6.205
(aka 6.111)

Sequential Logic II

Let’s make a Combination Lock

- 16-bit binary value entry:
    - Specify value with sw[0]
    - Push BTNU to enter value into shift buffer at lsb
- After 16 pushes check if entry matches code. If true, Make a beaver signal (turn on a buzzer)
- Default code: 16’hF9AB

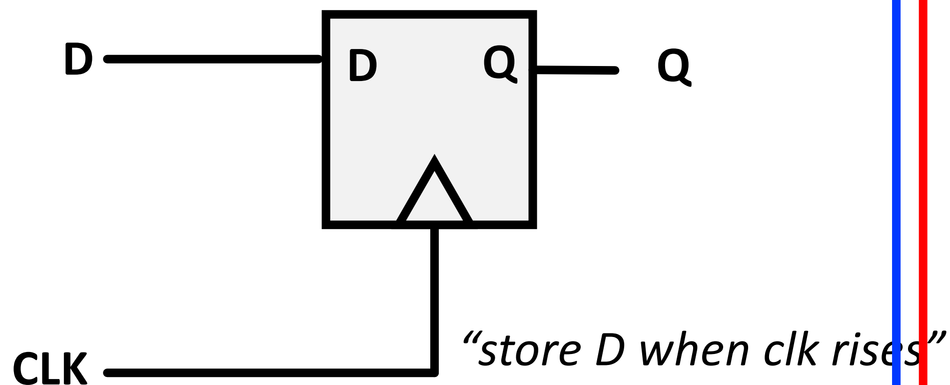
Registers, Latches, and Flip-Flops

- The terminology is a mess for historical reasons and just people in general, including myself. Here's one interpretation:
- A "register" is something that holds a value. Flip-flops and Latches *are* registers
- Further confusing the situation, people, including myself, often use "register" or "reg" to just refer to flip-flops

Use a lot!

D Flip-Flop

Edge-Triggered Sample-and-Hold Device

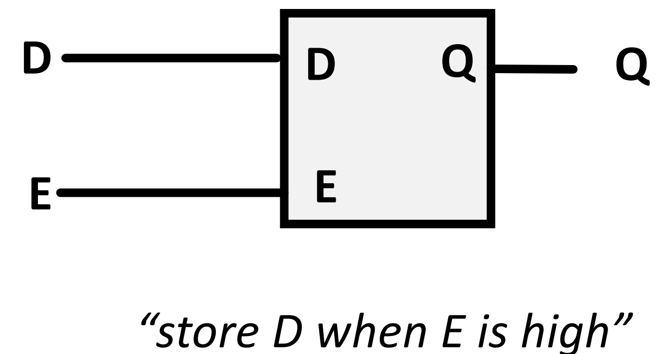


"store D when clk rises"

Won't use as much

D Latch

Level-Triggered Sample-and-Hold Device



"store D when E is high"

D Flip-Flop Registers Give Us A Few Critical Capabilities

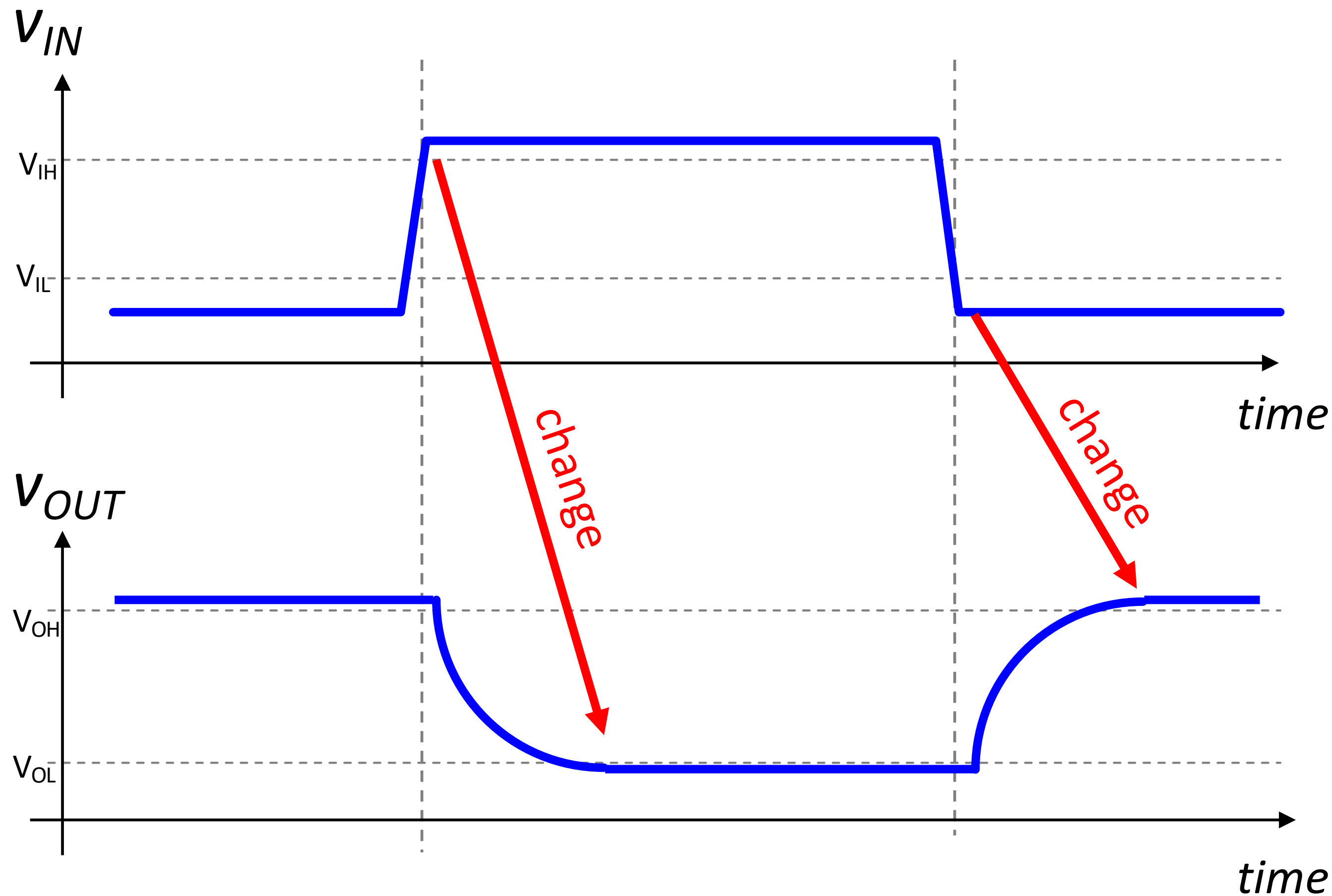
- We can store values for later use (simple memory)
- We can sample values at precise times
  - *A rising edge is as close to a delta-function like event as we can get*
- We can design in stages:
  - Allow us to non-destructively limit signal propagation which prevents:
    - Combinational loops (last week)
    - Glitches (today)

All Electronics are Non-Ideal

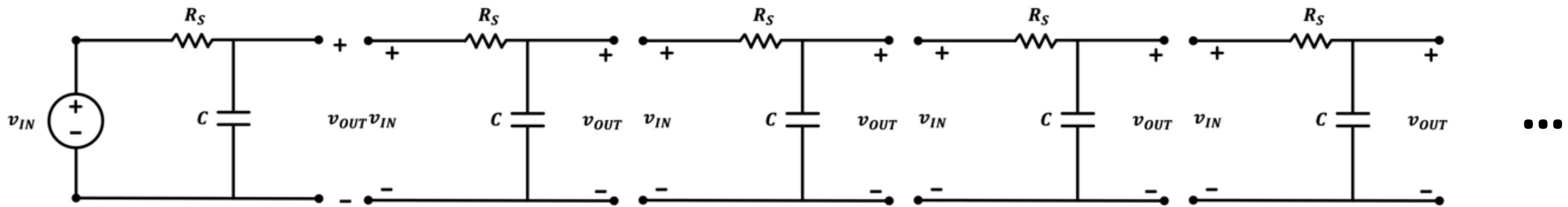
- Inherent to the logic is the need to charge and/or discharge parasitic capacitances and inductances through non-0 value resistances
- As 8.02, or 6.200/6.002 will have shown, this has an inherent time constant involved with it
- …meaning a finite time at which it will respond given a change
- Obviously we don’t want this, but we didn’t want Coronavirus either. What are you going to do? So it goes

When one digital circuit drives another digital circuit

- Inputs change outputs…but takes time



The more complex/more layers/the more delay you'll get



- Each individual "stage" needs to charge up/down before it can influence the next stage.
- Very complicated/deep logic will take time

It’ll take time to transition

- Response of a function will take time (and energy)
- So if we move around on a truth table it can’t be instantaneous

*If you’re **here** on the truth table*

Truth Table

c	b	a	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

How much time?

*And transition to **here***

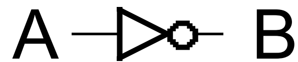
Truth Table

c	b	a	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Digital Delays

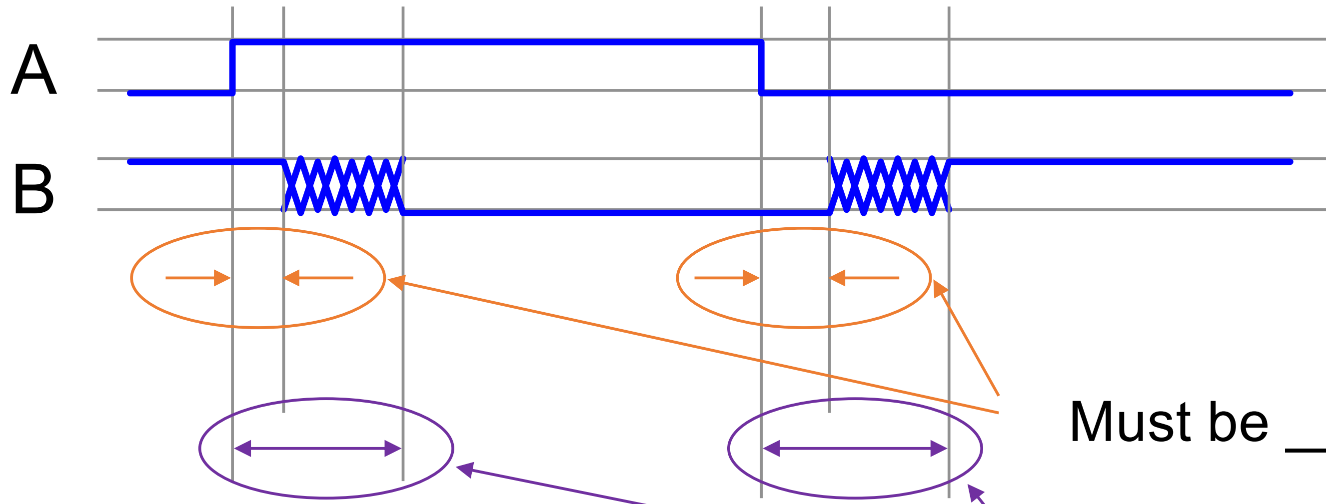
- For a given digital device, we need to quantify the delay
- Utilize two different numbers:
- For a given change at the inputs to a digital system:
  - **Contamination Delay:** How long before the system will start to respond at its output?
  - **Propagation Delay:** How long until we can be sure the system has updated to new value (stabilized)?

