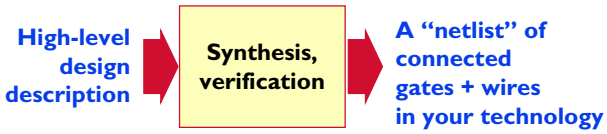
(*Lec 11*) From Logic…To *Layout*

- What you know...
  - Boolean, 1s and 0s stuff: synthesis, verification, representation
  - This is what happens in the "front end" of the ASIC design process



- What you don't know...
  - The "back end" problem -- turning these into IC masks
  - Basically a gates --> rectangles problem
  - Called "layout" or "physical design"

Copyright Notice

Where Are We?

- Physical design--how to geometrically *layout* gates in a netlist?

	M	T	W	Th	F	
Aug	27	28	29	30	31	1
Sep	3	4	5	6	7	2
	10	11	12	13	14	3
	17	18	19	20	21	4
	24	25	26	27	28	5
Oct	1	2	3	4	5	6
	8	9	10	11	12	7
Midsem break	15	16	17	18	19	8
	22	23	24	25	26	9
	29	30	31	1	2	10
Nov	5	6	7	8	9	11
	12	13	14	15	16	12
Thnxgive	19	20	21	22	23	13
	26	27	28	29	30	14
Dec	3	4	5	6	7	15
	10	11	12	13	14	16

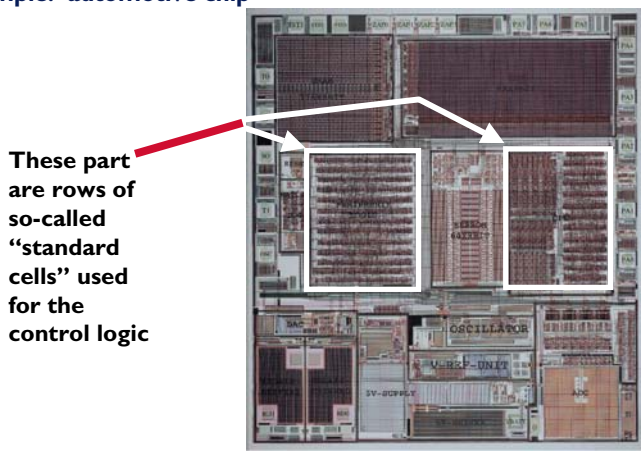
Introduction
Advanced Boolean algebra
JAVA Review
Formal verification
2-Level logic synthesis
Multi-level logic synthesis
Technology mapping
Placement
Routing
Static timing analysis
Electrical timing analysis
Geometric data structs & apps

Handouts

- Physical
  - Lecture 11 -- Logic to Layout
- Electronic
  - Nothing new...

Physical Design for ASICs

- We are focusing on a particular "niche" in layout
  - Row-based standard cells for semi-custom applications
  - Example: automotive chip

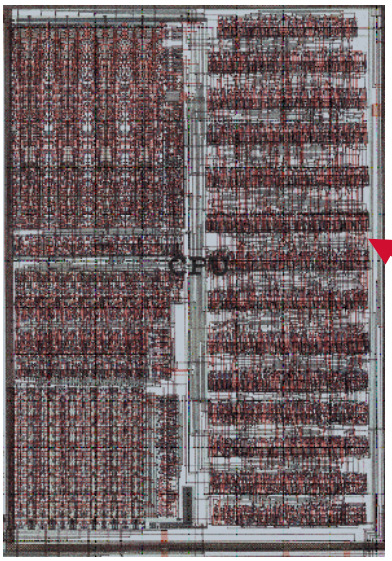


Zoom: Row-Based Control Logic

(BTW--this is an older technology, which is why you see explicit spaces for wires in between the rows…)

