



RTX51 Real-time Kernel

Let's Embed the World

[Home](#)
[Contents](#)
[Search](#)
[Index](#)
[What's New](#)

RTX51

[About RTX51](#)
[RTX51 Full](#)
[RTX51 Tiny](#)
[Specifications](#)
[CAN Support](#)
[RTX51 FAQ](#)

[How to Place an Order](#)
[Why Buy Tools from Us?](#)

[Information Request](#)
[Get a Quote](#)

About the RTX51 Real-time Operating System

Keil Software provides two different real-time operating systems for the 8051.

- [RTX51 Full](#) is a full-blown real-time kernel. It supports up to 256 tasks and provides control over semaphores, task signals, message queues, and memory pools. It is designed for 8051-based applications that have numerous tasks and many resources to manage. [CAN](#) support is provided for a number of 8051-based CAN microcontrollers as well as many external CAN controllers.
- [RTX51 Tiny](#) is a small real-time kernel (around 800 bytes). It supports applications that have reduced real-time requirements. It is perfect for applications that require task switching and limited inter-task communications. It is included with the [PK51 Professional Developer's Kit](#).

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.
Report any problems to the [webmaster](#).

RTX51 Full

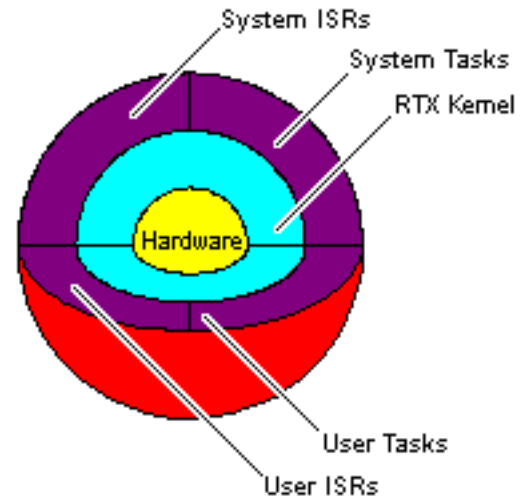
While embedded applications solve different real-world problems, many software developers are confronted with two fundamental problems:

- An operation must be executed within a relatively short time frame and must have a guaranteed response time. In other words, it must be a real-time operation.
- Several operations are time-dependent and/or logic-dependent and must execute simultaneously (multitasking).

These operations may be organized as independent computer processes that are usually referred to as tasks.

The RTX51 Real-Time Operating System combines a kernel library with the task definition syntax built into the Keil C51 Compiler to provide a simple, effective method to solve these problems.

While it is quite possible for an application programmer to implement the features found in a commercial real-time kernel, RTX51 saves time and effort and provides a tested, proven real-time interface.



RTX51 Product Information

- [Advantages of using a Real-Time Kernel](#)
- [System Calls](#)
- [Task Communication and Synchronization](#)
- [Task Switching](#)
- [Event Handling](#)
- [Interrupt System](#)
- [Time Management](#)
- [Memory Management](#)
- [Exception Handling](#)
- [CAN Interface](#)
- [Specifications](#)

Required Software Tools

RTX51 requires the Keil [C51 C Compiler](#).

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.
Report any problems to the [webmaster](#).

Advantages of using a Real-Time Kernel

There are a number of general advantages to using a real-time kernel.

- A program may be broken down into individual tasks that are easier to understand and implement.
- The modular approach to multitasking programs promotes software reuse and allows tasks to be used in other projects.
- Since the kernel addresses the real-time and multitasking issues, more time can be devoted to creating and testing the application.

There are specific advantages to using the RTX51 real-time operating system for your 8051 applications.

- RTX51 is integrated into the Keil C51 Compiler. This makes using the real-time kernel with your program much simpler than a less well-integrated solution.
- RTX51 fully supports all of the features found in C51 such as interrupt functions and memory models.
- RTX51 is easy to configure for all members of the 8051 microcontroller family.
- RTX51 provides powerful system functions for combined event testing and variable sized message passing.
- RTX51 is flexible and requires few system resources.

Ease of Use

RTX51 is easy to use because all of its features are tightly coupled with the Keil C51 C Compiler. C51 provides an extended function declaration syntax for defining tasks. The following function declaration shows how easy it is to declare tasks in C.

```
void func (void) _task_ task_number [_priority_ priority] [using register_bank]
```

Graphical Configuration Builder

RTX51 is easily configured using a graphical configuration utility. The configuration utility runs under MS-DOS and Windows 3.1x, Windows 95, Windows 98, and Windows NT. Using the configuration utility, the configuration options of RTX51 may be modified to meet your requirements for system functionality and RAM utilization.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

System Calls

In addition to declaring tasks, RTX51 supports a number of system calls you may use. The system calls are implemented as a library of C-callable function you use to manage and control your application and its RTX tasks.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

Task Communication and Synchronization

RTX51 provides three mechanisms that enable individual tasks to synchronize with and communicate with each other.

