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, ’22 and ‘23 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
Review

• Moore’s law is slowing down
  • There are continued improvements in technology, but at a slower pace

• Dennard’s scaling has ended two decades 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 actually implement a complex digital system?
Modern (Mostly) Digital Systems-On-A-Chip

- Multiple large CPUs
- Multiple small CPUs
- GPU
- Neural processing unit (NPU)
- Lots of memory
- DDR memory interfaces

https://www.semianalysis.com/p/apple-m2-die-shot-and-architecture

- TSMC N5 (5nm-class) CMOS
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?

Test (Does the part work?)
Tests and test vectors

Validation (is model implementing the specification and meeting the performance?)
Verification (logic/physical design correct)
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

---

EECS151 L02 DESIGN
Example: RISC-V Design Process

- Design through layers of abstractions

Specifications

- 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://riscv.org/specifications/
**Example: RISC-V Design Process**

- Design through layers of abstractions

### Simulators

<table>
<thead>
<tr>
<th>Name</th>
<th>Links</th>
<th>License</th>
<th>Maintainers</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBT-RISC-RISCV</td>
<td>github</td>
<td>BSD-3-Clause</td>
<td>MINRES Technologies</td>
</tr>
<tr>
<td>FireSim</td>
<td>website, mailing list, github ISCA 2018 Paper</td>
<td>BSD</td>
<td>Sagar Karandikar, Howard Mao, Dongyun Kim, David Biancolini, Ali Amidi, Berkeley Architecture Research</td>
</tr>
<tr>
<td>gem5</td>
<td>SW-dev thread, repository</td>
<td>BSD-style</td>
<td>Alec Roelke (University of Virginia)</td>
</tr>
<tr>
<td>Imperas</td>
<td>website</td>
<td>Proprietary, models available under Apache 2.0</td>
<td>Imperas</td>
</tr>
<tr>
<td>riscOVPsim</td>
<td>github, license</td>
<td></td>
<td>Imperas</td>
</tr>
<tr>
<td>OVPsim</td>
<td>website</td>
<td>Free for non commercial use, models available under Apache 2.0</td>
<td>Imperas</td>
</tr>
<tr>
<td>jor1k</td>
<td>website, github</td>
<td>BSD-2-Clause</td>
<td>Sebastian Mackle</td>
</tr>
<tr>
<td>Jupiter</td>
<td>github</td>
<td>GPL-3.0</td>
<td>Andrés Castellanos</td>
</tr>
<tr>
<td>MARSS-RISCV</td>
<td>github</td>
<td>MIT</td>
<td>Gaurav N Kothari, Parikshit P Sarnaik, Gokturk Yuksek (State University of New York at Binghamton)</td>
</tr>
<tr>
<td>QEMU</td>
<td>upstream</td>
<td>GPL</td>
<td>Sagar Karandikar (University of California, Berkeley), Bastian Koppelmans (University of Paderborn), Alex Suykov, Stefan O'Rear and Michael Clark (SiFive)</td>
</tr>
</tbody>
</table>

### Resources

- 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://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

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**

**S’22 Class chip**
(HotChips’23)
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

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

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

**Physical design**
(schematic, layout; ASIC, FPGA)
Open-Source Flows

• SkyWater 130nm is an open-source design kit
• OpenROAD (UCSD) and OpenLane (eFabless) are open-source design flows
  • Work with Sky130
  • ASIC labs can target SkyWater 130nm with commercial tools
Design, Verification and Validation

• Design steps

  - 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

• Verification and validation steps

  - Model/system performance validation (Simulation)
  - RTL verification (Simulation/formal)
  - Timing/power verification (Static analysis, simulation)
  - Post-silicon validation (Lab)
  - Equivalence checking (Formal)
Boolean Logic in A Nutshell
Boolean Logic and Logic Gates (From CS61C/EE16B)

- **Logic gates**

<table>
<thead>
<tr>
<th>Name</th>
<th>Boolean equation</th>
<th>Symbol</th>
<th>Truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT or Inverter</td>
<td>$\text{Out} = \overline{A}$</td>
<td>![Symbol] OUT/INV</td>
<td>![Truth Table]</td>
</tr>
<tr>
<td>Buffer</td>
<td>$\text{Out} = A$</td>
<td>![Symbol] BUF</td>
<td>![Truth Table]</td>
</tr>
<tr>
<td>NAND</td>
<td>$\text{Out} = \overline{A \cdot B}$</td>
<td>![Symbol] NAND2</td>
<td>![Truth Table]</td>
</tr>
<tr>
<td>NOR</td>
<td>$\text{Out} = \overline{A + B}$</td>
<td>![Symbol] NOR2</td>
<td>![Truth Table]</td>
</tr>
</tbody>
</table>

