Clocks and Time

Administrative Stuff

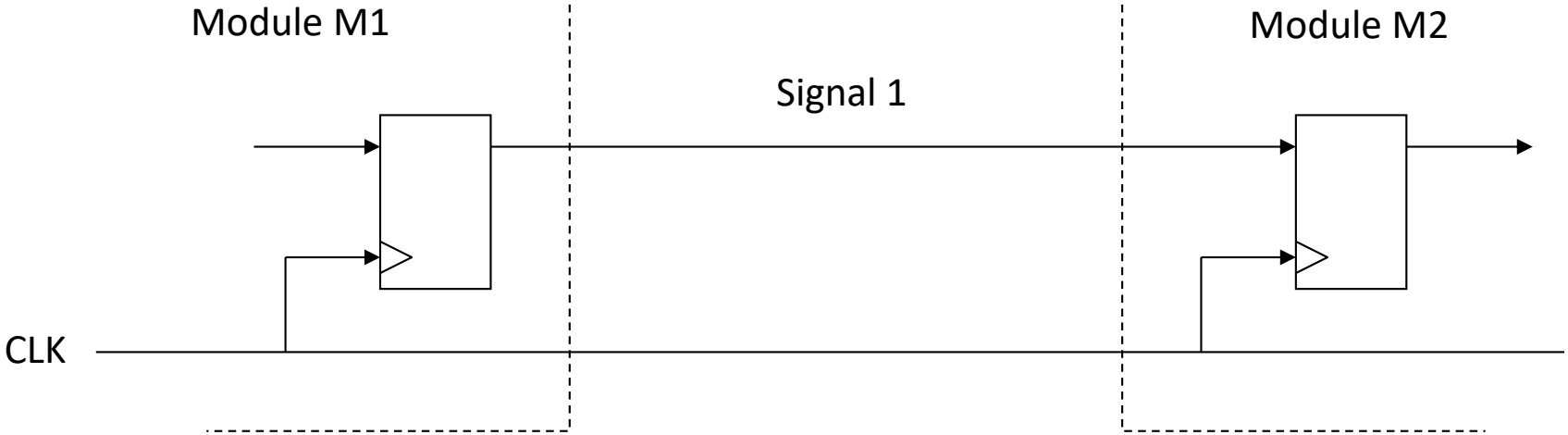
- Pset 08 out today, Due Thursday before class
- Lab 04A due Friday.

- Lab 04B out Thursday (Due following Thursday)
- Pset 09 out Thursday (Due following Thursday)

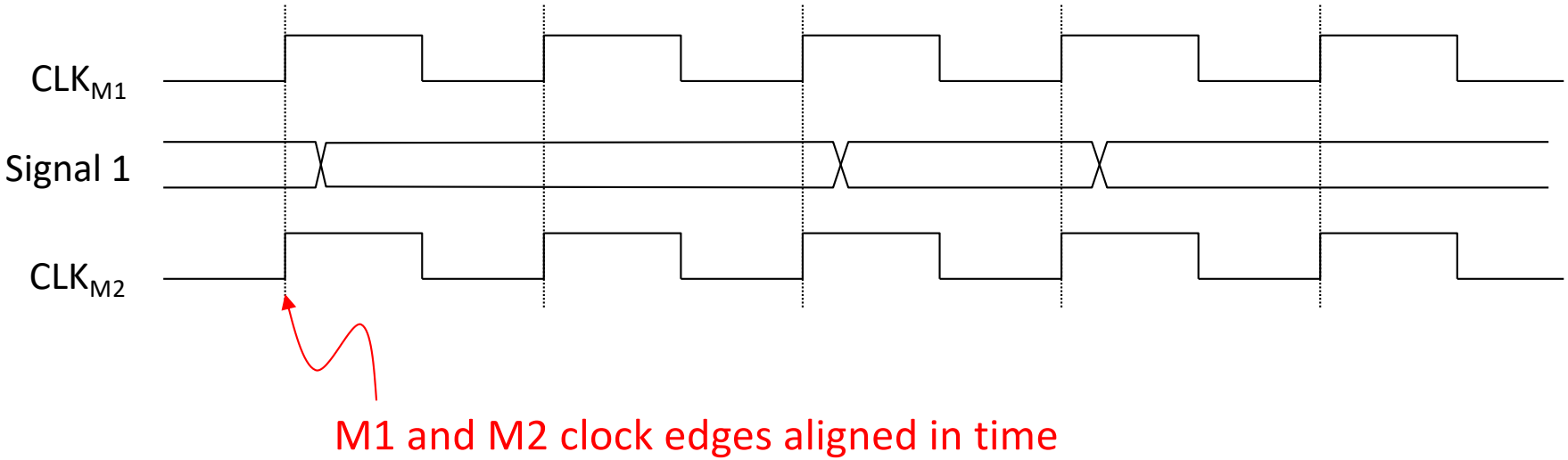
Topics Today

- Clocks and Skew
- Making Clocks
- Slack

Clocking and Synchronous Communication

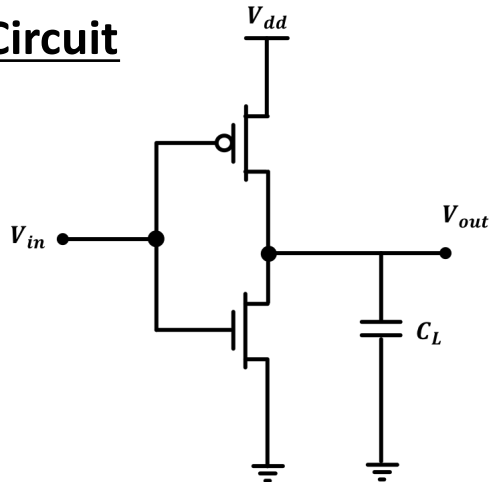


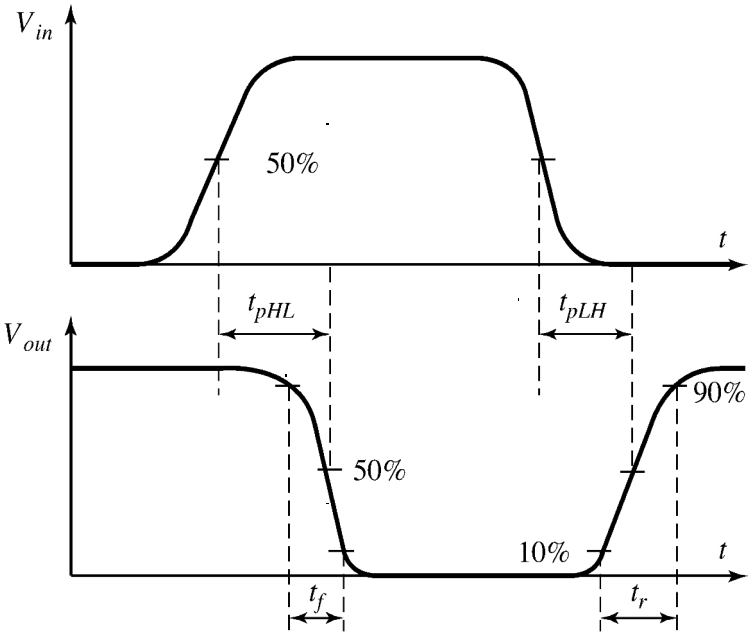
Ideal world:



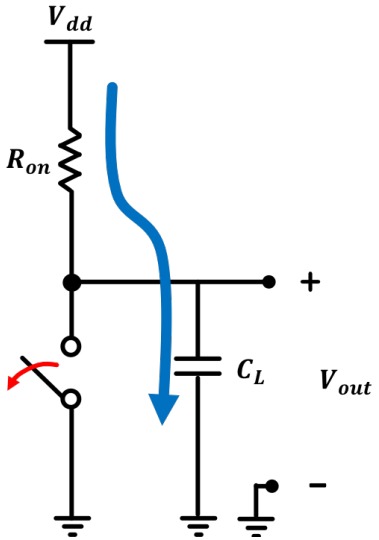
Delay Estimation: Simple RC Networks

Simple CMOS Circuit

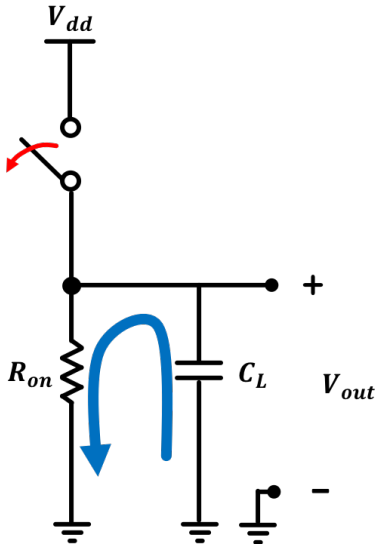




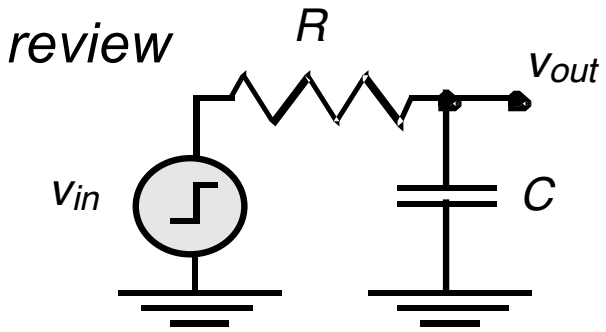
Low-to-High



High-to-Low



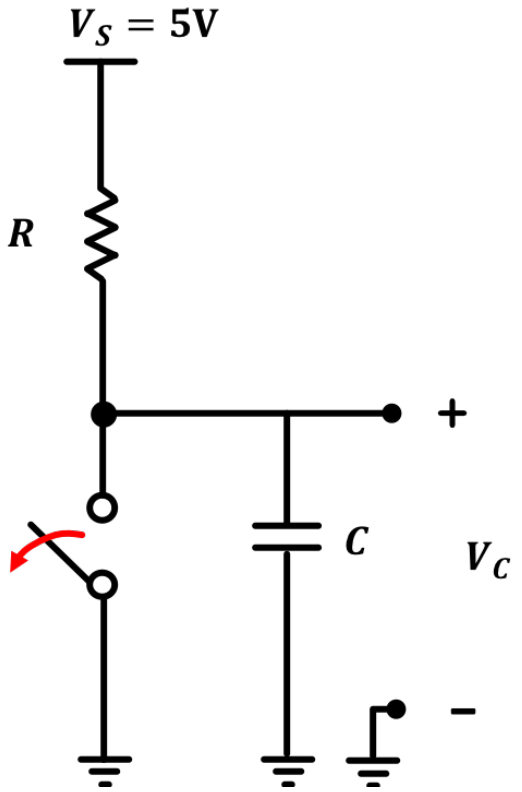
review



$$v_{out}(t) = (1 - e^{-t/\tau})\ V$$

$t_p = \ln(2)\ \tau = 0.69\ RC$

RC Equation



V_s = 5 V

Switch is closed t<0

Switch opens t>0

$V_s = V_R + V_C$

$V_s = i_R R + V_c \quad i_R = C\frac{dV_c}{dt}$

$V_s = RC\frac{dV_c}{dt} + V_c$

$$V_c = V_s\left(1 - e^{-\frac{t}{RC}}\right)$$

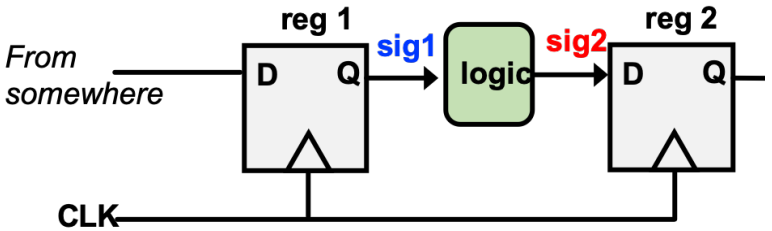
$$V_c = 5\left(1 - e^{-\frac{t}{RC}}\right)$$

So Signals Experience Delays

- ”Signals” generally have their delays expressed with:
  - Contamination Delay
  - Propagation Delay

