CODE TIME TECHNOLOGIES

mAbassi RTOS

Porting Document

SMP / ARM Cortex-A9 – CCS

Disclaimer

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-A9 processor, commonly known as the Arm9. The port should also be valid for the ARMv4, ARMv5, ARMv6 and ARMv7 core architectures. 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_CORTEXA9_CCS.s`	RTOS assembly file for the SMP ARM Cortex-A9 to use with Code Composer Studio
`Demo_3_SMP_PANDA_A9_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 the `SWI` (software interrupts) numbers 0 to 7. A hook is made available for the application to use the `SWI`, as long as the numbers used are above 7.

Some linker issues have been found when the test suite was run. One should avoid setting the optimization level to 4, as it was found a few times that calls to the library function `strcpy()` were replaced by faulty code. When the optimization is set to 4, it enables the linker to inline function code that is called once, or when in-lining the code produces a smaller than a function call. This optimizer problem is not unique to the Abassi code.

If the TI ABIs (`-abi=ti_arm9_abi` or `-abi=tiabi`) are used, the `COREstart#` functions cannot be overloaded, as these ABIs recognize but do not support the `.weak` declaration in assembler.

The port does not support the ARMv4 with 16 bits (`-code_state=16`) for the TI ABI (`-abi=tiabi`). All other combinations of versions, instruction sizes and ABIs are supported

1.3 Features

Depending on the selected build configuration, the application can operate either in privileged or user mode. Operating in privileged mode eliminates almost all sections that need to disable interrupts, as SWIs are not required to access privileged registers or peripherals. It also generates more real-time optimal code.

Fast Interrupts (FIQ) are not handled by the RTOS, and are left untouched by the RTOS to fulfill their intended purpose of interrupts not requiring kernel access. Only the interrupts mapped to the IRQ interrupt are handled by the RTOS.

The hybrid stack is not available in this port, as ARM's GIC (Generic Interrupt Controller) does not allow nesting of the interrupts (except FIQ nesting the IRQ). The ARM A9 intrinsically supports exactly the same functionality delivered by the hybrid stack. This is because the interrupts (IRQ) use a dedicated stack when in this processor mode.

The assembly file was coded using only ARMv4 non-superseded instructions. This means the RTOS for the Arm9 should also be usable with ARMv4, ARMv5, ARMv6, and ARMv7; so the Arm5, Arm8, Arm9 and the Arm15 are supported with this port.

The assembly file does not use the BL or BLX instructions when branching or calling any module. This was done to allow the assembly file to access the whole 4 GBytes address space.

The RTOS assembly file is coded with 32 bit instructions but co-exists with 16 bits instruction modules, either Thumb, Thumb2, or ThumbEE (Jazelle RCT).

The VFPv3 or VPFv3D16, and NEON, are supported and their registers are optionally saved as part of the task context save and / or interrupt context save.

Every one of the Code Composer application binary interfaces (tiabi, ti_arm9_abi, and eabi) 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_CORTEXA9_CCS.s` in the source files of the application project, and make sure the configuration settings in the file `mAbassi_SMP_CORTEXA9_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_CORTEXA9_CCS.s` takes care of all the start-up operations required for an application operating on a multi-core processor.

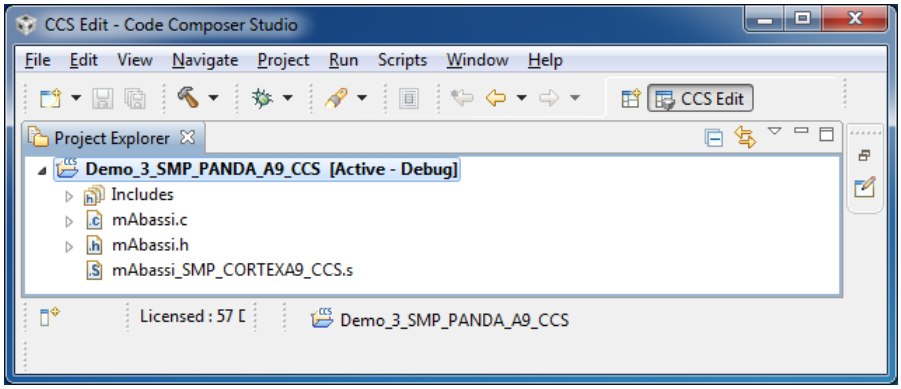

Figure 2-1 Project File List

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 Stacks Set-up

The Cortex Arm9 processor handles 6 individual stacks, which are selected according to the processor mode. The following table describes each stack and the build token used to define the size of the associated stack:

Table 2-1 Stack Size Tokens

Description	Token Name
User / System mode	`N/A`
Supervisor mode	`OS_SUPER_STACK_SIZE`
Abort mode	`OS_ABORT_STACK_SIZE`
Undefined mode	`OS_UNDEF_STACK_SIZE`
Interrupt mode	`OS_IRQ_STACK_SIZE`
Fast Interrupt mode	`OS_FIQ_STACK_SIZE`

The User / System mode 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.

