

Timing and Clocking

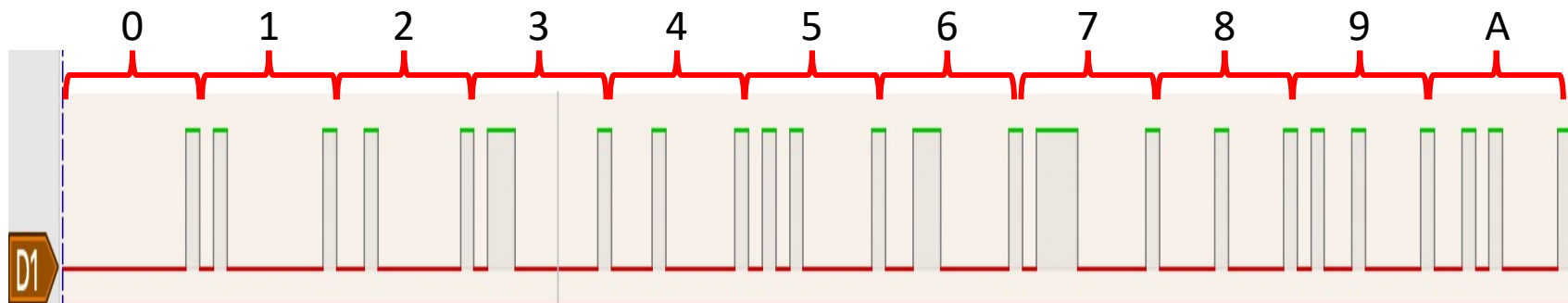
6.205

Planning Stuff

- Week 03 Due Tomorrow
- Week 04 Released Thursday (video)

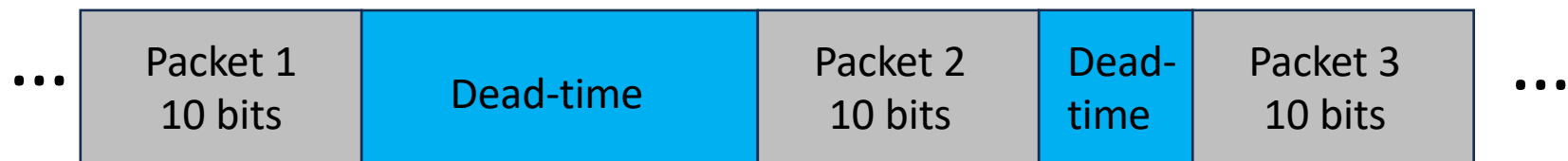
Notes on UART RX

- I went and from the computer decided to send down 0, then 1, then 2, then 3...to 255 over UART.
- This was the trace on the UART_RXD line



UART Packet

- In the UART standard there is no guarantee in regards to inter-byte spacing between bytes sent down.
- It can vary (and often does)

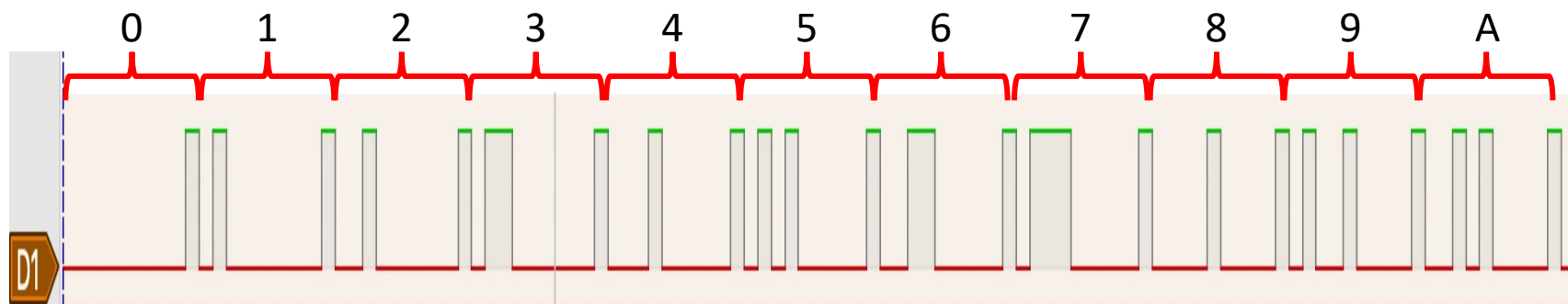


Robustness in Measuring UART

- General recommendation is to not just verify the start bit is low once and the stop bit is high once.
- Instead recommend to verify start bit is low up until $0.5 * \text{BAUD}$
- And recommend to verify stop bit is high from $0.5 * \text{BAUD}$ to $1 * \text{BAUD}$

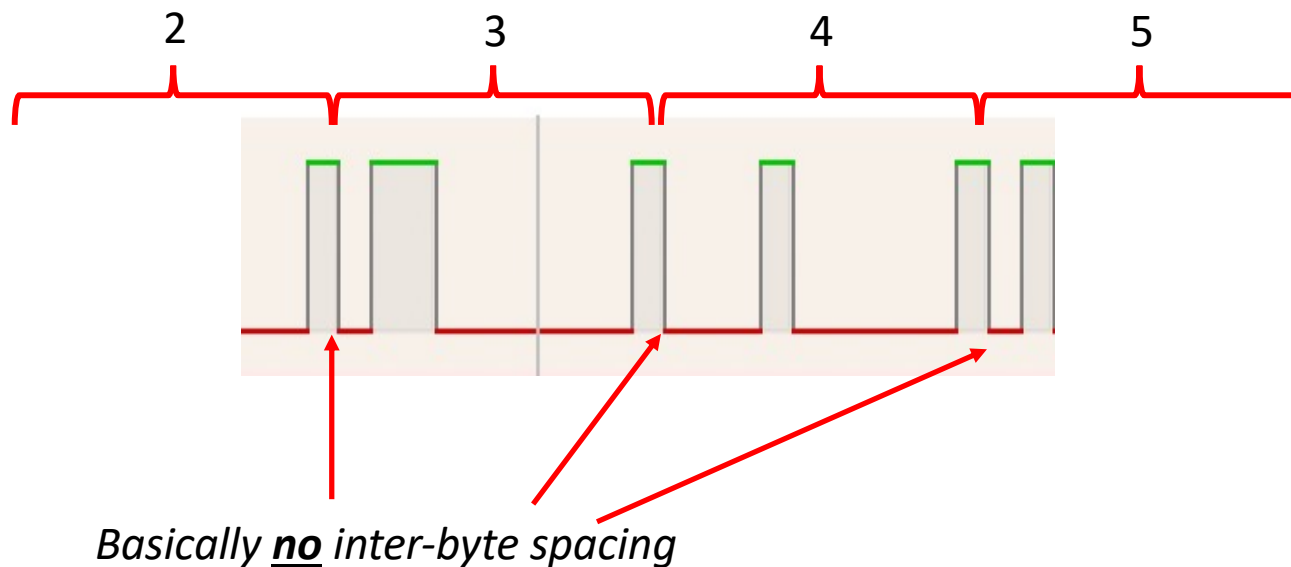
However...the FT2232 chip on our board that handles the UART...

- Packs the UART packets very tightly



However...the FT2232 chip on our board that handles the UART...

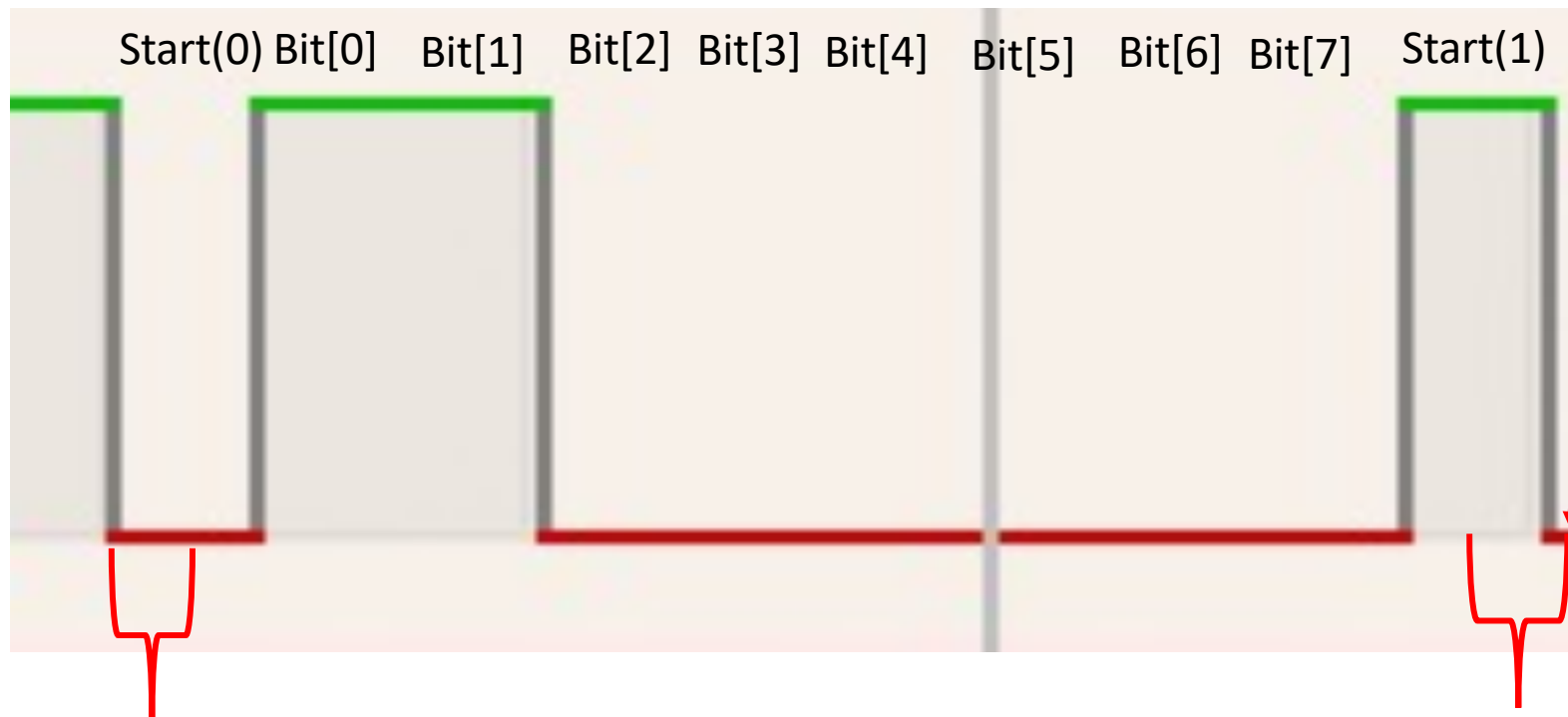
- Packs the UART packets very tightly



As a result...

Could fall off cliff into next start bit... :/

- Recommend to verify stop bit is high from $0.5 \times \text{BAUD}$ to $1 \times \text{BAUD}$ could cause issues.



Verify low for half BAUD

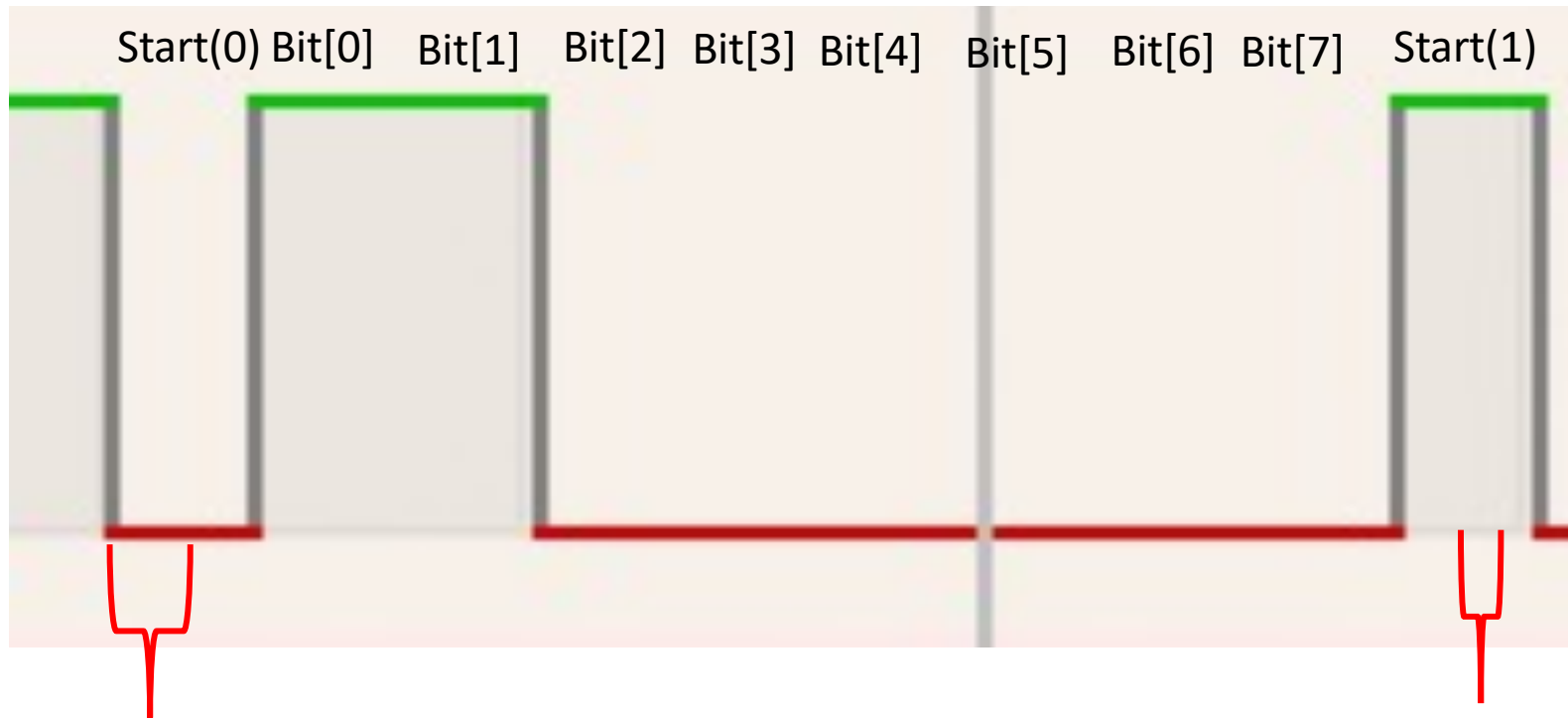
9/24/24

Verify high from 0.5BAUD to 1 BAUD

<https://fpga.mit.edu/6205/F24>

Solution...

