Eric Verhulst, CEO/CTO

occam's Rule Applied – Separation of Concerns as a Key to Trustworthy Embedded Systems Engineering

Content

- A bit of history: Altreonic profile
- Trustworthy Systems Engineering
- Unified Semantics: GoedelWorks
- ARRL: Assured Reliability and Resilience Level
- Interacting Entities: VirtuosoNext Designer
- New VirtuosoNext 2.0:
  - Fine Grain Space and Time Partitioning
  - Non-Stop capability

Altreonic profile

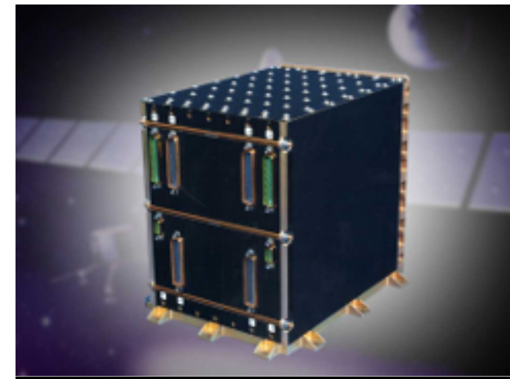
- 30 years of aero-space-defense experience (**Eonic Systems NV**)
  - Specialised in parallel **Virtuoso** Real-Time Operating System
  - Used from 1 to 1600 processors (sonar, radar) to > 12000 nodes
  - Used by ESA (Virtuoso RTOS on Rosetta mission)
  - Virtuoso acquired by Wind River Systems Inc. in 2001
- **Altreonic**: created as new spin-off in 2008 after R&D
  - Focus on **<u>trustworthy scalable embedded systems</u>**
    - Safety, Security, Usability, Privacy
  - Using formal methods => **VirtuosoNext Designer (RTOS)**
  - Portal based environment to support SE: **GoedelWorks**
  - Unique "**Open Technology License**" model
  - Competitive advantage to develop novel **KURT Light e-Vehicle**

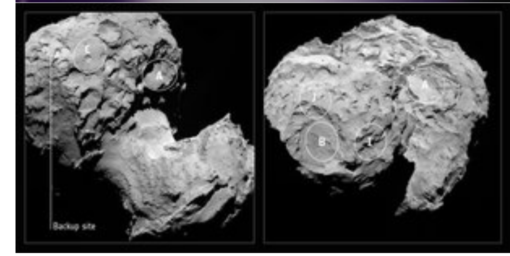
The Virtuoso RTOS

- Adapt service “semantics” to be MP compatible
  - Pass by value, not pass by reference
  - Packets replace remote function calls (under the hood)
  - Semaphores, FIFOs, ….
- How to make it MP and transparent?
  - If remote, put service request and params in a packet across the wire
  - Simple routing: look-up table for next link
  - Took about 2 months
- First RTOS on transputer in 1991, Sunnyvale

Going into space

- 21020 RT: radiation hardened DSPs (20 MHz)
  - With SMCS SpaceWire (=T9000) links
- Boards developed by EADS
- Used in multiple scientific missions
  - Giotto
  - Rosetta: landing on a comet in 2014
  - Still in use on e.g. ISS





Space related projects/proposals

- Virtuoso multi-DSP project Mosaic-20 development with EADS.
- Virtuoso used on several scientific missions.
- Feasibility study for a reconfigurable telecom satellite
- Proposal for a rad-hard family of processors and FPGA
- EU-project for long-life electronics (100 yrs)
- Emulators of Future NGMP Multicore Processors
- VirtuosoNext has been ported to LEON-3

Trustworthy Systems Engineering

Trustworthy Systems Engineering

- Confusion: what do people really want / need ?
- Mixing up problem domain with a known solution
- Thinking about what can go wrong
  - Law of Murphy always applies, not just probabilities
- Language: semantics
- Nobody sees everything, but thinks his view is the dominant one
- Productivity!
  - Time to safety < > time to market: conflict?
- Software could be error-free, hardware never is

What is "trustworthy"?

- **Trust**: from the point of view of the "user"
  - **Safety**: absence of risk of being harmed/killed
    - Hazards and faults
  - **Security**:
    - Maliciously injected fault (=> subcase of safety)
  - **Usability**:
    - Man Machine Interface: intuitive ? (=> safety)
  - **Privacy**:
    - Harm is financial, emotional, …

Meta-modeling vs. modeling

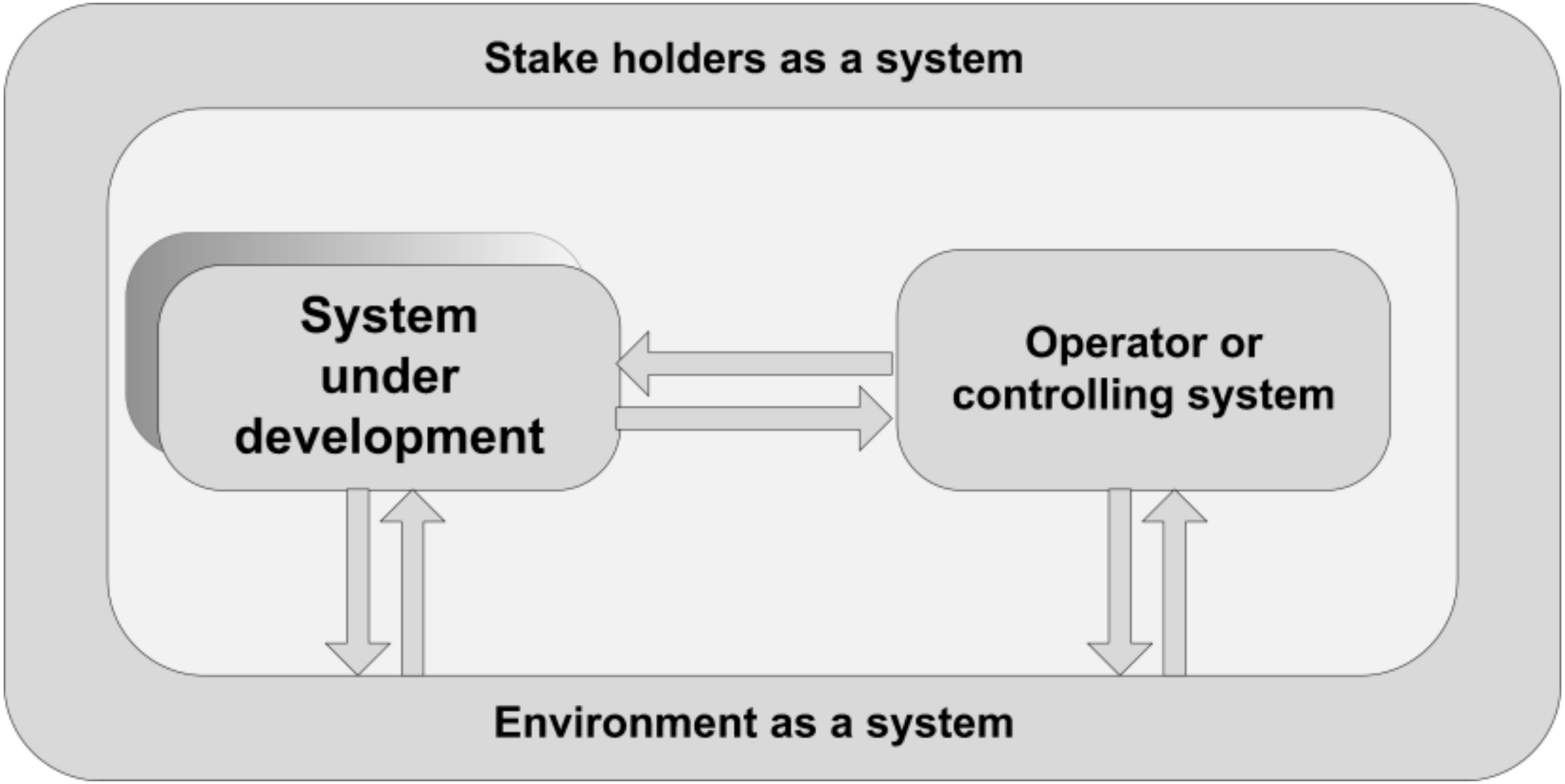
- Model-driven engineering
  - Model then implement
- Is UML modeling?
  - Or a description? How precise is UML?
- Modeling means abstraction! (> 1 level)
- Models need to be univoque!
- If a system is an implemented model, how do we describe its properties and architecture in a univoque way?

Key paradigms

- Issue: all (point) tools and methodologies speak a different language
- One needs multiple tools to build a single system (for productivity!)
- N tools = NxN translators, but possible?
- Key paradigm: **Unified Semantics / Methodology**
  - **Speak same language everywhere**
- Key paradigm: **Interacting Entities**
  - **Meta-level description of architecture**

