EECS 151/251A
Spring 2021
Digital Design and Integrated Circuits

Instructor:
J. Wawrzynek

Lecture 2: Design

Outline

- Details of Design Metrics
- Digital Logic – Basic Concepts
- Design Implementation Alternatives
- Design Flows
- ASICs

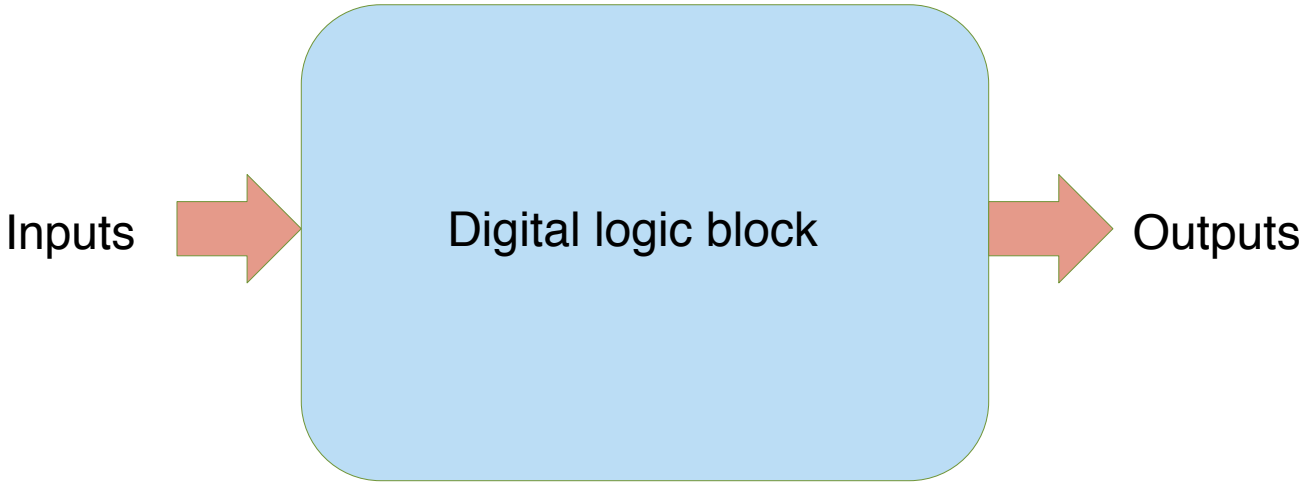
Review from Lecture 1

- Moore's law is slowing down
  - There are continued improvements in technology, but at a slower pace, and manufacturing costs
- Dennard's scaling has ended a decade ago
  - All designs are now power limited
- Multi-cores, specialization and customization provides added performance
  - Under power constraints and slowing technology advances
- Design costs are high
  - Methodology and better tools to rescue!
- All design decisions involve tradeoffs between *performance, cost, and power*
  - Pareto optimally defines the best designs.

Digital Logic

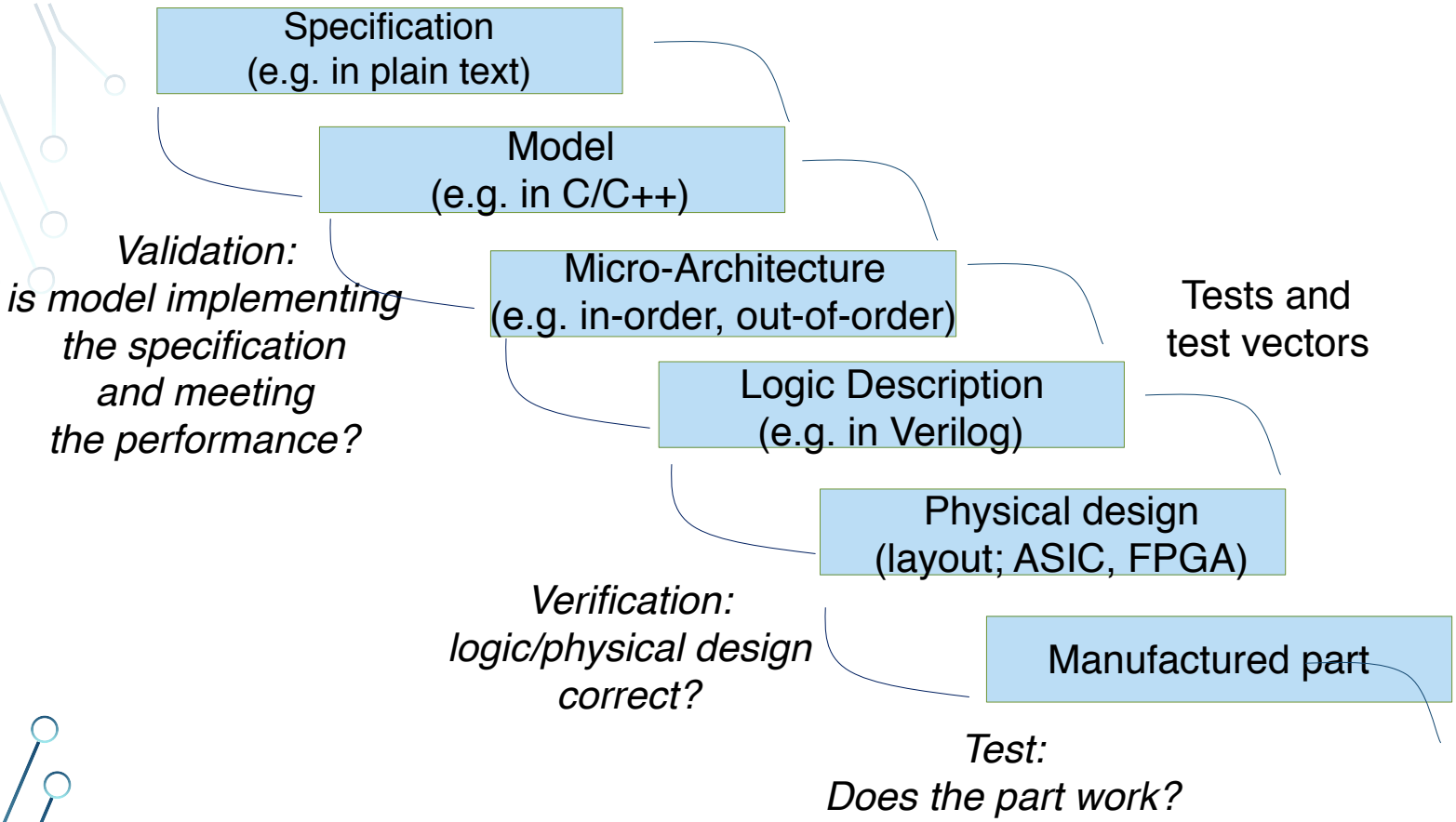
Implementing Digital Systems

- Given a functional description and performance, cost, & power constraints, come up with an implementation using a set of primitives.
- Digital systems are implemented as a set of *combinational logic and state elements*:



- What is the methodology we use to implement a digital system?

Design Process through layers of **abstractions**

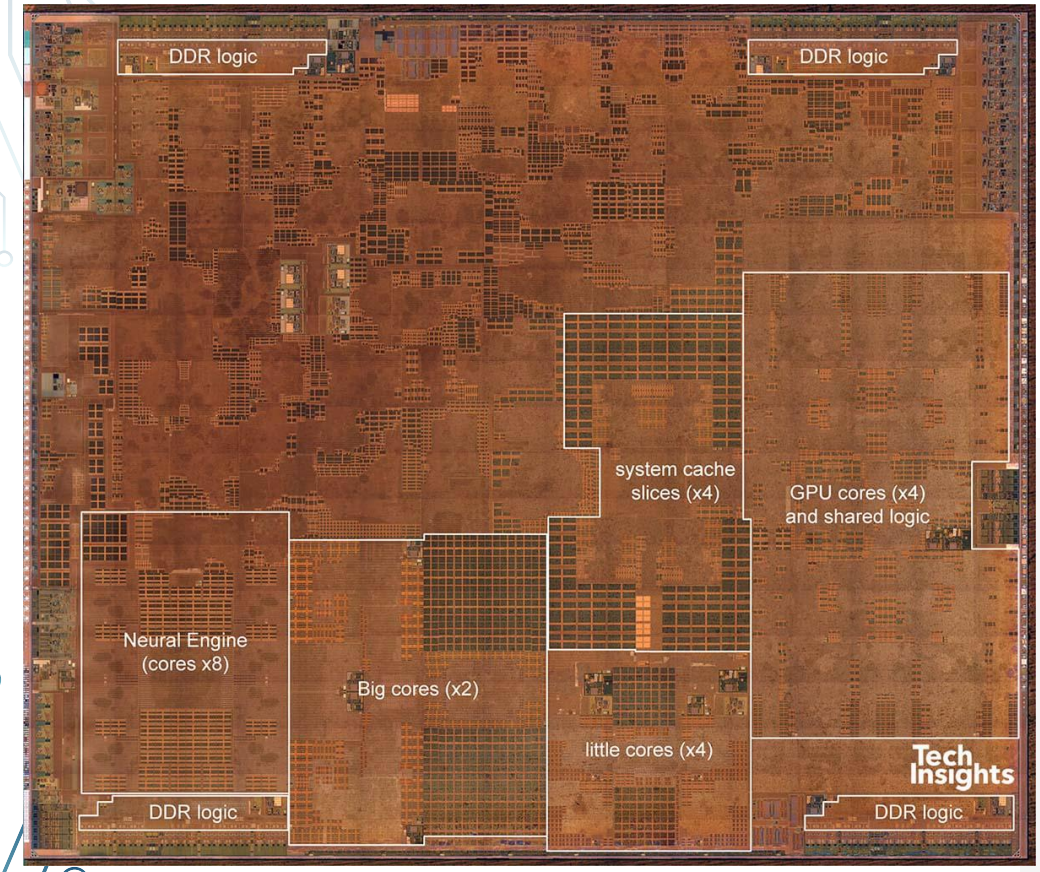


Validation: Have we built the right thing?
Verification: Have we built the thing correctly?

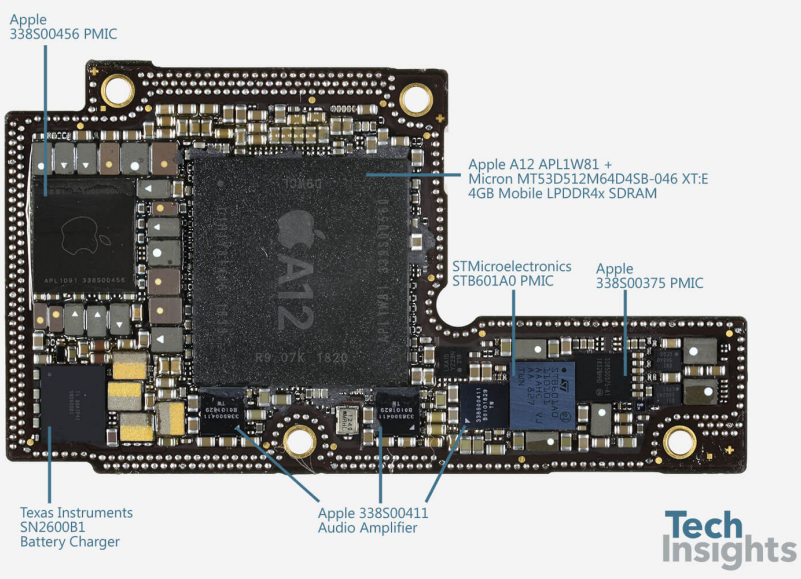
The key to success is that each layer preserves the essential functionality and constraints from above, but adds more details.

