

# CPEN 311: Digital Systems Design

## Slide Set 12: Circuit Timing: Part 1: The Basics

2016/2017 Term 1

Instructor: Steve Wilton

[steve.w@ece.ubc.ca](mailto:steve.w@ece.ubc.ca)

# Learning Objectives

---

1. Understand the RC model for gate delay and where it comes from
2. Be able to find the critical path, and hence the maximum clock speed, of a logic circuit (with or without flip-flops)
3. To understand set-up time, clk-q time, and hold time of a flip-flop
4. Understand setup and hold timing requirements of a design
5. Given a circuit, be able to calculate the maximum clock frequency
6. Given a circuit, be able to calculate the minimum hold time on the flip-flops

# The Need for Speed

---

Speed (“Performance”) of your circuit is one of the top two optimization goals (power is the other)

Often, a circuit has strict timing constraints that need to be met:

- Throughput constraints (eg. handle 10 GB/sec sustained)
- Processing constraints: total time required for answer

Time perform an operation = # cycles \* cycle time

- Number of cycles is optimized when designing datapath/controller
- Cycle time is the focus of this (and a subsequent) slide set

This slide set: Basics of timing analysis

Coming later: Practical considerations



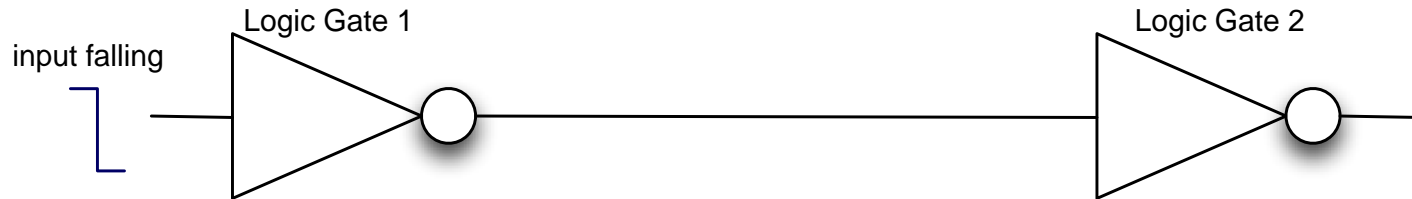
What does delay mean in hardware?

- You might imagine the delay of a gate might somehow be “the time for electrons to get through a gate

That is not quite right. Let's be a bit more precise.

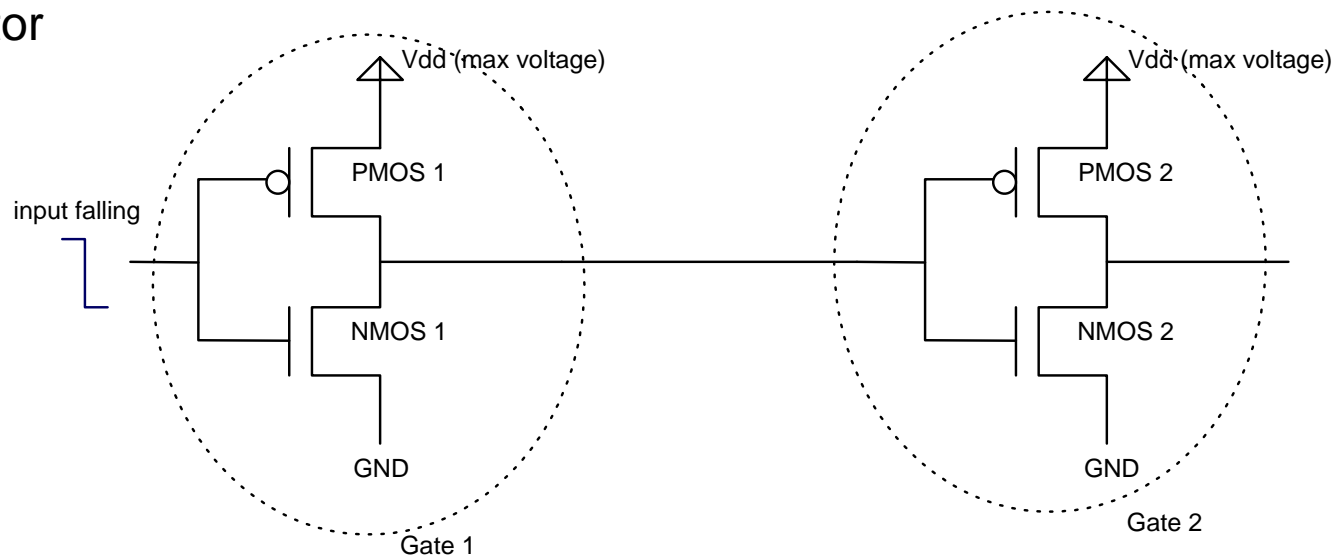
# Fundamentals: Delay of a gate

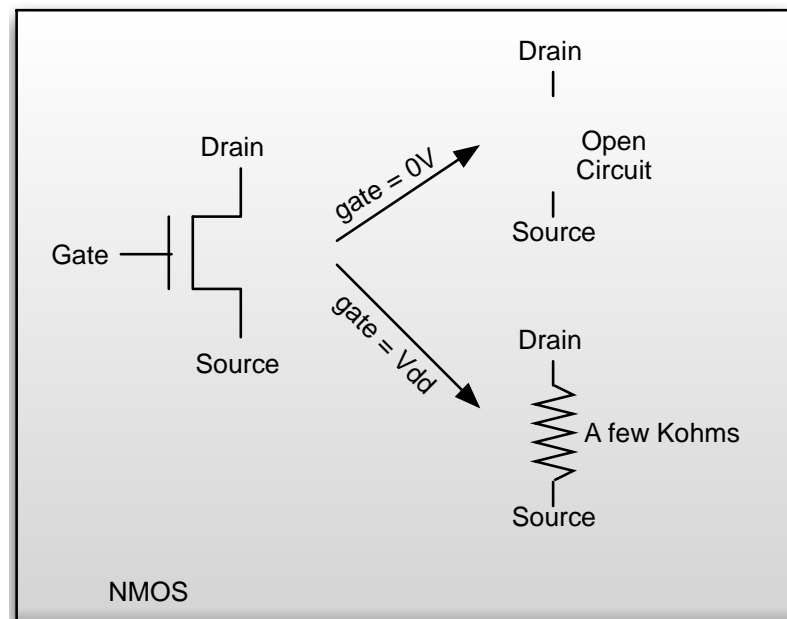
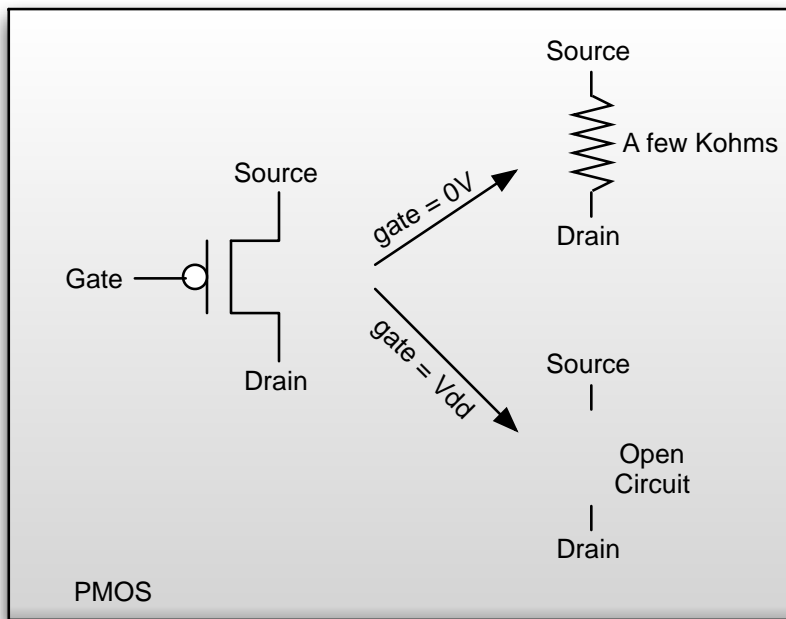
Consider Logic Gate 1 driving Logic Gate 2:



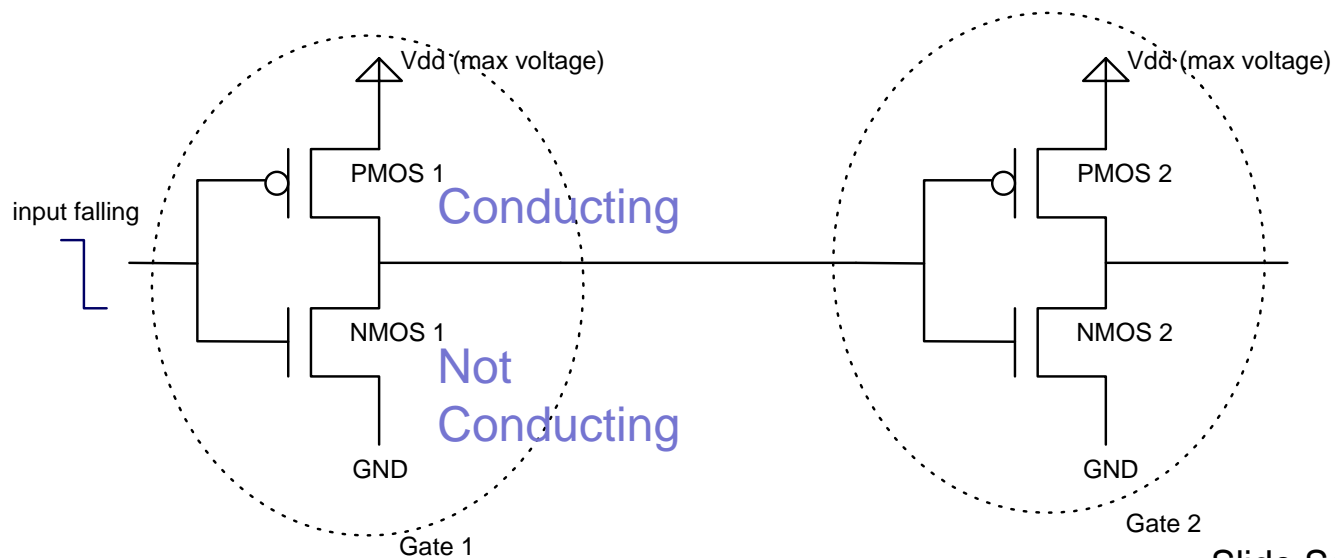
Physically, what happens?

Recall from CPEN 211, and inverter consists of one PMOS and one NMOS transistor



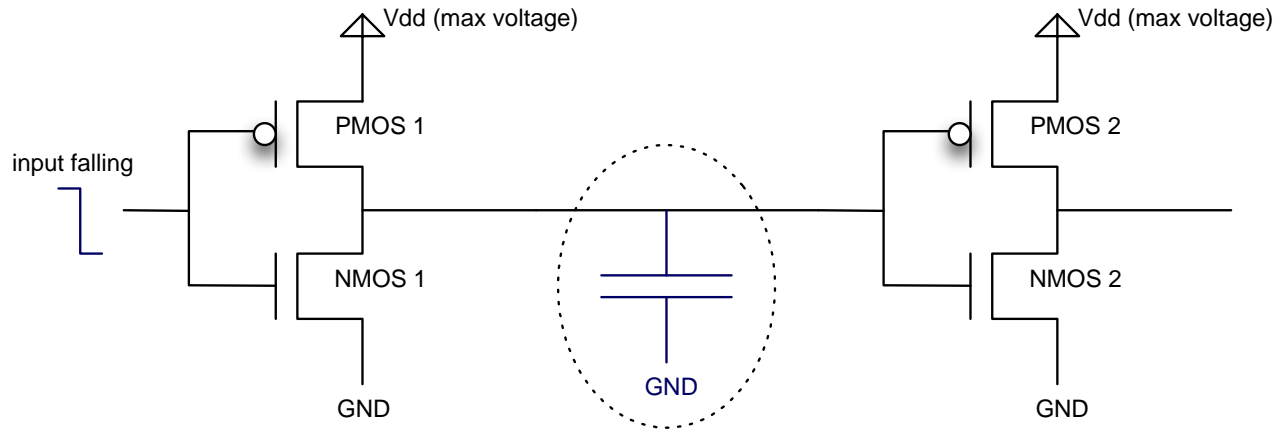


So what state are the transistors in when input is 0?

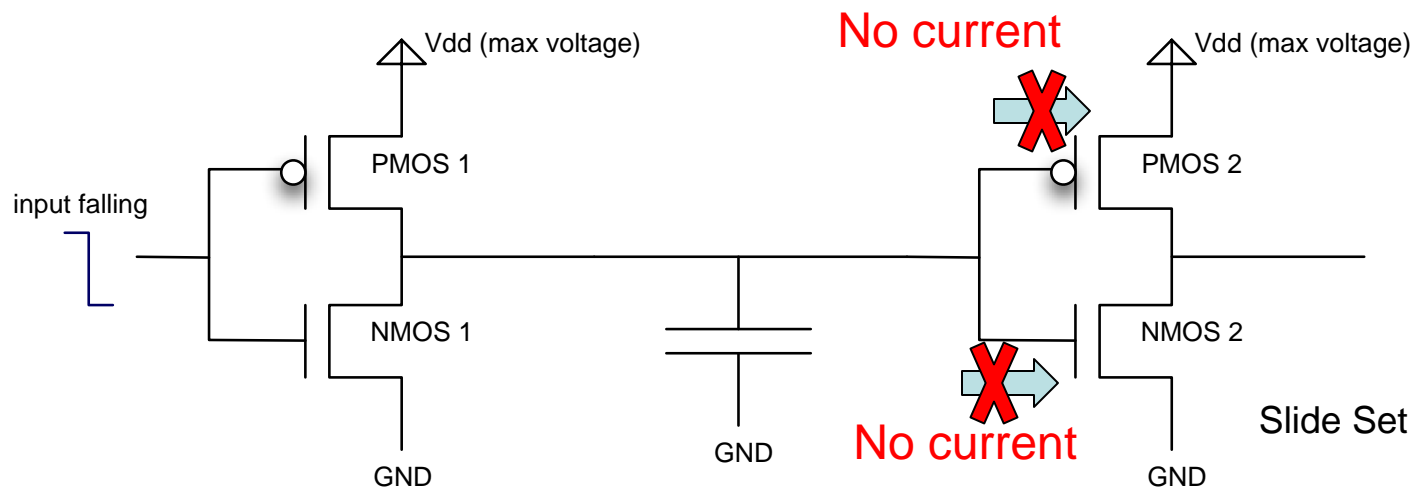


Two other facts you need to know:

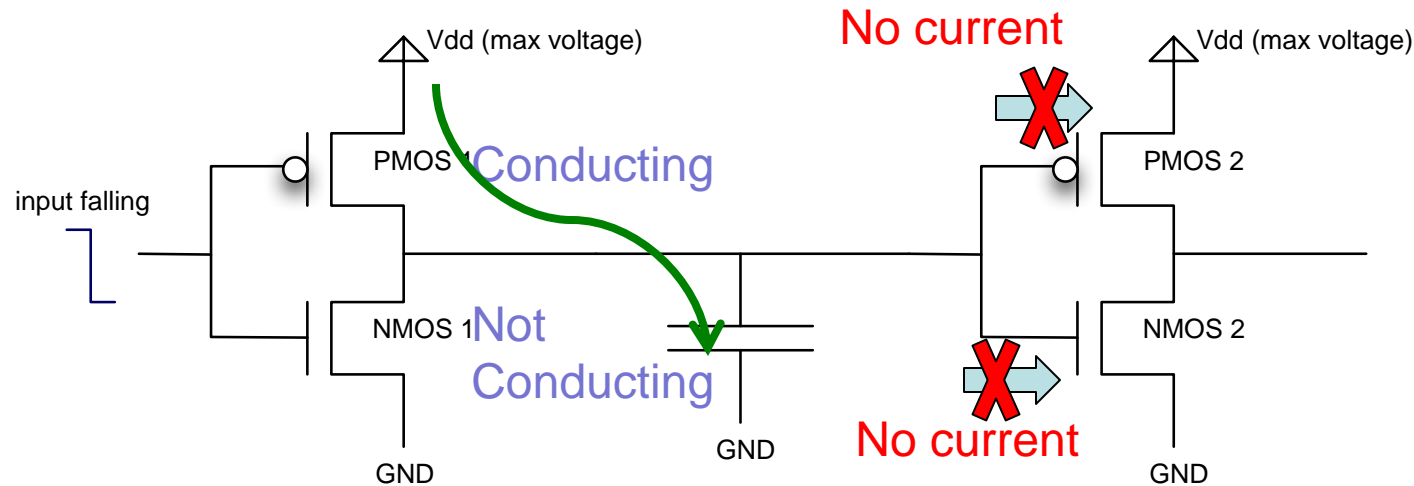
1. There is a “parasitic capacitance” attached to every wire and transistor
  - Due to the physical transistor structure



2. No current goes through the gate of a PMOS or NMOS Transistor



Given all that, what physically happens when input to first gate goes to 0?



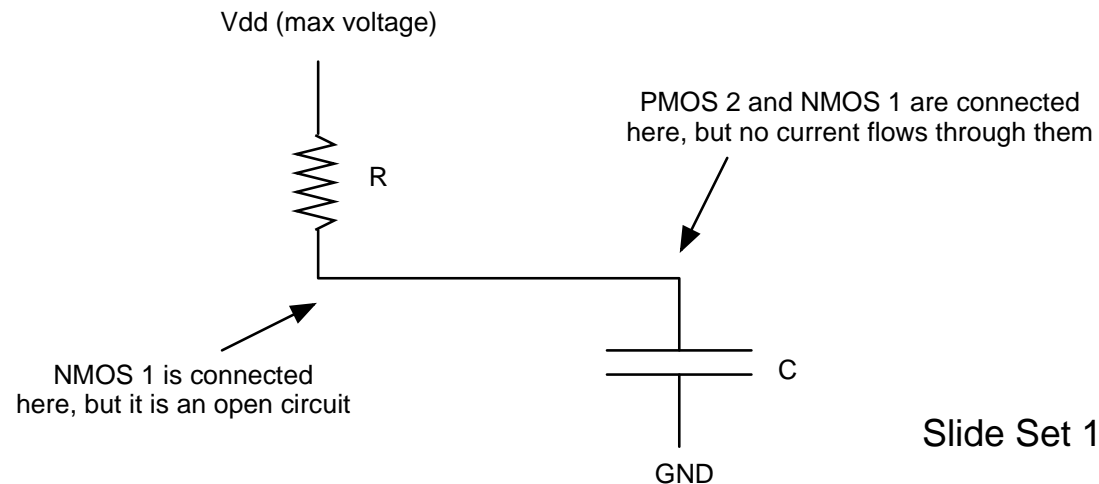
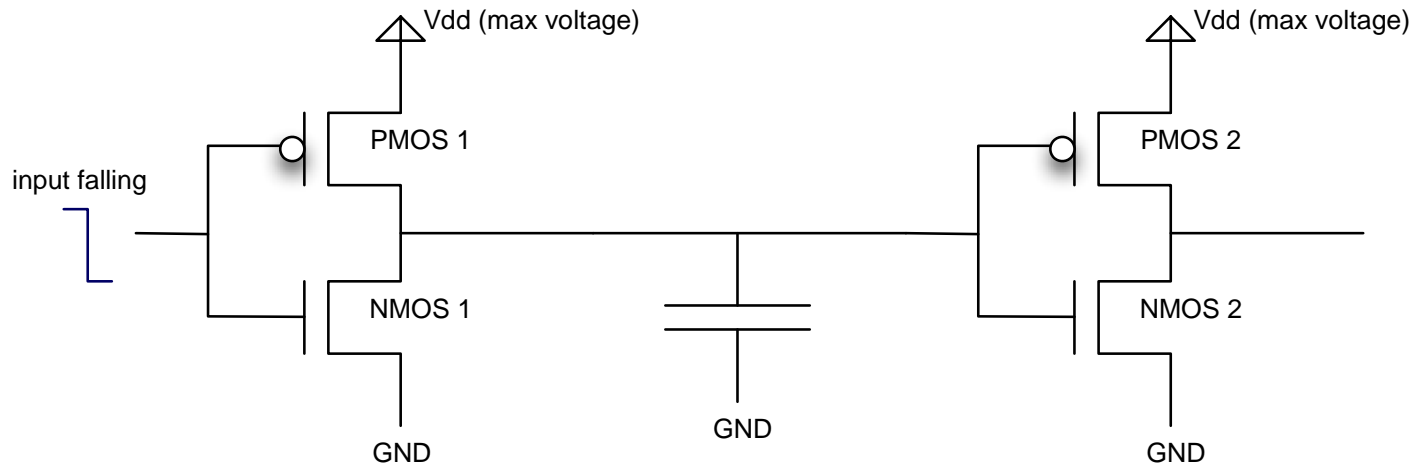
1. Initially the output of the first gate is 0.
2. When the input of the first gate falls, PMOS 1 becomes a resistance.
3. A current flows from Vdd, through PMOS 1, and charges the parasitic capacitance.
4. At some point, the parasitic capacitance reaches the threshold voltage of the second gate. At this point, the first gate is deemed to have “switched”, and the second gate starts switching.



**Observation:** So the “delay” of a logic gate 1 is the time for logic gate 1 to charge the capacitance on its output. Note: nothing to do with electrons “passing through” the logic gate. That does not happen.

# Transistor-Level Timing

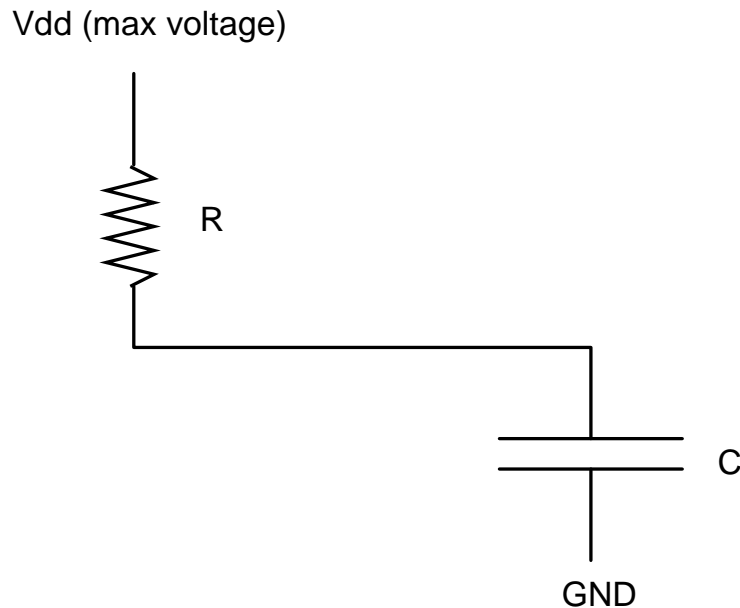
We can go farther and create an “equivalent circuit”:



# Transistor-Level Timing

---

How long does it take to charge the node?



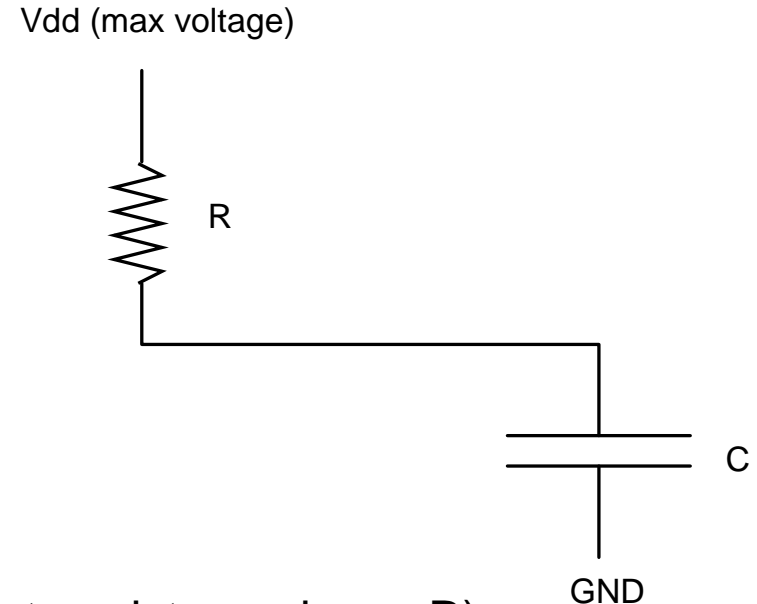
This is a second year problem. Initial condition output node = 0V, final condition is output node =  $V_{dd}/2$

Works out to be proportional to  $R \cdot C$

# Transistor-Level Timing

Delay of a gate is proportional to  $R \cdot C$ .

Delay of  $n$  gates is proportional to  $n \cdot R \cdot C$



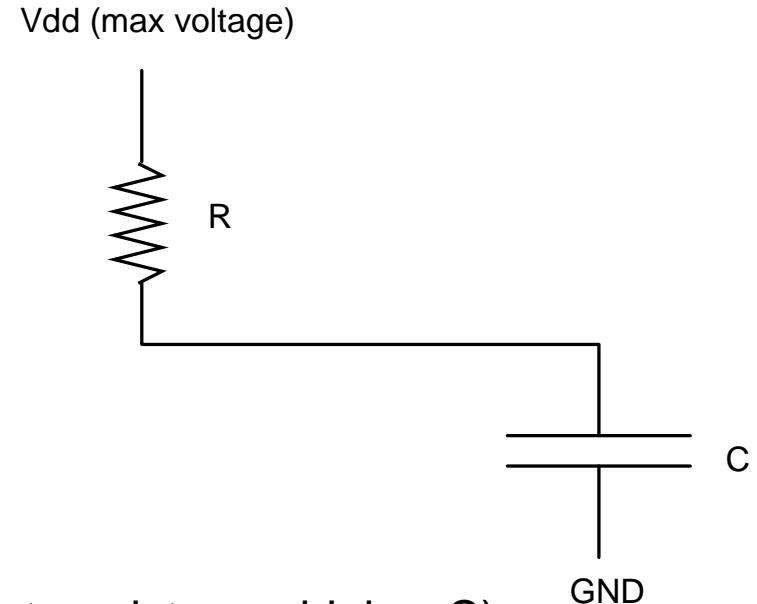
What does  $R$  depend on?

1. Size of transistors in the logic gate (bigger transistors = lower  $R$ )
  - The FPGA vendor has set this, as a designer you can not change them
2. Length of wire connecting the gates: long wires have non-negligible resistance (longer wire has more resistance)
  - This is why the placement tool tries to place source and destination close together

# Transistor-Level Timing

Delay of a gate is proportional to  $R \cdot C$ .

