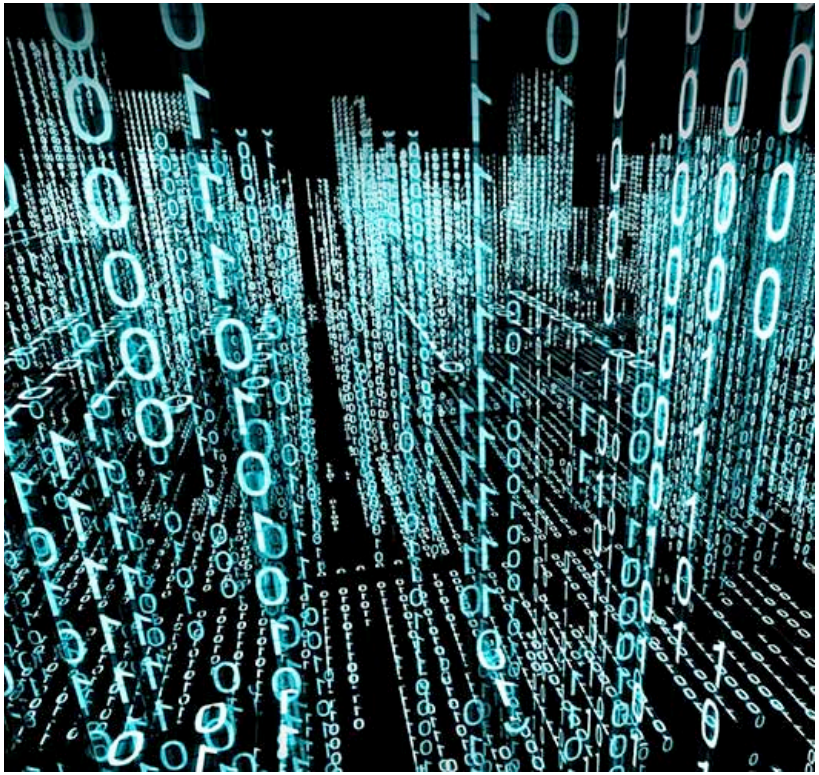
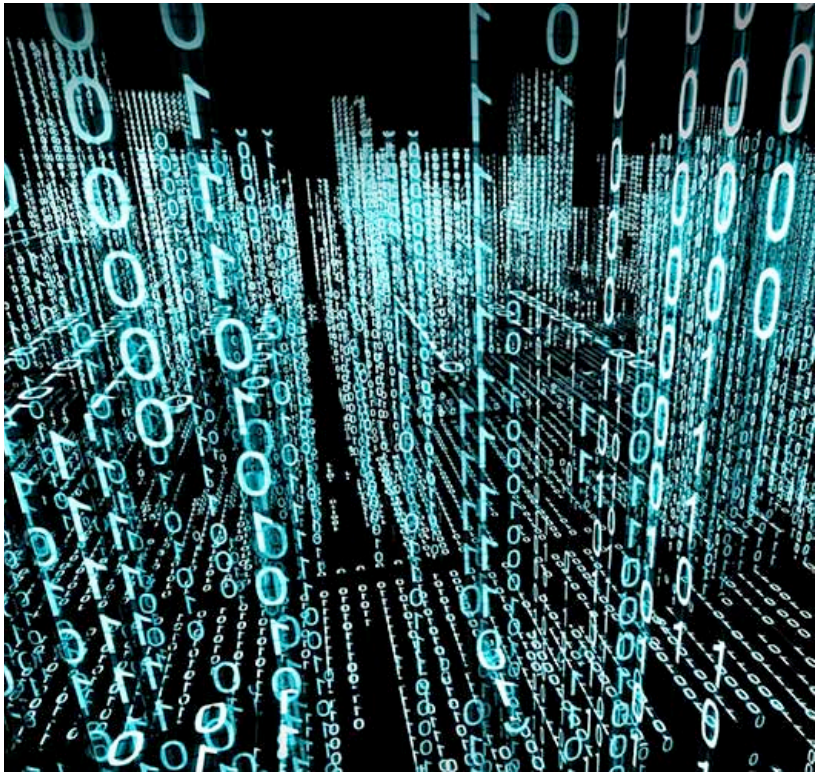


EECS 151/251A
Spring 2023
Digital Design and Integrated Circuits

Instructor:
J. Wawrzynek

Lecture 2: Design



Outline

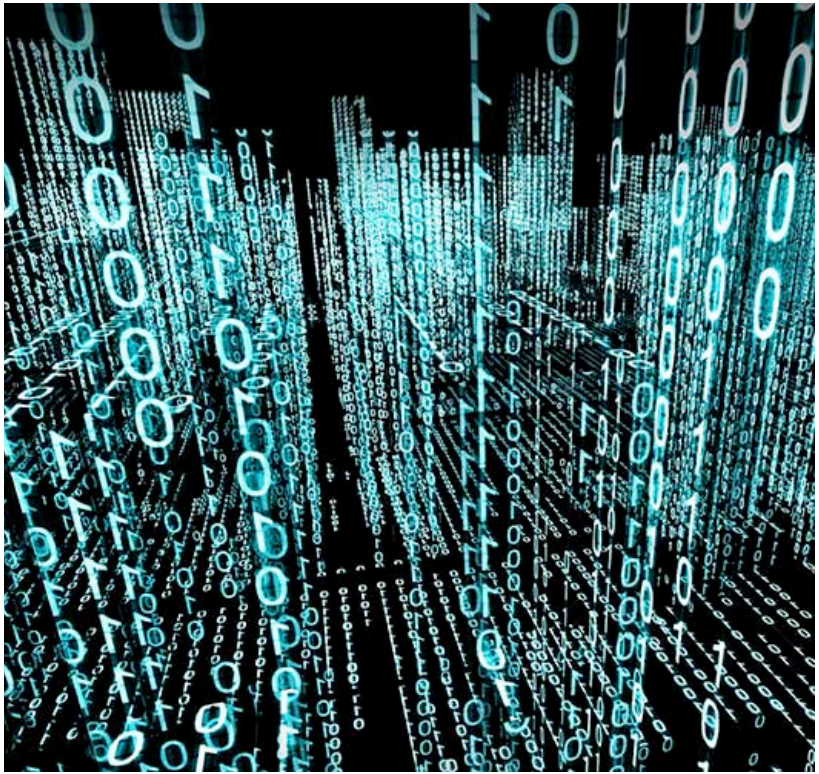
- Digital Design Defined
- Design Challenges and Methodology
- Details of Design Metrics
- Digital Logic – Basic Concepts
- Design Implementation Alternatives
- Design Flows
- ASICs

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.

Announcements

- Wawrzynek office hours Monday 3:30 - 631 Soda
- GSI office hours now posted on website
- Remember, all lab sessions start this week.
- Problem Set 1 posted (easy one), due in 1 week (start early!)
- Concurrent enrollment applicants
  - If not approved yet, see me in person office hour
  - I need to understand your background, situation, and what other courses or experiences you have had that prepare you

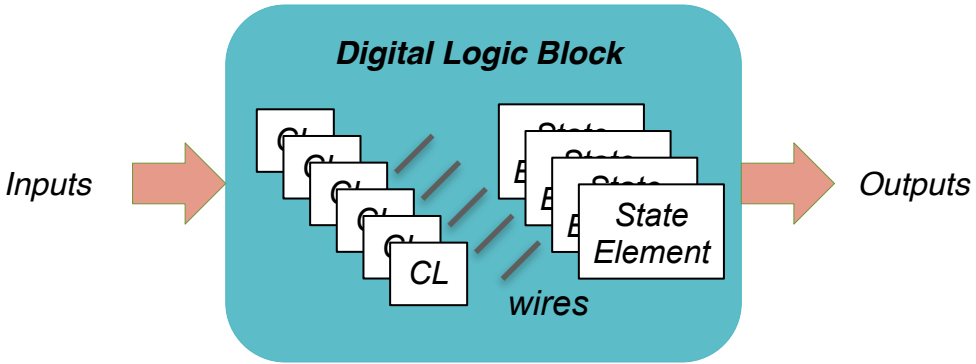


Digital Design

What is "digital design"?

- Given a description of inputs/outputs and function plus performance, cost, & power constraints, come up with an implementation using a set of primitives.

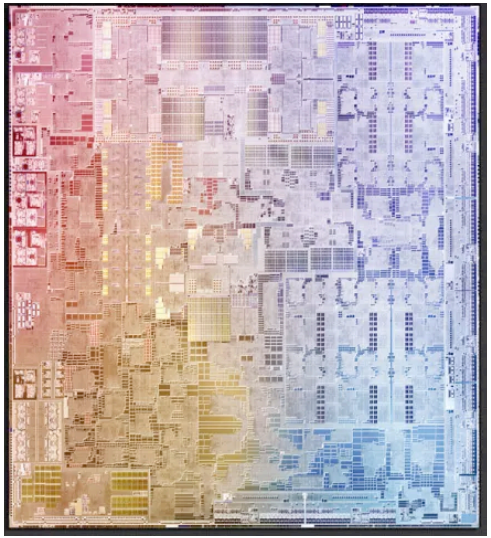
- Digital systems are implemented as a interconnection of *combinational logic and state elements*:



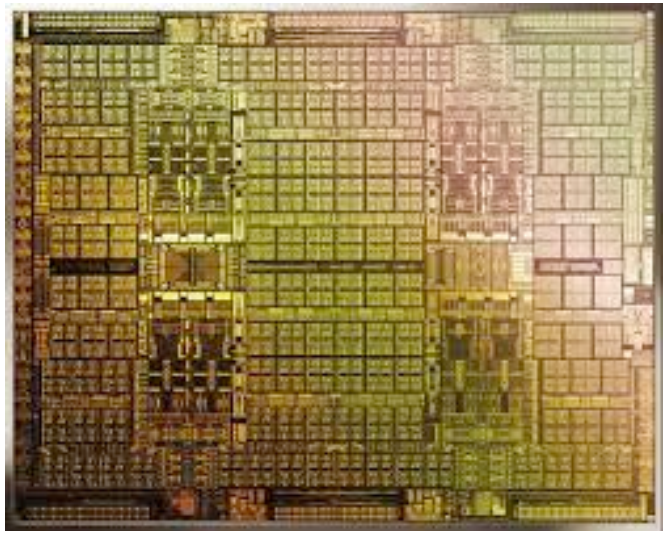
- *What are the challenges in designing a digital system?*

