EENG 426/CPSC 459/ENAS 876
Silicon Compilation

Rajit Manohar

Computer Systems Lab

http://csl.yale.edu/~rajit

Fall 2018

Yale

AVLSI

What is EENG 426?

"Computers in the future may weigh no more than 1.5 tons."
— *Popular Mechanics, 1949*

"I think there is a world market for maybe five computers."
— *Thomas Watson, Chairman of IBM, 1943*

"I have traveled the length and breadth of the country and talked with the best people, and I can assure you that data processing is a fad that won't last out the year."
— *editor in charge of Prentice Hall business books, 1957*

... so what happened?

Yale

AVLSI

What is EENG 426?

The microelectronics revolution!

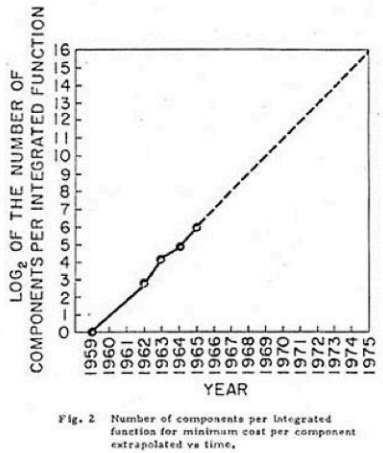


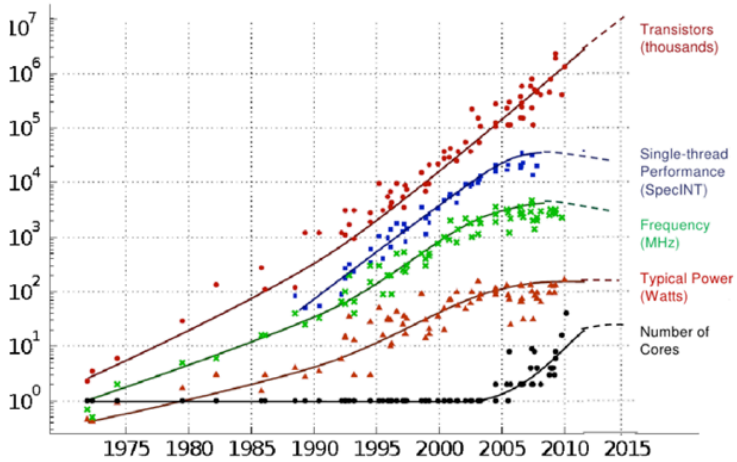
Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

Moore, 1965

Yale

AVLSI

What is EENG 426?



Horowitz/Olokutun groups, ≈2010

Yale

AVLSI

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

Yale

Projects

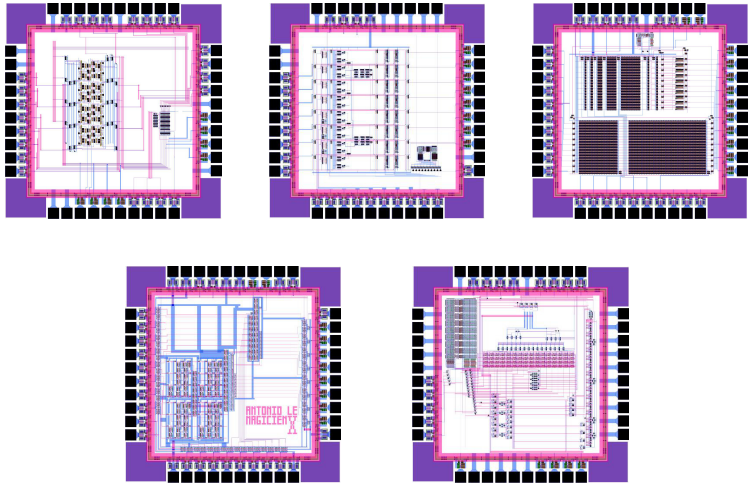
Examples from previous years? There are no previous years...

... but for the VLSI design class:

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

Yale

Projects



Yale

Administrivia

- 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

Yale

Administrivia

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

Yale AVLSI

Administrivia

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

Yale AVLSI

Administrivia

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.

