The Interacting Entities concept of OpenComRTOS

Bernhard H.C. Sputh

bernhard.sputh@altreonic.com,
http://www.altreonic.com

Altreonic

From Deep Space to Deep Sea

Outline

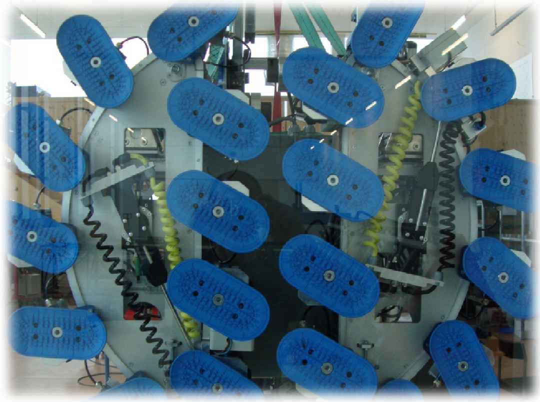
- Introduction
- Introduction to OpenComRTOS
- Performance Figures
- Demonstrations

Altreonic

From Deep Space to Deep Sea

Why Scalability is needed

- Building robots / systems out of smart sensors and actuators.
- Central control moves towards distributed control.



Scalability / Distribution

- Application Domains:
  - Multi sensor fusion,
  - Image processing,
  - radar, sonar
- Applications can utilize additional resources.
  - Additional CPU-Cores
  - Additional communication links
- Potential problems of Distributed Control:
  - Design complexity increases
  - Probability of failure increases

Introduction to OpenComRTOS

- Supported Targets
- OpenComRTOS Designer
- Interacting Entities
- Virtual Single Processor
- Open Tracer
- Open System Inspector
- Safe Virtual Machine

Supported Targets

- Host Operating Systems:
  - MS-Windows 32
  - POSIX 32 (Linux 2.6 / 3.0)
- Native Support:
  - ARM-Cortex-M3
  - PowerPC e600
  - TI C66x
  - XMOS XS1
- Dormant Ports: Xilinx Microblaze, ESA Leon3 , MLX16, NXP CoolFlux,

OpenComRTOS Designer

- OpenComRTOS Designer, offers to:
  - Use 1 – 2^{24} Nodes (CPU-Cores) in one System.
  - Support heterogeneous systems.
  - Use different communication technologies between Processing-Nodes (RS232, Ethernet, PCIe, RapidIO, etc.)
- Paradigms:
  - Interacting Entities
  - Virtual Single Processor (VSP) Programming Model