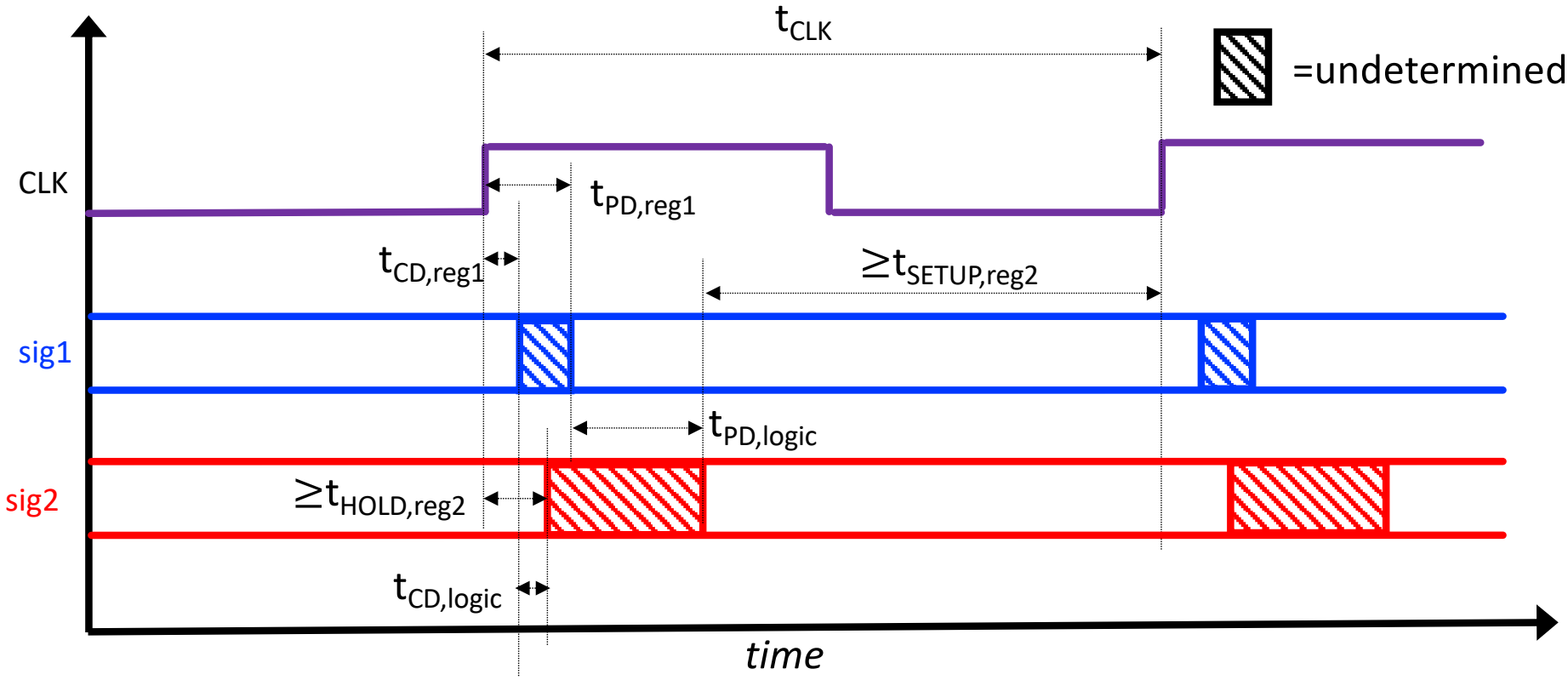
- But the clock experiences Delay too!

jeeze, this diagram again!?



=determined state

=undetermined state

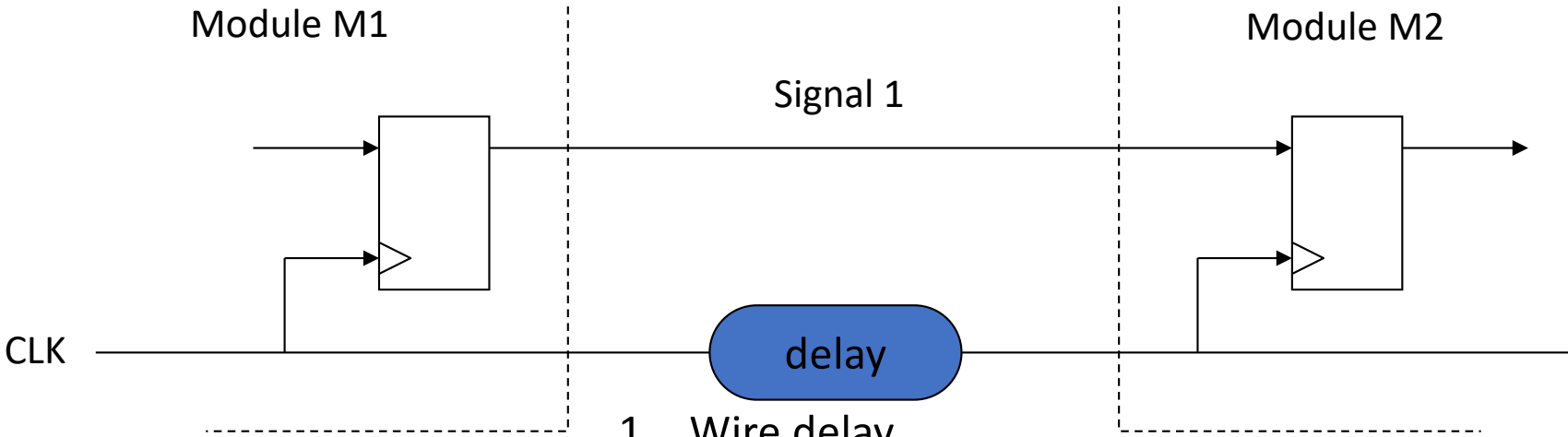


Two Requirements/ Conclusions:

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

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

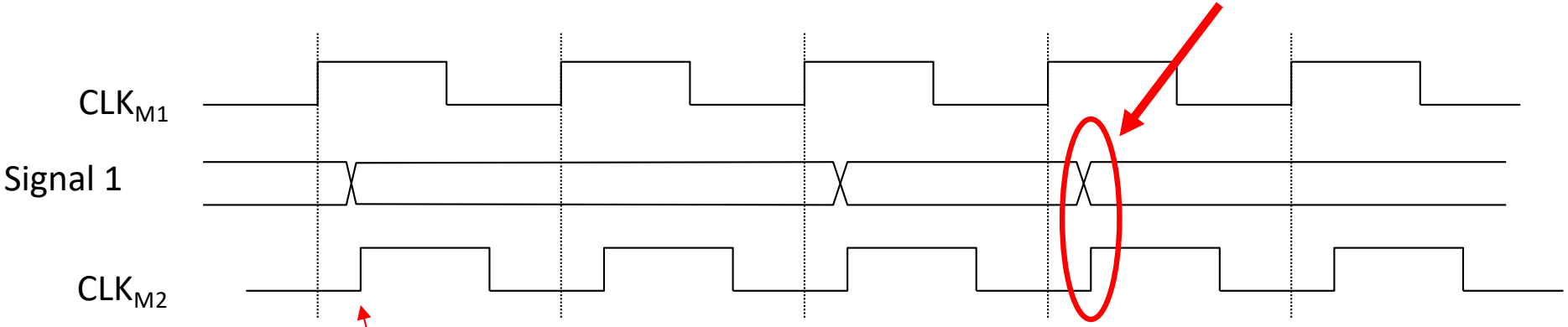
Clock Skew



1. Wire delay
2. Different clocks!

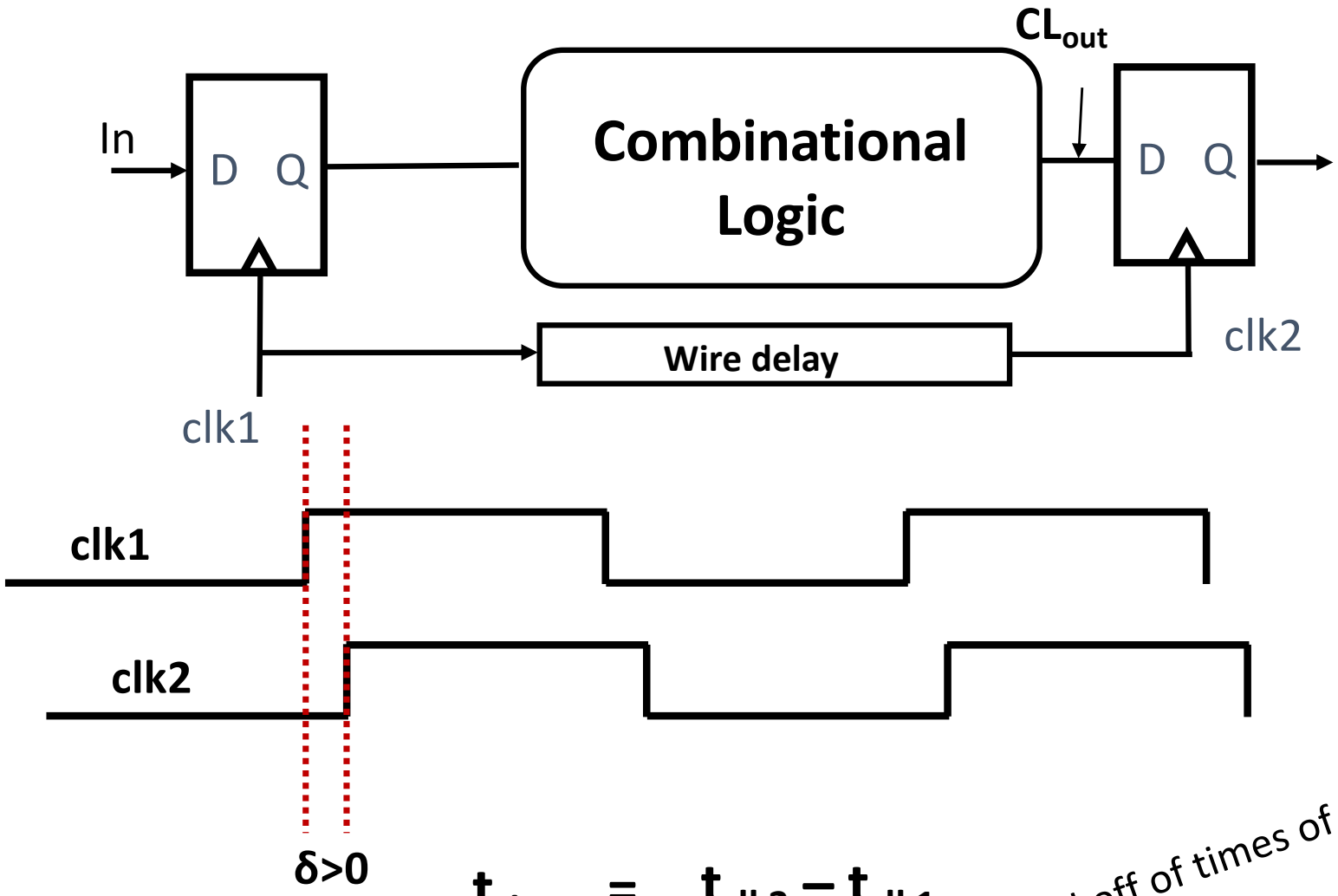
Real world has clock skew:

Oops! Skew has caused a hold time problem!



M2 clock delayed with respect to M1 clock

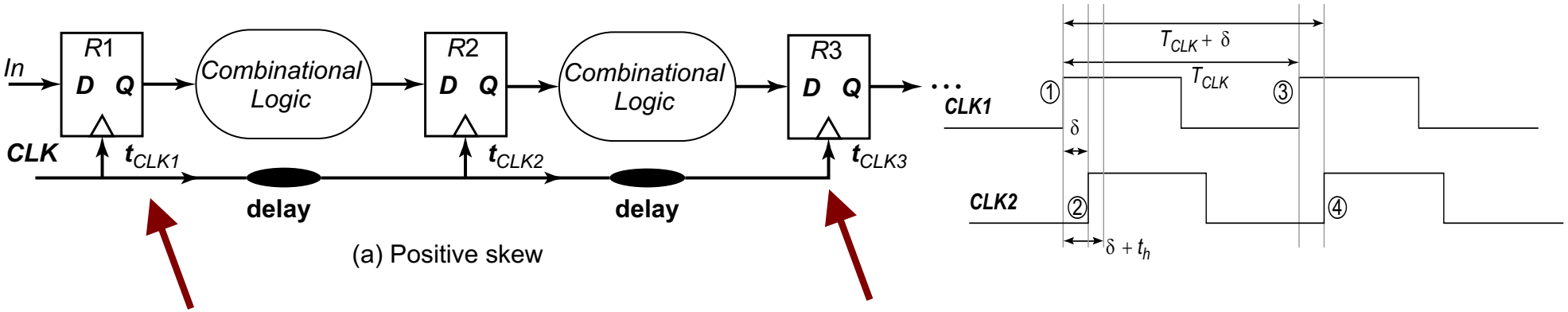
Clocks are Not Perfect: Clock Skew



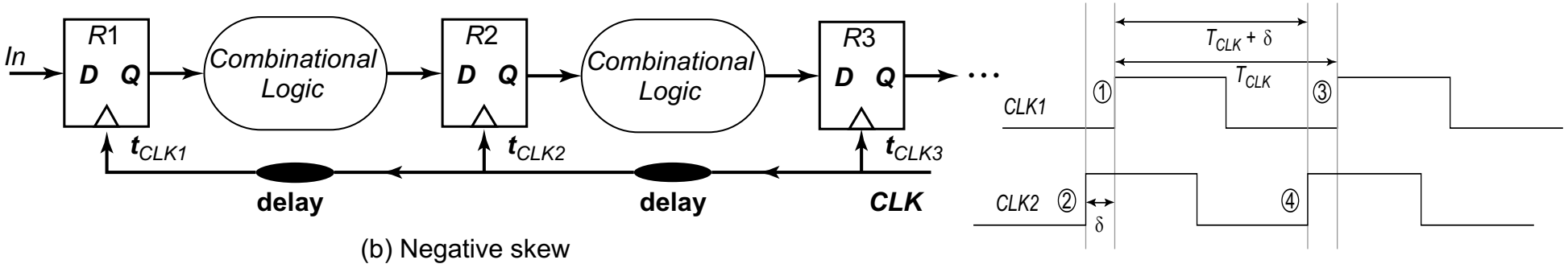
$$t_{skew} = t_{clk2} - t_{clk1}$$

Based off of times of rising edges.
Not periods!

