

This chapter describes the MicroC/OS-II real-time kernel for the Nios® II embedded processor. This chapter contains the following sections:

- “Overview of the MicroC/OS-II RTOS” on page 10–1
- “Other RTOS Providers” on page 10–2
- “The Nios II Implementation of MicroC/OS-II ” on page 10–2
- “Implementing MicroC/OS-II Projects for the Nios II Processor” on page 10–6

Overview of the MicroC/OS-II RTOS

MicroC/OS-II is a popular real-time kernel produced by Micrium Inc. MicroC/OS-II is a portable, ROMable, scalable, pre-emptive, real-time, multitasking kernel. First released in 1992, MicroC/OS-II is used in hundreds of commercial applications. It is implemented on more than 40 different processor architectures in addition to the Nios II processor.

MicroC/OS-II provides the following services:

- Tasks (threads)
- Event flags
- Message passing
- Memory management
- Semaphores
- Time management

The MicroC/OS-II kernel operates on top of the hardware abstraction layer (HAL) board support package (BSP) for the Nios II processor. Because of this architecture, MicroC/OS-II development for the Nios II processor has the following advantages:

- Programs are portable to other Nios II hardware systems.
- Programs are resistant to changes in the underlying hardware.
- Programs can access all HAL services, calling the UNIX-like HAL application program interface (API).
- ISRs are easy to implement.

Further Information

- This chapter discusses the details of how to use MicroC/OS-II for the Nios II processor only. For complete reference of MicroC/OS-II features and usage, refer to *MicroC/OS-II - The Real-Time Kernel* by Jean J. Labrosse (CMP Books). You can obtain further information from Micrium (www.micrium.com).

Licensing

Altera distributes MicroC/OS-II in the Nios II Embedded Design Suite (EDS) for evaluation purposes only. If you plan to use MicroC/OS-II in a commercial product, you must obtain a license from Micrium (www.micrium.com).



Micrium offers free licensing for universities and students. Contact Micrium for details.

Other RTOS Providers

Altera distributes MicroC/OS-II to provide you with immediate access to an easy-to-use RTOS. In addition to MicroC/OS-II, many other RTOSs are available from third-party vendors.



For a complete list of RTOSs that support the Nios II processor, visit the [Embedded Software](#) page of the Altera® website.

The Nios II Implementation of MicroC/OS-II

Altera has ported MicroC/OS-II to the Nios II processor. Altera distributes MicroC/OS-II in the Nios II EDS, and supports the Nios II implementation of the MicroC/OS-II kernel. Ready-made, working examples of MicroC/OS-II programs are installed with the Nios II EDS. In addition, Altera development boards are preprogrammed with a web server reference design based on MicroC/OS-II and the NicheStack® TCP/IP Stack - Nios II Edition.

The Altera implementation of MicroC/OS-II is designed to be easy to use. Using the Nios II project settings, you can control the configuration for all the RTOS's modules.

You need not modify source files directly to enable or disable kernel features. Nonetheless, Altera provides the Nios II processor-specific source code in case you wish to examine it. The MicroC/OS-II source code is located in the following directories:

- Processor-specific code: *<Nios II EDS install path>/components/altera_nios2/UCOSII*
- Processor-independent code: *<Nios II EDS install path>/components/micrium_uc_osii*

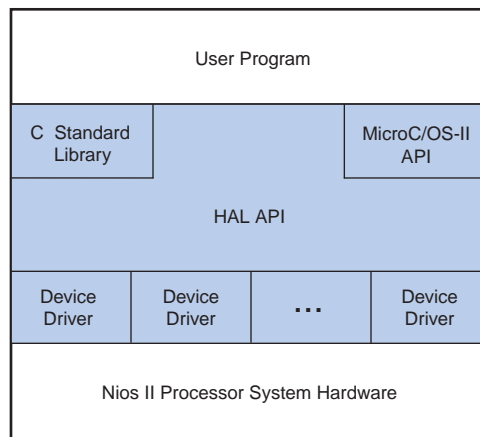
The MicroC/OS-II software package behaves like the drivers for hardware components: When MicroC/OS-II is included in a Nios II project, the header and source files from **components/micrium_uc_osii** are included in the project path, causing the MicroC/OS-II kernel to compile and link as part of the project.

MicroC/OS-II Architecture


The Altera implementation of MicroC/OS-II for the Nios II processor extends the single-threaded HAL environment to include the MicroC/OS-II scheduler and the associated MicroC/OS-II API. The complete HAL API is available to all MicroC/OS-II projects.

Figure 10-1 shows the architecture of a program based on MicroC/OS-II and its relationship to the HAL API.

Figure 10-1. Architecture of MicroC/OS-II Programs




The multi-threaded environment affects certain HAL functions.

 For details about the consequences of calling a particular HAL function in a multi-threaded environment, refer to the *HAL API Reference* chapter of the *Nios II Software Developer's Handbook*.

