Precision Timed (PRET) Machines

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

BWRC Open House,
Berkeley, CA
February , 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*

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.

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!

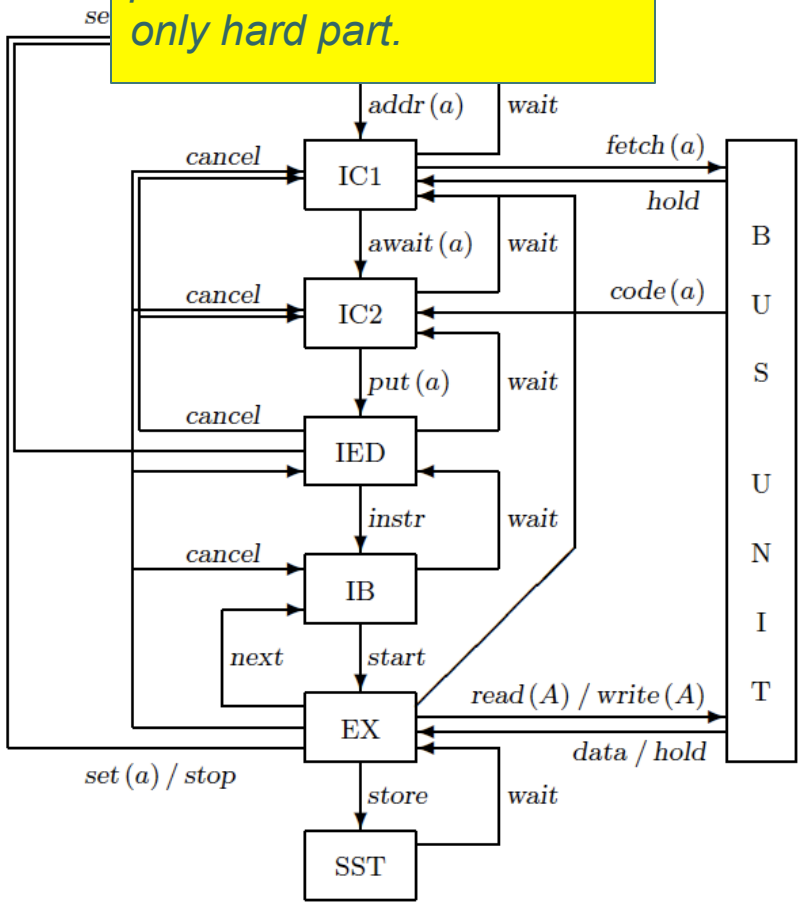
Analyzing software for timing behavior requires:

- 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.

Our goal is to reduce the problem so that this is the only hard part.



Wilhelm, et al. (2008). "The worst-case execution-time problem - overview of methods and survey of tools." ACM TECS 7(3): p1-53.

PRET Machines

- **PRE**cision-**T**imed processors = **PRET**
- **P**redictable, **RE**peatable **T**iming = **PRET**
- **P**erformance *with* **RE**peatable **T**iming = **PRET**

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

+

= PRET

Computing *With time*

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();
}
```

```
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.

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!

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

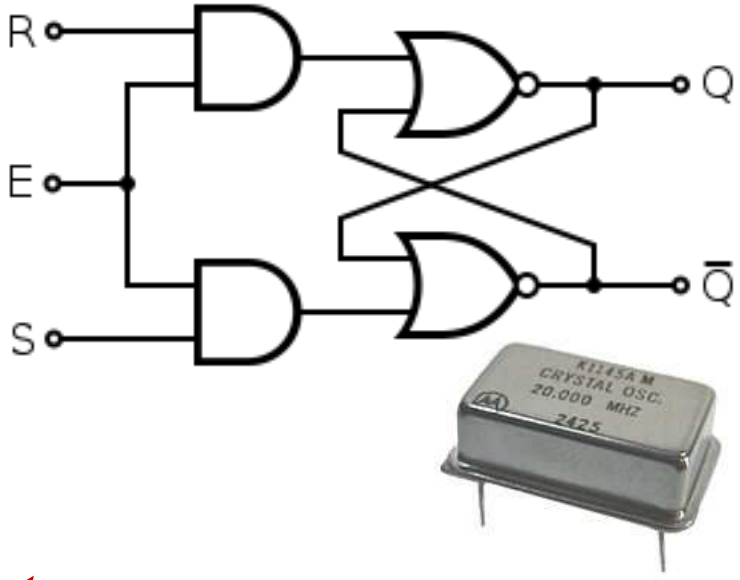
```
set_time r1, 1s
exception_on_expire r1, 1
// Code block
deactivate_exception 1
delay_until r1
```

