EECS 151/251A
Spring 2019
Digital Design and Integrated Circuits

Instructor:
J. Wawrzynek

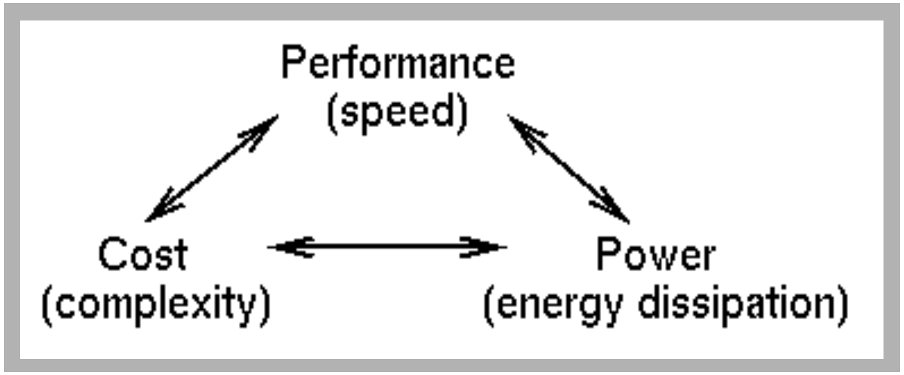
Lecture 2

Outline

- Methodology Basics
- Digital Logic – Basic Concepts
- Early Design
- Design Implementation Alternatives
- Design Flows
- ASICs

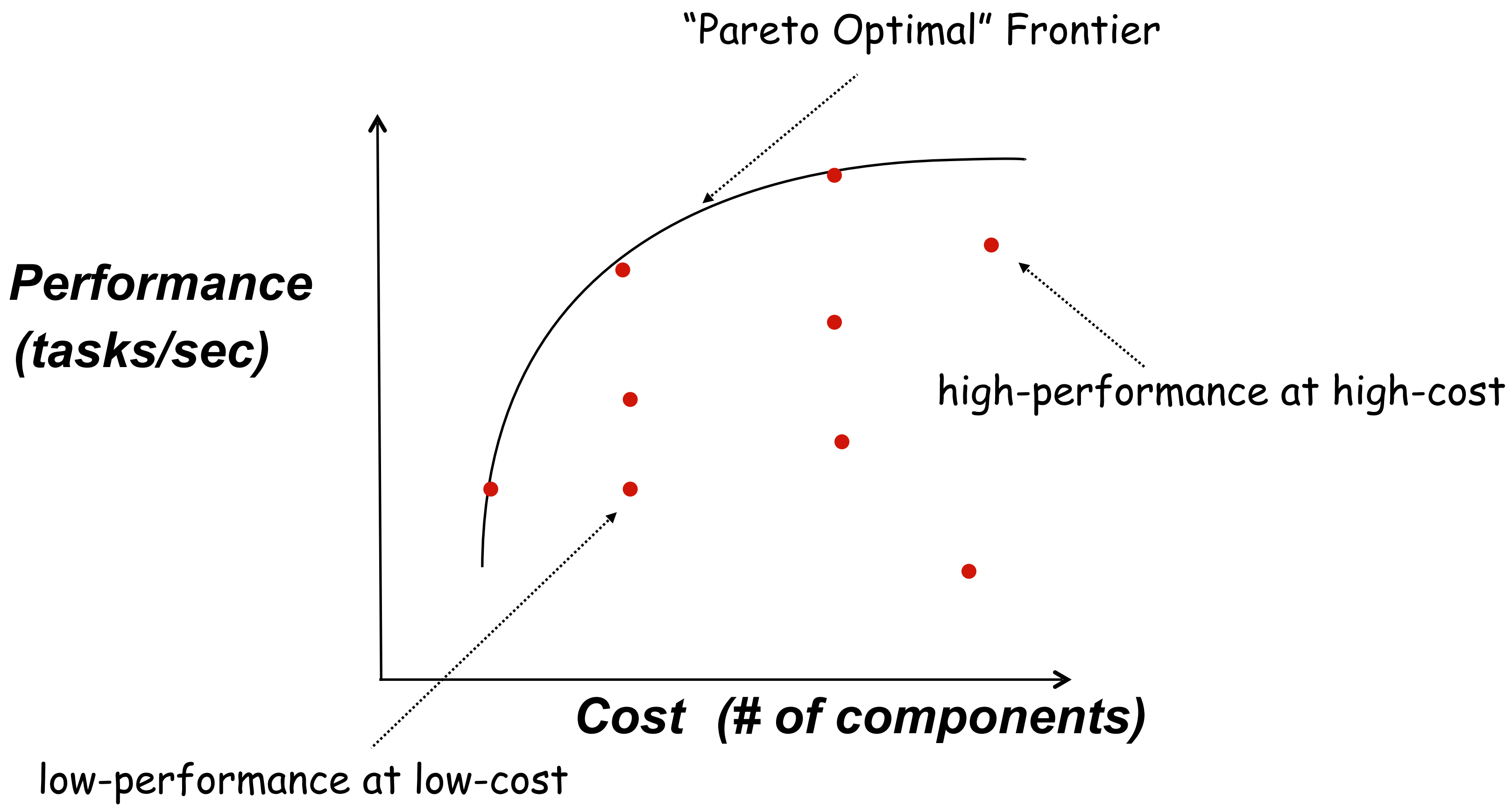
Methodology Basics

Basic Design Tradeoffs



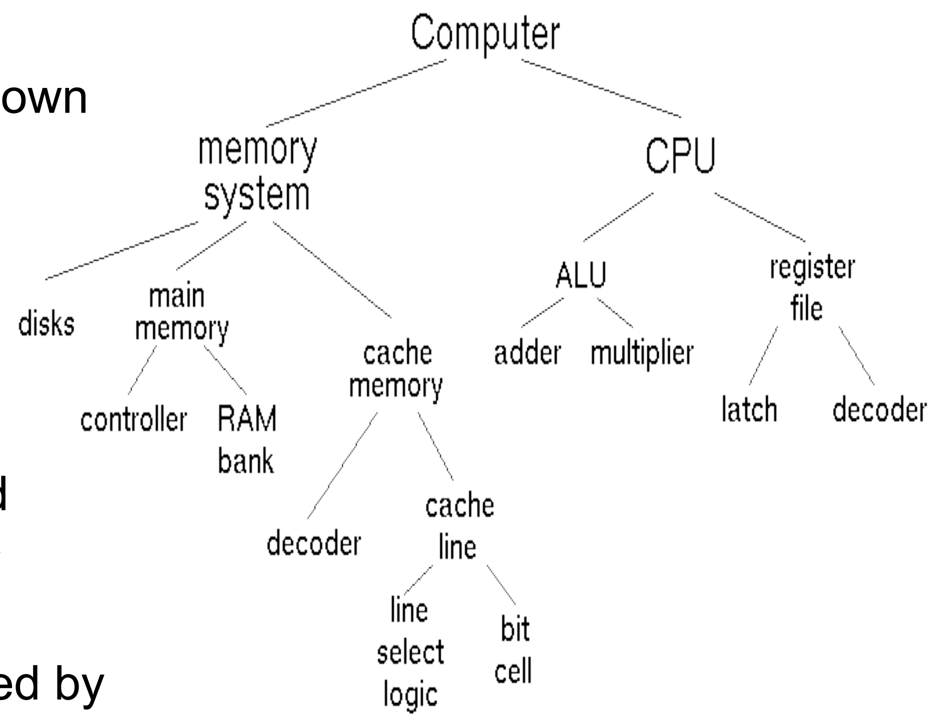
- Improve on one at the expense of the others
- Tradeoffs exist at every level in the system design
- Design Specification
  - Functional Description
  - Performance, cost, power constraints
- Designer must make the tradeoffs needed to achieve the function within the constraints

Design Space & Optimality



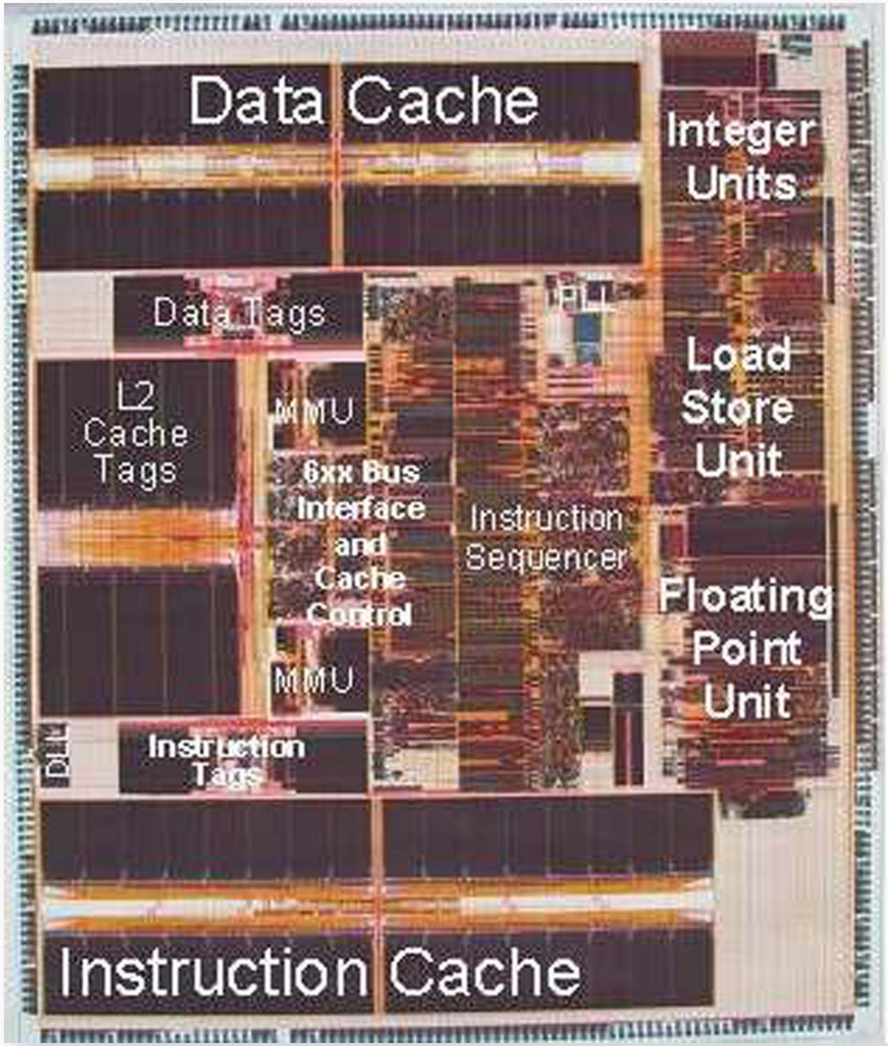
Design Methodologies

- Top-Down Design
  - Starts at the top (root) and works down by successive refinement.
- Bottom-up Design
  - Starts at the leaves & puts pieces together to build up the design.
- Which is better?
  - In practice both are needed & used
  - Top-down to handle the complexity (divide and conquer)
  - Bottom-up since structure influenced by available primitives (in a well designed system)



Digital Logic
Basic Concepts

Digital Integrated Circuit Example



- (Old) PowerPC microprocessor micro-photograph
  - Superscalar (3 instructions/cycle)
  - 6 execution units (2 integer and 1 double precision IEEE floating point)
  - 32 KByte Instruction and Data L1 caches
  - Dual Memory Management Units (MMU)
  - External L2 Cache interface with integrated controller and cache tags.

Comprises only transistors and wires.