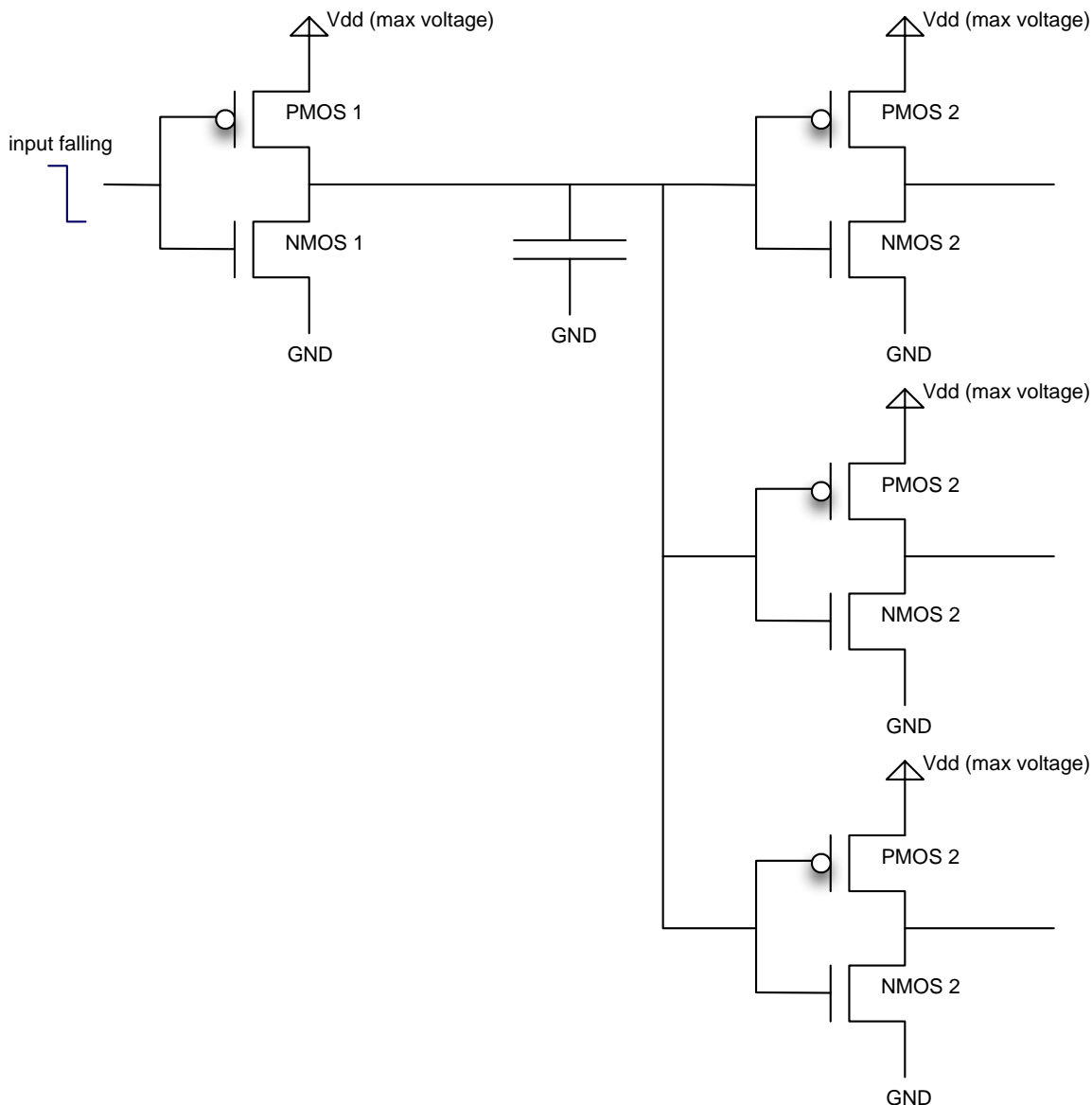
Delay of  $n$  gates is proportional to  $n \cdot R \cdot C$



What does  $C$  depend on?

1. Size of transistors in the logic gate (bigger transistors = higher  $C$ )
  - The FPGA vendor has set this, as a designer you can not change them
2. Number of gates driven (fanout): see next slide
3. Length of wire connecting the gates: long wires add capacitance -
  - Another reason the placement tool tries to place source and destination close together

# Fan-Out



The first gate drives three gates. Each destination gate provides some parasitic capacitance. Since these caps are in parallel, you add them.

Hence, capacitance seen by driver is  $\sim 3x$  in the first example

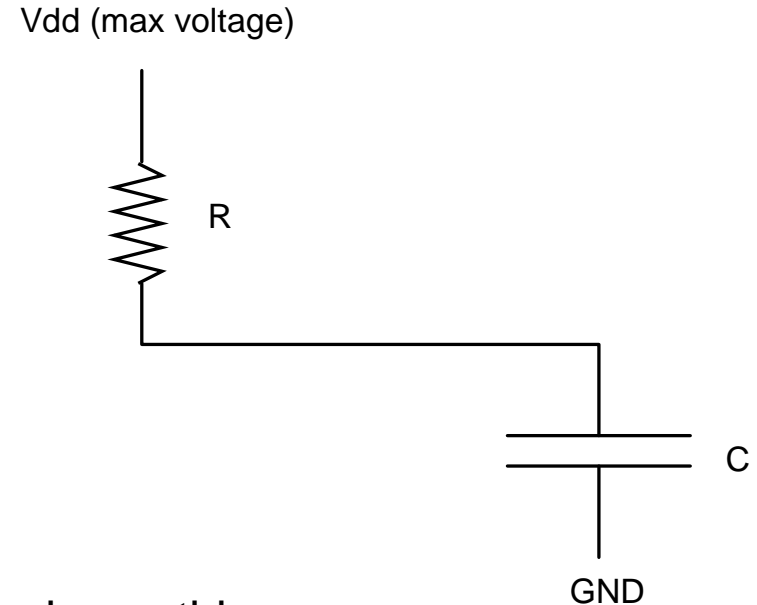
# Transistor-Level Timing

Delay of a gate is proportional to  $R \cdot C$ .

Delay of  $n$  gates is proportional to  $n \cdot R \cdot C$

What does  $n$  depend on?

Number of gates in the path. You have control over this.



Take-Away from the previous slides:

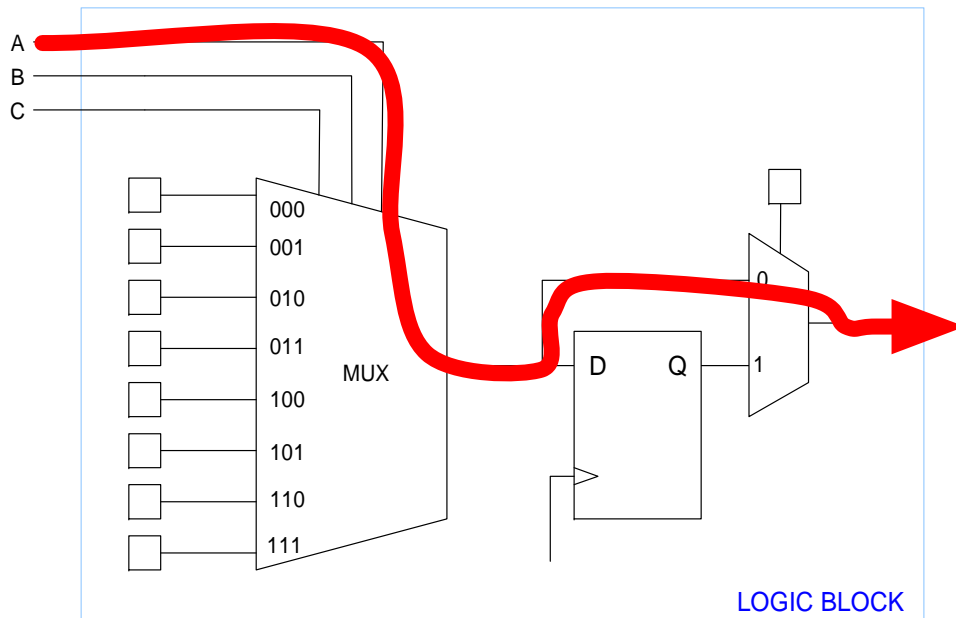
1. Delay of a logic gate depends on physical structure of gate as well as the length of the interconnect and how many gates are driven.
2. To find path delay, add delays of all gates in a path

To learn a lot more about this, take ELEC 402 next year



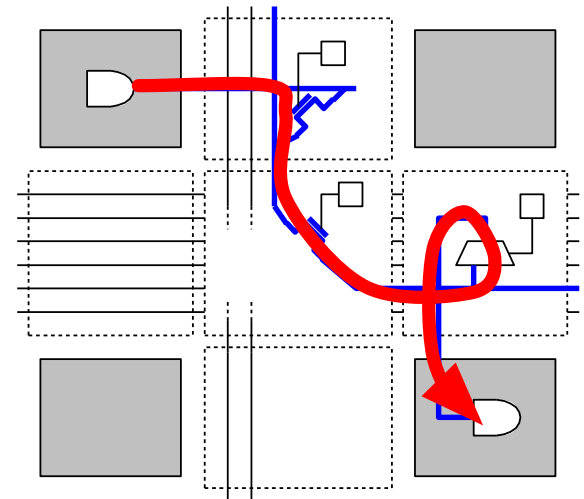
# Where does Delay Come From in an FPGA?

Logic Delay:



On the order of 0.1 ns

Routing Delay:



On the order of 1 ns

# Delay Modeling

Given an FPGA implementation of a circuit, you \*could\* calculate all the Rs and Cs and use 2<sup>nd</sup> year techniques to calculate delay

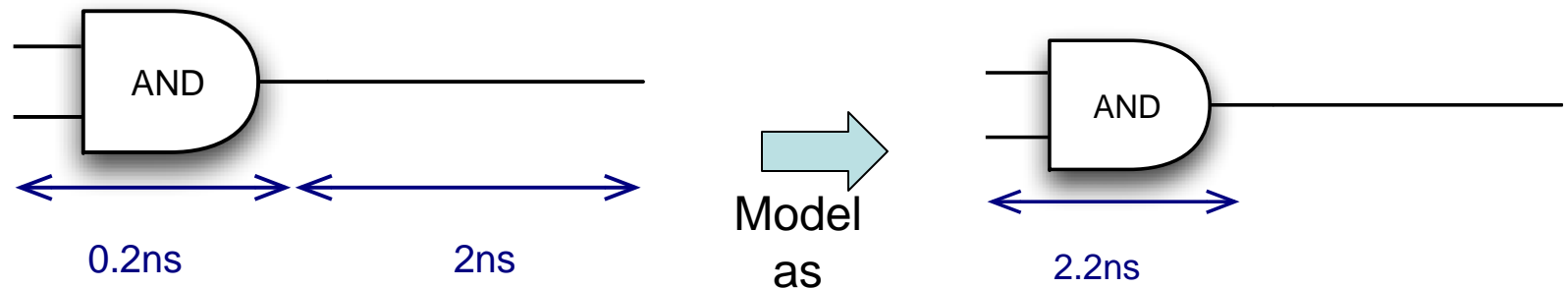
- Or you could use a simulation tool (SPICE) which operates at transistor level

This is not really feasible for a large circuit, so we need an abstraction.

- Pre-calculate delays of gates

In the previous slide, it should be clear that routing delay dominates.

For simplicity, for “back of the envelope timing calculations”, routing delays and gate delays are usually lumped together and we only speak of gate delays



# Timing Concepts and Terminology

---

## **Combinational Timing Constraints**

- Gate Propagation Delay
- Critical Path Delay

## **Flip-Flop Timing Constraints**

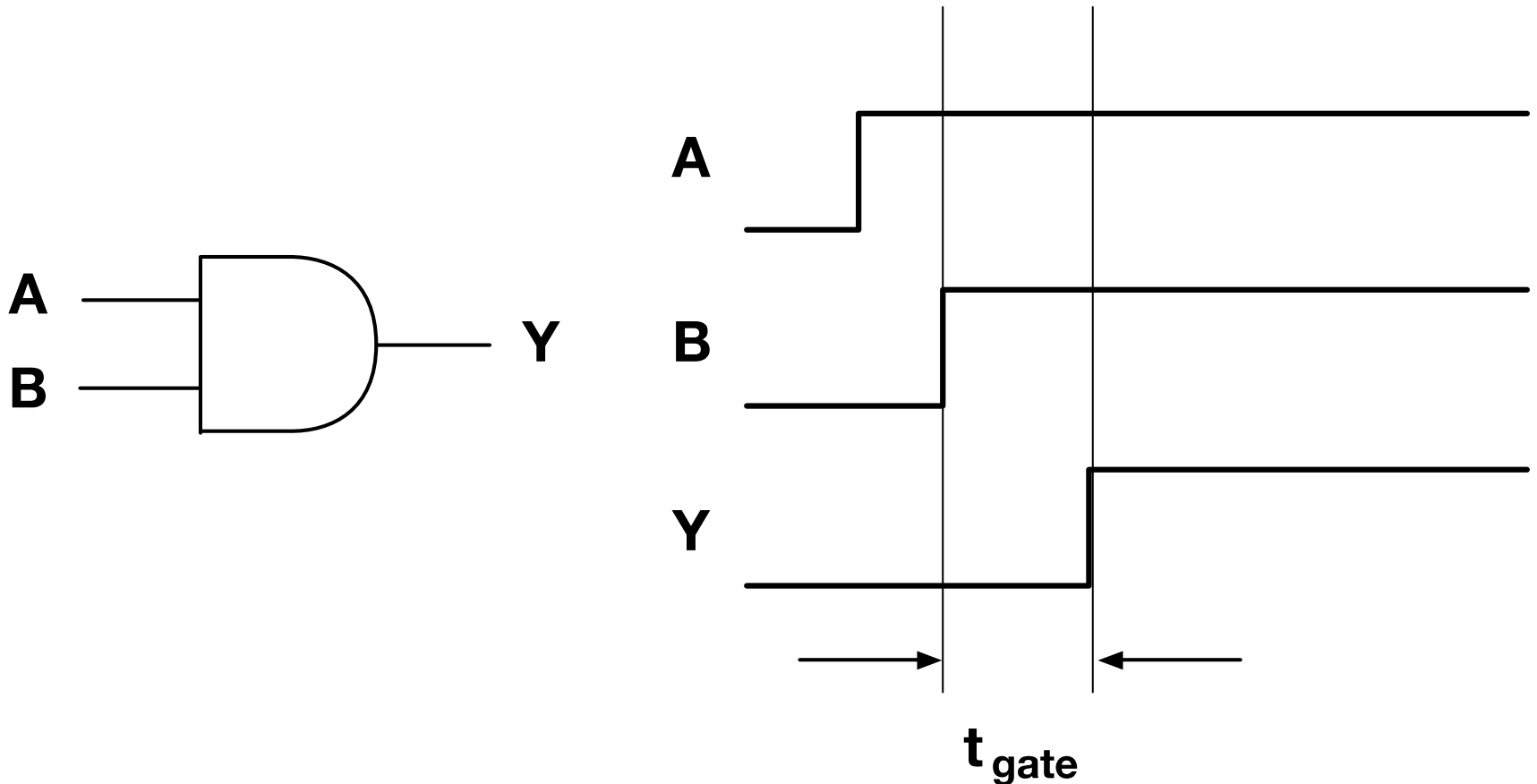
- Clock-to-Q Delay
- Setup Time
- Hold Time

## **Circuit Speed**

- Minimum Clock Period
- Maximum Clock Frequency

# Gate Delay (Propagation Delay)

Time that it takes for combinational gate output to change after inputs change



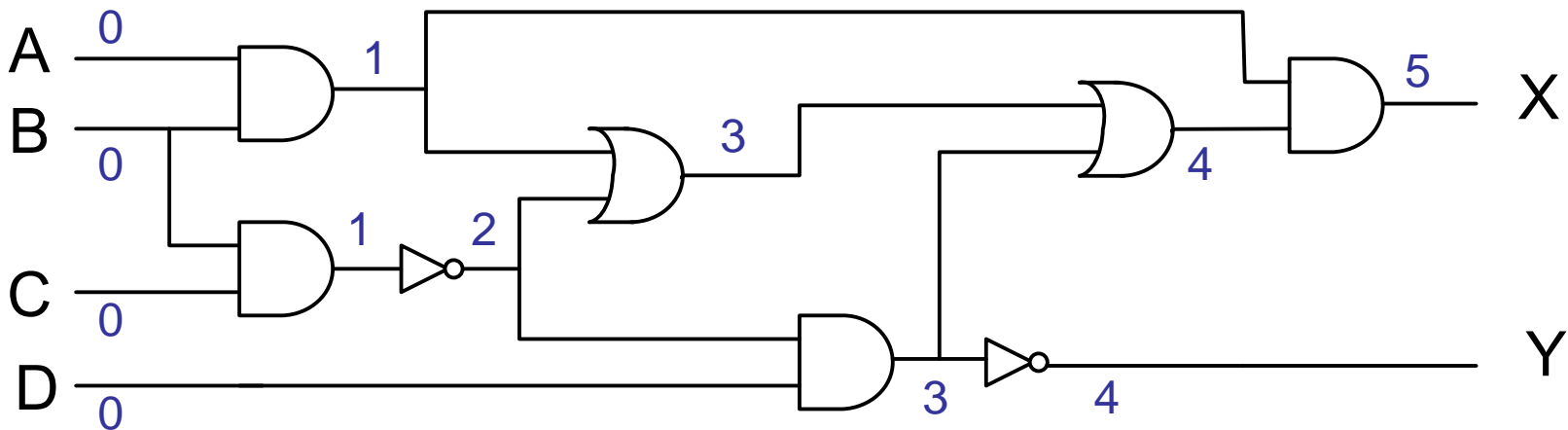
# Path Delay

Delay through a series of combinational gates.

Specifically, the time it takes for the output of the series of gates to change after the inputs to the path change.

## Example:

- Propagation delay for each gate is the same (1ns in this example)

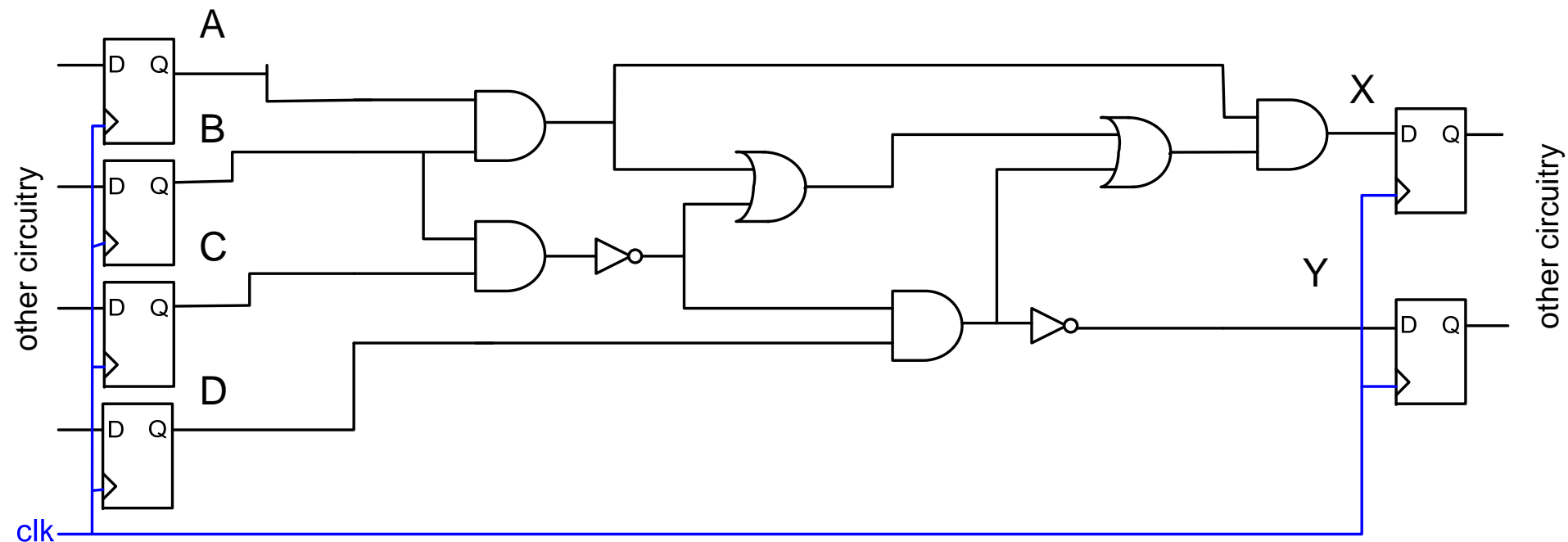


In other words, in the previous circuit, if we apply all inputs at time 0, then after 5ns, all the outputs have settled to their final values.

- Note:
1. Some outputs might settle earlier
  2. Some outputs may switch back and forth a few times before settling to a final value

# How to synchronize inputs?

Add flip-flops!



Rising edge on clock at time 0. Assuming no delay in the flip-flops, the outputs of the source (left four) flip-flops change at time 0.

Some time later (one clock cycle), the clock goes high again, and the destination flip-flops read in X and Y

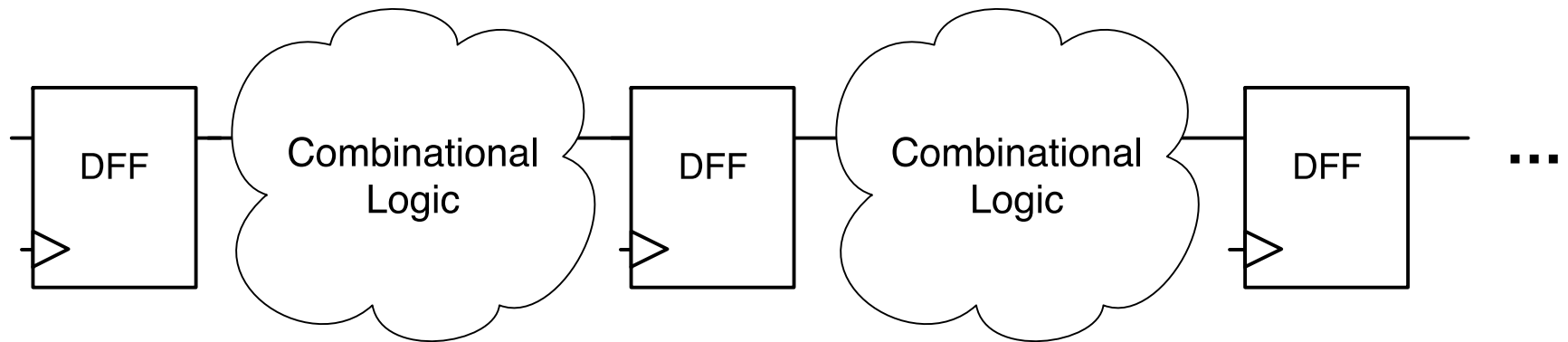
**How fast can we run the clock?**



# How fast can we run the clock?

---

**Clock's purpose is to tell flip-flops when to read in data**

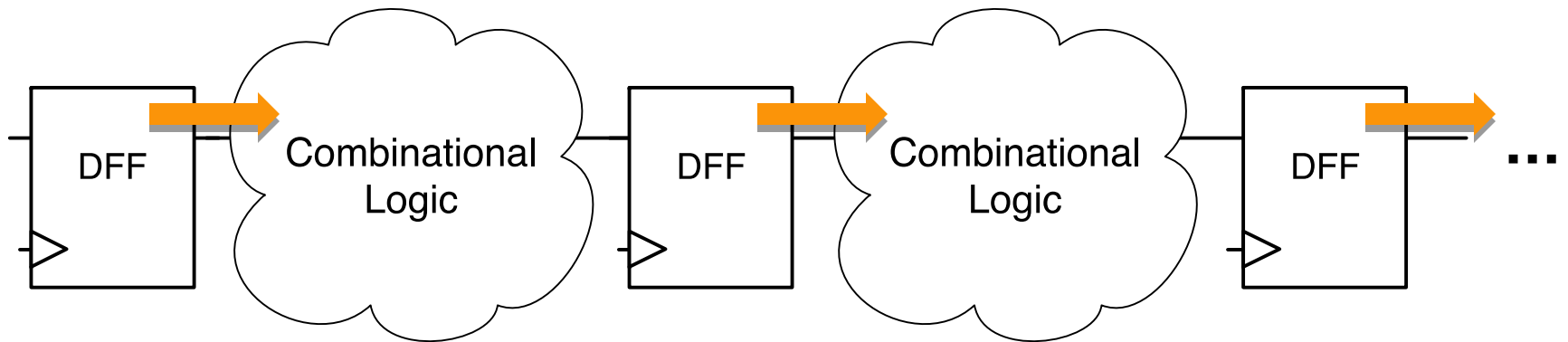


# How fast can we run the clock?

---

## Key Idea:

1. On clock edge, new inputs sent into combinational logic

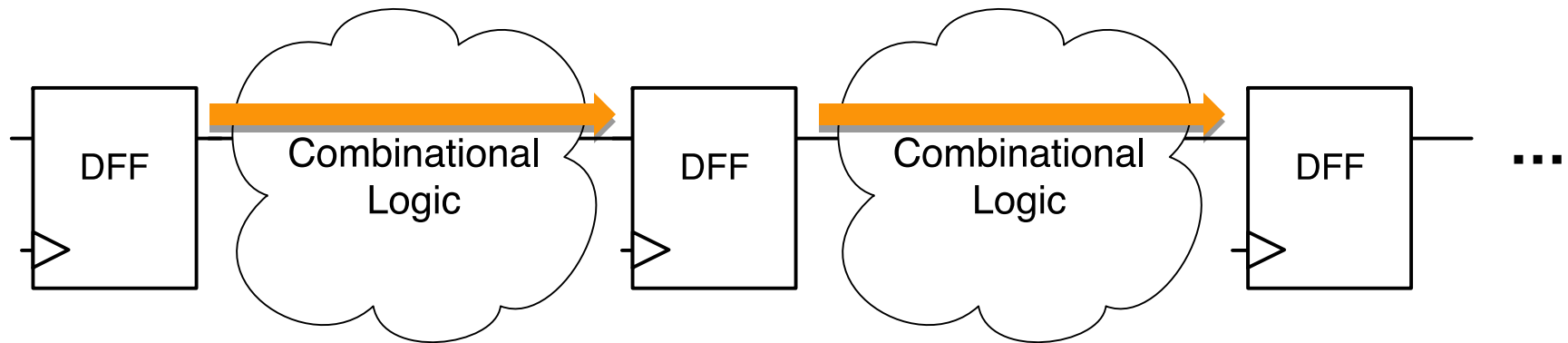


# How fast can we run the clock?

---

## Key Idea:

1. On clock edge, new inputs sent into combinational logic
2. Takes some time for outputs to settle

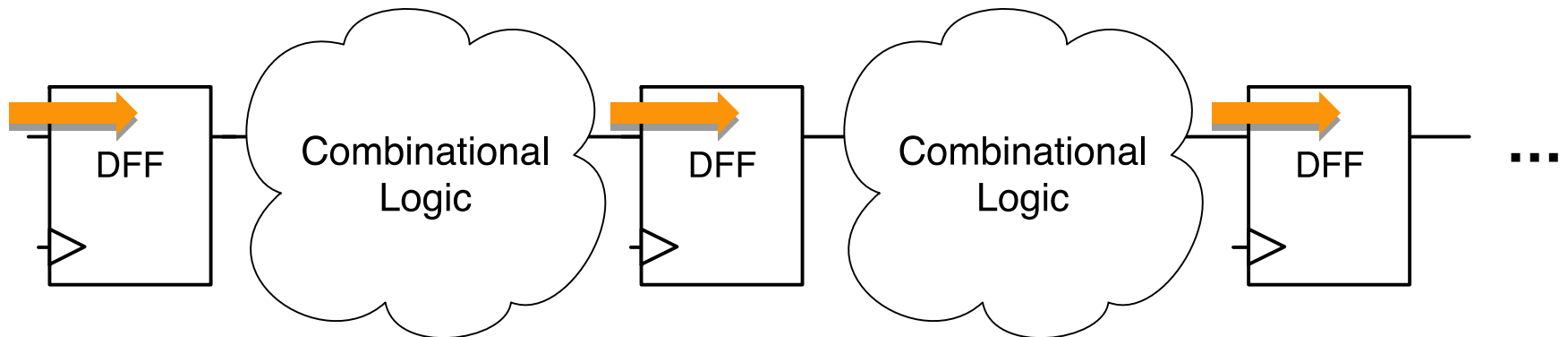


# How fast can we run the clock?

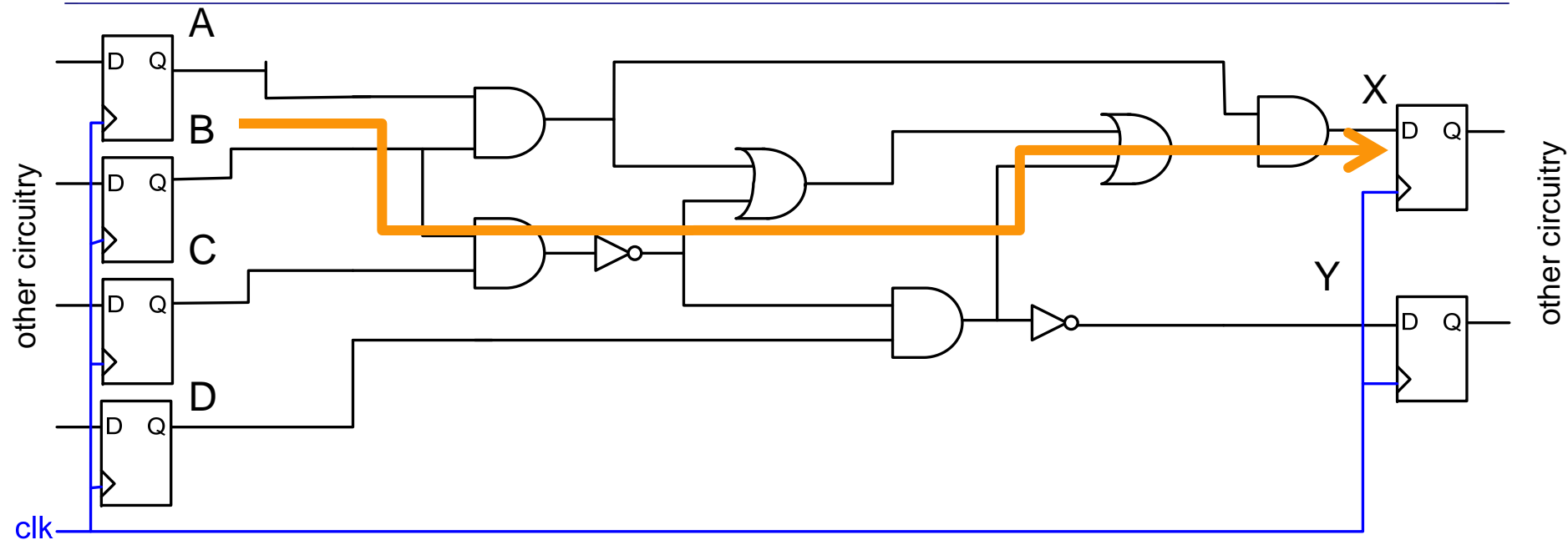
---

## Key Idea:

1. On clock edge, new inputs sent into combinational logic
2. Takes some time for outputs to settle
3. Don't want to read in new outputs before they are ready