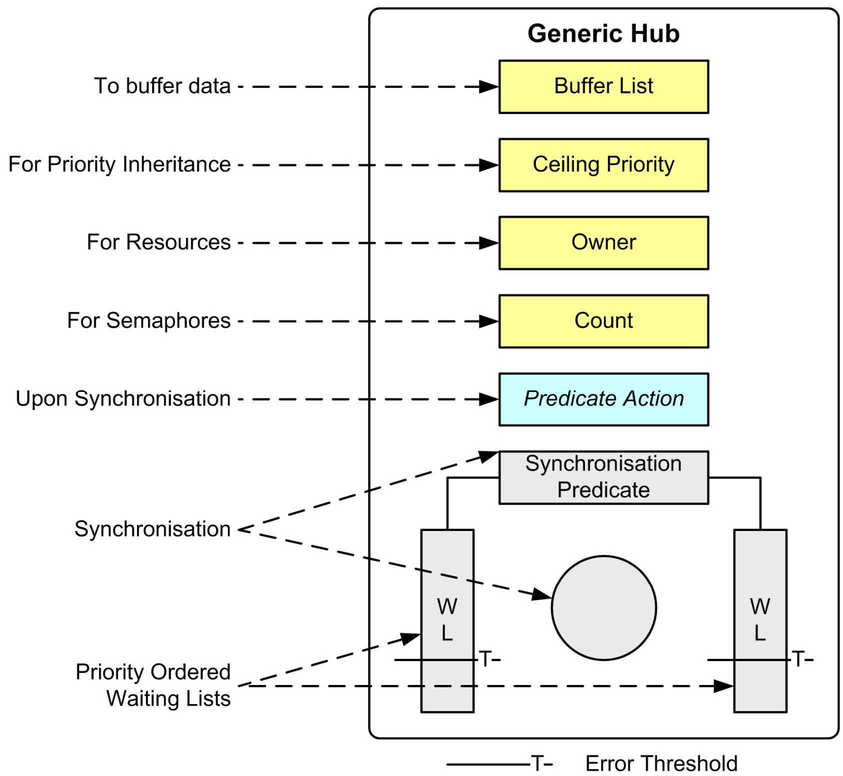
Interacting Entities

- Entities:
  - Active Entities (Tasks)
  - Passive Entities (Hubs)
- Interactions:
  - Service Requests from a Task to a Hub;
  - Represented by *packet exchanges*, not function calls!
  - Have the following interaction semantics:
    - _W: waiting / blocking
    - _NW: non waiting
    - _WT: waiting with timeout
    - _A: asynchronous

Available Passive Entities (Hubs)

- Port: Data exchange between Tasks
- Event: Boolean signal
- Semaphore: Counting Event
- Resource: Mutual Exclusion (Mutex / Lock)
  - Provides distributed Priority Inheritance.
- FIFO: Buffered data exchange between Tasks
- Memory Pool: Dynamic allocation of memory-blocks.

Generic Hub Model



Virtual Single Processor (1)

Separates two areas of concern:

- Hardware Configuration (Topology View)
- Application Configuration (Application View)

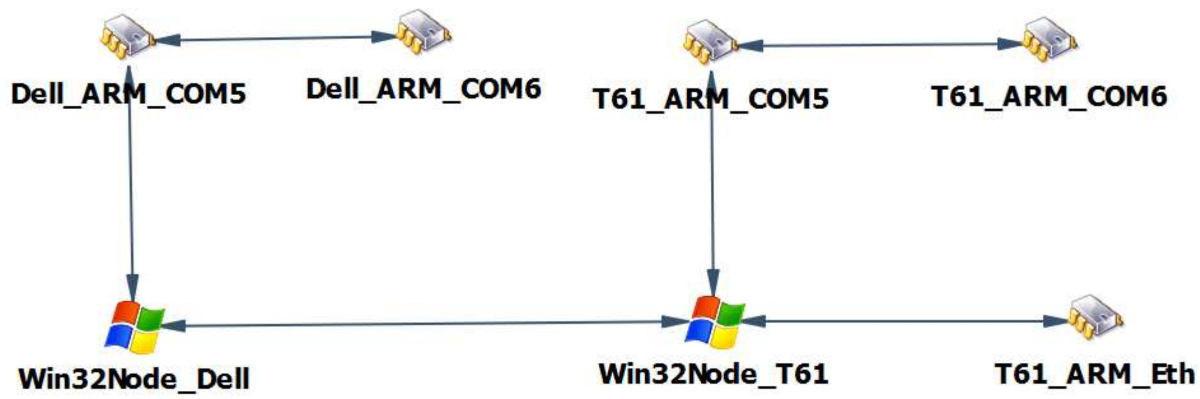
Benefits:

- Transparent parallel programming
- System wide priority management, including distributed Priority Inheritance.

Virtual Single Processor (2)

Topology View consists of:

- Nodes (CPU-Cores)
- Links: BHCS1
  - Prioritized packet communication between Nodes)

