inst.eecs.berkeley.edu/~eecs151

EECS151 : Introduction to Digital Design and ICs

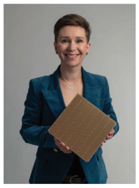
Lecture 2 – Design Process

Bora Nikolić

At HotChips'19 Cerebras announced the largest chip in the world at 8.5 in x 8.5in with 1.2 trillion transistors, and 15kW of power, aimed for training of deep-learning neural networks

At HotChips'21 they showed the next version in 7nm CMOS, with >2x transistor count

46,225 mm² silicon
2.6 Trillion transistors
850,000 AI optimized cores
40 Gigabytes on-chip memory
20 Petabyte/s memory bandwidth
220 Petabit/s fabric bandwidth
7nm Process technology at TSMC

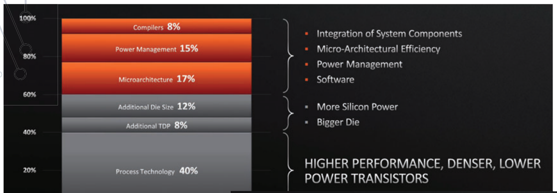


Sean Lie, HotChips'21

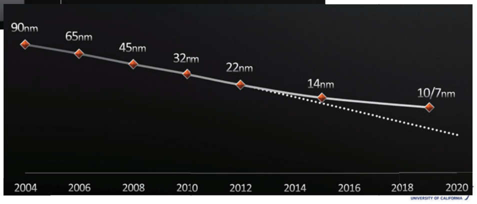
Review

- Moore's law is slowing down
  - There are continued improvements in technology, but at a slower pace
- Dennard's scaling has ended a decade ago
  - All designs are now power limited
- Specialization and customization provides added performance
  - Under power constraints and stagnant technology
- Design costs are high
  - Methodology and better reuse to rescue!
  - Abstraction, modularity, regularity are the keys
    - And creativity!

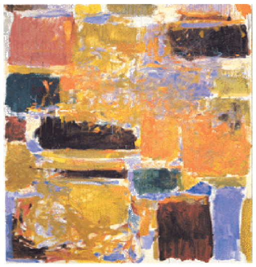
Putting it in Perspective



Performance gains over the past decade



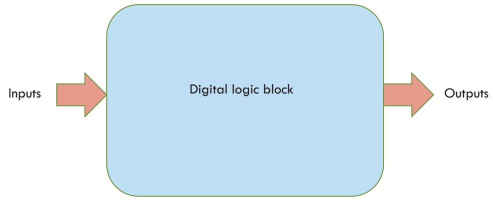
Lisa Su, HotChips'19 keynote



Digital Logic

Implementing Digital Systems

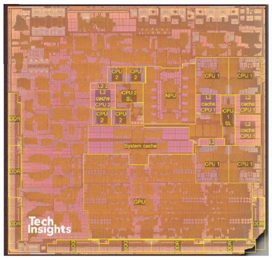
- Digital systems implement a set of Boolean equations



- How do we implement a digital system?

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

- 4x 'Firestorm' Large CPUs
- 4x 'Icestorm' Small CPUs
- GPU
- Neural processing unit (NPU)
- Lots of memory
- DDR memory interfaces



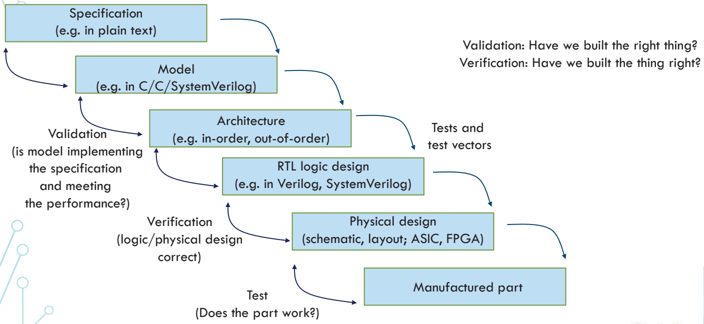
TechInsights

- 5nm CMOS
- Up to 2.49GHz

By Henriok
https://commons.wikimedia.org/w/index.php?curid=96026688

Design Process

- Design through layers of abstractions



Specification (e.g. in plain text)

Model (e.g. in C/C/SystemVerilog)

Architecture (e.g. in-order, out-of-order)

RTL logic design (e.g. in Verilog, SystemVerilog)

Physical design (schematic, layout; ASIC, FPGA)

Manufactured part

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

Validation (is model implementing the specification and meeting the performance?)

