CPE 523: Digital System Design

Slides modified from: Mark Horowitz,
With contributions from Don Stark, and Subhasish Mitra

Notes on the Lecture Notes / Book

The lecture notes are the principle reference material that you will use in the class. But the notes are no where as complete as a book, so I recommend that you also get Weste/Harris.

If you are confused on a specific topic, you should check the book to see if it explains the material a little better. David Harris was my PhD student, and his book follows my approach to the class.

The only problem with the book is that my emphasis has moved to higher level design issues, while the book is focused more on lower level circuit issues. So in the 2nd half of the class we will cover topics that are not as well handled in the book.

Lecture Note Notation

To try to be a little more complete, I include material in my notes that I won't have time to cover in class. This material is comes in two different flavors:

+ in a slide title means that this slide give addition information about the previous slide, providing either an interesting aside, or some additional background that might be helpful to understand the material. It is for your benefit and is not part of the core material. This is also true for readings. A + reading is optional

- in a slide title give some examples of the previous material to make sure you really understand what is going on.

Reading

- I am handing out all the lecture notes BEFORE class
  - The readings for each lecture should be done before class
  - I will be honest about
    - What section you should read
    - And which sections you only need to skim
  - Readings will be from:
    - The book
    - Tutorials for how to use the tools
    - Papers that are interesting to read

- Like the lecture notes:

  + means the reading is option, it gives more depth/background

  - means the reading is option, it give more review on topic

Course Goals and Prerequisites

- Goals
  - Demystify the “magic” of digital design by explaining how these very complex systems are constructed by repetitive use of only a few components, and the tools and methods we use to create “perfect” systems of extreme complexity.
  - Build skills for designing/debugging/using complex systems
  - Provide you a feeling for when/why custom circuit solution might be attractive/necessary, and introduce you to some of the approaches and tools that you will need in that situation

- Prerequisites
  - Basic understanding of circuits and electronics
  - Basic understanding of logic design, Verilog or VHDL (133, 233, or 439)

Lecture 1

Digital Design: Insurmountable Opportunities:
Welcome Back My Friends,
To the Scaling that Never Ends.

Andrew Danowitz
Adapted from slides by:
Mark Horowitz

Overview

Reading

Gordon Moore, *"Cramming More Components Onto Integrated Circuits"*, Electronics, Volume 38, Number 8, April 19, 1965

ftp://download.intel.com/research/silicon/moorespaper.pdf

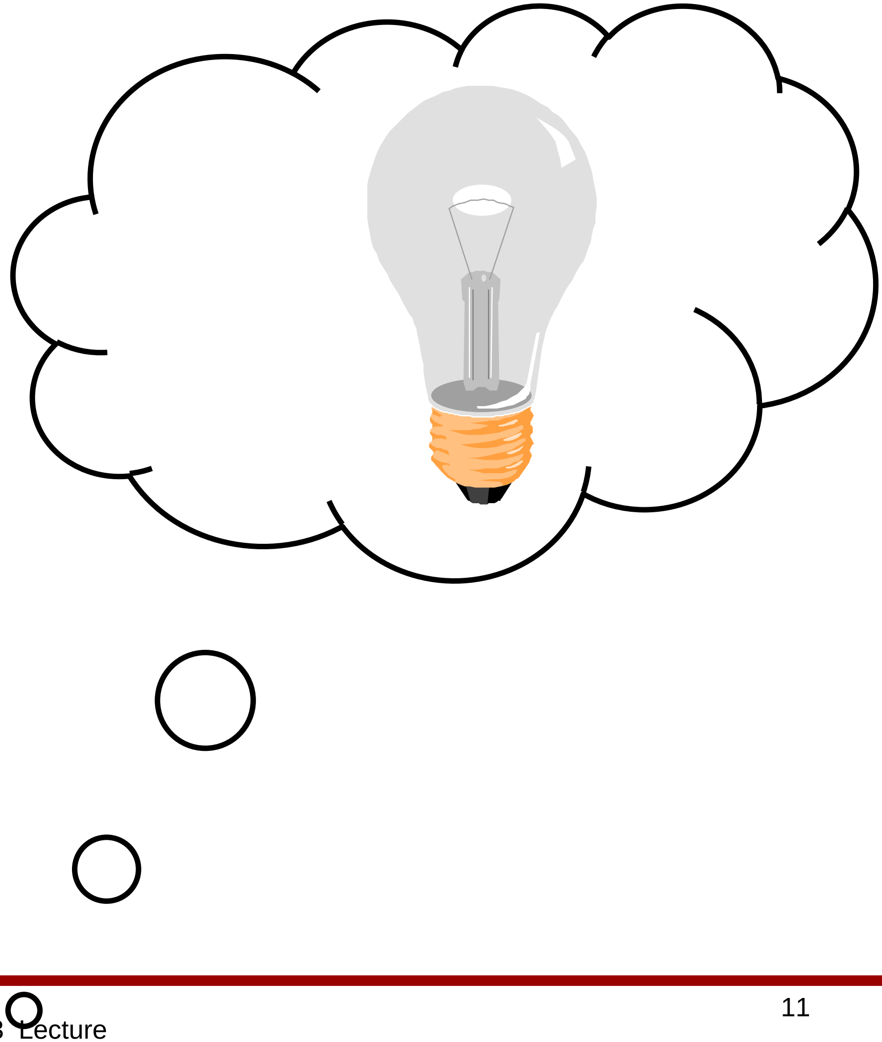
+ 2013 International Technology Roadmap for Semiconductors
http://www.itrs2.net/2013-itrs.html

Background

Rarely does a single technology takes over the world in quite the way that integrated circuits have done. From their invention around 50 years ago, IC are now found in almost everything we do, from cards to computers, and will be come even more prevalent in the future. This lecture will look a little at why this happened, and the tremendous opportunities it creates. Our current challenge is to figure out how to transform the capability of this technology into something of value for the consumer without becoming lost in complexity.

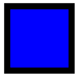
Big Picture

- Want to go from this:

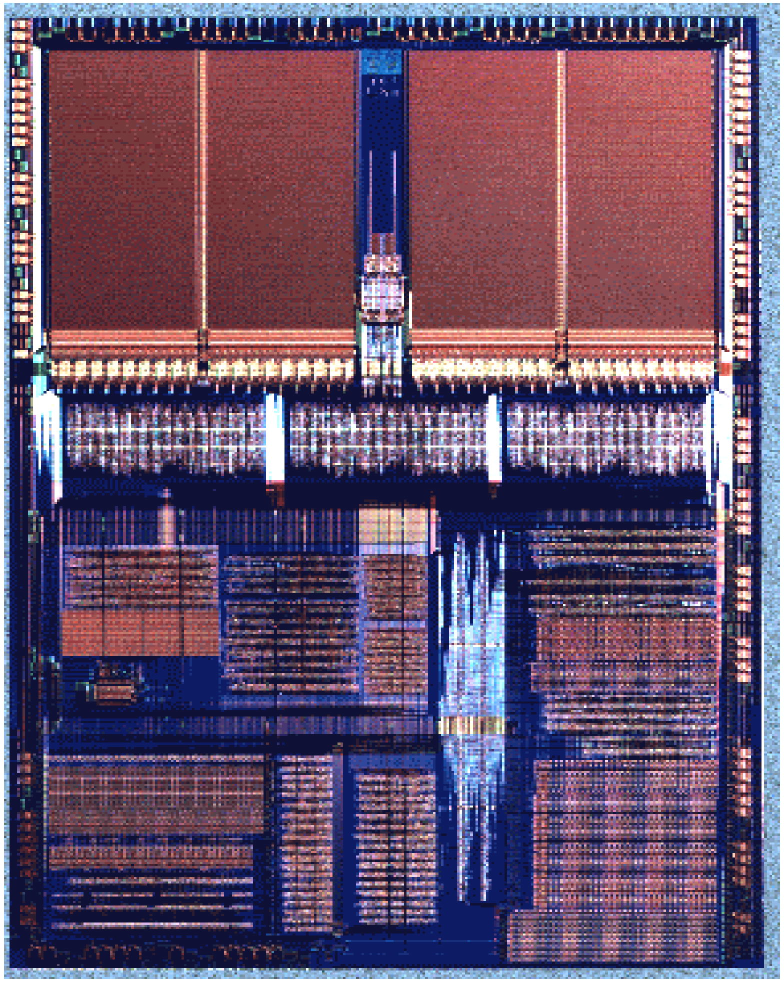


To

This:



Magnified

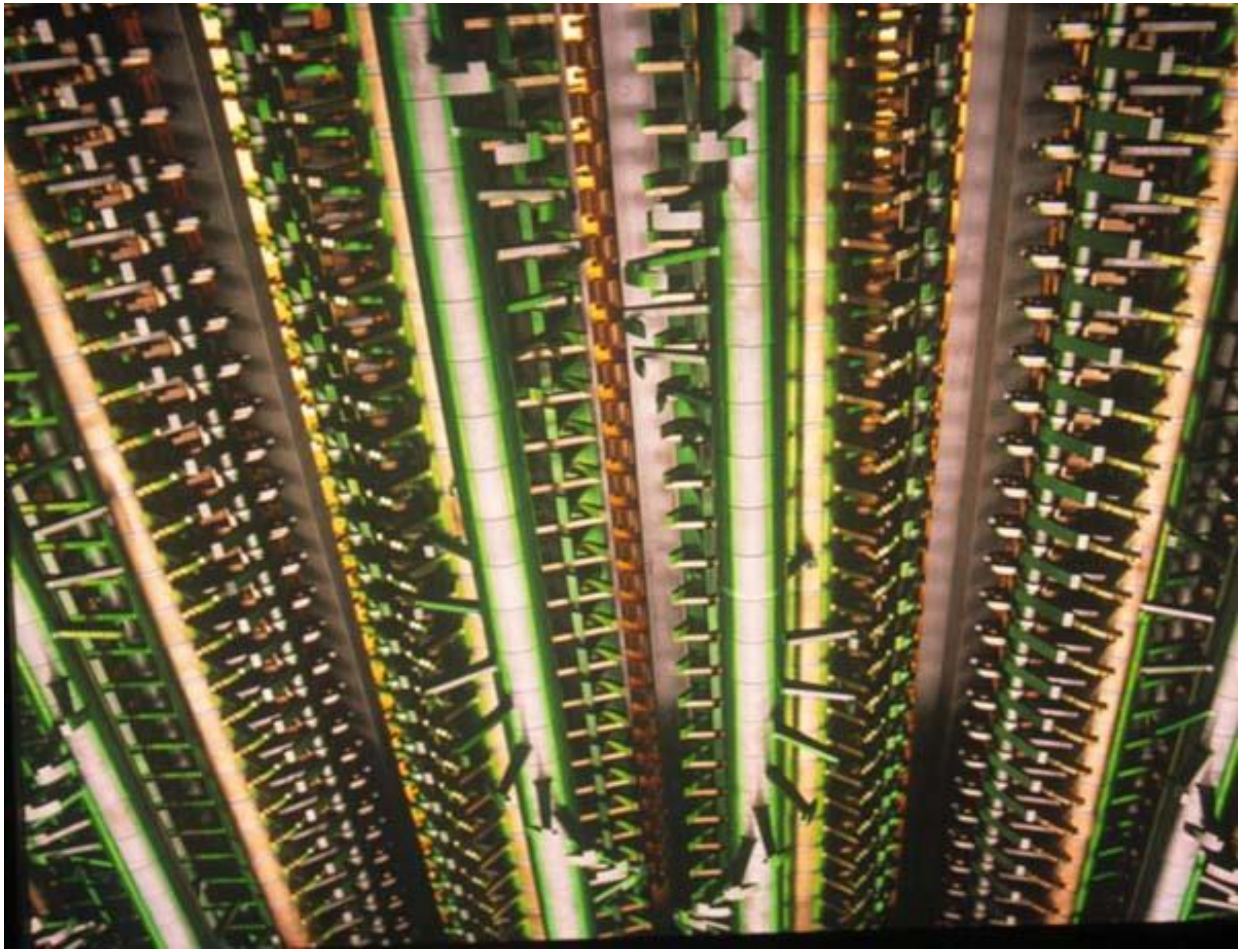


Why Integrated Circuits?

- Break this question into two questions
  - Why electronics
  - Why use ICs to build electronics

- Why use electronics
  - Electrons are easy to move / control
    - Easier to move/control electrons than real stuff
  - If you don’t believe me look at a mechanical computer
    - http://en.wikipedia.org/wiki/Difference_engine
    - Move information, not things (phone, fax, email, WWW, etc.)
    - Takes much less energy and $

Mechanical Alternative to Electronics



Picture of a version of the Babbage difference engine built by the Museum of Science UK

“The calculating section of Difference Engine No. 2, has 4,000 moving parts (excluding the printing mechanism) and weighs 2.6 tons. It is seven feet high, eleven feet long and eighteen inches in depth”

From Tubes

1945

ENIAC filled an entire room!

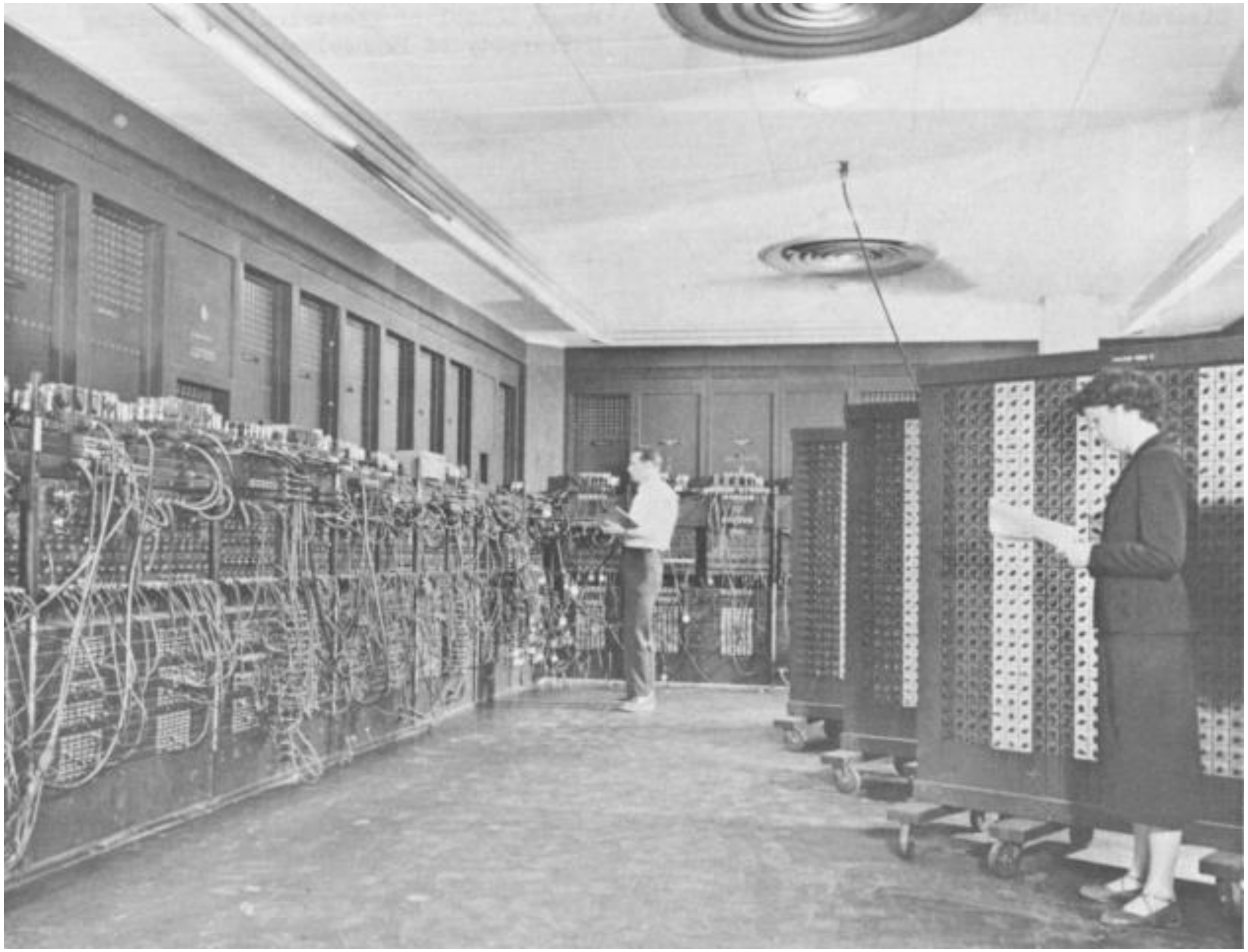
17,468 vacuum tubes,
70,000 resistors, and
10,000 capacitors

6,000 manual switches

and many blinking lights!

could add 5,000 numbers in a single second

ENIAC



To Transistors

1953

first computer with no tube

800 transistors and

10,000 germanium crystal rectifiers

only 100 watts

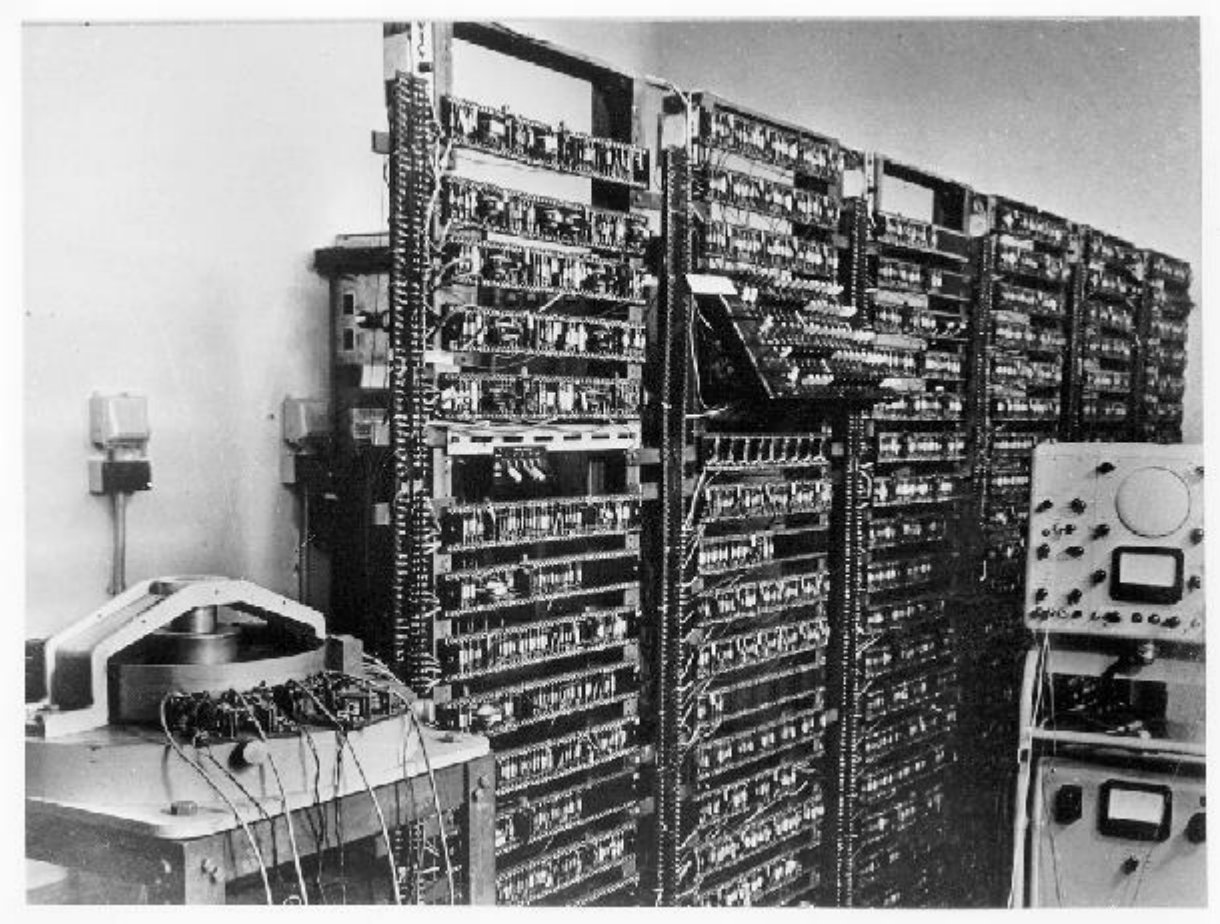
1947

point-contact transistor

TX-0 (MIT) and the Transistor 1 (Manchester)

TX-0

Both finished in 1953



What is an Integrated Circuit?