Hardware Design Challenges

1. The HW systems we find useful are huge and the functions we implement are complex:



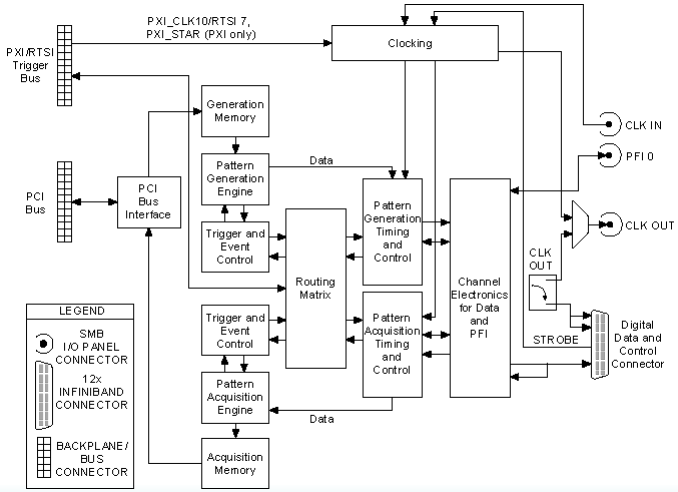
Apple M2/MAX
67 billion transistors

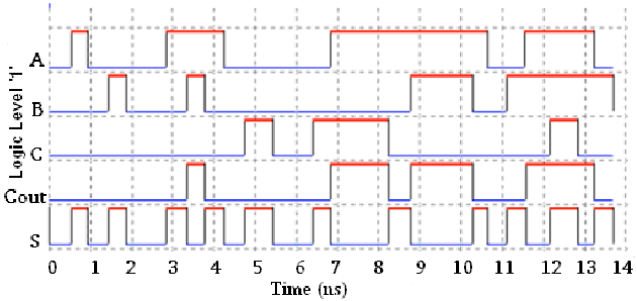


Nvidia A100
54.2 billion transistors

Hardware Design Challenges

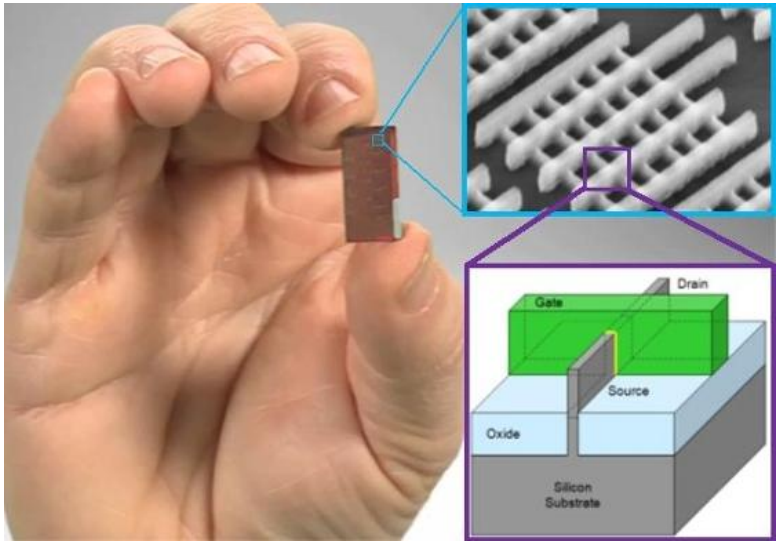
2. Hardware is inherently parallel (the secret sauce for performance scaling). Correct design of parallel systems requires management of timing and synchronization. (Unlike SW that is inherently sequential). To make matters worse, HW is parallel at every level (transistors, gates, blocks, CPUs, Memories, etc.)





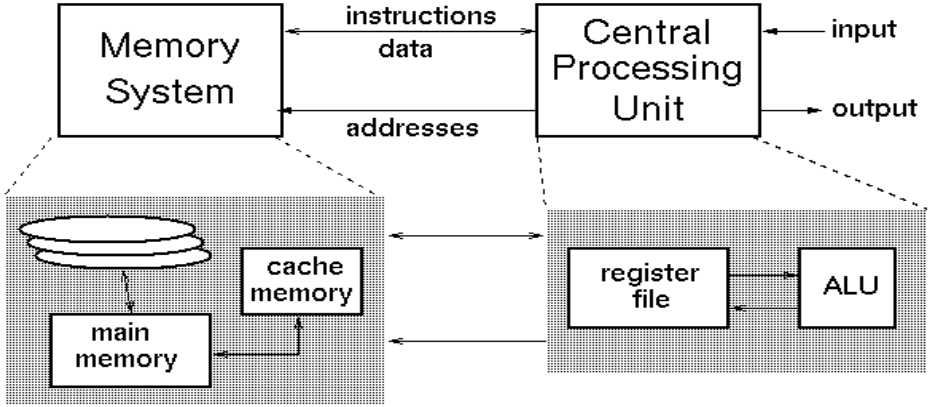
Hardware Design Challenges

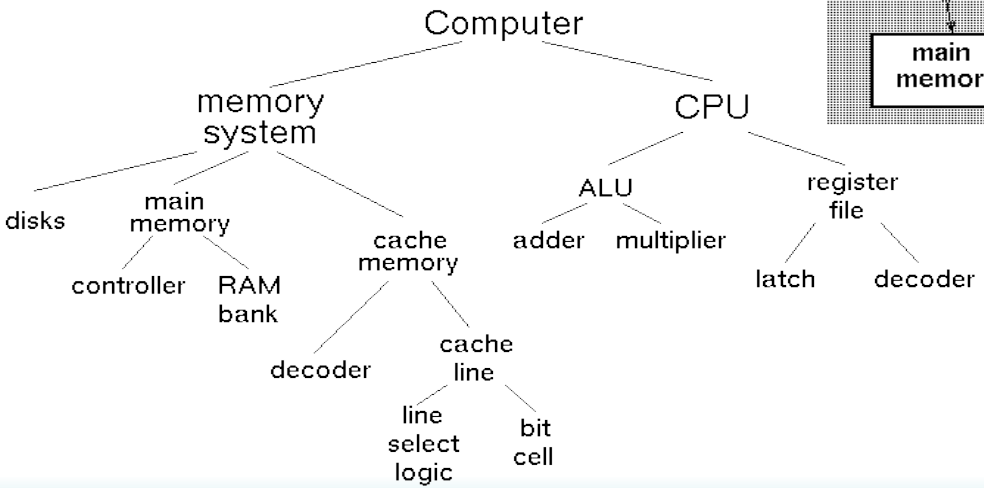
3. The technology we use imposes physical constraints that effect *cost*, *speed*, and *power*.
   - The wires we use to connect components, on and off chip, consume power, take area, and slow signaling
   - The transistors we use to implement components, consume power, take area, and slow signaling.
   - Making design decisions that find the right combination of power, performance, and cost involves an exploration of a huge design space.



