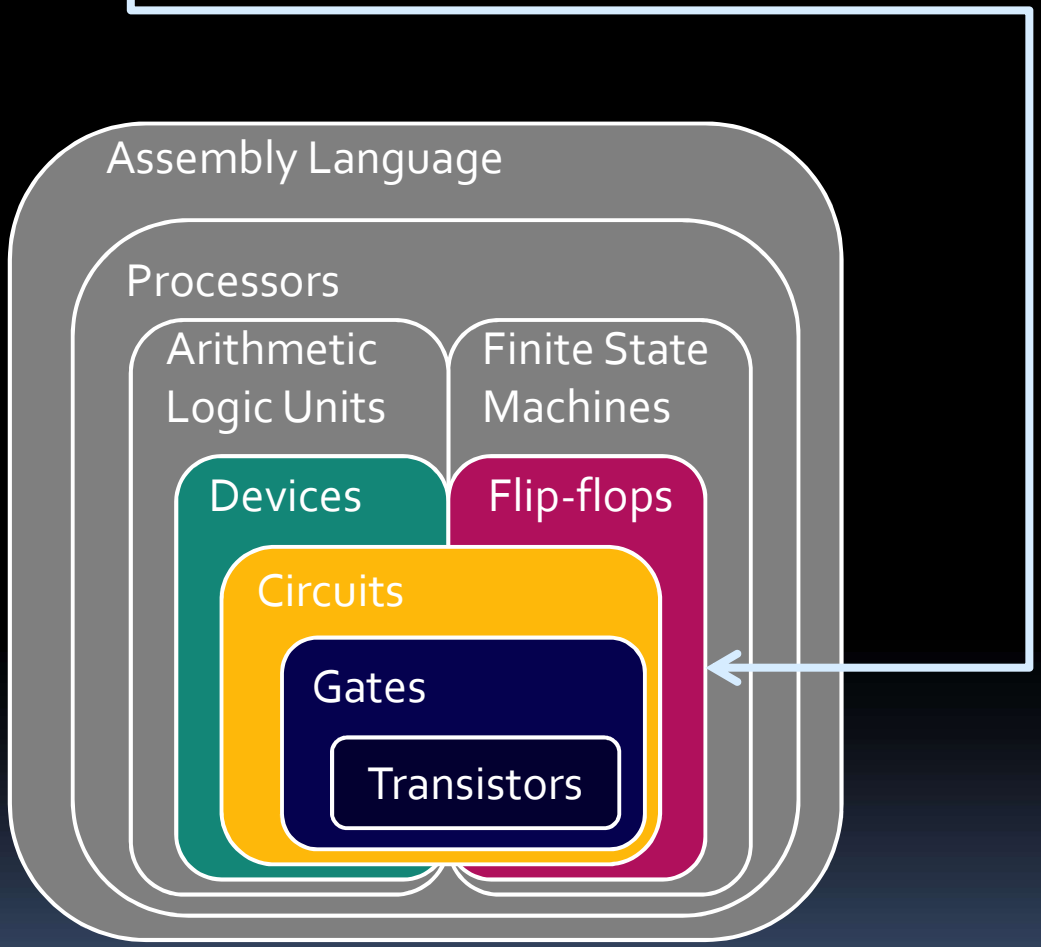
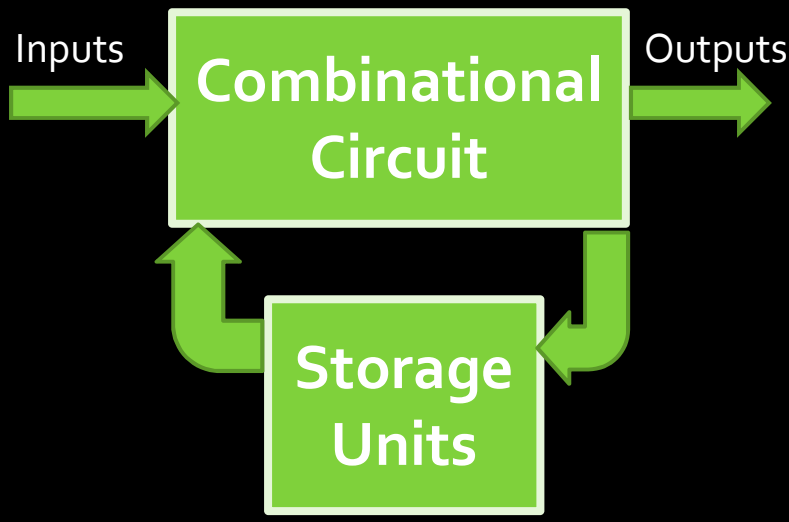
Sequential Circuit Design

We are here


Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

Circuits using flip-flops

- Now that we know about flip-flops and what they do, how do we use them in circuit design?
- What's the benefit in using flip-flops in a circuit at all?

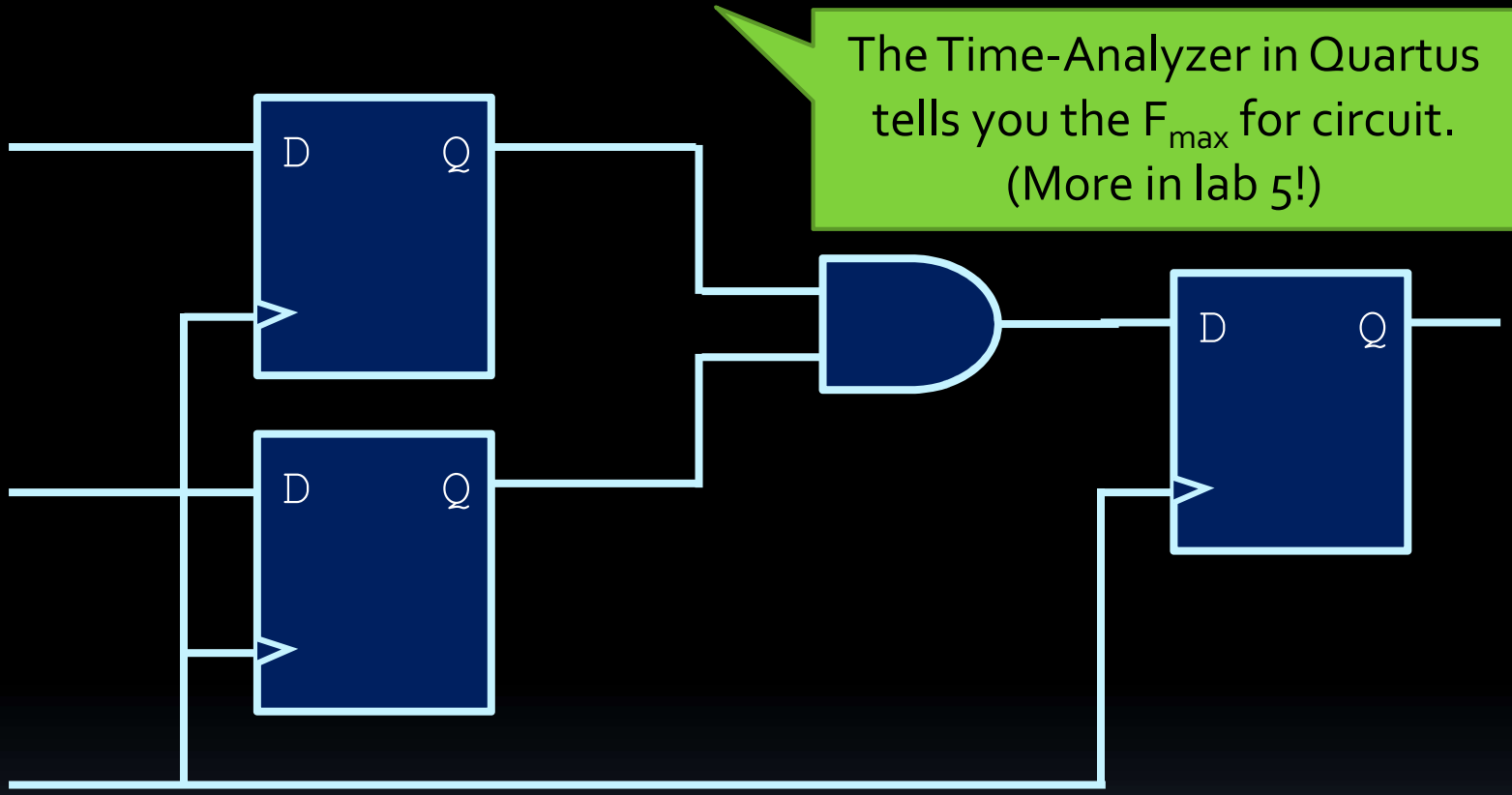


Timing Considerations

Flip-Flop Timing

- Input should NOT be changing at the same time as the active edge of the clock.
- Setup Time: Input should be stable for some time before active clock edge
- Hold Time: Input should be stable for some time immediately after the active clock edge.