# Example: Critical Path Delay



**Critical Path** is

$B \rightarrow \text{AND} \rightarrow \text{INV} \rightarrow \text{OR} \rightarrow \text{OR} \rightarrow \text{AND} \rightarrow X$

**Critical Path Delay** is 5ns

**Clock period cannot be smaller than 5ns or else X register will read in wrong data**

# Max Clock Frequency / Min Clock Period

---

Critical path delay limits the clock period  
(and hence clock frequency)

$$t_{ClockMin} \geq t_{CriticalPath}$$

From the example

$$t_{CriticalPath} = 5ns$$

$$\rightarrow t_{ClockMin} = 5ns$$

$$\rightarrow f_{ClockMax} = \frac{1}{5ns} = 200MHz$$

## What happens if you run the clock slower?

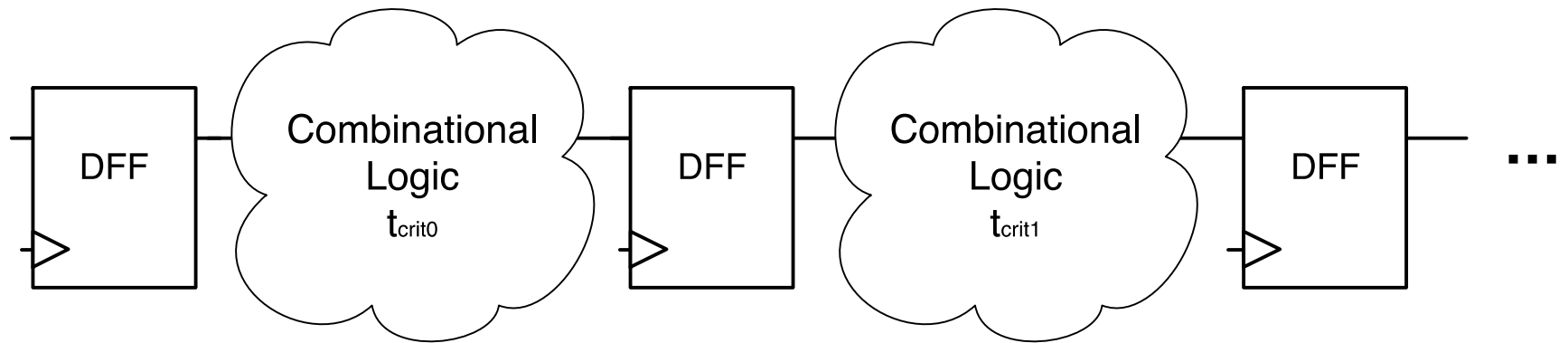
$$f_{Clock} < f_{ClockMax} \quad \Leftrightarrow \quad t_{Clock} > t_{CriticalPath}$$

No problem.

Combinational logic has more than enough time to settle

# Critical Path in Entire Design

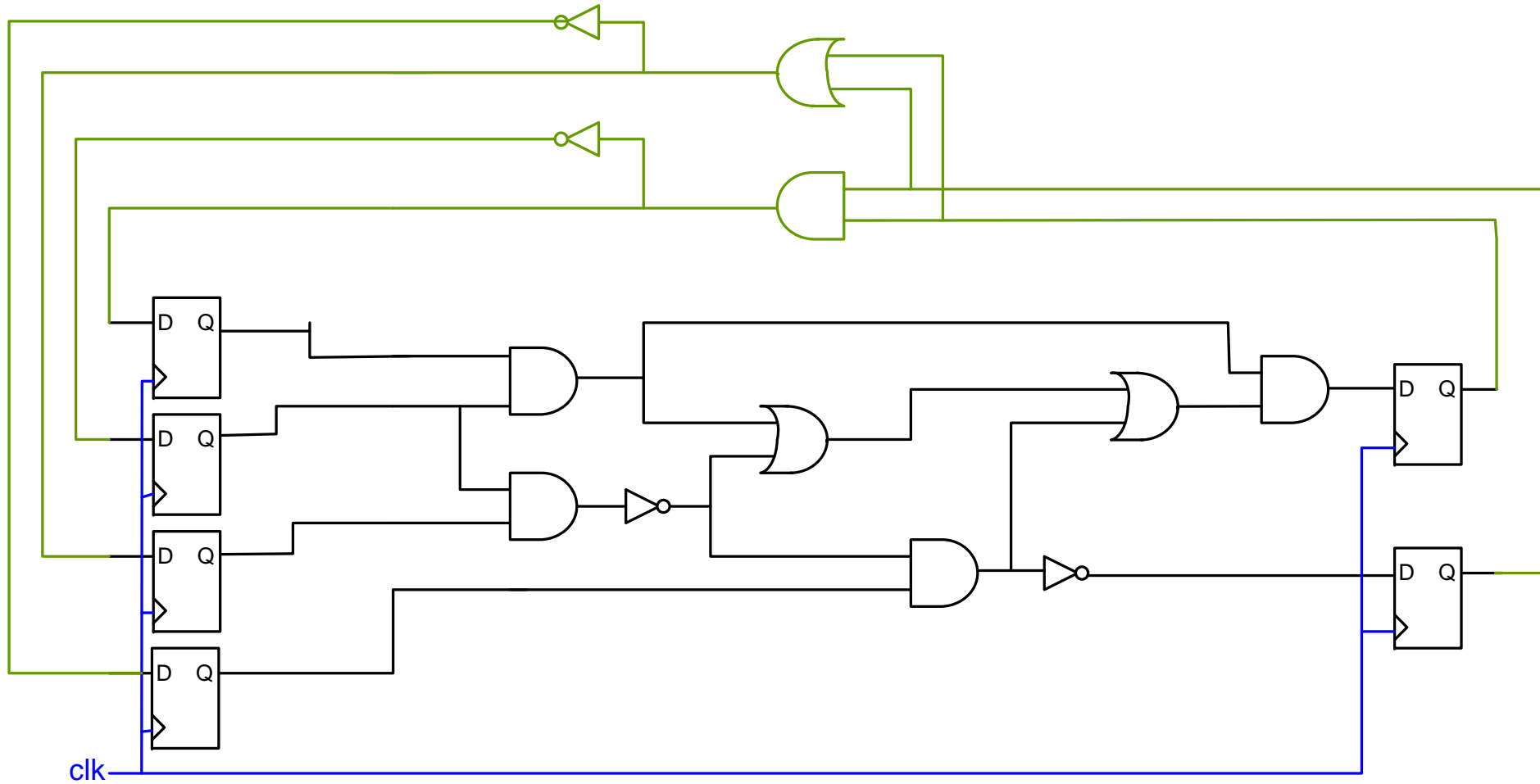
Since a **single clock** controls all flip-flops in all stages, you need to look for the critical path amongst **all stages**



$$t_{CriticalPath} = \max_{foreach\ path\ i} (t_{crit\_i})$$



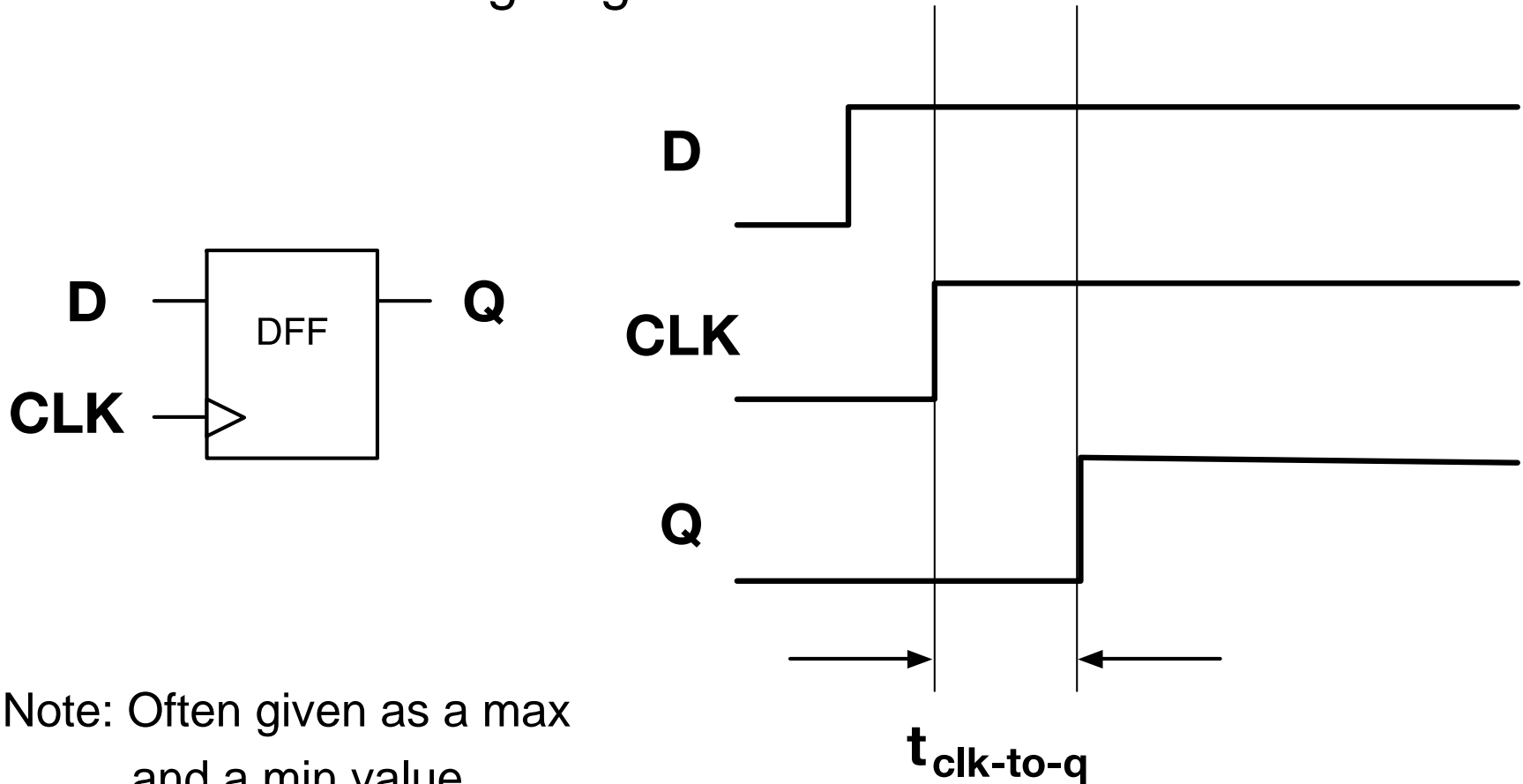
# For Example



# Flip-Flop Timing Constraints

# Clock-to-Q Delay

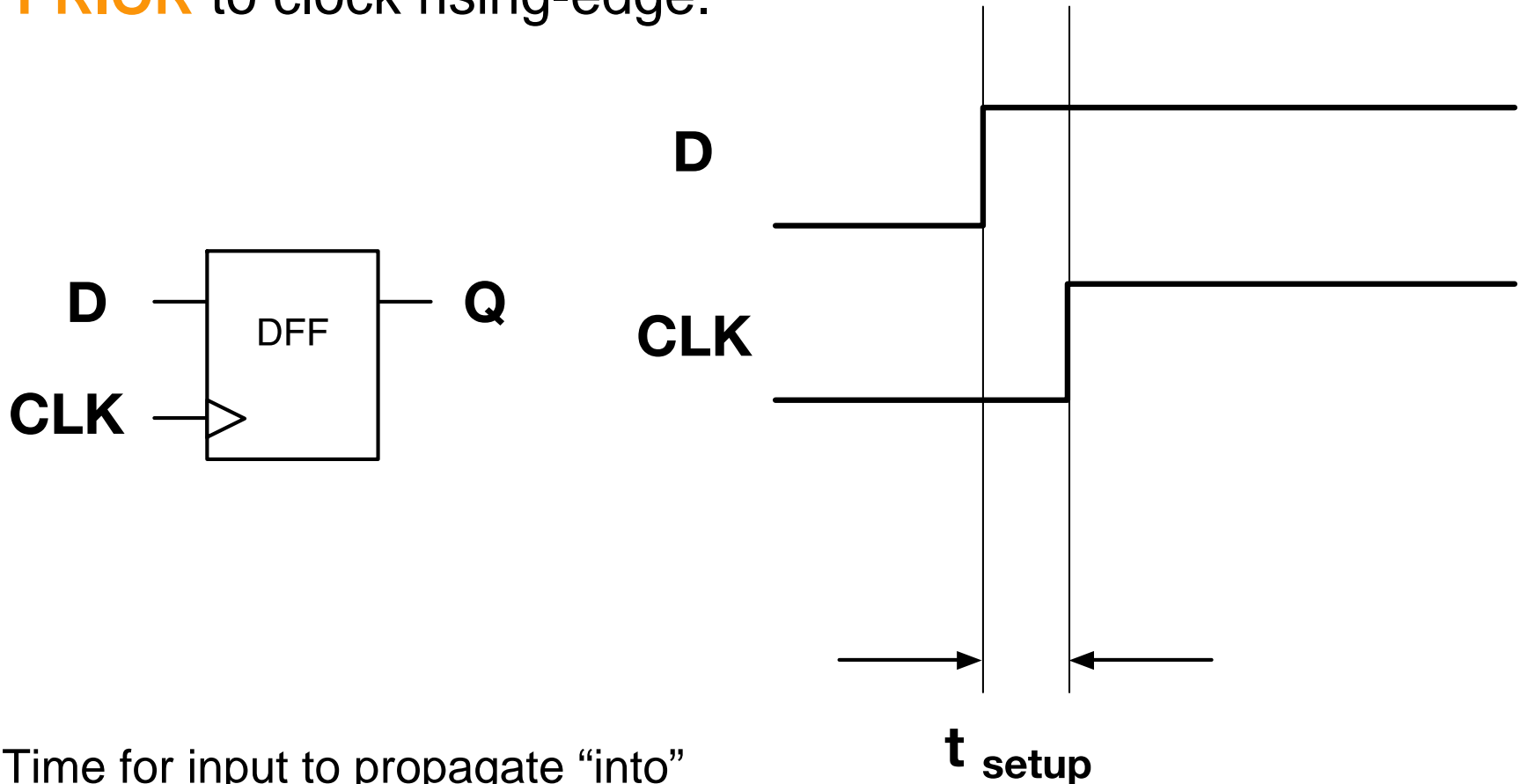
The time that it takes for the output of a Flip-Flop to settle after the clock rising-edge occurs



Note: Often given as a max and a min value

# Setup Time

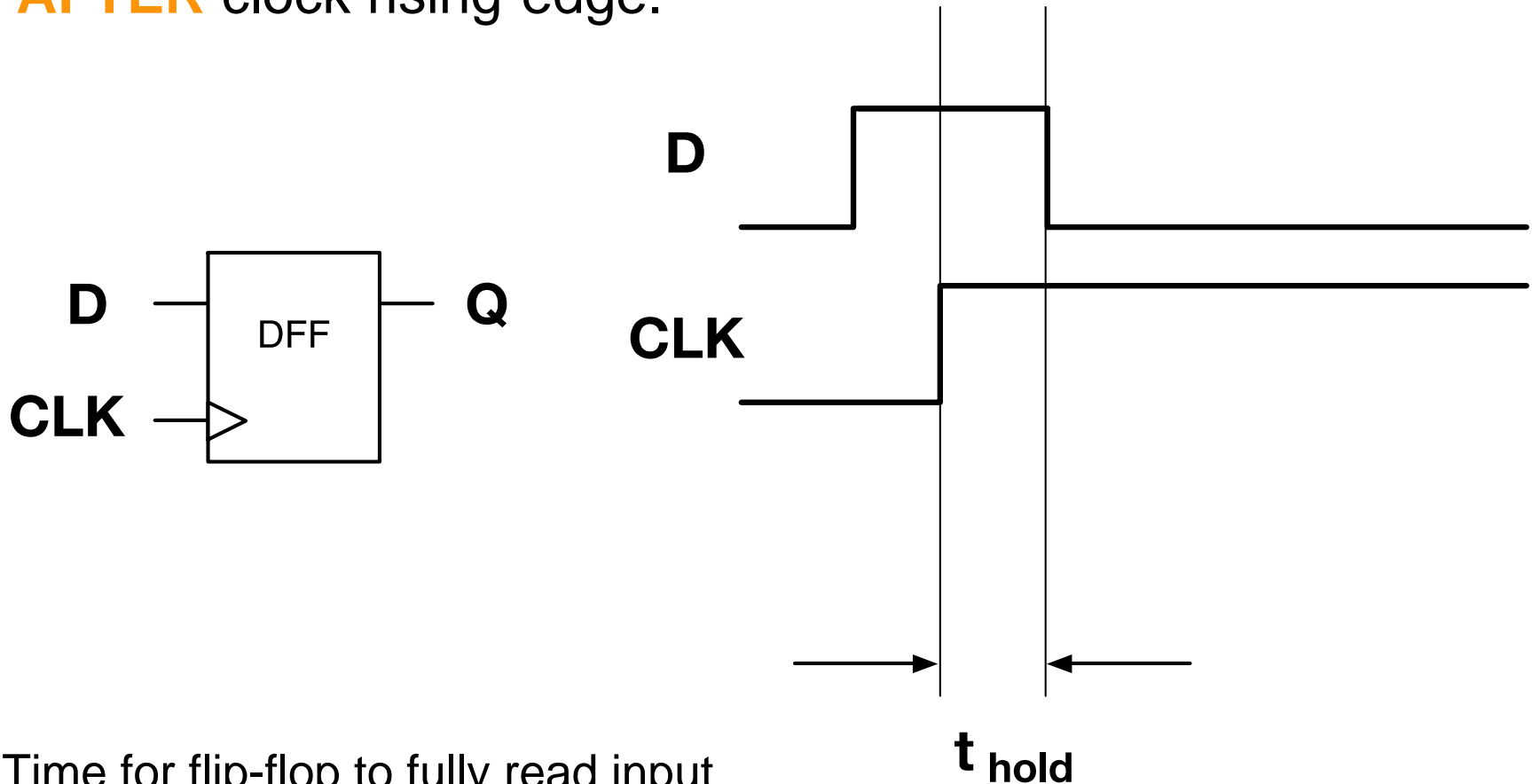
Input to flip-flop cannot change for a certain amount of time **PRIOR** to clock rising-edge.



Time for input to propagate “into”  
the flip-flop before clock arrives

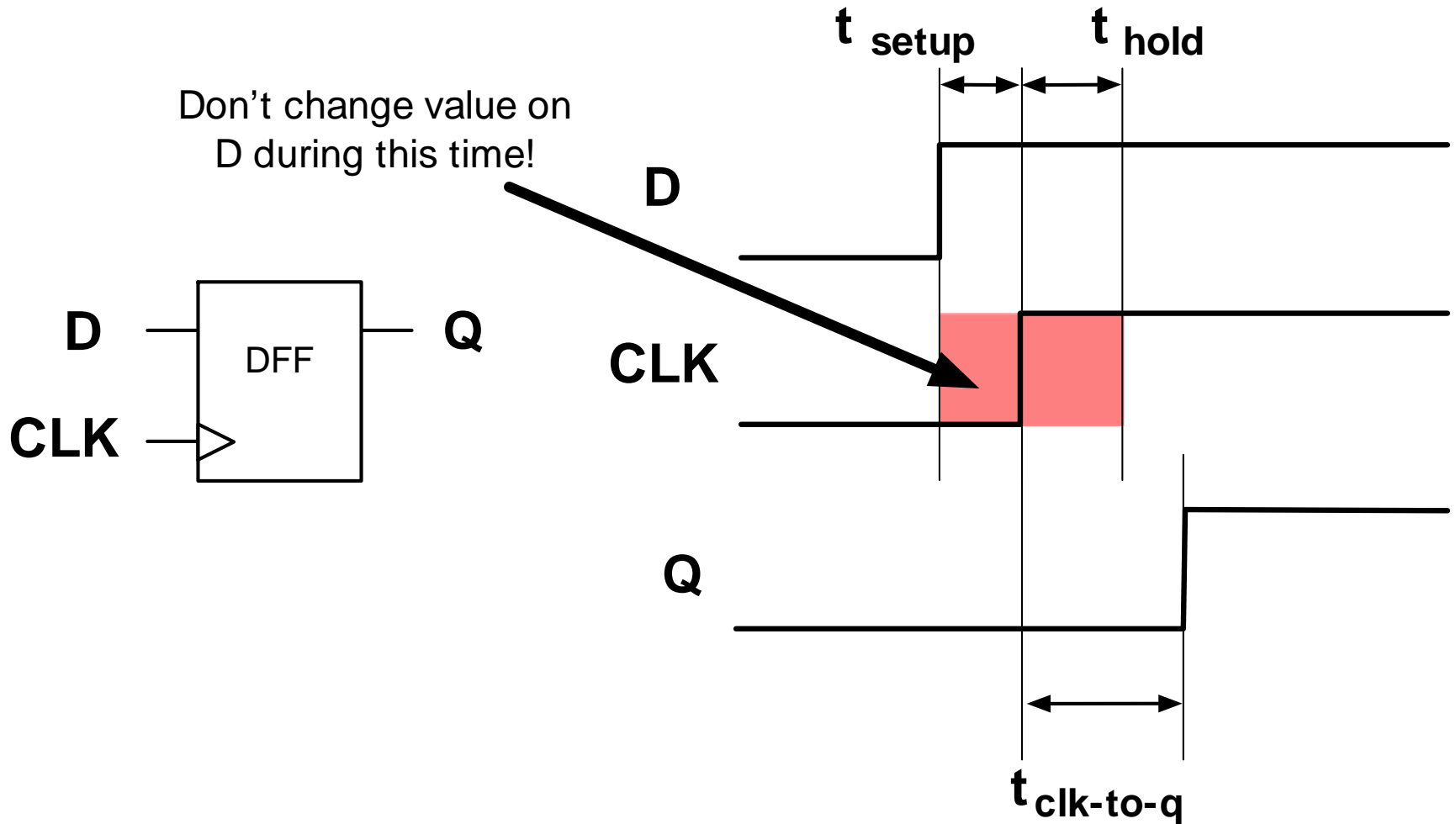
# Hold Time

Input to flip-flop cannot change for a certain amount of time **AFTER** clock rising-edge.



Time for flip-flop to fully read input  
before the value is "taken away"

# Summary of Flip-Flop Timing



# Two Requirements

---

## Setup Requirement

Input values must be ready for destination flop ahead of when the rising clock edge arrives.

## Hold Requirement

Input values of destination flop must not change for a short time after rising clock edge.