HW Design is Hard

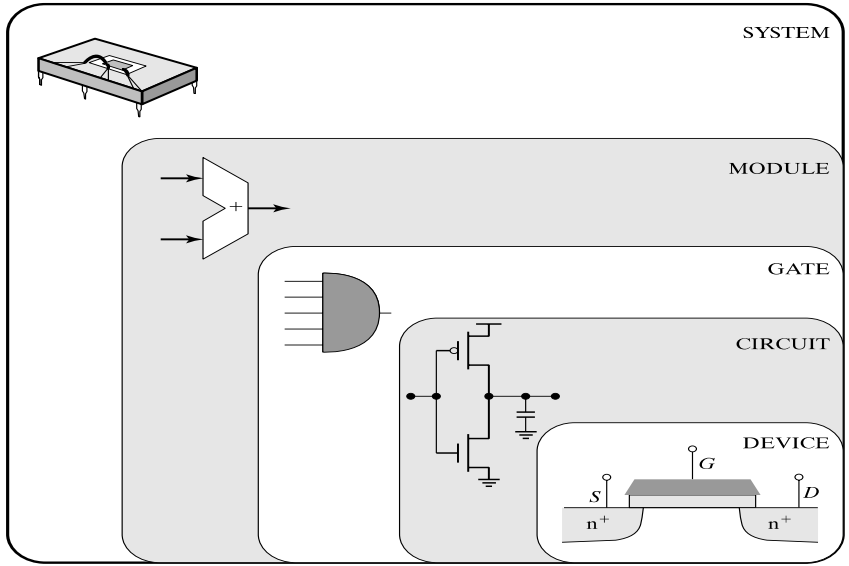
1. Hierarchical Design Representations

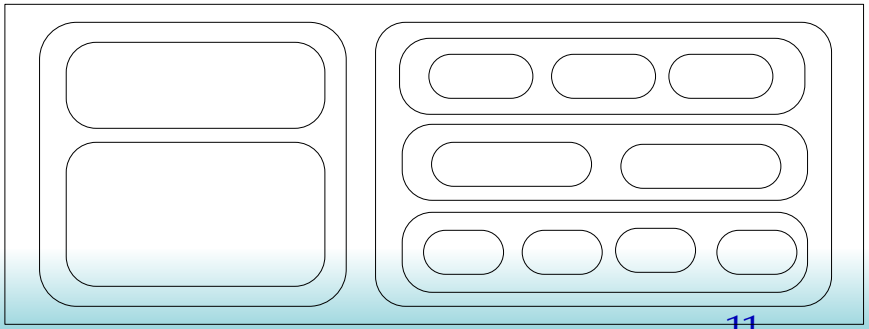




HW Design is Hard

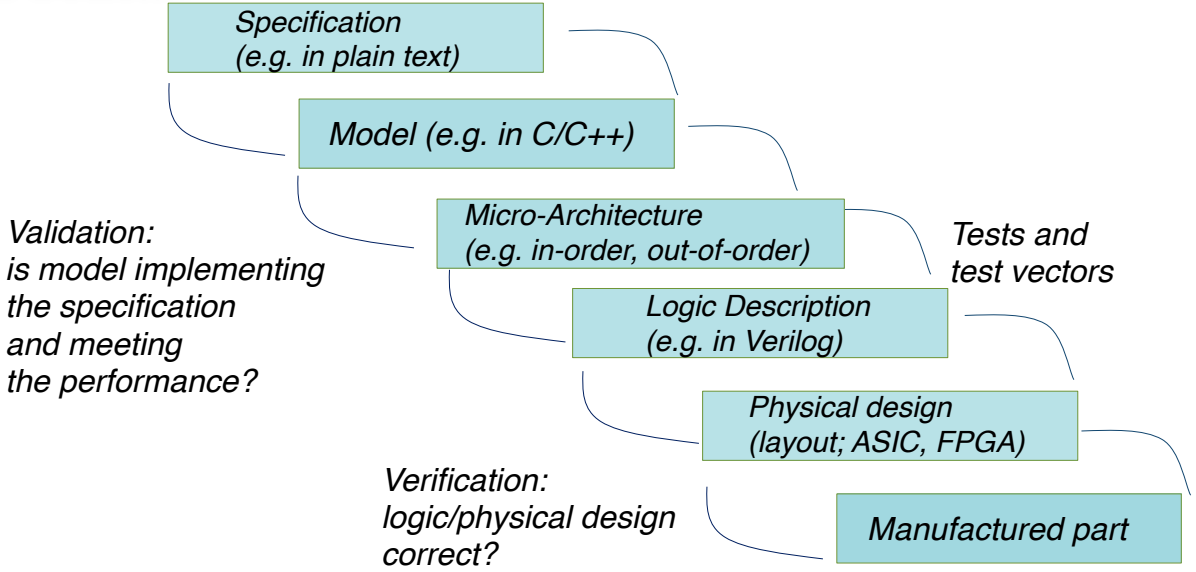
2. Design abstraction
   - hides details and reduce number of things to handle at any time
3. Modular design
   - Permits Divide and conquer
   - Simplifies implementation and debugging





Do these help with timing and optimization?

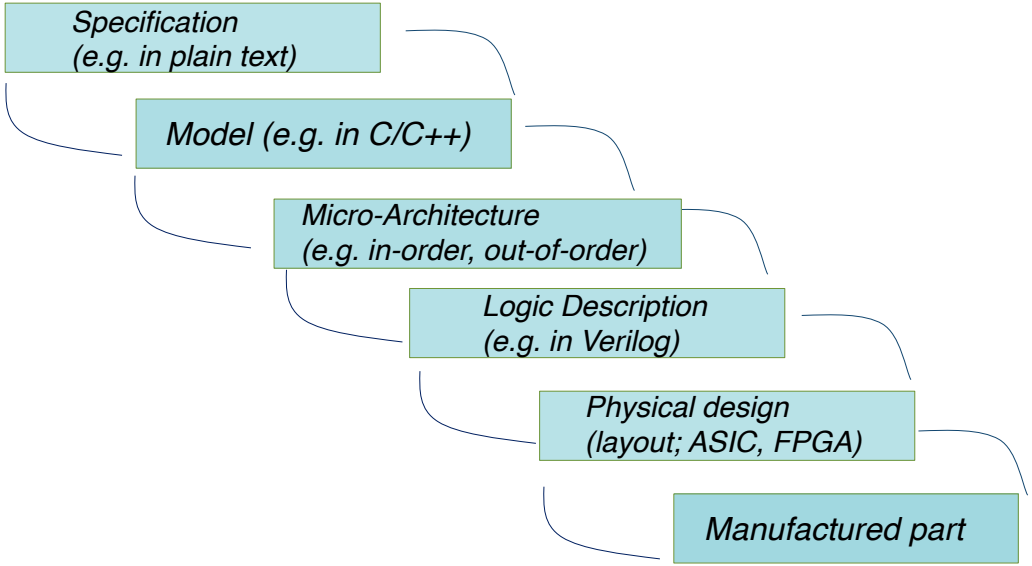
Design Challenges are met by using layers of abstractions. The design process moves from top to bottom.



Validation: Have we built the correct 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.

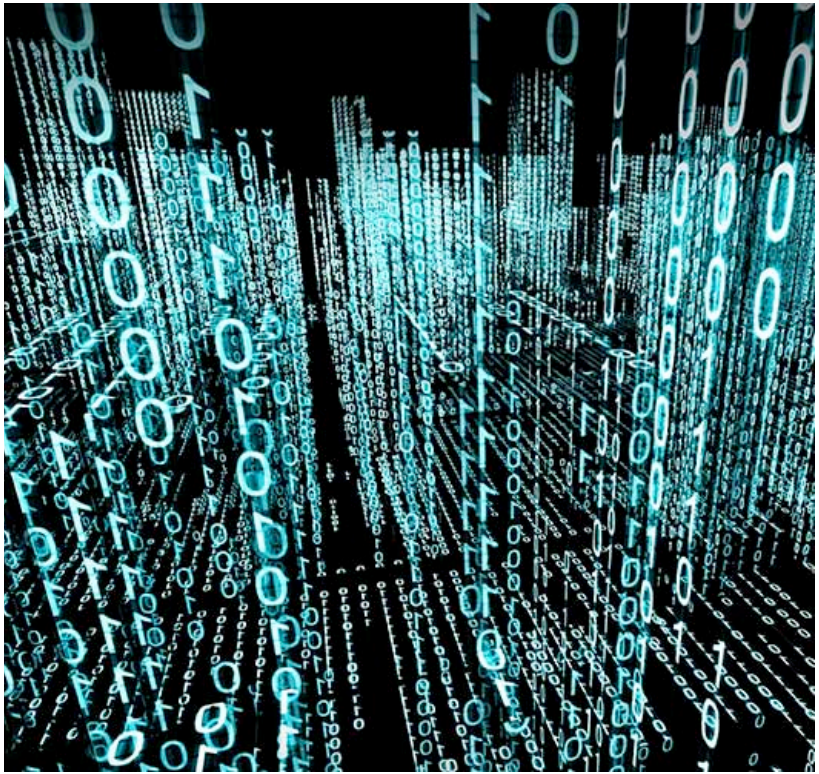
Design Challenges are met by through layers of abstractions.



Computer Aided Design (CAD) tools are used:

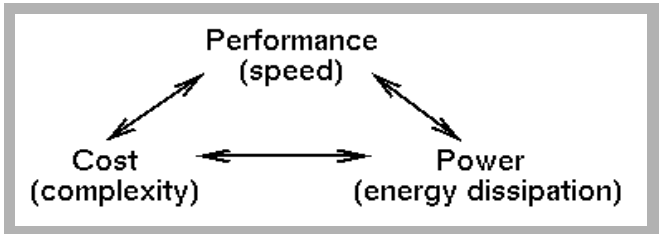
1. at each layer to check correctness and constraint satisfaction

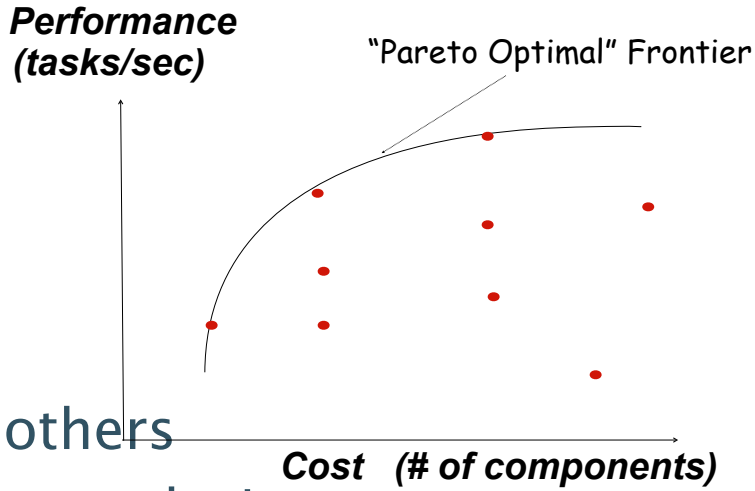
2. to help convert from one abstraction layer to another



Design Metrics

Basic Design Tradeoffs





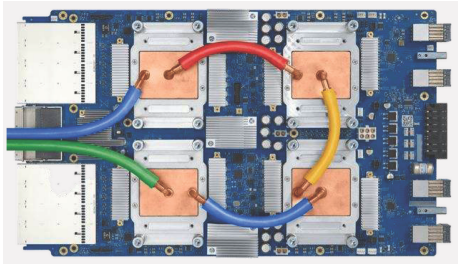
- Improve on one at the expense of the others
- Tradeoffs exist at every level in the system design
- Design Specification
  - Functional Description
  - Performance, cost, power constraints
- Designer must make the tradeoffs needed to achieve the function within the constraints

Other Design Metrics of Interest

- Non-recurring engineering (NRE) costs
  - Cost to develop a design (product) - people, tools, fabrication masks
  - Amortized over all units shipped
- Time-to-market (TTM)
  - First to market is often the winner, even if not the best design
- Flexibility
  - Can the device be reprogrammed for different tasks?
  - Always comes at the expense of area cost, energy efficiency, and performance
- Reliability
  - Does the device fail in the field?

