CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

ColdFire – IAR

Copyright Information

This document is copyright Code Time Technologies Inc. ©2012. All rights reserved. No part of this document may be reproduced or distributed in any form by any means, or stored in a database or retrieval system, without the written permission of Code Time Technologies Inc.
Code Time Technologies Inc. may have patents or pending applications covering the subject matter in this document. The furnishing of this document does not give you any license to these patents.

Disclaimer

Code Time Technologies Inc. provides this document "AS IS" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.
Code Time Technologies Inc. does not warrant that the contents of this document will meet your requirements or that the document is error-free. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. Code Time Technologies Inc. may make improvements and/or changes in the product(s) and/or program(s) described in the document at any time. This document does not imply a commitment by Code Time Technologies Inc. to supply or make generally available the product(s) described herein.

IAR Embedded Workbench is a trademark owned by IAR Systems AB. ColdFire and CodeWarrior are registered trademarks of Freescale Semiconductor, Inc. All other trademarks are the property of their respective owners.

Table of Contents

1 INTRODUCTION 6
1.1 DISTRIBUTION CONTENTS 6
1.2 LIMITATIONS 6
2 TARGET SET-UP 7
2.1 INTERRUPT STACK SET-UP 7
2.2 ASSEMBLY INCLUDE FILE 8
2.3 MAC / EMAC UNIT 9
3 INTERRUPTS 12
3.1 INTERRUPT HANDLING 12
3.1.1 First 64 Table Entries 12
3.1.2 Interrupt Table Size 12
3.1.3 OSeint() and OSdint() 14
3.1.4 Interrupt Installer 15
3.2 FAST INTERRUPTS 17
3.3 NESTED INTERRUPTS 19
4 STACK USAGE 20
5 SEARCH SET-UP 21
6 CHIP SUPPORT 24
7 MEASUREMENTS 25
7.1 MEMORY 25
7.2 LATENCY 27
8 APPENDIX A: BUILD OPTIONS FOR CODE SIZE 33
8.1 CASE 0: MINIMUM BUILD 33
8.2 CASE 1: + RUNTIME SERVICE CREATION / STATIC MEMORY 34
8.3 CASE 2: + MULTIPLE TASKS AT SAME PRIORITY 35
8.4 CASE 3: + PRIORITY CHANGE / PRIORITY INHERITANCE / FCFS / TASK SUSPEND 36
8.5 CASE 4: + TIMER & TIMEOUT / TIMER CALL BACK / ROUND ROBIN 37
8.6 CASE 5: + EVENTS / MAILBOXES 38
8.7 CASE 6: FULL FEATURE BUILD (NO NAMES) 39
8.8 CASE 7: FULL FEATURE BUILD (NO NAMES / NO RUNTIME CREATION) 40
8.9 CASE 8: FULL BUILD ADDING THE OPTIONAL TIMER SERVICES 41

List of Figures

FIGURE 2-1 PROJECT FILE LIST 7
FIGURE 2-2 GUI SET OF OS_ISR_STACK 8
FIGURE 2-3 GUI SET OF NO MAC/EMAC SUPPORT 9
FIGURE 2-4 GUI SET OF MAC SUPPORT 10
FIGURE 2-5 GUI SET OF EMAC SUPPORT 11
FIGURE 3-1 GUI SET OF INTERRUPT TABLE SIZING 13
FIGURE 3-2 GUI SET OF INTERRUPT TABLE SIZING 14
FIGURE 3-3 GUI SET OF OX_TIM_TICK_ACK 16
FIGURE 7-1 MEMORY MEASUREMENT CODE OPTIMIZATION SETTINGS 25
FIGURE 7-2 LATENCY MEASUREMENT CODE OPTIMIZATION SETTINGS 27

List of Tables

TABLE 1-1 DISTRIBUTION 6
TABLE 2-1 OS_ISR_STACK 7
TABLE 2-2 COMMAND LINE SET OF OS_ISR_STACK 8
TABLE 2-3 ASSEMBLY INCLUDE FILE 8
TABLE 2-4 ENABLING MAC REGISTER PROTECTION 9
TABLE 2-5 ENABLING EMAC REGISTER PROTECTION 10
TABLE 3-1 ABASSI_CF_IAR.S INTERRUPT TABLE SIZING 12
TABLE 3-2 COMMAND LINE SET THE INTERRUPT TABLE SIZE 64 13
TABLE 3-3 OVERLOADING THE INTERRUPT TABLE SIZING FOR ABASSI.C 13
TABLE 3-4 OSDINT() RETURN VALUE 14
TABLE 3-5 ATTACHING A FUNCTION TO AN INTERRUPT 15
TABLE 3-6 INVALIDATING AN ISR HANDLER 15
TABLE 3-7 DISTRIBUTION INTERRUPT TABLE CODE 17
TABLE 3-8 MCF52233 PIT 0 / 1 FAST INTERRUPTS 17
TABLE 3-9 FAST INTERRUPT WITH DEDICATED STACK 18
TABLE 3-10 REMOVING INTERRUPT NESTING 19
TABLE 3-11 PROPAGATING INTERRUPT NESTING 19
TABLE 4-1 CONTEXT SAVE STACK REQUIREMENTS 20
TABLE 5-1 SEARCH ALGORITHM CYCLE COUNT 22
TABLE 7-1 "C" CODE MEMORY USAGE 26
TABLE 7-2 ASSEMBLY CODE MEMORY USAGE 26
TABLE 7-3 MEASUREMENT WITHOUT TASK SWITCH 28
TABLE 7-4 MEASUREMENT WITHOUT BLOCKING 28
TABLE 7-5 MEASUREMENT WITH TASK SWITCH 28
TABLE 7-6 MEASUREMENT WITH TASK UNBLOCKING 29
TABLE 7-7 LATENCY MEASUREMENTS (NO MAC) 30
TABLE 7-8 LATENCY MEASUREMENTS (MAC) 31
TABLE 7-9 LATENCY MEASUREMENTS (EMAC / EMAC_B) 32
TABLE 8-1: CASE 0 BUILD OPTIONS 33
TABLE 8-2: CASE 1 BUILD OPTIONS 34
TABLE 8-3: CASE 2 BUILD OPTIONS 35
TABLE 8-4: CASE 3 BUILD OPTIONS 36
TABLE 8-5: CASE 4 BUILD OPTIONS 37
TABLE 8-6: CASE 5 BUILD OPTIONS 38
TABLE 8-7: CASE 6 BUILD OPTIONS 39
TABLE 8-8: CASE 7 BUILD OPTIONS 40
TABLE 8-9: CASE 8 BUILD OPTIONS 41

