Lecture 9:

Clocking for High Performance Processors

Computer Systems Lab

Stanford University

horowitz@stanford.edu

Overview

Reading

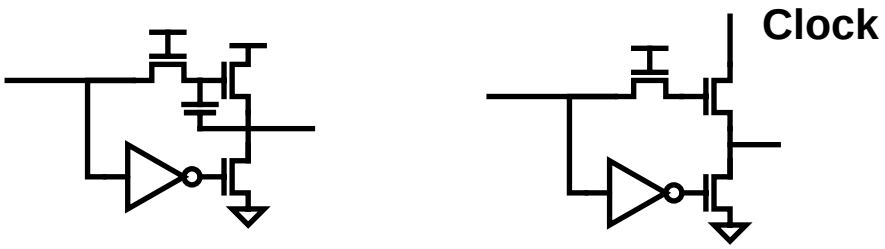
Bailey	Clocking on the Alpha
Stojanovic	Evaluation of different latches
Harris	Skew tolerant domino design

Introduction

In addition to the design of the circuits on a chip, clocking and flop/latch design has a large influence on the circuit's power and performance. As was discussed in EE271, the role of the clocks is to keep signals correlated in time. There are many approaches to this problem, some even that don't require clocks. If clocks are used, it is critical to minimize the overhead caused by the clocks, which includes clock skew, and latch/flop overheads. This lecture looks at the sequencing issue and explores a couple of approaches to this problem including eliminating clocks (self-timing). The next lecture will look at flop and latch design in more detail. We start with a brief look at history of clock design.

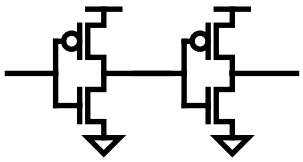
History

- Clocking was critical issue in early 80's
  - nMOS design
  - Needed fast but low power buffers
  - Name of the game was bootstrapping



History, cont'd

- CMOS changed the rules
  - Solved the clock buffer problem



- Clock circuitry became less interesting
  - Until the 90's …
- Clocks once again are a difficult circuit issue

Overview of Talk

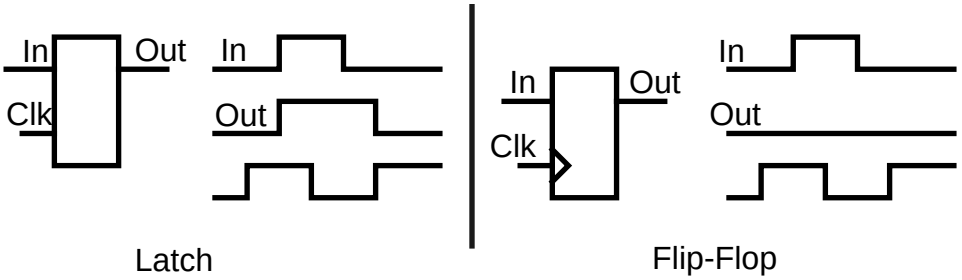
- Background:
  - Role of clocks
  - Why clocks are bad (clock overhead)
- Self-timed design
  - Why no clocks are bad
- Real world - (how to live with badness)
  - Clock distribution issues
  - Skew tolerant designs
- Summary

Common View of Clock's Function

Clocks work with Latch or Flip-Flop to hold state

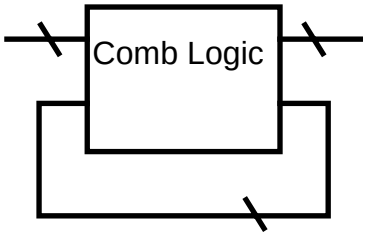
- Latch
  - Stores data when the clock is low

- Flip-Flop
  - Stores In when clock rises



Another View

If the delay of every path **was EXACTLY** the same

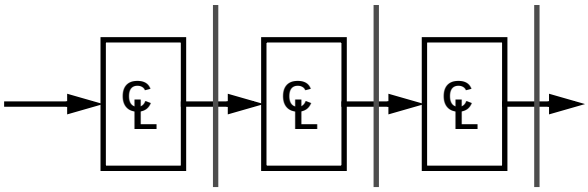


- I would not need clocks
  - The ‘state’ is stored in the gates and the wires.
- Signals stay naturally correlated in time
  - (wave pipelining)

