

Advanced VLSI Design

Lecture 6: Circuit and System Representations

Shaahin Hessabi

Department of Computer Engineering

Sharif University of Technology

Adapted, with modifications from lecture notes
from Rutgers University

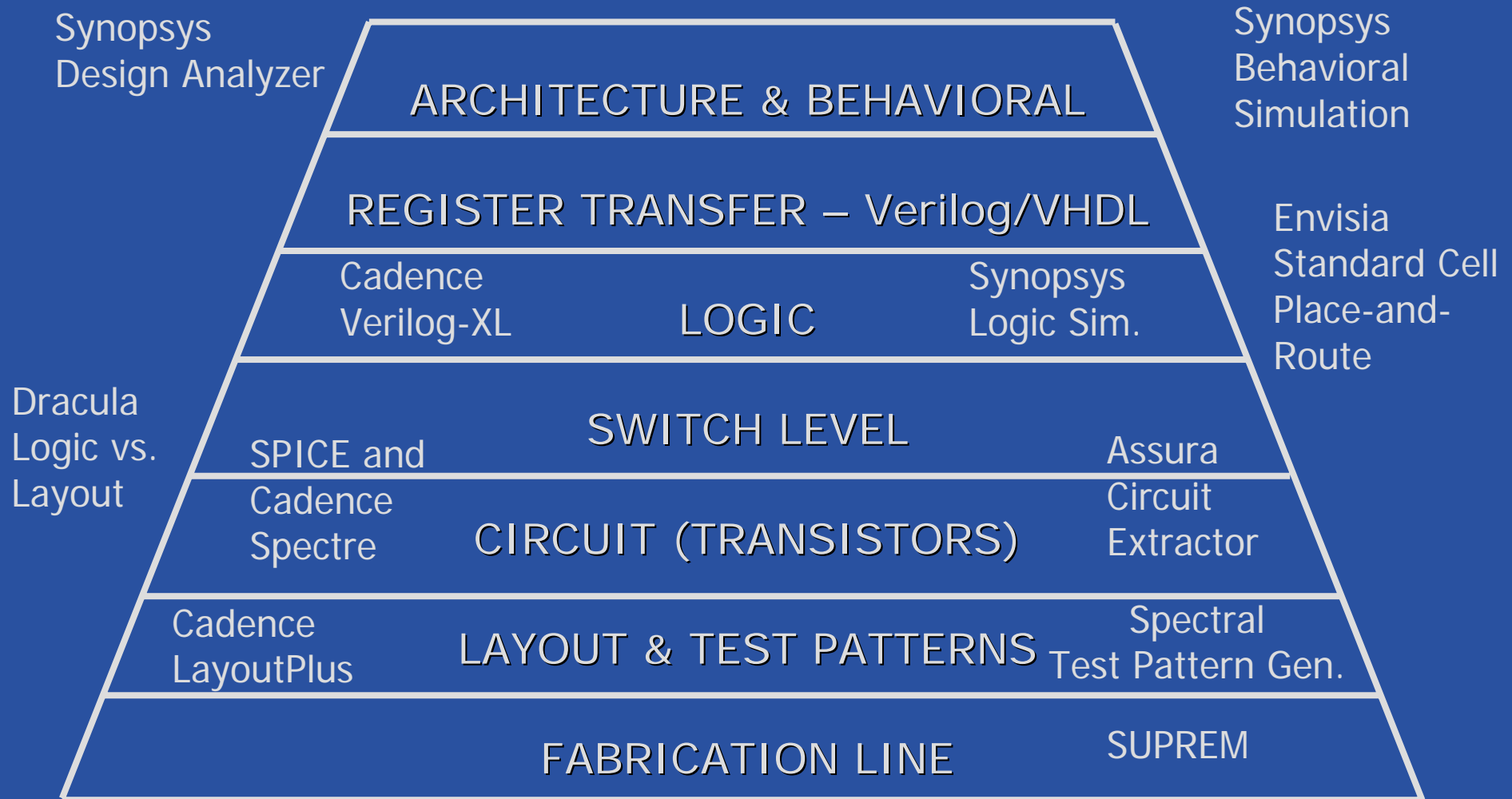
Outline

- ◆ Design Partitioning
- ◆ Abstraction Levels
 - Architecture
 - Microarchitecture
 - Logic Design
 - Circuit Design
 - Physical Design
- ◆ Fabrication, Packaging, Testing
- ◆ Summary