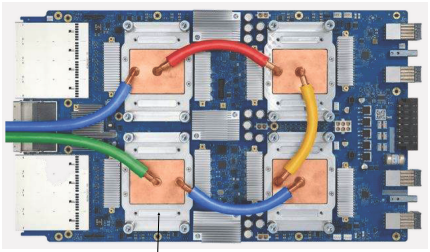
Performance

- **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 public 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, …



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

aka recurring cost

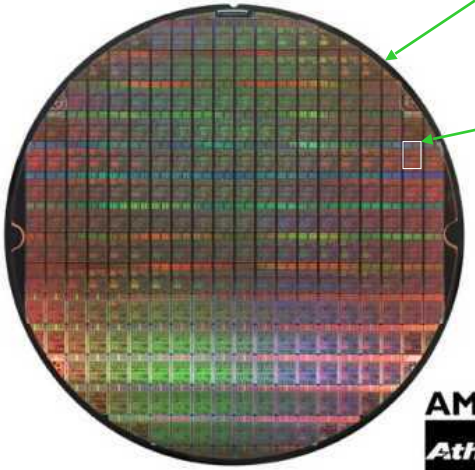
aka NRE cost

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

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

Die Cost

Single die



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

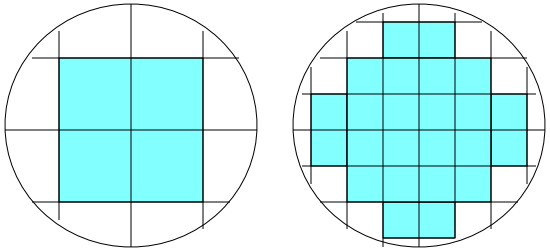
From: http://www.amd.com

Yield & Die Cost

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

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

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

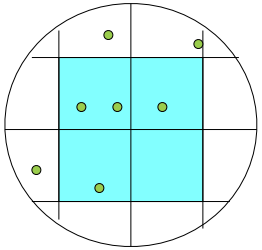


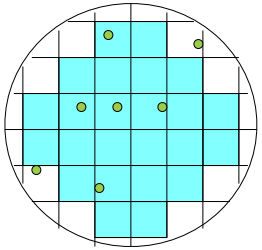
How do we calculate "die yield"?

Defects

Examples:

Yield = 0.25





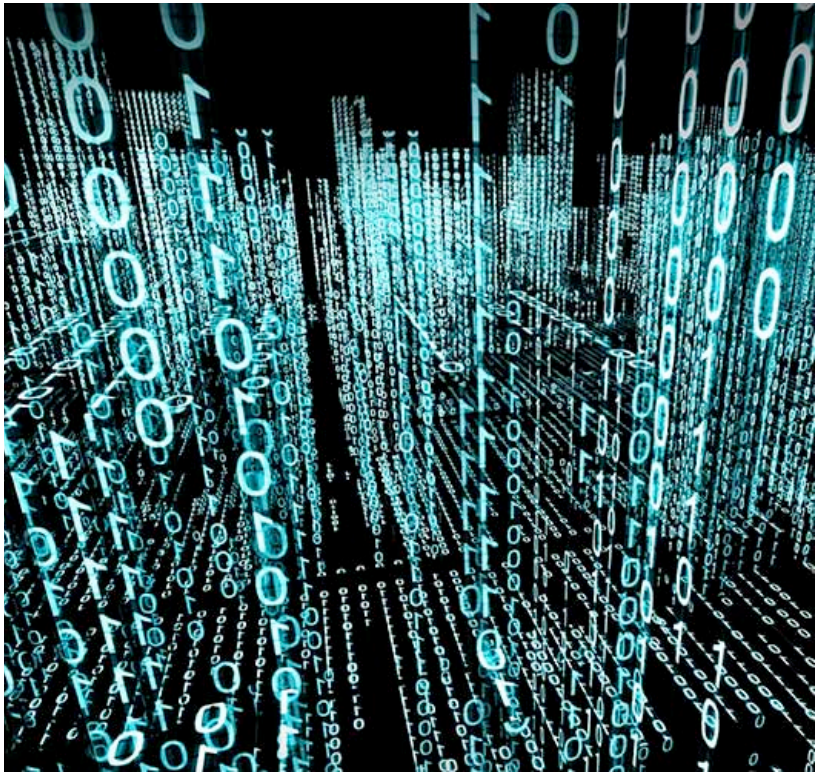
Yield = 0.76

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

α is approximately 3

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

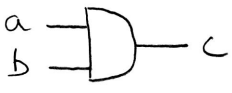
Given *defect density* and die area can compute yield and therefore cost.



Digital Logic
Basic Concepts

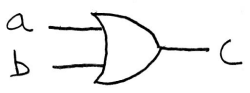
Common Logic Gates

AND



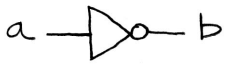
ab	c
00	0
01	0
10	0
11	1

OR



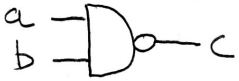
ab	c
00	0
01	1
10	1
11	1

NOT



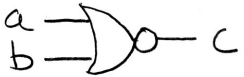
a	b
0	1
1	0

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?

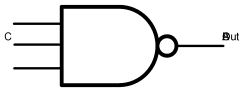
Multi-Input Gates

3-Input NAND

NAND3 *Boolean equation*

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

NAND3



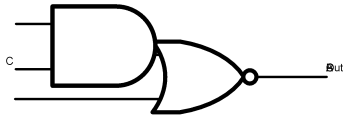
NAND3

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*

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



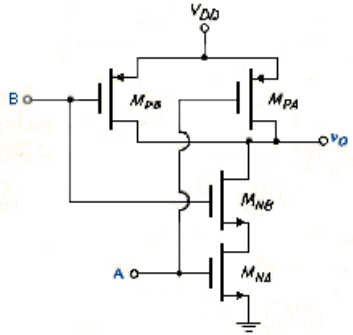
And-Or-Invert

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

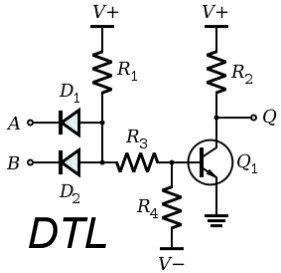
- Single gate in modern CMOS usually doesn't have more than 3-4 inputs

Logic Gate Implementation

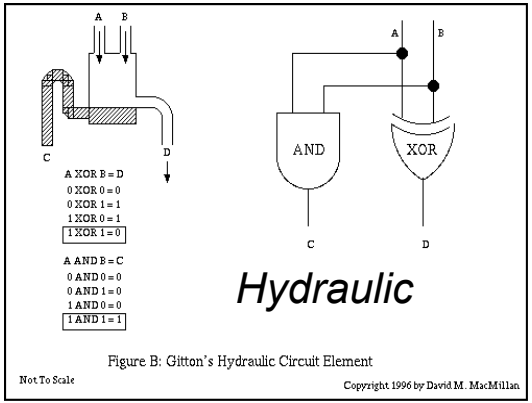
- Logic circuits have been built out of many different technologies. If we have a basic logic gate (AND or OR) and inversion we can build a complete logic family.



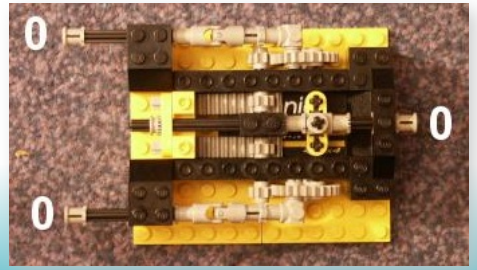
CMOS Gate



DTL



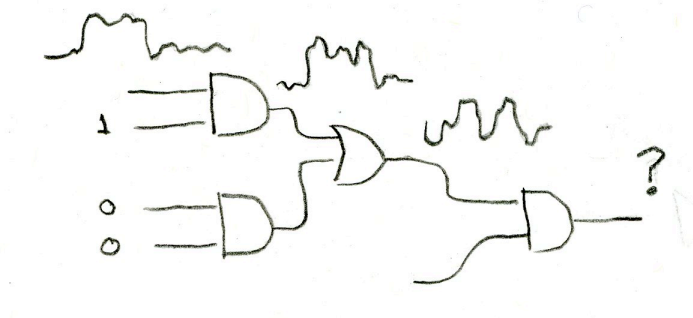
Hydraulic



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

Restoration/Regeneration

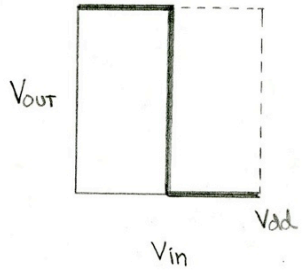
- A necessary property of any suitable technology for logic circuits is "Restoration" or “Regeneration”
- Circuits need:
  - to ignore noise and other non-idealities at the their inputs, and
  - generate "cleaned-up" signals at their output.
- Otherwise, each stage propagates input noise to their output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



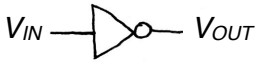
Logic Gate Restoration

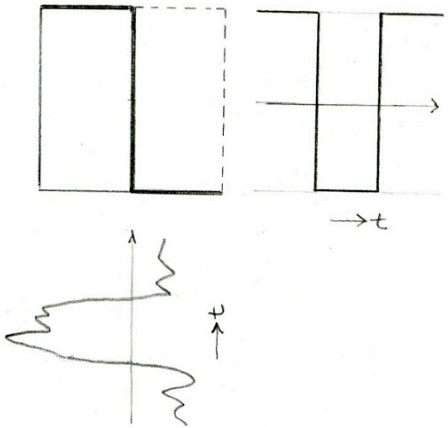
Example (look at 1-input gate, to keep it simple):

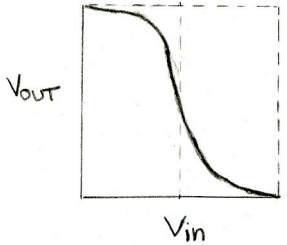
- Inverter acts like a “non-linear” amplifier
- The non-linearity is critical to restoration
- Other logic gates act similarly with respect to input/output relationship.



