It's About Time

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

Keynote
ReConFig: Reconfigurable Computing and FPGAs

Dec. 8-10, 2014.
Cancun, Mexico

Special Thanks to:
- David Broman
- Isaac Liu
- Hiren Patel
- Jan Reineke
- Michael Zimmer
The Context for this Talk: Cyber-Physical Systems

Orchestrating networked computational resources and physical systems.

Roots of the term:
• Coined around 2006 by Helen Gill at the National Science Foundation in the US
• **Cyberspace**: attributed William Gibson, who used the term in the novel *Neuromancer*.
• **Cybernetics**: coined by Norbert Wiener in 1948, to mean the conjunction of control and communication.
In CPS, “cyber” == “software” and “physical” == “not software”. Digital hardware sits in a gray area…
The Theme of This Talk

Determinacy

or

Better Engineering through Better Models
Sources of Nondeterminism

- Physical noise
- Imperfect actuation
- Parts failures
- Unknowable delays
- Unknowable execution times
- Uncontrollable scheduling

Diagram:
- Computational Platform → Network Fabric → Computational Platform
- Physical plant
In the face of such nondeterminism, does it make sense to talk about deterministic models for cyber-physical systems?
Models vs. Reality


You will never strike oil by drilling through the map!

Solomon Wolf Golomb (1932) mathematician and engineer and a professor of electrical engineering at the University of Southern California. Best known to the general public and fans of mathematical games as the inventor of polyominoes, the inspiration for the computer game Tetris. He has specialized in problems of combinatorial analysis, number theory, coding theory and communications.
But this does not, in any way, diminish the value of a map!
The Kopetz Principle

Many (predictive) properties that we assert about systems (determinism, timeliness, reliability, safety) are in fact not properties of an implemented system, but rather properties of a model of the system.

We can make definitive statements about models, from which we can infer properties of system realizations. The validity of this inference depends on model fidelity, which is always approximate.

(paraphrased)
Deterministic Models of Nondeterministic Systems

Physical System

Model

Synchronous digital logic

Image: Wikimedia Commons
Deterministic Models of Nondeterministic Systems

Physical System

Model

```java
/*** Reset the output receivers, which are the inside receivers of
 * the output ports of the container.
 * @exception IllegalActionException if getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalActionException {
    List<IPort> outputs = (_Actor).getContainer().outputPortList();
    for (IPort output : outputs) {
        if ((_debugging) {
            _debug("Resetting inside receivers of output port: "
                + output.getName());
        }
        Receiver[][] receivers = output.getInsideReceivers();
        if (receivers != null) {
            for (int i = 0; i < receivers.length; i++) {
                if (receivers[i] != null) {
                    for (int j = 0; j < receivers[i].length; j++) {
                        if (receivers[i][j] instanceof FSMReceiver) {
                            receivers[i][j].reset();
                        }
                    }
                }
            }
        }
    }
}
```

Image: Wikimedia Commons

Single-threaded imperative programs

Lee, Berkeley
Deterministic Models of Nondeterministic Systems

Physical System

Model

\[
\dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau
\]

Differential Equations

Image: Wikimedia Commons
A Major Problem for CPS: Combinations of these Models are Nondeterministic

Image: Wikimedia Commons

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

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

Programmers have to step *outside* the programming abstractions to specify timing behavior.

Programmers have no map!

Lee, Berkeley
The Model

\[ \dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau \]
The Reality

Computational Platform → Network Fabric → Computational Platform

Physical plant

- USB interface
- JTAG and SWD interface
- graphics display
- CAN bus interface
- Ethernet interface
- analog (ADC) inputs
- removable flash memory slot
- microcontroller
- GPIO connectors
- PWM outputs
- speaker connected to GPIO or PWM
- switches connected to GPIO pins
- connected to GPIO or PWM

Image: Wikimedia Commons
The Model is not much more deterministic than the reality.

The modeling languages have disjoint, incompatible semantics.
System dynamics emerges from the physical realization

... leading to a “prototype and test” style of design
The first edition of Hennessy and Patterson (1990) revolutionized the field of computer architecture by making performance metrics the dominant criterion for design.

Today, for computers, timing is merely a performance metric.

It needs to be a correctness criterion.
Correctness criteria

We can safely assert that line 8 does not execute

(In C, we need to separately ensure that no other thread or ISR can overwrite the stack, but in more modern languages, such assurance is provided by construction.)

```c
void foo(int32_t x) {
    if (x > 1000) {
        x = 1000;
    }
    if (x > 0) {
        x = x + 1000;
        if (x < 0) {
            panic();
        }
    }
}
```

We can develop absolute confidence in the software, in that only a hardware failure is an excuse.

But not with regards to timing!!
The hardware out of which we build computers is capable of delivering “correct” computations and precise timing...

The synchronous digital logic abstraction removes the messiness of transistors.

... but the overlaying software abstractions discard the timing precision.

// Perform the convolution.  
for (int i=0; i<10; i++) {  
    x[i] = a[i]*b[j-i];  
    // Notify listeners.  
    notify(x[i]);  
}
PRET Machines – Giving Programs the Capabilities their Hardware Already Has.

