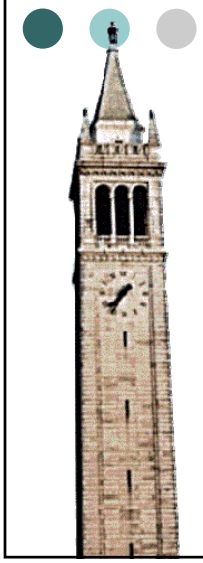


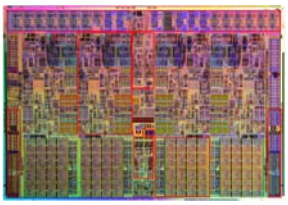
Fundamental Algorithms for System Modeling, Analysis, and Optimization

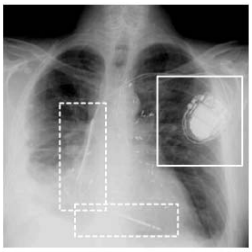
Jaijeet Roychowdhury, Stavros Tripakis
UC Berkeley
EECS 144/244
Fall 2015

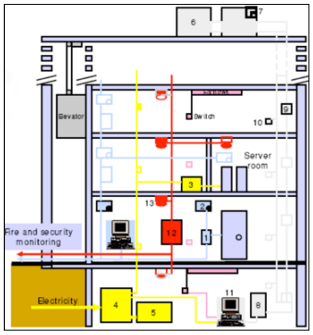
Copyright © 2010-date, E. A. Lee, J. Roychowdhury, S. A. Seshia, S. Tripakis, All rights reserved

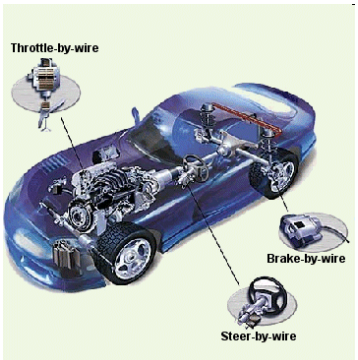
Lecture 1: Introduction, Logistics

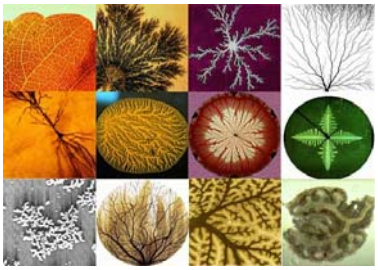
Models and Algorithms in the Design & Analysis of Complex Systems



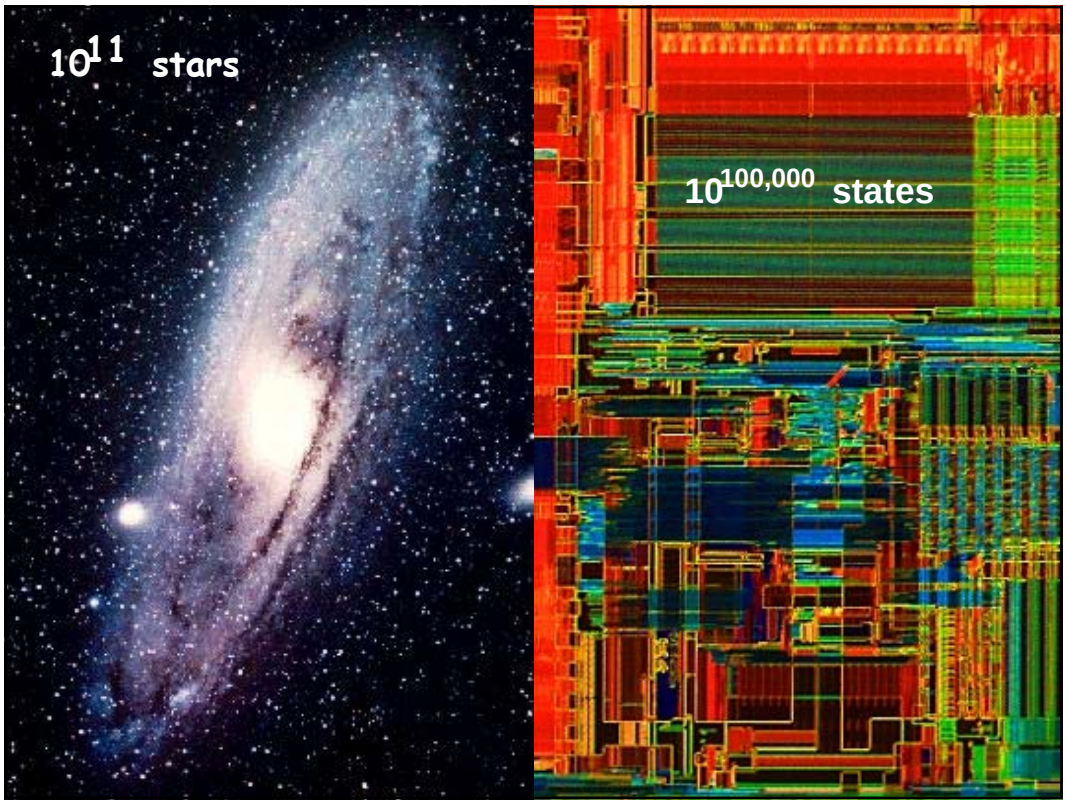




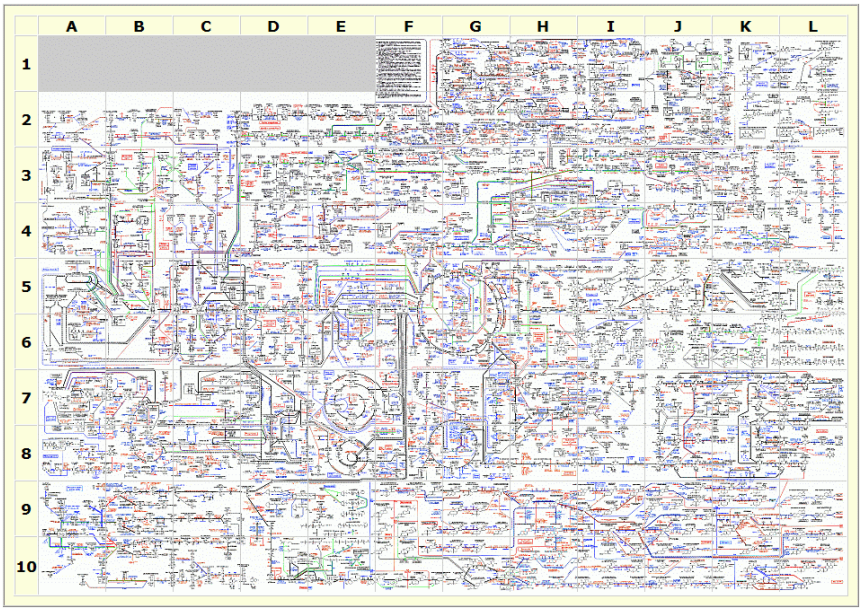




Complex systems



Metabolic Pathway Maps



credits: expasy.ch/biomap/

A simple program

```
int x := input an integer number > 1;

while x > 1 {
  if x is even
      x := x / 2;
  else
      x := 3*x + 1;
}
```

A simple program?

