CODE TIME TECHNOLOGIES

mAbassi RTOS

Porting Document

SMP / ARM Cortex-M3 – CCS

Table of Contents

List of Figures

List of Tables

1 Introduction

This document details the port of the SMP / BMP multi-core mAbassi RTOS to the ARM Cortex-M3. The software suite used for this specific port is the Code Composer Studio from Texas Instruments (abbreviated CCS); the version used for the port and all tests is Version 5.2.0.00069.

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
`mAbassi.h`	Include file for the RTOS
`mAbassi.c`	RTOS "C" source file
`cmsis.h`	Optional CMSIS V 3.0 RTOS API include file
`cmsis.c`	Optional CMSIS V 3.0 RTOS API source file
`mAbassi_SMP_CORTEXM3_CCS.s`	RTOS assembly file for the SMP ARM Cortex-M3 to use with Code Composer Studio
`Demo_3_SMP_PANDA_M3_CCS.c`	Demo code that runs on the Pandaboard 4460 ES evaluation board
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

The RTOS reserves `SVC` (Supervisor call, interrupt vector #11) numbers 0 to 7. A hook is made available for the application to use a `SVC`, as long as the numbers used are above 7.

To optimize reaction time of the mAbassi RTOS components, it was decided to require the processor to always operate in privileged mode (which is the default 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.

1.3 Features

The assembly file does not use the `BL` instruction when calling any modules. This was done to allow the assembly file access the whole program address space when it is larger than 16 Mbytes. One has to remember that the `BL` instruction has a limited addresses range, which is between ±16 Mbytes.

Every one of the Code Composer application binary interfaces (legacy coff, ti_arm9_abi, and eabi; tiabi cannot be used with the Cortex-M3) are supported in the assembly file.

2 Target Set-up

Very little is needed to configure the Code Composer Studio development environment to use the mAbassi RTOS in an application. All there is to do is to add the files `mAbassi.c` and `mAbassi_SMP_CORTEXM3_CCS.s` in the source files of the application project, and make sure the configuration settings in the file `mAbassi_SMP_CORTEXM3_CCS.s` (described in the following sub-sections) 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 `mAbassi.h`. There is no need to include a start-up file, as the file `mAbassi_SMP_CORTEXM3_CCS.s` takes care of all the start-up operations required for an application operating on a multi-core processor.

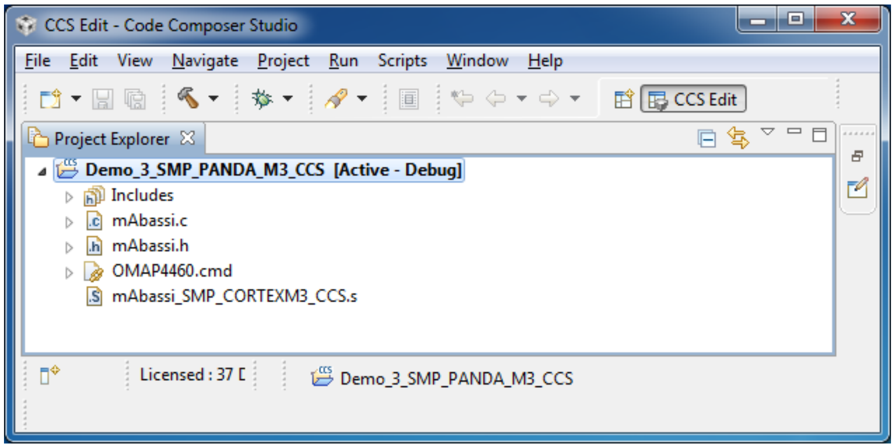

CCS Edit - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
CCS Edit
Project Explorer
Demo_3_SMP_PANDA_M3_CCS [Active - Debug]
Includes
mAbassi.c
mAbassi.h
OMAP4460.cmd
mAbassi_SMP_CORTEXM3_CCS.s
Licensed : 37 D
Demo_3_SMP_PANDA_M3_CCS

Figure 2-1 Project File List

NOTE: By default, the Code Composer Studio runtime libraries are not multithread-safe, but Code Composer Studio has a rudimentary hook to make some part of the libraries multithread-safe. The required hooks are applied in the file `mAbassi.h` by attaching the mAbassi internal mutex (`G_OSmutex`) during runtime in `OSstart()`. This implies that any of the Code Composer Studio runtime libraries protected against multi-threading cannot be used in an interrupt as locking a mutex in an interrupt is an invalid kernel request.

2.1 Platform Set-up

The ARM Cortex-M3, contrary to the Arm9, Arm11 or Arm15, does not have a predefined multi-core interface. So, the different multi-core Cortex-M3 devices use different techniques to implement a multi-core version, based on the intrinsically single core Cortex-M3. These unique techniques include, but are not limited to, inter-core interrupts, identification of the core (core ID), reset handling, etc. The build option `OS_PLATFORM` is needed by mAbassi to identify and properly handle the different devices.

The definition of the build option `OS_PLATFORM` must be done for both the `mAbassi.c` file and the `mAbassi_SMP_CORTEXM3_CCS.s` file. In the case of the file `mAbassi.c`, `OS_PLATFORM` is one of the rare extra build options. In the case of the file `mAbassi_SMP_CORTEXM3_CCS.s`, to modify the target platform, all there is to do is to change the numerical value associated to the token, located around line 45; this is shown in the following table:

Table 2-1 `OS_PLATFORM` modification

```
  .if !($$defined(OS_PLATFORM))
OS_PLATFORM   .equ 4460
  .endif
```

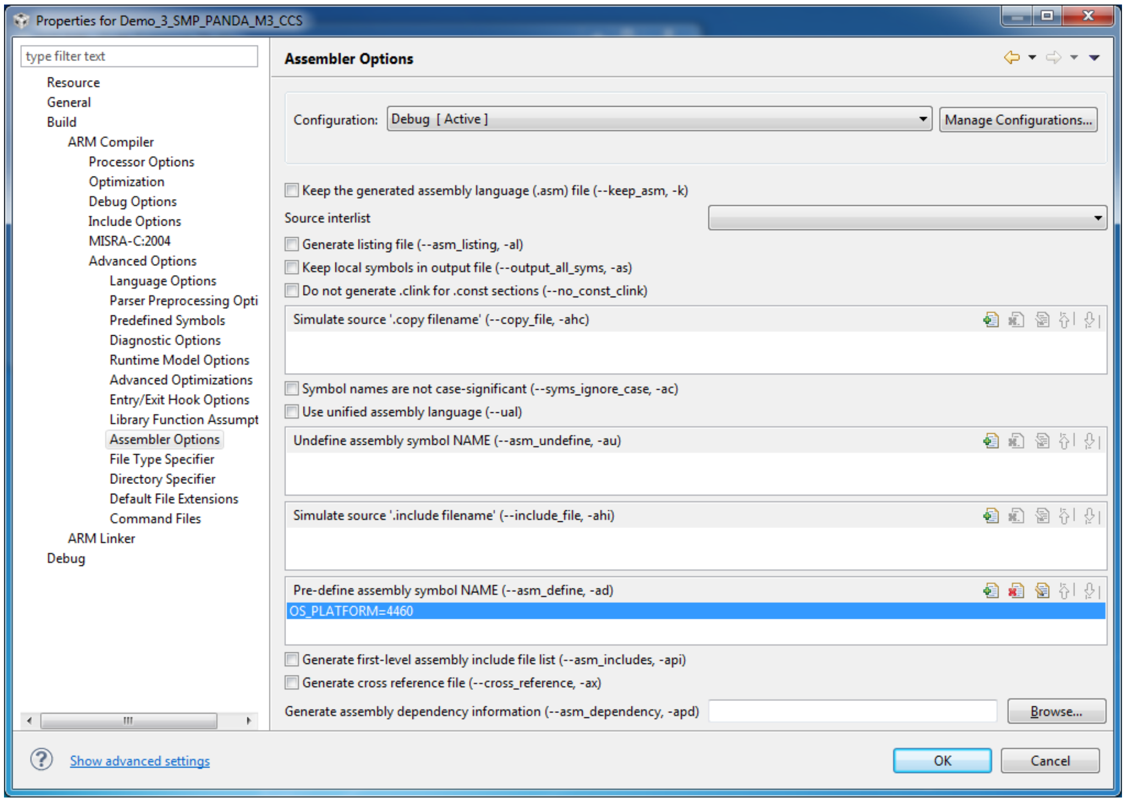
The supported platforms are described in comments to the right of the definition of `OS_PLATFORM`.