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

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

http://chess.eecs.berkeley.edu/pret

Lee, Berkeley
Major Challenges
and existence proofs that they can be met

- Pipelines
  - fine-grain multithreading
- Memory hierarchy
  - memory controllers with controllable latency
- I/O
  - threaded interrupts, with bounded effects on timing
Major Challenges, Yes, but Leading to Major Opportunities

- Improved determinism
- Better testability
- Reduced energy consumption
- Reduced overdesign (cost, weight)
- Improved confidence and safety
- Substitutable hardware
Three Generations of PRET Machines at Berkeley

- PRET1, Sparc-based (simulation only)
  - [Lickly et al., CASES, 2008]

- PTARM, ARM-based (FPGA implementation)
  - [Liu et al., ICCD, 2012]

- FlexPRET, RISC-V-based (FPGA + simulation)
  - [Zimmer et al., RTAS, 2013]
Our Second Generation PRET

*PTArm*, a soft core on a Xilinx Virtex 5 FPGA (2012)

- **Hardware thread**
  - registers
  - Interleaved pipeline with one set of registers per thread

- **scratch pad**
- **memory**
  - SRAM scratchpad shared among threads
  - 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!
Performance Cost?

Not really!

In microarchitecture design, the PRET project has shown that you do not need to sacrifice performance to get control over timing.

Our Third-Generation PRET: Open-Source FlexPRET (Zimmer et al., 2014)

- 32-bit, 5-stage thread interleaved pipeline, RISC-V ISA
  - **Hard real-time HW threads:** scheduled at constant rate for isolation and predictability
  - **Soft real-time HW threads:** share all available cycles (e.g. HW thread sleeping) for efficiency
- Deployed on Xilinx FPGA (area comparable to Microblaze)

![Diagram of thread scheduling]

- Every 3 cycles (unless done)
- Clock cycles
- Instructions
- HRTT 0
- SRTT 1
- SRTT 2
- Whenever cycle available (arbitrary interleaving)

Digilent Atlys (Spartan 6) and NI myRIO (Zync)
FlexPRET

**Hard-Real-Time (HRT) Threads**

*Interleaved with Soft-Real-Time (SRT) Threads*

**HRT threads have deterministic timing.**

SRT threads share available cycles

SRAM scratchpad shared among threads

DRAM main memory provides deterministic latency for HRT threads. Conventional behavior for the rest.

Michael Zimmer
FlexPRET I/O

*Interrupt-Driven I/O is notorious for disrupting timing*

Interrupts have *no effect* on HRT threads, and *bounded effect* on SRT threads!

*Michael Zimmer*
FlexPRET Shows:

- Not only is there no performance cost for appropriate workloads, but there is also no performance cost for inappropriate workloads!

- Pipelining, memory hierarchy, and interrupt-driven I/O can all be done without loosing timing determinacy!
Software
Example of one sort of mechanism we can achieve (with some difficulty!) today:

```c
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!

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

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

Pseudocode showing how to do this today.
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
dec activates_exception 1
delay_until r1
```

[V4] Exact execution:

```
set_time r1, 1s
// Code block
MTFD r1
```
But Wait…

The whole point of an ISA is that the same program does the same thing on multiple hardware realizations.

Isn’t this incompatible with deterministic timing?
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 many implementations of the ISA, and that correctness can be checked for each implementation.
How to Make PRET Widespread? Real-Time Units (RTUs)

- Offload timing-critical functions to the RTU
  - Compare with dedicated hardware
- Software peripherals
  - Bit-banging for custom protocols
- Software API: OpenRT?
  - Richer interface for smart sensors/actuators
PRET Enables **Deterministic Interaction**
Between the Cyber and the Physical

\[
\dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau
\]
But what about the Network?

We have also developed *deterministic models* for distributed real-time software, using a technique called PTIDES.
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

See

http://chess.eecs.berkeley.edu/ptides

(or invite me back next year)
One Last Comment…
Model Fidelity

- In *science*, a good model matches well the behavior of the physical world.
- In *engineering*, a good physical implementation matches well the behavior of the model.

*In engineering, model fidelity is a two-way street!*

*For a model to be useful, it is necessary (but not sufficient) to be able to construct a faithful physical realization.*
A Model
A Physical Realization
Model Fidelity

- To a *scientist*, the model is flawed.

- To an *engineer*, the physical realization is flawed.

I’m an engineer...
Determinism?

For a model to be useful, it is necessary (but not sufficient) to be able to construct a faithful physical realization.

- The real world is highly unpredictable.
- So, are deterministic models useful?

Is synchronous digital logic useful?
Single-threaded imperative programs?
Differential equations?
Determinism?

Deterministic models do not eliminate the need for robust, fault-tolerant designs.

In fact, they *enable* such designs, because they make it much clearer what it means to have a fault!
PRET enables a programming model with \textit{deterministic} timing.

\url{http://chess.eecs.berkeley.edu/pret}
Conclusion

Today, timing behavior in computers emerges from the physical realization.

Tomorrow, timing behavior will be part of the programming abstractions and their hardware realizations.