Multicore Modeling with AADL

Julien Delange

March 24, 2015

This document discusses the different modeling patterns to represent multicore processors with AADLv2. The main motivation for such an article is the modeling of multi-core architecture for ARINC653 applications. In ARINC653, partitions are still scheduled using a fixed time-slice protocol. Partitions cannot be executed concurrently on separate core. On the other hand, the configuration indicate what core are used for each partition. In other hands, on a four cores system, one partition can use three cores, another one can use just two and another one use all of them. The core not being used are then inactive.

When starting to represent multi-core concepts, the author start to face several issues. First of all, the use of `virtual processor` to model core in the context of ARINC653 systems introduce a semantic issue, the same component is then used for multiple purposes (partition runtime and core). In addition, when using the `virtual processor` component, this removes the ability to represent physical interaction with shared resources between multiple cores.

We present several modeling patterns, discuss their advantages/drawbacks and recommend a solution for standardizing the modeling of multi-core architectures.

1 Processor with virtual processors

```
package onepart_multicore

public


with Data_Model;
with ARINC653;

virtual processor partition_runtime
end partition_runtime;

virtual processor implementation partition_runtime.impl
end partition_runtime.impl;

virtual processor core
end core;

virtual processor implementation core.impl
end core.impl;

processor module
end module;

processor implementation module.impl
subcomponents
   core1 : virtual processor core.impl;
properties
   ARINC653::Module_Major_Frame => 1000ms;
end module.impl;

process myprocess
end myprocess;

system node
end node;

system implementation node.impl
```

```
subcomponents
   cpu    : processor module.impl;
   part1  : process myprocess;
   partition_runtime : virtual processor partition_runtime;
properties
   Actual_Processor_Binding => (reference (cpu.core1)) applies to partition_runtime;
   ARINC653::Module_Schedule =>
       ( [Partition => reference (partition_runtime);
          Duration  => 10 ms;
          Periodic_Processing_Start => false;]
    ) applies to cpu;
end node.impl;

end onepart_multicore;
```

Listing 1: Processor with Virtual Processors

1.1 Pros

1. Do not need to change the standard

1.2 Cons

1. **Do not capture the hardware nature of a core** and potential interactions with buses. This might be critical when analyzing the timing and shared resources of a core.
2. **Conflict with the ARINC653 annex**: by using such a pattern, the `virtual processor` is used to capture a core and a partition runtime. This is conflicting and one might one to distinguish these concepts.
3. **Compatibility of properties**: properties of processors (e.g. speed) should also apply to cores. Then, by representing each core with virtual processor, property sets must be updated.

2 Systems with processors

```
package onepart_multicore

public


with Data_Model;
with ARINC653;

virtual processor partition_runtime
end partition_runtime;

virtual processor implementation partition_runtime.impl
end partition_runtime.impl;

processor core
end core;

processor implementation core.impl
end core.impl;



system module
end module;

system implementation module.impl
subcomponents
   core1 : processor core.impl;
--properties
--    ARINC653::Module_Major_Frame => 1000ms;
end module.impl;

process myprocess
```

```
end myprocess;

system node
end node;

system implementation node.impl
subcomponents
   cpu    : system module.impl;
   part1  : process myprocess;
   partition_runtime : virtual processor partition_runtime;
properties
   Actual_Processor_Binding => (reference (cpu.core1)) applies to partition_runtime;
   ARINC653::Module_Schedule =>
       ( [Partition => reference (partition_runtime);
          Duration  => 10 ms;
          Periodic_Processing_Start => false;]
    ) applies to cpu;
end node.impl;

end onepart_multicore;
```

Listing 2: Processor with Virtual Processors

2.1 Pros

1. **Do not need to change the standard**
2. **Capture the physical aspects**

2.2 Cons

1. **Do not capture the hardware nature of a core** and potential interactions with buses. This might be critical when analyzing the timing and shared resources of a core.
2. **Backwards compatibility of analysis tools** (some analysis tools will not consider a system as a processor)
3. **Compatibility of properties**: properties using the processor as the OS needs to be updated. In that case, OS-related properties must also apply to `system` components then.
4. **Lack of semantics**: use the system as a container whereas a processor is a physical entity

3 Processors with processors

4 Systems with processors

```
package onepart_multicore

public


with Data_Model;
with ARINC653;

virtual processor partition_runtime
end partition_runtime;

virtual processor implementation partition_runtime.impl
end partition_runtime.impl;

processor core
end core;

processor implementation core.impl
end core.impl;
```

```
processor module
end module;

processor implementation module.impl
subcomponents
   core1 : processor core.impl;
--properties
--    ARINC653::Module_Major_Frame => 1000ms;
end module.impl;

process myprocess
end myprocess;

system node
end node;

system implementation node.impl
subcomponents
   cpu   : system module.impl;
   part1 : process myprocess;
   partition_runtime : virtual processor partition_runtime;
properties
   Actual_Processor_Binding => (reference (cpu.core1)) applies to partition_runtime;
   ARINC653::Module_Schedule =>
       ( [Partition => reference (partition_runtime);
          Duration  => 10 ms;
          Periodic_Processing_Start => false;]
    ) applies to cpu;
end node.impl;

end onepart_multicore;
```

Listing 3: Processor with Processors

4.1 Pros

1. **No conflict with the ARINC653** and the concept of core and partition runtime
2. **Backward compatible with older versions**
3. **Current analysis tools still compatible**
4. **Represent physical aspects**

4.2 Cons

1. **Need to change the standard**: the AADL standardization committee need then to change the standard and approve `processor` subcomponents inside processors.

5 Conclusion

We advocate to update the AADL standard and allow the use of `processor` subcomponents inside components. This would then bring the benefits of capturing multicore architecture with different levels of fidelity (modeling at a high level or also having the ability to represent low-level details and shared resources).

A `processor` component would then represent the physical component (the main hardware chip) or one core (when being used as a subcomponent. On the other hand, a `virtual processor` is a software layer representing an processing resource, such as the partition runtime of an ARINC653 or MILS system.

We also recommend to introduce new properties dedicated to multi-core applications. In particular, having a property *Core_Id* that represents the core identifier so that system designer can use it to generate code. Such a concept was already introduced in POSIX and is currently being standardized by the ARINC653 committee. Having this concept built-in in the AADL core properties would then be a significant improvement.

6 Copyright