National Workshop on the New Clockwork for Time-Critical Systems
October 26, 2012

David Broman
broman@eecs.berkeley.edu
UC Berkeley and
Linköping University

Stephen A. Edwards
sedwards@cs.columbia.edu
Columbia University

Edward A. Lee
eal@eecs.berkeley.edu
UC Berkeley

PRET Infrastructure at Berkeley
David Broman
Jian Cai
Hokeun Kim
Yooseong Kim
Edward A. Lee
Aviral Shrivastava
Michael Zimmer

PRET Machine Collaborators and Alumni
Steven A. Edwards
Isaac Liu
Jeff Jensen
Slobadan Matic
Sanjit Seshia
Sungjun Kim
Hiren Patel
Jia Zou
Jan Reineke

Agenda

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Summary of Challenges
Part I
Cyber-Physical Systems

Modeling, Simulating, and Compiling
Cyber-Physical Systems

\[
\begin{align*}
J_1 \omega_1 &= M_s - M_l \\
J_2 \omega_2 &= M_h - M_l \\
\omega_1 &= -r \omega_2 \\
M_1 &= -r^{-1} M_2 
\end{align*}
\]

Equation-based model

Various models of computation (MoC)

Simulation with timing properties

Modeling

System

Physical system (the plant)

Cyber system: Computation (embedded) + Networking
Rapid development of CPS with high confidence of correctness is a co-design problem

The design of
Physical system
(the plant)

The design of
Cyber system:
Computation (embedded)
+ Networking

influence each other

Cyber/Physical Co-design Problem

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Summary of Challenges

Cyber/Physical Co-design

Model
Equation-based model

Model fidelity problem
“Ensuring that the model accurately imitates the real system”

Challenge #1:
Compile/synthesize the model’s cyber part, such that the simulated model and the behavior of the real system coincide.

The main challenge is to guarantee correct timing behavior.
Part II
Precision Timed Infrastructure

A Story…

Fly-by-wire technology controlled by software.
Safety critical ➔
Rigorous validation and certification

Success?

Why?

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

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

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Summary of Challenges
What is PRET?

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

Our Objective

Programming Model
Timing Dependent on the Hardware Platform

Make time an abstraction within the programming model
Timing is independent of the hardware platform (within certain constraints)

What is Precision Timed (PRET) Infrastructure?

A vision of making time first class citizen in both software and hardware.

PRET Infrastructure

- PRET Language (Language with timing semantics)
- PRET Compiler (Timing aware compilation)
- PRET Machine (Computer Architecture)

Focus until now has been on PRET machines
What do mean by precision, predictable, and repeatable timing?

Focus on cyber-physical systems with real-time constraints

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

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

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

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

**Languages with timing semantics**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Languages</td>
<td>Real-time Concurrent C (Gehani and Ramamritham, 1991)</td>
<td>PRET-C (Andalam et al., 2009)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Assembly Languages**
- The assembly languages for today’s processors lack the notion of time
Rethink the ISA
Timing has to be a correctness property not only a performance (quality) property

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

Related Work

<table>
<thead>
<tr>
<th>Java Optimized Processor (JOP) (Schoeberl, 2008)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARPRET (Andalam et al., 2009)</td>
</tr>
</tbody>
</table>

Our Current PRET Architecture

PTARM, a soft core on Xilinx Virtex 5 FPGA

Xilinx Virtex 5, FPGA, 75 MHz

Hardware thread

scratch pad

main memory

I/O devices

Thread-interleaved Pipeline

Scratchpad shared among threads

DRAM main memory, separate banks per thread

4 threads, 5 stage pipeline
Subset of ARMv4 ISA extended with timing constructs

broman@eecs.berkeley.edu

15

Part I

Cyber-Physical Systems

Part II

Precision Timed Infrastructure

Part III

Summary of Challenges

New instruction `get time (gt)`

```
예: `gt r1, r2 ; get time (ns)`
```

-- Code block --
```
예: `adds r2, r2, #500 ; add 500 ns`
```
```
예: `adc r1, r1, #0`
```
```
예: `mtfd r1, r2 ; takes at most 500 ns`
```
```
예: `du r1, r2 ; takes at least 500 ns`
```

At runtime – machine error exception. (Critical error mode – “stop the train”)

Early – before execution. Needs to be guaranteed by static analysis of program code