■ **Signals** are the fastest form of task synchronization since no actual information is exchanged. A signal is only a stimulus for a task. For example, Task A may wait (using **os_wait**) for a signal from another task. When Task B sends a signal to Task A, Task A resumes execution.

■ **Messages** are actual data (fixed or variable length) that are transferred via mailboxes. Mailboxes support the buffered exchange of data between tasks. Each mailbox holds up to eight messages in a FIFO. The first message stored in the mailbox is always the first message retrieved from the mailbox.

When a task waits for a message (using **os_wait**), the task is queued with other tasks waiting for messages from that mailbox. Several different tasks may all be waiting for a message from the same mailbox. When a message arrives in the mailbox, the task with the highest priority receives the message.

RTX51 supports two types of mailboxes: **Variable Size** mailboxes accept messages of variable size (no two messages sent to a certain mailbox must have equal size) while **Fixed Size** mailboxes use a fixed message format of pointer size (this is typically used to exchange buffer addresses).

■ **Semaphores** are simple mechanisms typically used to share common resources. You may use semaphores to share limited resources like serial ports or communication buffers. When a task needs to use a controlled resource (like the serial port), it requests the semaphore for the serial port. If no other tasks is using the serial port, the semaphore is granted. When the task is finished using the serial port, it must give up the semaphore. If a task requests a semaphore for a resource that is already in use, the task is suspended until the semaphore is available. RTX51 supports only binary semaphores.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

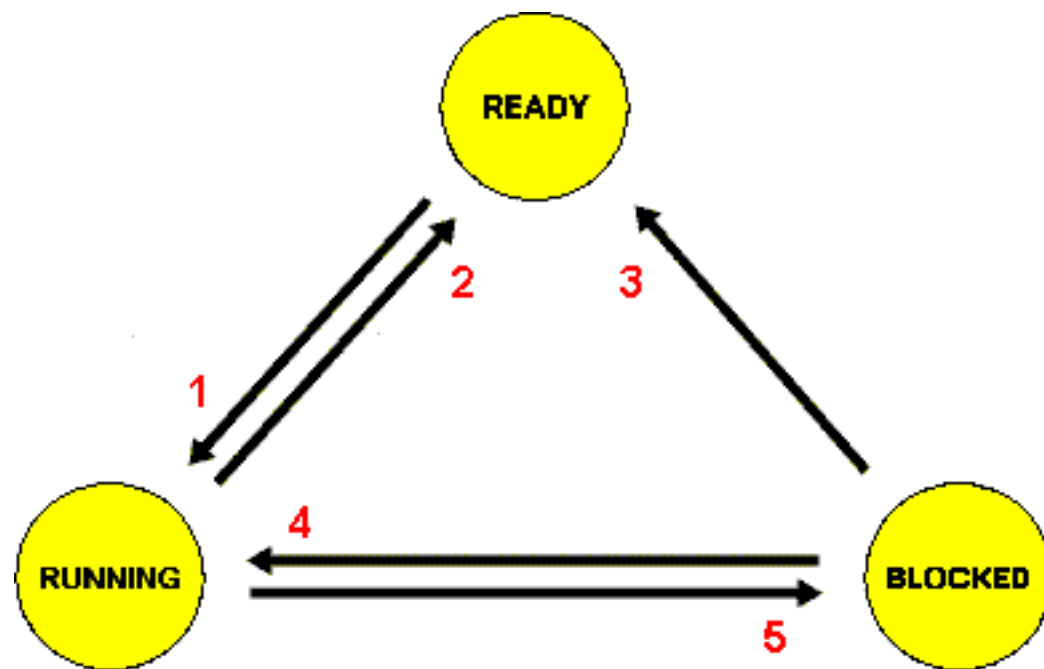
Task Switching

RTX51 recognizes four task states: **Ready**, **Running**, **Blocked**, and **Sleeping**. A task may be in one and only one of these states at any given time.

Task State	Description
Ready	All tasks that are not blocked or running and that are ready to run are ready .
Running (Active)	Only one task is running . This is the task that is currently being executed by the CPU.
Blocked (Waiting)	A blocked task is a task that is waiting for an event like a time-out, message, semaphore, or a signal.
Sleeping	Any task which has been declared but which has not started is a sleeping task. Tasks that have run but have terminated are in the sleeping state.

