CPEN 311: Digital Systems Design

Slide Set 15: Metastability

2015/2016 Term 2

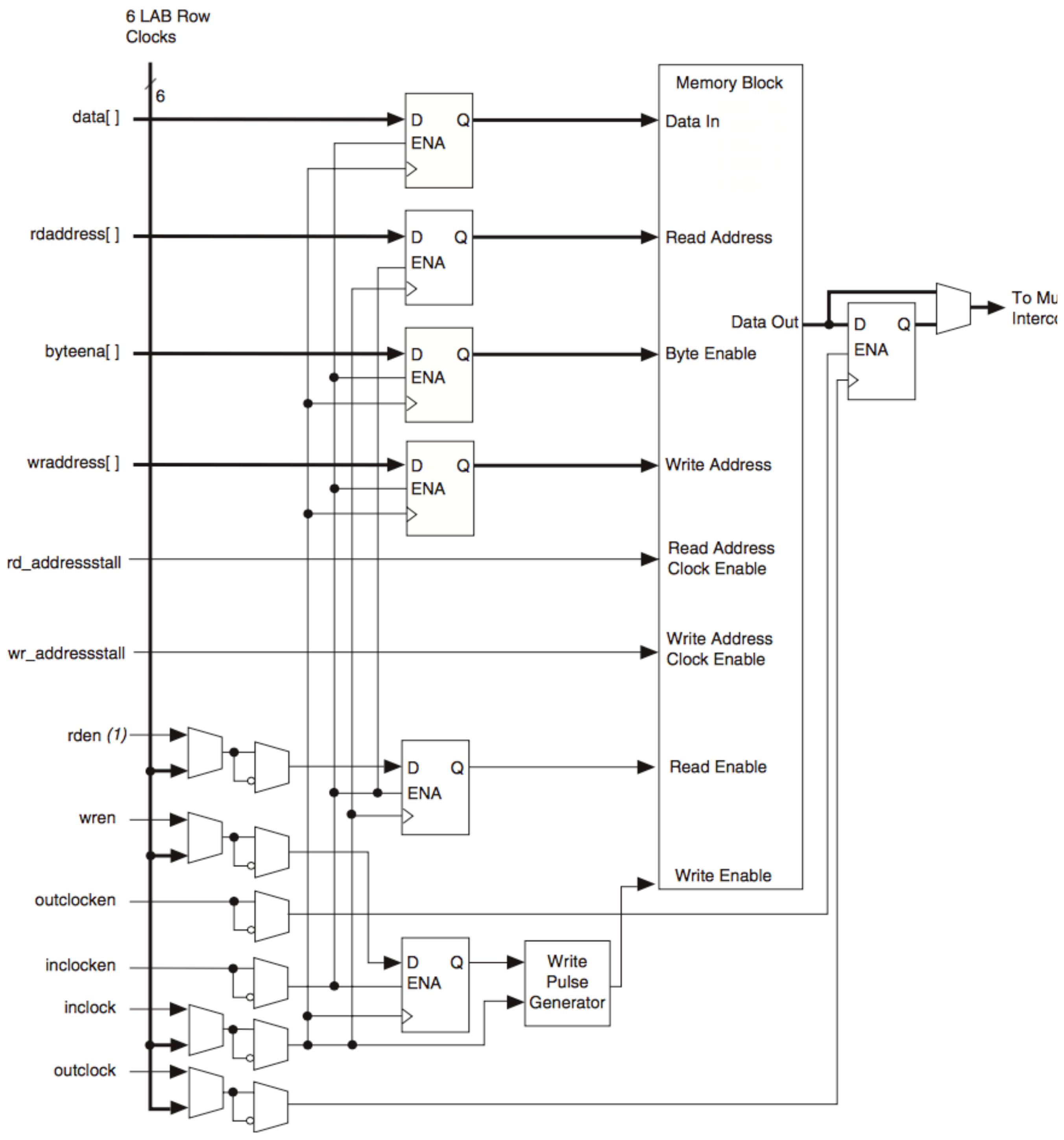
Instructor: Steve Wilton
stevew@ece.ubc.ca

Learning Objectives

1. Understand what metastability is, and how it can cause failure
2. Understand why metastability happens
3. Be able to design a circuit to reduce the probability that metastability causes system failure
4. To be able to calculate the mean time between failure due to metastability
5. To understand how FIFOs can be used to synchronize signals in a multi-clock domain circuit

Chapter 29 of the Dally textbook (on Connect) talks about Metastability.

Simple Dual-Port Memory: Cyclone II details



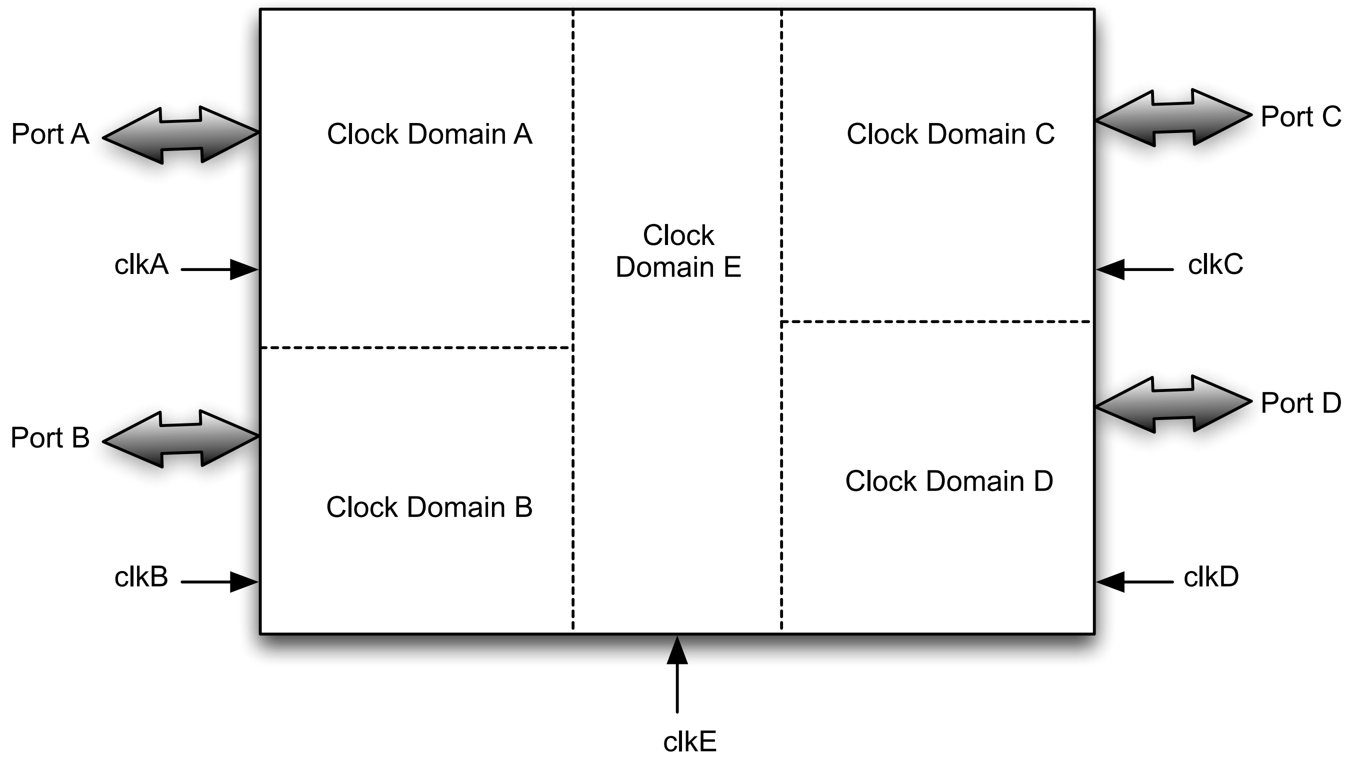
From Altera, Cyclone II handbook

Multiple Clock Domains

In all our examples so far, the entire circuit is clocked on a single clock

In reality, there are often multiple clock domains

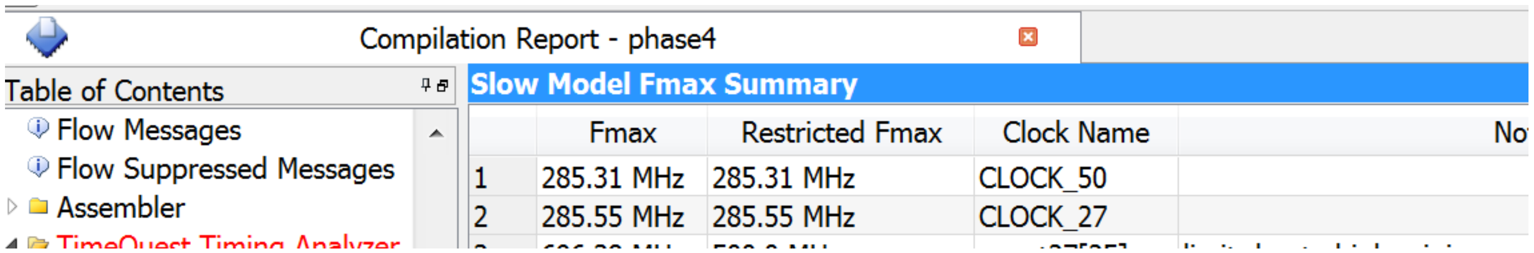
Example switching chip:



All flip-flops in each clock domain are clocked by the same clock

The clocks may be multiples of each other, or they may be entirely independent

There is nothing special you have to do to specify a circuit with multiple clocks. The CAD tool can recognize clock domains by the connections to the flip-flops:



Compilation Report - phase4

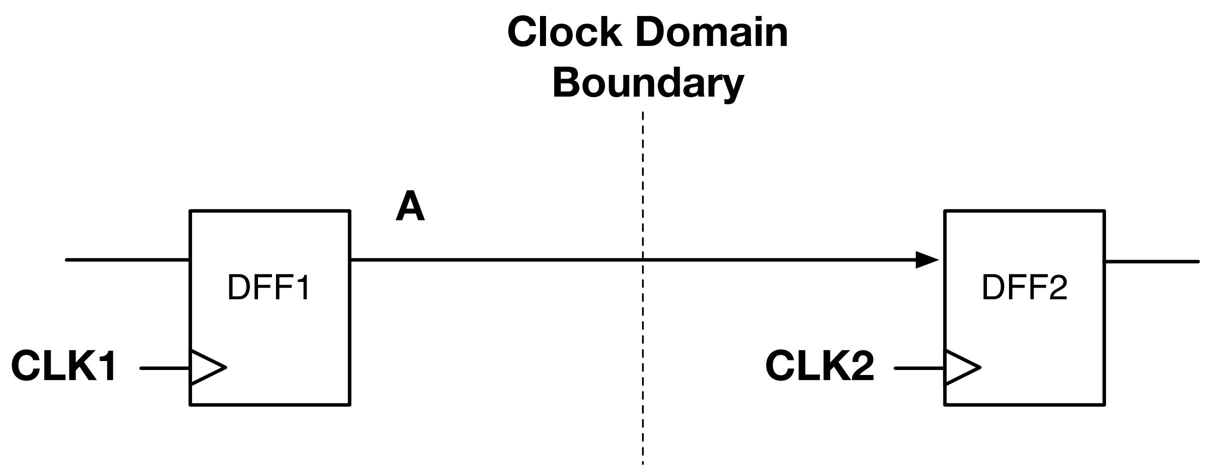
Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	No
1	285.31 MHz	285.31 MHz	CLOCK_50	
2	285.55 MHz	285.55 MHz	CLOCK_27	

Need extra care for signals that cross from one clock domain to another.

Clock Domain Crossings

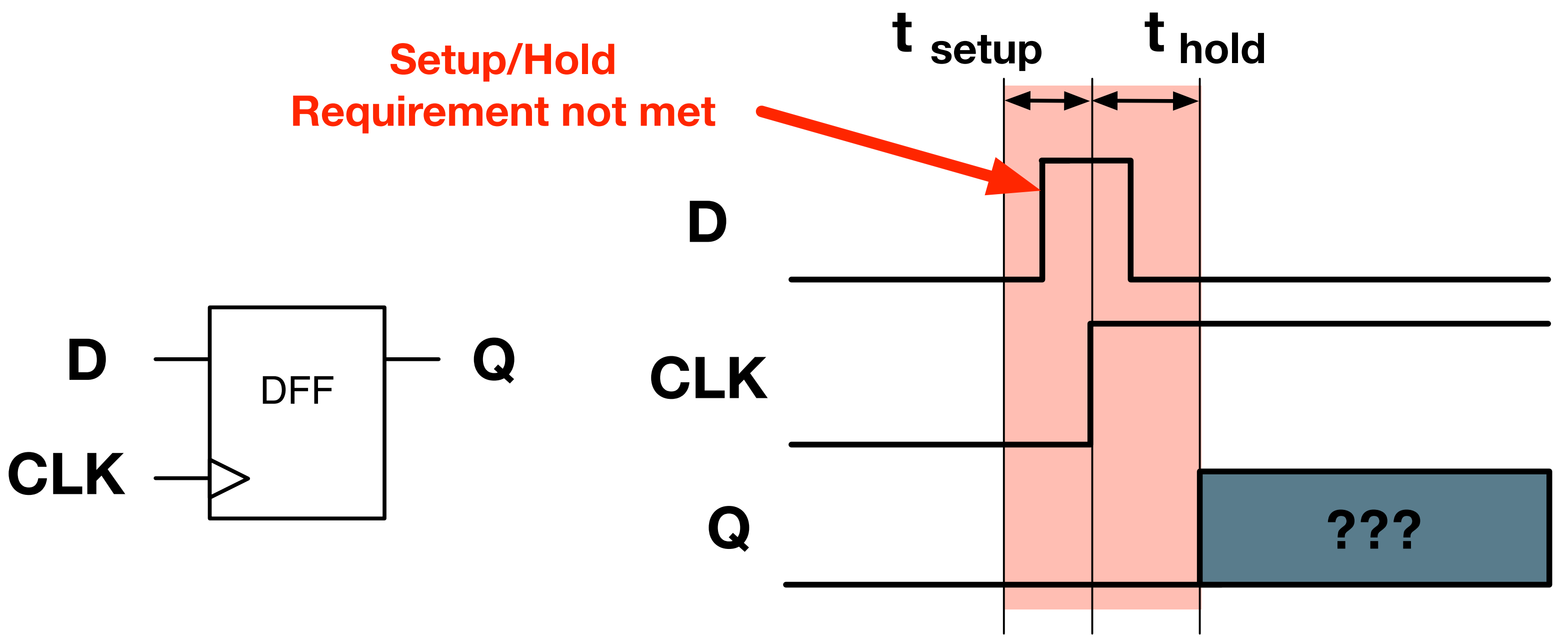
Some signals will cross clock domains.