Alternatively, it is possible to overload the `OS_PLATFORM` value set in `mAbassi_SMP_CORTEXM3_CCS.s` by using the assembler command line option `–asm_define` and specifying the required number of cores as shown in the following example, where the number of cores is set to 3:

Table 2-2 Command line set of `OS_PLATFORM`

```
cl470  … -asm_define=OS_PLATFORM=4460  …
```

The target platform can also be set through the GUI, in the "*Build / ARM Compiler / Advanced Options / Assembler Options*" menu, as shown in the following figure:


Properties for Demo_3_SMP_PANDA_M3_CCS

type filter text

- Resource
- General
- Build
  - ARM Compiler
    - Processor Options
    - Optimization
    - Debug Options
    - Include Options
    - MISRA-C:2004
    - Advanced Options
      - Language Options
      - Parser Preprocessing Opti
      - Predefined Symbols
      - Diagnostic Options
      - Runtime Model Options
      - Advanced Optimizations
      - Entry/Exit Hook Options
      - Library Function Assumpt
      - Assembler Options
      - File Type Specifier
      - Directory Specifier
      - Default File Extensions
      - Command Files
  - ARM Linker
- Debug

Assembler Options

Configuration: Debug [Active] Manage Configurations...

- Keep the generated assembly language (.asm) file (--keep_asm, -k)
- Source interlist
- Generate listing file (--asm_listing, -al)
- Keep local symbols in output file (--output_all_syms, -as)
- Do not generate .clink for .const sections (--no_const_clink)
- Simulate source '.copy filename' (--copy_file, -ahc)
- Symbol names are not case-significant (--syms_ignore_case, -ac)
- Use unified assembly language (--ual)
- Undefine assembly symbol NAME (--asm_undefine, -au)
- Simulate source '.include filename' (--include_file, -ahi)
- Pre-define assembly symbol NAME (--asm_define, -ad)
  - OS_PLATFORM=4460
- Generate first-level assembly include file list (--asm_includes, -api)
- Generate cross reference file (--cross_reference, -ax)
- Generate assembly dependency information (--asm_dependency, -apd) Browse...

Show advanced settings OK Cancel

Figure 2-2 GUI set of `OS_PLATFORM`

`OS_PLATFORM` must be also defined for `mAbassi.c` using either the GUI or the command line –D option as with any other build option for mAbassi.

2.2 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 `mAbassi_CORTEXM3_CCS.s`. To disable this feature, set the definition of `OS_ISR_STACK` to a value of zero. To enable it, and specify the hybrid 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 stack size of 256 bytes is allocated; this is shown in the following table:

Table 2-3 `OS_ISR_STACK`

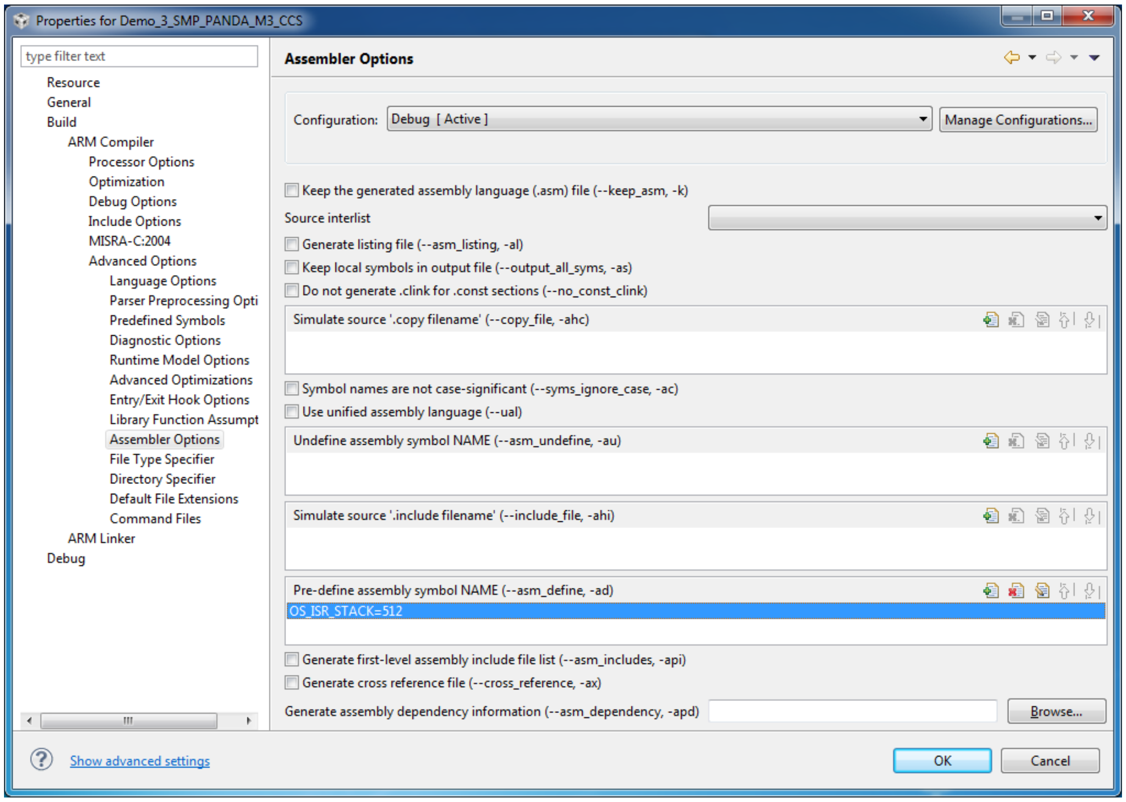
```
  .if !($$defined(OS_ISR_STACK))
OS_ISR_STACK   .equ 256              ; If using a dedicated stack for the nested ISRs
  .endif                             ; 0 if not used, otherwise size of stack in bytes
```

Alternatively, it is possible to overload the `OS_ISR_STACK` value set in `mAbassi_CORTEXM3_CCS.s` by using the assembler command line option `–asm_define` and specifying the desired hybrid stack size as shown in the following example, where the hybrid stack size is set to 512 bytes:

Table 2-4 Command line set of `OS_ISR_STACK`

```
cl470  … -asm_define=OS_ISR_STACK=512  …
```

The hybrid stack size can also be set through the GUI, in the "*Build / ARM Compiler / Advanced Options / Assembler Options*" menu, as shown in the following figure:



Assembler Options

Configuration: Debug [Active]

- [ ] Keep the generated assembly language (.asm) file (--keep_asm, -k)
- Source interlist
- [ ] Generate listing file (--asm_listing, -al)
- [ ] Keep local symbols in output file (--output_all_syms, -as)
- [ ] Do not generate .clink for .const sections (--no_const_clink)
- Simulate source '.copy filename' (--copy_file, -ahc)
- [ ] Symbol names are not case-significant (--syms_ignore_case, -ac)
- [ ] Use unified assembly language (--ual)
- Undefine assembly symbol NAME (--asm_undefine, -au)
- Simulate source '.include filename' (--include_file, -ahi)
- Pre-define assembly symbol NAME (--asm_define, -ad): OS_ISR_STACK=512
- [ ] Generate first-level assembly include file list (--asm_includes, -api)
- [ ] Generate cross reference file (--cross_reference, -ax)
- Generate assembly dependency information (--asm_dependency, -apd)