The **ready**, **running**, and **blocked** states are considered to be active states since tasks of these states were started by the user program. The **sleeping** state is an inactive state since tasks of this state either have not been started or have been terminated.

The following directed graph illustrates how tasks may transition from state to state.



Transition	Description
1	A task in the ready state is selected for execution. RTX51 changes the task's state to running and resumes execution of the task. The previously running task is switched to the blocked or ready state.
2	A task in the running state is changed to ready . This typically occurs when a higher priority task is ready to run.
3	A task in the blocked state is changed to the ready state. When a task is waiting for an event (signal, time-out, message, or semaphore) it is in the blocked state. When the event occurs, the task may be switched to the ready state.

4	A task in the blocked state is changed to the running state. A task is waiting for an event is in the blocked state. When the event occurs, the task may be switched to the running state if it has a higher priority than the current running task.
5	A task in the running state is changed to the blocked state. This occurs when a task calls os_wait to wait for an event that has not yet occurred.

By default, RTX51 performs preemptive multitasking using an event-driven task switching scheme with priorities. Basically, the task with the highest priority that is **ready** gets to run.

You may enable a time-slice task switching scheme otherwise known as round-robin multitasking. In round-robin, a task executes for a predetermined amount of time. The kernel then switches to another task, of the same priority level, for the same time-slice, and so on.








RTX51 recognizes 4 user task priority levels.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

Event Handling

A task may wait for various events to occur without using any CPU time. While one task is waiting for an event, other tasks that are **ready** may run. Events that a task can wait for may be:

-  Receipt of a message,
-  A signal,
-  An interrupt,
-  A semaphore,
-  An interval,
-  A time-out,
-  Any combination of the above.

Tasks wait for these events using the **os_wait** system call. The following forms of waiting are supported by RTX51.

Method	Description
Normal	The waiting or blocked task may be blocked for an arbitrary amount of time until the corresponding event occurs.
Conditional	The waiting task is never blocked . The task can evaluate if the event occurred by using the return value from os_wait .
Normal with Time-Out	The task is blocked until the event occurs or until the specified time limit is exceeded. This is useful when waiting for infrequent events or when an error-recovery time-out is necessary.

You may combine any of the following events in the **os_wait** system call.

1. wait for a message from a mailbox
2. wait for an interrupt
3. wait for a signal
4. wait for a time-out
5. wait for end of interval
6. wait for a token from a semaphore

Combining events in the **os_wait** system call can keep program structure simple. For example, if a task needs a resource that is controlled by a semaphore and it needs to check a mailbox and it periodically must update a register (using the interval event), a single **os_wait** call may be used to check all of these events at once. If one of them has already occurred, RTX51 immediately returns control to the task along with the proper status. If one of the events has not occurred, the task is **blocked** until one does occur. Then, the task is **ready** for execution again.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

Interrupt System

RTX51 synchronizes tasks with external events using the CPU's interrupt system. Two types of interrupt processing are basically supported in this case:

1. **C51 Interrupt Functions**

Interrupts are processed by C interrupt functions. They may exchange signals and data with RTX51 tasks.

2. **Task Interrupts**

Interrupts are processed by fast or standard tasks of RTX51.

The method of interrupt processing may be selected depending on the application. The individual methods may also be combined in a single application.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

Time Management

RTX51 maintains an internal time counter that measures the relative time passed since the system was started. The physical source of the time-base is an interrupt from one of the CPU's on-chip hardware timers. The time that passes between timer-tick interrupts is called a system time slice or a system tick. The hardware timer used and the time between interrupts are both user-configurable.

The counter supports time-dependent services such as

- Pausing a task,
- Time-outs while waiting for semaphores, messages, or signals,
- Cyclic task activation,
- Time-out checks in polling loops,
- Simple task delays.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.
Report any problems to the [webmaster](#).

Dynamic Memory Management

Dynamic memory space is often necessary in a multitasking system for storing intermediate results or messages. The time required for memory allocation and de-allocation must fit within constant time limits in a real-time system. For this reason, memory management routines that work with variable sized blocks (such as the standard C functions **malloc** and **free**) and not well-suited for real-time applications.

RTX51 uses a fast, effective memory management scheme that works with memory pools of fixed-size memory blocks. You may create up to 66 different memory pools (each using a different fixed-size block). Each pool may contain up to 255 blocks.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

Exception Handling

Exception handling is one of the most important aspects of a real-time computer system. It is vital that the system is able to detect, isolate, identify, and correct abnormal conditions that occur during the execution of any task. This is especially true in systems with time-critical requirements or other aspects that make it imperative to maintain control.

RTX51 supports different approaches of error handling in a real-time application.