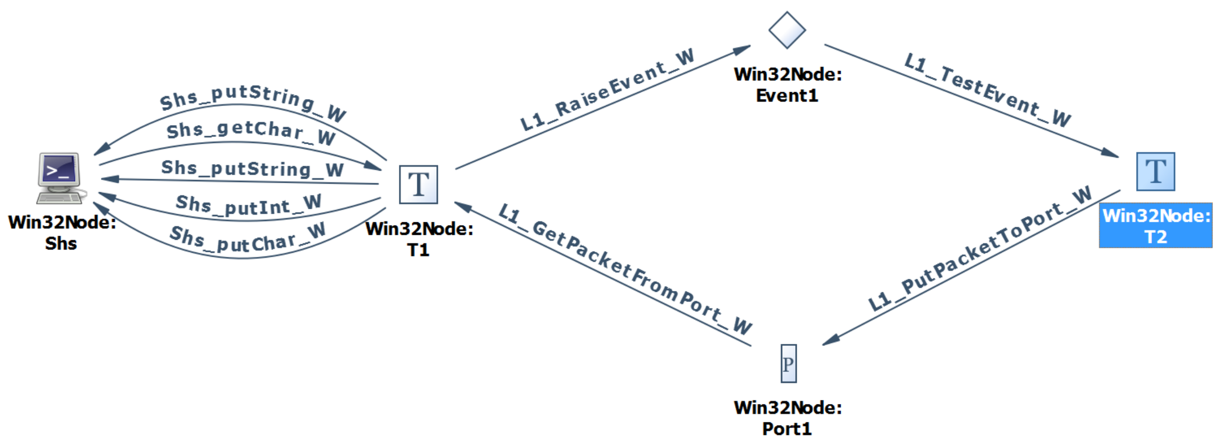


BHCS1 Chnange the image to show more targets., more complex please
Bernhard.Sputh, 1/10/2012

Virtual Single Processor (3)

Application View, consists of the following entities:

- Tasks
- Hubs
- Interactions, OpenComRTOS routes them to their destination Entity.



Virtual Single Processor (4)

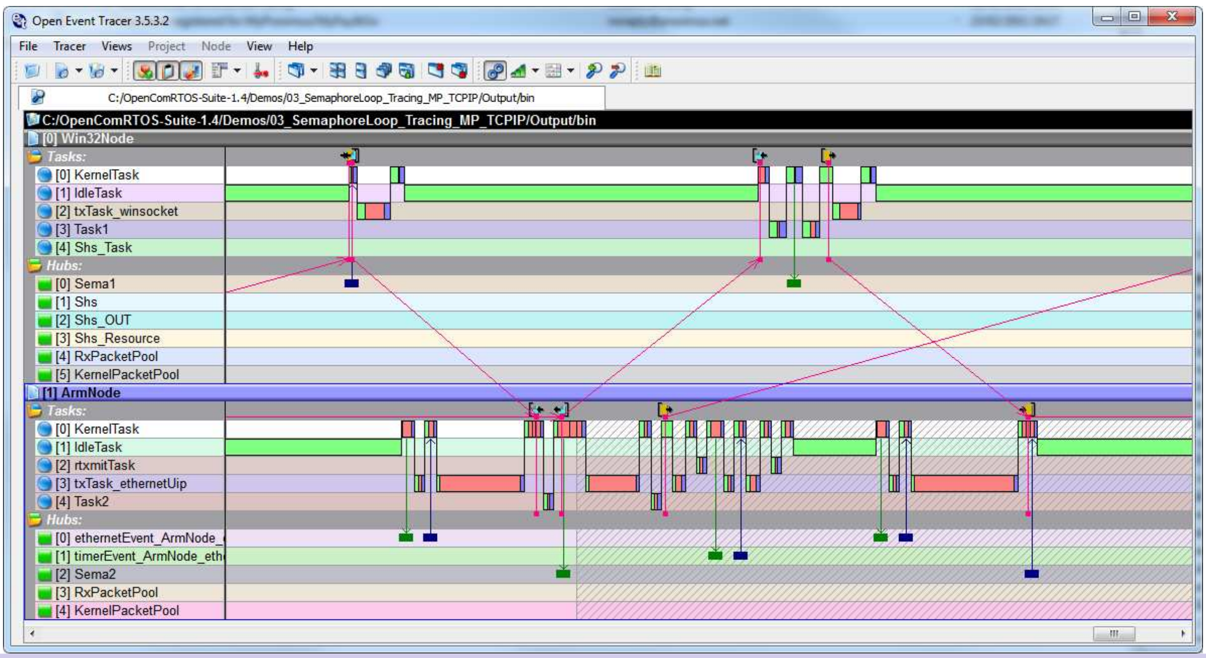
Topology Diagram Entities are represented by meta models (XML-based), which contain the information about the following:

- CPU-Core(s) (type, interconnect, compiler, …)
- Devices and their Device Drivers
- Link-Ports
- File Templates for Node Entry Point (main()).
- Hierarchy information (SoC, board, rack, cluster)

This makes it easy to deal with complex SoCs such as the TMS320C6678 (8 core DSP) or the MPC8640D (Dual core network accelerator).

