EECS 151/251A Discussion 4

Alisha Menon

9/20/21, 9/21/21, 9/23/21

Administrativia

- Homework 3 is out – due Monday **9/27, 12:00am**
- Homework 4 out this week.

Agenda

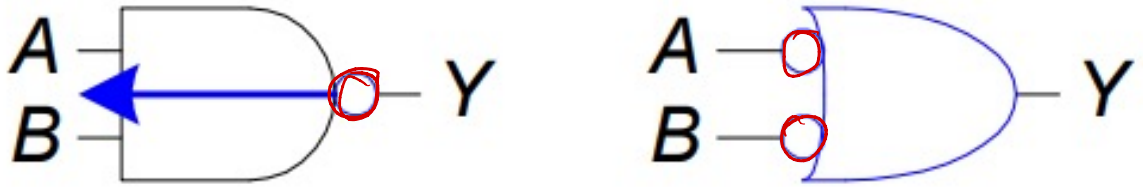
- DeMorgan's Law
  - Bubble pushing
- Karnaugh maps
  - POS
  - SOP
- Finite state machines

DeMorgan's Law: Bubble Pushing

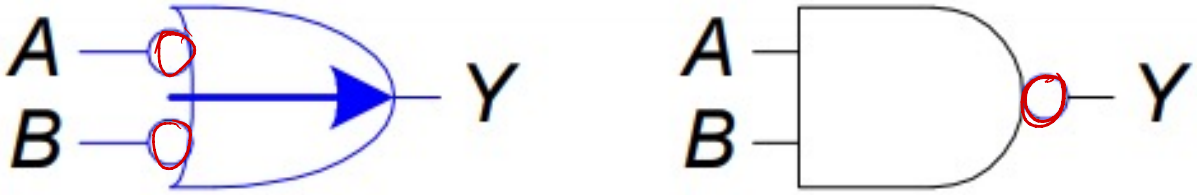
- (x+y)' = x'y'
- (xy)' = x'+y'
- Bubble = inversion (NOT)

DeMorgan's Law: Bubble Pushing

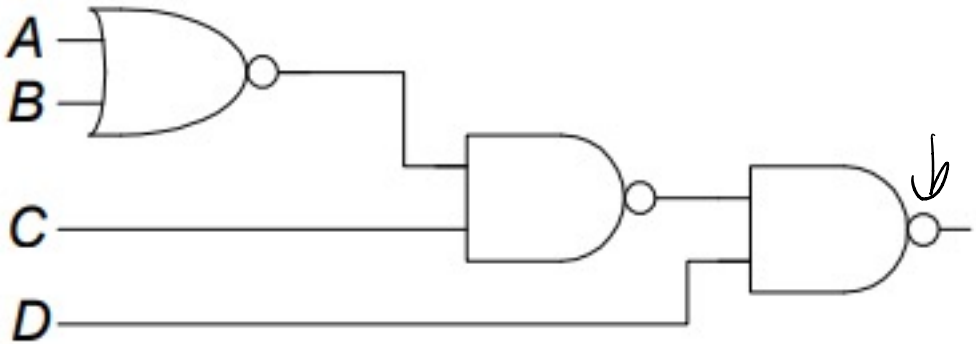
- $(x+y)' = x'y'$
- $(xy)' = x'+y'$
- Bubble = inversion (NOT)
- For a single gate:
  - Swap AND for OR & vice versa
  - Backward pushing: add bubbles to input



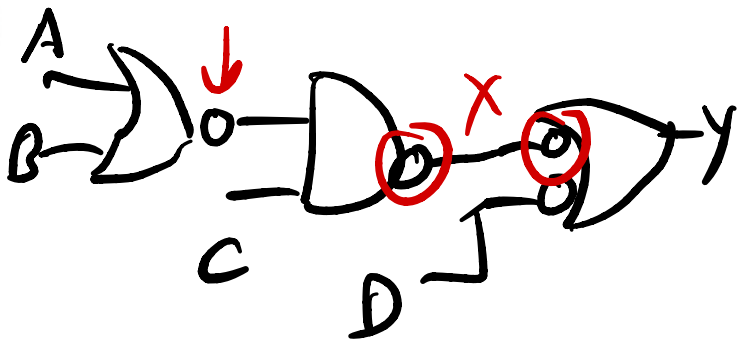
  - Forward pushing: add bubbles to output

