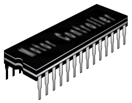


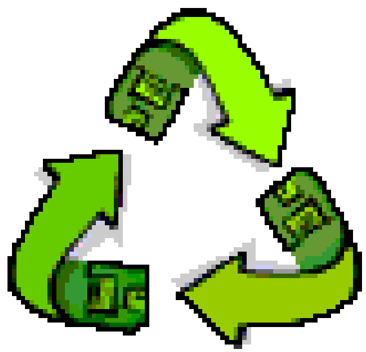
EL 653

Integrated Circuit Design Introduction

Design Process

Design: specify and enter the design intent

Verify: verify the correctness of design and implementation

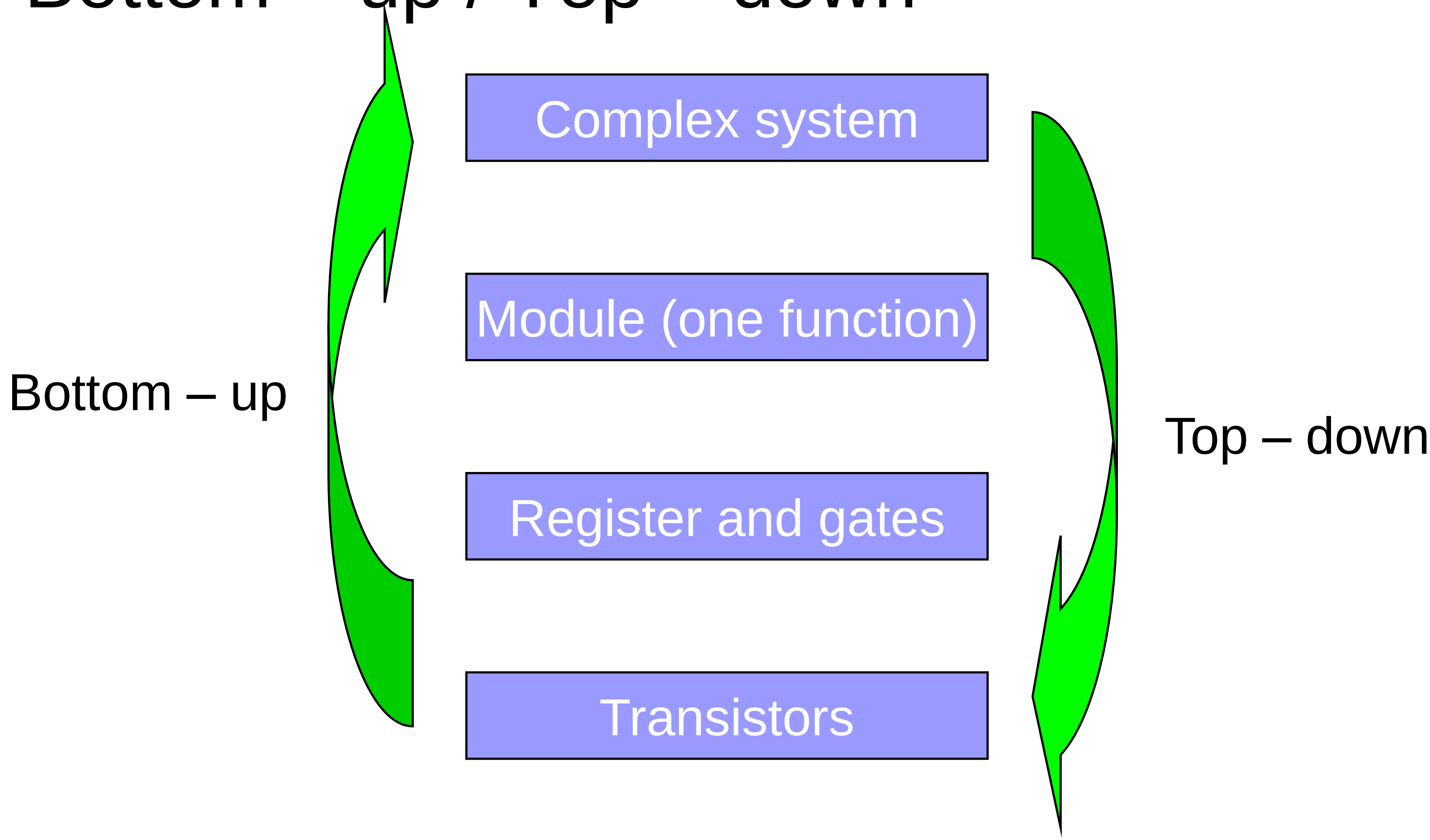


Implement: refine the design through all phases

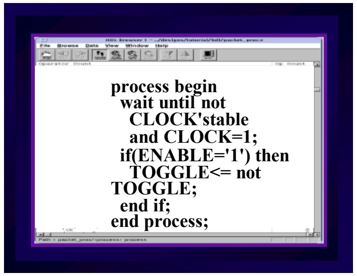
Bottom – up / Top – down

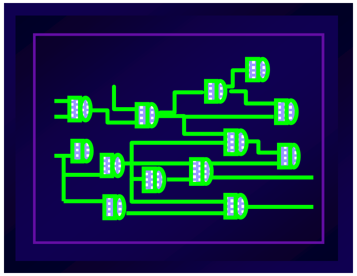
- Bottom – up
  - Start from simple modules
  - Goes to complex modules
  - Suitable to create small parts that will be reused
- Top – down
  - Start from complex modules
  - Goes to simple modules
  - Suitable for big systems