Modern (Mostly) Digital System-On-A-Chip (SOC)

- Apple A12 Bionic



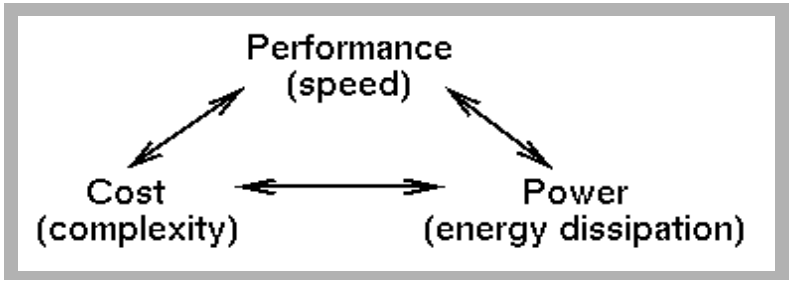
- 2x Large CPUs
- 4x Small CPUs
- GPUs
- Neural processing unit (NPU)
- Lots of memory
- DDR memory interfaces



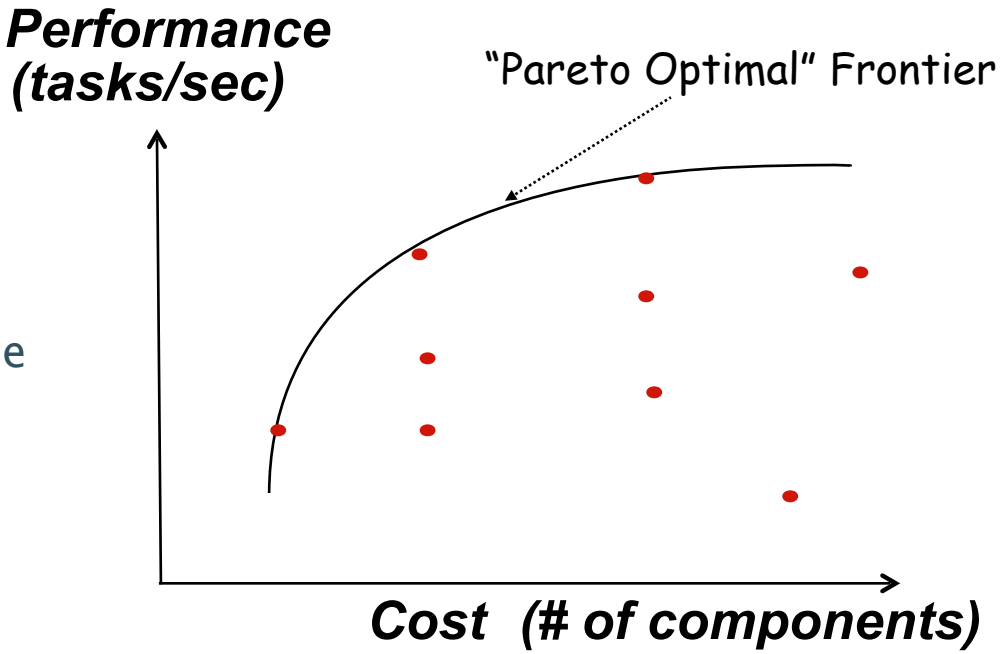
- 7nm CMOS
- Up to 2.49GHz

Design Metrics

Basic Design Tradeoffs

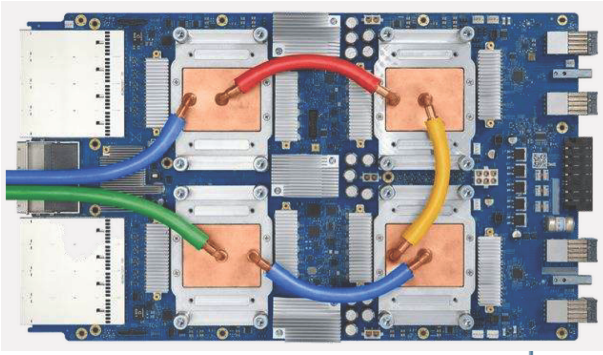


- Improve on one at the expense of the others
- Tradeoffs exist at every level in the system design
- Design Specification
  - Functional Description
  - Performance, cost, power constraints
- Designer must make the tradeoffs needed to achieve the function within the constraints



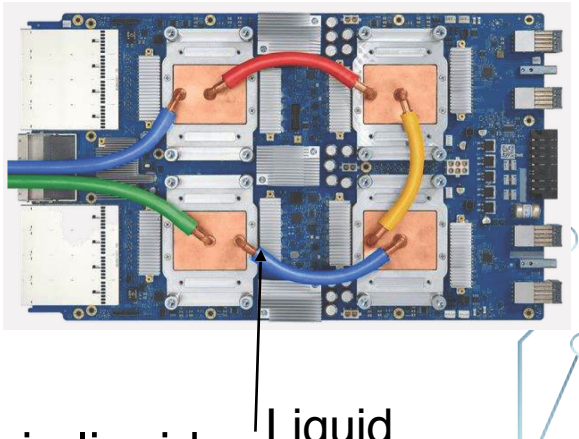
Performance

- **Throughput**
  - Number of tasks performed in a unit of time (operations per second)
  - E.g. Google TPUv3 board performs 420 TFLOPS (10^{12} floating-point operations per second, where a floating point operation is BFLOAT16)
  - Watch out for 'op' definitions – can be a 1-b ADD or a double-precision FP add (or more complex task)
  - Peak vs. average throughput
- **Latency**
  - How long does a task take from start to finish
  - E.g. facial recognition on a phone takes 10's of ms
  - Sometime expressed in terms of clock cycles
  - Average vs. 'tail' latency



Energy and Power

- **Energy** (in joules (J))
  - Needed to perform a task (energy efficiency tells us J/op)
  - Ex: add two numbers or fetch a datum from memory
  - Battery stores certain amount of energy (in Ws = J or Wh)
  - That is what utility charges for (in kWh)
- **Power** (in watts (W))
  - Energy dissipated per unit time (W = J/s)
  - Sets cooling requirements
    - Heat spreader, size of a heat sink, forced air, liquid, …



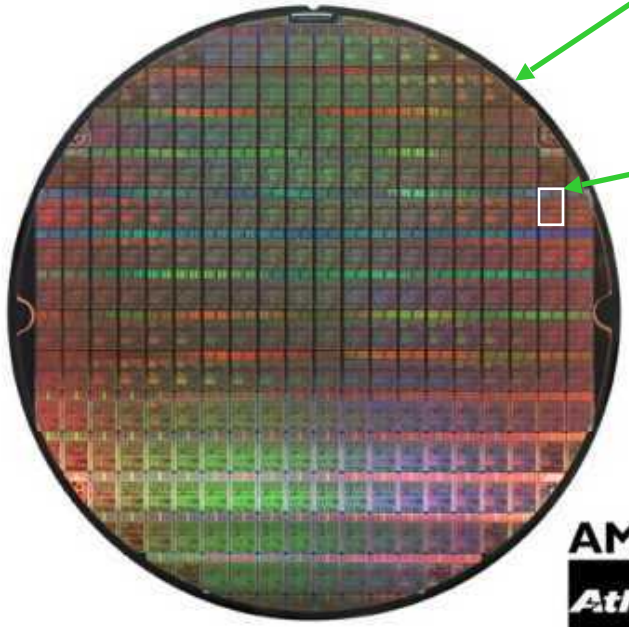
Cost

- **Non-recurring** engineering (NRE) costs
- Cost to develop a design (product) - *people, tools, masks*
  - Amortized over all units shipped
  - E.g. $20M in development adds $.20 to each of 100M units
- **Recurring** costs
  - Cost to manufacture, test and package a unit
  - Processed wafer cost is ~10k (around 16nm node) which yields:
    - 50-100 large FPGAs or GPUs
    - 200 laptop CPUs
    - >1000 cell phone SoCs

$$\text{cost per IC} = \text{variable cost per IC} + \frac{\text{fixed cost}}{\text{volume}}$$

$$\text{variable cost} = \frac{\text{cost of die} + \text{cost of die test} + \text{cost of packaging}}{\text{final test yield}}$$

Die Cost



$$\text{cost of die} = \frac{\text{cost of wafer}}{\text{dies per wafer} * \text{die yield}}$$

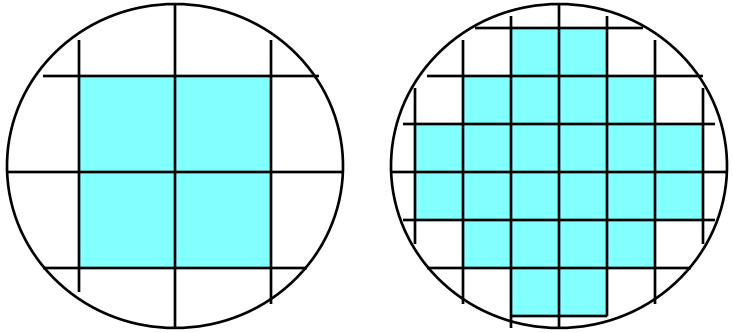
From: http://www.amd.com

Yield

$$Y = \frac{\text{No. of good chips per wafer}}{\text{Total number of chips per wafer}} \times 100\%$$

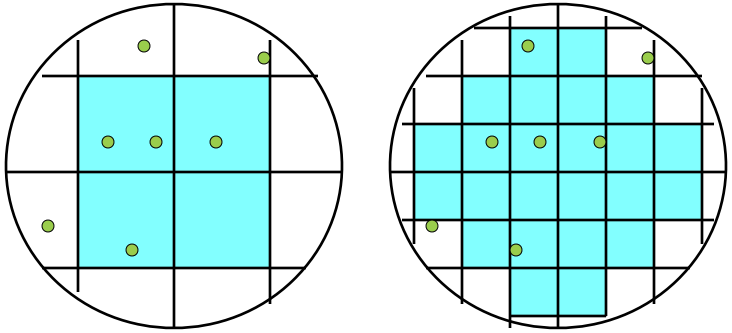
$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{wafer diameter}/2)^2}{\text{die area}} - \frac{\pi \times \text{wafer diameter}}{\sqrt{2 \times \text{die area}}}$$



Defects

Yield = 0.25



Yield = 0.76

$$\text{die yield} = \left(1 + \frac{\text{defects per unit area} \times \text{die area}}{\alpha}\right)^{-\alpha}$$

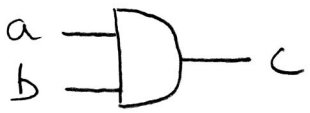
α is approximately 3

$$\text{die cost} = f(\text{die area})^4$$

Digital Logic
Basic Concepts

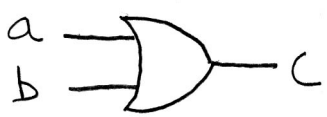
Logic Gates

AND



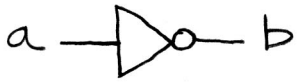
ab	c
00	0
01	0
10	0
11	1

OR



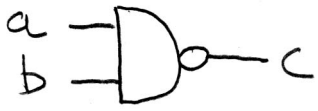
ab	c
00	0
01	1
10	1
11	1

NOT



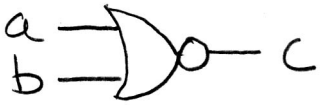
a	b
0	1
1	0

NAND



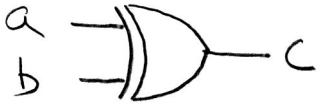
ab	c
00	1
01	1
10	1
11	0

NOR



ab	c
00	1
01	0
10	0
11	0

XOR



ab	c
00	0
01	1
10	1
11	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?

