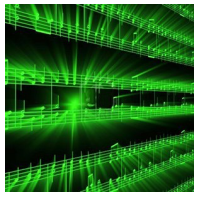
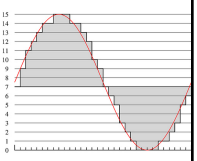
Penn Engineering

ESE





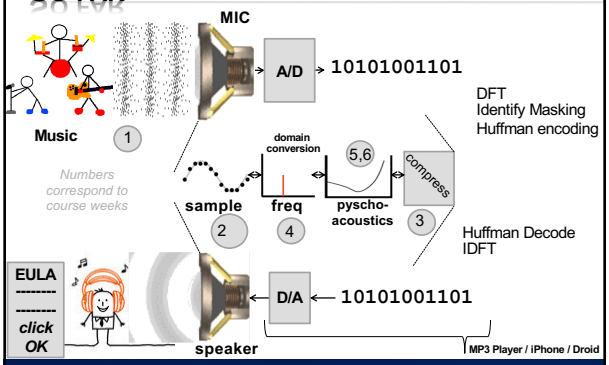
Lecture #7 – Digital Logic

ESE 150 – Digital Audio Basics

ESE150 Spring 2020

Based on slides © 2009--2020 DeHon

So far



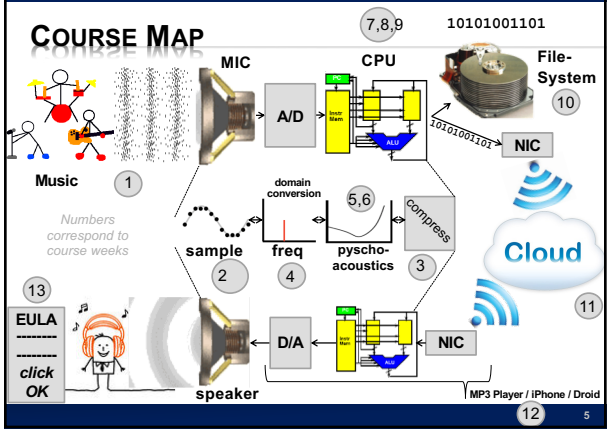
How Process

- **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

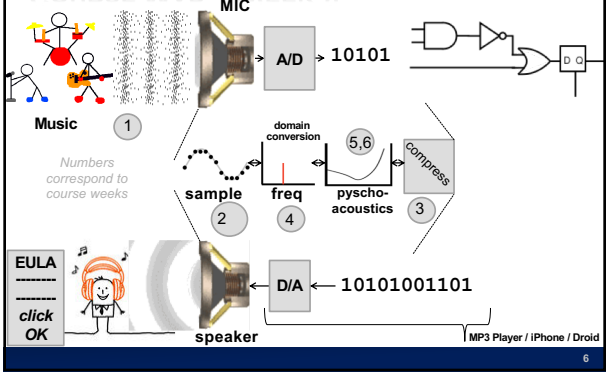
Lecture Topics

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

Course Map



Course Map – Week 8



COMBINATIONAL LOGIC

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	

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

NOT GATE

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

input	NOT
0	1
1	0

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	

CLAIM

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

Model: Combinational Logic

- **Compute some "function"**
  - $f(i_0,i_1,\ldots i_n) \rightarrow o_0,o_1,\ldots o_m$
- **Each unique input vector**
  - implies a particular, deterministic, output vector

Big AND

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

Big OR

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

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?**

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

Multiple Output Function

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

Combinational Logic as Gates

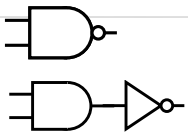
- **Start with truth table**
- **Single output {0, 1}**
  - Use inverters to produce complements of inputs
  - For each input case
    - If output is a 1
      - Develop an AND to detect that case
        - Decompose AND into gates
  - OR together the output of all such AND functions
    - Decompose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

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

Conclude

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

NAND2 Gate

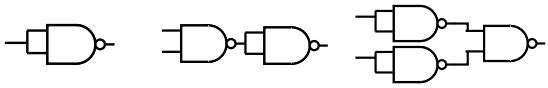


- **NAND = NOT AND**
  - Output is 0 (true) when all inputs are 1 (true); 0 otherwise

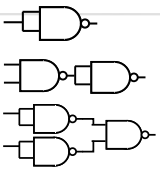
Input 0	Input 1	NAND
0	0	1
0	1	1
1	0	1
1	1	0

NAND Universality

- **How implement**
  - What function does each circuit implement?