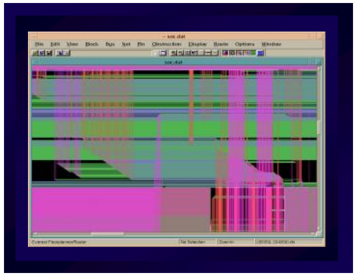
Bottom – up / Top – down

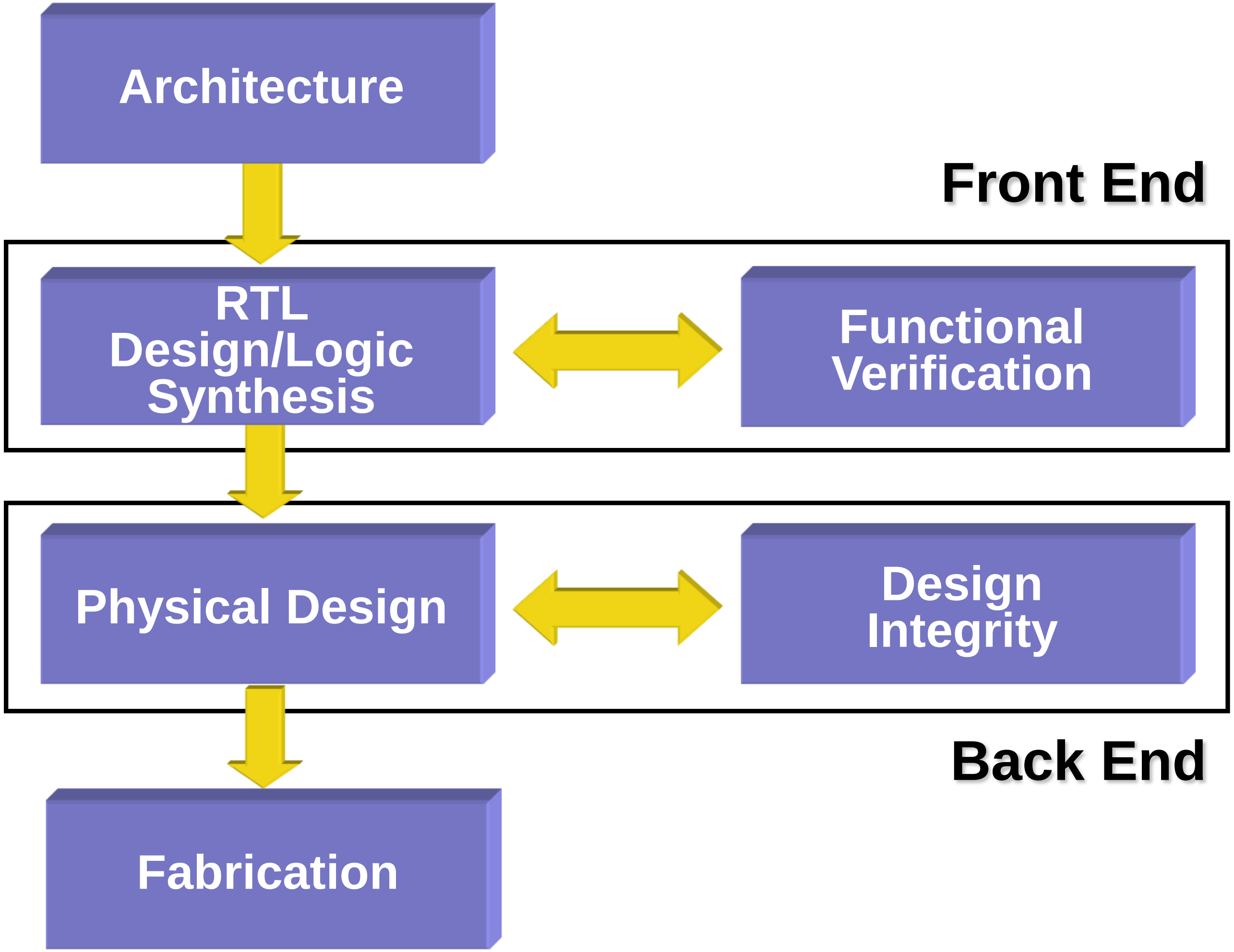


IC Design . . . A Simplified Explanation









The Front End

- Architecture:
  - Key Algorithms (filtering, for example)
  - Amount of on-chip Memories, sizes?
  - How many Integer Proc Units?

- RTL: Register Transfer Language
  - Verilog (1988), VHDL, SystemVerilog: an executable spec for the chip, amounting to over a million lines of code
  - Lots of simulations to verify the spec (literally billions of cycles)
  - Timing constraints, clock definitions, etc

The Front End

- Logic Design: convert the RTL to logic gates (NAND-NORs, NOTs, Registers)
  - A manual process in the past, still mostly manual for Analog
  - Logic Synthesis (1989): automate the process
  - Many discrete optimization techniques used here: boolean minimization, static timing analysis, state equivalence, etc, etc.
  - End point is a “netlist”, meaning a set of logic gates and their connections. A large netlist is in the 10s of millions of gates
  - Can be simulated or “formally verified” versus the RTL.
  - Key technique: how do you prove that two logic equations are equivalent?

The Back End

- Floorplanning
  - Where do we place the large blocks? Where do we place the “random” logic and “structured blocks”? A combination of manual and automated approaches is used
  - Need to keep connections short to meet timing, but also cannot “congest” the design too much or we cannot complete the connections
  - Note that connections do have R and C (to substrate and coupling between wires) so they introduce delay! Meeting timing can be very difficult!
  - The Power and Ground lines usually get decided here