Open Tracer

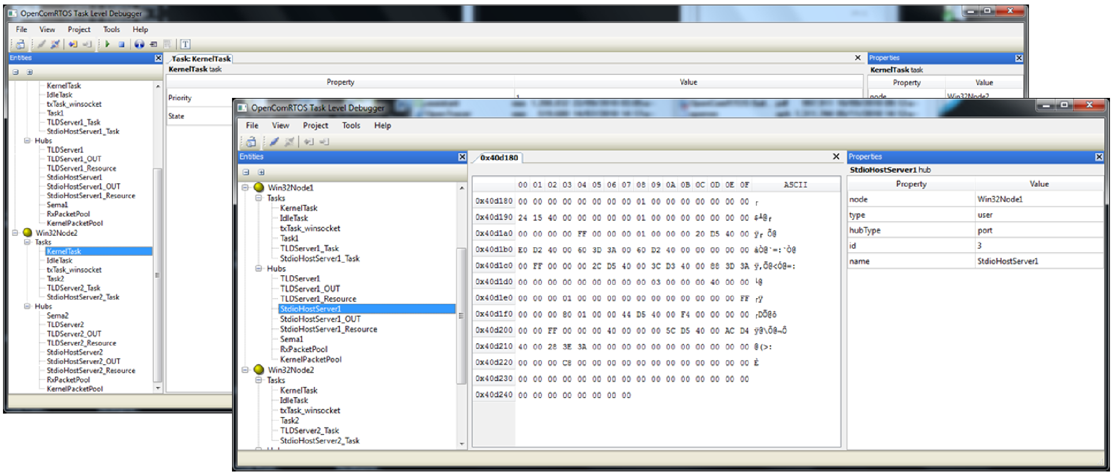
Visualizes: Context Switches, Hub Interactions, Packet exchanges between Nodes.



Open System Inspector

Allows, to inspect and modify the state of the system during runtime:

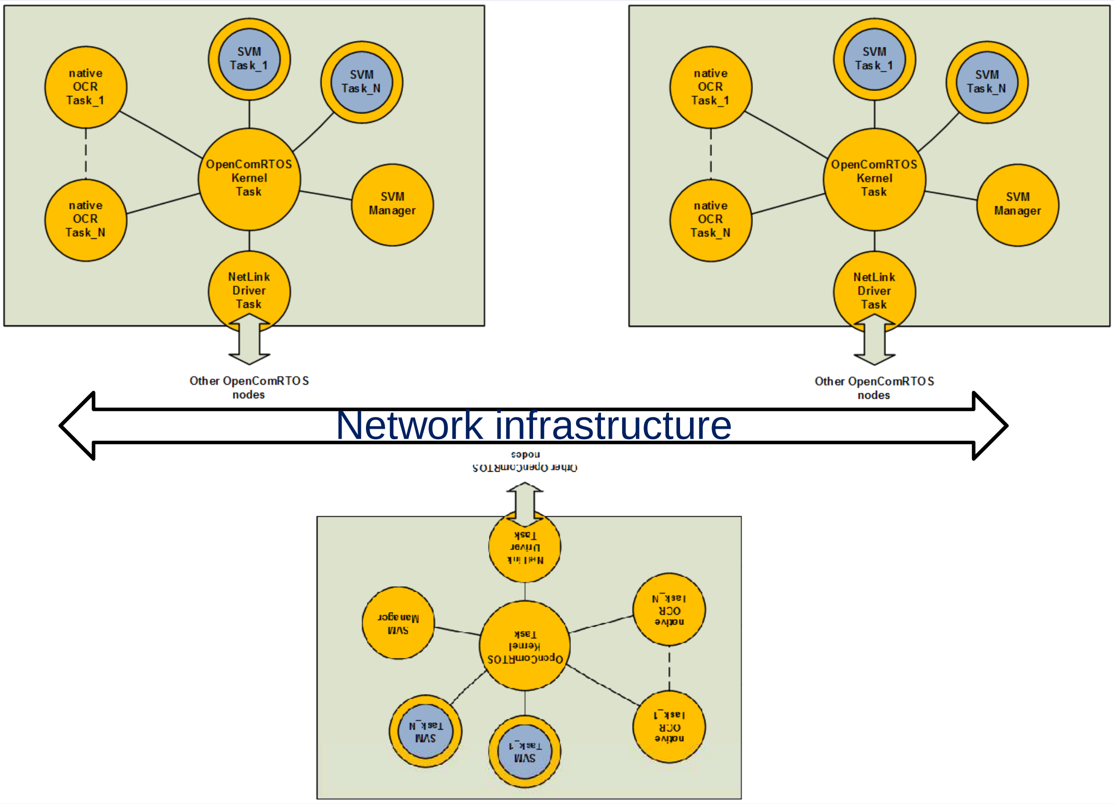
- Monitoring of the Hub state
- Peek and Poke of memory regions
- Starting and Stopping of Tasks.



