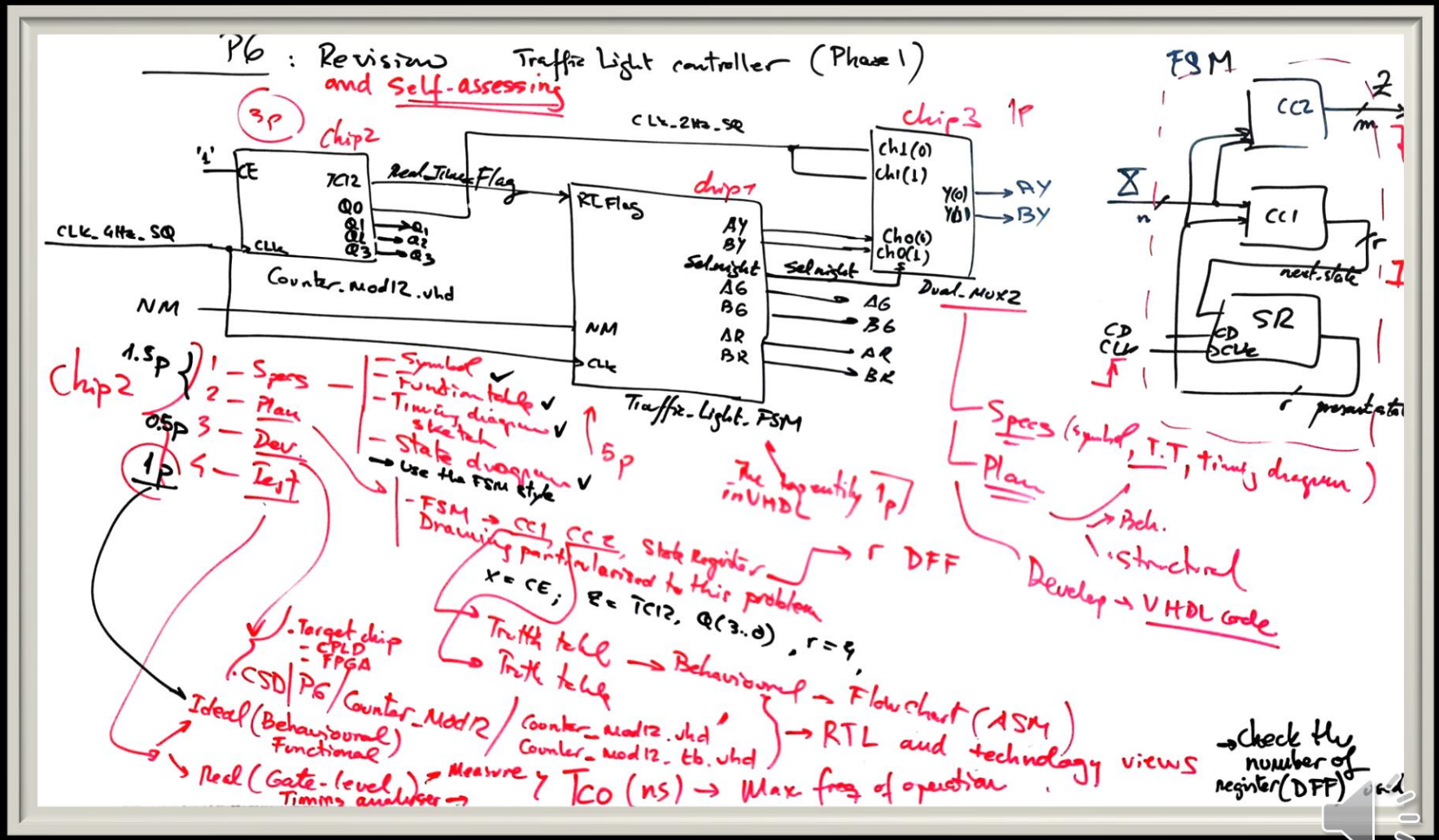
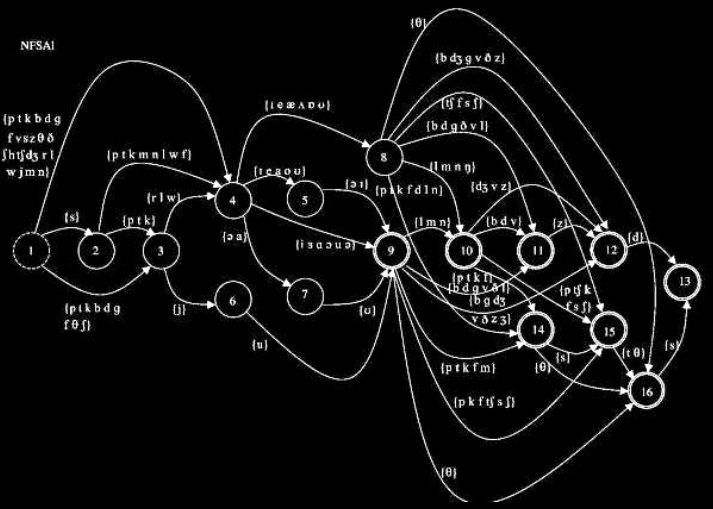
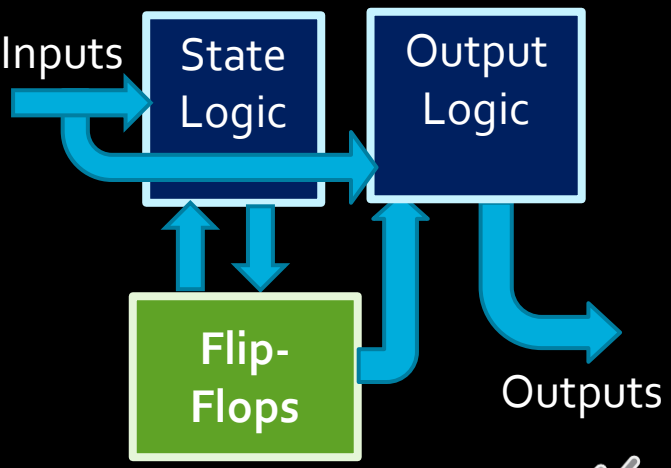
Week 5, part D: FSM Design



FSM Design Steps

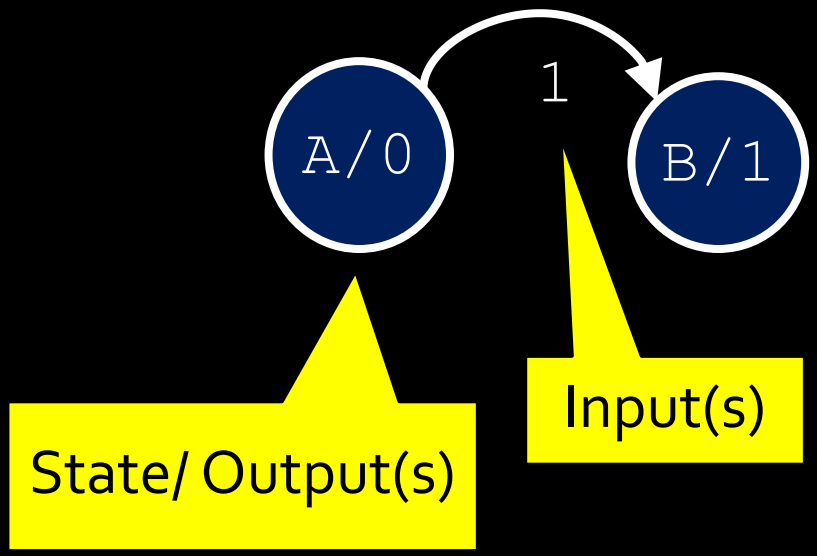
1. Draw state diagram
2. Derive state table from state diagram
3. Assign flip-flop value configuration to each state
   - At least $\lceil \log_2(\#\text{ of states}) \rceil$ flip-flops
4. Redraw state table with flip-flop values
5. Derive combinational circuit for output and for each flip-flop input
   - State logic: figure out next state
   - Output logic: figure out output

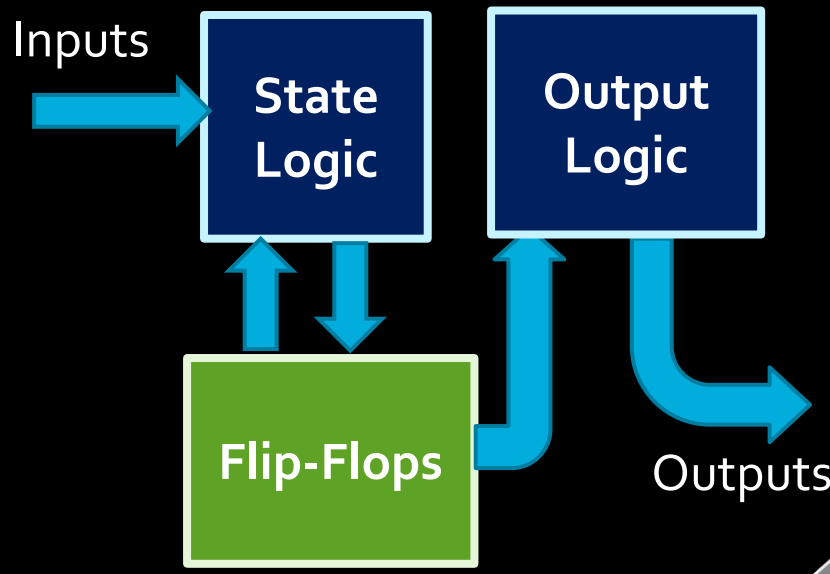




State Diagrams with output

- For now, lets assume output only depends on current state.





