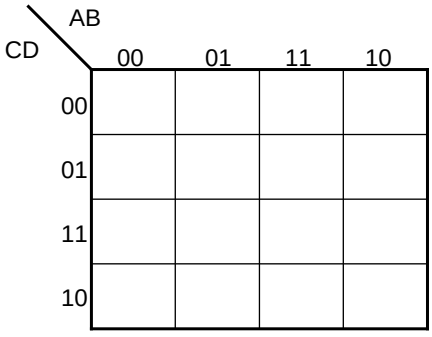
CSE 370

Sample Final Exam Questions

1) Logic Minimization



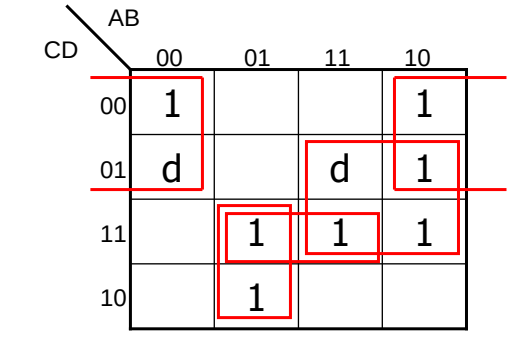
F = **Σ**m(0,6,7,8,9,11,15) + d(1,13)

1) Logic Minimization

CD \ AB	00	01	11	10
00	1			1
01	d		d	1
11		1	1	1
10		1		

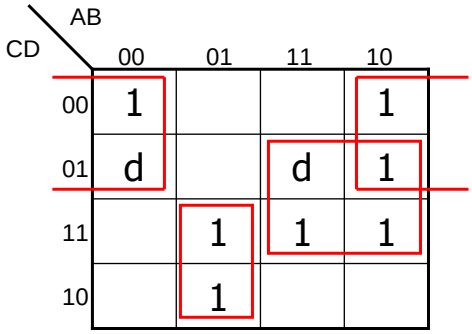
F = **Σ**m(0,6,7,8,9,11,15) + d(1,13)

1) Prime Implicants



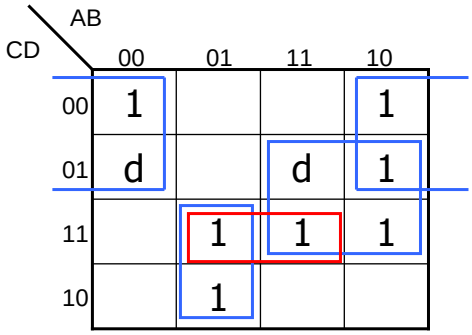
Note: Prime Implicant = largest box covering a 1

1) Essential PI's



Note: Essential PI = PI's that cover a 1 not covered by any other PI's

1) Logic Minimization

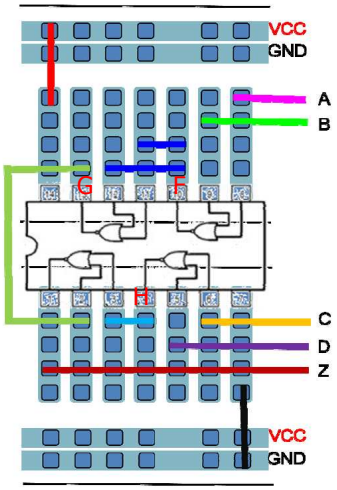


Prime Implicants: B'C', AD, A'BC, BCD

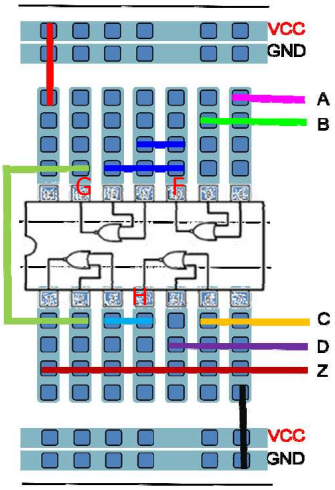
Essential Prime Implicants: B'C', AD, A'BC

