CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

ARM Cortex-M4 – Atollic

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

This document details the port of the Abassi RTOS to the ARM Cortex-M4 processor. The software suite used for this specific port is the Atollic TrueSTUDIO for ARM; the version used for the port and all tests is V3.1.0 Pro.

1.1 Distribution Contents

The set of files supplied with this distribution are listed in Table 1-1 below:

Table 1-1 Distribution

File Name	Description
`Abassi.h`	Include file for the RTOS
`Abassi.c`	RTOS "C" source file
`syscalls.c`	Minimum system call file modified for multi-tasking
`Abassi_CORTEXM4_ATOLLIC.s`	RTOS assembly file for the ARM Cortex-M4 to use with the ATOLLIC TrueSTUDIO.
`Demo_1_STM32_P407_ATOLLIC.c`	Demo code that runs on the Olimex STM32-P407 evaluation board
`Demo_3_STM32_P407_ATOLLIC.c`	Demo code that runs on the Olimex STM32-P407 evaluation board
`Demo_5_STM32_P407_ATOLLIC.c`	Demo code that runs on the Olimex STM32-P407 evaluation board
`Demo_7_STM32_P407_ATOLLIC.c`	Demo code that runs on the Olimex STM32-P407 evaluation board
`AbassiDemo.h`	Build option settings for the demo code

NOTE: The supplied file `syscalls.c` MUST be used with all applications based on Abassi. This `syscalls.c` is a slightly modified version of the default Atollic file. Not using the supplied file will most likely fail memory allocation through `malloc()`, or not report `malloc()` failures. Even if the application does not use memory allocation, the library itself internally calls `malloc()`.

1.2 Limitations

To optimize reaction time of the Abassi RTOS components, it was decided to require the processor to always operate in privileged mode (which is the default start-up mode for Cortex-M microcontrollers) and to always use the main stack pointer (MSP). The start-up code supplied in the distribution fulfills these constraints and one must be careful to not change these settings in the application.

The `SVCall` interrupt (interrupt number -5 / interrupt vector number 11) is not available as it is reserved for the OS, and the Abassi RTOS uses it.

2 Target Set-up

Very little is needed to configure the Atollic TrueSTUDIO development environment to use the Abassi RTOS in an application. All there is to do is to add the files `Abassi.c`, `Abassi_CORTEXM4_ATOLLIC.s` and `syscalls.c` (the one supplied with the distribution) in the source files of the application project, and make sure the four configuration settings in the file `Abassi_CORTEXM4_ATOLLIC.s` (`OS_ISR_STACK` as described in Section 2.1, `OS_HANDLE_PSR_Q` as described in Section 2.2, `OS_FPU_ON_OFF` described in Section 2.3, and `OS_ATOLLIC_REENT` described in Section 2.4) are set according to the needs of the application. As well, update the include file path in the C/C++ compiler preprocessor options with the location of `Abassi.h`. There is no need to include a start-up file, nor a file for the interrupt table, as the `Abassi_CORTEXM4_ATOLLIC.s` file contains all the start-up operations, including the interrupt table and exception handlers.

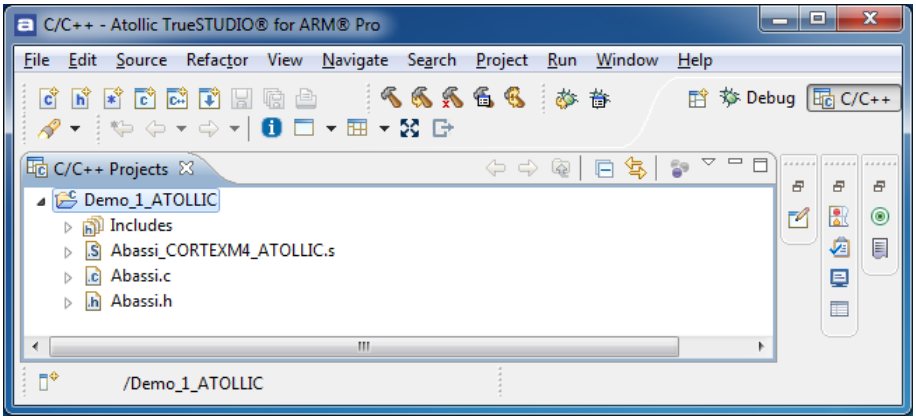

Figure 2-1 Project File List

2.1 Interrupt Stack Set-up

It is possible, and is highly recommended, to use a hybrid stack when nested interrupts occur in an application. Using this hybrid stack, specially dedicated to the interrupts, removes the need to allocate extra room to the stack of every task in the application to handle the interrupt nesting. This feature is controlled by the value set by the definition `OS_ISR_STACK`, located around line 25 in the file `Abassi_CORTEXM4_ATOLLIC.s`. To disable this feature, set the definition of `OS_ISR_STACK` to a value of zero. To enable it, and specify the interrupt stack size, set the definition of `OS_ISR_STACK` to the desired size in bytes (see Section 4 for information on stack sizing). As supplied in the distribution, the hybrid stack feature is enabled, and a size of 1024 bytes is allocated; this is shown in the following table:

Table 2-1 `OS_ISR_STACK`

```
#ifndef OS_ISR_STACK
  .equ OS_ISR_STACK, 1024       /* If using a dedicated stack for the nested ISRs  */
#endif                          /* 0 if not used, otherwise size of stack in bytes */
```

Alternatively, it is possible to overload the `OS_ISR_STACK` value set in `Abassi_CORTEXM4_ATOLLIC.s` by using the assembler command line option `-D` and specifying the desired hybrid stack size, as shown in the following example, where the hybrid stack size is set to 512 bytes:

Table 2-2 Command line set of `OS_ISR_STACK`

```
arm-atollic-eabi-gcc  assembler-with-cpp … -DOS_ISR_STACK=512 …
```

The hybrid stack size can also be set through the GUI, in the "*C/C++ Build → Settings → Tool Setting → Assembler → Symbols*" menu, as shown in the following figure:

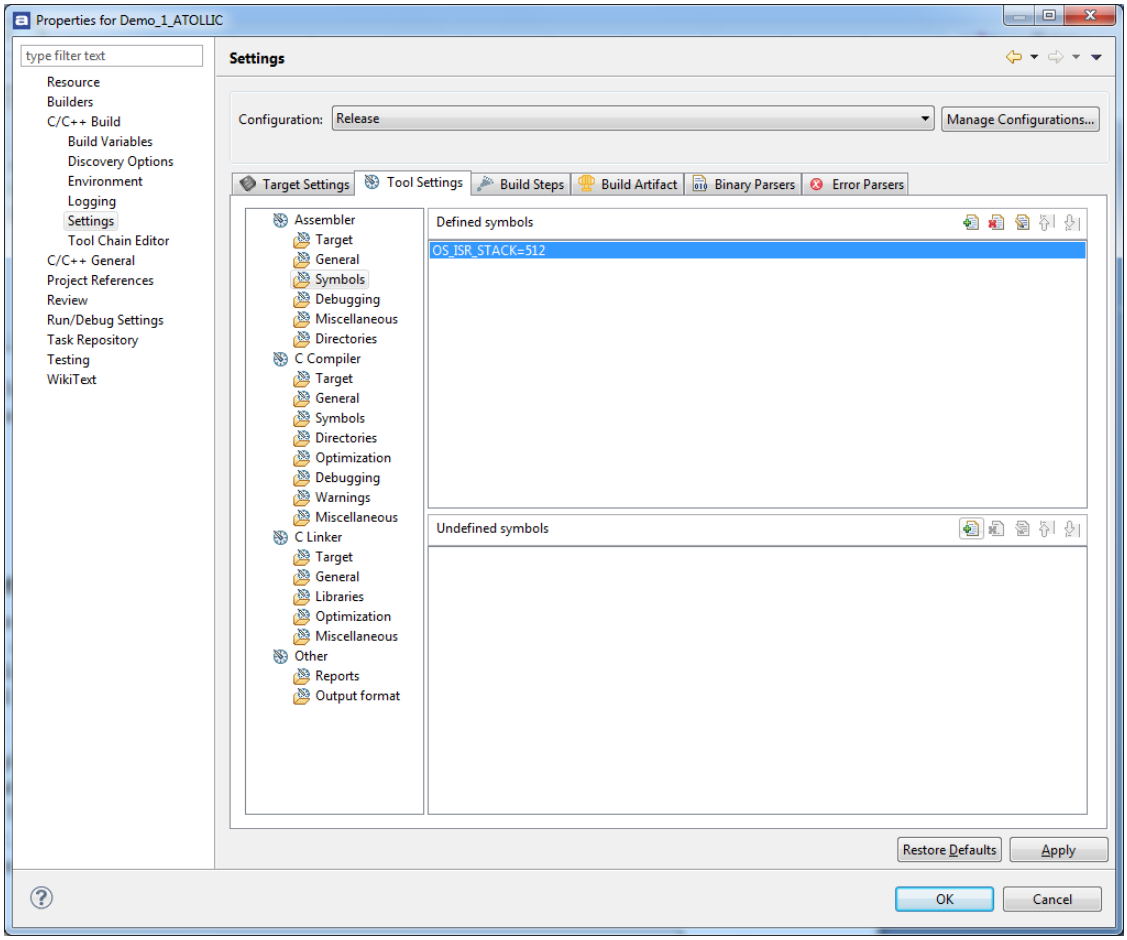

Figure 2-2 GUI set of `OS_ISR_STACK`

2.2 Saturation Bit Set-up

In the ARM Cortex-M4 status register, there is a sticky bit to indicate if an arithmetic saturation or overflow has occurred during a DSP instruction; this is the Q flag in the status register (bit #27). By default, this bit is not kept localized at the task level, as it needs extra processing during a context switch to do so; instead, it is propagated across all tasks. This choice was made because most applications do not care about the value of this bit.