Positive and Negative Skew



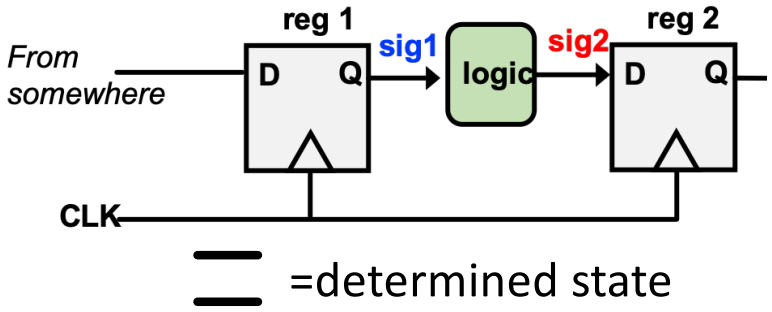
Launching edge arrives before the receiving edge (positive skew)

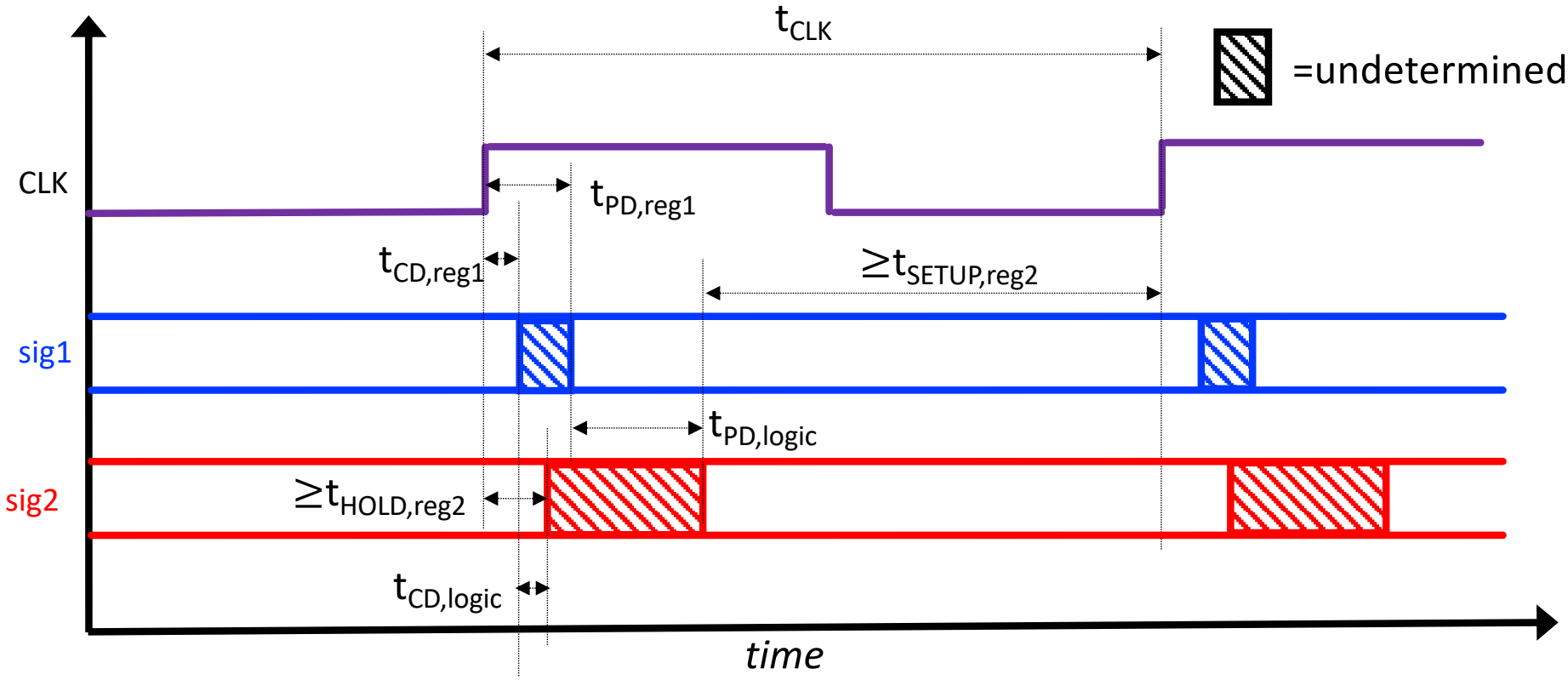


Receiving edge arrives before the launching edge (negative skew)

➢**Adapted from J. Rabaey, A. Chandrakasan, B. Nikolic,**
"Digital Integrated Circuits: A Design Perspective" Copyright 2003 Prentice Hall/Pearson.

Timing



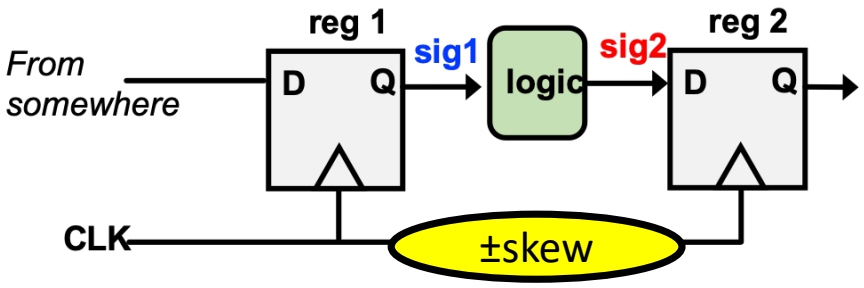


Two Requirements/ Conclusions:

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

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

D-Register Timing With Skew



In the real world the clock signal arrives at different registers at different times. The difference in arrival times (pos or neg) is called the *clock skew* t_{skew}.

$t_{skew} = t_{Rn,clk2} - t_{Rn,clk1}$

We can update our two timing constraints to reflect the worst-case skew

Setup time: $t_{Rn,clk} = t_{Rn+1,clk}$

$t_{Rn,clk1}+t_{PD,reg1}+t_{PD,logic}+t_{SETUP,reg2} \leq t_{Rn+1,clk2}$

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

Hold time:

$t_{Rn,clk1}+t_{CD,reg1}+t_{CD,logic} \geq t_{Rn,clk2}+t_{HOLD,reg2}$

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

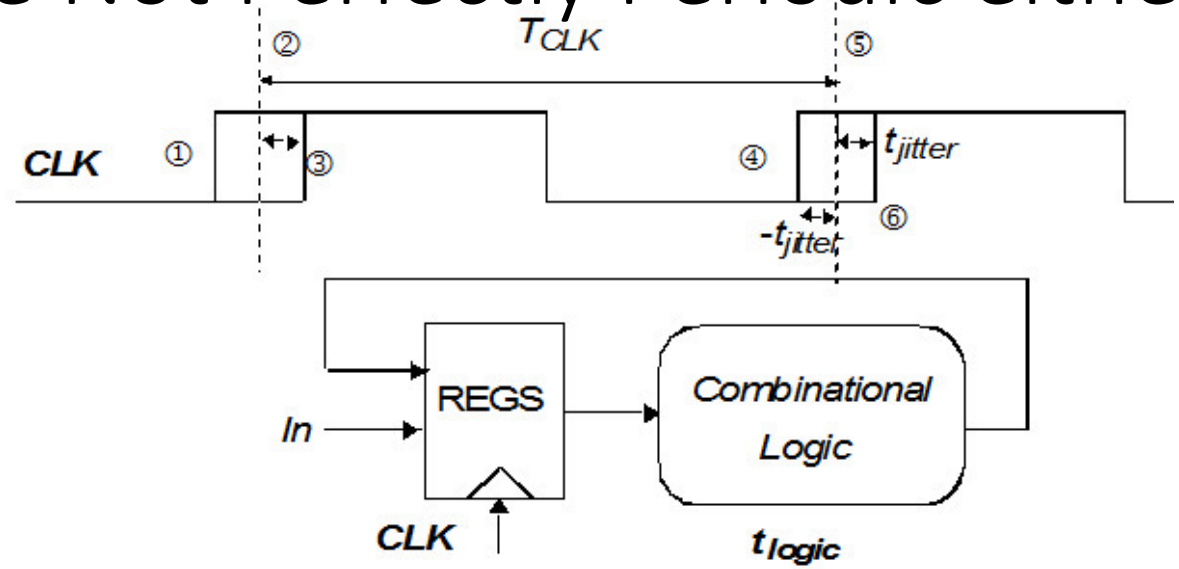
Thus positive clock skew improves the minimum cycle time of our design but makes it harder to meet register hold times.

Thus negative clock skew hurts the minimum cycle time of our design but makes it easier to meet register hold times.

Which skew is tougher to deal with (pos or neg)?

Positive skew

Clocks Are Not Perfectly Periodic either: Jitter



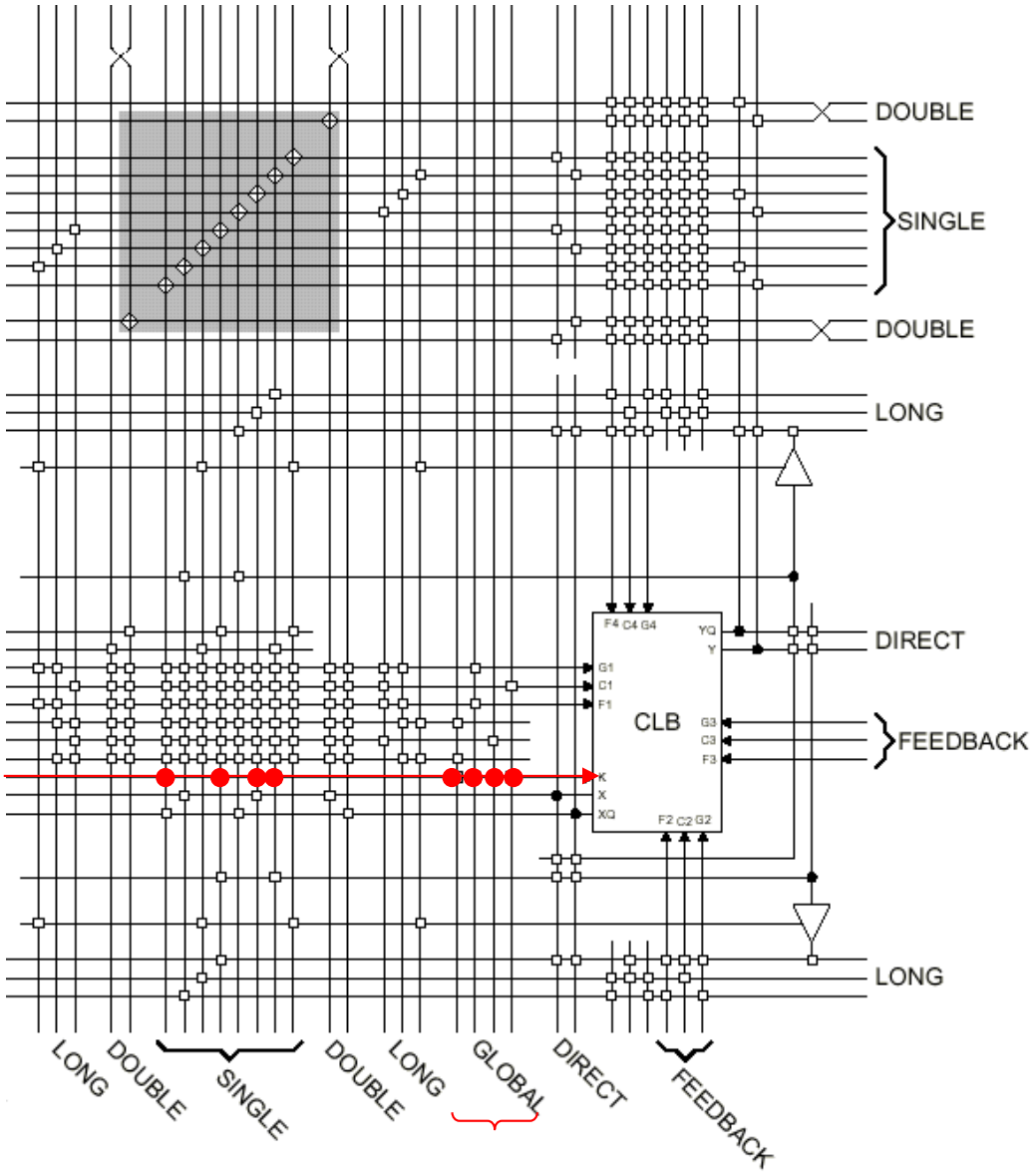
- Jitter is an approximation of how much the clock period can increase/decrease:
- Can make it harder to meet timing since it effectively shortens t_{clk}

$$t_{PD,reg1}+t_{PD,logic}+ t_{SETUP,reg2} \leq t_{CLK} - 2t_{jitter}$$

Typical crystal oscillator
100mhz (10ns)
Jitter: 1ps

Low-skew Clocking in FPGAs

- When Vivado is doing place-and-route it tries to position logic so that skew is minimized wherever possible
- Special clock paths and buffers exist throughout the chip to distribute the clock as effectively as possible.



