CODE TIME TECHNOLOGIES

mAbassi RTOS

Porting Document

SMP / ARM Cortex-A53 – GCC

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

This document is a complement to the mAbassi [R1] and Abassi User Guides [R2] and it details the port for the GCC tool chain of the SMP / BMP multi-core mAbassi RTOS to the ARM Cortex-A53 multi-core processor, commonly known as the A53 MPcore.

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

Table 1-1 Distribution

File Name	Description
`mAbassi.h`	RTOS include file
`mAbassi.c`	RTOS "C" source file
`ARMv8_SMP_L1_L2_GCC.s`	L1 and L2 caches, MMU, and SCU set-up module for the MPcore A53 / GCC
`mAbassi_SMP_CORTEXA53_GCC.s`	RTOS assembly file for the SMP ARM Cortex-A53 to use with the GCC tool chain

1.1 Limitations

Using mAbassi, the A53 always operates (RTOS and application) in the "Secure Monitor" exception level (EL3), which is the highest. Other exception levels should not be used and are not supported. The port currently only operates using the 64 bit instruction set (A64) in the AArch64 mode and the ABI does not support LP32 mode; it is compatible with both LP64 and LLP64. In LP64, `int` are 32 bit, pointers and `long int` are 64 bits. For LLP64, `int` and `long int` are 32 bits, pointers and `long long int` are 64 bits.

1.2 Features

- 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 assembly file never uses `BL` *addr* instructions when calling a function; it uses instead `BLR` `rn` with `rn` holding the full 64 bit address. This was chosen to make the assembly file able to access the whole address space without the need of a trampoline.

2 Target Set-up

Very little is needed use the mAbassi RTOS with an application. All there is to do is to add the files `mAbassi.c`, `mAbassi_SMP_CORTEXA53_GCC.s` and `ARMv8_SMP_L1_L2_GCC.s` to the application build (either through a makefile or with the tool-suite GUI), and make sure the configuration settings in the file `mAbassi_SMP_CORTEXA53_GCC.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_CORTEXA53_GCC.s` takes care of all the start-up operations required for an application to operate on the A53 multi-core processor.

3 Build Options

3.1 Option setting

mAbassi is configured through many build options [R1][R2]. The way to set a build option for the compiler is either by defining the symbol & value through the tool-suite GUI or using the compiler command line option –D *build_option=value* in a makefile.

Table 3-1 Compiler build options setting

```
aarch64-none-eabi-gcc … -DOS_N_CORE=2 …
```

For the assembly files, again, being through the GUI or in the make file with the assembler command line option –defsym *build_option=value*. For example, if the compiler is used for the assembly phase:

Table 3-2 Assembly build options passed through the compiler

```
aarch64-none-eabi-gcc … -Wa,--defsym -Wa,OS_N_CORE=2 …
```

Alternatively, if the assembler is used directly:

Table 3-3 Assembly build options passed through the assembler

```
aarch64-none-eabi-as … --defsym OS_N_CORE=2 …
```

NOTE: most tool-suite A53 GUIs don't offer a "symbol definition" interface or box to fill, therefore the --symdef command line option has to be used in the miscellaneous or extra section. For the correct setting, double check if the compiler or the assembler is used as to assemble code. It is preferable to use the assembler as the miscellaneous / extra section is simpler to fill.

Another way to set the build options for the assembly file is to change the default value directly in the file `mAbassi_SMP_CORTEXA53_GCC.s`. All build options are declared at the top of the file and the individual declarations all look like this:

Table 3-4 Assembly file default build option setting

```
#ifndef OS_N_CORE
 .ifndef OS_N_CORE          // Number of cores the device has (2: min for multicore)
  .equ  OS_N_CORE,      4   // ==1: code almost identical to Abassi single core
 .endif
#endif
```

There build options that allow the assembly file to be configured to the specific target device and needs of the application are listed in the following table:

Table 3-5 Build Options

File Name	Default	Description
OS_PLATFORM	0x00007753	Number indicating the target platform.
OS_N_CORE	4	Number of cores

OS_HANDLE_PSR_Q	0	Handling of the overflow sticky bit
OS_FPCR_LOCAL	1	Local or global FPU configuration
OS_CPU_CLK	Target Dependent	Core clock frequency
OS_NEWLIB_REENT	0	Newlib reentrance protection set-up
OS_SPINLOCK_DELAY	1	Random delay for spinlocks
OS_PERF_TIMER_BASE	1	Base address of the performance timer
OS_PERF_TIMER_DIV	0	Clock divider of the performance timer
OS_PERF_TIMER_ISR	0	Interrupt number of the performance timer
OS_CODE_SOURCERY	1	Variant of GCC tool-chain used

