

Static Analysis of Signal Programs for Efficient Design of Multi-Clocked Embedded Systems

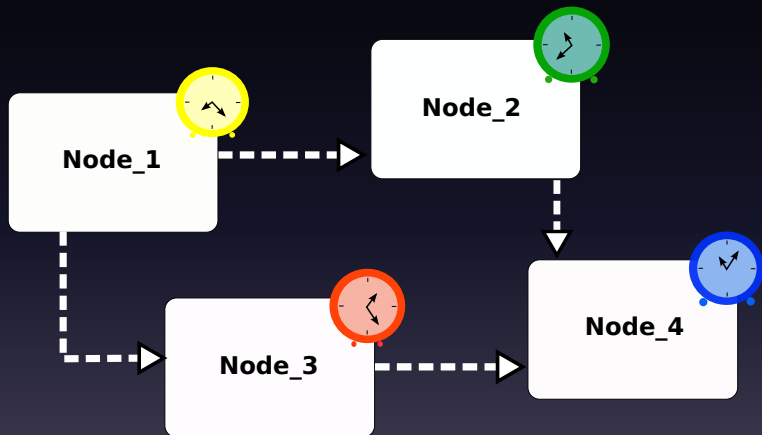
Abdoulaye Gamatié

Laure Gonnord

CNRS/LIFL and INRIA, France

April 2011

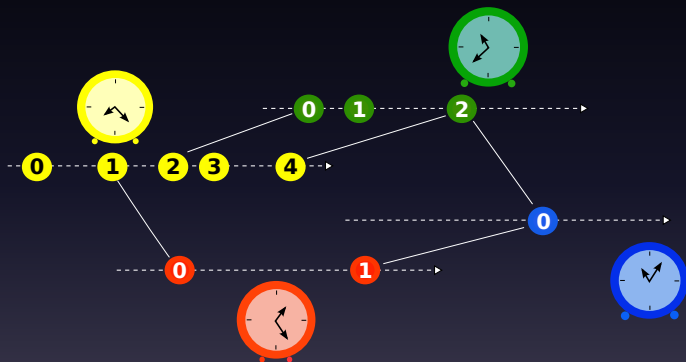
Multi-clocked embedded systems



Examples: GALS systems, MPSoCs

Multi-clocked system modeling

Node interactions specified using abstract clock relations



Examples: Clock Constraint Specification Language (CCSL), Multi-Rate Instantaneous Channel connected Data Flow (MRICDF), **Signal**

In this talk...

- 1 Synchronous language Signal
 - main concepts
 - compilation: static analysis & code generation
- 2 A solution for improving the compilation
 - a new abstraction for programs
 - illustration and implementation
- 3 Concluding remarks

Signal language

Basic notions

- **signal** x : sequence $(x_{t_i})_{t_i \in \mathbb{N}}$ of typed values (\perp is absence)
- **clock** \widehat{x} of a signal x : instants where values $\neq \perp$
- **process** $y := x + 1$: relations between values/clocks of signals

<i>time</i>	:	t_0	t_1	t_2	t_3	t_4	t_5	...
x	:	1	5	\perp	6	\perp	0	...
\widehat{x}	:	<i>tt</i>	<i>tt</i>	<i>ff</i>	<i>tt</i>	<i>ff</i>	<i>tt</i>	...
y	:	2	6	\perp	7	\perp	1	...

Primitive operators on signals

Synchronous operators: signals have the same clock

- Relations. $y := f(x_1, \dots, x_n)$

Example: $\text{level} := \text{pre_level} - \text{pump}$

pre_level	:	\perp	0.3	\perp	\perp	-2.7	5	1	...
pump	:	\perp	3	\perp	\perp	-7.7	4	0.5	...
level	:	\perp	-2.7	\perp	\perp	5	1	0.5	...

Primitive operators on signals

Synchronous operators: signals have the same clock

- **Relations.** $y := f(x_1, \dots, x_n)$

Example: $\text{level} := \text{pre_level} - \text{pump}$

pre_level	:	\perp	0.3	\perp	\perp	-2.7	5	1	...
pump	:	\perp	3	\perp	\perp	-7.7	4	0.5	...
level	:	\perp	-2.7	\perp	\perp	5	1	0.5	...

- **Delay.** $y := x \$ 1 \text{ init } c$

Example: $\text{pre_level} := \text{level} \$ 1 \text{ init } 0.3$

level	:	\perp	-2.7	\perp	\perp	5	1	0.5	...
pre_level	:	\perp	0.3	\perp	\perp	-2.7	5	1	...