MicroC/OS-II Thread-Aware Debugging

When you debug a MicroC/OS-II application, the debugger can display the current state of all threads in the application, including backtraces and register values. You cannot use the debugger to change the current thread, so it is not possible to use the debugger to change threads or to single-step a different thread.

 Thread-aware debugging does not change the behavior of the target application in any way.

MicroC/OS-II Device Drivers

Each peripheral (that is, each hardware component) can provide include files and source files in the `inc` and `src` subdirectories of the component's `HAL` directory.

 For more information, refer to the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer's Handbook*.

In addition to the `HAL` directory, a component can optionally provide a `UCOSII` directory that contains code specific to the MicroC/OS-II environment. Similar to the `HAL` directory, the `UCOSII` directory contains `inc` and `src` subdirectories.

When you create a MicroC/OS-II project, these directories are added to the search paths for source and include files.

The Nios II Software Build Tools (SBT) copies the files to your BSP's `obj` subdirectory.

- For more information about specifying file paths with the Nios II SBT, refer to “Nios II Embedded Software Projects” in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer’s Handbook*.

You can use the **UCOSII** directory to provide code that is used only in a multi-threaded environment. Other than these additional search directories, the mechanism for providing MicroC/OS-II device drivers is identical to the process for any other device driver.

- For details about developing device drivers, refer to the *Developing Device Drivers for the Hardware Abstraction Layer* chapter of the *Nios II Software Developer’s Handbook*.

The HAL system initialization process calls the MicroC/OS-II function `OSInit()` before `alt_sys_init()`, which instantiates and initializes each device in the system. Therefore, the complete MicroC/OS-II API is available to device drivers, although the system is still running in single-threaded mode until the program calls `OSStart()` from within `main()`.

Thread-Safe HAL Drivers

To enable a driver to be ported between the HAL and MicroC/OS-II environments, Altera defines a set of operating system-independent macros that provide access to operating system facilities. These macros implement functionality that is only relevant to a multi-threaded environment. When compiled for a MicroC/OS-II project, the macros expand to MicroC/OS-II API calls. When compiled for a single-threaded HAL project, the macros expand to benign empty implementations. These macros are used in Altera-provided device driver code, and you can use them if you need to write a device driver with similar portability.

Table 10-1 lists the available macros and their functions.

- For more information about the functionality in the MicroC/OS-II environment, refer to *MicroC/OS-II: The Real-Time Kernel*.

The path listed for the header file is relative to the `<Nios II EDS install path>/components/micrium_uc_osii/UCOSII/inc` directory.

Table 10-1. OS-Independent Macros for Thread-Safe HAL Drivers (Part 1 of 2)

Macro	Defined in Header	MicroC/OS-II Implementation	Single-Threaded HAL Implementation
<code>ALT_FLAG_GRP(group)</code>	<code>os/alt_flag.h</code>	Create a pointer to a flag group with the name <code>group</code> .	Empty statement.
<code>ALT_EXTERN_FLAG_GRP(group)</code>	<code>os/alt_flag.h</code>	Create an external reference to a pointer to a flag group with name <code>group</code> .	Empty statement.
<code>ALT_STATIC_FLAG_GRP(group)</code>	<code>os/alt_flag.h</code>	Create a static pointer to a flag group with the name <code>group</code> .	Empty statement.
<code>ALT_FLAG_CREATE(group, flags)</code>	<code>os/alt_flag.h</code>	Call <code>OSFlagCreate()</code> to initialize the flag group pointer, <code>group</code> , with the flags value <code>flags</code> . The error code is the return value of the macro.	Return 0 (success).


Table 10–1. OS-Independent Macros for Thread-Safe HAL Drivers (Part 2 of 2)

Macro	Defined in Header	MicroC/OS-II Implementation	Single-Threaded HAL Implementation
ALT_FLAG_PEND(group, flags, wait_type, timeout)	os/alt_flag.h	Call OSFlagPend() with the first four input arguments set to group, flags, wait_type, and timeout respectively. The error code is the return value of the macro.	Return 0 (success).
ALT_FLAG_POST(group, flags, opt)	os/alt_flag.h	Call OSFlagPost() with the first three input arguments set to group, flags, and opt respectively. The error code is the return value of the macro.	Return 0 (success).
ALT_SEM(sem)	os/alt_sem.h	Create an OS_EVENT pointer with the name sem.	Empty statement.
ALT_EXTERN_SEM(sem)	os/alt_sem.h	Create an external reference to an OS_EVENT pointer with the name sem.	Empty statement.
ALT_STATIC_SEM(sem)	os/alt_sem.h	Create a static OS_EVENT pointer with the name sem.	Empty statement.
ALT_SEM_CREATE(sem, value)	os/alt_sem.h	Call OSSemCreate() with the argument value to initialize the OS_EVENT pointer sem. The return value is zero on success, or negative otherwise.	Return 0 (success).
ALT_SEM_PEND(sem, timeout)	os/alt_sem.h	Call OSSemPend() with the first two argument set to sem and timeout respectively. The error code is the return value of the macro.	Return 0 (success).
ALT_SEM_POST(sem)	os/alt_sem.h	Call OSSemPost() with the input argument sem.	Return 0 (success).