Zoom: CPU Is Itself a Hierarchy of Cells

More "regular" layout structures: ALU, registers



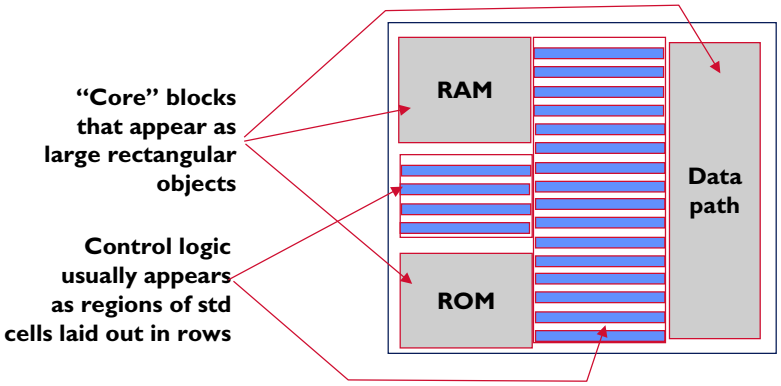
Row-based control logic

ASIC-related Terminology: Buzzword Lexicon

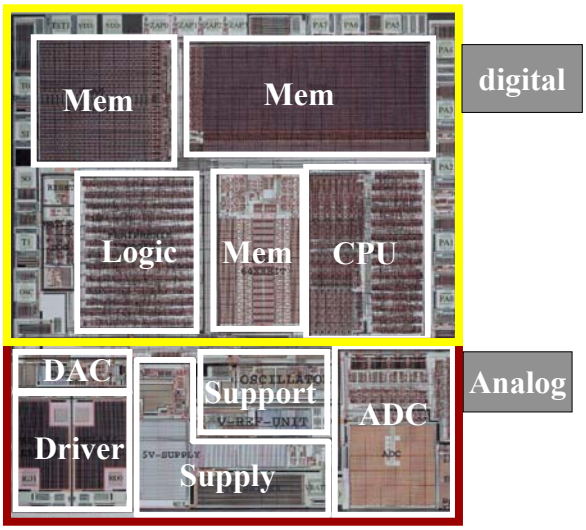
- Some terms
  - *Custom layout:* you do it all by hand. Think of this as design at the transistor level. Also called (derisively) "polygon pushing".
  - *Semi-custom layout:* you pick up some pre-laid-out pieces from some library and then arrange these to do your complete layout
  - *ASIC, Application-Specific Integrated Circuit:* a semi-custom IC designed to do one thing (eg, an MPEG decoder) as opposed to something arbitrarily programmable, like a CPU
  - *Standard cell:* the smallest, most common thing in your library of pre-laid-out stuff. Basically, a gate or FF. All the stuff in your technology library for Tech Mapping has a layout in this library.
  - *Core:* a huge block (cell) in your library of pre-laid-out stuff, like an entire CPU. More recently, called *intellectual property (IP)*
  - *SoC:* system-on-a-chip. Assemble a large chip from many pre-designed semi-custom pieces, like memories, CPU cores, random logic, etc

System On Chip Style

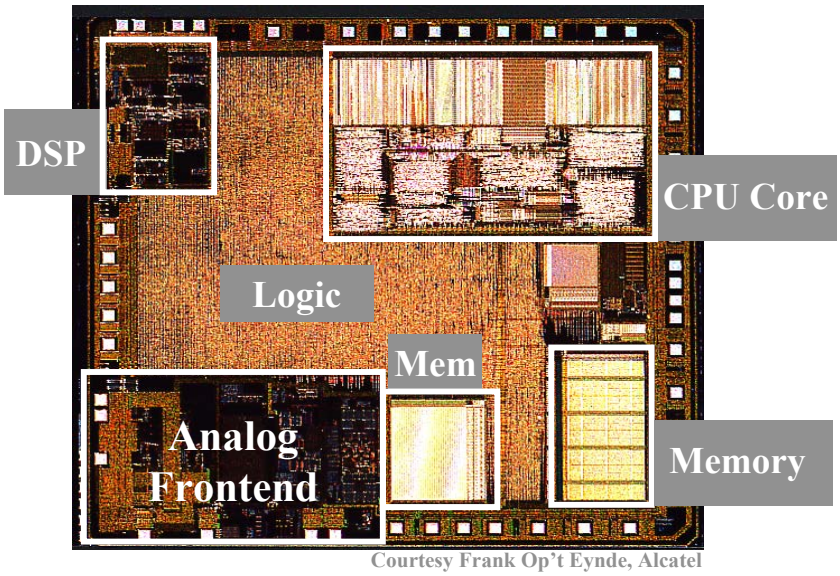
- Abstract example



Example Revisited: Automotive ASIC



A Larger Example: Telecom ASIC



Courtesy Frank Op't Eynde, Alcatel

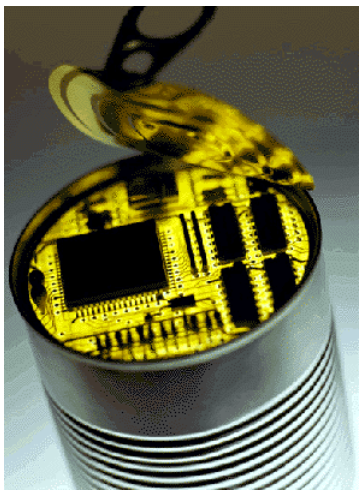
ASIC Design *Methodology*

- Methodology == ?
  - The sequence of CAD tools, design representations, design steps you use to go from an idea to silicon
  - Includes both synthesis steps, to make designs more detailed, and verification steps, to check correctness
- Modern methodology (simplified)
  - **Simulation /validation:** spec design as executable code (eg, Verilog), simulate to try test cases
  - **Synthesis:** high-level synthesis (eg, from Verilog) and logic synthesis, to go from high-level spec to gates
  - **Mapping:** to get the gates onto your implementable library
  - **Formal verification:** used where possible to prove equivalence
  - **Physical Design:** floorplan the chip, place the blocks/gates, route them
  - **Timing verification:** (in all steps) ensure meet timing goals (ie, MHz)

