# On Timed-Automata semantics for ARTS

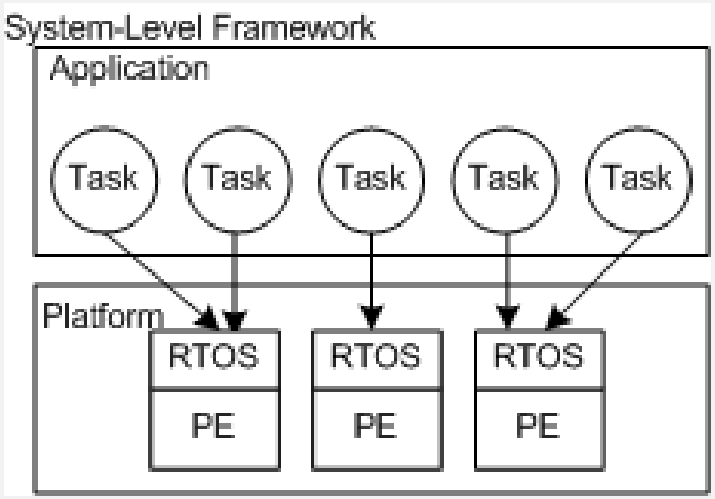
**Jan Madsen, Michael R. Hansen, Aske Brekling**

Informatics and Mathematical Modelling

Technical University of Denmark

# The ARTS modelling framework

ARTS: A system-level MPSoC simulation framework

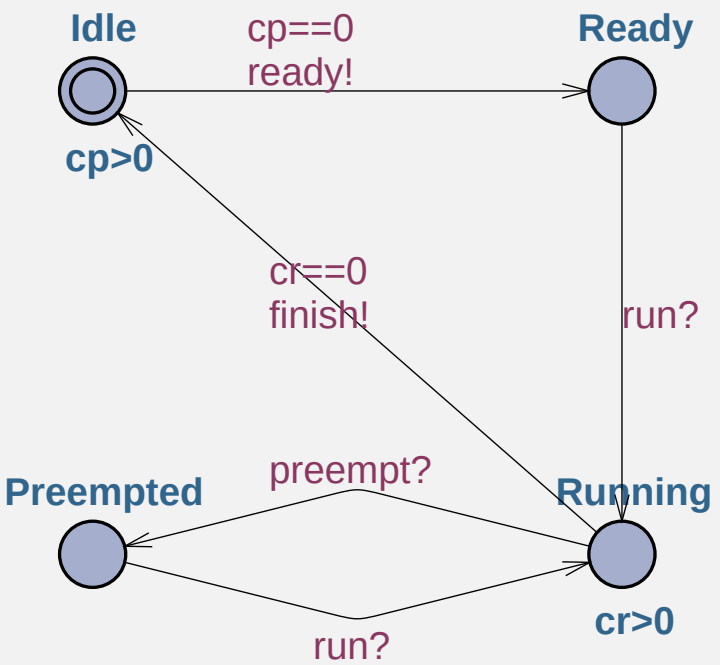



- Properties of the solution depends on the design of the application as well as the configuration of the platform.

design space exploration

# A simplified task in ARTS

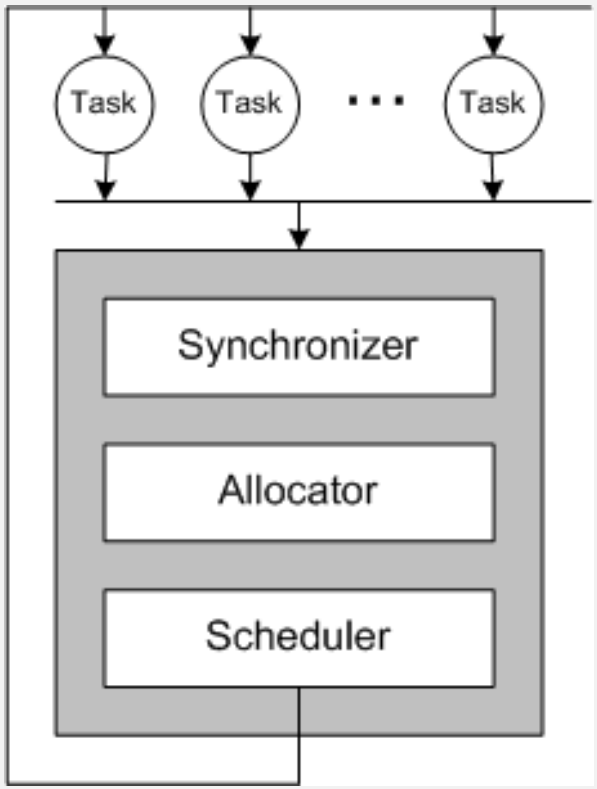
- Simple periodic tasks, which can be preempted