Signal A appears asynchronous to DFF2

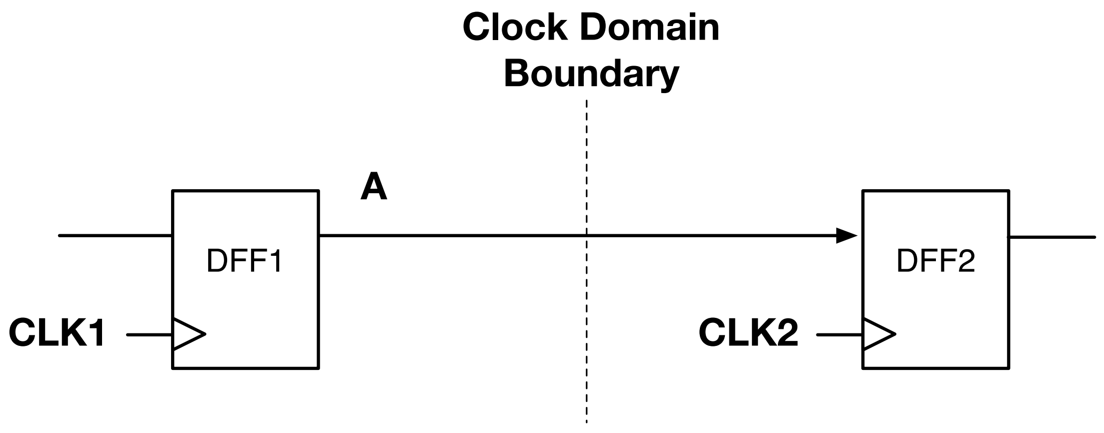
This is a problem. Why?

Recall from earlier, there is a “forbidden zone” when the D input of a flip-flop should not change.



If the clocks are truly asynchronous, there is no way we can guarantee that the clock does not arrive at the second flip-flop in the forbidden zone.

Probability that input to DFF2 will change in the forbidden zone:



Signal A appears asynchronous to DFF2

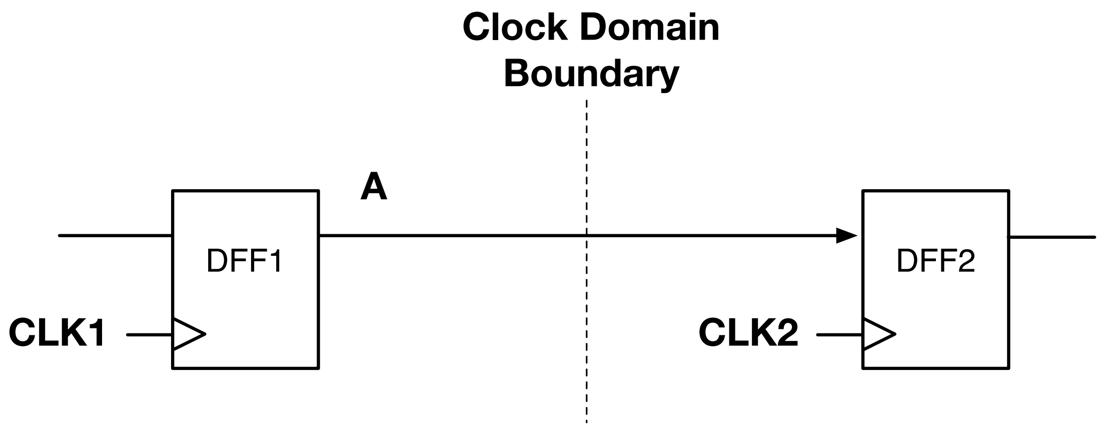
Cycle time of CLK2 = 1/freq(clk2)

Forbidden zone length = $t_{setup} + t_{hold}$

Probability that this might happen each cycle of CLK2:

$$freq(clk2)*(t_{setup} + t_{hold})$$

Probability that input to DFF2 will change in the forbidden zone:



Signal A appears asynchronous to DFF2

Probability that this might happen each cycle of CLK2:

$$\text{freq(clk2)}*(t_{setup} + t_{hold})$$

There are up to freq(clk1) changes of the output of DFF1 every second

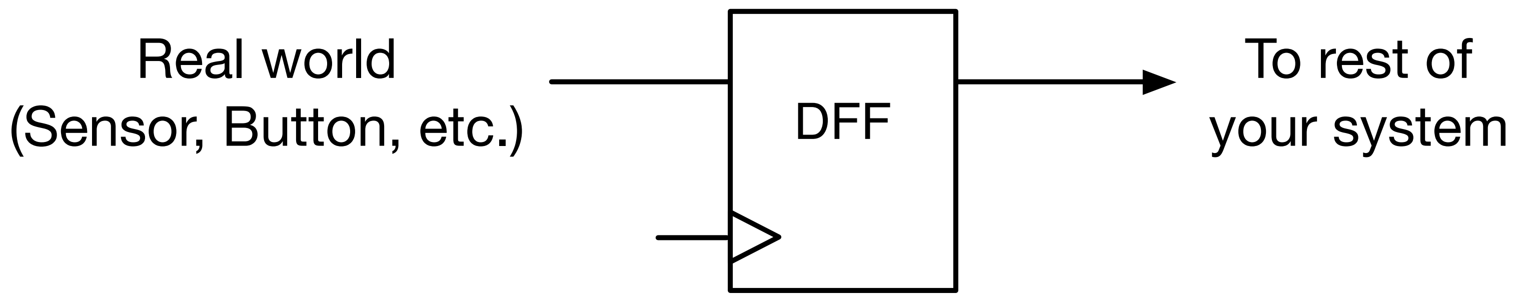
Therefore, number of possible violations of the forbidden zone per sec

$$= \text{freq(clk1)} * \text{freq(clk2)} * (t_{setup} + t_{hold})$$

Asynchronous Signals

A related problem: Real World Signals

- Real world signals, like buttons, don’t change aligned to your clock
- Non-zero chance of input changing too close to clock edge



You can use the same analysis as previous slides to work out the probability of the violation based on the expected number of transitions of the real world signal per second

Important point from previous slides:

In a system with one clock, we can adjust the clock to ensure we never violate the forbidden zone

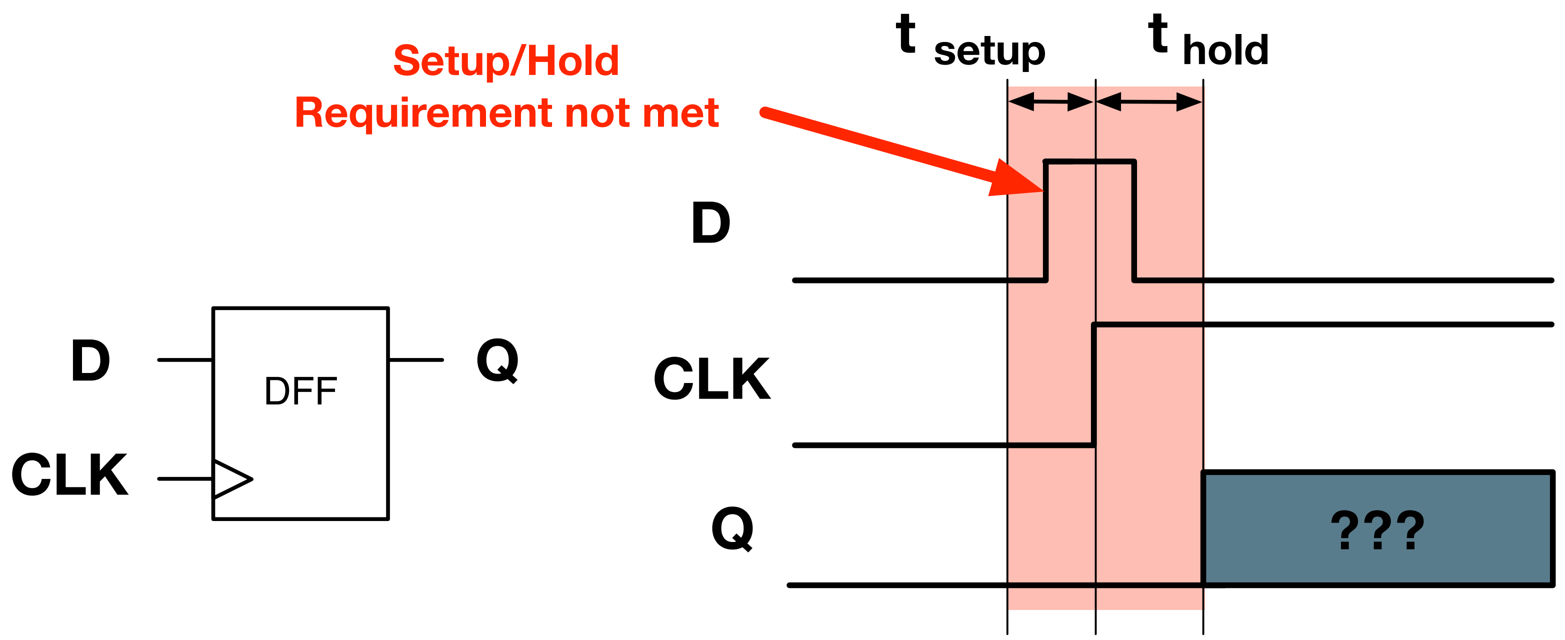
In a system with multiple independent clocks or asynchronous inputs, we can't

- and, violations are going to happen often…

So we better talk about what happens if we change the flip-flop in the forbidden zone…

What happens if we violate a flip-flop’s setup or hold time requirements?

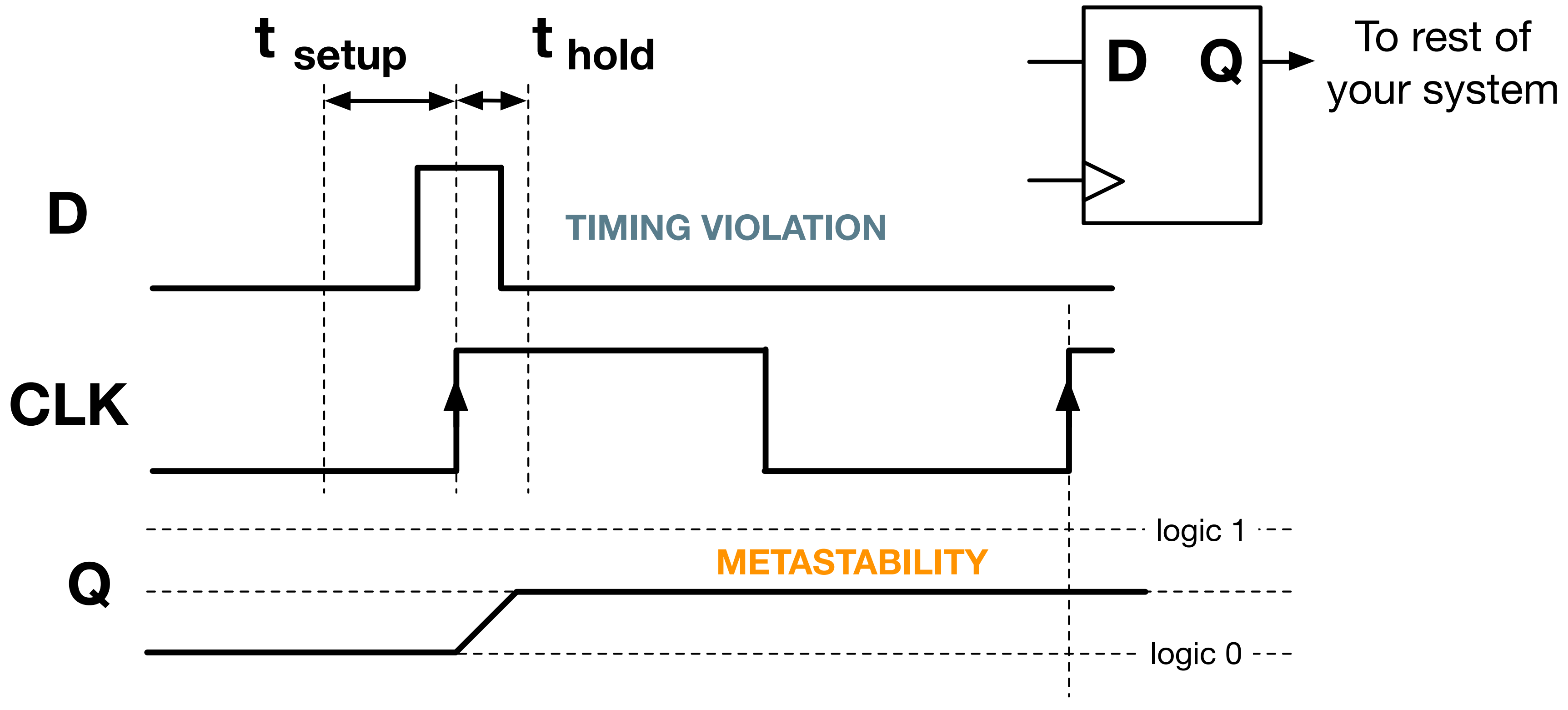
Violating Flip-Flop Setup/Hold Times



Possible Outcomes: Q may…

- get the wrong value (0 or 1)
- get the correct value
- become **metastable** - get a value between 0 and 1
  - Remember, in reality, we are working with voltages which can take on a continuous range of values

