L9_1 Combinational-Logic-Timing

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

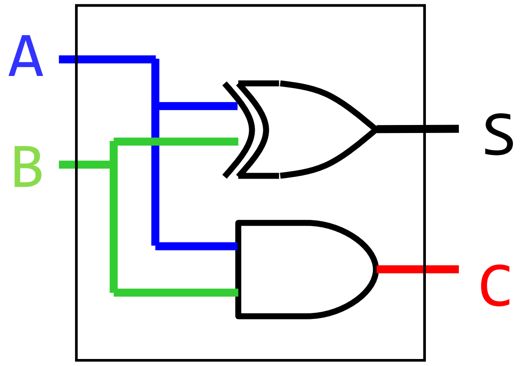
- To identify the propagation delay in combinational logic circuits.

Combinational Circuits Implement Boolean Expressions

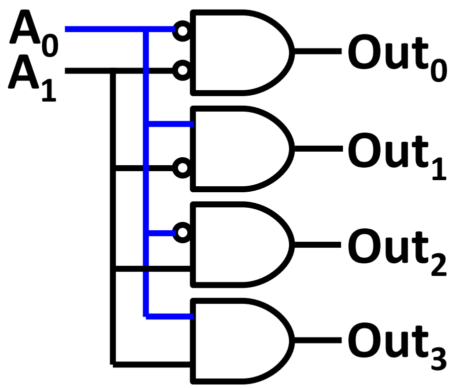
Comb. Logic Review

- Output is determined exclusively by the input
- No memory: Output is valid only as long as input is
  - Adder is the basic gate of the ALU (this lecture)
  - Decoder is the basic gate of indexing (we will use this next lecture)
  - MUX is the basic gate controlling data movement

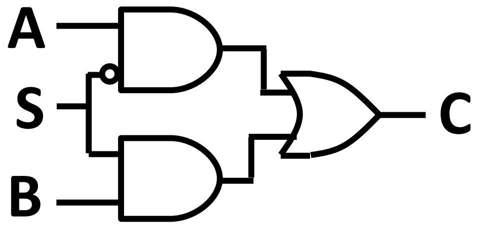
Half-Adder



Decoder



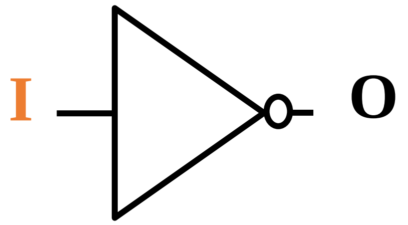
Mux

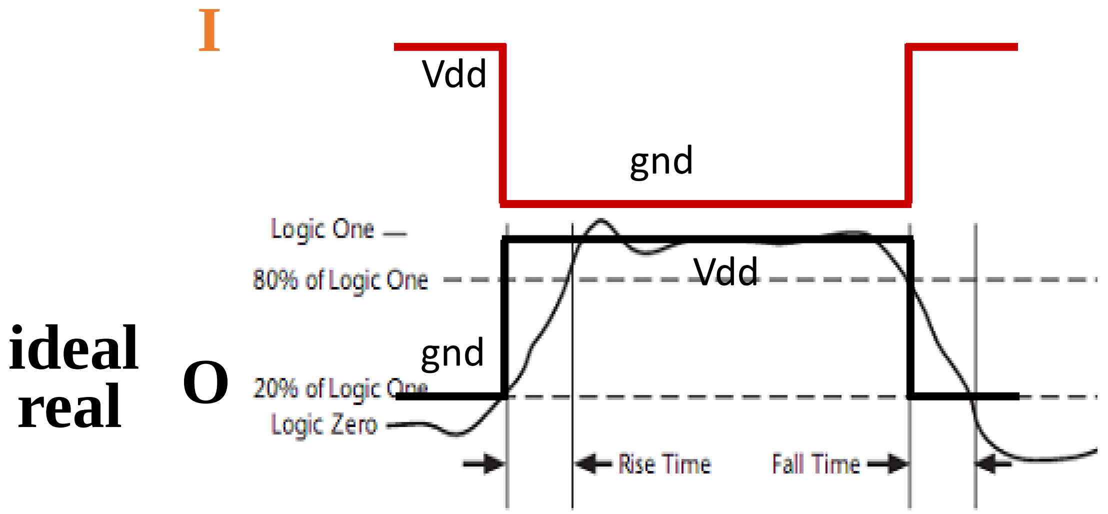


Propagation Delay in Combinational Gates

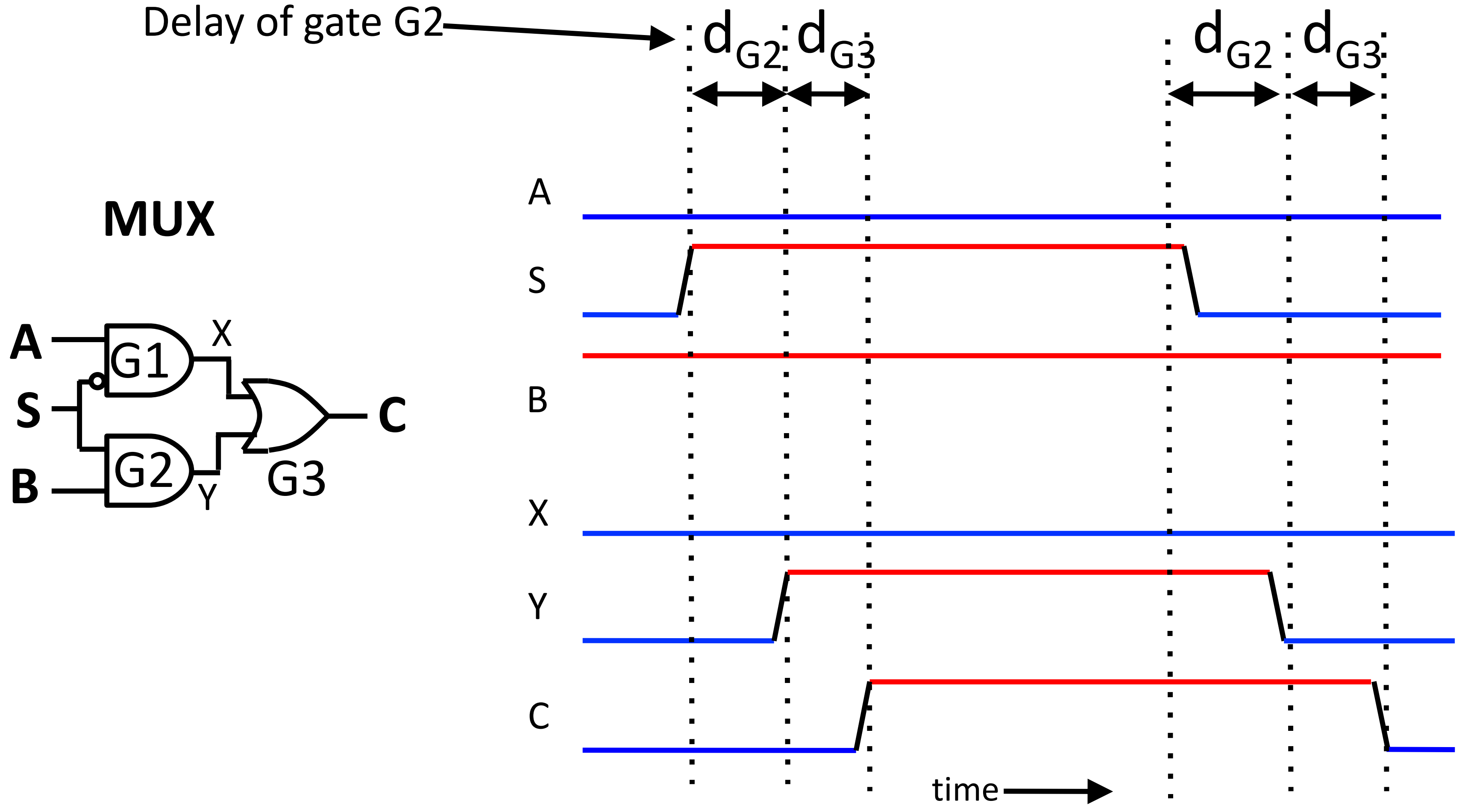
- Gate outputs do not change exactly when inputs do.
  - Transmission time over wires (~speed of light)
  - Saturation time to make transistor gate switch

Every combinatorial circuit has a propagation delay (time between input and output stabilization)





