Micrium®
Embedded Software

# µC/OS-III® Performance Optimization ARM Cortex-M (Part 2)

**June 23, 2017 V.1.0**

## Introduction

µC/OS-III has a rich set of built-in instrumentation that collects real-time performance data. This data can be used to provide invaluable insight into your kernel-based application, allowing you to have a better understanding of the run-time behavior of your system. Having this information readily available can, in some cases, uncover potential real-time programming errors and allow you to optimize your application.

Once your application performs satisfactorily and meets your real-time needs, you can compile-out the instrumentation code, which would not only reduce code and data size, but would also improve the kernel's performance.

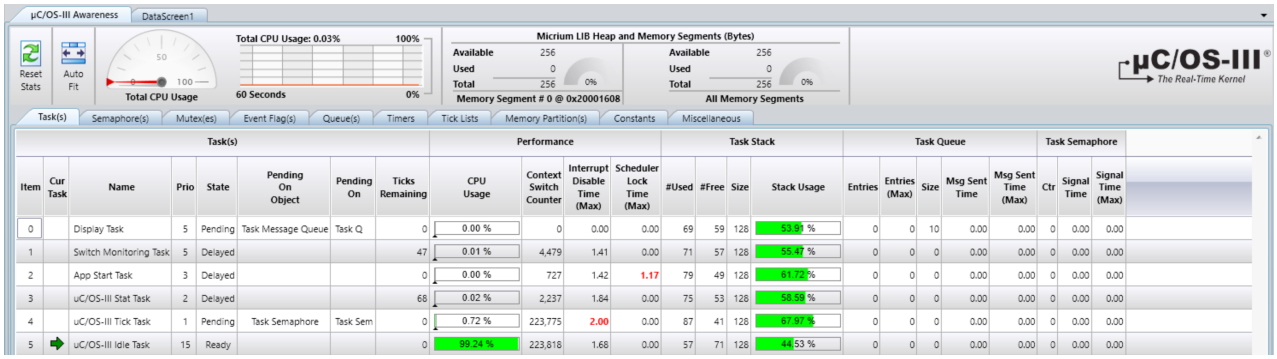
https://www.micrium.com/exploring-ucos-iiis-built-in-performance-measurements-part-i/

https://www.micrium.com/exploring-ucos-iii-built-in-performance-measurements-part-ii/

When enabled, µC/OS-III's kernel instrumentation affects performance but could be removed for release builds. In this second blog, we'll look at how you can improve performance by selectively disabling instrumentation capabilities in µC/OS-III.

We'll assume you are using a Cortex-M in our examples, but these improvements are applicable to other CPU architectures. This document also assumes that you are familiar with µC/OS-III.

Figure 1 shows a screenshot of a µC/OS-III-based application being monitored with µC/Probe. In the next sections, we will explain how to improve performance and what performance metric(s) will be sacrificed in exchange.




| Task(s) | | | | | | | | Performance | | | | Task Stack | | | | Task Queue | | | | | Task Semaphore | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Item | Cur Task | Name | Prio | State | Pending On Object | Pending On | Ticks Remaining | CPU Usage | Context Switch Counter | Interrupt Disable Time (Max) | Scheduler Lock Time (Max) | #Used | #Free | Size | Stack Usage | Entries | Entries (Max) | Size | Msg Sent Time | Msg Sent Time (Max) | Ctr | Signal Time | Signal Time (Max) |
| 0 | | Display Task | 5 | Pending | Task Message Queue | Task Q | 0 | 0.00 % | 0 | 0.00 | 0.00 | 69 | 59 | 128 | 53.91 % | 0 | 0 | 10 | 0.00 | 0.00 | 0 | 0.00 | 0.00 |
| 1 | | Switch Monitoring Task | 5 | Delayed | | | 47 | 0.01 % | 4,479 | 1.41 | 0.00 | 71 | 57 | 128 | 55.47 % | 0 | 0 | 0 | 0.00 | 0.00 | 0 | 0.00 | 0.00 |
| 2 | | App Start Task | 3 | Delayed | | | 0 | 0.00 % | 727 | 1.42 | 1.17 | 79 | 49 | 128 | 61.72 % | 0 | 0 | 0 | 0.00 | 0.00 | 0 | 0.00 | 0.00 |
| 3 | | uC/OS-III Stat Task | 2 | Delayed | | | 68 | 0.02 % | 2,237 | 1.84 | 0.00 | 75 | 53 | 128 | 58.59 % | 0 | 0 | 0 | 0.00 | 0.00 | 0 | 0.00 | 0.00 |
| 4 | | uC/OS-III Tick Task | 1 | Pending | Task Semaphore | Task Sem | 0 | 0.72 % | 223,775 | 2.00 | 0.00 | 87 | 41 | 128 | 67.97 % | 0 | 0 | 0 | 0.00 | 0.00 | 0 | 0.00 | 0.00 |
| 5 | | uC/OS-III Idle Task | 15 | Ready | | | 0 | 99.24 % | 223,818 | 1.68 | 0.00 | 57 | 71 | 128 | 44.53 % | 0 | 0 | 0 | 0.00 | 0.00 | 0 | 0.00 | 0.00 |

