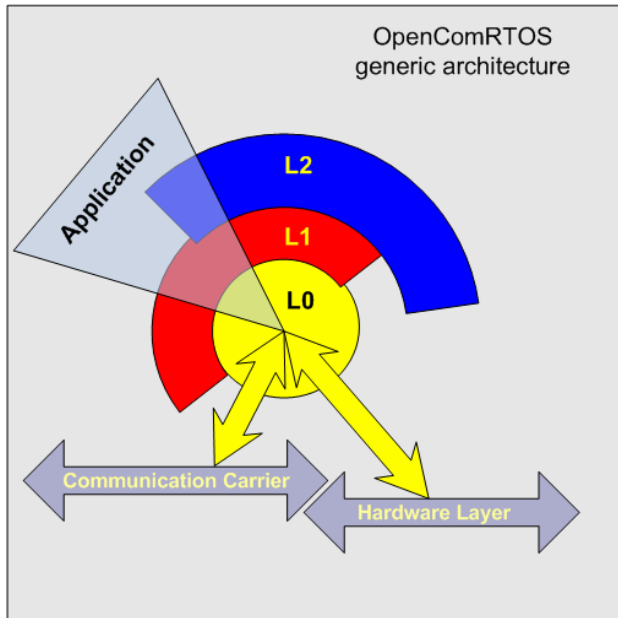


# OPEN LICENSE SOCIETY

Systematic & Unified Systems Engineering Methodologies with Trustworthy Embedded Components

## OpenComRTOS: a scalable and flexible distributed RTOS for embedded applications.

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

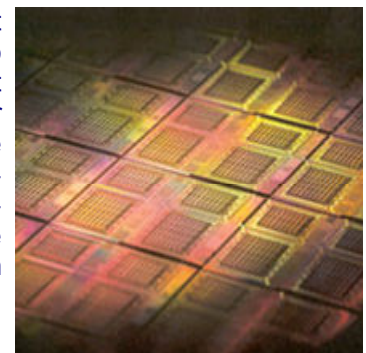
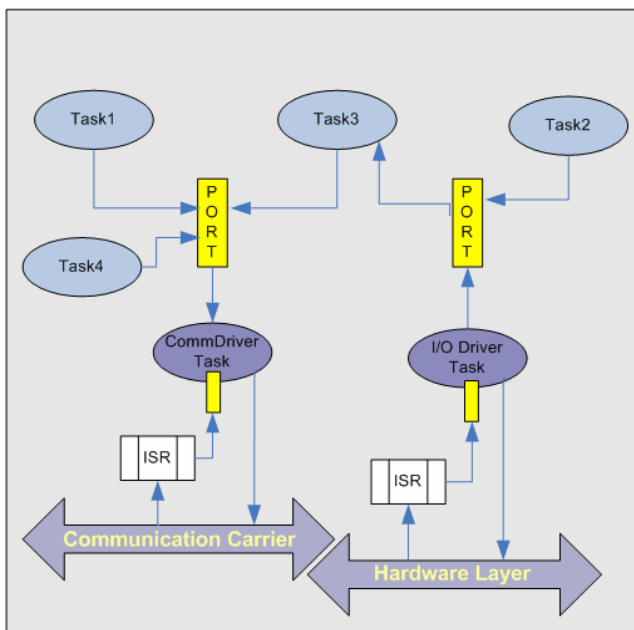


OpenComRTOS breaks new ground in the field of Real-Time Operating Systems. Firstly, from the start it was developed as a scalable communication layer to support multi-processor systems ranging from multi-core chips with little memory to widely distributed systems that are physically connected through third party communication networks. Secondly, it was from the ground up developed using formal modeling. While the first approach was inspired by the Virtual Single Processor model adopted by the Virtuoso RTOS, the second 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 architectural approach is that the RTOS kernel can even be multiplied on the same processing node, e.g. to provide monitoring and supervision functions for safety critical applications.

The first release of OpenComRTOS features the so-called L0 layer. It provides kernel services for starting and stopping tasks, priority based preemptive scheduling, Packet allocation and deallocation and sending and receiving such Packets between the Tasks using intermediate Ports for synchronisation and communication. Entirely written in ANSI-C (MISRA checked), except the context switch, it only needs about 876 bytes in a single processor implementation and 1600 bytes in a multi-processor implementation. The data memory needs can be as low as 100 bytes. All services can be called in a blocking, non-blocking, blocking with time-out and asynchronous mode (at least when appropriate for the service). The kernel itself as well as the drivers are also tasks, increasing the modularity and decreasing the critical sections. While from the RTOS point of view the

kernel essentially shuffles Packets around, for the application the Ports play the dominant role. Packets are sent to a Port where they synchronise with Packet receive request from other tasks (or vice versa). If no request is available, the Packets are put in a priority sorted waiting list. By its design, such buffers cannot overflow.

