

Micrium, Inc.

949 Crestview Circle
Weston, FL 33327
U.S.A.
www.Micrium.com

μC/OS-II

The Real-Time kernel

V2.60

Release Notes

© Copyright 2002, Micrium, Inc.
All Rights reserved

Phone: +1 954 217 2036

FAX: +1 954 217 2037

V2.60

(2002/09/28)

Changes were made to V2.52 for the following reasons:

- a) To fix minor issues with V2.52.
- b) To simplify FAA Level A certification by removing all MCDC (Modified Condition Decision Coverage).
- c) To follow most of the guidelines of *The Motor Industry Software Reliability Association* “Guidelines for the use of the C language in vehicle based software”.
- d) To add support for kernel awareness.
- e) To directly support μ C/OS-View.
- f) Added new features.
- g) Made some changes to the code.

FIXED ISSUES WITH V2.52

Bug V2.52-001:

In `OS_CORE.C`, function `OS_InitMisc()`, there is no need to test `OS_TASK_CREATE_EXT_EN`:

```
#if (OS_TASK_STAT_EN > 0) && (OS_TASK_CREATE_EXT_EN > 0)
    OSIdleCtrRun = 0L;
    OSIdleCtrMax = 0L;
    OSStatRdy    = FALSE;
#endif
```

The correct code is thus:

```
#if OS_TASK_STAT_EN > 0
    OSIdleCtrRun = 0L;
    OSIdleCtrMax = 0L;
    OSStatRdy    = FALSE;
#endif
```

Bug V2.52-002:

In `OS_TASK.C`, function `OSTaskDel()`, the variable `self` was never used. The variable is now removed.

Bug V2.52-003:

In `OS_TASK.C`, function `OSTaskStkChk()` was missing a test. The incorrect code was:

```
ptcb = OSTCBPrioTbl[prio];
if (ptcb == (OS_TCB *)0) {
    OS_EXIT_CRITICAL();
    return (OS_TASK_NOT_EXIST);
}
```

The correct code is:

```
ptcb = OSTCBPrioTbl[prio];
if (ptcb == (OS_TCB *)0 || ptcb == (OS_TCB *)1) {
    OS_EXIT_CRITICAL();
    return (OS_TASK_NOT_EXIST);
}
```

Bug V2.52-004:

In OS_MUTEX.C, function OSMutexPost() the condition to check to see if the current task is the owner of the mutex has been changed from:

```
if (OSTCBCur->OSTCBPrio != pip &&
    OSTCBCur->OSTCBPrio != prio) {
    OS_EXIT_CRITICAL();
    return (OS_ERR_NOT_MUTEX_OWNER);
}
```

To:

```
if (OSTCBCur != (OS_TCB *)pevent->OSEventPtr) {
    OS_EXIT_CRITICAL();
    return (OS_ERR_NOT_MUTEX_OWNER);
}
```

This change allows a task to obtain multiple mutexes. A task could thus have the following code:

```
Acquire Mutex #1;
Acquire Mutex #2;
Release Mutex #2;
Release Mutex #1;
```

Mutexes MUST be released in the same order as they were acquired.

Bug V2.52-005:

In OS_MUTEX.C, the function OSMutexPend() was changed to allow a mutex owner to pend on another kernel object such as a semaphore. In other words, a task could have the following code:

```
Acquire Mutex;          /* Mutex is available, task now owns it      */
Acquire Semaphore;      /* Semaphore is NOT available, suspend task! */
:
:
```

Then, a high priority task that would call OSMutexPend() on the same mutex would notice that the mutex owner has a lower priority than the task that needs the mutex. OSMutexPend() would then raise the priority of the task that owns the mutex and will notice that the task is also waiting on a semaphore. OSMutexPend() would then change the priority of the mutex owner in the semaphore wait list.

SIMPLIFYING 'FAA LEVEL A' CERTIFICATION

Changes were made to V2.52 to remove all MCDC (Modified Condition Decision Coverage). MCDCs are basically conditionals with multiple possible outcomes. For example, in the following code, there are eight (8) possible outcomes based on the different values of a, b, c, d, e and f:

```
if (a == b && c == d && e == f) {  
    /* Conditions met */  
}
```

A better way to write the above code (from a certification perspective) is shown below:

```
if (a == b) {  
    if (c == d) {  
        if (e == f) {  
            /* Conditions met */  
        }  
    }  
}
```

I went through all the μ C/OS-II code and removed the MCDCs. Of course, the code behaves exactly the same as before.

FOLLOWED MOST OF THE MISRA GUIDELINES

MISRA stands for “The Motor Industry Software Reliability Association” and this association published back in April 1998, a list of 127 guidelines for programming applications using the C programming language. You can obtain this document by visiting:

<http://www.misra.org.uk>

The document is called:

“Guidelines For The Use Of The C Language In Vehicle Based Software”
ISBN 0 9524156 9 0

It so happens that μ C/OS-II was written by following most of the MISRA guidelines even before the guidelines were ever published. At this time, μ C/OS-II is not ‘compliant’ with the guidelines but simply follows most of them.

One of the most significant changes to μ C/OS-II’s code is the removal of assignments inside conditionals. For instance, the following code:

```
if ((pevent->OSEventTbl[y] &= ~bitx) == 0) {  
    /* ... */  
}
```

Has been replaced by:

```
pevent->OSEventTbl[y] &= ~bitx;  
if (pevent->OSEventTbl[y] == 0) {  
    /* ... */  
}
```

SUPPORT OF KERNEL AWARE DEBUGGERS

Variables and constants have been added to help support kernel aware debuggers. Specifically, a number of variables can be queried by a debugger to find out about compiled-in options. For example, the debugger can find out the size of an `OS_TCB`, μ C/OS-II's version number, the size of an event flag group (`OS_FLAG_GRP`) and much more. Those variables are enabled by `OS_DEBUG_EN` in `OS_CFG.H`.

SUPPORT OF μ C/OS-View

Variables in `OS_TCB` have been added (see `OS_TASK_PROFILE_EN`) to support profiling tools such as μ C/OS-View.

Also `OS_TaskStat()` can now check the stack of each of the active tasks (see `OS_TASK_STAT_STK_CHK_EN`).

An `OS_TCB` can also contain the name of each task which can then be displayed on the μ C/OS-View Windows application.

μ C/OS-View can 'step' tick interrupts one at a time. In other words, through a command sent by a user of μ C/OS-View, μ C/OS-II can process one tick at a time. Each tick requires a user to press a key from the μ C/OS-View application.

ADDED NEW FEATURES

1) Find out which flag(s) caused task to wakeup.

Added the function `OSFlagPendGetFlagsRdy()` (file `OS_FLAG.C`) to allow to determine which flag(s) caused the current task to become ready. In other words, you will now be able to know what event flag(s) caused the pending task to wake up.

2) Posting NULL pointers to queues.

It is now possible to send NULL pointer message through queues. `OSQPost()` and `OSQPostFront()` no longer blocks NULL pointers from being deposited into queues. This means that `OSQPend()` will thus be able to receive NULL pointer messages. You should now check the status of the `err` argument to determine whether the return from the `pend` was caused by a timeout or the actual reception of a message.