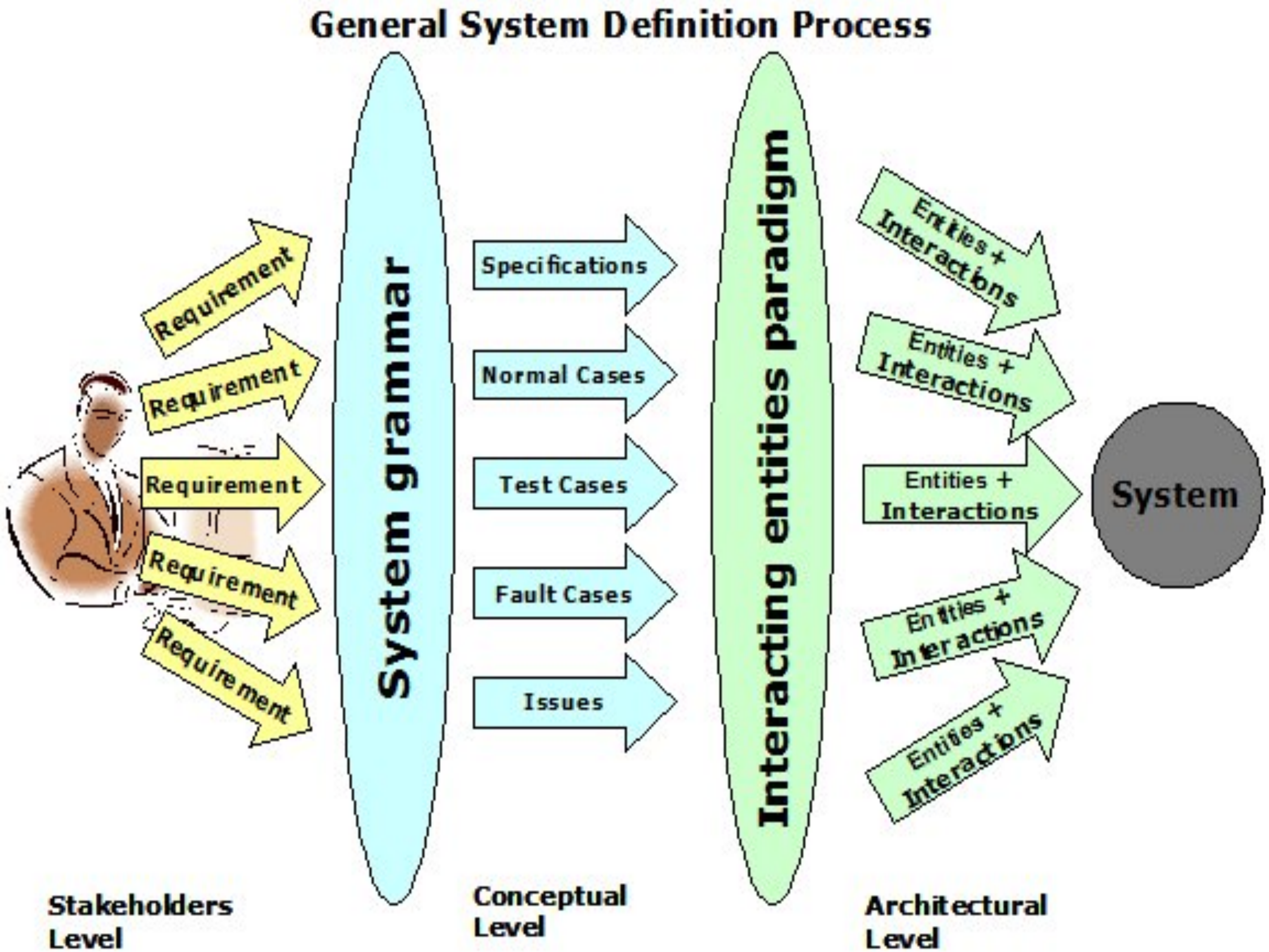
Unified Semantics: GoedelWorks

A system is never alone

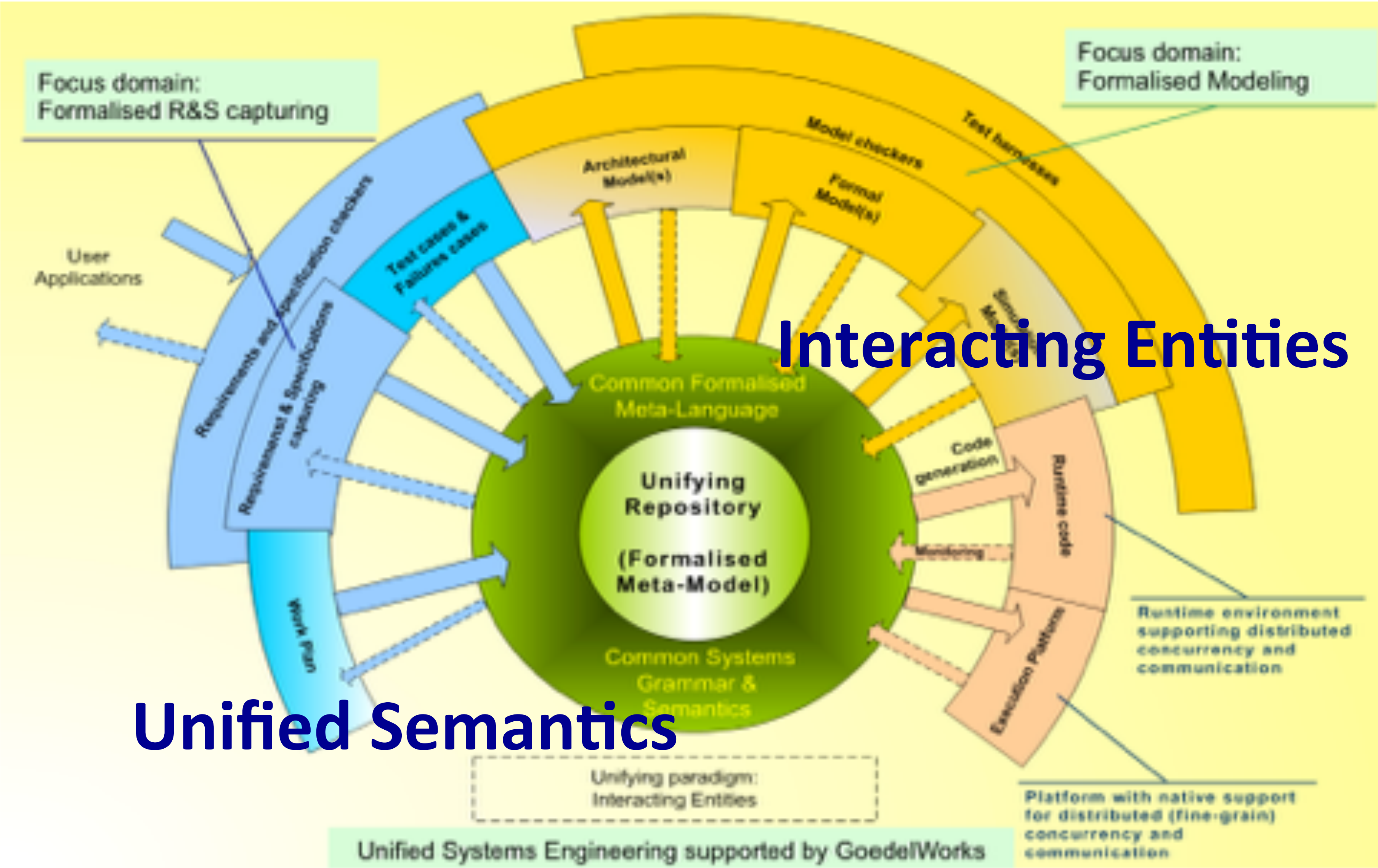


See workhop on hybrid systems

Human factors: from idea to reality



Altreonic's methodology

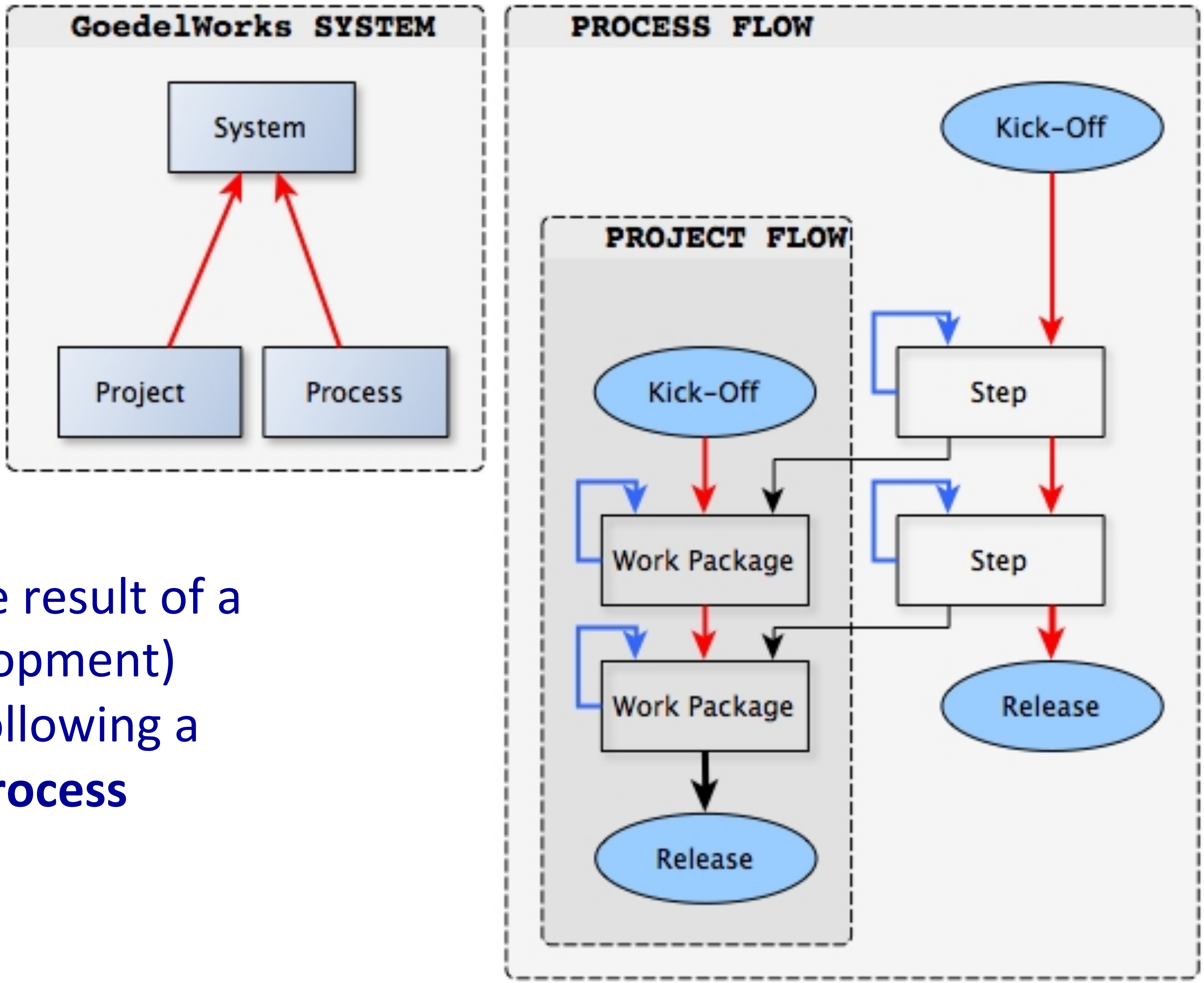


Unifying metamodel in GoedelWorks

- Model is generic for all engineering domains
- Bias towards embedded systems:
  - Software
  - Hardware
  - Mechatronics
- Simple yet complete
- Customisation for each organisation:
  - Merge organisational process flow with standard's process requirement

Orthogonal core concepts (1)

Reference	Any relevant information, external and generic, but of potential importance for the Project: datasheet, standard, paper, …
Requirement	Any statement by any stakeholder about the system to be developed: technical as well as non-technical
Specification	A Requirement that by refinement and decomposition can be tested and verified. (requires: Test Case)
Work Package (implements Process Step)	A collection of coherent and planned Activities that result in the availability of an Item or Work Product that meets the Specifications. "Develop the right things" (what), "Develop it right" (how)
Resource	An Item or Work Product needed in a Work Package in order to execute the Activities in a Work Package or Step.
Work Product: Models and	The result of a Work Package. An Artefact is the supporting evidence

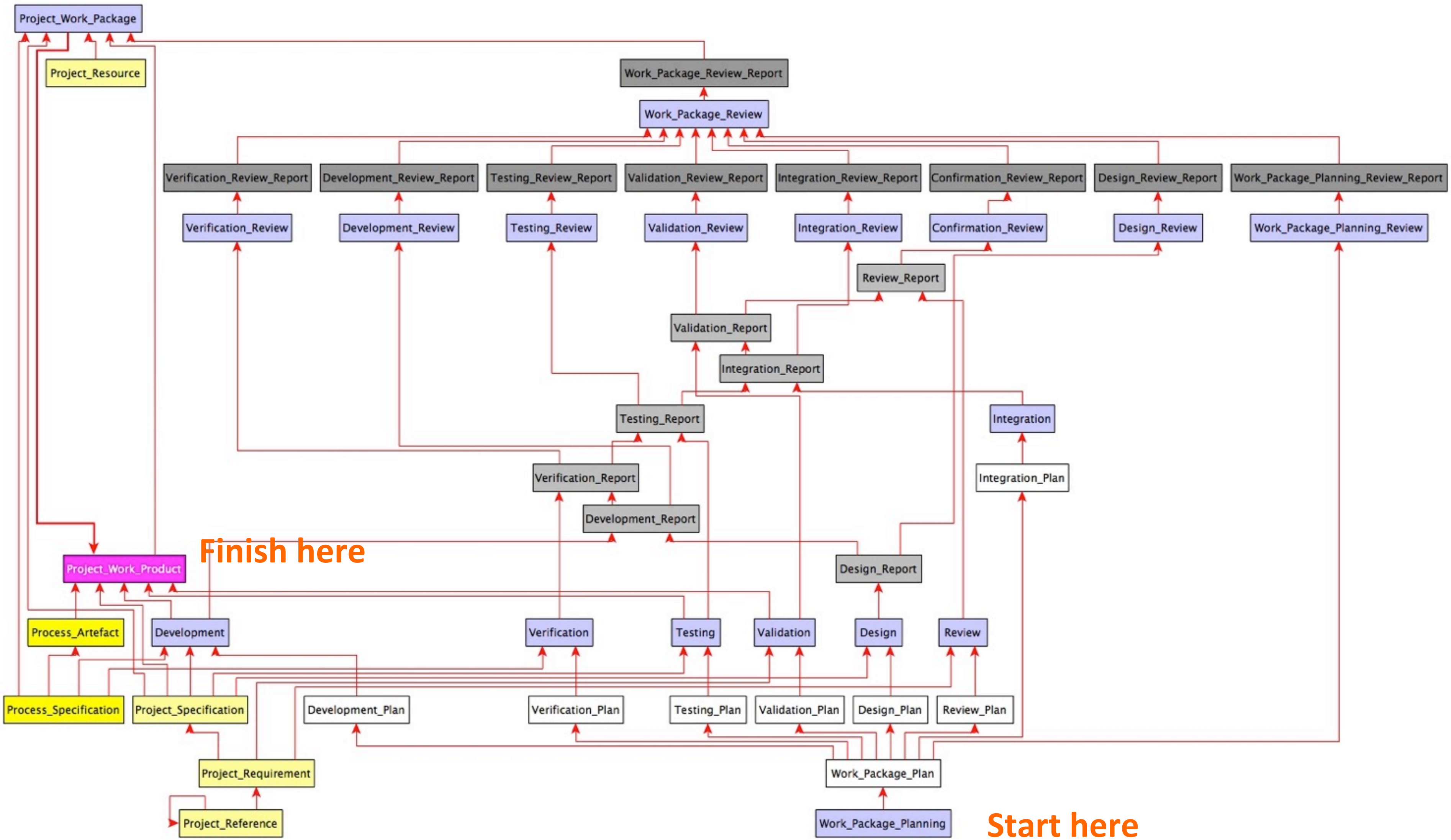


A system is the result of a **Project** (development) executed by following a (prescribed) **Process**

Work Package pattern as template

8 Work Package Activities	
Planning Design Development Verification	Testing Integration Validation Review
4 phases each	
Planning – Doing – Document - Confirmation	

Standard template pattern for WP



Orthogonal concepts (2)

Planning	Describe how an Activity will be performed. Includes what? How? When? Where? With what? ….
Design	Specify the architecture (incl. interfaces)
Development	The actual activity that takes all inputs and develops a concrete Item instance that fulfills the Specifications
Verification	Verifying that the Development was done according to the Process Specifications. "Was the work done as it should have been?"
Testing	Verifying that the Item meets its Specifications (execute Test Case)
Integration	Assemble the Items into a system or subsystem component.
Validation	Verify that the Integrated Items meet the Requirements and Specifications as a whole.
Review	Double check (using independent people)

Orthogonal concepts (3)

For each Activity:

Planning	Describe how an Activity will be performed. Includes what? How? When? Where? With what? ….
Doing	The actual activity that takes all inputs and develops a concrete Item instance that fulfills the Specifications
Document	Leaving a "trace" (evidence) for later reference.
Confirmation	Independent Review

Hence: iterative, double check at higher level

WP Internal Resources

For each WP and its Actvities:

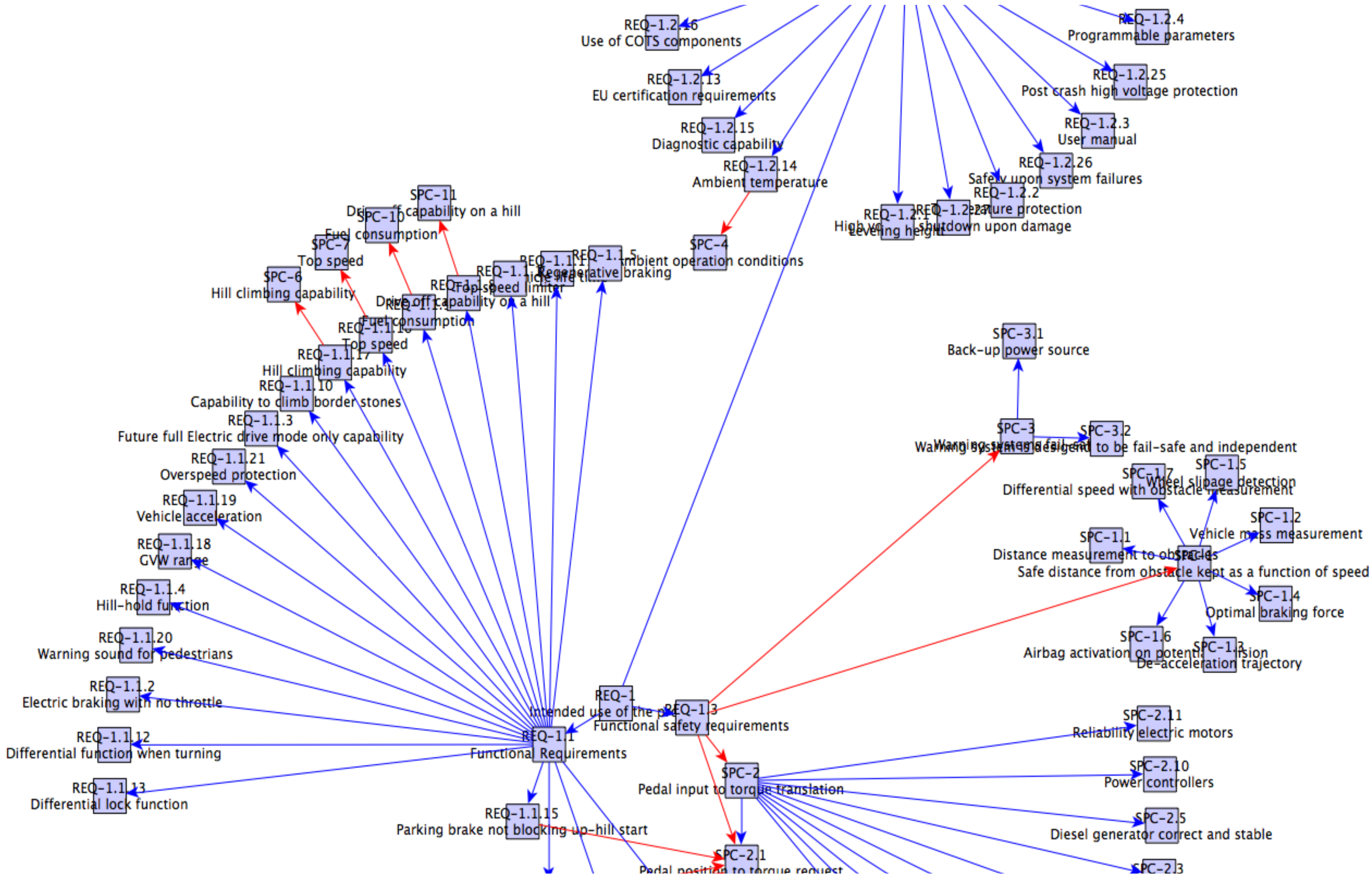
Internal Resource	Generic Resources (human, equipment, tool, financial) assigned to a WP Activity. Typically at Kick-Off.
Person has capabilities	Specified and verified/tested capabilities of a person to fulfill a specified Role: e.g. Test Engineer, Team Leader, Safety Assessor
Person has Roles	RACI: Responsible, Accountable, Consulted, Informed
External Resource	Resource that is produced outside the WP, hence creating a dependency.

Human factor made explicit

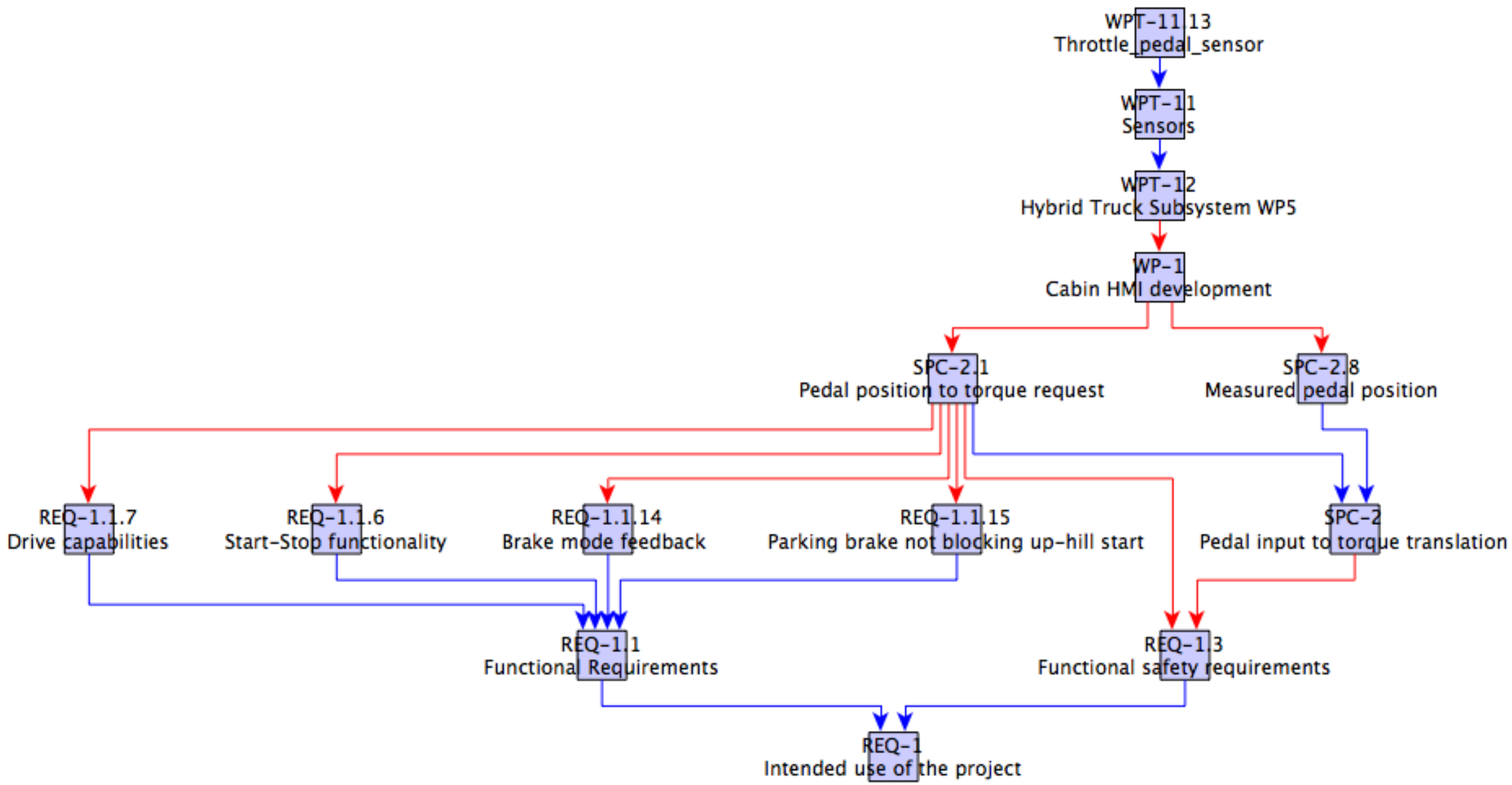
Traceability

- From any entity, dependency graph shows traceability and change impact
- Also shows incompleteness in repository (missing links, entities)
- Takes into account decomposition
- Activities can be concurrent/agile or not
- **Dependency tree used to approve entities in order of dependency**!