Safe Virtual Machine

- Goals:
  - CPU independent programming
  - Low memory needs (embedded!)
  - Mobile, dynamic code => “embedded apps”
  - Allows to reuse legacy binary code on any processor
- Results:
  - Selected ARM Thumb1 instruction set of VM target
    - Widely used CPU
    - < 3 Kbytes of code for VM
    - Executes binary compiled code
    - Capable of native execution on ARM targets
  - VM enhanced with safety support (option):
    - Memory violations
    - Stack violations
    - Numerical exceptions

SVM System Composition



Future Extensions

- DSP specific support:
  - Distributed data management
- Platform metamodels for complex targets such as the Rack on a SoC.
- Asynchronous Services
- Automatic triplication and voting for Fault Tolerance
- …

Performance Figures

- Code-size Figures
- Task switching Figures
- Interrupt Latency

From Deep Space to Deep Sea

OCR Code-size Figures

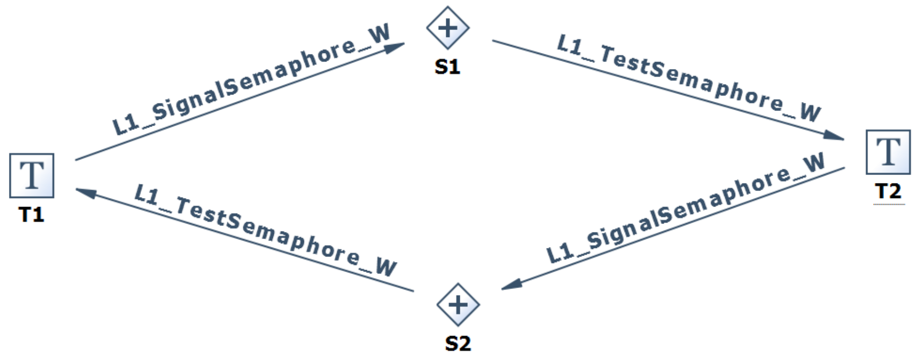
BHCS2

CPU Type	Codesize
ARM-Cortex-M3	2.5 – 4.0kB
XMOS-XS1	5.0 – 7.5kB
PowerPC e600	7.1 – 9.8kB
TI-C66x (DSP)	5.1 – 7.7kB

Code-size depends on the application, the system automatically removes unused services.