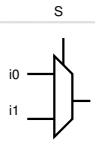


NAND Universality



- **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**

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?

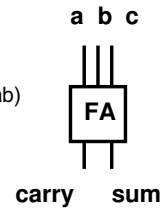
AND, OR, NOT Implementation?

Arithmetic

- **Addition is also a digital logic function**
  - Maps set of inputs (a3 a2 a1 a0 b3 b2 b1 b0)
  - To an output bit vector (c4 c3 c2 c1 c0)
- **…as is subtraction, multiplication, division, square root….**

Full Adder

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



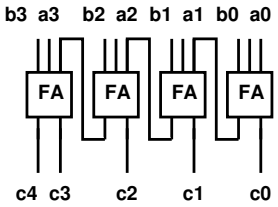
Example: Bit-Level Addition

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

C: 1101101010000
A: 01101101010
B: 01100101100
S: 11010010110

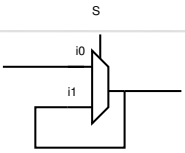
N-Bit Adder

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



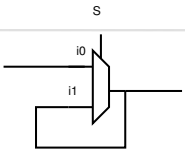
Sequential Logic

Mux with Feedback



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

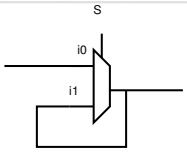
MUX WITH FEEDBACK

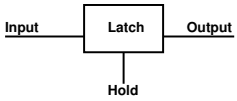


- **Assuming i0 doesn't change what happens when S goes from 0 to 1?**

LATCH

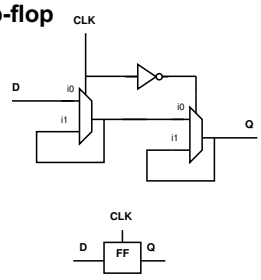
- **Element that can hold a previous value of an input**





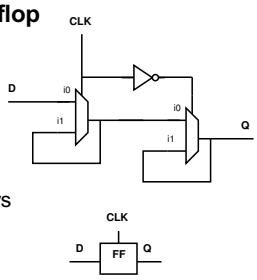
FLIP-FLOP (FF)

- **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?



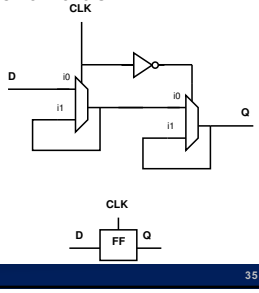
FLIP-FLOP (FF)

- **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



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

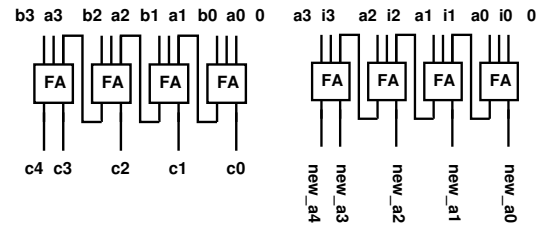


ACCUMULATOR

- **Sum a sequence of values**

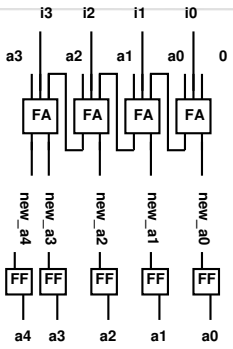
ACCUMULATOR

- **Start with an Adder**

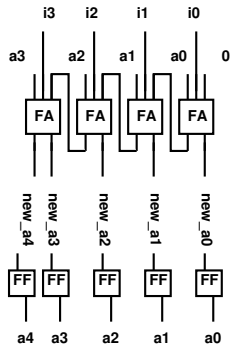


ACCUMULATOP

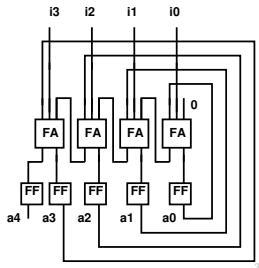
- **Store running sum as state in registers**



