µC/OS-II Configuration Manual

This chapter provides a description of the configurable elements of µC/OS-II. Because µC/OS-II is provided in source form, configuration is done through a number of `#define` constants, which are found in `OS_CFG.H` and should exist for each project/product that you develop. In other words, configuration is done via conditional compilation.

Instead of creating an `OS_CFG.H` file from scratch, it is recommended that you copy and modify one of the `OS_CFG.H` files provided in one of the examples that came with µC/OS-II. `OS_CFG.H` is independent of the type of CPU used.

This section describes each of the `#define` constants in `OS_CFG.H`.

17.00 Miscellaneous

OS_APP_HOOKS_EN

When set to `1`, this `#define` specifies that application defined hooks are called from µC/OS-II's hooks. See also `OS_CPU_HOOKS_EN`. Specifically:

The µC/OS-II hook …	Calls the Application-define hook …
`OSTaskCreateHook()`	`App_TaskCreateHook()`
`OSTaskDelHook()`	`App_TaskDelHook()`
`OSTaskIdleHook()`	`App_TaskIdleHook()`
`OSTaskStatHook()`	`App_TaskStatHook()`
`OSTaskSwHook()`	`App_TaskSwHook()`
`OSTCBInitHook()`	`App_TCBInitHook()`
`OSTimeTickHook()`	`App_TimeTickHook()`

OS_ARG_CHK_EN

OS_ARG_CHK_EN indicates whether you want most of µC/OS-II functions to perform argument checking. When set to 1, µC/OS-II will ensure that pointers passed to functions are non-NULL, that arguments passed are within allowable range and more. OS_ARG_CHK_EN was added to reduce the amount of code space and processing time required by µC/OS-II. Set OS_ARG_CHK_EN to 0 if you must reduce code space to a minimum. In general, you should always enable argument checking and thus set OS_ARG_CHK_EN to 1.

OS_CPU_HOOKS_EN

OS_CPU_HOOKS_EN indicates whether OS_CPU_C.C declares the hook function (when set to 1) or not (when set to 0). Recall that µC/OS-II expects the presence of nine functions that can be defined either in the port (i.e., in OS_CPU_C.C) or by the application code. These functions are:

```
OSInitHookBegin()
OSInitHookEnd()
OSTaskCreateHook()
OSTaskDelHook()
OSTaskIdleHook()
OSTaskStatHook()
OSTaskSwHook()
OSTCBInitHook()
OSTimeTickHook()
```

OS_DEBUG_EN

When set to 1, this #define adds ROM constants located in OS_DEBUG.C 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, µC/OS-II's version number, the size of an event flag group (OS_FLAG_GRP) and much more.

OS_EVENT_MULTI_EN

This constant determines whether the code to support pending on multiple events will be enabled (1) or not (0). This constant thus enables code for the function OSEventPendMulti(). This #define was added in V2.86.

OS_EVENT_NAME_EN

This constant determines whether names can be assigned to either a semaphore, a mutex, a mailbox or a message queue. If OS_EVENT_NAME_EN is set to 0, this feature is disabled. You should note that need to use OSEventNameSet() to set the name of either a semaphores, a mutex, a mailbox or a message queue. You need to use OSEventNameGet() to obtain the name of either a semaphores, a mutex, a mailbox or a message queue.

