So what has changed in 30 years?
Moore's Law for CPUs and DRAMs

Secondary driver: Wafer size


Wafer size conversions offset trend of increasing wafer processing cost

Processing advances

45nm
IC Technology Stuff (1)

- Feature size:
  then: $\approx 4 \mu m$  now: $\approx 0.045 \mu m$

- Interconnect:
  then: 2 layers  now: 10 layers
  then: aluminum  now: copper

- Transistors:
  then: planar MOSFET  now: same

- Layout and GDRs:

- Circuits:
  then: clocked static CMOS  now: same (lots of crazy stuff in between)

  Most CMOS circuits and layouts designed in 1980 would work if fabricated on today's IC process.

IC Technology Stuff (2)

- Transistors:
  then: near perfect switch  now: leaky

- Power consumption:
  then: dynamic (switching) energy  now: approaching 50% static leakage (back to the future - nMOS has similar problem)

- New improved devices on the horizon: FinFETs

- Chip Input/Output
  then: parameter pads  now: often area pads

- Lithographic Mask Costs:
  then: few $\&k$  now: $\&M$ (full die, 65, 45, 22nm)
IC Technology Stuff (3)

- **Device reliability:**
  
  *then:* devices nearly never fail  *future* (<65nm): high soft and hard error rates

- **Process variations across die, die-to-die:**
  
  - Statistical variations in processing (wire widths/resistivity, transistor dimensions/strengths, doping inconsistencies) become apparent at smaller geometries.
  - Some circuits fast, others slow. Some high-power, some low.
  - Worst case design results in very bad overall performance.

- **Yield on leading edge processes dropping dramatically**
  
  - IBM quotes yields of 10 – 20% on Cell processor

Design Stuff

- **Chip functionality:**

  *then:* limited by area  *now:* often limited by energy dissipation

- **Design cost:**

  *now:* design costs in $50M range for full-die custom designs (high percentage in verification)

- **Implementation Alternatives:** more alternatives that trade up-front design costs for per unit costs.

- **FPGA compete aggressively with custom silicon**

  *then:* most custom designs implemented at silicon level
  
  *now:* many more custom designs implemented with FPGAs

- **Standard design abstraction:**

  *then:* transistors circuits  *now:* RTL in HDLs, standard “cores” and standard cells (higher productivity, somewhat less area/energy efficient) -
Implementation Alternatives

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

By “ASIC”, most people mean “Standard-cell” based implementation.

What are the important metrics of comparison?

The Important Distinction

- Instruction Binding Time
  - When do we decide what operation needs to be performed?

- 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
- Optimizing for small die can result in low per unit costs, extreme-low-power, or extreme-high-performance.
- Common 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, 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).
- Requires full set of masks - nothing prefabricated.
- Non-optimal use of area and power, leading to higher per die costs than full-custom.
- Commonly used with other design implementation strategies (large blocks for memory, I/O blocks, etc.)
Gate Array

- Store prefabricated wafers of “active” & gate layers & local interconnect, comprising, primarily, rows of transistors. Customize as needed with “back-end” metal processing (contact cuts, metal wires). Could use a different factory.

- Shifts large portion of design and mask NRE to vendor.
- Shorter design and processing times, reduced time to market.
- 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:
Field Programmable Gate Arrays

- Two-dimensional array of simple logic- and interconnection-blocks.
- Typical architecture: LUTs implement any function of n-inputs (n=3 in this case).
- Optional 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, ...

Traditional FPGA versus ASIC argument (circa 2000)

- ASIC: High NRE costs ($2M for 0.35μm chip). Relatively Low cost per die.
- FPGAs: Very low NRE costs. Relatively low silicon efficiency ⇒ high cost per part.
- Cross-over volume from cost effective FPGA design to ASIC in the 10K range.
Cross-over Point has Moved Right

- **ASIC**: Increasing NRE costs (>40M for 90nm chip\(^1\)) (verification, mask costs\(^2\), etc.)
  - Fewer silicon designs becomes inevitable.
- **FPGAs**: Move in to fill the need, furthermore, FPGAs better able to follow Moore's Law, relatively cheaper to test.
- Cross-over volume now >100K.

\(^1\) Vahid Manian, VP manufacturing and operations, Broadcom Corp.
\(^2\) Roger Minear, Agere Systems Inc, 30-35-layer mask set ≈$650,000 for 130nm and $1.4M for 90nm.

Post-fabrication Customization

- **Gate Array** like devices (structured ASICs) return to fill the gap. Post-fab customization with limited mask layers.
  - Lower NREs than ASICs, more silicon efficiency than FPGAs.
Hybrids Chip Implementations Abound

- Ex: standard practice in microprocessors that data-paths are full-custom and control (instruction decode, pipeline control) in standard-cells. (Less common recently)

Control ("random") logic difficult to "regularize". Relatively small percentage of die area/power. Permits late binding of design changes.