Dependency graph provides insight



Partial trees



Other features

- Teamwork: locking, repository, user profiles
- Glossary
- "Links": relationships between Entities reflect dependencies and decomposition
- Version/Configuration management: logging of changes, undo, import/export, attaching documents, …
- Document generation: read-only snapshot
- GANTT chart: display planning

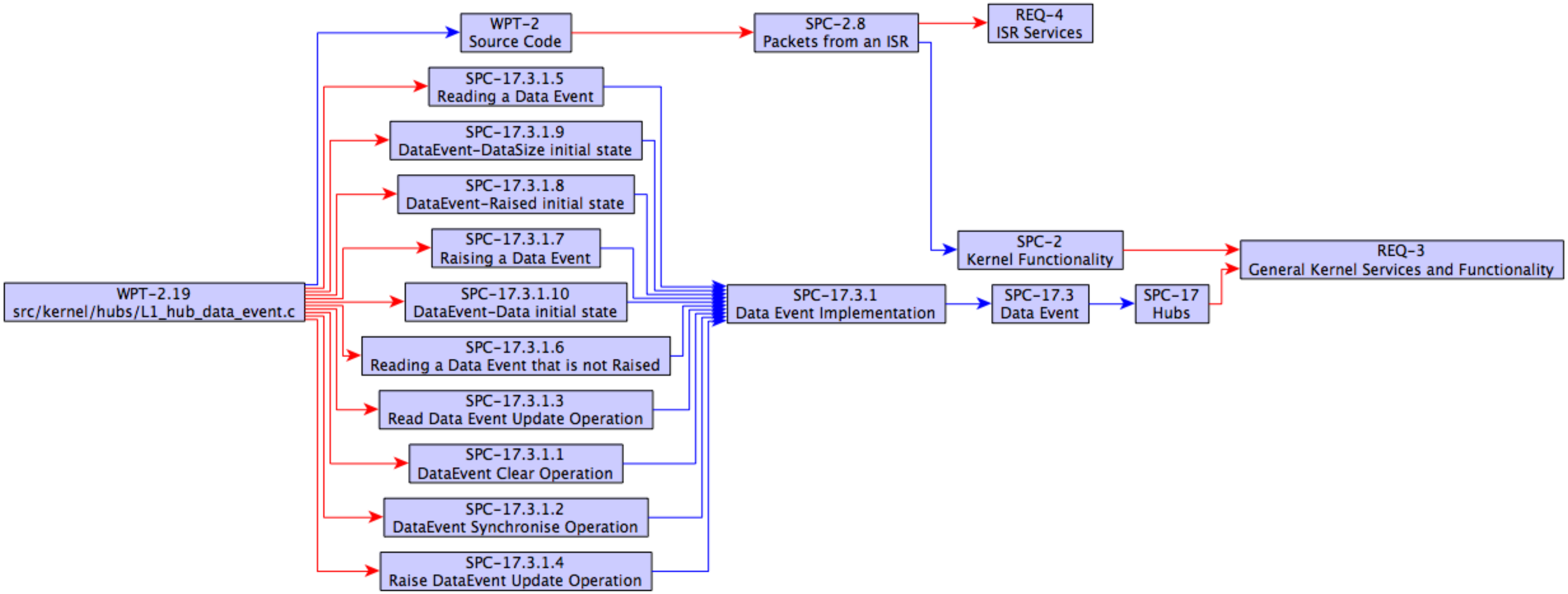
Validation of GoedelWorks

- Input: ASIL project of Flanders Drive
  - **A**utomotive **S**afety **I**ntegrity **L**evel
- Goal: develop common safety engineering process based on existing standards:
  - Automotive: off-highway, on-highway
  - Machinery
- IEC 61508, IEC 62061, ISO DIS 26262, ISO 13849, ISO DIS 25119 and ISO 15998
- 3500 Process Requirements, 355 STEPs, 100 WPTs,
- Other standards: customer specific

PRJ-1: OpenComRTOS Qualification Pack

- Formally developed network-centric RTOS
- Scope: Kernel + PowerPC-e600 HAL only (DO-178 accepted)
- LOC of kernel: 6550 lines of C and Assembly.
- Number of Entities in Project: 1280
  - Specifications: 307
- Number of Links in Project: 2840
- Number of tests: 337
  - Unit Tests: 175
  - Functional Tests: 162 (covering 99.8% of all lines, and 99.8% of all branches, for SP)
- Number of reports: 29
- PDF output: 2043 pages (excl. source test projects).

Source to REQ precedence tree



Impact analysis at the push of a button

23/10/17
Altreonic - From Deep Space to Deep Sea
30

ARRL: Assured Reliability & Resilience Level

Then came ARRL (0 to 7):

Assured Reliability and Resilience Level

- Criterion defines properties of components/ systems taking into account fault behavior
- Leads to the notion of anti-fragile systems

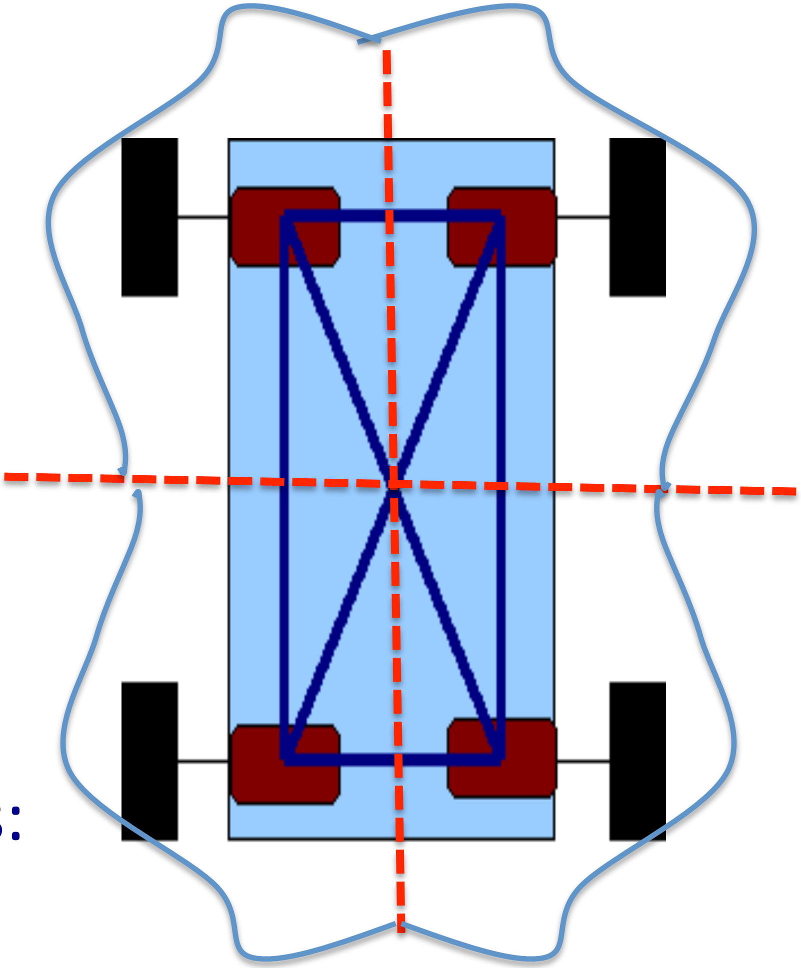
ARRL 0	It might work (use as is)
ARRL 3	ARRL 2 + goes to fail-safe or reduced operational mode upon fault (requires monitoring + redundancy) – fault behavior is predictable as well as next state
ARRL 7	The component (subsystem) is part of a **system of systems** and a **process is in place** that includes

VirtuosoNext: ARRL-3/4 at work

- Static programming is safer but:
  - Dynamic is norm: software always has errors
  - Hardware is almost perfect but will fail
  - External factors compromise the QoS
  - SoCs: very powerful, complex, single point of failure
- Need to physically isolate program segments from each other (error propagation)
- Fine grain space and time partitioning
- Overhead is very small
- RTOS real-time behavior not jeopardised
- Recovery in microseconds => RT fault-tolerance

Interacting Entities in the real-world of systems: KURT novel Modular and Redundant architecture

- Patent pending
- Combine reusable units
  - Economy of scale (COG!)
  - Redundancy (fault tolerant)
- Propulsion Unit =
  - Battery (LiOn, other) + motor
  - Suspension + wheels
  - Vectoring steer/drive by wire/4x4
- Smart environmental awareness:
  - Obstacle detection/avoidance
  - Assisted auto-navigation



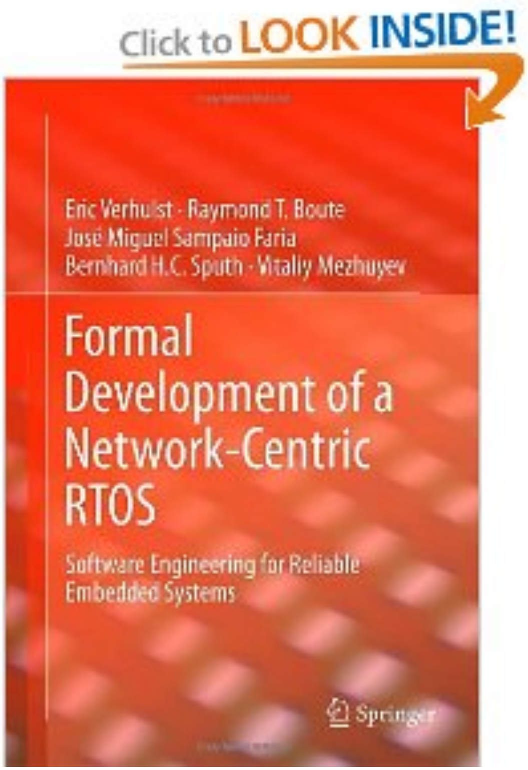
Interacting Entities: VirtuosoNext Designer

Formal methods?

- Can formal techniques help in getting the system / software right?
- Formal = proven = correct?
  - Beware of Goedel's theorema!
- OpenComRTOS project (2005):
  - Redevelop Virtuoso-like RTOS from scratch
  - Use formal techniques
  - Use of TLA+/TLC and a bit of UPPAAL

From FM to an environment

- Many goals:
  - Portability
  - OpenComRTOS kernel
  - Visual Designer
  - Event Tracer
  - System Inspector
  - Safe Virtual Machine
- Unexpected:
  - Code size shrunk 10x !
- Springer book:



Formal Development of a Network-Centric RTOS, *Software Engineering for Reliable Embedded Systems, Verhulst, E., Boute, R.T., Faria, J.M.S., Sputh, B.H.C., Mezhuyev, V*

Results

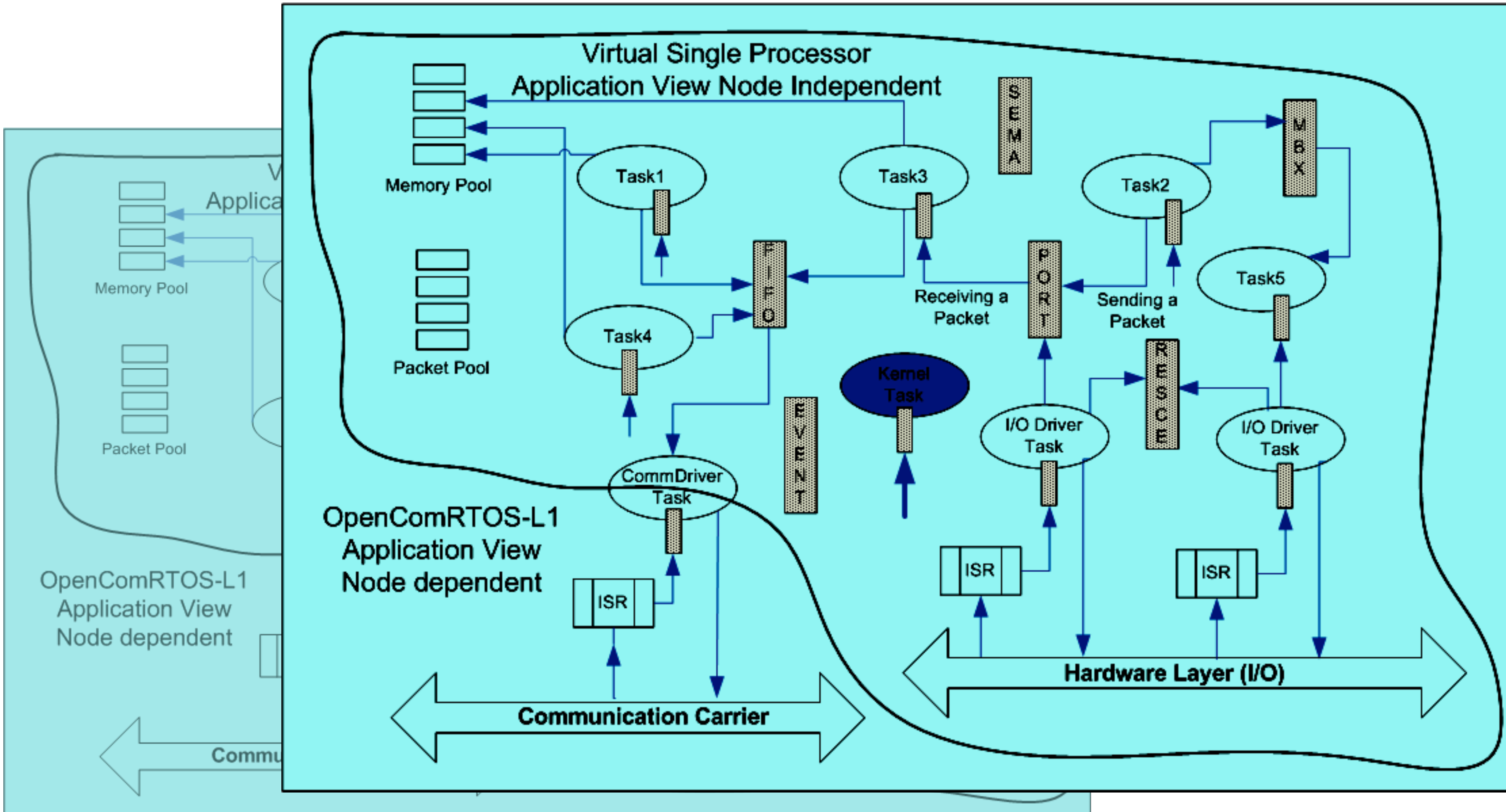
- Best use of formal techniques is not to verify and proof, certainly not source code
  - Change one line and one can start all over
- Best use: tool for abstract ~~modeling~~ thinking
  - Thinking about "correct" / "best" behavior
  - It helps the mind to undo brainwashing experiences (carpenter's problem: nail+hammer) & syndromes
  - Separation of concerns:
    - Clean and orthogonal architecture
- Code size dropped with factor 10 (=> 5 to 15 KB)