# How fast can we run the clock?

Revisited to account for flip-flop constraints

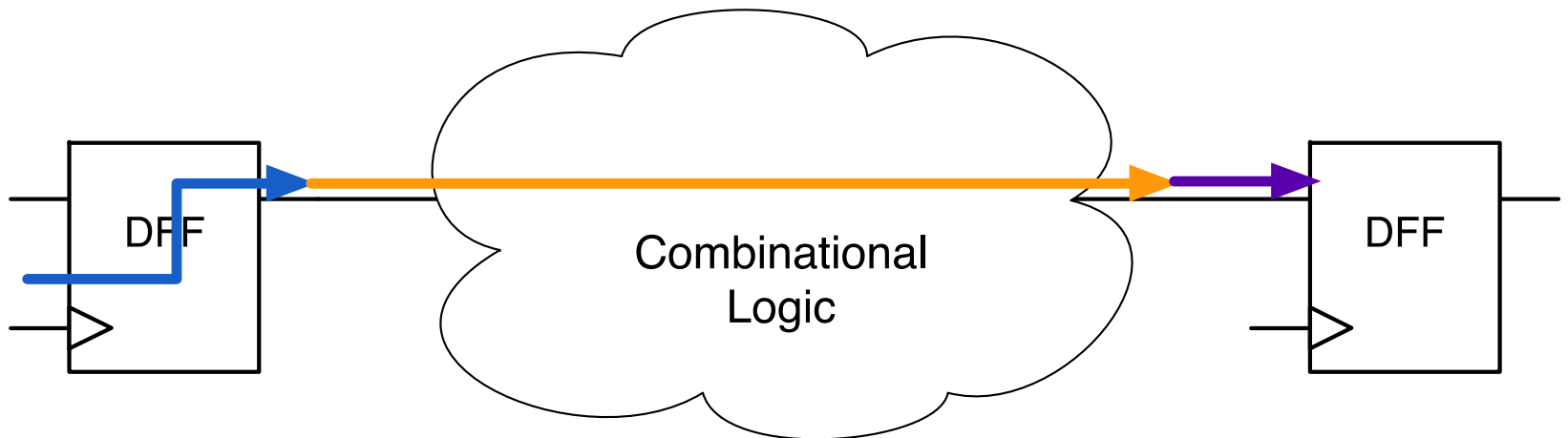


# Setup Requirement

$$t_{Clock} \geq t_{clk-to-q\max} + t_{CriticalPath} + t_{setup}$$

After clock rising-edge, there must be enough time so that

1. source flip-flop's output can settle to correct value
2. critical path can settle to correct value
3. value arrives at destination flip-flop early enough for setup time



# **Key Concept for Setup Time:**

**Must have enough time for  
input of FF to settle  
before FF reads in value**

# Hold Requirement

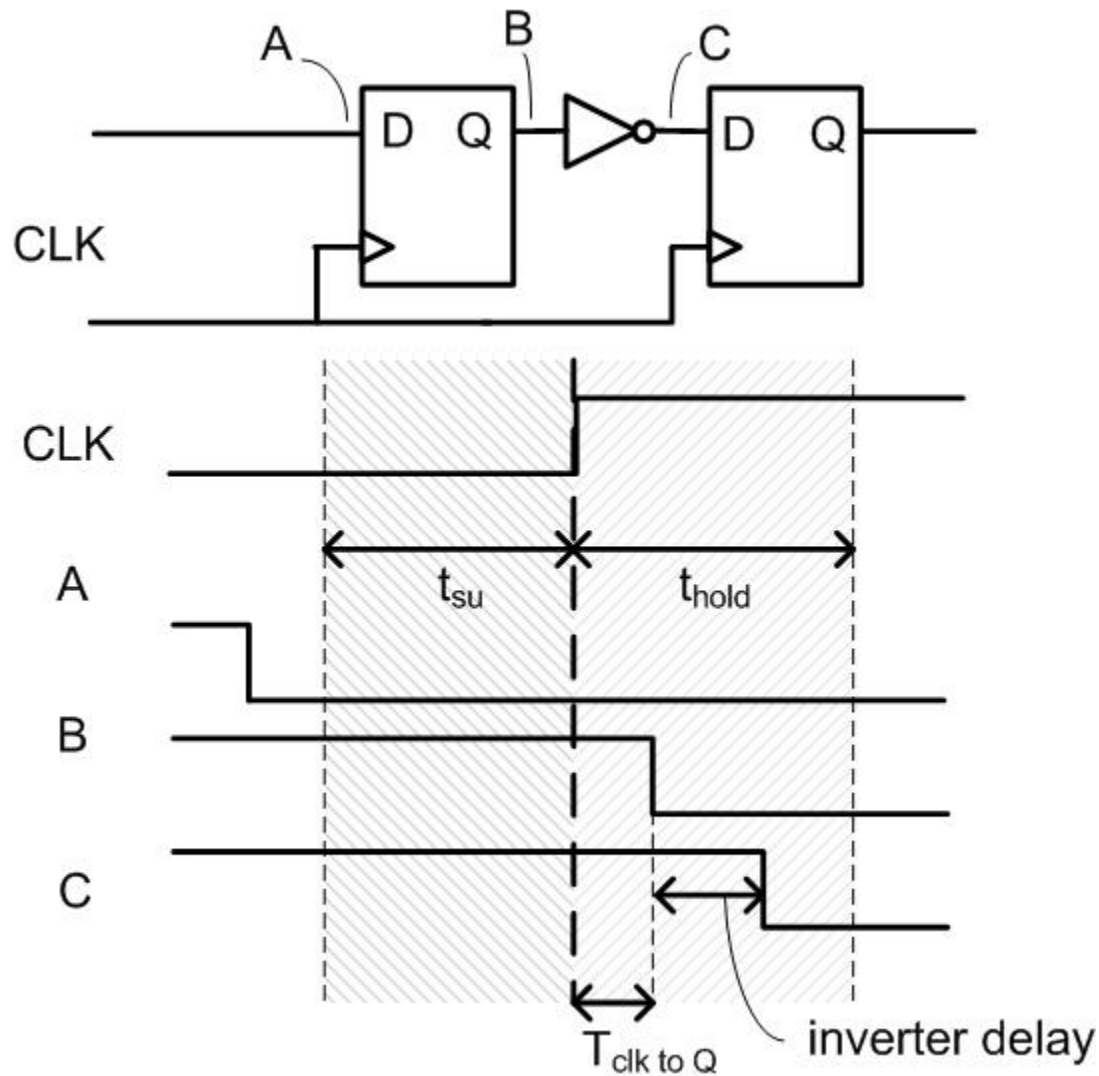
---

Remember, after rising edge of clock, FF is now reading in data.

Input to the FF cannot change for  $t_{\text{hold}}$  amount of time

- Unlikely to be a problem for Critical Path
- Need to be careful for the **fast (non-critical) paths**

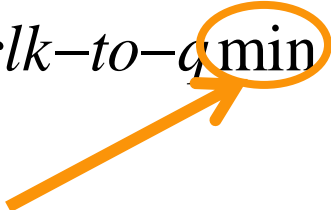
Suppose the hold time is really big...



# Hold Requirement

---

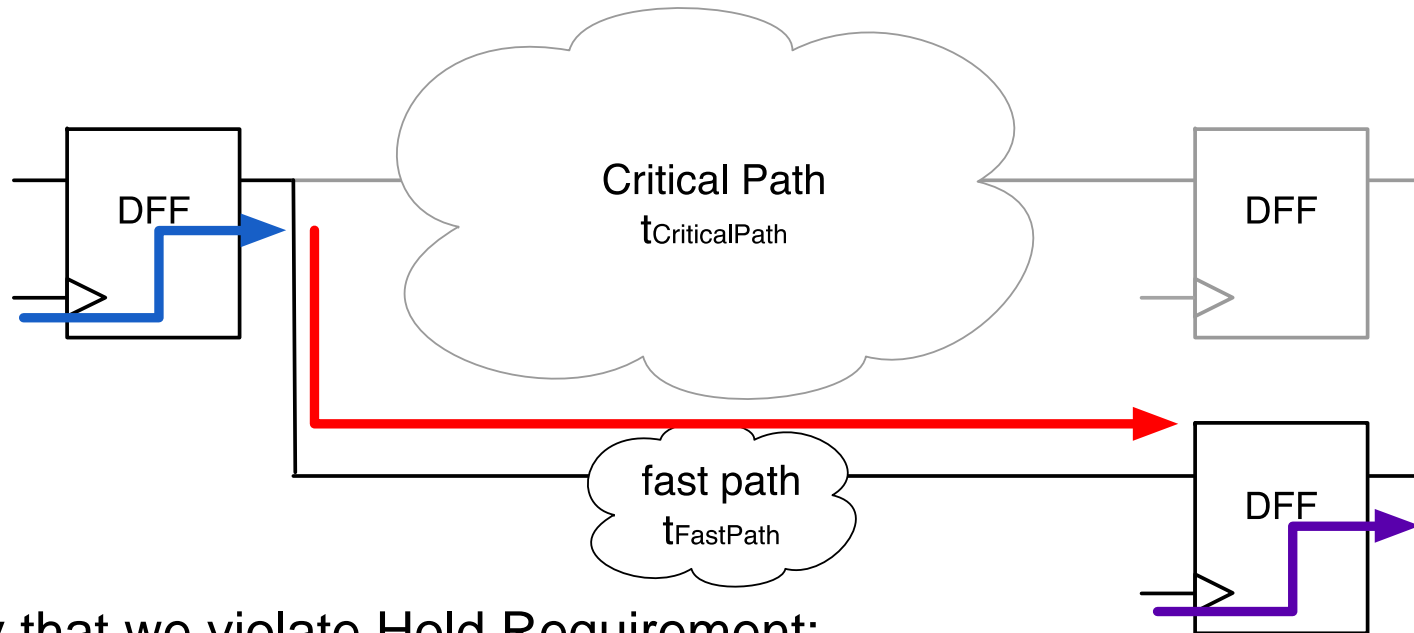
In general, for all paths:

$$t_{clk-to-q\min} + t_{path} \geq t_{hold}$$


Minimum clk-to-q delay because we need to design for the fastest possible arrival time of signal to destination flop

Note: On each clock cycle, flip-flop is BOTH reading in a new value (when you look at it as a destination flip-flop) AND providing a value to the next stage (when you look at it as a source flip-flop)

# Hold Requirement Violation Example



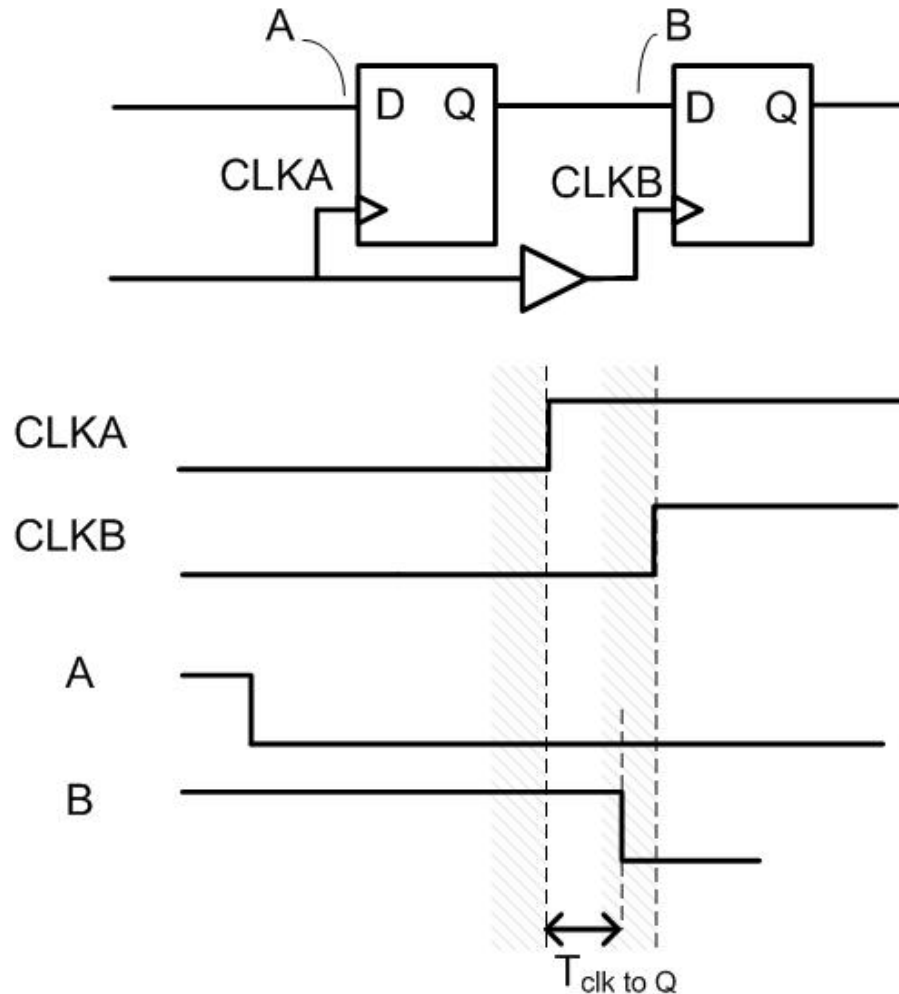
Say that we violate Hold Requirement:

$$t_{clk-to-q\min} + t_{FastPath} < t_{hold}$$

After clock rising-edge,

1. source flip-flop's output settles
2. fast path settles **SUPER QUICKLY** to correct value
3. destination flop is still reading in “old” value from fast path logic (i.e. hold time has not yet passed for that reading)

Often a problem if the clock is delayed (due to long wires, for example)



We will come back to clock skew later

# How To Fix Violations

---

## Setup Violation

- Slow down clock
- Move registers around (reduce length of critical path)

## Hold Violation

- **CANNOT** address by changing clock frequency!
- Can add more gate-delays to path (e.g. add buffers)

**GOOD NEWS! If you are using an FPGA, the CAD tools handle all this for you!**



# Learning Objectives

---

1. Understand the RC model for gate delay and where it comes from
2. Be able to find the critical path, and hence the maximum clock speed, of a logic circuit (with or without flip-flops)
3. To understand set-up time, clk-q time, and hold time of a flip-flop
4. Understand setup and hold timing requirements of a design
5. Given a circuit, be able to calculate the maximum clock frequency
6. Given a circuit, be able to calculate the minimum hold time on the flip-flops

# CPEN 311: Digital Systems Design

## Slide Set 13: Circuit Timing: Part 2: Practical Issues

2016/2017 Term 1

Instructor: Steve Wilton

[steve.w@ece.ubc.ca](mailto:steve.w@ece.ubc.ca)

# Learning Objectives

---

1. Understand what timing closure is and why it is difficult
2. Understand how pipelining can help with timing closure
3. Understand retiming and be able to apply it to a circuit
4. Be able to discuss the effects of clock skew
5. Understand what a PLL is used for
6. Understand the cause and impact of glitches caused by unequal combinational path delays

# Timing Closure

---

**Timing Closure:** Ensuring your design meets timing constraints

So far you have been compiling and just hoping it runs at the required speed  
(eg. If you are using CLOCK\_50, the critical path  $\leq 20$  ns)

That was fine for these simple labs, but for complex designs, a simple compile may not lead to you meet your timing constraints.

The screenshot displays the Quartus II software interface during a compilation process. The main window shows the 'Compilation Report' for the 'Slow 1200mV 85C Model Fmax Summary'. A red arrow points from the 'Fmax Summary' entry in the 'Table of Contents' to the summary table.

**Table of Contents:**

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- TimeQuest Timing Analyzer
  - Summary
  - Clocks
  - Slow 1200mV 85C Model
    - Fmax Summary

**Slow 1200mV 85C Model Fmax Summary**

	Fmax	Restricted Fmax	Clock Name	Note
1	180.38 MHz	180.38 MHz	clk	

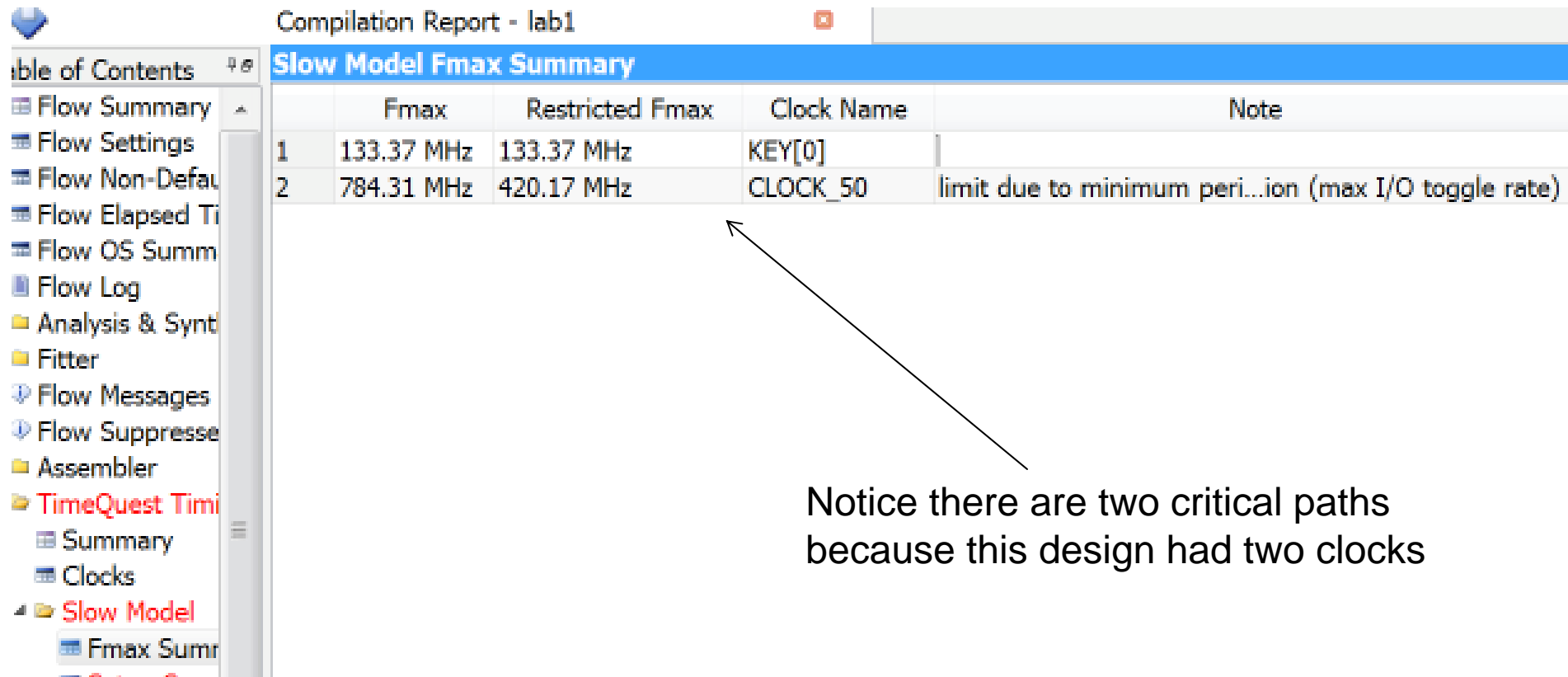
This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack reports for

**Messages:**

- Info (204019): Generated file lab3\_6\_1200mv\_0c\_vhd\_slow.sdo in folder "/Desktop/temp2/simulation/modelsim/" f
- Info (204019): Generated file lab3\_min\_1200mv\_0c\_vhd\_fast.sdo in folder "/Desktop/temp2/simulation/modelsim/"
- Info (204019): Generated file lab3\_vhd.sdo in folder "/Desktop/temp2/simulation/modelsim/" for EDA simulation
- Info: Quartus II 32-bit EDA Netlist Writer was successful. 0 errors, 0 warnings
- Info (293026): Skipped module PowerPlay Power Analyzer due to the assignment FLOW\_ENABLE\_POWER\_ANALYZER
- Info (293000): Quartus II Full Compilation was successful. 0 errors, 44 warnings

Based on register-register delays

From my implementation of Lab 1:



Slow Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	133.37 MHz	133.37 MHz	KEY[0]	
2	784.31 MHz	420.17 MHz	CLOCK_50	limit due to minimum period (max I/O toggle rate)

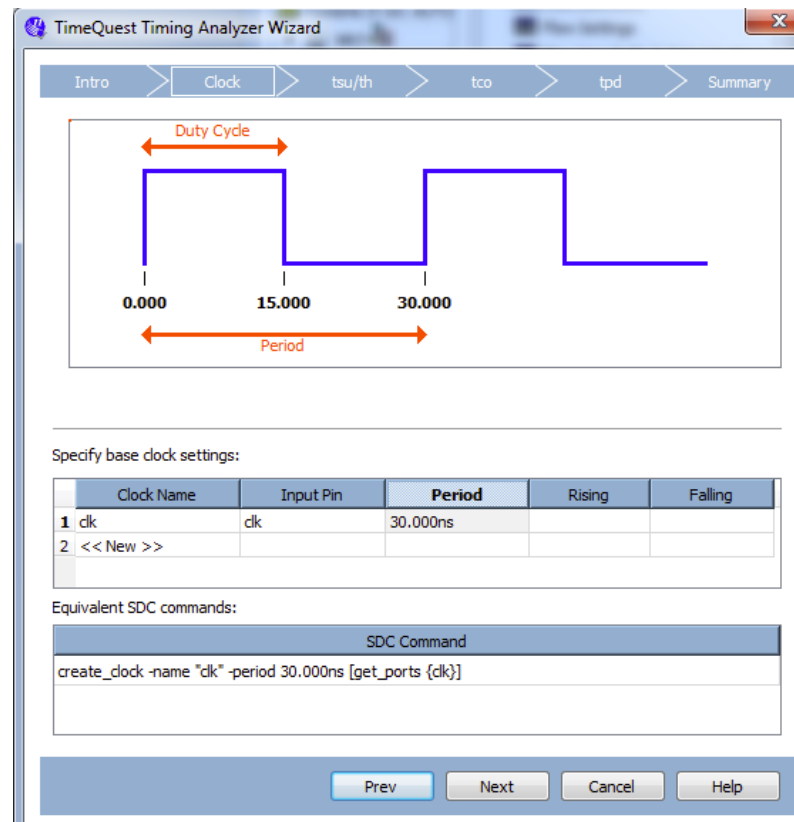
Notice there are two critical paths  
because this design had two clocks

*Fmax* means the maximum frequency the circuit can run at

# Timing Constraints in Quartus II

You can specify a desired target clock frequency  
aka **Timing Constraint**

Assignments Menu → Timing Analysis Wizard



# Quartus Timing Optimization

---

- Tries to optimize your design to meet your desired clock frequency
- May not necessarily meet your target
  - Limited by the delays of the various logic and routing resources inside of the FPGA
- If you don't set a timing constraint, tool sets an unachievable value (e.g. 1GHz)

Effects on optimization when you ask for

## **Slow clock frequency**

Leads to smaller circuits. More weight on optimizing for logic utilization

## **Fast clock frequency**

Leads to larger circuit – more things happening in parallel

**All about tradeoffs!**



# Tradeoffs: Lab 1

---

Experiments conducted on a DE2 board:

Scenario	Achieved Fmax	Area of Circuit
No explicit timing constraints (1 GHz)	133 MHz	148 Logic Elements
Timing constraint of 130 MHz	130 MHz	143 Logic Elements
Timing constraint of 10 MHz	112 MHz	135 Logic Elements

# Can Go Faster Than Predicted $F_{max}$ ?

---

## Overclocking

- Quartus gives a conservative estimate
- Actual delays within the chip vary from chip-to-chip

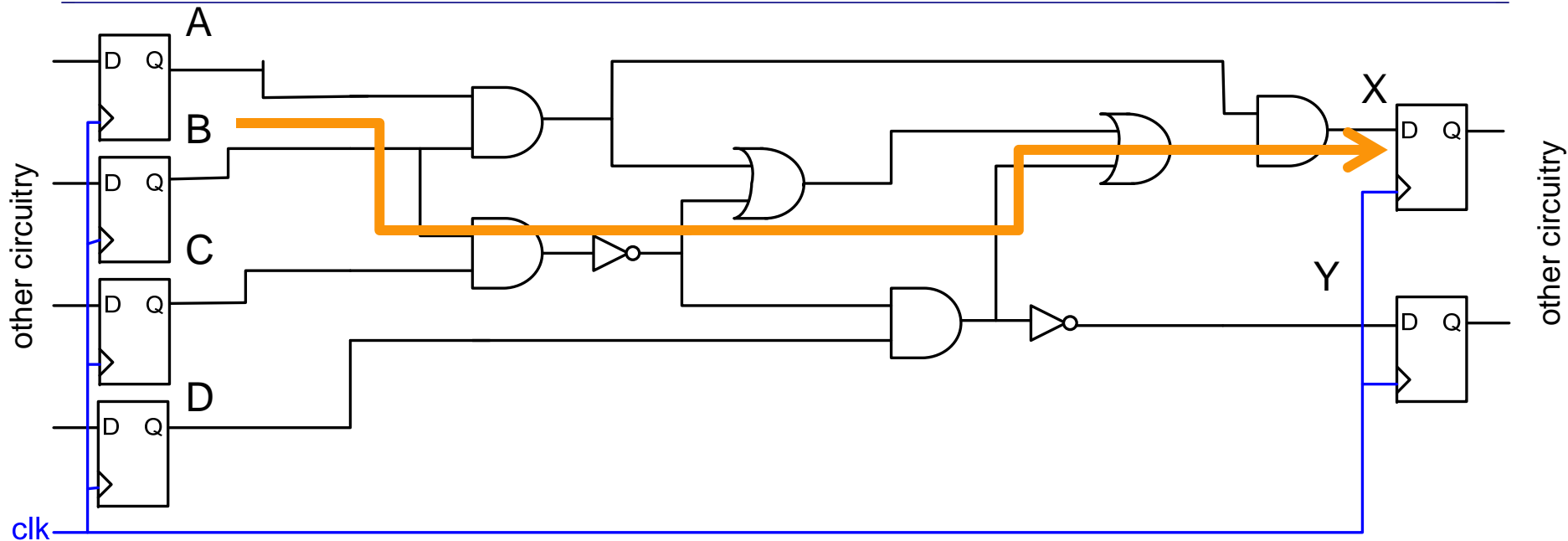
## Dangerous... don't do it

- In your testing, it might appear to work. But, how do you know that you are exercising the critical path?
- Some chips will be slower than the one you use for prototyping
- Operating conditions such as temperature affect actual performance



# PIPELINING

# From earlier: Critical Path Delay

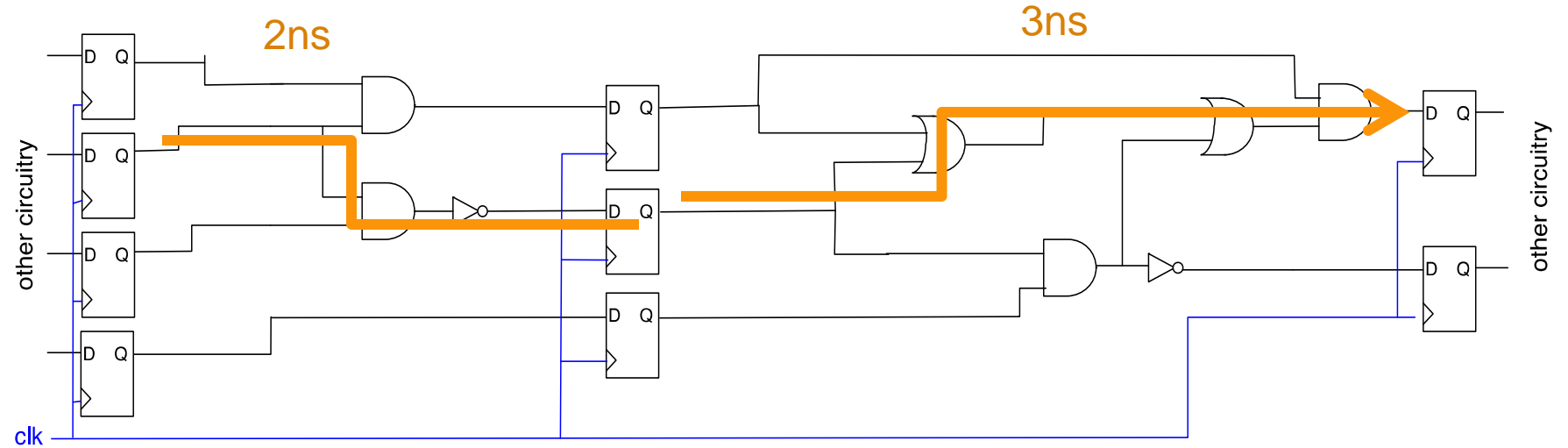
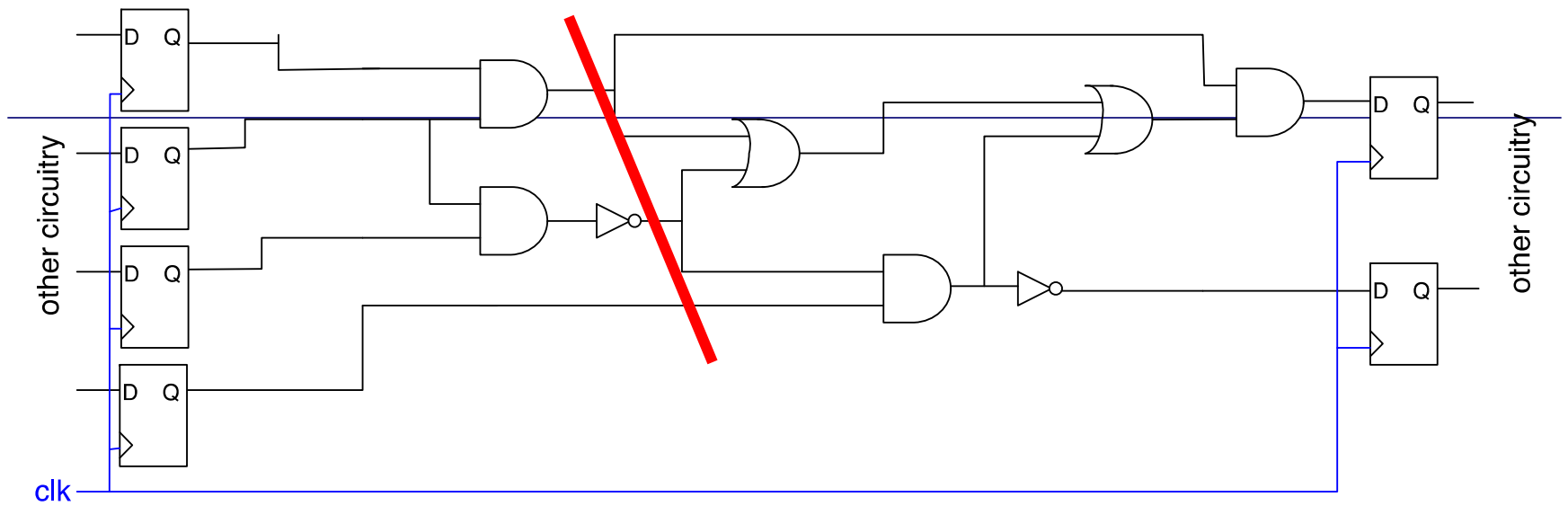