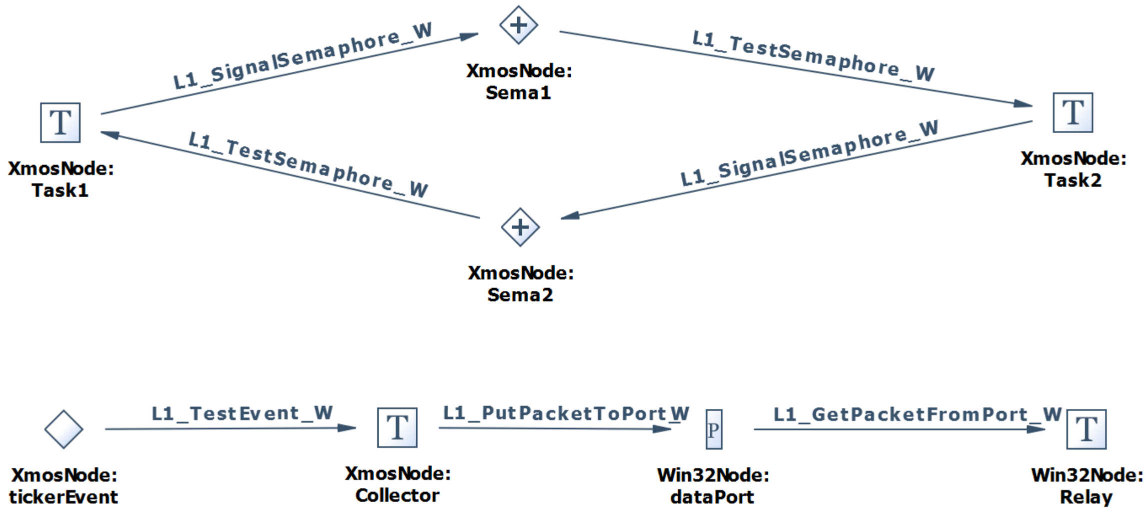
BHCS2 Add SCC figures
Bernhard.Sputh, 1/10/2012

Task Switching Figures



	Memory	Loop Time
ARM-Cortex-M3	internal	2360 cycles
XMOS-XS1	internal	2130 cycles
PowerPC e600	Simulator (psim)	1638 cycles
TI C66x (DSP)	L2-SRAM	4470 cycles

Interrupt Latency Measurement



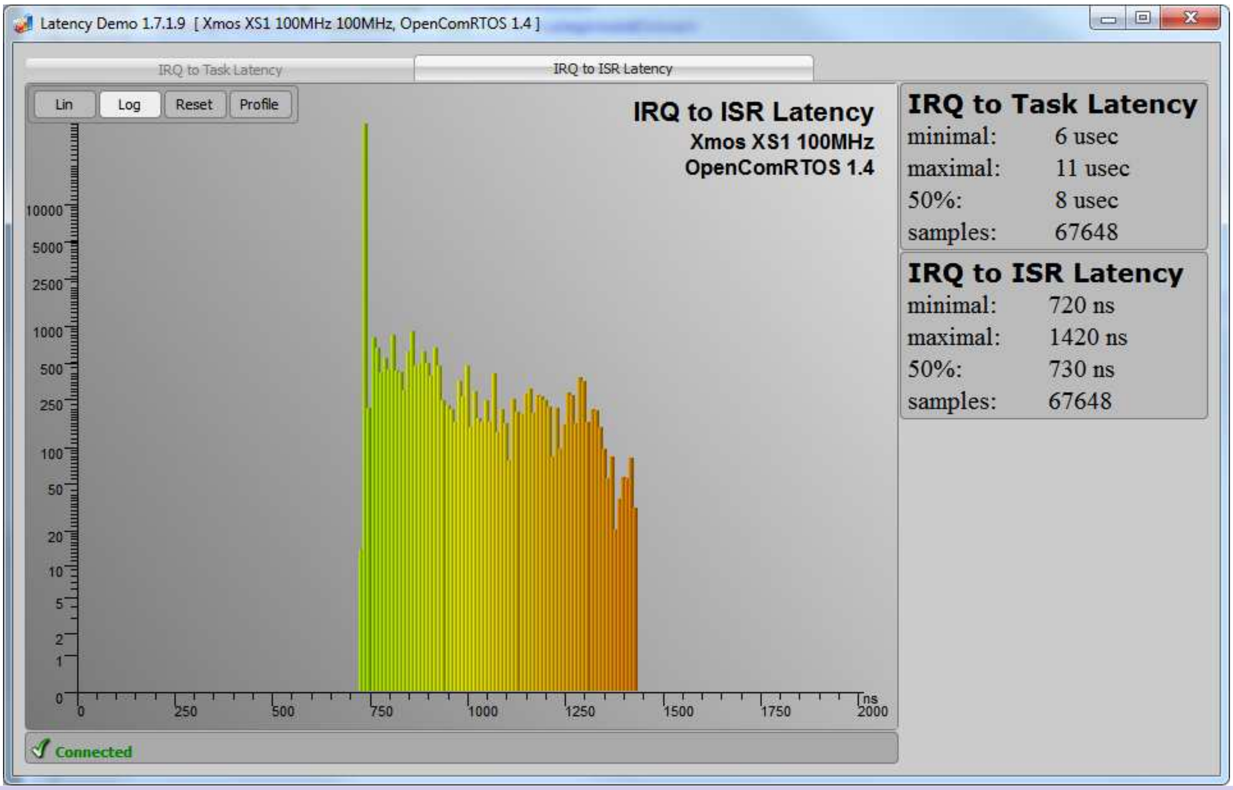
- IRQ 2 ISR: The time that elapsed between the IRQ and the first useful instruction of the ISR.
- IRQ 2 Task: The time that elapsed between the IRQ and the first useful instruction of a Task triggered by the ISR.

