Modeling, Simulating, and Compiling with Timing Semantics

TU Vienna
Austria
October 4, 2012

University of Salzburg
Austria
October 2, 2012

David Broman
broman@eecs.berkeley.edu

EECS Department
UC Berkeley, USA

and

Linköping University, Sweden

Modelyze project
David Broman
Jeremey G. Siek

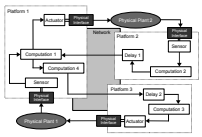
PRET Infrastructure
David Broman
Jian Cai
Hokeun Kim
Yooseong Kim
Edward A. Lee
Aviral Shrivastava
Michael Zimmer

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

Agenda

Part I

Cyber/Physical Co-Design Challenges



Part II

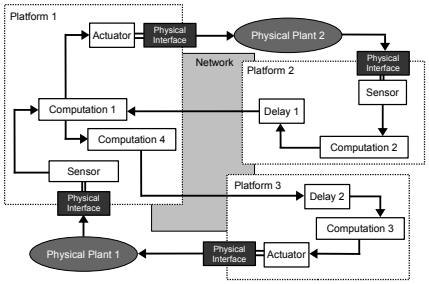
Modelyze

Part III

Precision Timed Infrastructure

broman@eecs.berkeley.edu

Part I
Cyber/Physical Co-design Challenges



Cyber-Physical Systems (CPS)

broman@eecs.berkeley.edu

Industrial Robots

Power Plants

Aircraft

Modeling, Simulating, and Compiling Cyber-Physical Systems

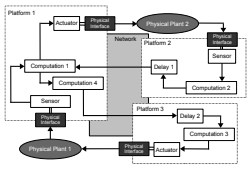
broman@eecs.berkeley.edu

Model

$$
\begin{aligned}
J_1\dot{\omega}_1 &= M_v - M_1 \\
J_2\dot{\omega}_2 &= M_h - M_2 \\
\omega_1 &= -r\omega_2 \\
M_1 &= -r^{-1}M_2
\end{aligned}
$$

Equation-based model

Simulation with timing properties



Various models of computation (MoC)

Modeling

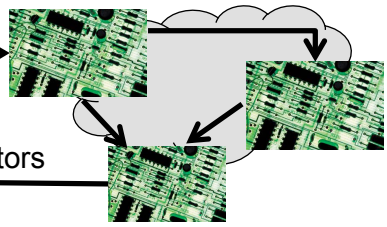
Hardware-in-the-loop (HIL) simulation

Modeling

Physical system available?

System

Sensors



Actuators

Physical system (the plant)

Cyber system: Computation (embedded) + Networking

Cyber/Physical Co-Design Problem

broman@eecs.berkeley.edu

Rapid development of CPS with high confidence of correctness is a co-design problem

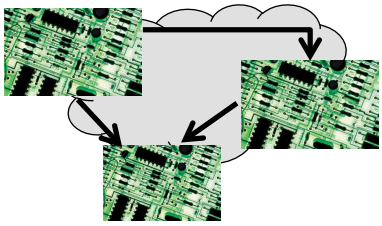
The design of

Physical system (the plant)

influence each other

The design of

Cyber system: Computation (embedded) + Networking



Cyber/Physical Co-design Challenge #1

broman@eecs.berkeley.edu

Expressiveness problem

"A modeling language's capability to handle new modeling needs"

Analyzability problem

"A modeling language's capability to prove or test model properties"

Challenge #1:
Making modeling languages expressive and analyzable

System

Actuators

Physical system (the plant) **Cyber system**: Computation (embedded) + Networking

Cyber/Physical Co-design Challenge #2

broman@eecs.berkeley.edu

Model

$$
\begin{aligned}
J_1\dot{\omega}_1 &= M_v - M_1 \\
J_2\dot{\omega}_2 &= M_h - M_2 \\
\omega_1 &= -r\omega_2 \\
M_1 &= -r^{-1}M_2
\end{aligned}
$$

Equation-based model

Various models of computation (MoC)

Physical prototyping

Modeling

Model fidelity problem

"Ensuring that the model accurately imitates the real system"

Modeling

Compiling/ synthesizing

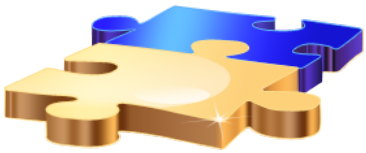
Challenge #2:
Compile/synthesize the model's cyber part, such that the simulated model and the behavior of the real system coincide.

The main challenge is to guarantee correct timing behavior.

broman@eecs.berkeley.edu

Part II

Modelyze



What is Modelyze?

broman@eecs.berkeley.edu

Modelyze
(MODEL and analYZE)

Purpose: Research language – address the expressiveness and analyzability problem by making the language <u>extensible</u>

Small, simple, host language for embedding domain-specific languages (DSL) of different models of computation (MoC)

Key aspect: Both the DSL and models in the DSL are defined in Modelyze

Gradually typed functional language (call-by-value)