Coping with Complexity

- ◆ How to design System-on-Chip?
 - Many millions (soon billions!) of transistors
 - Tens to hundreds of engineers
- 1. Structured Design (discussed later)
- 2. Design Partitioning

Abstraction Levels

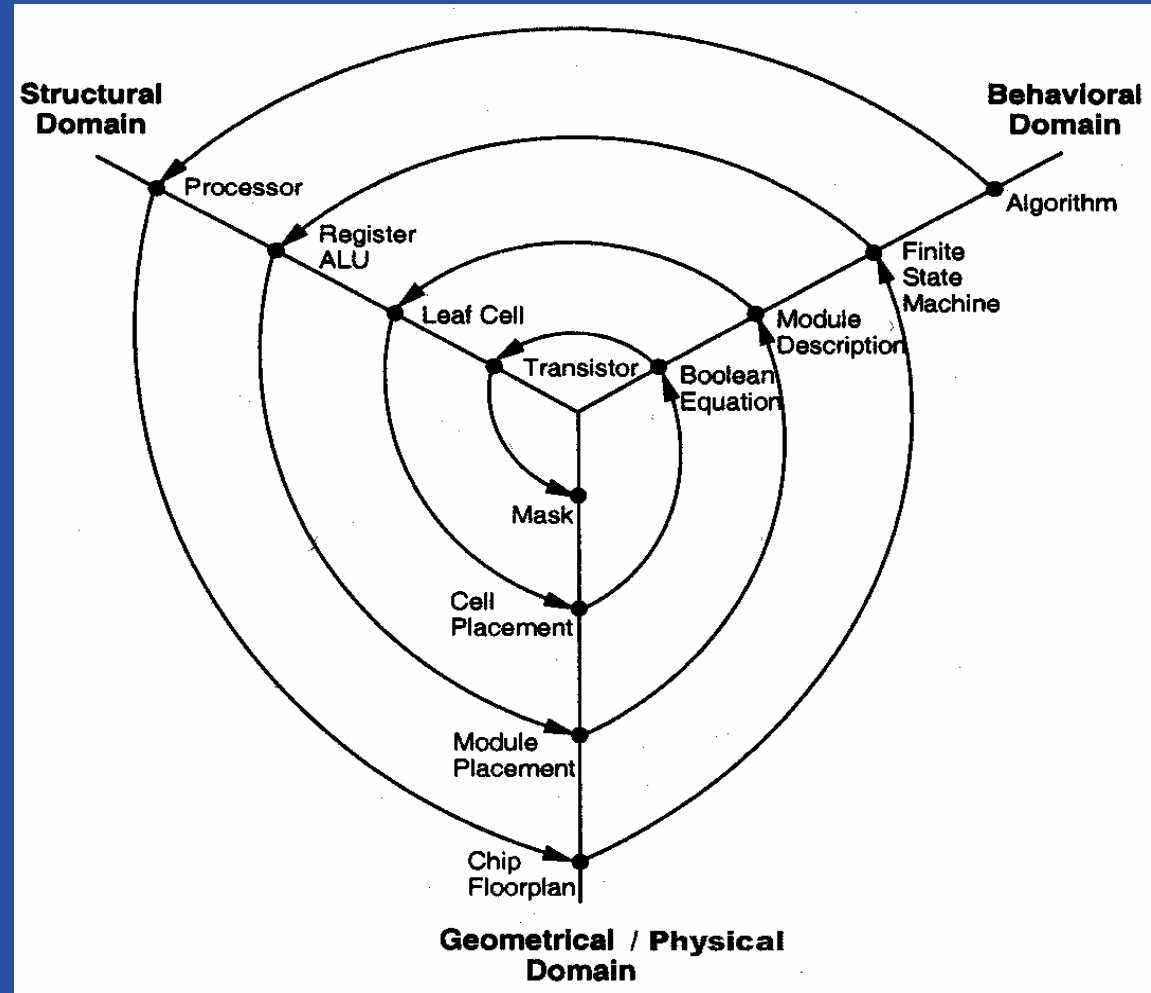


Design Partitioning

- ◆ **Architecture:** User's perspective, what does it do?
 - Instruction set, registers
 - MIPS, x86, Alpha, PIC, ARM, ...
- ◆ **Microarchitecture**
 - Single cycle, multi-cycle, pipelined, superscalar?
- ◆ **Logic:** how are functional blocks constructed
 - Ripple carry, carry look-ahead, carry select adders
- ◆ **Circuit:** how are transistors used
 - Complementary CMOS, pass transistors, domino