From modelling to programming

- Modeling is the core of engineering
  - Abstraction allows to think away from the implementation details
  - We can verify the model without the tedious work of programming
  - When done well => can be translated into different implementations
- Code generation can relieve us from the error-prone repetitive tasks in programming

VirtuosoNext Interacting Entities

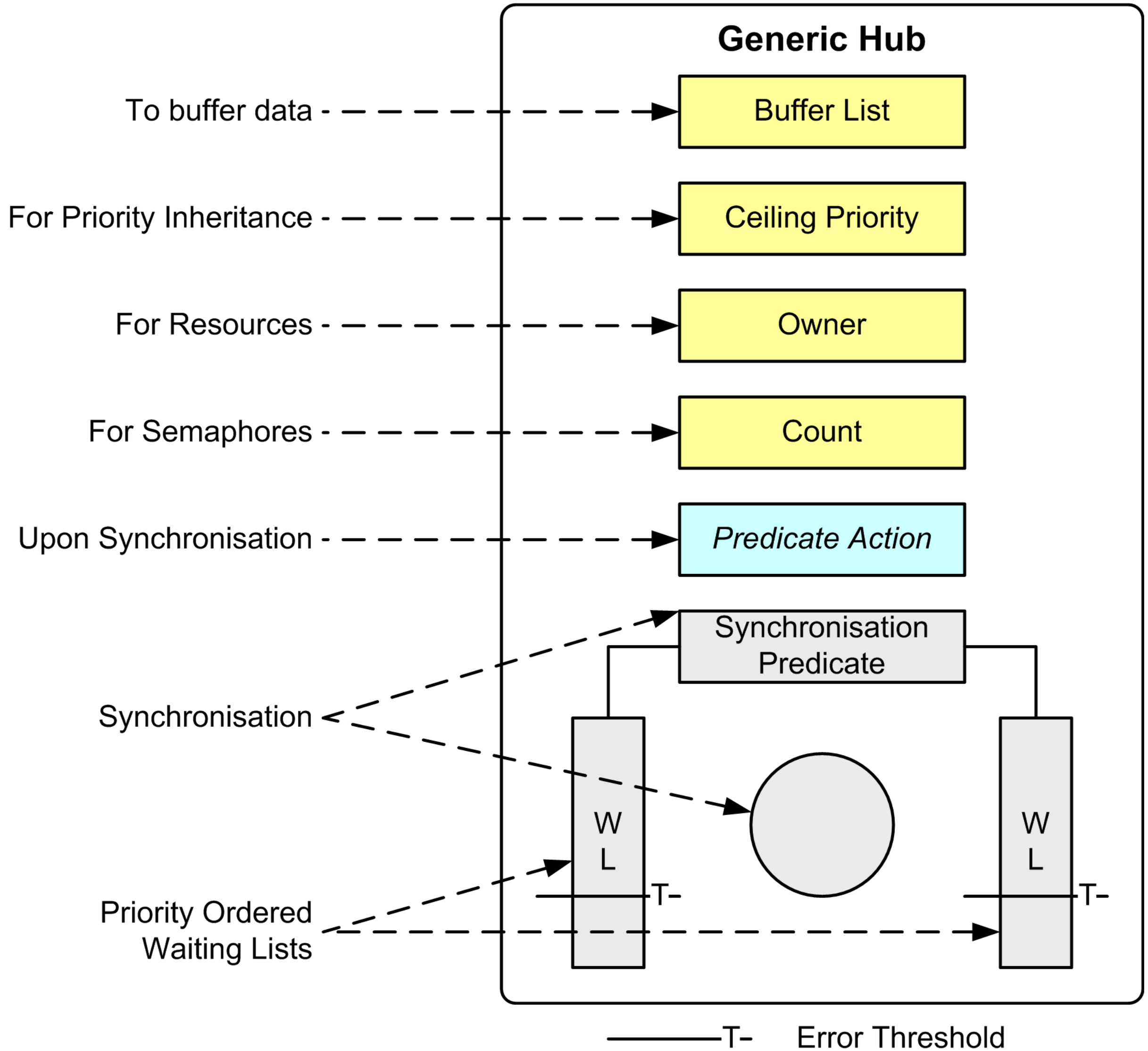
Any entity can be mapped anywhere in the target system



The Hub: this is not a channel

- Occam has channels, VirtuosoNext has Hubs
  - "pragmatic superset of CSP"
- Hub = channel + behavior (= 2 channels + process)
- Hub = synchronisation predicate (guard) + action = guarded atomic action
- Decouples Tasks (processors) fully
- Shared functionality saves a lot of code and errors! Lower certification efforts!
- Makes it easy to add new types of Hubs:
  - Event, Semaphore, FIFO, Resource, BlackBoard, …

Generic Hub model



Hub services

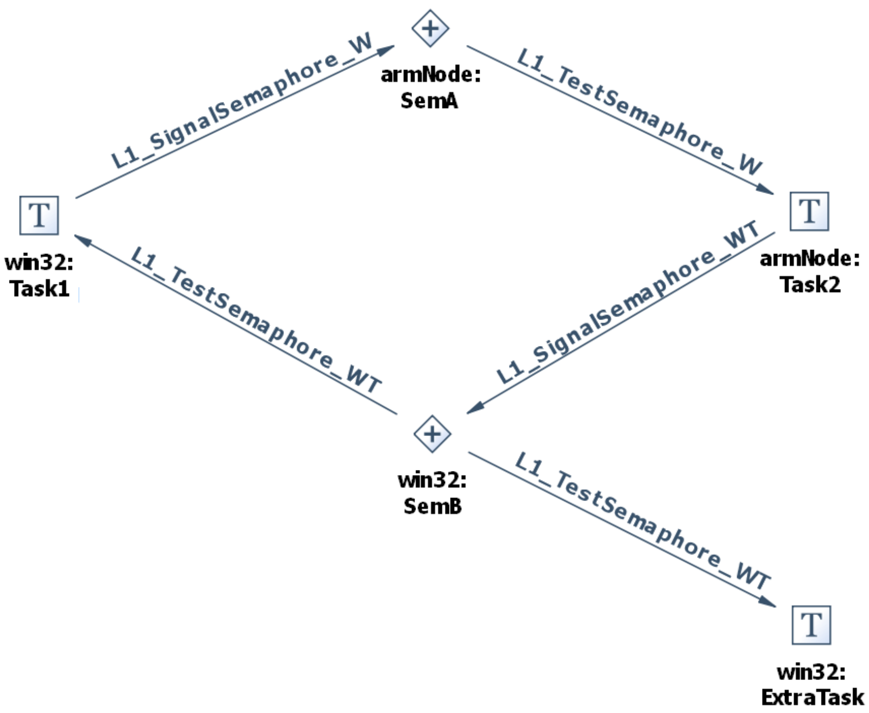
Event	Synchronise on Boolean condition (binary semaphore)
Semaphore	Synchronise on past Events (counting Semaphore)
Fifo	Data communication with buffering
Port	Exchange Packets
Resource	Entering / exiting a critical section (atomic access)
Data Event	Event with Data transfer upon synchronisation
Black Board	Protected shared data structure
Memory Block Queue	Priority sorted buffer list of Memory Blocks (SP only)

Interaction semantics

- _W: wait for Hub boolean condition to be true (often called blocking)
- _NW: test and return (RC_OK, RC_Fail)
- _WT: waiting with a timeout
- _A: asynchronous (SP only, for drivers).
  - Put a request and synchronise later

Ex.: Semaphore Hub Interactions

```
L1_SignalSemaphore_W();
L1_SignalSemaphore_NW();
L1_SignalSemaphore_WT();
L1_TestSemaphore_W();
L1_TestSemaphore_NW();
L1_TestSemaphore_WT();
```



Note:

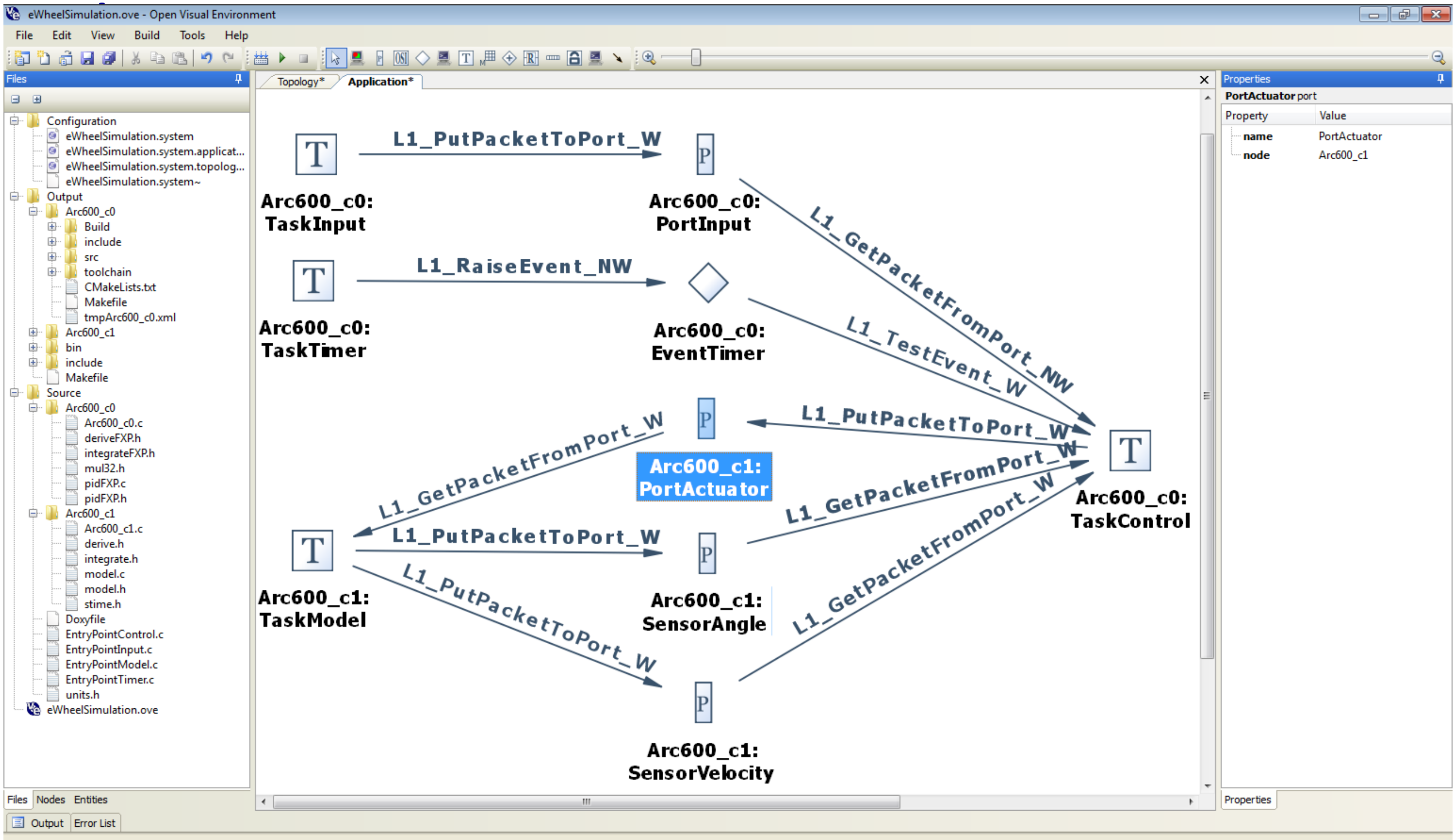
Source code becomes master to regenerate visual model

Virtual Single Processor model

- Separates two areas of concern:
  - Hardware Configuration (Topology View)
  - Application Configuration (Application View)
- Benefits:
  - Transparent parallel programming
  - System wide priority management