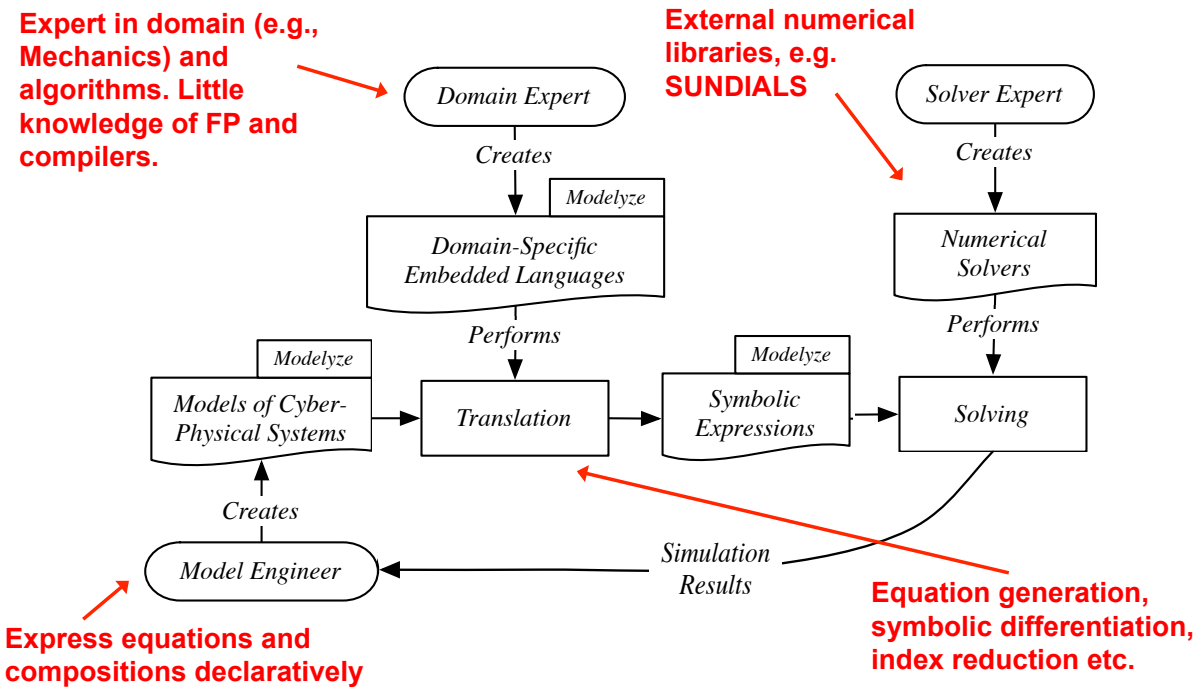
Novelty: Typed symbolic expressions

Formal semantics for a core of the language. Proven type soundness for the core.

Prototype implementation (interpreter). Evaluated for series of equation-based DSLs.

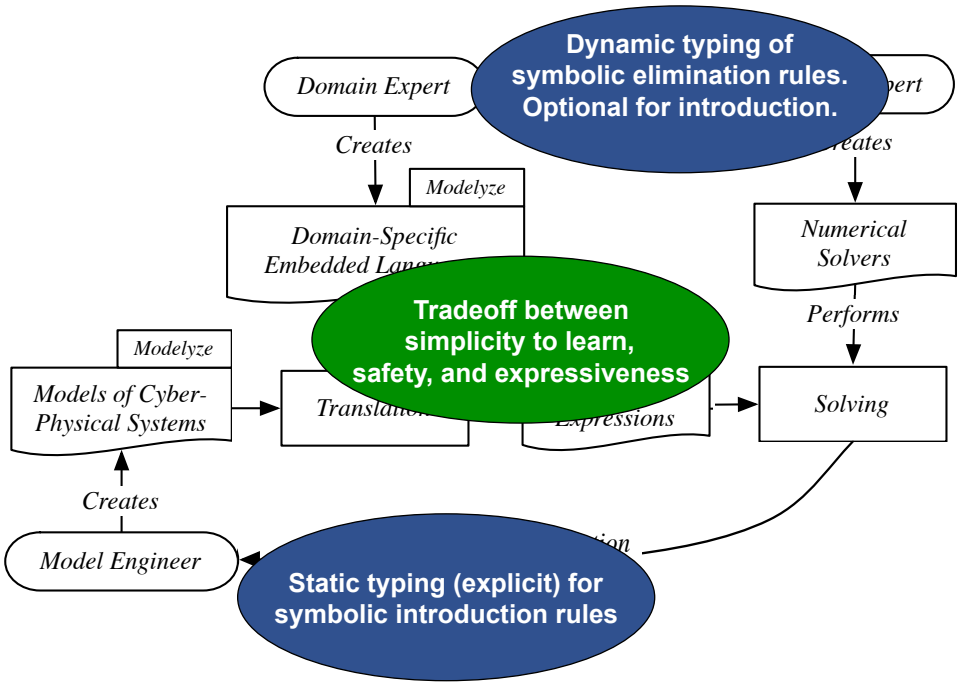
Process Overview

broman@eecs.berkeley.edu



Process Overview

broman@eecs.berkeley.edu



Related Work

broman@eecs.berkeley.edu

Implementing DSLs

Compiler construction

- JastAdd (Ekman & Hedin, 2007)
- MetaModelica (Pop & Fritzson, 2006)

Preprocessing and template metaprogramming

- C++ Templates (Veldhuizen, 1995)
- Template Haskell (Sheard & Peyton Jones, 2002)
- Stratego/XP (Bravenboer et al., 2008)

