VLSI Clock Distribution

The clock distribution network (or clock tree) is the metal and buffer network that distributes clock to all clocked elements.

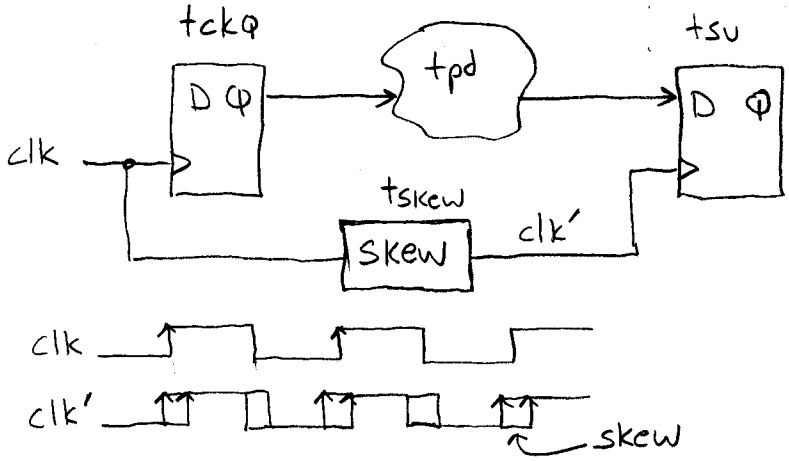
Clock trees are usually built by clock tree synthesis tools.

The main job of clock tree synthesis is to vary routing paths and placement of clocked cells and clock buffers to meet maximum skew specifications.

Clock skew is the difference in arrival time of a clock edge at any two flip-flop clock inputs.

Clock skew adds directly to clock cycle times, reducing the system clock speed.

- minimum cycle time = $t_{ckq} + t_{pd} + t_{su} + t_{skew}$



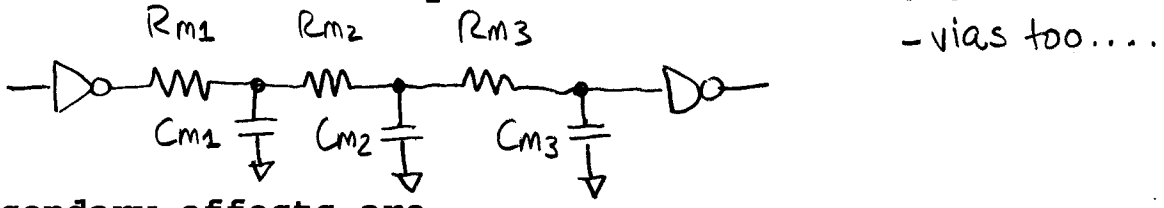
Skew of high performance chips should be less than about 5% of the clock cycle time. (<100ps on leading edge chips)

Variations in clock signal arrival times cause FF malfunctions.

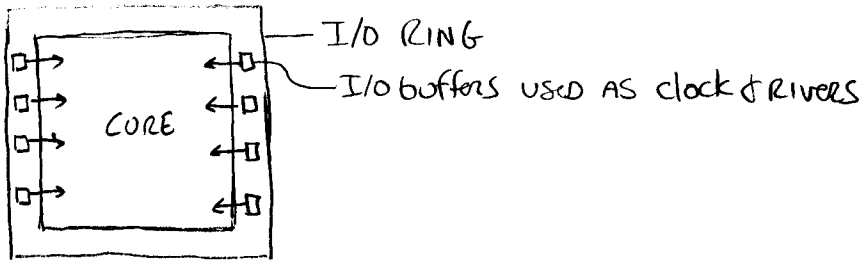
- t_{su} violations
- t_{hold} violations
- metastability
- failure at lower speeds than expected

There are actually many factors to optimize the clock tree for:
- -skew
- -latency
- -jitter
- -power dissipation
- -signal integrity
- -reliability

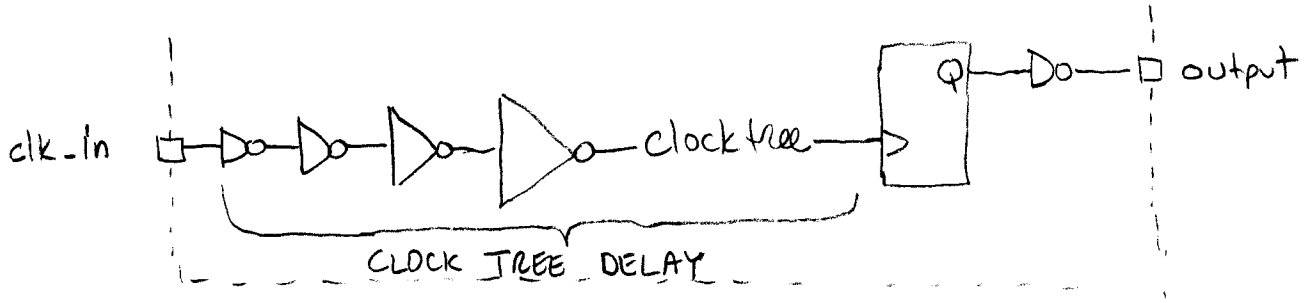
Skew is mostly a function of two parameters
- -loading mismatch of logic being clocked
  - -different FFs/Latches present different loads
  - -may necessitate using "dummy terminator cells"
- -RC delay of the clock-line interconnect
  - -different metal layers have different L,R,C values



- -secondary effects are
  - -clock buffer speed differences
  - -temperature, voltage, and process variations
    - -at lower voltages, v_{th} will become a problem
    - -clock drivers on die perimeter reduce V_{dd} variation
    - -distributing clock drivers reduces "hot spots"
  - -data dependencies
    - -data patterns cause clock skew because of Vdd variations
    - -packages with better Vdd distribution have less skew