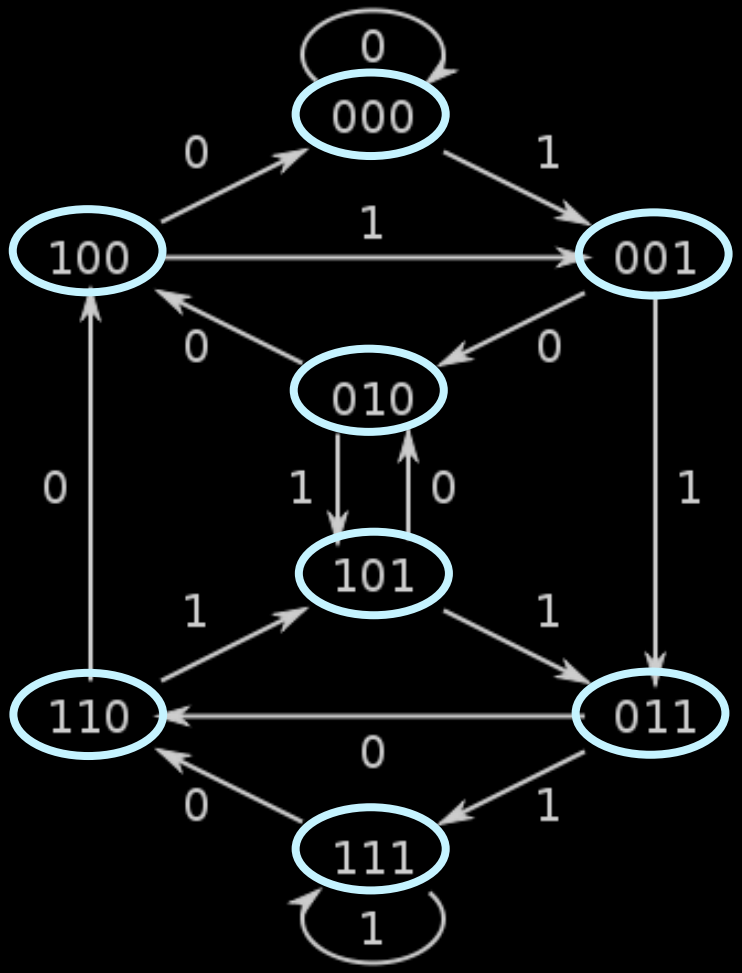
Example #4: Sequence Recognizer

- Recognize a sequence of input values, and raise a signal if that input has been seen.
- Example: Three high values in a row
  - Detect that the input has been high for three rising clock edges.
  - Assumes a single input `IN` and a single output `Z`.

- What are the states?

Step 1: State diagram

- In this case, we will label states with the three most recent input bits.
- Transitions between states depends on the input `IN`.
- Indicated by the values on the transition arrows.



Step 2: State table

- Make sure that the state table lists all the states in the state diagram, and all the possible inputs that can occur at that state.

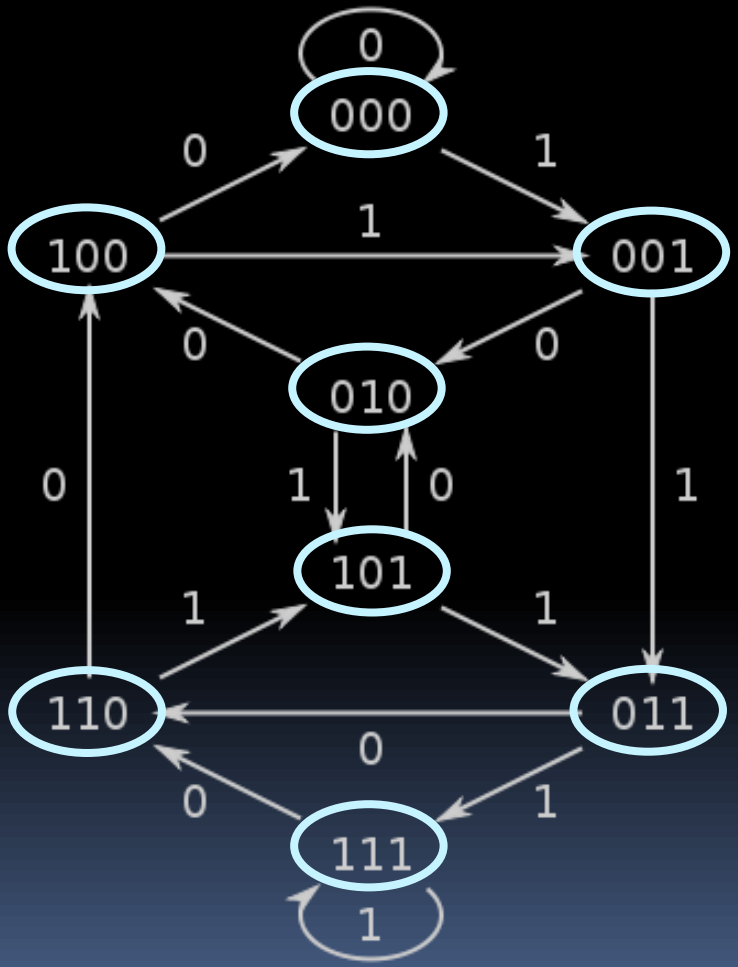
Previous State	Input	Next State
"000"	0	"000"
"000"	1	"001"
"001"	0	"010"
"001"	1	"011"
"010"	0	"100"
"010"	1	"101"
"011"	0	"110"
"011"	1	"111"
"100"	0	"000"
"100"	1	"001"
"101"	0	"010"
"101"	1	"011"
"110"	0	"100"
"110"	1	"101"
"111"	0	"110"
"111"	1	"111"

Step 3: Assign flip-flops

- The flip-flops are responsible for storing state.
- A single flip-flop can store either 0 or 1
  → supports two states.
- How many states can be stored with each additional flip-flop?
  - One flip-flop → 2 states
  - Two flip-flops → 4 states
  - Three flip-flops → 8 states
  - …
  - Eight flip-flops? → $2^8 = 256$ states

For n states we need $\lceil \log_2(n) \rceil$ flip-flops

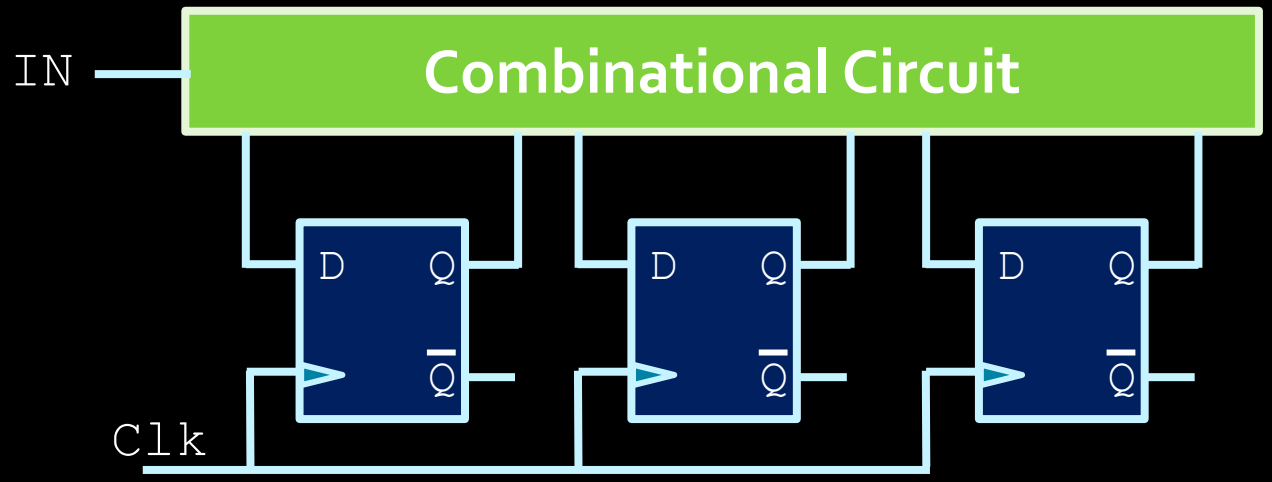
How many flip-flops for this one?