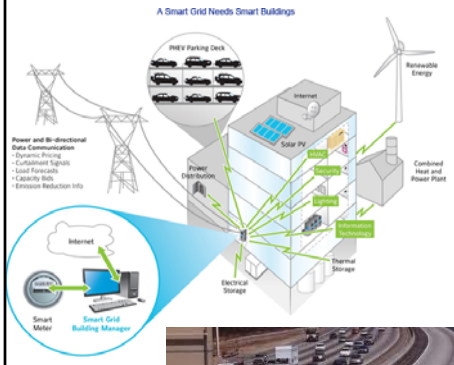
```
int x := input an integer number > 1;

while x > 1 {
  if x is even
      x := x / 2;
  else
      x := 3*x + 1;
}
```

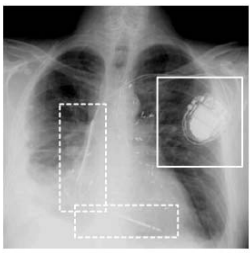
```
Run starting at 31: 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466
233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425
1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077
9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20
10 5 16 8 4 2
```

Collatz conjecture:
the program terminates for every input.
Open problem in mathematics.

Safety-critical systems

- Smart cars, roads, buildings, power grid, cities, ...





...

How to design such systems?

"By hand" is not an option…

Designers need tools!

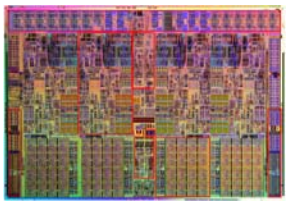
Not just paper and pencil: computer automation.

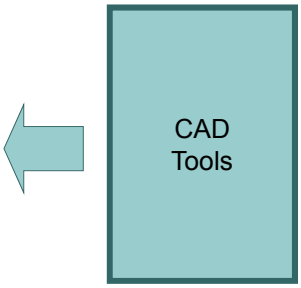
=> **computer-aided design**

Goal of this course:
Teach you the fundamentals so that you become a good tool user, but also a tool maker.

Computer-Aided Design (CAD) for ICs / Electronic Design Automation (EDA)

731M transistors





Approaches to system design (1)

- ***Trial-and-error*** approach:
  - Build prototype
  - Test it, find errors
  - Fix errors
  - Repeat

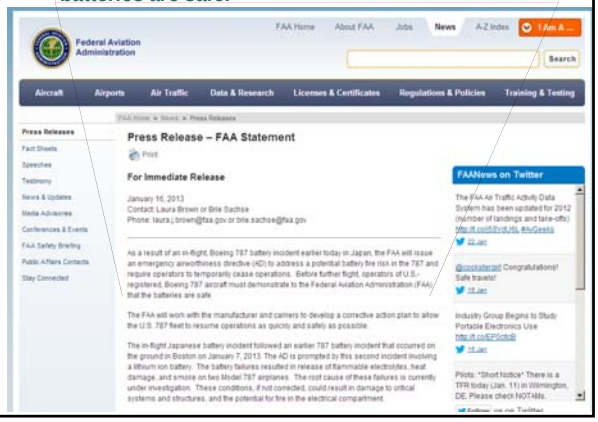
Design by trial-and-error

- Toyota unintended acceleration incidents
- Millions of cars recalled
- Cost: $ billions
- U.S. National Highway Transportation Safety Administration's (NHTSA) report concluded that electronic throttle control systems were not the cause.

Design by trial-and-error

- Boeing 787 grounded
- "*All-Nippon today announced it had canceled 320 flights, including 51 international flights, on 787s affecting a total of 46,800 passengers*" [San Jose Mercury News, 1/22/2013]
- FAA restriction finally lifted in April 2013.

As a result of an in-flight, Boeing 787 battery incident earlier today in Japan, the FAA will issue an emergency airworthiness directive (AD) to address a potential battery fire risk in the 787 and require operators to temporarily cease operations.
Before further flight, operators of U.S.-registered, Boeing 787 aircraft must demonstrate to the Federal Aviation Administration (FAA) that the batteries are safe.

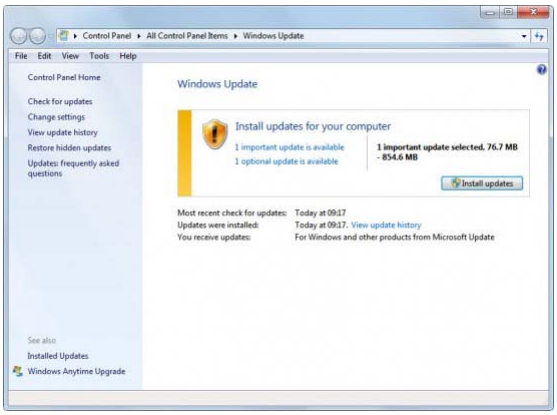


Design by trial-and-error

- Last but not least ...

Design by trial-and-error

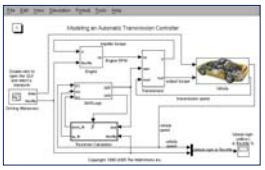
- Software!



Approaches to system design (2)

- Rigorous, ***"model-based"*** design:
  - Build model ("executable specification") of system
    - Before building a prototype of the system itself
  - Analyze the model, find errors
  - Fix errors in the design (model)
  - Repeat until the design seems OK
  - Give models/specs to someone (or to a computer) to implement them
    - Need to ensure properties are preserved during implementation
- Better for affordability:
  - Catch design errors early => easier / less costly to fix
- Better for dependability:
  - Sometimes can formally prove that design is correct
- Gaining acceptance in the industry

The Elements of Model-Based Design



How to describe what we want?

Modeling

How to be sure that this is what we want?

Analysis

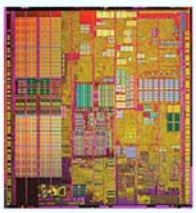
Implementation, Optimization

How to build it?
Automatically
Correct-by-construction

From standard compilers ...

```
class HelloWorldApp {
  public static void
  main(String[] args) {
    System.out.println(
      "Hello World!");
  }
}
```

compiler



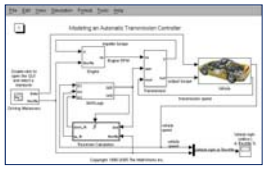
source code
(C, Java, ...)

machine code

type checking, debugging, static analysis, ...

… to system compilers

Vision: *modeling/simulation languages of today will become the **system-programming** languages of tomorrow*



Rich languages: concurrency, time, robustness, reliability, energy, security, …

system compiler

Complex execution platforms: networked, distributed, multicore, …

Powerful analyses: model-checking, WCET analysis, schedulability, performance analysis, reliability analysis, …

The promise of rigorous, "formal" methods

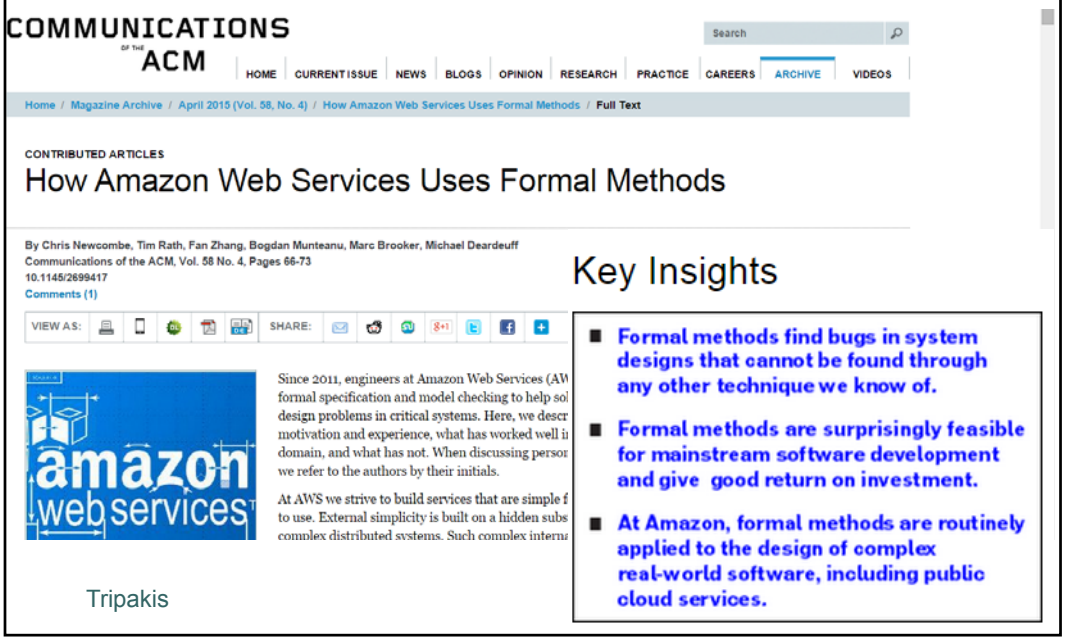
Beyond simple simulation and testing: prove correctness!