Embedded DSLs

- Haskel DSELs, e.g., Fran (Ellito & Hudak, 1997), Lava (Bjesse et al. 1998), and Paradise(Augustsson, 2008)
- FHM (Nilsson et al., 2003)
- Pure embedding (Higher-order functions, polymorphism, lazy evaluation, type classes) (Hudak, 1998)

Combining Dynamic and Static Typing

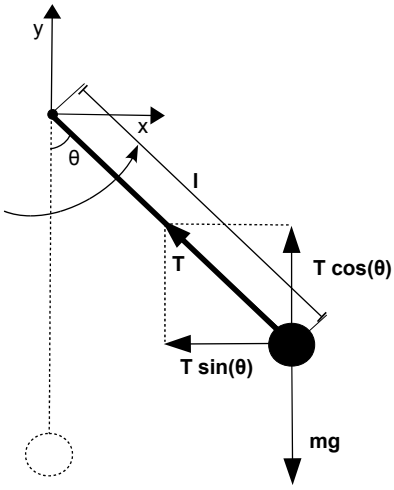
- Gradual Typing (Siek & Taha, 2007)
- Soft Typing (Cartwright & Fagan, 1991)
- Dynamic type with typecase (Abadi et al., 1991)
- Typed Scheme, Racket (Tobin-Hochstadt, Felleisen, 2008)
- Thorn, like types (Wrigstad et al., 2010)

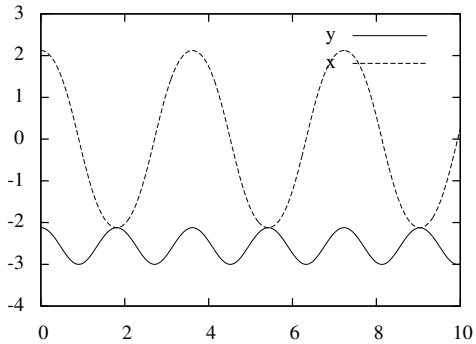
Representing Code and Data type

- Dynamic languages LISP, Mathematica
- MetaML <T> (Taha & Sheard, 2000)
- GADT (Peyton Jones et al.,2006; Xi et al., 2003; Cheney & Ralf, 2003)
- Open Data types (Löh & Hinze, 2006)
- Pattern Calculus (Jay, 2009)

Pendulum Example

broman@eecs.berkeley.edu





$$-T \cdot \frac{x}{l} = m\ddot{x}$$

$$-T \cdot \frac{y}{l} - mg = m\ddot{y}$$

$$x^2 + y^2 = l^2$$

$$x(0) = l\sin(\theta_s)$$

$$y(0) = -l\cos(\theta_s)$$

Differential-Algebraic equations

Algebraic constraint

Initial values

Declarative Mathematical Model

broman@eecs.berkeley.edu

Using function abstraction to define the model

Unknowns are given types but not bound to values

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

Equations and initial values are defined declaratively, just as the mathematical equations

$$-T \cdot \frac{x}{l} = m\ddot{x} \qquad x(0) = l\sin(\theta_s)$$

$$-T \cdot \frac{y}{l} - mg = m\ddot{y} \qquad y(0) = -l\cos(\theta_s)$$

$$x^2 + y^2 = l^2$$

Declarative Mathematical Model

broman@eecs.berkeley.edu

Which parts are part of the host language (Modelyze)?

Unknowns are internally represented as <u>typed symbols</u>

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

$s : \tau$

Fresh (unique) symbol

Tagged with a type

Variable `x` is bound to fresh symbol of type `<Real>`

$<\tau>$

Symbolic type

Release the user from annotation burden

broman@eecs.berkeley.edu

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

Symbols cannot be bound to values, so `x^2` would crash at runtime

Use quasi-quoting to mix symbolic expressions and program code?

Using MetaML syntax `< >` for quotation and `~` for anti-quoting (escape)

```
<~x^2 + ~y^2 = ~((fun t -> <t>)l^2)>;
```

Heavy annotation burden for the end-user

Symbol Lifting Analysis (SLA)

broman@eecs.berkeley.edu

Symbol Lifting Analysis (SLA): During type checking, lift expressions that cannot be safely evaluated at runtime into symbolic expressions (data).

$$\Gamma \vdash_L e \rightsquigarrow e' : \tau$$

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

Rewritten to prefix curried form

```
((/) x) l)
```

where

```
(/):Real-> Real -> Real
x:<Real>
l:Real
```

```
(((lift(/):Real-Real->Real) @ x) @ (lift l:Real))
```

Resulting type `<Real>`

Division cannot be performed, lift expression to type `<Real-> Real -> Real>`.

Term `lift e:T` wrapps `e` and results in type `<T>`

Term `e1@e2` is a symbolic application, represented as a tuple.

Pattern Matching on Symbolic Expressions

broman@eecs.berkeley.edu

Dynamic symbolic type <?>

Accumulator Sets of symbolic type <Real>

Query for all unknowns in a model instance

```
def getUnknowns(exp:<?>, acc:(Set <Real>)) -> (Set <Real>) = {
    match exp with
    | e1 e2 -> getUnknowns(e2,getUnknowns(e1,acc))
    | sym:Real -> Set.add exp acc
    | _ -> acc
}
```

Uniform data structure, no boilplate code (matching on symbolic applications)

Match all symbols of type <Real> i.e., unknowns in the model.

```
getUnknowns(Pendulum(5,3,45*pi/180),Set.empty)
```

Returns a set with 3 symbols (representing x, y, and T).