The newlib ANSI C Standard Library

Programs based on MicroC/OS-II can also call the ANSI C standard library functions. Some consideration is necessary in a multi-threaded environment to ensure that the C standard library functions are thread-safe. The newlib C library stores all global variables in a single structure referenced through the pointer `_impure_ptr`. However, the Altera MicroC/OS-II implementation creates a new instance of the structure for each task. During a context switch, the value of `_impure_ptr` is updated to point to the current task's version of this structure. In this way, the contents of the structure pointed to by `_impure_ptr` are treated as thread local. For example, through this mechanism each task has its own version of `errno`.

This thread-local data is allocated at the top of the task's stack. You must make allowance for thread-local data storage when allocating memory for stacks. In general, the `_reent` structure consumes approximately 900 bytes of data for the normal C library, or 90 bytes for the reduced-footprint C library.

 For further details about the contents of the `_reent` structure, refer to the newlib documentation. On the Windows Start menu, click **Programs > Altera > Nios II > Nios II Documentation**.


In addition, the MicroC/OS-II implementation provides appropriate task locking to ensure that heap accesses (calls to `malloc()` and `free()`) are also thread-safe.

Interrupt Service Routines for MicroC/OS-II

Implementing ISRs for MicroC/OS-II normally involves some housekeeping details, as described in *MicroC/OS-II: The Real-Time Kernel*. However, because the Nios II implementation of MicroC/OS-II is based on the HAL, several of these details are taken care of for you. The HAL performs the following housekeeping tasks for your interrupt service routine (ISR):


- Saves and restores processor registers
- Calls `OSIntEnter()` and `OSIntExit()`


The HAL also allows you to write your ISR in C, rather than assembly language.

 For more detail about writing ISRs with the HAL, refer to the *Exception Handling* chapter of the *Nios II Software Developer's Handbook*.

Implementing MicroC/OS-II Projects for the Nios II Processor

To create a program based on MicroC/OS-II, start by setting the BSP properties so that it is a MicroC/OS-II project. You can control the configuration of the MicroC/OS-II kernel using BSP settings with the Nios II SBT for Eclipse™, or on the Nios II command line.

 You do not need to edit header files (such as `OS_CFG.h`) or source code to configure the MicroC/OS-II features. The project settings are reflected in the BSP's `system.h` file; `OS_CFG.h` simply includes `system.h`.

 For a list of available MicroC/OS-II BSP settings, refer to “Settings Managed by the Software Build Tools” in the *Nios II Software Build Tools Reference* chapter of the *Nios II Software Developer's Handbook*. MicroC/OS-II settings are identified by the prefix `ucosii`. For information about how to configure MicroC/OS-II with BSP settings, refer to the *Getting Started with the Graphical User Interface, Nios II Software Build Tools*, and *Nios II Software Build Tools Reference* chapters of the *Nios II Software Developer's Handbook*. The meaning of each setting is defined fully in *MicroC/OS-II: The Real-Time Kernel*.

Document Revision History

Table 10-2 shows the revision history for this document.

Table 10-2. Document Revision History (Part 1 of 2)

Date	Version	Changes
May 2011	11.0.0	Introduction of Qsys system integration tool
February 2011	10.1.0	Removed “Referenced Documents” section.
July 2010	10.0.0	Maintenance release.

Table 10–2. Document Revision History (Part 2 of 2)

Date	Version	Changes
November 2009	9.1.0	<ul style="list-style-type: none"> ■ Introduced the Nios II Software Build Tools for Eclipse. ■ Remove tables of Nios II IDE-specific setting names. Refer solely to BSP setting names.
March 2009	9.0.0	<ul style="list-style-type: none"> ■ Reorganized and updated information and terminology to clarify role of Nios II Software Build Tools. ■ Corrected minor typographical errors.
May 2008	8.0.0	Maintenance release.
October 2007	7.2.0	<ul style="list-style-type: none"> ■ Added documentation for MicroC/OS-II development with the Nios II Software Build Tools. ■ Added description of HAL ISR support.
May 2007	7.1.0	<ul style="list-style-type: none"> ■ Added table of contents to “Introduction” section. ■ Added Referenced Documents section.
March 2007	7.0.0	Maintenance release.
November 2006	6.1.0	Maintenance release.
May 2006	6.0.0	Maintenance release.
October 2005	5.1.0	Maintenance release.
May 2005	5.0.0	Maintenance release.
December 2004	1.2	
September 2004	1.1	Added thread-aware debugging paragraph.
May 2004	1.0	Initial release.