1 Introduction

This document details the port of the Abassi RTOS to the ColdFire processor line. The port is for the V1, V2 and V3 versions of the ColdFire core. The software suite used for this specific port is the IAR Embedded Workbench for ColdFire; the version used for the port and all tests is Version 1.23.4, packaged as Version 5.4.

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
`Abassi_CF_IAR.s`	RTOS assembly file for the ColdFire to use with the IAR Embedded Workbench
`Demo_0_M52233DEMO_IAR.c`	Demo code that runs on the Freescale M52233DEMO evaluation board
`Demo_3_M52233DEMO_IAR.c`	Demo code that runs on the Freescale M52233DEMO evaluation board
`Demo_7_M52233DEMO_IAR.c`	Demo code that runs on the Freescale M52233DEMO evaluation board
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

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

Only the `ISA A` and `ISA A+` instruction sets have been tested. The assembly file is correctly processed when the instruction set is either `ISA B` or `ISA C`, but the operation was not verified.

2 Target Set-up

Very little is needed to configure the IAR Embedded Workbench development environment to use the Abassi RTOS in an application. All there is to do is to add the files `Abassi.c` and `Abassi_CF_IAR.s` in the source files of the application project, and make sure the two configuration settings in the file `Abassi_CF_IAR.s` (`OS_ISR_STACK` as described in Section 2.1, and `OS_N_INTERRUPTS` as described in Section 3.1.2) are set according to the needs of the application. Also, an include file in `Abassi_CF_IAR.s` must be set to the target device (Section 2.2). 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, as the `Abassi_CF_IAR.s` file contains all the start-up operations.

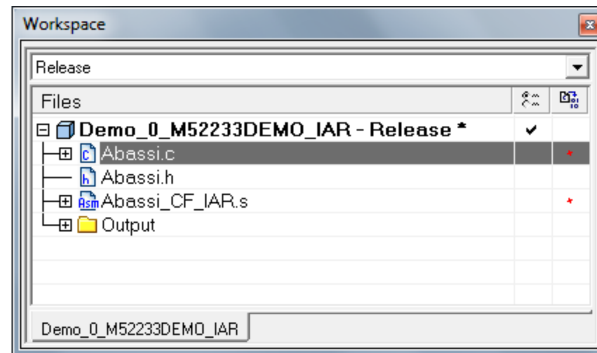

Figure 2-1 Project File List

NOTE: By default, some functions in the ColdFire IAR Embedded Workbench C/C++ run-time libraries are not multithread-safe. If these functions are only used in one task, then there is no problem. But if they are used by more than one task, they need to be protected by an Abassi mutex. The preferred way is to re-use the `G_OSmutex` mutex for all non-multithread-safe function, as this will avoid deadlocks. The list of non-reentrant functions is given in IAR C/C++ compiler documentation.

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 30 in the file `Abassi_CF_IAR.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
  #define 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_CF_IAR.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`

```
acf … -DOS_ISR_STACK=512 …
```

The hybrid stack size can also be set through the GUI, in the "*Assembler / Preprocessor*" menu, as shown in the following figure:

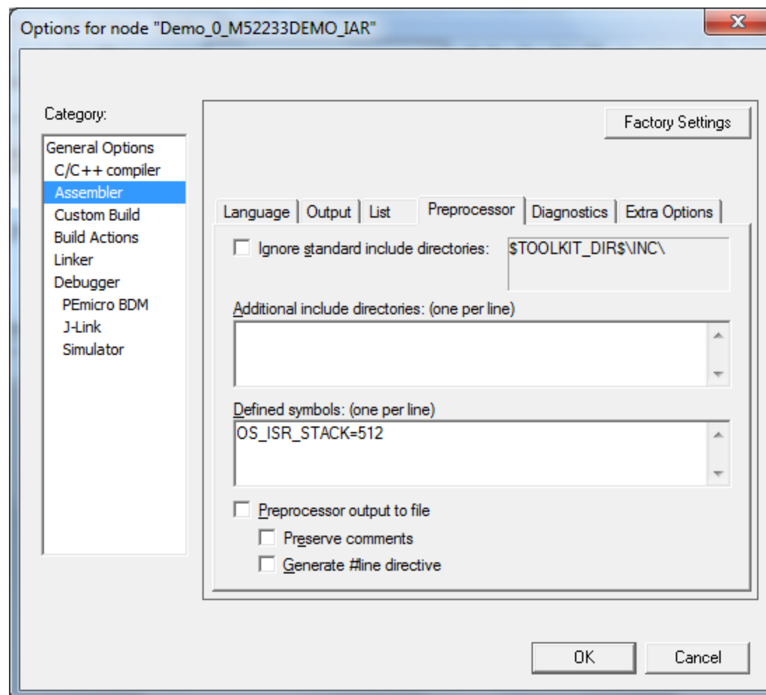

Figure 2-2 GUI set of `OS_ISR_STACK`

2.2 Assembly include file

The assembly file sets-up the 4 processor base address registers upon start-up. As the addresses of these registers are not always the same across different families of ColdFire devices, the assembly file obtains the register addresses by including a device definition file. The file inclusion is located around line 25 in `Abassi_CF_ISR.s`, as shown in the following:

Table 2-3 Assembly Include File

```
#include "io52233.h"              /* Replace with the target device defintion file */
```

Replace the file name (`io52233.h` here) with the one matching the target device.