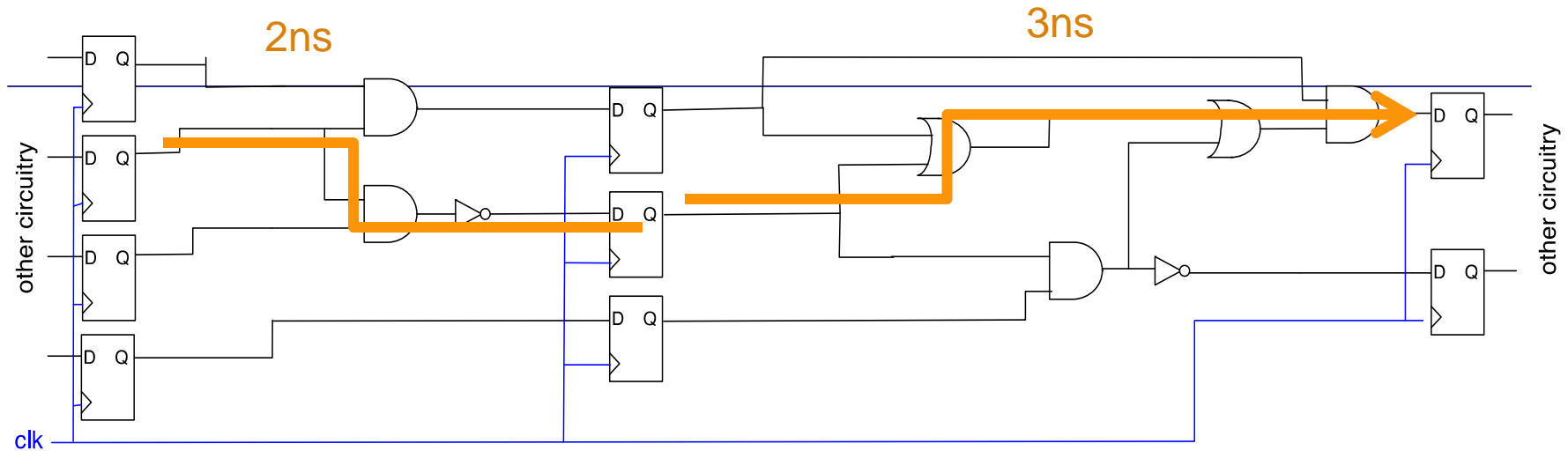
**Critical Path** is

$B \rightarrow \text{AND} \rightarrow \text{INV} \rightarrow \text{OR} \rightarrow \text{OR} \rightarrow \text{AND} \rightarrow X$

**Critical Path Delay** is 5ns

**Clock period cannot be smaller than 5ns or else X register will read in wrong data**





The new longest path between registers is 3 ns.

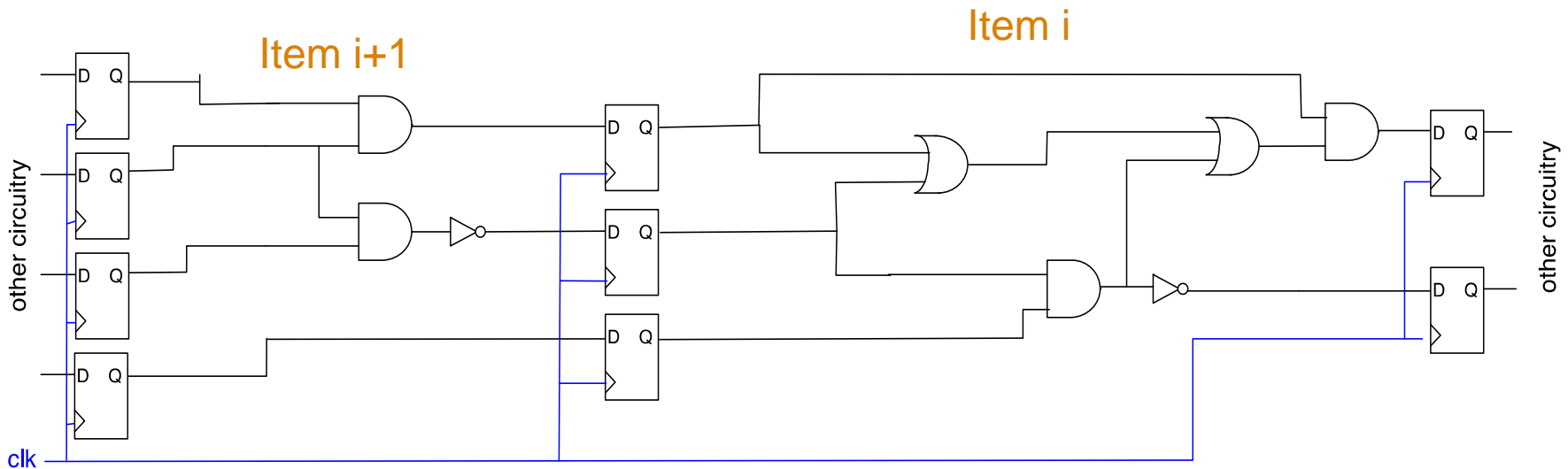
So, this now goes at  $1/3\text{ns} = 333\text{ MHz}$  (before it was  $1/5 = 200\text{ MHz}$ )

But it takes 2 cycles to get a result now.

Have we really improved anything?

Pipelining might help if:

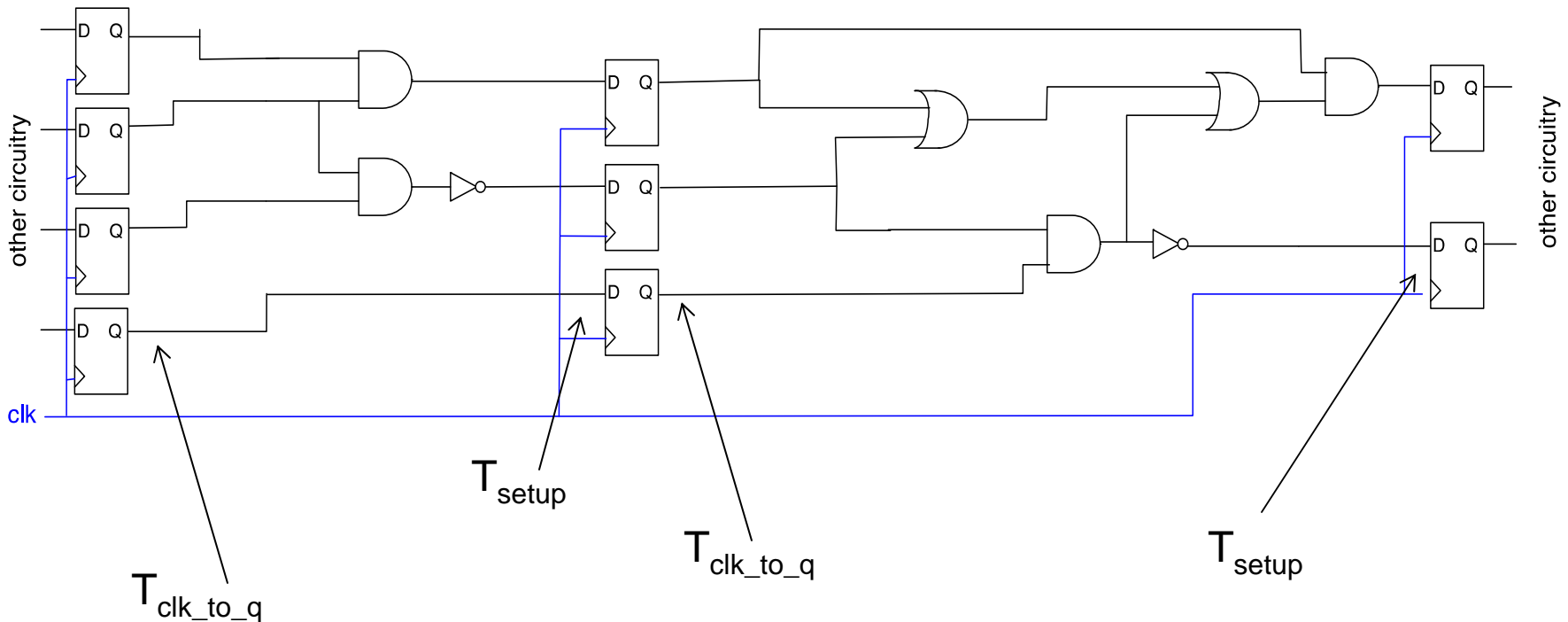
1. One (or several) of the paths are very long, compared to the others.  
In that case, these slow paths dictate performance. If we pipeline the slow parts, we can run the whole circuit faster.
2. We can sometimes have multiple data items “in fly” at once:



# Limits to pipelining

There is a limit to how much benefit you can get from pipelining:

- Every pipeline stage has the overhead of  $t_{\text{clk\_to\_q}}$  and  $t_{\text{setup}}$ .





In an FPGA it is even worse if you have to go into a logic element just to use it's flip-flop.

Pipelining changes the timing behaviour of your circuit

- Need to take this into account when you are designing your datapath / controller

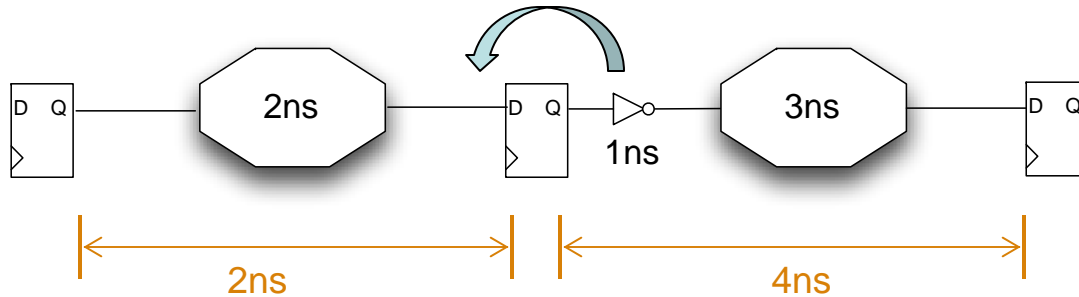
Quartus II will **not** pipeline your circuit for you. Why?

- Synthesis tools are “cycle accurate”: the behaviour of the circuit must be the same, cycle-by-cycle, as the VHDL specification.
- Pipelining a design would create a different cycle-by-cycle behaviour

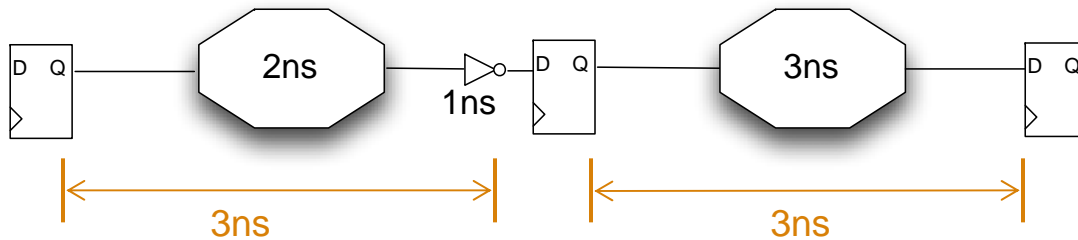
# RETIMING

Key to pipelining is making every state balanced

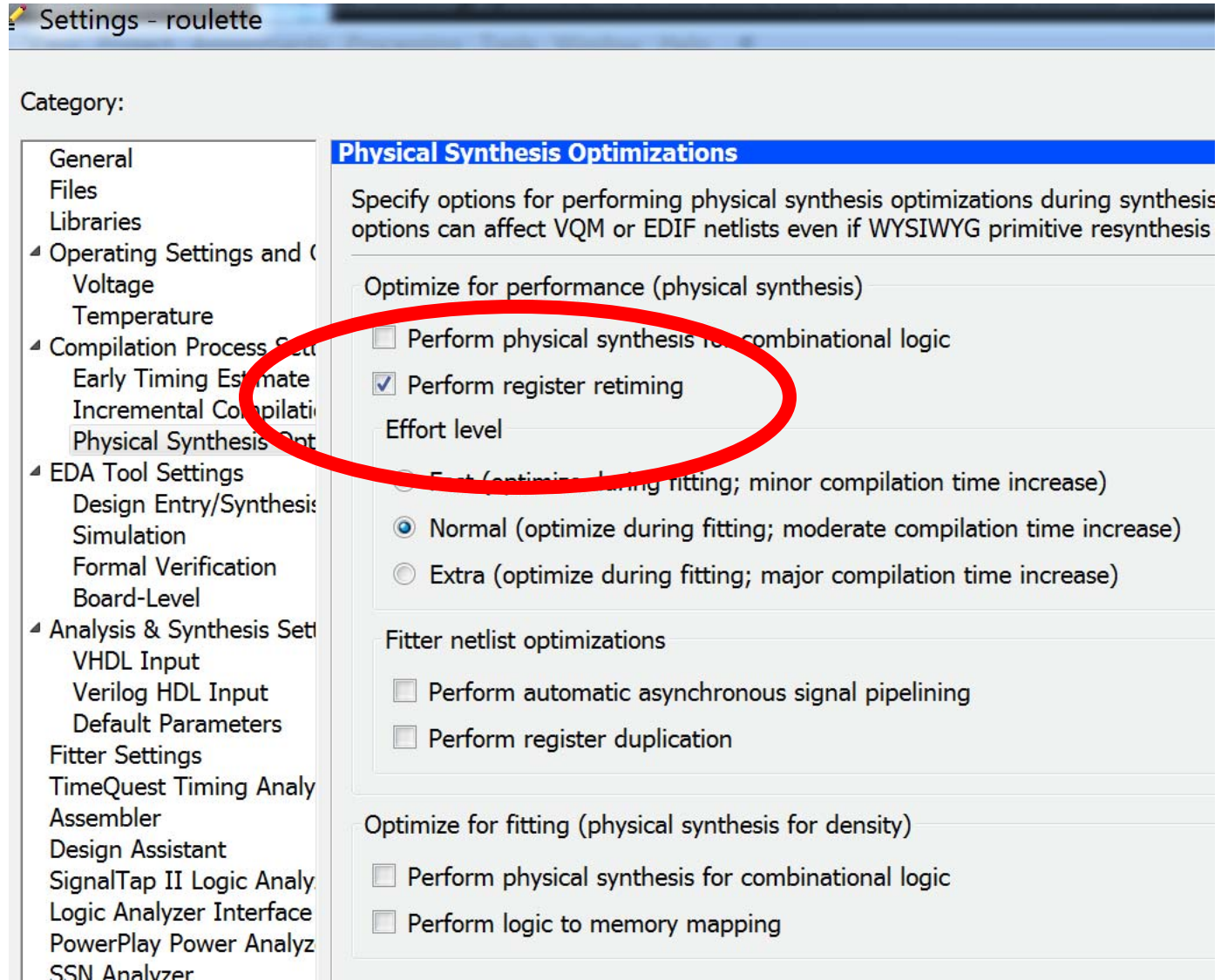
In some cases, it might make sense to move a gate from one side of the flip-flop to another:



$$\begin{aligned}\text{Max Freq} &= \\ 1/4\text{ns} &= \\ 250 \text{ MHz}\end{aligned}$$



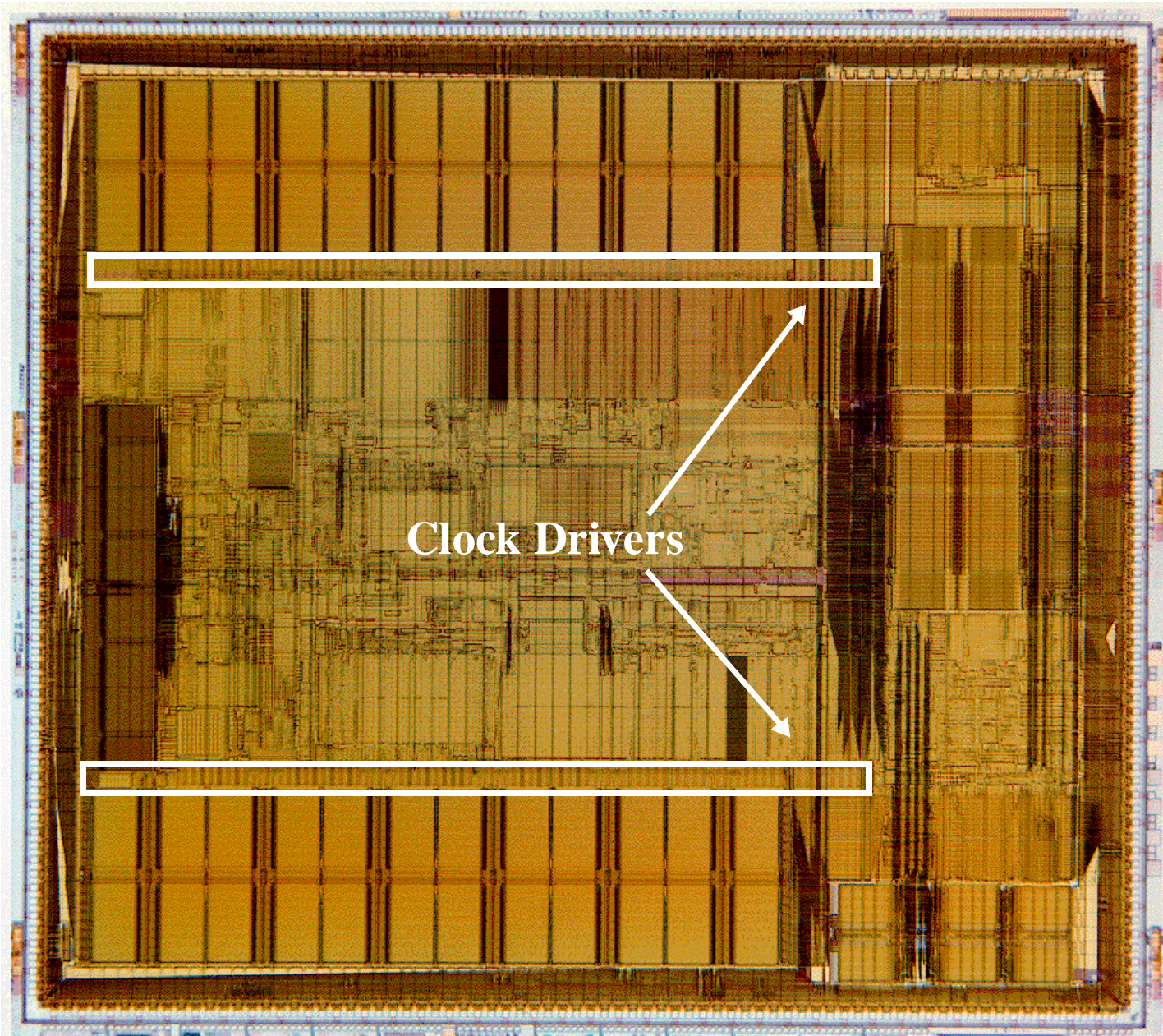
$$\begin{aligned}\text{Max Freq} &= \\ 1/3\text{ns} &= \\ 333 \text{ Mhz}\end{aligned}$$



In the Lab 4 case study, turning this on increased my Fmax from 59 MHz to 76 MHz !!

# Clock Skew





Clock Drivers

# Clock distribution network

- Ideal clock: clock's rising edges arrive at FFs at the same time
- Real implementation:
  - Driving capability of each cell is limited
  - Need a network of buffers to drive all FFs
  - In ASIC: done by clock synthesis (a step in physical synthesis)
  - In FPGA: pre-fabricated clock distribution network



# Clock Skew:

---

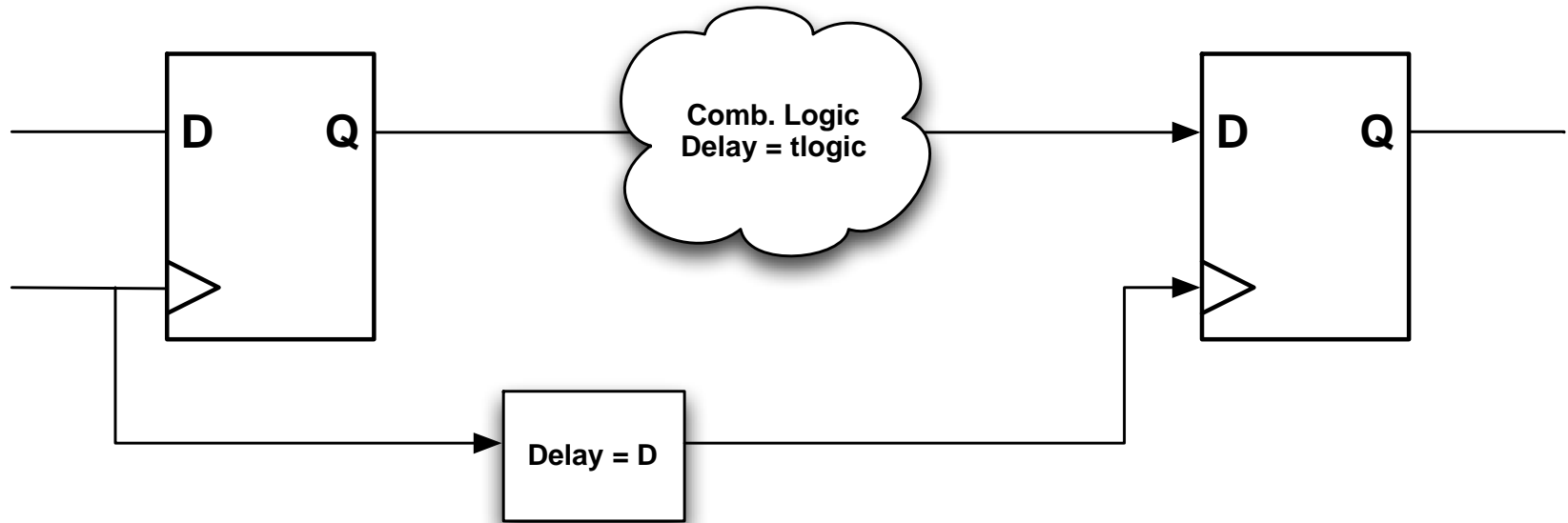
Clock Skew is very real:

- We can not guarantee that the clock edge arrives at all flip-flops at the same time

Implications range from:

- Improvement in  $F_{max}$  (not very likely)
- Reduction in  $F_{max}$
- Failure of the design, regardless of  $F_{max}$

# Clock Skew:



If  $D$  is 0 (no clock skew), we know from the previous slide set that:

$$t_{\text{clock}} \geq t_{\text{clk\_to\_q}} + t_{\text{logic}} + t_{\text{setup}}$$

What if  $D > 0$  (clock skew)?

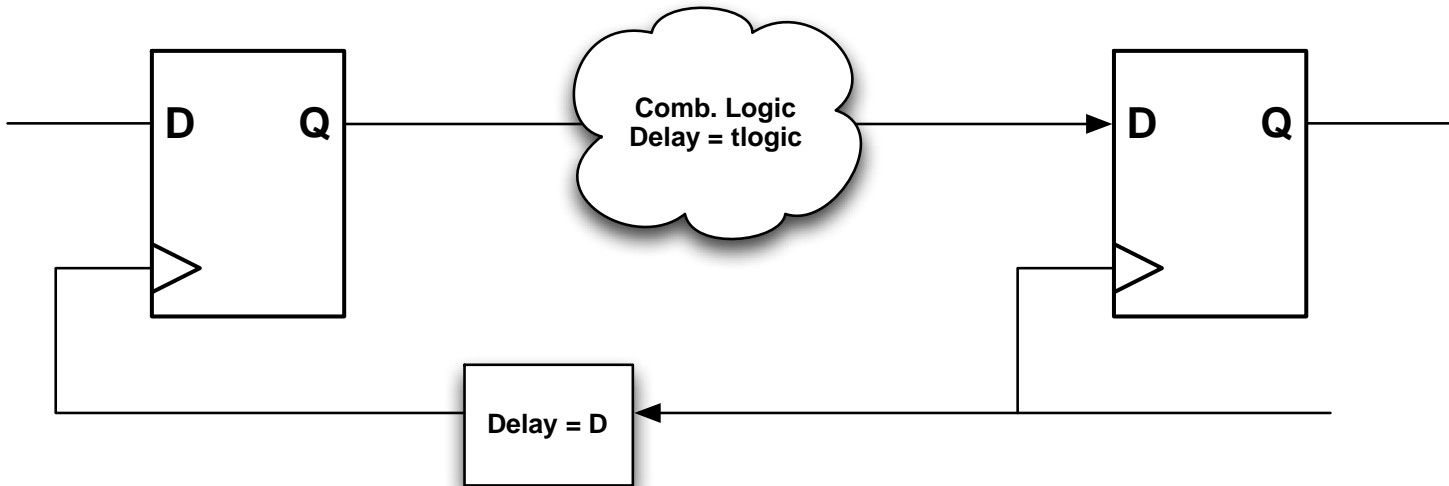
$$t_{\text{clock}} + D \geq t_{\text{clk\_to\_q}} + t_{\text{logic}} + t_{\text{setup}}$$

Or:

$$t_{\text{clock}} \geq t_{\text{clk\_to\_q}} + t_{\text{logic}} + t_{\text{setup}} - D$$

In this case, clock skew  
increased our clock speed!