- In CMOS, basic logic gates are inverting
More Logic Gates

<table>
<thead>
<tr>
<th>Name</th>
<th>Boolean equation</th>
<th>Symbol</th>
<th>Truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>( \text{Out} = A \cdot B )</td>
<td>![AND symbol]</td>
<td>![AND truth table]</td>
</tr>
<tr>
<td>OR</td>
<td>( \text{Out} = A + B )</td>
<td>![OR symbol]</td>
<td>![OR truth table]</td>
</tr>
</tbody>
</table>

In CMOS
## More Logic Gates

<table>
<thead>
<tr>
<th>Name</th>
<th>Boolean equation</th>
<th>Symbol</th>
<th>Truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive OR</td>
<td>$\text{Out} = A \oplus B$</td>
<td><img src="image1.png" alt="Symbol" /></td>
<td><img src="image2.png" alt="Truth Table" /></td>
</tr>
<tr>
<td>XOR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exclusive NOR</td>
<td>$\text{Out} = A \uparrow B$</td>
<td><img src="image3.png" alt="Symbol" /></td>
<td><img src="image4.png" alt="Truth Table" /></td>
</tr>
<tr>
<td>XNOR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* XOR and XNOR are both inverting and non-inverting*
Multi-Input Gates

3-Input NAND
NAND3  Boolean equation
\[ \text{Out} = \overline{A \cdot B \cdot C} \]

And-Or-Invert
AOI21  Boolean equation
\[ \text{Out} = \overline{A \cdot B + C} \]

• 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.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(0,0,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(0,0,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(0,0,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(0,0,1,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(0,1,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(0,1,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(0,1,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(0,1,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(1,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(1,0,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(1,0,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(1,0,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(1,1,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(1,1,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(1,1,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(1,1,1,1)</td>
</tr>
</tbody>
</table>
Example CL Block

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

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

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/notice
Logic Circuit

- A logic gate can be implemented in different ways

**NAND**
\[ \text{Out} = \overline{A \cdot B} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**CMOS**

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

**DTL**

Mechanical LEGO logic gates. A clockwise rotation represents a binary “one” while a counterclockwise 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
• Apple NSI event: Wednesday, August 30, 11:45am, Banatao Auditorium
  • Burritos + bobba
Design Metrics: Robustness
What Makes Circuits Digital?

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

![Logic gate diagram]

• 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

In CMOS:
- $V_{OH} = V_{DD}$
- $V_{OL} = 0$
- $V_{M} \approx V_{DD}/2$

Nominal Voltage Levels
- $V_{OH} = f(V_{OL})$
- $V_{OL} = f(V_{OH})$
- $V_{M} = f(V_{M})$

Switching Threshold

$V(y) = V(x)$
Mapping Between Analog Voltages and Digital Signals

Slope = -1

"0"

"1"

Undefined Region

V_{IL}

V_{OL}

V_{IH}

V_{OH}

V_{out}

V_{in}

Slope = -1

Slope = -1
Definition of Noise Margins

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.

![Diagram of regenerative and non-regenerative gates]

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 TPUv4 chip performs 275 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)
  • Energy footprint matters for the environment!

• Power (in watts (W))
  • Energy dissipated in time (W = J/s)
  • Sets cooling requirements
    • Heat spreader, size of a heat sink, forced air, liquid, …

TPU v4. ~200W
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 wafer-scale chip
    • 50-100 large FPGAs or GPUs
    • 500 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

\[
\text{cost of die} = \frac{\text{cost of wafer}}{\text{dies per wafer} \times \text{die yield}}
\]

From: http://www.amd.com
Yield

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

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

\[ \text{Dies per wafer} = \frac{\pi \times \left( \text{wafer diameter}/2 \right)^2}{\text{die area}} - \frac{\pi \times \text{wafer diameter}}{\sqrt{2} \times \text{die area}} \]
Defects

\[ \text{Yield} = 0.25 \quad \text{Yield} = 0.76 \]

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

\( \alpha \) is approximately in the range 0.5-5

If \( \alpha = 3 \), 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