Abstract Clock-based Approaches in the Programming, Design and Analysis of Embedded Systems

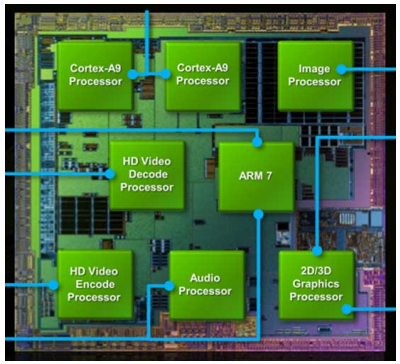
Abdoulaye GAMATIÉ
LIRMM/CNRS, Montpellier
abdoulaye.gamatie@lirmm.fr

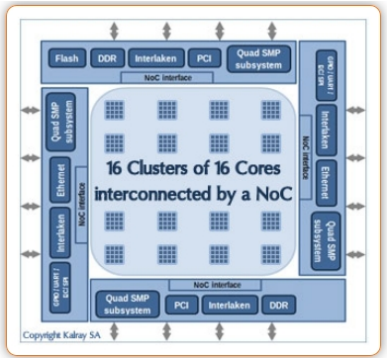
Laboratoire
Informatique
Robotique
Microélectronique
Montpellier

Embedded applications

- smart & data-intensive: high #functions, data amounts
- real-time: time-bounded reactivity w.r.t. environment

Multi-Processor System-on-Chip (MPSoC)

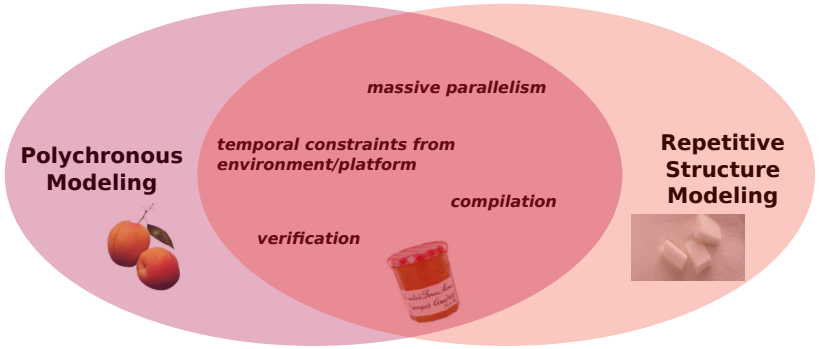




- Parallel and distributed implementations of applications
- Execution correctness, performance an energy-efficiency

Rational for high-level design approach

Fast, easy, costless and relevant design for multi-clock and data-intensive applications with high-level concepts



Outline

Polychrony and Signal programming

Clock Constraint Specification Language (CCSL)

Abstract clock-based design analysis for MPSoCs

Outline

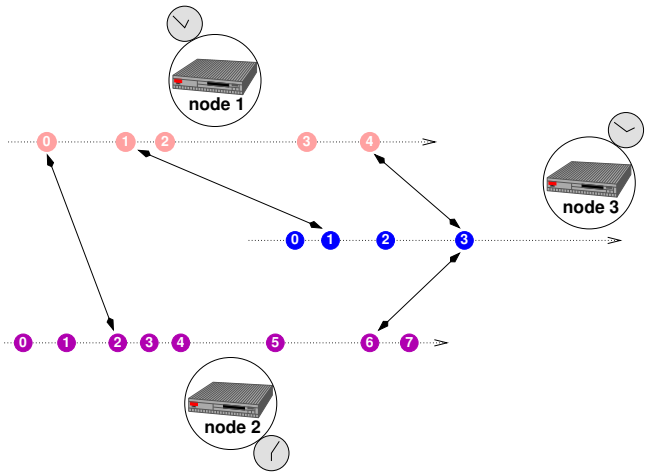
Polychrony and Signal programming

Clock Constraint Specification Language (CCSL)

Abstract clock-based design analysis for MPSoCs

Multi-clocked systems

Globally Asynchronous Locally Synchronous



Polychronous design

Non monolithic vision of system design

- No need to design *a priori* a global common clock: different activation clocks
- Partially ordered events
- Component-oriented (incremental) design
- Non determinism: beyond the system, its environment

Polychronous formalisms

Signal (around 1981)

- IRISA/Inria Rennes (France)
- Polychrony (`http://www.irisa.fr/espresso/Polychrony`)

Polychronous formalisms

Signal (around 1981)

- IRISA/Inria Rennes (France)
- Polychrony (`http://www.irisa.fr/espresso/Polychrony`)

Clock Constraint Specification Language – CCSL (around 2006)

- I3S/Inria Sophia-Antipolis (France)
- TimeSquare (`http://timesquare.inria.fr`)

Polychronous formalisms

Signal (around 1981)

- IRISA/Inria Rennes (France)
- Polychrony (`http://www.irisa.fr/espresso/Polychrony`)

Clock Constraint Specification Language – CCSL (around 2006)

- I3S/Inria Sophia-Antipolis (France)
- TimeSquare (`http://timesquare.inria.fr`)

Multi-Rate Instantaneous Channel connected Data Flow – MRICDF (around 2008)

- Virginia Tech (Blacksburg, USA)
- EmCodeSyn (`http://www.fermat.ece.vt.edu`)

Programming concepts of Signal

Polychrony at a glance

- A formal design model for multi-clocked systems such as GALS [Le Guernic et al.'03]: polychronous[1] model
- Logical time, as in other synchronous languages
- SIGNAL language
- `Polychrony` (academic) environment (`http://www.irisa.fr/espresso/Polychrony`)
  - GUI for modeling/specifying/programming in SIGNAL
  - Compiler
  - Connection to the Sigali tool
  - ...
- `RT-Builder` (commercial) environment of the Geensoft spin-off of Dassault Syst. (`http://www.geensoft.com`)

[1] From the Greek "poly chronos", meaning multiple clocks.

The SIGNAL language

Basic notions

- signal `x`: infinite series of typed values $(x_t)_{t\in\mathbb{N}}$

	t_0	t_1	t_2	t_3	t_4	t_5	...
x :	1	5	$\perp$	6	$\perp$	0	...

- absence of events: $\perp$
- constant signals: series of identical values
- usual data types: `boolean`, `integer`, `real`, `complex`, `char`, `string`, etc.
- pure events: `event` (sub-type of `boolean`)

The SIGNAL language (cont'd)

Basic notions

- signal `x`: infinite series of typed values $(x_t)_{t\in\mathbb{N}}$

	t_0	t_1	t_2	t_3	t_4	t_5	...
x :	1	5	$\perp$	6	$\perp$	0	...

The SIGNAL language (cont'd)

Basic notions

- signal `x`: infinite series of typed values $(x_t)_{t\in\mathbb{N}}$

	t_0	t_1	t_2	t_3	t_4	t_5	...
x :	1	5	$\perp$	6	$\perp$	0	...