OS_LOWEST_PRIO

OS_LOWEST_PRIO specifies the lowest task priority (i.e., highest number) that you intend to use in your application and is provided to reduce the amount of RAM needed by µC/OS-II. As of V2.80 µC/OS-II priorities can go from 0 (highest priority) to a maximum of 254 (lowest possible priority). Setting OS_LOWEST_PRIO to a value less than 254 means that your application cannot create tasks with a priority number higher than OS_LOWEST_PRIO. In fact, µC/OS-II reserves priorities OS_LOWEST_PRIO and OS_LOWEST_PRIO–1 for itself; OS_LOWEST_PRIO is reserved for the idle task, OS_TaskIdle(), and OS_LOWEST_PRIO–1 is reserved for the statistic task, OS_TaskStat(). The priorities of your application tasks can thus take a value between 0 and OS_LOWEST_PRIO–2 (inclusive). The lowest task priority specified by OS_LOWEST_PRIO is independent of OS_MAX_TASKS. For example, you can set OS_MAX_TASKS to 10 and OS_LOWEST_PRIO to 32 and have up to 10 application tasks, each of which can have a task priority value between 0 and 30 (inclusive). Note that each task must still have a different priority value. You must always set OS_LOWEST_PRIO to a value greater than the number of application tasks in your system. For example, if you set OS_MAX_TASKS to 20 and OS_LOWEST_PRIO to 10, you can not create more than eight application tasks (0 to 7) since priority 8 is the statistics task and priority 9 is the idle task. You are simply wasting RAM.

OS_MAX_EVENTS

OS_MAX_EVENTS specifies the maximum number of event control blocks that can be allocated. An event control block is needed for every message mailbox, message queue, mutual exclusion semaphore, or semaphore object. For example, if you have 10 mailboxes, five queues, four mutexes, and three semaphores, you must set OS_MAX_EVENTS to at least 22. OS_MAX_EVENTS must be greater than 0. See also OS_MBOX_EN, OS_Q_EN, OS_MUTEX_EN, and OS_SEM_EN.

OS_MAX_FLAGS

OS_MAX_FLAGS specifies the maximum number of event flags that you need in your application. OS_MAX_FLAGS must be greater than 0. To use event-flag services, you also need to set OS_FLAG_EN to 1.

OS_MAX_MEM_PART

OS_MAX_MEM_PART specifies the maximum number of memory partitions that your application can create. To use memory partitions, also need to set OS_MEM_EN to 1. If you intend to use memory partitions, OS_MAX_MEM_PART must be set to at least the number of partitions you wish to create. For example, by setting OS_MAX_MEM_PART to 3, your are allowed to create and use up to three memory partitions. Setting OS_MAX_MEM_PART to a number greater than the number of memory partitions your application uses will not cause problems but is unnecessary and a waste of RAM.

OS_MAX_QS

OS_MAX_QS specifies the maximum number of message queues that your application can create. To use message queues, you also must set OS_Q_EN to 1. If you intend to use message queues, OS_MAX_QS must be set to at least the number of queues you wish to create. For example, if you set OS_MAX_QS to 3, you are allowed to create and use up to three message queues. Setting OS_MAX_QS to greater than the number of message queues your application uses will not cause problems but is unnecessary and a waste of RAM.

OS_MAX_TASKS

OS_MAX_TASKS specifies the maximum number of *application* tasks that can exist in your application. Note that OS_MAX_TASKS cannot be greater than 253 (as of V2.80) because µC/OS-II currently reserves two tasks for itself (see OS_N_SYS_TASKS in uCOS_II.H). If you set OS_MAX_TASKS to the exact number of tasks in your system, you need to make sure that you revise this value when you add additional tasks. Conversely, if you make OS_MAX_TASKS much higher than your current task requirements (for future expansion), you are wasting valuable RAM.

OS_SCHED_LOCK_EN

This constant enables (when set to 1) or disables (when set to 0) code generation for the two functions OSSchedLock() and OSSchedUnlock().

OS_TICK_STEP_EN

µC/OS-View (a Micrium product that allows you to display run-time data about your tasks on a Windows-based PC) can now 'halt' µC/OS-II's tick processing and allow you to issue 'step' commands from µC/OS-View. In other words, µC/OS-View can prevent µC/OS-II from calling OSTimeTick() so that timeouts and time delays are no longer processed. However, though a keystroke from µC/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_TICKS_PER_SEC

OS_TICKS_PER_SEC specifies the rate at which you call OSTimeTick(). It is up to your initialization code to ensure that OSTimeTick() is invoked at this rate. This constant is used by OSStatInit(), OS_TaskStat(), and OSTimeDlyHMSM().