2.3 MAC / EMAC unit

Some ColdFire devices have either the MAC (Multiply-Accumulate) unit, or the EMAC (Enhanced Multiply-Accumulate) unit, or none. The Abassi RTOS has provision to preserve the registers of the MAC or EMAC as part of a task context; the ISR dispatcher is also able to protect these registers. In the IAR compiler documentation it is stated that the command line option `--mac` does not have any effect. One would assume this means the MAC or EMAC is not used in the code generated by the compiler. Therefore, if the MAC or EMAC are not used through assembly language in an application, there is no need to protect the MAC / EMAC registers.

The file `Abassi_CF_IAR.s` obtains the information on the presence and type of MAC through the command line option `--mac`. When the GUI is used, the command line option is set through the menu "*General Options / Target / Instruction support*". So, if the MAC / EMAC is not used through assembly language, you should specify that no MAC / EMAC is used, as this will reduce the context save size, the context switch time, the interrupt stack needs, and the interrupt response time. The following figure shows how to remove the protection of the MAC / EMAC registers:

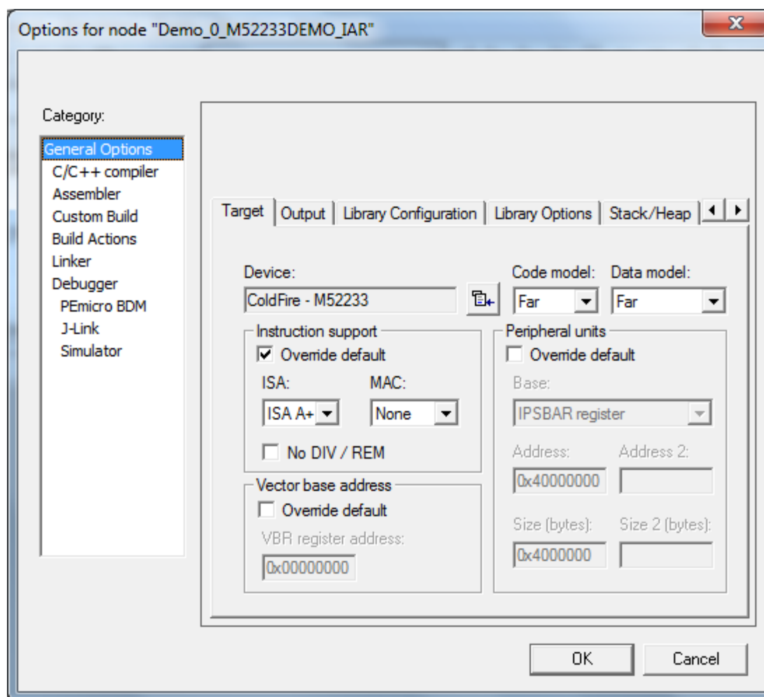

Figure 2-3 GUI set of No MAC/EMAC Support

When the assembler is used on the command line, do not specify the `--mac` option, and this will inform `Abassi_CF_IAR.s` to not protect the MAC / EMAC registers.

If the application accesses the MAC / EMAC through assembly code, it may not be necessary to inform `Abassi_CF_IAR.s` to protect the MAC / EMAC registers. If the MAC / EMAC are accessed in a single task only, or if they are accessed in a single interrupt handler, there is no need to protect the registers. Under any other conditions, the registers must be protected.

If the application uses a MAC unit, and its registers need to be multi-thread protected, inform `Abassi_CF_IAR.s` through the assembler command line as shown below:

Table 2-4 Enabling MAC register protection

```
acf … --mac=mac … Abassi_CF_IAR.s
```

Alternatively, this can be set through the GUI in the "*General Options / Target / Instruction support*" menu:

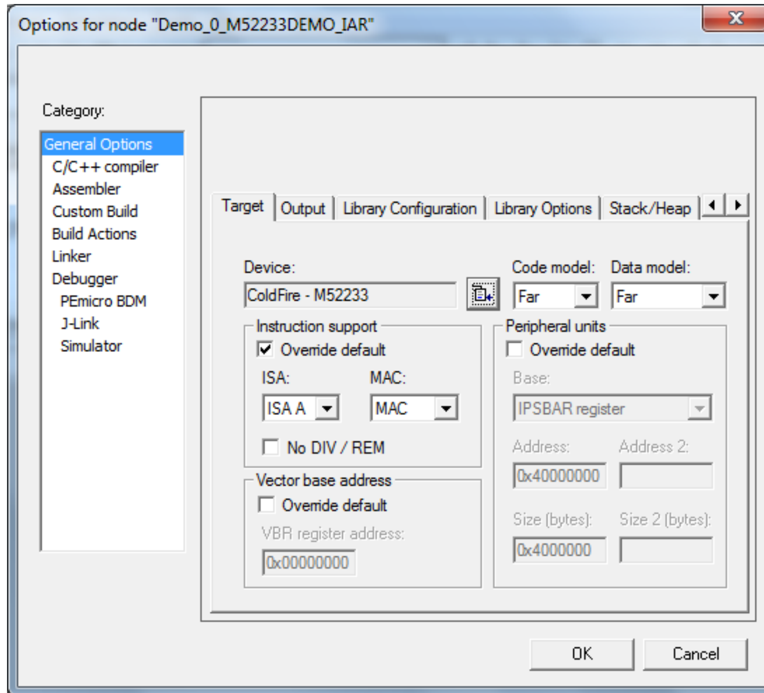

Figure 2-4 GUI set of MAC Support

If the application uses an EMAC unit, and its registers need to be multi-thread protected, inform `Abassi_CF_IAR.s` through the assembler command line as shown below:

Table 2-5 Enabling EMAC register protection

```
acf … --mac=emac … Abassi_CF_IAR.s
```

Alternatively, this can be set through the GUI in the “*General Options / Target / Instruction support*” menu:

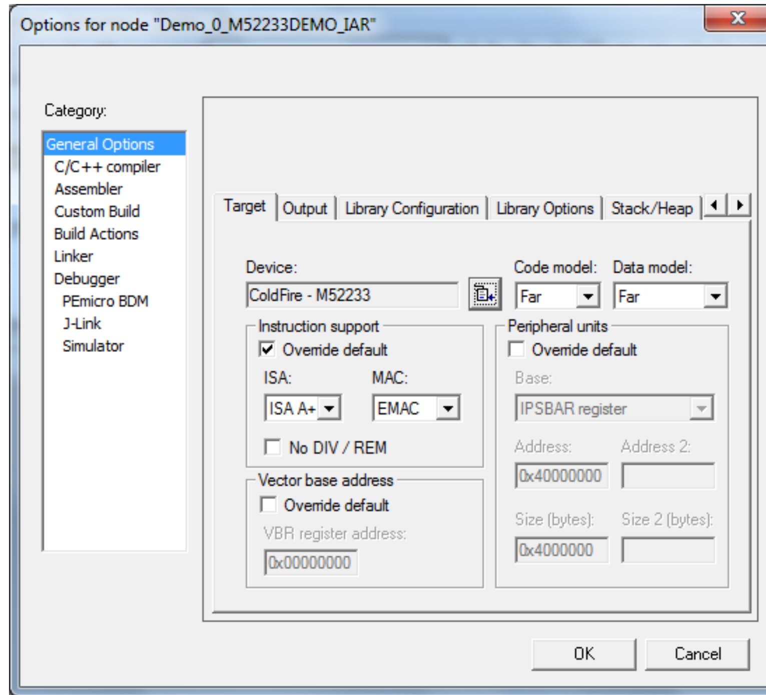

Figure 2-5 GUI set of EMAC Support

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 for interrupt numbers less than 64) 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.

The distribution makes provision for 192 sources of interrupts handled through the interrupt dispatcher, as specified by the token `OS_N_INTERRUPTS` in the file `Abassi_CF_IAR.s`, and the internal default value used by `Abassi.c`. Even though the ColdFire interrupt table holds 256 interrupts, the first 64 entries of the interrupt vector table are hard mapped to dedicated handlers and not handled by the dispatcher.

3.1 Interrupt Handling

3.1.1 First 64 Table Entries

The first 64 entries of the interrupt table are not handled through the dispatcher. Instead, they are mapped to dedicated handlers: simple infinite loops. These handlers are declared `weak`, allowing an application to overload the handlers defined in `Abassi_CF_IAR.s` and use its own. The sixteen TRAP handlers are named from `TRAP_0_hndl()` to `TRAP_15_hndl()`, the seven auto vectors are named from `AUTO_1_hndl()` to `AUTO_7_hndl()`, and all other entries in the lower 64 (excluding the 1st and 2nd) are mapped to `FAULThnld()`.

3.1.2 Interrupt Table Size

Most application do not require all 256 interrupts to be handled, as they either typically do not use all the entries of the table, and/or the peripherals mapped to the higher entries are not using 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_CF_IAR.s`, the ColdFire interrupt table itself must be shrunk, and the value used in the file `Abassi.c`, to reduce the ISR dispatcher table look-up. The interrupt table size is defined by the token `OS_N_INTERRUPTS` in the file `Abassi_CF_IAR.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 192; that is the maximum of 256, minus the lower 64.

For example, on a MCF52233 device, none of the last 128 entries of the interrupt table are attached to peripherals (index 127 is the last peripheral, which is the Real Time Clock). The 256 entry table can therefore be reduced to only 128 entries. The value to set in Ab`assi_CF_IAR.s files` is 64, which is the total of 128 entries minus 64 (the lower 64 entries). The changes are shown in the following table:

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

```
  …

#ifndef OS_N_INTERRUPTS          ; # of entries in the interupt table mapped to
OS_N_INTERUPTS    EQU 64         ; ISRdispatch()
#endif

   …
```

Alternatively, it is possible to overload the `OS_N_INTERRUPTS` value set in `Abassi_CF_IAR.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 64

```
acf … -DOS_N_INTERRUPTS=64 …
```

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`

```
iccсf … -DOS_N_INTERRUPTS=64 …
```

The interrupt table size used by `Abassi_CF_IAR.s` can also be set through the GUI, in the "*Assembler / Preprocessor*" menu, as shown in the following figure:

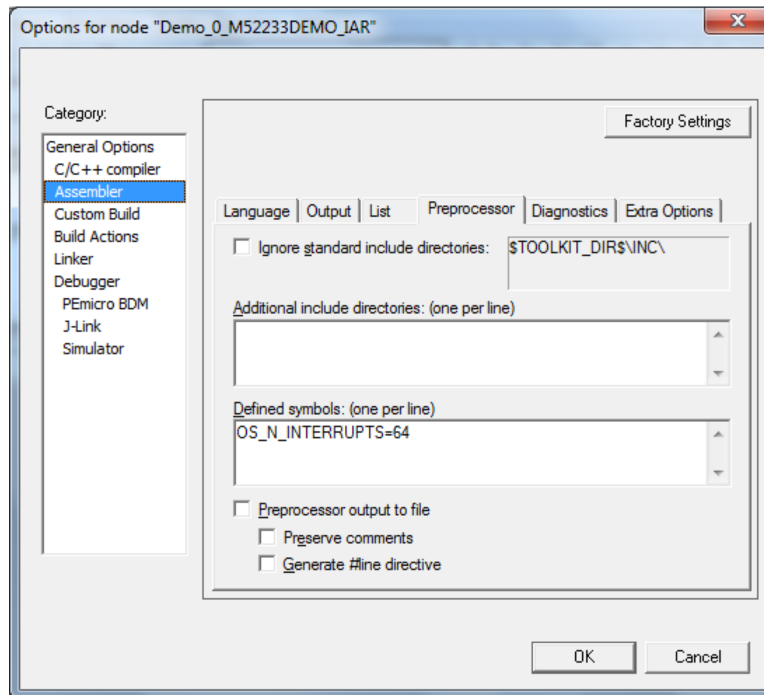

Figure 3-1 GUI set of interrupt table sizing

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

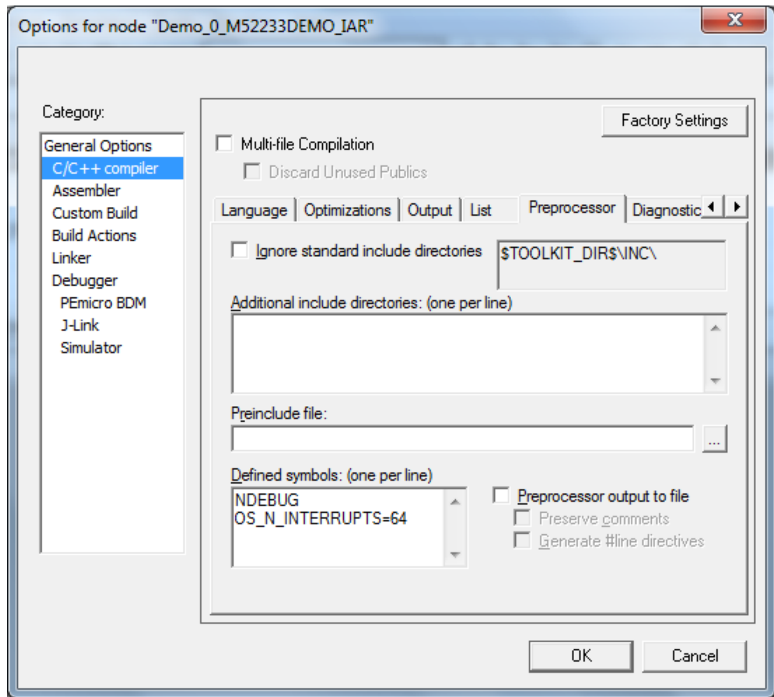

Figure 3-2 GUI set of interrupt table sizing

3.1.3 OSeint() and OSdint()

The ColdFire implementation of `OSeint()` and `OSdint()` is a bit different than for all other Abassi ports. The reason for this idiosyncrasy is due to the fact the ColdFire processor does not possess a way to control a global interrupt disable/enable; instead the interrupts are "enable/disable" by setting the priority level value of the 3 IPL bits in the processor status register. As the philosophy behind `OSdint()` and `OSeint()` is to perform a global interrupt disabling, with restoring of the interrupt state before the disabling, it becomes necessary to keep track of the interrupt level (IPL bits), as disabling the interrupt is done by setting the IPL bits to `111b`. So, the `OSdint()` return value is 0x00002700 for previously disabled interrupts, and between 0x00002000 to 0x00002600 for previously enabled interrupts; see Table 3-4.

Table 3-4 OSdint() return value

Value	Description
`0x00002000`	The 3 IPL bits were 000 before disabling the interrupt
`0x00002100`	The 3 IPL bits were 001 before disabling the interrupt
`0x00002200`	The 3 IPL bits were 010 before disabling the interrupt
`0x00002300`	The 3 IPL bits were 011 before disabling the interrupt
`0x00002400`	The 3 IPL bits were 100 before disabling the interrupt
`0x00002500`	The 3 IPL bits were 101 before disabling the interrupt
`0x00002600`	The 3 IPL bits were 110 before disabling the interrupt
`0x00002700`	The 3 IPL bits were 111 before disabling the interrupt

When using `OSeint()`, the interrupt level can be set by using any of the values indicated in the above table. If a value of 0 is used with `OSeint()`, then the interrupts are disabled by setting the 3 IPL bits to 111. Any value other than 0, or the ones in the above table, enables the interrupts by setting the 3 IPL bits to 000.

3.1.4 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 3-5 shows the code required to attach on a MCF52233 the `PIT0` (Programmable Interrupt Timer #0) interrupt to the RTOS timer tick handler (`TIMtick`). The `PIT0` interrupt is mapped to the index #119 in the interrupt vector

Table 3-5 Attaching a Function to an Interrupt

```
#include "Abassi.h"

   …
   OSstart();
   …
   OSisrInstall(119, &TIMtick);
   /* Set-up the count reload and enable SysTick interrupt */

   … /* More ISR setup */

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

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 3-6:

Table 3-6 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 (See Section 3.1.3 for more details about using `OSdint()` and `OSeint()` on a ColdFire).

The interrupt controller on the ColdFire does not clear the interrupt generated by a peripheral; neither does the RTOS. This means the peripheral generating the interrupt must be informed to remove the interrupt request. This operation must be performed in every interrupt handler otherwise the interrupt will be re-entered over and over. In the case of the RTOS timer tick interrupt handler, define the token `OX_TIM_TICK_ACK`, or do it in the timer call back function, if used, and called at every timer tick.

For example, re-using the `PIT0` on the MCF52233 as the RTOS timer tick source of interrupt, the interrupt can be cleared inside the RTOS timer tick interrupt handler through:

```
icccf … -DOX_TIM_TICK_ACK="MCF_PIT0_PCR|=MCF_PIT_PCSR_PIF" …
```

Alternatively, this can be set through the GUI:

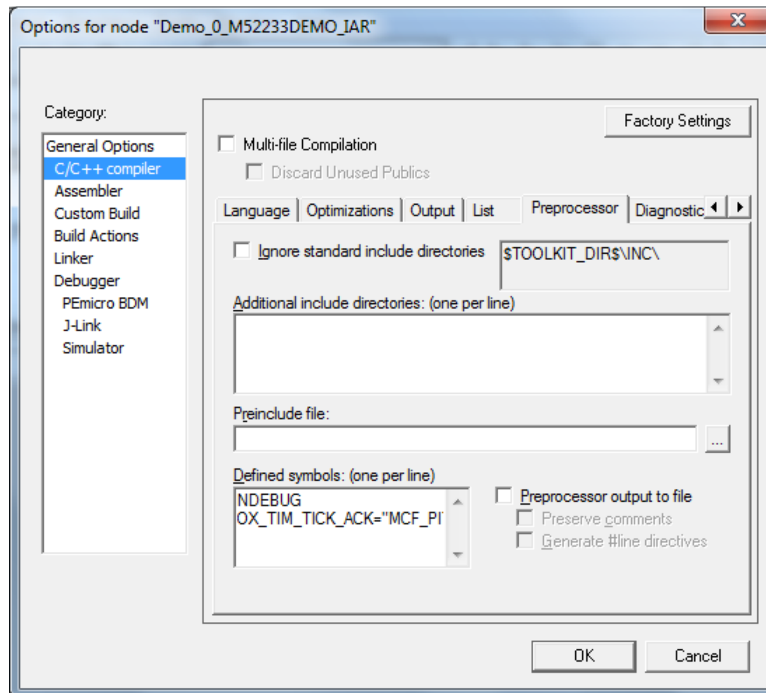

Figure 3-3 GUI set of `OX_TIM_TICK_ACK`

3.2 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 ColdFire processor. The area of the interrupt vector table to modify is located in the file `Abassi_CF_IAR.s` around line 275. For example, on a MCF52233 device, `PIT0` is mapped to interrupt number 119 and `PIT1` is mapped to the interrupt number 120. 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 3-7 Distribution interrupt table code

```
INT_NMB   SET   64                 ; INT_NMB is used to track the interrupt number
    REPT    OS_N_INTERRUPTS        ; Map all the external interrupts to ISRdispatch()
        DC32    ISRdispatch
INT_NMB SET  INT_NMB+1
    ENDR
```

Attaching a fast interrupt handler to the `PIT0`, and another one to `PIT1`, assuming the names of the interrupt functions to attach are respectively `PIT0_IRQhandler()` and `PIT1_IRQhandler()`, is shown in the following table:

Table 3-8 MCF52233 PIT 0 / 1 Fast Interrupts

```
    EXTERN  PIT0_IRQhandler
    EXTERN  PIT1_IRQhandler

    …


INT_NMB   SET   64                 ; INT_NMB is used to track the interrupt number
    REPT    OS_N_ INTERRUPTS       ; Map all the external interrupts to ISRdispatch()
        IF INT_NMB == 119          ; When is interrupt #119, set the PIT #0 handler
            DC32    PIT0_IRQhandler
        ELSEIF INT_NMB == 120      ; When is interrupt #120, set the PIT #1 handler
            DC32    PIT1_IRQhandler
        ELSE                       ; All others interrupt # mapped to ISRdispatch()
            DC32    ISRdispatch
        ENDIF
INT_MB  SET   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 `PIT0` on the MCF52233 device, this would look something like:

Table 3-9 Fast Interrupt with Dedicated Stack

```
    …

    ELSEIF INT_NMB == 119
       DC32     PIT0preHandler             ; Set the addres of the pre handler

    …
    …

    RSEG     RCODE:CODE(1)

    EXTERN  UART0handler

PIT0preHandler:
    …  …  …                                 ; ISR context save

    movea.l  SP, A0                         ; Save a copy of current stack pointer
    movea.l  #PIT0_stack, SP                ; Set-up ISR stack for PIT 0 interrupt
    pea      (A0)                           ; Save original stack pointer on the stack

    bl      PIT0handler                     ; Enter the interrupt handler

    movea.l (SP), SP                        ; Recover original sp
    …
    …

    RSEG    FAR_Z:DATA:SORT:NOROOT(2)
    ALIGN   4

    DS8     PIT0_stack_size                ; Room for the fast interrupt stack
PIT0_stack:

    …
```

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

3.3 Nested Interrupts

The ColdFire 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 7 is the highest (interrupts are disable but not the exceptions faults) and 0 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 Freescale ColdFire. 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 3-10 below and the line to modify is the one with `#define OX_NESTED_INTS 1`:

Table 3-10 Removing interrupt nesting

```
#elif defined(__ICCCF__)
  #define OX_NESTED_INTS 0             /* CF has 8 nested interrupt levels */
```