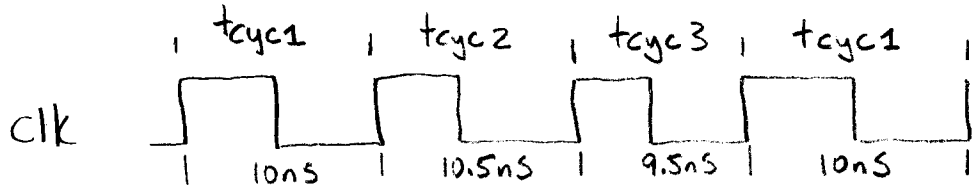


Latency
- -due mostly to powering up the clock tree.
- -primarily effects the chip-to-chip, clock-to-out delay
- -can be essentially eliminated with on-chip PLL



Jitter

- -the cycle time variation of consecutive clock periods
- -typically caused by data-dependent loads, coupling events
- -causes cycle time compression; lowers operating frequency



Power Consumption

- -clock node consumes more power than any other node on a chip
- -clock tree should use no power beyond the clock load.
- -for an ungated clock $P = C_1 Vdd^2 f$
  - -no activity factor since clock changes twice every cycle
- -clock tree power is also composed of:
  - -crossover or direct-path current. (can be 10%-20% total I)
    - -note: faster edge -> less power dissipation
  - -leakage current a factor at <.1um and high temp
  - -on a uP, clock tree can dissipate up to %40 of total power

Signal integrity

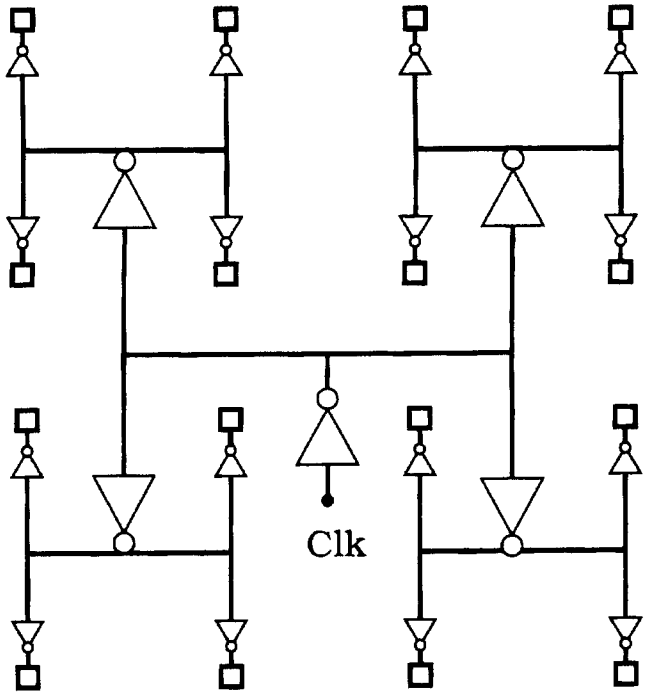
- -the clock network is widely distributed
- -edge rate is high
- -care must be taken to not route parallel to clock

Reliability

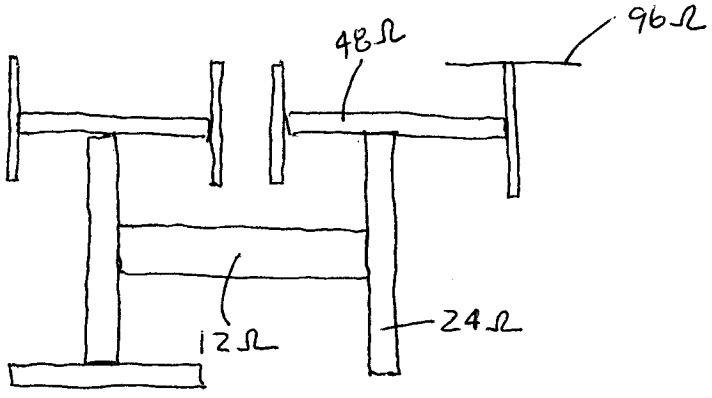
- -electomigration:
- -movement of metal material caused by net flow of electrons.
- -can eventually cause opens
- -a hazard for interconnect that carry large currents
- -worse for unidirectional current

Implementation of the clock tree
- -The problem is logically simple, physically difficult
- -For low skew, fast edges and low power.....
  clock drivers should be located close to their loads.

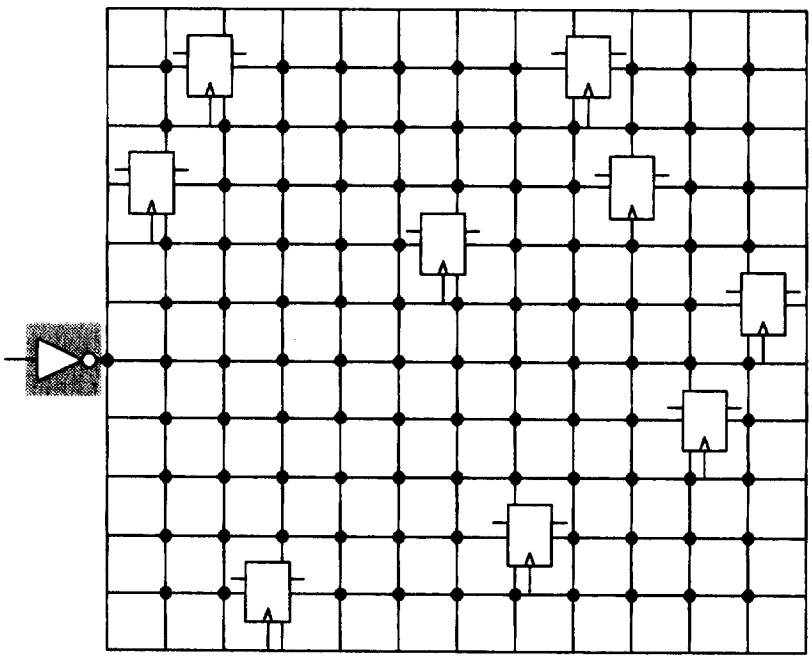
Typical Structures used
- -H-tree:
  - -usually a custom implementation
  - -very low skew
  - -requires big driver, thus lots of power
  - -lots or routing resource
  - -special metal widths needed, 100um M1