> *"Testing shows the presence, not the absence of bugs."* – **Dijkstra**

Formal verification (model checking): "exhaustive simulation" (check all possible system behaviors)

Success story in the hardware industry: "verification engineer" standard title in today's EDA companies

“Formal methods” in the software industry?



Tripakis

Caveat

In real life, we need both MBD and trial-and-error methods. Why?

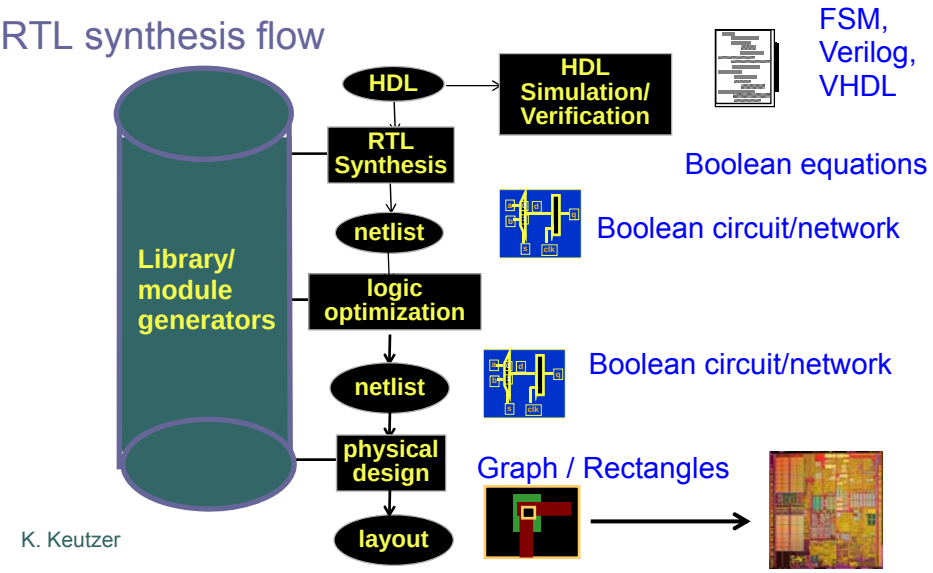
1. We cannot trust our models 100%
2. All models are abstractions of reality. They make assumptions that need not hold.
   - E.g., road condition, weather condition, …
3. Analysis and optimization methods also have their limitations.
   - As we will see in this course.

Model-based design seems fine, but …

- There are many systems, of different kinds
- People have been designing these for decades
- Can we pretend to find a single design method that works for every kind of system?
- Of course not
- Thesis:
  - System design is a science
  - There is a body of knowledge (models, algorithms, …) which is fundamental to that science
  - This body of knowledge is applicable to many application domains (circuits, SW, embedded systems, bio, …)

Example of a successful model-based design flow

RTL synthesis flow



CAD at Berkeley: History

- CAD research at Berkeley: design tools with an impact

late 60s and 70s
- CANCER, SPICE (Rohrer, Nagel, Cohen, Pederson, ASV, Newton, etc.)
- SPLICE (Newton)

80s
- MAGIC (Ousterhout et. al.)
- Espresso (Brayton, ASV, Rudell, Wang et. al.)
- MIS (Brayton, ASV, et. al.)

90s
- SIS, HSIS (Brayton, ASV et. al.)
- VIS (Brayton, ASV, Somenzi et. al.)
- Ptolemy (Lee et. al.)

2000-date
- MVSIS (Brayton, Mishchenko et. al.)
- BALM (Mishchenko, Brayton et. al.)
- ABC (Mishchenko, Brayton et. al.)
- MetroPolis, Metro II, Clotho (ASV et. al.)
- Ptolemy II, HyVisual (Lee et. al.)
- UCLID, GameTime, Beaver (Seshia et. al.)

Lecture Outline

- Introduction to Jaijeet, Stavros, and all of you
- Some of the topics covered in this course
  - Digital systems (circuits)
  - Cyber-Physical systems
  - Continuous-time systems
- Course logistics

Jaijeet Roychowdhury

- Past:
  - PhD: UCB EECS (1993)
  - Bell Labs (1993-2000)
  - CeLight (2000-2001)
  - Univ. of Minnesota (2001-2008)
  - co-founder, Berkeley Design Automation (2003)
- Research interests:
  - modelling/simulation of continuous-time systems
  - widely-separated time scale problems
  - noise and variability
  - automated macromodelling

Stavros Tripakis

Adjunct Assoc. Prof. and Researcher

UC Berkeley (2009 – now)

- Past:
  - Research Scientist: Cadence Design Systems, Berkeley, 2006 -- 2008
  - Postdoc: Berkeley, 1999 – 2001
  - Research Scientist: CNRS, Verimag, France, 2001 – 2006
  - PhD: Verimag Laboratory, Grenoble, France, 1998
  - Undergrad: University of Crete, Greece, 1992
- Research interests
  - System design, modeling, and analysis (DMA)
  - Formal methods
  - Computer-aided verification and synthesis
  - Compositionality, contracts, interfaces
  - Embedded and cyber-physical systems

Round of introductions

Your name, research/professional interests, grad/undergrad student, ...

29

Quiz

- Express the following in your favorite mathematical formalism:
  - You can fool some people sometimes
  - You can fool some of the people all of the time
  - You can fool some people sometimes but you can't fool all the people all the time [Bob Marley]
  - You can fool some of the people all of the time, and all of the people some of the time, but you can not fool all of the people all of the time [Abraham Lincoln]

30

Course topics

Algorithms for Discrete Models

- Automata, state machines, transition systems, logic, temporal logic
- State-space exploration, reachability analysis, model-checking
- Boolean function representation and manipulation
- Synchronous and asynchronous composition

Algorithms for Continuous Models

- Solving non-linear equations

Algorithms for Cyber-Physical Models

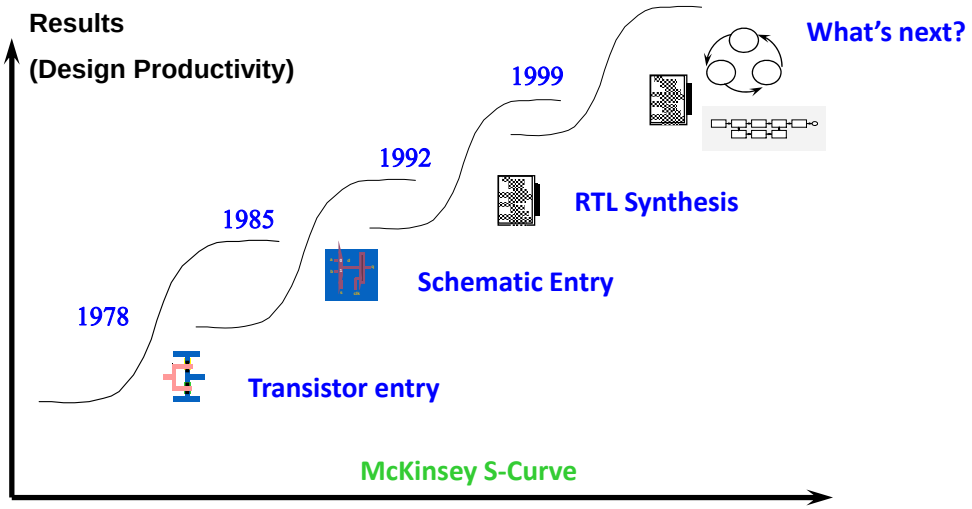
- Timed and discrete-event systems
- Discrete-event simulation