3.2 OS_PLATFORM - 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 by the value assigned to the token OS_PLATFORM. At the time of writing this document, the following platforms are supported:

Table 3-6 OS_PLATFORM valid settings

Target Platform	OS_PLATFORM value
Xilinx / UltrasScale+	0x00007753

If in the future if 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_CORTEXA53_GCC.s., right beside the internal definition of OS_PLATFORM (around line 35). Only the lower 24 bits of value assigned to OS_PLATFORM are taken into account as the upper 8 bits define the target platform. The platform numbering is the one described in the file Platform.txt.

3.3 OS_N_CORE - 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_CORTEXA53_GCC.s file, through the setting of the build option OS_N_CORE (the build option is the same for the "C file and the assembly file). 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_CORTEXA53_GCC.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. By default, the number of cores is set to 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-A53 MPCore, the single core application always executes on core #0.

3.4 L1 & L2 Cache Set-up

Contrary to some other ports, the cache & MMU cannot be disabled and are always used in this port. For more information on how to configure the property of memory regions, refer to the ARMv8 cache document [R3].

3.5 OS_HANDLE_PSR_Q - Saturation Bit Set-up

In the ARM Cortex-A53 floating-point status register (FPSR), there are a few a sticky (cumulative) bits to indicate if overflow, saturation, exception, etc. have occurred. They are the IOC (bit #0), DZC (bit #1), OFC (bit #2), UFC (bit #3), IXC (bit #4), IDC (bit #7) and QC (bit #27). By default, these bit are not kept localized at the task level, as extra processing is required during the task context switch to do so; instead, their values is shared across all tasks. This choice was made because most applications do not care about the value of this bit.

If these bits are relevant for an application, even for a single task, then they must be kept local to each task. To keep the value of these bits localized, the token `OS_HANDLE_PSR_Q` must be set to a non-zero value; to disable the localization as it is by default it must be set to a zero value.

The token name, `OS_HANDLE_PSR_Q`, is a misnomer but is retained because it's the name of the token used in other ports.

3.6 OS_CPU_CLK – Processor clocking frequency

The A53 generic timer needs to be informed of the processor clocking frequency. The build option `OS_CPU_CLK` is used to set-up the generic timer frequency register. The default value is determined by the build option `OS_PLATFORM`, but can be overloaded by externally defining the built option `OS_CPU_CLK`.

3.7 OS_NEWLIB_RENT - Multithreading

This build option is required when the Newlib has to be multi-thread safe. Refer to the library protection document [R4].

3.8 OS_SPINLOCK_DELAY - Spinlock implementation

There is a possible race condition when using spinlocks on a target processor with 3 cores or more. That race condition occurs when the tasks running on 3+ cores are all trying non-stop to lock and unlock the same spinlock. When this situation arises, it is possible the same 2 cores always get the spinlock, starving the other one(s). This is not an issue with mAbassi itself but it is due to the fact that it looks like the snoop control unit (SCU) of the cache, when 2 or more core are trying to obtain a lock at exactly the same time, does not assign the lock in a random order nor in round-robin but instead, it looks like it assigns the lock to the core with the lowest index. To randomize which core is 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, enabling the random delay on spinlock when the number of cores, indicated by `OS_N_CORE`, is greater than 2.

3.9 Performance Monitoring

The performance monitoring facility relies on a fine resolution timer / counter and this timer / counter is set-up and handled in the file `mAbassi_SMP_CORTEXA53_GCC.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-A53 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 1. The two other options are ignored when the global timer is selected.

For other counter / timer options, please look directly into the file `mAbassi_SMP_CORTEXA53_GCC.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.

3.10 OS_CODE_SOURCERY - Code Sourcery / Linaro

There are two main trunks of the GCC for ARM; one is maintained by Mentor Graphics and is named Code Sourcery, the other is named Linaro, with Atollic, for example, using this variant; Also know as the later are Yagarto and the GCC tool chain maintained by KEIL/ARM.

There are small differences between the two related to the start-up code and the "C" library. Both variants are supported as indicated through the token `OS_CODE_SOURCERY`. By default, this token is set to a non-zero value to fulfill the start-up and library requirements of the Code Sourcery variant. If the Linaro variant is used instead, the token `OS_CODE_SOURCERY` must be set to a value of zero.