Aside: From Methodology to Intellectual Property

- Intellectual property (IP)
  - Something you can buy for $$$ that embodies some design expertise
  - Easier to buy it than to build it
- Historical forms of IP
  - **Chips**: You buy the hardware, plug it into a board, wire it up and go
  - **Software**: You buy it, load it, boot it, run it.
- Evolving forms of IP
  - **Soft IP:** buy the Verilog source code. Synthesize it yourself.
  - **Firm IP:** buy the gate/block level netlist. Do your own mapping, layout
  - **Hard IP:** buy the actual layout. May have to "adjust" to your fab
  - Ability to buy tools that do synthesis/layout is one *big* factor driving IP

IP -- the Ultimate Fantasy…



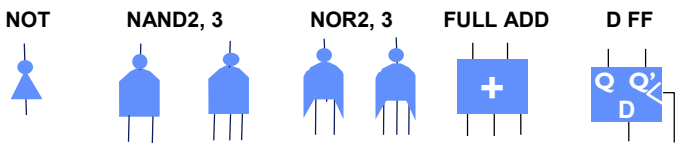
Electronic IP

What Are We Doing Now in 760?

- Big steps in physical design "backend" of ASIC methodology
- Synthesis steps
  - Placement & partitioning
  - Routing
- Verification steps
  - Static timing analysis & electrical timing analysis
  - Algorithms for representing/checking mask geometry
- (Aside: 18-360 vs 18-760:
  - 360: a little annealing, partitioning, maze routing
  - 760: a lot more annealing, 2 more placement algorithms, a new partitioning algorithm, a lot more routing, new stuff on timing and on geometric representation)

Physical Design for ASICs

- Example of (trivial) standard cell library



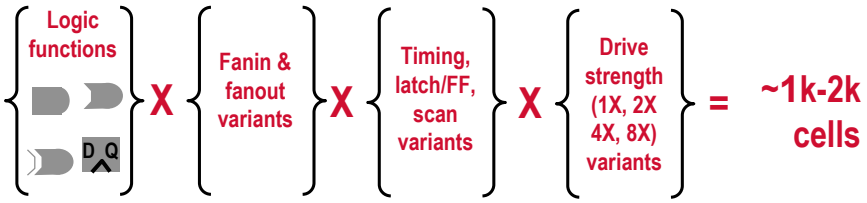
- As layouts...
  - Each a little rectangle of different mask layouts
  - Usually a common height (Y direction)
  - Varying widths, depending on number of IOs
  - Pins may be available at the top & bottom of cell, or just someplace inside the cell boundary, on metal layers "visible" to the router

3-input gate: pins available top & bottom

3-input gate: pins available in middle

How Big is a Cell Library--How Many Cells?

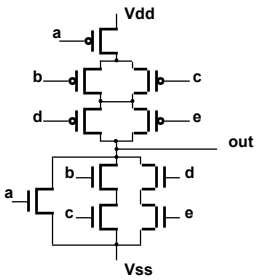
- Often, pretty big
  - Big enough to get all necessary logic functions, IO variants, timing variants, drive strengths, to first order

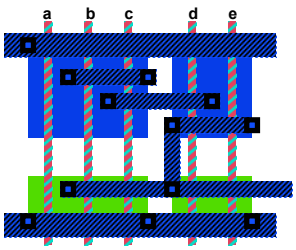


- Suggested way think about a standard cell
  - It's a nice, simple abstraction of a geometric "*container*" for the circuits you need to make logic. Its *hides* messy circuit details.
  - Inside the cell: complex device & mask & electrical issues
  - Outside the cell: a box with pins

Physical Design for ASICs

- A simple CMOS logic cell
  - For static CMOS, inputs on polysilicon, P devices on top, N on bottom

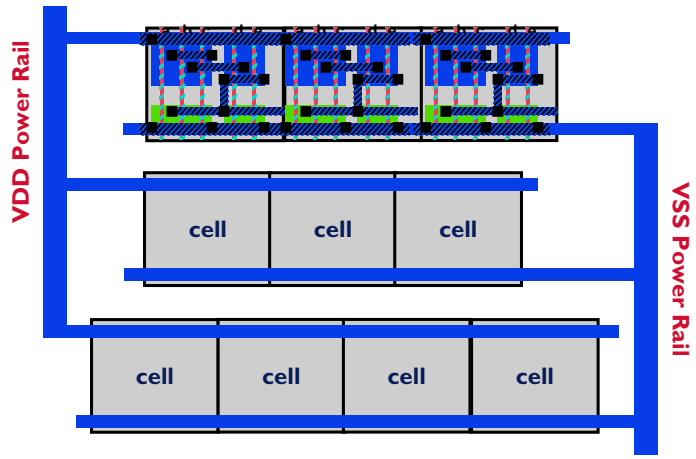




- Way think about a standard cell
  - It's a nice, simple geometric abstraction of a geometric "*container*" for the circuits you need to make logic. Its *hides* messy circuit details.