Idealize Inverter

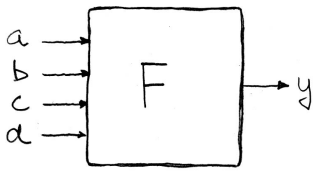






Actual Inverter voltage transfer characteristic (VTC)

Combinational Logic Blocks



Example four-input Boolean 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.

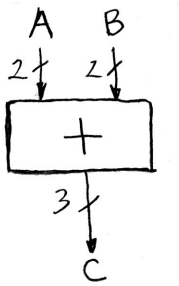
a b c d	y
0 0 0 0	F(0,0,0,0)
0 0 0 1	F(0,0,0,1)
0 0 1 0	F(0,0,1,0)
0 0 1 1	F(0,0,1,1)
0 1 0 0	F(0,1,0,0)
0 1 0 1	F(0,1,0,1)
0 1 1 0	F(0,1,1,0)
1 1 1 1	F(0,1,1,1)
1 0 0 0	F(1,0,0,0)
1 0 0 1	F(1,0,0,1)
1 0 1 0	F(1,0,1,0)
1 0 1 1	F(1,0,1,1)
1 1 0 0	F(1,1,0,0)
1 1 0 1	F(1,1,0,1)
1 1 1 0	F(1,1,1,0)
1 1 1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

Truth Table

Example CL Block

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

$C = A + B$



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

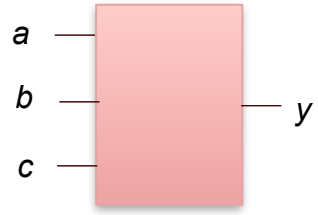
Truth Table

a1 a0 b1 b0	c2 c1 c0
0 0 0 0	000
0 0 0 1	001
0 0 1 0	010
0 0 1 1	011
0 1 0 0	001
0 1 0 1	010
0 1 1 0	011
1 1 1 1	100
1 0 0 0	010
1 0 0 1	011
1 0 1 0	100
1 0 1 1	101
1 1 0 0	011
1 1 0 1	100
1 1 1 0	101
1 1 1 1	110

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

Example Logic Circuit

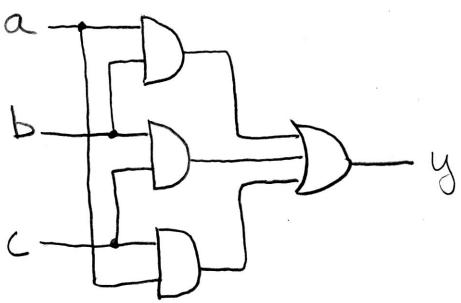
How do we know that these two representations are equivalent?



block diagram

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

Truth Table

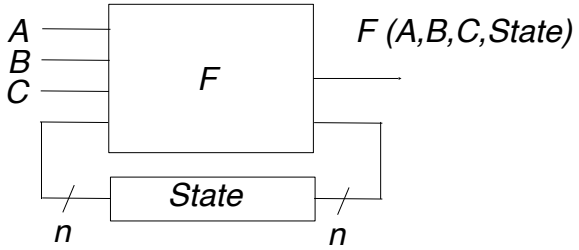


One possible logic gate implementation

Will come back to this later!

Sequential Logic Blocks

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

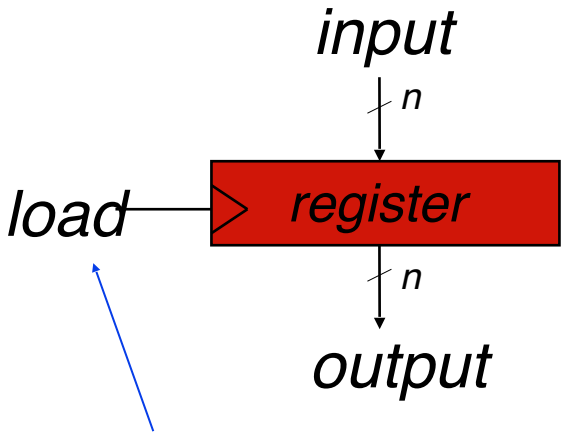


- "State" stored as memory.
- State is a function of previous inputs.
- In synchronous digital systems, state is updated on each clock tick.
- "F" is just a combinational logic block.

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

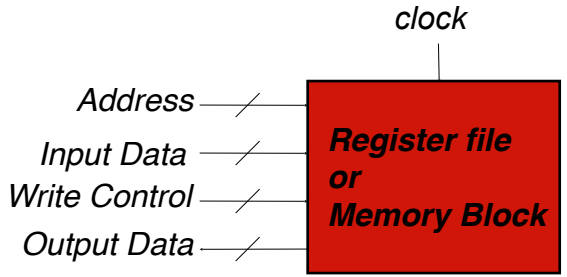
State Elements: circuits that store info

- 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.



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.

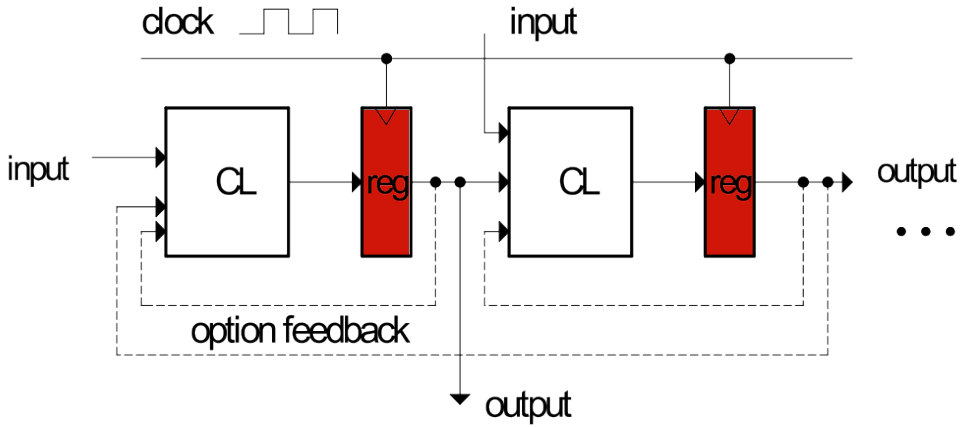


Sometimes used in large groups for "long-term" data storage.

Register Transfer Level Abstraction (RTL)

Any synchronous digital circuit can be represented with:

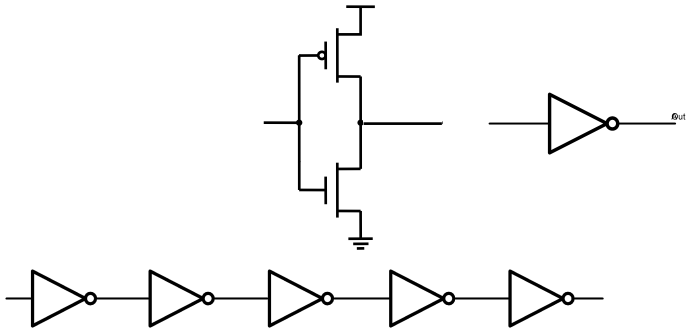
- Combinational Logic Blocks (CL), plus
- State Elements (registers or memories)



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

Digital Logic Delay

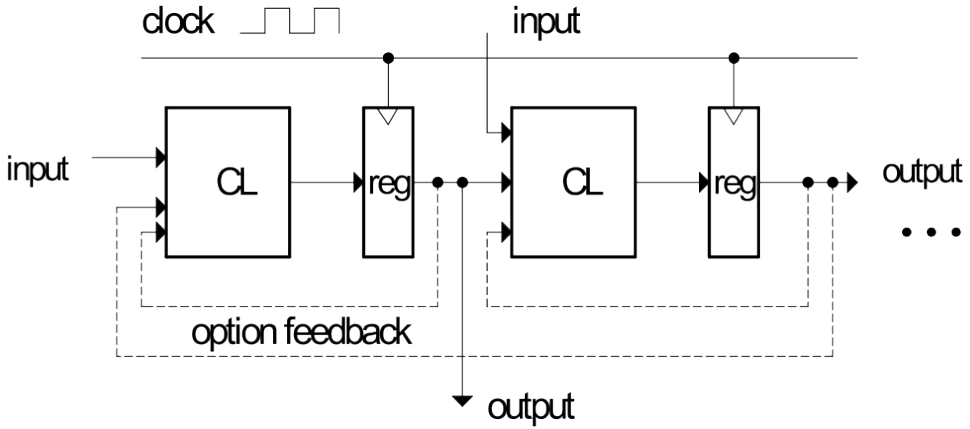
- Changes at the inputs do not instantaneously appear at the outputs
  - There are finite conductances and capacitances in each gate…



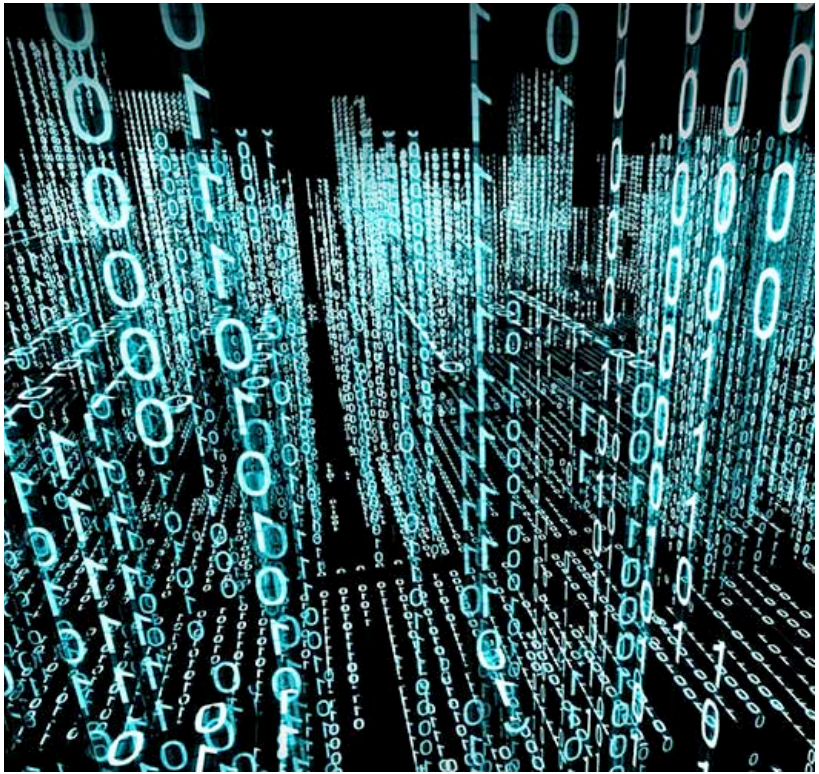
- *Propagation through a chain of gates is roughly the sum of the delay through the individual gates*

