Polychrony as an abstract model of computation for GALS architectures

Time Oriented Reliable Embedded Networked Systems
Toulouse – Nov. 2., 2010

Jean-Pierre Talpin, INRIA
Plan

Motivation
Models of computation
From synchrony to asynchrony
A definition of polychrony
An old fashioned watch
Clock synthesis
Compositional correctness
Conclusion and perspectives
Motivation

Heterogeneity of skills, teams, tools, methods

CATIA, Nastran, Simulink, Scade, Rhapsody, ...

CAN, Flexray, ARINC 653, AADL, Profiling, Energy, ...

Heterogeneity of skills, teams, tools, methods...
Motivation

- CATIA
- Nastran
- Simulink
- Scade
- Rhapsody
- ...

- CAN
- Flexray
- ARINC 653
- AADL
- Profiling
- Energy
- ...

Co-modeling

- Analyse
- Verify
- Test

Map

Simulate
Motivation

Simulink

AADL

co-modeling

analyse

verify

test

map

simulate
Case study of an airplane doors control system

Functional specification

A suitable GALS model of computation

Structural specification

Simulation, verification, performance evaluation, scheduling, distribution …
Plan

Motivation

**Models of computation**
From synchrony to asynchrony
A definition of polychrony
An old fashioned watch
Clock synthesis
Compositional correctness
Conclusion and perspectives
Asynchrony

Asynchronous architecture models

AADL diagrams (or UML, or SysML)
Synchronous behavior models

Elements of Simulink (or Geneauto, Lustre, Scade)
Asynchrony, synchrony, polychrony

A polychronous model of computation

Signal (or RT-Builder, Marte’s CCSL, MRICDF)
Plan

Motivation

Models of computation

**From synchrony to asynchrony**

A definition of polychrony

An old fashioned watch

Clock synthesis

Compositional correctness

Conclusion and perspectives
Globally asynchronous implementation of a multi-clocked network of synchronous modules preserving functional correctness.

From synchrony to asynchrony

A polychronous network

a multi-clocked network of synchronous modules
Globally asynchronous implementation of a multi-clocked network of synchronous modules.
Goal
Globally asynchronous implementation of synchronous modules preserving functional correctness
From synchrony to asynchrony

Issue
Different timing domains
Synchronous signals vs. asynchronous channels
Reaction trigger vs. core activation
Issue
1. To ensure asynchrony incurs no extra behavior

=> Deadlock-freedom
From synchrony to asynchrony

**Issue**
2. To ensure determinism given asynchronous inputs

=> Confluence and no priority
From synchrony to asynchrony

Goal
Synthesize a correctness preserving container and the activation clock
From synchrony to asynchrony

Challenge
Can this be done modularly, iteratively, compositionally?
Plan

Motivation

Models of computation

From synchrony to asynchrony

A definition of polychrony

An old fashioned watch

Clock synthesis

Compositional correctness

Conclusion and perspectives
Globally asynchronous implementation of a network of synchronous modules with multiple clocks preserving functional correctness.

Polychrony

c = a default b

\[ T_c = T_a \cup T_b \]

a network of synchronous modules with multiple clocks
Polychrony

Globally asynchronous implementation of a network of synchronous modules with multiple clocks preserving functional correctness

\[ T_c = T_a \cap T_b \]

a network of synchronous modules with multiple clocks
Default and when are not flow functions

c = a default b
Implementing polychrony amounts to synthesizing a network of synchronized flow functions: a Kahn network.
Plan