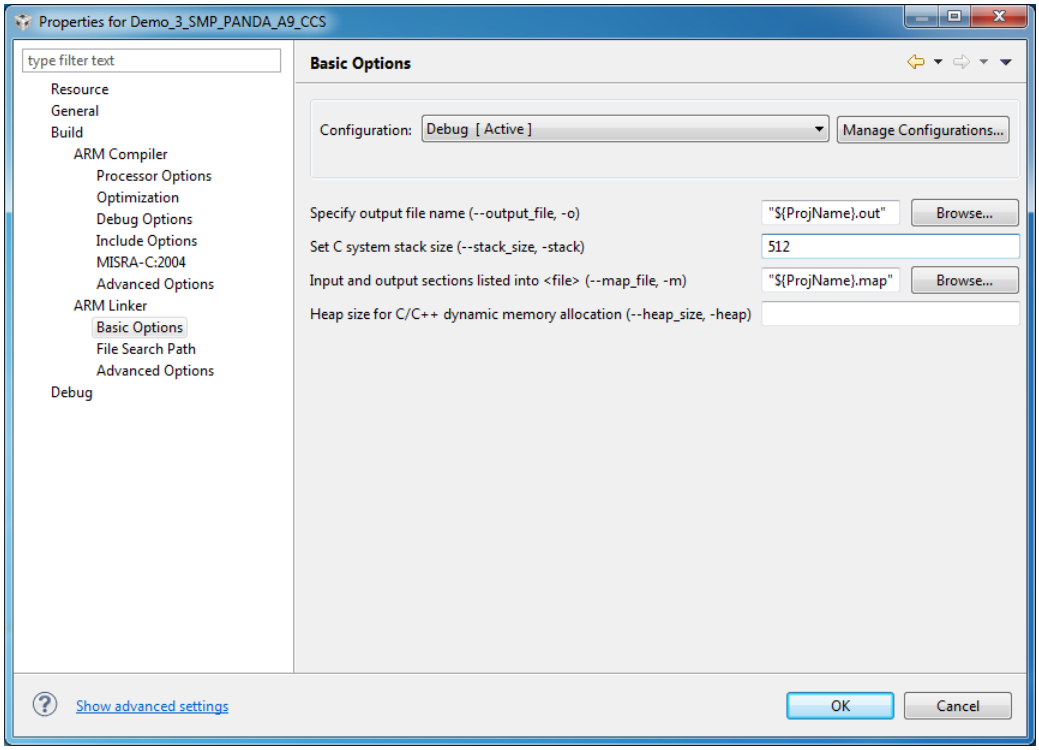

Figure 2-2 GUI setting of Adam & Eve stack size

The other stack sizes are individuality controlled by the values set by the `OS_?????_STACK_SIZE` definitions, located between line 40 and 60 in the file `mAbassi_SMP_CORTEXA9_CCS.s`. To not reserve a stack for a processor mode, set the definition of `OS_?????_STACK_SIZE` to a value of zero. To specify the stack size, set the definition of `OS_?????_STACK_SIZE` to the desired size in bytes (see Section 4 for more information on stack sizing). Note that each core on the device (up to the number specified by the build option `OS_N_CORE`) will use the same stack sizes for a processor mode. This equal distribution of stack size may not be optimal; if a non-equal distribution is required, contact Code Time Technologies for additional information.

To modify the size of a stack, taking the IRQ stack for example and reserving a stack size of 256 bytes for the IRQ processor mode, all there is to do is to change the numerical value associated with the token; this is shown in the following table:

Table 2-2 `OS_IRQ_STACK_SIZE` modification

```
  .if !($$defined(OS_IRQ_STACK_SIZE))
OS_IRQ_STACK_SIZE   .equ 256
  .endif
```

Alternatively, it is possible to overload the `OS_?????_STACK_SIZE` value set in `mAbassi_SMP_CORTEXA9_CCS.s` by using the assembler command line option `-asm_define` and specifying the desired stack size as shown in the following example, where the IRQ stack size is set to 128 bytes:

Table 2-3 Command line set of `OS_IRQ_STACK_SIZE`

```
cl470  … -asm_define=OS_IRQ_STACK_SIZE=128  …
```

The 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:

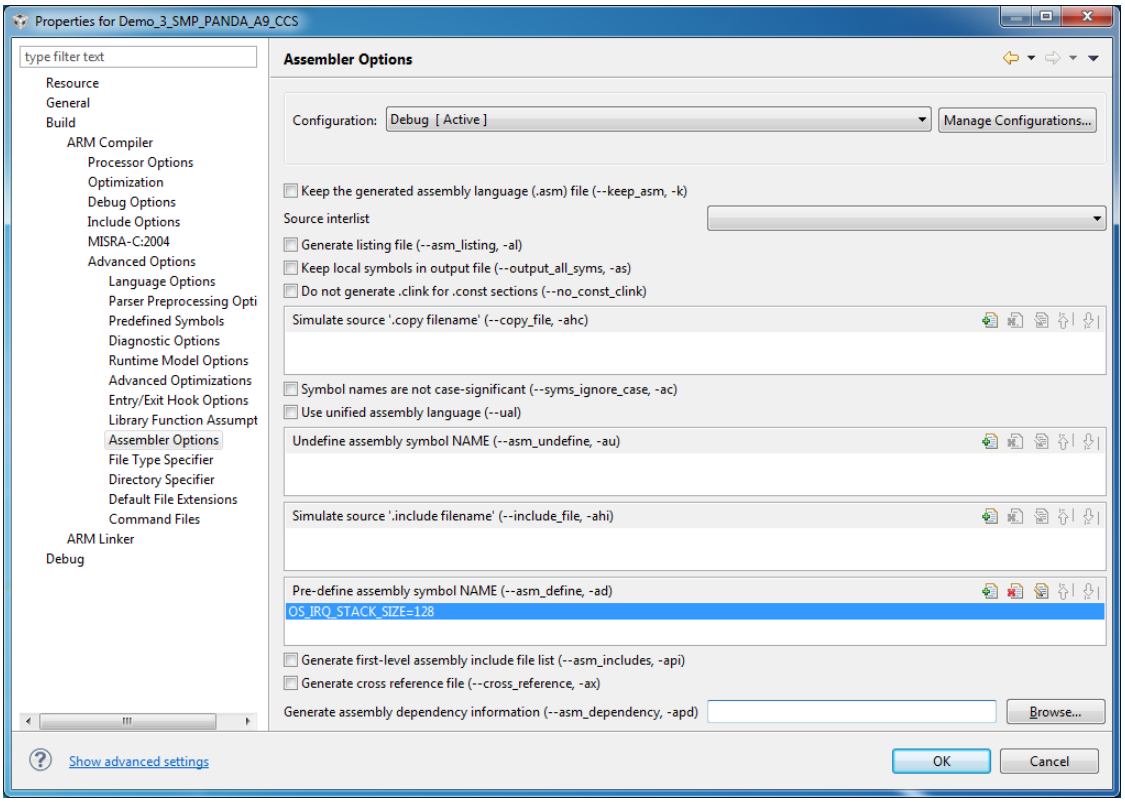

Figure 2-3 GUI set of `OS_IRQ_STACK_SIZE`

2.2 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_CORTEXA9_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_CORTEXA9_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 2-4 `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_CORTEXA9_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 2-5 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:

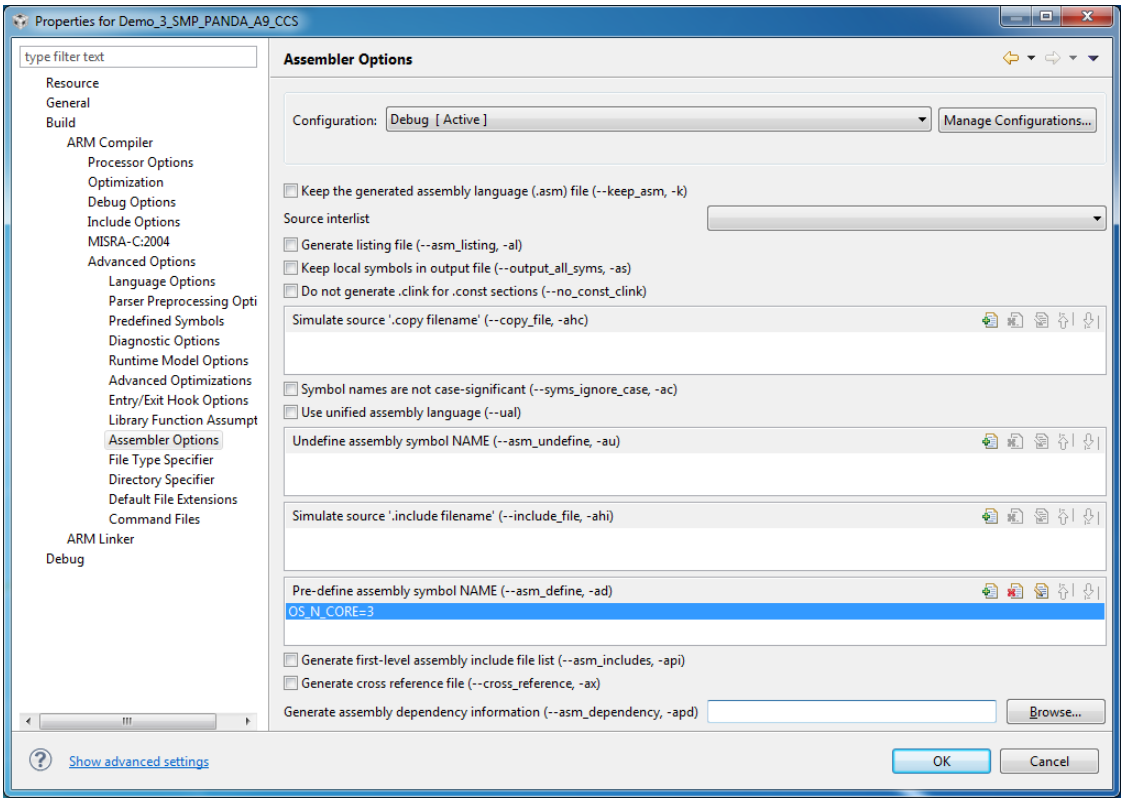

Figure 2-4 GUI set of `OS_N_CORE`

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 A9 MPCore, the application must execute on core #0.

2.3 Privileged mode

It is possible to configure mAbassi for the Cortex A9 MPCore to make the application execute in the USER processor mode (un-privileged) or in the SYS processor mode (privileged). Having the application executing in the SYS processor mode (privileged) delivers two main advantages over having it executing in the USER mode (un-privileged). The first one is, when in the SYS mode, Software interrupts (SWI) are needed to read or write the registers and peripherals only accessible in privileged mode. Having to use SWI disables the interrupts during the time the SWI executes. The second advantage of executing the application in SYS mode is again related to the SWI, but this time it is one of CPU efficiency: the code required to replace the functionality of the SWI is much smaller, therefore less CPU is needed to execute the same operation.

There is no fundamental reason why an application should be executing in the un-privileged mode with mAbassi. First, even though the mAbassi kernel is a single function, it always executes within the application context. There are no service requests, alike the A9 SWI, involved to access the kernel. And second, mAbassi was architected to be optimal for embedded application, where the need to control accesses to peripherals or other resources, as in the case of a server level OS, is not applicable.

Only the file `mAbassi_SMP_CORTEXA9_CCS.s` requires the information if the application executes in the privileged mode or not. To select if the application executes in privileged mode or not, all there is to do is to change the value associated to the definition of the token `OS_RUN_PRIVILEGE`, located around line 30. Associating a numerical value of zero to the build option configures mAbassi to let the application execute in the USER processor mode, which is un-privileged. Associating a numerical value different than zero to the build option configures mAbassi to let the application execute in the SYS processor mode, which is privileged. The latter case is shown in the following table:

Table 2-6 `OS_RUN_PRIVILEGE` modification

```
  .if !($$defined(OS_RUN_PRIVILEGE))
OS_RUN_RPIVILEGE    .equ 1
  .endif
```

Alternatively, it is possible to overload the `OS_RUN_RPIVILEGE` value set in `mAbassi_SMP_CORTEXA9_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 2-7 Command line set of `OS_RUN_PRIVILEGE`

```
cl470  … -asm_define=OS_RUN_PRIVILEGE=1  …
```

The selection between privileged mode or un-privileged mode can also be set through the GUI, in the "*Build / ARM Compiler / Advanced Options / Assembler Options*" menu, as shown in the following figure:

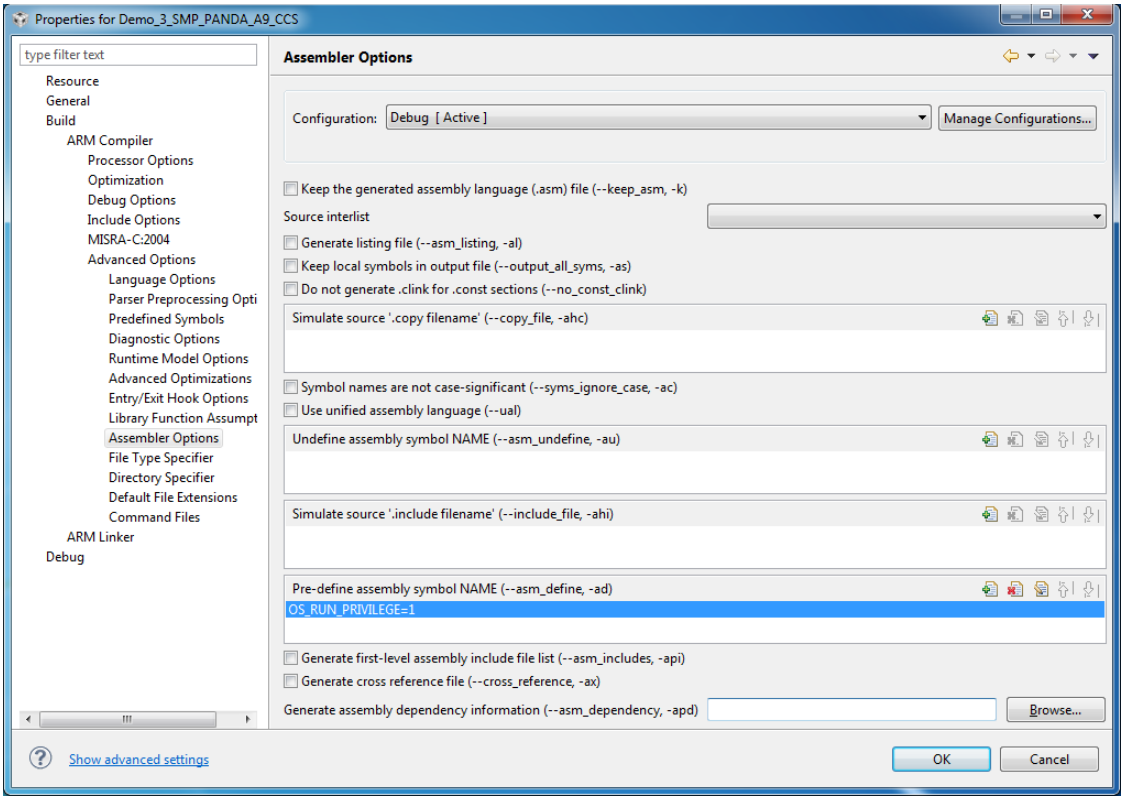

Figure 2-5 GUI set of `OS_RUN_PRIVILEGE`

2.4 Saturation Bit Set-up

In the ARM Cortex-A9 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 extra processing is required 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 the localization, it must be set to a zero value. This is located at around line 35 in the file `mAbassi_SMP_CORTEXA9_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 2-8 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_SMP_CORTEXA9_CCS.s` by using the assembler command line option `-asm_define` and specifying the desired setting with the following:

Table 2-9 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:

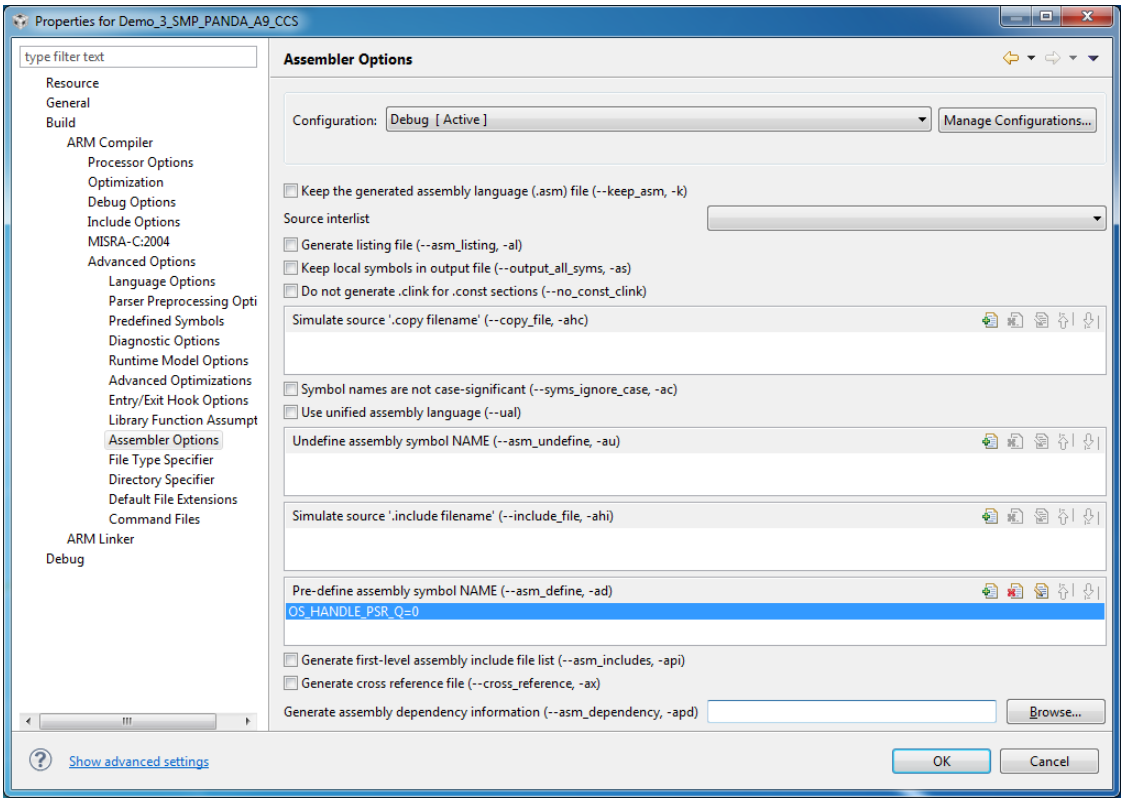

Figure 2-6 GUI set of Saturation Bit configuration

NOTE: The saturation bit is not supported by the ARMv4 architecture. The setting of the build option `OS_HANDLE_PSR_Q` is ignored with ARMv4 architecture.