Early miss detection

```
delay until (du).
```

Processor frequency

Task

(clock cycles)

Deadline

Time

PRET Infrastructure

broman@eecs.berkeley.edu

Part I

Cyber-Physical Systems

Part II

Precision Timed Infrastructure

Part III

Summary of Challenges

Modeling Languages

Simulink/Stateflow (Mathworks)

Modelica (Modelica Associations)

Ptolemy II (Eker et al., 2003)

Giotto (Henzinger, Horowitz, and Kirsch, 2003)

Modelyze (Broman and Siek, 2012)

Programming Languages

Semantic gap between timed high level modeling languages and PRET ISA

Assembly Languages

PRET ISA

16
Part I  
Cyber-Physical Systems

Part II  
Precision Timed Infrastructure

Part III  
Summary of Challenges

---

**PRETIL vision**

Modeling Languages

- Simulink/Stateflow (Mathworks)
- Modelica (Modelica Associations)
- Ptolemy II (Eker et al., 2003)

Programming Languages

- C extended with high-level timing constructs.
- Can be seen both as an intermediate and programming language

Assembly Languages

- Expose timing constructs
- Abstracting away memory the hierarchy (scratchpad, DRAM etc.)

**ptC**

- E machine (Henzinger, and Kirsch, 2007)

**PRETIL**

Challenge #2: How do we *guarantee* the correctness of synthesis/compilation?

Challenge #3: In an intermediate language, what is the right abstraction level for expressing semantics of time and concurrency?
Execution Time and Deadlines

Goal: Guarantee that \( e_i \leq D_i \)

But, the execution time may depend on:
- Input data (e.g., an image)
- Machine states (e.g., caches and pipelines)

Execution time, \( e_i \)

Relative deadline \( D_i \) (derived from MTFD)

Release time, \( r_i \)

absolute deadline, \( d_i \)

---

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Summary of Challenges

---

Worst-Case Execution Time (WCET)

Measurement-based approach
- Cannot guarantee to find WCET
- Applicable for any task

Worst-case execution time (WCET)

Static program analysis approach
- Upper bound of WCET
- Cannot handle any task (conservative)

Average-case execution time (ACET)

WCET overview (Wilhelm et al., 2008)

Challenges
- To make it safe: \( \text{upper bound} \geq \text{WCET} \)
- To make it tight: minimize \( \text{upper bound} - \text{WCET} \)
Sub-problems for timing analysis

Analysis on Code
- Flow analysis
  - Loop bounds
  - Infeasible paths
- Implicit Path Enumeration Technique (IPET) (Li and Malik, 1995)

Low-level Analysis
- Timing of basic blocks
- Caches (Reineke, 2008)
- Pipelines (Thesing, 2004)

Computation – an ILP problem

**Is the end result the WCET?**

No, the result is in clock cycles: **Worst-Case no of Clock Cycles (WCCC)**

Part I
Cyber-Physical Systems

Part II
Precision Timed Infrastructure

Part III
Summary of Challenges

Relating clock cycles and time

Simple translation to worst-case execution time:

\[
\text{WCCC} / \text{clock\_frequency} = \text{WCET}
\]

**Example 1:**

10’000 cycles / 100 MHz = 0.1 ms

Based on assumptions:
- The clock frequency is constant (e.g., not the case for frequency/voltage scaling)
- The CPU’s clock (oscillator) is accurate (which is typically not the case).

**Example 2: Clock synchronization**

- Real-time clock
  - RT = 0.51 ms after computation finished
- Clock freq. = 100MHz,
  - 1 cycle = 10ns
  - \( D_i = 0.5 \) ms
  - WCCC = 49 000
  - WCET = 0.49 ms

**Challenge #5:**
How to relate worst-case no of clock cycles with real time, when clocks are dynamically corrected?

| WCET < \( D_i \) but RT > \( D_i \) |
Part II
Summary of Challenges

#1: Compile/synthesize the model's cyber part, such that the simulated model and the behavior of the real system coincide.

#2: How do we guarantee the correctness of synthesis/compilation?

#3: In an intermediate language, what is the right abstraction level for expressing semantics of time and concurrency?

#4: How to make safe, tight, WCET analysis to scale for large complex tasks?

#5: How to relate worst-case no of clock cycles with real time, when clocks are dynamically corrected?

Thank you for listening!