- In-line error handling by examining system call return status (local).
- Common error handling for all tasks (system wide).
- Separate error handling for each task (per task).
- Combinations of any or all three above mentioned methods.

RTX51 system calls validate parameters and catch most invalid values before they can crash the system.

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.

Report any problems to the [webmaster](#).

CAN Interface




RTX51 supports the [Controller Area Network](#) (CAN) controller built into the Intel 82526 and 82527; Philips 82C200, 8xC592, and 8xC598; and the Siemens 81C90, 81C91, C505C, and C515C. Using the CAN interface built-in to RTX51, you can easily implement multi-processor systems using any combination of 8051 and 166 CPUs.







[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.









Report any problems to the [webmaster](#).









RTX51 Specifications






There are two versions of RTX51: RTX51 Full (part number FR51) and RTX51 Tiny (part number TR51). Both kernels help you create applications with multiple tasks.

Multitasking		<u>FR51</u> RTX51 Full	<u>TR51</u> RTX51 Tiny
	Round Robin Multitasking	✓	✓
	Preemptive Multitasking	✓	-
	Cooperative Multitasking	✓	✓

Events		<u>FR51</u> RTX51 Full	<u>TR51</u> RTX51 Tiny
	Timeouts	✓	✓
	Intervals	✓	✓
	Signals	✓	✓
	Messages	✓	-
	Binary Semaphores	✓	-
	Memory Pools	✓	-

Parametric Specifications		<u>FR51</u> RTX51 Full	<u>TR51</u> RTX51 Tiny
	Maximum Number of Tasks	256	16
	Maximum Active Tasks	19	16
	CODE Space Required	6-8 Kbytes	900 Bytes
	DATA Space Required	40-46 Bytes	7 Bytes
	Stack (IDATA) Space Required	20-200 Bytes	3 Bytes for each task
	XDATA Space Required	650 Bytes minimum	-
	Timer Used	0, 1, or 2	0
	System Clock Divisor	1,000-40,000 cycles	1,000-65,535 cycles

	Interrupt Latency	< 50 cycles	< 20 cycles
	Context Switch Time (Fast Task) (depends on stack load)	70-100 cycles	-
	Context Switch Time (Standard Task) (depends on stack load)	180-700 cycles	100-700 cycles
	Task Priority Levels	4	-
	Semaphores	8 maximum	-
	Mailboxes	8 maximum	-
	Mailbox Size	8 entries	-
	Memory Pools	16 maximum	-

Other Features		<u>FR51</u> RTX51 Full	<u>TR51</u> RTX51 Tiny
	Code Banking Support		
	CAN Support (for the Intel 82526 and 82527; Philips 82C200 and 8xC592; Siemens 81C90, 81C91, C505C, and C515C)		-

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.
Report any problems to the [webmaster](#).



8051 Development Tools

Let's Embed the World

[Home](#)
[Contents](#)
[Search](#)
[Index](#)
[What's New](#)

8051 Tools

[About C51](#)
[Demo Tools](#)
[Evaluation Boards](#)
[C51 FAQ](#)
[How to Get Support](#)
[Latest Versions](#)
[RTX51 RTOS](#)
[C51 Updates](#)

[How to Place an Order](#)
[Why Buy Tools from Us?](#)

[Information Request](#)
[Get a Quote](#)

About

Keil Software development tools for the 8051 support every level of developer from the professional applications engineer to the student just learning about embedded software development. We provide C Compilers, Macro Assemblers, Debuggers, Real-time Kernels, and Single-board Computers.

The following table shows our Products (across the top) and the Components that are included (along the left side). You may use this information to find the development tool kit that best fits your needs.

Part Number Development Tools	A51 Assembler Kit	CA51 Compiler Kit	DK51 Developer's Kit	PK51 Prof. Developer's Kit
µVision IDE	✓	✓	✓	✓
A51 Macro Assembler	✓	✓	✓	✓
BL51 Code Banking Linker	✓	✓	✓	✓
OH51 Object-HEX Converter	✓	✓	✓	✓
C51 ANSI C Compiler	-	✓	✓	✓
OC51 Banked Object Converter	-	✓	✓	✓
dScope Simulator/Debugger	-	-	✓	✓
MON51 Target Monitor	-	-	✓	✓

[RTX51 Tiny](#)
[Real-time Kernel](#)

- - -



See Also

- [Keil C251 Development Tools](#)
- [Keil C166 Development Tools](#)
- [Keil Real-time Operating Systems](#)
- [PC-Lint Diagnostic Tool](#)
- [Training Courses](#)

[Copyright](#) © 1996-1999 [Keil Software, Inc.](#) All rights reserved.
Report any problems to the [webmaster](#).