- Verify value remains 1 from 0.5 BAUD to 0.75BAUD

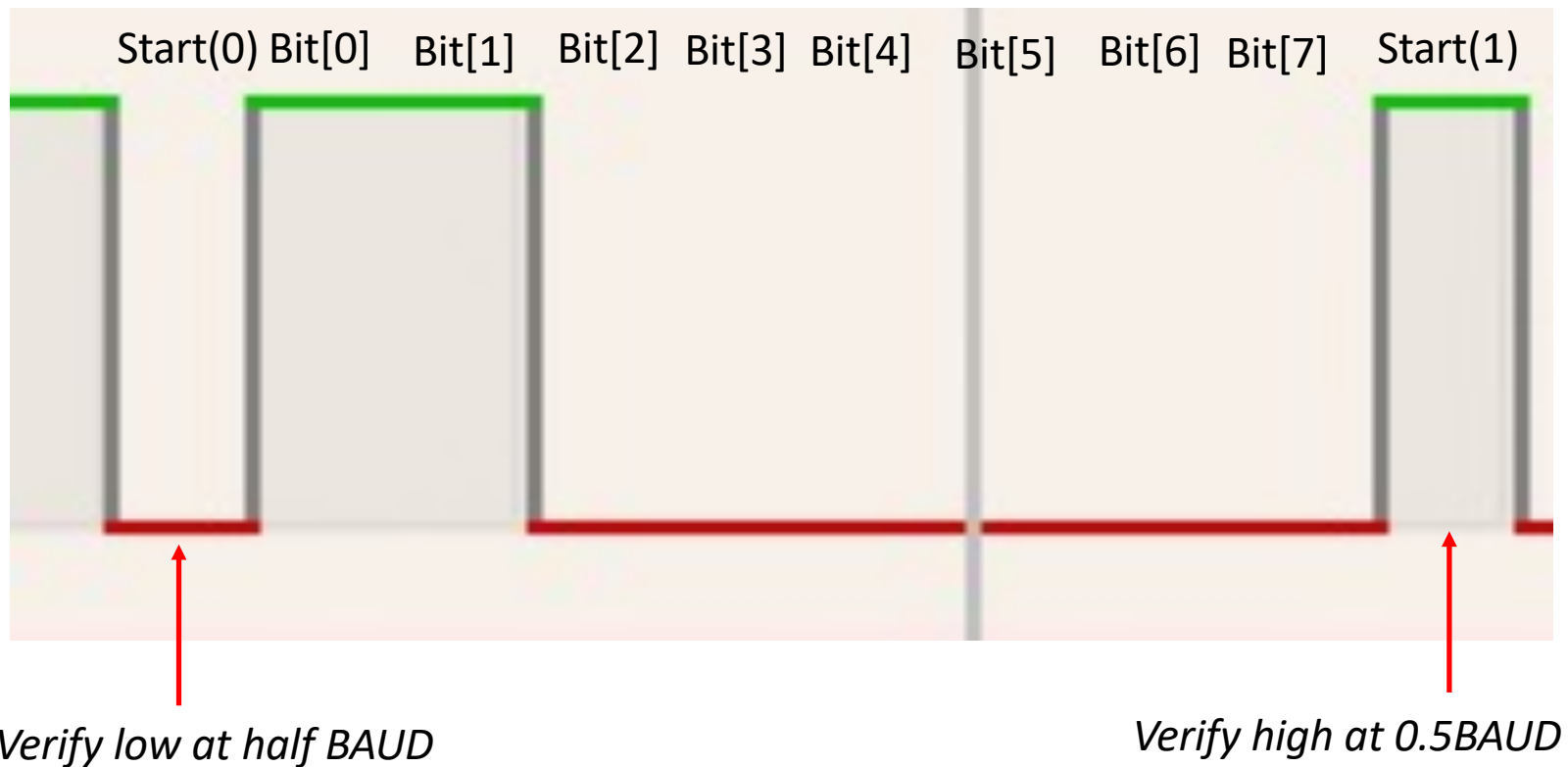


Verify low for half BAUD

Verify high from 0.5BAUD to 0.75BAUD

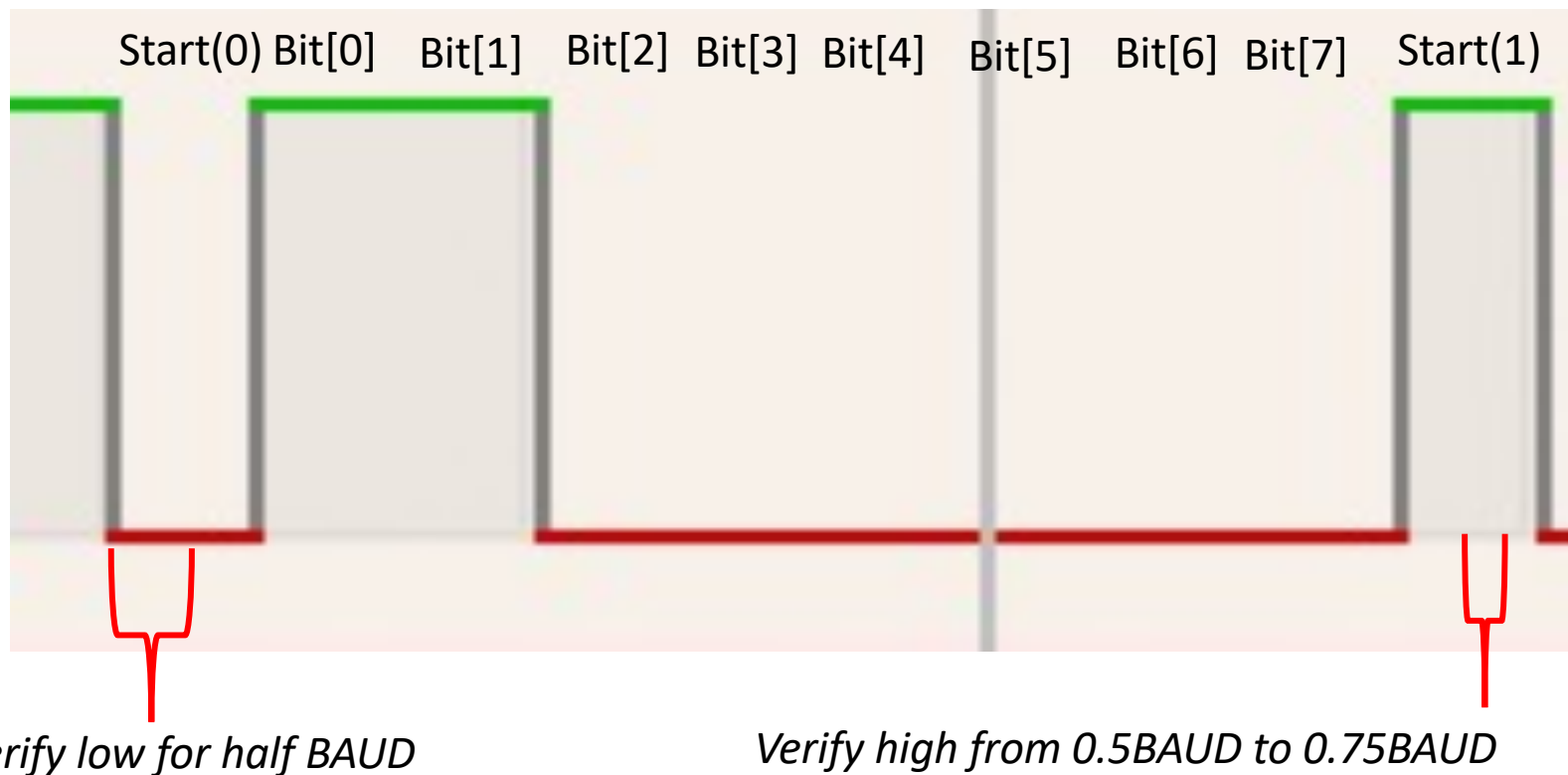
Quality Difference Demo I

- Just Verifying once at start and stop



Quality Difference Demo II

- Verifying continuously in two shown regions



FSMs in History

Car Alarm FSM

- Up until 2021 the “FSM” lab in 6.111 was making a car alarm
- FSM design was and still can be a very common approach to digital circuit design

Fall 2019

Home
Announcements

Labkit
News4 DDR


Handouts
» Lectures
» Labs: 1, 2, 3, 4, 5A, 5B

Final Projects
» Project info
» Project list
» Memorable projects
» Past projects – all
» Schedule

*MIT cert required
*On-line Grades
*Submit PDFs
*Submit Verilog
*Staffed Lab Hours

Course info
Course objectives
Course calendar

Tools
Pizza (new tab)



Description of Anti-Theft System

Since your client is completely focused on her start-up, she wants an anti-theft system that's highly automated. The system is armed automatically after she turns off the ignition and both the driver's door and passenger's doors are open, the system arms itself after all the doors have been closed and T_ARM_DELAY has passed; that delay is T_ARM_DELAY.

Once the system has been armed, opening the driver's door the system begins a countdown. If the ignition is not turned on within the countdown interval (T_DRIVER_DOOR_DELAY) (T_ALARM_ON) after the door closes, at which time the system resets to the armed but silent state. If the ignition is turned on within the countdown interval, the system disarms itself.

Always a paragon of politeness, your client opens the passenger door first if she's transporting a guest. When the passenger door is opened first, a separate, presumably longer, countdown interval (T_PASSSENGER_DOOR_DELAY) begins. If the driver's door is opened first, a separate, presumably longer, countdown interval (T_DRIVER_DOOR_DELAY) begins. If the passenger door is opened first, a separate, presumably longer, countdown interval (T_PASSSENGER_DOOR_DELAY) begins. If the driver's door is opened first, a separate, presumably longer, countdown interval (T_DRIVER_DOOR_DELAY) begins.

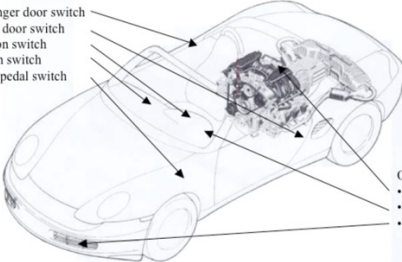
There is a status indicator LED on the dash. It blinks with a two-second period when the system is armed. It is constantly illuminated either the system is in the countdown interval or the system is armed.

So far this all is ordinary alarm functionality. But you're worried that a knowledgeable thief might disable the siren and then just drive off with the car. So you've added an actuator that disables the siren when the ignition is turned on and then the driver presses both a hidden switch and the brake pedal simultaneously. Power is then latched on until the ignition is turned off.

The diagram below lists all the sensors (inputs) and actuators (outputs) connected to the system.

Inputs:

- passenger door switch
- driver door switch
- ignition switch
- hidden switch
- brake pedal switch



Outputs:

- fuel pump power
- status indicator
- siren

Figure 1: System diagram showing sensors (inputs) and actuators (outputs)

The system timings are based on four parameters (in seconds): the delay between exiting the car and the arming of the alarm (T_ARM_DELAY), the length of the countdown interval (T_PASSSENGER_DOOR_DELAY), and the length of time the siren sounds (T_ALARM_ON). The default value for each parameter is listed in the table below, but each may be changed. Time_Parameter_Selector switches specify the parameter number of the parameter to be changed. Time_Value switches are a 4-bit value representing the value to be programmed. Note that your system should behave correctly even if one or more of the parameters is set to 0.

Default Timing Parameters				
Interval Name	Symbol	Parameter Number	Default Time (sec)	Time Value
Arming delay	T_ARM_DELAY	00	6	0110
Countdown, driver's door	T_DRIVER_DOOR_DELAY	01	8	1000
Countdown, passenger door	T_PASSSENGER_DOOR_DELAY	10	15	1111
Siren ON time	T_ALARM_ON	11	10	1010

Block Descriptions/Implementation

The following diagram illustrates a possible organization of your design into modules.

Car Alarm FSM

- When Gim graduated from MIT he got a job with DEC (Digital Equipment Corporation) that made the PDP-1 among other computers and then TI
- Got big signing bonus and bought a nice convertible
- Parked Convertible went into apartment.
- Convertible was not there came out



Gim Hom

Took 6.111 in 1969..graduated in 1970

6.111 Instructor 2013-2021

Now retired

Car Theft FSMs

- 2016, MA: ~7 million people, 6,600 car thefts for year
- 1975, MA: 5.8 million people, 91,000 car thefts for year (peak)

~5% of cars were stolen per year in MA

#1 state for car theft for 1965-1987

Massachusetts Population and Rate of Crime Rank Compared to other States

State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	1960	8	35	36	33	41	38	32	36	34	38	11
Massachusetts	1961	8	29	35	28	42	35	31	34	29	32	7
Massachusetts	1962	9	26	33	24	40	37	25	31	26	31	7
Massachusetts	1963	9	28	33	27	40	39	27	33	27	33	4
Massachusetts	1964	9	26	33	25	37	37	24	38	25	34	2
Massachusetts	1965	10	23	33	22	36	39	21	36	23	35	1
Massachusetts	1966	10	24	32	23	38	41	19	36	21	37	1
Massachusetts	1967	10	25	34	25	38	37	23	37	25	38	1
Massachusetts	1968	10	21	31	20	35	35	21	35	19	35	1
Massachusetts	1969	10	19	29	18	35	38	19	35	15	34	1

State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	1970	10	20	31	18	38	33	19	35	14	35	1
Massachusetts	1971	10	16	27	16	36	38	13	33	13	32	1
Massachusetts	1972	10	17	26	16	38	39	11	29	13	35	1
Massachusetts	1973	10	15	20	14	36	34	10	29	12	33	1
Massachusetts	1974	10	12	20	11	37	37	10	29	12	32	1
Massachusetts	1975	10	11	16	12	39	29	9	26	13	34	1
Massachusetts	1976	10	13	18	12	40	37	11	22	11	36	1
Massachusetts	1977	10	15	17	16	43	31	12	19	13	37	1
Massachusetts	1978	10	16	16	17	40	31	13	19	12	38	1
Massachusetts	1979	10	16	15	15	41	32	12	16	11	36	1

State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	1980	11	16	13	17	39	31	9	13	13	39	1
Massachusetts	1981	11	20	12	21	40	30	8	15	17	40	1
Massachusetts	1982	11	18	14	21	37	31	10	13	19	40	1
Massachusetts	1983	11	20	12	22	40	28	9	10	20	41	1
Massachusetts	1984	12	25	13	24	36	25	12	12	25	41	2
Massachusetts	1985	12	23	16	25	39	23	12	14	24	40	1
Massachusetts	1986	12	26	18	27	38	26	14	17	28	43	1
Massachusetts	1987	13	26	14	28	42	26	15	14	31	45	1
Massachusetts	1988	13	23	13	25	38	25	14	11	29	41	3

State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	2000	13	42	21	44	41	36	27	14	41	49	16
Massachusetts	2001	13	41	20	43	42	29	25	15	40	47	18
Massachusetts	2002	13	39	18	42	38	34	24	17	39	46	18
Massachusetts	2003	13	28	17	40	42	30	19	18	36	46	18
Massachusetts	2004	13	39	18	42	37	33	20	18	38	47	22
Massachusetts	2005	13	42	19	45	37	36	20	18	35	48	30
Massachusetts	2006	13	41	20	44	35	37	21	19	34	45	33
Massachusetts	2007	14	42	22	42	37	38	23	18	33	45	36
Massachusetts	2008	14	37	20	44	41	39	24	14	32	45	35
Massachusetts	2009	15	37	17	43	38	38	23	14	33	45	33

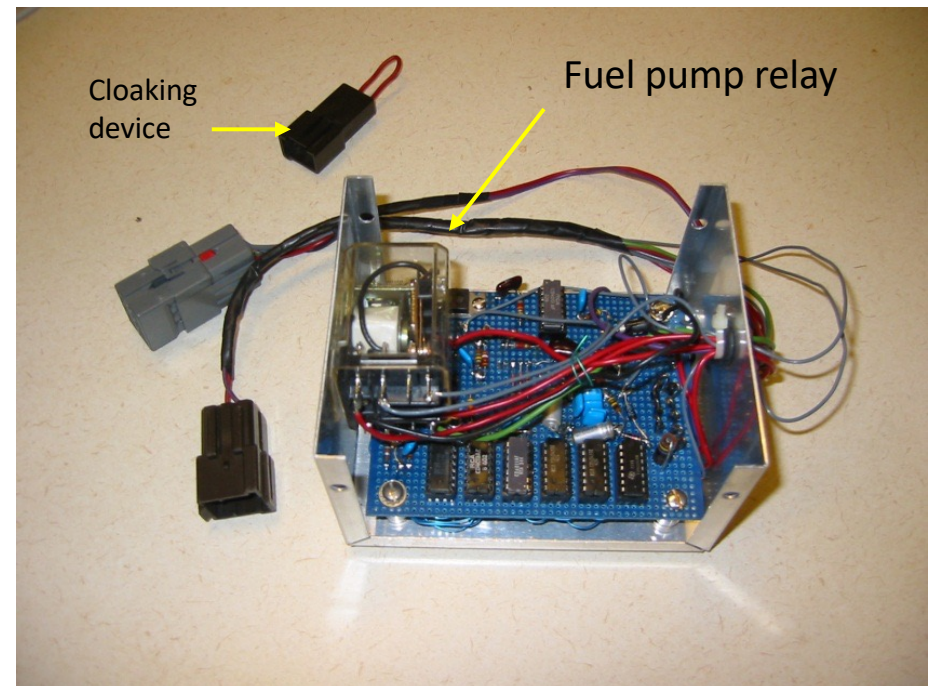
State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	2010	14	32	13	38	31	35	18	11	29	41	34
Massachusetts	2011	14	36	14	42	37	37	20	12	32	43	34
Massachusetts	2012	14	40	19	44	46	39	20	14	33	48	39
Massachusetts	2013	14	36	16	45	43	16	18	15	35	46	39
Massachusetts	2014	14	39	18	46	43	35	20	14	38	47	40
Massachusetts	2015	15	42	18	47	45	38	27	14	42	48	42
Massachusetts	2016	15	45	23	47	47	42	26	20	45	49	44

Car Alarm FSM

- Gim built a car alarm for his car.
- Designed it using an FSM-based approach.
- Had ~11 states
- Built it just like we talked about last week (bubble-diagram...developed logic, implemented...)



*Gim's FSM-based car alarm for his car
Built using 4000-series CMOS chips
(top of the line at the time)*



Magnavox Odyssey (1972)

- First commercially available game system
- Completely FSM-based

1TL200 BLAK & BK12 ODYSSEY GAME SIMULATOR

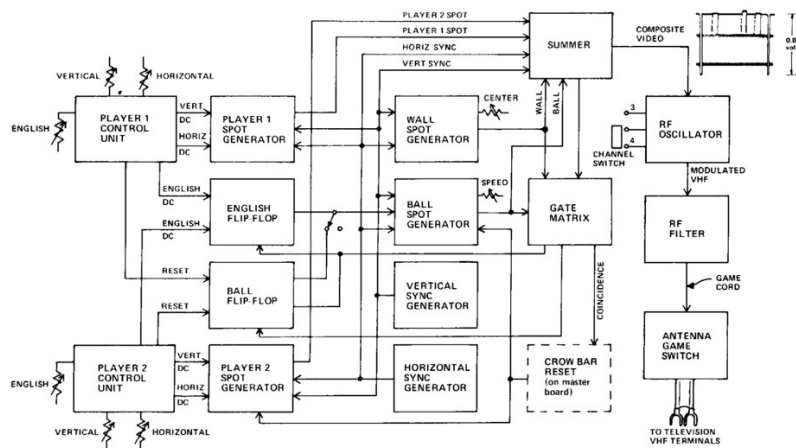
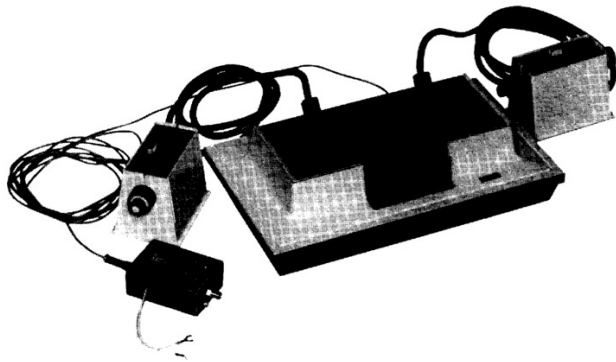


Figure 1 -- Odyssey Block Diagram

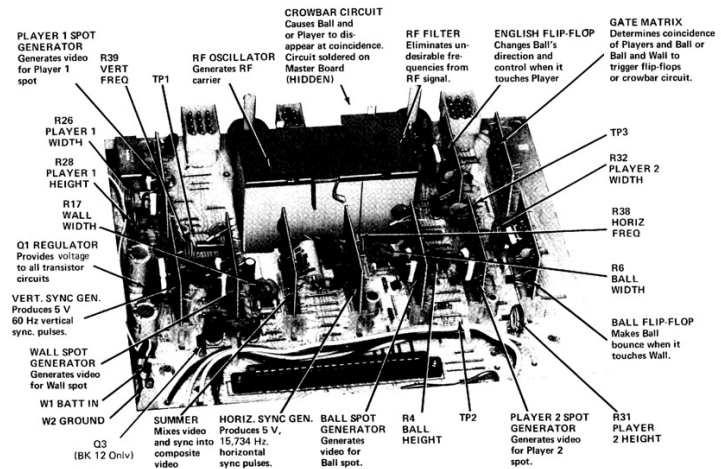


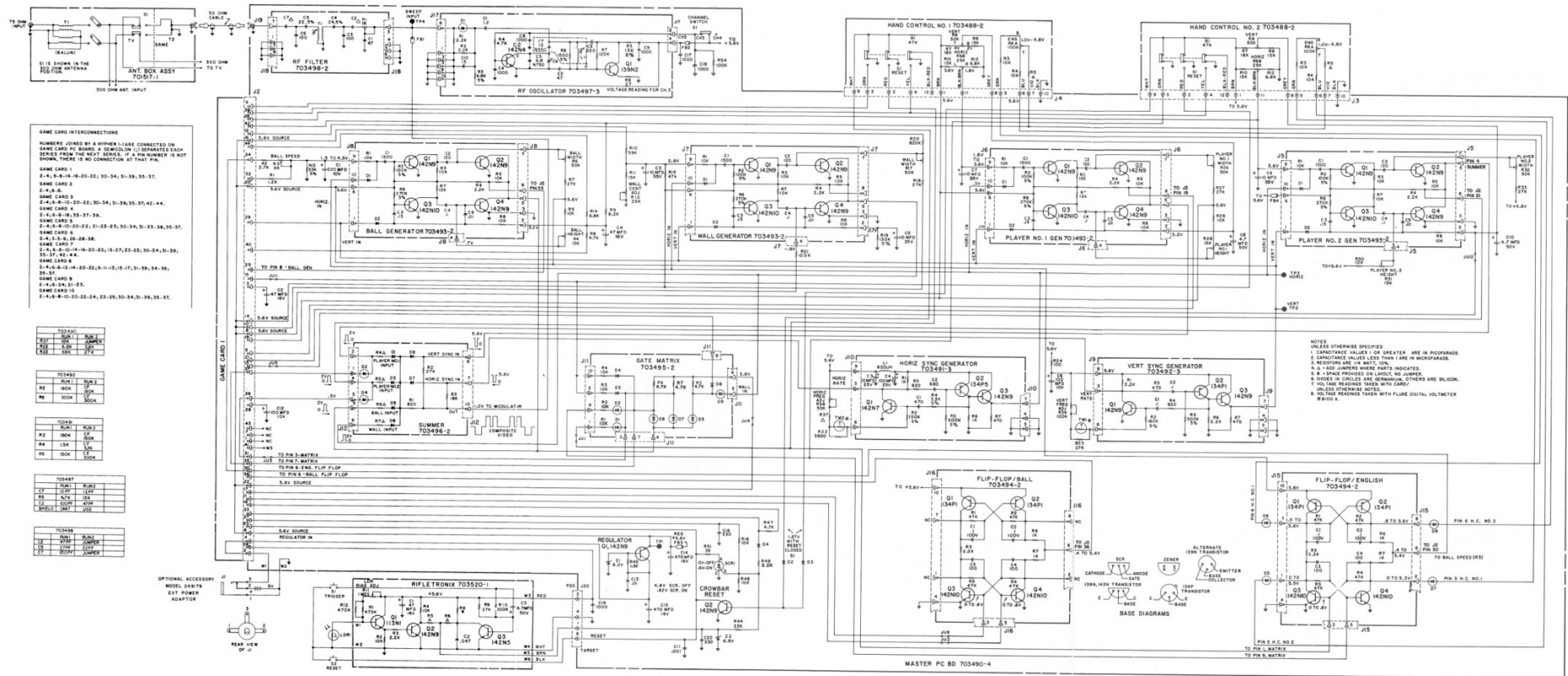
Figure 3 -- Master Board Module Location