Because of this change, I had to change the API for `OSQAccept()` so that it returns an error code indicating the outcome of the call.

3) Assigning names to Tasks and other kernel objects.

It is now possible to assign names to Tasks, Memory Partitions, Semaphores, Mutexes, Event Flags, Mailboxes and Queues. The names are useful when debugging applications. You assign names by calling one of the following functions:

```
OSEventNameSet ()
OSFlagNameSet ()
OSMemNameSet ()
OSTaskNameSet ()
```

You can obtain the name of a task or a kernel object by calling the following functions:

```
OSEventNameGet ()
OSFlagNameGet ()
OSMemNameGet ()
OSTaskNameGet ()
```

This version doesn't allow you to manipulate kernel objects using names. For example, you can't delete a task by specifying its name, you can't post to a queue by specifying the queue by its name, etc.

4) Disable calls to OSTaskSwHook() and OSTimeTickHook()

It is now possible to disable (at compile time) the need to have the functions OSTaskSwHook() and OSTimeTickHook(). This feature was requested because of the overhead involved in calling empty functions during a context switch and also every tick.

To disable OSTaskSwHook(), simply set OS_TASK_SW_HOOK_EN to 0 in OS_CFG.H. Of course, the port (OS_CPU_A.ASM) for the processor you are using must not call OSTaskSwHook().

To disable OSTimeTickHook(), simply set OS_TIME_TICK_HOOK_EN to 0 in OS_CFG.H.

5) Added variables in OS_TCB to allow profiling

Variables have been added to OS_TCB to allow each task to be profiled. In other words, µC/OS-II contains variables that register the number of time a task is 'switched-in', how long a task takes to execute, how much stack space each task consumes and more. These variables have been added to better support µC/OS-View and other profiling tools.

6) Added tick stepping support for μ C/OS-View

μ C/OS-View can now ‘halt’ μ C/OS-II’s tick processing and allow you to issue ‘step’ commands from μ C/OS-View. In other words, μ C/OS-View can prevent μ C/OS-II from calling `OSTimeTick()` so that timeouts and time delays are no longer processed. However, though a keystroke from μ C/OS-View, you can execute a single tick at a time. If enabled, `OSTimeTickHook()` is still executed at the regular tick rate in case you have time critical items to take care of in your application.

7) Added new #defines in OS_CFG.H

Instead of editing your `OS_CFG.H`, I recommend that you copy one of the `OS_CFG.H` files provided with the V2.60 release and then modify it to satisfy your application requirements.

`OS_DEBUG_EN`

This #define adds ROM constants to help support kernel aware debuggers. Specifically, a number of named ROM variables can be queried by a debugger to find out about compiled-in options. For example, the debugger can find out the size of an `OS_TCB`, μ C/OS-II’s version number, the size of an event flag group (`OS_FLAG_GRP`) and much more.

`OS_EVENT_NAME_SIZE`

This #define determines the size of ASCII strings used to name either semaphores, mutexes, mailboxes and queues. If set to 0, this feature will be disabled: no RAM will be allocated and the functions `OSEventNameGet()` and `OSEventNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the NUL character and if you do use a non-zero value, you should have a minimum of 2 for this value.

`OS_FLAG_NAME_SIZE`

This #define determines the size of ASCII strings used to name event flag groups. If set to 0, this feature will be disabled: no RAM will be allocated and the functions `OSFlagNameGet()` and `OSFlagNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the NUL character and if you do use a non-zero value, you should have a minimum of 2 for this value.

`OS_MEM_NAME_SIZE`

This #define determines the size of ASCII strings used to name memory partitions. If set to 0, this feature will be disabled: no RAM will be allocated and the functions `OSMemNameGet()` and `OSMemNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the NUL character and if you do use a non-zero value, you should have a minimum of 2 for this value.

`OS_TASK_NAME_SIZE`

This #define determines the size of ASCII strings used to name tasks. If set to 0, this feature will be disabled: no RAM will be allocated and the functions `OSTaskNameGet()` and `OSTaskNameSet()` will not be compiled. If set to a non-zero value, it determines the number of bytes allocated for the name. Please note that you need to accommodate for the NUL character and if you do use a non-zero value, you should have a minimum of 2 for this value.

`OS_TASK_PROFILE_EN`

This `#define` is used to allocate storage for variables used for run-time task profiling. These variables are used by μ C/OS-View and some kernel aware debuggers.

`OS_TASK_STAT_STK_CHK_EN`

This `#define` allows the statistic task to do run-time checking of all the stacks of all the active tasks. In other words, when set to 1, `OS_TaskStat()` calls the function `OS_TaskStatStkChk()`. Of course, for this to happen, `OS_TASK_STAT_EN` must also be set to 1.

`OS_TASK_SW_HOOK_EN`

It is now possible to disable (at compile time) the need to have the functions `OSTaskSwHook()`. This feature was requested because of the overhead involved in calling empty functions during a context switch and also every tick. To disable `OSTaskSwHook()`, simply set `OS_TASK_SW_HOOK_EN` to 0 in `OS_CFG.H`. Of course, the port (`OS_CPU_A.ASM`) for the processor you are using must not call `OSTaskSwHook()`.

`OS_TICK_STEP_EN`

μ C/OS-View can now ‘halt’ μ C/OS-II’s tick processing and allow you to issue ‘step’ commands from μ C/OS-View. In other words, μ C/OS-View can prevent μ C/OS-II from calling `OSTimeTick()` so that timeouts and time delays are no longer processed. However, though a keystroke from μ C/OS-View, you can execute a single tick at a time. If `OS_TIME_TICK_HOOK_EN` (see below) is set to 1, `OSTimeTickHook()` is still executed at the regular tick rate in case you have time critical items to take care of in your application.

`OS_TIME_TICK_HOOK_EN`

It is now possible to disable (at compile time) the need to have the functions `OSTimeTickHook()`. This feature was requested because of the overhead involved in calling empty functions during a context switch and also every tick. To disable `OSTimeTickHook()`, simply set `OS_TIME_TICK_HOOK_EN` to 0 in `OS_CFG.H`.

CHANGES

1) Added ‘extern C’ in `uCOS_II.H`

An “extern C” statement has been added to allow you to compile μ C/OS-II using a C++ compiler.

2) Renamed ALL files to lower case

All the μ C/OS-II files have been renamed to lower case to make it easier to compile under UNIX environments.

3) Changed the structure of `OSTaskChangePrio()`

I changed the structure of the code for `OSTaskChangePrio()` to reduce the indentation, simplify and make the code cleaner. I also removed the re-enabling of

interrupts when computing `x`, `y`, `bitx` and `bity`. There is thus, there is no need to 'reserve' the `OSTCBPrioTbl[]` entry by setting it to `(OS_TCB *) 1`.

4) Assigning a NULL pointer to `OSTCBStkPtr`

I now assign a NULL pointer to `OSTCBStkPtr` when the free list of TCBs is created and when a task is deleted.

5) Posting NULL pointers to queues.

Because it is now possible to post NULL pointers to queues, I had to change the API for `OSQAccept()` so that it returns an error code indicating the outcome of the call.

6) Removed assignments inside `if ()` statements.

