Languages for Embedded System Design

COMS W4995-02

Prof. Stephen A. Edwards
Fall 2002
Columbia University
Department of Computer Science

What are Embedded Systems?

Computers masquerading as non-computers.

Casio Camera Watch

Nokia 7110 Browser Phone

Sony Playstation 2

Philips DVD Player

Philips TiVo Recorder

Embedded System Challenges

Differs from general-purpose computing:

Real-time Constraints

Power Constraints

Exotic Hardware

Concurrency

Control-dominated systems

Signal-processing

User Interfaces

Laws of Physics

The Role of Languages

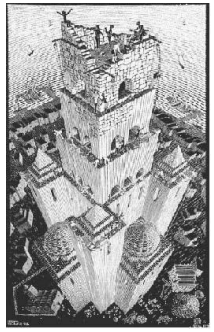
Language shapes how you solve a problem.

Java, C, C++ and their ilk designed for general-purpose systems programming.

Do not address timing, concurrency.

Domain-specific languages much more concise.

Problem must fit the language.



Syllabus

Software languages: Assembly, C, and C++

Concurrency in Java and Real-Time Operating Systems

Dataflow Languages (SDF)

Hardware Languages (Verilog)

Esterel and SystemC

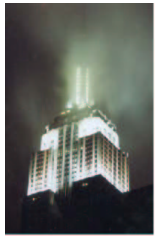
Goal of the Class

Breadth: Lectures and Homework

- Languages embody methodologies
- Knowledge of many languages
- More languages, bigger bag of tricks

Depth: Project

- Learn one language in detail
- Learn how to write a scholarly paper
- Something to brag about to future employers/schools



How to Listen to a Lecture

Ask Questions!

Hint: Presenters do a better job when they think someone is listening

I'm from Berkeley. There, every VW bus sports a bumper sticker saying

QUESTION AUTHORITY

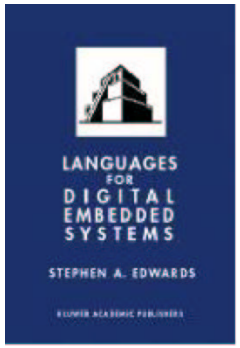
"And should there be a sudden loss of consciousness during this meeting, oxygen masks will drop from the ceiling."

Required Text

Languages for Digital Embedded Systems

Available at Papyrus, 114th and Broadway

(Textbooks are in the basement)



Class Website

www.cs.columbia.edu/ sedwards/classes/2002/w4995-02/

Everything will be there. Please check regularly.

Class Structure

Five Homework Assignments

- Collaboration permitted, but work must be your own

Two In-class exams

- First covers first half of class
- Second covers second

One Big Project

- Project proposal in two weeks
- Literature review
- Presentation of lit. review
- Final project write-up
- Presentation of final project

The Project

Goal is to produce a workshop-caliber paper, but you don't have to submit it. Final writeup will consist of

- Introduction
- Literature Survey
- Technical Details
- Experimental Results
- Conclusions

Literature survey due at midterm time

Project Ideas

"Use the languages"

- Compare Verilog and System C simulation performance
- Compare the performance of Linux and an RTOS
- Model some sort of controller in different langauges

"Analyze or implement"

- Verilog Hierarchy Browser
- Implement Kahn Process Networks
- Simple Java-to-C translator
- Simulator for Esterel

Goal of the Project

Lecture and homework can't go into depth with any one language

I want to give you a more intimate experience with at least one of the languages

You'll have to explore the literature and do independent research

Lecture and homework more theoretical; project will apply that theory

I'm hoping to promote your research

I'm hoping to promote my research

Project Schedule

One-paragraph project proposal due September 25

Literature survey presentations October 30

Literature survey due October 30

Project presentation due during finals

Project writeup due December 10

Multiple teams may choose the same project

Project Ideas

Hierarchy browser for the Verilog language

- Verilog models hardware
- Systems contain modules with instances of others
- Project: Use an existing parser, extract connectivity information, display it attractively
- Prerequisites: Understand the Verilog language (use the text), understand a freely-available parser

Compiled event-driven simulator for Esterel

- Divide behavior into events, schedule them in a queue
- Project: Apply this to (part of) Esterel
- Prerequisites: Understand Esterel, understand my compiler for it

Project Ideas

Compare Verilog and SystemC

- Both are able to model hardware, but which is better
- Project: Pick some example (e.g., a processor cache controller) and implement it in both
- Prerequisites: Some hardware design knowledge, understand both languages