Metastability

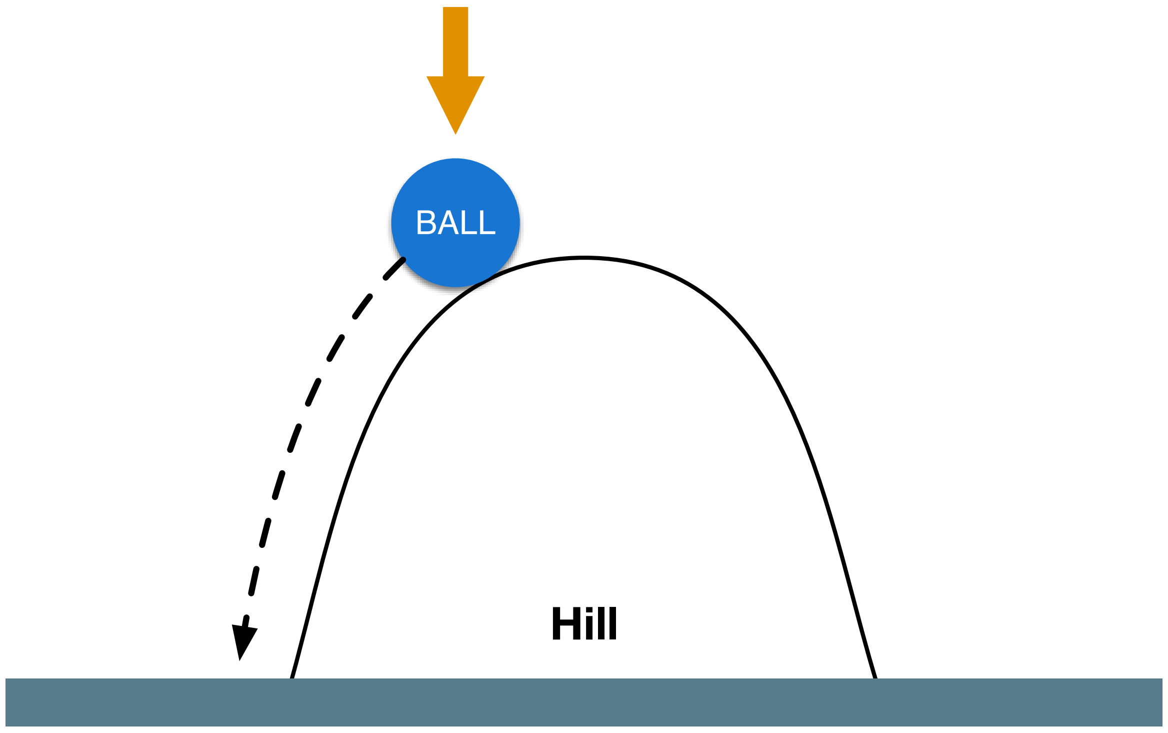


METASTABLE VALUE PROPOGATED TO OTHER PARTS OF SYSTEM

MAY CAUSE SYSTEM-WIDE FAILURE

Why does metastability happen?

Mechanical Analogy



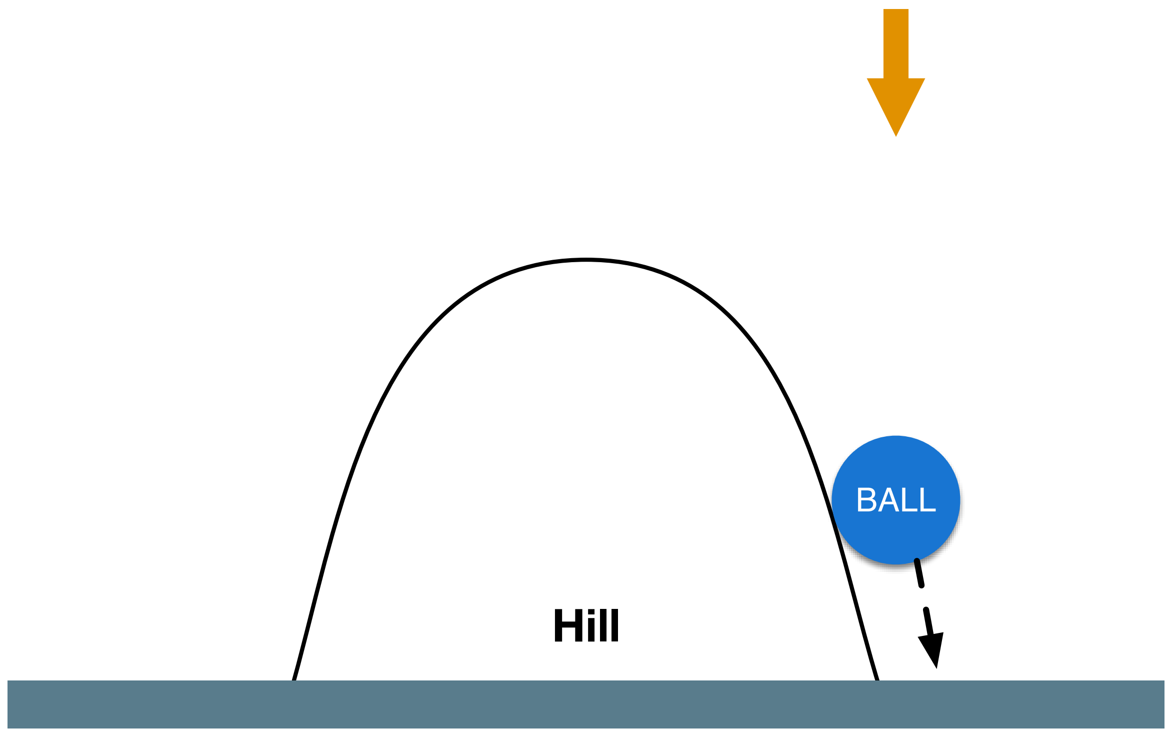
Imagine dropping a ball on a hill

- perfectly smooth and symmetrical hill
- no forces except gravity

Depending on where you drop the ball, it will settle in one of two states

- either on the left side or the right side

Mechanical Analogy

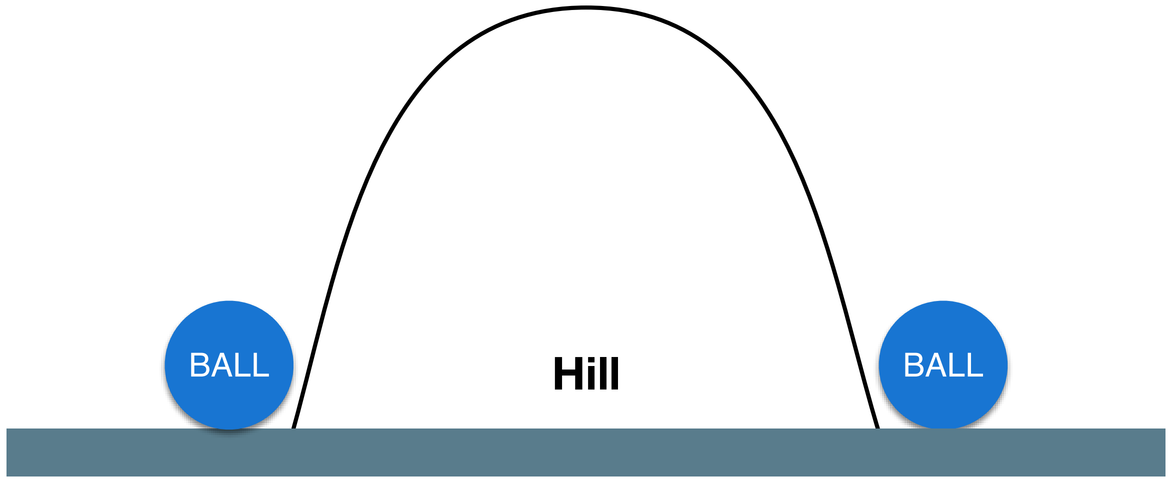


The closer you drop the ball near either side, the faster it settles

- Shorter distance
- Steeper slope

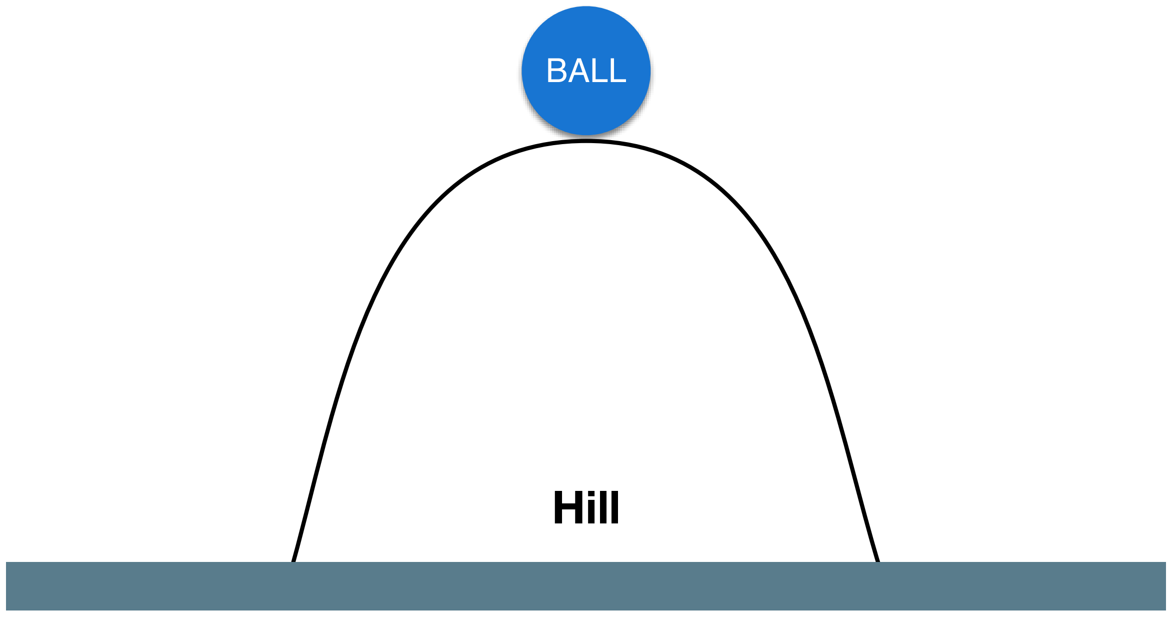
Mechanical Analogy

Two Stable States



Once ball reaches either state, it will stay there forever

There is a THIRD "metastable" state!



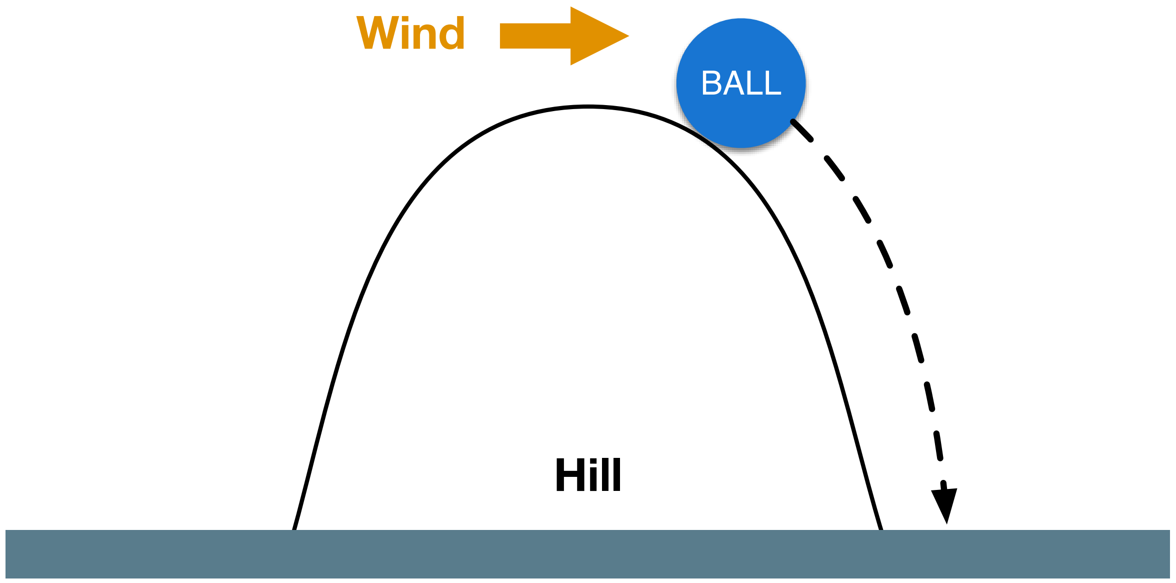
Imagine the ball is dropped perfectly in the middle