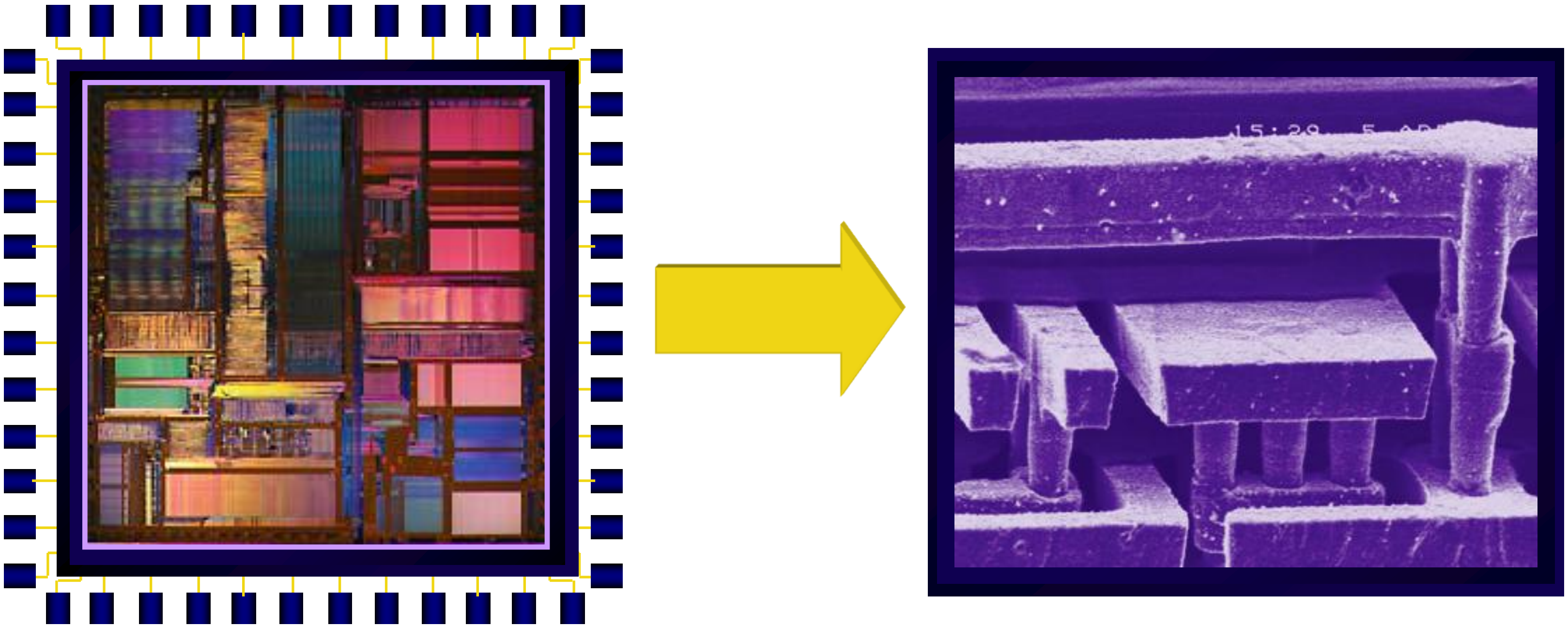
- Placement:
  - Now we need to complete the exact details of where each block and gate will be
  - Automation has been a key for many years (1980). A block may contain hundreds of thousands of cells, so it is very hard problem: minimize area, be routable and meet timing
  - Note may have to add logic: repeaters to restore signals a key example

The Back End

- Routing
  - Complete all the connections!
  - But, need to meet timing and keep signal integrity. This also involve separating some wires, for example, to avoid bad couplings
  - Automation is the norm here (1980)

- Verification:
  - Spacing and sizing rules are checked for all polygons (1980)
  - Parasitics are extracted, netlists back annotated and time analyzed using static techniques (1990)
  - Manufacturing requires complicated rules, such as wire density been “uniform”

Design Goes to Fabrication



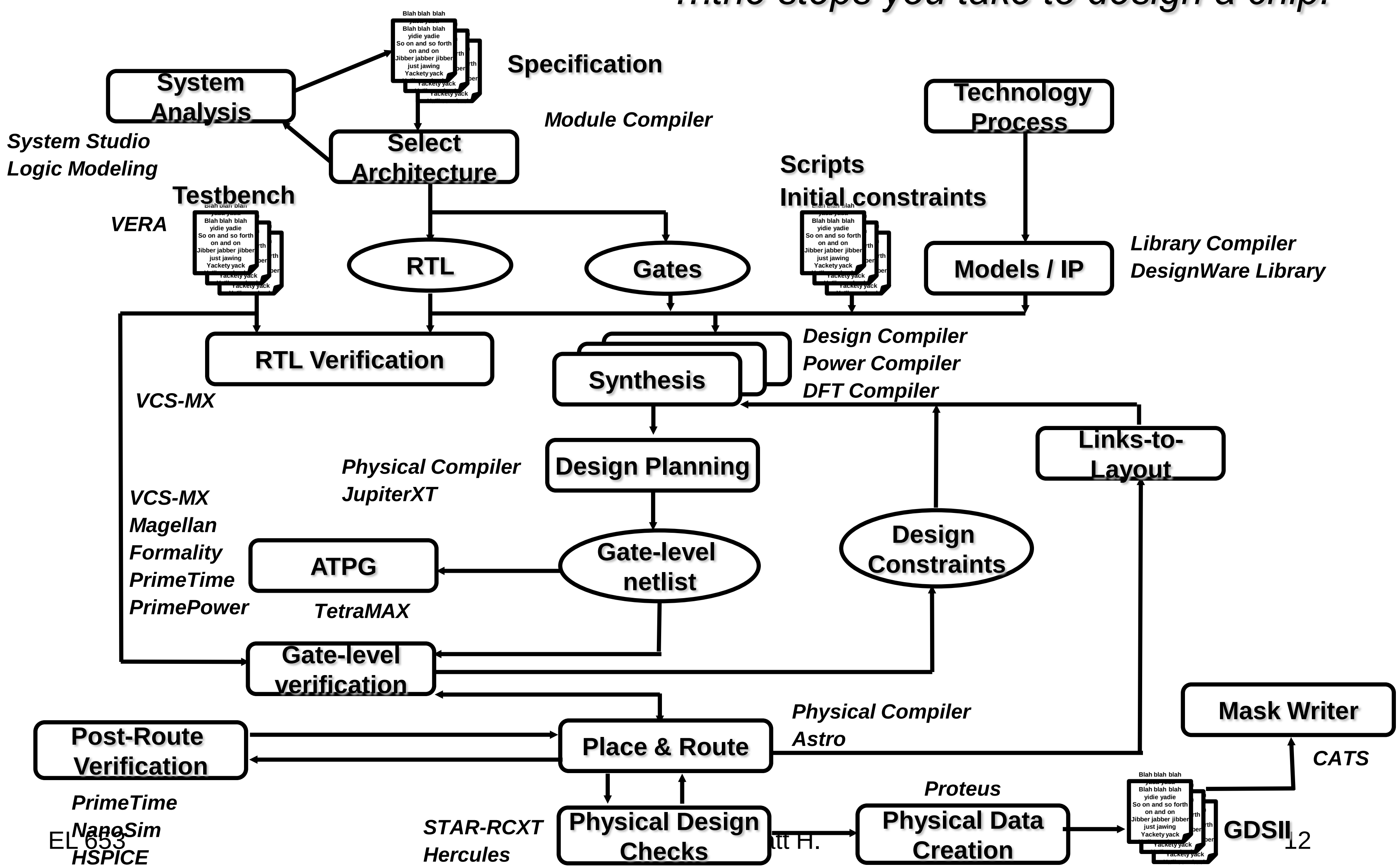
Sounds simple, but have a host of very hard problems to solve!