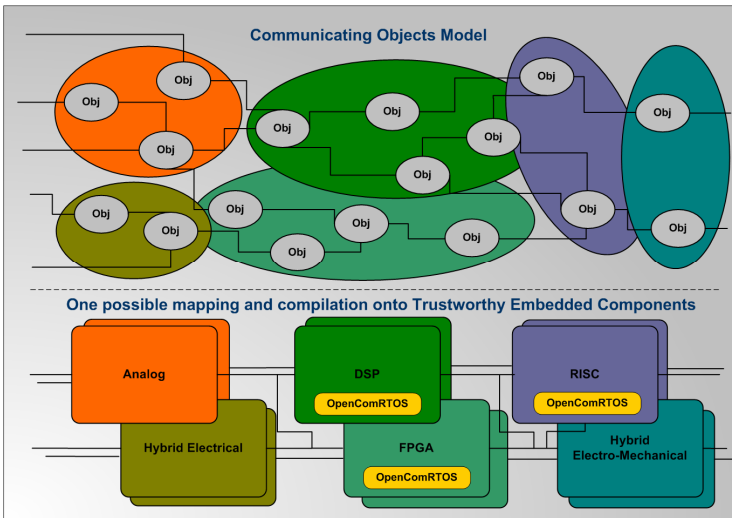
As simulation is very important, the initial kernel was developed first on top of Microsoft Windows XP, whereby internode communication is emulated using sockets. While this simulator provides for a logically correct operation, it also allows to integrate existing host operating systems or existing RTOS with the nodes running OpenComRTOS. A simple serial connection can be sufficient to establish communication.



info.request@OpenLicenseSociety.org  
[www.OpenLicenseSociety.org](http://www.OpenLicenseSociety.org)



Open License Society  
Zavelstraat 160  
B3010 Leuven, Belgium  
T. +32 16 350517



The L0 services have been designed as the basic functions that are needed in embedded (distributed or not) applications. While already rich in semantic behaviour, more elaborate and specialised services can be added using the L1 and L2 layer. The L1 layer will typically be used to provide traditional RTOS support like e.g. events, semaphores, FIFO queues, mailboxes, memory pools and resources. The first L1 layer to be developed is an emulation of the Virtuoso API but the architecture allows supporting other RTOS API as well. In the L2 layer, more elaborate often application specific services can be added. The L1 and L2 layer however are optional.

To reduce further the code and memory requirements, L0 and L1 are statically linked with most datastructures being generated at compile time. The developer specifies his topology and application objects in an xml database file. Support is currently added in Eclipse to provide a graphical configuration tool.

One of the first customers of OpenComRTOS is Melexis, a leading supplier of semiconductor chips for the 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 OpenComRTOS test program using 2 tasks continually sending and receiving 8byte packets through an intermediate ports were benchmarked at 9812 Packets/second on the 7.5 Mips MelexCM chip. In the continuous loop consisting of 2 task switches, 2 send and 2 receive services, this comes down to 102 microseconds for each loop. Total memory requirements were 1192 Bytes of flash memory and 266 Bytes of data. Timing measurements were done using the on-chip high resolution timer.

The application domain for OpenComRTOS is wide. As a trustworthy component, it forms a good basis for a developing applications that need safety and security support but have few processing and memory resources available. 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 are another interesting application domain. OpenComRTOS also addresses the market of embedded chips that increasingly use multi-core CPUs for higher performance and lower power consumptions. In all such systems, zero-wait state memory is a scarce resource and the performance benefits from the low latency communication as well as from the low memory requirements. At the other end of the spectrum, OpenComRTOS can be used as a thin communication layer that connects heterogeneous systems together.

**Available OpenComRTOS L0-services:**

- StartTask (task, priority, entrypoint), StopTask (task), SuspendTask(task), ResumeTask (task)
- Allocate/DeallocatePacket
- SendPacket (Packet, Port)
- ReceivePacket (Port)
- WaitForPacket (Port)

**Key size figures\*:**

Code size on MLX16X8: 876 Bytes

Static data size: 18 Bytes

Dynamic datasize: typically 50 Bytes/Task

**Performance data\*:**

Measured on 30 MHz (7.5 Mips) MLX16X8

Send-Receive loop between two tasks: 103 us (2 x send, 2 x receive, 2 context switches).

Total code size for this test application: 1230 Bytes and 228 Bytes of data.

\*: code size optimised version for MLX16X8

