EECS 151/251A Discussion 3

02/09/2018

Agenda

- Announcements
- FSM
- Karnaugh Maps
- CMOS logic

Announcements

- Midterm next Thursday
  - 3 hour exam (though we don't expect you'll need the entire time)
  - In the lecture slot next Thursday with extra time; 5 pm - 8 pm
  - Closed book, but you are allowed one hand-written two-sided cheat sheet (8.5x11 inch)
  - Location to-be-decided
- I'll be away for two weeks
  - Only an email away. Or an SMS. Or Piazza. Or Skype. It's 2018, c'mon
  - Taehwan will attend Thursday FPGA labs
  - No one will staff Wednesday labs; I can be available in real-time on Slack
  - FPGA labs 3, 4 and 5 will be not need to be checked off until Friday, 2 March
  - Discussions will be moved temporarily so Taehwan can staff 'em
    - This is the last one before your first midterm

Warmup

1. Use blocking assignments to model combinational logic within an always block (“=”). Why?
2. Use non-blocking assignments to implement sequential logic (“<=”). Why?
3. Do not mix blocking and non-blocking assignments in the same always block.
4. Do not make assignments to the same variable from more than one always block.

Be rigorous and disciplined to minimise unexpected results!

Finite State Machines

Moore vs Mealy?

Traffic lights!

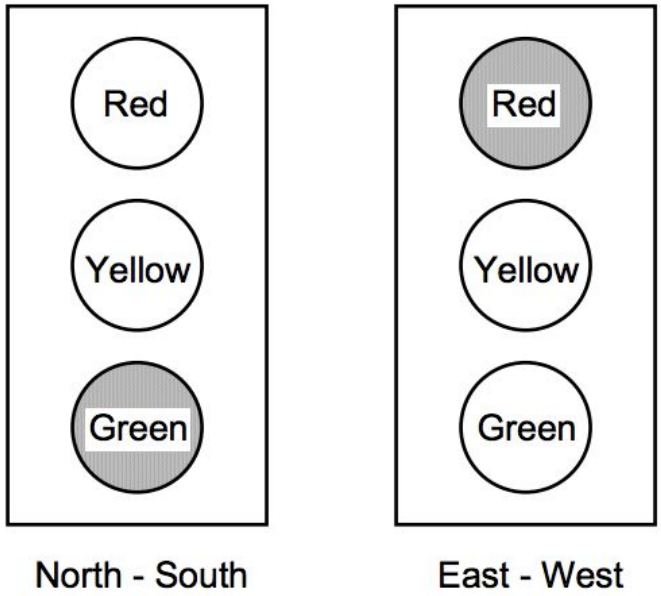


Figure 8.13 Six colored LEDs can represent a set of traffic lights

Traffic Light FSM

Table 8.2 Traffic Light States

State	North - South	East - West	Delay (sec.)
0	Green	Red	5
1	Yellow	Red	1
2	Red	Red	1
3	Red	Green	5
4	Red	Yellow	1
5	Red	Red	1