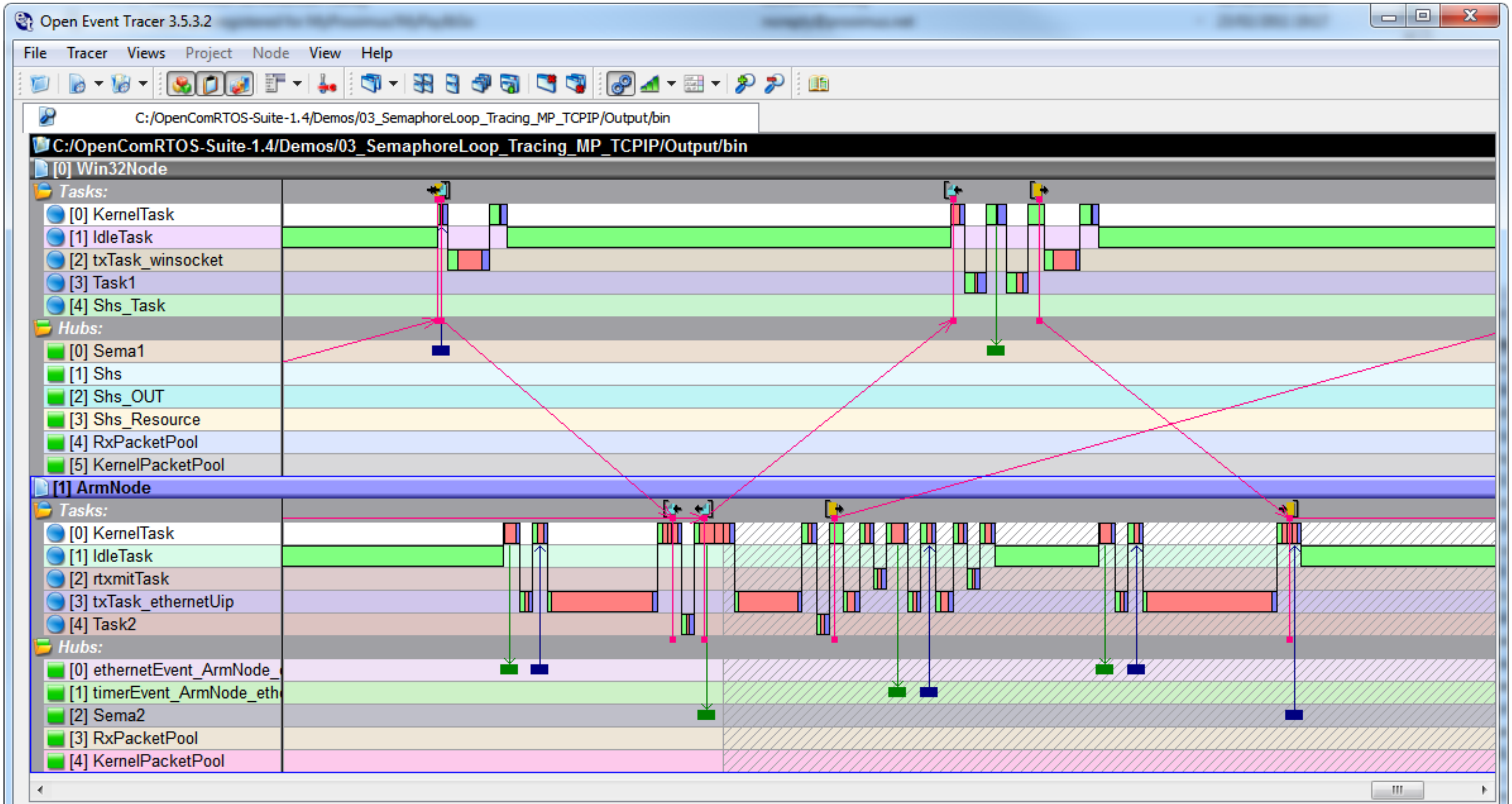
Visual Designer

Model visually, then regenerate model from source



Event Tracer

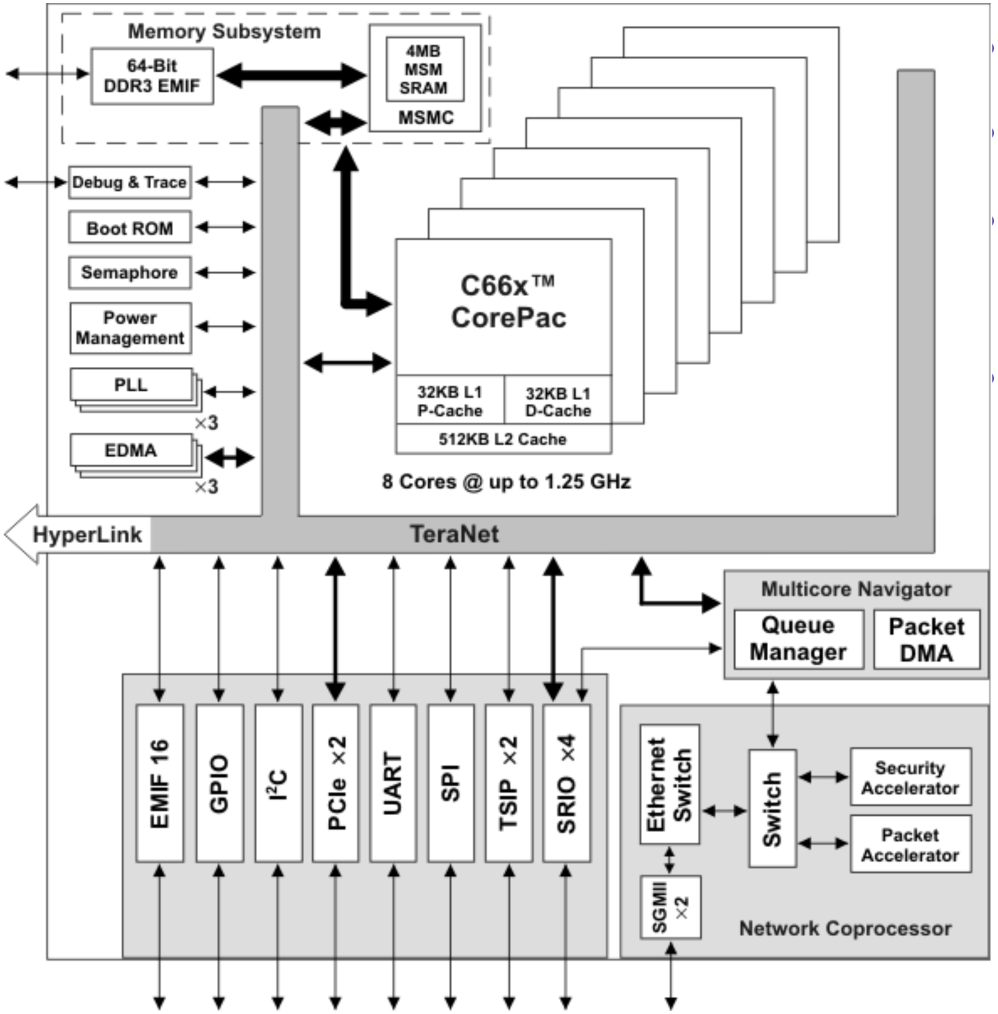
- Visualizes: Context Switches, Hub Interactions, Packet exchanges between Nodes.



Metamodels

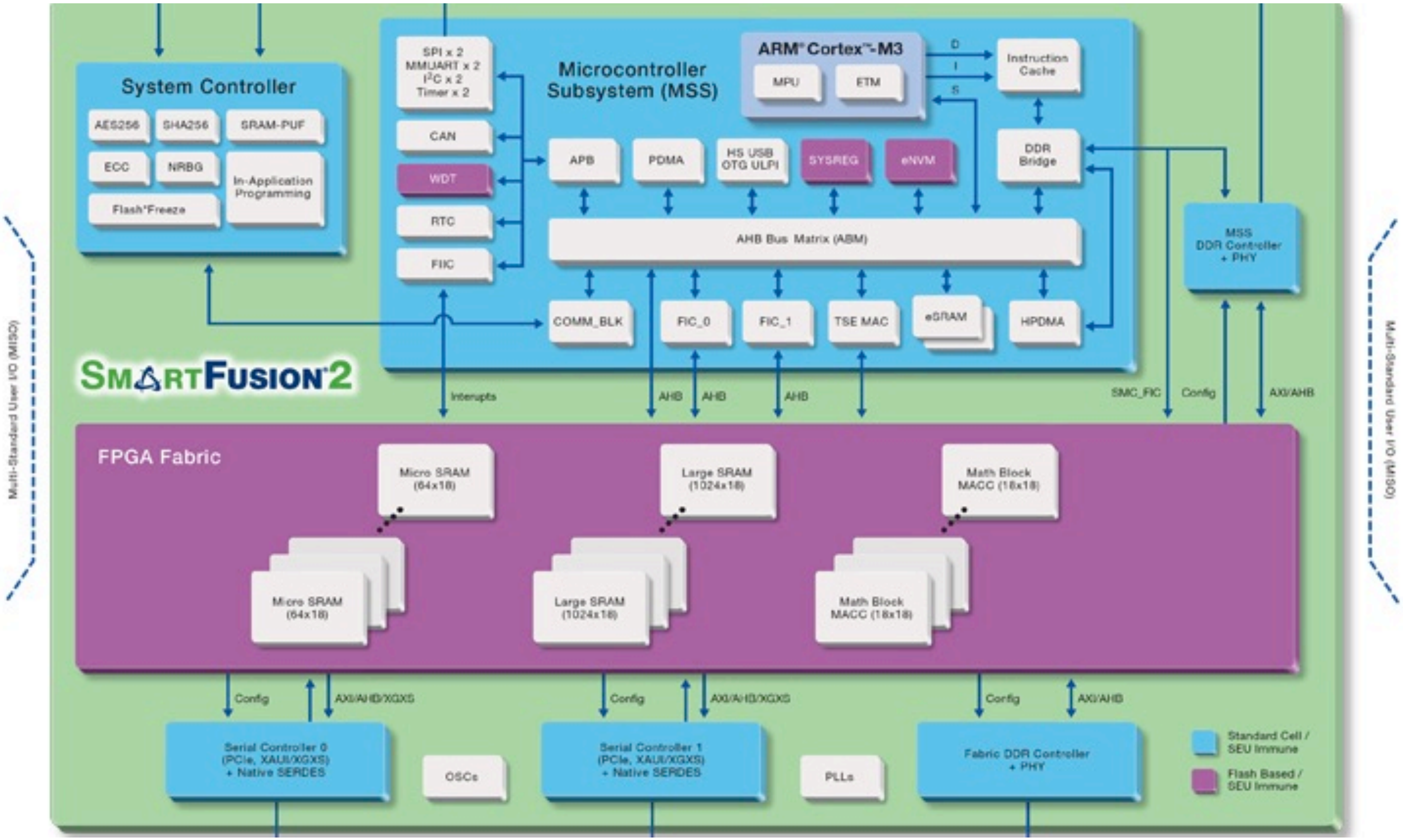
- Visual Designer generates code from visual models
- Runtime executables via CMake/gprbuild
- Target specific support (processor, BSP, service modules) isolated in metamodels (XML based)
- Hence, adding now targets is just adding a new target folder

TI TMS320C6678: a ROC (Rack On a Chip)

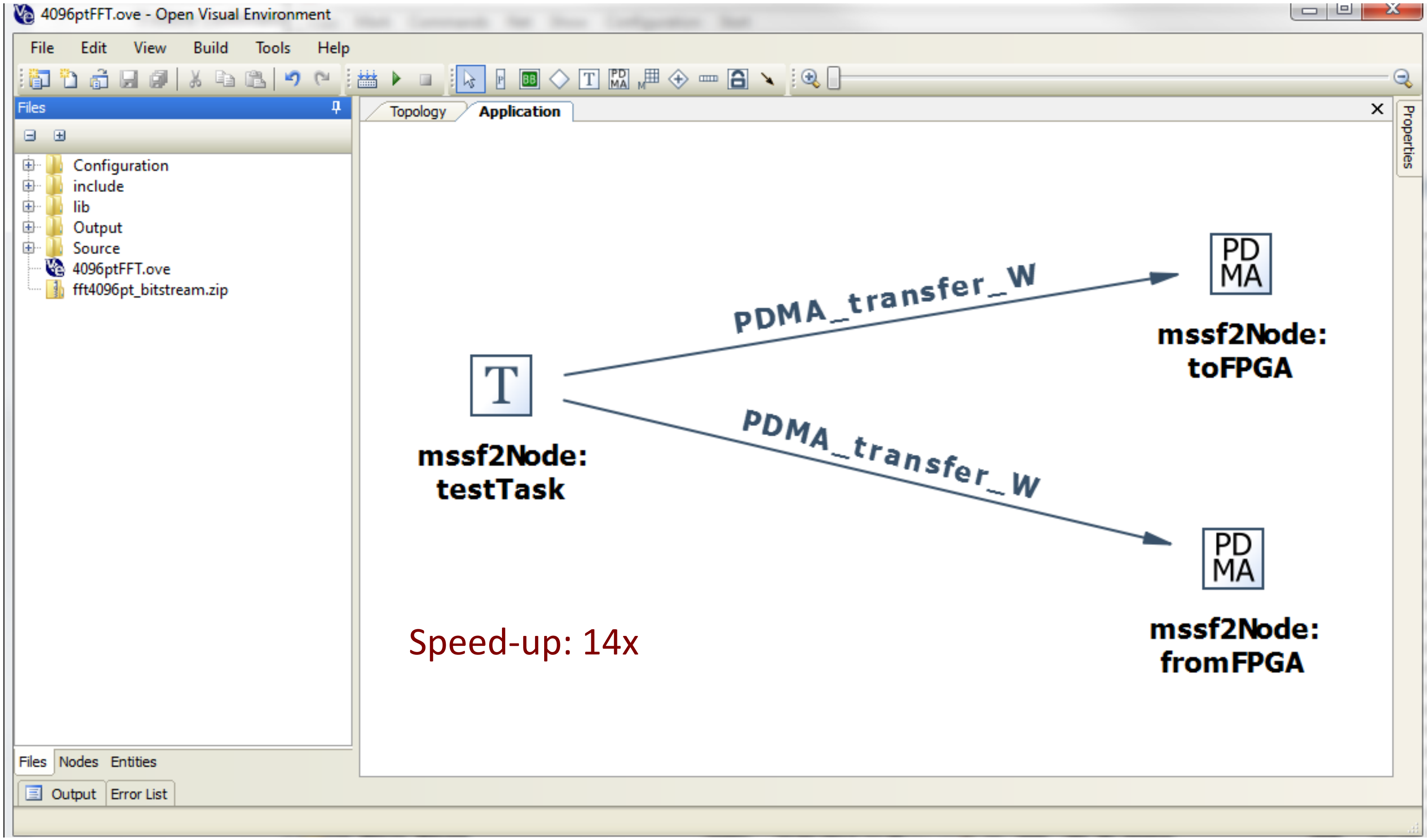


- OpenComRTOS code size: 5056 to 7648 Bytes
- Interrupt latency tot Task: 1367 cycles
- Task to Task switch: about 1125 cycles

Using FPGA as co-processor (SF-II)



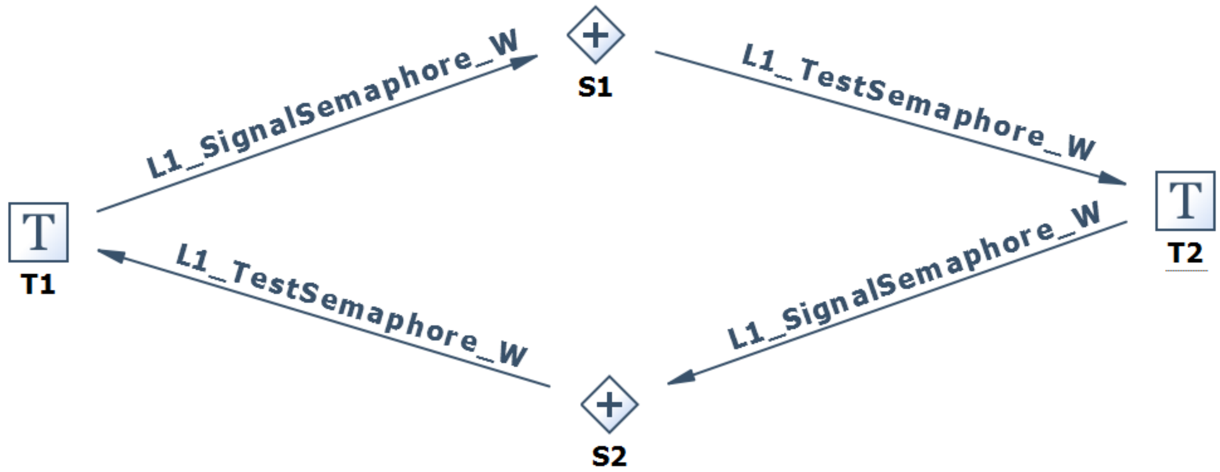
Hub as interface to “FPGA-server”



Some code sizes (-Os, bytes)

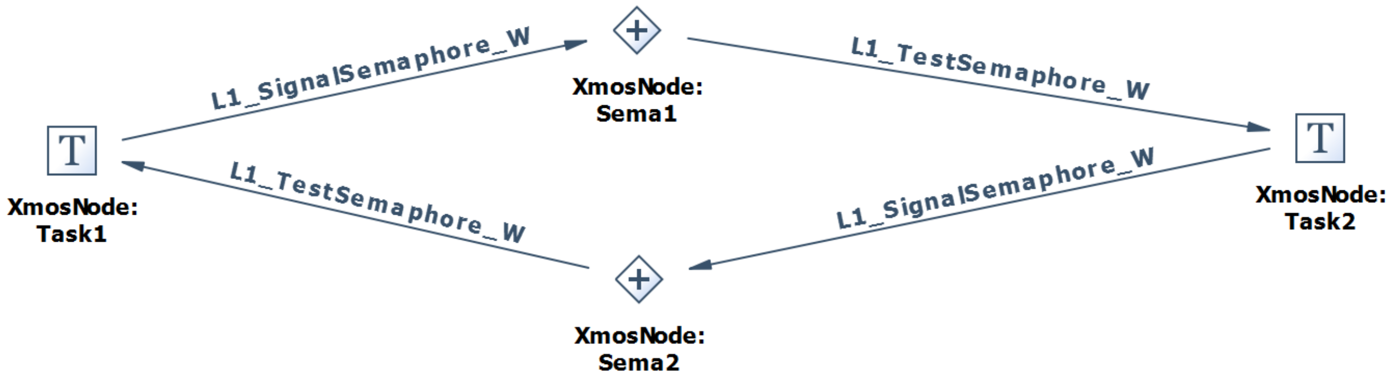
VirtuosoNext Version	1.4	1.4	1.6	1.6
CPU	ARM-M3	TI-C6678	ARM-M3	PPC-e600
Minimum	2564	5056	4496	9116
All services	4000	7648	8656	15724
	Only kernel code, excluding compiler runtime libs		Building a minimum application, includes compiler runtime libs	

Task Switching Figures



VirtuosoNext Version	1.4	1.4	1.6	1.6
CPU	ARM-M3	TI-C6678	ARM-M3	PPC-e600
Loop time in Cycles	2360	4470	2745	3826