- Impossible to do in practice, so …

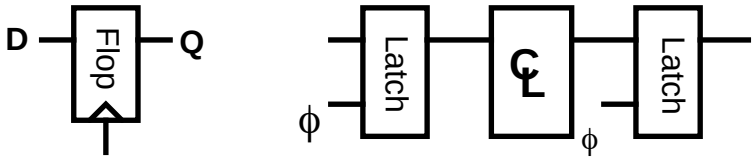
Clock’s Function:

Keep values in a system correlated in time



Keep signals from racing ahead of others

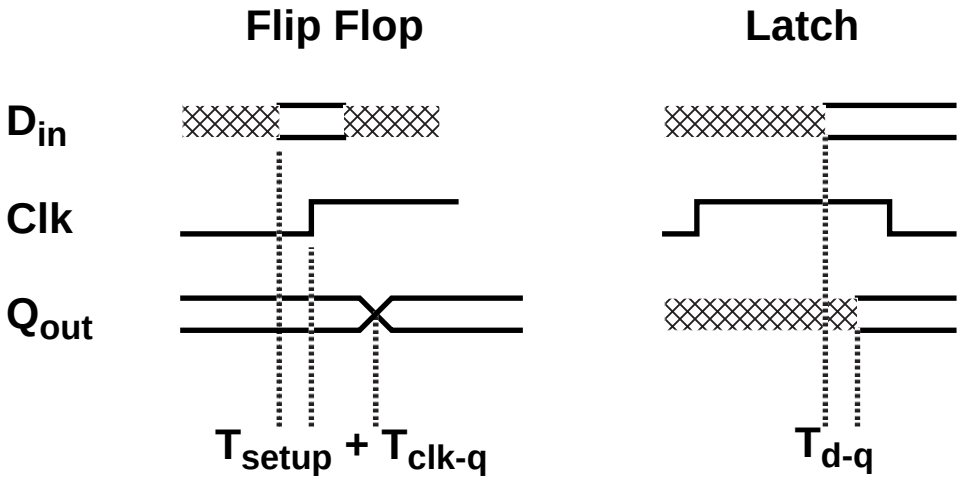
- Slow down signals that arrive too fast



- A flop is almost always built from two latches back to back

Clock Overhead

Unfortunately, clocks delay slow paths too



Clock Skew

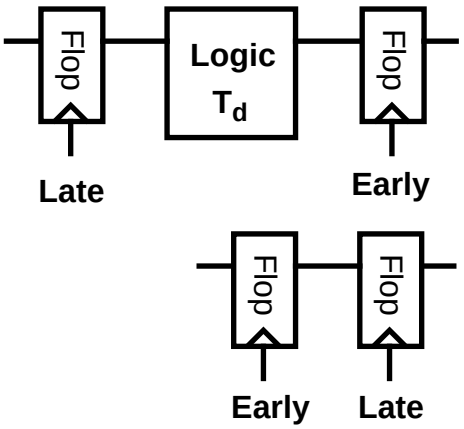
Not all clocks arrive at the same time

- Two problems:
  - Adds more overhead:[1]

    $T_{cyc} = T_d + T_{setup} + T_{clk-q} + T_{skew}$

  - Can get the wrong answer:

    $T_{skew} < T_{clk-q} - T_{hold}$



Low overhead -> Fast latches, low clock skew

1. As one of the reading points out, it is hard to break the flop delay into a setup and clk-q delay, since the clk-q delay can increase when the setup time is small. We will ignore this type of issue until we talk about flops in the next lecture

Microprocessors

To understand why clock design is getting harder, all we need to do is look at a couple of processor designs over the past 20 years.

- Take a look at a few different processors
  - 8086 (1978)
  - R2000(1986)
  - 21064(1992)
  - 21164(1995)
  - 21264(1998)
- Getting Larger
  - 30mm^2, 80mm^2, 220mm^2, 300mm^2, 300mm^2
- Getting Faster
  - 5 MHz, 16MHz, 150MHz, 300MHz, 600MHz
  -

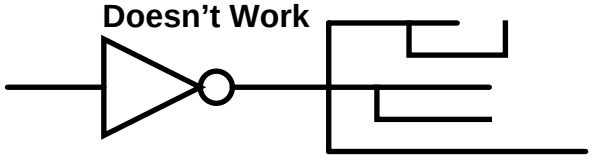
Processor Trends

Performance:

- Part of speed increase is technology
  - Technology from 3μ nMOS to 0.25μ CMOS
  - CMOS FO4 gate delay is roughly 0.5ns/μ L
- Part is better circuit design
  - 200, 50, 20-25, 20 FO4 inv delays/cycle

