A Personal View of Real-Time Computing

Edward A. Lee
Professor of the Graduate School

Invited Talk

Center for Information and Systems Engineering (CISE) seminar series
Boston University, Boston, MA, April 19, 2019
Predictability requires determinacy and depends on timing, including execution times and network delays.
What is Real Time?

- fast computation
- prioritized scheduling
- computation on streaming data
- bounded execution time
- temporal semantics in programs
- temporal semantics in networks

Lee, Berkeley
What is Real Time?

- fast computation
- prioritized scheduling
- computation on streaming data
- bounded execution time
- temporal semantics in programs
- temporal semantics in networks

These are very different from one another. We have to decide which to focus on.

Lee, Berkeley
Correct execution of a program in all widely used programming languages, and correct delivery of a network message in all general-purpose networks has nothing to do with how long it takes to do anything.

Programmers have to step outside the programming abstractions to specify timing behavior.
Achieving Real Time

- overengineering
- using old technology
- response-time analysis
  - real-time operating systems (RTOSs)
  - specialized networks
  - extensive testing and validation
Timing of programs emerges from the implementation

- Pipeline hazards
- Cache effects
- Variable DRAM latencies
- Speculative execution
- Interrupts
- Forwarding
- Dynamic voltage/frequency
- ...

Image from Lee & Seshia, Introduction to Embedded Systems
MIT Press, 2017
Time becomes a mess with interrupts and threads

```
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}

volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}

int main(void) {
    SysTickIntRegister(&ISR);
    // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        ... code to run for 2 seconds
    }
    ... // other code
}
```
Current Trends in Real-Time Software

- Model the details
- Analyze the models

Result is expensive, intractable models.

Even deterministic real-time models can lead to chaos.

[Thiele and Kumar, EMSOFT 2015]

Lee, Berkeley
Newtonian time advances uniformly in a continuum, observable everywhere.

\[ \dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau \]
When realized in a software-based model:

1. The precision of time should be finite and the same for all observers.

2. The precision of time should be independent of the absolute magnitude of the time.

3. Addition of time should be associative. That is, for any three time intervals \( t_1, t_2, \) and \( t_3 \),

\[
(t_1 + t_2) + t_3 = t_1 + (t_2 + t_3)
\]

Floating point numbers do not satisfy these.


Lee, Berkeley
• “Continuum” does not imply “continuous.”
Bring the cyber and the physical together and Boom

```c
void initTimer(void) {
    SysTickPeriodSet(SysCtlClockGet() / 1000);
    SysTickEnable();
    SysTickIntEnable();
}
volatile uint timer_count = 0;
void ISR(void) {
    if(timer_count != 0) {
        timer_count--;
    }
}
int main(void) {
    SysTickIntRegister(&ISR);
    // other init
    timer_count = 2000;
    initTimer();
    while(timer_count != 0) {
        // code to run for 2 seconds
    }
    // other code
}
```

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

- overengineering
- using old technology
- response-time analysis
- real-time operating systems (RTOSs)
- specialized networks
- extensive testing and validation

Maybe we can do better?

Lee, Berkeley
An Epiphany

The Creative Partnership of Humans and Technology

PLATO AND THE NERD

EDWARD ASHFORD LEE
In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.

In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

A scientist asks, “Can I make a model for this thing?”
An engineer asks, “Can I make a thing for this model?”
In this example, the **modeling framework** is calculus and Newton’s laws. 

**Fidelity** is how well the model and its target match.
A Model

Image by Dominique Toussaint, GNU Free Documentation License, Version 1.2 or later.

Lee, Berkeley
A Physical Realization

Lee, Berkeley
Model Fidelity

- To a *scientist*, the model is flawed.
- To an *engineer*, the realization is flawed.

I’m an engineer...
Perhaps we should be making our realizations more faithful to our models rather than the other way around?
“Essentially, all models are wrong, but some are useful.”


“Essentially, all system implementations are wrong, but some are useful.”

Lee and Sirjani, “What good are models,” FACS 2018.
“Simulation is doomed to succeed.”
[anonymous]

Could this statement be confusing engineering models for scientific ones?