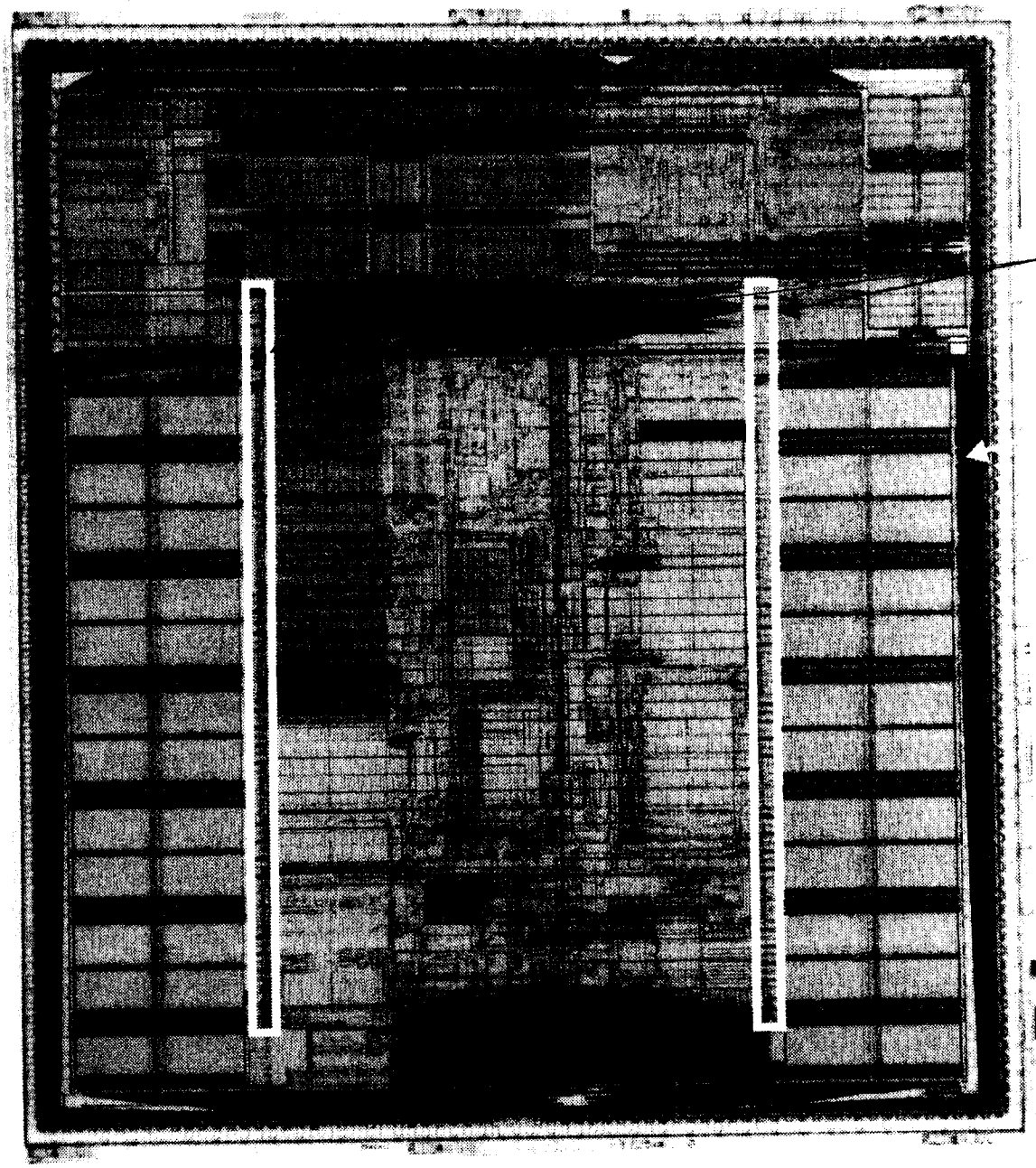


must match metal widths at higher freqs (multi Ghz)