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

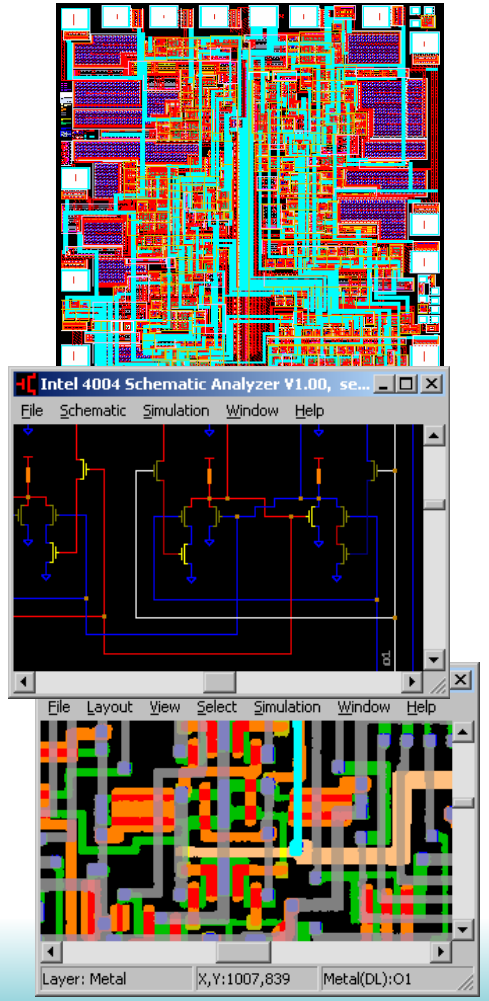
Implementation Alternative Summary

Full-custom:	All circuits/transistors layouts optimized for application.
Standard-cell (ASIC):	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).

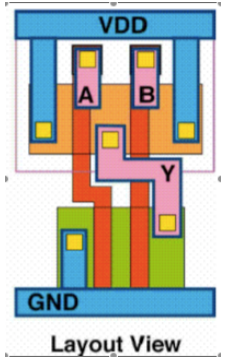
*Wh*at are the important metrics of comparison?

Full-Custom

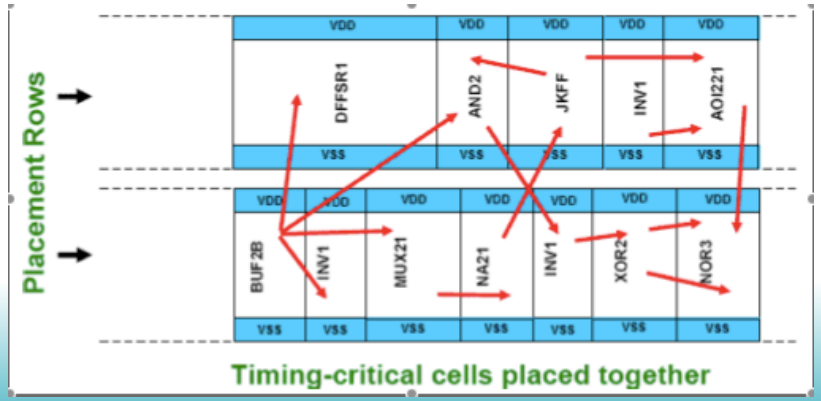
- Circuit styles and transistors are custom sized and drawn to optimize die, size, power, performance.
- High NRE (non-recurring engineering) costs
  - Time-consuming and error prone layout
- Hand-optimizing the layout can result in small die for low per unit costs, extreme-low-power, or extreme-high-performance.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

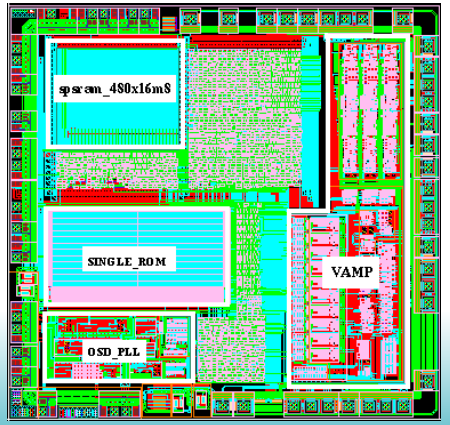


Standard-Cell* ASIC Design



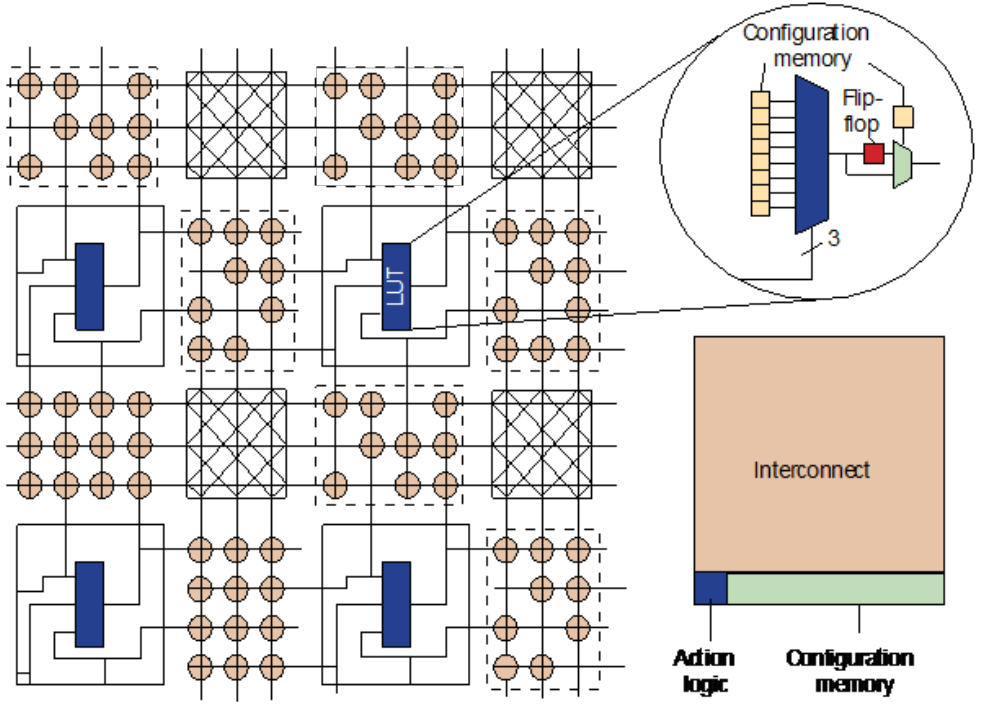
- Based around a set of pre-designed (and verified) cells
  - Ex: NANDs, NORs, Flip-Flops, counters slices, buffers, …
- Each cell comes complete with:
  - layout (perhaps for different technology nodes and processes),
  - Simulation, delay, & power models.
- Chip layout is automatic, reducing NREs (usually 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.)





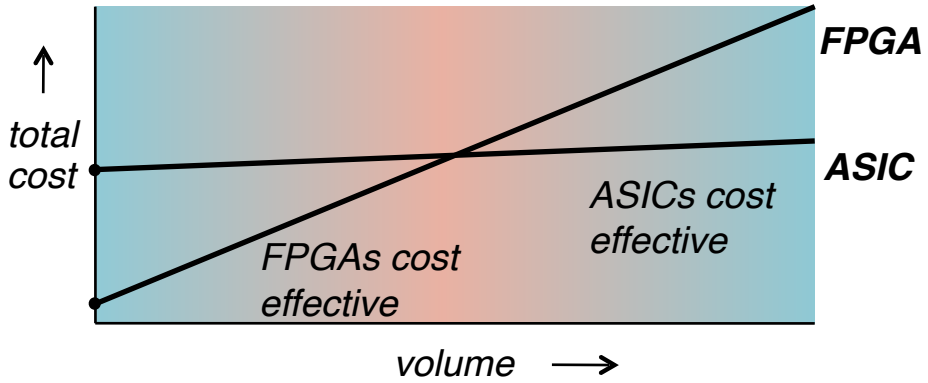
Field Programmable Gate Arrays (FPGA)

- *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, ...

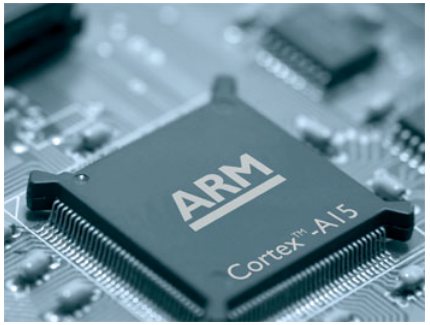
FPGA versus ASIC



- **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 a 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 ASICs, FPGA, other processors?