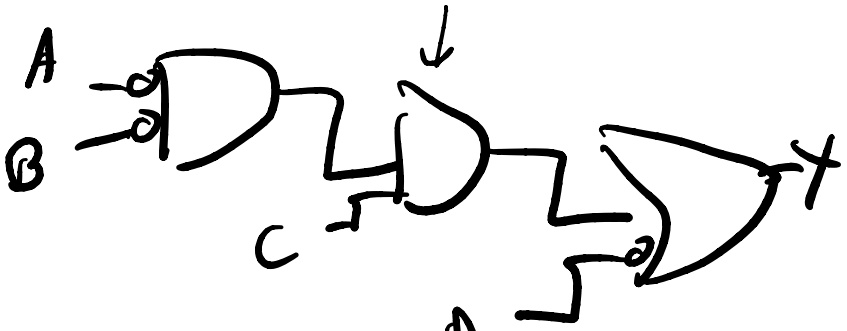


Bubble Pushing Example



$$Y = \overline{(A+B)} \cdot C \cdot D$$





$$Y = \bar{A}\bar{B}C + \bar{D}$$

SoP & PoS

A	B	C	Out
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

SoP

$out = \bar{A}\bar{B}C + A\bar{B}\bar{C} + \dots$

PoS

$out = (A+B+C)(A+\bar{B}+\bar{C})(\bar{A}+B+C)$

K-maps

- K-Maps: visual & systematic Boolean simplification
- 2 important Boolean identities:
  - **(1+A)=1**
  - **(A+$\bar{A}$)=1**
- Leverages **gray coding** to organize neighboring minterms
  - Adjacent minterms only differ by a single bit!
- Key to solving: form groups of 1's by ~~multiples~~ of 2
  - As large & as few as possible
  - Overlapping is OK, wrap boundary where possible
  - Write AND expression for each group
  - Make new SoP expression

K-map example

1, 2, 4, 8, 16...

