

An example of early scheduling analysis with AADL

S. Rubini+, N. Tran+, M. Dridi+, V. Gaudel+, J. Boukhobza+,
A. Plantec+, C. Fotsing+, F. Singhoff (speaker)+,
P. Dissaux*, J. Legrand*, A. Schach*

* Ellidiss Technologies
+ Lab-STICC UMR CNRS 6285/UBO



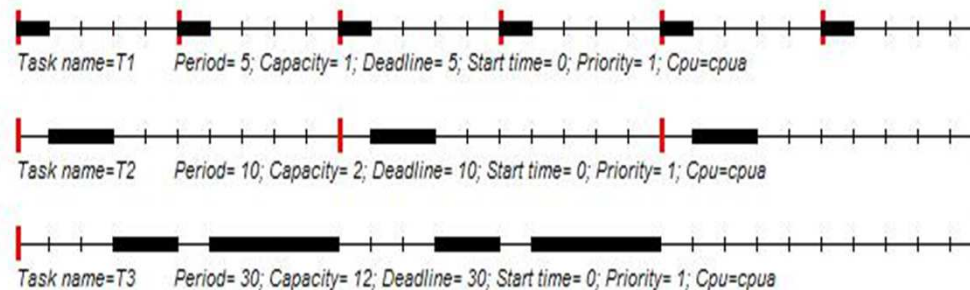
Summary

- 1. About scheduling analysis for early verification**
- 2. Cheddar projet**
- 3. AADL features to scheduling analysis**
- 4. Example of AADL software modeling and scheduling analysis**
- 5. Example of AADL execution platform modeling and scheduling analysis**
- 6. Conclusion**

About scheduling analysis

- ❑ **Simplified models of functions** : e.g. periodic task: processor demand + deadline.
- ❑ **Analysis**: feasibility tests, simulations, formal methods, ...

1. Scheduling Simulation:



2. Schedulability tests :

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{P_j} \right\rceil \cdot C_j$$

3. Formal methods (e.g. model checking)

Not used as much we can expect ☹️

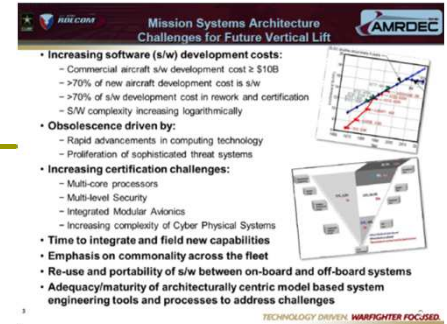
About scheduling analysis and early verification

❑ Motivation for early verification (source AMRDEC):

- ❑ 70% of fault are introduced during the design step ; Only 3% are found/solved. Cost : x1
- ❑ Unit test step: 20% of fault are introduced ; 16% are found/solved. Cost : x5
- ❑ Integration test step: 10% of fault are introduced ; 50% are found/solved. Cost : x16

❑ **Objective:** increase the number of faults found at design step!

❑ **Early verification:** multiple verifications, including expected performances, i.e deadlines can be met?



The slide is titled "Mission Systems Architecture Challenges for Future Vertical Lift" and features the AMRDEC logo. It contains a bulleted list of challenges and trends, accompanied by two charts. The first chart is a line graph showing exponential growth in software development costs. The second chart is a pyramid diagram illustrating the integration of various system components.

- **Increasing software (s/w) development costs:**
 - Commercial aircraft s/w development cost ≥ \$10B
 - >70% of new aircraft development cost is s/w
 - >70% of s/w development cost is rework and certification
 - SW complexity increasing logarithmically
- **Obsolescence driven by:**
 - Rapid advancements in computing technology
 - Proliferation of sophisticated threat systems
- **Increasing certification challenges:**
 - Multi-core processors
 - Multi-level Security
 - Integrated Modular Avionics
 - Increasing complexity of Cyber Physical Systems
- **Time to integrate and field new capabilities**
- **Emphasis on commonality across the fleet**
- **Re-use and portability of s/w between on-board and off-board systems**
- **Adequacy/maturity of architecturally centric model based system engineering tools and processes to address challenges**

TECHNOLOGY DRIVEN. WARFIGHTER FOCUSED.