- A device having multiple electrical components and their interconnects manufactured on a single substrate.

- First IC 1958
  - Jack Kilby at TI
  - Germanium
  - A hack
    - Wax support
  - Made history

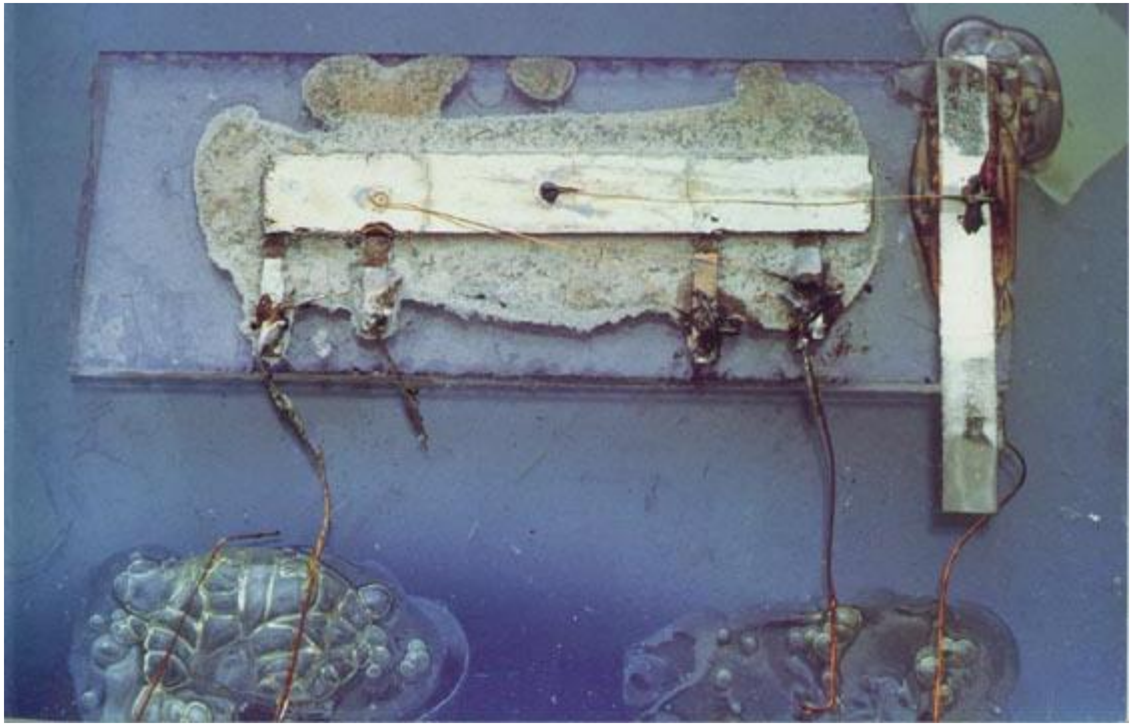


Image from State of the Art © Stan Augarten

First Useful IC 1961

- Planar Process (Fairchild)
  - Bob Noyce
  - Silicon

- People initially named things
  - SSI – Small scale
    - 60's
  - MSI – Medium scale
    - Early 70's
  - LSI – Large scale
    - Late 70s
  - VLSI – Very large scale
    - Early 80's
    - [Adjectives run out here]

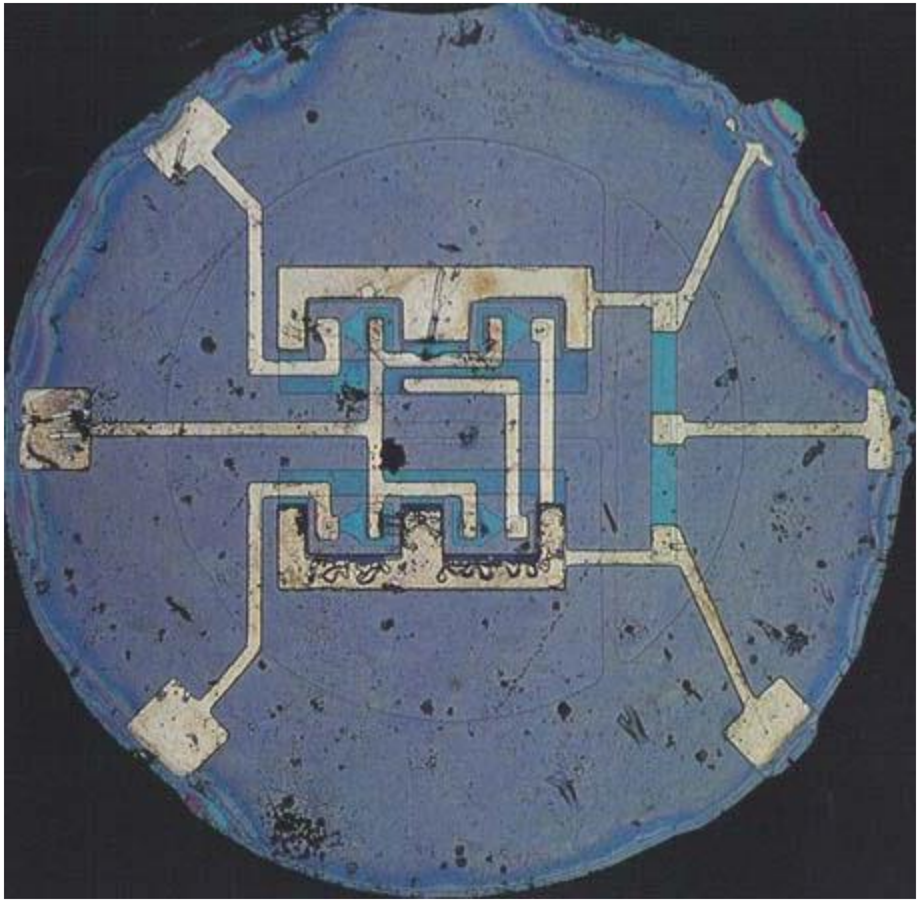


Image from State of the Art © Stan Augarten

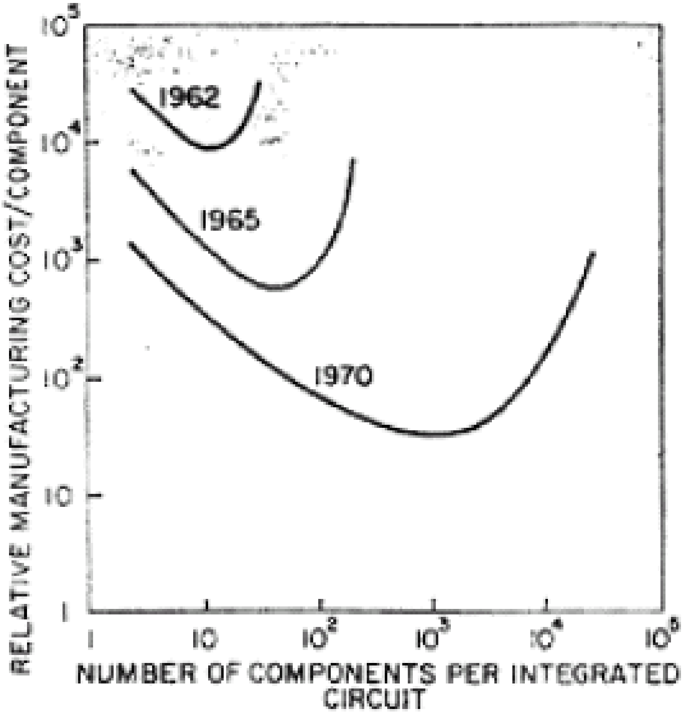
Electronics

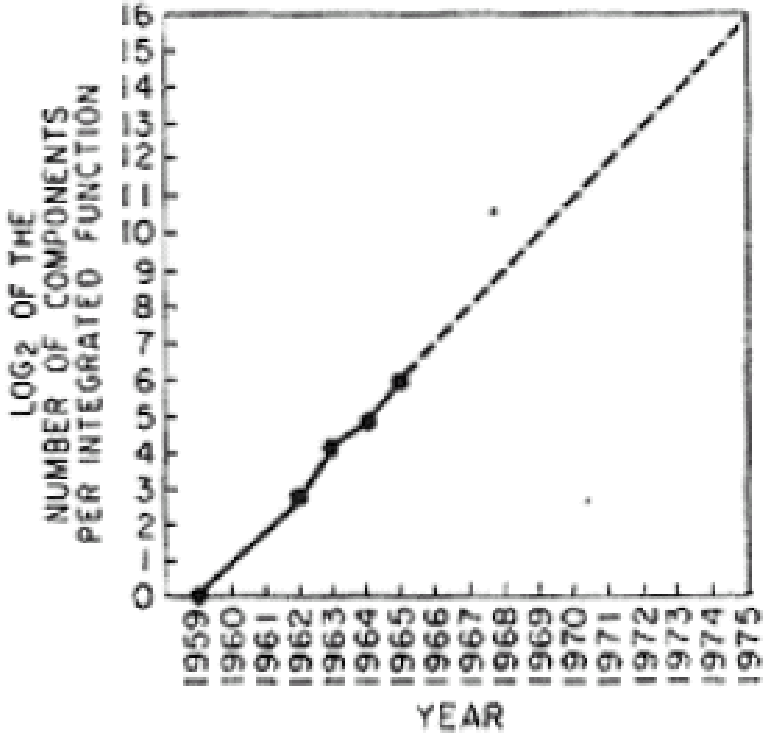
- Building electronics:
  - Started with tubes, then miniature tubes
  - Transistors, then miniature transistors
  - Components were getting cheaper, more reliable but:
    - There is a minimum cost of a component (storage, handling …)
    - Total system cost was proportional to complexity

- Integrated circuits changed that
  - Printed a circuit, like you print a picture,
    - Create components in parallel
    - Cost no longer depended on # of devices
  - What happens as resolution goes up?