→ **F(A,B) = $\bar{A}B + \bar{A}$**

$1+B=1$, $A+\bar{A}=1$

$Y = A + \bar{A}B$

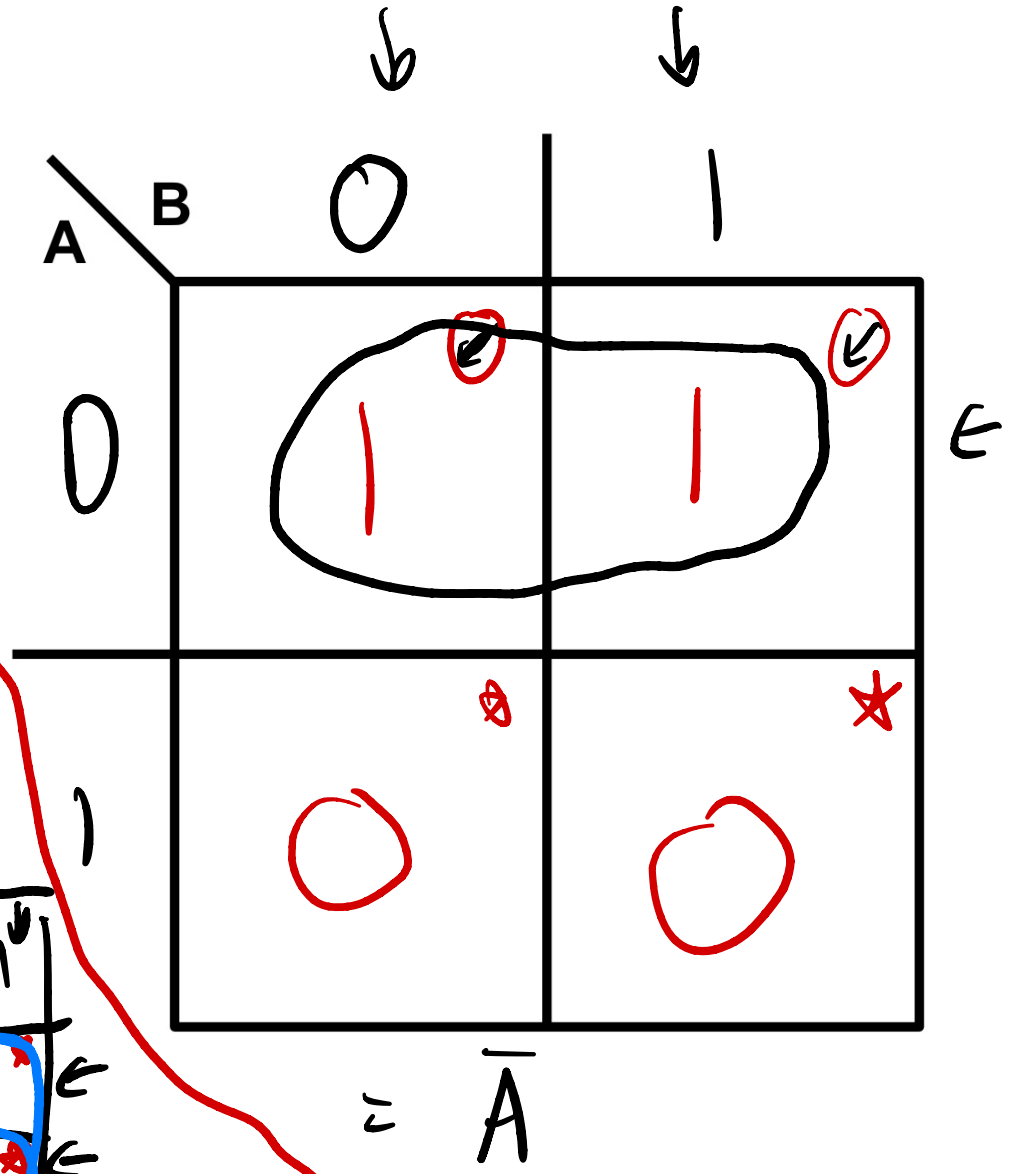
$= A(1+B) + \bar{A}B$

$= A + AB + \bar{A}B$

$= A + B(A+\bar{A})$

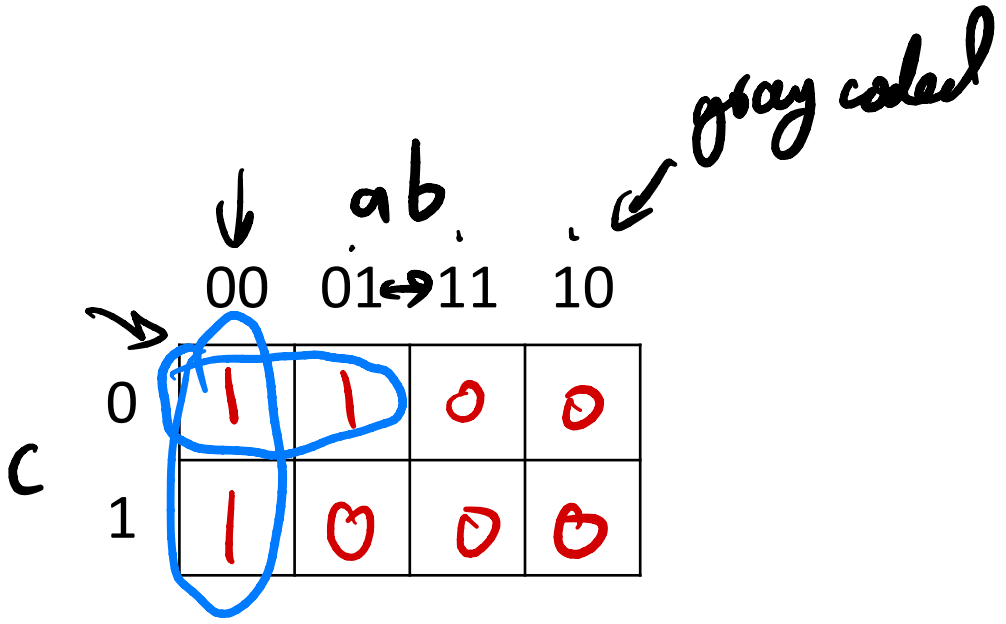
$= A + B$

$\simeq A + B$



Simplification – Karnaugh maps (SOP)

