CODE TIME TECHNOLOGIES

mAbassi RTOS

Porting Document

SMP / ARM Cortex-A9 – DS5 (ARMCC)

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

This document is a complement to the user guide and it details the port of the SMP / BMP multi-core mAbassi RTOS to the ARM Cortex-A9 multi-core processor, commonly known as the Arm9 MPcore. The software suite used for this specific port is the ARM Development Studio (DS-5) from ARM Ltd; the versions used for the port and all tests are Version 5.14 and 5.22.

1.1 Distribution Contents

The main set of files supplied with this distribution are listed in Table 1-1 below (the ones that are not listed are support files needed for the different demos):

Table 1-1 Distribution (main files)

File Name	Description
`mAbassi.h`	RTOS include file
`mAbassi.c`	RTOS "C" source file
`ARMv7_SMP_L1_L2_ARMCC.s`	L1 and L2 caches, MMU, and SCU set-up module for the MPcore A9 / ARMCC
`cmsis_os.h`	Optional CMSIS V 3.0 RTOS API include file
`cmsis_os.c`	Optional CMSIS V 3.0 RTOS API source file
`mAbassi_SMP_CORTEXA9_ARMCC.s`	RTOS assembly file for the SMP ARM Cortex-A9 to use with the ARMCC tool chain
`Demo_3_SMP_AR5_CY5_A9_ARMCC.c`	Demo code that runs on the Arria V and the Cyclone V SoC FPGA development kit
`Demo_8_SMP_AR5_CY5_A9_ARMCC.c`	Demo code that runs on the Arria V and the Cyclone V SoC FPGA development kit
`Demo_9_SMP_AR5_CY5_A9_ARMCC.c`	Demo code that runs on the Arria V and the Cyclone V SoC FPGA development kit
`Demo_10-13_SMP_AR5_CY5_A9_ARMCC.c`	Demo code that runs on the Arria V and the Cyclone V SoC FPGA development kit
`Demo_110-113_SMP_AR5_CY5_A9_ARMCC.c`	Demo code that runs on the Arria V and the Cyclone V SoC FPGA development kit (CMSIS only)
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

The RTOS reserves the `SWI` (software interrupts) numbers 0 to 6 when mAbassi is configured to operate in user mode. A hook is made available for the application to use the `SWI`, as long as the numbers used are above 6. This keeps mAbassi compatible with the ARM semi-hosting protocol.

IMPORTANT: The linker data compression must be turned off using the command-line option `--datacompressor`. As specified in the ARM linker manual, data decompression (done in the standard library start-up code) should be performed before the enabling of the caches; but mAbassi, in order to minimize the boot time, sets and enables the caches before running the standard library start-up code. Due to this, data compression cannot be used. If the linker data compression is needed because the image ROM size is restricted, refer to Section 2.11 to set mAbassi to enable the cache after the library start-up code has executed.

Table 1-2 Disabling data compression

```
armlink  … --datacompressor off …
```

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 the code areas that disable interrupts, as SWIs are not required to access privileged registers or peripherals (when the processor processes a SWI, the interrupts are disabled). Selecting to run in privileged mode 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 support nesting of the interrupts (except FIQ nesting the IRQ). The ARM Cortex-A9 intrinsically supports exactly the same functionality delivered by mAbassi's hybrid stack. This is because the interrupts (IRQ) use a dedicated stack when in this processor mode.

The assembly file does not use the `BL` *addr* instruction when calling a function. This was chosen to allow the assembly file to access the whole 4 GB address space.

The VFPv3 or VPFv3D16, and NEON floating-point peripherals are supported by this port, and their registers are saved as part of the task context save and the interrupt context save.

2 Target Set-up

Very little is needed to configure the ARM Development Studio 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_ARMCC.s` in the application project, and make sure the configuration settings in the file `mAbassi_SMP_CORTEXA9_ARMCC.s` (described in the following sub-sections) match 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_ARMCC.s` takes care of all the start-up operations required for an application to operate on a multi-core processor.

2.1 Target device

Each manufacturer uses a different method to release from reset the cores other than core #0. As such, the start-up code must to be tailored for each target device. This information is specified in the assembly file with the value assigned to the token `OS_PLATFORM`. At the time of writing this document, the following platforms are supported:

Table 2-1 OS_PLATFORM valid settings

Target Platform	`OS_PLATFORM` value
Altera / Arria V Soc FPGA	`0xAAC5 (Same as Cyclone V)`
Altera / Cyclone V Soc FPGA	`0xAAC5 (Same as Arria V)`
Texas Instruments / OMAP 4460	`0x4460`
Xilinx / Zynq XC7Z020	`0x7020`
Freescale i.MX6	`0xFEE6`

If in the future there are platforms that are not listed in the above table, the numerical values assigned to the platform are specified in comments in the file `mAbassi_SMP_CORTEXA9_ARMCC.s`., right beside the internal definition of `OS_PLATFORM` (around line 75).

To select the target platform, all there is to do is to change the numerical value associated with the token `OS_PLATFORM` located around line 75 in the file `mAbassi_SMP_CORTEXA9_ARMCC.s`. By default, the target platform is the Altera Cyclone V / Arria V, therefore `OS_PLATFORM` is assigned the numerical value `0xAAC5`. The following table shows how to set the target platform to the Freescale i.MX6, which is assigned the numerical value `0xFEE6`:

Table 2-2 `OS_PLATFORM` modification

```
  IF :LNOT:(:DEF: OS_PLATFORM)
OS_PLATFORM          EQU     0xFEE6
  ENDIF
```

Alternatively, it is possible to overload the `OS_PLATFORM` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the target platform numerical value:

Table 2-3 Command line set of `OS_PLATFORM`

```
armasm  … --define “OS_PLATFORM SETA 0xFEE6”  …
```

2.2 Stacks and Heap Set-up

The Cortex-A9 processor uses 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 2-4 Stack Size Tokens

Description	Token Name
User / System mode (`main()` / A&E)	`OS_STACK_SIZE`
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 stack sizes are individuality controlled by the values set by the `OS_?????_STACK_SIZE` definitions, located between lines 45 and 75 in the file `mAbassi_SMP_CORTEXA9_ARMCC.s`. To not reserve a stack for a processor mode, all there is to do is to set the definition of `OS_?????_STACK_SIZE` to a value of zero (0). To specify the stack size, the definition of `OS_?????_STACK_SIZE` is set to the desired size in bytes (see Section 3 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 is the value assigned to the token `OS_?????_STACK_SIZE`. This equal distribution of stack size may not be optimal; if a non-equal distribution is required, contact Code Time Technologies for additional information on the code modifications involved.

