8. MicroC/OS-II Real-Time Operating System

NII52008-7.0.0

Introduction

This chapter describes the MicroC/OS-II real-time kernel for the Nios® II processor.

Overview

MicroC/OS-II is a popular real-time kernel produced by Micrium Inc., and is documented in the book *MicroC/OS-II - The Real Time Kernel* by Jean J. Labrosse (CMP Books). The book describes MicroC/OS-II as a portable, ROMable, scalable, preemptive, real-time, multitasking kernel. MicroC/OS-II has been used in hundreds of commercial applications since its release in 1992, and has been ported to over 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) system library for the Nios II processor. Because of the HAL, programs based on MicroC/OS-II are more portable to other Nios II hardware systems, and are resistant to changes in the underlying hardware. Furthermore, MicroC/OS-II programs have access to all HAL services, and can call the familiar HAL advanced programming interface (API) functions.

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*. Further information is also available on the Micrium website, **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 contact Micrium to obtain a license at **Licensing@Micrium.com** or **http://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 real-time operating system (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 Nios II homepage at **www.altera.com/nios2**.

The Altera Port of MicroC/OS-II

Altera ported MicroC/OS-II to the Nios II processor. Altera distributes MicroC/OS-II in the Nios II EDS, and supports the Nios II port of the MicroC/OS-II kernel. Ready-made, working examples of MicroC/OS-II programs are installed with the Nios II EDS. In fact, Nios development boards are pre-programmed with a web server reference design based on MicroC/OS-II and the Lightweight IP TCP/IP stack.

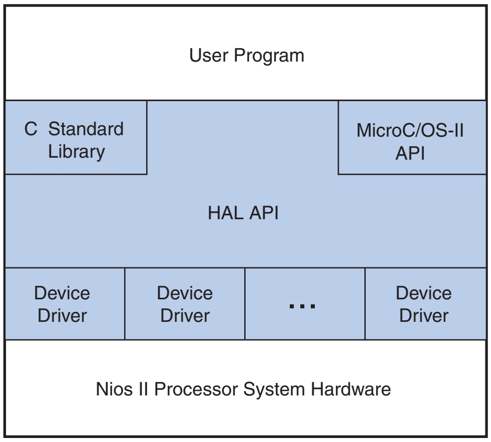
The Altera® port of MicroC/OS-II is designed to be easy-to-use from within the Nios II IDE. Using the Nios II IDE, you can control the configuration for all the RTOS's modules. You need not modify source files directly to enable or disable kernel features. Nonetheless, Altera provides the Nios II processor-specific source code if you ever wish to examine it. The code is provided in directory <*Nios II EDS install path*>**/components/altera_nios/UCOSII**. The processor-independent code resides in <*Nios II EDS install path*>**/components/micrium_uc_osii**. The MicroC/OS-II software component behaves like the drivers for SOPC Builder hardware components: When MicroC/OS-II is included in a Nios II integrated development environment (IDE) 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 into the project.

MicroC/OS-II Architecture

The Altera port of MicroC/OS-II for the Nios II processor is essentially a superset of the HAL. It is the HAL environment extended by the inclusion of the MicroC/OS-II scheduler and the associated MicroC/OS-II API. The complete HAL API is available from within MicroC/OS-II projects.

Figure 8–1 shows the architecture of a program based on MicroC/OS-II and the relationship to the HAL.

Figure 8–1. Architecture of MicroC/OS-II Programs



The multi-threaded environment affects certain HAL functions.

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

MicroC/OS-II Thread-Aware Debugging

When debugging a MicroC/OS-II application, the debugger can display the current state of all threads within 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 (i.e., an SOPC Builder component) can provide include files and source files within the **inc** and **src** subdirectories of the component's **HAL** directory.

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

In addition to the **HAL** directory, a component may elect to 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. These directories are automatically added to the source and include search paths when building MicroC/OS-II projects in the Nios II IDE.

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 described in the *Developing Device Drivers for the HAL* 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 allow the same driver to be portable across the HAL and MicroC/OS-II environments, Altera defines a set of OS-independent macros that provide access to operating system facilities. When compiled for a MicroC/OS-II project, the macros expand to a MicroC/OS-II API call. 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 drivers with similar portability.

Table 8–1 lists the available macros and their function.

For more information on the functionality in the MicroC/OS-II environment, see *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 8–1. OS-Independent Macros for Thread-Safe HAL Drivers (Part 1 of 2)

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

Table 8–1. 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_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 upon 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 within a single structure referenced through the pointer `_impure_ptr`. However, the Altera MicroC/OS-II port creates a new instance of the structure for each task. Upon 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. Therefore, you need to make allowance 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 on the contents of the `_reent` structure, refer to the newlib documentation. On the Windows Start menu, click Programs, Altera, Nios II <version>, **Nios II Documentation**.