17.01 Event Flags

OS_FLAG_EN

`OS_FLAG_EN` enables (when set to 1) or disables (when set to 0) code generation of **all** the event-flag services and data structures, which reduces the amount of code and data space needed when your application does not require the use of event flags. When `OS_FLAG_EN` is set to 0, you do not need to enable or disable any of the other `#define` constants in this section.

OS_FLAG_ACCEPT_EN

`OS_FLAG_ACCEPT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSFlagAccept()`.

OS_FLAG_DEL_EN

`OS_FLAG_DEL_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSFlagDel()`.

OS_FLAG_NAME_EN

This constant determines whether names can be assigned to event flag groups. If `OS_FLAG_NAME_EN` is set to `0`, this feature is disabled.

OS_FLAG_QUERY_EN

`OS_FLAG_QUERY_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSFlagQuery()`.

OS_FLAG_WAIT_CLR_EN

`OS_FLAG_WAIT_CLR_EN` enables (when set to 1) or disables (when set to 0) the code generation used to wait for event flags to be 0 instead of 1. Generally, you want to wait for event flags to be set. However, you might also want to wait for event flags to be clear, and thus you need to enable this option.

OS_FLAGS_NBITS

`OS_FLAGS_NBITS` has been introduced in V2.80 and specifies the number of bits used in event flags and MUST be either 8, 16 or 32.

17.02 Message Mailboxes

OS_MBOX_EN

This constant enables (when set to 1) or disables (when set to 0) the code generation of **all** message-mailbox services and data structures, which reduces the amount of code space needed when your application does not require the use of message mailboxes. When `OS_MBOX_EN` is set to 0, you do not need to enable or disable any of the other `#define` constants in this section.

OS_MBOX_ACCEPT_EN

This constant enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMboxAccept()`.

OS_MBOX_DEL_EN

This constant enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMboxDel()`.

OS_MBOX_PEND_ABORT_EN

`OS_MBOX_PEND_ABORT_EN` enables (when set to `1`) or disables (when set to `0`) the code generation of the function `OSMboxPendAbort()`.

OS_MBOX_POST_EN

`OS_MBOX_POST_EN` enables (when set to `1`) or disables (when set to `0`) the code generation of the function `OSMboxPost()`. You can disable code generation for this function if you decide to use the more powerful function `OSMboxPostOpt()` instead.

OS_MBOX_POST_OPT_EN

`OS_MBOX_POST_OPT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMboxPostOpt()`. You can disable code generation for this function if you do not need the additional functionality provided by `OSMboxPostOpt()`. `OSMboxPost()` generates less code.

OS_MBOX_QUERY_EN

`OS_MBOX_QUERY_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMboxQuery()`.

17.03 Memory Management

OS_MEM_EN

`OS_MEM_EN` enables (when set to 1) or disables (when set to 0) **all** code generation of the µC/OS-II partition-memory manager and its associated data structures. This feature reduces the amount of code and data space needed when your application does not require the use of memory partitions.

OS_MEM_NAME_EN

This constant determines whether names can be assigned to memory partitions. If `OS_MEM_NAME_EN` is set to `0`, this feature is disabled and no RAM is used in the `OS_MEM` for the memory partition for storage of names.

OS_MEM_QUERY_EN

`OS_MEM_QUERY_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMemQuery()`.

17.04 Mutual Exclusion Semaphores

OS_MUTEX_EN

`OS_MUTEX_EN` enables (when set to 1) or disables (when set to 0) the code generation of **all** mutual-exclusion-semaphore services and data structures, which reduces the amount of code and data space needed when your application does not require the use of mutexes. When `OS_MUTEX_EN` is set to 0, you do not need to enable or disable any of the other `#define` constants in this section.

OS_MUTEX_ACCEPT_EN

`OS_MUTEX_ACCEPT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMutexAccept()`.

OS_MUTEX_DEL_EN

`OS_MUTEX_DEL_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMutexDel()`.

OS_MUTEX_QUERY_EN

`OS_MUTEX_QUERY_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSMutexQuery()`.