If this bit is relevant for an application, even in a single task, then it must be kept locally in each task. To keep the meaning of the saturation bit localized, the token `OS_HANDLE_PSR_Q` must be set to a non-zero value; to disable it, it must be set to a zero value. This is located at around line 35 in the file `Abassi_CORTEXM4_ATOLLIC.s`. The distribution code disables the localization of the Q bit, setting the token `OS_HANDLE_PSR_Q` to zero, as shown in the following table:

Table 2-3 Saturation Bit configuration

```
#ifndef OS_HANDLE_PSR_Q
  .equ OS_HANDLE_PSR_Q, 0        /* If we keep the Q bit (saturation) on per tasks */
#endif
```

Alternatively, it is possible to overload the `OS_HANDLE_PSR_Q` value set in `Abassi_CORTEXM4_ATOLLIC.s` by using the assembler command line option `-D` and specifying the desired setting with the following:

Table 2-4 Command line set of Saturation Bit configuration

```
arm-atollic-eabi-gcc  assembler-with-cpp … -DOS_HANDLE_PSR_Q=1 …
```

The saturation bit configuration can also be set through the GUI, in the "*C/C++ Build → Settings → Tool Setting → Assembler → Symbols*" menu, as shown in the following figure:

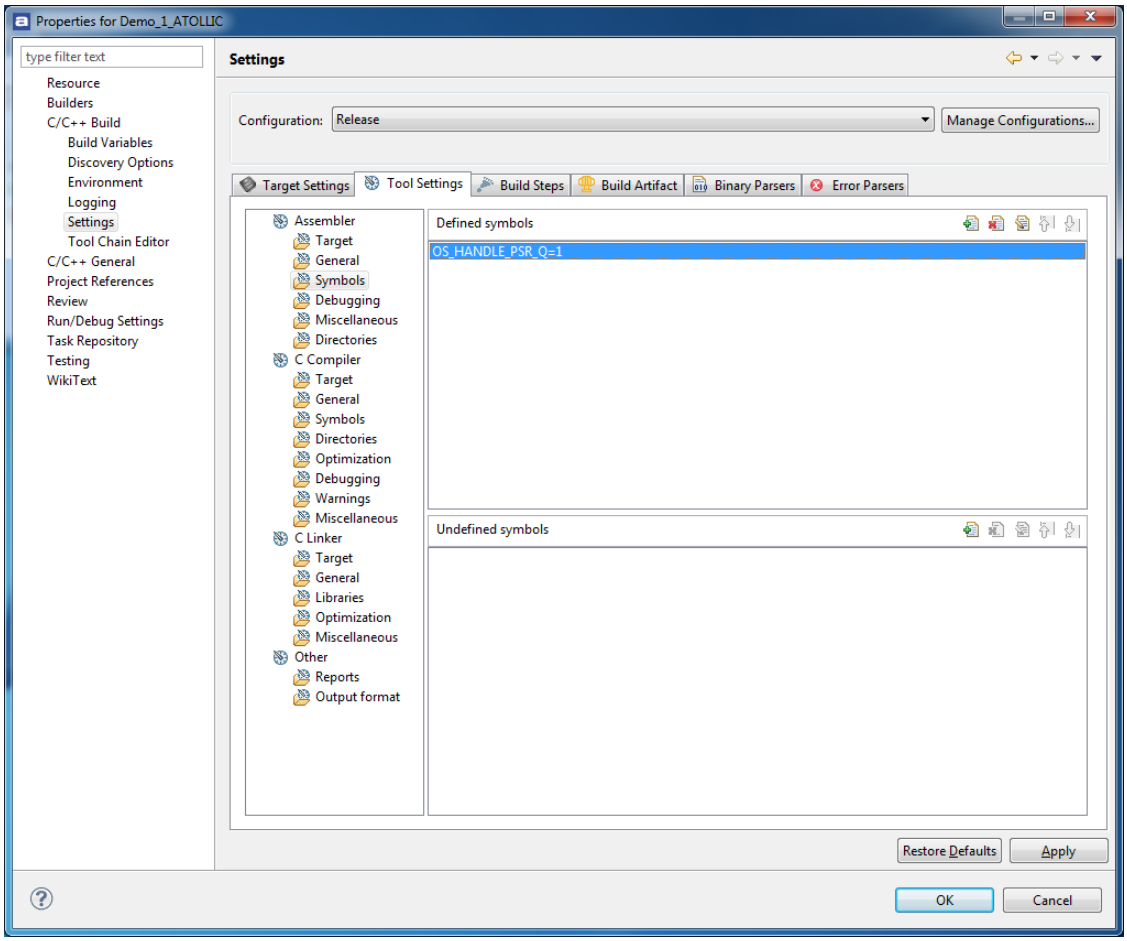

Figure 2-3 GUI set of Saturation Bit configuration

2.3 FPU set-up

The assembly file `Abassi_CORTEXM4_ATOLLIC.s`, depending on its configuration, handles three different types of FPU use. They are:

- The FPU is always disabled
- The FPU is always enabled
- The FPU is turned on and turned off during runtime

The file `Abassi_CORTEXM4_ATOLLIC.s` is aware of the enabling or disabling of the FPU by the assembly setting in the GUI in "*C/C++ Build → Settings → Tool Settings → Assembler → Target → Floating point"* which ends up defining the symbol `__SOFTFP__` when the floating point setting is set to *"Software implementation"*; the symbol `__SOFTFP__` is not defined for the two other settings, meaning the FPU is used.

There are two ways to set-up the assembler to support the FPU instructions. This is done on the command line through the option `-mfloat-abi=softfp`. If the command line option `-mfloat-abi=softfp` is specified, then all floating point operations use the software implementation, and the FPU is not used. If the command line option `-mfloat-abi=softfp` is not specified, then the FPU is used:

Table 2-5 Command line disabling the FPU

```
arm-atollic-eabi-gcc  assembler-with-cpp … -mfloat-abi=softfp …
```

The enabling of the FPU can also be performed through the GUI, in the "*C/C++ Build → Settings → Tool Settings → Assembler → Target → Floating point"* menu, by setting the Floating Point to a setting different than *Software implementation*. If the setting for Floating Point is set to *Software implementation*, then the FPU is turned off.

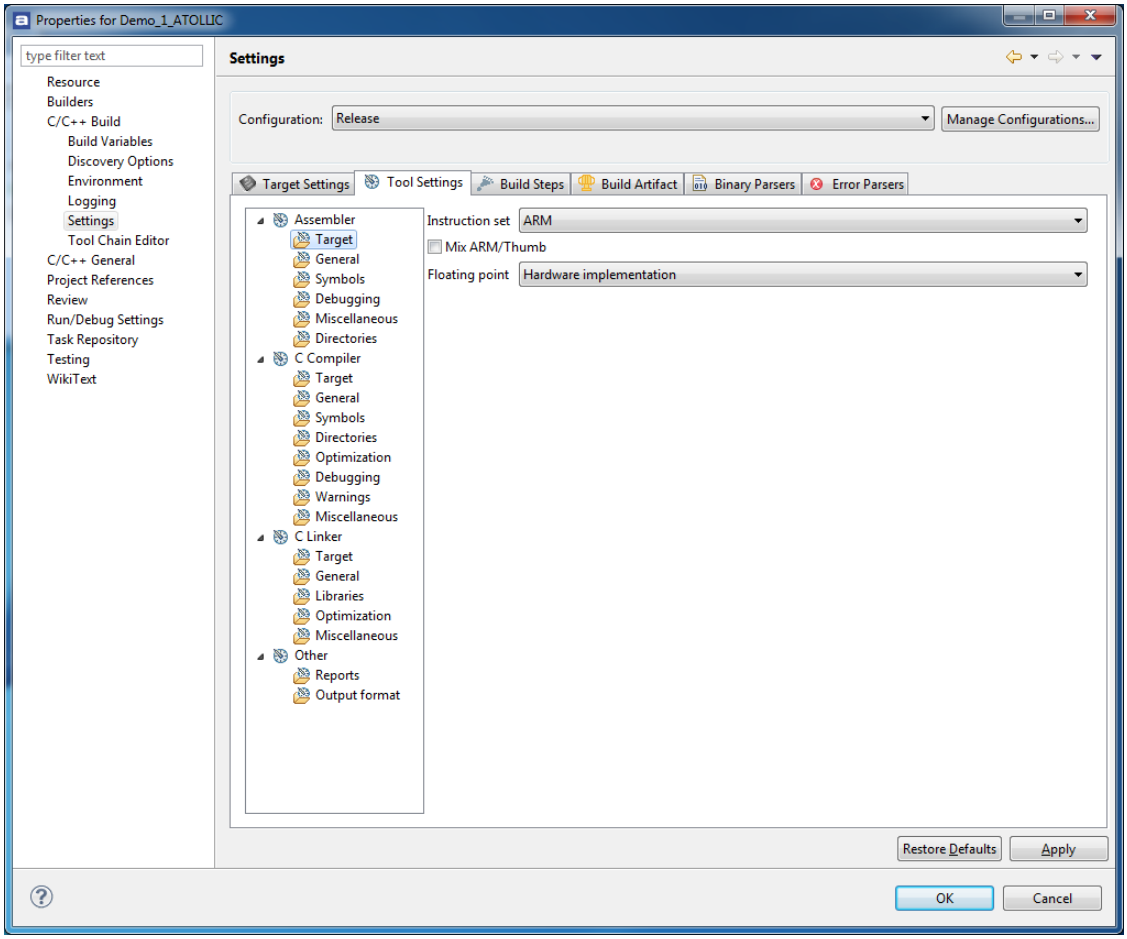

Figure 2-4 GUI enabling of the FPU

NOTE: The setting of the CPU must be identical for both the assembler AND the compiler. A mismatch between the two settings will most likely result in a run time hard fault.

When the FPU is enabled, each task can use a different configuration of the FPU (through the `FPCSR` register), as the contents of this register are part of the task context save. All tasks upon start will have their local `FPCSR` value set according to the value of `FPCSR` register upon calling `OSstart()`. This means if the application globally requires a different setting of the FPU than the default set by the compiler, the `FPCSR` must be modified before calling `OSstart()`.

It is also possible to turn on and turn off the FPU during runtime, and the ON / OFF setting is also kept on a per task basis. For this feature to be available, the FPU must be used (the token `__SOFTFP__` not defined, or the GUI configuration for the compiler and assembler not set to *"Software implementation"*). This means the FPU can be enabled in a set of tasks, and not for the other tasks in the application. All tasks, upon start, will inherit the same ON / OFF state of the FPU as when `OSstart()` was called. When this feature is required, the build option `OS_FPU_ON_OFF`, located around line 40 in the file `Abassi_CORTEXM4_ATOLLIC.s`, must be set to a non-zero value. The distribution code does not enable the capability of turning the FPU ON and OFF during runtime, setting the token `OS_FPU_ON_OFF` to zero, as shown in the following table:

Table 2-6 FPU run time ON / OFF configuration

```
#ifndef OS_FPU_ON_OFF
  .equ OS_FPU_ON_OFF, 0         /* If the FPU can be turned ON/OFF during runtime */
#endif
```

Alternatively, it is possible to overload the `OS_FPU_ON_OFF` value set in `Abassi_CORTEXM4_ATOLLIC.s` by using the assembler command line option `-D` and specifying the desired setting for `OS_FPU_ON_OFF` with the following:

Table 2-7 Command line set of `OS_FPU_ON_OFF`

```
arm-atollic-eabi-gcc  assembler-with-cpp … -mfloat-abi=softfp  -DOS_FPU_ON_OFF=1  …
```

The indication the FPU is turned on and off during runtime can also be set through the GUI, in the "*C/C++ Build → Settings → Tool Setting → Assembler → Symbols*" menu, as shown in the following figure:

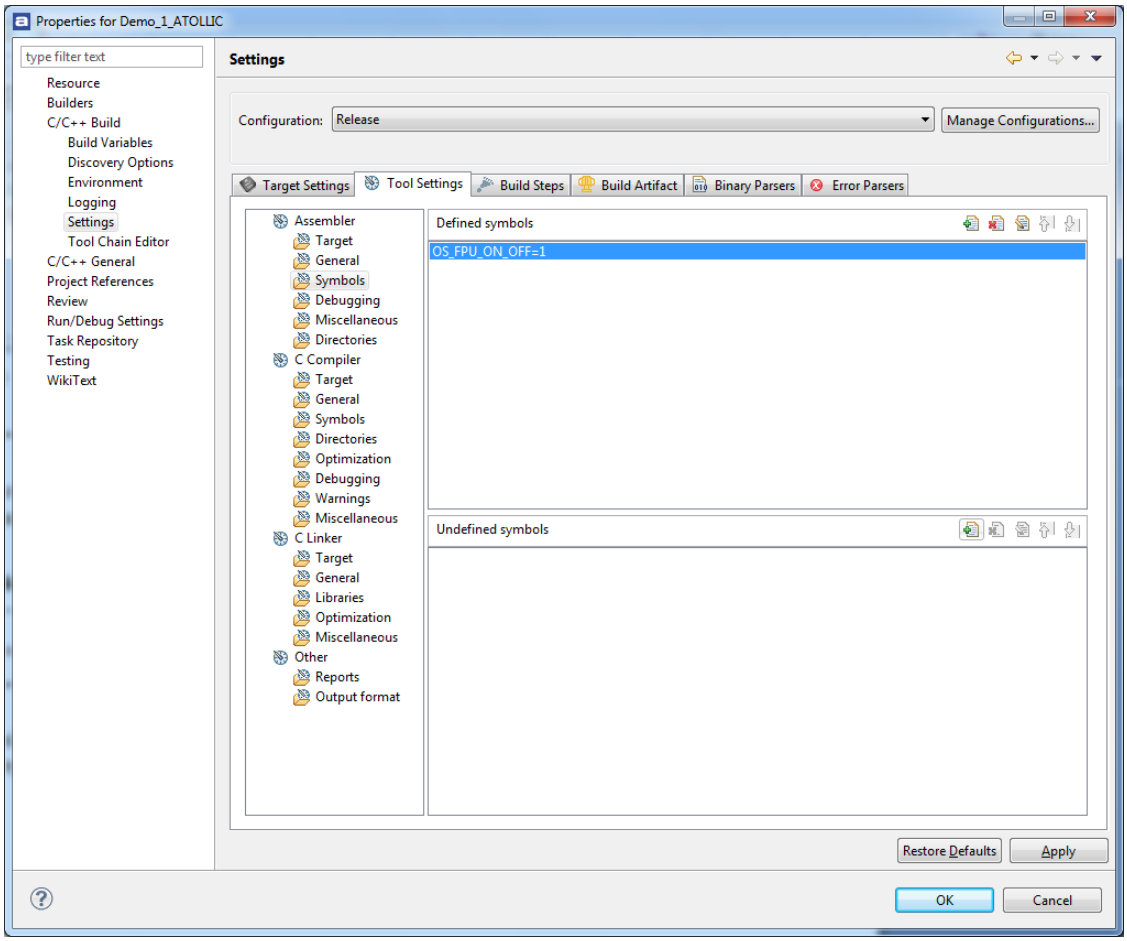

Figure 2-5 GUI set of `OS_FPU_ON_OFF`

There are two requirements to fulfill when the FPU is turned on and off during runtime. The first one, which is not related to the RTOS but is a restriction by the Cortex-M4 core, is to never have a different enable setting of the FPU between the entry and the exit of an ISR. This means that turning ON and then OFF the FPU in an interrupt is safe. But turning it ON, without turning it OFF before exiting the interrupt, will crash the application. If the FPU is ON upon entry in the interrupt and it gets turned OFF in the interrupt without being turned back ON, it will trigger an access fault exception. Beware of interrupt nesting when turning on and off the FPU in an interrupt. If two interrupts that can be nested turn on and off the FPU, then turning off the FPU at the end of the interrupt will break the required condition. Instead, the entry state of the FPU must be restored at the exit of the interrupt.

The second requirement when the FPU is turned ON and OFF during runtime is that it is necessary to set the `SVCall` (Service call exception vector #11, interrupt #-5) priority to the highest level. This is configured in the System Handler Priority Register 2 (SHPR2) register. If this register is not modified, then at start-up the priority of the `SVCall` exception is set to the higher level.

NOTE: When the FPU is turned OFF in a task, the setting of the `FPCSR` will quite likely be set back to the task start-up value upon turning ON the FPU afterward.

2.4 Multithreading

The Atollic `libc` library can be set to be fully reentrant and multithread-safe. The multithreading setting on how the library is used depends on the definition of the build option `OS_ATOLLIC_REENT`. If this build option is not defined, or if it is defined with a value of zero, the library is neither reentrant nor multithread-safe. If the build option is positive, the library is multithread-safe and reentrant for each one of the tasks. If the build option value is negative, only selected tasks use the library in a multithread-safe and reentrant manner.

2.4.1 Full multithreading

For full multithreading of the library, all there is to do is to define the build option `OS_ATOLLIC_REENT` with a positive value, for both the compiler and the assembler.

The build option can be set directly in the assembly file; this is located at around line 45 in the file `Abassi_CORTEXM4_ATOLLIC.s`. The distribution code disables multithreading, setting the token `OS_ATOLLIC_REENT` to zero, as shown in the following table:

Table 2-8 Assembly file multithread configuration

```
#ifndef OS_ATOLLIC_REENT    /* When library re-entrance is required, when +ve the  */
 .equ   OS_ATOLLIC_REENT, 0 /* task context switch updates the _impure_ptr variable*/
#endif                      /* with the task's libc context                        */
```

Alternatively, it is possible to overload the `OS_ATOLLIC_REENT` value set in `Abassi_CORTEXM4_ATOLLIC.s` by using the assembler command line option `-D` and specifying the desired setting with the following:

Table 2-9 Command line set of multithread configuration

```
arm-atollic-eabi-gcc  assembler-with-cpp … -DOS_ATOLLIC_REENT=1 …
```

The multithreading configuration can also be set through the GUI, in the "*C/C++ Build → Settings → Tool Setting → Assembler → Symbols*" menu, as shown in the following figure:

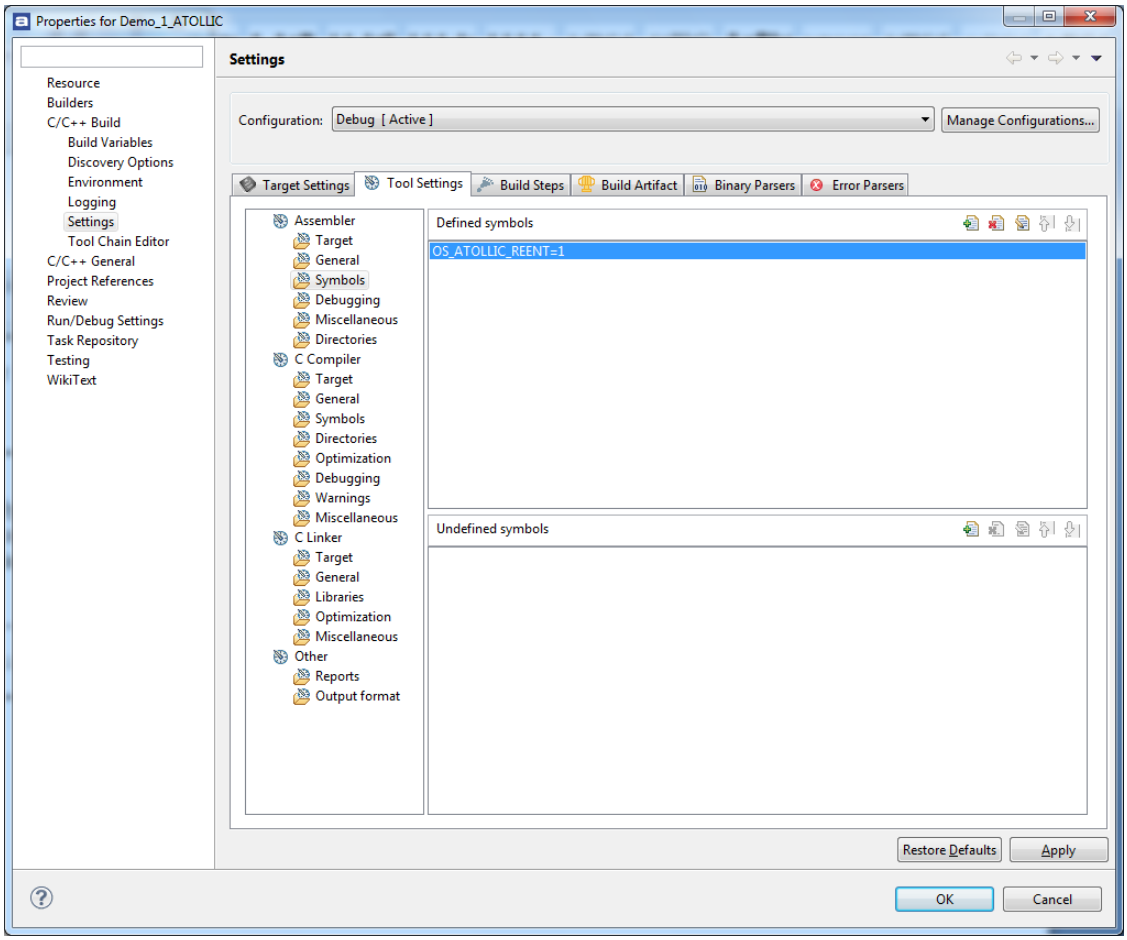

Figure 2-6 GUI set of `OS_ATOLLIC_REENT`

The exact same definition of `OS_ATOLLIC_REENT` as the one specified for the assembler must be must be given to the compiler. This can be done with the command line option `-D` and specifying the setting with the following:

Table 2-10 Command line set of multithread configuration

```
arm-atollic-eabi-gcc … -DOS_ATOLLIC_REENT=1 …
```

The multithreading configuration can also be set through the GUI, in the "*C/C++ Build → Settings → Tool Setting → C Compiler → Symbols*" menu, as shown in the following figure:

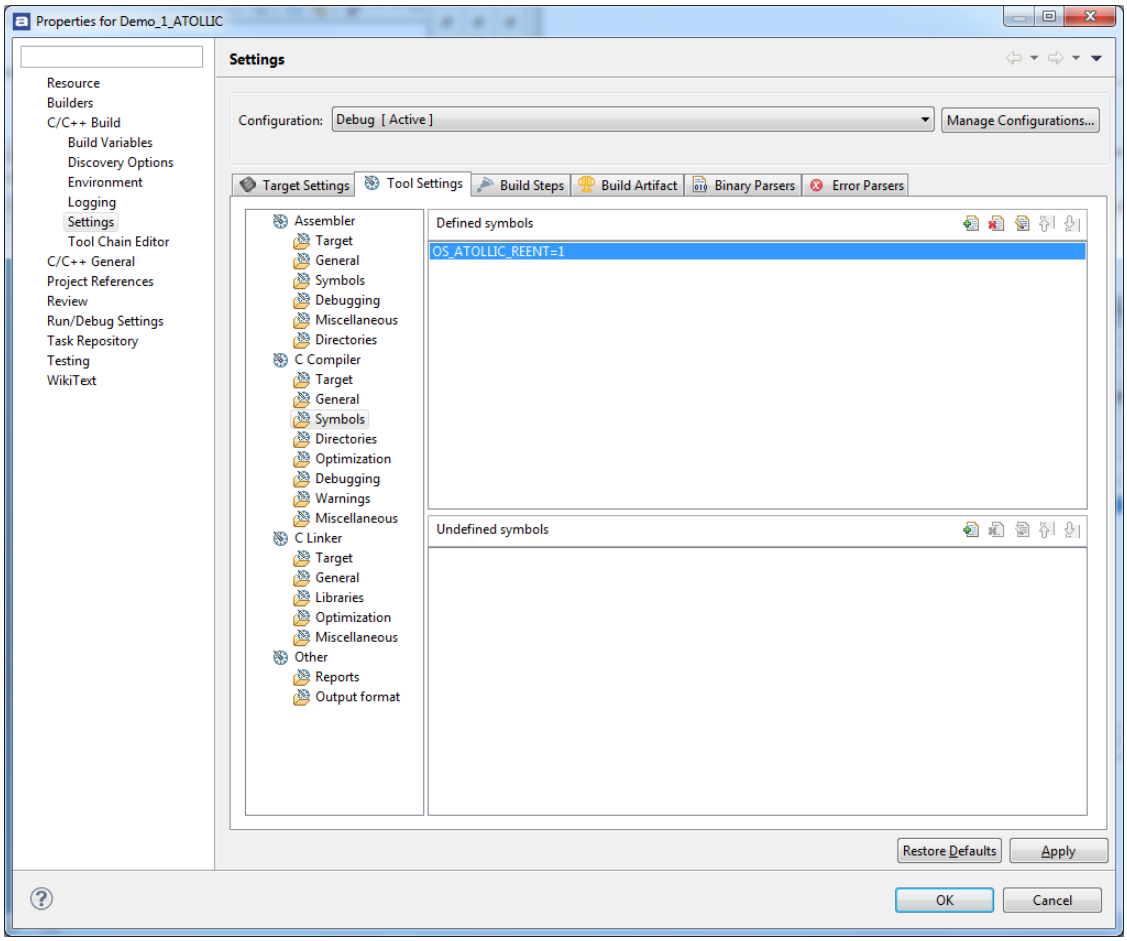

Figure 2-7 GUI set of `OS_ATOLLIC_REENT`

2.4.2 Partial multithreading

The use of full multithread protection for the library requires around ¼ kilobyte of extra data memory for each task in the application. The extra memory required is not due to Abassi, but is what the library requires to be set reentrant. On data memory restricted applications, it may be impossible to use full multithreading protection. Setting the build option `OS_ATOLLIC_REENT` to a negative value allows you to select specific tasks where reentrance is required. The library is still multithread-safe, even when the build option is negative, only the reentrance is selectable.

The build option `OS_ATOLLIC_REENT` is set the same way as described in the previous section.

Partial multithreading means that only the tasks that are set up to use the library in a reentrant manner require the ¼ kilobyte block of extra data memory. Not only is memory needed for the library context data structure, but if file I/O is used in the task, more memory is allocated for the buffering of the file or stream. It is a good practice to use the standard library function `setbuf()`, or `setvbuf()` to tailor each stream buffer size, as the library default buffer size (defined as `BUFSIZ` in `stdio.h`) is set to 1024 bytes. The most well known modules that are under reentrance control in the library are:

- The time structure `tm`
- `atexit()`
- stdio
- File I/O for stdin, stdout, stderr
- Rand / Rand48
- Errno
- Signals
- Locale
- And a few more

If a task uses none of the above modules, then the task does not need to access the library in a reentrant manner, so there is no need to reserve the memory block of ¼ kilobyte of data memory. If a task uses one or more modules, but it is the only task using this/these module(s), there is still no need to make the library reentrant for that task. Only when two or more tasks use the same modules for the library do these tasks need to access the library in a reentrant manner.

A task is set to use the library in a reentrant manner with the following:

Table 2-11 Setting a task to use re-entrant library

```
#inclde "Abassi.h"

TSK_t *TskReent_1
struct _reent Reent_1;

…
                                              /* First the task must be created    */
                                              /* in the suspended state            */
TskReent_1 = TSKcreate("TaskName", TskPrio, StackSize, TaskFct, 0);

REENT_INIT_PTR(&Reent_1);                     /* Initialzation of the libc context */

TskReent_1->XtraData[0] = (intptr_t)&Reent_1;  /* Attach the context to the task    */

TSKreseum(TskReent_1);                        /* The task may now be resumed       */
```

The declaration "`struct _reent Reent_1;`" can be replaced by a memory allocation of `sizeof(struct _reent)`.

3 Interrupts

The Abassi RTOS needs to be aware when kernel requests are performed inside or outside an interrupt context. For all interrupt sources (except interrupt numbers less than -1) the Abassi RTOS provides an interrupt dispatcher, which allows it to be interrupt-aware. This dispatcher achieves two goals. First, the kernel uses it to know if a request occurs within an interrupt context or not. Second, using this dispatcher reduces the code size, as all interrupts share the same code for the decision making of entering the kernel or not at the end of the interrupt: there is no need to add a preamble / epilogue in the functions handling the interrupts.

The distribution makes provision for 241 sources of interrupts, as specified by the token `OS_N_INTERRUPTS` in the file `Abassi_CORTEXM4_ATOLLIC.s`, and the internal default value used by `Abassi.c`. Even though the Nested Vectored Interrupt Controller (NVIC) peripheral supports a maximum of 256 interrupts on the Cortex-M4, the first 15 entries of the interrupt vector table are hard mapped to dedicated handlers (the interrupt number -1, which is attached to `SysTick`, is not hard mapped but is handled by the ISR dispatcher).

3.1 Interrupt Handling

3.1.1 Interrupt Table Size

Most devices do not require all 256 interrupts, as they typically only handle between 64 and 128 sources of interrupts. The interrupt table can be easily reduced to recover code space, and at the same time recover the same amount of data memory. There are two files affected: in `Abassi_CORTEXM4_ATOLLIC.s`, the ARM interrupt table itself must be shrunk, and the value used in the file `Abassi.c`, in order to reduce the ISR dispatcher table look-up. The interrupt table size is defined by the token `OS_N_INTERRUPTS` in the file `Abassi_CORTEXM4_ATOLLIC.s` around line 35. For the value used by `Abassi.c`, the default value can be overloaded by defining the token `OS_N_INTERRUPTS` when compiling `Abassi.c` . The distribution table size is set to 241; that is the NVIC maximum of 256 minus the 15 hard mapped exceptions.

For example, the STM32F407 device from ST Microelectronics uses only the first 100 entries of the interrupt table (84 external interrupts plus the standard 16 exceptions). The 256 entries table can therefore be reduced to 100. The value to set in `Abassi_CortexM4_ATOLLIC.s` is 85, which is the total of 100 entries minus 15 (there are 15 hard mapped exceptions). The changes are shown in the following table:

Table 3-1 `Abassi_CORTEXM4_ATOLLIC.s` interrupt table sizing

```
…

#ifndef OS_N_INTERRUPTS           /* # of entries in the interupt table mapped to  */
  .equ OS_N_INTERUPTS, 85         /* ISRdispatch()                                 */
#endif

 …
```

Alternatively, it is possible to overload the `OS_N_INTERRUPTS` value set in `Abassi_CORTEXM4_ATOLLIC.s` by using the assembler command line option `-D` and specifying the desired setting with the following:

Table 3-2 Command line set the interrupt table size

```
arm-atollic-eabi-gcc  assembler-with-cpp … -DOS_N_INTERRUPTS=85  …
```

The overloading of the default interrupt vector look-up table used by `Abassi.c` is done by using the compiler command line option `-D` and specifying the desired setting with the following:

Table 3-3 Overloading the interrupt table sizing for `Abassi.c`

```
arm-atollic-eabi-gcc … -DOS_N_INTERRUPTS=85  …
```

The interrupt table size used by `Abassi_CORTEXM4_ATOLLIC.s` can also be set through the GUI, in the "*C/C++ Build → Settings → Tool Setting → Assembler → Symbols*" menu, as shown in the following figure:

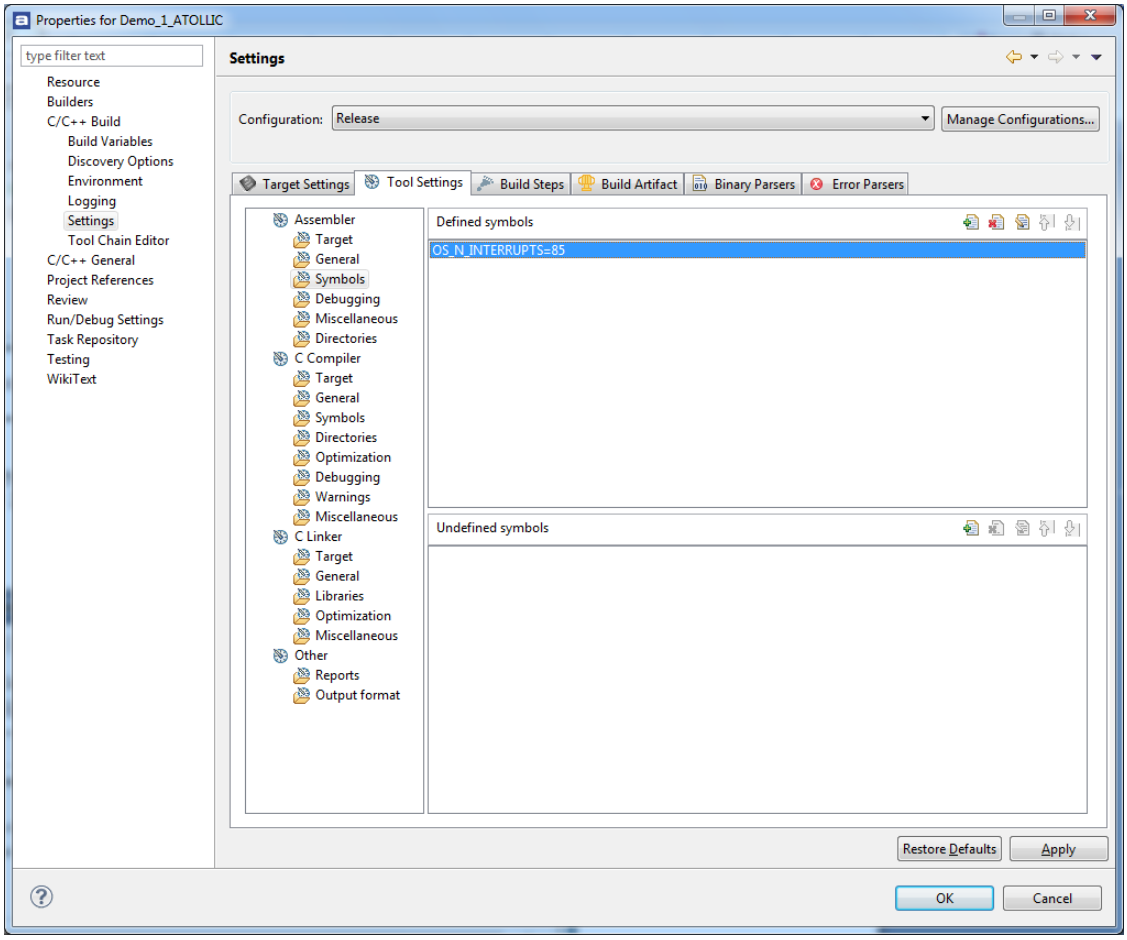

Figure 3-1 GUI set of `OS_N_INTERRUPTS`

The interrupt table look-up size used by `Abassi.c` can also be overloaded through the GUI, in the "*C/C++ Build → Settings → Tool Setting → C Compiler → Symbols*" menu, as shown in the following figure:

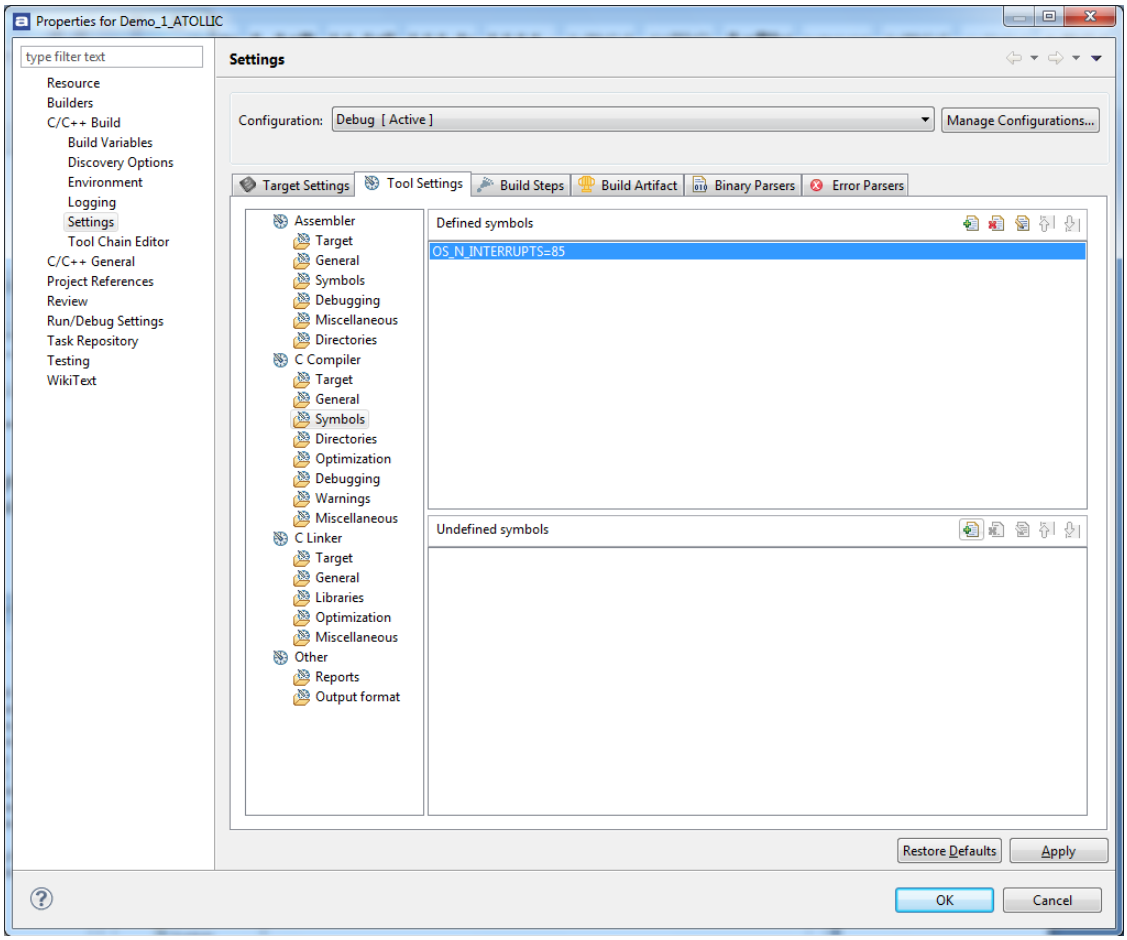

Figure 3-2 GUI set of `OS_N_INTERRUPTS`

3.1.2 Interrupt Installer

Attaching a function to a regular interrupt is quite straightforward. All there is to do is use the RTOS component `OSisrInstall()` to specify the interrupt number and the function to be attached to that interrupt number. For example, Table 3-4 shows the code required to attach the `SysTick` interrupt to the RTOS timer tick handler (`TIMtick`):

Table 3-4 Attaching a Function to an Interrupt

```
#include "Abassi.h"

  …
  OSstart();
  …
  OSisrInstall(-1, &TIMtick);
  /* Set-up the count reload and enable SysTick interrupt */

  … /* More ISR setup */

  OSeint(1);                              /* Global enable of all interrupts        */
```

NOTE: `OSisrInstall()` uses the interrupt number, NOT the interrupt vector number.

At start-up, once `OSstart()` has been called, all `OS_N_INTERRUPTS` interrupt handler functions are set to a "do nothing" function, named `OSinvalidISR()`. If an interrupt function is attached to an interrupt number using the `OSisrInstall()` component before calling `OSstart()`, this attachment will be removed by `OSstart()`, so `OSisrInstall()` should never be used before `OSstart()` has ran. When an interrupt handler is removed, it is very important and necessary to first disable the interrupt source, then the handling function can be set back to `OSinvalidISR()`. This is shown in Table 3-5:

Table 3-5 Invalidating an ISR handler

```
#include "Abassi.h"

  …
  /* Disable the interrupt source */
  OSisrInstall(Number, &OSinvalidISR);
  …
```

When an application needs to disable / enable the interrupts, the RTOS supplied functions `OSdint()` and `OSeint()` should be used.

The Nested Vectored Interrupt Controller (NVIC) on the Cortex-M4 does not clear the interrupt generated by a peripheral; neither does the RTOS. If the generated interrupt is a pulse (as for the `SysTick` interrupt), there is nothing to do to clear the interrupt request. However, if the generated interrupt is a level interrupt, the peripheral generating the interrupt must be informed to remove the interrupt request. This operation must be performed in the interrupt handler otherwise the interrupt will be re-entered over and over.

3.2 Interrupt Priority and Enabling

To properly configure interrupts, the interrupt priority must be set, and the peripheral configured to generate interrupts and enable them. There is no software provided to perform these operations, as this functionality is already available. First, Atollic supports the Cortex Microcontroller Software Interface Standard (CMSIS), which provides everything required to program the processor peripherals. Second, most chip manufacturers provide code to configure the specifics on their devices.

3.3 Fast Interrupts

Fast interrupts are supported on this port. A fast interrupt is an interrupt that never uses any component from Abassi, and as the name says, is desired to operate as fast as possible. To set-up a fast interrupt, all there is to do is to set the address of the interrupt function in the corresponding entry in the interrupt vector table used by the Cortex-M4 processor. The area of the interrupt vector table to modify is located in the file `Abassi_CORTEXM4_ATOLLIC.s` around line 100.

For example, on a ST Microelectronics STM32F407 device, UART #1 is attached to interrupt number 37 (interrupt vector number 53) and the UART #2 is attached to the interrupt number 38 (interrupt vector number 54). The code to modify is located in the macro loop that initializes the interrupt table that sets the ISR dispatcher as the default interrupt handler. All there is to do is add checks on the token holding the interrupt number, such that, when the interrupt number value matches the desired interrupt number, the appropriate address gets inserted in the table instead of the address of `ISRdispatch()`. The original macro loop code and modified one are shown in the following two tables:

Table 3-6 Distribution interrupt table code

```
.set  INT_NMB, -1
.rept OS_N_INTERRUPTS            /* Map all external interrupts to ISRdispatch() */
   .word   ISRdispatch
   .set    INT_NMB, INT_NMB+1
.endr
```

Attaching a fast interrupt handler to the UART #1 and another one to UART#2, assuming the names of the interrupt functions to attach are respectively `UART1_IRQhandler()` and `UART2_IRQhandler()` is shown in Table 3-7:

Table 3-7 STM32F407 UART 1 / 2 Fast Interrupts

```
 .global  USART0_IRQhandler
 .global  USART1_IRQhandler

 …

.set  INT_NMB, -1
.rept OS_N_INTERRUPTS            /* Map all external interrupts to ISRdispatch() */
   .if INT_NMB == 5              /* When is interrupt # 5, set UART #0 handler   */
        .word  USART0_IRQhandler
   .elseif INT_NMB == 6          /* When is interrupt # 6, set UART #1 handler   */
        .word  USART1_IRQhandler
   .else                         /* All others interrupt # set to ISRdispatch()  */
        .word   ISRdispatch
   .endif
   .set    INT_NMB, INT_NMB+1
.endr

 …
```

It is important to add the `EXTERN` statement, otherwise there will be an error during the assembly of the file.

NOTE: If an Abassi component is used inside a fast interrupt, the application will misbehave.

Even if the hybrid interrupt stack feature is enabled (see Section 2.1), fast interrupts will not use that stack. This translates into the need to reserve room on all task stacks for the possible nesting of fast interrupts. To make the fast interrupts also use a hybrid interrupt stack, a prologue and epilogue must be used around the call to the interrupt handler. The prologue and epilogue code to add is almost identical to what is done in the regular interrupt dispatcher. Reusing the example of the UART #1 on the STM32F407 device, this would look something like:

Table 3-8 Fast Interrupt with Dedicated Stack

```
    …

    .if INT_NMB == 5              /* When is interrupt # 5, set UART #0 handler   */
        .word    UART0preHandler

    …
    …

    .section .text.UART0preHandler
    .align2
    .code   16
    .thumb_func
    .type   OScontext, %function

    EXTERN  UART0handler

UART0preHandler:
    cpsid   I                           /* Disable ISR to protect against nesting */
    mov     r0, sp                      /* Memo current stack pointer             */
    ldr     sp, =UART0_stack            /* Stack dedicated to this fast interrupt */
    cpsie   I                           /* The stack is now hybrid, nesting safe  */
    push    {r0, lr}                    /* Preserve original sp & EXC_RETURN      */

    bl      UART0handler                /* Enter the interrupt handler            */

    pop     {r0, lr}                    /* Recover original sp & EXC_RETURN       */
    mov     sp, r0                      /* Recover pre-isr stack                  */
    bx      lr                          /* Exit from the interrupt                */

    …
    …

    .bss

    .space    UART0_stack_size           /* Room for the fast interrupt stack     */
UART0_stack:

    …
```

The same code, with unique labels, must be repeated for each of the fast interrupts.