- cp, cr: time-dependent variables for the period and run time

semi-formal framework: much implicit information

# ARTS model of a processing element




- Synchronizer — resolves dependencies before task selection
- Allocator — grants access to shared resources (busses, memories, ...)
- Scheduler — grants processor time to tasks according to some scheduling discipline

- Components are informally described and implemented in SystemC
- ARTS is a simulation framework only

# Timed-Automata models for ARTS

First set of goals:

- Precise definition of components and their communication, which can be communicated and discussed.
- Simulation (using UPPAAL) on a formal basis.
- Verification on a formal basis.

# Timed-Automata models for ARTS

First set of goals:

- Precise definition of components and their communication, which can be communicated and discussed.
- Simulation (using UPPAAL) on a formal basis.
- Verification on a formal basis.

An ARTS system is modelled as follows:

$$
\begin{aligned}
\mathit{System} &= \mathit{Application} \parallel \mathit{Platform} \\
\mathit{Application} &= \parallel_{i=1}^{n} \mathit{Task}_i \\
\mathit{Task}_i &= \mathit{TaskControl}_i \parallel \mathit{Idle}_i \parallel \mathit{Ready}_i \parallel \mathit{Running}_i \\
\mathit{Platform} &= \parallel_{j=1}^{m} \mathit{PE}_j \\
\mathit{PE}_j &= \mathit{Controller}_j \parallel \mathit{Synchronizer}_j \parallel \mathit{Scheduler}_j
\end{aligned}
$$

# Timed-Automata models for ARTS

First set of goals:

- Precise definition of components and their communication, which can be communicated and discussed.
- Simulation (using UPPAAL) on a formal basis.
- Verification on a formal basis.

An ARTS system is modelled as follows:

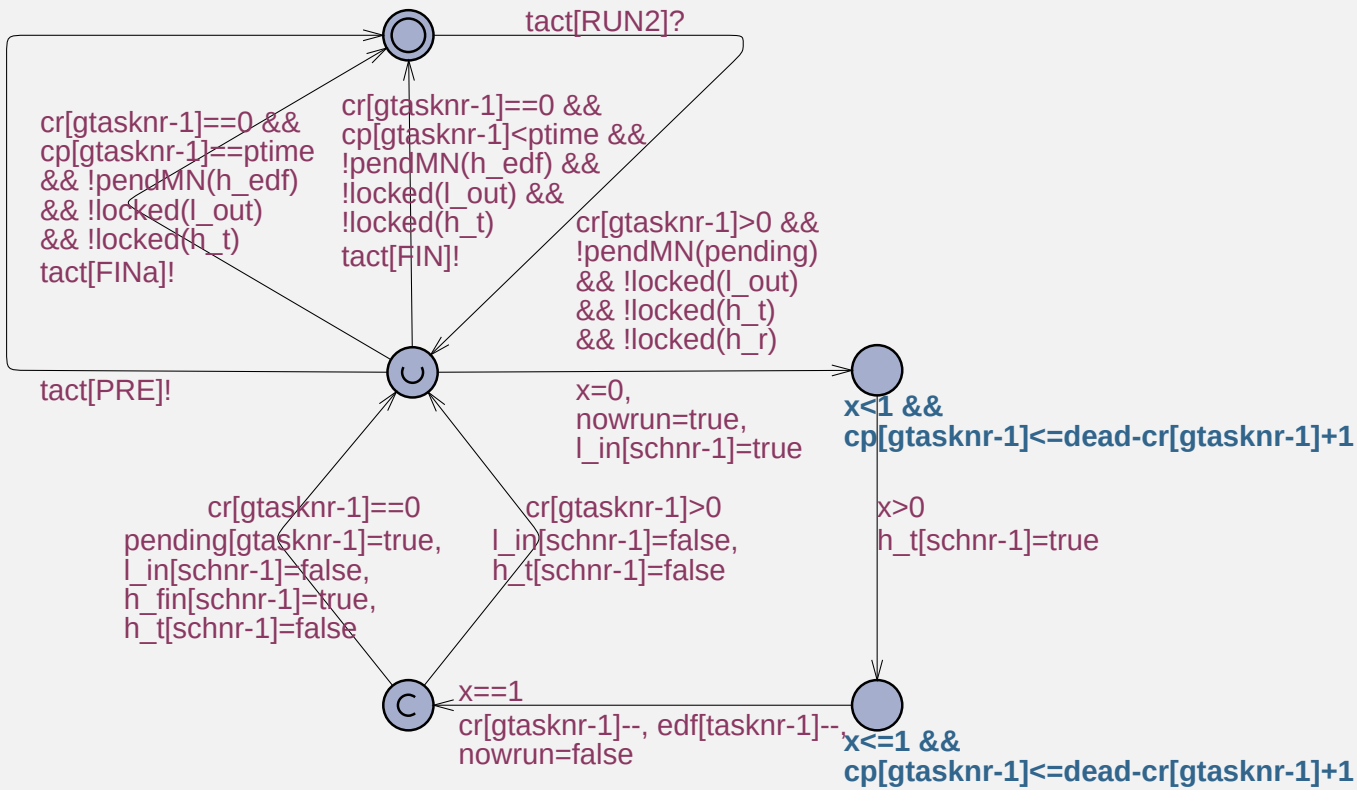
$$
\begin{aligned}
\mathit{System} &= \mathit{Application} \parallel \mathit{Platform} \\
\mathit{Application} &= \parallel_{i=1}^{n} \mathit{Task}_i \\
\mathit{Task}_i &= \mathit{TaskControl}_i \parallel \mathit{Idle}_i \parallel \mathit{Ready}_i \parallel \mathit{Running}_i \\
\mathit{Platform} &= \parallel_{j=1}^{m} \mathit{PE}_j \\
\mathit{PE}_j &= \mathit{Controller}_j \parallel \mathit{Synchronizer}_j \parallel \mathit{Scheduler}_j
\end{aligned}
$$