Maximum clock frequency



- Time period between two active clock edges cannot be shorter than longest propagation delay between any two flip-flops + setup time of the flip-flop

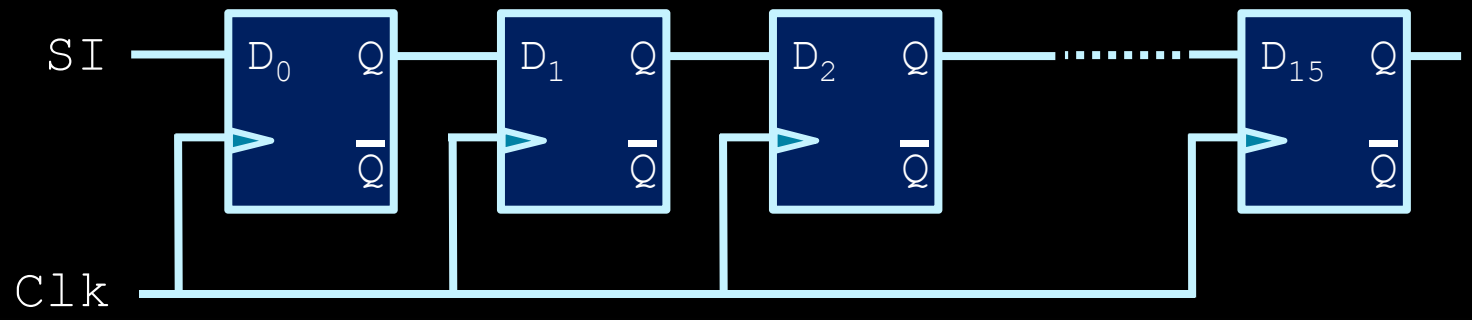
Resetting inputs

- Since flip-flops (FFs) have unknown state when they are first powered up, we need a convenient way to initialize them
- Reset signal resets the FF output to 0
  - This is unrelated to R input of SR latch
- Synchronous reset
  - The output is reset to 0 only on the active edge of the clock
- Asynchronous reset
  - The output is reset to 0 immediately (as soon as the asynchronous reset signal becomes active), independent of the clock signal

Example #1: Registers

Shift registers

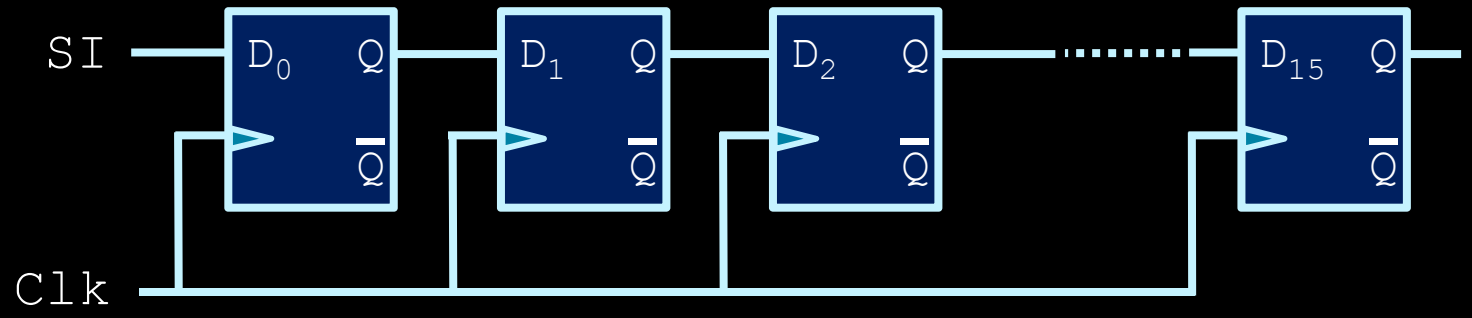
- A series of D flip-flops can store a multi-bit value (such as a 16-bit integer, for example).



- Data can be shifted into this register one bit at a time, over 16 clock cycles.
  - Known as a shift register.

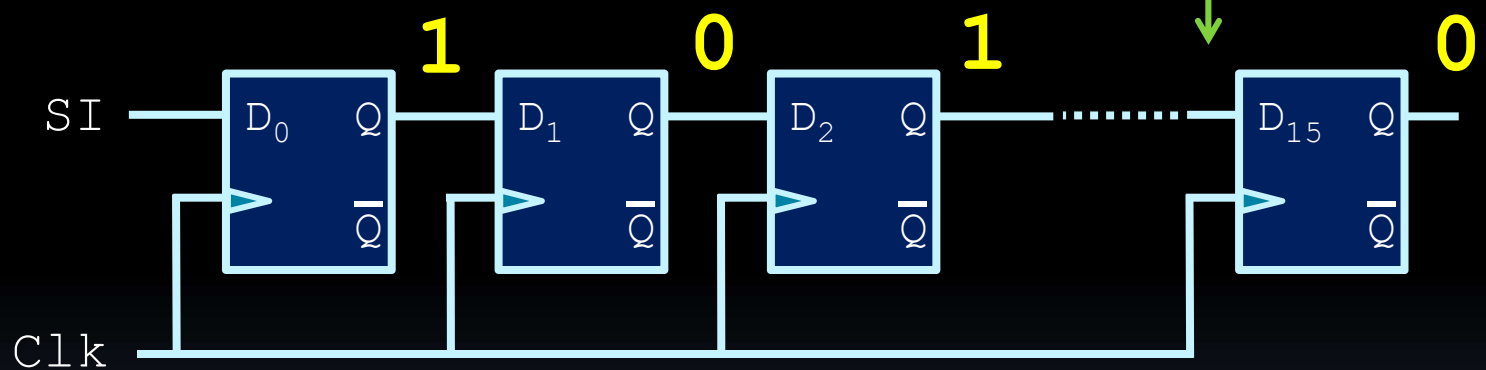
Shift registers

- Illustration: shifting in 0101010101010101



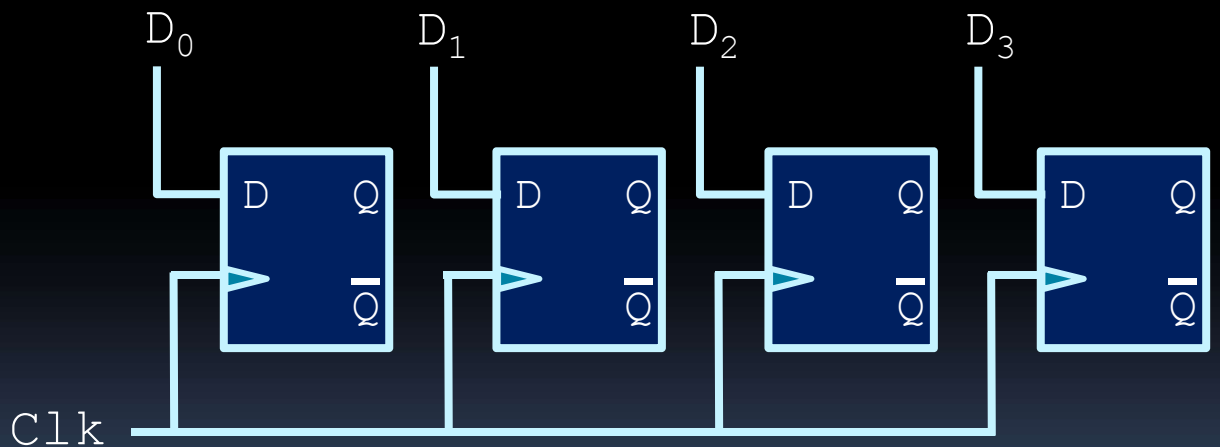
Shift registers

- Illustration: shifting in 0101010101010101
  - After 16 clock cycles….