Code like shown below:

```
if ((pevent->OSEventTbl[y] &= ~bitx) == 0) {
    /* ... */
}
```

Has been replaced by:

```
pevent->OSEventTbl[y] &= ~bitx;
if (pevent->OSEventTbl[y] == 0) {
    /* ... */
}
```

7) Removed MCDCs.

Code like shown below:

```
if (a == b && c == d && e == f) {
    /* Conditions met */
}
```

Has been replaced by the following code:

```
if (a == b) {
    if (c == d) {
        if (e == f) {
            /* Conditions met */
        }
    }
}
```

8) Added memset() to clear RAM

Added calls to `memset()` to clear (i.e. initialize) the `OSTCBPrioTbl[]`, `OSTCBTbl[]`, `OSMemTbl[]`, `OSFlagTbl[]` and `OSEventTbl[]`. The reason `memset()` was used was for speed and to reduce code size. These tables are cleared during initialization to prevent a kernel aware debugger to display uninitialized values.

In most cases, the initialization code for the different kernel objects has also been reduced.

V2.52

(2002/01/26)

This release is for the new edition of the book: *MicroC/OS-II, The Real-Time Kernel, 2nd Edition*.

V2.52 fixes minor bugs reported in V2.51.

Bug V2.51-003:

In `uCOS_II.H`, the following code was corrected as follows:

```
#ifndef OS_FLAG_QUERY_EN
#error "OS_CFG.H, Missing OS_FLAG_DEL_EN: Include code for OSFlagQuery()"
```

needs to be:

```
#ifndef OS_FLAG_QUERY_EN
#error "OS_CFG.H, Missing OS_FLAG_QUERY_EN: Include code for OSFlagQuery()"
```

Bug V2.51-002:

In `OS_Q.C`, the following code was corrected as follows:

The function `OSQQuery()` contains a BUG in the following code which is towards the end of the function.

```
pq = (OS_Q *)pevent->OSEventPtr;
if (pq->OSQEntries > 0) {
    pdata->OSMsg = pq->OSQOut; /* Get next message to return if available */
} else {
    pdata->OSMsg = (void *)0;
}
```

The **CORRECT** code is shown below. Note that `pq->OSQOut` was missing the `*`.

```
pq = (OS_Q *)pevent->OSEventPtr;
if (pq->OSQEntries > 0) {
    pdata->OSMsg = *pq->OSQOut; /* Get next message to return if available */
} else {
    pdata->OSMsg = (void *)0;
}
```

Bug V2.51-001:

In `OS_CPU_A.ASM`, the following code was corrected as follows:

The *NEW* ISRs MUST check to see if `OSIntNesting == 1` *BEFORE* you save the `SP` in the current task's `OS_TCB`. The incorrect 'pseudo' code is:

```
OSTCBCur->OSTCBStkPtr = SP          /* Save SP onto current task's stack */
```

and should be:

```
if (OSIntNesting == 1) {  
    OSTCBCur->OSTCBStkPtr = SP      /* Save SP onto current task's stack */  
}
```

The reason we need this change is that we don't want to save the current value of `SP` if the ISR is for a nested ISR!

V2.52 adds a few minor changes to V2.51.

OS_CORE.C:

I decided to split `OSInit()` into calls to multiple functions to make the code cleaner. The new functions should be self-explanatory:

```
static void OS_InitEventList(void);  
static void OS_InitMisc(void);  
static void OS_InitRdyList(void);  
static void OS_InitTaskIdle(void);  
static void OS_InitTaskStat(void);  
static void OS_InitTCBList(void);
```

In `OSIntEnter()`, I removed the `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()` macros because it is assumed that `OSIntEnter()` will be called with interrupts disabled. Also, I added a check to make sure `OSRunning` is set to `TRUE`.

In `OSIntExit()`, I added a check to make sure `OSRunning` is set to `TRUE`.

In `OSTimeTick()`, I added a check to make sure `OSRunning` is set to `TRUE` before going through the `OS_TCBs`.

In `OS_TaskStat()`, I changed the equation to prevent overflowing the calculation on very fast CPUs. The equation was written as:

```
CPU Usage (%) = 100 - 100 * OSIdleCtr / OSIdleCtrMax;
```

Because the compiler would first perform the $100 * OSIdleCtr$ operation, an `OSIdleCtr` greater than 42,949,763 would overflow the calculation and thus report an incorrect result. The equation is now written as:

$$\text{CPU Usage (\%)} = 100 - OSIdleCtr * (OSIdleCtrMax / 100);$$

This allows `OSIdleCtr` to reach 4,294,967,295 (i.e. $2^{32}-1$) before the equation fails. I don't expect this to happen for a while since `OSIdleCtr` is incremented in a loop. The loop contains instructions that would consume a few CPU cycles each iteration.

OS_MBOX.C:

In `OSMboxPend()` (`OS_MBOX.C`), I moved the check for `OSIntNesting` at the beginning of the function because you should NEVER call `OSMboxPend()` from an ISR.

OS_Q.C:

In `OSQPend()` (`OS_Q.C`), I moved the check for `OSIntNesting` at the beginning of the function because you should NEVER call `OSQPend()` from an ISR.

OS_SEM.C:

In `OSSemPend()` (`OS_SEM.C`), I moved the check for `OSIntNesting` at the beginning of the function because you should NEVER call `OSSemPend()` from an ISR.

V2.51

(2001/06/09)

Two weeks ago, I released V2.05 and today, I found a bug in it (bug V205-001). I decided to slightly change the numbering system of releases. Complex releases (like V2.04 to V2.05) will now increase by 0.10 and minor (bug fixes or slight improvements) will now be increasing by 0.01. This means that V2.51 is now called V2.50 and with this bug fix, the release is V2.51. The reason this is done is to allow you to call `OSVersion()` and get the proper release number. If I didn't change the numbering system, I would have had to call the release with the bug correction V2.06. I was reserving such releases as major releases.

Bug V2.51-001:

In the NEW port file, an ISR MUST first check to see if `OSIntNesting == 1` before we save the SP in the current task `OS_TCB`. This bug only applies to the NEW algorithm for the port files and thus does NOT affect previous ports.

See **New Algorithm For Ports** at the end of the V2.51 notes.

V2.51 is a big upgrade for μ C/OS-II for the following reasons:

- 1) In this release, I added Event Flags (see `OS_FLAG.C`). Event flags are described in AN-1007 which can be downloaded from www.Micrium.com.
- 2) I received numerous e-mails requesting to reduce the footprint of μ C/OS-II to a minimum. To address this issue, I added a number of `#define` constants in `OS_CFG.H` which allow you to take out most of the features in μ C/OS-II that you might not be using. Specifically, there are `#defines` to remove the code for `OS???Accept()`, `OS???Query()`, `OS???Post()`, `OSSchedLock()` and `OSSchedUnlock()` and more.
- 3) This release comes with NEW ports for the Intel 80x86. These ports have been revised to REMOVE the dependency on compilers. Specifically, you no longer need to change the function `OSIntCtxSw()` in order to adjust the value of the Stack Pointer (i.e. the `SP`) register based on compiler options. The modification to accomplish this feature can ALSO be added to most processor ports!