17.05 Message Queues

OS_Q_EN

`OS_Q_EN` enables (when set to 1) or disables (when set to 0) the code generation of **all** message-queue services and data structures, which reduces the amount of code space needed when your application does not require the use of message queues. When `OS_Q_EN` is set to 0, you do not need to enable or disable any of the other `#define` constants in this section. Note that if `OS_Q_EN` is set to 0, the `#define` constant `OS_MAX_QS` is irrelevant.

OS_Q_ACCEPT_EN

`OS_Q_ACCEPT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQAccept()`.

OS_Q_DEL_EN

`OS_Q_DEL_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQDel()`.

OS_Q_FLUSH_EN

`OS_Q_FLUSH_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQFlush()`.

OS_Q_PEND_ABORT_EN

`OS_Q_PEND_ABORT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQPendAbort()`.

OS_Q_POST_EN

`OS_Q_POST_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQPost()`. You can disable code generation for this function if you decide to use the more powerful function `OSQPostOpt()` instead.

OS_Q_POST_FRONT_EN

`OS_Q_POST_FRONT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQPostFront()`. You can disable code generation for this function if you decide to use the more powerful function `OSQPostOpt()` instead.

OS_Q_POST_OPT_EN

`OS_Q_POST_OPT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQPostOpt()`. You can disable code generation for this function if you do not need the additional functionality provided by `OSQPostOpt()`. `OSQPost()` generates less code.

OS_Q_QUERY_EN

`OS_Q_QUERY_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSQQuery()`.

17.06 Semaphores

OS_SEM_EN

`OS_SEM_EN` enables (when set to 1) or disables (when set to 0) **all** code generation of the µC/OS-II semaphore manager and its associated data structures, which reduces the amount of code and data space needed when your application does not require the use of semaphores. When `OS_SEM_EN` is set to 0, you do not need to enable or disable any of the other `#define` constants in this section.

OS_SEM_ACCEPT_EN

`OS_SEM_ACCEPT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSSemAccept()`.

OS_SEM_DEL_EN

`OS_SEM_DEL_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSSemDel()`.

OS_SEM_PEND_ABORT_EN

`OS_SEM_PEND_ABORT_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSSemPendAbort()`.

OS_SEM_QUERY_EN

`OS_SEM_QUERY_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSSemQuery()`.

OS_SEM_SET_EN

`OS_SEM_SET_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSSemSet()`.

17.07 Task Management

OS_TASK_TMR_STK_SIZE

`OS_TASK_TMR_STK_SIZE` specifies the size of the µC/OS-II timer task stack. The size is specified not in bytes but in number of elements. This is because a stack is declared to be of type `OS_STK`. The size of the timer-task stack depends on the processor you are using, the 'callback' functions that will be executed when each of the timer times out and the deepest anticipated interrupt-nesting level.

OS_TASK_STAT_STK_SIZE

`OS_TASK_STAT_STK_SIZE` specifies the size of the µC/OS-II statistic-task stack. The size is specified not in bytes but in number of elements. This is because a stack is declared as being of type `OS_STK`. The size of the statistic-task stack depends on the processor you are using and the maximum of the following actions:

- The stack growth associated with performing 32-bit arithmetic (subtraction and division)
- The stack growth associated with calling `OSTimeDly()`
- The stack growth associated with calling `OSTaskStatHook()`
- The deepest anticipated interrupt-nesting level

If you want to run stack checking on this task and determine its actual stack requirements, you must enable code generation for `OSTaskCreateExt()` by setting `OS_TASK_CREATE_EXT_EN` to 1. Again, the priority of `OS_TaskStat()` is always set to `OS_LOWEST_PRIO-1`.

OS_TASK_IDLE_STK_SIZE

`OS_TASK_IDLE_STK_SIZE` specifies the size of the µC/OS-II idle-task stack. The size is specified not in bytes but in number of elements. This is because a stack is declared to be of type `OS_STK`. The size of the idle-task stack depends on the processor you are using and the deepest anticipated interrupt-nesting level. Very little is being done in the idle task, but you should allow at least enough space to store all processor registers on the stack and enough storage to handle all nested interrupts.