Figure 2-3 GUI set of `OS_ISR_STACK`

2.3 Saturation Bit Set-up

In the ARM Cortex-M3 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 40 in the file `mAbassi_CORTEXM3_CCS.s`. The distribution code disables the localization of the Q bit, setting the token `HANDLE_PSR_Q` to zero, as shown in the following table:

Table 2-5 Saturation Bit configuration

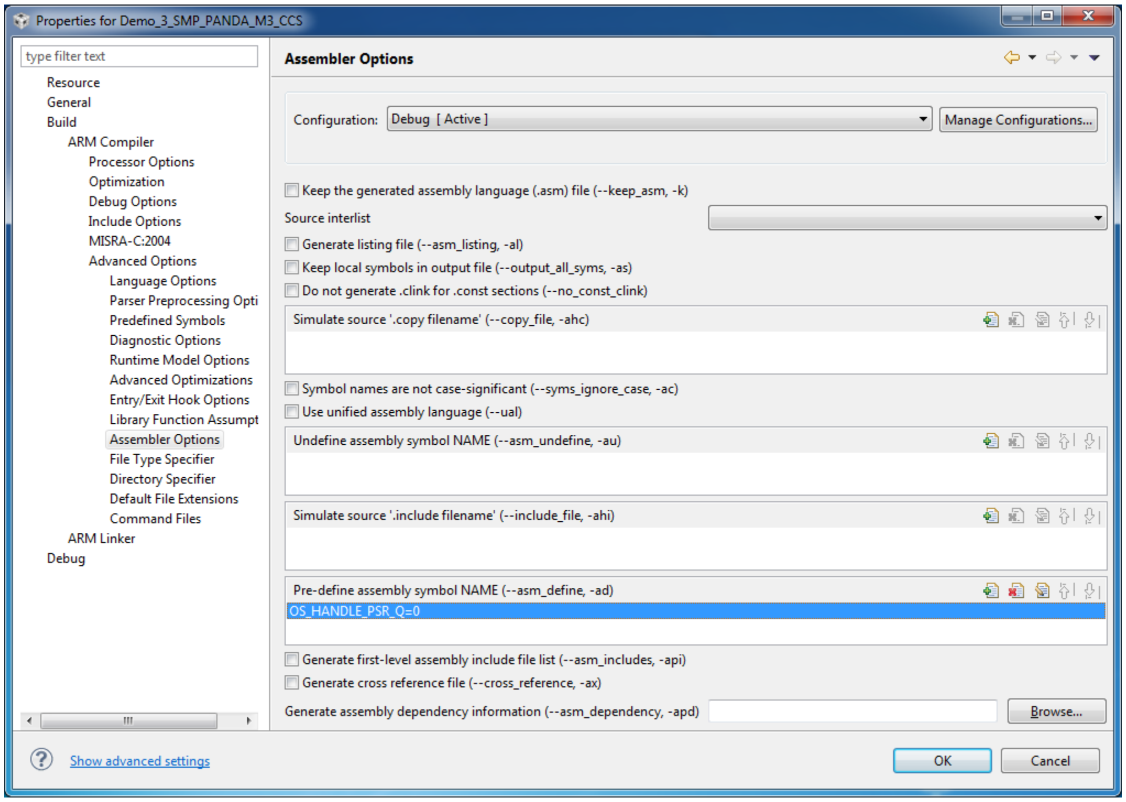
```
  .if !($$defined(OS_HANDLE_PSR_Q))
OS_HANDLE_PSR_Q      .equ 0            ; If we keep the Q bit (saturation) on per tasks
  .endif
```

Alternatively, it is possible to overload the `OS_HANDLE_PSR_Q` value set in `mAbassi_CORTEXM3_CCS.s` by using the assembler command line option `–asm_define` and specifying the desired setting with the following:

Table 2-6 Command line set of Saturation Bit configuration

```
cl470  … -asm_define=OS_HANDLE_PSR_Q=0 …
```

The saturation bit configuration can also be set through the GUI, in the "*Build / ARM Compiler / Advanced Options / Assembler Options*" menu, as shown in the following figure:



Properties for Demo_3_SMP_PANDA_M3_CCS

type filter text

- Resource
- General
- Build
  - ARM Compiler
    - Processor Options
    - Optimization
    - Debug Options
    - Include Options
    - MISRA-C:2004
    - Advanced Options
      - Language Options
      - Parser Preprocessing Opti
      - Predefined Symbols
      - Diagnostic Options
      - Runtime Model Options
      - Advanced Optimizations
      - Entry/Exit Hook Options
      - Library Function Assumpt
      - Assembler Options
      - File Type Specifier
      - Directory Specifier
      - Default File Extensions
      - Command Files
  - ARM Linker
- Debug

Assembler Options

Configuration: Debug [Active] Manage Configurations...

☐ Keep the generated assembly language (.asm) file (--keep_asm, -k)

Source interlist

☐ Generate listing file (--asm_listing, -al)

☐ Keep local symbols in output file (--output_all_syms, -as)

☐ Do not generate .clink for .const sections (--no_const_clink)

Simulate source '.copy filename' (--copy_file, -ahc)

☐ Symbol names are not case-significant (--syms_ignore_case, -ac)

☐ Use unified assembly language (--ual)

Undefine assembly symbol NAME (--asm_undefine, -au)

Simulate source '.include filename' (--include_file, -ahi)

Pre-define assembly symbol NAME (--asm_define, -ad)

OS_HANDLE_PSR_Q=0

☐ Generate first-level assembly include file list (--asm_includes, -api)

☐ Generate cross reference file (--cross_reference, -ax)

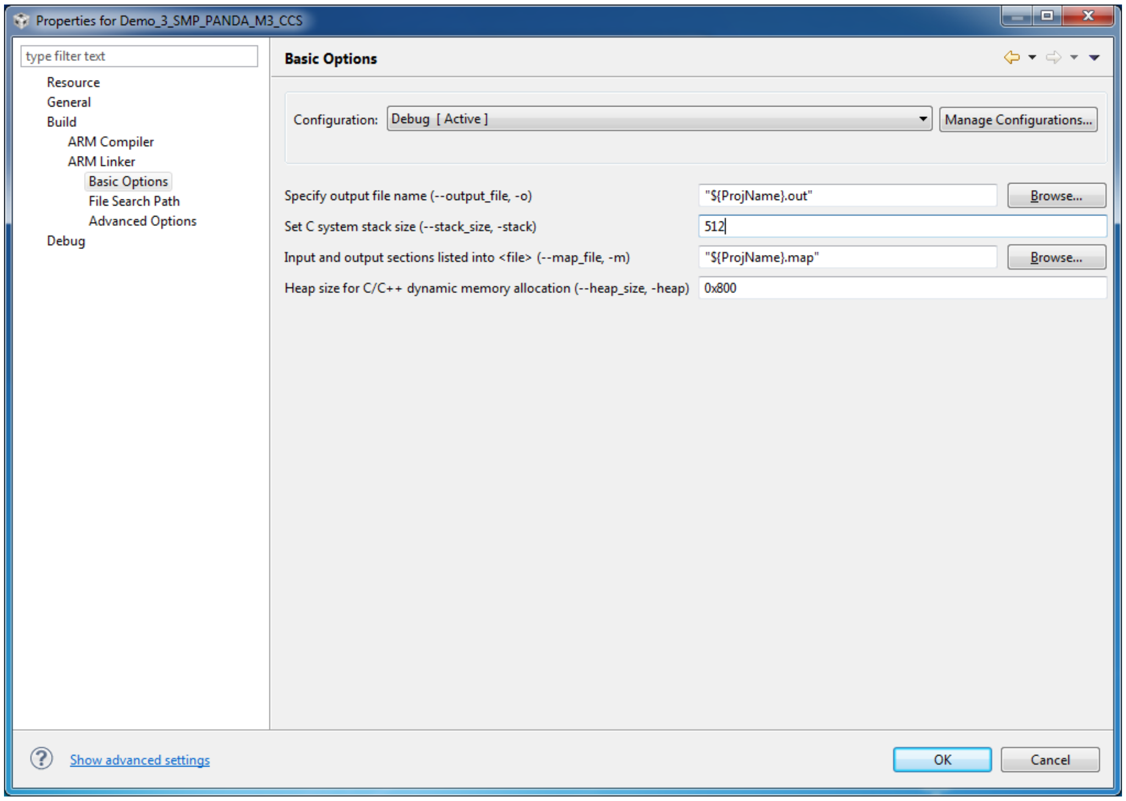
Generate assembly dependency information (--asm_dependency, -apd) Browse...