An Environment for Kahn Process Networks

- Kahn proposed a C-like language with send and receive statements
- Project: Create a library that allows you to run these systems (Java? C?) Main challenge: scheduler.
- Prerequisites: Understand the Kahn model, decent

Project Ideas

Develop a simulator for an assembly language

- Compare the many approaches (interpreter-based, object-to-object translation); pick one to implement
- Project: Build the simulator in C or Java
- Prerequisites: Intimate knowledge of one assembly language

A survey of language concurrency models

- Compare how Java, POSIX threads, Ada, Verilog, etc. handle concurrency
- Prerequisites: understand the concurrency models of each of these languages. Read some language reference manuals

Project Ideas

Create a simplified Verilog simulator

- Take an existing front-end and create a new back-end
- Try to make it faster
- Prerequisites: some knowledge of digital design, indepth understanding of Verilog (read text, papers)

Compare performance of Linux and an RTOS

- Figuring out how to measure this is the challenge
- Read some of the OS literature to figure this out
- Prerequisites: detailed OS knowledge

Project Ideas

Propose a language for device drivers

- Start from French group s work on video drivers
- Look for patterns in existing, handwritten drivers
- Propose a simple language capturing these patterns
- Write a simple compiler (perhaps using m4)

Propose a language for communication protocols

- Use some of the others as starting points
- Discuss their advantages and disadvantages
- Propose extensions, simplifications, others
- Consider different compilation techniques

Project Ideas

Software Estimation

- Read up on the software performance estimation literature
- Either use some existing tools or develop a new one
- Compare different approaches. How accurate are they?

Esterel compiler for small-footprint programs

- Automotive devices need to use minimal memory
- Project: Devise a way (probably an interpreter) that can produce very small Esterel programs
- Prerequisites: Understand Esterel, understand my compiler for it

Example Project

Implementing Process Networks in Java Arnab Basu and Hampapur P. Vijay Kishen, 2000

Done at UT Austin in Brian Evans class

Used Parks scheduling algorithm to resolve deadlocks

Writeup (from Brian s class site): Survey of different process networks
Description of other, similar projects
Description of their implementation
Experiments compare various scheduling policies

Project Proposal

One-paragraph description of what you plan to do

Due soon: September 25th

Use the class website for more ideas.

Feel free to imitate (but not copy) projects you find elsewhere.

Visit during office hours to discuss ideas, or send me email.

Collaboration

You may collaborate on homework, but whatever you turn in must be your own.

Project teams should be two or three people.

Late Policy

No credit for late assignments without prior permission.

Homework is due at the beginning of class.

Syntax, Semantics, and Model

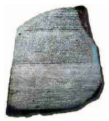
Marionette Model

You have control through the syntax of the language

The semantics of the language connect the syntax to the model

You ultimately affect a model

Syntax



Formally:

Language: infinite set of strings from an alphabet

Language	Alphabet
DNA	A T G C
Student Transcripts	w1007-02 w1009-01 w4995-02
English	aardvard abacus abalone . . .
Verilog	always module . . .

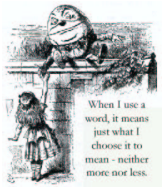
Computation Model

What the string ultimately affects

A language may have more than one

Language	Model
DNA	Proteins suspended in water
Student Transcripts	Your knowledge
	The admiration of others
English	Natural Language Understanding
Verilog	Discrete Event Simulator
	Netlist of gates and flip-flops

Semantics



How to interpret strings in the model

Also not necessarily unique

Language	Semantics
DNA	⟦AGA ⟧= Arginine
	⟦TAG ⟧= STOP
Student Transcripts	⟦w1007-02 ⟧= Java
English	⟦Look out! ⟧= Somebody's warning me
Verilog	⟦always @posedge clk ⟧= Flip-flop

Defining Syntax

Generally done with a grammar

Recursively-defined rules for constructing valid sentences

"Backus-Naur Form"

```
expr ::
       literal
   || expr + expr
   || expr * expr
```

Not a focus of this class: I'm assuming you've had a compilers class.

Operational Semantics

Describes the effect a program has on an abstract machine

Typical instruction observes and then advances machine state

Close to implementation, fairly easy to use to create the "obvious" implementation

Often includes too many details, can be hard to show that a particular implementation conforms

Denotational Semantics

Describes a program as a function that transforms input to a result

Shows how to construct this function by composing the function of each program statement.