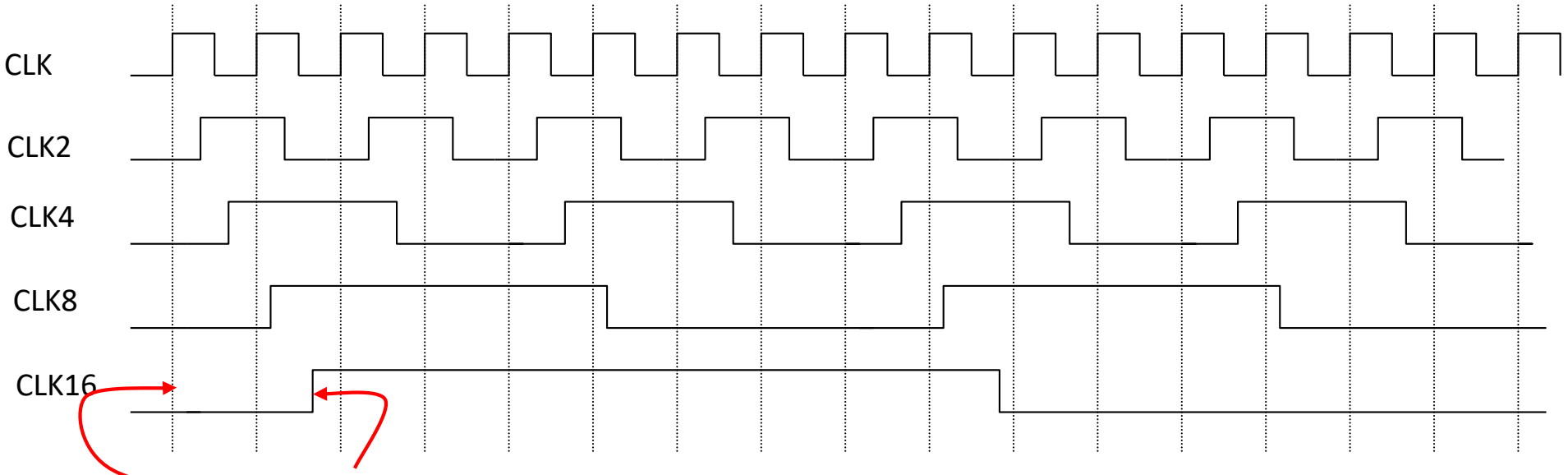
Figures from Xilinx App Notes

Goal: use as few clock domains as possible

Suppose we wanted signals at f/2, f/4, f/8, etc.:

No! don't do it this way

```
logic clk2,clk4,clk8,clk16;
always_ff @(posedge clk) clk2 <= ~clk2;
always_ff @(posedge clk2) clk4 <= ~clk4;
always_ff @(posedge clk4) clk8 <= ~clk16;
always_ff @(posedge clk8) clk16 <= ~clk16;
```



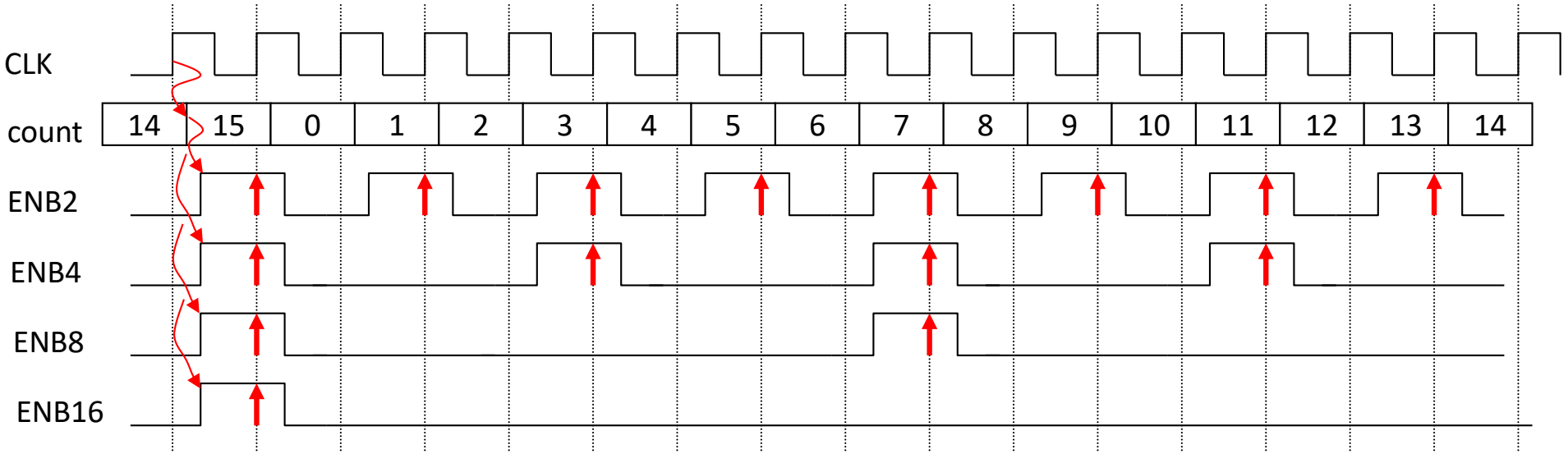
Very hard to have synchronous communication between clk and clk16 domains… Can lead to lots of timing violations!

Solution: One clock, Many enables

Use one (high speed) clock, but create enable signals to select a subset of the edges to use for a particular piece of sequential logic (much easier on timing requirements)

```
logic [3:0] count;
always_ff @(posedge clk) count <= count + 1;   // counts 0..15
logic enb2, enb4, enb8, enb16;
assign enb2 = (count[0] == 1'b1);
assign enb4 = (count[1:0] == 2'b11);
assign enb8 = (count[2:0] == 3'b111);
assign enb16 = (count[3:0] == 4'b1111);
```

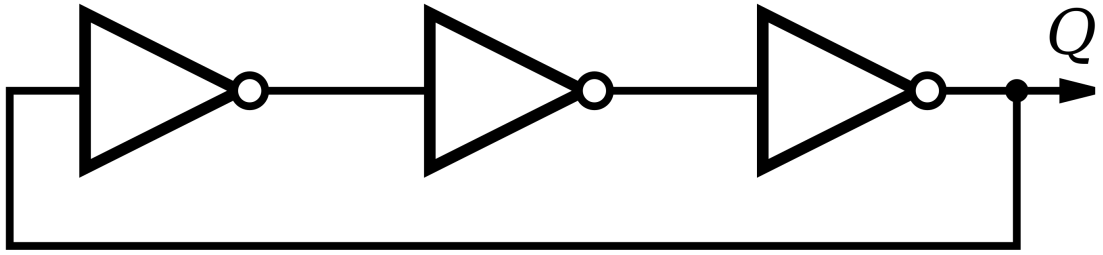
```
always_ff @(posedge clk)
  if (enb2) begin
    // get here every 2nd cycle
  end
```



= clock edge selected by enable signal

How to Make Frequencies

Where do we get frequencies?

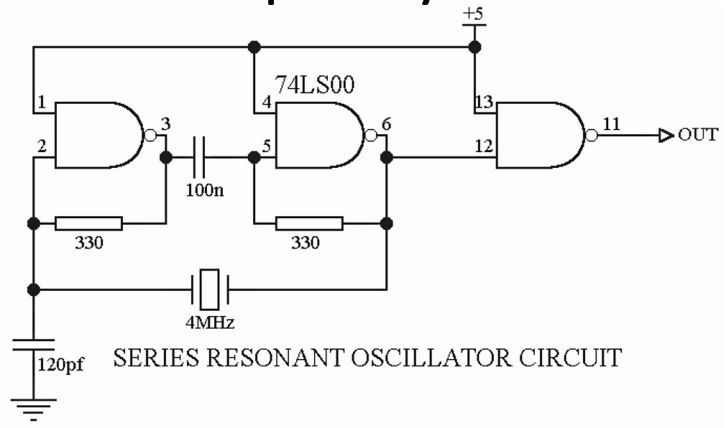


- Particular combinational circuits that are fed back onto themselves so that they cannot be stable can be made to form oscillators.
- The ring oscillator above is a classic example.
- There is no stable set of output states so this circuit perpetually oscillates.
- Period of oscillation is based on the delay of each element

Where do we get frequencies?

16MHz Crystal

- Most frequencies come from Crystal Oscillators made of quartz
- Equivalent to very High-Q LRC tank circuits
- https://en.wikipedia.org/wiki/Crystal_oscillator_frequencies
- Incorporate into circuit like that below and boom, you've got a square wave of some specified frequency dependent largely on the crystal



http://www.z80.info/uexosc.htm

https://en.wikipedia.org/wiki/Crystal_oscillator

High Frequencies

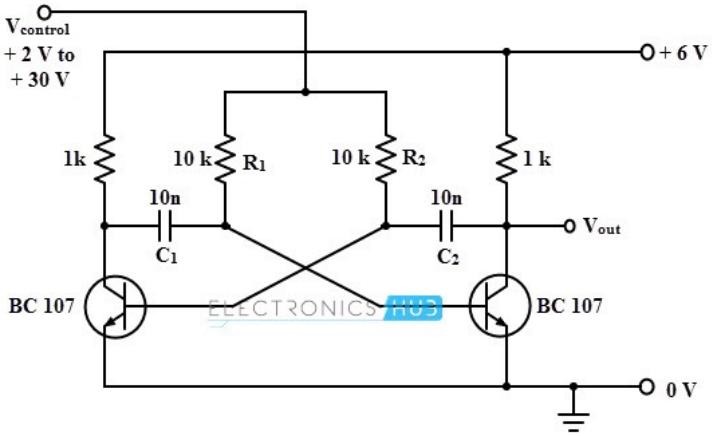
- Very hard to get a crystal oscillator to operate above ~200 MHz (7^{th} harmonic of resonance of crystal itself, which usually is limited to about 30 MHz due to fabrication limitations)

- Where does the 2.33 GHz clock of my iPhone come from then?

- Frequency Multipliers!

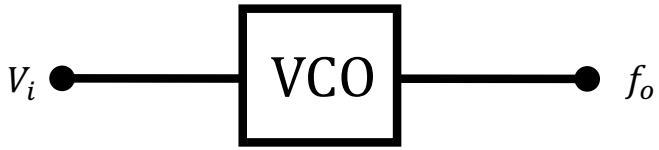
Voltage Controlled Oscillator

- It is very easy to make voltage-controlled oscillators that run up to 1GHz or more.
  - Low voltage circuit oscillates at low frequency
  - Higher voltage→higher frequency oscillation



A simple VCO (not type found in FPGA)

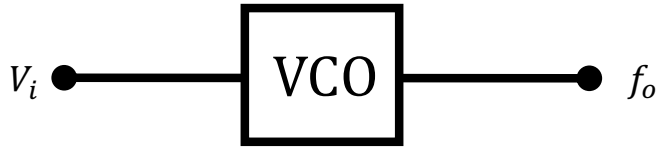
- Block Diagram



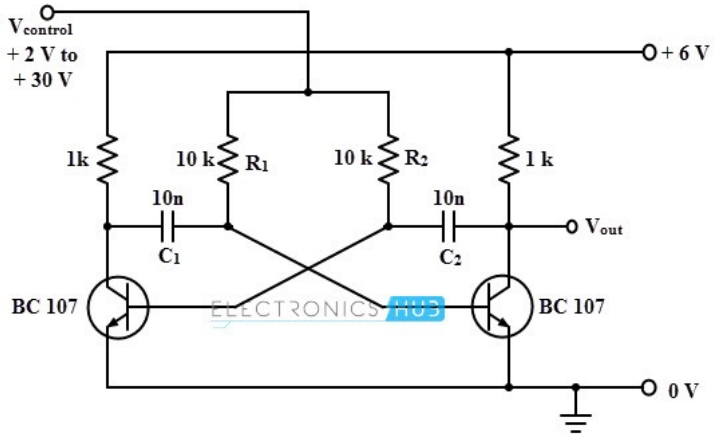
http://www.electronicshub.org/voltage-controlled-oscillators-vco/

Voltage Controlled Oscillator

- It is very easy to make voltage-controlled oscillators that run up to 1GHz or more.
- Why don't we just:



- Pick the voltage V_i that is needed to get the frequency we want f_o?
- That's gotta be specified right?