Alternatively, it is possible to use definitions of the stack extracted from the linker command file. When the value assigned to a stack definition token `OS_?????_STACK_SIZE` is set to -1, the stack uses the size and the data section reserved in the linker command file; then no memory is reserved for this stack by the file `Abassi_SMP_CORTEXA9_ARMCC.s`. Contrary to setting the definition token to a positive value, the stack size defined in the linker is the total stack size shared by all cores (except for `ARM_LIB_STACK`). This means for 2 cores, each core will be allocated half the stack size defined in the linker command file; for 4 cores, each core will be allocated a quarter of the stack size defined in the linker command file. The names of the base of the stack (which is at the highest memory address of data section) and the names of their sizes are listed in Table 2-5.

Table 2-5 Linker Command file Stack Tokens

Description	Stack Name
User / System mode	`ARM_LIB_STACK`
Supervisor mode	`SUPER_STACKS`
Abort mode	`ABORT_STACKS`
Undefined mode	`UNDEF_STACKS`
Interrupt mode	`IRQ_STACKS`
Fast Interrupt mode	`FIQ_STACKS`
Heap	`ARM_LIB_HEAP`

As supplied in the distribution, all stack size tokens are assigned the value of -1 meaning all stack information is supplied by the linker command file.

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 2-6 `OS_IRQ_STACK_SIZE` modification

```
  IF :LNOT:(:DEF: 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_ARMCC.s` by using the assembler command line option `--define` and specifying the desired stack size as shown in the following example, where the IRQ stack size is set to 512 bytes:

Table 2-7 Command line set of `OS_IRQ_STACK_SIZE`

```
armasm  … --define "OS_IRQ_STACK_SIZE SETA 512"  …
```

A third way to allocate the different stacks and specify their sizes is by setting to a value of -2 the token `OS_?????_STACK_SIZE` (does not apply to `OS_STACK_SIZE`). When this is done, the stacks memory and their sizes are supplied through external arrays and variables. The stack arrays must be dimensioned to: *#Core* * *stacksize* bytes; the run-time set-up of the stacks makes sure the stacks are aligned according to the Cortex-A9 requirements. The variable specifying the stack sizes indicates the number of bytes per core and not the number of bytes in the stack array.

Table 2-8 Variables Stack Names

Description	Stack Name	Stack Size Name
User / System mode	`n/a`	`n/a`
Supervisor mode	`G_SUPER_stack`	`G_SUPER_stackSize`
Abort mode	`G_ABORT_stack`	`G_ABORT_stackSize`
Undefined mode	`G_UNDEF_stack`	`G_UNDEF_stackSize`
Interrupt mode	`G_IRQ_stack`	`G_IRQ_stackSize`
Fast Interrupt mode	`G_FIQ_stack`	`G_FIQ_stackSize`

2.3 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 obviously. This must be done for both the `mAbassi.c` file and the `mAbassi_SMP_CORTEXA9_ARMCC.s` file, through the setting of 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_ARMCC.s`, to modify the number of cores, all there is to do is to change the numerical value associated to the token definition of `OS_N_CORE`, located around line 35; this is shown in the following table. By default, the number of cores is set to 2.

Table 2-9 OS_N_CORE modification

```
  IF :LNOT:(:DEF: 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_ARMCC.s by using the assembler command line option --define and specifying the required number of cores as shown in the following example, where the number of cores is set to 4:

Table 2-10 Command line set of OS_N_CORE (ASM)

```
armasm  … --define "OS_N_CORE SETA 4"  …
```

Exactly the same value of OS_N_CORE as specified for the assembler must be specified for the compiler; a mismatch between the assembly and "C" definition either generates a link error if the assembly file value is less than the "C" value or will freeze the application if the assembly file value is larger than the "C" value. In the following example, the number of cores is set to 4 for the "C" files:

Table 2-11 Command line set of OS_N_CORE (C)

```
armcc  … -D OS_N_CORE=4  …
```

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 single core application always executes on core #0.

2.4 Privileged mode

It is possible to configure mAbassi for the Cortex-A9 MPCore to make the application execute in either 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 USER mode, Software interrupts (SWI) are needed to read or write the registers and peripherals are only accessible in privileged mode. Having to use SWI disables the interrupts during the time a SWI is processed. 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 Arm9 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_ARMCC.s (ARM_SMP_L1_L2_ARMCC.s, release version 1.69.69 and upward, also needs this information) requires the information if the application executes in the privileged mode or not. By default, the distribution file sets the processor to operate in privileged mode. 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 40. 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:

Table 2-12 `OS_RUN_PRIVILEGE` modification

```
  IF :LNOT:(:DEF: OS_RUN_PRIVILEGE)
OS_RUN_PRIVILEGE  EQU     0
  ENDIF
```

Alternatively, it is possible to overload the `OS_RUN_PRIVILEGE` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the desired mode of execution as shown in the following example, where the selected mode is non-privilege:

Table 2-13 Command line set of `OS_RUN_PRIVILEGE`

```
armasm  … --define "OS_RUN_PRIVILEGE SETA 0"  …
```

2.5 L1 & L2 Cache Set-up

The build option `OS_USE_CACHE` controls if the MPcore L1 and L2 caches, the memory management unit (MMU) and the snoop control unit (SCU) are configured and enabled. Setting the token `OS_USE_CACHE` to a non-zero value configures and enables the caches, MMU and SCU; setting it to a value of zero (0) disables the caches, MMU and SCU. The `OS_USE_CACHE` token is defined around line 100 in the file `mAbassi_SMP_CORTEXA9_ARMCC.s` and is set to enable (non-zero) by default. This is shown in the following table:

Table 2-14 `OS_USE_CACHE` set-up

```
  IF :LNOT:(:DEF: OS_USE_CACHE)
OS_USE_CACHE  EQU     0
  ENDIF
```

Alternatively, it is possible to overload the `OS_USE_CACHE` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the desired mode of execution as shown in the following example, where the selected mode is non-privileged:

Table 2-15 Command line set of `OS_USE_CACHE`

```
armasm  … --define "OS_USE_CACHE SETA 0"  …
```

NOTE: When the caches are used, there is a dependency on the external function `COREcacheON()`. This function is located in the file `ARMv7_SMP_L1_L2_ARMCC.s` file, which is optional.