a	b	c	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

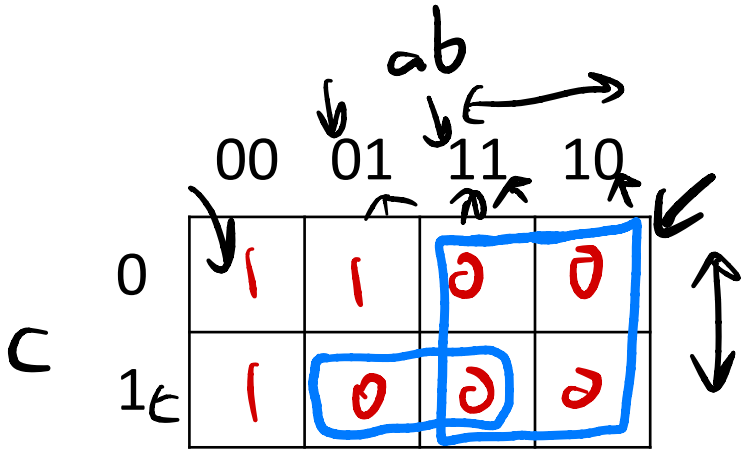


$$f = a'b' + a'c'$$

$$= a'(b'+c')$$

Simplification – Karnaugh maps (POS)

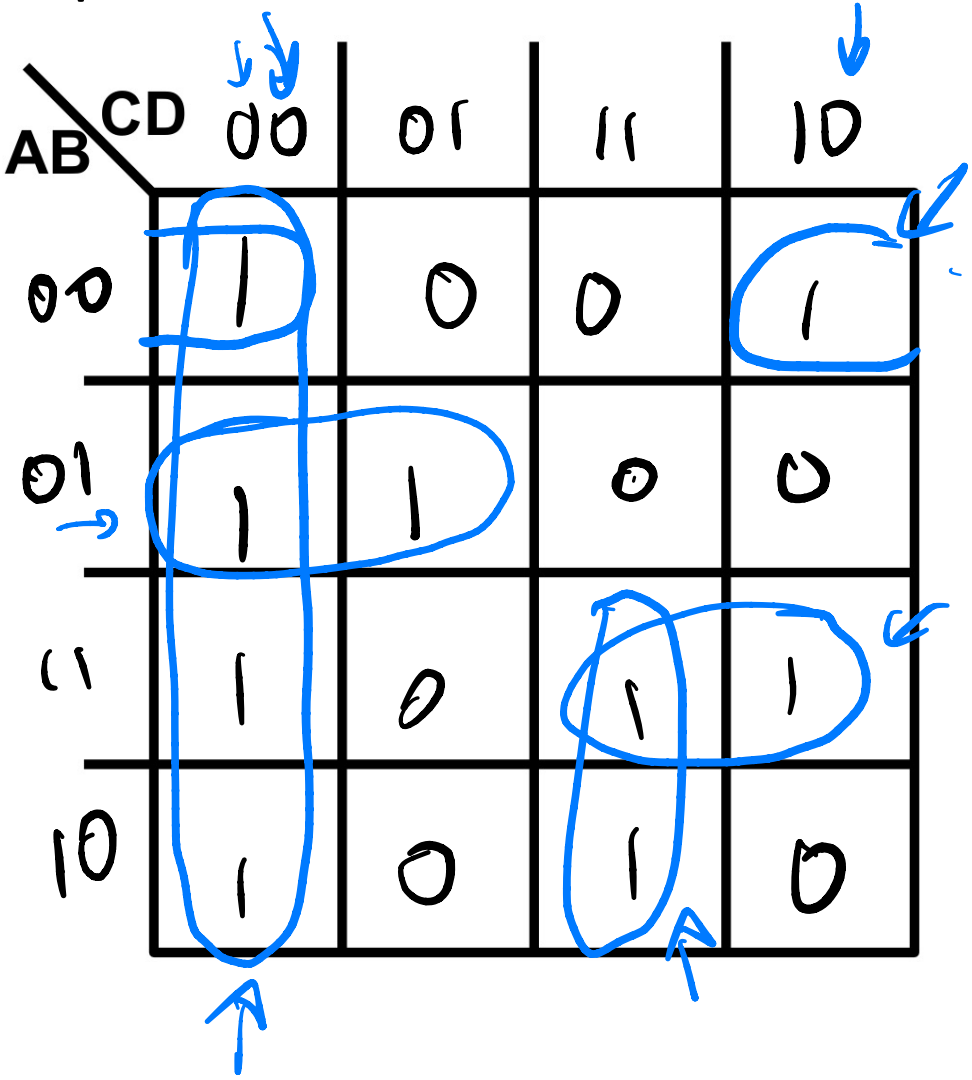
a	b	c	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