3

Step 3: Assign flip-flops

- In this case, we need to store 8 states.
  - 8 states = 3 flip-flops ($3 = \log_2 8$)
- For now, assign a flip-flop to each digit of the state names in the FSM & state table.



Step 4: State table

- Usually, the states have names that don't map over to flip-flops so easily.

Prev. State	IN	Next State
"000"	0	"000"
"000"	1	"001"
"001"	0	"010"
"001"	1	"011"
"010"	0	"100"
"010"	1	"101"
"011"	0	"110"
"011"	1	"111"
"100"	0	"000"
"100"	1	"001"
"101"	0	"010"
"101"	1	"011"
"110"	0	"100"
"110"	1	"101"
"111"	0	"110"
"111"	1	"111"

Step 4: State table

- Usually, the states have names that don't map over to flip-flops so easily.
- It may be an easy mapping, but is it a good one?
  - Not really, but we'll get to why later.

prev state (input)				next state (output)		
F_2	F_1	F_0	IN	F_2	F_1	F_0
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	1
0	1	0	0	1	0	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	0	1	1
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	1	1	0
1	1	1	1	1	1	1

Step 5: Circuit design

- Karnaugh map for $F_{2\,(next)}$:

	$\overline{F_0} \cdot \overline{IN}$	$\overline{F_0} \cdot IN$	$F_0 \cdot IN$	$F_0 \cdot \overline{IN}$
$\overline{F_2} \cdot \overline{F_1}$	0	0	0	0
$\overline{F_2} \cdot F_1$	1	1	1	1
$F_2 \cdot F_1$	1	1	1	1
$F_2 \cdot \overline{F_1}$	0	0	0	0

$$F_{2\,(next)} = F_1$$

Next state

Current state

Step 5: Circuit design

- Karnaugh map for $F_{1\,(next)}$:

	$\overline{F_0} \cdot \overline{IN}$	$\overline{F_0} \cdot IN$	$F_0 \cdot IN$	$F_0 \cdot \overline{IN}$
$\overline{F_2} \cdot \overline{F_1}$	0	0	1	1
$\overline{F_2} \cdot F_1$	0	0	1	1
$F_2 \cdot F_1$	0	0	1	1
$F_2 \cdot \overline{F_1}$	0	0	1	1

$$F_{1\,(next)} = F_0$$

Step 5: Circuit design

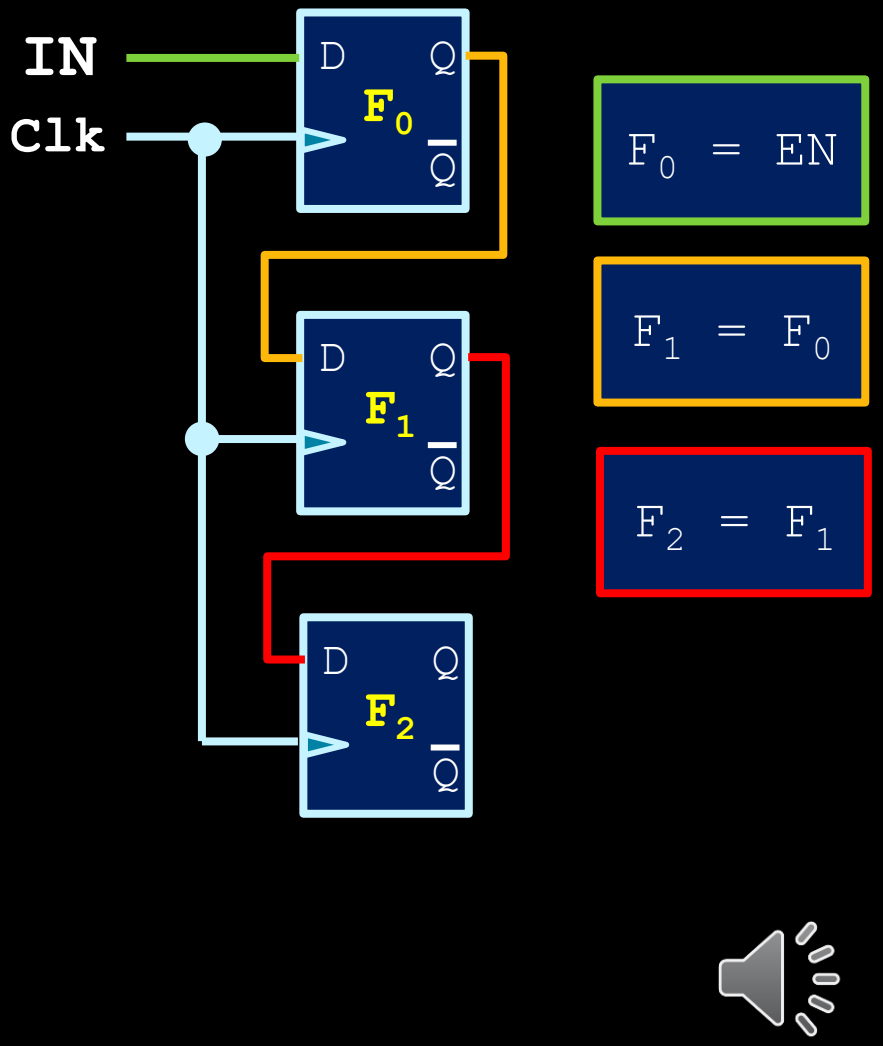
- Karnaugh map for $F_{0\,(next)}$:

	$\overline{F_0} \cdot \overline{IN}$	$\overline{F_0} \cdot IN$	$F_0 \cdot IN$	$F_0 \cdot \overline{IN}$
$\overline{F_2} \cdot \overline{F_1}$	0	1	1	0
$\overline{F_2} \cdot F_1$	0	1	1	0
$F_2 \cdot F_1$	0	1	1	0
$F_2 \cdot \overline{F_1}$	0	1	1	0

$$F_{0\,(next)} = IN$$

Step 5: Circuit design

- Resulting circuit looks like the circuit on the right →
- Will record the state and make the state transitions happen based on the input
- It makes sense: we want to look at the last 3 bits.
  - **F_2 = old F_1**
  - **F_1 = old F_0**
  - **F_0 = IN**

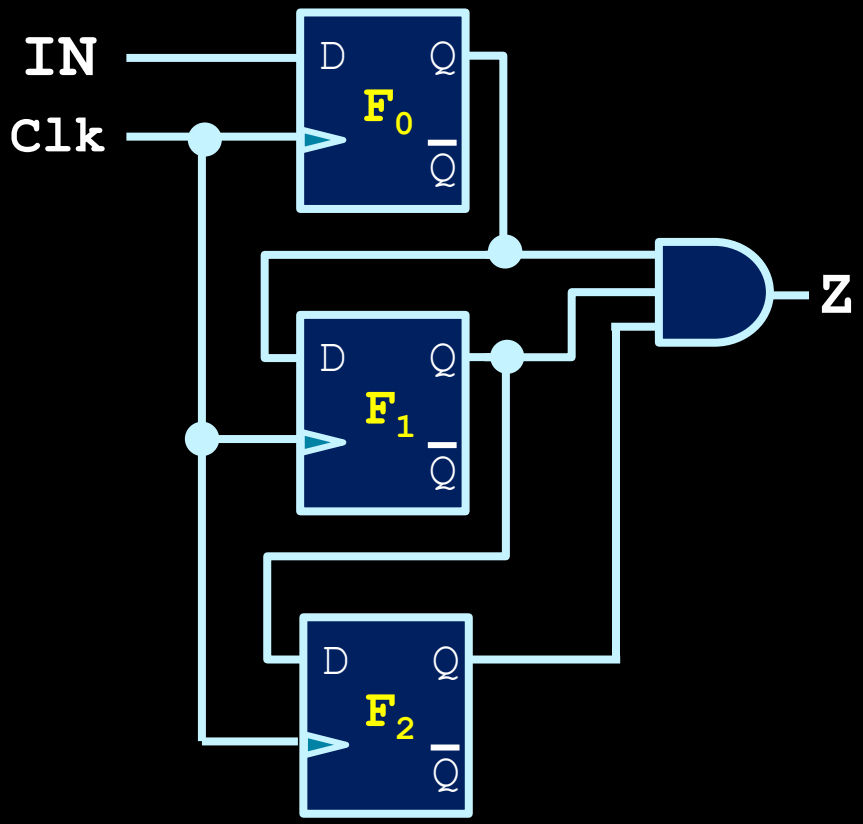


Step 5: Circuit design

- What about the output Z?
  - Z should go high when we see three high in a row.
  - When state is 111