Cross-cutting Topics

- Timing analysis
- Controller and program synthesis

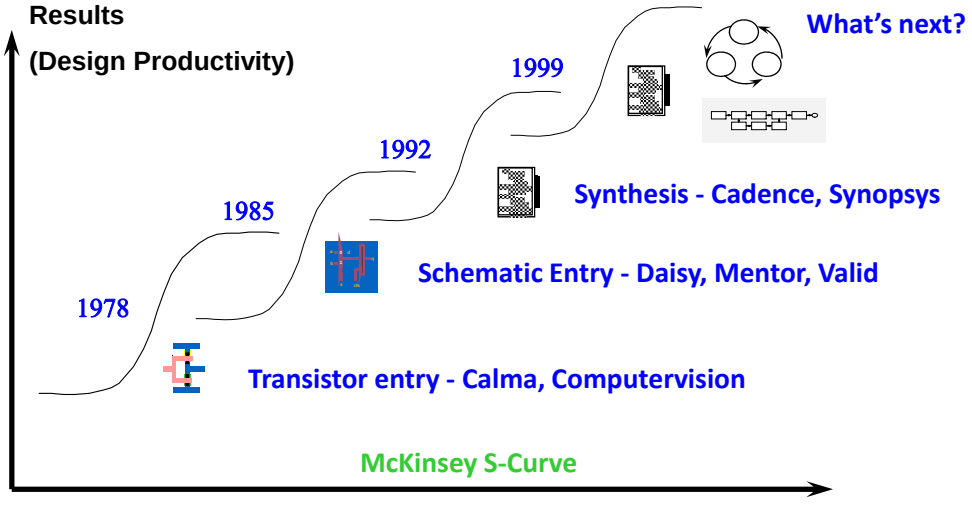
Part I: Digital Systems

Evolution of Digital IC Design



K. Keutzer

Evolution of the EDA Industry

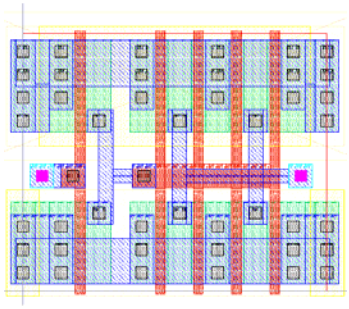


K. Keutzer

Transistor Era

Key tools:

- Transistor-level layout – e.g. Calma workstation
- Transistor-level simulation – e.g. Spice
- Bonus: transistor-level compaction – e.g. Cabbage



Size of circuits: 10's of transistors to few thousand

Key abstractions and technologies:

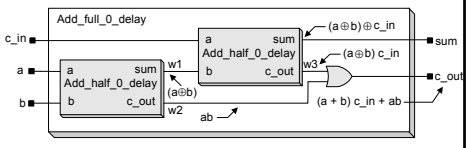
- Transistor-level modeling, simulation
- Logical gates- NAND, NOR, FF and cell libraries
- Layout compaction

K. Keutzer

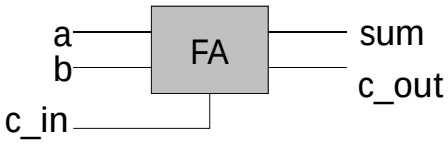
Gate-level Schematic Era

Key tools:

- gate-level layout editor – Daisy, Mentor, valid workstation
- Gate-level simulator
- Automated place and route

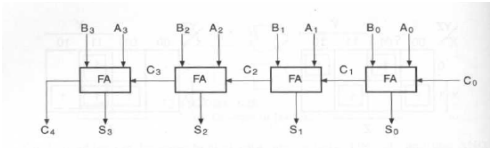


Size of circuits: 3,000 – 35,000 gates (12,000 to 140,000 transistors)



Key abstractions and technologies:

- Logic-level simulation
- Cell-based place and route
- Static-timing analysis



K. Keutzer

RTL Synthesis Era

Key tools:

- Hardware-description language simulator – Verilog, VHDL
- Logic synthesis tool - Synopsys
- Automated place and route – Cadence, Avant!, Magma

Size of circuits: 35,000 gates to …?

Key abstractions and technologies:

- HDL simulation
- Logic synthesis
- Cell-based place and route
- Static-timing analysis
- Automatic-test pattern generation
- Equivalence checking / verification

```
module Half_adder (Sum, C_out, A, B);
  output Sum, C_out;
  input A, B;

  xor      M1 (Sum, A, B);
  and      M2 (C_out, A, B);
endmodule
```

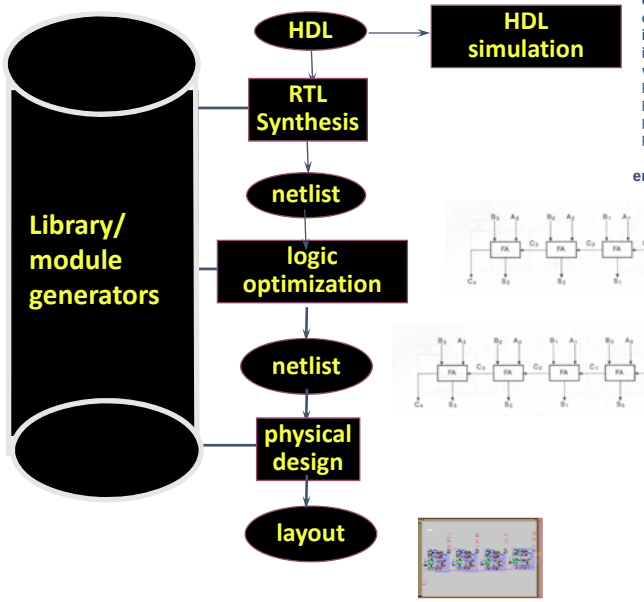
```
module Full_Adder (sum, c_out, a, b, c_in);
  output   sum, c_out;
  input    a, b, c_in;
  wire     w1, w2, w3;
  Half_adder          M1 (w1, w2, a, b);
  Half_adder          M2 (sum, w3, w2, c_in);
  or                  M3 (c_out, w2, w3);
endmodule
```

```
module Full_Adder_4 (sum, c_out, a, b, c_in);
  output     [3:0]sum;
  output      c_out;
  input      [3:0] a, b;
  input      c_in;
  wire        c_in2, c_in3, c_in4;
  Full_adder  M1 (sum[0], c_in2, a[0], b[0], c_in);
  Full_adder  M2 (sum[1], c_in3, a[1], b[1], c_in2);
  Full_adder  M3 (sum[2], c_in4, a[2], b[2], c_in3);
  Full_adder  M4 (sum[3], c_out, a[3], b[3], c_in4);

endmodule
```

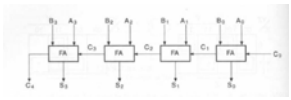
Important modern tool:
SAT solver

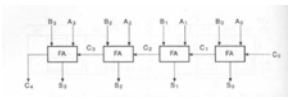
K. Keutzer

RTL Synthesis Flow




```
module Full_Adder_4 (sum, c_out, a, b, c_in);
  output     [3:0]sum;
  output      c_out;
  input      [3:0] a, b;
  input      c_in;
  wire        c_in2, c_in3, c_in4;
  Full_adder  M1 (sum[0], c_in2, a[0], b[0], c_in);
  Full_adder  M2 (sum[1], c_in3, a[1], b[1], c_in2);
  Full_adder  M3 (sum[2], c_in4, a[2], b[2], c_in3);
  Full_adder  M4 (sum[3], c_out, a[3], b[3], c_in4);

endmodule
```