Extra NAND or NOR gates were often added to control section, and some wafers left without metallization, to permit late design fixes through metal mask revisions (gate-array idea).

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: 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, ...

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

SIP, SOP, MCM interesting alternatives.
**Early '80's Design Methodology and Flow**

- **Schematic + Full-Custom Layout**
  - SPICE for critical path,
  - switch-level simulation for overall functionality,
  - hand layout,
  - no power analysis,
  - layout verified with LVS and GDRC

**Modern ASIC Methodology and Flow**

- **RTL Synthesis Based**
  - HDL specifies design as combinational logic + state elements
  - 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

---

**Specification**

- Transistor Schematics
  - SPICE
  - swtich simulator
  - hand layout

- CIF file
- layout vs. schematic
- geometric design rule checker

---

**Specification**

- RTL (Verilog/VHDL) + cell instantiations
  - "formal" verification
  - logic synthesis
  - event simulator

- gate netlist (with area/perf/pwr estimates)
  - cell place & route
  - GDS

- GDRC, LVS, other checks
  - timing/power analysis
Design Representations

Engineering Challenge

Application

Gap usually too large to bridge in one step, but there are exceptions...

Physics
Magnetic Compass

Application

Physics

Design Abstraction Stack

Application

Unit-Transaction Level (UTL)

Register-Transfer Level (RTL)

Gates

Circuits

Devices (Transistors)

Physics

Conduction Band

Valence Band

Valence Band

Application
Properties of a Useful Abstraction

- Hides less important details
  - e.g., for RTL, don’t worry how combinational logic is decomposed into logic gates
- Allows control of more important details
  - e.g., RTL designer still controls how much logic is performed between any two registers
- If done right, provides portable efficiency
  - i.e., same RTL can be implemented as custom logic, standard cells, FPGA, or even vacuum tube logic, with reasonably good results

CS250 Design Abstractions (Processor-centric)

- Processor Inst Set Arch (ISA)
- Processor micro-architecture
- Register-Transfer Level (RTL)
- Gates
- Circuits
- Devices (Transistors)
- Physics

UCB EE130/230
UCB EE141/241

lw  $t0, 0($2)
lw  $t1, 4($2)
sw  $t1, 0($2)
sw  $t0, 4($2)
Course Prerequisites

- B+ in CS150 for UCB undergrads, or equivalent for incoming grad students

- This means you should have experience with RTL and Verilog/VHDL before
  - We won’t be covering Verilog coding details in lecture, but some coverage in section + handouts
Logic Synthesis

- Verilog and VHDL started out as simulation languages, but quickly people wrote programs to automatically convert Verilog code into gate level netlists.

- Synthesis converts Verilog (or other HDL) descriptions to implementation technology specific primitives:
  - For FPGAs: LUTs, flip-flops, and RAM blocks
  - For ASICs: standard cell gate and flip-flop libraries. Memory blocks built with special "memory generator" and then hand-instantiated.

Why Logic Synthesis?

1. Automatically manages many details of the design process:
   - Fewer bugs
   - Improved productivity
2. Abstracts the design data (HDL description) from any particular implementation technology.
   - Designs can be re-synthesized targeting different chip technologies. Ex: first implement in FPGA then later in ASIC.
3. In most cases, leads to a more optimal design than could be achieved by manual means (ex: logic optimization)

Why Not Logic Synthesis?
Main Logic Synthesis Steps

- Parsing and Syntax Check
  - Load in HDL file, run macro preprocessor for `define, `include, etc..
  - Compute parameter expressions, process generates, create instances, connect ports.
- Design Elaboration
  - Recognize and insert special blocks (arithmetic structures, ...)
- Inference and Library Substitution
  - Expand combinational logic to primitive Boolean representation.
- Logic Expansion
  - Apply Boolean algebra and heuristics to simplify and optimize under constraints.
- Logic Optimization
  - Map generic logic representation to cell instances from chosen cell library.
- Technology Mapping
  - Modern tools incorporate preliminary layout & timing constraints, and attempt timing driven synthesis.

Operators and Synthesis

- Logical operators map into primitive logic gates
- Arithmetic operators map into adders, subtractors, ...
  - Unsigned 2s complement
  - Watch out for *, %, and /
- Relational operators generate comparators
- Shifts by constant amount are just wire connections
  - No logic involved
- Variable shift amounts a whole different story --- shifter
- Conditional expression generates logic or MUX

\[ Y = \sim X \ll 2 \]
Verilog Procedural Assignments are Evil

The sequential semantics of the blocking assignment allows variables to be multiply assigned within a single always block. Unexpected behavior can result from mixing these assignments in a single block. Standard rules:

1. Use blocking assignments to model combinational logic within an always block ("=").
2. Use non-blocking assignments to implement sequential logic ("<=").
3. Do not mix blocking and non-blocking assignments in the same always block.
4. Do not make assignments to the same variable from more than one always block.