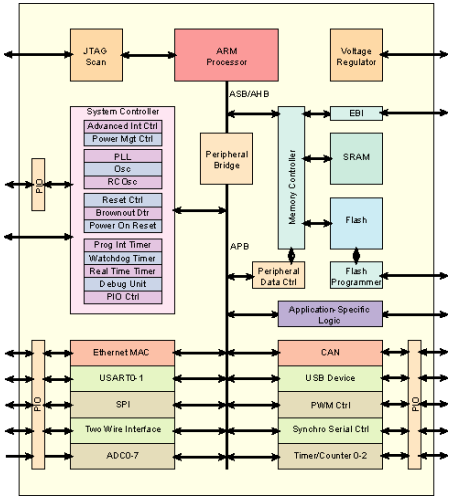


§	Assembler
	ADD{cond}{S} Rd, Rn, <Operand2>
	ADC{cond}{S} Rd, Rn, <Operand2>
5E	QADD{cond} Rd, Rm, Rn
5E	QDADD{cond} Rd, Rm, Rn
	SUB{cond}{S} Rd, Rn, <Operand2>
	SBC{cond}{S} Rd, Rn, <Operand2>
	RSB{cond}{S} Rd, Rn, <Operand2>
	RSC{cond}{S} Rd, Rn, <Operand2>
5E	QSUB{cond} Rd, Rm, Rn
5E	QDSUB{cond} 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)

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

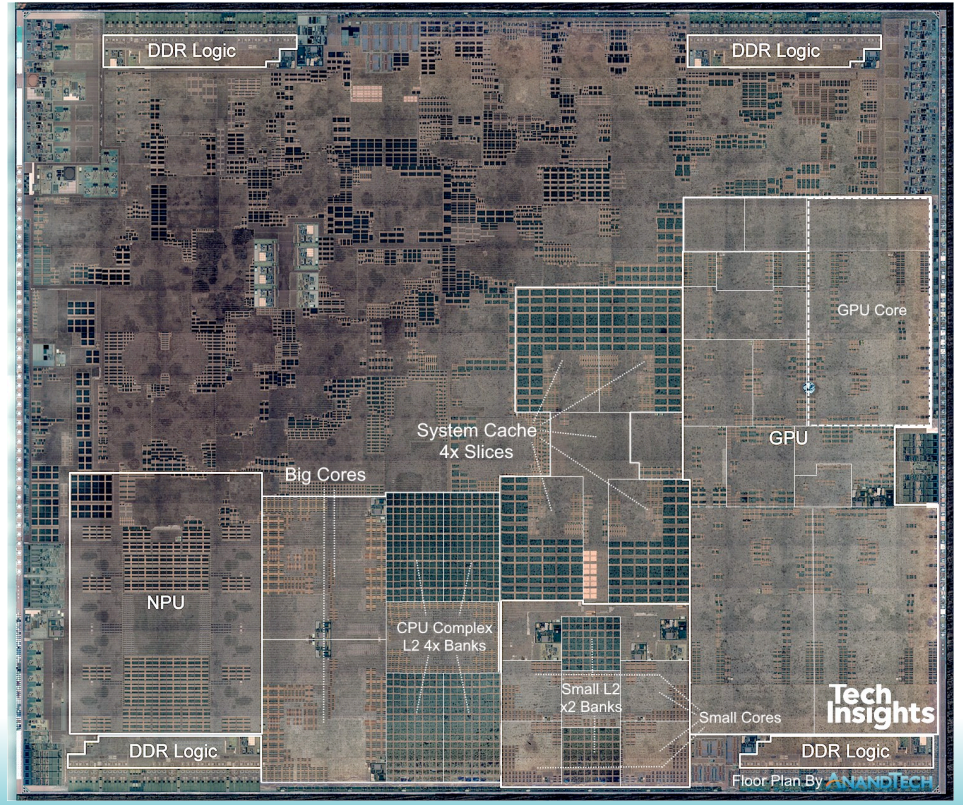
- *Brings together: standard cell blocks, custom analog blocks, processor cores, memory blocks, embedded FPGAs, …*
- *Standardized on-chip buses (or hierarchical interconnect) permit "easy" integration of many blocks.*
  - *Ex: AXI, AMBA, Sonics, …*
- *"IP Block" business model: Hard- or soft-cores available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



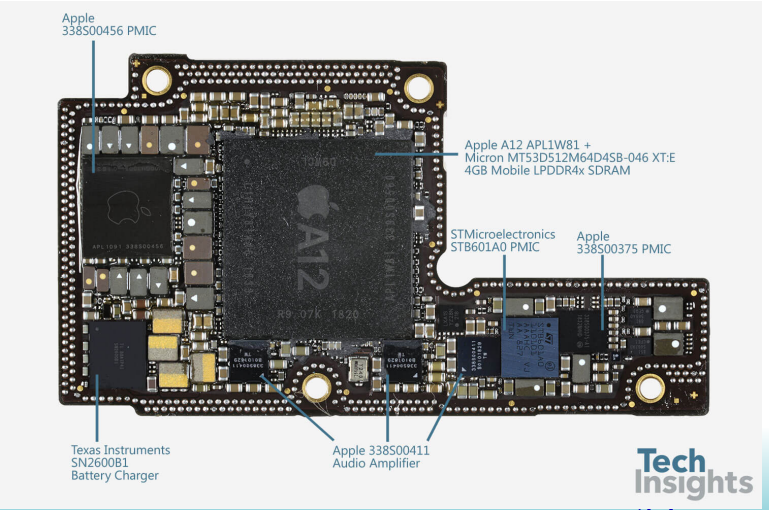
Qualcomm Snapdragon

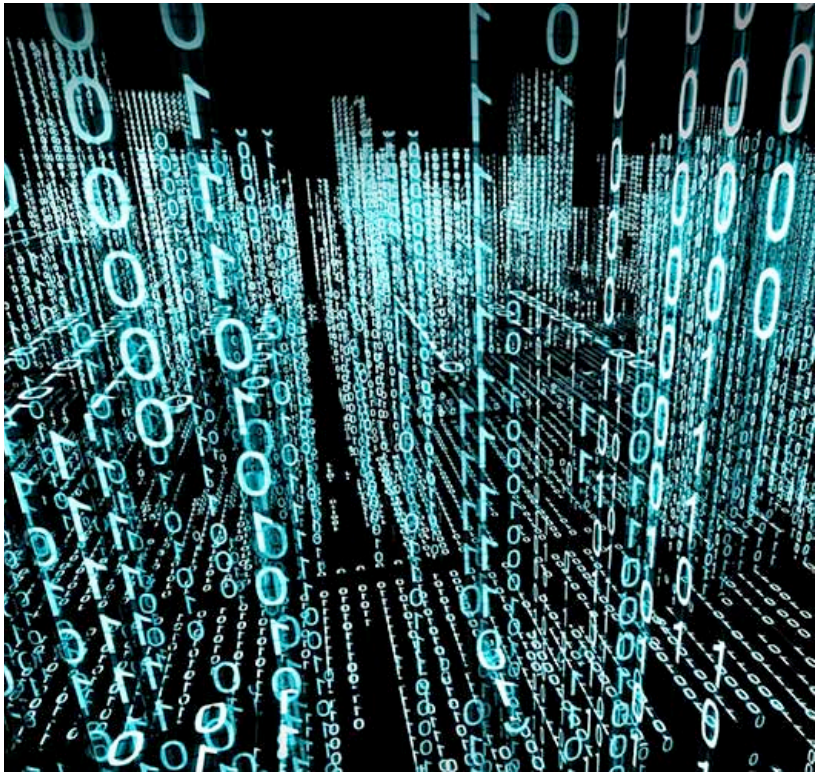
Modern(-ish) System-On-A-Chip (SOC)

- Apple A12 Bionic
  - *7nm CMOS, Up to 2.49GHz*



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





ASICs

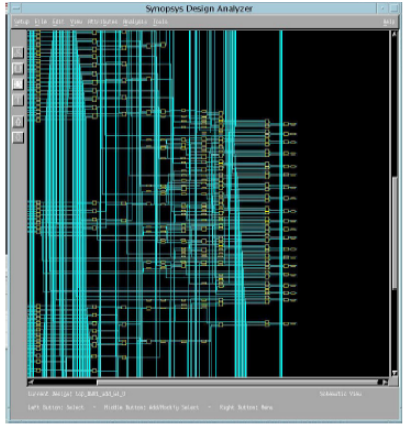
Verilog to ASIC layout flow

- "push-button" approach

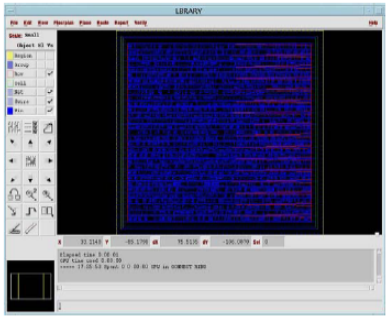
```
module adder64 (a, b, sum);
  input  [63:0] a, b;
  output [63:0] sum;

  assign sum = a + b;
endmodule
```