• Implemented completely with discrete transistors:

MAGNAVOX ODYSSEY 1TL200BLAK SCHEMATIC DIAGRAM

Scanned from original service manual by David WINTER

<http://www.pong-story.com>



<https://www.pong-story.com/odyssey.htm>

<https://fpga.mit.edu/6205/F24>

Was just a large finite state machine

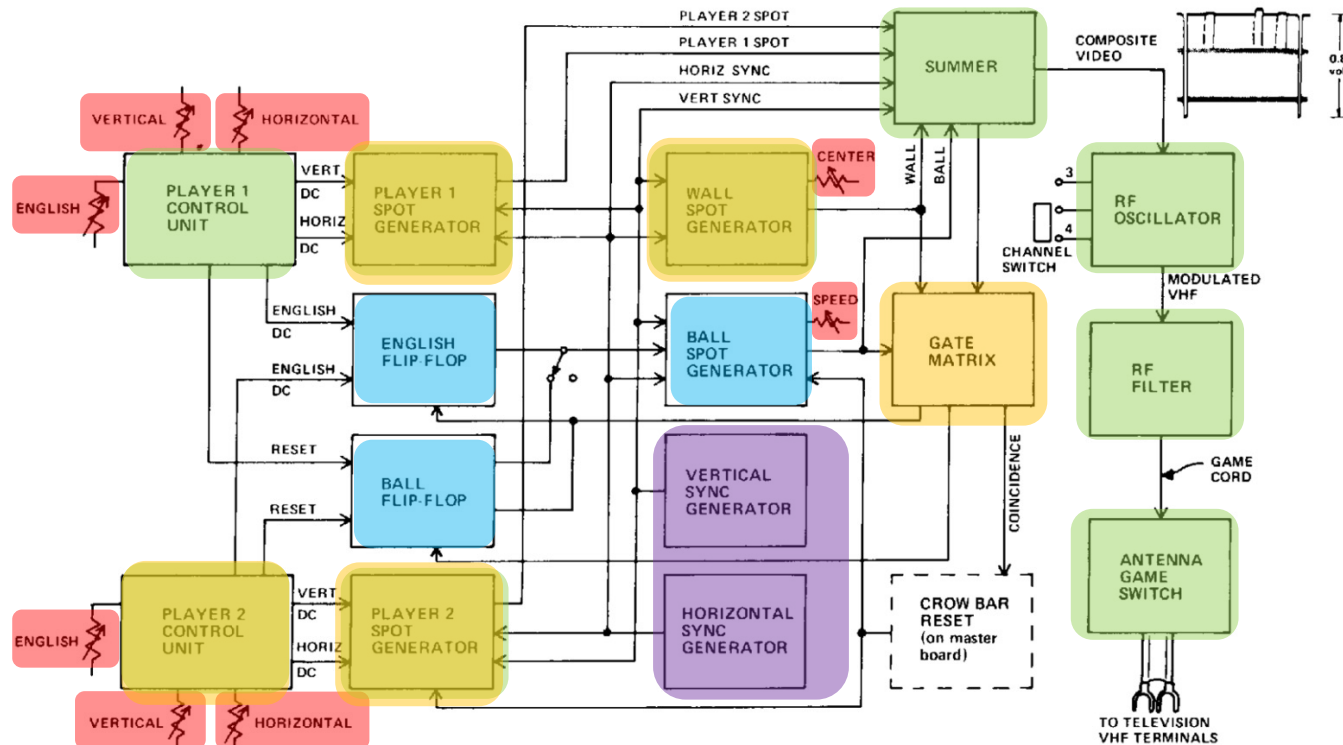


Figure 1 -- Odyssey Block Diagram

inputs

state

state-transition logic

output logic

“clock”

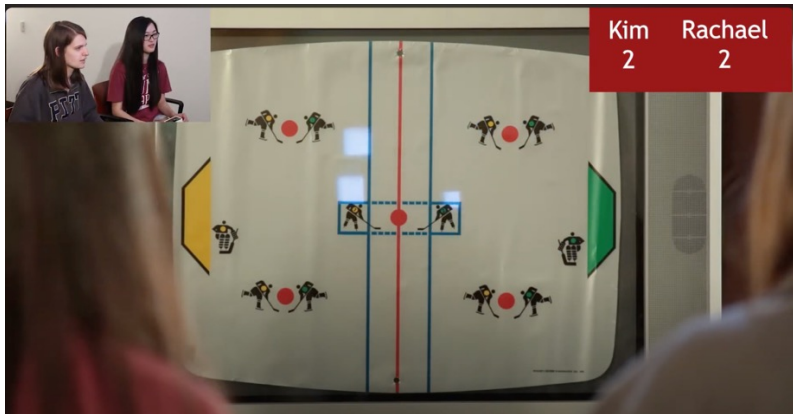
Magnavox Odyssey Game System



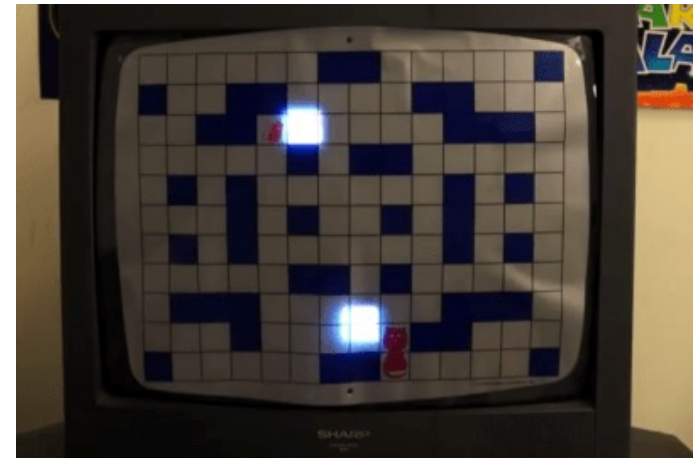
<http://odysim.blogspot.com/2020/>

Magnavox Odyssey Game System

Hockey



Cat and Mouse



Basketball



<https://youtube.com/playlist?list=PLtApm-Ri5WTIAEV1ClufPrca2MTj4uSvT&feature=shared>

https://www.youtube.com/watch?app=desktop&v=NsluZfTMRno&ab_channel=OdysseyNow

9/24/24

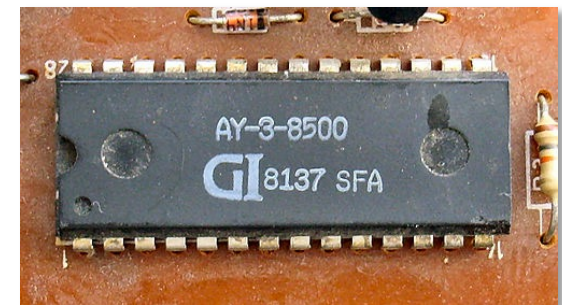
<https://fpga.mit.edu/6205/F24>

Early 1970s

- Most arcade systems were just FSMs implemented in discrete logic, including:
 - Pong
 - Breakout
- Space Invaders was first arcade machine to move some game logic to a Intel 8080 microprocessor.

Evolution of FSM-based Games

- As 1970s rolled on, entire game systems would get put on single chips
- “Ball-and-Paddle” Chips would be sold by companies and then other companies would buy them and put their own “skin” on them and sell them as their own. Many times it was the same game underneath
- Atari 2600 was first microprocessor-based home video game system



AY-3-8500 “Ball-and-Paddle” chip

• <http://www.pong-story.com/gi.htm>

<https://commons.wikimedia.org/wiki/File:AY-3-8500.jpg>

TV Fun



- Runs off a AY-3-8500
- Made by APF who started out importing Japanese 8-track players
- Company went bankrupt in the great video game crash of 1983.
- Have one set up in “lounge” area of lab in case anybody wants to play. If you need help setting it up, let me know.

Tiger Electronics Games



Tiger Electronics Games

- Tiger Electronics had 100's of versions of these in the 1980s and 1990s
- Almost all of them were based on three or four common **finite state machine** game chips
- They'd slap a different LCD skin and game art onto the same chip and resell



Modern Games

- Modern games are far too complex to be implemented with an FSM in any productive way (though it is still generally possible)
- However well-characterized chunks of game software is still used and re-used/skinned (for example game engines)
- But also stuff gets reskinned all the time

Candy Crush Saga



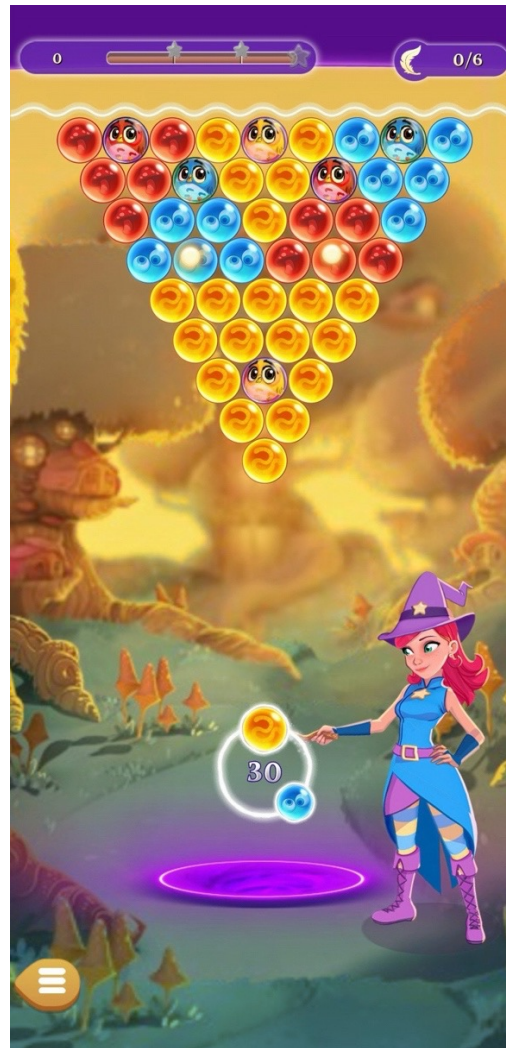
Pet Rescue Saga



Soda Saga



Bubble Witch 3 Saga



Farm Heroes Saga

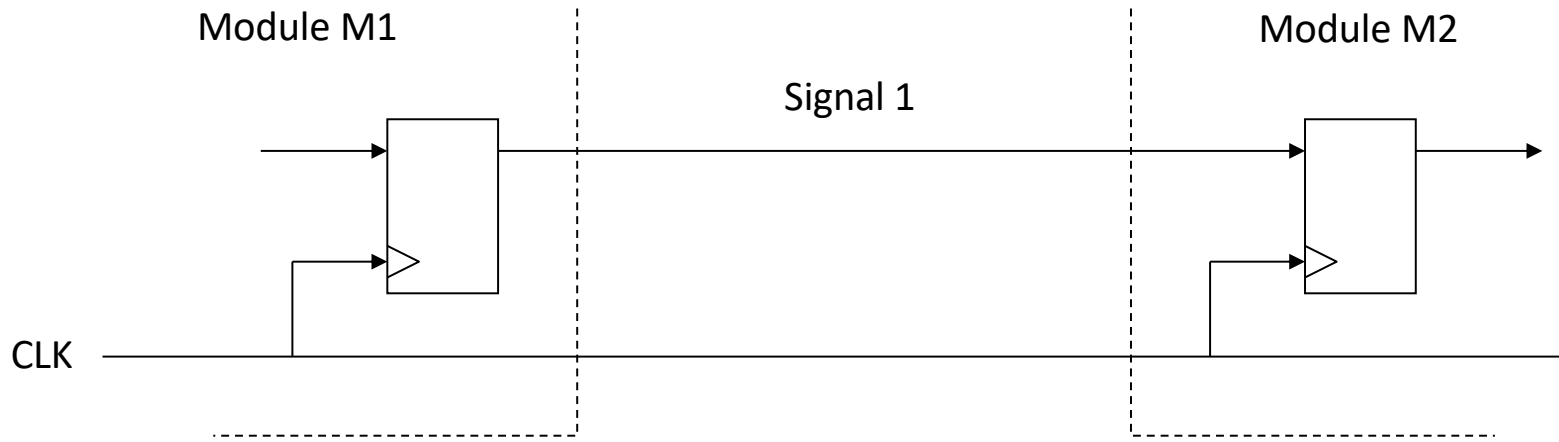


Finite State Machines Relevant

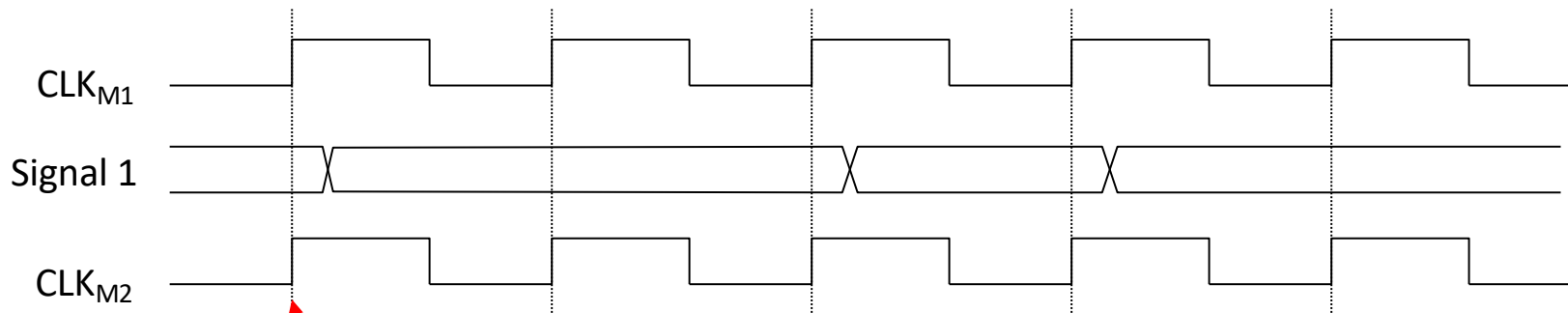
- Designing systems as finite state machines is still very common in digital design.
- Doing it in a structured way can make your HDL very transparent so you know what you're getting!
- You'll see data sheets and other places with FSM diagrams and many protocols express their functionality with FSMs.

Clocks and Time

Clocking and Synchronous Communication



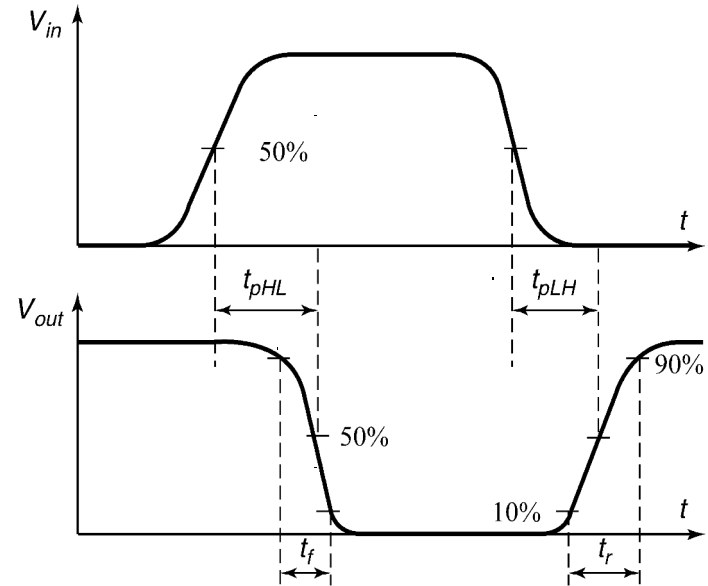
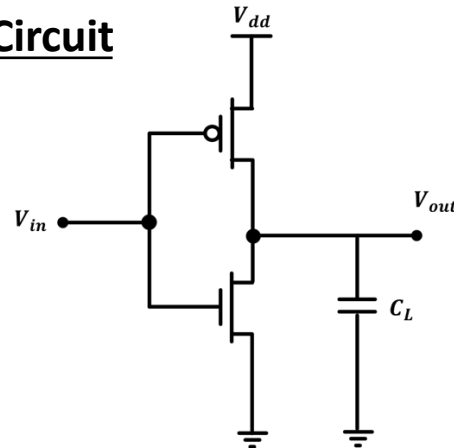
Ideal world:



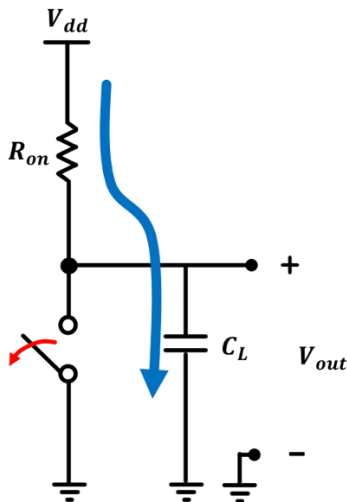
M1 and M2 clock edges aligned in time

Delay Estimation: Simple RC Networks

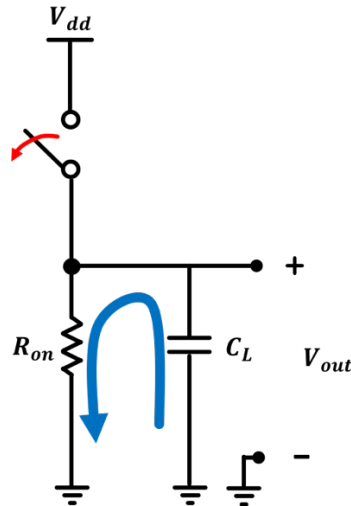
Simple CMOS Circuit



Low-to-High



High-to-Low

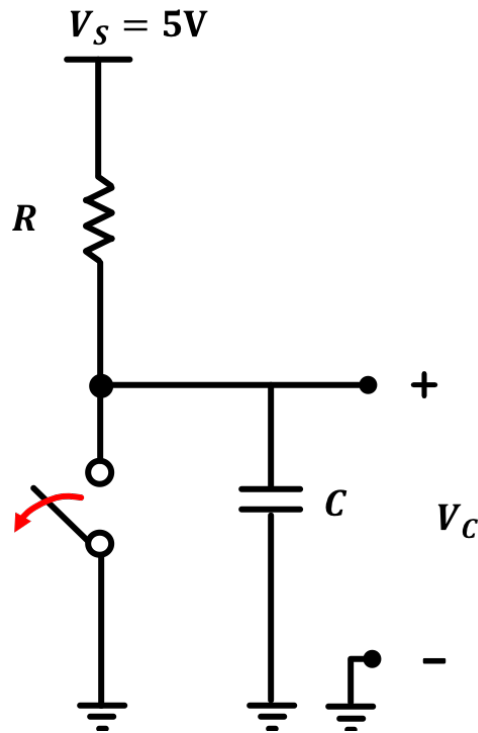


review

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

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

RC Equation



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

$$V_s = 5 \text{ V}$$

Switch is closed $t < 0$

Switch opens $t > 0$

$$V_s = V_R + V_C$$

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

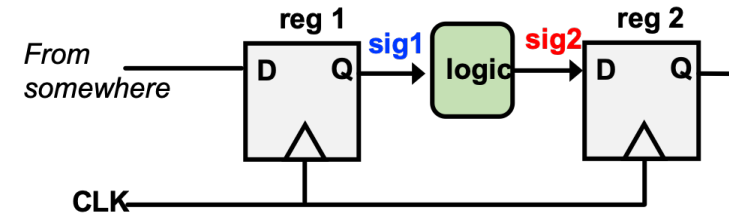
$$V_s = RC \frac{dV_c}{dt} + V_c$$


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

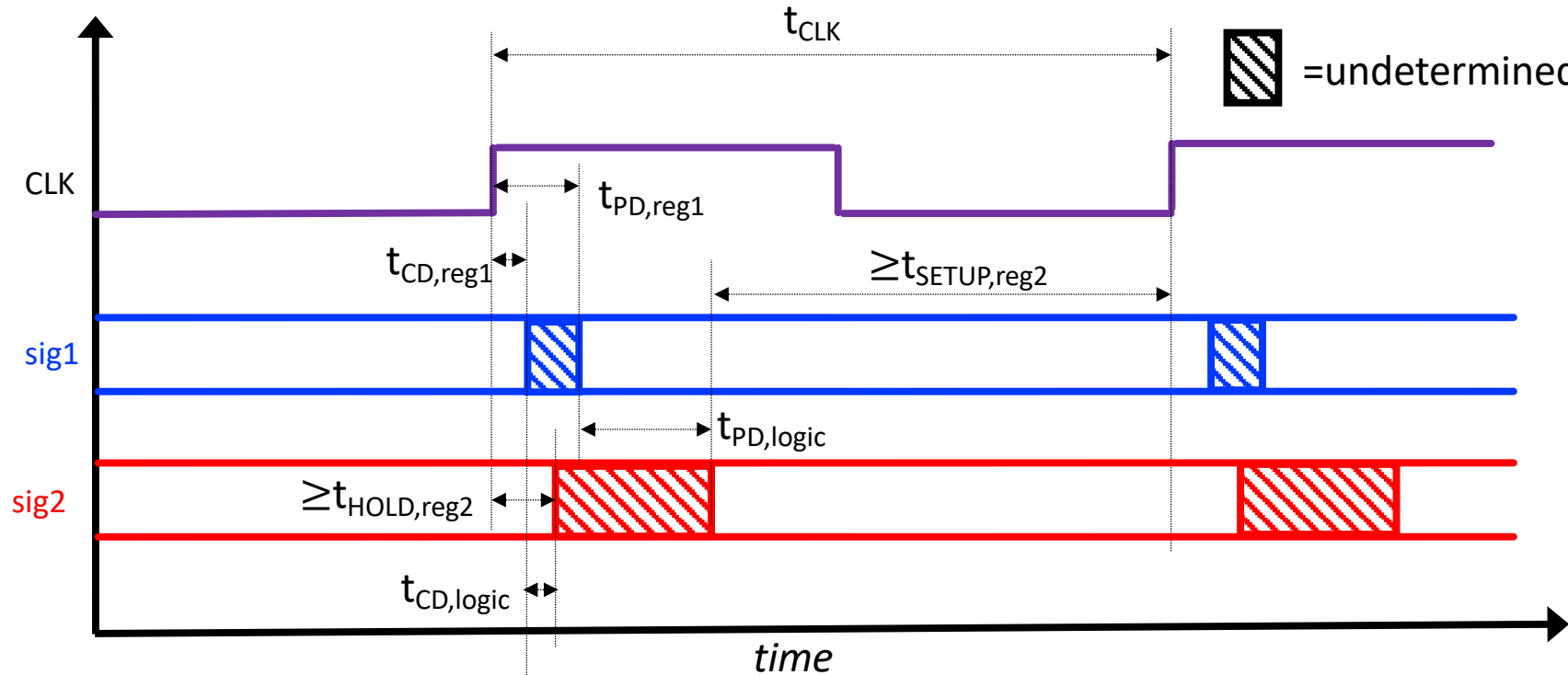
So Signals Experience Delays

- "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

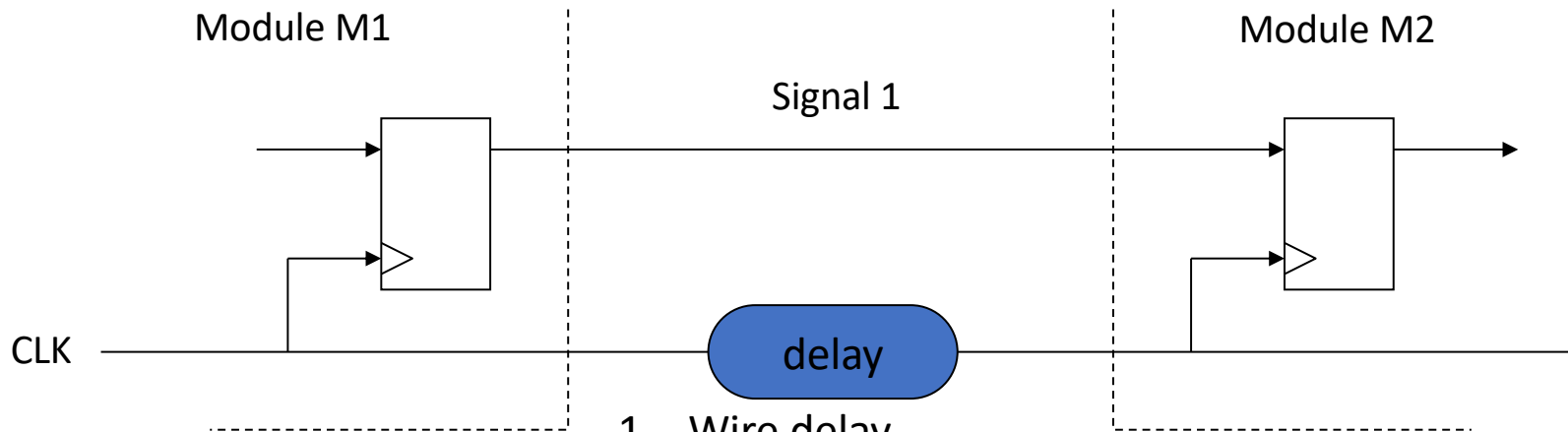


**Two Requirements/
Conclusions:**

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

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

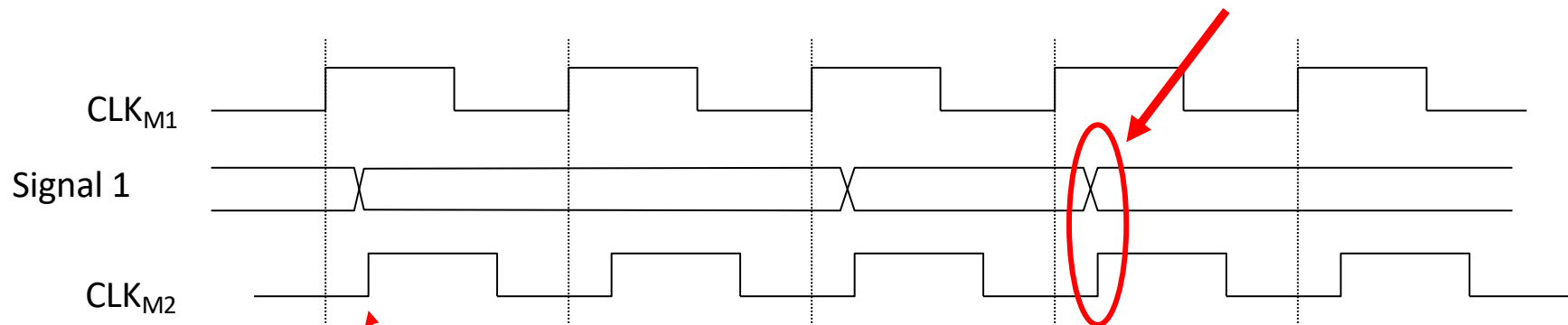
Clock Skew



1. Wire delay
2. Different clocks!

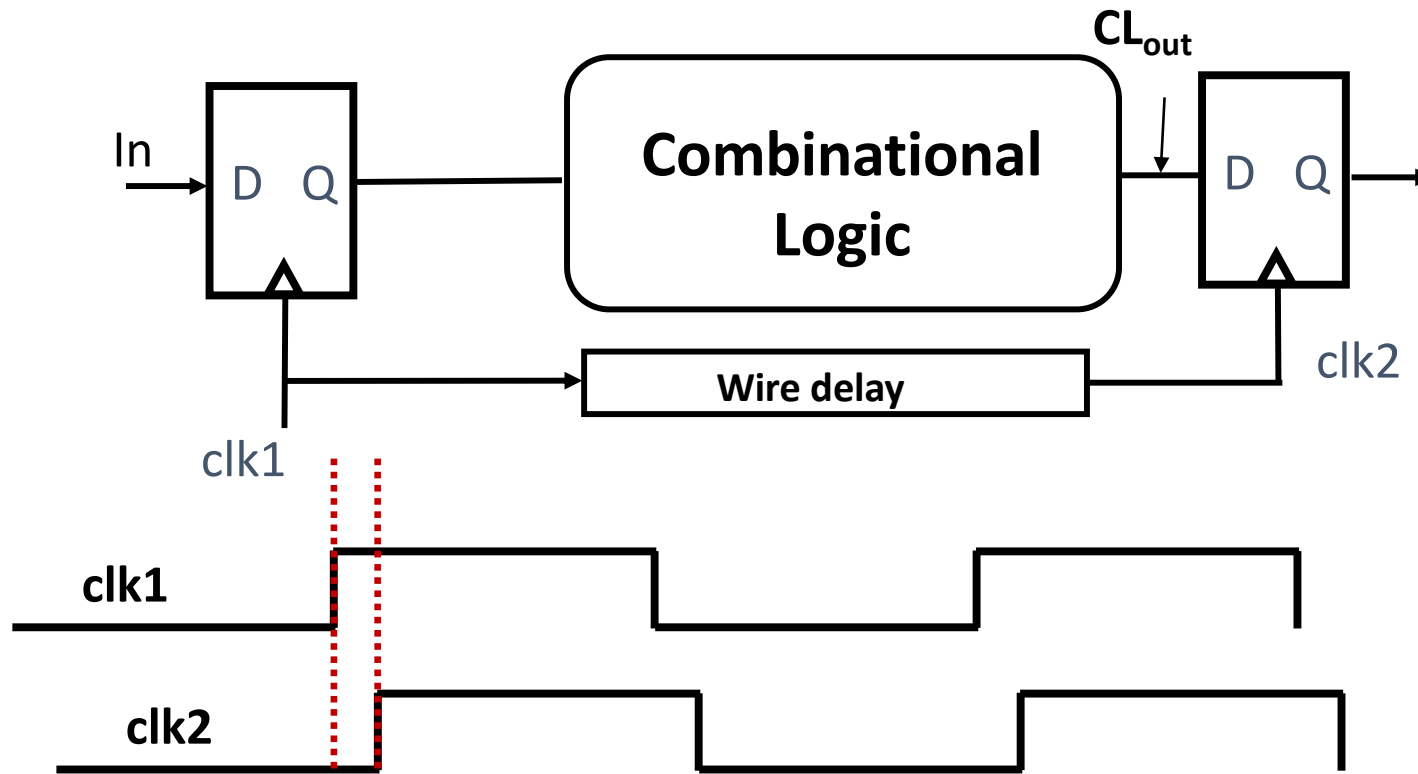
Oops! Skew has caused a hold time problem!

Real world has clock skew:



M2 clock delayed with respect to M1 clock

Clocks are Not Perfect: Clock Skew

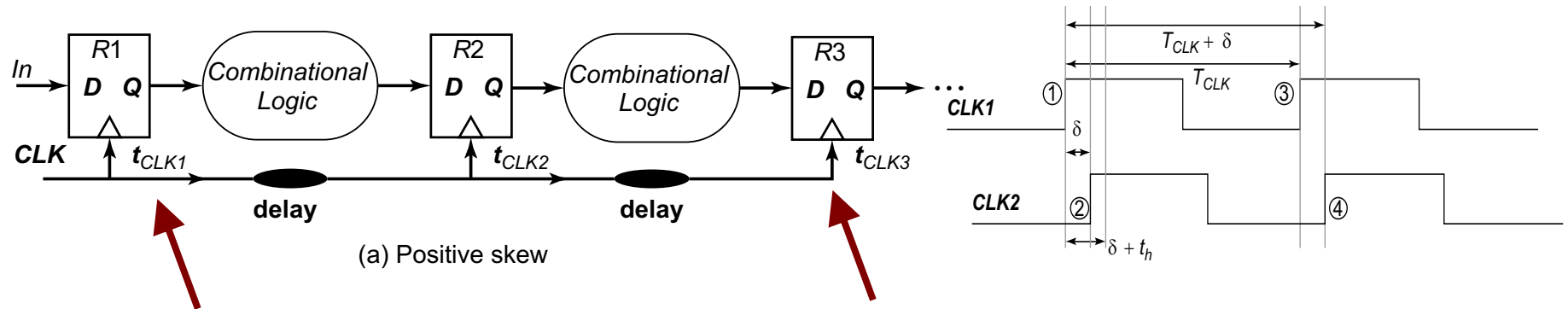


$\delta > 0$

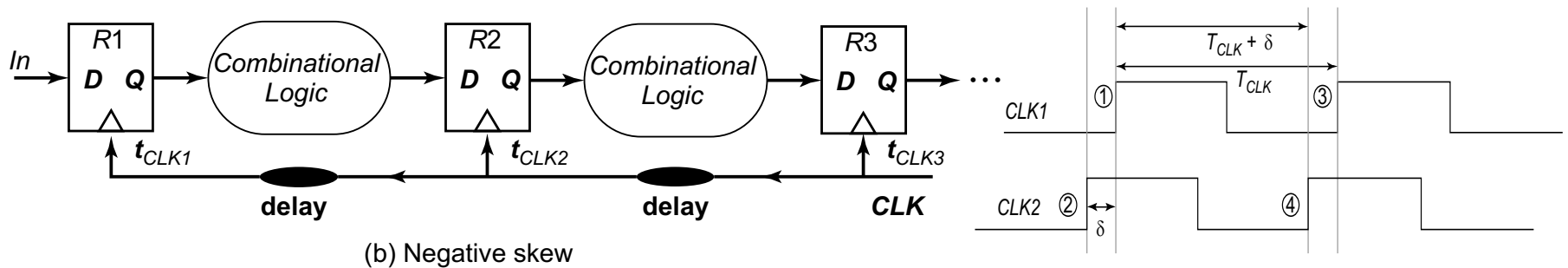
$$t_{\text{skew}} = t_{\text{clk2}} - t_{\text{clk1}}$$

Based off of times of rising edges.
Not periods!

Positive and Negative Skew



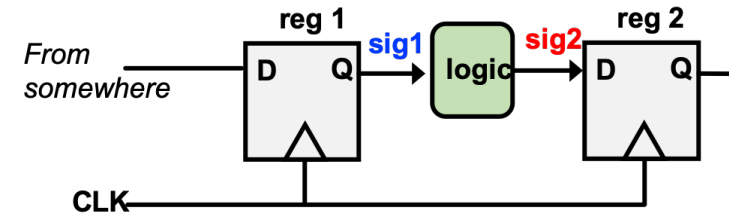
Launching edge arrives before the receiving edge (positive skew)



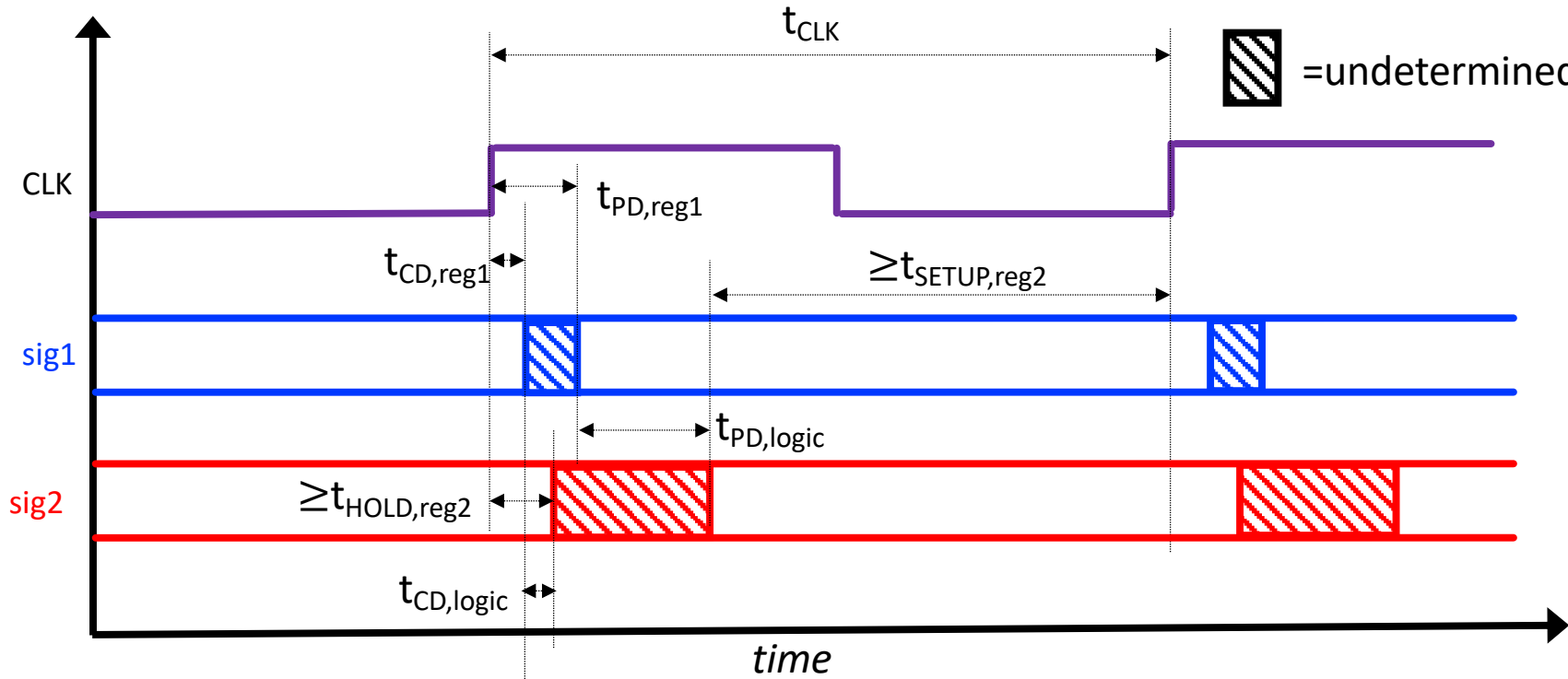
Receiving edge arrives before the launching edge (negative skew)