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

Implementing MicroC/OS-II Projects in the Nios II IDE

To create a program based on MicroC/OS-II, you must first set the properties for the system library to a MicroC/OS-II project. From there, the Nios II IDE offers RTOS options that allow you to control the configuration of the MicroC/OS-II kernel.

Traditionally, you had to configure MicroC/OS-II using `#define` directives in the file **OS_CFG.h**. Instead, the Nios II IDE provides a GUI that allows you to configure each option. Therefore, you do not need to edit header files or source code to configure the MicroC/OS-II features. The GUI settings are reflected in the system library's **system.h** file; **OS_CFG.h** simply includes **system.h**.

The following sections define the MicroC/OS-II settings available from the Nios II IDE. The meaning of each setting is defined fully in the *MicroC/OS-II – The Real-Timer Kernel* chapter of the *MicroC/OS-II Configuration Manual*.

For step-by-step instructions on how to create a MicroC/OS-II project in the Nios II IDE, refer to *Using the MicroC/OS-II RTOS with the Nios II Processor Tutorial*.

MicroC/OS-II General Options

Table 8–2 shows the general options.

Table 8–2. General Options (Part 1 of 2)

Option	Description
Maximum number of tasks	Maps onto the `#define OS_MAX_TASKS`. Must be at least 2
Lowest assignable priority	Maps on the `#define OS_LOWEST_PRIO`. Maximum allowable value is 63.
Enable code generation for event flags	Maps onto the `#define OS_FLAG_EN`. When disabled, event flag settings are also disabled. See "Event Flags Settings" on page 8–8.
Enable code generation for mutex semaphores	Maps onto the #define `OS_MUTEX_EN`. When disabled, mutual exclusion semaphore settings are also disabled. See "Mutex Settings" on page 8–8
Enable code generation for semaphores	Maps onto the `#define OS_SEM_EN`. When disabled, semaphore settings are also disabled. See "Semaphores Settings" on page 8–9.
Enable code generation for mailboxes	Maps onto the `#define OS_MBOX_EN`. When disabled, mailbox settings are also disabled. See "Mailboxes Settings" on page 8–9.

Table 8–2. General Options (Part 2 of 2)

Option	Description
Enable code generation for queues	Maps onto the `#define OS_Q_EN`. When disabled, queue settings are also disabled. See "Queues Settings" on page 8–9.
Enable code generation for memory management	Maps onto the `#define OS_MEM_EN`. When disabled, memory management settings are also disabled. See "Memory Management Settings" on page 8–10.

Event Flags Settings

Table 8–3 shows the event flag settings.

Table 8–3. Event Flags Settings

Setting	Description
Include code for wait on clear event flags	Maps on `#define OS_FLAG_WAIT_CLR_EN`.
Include code for `OSFlagAccept()`	Maps on `#define OS_FLAG_ACCEPT_EN`.
Include code for `OSFlagDel()`	Maps on `#define OS_FLAG_DEL_EN`.
Include code for `OSFlagQuery()`	Maps onto the `#define OS_FLAG_QUERY_EN`.
Maximum number of event flag groups	Maps onto the `#define OS_MAX_FLAGS`.
Size of name of event flags group	Maps onto the `#define OS_FLAG_NAME_SIZE`.

Mutex Settings

Table 8–4 shows the mutex settings.

Table 8–4. Mutex Settings

Setting	Description
Include code for `OSMutexAccept()`	Maps onto the `#define OS_MUTEX_ACCEPT_EN`.
Include code for `OSMutexDel()`	Maps onto the `#define OS_MUTEX_DEL_EN`.
Include code for `OSMutexQuery()`	Maps onto the `#define OS_MUTEX_QUERY_EN`.

Semaphores Settings

Table 8–5 shows the semaphores settings.

Table 8–5. Semaphores Settings

Setting	Description
Include code for `OSSemAccept()`	Maps onto the `#define OS_SEM_ACCEPT_EN`.
Include code for `OSSemSet()`	Maps onto the `#define OS_SEM_SET_EN`.
Include code for `OSSemDel()`	Maps onto the `#define OS_SEM_DEL_EN`.
Include code for `OSSemQuery()`	Maps onto the `#define OS_SEM_QUERY_EN`.

Mailboxes Settings

Table 8–6 shows the mailbox settings.

Table 8–6. Mailboxes Settings

Setting	Description
Include code for `OSMboxAccept()`	Maps onto `#define OS_MBOX_ACCEPT_EN`.
Include code for `OSMBoxDel()`	Maps onto `#define OS_MBOX_DEL_EN`.
Include code for `OSMboxPost()`	Maps onto `#define OS_MBOX_POST_EN`.
Include code for `OSMboxPostOpt()`	Maps onto `#define OS_MBOX_POST_OPT_EN`.
Include code fro `OSMBoxQuery()`	Maps onto `#define OS_MBOX_QUERY_EN`.

Queues Settings

Table 8–7 shows the queues settings.