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

Table 3-11 Propagating interrupt nesting

```
#elif defined(__ICCCF__)
  #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 ColdFire, the context save contents of a blocked or pre-empted task is different from the one used in an interrupt. The following table lists the number of bytes required by each type of context save operation:

Table 4-1 Context Save Stack Requirements

Description	No MAC	**MAC**	**EMAC**
Blocked/Preempted task context save	`40 bytes`	`52 bytes`	`72 bytes`
ISR dispatcher context save (`OS_ISR_STACK == 0`)	`28 bytes`	`40 bytes`	`80 bytes`
ISR dispatcher context save (`OS_ISR_STACK != 0`)	`32 bytes`	`44 bytes`	`84 bytes`

The numbers for the interrupt dispatcher context save include the 8 bytes 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 ColdFire processor needs alignment on 4 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 4 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_CF_IAR.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 PIT0 peripheral on a MCF52233 device, which decrements its counter once every 2 CPU cycle; the number listed in the following table are the cycles, not the PIT0 counter value.. 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 (IAR/ColdFire 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 code and data models are the far ones, and the instruction support was set to ISA A+ and with the EMAC. 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 340 cycles.

Table 5-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	342	384	474
2	350	392	474
3	358	400	474
4	366	408	474
5	374	416	474
6	382	424	474
7	390	432	474
8	398	388	474
9	406	386	474
10	414	404	474
11	422	412	474
12	430	420	474
13	438	428	474
14	446	436	474
15	454	444	474
16	462	402	474
17	470	410	474
18	478	418	474
19	486	426	474
20	494	434	474
21	502	442	474
22	510	450	474
23	518	458	474
24	526	414	474

When `OS_SEARCH_FAST` is set to 0, each extra priority level to traverse requires exactly 8 CPU cycles. When `OS_SEARCH_FAST` is set to 1, each extra priority level to traverse requires exactly 8 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 42 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 12 cycles needed, but without the 8 times 8 cycle accumulation. Finally, the third option, when `OS_SEARCH_FAST` is set to 5, delivers a 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 484, for 64 or more, it is 494).

The first observation, when looking at this table, is that the first option, when OS_SEARCH_FAST is set to 0, is the most CPU efficient when the priority span is less than 8. For more than 8 priority spans, the second option (when OS_SEARCH_FAST is set to 1) is overall more CPU efficient than the third option (when OS_SEARCH_FAST is set to 5) for all spans shown in the table. When the span reaches around 40, then the third option is more efficient than the second. So, the build option OS_SEARCH_FAST should never be set to 5, as it is not the most efficient method, unless the application has way more than 40 priority levels.

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. The IAR Embedded Workbench for the ColdFire supplies all the peripheral register definitions, exactly the same ones as in the Freescale's CodeWarrior development environment.

7 Measurements

This section gives an overview of the memory requirements and the CPU latency encountered when the RTOS is used on the Freescale ColdFire and compiled with IAR Embedded Workbench. The CPU cycles are exactly the CPU clock cycles.

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. Only the `far` data and code model have been measured. One would assume using `near` code model should provide smaller code. The code optimization settings of the compiler that were used for the memory measurements are:

1. Optimization level: High
2. Optimize for: Size
3. All transformations are enabled

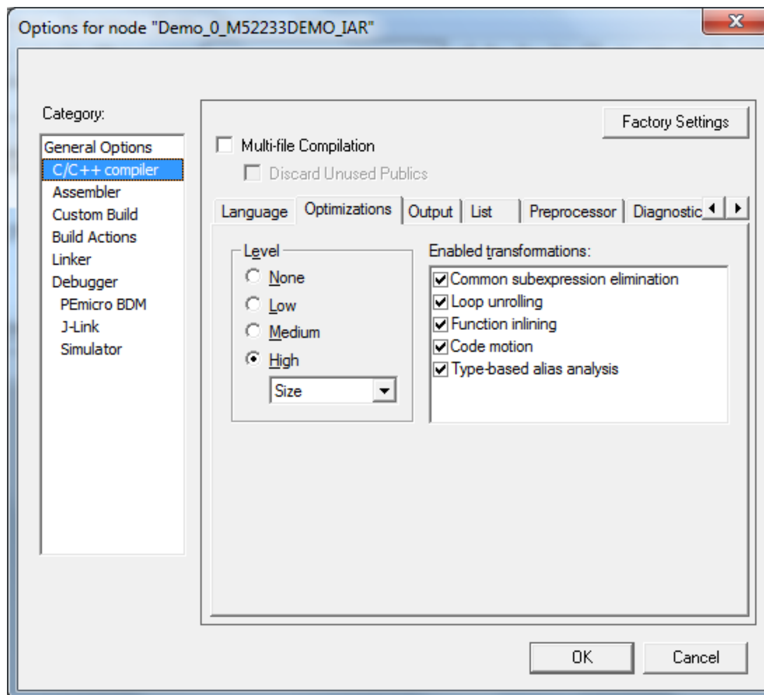

Figure 7-1 Memory Measurement Code Optimization Settings

Table 7-1 "C" Code Memory Usage

Description	ISA A / ISA A+	ISA B / ISA C
Minimal Build	< 875 bytes	< 875 bytes
+ Runtime service creation / static memory	< 1100 bytes	< 1100 bytes
+ Multiple tasks at same priority	< 1200 bytes	< 1200 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 1750 bytes	< 1725 bytes
+ Timer & timeout + Timer call back + Round robin	< 2250 bytes	< 2250 bytes
+ Events + Mailbox	< 3075 bytes	< 3050 bytes
Full Feature Build (no names)	< 3525 bytes	< 3475 bytes
Full Feature Build (no names / no runtime creation)	< 3250 bytes	< 3200 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 3600 bytes	< 3550 bytes

Table 7-2 Assembly Code Memory Usage

Description	ISA A / ISA A+	ISA B / ISA C
Assembly code size	290 bytes	310 bytes
Vector table (per interrupt handler entry)	+4 bytes	+4 bytes
Hybrid Stack Enabled	+26 bytes	+28 bytes
MAC unit protected	+40 bytes	+40 bytes
