

### Introducing Embedded Systems: A Cyber- Physical Systems Approach

#### Edward A. Lee

Robert S. Pepper Distinguished Professor UC Berkeley

CPS PI Meeting Education Keynote

National Harbor, Maryland October 5, 2012 *With special thanks to my collaborators:* 

- Jeff Jensen, National Instruments
- Sanjit Seshia, UC Berkeley

### Background: Five Years of Experience with "Introduction to Embedded Systems"

This course is intended to introduce students to the design and analysis of computational systems that interact with physical processes.

A major theme of this course will be on the interplay of practical design with formal models of systems, including both software components and physical dynamics. A major emphasis will be on building high confidence systems with real-time and concurrent behaviors.

The course has recently been extended to become a mezzanine-level course, aimed at advanced undergraduates and beginning graduate students. Dovetails with a new professional masters program with a focus on robotics and embedded systems.

- Cyber-Physical Systems
- Model-Based Design
- Sensors and Actuators
- Interfacing to Sensors and Actuators
- Actors, Dataflow
- Modeling Modal Behavior
- Concurrency: Threads and Interrupts
- Hybrid Systems
- Simulation
- Specification; Temporal Logic
- Reachability Analysis
- Controller Synthesis
- Control Design for FSMs and ODEs
- Real-Time Operating Systems (RTOS)
- Scheduling: Rate-Monotonic and EDF
- Concurrency Models
- Execution Time Analysis
- Localization and Mapping
- Real-Time Networking
- Distributed Embedded Systems

# Approach: Interplay of Modeling, Design, and Analysis





This Talk: Focus on *Design* (and specifically, design of *software*)

#### • Traditional design of embedded systems:

Embedded software is software on small computers. The technical problem is one of optimization (coping with limited resources and extracting performance).

#### • CPS-based design of embedded systems:

Computation and networking integrated with physical processes. The technical problem is managing dynamics, time, and concurrency in networked computational + physical systems.

### Our Approach: Emphasis on Critical Thinking

"Our view is that the field of cyber-physical systems is very young, and it would not serve our students well to leave them with the illusion that completing the course equates to mastery of the subject."

[Lee, Seshia, Jensen, WESE 2012]

We teach them to *think critically* about today's technology, not just to master it.

## An Introductory Lab in Embedded and Cyber-Physical Systems

Jeff Cameron Jensen Edward Ashford Lee Sanjit Arunkumar Seshia Design Lab: Structure

- 6 weeks of structured labs introducing students to some of the tools of the trade.
- 9 weeks of group projects.

Draft lab manual available...

### The Tools of the Trade

#### **Model-Based Design**

Concepts:

Concurrent models of computation, code generation, determinism, ...

#### **Real-Time Operating Systems**

Concepts:

Scheduling, priorities, mutual exclusion, nondeterminism, ...

#### **Bare-Iron Programming**

*Concepts:* Interrupts, polling, memory models, timing, ...

In the first six weeks, students get experience with three levels of abstraction in embedded software design.

# The Hardware Platform for the First Six Weeks

Modified iRobot Create with wireless networking, many built-in sensors, and a three-axis accelerometer.



## Computational Platform: Single-Board Rio (National Instruments)

This board provides all three layers of abstraction:

- Bare iron C programming on a Xilinx Microblaze soft core.
- RTOS C programming on a PowerPC running VxWorks.
- LabVIEW modelbased design with code generation.



Lee, Berkeley 10

### The "Harry Potter" Approach to Embedded Software Design (Bare Iron Level)

#### On an Atmega 168 (a popular 8-bit microcontroller):

```
// Set timer1 to generate an interrupt every 1ms
TCCR1A = 0x00;
TCCR1B = (_BV(WGM12) | _BV(CS12));
CCR1A = 71;
Expelliatmega!
```

## Learn the right spells, and express them with conviction...



# The Emphasis on *Critical Thinking* (Bare Iron Level)

TCCR1B = (BV(WGM12) | BV(CS12));

#define \_MMIO\_BYTE(mem\_addr)(\*(volatile uint8\_t \*)(mem\_addr))
#define \_SFR\_MEM8(mem\_addr) \_MMIO\_BYTE(mem\_addr)
#define \_BV(bit) (1 << (bit))
#define TCCR1B \_SFR\_MEM8 (0x81)
#define WGM12 3
#define CS12 2
Cpreprocessor</pre>

(\*(volatile uint8 t \*)(0x81)) = (1 << 3) | (1 << 2);

Although TCCR1B appears to be a C variable, it is not (and cannot be, since C provides no way to force a variable to reside at a particular memory address). Evidently, C is not a perfect match for the problem at hand!