Lee and Sirjani, “What good are models,” FACS 2018.
Changing the Question

Is the question whether our models describe the behavior of real-time systems (with high fidelity)?

Or

Is the question whether we can build real-time systems where behavior matches that of our models (with high probability)?
The hardware out of which we build computers is capable of delivering “correct” computations and precise timing…

Synchronous digital logic delivers precise, repeatable timing.

… but the overlaying software abstractions discard timing.

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

Lee, Berkeley
PRET Machines – Giving Software the Capabilities its Hardware Already Has.

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

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

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

Computing

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 zero effect on timing
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)
Our Second Generation PRET

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

**Isaac Liu, PhD Thesis, 2012**
Our Third-Generation PRET:
Open-Source FlexPRET (Zimmer 2014/15)

- 32-bit, 5-stage thread interleaved pipeline, RISC-V ISA
  - **Hard real-time HW threads:** scheduled at constant rate for isolation and repeatability.
  - **Soft real-time HW threads:** share all available cycles for efficiency.

- Deployed on Xilinx FPGA

![Diagram showing scheduling of threads]

Every 3 cycles (unless done)

Instructions

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 remaining cycles

SRAM scratchpad shared among threads

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

The real-time performance of a FlexPRET machine is never worse than that of a conventional machine.

Proof: A FlexPRET machine is a conventional machine if the memory-mapped registers controlling HRT and SRT threads is set to have only one thread, a SRT thread.
Benefits

• Four hardware threads is enough to eliminate all pipeline bubbles and memory latency variability.
• Unrealistic task models become realistic.
  – Exact, known WCET.
  – Zero-interference tasks.
  – Interrupts enabled at all times.
• High-precision timing instructions
  – Repeatable nanosecond precision

Lee, Berkeley
The Cost

Size:

[Zimmer, Broman, Shaver, Lee, RTAS 2014]

Lee, Berkeley
A **baseline RISC-V** without any complex instructions (floating point, integer division, packed instructions) can be realized on an FPGA with 580 flip flops and 2,788 LUTs.

A **4-thread FlexPRET** can be realized with 908 flip flops and 3,943 LUTs, an increase of 56% and 41% respectively.

Percentage is much lower with floating point, division, etc. [Zimmer, Broman, Shaver, Lee, RTAS 2014]
“[M]any a systems programmer’s grey hair bears witness to the fact that we should not talk lightly about the logical problems created by that feature”

- Edsger Dijkstra (1972)
Interrupts

• Nondeterministically interleaved with program
• Make response time > execution time
• Disrupt cache and branch predictors
• Overhead of context switching

• For WCET analysis, have to disable interrupts
• Disabling interrupts increases variability in response time
Interrupts

Scientific solution:
• Model all these effects

Engineering solution:
• Eliminate all these effects

The latter is what PRET machines do.
Such interrupts have *no effect* on HRT threads, and *bounded effect* on SRT threads!

A similar strategy is also used by XMOS, but with less isolation.
This paper shows that achieving deterministic response times that meet deadlines, when that is feasible, comes at no cost in worst-case response times.

This is shown for a task model of \( N \) sporadic independent tasks with deadlines.
Intuition

• \( N \) sporadic real-time tasks with minimum interarrival time \( T_i \), deadlines \( D_i \), and WCET \( C_i \).

**Theorem:** When \( T_i = D_i \), PRET yields deterministic response times no worse than the worst case response time of a conventional architecture.

When \( T_i > D_i \), if any processor can deliver deterministic response times, PRET will, with worst case response time no worse than a conventional architecture.
Benefits of PRET
(Even if you don’t care about determinism)

• Very low context switch overhead
  – Up to the number of hardware threads.
  – Conventional overhead above that.

