What is EENG 426?

The microelectronics revolution!

Moore, 1965
What is EENG 426/CPSC 459/ENAS 876?

- Converting algorithms to chips
- Algorithms
  - Expressed in a programming notation
  - Many “advanced” language features omitted
- Silicon
  - The “back end” of the compiler, our “assembly language”
  - Physical details will be viewed as optimization parameters whenever possible

Very Large Scale Integration

Projects

Examples from previous years? There are no previous years...

... but for the VLSI design class:

- Accumulator memory
- Floating-point multiplier
- Mastermind
- Karpus-Strong sound synthesis
- XOR-based scrambler/descrambler

Projects

Lectures: ML 104, MW 11:35–12:50
Drop-in office hours: DL 504, W 1:30-3:30pm
Piazza: Check regularly! Announcements will go here
Grading:

- 65% labs (5 labs: 5%+10%+15%+15%+20%)
- 10% quizzes (9, on Wednesdays, will drop lowest grade)
- 20% mid-term
- 5% instructor discretion
**Labs:**
- You can do them on your laptop.
- Unix environment (Linux): an image will be provided with all the software pre-installed, with additional pieces as the labs progress.
- All tools are either open source, or written by my research group.

**Lab writeup:**
- One page executive summary for each lab
- Electronic submission

**Late Policy:**
0 (with the usual exceptions)
If you contact me in advance, I can be flexible.

**Collaboration:**
- First few labs are individual; later labs can be in groups of two
- General discussions among students permitted
- Lab work is expected to be done separately

**Text:**
- No textbook.

- General reference for chip design
  - Weste/Harris, CMOS VLSI Design
  - Mead/Conway, Introduction to VLSI Systems
- Course notes will be posted online

**Chip:**
- "Tape-in" this semester for all students.
- "Tape-out" for students whose designs pass all final checks—hopefully everyone
  - test chip + testing report in the Spring.

**Topics**
- Transistors
- Switching networks
- Production rules
- Concurrency
- Syntax-directed translation
- High-level optimization
- Synthesis
- Datapath and function blocks
- Floorplanning
- Energy and delay
- Analog effects
- Leakage
- Metastability
- Voltage scaling
- Scaling
- System examples
Design Flow

... or how to make a chip:

- Functional specification
- Architectural specification
- Circuits
- Abstract mask geometry
- Physical masks
- Chip

Asynchronous Design

"Today, a chip goes to fab with 100 errors. It takes 8 runs to 'debug' it. It is finally shipped to the customer with 4 errors left. Most errors are timing errors."

(Carver Mead, 1993)

The design of the Pentium Pro required a total of 300 staff years for pre- and post-silicon validation.

(Source: Albert Yu, Intel)

Major design issues today:

- verification
- meeting timing budgets
- meeting power budgets

A digital circuit is asynchronous when it does not use a clock to implement sequencing.

A circuit is said to be delay insensitive (DI) when its correct operation is independent of the delays in gates and in the wires connecting the gates, assuming that the delays are finite and positive.

Ideally, we would use delay-insensitive circuits to abstract away from all physical details.


Asynchronous Design

Compromises:
- Quasi delay-insensitive circuits
  - “isochronic forks”
- Self-timed circuits
  - “isochronic regions”
- Timed circuits
  - path 1 is always faster than path 2

The Technology: CMOS

Complementary Metal Oxide Semiconductor
Goals:
- Correctness
- Performance
- Energy efficiency
- ...

“VLSI is a statement about system complexity, not transistor size or circuit performance.” – Mead, 1979

The Device

Metal Oxide Semiconductor Field Effect Transistor

Two basic types: \( n \)-channel and \( p \)-channel

Geometry

Rotate the device and look from the top:
Abstraction

Mask Geometry

Assumption:

- all dimensions are multiples of a scaling parameter $\lambda$

*Scalable CMOS (SCMOS) design rules.*

When technology improves, adjust $\lambda$ and reuse the design!

This is no longer a good abstraction (more later ...)