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

Timing



— =determined state
 ▨ =undetermined state



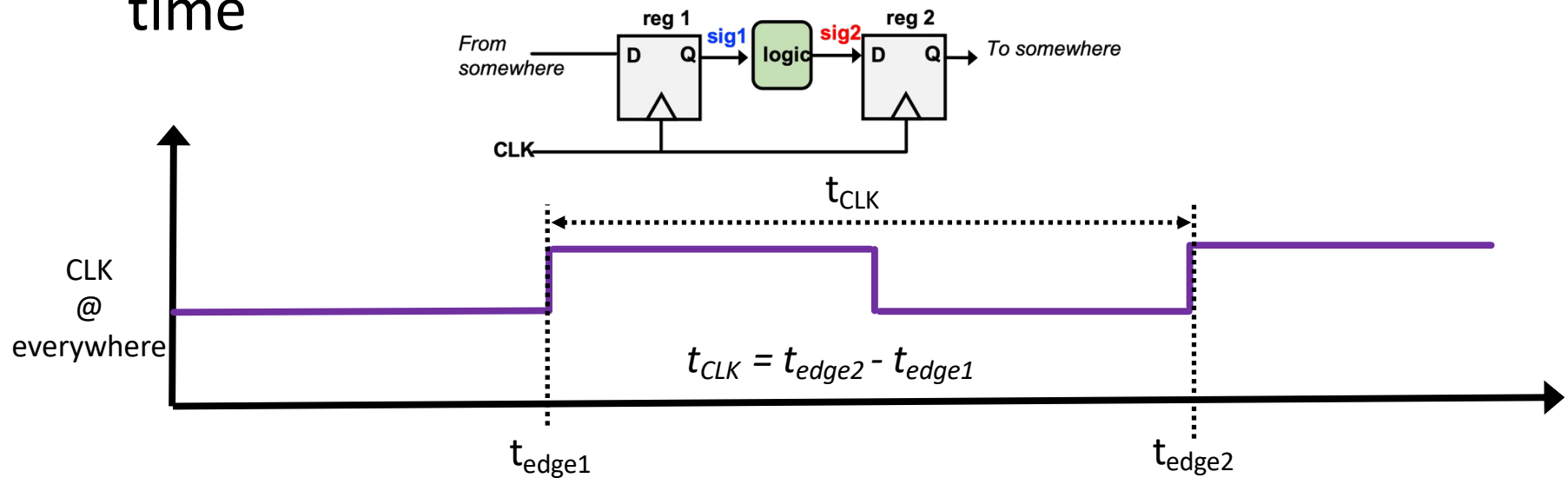
**Two Requirements/
Conclusions:**

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

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

How does Skew Affect Things?

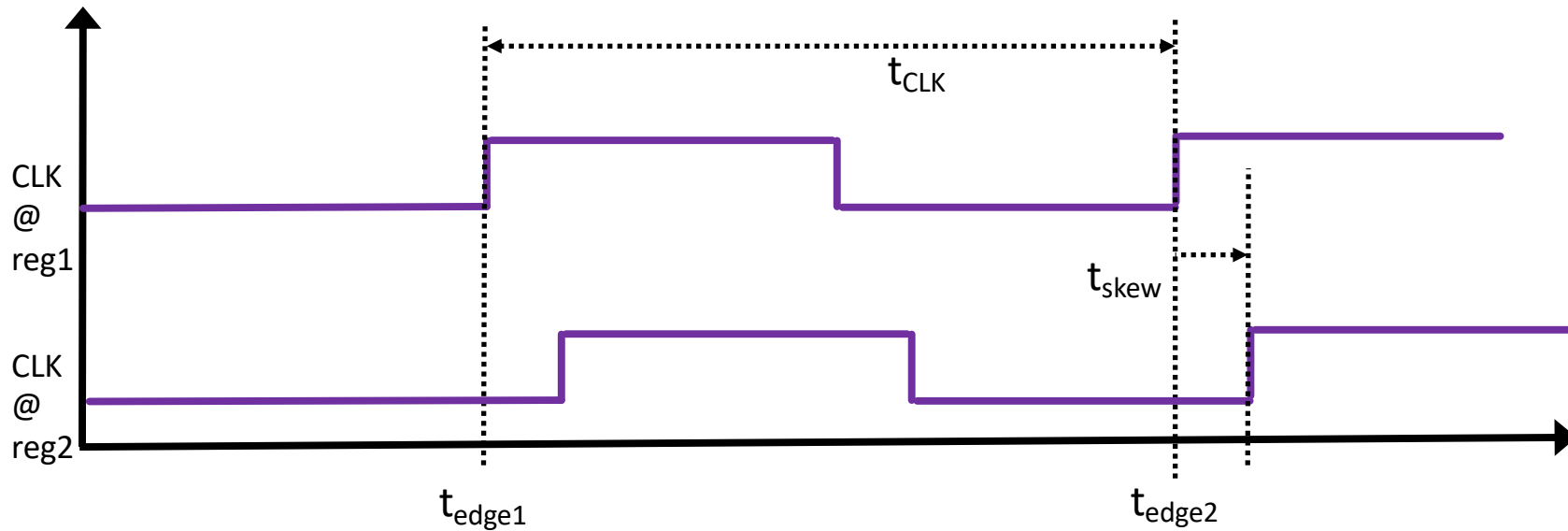
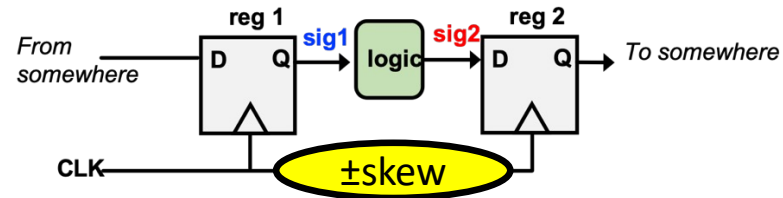
- Originally in our model circuit, we assume all devices experience the clock edges at the same time



- Setup equation $t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$
- was actually short-hand for:

$$t_{edge1} + t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{edge2}$$

With Skew



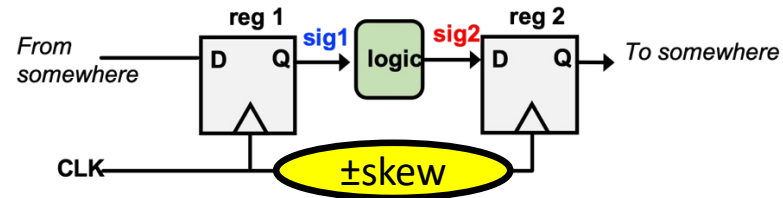
- The equation turns into:

$$t_{edge1} + t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{edge2} + t_{skew}$$

- Or since $t_{CLK} = t_{edge2} - t_{edge1}$

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

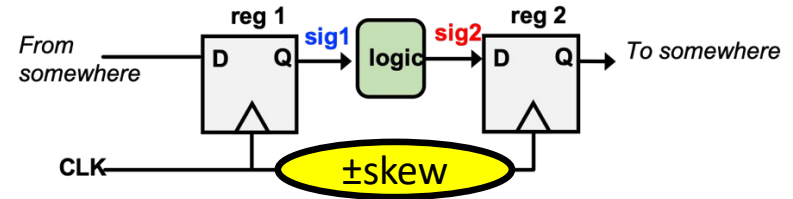
With Skew



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

- If that's now our modified setup equation...
- Positive skew is easier to satisfy
- Negative skew is harder to satisfy
- But you still have degree of freedom with $t_{PD,logic}$
...maybe you can change that?
- And you could also increase t_{clk} as well.

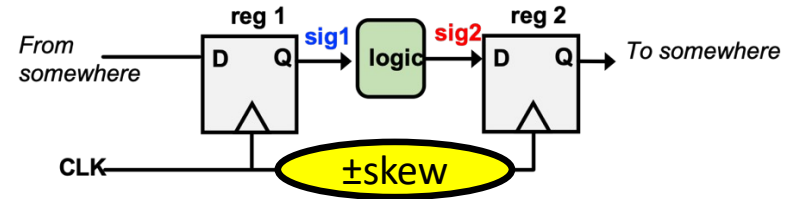
What about Hold Time?



- If the second register is getting its clock edge t_{skew} after the first register that means it needs hold the values at the input of reg2 for t_{skew} longer :/
- Hold Equation gets modified to be:

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

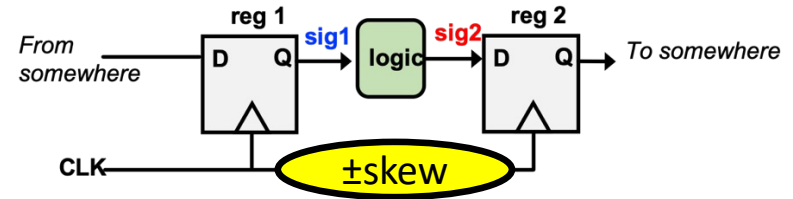
What about Hold Time?



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

- The “growth” from skew is not on low side of inequality so...
- Positive skew makes eq harder to satisfy.
 - Further there’s nothing you can do since contamination delays are usually very low and beyond our control
- Negative skew makes eq easier to satisfy.

Conclusions



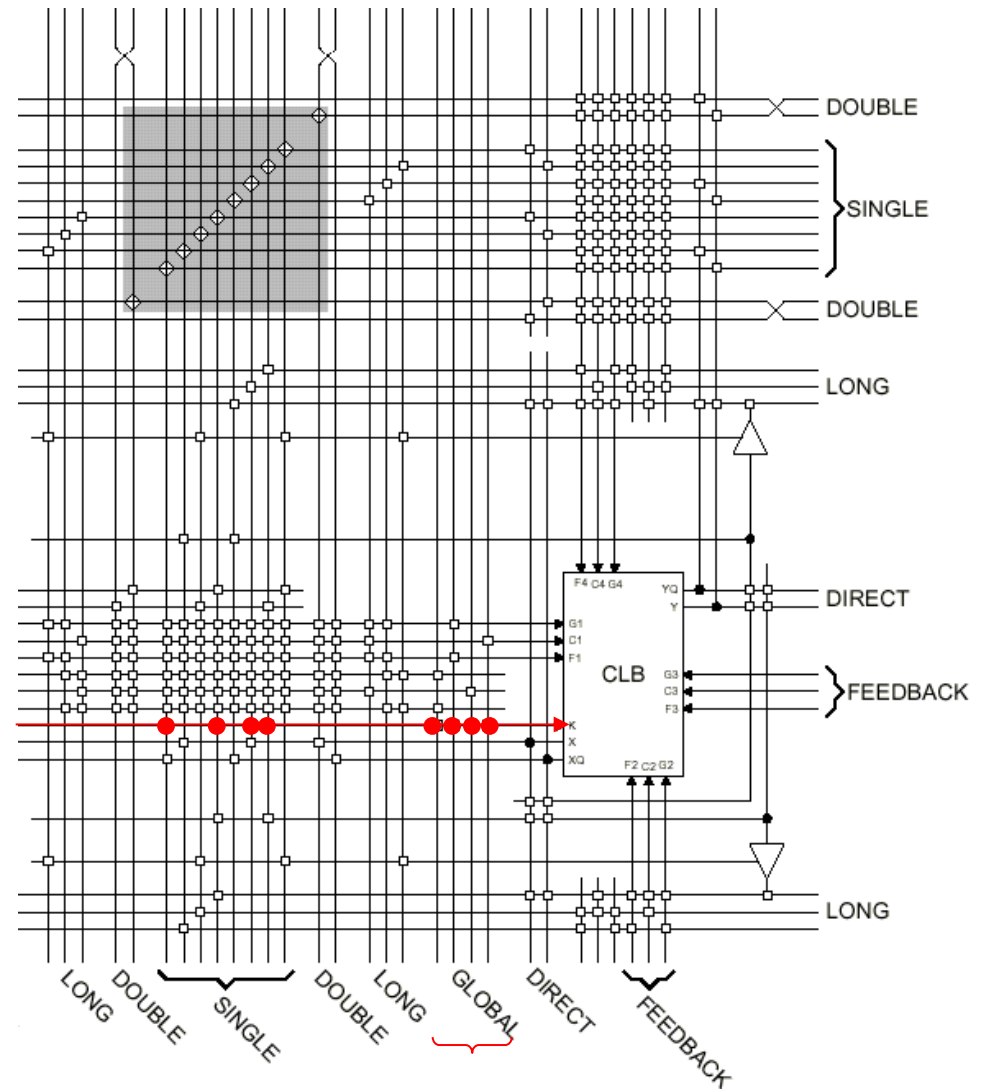
$$t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{clk} + t_{skew}$$

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

- **Positive clock skew** improves the minimum cycle time of our design but makes it harder to meet register hold times.
- **Negative clock skew** hurts the minimum cycle time of our design but makes it easier to meet register hold times.
- *Positive skew* is tougher to deal with

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

Other Problems...

- Stable Clock:



- Jittery Clock:



Other Problems...

- 50% Duty Cycle Clock



- Not 50% Duty Cycle Clock



Duty Cycle

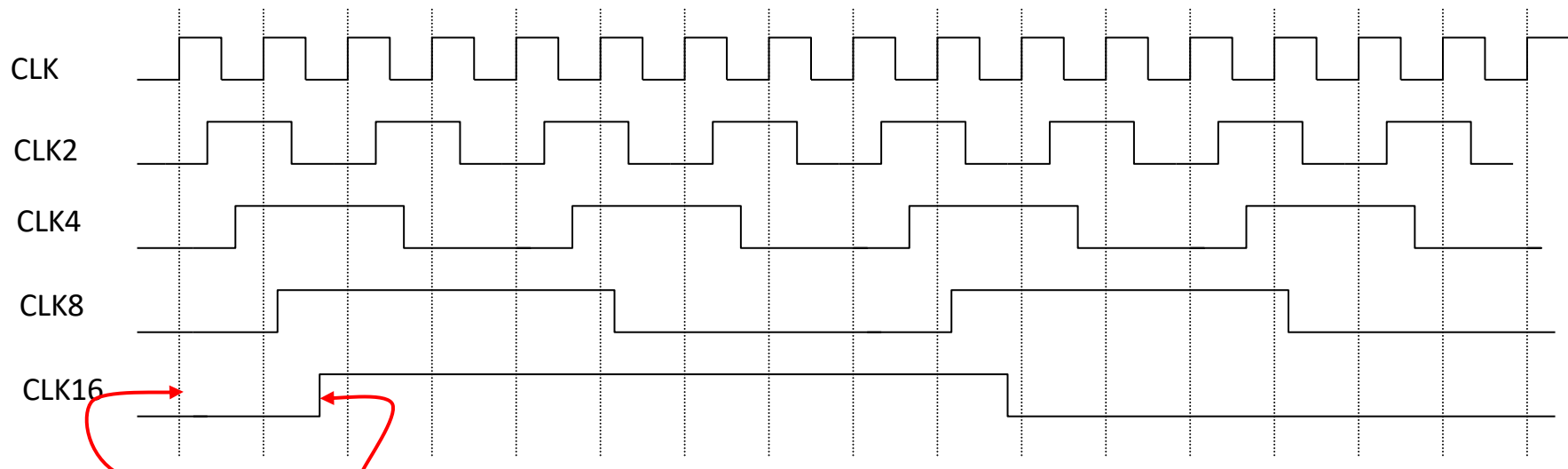
- Another reason we try avoid using neg-edge of clocks is it makes timing a lot easier
- Clocks will tend to deviate from 50/50 duty cycle due to variations/asymmetries in p-channel/n-channel transistor behavior
- Clock Buffers and things will try to clean this up, but it can be tough

Goal: use as few clock domains as possible

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

```
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;
```

*No! don't do
it this way*



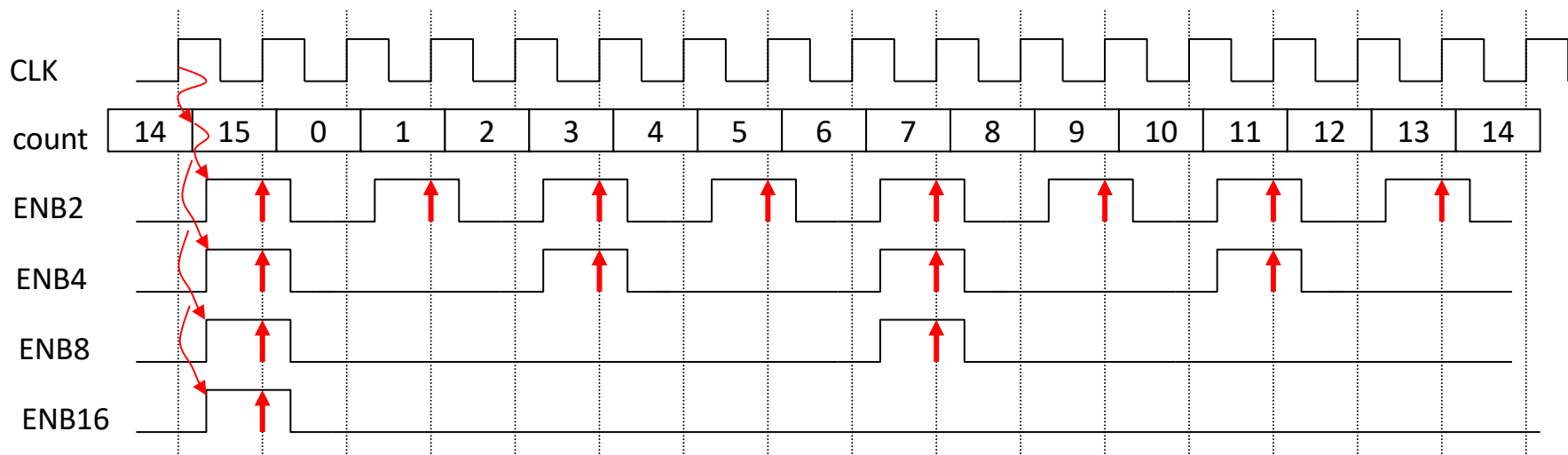
Very hard to have synchronous communication between clk and clk16 domains... Can lead to lots of timing violations!