Figure 1. µC/OS-III Application Monitored with µC/Probe

The upcoming sections will provide, when appropriate, an estimate of the number of CPU instructions saved. CPU instructions are used instead of actual time in microseconds.

## Reducing Critical Sections – CPU_CFG_INT_DIS_MEAS

µC/OS-III can measure the interrupt disable time of each task. The measurement is performed by introducing code in the critical section macros `CPU_CRITICAL_ENTER()` and `CPU_CRITICAL_EXIT()`. The instrumented code looks as follows, and adds about 85 CPU instructions of overhead:

```
Disable Interrupts;         // Enter critical section
CPU_IntDisMeasStart();      // 25 CPU instructions
  :
  :                         // Code within critical section
  :
CPU_IntDisMeasStop();       // 60 CPU instructions
Enable Interrupts;          // Exit critical section
```

You can disable *interrupt disable time measurement* by setting the `#if` to `0` in `cpu_cfg.h` as follows:

```
#if 0
#define CPU_CFG_INT_DIS_MEAS_EN
#endif
```

**µC/Probe**: When interrupt disable time measurement is turned off, the *Interrupt Disable Time (Max)* column will show all zeros.

### Summary:

| **Set** … | **In** … | **Saves** … |
|---|---|---|
| `#if 0`<br>`#define CPU_CFG_INT_DIS_MEAS_EN`<br>`#endif` | `cpu_cfg.h` | `85 instructions` |

## Reducing Scheduler Lock Time – OS_CFG_SCHED_LOCK_TIME_MEAS_EN

You can reduce the amount of time required to lock the scheduler with each `OSSchedLock()` call by disabling the lock time measurement. See `os_cfg.h` as follows:

```
#define  OS_CFG_SCHED_LOCK_TIME_MEAS_EN  DEF_DISABLE


Lock the scheduler;                 // Lock the scheduler
CPU_SchedLockTimeMeasStart();       // 15 CPU instructions
  :
  :                                 // Code executed while scheduler is locked
  :
CPU_SchedLockTimeMeasStop();        // 25 CPU instructions
Unlock the scheduler;               // Unlock the scheduler
```

**µC/Probe**: When scheduler lock time measurement is turned off, the *Scheduler Lock Time (Max)* column will show all zeros.

### Summary:

| **Set** … | **In** … | **Saves** … |
|---|---|---|
| `#define OS_CFG_SCHED_LOCK_TIME_MEAS_EN DEF_DISABLE` | `os_cfg.h` | `40 instructions` |

## Reducing Context Switch Time (OSTaskSwHook()) – OS_CFG_TASK_STK_REDZONE_EN

µC/OS-III measures interrupt disable time on a per-task basis, so disabling interrupt disable time measurement also reduces the context switch time by about 85 CPU instructions.

Similarly, µC/OS-III measures scheduler lock time on a per-task basis, so disabling scheduler lock time measurement reduces the context switch time by about 15 CPU instructions.

µC/OS-III can also perform red-zone stack checking during a context switch if that feature is enabled by `OS_CFG_TASK_STK_REDZONE_EN` in `os_cfg.h`. Checking an 8-entry redzone consumes about 120 CPU instructions.

### Summary:

| **Set** … | **In** … | **Saves** … |
|---|---|---|
| `#if 0`<br>`#define CPU_CFG_INT_DIS_MEAS_EN`<br>`#endif` | `cpu_cfg.h` | `85 CPU instructions per task-switch` |
| `#define OS_CFG_SCHED_LOCK_TIME_MEAS_EN DEF_DISABLE` | `os_cfg.h` | `15 CPU instructions per task-switch` |
| `#define OS_CFG_TASK_STK_REDZONE_EN DEF_DISABLE` | `os_cfg.h` | `120 CPU instructions per task-switch` |

## Disabling CPU Usage Calculations and Context Switch Counters – OS_CFG_TASK_PROFILE_EN

The statistics task in µC/OS-III computes global CPU usage (as a percentage) as well as CPU usage on a per-task basis. CPU usage is measured by taking a snapshot of the Cortex-M DWT cycles counter upon switching to a task and once again upon switching out of the task. The delta time is computed and saved for use by the statistics task. See `OSTaskSwHook()` in `os_cpu_c.c`.