- ◆ **Physical:** chip layout
 - Datapaths, memories, random logic

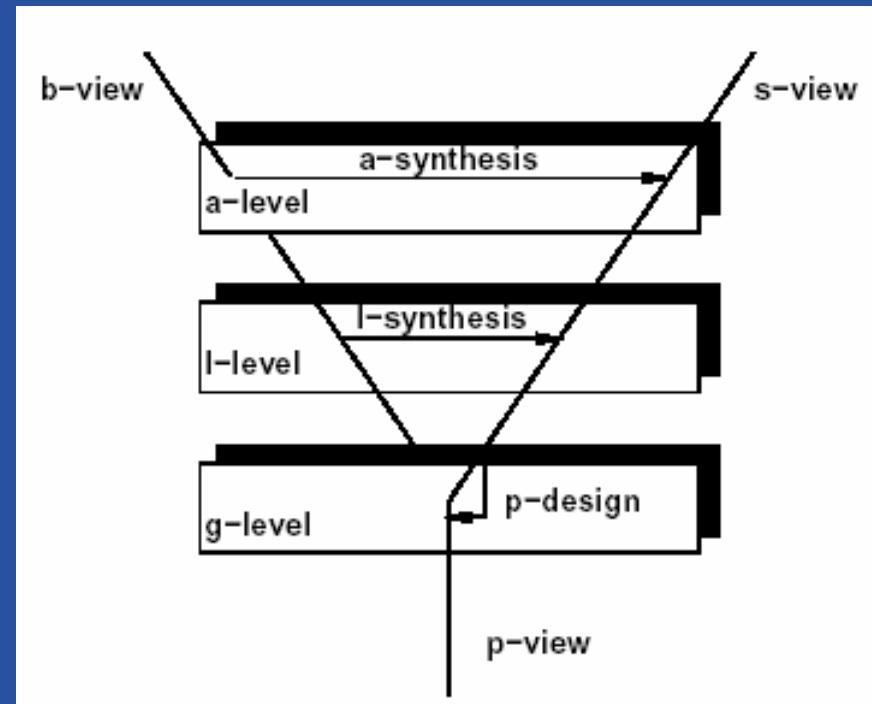
Gajski Y-Chart

- ◆ Behavioral view:
 - Abstract function
- ◆ Structural view:
 - Interconnection of parts
- ◆ Physical view:
 - Physical objects with size and positions



Y-Chart (cont'd)

- ◆ Correspondence with:
 - Synthesis
 - Analysis
 - Optimization
 - Refinement
 - Physical design
 - Extraction



HDLs

- ◆ Hardware Description Languages
 - Widely used in logic design
 - Verilog and VHDL
- ◆ Describe hardware using code
 - Document logic functions
 - Simulate logic before building
 - Synthesize code into gates and layout
 - ◆ Requires a library of standard cells