OS_TASK_CHANGE_PRIO_EN

`OS_TASK_CHANGE_PRIO_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSTaskChangePrio()`. If your application never changes task priorities after they are assigned, you can reduce the amount of code space used by µC/OS-II by setting `OS_TASK_CHANGE_PRIO_EN` to 0.

OS_TASK_CREATE_EN

`OS_TASK_CREATE_EN` enables (when set to 1) or disables (when set to 0) the code generation of the `OSTaskCreate()` function. Enabling this function makes µC/OS-II backward compatible with the µC/OS task-creation function. If your application always uses `OSTaskCreateExt()` (recommended), you can reduce the amount of code space used by µC/OS-II by setting `OS_TASK_CREATE_EN` to 0. Note that you must set at least `OS_TASK_CREATE_EN` or `OS_TASK_CREATE_EXT_EN` to 1. If you wish, you can use both.

OS_TASK_CREATE_EXT_EN

OS_TASK_CREATE_EN enables (when set to 1) or disables (when set to 0) the code generation of the function OSTaskCreateExt(), which is the extended, more powerful version of the two task-creation functions. If your application never uses OSTaskCreateExt(), you can reduce the amount of code space used by µC/OS-II by setting OS_TASK_CREATE_EXT_EN to 0. Note that you need the extended task-create function to use the stack-checking function OSTaskStkChk().

OS_TASK_DEL_EN

OS_TASK_DEL_EN enables (when set to 1) or disables (when set to 0) code generation of the function OSTaskDel(), which deletes tasks. If your application never uses this function, you can reduce the amount of code space used by µC/OS-II by setting OS_TASK_DEL_EN to 0.

OS_TASK_NAME_EN

This constant determines whether you can assign names to tasks. If OS_TASK_NAME_EN is set to 0, this feature is disabled and no RAM is used in the OS_TCB for the task name.

OS_TASK_PROFILE_EN

This constant allows variables to be allocated in each task's OS_TCB that hold performance data about each task. Specifically, if OS_TASK_PROFILE_EN is set to 1, each task will have a variable to keep track of the number of context switches, the task execution time, the number of bytes used by the task and more.

OS_TASK_STAT_EN

OS_TASK_STAT_EN specifies whether or not you can enable the µC/OS-II statistic task, as well as its initialization function. When set to 1, the statistic task OS_TaskStat() and the statistic-task-initialization function are enabled. OS_TaskStat() computes the CPU usage of your application. When enabled, it executes every second and computes the 8-bit variable OSCPUUsage, which provides the percentage of CPU use of your application. OS_TaskStat() calls OSTaskStatHook() every time it executes so that you can add your own statistics as needed. See OS_CORE.C for details on the statistic task. The priority of OS_TaskStat() is always set to OS_LOWEST_PRIO-1.

The global variables OSCPUUsage, OSIdleCtrMax, OSIdleCtrRun, OSTaskStatStk[], and OSStatRdy are not declared when OS_TASK_STAT_EN is set to 0, which reduces the amount of RAM needed by µC/OS-II if you don't intend to use the statistic task. OSIdleCtrRun contains a snapshot of OSIdleCtr just before OSIdleCtr is cleared to zero every second. OSIdleCtrRun is not used by µC/OS-II for any other purpose. However, you can read and display OSIdleCtrRun if needed.

OS_TASK_STAT_STK_CHK_EN

This constant allows the statistic task to determine the actual stack usage of each active task. If OS_TASK_STAT_EN is set to 0 (the statistic task is not enabled) but, you can call OS_TaskStatStkChk() yourself from one of your tasks. If OS_TASK_STAT_EN is set to 1, stack sizes will be determined every second by the statistic task.

OS_TASK_SUSPEND_EN

OS_TASK_SUSPEND_EN enables (when set to 1) or disables (when set to 0) code generation of the functions OSTaskSuspend() and OSTaskResume(), which allows you to explicitly suspend and resume tasks, respectively. If your application never uses these functions, you can reduce the amount of code space used by µC/OS-II by setting OS_TASK_SUSPEND_EN to 0.