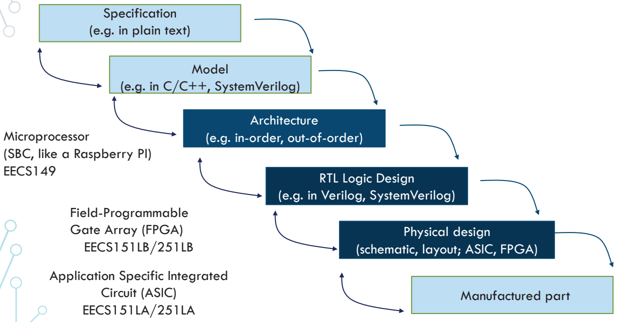
Tests and test vectors

Verification (logic/physical design correct)

Test (Does the part work?)

Design Abstractions in EECS151/251A

- Design through layers of abstractions



Specification (e.g. in plain text)

Model (e.g. in C/C++, SystemVerilog)

Architecture (e.g. in-order, out-of-order)

RTL Logic Design (e.g. in Verilog, SystemVerilog)

Physical design (schematic, layout; ASIC, FPGA)

Manufactured part

Microprocessor (SBC, like a Raspberry PI) EECS149

Field-Programmable Gate Array (FPGA) EECS151LB/251LB

Application Specific Integrated Circuit (ASIC) EECS151LA/251LA

Example: RISC-V Design Process

- Design through layers of abstractions

Specification
(e.g. in plain text)

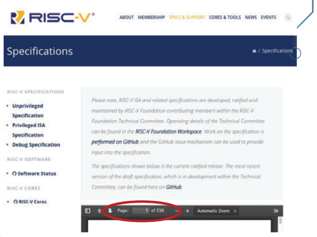
Model
(e.g. in C/C++/SystemVerilog)

Architecture
(e.g. in-order, out-of-order)

RTL Logic Design
(e.g. in Verilog/SystemVerilog)

Physical design
(schematic, layout; ASIC, FPGA)

Manufactured part



Specifications

RISC-V SPECIFICATIONS
- Unprivileged Specification
- Privileged ISA Specification
- Debug Specification

RISC-V SOFTWARE
- Software Status

RISC-V CORES
- RISC-V Cores

Please note, RISC-V ISA and related specifications are developed, ratified and maintained by RISC-V Foundation contributing members within the RISC-V Foundation Technical Committee. Operating details of the Technical Committee can be found in the ***RISC-V Foundation Workspace****. Work on the specification is* ***performed on GitHub*** *and the GitHub issue mechanism can be used to provide input into the specification.*

The specifications shown below is the current ratified release. The most recent version of the draft specification, which is in development within the Technical Committee, can be found here on ***GitHub****.*

https://riscv.org/specifications/

Example: RISC-V Design Process

- Design through layers of abstractions

Specification
(e.g. in plain text)

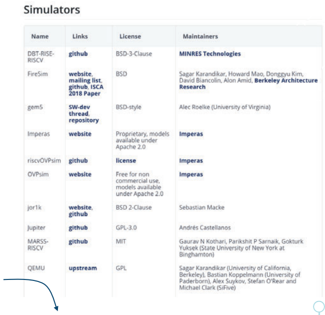
Model
(e.g. in C/C++/SystemVerilog)

Architecture
(e.g. in-order, out-of-order)

RTL Logic Design
(e.g. in Verilog/SystemVerilog)

Physical design
(schematic, layout; ASIC, FPGA)

Manufactured part



Simulators

Name	Links	License	Maintainers
DBT-RISE-RISCV	github	BSD-3-Clause	MINRES Technologies
FireSim	website, mailing list, github, ISCA 2018 Paper	BSD	Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Berkeley Architecture Research
gem5	SW-dev thread, repository	BSD-style	Alec Roelke (University of Virginia)
Imperas	website	Proprietary, models available under Apache 2.0	Imperas
riscvOVPsim	github	license	Imperas
OVPsim	website	Free for non commercial use, models available under Apache 2.0	Imperas
jor1k	website, github	BSD 2-Clause	Sebastian Macke
Jupiter	github	GPL-3.0	Andrés Castellanos
MARSS-RISCV	github	MIT	Gaurav N Kothari, Parikshit P Sarnaik, Gokturk Yuksek (State University of New York at Binghamton)
QEMU	upstream	GPL	Sagar Karandikar (University of California, Berkeley), Bastian Koppelmann (University of Paderborn), Alex Suykov, Stefan O'Rear and Michael Clark (SiFive)

https://riscv.org/software-status/#simulators

Example: RISC-V Design Process

- Design through layers of abstractions

Specification
(e.g. in plain text)