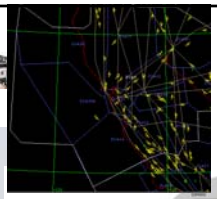
K. Keutzer

Part II: Cyber-Physical Systems

Cyber-Physical Systems (CPS):

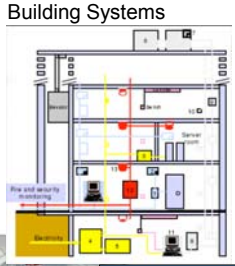
Orchestrating networked computational resources with physical systems

Avionics



Transportation (Air traffic control at SFO)

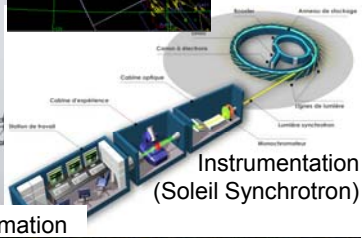
Building Systems



Automotive

E-Corner, Siemens

Telecommunications



Instrumentation (Soleil Synchrotron)

Daimler-Chrysler

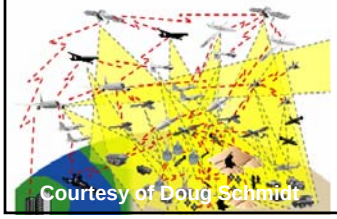
Power generation and distribution

Courtesy of General Electric

Factory automation

Courtesy of Kuka Robotics Corp.

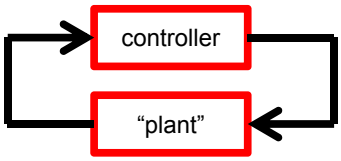
Military systems:



Courtesy of Doug Schmidt

Embedded, Cyber-Physical Systems

- Computers (HW+SW) “embedded” in a physical world
- Cyber = computers (literally “to govern”)
- Physical = the rest
- Typically in a closed-loop (feedback) control configuration (often there are many distributed controllers)



41

CPS Example – Printing Press

Bosch-Rexroth

- High-speed, high precision
  - Speed: 1 inch/ms
  - Precision: 0.01 inch
    -> Time accuracy: 10us
- Open standards (Ethernet)
  - Synchronous, Time-Triggered
  - IEEE 1588 time-sync protocol
- Application aspects
  - local (control)
  - distributed (coordination)
  - global (modes)

Edward A. Lee

Another Example of a CPS Application

STARMAC quadrotor aircraft (Tomlin, et al.)

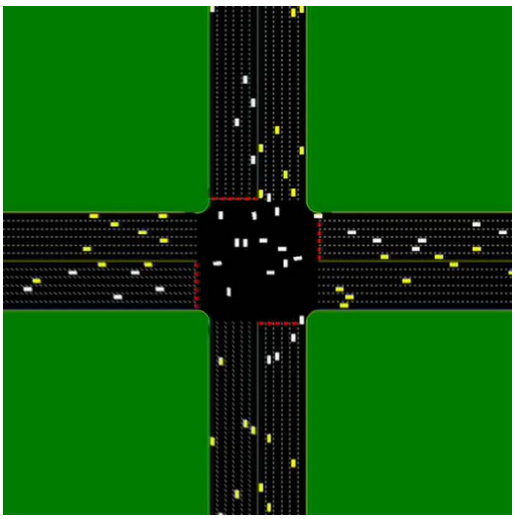
Modeling:
•Flight dynamics
•Modes of operation
•Transitions between modes
•Composition of behaviors
•Multi-vehicle interaction

Design:
•Processors
•Memory system
•Sensor interfacing
•Concurrent software
•Real-time scheduling

Analysis
•Specifying safe behavior
•Achieving safe behavior
•Verifying safe behavior
•Guaranteeing timeliness

Example of a CPS: autonomous intersection management

Courtesy AIM project,
CS Dept., UT Austin



http://www.cs.utexas.edu/~aim/

44

CPS challenges

How to model CPS? What are suitable mathematical formalisms which combine discrete & continuous dynamics?

How to analyze (simulate, verify, …) such models?

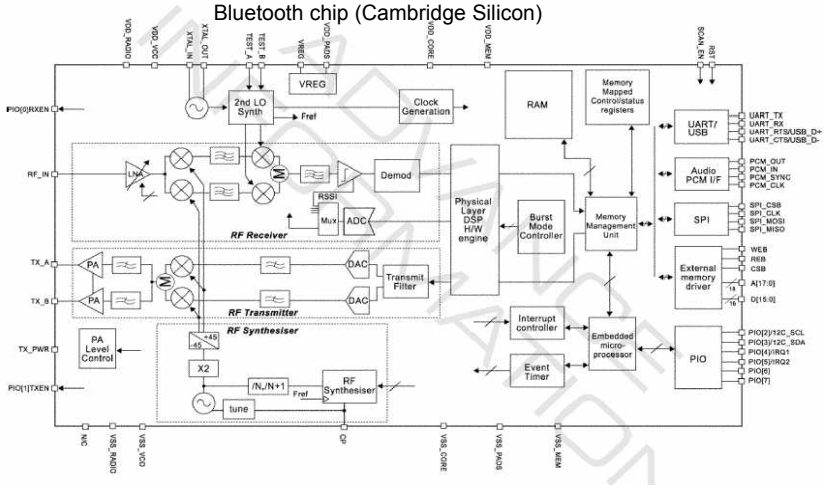
How to derive (ideally, automatically) valid implementations from the models?

Part III: Continuous-Time Systems

Only a few words here. Prof. Jaijeet Roychowdhury will introduce these more thoroughly later in the course.

Analog circuits

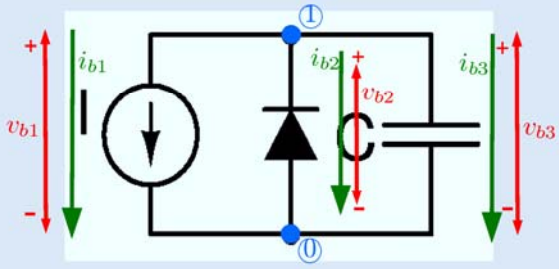
Bluetooth chip (Cambridge Silicon)



Courtesy of J. Roychowdhury, UC Berkeley

47

Diode-Capacitor (Nonlinear) Example



- 1 KCL eqn. at node 1: $i_{b1} + i_{b2} + i_{b3} = 0$
- 3 KVL equations: $v_{b1} = v_{b2} = v_{b3} = e_1$
- 3 BCRs: $i_{b1} = I(t); \quad i_{b2} = -\text{diode}(-v_{b2}; I_S, V_t); \quad i_{b3} = C\dfrac{dv_{b3}}{dt}$
- 7 unks, 7 eqns (incl. 1 differential, 1 nonlinear)
- eliminate by hand: ODEs
  - **(ODEs not always possible)**
  - **analytical soln not available!**
    - **numerical methods needed**