OS_TASK_SW_HOOK_EN

Normally, µC/OS-II requires that you have a context switch hook function called OSTaskSwHook(). When set to 0, this constant allows you to omit OSTaskSwHook() from your code. This configuration constant was added to reduce the amount of overhead during a context switch in applications that doesn't require the context switch hook. Of course, you will also need to remove the calls to OSTaskSwHook() from OSTaskStartHighRdy(), OSCtxSw() and OSIntCtxSw() in OS_CPU_A.ASM.

OS_TASK_TMR_PRIO (APP_CFG.H)

OS_TASK_TMR_PRIO specifies the priority of the timer management task. You can set the priority of the timer task to anything you want. Note that timer callback functions are executed by the timer task. OS_TASK_TMR_PRIO needs to be set in your application file called APP_CFG.H.

OS_TASK_QUERY_EN

OS_TASK_QUERY_EN enables (when set to 1) or disables (when set to 0) code generation of the function OSTaskQuery(). If your application never uses this function, you can reduce the amount of code space used by µC/OS-II by setting OS_TASK_QUERY_EN to 0.

17.08 Time Management

OS_TIME_DLY_HMSM_EN

`OS_TIME_DLY_HMSM_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSTimeDlyHMSM()`, which is used to delay a task for a specified number of hours, minutes, seconds, and milliseconds.

OS_TIME_DLY_RESUME_EN

`OS_TIME_DLY_RESUME_EN` enables (when set to 1) or disables (when set to 0) the code generation of the function `OSTimeDlyResume()`.

OS_TIME_GET_SET_EN

`OS_TIME_GET_SET_EN` enables (when set to 1) or disables (when set to 0) the code and data generation of the functions `OSTimeGet()` and `OSTimeSet()`. If you don't need to use the 32-bit tick counter `OSTime`, then you can save yourself 4 bytes of data space and code space by not having the code for these functions generated by the compiler.

OS_TIME_TICK_HOOK_EN

Normally, µC/OS-II requires the presence of a function called `OSTimeTickHook()` which is called at the very beginning of the tick ISR. When set to `0`, this constant allows you to omit `OSTimeTickHook()` from your code. This configuration constant was added to reduce the amount of overhead during a tick ISR in applications that doesn't require this hook.

17.09 Timer Management

Note that timer management requires semaphores and thus, you need to set **OS_SEM_EN** to **1**.

OS_TMR_EN

Enables (when set to 1) or disables (when set to 0) the code generation of the timer management services.

OS_TMR_CFG_MAX

Determines the maximum number of timers you can have in your application. Depending on the amount of RAM available in your product, you can have hundreds or even thousands of timers (max. is 65500). 36 entries are reserved.

OS_TMR_CFG_NAME_EN

This constant determines whether names can be assigned to timers. If `OS_TMR_CFG_NAME_EN` is set to 0, this feature is disabled and no RAM is used in the `OS_TMR` for the timer name.

OS_TMR_CFG_WHEEL_SIZE

Timers are updated using a rotating wheel. This 'wheel' allows to reduce the number of timers that need to be updated by the timer manager task. The size of the wheel should be a fraction of the number of timers you have in your application. In other words:

```
OS_TMR_CFG_WHEEL_SIZE <= OS_TMR_CFG_MAX
```

This value should be a number between 2 and 1024. Timer management overhead is somewhat determined by the size of the wheel. A large number of entries might reduce the overhead for timer management but would require more RAM. Each entry requires a pointer and a count (16-bit value). We recommend a number that is NOT a multiple of the tick rate. If your application has many timers then it's recommended that you have a high value. As a starting value, you could use `OS_TMR_CFG_MAX / 4`.

OS_TMR_CFG_TICKS_PER_SEC

This configuration constant determines the rate at which timers are updated (in Hz). Timer updates should be done at a fraction of the tick rate (i.e. `OS_TICKS_PER_SEC`). We recommend that you update timers at 10 Hz.

17.10 Function Summary

Table 17.1 lists each µC/OS-II function by type (**Service**), indicates which variables enable the code (**Set to 1**), and lists other configuration constants that affect the function (**Other Constants**).

