


Penn Engineering **ESE**

Work Preclass
4:35pm lecture start

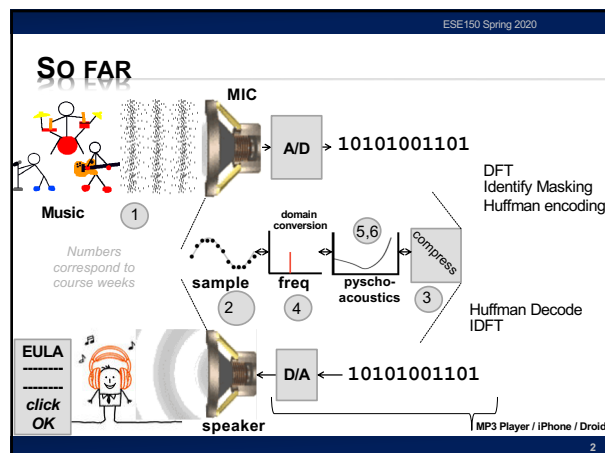


Lecture #7 – Digital Logic

ESE 150 – DIGITAL AUDIO BASICS

ESE150 Spring 2020

Based on slides © 2009–2020 DeHon



ESE150 Spring 2020

HOW PROCESS

- × How do we build a machine to perform these operations?
 - + From Digital Samples → compressed digital data → Digital Samples
- × Down to bottom
 - + If we can build **one** kind of primitive element,
 - × ...and connect together large collections of them
 - + can build a machine to perform *any* digital computation

3

ESE150 Spring 2020

LECTURE TOPICS

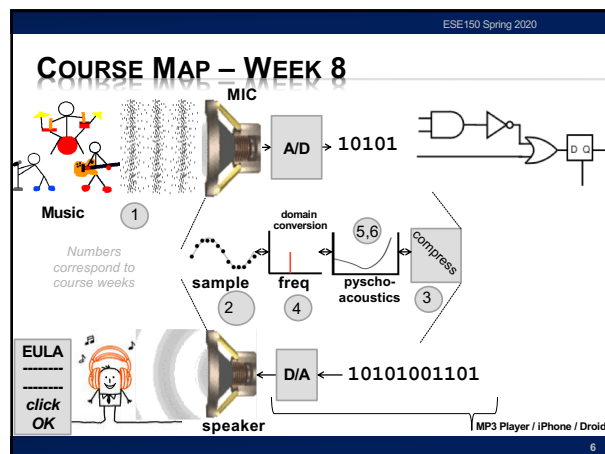
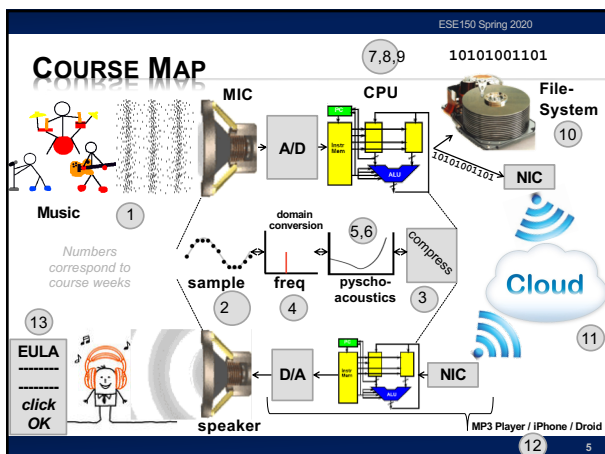
- × Setup
- × Where are we?
- × Combinational Logic
- × Sequential Logic
- × FPGAs
- × Next Lab

Will try asking questions like we usually do.

Go in Alphabetical order.

When called upon, unmute and answer.