Show advanced settings OK Cancel

Figure 2-4 GUI set of Saturation Bit configuration

2.4 Stacks Set-up

The start-up stack size is defined with the linker line option `–stack_size`. Or through the GUI in the *"Properties"* menu *"Build / ARM Linker / Basic Options / Set C system stack size (--stack_size, -stack)"*. That linker-set stack is assigned to the Adam & Eve task, which is the element of code executing upon start-up on core #0. The other cores start with the `COREstartN()` functions, which are fully described in the mAbassi User's Guide [R1], and their stack sizes are defined as part of the mAbassi standard build options.



Configuration: Debug [Active]

Specify output file name (--output_file, -o)	"${ProjName}.out"
Set C system stack size (--stack_size, -stack)	512
Input and output sections listed into <file> (--map_file, -m)	"${ProjName}.map"
Heap size for C/C++ dynamic memory allocation (--heap_size, -heap)	0x800

Figure 2-5 GUI setting of Adam & Eve stack size

2.5 Number of Cores

When operating the mAbassi RTOS on a platform, the RTOS needs to be configured for the number of cores it has access to, or will use. This number is most of the time the same as the number of cores the device has, but it also can be set to a value less than the total number of cores on the device, but not larger. This must be done in both the `mAbassi.c` file and the `mAbassi_SMP_CORTEXM3_CCS.s` file, by setting the build option `OS_N_CORE`. In the case of the file `mAbassi.c`, `OS_N_CORE` is one of the standard build options. In the case of the file `mAbassi_SMP_CORTEXM3_CCS.s`, to modify the number of cores, all there is to do is to change the numerical value associated to the token, located around line 30; this is shown in the following table:

Table 2-7 `OS_N_CORE` modification

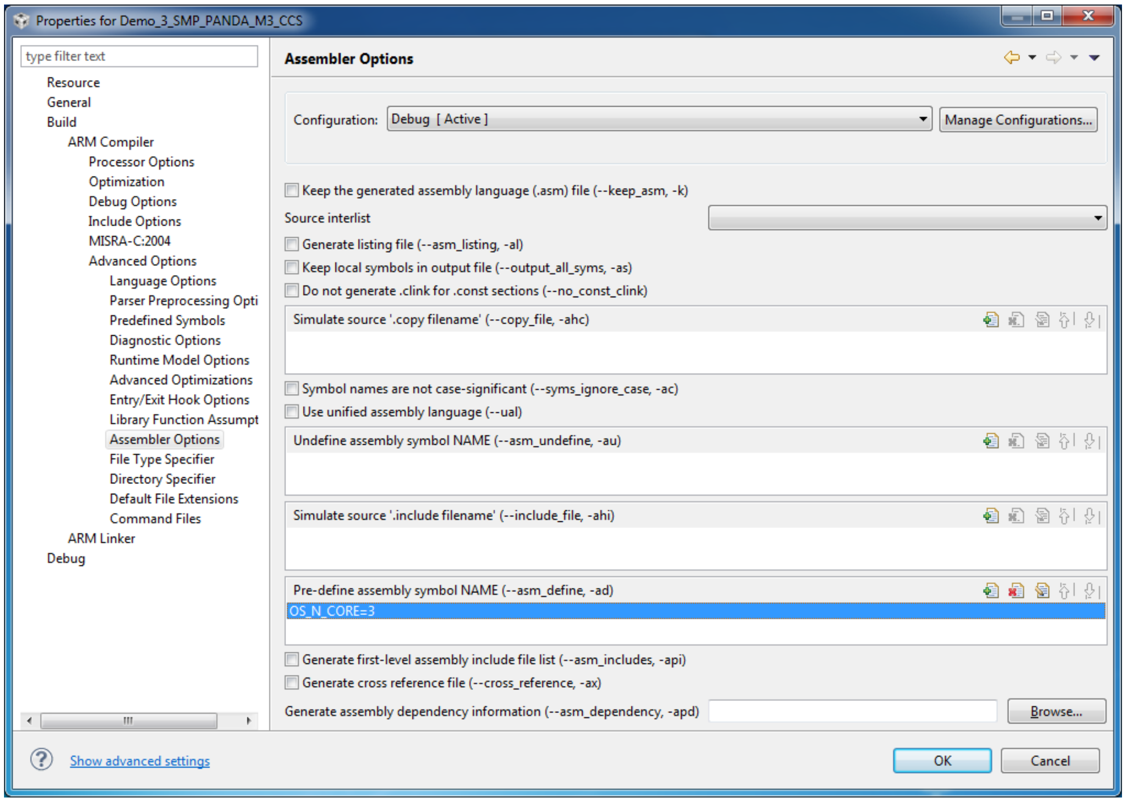
```
  .if !($$defined(OS_N_CORE))
OS_N_CORE   .equ 4
  .endif
```

Alternatively, it is possible to overload the `OS_N_CORE` value set in `mAbassi_SMP_CORTEXM3_CCS.s` by using the assembler command line option `–asm_define` and specifying the required number of cores as shown in the following example, where the number of cores is set to 3:

Table 2-8 Command line set of `OS_N_CORE`

```
cl470  … -asm_define=OS_N_CORE=3  …
```

The number of cores can also be set through the GUI, in the "*Build / ARM Compiler / Advanced Options / Assembler Options*" menu, as shown in the following figure:



Configuration: Debug [Active]

Keep the generated assembly language (.asm) file (--keep_asm, -k)

Source interlist

Generate listing file (--asm_listing, -al)

Keep local symbols in output file (--output_all_syms, -as)

Do not generate .clink for .const sections (--no_const_clink)

Simulate source '.copy filename' (--copy_file, -ahc)

Symbol names are not case-significant (--syms_ignore_case, -ac)

Use unified assembly language (--ual)

Undefine assembly symbol NAME (--asm_undefine, -au)

Simulate source '.include filename' (--include_file, -ahi)

Pre-define assembly symbol NAME (--asm_define, -ad)

OS_N_CORE=3

Generate first-level assembly include file list (--asm_includes, -api)

Generate cross reference file (--cross_reference, -ax)

Generate assembly dependency information (--asm_dependency, -apd)

Figure 2-6 GUI set of `OS_N_CORE`

NOTE: mAbassi can be configured to operate as the single core Abassi by setting `OS_N_CORE` to 1, or setting `OS_MP_TYPE` to 0 or 1. When configured for single core on the Cortex-M3 MPCore, the application must execute on core #0.

