EECS 370

Lab 5: Digital Logic

Upcoming Assignments

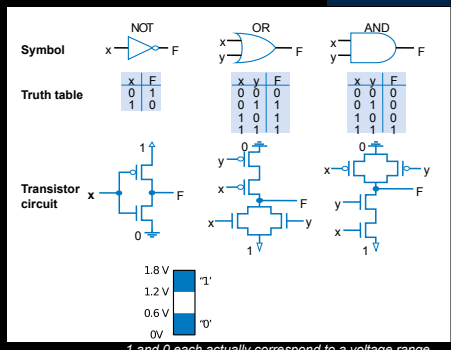
- Lab 5 due Wednesday 10/2
- Project 2a due Thursday 10/3
- Homework 2 due Monday 10/7
- Midterm Exam Wednesday 10/9 at 7 p.m.

What to cover today?

Logic Gates	Problems 1, 2	Slides 4-9
Muxes, Decoders, ALU	Problem 3	Slides 10-18
Propagation Delay	Problem 4	Slides 19-23
Timing Diagrams	Problem 5	Slides 22-23, 27-31
Latches & Flip Flops	Problem 5	Slides 24-31
Finite State Machines	Problems 6, 7	Slides 32-end

Logic Gates

- **Logic Gates** implement Boolean operators using transistors
  - Logic gates form the basis of computers.
  - Implemented under the hood with transistors



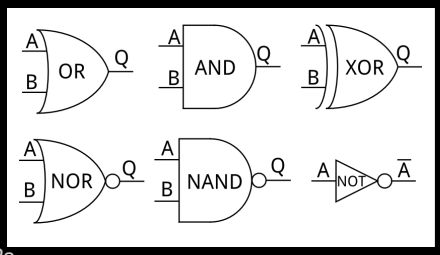
1 and 0 each actually correspond to a voltage range

Common Logic Gates

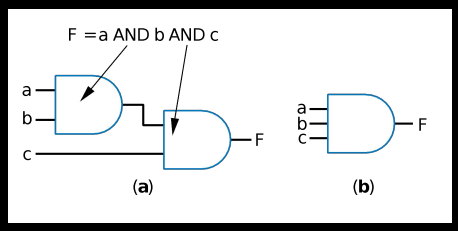
In *combinational logic*, output is determined solely by the inputs

On the right are the 6 most common logic gates

NOR and NAND are both functionally complete -- any boolean function can be constructed from NANDs or NORs



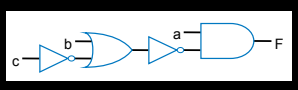
Using gates with more than 2 inputs



Can think of as AND(a,b,c)

Problem 1: Gates to Logical Equation

Convert the following logic gates to a logic equation.
(Hint: Start from the output, work back to the inputs.)



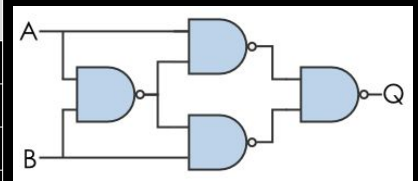
F = a && !(b || !c) //C/C++ Logical Form

Problem 2: Build another Gate out of NAND Gates

Given the below construction from 4 functionally complete NAND gates, what other elementary gate is represented?

NAND Gate Truth Table

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Problem 2: Build another Gate out of NAND Gates

Given the below construction from 4 functionally complete NAND gates, what other elementary gate is represented?

A ^ B = (A & ~B) | (~A & B) Definition

= ~~((A & ~B) | (~A & B)) Double Negation

= ~(~(A & ~B) & ~(~A & B)) DeMorgan's Law

= ~(~(A & ~(A & B)) & ~(~(A & B) & B)) Absorption 2x

~(~(A & ~(A & B)) & ~(~(A & B) & B))

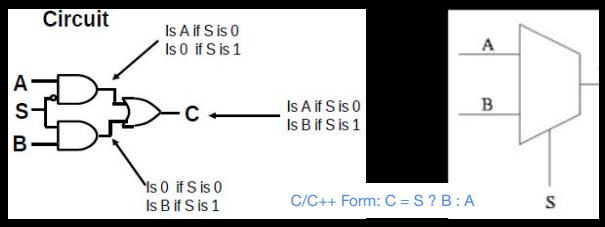
Final Gate Truth Table

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

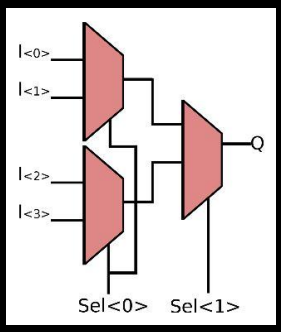
Multiplexers (MUX) Allow the Selection of Inputs

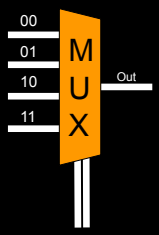
MUXs select one of N multiple inputs, and are controlled by $\log_2(N)$ bits.

