A Personal View of Real-Time Computing

Edward A. Lee

Professor of the Graduate School

Invited Talk

École Normale Supérieure
Paris, France, February 26, 2020

University of California at Berkeley
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
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.

Lee, Berkeley
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

Lee, Berkeley
Messy Time

Time becomes a mess with interrupts and threads

**Model**

```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
}
```

• Model the details
• Analyze the models

Even deterministic real-time models can lead to chaos.

[Thiele and Kumar, EMSOFT 2015]
Newtonian Time for Physical Dynamics

Physical System

Model

Newtonian time advances everywhere uniformly in a continuum.

Lee, Berkeley
Pitfall with Newtonian Time (1)

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

Lee, Berkeley  Image: Wikimedia Commons

```c
1  void initTimer(void) {
2      SysTickPeriodSet(SysCtlClockGet() / 1000);  
3      SysTickEnable();  
4      SysTickIntEnable();  
5  }
6  volatile uint timer_count = 0;
7  void ISR(void) {
8      if(timer_count != 0) {  
9          timer_count--;  
10     }
11  }
12  int main(void) {  
13      SysTickIntRegister(&ISR);  
14      // other init  
15      timer_count = 2000;  
16      initTimer();  
17      while(timer_count != 0) {  
18          // code to run for 2 seconds  
19      }  
20      // other code  
21  }
```

\[
\dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau
\]
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?
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?”
Models vs. Reality

The model

\[
x(t) = x(0) + \int_0^t v(\tau) d\tau
\]
\[
v(t) = v(0) + \frac{1}{m} \int_0^t F(\tau) d\tau.
\]

The target (the thing being modeled).

In this example, the *modeling framework* is calculus and Newton’s laws.

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

Lee, Berkeley
A Model

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

Lee, Berkeley
A Physical Realization

Lee, Berkeley
To a scientist, the model is flawed.
To an engineer, the realization is flawed.

To a realist, both are flawed...
Perhaps we should be making our realizations more faithful to our models rather than the other way around?

Lee, Berkeley
“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.

Lee, Berkeley
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]);
}
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

// 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)

**Hardware thread**
- Registers

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

**Scratch pad**
- SRAM scratchpad shared among threads

**Memory**
- DRAM main memory, separate banks per thread

**I/O devices**

Pipeline Interleaving is Old

First used in the CDC 6600.
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

Every 3 cycles (unless done)

- HRTT 0
- SRTT 1
- SRTT 2

Whenever cycle available (arbitrary interleaving)

Clock cycles

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.
• 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
The Cost

Size:

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

Lee, Berkeley
The Cost

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.

Lee, Berkeley
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.

Lee, Berkeley
Avionics Mixed Criticality Test Case

- Abstract workload of 21 tasks derived from a time-partitioned avionics system\(^1\) with 93% utilization
- Each iteration performs identical computation, but...

- Deployed on 8 HW threads
- Execution time dependent on HRTTs


Programming: Basic Timing Control in PRET

Extended RISC-V ISA with timing instructions. E.g. Hard-real-time periodic task:

```c
time r;
get_time(r);
while(1) {
    add_ms(r, 10);
    exception_on_expire(r);
    task();
    deactivate_exception();
    delay_until(r);
}
```

Four Patterns of Timed Code Blocks

**[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
```
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?*
Instruction Set Architecture (ISA)

The concept of an ISA hasn’t changed much since the 1960s.

Fred Brooks
Photo courtesy of computerhistory.org
Parametric PRET Machines

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.

Lee, Berkeley
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.

• No longer need to be restricted to polling I/O
  – Effect of interrupts is bounded.

Lee, Berkeley
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.
So why isn’t every modern processor a FlexPRET?

**Possibilities:**

- Our claims look too good to be true.
- It’s a paradigm shift.
- Programming models that can take advantage of this are missing.
A polyglot meta-language with DE semantics for implementation (not simulation) of deterministic, concurrent, time-sensitive systems.

Today: Lingua Franca

Lingua Franca Wiki

Topics

- Overview
- Language Specification
- Writing Reactors in C
- Accessors Target
- Downloading and Building

Papers

- EMSOFT 2019 work-in-progress paper.
- DAC 2019 paper on Reactors.

https://github.com/icyphy/lingua-franca/wiki

Contents

Overview
- Reactors
- Time
- Real-Time Systems
- References

Language Specification
- Import Statement
- Reactor Block
  - Parameter Declaration
  - State Declaration
  - Input Declaration
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
https://github.com/icyphy/lingua-franca
http://platoandthenerd.org