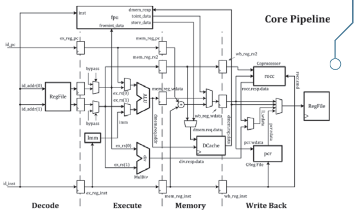
Model
(e.g. in C/C++/SystemVerilog)

Architecture
(e.g. in-order, out-of-order)

RTL Logic Design
(e.g. in Verilog/SystemVerilog)

Physical design
(schematic, layout; ASIC, FPGA)

Manufactured part



https://www.lowrisc.org/docs/tagged-memory-v0.1/rocket-core/
…and CS152

Example: RISC-V Design Process

- Design through layers of abstractions

Specification
(e.g. in plain text)

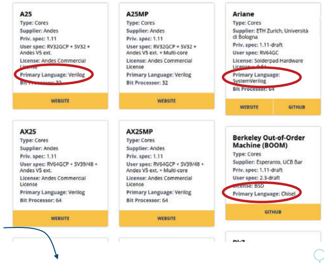
Model
(e.g. in C/C++/SystemVerilog)

Architecture
(e.g. in-order, out-of-order)

RTL Logic Design
(e.g. in Verilog/SystemVerilog)

Physical design
(schematic, layout; ASIC, FPGA)

Manufactured part



A25
Type: Cores
Supplier: Andes
Priv. spec: 1.11
User spec: RV32GCP + SV32 + Andes V5 ext.
License: Andes Commercial License
Primary Language: Verilog
Bit Processor: 32
WEBSITE

A25MP
Type: Cores
Supplier: Andes
Priv. spec: 1.11
User spec: RV32GCP + SV32 + Andes V5 ext. + Multi-core
License: Andes Commercial License
Primary Language: Verilog
Bit Processor: 32
WEBSITE

Ariane
Type: Cores
Supplier: ETH Zurich, Università di Bologna
Priv. spec: 1.11-draft
User spec: RV64GC
License: Solderpad Hardware License v. 0.51
Primary Language: SystemVerilog
Bit Processor: 64
WEBSITE GITHUB

AX25
Type: Cores
Supplier: Andes
Priv. spec: 1.11
User spec: RV64GCP + SV39/48 + Andes V5 ext.
License: Andes Commercial License
Primary Language: Verilog
Bit Processor: 64
WEBSITE

AX25MP
Type: Cores
Supplier: Andes
Priv. spec: 1.11
User spec: RV64GCP + SV39/48 + Andes V5 ext. + Multi-core
License: Andes Commercial License
Primary Language: Verilog
Bit Processor: 64
WEBSITE

Berkeley Out-of-Order Machine (BOOM)
Type: Cores
Supplier: Esperanto, UCB Bar
Priv. spec: 1.11-draft
User spec: 2.3-draft
License: BSD
Primary Language: Chisel
GITHUB

https://riscv.org/risc-v-cores/

Example: RISC-V Design Process

- Design through layers of abstractions

Specification
(e.g. in plain text)

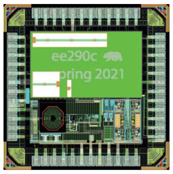
Model
(e.g. in C/C++/SystemVerilog)

Architecture
(e.g. in-order, out-of-order)

RTL Logic Design
(e.g. in Verilog/SystemVerilog)

Physical design
(schematic, layout; ASIC, FPGA)

Manufactured part



Example: RISC-V Design Process

- Design through layers of abstractions

Specification
(e.g. in plain text)

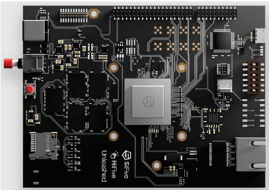
Model
(e.g. in C/C++/SystemVerilog)

Architecture
(e.g. in-order, out-of-order)

RTL Logic Design
(e.g. in Verilog/SystemVerilog)

Physical design
(schematic, layout; ASIC, FPGA)

Manufactured part



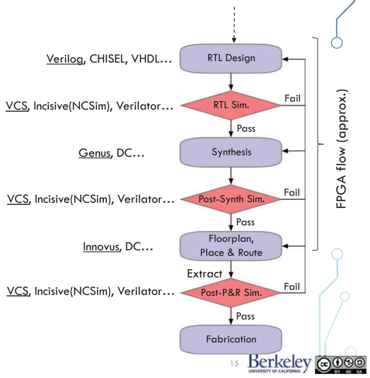
https://www.sifive.com/boards/hifive-unleashed

RTL → Physical Design

RTL Logic Design
(e.g. in Verilog/SystemVerilog)

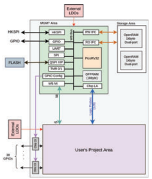
Physical design
(schematic, layout; ASIC, FPGA)

- Labs focus on a process of translating RTL to physical ASIC or FPGA by using industry-standard tools.
- Explores the entire design stack.