WARNING

If you use the NEW port files in your product you **WILL** need to change ALL your Interrupt Service Routines (ISRs) to handle the new way the port works.

See **New Algorithm For Ports** at the end of the V2.51 notes.

- 4) All μ C/OS-II **internal** functions are now prefixed with `OS_` instead of `OS`. This allows you to immediately determine that these functions should NOT be called by your application. Also, these functions have been moved at the end of their respective file to get them 'out-of-the-way'.
- 5) `OS_TaskIdle()` now calls `OSTaskIdleHook()` to allow you to do such things as STOP the CPU to conserve power when running the idle task. You will need to add code in `OSTaskIdleHook()` to execute whatever is necessary for your CPU to enter its power down mode.

- 6) I added `OSMboxPostOpt()` and `OSQPostOpt()`. The new calls allow you to 'broadcast' a message to **all** tasks waiting on either a message mailbox or a message queue. In addition, `OSQPostOpt()` can replace both `OSQPost()` AND `OSQPostFront()`. This was done to further reduce the amount of code space needed by μ C/OS-II. In other words, you can start using `OSQPostOpt()` INSTEAD of `OSQPost()` and `OSQPostFront()` and thus save a significant amount of code space.
- 7) Added `#error` directives in `uCOS_II.H` to have the compiler complain whenever there are missing `#defines` in your application. This will be useful to ensure that you have not forgotten any of the NEW `#defines` added in V2.51.
- 8) Previous versions required that you declared a minimum of 2 event control blocks, 2 message queues, and 2 memory partitions. V2.51 now allows you to reduce the RAM footprint by allowing you to declare only ONE of each of the data structures mentioned (and well as only 1 event flag group). In other words, you can now specify in `OS_CFG.H`:

```
#define OS_MAX_EVENTS      1
#define OS_MAX_FLAGS      1
#define OS_MAX_MEM_PART   1
#define OS_MAX_QS         1
```

- 9) All conditional compilation is now done as follows:

```
#if condition_name > 0
```

instead of:

```
#if condition_name
```

The condition name is checked for a non-zero value to enable the code. This will allow the compiler to complain in case you forget to define `condition_name`.

10) V2.51 correct the four know bugs that were reported in V2.04.

V2.04-001:

The wrong argument was being passed to the call `OSTaskCreateHook()` in `OSTCBInit()`. The bad code was:

```
OSTaskCreateHook(OSTCBPrioTbl[prio]);
```

It is now:

```
OSTaskCreateHook(ptcb);
```

V2.04-002:

The test in `OSMutexPost()` to see if the posting task owns the MUTEX was incorrect. The correct test needed to have `&&` instead of `||` as follows:

```
if (OSTCBCur->OSTCBPrio != pip &&
    OSTCBCur->OSTCBPrio != prio) {
    OS_EXIT_CRITICAL();
    return (OS_ERR_NOT_MUTEX_OWNER);
}
```

V2.04-003:

The function `OSMutexDel()` needed to release the priority of the PIP. The following line was added in `OSMutexDel()`:

```
OSTCBPrioTbl[pip] = (OS_TCB *)0;
```

V2.04-004:

The function prototype for `OSMutexDel()` needed to be added in `uCOS_II.H`.

OS_CFG.H:

Added a number of `#define` in `OS_CFG.H` to allow you to reduce the amount of code and data space. The reason this is done using `#defines` instead of simply using a librarian is to prevent having to support a large number of librarians and also to ensure that data space is also reduced when un-needed features (i.e. functions) also require data storage.

`OS_MAX_FLAGS` is used to determine how many event flags your application will support.

`OS_FLAG_EN` to Enable (1) or Disable (0) code generation for ALL event flag services and data storage. Also, `OS_FLAG_WAIT_CLR_EN` allows you to Enable (1) or Disable (0) code generation for code to wait for 'cleared' event flags.

The following table summarizes all the other `#define` constants ADDED in V2.51. The `#defines` are set to 1 by default, enabling the code.

#define name in OS_CFG.H	... to enable the function:
<code>OS_FLAG_ACCEPT_EN</code>	<code>OSFlagAccept()</code>
<code>OS_FLAG_DEL_EN</code>	<code>OSFlagDel()</code>
<code>OS_FLAG_QUERY_EN</code>	<code>OSFlagQuery()</code>
<code>OS_MBOX_ACCEPT_EN</code>	<code>OSMboxAccept()</code>
<code>OS_MBOX_POST_EN</code>	<code>OSMboxPost()</code>
<code>OS_MBOX_POST_OPT_EN</code>	<code>OSMboxPostOpt()</code>
<code>OS_MBOX_QUERY_EN</code>	<code>OSMboxQuery()</code>
<code>OS_MEM_QUERY_EN</code>	<code>OSMemQuery()</code>
<code>OS_MUTEX_ACCEPT_EN</code>	<code>OSMutexAccept()</code>
<code>OS_MUTEX_QUERY_EN</code>	<code>OSMutexQuery()</code>
<code>OS_Q_ACCEPT_EN</code>	<code>OSQAccept()</code>
<code>OS_Q_POST_EN</code>	<code>OSQPost()</code>
<code>OS_Q_POST_FRONT_EN</code>	<code>OSQPostFront()</code>
<code>OS_Q_POST_OPT_EN</code>	<code>OSQPostOpt()</code>
<code>OS_Q_QUERY_EN</code>	<code>OSQQuery()</code>
<code>OS_SEM_ACCEPT_EN</code>	<code>OSSemAccept()</code>
<code>OS_SEM_QUERY_EN</code>	<code>OSSemQuery()</code>
<code>OS_TASK_QUERY_EN</code>	<code>OSTaskQuery()</code>
<code>OS_TIME_DLY_HMSM_EN</code>	<code>OSTimeDlyHMSM()</code>
<code>OS_TIME_DLY_RESUME_EN</code>	<code>OSTimeDlyResume()</code>
<code>OS_TIME_GET_SET_EN</code>	<code>OSTimeGet()</code> and <code>OSTimeSet()</code>
<code>OS_SCHED_LOCK_EN</code>	<code>OSSchedLock()</code> and <code>OSSchedUnlock()</code>

Added the `typedef OS_FLAGS` to allow you to specify the width of flags in an event flag group.

IMPORTANT

You **WILL** need to add **ALL** of the above `#define` in your `OS_CFG.H` files because `uCOS_II.H` contains error checks that will make your compiler complain if you don't include these `#defines`. The easiest way to accomplish this is to simply copy one of the `OS_CFG.H` files supplied in this release and paste it into your application and enable/disable the features you need.

OS_CORE.C:

Added call to `OS_FlagInit()` in `OSInit()` to support event flags.

Added call to `OSTaskIdleHook()` in `OS_TaskIdle()` to allow you to do such things as STOP the CPU to conserve power when running the idle task. You will need to add code in `OSTaskIdleHook()` to execute whatever is necessary for your CPU to enter its power down mode.

Added conditional compilation so that when `OS_SCHED_LOCK_EN` is set to 1 in `OS_CFG.H`, the code for `OSSchedLock()` and `OSSchedUnlock()` will be produced.