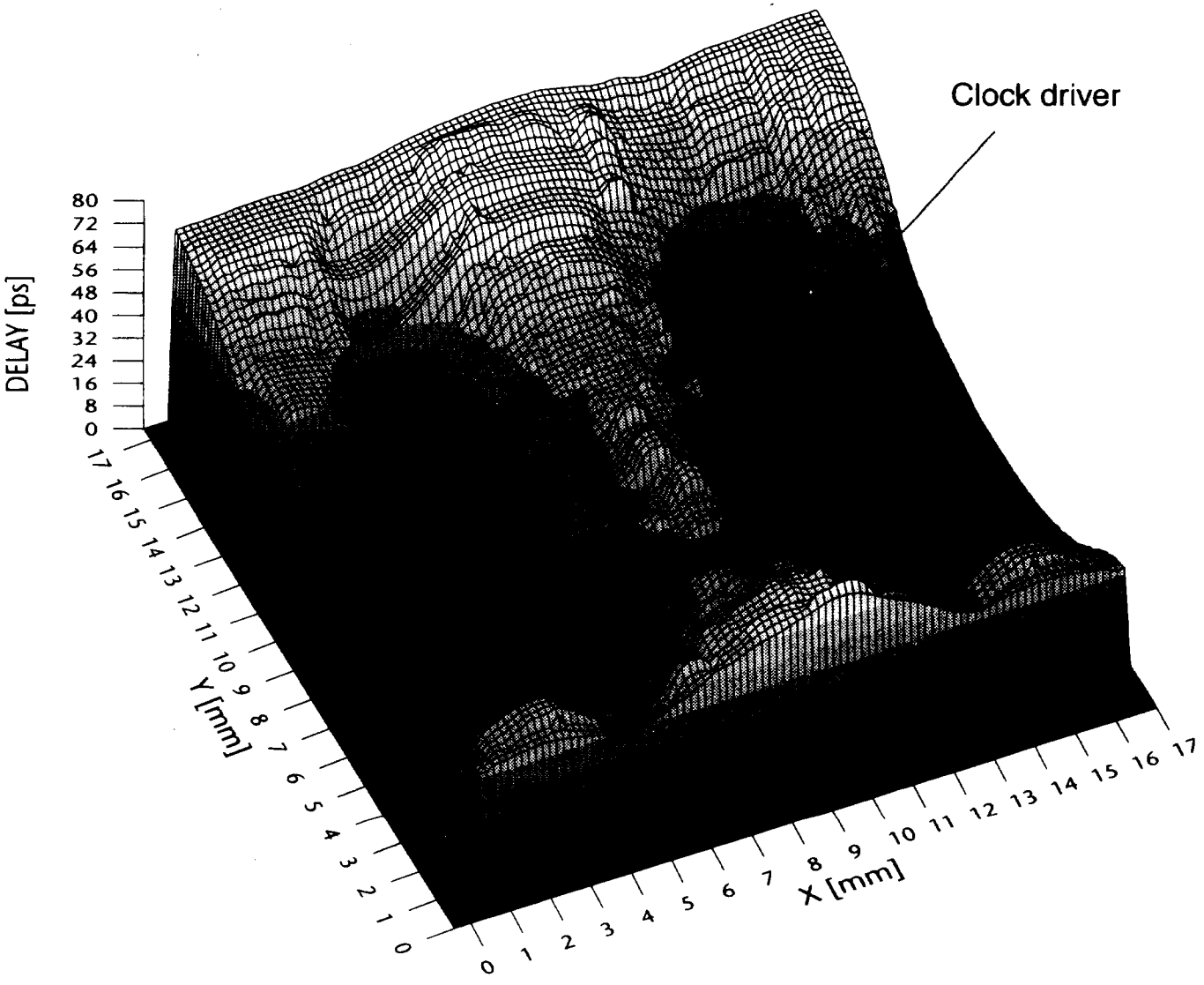


-Clock Grid:
  -usually a custom implementation, fairly simple to build
  -insensitive to load changes
  -adapts to late design changes
  -driver variations swamped, final drivers are in parallel
  -area inefficient (3% of upper metal)
  -power-hungry (60% of power is spent driving metal lines)
  -Alpha 21164 uses grid clock

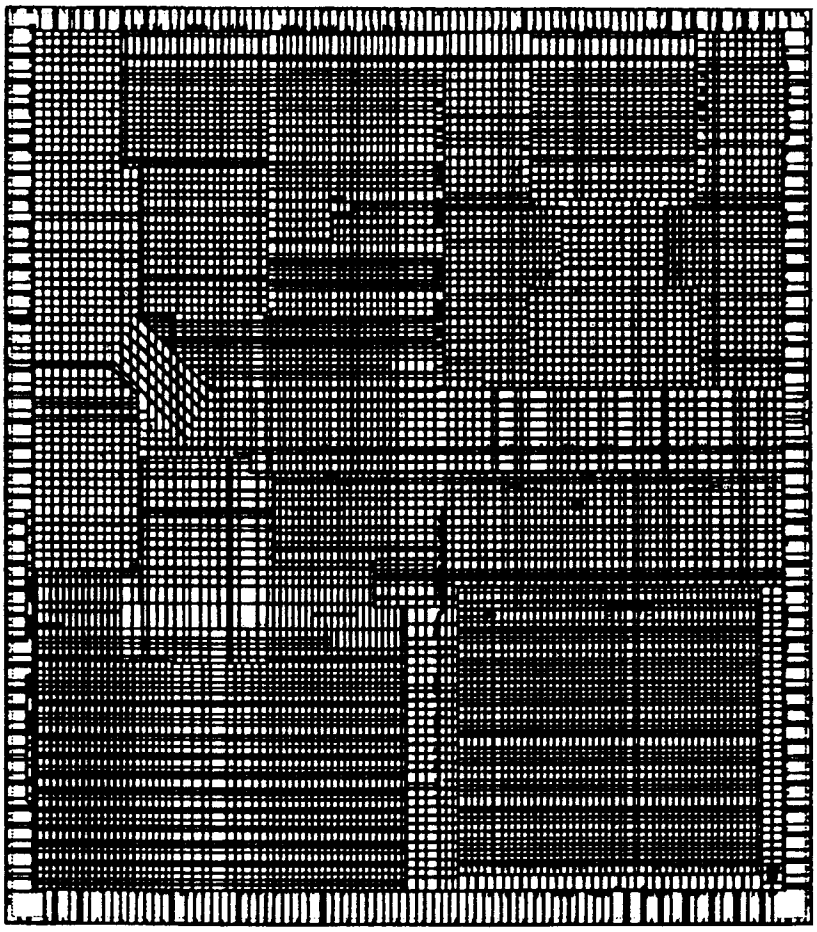




CLOCK DRIVERS FOR DEC ALPHA 21164
300 MHz, 0.55μm, 1995
20W dissipated by clock drivers
Equivalent width of final drivers is 58cm! (Not μm or mm)
Final drivers drive grid network