Moore’s Law

From Electronics, Volume 38, Number 8, April 19, 1965



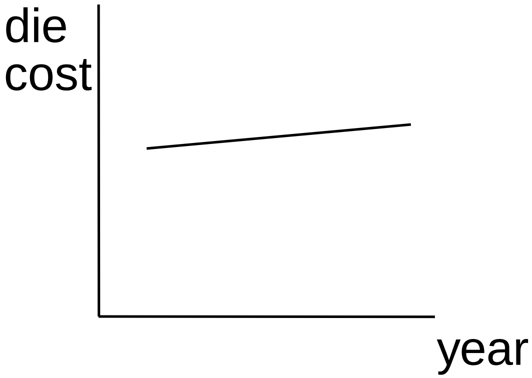


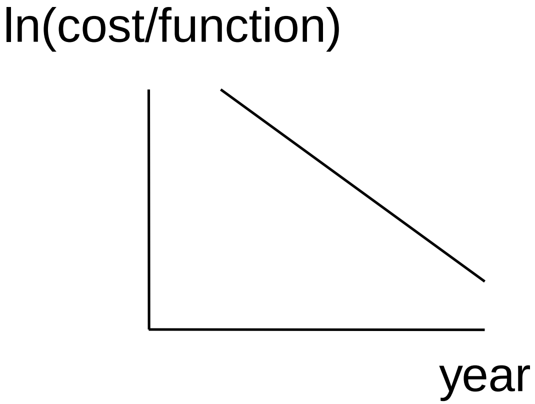
- Original “law”: number of components on IC doubles every year
- Later modified to doubling every 18 months, later to 2 years, etc

- Moore’s Law

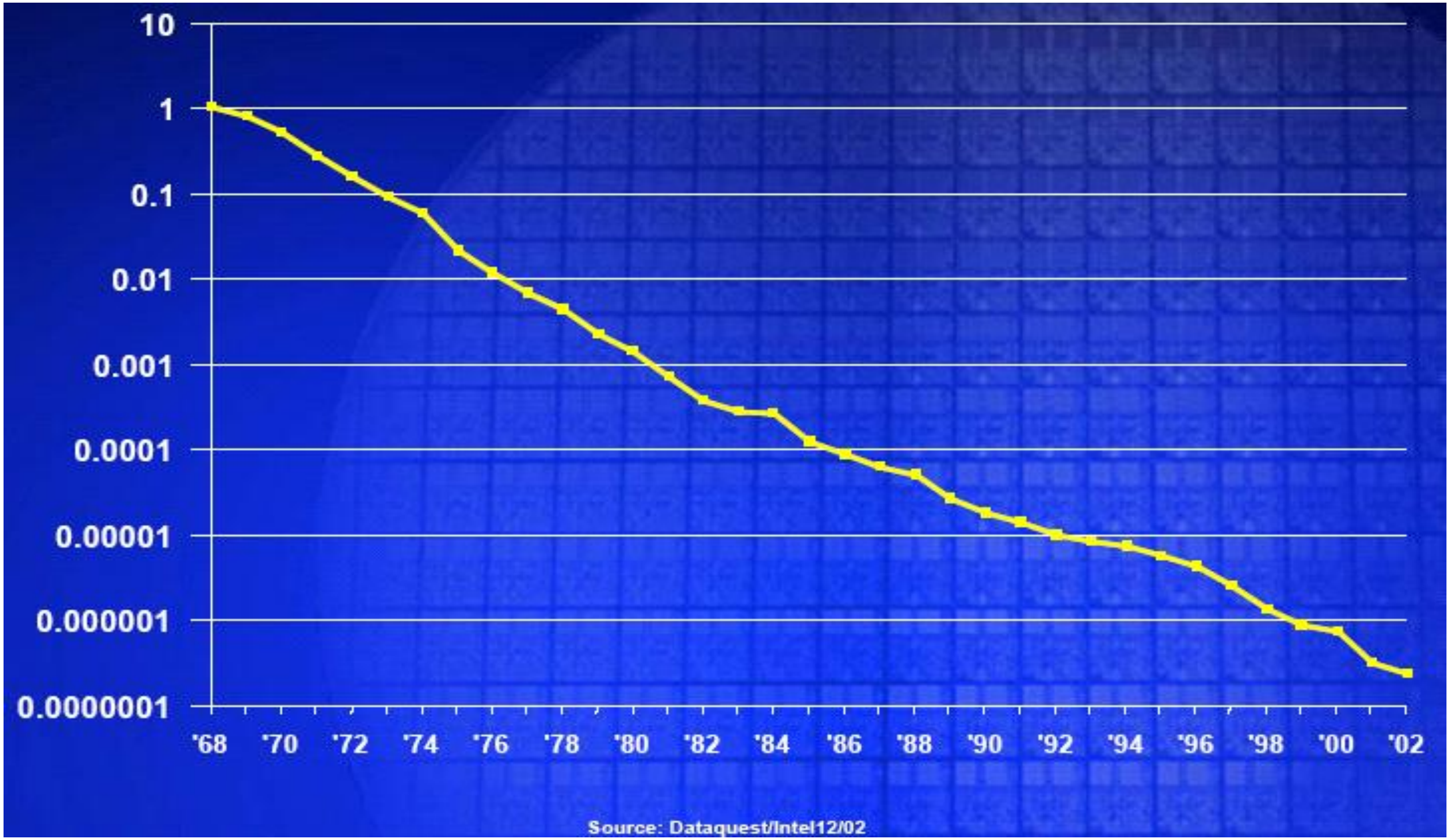
First stated by Intel’s Gordon Moore in the mid 60’s. Saw that the resolution of the printing process was improving exponentially (0.7x feature size every 3 years) and predicted that it would continue into the future

Since the cost of the printing process (called wafer fabrication) was growing at a modest rate, it implied that the cost per function, was dropping exponentially. At each new generations, each gate cost about 1/2 what it did 3 years ago. Shrinking an existing chip makes it cheaper!





Average Transistor Cost ($)



Source: Dataquest/Intel12/02

No Exponential is Forever...but We Can Delay 'Forever', Moore ISSCC 2002

How Much Can We Put on a Chip?

Bottom line: A lot

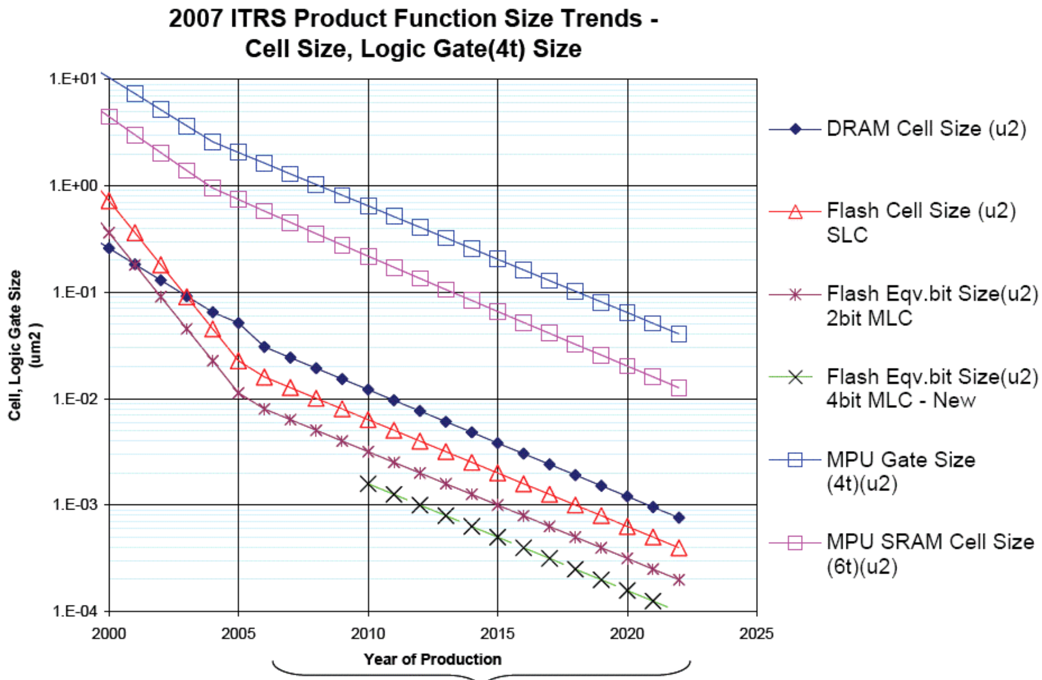
- Some more details:
  - Large chip is 20mm x 20mm
  - 2009 feature size is around 50nm (45nm)
  - 20 features /μ, 20K/mm, 0.4M x .4M = 1.6E11 on a large chip