Of course, `OS_CFG.H` must be included when µC/OS-II is built, in order for the desired configuration to take effect.

Table 17.1 µC/OS-II functions and `#define` configuration constants.

Service	*Set to 1*	*Other Constants*
Miscellaneous		
`OSEventNameGet()`	`OS_EVENT_NAME_EN`	N/A
`OSEventNameSet()`	`OS_EVENT_NAME_EN`	N/A
`OSEventPendMulti()`	`OS_EVENT_MULTI_EN`	

`OSInit()`	N/A	`OS_MAX_EVENTS`
		`OS_Q_EN and OS_MAX_QS`
		`OS_MEM_EN`
		`OS_TASK_IDLE_STK_SIZE`
		`OS_TASK_STAT_EN`
		`OS_TASK_STAT_STK_SIZE`

`OSSchedLock()`	`OS_SCHED_LOCK_EN`	N/A
`OSSchedUnlock()`	`OS_SCHED_LOCK_EN`	N/A
`OSStart()`	N/A	N/A
`OSStatInit()`	`OS_TASK_STAT_EN &&` `OS_TASK_CREATE_EXT_EN`	`OS_TICKS_PER_SEC`
`OSVersion()`	N/A	N/A
Interrupt Management		
`OSIntEnter()`	N/A	N/A
`OSIntExit()`	N/A	N/A
Event Flags		
`OSFlagAccept()`	`OS_FLAG_EN`	`OS_FLAG_ACCEPT_EN`
`OSFlagCreate()`	`OS_FLAG_EN`	`OS_MAX_FLAGS`
`OSFlagDel()`	`OS_FLAG_EN`	`OS_FLAG_DEL_EN`
`OSFlagNameGet()`	`OS_FLAG_EN`	`OS_FLAG_NAME_EN`
`OSFlagNameSet()`	`OS_FLAG_EN`	`OS_FLAG_NAME_EN`
`OSFlagPend()`	`OS_FLAG_EN`	`OS_FLAG_WAIT_CLR_EN`
`OSFlagPost()`	`OS_FLAG_EN`	N/A
`OSFlagQuery()`	`OS_FLAG_EN`	`OS_FLAG_QUERY_EN`

Message Mailboxes		
OSMboxAccept()	OS_MBOX_EN	OS_MBOX_ACCEPT_EN
OSMboxCreate()	OS_MBOX_EN	OS_MAX_EVENTS
OSMboxDel()	OS_MBOX_EN	OS_MBOX_DEL_EN
OSMboxPend()	OS_MBOX_EN	N/A
OSMboxPendAbort()	OS_MBOX_EN	OS_MBOX_PEND_ABORT_EN
OSMboxPost()	OS_MBOX_EN	OS_MBOX_POST_EN
OSMboxPostOpt()	OS_MBOX_EN	OS_MBOX_POST_OPT_EN
OSMboxQuery()	OS_MBOX_EN	OS_MBOX_QUERY_EN
Memory Partition Management		
OSMemCreate()	OS_MEM_EN	OS_MAX_MEM_PART
OSMemGet()	OS_MEM_EN	N/A
OSMemNameGet()	OS_MEM_EN	OS_MEM_NAME_EN
OSMemNameSet()	OS_MEM_EN	OS_MEM_NAME_EN
OSMemPut()	OS_MEM_EN	N/A
OSMemQuery()	OS_MEM_EN	OS_MEM_QUERY_EN
Mutex Management		
OSMutexAccept()	OS_MUTEX_EN	OS_MUTEX_ACCEPT_EN
OSMutexCreate()	OS_MUTEX_EN	OS_MAX_EVENTS
OSMutexDel()	OS_MUTEX_EN	OS_MUTEX_DEL_EN
OSMutexPend()	OS_MUTEX_EN	N/A
OSMutexPost()	OS_MUTEX_EN	N/A
OSMutexQuery()	OS_MUTEX_EN	OS_MUTEX_QUERY_EN
Message Queues		
OSQAccept()	OS_Q_EN	OS_Q_ACCEPT_EN
OSQCreate()	OS_Q_EN	OS_MAX_EVENTS OS_MAX_QS
OSQDel()	OS_Q_EN	OS_Q_DEL_EN
OSQFlush()	OS_Q_EN	OS_Q_FLUSH_EN
OSQPend()	OS_Q_EN	N/A
OSQPendAbort()	OS_Q_EN	OS_Q_PEND_ABORT_EN
OSQPost()	OS_Q_EN	OS_Q_POST_EN
OSQPostFront()	OS_Q_EN	OS_Q_POST_FRONT_EN
OSQPostOpt()	OS_Q_EN	OS_Q_POST_OPT_EN
OSQQuery()	OS_Q_EN	OS_Q_QUERY_EN