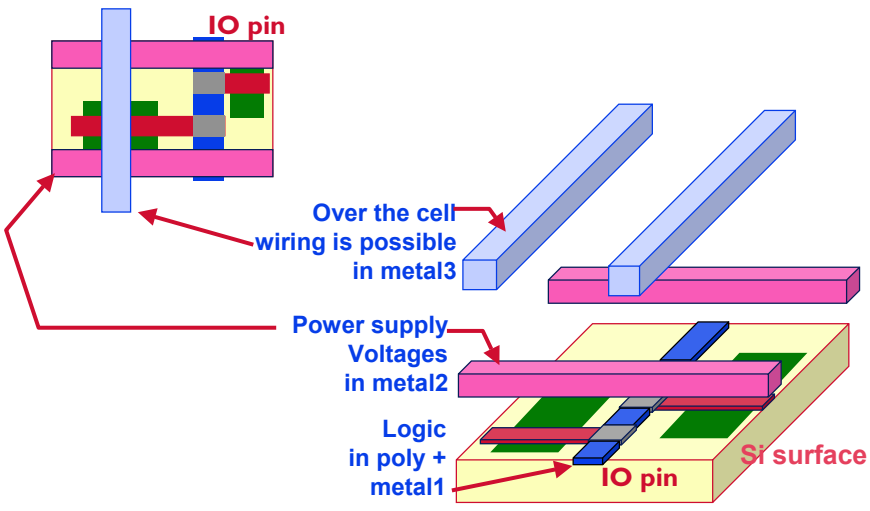
Row-Based Cell Layout for ASICs

- Cells "snap" together to connect power grid
  - Power wiring style called "interdigitated fingers" or "combs"



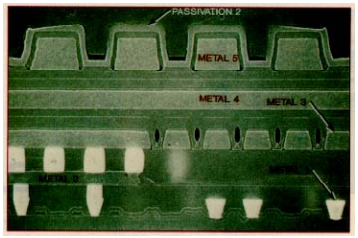
Impact of Available Metal Routing Layers

- Closer look at a more complicated, more modern cell style



About Metal Routing Layers

- As fab technology progresses, get more layers of metal wiring
  - Earliest (NMOS) technologies had only one metal, one poly for routing
  - Later we got 2 metal layers for routing.
  - Then 3 layers, 4 layers, 5 layers, 6 layers, 7 layers… today, ~8 layers
  - Strongly affects style of row-based ASIC layouts

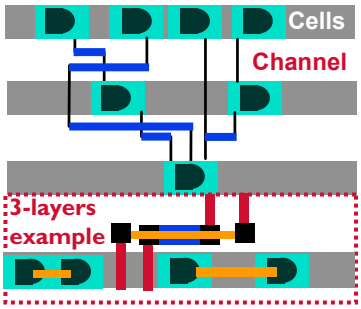


5-layer metal cross section
from early IBM PowerPC

ASIC Layout: 2-3 Routing Layers vs 4+ Layers

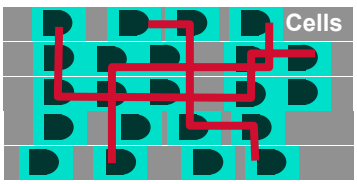
- 2-3 metal layers
  - 2-layers: 1 horizontal, 1 vertical;
    3-layers: typically 2 horiz, 1 vertical
  - We used to add spaces *inside* logic rows for vertical wires to cross

2,3-layers: Row-based Layout



- 4+ layers
  - Metals alternate horizontal, vertical, horizontal, etc
  - No channel spaces reserved at all, wires go right over the gates

4+ layers: Row-based Layout



Summary: Physical Design for ASICs

◥ What exactly are we going to look at?

1. Placement & partitioning
   - ▶ Once you know the gates for your chip, *where do you put them*?
2. Routing
   - ▶ Once you know where the gates are, how do you connect the wires?
3. Logical timing & electrical timing analysis
   - ▶ You have gates, have wires, have delay models: so how fast will it go?
4. "Big" geometry: representation & manipulation
   - ▶ How do you deal efficiently with 1,000,000,000 rectangles?