Scaling

<table>
<thead>
<tr>
<th>Linear dimension</th>
<th>$\lambda$</th>
<th>$\lambda/\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>$V$</td>
<td>$V/\alpha$</td>
</tr>
<tr>
<td>Current</td>
<td>$I$</td>
<td>$I/\alpha$</td>
</tr>
<tr>
<td>Power</td>
<td>$VI$</td>
<td>$VI/\alpha^2$</td>
</tr>
<tr>
<td>Delay</td>
<td>$t$</td>
<td>$t/\alpha$</td>
</tr>
<tr>
<td>Energy</td>
<td>$VIt$</td>
<td>$VIt/\alpha^3$</td>
</tr>
</tbody>
</table>

... per device.

Moore’s law: # of transistors doubles every 18 months

Complexity analogy: Seitz and Mead, 1979. Imagine a city where streets are wires with 200m between blocks.

<table>
<thead>
<tr>
<th>Year</th>
<th>Spacing</th>
<th>Chip size</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1963</td>
<td>50$\mu$m</td>
<td>1mm</td>
<td>town (4km)</td>
</tr>
<tr>
<td>1975</td>
<td>10$\mu$m</td>
<td>5mm</td>
<td>county (100km)</td>
</tr>
<tr>
<td>1985</td>
<td>2$\mu$m</td>
<td>10mm</td>
<td>state (100km)</td>
</tr>
<tr>
<td>1995</td>
<td>0.5$\mu$m</td>
<td>20mm</td>
<td>continent (8000km)</td>
</tr>
</tbody>
</table>

Today: 0.010$\mu$m (10nm), 25mm chip size!

Key observation: abstraction!

Reuse:

- design tools, methods, circuits, abstract geometry

... as long as we understand how scaling works.

Mainstream modern CMOS process:

- Gate: 0.014$\mu$m (14nm)
- Voltage: 0.9V
- FO1 inverter: < 1$\mu$s delay
A counter example:

\[
\text{COUNT} \equiv \star \left[ \begin{array}{c}
imc \rightarrow x \leftarrow x + 1; \text{inc} \\
\text{reset} \rightarrow x \leftarrow 0; \text{reset} \\
\text{read} \rightarrow \text{read}\{x\}
\end{array} \right]
\]

At this level, we have not specified any protocol, encoding, or even how many bits are used to represent \(x\).

Pick data encodings and communication protocols

- \((ai, ao) = \text{inc}_k\)
- \((ri, ro) = \text{inc}_{k+1}\)
- \((bi, bo) = \text{reset}_k\)
- \((si, so) = \text{reset}_{k+1}\)
- \((ci, cto, cfo) = \text{read}_k\)

Basic idea:

- Handshake protocols used to implement communication
- Data encoded in a fashion that encodes when it is valid

Connect \(N\) of these to get an \(N\)-bit counter.
Design example

Production rules

\[-xf \land s \rightarrow xt^\uparrow\]
\[ro \land ao \lor bo \rightarrow xt^\downarrow\]
\[ai \land xf \land ri \land ao \rightarrow ro^\uparrow\]
\[ri \land xf \rightarrow ro^\downarrow\]
\[ai \land ro \lor ai \land s \rightarrow ao^\uparrow\]
\[\neg ai \land ro \land s \rightarrow ao^\downarrow\]
\[\neg xt \land s \rightarrow xf^\uparrow\]
\[xf \land ci \rightarrow cto^\uparrow\]
\[\neg ci \rightarrow cto^\downarrow\]
\[ao \land s \rightarrow xf^\downarrow\]
\[xf \land ci \rightarrow cfo^\uparrow\]
\[\neg ci \rightarrow cfo^\downarrow\]

Design methodology

1. Begin with a simple, sequential description
2. Decompose it into small, concurrent, processes
3. Pick encodings and interface conventions
4. Transform each process into production rules

Most of the high-level architectural decisions are taken within the first two steps.

Good circuits are chosen in the last two steps.

Decomposition by program transformations.

- Decompose by using many small steps.
- Each step should be easily justifiable.
- The final program can be shown to be correct by the chain of transformations that constructed it.
- Transformation to production rules can be formally justified.
- The layout can be checked against the production rules.

\[\Rightarrow\] final chip can be shown to be a valid implementation of the original sequential specification.