2.5 VFP / NEON set-up

The assembly file `mAbassi_SMP_CORTEXA9_CCS.s`, depending on its configuration, handle four different types of VFP. They are:

- No VPU coprocessor
- VPUv3FPU
- VPUv3D16
- NEON

The file `mAbassi_SMP_CORTEXA9_CCS.s` is aware of the presence of a VFP, and the type of VFP, when the assembler command line option `-float_support` is used, or when set through the GUI, in the "*Build / ARM Compiler / Processor Options Options*" menu, as shown in the following figure:

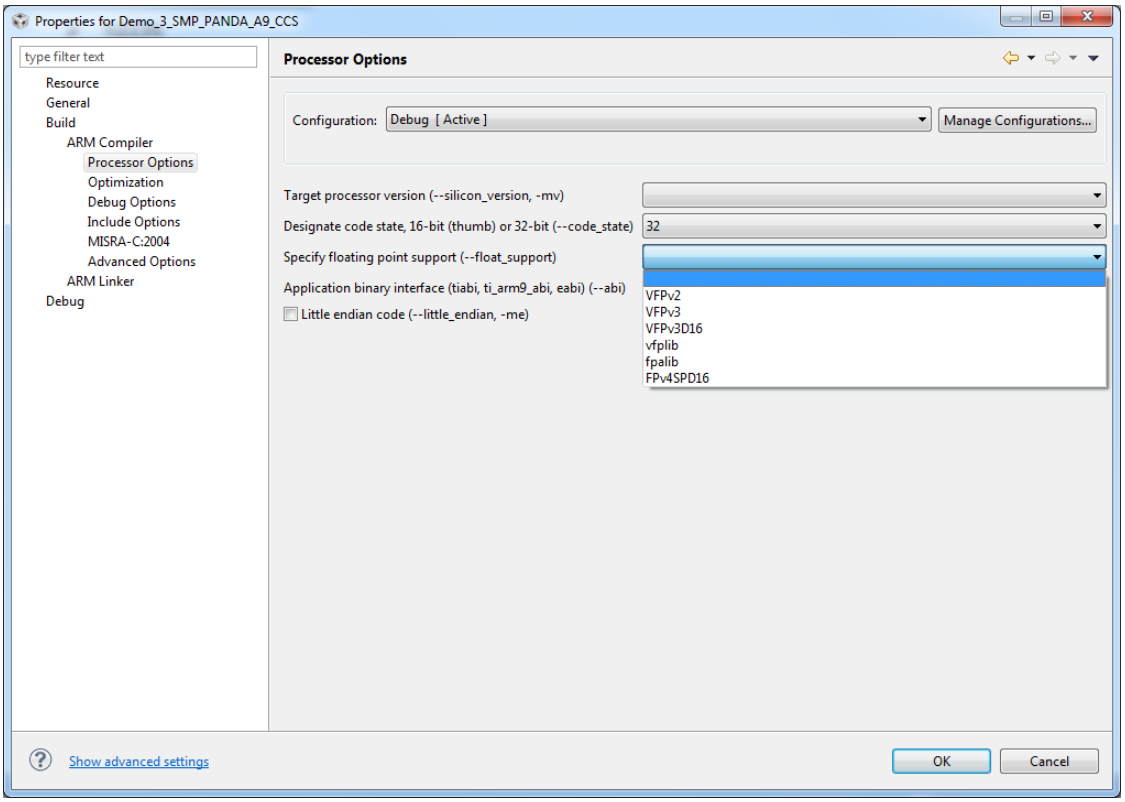

Figure 2-7 GUI enabling of VFP

Table 2-10 Command line enabling of the VFPv3

The following 3 tables show the command line setting for each of the supported floating point units:

```
Cl470 … --float_support=vfpv3 …
```

Table 2-11 Command line enabling of the VFPv3D16

```
Cl470 … --float_support=vfpv3d16 …
```

Table 2-12 Command line enabling of the NEON

```
Cl470 … --neon …
```

3 Interrupts

The mAbassi RTOS needs to be aware when kernel requests are performed inside or outside an interrupt context. For all IRQ sources 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 256 sources of interrupts, as specified by the token `OX_N_INTERRUPTS` in the file `mAbassi.h`[1], and the value of `OX_N_INTERRUPTS` is the internal default value used by `mAbassi.c`. Even though the Generic Interrupt Controller (GIC) peripheral supports a maximum of 1024 interrupts, it was decided to set the distribution value to 256, as this seems to be a typical maximum supported by the different devices on the market.

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; or some very large device require more than 256. The interrupt table can be easily reduced to recover data space. All there is to do is to define the build option `OS_N_INTERRUPTS` (`OS_N_INTERRUPTS` is used to overload the internal value of `OX_N_INTERRUPTS`) to the desired value. This can be done by using the compiler command line option `-D` and specifying the desired setting with the following:

Table 3-1 Command line set the interrupt table size

```
Cl470 … -d=OS_N_INTERRUPTS=49 …
```

[1] This is located in the port-specific definition area.

The interrupt table look-up size can also be set through the GUI, in the "*Build / ARM Compiler / Advance Options / Predefined Symbols*" menu, as shown in the following figure:

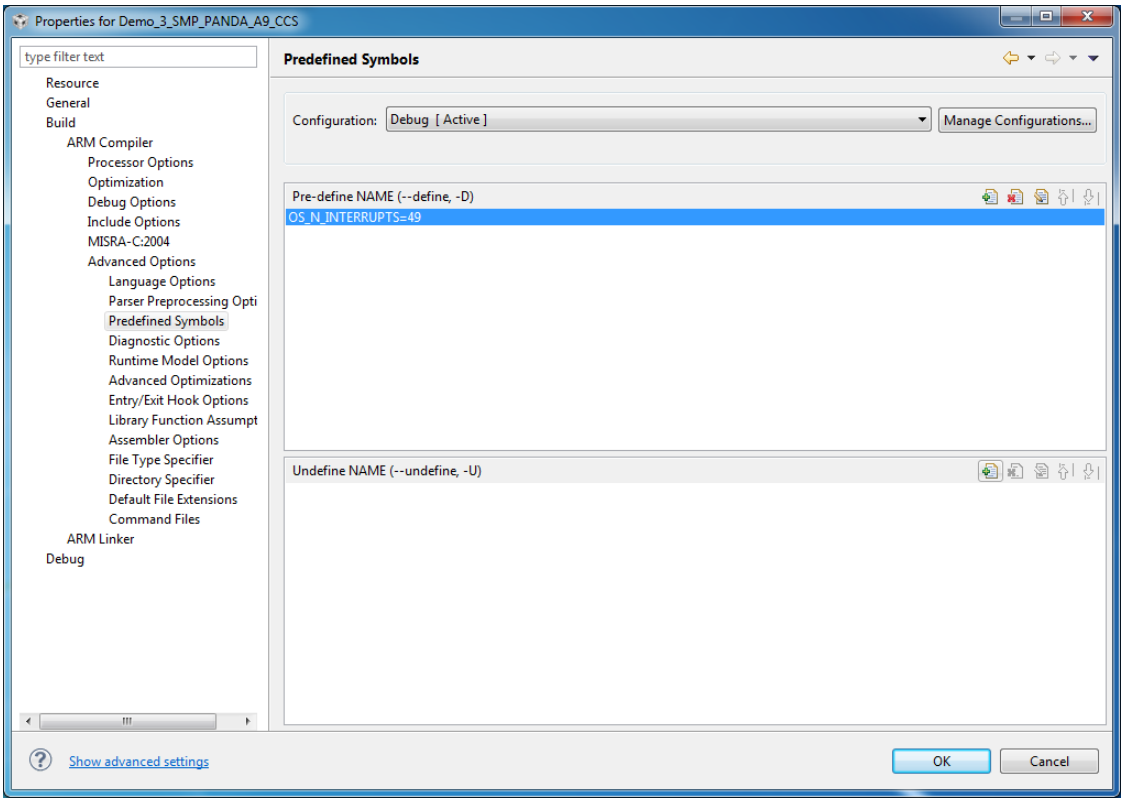

Figure 3-1 GUI set of `OS_N_INTERRUPTS`

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 3-2 shows the code required to attach the private timer interrupt on an OMAP4460 (ID #29) to the RTOS timer tick handler (`TIMtick`):

Table 3-2 Attaching a Function to an Interrupt

```
#include "mAbassi.h"

  …
  OSstart();
  …
  OSisrInstall(29, &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` (truly `OX_N_INTERRUPTS` if not overloaded) 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 executed. 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 3-3:

Table 3-3 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 interrupt number indicated by Generic Interrupt Controller (GIC) is acknowledged by the ISR dispatcher, but the dispatcher does not remove the request by a peripheral if the peripheral generate a level interrupt.

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 A9, there is no need to add a critical region protection, as interrupts are disabled when processing an interrupt.

3.2 Fast Interrupts

Fast interrupts are supported on this port as the FIQ interrupts. The ISR dispatcher is designed to only handle the IRQ interrupts. A default do-nothing FIQ handler is supplied with the distribution; the application can overload the default handler (Section 7.2).

3.3 Nested Interrupts

Interrupt nesting, other than a FIQ nesting an IRQ, is not supported on this port. The reason is simply based on the fact the Generic Interrupt Controller is not a nested controller. Supporting nesting becomes real-time inefficient as the interrupt context save is not stack based, but register bank based.

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-A9, 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	48 bytes
Blocked/Preempted task context save / VFP enable (VFPv3 or VFPv3D16)	112 bytes
Interrupt dispatcher context save (IRQ stack)	48 bytes
Interrupt dispatcher context save (User Stack)	64 bytes
Interrupt dispatcher context save (User Stack) / VFPv3D16 enable	128 bytes
Interrupt dispatcher context save (User Stack) / VFPv3 enable	256 bytes

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-A9 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.

5 Memory Configuration

The mAbassi kernel is not a kernel entered though a service request, such as the SWI on the Cortex-A9. 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-A9 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.

The memory caching / sharing set-up is done in the file `mAbassi_SMP_CORTEXA9_CCS.s` around line 190.

6 Search Set-up

The search results are identical to the single core Cortex-A9 port as Abassi and mAbassi use the same code for the search algorithm. Please refer to the single core Cortex-A9 port document [R2] for the measurements.

7 API

The ARM Cortex-A9 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 FIQ, which is a do-nothing with return from exception). 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 DATAabort_Handler

Synopsis

```
#include "mAbassi.h"

void DATAabort_Handler(void);
```

Description

`DATAabort_Handler()` is the exception handler for a data abort 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 data abort handler. As this is an exception, the return must be performed with a "`movs pc, lr`".

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

This is an exception function, executing in the abort processor mode. This means the abort stack is in use instead of the user stack, and the IRQ interrupts are disabled.

See also

`FIQ_Handler()` (Section 7.2)
`PFabort_Handler()` (Section 7.3)
`SWI_Handler()` (Section 7.4)

7.2 FIQ_Handler

Synopsis

```
#include "mAbassi.h"

void FIQ_Handler(void);
```

Description

`FIQ_Handler()` is the handler for a fast interrupt request. In the distribution code, this is implemented as a return only. If the application needs to handle fast interrupts, all there is to do is to include a function with the above function prototype and it will overload the supplied fast interrupt handler. As this is an exception, the return must be performed with a "`movs pc, lr`".

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

This is an exception function, executing in the FIQ processor mode. This means the FIQ stack is in use instead of the user stack, and the FIQ are now disabled and IRQ interrupts are also disabled.

See also

`DATAabort_Handler()` (Section 7.1)
`PFabort_Handler()` (Section 7.3)
`SWI_Handler()` (Section 7.4)

7.3 PFabort_Handler

Synopsis

```
#include "mAbassi.h"

void PFabort_Handler(void);
```

Description

`PFabort_Handler()` is the exception handler for a pre-fetch abort 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 data abort handler. As this is an exception, the return must be performed with a "`movs pc, lr`".

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

This is an exception function, executing in the abort processor mode. This means the abort stack is in use instead of the user stack, and the IRQ interrupts are disabled.

See also

`DATAabort_Handler()` (Section 7.1)
`FIQ_Handler()` (Section 7.2)
`SWI_Handler()` (Section 7.4)

7.4 SWI_Handler

Synopsis

```
#include "mAbassi.h"

void SWI_Handler(int SWInmb);
```

Description

`SWI_Handler()` is the exception handler for software interrupts that are not handled or reserved by mAbassi. The number of the software interrupt is passed through the function argument `SWInmb`. This is a regular function; do not use the exception instruction "`movs pc, lr`".

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.

Returns

`void`

Component type

Function

Options

Notes

This is a regular function, but executing in the supervisor processor mode. This means the supervisor stack is in use instead of the user stack, and the IRQ interrupts are disabled.
If the TI ABIs (`-abi=ti_arm9_abi`) or (`-abi=tiabi`) are used when overloading `SWI_Handler()`, the section of code implementing `SWI_Handler()` in the file `mAbassi_SMP_CORTEXA9_CCS.s` must be either commented or removed as these ABIs recognize but do not support the `.weak` declaration in assembler

See also

`DATAabort_Handler()` (Section 7.1)
`FIQ_Handler()` (Section 7.2)
`FPabort_Handler()` (Section 7.3)

8 Chip Support

Basic support for the Generic Interrupt Controller (GIC) is provided in this port as SMP/BMP multi-core on the Cortex-A9 MPCore device requires the use of interrupts. The following sub-sections describe the two support components.

8.1 GICenable

Synopsis