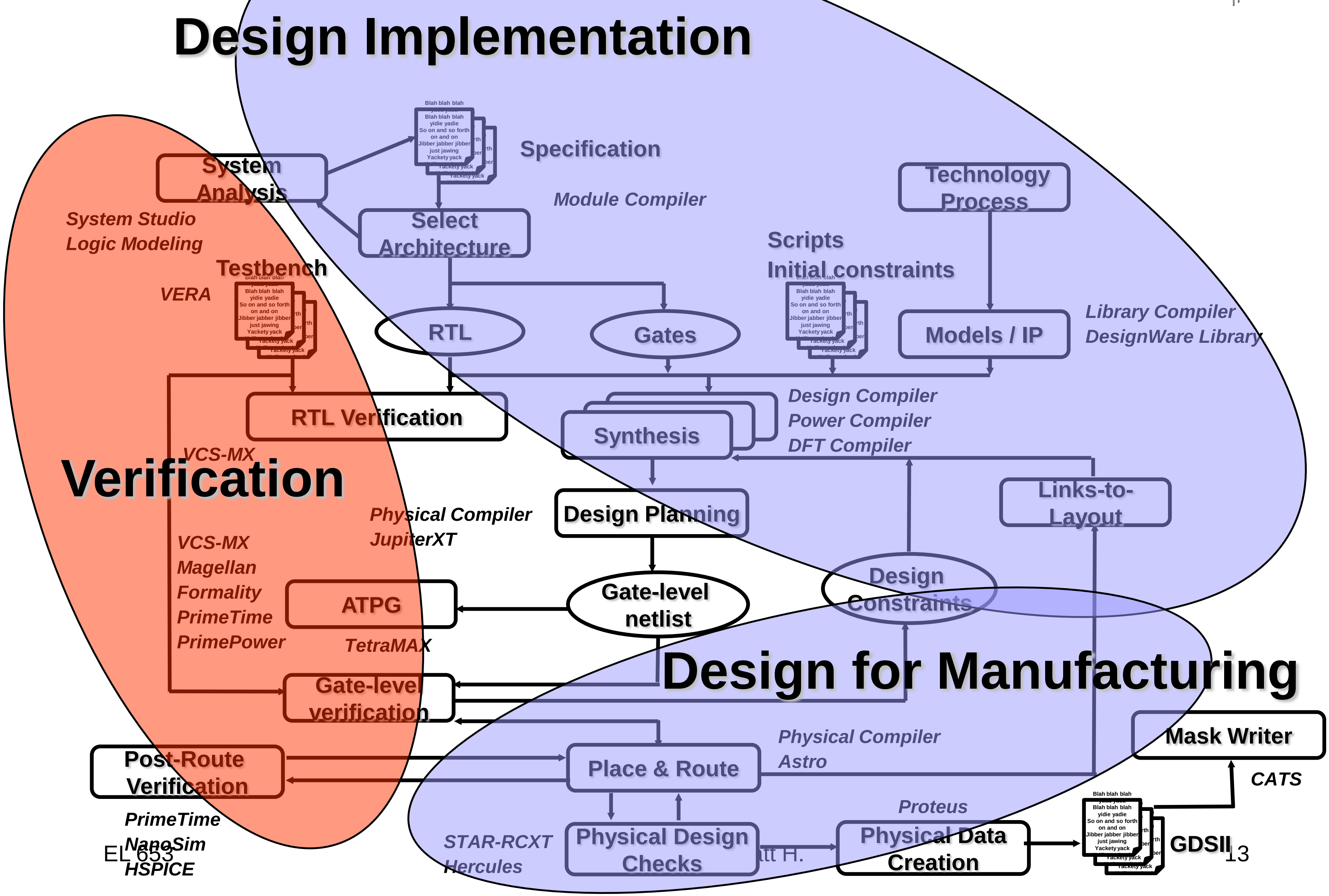
Fabrication

- Mask fabrication
- Wafer fabrication
- Wafer testing
- Assembly and packaging
- IC test

What's a design flow?

…the steps you take to design a chip!





Design Implementation
Specification
System Analysis
Select Architecture
Module Compiler
Technology Process
Scripts
Initial constraints
System Studio
Logic Modeling
Testbench
VERA
RTL
Gates
Models / IP
Library Compiler
DesignWare Library
RTL Verification
Synthesis
Design Compiler
Power Compiler
DFT Compiler
VCS-MX
Verification
Links-to-Layout
Physical Compiler
JupiterXT
Design Planning
VCS-MX
Magellan
Formality
PrimeTime
PrimePower
ATPG
Gate-level netlist
Design Constraints
TetraMAX
Design for Manufacturing
Gate-level verification
Mask Writer
Physical Compiler
Astro
Post-Route Verification
Place & Route
CATS
PrimeTime
NanoSim
HSPICE
Proteus
STAR-RCXT
Hercules
Physical Design Checks
Physical Data Creation
GDSII

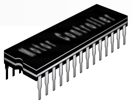
A simple design flow – step by step

1. “Spec” your chip
2. Generate the gates
3. Create a test
4. Ensure the gates will work
5. “Blueprint” your chip
6. Double-check your blueprint
7. Turn your design into sand

Design Implementation

Verification

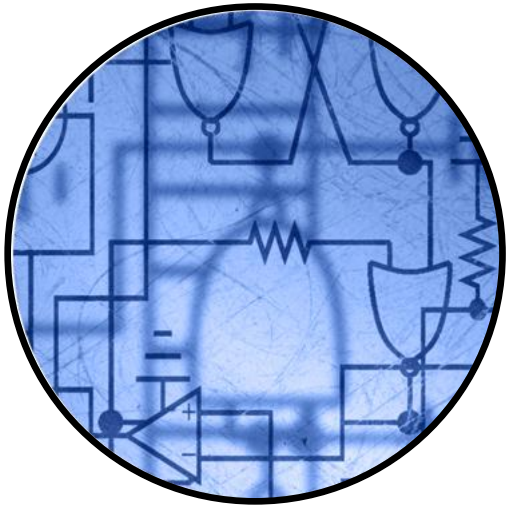
Design for Manufacturing



1. “Spec” your chip

- Describe what you want your chip to do
- Write a “spec” - use a language like Verilog or VHDL
  - A spec for a cell phone ringer might look like this:
    - if incoming_call AND line_is_available then RING;
- Today’s complex chip designs
  - can have a million lines of specification,
  - created by a team of 100 people,
  - working for 6 months!
- You can buy a ready-made spec
  - example: DesignWare Library
- Give the spec to the computer to begin automation
  - example: (V)HDL Compiler