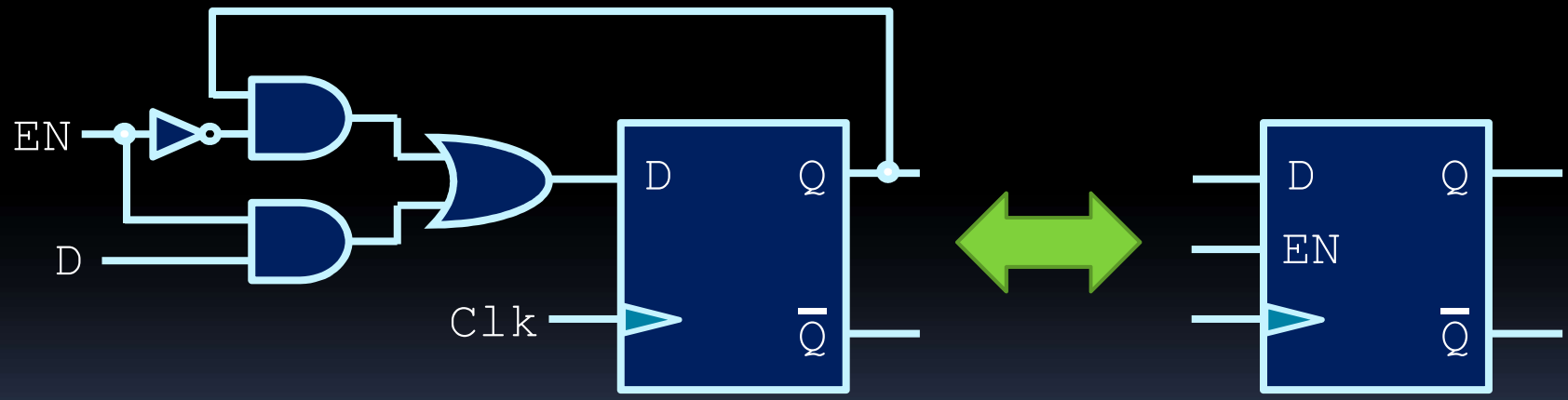
Load registers

- One can also load a register's values all at once, by feeding signals into each flip-flop:
  - In this example: a 4-bit load register.

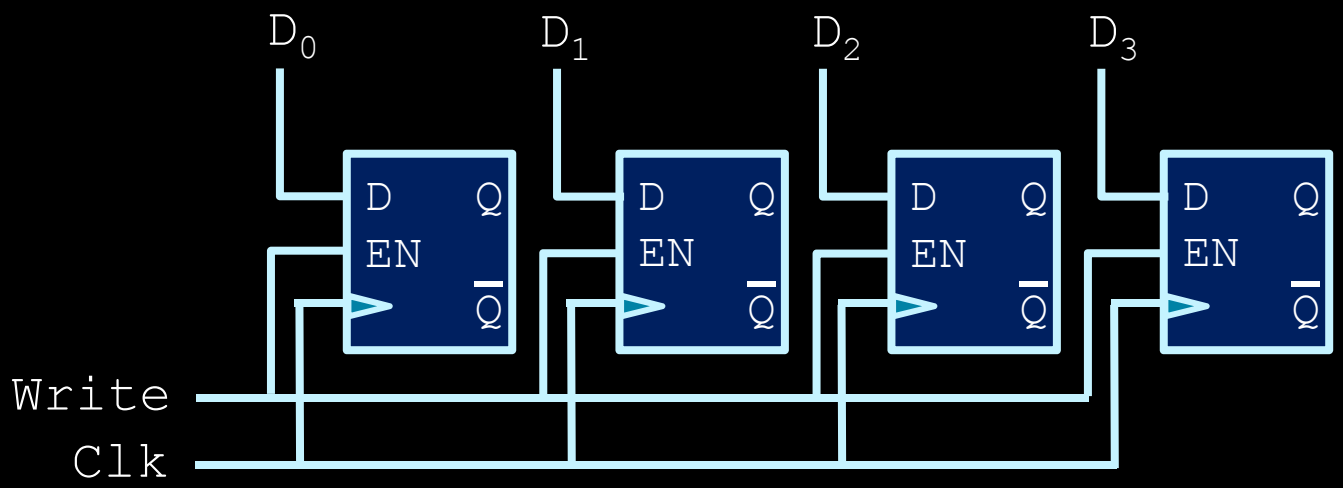


Load registers

- To control when this register is allowed to load its values, we introduce the D flip-flop with enable:



Load registers

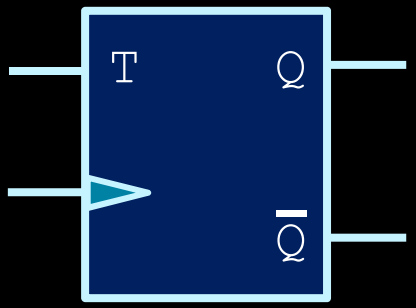


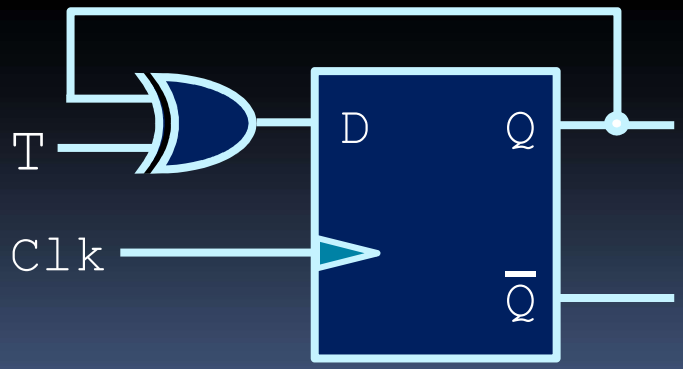
- Implementing the register with these special D flip-flops will now maintain values in the register until overwritten by setting EN high.