4



ESE150 Spring 2020

COMBINATIONAL LOGIC

7

ESE150 Spring 2020

GATE

- × **Primitive binary function**
 - + Computes a binary output from a small number of binary inputs
- × **Can specify function with a Truth Table**
 - + Defines the output for each input combination


Input 0	Input 1	Output
0	0	
0	1	
1	0	
1	1	

8

ESE150 Spring 2020

AND GATE

- × **AND**
 - + Output is 1 (true) when all inputs are 1 (true)




Input 0	Input 1	AND
0	0	0
0	1	0
1	0	0
1	1	1

9

ESE150 Spring 2020

NOT GATE

- × **Not**
 - + Output is opposite of input




input	NOT
0	1
1	0

10

ESE150 Spring 2020

OR GATE

- × **OR**
 - + Output is 1 (true) when any input is 1 (true)
 - + (fillin truth table for OR)




Input 0	Input 1	OR
0	0	
0	1	
1	0	
1	1	

11

ESE150 Spring 2020

OR GATE

- × **OR**
 - + Output is 1 (true) when any input is 1 (true)
 - + (fillin truth table for OR)



Input 0	Input 1	OR
0	0	0
0	1	1
1	0	1
1	1	1

12

ESE150 Spring 2020

CLAIM

- × Can compute any Boolean Function from AND, OR, NOT
 - + (actually from NAND)

13

ESE150 Spring 2020


MODEL: COMBINATIONAL LOGIC

- × Compute some “function”
 - + $f(i_0, i_1, \dots, i_n) \rightarrow o_0, o_1, \dots, o_m$
- × Each unique input vector
 - + implies a particular, deterministic, output vector

14

ESE150 Spring 2020

BIG AND




- × AND
 - + Output is 1 (true) when all inputs are 1 (true)
- × How build n-input AND from AND2 gates?

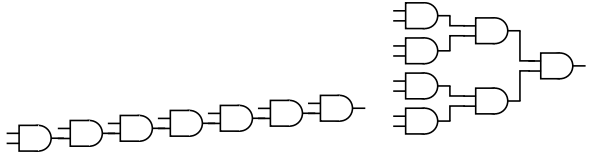
15

ESE150 Spring 2020

BIG AND




- × AND
 - + Output is 1 (true) when all inputs are 1 (true)
- × How build n-input AND from AND2 gates?



16

ESE150 Spring 2020

BIG OR




- × OR
 - + Output is 1 (true) when any input is 1 (true)
- × How build n-input OR from OR2?

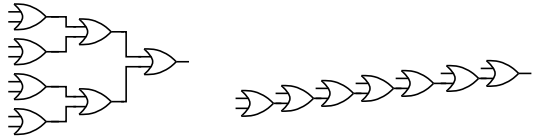
17

ESE150 Spring 2020

BIG OR



- × OR
 - + Output is 1 (true) when any input is 1 (true)
- × How build n-input OR from OR2?



18

ESE150 Spring 2020

INPUT CASE

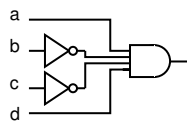
- × How can we create an expression that is true for a specific input case?
 - + E.g. have a function of 4 inputs: a, b, c, d
- × How many potential values for a, b, c, d?
 - + Rows in our truth table
- × Give one example of values for a, b, c, d?
- × How create an expression that is true for that case?

19

ESE150 Spring 2020

INPUT CASE

- × How can we create an expression that is true for a specific input case?
 - + E.g. have a function of 4 inputs: a, b, c, d
- × How many potential values for a, b, c, d?
 - + Rows in our truth table: $2^4 = 16$
- × Give one example of values for a, b, c, d?
 - + a & !b & !c & d
 - + (! for inversion, & for and)
- × How create an expression that is true for that case?



20

ESE150 Spring 2020

SINGLE OUTPUT DIGITAL FUNCTION

- × Given have logic to implement each input case
- × How implement entire function?

a	b	c	F(a,b,c)
0	0	0	1
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

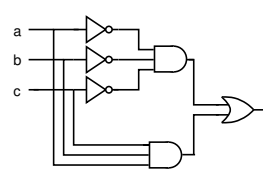
21

ESE150 Spring 2020

SINGLE OUTPUT DIGITAL FUNCTION

- × Given have logic to implement each input case
- × How implement entire function?

a	b	c	F(a,b,c)
0	0	0	1
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



22

ESE150 Spring 2020

MULTIPLE OUTPUT FUNCTION

- × What do you do if your Digital Function needs multiple output bits?