Table 8–7. Queues Settings (Part 1 of 2)

Setting	Description
Include code for `OSQAccept()`	Maps onto `#define OS_Q_ACCEPT_EN`.
Include code for `OSQDel()`	Maps onto `#define OS_Q_DEL_EN`.
Include code for `OSQFlush()`	Maps onto `#define OS_Q_FLUSH_EN`.
Include code for `OSQPost()`	Maps onto `#define OS_Q_POST_EN`.
Include code for `OSQPostFront()`	Maps onto `#define OS_Q_POST_FRONT_EN`.
Include code for `OSQPostOpt()`	Maps onto `#define OS_Q_POST_OPT_EN`.

Table 8–7. Queues Settings (Part 2 of 2)

Setting	Description
Include code for `OSQQuery()`	Maps onto `#define OS_Q_QUERY_EN`.
Maximum number of Queue Control blocks	Maps onto `#define OS_MAX_QS`.

Memory Management Settings

Table 8–8 shows the memory management settings.

Table 8–8. Memory Management Settings

Setting	Description
Include code for `OSMemQuery()`	Maps onto `#define OS_MEM_QUERY_EN`.
Maximum number of memory partitions	Maps onto `#define OS_MAX_MEM_PART`.
Size of memory partition name	Maps onto `#define OS_MEM_NAME_SIZE`.

Miscellaneous Settings

Table 8–9 shows the miscellaneous settings.

Table 8–9. Miscellaneous Settings (Part 1 of 2)

Setting	Description
Enable argument checking	Maps onto `#define OS_ARG_CHK_EN`.
Enable `uCOS-II` hooks	Maps onto `#define OS_CPU_HOOKS_EN`.
Enable debug variables	Maps onto `#define OS_DEBUG_EN`.
Include code for `OSSchedLock()` and `OSSchedUnlock()`	Maps onto `#define OS_SCHED_LOCK_EN`.
Enable tick stepping feature for `uCOS-View`	Maps onto `#define OS_TICK_STEP_EN`.
Enable statistics task	Maps onto `#define OS_TASK_STAT_EN`.
Check task stacks from statistics task	Maps onto `#define OS_TASK_STAT_STK_CHK_EN`.
Statistics task stack size	Maps onto `#define OS_TASK_STAT_STK_SIZE`.
Idle task stack size	Maps onto `#define OS_TASK_IDLE_STK_SIZE`.

Table 8–9. Miscellaneous Settings (Part 2 of 2)

Setting	Description
Maximum number of event control blocks	Maps onto `#define OS_MAX_EVENTS 60`.
Size of semaphore, mutex, mailbox, or queue name	Maps onto `#define OS_EVENT_NAME_SIZE`.

Task Management Settings

Table 8–10 shows the task management settings.

Table 8–10. Task Management Settings

Setting	Description
Include code for `OSTaskChangePrio()`	Maps onto `#define OS_TASK_CHANGE_PRIO_EN`.
Include code for `OSTaskCreate()`	Maps onto `#define OS_TASK_CREATE_EN`.
Include code for `OSTaskCreateExt()`	Maps onto `#define OS_TASK_CREATE_EXT_EN`.
Include code for `OSTaskDel()`	Maps onto `#define OS_TASK_DEL_EN`.
Include variables in `OS_TCB` for profiling	Maps onto `#define OS_TASK_PROFILE_EN`.
Include code for `OSTaskQuery()`	Maps onto `#define OS_TASK_QUERY_EN`.
Include code for `OSTaskSuspend()` and `OSTaskResume()`	Maps onto `#define OS_TASK_SUSPEND_EN`.
Include code for `OSTaskSwHook()`	Maps onto `#define OS_TASK_SW_HOOK_EN`.
Size of task name	Maps onto `#define OS_TASK_NAME_SIZE`.

Time Management Settings

Table 8–11 shows the time management settings.

Table 8–11. Time Management Settings (Part 1 of 2)

Setting	Description
Include code for `OSTimeDlyHMSM()`	Maps onto `#define OS_TIME_DLY_HMSM_EN`.
Include code `OSTimeDlyResume()`	Maps onto `#define OS_TIME_DLY_RESUME_EN`.

Table 8–11. Time Management Settings (Part 2 of 2)

Setting	Description
Include code for `OSTimeGet()` and `OSTimeSet()`	Maps onto `#define OS_TIME_GET_SET_EN`.
Include code for `OSTimeTickHook()`	Maps onto `#define OS_TIME_TICK_HOOK_EN`.

Document Revision History

Table 8–12 shows the revision history for this document.

Table 8–12. Document Revision History

Date & Document Version	Changes Made	Summary of Changes
March 2007, v7.0.0	No change from previous release.	
November 2006, v6.1.0	No change from previous release.	
May 2006, v6.0.0	No change from previous release.	
October 2005, v5.1.0	No change from previous release.	
May 2005, v5.0.0	No change from previous release.	
December 2004 v1.1	Added thread-aware debugging paragraph.	
May 2004 v1.0	First publication.	