Example #2: Counters

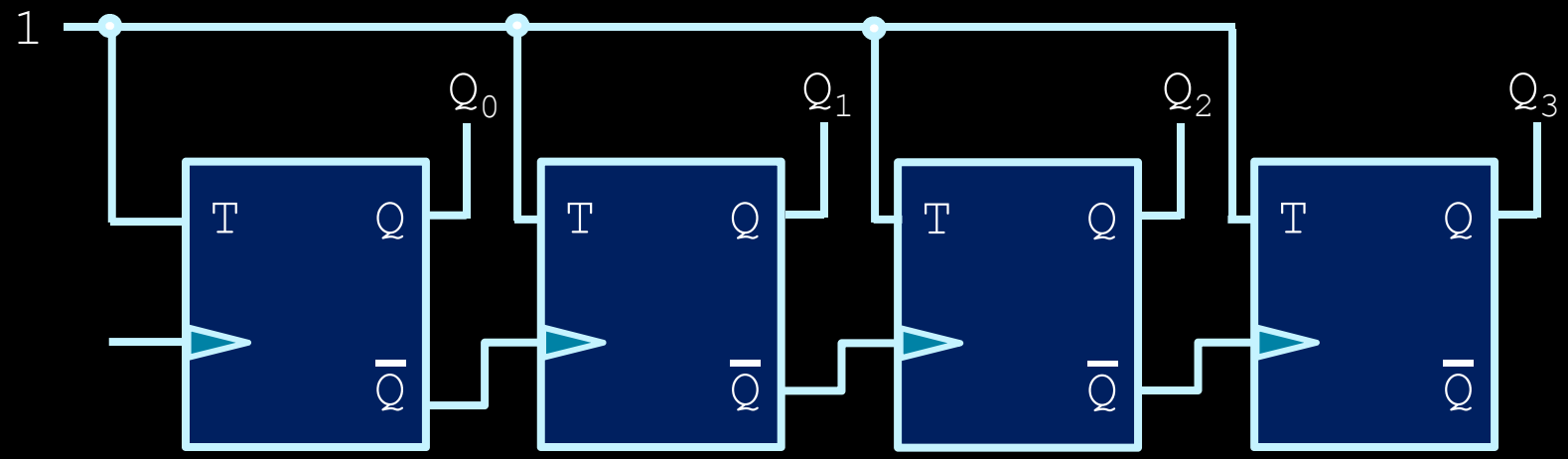
Counters

- Consider the T flip-flop:
  - Output is inverted when input T is high.
- What happens when a series of T flip-flips are connected together in sequence?
- More interesting:
  - Connect the *output* of one flip-flop to the *clock* input of the next!



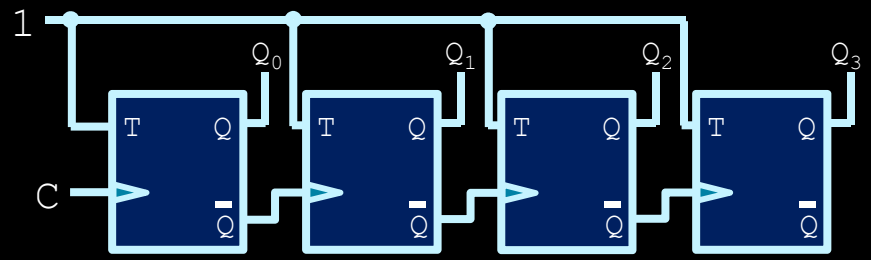


Counters

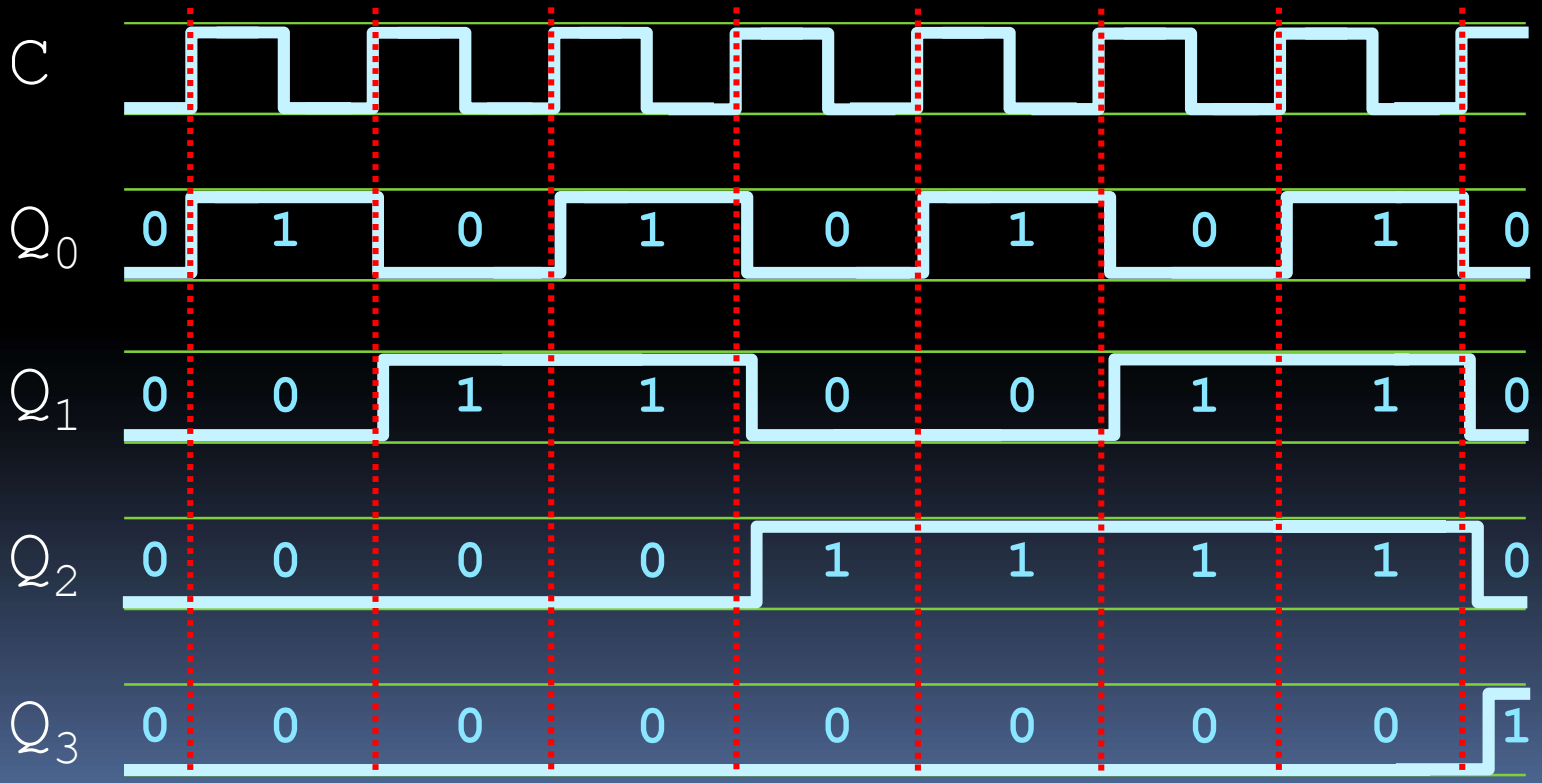


- This is a 4-bit ripple counter, which is an example of an asynchronous circuit.
  - Timing isn't quite synchronized with the rising clock pulse.
  - Cheap to implement, but unreliable for timing.

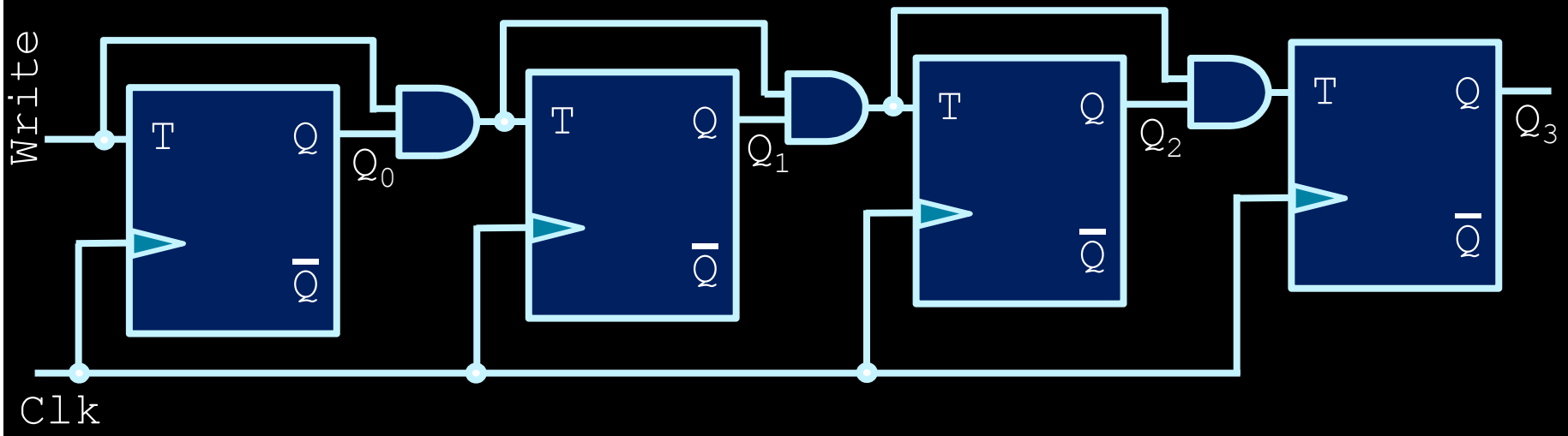
Counters



- Timing diagram
  - Note the propagation delays!