- Abstract clock of a signal: $\hat{}$`x` (instants of presence)

The SIGNAL language (cont'd)

Basic notions

- signal x: infinite series of typed values $(x_t)_{t\in\mathbb{N}}$

	t_0	t_1	t_2	t_3	t_4	t_5	...
x :	1	5	$\perp$	6	$\perp$	0	...

- Abstract clock of a signal: $\hat{}$x (instants of presence)
- Synchronous signals: x $\hat{}=$ y (same clock)

The SIGNAL language (cont'd)

Basic notions

- signal x: infinite series of typed values $(x_t)_{t\in\mathbb{N}}$

	t_0	t_1	t_2	t_3	t_4	t_5	...
x :	1	5	$\perp$	6	$\perp$	0	...

- Abstract clock of a signal: $\hat{}$x (instants of presence)
- Synchronous signals: x $\hat{}=$ y (same clock)
- Process: system of equations expressing functional and temporal relations between signals.

Exercise...

Which scenarios denote synchronous signals?

x :	$\perp$	5	$\perp$	$\perp$	4	2	0	$\perp$	...
y :	$\perp$	$\perp$	5	$\perp$	$\perp$	4	2	0	...

x :	$\perp$	5	$\perp$	$\perp$	4	2	0	$\perp$	...
y :	$\perp$	5	$\perp$	$\perp$	4	2	0	$\perp$	...

x :	$\perp$	5	$\perp$	$\perp$	4	2	0	$\perp$	...
y :	$\perp$	3	$\perp$	$\perp$	1	9	8	$\perp$	...

x :	$\perp$	5	-7	$\perp$	4	2	0	$\perp$	...
y :	$\perp$	3	$\perp$	$\perp$	1	9	8	$\perp$	...

Basic operators on signals

- Instantaneous functions/relations: z := x+ y

$$(\forall t \in \mathbb{N})\ z_t = \begin{cases} \perp & \text{if } x_t = y_t = \perp \\ x_t + y_t & \text{else} \end{cases}$$

x :	$\perp$	5	$\perp$	$\perp$	4	2	0	$\perp$	...
y :	$\perp$	3	$\perp$	$\perp$	1	9	8	$\perp$	...
z :	$\perp$	8	$\perp$	$\perp$	5	11	8	$\perp$	...

Basic operators on signals

- Instantaneous functions/relations: `z := x+ y`

$$(\forall t \in \mathbb{N})\ z_t = \begin{cases} \bot & \text{if } x_t = y_t = \bot \\ x_t + y_t & \text{else} \end{cases}$$

x :	$\bot$	5	$\bot$	$\bot$	4	2	0	$\bot$	...
y :	$\bot$	3	$\bot$	$\bot$	1	9	8	$\bot$	...
z :	$\bot$	8	$\bot$	$\bot$	5	11	8	$\bot$	...

- Delay: `y := x $ 1 init 3.14`
  - $(\forall t \in \mathbb{N})\quad x_t = \bot \quad \Leftrightarrow y_t = \bot$
  - $(\exists t_i \in \mathbb{N})\quad x_{t_i} \neq \bot \quad \Rightarrow y_{t_0} = 3.14,\quad (\forall t_i > 0)\ y_{t_{i+1}} = x_{t_i}$

x :	1.7	$\bot$	2.5	$\bot$	$\bot$	6.5	2.4	...
y :	3.14	$\bot$	1.7	$\bot$	$\bot$	2.5	6.5	...

Basic operators on signals

- Undersampling: `y := x when b`
  - `Extracts elements from` $(x_t)_{t\in\mathbb{N}}$, `at instants where b` is *true*
  - $\widehat{\texttt{y}} = \widehat{\texttt{x}} \cap$ `[b]`

x :	5	$\bot$	4	8	7	3	$\bot$	...
b :	*tt*	*ff*	$\bot$	*ff*	*tt*	$\bot$	*tt*	...
y :	5	$\bot$	$\bot$	$\bot$	7	$\bot$	$\bot$	...

Basic operators on signals

- Undersampling: `y := x when b`
  - `Extracts elements from` $(x_t)_{t\in\mathbb{N}}$, `at instants where b` is *true*
  - $\widehat{\texttt{y}} = \widehat{\texttt{x}} \cap$ `[b]`

x :	5	$\bot$	4	8	7	3	$\bot$	...
b :	*tt*	*ff*	$\bot$	*ff*	*tt*	$\bot$	*tt*	...
y :	5	$\bot$	$\bot$	$\bot$	7	$\bot$	$\bot$	...

- Merging: `y := u default v`
  - Deterministic functional merging of series $(u_t)_{t\in\mathbb{N}}$ *and* $(v_t)_{t\in\mathbb{N}}$
  - $\widehat{\texttt{y}} = \widehat{\texttt{u}} \cup \widehat{\texttt{v}}$

u :	5	$\bot$	4	8	$\bot$	$\bot$	3	...
v :	1	7	$\bot$	2	$\bot$	0	1	...
y :	5	7	4	8	$\bot$	0	3	...

Exercises...

Discuss the correctness of the following statements:

1. A signal is necessarily present whenever it holds a value different from $\bot$.
2. A constant signal is always present.
3. When a signal becomes absent, it implicitly keeps its previously carried value.
4. A signal of `event` type and a signal of `boolean` type are exactly the same.
5. The abstract clock of a signal defines the set of instants at which the signal occurs.
6. SIGNAL assumes a reference clock that enables to always decide the presence/absence of any defined signal.

Exercises...

1. In the expression s_n := R(s_1,...,s_{n-1}) where R is an instantaneous relation, if the signal s_1 is absent while all other arguments of R are present, s_n is calculated by considering some default value depending on the type of s_1.
2. In the expression s_2 := s_1 $ 1 init c, the signal s_2 may occur with the latest value of s_1 while s_1 is absent.
3. In the expression s_3 := s_1 default s_2, the signals s_1 and s_2 must have exclusive clocks.
4. In the expression s_3 := s_1 or s_2,
   - s_3 is true when s_1 is true and s_2 is absent;
   - s_3 is true when s_1 is true and s_2 is false;
   - s_3 is false when s_1 is absent and s_2 is false;
   - s_3 is false when s_1 is absent and s_2 is absent.
   - s_3 is absent when s_1 is absent and s_2 is absent.

Basic operators on processes

- Composition: `u := x + y | z := u when b`
  - Union of systems of equations (*static single assignment*)
  - P and Q communicate via their common signals

x :	⊥	5	⊥	⊥	4	2	0	⊥	...
y :	⊥	3	⊥	⊥	1	9	9	⊥	...
u :	⊥	8	⊥	⊥	5	11	9	⊥	...
b :	*tt*	*ff*	⊥	*ff*	*tt*	⊥	*tt*	⊥	...
z :	⊥	⊥	⊥	⊥	5	⊥	9	⊥	...