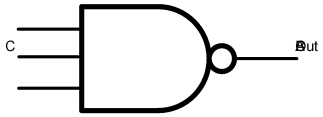
Multi-Input Gates

3-Input NAND

NAND3 Boolean equation

$$Out = \overline{A \cdot B \cdot C}$$

NAND3

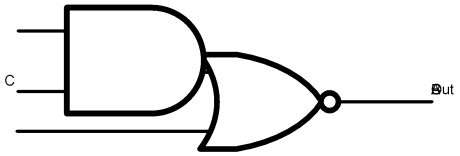


A	B	C	Out
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

And-Or-Invert

AOI21 Boolean equation

$$Out = \overline{A \cdot B + C}$$

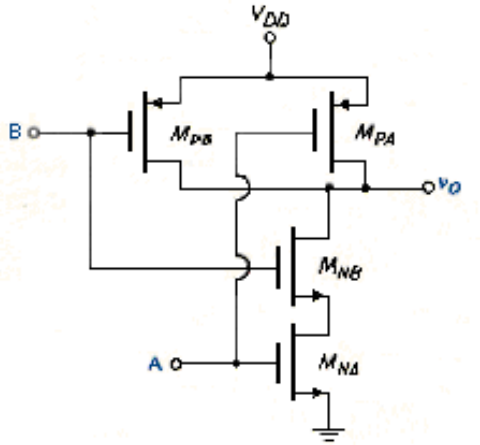


A	B	C	Out
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

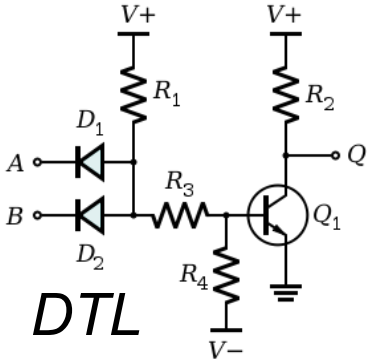
- Single gate in modern CMOS usually doesn't have more than 3-4 inputs

Logic Gate Implementation

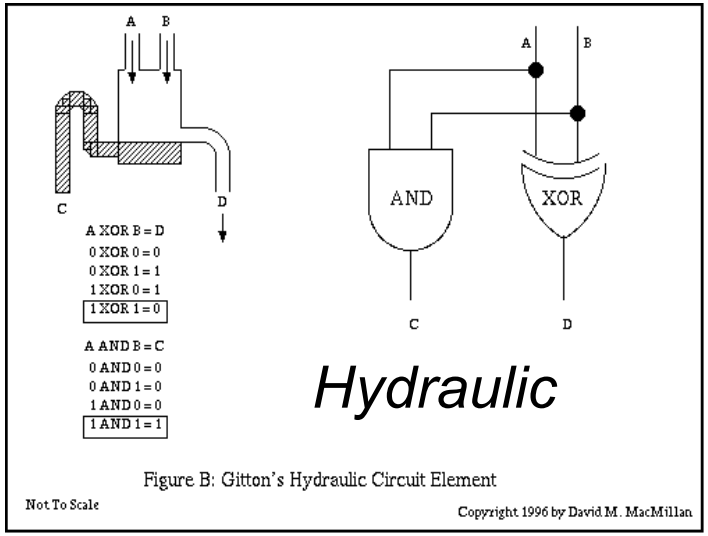
- Logic circuits have been built out of many different technologies. If we have a basic logic gate (AND or OR) and inversion we can build a complete logic family.



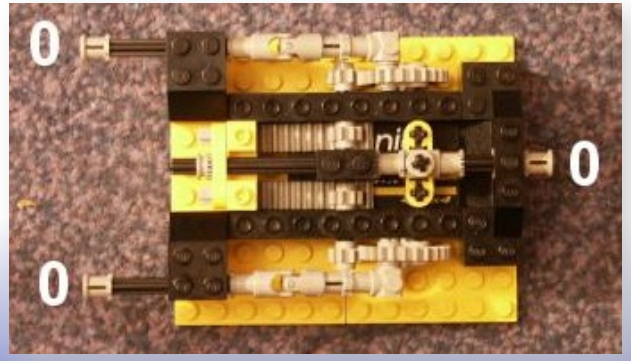
CMOS Gate



DTL



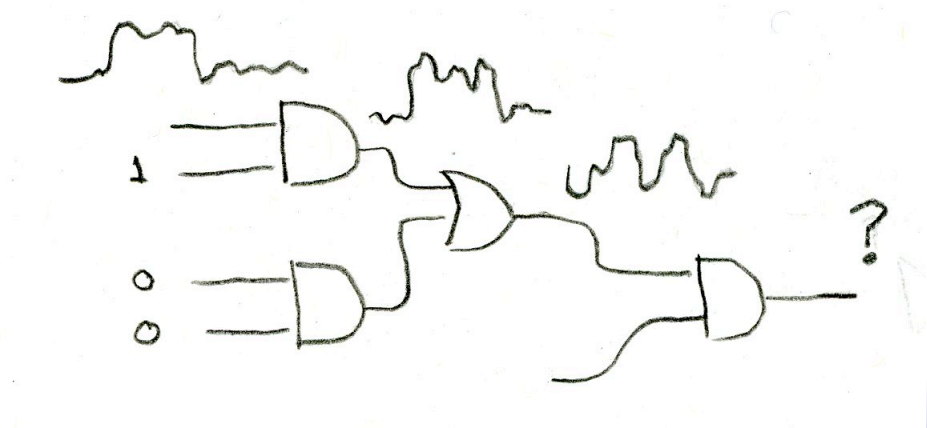
Hydraulic



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

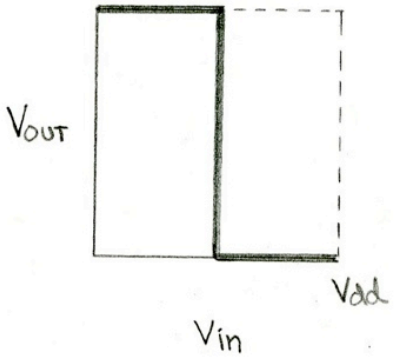
Restoration/Regeneration

- 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.

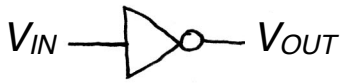


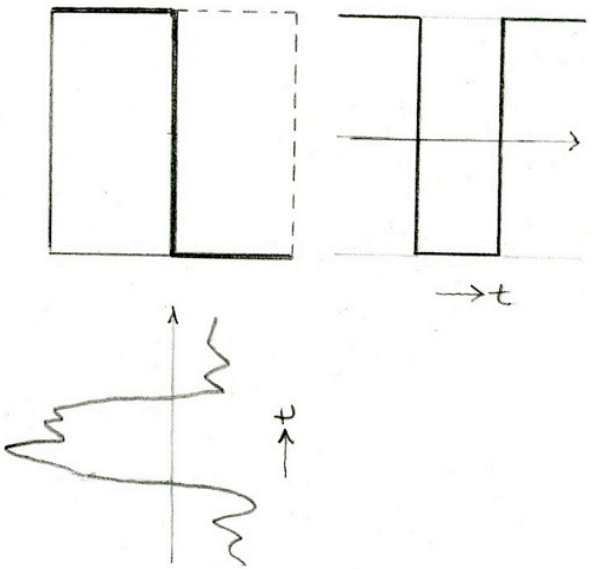
Inverter Example of Restoration

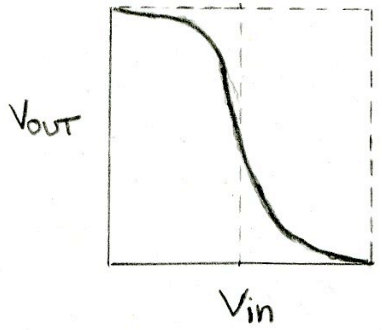
Example (look at 1-input gate, to keep it simple):



Idealize Inverter





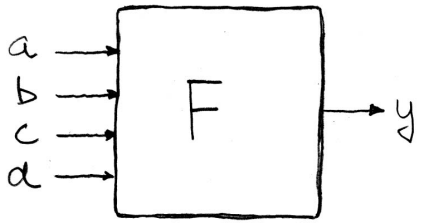


Actual Inverter voltage transfer characteristic (VTC)

- 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.

Combinational Logic Blocks

Example four-input Boolean function:



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)

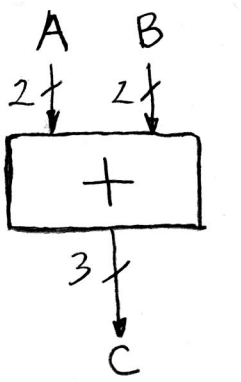
Truth Table

- 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.

Example CL Block

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

$C = A + B$

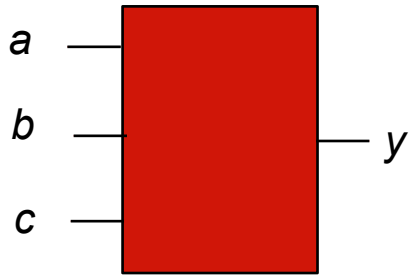


a1 a0	b1 b0	c2 c1 c0
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

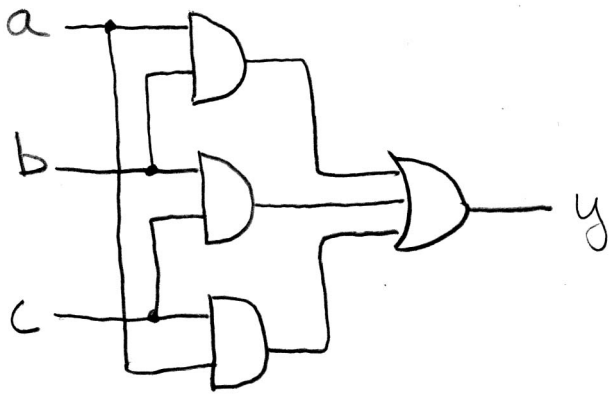
- Think about truth table for 32-bit adder. It's possible to write out, but it might take a while!

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

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

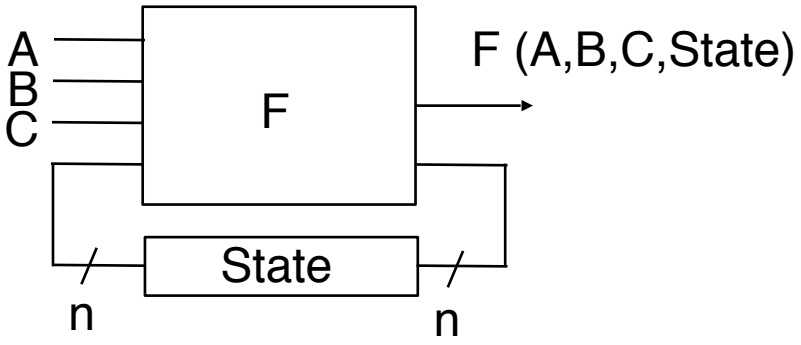


How do we know that these two representations are equivalent?

Will come back to this later!

Sequential Logic Blocks

- Output is a function of both the current inputs and the state.

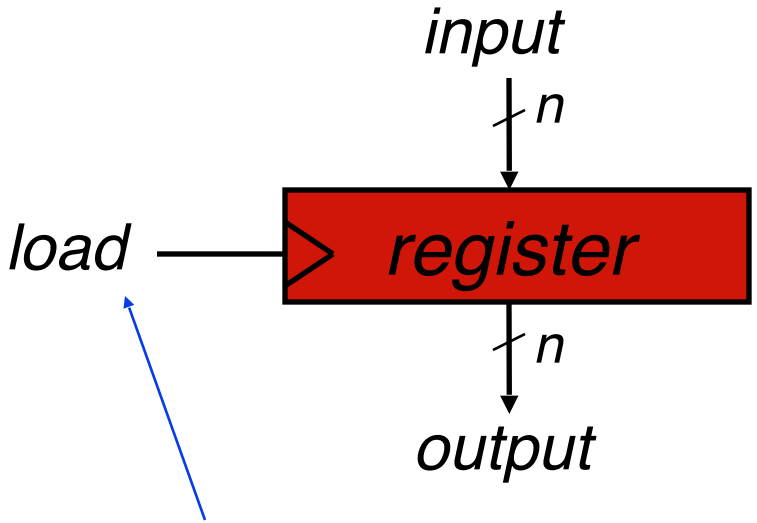


- "State" stored as memory.
- State is a function of previous inputs.
- In synchronous digital systems, state is updated on each clock tick.
- "F" is just a combinational logic block.

