Propogation Delay and State

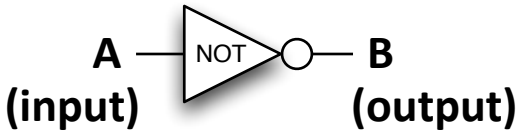
Today's lecture

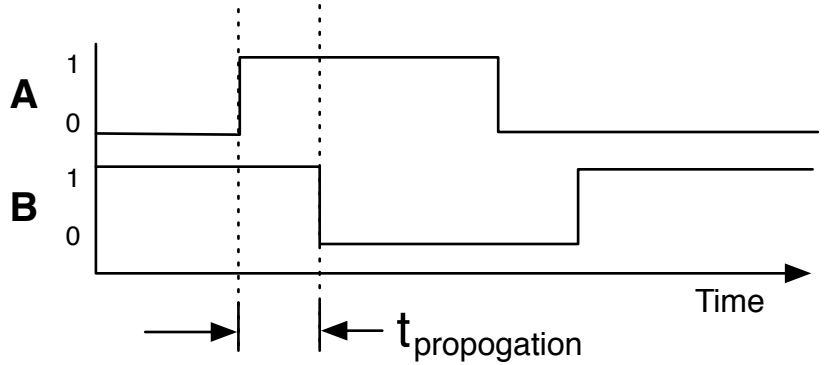
- **Propogation Delay**
  - Timing diagrams.
  - Delay of ALU32
- **Storing State**
  - SR Latch
- **Synchronous Design**
  - Clocks
  - D Flip Flops

Propogation Delay

- **Real gates don't switch instantaneously**
  - There is a latency between when the input changes and the output changes.
  - We call this latency the propagation delay