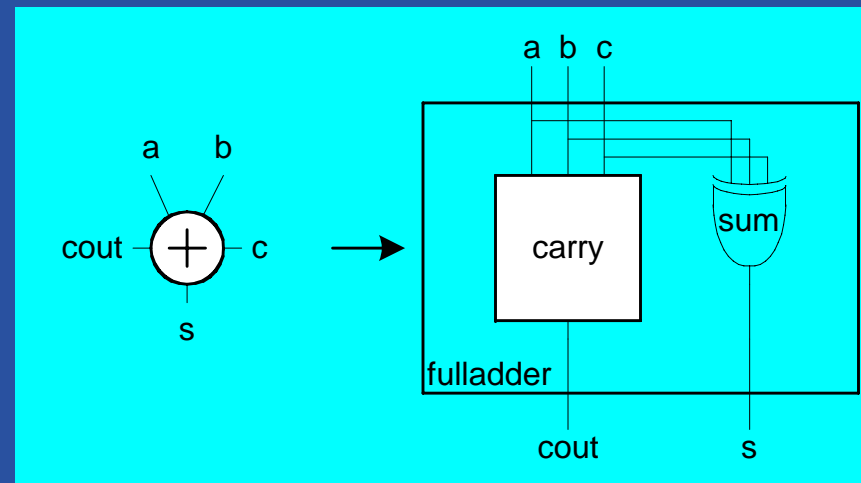
Verilog Example

```
module fulladder(input a, b,c,  
                output s, cout);
```

```
    sum    s1(a, b, c, s);  
    carry  c1(a, b, c, cout);  
endmodule
```

```
module carry(input  a, b, c,  
            output cout)
```

```
    assign cout=(a&b) | (a&c) | (b&c);  
endmodule
```

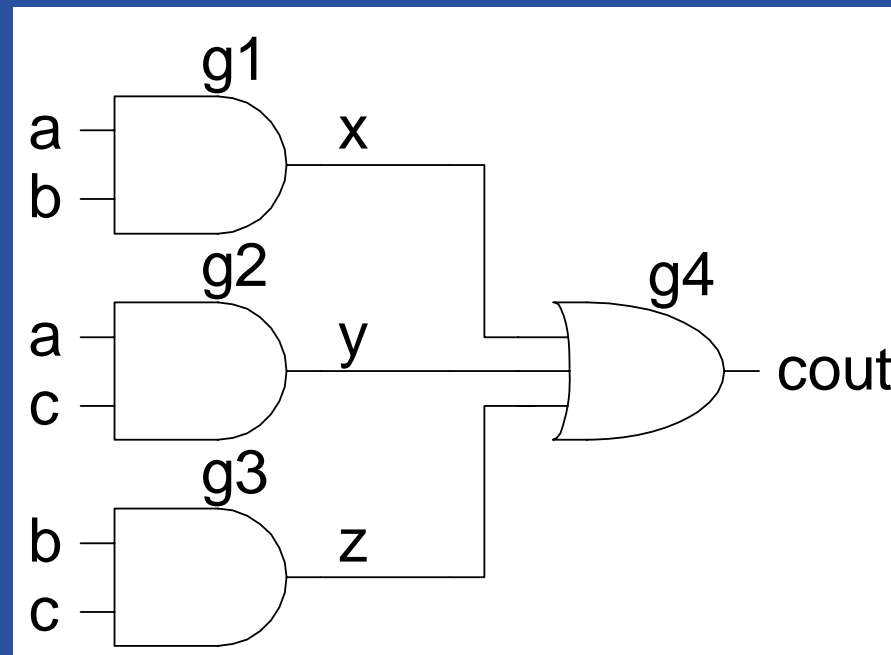


Circuit Design

- ◆ How should logic be implemented?
 - NANDs and NORs vs. ANDs and ORs?
 - Fan-in and fan-out?
 - How wide should transistors be?
- ◆ These choices affect speed, area, power
- ◆ Logic synthesis makes these choices for you
 - Good enough for many applications
 - Hand-crafted circuits are still better