The Combinational Contract



A	B
0	1
1	0

t_{PD} propagation delay
t_{CD} contamination delay



Must be $> t_{CD}$

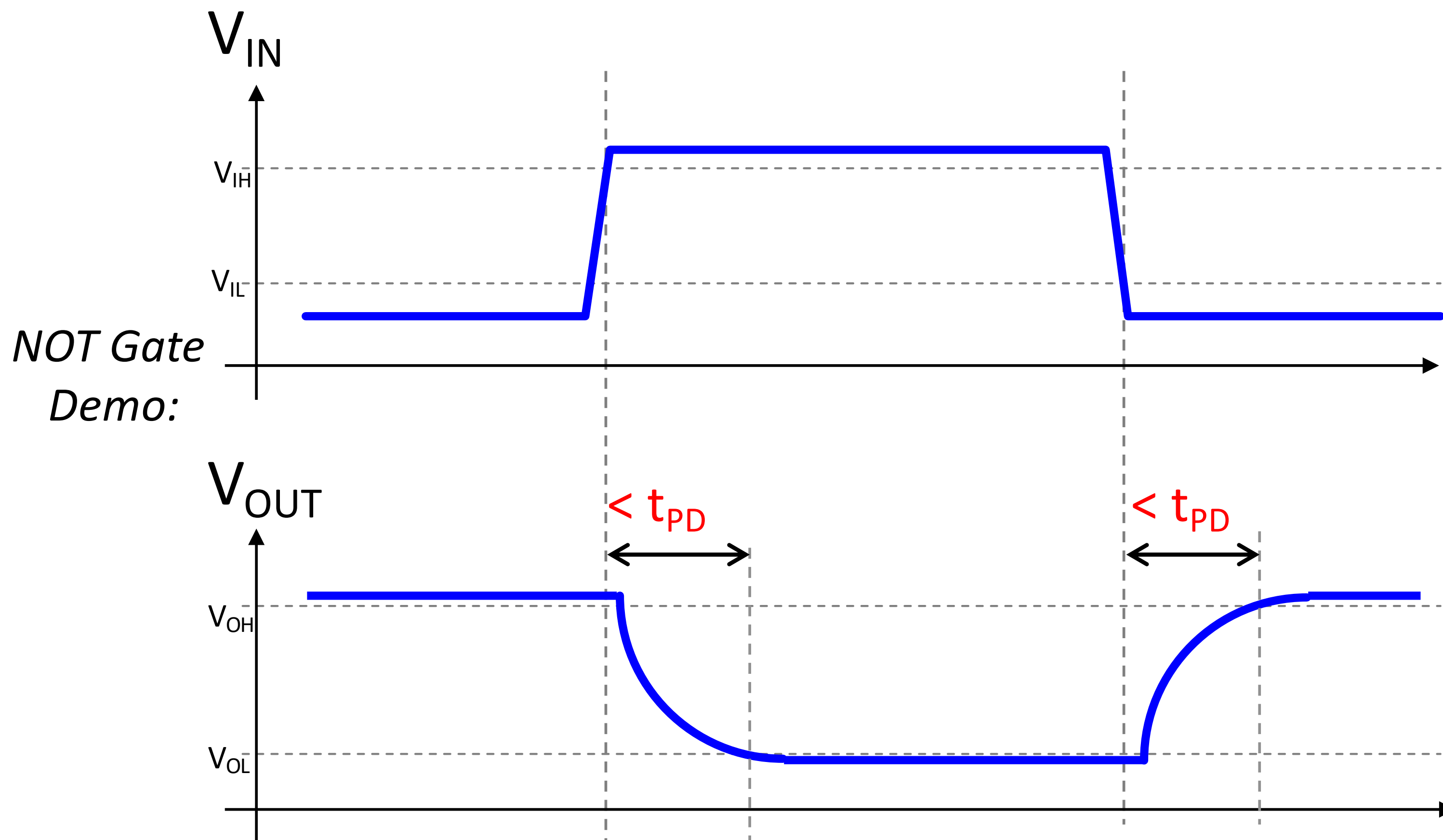
Must be $< t_{PD}$

Note:
1. No Promises during
2. Default (conservative) spec: $t_{CD} = 0$

Worst Case:

Propagation delay (t_{PD}): An upper bound on the delay from valid inputs to valid outputs (aka "$t_{PD,MAX}$")

NOT Gate Demo:



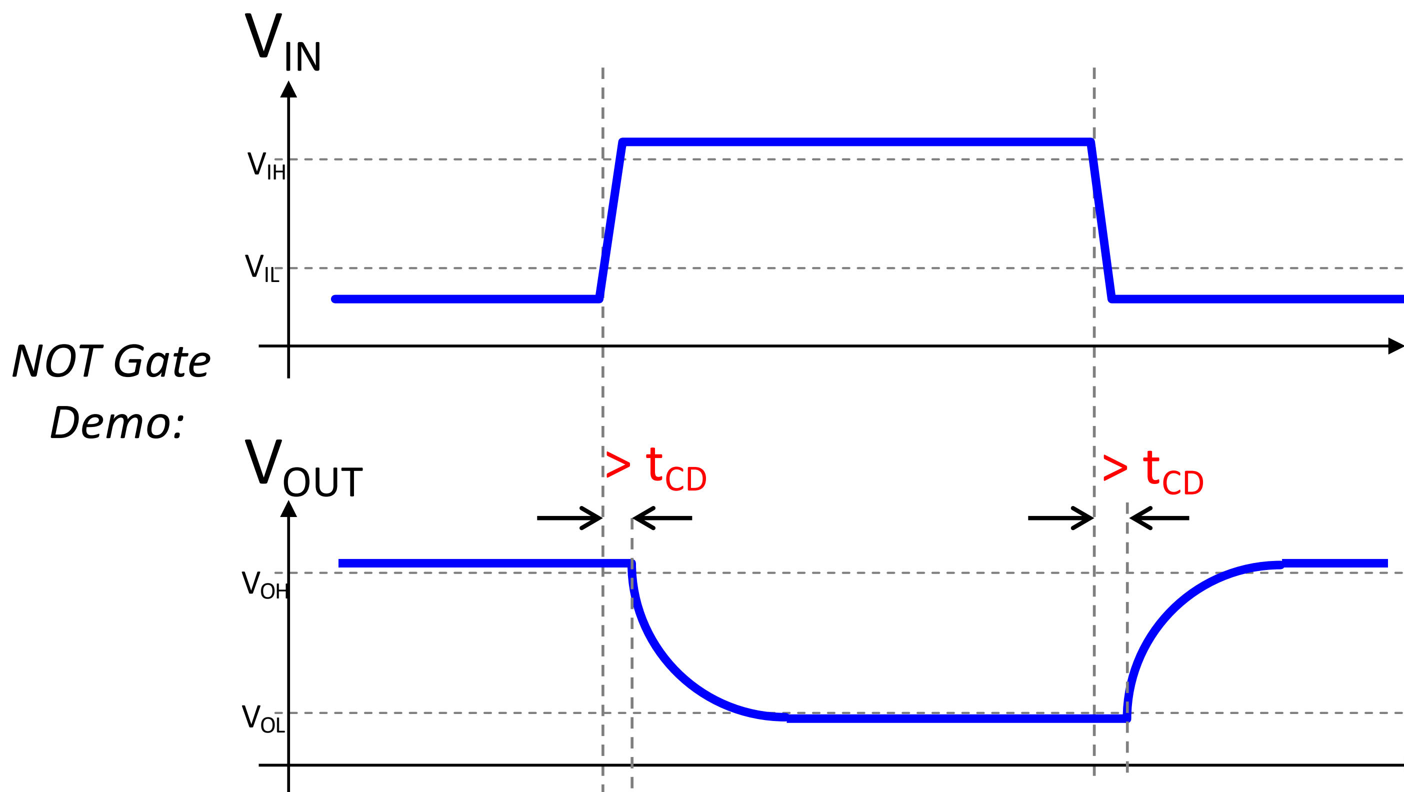
Design goal:

minimize propagation delay

Best Case:

Contamination delay(t_{CD}): A lower bound on the delay from invalid inputs to invalid outputs (aka "$t_{PD,MIN}$")

NOT Gate Demo:



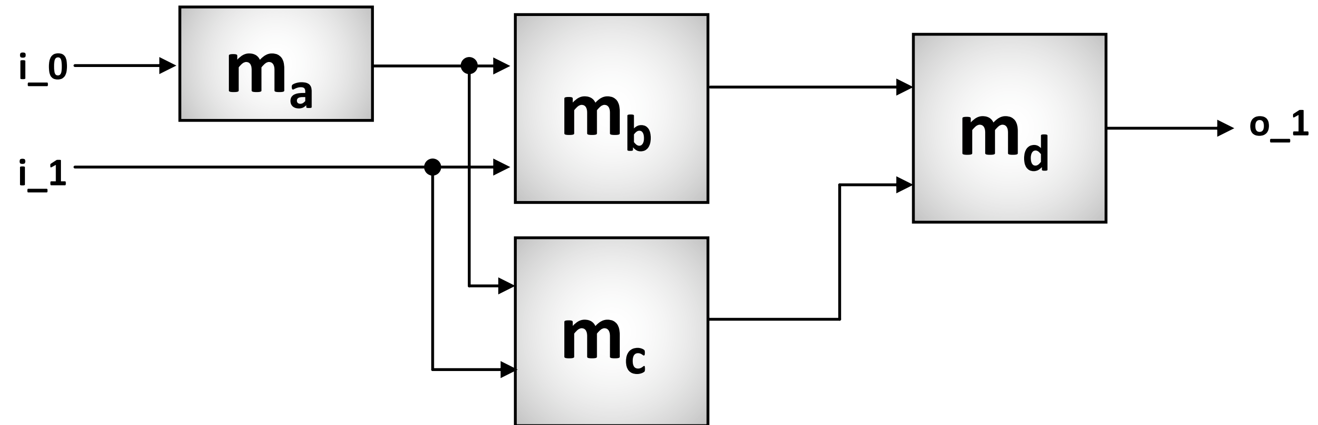
Do we really need t_{CD}?

Usually not... it'll be important when we design circuits with registers (coming next!)

If t_{CD} is not specified, safe to assume it's 0.

Review: Example System

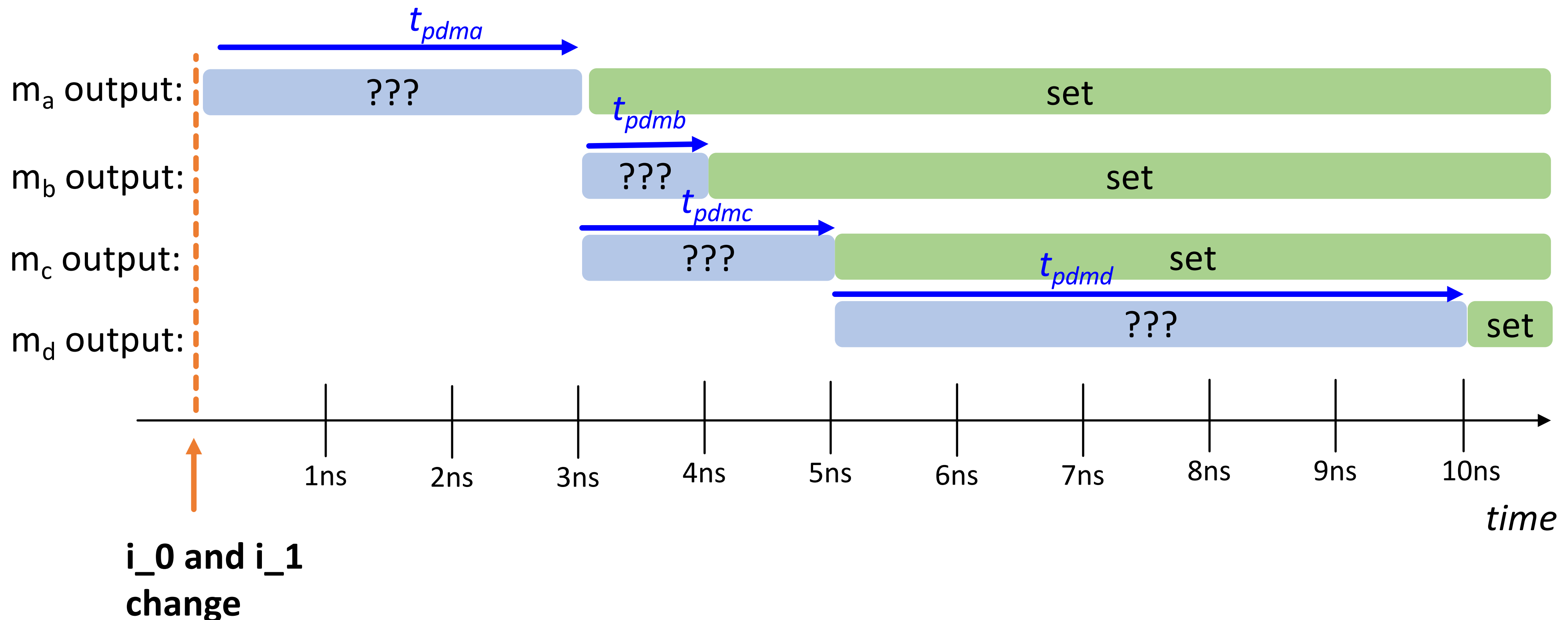
- Let's assume:
  - m_a has t_{pd} = 3ns
  - m_b has t_{pd} = 1ns
  - m_c has t_{pd} = 2ns
  - m_d has t_{pd} = 5ns
  - All four modules have t_{cd} = 0ns



- What is:
  - i_0 to o_1 t_{pd}?
  - i_1 to o_1 t_{pd}?
  - t_{cd} of system?
  - Critical Path of the system and t_{pd}?