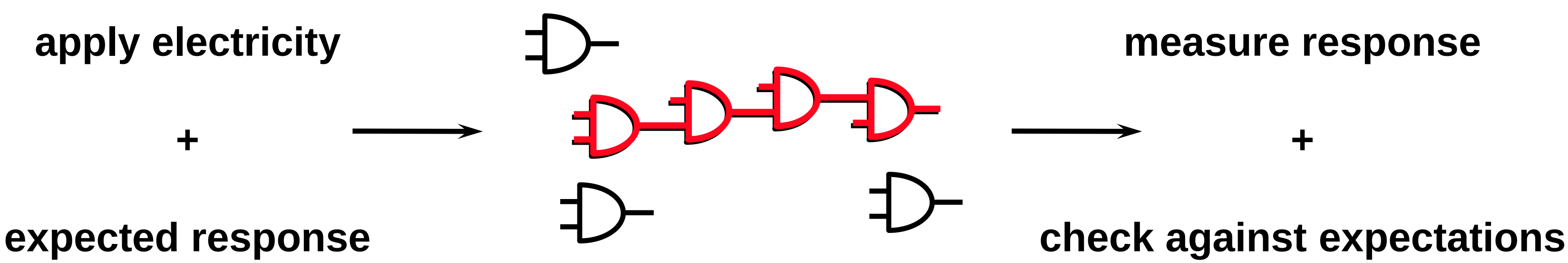
2. Generate the gates



- Figure out the detailed logic gates
  - Today's chip designs have hundreds of thousands of gates in each one!
  - Can't figure out all these logic gates with pencil and paper
- Use a computer program! An EDA tool!
- Synthesis
  - example: Design Compiler, Physical Compiler, IC Compiler
- Save the logic gates to use in a future design
  - example: DesignWare Library

3. Create a test

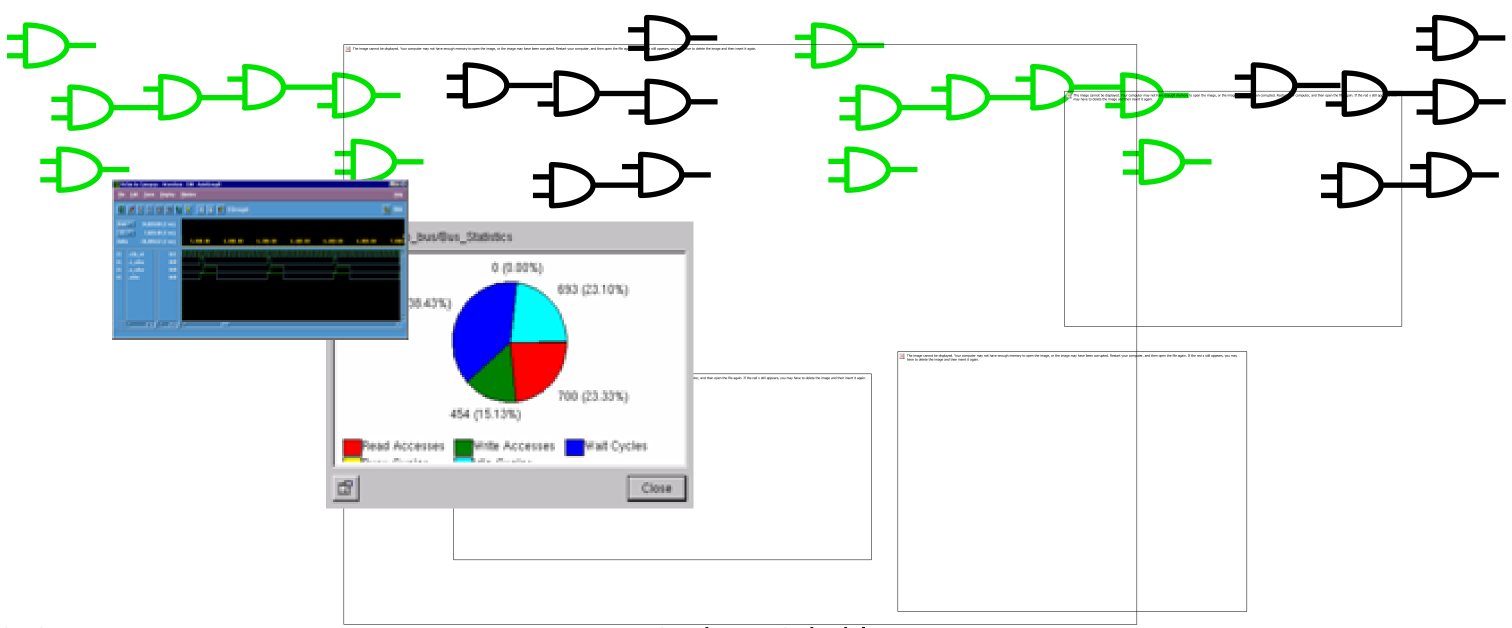
- For manufacturer to throw out defective chips
- Need millions of combinations of electrical stimuli
- Crazy to attempt this with a pencil and paper
- Use EDA tools! Test Synthesis & Automatic Test Program Generation
  - example: DFT Compiler, TetraMAX



4. Ensure the gates will work

- Verify that the logic gates will do what you want, when you want them to
  - Simulation, Timing Analysis, Testbench Generation
    - example: VCS-MX, PrimeTime, VERA

0
1
0
1
1

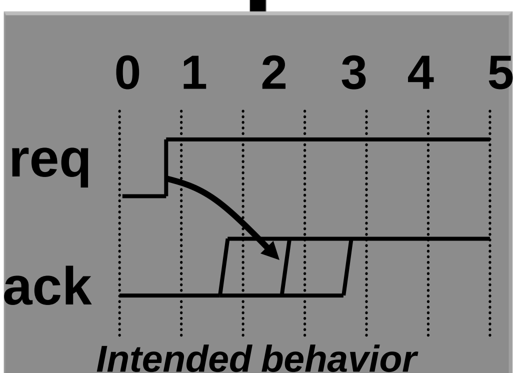


4. Ensure the gates will work

- Put statements in the design description
  - Assertion-based verification = “Smart Verification”