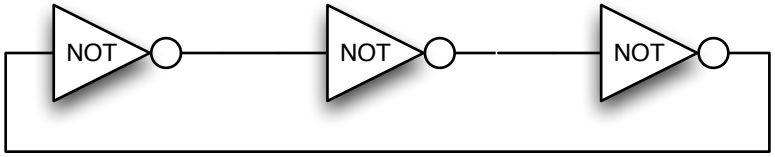
- **Consider an inverter:**



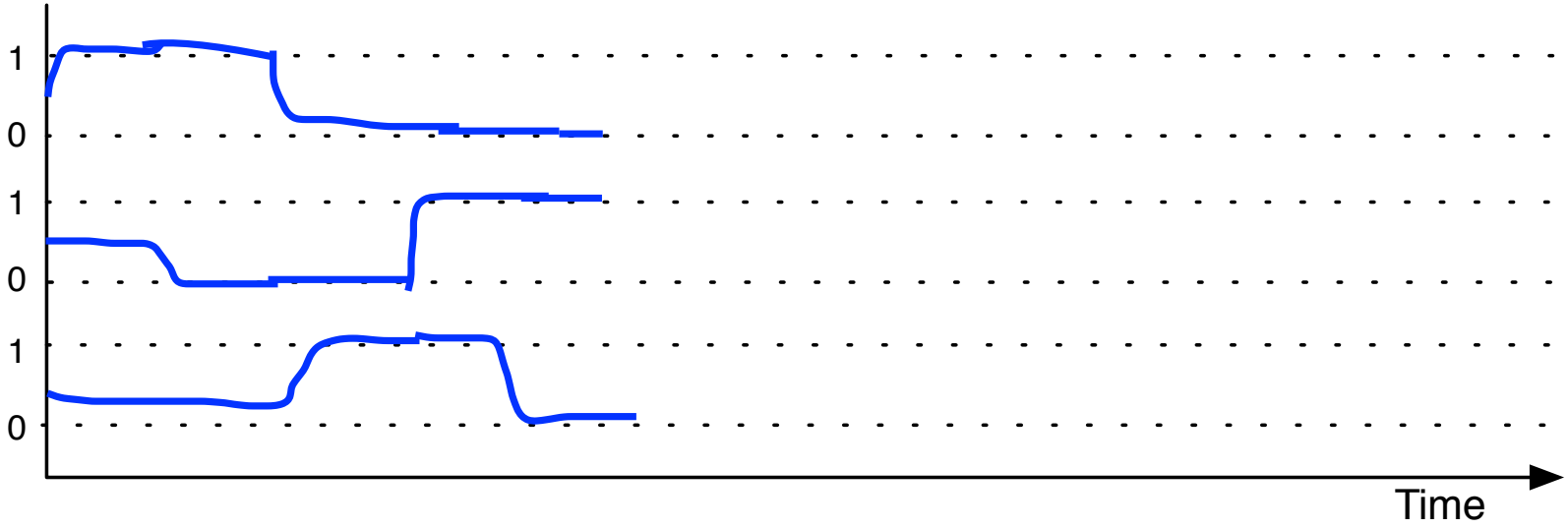


Ring Oscillator

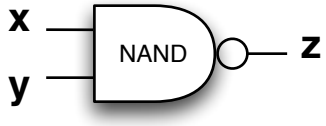
- **What happens when you connect an odd number of inverters in a circle?**



If we connect even number of inverters, the state would become stable and we can store a bit inside



Timing analysis



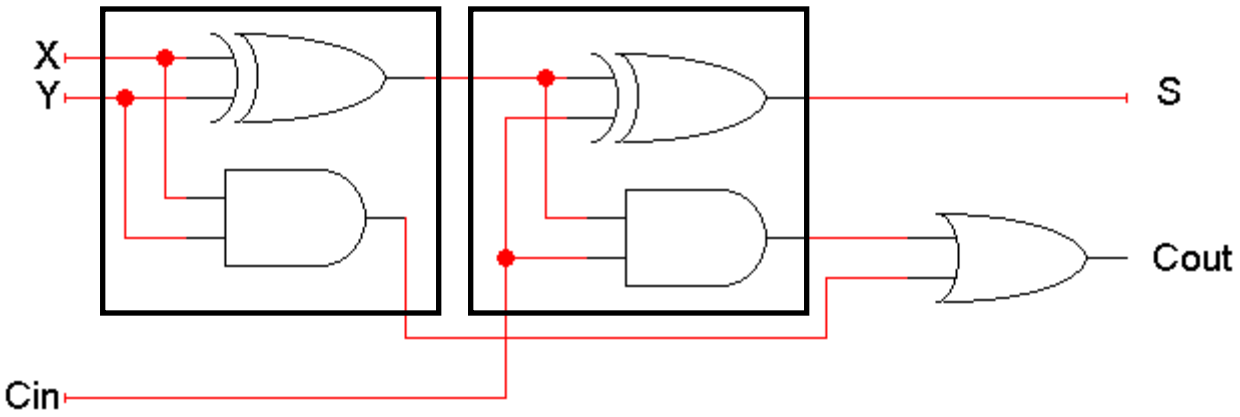
- **In reality, timing is very complicated**
  - The delay from **x** to **z** can be different on an 0➔1 transition than it is for a 1➔0 transition.
  - The delay from **x** to **z** can differ from the delay from **y** to **z**.
  - The number of gates connected to the same output (the fanout), the longer it will take to switch.
  - Long wires between gates slow things down as well.

- **In this class, we'll use simplifying assumptions:**
  - **Delay is a constant from any input to the output.**
  - **We'll ignore fanout and wire delay**

Analyzing propagation delay of circuits

- **Assume:**
  - Inverter: 20ps delay
  - 2-input gate: 30ps delay
  - 4-input gate: 50ps delay

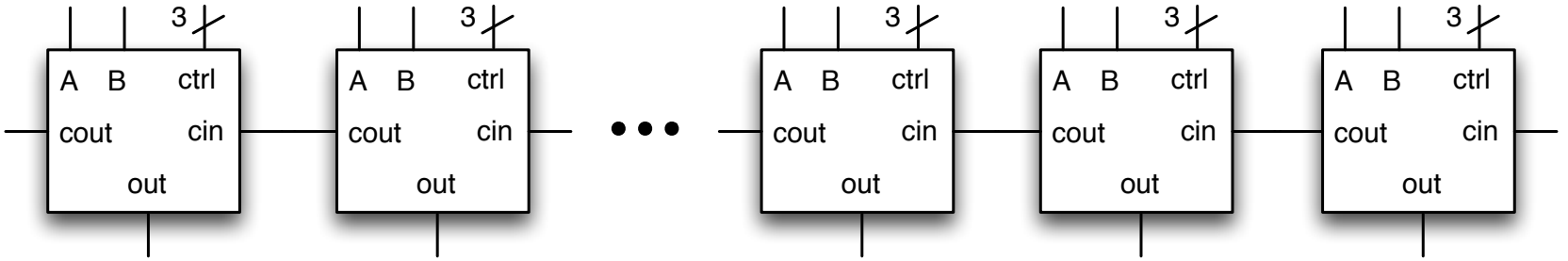
- **Find longest paths from input to output**