We assume that a selection of all 0s selects the top-most input



4 Bit Muxes

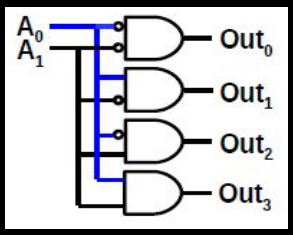




Decoders Turn n-digit Bf mbers to 2^n Outputs

Decoders can be used to select one output from a specific input

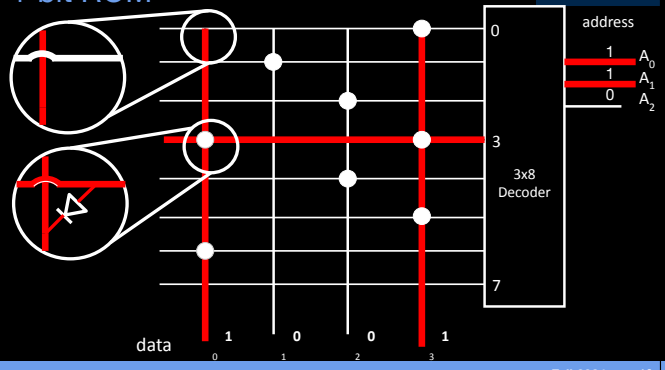
Uses "one-hot" encoding, where only one line can be enabled at once



Input		Output
A_1	A_0	Out_{3-0}
0	0	0001
0	1	0010
1	0	0100
1	1	1000

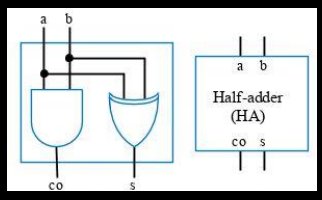
8-entry 4-bit ROM

- A diode only allows current to flow in one direction.
- It prevents a '1' from propagating to other lines.



Adding two 1-bit numbers together

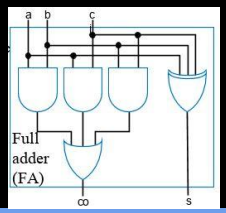
Circuit Adds A + B, outputs Sum and Carry



Inputs		Outputs	
a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Adding two 1-bit numbers together

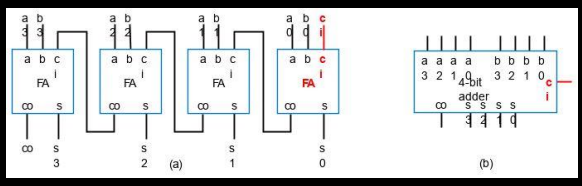
- To use the carry in the next bit, we use a **Full Adder**
- The previous was a **Half Adder**
- Full Adder: Adds A + B **+ Carry**, outputs Sum and Carry
- When Carry In is 0, the Full Adder behaves like a Half



Inputs			Outputs	
a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

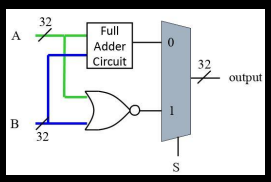
Ripple Carry Adder

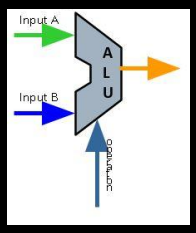
We can combine FAs together to create an adder that performs addition like we do by hand.



Arithmetic Logical Unit (ALU)

- The "heart" of the computer processor
- For LC2K: Can execute either an addition of two (32 bit) numbers, or a noring
- MUX Control Scheme: 0 - Addition; 1 - Nor

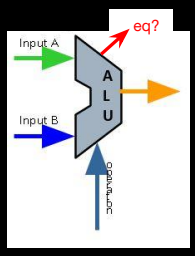




Equality Checking

- Our ALU always checks for equality.
- This can be done with bitwise XNOR, then AND on all bits.
- Also can be done by subtracting A-B and NOR on all bits.

