Smart Software Solutions
for Embedded Success

RTKernel-RISC Real-Time Scheduler for RISC Processors

Features

- Compact - 16k code, 6k data
- High Performance
  (See reverse for details)
- Multi-threaded
- Supports Coldfire, PowerPC, MPC5200, XScale, ARM
- Compatible with RTOS-32 from On-Time Software

Applications

- Embedded and Realtime commercial, military and consumer applications

RTKernel-RISC is a powerful real-time multitasking scheduler for embedded systems designed specifically for RISC platforms. RTKernel RISC is compact (about 16k code, 6k data), fast, and offers excellent real-time response times. All RTKernel-RISC threads run within a single program (single process, multi-threaded).

RTKERNEL-RISC THREADS

An RTKernel-RISC thread is implemented as a 'C' function. A program can dynamically create threads using the appropriate kernel API calls. These threads are subsequently executed in parallel with the main thread and any other threads.

Each thread has its own stack, a priority between I and 64, and a task state. Several threads can be started executing the same code; however, each is allocated its own stack and thus its own local variables. All threads have access to the program's global data. Thus, shared code and shared data are inherently implemented by RTKernel-RISC.

INTER-TASK COMMUNICATION

RTKernel-RISC offers three different mechanisms for inter-task communication:

- **Semaphores** allow the exchange of signals for activating and suspending threads. RTKernel-RISC supports counting, binary, event, resource, and mutex semaphores. Resource and mutex semaphores implement priority inheritance.
- **Mailboxes** (also known as queues or FIFO buffers) allow threads to exchange messages asynchronously. The maximum number and size of messages can be configured for each mailbox. High priority messages can be sent to a mailbox ahead of all others. Mailboxes can also be used to send data from hardware interrupt handlers and threads.
- **Message-passing** is used for a synchronous exchange of messages. No data is buffered, data is sent directly from one thread to another.

THE SCHEDULER

RTKernel-RISC's scheduler is event-driven. It was developed specifically for real-time requirements and adheres to the following rules:

- **Priority Scheduling**
  Of all threads in state **Ready**, the thread with the highest priority runs.
- **Round-Robin**
  If the kernel must choose from several Ready threads with the same priority, the thread that hasn't run for the longest time is activated.
- **Priority Ordered Queues**
  If several threads are waiting for an event, they are activated upon the occurrence of that event in sequence of their priorities.
- **Deterministic Scheduling**
  With the exception of time-slice task switches, a task switch is only performed if rule I would be violated.

The application can dynamically change thread priorities and it can turn preemptions and time-slicing on and off at run-time.

EBSnet, Inc. • 9 Goldsmith Street • Littleton, MA 01460 • Voice **978.486.4000** • Fax **978.486.4544** • **800.428.9340** (U.S. Only)

INTERRUPTS

The application's interrupt handlers can suspend or activate threads. Interrupt handlers can be programmed completely in 'C' within the application They can freely exchange signals or data with threads using semaphores or mailboxes. Semaphore or mailbox operations may then initiate a task switch, if required. Interrupts from any hardware can be processed.

DEBUGGING

RTKernel-RISC is delivered in two versions. The Standard Version is optimized for minimum size and best performance, while the Debug Version contains additional code for parameter and consistency checks at run-time. The Debug Version recognizes usage errors and issues corresponding error messages. Moreover, the Debug Version offers numerous debugging aids. For example, the current source code position of a thread can be displayed, all locked resources can be listed, or the CPU time requirements can be determined for each thread and interrupt handler. An especially powerful tool is the Kernel Tracer; it can log kernel and application events in real-time for off-line analysis.

As an additional aid to debugging, RTKernel-RISC (Debug and Standard Version) offers a number of informational functions. For example, a list of all threads, semaphores, or mailboxes can be displayed, or the state of a specific thread can be queried. Furthermore, RTKernel-RISC keeps statistics of the stack usage of all threads and interrupt handlers.

RTKERNEL-32 COMPATIBILITY

With a few exceptions Rtkernel-RISC is compatible with RTKernel-32. RTKernel-32 is part of the RTOS-32 development suite for protected mode X86. RTOS-32 has a large installed base and worldwide reputation for rock solid value and reliability.

SUPPLEMENTAL MODULES

RTKernel-RISC is supplied with the following supplemental modules, always delivered in full source code:

- **FineTime** - high resolution time measurement
- **Clock** - timer interrupt management
- **Console** - Serial and Telnet (requires EBSnet's TCP-IP stack) based console drivers
- **QUICC** - Full support for ethernet and UART with the 860 QUICC
- **Board Support** - Embedded Planet, Motorolla MBX, ADS, FADS.

OPTIONAL MODULES

Tightly integrated versions of the EBSnet's IPv4/IPv6 network stack and ERTFS DOS compatible file system packages are available for RTKernel-RISC.

RTKERNEL-RISC PERFORMANCE DATA

RTKernel RISC offers excellent performance. It comes with a benchmark program that may be used to measure its performance on any computer. The table to the right gives some results for four typical X86 based target computers and for a 24 MHz MPC860 processor.

Note: Times are given in µs • Note: Times may vary due to instruction caching

20MHz 386EX	33MHz 486	120MHZ Pentium	24MHz 860	RTKernel Operation
43	5	0.73	15	Round-Robin task switch
79	10	1.61	20	Semaphore task switch
37	6	1.18	16	Semaphore Signal
25	4	1.24	14	Semaphore Wait
100	13	3.13	23	Task activation (Signal, Wait)
31	12	3.95	22	Store data in a mailbox
30	10	2.77	20	Retrieve data from a mailbox
96	12	2.50	22	Task-to-task communication
107	18	4.03	38	Task-to-mailbox-to-task-communication