Yale AVLSI

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

Yale AVLSI

Design Flow

. . . or how to make a chip:

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

Yale AVLSI

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

Yale AVLSI

Asynchronous Design

The main issue: design complexity

- Functional verification (synchronization, functionality)
- Meeting performance and power targets
- Interfaces
- Productivity

$\Rightarrow$ Use high level of abstraction

$\Rightarrow$ Treat physical details as parameters to be adjusted for performance, **not** correctness.

Yale AVLSI

Asynchronous Design

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.

Alain J. Martin. "The limitations to delay insensitivity in asynchronous circuits." *Proc. ARVLSI*, 1990.

Rajit Manohar and Yoram Moses. "The Eventual C-Element Theorem for Delay-Insensitive Asynchronous Circuits." *Proc. ASYNC*, May 2017.

Yale AVLSI

Asynchronous Design

Compromises:

- Quasi delay-insensitive circuits
  - "isochronic forks"
- Self-timed circuits
  - "isochronic regions"
- Timed circuits
  - path 1 is always faster than path 2

Yale AVLSI

The Technology: CMOS

Complementary Metal Oxide Semiconductor
Goals:

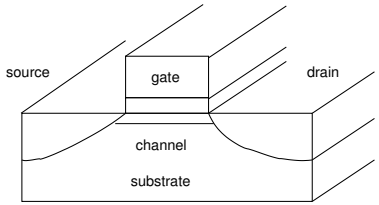
- Correctness
- Performance
- Energy efficiency
- ...

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

Yale AVLSI

The Device

Metal Oxide Semiconductor Field Effect Transistor

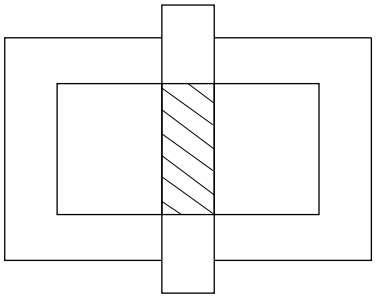


Two basic types: *n*-channel and *p*-channel

Yale AVLSI

Geometry

Rotate the device and look from the top:



Yale AVLSI

Abstraction

Mask Geometry

Assumption:

- all dimensions are multiples of a scaling parameter λ

Scalable CMOS (SCMOS) design rules.

When technology improves, adjust λ and reuse the design!

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

Yale

Scaling

Linear dimension	λ	λ/α
Voltage	V	V/α
Current	I	I/α
Power	VI	VI/α^2
Delay	t	t/α
Energy	VIt	VIt/α^3

... per device.

Yale

Scaling

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.

Year	Spacing	Chip size	City
1963	$50\mu m$	$1mm$	town ($4km$)
1975	$10\mu m$	$5mm$	county ($100km$)
1985	$2\mu m$	$10mm$	state ($100km$)
1995	$0.5\mu m$	$20mm$	continent ($8000km$)

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

Yale

Scaling

Key observation: abstraction!

Reuse:

- design tools, methods, circuits, abstract geometry

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

Mainstream modern CMOS process:

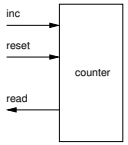
- Gate: $0.014\mu m$ ($14nm$)
- Voltage: $0.9V$
- FO1 inverter: $< 1ps$ delay

Yale

Design example

A counter example:

$$\begin{aligned} COUNT \equiv *[&[\overline{inc} \longrightarrow x := x+1; inc \\ &[]\overline{reset} \longrightarrow x := 0; reset \\ &[]\overline{read} \longrightarrow read!x \\ &]] \end{aligned}$$



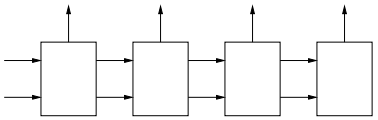
At this level, we have not specified any protocol, encoding, or even how many bits are used to represent x.

Design example

Pick a representation for x.

$$\begin{aligned} COUNT_k \equiv *[&[\overline{inc_k} \wedge x_k \longrightarrow x_k\downarrow; inc_{k+1}; inc_k \\ &[]\overline{inc_k} \wedge \neg x_k \longrightarrow x_k\uparrow; inc_k \\ &[]\overline{reset_k} \longrightarrow x_k\downarrow; reset_{k+1}; reset_k \\ &[]\overline{read_k} \longrightarrow read_k!x_k \\ &]] \end{aligned}$$

Connect N of these to get an N-bit counter.