Traffic Light FSM

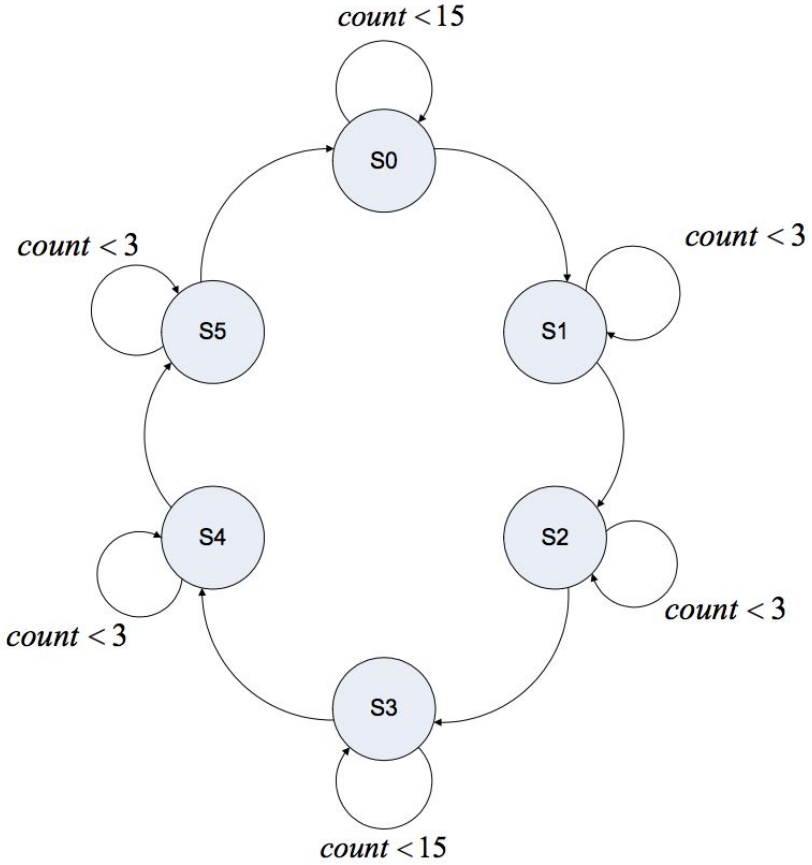


Figure 8.14 State diagram for controlling traffic lights

FSM design steps

1. ~~Specify/convince yourself of circuit function (natural language, e.g. English)~~
2. ~~Draw state transition diagram~~
3. ~~Write down symbolic~~ state transition table [kind of]
4. Assign encodings (bit patterns) to symbolic states
5. Code as Verilog behavioral description.. (Or just skip from 1 to 4+5...)

Listing 8.6 traffic.v

```
// Example 62a: traffic lights
module traffic (
input wire clk ,
input wire clr ,
output reg [5:0] lights
);
reg[2:0] state;
reg[3:0] count;

parameter  S0 = 3'b000, S1 =3'b001, S2 = 3'b010, // states
           S3 = 3'b011, S4 = 3'b100, S5 = 3'b101;
parameter  SEC5 = 4'b1111, SEC1 = 4'b0011;       // delays

always @(posedge clk or posedge clr)
  begin
      if (clr == 1)
        begin
            state <= S0;
            count <= 0;
        end
      else
        case(state)
```

```
always @(posedge clk or posedge clr)
  begin
      if (clr == 1)
        begin
            state <= S
            count <= 0
        end
      else
        case(state)
```

```
case(state)
    S0: if(count < SEC5)
            begin
              state <= S0;
              count <= count + 1;
            end
          else
            begin
              state <= S1;
              count <= 0;
            end
    S1: if(count < SEC1)
            begin
              state <= S1;
              count <= count + 1;
            end
          else
            begin
              state <= S2;
              count <= 0;
            end
    S2: if(count < SEC1)
            begin
              state <= S2;
              count <= count + 1;
            end
          else
            begin
```

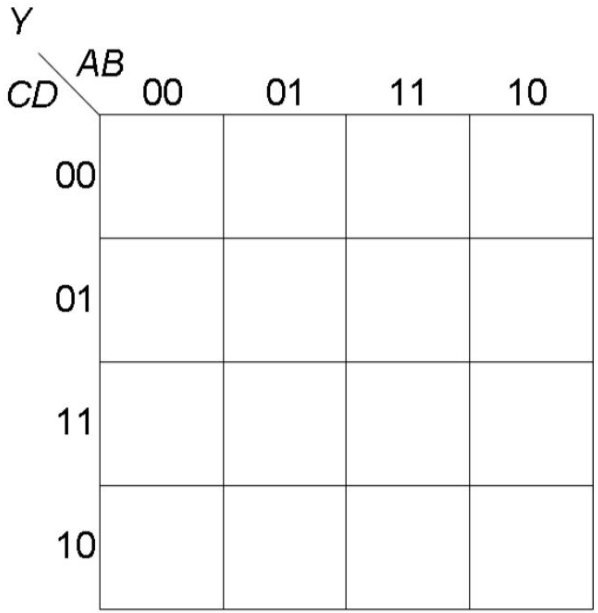
```
 always @(*)
   begin
       case(state)
              S0: lights = 6'b100001;
              S1: lights = 6'b100010;
              S2: lights = 6'b100100;
              S3: lights = 6'b001100;
              S4: lights = 6'b010100;
              S5: lights = 6'b100100;
              default lights = 6'b100001;
     endcase
  end
endmodule
```

Waiting for a green at night on empty streets

What about a sensor for cars waiting in each direction?