Example: Carry Logic

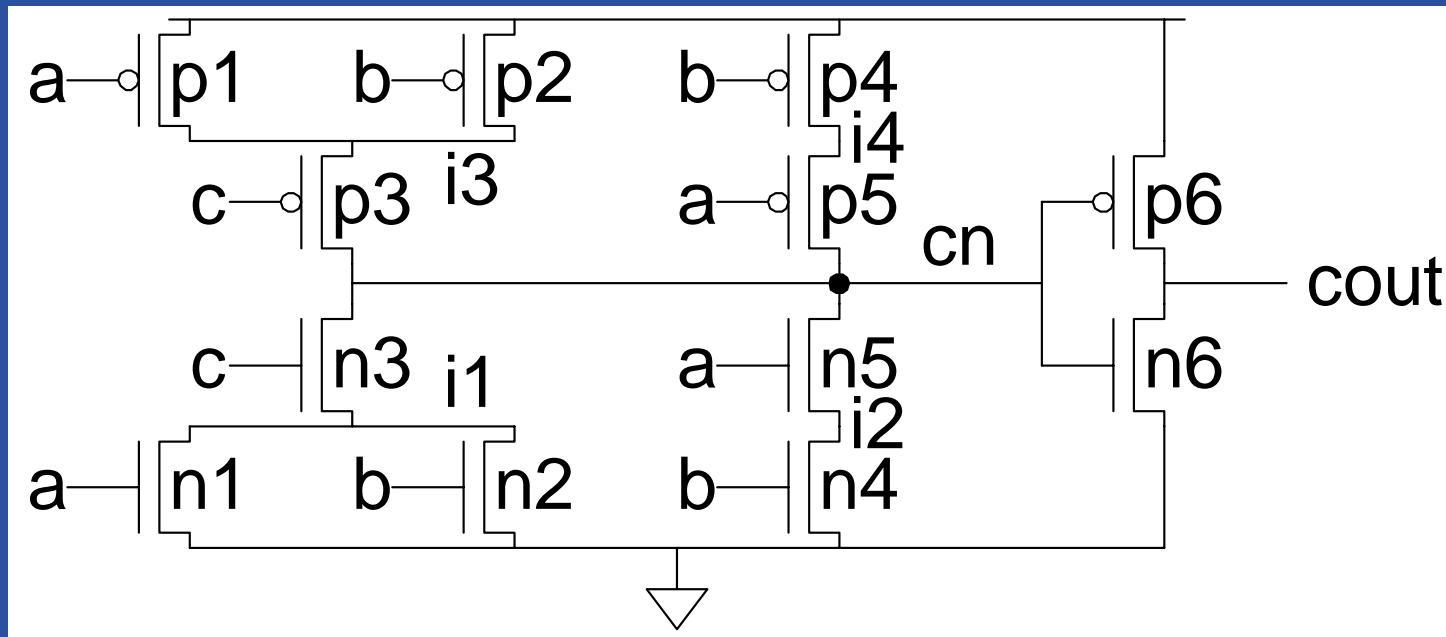
◆ `assign cout = (a&b) | (a&c) | (b&c);`



Transistors? Gate Delays?

Example: Carry Logic

◆ `assign cout = (a&b) | (a&c) | (b&c);`



Transistors? Gate Delays?

Gate-Level Netlist

```
module carry(input a, b, c,  
             output cout)
```

```
    wire x, y, z;
```

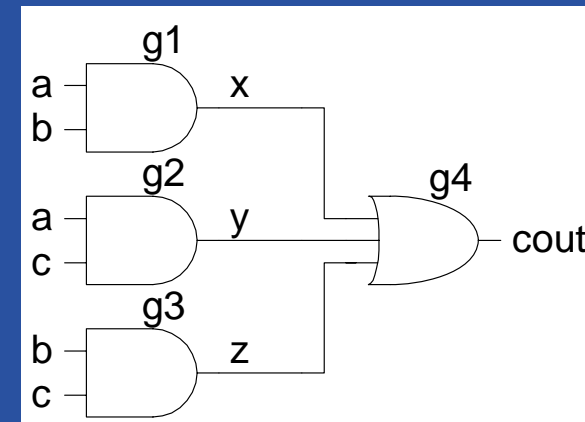
```
    and g1(x, a, b);
```

```
    and g2(y, a, c);
```

```
    and g3(z, b, c);
```