Minimal Cover: F = B'C' + AD + A'BC

2a) Gates to Functions back into Gates



2a) Gates to Functions back into Gates



Z = ~(H + G)

Z = ~H & ~G

Z = ~(~(C+D)) & ~(~(F+F))

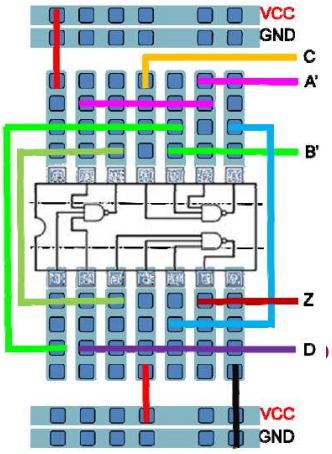
Z = (C+D) & F

Z = (C+D) & ~(A+B)

Z = (C+D) & ~A & ~B

Z = A'B'C + A'B'D

2b) Gates to Functions back into Gates



Z = A'B'C + A'B'D

Using deMorgan's Law…

Z = ~(~(A'B'C) & ~(A'B'D))

2c) Gates to Functions back into Gates

- NOR Circuit
  - 3 gate delays @ 7ns
  - 21ns total
- NAND Circuit
  - 2 gate delays @ 9ns
  - 18ns total

3a) Multiplexer Logic

A	B	C	D	Z
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

$Z = B'C + A'BD + AB'$

3a) Multiplexer Logic

A	B	C	D	Z
0	0	0	0	
0	0	0	1	
0	0	1	0	1
0	0	1	1	1
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	1
1	0	1	1	1
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

$Z = B'C + A'BD + AB'$

3a) Multiplexer Logic

A	B	C	D	Z
0	0	0	0	
0	0	0	1	
0	0	1	0	1
0	0	1	1	1
0	1	0	0	
0	1	0	1	1
0	1	1	0	
0	1	1	1	1
1	0	0	0	
1	0	0	1	
1	0	1	0	1
1	0	1	1	1
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Z = B'C + A'BD + AB'

3a) Multiplexer Logic

A	B	C	D	Z
0	0	0	0	
0	0	0	1	
0	0	1	0	1
0	0	1	1	1
0	1	0	0	
0	1	0	1	1
0	1	1	0	
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

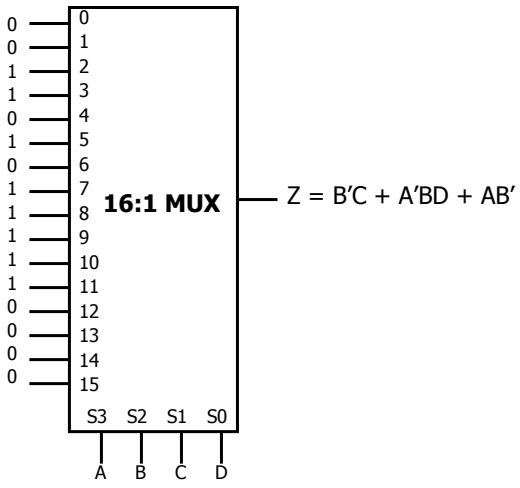
Z = B'C + A'BD + AB'

3a) Multiplexer Logic

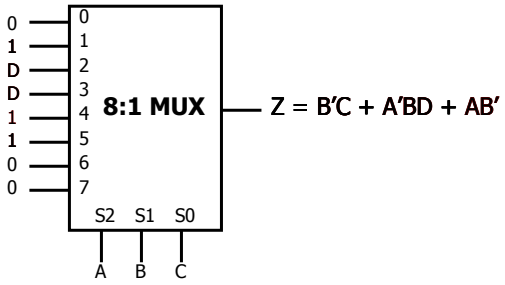
A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$Z = B'C + A'BD + AB'$