Bottom Line:

- Cycle time getting shorter, even in # of gates
  - One FO4 delay is a larger % of cycle time
- Die size is growing
  - More capacitance on clocks (nF)
  - More resistance in clock lines



Clock Overhead

Really two issues:

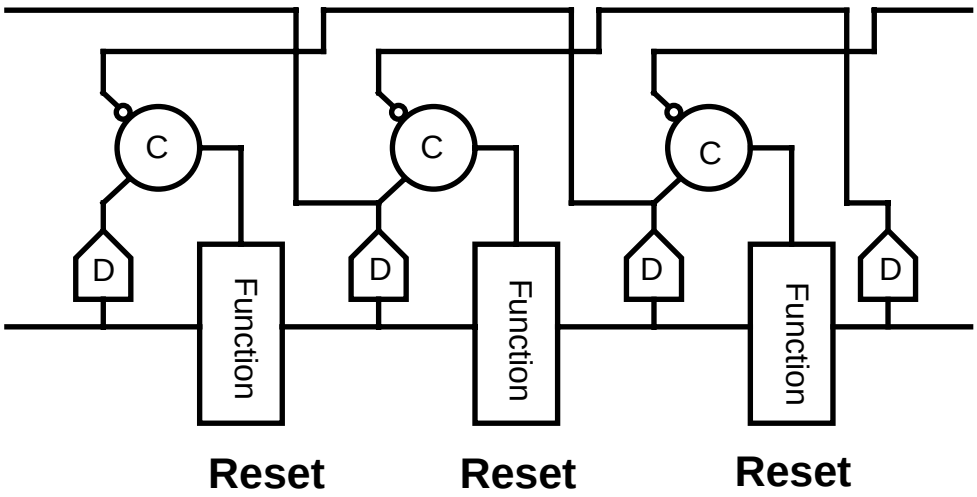
- Latch / flop delay
  - All latches I know of have a delay of ~1.5 FO4
  - Two latches / cycle is > 15% of cycle
- Clock skew
  - Keeping clock skew in ps constant is hard
  - But cycle times are falling
  - So engineering needed on the clock is growing.

If generating the clock is hard and getting harder, why do it

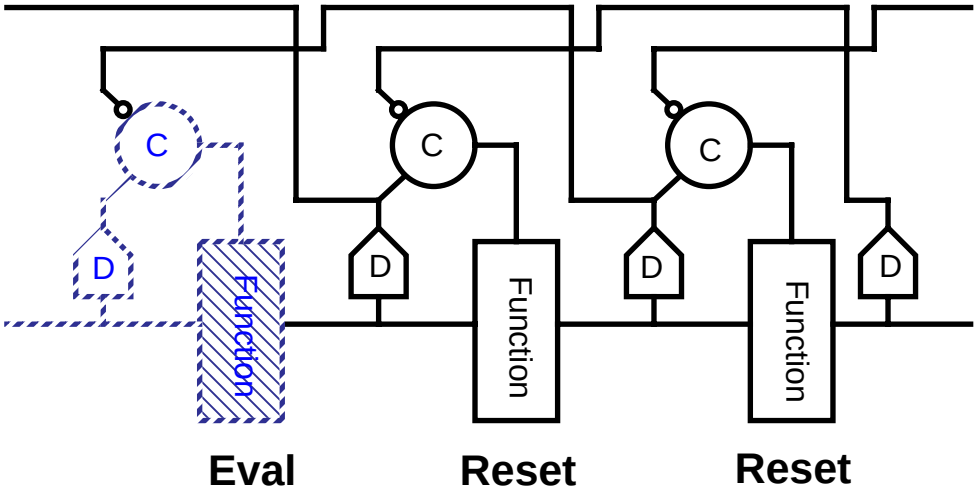
- Radical approach would be to eliminate it all together
  - Use local information instead
  - Called self-timed design