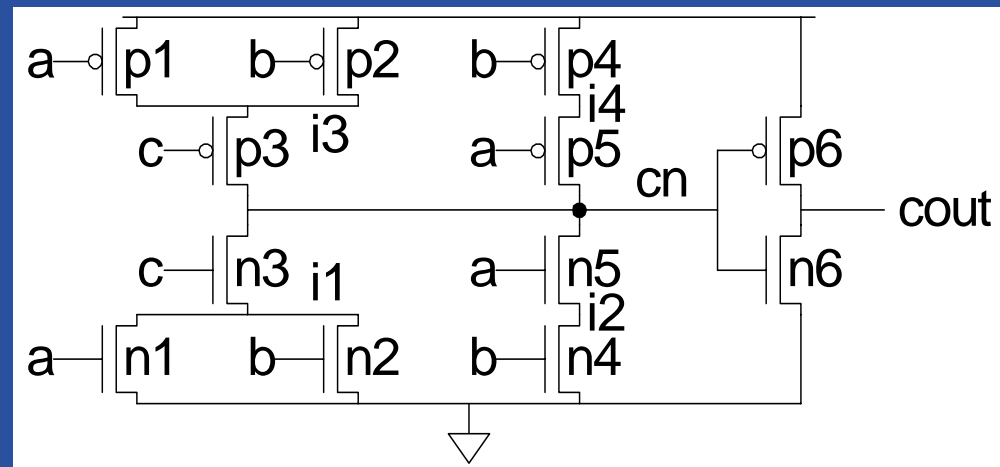
```
    or g4(cout, x, y, z);
```

```
endmodule
```



Transistor-Level Netlist

```
module carry(input  a, b, c,  
             output cout)  
  
    wire      i1, i2, i3, i4, cn;  
  
    tranif1 n1(i1, 0, a);  
    tranif1 n2(i1, 0, b);  
    tranif1 n3(cn, i1, c);  
    tranif1 n4(i2, 0, b);  
    tranif1 n5(cn, i2, a);  
    tranif0 p1(i3, 1, a);  
    tranif0 p2(i3, 1, b);  
    tranif0 p3(cn, i3, c);  
    tranif0 p4(i4, 1, b);  
    tranif0 p5(cn, i4, a);  
    tranif1 n6(cout, 0, cn);  
    tranif0 p6(cout, 1, cn);  
  
endmodule
```



