Static Analysis of Signal Programs for Efficient Design of Multi-Clocked Embedded Systems
work published at LCTES 2011

Abdoulaye Gamatié and Laure Gonnord

LIFL/CNRS/INRIA

Synchron, November 2011
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 ($\bot$ is absence)
- **clock** $\hat{x}$ of a signal $x$ : instants where values $\neq \bot$
- **process** $y := x+1$ : relations between values/clocks of signals

<table>
<thead>
<tr>
<th>time</th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1</td>
<td>5</td>
<td>$\bot$</td>
<td>6</td>
<td>$\bot$</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>$\hat{x}$</td>
<td>tt</td>
<td>tt</td>
<td>ff</td>
<td>tt</td>
<td>ff</td>
<td>tt</td>
<td>...</td>
</tr>
<tr>
<td>$y$</td>
<td>2</td>
<td>6</td>
<td>$\bot$</td>
<td>7</td>
<td>$\bot$</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>
Primitive operators on signals

Synchronous operators: signals have the same clock

- Relations: \( y := f(x_1, \ldots, x_n) \)

Example: \( \text{level} := \text{pre\_level} - \text{pump} \)

<table>
<thead>
<tr>
<th>pre_level</th>
<th>( \perp )</th>
<th>0.3</th>
<th>( \perp )</th>
<th>( \perp )</th>
<th>5</th>
<th>1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>pump</td>
<td>( \perp )</td>
<td>3</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>4</td>
<td>0.5</td>
<td>...</td>
</tr>
<tr>
<td>level</td>
<td>( \perp )</td>
<td>-2.7</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>1</td>
<td>0.5</td>
<td>...</td>
</tr>
</tbody>
</table>
Primitive operators on signals

Synchronous operators: signals have the same clock

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

Example: \( \text{level} := \text{pre}_\text{level} - \text{pump} \)

<table>
<thead>
<tr>
<th>Signal</th>
<th>0.3</th>
<th>3</th>
<th>-2.7</th>
<th>5</th>
<th>1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{pre}_\text{level}</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
<td>-2.7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>\text{pump}</td>
<td>\bot</td>
<td>3</td>
<td>\bot</td>
<td>\bot</td>
<td>-7.7</td>
<td>4</td>
</tr>
<tr>
<td>\text{level}</td>
<td>\bot</td>
<td>-2.7</td>
<td>\bot</td>
<td>\bot</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

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

Example: \( \text{pre}_\text{level} := \text{level} \; $\; 1 \; \text{init} \; 0.3 \)

<table>
<thead>
<tr>
<th>Signal</th>
<th>-2.7</th>
<th>5</th>
<th>1</th>
<th>0.5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{level}</td>
<td>\bot</td>
<td>5</td>
<td>1</td>
<td>0.5</td>
<td>...</td>
</tr>
<tr>
<td>\text{pre}_\text{level}</td>
<td>\bot</td>
<td>\bot</td>
<td>\bot</td>
<td>-2.7</td>
<td>5</td>
</tr>
</tbody>
</table>
Primitive operators on signals
Multi-clock operators: signals may have different clocks

- **Sampling.** $y := x \text{ when } b$

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

<table>
<thead>
<tr>
<th>empty</th>
<th>5</th>
<th>⊥</th>
<th>4</th>
<th>8</th>
<th>7</th>
<th>3</th>
<th>⊥</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>-1</td>
<td>5</td>
<td>⊥</td>
<td>3</td>
<td>-9</td>
<td>⊥</td>
<td>⊥</td>
<td>...</td>
</tr>
<tr>
<td>(0 &gt;= level)</td>
<td>$tt$</td>
<td>$ff$</td>
<td>⊥</td>
<td>$ff$</td>
<td>$tt$</td>
<td>⊥</td>
<td>⊥</td>
<td>...</td>
</tr>
<tr>
<td>alarm</td>
<td>5</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>7</td>
<td>⊥</td>
<td>⊥</td>
<td>...</td>
</tr>
</tbody>
</table>
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 | ⊥ | 4 | 8 | 7 | 3 | ⊥ |...
  | level   | −1| 5 | ⊥ | 3 | −9| ⊥ | ⊥ | ...
  | (0 >=level) | tt | ff | ⊥ | ff | tt | ⊥ | ⊥ | ...
  | alarm   | 5 | ⊥ | ⊥ | ⊥ | 7 | ⊥ | ⊥ | ...