Much more elegant handling of recursion and self-reference

Highly mathematical, most people find it cryptic

Formal program semantics could easily be a semster-long class. Not the focus of this one.

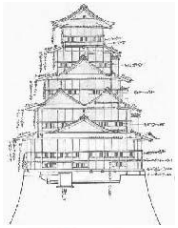
Specification and Modeling

How do you want to use the program?

Specification langauges say "build this please."

Modeling languages allow you to describe something that does or will exist

Distinction a function of the model and the language's semantics



Specification Versus Modeling

C is a specification language

- Semantics very operational
- Clear how the language is to be translated into assembly language

Verilog is a modeling language

- Semantics suggestive of a simulation procedure
- Good for building a model that captures digital hardware behavior (delays, race conditions, unknown values)
- Not as good for specification: how do you build something with a specific delay?

Concurrency


Photo by Thomas Donoghue

Why bother?

Harder model to program

Real world is concurrent

Good architecture: one concurrently-running process controls each independent system component

E.g., process for the right brake, process for the left brake, process for a brake pedal

Approaches to Concurrency

Shared memory / Every man for himself

- Adopted by Java, other software languages
- Everything's shared, nothing synchronized by default
- Synchronization through locks/monitors/semaphores
- Most flexible
- Easy to get wrong

Synchronous

- Global clock regulates passage of time
- Very robust in the presence of timing uncertainty
- Proven very successful for hardware design
- Synchronization overhead often onerous

Communication and Concurrency

Idea: Let processes run asynchronously
Only force them to synchronize when they communicate

C. A. R. Hoare's Communicating Sequential Processes

- Rendezvous-style communication:
- Processes that wish to communicate both wait until the other is ready to send/receive

Kahn Process Networks (later in the course)

- Communicate through channels
- Writer always continues
- Reader waits until data has arrived

Nondeterminism

Does a program mean exactly one thing?

Example from C:

```
a = 0;
printf("%d %d %d", ++a, ++a, ++a);
```

Argument evaluation order is undefined

Program behavior subject to the whim of the compiler

Are you sure your program does what you think?

Nondeterministic is not Random

Deterministic:	1 + 1 = 2 *always*
Random:	1 + 1 = 2 *50% of the time*, 3 *otherwise*
Nondeterministic:	1 + 1 = 2 or 3, but I'm not telling

Nondeterministic behavior can look deterministic, random, or something worse.

Murphy's law of nondeterminism: Something nondeterministic will choose the worst possible outcome at the worst possible time.

Nondeterminism is Awful

Much harder to be sure your specification or model is correct

True nondeterminstic language difficult to simulate

Should produce "any of these results"

Must maintain all possible outcomes, which grows exponentially

Idiosyncrasies of a particular implementation of a nondeterministic language often become the de facto standard

Example from Verilog

Concurrent procedure execution order undefined

```
always @(posedge clk) $write( a )
always @(posedge clk) $write( b )
```

First simulator moved procedures between two push-down stacks, producing

a b b a a b b a a b b a a b a

Later simulators had to match this now-expected behavior.

Nondeterminism is Great

True nondeterministic specification often exponentially smaller than deterministic counterpart

Implicit "all possible states" representation

E.g., nondeterministic finite automata for matching regular expressions

If system itself is truly nondeterministic, shouldn't its model also be?

Can be used to expose design errors

More flexible: only there if you want to use it

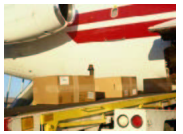
Correctness remains more elusive

Communication

Memory

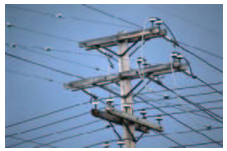
- Value written to location
- Value stays until written again
- Value can be read zero or more times after write
- No synchronization

FIFO Buffer



- Value written to buffer
- Value held until read
- Values read back in order they were written

Communication



Wires

- May or may not have explicit write operation
- Value immediately seen by all readers
- More like a system of equations than a sequence of operations

Hierarchy

Most languages have ability to create pieces and assemble them

Advantage: Information hiding

- User does not know details of a piece
- Easier to change implementation of piece without breaking whole system
- Easier to get small piece right
- Facilitates abstraction: easier to understand the whole

Advantage: Reuse

- Pieces less application-specific; can be used elsewhere

E.g., Functions in C, Classes in Java, Modules in Verilog