    Uses information bundled with the signals for sequencing

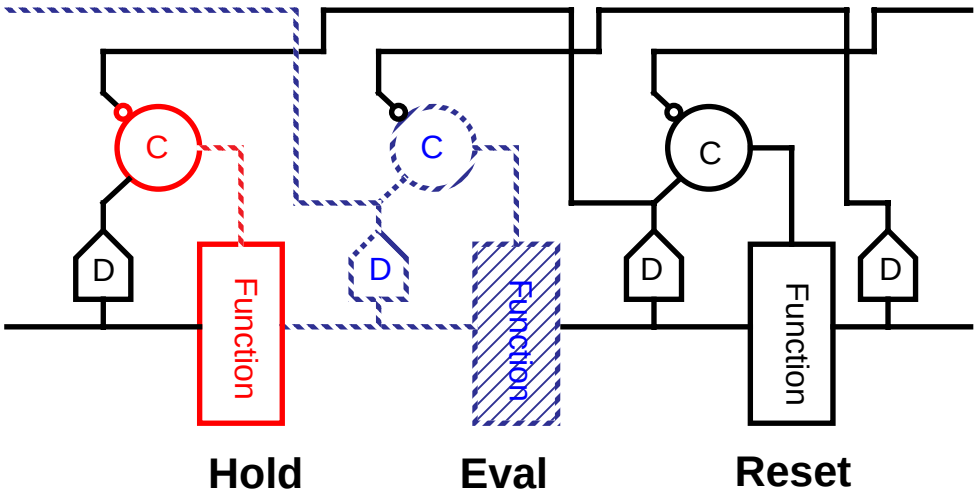
Simple Self-Timed Pipeline



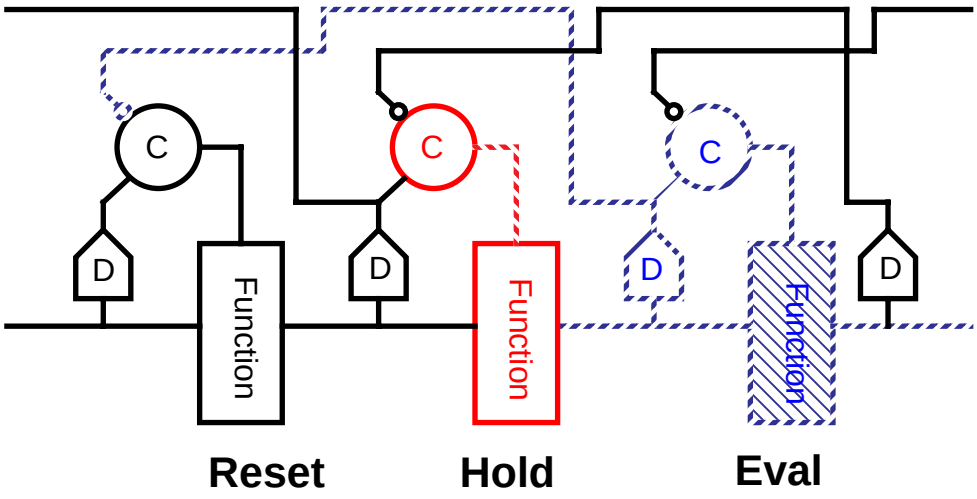
Simple Self-Timed Pipeline



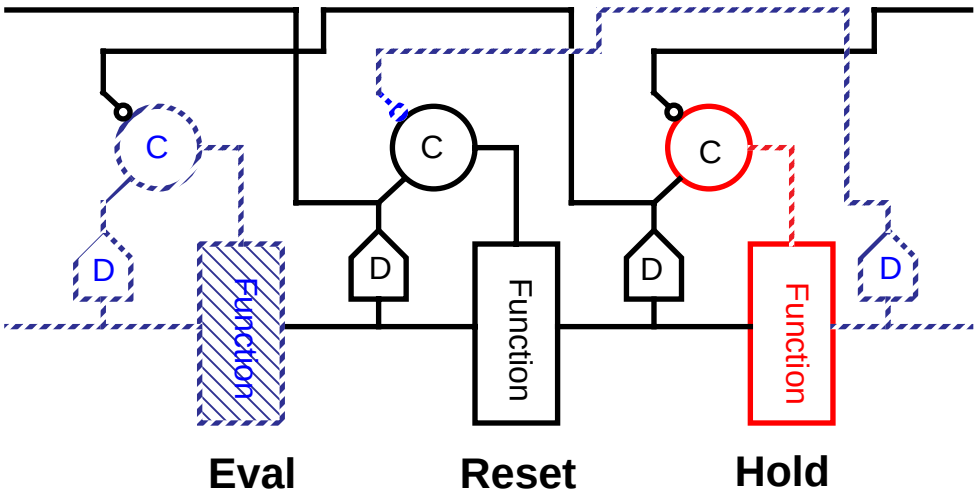
Simple Self-Timed Pipeline



Simple Self-Timed Pipeline



Simple Self-Timed Pipeline



Self-Timed Sequencing

Advantages:

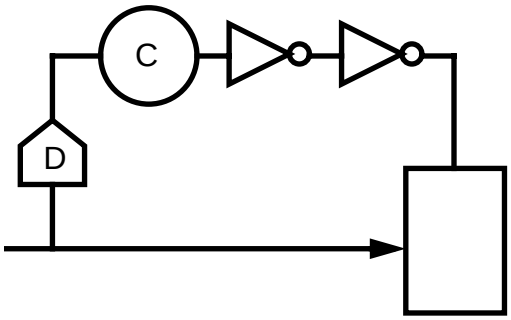
- No global clock
- No clock skew
- No worst-case operation constraint
- Speed depends on operating conditions
- Cycle not limited by worst possible case

Disadvantage:

- Lots of overhead!

Cause of the overhead:

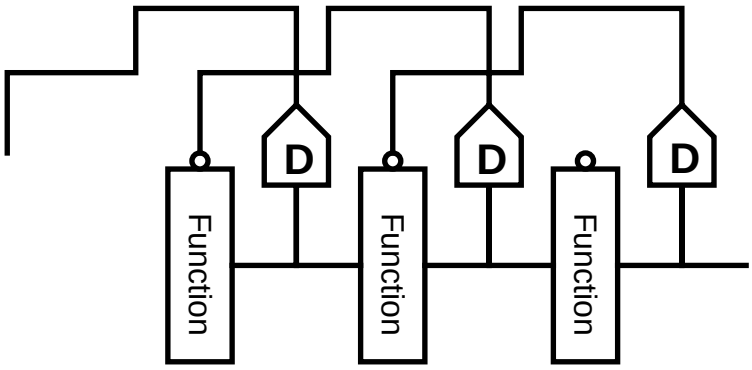
- Eliminated clock, not need for sequencing
  - Control generated from local signals
  - And that takes time
- Note: This is added delay not skew



Reducing Overhead

Make SMALL assumption about timing

- Remove D from forward path
- Data starts evaluation



More Problems

In these kinds of systems there are three constraints that might be a problem

- Forward constraints (data movement)
  - This is the topic we were looking at, how fast can a data token flow forward
- Backward constraints (bubble movement)
  - This is the constraint on how fast the bubbles flow backward, or how fast does the unit reset after having data. If this is slow, you can have many bubbles for each data so this is not a constraint, but it lowers the pipeline rate.
- Loop constraints (min cycle time)
  - How fast can a function unit reset to be able to start another evaluation

These constraints have large delays in them:

- t_D - completion detection tree, t_C - buffer delay to drive datapath control
- $t_D + t_C$ can be 1/2 cycle

Have fork, join issues too

Self-Timed vs. Clocked Systems

Went to self-timing to get rid of skew

- Got rid of skew and worst-case limits
- But got control overhead too
  - Control overhead is larger than clock skew
- Can be hidden (in theory) but complicated
  - Little tool support

So need to choose badness

- Most designers choose clocks

Goal

Be paranoid:

Make clock skew as small as possible

AND

Make your circuit insensitive to skew

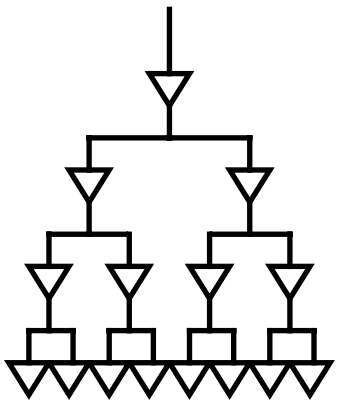
Clock Distribution

Need to reduce the skew on distributing the clock

- This requires us to reduce the wire delay, and the buffer delay
  - But we can't reduce the delay to the required levels (100ps) so
- Make the effective delay small, by balancing the delays of all the paths
  - Change a total delay problem to a matching problem
  - Make ΔT much smaller than T_{drive}

