

ENEAA OSE[®] EPSILON



OSE Epsilon RTOS Optimized for Microcontrollers

If your MCU has little memory, but needs the processing speed of a racecar, OSE™ Epsilon is the solution. OSE Epsilon has a small footprint of approximately 4 kB (measured on ARM Cortex-M3). The kernel is written completely in assembler, has low interrupt latencies and is always optimized on the respective processor. OSE Epsilon applications are API compatible with OSE programs.

OSE Epsilon is a powerful platform for the design of real-time embedded systems.

OSE's message based architecture achieves simplicity in complex and distributed systems enabling users to write efficient and code compact applications

Messages are allocated from the OSE memory pool. Memory is conserved and fragmentation is avoided. OSE Epsilon manages all of the details of buffer ownership as messages are passed from process to process without any copying, relieving the application of this responsibility.

OSE's built in error detection and handling enables cleaner code and more reliable and consistent exception handling in the systems.

Kernel Memory Footprint

The kernel memory footprint overhead is designed to be minimal, and is assembler optimized for each supported target. This allows resources to be spent on the user applications to maximize the utilization of the MCUs processing power. Enea OSE Epsilon has a modularity feature, which means that only system calls that are actually used are linked to the final target program image.

Fully Pre-Emptive

OSE Epsilon is fully pre-emptive. An interrupt can be served at anytime, even during the execution of a system call. The message-based architecture insures quick interrupt response times and general real-time behavior. Pre-emption can also occur while executing interrupt processes.

KERNEL MEMORY FOOTPRINT

ROM

Minimum Configuration (only the system calls alloc, send, receive, free_buf and delay are used)	
no debug:	2160 bytes
debug:	2760 bytes
Full Configuration (all system calls)	
no debug:	4040 bytes
debug:	4727 bytes

RAM

Normal Mode	
kernel data:	376 bytes
per process, PCB:	68 bytes
minimum stack per process:	160 bytes

Single Chip Mode

kernel data:	220 bytes
per process, PCB:	52 bytes
minimum stack per process:	160 bytes

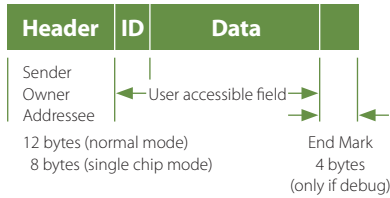
You have to add the pool size, temporary stack and interrupt stack (see file osarm.com) to find out the total data memory needs.

OSE EPSILON FOR ARM KERNEL PERFORMANCE

Kernel Configuration: No Debug	ARM7TDMI 64 MHz one wait state	ARM966 250 MHz no wait state
Complete Message Transaction: alloc - send - dispatch receive - free	13.0 us	0.67 us
alloc	3.1 us	0.16 us
send (no swap)	3.3 us	0.16 us
receive (no swap)	3.9 us	0.18 us
free_buf	2.1 us	0.10 us
Task Switch	3.5 us	0.12 us
Interrupt Latency	3.0 us	0.15 us

ENEAA OSE[®] EPSILON

OSE Epsilon for ARM Message Structure



Error Handling

OSE supports a sophisticated framework for creating error handlers at the system level. An error handler is automatically activated by error events during run-time. The error handler is either called from the application process or the OSE real-time kernel for ARM itself. The OSE for ARM does not simply return an error code to the user application if something goes wrong, which is the conventional solution. This is a very important safety issue.

CPU Transparency

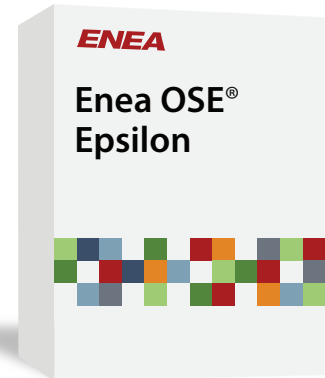
OSE message-passing is fully transparent between different CPU's. A message is sent in the same manner regardless

of whether the receiving process is running in the same or a remote CPU. OSE does not require a master CPU in a distrib-uted system. Each CPU is the same and has the same capabilities.

There is no difference in the way a globally visible or a local private process is created. A local process can become globally visible to the whole system without re-creation. It is perfectly straight-forward to design applications with OSE where the processes across multiple nodes can be viewed logically as a single system image. This unique capability of OSE enables complex problems and hardware configurations to be vastly simplified in the software model created for the platform.

OSE Epsilon Tools for Rapid Development

These OSE tools allow the user to update and configure their programs, making it more user friendly, more reliable, and more efficient. OSE builds in support for powerful application-level debugging by supporting the visualization of the events based on message-passing errors within the system.

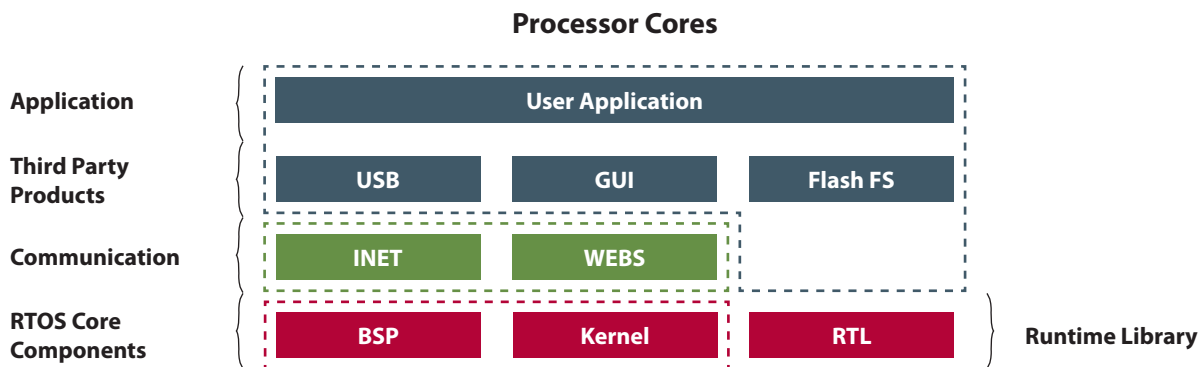


Message-level debugging is a powerful way of analyzing OSE systems where breakpoints are set on system events instead of source lines. As events in the system occur, they can be monitored, traced, and you have the ability to break the execution of your system. Application level debugging is often far more effective than simple source level instruction stepping for debugging the system design as a whole.

OSE Illuminator

The OSE Illuminator is a host-based suite of software tools for debugging

Runtime



Tools



ENEAA OSE® EPSILON



3

and analyzing OSE-based applications. OSE Illuminator connects to either an OSE real-time Kernel or Soft Kernel to debug OSE-based applications.

OSE Epsilon Soft Kernel Simulation Environment

OSE Soft Kernel simulates the system on the host. For development prior to hardware availability, the OSE Soft Kernel enables engineers to start developing and testing OSE code within a standard PC environment.

PRODUCT FEATURES

Available

- Kernel
- Board support packages
- Internet Protocols (TCP/IP)
- OSE Epsilon for ARM Web Server
- Internet Utilities
- Embedded Flash File System
- Graphics Support
- Link Handler: Enables transparent communication in distributed multi CPU systems
- Illuminator
- Soft Kernel

OSE EPSILON CPU SUPPORT

- ARM Cortex-M3
- ARM 7 & 9
- Infineon C166
- Infineon Tricore
- Freescale Coldfire
- Renesas SH2
- Mitsubishi M16C/M32C
- Atmel AVR
- Freescale 68HC11, 68HC12
- NEC V850

ENEAA