3b) Multiplexer Logic



3c) Multiplexer Logic



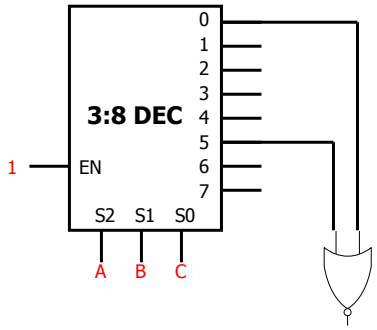
Note: Requires no extra logic

4) Decoder Logic

A	B	C	W
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

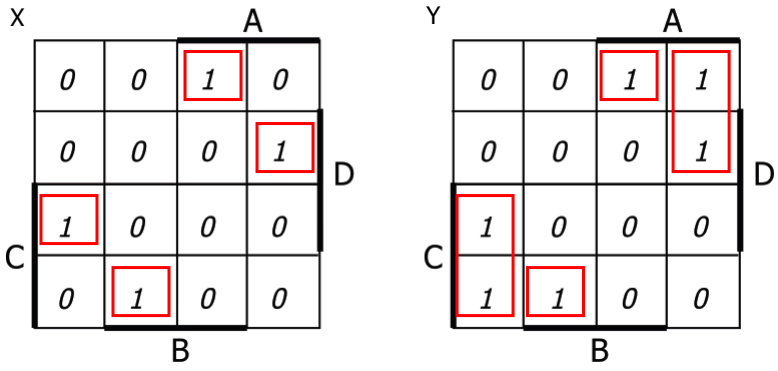
W = (A xor C) + B

4) Decoder Implementation



W = (A xor C) + B

5) Programmable Logic



Z = A'BCD' + ABC'D'

Note: 2 Common Prime Implicants can be used by X and Y

X = Z + A'B'CD + AB'C'D

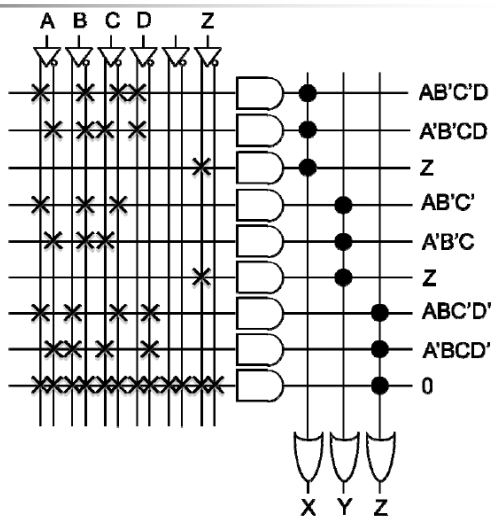
Y = Z + A'B'C + AB'C'

5) Programmable Logic

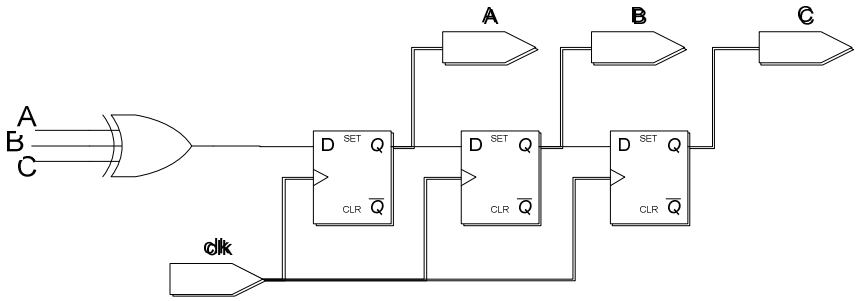
Z = A'BCD' + ABC'D'