About scheduling analysis and early verification

1. **How and what to model, in order to achieve early verifications?**
2. **Scheduling analysis requires advanced skills:**
 - Numerous theoretical results: how to choose the right one ?
 - Numerous assumptions for each result.
 - What to model ? What to abstract ?
3. **Engineers must be helped to use tools and methods:**
 - With which design languages ?
 - How to use scheduling tools ?
4. ...

**What is the role of an Architecture
Description Language?**

Summary

- 1. About scheduling analysis for early verification**
- 2. Cheddar projet**
- 3. AADL features to scheduling analysis**
- 4. Example of AADL software modeling and scheduling analysis**
- 5. Example of AADL execution platform modeling and scheduling analysis**
- 6. Conclusion**

Cheddar project : context and motivations

- ❑ **Motivations : how to apply real-time scheduling analysis at early steps?**
 - ❑ Started in 2002 by U. of Brest, partnership with Ellidiss Tech. since 2008 (industrial support).
 - ❑ Cheddar tool (open source, educational, research), AADLInspector (commercial product)
 - ❑ Modeling language : AADL as a driving line since 2004
 - ❑ **Other contributors** : Télécom-Paris-Tech (L. Pautet, E. Borde), ISAE (J. Hugues), Univ. Lisboa (J. Rufino), Univ. Sfax (B. Zalila), IUC (C. Fotsing)
 - ❑ **Main supports** : Ellidiss Tech., Brittany council, Brest City, Finistère council, Thalès communication, EGIDE/Campus France

Cheddar project : context and motivations

- 1. Propose, implement (early) scheduling analysis (Cheddar), investigate how to use them with AADL (AADLInspector)**
- 2. What to abstract from software and execution platform to achieve early scheduling analysis ?**
 - AADL (modeling language), Cheddar ADL (analysis language)
 - Scalability? Accuracy? Sustainability? To enforce analysis.
- 3. How to automatically perform scheduling analysis?**
- 4. How to achieve tools interoperability ?**
 - Various involved tools: model editors (Stood), code generators (Ocarina), WCET (aiT), system-C simulators, design exploration tools (Ramses), scheduling analysis (Cheddar), tool chains (TASTE)
 - Relationships between tools, formalize data interoperability

Summary

- 1. About Scheduling analysis for early verification**
- 2. Cheddar projet**
- 3. AADL features to scheduling analysis**
- 4. Example of AADL software modeling and scheduling analysis**
- 5. Example of AADL execution platform modeling and scheduling analysis**
- 6. Conclusion**

What to model to achieve early scheduling analysis

1. Software side:

- Workload: release time, execution time
- Timing constraints
- Software entity interferences, examples:
 - Tasks relationships/communication or synchronization : e.g. shared data, data flow
 - Task containers : ARINC 653 partition, process

2. Hardware (should be called execution platform) side:

- Available resources, e.g. computing capabilities
- Contention, interference, examples:
 - Processing units, cache, memory bus, NoC, ...

3. Deployment

=> Architecture models

=> It is the role of an Architecture Description Language to model those elements

AADL to the rescue?

❑ Why AADL helps:

❑ All required model elements are given for the analysis

- ❑ Component categories: thread, data, processor
- ❑ Feature categories: data access, data port, ...
- ❑ Properties: Deadline, Priority, WCET, Ceiling Priority, ...
- ❑ Annexes (e.g. behavior annex)

❑ AADL semantic: formal and natural language

- ❑ E.g. automata to define the concept of periodic thread
- ❑ Close to the real-time scheduling analysis methods

❑ Model engineering: reusability, several levels of abstraction

❑ Tools & chain tools: AADL as a pivot language (international standard)

- ❑ VERSA, OSATE, POLA/FIACRE/TINA, CARTS, MAST, Marzhin, Cheddar, ... by Ocarina/AADLInspector/RAMSES/MOSART/OSATE ...

AADL to the rescue?