Assertion

```
assert property (@(posedge clk) $rose(req)|-> ##[1:3] $rose(ack));
```

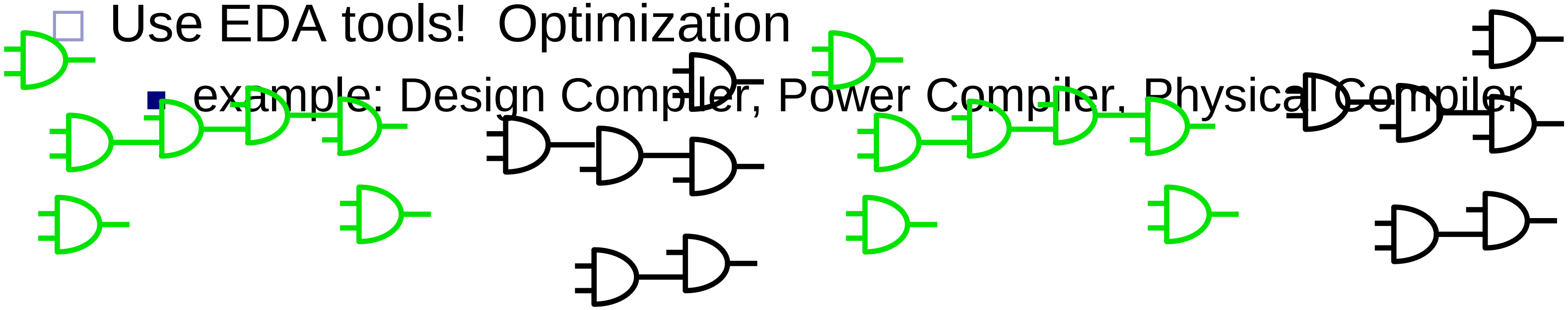


Old way

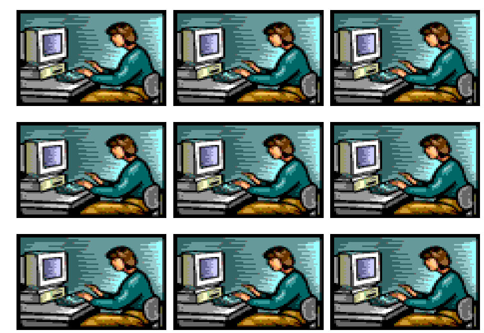
```
always @(posedge req)
  begin
     repeat (1) @(posedge clk);
     fork: pos_pos
        begin
           @(posedge ack)
           $display("Assertion Success",$time);
             disable pos_pos;
        end
       begin
          repeat (2) @(posedge clk);
          $display("Assertion Failure",$time);
          disable pos_pos;
       end
     join
  end // always
```

4. Ensure the modifications will work

- Tweak the whole thing to make it better
  - Faster (speed), smaller (area), less battery (power)
  - No way to figure it out with pencil and paper
  - Use EDA tools! Optimization
    - example: Design Compiler, Power Compiler, Physical Compiler

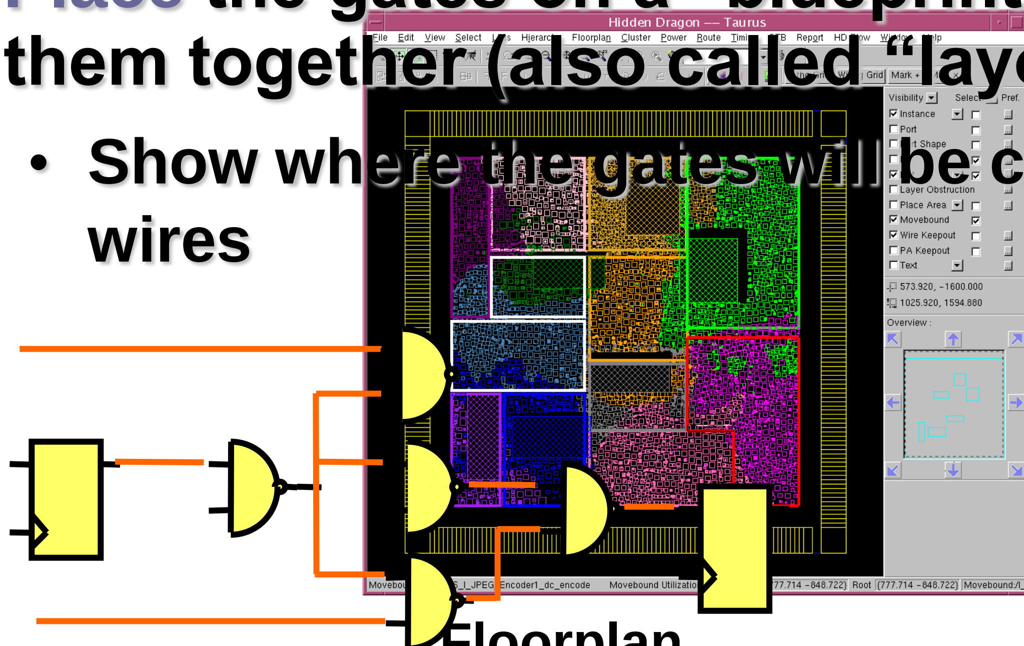


- Make sure it's still the same design - <u>Formal Verification</u>
    - ***example: Formality***

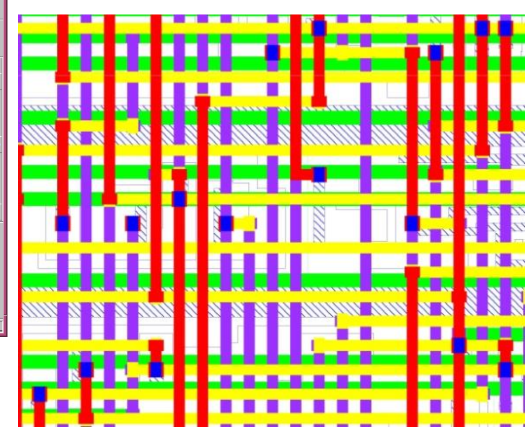


5. “Blueprint” your chip

- Design planning: create a “floorplan” so the gates will go where you want
- **Place the gates on a “blueprint” and Route them together (also called “layout ”)**
  - **Show where the gates will be connected with wires**