2.6 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 the task 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 for a single task, then it must be kept local to each task. To keep the value 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 45 in the file `mAbassi_SMP_CORTEXA9_ARMCC.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 2-16 Saturation Bit configuration

```
  IF :LNOT:(:DEF: OS_HANDLE_PSR_Q)
OS_HANDLE_PSR_Q   EQU     1
  ENDIF
```

Alternatively, it is possible to overload the `OS_HANDLE_PSR_Q` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the desired setting (here is to keep localized) with the following:

Table 2-17 Command line set of Saturation Bit configuration

```
armasm  … --define OS_HANDLE_PSR_Q=1 …
```

2.7 Spinlock implementation

All multi-core SMP RTOSes require the use of spinlocks to provide a short time exclusive access to shared resources. mAbassi internally uses two functions to lock and unlock the spinlocks, `CORElock()` and `COREunlock()` (refer to mAbassi User's Guide [R1]), and these functions are implemented in the `mAbassi_SMP_CORTEXA9_ARMCC.s` file. The spinlocks can be implemented using 3 different techniques:

- Pure software spinlock
- Using the `LDREX` and `STREX` instructions
- Using a hardware spinlock register

The difference between the 3 types of spinlocks can be resumed as follows:

- A pure software spinlock disables the interrupts for a short time but does not depend on any hardware resources nor peripherals
- The `LDREX`/`STREX` based spinlock does not disable the interrupts but the ARM Snoop Control Unit (SCU) must be enabled, which implies the D-Cache must be configured and enabled which in turn requires the Memory Management Unit (MMU) to be configured and enabled
- The hardware register spinlock does not disable the interrupts, uses less code than the pure software spinlock, but it uses more code than the `LDREX`/`STREX` based spinlock. This type of spinlock can only be used if the device has custom spinlock registers

The type of spinlock to use is specified with the token the token `OS_SPINLOCK`. It must be set to zero (0) for a pure software spinlock, one (1) for the `LDREX`/`STREX` based spinlock, or, for a hardware register base spinlock, to the same value as the token `OS_PLATFORM` (Section 2.1) when the target device supports hardware spinlocks.

NOTE: If the token `OS_SPINLOCK` is set to one (1) for the `LDREX` / `STREX` based spinlock and the cache is not enabled (Section 2.5), an assembly time error message is issued.

The distribution code uses the pure software spinlock, meaning the `OS_PLATFORM` token is set to zero, as shown in the following table. This is located around line 80 in the file `mAbassi_SMP_CORTEXA9_ARMCC.s`:

Table 2-18 `OS_SPINLOCK` specification

```
  IF :LNOT:(:DEF: OS_SPINLOCK)
OS_SPINLOCK   EQU     1
  ENDIF
```

It is possible to overload the `OS_SPINLOCK` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the type of spinlock as shown in the following example, where the spinlock is set to use the `LDREX`/`STREX` pair of instructions:

Table 2-19 Command line set of `OS_SPINLOCK`

```
armasm  … --define "OS_SPINLOCK SETA 1" …
```

Some target platforms have sets of hardware spinlock registers, e.g. Texas Instruments OMAP 4460. When the selected spinlock implementation is based on the hardware spinlock registers, mAbassi reserves a total of 4 spinlock registers. By default, the spinlock register indexes #0 to #5 are used by mAbassi and as such must not be used by anything else. If mAbassi needs to co-exist with other applications that are already using one or more registers in this set of 6, it is possible to make mAbassi use a different set of indexes. The base index of the set of 6 registers is specified with the value assigned to the token `OS_SPINLOCK_BASE`. The assembly file and the "C" file use same token and they must be set to the same value. As an example to make mAbassi use the hardware registers #22 to #27, all there is to do is to change the numerical value associated to the `OS_SPINLOCK_BASE` token, located around line 85; this is shown in the following table:

Table 2-20 `OS_SPINLOCK_BASE` modification

```
  IF :LNOT:(:DEF: OS_SPINLOCK_BASE)
OS_SPINLOCK_BASE  EQU     22
  ENDIF
```

It is also possible to overload the `OS_SPINLOCK_BASE` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the required base register index as shown in the following example:

Table 2-21 Command line set of `OS_SPINLOCK_BASE` (ASM)

```
armasm  … --define "OS_SPINLOCK_BASE SETA 22" …
```

The same numerical value must also be provided to the "C" file. This is show in the following table:

Table 2-22 Command line set of `OS_SPINLOCK_BASE` (C)

```
armcc  … -D OS_SPINLOCK_BASE=22 …
```

There is a possible race condition when using spinlocks on a target processor with 3 or more cores. This is the corner case when the tasks running on 3 cores or more are all trying non-stop to lock and unlock the same spinlock. When this condition arises, it is possible that the same 2 cores always get the spinlock, starving the other one(s). This is not an issue with mAbassi but it is due to the fact the memory accesses (S/W spinlock or H/W spinlock) and the cache give access based on the core numbering and not in a round robin or random fashion. To randomize the core given the spinlock, a random delay can be added when the number of cores (`OS_N_CORES`) is greater than 2. The delay is added when the build option `OS_SPINLOCK_DELAY` is set to a non-zero value and `OS_N_CORE` is greater than 2; by default, the token `OS_SPINLOCK_DELAY` is set to a non-zero value. As for other tokens, the numerical value associated to the `OS_SPINLOCK_DELAY` token, located around line 90, can be changed as shown in the following table:

Table 2-23 `OS_SPINLOCK_DELAY` modification

```
  IF :LNOT:(:DEF: OS_SPINLOCK_DELAY)
OS_SPINLOCK_DELAY EQU     0
  ENDIF
```

It is also possible to overload the `OS_SPINLOCK_DELAY` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the required base register index as shown in the following example:

Table 2-24 Command line set of `OS_SPINLOCK_DELAY` (ASM)

```
armasm  … --define “OS_SPINLOCK_DELAY SETA 0” …
```

2.8 Spurious Interrupt

The Cortex-A9 interrupt controller (GIC) generates an interrupt when a spurious interrupt is detected. The interrupt number for a spurious interrupt is 1023 and it cannot be disabled. When mAbassi is configured to handle less than 1024 interrupt sources (set by the build option `OS_N_INTERRUPTS`), then if a spurious interrupt occurs, the function pointer of the handler read from the interrupt table is invalid, as the interrupt table is not large enough to hold an entry for it. The build option `OS_SPURIOUS_INT` (new in version 1.66.65) can be set to inform the interrupt dispatcher to ignore interrupt #1023. The special handling of the spurious interrupt is enabled when the build option `OS_SPURIOUS_INT` is set to a non-zero value; by default, the token `OS_SPURIOUS_INT` is set to a non-zero value, enabling the special handling. As for other tokens, the numerical value associated to the `OS_SPURIOUS_INT` token, located around line 105, can be changed as shown in the following table:

Table 2-25 `OS_SPURIOUS_INT` modification

```
  IF :LNOT:(:DEF: OS_SPURIOUS_INT)
OS_SPURIOUS_INT   EQU     0
  ENDIF
```

It is also possible to overload the `OS_SPURIOUS_INT` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--defsym` and specifying the required base register index as shown in the following example:

Table 2-26 Command line set of `OS_SPURIOUS_INT` (ASM)

```
armasm  … --define OS_SPURIOUS_INT=0 …
```

Unless the build option `OS_N_INTERRUPTS` is set to 1024, the build option `OS_SPURIOUS_INT` should never be set to a value of zero. The real-time and code size impact of including the spurious interrupt special handling is very minimal: it adds 1 instruction when using 32-bit ARM instructions and 2 instructions with Thumb2.

2.9 Thumb2

The assembly support file (`mAbassi_SMP_CORTEXA9_ARMCC.s`) is by default using 32-bit ARM instructions. The build option `OS_ASM_THUMB` (new in version 1.66.66) can be set to use Thumb2 instructions instead. The use of Thumb2 instructions is enabled when the build option `OS_ASM_THUMB` is set to a non-zero value; by default, the token `OS_ASM_THUMB` is set to a non-zero value, enabling the special handling. As for other tokens, the numerical value associated to the `OS_ASM_THUMB` token, located around line 110, can be changed as shown in the following table:

Table 2-27 `OS_ASM_THUMB` modification

```
  IF :LNOT:(:DEF: OS_ ASM_THUMB)
OS_ ASM_THUMB    EQU     1
  ENDIF
```

It is also possible to overload the `OS_ASM_THUMB` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--defsym` and specifying the required base register index as shown in the following example:

Table 2-28 Command line set of `OS_ASM_THUMB` (ASM)

```
armasm  … --define OS_ASM_THUMB=1 …
```

NOTE: Never use the `--thumb` command line option with the `mAbassi_SMP_CORTEXA9_ARMCC.s` file.

2.10 VFP / NEON set-up

When a processor is coupled with a floating-point peripheral and this VFP is used by the application code, then mAbassi must be informed it must preserve the VFP register during the task context switch and the interrupt context save. mAbassi supports these three classes of VFP:

- No VFP coprocessor
- VFPv2 / VFPv3D16 / Neon with 16 registers
- VFPv3 / Neon with 32 registers

The type of FPU to support is obtained from the assembler command line option `–fpu` (and also from the command line option `--cpu`: e.g. `--cpu=Cortex-A9.no_neon.no_vfp` versus `--cpu=Cortex-A9`), and mAbassi determines from the value specified how many and the size of the floating point registers to save / restore during a context switch.

Table 2-29 Command line selection of the VFP

```
armasm  … --fpu=vfpv3 …
```

2.11 Linker data compression

By default, mAbassi enables the caches as soon as possible in order to minimize the boot-up time. The cache enabling is done before the library start-up code is executed, and it creates a conflict with the linker data decompression, as stated in the ARM linker documentation. When an application uses mAbassi in its normal configuration, the linker data compression must be disabled with the use of the `--datacompressor` command-line option:

Table 2-30 Disabling data compression

```
armlink  … --datacompressor off …
```

If it is necessary to use the linker data compression, then mAbassi can be set-up to configure and enable the caches right before entering `main()`. This is achieved through the use of the `$Super$$main` and `$Sub$$main` patterns to patch the function `main()`. To make mAbassi configure and enable the caches right before entering `main()`, do not set the linker `--datacompressor` command-line option to `off` (it still can be set to 0, 1, or 2) and set the token `OS_DATACOMPRESSOR` to a non-zero value in `mAbassi_SMP_CORTEXA9_ARMCC.s`. The token `OS_DATACOMPRESSOR` is located around line 100:

Table 2-31 `OS_DATACOMPRESSOR` modification

```
  IF :LNOT:(:DEF: OS_DATACOMPRESSOR)    ; If the linker data compressor is used
OS_DATACOMPRESSOR EQU     1             ; ==0: linker compressor not used
  ENDIF                                 ; !=0: linker compressor in use
```

It is also possible to overload the `OS_DATACOMPRESSOR` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the required base register index as shown in the following example:

Table 2-32 Command line set of `OS_DATACOMPRESSOR` (ASM)

```
armasm  … --define "OS_DATACOMPRESSOR SETA 1" …
```

When the linker data compressor is used, it is also necessary to enable interworking (mixed Thumb and 32-bit instruction calls/branches) when assembling `mAbassi_SMP_CORTEXA9_ARMCC.s`; this is done through the `apcs` command line option (or the check box in the GUI):

Table 2-33 Command line set of interworking (ASM)

```
armasm  … --apcs=/interwork …
```

2.12 Multithreading

By default, the ARMCC "C" runtime library is not multithread safe. There are two aspects to take into account when protecting the library for multithread. The first one involves reentrance; a few library functions are not reentrant, therefore two tasks accessing the same non-reentrant function at the same time can create major issues. A good example of non-reentrant functions are the dynamic memory allocation functions, `malloc()` and `free()`. As they internally use a static buffer, a few pointers, and some linked lists, if two tasks access the internals of the dynamic memory allocation at the same time, corruption could occur. Protecting the non-reentrant functions is straightforward: all there is to do is to make sure there is only a single task that can access the non-reentrant functions at any time. This is done with a mutex, as it is the perfect mechanism to guarantee exclusive access to a resource.