Primitive operators on signals

Multi-clock operators: signals may have different clocks

- Sampling. $y := x$ when b

Example: $\text{alarm} := \text{empty}$ when $(0 \geq \text{level})$

empty	:	5	\perp	4	8	7	3	\perp	...
level	:	-1	5	\perp	3	-9	\perp	\perp	...
$(0 \geq \text{level})$:	<i>tt</i>	<i>ff</i>	\perp	<i>ff</i>	<i>tt</i>	\perp	\perp	...
alarm	:	5	\perp	\perp	\perp	7	\perp	\perp	...

Primitive operators on signals

Multi-clock operators: signals may have different clocks

- **Sampling.** $y := x$ when b

Example: $\text{alarm} := \text{empty}$ when $(0 \geq \text{level})$

empty	:	5	\perp	4	8	7	3	\perp	...
level	:	-1	5	\perp	3	-9	\perp	\perp	...
$(0 \geq \text{level})$:	<i>tt</i>	<i>ff</i>	\perp	<i>ff</i>	<i>tt</i>	\perp	\perp	...
alarm	:	5	\perp	\perp	\perp	7	\perp	\perp	...

- **Merging.** $z := x$ default y

Example: $\text{global_alarm} := \text{scarce}$ default overflow

scarce	:	<i>ff</i>	\perp	<i>tt</i>	<i>tt</i>	\perp	\perp	\perp	...
overflow	:	<i>tt</i>	<i>tt</i>	\perp	<i>ff</i>	\perp	<i>ff</i>	\perp	...
global_alarm	:	<i>ff</i>	<i>tt</i>	<i>tt</i>	<i>tt</i>	\perp	<i>tt</i>	\perp	...

Primitive operators on processes

- Composition: $P_1 \mid P_2$

Example: $(! \text{scarce} := (0 \geq \text{level}) \mid \text{alarm} := \text{empty when scarce} !)$

empty	:	5	\perp	4	8	7	3	\perp	...
level	:	-1	5	\perp	3	-9	\perp	\perp	...
scarce	:	<i>tt</i>	<i>ff</i>	\perp	<i>ff</i>	<i>tt</i>	\perp	\perp	...
alarm	:	5	\perp	\perp	\perp	7	\perp	\perp	...

Primitive operators on processes

- Composition: $P_1 \mid P_2$

Example: $(! \text{scarce} := (0 \geq \text{level}) \mid \text{alarm} := \text{empty when scarce } !)$

empty	:	5	\perp	4	8	7	3	\perp	...
level	:	-1	5	\perp	3	-9	\perp	\perp	...
scarce	:	<i>tt</i>	<i>ff</i>	\perp	<i>ff</i>	<i>tt</i>	\perp	\perp	...
alarm	:	5	\perp	\perp	\perp	7	\perp	\perp	...

- Local declaration: $P \text{ where } u$

Example: $(! \text{scarce} := (0 \geq \text{level}) \mid \text{alarm} := \text{empty when scarce } !)$ where scarce

empty	:	5	\perp	4	8	7	3	\perp	...
level	:	-1	5	\perp	3	-9	\perp	\perp	...
alarm	:	5	\perp	\perp	\perp	7	\perp	\perp	...

Compilation of Signal programs

Signal programs



Compiler

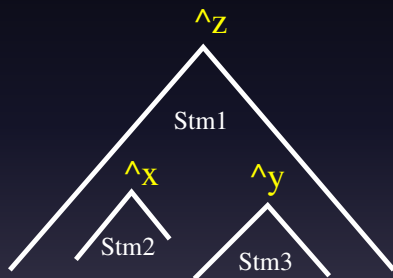
- *syntax & type analysis*
- *data dependency analysis*
 - *clock analysis*
 - *clock hierarchization for code generation*



**Code in general-purpose languages,
e.g. C, C++, Java**

Compilation of Signal programs (2)

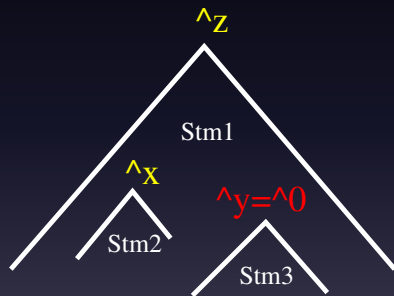
Hierarchization of clocks for code generation



```
if (clk_z)
{ Stm1;
  if (clk_x)
    { Stm2; };
  if (clk_y)
    { Stm3; };
}
```

Compilation of Signal programs (3)

Analysis of clock constraints for better results



```
if (clk_z)
{ Stm1;
  if (clk_x)
  { Stm2; };
  if (clk_y)
  { Stm3; };
}
```

Compilation of Signal programs (4)

