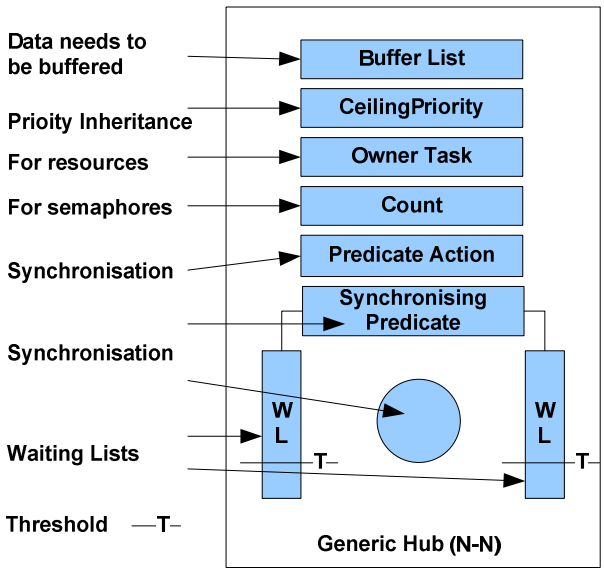
OpenComRTOS:
a Scalable and Open Network-Centric RTOS for Embedded Applications

Developed using formal modeling, the perfect RTOS for deeply embedded and distributed systems

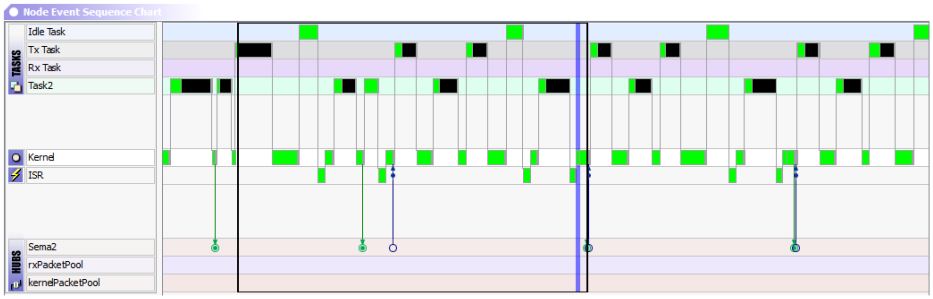
OpenComRTOS breaks new grounds in the field of Real-Time Operating Systems. From the start it was developed as a scalable communication layer to support **multi-processor** systems but runs equally well on a single processor. It supports small microcontrollers and **multi-core** chips with little memory but runs as well on **widely distributed systems.** Furthermore, it was from the ground up developed using **formal modeling**.

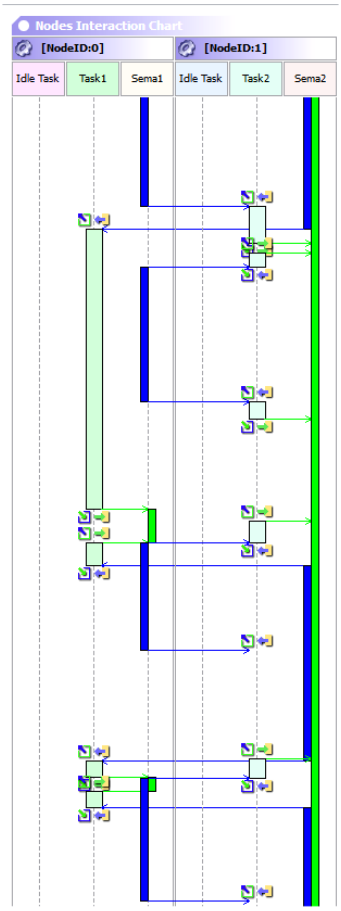


While the first approach was inspired by a **Virtual Single Processor programming** model, the formal approach was instrumental not only in achieving a trustworthy component, but also in achieving unparalleled performance with a very clean and portable architecture. An additional benefit of the unique architectural approach is that the RTOS kernel can be multiplied on the same processing node, e.g. to provide monitoring and supervision functions for safety critical applications.

OpenComRTOS provides similar kernel services as most RTOS, such as starting and stopping tasks, priority based preemptive scheduling with support for priority inheritance, Events, Semaphores, FIFOs, Ports, Hubs, Resources and Memory Pools. Generic Packet allocation/deallocation and Tasks sending/receiving such Packets using intermediate Ports for synchronisation and communication is the basis for small systems. Entirely written in ANSI-C (MISRA checked), except for the context switch, OpenComRTOS can be stripped down to about 1 KB in a single processor code size optimised implementation and 2 KB in a multi-processor implementation. The data memory requirements can be as low as 18 Bytes + 64 Bytes per task, depending on the target processor. All services can be called in blocking, non-blocking, blocking with time-out and asynchronous mode (when appropriate for the service). The kernel itself as well as the drivers are also tasks, increasing the modularity and reducing the critical sections. From the RTOS point of view the kernel essentially shuffles Packets around, while for the application the Hubs play the dominant role. Packets are sent to a Hub where they synchronise with requests from other tasks (or vice versa). If no request is available, the Packets are put in a priority sorted waiting queue. By design, such buffers cannot overflow. Another interesting feature of the Hub is that it allows the user to create his own application specific services independently of the RTOS.