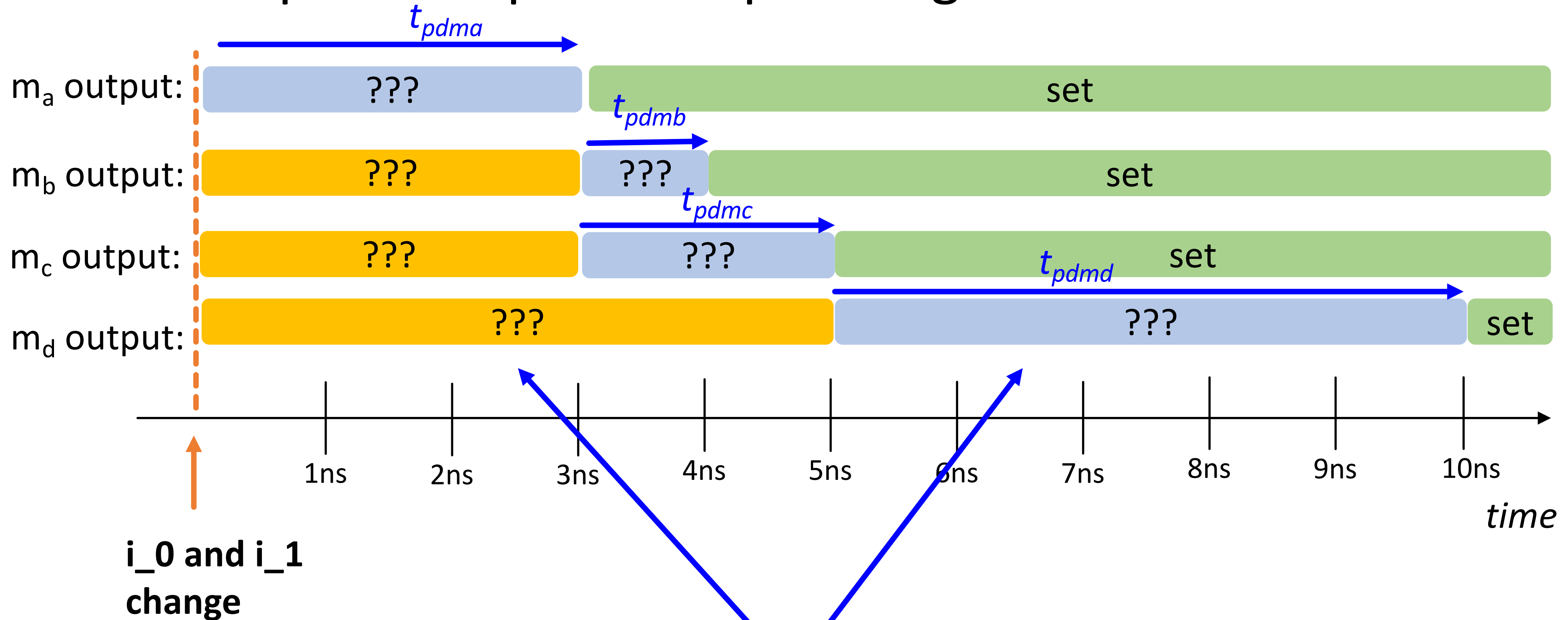
Timing Diagram

- The t_{pd} on any stage/module can't start being used until all inputs to it are set/stable:

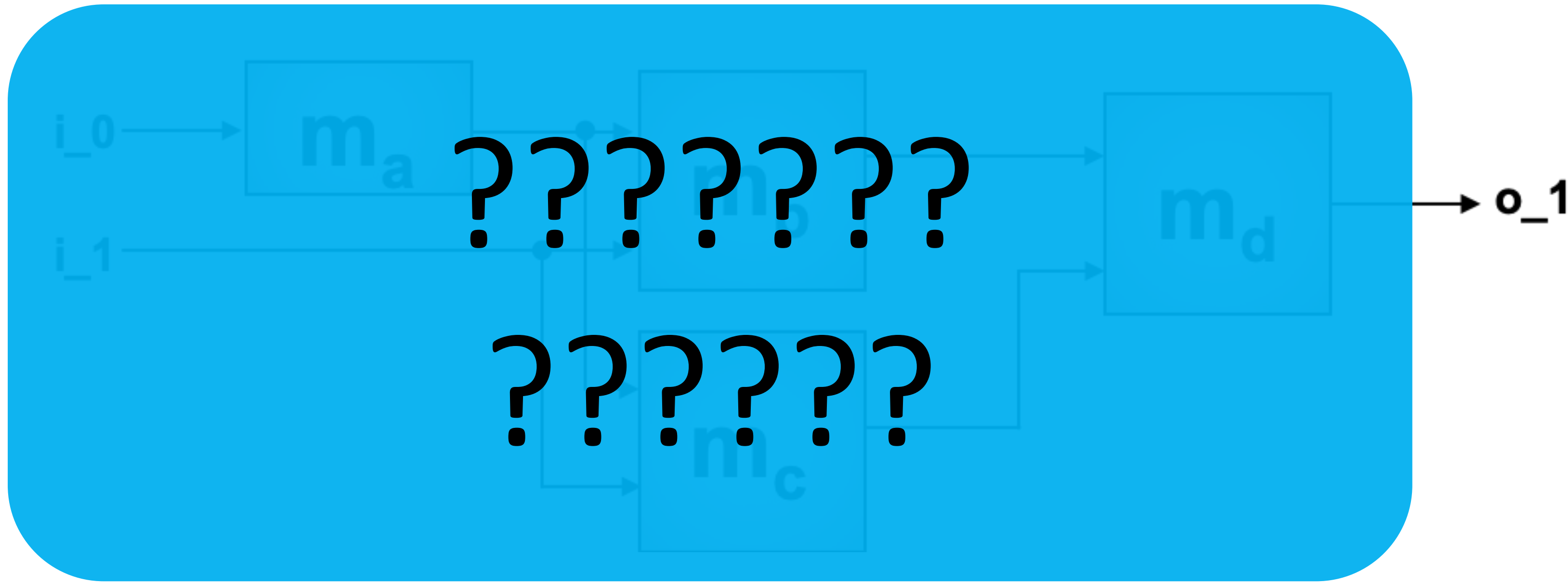


Timing Diagram

- Additionally: the "unknown" periods for subsequent outputs are quite large



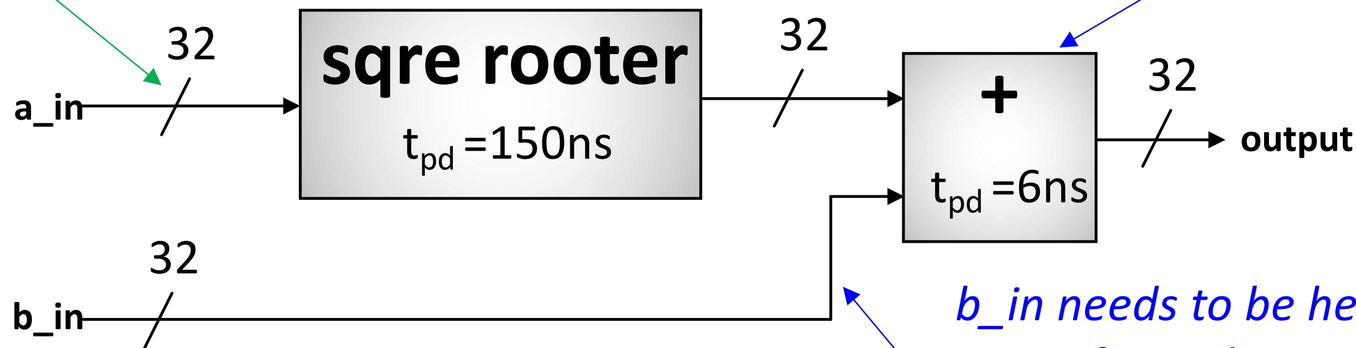
From the Outside



- If all you see is *o_1* how can you actually determine what is valid and what isn't?
- Is that **1'b1** on *o_1* valid or invalid? Who knows?
- Unless you know when you put in values and know the total t_{pd} of system very hard to discern what is good and what isn't

Another Way to Look At Problem

- You have a system, takes in two numbers, a_in and b_in and produces an output.
- System calculates square root of a_in an then adds b_in to it



Another Problem (being a real downer, I know…)

- Consider simple addition in binary (or any base):

```
  11
 10110
+00011
------
 11001
```

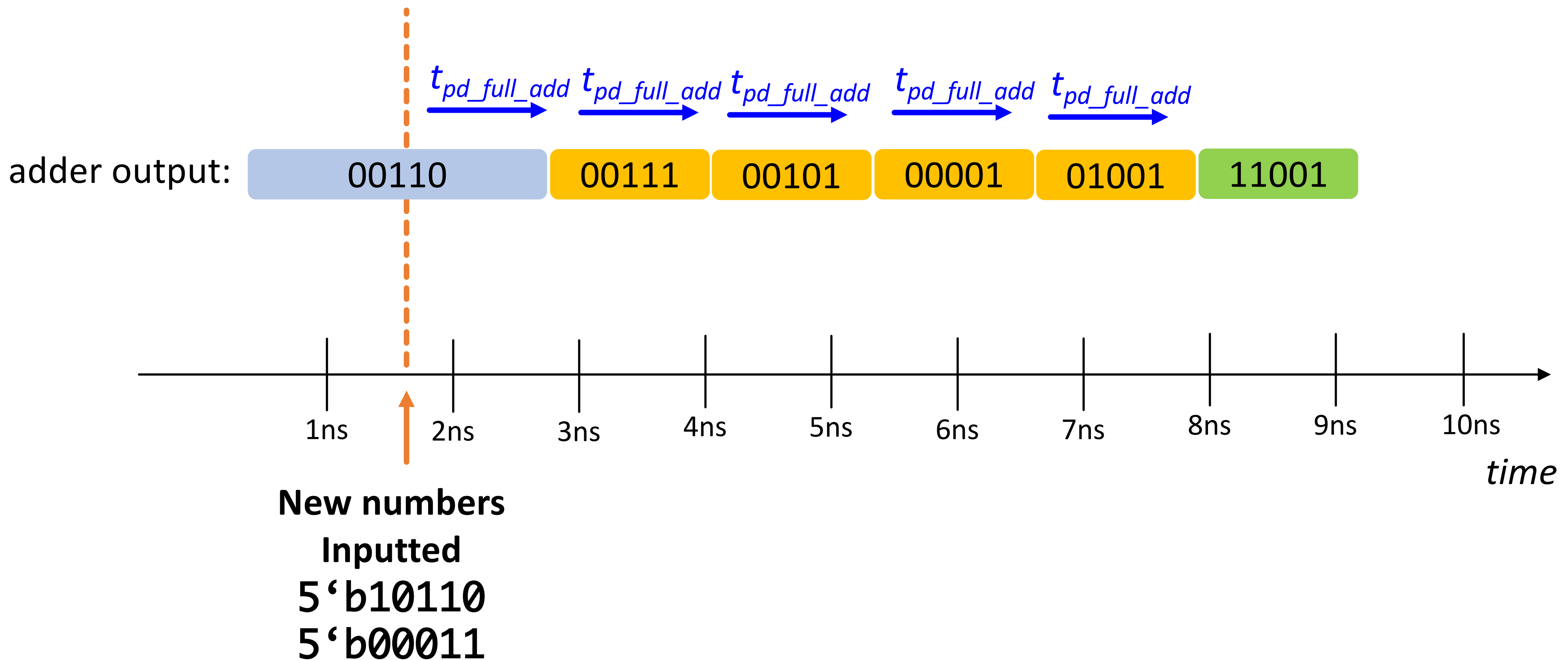
Math checks out:

```
  22
+  3
------
  25
```

- Notice how we need to calculate the lower digits first before we calculate the upper digits?
- Uh Oh…

Timing Diagram of Add

- Lots of invalids before the valid add!



What if we then had this circuit?

i_0

i_1

+

val_in

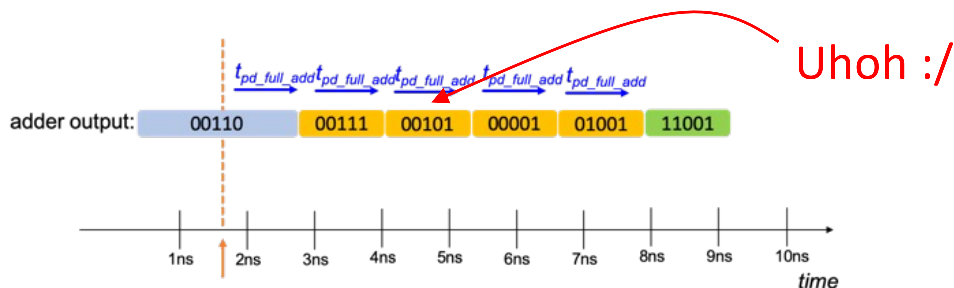
```
module grand_deity (input wire [4:0] val_in,
                    output logic destroy_world_out);

 assign destroy_world_out = val_in == 5'b00101;

endmodule
```

destroy_world_out

0?

1?

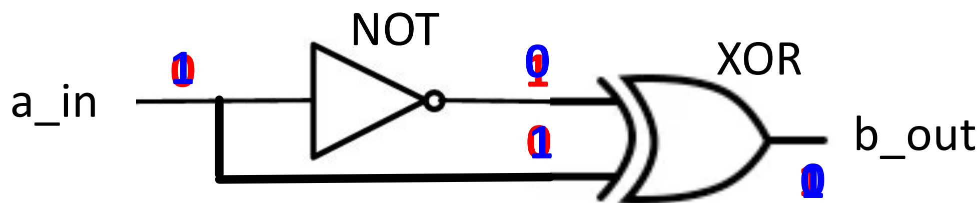
Adder from previous page



Uhoh :/

Combinational Glitches!