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);
```

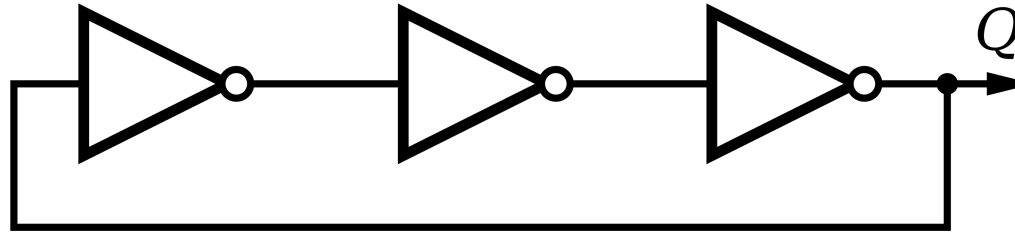
```
always_ff @(posedge clk)
  if (enb2) begin
    // get here every 2nd cycle
  end
```



= clock edge selected by enable signal

How to Make Frequencies and Clocks

Where do we get frequencies?



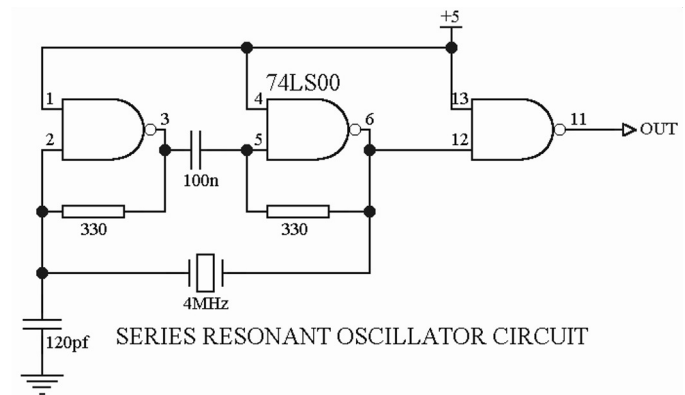
- Particular combinational circuits that are fed back onto themselves so that they cannot be stable can be made to form oscillators.
- The ring oscillator above is a classic example.
- There is no stable set of output states so this circuit perpetually oscillates.
- Period of oscillation is based on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

9/24/24

<https://fpga.mit.edu/6205/F24>

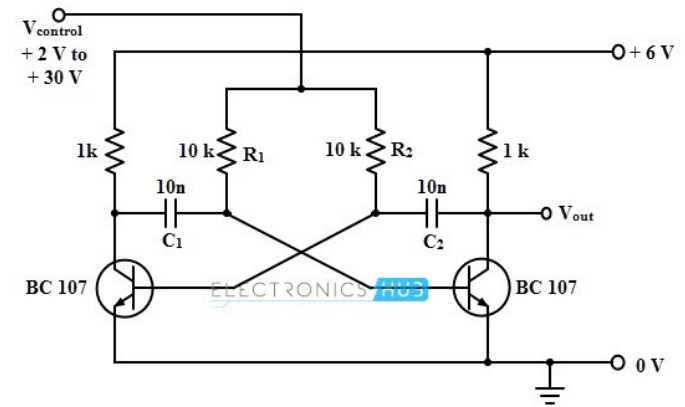
59

High Frequencies

- Very hard to get a crystal oscillator to operate above ~200 MHz (7th 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!

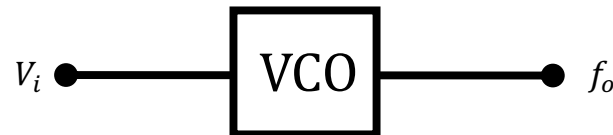
Voltage Controlled Oscillator

- 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 \rightarrow higher frequency oscillation



A simple VCO (not type found in FPGA)

- Block Diagram

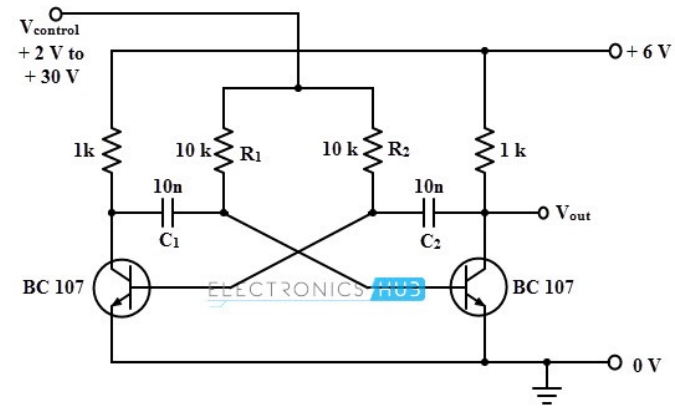


Voltage Controlled Oscillator

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



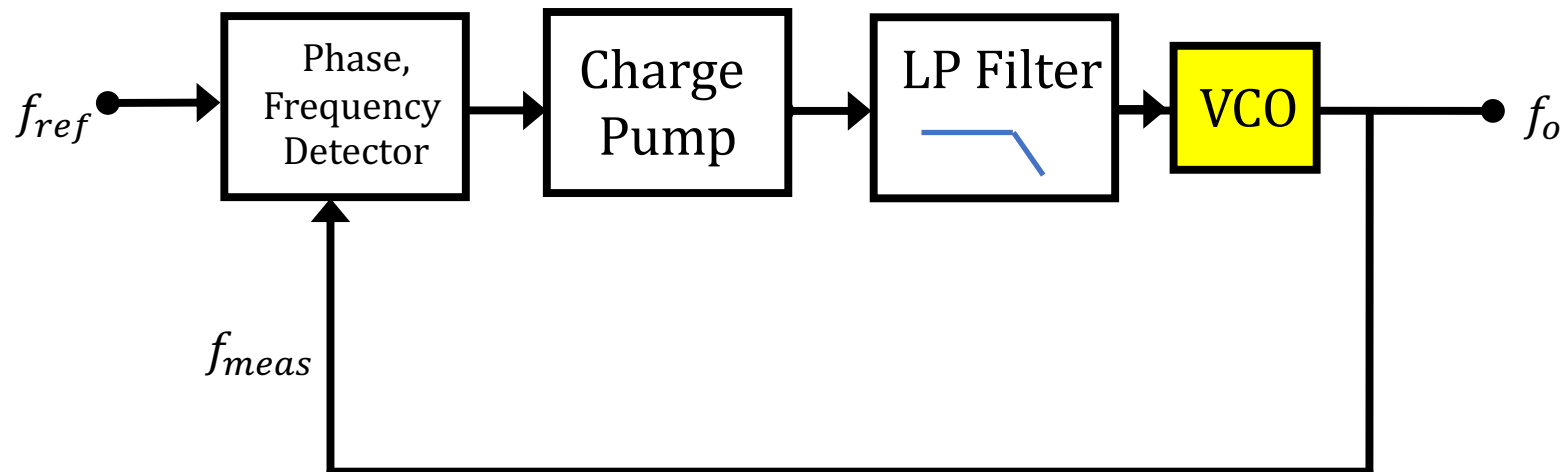
- Pick the voltage V_i that is needed to get the frequency we want f_o ?
- That's gotta be specified right?



A simple VCO (not type found in FPGA)

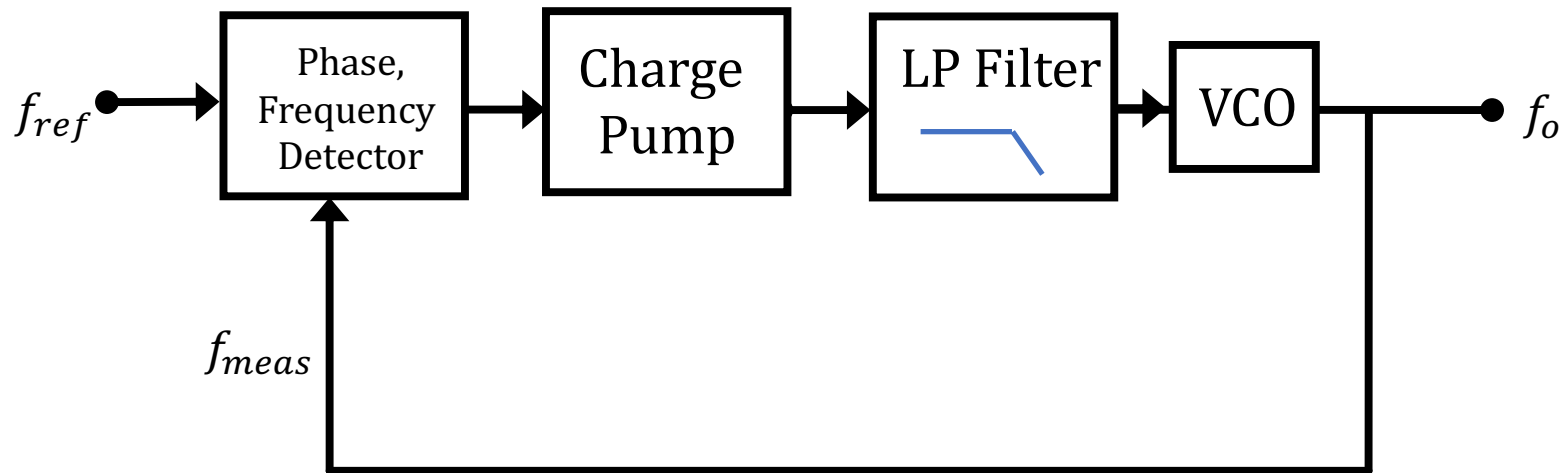
Phase Locked Loop

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

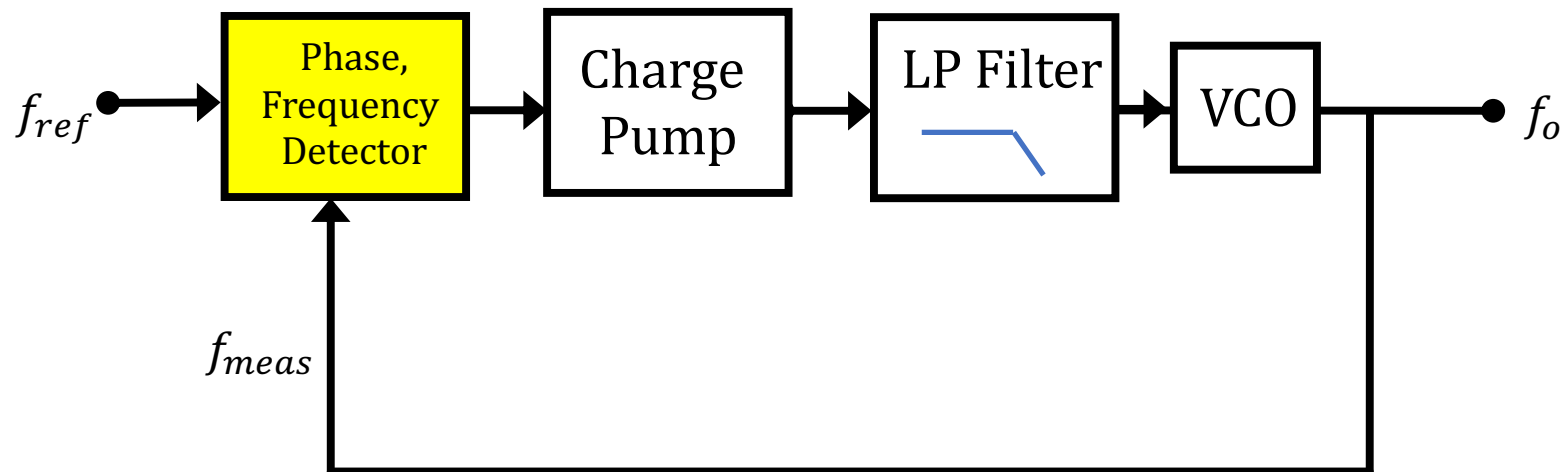


Phase Locked Loop

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

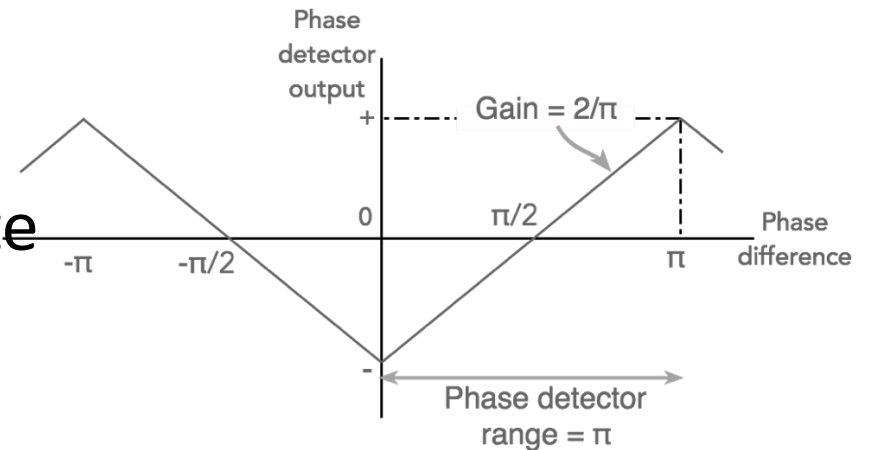


Phase, Frequency Detector



Phase Detector

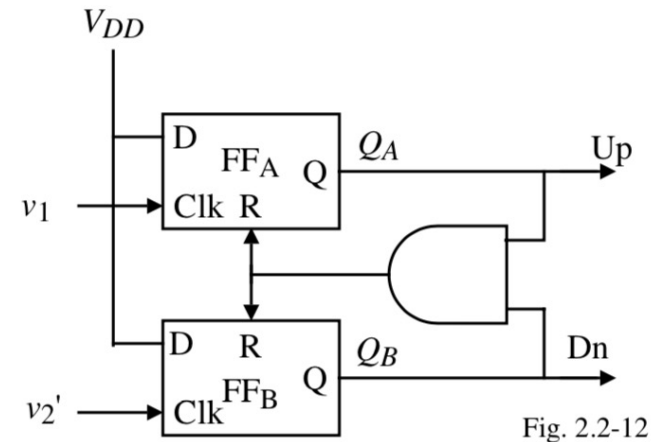
- Can be a simple XOR, XNOR gate
 - Low-pass the output



- 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

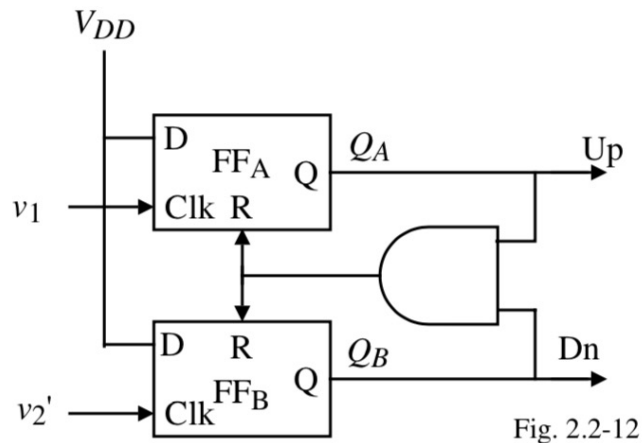
- Detects both change and which clock signal is consistently leading the other one
- Using MOSFETs you charge/discharge a capacitor accordingly which also with some resistors low-pass filters the signal
- The output voltage is then roughly proportional to the frequency error!



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

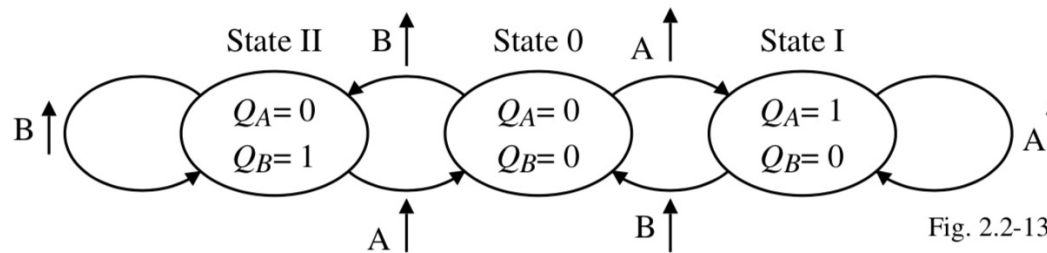
<http://www.globalspec.com/reference/72819/203279/2-7-phase-detectors-with-charge-pump-output>

Phase Frequency Detection



- 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

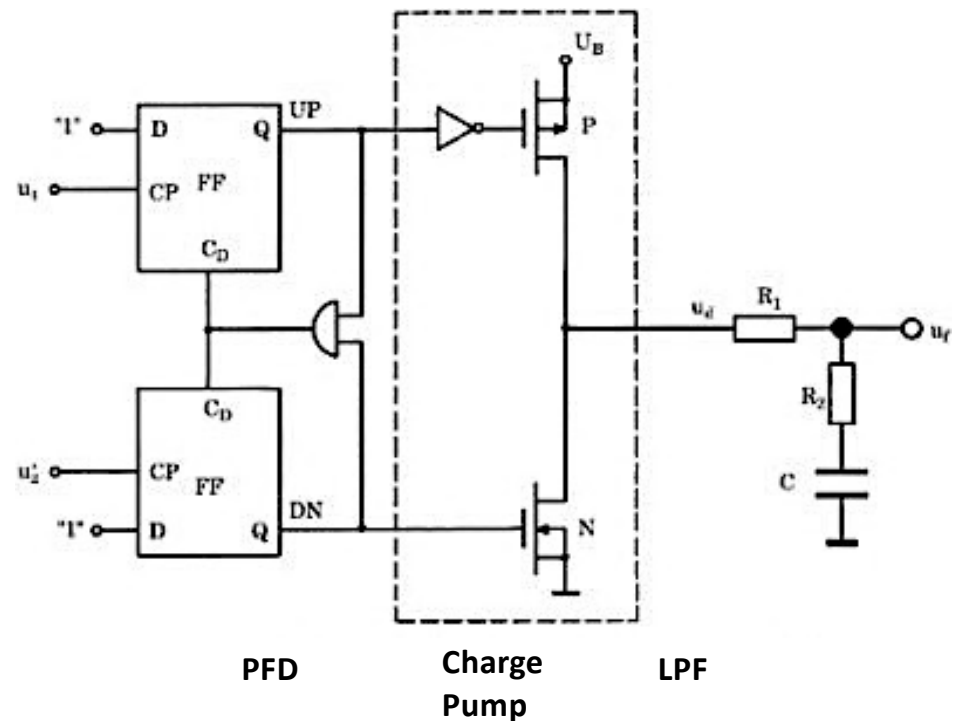
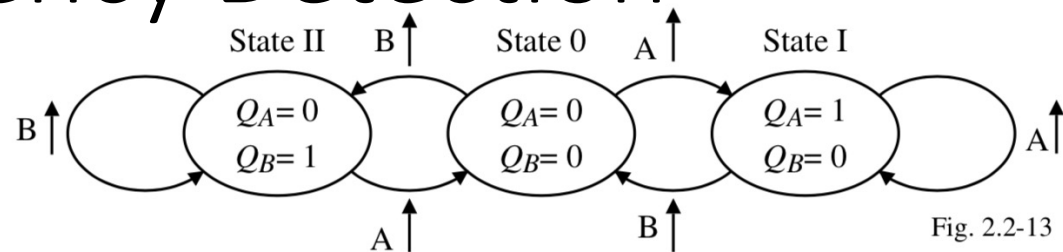
PFD State Diagram:



[1.pallen.ece.gatech.edu/Academic/ECE_6440/Summer_2003/L070-DPLL\(2UP\).pdf](http://1.pallen.ece.gatech.edu/Academic/ECE_6440/Summer_2003/L070-DPLL(2UP).pdf)

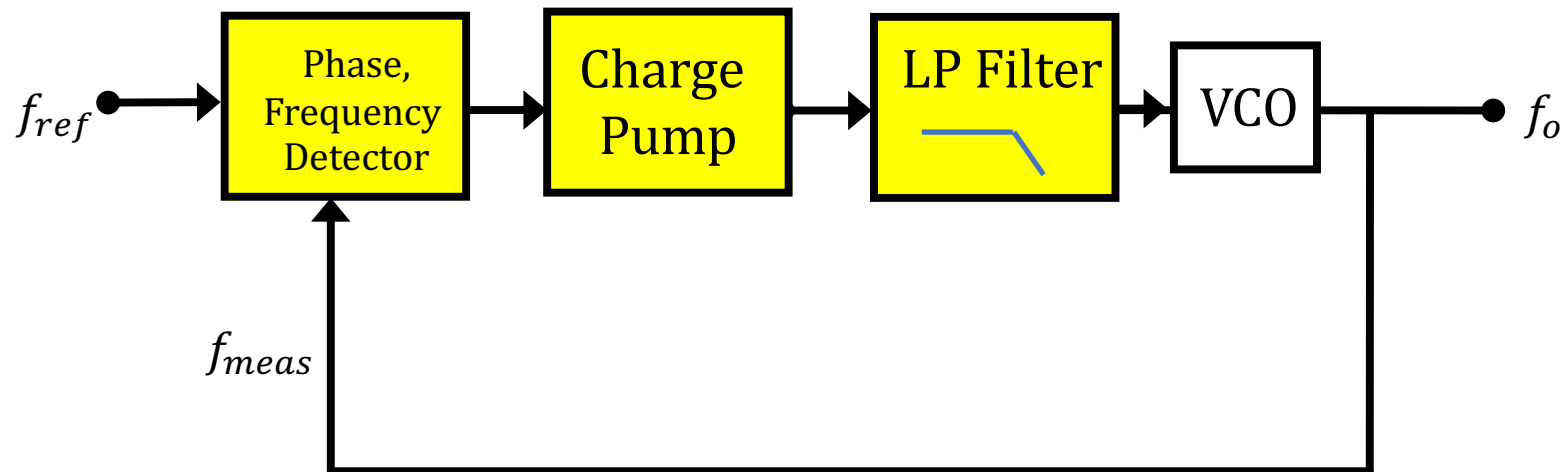
Phase-Frequency Detection

- 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



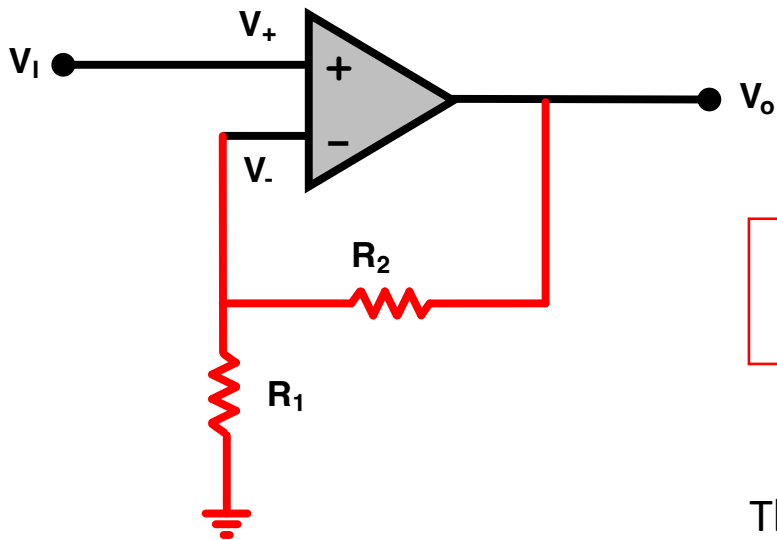
<http://www.globalspec.com/reference/72819/203279/2-7-phase-detectors-with-charge-pump-output>

PFD, Charge Pump, LP Filter



- So this circuit can make $f_o = f_{ref}$ 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:

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

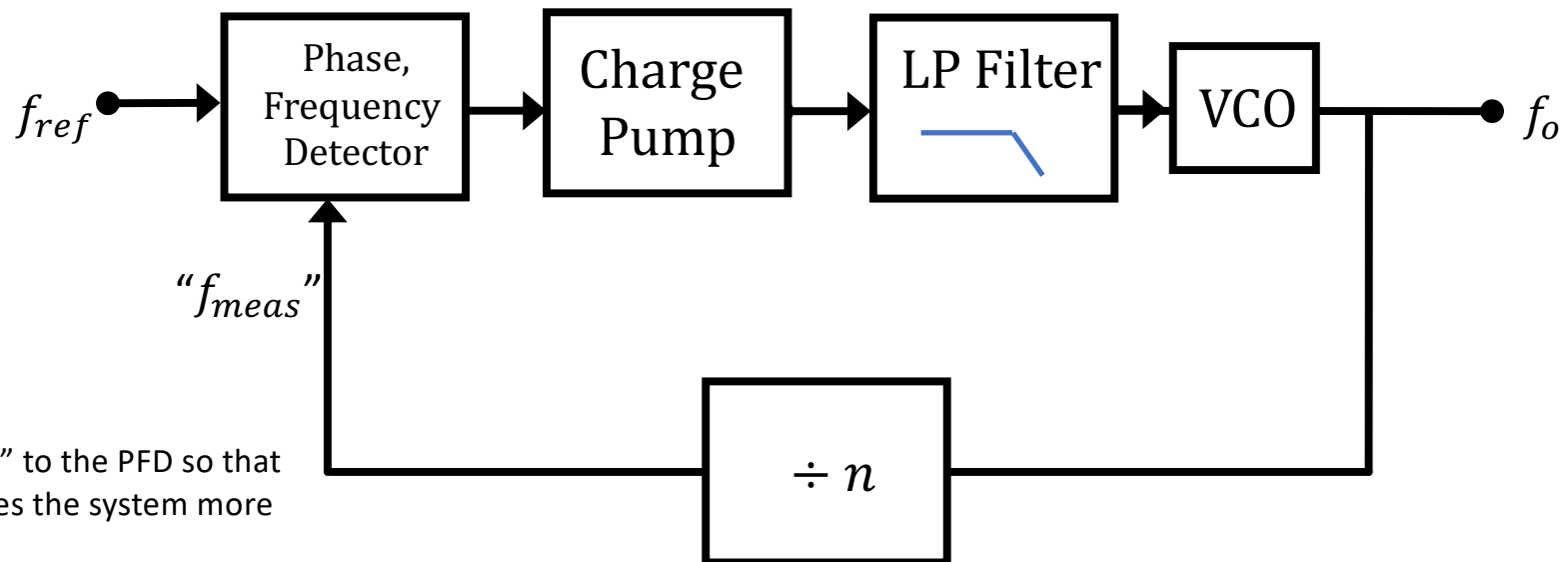
The gain of a “non-inverting amplifier”

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

Same Idea with Phase Locked Loops!

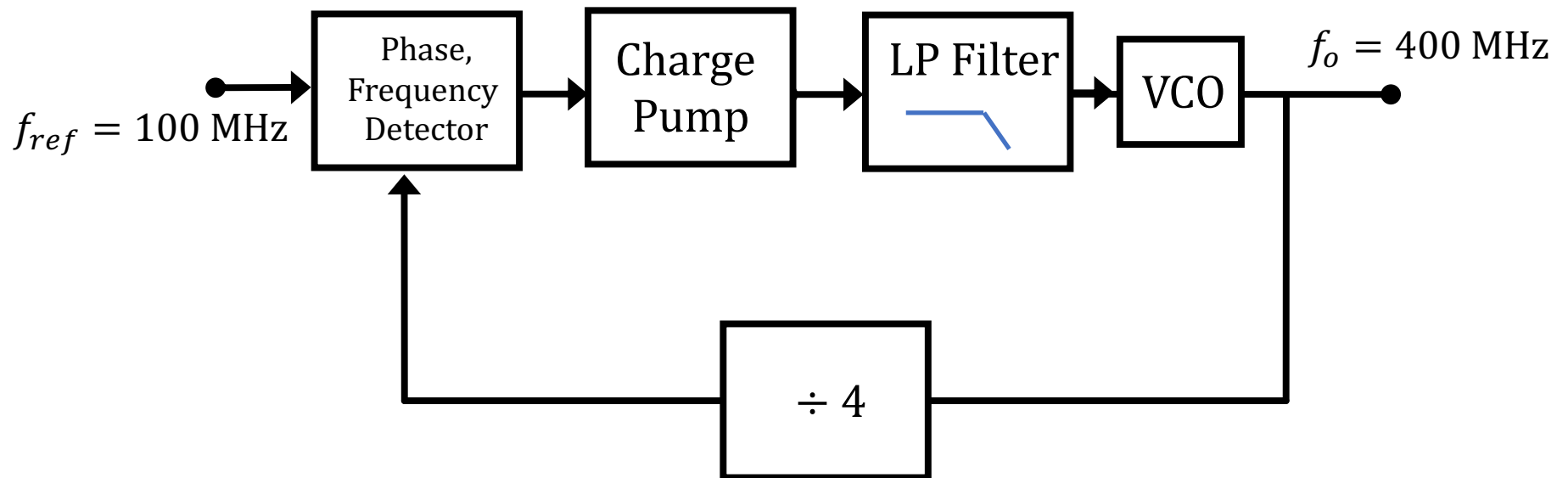
Use a Clock Divider in Feedback Path!

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



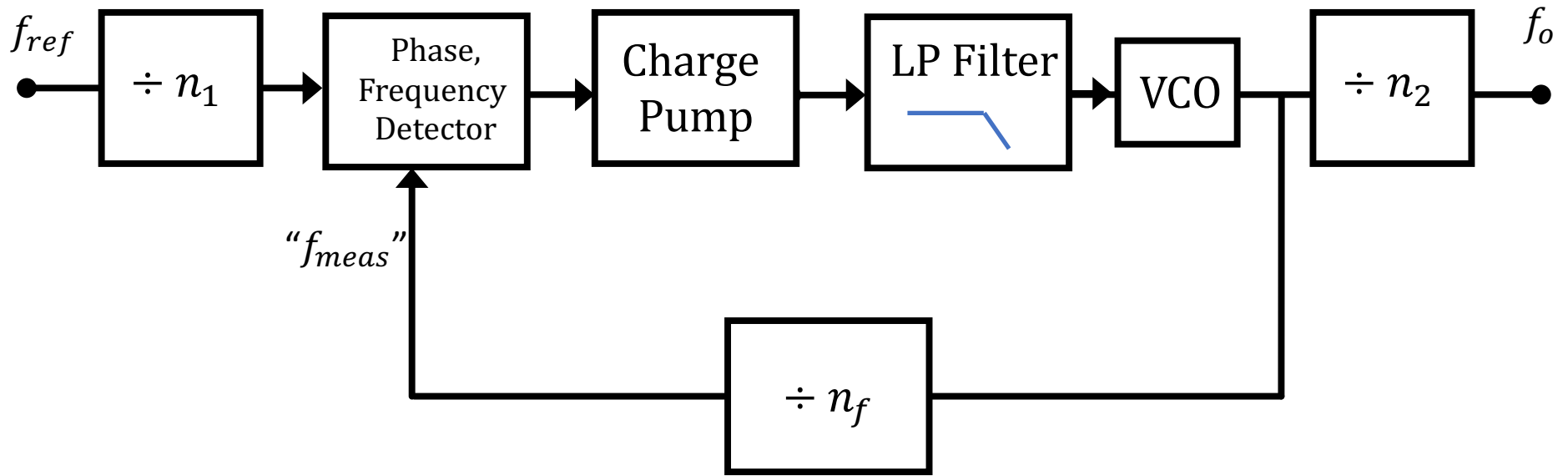
We "lie" to the PFD so that it pushes the system more

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?

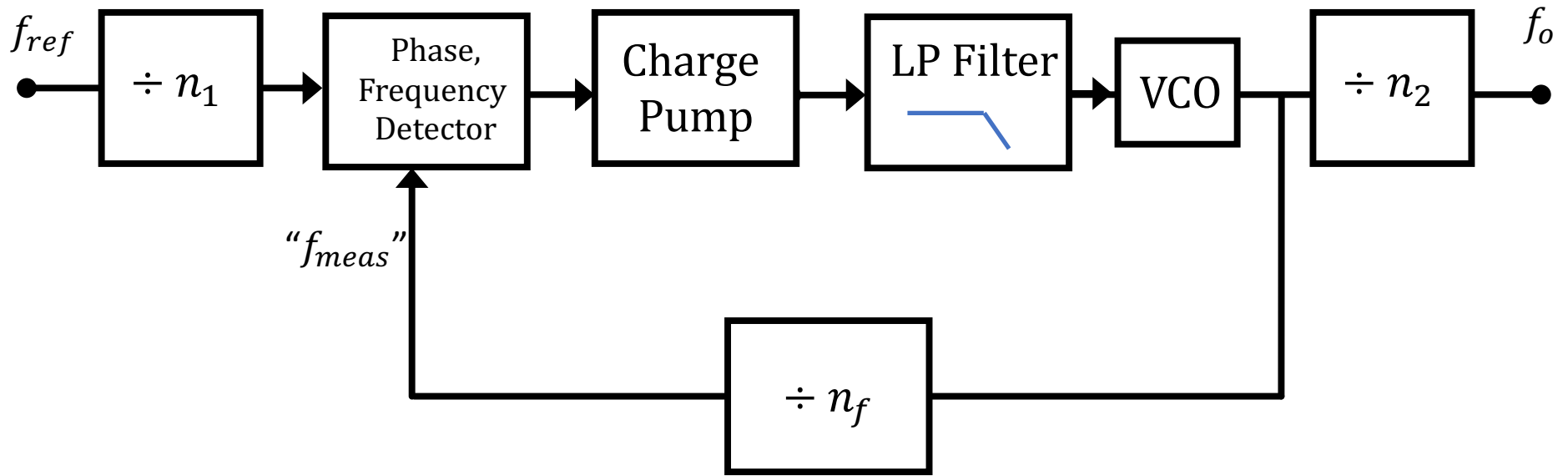
- How to make 65 MHz from 100 MHz?
 - Divide down (not too low)
 - Multiply up (not too high)
 - Divide down for final product

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),
```