Motivation

Models of computation

A definition of polychrony

From synchrony to asynchrony

An old fashioned watch
  - Semantics of Signal
    - Code generation
  
  Clock synthesis

Compositional correctness

Conclusion and perspectives
\[ X = \text{IN default } ZX - 1 \]
\[ ZX = X$ \text{ init } 0 \]
\[ \text{RST} = (ZX \leq 0) \]
\[ \text{IN} \ ^\wedge = \text{when RST} \]
An old-fashioned watch

\[ X = \text{IN default } ZX-1 \]
\[ ZX = X^\$ \text{ init } 0 \]
\[ \text{RST} = (ZX < 0) \]
\[ \text{IN} ^\^= \text{when RST} \]

\[ \text{IN} \ (t_0, 3) \quad (t_4, 3) \]
\[ \text{ZX} \ (t_0, 0) \ (t_1, 3) \ (t_2, 2) \ (t_3, 1) \ (t_4, 0) \]
\[ X \ (t_0, 3) \ (t_1, 2) \ (t_2, 1) \ (t_3, 0) \ (t_4, 3) \]
An old-fashioned watch

\[X = \text{IN default } ZX-1\]
\[ZX = X$ \text{ init } 0\]
\[\text{RST} = (ZX < 0)\]
\[\text{IN } ^\wedge = \text{ when RST}\]

\[ZX (t_0, 0) (t_1, 3) (t_2, 2) (t_3, 1) (t_4, 0)\]

\[X (t_0, 3) (t_1, 2) (t_2, 1) (t_3, 0) (t_4, 3)\]
An old-fashioned watch

X = IN default ZX-1
ZX = X$ init 0
RST = (ZX < 0)
IN ^= when RST

ZX (t₀, 0) (t₁, 3) (t₂, 2) (t₃, 1) (t₄, 0)
RST (t₀, 1) (t₁, 0) (t₂, 0) (t₃, 0) (t₄, 1)
X = IN default ZX-1
ZX = X$ init 0
RST = (ZX < 0)
IN ^= when RST

IN (t₀, 3) (t₄, 3)
RST (t₀, 1) (t₁, 0) (t₂, 0) (t₃, 0) (t₄, 1)
Plan

Motivation

Models of computation

From synchrony to asynchrony

A definition of polychrony

An old fashioned watch

- Semantics of Signal
- Code generation

Clock synthesis

Compositional correctness

Conclusion and perspectives
X = IN default ZX-1
ZX = X$ init 0
RST = (ZX ≤ 0)
IN ^= when RST

IN (t₀, 3) (t₄, 3)
ZX (t₀, 0) (t₁, 3) (t₂, 2) (t₃, 1) (t₄, 0)

A calculus to reason on synchronization relations

Cₓ = Cₗ + ^Czx
Czx = cx
Crst = czx
Cin = [RST]
A calculus to reason on scheduling relations

\[ X = \text{IN default } ZX-1 \]
\[ ZX = X^\$ \text{ init } 0 \]
\[ RST = (ZX \leq 0) \]
\[ \text{IN } \wedge = \text{ when RST} \]

\[
\begin{align*}
\text{IN} & (t_0, 3) \\
& (t_4, 3) \\
\Downarrow & \\
X & (t_0, 0) (t_1, 3) (t_2, 2) (t_3, 1) (t_4, 0) \\
\Uparrow & \\
ZX & (t_0, 0) (t_1, 3) (t_2, 2) (t_3, 1) (t_4, 0)
\end{align*}
\]

A calculus to reason on scheduling relations

\[
\begin{align*}
\text{IN} & \rightarrow X \text{ when (RST)} \\
ZX & \rightarrow X \text{ when (not RST)} \\
ZX & \rightarrow \text{ RST} \\
\text{RST} & \rightarrow \text{ IN}
\end{align*}
\]
Semantics and compilation

X = IN default ZX-1
ZX = X$ init 0
RST = (ZX <= 0)
IN ^= when RST

read CX;
RST = (X <= 0)
If RST then {
    read IN;
    X = IN;
}
else  X = X - 1;
write X;

Implementing the watch amounts to synthesize
1. A function that computes clocks

C_X = C_IN + C_ZX
C_ZX = C_X
C_RST = C_ZX
C_IN = [RST]
Semantics and compilation

\[
\begin{align*}
X &= \text{IN default ZX-1} \\
ZX &= \text{X$ init 0} \\
\text{RST} &= (ZX \leq 0) \\
\text{IN} &= \text{when RST}
\end{align*}
\]