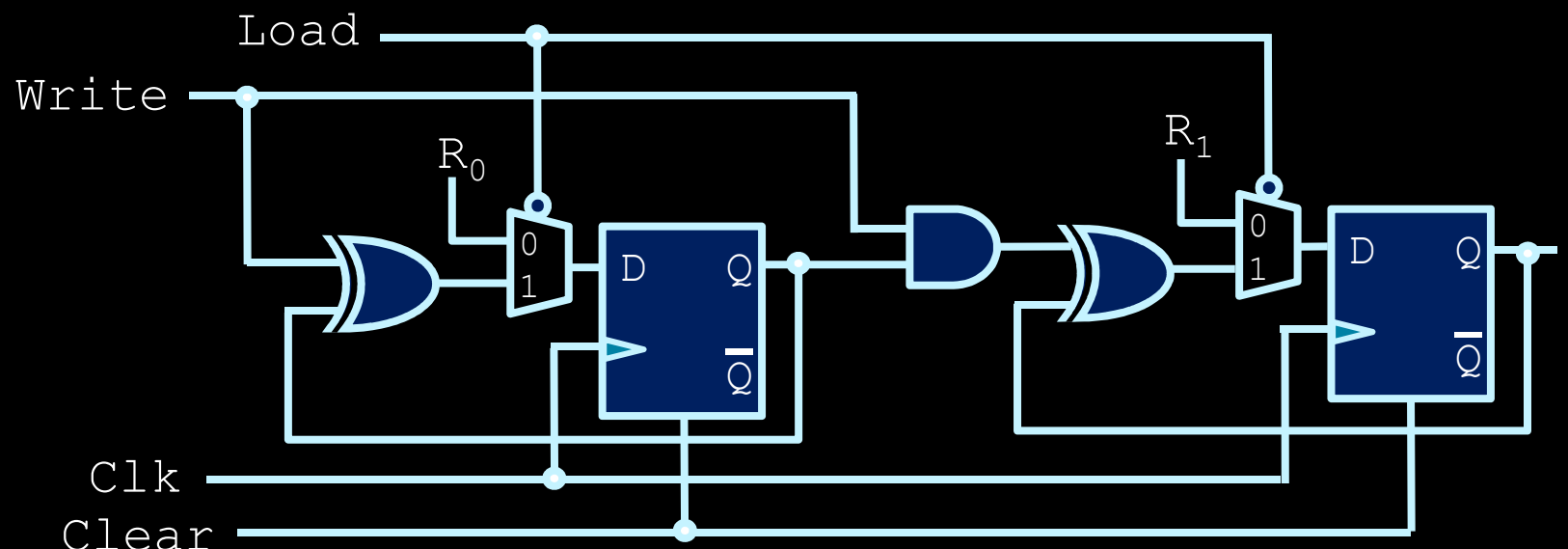


Counters



- This is a synchronous counter, with a slight delay.
  - Could be synchronized even more by having each AND gate combine outputs of all previous flip-flops.

Example #3: Counters

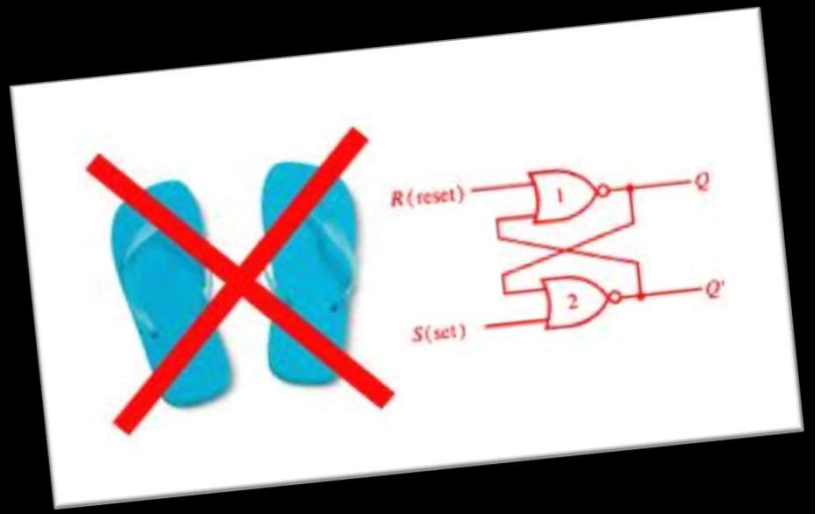


- Counters are often implemented with a parallel load and clear inputs.
  - Loading a counter value is used for countdowns.

State Machines

Designing with flip-flops

- Counters and registers are examples of how flip-flops can implement useful circuits that store values.
  - How do you design these circuits?
  - What would you design with these circuits?

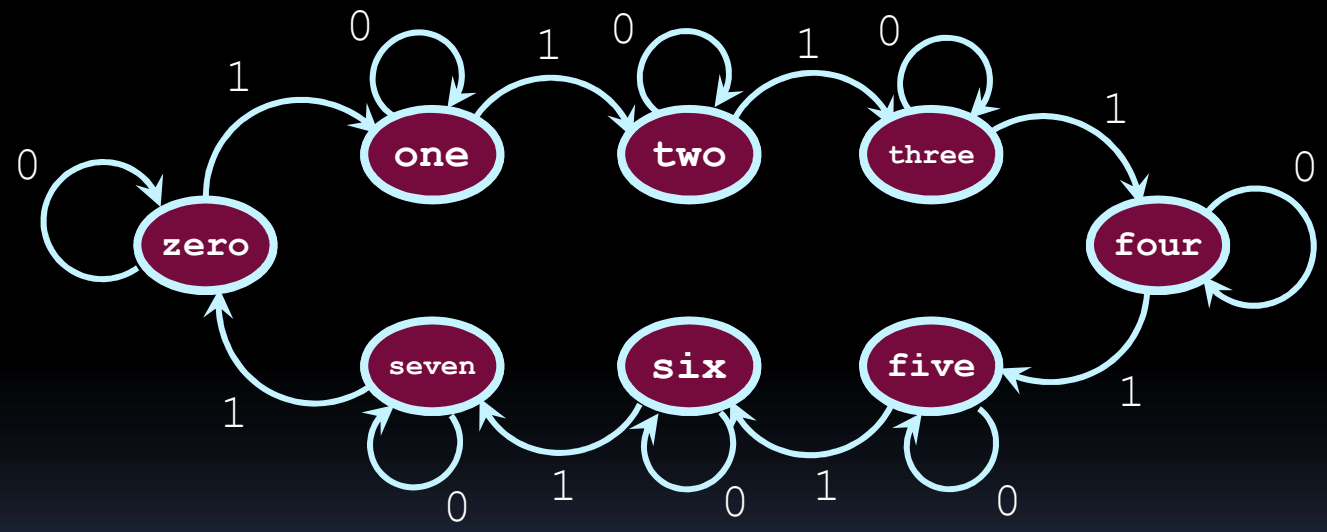


Designing with flip-flops

- Sequential circuits are the basis for memory, instruction processing, and any other operation that requires the circuit to remember past data values.
- These past data values are also called the states of the circuit.
- Sequential circuits use combinational logic to determine what the next state of the system should be, based on the past state and the current input values.