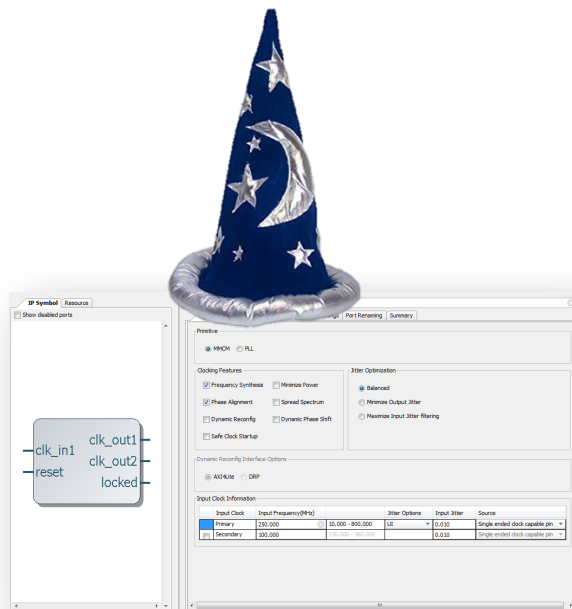
Add a Pre- and Post-Divider for Flex



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):

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

In Week 04

- We'll build HDMI video from scratch.
- For 720p we'll need:
 - a clock at 74.25 MHz (for the pixels)
 - A clock at 371.25 MHz (for the bits of the pixels to be sent serially)
 - We'll use this clock along with a device that is built to run using always @(posedge clk or negedge clk) to get 742.25 MHz of data out to drive the 720p data.

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 btn[0] is pushed:
 - DIVIDE the 16 bit numbers

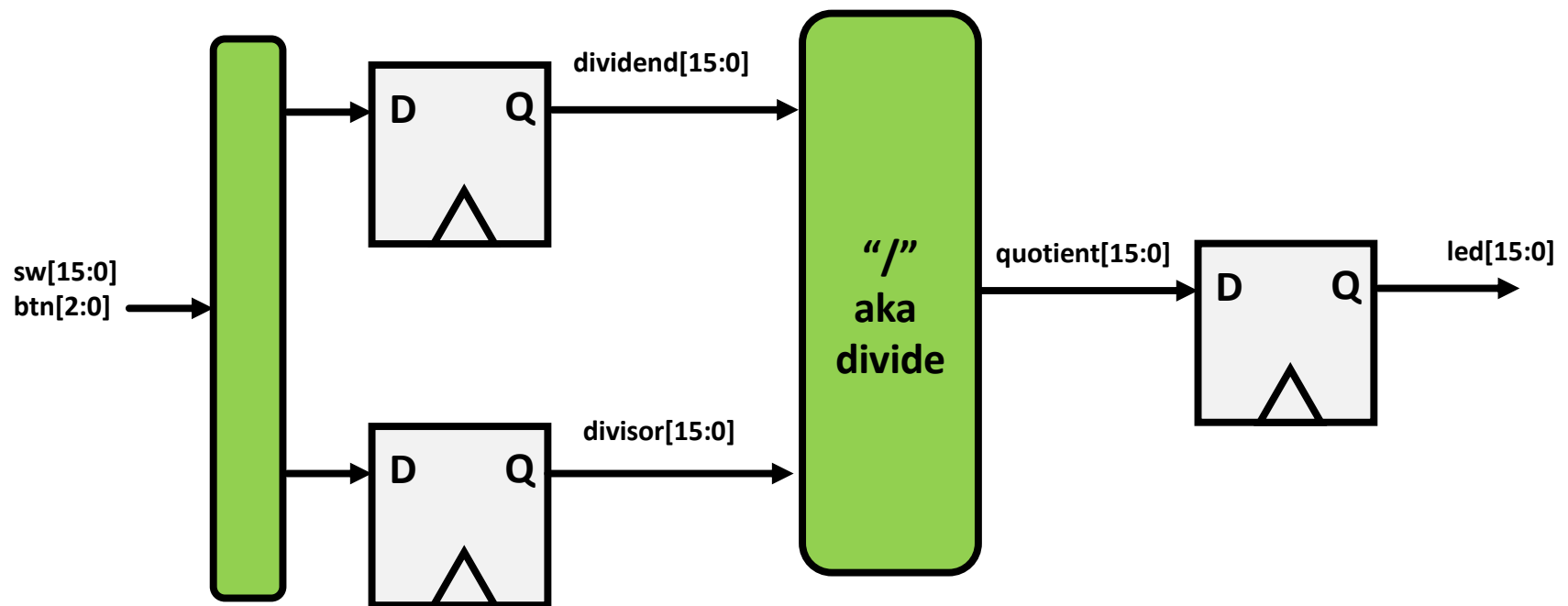
```
`timescale 1ns / 1ps
`default_nettype none

module top_level(
    input wire clk_100mhz, //clock @ 100 mhz
    input wire [15:0] sw, //switches
    input wire [3:0] btn, //all four momentary button switches
    output logic [15:0] led //just here for the funs
);

    logic [3:0] old_btn;
    logic [15:0] quotient;
    logic [15:0] dividend;
    logic [15:0] divisor;
    assign led = quotient;

    always_ff @(posedge clk_100mhz)begin
        for (int i=0; i<4; i=i+1)begin
            old_btn[i] <= btn[i];
        end
    end
    always_ff @(posedge clk_100mhz)begin
        if (btn[0] & ~old_btn[0])begin
            quotient <= dividend/divisor; //divide
        end
        if (btn[1] & ~old_btn[1])begin
            dividend <= sw; //divide //load dividend
        end
        if (btn[2] & ~old_btn[2])begin
            divisor <= sw; //divide //load dividend
        end
    end
endmodule

`default_nettype wire
```

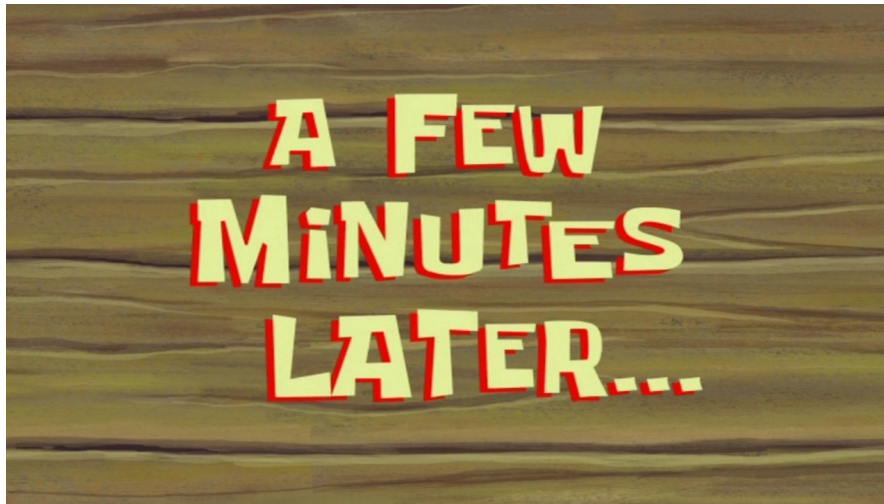


Let's Build it.