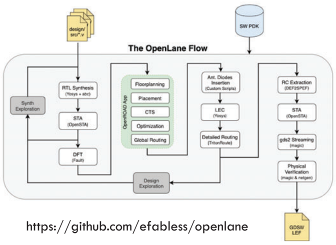


Open-Source Flows

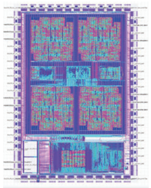
- Skywater 130nm is an open-source design kit
- OpenROAD (UCSD) and OpenLane (eFabless) are open-source design flows
  - Work with Sky130
  - A version of ASIC labs can target Sky130nm



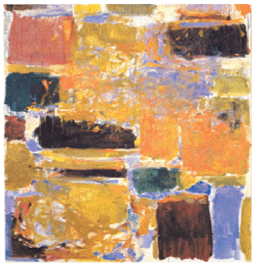
https://github.com/efabless/caravel

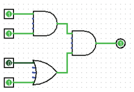


https://github.com/efabless/openlane



https://efabless.com/projects/35

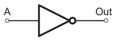
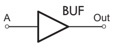
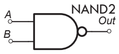
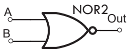




Boolean Logic in A Nutshell

Boolean Logic and Logic Gates (From CS61C/EE16B)

- Logic gates

Name	Boolean equation	Symbol
NOT or Inverter	Out = $\overline{A}$	
Buffer	Out = A	
NAND	Out = $\overline{A \cdot B}$	
NOR	Out = $\overline{A + B}$	

Truth table

A	Out
0	1
1	0

A	Out
0	0
1	1

Single input

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

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

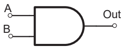
- In CMOS, basic logic gates are inverting

More Logic Gates

Name | Boolean equation | Symbol | Truth table

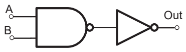
AND

Out =A · B



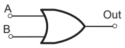
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

In CMOS



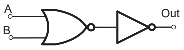
OR

Out =A + B



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

In CMOS

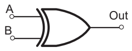


More Logic Gates

Name | Boolean equation | Symbol | Truth table

Exclusive OR
XOR

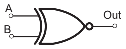
Out =A ⊕ B



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

Exclusive NOR
XNOR

Out =$\overline{A \oplus B}$



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

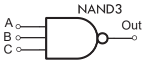
- XOR and XNOR are both inverting and non-inverting

Multi-Input Gates

3-Input NAND

NAND3 Boolean equation

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

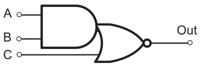


A	B	C	Out
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

And-Or-Invert

AOI21 Boolean equation

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

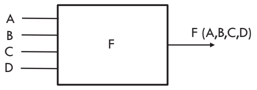


A	B	C	Out
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

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

Combinational Logic (CL) Blocks

Example four-input function:



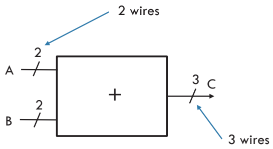
- Output a function only of the current inputs (no history).
- Truth-table representation of function. Output is explicitly specified for each input combination.
- In general, CL blocks have more than one output signal, in which case, the truth-table will have multiple output columns.

Truth Table

A	B	C	D	Out
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)
0	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)

Example CL Block

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



A1	A0	B1	B0	C2 C1 C0
0	0	0	0	0 0 0
0	0	0	1	0 0 1
0	0	1	0	0 1 0
0	0	1	1	0 1 1
0	1	0	0	0 0 1
0	1	0	1	0 1 0
0	1	1	0	0 1 1
0	1	1	1	1 0 0
1	0	0	0	0 1 0
1	0	0	1	0 1 1
1	0	1	0	1 0 0
1	0	1	1	1 0 1
1	1	0	0	0 1 1
1	1	0	1	1 0 0
1	1	1	0	1 0 1
1	1	1	1	1 1 0

- 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 network of simple logic gates.