Interrupt Latency Measurement

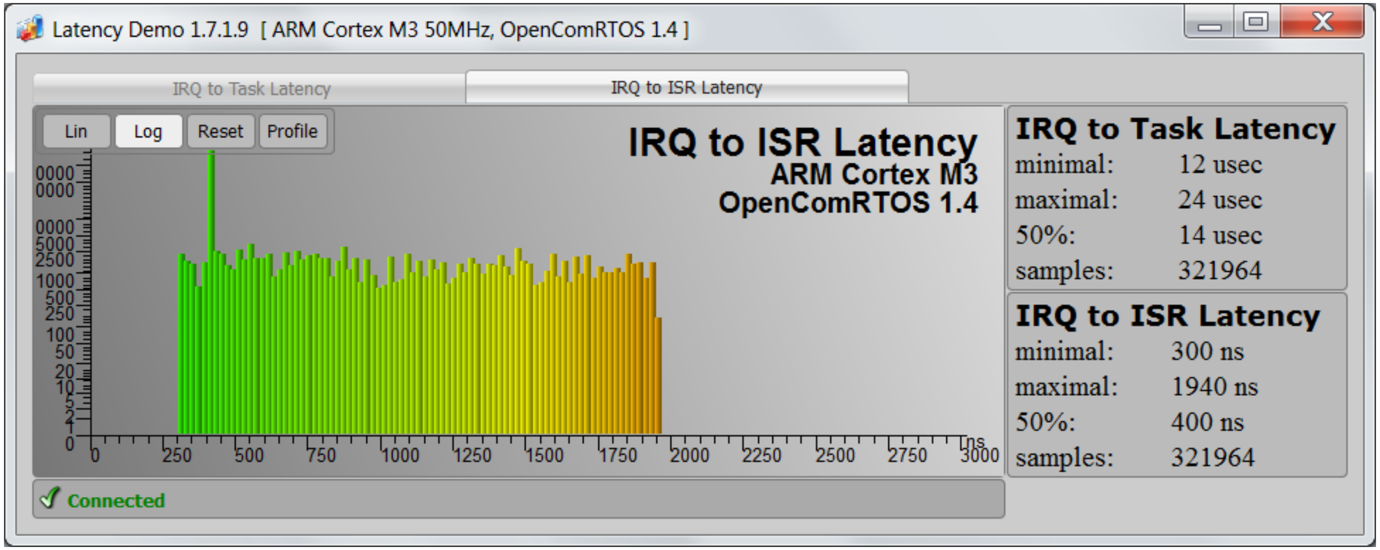


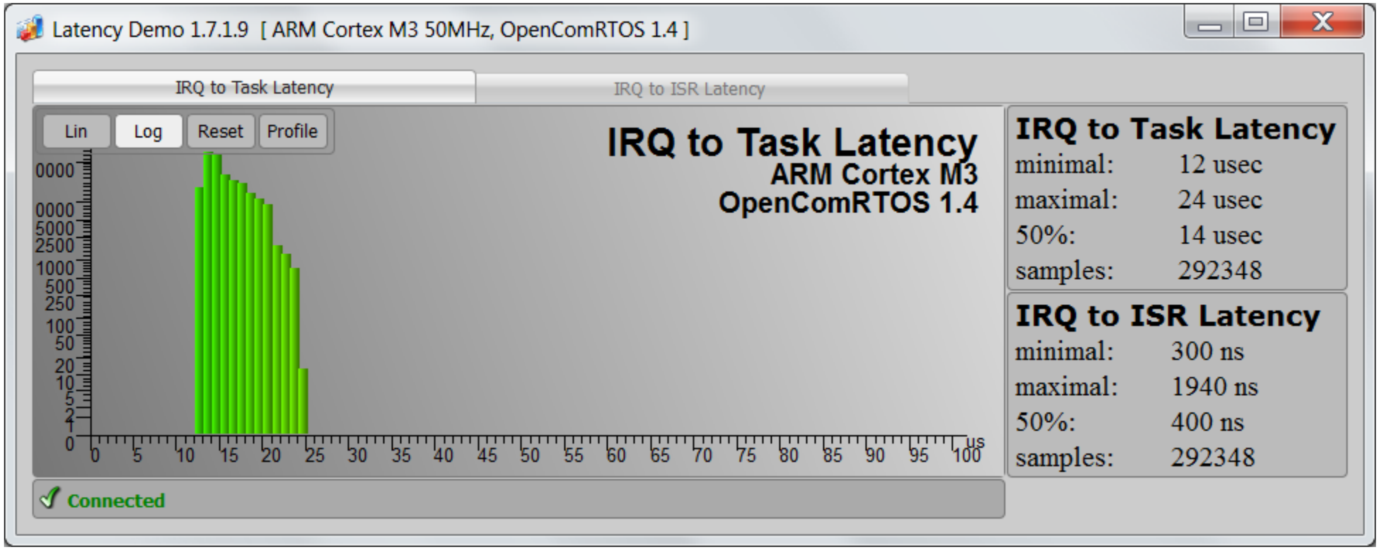
- IRQ 2 ISR: The time that elapsed between the IRQ and the first useful instruction of the ISR.
- IRQ 2 Task: The time that elapsed between the IRQ and the first useful instruction of a Task triggered by the ISR.
- Background loop as stress test

Interrupt Latency Figures

VirtuosoNext Version	**1.4***	**1.4***	**1.6**	**1.6**
CPU	ARM-M3	TI-C6678	ARM-M3	PPC-e600
IRQ to ISR	20	131	46	550**
IRQ to Task	600	1367	754	1990**
	*Direct interrupt		Generic Int handling + ** includes external interrupt.	

Interrupt Latencies: not a single number





Some data on Leon3 as target

LEON 3 codesize

	Elf .text segment in bytes
Kernel	21036
+ Task services	21060
+ Port hub services	21348
+ Event hub services	21536
+ Semaphore hub services	21844
+ Resource hub services	22264
+ BlackBoard hub services	24076
+ DataEvent hub services	24832
+ FIFO hub services	25468
+ MBQ hub services	27928

New VirtuosoNext 2.0 Fine Grain Space and Time Partitioning

VirtuosoNext RTOS support for safety and security

- Assumptions:
  - CPU hardware is very reliable
  - Software is correct
  - Most issues related to:
    - Data corruption
    - I/O
    - Numerical exceptions
    - Common mode issues are hardest
- Fault detection => prohibit error propagation
  - => fault recovery is time sensitive
  - Redundancy in space and time needed

Fine Grain: what does it mean?

- Issues with current hypervisor approaches:
  - Real-time becomes time-slicing in milliseconds
  - Unit of partitioning is complete application
  - Derived from server world
  - Requires lots of memory
- Fine-grain:
  - Task level memory protection
  - RTOS scheduler in order of priority keeps hard real-time properties
  - Memory overhead very limited

VirtuosoNext

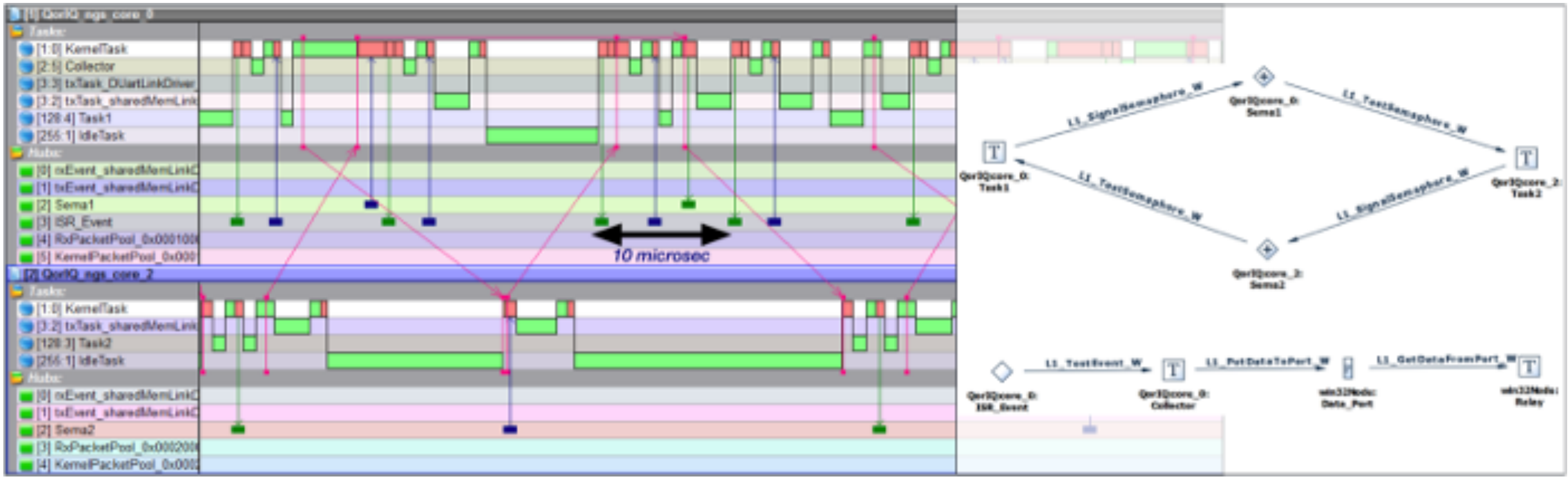
- Uses MPU+MMU to isolate tasks individually
- Code is target independent
- Code is MP/SMP transparent
- Error trapping and recovery at task level
- As fast as standard RTOS (microseconds)
- Code as small as 10 – 30 kBytes
- Supports ARM Mx/Rx - Ax, Freescale PPC,
- TI C6678 at core level
- Feasible for Leon

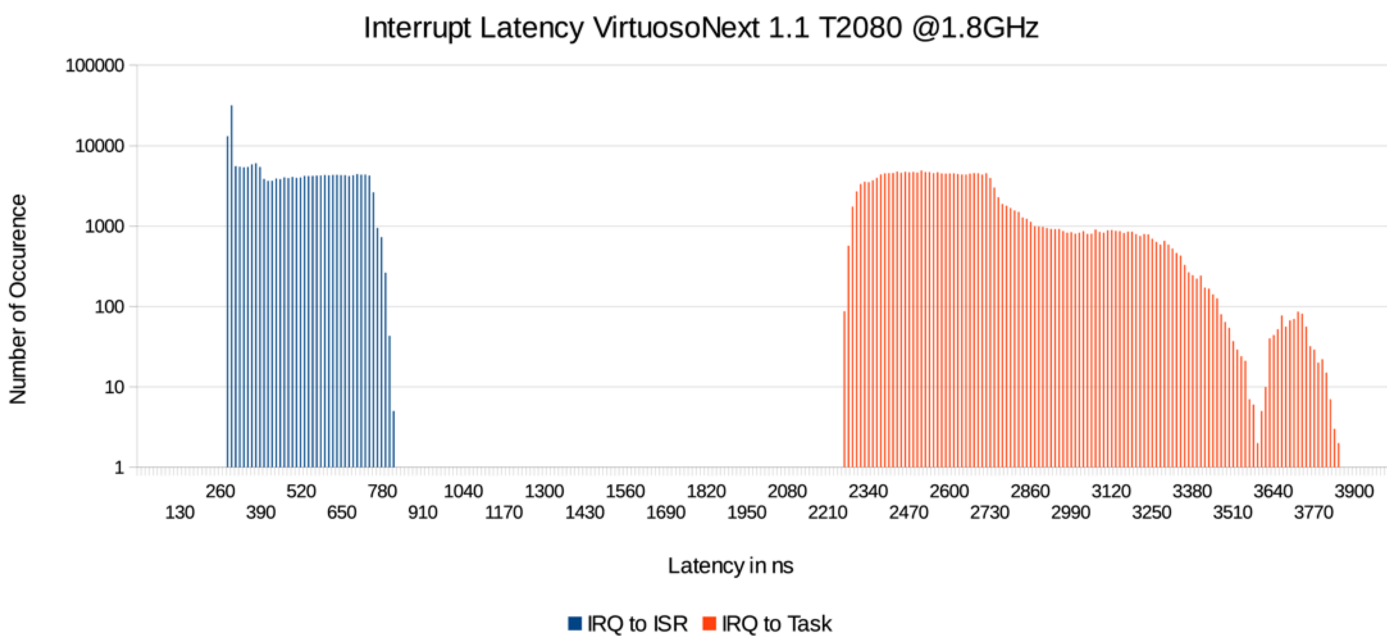
Minimal impact on code size

Target	Non-Partioned	Partitioned
ARM-Cortex-M3 (@50MHz)	8,656	11,564
ARM-Cortex-A9 (@700MHz)	15,144	21,844
C6678 (@1.25GHz)	26,448	n.a.
T2080 (@1.8GHz)	37,224	38,504

- Benefits: run code in L1 or L2 cache!

Latency measurements





Minimal impact on interrupt latency

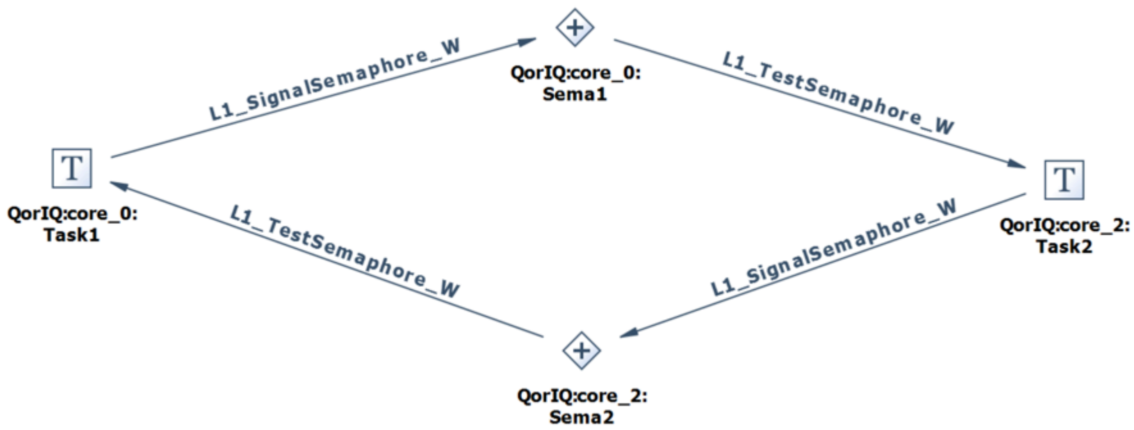
Table 5 Minimal and maximal IRQ to ISR Latencies in nanoseconds

Target	Non-Partioned	Partitioned
ARM-Cortex-M3 (@50MHz)	780 - 2500	960 - 4920
ARM-Cortex-A9 (@700MHz)	100 - 314	138 - 1150
C6678 (@1.25GHz)	160 - 260	n.a.
T2080 (@1.8GHz)	286 - 793	286 - 819

Table 6 Minimal and maximal IRQ to Task Latencies in microseconds

Target	Non-Partioned	Partitioned
ARM-Cortex-M3 (@50MHz)	15.0 - 35.0	16.0 - 39.0
ARM-Cortex-A9 (@700MHz)	0.994 - 2.182	1.726 - 4.228
C6678 (@1.25GHz)	936	n.a.
T2080 (@1.8GHz)	2.158 - 3.705	2.262 - 3.848

Semaphore loop times



In microseconds

Target	Non-Partioned	Partitioned
ARM-Cortex-M3 (@50MHz)	54.60	58.90
ARM-Cortex-A9 (@700MHz)	23.65	30.39
C6678 (@1.25GHz)	2.81	n.a.
T2080 (@1.8GHz)	5.64	6.01

VIRTUOSONEXT 2.0
ON FREESCALE T2080 (E6500)

Developed in EuroCPS NOFIST project (Thales)

Absolute time-based Scheduling

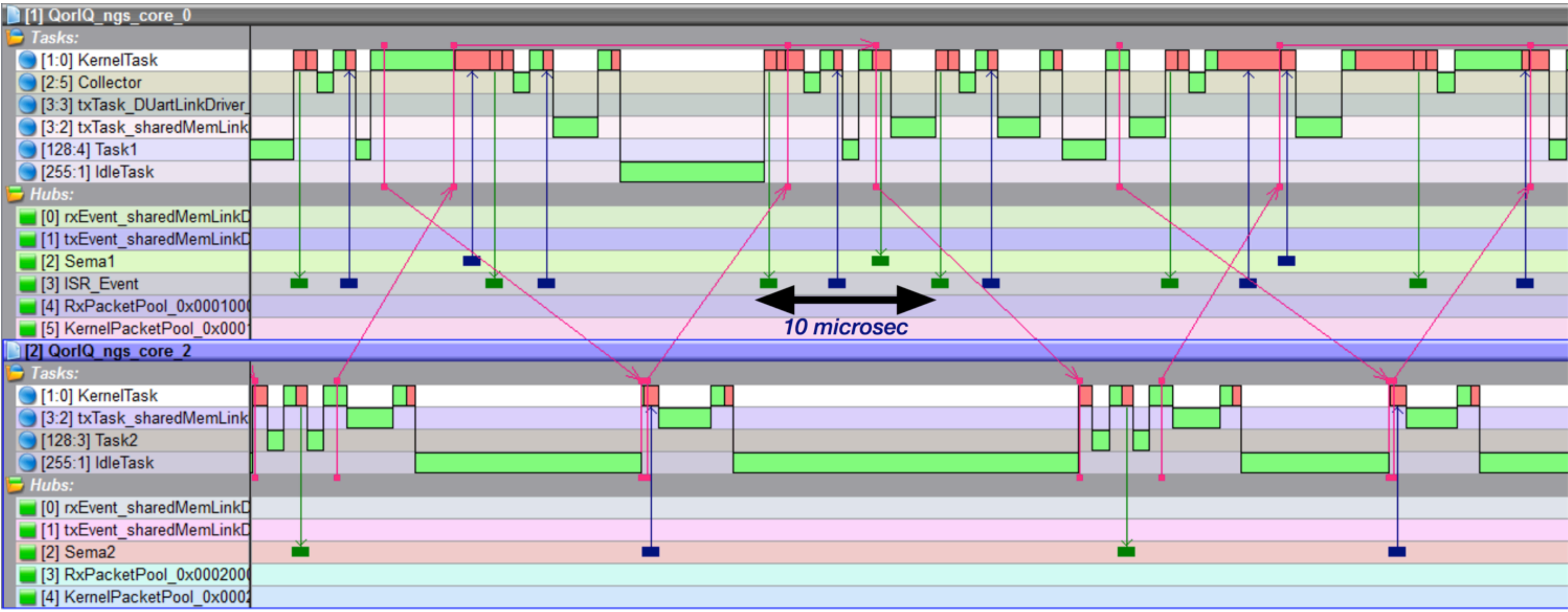
- Issue:
  - Tasks need to be scheduled at precise points in time.
  - Current L1_WaitTask_WT(L1_Timeout timeout) is relative and imprecise.
- Solution:
  - Periodic timer tick to maintain system clock.
  - Timer tick, typically 1ms, can be as low as 10usec (T2080).
  - Kernel maintains list of Tasks waiting to be rescheduled.
  - Tick causes the wake-up of a waiting Task, at the right point in time.