23

ESE150 Spring 2020

COMBINATIONAL LOGIC AS GATES

- × Start with truth table
- × Single output {0, 1}
 - + Use inverters to produce complements of inputs
 - + For each input case
 - × If output is a 1
 - × Develop an AND to detect that case
 - × Decompose AND into gates
 - + OR together the output of all such AND functions
 - × Decompose OR into gates
- × Multiple outputs
 - + Repeat for each output

This solution won't typically be the smallest or fastest...

24

ESE150 Spring 2020

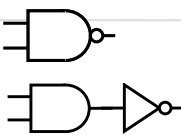
CONCLUDE

- × Can implement any combinational logic function out of a collection of
 - + OR2, AND2, NOT gates

25

ESE150 Spring 2020

NAND2 GATE



- × **NAND = NOT AND**
 - + Output is 0 (false) when all inputs are 1 (true); 0 otherwise

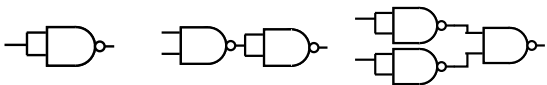
Input 0	Input 1	NAND
0	0	1
0	1	1
1	0	1
1	1	0

26

ESE150 Spring 2020

NAND UNIVERSALITY

- × How implement
 - + What function does each circuit implement?

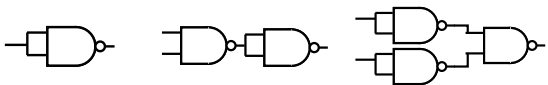


27

ESE150 Spring 2020

NAND UNIVERSALITY

- × How implement
 - + What function does each circuit implement?

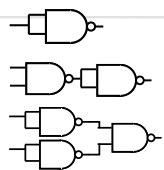


NOT AND OR

28

ESE150 Spring 2020

NAND UNIVERSALITY

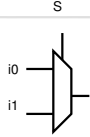


- × Can implement
 - + NOT from NAND2
 - + AND2 from NAND2
 - + OR2 from NAND2
- × Can implement any combinational logic function out of a collection of
 - + OR2, AND2, NOT gates
- × Therefore: Can implement any combinational logic function out of a collection of NAND2 gates

29

ESE150 Spring 2020

MULTIPLEXER GATE



- × **MUX**
 - + When S=0, output=i0
 - + When S=1, output=i1

S	i0	i1	Mux2(S,i0,i1)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Table?
AND, OR, NOT Implementation?

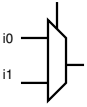
30

ESE150 Spring 2020

MULTIPLEXER GATE

- × **MUX**
 - + When $S=0$, output= i_0
 - + When $S=1$, output= i_1

S	i_0	i_1	Mux2(S, i_0 , i_1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



Truth Table?

AND, OR, NOT Implementation?

$(i_0 \& !S) \mid (i_1 \& S)$

& -- and
| -- or
! -- not

31

ESE150 Spring 2020

ARITHMETIC

- × **Addition is also a digital logic function**
 - + Maps set of inputs ($a_3 a_2 a_1 a_0 b_3 b_2 b_1 b_0$)
 - + To an output bit vector ($c_4 c_3 c_2 c_1 c_0$)
- × ...as is subtraction, multiplication, division, square root...