$f = (a')(c' + b')$

4-input K-map example

$$F(A,B) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + ABCD$$



$$F(A,B) = \bar{C}\bar{D} + A\bar{B}\bar{C} + CAB + CDA + \bar{A}\bar{B}\bar{D}$$

Finite state machines

FSM review

- Sequential circuit where output depends on **present and past inputs**
- Has finite number of states, and can only be in one state at a time
- Combinational logic used to calculate next state and output
- Represented by state transition diagram

Moore vs. Mealy FSM

	Moore	**Mealy**
Output function	based only on present state	based on both present state and inputs
# states	usually more	fewer
Output synchronous	synchronous	asynchronous (can glitch w/ inputs)
Output delay	delayed by one clock cycles	immediately available w/ input

Example - Vending Machine FSM

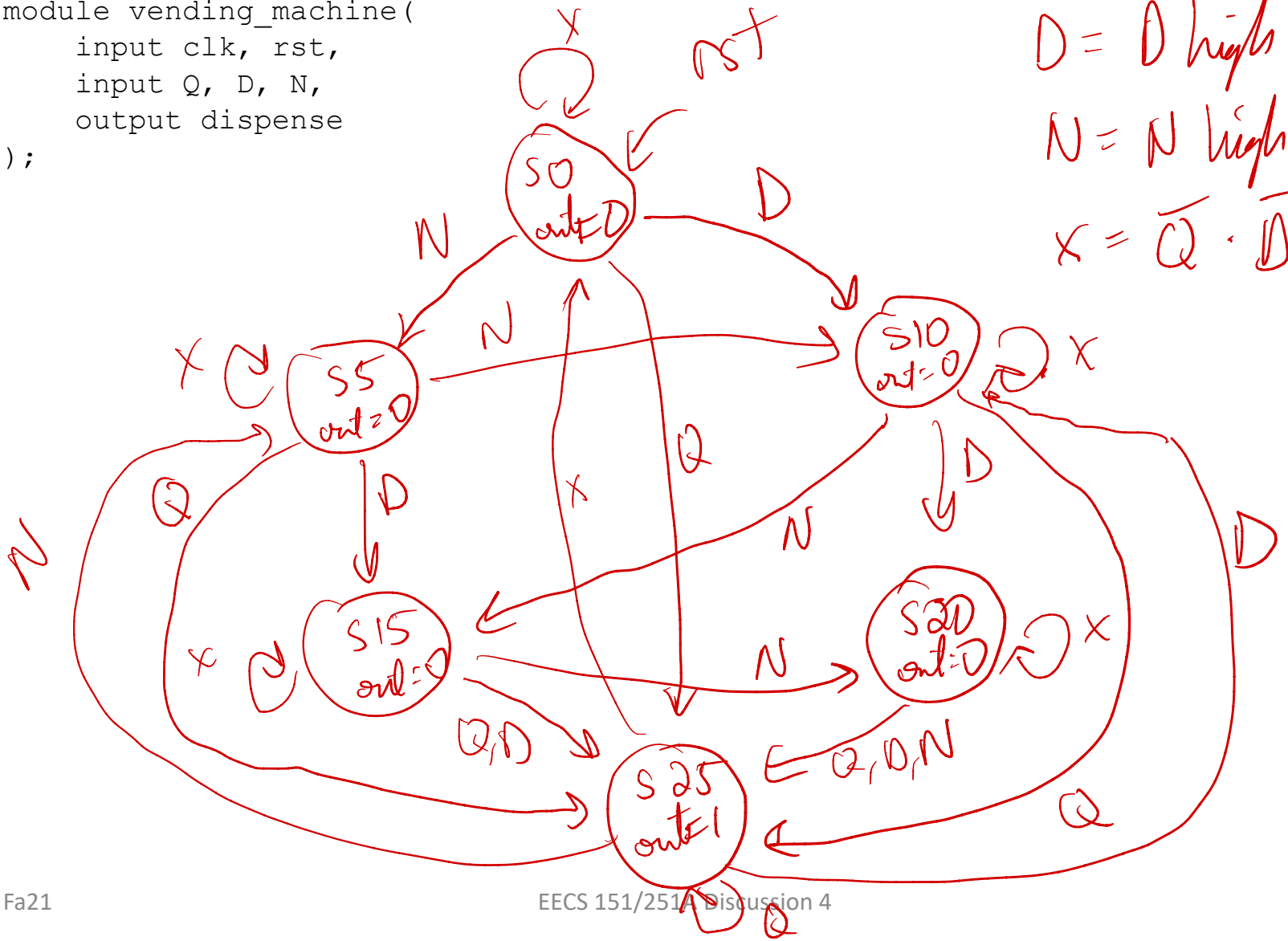
- Dispenses a soda if it receives at least 25 cents
  - Doesn't return change or rollover to next purchase
- Customer can insert three different coins:
  - Quarter – 25¢ – Q
  - Dime – 10¢ – D
  - Nickel – 5¢ – N