But AADL does not solve everything:

- AADL is a complex language
- How to ensure model elements are compliant with analysis requirements/assumptions, sustainability, accuracy, ...
- Not a unique AADL model for a given system to model
- Not a unique mapping between a design model and an analysis model
- Having AADL scheduling analysis tools is not enough too, how to use them?
- ...

Summary

- 1. About scheduling analysis for early verification**
- 2. Cheddar projet**
- 3. AADL features to scheduling analysis**
- 4. Example of AADL software modeling and scheduling analysis**
- 5. Example of AADL execution platform modeling and scheduling analysis**
- 6. Conclusion**

AADL “design pattern” approach to automatically perform scheduling analysis

- ❑ **Let assume we have to evaluate a given architecture model in a design exploration flow.**

- ❑ **Problem statement:**
 - ❑ Numerous schedulability tests ; how to choose the right one?
 - ❑ Numerous assumptions for each schedulability test ; how to enforce them for a given model?
 - ❑ How to automatically perform scheduling analysis ?

AADL “design pattern” approach to automatically perform scheduling analysis

□ Approach:

- **Define a set of AADL architectural design patterns of real-time (critical) systems:**

= models a typical thread communication or synchronization + a typical execution platform

= set of constraints on entities/properties of the model.

- **For each design pattern,** define schedulability tests that can be applied according to their applicability assumptions.

- **Schedulability analysis of an AADL model:**

1. Check compliancy of his model with one of the design-patterns ... which then gives him which schedulability tests we can apply.
2. Perform schedulability verification.

Example : «Ravenscar» design pattern

- ❑ **Specification of various design patterns:**
 - **Time-triggered** : sampling data port communication between threads
 - **Ravenscar** : PCP shared data communication between threads
 - **Queued buffer/ARINC653** : producer/consumer synchronization
 - **Black board/ARINC653** : readers/writers synchronization
 - ...
 - **Compositions of design patterns.**

- ❑ **Ravenscar:** used by TASTE/ESA

- ❑ **Constraints defining “Ravenscar” to perform the analysis with a given schedulability test:**
 - Constraint 1 : all threads are periodic
 - Constraint 2 : threads start at the same time
 - Constraint 3 : shared data with PCP
 - Constraint n : fixed preemptive priority scheduling + uniprocessor
 - ...

Example : «Ravenscar» design pattern

□ Properties related to processor component:

```
Preemptive_Scheduler : aadlboolean applies to (processor);
```

```
Scheduling_Protocol:
```

```
  inherit list of Supported_Scheduling_Protocols  
  applies to (virtual processor, processor);
```

```
-- RATE_MONOTONIC_PROTOCOL,
```

```
-- POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL, ..
```

Example : «Ravenscar» design pattern

□ Properties related to threads/data components:

```
Compute_Execution_Time: Time_Range
  applies to (thread, subprogram, ...);

Deadline: inherit Time => Period applies to (thread, ...)

Period: inherit Time applies to (thread, ...);

Dispatch_Protocol: Supported_Dispatch_Protocols
  applies to (thread);
-- Periodic, Sporadic, Timed, Hybrid, Aperiodic, Backg
...

Priority: inherit aadlinteger applies to (thread, ..., dat

Concurrency_Control_Protocol:
  Supported_Concurrency_Control_Protocols applies to (dat
-- None, PCP, ICPP, ...
```

Example : «Ravenscar» design pattern

thread implementation ordo_bus.impl

properties

Dispatch_Protocol => Periodic;

Compute_Execution_Time => 31 ms .. 50 ms;

Deadline => 250 ms;

Period => 250 ms;

end ordo_bus.impl;

data implementation black.impl

properties

Concurrency_Control_Protocol

=> PRIORITY_CEILING_PROTOCOL;

end black.impl;

process implementation Application.impl

subcomponents

camera : **thread** camera.impl;

ordo_bus : **thread** ordo_bus.impl;

target : **data** black.impl;

...

processor implementation leon2

properties

Scheduling_Protocol =>

POSIX1003_HPF_PROTOCOL;