This means the way the block responds to a particular input depends on what it has seen previously.

State Elements: circuits that store info

- Examples: registers, memory blocks
- Register: Stores one **word**. Under the control of the "load" signal, the register captures the input value and stores it indefinitely.

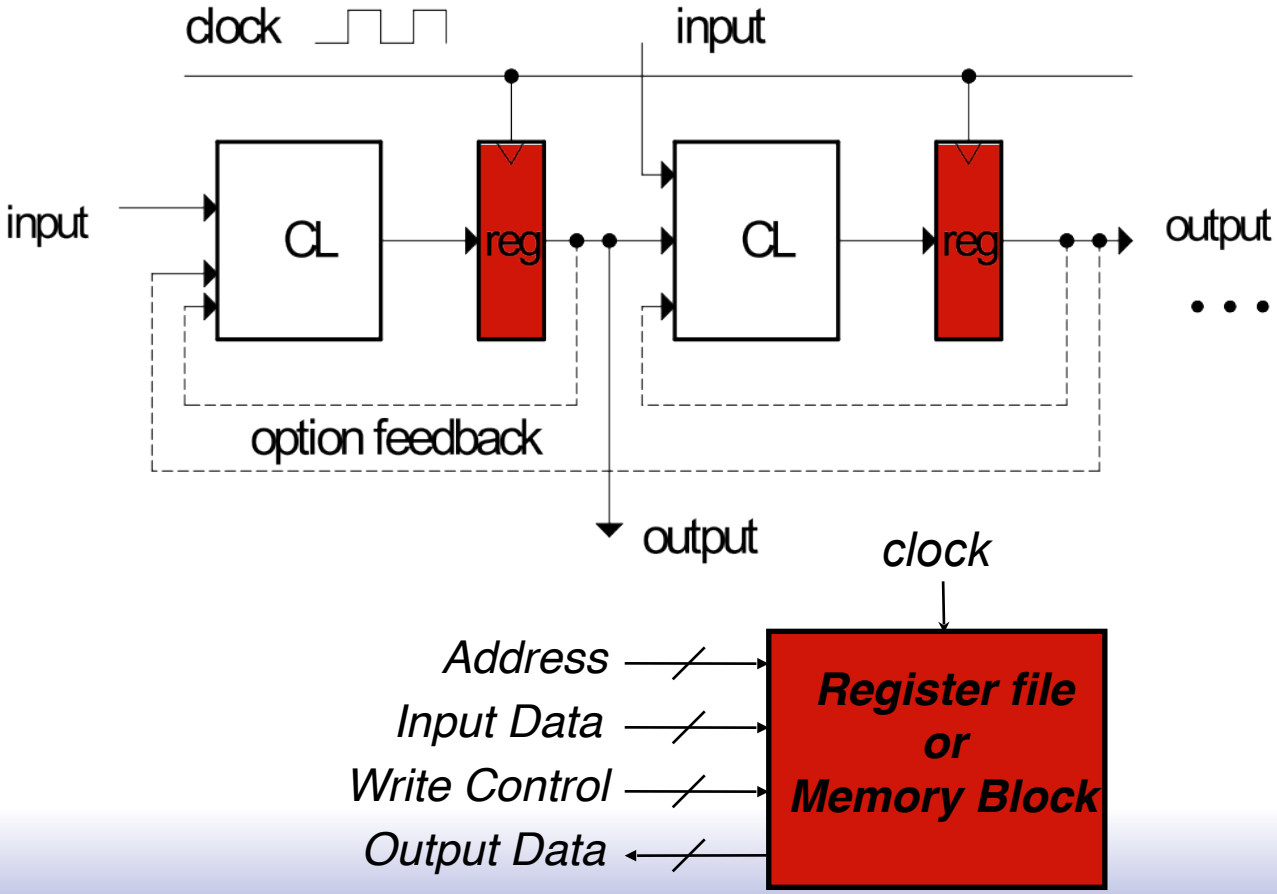


- The value stored by the register appears on the output (after a small delay).
- Until the next load, changes on the data input are ignored (unlike CL, where input changes change output).
- These get used for short term storage (ex: register file), and to help move coordinate data movement.

Register Transfer Level Abstraction (RTL)

Any synchronous digital circuit can be represented with:

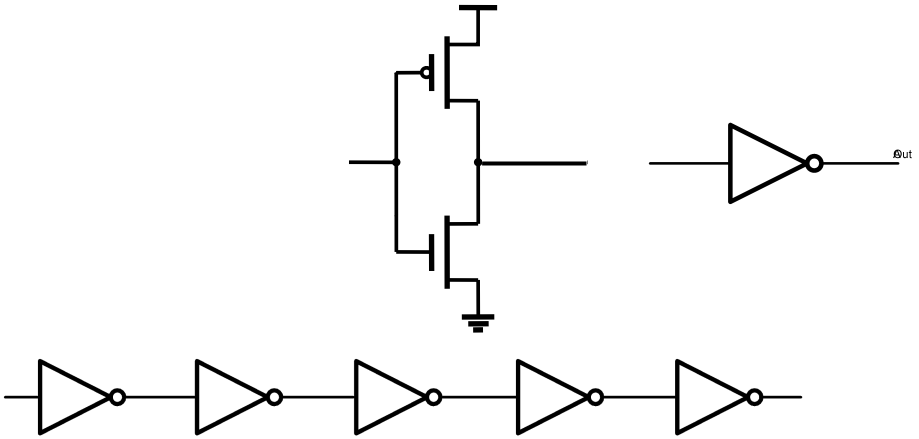
- Combinational Logic Blocks (CL), plus
- State Elements (registers or memories)



- State elements are mixed in with CL blocks to remember and to control the flow of data.
- Sometimes used in large groups by themselves for "long-term" data storage.

Digital Logic Delay

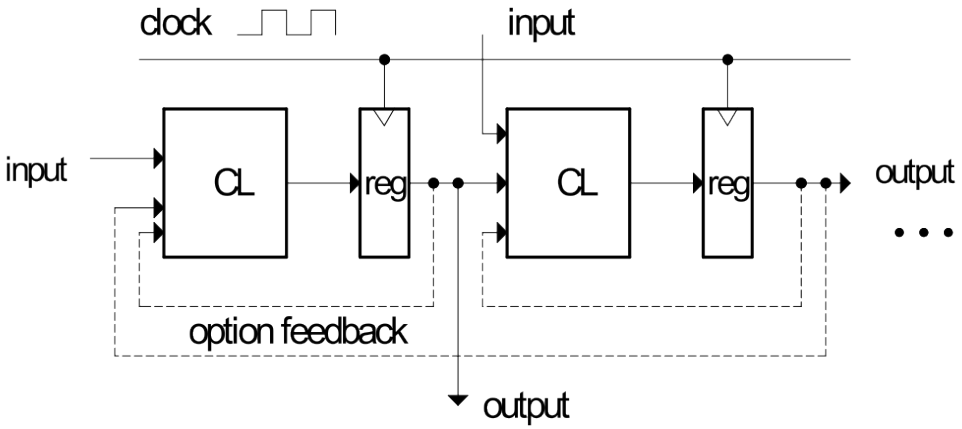
- Changes at the inputs do not instantaneously appear at the outputs
  - There are finite conductances and capacitances in each gate…



- Propagation through a chain of gates is roughly the sum of the delay through the individual gates

Digital Logic Timing

- The longest propagation delay through CL blocks sets the maximum clock frequency



- To increase clock rate:
  - Find the longest path
  - Make it faster

Implementation Alternatives & Design Flow

Implementation Alternative Summary

Full-custom:	All circuits/transistors layouts optimized for application.
Standard-cell:	Small function blocks/"cells" (gates, FFs) automatically placed and routed.
Gate-array (structured ASIC):	Partially prefabricated wafers with arrays of transistors 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, TPU).

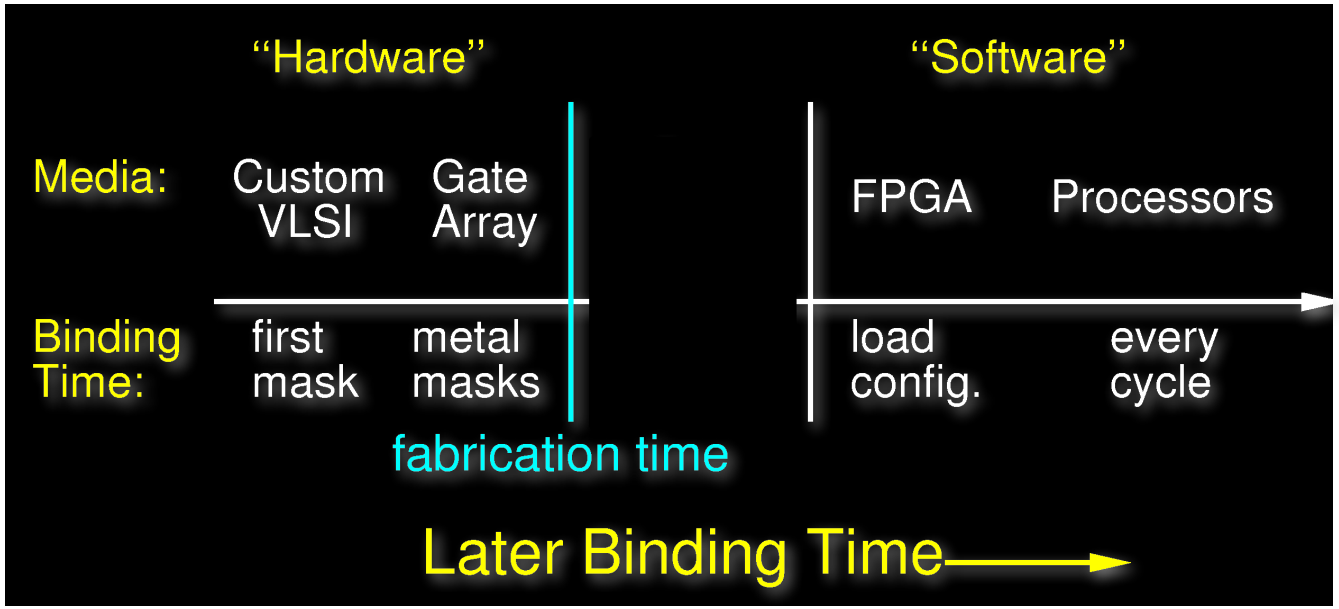
These days, "ASIC" almost always means Standard-cell.

What are the important metrics of comparison?