Quiz

Total number of possible truth tables with 4 inputs is:

a) 4

b) 16

c) 256

d) 16,384

e) 65,536

f) None of the above

www.yellkey.com/foot

Peer Instruction

Total number of possible truth tables with 4 inputs is:

a) 4

b) 16

c) 256

d) 16,384

e) 65,536

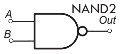
f) None of the above

www.yellkey.com/leg

Logic Circuit

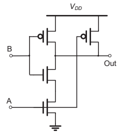
- A logic gate can be implemented in different ways

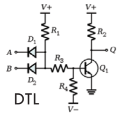
NAND Out = $\overline{A \cdot B}$



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

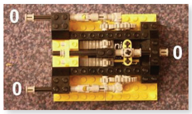
CMOS





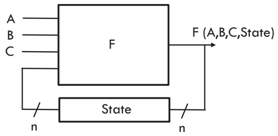
DTL

Sizing of transistors (W/L) in CMOS changes properties (delay, power, size) of a logic gate



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

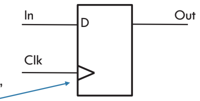
Sequential Logic Blocks



- Output is a function of both the current inputs and the state.
- State represents the memory.
- State is a function of previous inputs.
- In synchronous digital systems, state is updated on each clock tick.

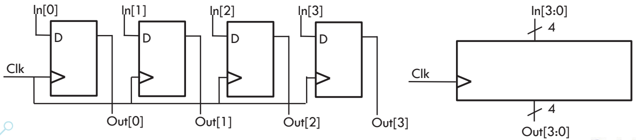
Flip-Flop as A Sequential Circuit

- Synchronous state element transfers its input to the output on a rising (or, rarely, falling) clock edge
- Flip-flop
  - Rising edge



Signifies 'edge triggered'

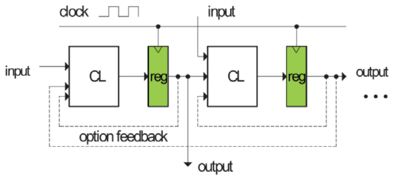
- 4-bit register



Register Transfer Level Abstraction (RTL)

Any synchronous digital circuit can be represented with:

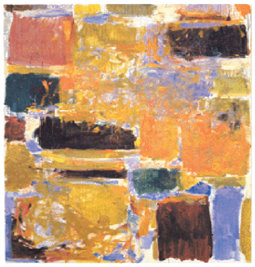
- Combinational Logic (CL) blocks, plus
- State elements (registers or memories)
- Clock orchestrates *sequencing* of CL operations



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

Administrivia

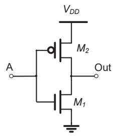
- Labs and discussions start this week
- Lab 1 posted, please start it before coming to the lab session
- Lab 2 is more involved
  - Be prepared
  - Verilog primer
- Homework 1 posted this week, due next Friday
  - Start early

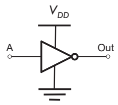


Design Metrics: Robustness

What Makes Circuits Digital?

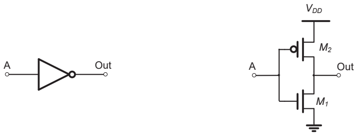
- Chips are noisy
- Supply noise will appear at the output of the logic gate





- The following logic gate should still interpret its inputs as 0s and 1s
- This necessary property is called "Restoration" or "Regeneration"
- A lot of money was spent in the past to unsuccessfully make logic out of non-regererative gates
  - Some of emerging CMOS replacements don't have gain…

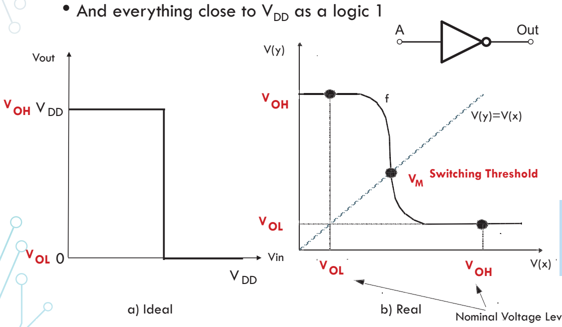
Beneath the Digital Abstraction



- Logic levels:
  - Mapping a continuous voltage onto a discrete binary logic variable
  - Low (0): $[0, V_L]$
  - High (1): $[V_H, V_{DD}]$
  - V_L V_H: nominal voltage levels

Voltage Transfer Characteristic

- A gate should interpret everything that is close to 0V as a logic 0
  - And everything close to V_{DD} as a logic 1