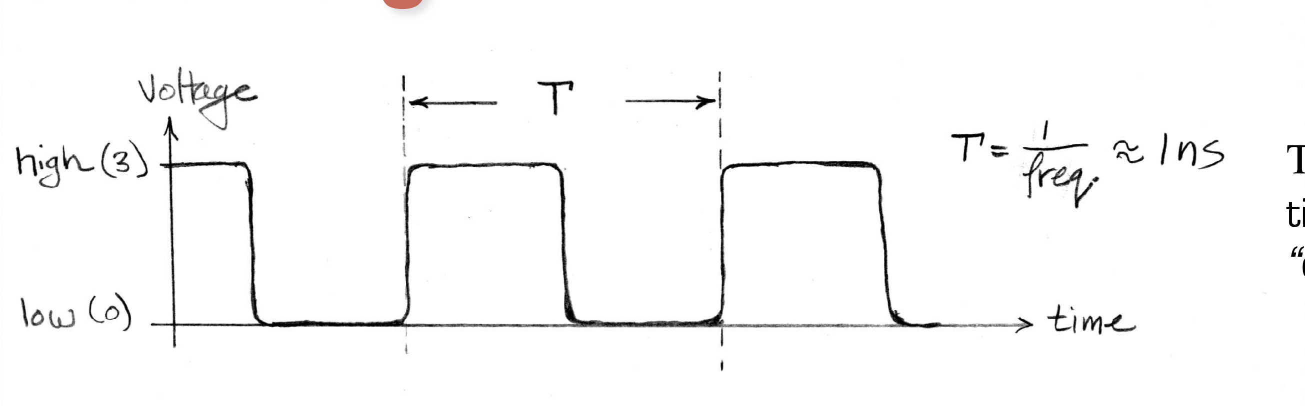
Connections to outside world (ex. motherboard)

Memory interface

Power (Vdd, GND)

Clock input

Clock Signal



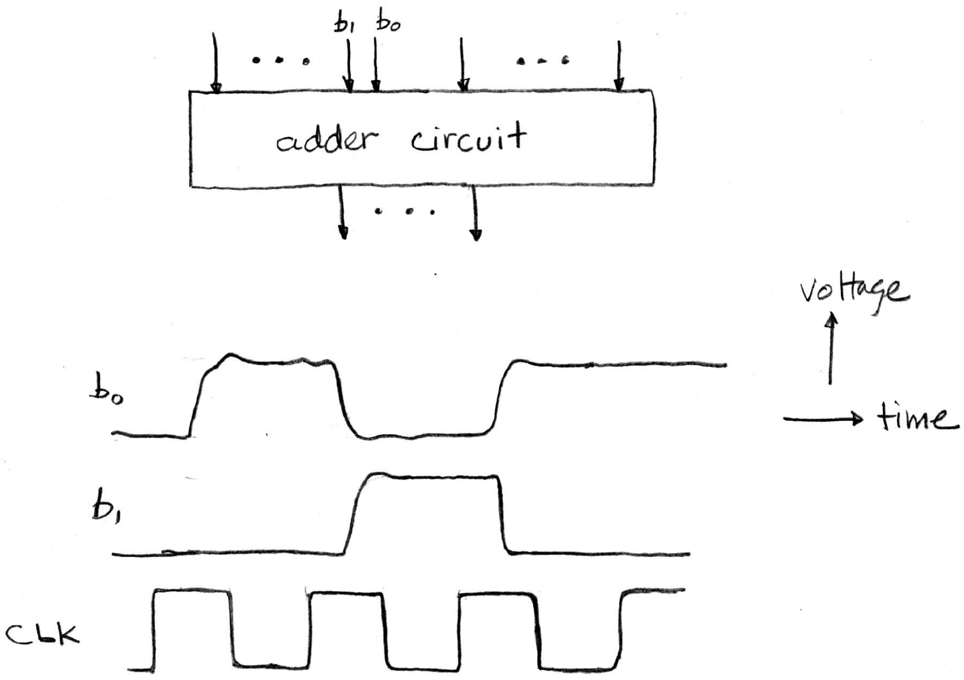
T represents the time of one clock "cycle".

A source of regularly occurring pulses used to measure the passage of time.

- Waveform diagram shows evolution of signal value (in voltage) over time.

- Usually comes from an off-chip crystal-controlled oscillator.
- One main clock per chip/system.
- Distributed throughout the chip/system.
- "Heartbeat" of the system. Controls the rate of computation by directly controlling all data transfers.

Data Signals



Random adder circuit at a random point in time:

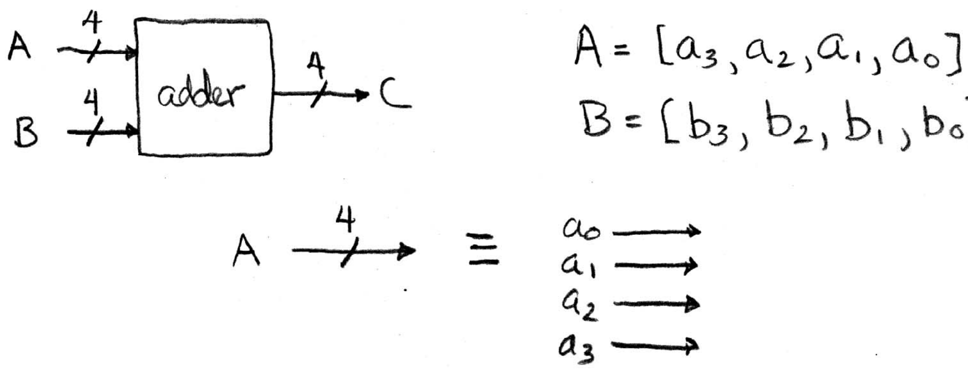
Observations:

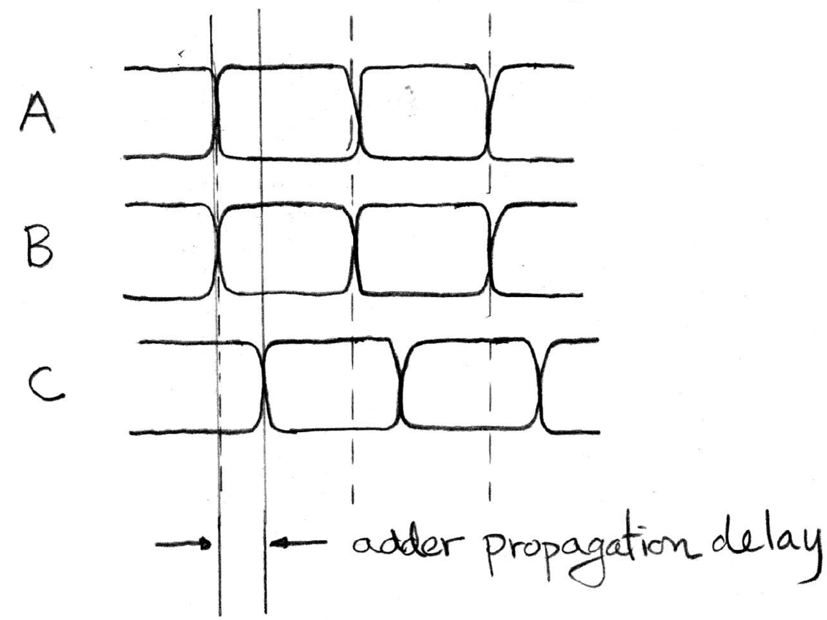
- Most of the time, signals are in either low- or high-voltage position.
- When the signals are at the high- or low-voltage positions, they are not all the way to the voltage extremes (or they are past).
- Changes in the signals correspond to changes in clock signal (but don't change every cycle).

The facts:

1. Low-voltage represents binary 0 and high-voltage, binary 1.
2. Circuits are designed and built to be tolerant of noise and "restoring". Deviations from ideal voltages are ignored. Outputs close to ideal.
3. In synchronous systems, all changes follow clock edges.

Circuit Delay



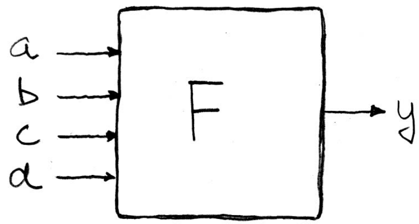


Digital circuits cannot produce outputs instantaneously.

- ❑ In general, the delay through a circuit is called the propagation delay. It measures the time from when inputs arrive until the outputs change.
- ❑ The delay amount is a function of many things. Some out of the control of the circuit designer:
  - Processing technology, the particular input values.
- ❑ And others under her control:
  - Circuit structure, physical layout parameters.

Combinational Logic Blocks

Example four-input function:



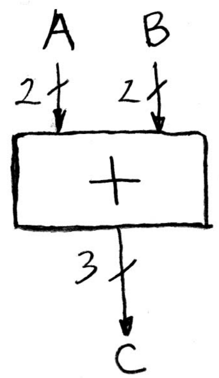
- Output a function only of the current inputs (no history).
- Truth-table representation of function. Output is explicitly specified for each input combination.
- In general, CL blocks have more than one output signal, in which case, the truth-table will have multiple output columns.

a b c d	y
0 0 0 0	F(0,0,0,0)
0 0 0 1	F(0,0,0,1)
0 0 1 0	F(0,0,1,0)
0 0 1 1	F(0,0,1,1)
0 1 0 0	F(0,1,0,0)
0 1 0 1	F(0,1,0,1)
0 1 1 0	F(0,1,1,0)
1 1 1 1	F(0,1,1,1)
1 0 0 0	F(1,0,0,0)
1 0 0 1	F(1,0,0,1)
1 0 1 0	F(1,0,1,0)
1 0 1 1	F(1,0,1,1)
1 1 0 0	F(1,1,0,0)
1 1 0 1	F(1,1,0,1)
1 1 1 0	F(1,1,1,0)
1 1 1 1	F(1,1,1,1)

Truth Table

Example CL Block

- 2-bit adder. Takes two 2-bit integers and produces 3-bit result.



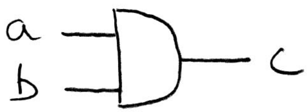
- Think about truth table for 32-bit adder. It's possible to write out, but it might take a while!

a1 a0	b1 b0	c2 c1 c0
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

Theorem: Any combinational logic function can be implemented as a networks of logic gates.

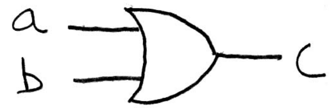
Logic Gates

AND



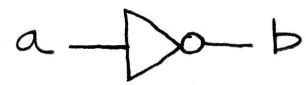
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

OR



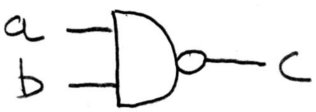
a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

NOT



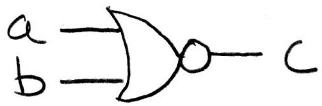
a	b
0	1
1	0

NAND



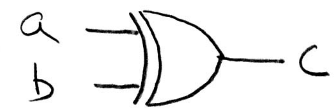
a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

NOR



a	b	c
0	0	1
0	1	0
1	0	0
1	1	0

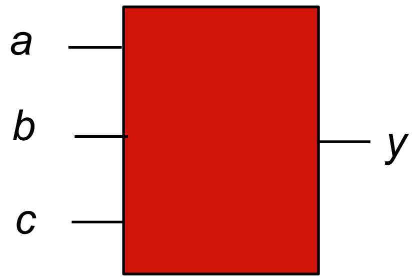
XOR



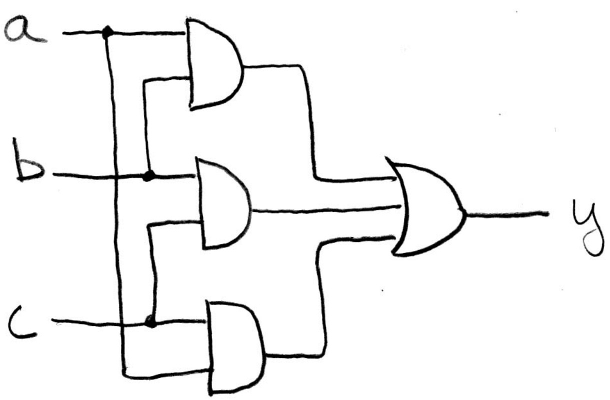
a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

- Logic gates are often the primitive elements out of which combinational logic circuits are constructed.
  - In some technologies, there is a one-to-one correspondence between logic gate representations and actual circuits (ASIC standard cells have gate implementations).
  - Other times, we use them just as another abstraction layer (FPGAs have no real logic gates).
- How about these gates with more than 2 inputs?
- Do we need all these types?

Example Logic Circuit



a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

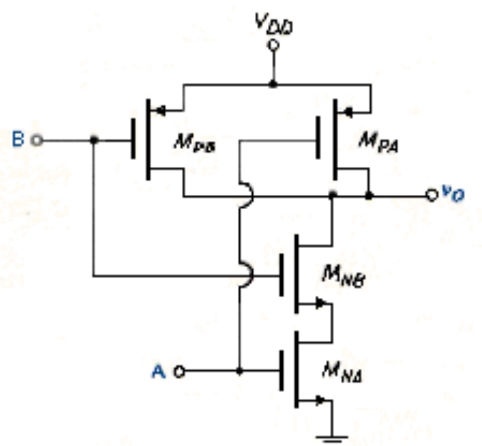


How do we know that these two representations are equivalent?

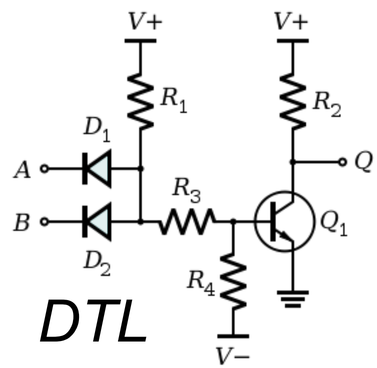
Will come back to this later!

Logic Gate Implementation

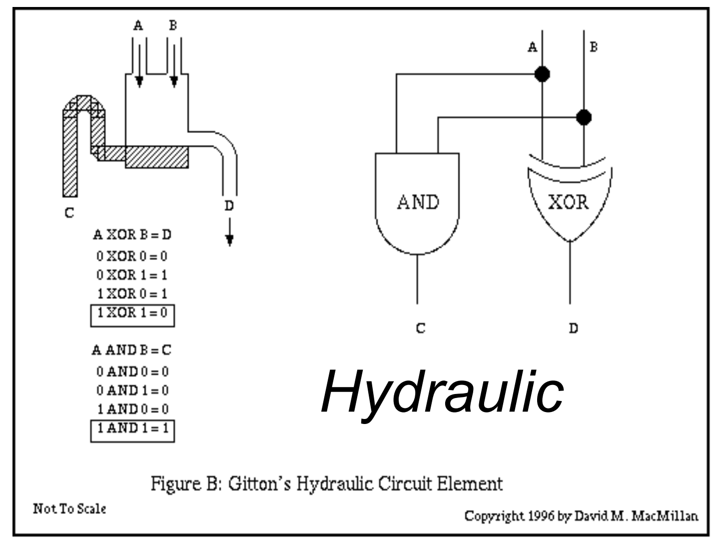
- Logic circuits have been built out of many different technologies. If we have a basic logic gate (AND or OR) and inversion we can build a complete logic family.