Each TA is approaching a comprehensible form

# Example TA: Running

- As an illustration of the level for complexity

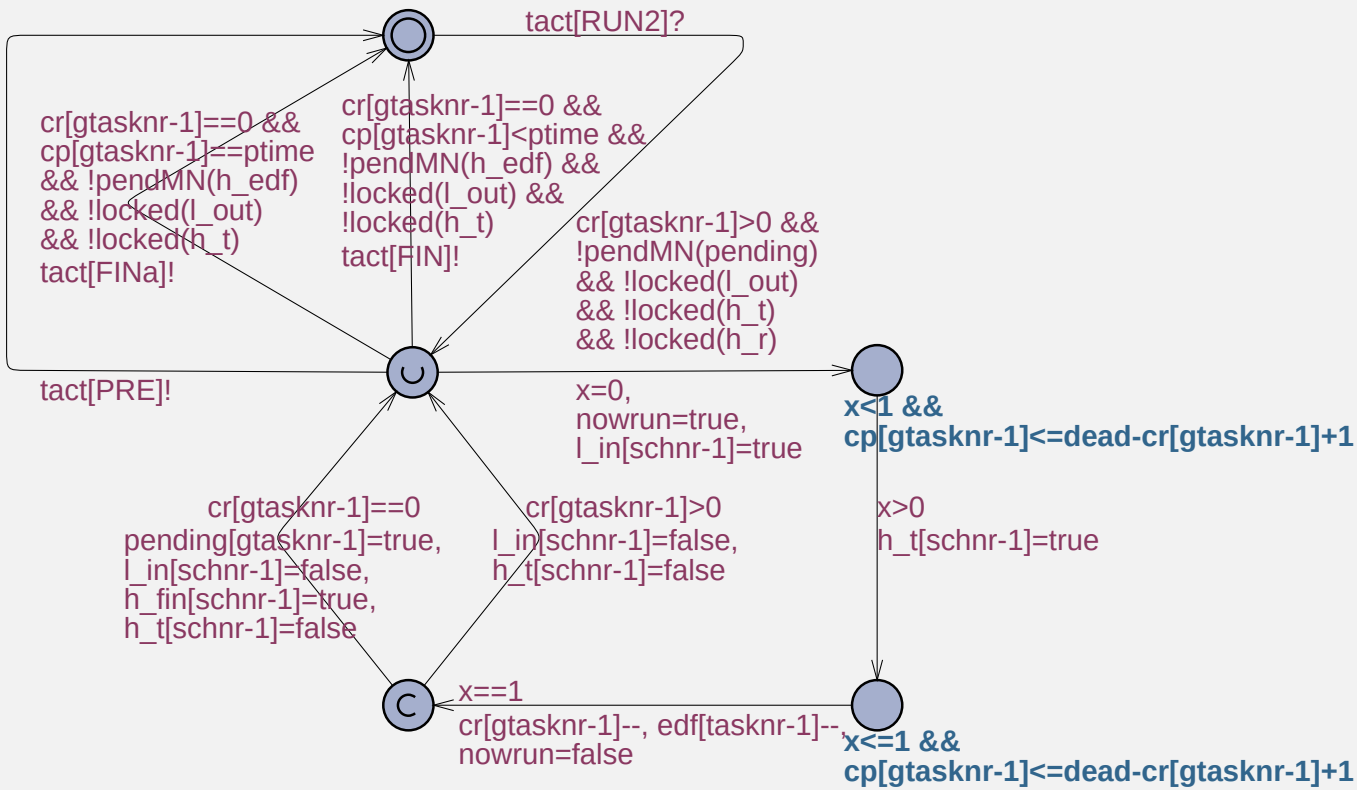
The Running state of ARTS is expanded to:



# Example TA: Running

- As an illustration of the level for complexity

The Running state of ARTS is expanded to:




- discretization of running time to cope with preemption.

# Status

- Simple model with
  - multiple processing elements, cyclic tasks, causal dependencies among tasks.
  - Scheduling disciplines: earliest deadline first, rate-monotonic scheduling, fixed priority scheduling, deadline-monotonic scheduling.
- Verification of a collection of small problems.
  - Up to 11 tasks on up to 6 processors are verified within 1.5 min on a small PC.
  - Single processor case: examples are verified and compared to results from TIMES tool.
  - In the multiprocessor case: Small examples from the literature are verified.

# Next steps?

- Refine and extend model to cope with non-cyclic tasks, communication, ...

# Next steps?

- Refine and extend model to cope with non-cyclic tasks, communication, ...
- Refine model to make it more suitable for verification

# Next steps?

- Refine and extend model to cope with non-cyclic tasks, communication, ...
- Refine model to make it more suitable for verification
- Model and reason about other costs than running time (energy consumption, ...) using UPPAAL Cora

# Next steps?

- Refine and extend model to cope with non-cyclic tasks, communication, ...
- Refine model to make it more suitable for verification
- Model and reason about other costs than running time (energy consumption, ...) using UPPAAL Cora
- Extensions inspired from the HARTEX kernels

# Next steps?

- Refine and extend model to cope with non-cyclic tasks, communication, ...
- Refine model to make it more suitable for verification
- Model and reason about other costs than running time (energy consumption, ...) using UPPAAL Cora
- Extensions inspired from the HARTEX kernels
- Extensions inspired from PAJ's platform

# Next steps?

- Refine and extend model to cope with non-cyclic tasks, communication, ...
- Refine model to make it more suitable for verification
- Model and reason about other costs than running time (energy consumption, ...) using UPPAAL Cora
- Extensions inspired from the HARTEX kernels
- Extensions inspired from PAJ’s platform
- "From specifications to task graphs"

# Next steps?

- Refine and extend model to cope with non-cyclic tasks, communication, ...
- Refine model to make it more suitable for verification
- Model and reason about other costs than running time (energy consumption, ...) using UPPAAL Cora
- Extensions inspired from the HARTEX kernels
- Extensions inspired from PAJ’s platform
- "From specifications to task graphs"
- Case studies
- . . .