```
#include "mAbassi.h"

void GICenable(int IntNmb);
```

Description

`GICenable()` is the component used to enable an interrupt number (called ID*nn* in the literature) on the Generic Interrupt Controller (GIC). The interrupt configuration is the following:

- Level sensitive
- Mapped to the core where `GICenable()` is excuting.

Availability

Always.

Arguments

`IntNmb` Interrupt number to enable

Returns

`void`

Component type

Function

Options

Notes

On the Cortex-A9 MPCore, some GIC registers are local to the core, while others are global across all cores. Care must be taken when using `GICenable()`.
When the interrupt number (argument `IntNmb`) is non-negative, then the GIC is programmed to target the interrupt to the core it's currently operating on. If the interrupt number is negative, then the interrupt number `-IntNmb` is targeted to all cores.
The function `GICenable()` is implemented in assembly language, in the file `mAbassi_SMP_CORTEXA9_CCS.s` as it avoids supplying 2 port files in the distribution. If the supplied functionality does not fulfill the application needs, `GICenable()` can be overloaded by adding a new `GICenable()` function in the application. The equivalent "C" code of the distribution implementation is supplied in comments in the assembly file.

See also

`GICinit()` (Section 8.2)

8.2 GICinit

Synopsis

```
#include "mAbassi.h"