The second type of function and variables that are not multithread safe are so due to internal data used by the library, data that is truly a global resource. Such examples of these are: the `errno` variable or the `locale` information. The only efficient way to protected these functions and variables against multithreading is to have the library setup to use unique variables for each task. There are multiple ways to implement the data swapping, but fundamentally, if the library does not provided such a swapping mechanism, it becomes cumbersome to solve the issue, as it would require a manually swapping the contents, copying each individual internal static variables of the library at every task switch.

ARMCC's standard library fully supports all mechanisms to make the library multithread safe. The MicroLib does not have such mechanisms. The following sub-sections describe how to make each of the two libraries multithread safe.

2.12.1 Standard Library Multithreading Protection

The ARMCC standard library (not the MicroLib, see Section 2.12.3 for the MicroLib) can be set to be completely protected against reentrance and also be multithread-safe. The type of multithreading protection is selected according to the definition of the build option `OS_KEIL_REENT`; this is not a standard build option as it only is used with the DS5 development suite on ARM processors. If this build option is not defined, or if it is defined with a value of zero, the library is neither protected against reentrance nor against multithreading. If the build option is positive, the library is fully multithread-safe and protected against reentrance for every tasks in the application. If the build option value is negative, only user-selected tasks that are configured access the library in a multithread-safe fashion; the library still remains protected against reentrance for all tasks.

NOTE: When the MicroLib is selected, the option `OS_KEIL_REENT` is ignored.

2.12.1.1 Full Protection

For full multithreading protection of the standard library, all there is to do is to define the build option `OS_KEIL_REENT` with a positive value. The build option `OS_KEIL_REENT` for the multithreading protection must be given to the compiler and the assembler. By default, the token `OS_KEIL_REENT` is set to a zero value in the file `mAbassi_SMP_CORTEXA9_ARMCC.s`. As for other tokens, the numerical value associated to the `OS_KEIL_REENT` token, located around line 95, can be changed as shown in the following table:

Table 2-34 `OS_KEIL_REENT` modification

```
  IF :LNOT:(:DEF: OS_KEIL_REENT)
OS_KEIL_REENT EQU     1
  ENDIF
```

It is also possible to overload the `OS_KEIL_REENT` value set in `mAbassi_SMP_CORTEXA9_ARMCC.s` by using the assembler command line option `--define` and specifying the required base register index as shown in the following example:

Table 2-35 Command line set of `OS_KEIL_REENT` (ASM)

```
armasm  … --define “OS_KEIL_REENT SETA 1” …
```

The same numerical value must also be provided to the “C” file. This is show in the following table:

Table 2-36 Command line set of `OS_KEIL_REENT` (C)

```
armcc  … -D OS_KEIL_REENT=1 …
```

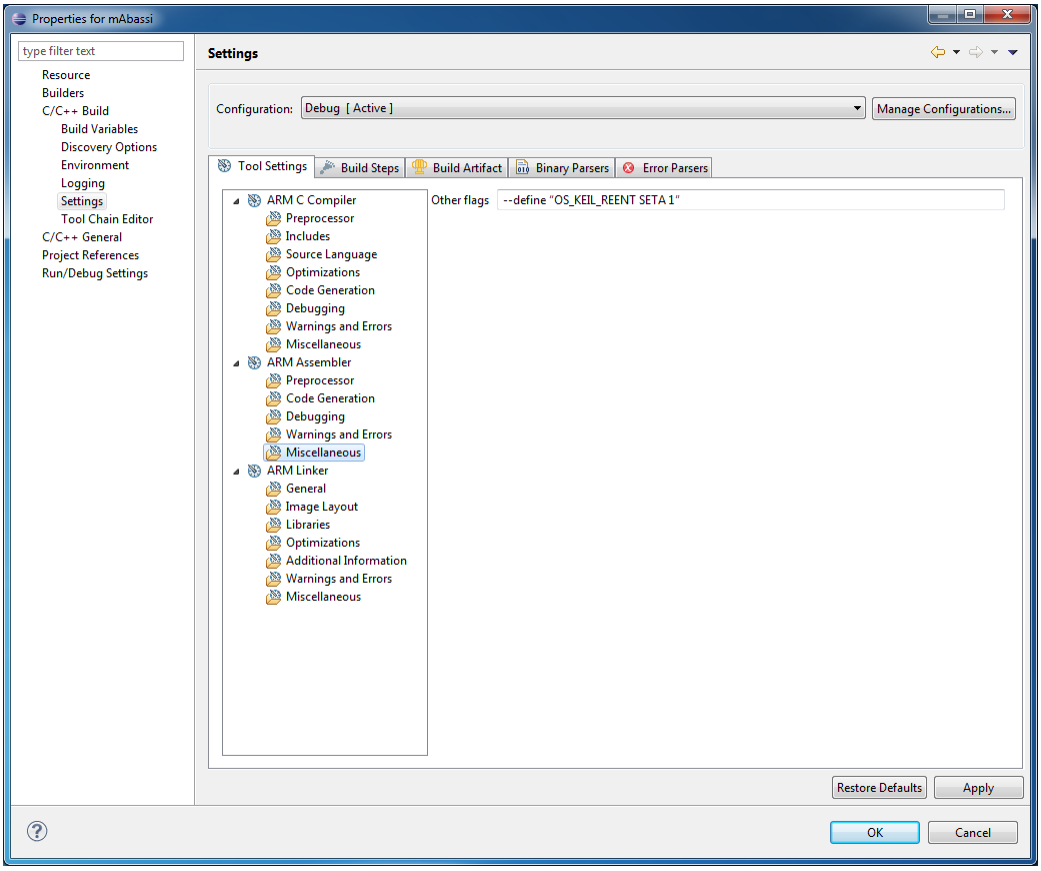

Figure 2-1 GUI set of `OS_KEIL_REENT` (ASM)