Preemptive_Scheduler => true;

end leon2;

system implementation finder

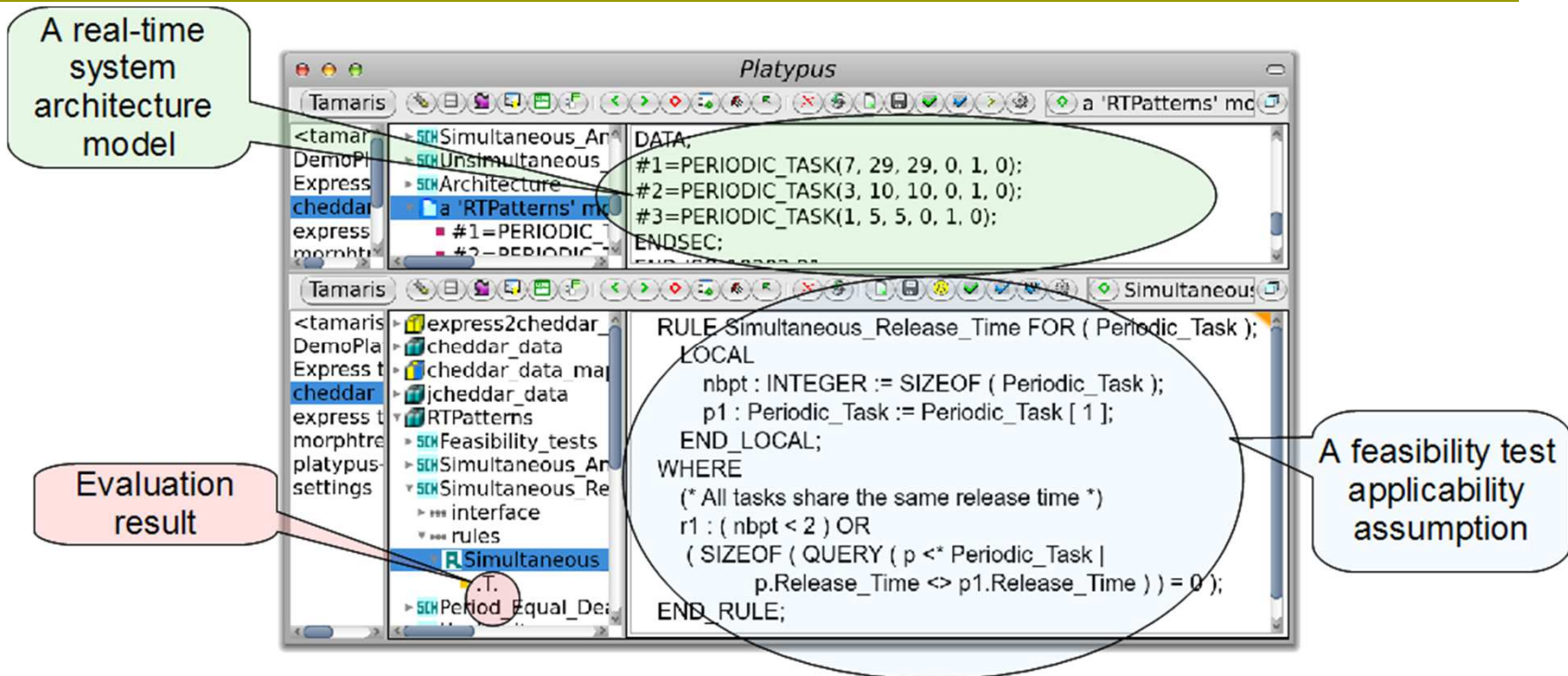
subcomponents

process1 : **process** soft.impl;

cpu1 : **processor** leon2;

...

Design pattern compliancy verification



- ❑ **Top right part:** real-time system architecture model to verify.
- ❑ **Bottom right part:** modeling of a feasibility test applicability assumption.
- ❑ **Left part:** result of the model compliancy analysis.