Implementing the watch amounts to synthesize
1. A function that computes clocks
2. A schedule that respects causality

\[
\begin{align*}
\text{read CX;} \\
\text{RST} &= (X \leq 0) \\
\text{If RST then} \{ \\
\text{read IN;} \\
X &= \text{IN;} \\
\} \\
\text{else } X &= X - 1; \\
\text{write X;}
\end{align*}
\]

\[
\begin{align*}
ZX &\rightarrow \text{RST} \\
\text{RST} &\rightarrow \text{IN} \\
\text{IN} &\rightarrow X \text{ when (RST)} \\
ZX &\rightarrow X \text{ when (RST)}
\end{align*}
\]
Implementing the watch amounts to
synthesize
1. A function that computes clocks
2. A schedule that respects causality

It yields a synchronized flow function

\[ X = \text{watch} \left( C_X, \text{IN} \right) \]
Plan

Motivation

Models of computation

From synchrony to asynchrony

A definition of polychrony

An old fashioned watch

Clock synthesis

Compositional correctness

Conclusion and perspectives
A clock $c$ is a set of symbolic instants in time

$$C_x \quad x \text{ is present}$$

$$[x] \quad x \text{ is present and true}$$

$$[\neg x] \quad x \text{ is present and false}$$

A clock relation is an invariant on timing properties

$$C_x = C_y \ast [z] \quad x \text{ is present iff } y \text{ is present and } z \text{ is true}$$

The goal of clock synthesis is to define a clock function from a set of relations
The computability of a clock by another defines an equivalence relation. For example,

For any Boolean signal $x$, the clocks $[x]$ and $[\neg x]$ are computable from $C_x$

Synchronous signals $x$ and $y$ are in the same equivalence class
A hierarchy is built by attaching all clock equations to the equivalence relation.

Suppose that

If $C_a = f(C_y, C_z)$ then we can add
Plan

Motivation

Models of computation

From synchrony to asynchrony

A definition of polychrony

An old fashioned watch

Clock synthesis

**Compositional correctness**

Conclusion and perspectives
Correctness

Endochrony

A program whose clock hierarchy is a tree is endochronous:

All clocks can be computed from a root equivalence class

Example of the watch:

```
read CX;
RST = (X <= 0)
If RST then {
    read IN;
    X = IN;
}
else  X = X - 1;
write X;
```
Correctness

Endochrony

A program whose clock hierarchy is a tree is endochronous:

All clocks can be computed from a root equivalence class

Unfortunately, endochrony is not compositional
However, asynchronous determinism (AD) is compositional

If a module p is endochronous then it is AD

If p and q are AD and if p | q is non blocking then p | q is AD

A simple (static) and **compositional** criterion
Correctness

Asynchronous determinism (AD) solves our two issues

1. Deadlock-freedom

2. Confluence and no priority
Corollary

A compositional methodology to iteratively build isochronous systems by the static analysis of clock and scheduling relations of synchronous modules.
Plan

Motivation
Models of computation
From synchrony to asynchrony
A definition of polychrony
An old fashioned watch
Clock synthesis
Compositional correctness

Conclusion and perspectives
Conclusion

The polychronous model of computation and communication

A clear and solid semantics
  Compositional correctness

A calculus of synchronization and scheduling
  Sequential, modular, distributed code generation
  Interface models, abstraction and refinement

Model transformations
  Heterogeneous architecture modeling and analysis
  Virtual prototyping

Tools

  RT-Builder by Geensoft
  Polychrony by INRIA team Espresso
  MRICDF by Virginia Tech and USAF laboratories
  UML Marte’s CCSL, Timesquare tool by INRIA team Aoste
A DSL for software architecture exploration
  • Data-flow for computation
  • Mode automata for control
  • Libraries for services

A toolbox of services
  • Code generation
  • Model transformation
  • Model checking
  • Controller synthesis

An eclipse interactive interface
  • Open import functionalities
  • High-level visual editor
  • Analysis and transformation
  • Visualization and traceability

To be released under GPL v2.0 licence at http://www.irisa.fr/espresso/Polychrony
On the model of computation

On desynchronization

On architecture modeling

On virtual prototyping

On model-driven engineering

Papers available from http://www.irisa.fr/prive/talpin