The slope at the exact middle is flat

There are no other forces besides gravity

In theory, ball will stay there forever

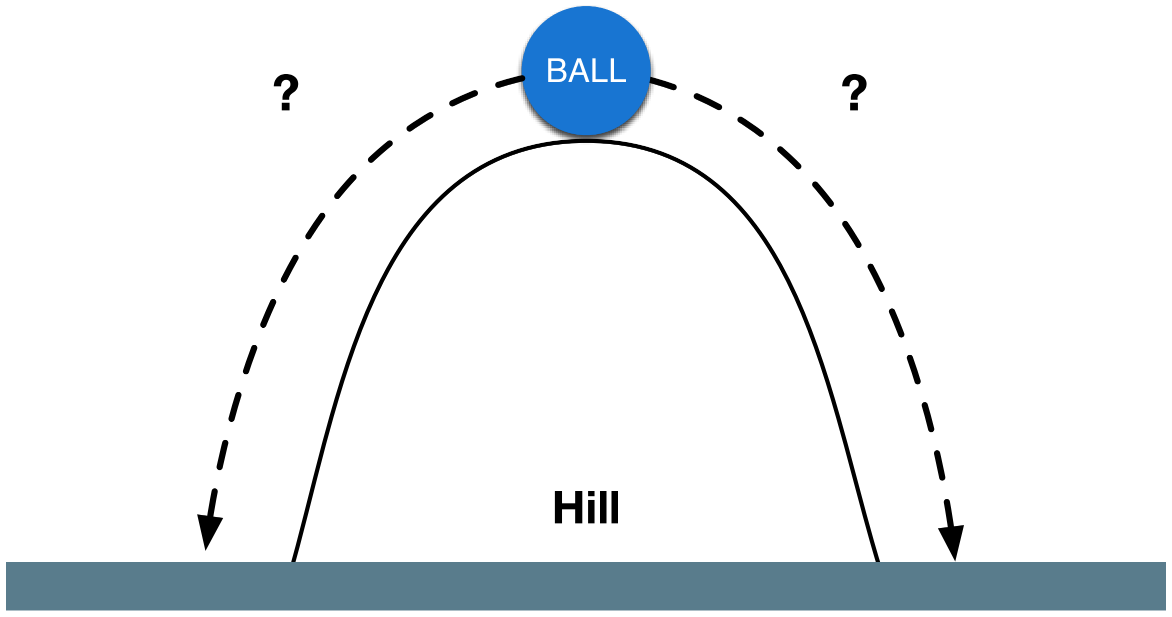
Mechanical Analogy



In practice, ball will eventually settle to a stable state

- Gust of wind
- Non-perfect hill
    - Not perfectly symmetrical
    - Slope not perfectly flat in the middle

Mechanical Analogy



Into which state will it settle?

How long before it settles?

Not predictable!

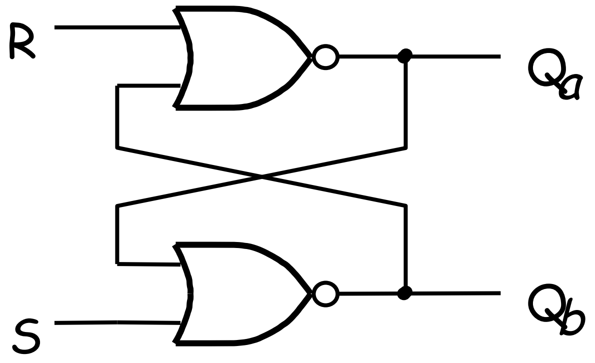
OK. But how does this apply to flip-flops?

Let’s first review what a DFF looks like inside

S-R Latch

We will only operate this under the following conditions:

1. Qa and Qb are always inverse of each other
2. R and S inputs are never 1 at the same time



If R=1, S=0, Qa=0 and Qb=1

If R=0, S=1, Qa=1 and Qb=0

If R=0, S=0, Qa and Qb maintain value

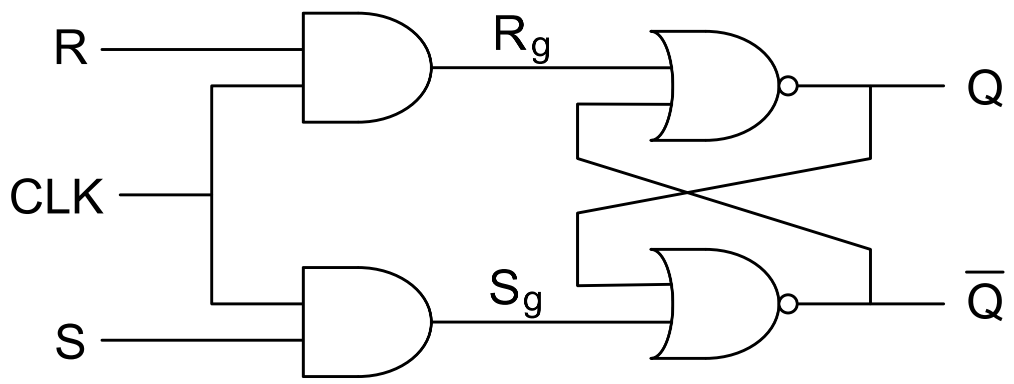
Important point: in an S-R latch:

- If the set signal goes high, the output Qa goes to 1
- If the reset signal goes high, the output Qa goes to 0
- If neither set nor reset are high, the output <u>maintains its value</u>

What about if S and R are high at the same time?

- Would not normally use an S-R latch this way

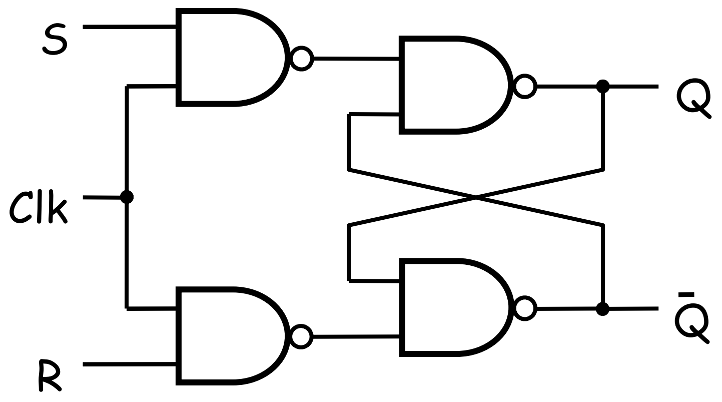
Gated S-R Latch



When clk is high, it operates like a normal S-R Latch
When clk is low, the latch maintains value, regardless of R and S

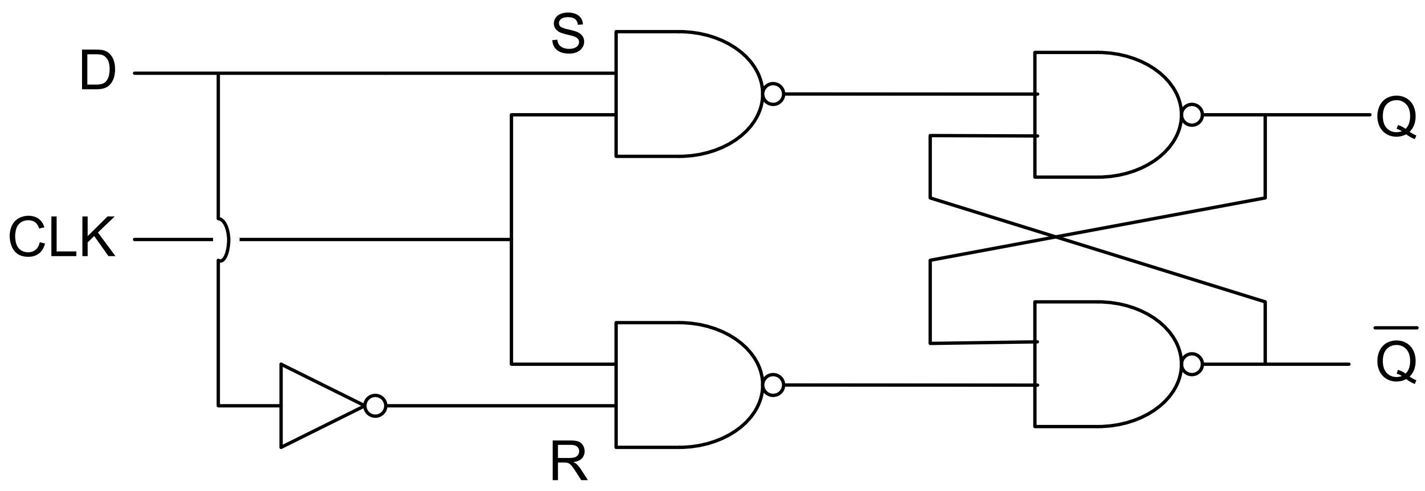
This is called <u>level-sensitive</u>, since the operation depends on the level of the clk signal.

By the way, this has the same function as the Gated S-R latch:



Who cares? It turns out it uses fewer transistors

Gated D Latch



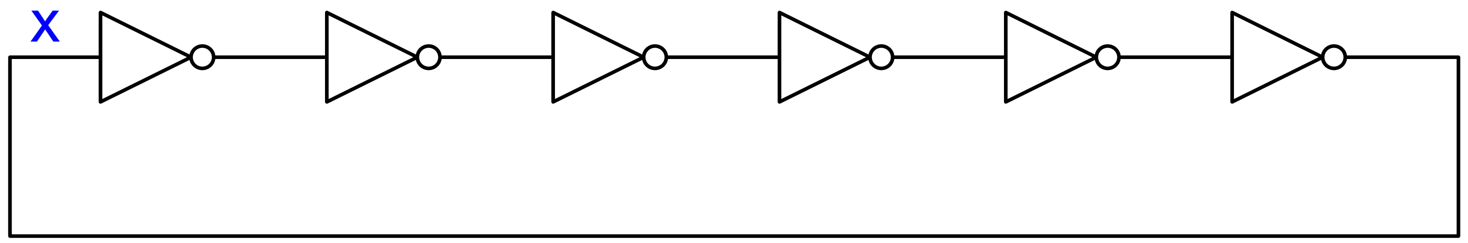
If CLK is high, Q becomes equal to D

If CLK is low, Q maintains its value (regardless of D)

This is level-sensitive

How does metastability happen?

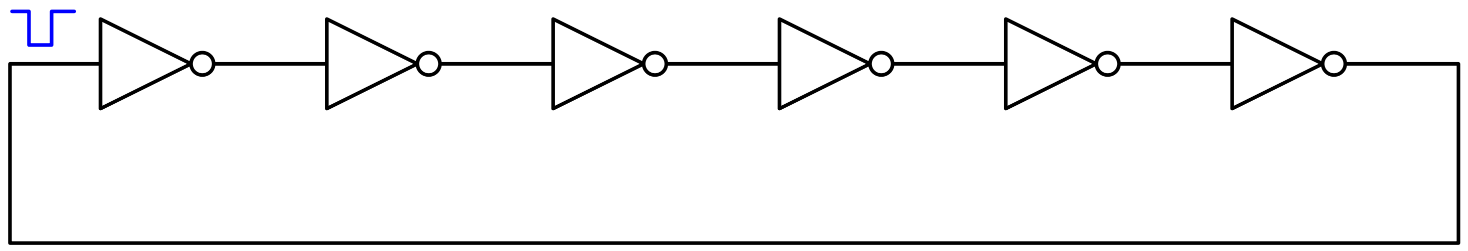
Before answering this question, consider:



Two steady states: x=0 or x=1

How does metastability happen?

Now put in a short pulse at x (about one gate delay long)



Pulse races around ring, causing oscillation

-Theoretically, it could oscillate forever

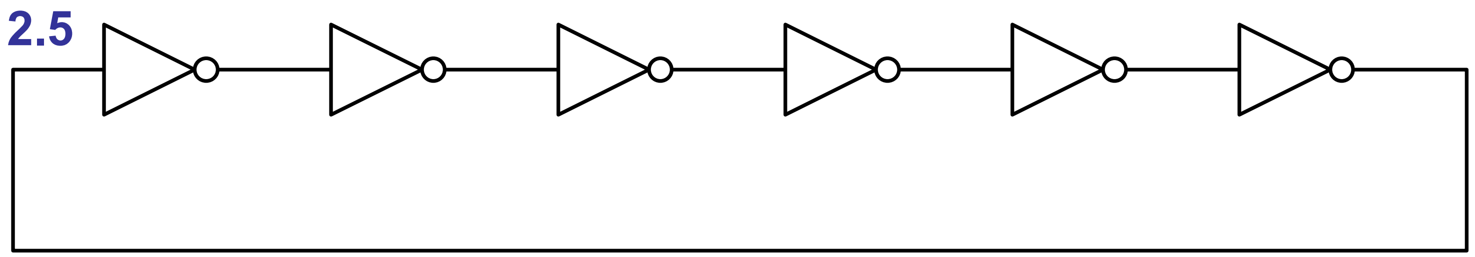
-Reality: eventually, the pulse will get narrower and narrower

- After some time, it is shrinks to nothing and stops oscillating

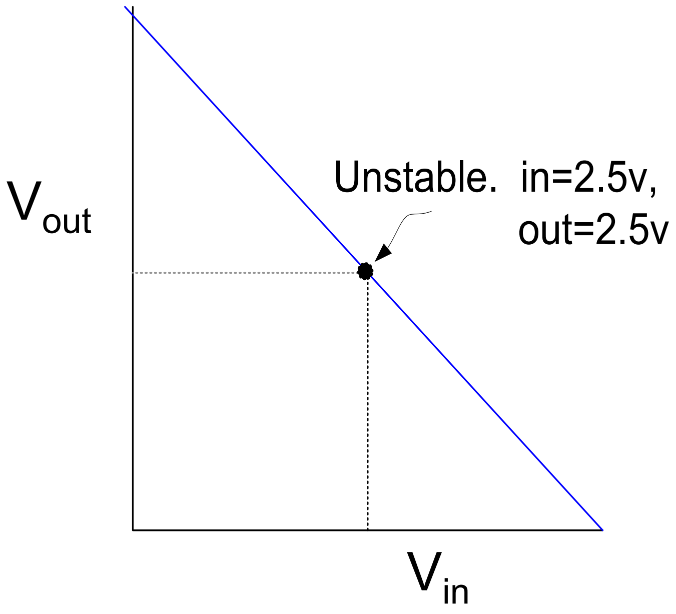
-NB: the pulse could also get wider and wider, but the effect is the same

How does metastability happen?