- DRAM cell is about $6F^2$, FLASH cell is effectively $4F^2$, std cell is $400F^2$
  - 400M std cells, 30B DRAM cells, 80B Flash cells
  - Of course you need other stuff (memory is about ½ cells)

- But you can build almost anything on a chip today
  - Next year you can build even more stuff on your chip

Moore’s Law Today

- International Technology Roadmap for Semiconductors (ITRS)



2007 - 2022 ITRS Range

http://www.itrs.net/Links/2007ITRS/ExecSum2007.pdf

Figure 8 2007 ITRS Product Function Size Trends:
MPU Logic Gate Size (4-transistor); Memory Cell Size [SRAM (6-transistor); Flash (SLC and MLC), and DRAM (transistor + capacitor)]

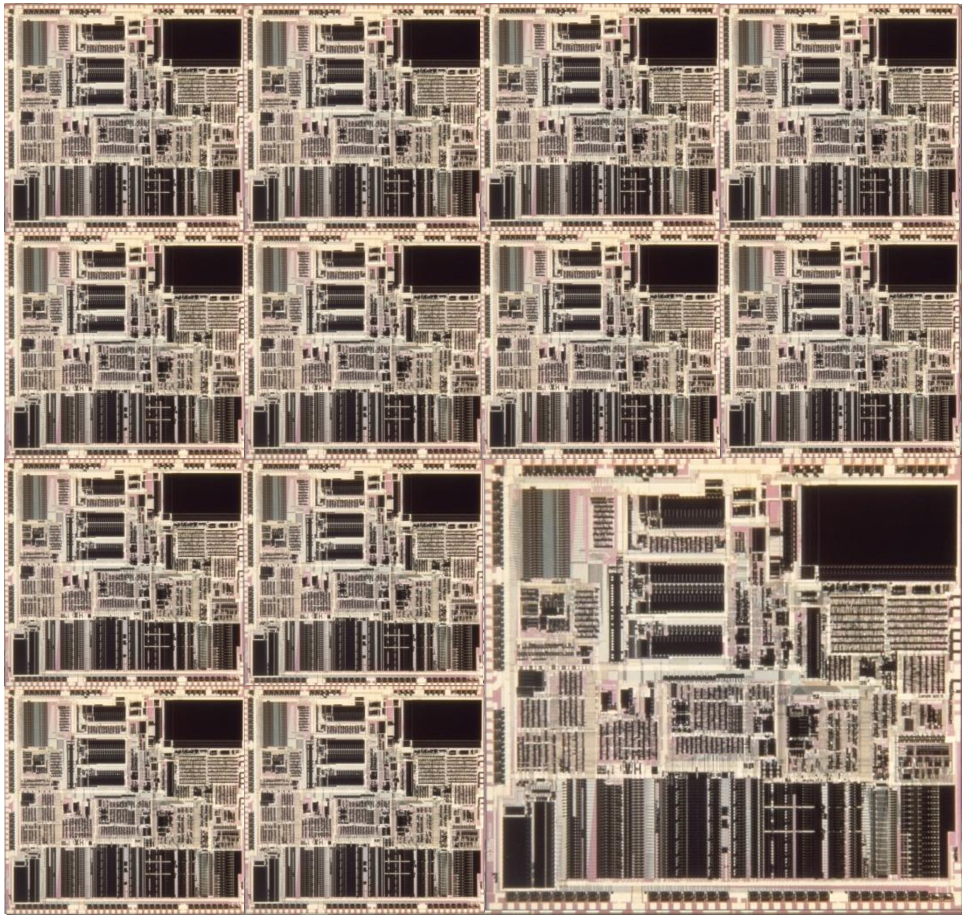
What This Means

- SoA Chip in 1985 (start of CMOS chips) (80386)

Five years later

Ten years later

A chip in 2009 could contain around 1000 80386 processors



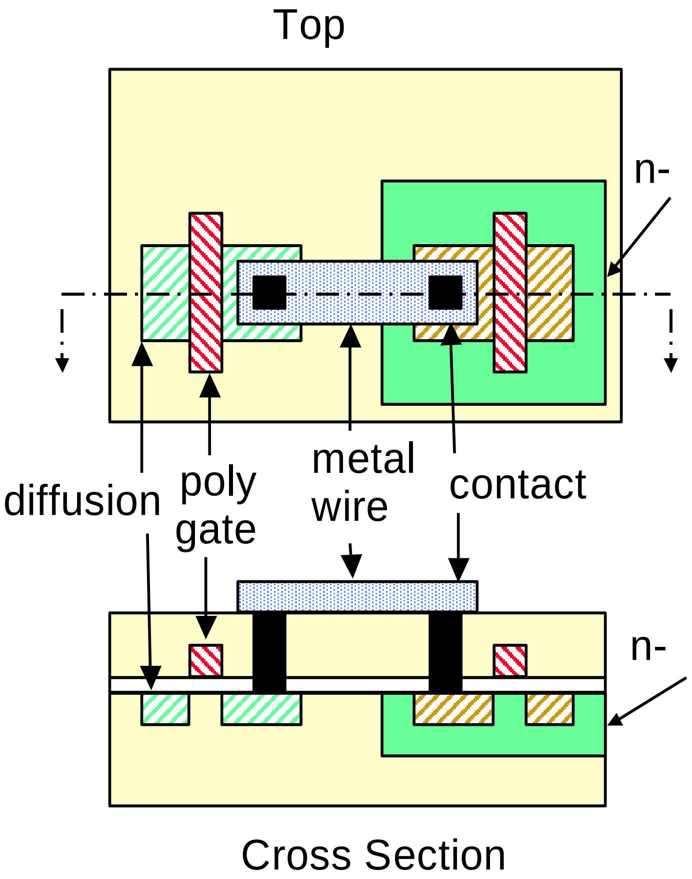
Perspective: In Your Lifetime

- Most of you were born 1996 +/- a few years
- Cray-1: world’s fastest computer 1976-1982
  - 64Mb memory (50ns cycle time)
  - 40Kb register (6ns cycle time)
  - ~1 million gates (4/5 input NAND)
  - 80MHz clock
  - 115kW
- In 45nm technology (circa 2008)
  - 64Mb => 2mm^2
  - 1 million NAND-4 gates => 1mm^2
  - 40Kb register => 0.04mm^2
  - Fits in a 1.7mm x 1.7mm die, run at over 1GHz

CRAY-1

Creating a Chip of Your Own

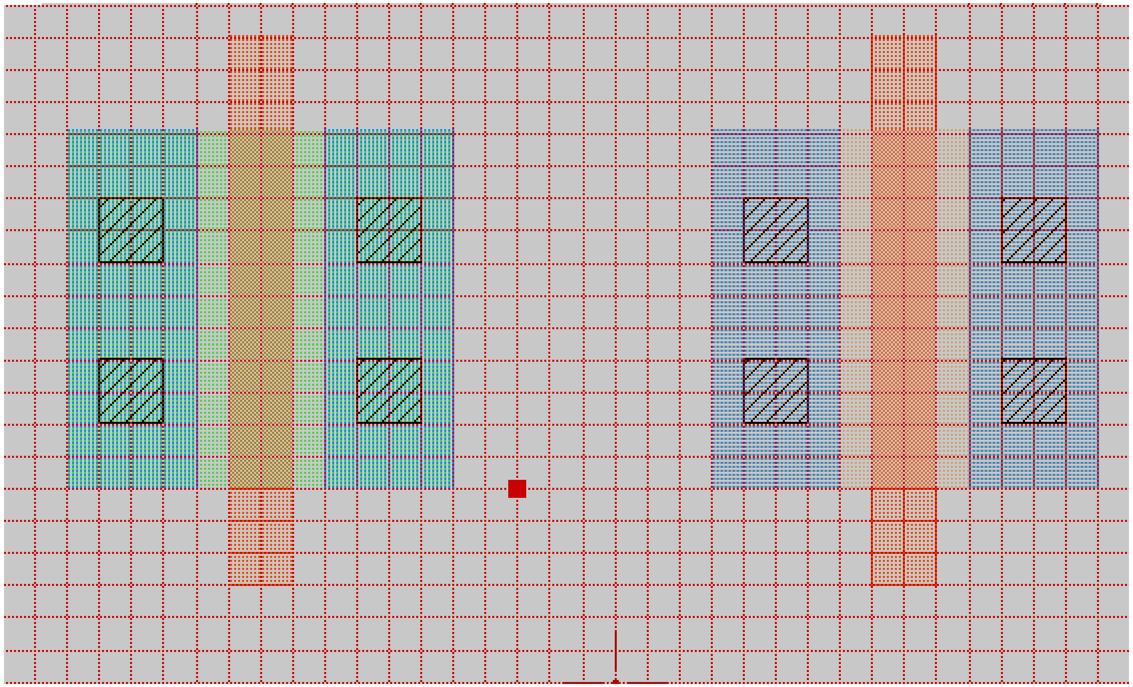
- Process is deceptively simple
  - Similar to printing
- Choose a fabrication company ("fab")
  - Like choosing a printer
- Produce a set of negatives ("masks")
  - Like creating color layers
    - 4+ negatives for transistors
    - 2 negatives per layer of wiring
- Send negatives and $$$ to fab
  - Much more $$$ than printing
- Wait 2-3 months for fabrication
  - Much slower than printing
- Package and test chips



Screen clipping taken: 8/28/2009, 2:39 PM

How Do You Know What to Draw? Part 1

- Beauty of modern chips:
  - Only two components: nMOS and pMOS transistors



  - And contact and wires

How Do You Know What to Draw? Part 2

- Still need to convert transistors to useful functions
  - Two step process

- Step One (next two lectures)
  - Model transistors as switches
  - Build simple logical functions from switches (next lecture)

- Step Two (rest of the class)
  - Build complex functions from simple “gates”
  - And check that they work the way you want them to