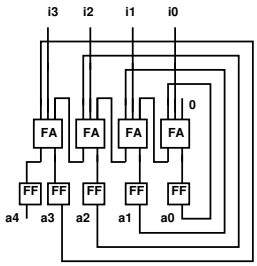
ACCUMULATOR



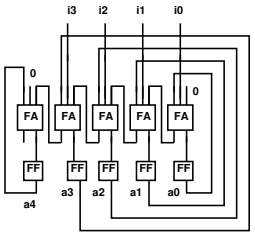
- Wrap register outputs back to inputs



ACCUMULATOR



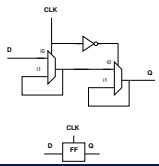
- Maybe extend accumulator bits to hold larger sum

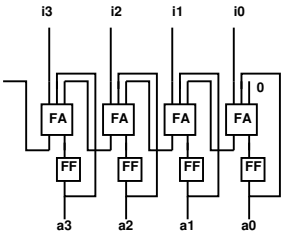


- Maybe more…

ACCUMULATOR

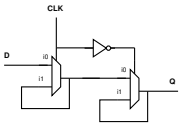
- **What happens:**
  - Start with a3:a0 at 0
  - CLK low
  - I3:i0=2 (0010)
  - FF inputs?

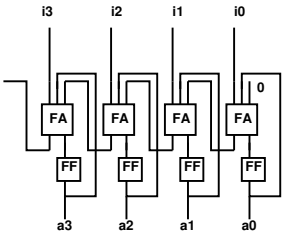




ACCUMULATOR

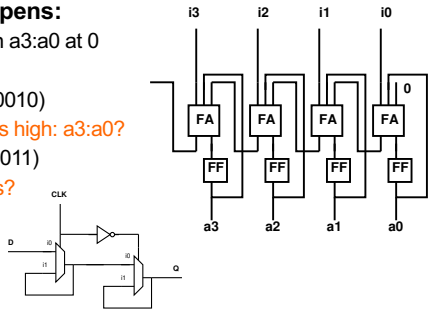
- **What happens:**
  - Start with a3:a0 at 0
  - CLK low
  - I3:i0=2 (0010)
  - FF inputs?
  - CLK goes high: a3:a0?





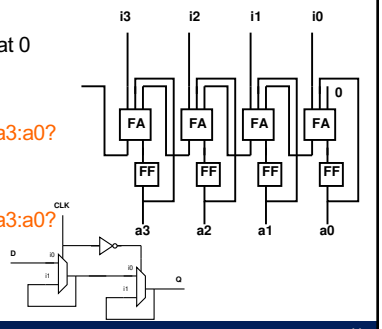
ACCUMULATOR

- **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?



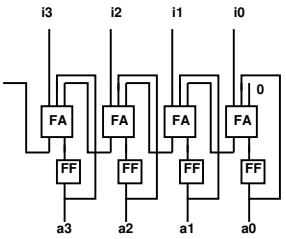
ACCUMULATOR

- **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?



ACCUMULATOR

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



STATE FOR SEQUENCING AND CONTROL

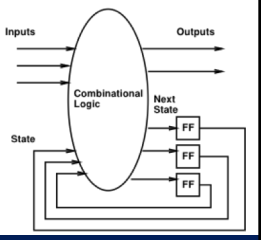
- **Useful when trying to control things**
  - E.g. Perform a sequence of operations
- **Robot**
  - Open-gripper
  - Move-forward
  - Close-gripper
  - Lift

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

FINITE-STATE MACHINE (FSM)

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

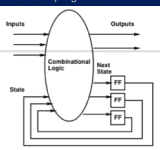


FSM Example

- **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)**

Truth Table Model

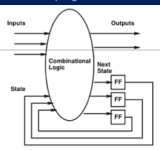
Complete together



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			

Truth Table Model

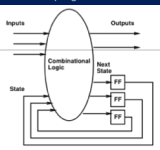
Complete together



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			

Truth Table Model

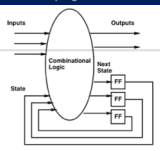
Complete together



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			

Truth Table Model

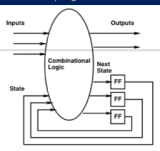
Complete together



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			

Truth Table Model

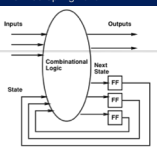
Complete together



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			

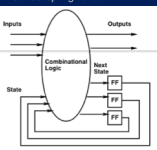
TRUTH TABLE MODEL

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			

TRUTH TABLE MODEL



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

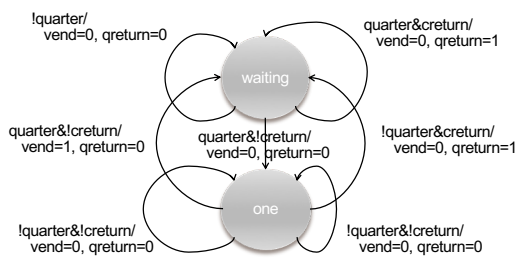
SWITCH-STATEMENT MODEL

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


SWITCH-STATEMENT MODEL (CONT.)