In	Out	Delay
X	S	
Y	S	
Cin	S	
X	Cout	
Y	Cout	
Cin	Cout	

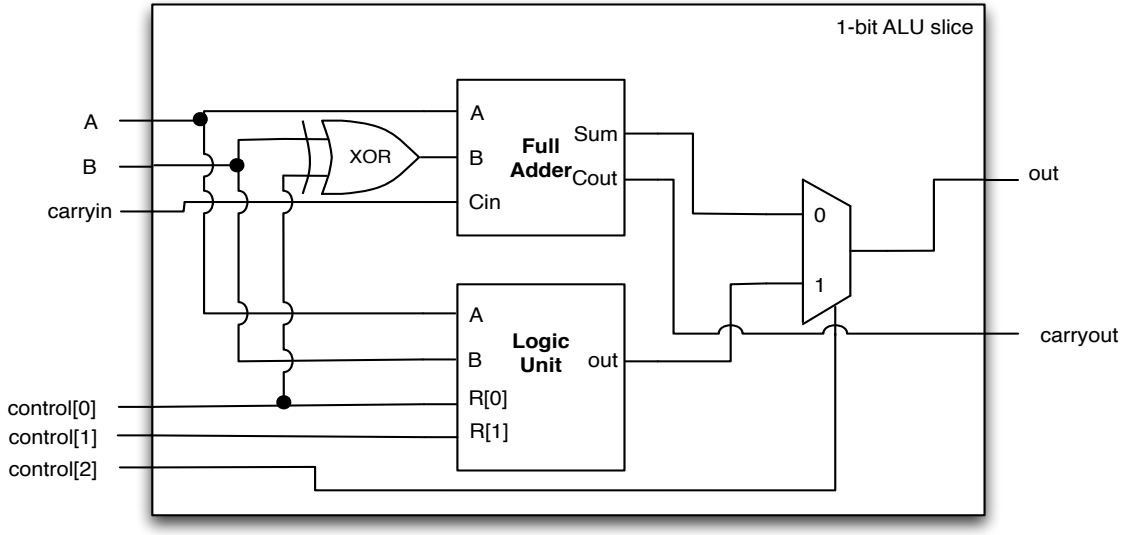
Computing the delay of alu32

- What is likely to be the longest path through this circuit?
  - From any data input (A, B) bit to any output bit



- How long will it take?

Computing components from ALU1



$t_{PropBCarryout} + 30(t_{PropCarryinCarryout}) + t_{PropCarryinOut} =$

XOR gate

In	Out	Delay
A,B	out	30ps

Full Adder

In	Out	Delay
A,B	Sum	
Cin	Sum	
A,B	Cout	
Cin	Cout	

Logic Unit

In	Out	Delay
A,B	out	110ps
R	out	10ps

2-to-1 Multiplexor

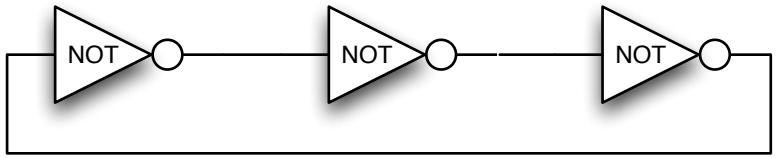
In	Out	Delay
A,B	out	60ps
R	out	80ps

Thinking about ALU32's delay

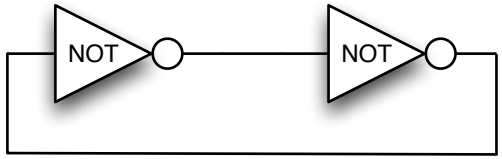
- **That is really bad. Really, really bad.**
- **Processors don't really implement ADDs this way**
  - This is what is called a "ripple carry adder"
  - It has latency ***O(n)*** where ***n*** is the # bits being added
- **There are much smarter ways to handle carries**
  - E.g., "carry lookahead adder" (Google it)
  - Has latency ***O(log$_2$(N))*** and only slightly larger
- **But, we won't cover them in this class.**