- Combinational glitches arise when outputs transition through unintended outputs in response to transitioning inputs
- Caused by differences in overall **OR** internal delays of logic



a_in	b_out
0	1
1	1

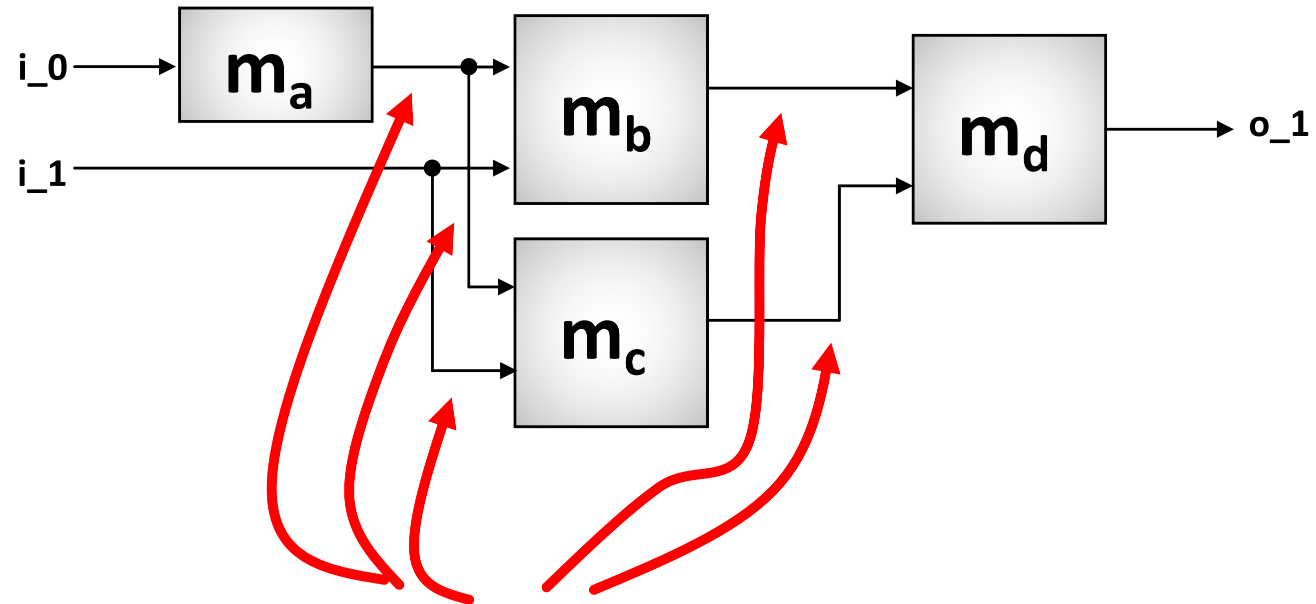
System should always have 1 as output, but during transitions from 0 → 1 or 1→0, b_out will glitch to 0.

Glitches Will Happen

- Inherent with complex and DEEP combinational logic!
- Perform calculations on irrelevant information
  - Waste energy
  - Very hard to debug
- Extremely difficult to design with reliably at scale
  - Too many related time constants
  - Too many invalid values
- Our only hope is to encode our data in glitch-free ways and limit the range that combinational glitches can propagate (next up)

Delays

- In modern digital devices the modules work so fast that their t_{pd} and t_{cd} are not the only concern…



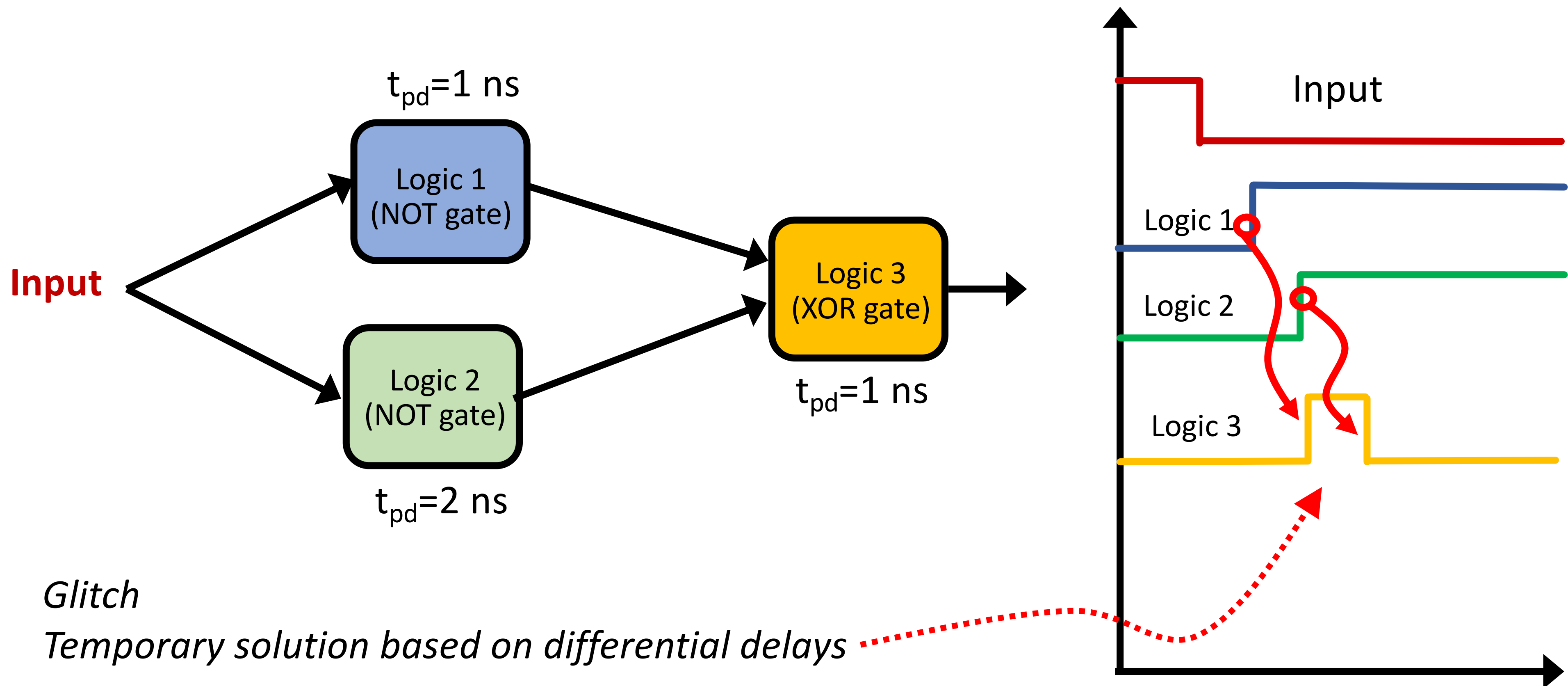
- Vivado will calculate delays for us (we will use it)
- But still have combinational issues! (glitches), various propagation delays!

Why didn't this matter in Lab 1?

- Let's say t_{pd} = 1ns (conservative)
- Human is the consumer
- You push the button…
  - System stabilizes before the photons emitted from the LEDs have even reached your eye
  - Human eye can only detect up to ~0.01s phenomena...lol 6 or 7 orders of magnitude difference
- Basically we can't appreciate the glitches…but they are likely there.

So How to Fix this?

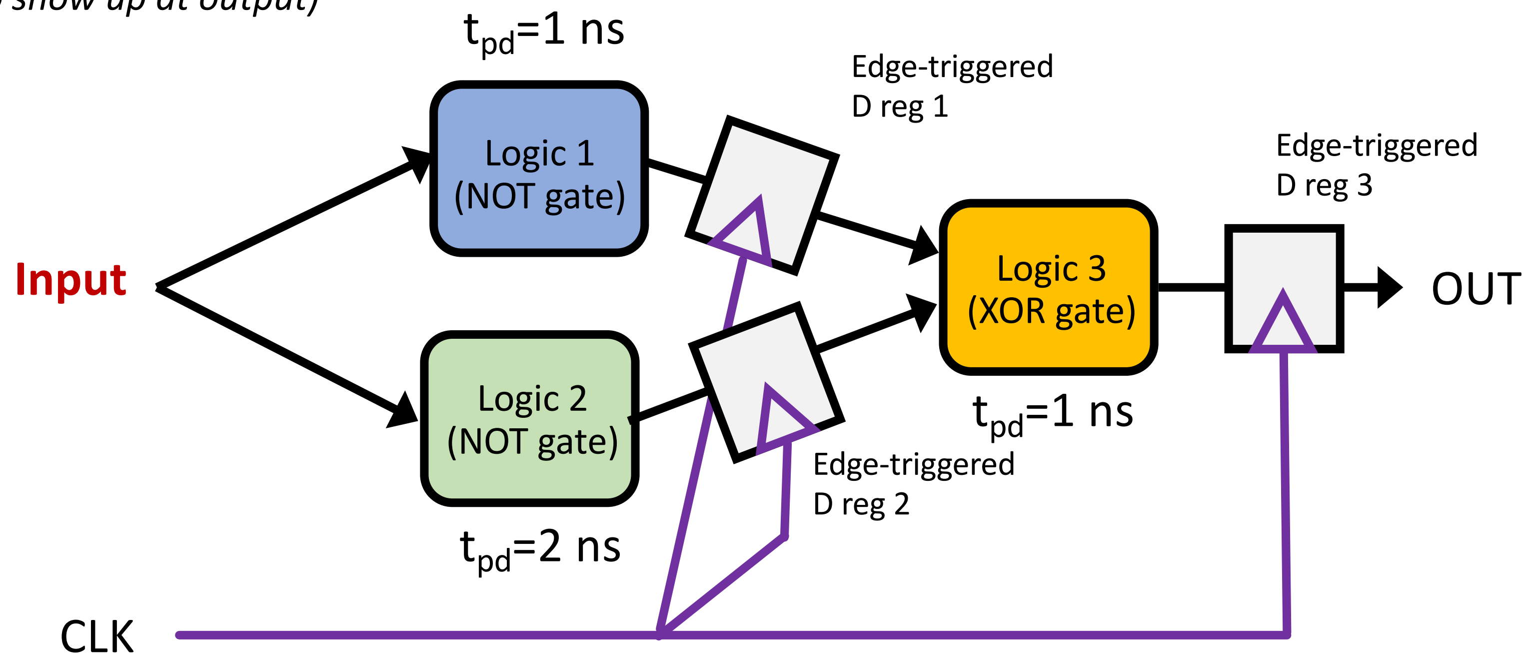
- Every combinational circuit has delays regarding how slowly (or quickly) its outputs change in response to inputs, and this varies based on design/complexity
  - t_{cd} minimum time input takes to start to change output
  - t_{pd} maximum time input takes to finish changing output



This is How We Fix This

- Registers let us isolate/limit signal propagation and synchronize stages

t_{pd} is propagation delay (how long input takes to show up at output)

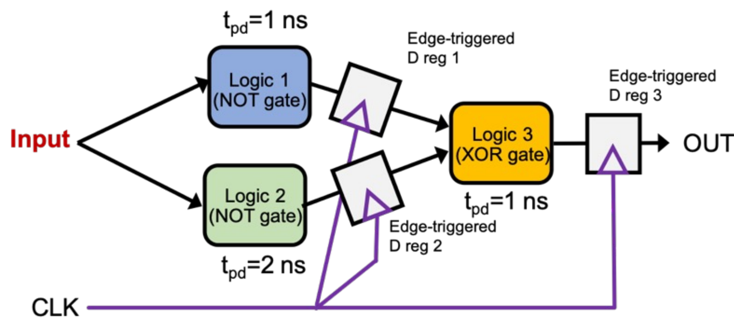


CLK is a synchronization signal

Remember about Delays in Logic

- Registers let us isolate/limit signal propagation and synchronize stages

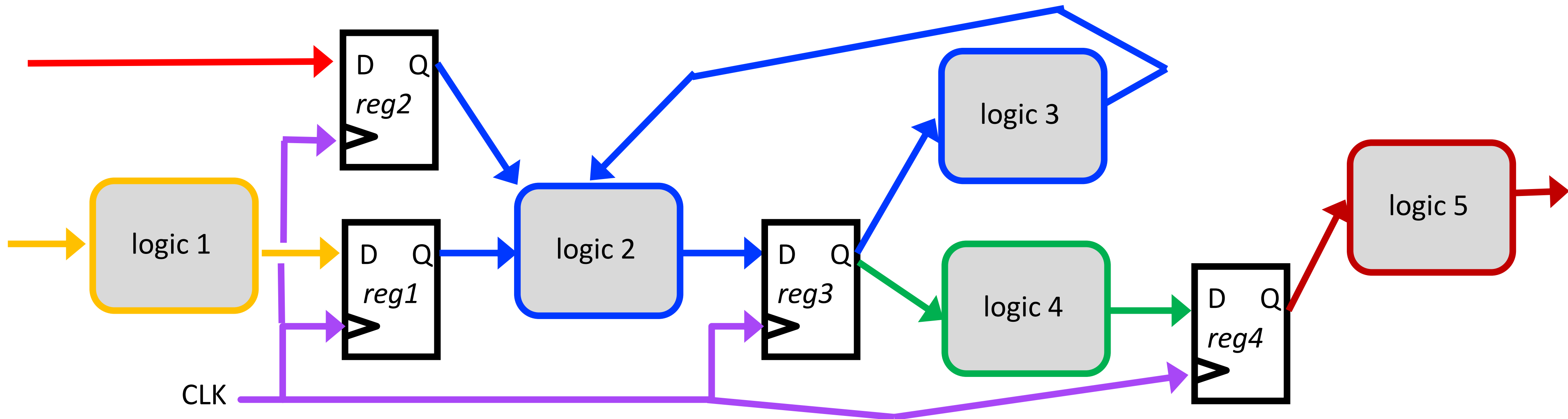
Previous page:



Intermediate glitches are minimized and suppressed in output

Input
Logic 1
Logic 2
CLK
D reg 1
D reg 2
Intermediate glitch
Logic 3
D reg 3
time

Design Complex Logic In Stages!