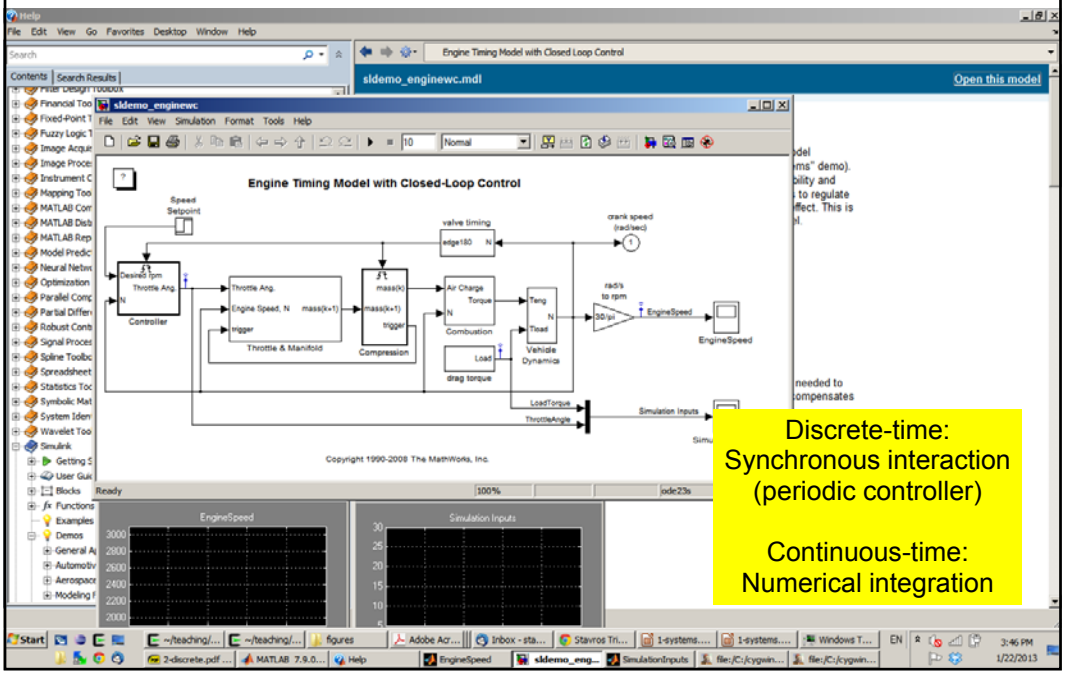
$$I(t) - \text{diode}(-e_1; I_S, V_t) + C\frac{de_1}{dt} = 0$$

$$\Rightarrow \boxed{\frac{de_1}{dt} = \frac{1}{C}\text{diode}(-e_1; I_S, V_t) - \frac{I(t)}{C}}$$

J. Roychowdhury, University of California at Berkeley

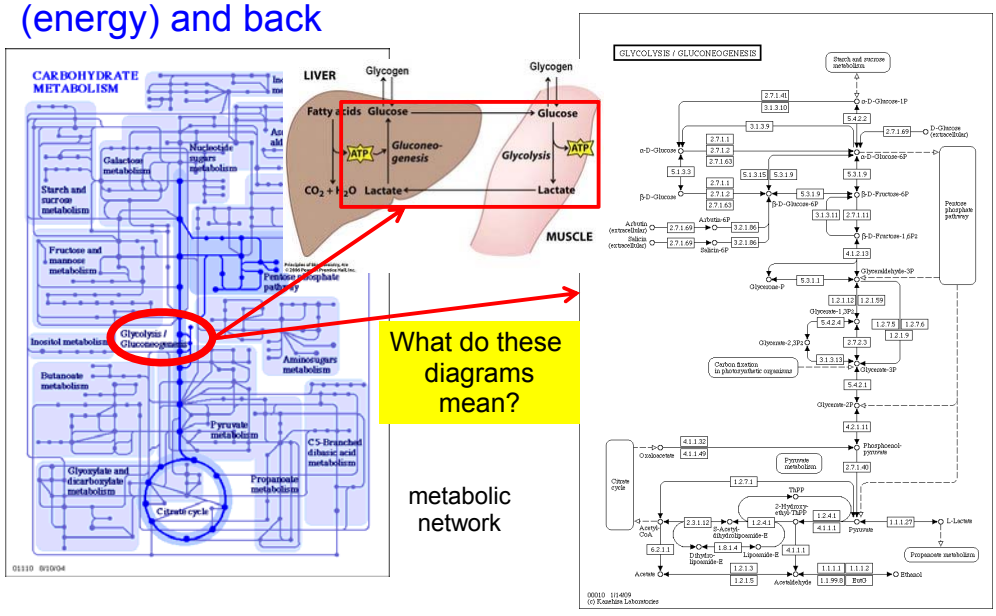
Slide 9

Control systems: plant + controller



Biochemical Systems

Reactions governing conversion of glucose to ATP (energy) and back



Pathway: Chain of Reactions

Enzyme (protein)

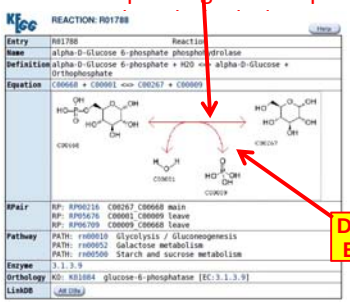
- glucose-6-phosphatase
  - catalyzes reaction

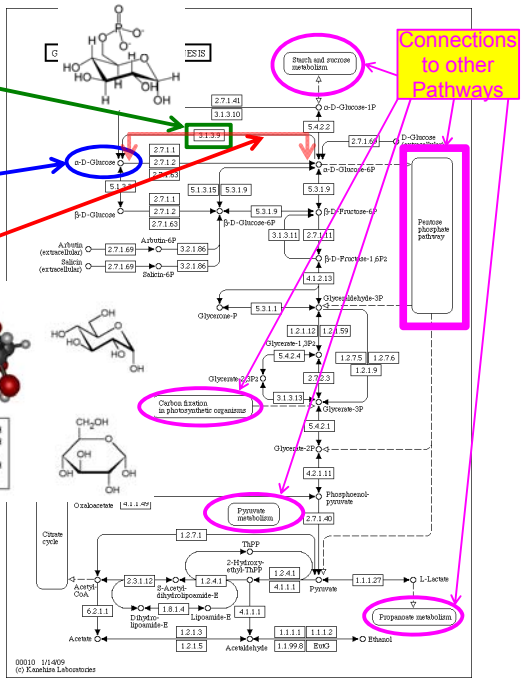
Compound

- alpha-D-Glucose

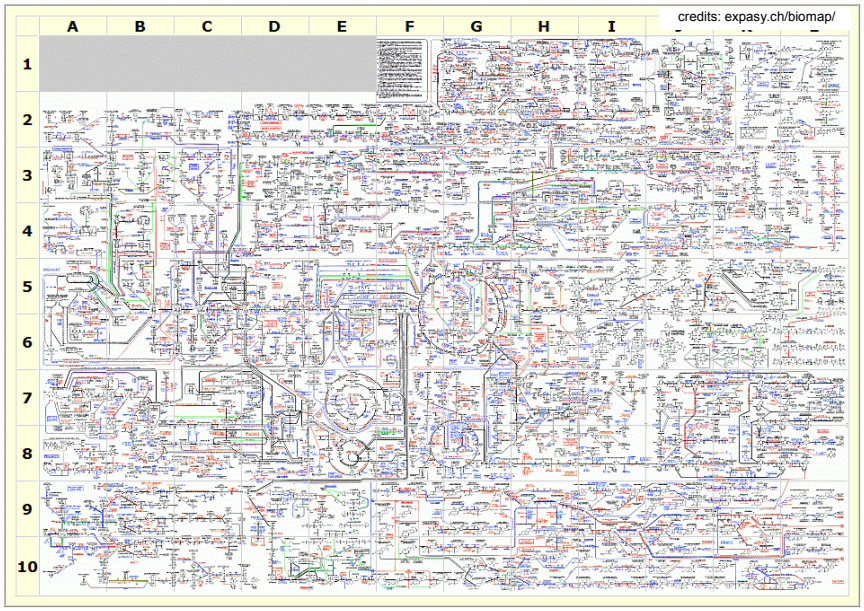
Reaction

- alpha-D-glucose 6-phos





Metabolic Pathway Maps



Course Logistics

Webpage, Books, etc.