Semaphore Management		
OSSemAccept()	OS_SEM_EN	OS_SEM_ACCEPT_EN
OSSemCreate()	OS_SEM_EN	OS_MAX_EVENTS
OSSemDel()	OS_SEM_EN	OS_SEM_DEL_EN
OSSemPend()	OS_SEM_EN	N/A
OSSemPendAbort()	OS_SEM_EN	OS_SEM_PEND_ABORT_EN
OSSemPost()	OS_SEM_EN	N/A
OSSemQuery()	OS_SEM_EN	OS_SEM_QUERY_EN
OSSemSet()	OS_SEM_EN	OS_SEM_SET_EN
Task Management		
OSTaskChangePrio()	OS_TASK_CHANGE_PRIO_EN	OS_LOWEST_PRIO
OSTaskCreate()	OS_TASK_CREATE_EN	OS_MAX_TASKS
OSTaskCreateExt()	OS_TASK_CREATE_EXT_EN	OS_MAX_TASKS OS_TASK_STK_CLR
OSTaskDel()	OS_TASK_DEL_EN	OS_MAX_TASKS
OSTaskDelReq()	OS_TASK_DEL_EN	OS_MAX_TASKS
OSTaskResume()	OS_TASK_SUSPEND_EN	OS_MAX_TASKS
OSTaskNameGet()	OS_TASK_NAME_EN	N/A
OSTaskNameSet()	OS_TASK_NAME_EN	N/A
OSTaskStkChk()	OS_TASK_CREATE_EXT_EN	OS_MAX_TASKS
OSTaskSuspend()	OS_TASK_SUSPEND_EN	OS_MAX_TASKS
OSTaskQuery()	OS_TASK_QUERY_EN	OS_MAX_TASKS
OS_TaskStatStkChk()	OS_TASK_STAT_STK_CHK_EN	N/A
Time Management		
OSTimeDly()	N/A	N/A
OSTimeDlyHMSM()	OS_TIME_DLY_HMSM_EN	OS_TICKS_PER_SEC
OSTimeDlyResume()	OS_TIME_DLY_RESUME_EN	OS_MAX_TASKS
OSTimeGet()	OS_TIME_GET_SET_EN	N/A
OSTimeSet()	OS_TIME_GET_SET_EN	N/A
OSTimeTick()	N/A	N/A
Timer Management		
OSTmrCreate()	OS_TMR_EN	N/A
OSTmrDel()	OS_TMR_EN	N/A
OSTmrNameGet()	OS_TMR_EN && OS_TMR_CFG_NAME_EN	N/A
OSTmrRemainGet()	OS_TMR_EN	N/A
OSTmrStart()	OS_TMR_EN	N/A
OSTmrStop()	OS_TMR_EN	N/A
OSTmrSignal()	OS_TMR_EN	OS_TMR_CFG_TICKS_PER_SEC

User-Defined Functions		
OSTaskCreateHook()	OS_CPU_HOOKS_EN	N/A
OSTaskDelHook()	OS_CPU_HOOKS_EN	N/A
OSTaskStatHook()	OS_CPU_HOOKS_EN	N/A
OSTaskSwHook()	OS_CPU_HOOKS_EN	OS_TASK_SW_HOOK_EN
OSTimeTickHook()	OS_CPU_HOOKS_EN	OS_TIME_TICK_HOOK_EN