Now consider this: what if we put in 2.5 volts (assume a 5 volt system?)

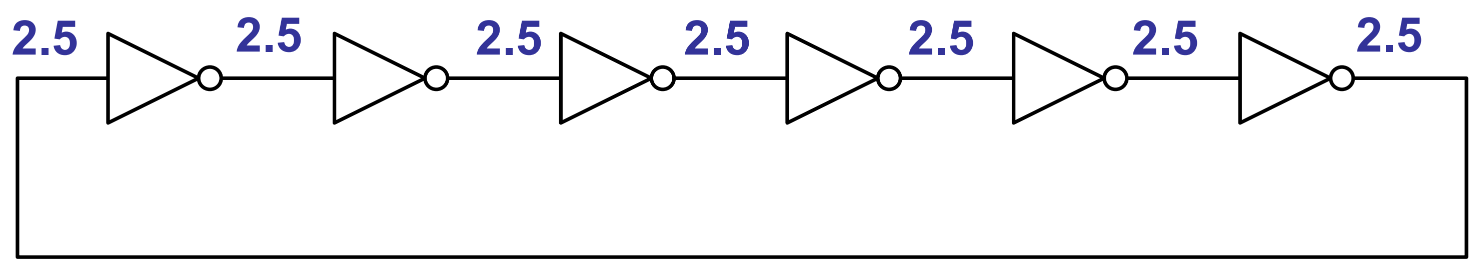


Remember the transfer function of an inverter:



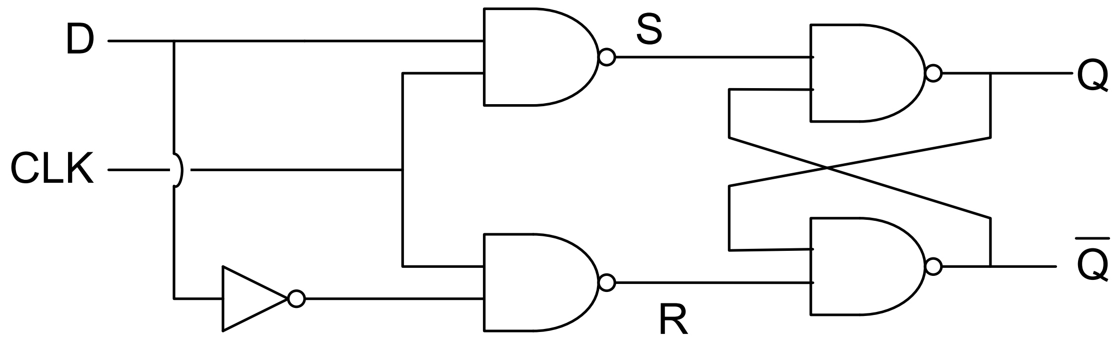
How does metastability happen?

So ideally, we might get:

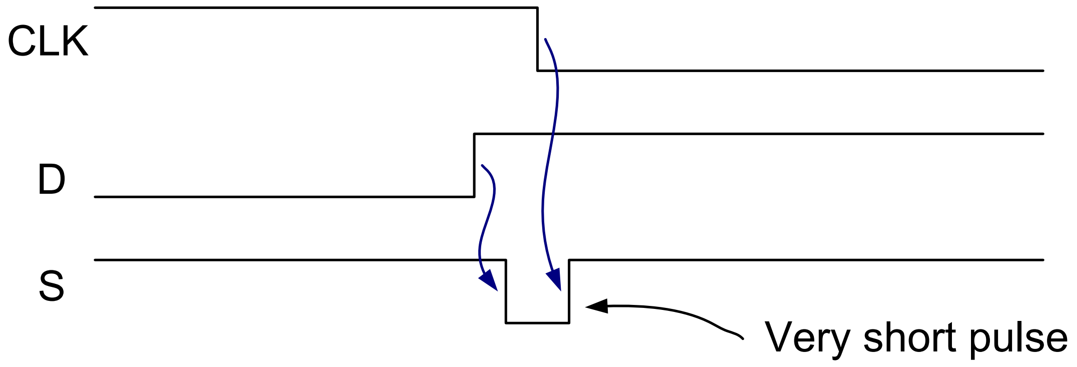


Of course, eventually noise will knock it one way or the other

Now go back to a flip-flop:



Case 1: If D changes close to CLK:

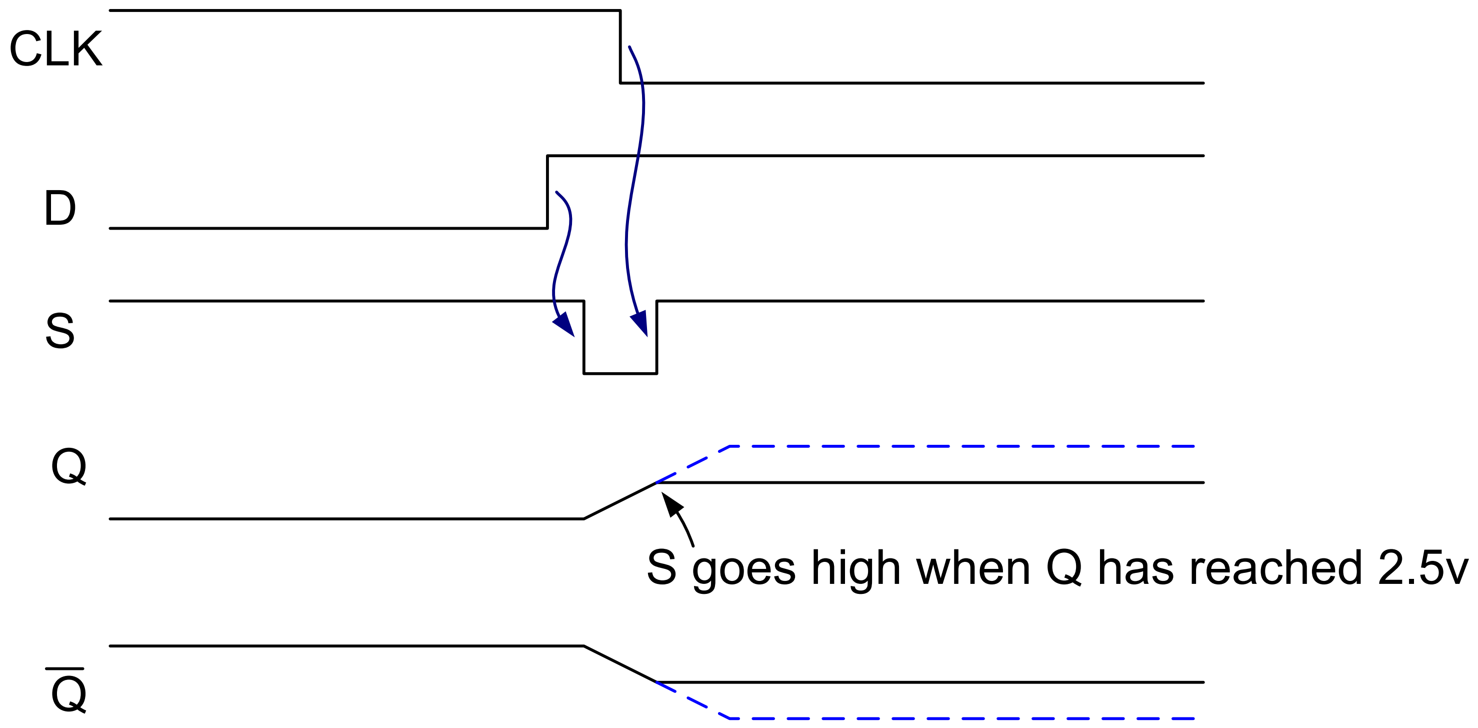


This short pulse races around the ring of two cross-coupled NAND gates, causing oscillation

-> Oscillates forever (theoretically)

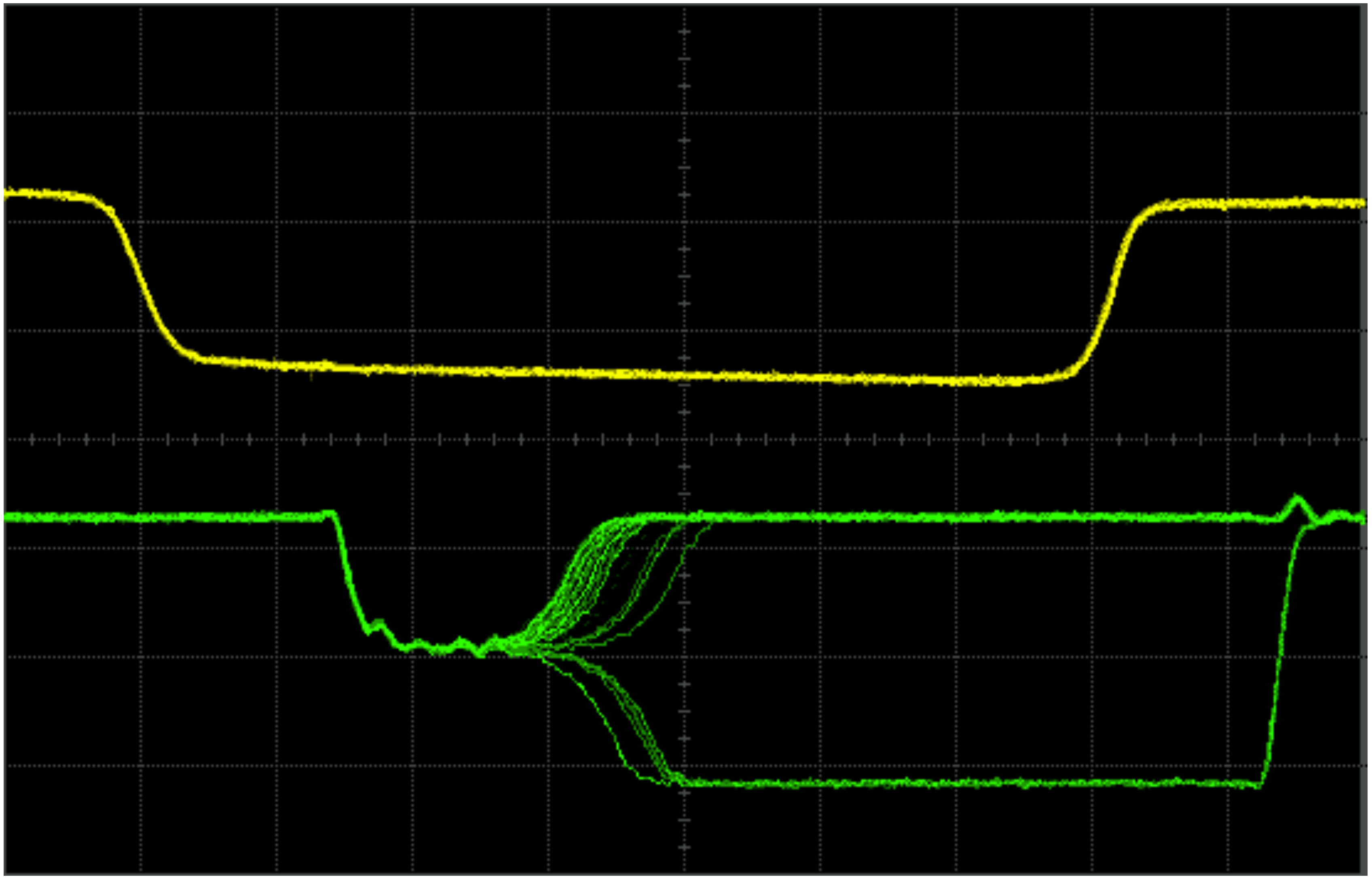
:

Case 2: If rise time of Q is very slow:

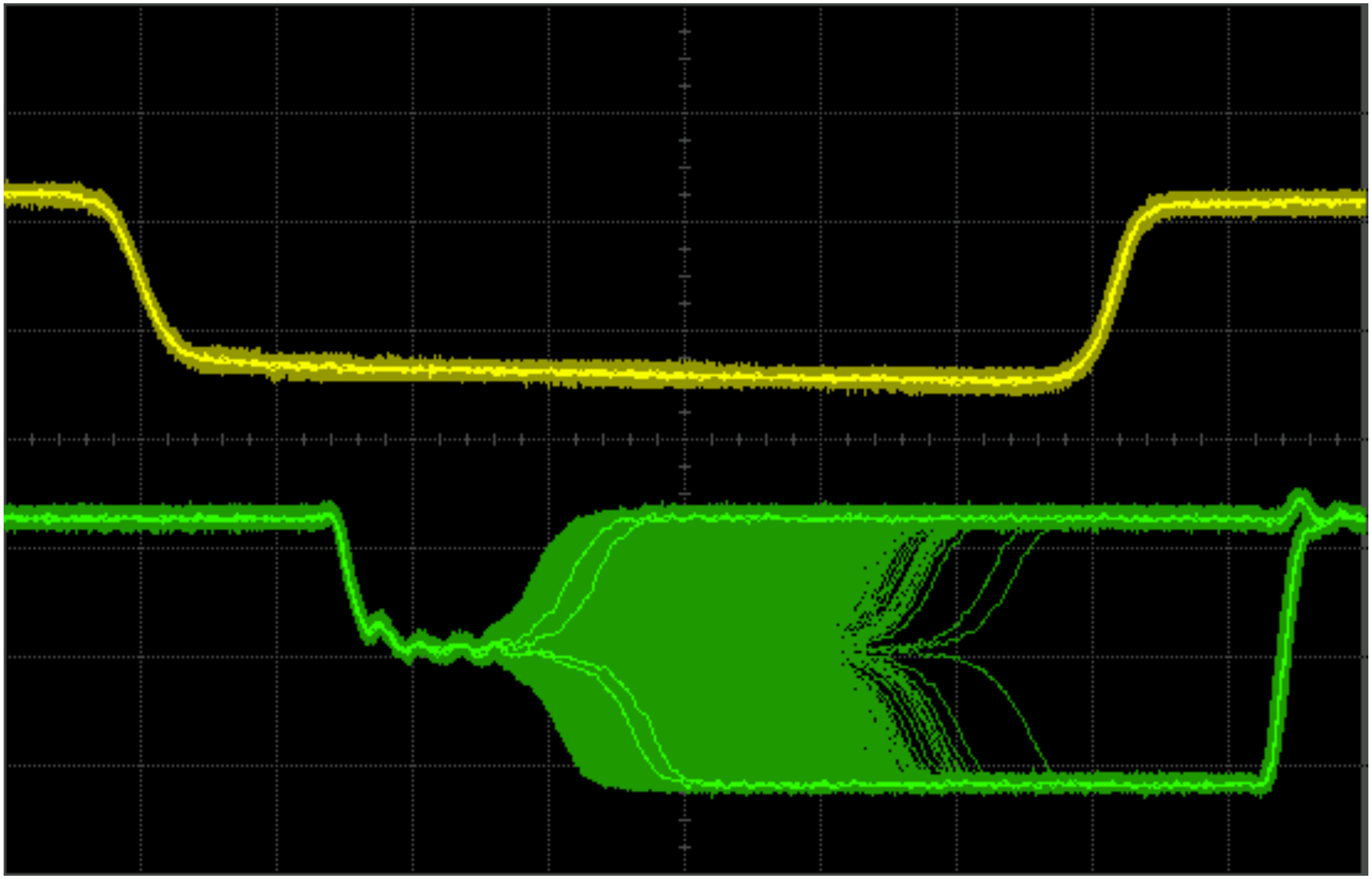


If timing is exactly "right", Q and Q-bar could be left at 2.5 volts
-> Output would remain at 2.5 volts (theoretically forever!)

Real Metastable Example

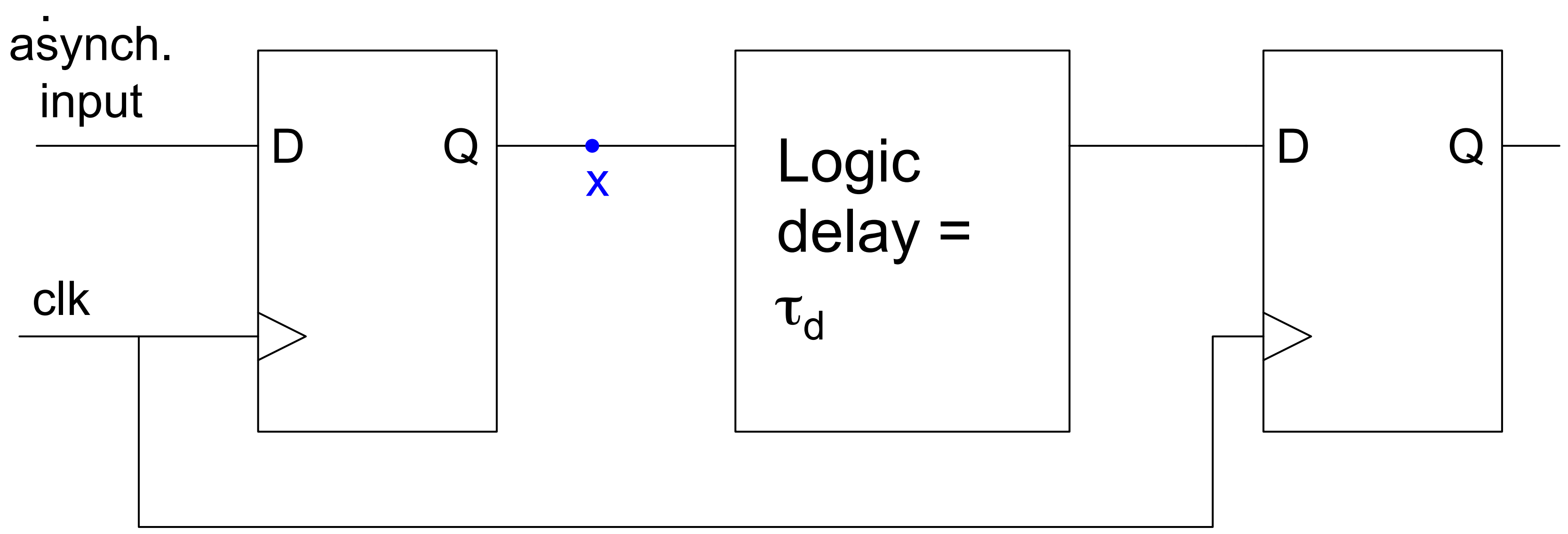

Picture taken from Dally textbook on Connect

Real Metastable Example



Picture taken from Dally textbook on Connect

How can metastability cause failure?

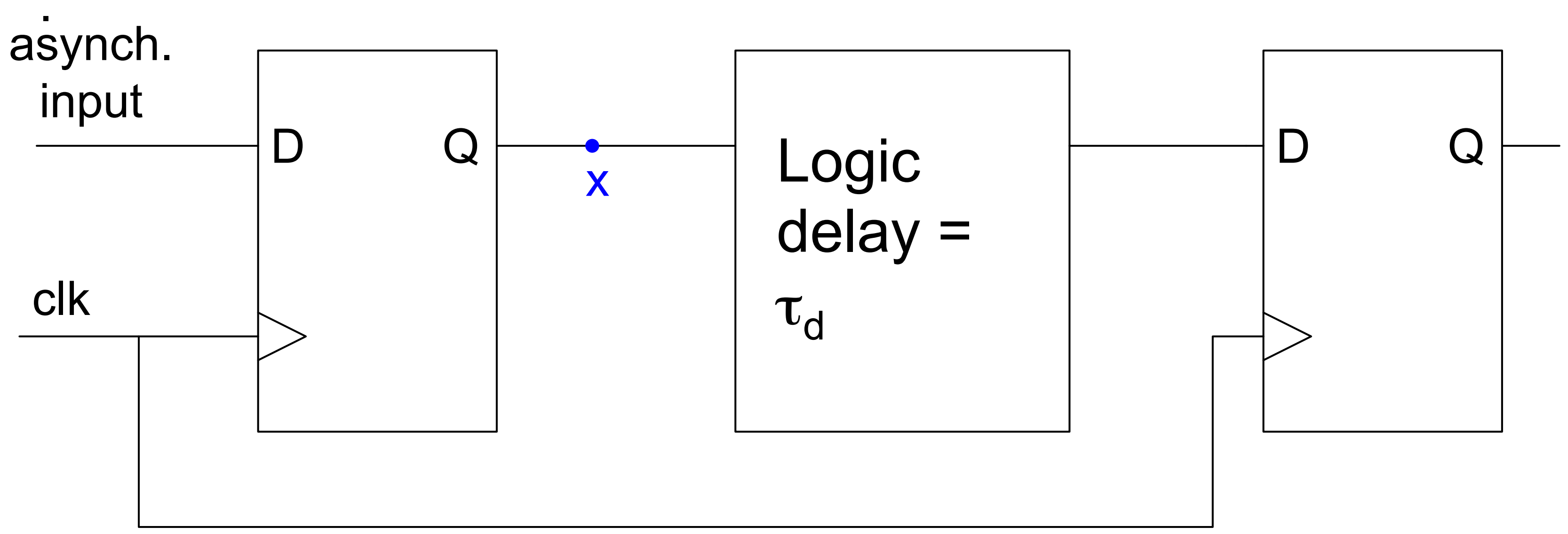


Definition:

$$t_{slack} = \tau_p - (t_{clktoq} + \tau_d + t_{setup})$$

This is a measure of how tight the timing is on this path
(think of it as the amount of "extra delay" that could be added to the path before we fail to meet timing)

How can metastability cause failure?



There is a finite chance node x will go metastable. If it does, denote the time until node x stops oscillating or settles to 0 or 1 as τ_m

If $\tau_m > t_{slack}$ then system failure!

Important point: the more slack there is, the more time there is for metastability to resolve itself

Mean Time Between Failure (MTBF)

Metric to estimate the average time between two failure-causing instances of metastability on a given signal transfer

$$MTBF\left(t_{slack}\right) = \frac{e^{t_{slack}/C_0}}{C_1 \cdot f_{CLK} \cdot f_{DATA}}$$

- f_{CLK} – Clock frequency
- f_{DATA} – Toggling frequency of the FF input
- t_{slack} – Amount of slack available on the path for metastability resolution
- C_o, C_1 – Constants dependent on operating conditions and process tech

Mean Time Between Failure (MTBF)

Metric to estimate the average time between two failure-causing instances of metastability on a given signal transfer

$$MTBF\left(t_{slack}\right) = \frac{e^{t_{slack}/C_0}}{C_1 \cdot f_{CLK} \cdot f_{DATA}}$$

Higher MTBF means more robust design

- maybe hundreds or thousands of years

Required MTBF depends on application

- Life-critical medical equipment would need higher MTBF than consumer video game system

MTBF Example Calculation - Synchronizer

Example without much slack:

$$f_{CLK} = 100\text{MHz}$$
$$f_{DATA} = 1\text{MHz}$$
$$t_{slack} = 0.5\text{ns}$$
$$C_o = 0.31\text{ns}$$
$$C_1 = 9.6 * 10^{-18}\text{ s}$$

MTBF = 1.53 hours

Example with more slack:

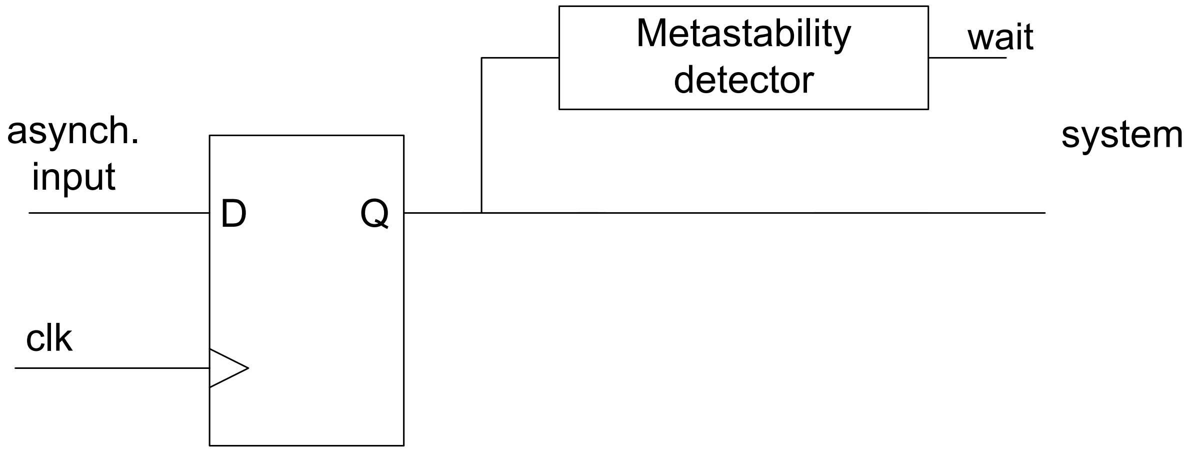
$$f_{CLK} = 100\text{MHz}$$
$$f_{DATA} = 1\text{MHz}$$
$$t_{slack} = 3\text{ns}$$
$$C_o = 0.31\text{ns}$$
$$C_1 = 9.6 * 10^{-18}\text{ s}$$

MTBF = 12 years

Dealing with Metastability:

Possible options:

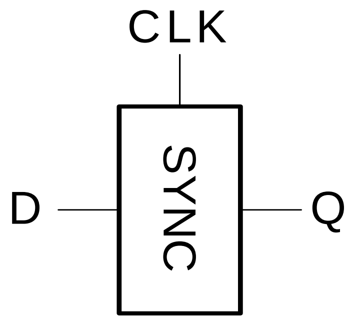
1. Eliminate possibility of metastability
   - Use only asynchronous circuits (usually not feasible)
2. Handle metastability when it occurs:



- If Q goes metastable, "wait" signal causes system to wait until metastability resolves itself

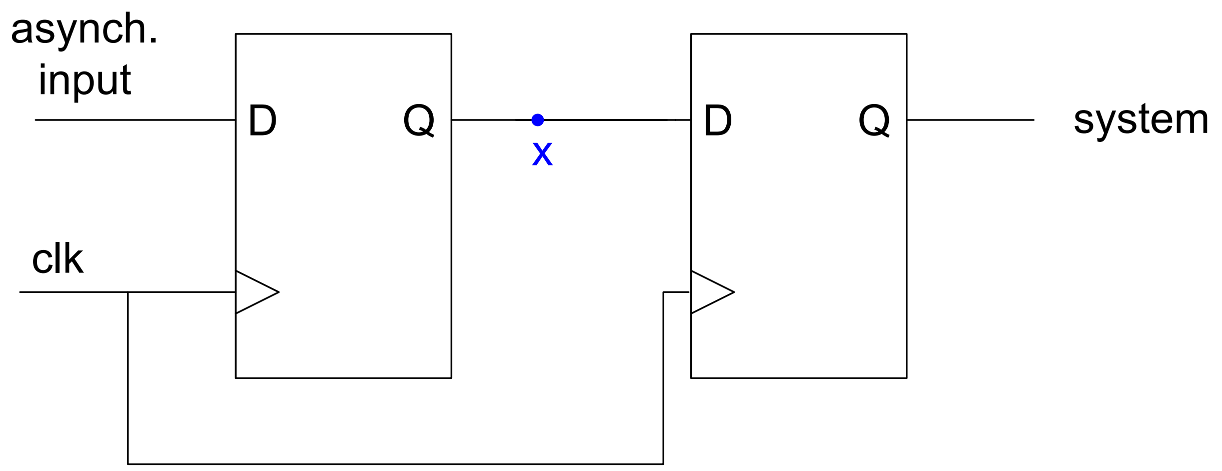
Synchronizers

- Asynchronous inputs (*D*) are inevitable (user interfaces, systems with different clocks interacting, etc.).
- The goal of a synchronizer is to make the probability of failure (the output *Q* still being metastable) low.
- A synchronizer cannot make the probability of failure 0.