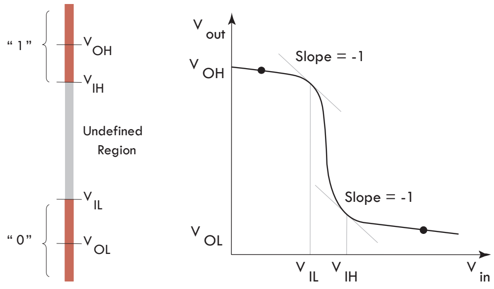


$V_{OH} = f(_{VOL})$
$V_{OL} = f(V_{OH})$
$V_M = f(V_M)$

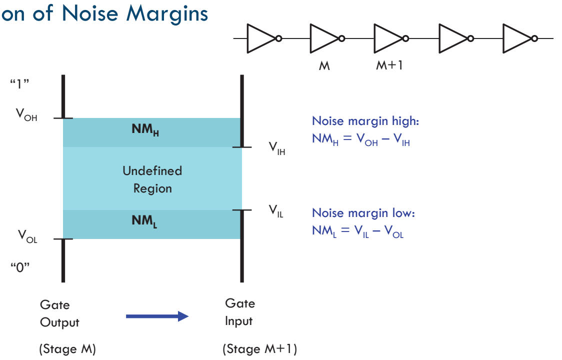
In CMOS:

$V_{OH} = V_{DD}$
$V_{OL} = 0$
$V_M \sim V_{DD}/2$

Mapping Between Analog Voltages and Digital Signals



Definition of Noise Margins



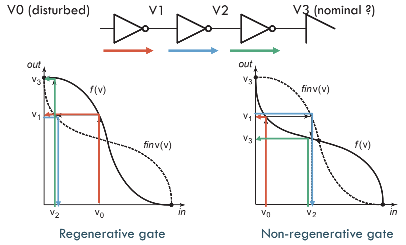
Noise margin high:
$NM_H = V_{OH} - V_{IH}$

Noise margin low:
$NM_L = V_{IL} - V_{OL}$

The amount of **noise** that could be added to **a worst-case output** so that the signal can still be interpreted correctly as **a valid input** to the next gate.

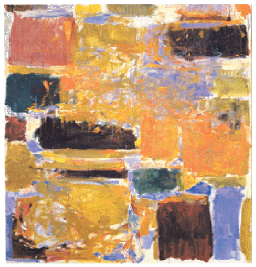
Regenerative Property

- Ensures that a **disturbed** signal gradually **regenerates** one of the **nominal voltage levels** after passing through a few logical stages.
  - Look for a sharp transition in voltage transfer characteristics.



Regenerative gate

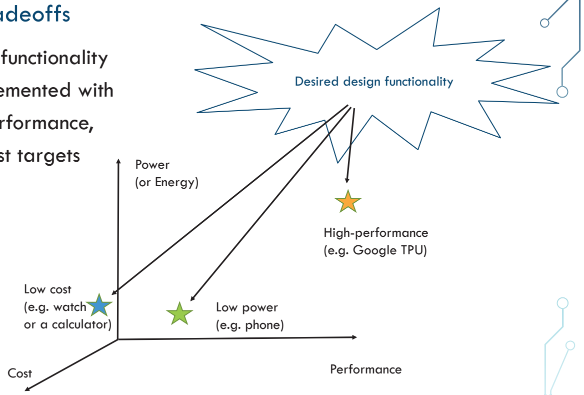
Non-regenerative gate



Design Metrics: Performance

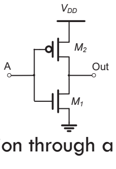
Design Tradeoffs

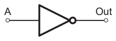
- The desired functionality can be implemented with different performance, power or cost targets



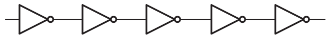
Digital Logic Delay

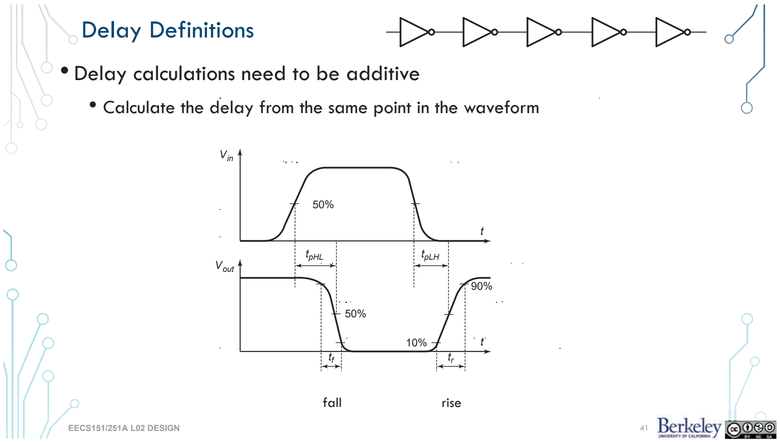
- Changes at the inputs do not instantaneously appear at the outputs
  - There are finite resistances and capacitances in each gate…