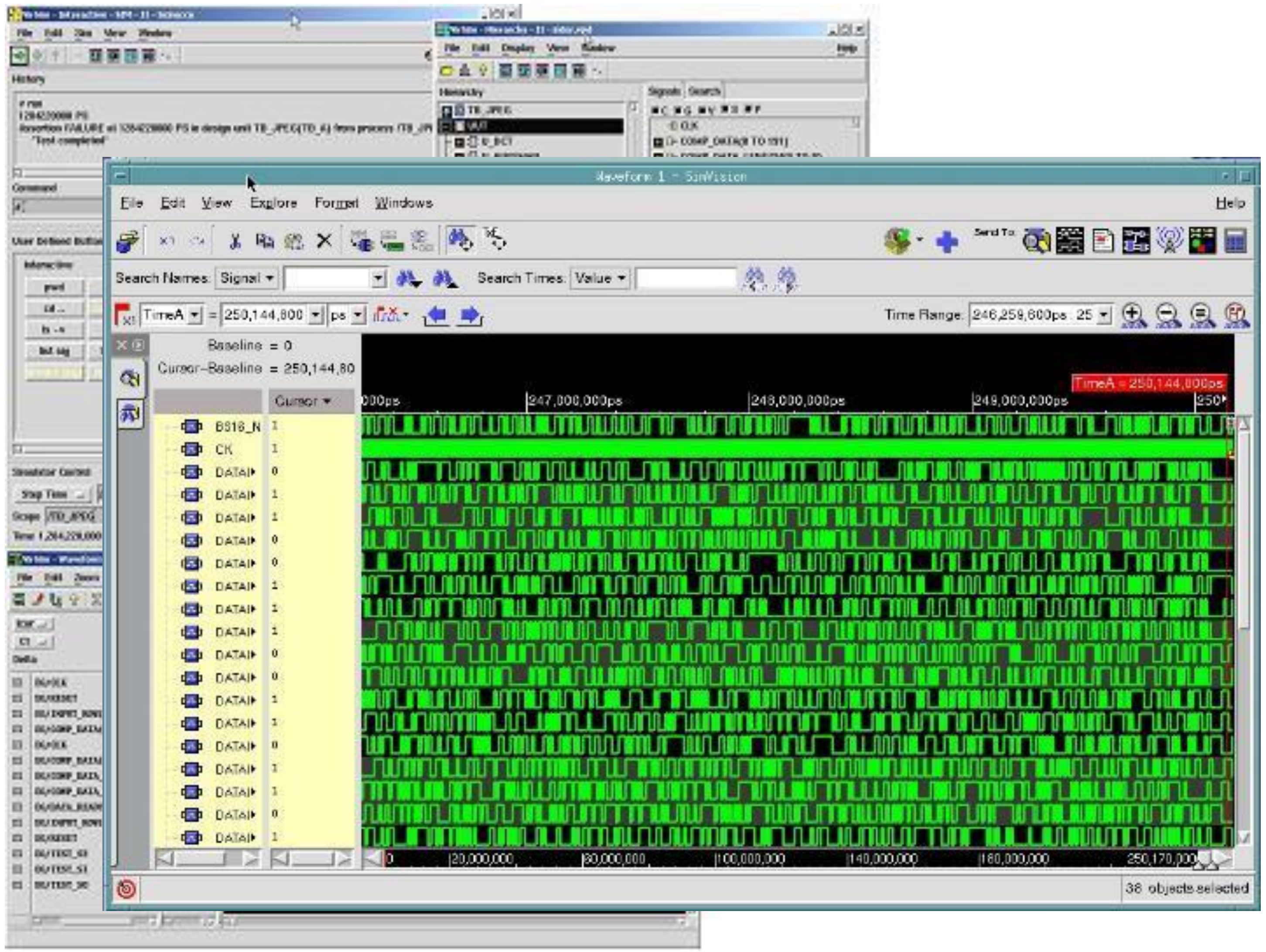
Timing in Combinational Circuits



What is the input/output delay (or simply, delay) of the MUX?

Waveform viewers are part of designers' daily life

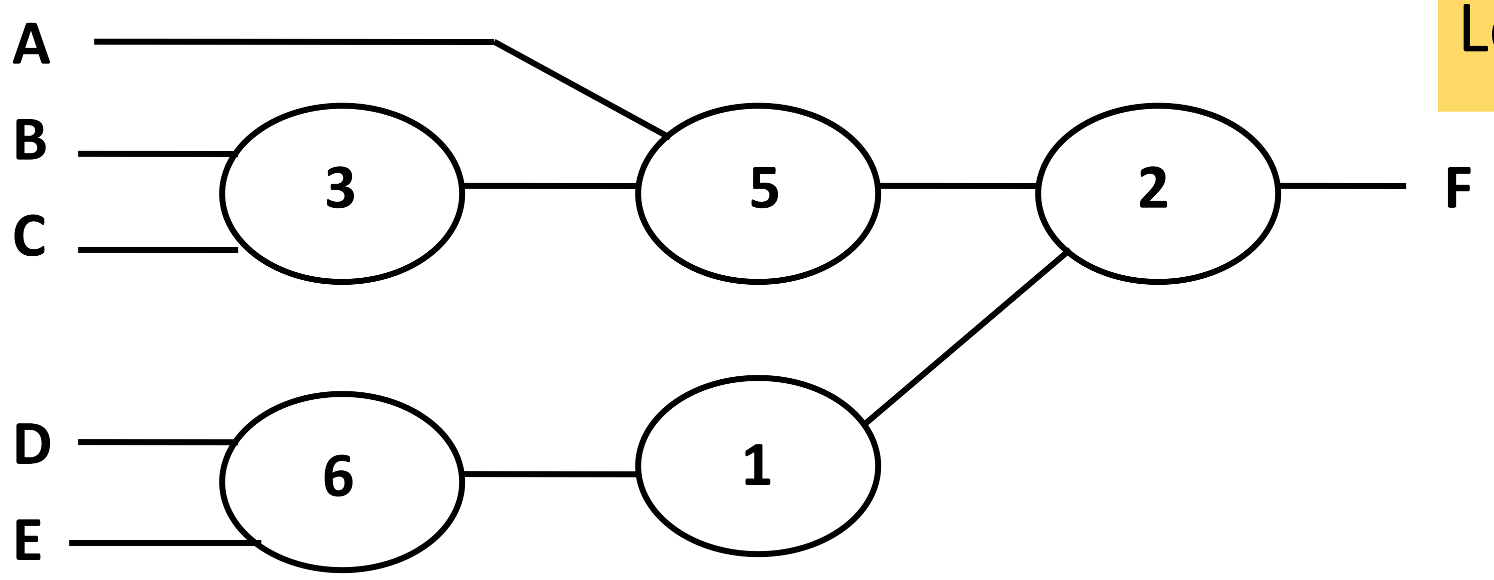
Comb. Logic



What is the delay of this Circuit?