Design example

Pick data encodings and communication protocols

- $(ai, ao) = inc_k$
- $(ri, ro) = inc_{k+1}$
- $(bi, bo) = reset_k$
- $(si, so) = reset_{k+1}$
- $(ci, cto, cfo) = read_k$

Basic idea:

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

Design example

$$\begin{aligned} &COUNT_k \equiv \\ &*[[\underbrace{ai}_{\overline{inc_k}} \wedge xt \longrightarrow \underbrace{xt\downarrow; xf\uparrow}_{x_k\downarrow}; \underbrace{ro\uparrow; [ri]; ro\downarrow; [\neg ri]}_{inc_{k+1}}; \underbrace{ao\uparrow; [\neg ai]; ao\downarrow}_{\text{rest of } inc_k} \\ &\quad []ai \wedge xf \longrightarrow \underbrace{xf\downarrow; xt\uparrow}_{x_k\uparrow}; ao\uparrow; [\neg ai]; ao\downarrow \\ &\quad [] \underbrace{bi}_{\overline{reset_k}} \longrightarrow xt\downarrow; xf\uparrow; \underbrace{so\uparrow; [si]; so\downarrow; [\neg si]}_{reset_{k+1}}; \underbrace{bo\uparrow; [\neg bi]; bo\downarrow}_{\text{rest of } reset_k} \\ &\quad [] \underbrace{ci}_{\overline{read_k}} \longrightarrow \underbrace{[xt \longrightarrow cto\uparrow \; [] \; xf \longrightarrow cfo\uparrow]; [\neg ci]; cto\downarrow; cfo\downarrow}_{\text{rest of } read_k!x_k} \\ &]] \end{aligned}$$

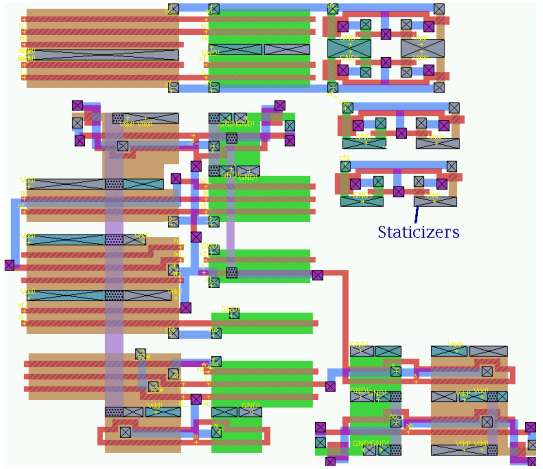
Design example

Production rules

$$
\begin{aligned}
\neg xf \wedge s &\rightarrow xt\uparrow \\
ro \wedge ao \vee bo &\rightarrow xt\downarrow \\
ai \wedge \neg xf \wedge \neg ri \wedge \neg ao &\rightarrow ro\uparrow \\
ri \wedge xf &\rightarrow ro\downarrow \\
ai \wedge ro \vee ai \wedge s &\rightarrow ao\uparrow \\
\neg ai \wedge \neg ro \wedge \neg s &\rightarrow ao\downarrow \\
\neg xt \wedge \neg s &\rightarrow xf\uparrow \\
ao \wedge s &\rightarrow xf\downarrow \\
ai \wedge \neg xt \wedge \neg ao &\rightarrow s\uparrow
\end{aligned}
\qquad
\begin{aligned}
\neg ai \wedge xt &\rightarrow s\downarrow \\
bi \wedge \neg si &\rightarrow so\uparrow \\
xf \wedge \neg bi \wedge si &\rightarrow so\downarrow \\
so &\rightarrow bo\uparrow \\
\neg so &\rightarrow bo\downarrow \\
xt \wedge ci &\rightarrow cto\uparrow \\
\neg ci &\rightarrow cto\downarrow \\
xf \wedge ci &\rightarrow cfo\uparrow \\
\neg ci &\rightarrow cfo\downarrow
\end{aligned}
$$

Yale

Design example



Yale

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.

Yale

Design methodology

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.

Yale