3.4 Nested Interrupts

The interrupt controller allows nesting of interrupts; this means an interrupt of higher priority will interrupt the processing of an interrupt of lower priority. Individual interrupt sources can be set to one of 8 levels, where level 0 is the highest and 7 is the lowest. This implies that the RTOS build option `OS_NESTED_INTS` must be set to a non-zero value. The exception to this is an application where all enabled interrupts handled by the RTOS ISR dispatcher are set, without exception, to the same priority; then interrupt nesting will not occur. In that case, and only that case, can the build option `OS_NESTED_INTS` be set to zero. As this latter case is quite unlikely, the build option `OS_NESTED_INTS` is always overloaded when compiling the RTOS for the ARM Cortex-M4. If the latter condition is guaranteed, the overloading located after the pre-processor directive can be modified. The code affected in `Abassi.h` is shown in Table 3-9 below and the line to modify is the one with `#define OX_NESTED_INTS 1`:

Table 3-9 Removing interrupt nesting

```
#elif defined(__GNUC__)    \
   && (defined(__ARM_ARCH_6M__) || defined(__ARM_ARCH_7M__) ||
defined(__ARM_ARCH_7EM__))

  #define OX_NESTED_INTS 0 /* The ARM has 8 nested (NIVC) interrupt levels */
```

Or if the build option `OS_NESTED_INTS` is desired to be propagated:

Table 3-10 Propagating interrupt nesting

```
#elif defined(__GNUC__)    \
   && (defined(__ARM_ARCH_6M__) || defined(__ARM_ARCH_7M__) ||
defined(__ARM_ARCH_7EM__))
  #define OX_NESTED_INTS OS_NESTED_INTS
```

The Abassi RTOS kernel never disables interrupts, but there is a few very small regions within the interrupt dispatcher where interrupts are temporarily disabled due to the nesting (a total of between 10 to 20 instructions).

The kernel is never entered as long as interrupt nesting exists. In all interrupt functions, when a RTOS component that needs to access some kernel functionality is used, the request(s) is/are put in a queue. Only once the interrupt nesting is over (i.e. when only a single interrupt context remains) is the kernel entered at the end of the interrupt, when the queue contains one or more requests, and when the kernel is not already active. This means that only the interrupt handler function operates in an interrupt context, and only the time the interrupt function is using the CPU are other interrupts of equal or lower level blocked by the interrupt controller.

4 Stack Usage

The RTOS uses the tasks' stack for two purposes. When a task is blocked or ready to run but not running, the stack holds the register context that was preserved when the task got blocked or preempted. Also, when an interrupt occurs, the register context of the running task must be preserved in order for the operations performed during the interrupt to not corrupt the contents of the registers used by the task when it got interrupted. For the Cortex-M4, the context save contents of a blocked or pre-empted task is different from the one used in an interrupt, and is also different if the compiler is set to use the FPU or not. The following table lists the number of bytes required by each type of context save operation:

Table 4-1 Context Save Stack Requirements

Description	Context save
Blocked/Preempted task context save (FPU OFF)	40 bytes
Interrupt dispatcher context save (`OS_ISR_STACK == 0`) (FPU OFF)	40 bytes
Interrupt dispatcher context save (`OS_ISR_STACK != 0`) (FPU OFF)	48 bytes
Blocked/Preempted task context save (FPU ON)	112 bytes
Interrupt dispatcher context save (`OS_ISR_STACK == 0`) (FPU ON)	120 bytes
Interrupt dispatcher context save (`OS_ISR_STACK != 0`) (FPU ON)	128 bytes

The numbers for the interrupt dispatcher context save include the 32 bytes (FPU OFF) or the 96 bytes (FPU ON) the processor pushes on the stack when it enters the interrupt servicing.

When sizing the stack to allocate to a task, there are three factors to take in account. The first factor is simply that every task in the application needs at least the area to preserve the task context when it is preempted or blocked. Second, one must take into account how many levels of nested interrupts exist in the application. As a worst case, all levels of interrupts may occur and becoming fully nested. So if N levels of interrupts are used in the application, provision should be made to hold N times the size of an ISR context save on each task stack, plus any added stack used by all the interrupt handler functions. Finally, add to all this the stack required by the code implementing the task operation.

NOTE: The ARM Cortex-M4 processor needs alignment on 8 bytes for some instructions accessing memory. When stack memory is allocated, Abassi guarantees the alignment. This said, when sizing `OS_STATIC_STACK` or `OS_ALLOC_SIZE`, make sure to take in account that all allocation performed through these memory pools are by block size multiple of 8 bytes.

If the hybrid interrupt stack (see Section 2.1) is enabled, then the above description changes: it is only necessary to reserve room on task stacks for a single interrupt context save (this excludes the interrupt function handler stack requirements) and not the worst-case nesting. With the hybrid stack enabled, the second, third, and so on interrupts use the stack dedicated to the interrupts. The hybrid stack is enabled when the `OS_ISR_STACK` token in the file `Abassi_CORTEXM4_ATOLLIC.s` is set to a non-zero value (see Section 2.1).

5 Search Set-up

The Abassi RTOS build option `OS_SEARCH_FAST` offers three different algorithms to quickly determine the next running task upon task blocking. The following table shows the measurements obtained for the number of CPU cycles required when a task at priority 0 is blocked, and the next running task is at the specified priority. The number of cycles includes everything, not just the search cycle count. The number of cycles was measured using the `SysTick` peripheral, which decrements the counter once every CPU cycle. The second column is when `OS_SEARCH_FAST` is set to zero, meaning a simple array traversing. The third column, labeled Look-up, is when `OS_SEARCH_FAST` is set to 1, which uses an 8 bit look-up table. Finally, the last column is when `OS_SEARCH_FAST` is set to 5 (Atollic/Cortex-M4 `int` are 32 bits, so 2^5), meaning a 32 bit look-up table, further searched through successive approximation. The compiler optimization for this measurement was set to Level High / Speed optimization. The RTOS build options were set to the minimum feature set, except for option `OS_PRIO_CHANGE` set to non-zero. The presence of this extra feature provokes a small mismatch between the result for a difference of priority of 1, with `OS_SEARCH_FAST` set to zero, and the latency results in Section 7.2.

When the build option `OS_SEARCH_ALGO` is set to a negative value, indicating to use a 2-dimensional linked list search technique instead of the search array, the number of CPU cycles is constant at 275 cycles.

Table 5-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	273	311	325
2	283	316	325
3	289	321	325
4	295	326	325
5	301	331	325
6	307	336	325
7	313	341	325
8	319	317	325
9	325	325	325
10	332	330	325
11	337	335	325
12	343	340	325
13	349	345	325
14	355	350	325
15	361	355	326
16	367	323	326
17	373	331	326
18	379	336	326
19	385	341	326
20	391	346	326
21	397	351	326
22	403	356	326
23	409	361	326
24	415	333	326

When `OS_SEARCH_FAST` is set to 0, each extra priority level to traverse requires exactly 6 CPU cycles. When `OS_SEARCH_FAST` is set to 1, each extra priority level to traverse requires exactly 5 CPU cycles, except when the priority level is an exact multiple of 8; then there is a sharp reduction of CPU usage. Overall, setting `OS_SEARCH_FAST` to 1 adds around 40 cycles of CPU for the search compared to setting `OS_SEARCH_FAST` to zero. But when the next ready to run priority is less than 8, 16, 24, … then there is an extra number of cycles needed, but without the 8 times 8 cycle accumulation. Finally, the third option, when `OS_SEARCH_FAST` is set to 5, delivers a quasi-perfectly constant CPU usage, as the algorithm utilizes a successive approximation search technique (when the delta is 32 or more, the CPU cycle count is 336/337, for 64 or more, it is 343/344).

The first observation, when looking at this table, is that the second option, when OS_SEARCH_FAST is set to 1, is either less CPU efficient than the first option, the one when OS_SEARCH_FAST is set to 0, or less efficient than the third option OS_SEARCH_FAST is set to 5. So, the build option OS_SEARCH_FAST should never be set to 1, as it is the least efficient method. The other observation is that the first option (OS_SEARCH_FAST set to 0) delivers better CPU performance than the third option (OS_SEARCH_FAST set to 5) when the search spans less than 8 to 9 priority levels. So, if an application has tasks spanning less than 8 to 9 priority levels, the build option OS_SEARCH_FAST should be set to 0; for all other cases, the build option OS_SEARCH_FAST should be set to 5.

Setting the build option OS_SEARCH_ALGO to a non-negative value minimizes the time needed to change the state of a task from blocked to ready to run, and not the time needed to find the next running task upon blocking/suspending of the running task. If the application needs are such that the critical real-time requirement is to get the next running task up and running as fast as possible, then set the build option OS_SEARCH_ALGO to a negative value.

6 Chip Support

No chip support is provided with the distribution code because Atollic TrueSTUDIO for ARM supports the Cortex Microcontroller Software Interface Standard (CMSIS). Therefore, all peripherals on the Cortex-M4 can be accessed through the CMSIS. Also, most device manufacturers provide code to configure the peripherals on their devices.

7 Measurements

This section gives an overview of the memory requirements and the CPU latency encountered when the RTOS is used on the ARM Cortex-M4 and compiled with Atollic TrueSTUDIO for ARM. The CPU cycles are exactly the CPU clock cycles, as the processor typically executes one instruction at every clock transition.

7.1 Memory

The memory numbers are supplied for the two limit cases of build options (and some in-between): the smallest footprint is the RTOS built with only the minimal feature set, and the other with almost all the features. For both cases, names are not part of the build. This feature was removed from the metrics because it is highly probable that shipping products utilizing this RTOS will not include the naming of descriptors, as its usefulness is mainly limited to debugging and making the opening/creation of components run-time safe.

The code size numbers are expressed with “less than” as they have been rounded up to multiples of 25 for the “C” code. These numbers were obtained using the beta release of the RTOS and may change. One should interpret these numbers as the “very likely” numbers for the released version of the RTOS.

The code memory required by the RTOS includes the "C" code and assembly language code used by the RTOS. The code optimization settings of the compiler that were used for the memory measurements are:

1. Optimization Level: Optimize for speed (`-Ofast`)
2. Prepare dead code removal: Enabled
3. Prepare dead data removal: Enabled

All other options are disabled as they do not affect the code generated.