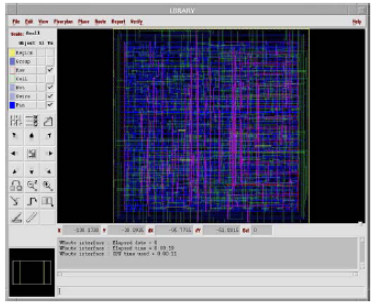
After Synthesis



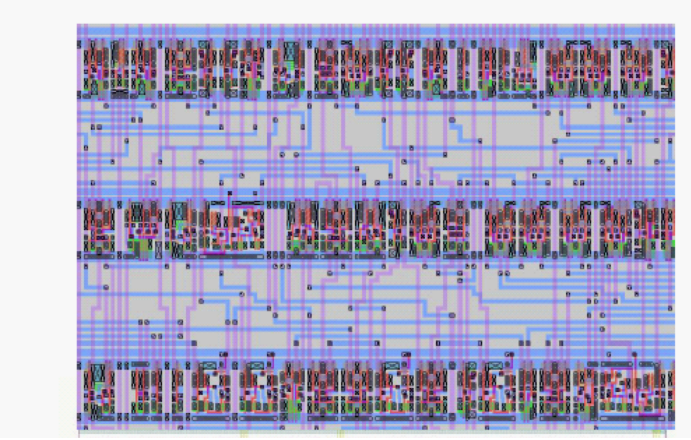
After Placement



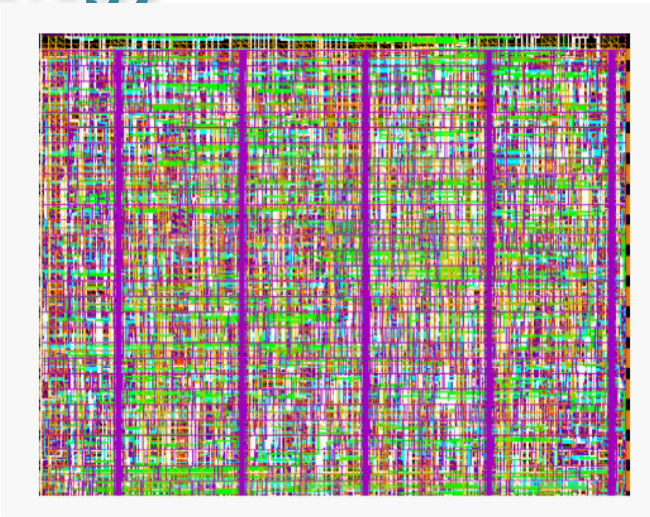
After Routing



Standard cell layout methodology



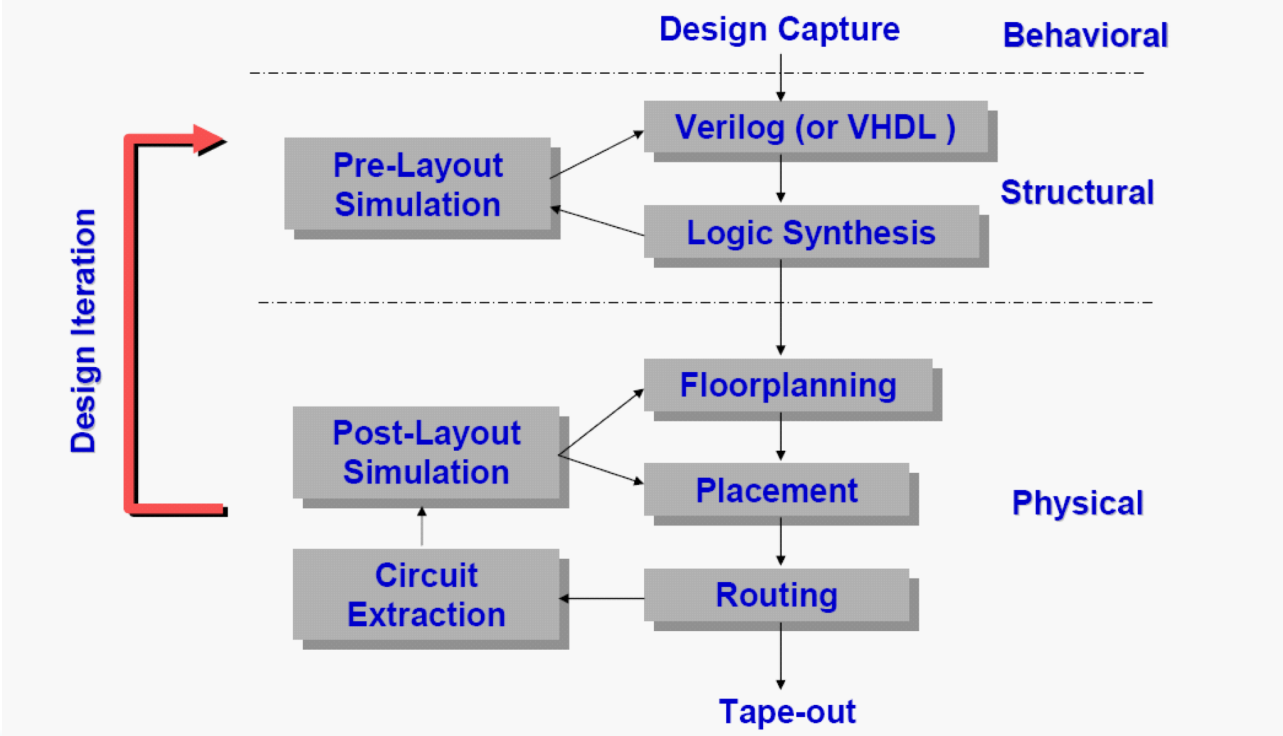
1um, 2-metal process



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

- With limited # metal layers, dedicated routing channels were needed
- Now, many layers and wires routed over cells. Currently area often dominated by wires

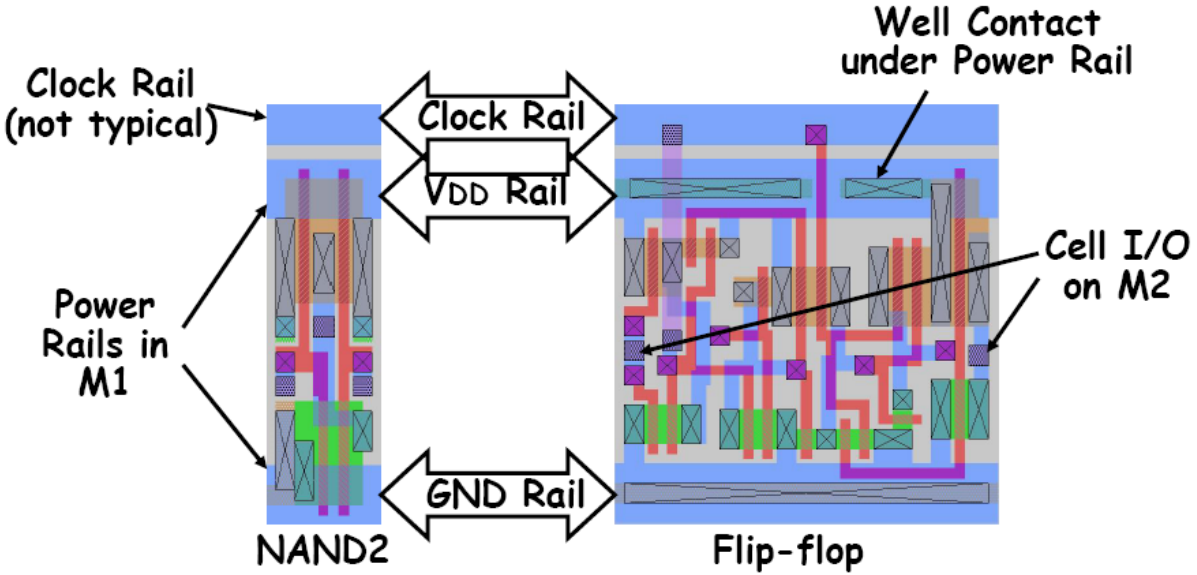
The ASIC flow



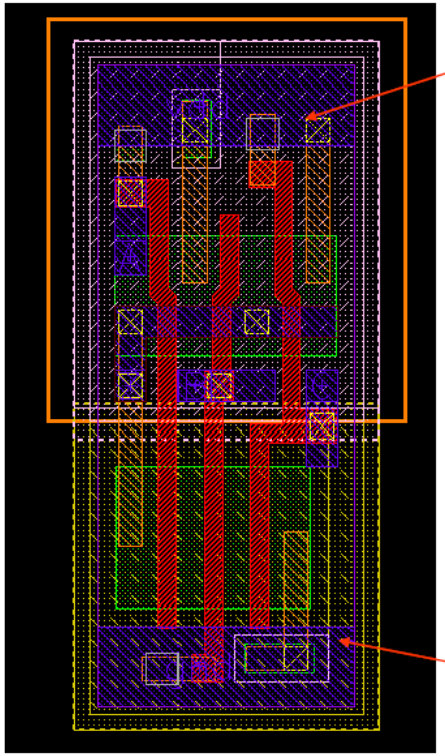
Standard cell design

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

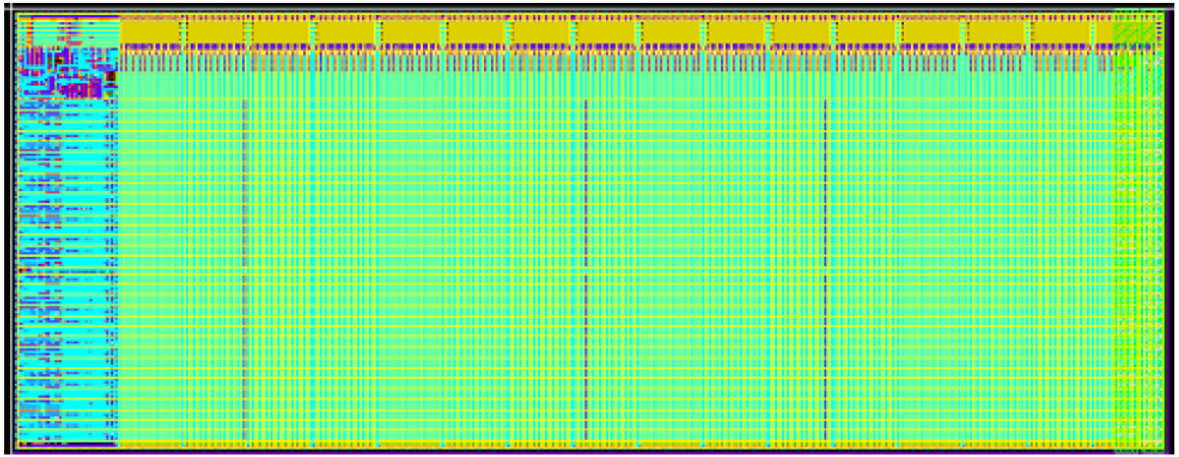
3-input NAND cell
(from ST Microelectronics):
C = Load capacitance
T = input rise/fall time

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 / cells

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

End of Lecture 2