Each oval represents one gate (the type does not matter)
# = delay of each gate

Longest path: 3 + 5 + 2 = 10

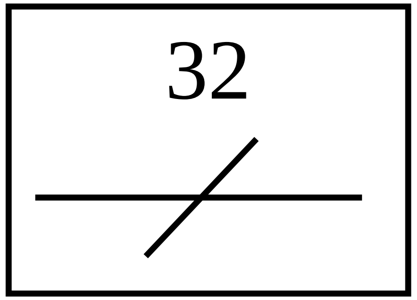


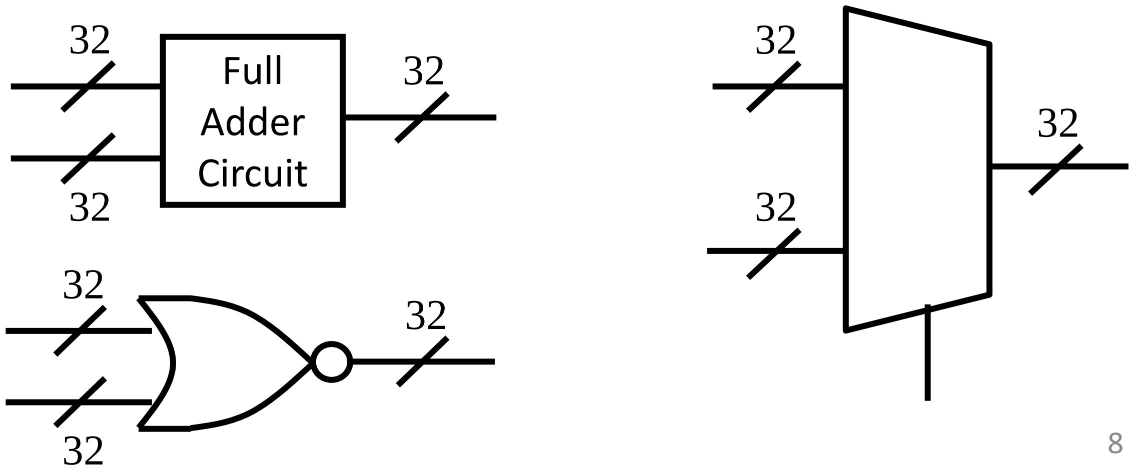
Example: Building a Circuit

Problem: Build an ALU (Arithmetic Logic Unit) for LC-2K

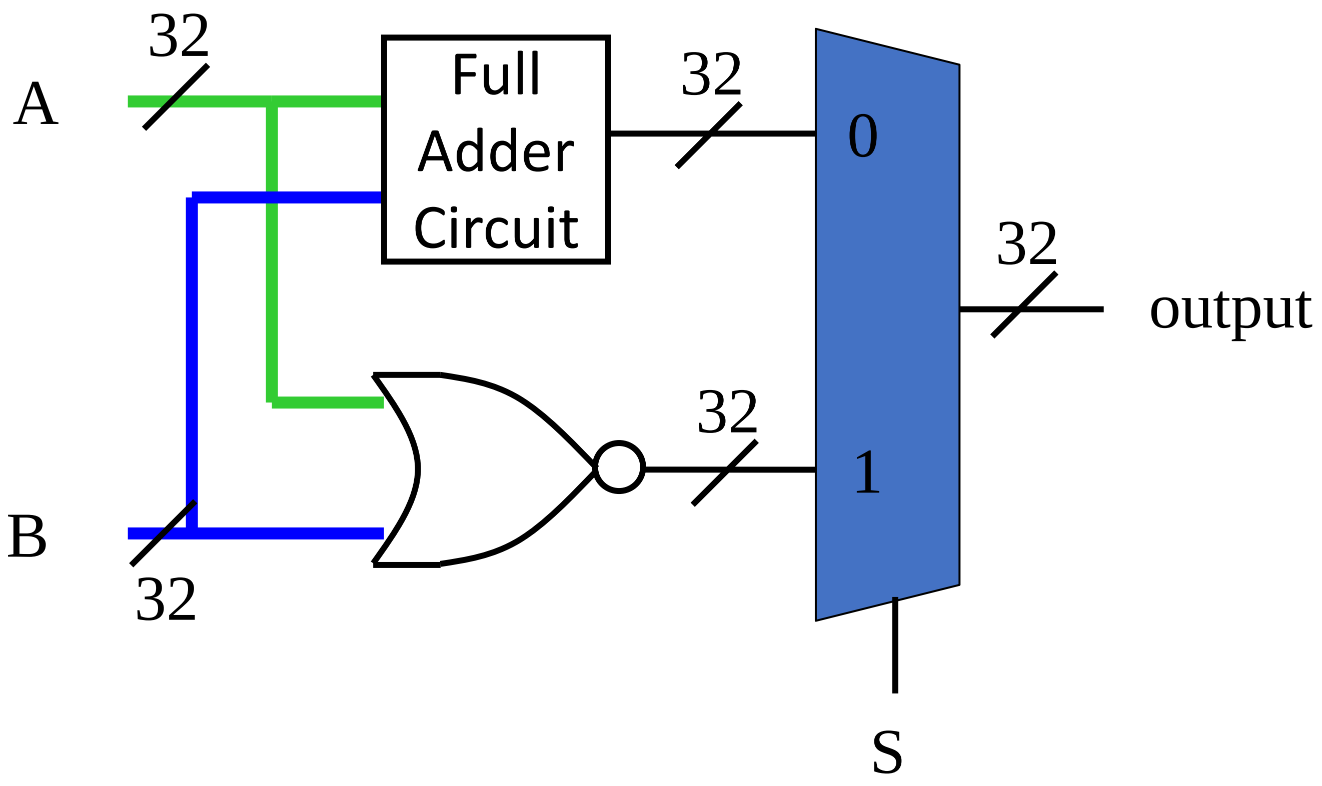
- Use some of the blocks we have learned about so far to build a circuit
  - Using - full adder, NOR, mux
  - Input A, 32 bits
  - Input B, 32 bits
  - Input S, 1 bit
  - Output, 32 bits
  - When S is low, the output is A+B, when S is high, the output is NOR(a,b)

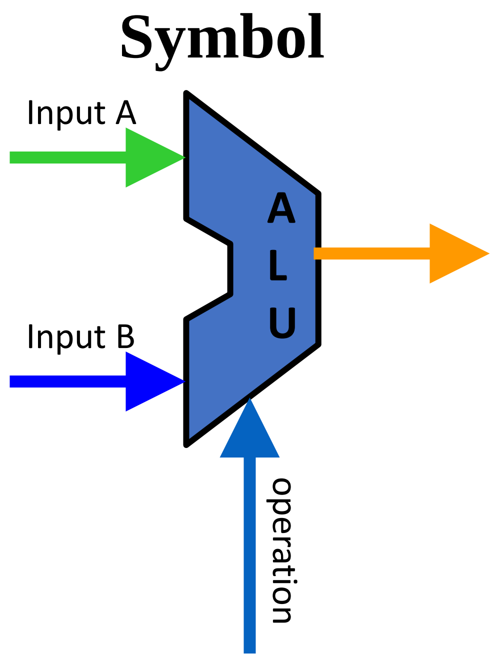
32 wires





LC-2K ALU (Arithmetic Logic Unit)





Logistics

- There are 3 videos for lecture 9
  - L9_1 – Combinational-Logic-Timing
  - L9_2 – Memory_Latches-Clocks
  - L9_3 – Finite-State-Machines
- There are two worksheet for lecture 9
  1. Circuit design – combinational logic – you are ready for this now
  2. Circuit design – sequential logic

L9_2 Memory_Latches-Clocks

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

- To identify and understand the operation of simple devices to retain memory in circuits.
- To understand the inclusion of timing with the clock circuit

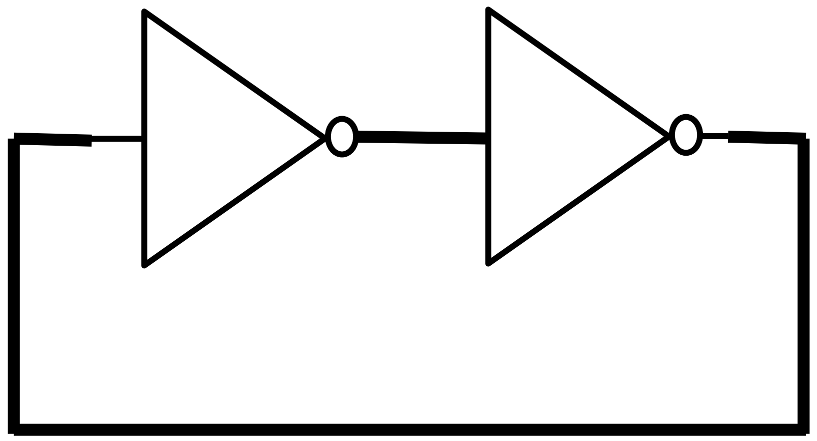
Sequential logic:
giving memory to circuits

Sequential Logic

What is sequential logic?

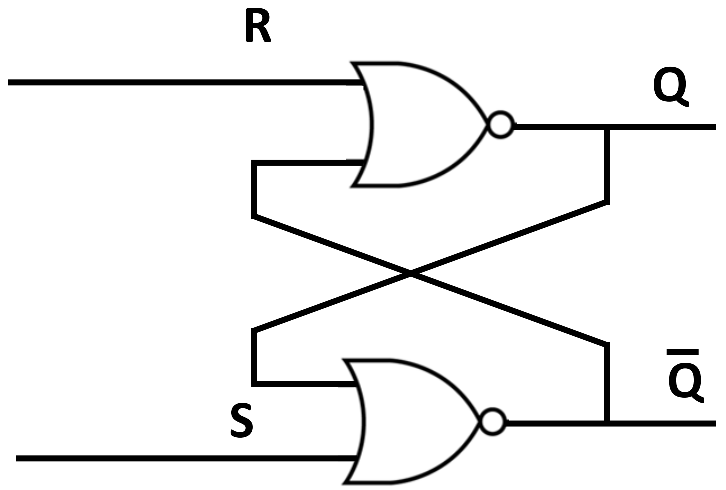
- So far, we've covered combinational
  - Output is determined from input
  - But computers don't work that way -- they have **state**
- Examples of state
  - Registers
  - Memory
- Sequential logic's output depends not only on the current input, but also on its current state
- This lecture will show you how to build sequential logic from gates
  - The key is feedback

Using Feedback to "Remember"



This remembers its initial value!
Very basic memory
What's wrong with this, though?

Your First Memory: S-R Latch



- Output Q and $\overline{Q}$ should have memory, i.e., retain their value for *some input changes*
- Output Q and $\overline{Q}$ should always have opposite values

"high" is:
- logical 1
- 1 state
- "set"
- high voltage

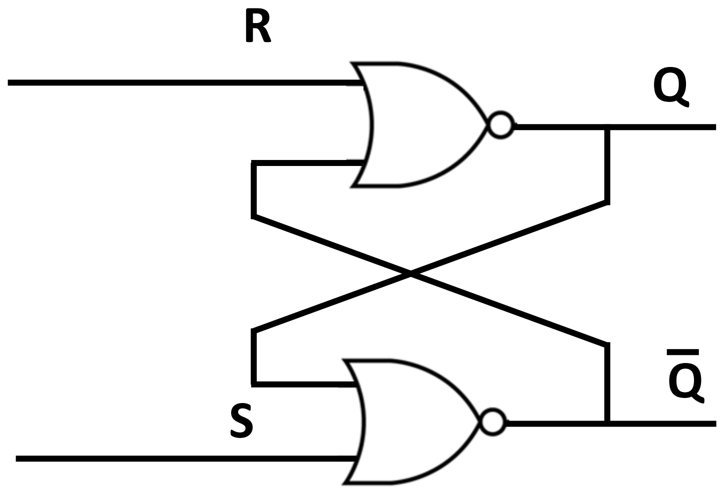
"low" is:
- logical 0
- 0 state
- "Unset"
- low voltage