3 Interrupts

The mAbassi 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 mAbassi 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 241 sources of interrupts, as specified by the token `OS_N_INTERRUPTS` in the file `mAbassi_CortexM3_CCS`.s, and the internal default value used by `mAbassi.c`. Even though the Nested Vectored Interrupt Controller (NVIC) peripheral supports a maximum of 256 interrupts on the Cortex-M3, 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 `mAbassi_CortexM3_CCS.s`, the ARM interrupt table itself must be shrunk, and the value used in the file `mAbassi.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 `mAbassi_CortexM3_CCS.s` around line 35. For the value used by `mAbassi.c`, the default value can be overloaded by defining the token `OS_N_INTERRUPTS` when compiling `mAbassi.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 Cortex-M3s on the OMAP4460 device from Texas Instruments use only the first 80 entries of the interrupt table (64 external interrupts plus the standard 16 exceptions). The 256 entry table can therefore be reduced to 80. The value to set in `mAbassi_CortexM3_CCS.s files` is 65, which is the total of 80 entries minus 15 (there are 15 hard mapped exceptions). The change is shown in the following table:

Table 3-1 `mAbassi_CortexM3_CCS.s` interrupt table sizing

```
   …

  .if !($$defined(OS_N_INTERRUPTS))      ; # of entries in the interupt table mapped to
OS_N_INTERRUPTS    .equ 65               ; ISRdispatch()
  .endif

   …
```

Alternatively, it is possible to overload the `OS_N_INTERRUPTS` value set in `mAbassi_CORTEXM3_CCS.s` by using the compiler command line option `–asm_define` and specifying the desired setting with the following:

Table 3-2 Command line set the interrupt table size

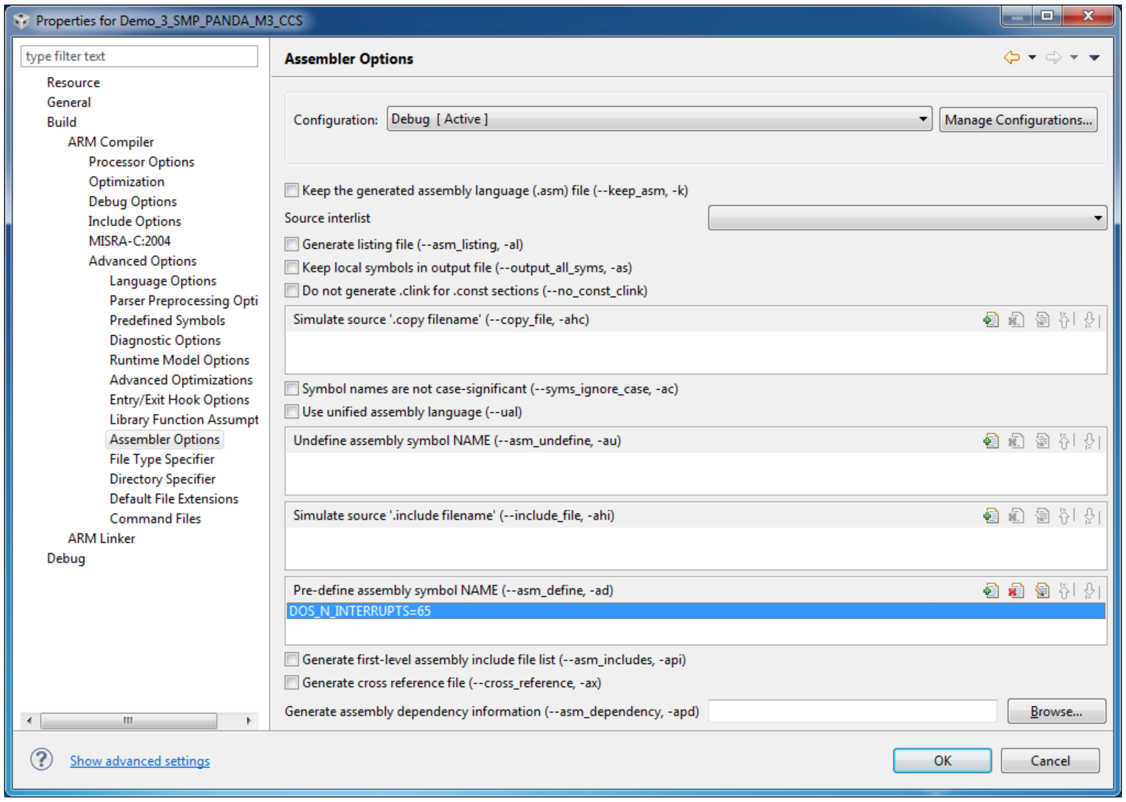
```
Cl470 … -asm_define=OS_N_INTERRUPTS=65 …
```

The overloading of the default interrupt vector look-up table used by `mAbassi.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 `mAbassi.c`

```
cl470  … -DOS_N_INTERRUPTS=65 …
```

The interrupt table size used by `mAbassi_CORTEXM3_CCS.s` can also be set through the GUI, in the "*Build / ARM Compiler / Advanced Options / Assembler Options*" menu, as shown in the following figure:


Properties for Demo_3_SMP_PANDA_M3_CCS

Assembler Options

Configuration: Debug [Active]

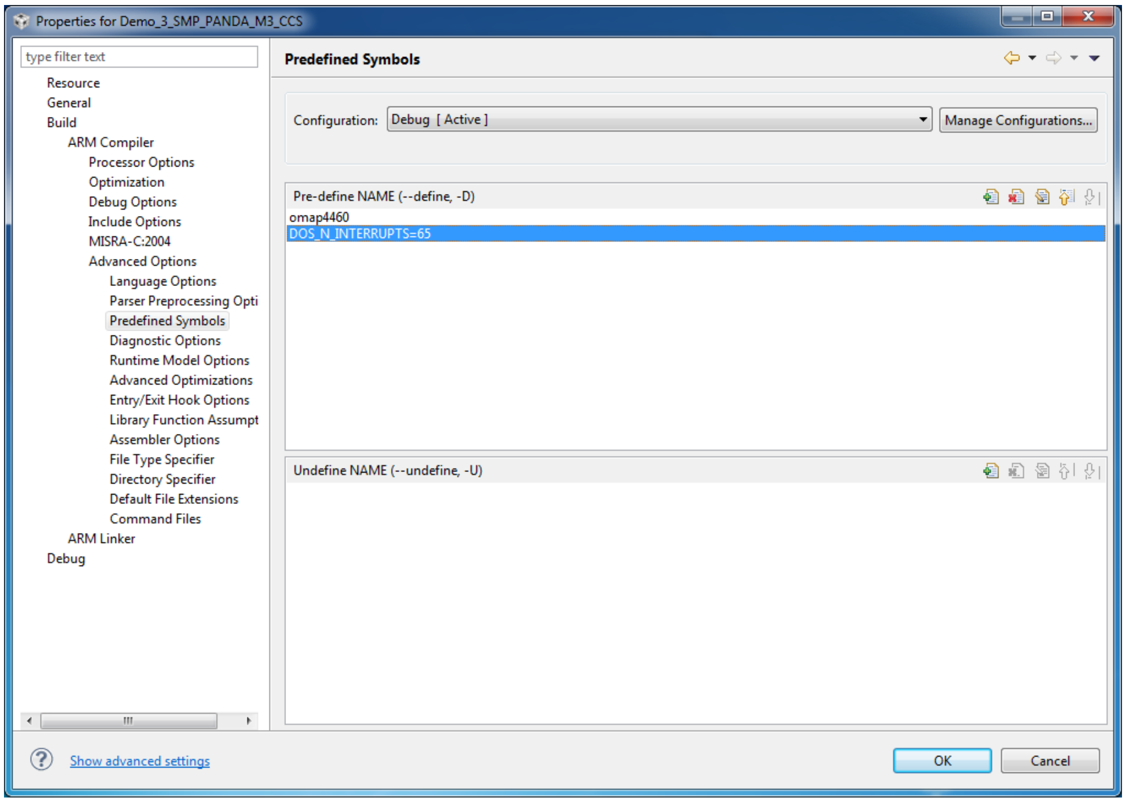
Pre-define assembly symbol NAME (--asm_define, -ad)

DOS_N_INTERRUPTS=65

type filter text
Resource
General
Build
ARM Compiler
Processor Options
Optimization
Debug Options
Include Options
MISRA-C:2004
Advanced Options
Language Options
Parser Preprocessing Opti
Predefined Symbols
Diagnostic Options
Runtime Model Options
Advanced Optimizations
Entry/Exit Hook Options
Library Function Assumpt
Assembler Options
File Type Specifier
Directory Specifier
Default File Extensions
Command Files
ARM Linker
Debug
Manage Configurations...
Keep the generated assembly language (.asm) file (--keep_asm, -k)
Source interlist
Generate listing file (--asm_listing, -al)
Keep local symbols in output file (--output_all_syms, -as)
Do not generate .clink for .const sections (--no_const_clink)
Simulate source '.copy filename' (--copy_file, -ahc)
Symbol names are not case-significant (--syms_ignore_case, -ac)
Use unified assembly language (--ual)
Undefine assembly symbol NAME (--asm_undefine, -au)
Simulate source '.include filename' (--include_file, -ahi)
Generate first-level assembly include file list (--asm_includes, -api)
Generate cross reference file (--cross_reference, -ax)
Generate assembly dependency information (--asm_dependency, -apd)
Browse...
Show advanced settings
OK
Cancel

Figure 3-1 GUI set of `OS_N_INTERRUPTS`

The interrupt table look-up size used by `mAbassi.c` can also be overloaded through the GUI, in the "*Build / ARM Compiler / Advance Options / Predefined Symbols*" menu, as shown in the following figure:


Properties for Demo_3_SMP_PANDA_M3_CCS

Predefined Symbols

Configuration: Debug [Active]

Pre-define NAME (--define, -D)

```
omap4460
DOS_N_INTERRUPTS=65
```

Undefine NAME (--undefine, -U)

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 "mAbassi.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 "mAbassi.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-M3 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.