} 3 inputs signals that pulse to 1 when a coin is inserted, only 1 signal is high at a time

```
module vending_machine(
  inputs clk, rst,
  input Q, D, N
  output dispense);
```

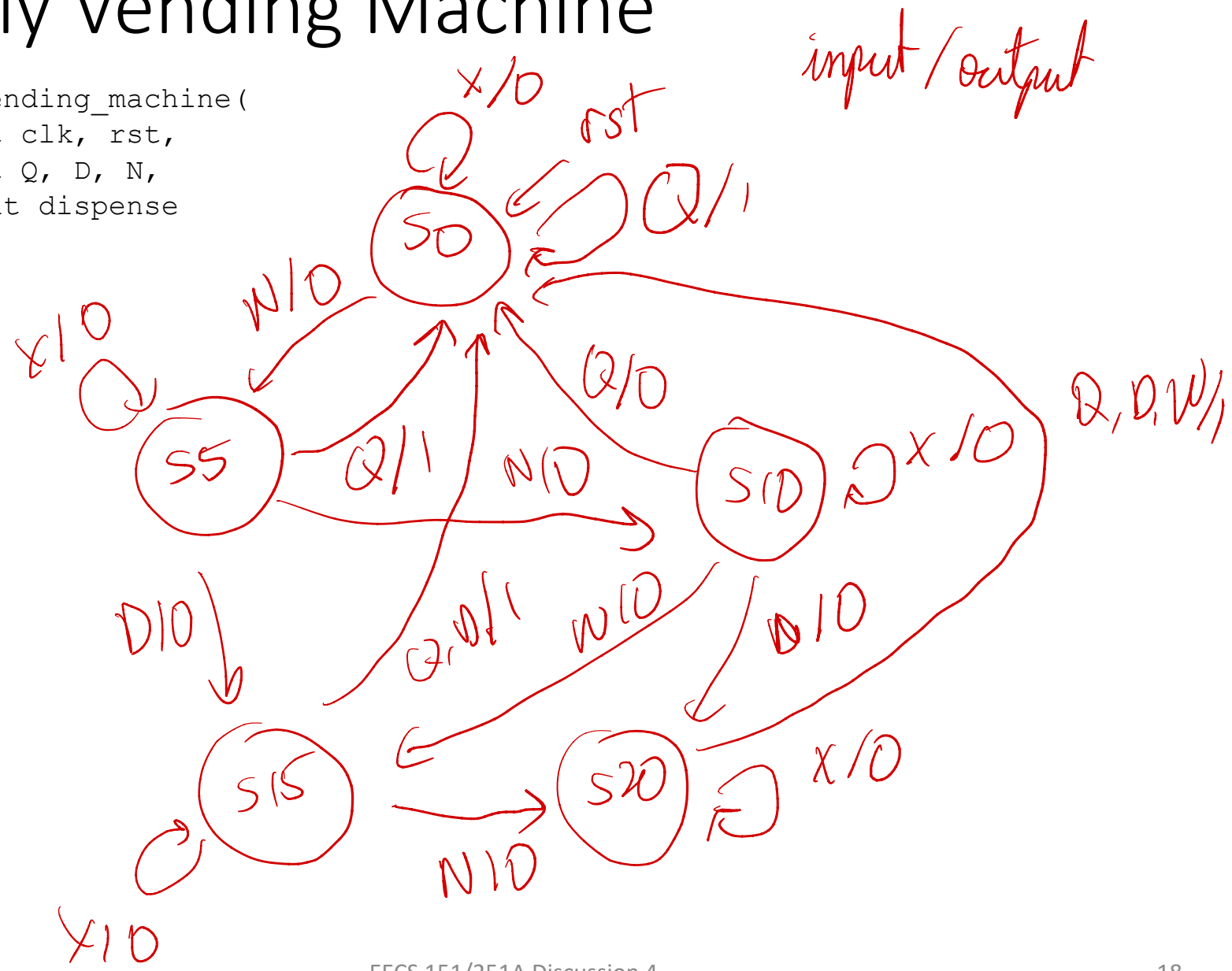
Moore Vending Machine

```
module vending_machine(
    input clk, rst,
    input Q, D, N,
    output dispense
);
```