- Still seems pretty simple

The Hard Part – Managing Complexity

- Chips have zero tolerance for errors

- The previous example had four polygons and two transistors
  - A real design will have
    - At least 10 million polygons and 1M transistors
    - Any single error in any of the polygons can ruin the chip
  - Need to make sure it works (functionally, performance, power)

- Mistakes are really expensive
  - At 45nm a full set of masks cost $2M

- No one person can really comprehend 1 million of anything
  - Much less 1 billion

Hard Part cont’d – Validation

- Remember that those polygons must match specification
  - Ensure each implementation matches specification

- But devices have pretty complex electrical behavior
  - Can’t simulate millions of them in a reasonable time
  - Need to create simpler abstractions
    - Use them to enable you to create solutions
    - Use them to validate system operation
  - Implies
    - Need to create tools to check abstractions/constraints
    - Need to test design at different levels of abstraction

Digital Design Is About Complexity Management

- Simplify the design problem
  - Create abstractions
    - Simplified model for a thing,
    - Works well in some subset of the design space
  - Constrain designers – Create design methodology
    - Needed to ensure that the abstractions are valid
    - Might work if you violate constraints, but guarantees are off
  - Enforce constraints
    - Create tools to check the constraints

- Validate the design
  - Functionality, performance, power
  - At different levels in the design hierarchy

And Methodology Creation

- Since scaling constantly changes the problem
  - At least makes it larger
  - And now larger and harder

- Need to create new methods to simplify design
  - You need to design each gate for ½ the cost of last time
  - Since you have 2x gates

- Requires understanding the underlying technology
  - Create new abstractions and constraints that work
  - Determine efficient solutions / template
    - In design time, area, power, or performance

Digital Design

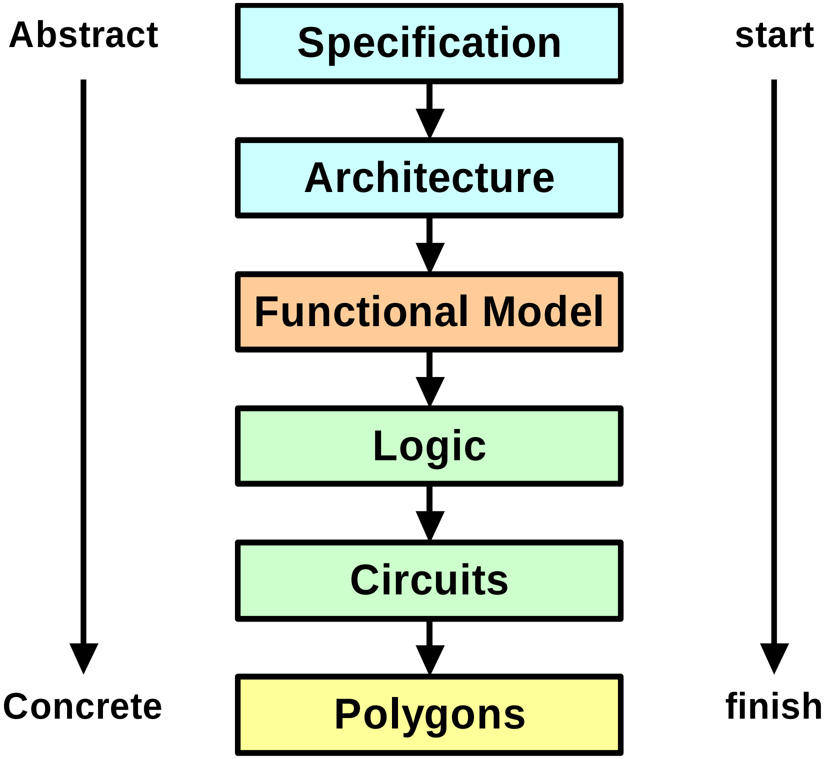
Besides all that,

I think it is fun.

I hope you agree.

Levels of Design Abstraction

- Have different levels of details
  - Top level is your goal
    - Initially not executable
    - Often becomes C++ code
  - Then create microArch
    - Rough hardware resources
    - Rough communication
    - Can be executable
  - Functional Model
- Design is never top down or bottom up. It is really iterations to match the constraints on both ends: hardware and spec.



Digital Design Flow

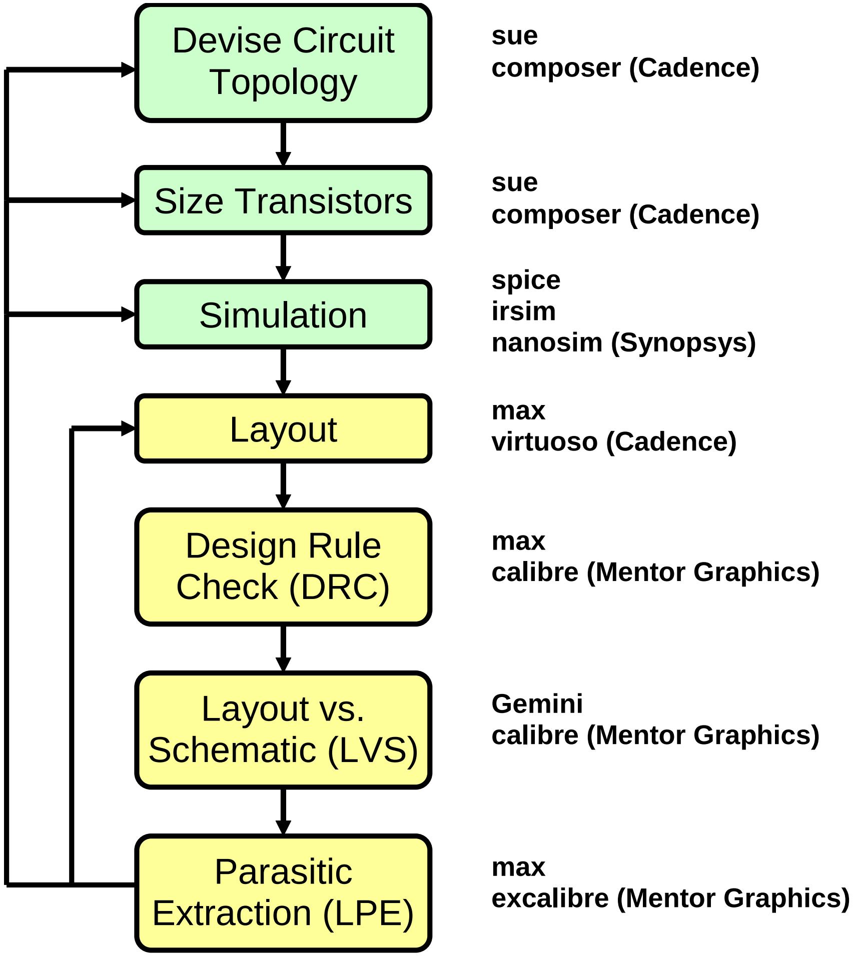
- Almost all designs use a cell based design flow
  - Most of the low-level layout has been done already
    - Layout for logic gates/flops exists in std cells
    - Memories are also done by generators
  - Can have larger functions completed as well
    - Can be laid out as a large cell (hard macro)
    - Can be given to you as a collection of std cells (soft macro)

- Sometimes there is a need to create some custom cells
  - Create std cell library
  - Could be for some mixed signal (analog circuits)
  - Or for some special function blocks that are critical
  - These cells follow a “full custom flow”

Full Custom Design Flow

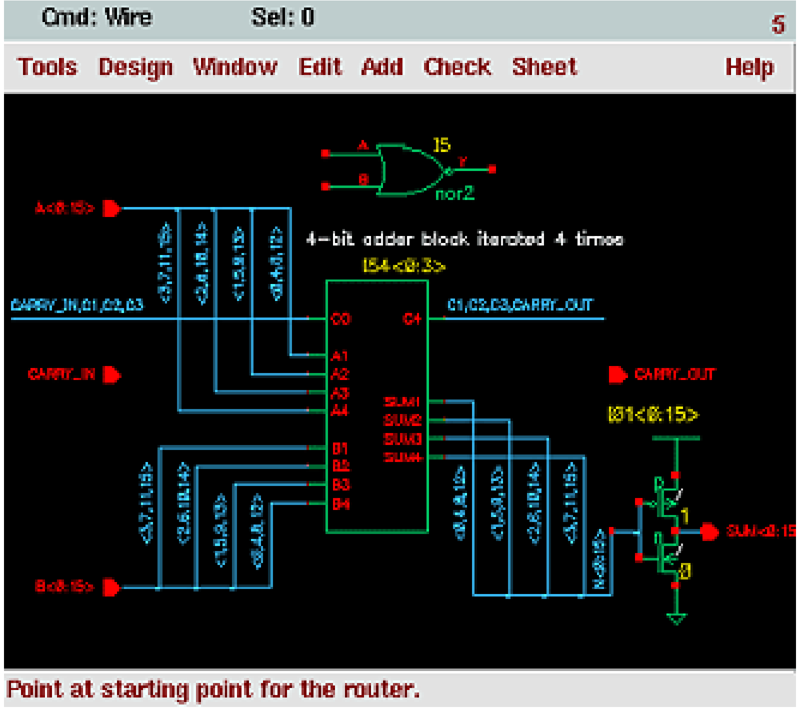
- Gives the designer the most freedom
  - Lots of rope
    - Can be clever
    - Can hang yourselves too

- For a specific function
  - Can achieve best performance
    - Speed, power, area, etc
  - Most work/time per function
  - Optimizations are at a low level
    - Circuit better be important
    - Think assembler, only worse



Schematic Capture/Simulation

- Circuit drawn many levels
  - Transistor, gate, and block
- Uses hierarchy
  - Blocks inside other blocks
  - Allows reuse of designs
- Tool create simulation netlists
  - Components and all connections