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


FSM GRAPH MODEL

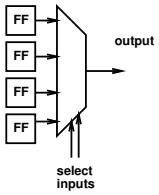


PROGRAMMABLE LOGIC

MUX CAN BE A PROGRAMMABLE GATE

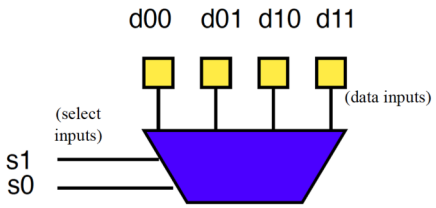
- **Programmable Gate**
  - Can be programmed to act as any gate
  - Use state (e.g. FF) to "program" truth table of a gate

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



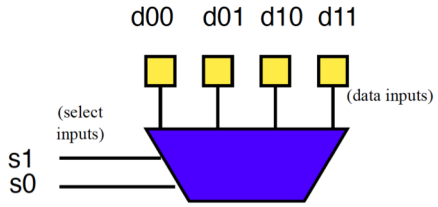
EXAMPLE: AND

- How do we program to behave as AND2?



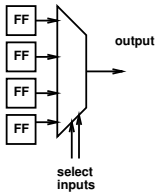
EXAMPLE: OR

- How do we program to behave as OR2?



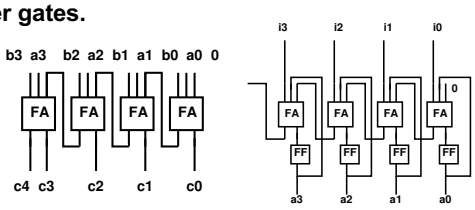
LOOK-UP TABLE (LUT)

- **Can generalize to any number of inputs**



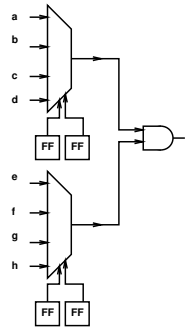
CONNECTING GATES

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

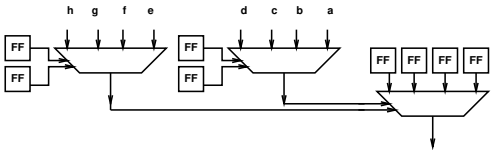


MUX CAN BE PROGRAMMABLE INTERCONNECT

Trick: Use multiplexer to programmably select gate input.

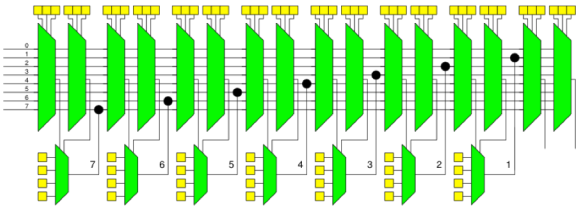


Programmable Blocks



Programmable Gates and Interconnect

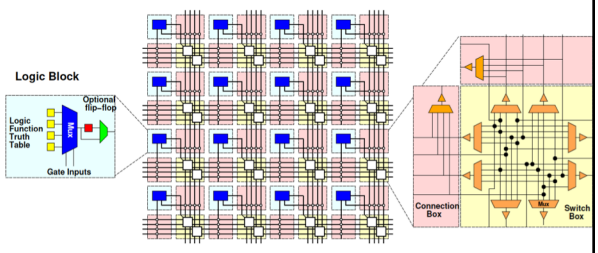
If time permits:
How program (fill in yellow programmable cells) to implement a full adder?



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

Field-Programmable Gate Array (FPGA)



Next Lab

- **Program an FPGA in Verilog**
  - Build an adder
  - Build an accumulator
- **Should have hardware**
- **Will need to install software on your computer**

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

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