Corrected a bug in `OS_TCBInit()`. `OSTaskCreateHook()` was being `OSTCBPrioTbl[prio]` passed **INSTEAD** of `ptcb`. `OSTCBPrioTbl[prio]` didn't contain a valid pointer when `OSTaskCreateHook()` was being called.

WARNING

If you use the NEW port files in your product you will need to change **ALL** your Interrupt Service Routines (ISRs) to handle the new way the port works.

See **New Algorithm For Ports** at the end of the V2.51 notes.

OS_FLAG.C:

Added event flags to μ C/OS-II, see AN-1007.

OS_MBOX.C:

Added conditional compilation so that when `OS_MBOX_ACCEPT_EN` is set to 1 in `OS_CFG.H`, the code for `OSMboxAccept()` will be produced.

Added conditional compilation so that when `OS_MBOX_POST_EN` is set to 1 in `OS_CFG.H`, the code for `OSMboxPost()` will be produced. This allows you to reduce the amount of code space. The reason this conditional compilation has been added is because I added the more powerful function `OSMboxPostOpt()` which can emulate `OSMboxPost()` and also allows you to broadcast messages to all tasks waiting on the mailbox.

Added `OSMboxPostOpt()` which can emulate `OSMboxPost()` and also allows you to broadcast messages to all tasks waiting on the mailbox. The `#define` constant `OS_MBOX_POST_OPT_EN` found in `OS_CFG.H` allows you to enable (when 1) or disable (when 0) this feature.

Added conditional compilation so that when `OS_MBOX_QUERY_EN` is set to 1 in `OS_CFG.H`, the code for `OSMboxQuery()` will be produced. This allows you to reduce the amount of code space.

OS_MEM.C:

Added code to test the argument `addr` to make sure it's not a NULL pointer in `OSMemCreate()`.

Added code to test the argument `pmem` to make sure it's not a NULL pointer in `OSMemGet()`.

Added code to test the argument `pmem` and `pblk` to make sure they are not NULL pointers in `OSMemGet()`.

Added conditional compilation so that when `OS_MEM_QUERY_EN` is set to 1 in `OS_CFG.H`, the code for `OSMemQuery()` will be produced. This allows you to reduce the amount of code space.

Added code to test the argument `pmem` and `pdata` to make sure they are not NULL pointers in `OSMemQuery()`.

Added conditional compilation to allow you to declare storage for a single memory partition. In other words, you are now allowed to set `OS_MAX_MEM_PART` to 1 in `OS_CFG.H`.

OS_MUTEX.C:

Added conditional compilation so that when `OS_MUTEX_ACCEPT_EN` is set to 1 in `OS_CFG.H`, the code for `OSMutexAccept()` will be produced. This allows you to reduce the amount of code space.

Added conditional compilation so that when `OS_MUTEX_QUERY_EN` is set to 1 in `OS_CFG.H`, the code for `OSMutexQuery()` will be produced. This allows you to reduce the amount of code space.

Fixed a bug in `OSMutexDel()`. The entry in `OSTCBPrioTbl[]` was not being freed at the priority inheritance priority. This has been corrected.

Fixed a bug in `OSMutexPost()`. The current task priority was being tested for `&&` instead of `||`. This has been corrected.

OS_Q.C:

Added conditional compilation so that when `OS_Q_ACCEPT_EN` is set to 1 in `OS_CFG.H`, the code for `OSQAccept()` will be produced. This allows you to reduce the amount of code space.

Added conditional compilation so that when `OS_Q_FLUSH_EN` is set to 1 in `OS_CFG.H`, the code for `OSFlushAccept()` will be produced. This allows you to reduce the amount of code space.

Added conditional compilation so that when `OS_Q_POST_EN` is set to 1 in `OS_CFG.H`, the code for `OSQPost()` will be produced. This allows you to reduce the amount of code space. The reason this conditional compilation has been added is because I added the more powerful function `OSQPostOpt()` which can emulate both `OSQPost()` and `OSQPostFront()` also allows you to broadcast messages to all tasks waiting on the queue.

Added conditional compilation so that when `OS_Q_POST_FRONT_EN` is set to 1 in `OS_CFG.H`, the code for `OSQPostFront()` will be produced. This allows you to reduce the amount of code space. The reason this conditional compilation has been added is because I added the more powerful function `OSQPostOpt()`.

Added `OSQPostOpt()` which can emulate both `OSQPost()` and `OSQPostFront()` and also allows you to broadcast messages to all tasks

waiting on the queue. The `#define` constant `OS_Q_POST_OPT_EN` found in `OS_CFG.H` allows you to enable (when 1) or disable (when 0) this feature.

Added conditional compilation so that when `OS_Q_QUERY_EN` is set to 1 in `OS_CFG.H`, the code for `OSQQuery()` will be produced. This allows you to reduce the amount of code space.

Added conditional compilation to allow you to declare storage for a single message queue. In other words, you are now allowed to set `OS_MAX_QS` to 1 in `OS_CFG.H`.

OS_SEM.C:

Added conditional compilation so that when `OS_SEM_ACCEPT_EN` is set to 1 in `OS_CFG.H`, the code for `OSSemAccept()` will be produced.

Added conditional compilation so that when `OS_SEM_QUERY_EN` is set to 1 in `OS_CFG.H`, the code for `OSSemQuery()` will be produced. This allows you to reduce the amount of code space.

OS_TASK.C:

Added call to `OS_FlagUnlink()` in `OSTaskDel()` to support event flags. Note that this code is conditionally compiled in when `OS_FLAG_EN` is set to 1.

Added conditional compilation so that when `OS_TASK_QUERY_EN` is set to 1 in `OS_CFG.H`, the code for `OSTaskQuery()` will be produced. This allows you to reduce the amount of code space.

OS_TIME.C:

Added conditional compilation so that when `OS_TIME_DLY_HMSM_EN` is set to 1 in `OS_CFG.H`, the code for `OSTimeDlyHMSM()` will be produced. This allows you to reduce the amount of code space in case you chose not to use this function.

Added conditional compilation so that when `OS_TIME_DLY_RESUME_EN` is set to 1 in `OS_CFG.H`, the code for `OSTimeDlyResume()` will be produced. This allows you to reduce the amount of code space in case you chose not to use this function.

Added conditional compilation so that when `OS_TIME_GET_SET_EN` is set to 1 in `OS_CFG.H`, the code for `OSTimeGet()` and `OSTimeSet()` will be produced. This allows you to reduce the amount of code space in case you chose not to use this function.

uCOS_II.C:

Added OS_FLAG.C.

uCOS_II.H:

Changed OS_VERSION to 205.

Added constants, data types and function prototypes to support Event Flags.

Added OS_POST_OPT_??? which are the options to specify in OSMboxPostOpt() and OSQPostOpt() calls.

The global variable OSTime is not allocated when OS_TIME_GET_SET_EN is set to 0. This reduces the RAM footprint by 4 bytes.

Added checks at the end of uCOS_II.H to ensure that you don't forget any #defines that are assumed to be declared in OS_CFG.H. If you do forget any of the required #defines in OS_CFG.H, the compiler will issue an error message. In other words, your compiler should complain about the fact that you didn't specify all the necessary #defines.