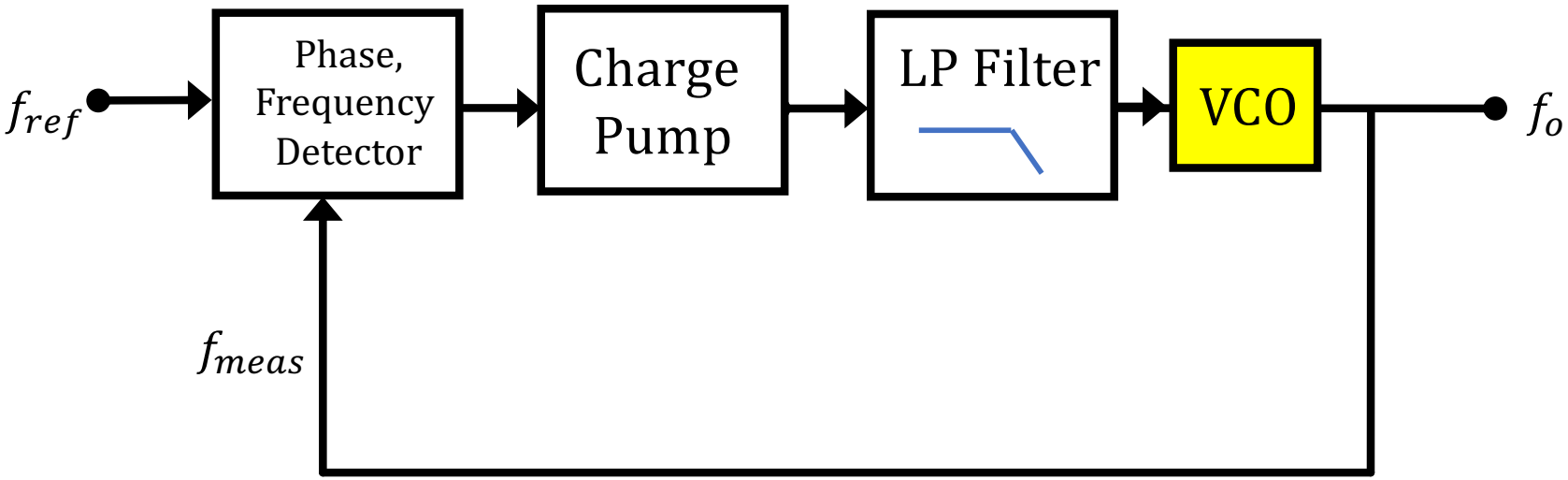


A simple VCO (not type found in FPGA)

http://www.electronicshub.org/voltage-controlled-oscillators-vco/

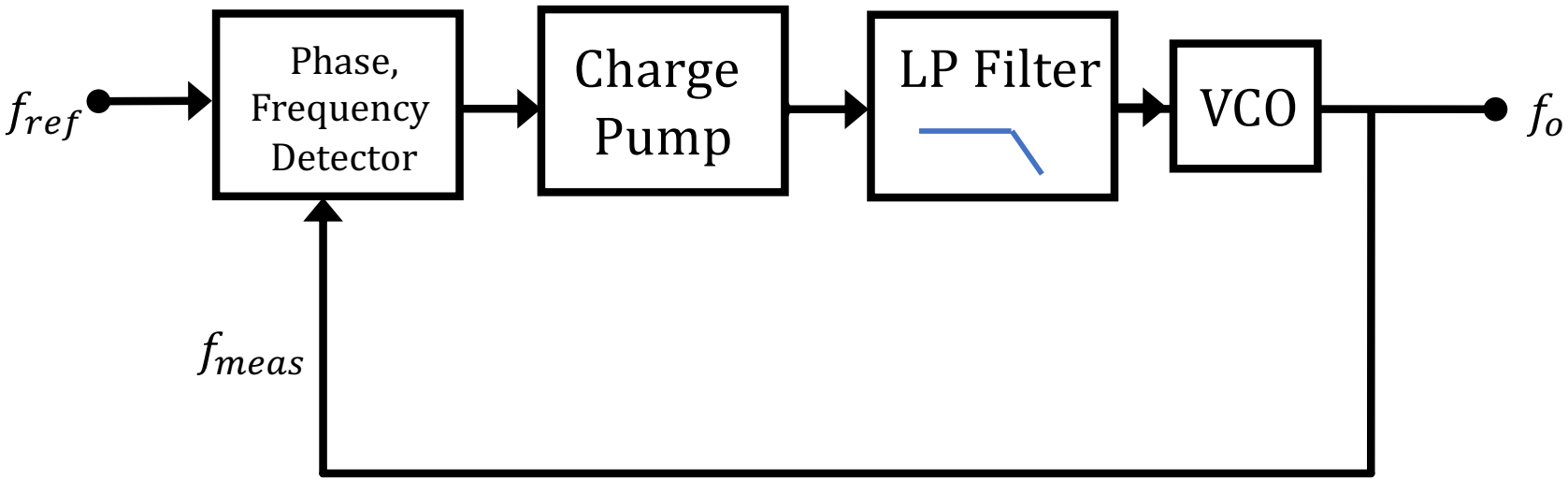
Phase Locked Loop

- Place the unstable, but capable VCO in a feedback loop.
- This type of circuit is a phase-locked loop variant

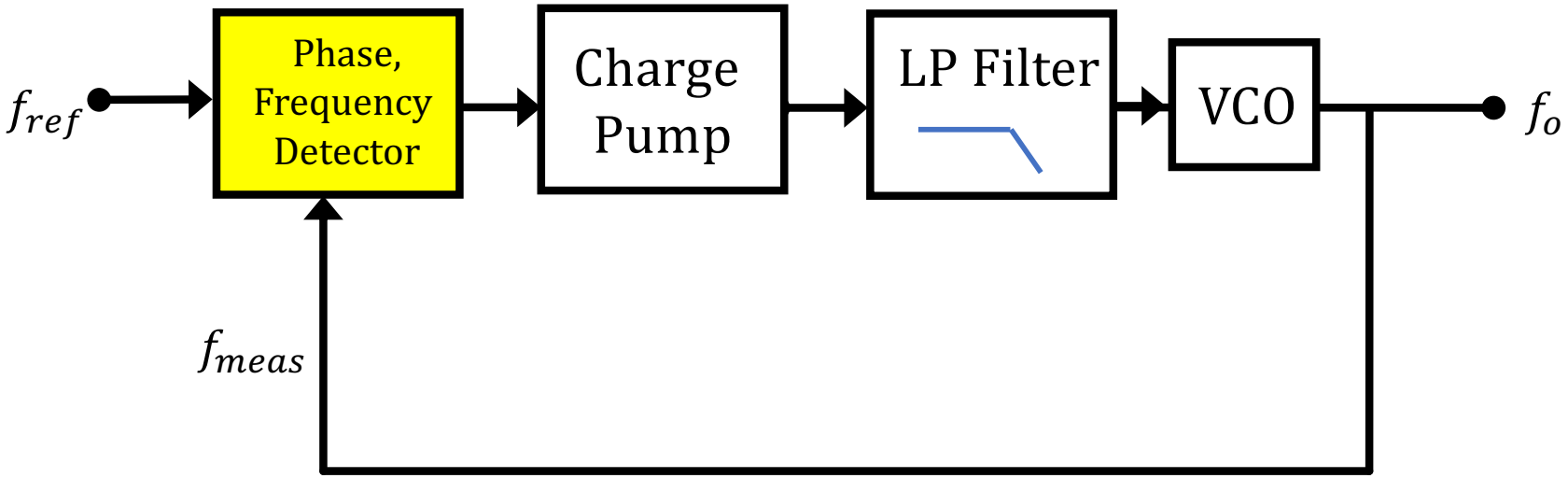


Phase Locked Loop

- Circuit that can track an input phase of a system and reproduce it at the output

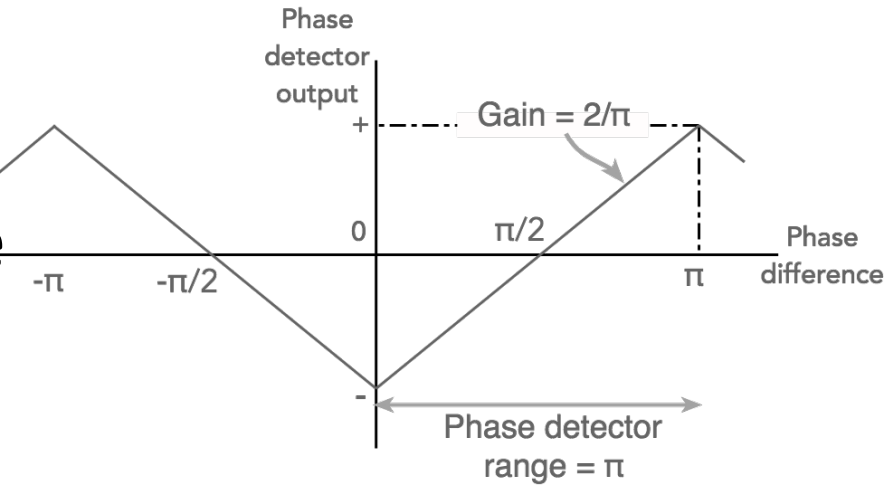


Phase, Frequency Detector



Phase Detector

- Can be a simple XOR, XNOR gate
  - Low-pass the output



- If near the desired frequency already this can work…if it is too far out, it won't and can be very unreliable since phase and frequency are related but not quite the same thing, it will lock onto harmonics, etc…

- For frequency we instead use a PFD:
  - Phase/Frequency Detector:

Phase-Frequency Detection

- Detects both change and which clock signal is consistently leading the other one
- Using MOSFETs you charge/discharge a capacitor accordingly which also with some resistors low-pass filters the signal
- The output voltage is then roughly proportional to the frequency error!

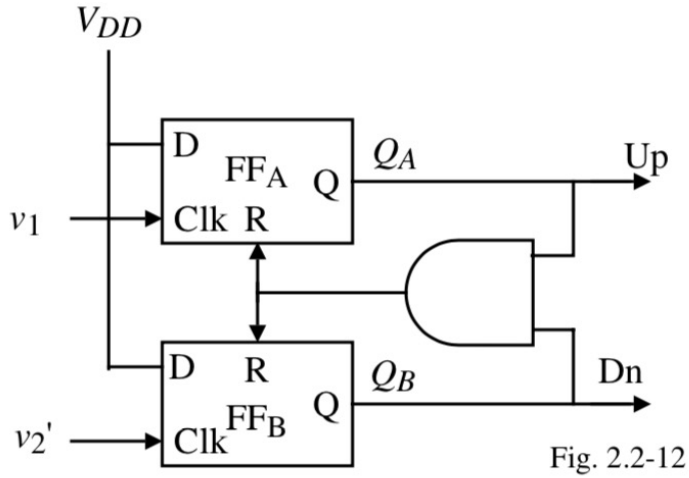


Fig. 2.2-12

The R input is the Reset of the flipflop When this is asserted, it sets Q back to 0 IMMEDIATELY

http://www.globalspec.com/reference/72819/203279/2-7-phase-detectors-with-charge-pump-output

Phase Frequency Detection

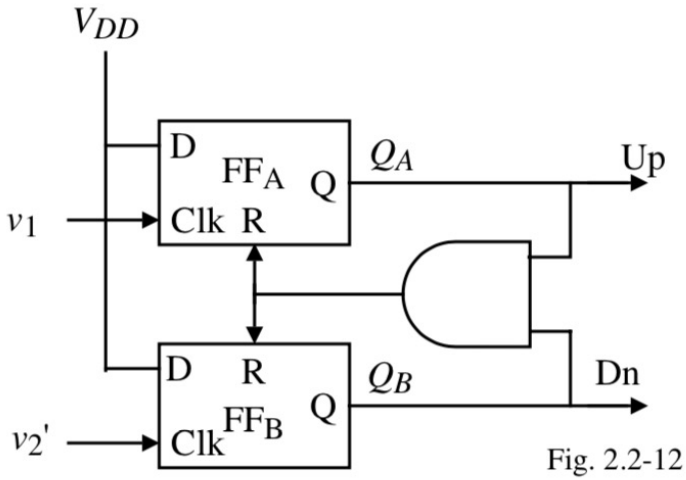


Fig. 2.2-12

- Clock 1 and clock 2 are constantly competing with one another to generate up and down signals
- The up signals charge up a capacitors through a pair of transistors…the down signal discharges the capacitor

PFD State Diagram:

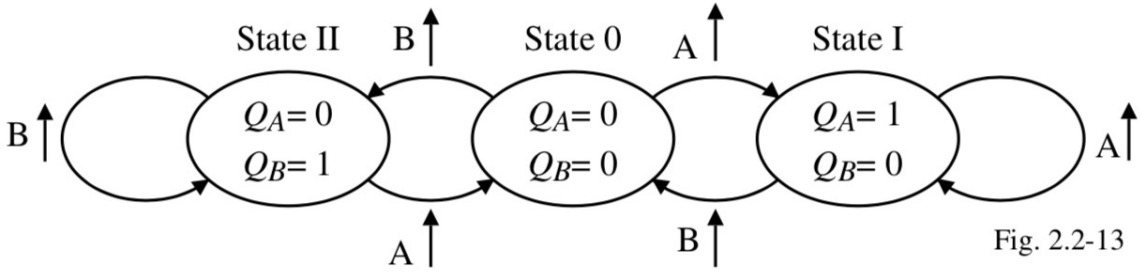


Fig. 2.2-13

1.pallen.ece.gatech.edu/Academic/ECE_6440/Summer_2003/L070-DPLL(2UP).pdf

Phase-Frequency Detection

- If you're in State I:
  - Increase voltage on capacitor
- If you're in State II:
  - Decrease voltage on capacitor
- The voltage that builds up will be tightly related to how different these two circuits are