- D flip-flops regulate signal propagation!
- Design complex logic systems in stages
- Worry only about affects of delays (t_{pd} and t_{cd}) and glitches within a given stage, rather than how they all interplay!

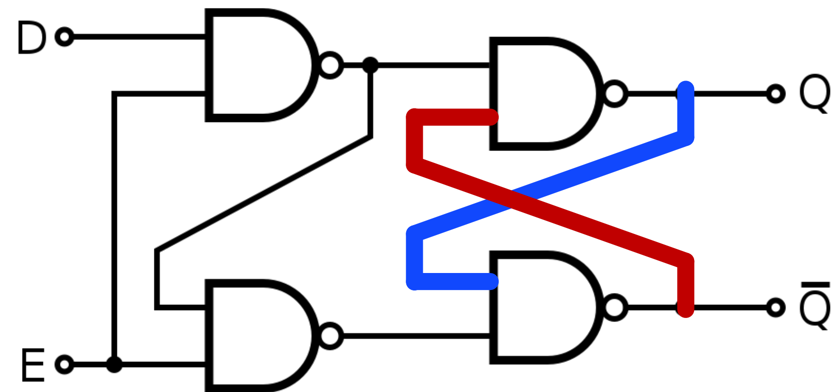
Is that All there is To It?

- No. No there’s not
- Let’s return to how Latches and Flip Flops actually work

The D Latch

- Made of gates (which are made of transistors, which are made of sand(currently))
- Something different though…what is it?

"latch" means it holds whatever value was already present…basically: "Previous Q"



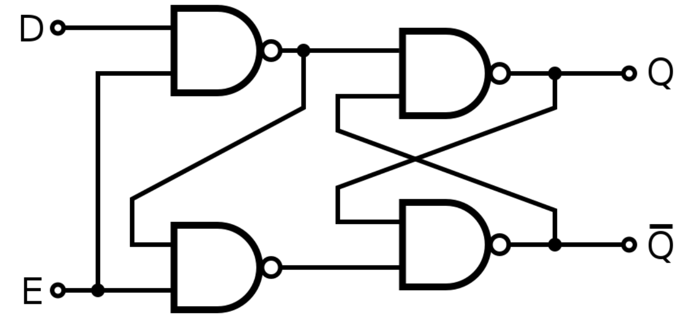
E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

E = "Enable" D = "Data"

Q = not sure, but it is the output

https://www.allaboutcircuits.com/textbook/digital/chpt-10/d-latch/

The D Latch Provides Memory!



1. Set E=1
2. Set your D value
3. Set E=0
4. Whatever D was is stored at Q forever until E is 1 again!
5. **Can we do better/different?**

E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

E = "Enable" D = "Data"

Q = not sure, but it is the output

The D Flip-Flop (Reg)

Two D-Latches in Series driven with opposite enable signals

Data propagates through first D Latch

D

Enabled

CLK C

LOW

!Enabled

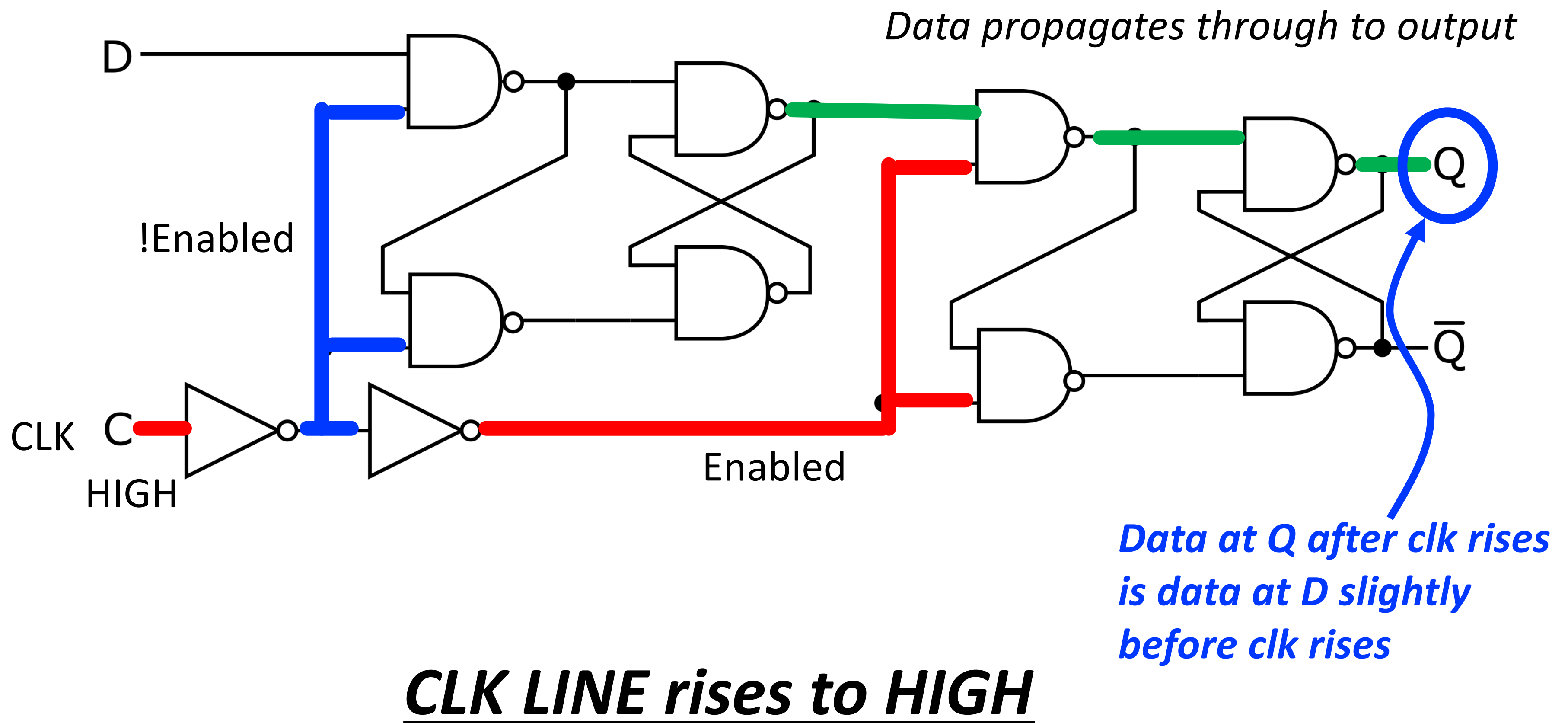
Q

$\overline{Q}$

<u>CLK LINE is LOW</u>

The D Flip-Flop (Reg)

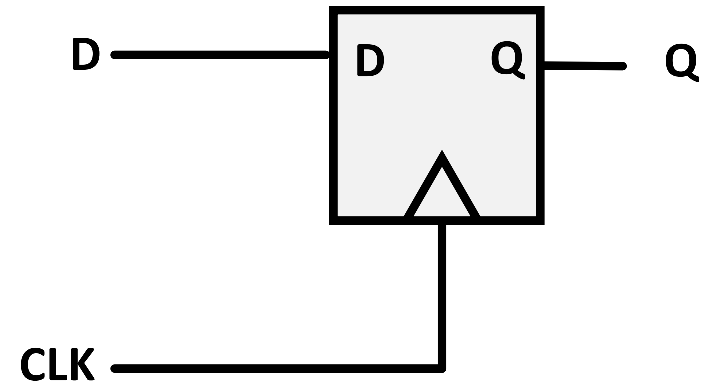
Two D-Latches in Series driven with opposite enable signals

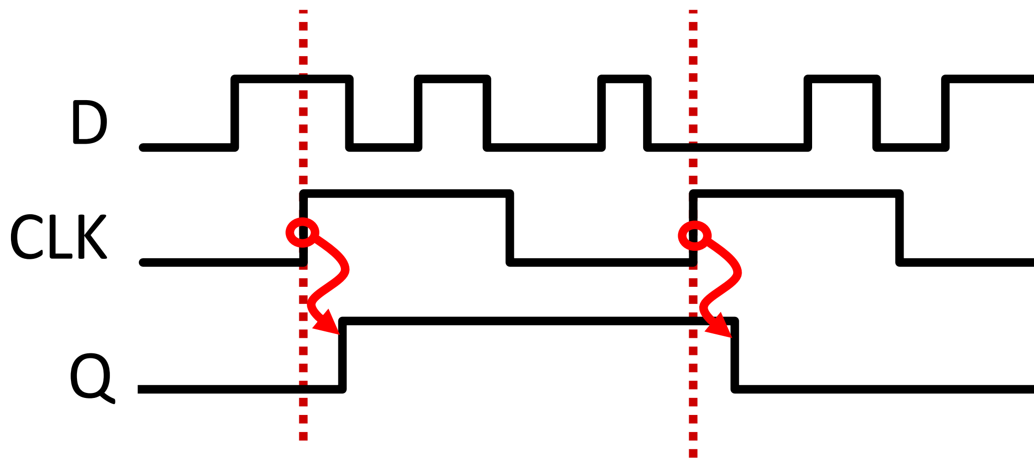


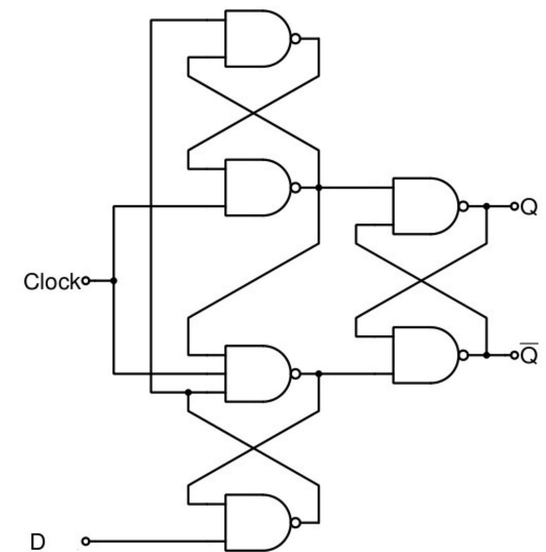
CLK LINE rises to HIGH

The Result: the D Flip-Flop

- The edge-triggered D register: *on the rising edge of CLK*, the value of D is saved in the register and then appears shortly afterward on Q.