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1



Problem 3: Short Answer Questions

If we have an *n*-input MUX (e.g. A_1, A_2, …, A_n) how many control bits do we need?

$\lceil \log_2(n) \rceil$

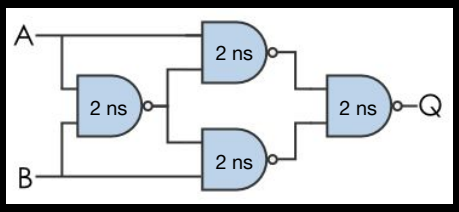
True or False: The number of select lines in an 8x1 MUX is equal to the number of input lines in a 3x8 Decoder

True

Propagation Delay Influences How Circuits Are Built

Physical changes in voltage levels take time to reach the other end

With circuits consisting of multiple paths, have to wait on all paths

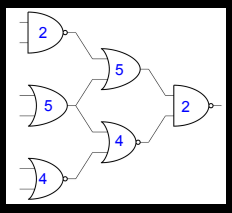


Consider the propagation delay for our XOR gate if each NAND gate takes 2ns

We have a 6ns delay

Problem 4: Calculate the Propagation Delay

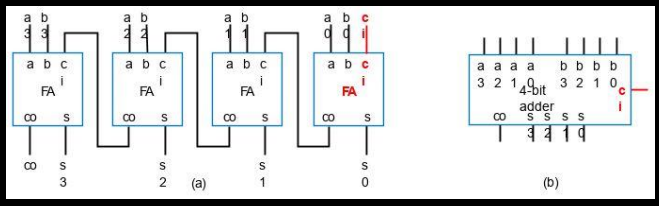
Calculate the propagation delay in the following circuit, assuming the delays are 2ns for NAND, 4ns for NOR, and 5ns for OR



Longest Path:
5ns + 5ns + 2ns = 12ns

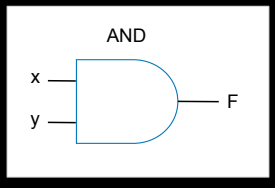
Ripple Carry Adder

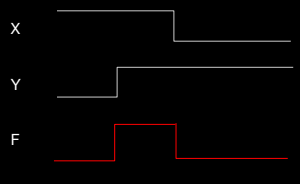
Because there are so many gates within each full adder, the propagation delay can be quite high!



Timing Diagrams

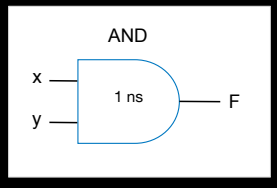
We use timing diagrams to represent the values of outputs given values of inputs at times.

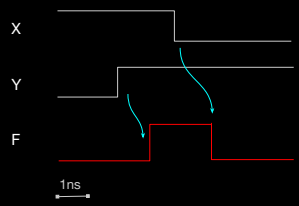




Timing Diagrams with Delay

Timing Diagrams can also represent delay.





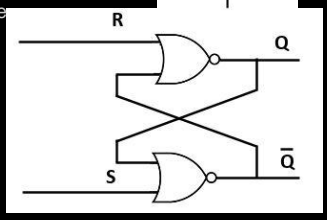
Bit Storage - Implementing Memory

- We want to save the state of a bit/signal
- i.e If we press a button to turn on a light, we want the light to stay on even after we stop pressing the button.
- More relevant example: read/write data in a register file
- Use: Latches & Flip Flops

SR Latch - Implementation of Memory

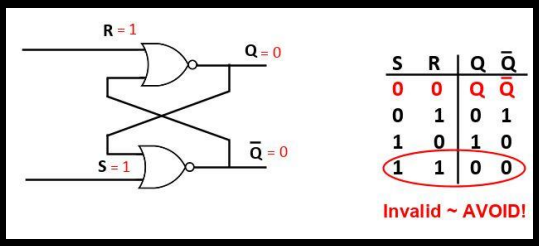
- Two inputs: S (Set) and R (Reset)
- Two outputs: Q and Not Q (opposites)
- SR Latches work because of feedback, notice how outputs from one NOR gate are driving outputs of the other.
- 0 0 state - "holds the current data"
- 0 1 state - reset, writes a 0 to data
- 1 0 state - set writes a 1 to data

S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0



SR Latch - Implementation of Memory

- But what happens when S=1 and R=1?