New Algorithm For Ports:

V2.51 comes with a new algorithm which prevents from having to adjust the stack pointer in `OSIntCtxSw()` and thus making the port independent of compilers and compiler options.

You should still be able to use your OLD (V2.04 and earlier) ports without change (except you'll need to add a few HOOK functions as described in the next section.

This new algorithm affects ALL your ISRs and thus you MUST play close attention to the following changes.

The OLD pseudo code for `OSIntCtxSw()` was:

```
OSIntCtxSw() :                               /* OLD */
    Adjust the SP to remove call to OSIntExit(),
    locals in OSIntExit() and the call to OSIntCtxSw();
    Save the stack pointer to OSTCBCur->OSTCBStkPtr;
    Call OSTaskSwHook()
    OSTCBCur          = OSTCBHighRdy;
    OSPrioCur        = OSPrioHighRdy;
    CPU Stack Pointer = OSTCBHighRdy->OSTCBStkPtr;
    POP all the CPU registers from the new task's stack;
    Execute a return from interrupt instruction;
```

The NEW pseudo code for `OSIntCtxSw()` is now:

```
OSIntCtxSw() :                               /* NEW */
    Call OSTaskSwHook()
    OSTCBCur          = OSTCBHighRdy;
    OSPrioCur        = OSPrioHighRdy;
    CPU Stack Pointer = OSTCBHighRdy->OSTCBStkPtr;
    POP all the CPU registers from the new task's stack;
    Execute a return from interrupt instruction;
```

You should notice that you NO LONGER need to adjust the SP. The reason this is possible is because, the SP of the task that can be switched out now NEEDS to be saved in ALL the ISRs as described below.

You MUST now change ALL your ISRs. The OLD pseudo code for your ISRs was:

```
YourISR():                               /* OLD */
    Save processor registers onto current task's stack;
    Call OSIntEnter() or increment OSIntNesting;
    .
    YOUR ISR Handler code;
    .
    Call OSIntExit();
    Restore processor registers from current task's stack;
    Execute a return from interrupt instruction;
```

The NEW pseudo code for OSIntCtxSw() is now:

```
YourISR():                               /* NEW */
    Save processor registers onto current task's stack;
    Call OSIntEnter() or increment OSIntNesting;
    if (OSIntNesting == 1) {
        Save the CPU's Stack Pointer onto current task's stack;
    }
    .
    YOUR ISR Handler code;
    .
    Call OSIntExit();
    Restore processor registers from current task's stack;
    Execute a return from interrupt instruction;
```

Upgrading from V2.04 (or earlier) to V2.51:

You should be able to use processor ports made for V2.04 or earlier. Because I added new features, you will most likely need to change the following files:

1) `OS_CFG.H`:

You will need to ADD all the new `#define` constants and also, declare the data type `OS_FLAGS`. As I mentioned previously, you can simply copy one of the `OS_CFG.H` files supplied with this release and paste it into your own and make the appropriate selection of features you need in your product.

2) `OS_CPU_C.C`:

You will need to ADD an empty function for `OSTaskIdleHook()` as follows unless you actually want to add your own code to the function:

```
void OSTaskIdleHook (void)
{
}
```

3) `OS_CPU_A.ASM`:

If you want to use the new ALGORITHM described in the previous section, you will need to change `OSIntCtxSw()`, `OSTickISR()` **AND** all your ISRs. You should be able to use your OLD ports without change if you don't want to use the new algorithm.

4) `OS_CPU.H`:

No change.

5) Your ISRs:

If you want to use the new ALGORITHM described in the previous section, you will need to change ALL your ISRs. You should be able to use your OLD ports without change if you don't want to use the new algorithm.

V2.04

(2000/10/31)

MISCELLANEOUS :

Removed revision history from all the source code. The revision history is now described in this document. This was done to reduce the amount of 'clutter' from the source files.

Added `OS_ARG_CHK_EN` to enable (when 1) MicroC/OS-II argument checking. By setting this configuration constant to 0, you would be able to reduce code size and improve on performance by not checking the range of the arguments passed to MicroC/OS-II functions. However, it is recommended to leave argument checking enabled.

Added Mutual Exclusion Semaphores (`OS_MUTEX.C`) that are described in `AN1002.PDF`.

Added support for `OS_CRITICAL_METHOD #3` that allows the status register of the CPU to be saved in a local variable. The status register is assumed to be saved by `OS_ENTER_CRITICAL()` in a local variable called `cpu_sr` of type `OS_CPU_SR`. The data type `OS_CPU_SR` is assumed to be declared in `OS_CPU.H`. The status register (and thus the state of the interrupt disable flag) is assumed to be restored by `OS_EXIT_CRITICAL()` from the contents of this variable. The macros would be declared as follows:

```
#define OS_ENTER_CRITICAL()  (cpu_sr = OSCPU_SaveSR())
#define OS_EXIT_CRITICAL()   (OSCPURestoreSR(cpu_sr))
```

Note that the functions `OSCPUSaveSR()` and `OSCPURestoreSR()` would be written in assembly language and would typically be found in `OS_CPU_A.ASM` (or equivalent).

The check for `OSIntNesting` in all μ C/OS-II services is now being done without disabling interrupts in order to reduce interrupt latency. In other words, the following code:

```
OS_ENTER_CRITICAL();
if (OSIntNesting > 0) {
    .
    .
    OS_EXIT_CRITICAL();
}
```

Has been replaced by:

```

    if (OSIntNesting > 0) {
        .
        .
    }

```

The reason is that ALL currently known processors will treat this byte size variable (OSIntNesting) indivisibly.

OS_CORE.C:

Moved all local variables to `uCOS_II.H` making them all global variables. This helps when testing.

Calls to `OSTaskCreate()` and `OSTaskCreateExt()` in `OSInit()` now return `(void)` to indicate that the return value is not being used. This prevents warnings from LINT.

Although not critical, `OSInit()` was optimized for speed.

Added `OSInitHookBegin()` at the beginning of `OSInit()` to allow for a processor port to provide additional 'OS' specific initialization which would be done BEFORE MicroC/OS-II is initialized.

Added `OSInitHookEnd()` at the end of `OSInit()` to allow for a processor port to provide additional 'OS' specific initialization which would be done AFTER MicroC/OS-II is initialized.

Initialized `.OSEventType` to `OS_EVENT_TYPE_UNUSED` in `OSInit()`.

Added boundary check for `OSIntNesting` in `OSIntEnter()` to prevent wrapping back to 0 if `OSIntNesting` is already at 255.

Added boundary check on `OSIntNesting` in `OSIntExit()` to prevent wrapping back to 255 if `OSIntNesting` is already at 0.

Changed the test for rescheduling in `OSIntExit()` and `OSSched()` from:

```

if ((--OSIntNesting | OSLockNesting) == 0) {

```

to

```

if ((OSIntNesting == 0) && (OSLockNesting == 0)) {

```

for sake of clarity.

Removed unreachable code in OSTaskStat() for CPU usage > 100%.

Added call to OSTCBInitHook() in OSTCBInit() to allow user (or port) specific TCB extension initialization.

Moved the increment of OSTimeTick() immediately after calling OSTimeTickHook().