The Important Distinction

"Instruction" Binding Time (cost of flexibility)

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



A. DeHon

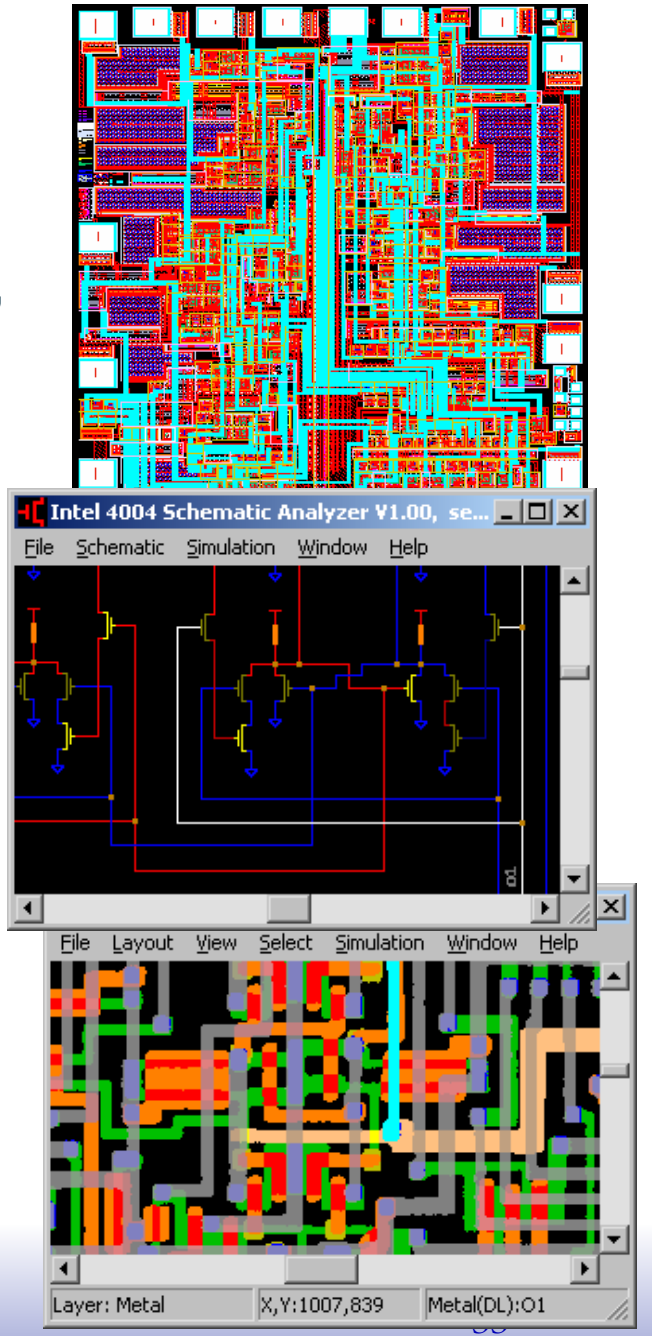
- ***General Principles***

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

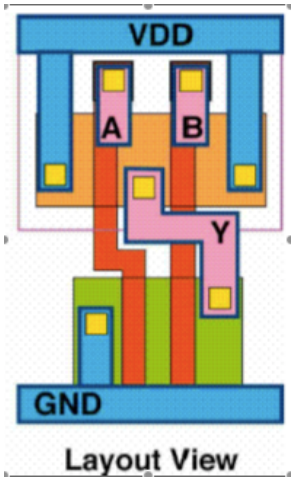
Later the decision is bound, the more flexible the device.

Full-Custom

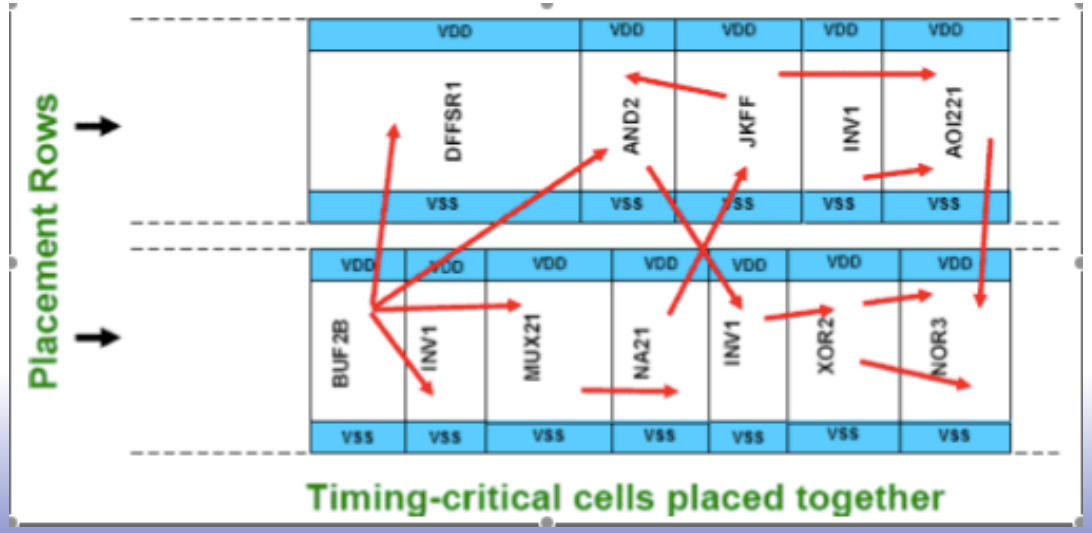
- Circuit styles and transistors are custom sized and drawn to optimize die, size, power, performance.
- High NRE (non-recurring engineering) costs
  - Time-consuming and error prone layout
- Hand-optimizing the layout can result in small die for low per unit costs, extreme-low-power, or extreme-high-performance.
- 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.

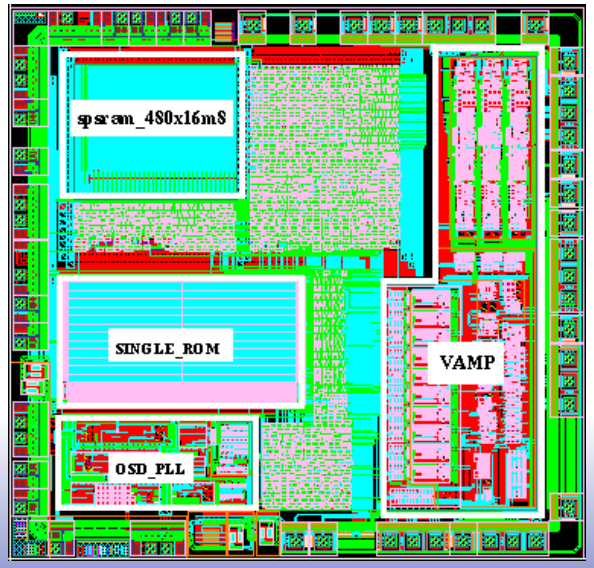


Standard-Cell*



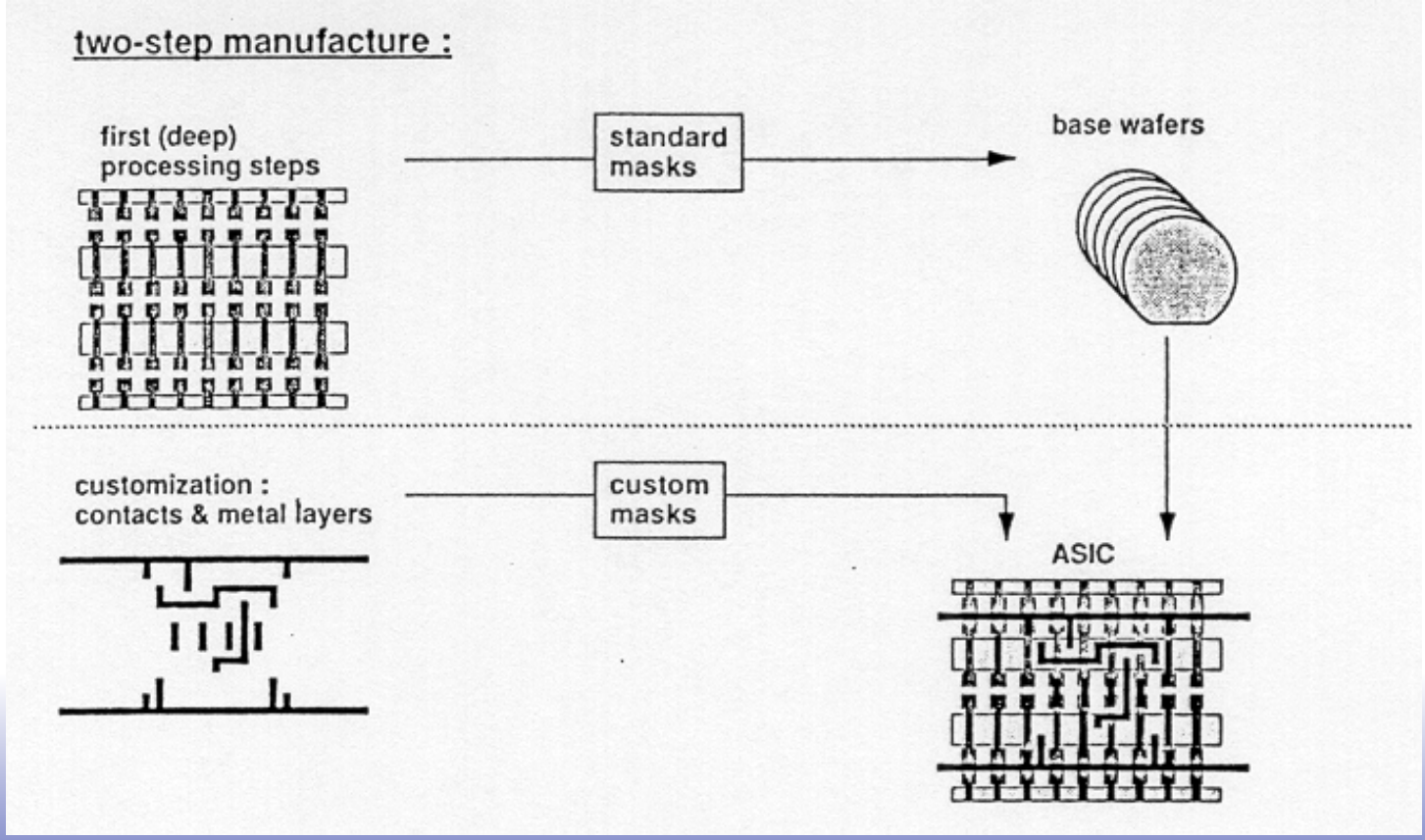
- Based around a set of pre-designed (and verified) cells
  - Ex: NANDs, NORs, Flip-Flops, counters slices, buffers, …
- Each cell comes complete with:
  - layout (perhaps for different technology nodes and processes),
  - Simulation, delay, & power models.
- Chip layout is automatic, reducing NREs (usually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





Gate Array

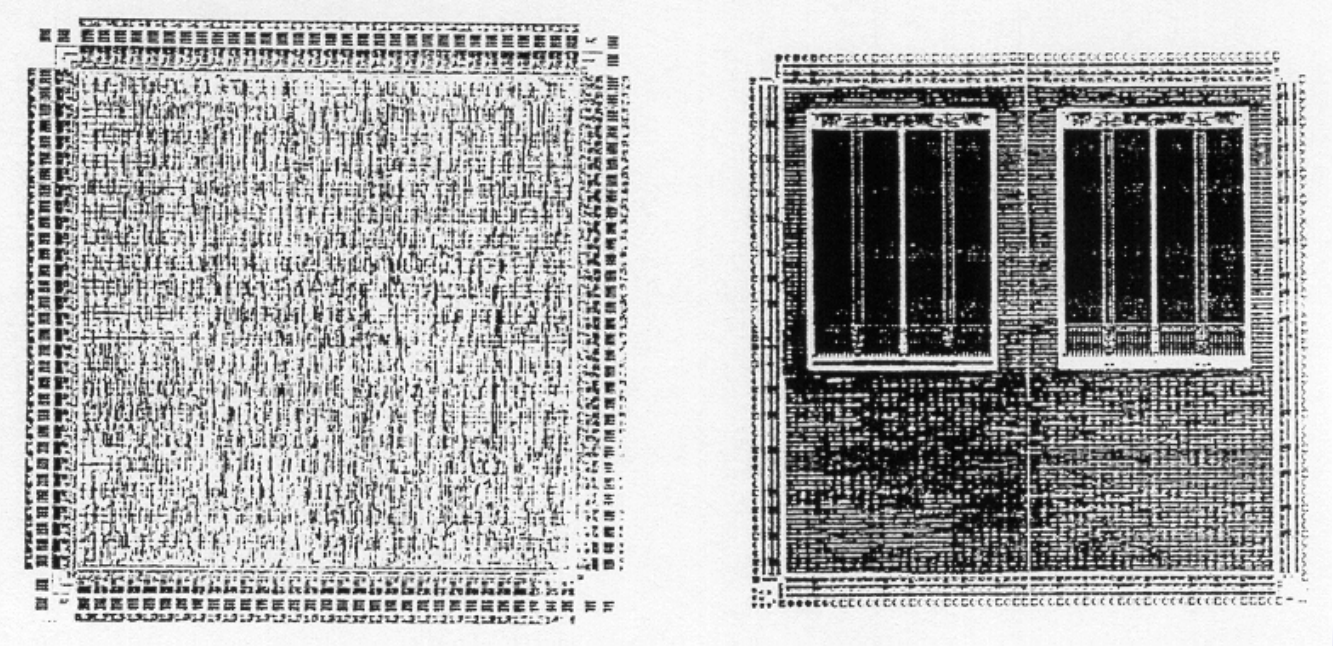
- Prefabricated wafers of, rows of transistors. Customize as needed with “back-end” metal processing (contact cuts, metal wires). Could use a different factory.
- CAD software understands how to make gates and registers.