Interrupt Latency Figures

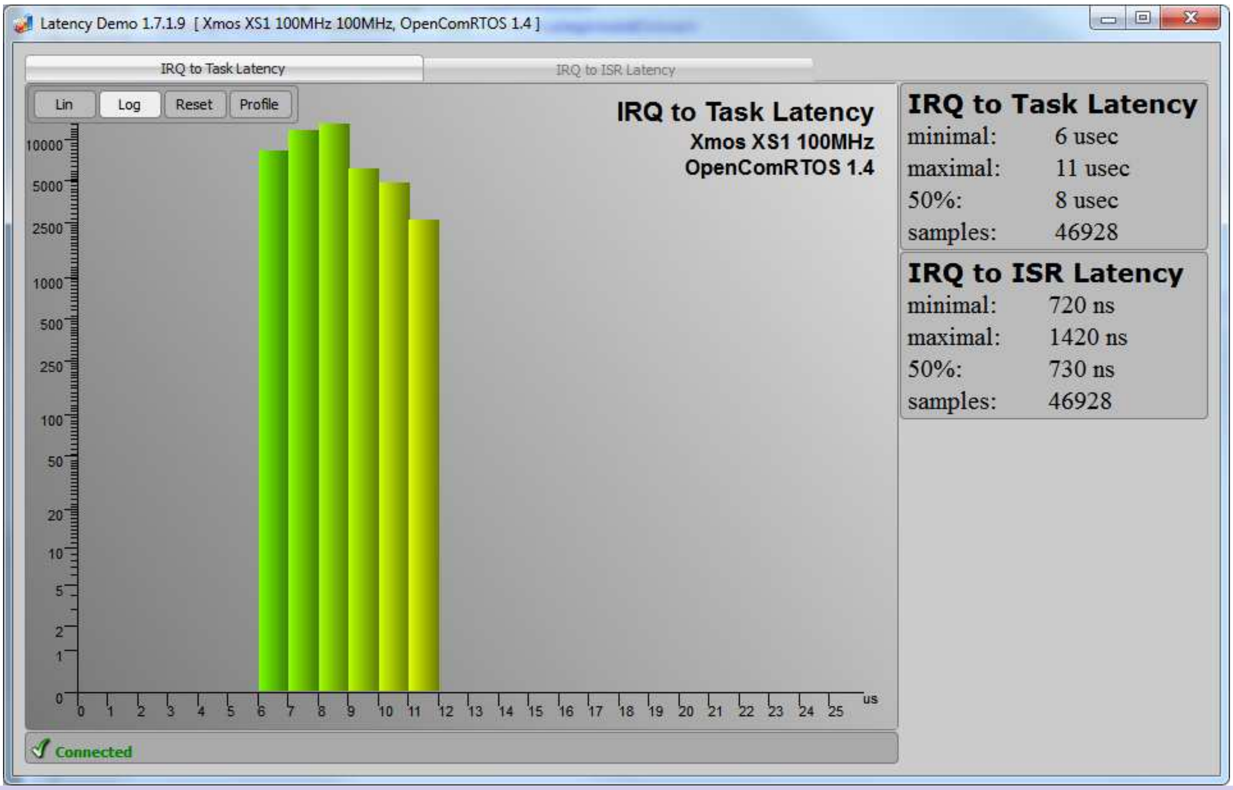
	Memory	IRQ 2 ISR	IRQ 2 Task
ARM-Cortex-M3	internal	15 – 81; (50%: 20)	600 – 1200; (50%: 800)
XMOS-XS1	internal	73 – 142; (50%: 88)	600 – 1100; (50%: 700)
PowerPC e600	Simulator	70	896
TI C66x (DSP)	L2-SRAM	136	1367

- 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.