The course webpage is the definitive source of information

http://embedded.eecs.berkeley.edu/eecsx44/

We'll also use **bCourses** (not bSpace)

No textbook. Readings will be posted / handed out for each set of lectures.

Some references will be placed on reserve in Engineering library.

GSI: Chris Shaver

See webpage for office hours, etc.

Format of Lectures

2 1.5 hour lectures per week (Mon-Wed 2:30 – 4 pm)

1 hr Discussion section / lecture each Mon 4-5 pm

- usually a topic supporting homeworks/projects; sometimes an extension of lectures

Grading
(from last year, tentative for this year)

2 Midterms (20% each)

8-10 homework assignments (total ~ 25%)

1 course project (~ 35%)

Course project:

❑ Graduate students (244): Must investigate a novel research idea

❑ Undergraduates (144): Encouraged to do 244-style project (join with grads!); also permitted to do implementation projects or literature surveys

SYSTEMS

57

What is a “system” ?

58

System: definition

- Something that has:
  - State
  - Dynamics: rules that govern the evolution of the state in time

59

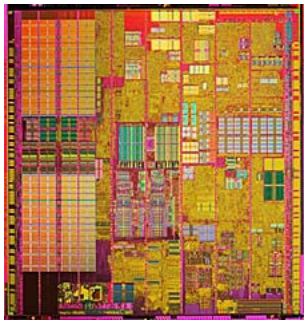
System: definition

- Something that has:
  - **State**
  - Dynamics: rules that govern the evolution of the state in **time**
- It may also have:
  - Inputs: they influence how system evolves
  - Outputs: this is what we observe

60

Example: digital circuits

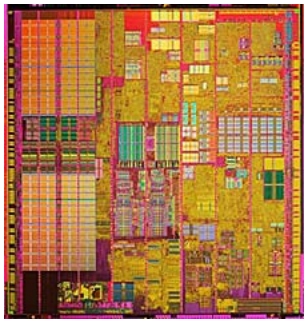
- Digital circuit:
  - State: ???
  - Dynamics: ???



61

Example: digital circuits

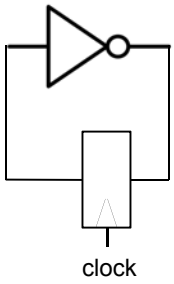
- Digital circuit:
  - State: value of every register, memory element
  - Dynamics:
    - Defined by the combinational part (logical gates)
    - Time: discrete, or “logical” (ticks of the clock)



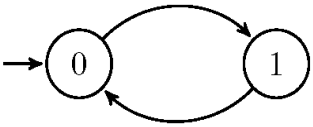
62

Example: digital circuits

- **Systems** vs. **models**



System
(the "real" circuit)



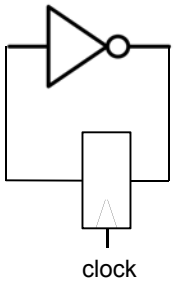
Model
(a finite-state machine)

To reason about systems (analyze, make predictions, prove things, ...), we need mathematical models

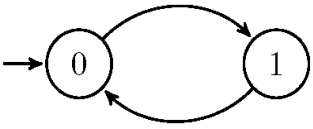
63

Example: digital circuits

- **Systems** vs. **models**



System
(the "real" circuit)




```
node Circuit ()
returns (Output: bool);
let
  Output = false -> not pre Output;
tel
```

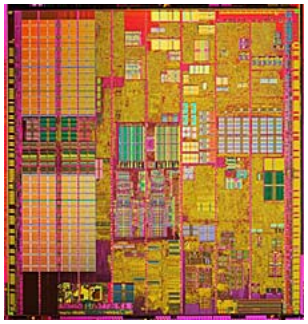
Different models
(finite-state machines)

Different representations (languages, syntaxes) of the same underlying mathematical model

64

Example: digital circuits

- Digital circuit as a system:
  - State: value of every register, memory element
  - Dynamics:
    - Defined by the combinational part (logical gates)
    - Time: discrete, or “logical” (ticks of the clock)
  - **Or**:
  - State: all currents and voltages at all transistors at a given time t
  - Dynamics: physics of electronic circuits (differential algebraic equations)



Different levels of **abstraction**

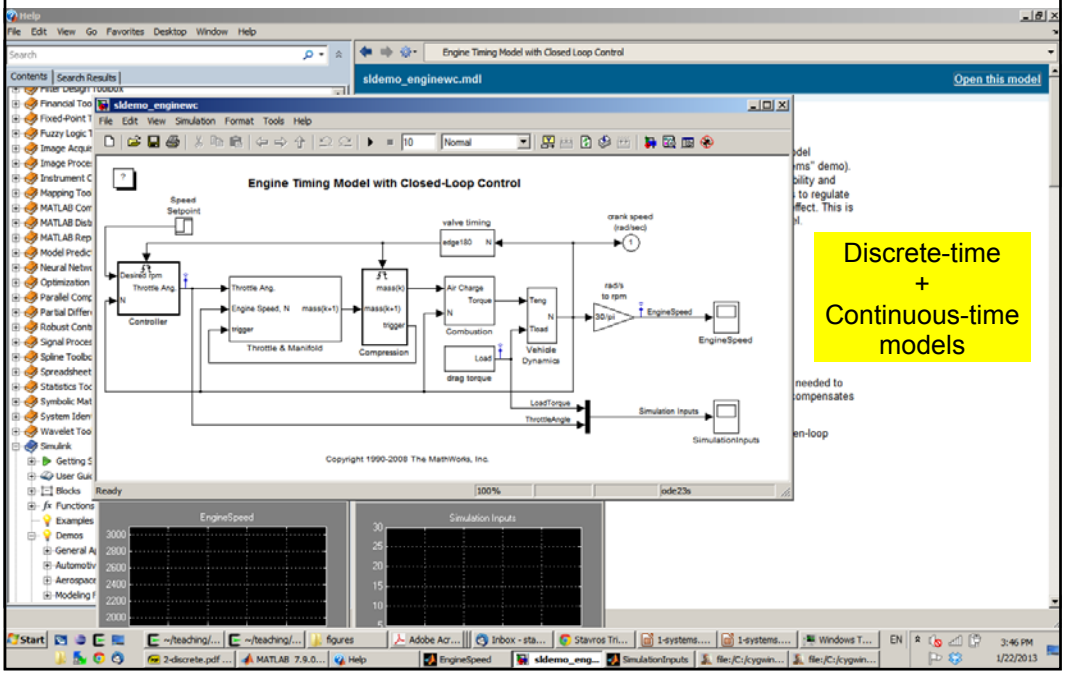
65

Multi-paradigm modeling

- Different representations (languages, syntaxes) of the same underlying formalism.
- Different modeling formalisms often needed to describe the same system, e.g., at different levels of abstraction.
- Different modeling formalisms often needed to describe different parts of the system (subsystems).

66

Example: plant + controller



Classes of systems/models surveyed in this course

- Discrete: state machines, transition systems, …
- Dataflow: process networks, SDF, …
- Timed: discrete-event systems, *timed automata*, …
- Continuous: differential equations, …
- *Probabilistic: Markov chains*, …

Back to the definition of “system”

- Many kinds of systems:
  - Software = many classes, objects, threads, …
  - Car = chassis + engine + computer + software + ...
  - Human body = heart + lungs + … = many cells = ...
  - Weather
  - Stock market
  - Internet
- How to describe each of these as states + dynamics?