Gate Array

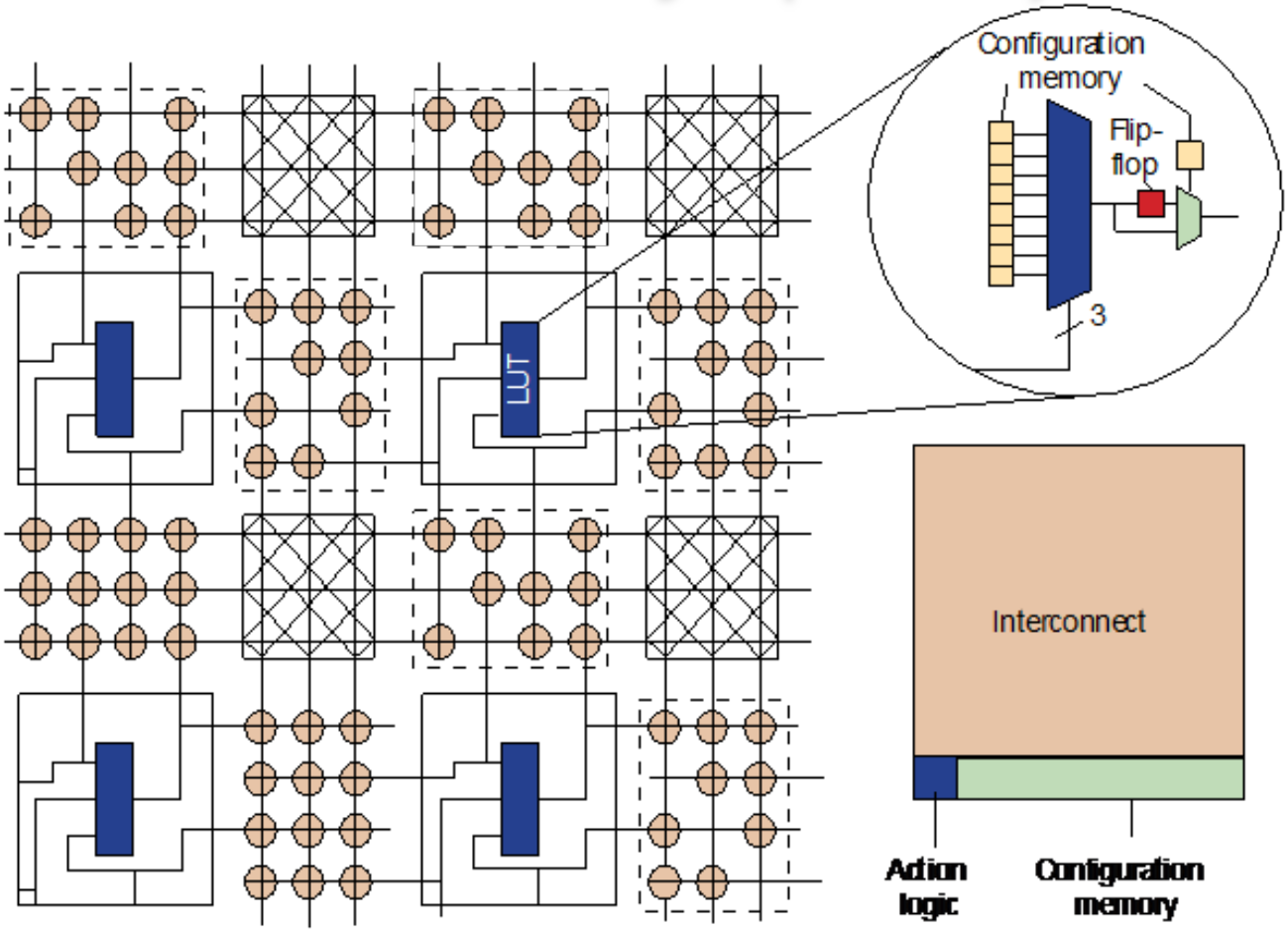
- *Shifts large portion of design and mask NRE to vendor.*
- *Shorter design and processing times, reduced time to market for user.*
- *Highly structured layout with fixed size transistors leads to large sub-circuits (ex: Flip-flops) and higher per die costs.*
- *Memory arrays are particularly inefficient,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

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



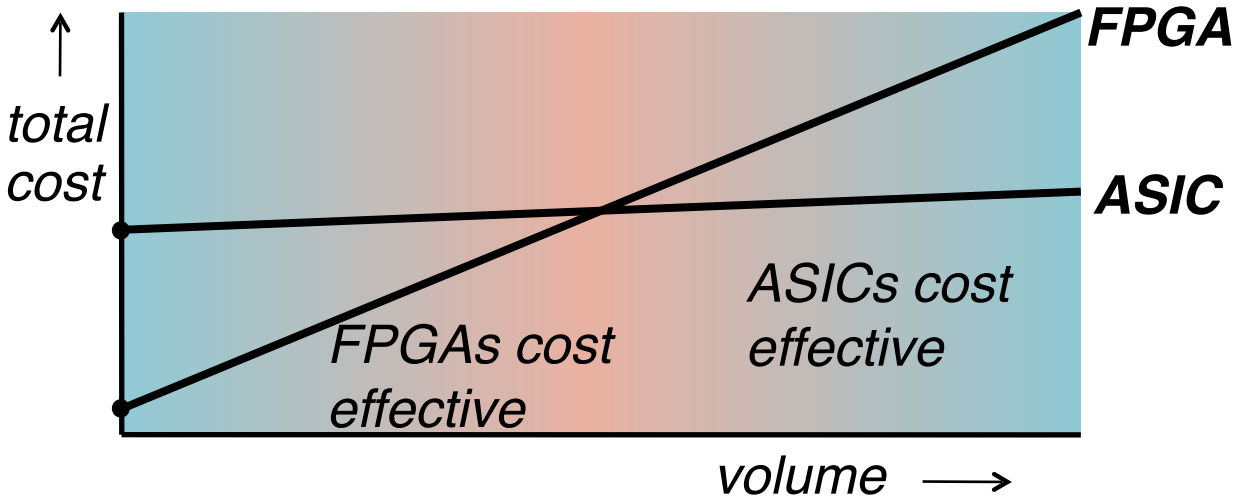
Field Programmable Gate Arrays (FPGA)

- *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.*



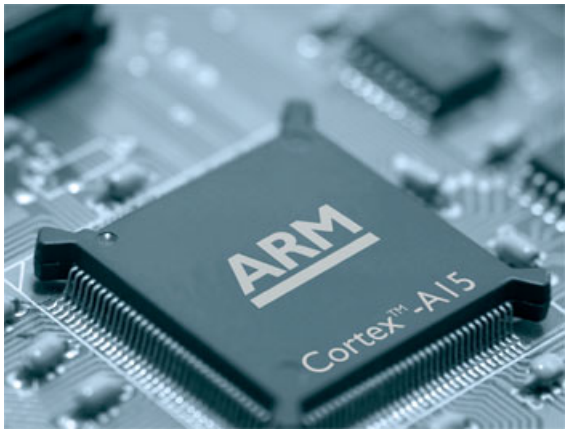
- Fuses, EPROM, or Static RAM cells are used to store the "configuration".
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Many FPGAs include special circuits to accelerate adder carry-chain and many special cores: RAMs, MAC, Enet, PCI, SERDES,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

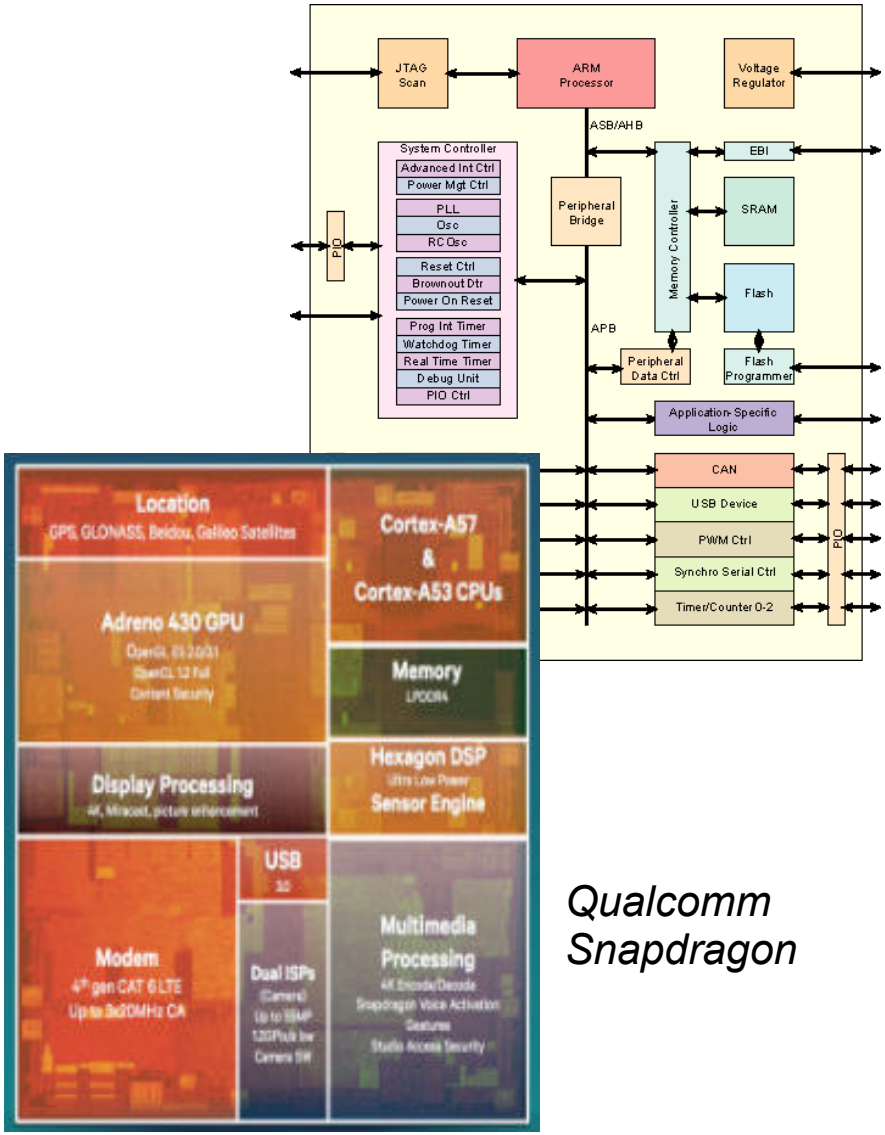


- Where relatively low performance and/or high flexibility is needed, a viable implementation alternative:
  - Software implements desired function
  - "Microcontroller", often with built in nonvolatile program memory and used as single function.
- Furthermore, instruction set processors (microprocessors) are an ubiquitous "abstraction" level.
  - "Synthesizable" RTL model ("soft core", available in HDL)
  - Often mixed into other digital designs
- 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