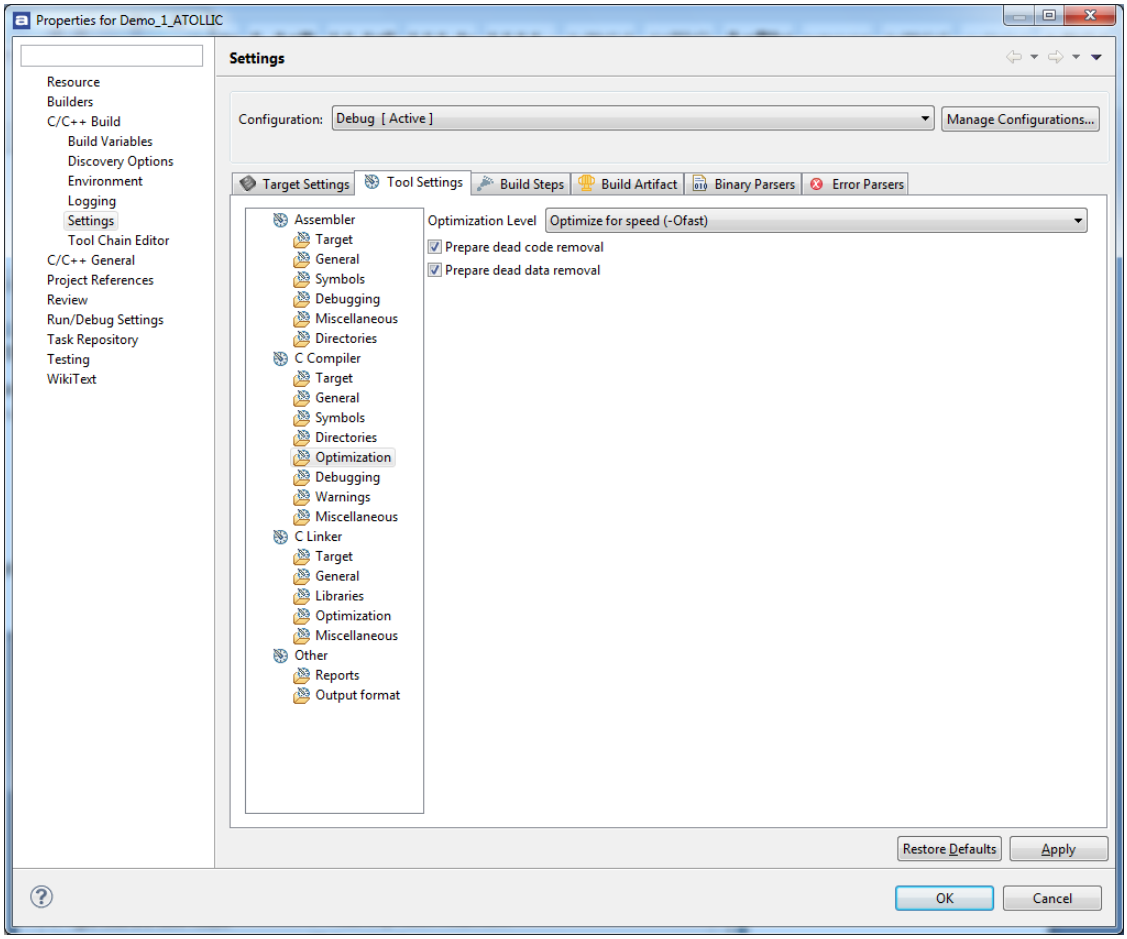

Figure 7-1 Memory Measurement Code Optimization Settings

Table 7-1 "C" Code Memory Usage

Description	Code Size
Minimal Build	< 675 bytes
+ Runtime service creation / static memory	< 925 bytes
+ Multiple tasks at same priority	< 1000 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 1475 bytes
+ Timer & timeout + Timer call back + Round robin	< 2175 bytes
+ Events + Mailbox	< 2975 bytes
Full Feature Build (no names)	< 3650 bytes
Full Feature Build (no names / no runtime creation)	< 3275 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 3225 bytes

Table 7-2 "C" library multi-threading protection

Description	Size
OS_ATOLLIC_REENT < 0	+180 bytes
OS_ATOLLIC_REENT > 0	+304 bytes

Table 7-3 Assembly Code Memory Usage

Description	Size
Assembly code size (FPU OFF)	248 bytes
Assembly code size (FPU ON)	312 bytes
Vector table (per interrupt handler entry)	+4 bytes
Hybrid Stack Enabled	+32 bytes
Saturation Bit Enabled	+44 bytes
FPU runtime ON / OFF	+232 bytes
OS_ATOLLIC_REENT < 0	+20 bytes
OS_ATOLLIC_REENT > 0	+8 bytes

There are two aspects when describing the data memory usage by the RTOS. First, the RTOS needs its own data memory to operate, and second, most of the services offered by the RTOS require data memory for each instance of the service. As the build options affect either the kernel memory needs or the service descriptors (or both), an interactive calculator has been made available on Code Time Technologies website.

7.2 Latency

Latency of operations has been measured on an Olimex STM32-P407 Evaluation board populated with a 168 MHz STM32F407 device. The clock setting for the measurement used the internal oscillator operating at 16 MHz, which allows running from the flash with 0 wait states. All measurements have been performed on the real platform. This means the interrupt latency measurements had to be instrumented to read the `SysTick` counter value. This instrumentation can add up to 5 or 6 cycles to the measurements. The code optimization settings that were used for the latency measurements are:

1. Optimization Level: Optimize for size (`-Os`)
2. Prepare dead code removal: Enabled
3. Prepare dead data removal: Enabled

All other options are disabled, as they do not affect the efficiency of the code generated.

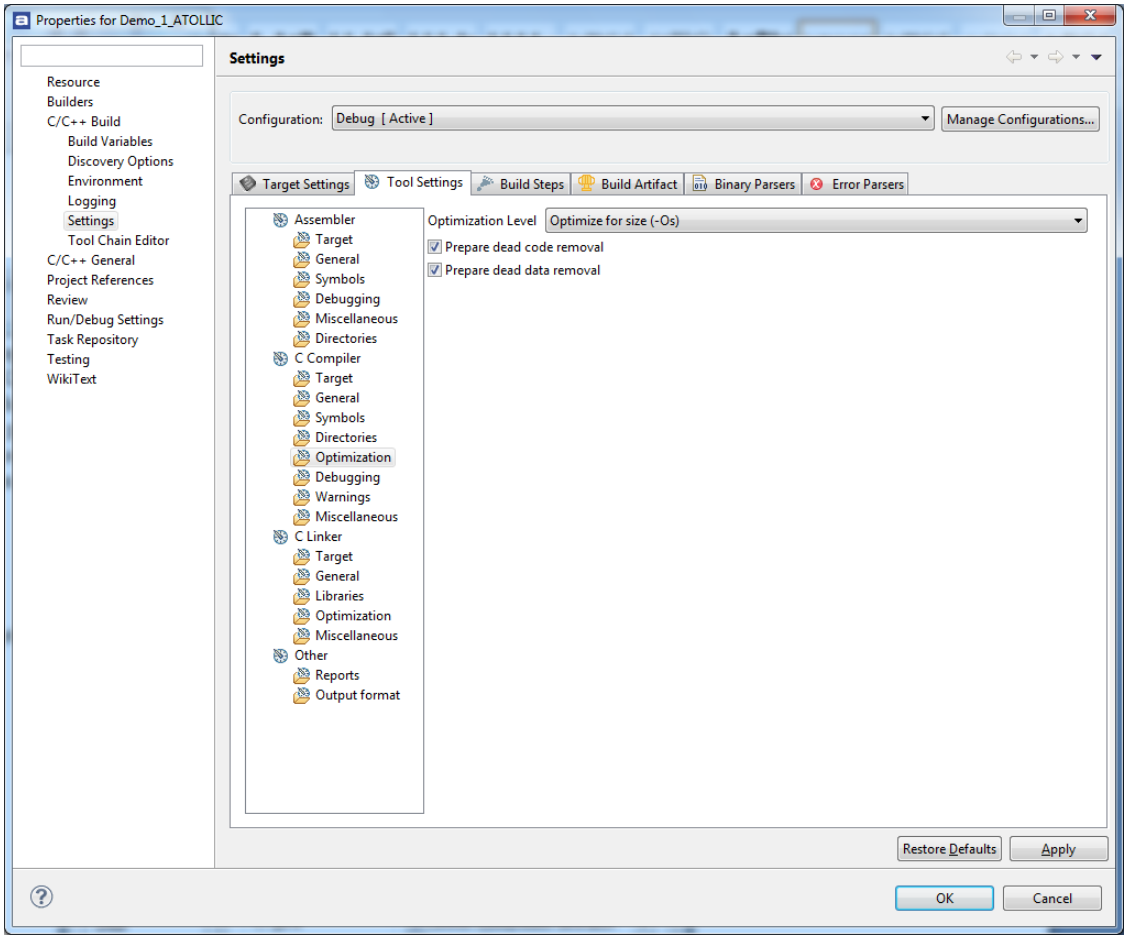

Figure 7-2 Latency Measurement Code Optimization Settings

There are 5 types of latencies that are measured, and these 5 measurements are expected to give a very good overview of the real-time performance of the Abassi RTOS for this port. For all measurements, three tasks were involved:

1. Adam & Eve set to a priority value of 0;
2. A low priority task set to a priority value of 1;
3. The Idle task set to a priority value of 20.

The sets of 5 measurements are performed on a semaphore, on the event flags of a task, and finally on a mailbox. The first 2 latency measurements use the component in a manner where there is no task switching. The third measurements involve a high priority task getting blocked by the component. The fourth measurements are about the opposite: a low priority task getting pre-empted because the component unblocks a high priority task. Finally, the reaction to unblocking a task, which becomes the running task, through an interrupt is provided.

The first set of measurements counts the number of CPU cycles elapsed starting right before the component is used until it is back from the component. For these measurement there is no task switching. This means:

Table 7-4 Measurement without Task Switch

```
Start CPU cycle count
SEMpost(…);  or  EVTset(…);  or  MBXput();
Stop CPU cycle count
```

The second set of measurements, as for the first set, counts the number of CPU cycles elapsed starting right before the component is used until it is back from the component. For these measurement there is no task switching. This means:

Table 7-5 Measurement without Blocking

```
Start CPU cycle count
SEMwait(…, -1);  or  EVTwait(…, -1);  or  MBXget(…, -1);
Stop CPU cycle count
```

The third set of measurements counts the number of CPU cycles elapsed starting right before the component triggers the unblocking of a higher priority task until the latter is back from the component used that blocked the task. This means:

Table 7-6 Measurement with Task Switch

```
main()
{
    …
    SEMwait(…, -1);  or  EVTwait(…, -1);  or  MBXget(…, -1);
    Stop CPU cycle count
    …
}

TaskPrio1()
{
    …
    Start CPU cycle count
    SEMpost(…);      or  EVTset(…);       or  MBXput(…);
    …
}
```

The forth set of measurements counts the number of CPU cycles elapsed starting right before the component blocks of a high priority task until the next ready to run task is back from the component it was blocked on; the blocking was provoked by the unblocking of a higher priority task. This means:

Table 7-7 Measurement with Task unblocking

```
main()
{
    …
    Start CPU cycle count
    SEMwait(…, -1);  or  EVTwait(…, -1);  or  MBXget(…, -1);
    …
}

TaskPrio1()
{
    …
    SEMpost(…);      or  EVTset(…);       or  MBXput(…);
    Stop CPU cycle count
    …
}
```

The fifth set of measurements counts the number of CPU cycles elapsed from the beginning of an interrupt using the component, until the task that was blocked becomes the running task and is back from the component used that blocked the task. The interrupt latency measurement includes everything involved in the interrupt operation, even the cycles the processor needs to push the interrupt context before entering the interrupt code. The interrupt function, attached with `OSisrInstall()`, is simply a two line function that uses the appropriate RTOS component followed by a return.

Table 7-8 lists the results obtained, where the cycle count is measured using the `SysTick` peripheral on the Cortex-M4. This timer decrements its counter by 1 at every CPU cycle. As was the case for the memory measurements, these numbers were obtained with a beta release of the RTOS. It is possible the released version of the RTOS may have slightly different numbers.

The interrupt latency is the number of cycles elapsed when the interrupt trigger occurred and the ISR function handler is entered. This includes the number of cycles used by the processor to set-up the interrupt stack and branch to the address specified in the interrupt vector table. But for this measurement, the STM32F407 Systick Timer is used to trigger the interrupt and measure the elapsed time. The latency measurement includes the cycles required to acknowledge the interrupt.