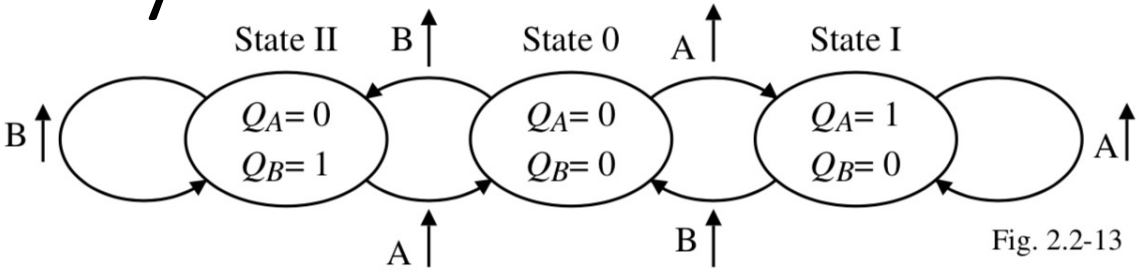
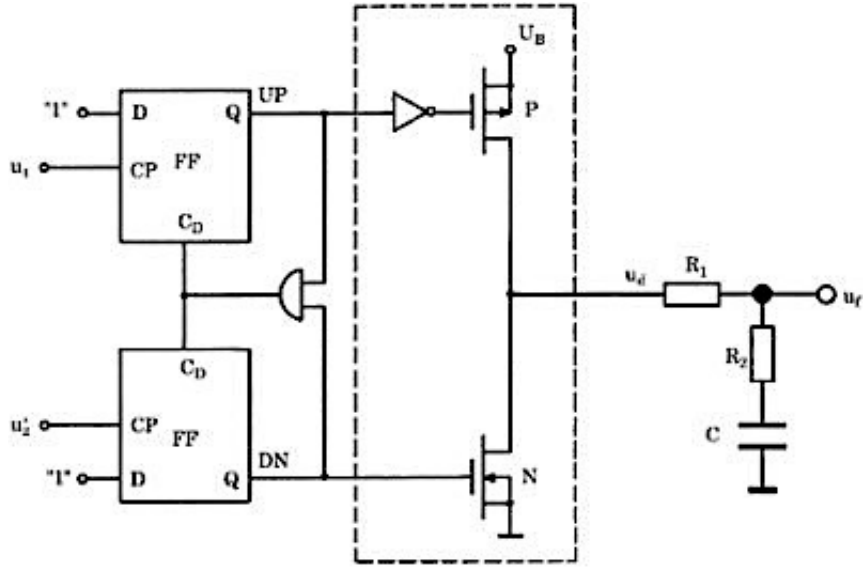


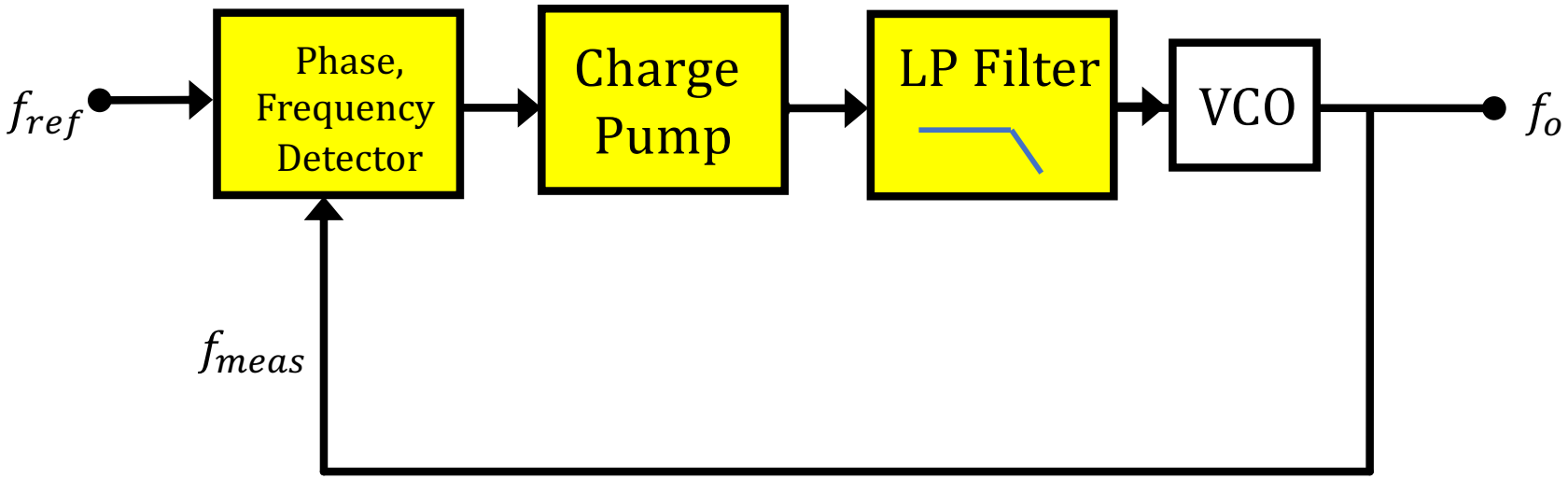
Fig. 2.2-13



PFD | Charge Pump | LPF

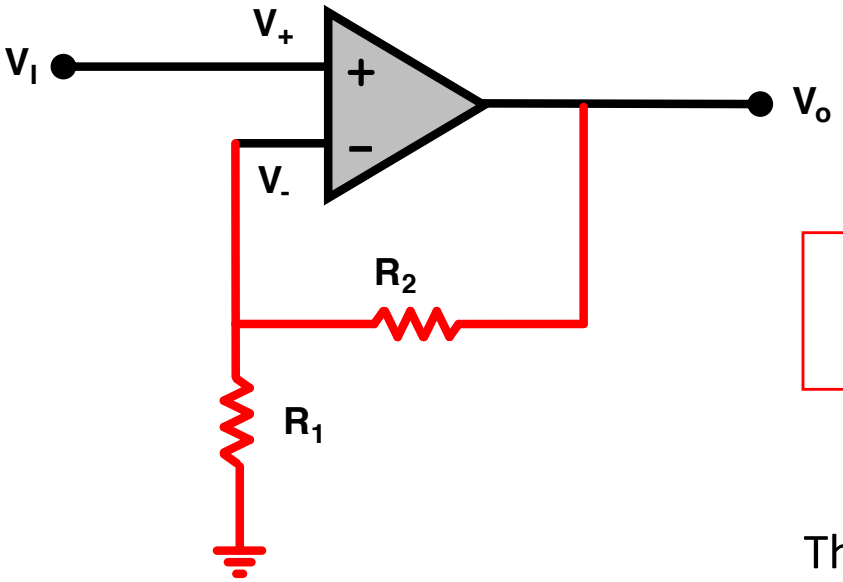
http://www.globalspec.com/reference/72819/203279/2-7-phase-detectors-with-charge-pump-output

PFD, Charge Pump, LP Filter



- So this circuit can make $f_0 = f_{ref}$ That doesn't help us!
- How can we make a higher frequency?

Use Resistors in Voltage Divider in Feedback Path!



- A voltage divider in feedback path gives us voltage gain!

$$K = \frac{1}{1 - p + G} \quad p \approx 0.9999 \text{ means} \quad K = \frac{1}{G}$$

$$G = \frac{R_1}{R_1 + R_2}$$

The gain A_v of this circuit is therefore:

$$A_v = \frac{R_1 + R_2}{R_1}$$

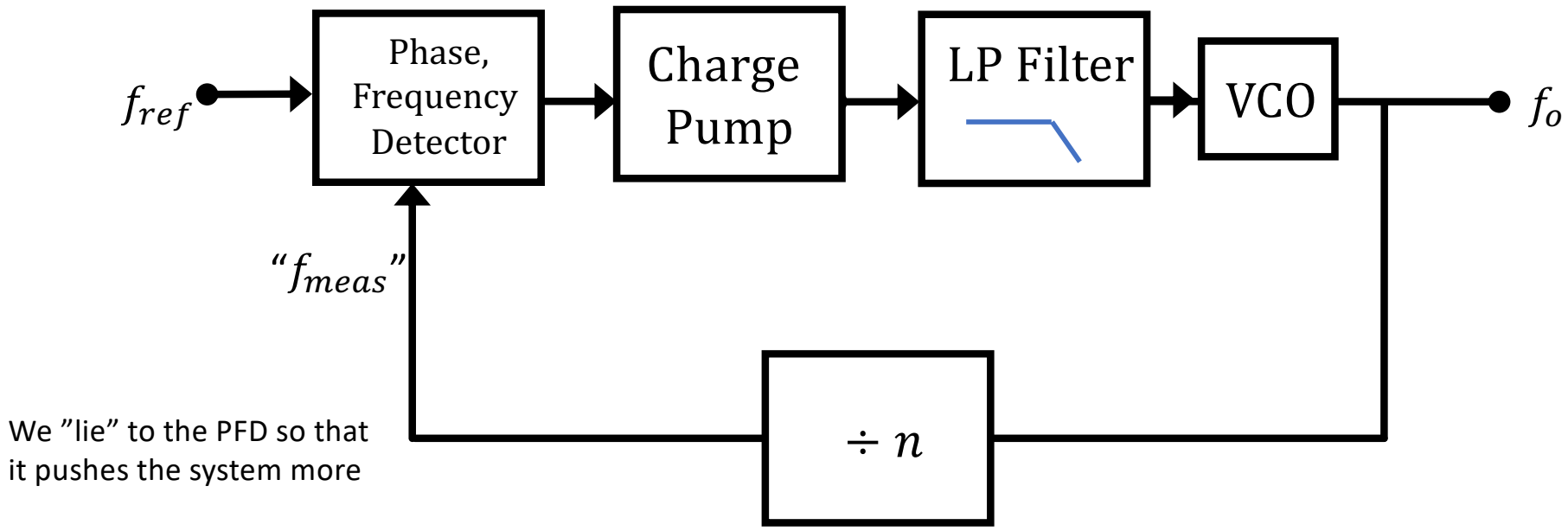
The gain of a "non-inverting amplifier"

$$V_- = V_o \frac{R_1}{R_1 + R_2}$$

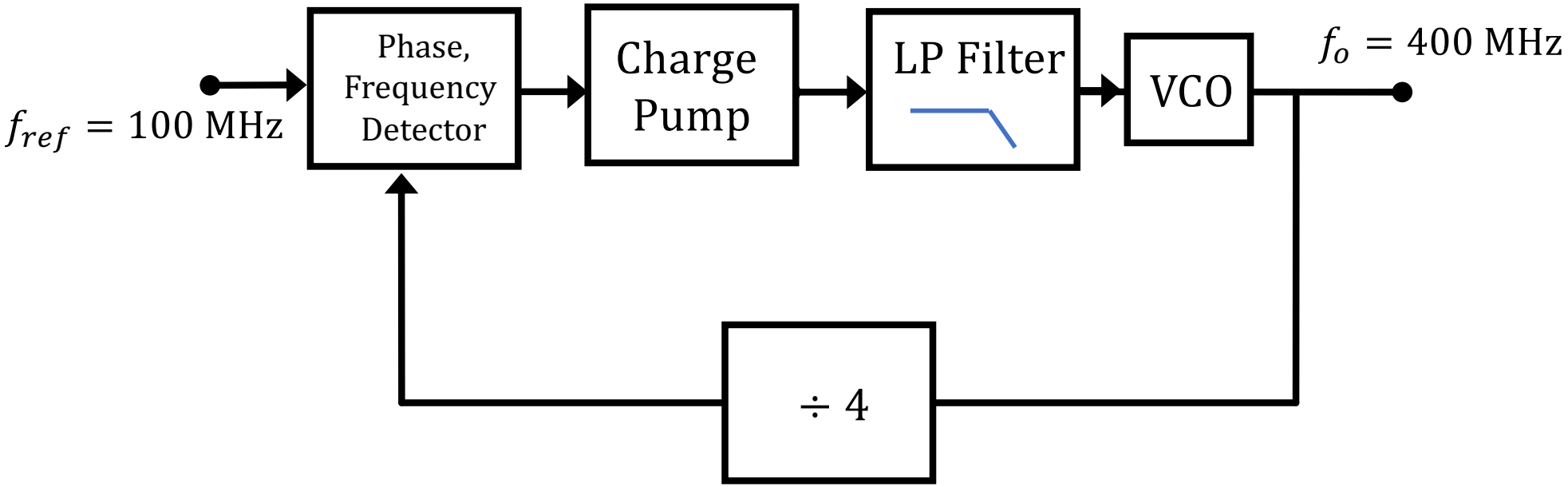
Same Idea with Phase Locked Loops!

Use a Clock Divider in Feedback Path!

- A clock divider in feedback path gives us clock gain!

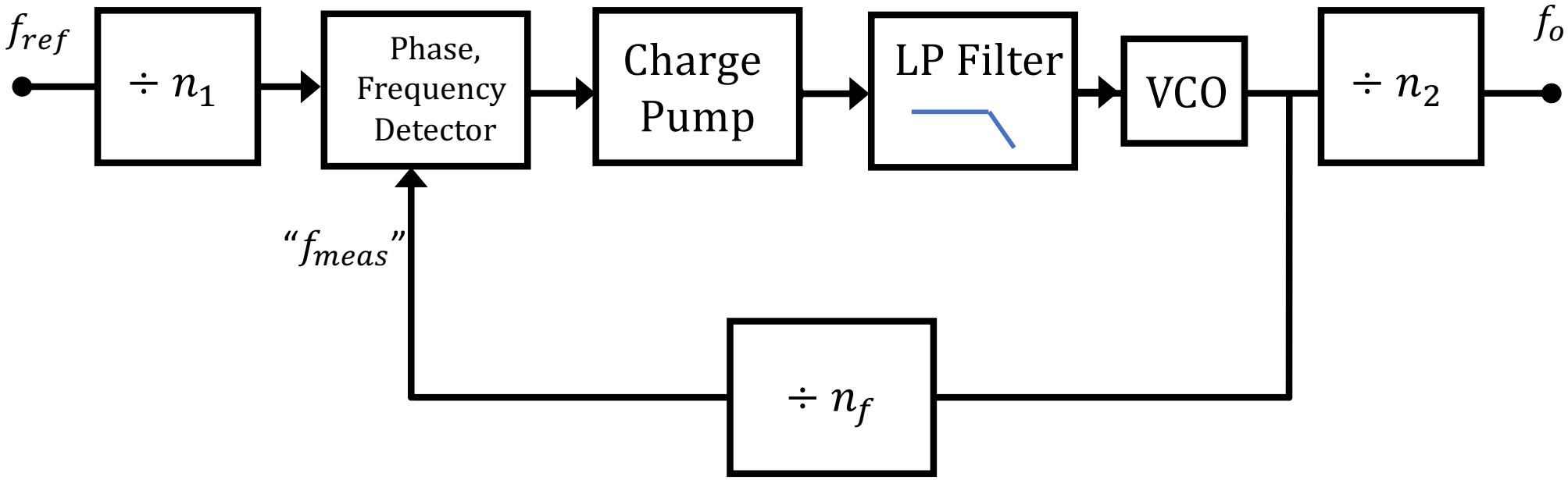


Use a Clock Divider in Feedback Path!




```
logic clk2,clk4,clk8,clk16;
always_ff @(posedge clk) clk2 <= ~clk2;
always_ff @(posedge clk2) clk4 <= ~clk4;
always_ff @(posedge clk4) clk8 <= ~clk16;
always_ff @(posedge clk8) clk16 <= ~clk16;
```

Add a Pre- and Post- Divider for Flex



So to Make 65 MHz?