- **Merging.** \( z := x \) default \( y \)

  Example: \( \text{global_alarm} := \text{scarce} \) default \( \text{overflow} \)

  | scarce  | ff | ⊥ | tt | tt | ⊥ | ⊥ | ⊥ |...
  | overflow | tt | tt | ⊥ | ff | ⊥ | ff | ⊥ | ...
  | global_alarm | ff | tt | tt | tt | ⊥ | ff | ⊥ |...
Primitive operators on processes

- Composition: $P_1 \mid P_2$

Example: ($\mid$ scarce := (0 >= level) $\mid$ alarm := empty when scarce $\mid$)

| empty  | 5 | ⊥ | 4 | 8 | 7 | 3 | ⊥ | ...
|--------|---|---|---|---|---|---|---|---
| level  | −1| 5 | ⊥ | 3 | −9| ⊥ | ⊥ | ...
| scarce | tt| ff| ⊥ | ff| tt| ⊥ | ⊥ | ...
| alarm  | 5 | ⊥ | ⊥ | ⊥ | 7 | ⊥ | ⊥ | ...
Primitive operators on processes

- **Composition**: $P_1 \ | \ P_2$

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

| empty   | 5 | ⊥ | 4 | 8 | 7 | 3 | ⊥ | ...
| level   | -1 | 5 | ⊥ | 3 | -9 | ⊥ | ⊥ | ...
| scarce  | tt | ff | ⊥ | ff | tt | ⊥ | ⊥ | ...
| alarm   | 5 | ⊥ | ⊥ | ⊥ | 7 | ⊥ | ⊥ | ...

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

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

| empty   | 5 | ⊥ | 4 | 8 | 7 | 3 | ⊥ | ...
| level   | -1 | 5 | ⊥ | 3 | -9 | ⊥ | ⊥ | ...
| alarm   | 5 | ⊥ | ⊥ | ⊥ | 7 | ⊥ | ⊥ | ...
Compilation of Signal programs

Signal Programs

Compiler

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

Code in general-purpose language (C ...)

Gamatié/Gonnord (LIFL/CNRS/INRIA) Improving the Static Analysis of Signal Programs 8/23 Nov 2011
Compilation of Signal programs (2)

Clock hierarchy and code generation, improving the final result:

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

[^x]
[^z]
[^alarm]
Compilation of Signal programs (2)

Clock hierarchy and code generation, improving the final result:

\[
\text{if (clk\_z) } \{
\text{Stm1;}
\text{if (clk\_x) } \{
\text{Stm2;}
\}
\text{if (clk\_alarm) } \{
\text{Stm3;}
\}
\}
\]
Compilation of Signal programs (3)

Boolean abstraction for clock analysis

\[
\begin{align*}
\text{scarce} & : (0 \geq \text{level}) \\
\text{overflow} & : (\text{level} > 7) \\
\text{alarm} & : (\text{true when scarce}) \text{ when overflow}
\end{align*}
\]

\[
\begin{align*}
\hat{\text{scarce}} & \iff \hat{\text{level}} \\
\text{scarce} & \iff (0 \geq \text{level}) \\
\hat{\text{overflow}} & \iff \hat{\text{level}} \\
\text{overflow} & \iff (\text{level} > 7)
\end{align*}
\]

Numerical expressions not fully addressed in abstraction:

\[
\hat{\text{alarm}} \iff \text{false} \text{ is not detected!}
\]
A new abstraction for Signal

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

\[
\begin{align*}
\phi(b_1 \text{ and } b_2) &= b_1 \land b_2 \\
\phi(e_1 + e_2) &= I_{e_1} + I_{e_2}
\end{align*}
\]

... 

first order logic formula \( \Phi(P) \) as a set of valuations \( \nu = (\hat{\_, \_, \_}) \):

\[
\begin{align*}
\hat{\_} & : X_P \to \{\text{true, false}\} \quad (\text{clock}) \\
\tilde{\_} & : X_P \to \mathbb{R} \cup \{\text{true, false}\} \quad (\text{value})
\end{align*}
\]
Abstraction of statement behaviors - 1/2

(y is numeric, similar rules for booleans.)

- Relations

\[ \Phi(y := f(x_1, \ldots, x_n)) = \bigwedge_{i=1}^{n} (\hat{y} \Leftrightarrow \hat{x}_i) \land (\hat{y} \Rightarrow \tilde{y} \in \phi(f(x_1, \ldots, x_n))) \]

\{ v \mid v = \langle x_1 \mapsto (ff, x_{11}), \ldots, x_n \mapsto (ff, x_{n1}), y \mapsto (ff, y_1) \rangle \text{ or } v = \langle x_1 \mapsto (tt, x_{11} \in I_{x1}), \ldots, x_n \mapsto (tt, x_{n1} \in I_{xn}), y \mapsto (tt, y_1 \in \tilde{f}(I_{x1} \ldots)) \rangle \}

- Delay

\[ \Phi(y := x \; \$ \; 1 \; \text{init} \; c) = (\hat{y} \Leftrightarrow \hat{x}) \land (\hat{y} \Rightarrow (\tilde{y} = \tilde{x} \lor \tilde{y} = c)) \]