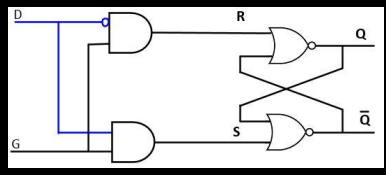


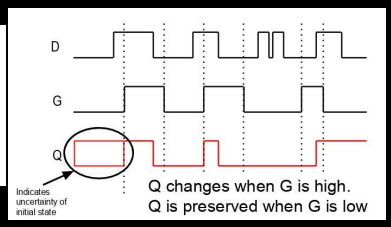
S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	0	0

Invalid ~ AVOID!

D-Latch - Another way to implement Memory

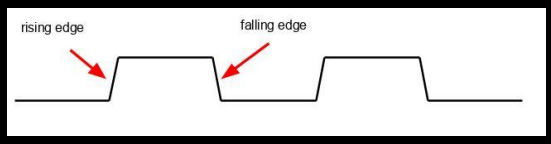
- Two signals being fed in: the Data and a Gating Signal
- State of the output changes while G is 1
- When G is 0, no set or reset: SR latch in state 0 0, so value is held





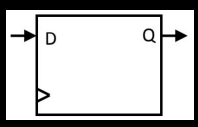
Clocks

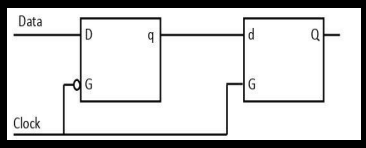
- Periodic pulse. Switches from 0 to 1 periodically (equal intervals)
- We can determine this interval, but it stays constant once set.
- This will be useful when we do operations with different propagation delays

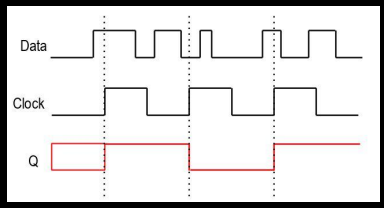


D-Flip Flop

- 2 D Latches chained together
- 2 signals being fed in: the Data and the Clock
- State of the output changes while clock is changing state
- Can be Rising-Edge Triggered OR Falling-Edge Triggered

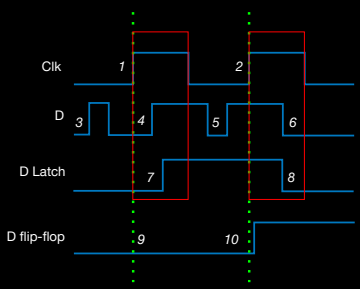






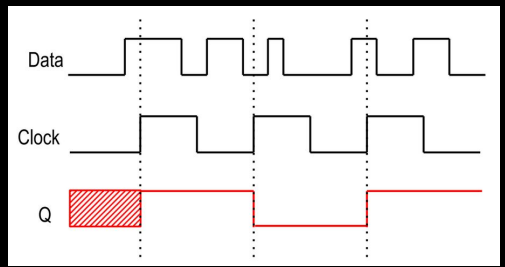
D Latch vs. D Flip-Flop

- Latch is level-sensitive
  - Stores D when Clk=1 (Boxes)
- Flip-flop is edge triggered
  - Stores D when Clk changes from 0 to 1 (Rising Edge, dotted lines)



Problem 5: Timing Diagram for Rising-Edge FF

What does the output signal Q look like for a rising-edge flip-flop?

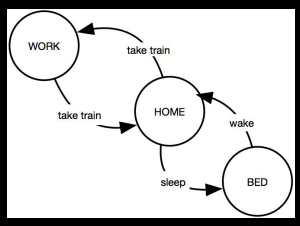


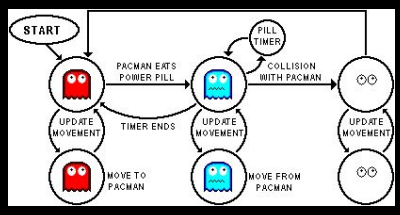
Finite-State Machine Definition

Describes desired behavior of sequential circuit.
- Like Boolean equations for combinational behavior.

FSM consists of:
- Set of inputs & outputs
  - These are digital logic: bits, wires, numbers
- Set of states
  - Each specifies output values
  - One is the Initial state, Ex: “Off”
- Set of transitions
  - Transition: <State, Input Value, Next State>

FSM Representations





We often draw FSM graphically as a ***state diagram***
Can also use **state table** or textual languages

How To Capture Desired Behavior as FSM

- *List states*
  - Give meaningful names, show initial state
  - Optionally add some transitions if they help
- *Create transitions*
  - For each state, define **all possible** transitions leaving that state.
    - Make a transition for every possible input value.
- *Refine the FSM*
  - Execute the FSM mentally and make any needed improvements.