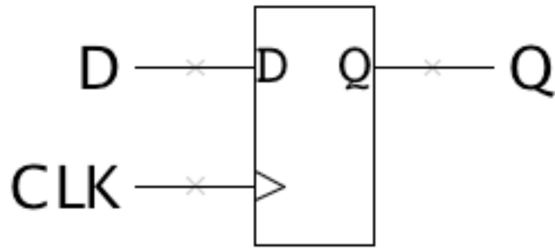


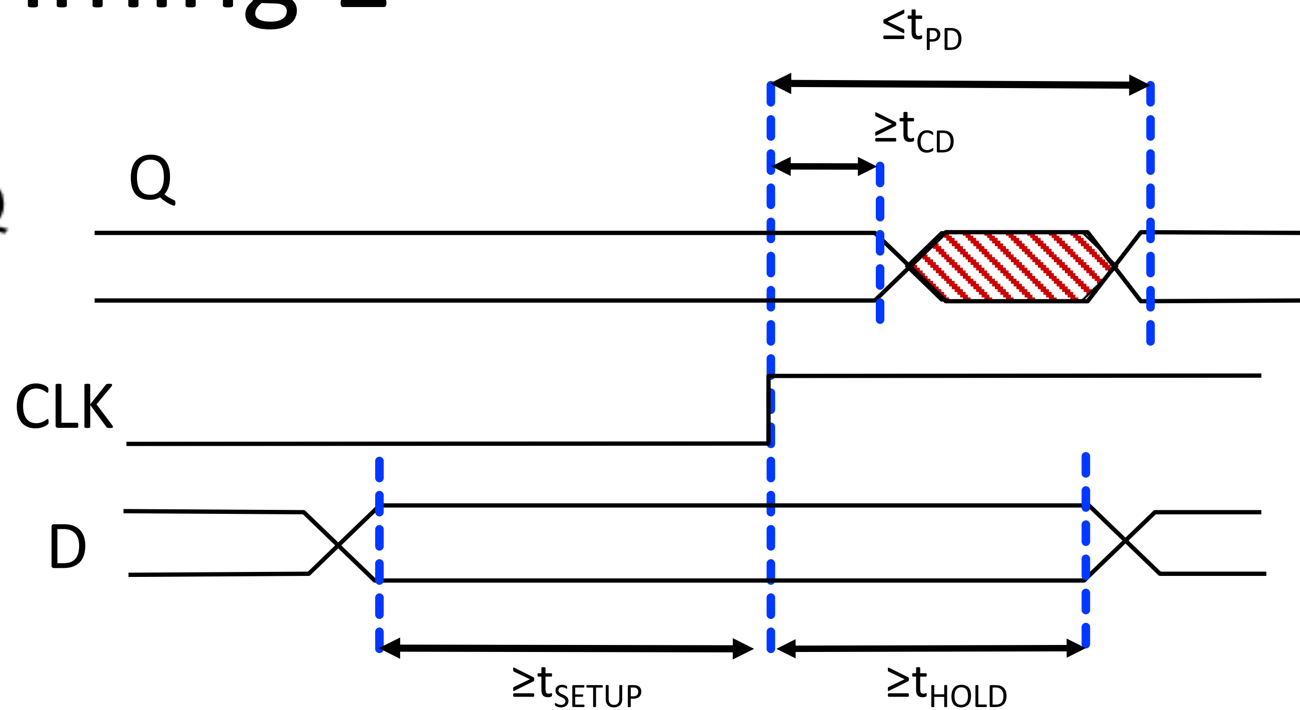


Example: 74LS74 internals

When you simplify some common/redundant logic between the two stages, you get to about ~25 transistors

D-Register Timing 1





IMPORTANT:

t_{PD}: maximum propagation delay, @posedge CLK D →Q

__Maximum__ time it takes for Q to change after rising edge of CLK

t_{CD}: minimum contamination delay, @posedge CLK D →Q

__Minimum__ time it takes for Q to start to change after rising edge of CLK

t_{SETUP}: setup time

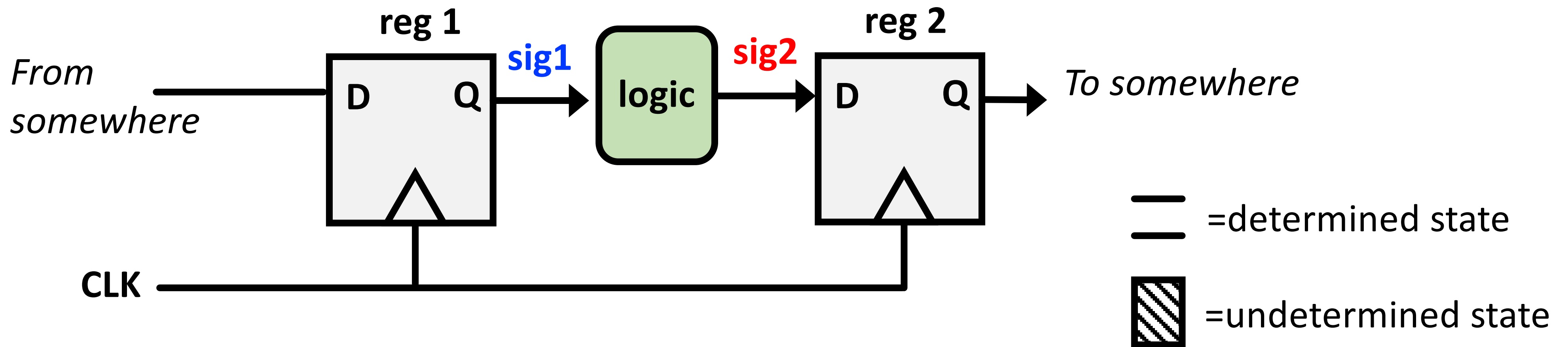
How long D must be stable __before__ the rising edge of CLK

t_{HOLD}: hold time

How long D must be stable __after__ the rising edge of CLK

New timing attributes for registers

Register-to-Register Timing



t_{CLK}

CLK

$t_{PD,reg1}$

$t_{CD,reg1}$

$\geq t_{SETUP,reg2}$

sig1

$t_{PD,logic}$

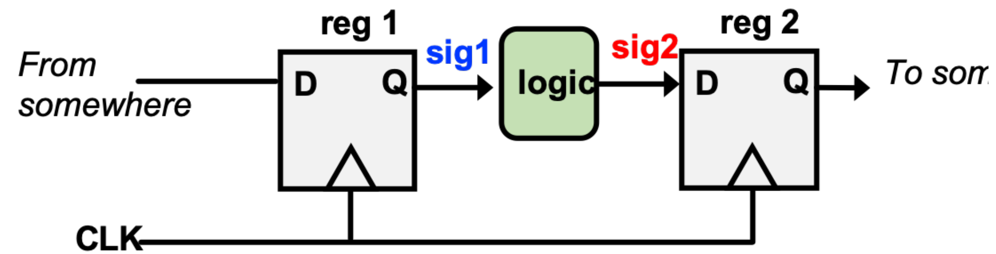
sig2

$\geq t_{HOLD,reg2}$

$t_{CD,logic}$

time

Register-to-Register Timing



=determined state

=undetermined state

t_{CLK}

CLK

$t_{PD,reg1}$

$t_{CD,reg1}$

$\geq t_{SETUP,reg2}$

sig1

$t_{PD,logic}$

sig2

$\geq t_{HOLD,reg2}$

$t_{CD,logic}$

time

Two Requirements/ Conclusions:

$$t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$$

$$t_{CD,reg1} + t_{CD,logic} \geq t_{HOLD,reg2}$$

D Register Timing Conclusions

$t_{PD,reg}$, $t_{SETUP,reg}$, $t_{CD,reg}$, $t_{HOLD,reg}$, and $t^*_{CD,logic}$ are all roughly fixed/ unchangeable

$$t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$$

*We may/will encounter this in 6.205! If we try to make our combinational logic **tooooo complex** and we won't satisfying timing. How do we fix? Two options:*

6.205 Design Space:

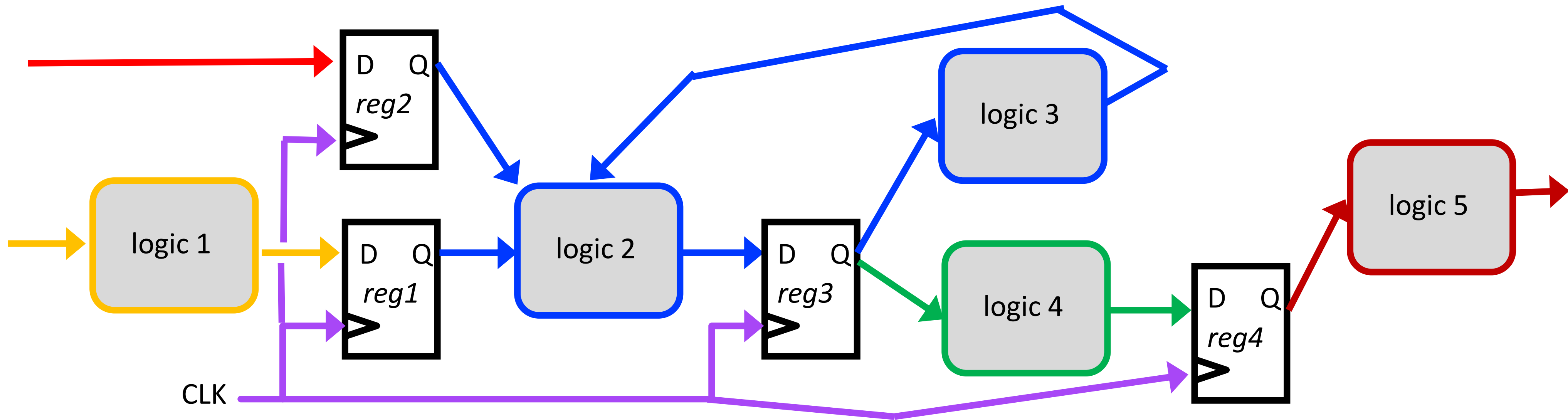
Slow down clock: $\uparrow t_{CLK}$

Chop up combinational logic: $\downarrow t_{PD,logic}$

$$t_{CD,reg1} + t_{CD,logic} \geq t_{HOLD,reg2}$$

*If you violate this, you have to change your design. This is more an issue for the device engineers…on our FPGAs the contamination delays (min change times) are usually longer than HOLD times, so it is hard for **us** to run into this problem in 6.205 (though it is a very real problem for people laying out circuits)*

Design Complex Logic In Stages!



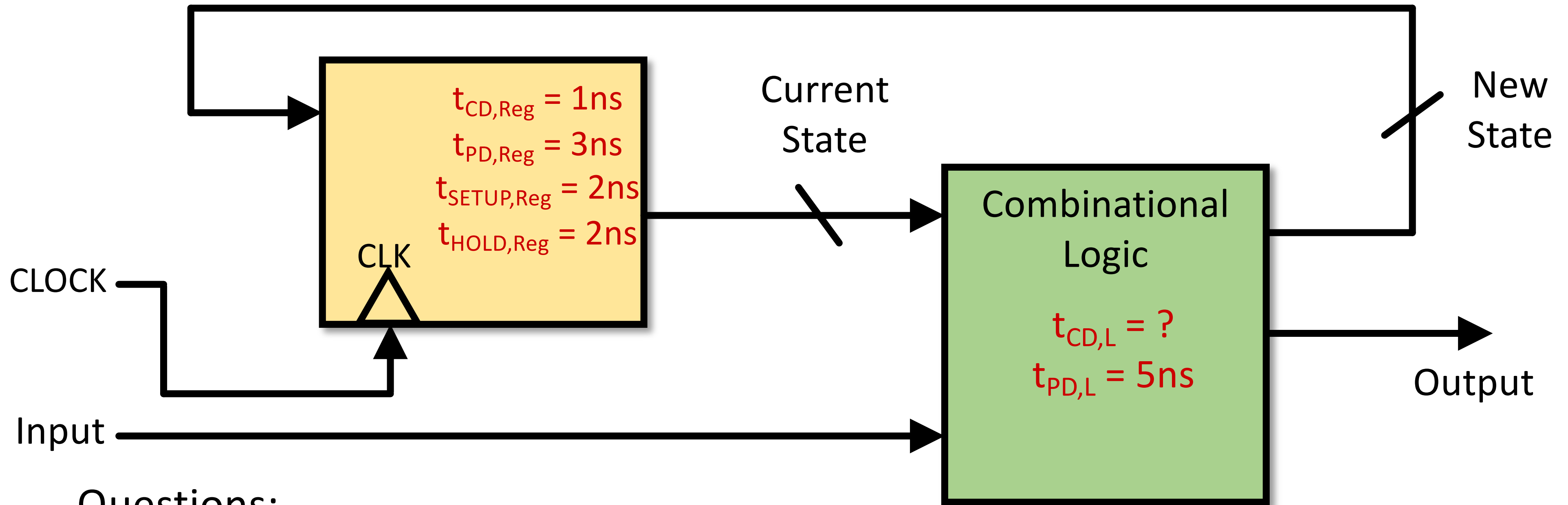
- Design complex logic systems in stages
- Worry only about effects of delays (t_{pd} and t_{clk}) within a given stage, rather than how they all interplay!

Single Clock Synchronous Discipline

- The timing requirements are already complicated enough with one clock. Avoid multiple clocks at all cost! <u>DO NOT clock flip flops on non-clock lines.</u>
- Single Clock signal shared among all clocked devices (one clock domain)
- Only care about the value of combinational circuits just before rising edge of clock
- Clock period greater than every combinational delay
- Change saved state after noise-inducing logic changes have stopped!

Sequential Circuit Timing

Assume input is also coming from a clocked system



Questions:

- Constraints on t_{CD} for the logic?
- Minimum clock period?
- Setup, Hold times for Inputs?

This is a simple *Finite State Machine* … more in future classes!