State example: Counters

- With counters, each state is the current number that is stored in the counter.



- On each clock tick, the circuit transitions from one state to the next, based on the inputs.

State Tables

- State tables help to illustrate how the states of the circuit change with various input values.
  - Transitions are understood to take place on the clock ticks.

State	Write	State
zero	0	zero
zero	1	one
one	0	one
one	1	two
two	0	two
two	1	three
three	0	three
three	1	four
four	0	four
four	1	five
five	0	five
five	1	six
six	0	six
six	1	seven
seven	0	seven
seven	1	zero

State Tables

- Same table as on the previous slide, but with the actual flip-flop values instead of state labels.
  - Note: Flip-flop values are both inputs and outputs of the circuit here.

F_1	F_2	F_3	Write	F_1	F_2	F_3
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	0
0	1	0	0	0	1	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	0	0	0

and this brings us to…

Finite State Machines

As seen in other courses…

- You may have seen finite state machines before, but in a different context.
  - Used mainly to describe the grammars of a language, or to model sequence data.
- In CSC258, finite state machines are models for an actual circuit design.
  - The states represent internal states of the circuit, which are stored in the flip-flop values.

Finite State Machines (FSMs)

- From theory courses…
  - A Finite State Machine is an abstract model that captures the operation of a sequential circuit.
- A FSM is defined (in general) as:
  - A finite set of states,
  - A finite set of transitions between states, triggered by inputs to the state machine,
  - Output values that are associated with each state or each transition (depending on the machine),
  - Start and end states for the state machine.

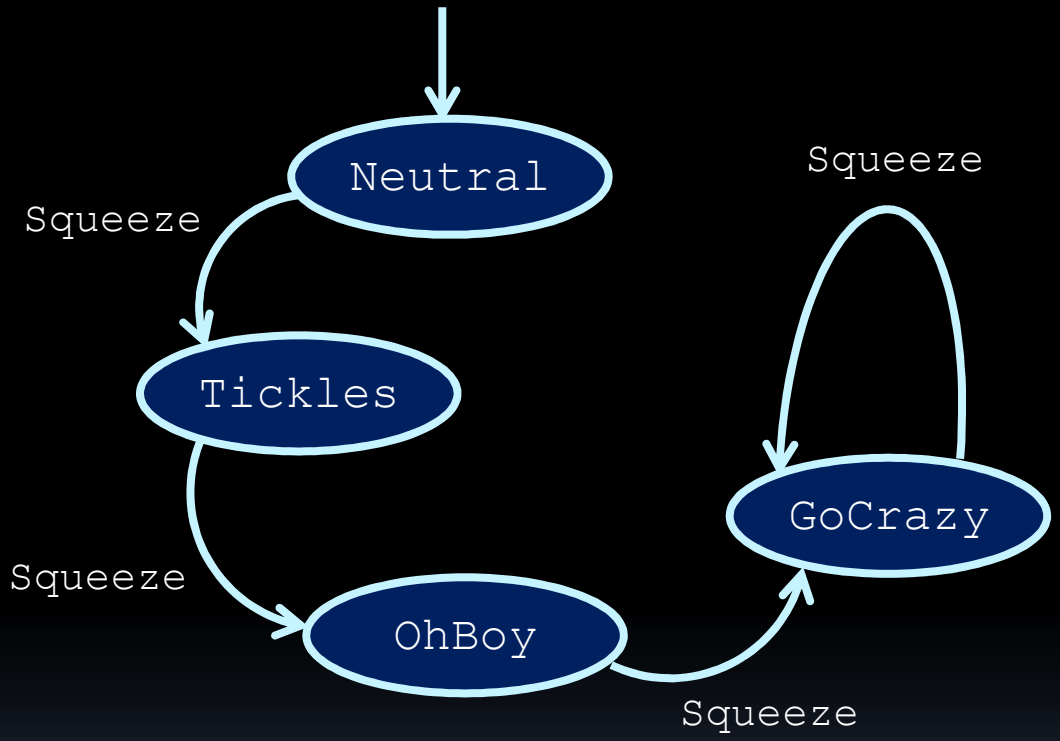
Example #1: Tickle Me Elmo

- Remember how the Tickle Me Elmo works!

Example #1: Tickle Me Elmo

- Toy reacts differently each time it is squeezed:
  - First squeeze → *"Ha ha ha…that tickles."*
  - Second squeeze → *"Ha ha ha…oh boy."*
  - Third squeeze → *"HA HA HA HA…HA HA HA HA…etc"*
- Questions to ask:
  - What are the inputs?
  - What are the states of this machine?
  - How do you change from one state to the next?

Example #1: Tickle Me Elmo



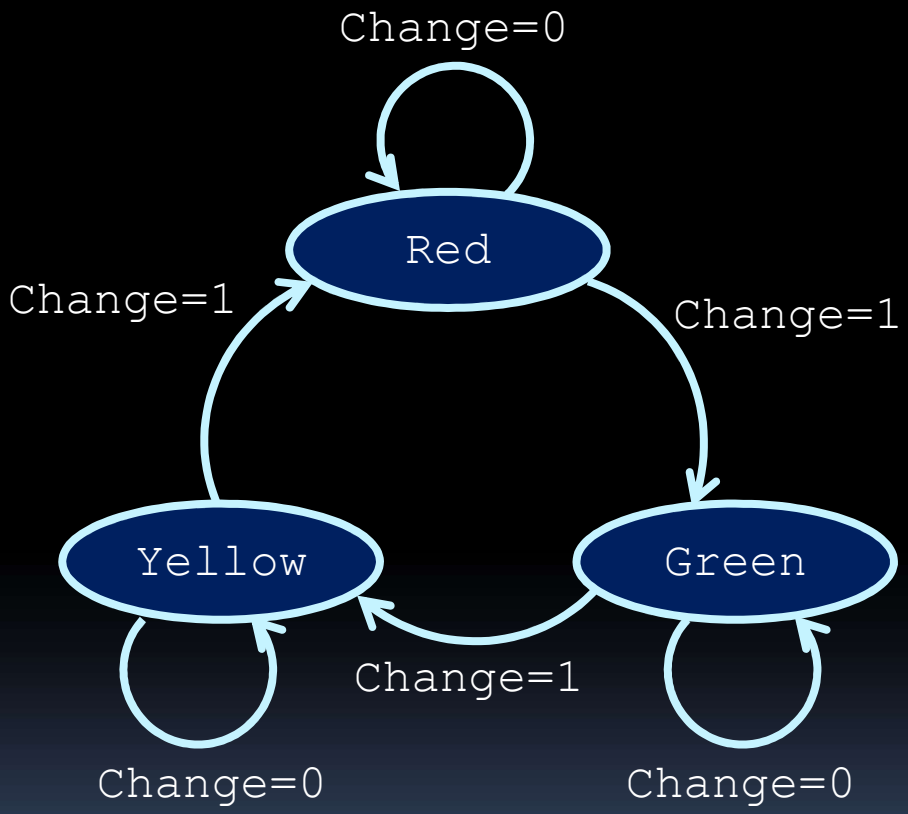
More elaborate FSMs

- Usually our FSM has more than one input, and will trigger a transition based on certain input values but not others.
- Also might have input values that don't cause a transition, but keep the circuit in the same state (transitioning to itself).