# Clock Skew:

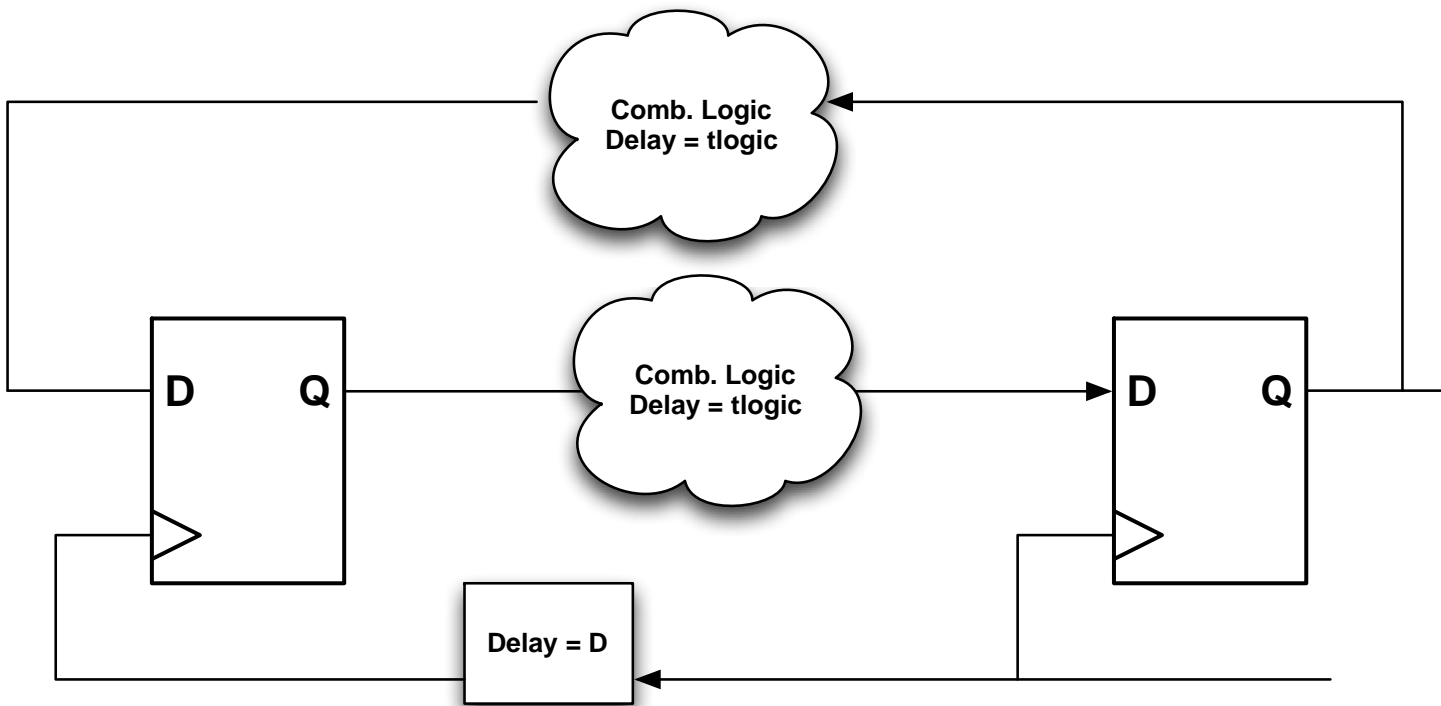


$$t_{\text{clock}} - D \geq t_{\text{clk\_to\_q}} + t_{\text{logic}} + t_{\text{setup}}$$

Or:

$$t_{\text{clock}} \geq t_{\text{clk\_to\_q}} + t_{\text{logic}} + t_{\text{setup}} + D$$

In this case, clock skew decreased our clock speed!



In this case, the minimum clock period decreases if we only consider the top both and increases if we only consider the bottom path.

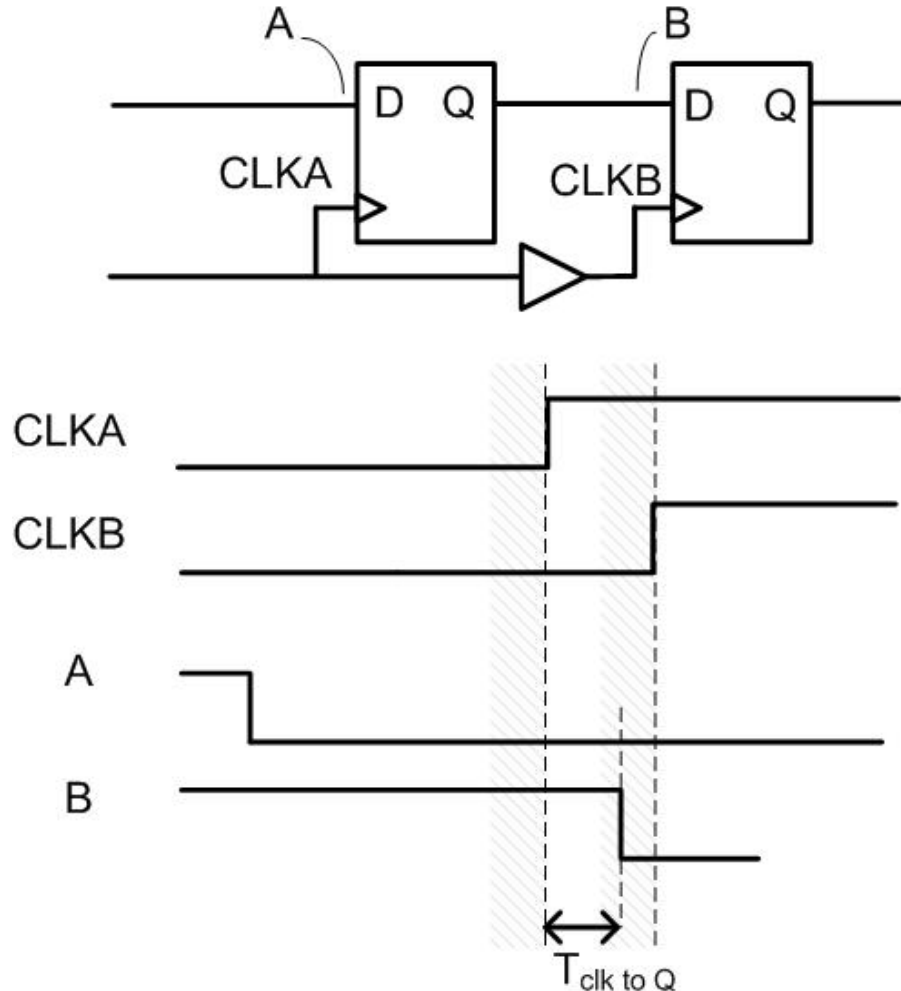
Since the critical path is the worst case path, in this case:

$$t_{\text{clock}} \geq t_{\text{clk\_to\_q}} + t_{\text{logic}} + t_{\text{setup}} + D$$

ie. Overall, the clock skew increases the critical path (slows us down)

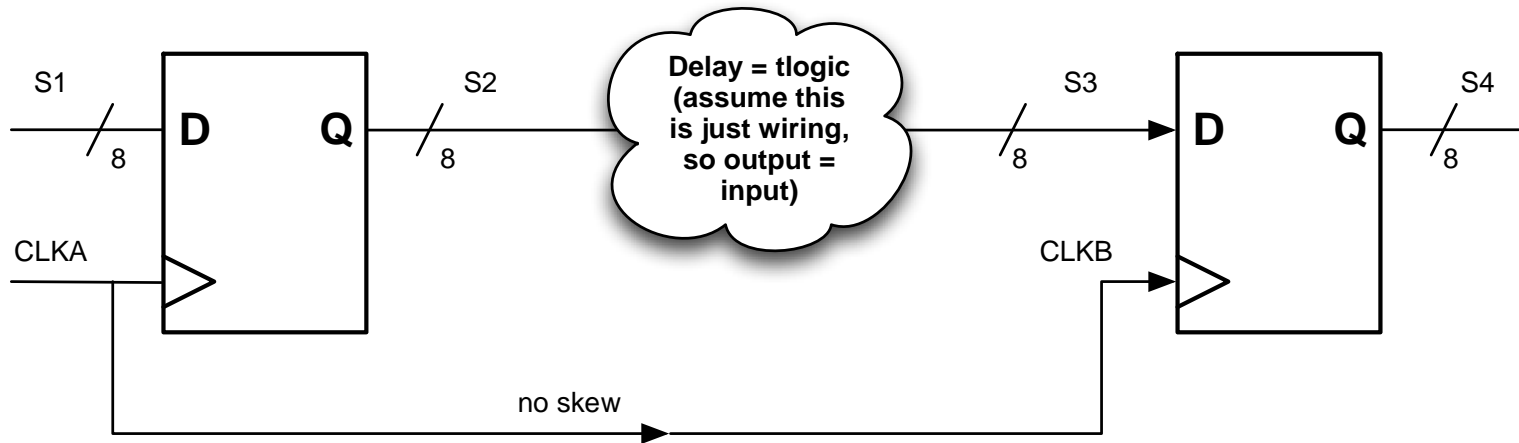
# Clock Skew:

Clock skew can also cause hold time violations:

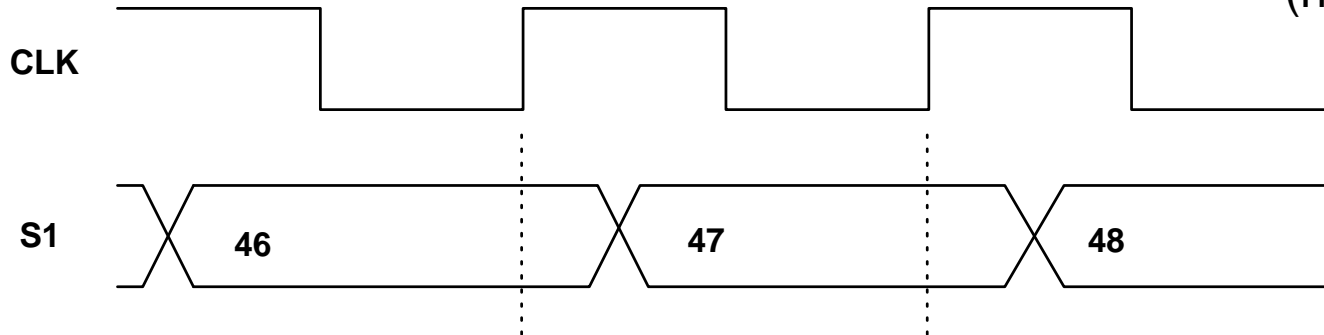


Tool needs to check for the worst case skew when adjusting routing to deal with hold time violations.

Clock skew can also functional problems, even if hold time is not an issue:

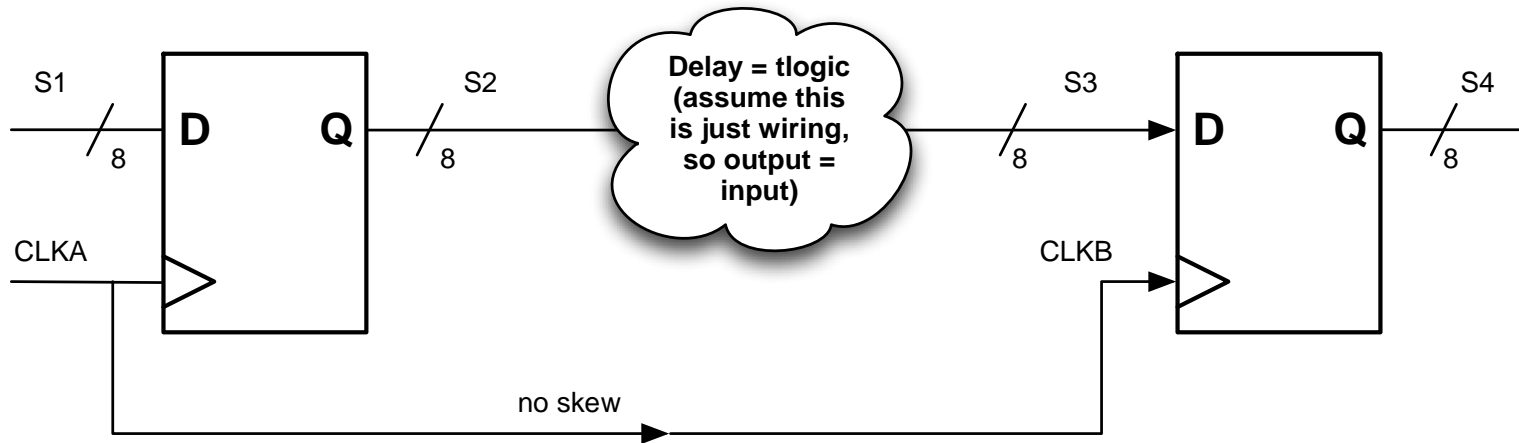


Correct Operation  
(no skew)

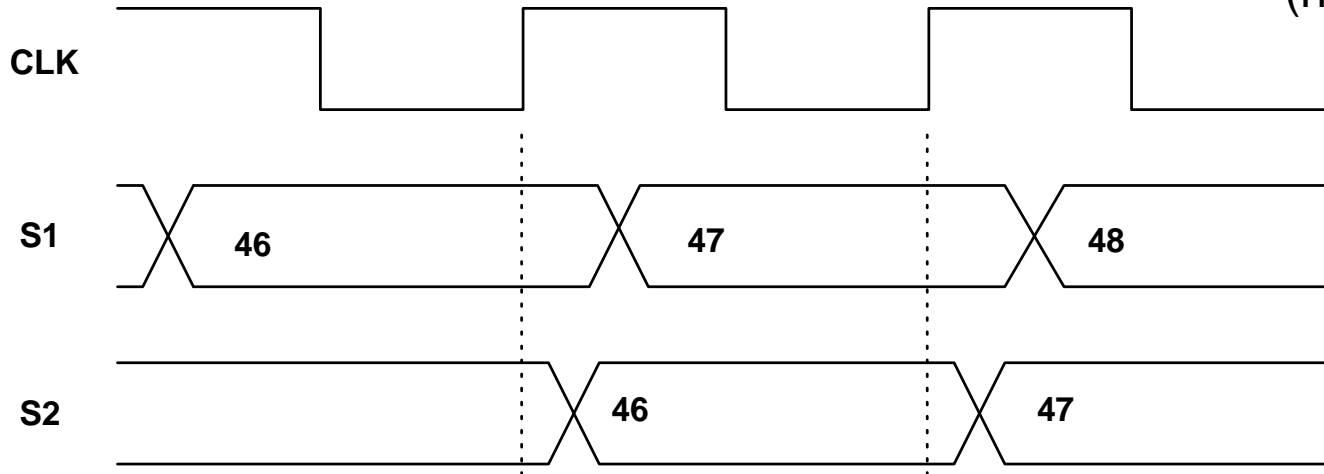


-  
-  
-  
-  
-

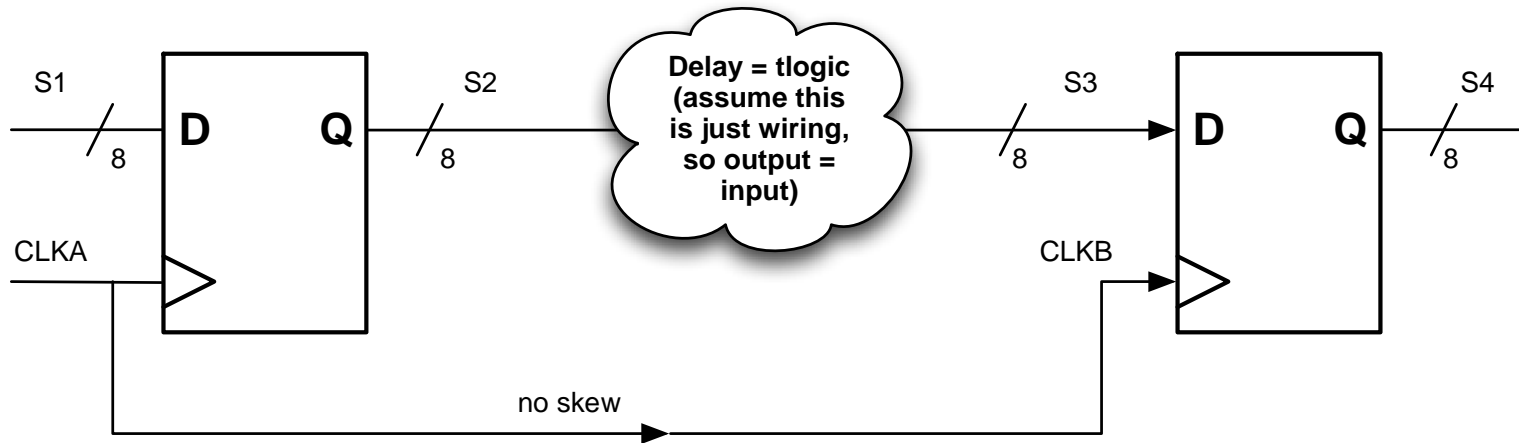
Clock skew can also functional problems, even if hold time is not an issue:



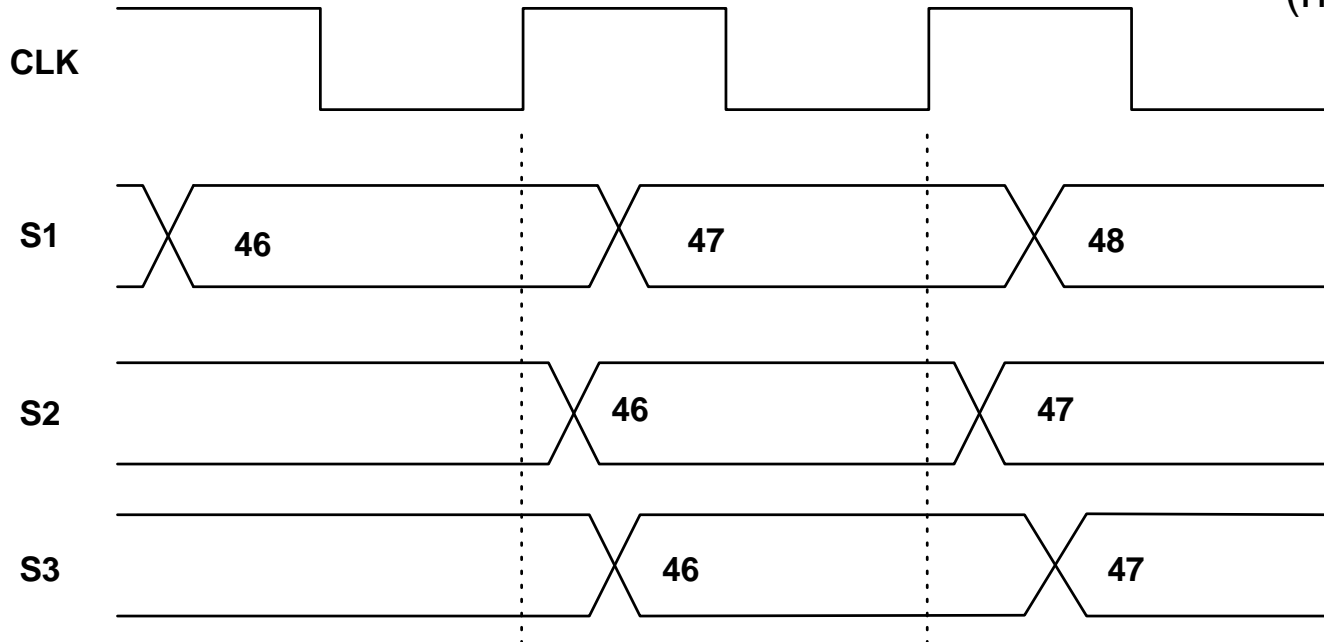
Correct Operation  
(no skew)



Clock skew can also functional problems, even if hold time is not an issue:

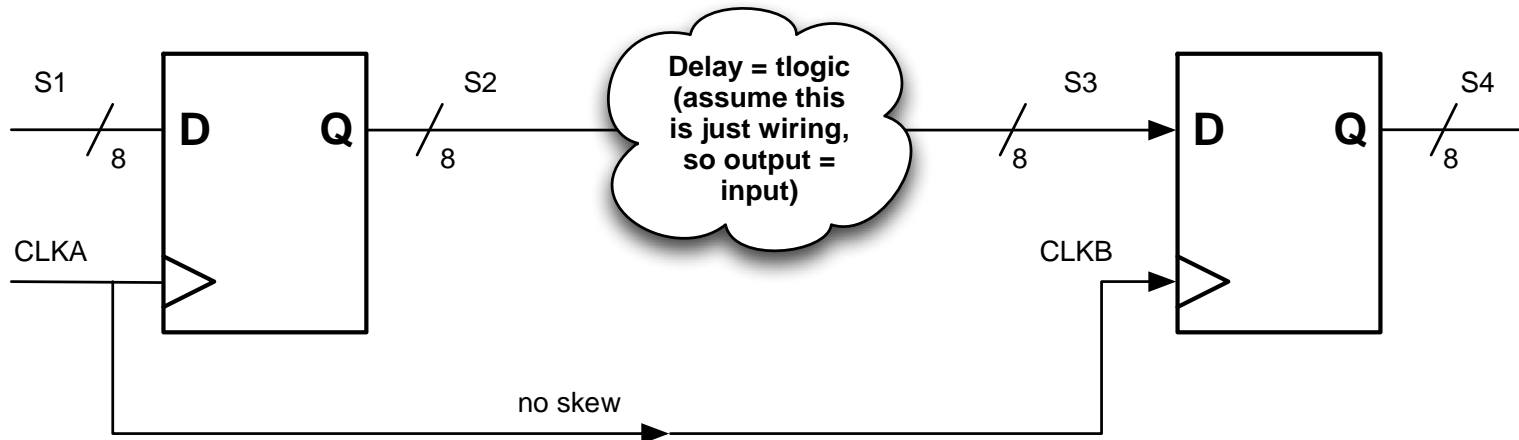


Correct Operation  
(no skew)

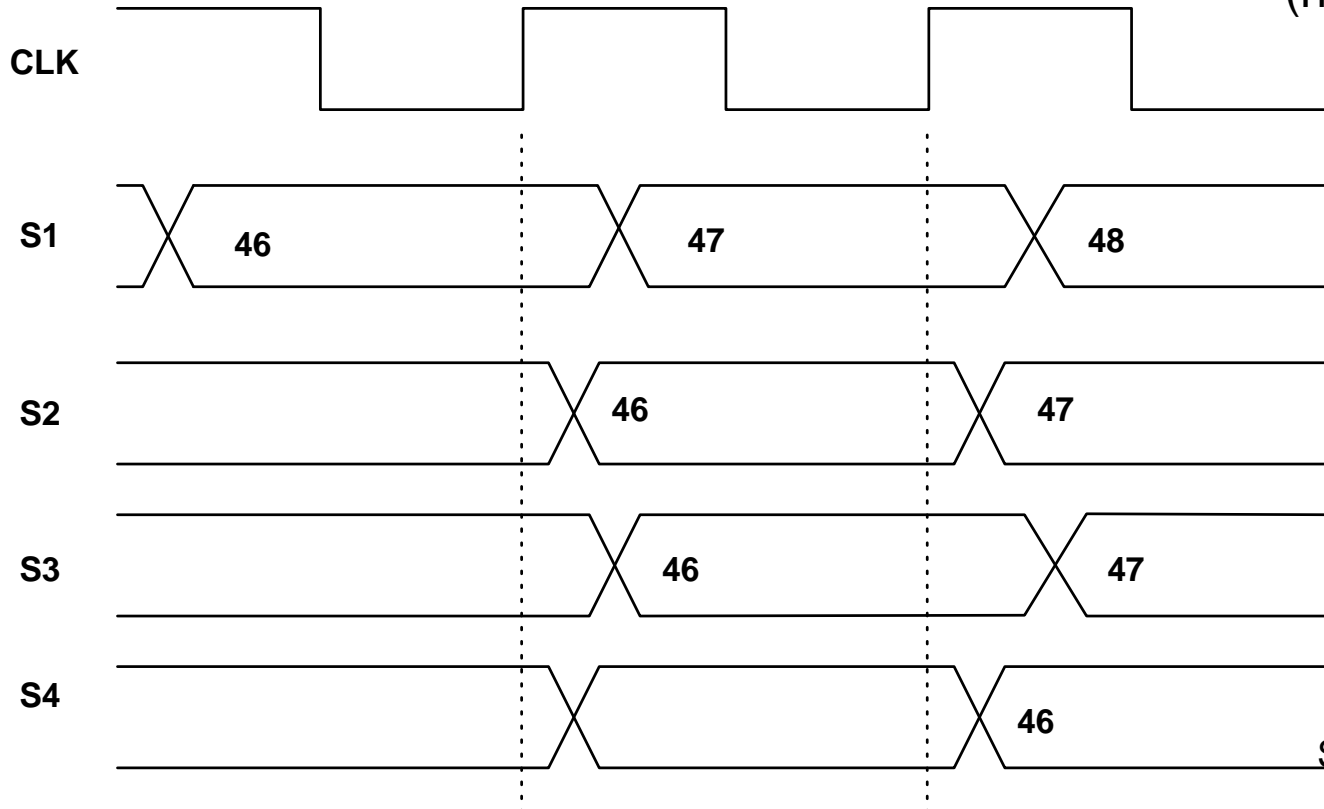




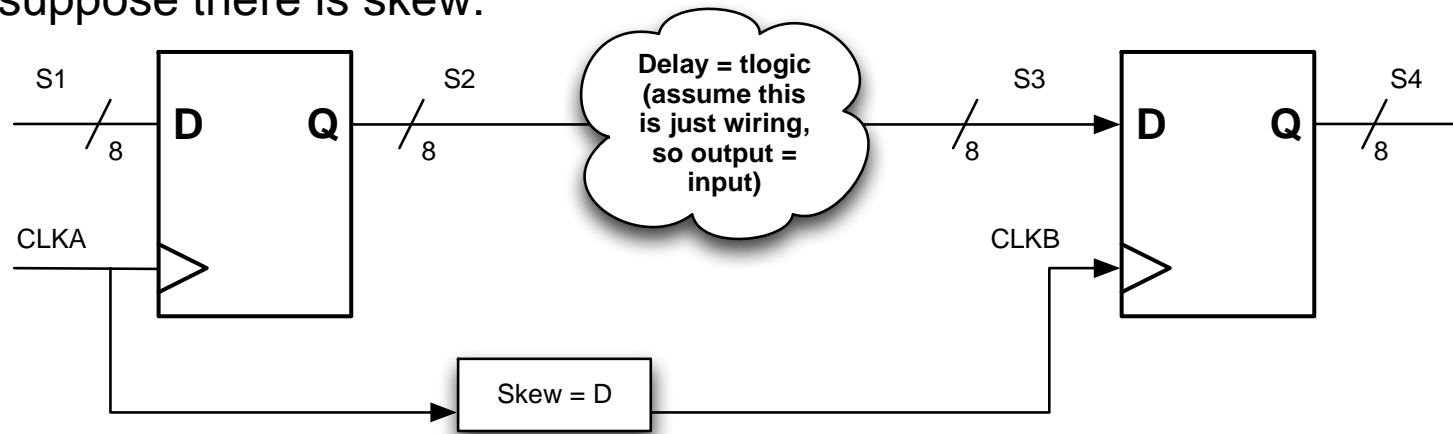
Clock skew can also functional problems, even if hold time is not an issue:



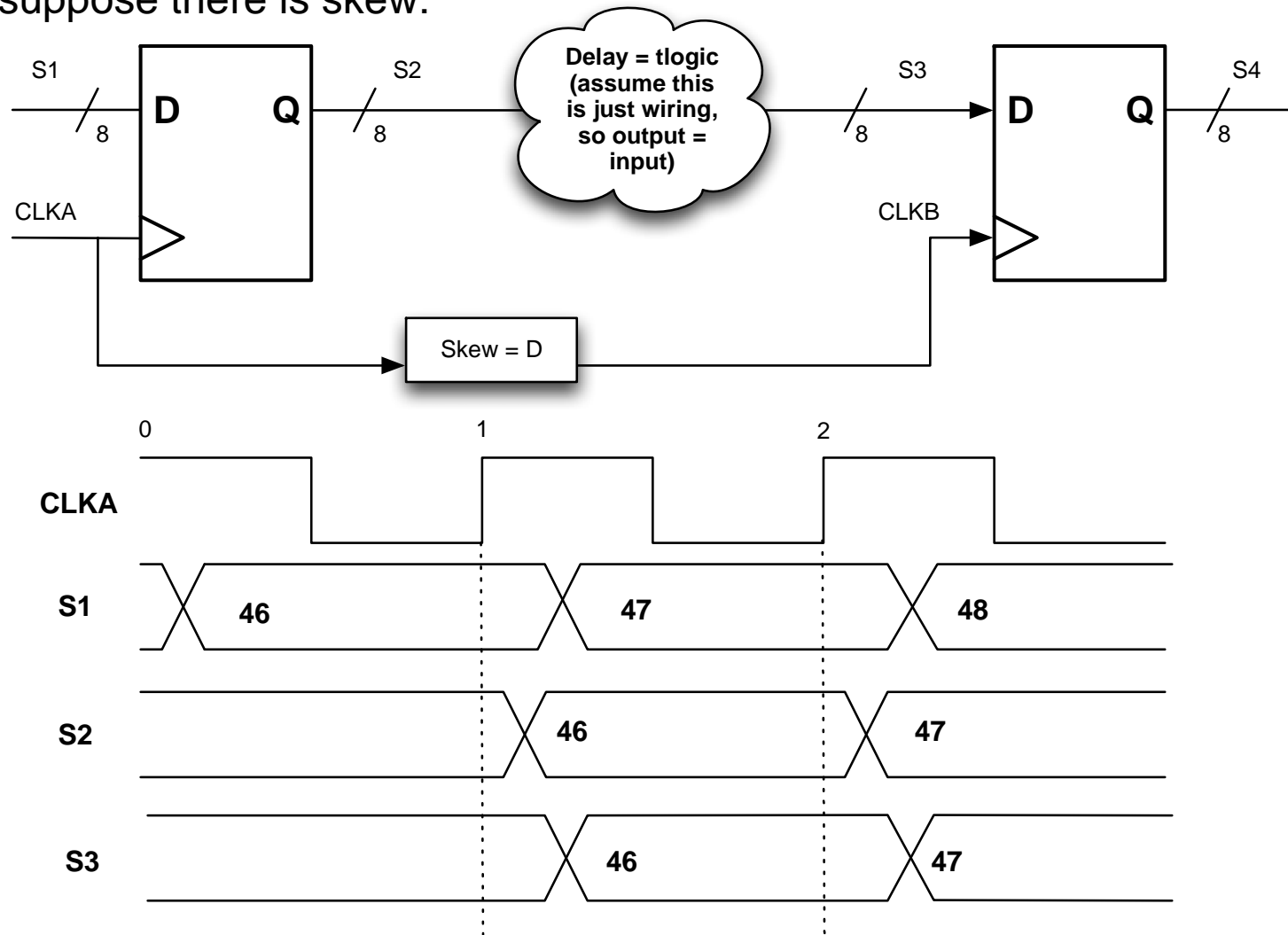
Correct Operation  
(no skew)



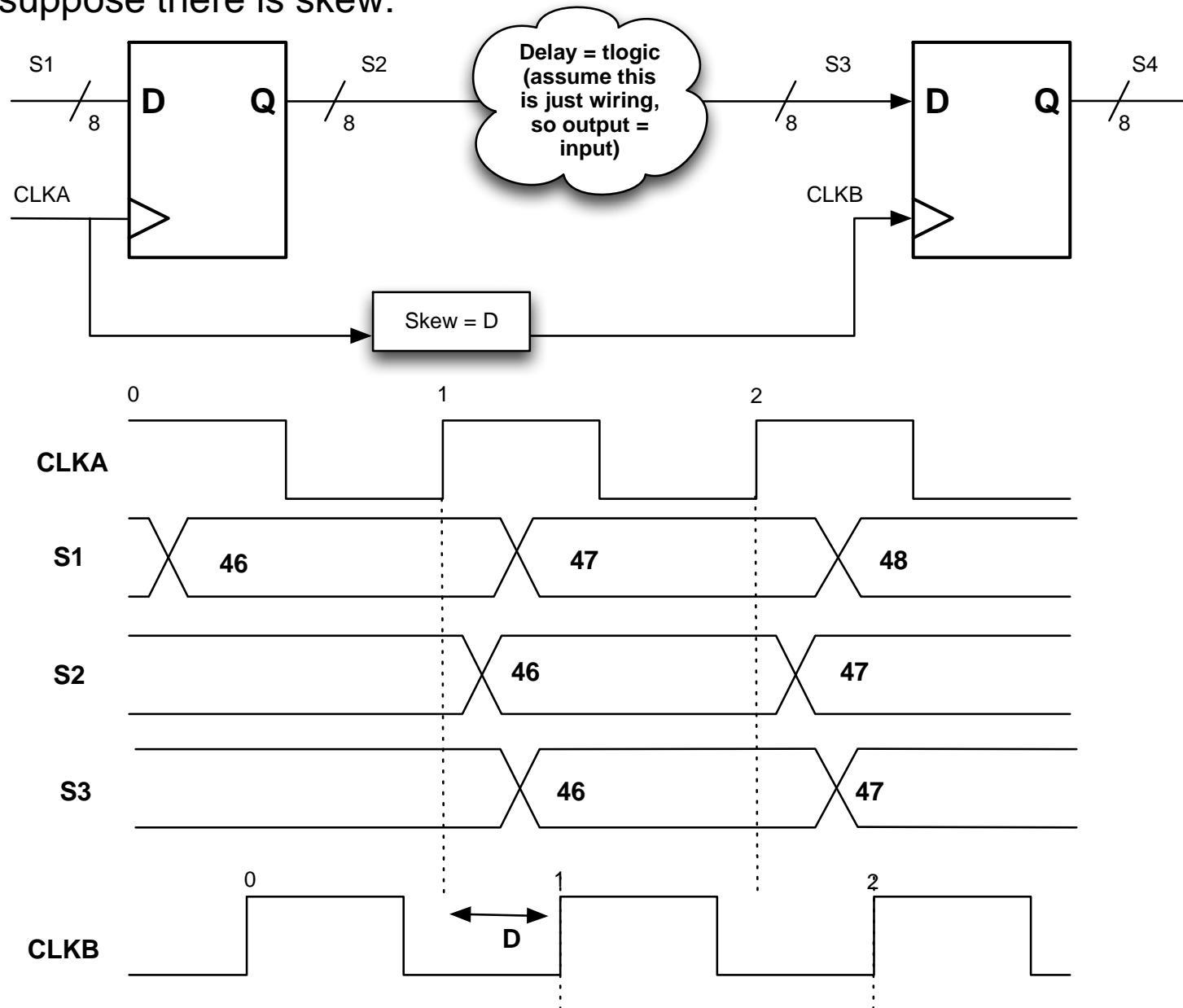
Now suppose there is skew:



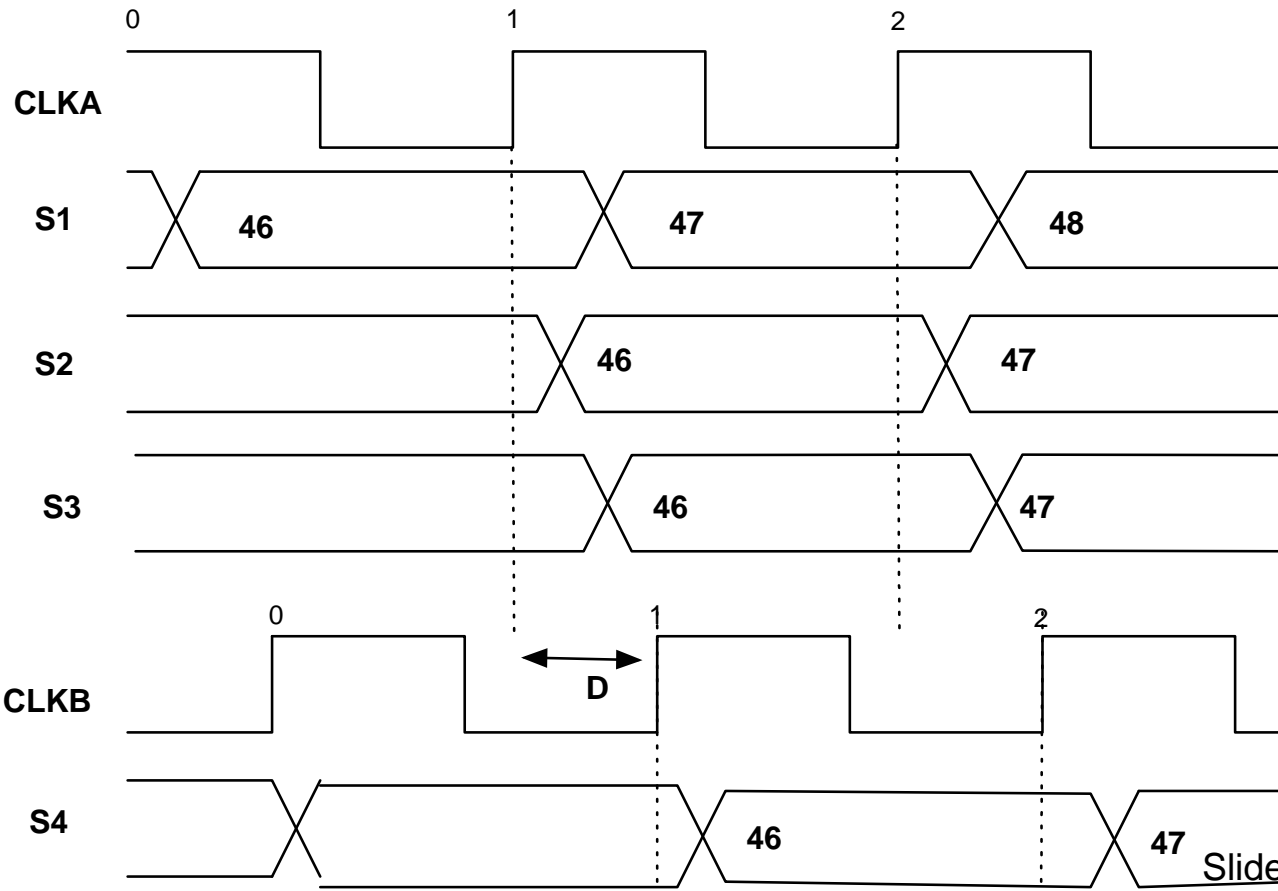
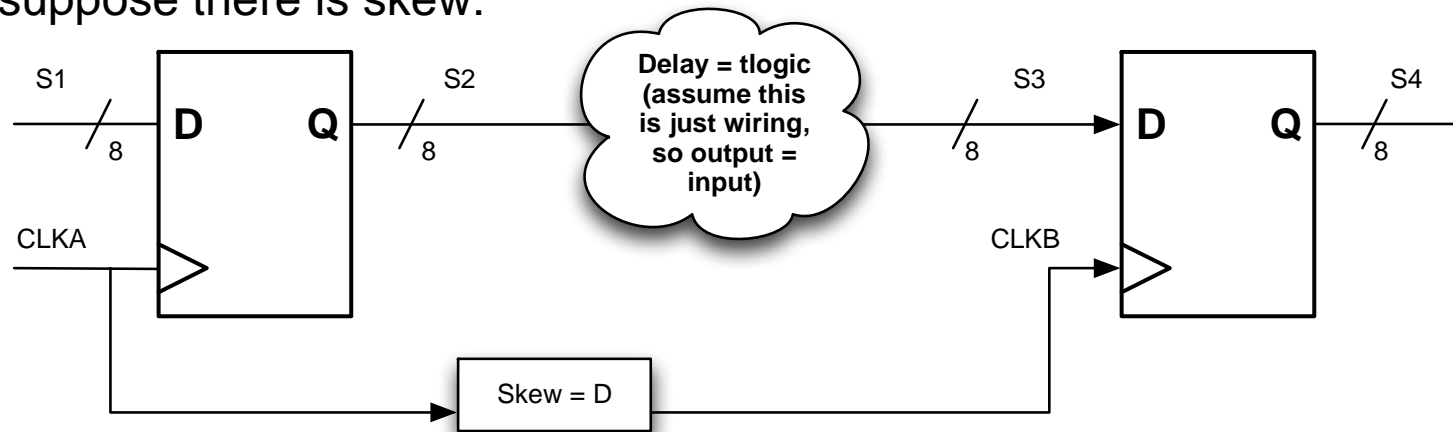
Now suppose there is skew:



Now suppose there is skew:



Now suppose there is skew:



Due to clock edge 1, 46 is latched into second register.

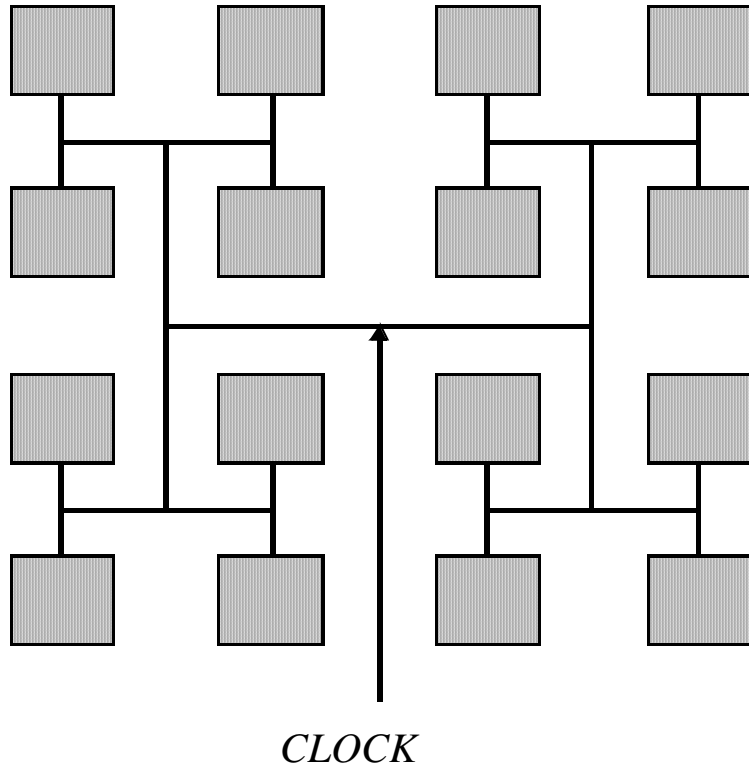
46 should be latched into second register at clock cycle 2

In this example, skew caused the second flip-flop to read in its value early.

This is most certainly a functional problem

Moral: Clock skew needs to be avoided as much as possible

# Clock Skew: What can we do? Custom Chip



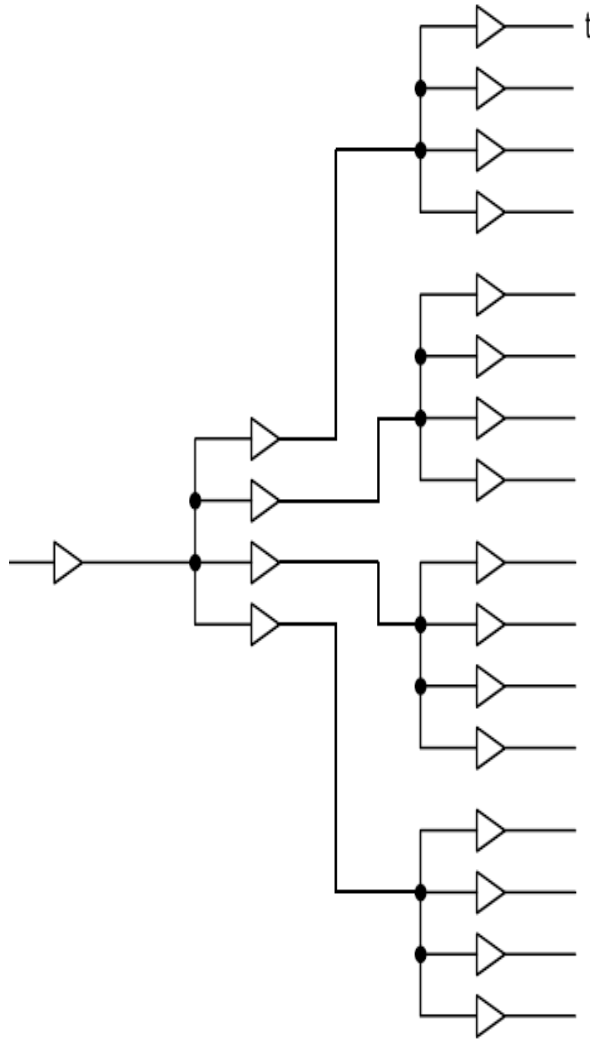
**H-Tree Network**

**Observe: Only Relative Skew is Important**

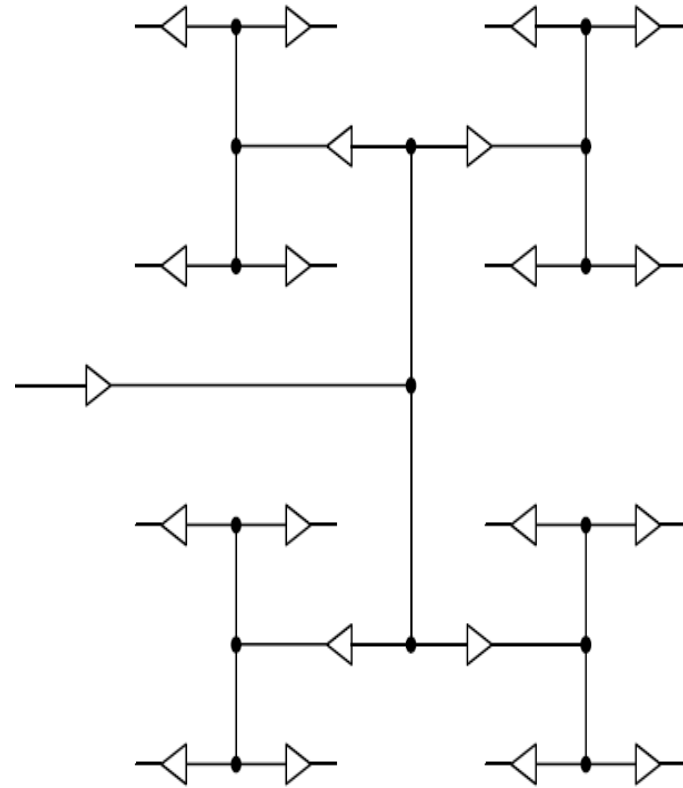
Layout clock signal as to minimize skew

Need to verify timing extensively to make sure skew is not going to cause failure. Across all process corners!

- Block diagram



- Ideal H-routing

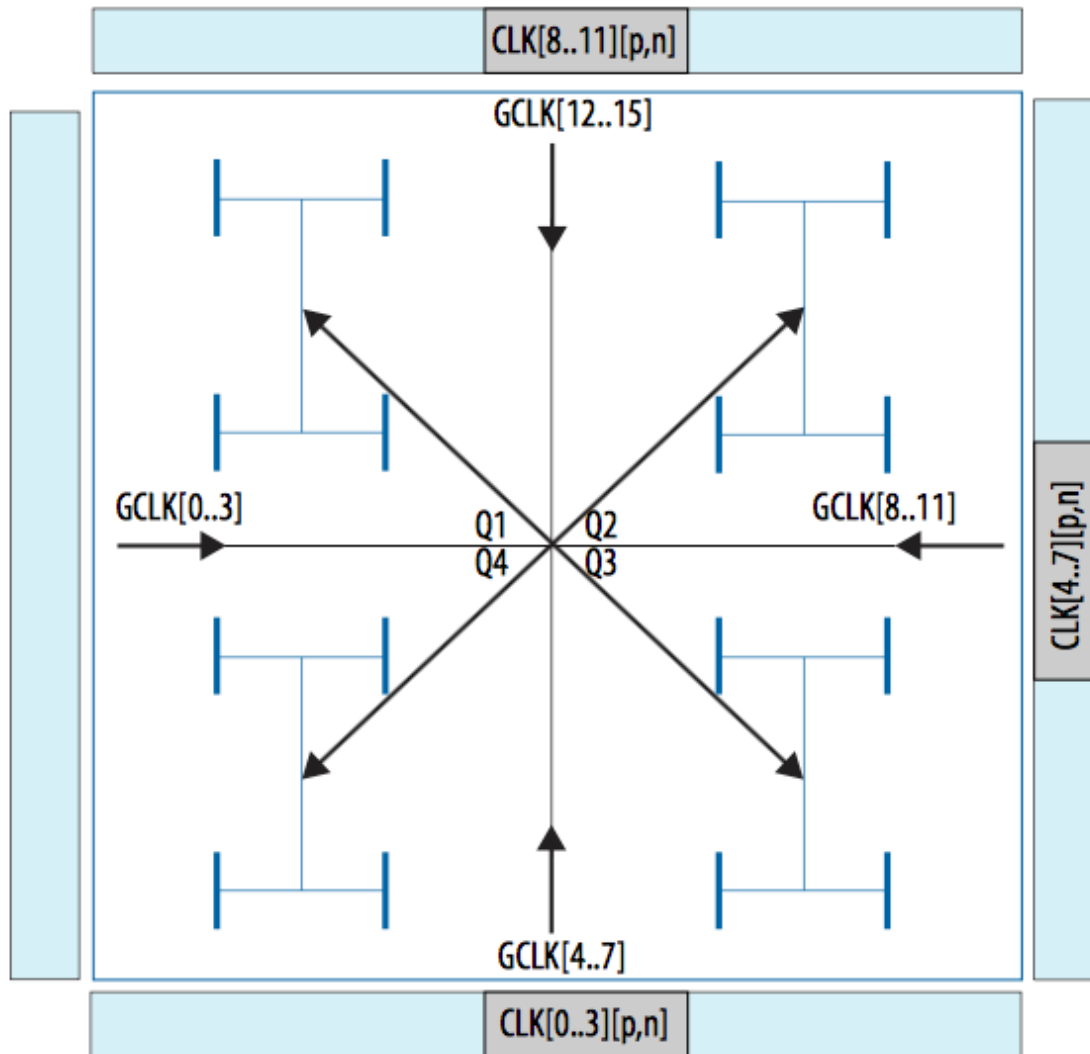




# Global and Local Clocks

- Global clocks use dedicate routing
  - Therefore there are not many global clocks (a few dozen at most)
  - Use them wisely!
  - Quartus will automatically assign them to high fanout clocks, you can overrule it
- Local clocks use regular routing
  - slower and prone to clock skew
  - but you can have an unlimited number of them
- In general try to have as little clock domains as possible (1 is ideal) and try to only use global clocks if possible.

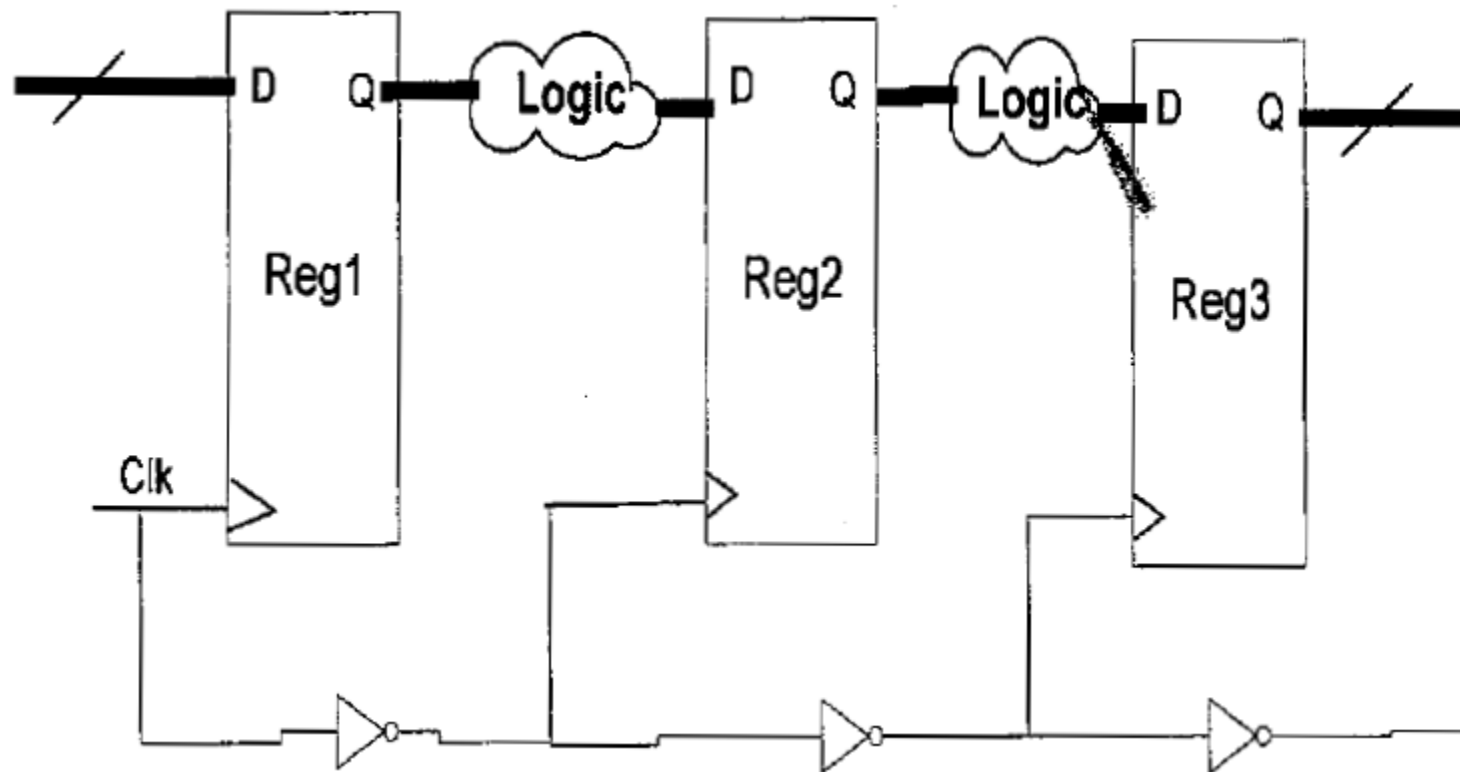
# Clock Skew: FPGAs



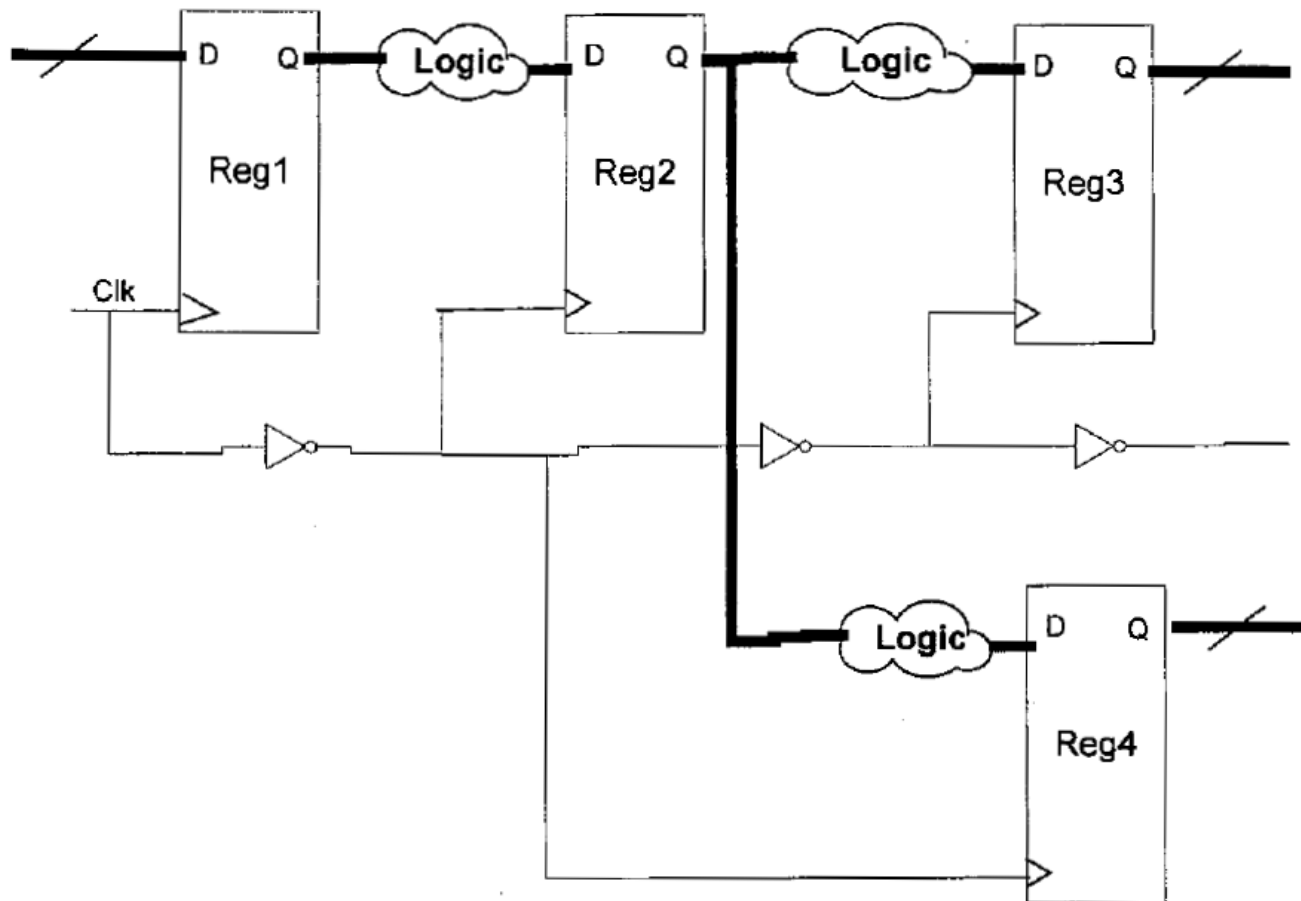
FPGA Vendors include complex clock networks in their chip to try to distribute clock edges as evenly as possible.