Static Error Checking at the DSL Level

broman@eecs.berkeley.edu

Syntactically correct model (host syntax)

Static type error instead of dynamic error during translation/pattern matching.

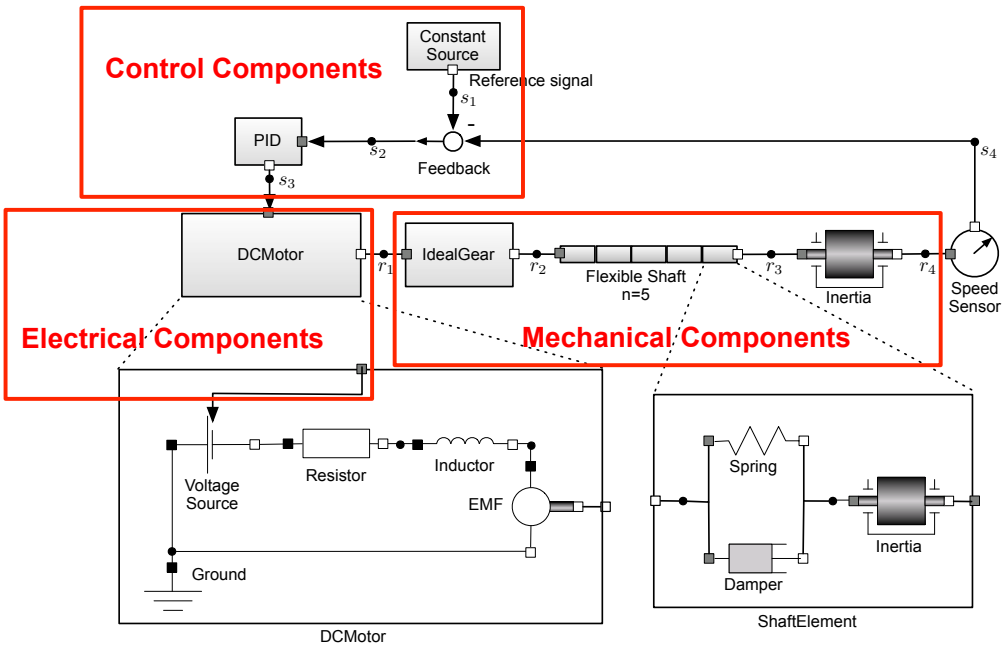
```
def ModifiedPendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y;                    //Error 1: Missing initial value

    -T*der/l = m*x'';          //Error 2: der used incorrectly
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

Quite intuitive error messages at the DSL level.