- Terminal Output:

```
jodalyst@Josephs-MBP lec06 % ./remote/r.py build.py build.tcl hdl/* xdc/* obj  
  
...  
...  
...  
  
Writing bitstream obj/final.bit...  
INFO: [Vivado 12-1842] Bitgen Completed Successfully.  
INFO: [Project 1-1876] WebTalk data collection is mandatory when using a ULT device.  
To see the specific WebTalk data collected for your design, open the  
usage_statistics_webtalk.html or usage_statistics_webtalk.xml file in the  
implementation directory.  
INFO: [Common 17-83] Releasing license: Implementation  
7 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.  
write_bitstream completed successfully  
write_bitstream: Time (s): cpu = 00:00:04 ; elapsed = 00:00:14 . Memory (MB): peak =  
2729.707 ; gain = 206.934 ; free physical = 2837 ; free virtual = 8407
```

"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 1322:

Verification completed successfully
Phase 20 Verifying routed nets | Checksum: 12923a084

Time (s): cpu = 00:00:12 ; elapsed = 00:00:13 . Memory (MB): peak = 2520.699 ; gain = 0.000 ; free physical = 3116 ; free virtual = 8674

Phase 21 Depositing Routes
Phase 21 Depositing Routes | Checksum: 14a6fdc22

Time (s): cpu = 00:00:12 ; elapsed = 00:00:13 . Memory (MB): peak = 2520.699 ; gain = 0.000 ; free physical = 3116 ; free virtual = 8674

Phase 22 Post Router Timing
INFO: [Route 35-20] Post Routing Timing Summary | WNS=-21.399| TNS=-129.552| WHS=0.090 | THS=0.000 |

Phase 22 Post Router Timing | Checksum: 1a0e6c79b

Time (s): cpu = 00:00:12 ; elapsed = 00:00:13 . Memory (MB): peak = 2520.699 ; gain = 0.000 ; free physical = 3116 ; free virtual = 8674

CRITICAL WARNING: [Route 35-39] The design did not meet timing requirements. Please run report_timing_summary for detailed reports.

Resolution: Verify that the timing was met or had small violations at all previous steps (synthesis, placement, power_opt, and phys_opt). Run report_timing_summary and analyze individual timing paths.

INFO: [Route 35-253] TNS is the sum of the worst slack violation on every endpoint in the design. Review the paths with the biggest WNS violations in the timing reports and modify your constraints or your design to improve both WNS and TNS.

INFO: [Route 35-16] Router Completed Successfully

Phase 23 Post-Route Event Processing
Phase 23 Post-Route Event Processing | Checksum: 3725a886

Time (s): cpu = 00:00:12 ; elapsed = 00:00:13 . Memory (MB): peak = 2520.699 ; gain = 0.000 ; free physical = 3116 ; free virtual = 8674

9/24/24

<https://fpga.mit.edu/6205/F24>

Look at

post_route_timing.rpt

Timing Report

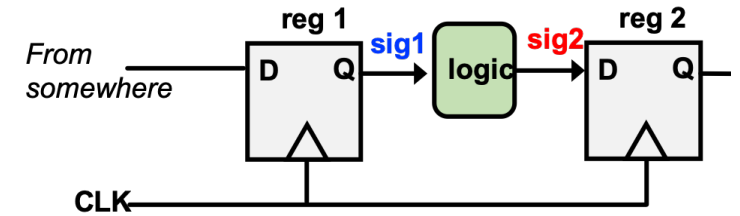
```
Slack (VIOLATED) : -21.399ns (required time - arrival time)
Source:          dividend_reg[15]/C
                 (rising edge-triggered cell FDRE clocked by gclk {rise@0.000ns fall@4.000ns period=10.000ns})
Destination:    quotient_reg[0]/D
                 (rising edge-triggered cell FDRE clocked by gclk {rise@0.000ns fall@4.000ns period=10.000ns})
Path Group:     gclk
Path Type:      Setup (Max at Slow Process Corner)
Requirement:    10.000ns (gclk rise@10.000ns - gclk rise@0.000ns)
Data Path Delay: 31.483ns (logic 21.642ns (68.742%) route 9.841ns (31.258%))
Logic Levels:   82 (CARRY4=80 LUT2=1 LUT3=1)
Clock Path Skew: 0.026ns (DCD - SCD + CPR)
  Destination Clock Delay (DCD): 4.926ns = ( 14.926 - 10.000 )
  Source Clock Delay (SCD): 5.079ns
  Clock Pessimism Removal (CPR): 0.179ns
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?

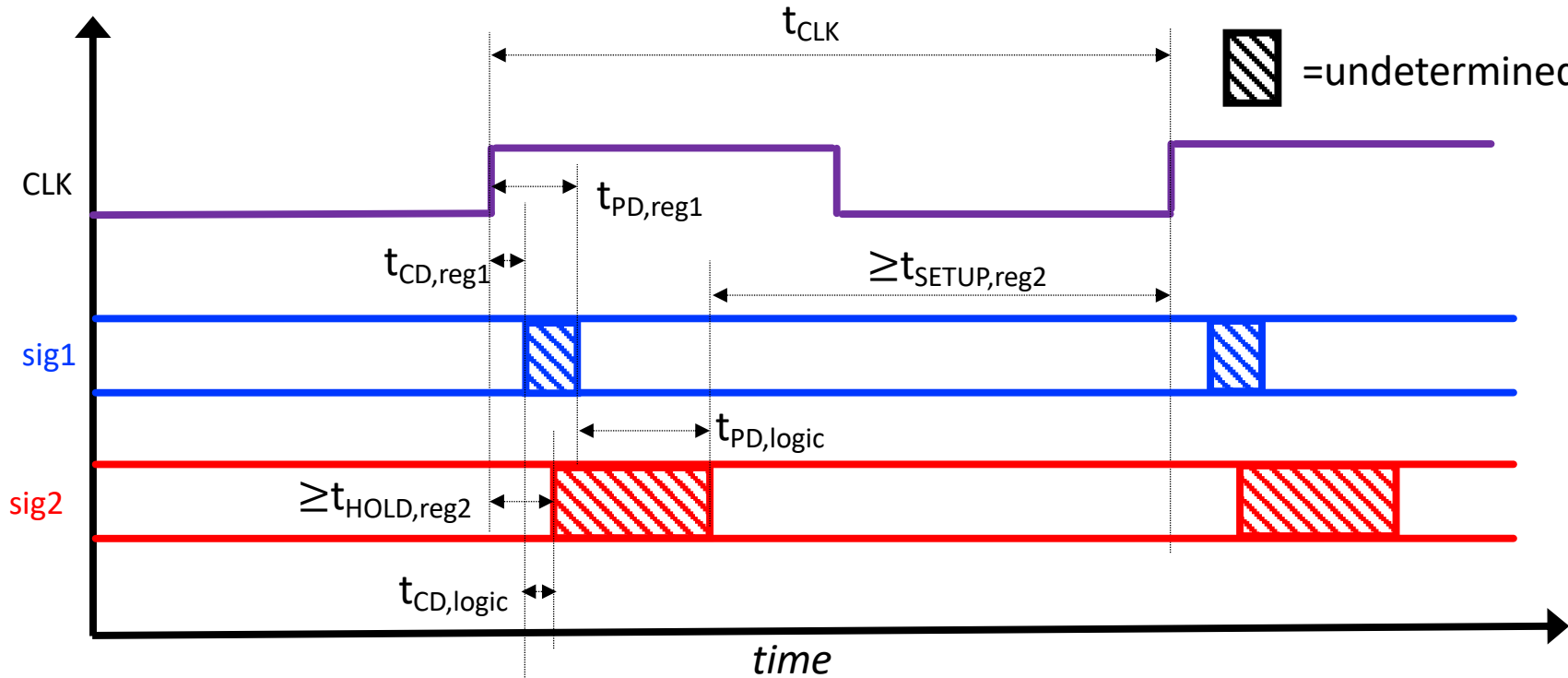
- Slack: measure of how safe your timing is
- The two big timing constraints we worry about are related to setup and hold
- Therefore there are two Slack values:
 - Setup slack: $t_{\text{required}} - t_{\text{actual}}$
 - Hold slack: $t_{\text{actual}} - t_{\text{required}}$

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

Timing Diagram



— = determined state
 = undetermined state

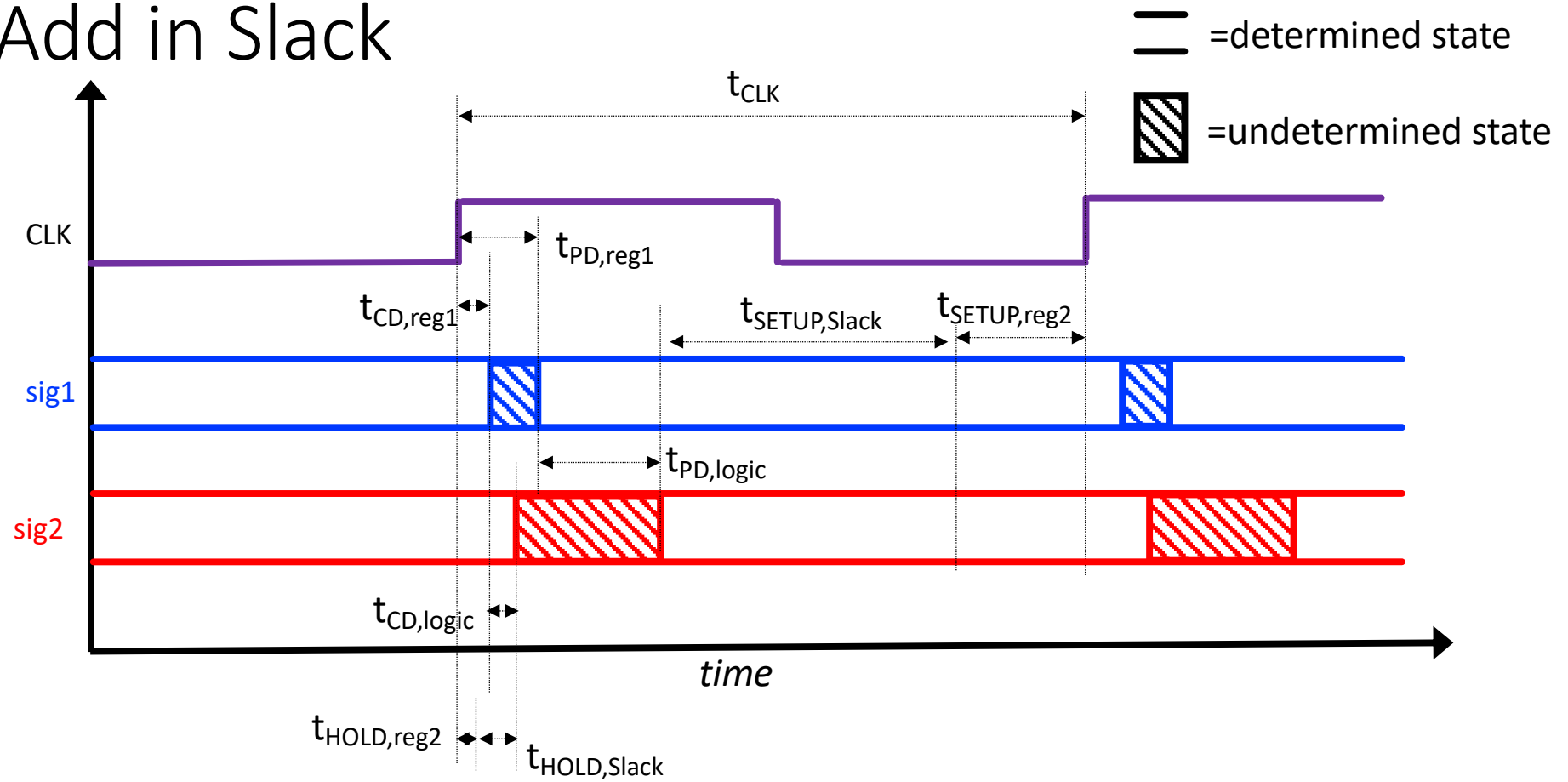


**Two Requirements/
Conclusions:**

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

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

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*

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

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

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) : -21.399ns (required time - arrival time)
Source:          dividend_reg[15]/C
                 (rising edge-triggered cell FDRE clocked by gclk {rise@0.000ns fall@4.000ns period=10.000ns})
Destination:    quotient_reg[0]/D
                 (rising edge-triggered cell FDRE clocked by gclk {rise@0.000ns fall@4.000ns period=10.000ns})
Path Group:     gclk
Path Type:      Setup (Max at Slow Process Corner)
Requirement:    10.000ns (gclk rise@10.000ns - gclk rise@0.000ns)
Data Path Delay: 31.483ns (logic 21.642ns (68.742%) route 9.841ns (31.258%))
Logic Levels:   82 (CARRY4=80 LUT2=1 LUT3=1)
Clock Path Skew: 0.026ns (DCD - SCD + CPR)
  Destination Clock Delay (DCD): 4.926ns = ( 14.926 - 10.000 )
  Source Clock Delay (SCD): 5.079ns
  Clock Pessimism Removal (CPR): 0.179ns
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

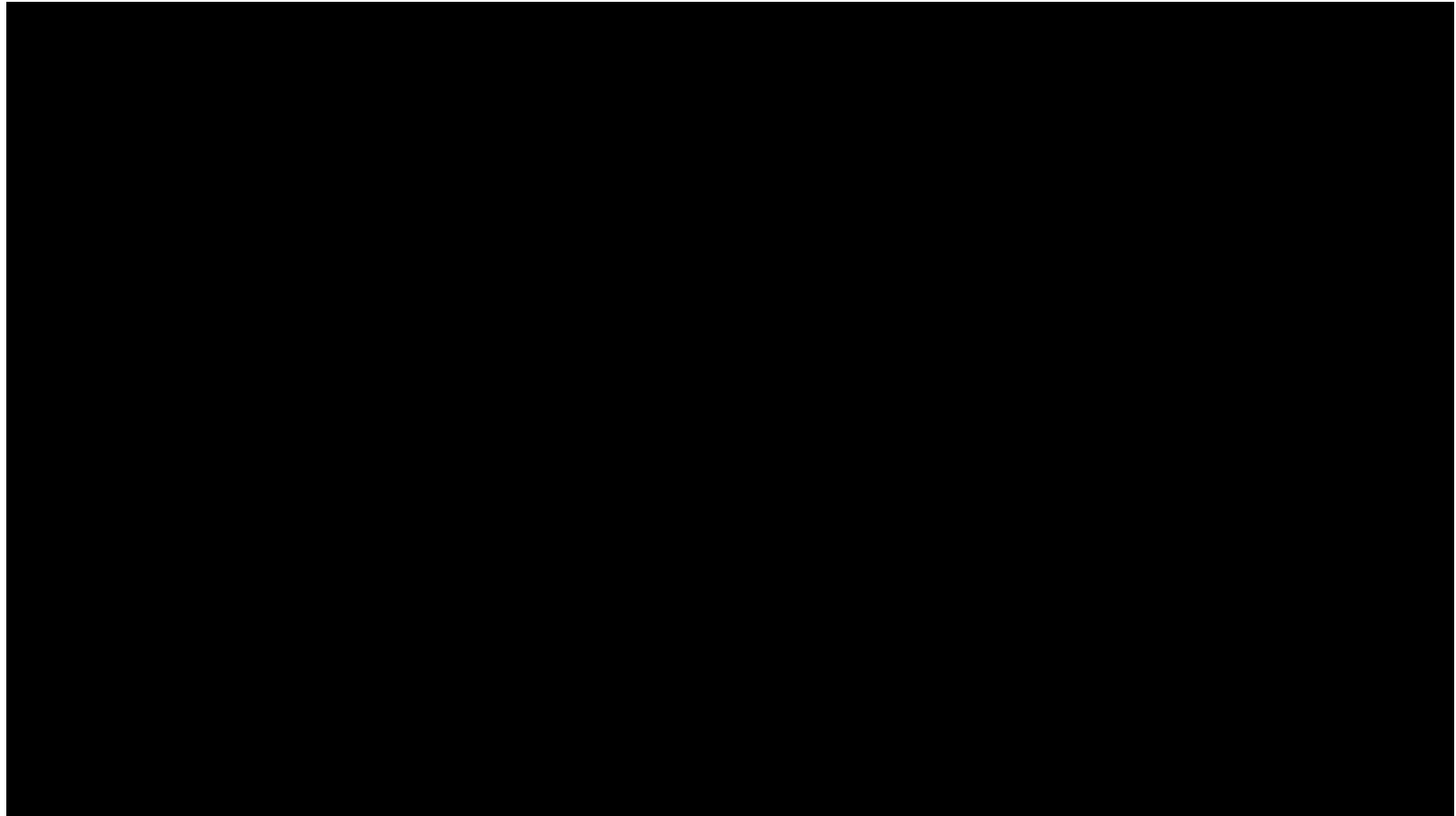
- By default Vivado only gives you a few offending paths (our default is one) and it provides them in order of worst to best
- You can ask for more paths using different arguments;

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

Final Projects Coming Up

- In another ~week or so, we have to start thinking about planning on starting to get going on final projects.
- First part of that is teaming and teams benefit from targeting shared goals
- On the site, we'll put up an archive of final projects

Past Project (with Microphone)



Sudoku Solver