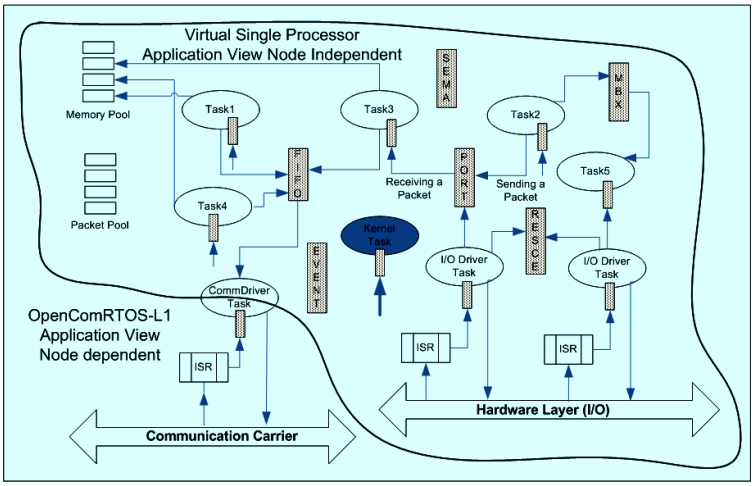
Simulation is very important, therefore the initial kernel was developed on top of Microsoft Windows. While this simulator provides for logically correct operations, it allows integrating existing host operating systems or existing RTOS with the nodes running OpenComRTOS. A simple serial connection can be sufficient to establish communication. A port to Linux is available as well (in conjunction with OpenVE. Tracer supports to analyze task scheduling and inter-node interaction.





info.request@Altreonic.com
www.Altreonic.com

Altreonic NV
Gemeentestraat 61A b1
B3210 Linden, Belgium
Tel. +32 16 202059



Available OpenComRTOS -services:

```
L1_Start/Stop/Suspend/ResumeTask
L1_SetPriority
L1_SendTo/ReceiveFromHub
L1_Raise/TestForEvent
L1_Signal/TestSemaphore
L1_Send/ReceivePacket
L1_Send/ReceiveDataPacket
L1_Enqueue/DequeueFifo
L1_Lock/UnlockResource
L1_Allocate/DeallocatePacket
L1_Get/ReleaseMemoryBlock
L1_MoveData
L1_SetEventTimerList
…
_(N)W(T)_Async: non-blocking, blocking, block-
ing with timeout, asynchronous.
```

Key size figures:

Code size on Melexis 16bit MLX16X8: 996 Bytes
(optimised implementation)
Static data size: 18 Bytes (minimum)*
Dynamic datasize: typically < 64 Bytes/Task

Other code size figures:

Minimum RTOS with Port
SP small: 996 Bytes
MP full: 3150 Bytes
L1 with hub, Port, Event, Semaphore, Resource, Fifo:
SP small: 2104 Bytes
MP full: 4532 Bytes

Performance data:

Measured on a 27 MHz (6.5 Mips) MLX16X8:

Send-Receive loop between two tasks and
using two ports: 93 us
Interrupt latency in ISR: 4 us.
Interrupt latency to task: 52 us.

*: total data requirements depend on application.

Other ports: Atmel AVR, Xilinx MicroBlaze,
LEON32, ARM, ...

OpenComRTOS services have been designed as the basic functions which are needed in embedded applications. While already rich in semantic behaviour, more elaborate and specialised services can be added using the generic Hub. The architecture allows supporting other RTOS API as well. OpenComRTOS also supports heterogenuous target systems allowing to mix 8bit, 16bit and 32bit processors or even host nodes running a tradional OS. To reduce code and memory requirements, the code is statically linked with most datastructures being generated at compile time. The developer specifies his topology and application graphically or edits directly the configuration.

One of the first customers of OpenComRTOS is Melexis, a leading supplier of semiconductor chips for automotive and consumer markets. The latest range of products, called the MelexCM, features a dual-core CPU with up to 32 KBytes of on-chip program flash memory and just 2KBytes of on-chip data memory, an area that most RTOS can't even reach.

The application domain for OpenComRTOS is wide. As a trustworthy component, it forms a good basis for developing applications that need safety and security support but have only scarce processing and memory resources. High performance, communication intensive applications will benefit from its very low memory requirements and transparent support for multi-processor applications. A natural candidate are FPGA based systems used in high band-width DSP applications. Sensor networks is another promising application domain. Furthermore, OpenComRTOS addresses the market of embedded chips that increasingly use **multi-core CPUs** for higher performance and lower power consumption. In all these systems, zero-wait state memory is a scarce resource. The performance benefits of using OpenComRTOS come from its low latency communication as well as from its low memory requirements. At the other end of the spectrum, OpenComRTOS can be used as a thin communication layer that connects heterogeneous systems together.

OpenComRTOS is bundled with an open Visual Development Environment under a binary as well as under a unique Open License that leaves no surprises. The latter includes source code and all design documents. A kernel porting kit allows porting to new targets.

OpenComRTOS is not just another RTOS. It reinvents the very concept. For the first time it combines trustworthiness with small code size, performance and ease of use, even for distributed applications. OpenComRTOS is a concurrent programming paradigm that was designed to be used.

info.request@Altreonic.com
www.Altreonic.com