SMP Support 1

- Solution:
  - All Cores of the T2080 SoC share the same set of symbols.
  - VirtuosoNext assumes unique symbol names per Node (one instance per Core).
- Solution:
  - Each Core gets its own set of the necessary global variables, whereby the symbol names have the Node-Id appended.
  - At the SoC-Level arrays are provided, whereby pointers to the Core specific global variables are stored at the offset that is corresponding to the Core-Number.
  - The Kernel uses the Core-Number to find the correct instance of
  - the global variable for the Core it executes on.

Trace with 10 usec ticker

- - Distributed semaphore loop on 2 nodes



SMP Support 2

- Challenge:
  - Modeling of the SoC complexity
  - Complexity of the Code-generator
- Solution:
  - NodeGroup-Metamodels, which consists of:
    - Node-, Netlink-Driver-, and Netlink-Instances.
  - Visual Designer can instantiate a NodeGroup and utilise the Nodes specified by them.
  - ProjectGen generates an intermediate XML-File for each Node of the NodeGroup and one for the NodeGroup. From these the final code and the build system are generated

SMP Support 3

- Challenge:
  - Building a multi-node binary.
- Solution:
  - Each Node is compiled and linked into a static library.
  - The NodeGroup combines these libraries, with the Kernel-libraries and the bootloader to create the bootable ELF-File.

Support for fine-grain partitioning 1

- Each Task has its own .data and .bss segments in the linker script.
- Each segment gets aligned to a page-size supported by the PowerPC-e6500-MMU using the SectionAnalyser tool.
- Two privilege levels:
  - L1_TASK_PRIV_USER: Can only access their own memory regions listed above. No permission to access device memory.
  - L1_TASK_PRIV_SUPERVISOR: Can access any memory location, including device memory.

Support for fine-grain partitioning 2

- Memory-Regions permanently defined:
  - Shared Code Region: Code shared by all Tasks on all Cores of the SoC.
  - Shared Data Region: Data shared by all Tasks on all Cores of the SoC.
  - Shared BSS Region: BSS shared by all Tasks on all Cores of the SoC.
  - Shared Core Data Region: Data specific to the Tasks on one Core.
  - Shared Core BSS Region: BSS specific to the Task on one Core.
- Access Permissions per Region:
  - Code: Read Only
  - Data / BSS: Non-Executable

Support for fine-grain partitioning 3

- Each Task has its own protected Memory-Regions:
  - Stack, Heap, Context, Task Control Record.
- Task specific Memory-Regions:
  - Data: Task specific Data.
  - Bss: Task specific Bss.
- The context switch reconfigures the MMU to the Task specific Memory-Regions during a context switch.

Support for fine-grain partitioning 4

- Memory-Groups, may consist of: Tasks, Hubs, Components, sharing the same memory region.
- Members of a Memory-Group may access each other's Memory-Regions. Thus:
    - Weaker Protection for individual Tasks.
    - Stronger Protection for Hub Data.
    - Stronger Protection for Components, which now have their data structures in Component-Private Memory-Region.

SMP Support 2

- Challenge:
  - Modeling of the SoC complexity
  - Complexity of the Code-generator
- Solution:
  - NodeGroup-Metamodels, which consists of:
    - Node-, Netlink-Driver-, and Netlink-Instances.
  - Visual Designer can instantiate a NodeGroup and utilise the Nodes specified by them.
  - ProjectGen generates an intermediate XML-File for each Node of the NodeGroup and one for the NodeGroup. From these the final code and the build system are generated.

SMP Support 3

- Challenge:
  - Building a multi-node binary.
- Solution:
  - Each Node is compiled and linked into a static library.
  - The NodeGroup combines these libraries, with the Kernel-libraries and the bootloader to create the bootable ELF-File.

Support for fine-grain partitioning 1

- Each Task has its own .data and .bss segments in the linker script.
- Each segment gets aligned to a page-size supported by the PowerPC-e6500-MMU using the SectionAnalyser tool.
- Two privilege levels:
  - L1_TASK_PRIV_USER: Can only access their own memory regions listed above. No permission to access device memory.
  - L1_TASK_PRIV_SUPERVISOR: Can access any memory location, including device memory.

Support for fine-grain partitioning 2

- Memory-Regions permanently defined:
  - Shared Code Region: Code shared by all Tasks on all Cores of the SoC.
  - Shared Data Region: Data shared by all Tasks on all Cores of the SoC.
  - Shared BSS Region: BSS shared by all Tasks on all Cores of the SoC.
  - Shared Core Data Region: Data specific to the Tasks on one Core.
  - Shared Core BSS Region: BSS specific to the Task on one Core.
- Access Permissions per Region:
  - Code: Read Only
  - Data / BSS: Non-Executable

Support for fine-grain partitioning 3

- Each Task has its own protected Memory-Regions:
  - Stack, Heap, Context, Task Control Record.
- Task specific Memory-Regions:
  - Data: Task specific Data.
  - Bss: Task specific Bss.
- The context switch reconfigures the MMU to the Task specific Memory-Regions during a context switch.

Support for fine-grain partitioning 4

- Memory-Groups, may consist of: Tasks, Hubs, Components, sharing the same memory region.
- Members of a Memory-Group may access each other's Memory-Regions. Thus:
  - •Weaker Protection for individual Tasks.
  - •Stronger Protection for Hub Data.
  - •Stronger Protection for Components, which now have their data structures in Component-Private Memory-Region.

Support for fine-grain partitioning 5

- VirtuosoNext's approach,
  - Prevents that Tasks access each others private memory. Each Task is in its own "Partition".
  - Tasks / Partitions exchange data using safe copy semantics, via Hubs (" pass by value")
  - Partition boundaries can be extended using Memory-Groups.
  - Context switch times stay deterministic independently of the size of partition.

Support for real-time recovery

- How:
  - Trapping exceptions.
  - Reinitialising the Task (Stack) after an exception has been trapped.
  - AbortHandler as secondary entry point, automatically invoked.
  - Restarting the Task after the AbortHandler has terminated.
  - Possibility to recover the previous Task data, using a (Blackboard) Hub.

Enabling non-stop real-time

- Fine grain partitioning isolates faults and errors
- Fast recovery = no disruption of application
- Consistent data can be preserved
- Small grain size + recovery = no reboot
- Additional redundancy possible:
  - Duplication in time and space
  - Triplication and diversity (transparent) (common mode failures)
- Static code generation, dynamic handling of fluctuations and runtime faults

Results

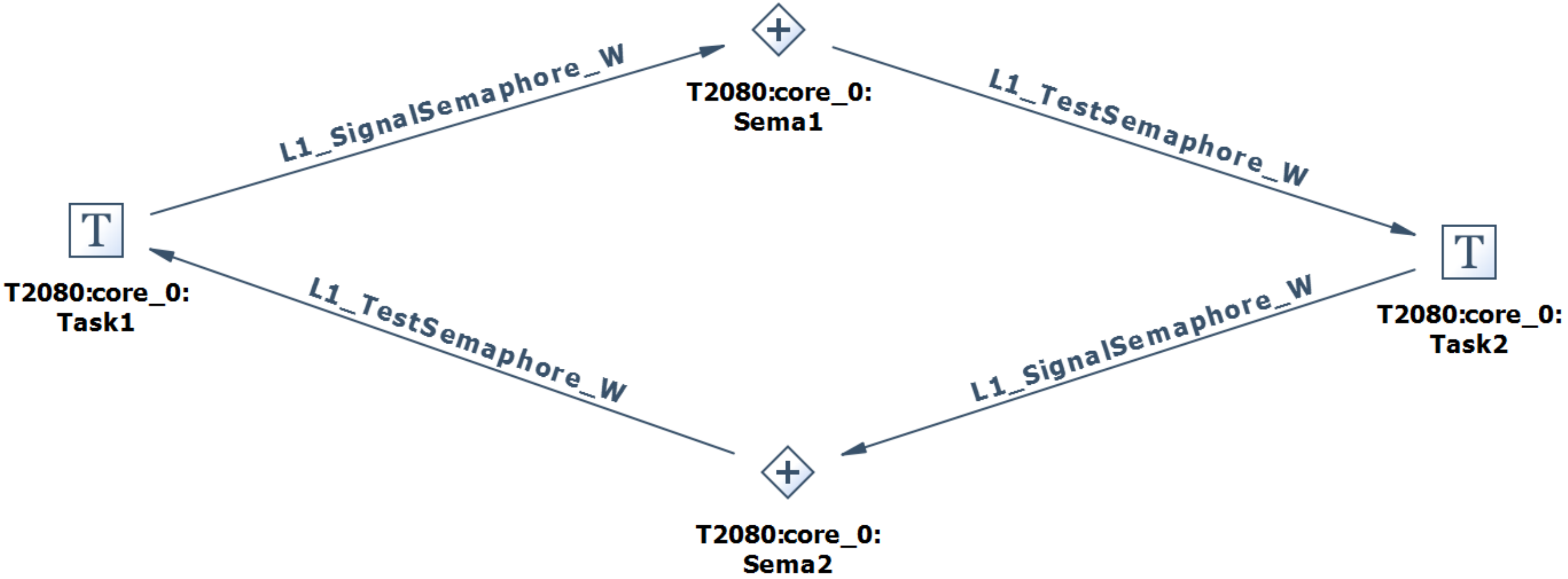
- Code Size
- Context Switch Performance
- Interrupt Latency Measurements
- Abort Handling Performance

Codesize (executable images)

	Size of the .text segment in Bytes
Baseline	17328
+ Task services	37708
+ Port-Hub services	38220
+ Event-Hub services	38460
+ Semaphore-Hub services	38820
+ Resource-Hub services	39188
+ BlackBoard-Hub services	40340
+ DataEvent-Hub services	41348
+ FIFO-Hub services	42148
+ MemoryBlockQueue-Hub services	44372

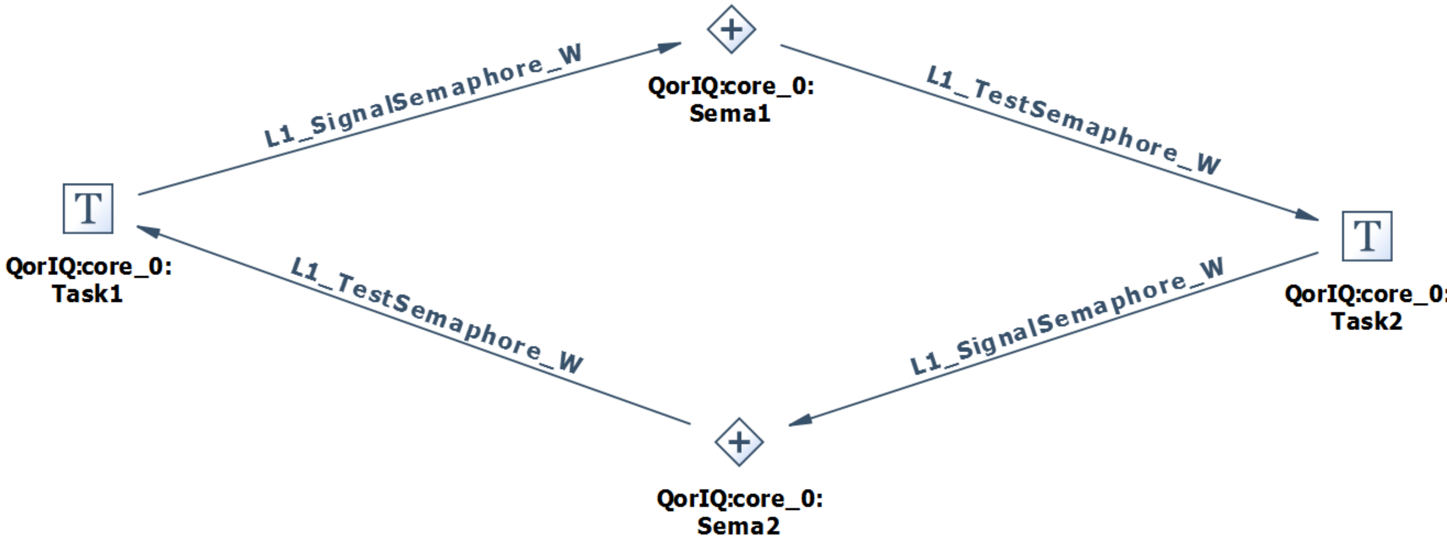
Context Switch Performance

- Semaphore Loop Single Processor:
  - Non-Partitioned: 5.64usec
  - Partitioned: 6.01usec

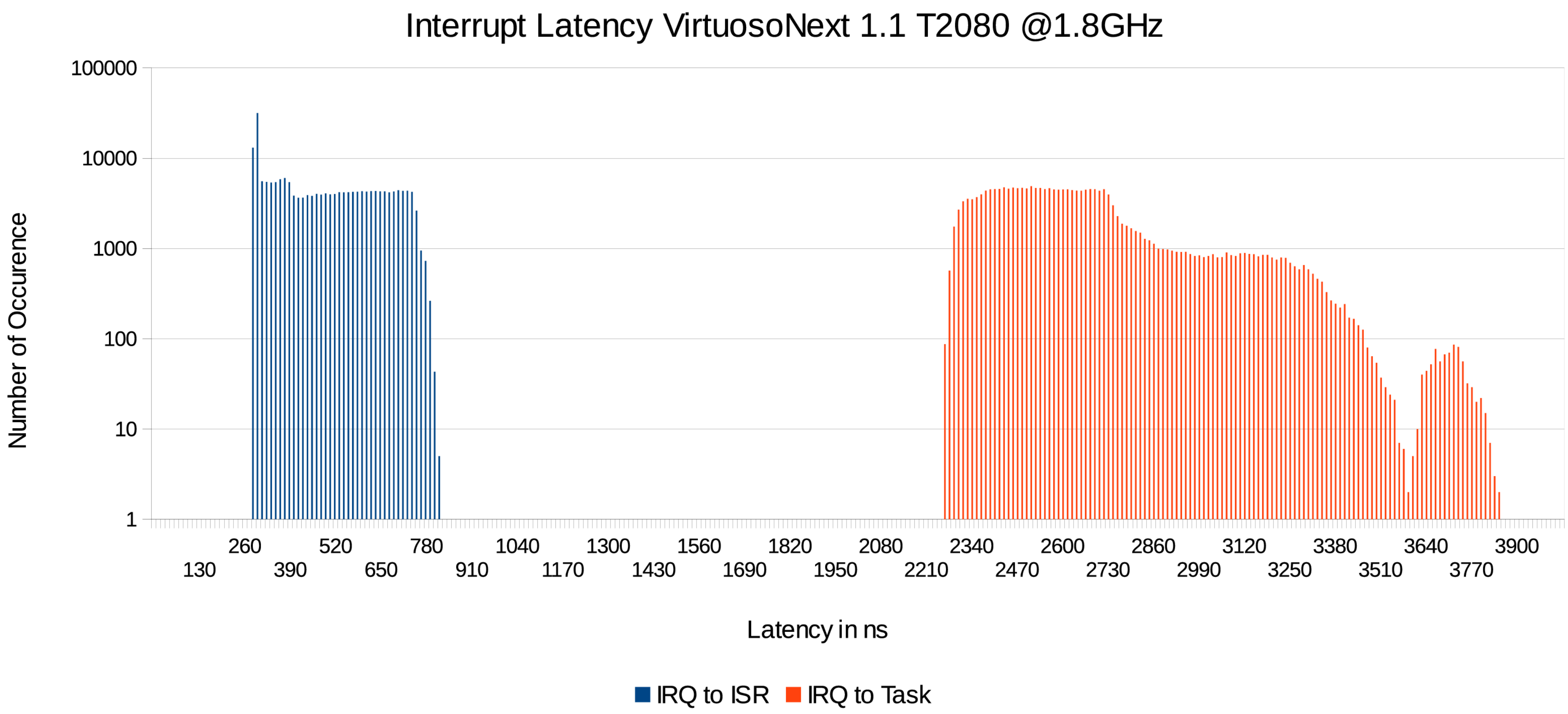


Interrupt Latency Performance

- IRQ to ISR Latency: Non-Partitioned: 286ns – 793ns, Partitioned: 286ns - 819ns
- IRQ to Task Latency: Non-Partitioned: 2.158usec – 3.705usec; Partitioned: 2.262usec – 3.848usec