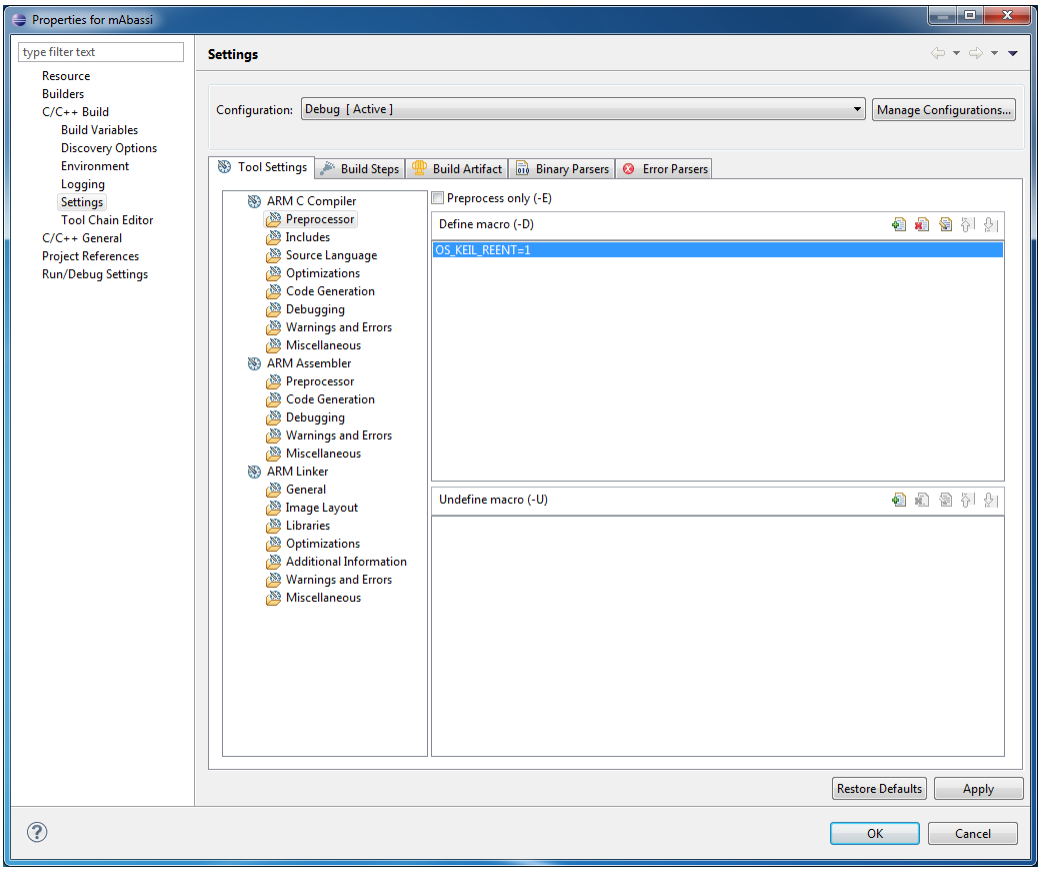

Figure 2-2 GUI set of `OS_KEIL_REENT` (C)

Exactly 96 bytes per task are needed to support full re-entrance. This extra memory is not an integral part of the task descriptor; instead a pointer in the task descriptor holds the location of this per task extra memory. The 96 bytes are allocated using the component `OSalloc()`, which means either enough memory must be reserved with `OS_ALLOC_SIZE` or, if `OS_ALLOC_SIZE` is set to zero, then enough heap area must be allocated for `malloc()`.

2.12.1.2 Partial Protection

The use of full multithread protection for the library requires 96 bytes of extra data memory for each task in the application. The extra memory required is not due to Abassi, but it is the amount of memory the library requires to hold all its internal static data. It may not be desirable to use multithread protection for all tasks, or on data memory restricted applications it may be impossible to use full multithreading protection. Setting the build option `OS_KEIL_REENT` to a negative value allows the designer to select the tasks where multithreading protection is used. The library modules that are non-reentrant are still protected by a mutex, only the static area of the library becomes under control. The build option `OS_KEIL_REENT` is set the same way as described in Section 2.12.1.1, only it must be set to a negative value for the partial protection.

Partial multithreading means that only the tasks that are set up to use the library in a multithread safe manner will require the 96 bytes block of extra data memory. Not only is memory needed for the library internal data, but if file I/O is used in the task, more memory is also needed for the buffering of the file or stream. It is good practice to use the standard library function `setbuf()`, or `setvbuf()` to tailor each stream buffer size.

If a task uses none of the library multithread unsafe static data, then the task does not need to access the library internal data in an exclusive manner, so there is no need to reserve and assign the memory block of 96 bytes of data memory. If a task uses one or more of the library multithread unsafe static data, but it is the only task using that data, there is still no need to make the library multithread safe for that task. Only when two or more tasks use the same internal data of the library do these tasks need to access the library in a multithread safe manner.

For more information on which library functions and/or variables are non-reentrant and/or multithread unsafe, refer to Section 2.12.3.

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

Table 2-37 Setting a task to use re-entrant library

```
#include "Abassi.h"

TSK_t *TskReent
int ReentData[96/sizeof(int);

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

memset(&ReentData[0], 0, sizeof(ReentData));    /* Buffer must be set to zero        */

TskReent->XtraData[0] = (intptr_t)&ReentData;   /* Attach the libspace to the task   */

TSKresume(TskReent);                            /* The task may now be resumed       */
```

The declaration "`int Reent[96/sizeof(int)];`" can be replaced by a dynamic memory allocation of `(size_t)96.` If a task does not require access to the library in a multithread safe way the above code is not required.

2.12.2 Thread-unsafe functions / variables

The list of thread-unsafe library functions that can be made thread-safe with mAbassi multi-threading protection is obtained by performing a search in the help using the keyword `Thread-safe`, looking into the C user guide section titled "*Thread-safe C library functions*".

For the multithread unsafe functions and/or variables, there is no simple way to make these functions or variables multi-thread safe. The ARMCC library supports a non-standard implementation of these functions/variables. The list can be obtained by performing a search in the help using the keyword `Thread-safe`, looking into the C user guide section titled "*C library functions that are not thread-safe*".