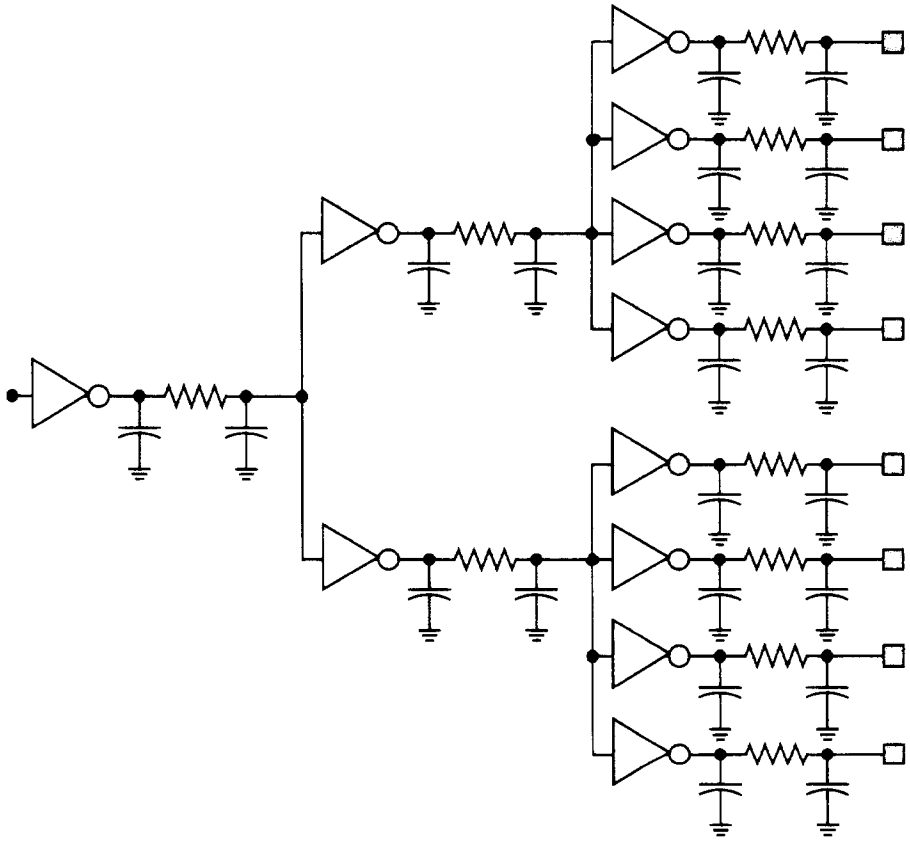


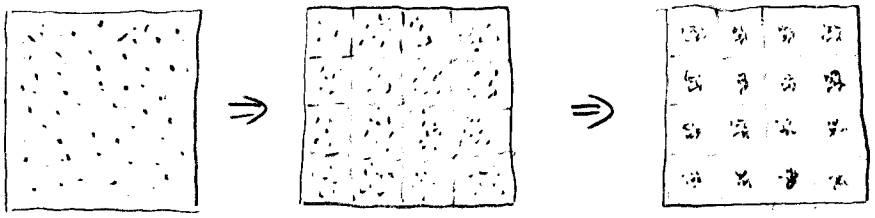
Skew Across die of Alpha 21164



clock grid distribution of 21164

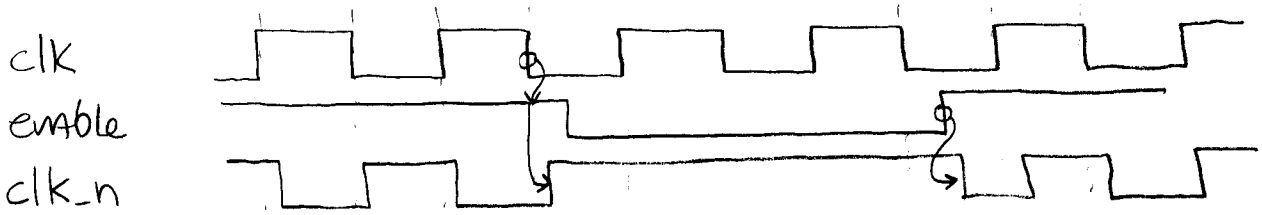
-Balanced Tree:
  -most common topology for high performance chips
  -usually heavy use of automated tools
    -place buffers first and cluster FFs around them
    -must match interconnect RC delay
  -takes 3 to 4 metal layers to implement
  -can use general purpose routing resource
  -uses small metal
    -low C load
    -small buffers
    -smaller I, thus lower crosstalk likelihood

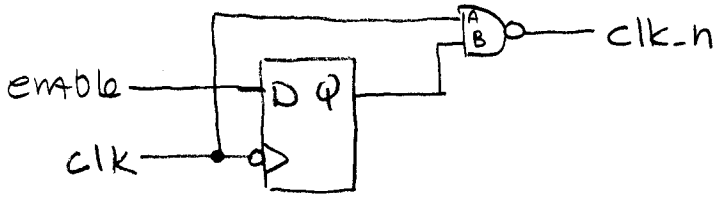


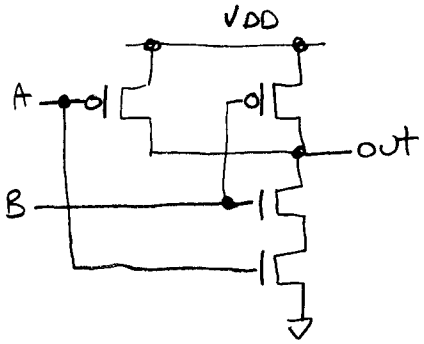


Clock Gating

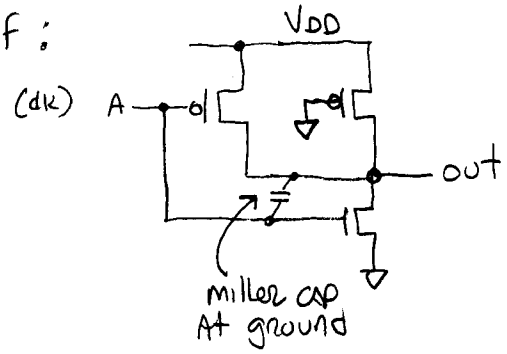
- **-can save bunches of power**
- **-not usually done at designer level but at block level**
- **-clock load is greater if clock is enabled!**
  - **-think about structure of NAND or NOR as to why**