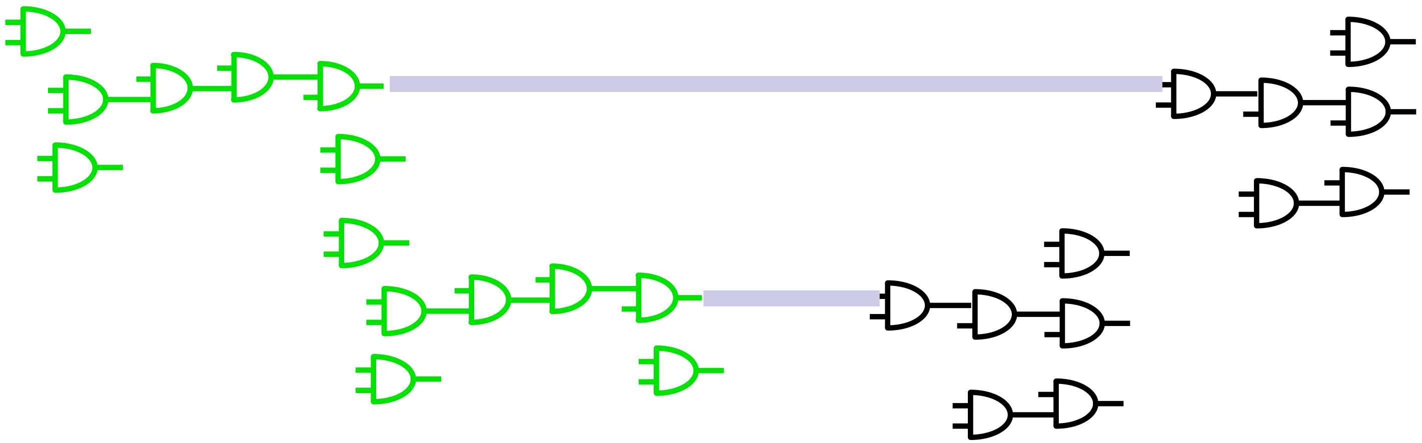
Floorplan



- ***example: Jupiter-XT, Physical Compiler, IC Compiler, Astro***

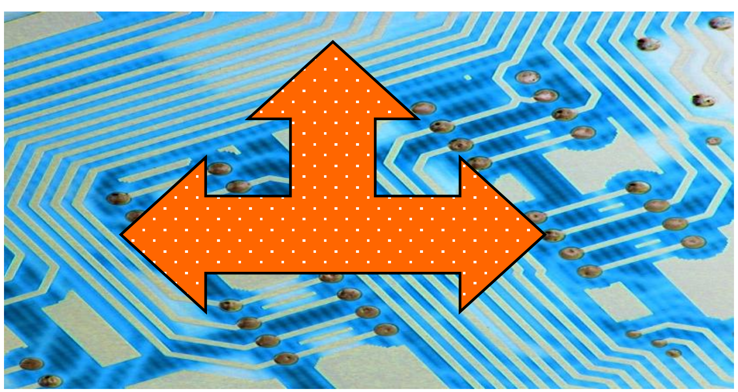
6. Double-check your blueprint

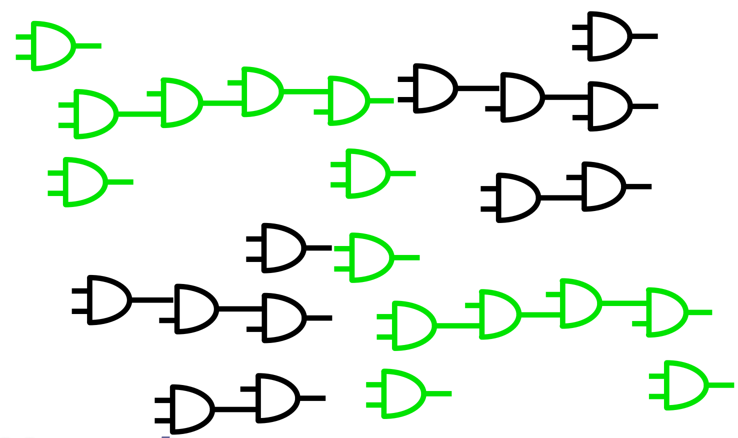
- Length of the connecting wires will change how fast the chip will run
- Simulate it again - Post-route (post-layout) verification
  - example: VCS, VCS-MX, NanoSim, HSPICE



6. Double-check your blueprint

- Make sure "what you see is what you get"
  - Compare what you designed to what's in your layout
  - Layout versus Schematic (LVS)
- **Follow the manufacturer’s rules**
  - **Perform Design Rule Checks (DRC)**