Rings of Inverters

- **We saw an odd number of inverters creates a ring oscillator:**



- **What happens if you have an even number of inverters?**



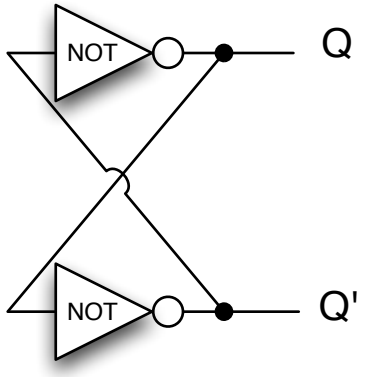
If we connect even number of inverters, the state would become stable and we can store a bit inside

Building a useful storage mechanism

- **A memory should have at least three properties.**

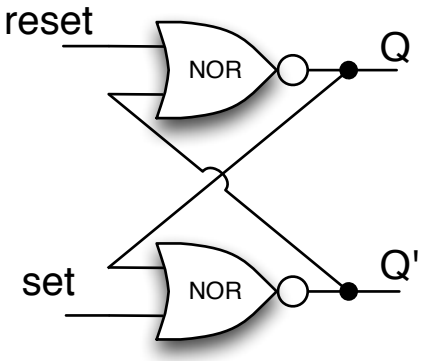
  1. It should be able to hold a value. ✓
  2. You should be able to *read* the value that was saved. ✓
  3. You should be able to *change* the value that's saved. ✗

Cross-coupled inverters:



Set/Reset Latch (SR latch)

- **Cross-coupled NOR gates**
  - Two inputs:
    - **reset**: when 1, sets Q=0
    - **set**: when 1, sets Q=1
  - When **reset**=**set**=0, Q keeps its value
    - ***reset=set=1*** *causes bad things. Make sure this doesn't happen.*



- **This circuit has feedback, its outputs (Q, Q') are also inputs!**
  - Current values of Q, Q' depend on past values of Q, Q'