- Boolean equation for Z:

$$Z = F_0 \cdot F_1 \cdot F_2$$

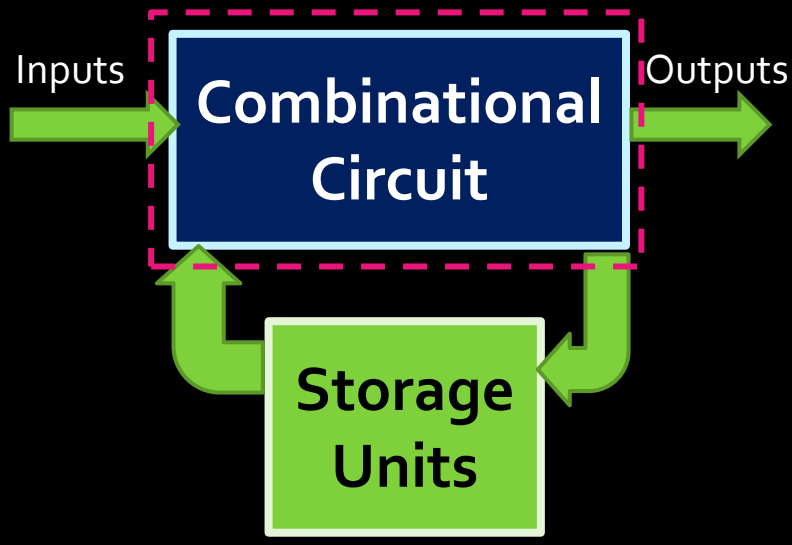


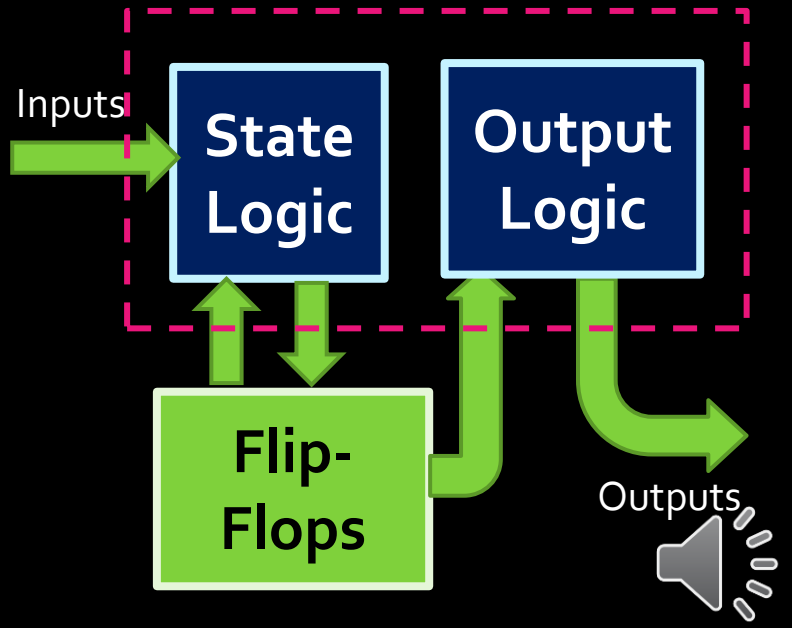
Defining outputs

- Two ways to design the output of a state machine:
  - Moore machine:
    - The output for the FSM depends solely on the current state.
  - Mealy machine:
    - FSM output depends on both the state and the input (based on input actions).
    - Being in state X can result in different output, depending on the input that caused that state.

Moore Machines

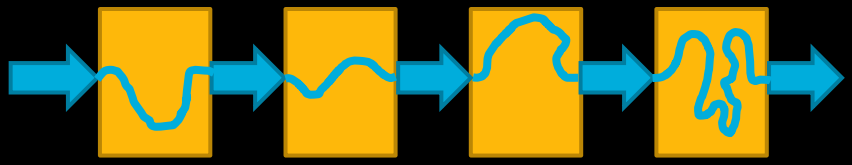
- Output is determined solely based on current state
  - Flip-flop values.
- For simplicity, most of our examples will focus on Moore machines.

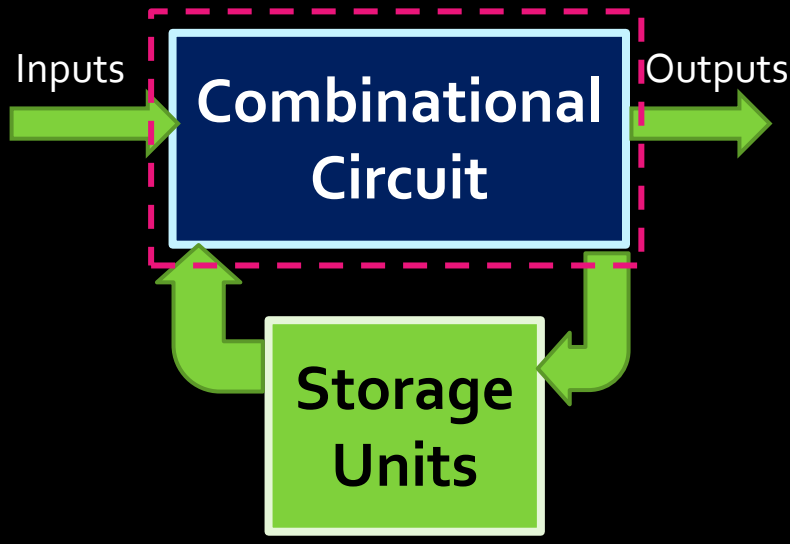


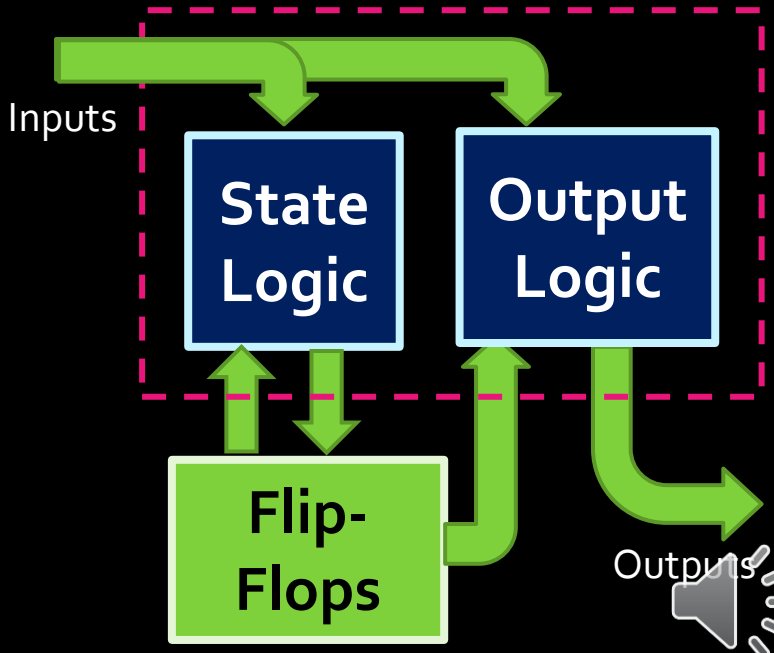


Mealy Machines

- Output is determined by both the current state and the current input values.
- Long paths of connected Mealy machines may be combinatorial!



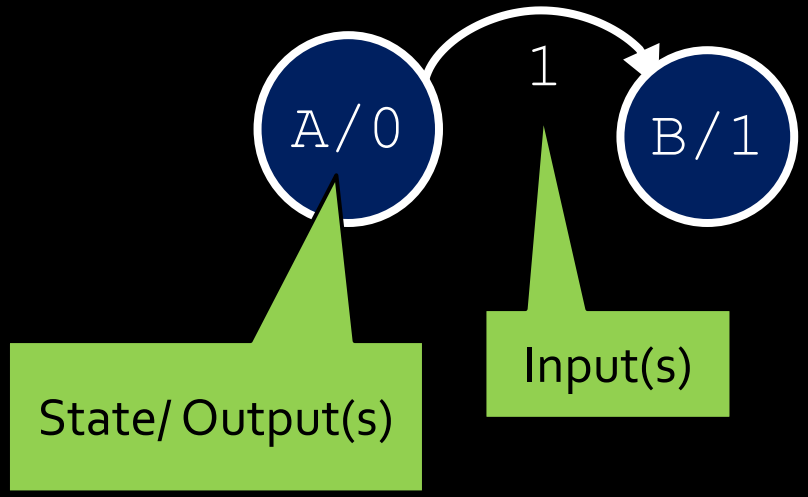




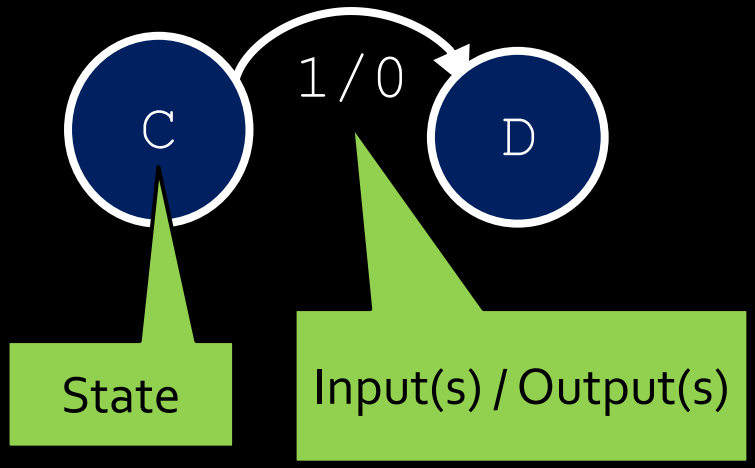
State diagrams with output

- Output values are incorporated into the state diagram, depending on the type of machine.