X = Z + A'B'CD + AB'C'D

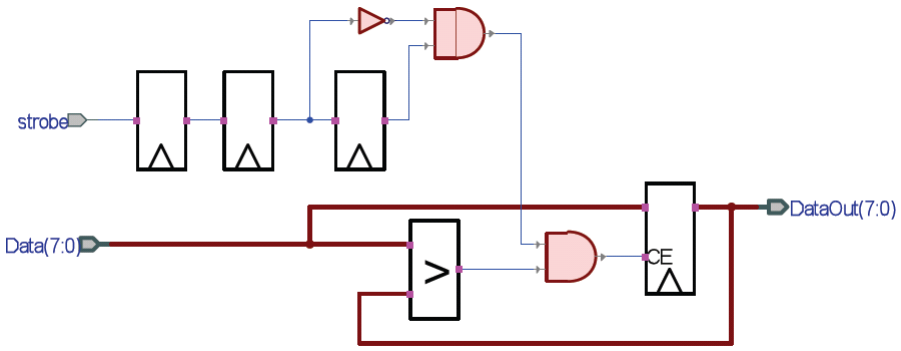
Y = Z + A'B'C + AB'C'



6) Verilog to Circuit



7) Circuit to Verilog



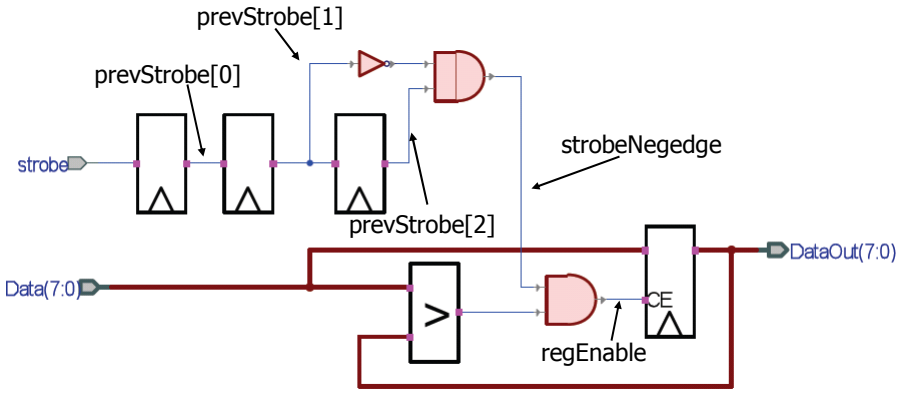
7) Circuit to Verilog

```
module maxValue (DataOut, strobe, Data);
        // Port Definitions
        output reg [7:0] DataOut;
        input strobe;
        input [7:0] Data;

        // Behavioral Statements for DataOut

        // other required internal signals

endmodule
```

7) Circuit to Verilog

```
// Behavioral Statements for DataOut
always @ (posedge clk) begin
        if (regEnable) DataOut <= Data;
        else DataOut <= DataOut;
end
```

7) Circuit to Verilog



7) Circuit to Verilog

```
// other required internal signals
wire regEnable, strobeNegedge;
reg prevStrobe[2:0];

assign regEnable = (Data > DataOut) && strobeNegedge
assign strobeNegedge = prevStrobe[2] & ~prevStrobe[1];

always @ (posedge clk)
        prevStrobe = {prevStrobe[1:0],strobe};
```

8) Carry Look-ahead Adder

Recall:

Sum = A xor B xor C

Cout = AB + BC + AC

Rewrite Cout:

Cout = AB + C (A + B)

8) Carry Look-ahead Adder

A	B	P_{XOR}	P_{OR}
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

2 functions are only different if A and B are 1

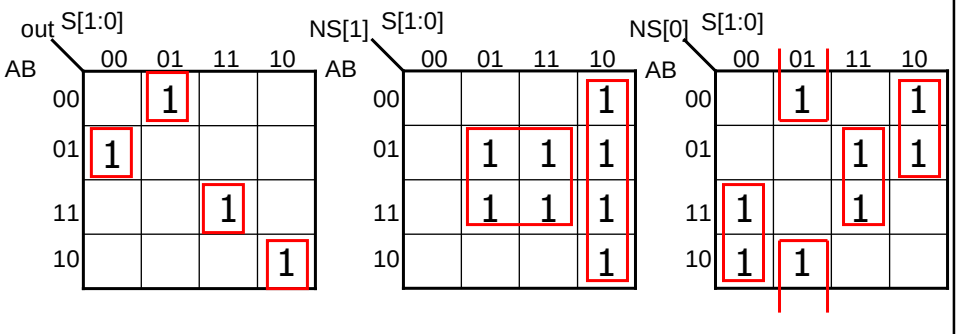
8) Carry Look-ahead Adder

- Need carry out if A and B are 1
  - carry is generated in this case
  - don't need propagate to cover this case
  - these are then functionally equivalent
- Recall: Sum = A xor B xor C
  - P = A + B: have to calculate A xor B
  - P = A xor B: reuse this to compute Sum
  - P = A xor B uses less logic