```
modifiedpendulum.moz 4:10-4:10 error: Missing argument
of type 'Real'.
```

Mechatronic Control Example

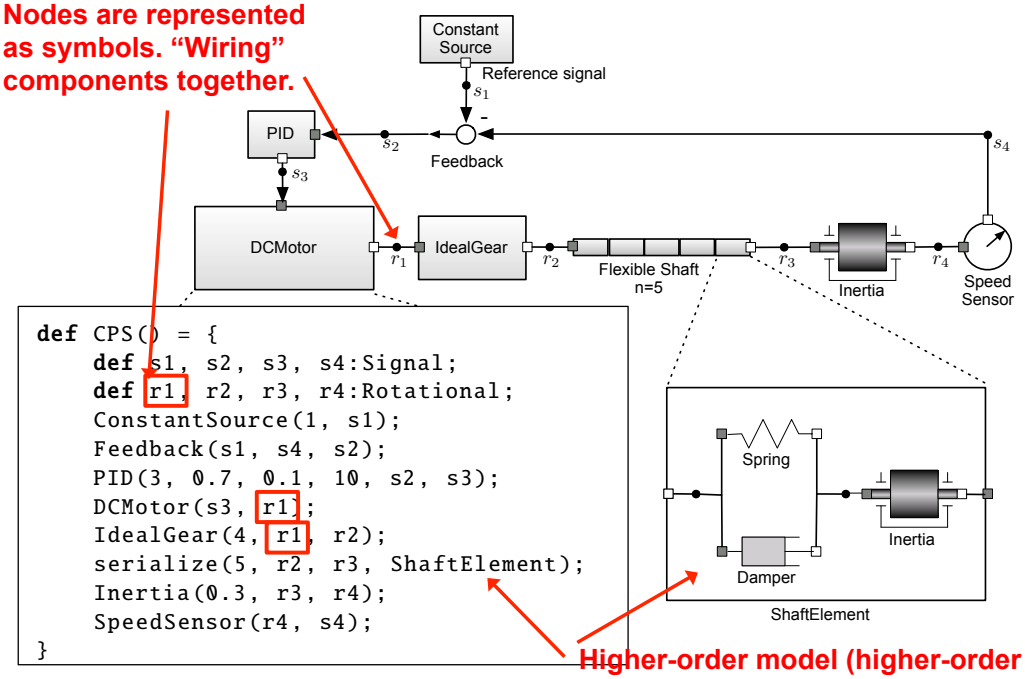
broman@eecs.berkeley.edu



Mechatronic Control Example

broman@eecs.berkeley.edu

Nodes are represented as symbols. "Wiring" components together.

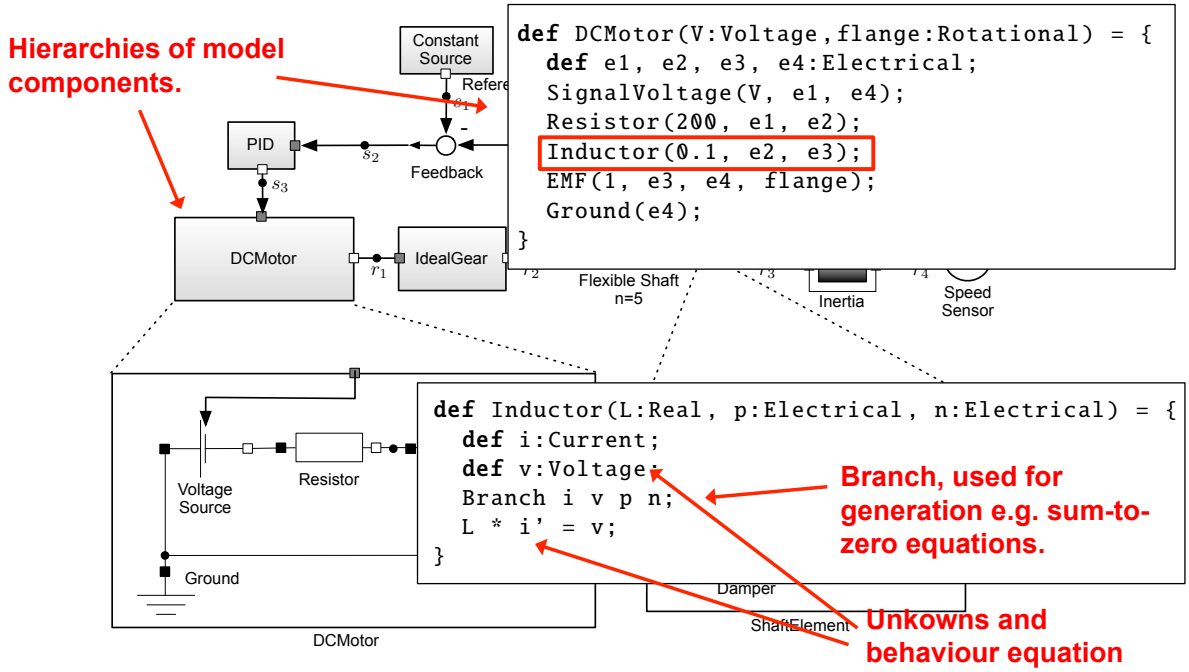



```
def CPS() = {
    def s1, s2, s3, s4:Signal;
    def r1, r2, r3, r4:Rotational;
    ConstantSource(1, s1);
    Feedback(s1, s4, s2);
    PID(3, 0.7, 0.1, 10, s2, s3);
    DCMotor(s3, r1);
    IdealGear(4, r1, r2);
    serialize(5, r2, r3, ShaftElement);
    Inertia(0.3, r3, r4);
    SpeedSensor(r4, s4);
}
```

Higher-order model (higher-order function)

Mechatronic Control Example

broman@eecs.berkeley.edu




```
def DCMotor(V:Voltage,flange:Rotational) = {
  def e1, e2, e3, e4:Electrical;
  SignalVoltage(V, e1, e4);
  Resistor(200, e1, e2);
  Inductor(0.1, e2, e3);
  EMF(1, e3, e4, flange);
  Ground(e4);
}
```