Karnaugh Maps

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Karnaugh Maps

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Y: CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

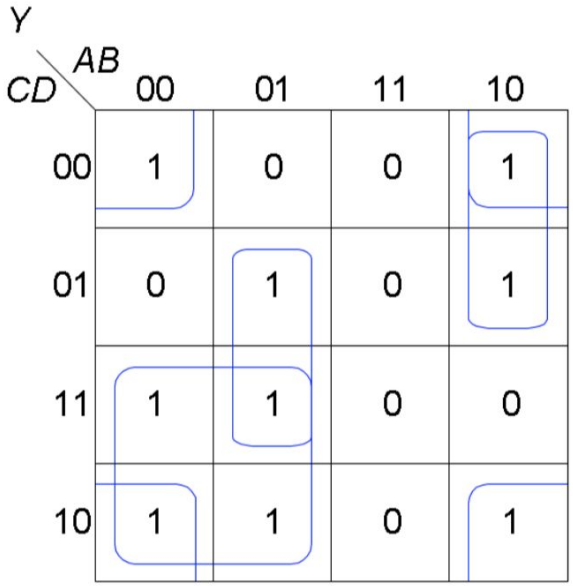
Karnaugh Maps

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Y: CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

Karnaugh Maps

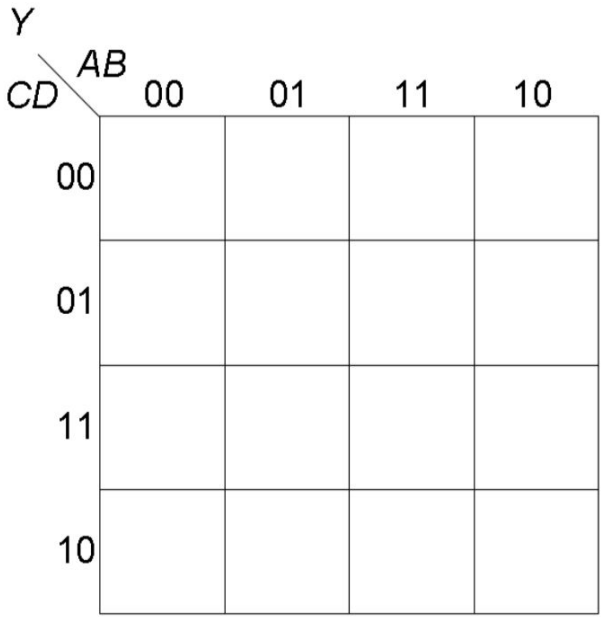
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Sum of products?

More Karnaugh Maps

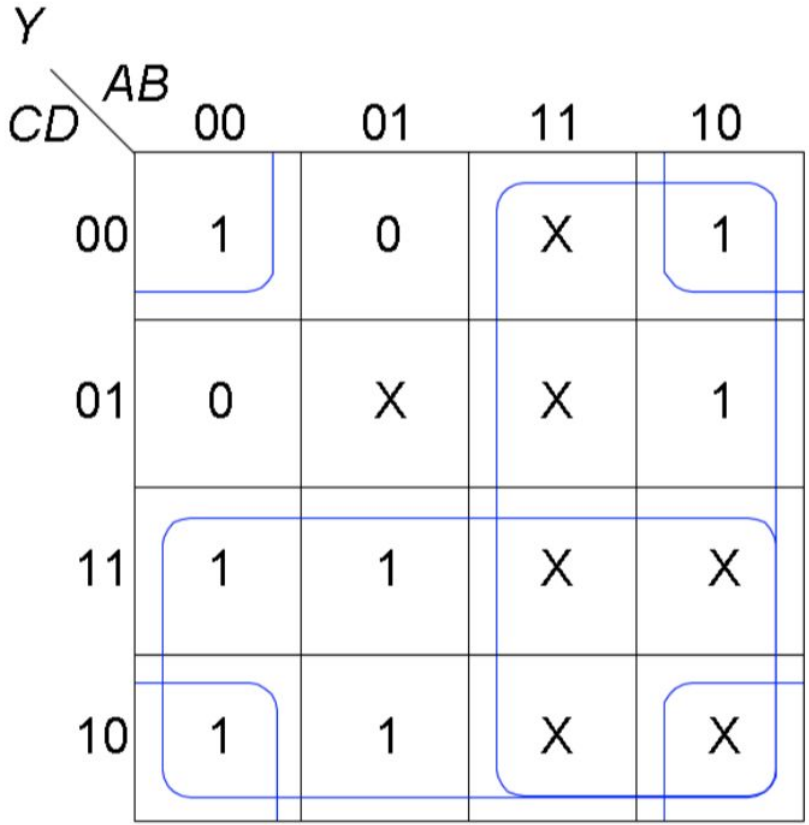
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Y	AB 00	01	11	10
CD 00	1	0	X	1
01	0	X	X	1
11	1	1	X	X
10	1	1	X	X