Difficult (impossible) to describe some systems using our current definition

69

System: monolithic definition

- Something that has:
  - State
  - Dynamics: rules that govern the evolution of the state in time
- It may also have:
  - Inputs: they influence how system evolves
  - Outputs: this is what we observe

70

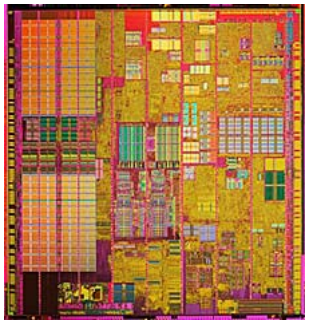
System: compositional definition

- A collection of subsystems that interact
- So, we must describe:
  - Each subsystem (recursive definition)
  - The interaction (or **composition**) rules

71

Example: digital circuits

- Subsystems: latches, gates, …
- All governed by the same clock
- Synchronous interaction



72

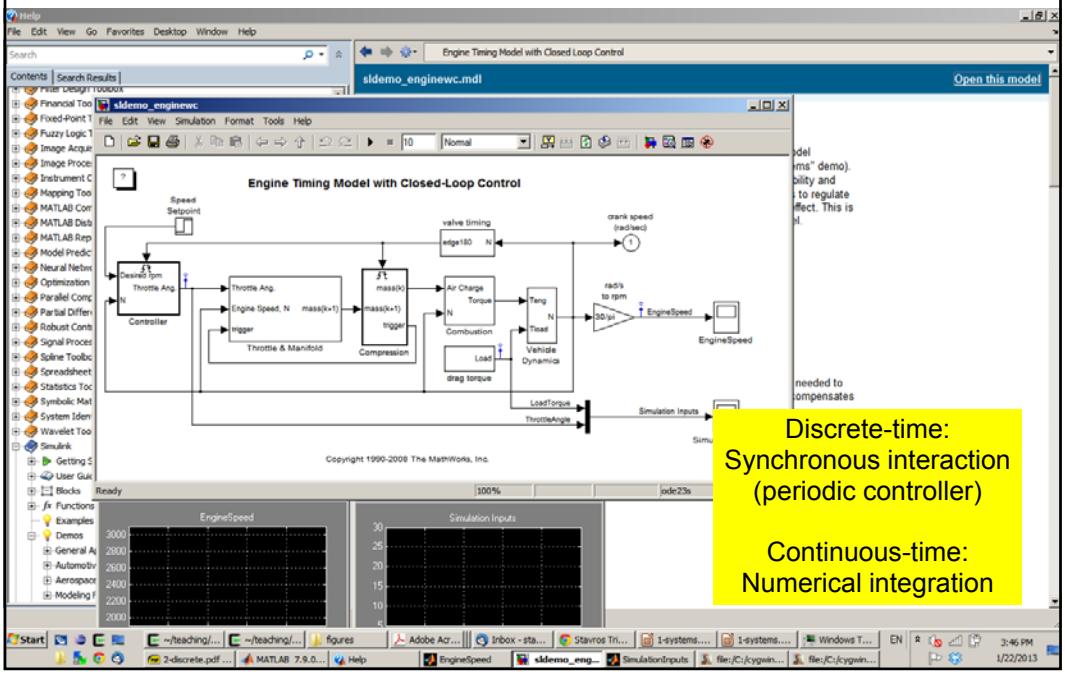
Example: concurrent software

- Multi-thread Java program:

Asynchronous interaction (*interleaving*)

```
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void bow(Friend bower) {
            System.out.format("%s: %s"
                + "  has bowed to me!%n",
                this.name, bower.getName());
            bower.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: %s"
                + " has bowed back to me!%n",
                this.name, bower.getName());
        }
    }

    public static void main(String[] args) {
        final Friend alphonse =
            new Friend("Alphonse");
        final Friend gaston =
            new Friend("Gaston");
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

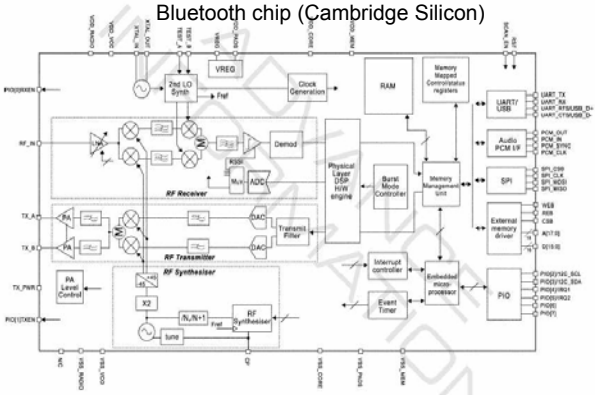
Example: plant + controller



Example: analog/mixed-signal circuits

- Subsystems: ADC, DAC, microprocessor, …, wires
- Interaction rules (partial list):
  - Kirchoff's laws:
    - At every node of the circuit: sum(all currents) = 0

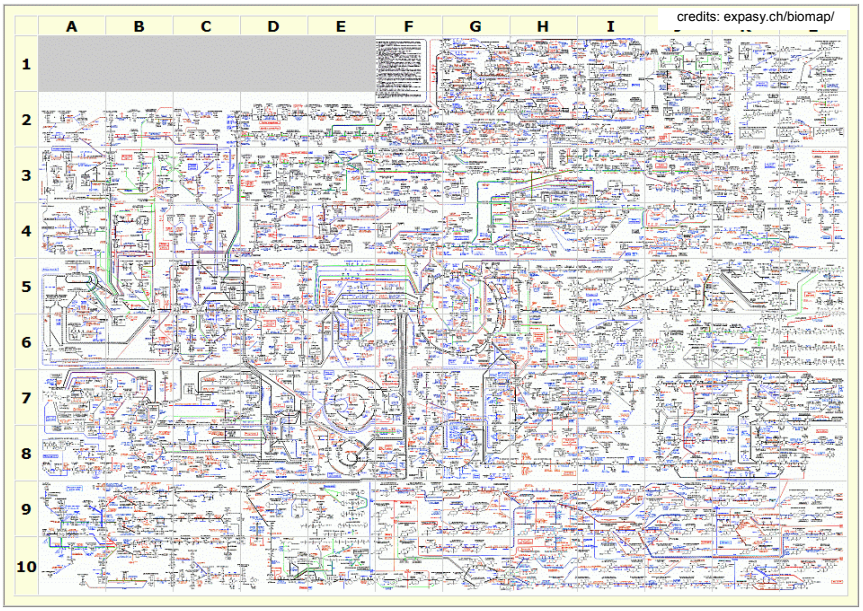
Bluetooth chip (Cambridge Silicon)



Courtesy of J. Roychowdhury, UC Berkeley

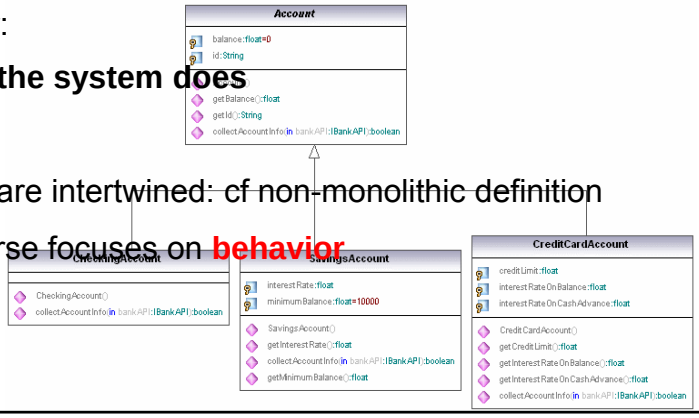
75

Biochemical Systems



Systems: structure + behavior

- Structure:
  - **What the system is made of**, its parts, sub-systems, …
  - Some modeling languages focus on structure: e.g., UML class diagrams
- Behavior:
  - **What the system does**
- The two are intertwined: cf non-monolithic definition
- This course focuses on **behavior**



77