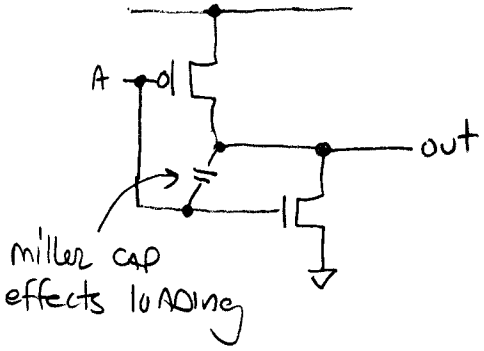




enable off:

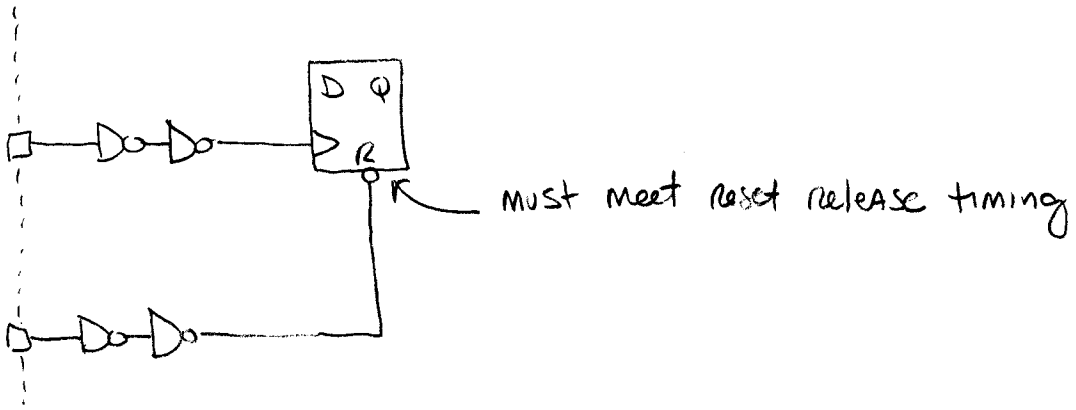


enable on:



Another important tree: The Reset tree

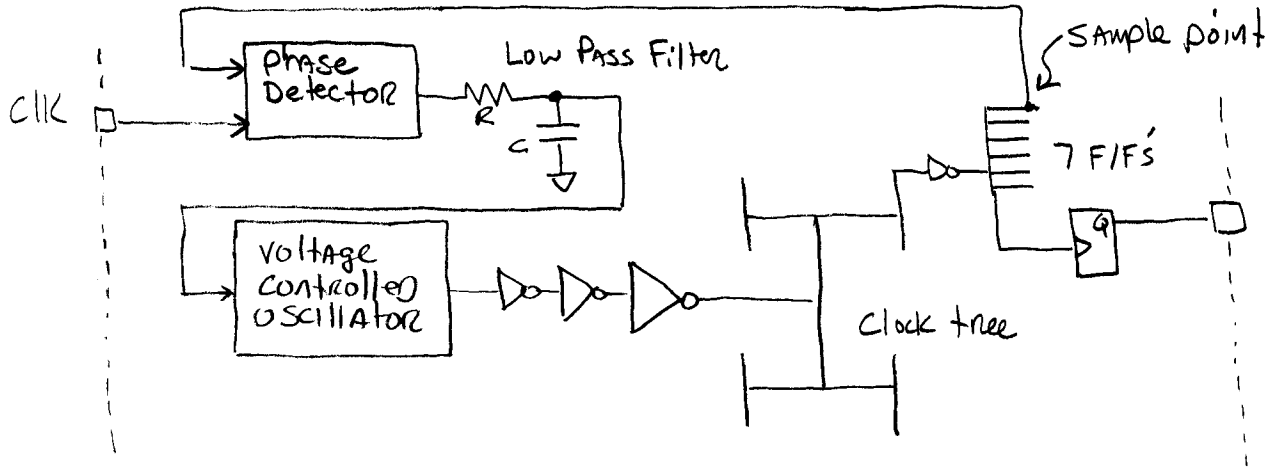
- -at release of reset, timing violations may occur
- -reset must meet release or setup time prior to clock edge
- -many of the same problems as with clock
- -reset is a large load like clock, but often 2x of clock load



PLLs - What are they for?

- **-skew reduction**
  - **-phase align different blocks**
- **-clock alignment**
  - **-two clocks with unknown phase relationship**
- **-latency reduction**
  - **-elimination of clock-in to clock-out delay**
- **-clock multiplication**
  - **-low speed external clock multiplied to high speed clock**

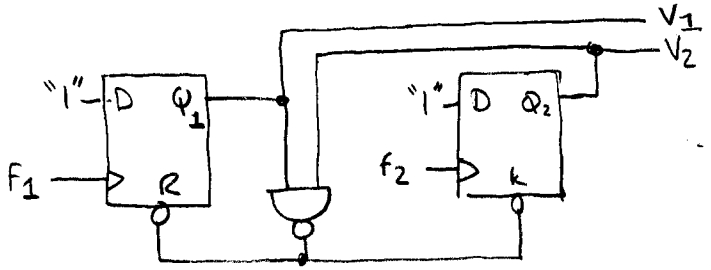
What does a PLL look like?

