Why CS250 and not EE250

- Put IC design expertise into the hands of those best qualified to take advantage of its potential:
- Those with intimate knowledge of computation and algorithms: computer scientists!
- Traditionally, and often today IC design is stratified:
  - Algorithm / architecture
  - Micro-architecture
  - Circuit design
  - Layout

- Better option is “tall thin designer”. Spans all levels of the design and implementation stack.
- Leads to more successful innovation and highly optimized designs.
Enabling System Architects

- Managing the complexity is the key challenge. Manipulating multiple levels of design complexity is difficult and continually getting worse (remember Moore’s Law).

- Approach:
  1. Borrow ideas from software (hierarchy, libraries, design patterns, ...)
  2. Focus on design representations
  3. Practice using computer aided design tools
  4. In the context of some application domain
  5. Access to Silicon “foundries” for fabrication

Course Format (1)

The new CS250 (as of Fall 2009)

- VLSI design for system architects.
  - Focus on common ASIC design methodology:
    - RTL synthesis and standard cell implementation. No transistor level layout.
  - Back to a “design centric course”. Learn by doing.
    - Requires a lot of infrastructure set up (thanks to Yunsup Lee, Brian Zimmer, Brian Richards)
  - Entire class worked implementing RISC processors.
    - Many variations on a theme.
    - This semester focus on image processors - more details later.
Course Format (2)

- Most closely related courses:
  - CS 150 - undergraduate digital design. Prerequisite.
  - CS 152/252 Computer Architecture / Microarchitecture.
  - EE 141/242 Transistor level circuits and layout.
  - EE 244 Computer Aided Design of ICs (CAD algorithms)

Course Theme:
How do we get the best design results from the standard design flow using tradeoffs in area/performance/energy and exploring micro-architectural alternatives.

Course Structure

- Check Website Calendar/Info for details
- Weeks 1-7:
  - Lectures on fundamentals of “ASIC” design
  - Lab exercises to learn CAD tools
- Weeks 8-14:
  - Project related activities
    - Project group presentation (proposal, progress, final report)
    - “private project meetings” : instructors meet in private with groups
  - Grading: 5% Class Participation, 25% Labs, 70% Project
- Please, no Laptop/iPad/handheld use in class. We will have a short break midway in each class so you can catchup on email, etc.
Some Important Tentative Dates

- Lab 1 Due: Sep 10 (Monday)
- Lab 2 Due: Sep 24
- Brief Oral Project Proposal: Oct 4
- Written Project Proposal Due: Oct 8
- Lab 3 Due: Oct 15
- Project Final Presentations: Dec 7 (RRR week)
- Final Project Report: Dec 12 midnight

These are all hard deadlines, so please budget your time accordingly. Total of 4 late days for labs.

We will assist you all we can to help you make the deadlines.

More Course Details

- Discussion section TBA.
  - Very important for tips on doing the labs and project
- You will need to get a named instructional account to log onto our servers installed with the CAD tools.
- Piazza for all Q/A, announcements, etc., check website.
- Instructor office hours on the web.
- Enrollment
  - Undergrad: need to have taken CS150 (or equivalent) with B+ or better.
  - Grad: we assume you have taken undergraduate digital design. If not, see us for remedial materials.
- Design Language
  - For all, we assume Verilog/VHDL experience.
  - However, we will be introducing you to a brand new hardware design language, call Chisel (under construction.)
Project Details

- Project groups of 2 people.
- Start with functional specification for an image processor, explore multiple micro-architectural variations to optimize performance or energy efficiency.
  - Examples: edge detection, segmentation, optical flow detection, compression, ...
- Within the pattern(s) that you choose, generate a set of VLSI implementations performing a design space exploration determining the Pareto optimal points in the performance, area, and energy efficiency space.
- Lots of background in a few weeks.

End of Introduction part 1
What has changed in 30 years since the early days of chip design?

Silicon Technology Advances

Feature Size Scaling

0.7x every 3 years

0.7x every 2 years

New technology generation every 2 years
Secondary driver: Wafer size

Wafer size conversions offset trend of increasing wafer processing cost


Processing advances

4µm

45nm
IC Technology Stuff (1)

- **Feature size:**
  - then: ~4µm  now: .028µm

- **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)

Interesting, though, 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 coming soon:** FinFETs

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

- **Lithographic Mask Costs:**
  - then: few $k  now: $M (full die, 65, 45, 28nm)
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.

- 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*: usually 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 Alternative</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 sizes are customized 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, buffers, ...
- Each cell comes complete with:
  - layout (perhaps for different technology nodes and processes),
  - Behavioral 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, vias, 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:

  Sea-of-gates, structured ASIC, master-slice.
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 0.35um chip). Relatively Low cost per die.
- **FPGAs**: Very low NRE costs. Relatively low silicon efficiency $\Rightarrow$ 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 \(\approx \$650,000\) for 130nm and \(\$1.4M\) for 90nm.

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, ...

**SIP, SOP, MCM interesting alternatives.**

Modern ASIC Methodology and Flow

**RTL Synthesis Based**

- HDL specifies design as combinational logic + state elements
- 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
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

Eg
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

Logic Synthesis

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

  Verilog HDL \[\rightarrow\] Synthesis Tool \[\rightarrow\] circuit netlist

- 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

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

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

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

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 ...
Fabrication

Mask set for an n-Fet ...

Vd = 1V  Vg = 1V  Vs = 0V

Top-down view:

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

How does a fab use a mask set to make an IC?
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!
p-channel Transistors

Final product...

"The planar process"

Jean Hoerni,
Fairchild Semiconductor
1958

Top-down view:

p- 

oxide

n+

n+

Vd

Vs

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

\[ V_{well} = V_s = 1V \quad V_g = 0V \quad V_d = 0V \]

- \( l = \mu A \)
- **dielectric**

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 -> 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

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

- 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.

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>
Processing Enhancements

- **Trench isolation**: Shallow trench isolation (STI), a.k.a. Box Isolation Technique, prevents current leakage between n-well and p-well devices.

- **High-K dielectrics / Metal gate**: Replacing the silicon dioxide gate dielectric with a high-$\kappa$ 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