$$Q_{t=x} =$$

$$Q'_{t=x} =$$

Analyzing SR latch

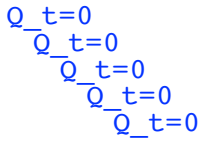
$Q_{t=x} = (reset + Q'_{t=x-1})'$

$Q'_{t=x} = (set + Q_{t=x-1})'$

- **Case 1:** reset = set = 0 @ t = 0

  $Q_{t=1}$ =

  $Q'_{t=1}$ =



- **Case 2:** reset = 1, set = 0

  $Q_{t=1}$ =

  $Q'_{t=1}$ =

- **Case 3:** reset = 0, set = 1

  $Q_{t=1}$ =

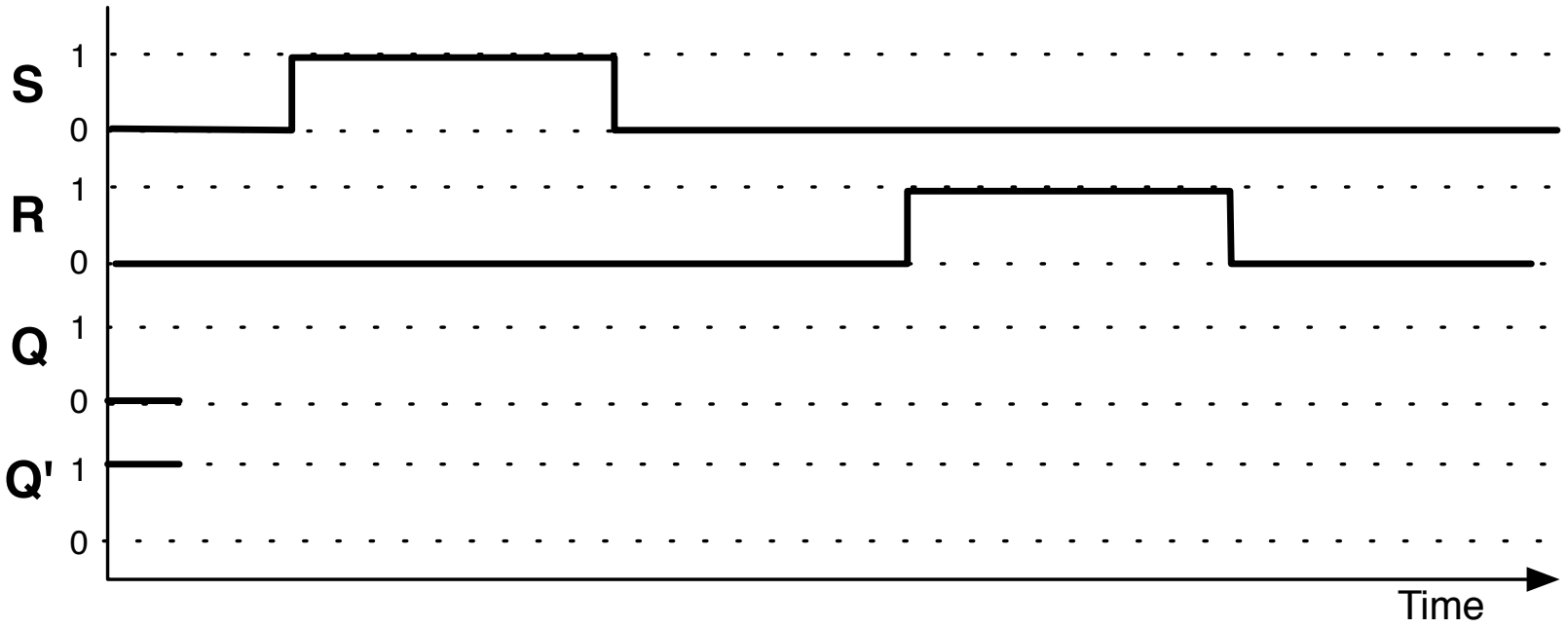
  $Q'_{t=1}$ =

Timing diagram of an SR Latch

Initialized to Q=0

$Q_{t=x} = (reset + Q'_{t=x-1})'$

$Q'_{t=x} = (set + Q_{t=x-1})'$



SR Latches are a useful storage mechanism

- **An SR latch provides the three features:**
  - It holds its value
  - We can read its value
  - We can change its value

S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

- **We call the data stored (Q) the state of the latch.**
  - 1 bit of information is stored

- **We the behavior as a state table, which explicitly shows that the *next* values of Q and Q' depend on their *current* values, as well as on the inputs S and R.**

Inputs		Current		Next	
S	R	Q	Q'	Q	Q'
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

Aside: SR Latches are not combinational circuits

- **In the 2nd lecture we defined Cominational Logic as:**

  Boolean circuits where the output is a pure function of the present input only.

- **Below we can see this doesn't hold for the SR Latch**
  - When **S = R = 0** we can have two different outputs

- **The SR Latch is a Sequential circuit**

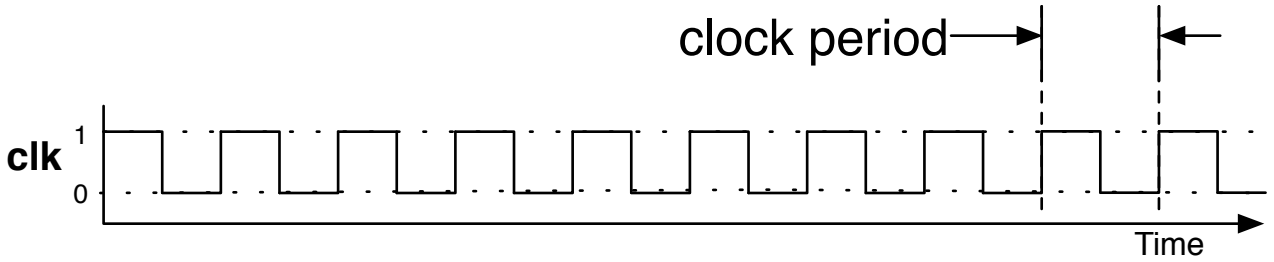
The outputs of a sequential circuit depend on not only the inputs, but also the state, or the current contents of some memory.