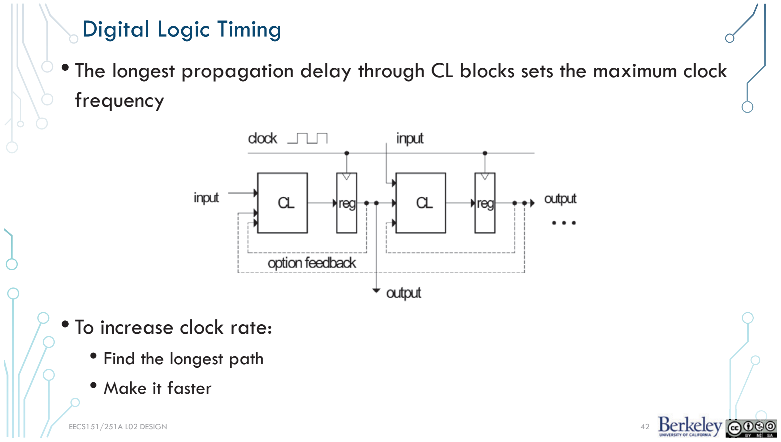
- Propagation through a chain of gates





Delay Definitions

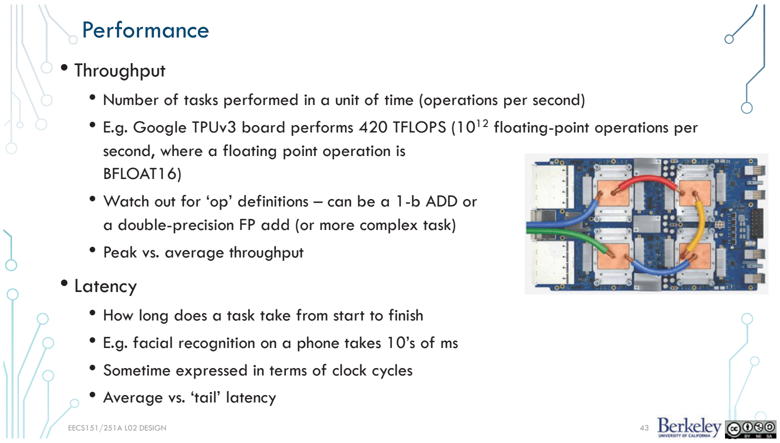
- Delay calculations need to be additive
  - Calculate the delay from the same point in the waveform



Digital Logic Timing

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

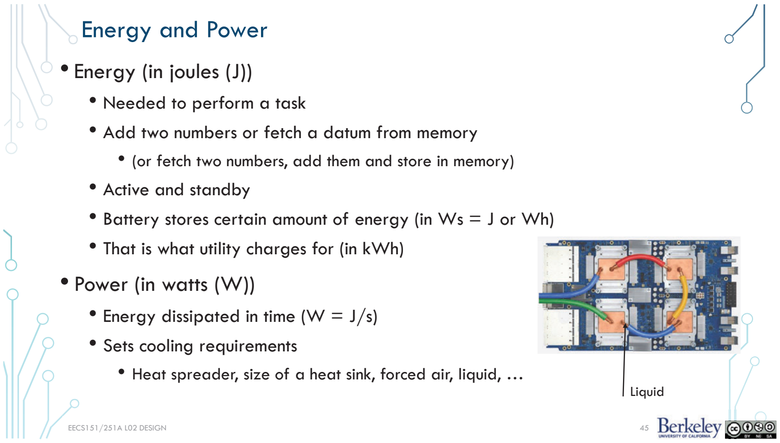
- To increase clock rate:
  - Find the longest path
  - Make it faster