Interfacing to Sequential Logic

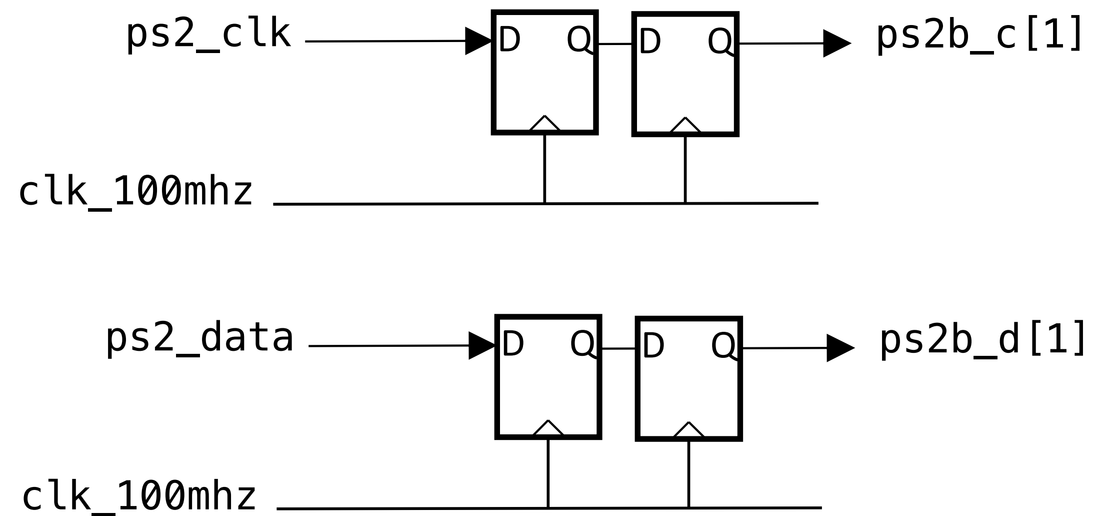
…Or what are the problems with working with Sequential Logic?....

Huh?

- In Lab 2:

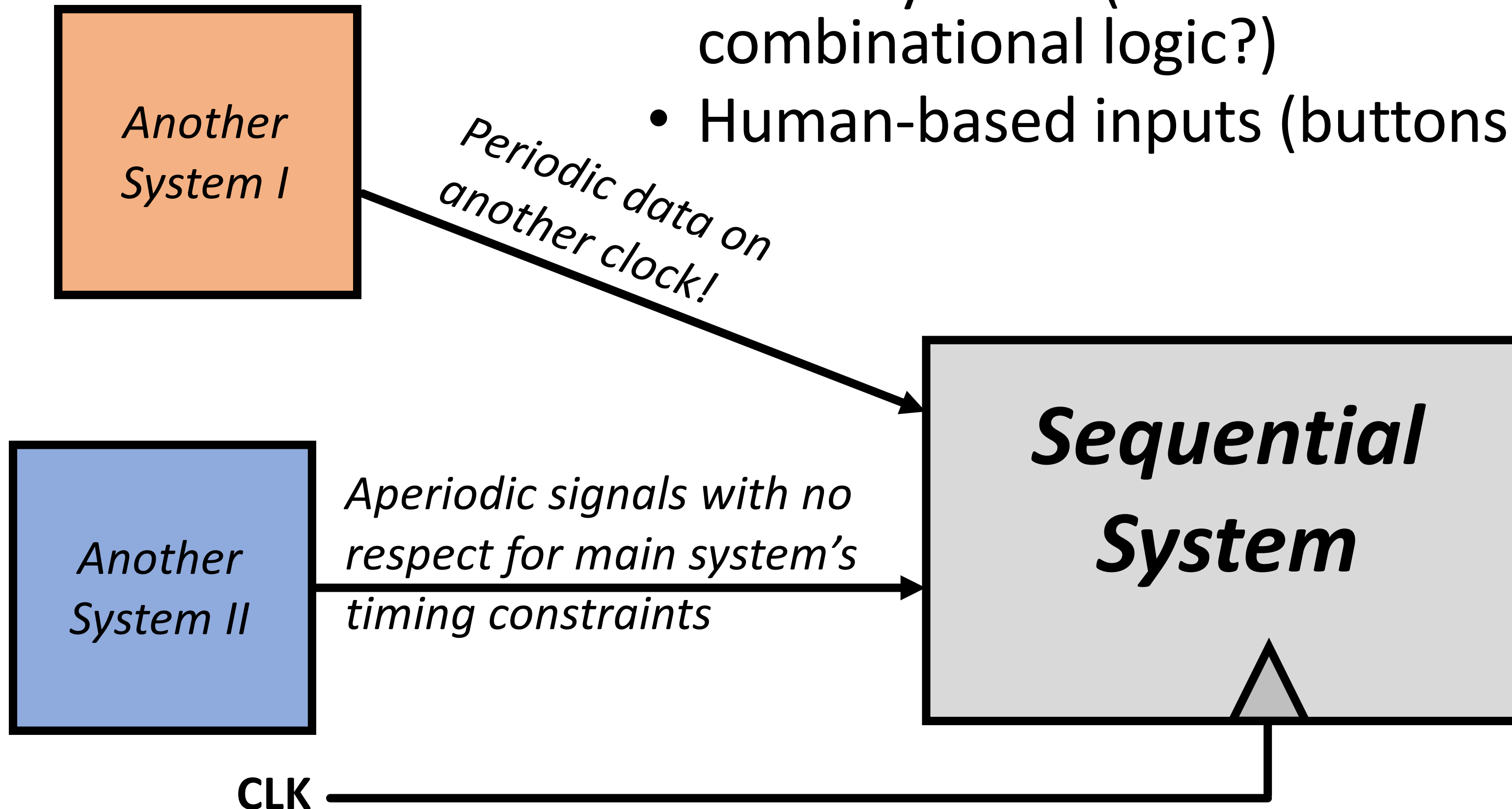
```
logic [1:0] ps2b_c;
logic [1:0] ps2b_d;

always_ff @(posedge clk_100mhz)begin
  ps2b_c[0] <= ps2_clk;
  ps2b_d[0] <= ps2_data;
  ps2b_c[1] <= ps2b_c[0];
  ps2b_d[1] <= ps2b_d[0];
end
```

- Basically builds this:



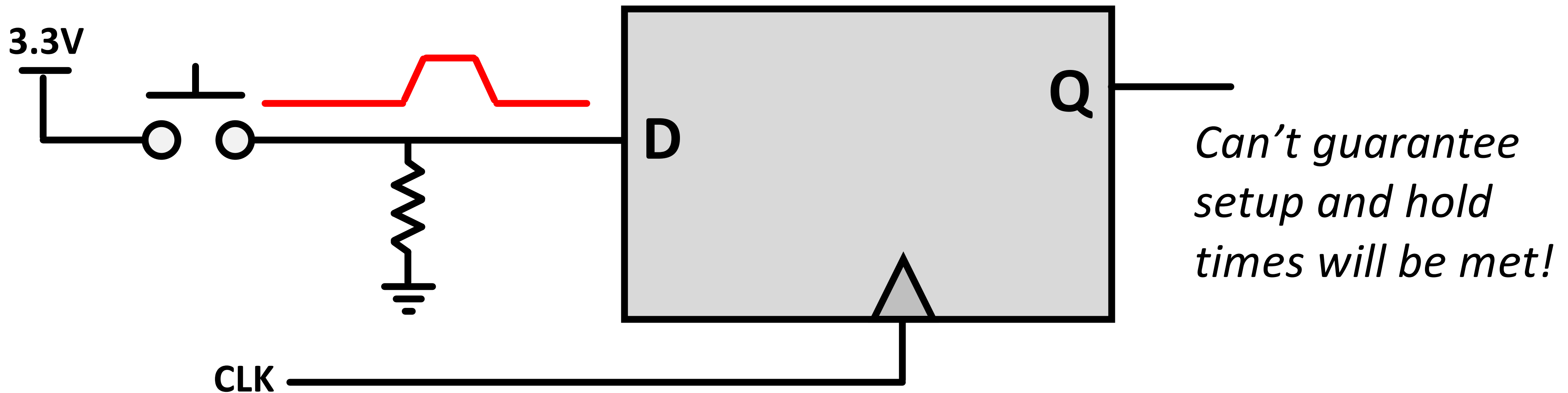
What if…?

- …we need to interface with outside equipment:
  - Other systems (on different clocks or from combinational logic?)
  - Human-based inputs (buttons)



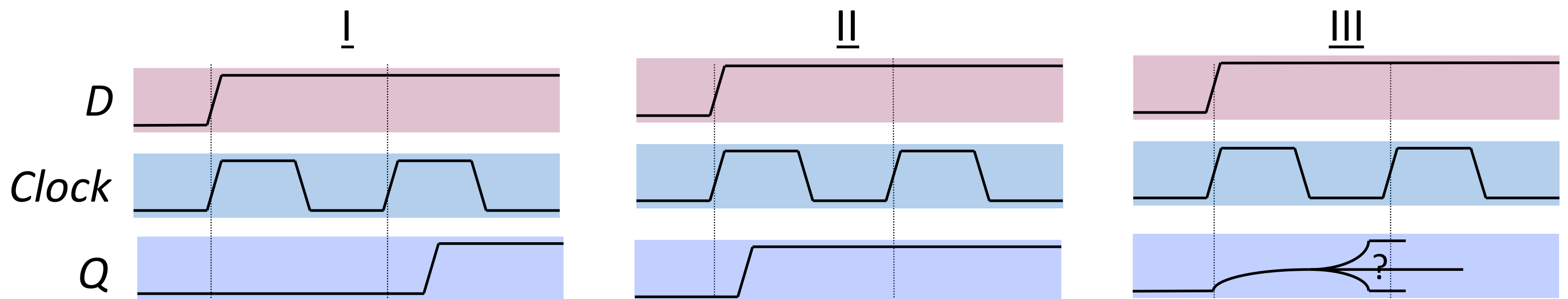
Can't guarantee setup and hold times will be met!

Example: Asynchronous Inputs in Sequential Systems



Can't guarantee setup and hold times will be met!

When an asynchronous signal causes a setup/hold violation...



Transition is missed on first clock cycle, but caught on next clock cycle.

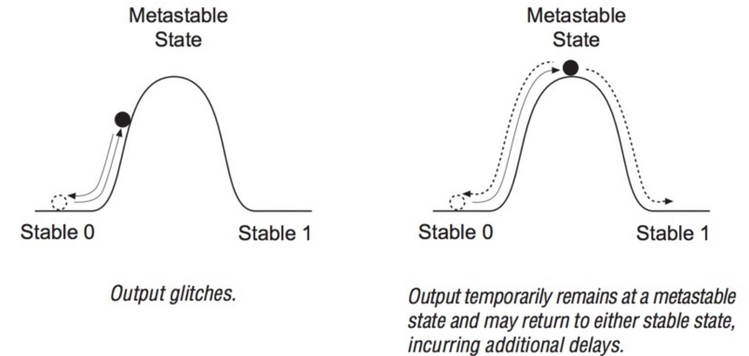
Transition is caught on first clock cycle.

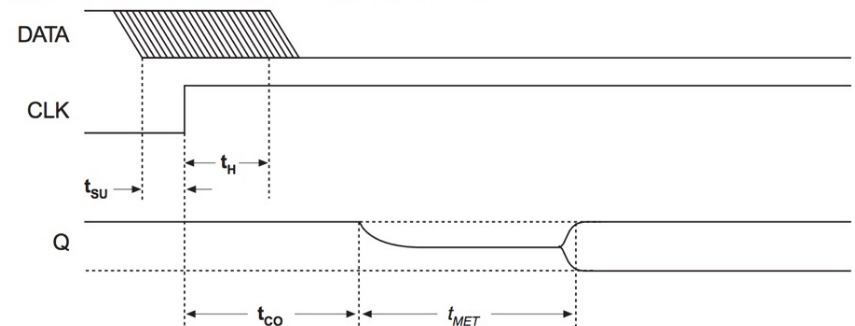
Output is metastable for an indeterminate amount of time.

Q: Which cases are problematic?

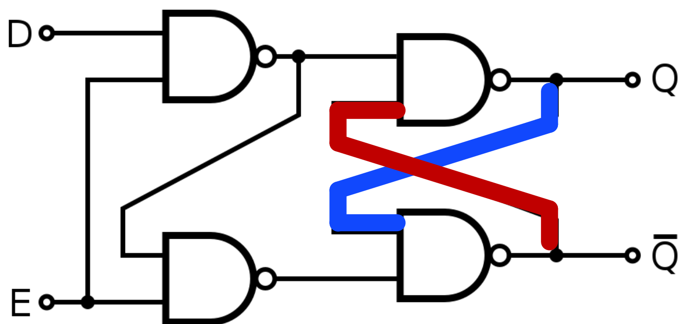
Metastability

- D-registers have issues with all that feedback and stuff going on. Can go **metastable**
- Metastability is where the system hovers between Logic High and Logic Low in an unpredictable way





Metastability in Altera (☹) Devices
Altera Application Note 42 (1999)



t_{co} = *"min time from clock to output"*
....think of it as t_{pd} here (not exactly the same,

Handling Metastability

- Can’t globally prevent metastability, but can isolate it!
- Stringing several registers together can isolate any freakouts!