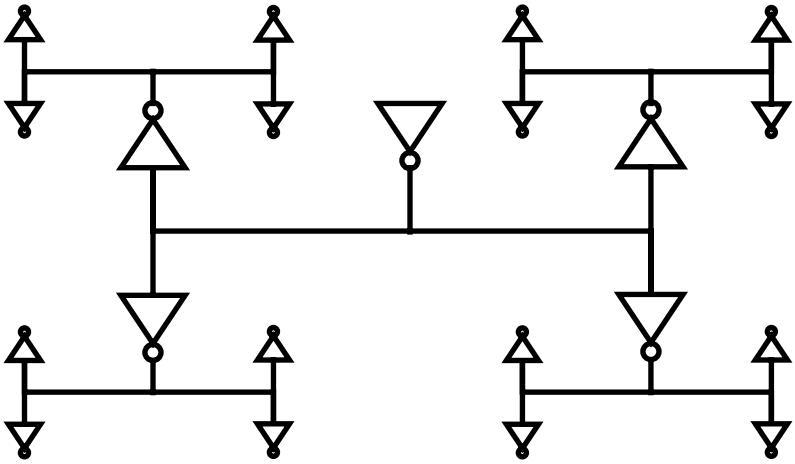
Use a clock trees

- Match the delay on different branches of tree
  - If the buffer delay matches
  - If the wire delay matches
  - Skew will be zero
- Obvious question:
  - How well can you match delays?



H Trees

Space filling pattern that matches wire delays



Lots of papers on these things, but not real issue

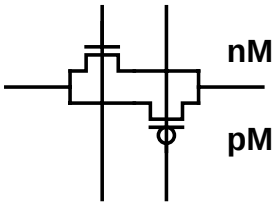
Real Matching Issues

There are function blocks on chip, that mess up your nice abstract H-tree

- Wiring and buffers will need to fit
- Buffers eventually need to drive latches
  - Load of a latch is data dependent, since the source voltage can change

    Gate loading depends on whether channel is formed

    Variation depends on technology, but is around 2:1 in capacitance



nMOS cap larger when 0

pMOS cap larger when 1

- Chip environments are not perfect
  - IR drops on the power supply lines
  - Temperature gradients across the chip
- Fabrication is not perfect
  - proximity effects / process tilt

Wire Load Matching

Each wire has a different mix of components

- Not only gate-cap vs. wire cap
- Also % M1 - M2, M1 - M1, M1 fringe, etc.
- Need to find the worst-case skew

Process corners don't help

- Don't vary wires relative each other, don't account for data dependence

No real tools to help

- Problem for simulating matching of any kind, you need to simulate the worse-case for the matching, and this might not be either the slow or fast case.

Buffer Matching

- Buffer delay depend on Vdd, can vary over a chip due to IR drops (over 10%)
- Fabrication matching -- process tilt and proximity effects

Process Tilt and Proximity Effects

Chips are large (2cm on a side), and transistors features are small

- Inverters on different sides on chip will be different
  - Difference is not in corner files, since corners make all transistors the same

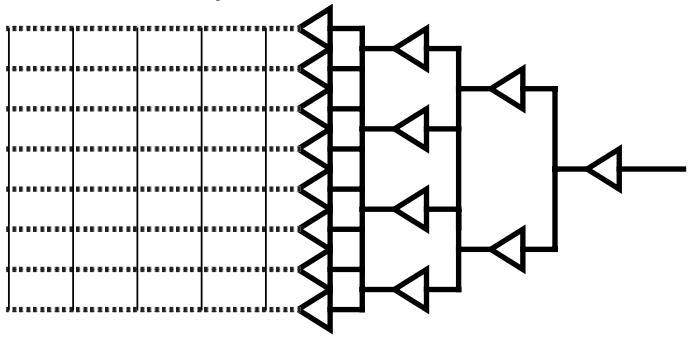
    This data is not usually given to designers

    It is essential for simulating clock skew
- Proximity Issues
  - Poly width sets channel length / speed of circuit
  - Current gate lengths are 0.25μ, Poly control must be a few 0.01μ
  - Local poly environment affects etching rate, so it affects channel width
  - Matched inverters, need matched layout, and matched environments

    (region around buffer needs to be same -- add dummy buffers)

Single Clock Distribution - 21064

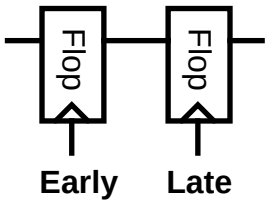
- Thick metal layer for clocks, M3 - 2μ thick
- Large clock buffer (entire vertical height of the chip)
  - Use a tree to balance the delay in this direction



- Shorted together all the local clock wires
  - Main difference with a conventional tree; reduces the effects of mismatches
  - Especially effective for reducing local skew
- More recent processors have more clock buffers to keep skew small