2.12.3 MicroLIB Multithreading Protection

Contrary to the standard library, the MicroLIB does not offer internal support for multi-threading protection. Some functions in the ARMCC C MicroLIB runtime library are not reentrant. If these functions are only used in one task, then there will be no problems. But if they are used by more than one task, they need to be protected by an Abassi mutex. The preferred way is to re-use the `G_OSmutex` for all non-multithread-safe functions, as this will avoid deadlocks. Therefore, non-reentrant functions must be manually protected with a mutex.

For the multithread unsafe function and/or variables, there is no simple way to make these functions or variables multi-thread safe. The ARMCC library supports a non-standard implementation of these functions/variables. The list can be obtained by performing a search in the help using the keyword `Thread-safe`, looking into the C user guide section titled "*C library functions that are not thread-safe*".

2.13 Performance Monitoring

New in version 1.69.69, is the performance monitoring add-on. This facility relies on a fine resolution timer / counter and this timer / counter is set-up and handled in the file `mAbassi_SMP_CORTEXA9_ARMCC.s`. There are 3 build options related to the performance monitoring timer / counter:

- `OS_PERF_TIMER_BASE`
- `OS_PERF_TIMER_DIV`
- `OS_PERF_TIMER_ISR`

All Cortex-A9 MPCores have a global timer / counter and it is a natural candidate to be used by the performance monitoring add-on. To use the global timer / counter, set the build option `OS_PERF_TIMER_BASE` to a value of 0. If the pre-scaling value of the global timer / counter is set in the application, then set the build option `OS_PERF_TIMER_DIV` to a value of 0. If the prescaler is not set by the application, then set build option `OS_PERF_TIMER_DIV` to the desired prescaler value. For the global timer / counter, the build option `OS_PERF_TIMER_ISR` is ignored as it does not requires interrupts as it is a 64-bit timer (64 bits is the size of the counters used by the performance monitoring add-on).

For other counter / timer options, please look directly into the file `mAbassi_SMP_CORTEXA9_ARMCC.s`, searching for the `OS_PERF_TIMER_BASE` token. There is a very detailed table explaining all the offerings and specifying the values to set the 3 related build options to for each the timer / counter supported.

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 1020 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.

2.14 Interrupt Handling

2.14.1 Interrupt Table Size

Most devices do not require all 256 interrupts, as they may only handle between 64 and 128 sources of interrupts; or some very large device may 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 mAbassi 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 0-1 Command line set the interrupt table size

```
armcc … -D OS_N_INTERRUPTS=49 …
```

2.14.2 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 0-2 shows the code required to attach the private timer interrupt on an ARM MPcore processor (Interrupt ID #29) to the RTOS timer tick handler (`TIMtick`):

Table 0-2 Attaching a Function to an Interrupt

```
#include “mAbassi.h”

   …
   OSstart();
   …
   GICenable(29, 128, 1);                  /* Timer set mid prioirty edge triggered  */
   OSisrInstall(29, &TIMtick);

   /* Set-up the count reload and enable private timer interrupt and start the timer */

   … /* More ISR setup */

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

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

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 0-3:

Table 0-3 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, as reported 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. The removal of the interrupt request must be performed within the interrupt handler function.

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 flow. 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 private 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.

2.15 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 6.2).

2.16 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. Also, supporting nesting on this processor architecture becomes real-time inefficient as the processor interrupt context save is not stack based, but register bank based.

3 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 3-1 Context Save Stack Requirements

Description	Context save
Blocked/Preempted task context save	48 bytes
Blocked/Preempted task context save / VFP enable (`OS_FPU_TYPE != 0`)	+112 bytes
Interrupt dispatcher context save (User Stack)	64 bytes
Interrupt dispatcher context save (User Stack) / 16 - VFP (`OS_FPU_TYPE == 16`)	+136 bytes
Interrupt dispatcher context save (User Stack) / 32 - VFP (`OS_FPU_TYPE == 32`)	+264 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 bytes 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.

4 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.

5 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.

6 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.

6.1 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 "`subs pc, lr, #8`".

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 6.2)
`PFabort_Handler()` (Section 6.3)
`SWI_Handler()` (Section 6.4)
`Undef_Handler()` (Section 6.5)

6.2 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 "`subs pc, lr, #4`".

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 6.1)
`PFabort_Handler()` (Section 6.3)
`SWI_Handler()` (Section 6.4)
`Undef_Handler()` (Section 6.5)

6.3 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 "`subs pc, lr, #4`".

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 6.1)
`FIQ_Handler()` (Section 6.2)
`SWI_Handler()` (Section 6.4)
`Undef_Handler()` (Section 6.5)

6.4 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.

See also

`DATAabort_Handler()` (Section 6.1)
`FIQ_Handler()` (Section 6.2)
`FPabort_Handler()` (Section 6.3)
`Undef_Handler()` (Section 6.5)

6.5 Undef_Handler

Synopsis

```
#include "mAbassi.h"

void Undef_Handler(void);
```

Description

`Undef_Handler()` is the exception handler for a undefined instruction fault. In the distribution code, this is implemented as an infinite loop. If the application needs to perform special processing when an undefined instruction fault occurs, all there is to do is to include a function with the above function prototype, and it will overload the supplied undefined instruction 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 6.1)
`FIQ_Handler()` (Section 6.2)
`PFabort_Handler()` (Section 6.3)
`SWI_Handler()` (Section 6.4)

7 Chip Support

No custom chip support is provided with the distribution because most device manufacturers provide a BSP, i.e. code to configure the peripherals on their devices. The distribution code contains some of the manufacturer's open source libraries, e.g. Altera.

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.

7.1 GICenable

Synopsis