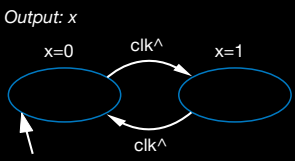
Capturing Sequential Circuit Behavior as FSM

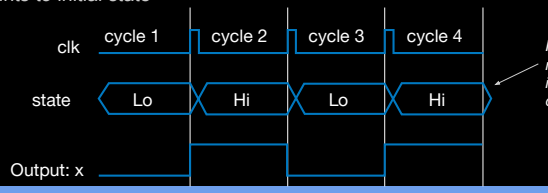
Example: Toggle x every clock cycle

States: “Lo” (x=0), and “Hi” (x=1)

Transition from on rising clock edge (clk^)

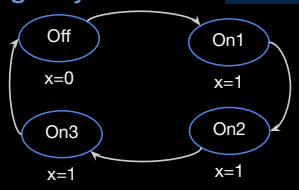
Arrow points to initial state

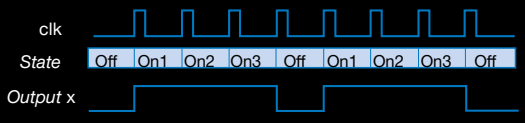




FSM Example: Three Cycles High System

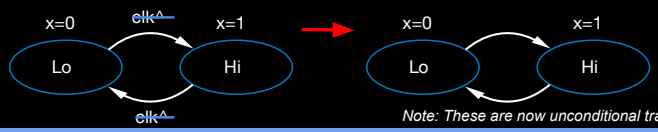
- Repeat 0, 1, 1, 1...
  - For one clock cycle each
- FSM
  - Four states: 0, first 1, second 1, third 1
  - Transition on rising clock edge to next state





FSM Simplification: Rising Clock Edges Implicit

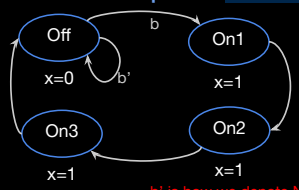
- In this course, assume every transition is on the clock edge.
- What if we wanted a transition *without* a rising edge?
  - Such asynchronous FSMs are less common, but are a more advanced topic
- In this course, only consider ***synchronous*** FSMs.



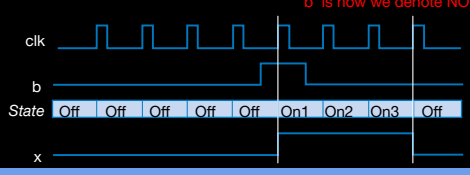
Note: These are now unconditional transitions

Three-Cycles High System with Button Input

- Same 4 states
- Wait in “Off” while b is 0
- When b is 1, go to On1
  - Sets x=1
  - Then go to On2, On3, Off again
- x=1 for 3 cycles



b' is how we denote NOT b



FSM: Moore vs. Mealy Machines

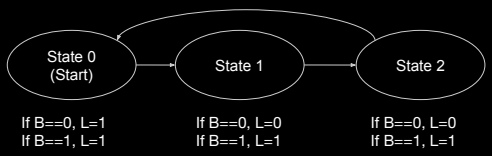
- **Moore Machine**
  - Output function based on **current state** P(S) only
- **Mealy Machine**
  - Output function based on **current state** *and* **current input** P(S, I)

Question: which generally has more states?

Problem 6a: Design a Mealy Machine

Design a Mealy finite-state machine for a button-activated LED.

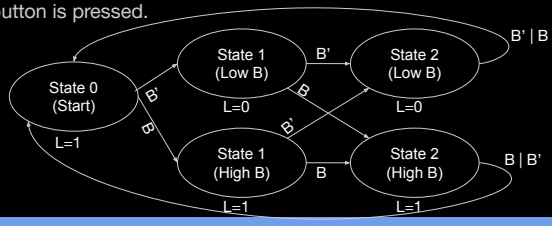
The light should remain lit while a button *B* is constantly pressed, or blink every third cycle while the button is not pressed. Assume the timer does not reset when the button is pressed.



Problem 6b: Design a Moore Machine

Design a Moore finite-state machine for a button-activated LED.