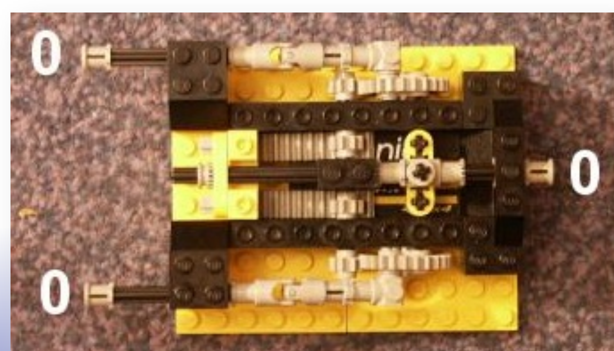


CMOS Gate



DTL

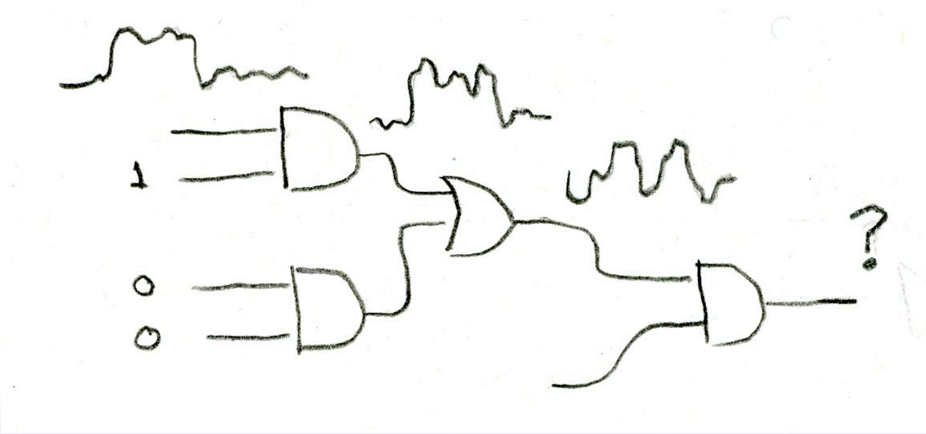




***Mechanical** LEGO logic gates. A clockwise rotation represents a binary "one" while a counter-clockwise rotation represents a binary "zero."*

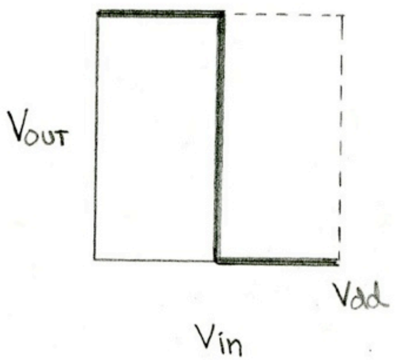
Restoration/Regeneration

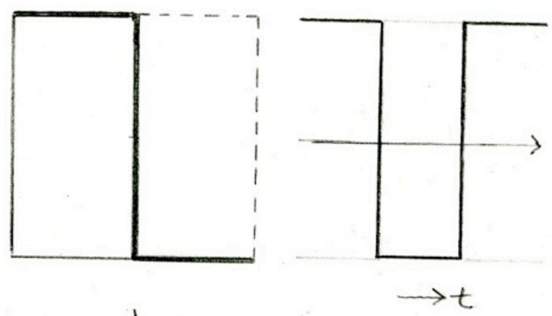
- A necessary property of any suitable technology for logic circuits is "Restoration" or "Regeneration"
- Circuits need:
  - to ignore noise and other non-idealities at the their inputs, and
  - generate "cleaned-up" signals at their output.
- Otherwise, each stage propagates input noise to their output and eventually noise and other non-idealities would accumulate and signal content would be lost.

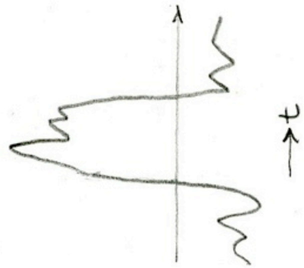


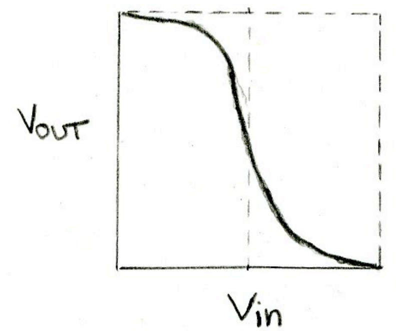
Inverter Example of Restoration

Example (look at 1-input gate, to keep it simple):



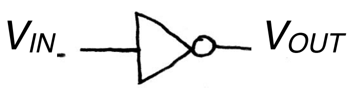






Actual Inverter

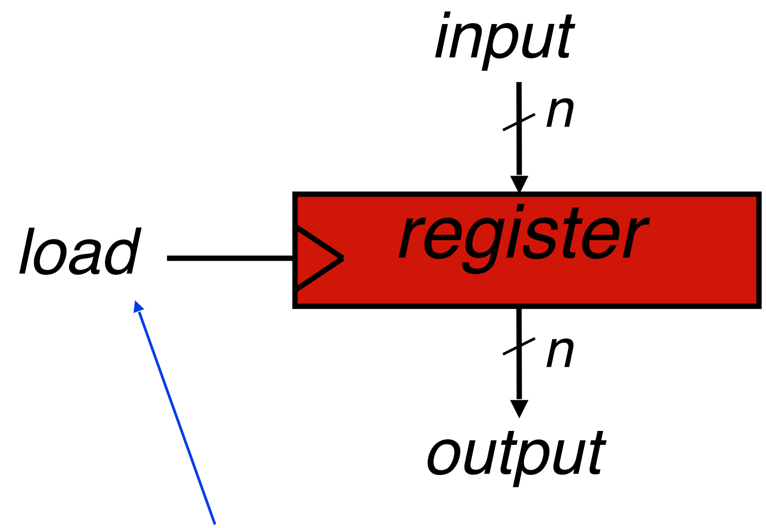
Idealize Inverter



- Inverter acts like a “non-linear” amplifier
- The non-linearity is critical to restoration
- Other logic gates act similarly with respect to input/output relationship.

State Elements: circuits that store info

- Examples: registers, memories
- Register: Under the control of the "load" signal, the register captures the input value and stores it indefinitely.



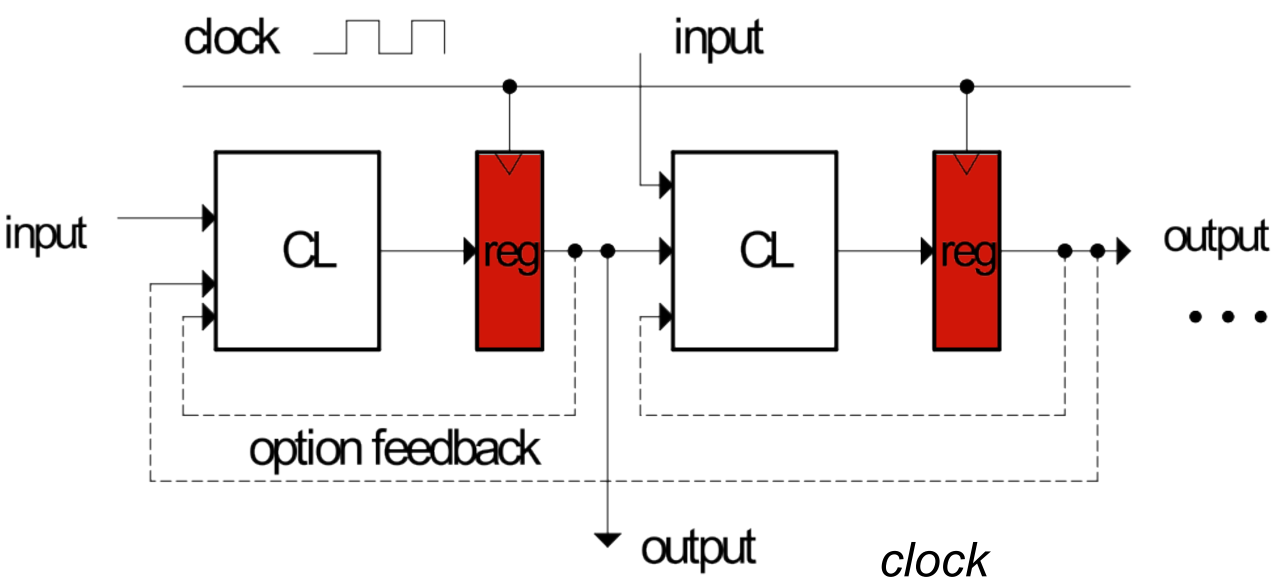
often replace by clock signal (clk)

- The value stored by the register appears on the output (after a small delay).
- Until the next load, changes on the data input are ignored (unlike CL, where input changes change output).
- These get used for short term storage (ex: register file), and to help move coordinate data movement.

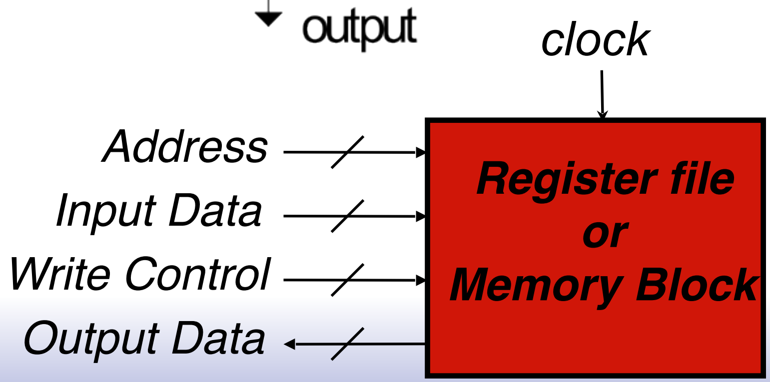
Register Transfer Level Abstraction (RTL)

Any synchronous digital circuit can be represented with:

- Combinational Logic Blocks (CL), plus
- State Elements (registers or memories)



- State elements are mixed in with CL blocks to control the flow of data.

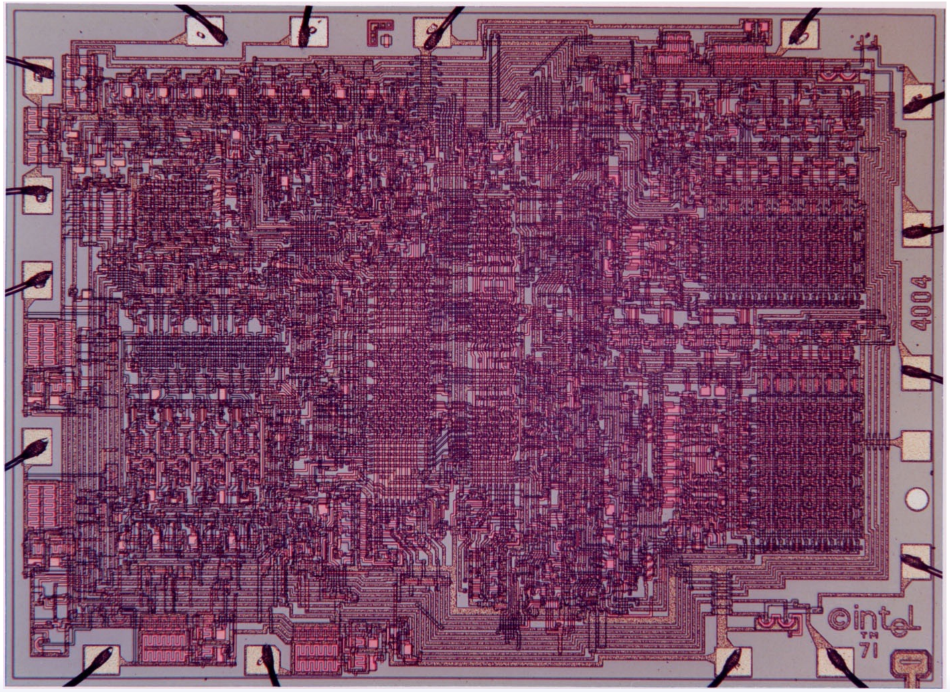


- Sometimes used in large groups by themselves for "long-term" data storage.

Early Design

IC Design in the 70's and early 80's

- Circuit design, layout, and processing tightly linked.
- Logic design and layout was all done by-hand in an ad-hoc way
- Chip design was the domain of industry (Fairchild, Intel, Texas Instruments, …). These were IC processing companies. Those who controlled the physics controlled the creative agenda!



Federico Faggin,
Ted Hoff,
Stan Mazor

Introduced to help sell memory chips!

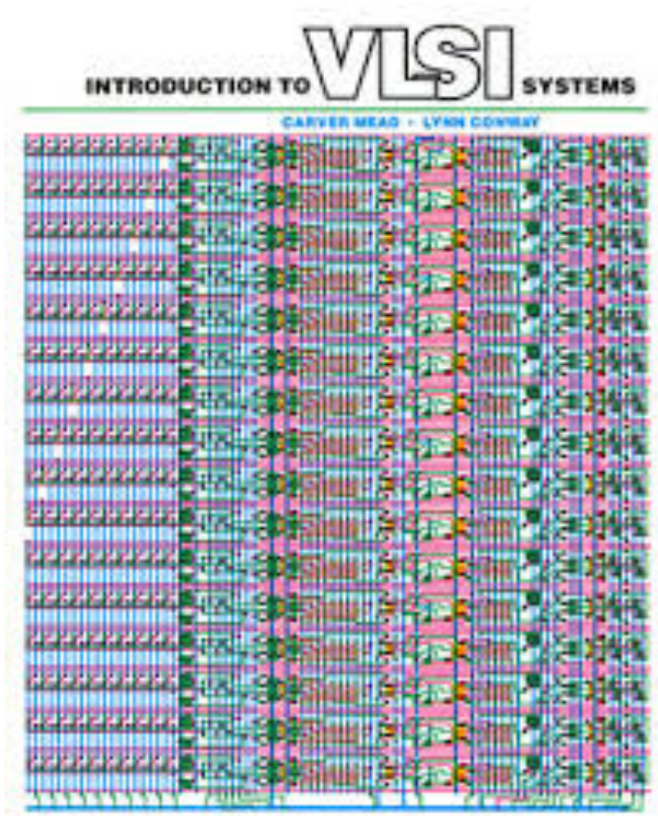
The Intel 4004 microprocessor, which was introduced in 1971. The 4004 contained 2300 transistors and performed 60,000 calculations per second. Courtesy: Intel.