[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.

```
// 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:

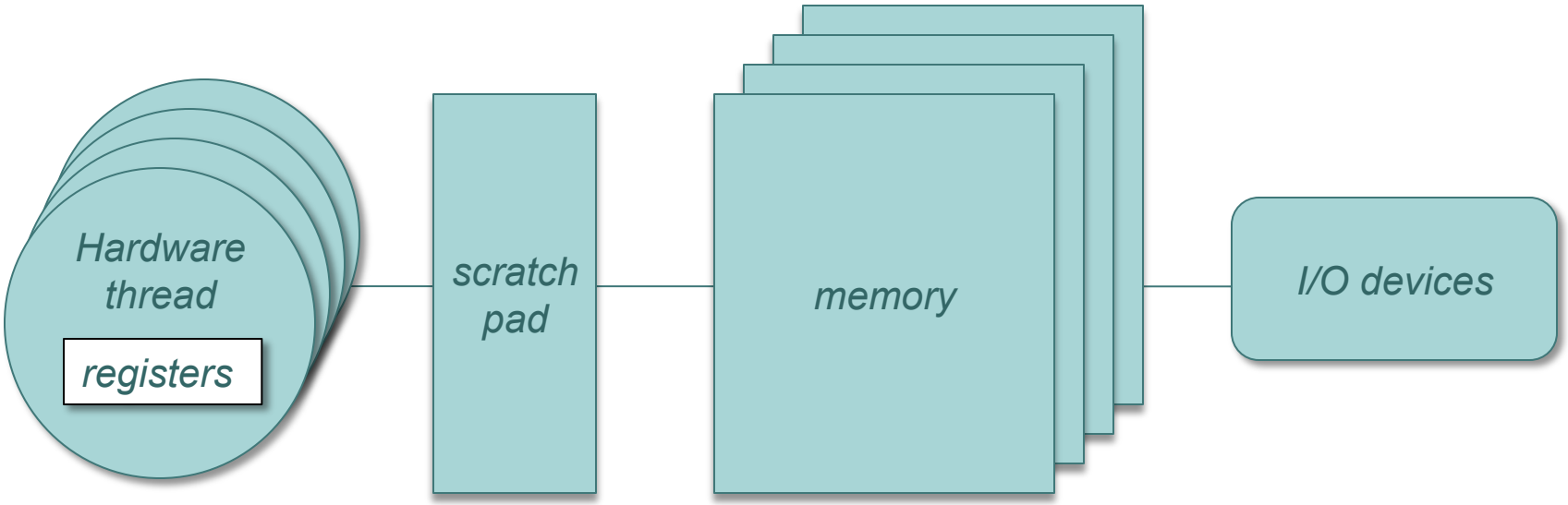
- Pipelining
- Memory hierarchy
- I/O (DMA, interrupts)
- Power management (clock and voltage scaling)
- On-chip communication
- Resource sharing (e.g. in multicore)

Our Current PRET Architecture

PTArm, a soft core on a Xilinx Virtex 5 FPGA

Note inverted memory compared to multicore!

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



Interleaved pipeline with one set of registers per thread

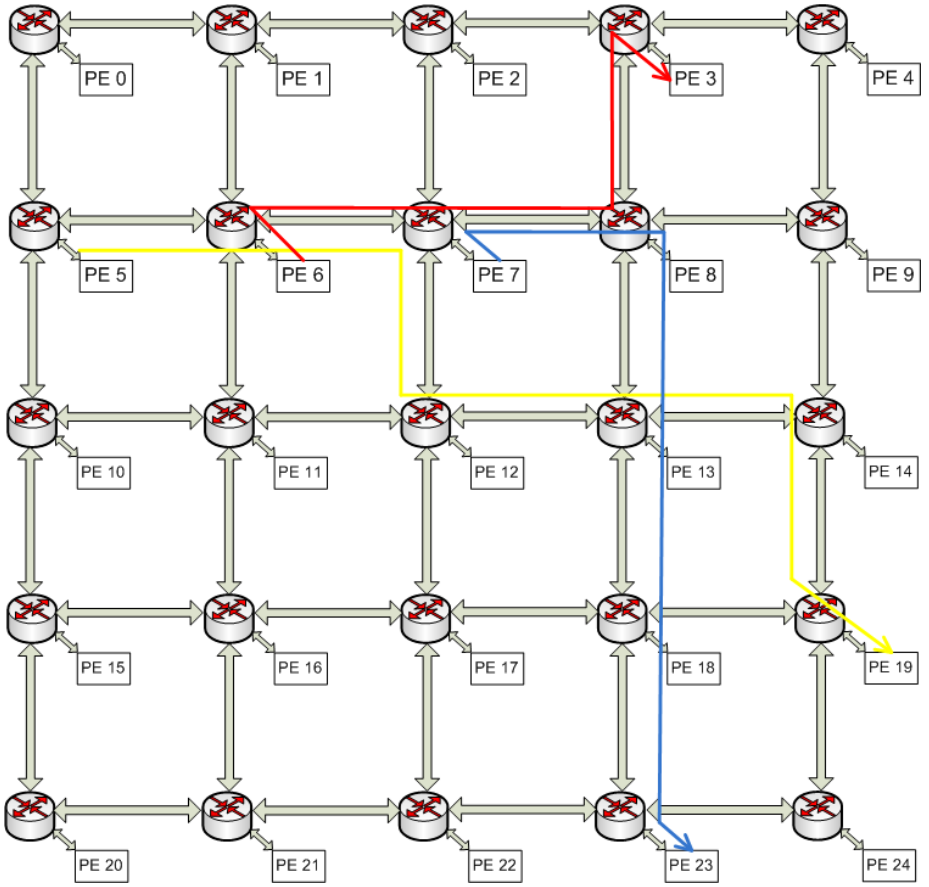
SRAM scratchpad shared among threads

DRAM main memory, separate banks per thread

On a Virtex 6, we can fit 55 cores, for a total of 330 concurrent threads with perfectly controllable timing.

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