Boolean abstraction for clock analysis

Example:

```
(| scarce := (0 >= level)
 | overflow := (level > 7)
 | alarm := (true when scarce) when overflow
 |)
```

$$\left\{ \begin{array}{l} \widehat{\text{scarce}} \Leftrightarrow \uparrow \text{level} \\ \text{scarce} \Leftrightarrow (0 \geq \text{level}) \end{array} \right.$$

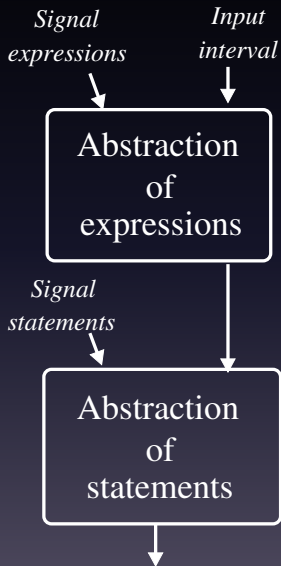
$$\left\{ \begin{array}{l} \widehat{\text{overflow}} \Leftrightarrow \uparrow \text{level} \\ \text{overflow} \Leftrightarrow (\text{level} > 7) \end{array} \right.$$

$$\left\{ \begin{array}{l} \widehat{\text{alarm}} \Leftrightarrow [\text{scarce}] \wedge [\text{overflow}] \\ \widehat{\text{scarce}} \Leftrightarrow [\text{scarce}] \vee [\neg \text{scarce}] \\ \text{false} \Leftrightarrow [\text{scarce}] \wedge [\neg \text{scarce}] \\ \widehat{\text{overflow}} \Leftrightarrow [\text{overflow}] \vee [\neg \text{overflow}] \\ \text{false} \Leftrightarrow [\text{overflow}] \wedge [\neg \text{overflow}] \\ \text{alarm} \Leftrightarrow [\text{scarce}] \wedge [\text{overflow}] \end{array} \right.$$

Numerical expressions not fully addressed in abstraction:

$\widehat{\text{alarm}} \Leftrightarrow \text{false}$ is not detected!

A new abstraction for Signal



given variation intervals of input signals $x_i \in X_P$ of a process P

$$\phi(b1 \text{ and } b2) = b_1 \wedge b_2$$

$$\phi(e1 + e2) = I_{e_1} \tilde{+} I_{e_2}$$

...

first order logic formula $\Phi(P)$ as a set of valuations $v = (\hat{\cdot}, \tilde{\cdot})$:

$$\hat{\cdot} : X_P \rightarrow \{true, false\} \quad (\text{clock})$$

$$\tilde{\cdot} : X_P \rightarrow \mathbb{R} \cup \{true, false\} \quad (\text{value})$$

Abstraction of statement behaviors

Examples (y is numeric)

- Relations

$$\Phi(y:=f(x_1, \dots, x_n)) = \bigwedge_{i=1}^n (\hat{y} \Leftrightarrow \hat{x}_i) \wedge (\hat{y} \Rightarrow \tilde{y} \in \phi(f(x_1, \dots, x_n)))$$

$$\{v \mid v = \langle x_1 \mapsto (\mathbf{ff}, x_{11}), \dots, x_n \mapsto (\mathbf{ff}, x_{n1}), y \mapsto (\mathbf{ff}, y_1) \rangle \text{ or} \\ v = \langle x_1 \mapsto (\mathbf{tt}, x_{11} \in I_{x_1}), \dots, x_n \mapsto (\mathbf{tt}, x_{n1} \in I_{x_n}), y \mapsto (\mathbf{tt}, y_1 \in \tilde{f}(I_{x_1} \dots)) \rangle\}$$

- Sampling

$$\Phi(y := x \text{ when } b) = (\hat{y} \Leftrightarrow (\hat{x} \wedge \hat{b} \wedge \tilde{b})) \bigwedge (\hat{y} \Rightarrow \tilde{y} = \tilde{x})$$

- Composition

$$\Phi(P_1 | P_2) = \Phi(P_1) \wedge \Phi(P_2)$$

Concretization of a formula Φ

- Set of events according to all valuations $v \models \Phi$:

E.g., $v = \langle x_1 \mapsto (true, x_{11}), x_2 \mapsto (false, x_{21}) \dots x_n \mapsto (true, x_{n1}) \rangle$

	v	v'	v''
	\Downarrow	\Downarrow	\Downarrow
x_1 :	x_{11}	\perp	x_{12}
x_2 :	\perp	x_{21}	x_{22}
...			
x_n :	x_{n1}	x_{n2}	x_{n3}

Concretization of a formula Φ (2)