```
def Inductor(L:Real, p:Electrical, n:Electrical) = {
  def i:Current;
  def v:Voltage;
  Branch i v p n;
  L * i' = v;
}
```

Part I
Cyber/Physical
Co-design Challenges

Part II
Modelyze

Part III
Precision Timed Infrastructure

Conclusions

Modelyze: main takeaway points

Addressing the modeling expressiveness and analyzability problems using an embedded DSL approach.

Both DSLs and models are defined in the same language.

Symbol lifting analysis (SLA) is used to release the annotation burden from the end user

Typed symbols with gradual typing is used for pattern matching and DSL-level error reporting

For more information, see preprint of submitted journal paper:

David Broman and Jeremy G. Siek. “Modelyze: a Gradually Typed Host Language for Embedding Equation-Based Modeling Languages”, Tech. Report UCB/EECS-2012-173, University of California, Berkeley, June, 2012.

broman@eecs.berkeley.edu

Part II
Precision Timed Infrastructure

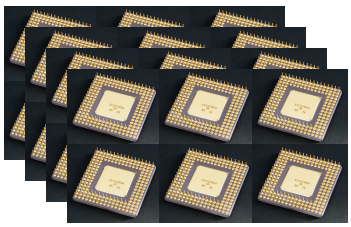
A Story…

broman@eecs.berkeley.edu

Fly-by-wire technology controlled by <u>software.</u>

<u>Safety critical</u> ➔
Rigorous validation and certification

Success?



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

Why?

Apparently, the <u>software</u> does not specify the behaviour that has been validated and certified!

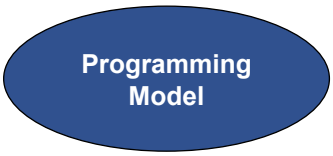
What is PRET?

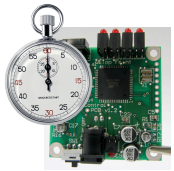
broman@eecs.berkeley.edu

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.

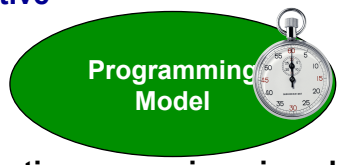
Traditional Approach



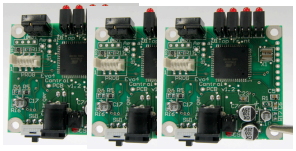


Timing Dependent on the Hardware Platform

Our Objective



Make time an engineering abstraction within the programming model



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

What is Precision Timed (PRET) Infrastructure?

broman@eecs.berkeley.edu

PRET Infrastructure

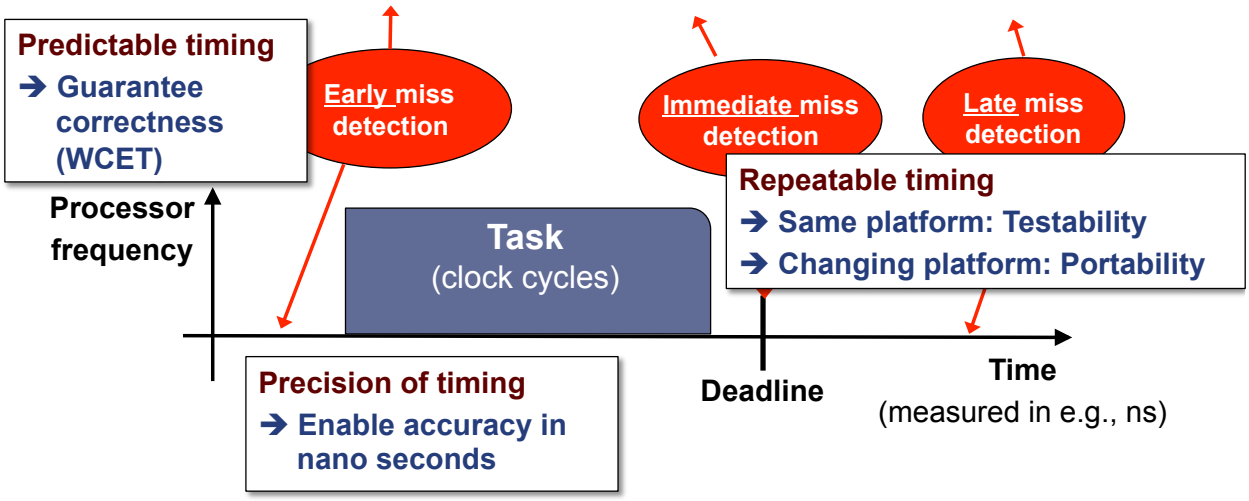
- **PRET Language (Language with timing semantics)**
- **PRET Compiler (Timing aware compilation)**
- **PRET Machine (Computer Architecture)**

What do mean by precision, predictable, and repeatable timing?

broman@eecs.berkeley.edu

Focus on cyber-physical systems with real-time constraints

	Hard task	Firm task	Soft task
Missed deadline	Catastrophic consequence	Result is useless, but causes no damage	Result has still some utility



Languages with timing semantics

broman@eecs.berkeley.edu

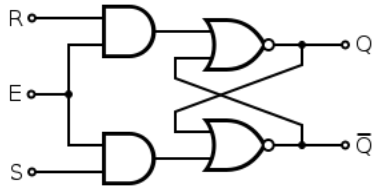
Modeling Languages	Simulink/ Stateflow (Mathworks)	Modelica (Modelica Associations)	Ptolemy II (Eker et al., 2003)	Giotto (Henzinger, Horowitz, and Kirsch, 2003)	Modelyze (Broman and Siek, 2012)
Programming Languages	Real-time Concurrent C (Gehani and Ramamritham, 1991)	PRET-C (Andalam et al., 2009)			
Assembly Languages	The assembly languages for todays processors lack the notion of time				

We need implementations that deliver deterministic timing behavior

broman@eecs.berkeley.edu

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, interupts)**
- **Power management**
- **On-chip communication**
- **Reseource sharing (e.g., in multicore)**

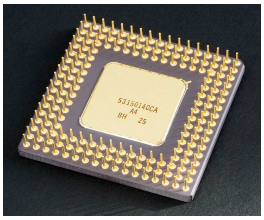


Photo by Andrew Dunn, 2005

Instruction set architecture (ISA)

broman@eecs.berkeley.edu

Rethink the ISA

Timing has to be a *correctness* property not only a *performance* (quality) property

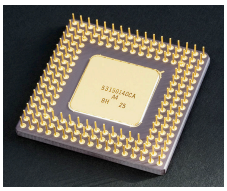


Photo by Andrew Dunn, 2005

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) (Schoeberl, 2008)

ARPRET (Andalam et al., 2009)

ARMv4 ISA extended with timing constructs

broman@eecs.berkeley.edu

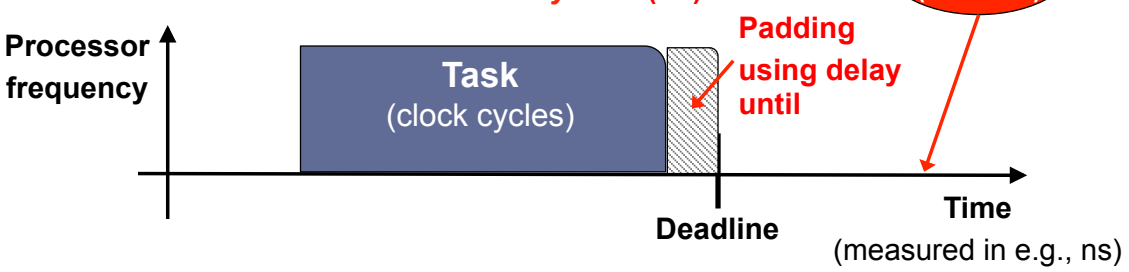
Padding and late miss detection

New instruction *get time* (gt)