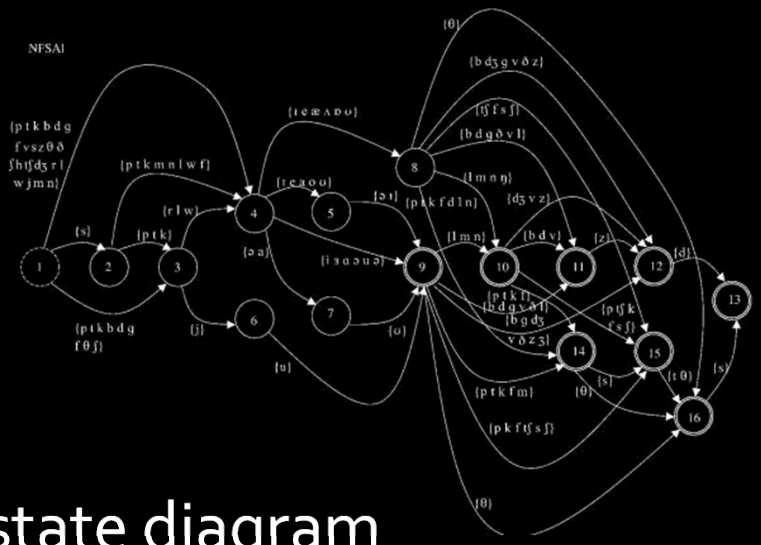
Example #2: Alarm Clock

- Internal state description:
  - Starts in neutral state, until timer signal goes off.
    - Clock moves to alarm state.
  - Alarm state continues until:
    - snooze button is pushed (move to snooze state)
    - alarm is turned off (move to neutral state)
    - timer goes off again (move to neutral state)
  - In snooze state, clock returns to alarm state when the timer signal goes off again.

Example #3: Traffic Light



FSM design



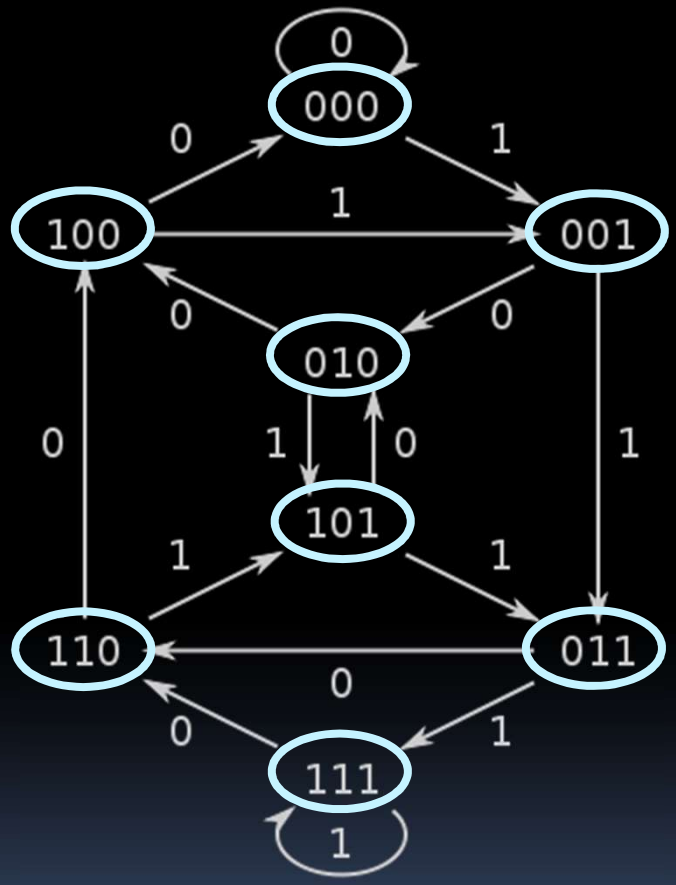
- Design steps for FSM:
  1. Draw state diagram
  2. Derive state table from state diagram
  3. Assign flip-flop configuration to each state
     - Number of flip-flops needed is: $\lceil \log_2(\#\text{ of states}) \rceil$
  4. Redraw state table with flip-flop values
  5. Derive combinational circuit for output and for each flip-flop input.

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
  - Understood to mean that the input has been high for three rising clock edges.
  - Assumes a single input `X` and a single output `Z`.

Step 1: State diagram

- In this case, the states are labeled with the input values that have been seen up to now.
- Transitions between states are 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.

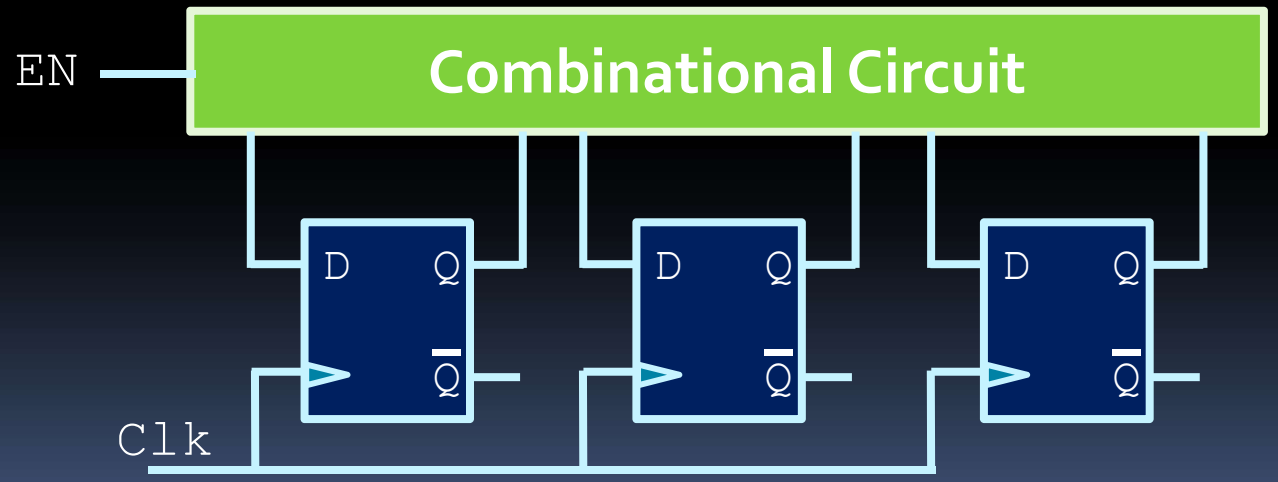
Previous State	EN	Next State
000	0	000
000	1	001
001	0	010
001	1	011
010	0	100
010	1	101
011	0	110
011	1	111
100	0	000
100	1	001
101	0	010
101	1	011
110	0	100
110	1	101
111	0	110
111	1	111

Step 3: Assign flip-flops

- The flip-flops are the circuit units that are responsible for actually storing states.
- When deciding how many states are needed, remember that a single flip-flop can store two values (`0` and `1`), and thus 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

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.
- It may be an easy mapping, but is it a good one?
  - Not really, but we'll get to why later.