4 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 1024 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`. The Generic Interrupt Controller (GIC) peripheral supports a maximum of 1024 interrupts and setting `OX_N_INTERRUPTs` to a smaller value reduces the size of the interrupt table.

4.1 Interrupt Handling

4.1.1 Interrupt Table Size

Most devices do not require all 1024 interrupt, as they may only handle between 256 and 512 sources of interrupts; or some very large device may require more than 512. The interrupt table can be easily reduced to recover data space if needed. 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 4-1 Command line set the interrupt table size

```
arm-xilinx-eabi-gcc … -D=OS_N_INTERRUPTS=49 …
```

4.1.2 Interrupt Installer

Attaching a function to a regular interrupt is quite straightforward. All there is to do is use the RTOS component `OSisrInstall()` to specify the interrupt number and the function to be attached to that interrupt number. For example, Table 4-2 shows the code required to attach the interrupt number fedined by the token `OS_TICK_INT` to the RTOS timer tick handler (`TIMtick`):

Table 4-2 Attaching a Function to an Interrupt

```
#include “mAbassi.h”

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

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

   … /* More ISR setup */

   OSintOn();                               /* 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 4-3:

Table 4-3 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 `OSintOff()` and `OSintBack()` 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

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-A53, there is no need to add a critical region protection, as interrupts are disabled when processing an interrupt.

4.2 Fast Interrupts

Fast interrupts are supported on this port as the FIQ interrupts. The ISR dispatcher is designed to only handle the IRQ interrupts. A default do-nothing FIQ handler is supplied with the distribution; the application can overload the default handler (Section 8.1).

4.3 Nested Interrupts

Interrupt nesting, other than a FIQ nesting an IRQ, is not supported on this port. The reason is simply based on the fact the Generic Interrupt Controller is not a nested controller.

5 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; this includes the integer registers and the FPU registers). 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-A53, 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 5-1 Context Save Stack Requirements

Description	Context save
Blocked/Preempted task context save	`240 bytes`
Blocked/Preempted task context save (FPCR local: `OS_FPCR_LOCAL != 0`)	`+16 bytes`
Interrupt dispatcher context save	`624 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.

6 Memory Configuration

The mAbassi kernel is not a kernel entered though a service request, such as the SVC / HVC / SMC on the Cortex-A53. The kernel is a regular function, protected against re-entrance / multiple core entrance. The kernel code executes as part of the application code, with the same processor mode and access privileges.

7 Search Set-up

The mAbassi RTOS build option `OS_SEARCH_ALGO` offers three different algorithms to quickly determine the next running task upon task blocking. The following table shows the measurements obtained for the number of CPU cycles required when a task at priority 0 is blocked, and the next running task is at the specified priority. The number of cycles includes everything, not just the search cycle count. The number of cycles was measured using the performance monitoring cycle counter, which increments the counter once every CPU cycle. The second column is when `OS_SEARCH_ALGO` is set to zero, meaning a simple array traversing. The third column, labeled Look-up, is when `OS_SEARCH_ALGO` is set to 1, which uses an 8 bit look-up table. Finally, the last column is when `OS_SEARCH_ALGO` is set to 5 (GCC/Cortex-A53 `int` are 32 bits, so 2^5), meaning a 32 bit look-up table, further searched through successive approximation. The compiler optimization for this measurement was set to `-O3`, meaning maximum optimization for speed. The RTOS build options were set to the minimum feature set, except for option `OS_PRIO_CHANGE` set to non-zero. The presence of this extra feature provokes a small mismatch between the result for a difference of priority of 1, with `OS_SEARCH_FAST` set to zero, and the latency results in Section 9.2.

When the build option `OS_SEARCH_ALGO` is set to a negative value, indicating to use a 2-dimensional linked list search technique instead of the search array, the number of CPU cycles is constant at 202 cycles.

Table 7-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	211	237	240
2	213	237	240
3	234	244	240
4	226	251	240
5	233	258	240
6	240	265	240
7	247	272	240
8	269	236	240
9	261	233	240
10	261	240	240
11	283	247	240
12	295	254	240
13	302	261	240
14	313	268	240
15	320	375	240
16	327	232	240
17	337	237	240
18	344	244	240
19	351	251	240
20	358	258	240
21	365	265	240
22	372	272	240
23	379	279	240
24	386	236	240

When `OS_SEARCH_FAST` is set to 0, each extra priority level to traverse requires about 7 CPU cycles. When `OS_SEARCH_FAST` is set to 1, each extra priority level to traverse requires 7 CPU cycles, except when the priority level is an exact multiple of 8; then there is a sharp reduction of CPU usage. Overall, setting `OS_SEARCH_FAST` to 1 adds around 20 cycles of CPU for the search, compared to setting `OS_SEARCH_FAST` to zero. But when the next ready to run priority is less than 8, 16, 24, … then there are no extra cycles needed, but without the 8 times 6 cycle accumulation. Finally, the third option, when `OS_SEARCH_FAST` is set to 5, delivers a constant CPU usage, as the algorithm utilizes a successive approximation search technique (when the delta is 32 or more, the CPU cycle count is 244±1, for 64 or more, it is 252±1, …).

8 API

The ARM Cortex-A53 supports a few 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 preceded by an "`hlt`" instruction to help during debugging (except FIQ, which is a do-nothing with return from exception). The "`hlt`" instruction is followd by an infinite loop on the "`wfi`" instruction. All handlers (including the default) are called after all callee modifiable integer registers (`X0` to `X19`, and `X30`) have been preserved on the stack. The following sub-sections describe each one of the default exception handlers.

8.1 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. Although this is an exception handler, it is called through a trampoline that saves all registers (integer and FPU) a callee can modify, therefore the return must be performed with a "`ret`", and not an `"eret"`.

Availability

Always.

Arguments

`void`

Returns

`void`

Component type

Function

Options

Notes

See also

`SError_Handler()` (Section **Error! Reference source not found.**)
`Sync_Handler()` (Section 8.2)

8.2 SError_Handler

Synopsis