Made OSTime volatile.

OS_MBOX.C:

Removed checking of pevent from the critical section to reduce interrupt latency.

Removed checking of msg from the critical section to reduce interrupt latency.

Added OSMBxDel() to delete a message mailbox and free up its Event Control Block. All tasks pending on the mailbox will be readied. This feature is enabled by setting OS_MBOX_DEL_EN to 1.

Changed test:

```
    if (pevent->OSEventGrp)
to
    if (pevent->OSEventGrp != 0x00).
```

OS_MEM.C:

Moved the local variables OSMemFreeList and OSMemTbl[] to uCOS_II.H.

Added code to initialize all the fields of the last node in OSMemInit().

OS_MUTEX.C:

Added services to support Mutual Exclusion Semaphores that are used to reduce priority inversions.

OS_Q.C:

Removed checking of pevent from the critical section to reduce interrupt latency.

Removed checking of `msg` from the critical section to reduce interrupt latency.

Added `OSQDel()` to delete a message queue and free up its Event Control Block. All tasks pending on the queue will be readied. This feature is enabled by setting `OS_Q_DEL_EN` to 1.

Changed test:

```
    if (pevent->OSEventGrp)
to
    if (pevent->OSEventGrp != 0x00).
```

Moved the definition of the data type `OS_Q` to `uCOS_II.H`.

OS_SEM.C:

Removed checking of `pevent` from the critical section to reduce interrupt latency.

Added `OSSemDel()` to delete a semaphore and free up its Event Control Block. All tasks pending on the semaphore will be readied. This feature is enabled by setting `OS_SEM_DEL_EN` to 1.

Changed test:

```
    if (pevent->OSEventGrp)
to
    if (pevent->OSEventGrp != 0x00).
```

OS_TASK.C:

Task stack is now cleared in `OSTaskCreateExt()` when either options `OS_TASK_OPT_STK_CHK` or `OS_TASK_OPT_STK_CLR` is set. The new code is:

```
    if (((opt & OS_TASK_OPT_STK_CHK) != 0x0000) ||
        ((opt & OS_TASK_OPT_STK_CLR) != 0x0000)) {
```

`OSTaskCreateHook()` has been removed from `OSTaskCreate()` and `OSTaskCreateExt()` and moved to `OSTCBInit()` so that the hook is called BEFORE the task is made ready-to-run. This avoids having the possibility of readying the task before calling the hook function.

If you don't specify any Mailboxes (`OS_MBOX == 0`), Queues (`OS_Q == 0`), Semaphores (`OS_SEM == 0`) or Mutexes (`OS_MUTEX == 0`) in `OS_CFG.H` in order to create a minimal system, `OSTaskChangePrio()` and `OSTaskDel()` will no longer reference `OSTCBEventPtr`.

OS_TIME.C:

Added cast to INT16U for all references of `tick` in `OSTimeDlyHMSM()`.

uCOS_II.C:

Added `OS_MUTEX.C`.

uCOS_II.H:

Changed `OS_VERSION` to 204.

Moved all 'local' variables from `OS_MEM.C`, `OS_Q.C` and `OS_TASKS.C` to simplify debugging and unit testing.

Added constants, data types and function prototypes to support Mutual Exclusion Semaphores.

This page is intentionally blank.

V2.03

(1999/09/09)

MISCELLANEOUS :

The distribution of μ C/OS-II now assumes the Borland C/C++ V4.51 or higher compiler instead of the V3.1 compiler. The code should, however, compile and run using V3.1.

This release contains a slightly different directory structure. The name of the compiler is added to the directory structure in order to support multiple compilers and have the same directory structure for all of these.

`\SOFTWARE\uCOS-II\SOURCE`

Contains the source files for the processor independent code of μ C/OS-II.

`\SOFTWARE\uCOS-II\I \times 86L\BC45`

Contains the source files for the 80x86 real mode, large model port. The port now contains the function `OSTaskStkInit_FPE_x86()` which needs to be called before you create a task that will use Borland C/C++'s floating-point emulation (FPE) library. See application note AN-1001 found on www.Micrium.com.

`\SOFTWARE\uCOS-II\I \times 86L-FP\BC45`

Contains the source files for the 80x86 real mode, large model port. This port also contains hardware floating-point support. In other words, μ C/OS-II performs a context switch on the floating-point registers as well as the integer registers. This port was not present on the original distribution of μ C/OS-II (i.e. V2.00).

`\SOFTWARE\uCOS-II\EX1_x86L\BC45\SOURCE`

Contains the source code for the sample code of Example #1

`\SOFTWARE\uCOS-II\EX1_x86L\BC45\TEST`

Contains the build files (`MAKETEST.BAT` and `TEST.MAK`) as well as the executable for Example #1. To build the executable for example #1, simply type `MAKETEST` at the DOS prompt. You may have to change `TEST.MAK` to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the `E:\BC45` directory. To execute example #1, type `TEST` at the DOS prompt.

`\SOFTWARE\uCOS-II\EX2_x86L\BC45\SOURCE`

Contains the source code for the sample code of Example #2

`\SOFTWARE\uCOS-II\EX2_x86L\BC45\TEST`

Contains the build files (MAKETEST.BAT and TEST.MAK) as well as the executable for Example #2. To build the executable for example #2, simply type MAKETEST at the DOS prompt. You may have to change TEST.MAK to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the E:\BC45 directory. To execute example #2, type TEST at the DOS prompt.

\SOFTWARE\uCOS-II\EX3_x86L\BC45\SOURCE

Contains the source code for the sample code of Example #3

\SOFTWARE\uCOS-II\EX3_x86L\BC45\TEST

Contains the build files (MAKETEST.BAT and TEST.MAK) as well as the executable for Example #3. To build the executable for example #3, simply type MAKETEST at the DOS prompt. You may have to change TEST.MAK to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the E:\BC45 directory.

To execute example #3, type TEST at the DOS prompt.

\SOFTWARE\uCOS-II\EX4_x86L.FP\BC45\SOURCE

Contains the source code for the sample code of Example #4

\SOFTWARE\uCOS-II\EX4_x86L\BC45\TEST

Contains the build files (MAKETEST.BAT and TEST.MAK) as well as the executable for Example #4. Example #4 demonstrate the use of Ix86L-FP, the port that saves/restores the 80x86's floating-point registers during a context switch. This of course applies for 80x86 processors having a floating-point unit. You may have to change TEST.MAK to tell it where the Borland C/C++ V4.51 compiler is located. My compiler was located in the E:\BC45 directory. To execute example #1, type TEST at the DOS prompt.

\SOFTWARE\BLOCKS\PC\BC45

Contains the source files for the PC services used to display characters on the screen, read the keyboard etc.

EXAMPLES :

Example #1 (V2.00)

TEST.C was previously called EX1L.C

PC_DisPClrLine() has been changed to PC_DisPClrRow().

TaskClk() now calls PC_GetDateTime().

The floating-point code in TaskStart() has been removed so that the task only executes integer arithmetic instructions.