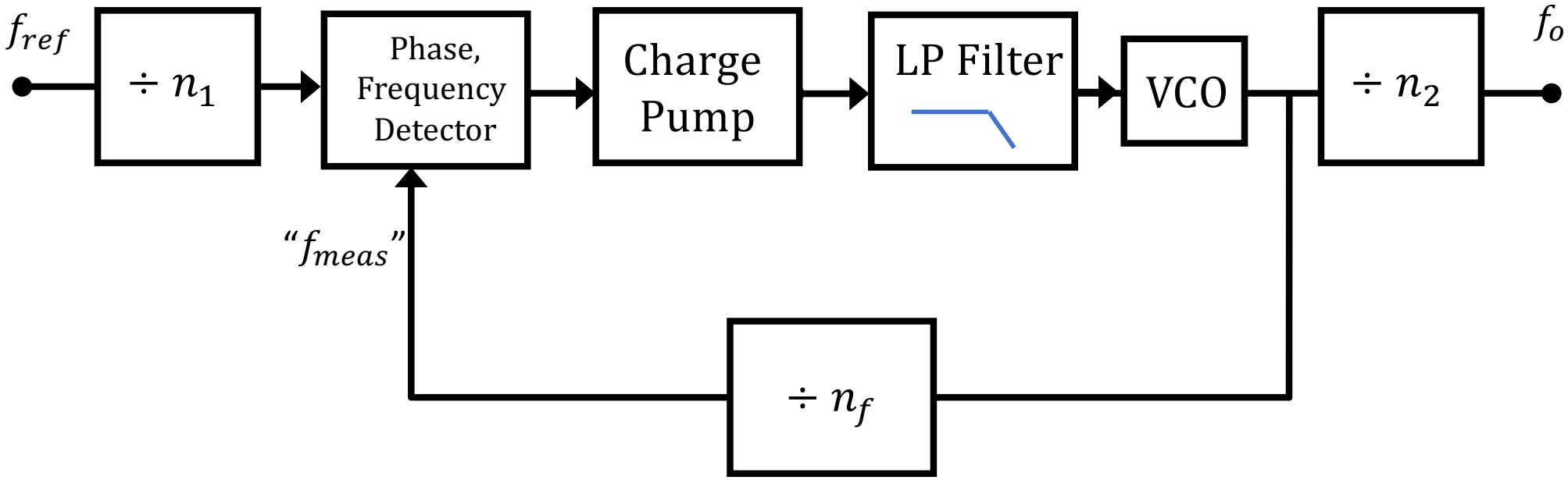
- How to make 65 MHz from 100 MHz?
  - Divide down (not too low)
  - Multiply up (not too high)
  - Divide down for final product

So to Make 65 MHz?

- How to make 65 MHz from 100 MHz?

```
MMCME2_ADV
#(.BANDWIDTH ("OPTIMIZED"),
.CLKOUT4_CASCADE ("FALSE"),
.COMPENSATION ("ZHOLD"),
.STARTUP_WAIT ("FALSE"),
.DIVCLK_DIVIDE (5),
.CLKFBOUT_MULT_F (50.375),
.CLKFBOUT_PHASE (0.000),
.CLKFBOUT_USE_FINE_PS ("FALSE"),
.CLKOUT0_DIVIDE_F (15.500),
.CLKOUT0_PHASE (0.000),
.CLKOUT0_DUTY_CYCLE (0.500),
.CLKOUT0_USE_FINE_PS ("FALSE"),
.CLKIN1_PERIOD (10.0))
mmcm_adv_inst
// Output clocks
(
.CLKFBOUT (clkfbout_clk_wiz_0),
.CLKFBOUTB (clkfboutb_unused),
```

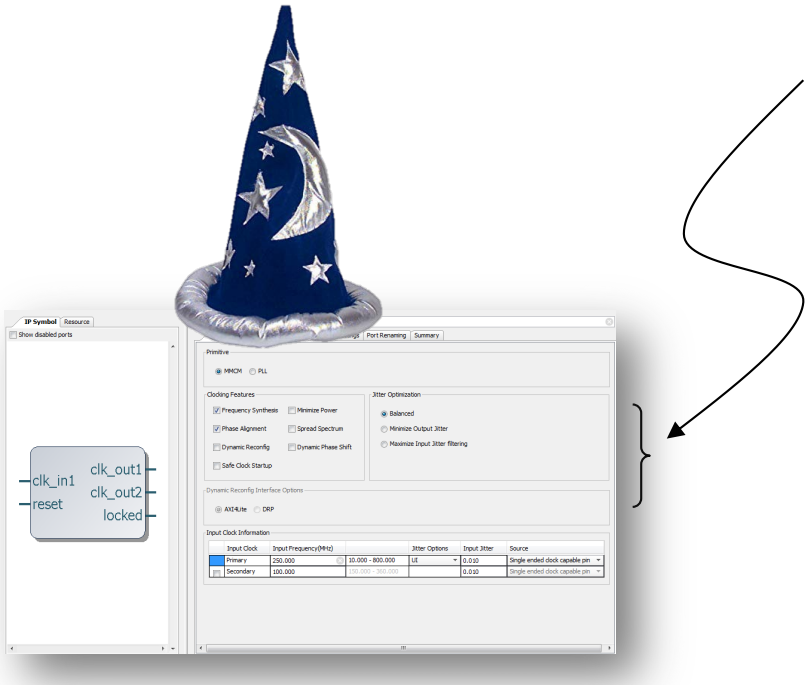
Add a Pre- and Post- Divider for Flex



n_f and n_2 can generally be fractions by switching between several dividers with a weighted average

Generating Other Clock Frequencies (again)

The Nexys4 board has a 100MHz crystal (10ns period). Use "clock wizard" to generate other frequencies e.g., 65MHz to generate 1024x768 VGA video.



Clock Wizard can also synthesize certain multiples/fractions of the CLKIN frequency (100 MHz):

$$f_{CLKFX} = \left(\frac{M}{D}\right) f_{CLKIN}$$

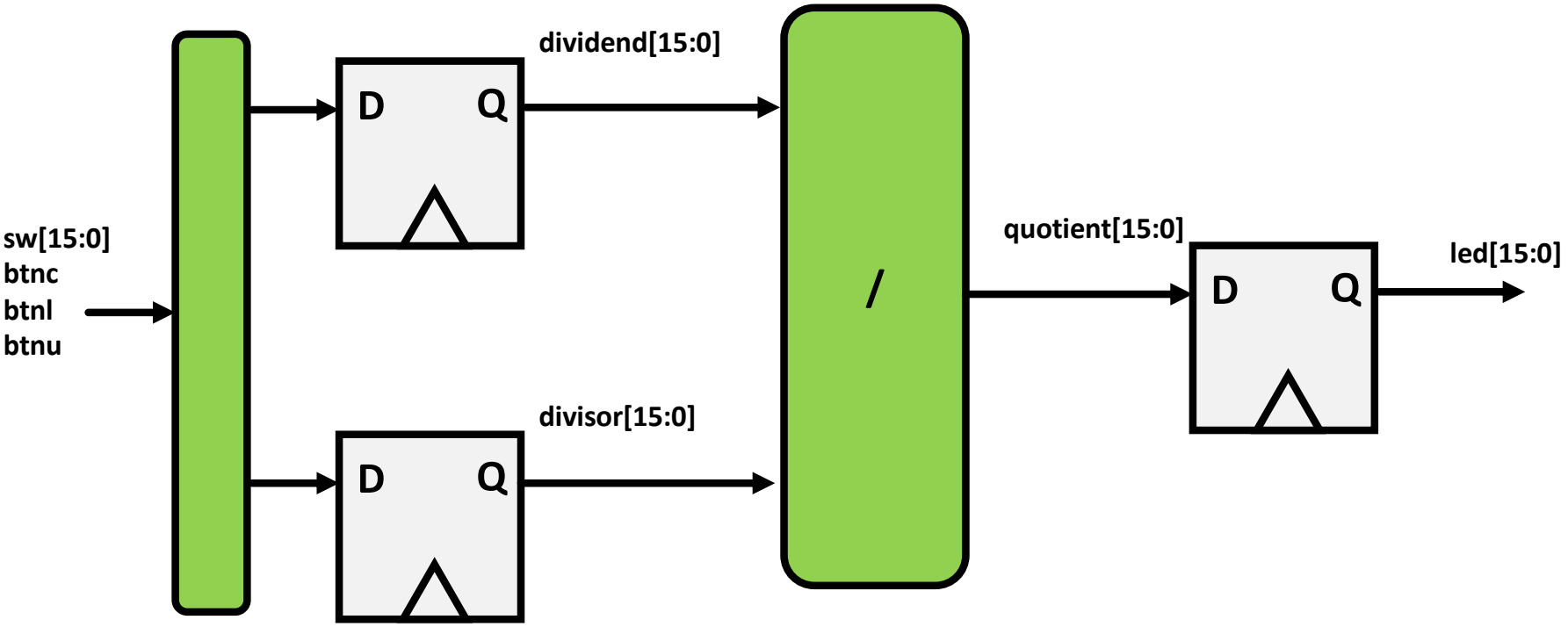
Timing in Vivado

Starting to Look

Let's Look at Some Code:

- Very Simple top_level:
- Use sw[15:0] and buttons to seed two values into 16 bit registers:
  - Dividend
  - Divisor
- When btnu is pushed:
  - DIVIDE the 16 bit numbers

```
module top_level(
      input wire clk_100mhz, //clock @ 100 mhz
      input wire [15:0] sw, //switches
      input wire btnc, //btnc (used for reset)
      input wire btnu, //btnc (used for reset)
      input wire btnl, //btnc (used for reset)
      output logic [15:0] led //just here for the funs
      );
  logic old_btnl;
  logic old_btnu;
  logic old_btnc;
  logic [15:0] quotient;
  logic [15:0] dividend;
  logic [15:0] divisor;
  assign led = quotient;
  always_ff @(posedge clk_100mhz)begin
    old_btnl <= btnl;
    old_btnu <= btnu;
    old_btnc <= btnc;
  end

  always_ff @(posedge clk_100mhz)begin
    if (btnu & ~old_btnu)begin
      quotient<= dividend/divisor; //divide
    end
    if (btnc & ~old_btnc)begin
      dividend <= sw; //divide //load dividend
    end
    if (btnl & ~old_btnl)begin
      divisor <= sw; //divide //load divisor
    end
  end
endmodule
```

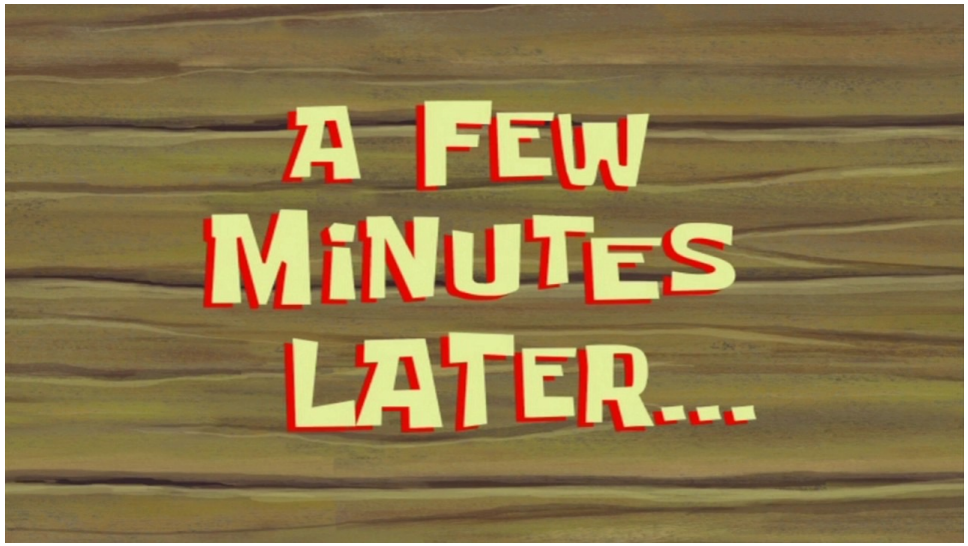

sw[15:0]
btnc
btnl
btnu
D
Q
dividend[15:0]
D
Q
divisor[15:0]
/
quotient[15:0]
D
Q
led[15:0]

Let's Build it.

