The Challenges of Embedded System Design

Edward A. Lee
Robert S. Pepper Distinguished Professor
UC Berkeley

Invited Talk

Xilinx Emerging Technology Symposium (ETS)
San Jose, CA
February 1, 2012

Key Collaborators on work shown here:

• Steven Edwards
• Jeff Jensen
• Sungjun Kim
• Isaac Liu
• Slobodan Matic
• Hiren Patel
• Jan Reinke
• Sanjit Seshia
• Mike Zimmer
• Jia Zou
Abstract

All widely used software abstractions lack temporal semantics. The notion of correct execution of a program written in every widely-used programming language today does not depend on the temporal behavior of the program. But temporal behavior matters in almost all systems, particularly in networked systems. Even in systems with no particular real-time requirements, timing of programs is relevant to the value delivered by programs, and in the case of concurrent and distributed programs, also affects the functionality. In systems with real-time requirements, including most embedded systems, temporal behavior affects not just the value delivered by a system but also its correctness.

This talk will argue that time can and must become part of the semantics of programs for a large class of applications. It will argue that temporal behavior is not always just a performance metric, but is often rather a correctness criterion. To illustrate that this is both practical and useful, we will describe recent efforts at Berkeley in the design and analysis of timing-centric software systems. In particular, we will focus on two projects, PRET, which seeks to provide computing platforms with repeatable timing, and PTIDES, which provides a programming model for distributed real-time systems.
Cyber-Physical Systems (CPS): Orchestrating networked computational resources with physical systems

Power generation and distribution

Military systems:

Transportation (Air traffic control at SFO)

Avionics

Telecommunications

Factory automation

Instrumentation (Soleil Synchrotron)

Daimler-Chrysler

E-Corner, Siemens

Courtesy of Kuka Robotics Corp.

Courtesy of General Electric

Courtesy of Doug Schmidt

Lee, Berkeley
Claim

For CPS, \textit{programs} do not adequately specify \textit{behavior}. 
A Story

The Boeing 777 was Boeing’s first fly-by-wire aircraft, controlled by software. It is deployed, appears to be reliable, and is succeeding in the marketplace. Therefore, it must be a success. However...

Boeing was forced to purchase and store an advance supply of the microprocessors that will run the software, sufficient to last for the estimated 50 year production run of the aircraft and another many years of maintenance.

Why?
Lesson from this example:

Apparently, the software does not specify the behavior that has been validated and certified!

Unfortunately, this problem is very common, even with less safety-critical, certification-intensive applications. Validation is done on complete system implementations, not on software.
Structure of a Cyber-Physical System

Problems that complicate analysis of system behavior:

- Sensors may be locked out for an indeterminate amount of time
- Platforms’ measurements of time differ
- Messages from different sources interleave nondeterministically
- Variability of execution times affects results (not just WCET)
- A fault in a remote component may disrupt a critical local activity
- A fault in a remote component may go undetected for a long time
- Interrupt-driven I/O disrupts timing
- Etc…

Lee, Berkeley 7
A Key Challenge: Timing is not Part of Software Semantics

Correct execution of a program in C, C#, Java, Haskell, OCaml, etc. has nothing to do with how long it takes to do anything. All our computation and networking abstractions are built on this premise.

Programmers have to step outside the programming abstractions to specify timing behavior.
Execution-time analysis, by itself, does not solve the problem!

Analyzing software for timing behavior requires:

• Paths through the program (undecidable)
• Detailed model of microarchitecture
• Detailed model of the memory system
• Complete knowledge of execution context
• Many constraints on preemption/concurrency
• Lots of time and effort

And the result is valid only for that exact hardware and software!

Fundamentally, the ISA of the processor has failed to provide an adequate abstraction.

Part 1: PRET Machines

- **PREcision-Timed processors** = PRET
- **Predictable, REpeatable Timing** = PRET
- **Performance with REpeatable Timing** = PRET

```c
// Perform the convolution.
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];
    // Notify listeners.
    notify(x[i]);
}
```
Dual Approach

- Rethink the ISA
  - Timing has to be a correctness property not a performance property.

- Implementation has to allow for multiple realizations and efficient realizations of the ISA
  - Repeatable execution times
  - Repeatable memory access times
Example of one sort of mechanism we would like:

```
tryin (500ms) {
     // Code block
} catch {
    panic();
}
```

If the code block takes longer than 500ms to run, then the panic() procedure will be invoked.