Prev. State	**EN**	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 5: Circuit design

- Karnaugh map for F_2:

	$\overline{F_0} \cdot \overline{EN}$	$\overline{F_0} \cdot EN$	$F_0 \cdot EN$	$F_0 \cdot \overline{EN}$
$\overline{F_2} \cdot \overline{F_1}$	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

$$F_2 = F_1$$

Step 5: Circuit design

- Karnaugh map for F_1:

	$\overline{F_0} \cdot \overline{EN}$	$\overline{F_0} \cdot EN$	$F_0 \cdot EN$	$F_0 \cdot \overline{EN}$
$\overline{F_2} \cdot \overline{F_1}$	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

$$F_1 = F_0$$

Step 5: Circuit design

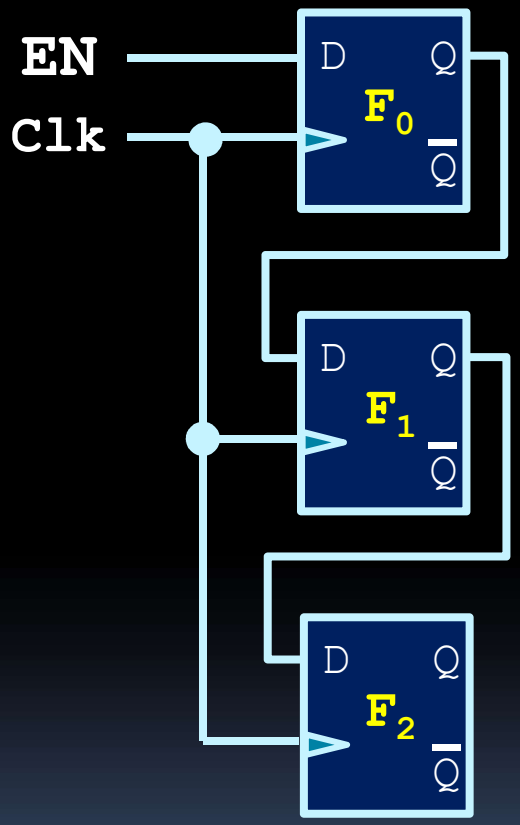
- Karnaugh map for F_0:

	$\overline{F_0} \cdot \overline{EN}$	$\overline{F_0} \cdot EN$	$F_0 \cdot EN$	$F_0 \cdot \overline{EN}$
$\overline{F_2} \cdot \overline{F_1}$	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

$$F_0 = EN$$

Step 5: Circuit design

- Resulting circuit looks like the diagram on the right.
- This will record the states and make the state transitions happen based on the input, but what about the output value Z?
  - Z should go high when EN has been high for three clock cycles in a row.



Step 5: Circuit design

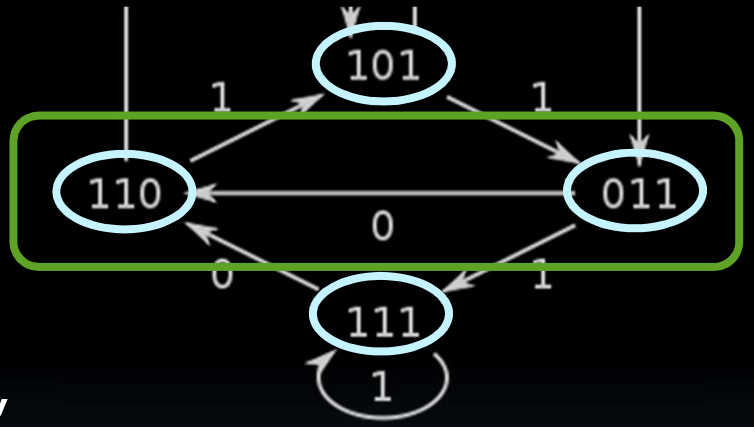
- Boolean equation for Z:

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

- Two ways to derive the circuitry needed for the output values of the state machine:
  - Moore machine:
    - The output for the FSM depends solely on the current state (based on entry actions).
  - Mealy machine:
    - The output for the FSM depends on 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.

Timing and state assignments

- When assigning states, you need to consider the issue of timing with the states.
- Example: if recognizer circuit is in state 011 and gets a 0 as an input, it moves to state 110.
  - The first and last digits change "at the same time"
  - If the first flip-flop changes first, the output would go high for an instant, which could cause unexpected behaviour.

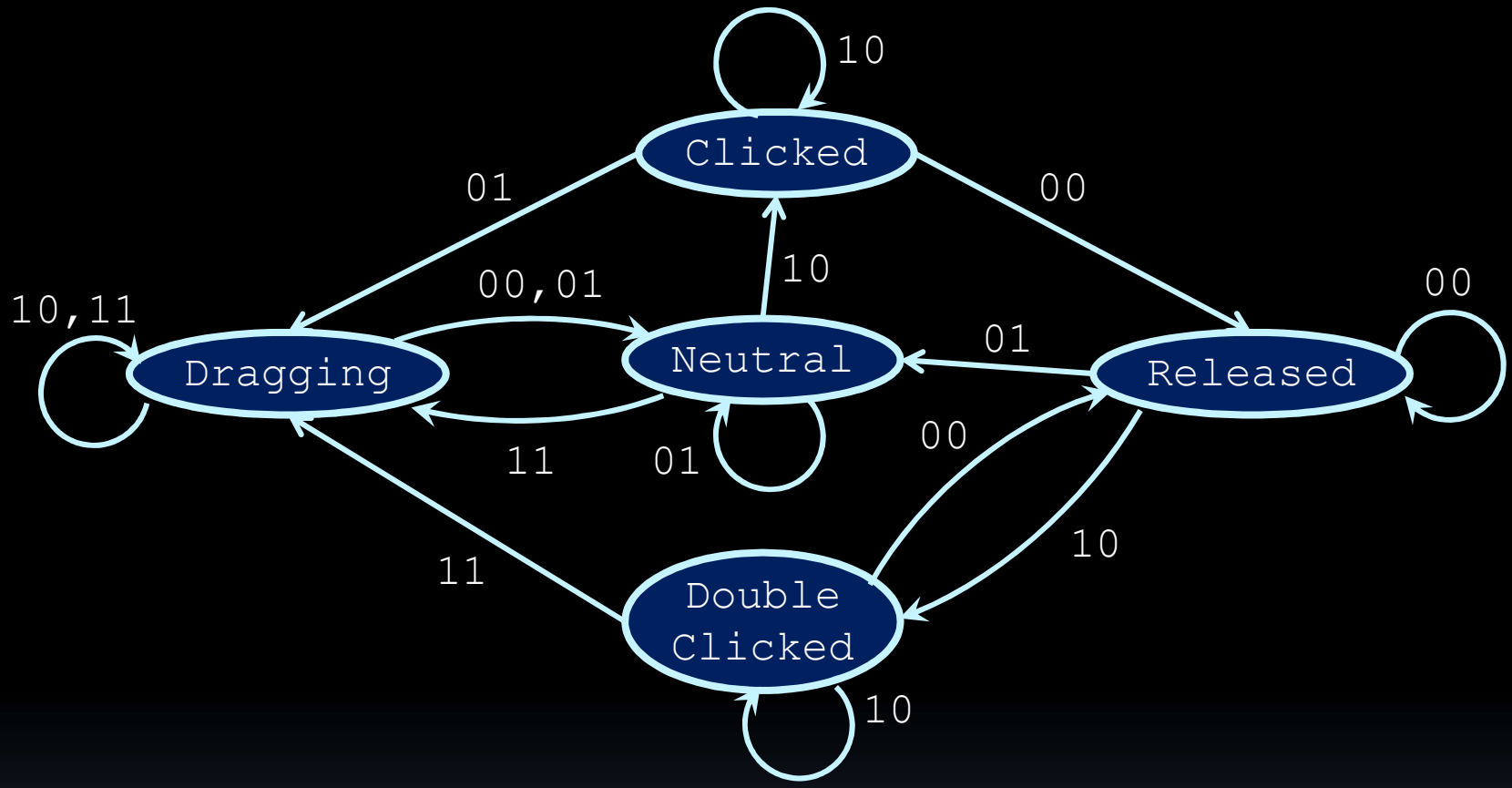


Timing and state assignments

- So how do you solve this?
- Two possible solutions:
  1. Whenever possible, make flip-flop assignments such that neighbouring states differ by at most one flip-flop value.
     - Intermediate states can be allowed if the output generated by those states is consistent with the output of the starting or destination states.
  2. If the intermediate states are unused in the state diagram, you can set the output for these states to provide the output that you need.
     - Might need to add more flip-flops to create these states.

Example #5: Mouse clicks

- Design a circuit that takes in two signals:
  - A signal P, which is high if the user is pressing the mouse button,
  - A signal M, which is high if the mouse is being moved.
- Based on the inputs, indicate whether the user is clicking, double-clicking, or dragging the mouse on the screen.



- Transitions indicate the values of P&M.
- Outputs depend on the state (Moore machine)