➢ Moore Machine



➢ Mealy Machine



Moore vs Mealy

Moore

Pros:

- A bit simpler.
- Always a flip-flop between input and output
  - No long combinatorial paths:

Cons:

- May need more states and flip-flops.
- Takes at least one cycle to respond to input

Mealy

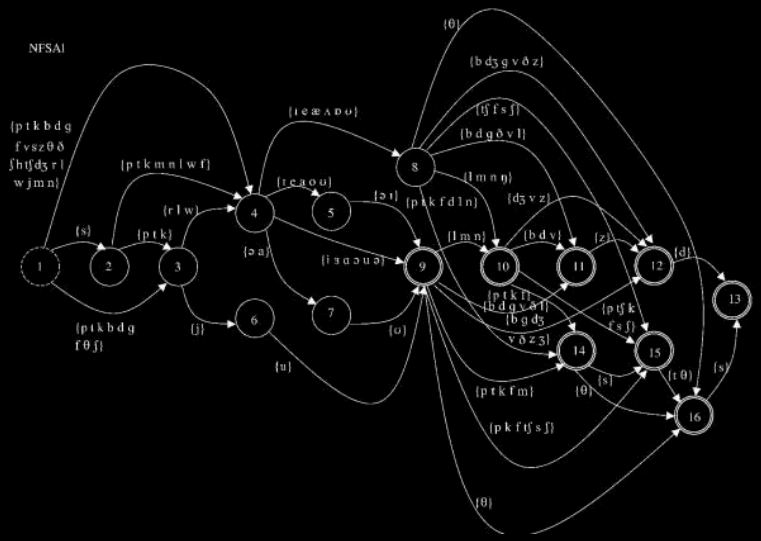
Cons:

- Potentially long path from input to output does not go through a flip-flop.
  - Output is transparent

Pros:

- Sometimes need fewer states / FFs for same task
- Output can respond sooner to inputs.

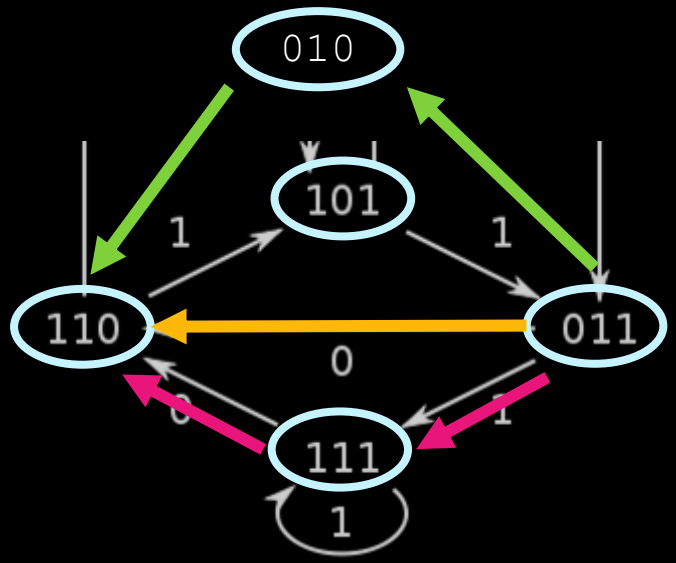
FSM design



- Design steps for FSM:
  1. Draw state diagram
  2. Derive state table from state diagram
  3. Assign flip-flop configuration to each state
  4. Redraw state table as truth table using flip-flop values
  5. Derive combinational circuit for output and for each flip-flop.

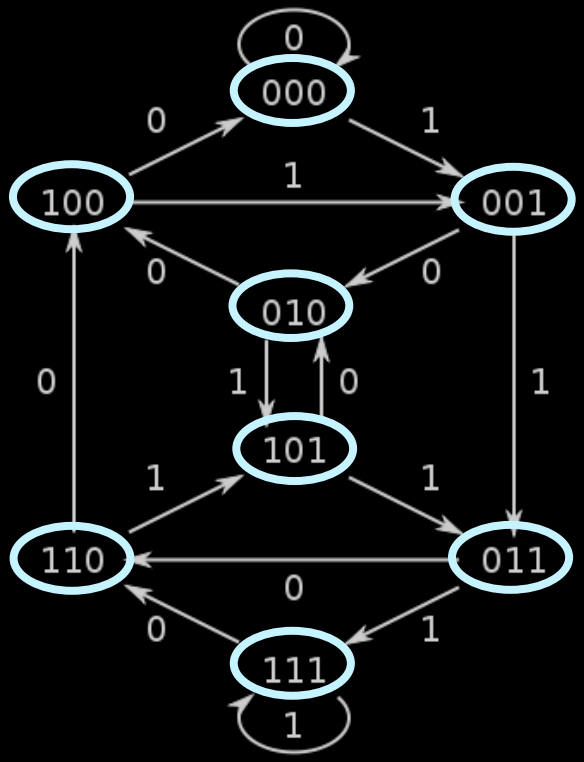
Timing and Unsafe Transitions

- Example: if recognizer circuit is in state **011** and gets the input **0**, it moves to state **110**.
- The first and last flipflops should change "at the same time", but they can't.
- This race condition can be unsafe.
  - If the first flip-flop changes first, the state will change to **111**, and the output would go high for an instant, which is unexpected behaviour.
  - If the second flip-flop changes first, it is fine since the intermediate state **010** has the same low output.



Timing and state assignments

- So how do you solve this?
- Possible solutions:
  1. Whenever possible, make flip-flop assignments such that neighbouring states differ by at most one flip-flop value (state encoding differs by one bit).
  2. If "intermediate" state output is the same as starting or destination state → no problem
  3. Add intermediate transition states between start and end
     - (Use unused flip-flop states or may need to add more)



Previous State	IN	Next State
000	0	000
000	1	001
001	0	010
001	1	011
010	0	100
010	1	101
011	0	110
011	1	111
100	0	000
100	1	001
101	0	010
101	1	011
110	0	100
110	1	101
111	0	110
111	1	111

State ooo does not have to have flip-flop values ooo, it can be anything you want to assign.

Try to re-assign the states so the timing issue doesn't exist

Food for thought?

Try to think about the following at home:

1. Can you re-assign the flip-flop confiruration to avoid the dangerous transition?
2. How many states do you need to recognize an n-bit pattern?
3. Can we do better?
   - Come to the review on Tuesday!