Example : «Ravenscar» design pattern

The screenshot displays the AADL Inspector interface for a file named 'exo2.aadl'. The left pane shows the source code, and the right pane shows a static analysis timeline.

```
85 end mesure;  
86  
87 thread implementation mesure.Impl  
88 end mesure.Impl;  
89  
90 thread meteo  
91 features  
92 fe : requires data access black.Impl;  
93 properties  
94 Dispatch_Protocol => Periodic;  
95 Period => 5000 ms;  
96 Deadline => 5000 ms;  
97 Compute_Execution_Time => 50 ms .. 75 ms;  
98 Priority => 4 ;  
99 end meteo;  
100 thread implementation meteo.Impl  
101 end meteo.Impl;  
102  
103  
104  
105 process Application  
106 end Application;  
107  
108 process implementation Application.Impl  
109 subcomponents  
110 ordo_bus : thread ordo_bus.Impl;  
111 donnees : thread donnees.Impl;  
112 pilotage : thread pilotage.Impl;  
113 radio : thread radio.Impl;  
114 camera : thread camera.Impl;  
115 mesure : thread mesure.Impl;  
116 meteo : thread meteo.Impl;  
117 black : data black.Impl;  
118 connections  
119 cx1 : data access black -> donnees.fe;  
120 cx2 : data access black -> pilotage.fe;  
121 cx3 : data access black -> mesure.fe;
```

The static analysis timeline on the right shows the execution of various threads on 'cpu1'. The threads are: process1 (green), ordo_bus (black), donnees (black), pilotage (black), radio (black), camera (black), mesure (black), meteo (black), and black (black). The timeline shows the execution of these threads over time, with a scale from 0 to 105 ms.

Summary

- 1. About scheduling analysis for early verification**
- 2. Cheddar projet**
- 3. AADL features to scheduling analysis**
- 4. Example of AADL software modeling and scheduling analysis**
- 5. Example of AADL execution platform modeling and scheduling analysis**
- 6. Conclusion**

AADL Modeling of Multiprocessor Systems

□ Problem statement:

1. Which design patterns for multiprocessor architectures?
2. From scheduling analysis point of view: how to model classical multiprocessor scheduling analysis concepts:

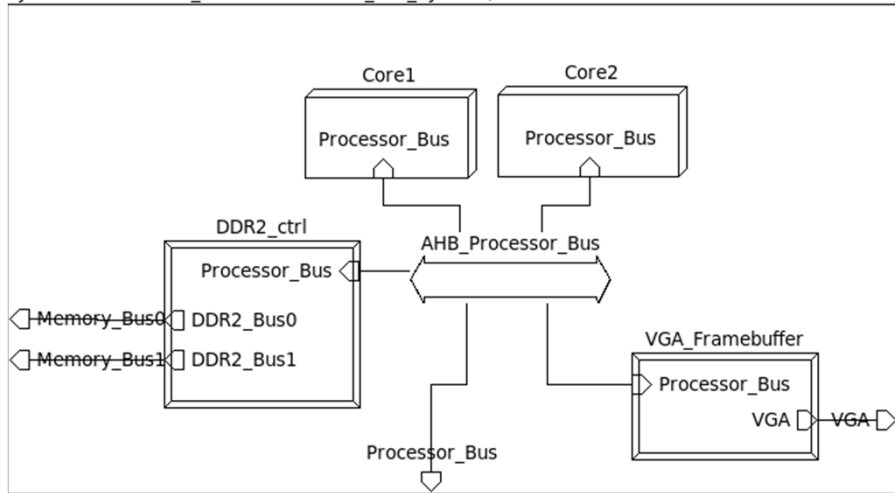
partitioned scheduling or global scheduling

3. Various multiprocessor architectures?
4. With (or without) shared resources? cache unit, NoC?
5. Which analysis methods?
6. Scalability, accuracy and sustainability? To enforce analysis.

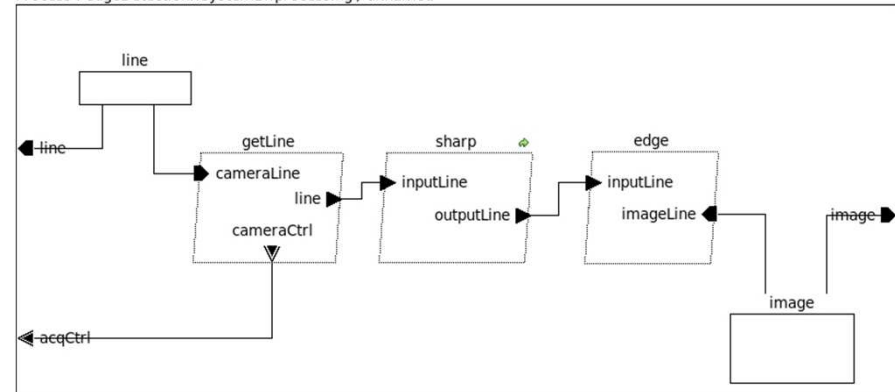
□ Example of a SoC with two Leon4 cores, with L1 cache and a multimedia application