One has to remember the mAbassi interrupt table is shared across all the cores. Therefore, if the same interrupt number is used on multiple cores, but the processing is different amongst the cores, a single function to handle the interrupt must be used in which the core ID controls the processing dispatch. The core ID is obtained through the `COREgetID()` component of mAbassi. One example of such situation is if the private timer is used on each of two cores, but each core timer has a different purpose, e.g.:

1- on one core, it is the RTOS timer base

2- on the other core, it is the real-time clock tick.

At the application level, when the core ID is used to select specific processing, a critical region exists that must be protected by having the interrupts disabled (see mAbassi User's Guide [R1]). But within an interrupt handler, as nested interrupts are not supported for the Cortex M3, there is no need to add a critical region protection, as interrupts are disabled when processing an interrupt.

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, Code Composer Studio 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 mAbassi, 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-M3 processor. The area of the interrupt vector table to modify is located in the file `mAbassi_CORTEXM3_CCS.s` around line 70. For example, on a Texas Instruments OMAP4460 device, CTM timer #1 is attached to interrupt number 2 (interrupt vector number 18) and the CTM timer #2 is attached to the interrupt number 6 (interrupt vector number 22). The code to modify is located in the macro loop that initializes the interrupt table to set 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

```
.eval  -1, INT_NMB
.loop OS_N_INTERRUPTS            ; Map all external interrupts to ISRdispatch()
   .field   ISRdispatch, 32
   .eval    INT_NMB+1, INT_NMB
.endloop
```

Attaching a fast interrupt handler to the CTM timer #1 and another one to CTM timer #2, assuming the names of the interrupt functions to attach are respectively `CTM1_IRQhandler()` and `CTM2_IRQhandler()`, is shown in the following table:

Table 3-7 OMAP4460 CTM 1 / 2 Fast Interrupts

```
 .ref  CTM1_IRQhandler
 .ref  CTM2_IRQhandler

 …

.eval  -1, INT_NMB
.loop OS_N_INTERRUPTS                 ; Map all external interrupts to ISRdispatch()
     .if INT_NMB == 2                 ; When is interrupt #2, set CTM #1 handler
        .long  CTM1_IRQhandler
     .elseif INT_NMB == 6             ; When is interrupt #6, set CTM #2 handler
        .long  CTM2_IRQhandler
     .else                            ; All others interrupt # set to ISRdispatch()
        .field  ISRdispatch, 32
   .endif
   .eval    INT_NMB+1, INT_NMB
.endloop

 …
```

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

NOTE: If an mAbassi 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 CTM #1 on the OMAP4460 device, this would look something like:

Table 3-8 Fast Interrupt with Dedicated Stack

```
    …

        .if INT_NMB == 2                 ; When is interrupt #1, set CTM #1 handler
           .long  CTM1prehandler

    …
    …

    .text
    .align   4
    .thumb

    .ref  UART0handler

CTM1preHandler:
    cpsid   I                               ; Disable ISR to protect against nesting
    mov     r0, sp                          ; Memo current stack pointer
    ldr     sp, $$CTM1_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      CTM1handler                     ; 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

$$UART0_stack:
    .field  CTM1_s_base+CTM1_stack_size, 32

    .bss    CTM1_s_base, CTM1_stack_size, 8 ; Room for the fast interrupt 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 in 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-M3. If the latter condition is guaranteed, the overloading located after the pre-processor directive can be modified. The code affected in `mAbassi.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(__TI_COMPILER_VERSION__) && defined(__TI_TMS470_V7M3__)
  #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(__TI_COMPILER_VERSION__) && defined(__TI_TMS470_V7M3__)
  #define OX_NESTED_INTS OS_NESTED_INTS
```

The mAbassi 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-M3, 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	Context save
Blocked/Preempted task context save	`40 bytes`
Interrupt dispatcher context save (`OS_ISR_STACK == 0`)	`40 bytes`
Interrupt dispatcher context save (`OS_ISR_STACK != 0`)	`48 bytes`[1]

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, add to all this the stack required by the code implementing the task operation, or the interrupt operation.

NOTE: The ARM Cortex-M3 processor needs alignment on 8 byes for some instructions accessing memory. When stack memory is allocated, mAbassi 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.

[1] This number included the 8 word context save performed by the processor upon servicing an interrupt.

5 Memory Configuration

The mAbassi kernel is not a kernel entered though a service request, such as the SVC on the Cortex-M3. The kernel is a regular function, protected against re-entrance or multiple core entrance. The kernel code executes as part of the application code, with the same processor mode and access privileges.

A fair amount of the effort to use an embedded RTOS on a multi-core platform involves configuring the cache and sharing of the memory. As a starting point, because the kernel is used by all the tasks in the application and, assuming SMP, not BMP, the task can execute on any core, this implies that the whole application code, including the mAbassi code, must share the memory. From a data point of view, exactly the same applies. From a cache point of view, the Cortex-M3 caches are coherent, so caching can be used, except that there is a single variable (`G_OSstate`) that needs to be non-cached, as the `ldrex` & `strex` instructions are used to give mutually exclusive access to the kernel amongst the different cores. The distribution does not treat this variable differently than the rest as it was determined that as a starting point, the mAbassi RTOS should be brought up and running on the target platform with caching disabled and with full memory sharing. Doing so eliminates many issues. Then, once the RTOS is up and running, the designer can start modifying the caching and sharing set-up according to the needs of the application.

6 Search Set-up

The search results are identical to the single core Cortex-M3 port as Abassi and mAbassi use the same code for the search algorithm. Please refer to the single core Cortex-M3 port document [R2] for the measurements.

7 API

The ARM Cortex-M3 supports multiple types of exceptions. Defaults exception handlers are supplied with the distribution code, but each one of them can be overloaded by an application specific function. The default handlers are simply an infinite loop (except SVC, which is a do-nothing with return). The choice of an infinite loop was made as this allows full debugging, as all registers are left untouched by the defaults handlers. The following sub-sections describe each one of the default exception handlers.

7.1 BusFault_Handler

Synopsis

```
#include “mAbassi.h”

void BusFault_Handler(void);
```

Description

`BusFault_Handler()` is the exception handler for a bus fault. In the distribution code, this is implemented as an infinite loop. If the application needs to perform special processing when a data fault occurs, all there is to do is to include a function with the above function prototype, and it will overload the supplied bus fault handler.

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

See also

`HardFault_Handler()` (Section 7.2)
`MemMenage_Handler()` (Section 7.3)
`NMIfault_Handler()` (Section 7.4)
`SVC_Handler()` (Section 7.5)
`UsageFault_Handler()` (Section 7.6)

7.2 HardFault_Handler

Synopsis

```
#include “mAbassi.h”

void HardFault_Handler(void);
```

Description

`HardFault_Handler()` is the handler for a hardware fault. In the distribution code, this is implemented as a return only. If the application needs to perform special processing when a data fault occurs, all there is to do is to include a function with the above function prototype, and it will overload the supplied memory fault handler.