void GICinit(void);
```

Description

`GICinit()` is the component used to initialize the Generic Interrupt Controller (GIC) for the needs of mAbassi. It must be used after using the `OSstart()` component and before `GICenable()` and / or `OSeint()` components. Also, it must be used in every `COREstartN()` function.
Consult the mAbassi User guide for more information on this topic [R1].

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

The function `GICinit()` is implemented in assembly language, in the file `mAbassi_SMP_CORTEXA9_CCS.s` as it avoids supplying 2 port files in the distribution. If the supplied functionality does not fulfill the application needs, `GICinit()` can be overloaded by adding a new `GICinit()` function in the application. The equivalent "C" code of the distribution implementation is supplied in comments in the assembly file.

See also

`GICenable()` (Section 8.1)

9 Measurements

This section gives an overview of the memory requirements encountered when the RTOS is used on the Arm9e 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 5e

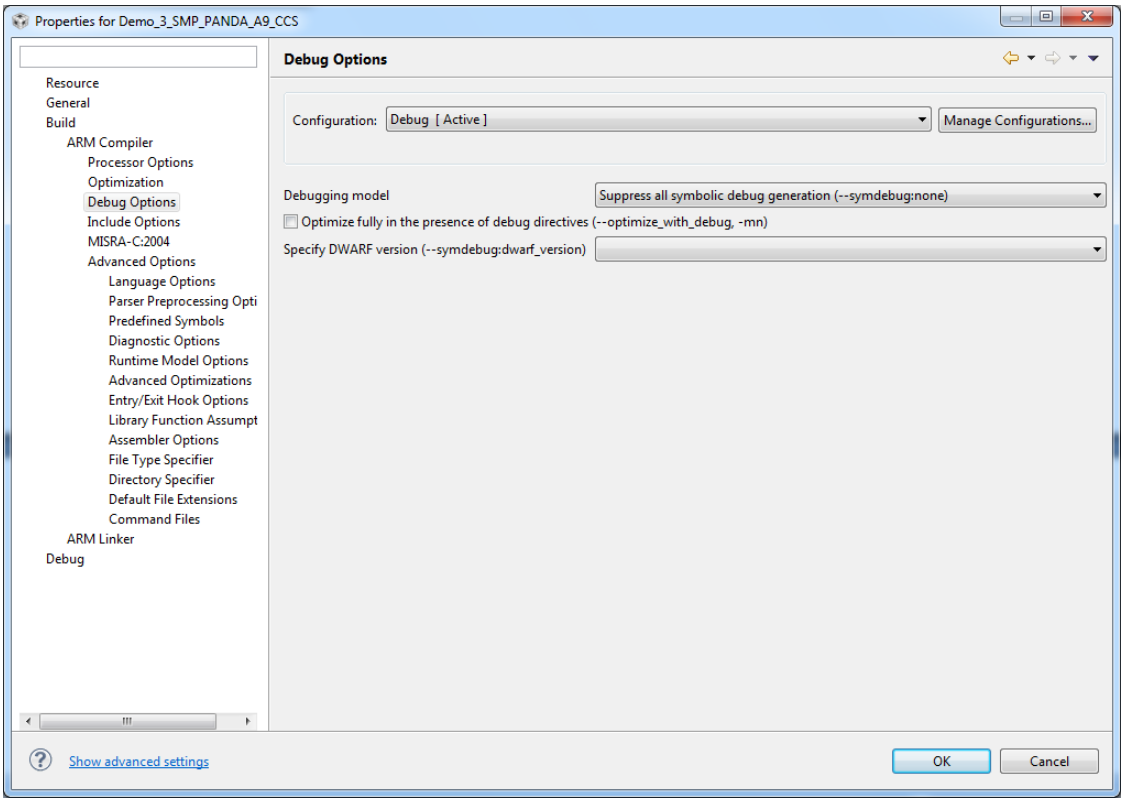

Figure 9-1 Debug Options Settings

[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.

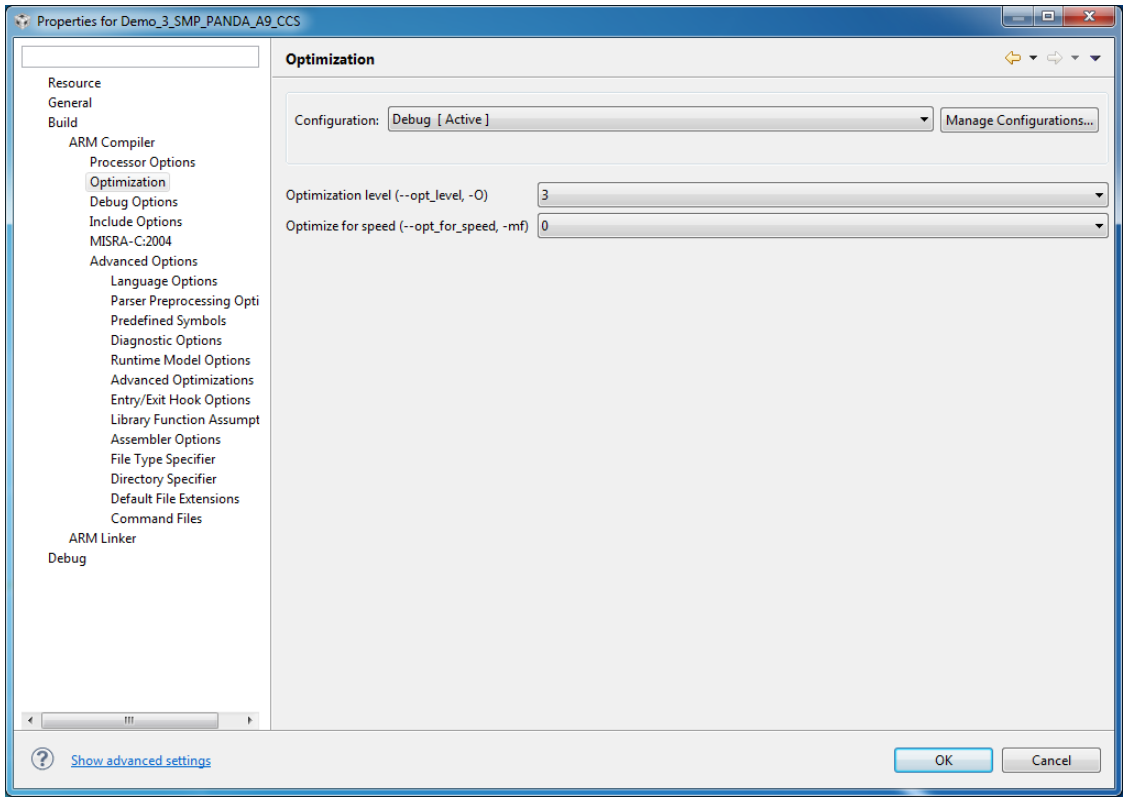

Figure 9-2 Optimization Settings

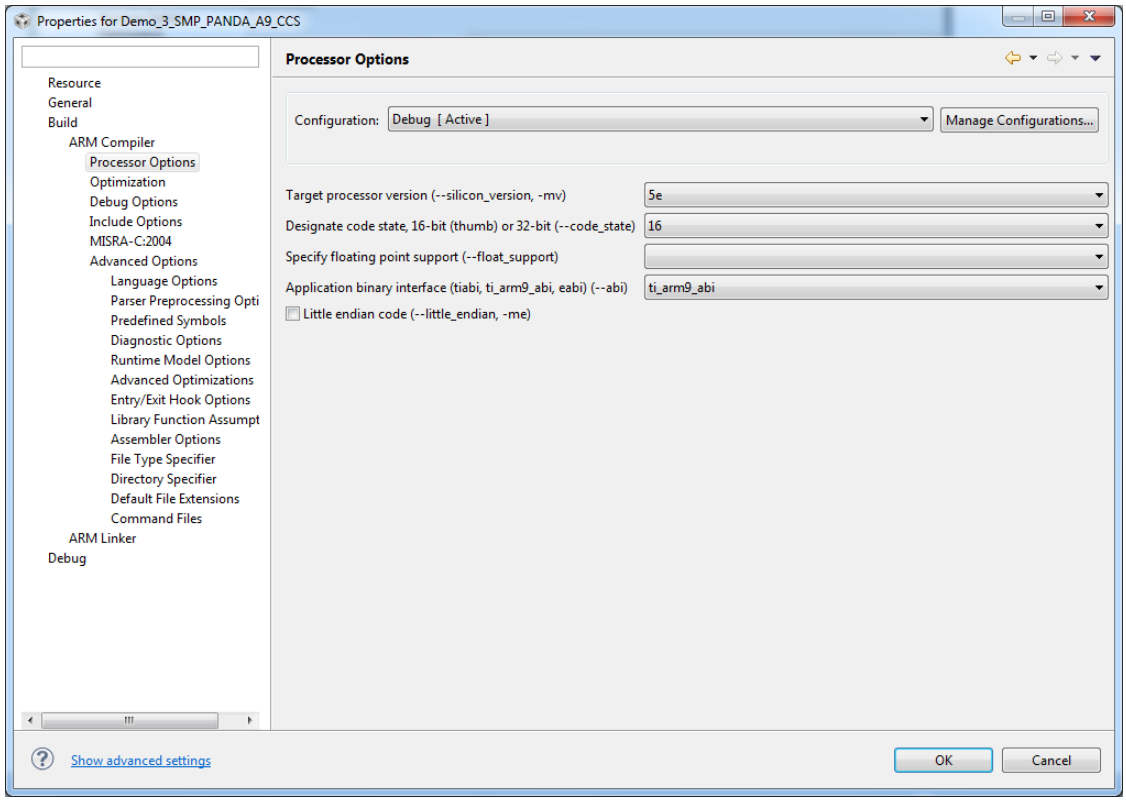

Figure 9-3 Processor Options Settings

Table 9-1 "C" Code Memory Usage

Description	Code Size
Minimal Build	< 1375 bytes
+ Runtime service creation / static memory	< 1650 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 2150 bytes
+ Timer & timeout + Timer call back + Round robin	< 2925 bytes
+ Events + Mailbox	< 3550 bytes
Full Feature Build (no names)	< 4175 bytes
Full Feature Build (no name / no runtime creation)	< 3775 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 (non-privilege / >1 core)	1588 bytes
Assembly code size (non-privilege / ==1 core)	1088 bytes
Assembly code size (privilege / >1 core)	1436 bytes
Assembly code size (privilege / ==1 core)	952 bytes
VFPv3	+128 bytes
VFPv3D16	+116 bytes
Saturation Bit Enabled	+36 bytes
`GICinit()`	136 bytes
`GICenable()`	104 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 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 A9, available at http://www.code-time.com