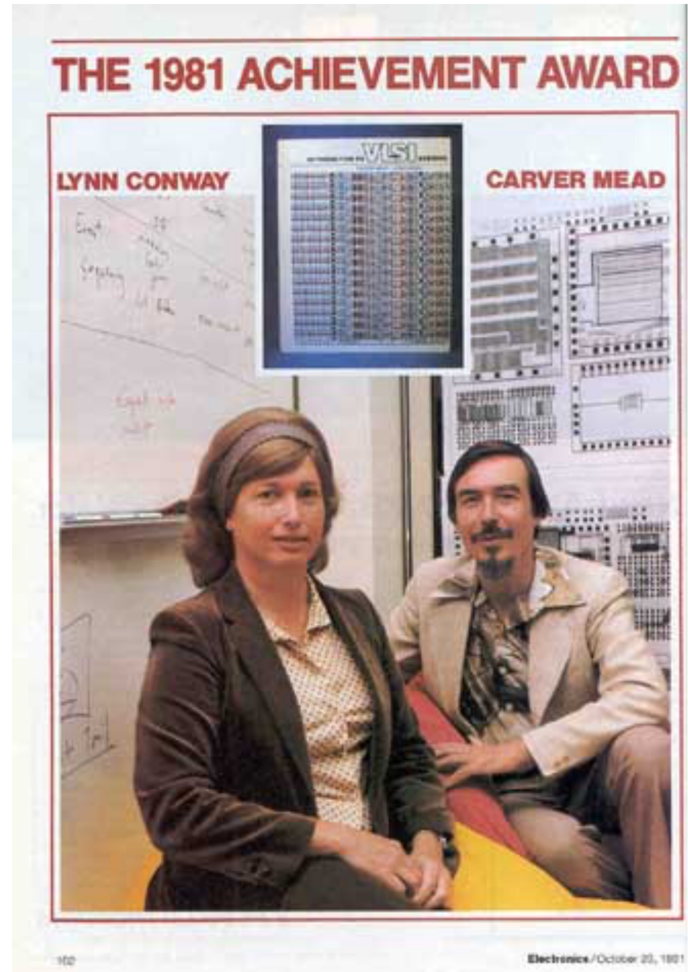
Early Design Practice

- Initially, designs were represented by hand drawings. Then masks where made by transferring drawings to rubylith.
  - Base layer of heavy transparent dimensionally stable Mylar. A thin film of deep red cellophane-like material covers the base layer. Patterns formed by cutting (often by hand) the transparent covering.

- *Later transition to an electronic format (CIF, GDS) meant: Layouts easily be stored and transmitted. Written to tape and transferred to manufacturer* (tape-out)*. Transmitted over the network (new idea back then). Software could automatically check for layout errors. Generated from a program -* ***huge idea.***

The start of the IC Design Revolution





Geometric Design Rules

- Early on, to generate the mask information for fabrication, the designer needed intimate knowledge of the manufacturing process. Even once this knowledge was distilled to a set of “Geometric Design Rules”, this set of rules was voluminous with many special cases.
- Academics (C. Mead and others) came up with a much simplified set of design rules (single page description). A sort of “API” or abstraction of the process (back-end processing could automatically convert this information into masks).

  - *Sufficiently small set that designers could memorize. Sufficiently abstract to allow process engineers to shrink the process and preserve existing layouts. Process resolution becomes a “parameter”,* ***λ****.*

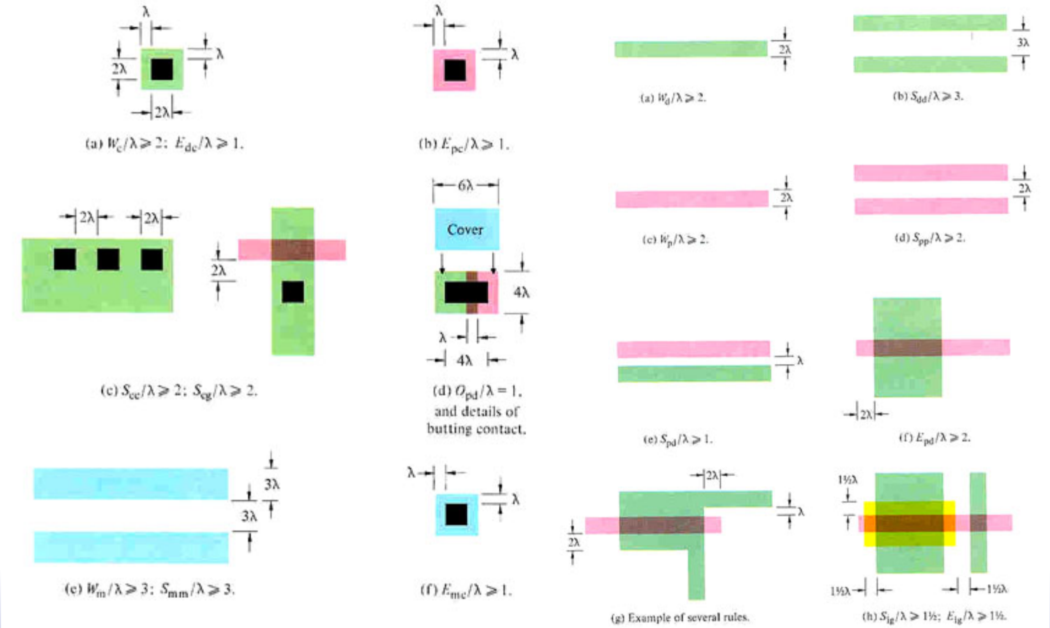


PLATE 3 nMOS design rules (continued)

PLATE 2 nMOS design rules

Key Development: Silicon Foundries

- Separate the designer from the fabricator: Modeled after the printing industry. (Very few authors actually own and run printing presses!)
- Simple standard geometric design rules where the key: these form the “contract” between the designer and manufacturer.
- Designer sends the layout (in CIF format), foundry manufactures the chip and send back. Designer promises not to violate the design rules. Foundry promises to accurately follow layout.

- *A scalable model for the industry:*
  IC fab is expensive and complex. Amortizes the expense over many designers (batch processing with deep queues help). Designers and companies not held back by need to develop and maintain large expensive factories.“fabless” semiconductor companies - lots of these and very few foundries.

TSMC, Global Foundries, UMC, Samsung, SMIC, …

Computer Aided Design (1)

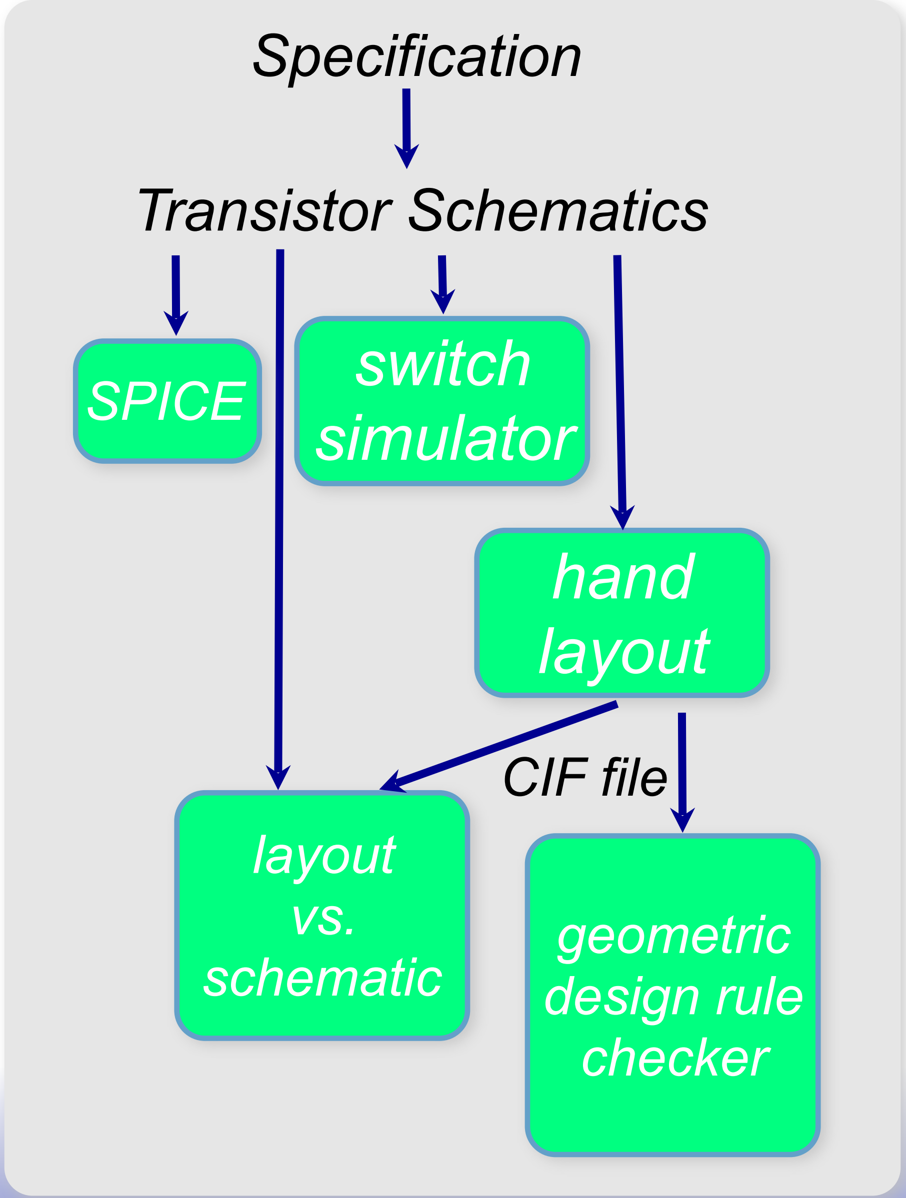
Several advances lead to the development of interactive tools for generating layout:

- Computer based layout representation (CIF, GDS).
- Advances in computer graphics (thanks to Ivan Sutherland and friends) and display devices.
- Personal “workstation” (Xerox Alto - Chuck Thacker). “Back room” computers didn’t have the necessary bandwidth to the display.
- Berkeley version - MAGIC

Early '80's Design Methodology and Flow

Schematic + Full-Custom Layout

- SPICE for timing,
- switch-level simulation for overall functionality,
- hand layout,
- no power analysis,
- layout verified with geometric Design Rule Checker (DRC) and later also Layout versus Schematic (LVS) Checkers



Computer Aided Design (2)

- For some time after CIF was invented: Layout was generated by hand, then typed in as a CIF file with a text editor.
- Layout compilers
  - Soon some designers started embedding CIF primitives in conventional programming languages: LISP, pascal, fortran, C.
  - This allows designers to write programs that generated layout. Such programs could be parameterized:

```
define GENERATE_RAM(rows, columns) {
  for I from 1 to rows
   for J from 1 to columns
     (GENERATE_BITCELL)}
GENERATE_RAM(128, 32);
```

  - *Lead to circuit/layout generation from higher level descriptions.*
- *Eventually, Cadence and Synopsys formed out of Berkeley.*

Implementation Alternatives & Design Flow

Implementation Alternative Summary

Full-custom:	All circuits/transistors layouts optimized for application.
Standard-cell:	Arrays of small function blocks (gates, FFs) automatically placed and routed.
Gate-array (structured ASIC):	Partially prefabricated wafers customized with metal layers or vias.
FPGA:	Prefabricated chips customized with loadable latches or fuses.
Microprocessor:	Instruction set interpreter customized through software.
Domain Specific Processor:	Special instruction set interpreters (ex: DSP, NP, GPU).