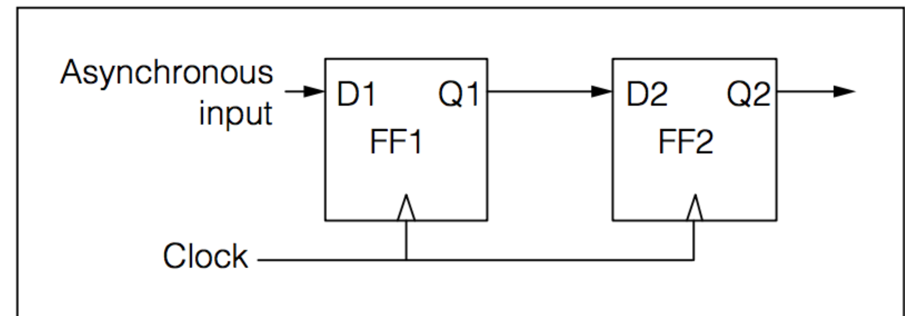
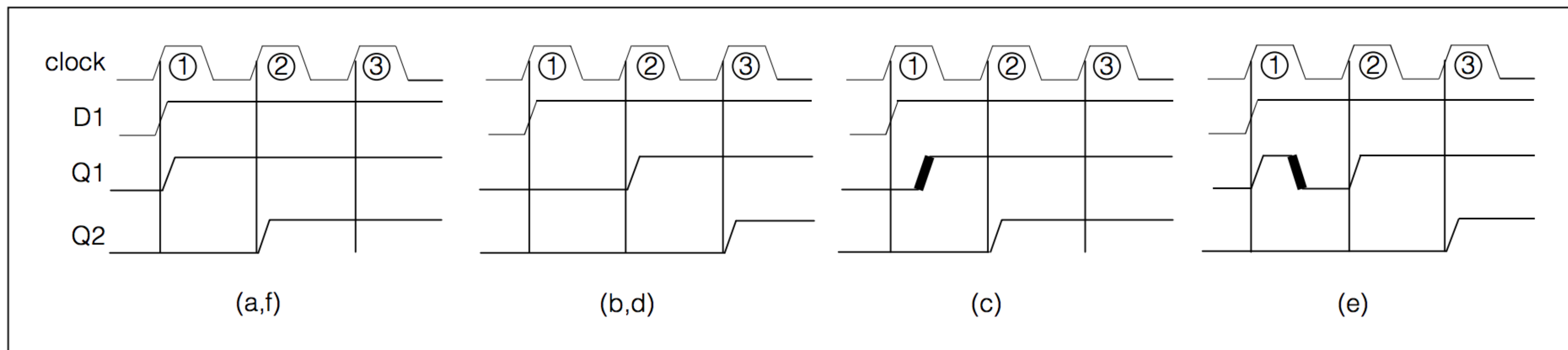


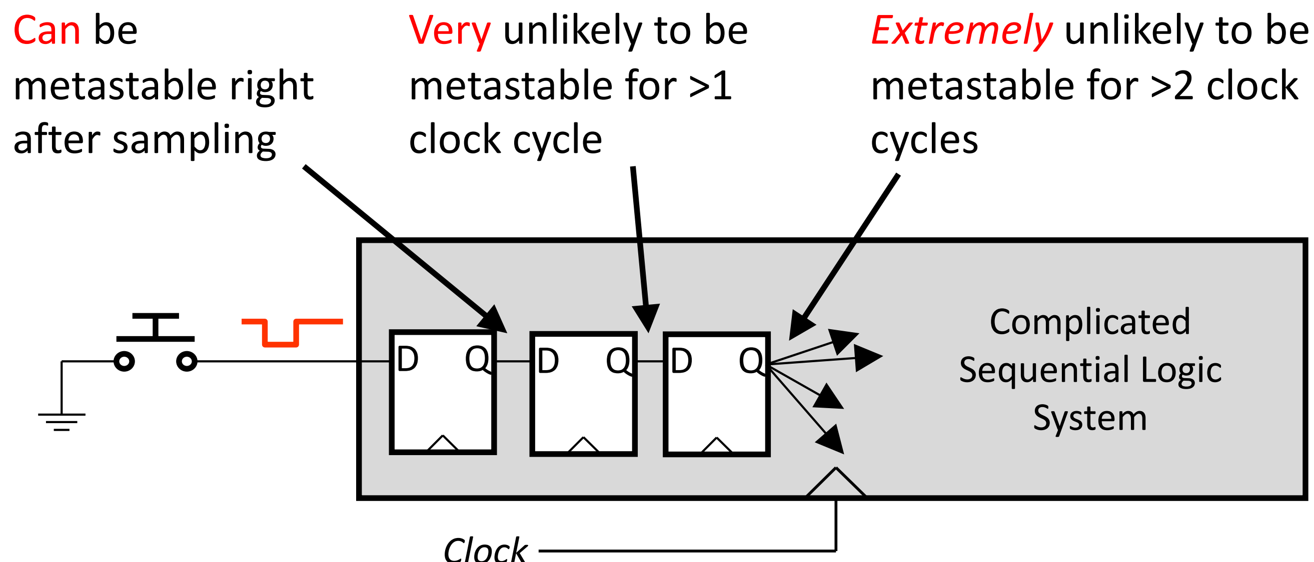
Figure 8. Two-flip-flop synchronization circuit.



“Metastability and Synchronizers: A Tutorial”
Ran Ginosar, Technion Israel Institute of Technology

Handling Metastability

- Completely preventing metastability turns out to be an impossible problem
- High gain of digital devices makes it likely that metastable conditions will resolve themselves quickly
- Solution to metastability: allow time for signals to stabilize

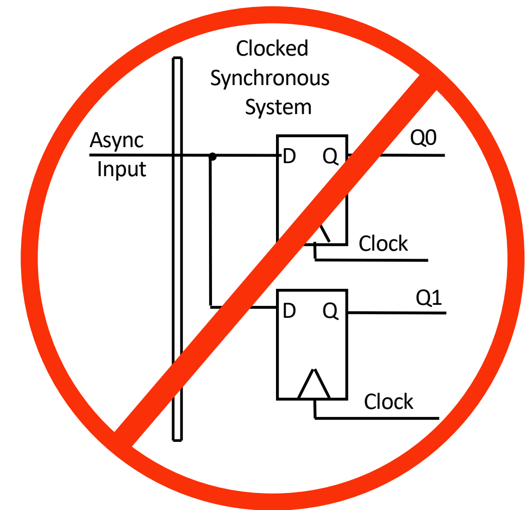


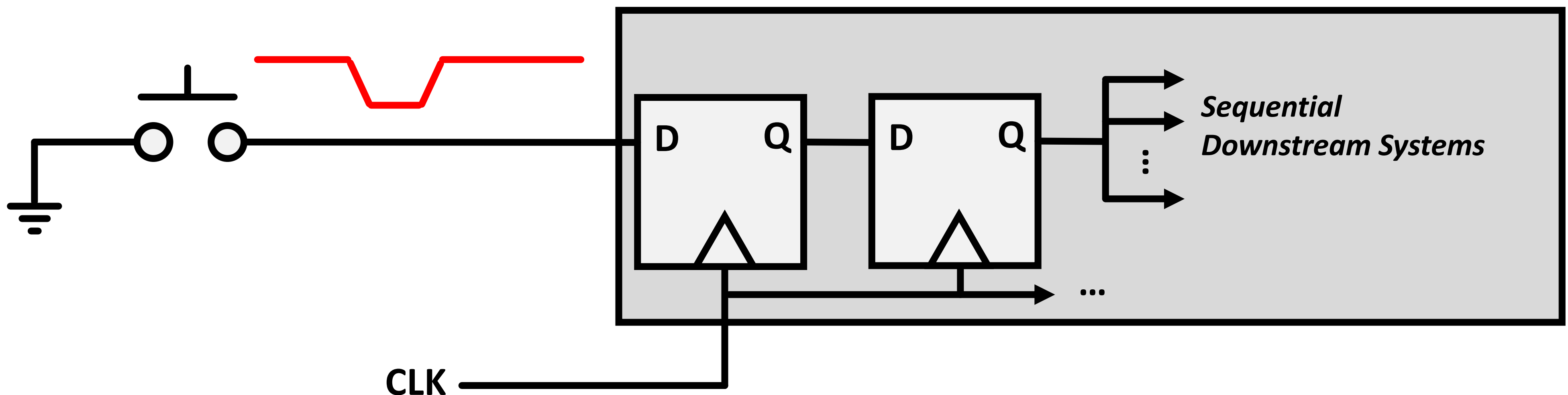
How many registers are necessary in 6.205?

- Depends on many design parameters (clock speed, device speeds, …)
- In 6.205, **a pair of synchronization** registers is sufficient
- And for simple designs…with low t_{pd} you may not even need anything

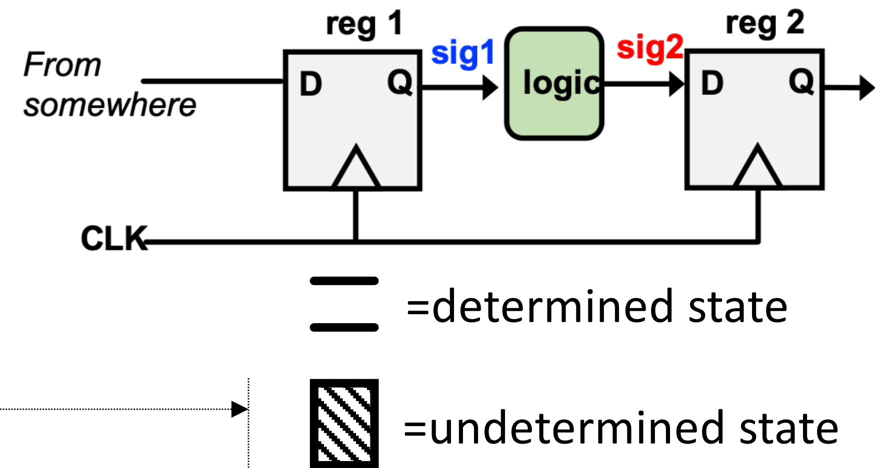
Handling Metastability

- Don't break off an asynchronous input until it has gone through some registers
- Basically: Ensure that external signals feed **exactly one** flip-flop chain before branching





D Register Timing 2



t_{CLK}

CLK

$t_{PD,reg1}$

$t_{CD,reg1}$

$\geq t_{SETUP,reg2}$

sig1

$t_{PD,logic}$

sig2

$\geq t_{HOLD,reg2}$

$t_{CD,logic}$

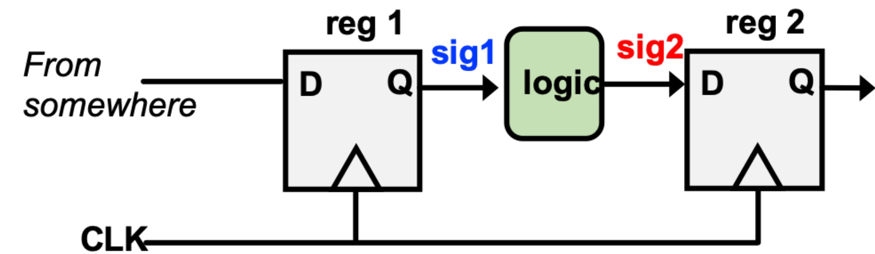
time

Two Requirements/ Conclusions:

$$t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$$

$$t_{CD,reg1} + t_{CD,logic} \geq t_{HOLD,reg2}$$

D Register Timing 2



=determined state

=undetermined state

t_{CLK}

CLK

$t_{PD,reg1}$

$t_{CD,reg1}$

$\geq t_{SETUP,reg2}$

sig1

$t_{PD,logic}$

sig2

$\geq t_{HOLD,reg2}$

$t_{CD,logic}$

time

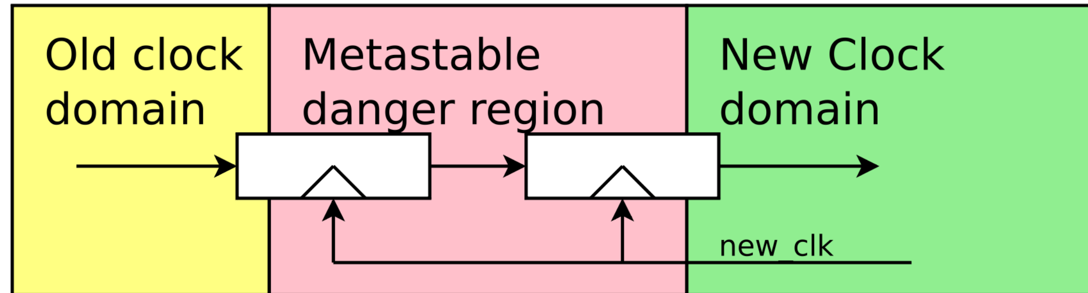
Two Requirements/ Conclusions:

$$t_{PD,reg1} + t_{met} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$$

$$t_{CD,reg1} + t_{CD,logic} \geq t_{HOLD,reg2}$$

Clock Domain Crossing

- For example:
  - Data gets sent in at 25 MHz from one device (running on its own clock)
  - Your system runs at 50 MHz




```
1  //xfer_pipe can be >2 bits wide (2 is usually fine...3 better)
2  always_ff @(posedge new_clock)
3    { new_val, xfer_pipe } <= { xfer_pipe, i_val };
```

- This only works when original clock domain frequency is less than or equal to new clock domain frequency

https://zipcpu.com/blog/2017/10/20/cdc.html