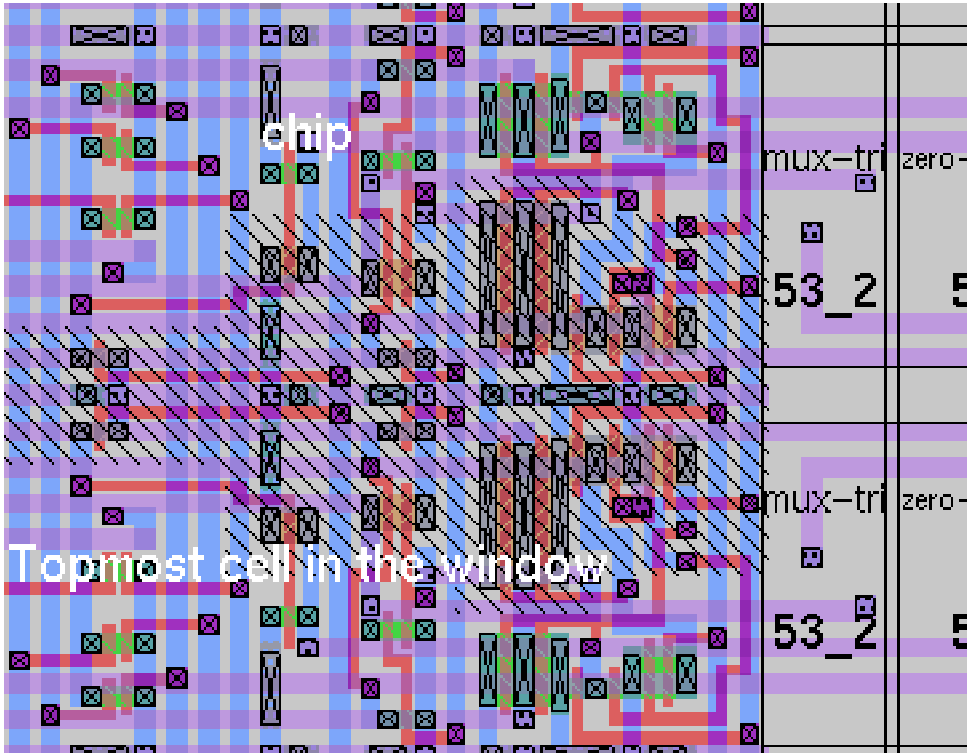


Point at starting point for the router.

Cadence Virtuoso Schematic Composer

Layout

- Draw and place transistors for all devices in schematic
- Rearrange transistors to minimize interconnect length
- Connect all devices with routing layers
- Possible to place blocks within other blocks
  - Layout hierarchy should match schematic hierarchy



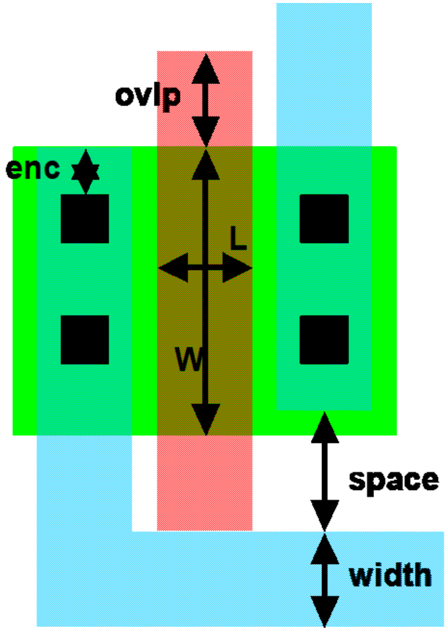
magic layout editor

Design Rule Checking (DRC)

- Fab has rules for the polygons
  - Required for manufacturability

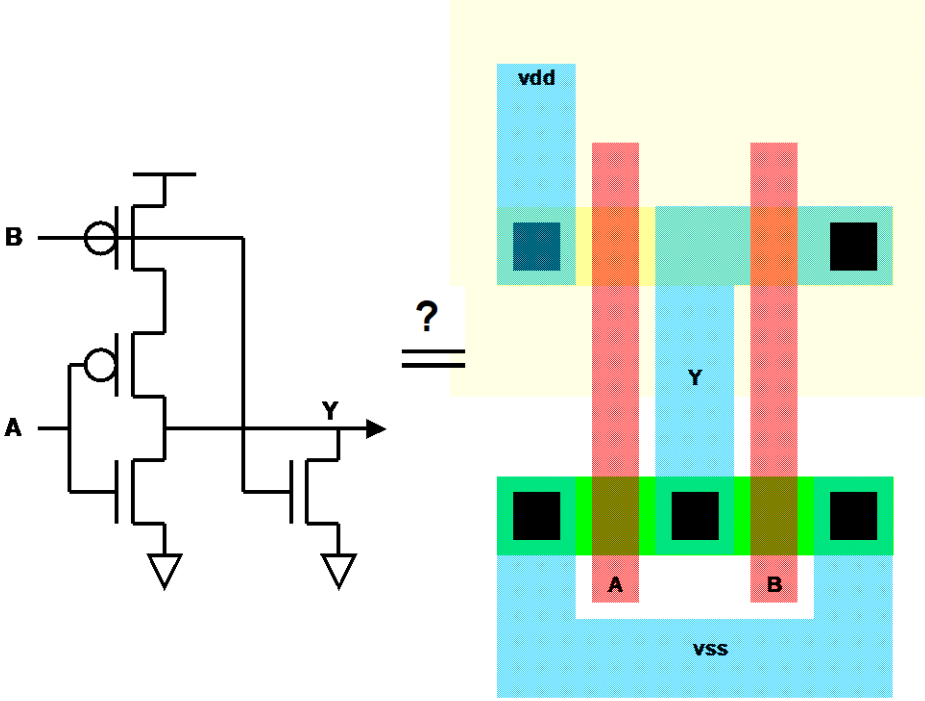
- DRC checker looks for errors
  - Width
  - Space
  - Enclosure
  - Overlap
  - Lots of complex stuff (more later)

- Violations flagged for later fixup

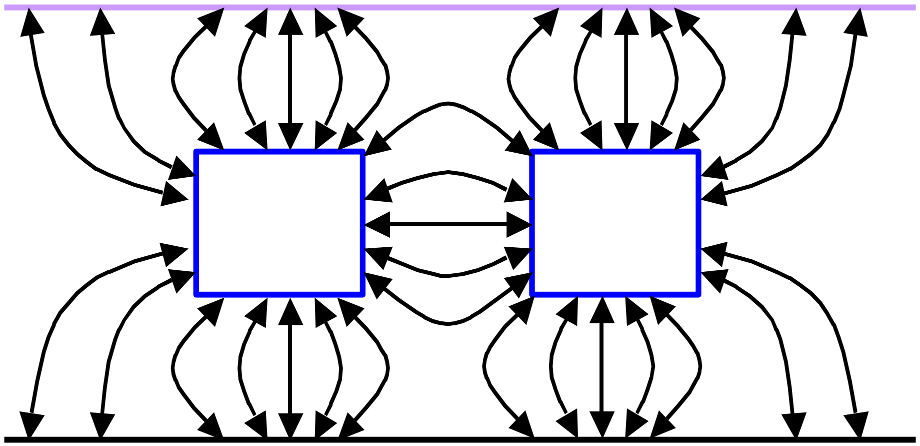


Layout Versus Schematic (LVS)

- Extracts netlist from layout by analyzing polygon overlaps
- Compares extracted netlist with original schematic netlist
- When discrepancies occur, tries to narrow down location



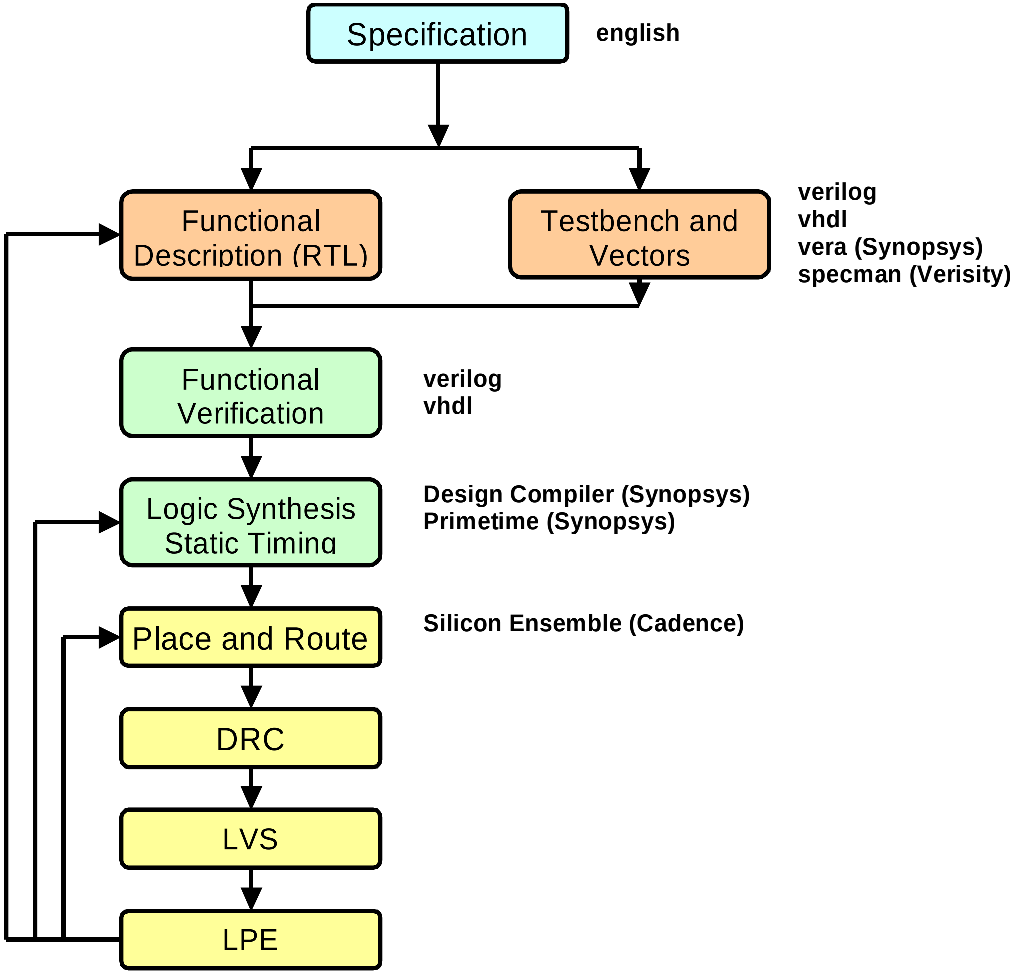
Layout Parasitic Extraction (LPE)