9) Verilog State Machine

S[1]	S[0]	A	B	NS[1]	NS[0]	out
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	0	1	1	1	0
1	0	1	0	1	0	1
1	0	1	1	1	0	0
1	1	0	0	0	0	0
1	1	0	1	1	1	0
1	1	1	0	0	0	0
1	1	1	1	1	1	1

9) Verilog State Machine

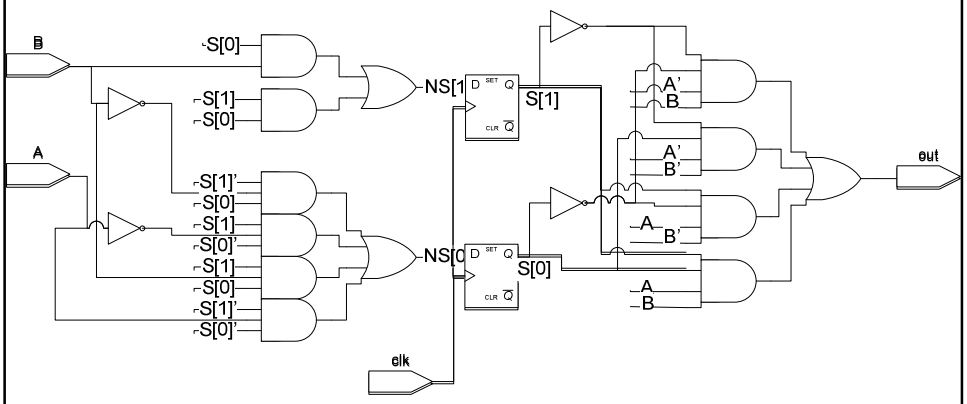


out = S[1]'S[0]'A'B + S[1]'S[0]A'B' + S[1]S[0]'AB' + S[1]S[0]AB

NS[1]=S[0]B + S[1]S[0]'