Example: Design-Pattern for Partitioned Scheduling

System : board::SoC_LEON4::Processor_Bus_System / unnamed



Process : edgeDetection::System1::processing / unnamed



SYSTEM IMPLEMENTATION product.impl

SUBCOMPONENTS

```
hard : SYSTEM soc_leon4::soc.asic_leon4;
bank0 : MEMORY ram. ddr2;
bank2 : MEMORY ram. ddr2;
soft : PROCESS edgeDetection.impl;
```

PROPERTIES

```
actual_processor_binding => (REFERENCE(hard.Proc_System.Core1)) APPLIES TO soft.getLine;
actual_processor_binding => (REFERENCE(hard.Proc_System.Core2)) APPLIES TO soft.sharp;
actual_processor_binding => (REFERENCE(hard.Proc_System.Core2)) APPLIES TO soft.edge;
Scheduling_Protocol => (Rate_Monotonic_Protocol) applies to hard.core1;
Scheduling_Protocol => (Rate_Monotonic_Protocol) applies to hard.core2;
```

```
END product.impl;
```

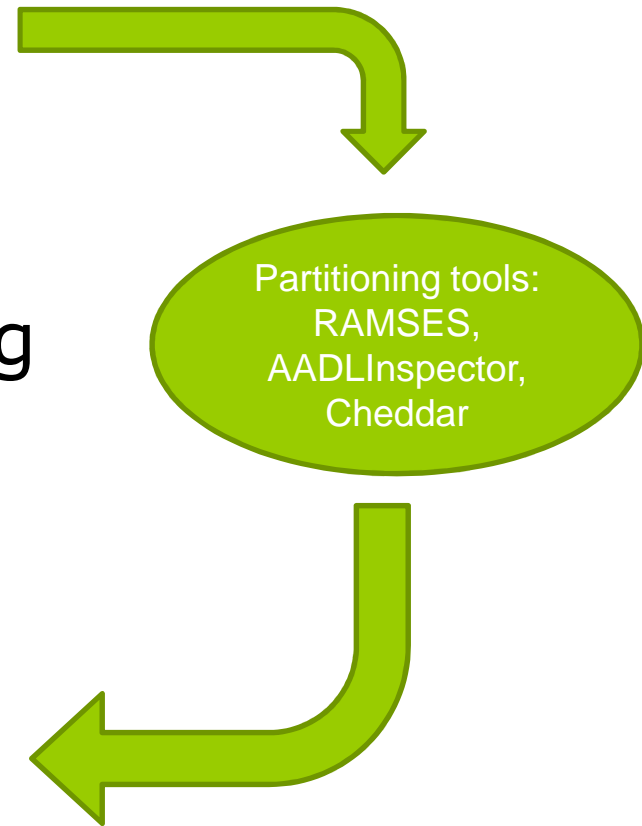

Design exploration, i.e. task partitioning

AADL model before partitioning

```
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                             APPLIES TO soft.getLine;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                             APPLIES TO soft.sharp;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                             APPLIES TO soft.edge;
```

AADL model after partitioning

```
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                             APPLIES TO soft.getLine;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                             APPLIES TO soft.sharp;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                             APPLIES TO soft.edge;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1))
                           APPLIES TO soft.getLine;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1))
                           APPLIES TO soft.sharp;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core2))
                           APPLIES TO soft.edge;
```



Example: Design-Pattern for Global Scheduling

```
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),  
                             REFERENCE(hard.Proc_System.Core2))  
                             APPLIES TO soft.getLine;
```