```
#include "mAbassi.h"

void SError_Handler(uint32_t Syndrome, void *Ret, uint32_t PSstate,
                    int Level);
```

Description

`SError_Handler()` is the exception handler for a system error. In the distribution code, this is implemented as an infinite loop, preceded with an "`hlt`" instruction. If the application needs to perform special processing when a system error 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. Although this is an exception handler, it is called through a trampoline that saves all registers (integer and FPU) a callee can modify, therefore the return must be performed with a "`ret`", and not an `"eret"`.

Availability

Always.

Arguments

`Syndrome`	Syndrome of the exception (`ESR_ELn` register).
`Ret`	Return address, which is the address of the instruction right after the one that triggered the exception (`ELR_ELn` register).
`PState`	PState register when the exception occurred (`SPSR_ELn` register).
`Level`	Exception level at which this exception occurred. This is normally 3

Returns

`void`

Component type

Function

Options

Notes

See also

`FIQ_Handler()` (Section 8.1)
`Sync_Handler()` (Section 8.3)

8.3 Sync_Handler

Synopsis

```
#include "mAbassi.h"

void Sync_Handler(uint32_t Syndrome, void *Ret, uint32_t PSstate,
                  int Level);
```

Description

`Sync_Handler()` is the exception handler for synchronous abort. In the distribution code, this is implemented as an infinite loop, preceded with an "`hlt`" instruction. If the application needs to perform special processing when a system error 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. Although this is an exception handler, it is called through a trampoline that saves all registers (integer and FPU) a callee can modify, therefore the return must be performed with a "`ret`", and not an `"eret"`.

Availability

Always.

Arguments

`Syndrome`	Syndrome of the exception (`ESR_ELn` register).
`Ret`	Return address, which is the address of the instruction right after the one that triggered the exception (`ELR_ELn` register).
`PState`	PState register when the exception occurred (`SPSR_ELn` register).
`Level`	Exception level at which this exception occurred. This is normally 3

Returns

`void`

Component type

Function

Options

Notes

See also

`FIQ_Handler()` (Section 8.1)
`SError_Handler()` (Section 8.2)

8.4 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 (0 to 1019)
`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-A53 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_CORTEXA53_GCC.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 0)

8.5 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_CORTEXA53_GCC.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 8.4)

9 Measurements

This section provides an overview of the memory requirements encountered when the RTOS is used on the Arm9 and compiled with Mentor's Code Sourcery 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 9.2)

9.1 Memory

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

The code size numbers are expressed with "less than" as they have been rounded up to multiples of 25 for the "C" code. These numbers were obtained using the release version 1.99.100 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: `-Os`
3. Target `–mcpu=cortex-A53`

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

Table 9-1 "C" Code Memory Usage

Description	AArch64
Minimal Build	< 2550 bytes
+ Runtime service creation / static memory	< 3075 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 3750 bytes
+ Timer & timeout + Timer call back + Round robin	< 4850 bytes
+ Events + Mailbox	< 6450 bytes
Full Feature Build (no names)	< 7525 bytes
Full Feature Build (no name / no runtime creation)	< 6775 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 7325 bytes
OS_NEWLIB_REENT > 0	+TBD bytes
OS_NEWLIB_REENT < 0	+TBD bytes
True SMP (OS_MP_TYPE == 2)	+0 bytes
Packed SMP (OS_MP_TYPE == 3)	~ +50 bytes
True BMP (OS_MP_TYPE == 4)	~ +300 bytes
Packed BMP (OS_MP_TYPE == 5)	~ +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 9-2 Assembly Code Memory Usage

Description	Size