Verilog in CS250

- Primarily use low-level behavioral modeling with instantiation to 1) build hierarchy and, 2) map to resources not supported by synthesis or for efficiency (ex: register file)
- Primary Style Guidelines:
  - Favor continuous assign and avoid always blocks unless:
    - no other alternative: ex: state elements, case
    - they help readability and clarity of code: ex: large nested if-else-if
  - Use named ports.
  - Separate CL logic specification from state elements.
  - Follow our rules for procedural assignments.
- Verilog is a big language.
  - Read and understand the Verilog we give you.
  - When needed look at online IEEE Std 1364-2001 document.
  - Be careful of what you read on the web! Many bad examples out there.
Final thoughts on Verilog

Verilog may look like C, but it describes hardware! (Except in simulation test-benches - which actually behave like programs.)

Multiple physical elements with parallel activities and temporal relationships.

A large part of digital design is knowing how to write Verilog that gets you the desired circuit. First understand the circuit you want, then figure out how to code it in Verilog. If you do one of these activities without the other, you will struggle. These two activities will merge at some point for you.

Be suspicious of the synthesis tools! Check the output of the tools to make sure you get what you want.

Logic Synthesis

Following slides by Srini Devadas, MIT
Logic optimization flow

LOGIC EQUATIONS

TECHNOLOGY-INDEPENDENT OPTIMIZATION

Factoring
Commonality Extraction

TECH-DEPENDENT OPTIMIZATION
(MAPPING, TIMING)

LIBRARY

OPTIMIZED LOGIC NETWORK
Boolean Optimizations

Involves:
Finding common subexpressions.
Substituting one expression into another.
Factoring single functions.

\[
F = \begin{cases} 
  f_1 = AB + AC + AD + AE + ABCDE \\
  f_2 = AB + AC + AD + AF + ABCDF 
\end{cases}
\]

Find common expressions

\[
F = \begin{cases} 
  f_1 = A(B + C + D + E) + ABCDE \\
  f_2 = \overline{A}(B + C + D + F) + ABCDF 
\end{cases}
\]

Extract and substitute common expression

\[
G = \begin{cases} 
  g_1 = B + C + D \\
  f_1 = A(g_1 + E) + \overline{AE}g_1 \\
  f_2 = \overline{A}(g_1 + F) + AFg_1 
\end{cases}
\]

Logic optimization flow

LOGIC EQUATIONS

TECHNOLOGY-INDEPENDENT OPTIMIZATION

Factoring
Commonality Extraction

LIBRARY

TECH-DEPENDENT OPTIMIZATION
(MAPPING, TIMING)

OPTIMIZED LOGIC NETWORK
"Closed Book" Technologies

A standard cell technology or library is typically restricted to a few tens of types (NAND, NOR, NOT, AOI) of gate.

- e.g., MSU library: 31 cells
- but may have many different sizes/skews of each

**Standard Cell Library**

- For each cell
  - Functional information
  - Timing information
    - Input slew
    - Intrinsic delay
    - Output capacitance
  - Physical footprint and layout
  - Power characteristics
Sample Library

Gate Type

INVERTER (2)

NAND2 (3)

NAND3 (4)

NAND4 (5)

Gate Area

Alternative mappings to NAND2 gates

Sample Library - 2

AOI21 (4)

AOI22 (5)
**Mapping via DAG* Covering**

- Represent network in canonical form ⇒ subject DAG
- Represent each library gate with canonical forms for the logic function ⇒ primitive DAGs
- Each primitive DAG has a cost
- Goal: Find a minimum cost covering of the subject DAG by the primitive DAGs

* Directed Acyclic Graph

---

**Trivial Covering**

Reduce netlist into ND2 gates → subject DAG

7 NAND2 = 21
5 INV = 10

31 (area cost)
Covering #1

2 INV = 4
2 NAND2 = 6
1 NAND3 = 4
1 NAND4 = 5
19 (area cost)

Covering #2

1 INV = 2
1 NAND2 = 3
2 NAND3 = 8
1 AOI21 = 4
17 (area cost)
CMOS From the Bottom, Up

IC Fabrication and Layout Representation

“Mask” drawings sent to the fabrication facility to make the chips.
Mask set for an n-Fet (circa 1986)

- **Vd = 1V**
- **Vg = 0V**
- **Vs = 0V**

<table>
<thead>
<tr>
<th>Masks</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1: n+ diffusion</td>
</tr>
<tr>
<td>#2: poly (gate)</td>
</tr>
<tr>
<td>#3: diff contact</td>
</tr>
<tr>
<td>#4: metal</td>
</tr>
</tbody>
</table>

Top-down view:

- p-
- n+
- Vd = 1V
- Vg = 0V
- Vs = 0V
- I ≈ nA
- dielectric

Layers to do:
- p-Fet not shown.
- Modern processes have 6 to 10 metal layers (or more) (in 1986: 2).