```
gt        r1, r2          ; get time (ns)
  -- Code block --
adds      r2, r2, #500    ; add 500 ns
adc       r1, r1, #0      ; add with carry (time in 2 32-bit reg)
du        r1, r2          ; delay until 500ns have elapsed
```

Do get time again and compare

New instruction *delay until* (du).

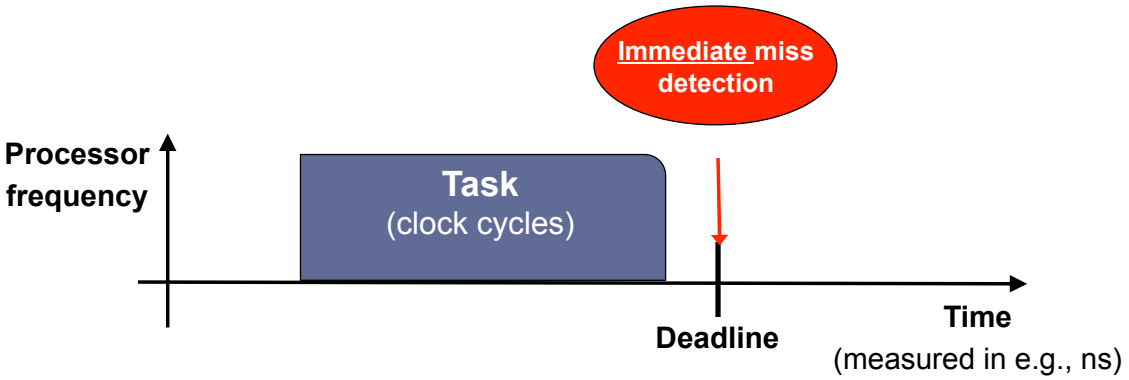


ARMv4 ISA extended with timing constructs

broman@eecs.berkeley.edu

Immediate miss detection

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



ARMv4 ISA extended with timing constructs

broman@eecs.berkeley.edu

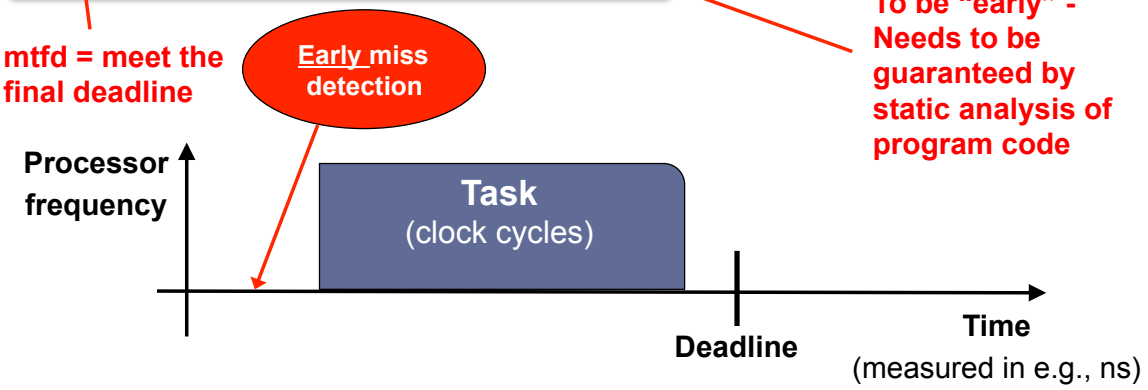
Early miss detection