Question #4

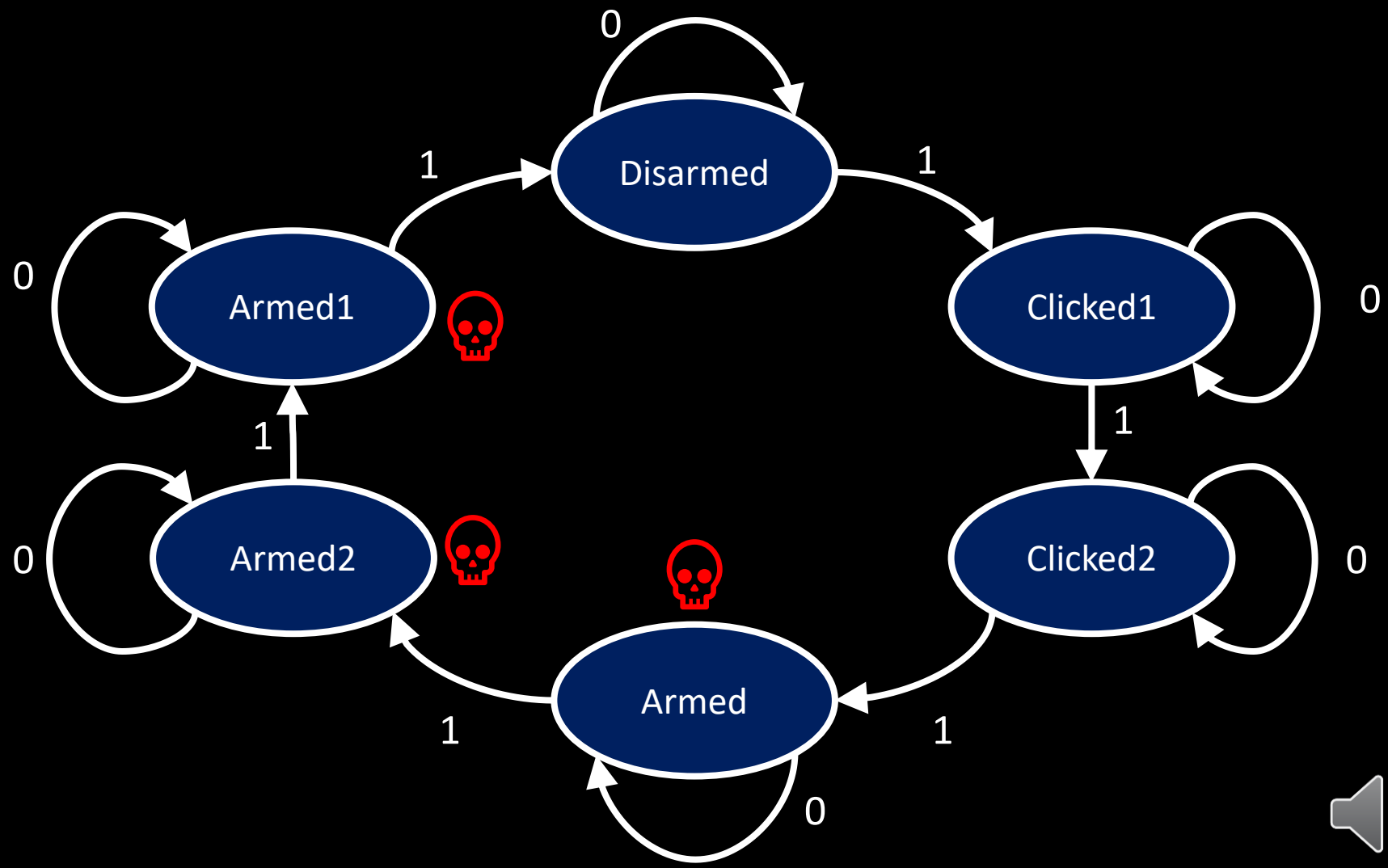
- How would we make the following Finite State Machine?

https://youtu.be/Vi4LmILZUog

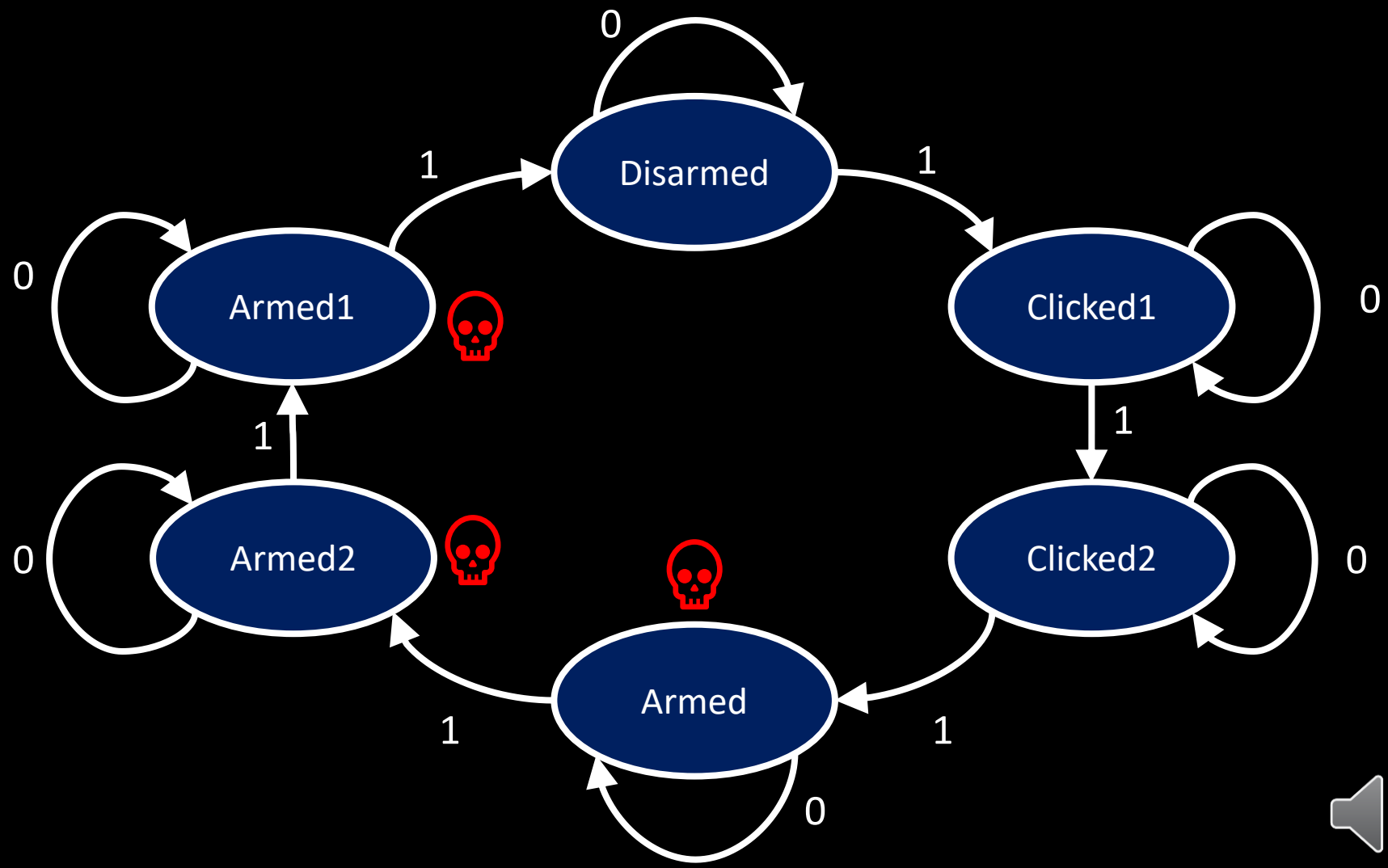
Example #5

- Exploding Pen
  - Starts disarmed
  - 3 clicks to arm
  - 3 clicks to disarm
- Is this a good design?
- How do we build it?
  - Note: Please do not use the knowledge you've gained in this course to develop exploding pens.
  - Note 2: If you do, please don't use them for evil