IRQ 2 ISR on XMOS 100MHz



IRQ 2 Task on XMOS 100MHz



Demonstrations

- Open Tracer
- Protecting a Shared Resource
- Open System Inspector
- Safe Virtual Machine for C
- Interrupt Latency
- eWheel Controller Simulation

Single Node Semaphore Loop

Goal: Implementing a Semaphore Loop:

1. Create a Topology with one Win32 Node;
2. Create two Tasks;
3. Create two Semaphore Hubs;
4. Establish the Interactions between Tasks and Hubs;
5. Compile the project;
6. Execute the project.

Multi Node Semaphore Loop

Goal: Execute the Semaphore Loop distributed over two Nodes:

1. Extend the Topology by an addition Node:
2. Add an ARM Node
3. Add a connection between the ARM and Win32 nodes
4. Map one Task and one Hub onto the new ARM Node
5. Compile Project
6. Flash ARM node and Execute

Properties of the ARM Node

- Based on Luminary Micro LM3S6965.
- ARM-Cortex-M3 @ 50MHz
- 64kB RAM
- 256kB Flash
- Communicating either via:
  - RS232 @ 921600bps
  - 100Mbps Ethernet (TCP-IP)

Open Tracer

Goal: Obtain a trace from the Semaphore Loop running on the ARM and Windows:

- Add a Stdio-Host-Server to the Win32 Node.
- Write the contents of the ARM Node trace buffer onto the disk of the Win32 Node.
- Write the contents of the Win32 Node trace buffer onto the disk of the Win32 Node.
- Display the Trace using OpenTracer.

Open System Inspector

- Goal: Investigate and influence the State of the System during runtime:
  - Starting from the `Distributed Semaphore Loop' example
  - Add two OSI-Server components, one for each Node.
  - Add a OSI-Relay component to the Win32-Node.
  - Build and run
  - Start the Open System Inspector (OSI) and load the project.
  - Investigate the state of the system and influence it.

Protecting a Shared Resource

Goal: share one Screen between an ARM Node and a Windows Node:

- Insert a Resource, which provides mutual exclusive access to the StdioHostServer.
- Claim the Resource using L1_LockResource_W() before accessing the StdioHostServer.
- Release the Resource by calling L1_UnlockResource_W()

Safe Virtual Machine for C

Goal: Make Tasks loadable during runtime, and have a standard binary format for them (ARM Thumb-1)

- Starting from the the `Single Node Semaphore Loop' example
- Add an SVM node to the Topology Diagram
- Add an SVM-Component to the Application Diagram and map it to the Win32-Node, this is the VM.
- Map one of the tasks to the Node called `svm'. Thus now it will be compiled into an ARM-Thumb1 binary
- Modify a native task to load the binary image (Taskname.bin), and then start the VM.

Interrupt Latency

This demo measures two separate latencies using the Timer IRQ:

- IRQ to ISR --- How long does it take after an IRQ occurred until the first useful statement in the ISR gets executed.
- IRQ to Task --- How long does it take after an IRQ occurred until the first useful statements in the Task handling this IRQ gets executed.

eWheel Controller Simulation

This demonstration simulates a Segway type wheel, and consists of the following parts:

- eWheel Visualisation
- eWheel Controller
- Physical Model

Conclusions

- OpenComRTOS Designer allows you to master the complexity of distributed heterogeneous systems.
- OpenComRTOS has a formal foundation.
- OpenComRTOS services have been modeled and checked.
- OpenComRTOS has a small memory foot-print.
- OpenComRTOS has a high performance.
- Trace information from embedded targets can be obtained without using expensive instrumentation.
- Open System Inspector allows to inspect a running system.

Questions?

Thank You for your attention

“If it doesn't work, it must be art.
If it does, it was real engineering”