Example #2 (V2.00)

TEST.C was previously called EX2L.C

Added TaskStartCreateTasks() to create all the application tasks.
TaskStart() now uses the Borland C/C++ Floating-Point Emulation library and thus, the stack needs to be 'preconditioned' by calling the function OSTaskStkInit_FPE_x86() (see www.Micrium.com, AN-1001).

PC_DisPClrLine() has been changed to PC_DisPClrRow().

TaskClk() now calls PC_GetDateTime().

Example #3 (V2.00)

TEST.C was previously called EX3L.C

Added TaskStartCreateTasks() to create all the application tasks.

PC_DisPClrLine() has been changed to PC_DisPClrRow().

TaskClk() now calls PC_GetDateTime().

Floating-point operations have been replaced with integer operations.

Example #4 (V2.00)

Example #4 is a new example using hardware assisted floating-point.

TEST.C was previously called EX4L.C

PC_DisPClrLine() has been changed to PC_DisPClrRow().

TaskClk() now calls PC_GetDateTime().

PC Services (V2.00)

PC.C:

Functions are now listed in alphabetical order in the file.

PC_ElapsedStart() and PC_ElapsedStop() now protect the critical section of code that accesses the timer ports.

PC_VectGet() and PC_VectSet() no longer depend on the Borland C/C++ functions getvect() and setvect(). This should make these functions more portable.

Changed the name of PC_DispClrLine() to PC_DispClrRow().

Added function PC_DispClrCol().

The following function now cast MK_FP() to (INT8U far *):

```
PC_DispChar()  
PC_DispClrLine()  
PC_DispClrScr()  
PC_DispStr()
```

PC_ElapsedStop(), cast inp() to INT8U.

PC_GetKey(), cast getch() to INT16S.

PC.H:

Function prototypes are now listed in alphabetical order.

Added prototype for PC_DispClrCol().

OS_CORE.C:

Changed the return type of `OSEventTaskRdy()` from `void` to `INT8U` to return the priority of the task readied even though the current version of MicroC/OS-II doesn't make use of this feature. This change was done to support future versions.

Moved `OSDummy()` from `OS_TASK.C` to `OS_CORE.C` to be able to call `OSDummy()` from other services.

OS_MBOX.C:

Added check in `OSMboxPost()` to see if the caller is attempting to post a `NULL` pointer. By definition, you should NOT send a `NULL` pointer message. If you attempt to post a `NULL` pointer, `OSMboxPost()` will return `OS_ERR_POST_NULL_PTR`.

Added checks to make sure `pevent` is not a `NULL` pointer. If `pevent` is a `NULL` pointer, each of the following functions will return `OS_ERR_EVENT_NULL`:

`OSMboxPost()`
`OSMboxQuery()`

Note that `OSMboxAccept()` will return a `NULL` pointer because it doesn't provide the capability of returning an error code.

`OSMboxPend()` sets `*err` to `OS_ERR_EVENT_NULL` if `pevent` is a `NULL` pointer.

OS_Q.C:

Added check in `OSQPost()` and `OSQPostFront()` to see if the caller is attempting to post a `NULL` pointer. By definition, you should NOT send a `NULL` pointer message. If you attempt to post a `NULL` pointer, `OSQPost()` and `OSQPostFront()` will return `OS_ERR_POST_NULL_PTR`.

Added checks to make sure `pevent` is not a NULL pointer. If `pevent` is a NULL pointer, each of the following functions will return `OS_ERR_PEVENT_NULL`:

```
OSQFlush()  
OSQPost()  
OSQPostFront()  
OSQQuery()
```

Note that `OSQAccept()` simply returns a NULL pointer because it doesn't provide the capability of returning an error code.

`OSQPend()` sets `*err` to `OS_ERR_PEVENT_NULL` if `pevent` is a NULL pointer.

OS_SEM.C:

Added checks to make sure `pevent` is not a NULL pointer. If `pevent` is a NULL pointer, each of the following functions will return `OS_ERR_PEVENT_NULL`:

```
OSSemPost()  
OSSemQuery()
```

Note that `OSSemAccept()` returns 0 because it doesn't provide the capability to return an error code.

`OSSemPend()` sets `*err` to `OS_ERR_PEVENT_NULL` if `pevent` is a NULL pointer.

OS_TASK.C:

Moved `OSDummy()` to `OS_CORE.C`

uCOS_II.H:

Added error code `OS_ERR_POST_NULL_PTR` (value is 3).

Changed the return type of `OSEventTaskRdy()` from `void` to `INT8U` to return the priority of the task readied.

Added function prototype for `OSDummy()` .

Added error code `OS_ERR_PEVENT_NULL` (value is 4)

V2.02

(1999/07/18)

OS_MBOX.C:

Removed last `else` statement in `OSMboxPend()` because the code is unreachable.

OS_Q.C:

Removed last `else` statement in `OSQPend()` because the code is unreachable.

OS_TASK.C:

`OSTaskCtr` is always included.

uCOS_II.C:

Added check for definition of macro `OS_ISR_PROTO_EXT` so that the prototype of `OSCtxSw()` and `OSTickISR()` can be changed based on compiler specific requirements. To use a different prototype, simply add:

```
#define OS_ISR_PROTO_EXT
```

in `OS_CPU.H` of the port and then define the new prototype format for `OSCtxSw()` and `OSTickISR()` in `OS_CPU.H` of the port.

`OSTaskCtr` is always included. Previously it was conditionally compiled only if `OS_TASK_CREATE_EN`, `OS_TASK_CREATE_EXT_EN` or `OS_TASK_DEL_EN` was set to 1. It turns out that you **MUST** always have either `OS_TASK_CREATE_EN` or `OS_TASK_CREATE_EXT_EN` set to 1 anyway!

This page is intentionally blank.

V2.01

(1999/07/15)

OS_CORE.C:

Changed for loop inside `OSEventWaitListInit()` to inline code for speed. This eliminates the loop overhead.

The argument `stk_size` in `OSTCBInit()` has been changed from `INT16U` to `INT32U` to accommodate large stacks.

OS_MBOX.C:

Changed 'for' loop inside '`OSMboxQuery()`' to inline code for speed. This eliminates the loop overhead.

OS_Q.C:

Added typecast to avoid compiler error/warning:

```
pq = (OS_Q *)pevent->OSEventPtr;  
      ^^^^^^^^
```

Affected functions:

```
OSQAccept()  
OSQFlush()  
OSQPend()  
OSQPost()  
OSQPostFront()
```

Changed for loop inside `OSQQuery()` to inline code for speed. This eliminates the loop overhead.

Added `msg = (void *)0;` in if (`OSIntNesting > 0`) case.

OS_SEM.C:

Second if statement in function `OSSemPend()` needed to be and if/else clause.

OS_TASK.C:

Stack filling is now done using the ANSI C function `memset()` for speed.

Copying of the `OS_TCB` structure in `OSTaskQuery()` is now done using `memcpy()` for speed.

Function `OSTaskStkChk()` now cast the value 0 to `(OS_STK)0` in while loops.

uCOS_II.C:

Changed the comment for `OSTCBStkSize` in the `OS_TCB` structure to indicate that the size is in number of elements and not bytes.

The argument `stk_size` in `OSTCBInit()` has been changed from `INT16U` to `INT32U` to accommodate large stacks.