Inputs		Current		Next	
S	R	Q	Q'	Q	Q'
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

Synchronous Design

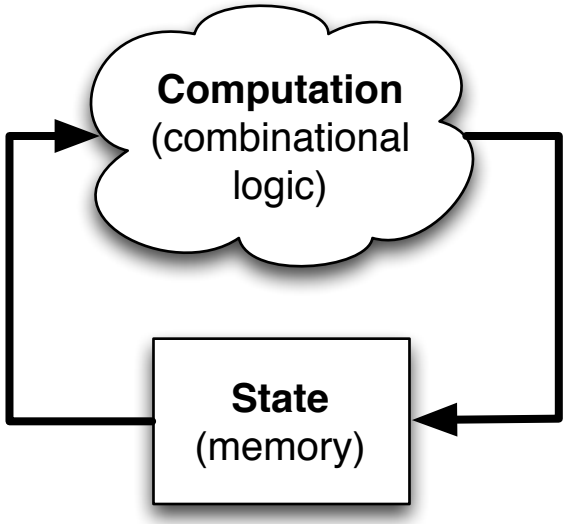
- ***The easiest (and most common) way to build computers***
- **All state elements get updated at the same time**
  - Using a clock signal

- **Clock signal**
  - A square wave with a constant period

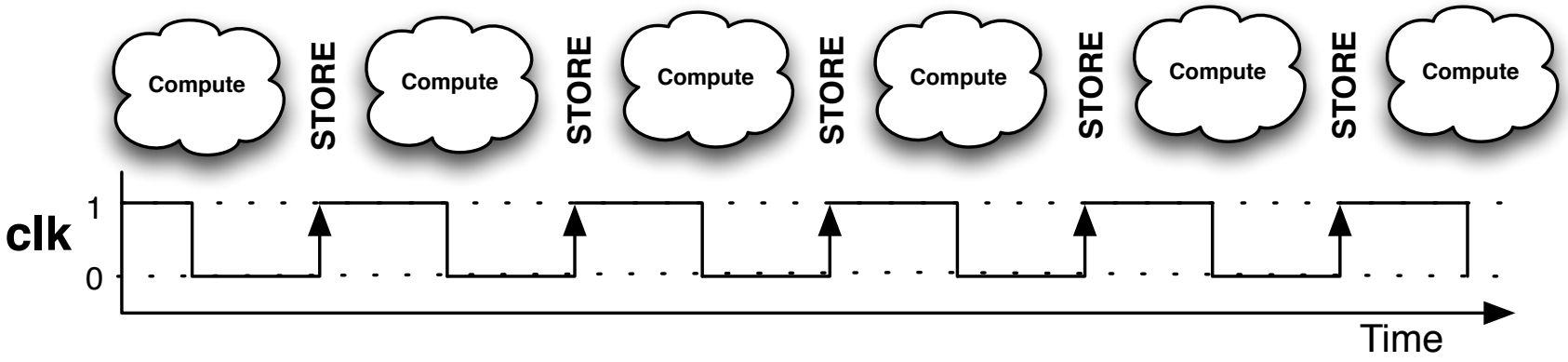


- **We always update state at the same point in wave**
  - E.g., the rising edge

Synchronous Design, cont.