Basic operators on processes

- Composition: `u := x + y | z := u when b`
  - Union of systems of equations (*static single assignment*)
  - P and Q communicate via their common signals

x :	⊥	5	⊥	⊥	4	2	0	⊥	...
y :	⊥	3	⊥	⊥	1	9	9	⊥	...
u :	⊥	8	⊥	⊥	5	11	9	⊥	...
b :	*tt*	*ff*	⊥	*ff*	*tt*	⊥	*tt*	⊥	...
z :	⊥	⊥	⊥	⊥	5	⊥	9	⊥	...

- Hiding (local declaration): `P where u`
  - Restricts the visibility scope of signal u to P

x :	⊥	5	⊥	⊥	4	2	0	⊥	...
y :	⊥	3	⊥	⊥	1	9	9	⊥	...
b :	*tt*	*ff*	⊥	*ff*	*tt*	⊥	*tt*	⊥	...
z :	⊥	⊥	⊥	⊥	5	⊥	9	⊥	...

Syntactic elements: program/process

```
process <IDENTIFIER> =
  %<SOME_COMMENTS>%
  {<STATIC_PARAMETERS>;}
  (? <INPUT_PARAMETERS>;
   ! <OUTPUT_PARAMETERS>; )
  (| <EQUATION_1>
   | ...
   | <EQUATION_K>
   | (| <EQUATION_K1>
      |  ...
      | <EQUATION_Kp>
      |)
   | <EQUATION_N>
   | ...
   |)
 where
    <LOCAL_DECLARATIONS>;
end;
```

Example: a counter modulo N

```
process COUNTER_N =
  {integer N;}
  (? event tick;
   ! integer cnt; )
 (| cnt ^= tick
  | cnt := (0 when reset) default cnt_pre + 1
  | cnt_pre := cnt $ 1 init 0
  | reset := true when (cnt_pre = N-1)
  |)
 where
   integer cnt_pre; event reset;
end;%COUNTER_N%
```

tick :	*tt*	*tt*	*tt*	⊥	*tt*	*tt*	⊥	*tt*	*tt*	...
cnt :	1	2	0	⊥	1	2	⊥	0	1	...
cnt_pre :	0	1	2	⊥	0	1	⊥	2	0	...

Exercises...

1. Define a process `Sum` in which on each occurrence of its unique input `i` of `real` type, the sum of occurred values of `i` until now is computed in the output signal `s`.
2. Define a process `Average` in which on each occurrence of its unique input `i` of `real` type, the average of occurred values of `i` until now is computed in the output signal `avg`.

Exercises... (cont'd)

1. Given a constant parameter `N` of `integer` type, define a process `AverageN` in which on each occurrence of its unique input `i` of `real` type, the average of the `N` previous values[2] of `i` from now is computed in the output signal `avg`.
2. Let `i` be a positive `integer` signal. Define a signal `max` in a process `Maximum`, which takes at the current logical instant the most recent maximum value held by `i`.

[2] **Note**: `i$N` may be initialized to 0.0 by using the following expression: `init[to N:0.0]`.

Clock-oriented specifications

Extended constructs for clock manipulation

- Set operations on clocks
  - Empty/null clock: ^0
  - Union (upper bound): x1 ^+ x2
  - Intersection (lower bound): x1 ^* x2
  - Relative complement (difference): x1 ^- x2

Clock-oriented specifications

Extended constructs for clock manipulation

- Set operations on clocks
  - Empty/null clock: ^0
  - Union (upper bound): x1 ^+ x2
  - Intersection (lower bound): x1 ^* x2
  - Relative complement (difference): x1 ^- x2
- Clock comparison
  - Inferiority: x1 ^< x2
  - Superiority: x1 ^> x2
  - Exclusion: x1 ^# x2
  - Equality: x1 ^= x2
- Set of instants at which a condition holds: `when c`

Exercise...

- Let `s1` and `s2` be two signals of any type and independent from each other. Define a signal `present` of `event` type that occurs whenever `s1` and `s2` are present at the same time.
- What about the following expression as a solution:

  `^s1 and ^s2`?

Static analysis and compilation in Polychrony

Program analysis with a compiler

Usual compilation functionality: syntax, type, etc.

Clock calculus based on a Boolean abstraction

- clock exclusion: mutual exclusion
- empty clock: absence of reaction, or undesired events
- clock hierarchy (inclusion): structuring of statements
- synthesis of a master clock from a program: determinism

(Clocked-) Data-dependency analysis

- absence of dependency cycles (deadlocked behavior)

Clock analysis

What happens with the following program?

```
process P1 =
  ( ? integer s1;
    ! integer s2, s3, s4; )
  (| s2 := s1 when (s1 > 0)
   | s3 := s1 when not (s1 > 0)
   | s4 := s2 + s3
   |);
```

Clock analysis

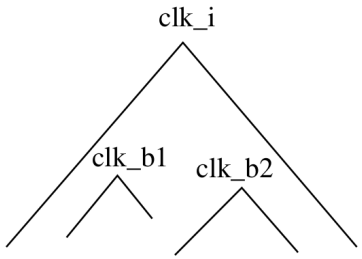
What happens with the following program[3]?

```
process P2 =
  ( ? dreal s2, s4;
    ! dreal s1, s3; )
  (| s3 := sin(s1) + s2
   | s1 := s4 default s3
   |);
```

[3] `dreal` type denotes real double precision

Automatic code generation

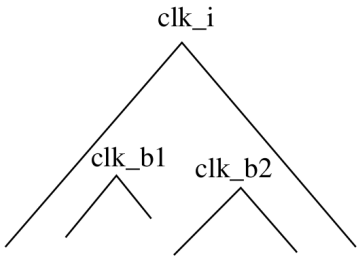
Clock inclusion hierarchy for efficient control-flow (beyond data-dependency analysis)




```
if (clk_i)
  { ...;
    if (clk_b1)
      { ... };
    if (clk_b2)
      { ... };
  }
```

Automatic code generation

Clock inclusion hierarchy for efficient control-flow (beyond data-dependency analysis)

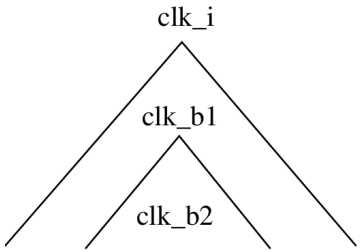



```
if (clk_i)
  { ...;
    if (clk_b1)
      { ... };
    if (clk_b2)
      { ... };
  }
```