Dealing with Metastability:

3. Reduce the probability that metastability would cause system failure:

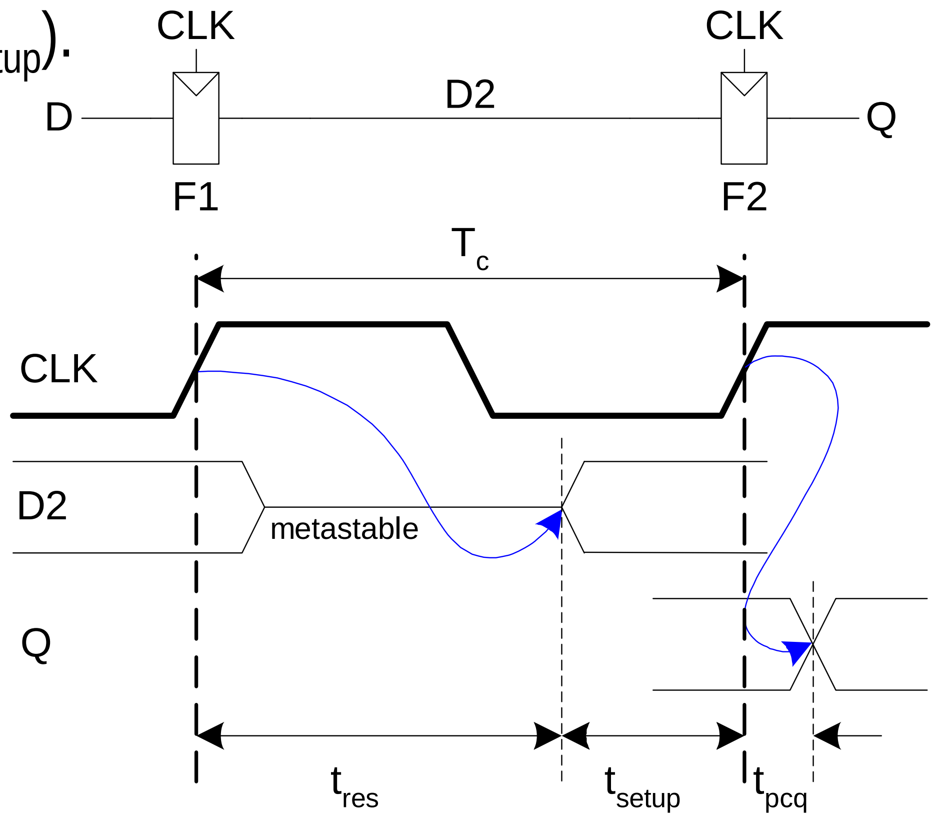


Double-Register each asynchronous input

- If X goes metastable, it has an entire clock cycle to resolve itself
- This technique is very common

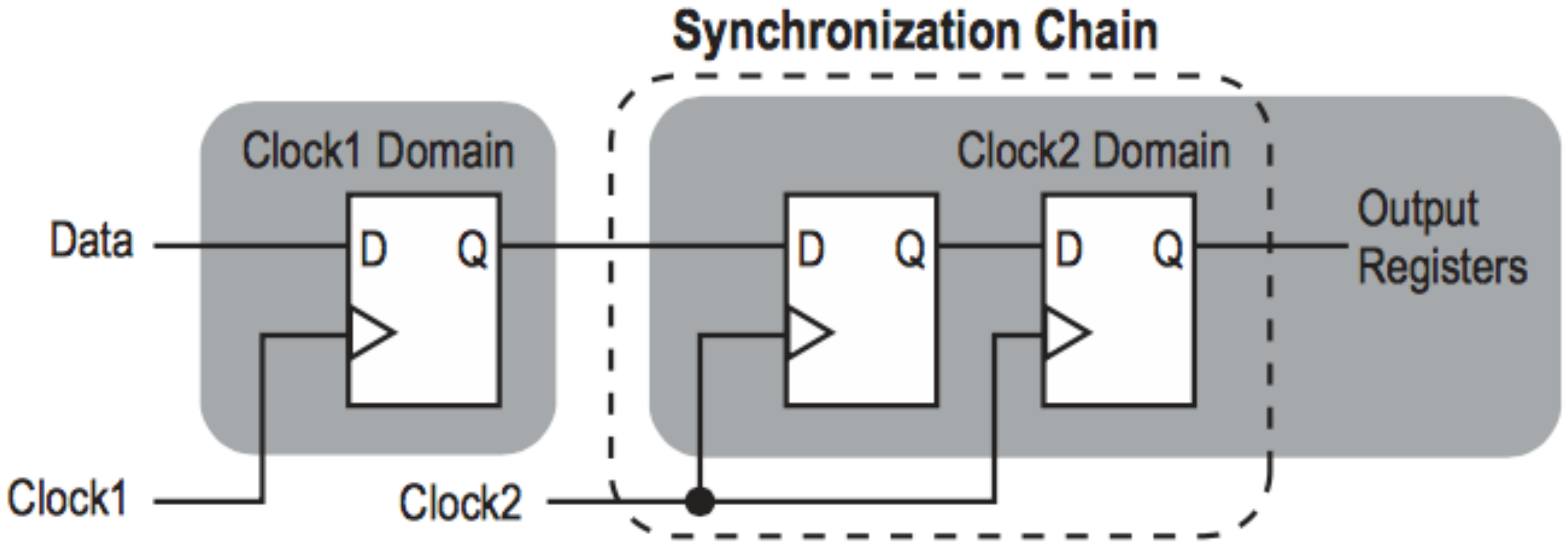
Synchronizer Internals

- A synchronizer can be built with two back-to-back flip-flops.
- Suppose the input D is transitioning when it is sampled by flip-flop 1, F1.
- The amount of time the internal signal D2 can resolve to a 1 or 0 is (T_c - t_{setup}).



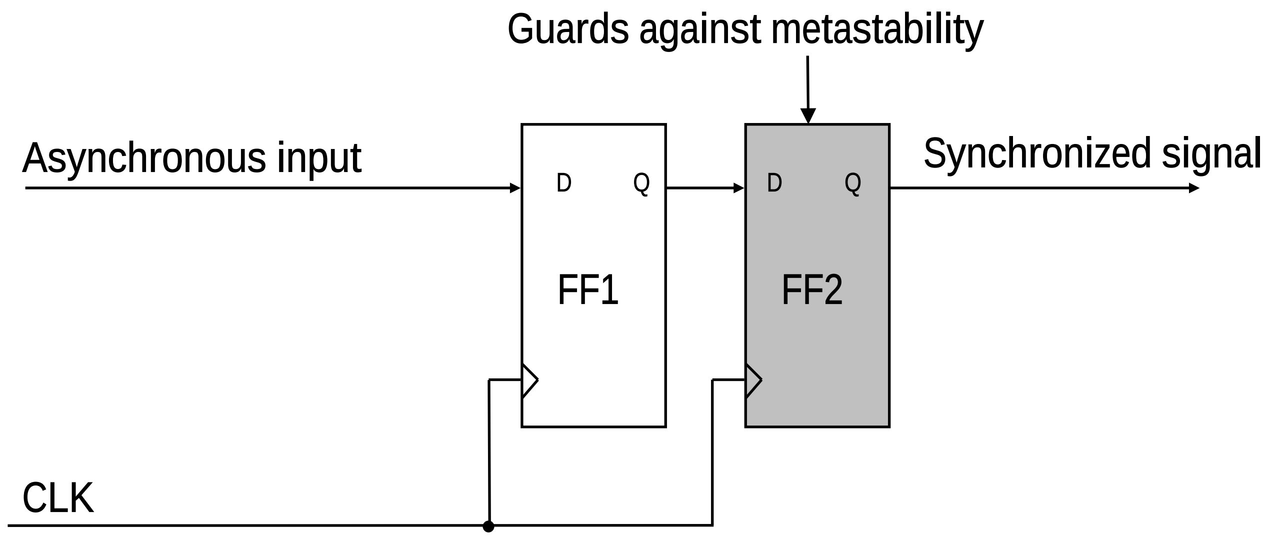
Metastability

- Flip-flop output enters a transitory state
  - Neither a valid 0 nor a valid 1
    - Can be interpreted as 0 by some loads and as 1 by others
  - Remains in this state for an unpredictable length of time before settling to a valid 0 or 1
- Due to a statistical nature, the occurrence of metastable events can only be reduced, not eliminated
- Mean Time Between Failure (MTBF) is exponentially related to the length of time the flip-flop is given to recover
  - A few extra ns of recovery time can dramatically reduce the chances of a metastable event


Synchronization Chain
Clock1 Domain
Clock2 Domain
Data
D
Q
D
Q
D
Q
Output Registers
Clock1
Clock2

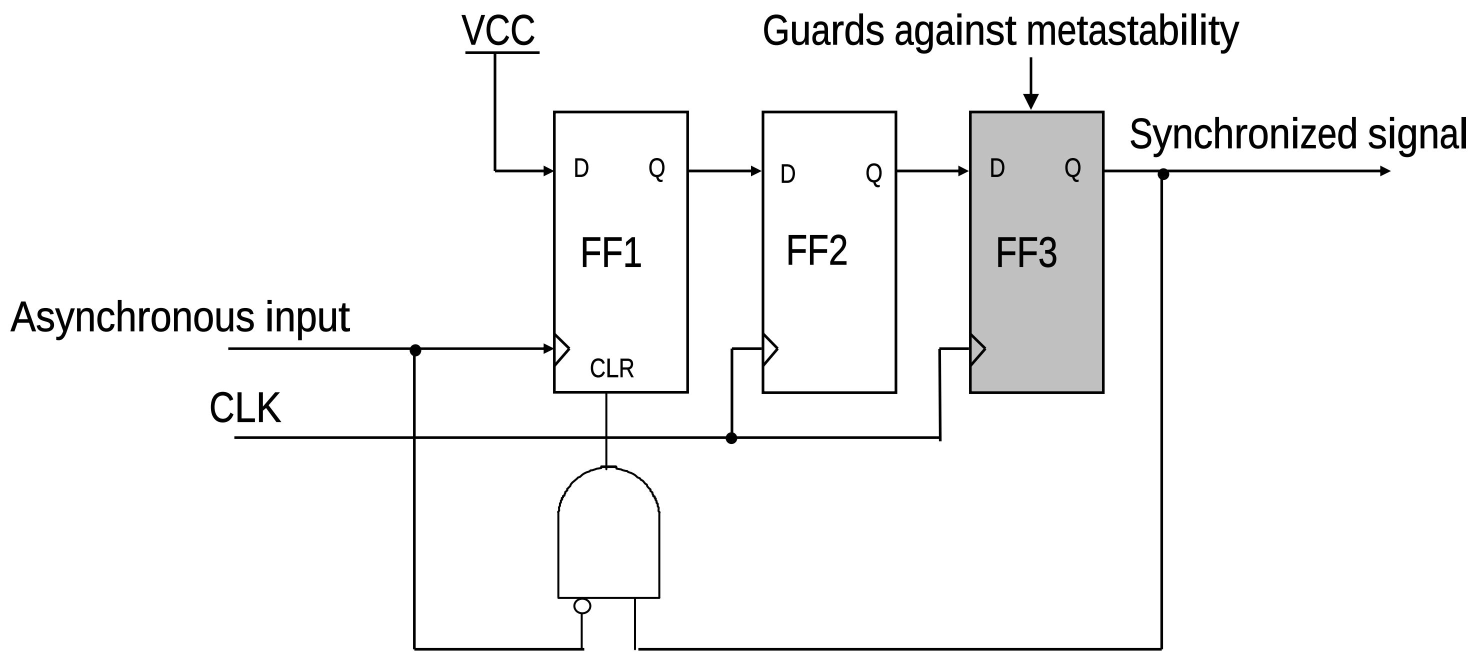
Synchronization Circuit 1

- Use when input pulses will always be at least one clock period wide
- The "extra" flip-flops guard against metastability