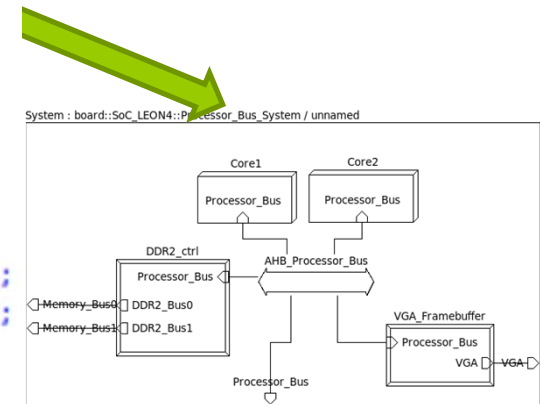
```
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),  
                             REFERENCE(hard.Proc_System.Core2))  
                             APPLIES TO soft.sharp;
```

```
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),  
                             REFERENCE(hard.Proc_System.Core2))  
                             APPLIES TO soft.edge;
```

```
Scheduling_Protocol => Rate_Monotonic_Protocol applies to hard.Core1;
```

```
Scheduling_Protocol => Rate_Monotonic_Protocol applies to hard.Core2;
```

```
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),  
                             REFERENCE(hard.Proc_System.Core2)) applies to soft;
```



Example of analysis with AADLInspector

The screenshot displays the AADL Inspector interface with the following components:

- Code Editor:** Shows AADL code for a system implementation, including subcomponents, processor bindings, and properties.
- Static Analysis:** A Gantt chart showing execution timelines for components like `ea_core1`, `ea_core2`, `getLine`, `sharp`, and `edge` across two cores.
- Edit real time properties dialog:** A table for configuring processor placement for various components.

```
574 SYSTEM IMPLEMENTATION product.impl
575 SUBCOMPONENTS
576   hard : SYSTEM soc_leon4::soc.asic_leor
577   bank0 : MEMORY ram.ldr2;
578   bank2 : MEMORY ram.ldr2;
579   soft : PROCESS edgeDetection.impl;
580 -- Use the Processor Placement Wizard to
581 -- Actual Processor Bindings
582 PROPERTIES
583 allowed_processor_binding => (
584   REFERENCE(hard.Proc_System.Core1),
585   REFERENCE(hard.Proc_System.Core2) )
586 APPLIES TO soft.getLine;
587 allowed_processor_binding => (
588   REFERENCE(hard.Proc_System.Core1),
589   REFERENCE(hard.Proc_System.Core2) )
590 APPLIES TO soft.sharp;
591 allowed_processor_binding => (
592   REFERENCE(hard.Proc_System.Core1),
593   REFERENCE(hard.Proc_System.Core2) )
594 APPLIES TO soft.edge;
595 END
596
597 MEMORY
598 END
599
600 MEMORY
601 END
602
603 DATA
604 END
605
606 DATA
607 END
608
```

Name	Actual Processor(s)	Allowed Processor(s)
soft.getLine	hard.proc_system.core1	hard.proc_system.core1,hard.proc_system.core2
soft.sharp	hard.proc_system.core2	hard.proc_system.core1,hard.proc_system.core2
soft.edge	hard.proc_system.core2	hard.proc_system.core1,hard.proc_system.core2

Shared resource example: cache unit

- ❑ **Problem statement:** What to model to priority assignment?
 - ❑ Thread code: to compute execution time and CFG
 - ❑ Cache and memory configuration: to compute thread cache access profile
 - ❑ Scheduling parameters

- ❑ **AADL:** is able to model thread execution time, scheduling parameters, cache and memory configuration
 - ❑ CFG? Behavior annex?
 - ❑ Cache access profile (computed)? Properties? Behavior annex?