Endochronous versus exochronous programs

Automatic code generation

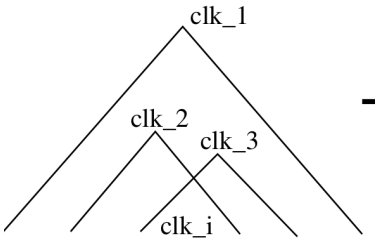
Clock inclusion hierarchy for efficient control-flow (beyond data-dependency analysis)




```
if (clk_i)
  { ...;
   if (clk_b1)
     { ...;
      if (clk_b2)
        { ... };
     }
  }
```

Automatic code generation

Clock inclusion hierarchy for efficient control-flow (beyond data-dependency analysis)




```
if (clk_1)
  { ...;
    if (clk_2)
      { ... };
    if (clk_3)
      { ... };
    if (clk_2 && clk_3)
      { if (clk_i)
          { ... };
          ...;
      }...;
  }
```

Data dependency analysis

What happens with the following program?

```
process P3 =
  ( ? dreal s2, s4;
    ! dreal s1, s3; )
  (| s3 := sin(s1) + cos(s2)
   | tmp := s3 / 3.14
   | s1 := tmp * tmp
   |)
 where
  dreal tmp;
end;
```

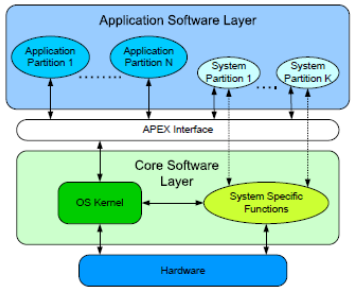
A demo on C code generation?

Other facilities of Polychrony

Design libraries and model-checking

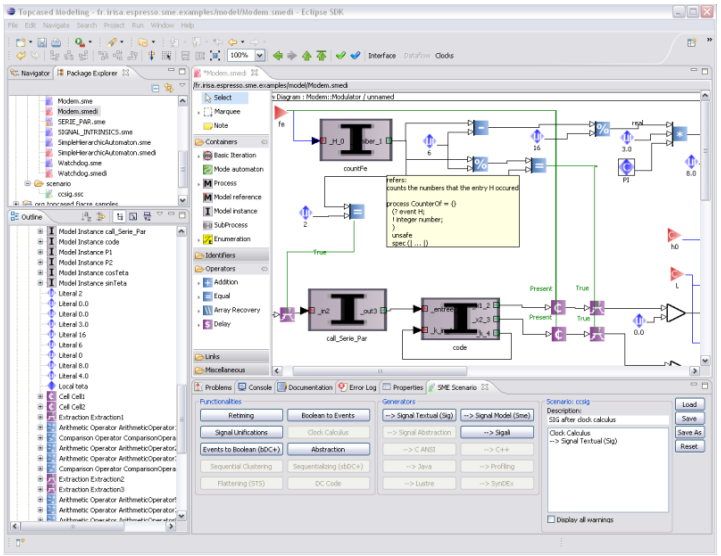
Sigali tool: verification (model-checking) of reactive systems and discrete controller synthesis (`http://www.irisa.fr/vertecs/Softwares/sigali.html`)

APEX-ARINC 653 library: for integrated modular avionics



Avionic application design, real-time java code re-engineering

Environment front-end



Bibliographic notes

- Paul Le Guernic, Jean-Pierre Talpin, and Jean-Christophe Le Lann. Polychrony for system design. Journal for Circuits, Systems and Computers, Special Issue on Application Specific Hardware Design, World Scientific, April 2003
- A Gamatié "Designing Embedded Systems with the SIGNAL Programming Language", (260 pages). Springer NY editor, 2010.
- Paul Le Guernic and Thierry Gautier. Data-Flow to von Neumann: the Signal approach. In Advanced Topics in Data-Flow Computing, J.-L. Gaudiot and L. Bic, Eds, Prentice-Hall, 1991, 413-438

Outline

Clock Constraint Specification Language: basic notions

Clock Constraint Specification Language

UML/Marte standard modeling profile: Modeling and Analysis of Real-Time and Embedded systems

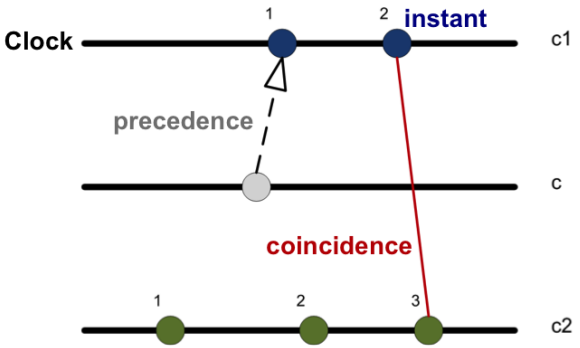
Semantic issues related to time: CCSL brings a formal foundation, with suitable expressivity