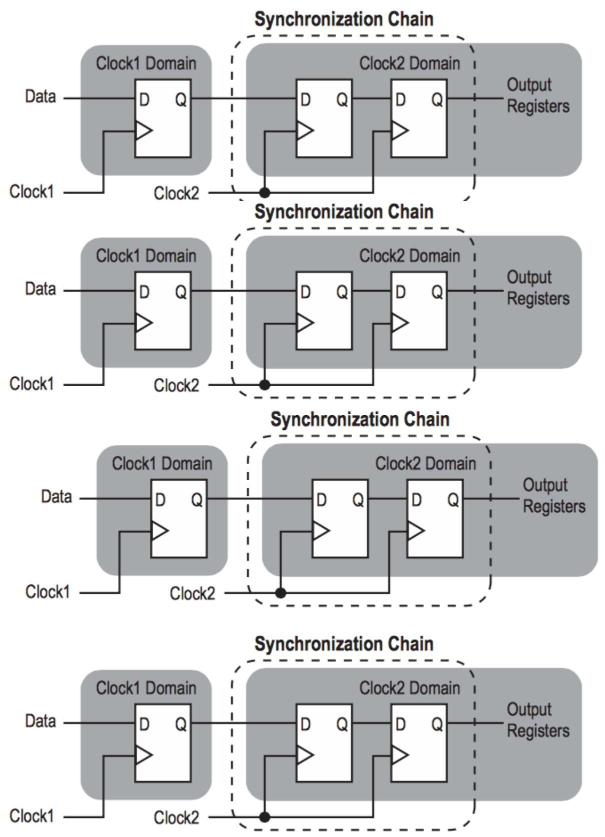
Synchronization Circuit 2

- Use when input pulses may be less than one clock period wide
  - FF1 captures short pulses



Transmitting Multi-bit Words

Transmitting multi-bit words (buses) between timing domains is also troubling. Suppose we have a four bit word. We could use one synchronizer on each bit:



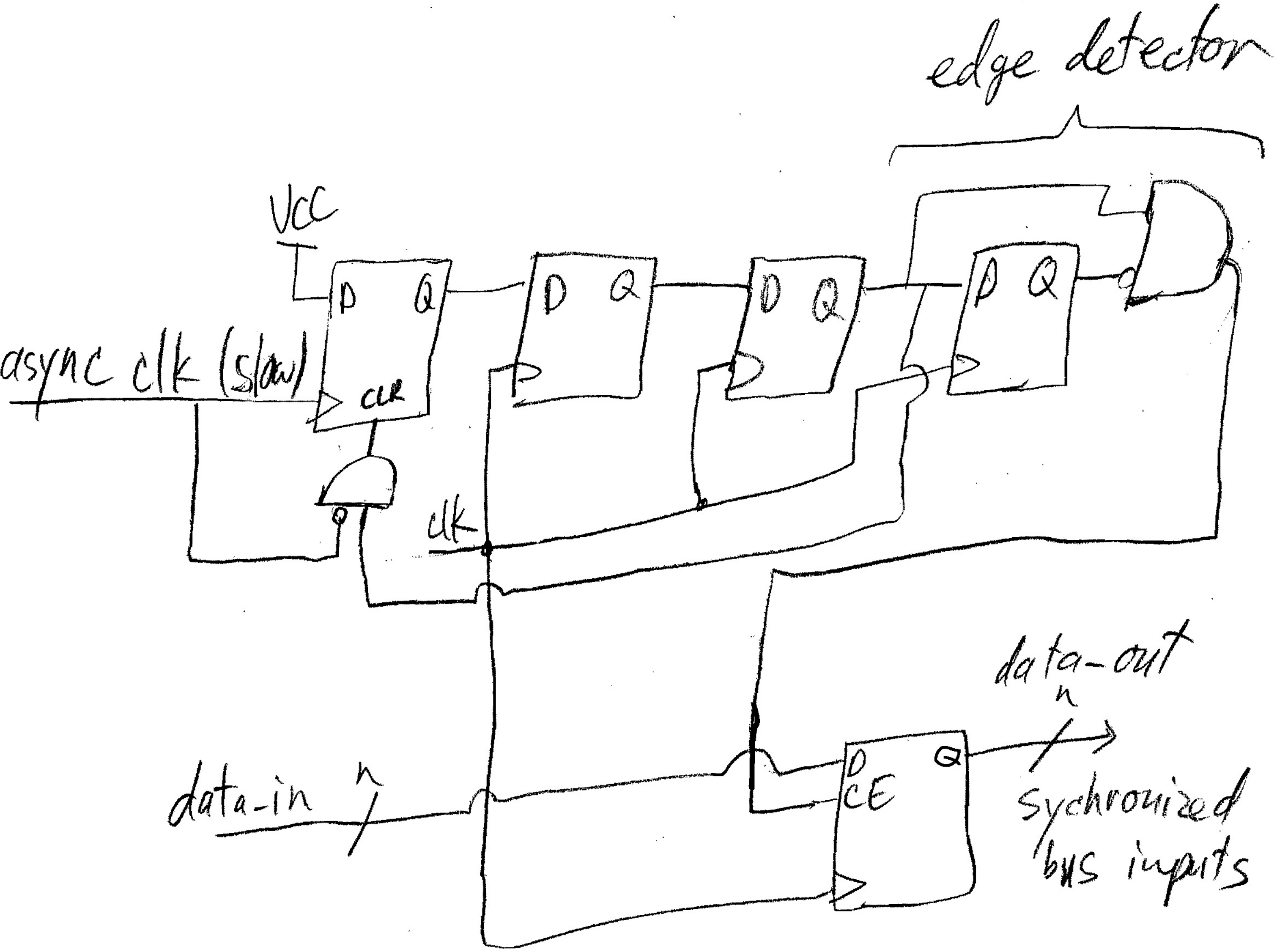
Suppose each bit has a slightly different delay. In this case, if the change is close to the receiving clock edge, some bits may take their old value and some may take their new value

Previous data item: 0 1 1 0
Next data item: 1 0 1 1

Suppose the MSB is slower than the others. It may be that the MSB arrives just after the receiving clock edge, while the other bits arrive just before. So you end up receiving something like:

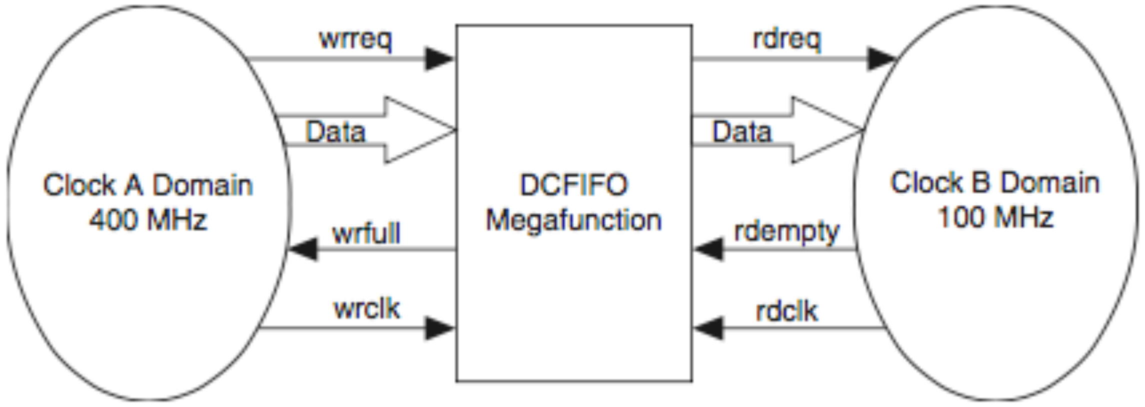
0 0 1 1   instead of 1 0 1 1

It would be resolved in the next cycle, but for one cycle, you are operating on an invalid word. May lead to system failure if your receiver is not expecting it.


edge detector
VCC
D
Q
async clk (slow)
CLR
D
Q
D
Q
D
Q
clk
data-out
n
D
Q
CE
data-in
n
sychronized
bus inputs

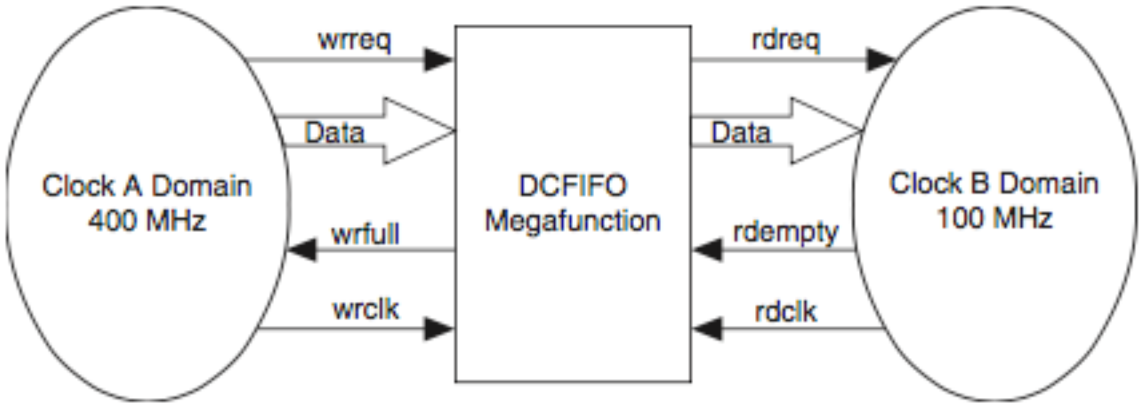
This can be resolved if the data is always in “Gray code” (at most one bit changes per cycle). But this is not always feasible.

Solution: Use a FIFO. The write side of the FIFO is clocked by the sender, and the read side is clocked by the receiver.



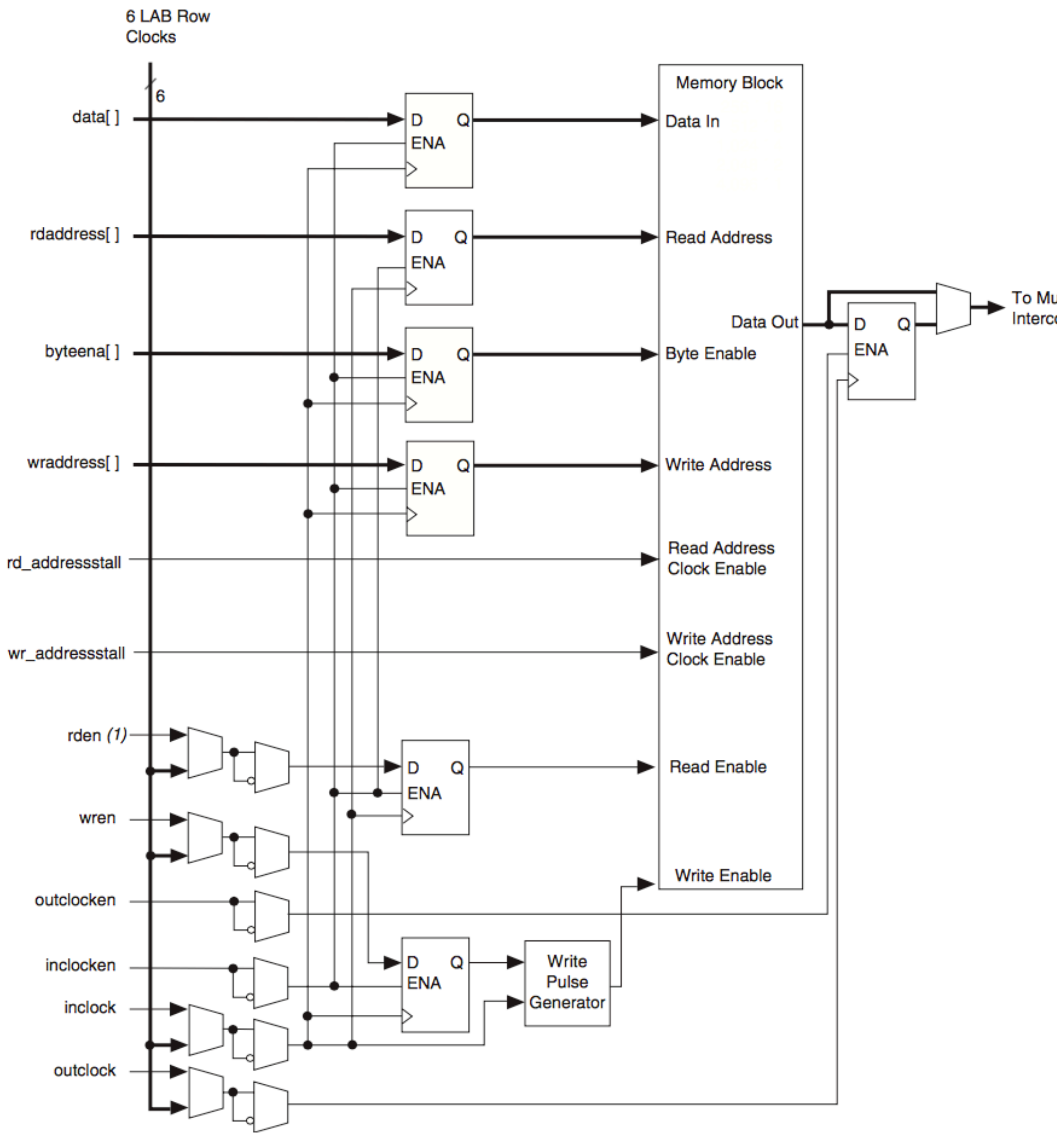
In this course, we will not go into the details of what is inside the FIFO, but if you want to understand how the FIFO works inside, see Chapter 29 of the Dally book).

Another advantage of a FIFO: flow control



If the producer and consumer don't generate/produce data at the same rate, you need flow control (as in the handshaking in Lab 5).

In our device, can implement FIFO using an embedded RAM



From Altera, Cyclone II handbook

Learning Objectives

1. Understand what metastability is, and how it can cause failure
2. Understand why metastability happens
3. Be able to design a circuit to reduce the probability that metastability causes system failure
4. To be able to calculate the mean time between failure due to metastability
5. To understand how FIFOs can be used to synchronize signals in a multi-clock domain circuit