Local Skew

Important for race-through



- Can get the wrong answer:
  - $T_{skew} < T_{clk\text{-}q} - T_{hold}$

Only occurs if delay is less than skew

- Delay is small only when elements are close
  - So only occurs when local skew is large
  - Shorting clocks together can reduce local skew

    But also limits your design

    Only have one clock and it is on all the time

    Not the lowest power solution

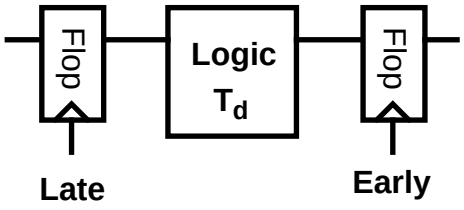
    Can avoid this problem by not having short paths
  - Need to have tool to check (and fix) min-delay

    (Easy, make all your flops with the ability to have a long clk-q delay)

Global Skew

Still have long path problem

- Skew adds more overhead:
  - $T_{cyc} = T_d + T_{setup} + T_{clk\text{-}q} + T_{skew}$



So don't use flops!

- The situation with latches is a little different

Clocking Design

- Trade off between overhead / robustness / complexity

  Constraints on the logic

  vs.

  Constraints on the clocks

- For performance, you need to worry about
  - Overhead of the sequencing

    Delay through the latches/flops

    Wasted time from clock-skew
- Look at a number of different clocking methods:

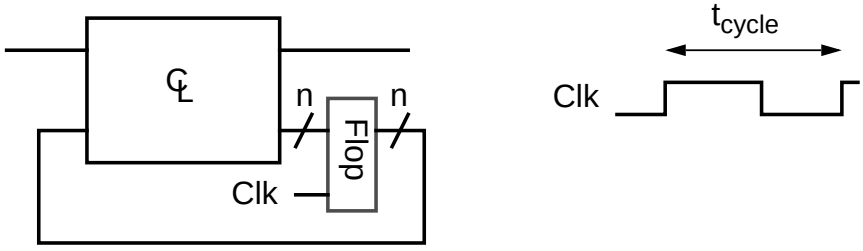
  Edge triggered clocking

  Pulse mode clocking

  Two phase clocking (might only have one clock)

Edge Triggered Flop Design

- Most popular design style (comes from old TTL designs)
- Used in many ASIC designs (Gate Arrays and standard Cells)
- Using a single clock, breaks every cycle with a flip-flop



- Timing Constraints

$$t_{dmax} < t_{cycle} - t_{setup} - t_{clk\text{-}q} - t_{skew}$$

$$t_{dmin} > t_{skew} + t_{hold} - t_{clk\text{-}q}$$

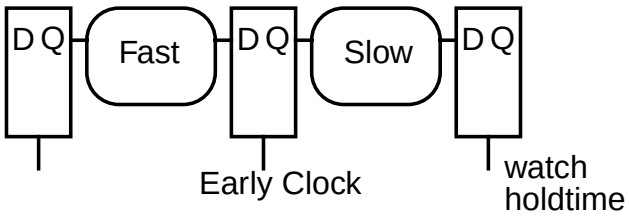
- If skew is large enough, you have two sided timing constraints

Flop Design

Flops introduce hard timing boundaries in to the circuit

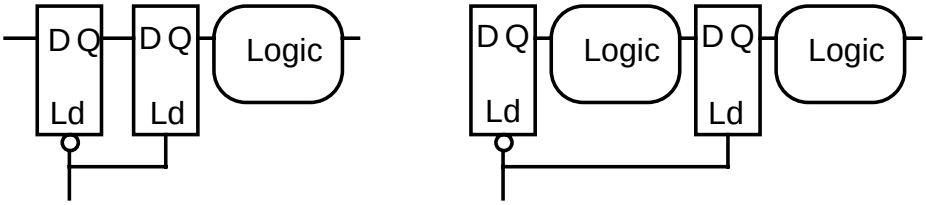
- Data must setup before the clock edge
- Output does not change until after the clock edge
  - Any uncertainty in clock, or data is wasted
  - Need to know precisely when the data will arrive

If some section of logic will be done early, to use that extra time, you need to move the clock to the flop early, so the next cycle has more time (and you had better check the hold time of the following flop)



Latch Based Design

- Break flop into its two latches, and place logic between the latches.