Shared resource example: cache unit

❑ Example: an example of user-defined property sets

```
PROCESSOR IMPLEMENTATION core.leon4
SUBCOMPONENTS
  ll_inst_cache : MEMORY cache.impl {
    BYTE_COUNT => 8192 Byte;
    CACHE_PROPERTIES::LINE_SIZE => 32 Byte;
    CACHE_PROPERTIES::CACHE_TYPE => Instruction_Cache;
    CACHE_PROPERTIES::SET_ASSOCIATIVITY => Set_Associative;
    CACHE_PROPERTIES::WRITE_POLICY => No_Allocated_Write_Through;
    CACHE_PROPERTIES::REPLACEMENT_POLICY => LRR;
    CACHE_PROPERTIES::CACHE_LEVEL => 1;
    CACHE_PROPERTIES::SET_SIZE => 2;
    CACHE_PROPERTIES::CACHE_COHERENCY_PROTOCOL => Private_Invalid;
  };
  ll_data_cache : MEMORY cache.impl {
    BYTE_COUNT => 4096 Byte;
    CACHE_PROPERTIES::LINE_SIZE => 32 Byte;
    CACHE_PROPERTIES::CACHE_TYPE => Data_Cache;
    CACHE_PROPERTIES::SET_ASSOCIATIVITY => Set_Associative;
    CACHE_PROPERTIES::WRITE_POLICY => No_Allocated_Write_Through;
    CACHE_PROPERTIES::REPLACEMENT_POLICY => LRR;
    CACHE_PROPERTIES::CACHE_LEVEL => 1;
    CACHE_PROPERTIES::SET_SIZE => 2;
    CACHE_PROPERTIES::CACHE_COHERENCY_PROTOCOL => Private_Invalid;
  };
END core.leon4;
```

Cache/CRPD-Aware Priority Assignment Algorithm

- ❑ In fixed priority preemptive scheduling context, tasks can preempt and evict data of other tasks in the cache.
- ❑ Cache related preemption delay (**CRPD**): additional time to refill the cache with the cache blocks evicted by the preemption.

- ❑ **Problem statement:**
 - ❑ CRPD may be high, non-negligible preemption cost (Pellizzoni et al., 2007).
 - ❑ No fixed priority assignment algorithm takes CRPD into account.

Cache/CRPD-Aware Priority Assignment Algorithm

□ Approach:

- Extend Audsley's priority assignment algorithm (Audsley, 1995) to take into account CRPD.
- CRPD-aware priority assignment algorithms (**CPA**) that assign priority to tasks and verify their schedulability.
- 5 algorithms with different levels of schedulability efficiency (1) and complexity (2), scalability (3).
- Implemented into Cheddar, Not available in AADLInspector today

	CPA-PT-Simplified	CPA-PT	CPA-Tree	Exhaustive Search
(1)	0.65	0.72	0.80	0.87
(2)	Low	Medium	High	
(3)	100 tasks	30 tasks	10 tasks	

Scheduling Simulator with execution platform shared resources

Problem statement:

- How to model shared resources? Abstraction level?
- Tick accurate versus cycle accurate
- Take into account various parameters from different sources (tools)
- Theoretical issues (feasibility interval, sustainability, accuracy)

Focus on instruction cache, NoC

Not available in AADLInspector today

Example with NoC:

- Dual Task and Flow Model (DTFM) : computes the flow model from a task model, task mapping, precedence constraints on a NoC
- Identify/compute delays induced by Wormhole XY NoC and perform scheduling analysis for AADL data port software design
- Evaluation with tick + cycle accurate simulator (SHOC SystemC NoC simulator + Cheddar)

Summary

- 1. About scheduling analysis for early verification**
- 2. Cheddar projet**
- 3. AADL features to scheduling analysis**
- 4. Example of AADL software modeling and scheduling analysis**
- 5. Example of AADL execution platform modeling and scheduling analysis**
- 6. Conclusion**

Conclusion

AADL & early scheduling analysis:

- Design-pattern approach
- Constraints on model entities (e.g. component categories, feature categories, properties, annexes) to enforce analysis
- One model for a given intend
- Examples of both software and execution platform modeling/analysis
- Current work: multiprocessor architectures
- Evaluation, details => see publications

Lessons learnt:

- AADL v2 is enough to model (most of) the software/execution platforms we investigated
- Tools can interact: STOOD, RAMSES, AADLInspector, Ocarina, OSATE, WCET tools, System-C simulators, Cheddar, ...
- Analysis features developed by the Lab-STICC: prototyped with Cheddar, available for AADL with AADLInspector