- Sampling

\[ \Phi(y := x \; \text{when} \; b) = (\hat{y} \Leftrightarrow (\hat{x} \land \hat{b} \land \tilde{b})) \land (\hat{y} \Rightarrow \tilde{y} = \tilde{x}) \]
Merging

\[ \Phi(y := x \text{ default } z) = \left( \hat{y} \iff (\hat{x} \lor \hat{z}) \right) \land \]

\[ \left( \hat{y} \Rightarrow ((\hat{x} \land (\hat{y} = \hat{x})) \lor (\neg \hat{x} \land (\hat{y} = \hat{z}))) \right) \]

Local signals

\[ \Phi(P \text{ where } x) = \exists \hat{x}, \exists \hat{x} . \Phi_P \]

Composition

\[ \Phi(P_1 | P_2) = \Phi(P_1) \land \Phi(P_2) \]
Concretization of a formula $\Phi$

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

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

<table>
<thead>
<tr>
<th>$v$</th>
<th>$v'$</th>
<th>$v''$</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
</tbody>
</table>

- $x_1 : x_{11}$, $\bot$, $x_{12}$
- $x_2 : \bot$, $x_{21}$, $x_{22}$
- $\ldots$
- $x_n : x_{n1}$, $x_{n2}$, $x_{n3}$

$\Gamma(\Phi) =$ set of all possible traces built from valid events.
Abstraction soundness

**Proposition**: Given a process $P^1$ and a formula $\varphi$,

if $\Phi_P \Rightarrow \varphi$, then $[P] \subseteq \Gamma(\varphi)$  --  $P$ satisfies $\varphi$

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

---

1. $[P]$ denotes the set of all possible traces satisfying $P$. 
A simple example

Let $P$ be:

\[
\begin{align*}
| \text{scarce} &:= (0 \geq \text{level}) \\
| \text{overflow} &:= (\text{level} > 7) \\
| \text{alarm} &:= (\text{true when scarce}) \text{ when overflow}
\end{align*}
\]

To verify that $\text{alarm}$ never occurs when $P$ executes, we consider

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

\[
\begin{align*}
\left\{ \\
(\neg \text{overflow} \iff \neg \text{level} \iff \neg \text{scarce}) \\
\land (\neg \text{scarce} \iff (\text{level} \in ] - \infty, 0]) \\
\land (\neg \text{overflow} \iff (\text{level} \in ] 7, +\infty[)) \\
\neg \text{overflow}) \\
\land (\neg \text{alarm} \iff (\neg \text{scarce} \land \neg \text{scarce} \land \neg \text{overflow} \land \neg \text{overflow}))
\end{align*}
\]

2. a property $\varphi = \neg (\neg \text{alarm})$
A simple example (2)

Recall $\varphi = \neg(\neg\hat{alarm})$.

- Yices gives: $\Phi(p) \Rightarrow \varphi$.
- (def of $\Gamma$) All traces in $\Gamma(\varphi)$ satisfy:

$$\forall t, alarm_t = \bot$$

Let $P'$ be:

$$(\mid (\mid \text{scarce} := (0 \geq \text{level}) \\
\mid \text{overflow} := (\text{level} > 7) \\
\mid \text{alarm} := \text{true when scarce when overflow} \\
\mid ))$$

Thanks to [jcsc03], $\llbracket P' \rrbracket = \llbracket P \rrbracket$ : 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

<table>
<thead>
<tr>
<th>process</th>
<th>clock issues</th>
<th>C code (#lines)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bathtub</td>
<td>clock constraints</td>
<td>88</td>
</tr>
<tr>
<td>Bathtub</td>
<td>P1</td>
<td>one null clock solved</td>
</tr>
<tr>
<td>Bathtub</td>
<td>P1</td>
<td>P2</td>
</tr>
</tbody>
</table>
Overview of the solution of the paper

1. Interval Pre-Analyzer
2. Abstraction & Property Definition
3. SMT Solver
4. Identification of Proven Formulas
5. SIGNAL Compiler

SIGNAL Program → Interval Pre-Analyzer → Abstraction & Property Definition → SMT Solver → Identification of Proven Formulas → SIGNAL Compiler → SIGNAL Program

C, C++, Java Code
Identifying properties and their abstraction - WIP

Idea:
- do not use SMT-solver as blackbox anymore.
- use a SMT-simplifier!

Given $\Phi$, an SMT solver/simplifier:
- construct a list of disjuncts subformulas.
- is able to detect for each subformula the list of false monoms.

▶ Get the list of false monoms for all subformulas (intersection)
On the example

\begin{verbatim}
| | scarce := (0 >= level)
| | overflow := (level > 7)
| | alarm := (true when scarce) when overflow
| |
\end{verbatim}

14 models, only alarm is false in every model.
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)
- find more interesting properties in the SMT outputs.
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