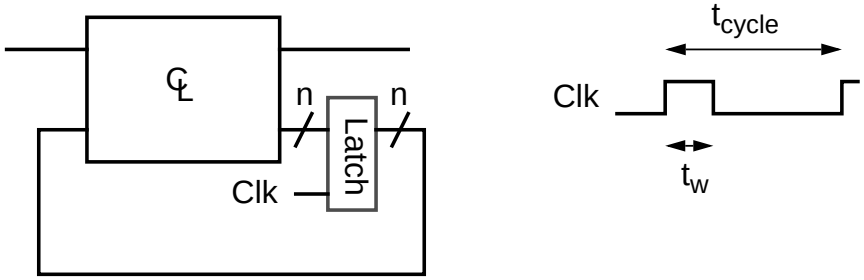
- There are no hard boundaries in latches
  - Pass data when clock is high
- Latching event is the load to hold transition
  - If data arrives early it is passed through
  - Can borrow time naturally, and
  - Is insensitive to clock skew, for critical paths, data sets the timing (well generally)

Pulse Mode Clocking

Two requirements:

- All loops of logic are broken by a single latch
- The clock is a narrow pulse

  It must be shorter than the shortest path through the logic



  Clock is usually generated inside the latch

- Timing Requirements

$$t_{dmax} < t_{cycle} - t_{d\text{-}q} - t_{skew}$$

$$t_{dmin} > t_w - t_{d\text{-}q} + t_{skew}$$

Pulse Mode Clocking

- Used in the original Cray computers (ECL machines)
- Advantage is it has a very small clocking overhead
  - One latch delay added to cycle
- Leads to double sided timing constraints
  - If logic is too slow OR too fast, the system will fail
  - But there is some flow time when the latch is enabled (softer edge)
- Pulse width is critical
  - Hard to maintain narrow pulses through inverter chains
- People are starting to use this type of clocking for MOS circuits
  - Pulse generation is done in each latch.
  - Clock distributed is 50% duty cycle
  - CAD tools check min delay
  - Called a glitch flop, but it is not a flop, it is a glitch latch!

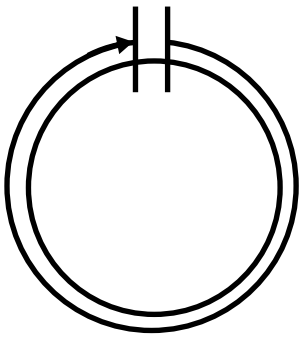
Thinking About Timing

Image your arranging your netlist on a sheet where all the flops at the top

- The gates distance from the top indicates the settling time of its output
- Gates at the end of long paths would be at the bottom of the sheet
- Some of the outputs are the inputs to the flops, so we roll the sheet
- Forms a cylinder, where the circumference is equal to the cycle time

With flops, the input has to settle before its clock rises, and the output can't change until its clock falls so

- To guarantee operation, need to waste skew time
- Hard edge is the problem



Latches

Are hard to analyze, since the timing of the output is not completely set by clock

- Output is valid clk-q delay after clock rises

  If input was valid when clock was low

- Output is valid a d-q delay after input

  If input becomes valid when clock is high

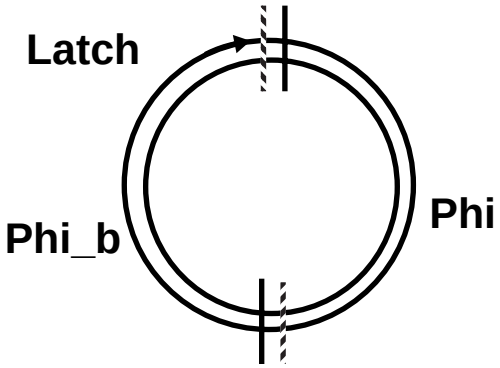
Skew changes the relative timing of clock and data

- Since latches are soft barriers it does not change the output arrival time!
- Latch based system can tolerate skew

  $=T_{cyc} - T_{max}$

  T_{cyc} = cycle time

  T_{max} = Max delay between latches



Summary

Clocks have skew

- Today skew is caused by mismatches of balanced paths
  - Simulating matching is VERY HARD since need data that is not available
  - Need your our simulation environment since assuming perfect matching gives you get best-case skew
- I believe that keeping skew under 200ps is hard
  - Can do better than that over smaller regions (for fixed functionality, skew should roughly scale with inverter speed)
- Need to have circuits that deal with skew
  - Need to prevent race-through by padding short paths (and min. local skew)
  - Prevent cycle-time impact by using latch based design techniques

    Use skew tolerant domino (but that is another lecture)

If you are careful clocks will work fine for chips running at GHz rates.