Temporal behavior modeling and reasoning

- causal relations between events
- polychronous
- partially ordered events

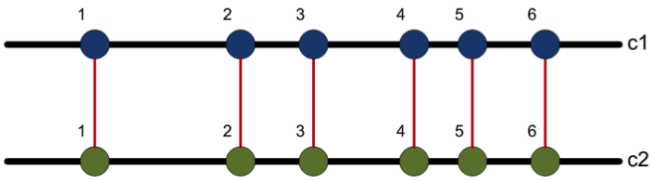
CCSL

Basic notons: logical clocks and instant relations



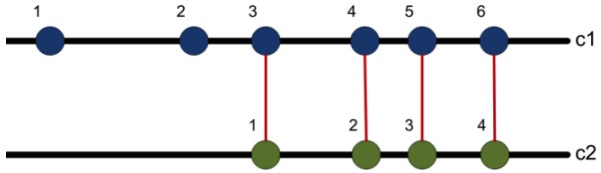
CCSL: clock relations

Coincidence relation: c1 equals to c2



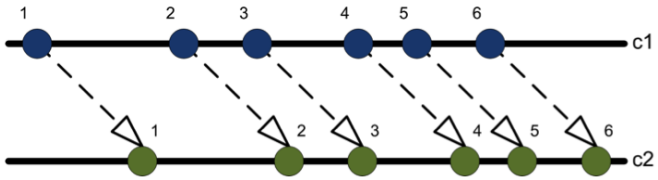
CCSL: clock relations

Clock containment: c2 is a sub-clock of c1



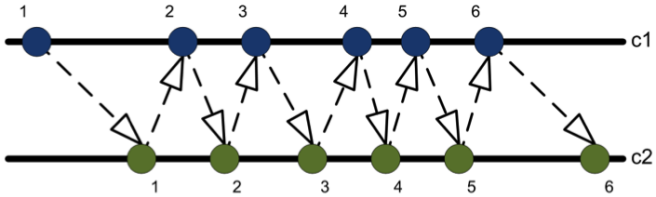
CCSL: clock relations

Infinitely many precedence relations: c2 precedes c1



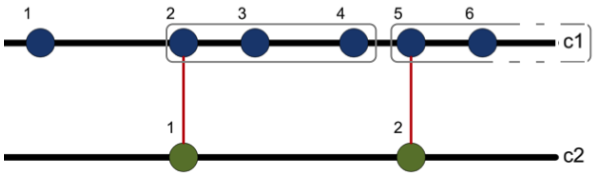
CCSL: clock relations

Instant alternation



CCSL: clock relations

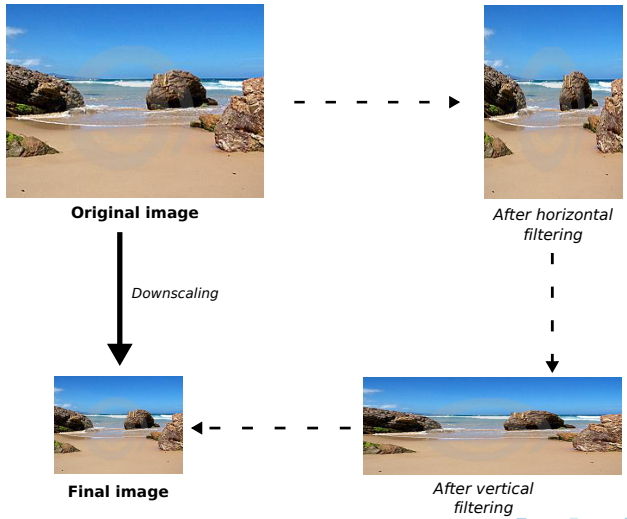
Periodicity: c2 is periodic on c1



Example of specification

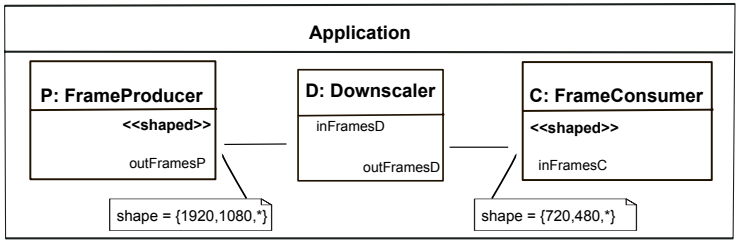
Downscaling transformation

Intuitive description



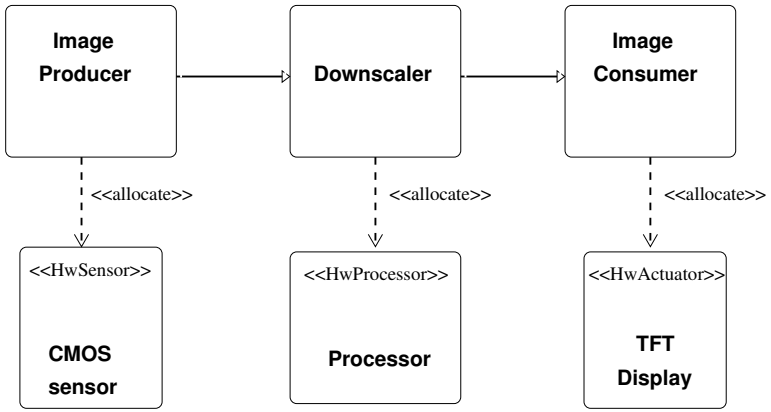
A downscaling application model

Functional specification



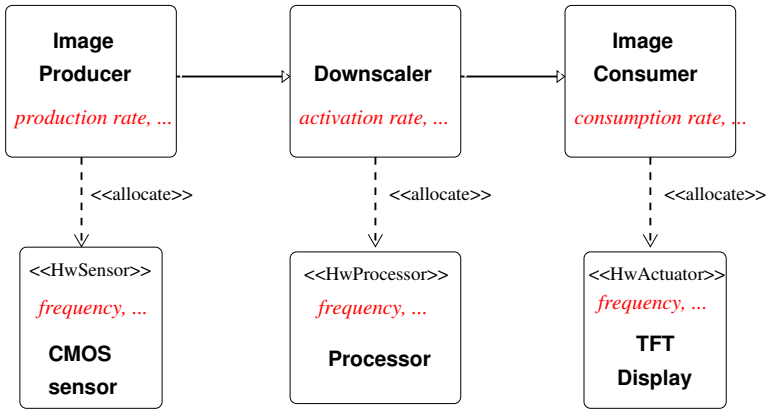
Mapped downscaling application

Allocation specification on a platform



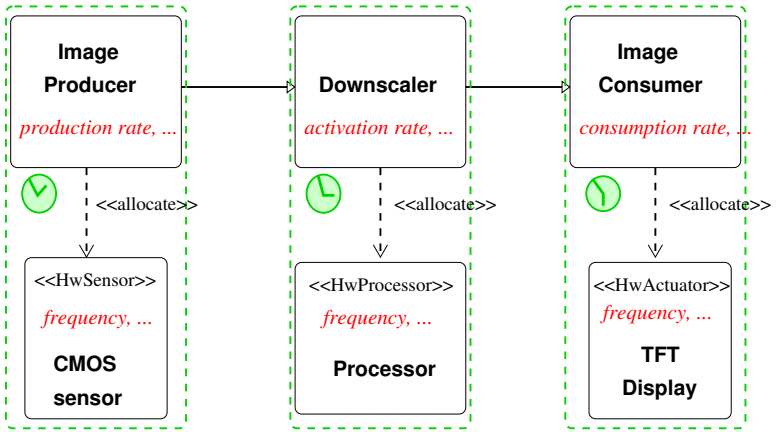
Mapped downscaling application

Frequency and rate constraint specification



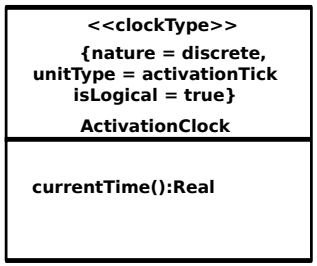
Mapped downscaling application

Capture by abstract clock relations

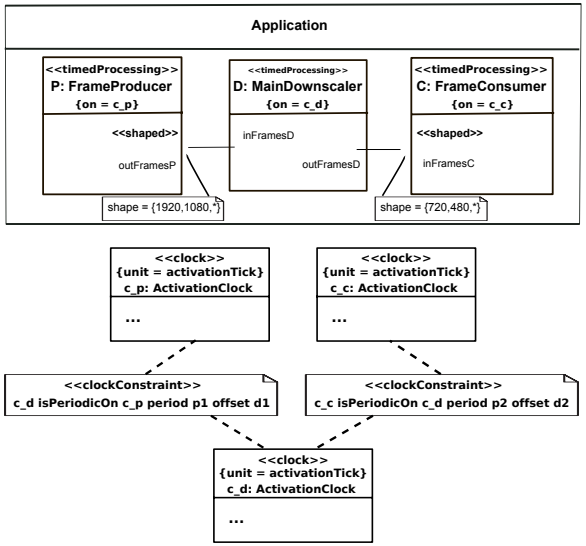


Basic clock specification

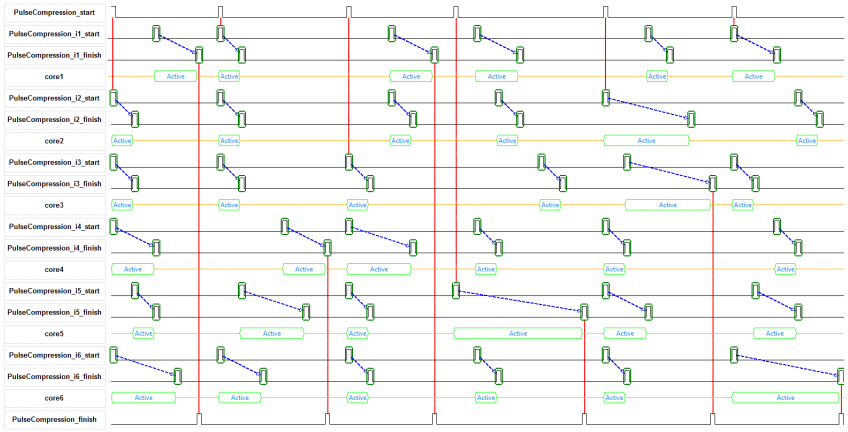
Definition of a clock type, named "ActivationClock"



Clock constraints for downscaling



Design analysis with TimeSquare