The interrupt overhead without entering the kernel is the measurement of the number of CPU cycles used between the entry point in the interrupt vector and the return from interrupt, with a "do nothing" function in the `OSisrInstall()`. The interrupt overhead when entering the kernel is calculated using the results from the third and fifth tests. Finally, the OS context switch is the measurement of the number of CPU cycles it takes to perform a task switch, without involving the wrap-around code of the synchronization component.

The hybrid interrupt stack feature was not enabled, neither was the saturation bit, in any of these tests. When the FPU is on, the runtime FPU ON / OFF feature of Abassi is not enabled. The library re-entrance and multi-thread protection are not enable.

In the following two tables, the latency numbers between parentheses are the measurements when the build option `OS_SEARCH_ALGO` is set to a negative value. The regular number is the latency measurements when the build option `OS_SEARCH_ALGO` is set to 0.

Table 7-8 Latency Measurements FPU OFF

Description	Minimal Features	Full Features
Semaphore posting no task switch	118 (128)	201 (205)
Semaphore waiting no blocking	122 (131)	210 (216)
Semaphore posting with task switch	182 (209)	317 (341)
Semaphore waiting with blocking	199 (208)	352 (355)
Semaphore posting in ISR with task switch	367 (395)	506 (528)
Event setting no task switch	n/a	198 (205)
Event getting no blocking	n/a	218 (223)
Event setting with task switch	n/a	330 (357)
Event getting with blocking	n/a	368 (371)
Event setting in ISR with task switch	n/a	521 (546)
Mailbox writing no task switch	n/a	249 (256)
Mailbox reading no blocking	n/a	257 (265)
Mailbox writing with task switch	n/a	363 (388)
Mailbox reading with blocking	n/a	411 (416)
Mailbox writing in ISR with task switch	n/a	563 (585)
Interrupt Latency	35	35
Interrupt overhead entering the kernel	185 (186)	189 (187)
Interrupt overhead NOT entering the kernel	56	56
Context switch	35	44

Table 7-9 Latency Measurements FPU ON

Description	Minimal Features	Full Features
Semaphore posting no task switch	118 (128)	201 (205)
Semaphore waiting no blocking	122 (131)	210 (216)
Semaphore posting with task switch	226 (253)	361 (385)
Semaphore waiting with blocking	243 (252)	396 (399)
Semaphore posting in ISR with task switch	439 (467)	578 (600)
Event setting no task switch	n/a	198 (205)
Event getting no blocking	n/a	218 (223)
Event setting with task switch	n/a	374 (401)
Event getting with blocking	n/a	412 (415)
Event setting in ISR with task switch	n/a	593 (618)
Mailbox writing no task switch	n/a	249 (256)
Mailbox reading no blocking	n/a	257 (265)
Mailbox writing with task switch	n/a	407 (432)
Mailbox reading with blocking	n/a	455 (460)
Mailbox writing in ISR with task switch	n/a	635 (657)
Interrupt Latency	47	47
Interrupt overhead entering the kernel	213 (214)	217 (215)
Interrupt overhead NOT entering the kernel	88	88
Context switch	81	90

8 Appendix A: Build Options for Code Size

8.1 Case 0: Minimum build

Table 8-1: Case 0 build options

```
#define OS_ALLOC_SIZE       0    /* When !=0, RTOS supplied OSalloc                  */
#define OS_COOPERATIVE      0    /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS           0    /* If event flags are supported                     */
#define OS_FCFS             0    /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK       0    /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE     0    /* Type of logging to use                           */
#define OS_MAILBOX          0    /* If mailboxes are used                            */
#define OS_MAX_PEND_RQST    2    /* Maximum number of requests in ISRs               */
#define OS_MTX_DEADLOCK     0    /* This test validates this                         */
#define OS_MTX_INVERSION    0    /* To enable protection against priority inversion  */
#define OS_NAMES            0    /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS      0    /* If operating with nested interrupts              */
#define OS_PRIO_CHANGE      0    /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN         2    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME        0    /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN      0    /* Use round-robin, value specifies period in uS    */
#define OS_RUNTIME          0    /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO      0    /* If using a fast search                           */
#define OS_STARVE_PRIO      0    /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX   0    /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX  0    /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX   0    /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX       0    /* If !=0 how many mailboxes                        */
#define OS_STATIC_NAME      0    /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM       0    /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK     0    /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK      0    /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND     0    /* If a task can suspend another one                */
#define OS_TIMEOUT          0    /* !=0 enables blocking timeout                     */
#define OS_TIMER_CB         0    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module           */
#define OS_TIMER_US         0    /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG     0    /* If tasks have arguments                          */
```

8.2 Case 1: + Runtime service creation / static memory

Table 8-2: Case 1 build options

```
#define OS_ALLOC_SIZE       0    /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE      0    /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS           0    /* If event flags are supported                      */
#define OS_FCFS             0    /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK       0    /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE     0    /* Type of logging to use                            */
#define OS_MAILBOX          0    /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST    2    /* Maximum number of requests in ISRs                */
#define OS_MTX_DEADLOCK     0    /* This test validates this                          */
#define OS_MTX_INVERSION    0    /* To enable protection against priority inversion   */
#define OS_NAMES            0    /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS      0    /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE      0    /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN         2    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME        0    /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN      0    /* Use round-robin, value specifies period in uS     */
#define OS_RUNTIME          1    /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO      0    /* If using a fast search                            */
#define OS_STARVE_PRIO      0    /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX   0    /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX  0    /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX   0    /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX       0    /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME      0    /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM       5    /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK     128  /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK      5    /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND     0    /* If a task can suspend another one                 */
#define OS_TIMEOUT          0    /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB         0    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module            */
#define OS_TIMER_US         0    /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG     0    /* If tasks have arguments                           */
```

8.3 Case 2: + Multiple tasks at same priority

Table 8-3: Case 2 build options

```
#define OS_ALLOC_SIZE       0    /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE      0    /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS           0    /* If event flags are supported                      */
#define OS_FCFS             0    /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK       0    /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE     0    /* Type of logging to use                            */
#define OS_MAILBOX          0    /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST    32   /* Maximum number of requests in ISRs                */
#define OS_MTX_DEADLOCK     0    /* This test validates this                          */
#define OS_MTX_INVERSION    0    /* To enable protection against priority inversion   */
#define OS_NAMES            0    /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS      0    /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE      0    /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN         20   /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME        1    /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN      0    /* Use round-robin, value specifies period in uS     */
#define OS_RUNTIME          1    /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO      0    /* If using a fast search                            */
#define OS_STARVE_PRIO      0    /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX   0    /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX  0    /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX   0    /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX       0    /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME      0    /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM       5    /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK     128  /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK      5    /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND     0    /* If a task can suspend another one                 */
#define OS_TIMEOUT          0    /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB         0    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module            */
#define OS_TIMER_US         0    /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG     0    /* If tasks have arguments                           */
```

8.4 Case 3: + Priority change / Priority inheritance / FCFS / Task suspend

Table 8-4: Case 3 build options

```
#define OS_ALLOC_SIZE       0    /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE      0    /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS           0    /* If event flags are supported                      */
#define OS_FCFS             1    /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK       0    /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE     0    /* Type of logging to use                            */
#define OS_MAILBOX          0    /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST    32   /* Maximum number of requests in ISRs                */
#define OS_MTX_DEADLOCK     0    /* This test validates this                          */
#define OS_MTX_INVERSION    1    /* To enable protection against priority inversion   */
#define OS_NAMES            0    /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS      0    /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE      1    /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN         20   /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME        1    /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN      0    /* Use round-robin, value specifies period in uS     */
#define OS_RUNTIME          1    /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO      0    /* If using a fast search                            */
#define OS_STARVE_PRIO      0    /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX   0    /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX  0    /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX   0    /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX       0    /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME      0    /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM       5    /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK     128  /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK      5    /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND     1    /* If a task can suspend another one                 */
#define OS_TIMEOUT          0    /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB         0    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module            */
#define OS_TIMER_US         0    /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG     0    /* If tasks have arguments                           */
```

8.5 Case 4: + Timer & timeout / Timer call back / Round robin

Table 8-5: Case 4 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS          0     /* If event flags are supported                      */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                            */
#define OS_MAILBOX         0     /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                */
#define OS_MTX_DEADLOCK    0     /* This test validates this                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion   */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN     100000/* Use round-robin, value specifies period in uS     */
#define OS_RUNTIME         1     /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO     0     /* If using a fast search                            */
#define OS_STARVE_PRIO     0     /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX  0     /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX 0     /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX  0     /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX      0     /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                 */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module            */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG    0     /* If tasks have arguments                           */
```

8.6 Case 5: + Events / Mailboxes

Table 8-6: Case 5 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS          0     /* If event flags are supported                      */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                            */
#define OS_MAILBOX         0     /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                */
#define OS_MTX_DEADLOCK    0     /* This test validates this                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion   */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN     100000/* Use round-robin, value specifies period in uS     */
#define OS_RUNTIME         1     /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO     0     /* If using a fast search                            */
#define OS_STARVE_PRIO     0     /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX  0     /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX 0     /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX  0     /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX      0     /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                 */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module            */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG    0     /* If tasks have arguments                           */
```

8.7 Case 6: Full feature Build (no names)

Table 8-7: Case 6 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS          1     /* If event flags are supported                      */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                            */
#define OS_MAILBOX         1     /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                */
#define OS_MTX_DEADLOCK    0     /* This test validates this                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion   */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN     -100000 /* Use round-robin, value specifies period in uS   */
#define OS_RUNTIME         1     /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO     0     /* If using a fast search                            */
#define OS_STARVE_PRIO     -3    /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX  -10   /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX -100  /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX  100   /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX      2     /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                 */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module            */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG    1     /* If tasks have arguments                           */
```

8.8 Case 7: Full feature Build (no names / no runtime creation)

Table 8-8: Case 7 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                    */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS          1     /* If event flags are supported                       */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                             */
#define OS_MAILBOX         1     /* If mailboxes are used                              */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                 */
#define OS_MTX_DEADLOCK    0     /* This test validates this                           */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts                */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN     -100000 /* Use round-robin, value specifies period in uS  */
#define OS_RUNTIME         0     /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO     0     /* If using a fast search                             */
#define OS_STARVE_PRIO     -3    /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX  -10   /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX -100  /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX  0     /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX      0     /* If !=0 how many mailboxes                          */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM      0     /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK    0     /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK     0     /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                       */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module           */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG    1     /* If tasks have arguments                            */
```

8.9 Case 8: Full build adding the optional timer services

Table 8-9: Case 8 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS          1     /* If event flags are supported                      */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                            */
#define OS_MAILBOX         1     /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                */
#define OS_MTX_DEADLOCK    0     /* This test validates this                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion   */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN     -100000 /* Use round-robin, value specifies period in uS   */
#define OS_RUNTIME         0     /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO     0     /* If using a fast search                            */
#define OS_STARVE_PRIO     -3    /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX  -10   /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX -100  /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX  100   /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX      2     /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                 */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV       1     /* !=0 includes the timer services module            */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG    1     /* If tasks have arguments                           */
```