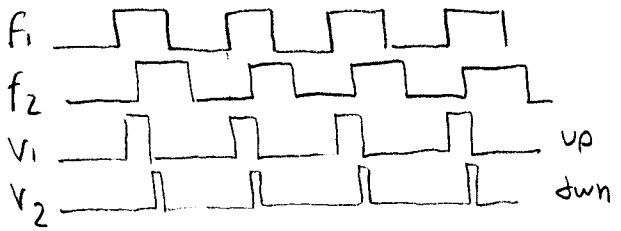


PLL eliminates internal clock tree delay
Internal clock identical in phase to external clock

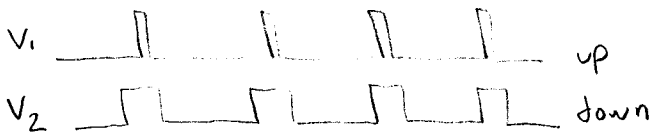
Key components

phase detector:



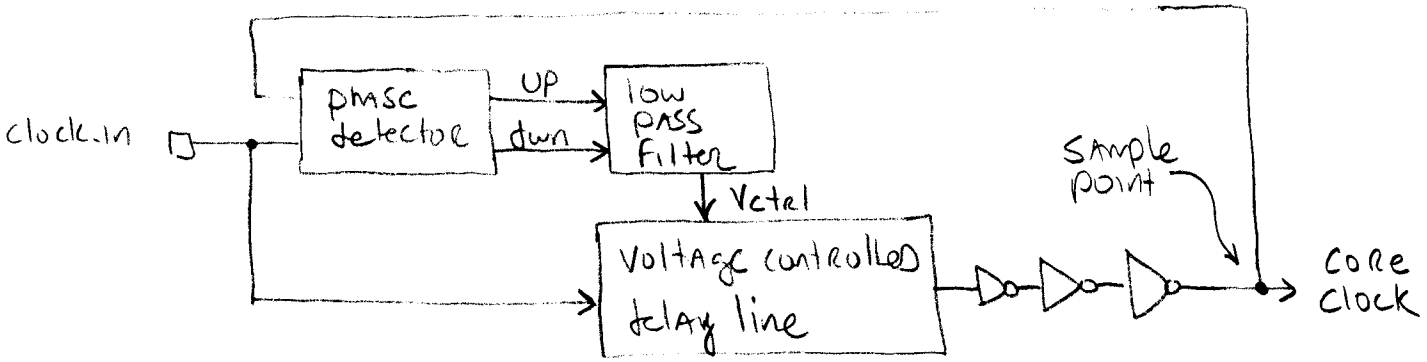


f_1 lagging f_2

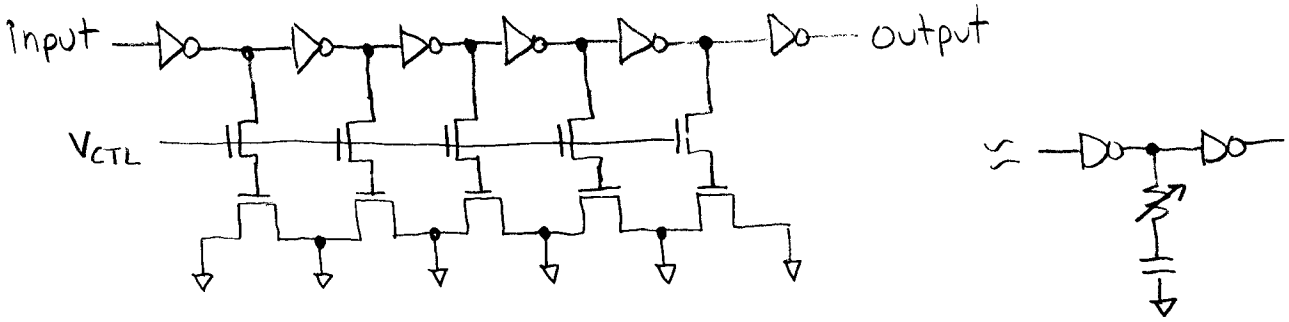


f_1 leading f_2

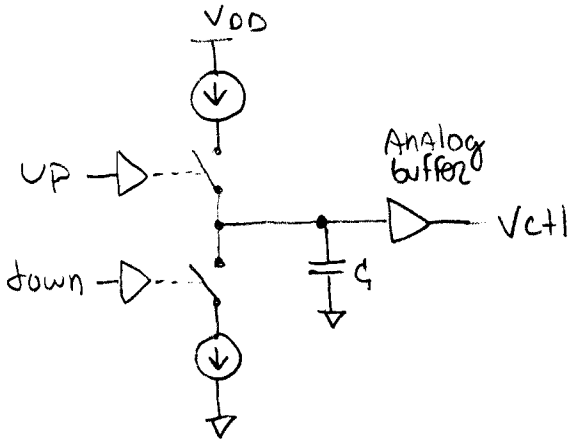
The delay locked loop:



voltage controlled delay line (easially converted to a VCO)



low pass filter (charge pump)



Is there a better way to design digital logic?