The light should remain lit while a button *B* is constantly pressed, or blink every third cycle while the button is not pressed. Assume the timer does not reset when the button is pressed.



Problem 7a: FSM Table

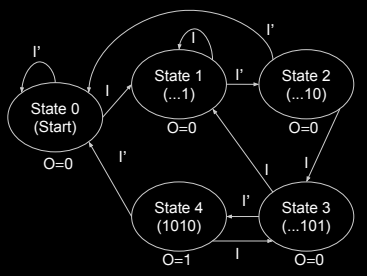
Design a finite state machine which solves the following problem and fill out the following table.
There is one input, I, and one output, O. O is *high* if the last 4 inputs on I were 1010 (the rightmost 0 is the most recent input bit).

Current State #	Next State when I = 0	Next State when I = 1	Output O (0/1)
000 (no bits)	000	001	0
001 (rec'vd 1)	010	001	0
010 (...10)	000	011	0
011 (...101)	100	001	0
100 (...1010)	000	011	1

Problem 7b: FSM Diagram

Design a finite state machine which solves the following problem and fill out the following tables. There is one input, I, and one output, O. O is high if the last 4 inputs on I were 1010 (the rightmost 0 is the most recent input bit).

b. Now, create your diagram following the table we just made.



FSM Example: Secure Car Key

- Many new car keys include tiny computer chip

When key turned, car's computer (under engine hood) requests identifier from key

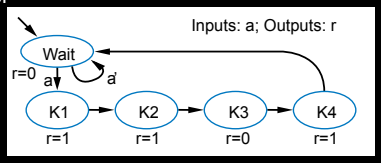
Key transmits identifier

Else, computer doesn't start car

- FSM

Wait until computer requests ID (a=1)

Transmit ID (in this case, 1 1 0 1)

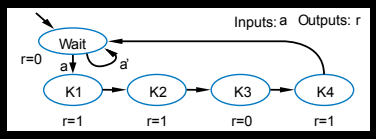


FSM Example: Secure Car Key

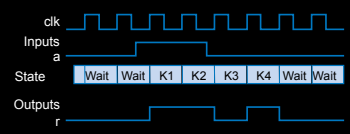
- Nice feature of FSM

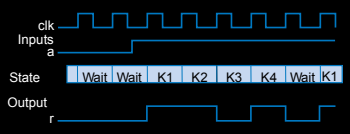
Can evaluate output behavior for different input sequence

Timing diagrams show states and output values for different input waveforms



Q: Determine states and r value for given input waveform:





FSM Capture Example: Code Detector

- Unlock door (u=1) only when buttons pressed in sequence:

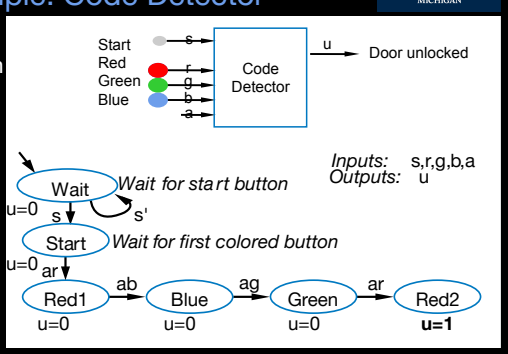
start, then red, blue, green, red

- Input from each button: *s, r, g, b*

Also, output *a* indicates that some colored button pressed

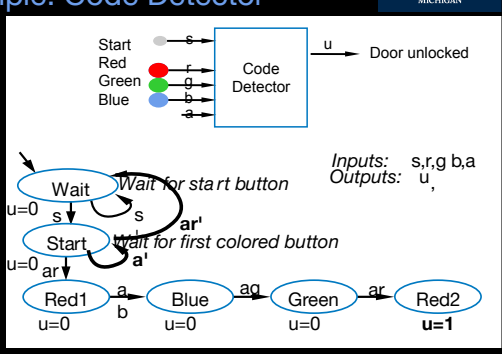
- Capture as FSM
  - ***List states***

Some transitions included



FSM Capture Example: Code Detector

- Capture as FSM
  - *List states*
  - ***Create transitions***



FSM Capture Example: Code Detector

- Capture as FSM
  - *List states*
  - *Create transitions*
    - Repeat for remaining states
  - Refine FSM
    - Mentally execute
    - Works for normal sequence
    - Check unusual cases
    - All colored buttons pressed
      - Door opens!
      - Change conditions: other buttons NOT pressed also