```
gt      r1, r2        ; get time (ns)
  -- Code block --
adds    r2, r2, #500  ; add 500 ns
adc     r1, r1, #0
mtfd    r1, r2        ; takes at most 500 ns
du      r1, r2        ; takes at least 500 ns
```

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

To be "early" - Needs to be guaranteed by static analysis of program code

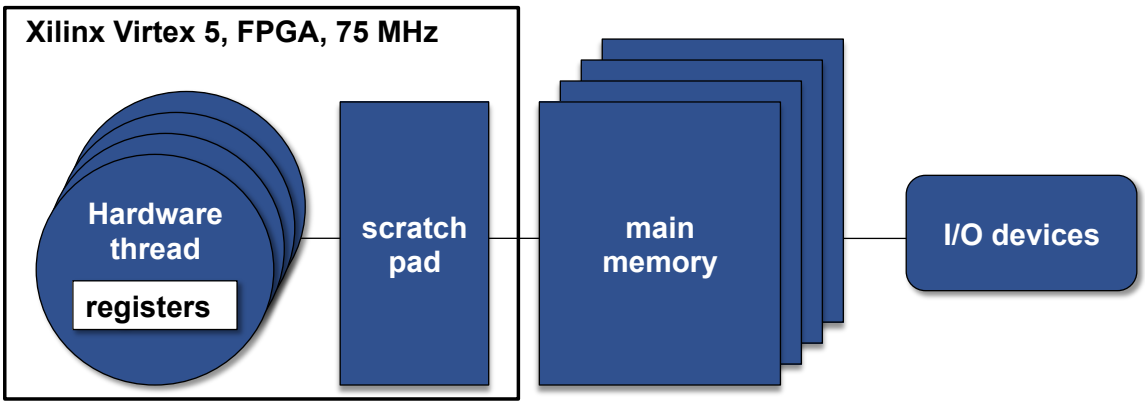
mtfd = meet the final deadline



Our Current PRET Architecture

broman@eecs.berkeley.edu

PTARM, a soft core on Xilinx Virtex 5 FPGA



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
(1 thread cycle = 4 processor cycles)

PRET Infrastructure

broman@eecs.berkeley.edu

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

PRET Infrastructure

broman@eecs.berkeley.edu

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

Can we just compile directly down to PTARM?

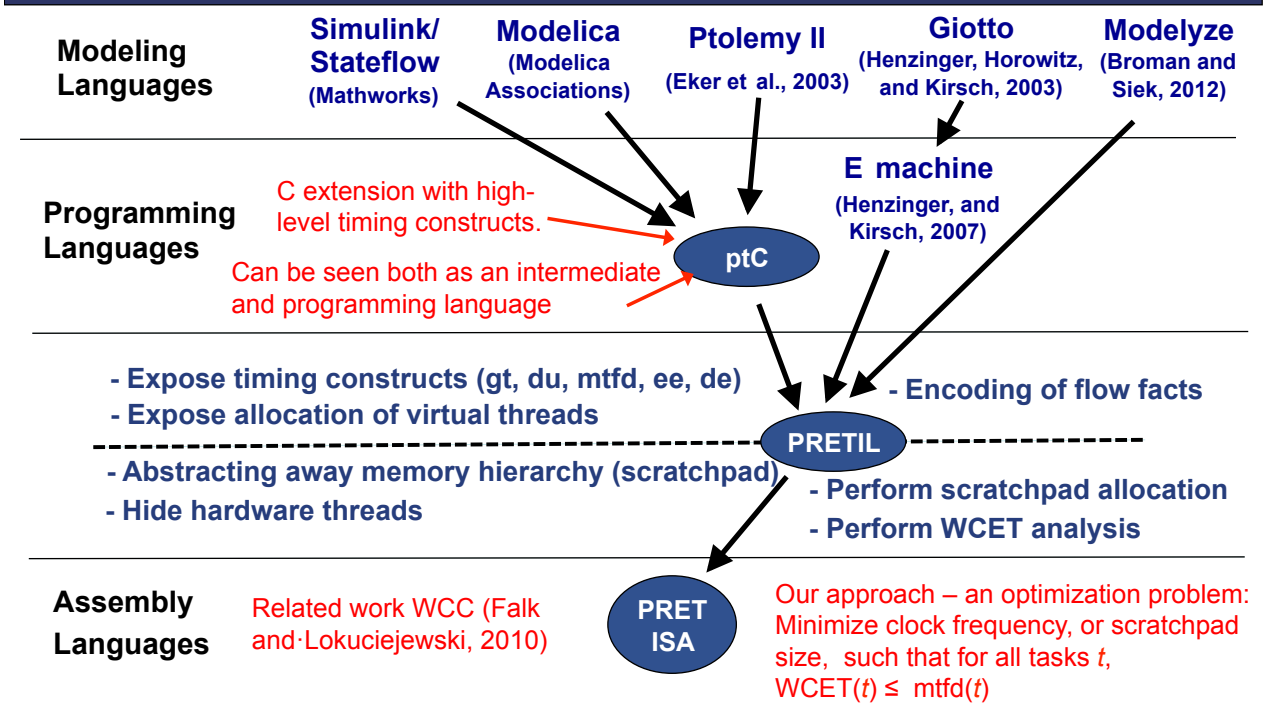
Lots of redundant work…

Assembly Languages

PRET ISA

PRETIL (work in progress)

broman@eecs.berkeley.edu



Part I
Cyber/Physical
Co-design Challenges

Part II
Modelyze

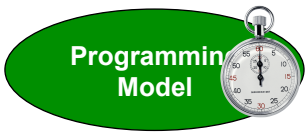
Part III
Precision Timed Infrastructure

Conclusions

PRET Infrastructure: 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 construct *within* the *programming model*

Thank you for listening!

For more information about PRET, see

Isaac Liu, Jan Reineke, David Broman, Michael Zimmer, and Edward A. Lee. A PRET Microarchitecture Implementation with Repeatable Timing and Competitive Performance. To appear in Proceedings of the 30th IEEE International Conference on Computer Design (ICCD), Montreal, Quebec, Canada, 2012.