Also, a per-task counter is incremented on each context switch, which is used to show CPU activity for the task.

If you don't need that feature, you can turn it OFF by setting `OS_CFG_TASK_PROFILE_EN` to `DEF_DISABLED` in `os_cfg.h`.

**µC/Probe**: When task profiling is turned off, the *Context Switch Counter* column will show all zeros.

**Summary:**

| **Set …** | **In …** | **Saves …** |
|---|---|---|
| `#define OS_CFG_TASK_PROFILE_EN  DEF_DISABLED` | `os_cfg.h` | `~50 CPU instructions` |

## Turn OFF Time Stamping – OS_CFG_TS_EN

µC/OS-III is able to measure the time it takes between posting to a task and when the task receives the post. This is done by sampling the DWT timer, which measures time in CPU clock cycles. If you don't need that feature, you can turn it OFF by setting `OS_CFG_TS_EN` to `DEF_DISABLED` in `os_cfg.h`.

**µC/Probe**: When time stamping is disabled, the *Msg Sent Time*, *Msg Sent Time (Max)*, *Signal Sent Time* and *Signal Sent Time (Max)* columns will show all zeros.

**Summary:**

| Set … | In … | Saves … |
|---|---|---|
| `#define OS_CFG_TS_EN DEF_DISABLED` | `os_cfg.h` | `~50 CPU instructions` |

## Turn OFF Debug – OS_CFG_DBG_EN

The µC/OS-III code contains additional variables that are used to keep track of how many objects have been allocated, and creates linked lists of objects so they can be monitored by µC/Probe or other kernel awareness tools, and more. You can disable this feature, which will save time when creating and deleting kernel objects. This is accomplished by setting `OS_CFG_DBG_EN` to `DEF_DISABLED` in `os_cfg.h`.

**µC/Probe**: When debug is disabled, the *Task Queue (Entries, Entries (Max) and Size)* columns will display zeros. The *Task Semaphore Ctr* column will display zeros.

**Summary:**

| **Set …** | **In …** | **Saves …** |
|---|---|---|
| `#define OS_CFG_DBG_EN  DEF_DISABLED` | `os_cfg.h` | `25 CPU instructions per object create/delete` |

## Disabling the Stack Checking – OS_CFG_STAT_TASK_STK_CHK_EN

If you don't need the ability to monitor or display task stack usage, then you can disable this feature by setting `OS_CFG_STAT_TASK_STK_CHK_EN` to `DEF_DISABLED` in `os_cfg.h`. Stack usage is probably one of the most important pieces of information you can get from µC/OS-III's statistics task. It's recommended that you keep this feature unless you need to squeeze every ounce of performance out of your application.

**µC/Probe**: When you disable stack checking, you will lose the ability to monitor and display task stack usage.

### Summary:

| Set … | In … | Saves … |
|---|---|---|
| `#define OS_CFG_STAT_TASK_STK_CHK_EN DEF_DISABLED` | `os_cfg.h` | `1000s of non-time- critical CPU instructions` |

## Disabling the Statistics Task – OS_CFG_STAT_TASK_EN

If you don't need the ability to display global CPU usage, CPU usage on a per-task basis or stack usage on a per-task basis, then you can disable the statistics task by setting `OS_CFG_STAT_TASK_EN` to `DEF_DISABLED` in `os_cfg.h`. However, the statistics task has a low priority and fairly low overhead.

**µC/Probe**: When you disable the statistics task, you will be losing the ability to monitor and display task stack usage along with a few other statistics.

### Summary:

| Set … | In … | Saves … |
|---|---|---|
| `#define OS_CFG_STAT_TASK_EN  DEF_DISABLED` | `os_cfg.h` | 1000s of non-time-critical CPU instructions |

## Further Reading

To learn more about µC/OS-III including not only how the kernel works but also an API reference in detail, read the full µC/OS-III Documentation.

## Getting Help

Don't hesitate to contact us at Micrium: www.micrium.com.