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



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

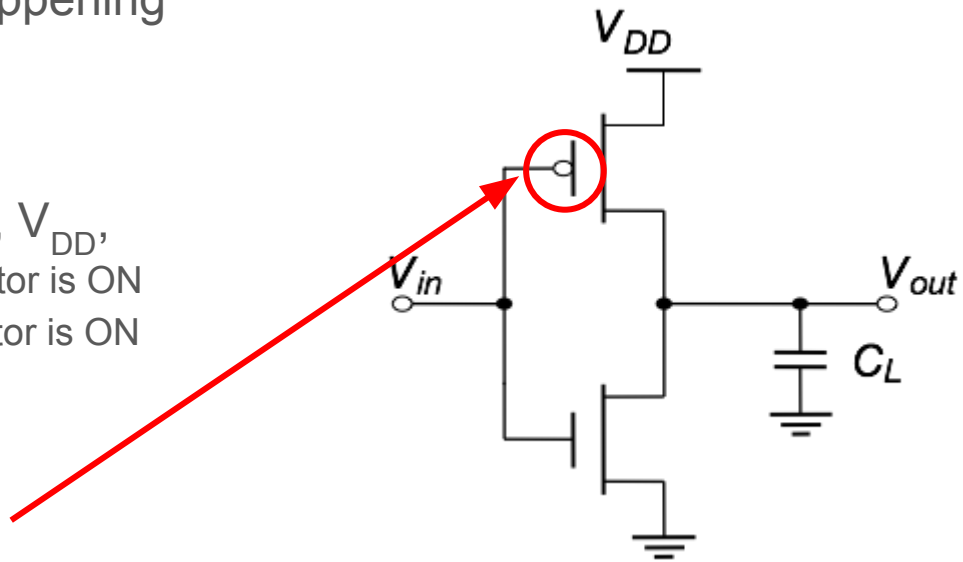
CMOS logic

nMOS + pMOS

- Several conceptual inversions happening
  - n-type vs p-type channels
  - gate-to-source voltage vs threshold
  - a few different symbols
- For the same +ve source voltage, V_{DD},
  - nMOS: logic 1 (near V_{DD}), the transistor is ON
  - pMOS: logic 0 (near V_{SS}), the transistor is ON
- Drawn to show logic inversion