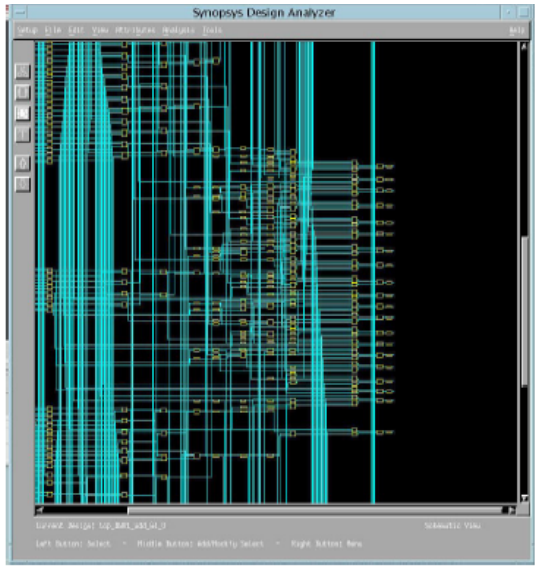
Verilog to ASIC layout flow

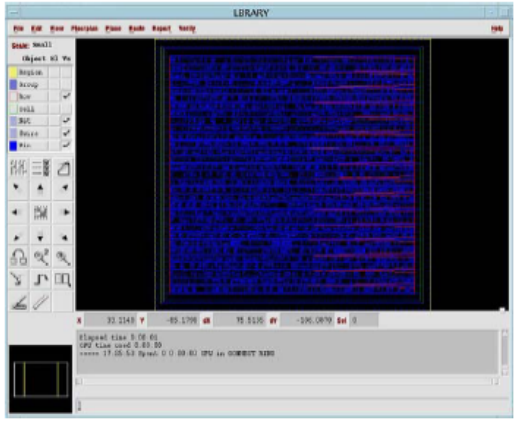
- "push-button" approach

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

  assign sum = a + b;
endmodule
```

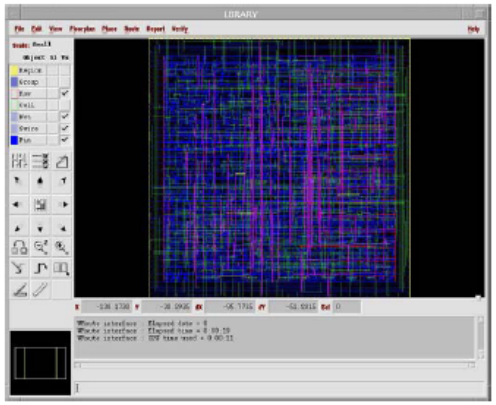
After Synthesis



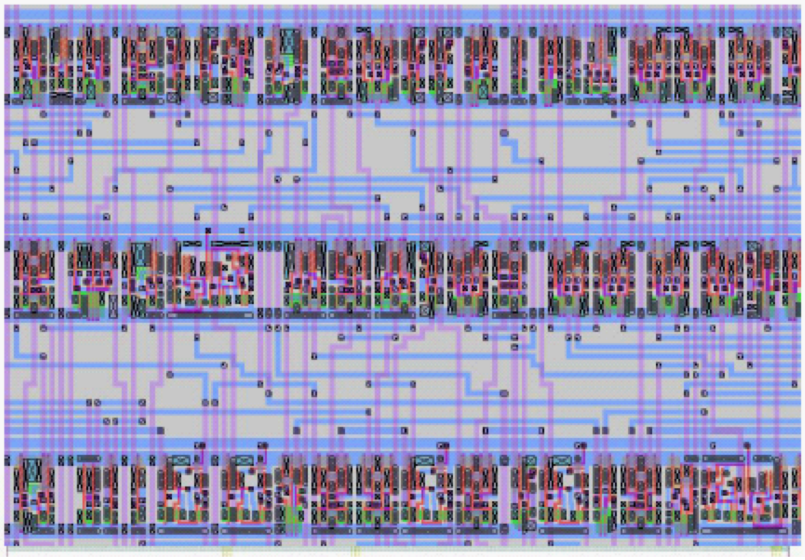


After Placement

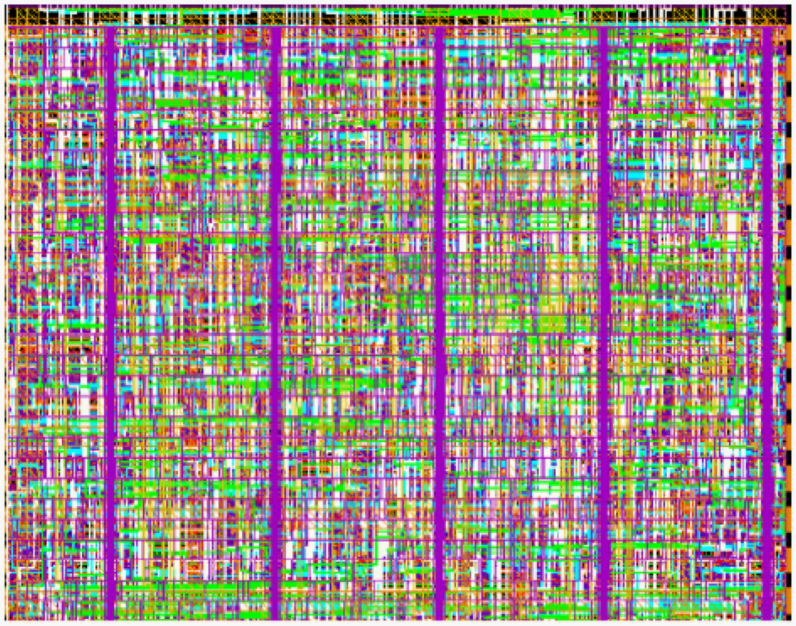
After Routing



Standard cell layout methodology



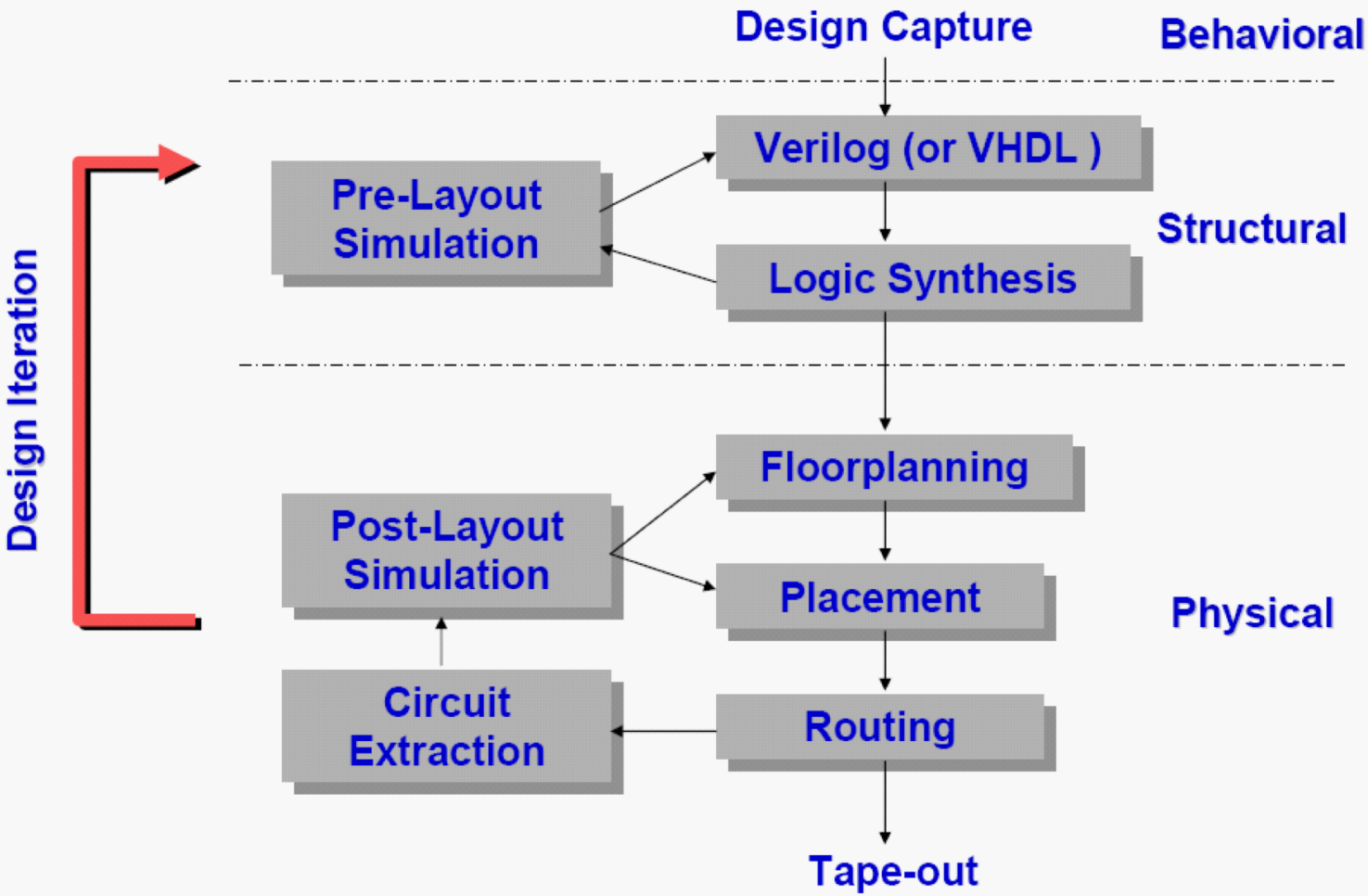
1um, 2-metal process



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

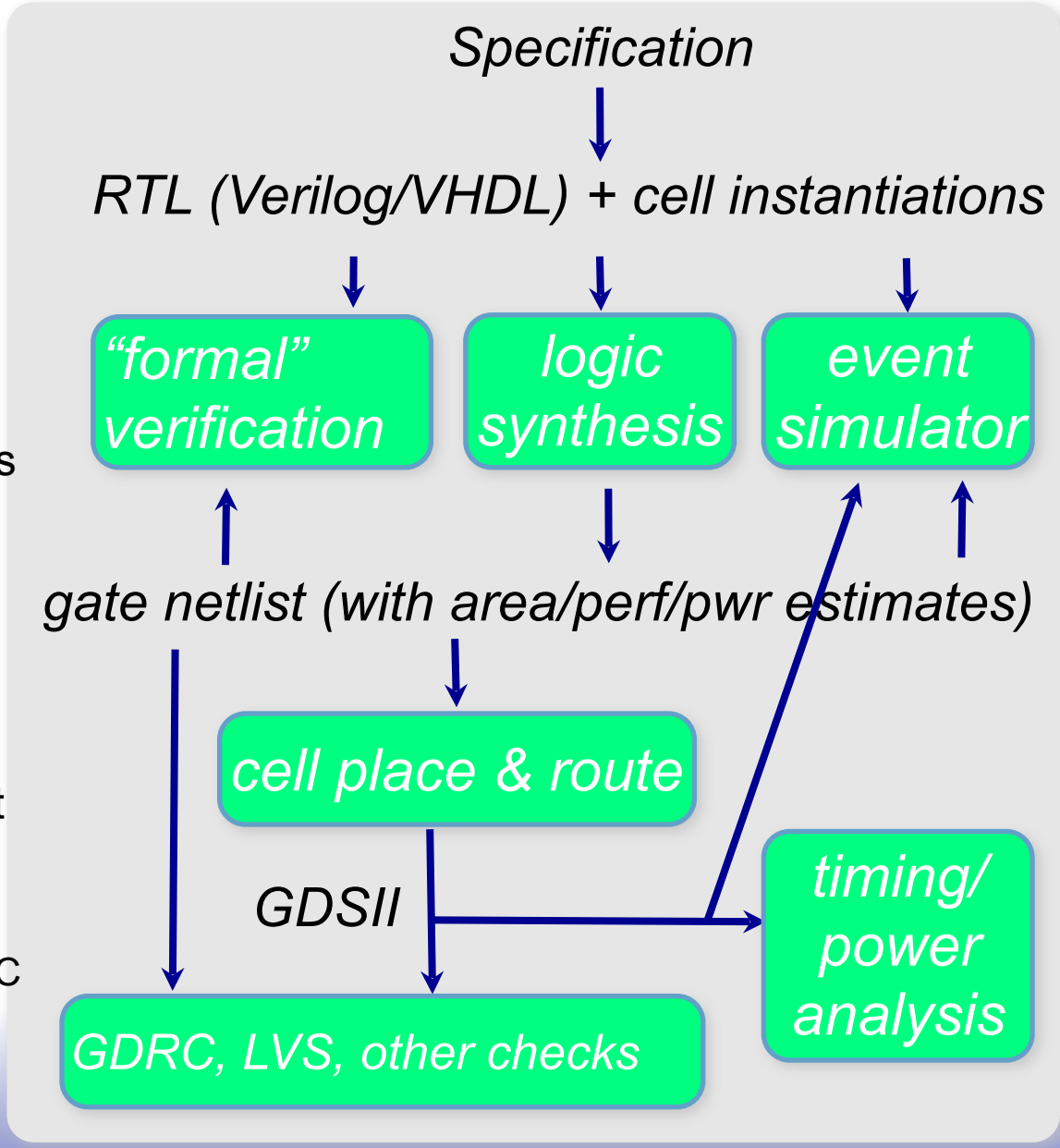
The ASIC flow



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

