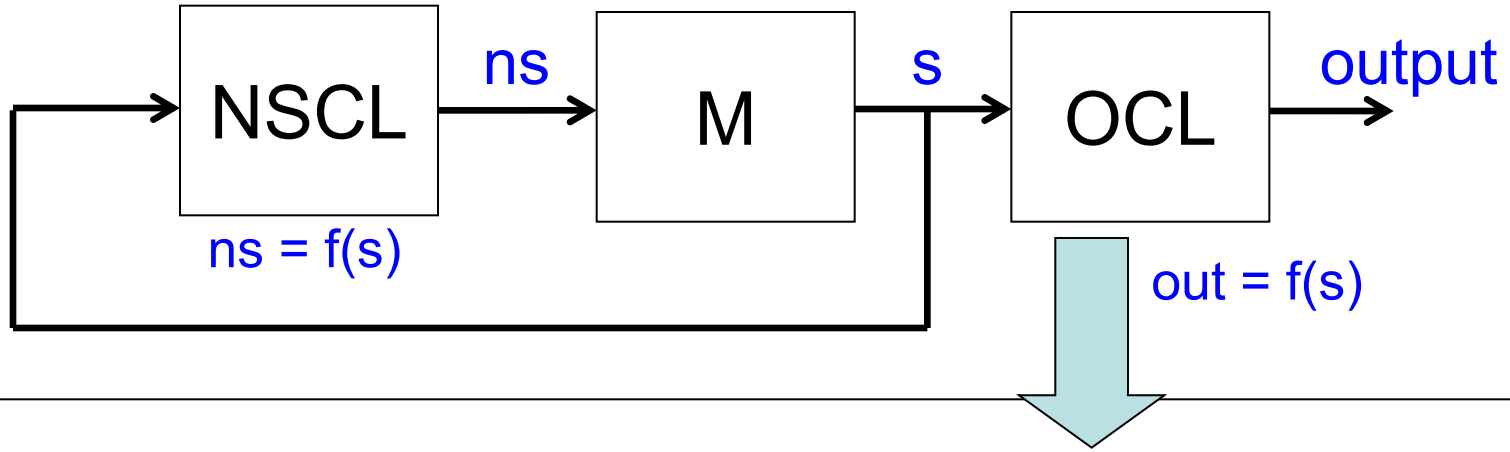


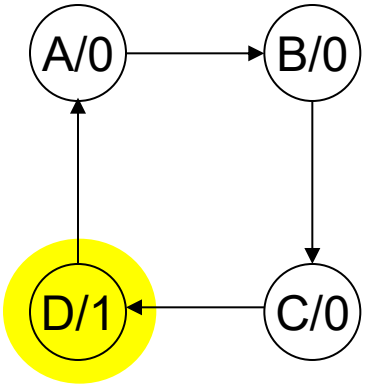
A = 00
B = 01
C = 10
D = 11

```
reg [1:0] s, ns;

always @(posedge clk)
    s <= #1 ns;
```

M design

Back to Hardware (Verilog version):