- $\Gamma(\Phi)$ = set of all possible traces built from valid events:

$$T_1 = \begin{array}{l} \mathbf{x1} : \quad \perp \quad \dots \quad x_{11} \quad \dots \\ \mathbf{x2} : \quad x_{21} \quad \dots \quad \perp \quad \dots \\ \dots \\ \mathbf{xn} : \quad v_{n2} \quad \dots \quad v_{n1} \quad \dots \end{array}$$

$$T_2 = \begin{array}{l} \mathbf{x1} : \quad x_{11} \quad x_{12} \quad \perp \quad \dots \\ \mathbf{x2} : \quad \perp \quad x_{22} \quad x_{21} \quad \dots \\ \dots \\ \mathbf{xn} : \quad x_{n1} \quad x_{n3} \quad x_{n2} \quad \dots \end{array}$$

Abstraction soundness

Proposition: *Given a process P^1 and a formula φ ,*

if $\Phi_P \Rightarrow \varphi$, then $\llbracket P \rrbracket \subseteq \Gamma(\varphi)$ — — P satisfies φ

Yices SMT solver is used to check $\Phi_P \Rightarrow \varphi$

¹ $\llbracket P \rrbracket$ denotes the set of all possible traces satisfying P .

A simple example

Example 2 (cont'd): let program P be:

```
(| scarce := (0 >= level)
 | overflow := (level > 7)
 | alarm := (true when scarce) when overflow
 |)
```

To verify that alarm never occurs when P executes, we consider

- 1 the abstraction $\Phi(P)$ yields

$$\Phi_P = \begin{cases} (\widehat{\text{overflow}} \Leftrightarrow \widehat{\text{level}} \Leftrightarrow \widehat{\text{scarce}}) \\ \wedge (\widetilde{\text{scarce}} \Leftrightarrow (\widehat{\text{level}} \in]-\infty, 0]) \\ \wedge (\widetilde{\text{overflow}} \Leftrightarrow (\widehat{\text{level}} \in]7, +\infty[)) \\ \dots \end{cases}$$

- 2 a property φ of interest

$$\varphi = \neg((\widetilde{\text{scarce}} \wedge \widehat{\text{scarce}}) \wedge (\widetilde{\text{overflow}} \wedge \widehat{\text{overflow}}))$$

A simple example (2)

Since $\Phi(P)$ implies φ , we define $P' = P \mid \Gamma(\varphi)$ as:

```
(| (| scarce := (0 >= level)
  | overflow := (level > 7)
  | alarm := true when scarce when overflow
  |)
| (| true when scarce when overflow ^= ^0 |)
|)
```

Thanks to [jpsc03], $\llbracket P' \rrbracket = \llbracket P \rrbracket$: numerical properties abstracted away by Boolean abstraction of P are made explicit in P' now!

Another example: Bathtub

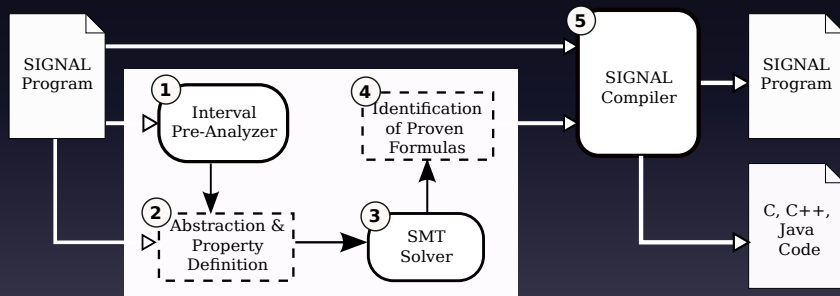
Description and analysis of a Bathtub (see paper)

- process Bathtub: 18 lines in Signal
- Identified clock properties: P1, P2, P3

Resulting clock analysis and code generation

<i>process</i>	<i>clock issues</i>	<i>C code (#lines)</i>
Bathtub	clock constraints	88
Bathtub P1	one null clock solved	78
Bathtub P1 P2 P3	five null clocks solved	35

Overview of the solution



Concluding remarks

Improvement of static analysis for multi-clock designs in Signal

- an expressive abstraction (Boolean and numeric parts of programs)
- efficient clock consistency analysis
- optimized automatic code generation
- related works: SAT [fac04] and IDD [ecbs08] for Signal, SMT [fmcad08] for Lustre, Polyhedra abstract inter. [sas99] for synchronous languages

Next steps...

- benchmarks and integration in Signal design environment (Polychrony)
- Signal encoding of formula concretizations

Thanks!

In this talk...

- 1 Synchronous language Signal
 - main concepts
 - compilation: static analysis & code generation
- 2 A solution for improving the compilation
 - a new abstraction for programs
 - illustration and implementation
- 3 Concluding remarks