$\overline{Q}$ is not Q

Sequential Logic

Your First Memory: S-R Latch

Problem: Create a truth table for this circuit



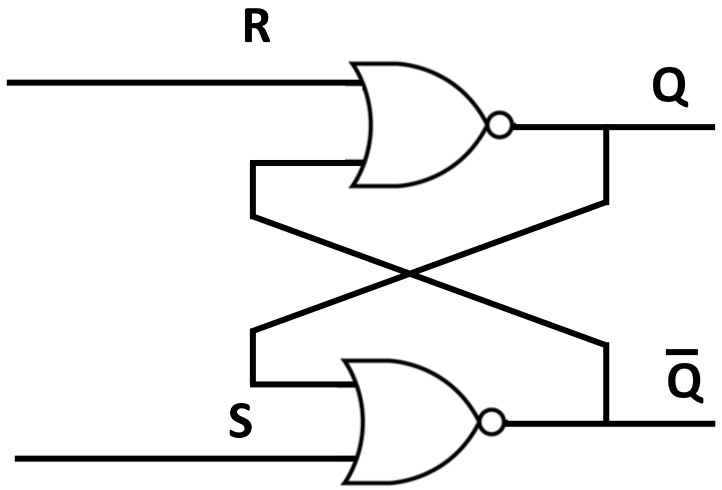
S	R	Q	$\overline{Q}$

- Output Q and $\overline{Q}$ should have memory, i.e., retain their value for *some input changes*
- Output Q and $\overline{Q}$ should always have opposite values

Your First Memory: S-R Latch

Sequential Logic

Problem: Create a truth table for this circuit



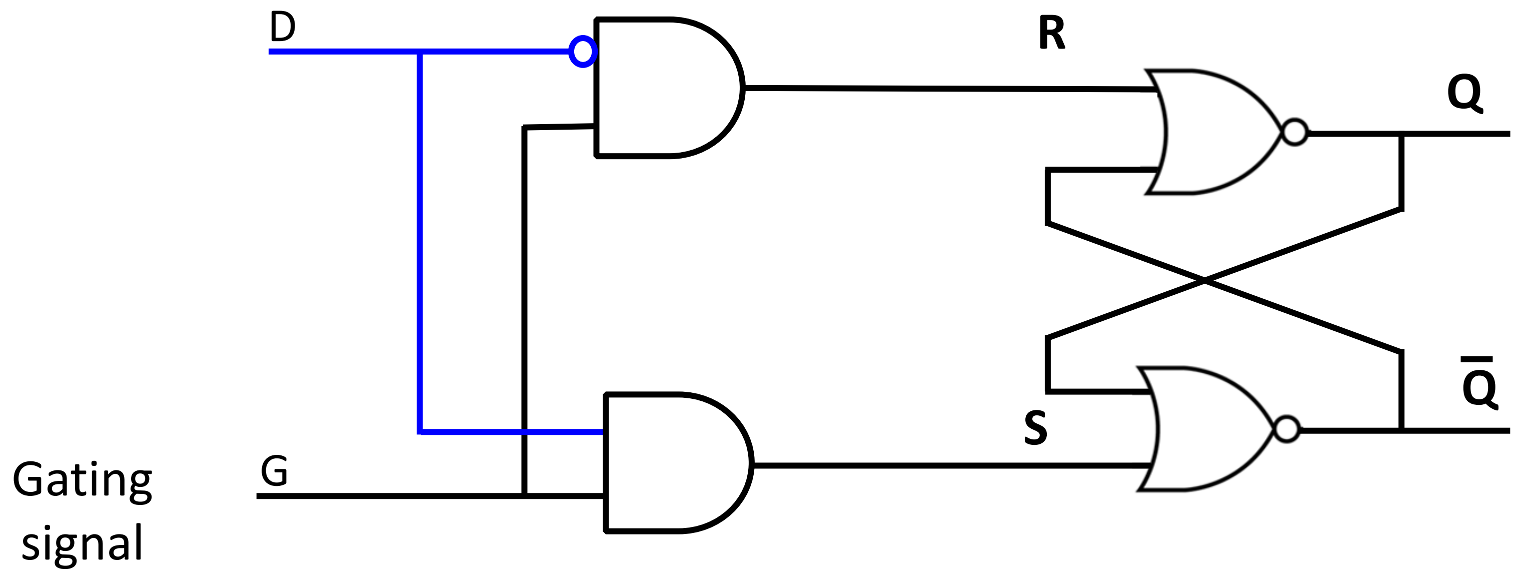
S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	0	0

Invalid ~ AVOID!

- Output Q and $\overline{Q}$ should have memory, i.e., retain their value for *some input changes*
- Output Q and $\overline{Q}$ should always have opposite values

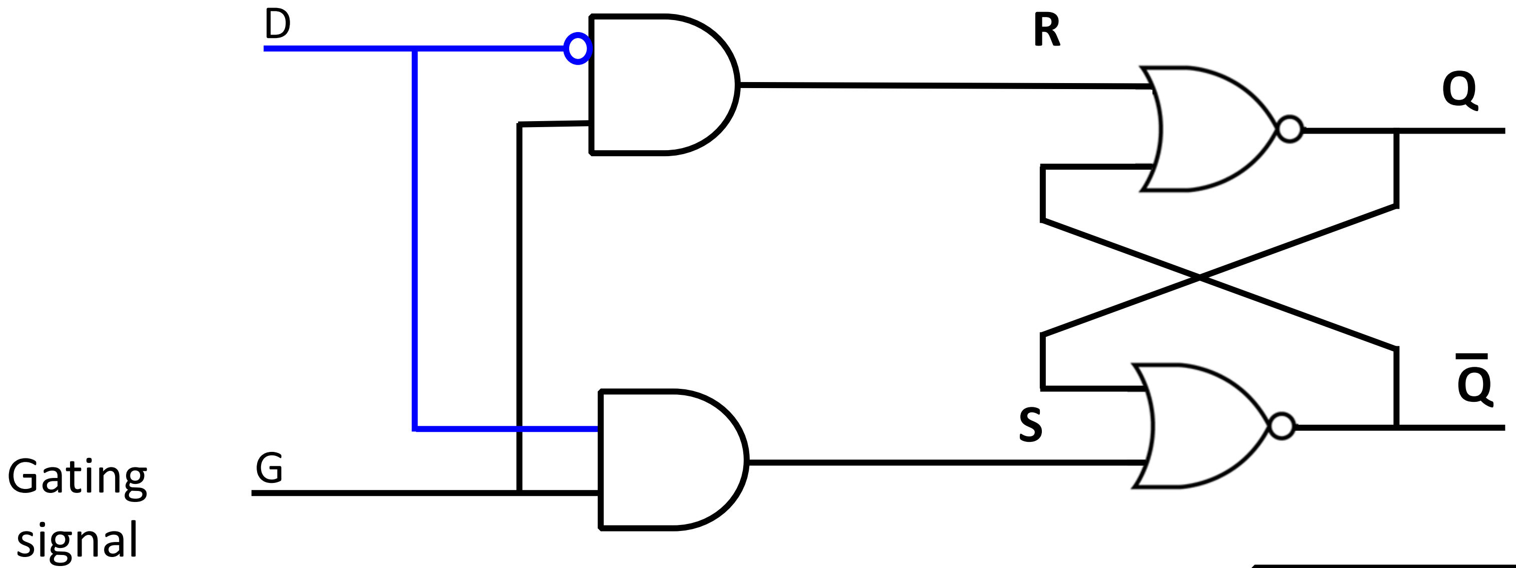
Q and $\overline{Q}$ are supposed to be opposite of each other, so **this is a state we avoid.** This state can also lead to unstable future states. Try setting S = 0 and R = 0 now!