32

ESE150 Spring 2020

FULL ADDER

- × **Adds 3 inputs to produce 2b output**
 - + Binary inputs: a, b, c
 - + Binary outputs: carry, sum
 - + Two bit result:
 - + $carry \cdot 2 + sum = a + b + c$
 - + Can produce truth table and logic (Lab)

a b c



carry sum

33

ESE150 Spring 2020

EXAMPLE: BIT-LEVEL ADDITION

- × **Addition**
 - + Base 2 example
 - + *Work together*

C: 1101101000

A: 01101101010

B: 01100101100

S: 11010010110

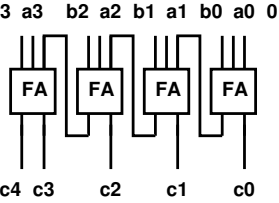
34

ESE150 Spring 2020

N-BIT ADDER

- × **Given Full Adders**
 - + Can build N-bit adder by connecting N full adders

b3 a3 b2 a2 b1 a1 b0 a0 0



c4 c3 c2 c1 c0

35

ESE150 Spring 2020

SEQUENTIAL LOGIC

36

ESE150 Spring 2020

MUX WITH FEEDBACK

- × What happens when S=0?
- × What happens when S=1?

37

ESE150 Spring 2020

MUX WITH FEEDBACK

- × What happens when S=0?
 - + Out=i0 – passes input to output
- × What happens when S=1?
 - + Out=Out – holds values

38

ESE150 Spring 2020

MUX WITH FEEDBACK

- × Assuming i0 doesn't change what happens when S goes from 0 to 1?

39

ESE150 Spring 2020

MUX WITH FEEDBACK

- × Assuming i0 doesn't change what happens when S goes from 0 to 1?
 - + Samples value from i0 and holds it on output

40

ESE150 Spring 2020

LATCH

- × Element that can hold a previous value of an input

41

ESE150 Spring 2020

FLIP-FLOP (FF)

- × Use a pair to create a flip-flop
 - + Also call register
- × What happens when
 - + CLK is low (0) ?
 - + CLK is high (1) ?
 - + CLK transitions from 0 to 1?

42

ESE150 Spring 2020

FLIP-FLOP (FF)

- × Use a pair to create a flip-flop
 - + Also call register
- × What happens when
 - + CLK is low (0)?
 - ▷ D input passes to second mux,
 - ▷ Second mux holds output
 - + CLK is high (1)?
 - ▷ First mux holds input
 - ▷ Second mux passes output of first
 - + CLK transitions from 0 to 1?
 - ▷ D input at 0 latched (held)
 - ▷ Second mux passes that D value to output

43

ESE150 Spring 2020

FLIP-FLOP (FF)

- × Use a pair to create a flip-flop
 - + Also call register
- × Sample D input on 0→1 transition of clock (CLK)
- × Never an open path from D→Q
 - + One of the mux latches always in hold state

44

ESE150 Spring 2020

STATE ELEMENT

- × Latch or Register is a state element
- × Allows circuit to remember a value
- × Build computations that
 - + Depend on past inputs
 - + Reuse hardware in time

45

ESE150 Spring 2020

ACCUMULATOR

- × Sum a sequence of values

46

ESE150 Spring 2020

ACCUMULATOR

- × Start with an Adder

47

ESE150 Spring 2020

ACCUMULATOR

- × Store running sum as state in registers

48

ESE150 Spring 2020

ACCUMULATOR

× Wrap register outputs back to inputs

ESE150 Spring 2020

ACCUMULATOR

× Maybe extend accumulator bits to hold larger sum

× Maybe more...

ESE150 Spring 2020

ACCUMULATOR

× What happens:

- + Start with a3:a0 at 0
- + CLK low
- + I3:i0=2 (0010)
- + FF inputs?

ESE150 Spring 2020

ACCUMULATOR

× What happens:

- + Start with a3:a0 at 0
- + CLK low
- + I3:i0=2 (0010)
- + FF inputs?
- + CLK goes high: a3:a0?

ESE150 Spring 2020

ACCUMULATOR

× What happens:

- + Start with a3:a0 at 0
- + CLK low
- + I3:i0=2 (0010)
- + CLK goes high: a3:a0?
- + I3:0=3 (0011)
- + FF inputs?

ESE150 Spring 2020

ACCUMULATOR

× What happens:

- + Start with a3:a0 at 0
- + CLK low
- + I3:i0=2 (0010)
- + CLK goes high: a3:a0?
- + I3:0=3 (0011)
- + FF inputs?
- + CLK goes high: a3:a0?