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

See also

`BusFault_Handler()` (Section 7.1)
`MemMenage_Handler()` (Section 7.3)
`NMIfault_Handler()` (Section 7.4)
`SVC_Handler()` (Section 7.5)
`UsageFault_Handler()` (Section 7.6)

7.3 MemManage_Handler

Synopsis

```
#include “mAbassi.h”

void MemManage_Handler(void);
```

Description

`MemManage_Handler()` is the exception handler for a memory management fault. In the distribution code, this is implemented as an infinite loop. If the application needs to perform special processing when a pre-fetch fault occurs, all there is to do is to include a function with the above function prototype and it will overload the supplied memory management fault handler.

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

See also

`BusFault_Handler()` (Section 7.1)
`HardFault_Handler()` (Section 7.2)
`NMIfault_Handler()` (Section 7.4)
`SVC_Handler()` (Section 7.5)
`UsageFault_Handler()` (Section 7.6)

7.4 NMIfault_Handler

Synopsis

```
#include "mAbassi.h"

void NMIfault_Handler(void);
```

Description

`NMIfault_Handler()` is the exception handler for the non-maskable interrupt In the distribution code, this is implemented as an infinite loop. If the application needs to perform special processing when a pre-fetch fault occurs, all there is to do is to include a function with the above function prototype and it will overload the supplied non-maskable interrupt handler.

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

See also

`BusFault_Handler()` (Section 7.1)
`HardFault_Handler()` (Section 7.2)
`MemMenage_Handler()` (Section 7.3)
`SVC_Handler()` (Section 7.5)
`UsageFault_Handler()` (Section 7.6)

7.5 SVC_Handler

Synopsis

```
#include "mAbassi.h"

void SVC_Handler(int SVCnmb, int *Stack);
```

Description

`SVC_Handler()` is the exception handler for service calls that are not handled or reserved by mAbassi. The number of the service call is passed through the function argument `SVCnmb`. The sevrice call context save, holding r0, r1, r2, r3, r12, lr, pc and xPSR is accessible through the argument `Stack`.

Availability

Always.

Arguments

`SWInmb`	Number of the software interrupt. The interrupt numbers 0 to 7 must not be used by the application as they are used / reserved by the RTOS.
`Stack`	Exception context save. The saved registers are accessible using the following: `Stack[0]` : R0 `Stack[1]` : R1 `Stack[2]` : R2 `Stack[3]` : R3 `Stack[4]` : R12 `Stack[5]` : LR `Stack[6]` : PC `Stack[7]` : xPSR

Returns

`void`

Component type

Function

Options

Notes

See also

`BusFault_Handler()` (Section 7.1)
`HardFault_Handler()` (Section 7.2)
`MemMenage_Handler()` (Section 7.3)
`NMIfault_Handler()` (Section 7.4)
`UsageFault_Handler()` (Section 7.6)

7.6 UsageFault_Handler

Synopsis

```
#include “mAbassi.h”

void UsageFault_Handler(void);
```

Description

`UsageFault_Handler()` is the exception handler for a usage access fault. In the distribution code, this is implemented as an infinite loop. If the application needs to perform special processing when a usage fault occurs, all there is to do is to include a function with the above function prototype and it will overload the supplied usage fault handler.

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

See also

`BusFault_Handler()` (Section 7.1)
`HardFault_Handler()` (Section 7.2)
`MemMenage_Handler()` (Section 7.3)
`NMIfault_Handler()` (Section 7.4)
`SVC_Handler()` (Section 7.5)

8 Chip Support

No chip support is provided with the distribution code because Code Composer Studio for the ARM supports the Cortex Microcontroller Software Interface Standard (CMSIS). Therefore, all peripherals on the Cortex-M3 can be accessed through the CMSIS. Also, most device manufacturers provide code to configure the peripherals on their devices.

9 Measurements

This section gives an overview of the memory requirements encountered when the RTOS is used on the Cortex-M3 and compiled with Code Composer Studio. No CPU latency measurements are provided simply because latency measurements are highly dependent on 3 factors. Latency depends on how many cores are used, if mAbassi is configured in SMP or BMP, and if the load balancing algorithm is the True or the Packed one. All these possible configurations are one part of the complexity. A second part of the complexity is where the task switch was detected and on which core(s) the task switch will occur due to that change of state. Finally, the third factor is if a core is already executing in the kernel when another needs to enter the kernel. Any combination of these dynamic factors affects differently the CPU latency of mAbassi.

Although the latency measurements are not provided for mAbassi, if one looks for latency measurements affecting everything on one and only one core, then the single core measurements are very representative [R2]. The multi-core mAbassi implementation increases the cycle count by around 5% over the single core.

9.1 Memory

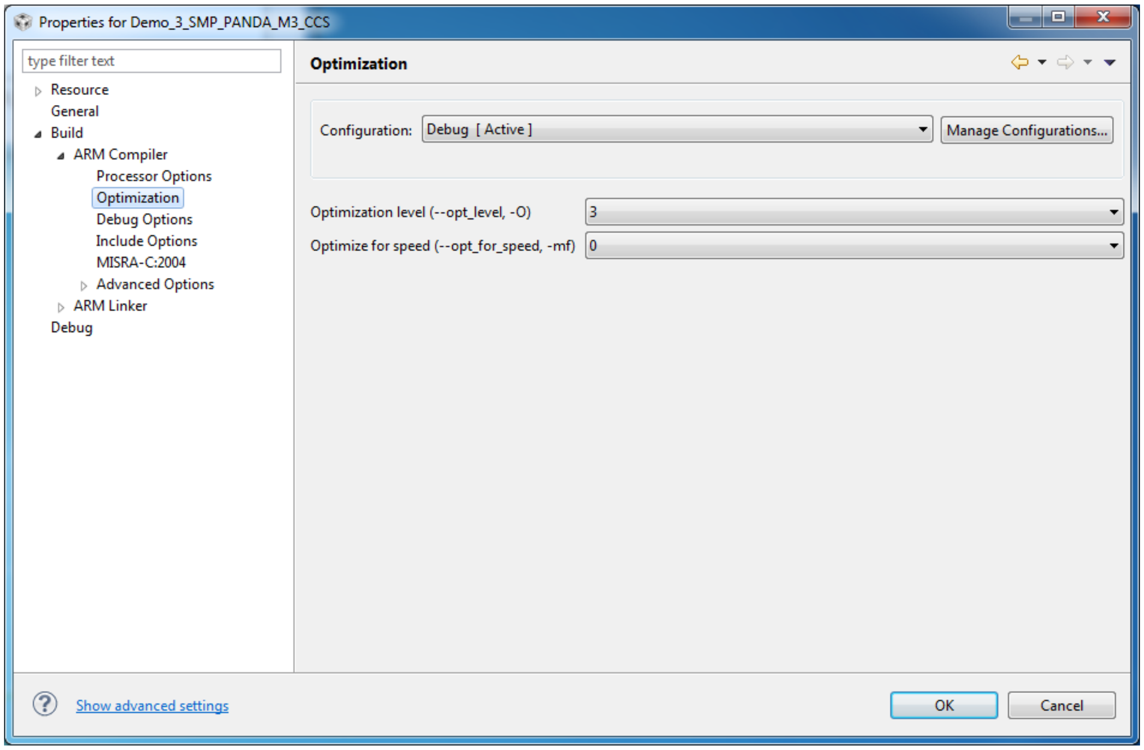
The memory numbers are supplied for the two limit cases of build options (and some in-between): the smallest footprint is the RTOS built with only the minimal feature set, and the other with almost all the features. For both cases, names are not part of the build. This feature was removed from the metrics 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 runtime 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.	Debugging model:	Off[2]
2.	Optimization level:	3[3]
3.	Optimize for speed:	0
4.	Instruction size	16
5.	Target	7M3

Figure 9-1 Debug Options Settings



Properties for Demo_3_SMP_PANDA_M3_CCS

Optimization

Configuration: Debug [Active]