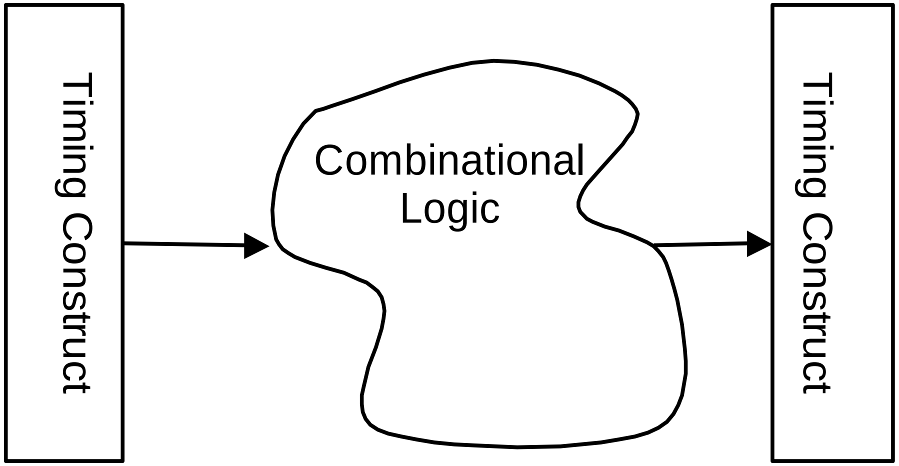
- Estimates capacitance between structures in the layout
- Calculates resistance of wires
- Output is either a simulation netlist or a file of interblock delays

Cell Based Design Flow (ASIC Flow)

- Separate teams to design and verify
- Physical design is (semi-) automated
- Loops to get device operating frequency correct can be troubling



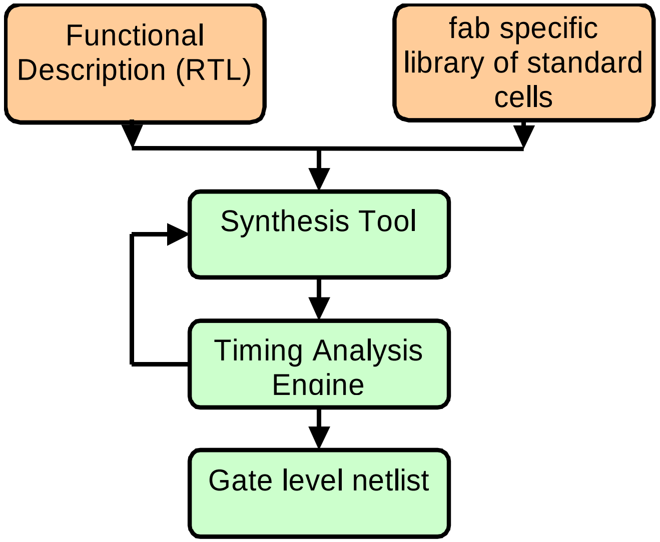
Register Transfer Level (RTL)



- Sections of combinational Goo separated by timing statements
  - Defines behavior of part on every clock cycle boundary

Logic Synthesis

- Changes cloud of combinational functionality into standard cells (gates) from fab-specific library
- Chooses standard cell flip-flop/latches for timing statements
- Attempts to minimize delay and area of resulting logic

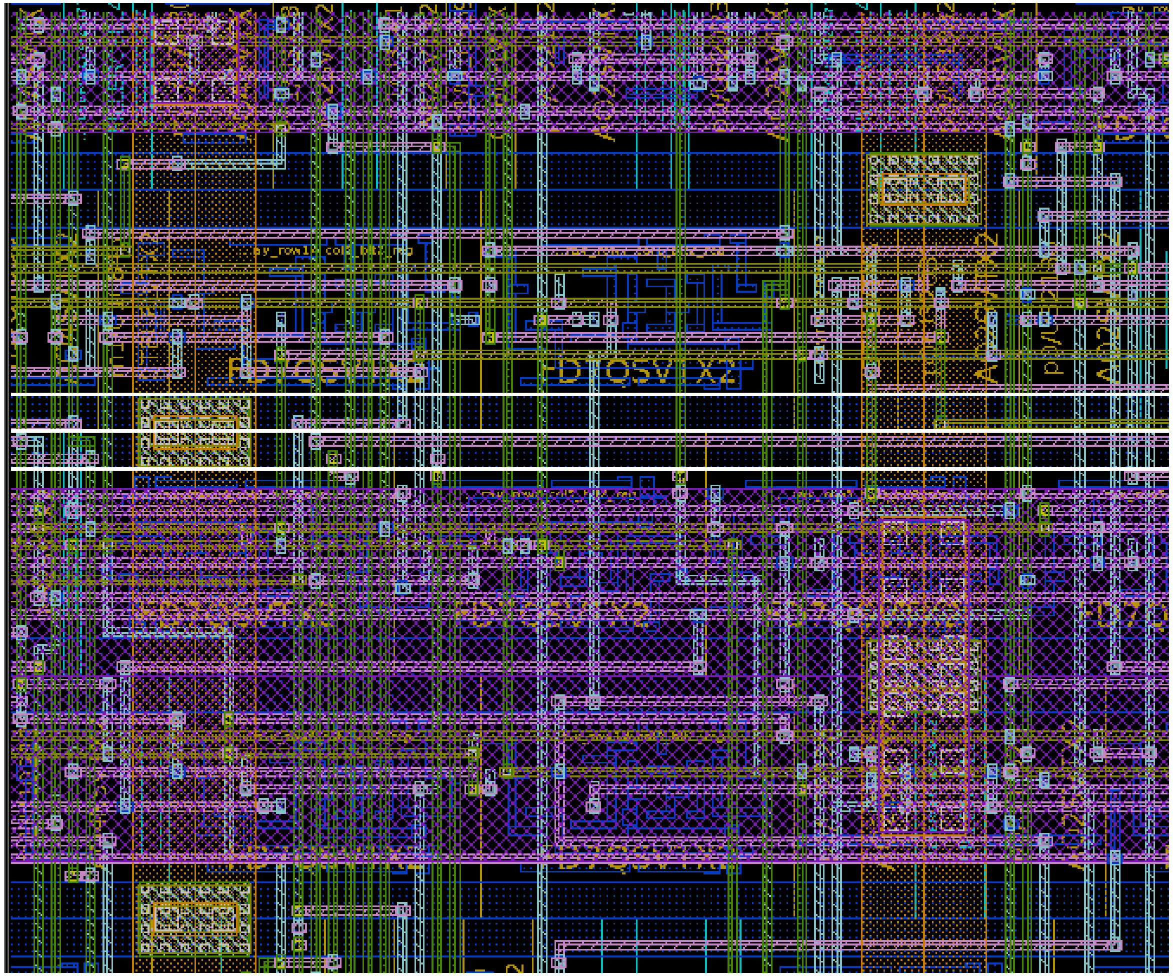

Functional Description (RTL)
fab specific library of standard cells
Synthesis Tool
Timing Analysis Engine
Gate level netlist

Standard Cell Placement and Routing

- Place layout for each gate ("cell") in design into block

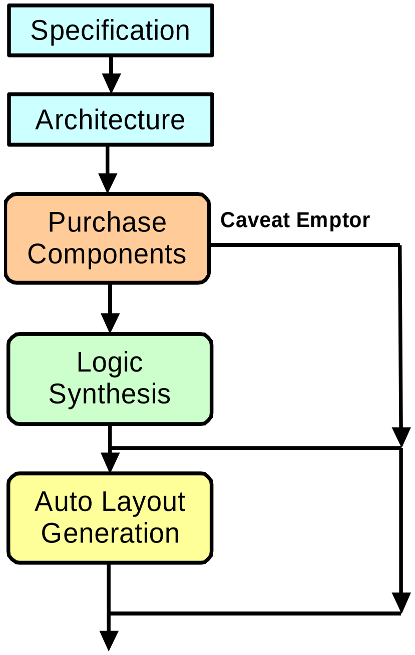
- Rearrange cell layouts to minimize routing

- Connect up cells



Using Higher-Level Functions

- Can buy “Intellectual Property” (IP) from various vendors
- “Soft IP”: RTL or gate level description
  - Synthesize and Place and Route for your process.
  - Examples: Ethernet MAC, USB
- “Hard IP”: Polygon level description
  - Just hook it up
  - Examples: XAUI Backplane driver, embedded DRAM



Chip Assembly

- Integrate blocks from previous steps
  - Real chips have different types of blocks
- Can resemble picture on right
  - Key is to have a early plan
  - And continue to update it
  - Need to have accurate floorplan
- Early Floorplanning is key
  - Sets the specs for the components
  - Functional, physical, timing

Most chips are constructed by correction; not correct by construction!

Validation and Tape Out

- Making a mistake is very expensive
  - Have a tool check all previous types of mistakes
    - Check all errors, sign off on false positives, fix errors
  - Run check tool again

- Tape out
  - Used to write 9-track computer tapes for mask making
  - Now, transfer polygons to fabrication company via ftp

- You’re done! (Except for documentation, test vector generation, device bringup, skew lots, reliability tests, burnin…)