Performance

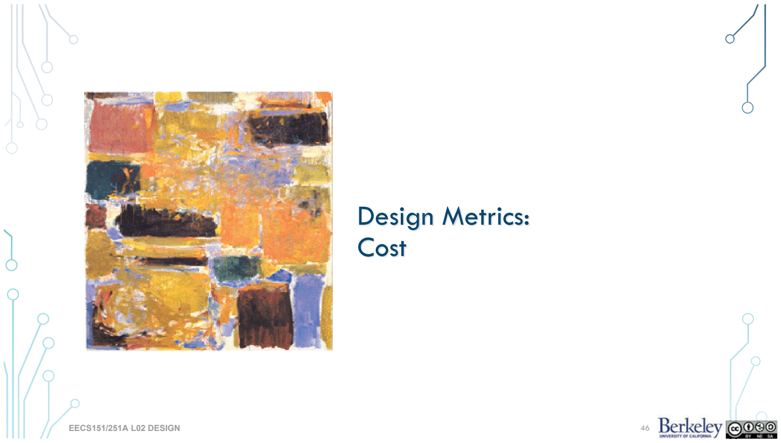
- Throughput
  - Number of tasks performed in a unit of time (operations per second)
  - E.g. Google TPUv3 board performs 420 TFLOPS (10^{12} floating-point operations per second, where a floating point operation is BFLOAT16)
  - Watch out for 'op' definitions – can be a 1-b ADD or a double-precision 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

Design Metrics: Energy and Power

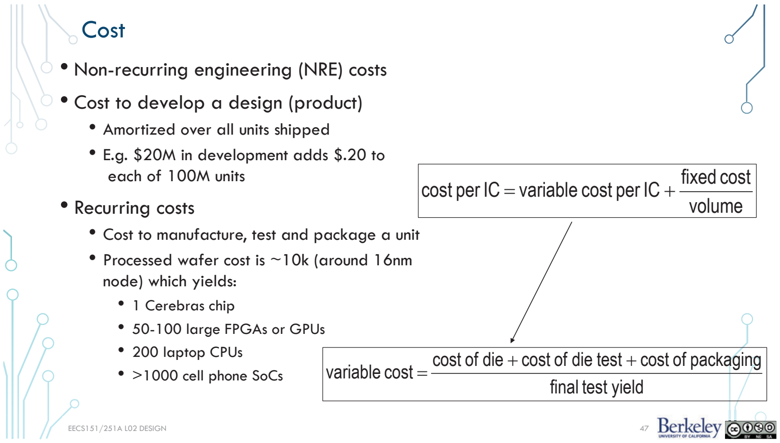


Energy and Power

- Energy (in joules (J))
  - Needed to perform a task
  - Add two numbers or fetch a datum from memory
    - (or fetch two numbers, add them and store in memory)
  - Active and standby
  - 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 in time (W = J/s)
  - Sets cooling requirements
    - Heat spreader, size of a heat sink, forced air, liquid, …



Design Metrics: Cost

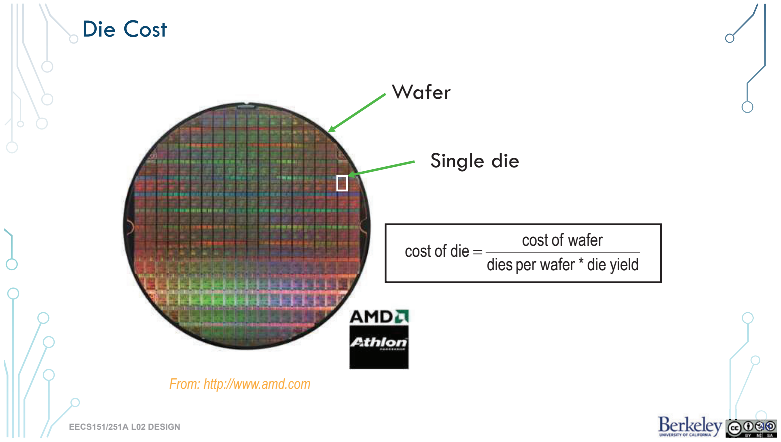


Cost

- Non-recurring engineering (NRE) costs
- Cost to develop a design (product)
  - 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:
    - 1 Cerebras chip
    - 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

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

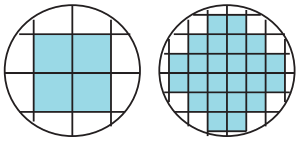
From: http://www.amd.com

Yield

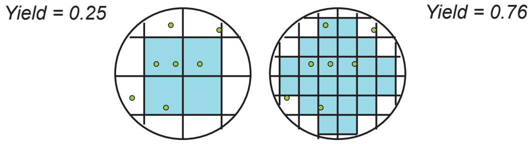
$$Y = \frac{\text{No. of good chips per wafer}}{\text{Total number of chips per wafer}} \times 100\%$$

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

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



Defects



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

α is approximately 3

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

Summary

- The design process involves traversing the abstraction layers of specification, modeling, architecture, RTL design and physical implementation
- Tests follow the design refinements
- Targets are processors, FPGAs or ASICs
- Automated design flows help manage the complexity
- Optimize for performance, energy and cost