Bibliographic notes

- Charles André. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). Rapport de recherche INRIA, No 6925, 2009.
- Charles André, Frédéric Mallet, Robert de Simone. Modeling Time(s). In 10th Int. Conf on Model Driven Engineering Languages and Systems (MODELS '07), LNCS, Pages 559-573, Nashville, TN, USA, Septembre 2007.
- Calin Glitia, Julien Deantoni, Frédéric Mallet, Jean-Vivien Millo, Pierre Boulet, Abdoulaye Gamatié. Progressive and explicit refinement of scheduling for multidimensional data-flow applications using uml marte. Design Automation for Embedded Systems, Springer, 2012, 16 (2), pp. 137-169.

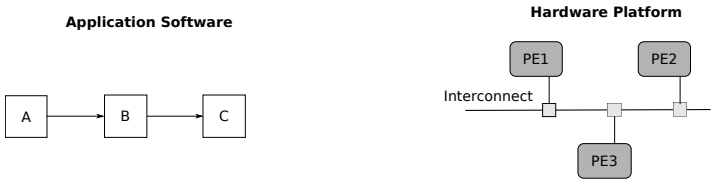
Outline

MPSoC design challenges

Cost-effective and safe design for *adaptive* MPSoCs

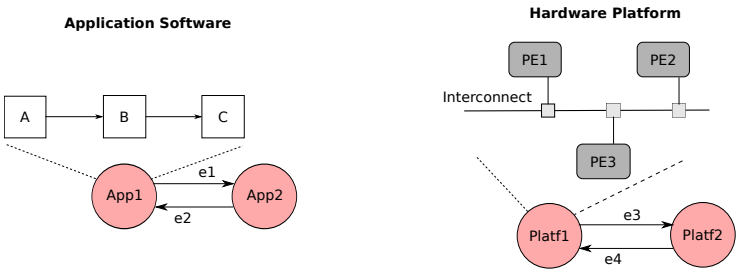
MPSoC design challenges

Cost-effective and safe design for *adaptive* MPSoCs



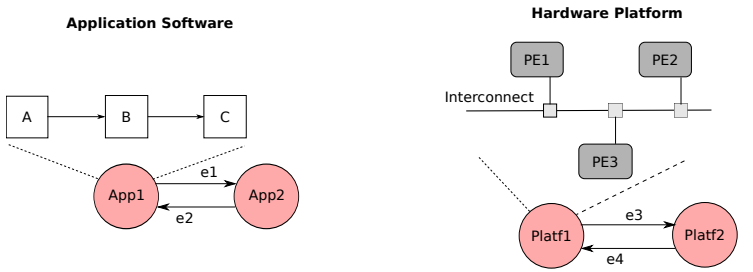
MPSoC design challenges

Cost-effective and safe design for *adaptive* MPSoCs



MPSoC design challenges

Cost-effective and safe design for *adaptive* MPSoCs

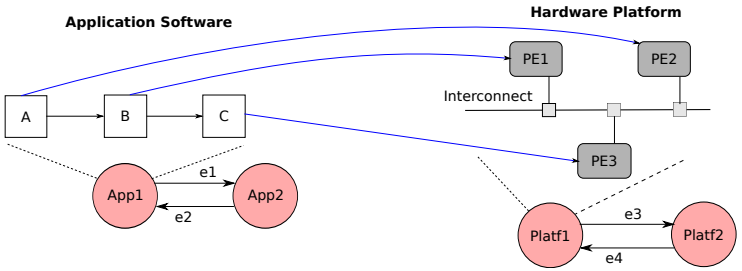


Design Issue for adaptive MPSoCs

Identifying **efficient application-architecture mappings**, w.r.t. adaptive behaviors: frequencies, task migration...

MPSoC design challenges

Cost-effective and safe design for *adaptive* MPSoCs

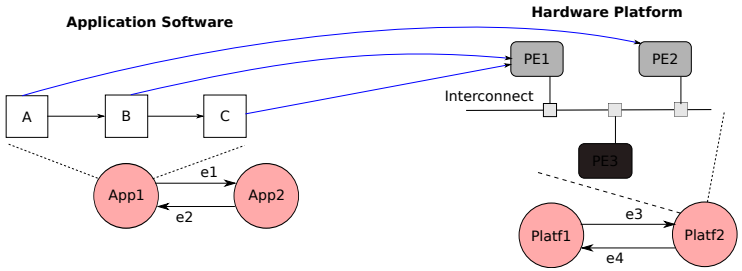


Design Issue for adaptive MPSoCs

Identifying **efficient application-architecture mappings**, w.r.t. adaptive behaviors: frequencies, task migration...