ESE150 Spring 2020

ACCUMULATOR

- × **a=0**
- × **while (true)**
 - + `a=a+getInput();`
- × **Accumulates input values i**
 - + Integration or summation

55

ESE150 Spring 2020

STATE FOR SEQUENCING AND CONTROL

- × **Useful when trying to control things**
 - + E.g. Perform a sequence of operations
- × **Robot**
 - + Open-gripper
 - + Move-forward
 - + Close-gripper
 - + Lift

56

ESE150 Spring 2020

STATE FOR CONDITIONAL CONTROL

- × **Useful when need to behave differently based on something in the past**
 - + Remember if elevator going up or down
 - + Remember/count coins from consumer
 - + Remember some mode set by user
 - Displaying in Centigrade or Fahrenheit
- × **Idea**
 - + Store state
 - + Use as input to logic

57

ESE150 Spring 2020

FINITE-STATE MACHINE (FSM)

- × **Sequential model of computation**
- × **State (in registers) + combinational logic**
- × **Compute outputs and next state from inputs and state**

58

ESE150 Spring 2020

FSM EXAMPLE

- × **Simplified Vending Machine**
 - + Only input quarters
 - + Only vend one item (output signal to indicate vending)
 - + Item costs 2 quarters
 - + Coin Return request and control
- × **Two states: waiting, one-quarter (one)**
- × **Two inputs: quarter, coin-return (creturn)**
- × **Two outputs: vend, return-quarter (qreturn)**

59

ESE150 Spring 2020

TRUTH TABLE MODEL

Complete together

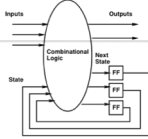
state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1			
waiting	1	0	0	0	one
waiting	1	1			
one	0	0			
one	0	1			
one	1	0			
one	1	1			

60

ESE150 Spring 2020

TRUTH TABLE MODEL

Complete together



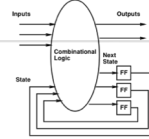
state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1			
one	0	0			
one	0	1			
one	1	0			
one	1	1			

61

ESE150 Spring 2020

TRUTH TABLE MODEL

Complete together



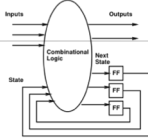
state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0			
one	0	1			
one	1	0			
one	1	1			

62

ESE150 Spring 2020

TRUTH TABLE MODEL

Complete together



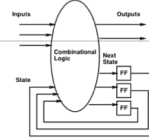
state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0	0	0	one
one	0	1			
one	1	0			
one	1	1			

63

ESE150 Spring 2020

TRUTH TABLE MODEL

Complete together



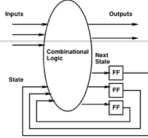
state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0	0	0	one
one	0	1	0	1	waiting
one	1	0			
one	1	1			

64

ESE150 Spring 2020

TRUTH TABLE MODEL

Complete together



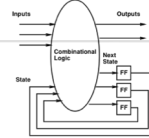
state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0	0	0	one
one	0	1	0	1	waiting
one	1	0	1	0	waiting
one	1	1			

65

ESE150 Spring 2020

TRUTH TABLE MODEL

Complete together



state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0	0	0	one
one	0	1	0	1	waiting
one	1	0	1	0	waiting
one	1	1	0	1	one

66

ESE150 Spring 2020

SWITCH-STATEMENT MODEL

```

× While (true)
×   switch (state) {
×     case waiting:
×       if (quarter && !creturn)
×         state=one;
×       else
×         state=waiting;
×       qreturn=quarter && creturn;
×       vend=0;
×       break;