D Latch



D	G	Q	$\bar{Q}$

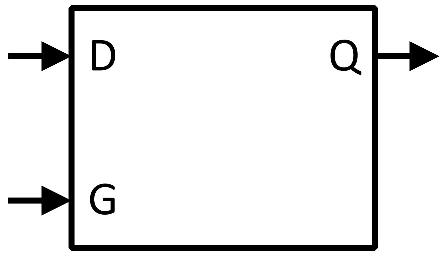
D Latch



D	G	Q	Q̄
0	0	Q	Q̄
0	1	0	1
1	0	Q	Q̄
1	1	1	0

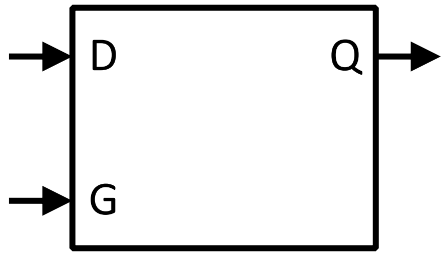
Next state is set

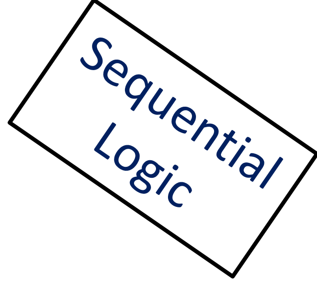
Set state is retained when gate is low

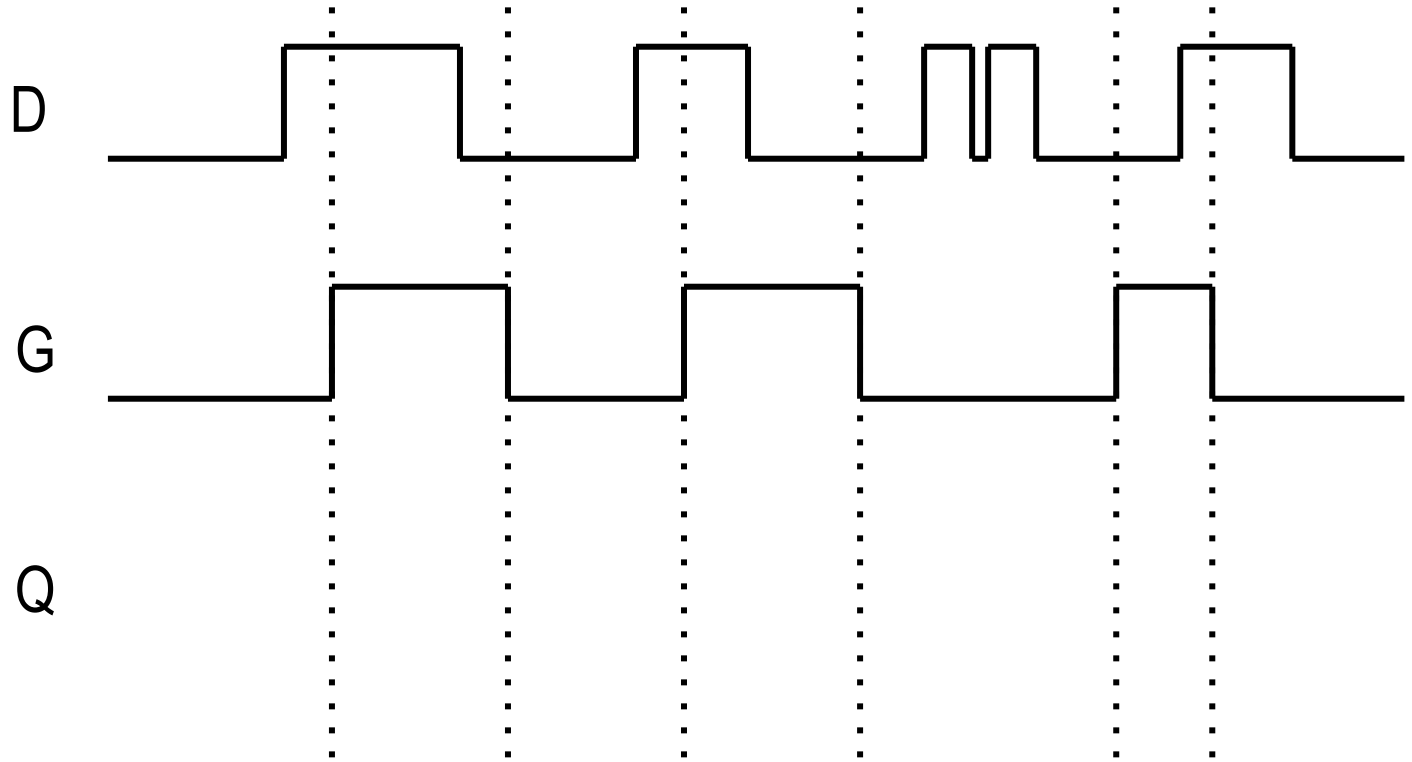


No invalid states 😊

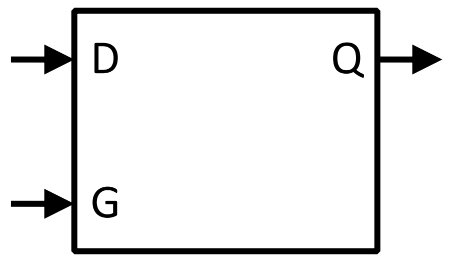
D Latch – Gate and Data



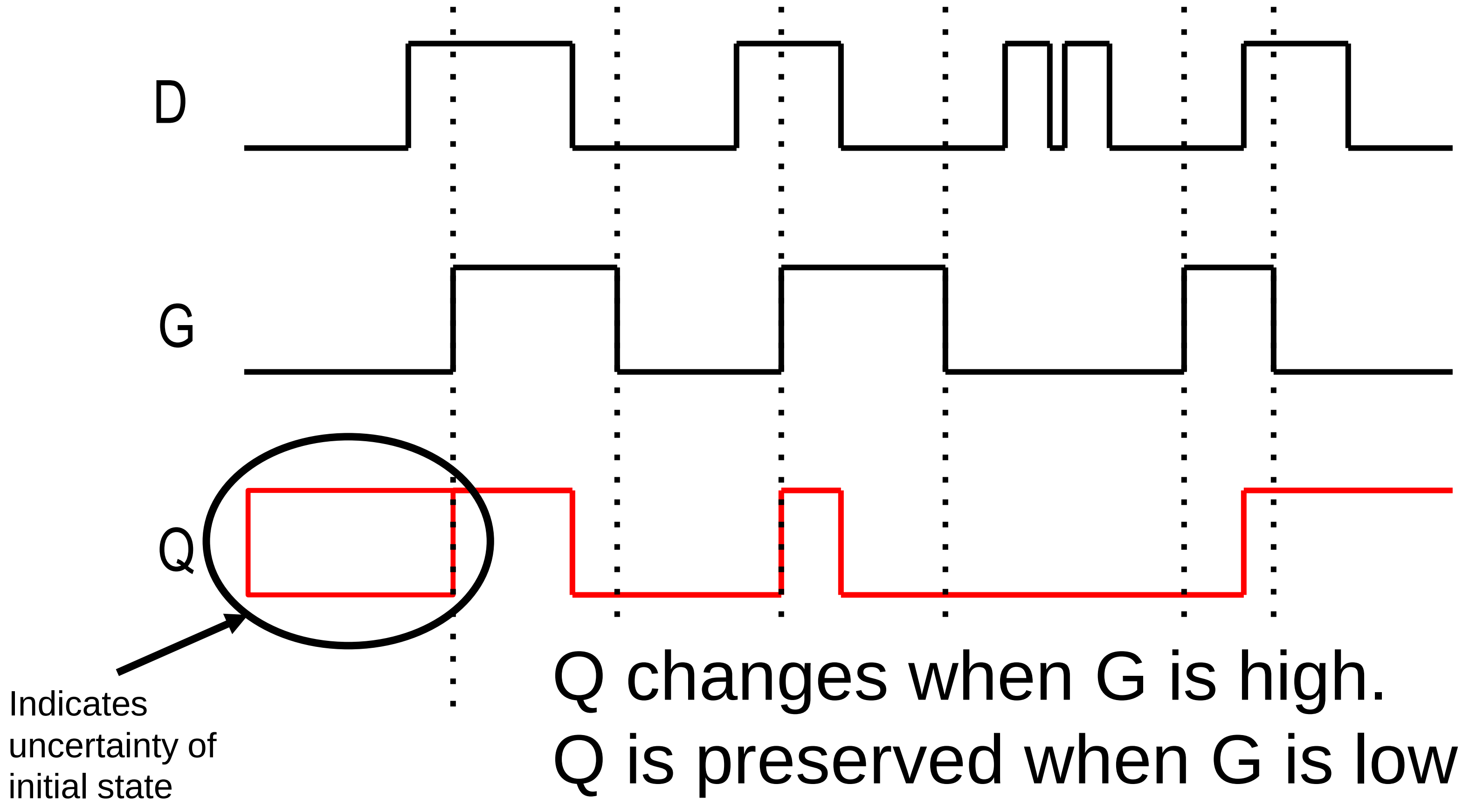




D Latch – Gate and Data



Sequential Logic



Q changes when G is high.
Q is preserved when G is low

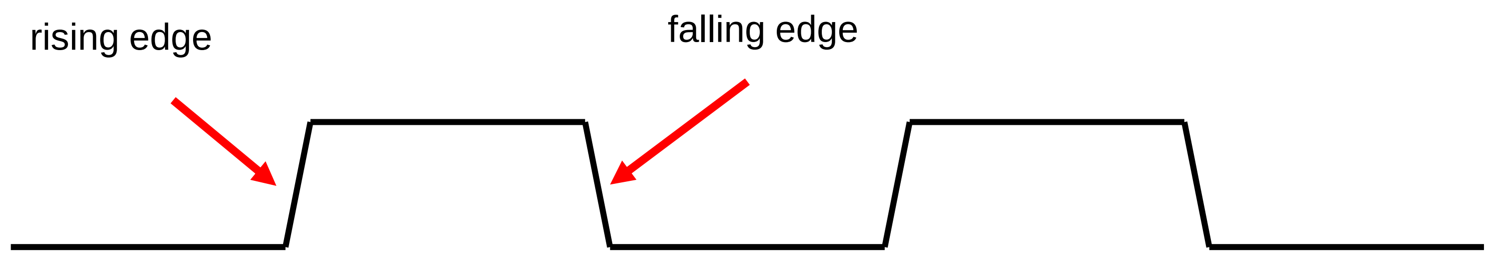
Adding a Clock to the Mix

- Problem if we build complex circuits with feedback, these "latches" can become unstable when transparent
  - "Glitches" propagate around and around
  - Take 270 to learn more
- We can solve this if we introduce a clock
  - Alternating signal that switches between 0 and 1 states at a fixed frequency (e.g., 100MHz)
  - Only store the value the instant the clock changes
- What should the clock frequency be?
  - It depends on the longest propagation delay between state and next state combination logic
  - And a few other things outside of the scope of 370 (shout out 270)