EMAC unit protected	+136 bytes	+136 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 a Freescale M52233DEMO evaluation board populated with a 60 MHz MCF52233 device. All measurements have been performed on the real platform using the PIT0 timer to count the cycles. This means the interrupt latency measurements had to be instrumented to read the PIT0 counter value. This instrumentation can add up to 5 or 6 cycles to the measurements. Also, the PIT timers are decrementing their counter by one every 2 CPU cycles, meaning the measured values were all doubled to represent CPU cycles. Both data and code model were set to far and the instruction set was set to ISA A+ as this is the native instruction set of the MCF52233 device. The code optimization settings that were used for the latency measurements are:

1. Optimization level: High
2. Optimize for: Speed
3. All transformations are enabled

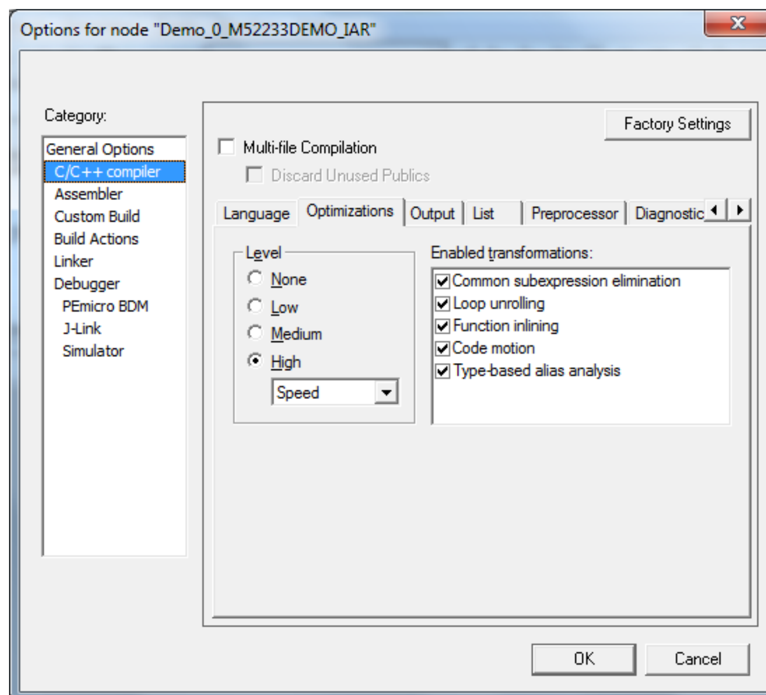

Figure 7-2 Latency Measurement Code Optimization Settings

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 7-3 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 7-4 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 7-5 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 fourth 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 7-6 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 7-7, Table 7-8 and Table 7-9 list the results obtained using a Freescale M52233DEMO evaluation board, where the cycle count is measured using the Programmable Interrupt Timer #0 (PIT0) peripheral on the MCF52233. This timer is set-up to decrements its counter by 1 at every 2 CPU cycle (there is a hard-wired pre-scaler of 2 on the PIT0). 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. For this measurement, the MCF52233 Programmable Interrupt Timer #0 (PIT0) is again 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 in any of these tests.

In the following three tables, the latency numbers between parentheses are the measurements when the build option OS_SEARCH_ALGO is set to a negative value, indicating to use a linked list instead of a look-up table. The regular number is the latency measurements when the build option OS_SEARCH_ALGO is set to 0.

Table 7-7 Latency Measurements (NO MAC)

Description	Minimal Features	Full Features
Semaphore posting no task switch	166 (164)	230 (258)
Semaphore waiting no blocking	172 (170)	244 (276)
Semaphore posting with task switch	242 (270)	382 (442)
Semaphore waiting with blocking	260 (258)	434 (452)
Semaphore posting in ISR with task switch	566 (588)	712 (758)
Event setting no task switch	n/a	226 (256)
Event getting no blocking	n/a	278 (304)
Event setting with task switch	n/a	412 (476)
Event getting with blocking	n/a	470 (482)
Event setting in ISR with task switch	n/a	748 (798)
Mailbox writing no task switch	n/a	284 (312)
Mailbox reading no blocking	n/a	304 (328)
Mailbox writing with task switch	n/a	442 (496)
Mailbox reading with blocking	n/a	506 (528)
Mailbox writing in ISR with task switch	n/a	784 (834)
Interrupt Latency	64	64
Interrupt overhead entering the kernel	324 (318)	330 (316)
Interrupt overhead NOT entering the kernel	124	124
Context switch	48	46

Table 7-8 Latency Measurements (MAC)

Description	Minimal Features	Full Features
Semaphore posting no task switch	166 (164)	230 (258)
Semaphore waiting no blocking	172 (170)	244 (376)
Semaphore posting with task switch	262 (290)	402 (462)
Semaphore waiting with blocking	280 (278)	454 (472)
Semaphore posting in ISR with task switch	596 (618)	742 (788)
Event setting no task switch	n/a	226 (256)
Event getting no blocking	n/a	278 (304)
Event setting with task switch	n/a	432 (496)
Event getting with blocking	n/a	490 (502)
Event setting in ISR with task switch	n/a	778 (828)
Mailbox writing no task switch	n/a	284 (312)
Mailbox reading no blocking	n/a	304 (328)
Mailbox writing with task switch	n/a	462 (516)
Mailbox reading with blocking	n/a	526 (550)
Mailbox writing in ISR with task switch	n/a	814 (864)
Interrupt Latency	74	74
Interrupt overhead entering the kernel	334 (328)	340 (326)
Interrupt overhead NOT entering the kernel	168	168
Context switch	68	66

Table 7-9 Latency Measurements (EMAC / EMAC_B)

Description	Minimal Features	Full Features
Semaphore posting no task switch	162 (160)	230 (256)
Semaphore waiting no blocking	168 (166)	246 (274)
Semaphore posting with task switch	300 (328)	442 (500)
Semaphore waiting with blocking	312 (310)	490 (508)
Semaphore posting in ISR with task switch	652 (670)	798 (844)
Event setting no task switch	n/a	224 (256)
Event getting no blocking	n/a	276 (306)
Event setting with task switch	n/a	472 (536)
Event getting with blocking	n/a	526 (538)
Event setting in ISR with task switch	n/a	832 (884)
Mailbox writing no task switch	n/a	284 (310)
Mailbox reading no blocking	n/a	304 (328)
Mailbox writing with task switch	n/a	500 (556)
Mailbox reading with blocking	n/a	564 (588)
Mailbox writing in ISR with task switch	n/a	872 (914)
Interrupt Latency	90	90
Interrupt overhead entering the kernel	352 (342)	356 (344)
Interrupt overhead NOT entering the kernel	233	233
Context switch	100	102

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                           */
```