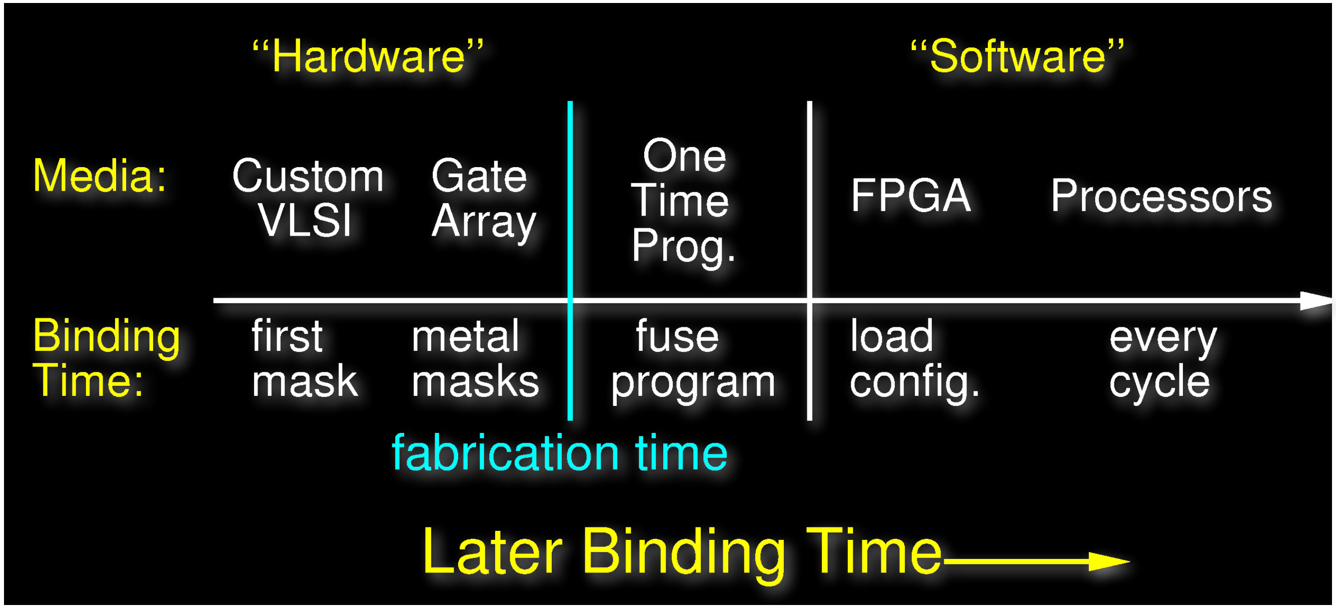
These days, "ASIC" almost always means Standard-cell.

What are the important metrics of comparison?

The Important Distinction

Instruction Binding Time

- *When* do we decide the functions (what operation is to be performed)?



A. DeHon

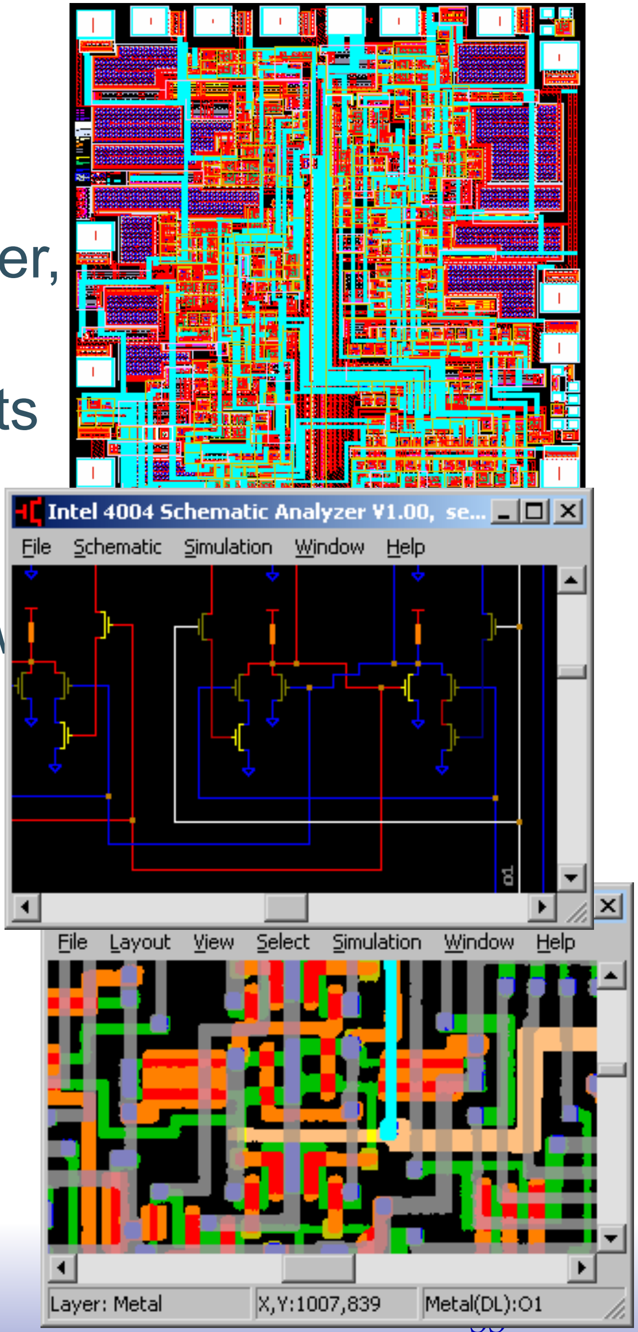
- ***General Principles***

Earlier the decision is bound, the less area, delay/energy required for the implementation.

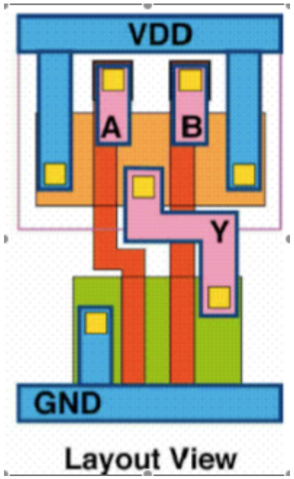
Later the decision is bound, the more flexible the device.

Full-Custom

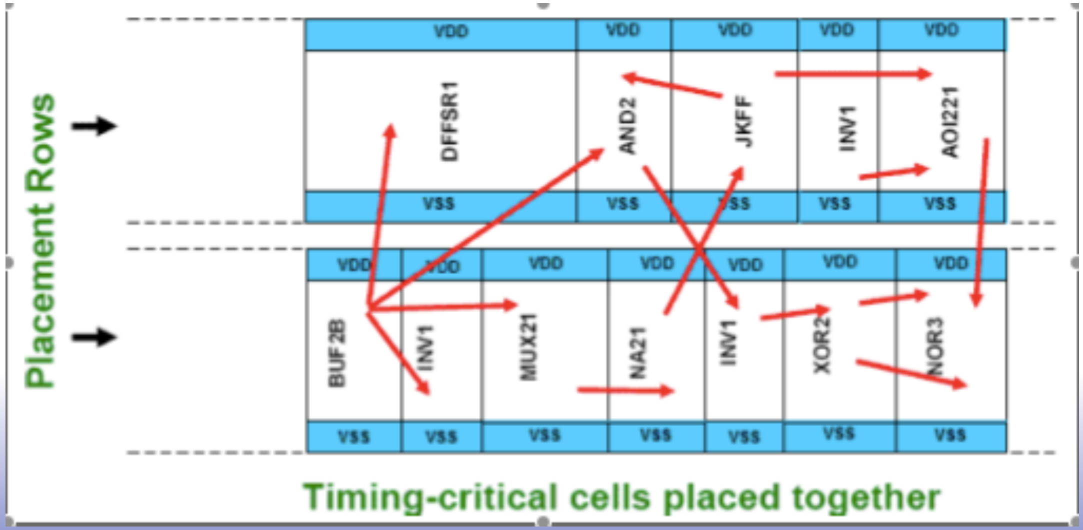
- Circuit styles and transistors are custom sized and drawn to optimize die, size, power, performance.
- High NRE (non-recurring engineering) costs
  - Time-consuming and error prone layout
- Hand-optimizing the layout can result in small die for low per unit costs, extreme-low power, or extreme-high-performance.
- Common today for **analog design**.
- Requires full set of custom masks.
- High NRE usually restricts use to high-volume applications/markets or highly-constrained and cost insensitive markets.

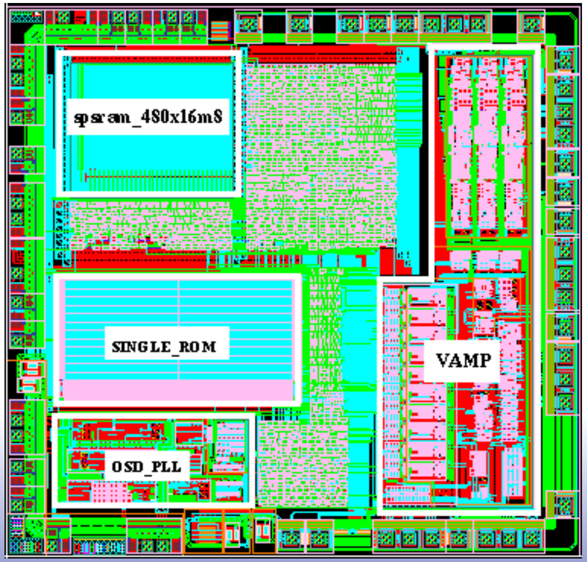


Standard-Cell*



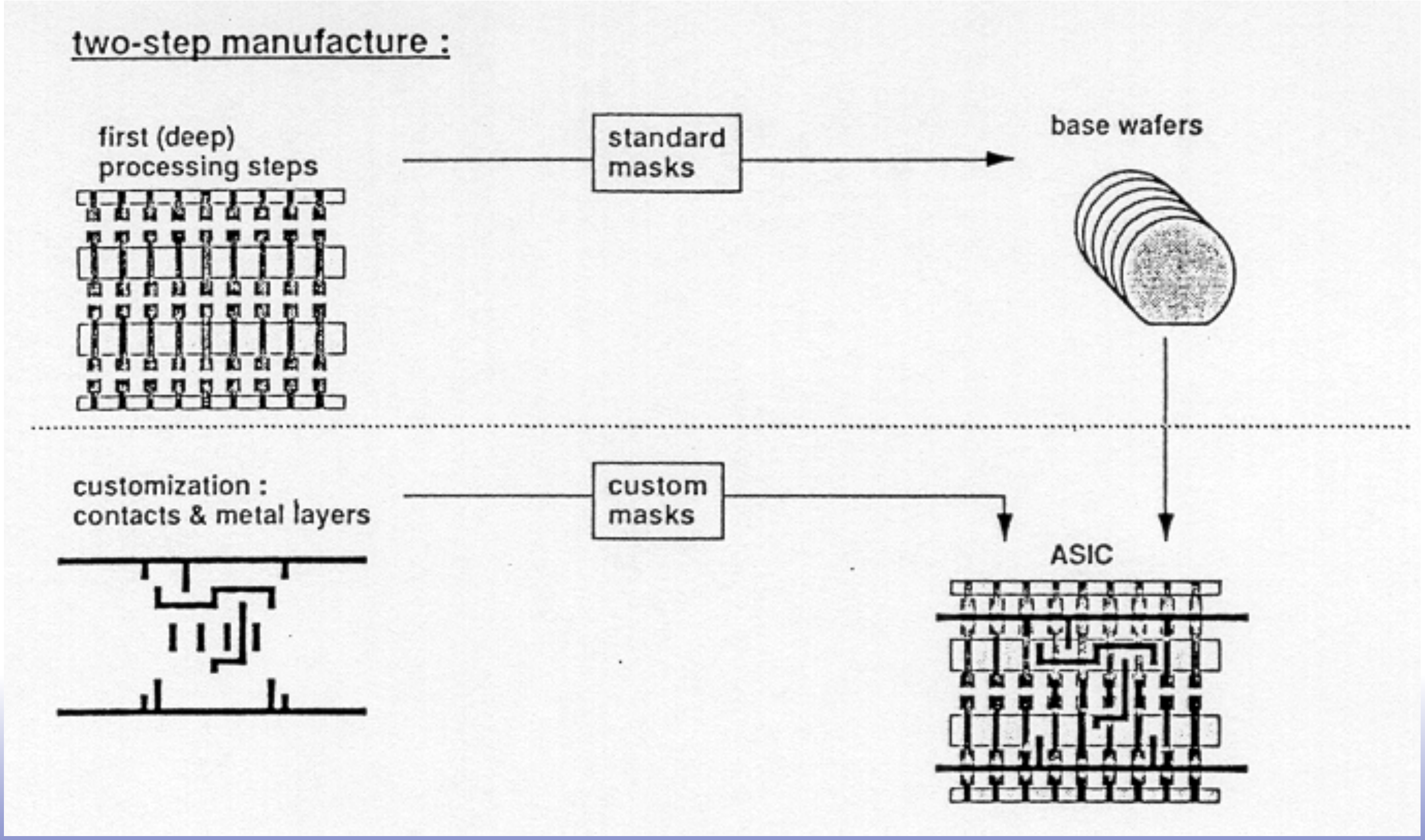
- Based around a set of pre-designed (and verified) cells
  - Ex: NANDs, NORs, Flip-Flops, counters slices, buffers, …
- Each cell comes complete with:
  - layout (perhaps for different technology nodes and processes),
  - Simulation, delay, & power models.
- Chip layout is automatic, reducing NREs (usually no hand-layout).
- Less optimal use of area and power, leading to higher per die costs than full-custom.
- Commonly used with other design implementation strategies (large blocks for memory, I/O blocks, etc.)