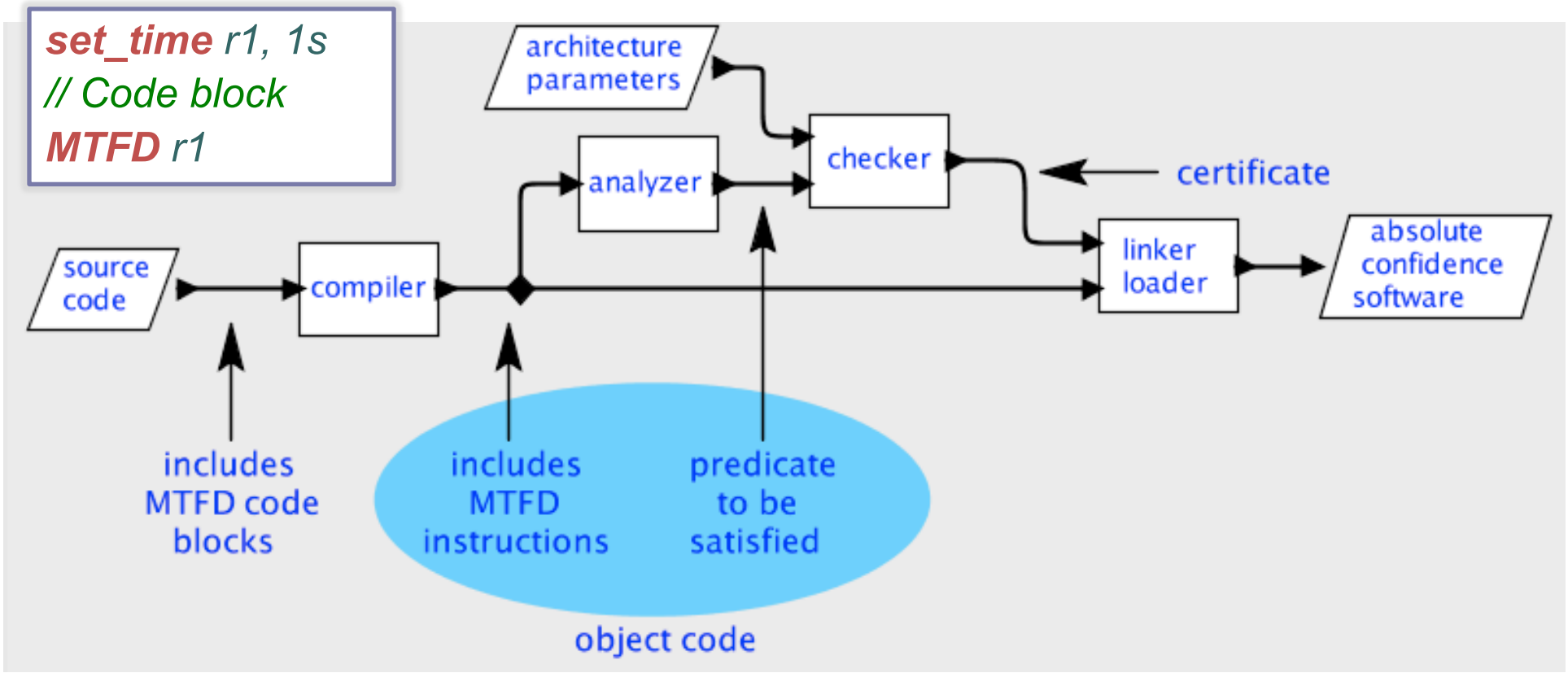
```
set_time r1, 1s
// Code block
MTFD r1
```

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

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

- S. Edwards and E. A. Lee, "**The Case for the Precision Timed (PRET) Machine**," in the *Wild and Crazy Ideas* Track of DAC, June 2007.
- B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards and E. A. Lee, "**Predictable programming on a precision timed architecture**," CASES 2008.
- S. Edwards, S. Kim, E. A. Lee, I. Liu, H. Patel and M. Schoeberl, "**A Disruptive Computer Design Idea: Architectures with Repeatable Timing**," ICCD 2009.
- D. Bui, H. Patel, and E. Lee, "**Deploying hard real-time control software on chip-multiprocessors**," RTCSA 2010.
- Bui, E. A. Lee, I. Liu, H. D. Patel and J. Reineke, "**Temporal Isolation on Multiprocessing Architectures**," DAC 2011.
- J. Reineke, I. Liu, H. D. Patel, S. Kim, E. A. Lee, **PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation** (to appear), CODES +ISSS, Taiwan, October, 2011.
- S. Bensalem, K. Goossens, C. M. Kirsch, R. Obermaisser, E. A. Lee, J. Sifakis, **Time-Predictable and Composable Architectures for Dependable Embedded Systems**, Tutorial Abstract (to appear), EMSOFT, Taiwan, October, 2011

Conclusions

Overview References:

- *Bui, et al. Temporal Isolation on Multiprocessing Architectures, DAC 2011*
- *Lee. **Computing needs time**. CACM, 52(5):70–79, 2009*
- *Edwards & Lee, The case for Precision Timed (PRET) Machines, DAC 2007*

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

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

Raffaello Sanzio da Urbino – The Athens School