But then we would like to verify that panic() is never invoked!

```
jmp_buf buf;

if ( !setjmp(buf) ){
    set_time r1, 500ms
    exception_on_expire r1, 0
    // Code block
    deactivate_exception 0
} else {
    panic();
}

exception_handler_0 () {
    longjmp(buf)
}
```

Pseudocode showing the mechanism in a mix of C and assembly.
Extending an ISA with Timing Semantics

[V1] Best effort:

```
set_time r1, 1s
// Code block
delay_until r1
```

[V2] Late miss detection

```
set_time r1, 1s
// Code block
branch_expired r1, <target>
delay_until r1
```

[V3] Immediate miss detection

```
set_time r1, 1s
exception_on_expire r1, 1
// Code block
deactivate_exception 1
delay_until r1
```

[V4] Exact execution:

```
set_time r1, 1s
// Code block
MTFD r1
```
To provide timing guarantees, we need implementations that deliver repeatable timing.

Fortunately, electronics technology delivers highly reliable and precise timing…

…but the overlaying software abstractions discard it. Chip architects heavily exploit the lack of temporal semantics.

```c
// Perform the convolution.
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];  // Notify listeners.
    notify(x[i]);
}
```
To deliver repeatable timing, we have to rethink the microarchitecture

Challenges:

- Pipelining
- Memory hierarchy
- I/O (DMA, interrupts)
- Power management (clock and voltage scaling)
- On-chip communication
- Resource sharing (e.g. in multicore)
Our Current PRET Architecture

*PTArm*, a soft core on a Xilinx Virtex 5 FPGA

Interleaved pipeline with one set of registers per thread

Hardware thread

Registers

Scratch pad

SRAM scratchpad shared among threads

Memory

DRAM main memory, separate banks per thread

I/O devices

Note inverted memory compared to multicore!

Fast, close memory is shared, slow remote memory is private!
Multicore PRET

In today’s multicore architectures, one thread can disrupt the timing of another thread even if they are running on different cores and are not communicating!

Our preliminary work shows that control over timing enables conflict-free routing of messages in a network on chip, making it possible to have non-interfering programs on a multicore PRET.
Status of the PRET project

- Results:
  - PTArm implemented on Xilinx Virtex 5 FPGA.
  - UNISIM simulator of the PTArm facilitates experimentation.
  - DRAM controller with repeatable timing and DMA support.
  - PRET-like utilities implemented on COTS Arm.

- Much still to be done:
  - Realize MTFD, interrupt I/O, compiler toolchain, scratchpad management, etc.
A Key Next Step: Parametric PRET Architectures

ISA that admits a variety of implementations:

- Variable clock rates and energy profiles
- Variable number of cycles per instruction
- Latency of memory access varying by address
- Varying sizes of memory regions
- ...

A given program may meet deadlines on only some realizations of the same parametric PRET ISA.
Realizing the MTFD instruction on a parametric PRET machine

The goal is to make software that will run correctly on a variety of implementations of the ISA, and that correctness can be checked for each implementation.
PRET Publications

Part 2: How to get the Source Code?

The input (mostly likely C) will ideally be generated from a model, like Simulink or SCADE. The model specifies temporal behavior at a higher level than code blocks, and it specifies a concurrency model that can limit preemption points. However, Simulink and SCADE have naïve models of time.
Recall Structure of a Cyber-Physical System

Problems that complicate analysis of system behavior:

- Sensors may be locked out for an indeterminate amount of time
- Messages from different sources interleave nondeterministically
- Variability of execution times affects results (not just WCET)
- A fault in a remote component may disrupt a critical local activity
- A fault in a remote component may go undetected for a long time
- Interrupt-driven I/O disrupts timing
- Etc…
Ptides: First step: Time-stamped messages.
Ptides: Second step:
Network time synchronization

GPS, NTP, IEEE 1588, time-triggered busses, etc., all provide some form of common time base. These are becoming fairly common.
Ptides: Third step: Bind time stamps to real time at sensors and actuators

- Input time stamps are $\geq$ real time
- Output time stamps are $\leq$ real time

Clock synchronization gives global meaning to time stamps.
Global latencies between sensors and actuators become controllable, which enables analysis of system dynamics.

Ptides: Fourth step: Specify latencies in the model

Model includes manipulations of time stamps, which control latencies between sensors and actors.

Actuators may be designed to interpret input time stamps as the time at which to take action.

Feedback through the physical world
Ptides: Fifth step
Safe-to-process analysis (ensures determinacy)

Safe-to-process analysis guarantees that the generated code obeys time-stamp semantics (events are processed in time-stamp order), given some assumptions.

Assume bounded sensor delay $s$

Assume bounded network delay $d$

Assume bounded clock error $e$

Application specification of latency $d_2$

An earliest event with time stamp $t$ here can be safely merged when real time exceeds $t + s + d + e - d_2$
Ptides Schedulability Analysis
Determine whether deadlines can be met

Schedulability analysis incorporates computation times to determine whether we can guarantee that deadlines are met.