Lee, Berkeley 12

Hunt for header files used by the compiler

# The Emphasis on *Critical Thinking* (RTOS Level)



### A Scenario

Under Integrated Modular Avionics, software in the aircraft engine continually runs diagnostics and publishes diagnostic data on the local network.



Proper software engineering practice suggests using the observer pattern.



Design Patterns

Elements of Reusable Object-Oriented Software ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Erich Gamma Richard Helm Ralph Johnson John Vlissides



Foreword by Grady Booch

An observer process updates the cockpit display based on notifications from the engine diagnostics.

```
#include <pthread.h>
```

```
int value;
pthread_mutex_t lock;
```

```
void addListener(notify listener) {
    pthread_mutex_lock(&lock);
    ... add the listener to the list ...
    pthread_mutex_unlock(&lock);
}
```

```
void update(int newValue) {
   pthread_mutex_lock(&lock);
   value = newValue;
   ... copy the list of listeners ...
   pthread_mutex_unlock(&lock);
   ... notify the listeners on the copy ...
}
int main(void) {
   pthread_mutex_init(&lock, NULL);
```

```
... start diagnostic & observer threads.
```

## Threads: the Prevailing Concurrency Model

A carefully constructed "thread safe" multitasking solution.

It turns out it carries risk of lurking errors...

If multiple threads call update(), the updates will occur in some order. But there is no assurance that the listeners will be notified in the same order. Listeners may be mislead about the "final" value.

#### **Recall the Scenario**

Under Integrated Modular Avionics, software in the aircraft engine continually runs diagnostics and publishes diagnostic data on the local network.



Proper software engineering practice suggests using the observer pattern.



Design Patterns

Elements of Reusable Object-Oriented Software ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Erich Gamma Richard Helm Ralph Johnson John Vlissides



Foreword by Grady Booch

An observer process updates the cockpit display based on notifications from the engine diagnostics.

# The Emphasis on *Critical Thinking* (Model-Based Design Level)



## Model-Based Design

Emphasis on concurrency and timing.

Lab experience with LabVIEW, classroom discussion of other model-based design formalisms.





#### Control loop (40 Hz)



## **Capstone Projects**

- Cegway-like two-wheel robot
- Distributed Pacman
- Cooperative self-parking vehicles
- Face-tracking quadrotor
- Robotic convoys
- Elevator operator
- Automatic xylophone
- Dataglove gesture replicator
- Gesture-driven robot steering
- Mapping and localization
- Robotic summo wrestling

#### **Capstone Projects**



#### The Canon

Teachers should teach what they know.

Actually, the most valuable teachers are the ones who teach what is not known...

#### Conclusion

Our job isn't to get our students to replicate us.

Our job is to get our students to *replace* us.

If we succeed, our students will make us obsolete.

If we fail, their students will make us obsolete.

#### **Backup Slides**

### Applications First? Or Foundations First?





#### Bottom-up:

- foundations first

- derive the applications

*Top-down:* - applications first - derive the foundations

## Class Projects are Defined the Students. Example:



One of the five project teams in 2008 developed a balancing robot inspired by the Segway. They used a Nintendo Wiimote as a controller communicating with a PC running LabVIEW, communicating with a Lego Mindstorm NXT, which they programmed in C.



Modeling Physical Dynamics

- Orientation:  $\theta \colon \mathbb{R} \to \mathbb{R}^3$
- Angular velocity:  $\dot{\theta} \colon \mathbb{R} \to \mathbb{R}^3$
- Angular acceleration:  $\ddot{\theta} \colon \mathbb{R} \to \mathbb{R}^3$
- Torque:  $\mathbf{T} \colon \mathbb{R} \to \mathbb{R}^3$

$$\theta(t) = \begin{bmatrix} \dot{\theta}_x(t) \\ \dot{\theta}_y(t) \\ \dot{\theta}_z(t) \end{bmatrix} = \begin{bmatrix} \text{roll} \\ \text{yaw} \\ \text{pitch} \end{bmatrix}$$



#### Actor Model of Systems

A system is a function that accepts an input signal and yields an output signal.

The domain and range of the system function are sets of signals, which themselves are functions.

Parameters may affect the definition of the function *S*.



#### State Machines and Modal Models