MPSoC design challenges

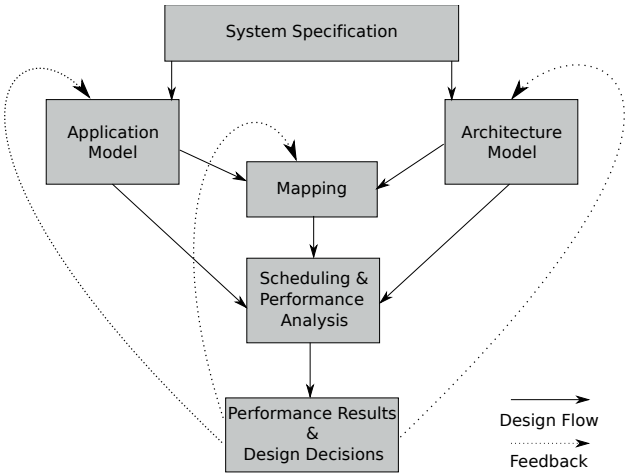
Cost-effective and safe design for *adaptive* MPSoCs



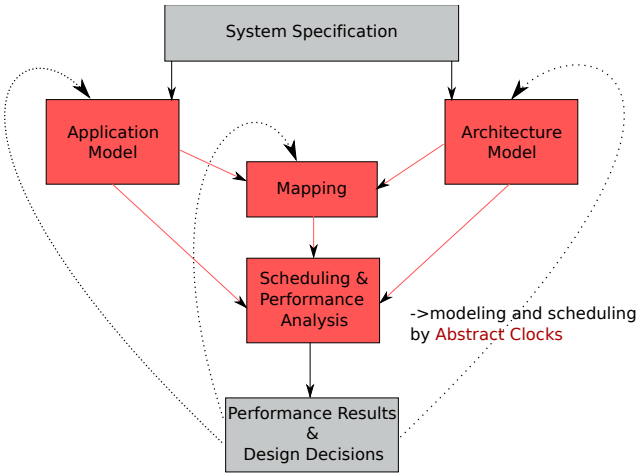
Design Issue for adaptive MPSoCs

Identifying **efficient application-architecture mappings**, w.r.t. adaptive behaviors: frequencies, task migration...

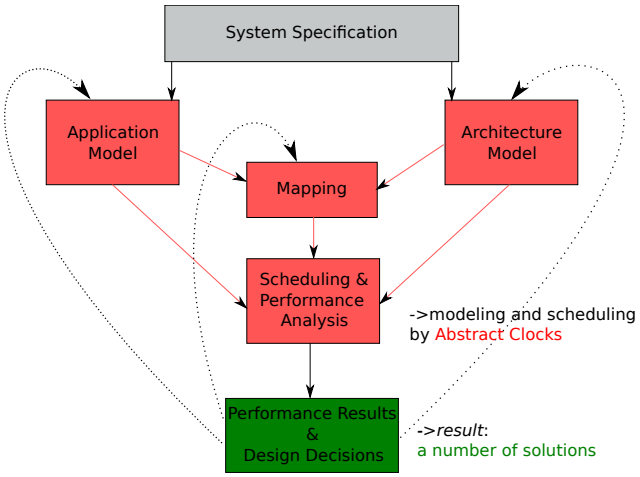
Starting point: Y-chart



Starting point: Y-chart (focus)



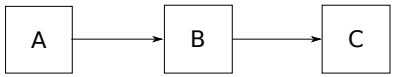
Starting point: Y-chart (focus)



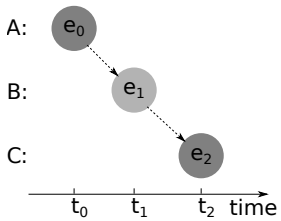
CLASSY DESIGN ANALYSIS FRAMEWORK

Application behavior

Application specification

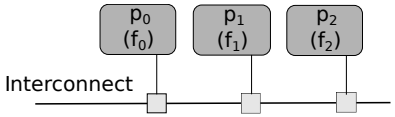


Clock modeling: events: e_0, e_1, e_2



Architecture behavior

Architecture specification



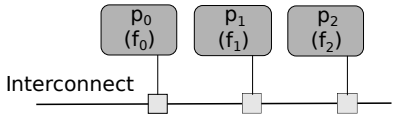
- $f_0 = 300, f_1 = 200, f_2 = 150$;
- frequency of reference clock $\mathcal{K}$: $LCM(f_0, f_1, f_2) = 600$.

Clock modeling

	0	1	2	3	4	5	6	7	8	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	...
p_0										
p_1										
p_2										

Architecture behavior

Architecture specification



- $f_0 = 300, f_1 = 200, f_2 = 150$;
- frequency of reference clock $\mathcal{K}$: $LCM(f_0, f_1, f_2) = 600$.

Clock modeling

	0	1	2	3	4	5	6	7	8	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	...
p_0	•		•		•		•		•	...
p_1										
p_2										

Architecture behavior

Architecture specification

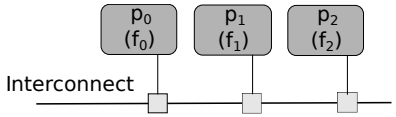
- $f_0 = 300, f_1 = 200, f_2 = 150$;
- frequency of reference clock $\mathcal{K}$: $LCM(f_0, f_1, f_2) = 600$.

Clock modeling

	0	1	2	3	4	5	6	7	8	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	...
p_0	•		•		•		•		•	...
p_1	•			•			•			...
p_2										

Architecture behavior

Architecture specification



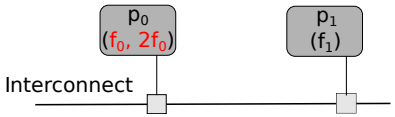
- $f_0 = 300, f_1 = 200, f_2 = 150$;
- frequency of reference clock $\mathcal{K}$: $LCM(f_0, f_1, f_2) = 600$.

Clock modeling

	0	1	2	3	4	5	6	7	8	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	...
p_0	•		•		•		•		•	...
p_1	•			•			•			...
p_2	•				•				•	...

Adaptive architecture behavior

Architecture specification



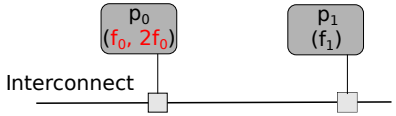
- instant 4: $p_0 : f_0 \rightarrow 2f_0$, **adaptation penalty**: one cycle.

Clock modeling

	0	1	2	3	4	5	6	7	8	9	10	11	12	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•	•	•	•	...
p_0														
p_1														

Adaptive architecture behavior

Architecture specification



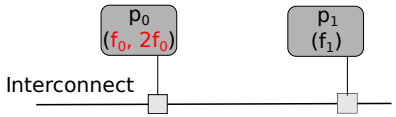
- instant 4: $p_0 : f_0 \rightarrow 2f_0$, **adaptation penalty**: one cycle.

Clock modeling

	0	1	2	3	4	5	6	7	8	9	10	11	12	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•	•	•	•	...
p_0	•		•		•									
p_1	•			•										

Adaptive architecture behavior

Architecture specification



- instant 4: $p_0 : f_0 \rightarrow 2f_0$, **adaptation penalty**: one cycle.

Clock modeling

	0	1	2	3	4	5	6	7	8	9	10	11	12	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•	•	•	•	...
p_0	•		•		•		•							
p_1	•			•			•							

Adaptive architecture behavior

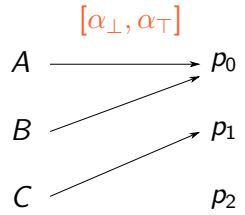
Architecture specification

- instant 4: $p_0 : f_0 \rightarrow 2f_0$, **adaptation penalty**: one cycle.