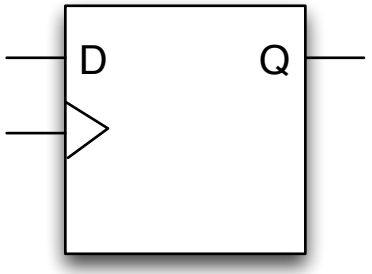


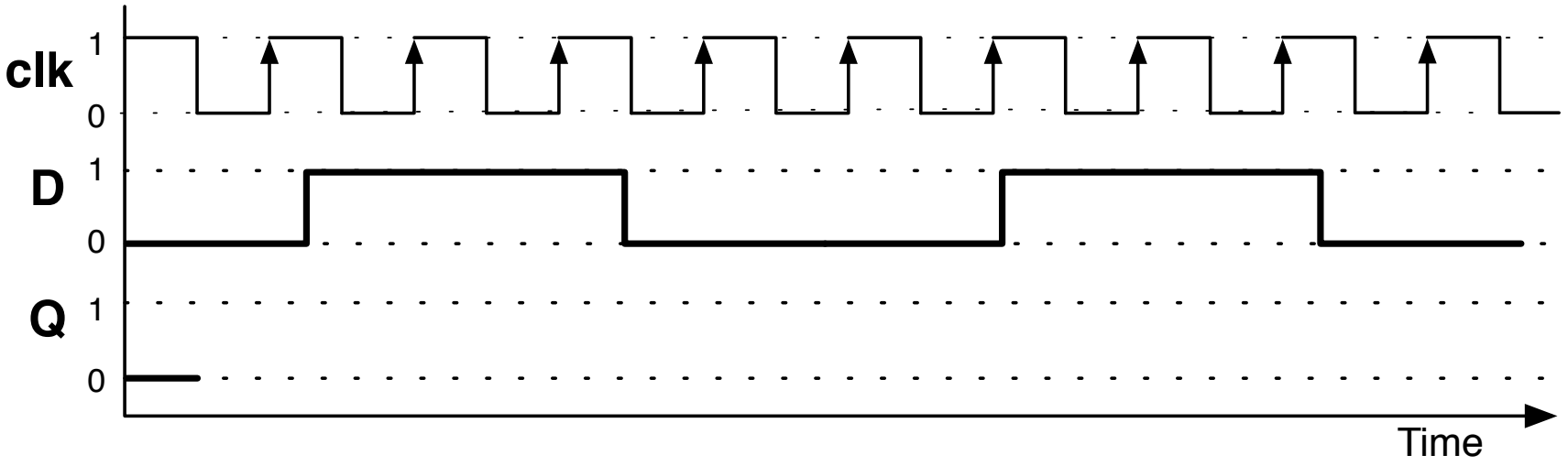
- **Alternate between computation and updating state.**



The state element that we really want…

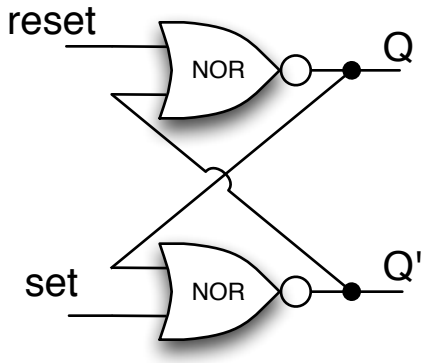
- **The D flip flop**
  - Holds 1 bit of state
    - Output as Q.
  - Inputs
    - Copies D input into state on rising edge of clock.





Implementing the D-type Flip Flop

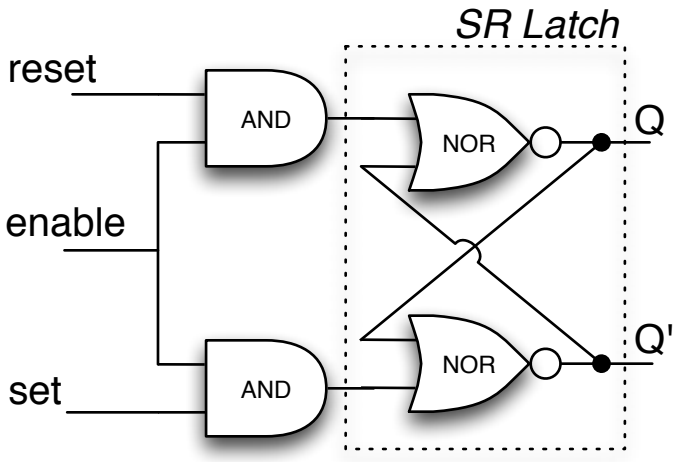
- **Remember the SR Latch?**



S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

- **We're going to two special kinds of latches.**
  - SR Latch with enable
  - D Latch with enable