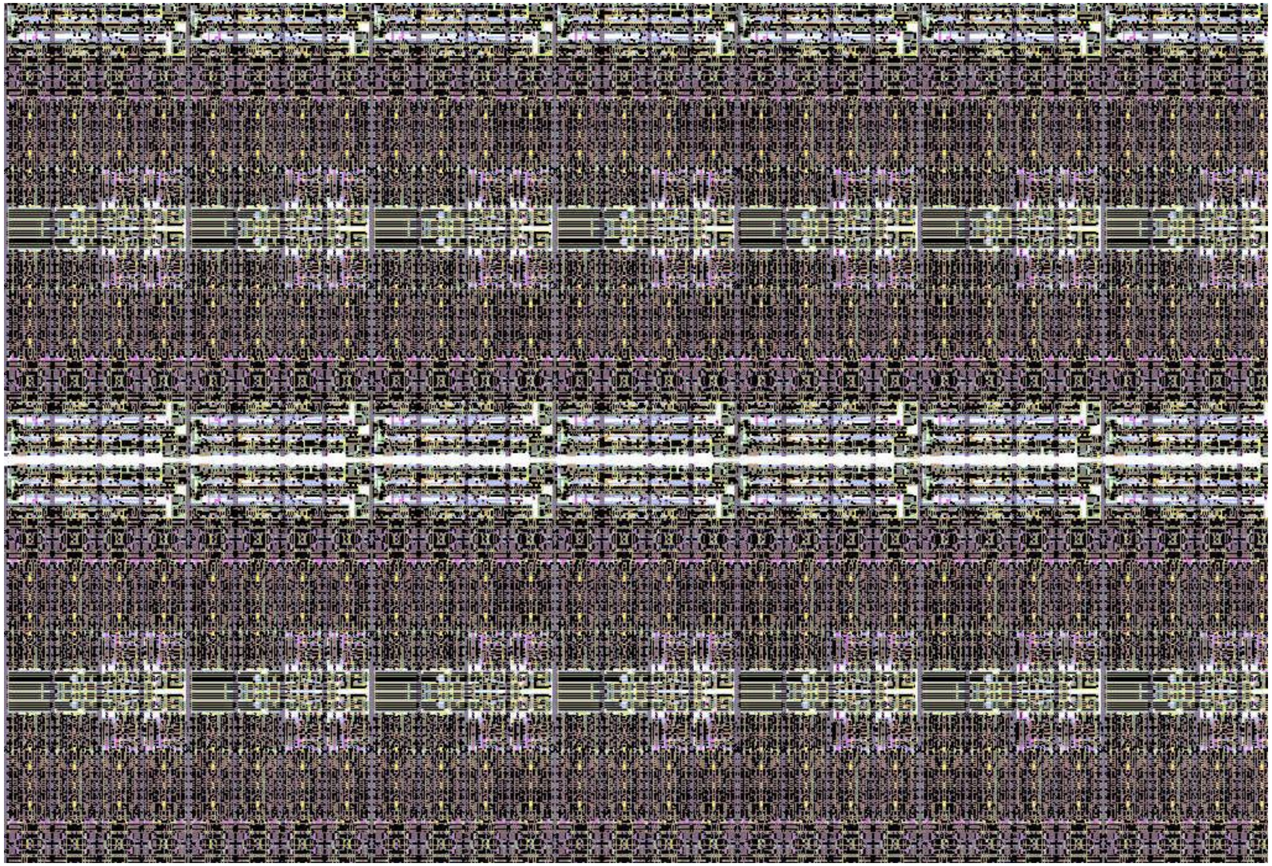




- ***example: Star-RCXT, Hercules***

7. Turn your design into sand

- Produce the layout data: GDSII
  - example: Astro



GDSII - text view

STRNAME EXAMPLE
BOUNDARY
LAYER
1
DATATYPE
0
XY
-10000
10000
20000
10000
20000
-10000
-10000
-10000
-10000
10000
ENDEL
ENDSTR

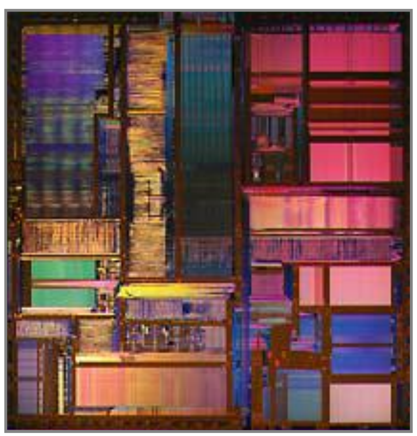
OR,

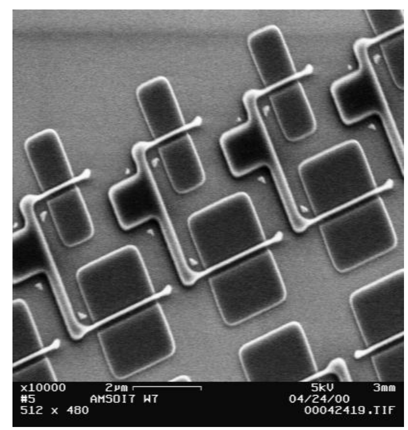
```
02000200 60000201 1C000300 02000600 ...........`.... 000000
01000E00 02000200 60002500 01000E00 .....%.`........ 000010
42494C45 4C504D41 58450602 12002500 .%....EXAMPLELIB 000020
413E0503 14000300 02220600 59524152 RARY.."......>A 000030
1C00545A 9BA02FB8 4439EFA7 C64B3789 .7K...9D./..ZT.. 00004
60000000 01000E00 02000200 60000205 ...`...........` 000050
58450606 0C001100 01000E00 02000200 ..............EX 000060
0100020D 06000008 04000045 4C504D41 AMPLE........... 000070
0000F0D8 FFFF0310 2C000000 020E0600 .......,........ 000080
FFFF204E 00001027 0000204E 00001027 '...N ..'...N .. 000090
0000F0D8 FFFFF0D8 FFFFF0D8 FFFFF0D8 ................ 0000A0
00000004 04000007 04000011 04001027 '............... 0000B0
00000000 00000000 00000000 00000000 ................ 0000C0
```

Files are usually 20 to 30 gigabytes in size, but after you correct the image, the size can reach a 150 gigabytes - that’s 150 billion bytes or over a trillion bits!

7. Turn your design into sand

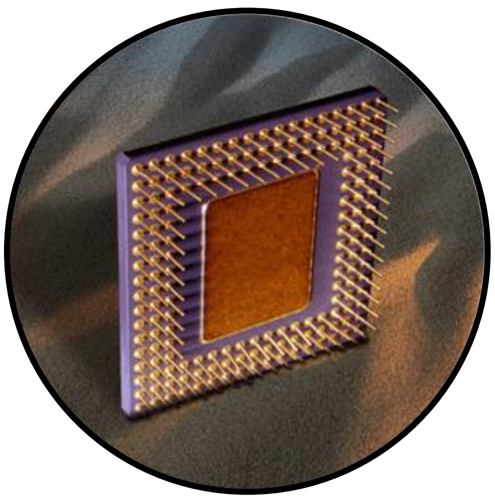
- Correct the image
  - example: Proteus, Progen, IC Workbench, SiVL/LRC, iN-Phase, CATS





7. Turn your design into sand

Take a deep breath and “Sign Off”!!



- Your favorite semiconductor vendor will:
  - Rerun your design through the tools
  - Generate a huge database for manufacturing
  - Perform the photolithography process
  - Put your chip into a package
  - Run your test program
  - Deliver your finished chips