Interrupt Latency Partitioned



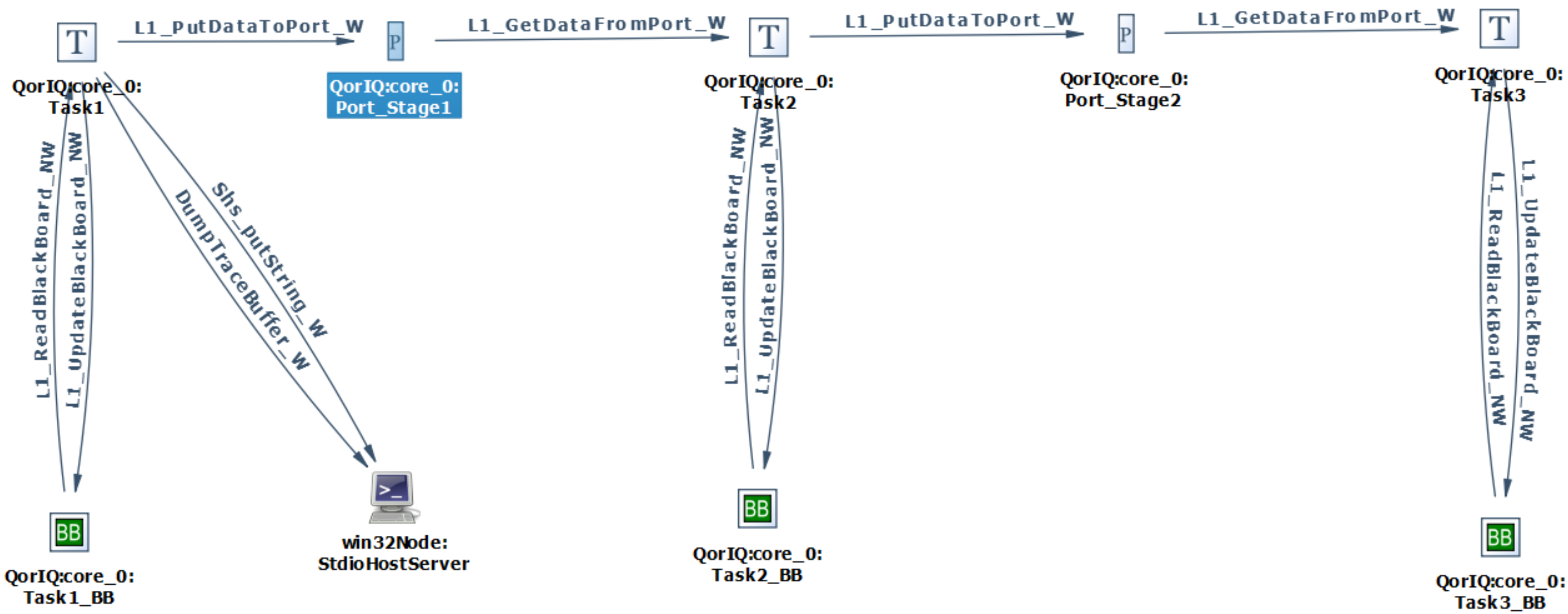
Real-time recovery: microseconds

Performance: From failure causing instruction till Task is recovered 2.3usec (172 75MHz ticks) expire.

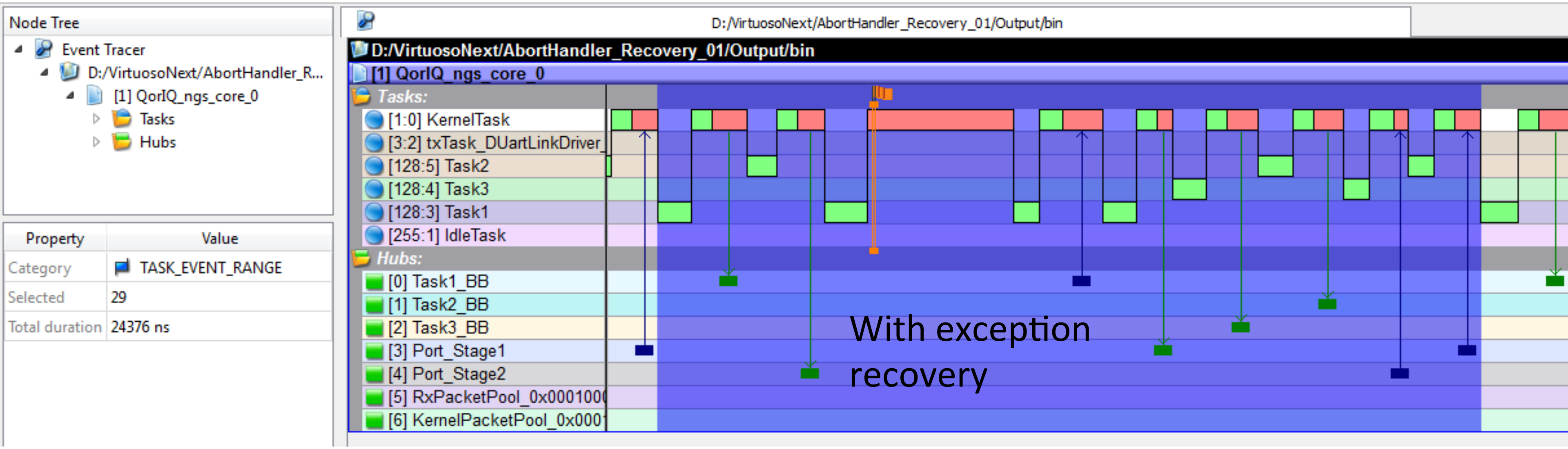
Measurement Setup:

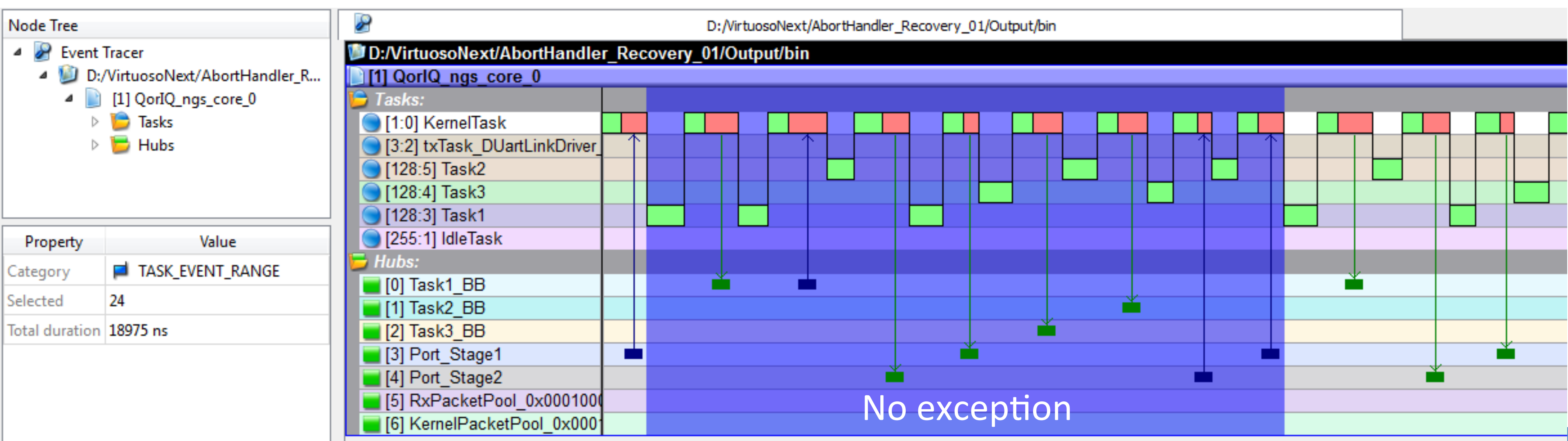
- Time-stamp.
- Exception-Handler invoked.
- Kernel-Task scheduled.
- Offending Task stopped and restarted.
- Task Abort-Handler (empty) executed.
- Task Entry Point reached.
- Time-stamp.

Real-time recovery: test set-up



Real-time recovery: event trace





VIRTUOSONEXT 2.0
ON ARM- (MX, RX)

Support for ARM

- Available for all ARM-Cortex M / R
- Selected ports on ARM Ax

Codesize

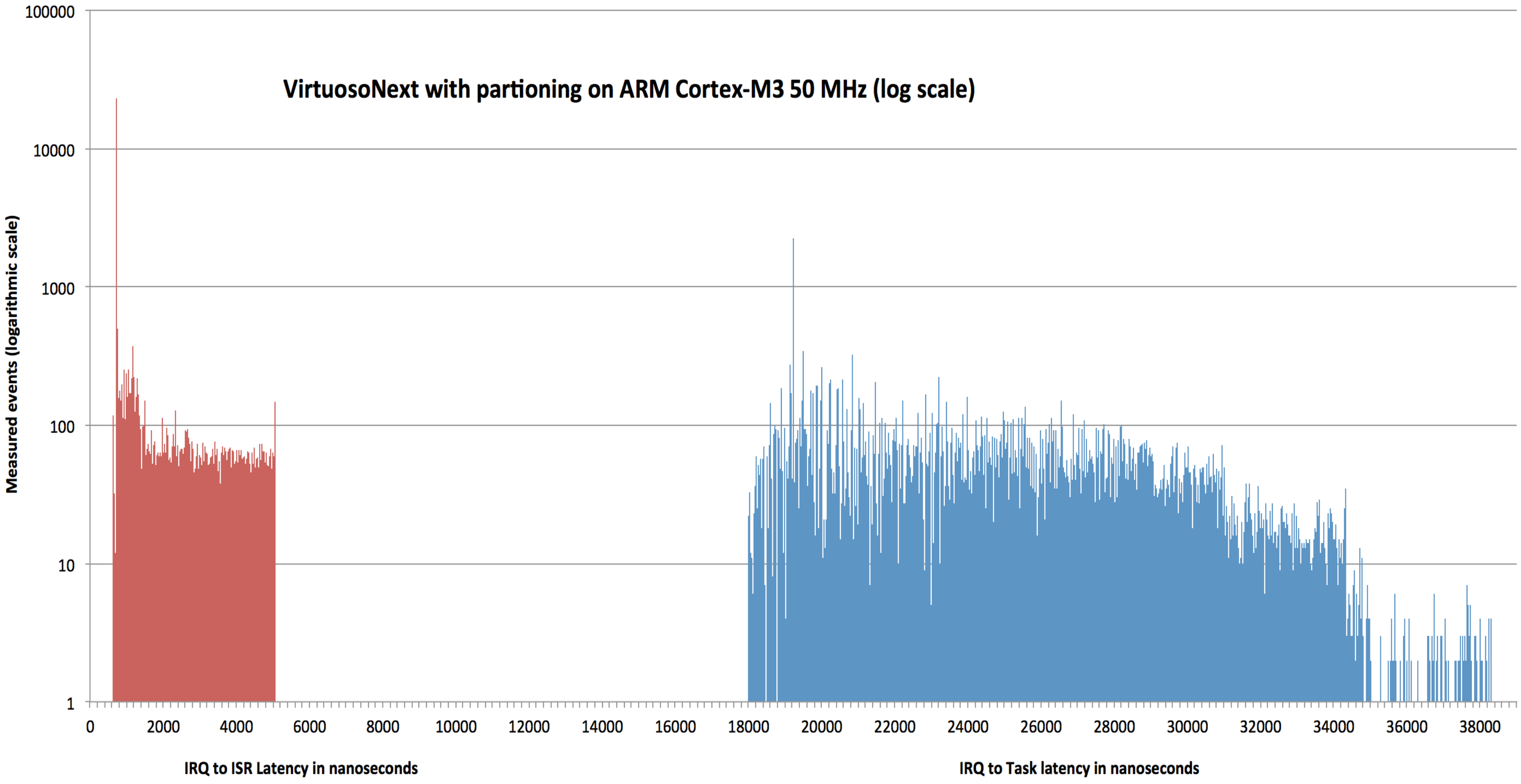
- Measured on ARM-Cortex M3
- Minimal kernel: 9376 bytes
- Kernel with all services: 13692 bytes

Interrupt latencies

- Non-partitioning VirtuosoNext:
  - IRQ to ISR: 620 to 2460 nanoseconds (50% median 700 nanoseconds)
    IRQ to Task: 16 to 35 microseconds (50% median 22 microseconds)
- VirtuosoNext with partitioning enabled:
  - IRQ to ISR: 620 to 5180 nanoseconds (50% median 700 nanoseconds)
    IRQ to Task: 23 to 49 microseconds (50% median 30 microseconds)

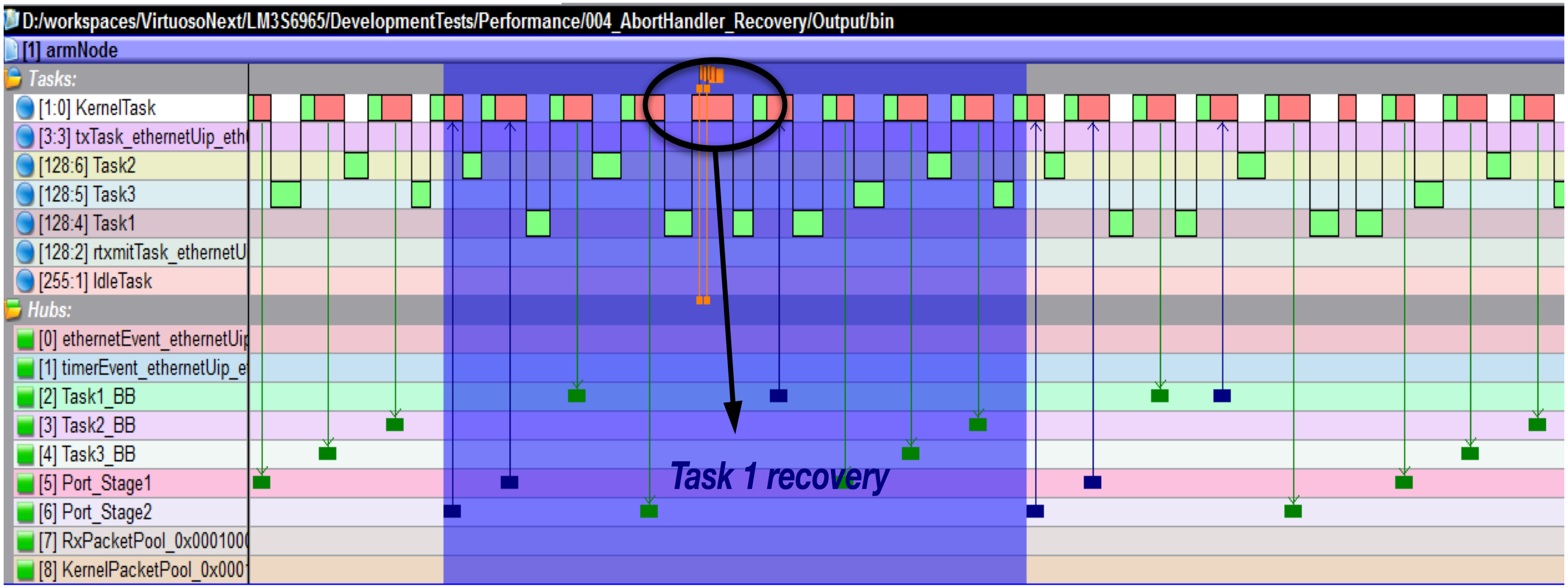
Interrupt latencies histogram

- ARM – M3 @ 50 Mhz (fully loaded)



Fault recovery

- 44 microsecs @50 MHz
  - Includes tracing overhead (50%)
  - Includes restore recovery point



More information

- www.altreonic.com

- Info.request (@) altreonic.com