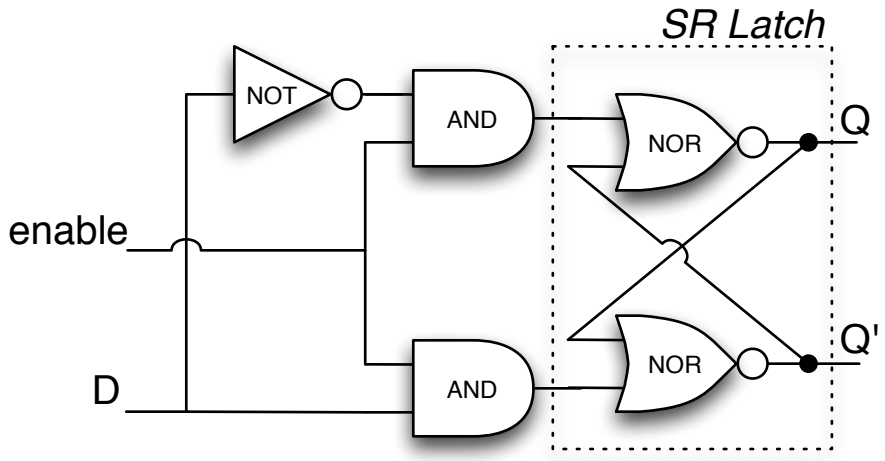
SR Latch with Enable



enable	S	R	S'	R'	Q
0	x	x	1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	Avoid!

- **Take SR Latch, make it ignore input when enable = 0**

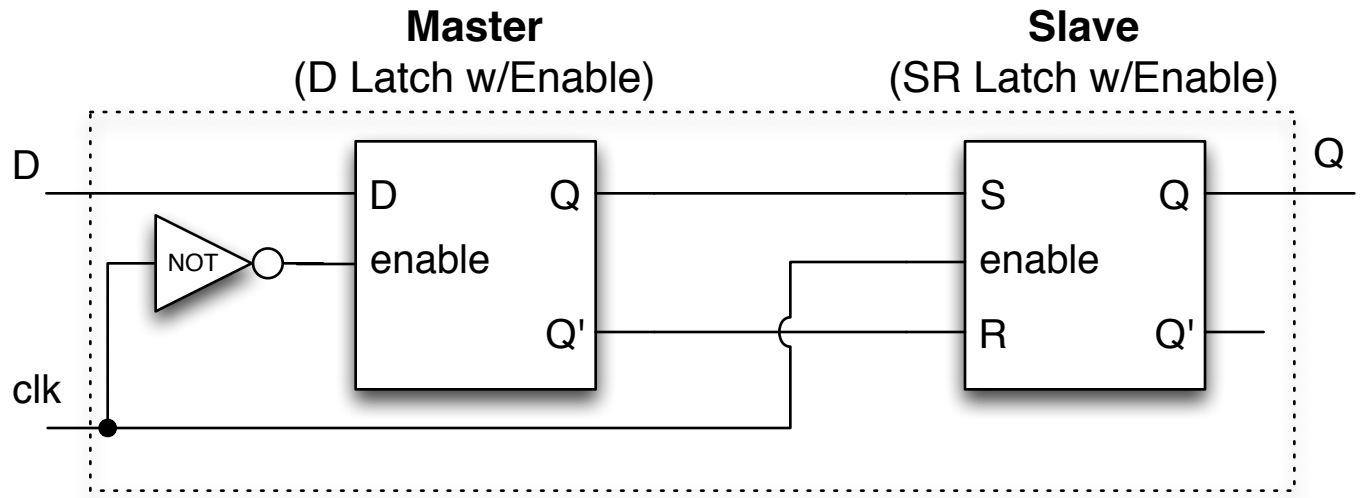
D-latch with Enable



Enable	D	Q
0	X	No change
1	0	0
1	1	1

- **Take SR Latch with Enable, make reset = !set**
  - State get set to whatever D is.

The D Flip Flop



- **Only one of the latches is enabled at a time**
  - When clk=0, the master is transparent; slave holds its value
  - When clk=1, the master holds its value; slave transparent

- **On rising edge, value at master's input passed to slaves output.**