Gate Array

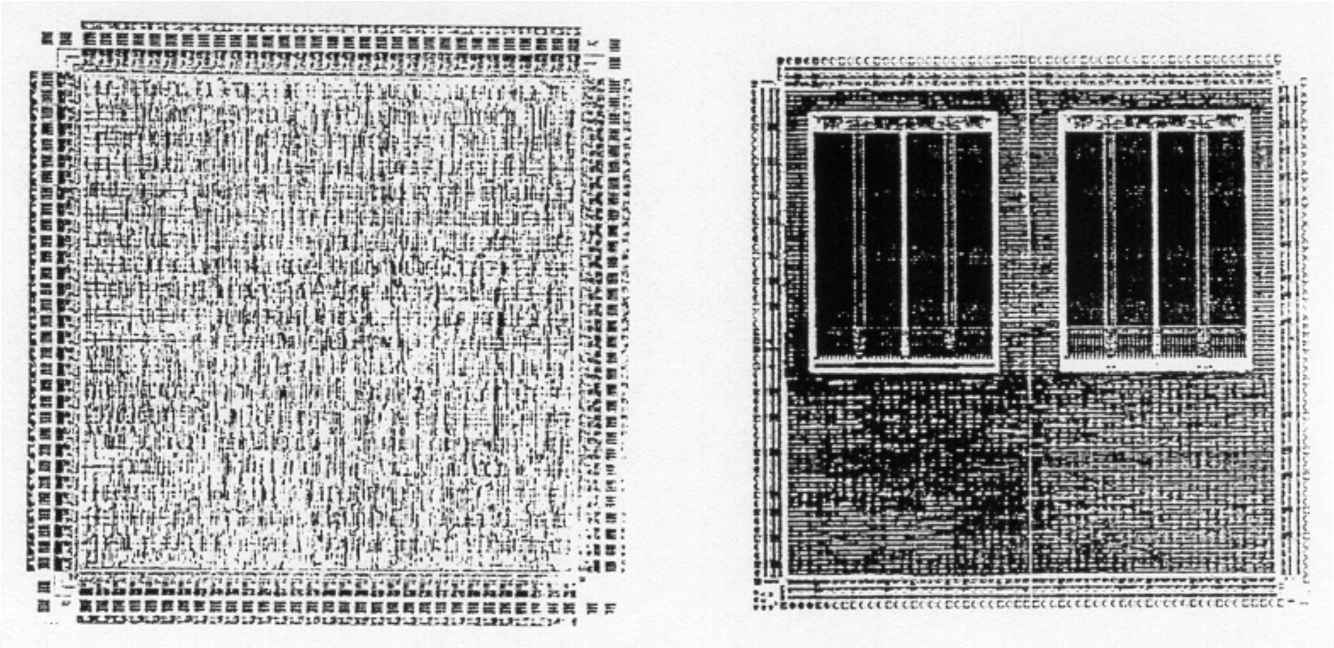
- Prefabricated wafers of "active" & gate layers & local interconnect, comprising, primarily, rows of transistors. Customize as needed with "back-end" metal processing (contact cuts, metal wires). Could use a different factory.
- CAD software understands how to make gates and registers.



Gate Array

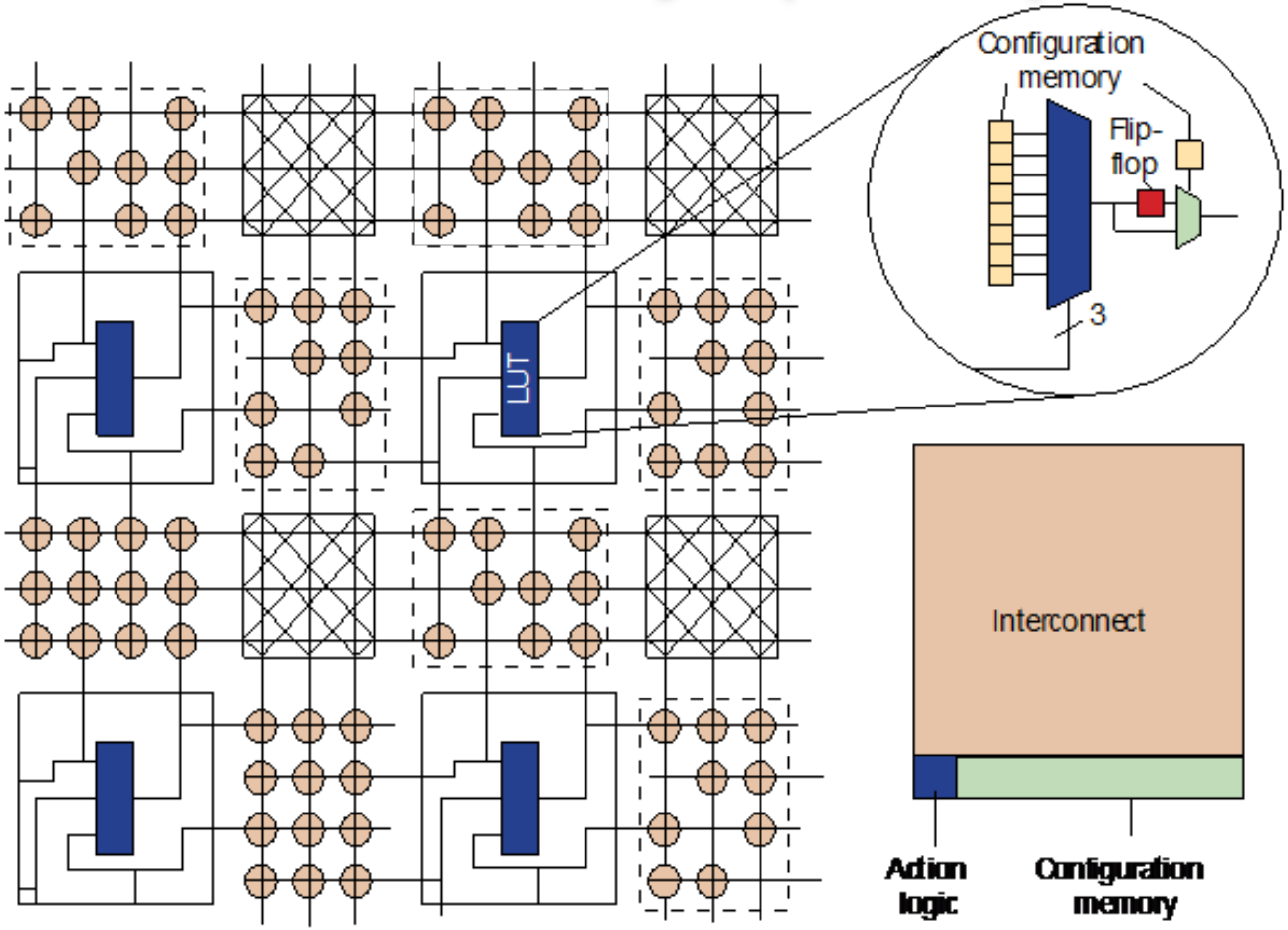
- *Shifts large portion of design and mask NRE to vendor.*
- *Shorter design and processing times, reduced time to market for user.*
- *Highly structured layout with fixed size transistors leads to large sub-circuits (ex: Flip-flops) and higher per die costs.*
- *Memory arrays are particularly inefficient, so often prefabricated, also:*

Sea-of-gates, structured ASIC, master-slice.



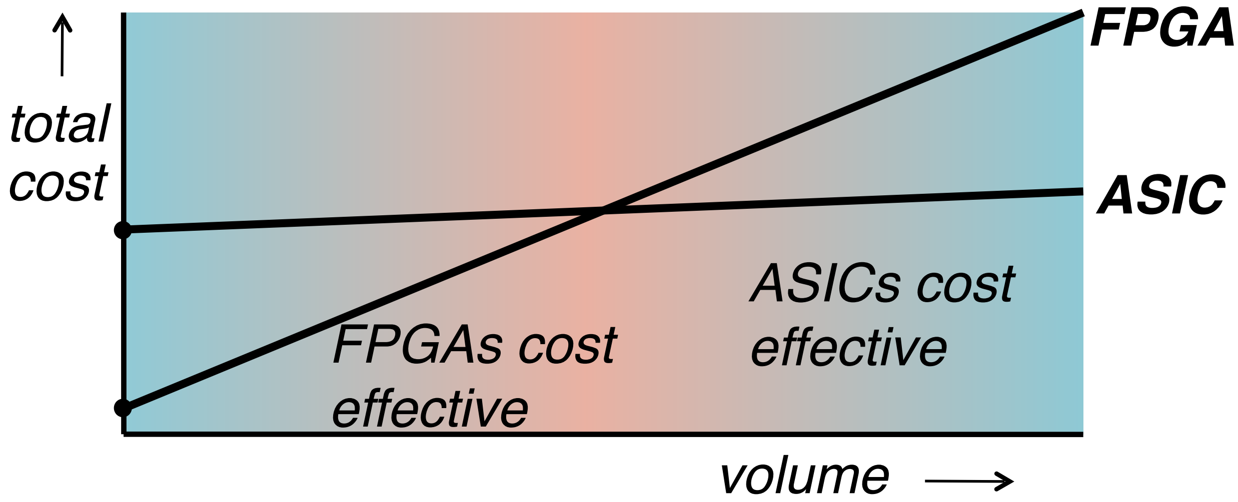
Field Programmable Gate Arrays (FPGA)

- *Two-dimensional array of simple logic- and interconnection-blocks.*
- *Typical architecture: Look-up-tables (LUTs) implement any function of n-inputs (n=3 in this case).*
- *Optional connected Flip-flop with each LUT.*



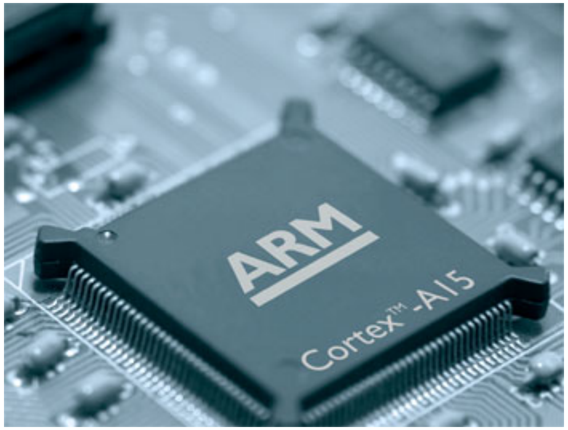
- Fuses, EPROM, or Static RAM cells are used to store the “configuration”.
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Many FPGAs include special circuits to accelerate adder carry-chain and many special cores: RAMs, MAC, Enet, PCI, SERDES, CPUs, ...

FPGA versus ASIC



- **ASIC:** Higher NRE costs (10's of \$M). Relatively Low cost per die (10's of \$ or less).
- **FPGAs:** Low NRE costs. Relatively low silicon efficiency ⇒ high cost per part (> 10's of \$ to 1000's of \$).
- **Cross-over volume** from cost effective FPGA design to ASIC was often in the 100K range.

Microprocessors



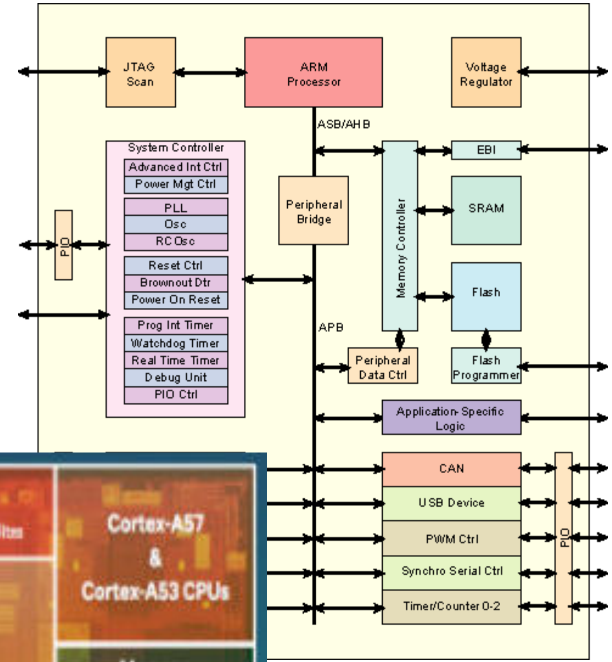
- Where relatively low performance and/or high flexibility is needed, a viable implementation alternative:
  - Software implements desired function
  - "Microcontroller", often with built in nonvolatile program memory and used as single function.
- Furthermore, instruction set processors are an "abstraction" level. Two ways:
  - Instruction Set Architecture (ISA)
  - "Synthesizable" RTL model ("soft core", available in HDL)
- Their implementation hosted on a variety of implementation platforms: standard-cell, gate-array, FPGA, other processors?

§	Assembler
	ADD{cond}{S} Rd, Rn, <Operand2>
	ADC{cond}{S} Rd, Rn, <Operand2>
5E	QADD{cond} Rd, Rm, Rn
5E	QDADD{cond} Rd, Rm, Rn
	SUB{cond}{S} Rd, Rn, <Operand2>
	SBC{cond}{S} Rd, Rn, <Operand2>
	RSB{cond}{S} Rd, Rn, <Operand2>
	RSC{cond}{S} Rd, Rn, <Operand2>
5E	QSUB{cond} Rd, Rm, Rn
5E	QDSUB{cond} Rd, Rm, Rn
2	MUL{cond}{S} Rd, Rm, Rs
2	MLA{cond}{S} Rd, Rm, Rs, Rn
M	UMULL{cond}{S} RdLo, RdHi, Rm, Rs
M	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs
6	UMAAL{cond} RdLo, RdHi, Rm, Rs

System-on-chip (SOC)

- *Brings together: standard cell blocks, custom analog blocks, processor cores, memory blocks, embedded FPGAs, …*
- *Standardized on-chip buses (or hierarchical interconnect) permit "easy" integration of many blocks.*
  – *Ex: AXI, AMBA, Sonics, …*
- *"IP Block" business model: Hard- or soft-cores available from third party designers.*

- *ARM, inc. is the shining example. Hard- and "synthesizable" RISC processors.*
- *ARM and other companies provide, Ethernet, USB controllers, analog functions, memory blocks, …*



Qualcomm Snapdragon

- Pre-verified block designs, standard bus interfaces (or adapters) ease integration - lower NREs, shorten TTM.