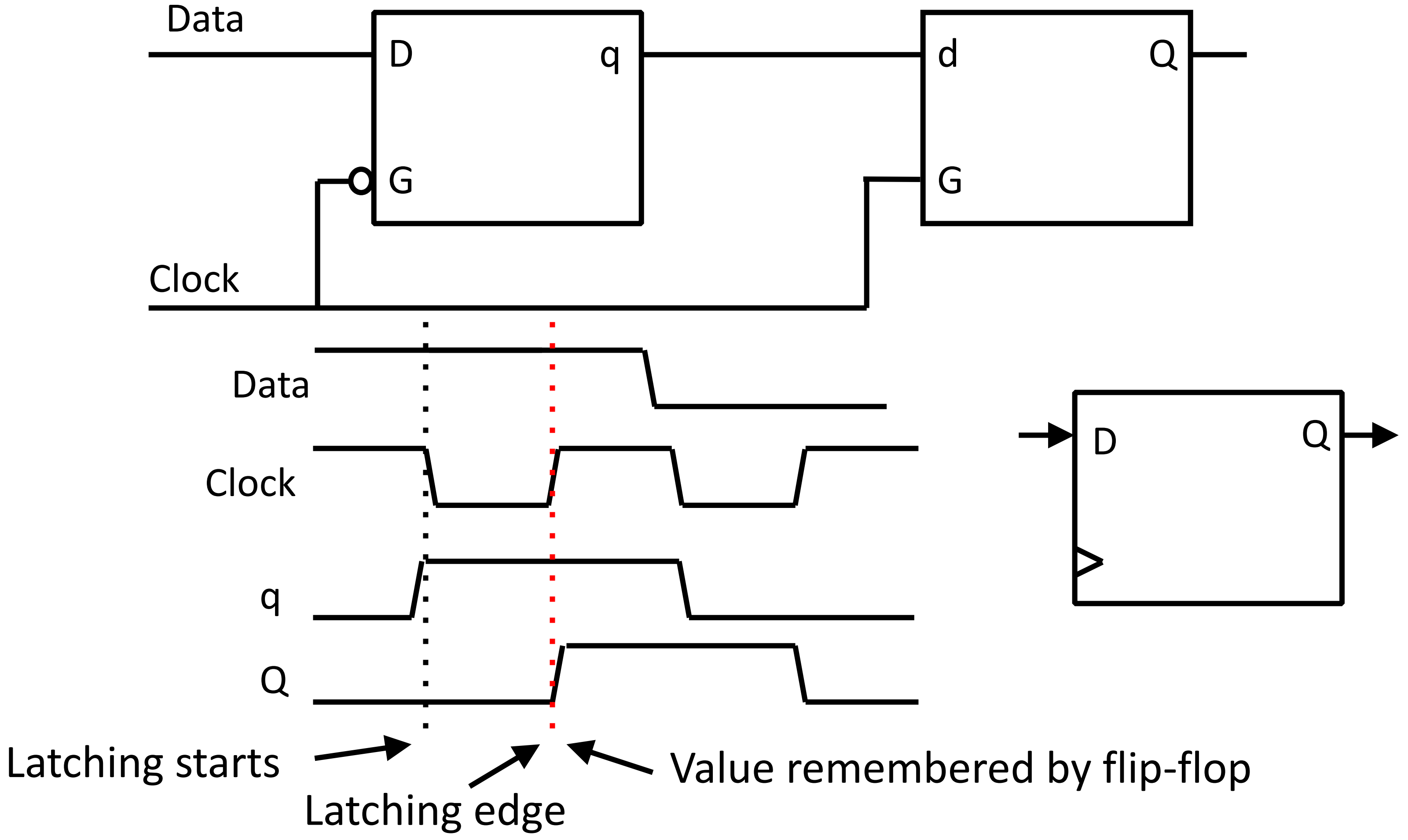
Clocks

Sequential Logic

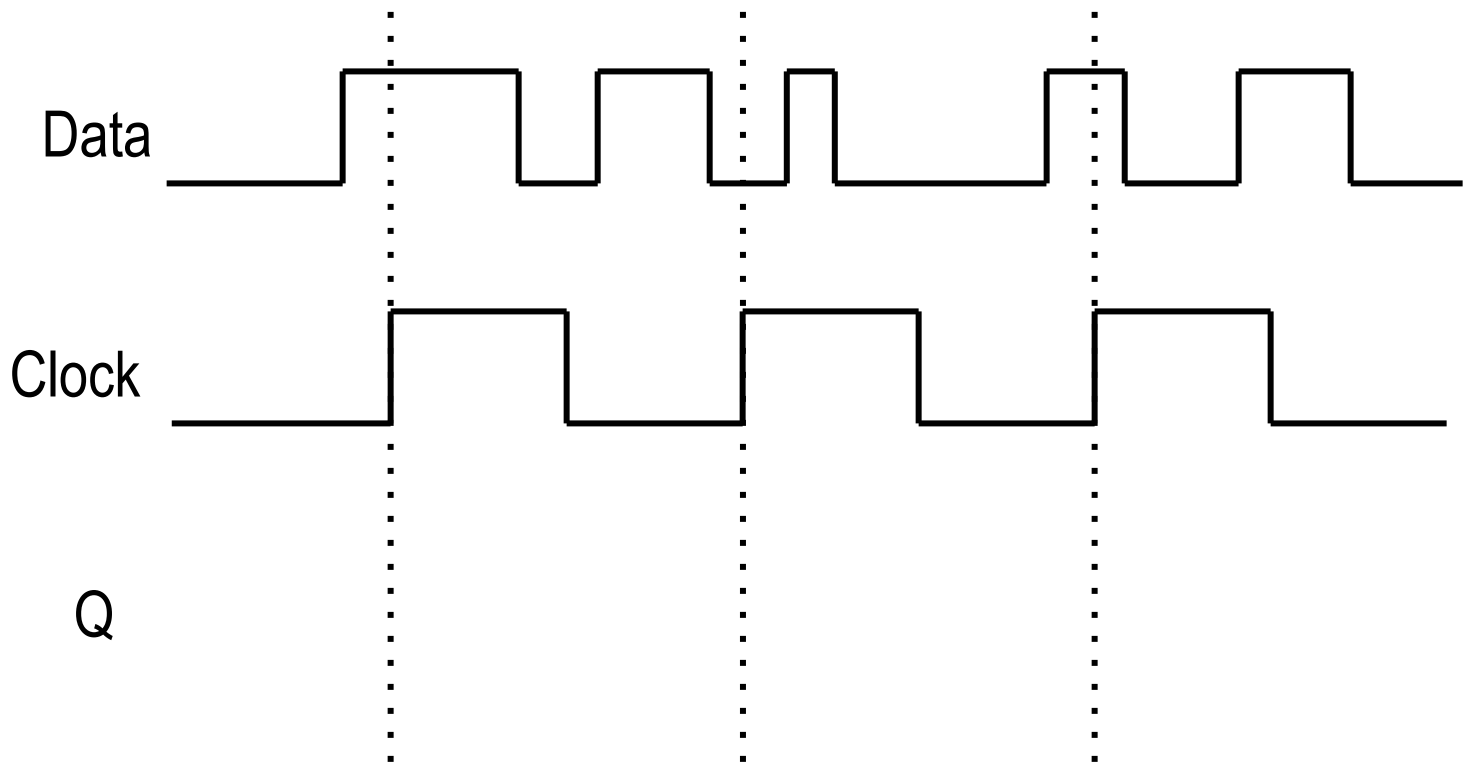
- Clock signal
  - Periodic pulse
  - Generated using oscillating crystal or ring oscillator
  - Distributed throughout chip using clock distribution net