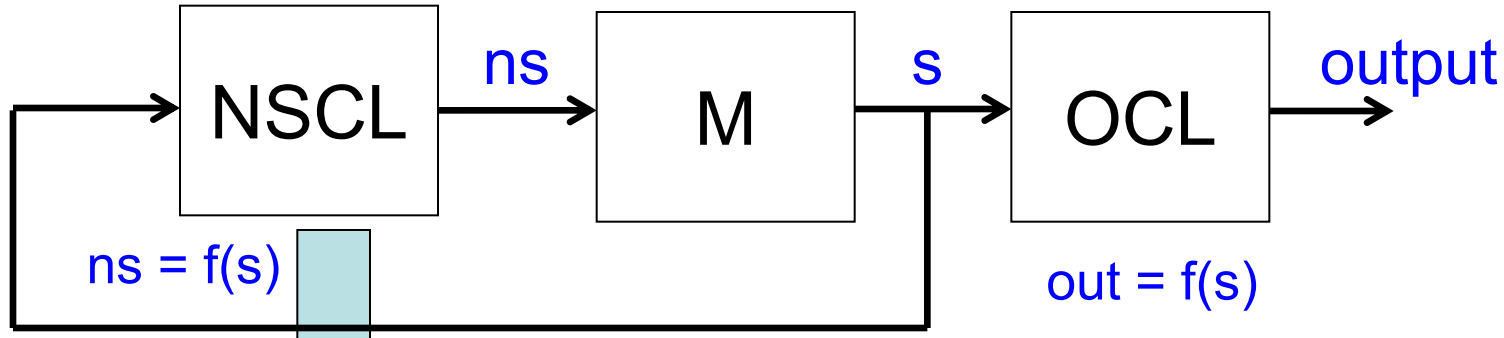


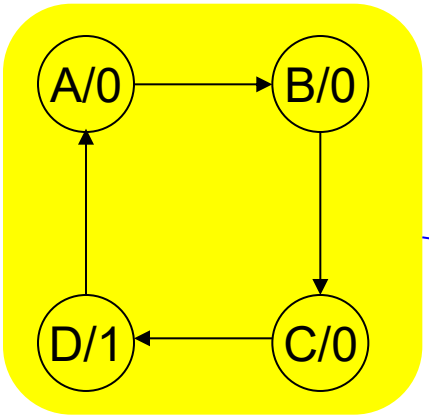
A = 00
B = 01
C = 10
D = 11

```
reg out;

always @(s)
    if (s == 2'b11)
        out = 1;
    else
        out = 0;
```

OCL design

Back to Hardware (Verilog version):



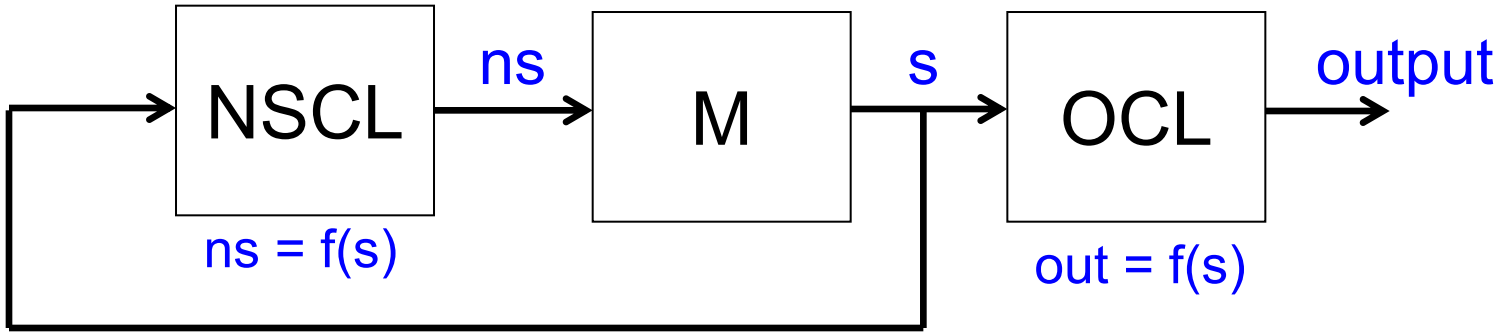


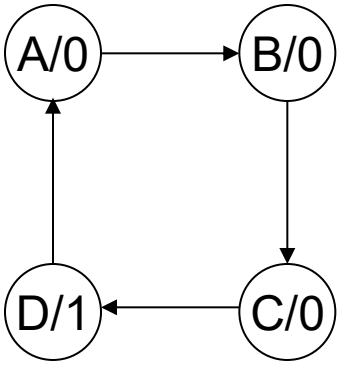
A = 00
B = 01
C = 10
D = 11

```
always @(s)
    case (s)
        2'b00: ns = 2'b01;
        2'b01: ns = 2'b10;
        2'b10: ns = 2'b11;
        2'b11: ns = 2'b00;
    endcase
```

NSCL design

Back to Hardware (Verilog version):





A = 00
B = 01
C = 10
D = 11

```
module fsm(clk, out);
    input clk;
    output [1:0] out;
    reg [1:0] s, ns;
    reg out;

    // M (Memory)
    always @(posedge clk)
        s <= #1 ns;

    // OCL (Output CL)
    always @(s)
        if (s == 2'b11)
                out = 1;
        else
                out = 0;
```

```
// NSCL (Next State CL)
always @(s)
    case (s)
        2'b00: ns = 2'b01;
        2'b01: ns = 2'b10;
        2'b10: ns = 2'b11;
        2'b11: ns = 2'b00;
    endcase

endmodule
```