ASICs

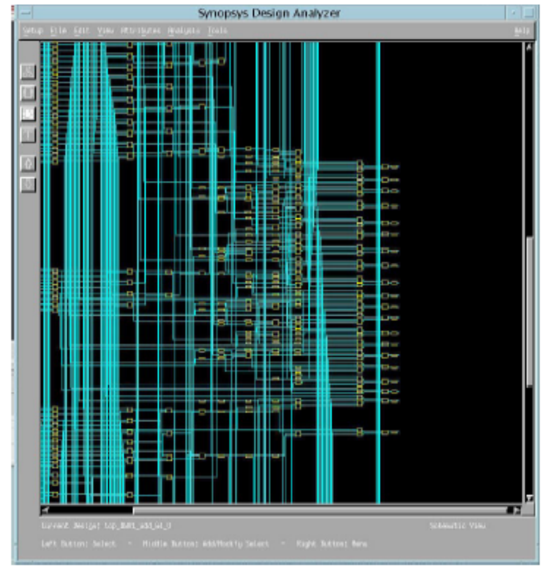
Verilog to ASIC layout flow

- "push-button" approach

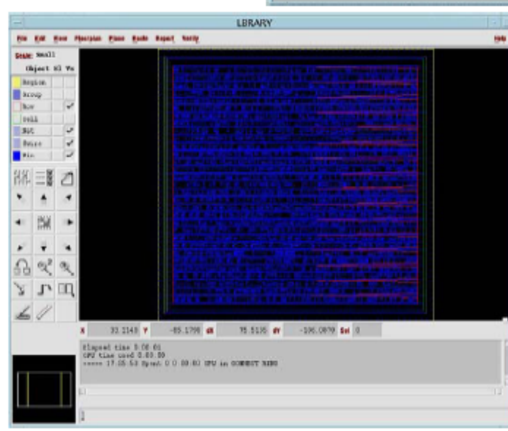
```
module adder64 (a, b, sum);
  input  [63:0] a, b;
  output [63:0] sum;

  assign sum = a + b;
endmodule
```

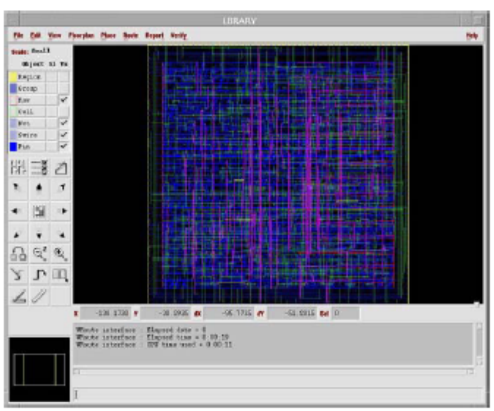
After Synthesis



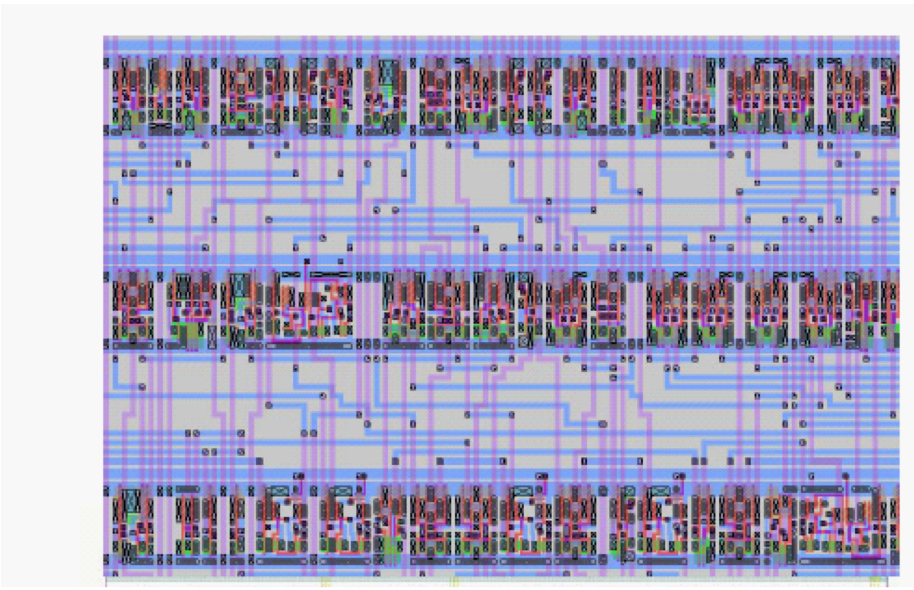
After Placement



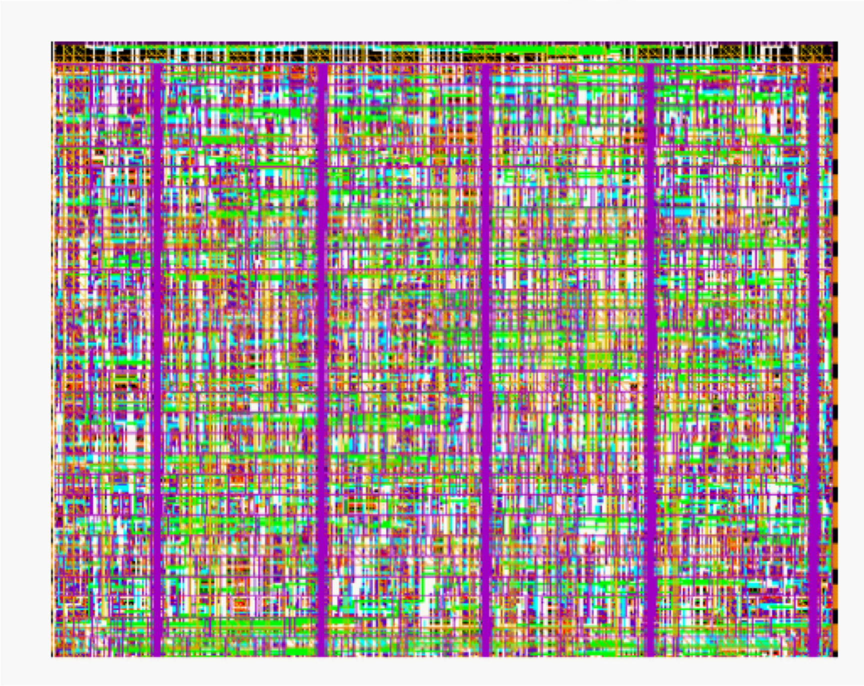
After Routing



Standard cell layout methodology



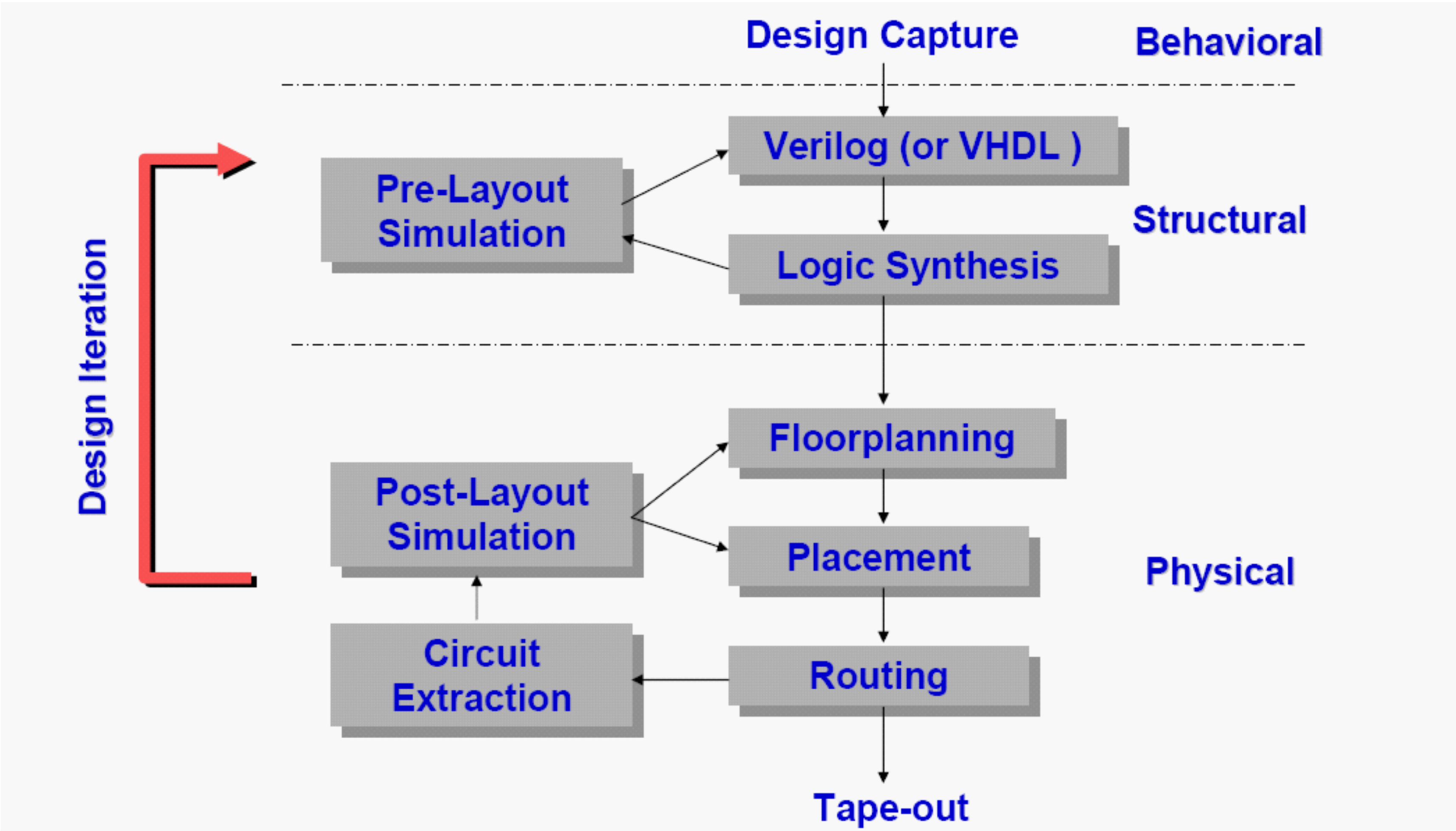
1um, 2-metal process



Modern sub-100nm process
"Transistors are free things that fit under wires"

- With limited # metal layers, dedicated routing channels were needed
- Currently area dominated by wires

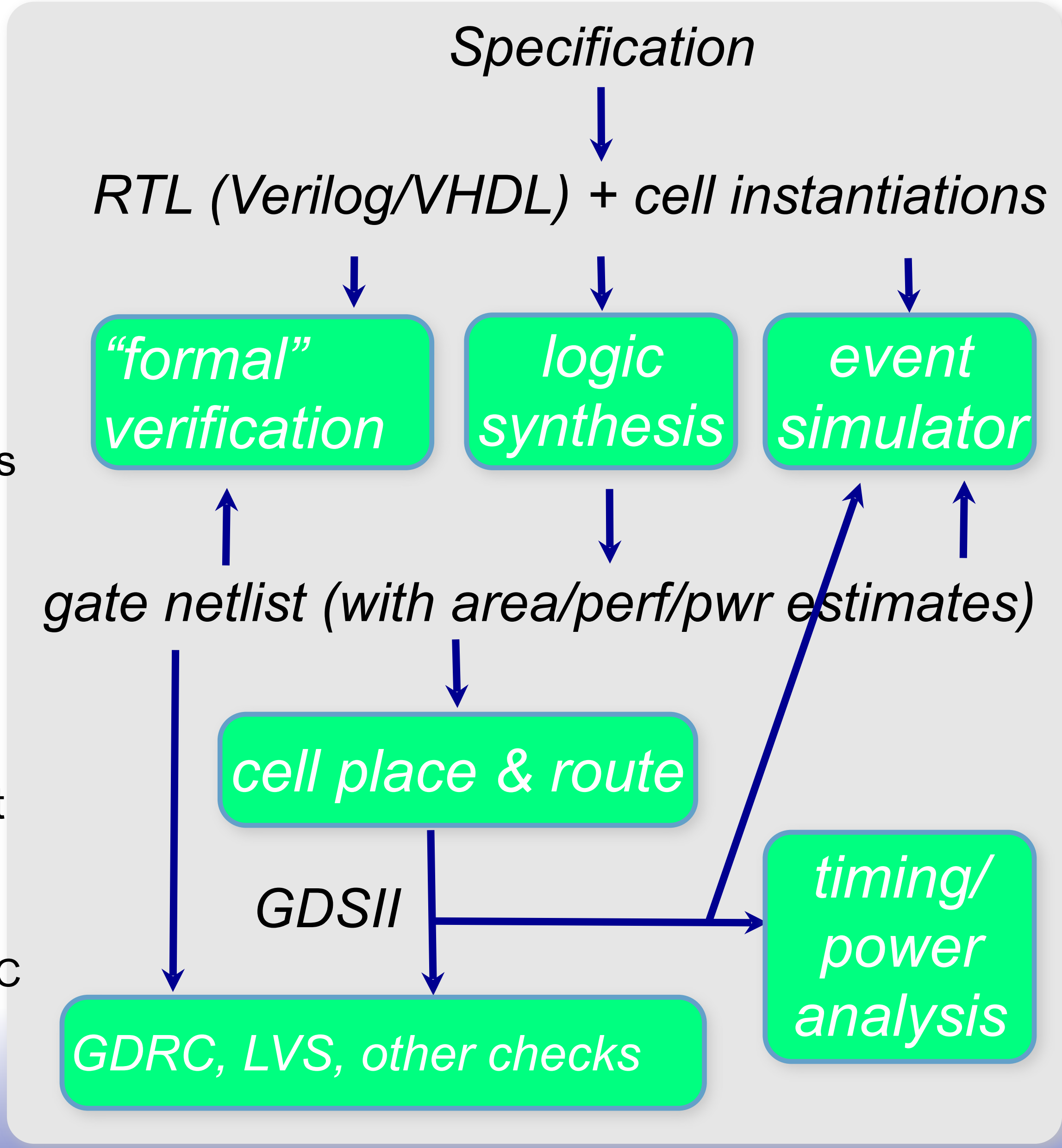
The ASIC flow



Modern ASIC Methodology and Flow

RTL Synthesis Based

- HDL specifies design as combinational logic + state elements
- Logic Synthesis converts hardware description to gate and flip-flop implementation
- Cell instantiations needed for blocks not inferred by synthesis (typically RAM)
- Event simulation verifies RTL
- "Formal" verification compares logical structure of gate netlist to RTL
- Place & route generates layout
- Timing and power checked statically
- Layout verified with LVS and GDRC