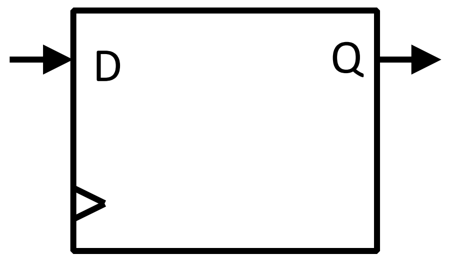


Rising-Edge Triggered D Flip-Flop

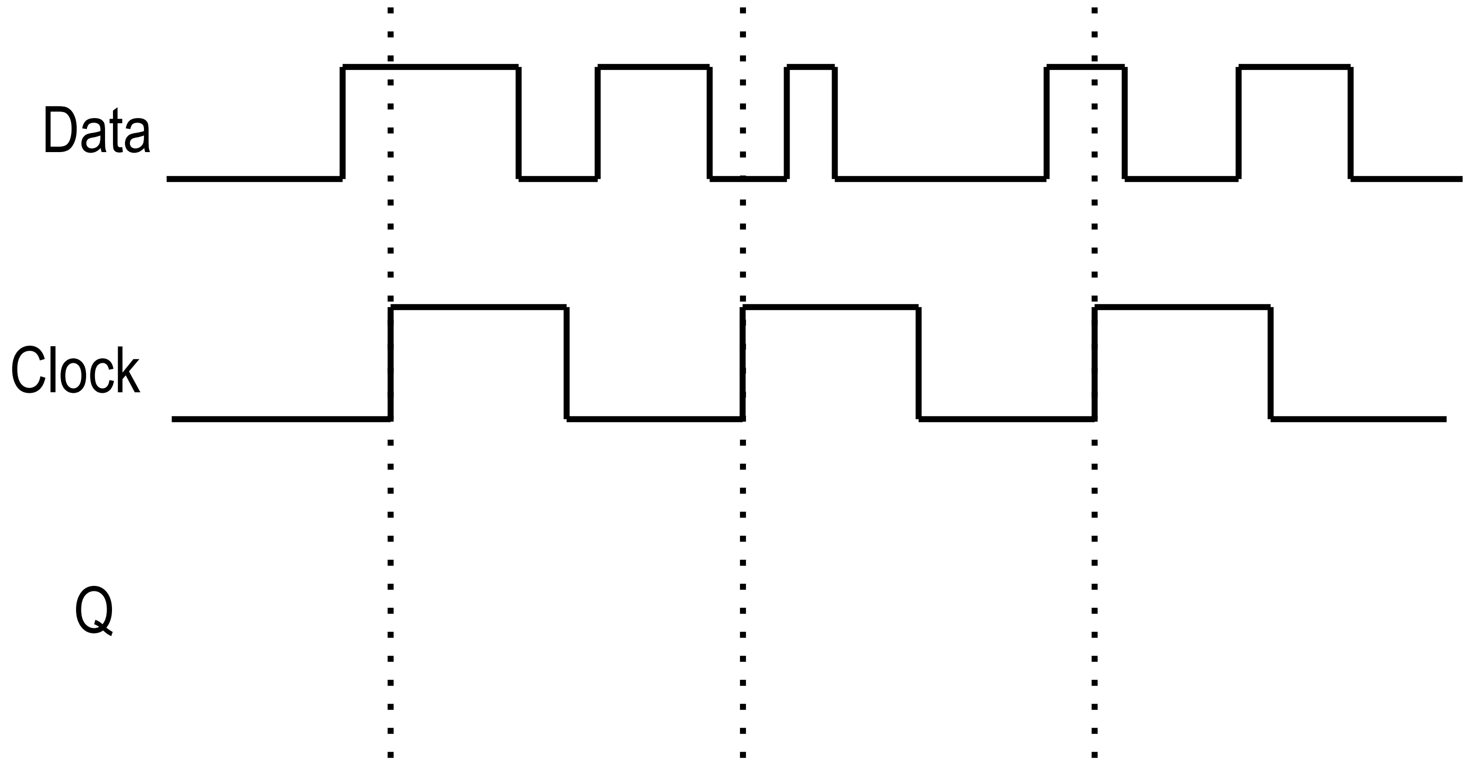


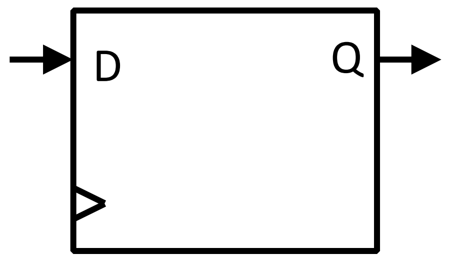
D Flip-Flop – Clock and Data



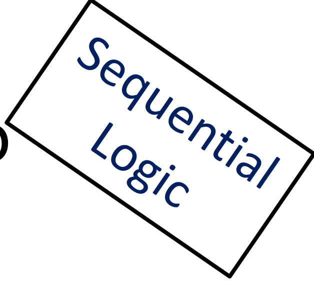


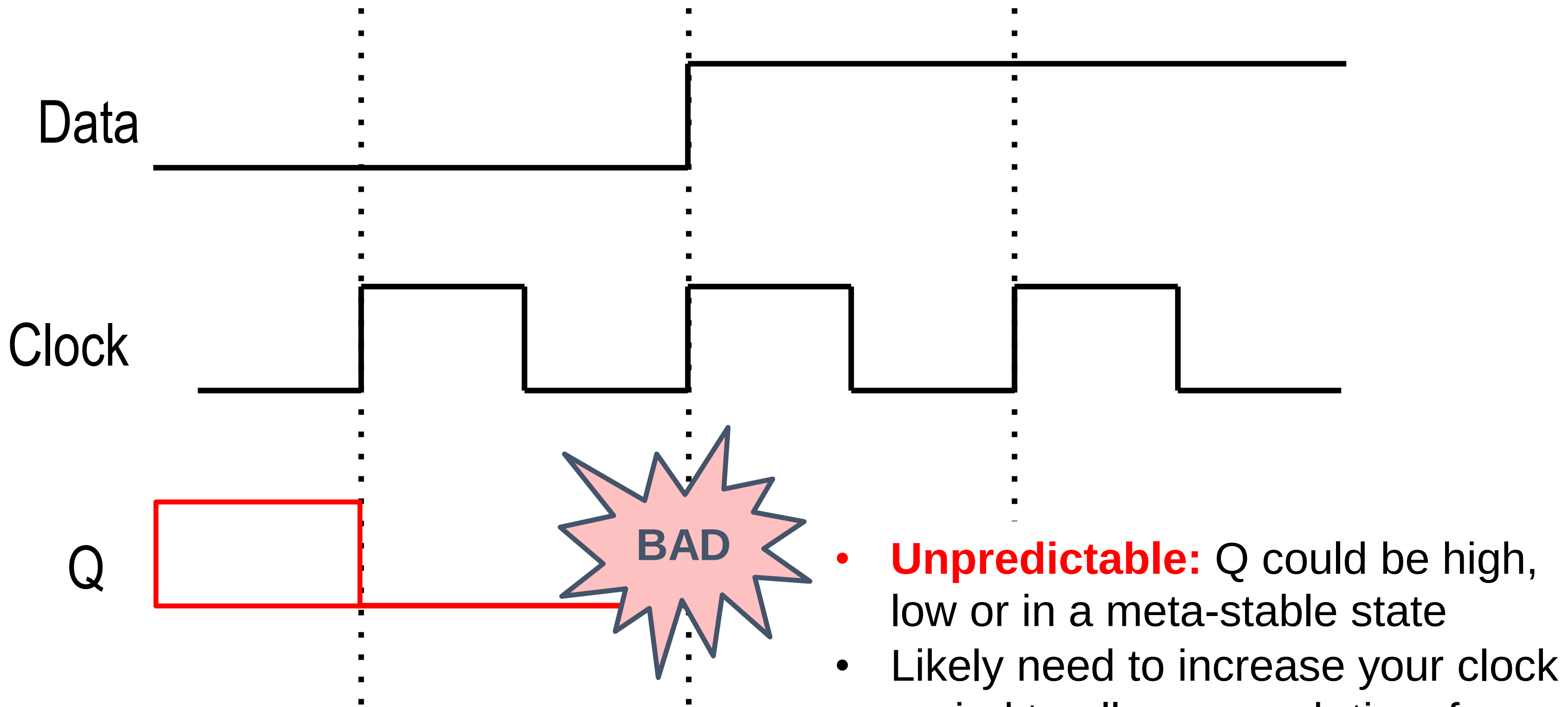
D Flip-Flop – Clock and Data

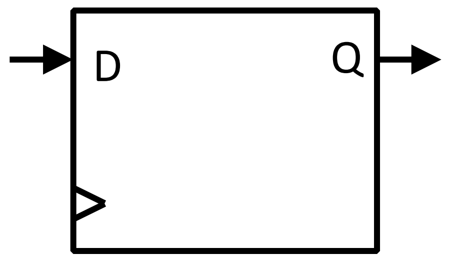




What Happens if Data Changes on a Clock Edge?



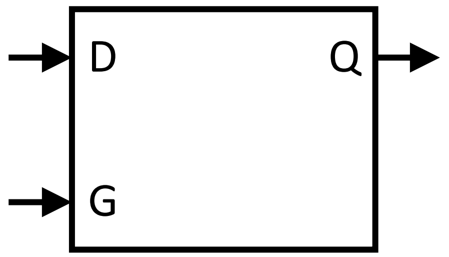




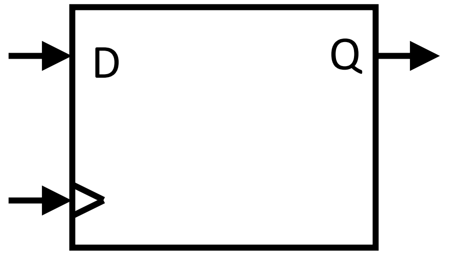
- **Unpredictable:** Q could be high, low or in a meta-stable state
- Likely need to increase your clock period to allow enough time for signal propagation

Why Edge-Triggered Flip-Flops?

Latch



Flip-flop

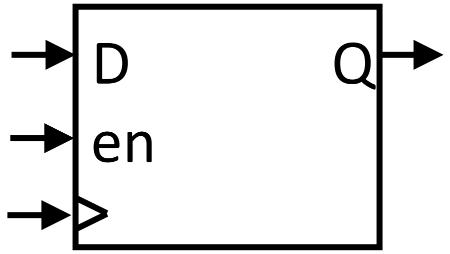


In edge-triggered flip-flops, the latching edge provides convenient abstraction of “instantaneous” change of state.

Adding an Enable Input

- Q only updates on a positive clock edge if ‘en’ is high

- Think of ‘en’ as ‘write enabled’



Logistics

- There are 3 videos for lecture 9
  - L9_1 – Combinational-Logic-Timing
  - L9_2 – Memory_Latches-Clocks
  - L9_3 – Finite-State-Machines
- There are two worksheet for lecture 9
  1. Circuit design – combinational logic
  2. Circuit design – sequential logic – you are ready for this now

L9_3 Finite-State-Machines

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

- To define and understand the concept of state as it pertains to architecture
- Ability to model a controller as a machine expressed as states and transitions, i.e., a finite state machine.