• Tighter WCET analysis
  – Particularly when activating enough threads to eliminate pipeline bubbles and memory access order dependencies.
Benefits of PRET
(If you take advantage of determinism)

- **Modularity**
  - Non-interference between tasks.
  - Interrupts have *exactly no effect* on hard-real-time tasks.
- **Exactness**
  - Can get not just WCET, but actual ET.
  - Not just ET, but *response* time.
- **Repeatability**
  - Works in the field like on the bench.
  - Event ordering can be made invariant.
- **Complexity**
  - More hard-real-time tasks is better than fewer.
- **Certifiability**
  - Every *correct* execution of the software gives the same behavior.
- **Energy**
  - Reduce voltage and frequency to the bare minimum to meet deadlines.
Opportunities for the Future

- Synthesizing HRT thread schedules on the fly
- OS support for dynamic HRT thread instantiation
- Exploiting potential reduction in energy
- Programming models with temporal semantics
• **PRET**: time-deterministic architectures  
  – [http://chess.eecs.berkeley.edu/pret](http://chess.eecs.berkeley.edu/pret)

• **PTIDES**: distributed real-time software  
  – [http://chess.eecs.berkeley.edu/ptides](http://chess.eecs.berkeley.edu/ptides)

These enable models with tightly controlled timing and deterministic behaviors.

We have shown that these models are practically realizable at reasonable cost.
Using Time Instead of Timeout for Fault-Tolerant Distributed Systems

LESLIE LAMPORT
SRI International

A general method is described for implementing a distributed system with any desired degree of fault-tolerance. Instead of relying upon explicit timeouts, processes execute a simple clock-driven algorithm. Reliable clock synchronization and a solution to the Byzantine Generals Problem are assumed.


General Terms: Design, Reliability

Additional Key Words and Phrases: Clocks, transaction commit, timestamps, interactive consistency, Byzantine Generals Problem

ACM Transactions on Programming Languages and Systems, 1984.

Lee, Berkeley
Abstract: Discrete-event (DE) models are formal system specifications that have analyzable deterministic behaviors. Using a global, consistent notion of time, DE components communicate via time-stamped events. DE models have primarily been used in performance modeling and simulation, where time stamps are a modeling property bearing no relationship to real time during execution of the model. In this paper, we extend DE models with the capability of relating certain events to physical time...
Google independently developed a very similar technique and applied it to distributed databases.
PTIDES: Discrete-Event Semantics + Synchronized Clocks + Sensors and Actuators

Time-stamped events that are processed in time-stamp order.

This MoC is widely used in simulation and HDLs.

Given time-stamped inputs, it is a deterministic concurrent MoC.

A few texts that use the DE MoC

Lee, Berkeley
Google Spanner – A Reinvention of PTIDES

Update to a record comes in. Time stamp $t_1$.

Query for the same record comes in. Time stamp $t_2$.

Distributed database with redundant storage and query handling across data centers.
If $t_2 < t_1$, the query response should be the pre-update value. Otherwise, it should be the post-update value.
Update to a record comes in. Time stamp $t_1$.

Communication latency bound $b$.

Synchronize clocks with error bound $e$.

Query for the same record comes in. Time stamp $t_2$.

When the local clock time exceeds $t_2 + e + b$, issue the current record value as a response.
Update to a record comes in. Time stamp $t_1$.

Communication latency bound $b$.

Synchronize clocks with error bound $e$.

Query for the same record comes in. Time stamp $t_2$.

If after sending a response, we receive a record update with time stamp $t_1 < t_2$ declare a fault. Spanner handles this with a transaction schema.
Assume bounds on:

- *clock synchronization error*
- *network latency*

then *events are processed in time-stamp order* at every component. If in addition we assume

- *bounds on execution time*

then events are delivered to actuators on time.

See http://chess.eecs.berkeley.edu/ptides
PTIDES Requires Synchronized Clocks with Bounded Error

Every engineered design makes assumptions about its execution platform.

If we can count on ubiquitous clock synchronization, we have a new and powerful tool.

Lee, Berkeley
A meta-language for PRET, Ptides, and predictable concurrent systems in general.

To Appear, Design Automation Conference (DAC), June, 2019.
Lingua Franca

A meta-language for PRET, Ptides, and predictable concurrent systems in general.

To Appear, Design Automation Conference (DAC), June, 2019.
Conclusion

- In science, the value of a model lies in how well its behavior matches that of the physical system.

- In engineering, the value of the physical system lies in how well its behavior matches that of the model.

My message: Do less science and more engineering.

http://ptolemy.berkeley.edu/pret
http://ptolemy.berkeley.edu/ptides

http://platoandthenerd.org