SPICE Netlist

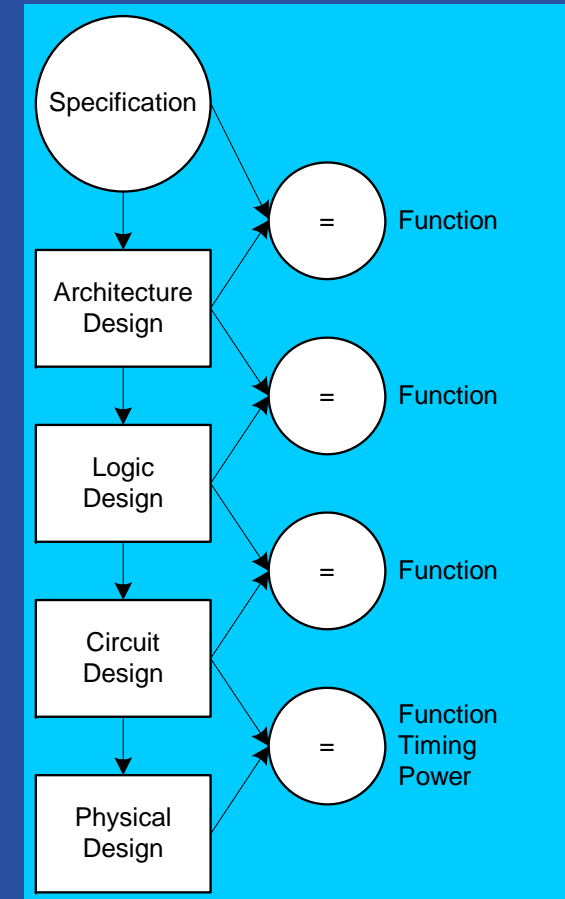
```
.SUBCKT CARRY A B C COUT VDD GND
MN1 I1 A GND GND NMOS W=1U L=0.18U AD=0.3P AS=0.5P
MN2 I1 B GND GND NMOS W=1U L=0.18U AD=0.3P AS=0.5P
MN3 CN C I1 GND NMOS W=1U L=0.18U AD=0.5P AS=0.5P
MN4 I2 B GND GND NMOS W=1U L=0.18U AD=0.15P AS=0.5P
MN5 CN A I2 GND NMOS W=1U L=0.18U AD=0.5P AS=0.15P
MP1 I3 A VDD VDD PMOS W=2U L=0.18U AD=0.6P AS=1 P
MP2 I3 B VDD VDD PMOS W=2U L=0.18U AD=0.6P AS=1P
MP3 CN C I3 VDD PMOS W=2U L=0.18U AD=1P AS=1P
MP4 I4 B VDD VDD PMOS W=2U L=0.18U AD=0.3P AS=1P
MP5 CN A I4 VDD PMOS W=2U L=0.18U AD=1P AS=0.3P
MN6 COUT CN GND GND NMOS W=2U L=0.18U AD=1P AS=1P
MP6 COUT CN VDD VDD PMOS W=4U L=0.18U AD=2P AS=2P
CI1 I1 GND 2FF
CI3 I3 GND 3FF
CA A GND 4FF
CB B GND 4FF
CC C GND 2FF
CCN CN GND 4FF
CCOUT COUT GND 2FF
.ENDS
```

Physical Design

- ◆ Floorplan
- ◆ Standard cells
 - Place & route
- ◆ Datapaths
 - Slice planning
- ◆ Area estimation

Design Verification

- ◆ Fabrication is slow & expensive
 - 180 nm: \$1 M
 - 65 nm: \$30 M
 - 45 nm: \$45 M
- ◆ Debugging chips is very hard
 - Limited visibility into operation
- ◆ Prove design is right before building!
 - Logic simulation
 - Ckt. simulation / formal verification
 - Layout vs. schematic comparison
 - Design & electrical rule checks
- ◆ Verification is > 50% of effort on most chips!

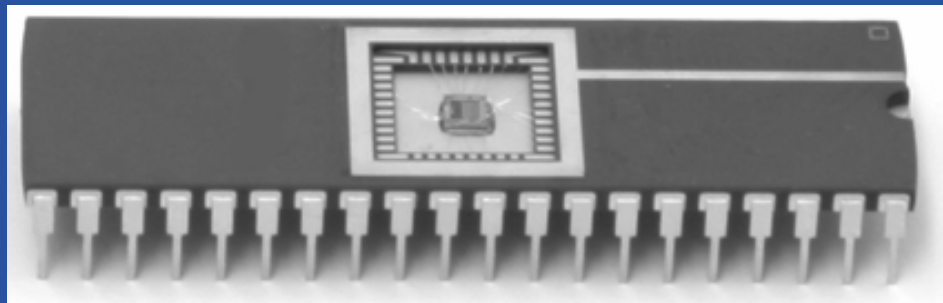


Technology CAD

- ◆ Design Rule Checker
- ◆ Electrical Rule Checker
 - Circuit Extraction from layout
 - ◆ Logic vs. layout check

Fabrication & Packaging

- ◆ Tapeout final layout
- ◆ Fabrication
 - 6, 8, 12" wafers
 - Optimized for throughput, not latency (10 weeks!)
 - Cut into individual dice
- ◆ Packaging
 - Bond gold wires from die I/O pads to package



Testing

- ◆ Test that chip operates
 - Design errors
 - Manufacturing errors
- ◆ A single dust particle or wafer defect kills a die
 - Yields from 90% to $< 10\%$
 - Depends on die size, maturity of process
 - Test each part before shipping to customer