- Terminal Output:

```
jodalyst@Josephs-MBP lec08 % python3 lab-bc.py -d
Output target will be obj/out.bit
Building your code ... (this may take a while, be patient)
Build succeeded, output at obj/out.bit
Log file available at obj/build.log
Diagnostics available in obj
jodalyst@Josephs-MBP lec08 % openFPGALoader -b arty_a7_100t
obj/out.bit
Jtag frequency : requested 10.00MHz -> real 10.00MHz
Open file DONE
Parse file DONE
load program
Flash SRAM: [==================================================]
100.00%
Done
```

"Hmmm Looks good."



“Jeeze when I deploy this in a high-throughput system where I have a new pair of numbers to divide every 10ns, the division results are trash. What’s going on?”…

You look through the output from the build…

Starting at line 1100:

```
Phase 9 Depositing Routes
Phase 9 Depositing Routes | Checksum: 1ca950f40

Time (s): cpu = 00:00:30 ; elapsed = 00:00:23 . Memory (MB): peak = 3032.945 ; gain = 0.000 ; free
physical = 1737 ; free virtual = 7502

Phase 10 Post Router Timing
INFO: [Route 35-57] Estimated Timing Summary | WNS=-22.720| TNS=-140.635| WHS=0.344  | THS=0.000  |

WARNING: [Route 35-328] Router estimated timing not met.
Resolution: For a complete and accurate timing signoff, report_timing_summary must be run after
route_design. Alternatively, route_design can be run with the -timing_summary option to enable a
complete timing signoff at the end of route_design.
Phase 10 Post Router Timing | Checksum: 1ca950f40

Time (s): cpu = 00:00:30 ; elapsed = 00:00:23 . Memory (MB): peak = 3032.945 ; gain = 0.000 ; free
physical = 1737 ; free virtual = 7503
INFO: [Route 35-16] Router Completed Successfully

Time (s): cpu = 00:00:30 ; elapsed = 00:00:23 . Memory (MB): peak = 3032.945 ; gain = 0.000 ; free
physical = 1780 ; free virtual = 7546

Routing Is Done.
INFO: [Common 17-83] Releasing license: Implementation
16 Infos, 1 Warnings, 0 Critical Warnings and 0 Errors encountered.
route_design completed successfully
```

Look at

routerpt_report_timing.rpt

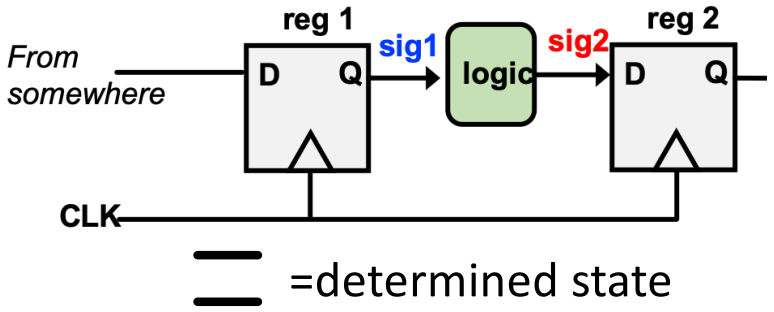
```
Timing Report

Slack (VIOLATED) :         -22.722ns  (required time - arrival time)
  Source:                 divisor_reg[0]/C
                            (rising edge-triggered cell FDRE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            quotient_reg[0]/D
                            (rising edge-triggered cell FDRE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             sys_clk_pin
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (sys_clk_pin rise@10.000ns - sys_clk_pin rise@0.000ns)
  Data Path Delay:        32.595ns  (logic 21.578ns (66.200%)  route 11.017ns (33.800%))
  Logic Levels:           86  (CARRY4=74 LUT2=2 LUT3=10)
  Clock Path Skew:        -0.138ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    5.010ns = ( 15.010 - 10.000 )
    Source Clock Delay      (SCD):    5.328ns
    Clock Pessimism Removal (CPR):    0.180ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

What is Slack?

- Slack: measure of how safe your timing is
- The two big timing constraints we worry about are related to setup and hold
- Therefore there are two Slack values:
  - Setup slack: $t_{required} - t_{actual}$
  - Hold slack: $t_{actual} - t_{required}$

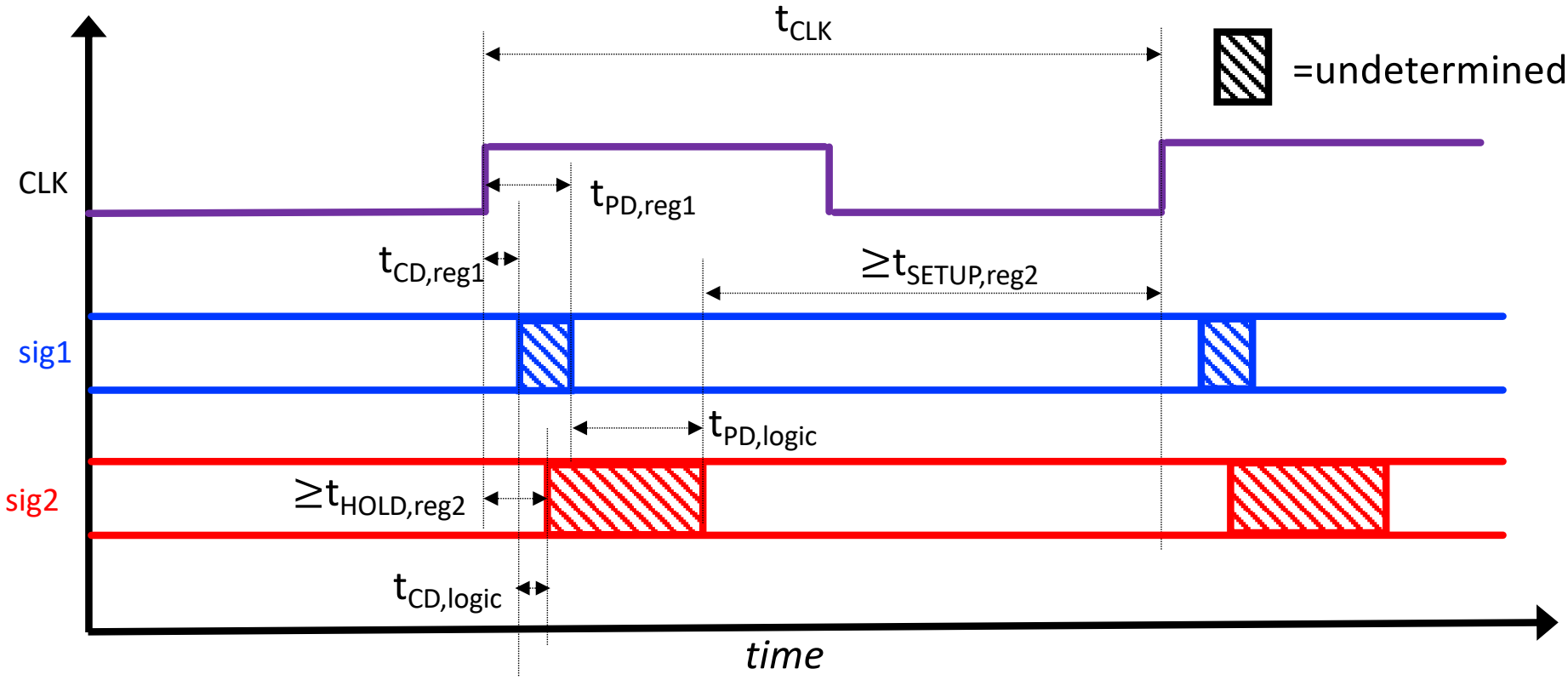
These are defined such that Positive is GOOD, Negative is BAD for both

Timing Diagram



=determined state

=undetermined state

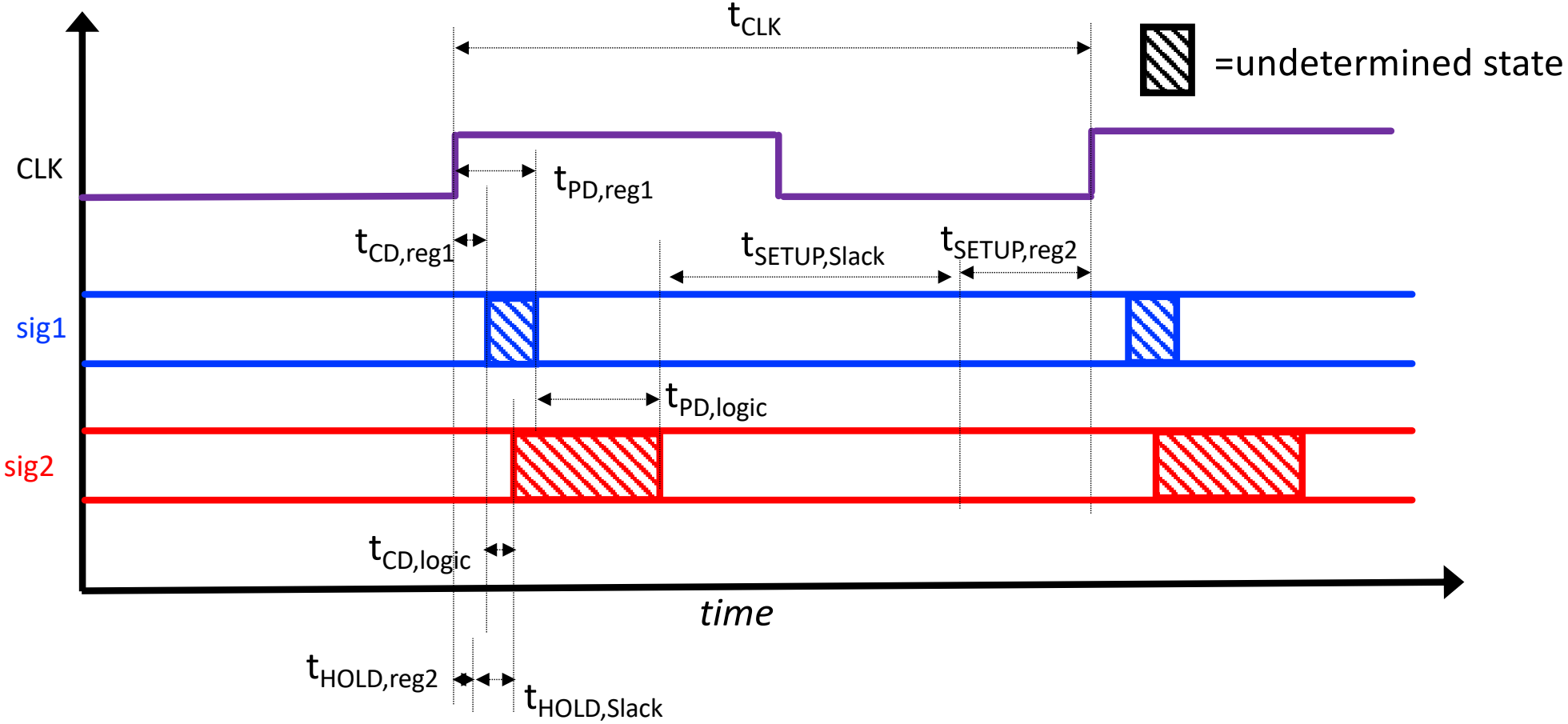


Two Requirements/ Conclusions:

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

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

Add in Slack



$$t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} + t_{SETUP,Slack} = t_{CLK}$$

$$t_{CD,reg1} + t_{CD,logic} = t_{HOLD,reg2} + t_{HOLD,Slack}$$

Equations* $t_{SETUP,Slack} = t_{CLK} - (t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2})$

$t_{HOLD,Slack} = t_{CD,reg1} + t_{CD,logic} - t_{HOLD,reg2}$

**not inequalities*

Conclusion

- Positive Slack is **GOOD**
- Negative Slack is **BAD**

Look at

This is not good Negative Slack

routerpt_report_timing.rpt

```
Timing Report

Slack (VIOLATED) :        -22.722ns  (required time - arrival time)
  Source:                 divisor_reg[0]/C
                            (rising edge-triggered cell FDRE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            quotient_reg[0]/D
                            (rising edge-triggered cell FDRE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             sys_clk_pin
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (sys_clk_pin rise@10.000ns - sys_clk_pin rise@0.000ns)
  Data Path Delay:        32.595ns  (logic 21.578ns (66.200%)  route 11.017ns (33.800%))
  Logic Levels:           86  (CARRY4=74 LUT2=2 LUT3=10)
  Clock Path Skew:        -0.138ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    5.010ns = ( 15.010 - 10.000 )
    Source Clock Delay      (SCD):    5.328ns
    Clock Pessimism Removal (CPR):    0.180ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

2/3 from routing

1/3 from routing

Results

- By default Vivado only gives you a few offending paths (our default is one) and it provides them in order of worst to best
- You can ask for more paths using different arguments;

https://docs.xilinx.com/r/2020.2-English/ug835-vivado-tcl-commands/report_timing

Up Next

- We'll start to work with these reports in the next lecture.

Past Project #3

Sudoku Solver