NS[0] = S[1]'S[0]B' + S[1]S[0]'A' + S[1]S[0]B + S[1]'S[0]'A

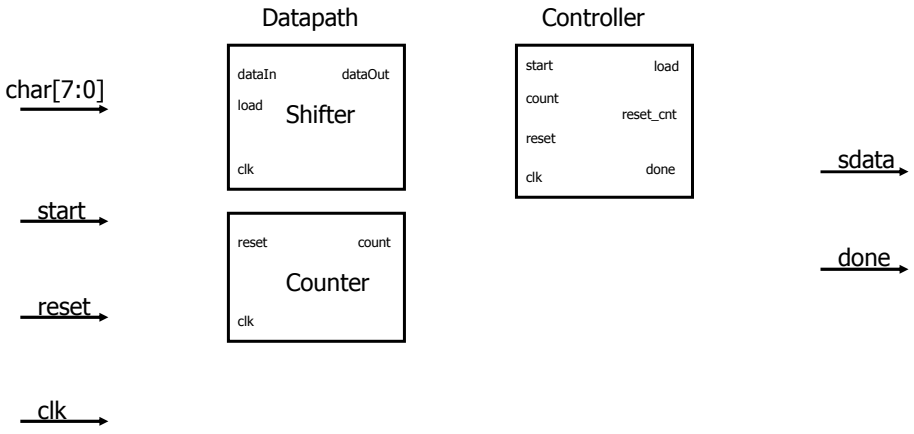
9) Verilog State Machine



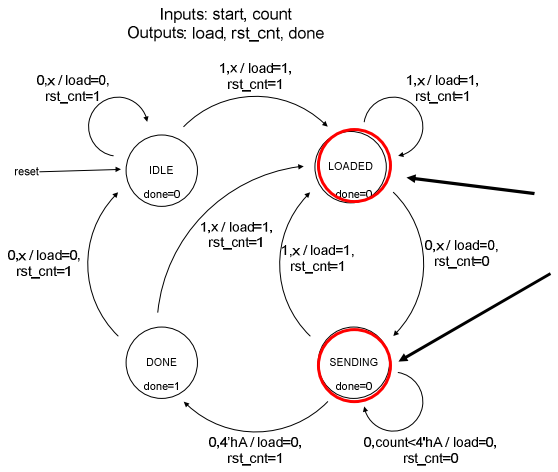
10) Design: Parallel to Serial

- Shift Register
  - Load parallel data
  - Shift out serial data
- Counter
  - How many bits have shifted out
- Controller
  - Shift Enable
  - Done Shifting

10a) Design: Parallel to Serial

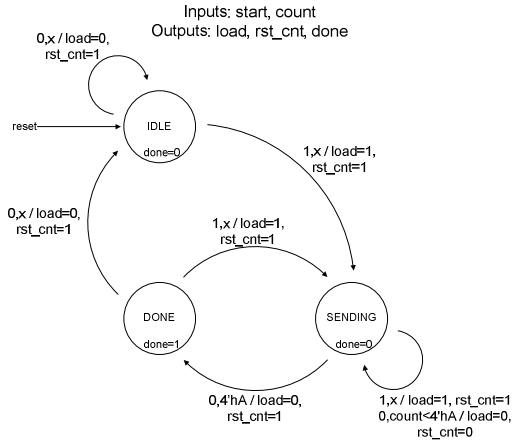


10b) Design: Parallel to Serial

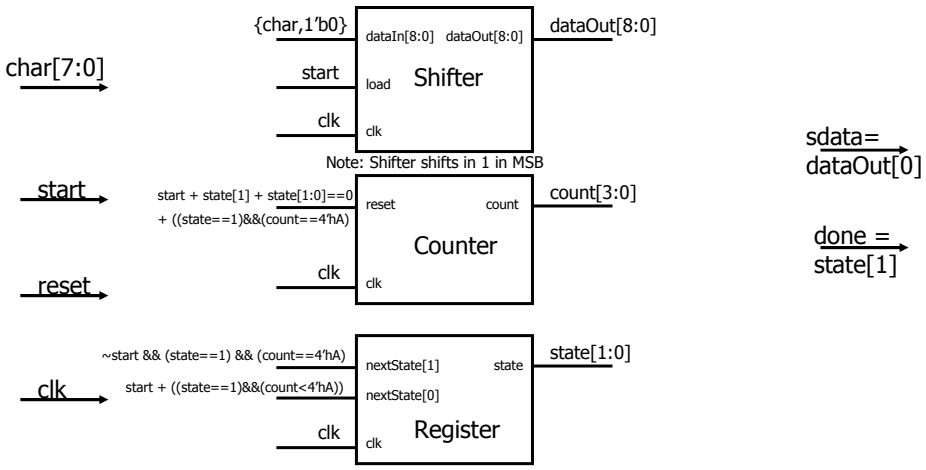


Can we collapse these to 1 state?

10b) Design: Parallel to Serial



10c) Design: Parallel to Serial



11) x370 CALL Instruction

- CALL Requirements
  - Load address into PC from instruction
  - Store PC+1 into RD

11) Implementation

- Load address into PC
  - inst[5:0] muxed to input of PC
  - Load PC asserted
- Store PC+1 into RD
  - PC muxed to A input of ALU
  - ALU inst set to INC
  - ALU output muxed to reg write data
  - Reg write address set to R_D

Final Notes

- Homework 9
  - Due today 5:30 pm
  - Can accept until Sunday 5:30 if dropbox doesn't close
- Final Exam
  - Wednesday June 9, 8:30-10:20
  - EEB 045
  - Don't be late!!!

Final Questions?