The CAD tools automatically verify designs to make sure there are no clock skew problems.

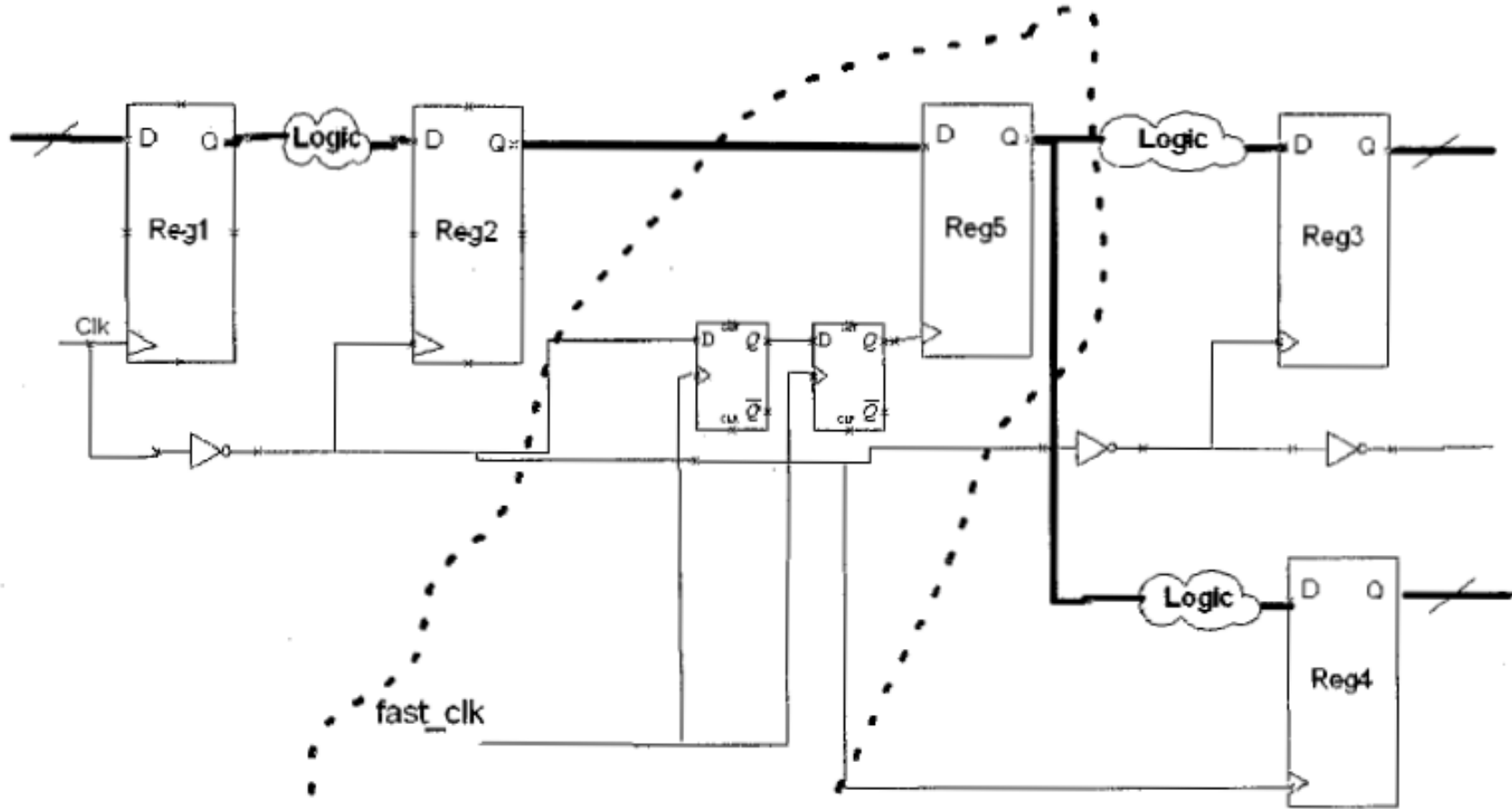
# Domino Logic



# What if you have this topology?



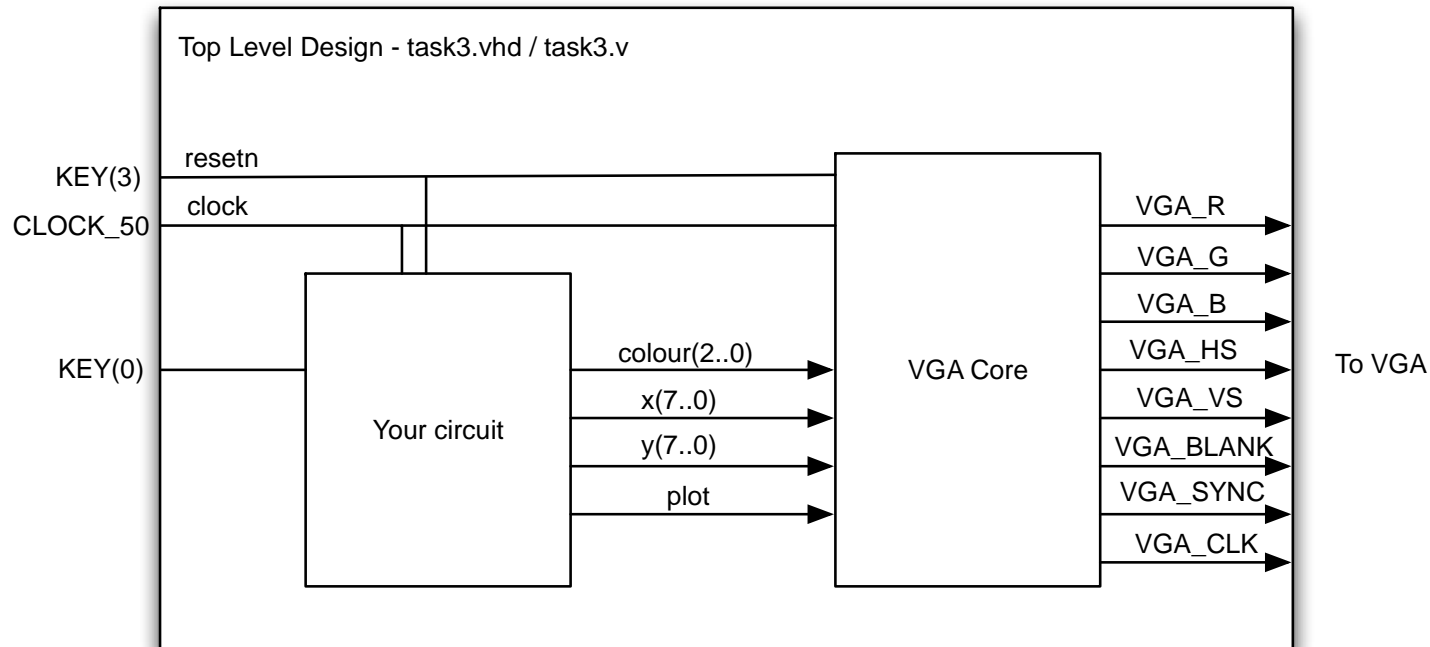
# Solution: Generate hold time



This module was  
called "hold\_gen"

# PLLs: Motivation

Consider the VGA core from Labs 2/3:



Your chip receives a 50 MHz clock

The VGA spec says VGA\_CLK needs to run at 25 MHz.

The designer of the VGA core had to make a 25MHz clock out of a 50MHz clock

# PLLs: Motivation

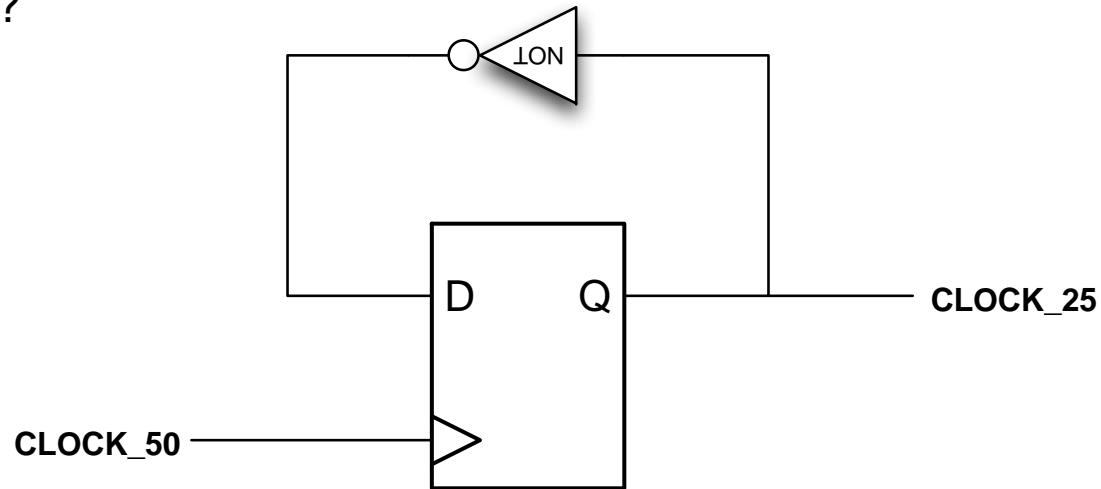
---

How would you make a circuit that creates a 25 MHz clock from a 50 Mhz clock?

# PLLs: Motivation

---

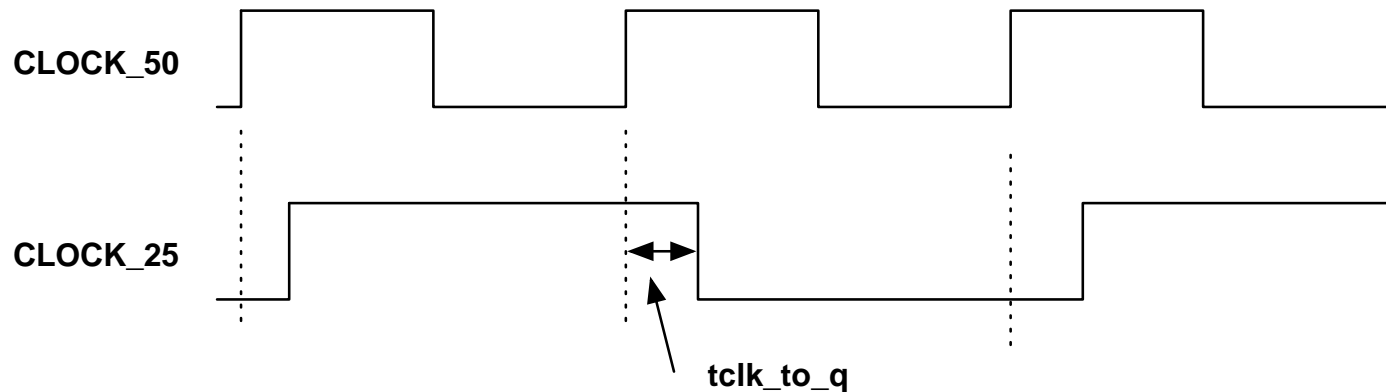
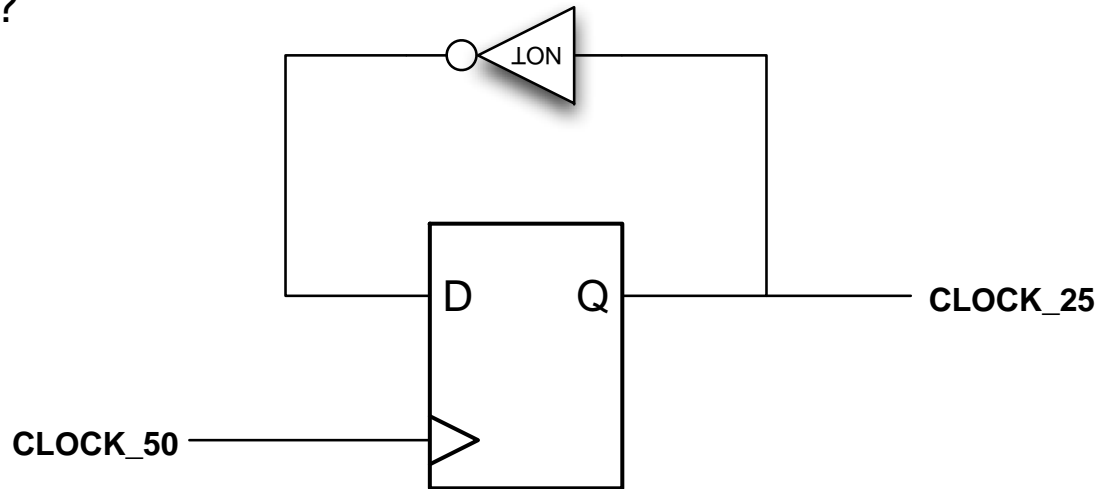
How would you make a circuit that creates a 25 MHz clock from a 50 MHz clock?





# PLLs: Motivation

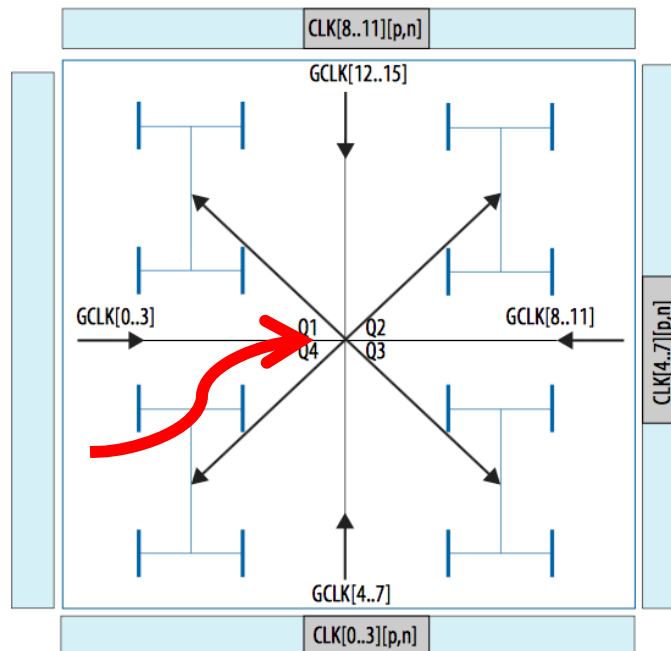
How would you make a circuit that creates a 25 MHz clock from a 50 Mhz clock?



# PLLs: Motivation

Two problems with this solution:

- 1.If you wanted to use both CLOCK\_50 and CLOCK\_25 in your design, there is skew between them.
- 2.Even if you only used CLOCK\_25, you need to find a way to distribute this across the chip. FPGA routing is slow and unpredictable.

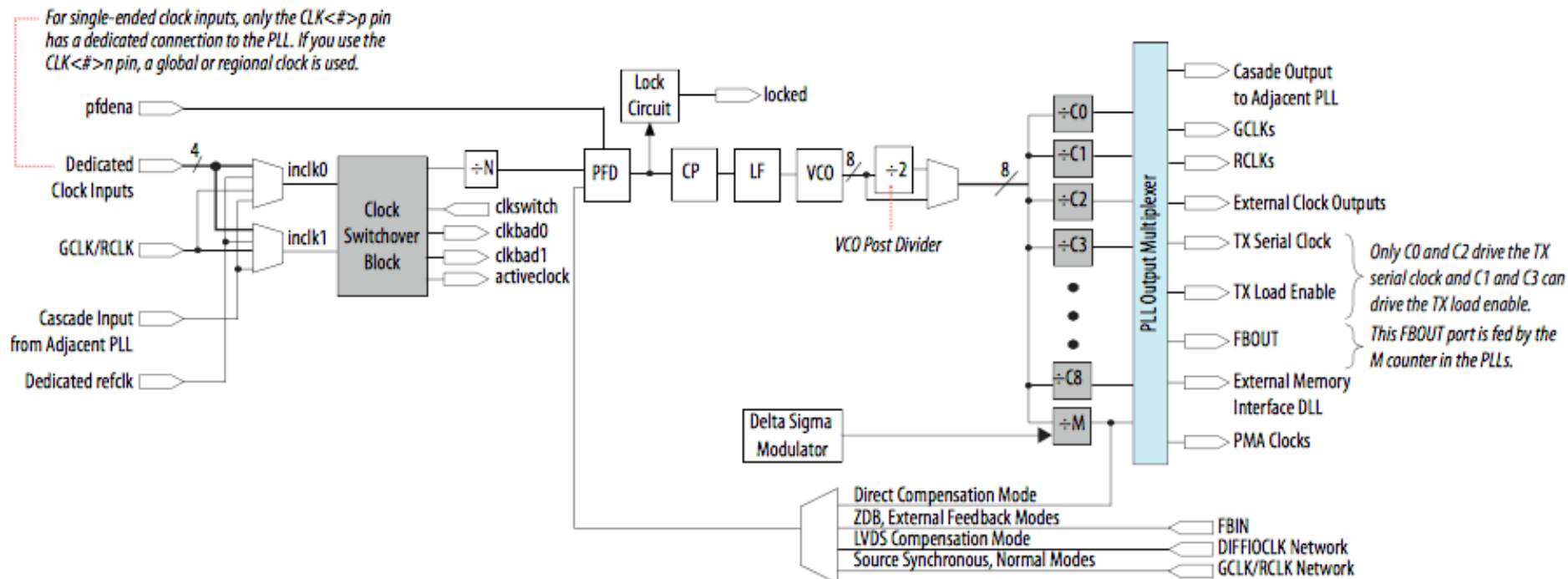


Even if we want to use the global clock wires, we still have to route from our FF output to the source of these wires.

# Phase Locked Loops

Modern FPGAs contain Phase Locked Loop hard cores:

- A mixed signal ( = both analog and digital) circuit that generates output clocks aligned to an input clock
- Can act as a clock speed divider or multiplier
- Can directly feed clock distribution network
- Automatically adjusts phase to account for clock distribution network



# Phase Locked Loops

---

If you look inside vga\_pll.v and vga\_adapter.v :

```
altpll      altpll_component (.inclk (clock_input_bus),  
.clk (clock_output_bus));  
  defparam  
    altpll_component.operation_mode = "NORMAL",  
    altpll_component.intended_device_family = "Cyclone II",  
  
    altpll_component.lpm_type = "altpll",  
    altpll_component.pll_type = "FAST",  
    altpll_component.inclk0_input_frequency = 20000,  
    altpll_component.primary_clock = "INCLK0",  
    altpll_component.compensate_clock = "CLK0",  
    altpll_component.clk0_phase_shift = "0",  
    altpll_component.clk0_divide_by = 2,  
    altpll_component.clk0_multiply_by = 1,  
    altpll_component.clk0_duty_cycle = 50;
```

```
vga_pll mypll (CLOCK_50, clock_25);
```

They use this to divide the clock frequency by 2. Can also multiply clock frequency.

Don't worry about details. If you ever need it you can look it up      Slide Set 13, Page 64

# Glitches

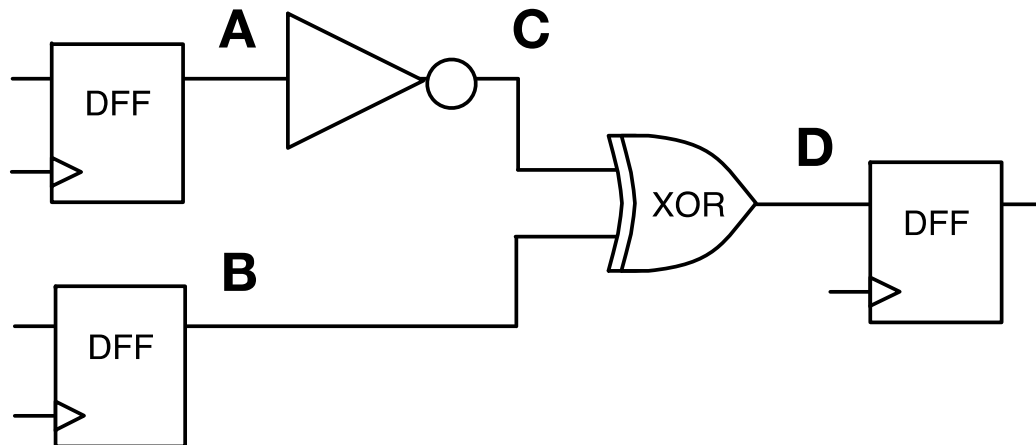
# Glitch

An undesired short-lived pulse that occurs before a signal settles to its intended value

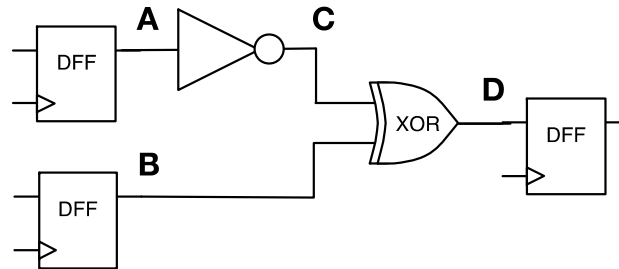
## Causes

- Unequal arrival times of inputs on combinational gates
- Various electrical effects (not covered in this course)

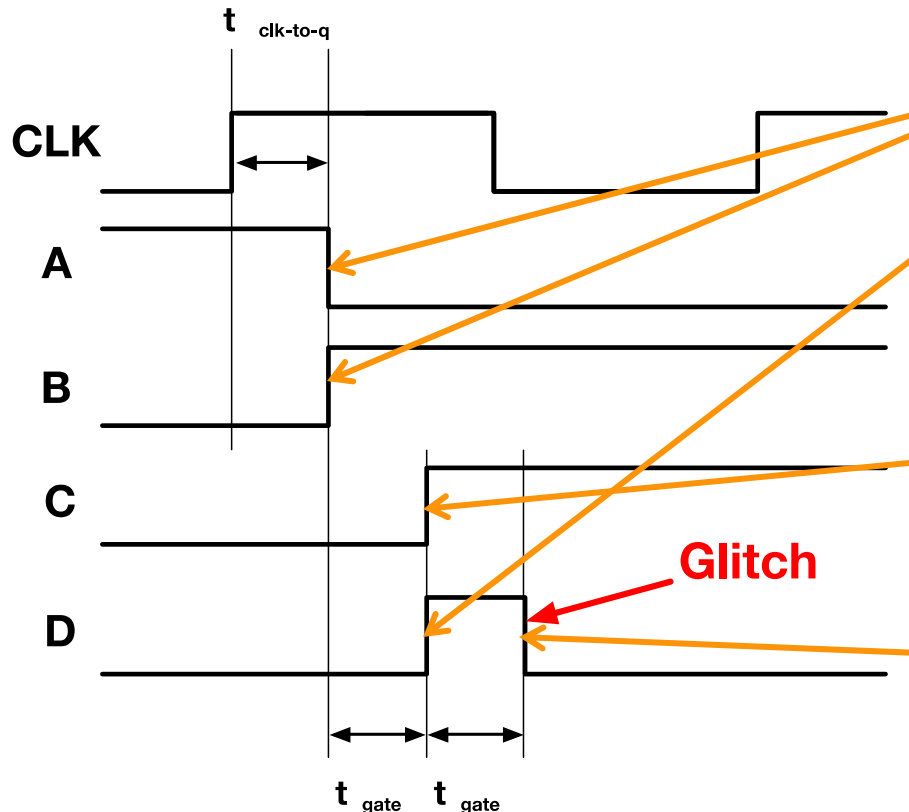
Example of a circuit with glitching:



# Glitch Example



B	C	D
0	0	0
0	1	1
1	0	1
1	1	0



1) A, B settle after clock-to-q delay

2a) B arrives immediately at XOR, causing a transition to 1 on D after one gate delay

2b) At the same time, A arrives at the INV immediately, causing a transition to 1 on C after one gate delay

3) After C settles, its value arrives at the XOR after one additional gate delay, causing a transition back to 0 on D

# Glitches – Key Take Aways

---

A signal may switch several times before settling to final intended value

## **This is OK if ...**

it settles before anyone tries to do something with that data.

i.e. Before a flip-flop tries to read it on the rising edge of next clock cycle

## Power and Energy

- Designers care about glitches because they consume unnecessary power!
- Any transition consumes power because the circuit is doing work to move electrons around.



# Learning Objectives

---

1. Understand what timing closure is and why it is difficult
2. Understand how pipelining can help with timing closure
3. Understand retiming and be able to apply it to a circuit
4. Be able to discuss the effects of clock skew
5. Understand what a PLL is used for
6. Understand the cause and impact of glitches caused by unequal combinational path delays