Finite State Machines

- So far we can do two things with gates:
  1. Combinational Logic: implement Boolean expressions
     - Adder, MUX, Decoder, logical operations etc
  2. Sequential Logic: store state
     - Latch, Flip-Flops

- How do we combine them to do something interesting?
  - Let's take a look at implementing the logic needed for a vending machine
  - Discrete states needed: remember how much money was input
    - Store sequentially
  - Transitions between states: money inserted, drink selected, etc
    - Calculate combinationally or with a control ROM (more on this later)

State

Very important concept in architecture

- Represents all the stored information in a system at a point in time
- Finite State Machine:
  - Model of a system which enumerates all states that system may be in, and the conditions which allow transitions between states
  - Often expressed as a directed graph or table

FSM Example – Vending Machine

- We could use a general purpose processor
- However, a custom controller will be:
  - Faster
  - Lower power
  - Cheaper to produce in high volume
- On the other hand, a custom controller:
  - Will be slower to design
  - More expensive in low volume
- Goals:
  - Take money, vend drinks.

FSM

Input and Output

- Inputs:
  - Coin trigger
  - Refund button
  - 10 drink selectors
  - 10 pressure sensors
    - Detect if there are still drinks left
- Outputs:
  - 10 drink release latches
  - Coin refund latch

Operation of Machine

- Accepts quarters only
- All drinks are $0.75
- Once we get the money, a drink can be selected
- If they want a refund, release any coins inserted

- No free drinks!
- No stealing money.

FSM

Building the Controller

- Finite State
  - Remember how many coins have been put in the machine and what inputs are acceptable
- Read-Only Memory (ROM)
  - Define the outputs and state transitions
- Custom combinational circuits
  - Reduce the size (and therefore cost) of the controller

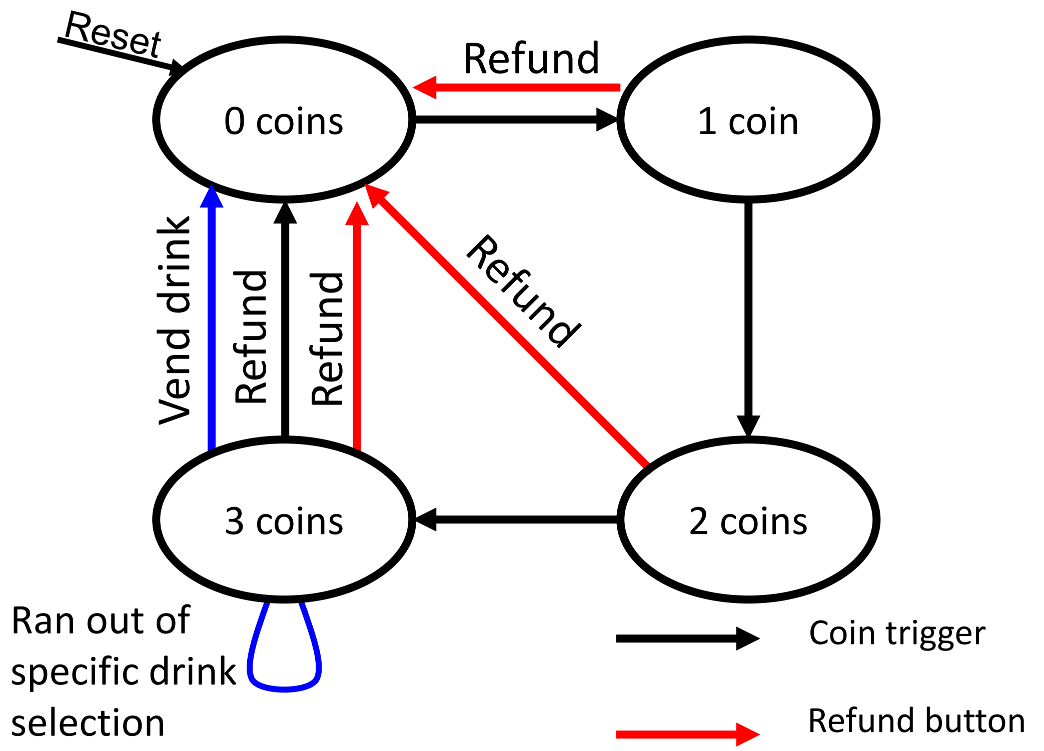
Finite State Machines

A Finite State Machine (FSM) consists of:

- K states: `S = {s1, s2, … ,sk}, s1 is initial state`
- N inputs: `I = {i1, i2, … ,in}`
- M outputs: `O = {o1, o2, … ,om}`
- Transition function `T(S,I)` mapping each current state and input to next state
- Output Function `P(S)` or `P(S,I)` specifies output
  - `P(S)` is a Moore Machine
  - `P(S,I)` is a Mealy Machine

FSM

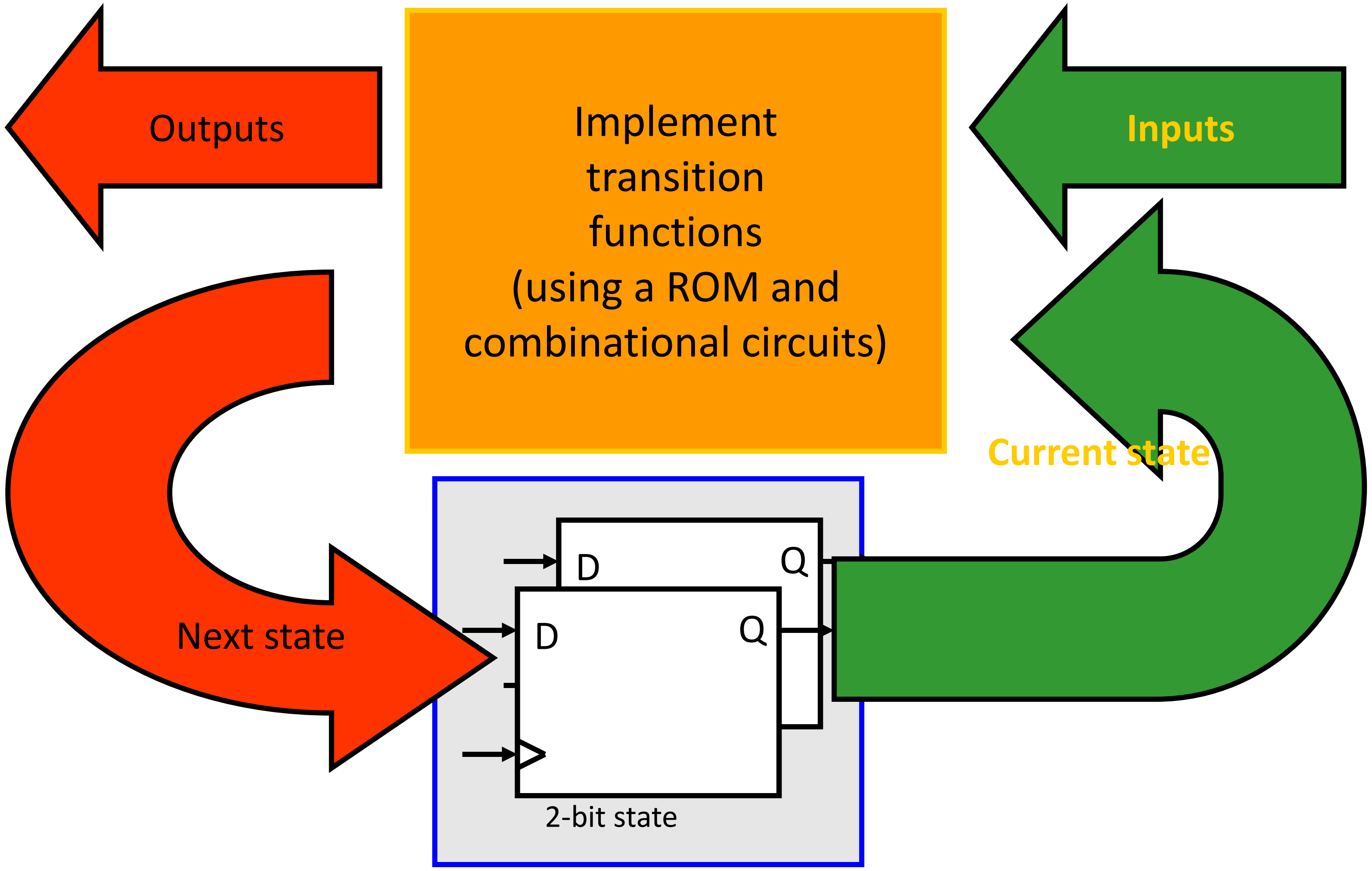
FSM for Vending Machine



Is this a Mealy or Moore Machine?

Mealy ~ output is based on current state *AND* input

Implementing a FSM


Outputs
Implement transition functions (using a ROM and combinational circuits)
Inputs
Current state
Next state
D
Q
D
Q
2-bit state

Logistics

- There are 3 videos for lecture 9
  - L9_1 – Combinational-Logic-Timing
  - L9_2 – Memory_Latches-Clocks
  - L9_3 – Finite-State-Machines
- There are two worksheet for lecture 9
  1. Circuit design – combinational logic
  2. Circuit design – sequential logic