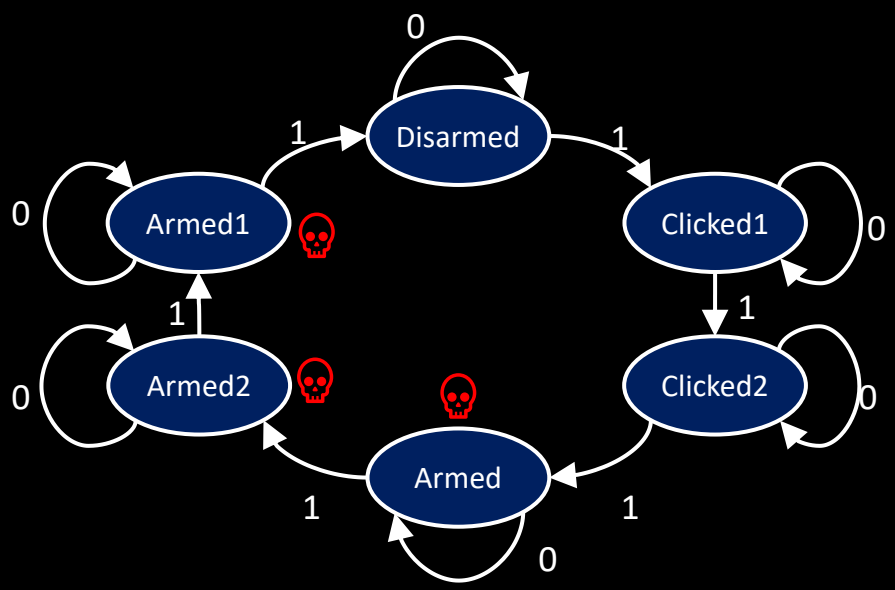
Step 1: State Diagram



Step 1: State Diagram

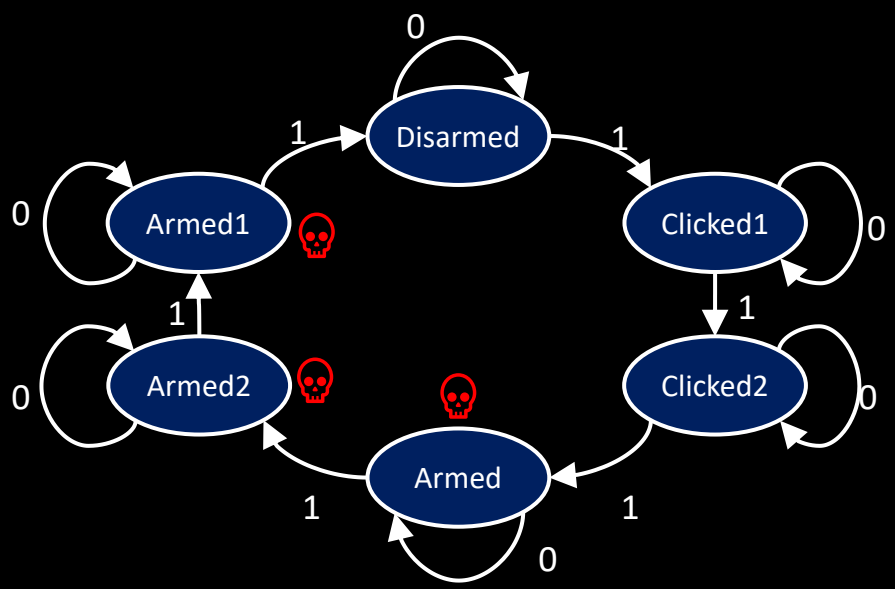


Step 2: State Table



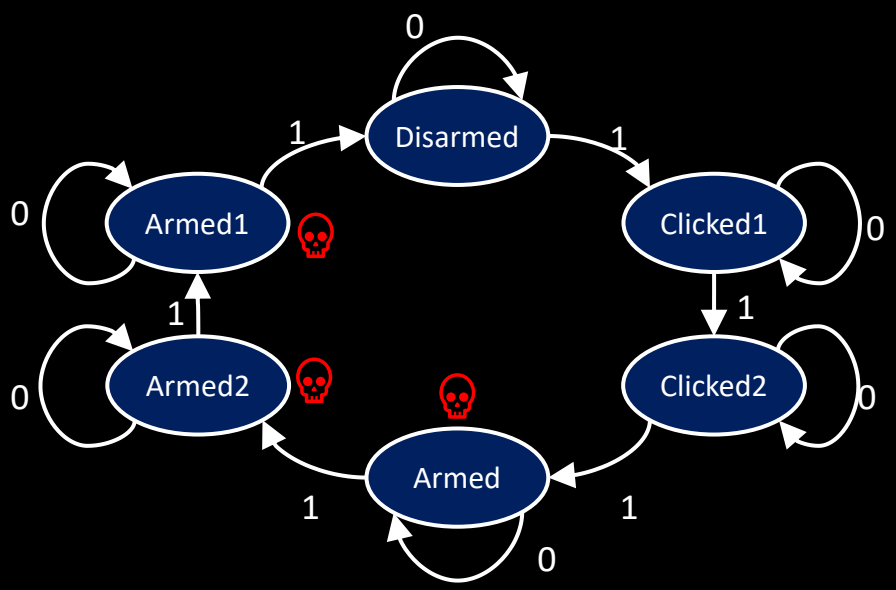
Previous State	Input	Next State

Step 2: State Table



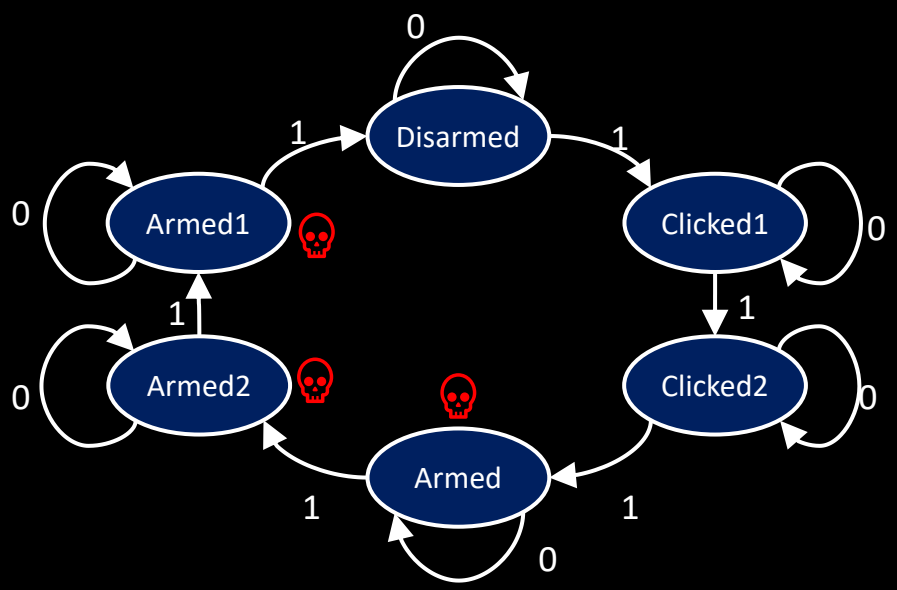
Previous State	Input	Next State
Disarmed	0	
Disarmed	1	

Step 2: State Table



Previous State	Input	Next State
Disarmed	0	
Disarmed	1	
Clicked1	0	
Clicked1	1	
Clicked2	0	
Clicked2	1	
Armed	0	
Armed	1	
Armed2	0	
Armed2	1	
Armed1	0	
Armed1	1	

Step 2: State Table



Previous State	Input	Next State
Disarmed	0	Disarmed
Disarmed	1	Clicked1
Clicked1	0	Clicked1
Clicked1	1	Clicked2
Clicked2	0	Clicked2
Clicked2	1	Armed
Armed	0	Armed
Armed	1	Armed2
Armed2	0	Armed2
Armed2	1	Armed1
Armed1	0	Armed1
Armed1	1	Disarmed

Step 3: FlipFlop Assignment

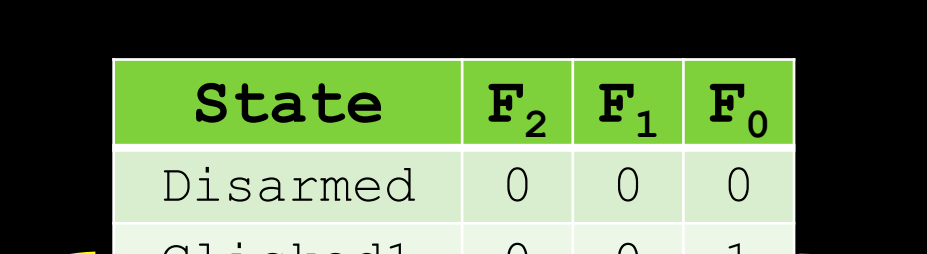
- Let's just try the standard binary order.

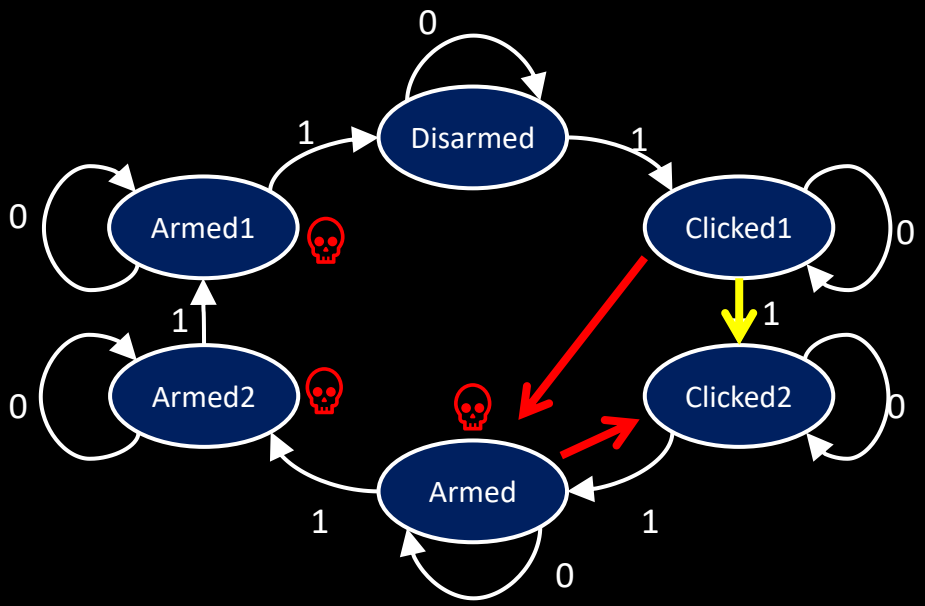
State	F_2	F_1	F_0
Disarmed	0	0	0
Clicked1	0	0	1
Clicked2	0	1	0
Armed	0	1	1
Armed2	1	0	0
Armed1	1	0	1

Step 3: FlipFlop Assignment

- Not safe!
- Transition from Clicked1 to Clicked 2 might go through Armed!