Deadline for delivery of event with time stamp \( t \) here is \( t - c_3 - d_2 \)

Assume bounded computation time \( c_1 \)

Assume bounded computation time \( c_2 \)

Assume bounded computation time \( c_3 \)
PtidyOS: A lightweight microkernel supporting Ptides semantics

PtidyOS runs on
- Arm (Luminary Micro)
- Renesas
- XMOS

Occupies about 16 kbytes of memory.

An interesting property of PtidyOS is that despite being highly concurrent, preemptive, and EDF-based, it does not require threads.

A single stack is sufficient!

The name “PtidyOS” is a bow to TinyOS, which is a similar style of runtime kernel.
Workflow Structure

- **Ptides Model**
  - **Code Generator**
  - **Software Component Library**
  - **Mixed Simulator**
  - **Plant Model**
  - **Network Model**

- **Code**
  - **PtidyOS**

- **HW Platform**
  - **Luminary Micro 8962**

- **HW in the Loop Simulator**
  - **IEEE 1588 Network time protocol**

**Ptolemy II Ptides domain**

**Ptolemy II Discrete-event, Continuous, and Wireless domains**

- **Causality Analysis**
  - **Schedulability Analysis**
  - **Program Analysis**

**Analysis**
A Typical Cyber-Physical System

Printing Press

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

Bosch-Rexroth

Goal: Orchestrated networked resources built with sound design principles on suitable abstractions
Example – Flying Paster

Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.html
Source: http://offsetpressman.blogspot.com/2011/03/how-flying-paster-works.html

Flying Paster
Printing Press – Model in Ptolemy II

This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED

Model by Patricia Derler
This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED
This design demonstrates DC motors driving a feed roller and a drive roller. The PID-based motor controllers minimize the error between the paper velocity produced by the roller and the target profile velocity produced by the Target Profile actor. The tracking error input allows one such roller to track the other to remove small differences in paper velocity.

The target profile is either a profile from 0 to maxPaperVelocity starting at time 0 and reaching the maximum value at time Interval seconds. The profile and its derivative are continuous.

SENSOR, ACTUATOR and NETWORK ACTORS STILL NEED TO BE ADDED
Determinate timing at sensors and actuators

Platform independent model of functional and timing behavior

Code Generation to multiple target platforms

Simulation

Same I/O behavior w.r.t. value and timing

e.g.: XMOS development board with 4 XCores.

e.g.: Renesas 7216 Demonstration Kit
**Determinate timing at sensors and actuators**

**Platform independent model of functional and timing behavior**

**Code Generation to multiple target platforms**

**Simulation**

- Reserve Velocity (red), Target Velocity (green) and Tracking Error
- Time in seconds
- Velocity m/s

- Contact (red), Top Dead Center (green), Cut (blue) and Arm (black)
- Time in seconds
- Events

---

**XMOS**
- Predictable timing
- Multiple cores
- No analog I/O
- No FPU
- No hardware clock

---

**Renesas**
- PHY chip for accurate timestamping of inputs, Analog I/O

---

**Same I/O behavior w.r.t. value and timing**

---

**e.g.: XMOS development board with 4 XCores.**

---

**e.g.: Renesas 7216 Demonstration Kit**

Lee, Berkeley 39
Renesas vs. XMOS: Measured I/O timing

Simulation

Renesas

XMOS

Contact (red), Top Dead Center (green), Cut (blue) and Arm (black)

Oscilloscope traces on GPIO pins
Renesas vs. XMOS:
I/O timing

Simulation

Renesas

XMOS

Oscilloscope traces on GPIO pins

topDeadCenter
armContact
tapeDetector
contact
cut
Renesas vs. XMOS: Busy vs. Idle Time

Simulation

Renesas

XMOS

Contact (red), Top Dead Center (green), Cut (blue) and Arm (black)

Oscilloscope traces on GPIO pins
Ptides Publications


Conclusions

Today, timing behavior is a property only of realizations of software systems.

Tomorrow, timing behavior will be a semantic property of programs and models.

Overview References:

Raffaello Sanzio da Urbino – The Athens School
A Test Case for PtidyOS

This device, designed by Jeff Jensen, mixes periodic, quasi-periodic, and sporadic real-time events.

Tunneling Ball Device
– sense ball
– track disk
– adjust trajectory
Tunneling Ball Device in Action
Tunneling Ball Device

**Mixed event sequences**

- **Periodic Events**
- **Quasi Periodic Events**
- **Sporadic Events**
Distributed PTIDES Relies on Network Time Synchronization with Bounded Error

This may become routine!
With this PHY, clocks on a LAN agree on the current time of day to within 8ns, far more precise than older techniques like NTP.

A question we are addressing at Berkeley: How does this change how we develop distributed CPS software?
An Extreme Example: The Large Hadron Collider

The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within about 80 psec using a combination of IEEE 1588 PTP and synchronous ethernet.