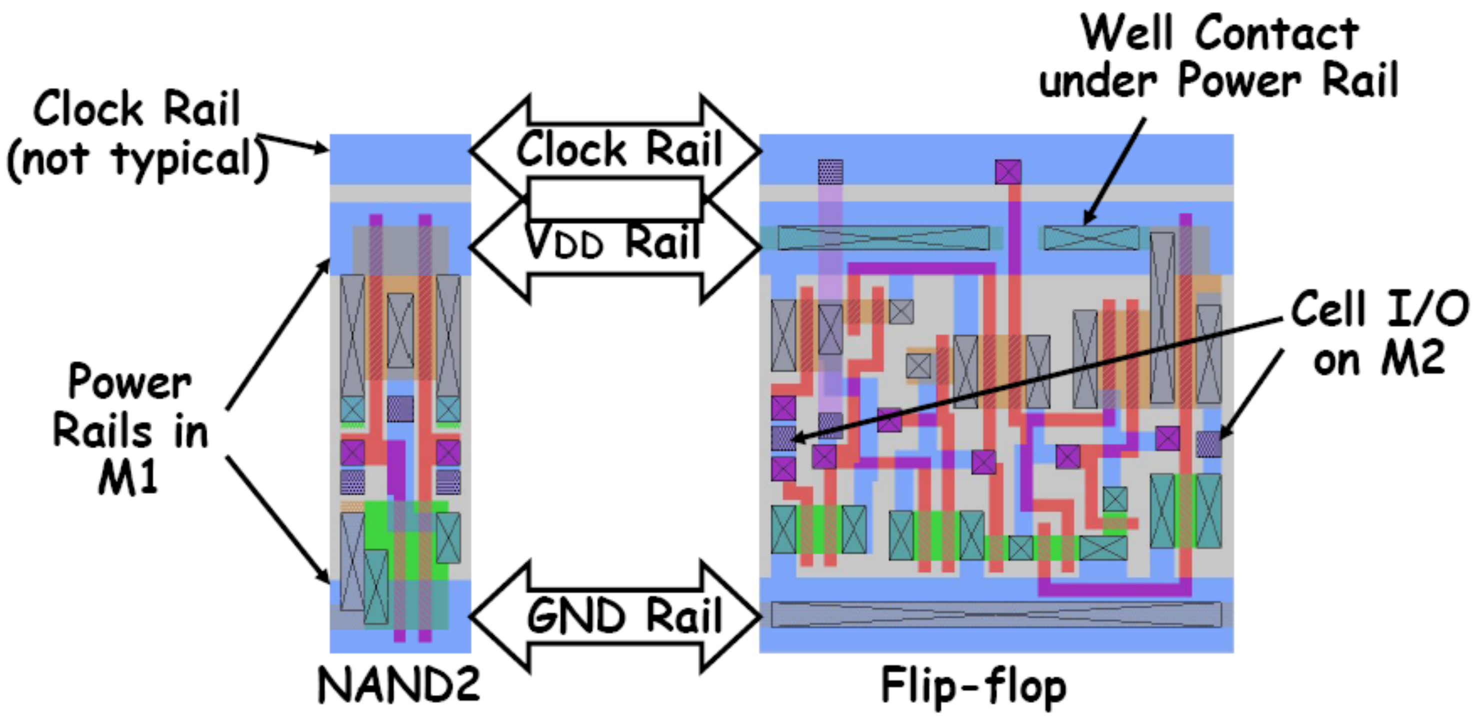


Standard cell design

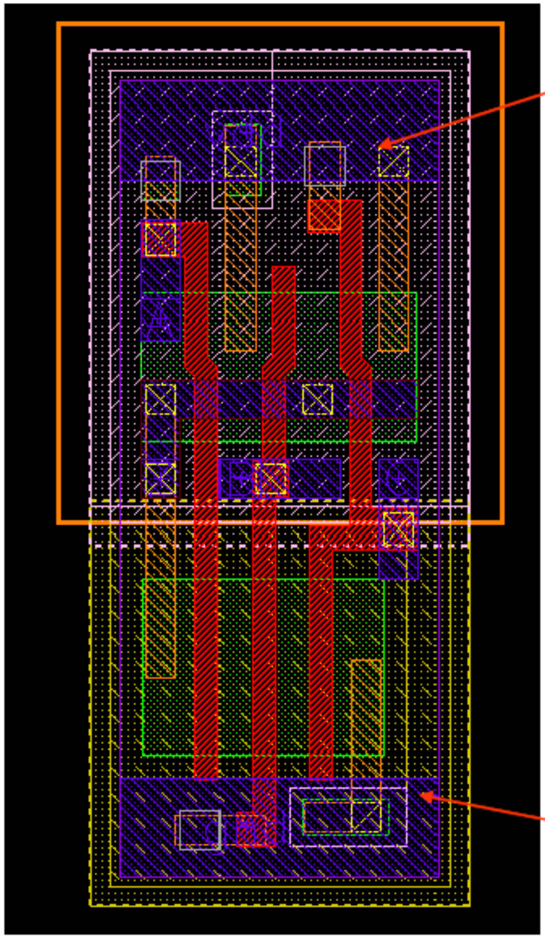
- Layout considerations

Cells have standard height but vary in width

Designed to connect power, ground, and wells by abutment



Standard cell characterization



Power Supply Line (V_{DD})

Delay in (ns)!!

Path	1.2V - 125°C	1.6V - 40°C
$In1—t_{pLH}$	$0.073+7.98C+0.317T$	$0.020+2.73C+0.253T$
$In1—t_{pHL}$	$0.069+8.43C+0.364T$	$0.018+2.14C+0.292T$
$In2—t_{pLH}$	$0.101+7.97C+0.318T$	$0.026+2.38C+0.255T$
$In2—t_{pHL}$	$0.097+8.42C+0.325T$	$0.023+2.14C+0.269T$
$In3—t_{pLH}$	$0.120+8.00C+0.318T$	$0.031+2.37C+0.258T$
$In3—t_{pHL}$	$0.110+8.41C+0.280T$	$0.027+2.15C+0.223T$

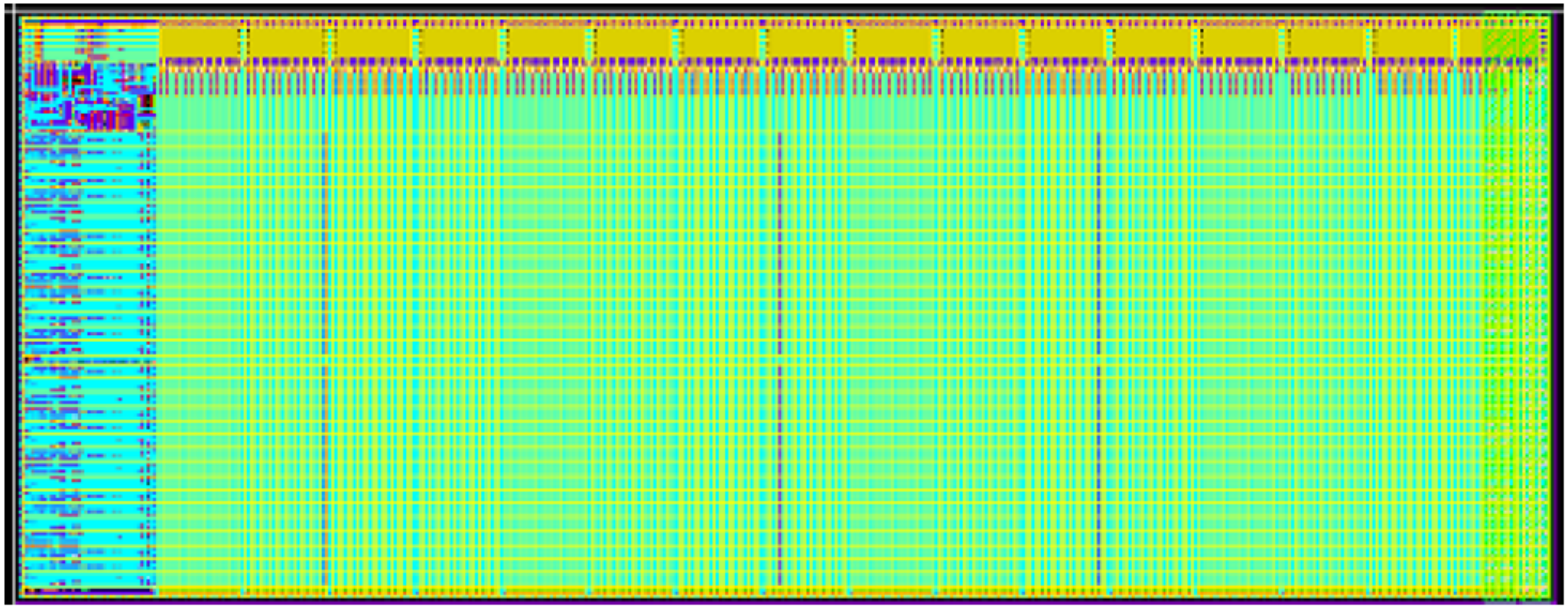
3-input NAND cell
(from ST Microelectronics):
C = Load capacitance
T = input rise/fall time

Ground Supply Line (GND)

- **Each library cell (FF, NAND, NOR, INV, etc.) and the variations on size (strength of the gate) is fully characterized across temperature, loading, etc.**

Macro modules

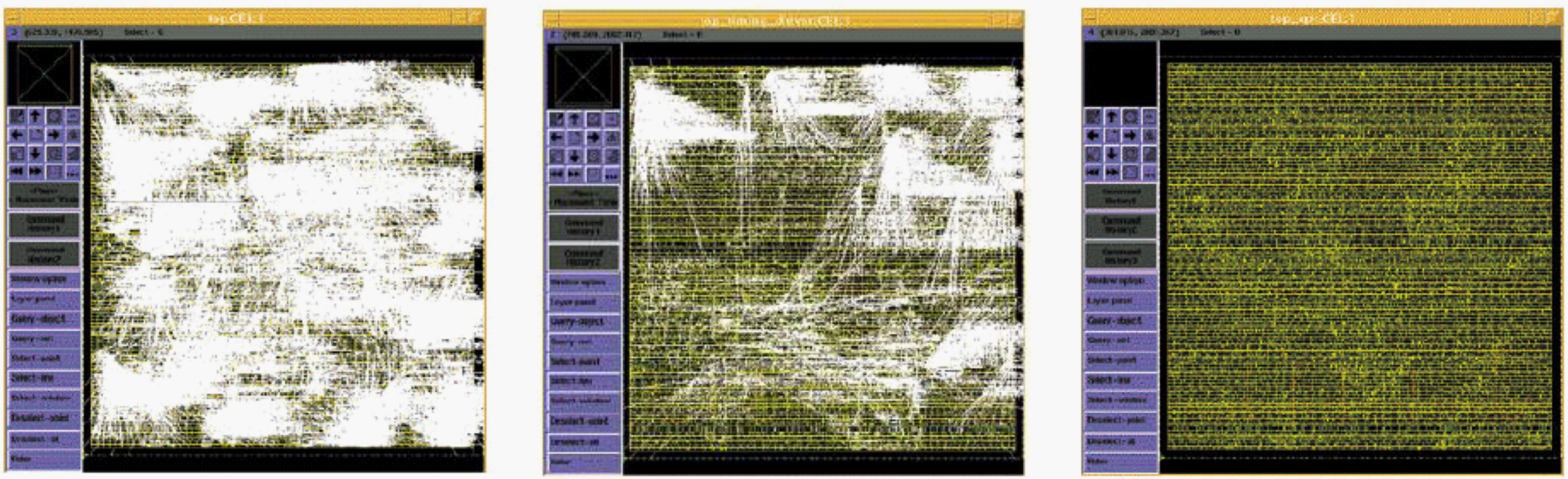
256×32 (or 8192 bit) SRAM Generated by hard-macro module generator



- Generate highly regular structures (entire memories, multipliers, etc.) with a few lines of code
- Verilog models for memories automatically generated based on size

The "Design closure" problem

- Biggest problem are wires (signals and clock)



Iterative Removal of Timing Violations (white lines)