State	F_2	F_1	F_0
Disarmed	0	0	0
Clicked1	0	0	1
Clicked2	0	1	0
Armed	0	1	1
Armed2	1	0	0
Armed1	1	0	1





Step 3: FlipFlop Assignment

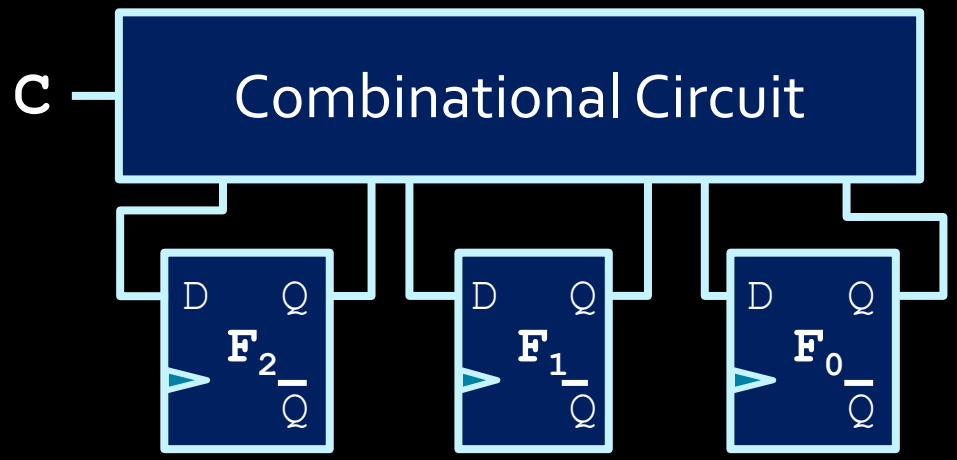
- Much nicer: one bit change for all transitions.
- Also: F2 is high if and only if state is armed!

State	F_2	F_1	F_0
Disarmed	0	0	0
Clicked1	0	0	1
Clicked2	0	1	1
Armed	1	1	1
Armed2	1	1	0
Armed1	1	0	0

Step 4: Update State Table

F_2	F_1	F_0	Input	F_2	F_1	F_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	0
1	1	0	0	1	1	0
1	1	0	1	1	0	0
1	0	0	0	1	0	0
1	0	0	1	0	0	0

Step 5: Circuit Design



F_0	$\overline{F_0}\overline{C}$	$\overline{F_0}C$	F_0C	$F_0\overline{C}$
$\overline{F_2}\overline{F_1}$	0	1	1	1
$\overline{F_2}F_1$	X	X	1	1
F_2F_1	0	0	0	1
$F_2\overline{F_1}$	0	0	X	X

F_1	$\overline{F_0}\overline{C}$	$\overline{F_0}C$	F_0C	$F_0\overline{C}$
$\overline{F_2}\overline{F_1}$	0	0	1	0
$\overline{F_2}F_1$	X	X	1	1
F_2F_1	1	0	1	1
$F_2\overline{F_1}$	0	0	X	X

F_2	$\overline{F_0}\overline{C}$	$\overline{F_0}C$	F_0C	$F_0\overline{C}$
$\overline{F_2}\overline{F_1}$	0	0	0	0
$\overline{F_2}F_1$	X	X	1	0
F_2F_1	1	1	1	1
$F_2\overline{F_1}$	1	0	X	X

Circuit Design

State Logic

$$F_2 = F_2\overline{Click} + F_1Click$$

$$F_1 = F_1\overline{Click} + F_0Click$$

$$F_0 = F_0\overline{Click} + \overline{F_2}Click$$

Output Logic

$$Y = F_2$$

Circuit Design

State Logic

$$F_2 = F_2\overline{Click} + F_1 Click$$

$$F_1 = F_1\overline{Click} + F_0 Click$$

$$F_0 = F_0\overline{Click} + \overline{F_2} Click$$

Output Logic

$$Y = F_2$$

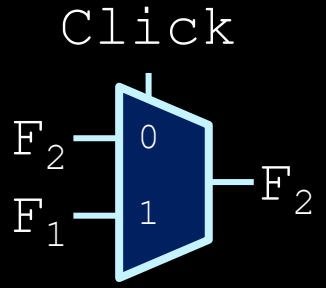
Circuit Design

State Logic

$$F_2 = F_2\overline{Click} + F_1Click$$

$$F_1 = F_1\overline{Click} + F_0Click$$

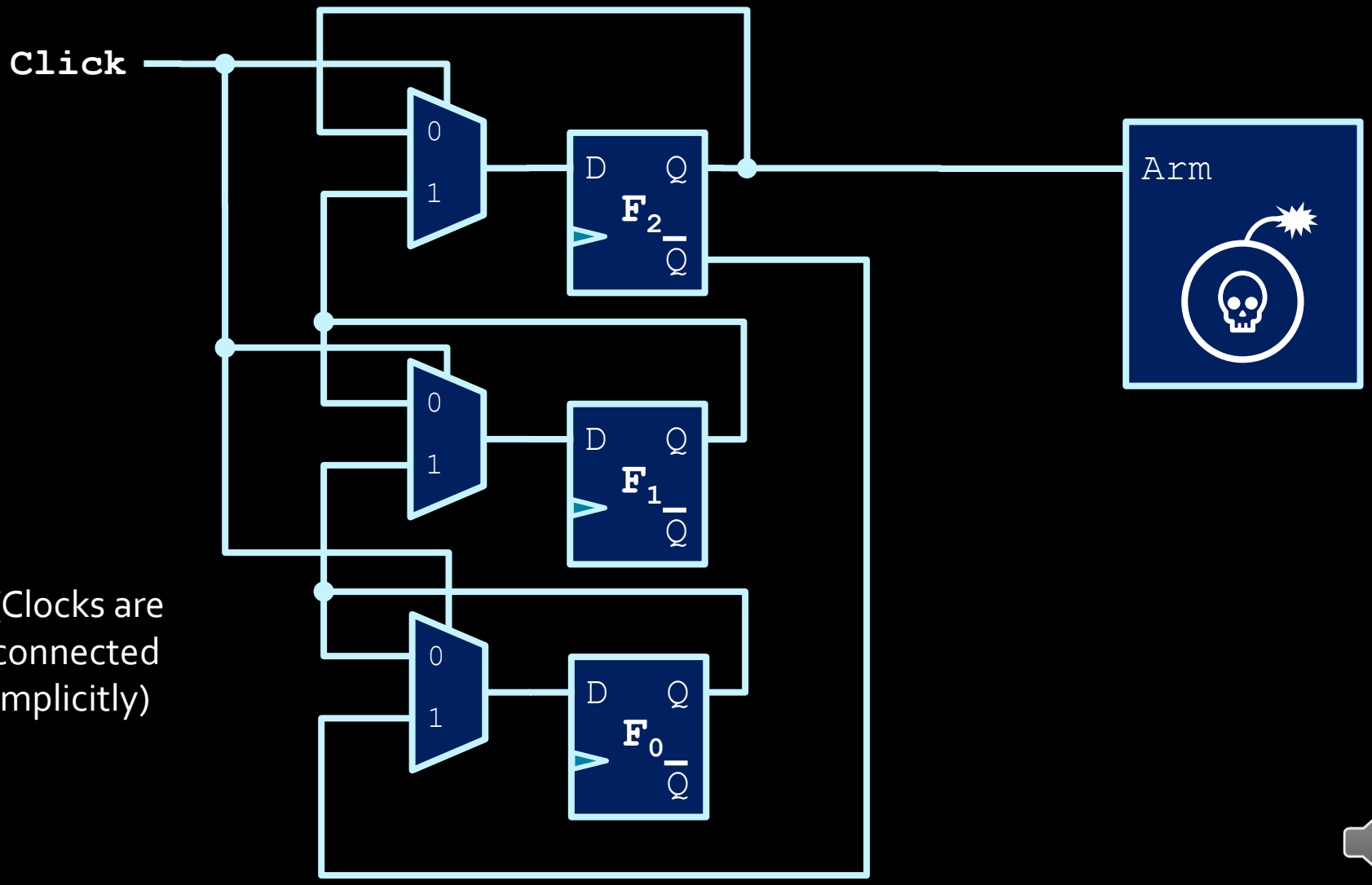
$$F_0 = F_0\overline{Click} + \overline{F_2}Click$$



Output Logic

$$Y = F_2$$

Circuit



(Clocks are connected implicitly)

We've gone quite far!

- We know how to build combinatorial and synchronous circuits.
- Starting next time, we will learn to build computers