Clock modeling

	0	1	2	3	4	5	6	7	8	9	10	11	12	...
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•	•	•	•	...
p_0	•		•		•		•							...
p_1	•			•			•							...

Mapping of application on platform



- **Function** $T \rightarrow P$;
- **Elementary costs** α, i.e., *time* and *energy* costs;
- **Best and Worst Case costs** $[\alpha_\perp, \alpha_\top]$.

Abstract clock modeling of scheduling

- $A = \{e_0, e_1\}$ with costs $[2, 2]$ and $[1, 1]$ cycles;

	0	1	2	3	4	5	6	7	8	9
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•
p_0	•		•		•		•		•	
$\mathbf{clk(A/p_0)}$										

- **event executions**: 1 followed by -1;

Abstract clock modeling of scheduling

- $A = \{e_0, e_1\}$ with costs $[2, 2]$ and $[1, 1]$ cycles;

	0	1	2	3	4	5	6	7	8	9
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•
p_0	•		•		•		•		•	
$\mathbf{clk(A/p_0)}$	1	-1	-1	-1						

- **event executions**: 1 followed by -1;

Abstract clock modeling of scheduling

- $A = \{e_0, e_1\}$ with costs $[2, 2]$ and $[1, 1]$ cycles;

	0	1	2	3	4	5	6	7	8	9
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•
p_0	•		•		•		•		•	
$\mathbf{clk(A/p_0)}$	1	-1	-1	-1			1	-1		

- **event executions**: 1 followed by -1;

Abstract clock modeling of scheduling

- $A = \{e_0, e_1\}$ with costs $[2, 2]$ and $[1, 1]$ cycles;

	0	1	2	3	4	5	6	7	8	9
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•
p_0	•		•		•		•		•	
$\mathbf{clk(A/p_0)}$	1	-1	-1	-1	**0**	**-1**	1	-1		

- **event executions**: 1 followed by -1;
- **non-execution**: 0 followed by -1.

Abstract clock modeling of scheduling

- $A = \{e_0, e_1\}$ with costs $[2, 2]$ and $[1, 1]$ cycles;

	0	1	2	3	4	5	6	7	8	9
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•
p_0	•		•		•		•		•	
$\mathbf{clk(A/p_0)}$	1	-1	-1	-1	**0**	**-1**	1	-1		

- **event executions**: 1 followed by -1;
- **non-execution**: 0 followed by -1.
- As Soon As Possible scheduling preserving *precedence relations*.

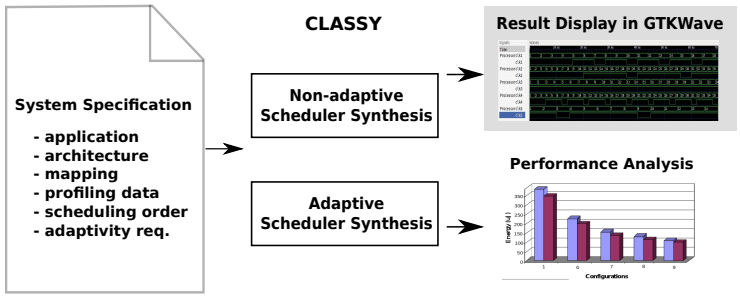
Performance analysis

	0	1	2	3	4	5	6	7	8	9	10	11	12
$\mathcal{K}$	•	•	•	•	•	•	•	•	•	•	•	•	•
p_0	•		•		•		•		•		•		•
$clk(A/p_0)$	**1**	**-1**	0	-1	**1**	**-1**							
p_1	•			•			•			•			•
$clk(B/p_1)$	0	-1	-1	**1**	**-1**	**-1**	**1**	**-1**	**-1**				

- execution time: the longest clock, e.g., $9/f_{\mathcal{K}}$;
- usage ratio of PEs: **busy cycles**/overall cycles, e.g., $p_0 : 2/3$;
- energy consumption of PEs: energy costs for (running tasks + being idle)

Design space exploration: CLASSY Tool

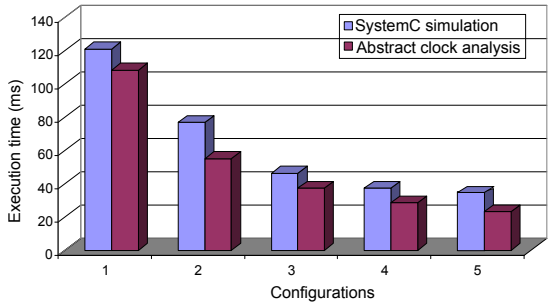
`http://www.lirmm.fr/~gamatie/pages/Tool/Classy.html`



- Exhaustive and heuristic-based **exploration methods**
- Pareto-optimal mapping and platform config. solutions w.r.t. **time** and **energy**

Experimental results: CLASSY vs SoCLib

Performance analysis[4] on JPEG encoder

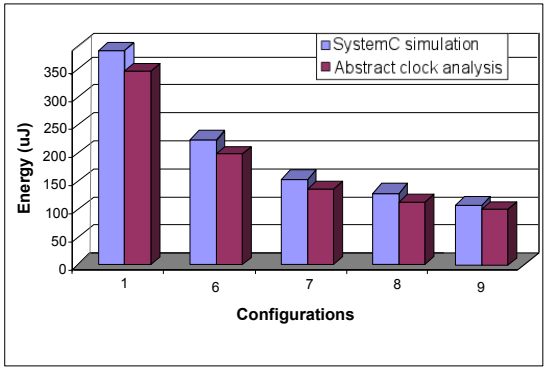


- less precise but similar observation tendency.

[4] Assuming deadlock-free communications.

Experimental results: CLASSY vs SoCLib

Energy consumption analysis[5] on JPEG encoder



[5]Assuming deadlock-free communications.

Summary

Abstract clock-based approaches for

- programming with Signal
- reasoning about temporal properties
- design space exploration for MPSoCs

Bibliographic notes

- Xin An, Sarra Boumedien, Abdoulaye Gamatie and Eric Rutten "CLASSY: a Clock Analysis System for Rapid Prototyping of Embedded Applications on MPSoCs", 15th International Workshop on Software and Compilers for Embedded Systems, - SCOPES'2012, Schloss Rheinfels, St. Goar, Germany, May 15-16, 2012. ACM Press.
- Abdoulaye Gamatie "Design of Streaming Applications on MPSoCs using Abstract Clocks", Design, Automation and Test in Europe - DATE'2012, Dresden, Germany, March 2012.
- Adolf Abdallah, Abdoulaye Gamatié, Rabie Ben Atitallah and Jean-Luc Dekeyser. 'Abstract Clock-based Design of a JPEG Encoder', IEEE Embedded System Letters, vol 4, n. 2, June 2012.

THE END...