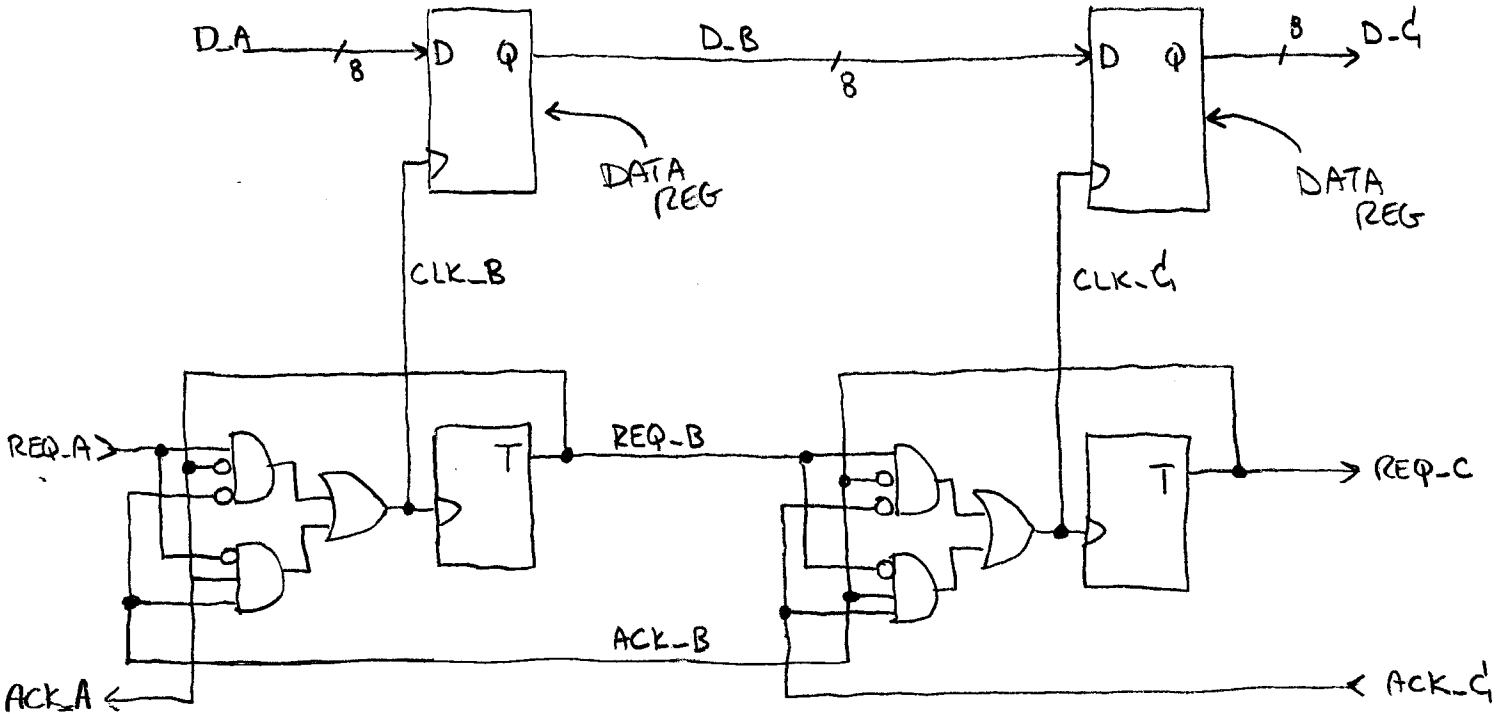
Is asynchronous design as evil as it seems?

Drawbacks of synchronous techniques
- -minimum cycle time limited by a single longest path
  - -what if the rest of the circuit could run faster?
- -designer must add worst case "padding"
  - -temperature, voltage, process (K_p K_t K_v)
  - -2x reduction in cycle time to "be safe"
- -clocks run all the time, but aren't always needed
  - -neglecting clock gating
- -need to distribute a very low skew signal to all FFs

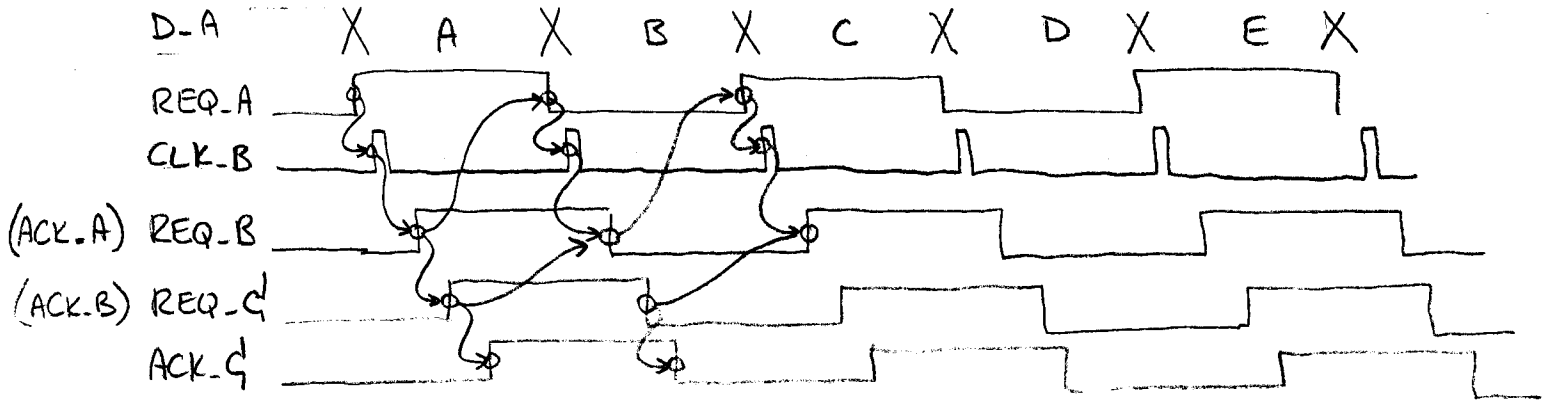
How can we do it a different way?
- -self-timed logic
- -two-cycle-interlocked self-timed circuits
- -uses both "clock" edges
- -runs "as fast as it can" but not faster
- -runs as fast as it can for local conditions
- -no clock except locally, never runs except when needed

Drawbacks
- -hard (tricky) to design
- -almost no CAD tools available
- -history of problems with asynchronous design

EXAMPLE 8-BIT SELF-TIMED PIPELINE



* IN 1 μm CMOS ≈ 200 MB/s
* 50MB/s IF Synchronous
* EXTERNAL STREAMING MODE 300 MB/S
* 72bit internal pipelines run AT 1.8GB/s



SELF-TIMED 2:1 DATA MULTIPLEXER TIMING & CONTROL

11/15/90
RCT