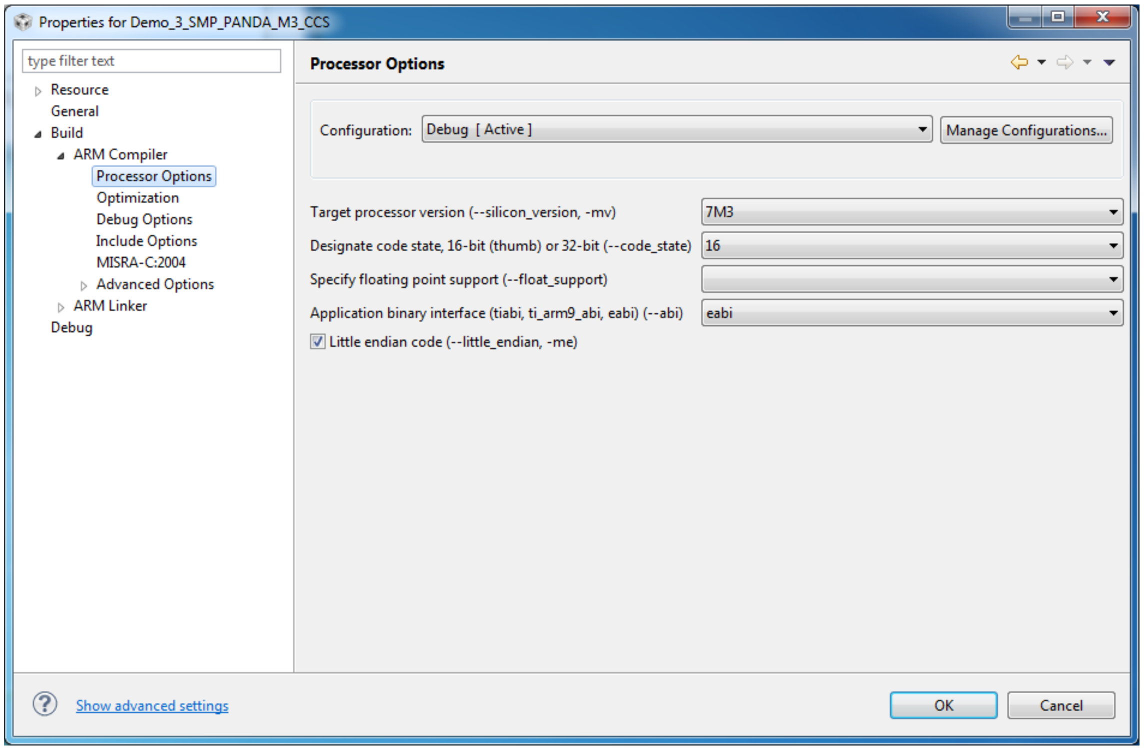
Optimization level (--opt_level, -O) 3

Optimize for speed (--opt_for_speed, -mf) 0

Figure 9-2 Optimization Settings

[2] Debugging is turned off as it restricts the optimizer.

[3] The highest optimization level on Code Composer is 4, but level 4 adds linker optimization over what optimization level 3 does. The linker optimization is not used for the memory measurements as it converts small function into in-line operations, removing these functions from the memory map, skewing the memory sizing measurements.



Configuration: Debug [Active]

Target processor version (--silicon_version, -mv): 7M3

Designate code state, 16-bit (thumb) or 32-bit (--code_state): 16

Specify floating point support (--float_support):

Application binary interface (tiabi, ti_arm9_abi, eabi) (--abi): eabi

[✓] Little endian code (--little_endian, -me)

Figure 9-3 Processor Options Settings

Table 9-1 "C" Code Memory Usage

Description	Code Size
Minimal Build	< 1375 bytes
+ Runtime service creation / static memory	< 1675 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 2200 bytes
+ Timer & timeout + Timer call back + Round robin	< 2950 bytes
+ Events + Mailbox	< 3575 bytes
Full Feature Build (no names)	< 4200 bytes
Full Feature Build (no name / no runtime creation)	< 3800 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 4175 bytes

The selection of load balancing type does not really affect the "C" code size; there is a difference of no more than 4 to 8 bytes between True and Packed load balancing; the latter requiring less code space. In the measurements, True load balancing was used in SMP mode. The same does not apply when selecting BMP instead of SMP. With BMP, the "C" code size increases by around 200 bytes compared to SMP.

Table 9-2 Assembly Code Memory Usage

Description	Size
Assembly code size (>1 core)	`724 bytes`
Assembly code size (==1 core)	`288 bytes`
Vector table (per interrupt handler entry)	`+4 bytes`
Hybrid Stack Enabled	`+24 bytes`
Saturation Bit Enabled	`+24 bytes`

These memory usage numbers are for the OMPA4460 port. Depending on the target device, the following sections of code may slightly change the memory requirements:

Table 9-3 Device dependent code

Description
Start-up code
Setting-up / Accessing the core number
Core initialization
Inter-core interrupt triggering / handling
Spinlock

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 the Code Time Technologies website.

10 Appendix A: Build Options for Code Size

10.1 Case 0: Minimum build

Table 10-1: 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_MP_TYPE          2    /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK     0    /* This test validates this                         */
#define OS_MTX_INVERSION    0    /* To enable protection against priority inversion  */
#define OS_N_CORE           2    /* Number of cores to handle                        */
#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        1    /* 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_STACK_START      256  /* Stack sie of the start-up / ilde functions       */
#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                          */
```

10.2 Case 1: + Runtime service creation / static memory + Multiple tasks at same priority

Table 10-2: 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    32   /* Maximum number of requests in ISRs             */
#define OS_MP_TYPE          2    /* SMP vs BMP and load balancing selection        */
#define OS_MTX_DEADLOCK     0    /* This test validates this                       */
#define OS_MTX_INVERSION    0    /* To enable protection against priority inversion  */
#define OS_N_CORE           2    /* Number of cores to handle                      */
#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_STACK_START      256  /* Stack sie of the start-up / ilde functions       */
#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                        */
```

10.3 Case 2: + Priority change / Priority inheritance / FCFS / Task suspend

Table 10-3: 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             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_MP_TYPE          2    /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK     0    /* This test validates this                         */
#define OS_MTX_INVERSION    1    /* To enable protection against priority inversion  */
#define OS_N_CORE           2    /* Number of cores to handle                        */
#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_STACK_START      256  /* Stack sie of the start-up / ilde functions       */
#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                          */
```

10.4 Case 3: + Timer & timeout / Timer call back / Round robin

Table 10-4: 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_MP_TYPE         2     /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK    0     /* This test validates this                         */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_N_CORE          2     /* Number of cores to handle                        */
#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_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#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                          */
```

10.5 Case 4: + Events / Mailboxes

Table 10-5: 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_MP_TYPE         2     /* SMP vs BMP and load balancing selection           */
#define OS_MTX_DEADLOCK    0     /* This test validates this                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_N_CORE          2     /* Number of cores to handle                         */
#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_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#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                           */
```

10.6 Case 5: Full feature Build (no names)

Table 10-6: 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          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_MP_TYPE         2     /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK    0     /* This test validates this                         */
#define OS_N_CORE          2     /* Number of cores to handle                        */
#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_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#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                          */
```

10.7 Case 6: Full feature Build (no names / no runtime creation)

Table 10-7: 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_MP_TYPE         2     /* SMP vs BMP and load balancing selection        */
#define OS_MTX_DEADLOCK    0     /* This test validates this                       */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_N_CORE          2     /* Number of cores to handle                      */
#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_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#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                        */
```

10.8 Case 7: Full build adding the optional timer services

Table 10-8: 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_MP_TYPE         2     /* SMP vs BMP and load balancing selection         */
#define OS_MTX_DEADLOCK    0     /* This test validates this                        */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_N_CORE          2     /* Number of cores to handle                       */
#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_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#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                         */
```

11 References

[R1] mAbassi RTOS – User Guide, available at http://www.code-time.com
[R2] Abassi Port – Cortex M3, available at http://www.code-time.com