```
#include "mAbassi.h"

void GICenable(int IntNmb, int Prio, int Edge);
```

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 always applied to the core on which `GICenable()` is executing.

Availability

Always.

Arguments

`IntNmb`	Interrupt number to enable
`Prio`	Priority of the interrupt `0` : highest priority `255` : lowest priority
`Edge`	Edge or level detection `== 0` : level detection `!= 0` : edge detection

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_ARMCC.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. As the supplied assembly function is declared weak, it will not be included during the link process. The equivalent "C" code of the distribution implementation is supplied in comments in the assembly file.

See also

`GICinit()` (Section 7.2)

7.2 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_ARMCC.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. As the supplied assembly function is declared weak, it will not be included during the link process. The equivalent "C" code of the distribution implementation is supplied in comments in the assembly file.

See also

`GICenable()` (Section 7.1)

8 Measurements

This section provides an overview of the memory requirements encountered when the RTOS is used on the Arm9 and compiled with the DS5 tool chain. Latency measurements are provided, but one should remember CPU latency latencies are highly dependent on 3 factors. It first depends on how many cores are used; it also depends on the type of load balancing, i.e. 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. The specific configuration and run-time conditions are described in the latency subsection (Section 8.2)

8.1 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 release version 1.64.63 of the RTOS and may change in other versions. One should interpret these numbers as the “very likely” numbers for other released versions 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] (option `–g` not specified)
2. Optimization level: Maximum / Size (`-O3 -Ospace`)
3. Target `--cpu=Cortex-A9`
4. FPU `--fpu=VFPv3`

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

Table 8-1 "C" Code Memory Usage

Description	Thumb Size	32-Bit Size
Minimal Build	< 1425 bytes	< 2175 bytes
+ Runtime service creation / static memory	< 1675 bytes	< 2475 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 1675 bytes	< 2475 bytes
+ Timer & timeout + Timer call back + Round robin	< 2125 bytes	< 3225 bytes
+ Events + Mailbox	< 2675 bytes	< 4125 bytes
Full Feature Build (no names)	< 3275 bytes	< 5125 bytes
Full Feature Build (no name / no runtime creation)	< 3950 bytes	< 6275 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 3625 bytes	< 5775 bytes
OS_KEIL_REENT > 0	~ +200 bytes	~ +300 bytes
OS_KEIL_REENT < 0	~ +125 bytes	~ +175 bytes
True SMP (OS_MP_TYPE == 2)	+0 bytes	+0 bytes
Packed SMP (OS_MP_TYPE == 3)	~ +25 bytes	~ +25 bytes
True BMP (OS_MP_TYPE == 4)	~ +200 bytes	~ +350 bytes
Packed BMP (OS_MP_TYPE == 5)	~ +200 bytes	~ +350 bytes

The selection of load balancing type affects the "C" code size. The added memory requirements are indicated as approximate because depending on the build option combination, the kernel code is different. As such, the optimizer does not deliver the same code size.

Table 8-2 Assembly Code Memory Usage

Description	Size
Assembly code size (non-privilege / >1 core)	2032 bytes
Assembly code size (non-privilege / ==1 core)	1168 bytes
Assembly code size (privilege / >1 core)	1712 bytes
Assembly code size (privilege / ==1 core)	928 bytes
VFPv3	+120 bytes
VFPv3D16	+112 bytes
Saturation Bit Enabled	+36 bytes
Altera Cyclone V Support	+256 bytes
Freescale i.MX6 Support	+48 bytes
TI OMAP 4460 Support	+16 bytes
Xilinx Zynq Support	+84 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.

8.2 Latency

Latency of operations has been measured on an Altera Cyclone V Evaluation board populated with a 800MHz dual-core Cortex-A9. All measurements have been performed on the real platform. This means the interrupt latency measurements had to be instrumented to read the `SysTick` counter value. This instrumentation can add up to 5 or 6 cycles to the measurements. The code optimization setting that was used for the latency measurements is `-O3 -Otime`, which optimizes the code generated for the best speed. The debugging option was turned off as the debugging sometimes restricts the optimizer. All operations are performed on core #0; the FPU was enabled (VFPv3 / Neon) and the type of multi-processing was set to true SMP (`OS_MP_TYPE` set to 2). The cache is enabled and the spinlock type is the one using LDREX/STREX. All measurements shown are the resulting average of the last 128 runs out of 256. This averaging is done as the presence of the cache does not guarantee a deterministic operation of the test suite. One must remember the latencies measured apply to the test suite; any other application specific latencies depend if the mAbassi code and data are or are not in the cache.

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

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

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

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

Table 8-3 Measurement without Task Switch

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

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

Table 8-4 Measurement without Blocking

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

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

Table 8-5 Measurement with Task Switch

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

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

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

Table 8-6 Measurement with Task unblocking

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

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

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

Table 8-7 lists the results obtained, where the cycle count is measured using core #0 private timer. This timer decrements its counter by 1 at every 4 CPU cycle

The interrupt latency is the number of cycles elapsed when the interrupt trigger occurred and the ISR function handler is entered. This includes the number of cycles used by the processor to set-up the interrupt stack and branch to the address specified in the interrupt vector table

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

Table 8-7 Latency Measurements

Description	Minimal Features	Full Features
Semaphore posting no task switch	212 (232)	252 (254)
Semaphore waiting no blocking	212 (246)	256 (258)
Semaphore posting with task switch	212 (224)	528 (526)
Semaphore waiting with blocking	364 (400)	480 (480)
Semaphore posting in ISR with task switch	364 (380)	620 (628)
Event setting no task switch	n/a	256 (254)
Event getting no blocking	n/a	390 (390)
Event setting with task switch	n/a	532 (536)
Event getting with blocking	n/a	664 (664)
Event setting in ISR with task switch	n/a	604 (604)
Mailbox writing no task switch	n/a	320 (318)
Mailbox reading no blocking	n/a	306 (308)
Mailbox writing with task switch	n/a	596 (596)
Mailbox reading with blocking	n/a	500 (500)
Mailbox writing in ISR with task switch	n/a	660 (676)
Interrupt Latency	140	140
Context switch	48	48

9 Appendix A: Build Options for Code Size

9.1 Case 0: Minimum build

Table 9-1: 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                          */
```

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

Table 9-2: 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                          */
```

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

Table 9-3: 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                            */
```

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

Table 9-4: 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                            */
```

9.5 Case 4: + Events / Mailboxes

Table 9-5: 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                            */
```

9.6 Case 5: Full feature Build (no names)

Table 9-6: 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                          */
```

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

Table 9-7: 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                          */
```

9.8 Case 7: Full build adding the optional timer services

Table 9-8: 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                          */
```

10 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