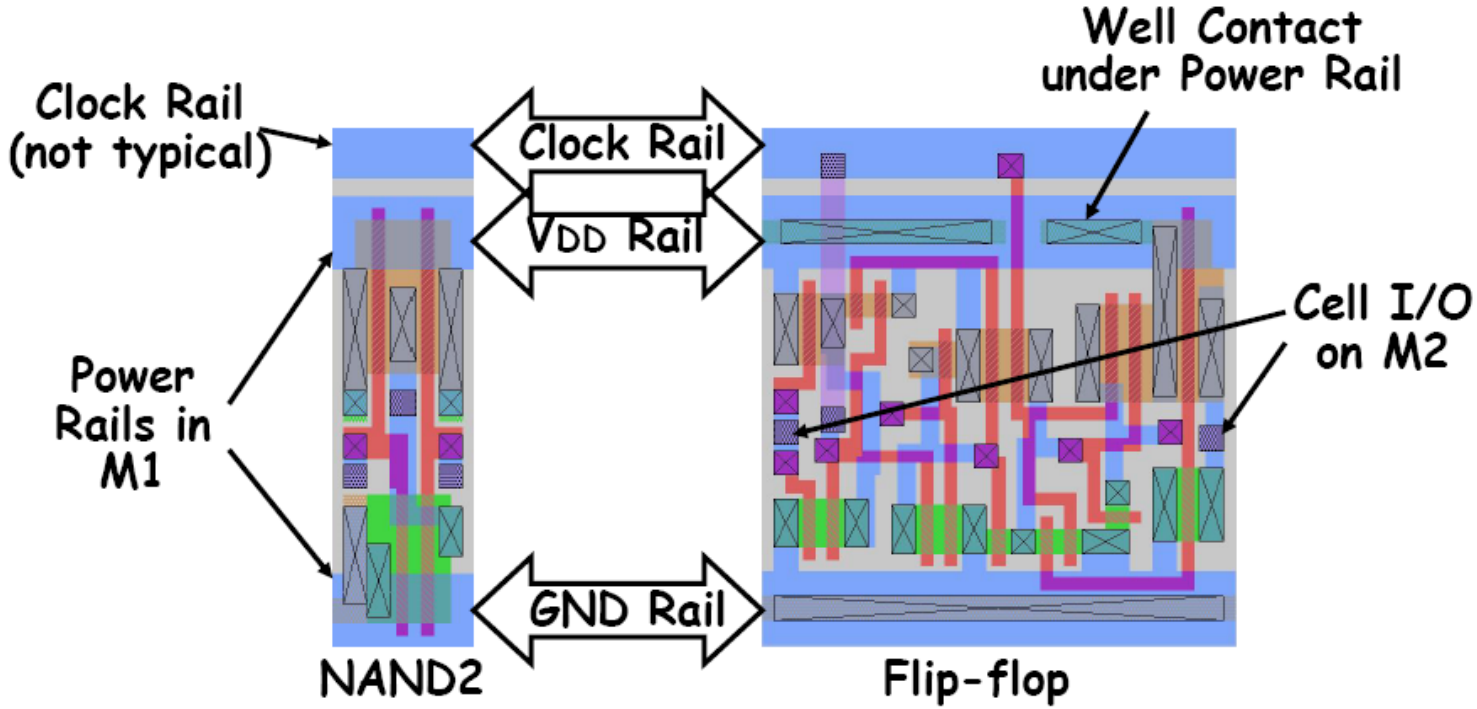


Standard cell design

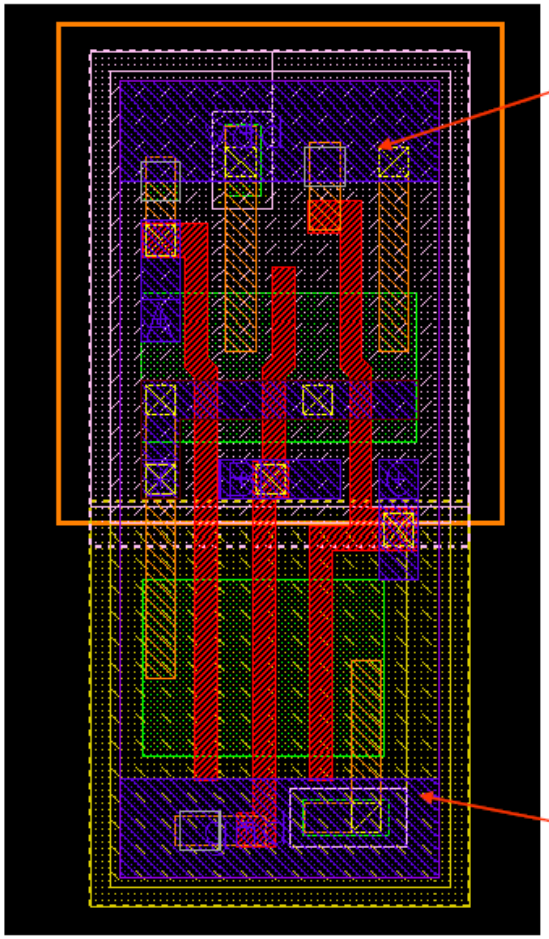
- Layout considerations

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

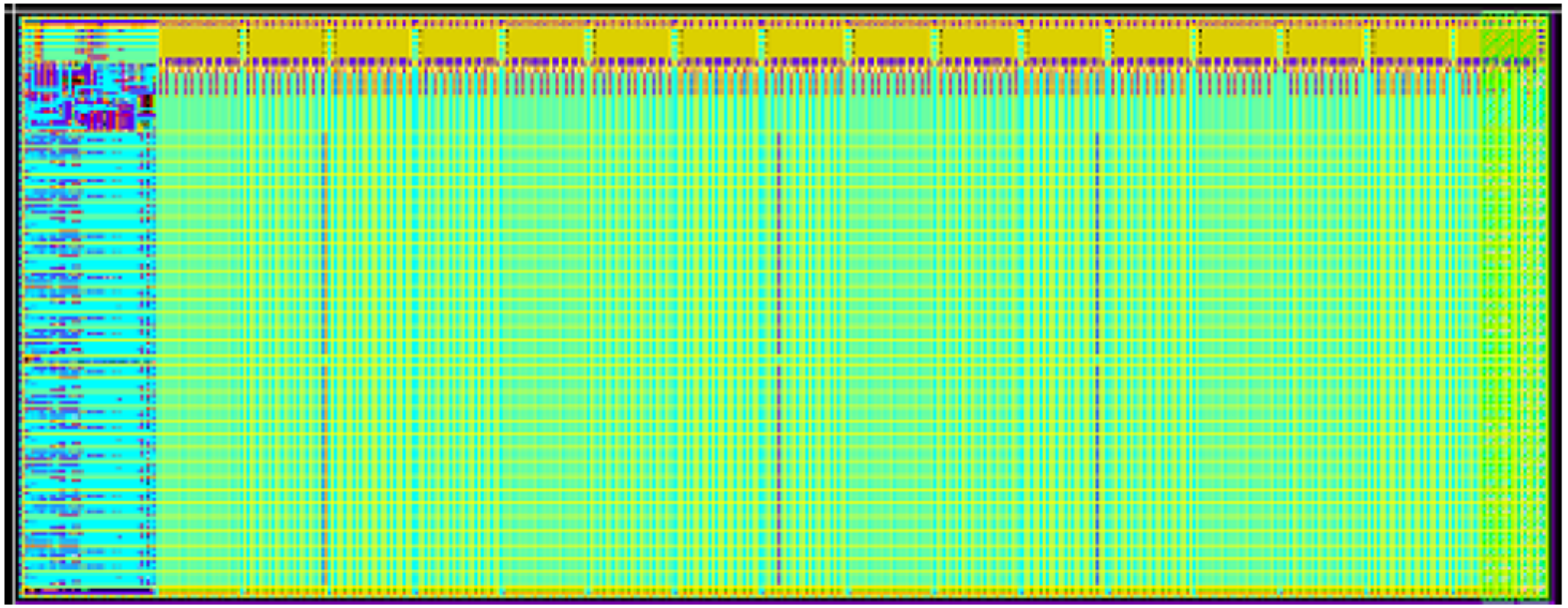
3-input NAND cell
(from ST Microelectronics):
C = Load capacitance
T = input rise/fall time

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

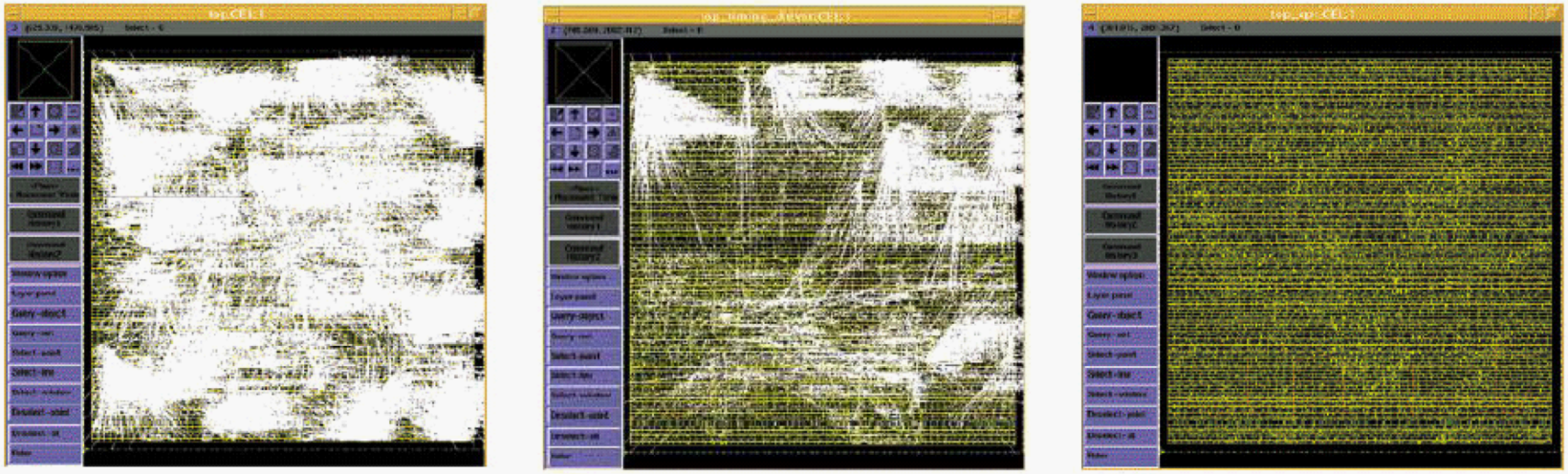
256×32 (or 8192 bit) SRAM Generated by hard-macro module generator



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

The “timing closure” problem

- Biggest problem are wires (signals and clock)



Iterative Removal of Timing Violations (white lines)

End of Lecture 2