  Note the bubble

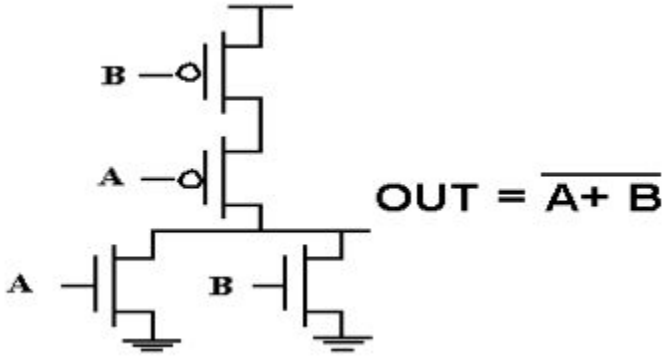
  Abstracts physical operation



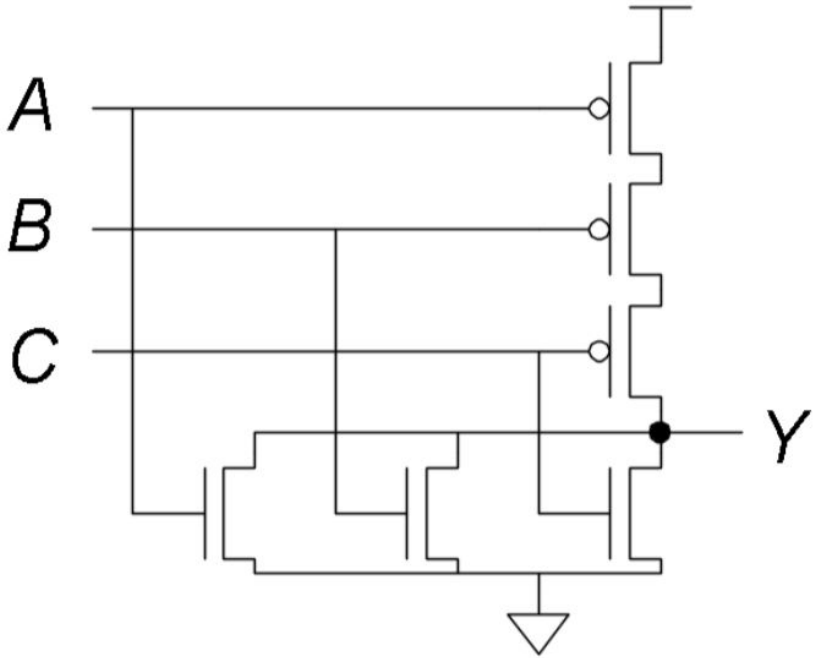
How do you build a three-input NOR gate?

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

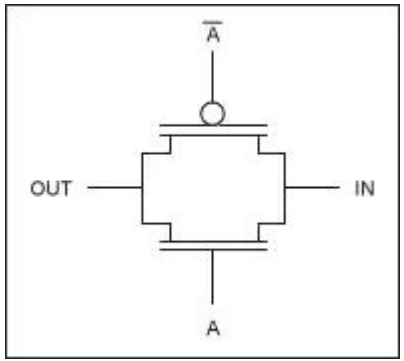
Truth Table of a 2 input NOR gate



How do you build a three-input NOR gate?



Transmission gates



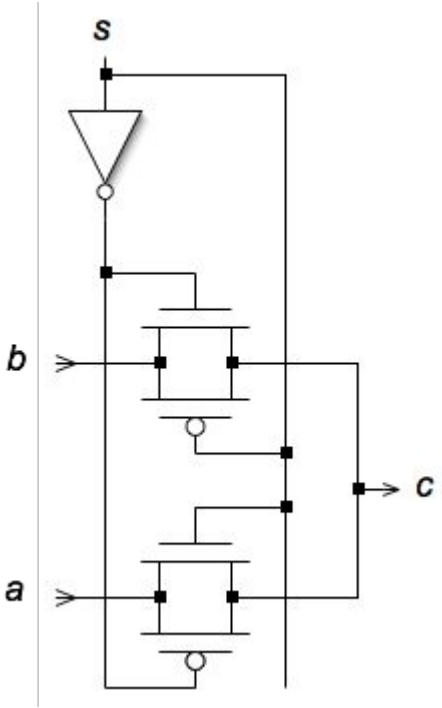
Logic gates:

nMOS passes 1s poorly; pMOS passes 0s poorly

Transmission gates:

In general, nMOS (bottom): pass zeros
pMOS (top): pass ones

Bi-directional, useful for simplifying circuits



Qu’est-ce que c’est?

References

- http://www.lbebooks.com/downloads/exportal/verilog_basys_example62-trafficlights.pdf
- Digital Design and Computer Architecture, David M. Harris & Sarah L. Harris
- Maxim Integrated, https://www.maximintegrated.com/en/app-notes/index.mvp/id/4243
- Wikipedia