Mealy Vending Machine

```
module vending_machine(
    input clk, rst,
    input Q, D, N,
    output dispense
);
```



Verilog Implementation

- Two main sections:
  - State transition (sequential)
  - State/output logic (combinational)

```
module vending_machine()
// inputs, outputs, clk, rst
// define state bits
// define state names as local params
// state transitions
// next state and output logic
endmodule
```

sequential

combo

Setup and state transitions

Moore and Mealy

```
module vending_machine(
    input clk, rst,
    input Q, D, N,
    output dispense
);

reg [2:0] NS, CS;

localparam S0 = 3'd0,
           S5 = 3'd1,
           S10 = 3'd2,
           S15 = 3'd3,
           S20 = 3'd4,
           S25 = 3'd5;

always @(posedge clk) begin
    if (rst) CS <= S0;
    else CS <= NS;
end

...
```

← module I/O set

← state register bits

← shorthand for readability

← state transition

Moore vs. Mealy combinational logic

```
always @(*) begin
    NS = CS;
    case (CS)
        S0: begin
            if (Q == 1'b1) NS = S25;
            if (D == 1'b1) NS = S10;
            if (N == 1'b1) NS = S5;
        end
        S5: begin
            if (Q == 1'b1) NS = S25;
            if (D == 1'b1) NS = S15;
            if (N == 1'b1) NS = S10;
        end
        ...
        S25: begin
            if (Q == 1'b1) NS = S25;
            if (D == 1'b1) NS = S10;
            if (N == 1'b1) NS = S5;
        end
        default: NS = S0;
    endcase
end

assign dispense = (CS == S25);

endmodule
```

```
reg dispense;
always @(*) begin
    NS = CS;
    dispense = 1'b0;
    case (CS)
        S0: begin
            if (Q == 1'b1) begin
                NS = S0;
                dispense = 1'b1;
            end
            if (D == 1'b1) NS = S10;
            if (N == 1'b1) NS = S5;
        end
        ...
        S15: begin
            if (Q == 1'b1) begin
                NS = S0;
                dispense = 1'b1;
            end
            if (D == 1'b1) begin
                NS = S0;
                dispense = 1'b1;
            end
            if (N == 1'b1) NS = S10;
        end
        ...
        default: NS = S0;
    endcase
end
endmodule
```