```

67

ESE150 Spring 2020

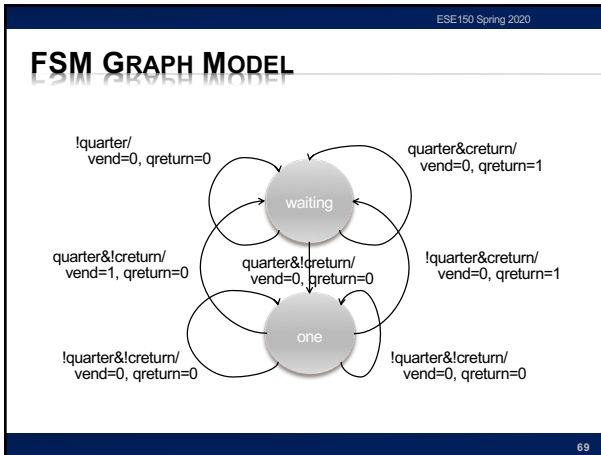
SWITCH-STATEMENT MODEL (CONT.)

```

×   case one:
×     if ((quarter && !creturn)||
×        (!quarter&&creturn))
×       state=waiting;
×     else
×       state=one;
×     qreturn=creturn;
×     vend=quarter&& !creturn;
×     break;
×   } // switch
× } // while

```

68



ESE150 Spring 2020

PROGRAMMABLE LOGIC

70

ESE150 Spring 2020

MUX CAN BE A PROGRAMMABLE GATE

- × Programmable Gate
 - + Can be programmed to act as any gate
 - + Use state (e.g. FF) to "program" truth table of a gate

Input 0	Input 1	Output
0	0	
0	1	
1	0	
1	1	

71

ESE150 Spring 2020

EXAMPLE: AND

- × How do we program to behave as AND2?

72

ESE150 Spring 2020

EXAMPLE: AND

- How do we program to behave as AND?

d00 d01 d10 d11
(data inputs)

(select inputs)
s1
s0

73

ESE150 Spring 2020

EXAMPLE: OR

- How do we program to behave as OR?

d00 d01 d10 d11
(data inputs)

(select inputs)
s1
s0

74

ESE150 Spring 2020

EXAMPLE: OR

- How do we program to behave as OR?

d00 d01 d10 d11
(data inputs)

(select inputs)
s1
s0

75

ESE150 Spring 2020

LOOK-UP TABLE (LUT)

- Can generalize to any number of inputs

output

select inputs

76

ESE150 Spring 2020

CONNECTING GATES

- Once we can build gates
- ...still need to connect the gates together.
- Select which gate outputs become inputs to other gates.

b3 a3 b2 a2 b1 a1 b0 a0 0

i3 i2 i1 i0

c4 c3 c2 c1 c0

a3 a2 a1 a0

77

ESE150 Spring 2020

MUX CAN BE PROGRAMMABLE INTERCONNECT

Trick: Use multiplexer to programmably select gate input.

78

ESE150 Spring 2020

PROGRAMMABLE BLOCKS

79

ESE150 Spring 2020

PROGRAMMABLE GATES AND INTERCONNECT

If time permits:
How program (fill in yellow programmable cells) to implement a full adder?

80

ESE150 Spring 2020

FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

- × **Collection of Programmable Gates**
 - + Can “program” by setting state bits
 - + LUTs that can be programmed to be any gate
 - × With optional Flip-Flops to use for state
 - + Programmable interconnect to “wire” the gates together

81

ESE150 Spring 2020

FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

82

ESE150 Spring 2020

NEXT LAB

- × **Program an FPGA in Verilog**
 - + Build an adder
 - + Build an accumulator
- × **Should have hardware**
- × **Will need to install software on your computer**
 - + Budget time for that
 - + Lab instructions out now

83

ESE150 Spring 2020

BIG IDEAS

- × **Can implement any combinational digital logic function from nand2 gates**
- × **Can implement any FSM from nand2 gates and registers**
- × **Can build a single chip that can be programmed to behave as any collection of gates**
 - + As long as don't need more gates than it provides

84

LEARN MORE

- × **CIS240 – do a bit more logic**
- × **ESE370 – how to implement gates, latches, and memories from transistors**
- × **ESE532 – how to build large-scale computations from logic**

REMINDER

- × **Formal Lab Report Due Sunday (11:59pm PDT)**
- × **Extra Lab Session to help**
 - + Don't get stuck for hours on any piece
- × **Lecture/Lab Feedback forms in Google Docs**
 - + Linked from syllabus