Precision Timed Infrastructure
Making Time an Engineering Abstraction

52nd IFIP WG 2.4 Meeting, Vadstena, Sweden
May 22, 2012

David Broman
broman@eecs.berkeley.edu
EECS Department
UC Berkeley, USA
and
Linköping University, Sweden

Key contributors and collaborators related to the PRET project
Edward A. Lee
Steven A. Edwards
Jeff Jensen
Isaac Liu
Slobadon Matic
Hiren Patel
Jan Reineke
Sanjit Seshia
Michael Zimmer
Jia Zou
Sungjun Kim

A Story…

Success?

Fly-by-wire technology controlled by software.

They have to purchase and store microprocessors for at least 50 years production and maintenance…

Why?

Safety critical ➔
Rigorous validation and certification

Apparently, the software does not specify the behaviour that has been validated and certified!
Key Challenge

Timing is not part of the software semantics

Correct execution of programs (e.g., in C, C++, C#, Java, Scala, Haskell, OCaml) has nothing to do with how long time things takes to execute.

Traditional Approach

Programming Model

Timing Dependent on the Hardware Platform

Our Objective

Programming Model

Make time an engineering abstraction within the programming model

Timing is independent of the hardware platform (within certain constraints)

Agenda

Part I
What is PRET?

Part II
Timing semantics at the ISA level

Part III
Timing semantics for high-level languages
Part I
What is PRET?

PRET Infrastructure

• PRET Language (Language timing semantics)
• PRET Compiler (Timing aware compilation)
• PRET Machine (Computer Architecture)
Focus on cyber-physical systems with real-time constraints

<table>
<thead>
<tr>
<th>Missed deadline</th>
<th>Hard task</th>
<th>Firm task</th>
<th>Soft task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic consequence</td>
<td>Result is useless, but causes no damage</td>
<td>Result has still some utility</td>
<td></td>
</tr>
</tbody>
</table>

**Predictable timing**
- Guarantee correctness (WCET)

**Early miss detection**

**Immediate miss detection**

**Late miss detection**

**Task**
- (execution time in clock cycles)

**Deadline**
- (measured in e.g., ns)

**Processor frequency**

**Precision of timing**
- Enable accuracy in nano seconds

**Repeatable timing**
- Same platform: Testability
- Changing platform: Portability

**Part II**
Timing semantics at the ISA level
Rethink the ISA
Timing has to be a correctness property not only a performance (quality) property

PRET Machine
• Repeatable and predictable execution time (instructions)
• Repeatable memory access time
• Timing instructions for handling missed deadline detection

Related Work
Java Optimized Processor (JOP) (Schoebner, 2008)
ARPRET (Andalam et al., 2009)

ARMv4 ISA extended with timing constructs

Best effort (with padding)

New instruction get time (gt)

New instruction delay until (du)

We do not detect missed deadlines!

Padding using delay until

Processor frequency

Task
(execution time in clock cycles)

Deadline
(Time (measured in e.g., ns)
Part I
What is PRET?

Part II
Timing semantics at the ISA level

Part III
Timing semantics for high-level languages

ARMv4 ISA extended with timing constructs

Late miss detection

Soft task
Result has still some utility

Late miss detection

Processor frequency

Deadline

Time

(measured in e.g., ns)

Task
(execution time in clock cycles)

Not handled at the ISA level

Needs static timing analysis! (Part III)

Handled using timing exceptions
New instructions:
- ee - Exception on Expire
- de - Deactivate Exception

Early and immediate miss detection

Early miss detection

Immediate miss detection

Processor frequency

Deadline

Time

(measured in e.g., ns)

Task
(execution time in clock cycles)
We need implementations that deliver repeatable timing

The good news
Fortunately, electronics technology delivers highly reliable and precise timing

The bad news…
The chip architectures introduces highly non-deterministic behavior (e.g., using caches, pipelines etc.).

We need to rethink the microarchitecture
• Pipelining
• Memory hierarchies
• I/O (DMA, interrupts)
• Power management
• On-chip communication
• Resource sharing (e.g., in multicore)

Our Current PRET Architecture

PTARM, a soft core on Xilinx Virtex 5 FPGA

Xilinx Virtex 5, FPGA, 100 MHz

Hardware thread

registers

scratch pad

main memory

I/O devices

Thread-interleaved Pipeline
4 threads, 5 stage pipeline

Scratchpad shared among threads
1 thread cycle for load/store

DRAM main memory, separate banks per thread

Load 4 thread cycle
Store 1-2 thread cycles
Part III
Timing semantics for high-level languages
(work-in-progress)

Timing Aware Compilation

<table>
<thead>
<tr>
<th>Modeling Language</th>
<th>Simulink</th>
<th>Ptolemy</th>
<th>Modelica</th>
<th>Real-time Maude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Language</td>
<td>Real-time Concurrent C (Gehani and Ramamritham, 1991)</td>
<td>PRET-C (Andalam et al., 2009)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WCC (Falk and Lokuciejewski, 2010)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Part I
What is PRET?

Part II
Timing semantics at the ISA level

Part III
Timing semantics for high-level languages
(work-in-progress)
Timing Aware Compilation

Part I
What is PRET?

Part II
Timing semantics at the ISA level

Part III
Timing semantics for high-level languages

Some of the main challenges

Relating real-time to execution-time
Execution time in clock cycles – oscillators not precise enough
- IEEE1588 clock synchronization (needs WCET margin)

Compiling with execution time
(1) Estimate WCET of ASM program
(2) WCET-aware compilation to minimize WCET
Scratchpad allocation is now a compiler problem
- Optimization problem with timing constraints
  (1) For a given frequency, optimize (average case) performance
  (2) Find lowest frequency fulfilling constraints

Making time independent of the hardware
For the same ISA, different platform parameters affect timing
(Scratchpad size, clock frequency, memory latency, etc.)
- Compiler generates a certificate – checked at load time
Conclusions

Main takeaway points

Time is a correctness factor – not just a performance (quality) factor

Today, timing behavior is a property of the realizations of a software system

The PRET approach aims at making time an engineering abstraction within the programming model

Thank you for listening!