"Design rules" for masks, 1986 ...

- Poly overhang.
  - So that if masks are misaligned, channel doesn't short out.

- Minimum gate length.
  - So that the source and drain depletion regions do not meet!

- Metal rules:
  - Contact separation from channel, one fixed contact size, overlap rules with metal, etc ...

- Metal rules:
  - Contact separation from channel, one fixed contact size, overlap rules with metal, etc ...

- #1: n+ diffusion
- #2: poly (gate)
- #3: diff contact
- #4: metal
Fabrication

Mask set for an n-Fet ...

\[ V_d = 1V \]
\[ V_g = 1V \]
\[ V_s = 0V \]

\[ I = \mu A \]

Top-down view:

How does a fab use a mask set to make an IC?

Masks

*#1: n+ diffusion
*#2: poly (gate)
*#3: diff contact
*#4: metal
Start with an un-doped wafer ...

UV hardens exposed resist. A wafer wash leaves only hard resist.

Steps
#1: dope wafer p-
#2: grow gate oxide
#3: deposit undoped polysilicon
#4: spin on photoresist
#5: place positive poly mask and expose with UV.

Wet etch to remove unmasked ...

HF acid etches through poly and oxide, but not hardened resist.

After etch and resist removal
Use diffusion mask to implant n-type accelerated donor atoms

Notice how donor atoms are blocked by gate and do not enter channel.

Thus, the channel is "self-aligned"; precise mask alignment is not needed!

Metallization completes device

Grow a thick oxide on top of the wafer.

Mask and etch to make contact holes

Put a layer of metal on chip. Be sure to fill in the holes!
Final product ...

Top-down view:

"The planar process"
Jean Hoerni, Fairchild Semiconductor 1958

p-channel Transistors
**p-Fet: Change polarity of everything**

\[
\begin{align*}
V_{\text{well}} &= V_s = 1V \\
V_g &= 0V \\
V_d &= 0V
\end{align*}
\]

```
V_g \quad \downarrow \quad I_{sd} \\
V_s \\
V_d
```

- **New “n-well” mask**
- "Mobility" of holes is slower than electrons.
- p-Fets drive less current than n-Fets, all else being equal

**Bulk versus SOI Processing**

- "Silicon on Insulator"
  - Lower parasitic capacitance \(\rightarrow\) lower energy, higher-performance
  - Also used for "radiation hard" application (space craft) - sapphire instead of Oxide.
  - 10 - 15% increase in total manufacturing cost due to substrate cost.
Lithography

Current state-of-the-art photolithography tools use deep ultraviolet (DUV) light with wavelengths of 248 and 193 nm, which allow minimum feature sizes down to 50 nm.

Optical proximity correction (OPC) is an enhancement technique commonly used to compensate for image errors due to diffraction or process effects.

Modern Processing Parameters

From 2009 ITRS Roadmap

<table>
<thead>
<tr>
<th></th>
<th>2010</th>
<th>2014</th>
</tr>
</thead>
<tbody>
<tr>
<td># Mask Levels—MPU</td>
<td>35</td>
<td>37</td>
</tr>
<tr>
<td># Mask Levels—DRAM</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Maximum Lithography Field Size—area (mm^2)</td>
<td>858</td>
<td>858</td>
</tr>
<tr>
<td>Maximum Lithography Field Size—length (mm)</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>Maximum Lithography Field Size—width (mm)</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Bulk or epitaxial or SOI wafer size (mm)</td>
<td>300</td>
<td>450</td>
</tr>
</tbody>
</table>

International Technology Roadmap for Semiconductors
http://www.itrs.net/
Processing Enhancements

- **Trench isolation**: Shallow trench isolation (STI), a.k.a. Box Isolation Technique, prevents electrical current leakage between adjacent semiconductor device components.

- **High-K dielectrics / Metal gate**: Replacing the silicon dioxide gate dielectric with a high-κ material allows increased gate capacitance without the concomitant leakage effects.

- **Strained Silicon**: A layer of silicon in which the silicon atoms are stretched beyond their normal interatomic distance leading to better mobility, resulting in better chip performance and lower energy consumption.

- **“Gate Engineering”**: for within-die choice of multiple transistor threshold voltages (Vt) to optimize delay or power.

---

End of Introduction

part 2