Vector table & error handlers	1972 bytes
Assembly code size (>1 core)	2116 bytes
Assembly code size (==1 core)	1512 bytes
Saturation Bit Enabled (`OS_HANDLE_PSR_Q != 0`)	+24 bytes
`GICinit()` (>1 core)	+196 bytes
`GICinit()` (=1 core)	+120 bytes
`GICenable()` (>1 core)	+400 bytes
`GICenable()` (=1 core)	+276 bytes
`OS_NEWLIB_REENT > 0`	+TBD bytes
`OS_NEWLIB_REENT < 0`	+TBD bytes
Xilinx UltraScale+ Support	+92 bytes
Cache Driver	1604 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.

9.2 Latency

Latency of operations has been measured on a Xilinx ZCU-102 Evaluation board populated with a 1.2GHz quad-core Cortex-A53. All measurements have been performed on the real platform. This means the interrupt latency measurements had to be instrumented to read the performance monitor cycle counter value. This instrumentation adds a few cycles to the measurements but is measured and removed from the final cycle count. The code optimization setting used for the latency measurements is `-O3`, 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 type of multi-processing was set to true SMP (`OS_MP_TYPE` set to 2). The cache is enabled. 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 9-3 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 9-4 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 9-5 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 measurements for task switching include the context switch cycle count.

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 9-6 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 measurements for task unblocking include the context switch 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 9-7 lists the results obtained, where the cycle count is measured using the performance monitoring cycle counter.

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 save registers, retrieve the address of the handler from the interrupt vector table and call the handler.

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 (linked list search). The regular number is the latency measurements when the build option `OS_SEARCH_ALGO` is set to 0.

Table 9-7 Latency Measurements

Description	Minimal Features	Full Features
Semaphore posting no task switch	76 (68)	107 (107)
Semaphore waiting no blocking	87 (79)	117 (121)
Semaphore posting with task switch	173 (195)	252 (252)
Semaphore waiting with blocking	205 (194)	260 (261)
Semaphore posting in ISR with task switch	--- (---)	--- (---)
Event setting no task switch	n/a	88 (88)
Event getting no blocking	n/a	117 (117)
Event setting with task switch	n/a	260 (260)
Event getting with blocking	n/a	269 (270)
Event setting in ISR with task switch	n/a	--- (---)
Mailbox writing no task switch	n/a	122 (122)
Mailbox reading no blocking	n/a	143 (144)
Mailbox writing with task switch	n/a	286 (291)
Mailbox reading with blocking	n/a	270 (270)
Mailbox writing in ISR with task switch	n/a	--- (---)
Interrupt Latency	---	---
Context switch	81	81

10 Appendix A: Build Options for Code Size

10.1 Case 0: Minimum build

Table 10-1: Case 0 build options

```
#define OS_ALLOC_SIZE       0    /* When !=0, RTOS supplied OSalloc                  */
#define OS_COOPERATIVE      0    /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS           0    /* If event flags are supported                     */
#define OS_FCFS             0    /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK       0    /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE     0    /* Type of logging to use                           */
#define OS_MAILBOX          0    /* If mailboxes are used                            */
#define OS_MAX_PEND_RQST    2U   /* Maximum number of requests in ISRs               */
#define OS_MP_TYPE          2U   /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK     0    /* This test validates this                         */
#define OS_MTX_INVERSION    0    /* To enable protection against priority inversion  */
#define OS_N_CORE           2    /* Number of cores to handle                        */
#define OS_NAMES            0    /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS      0    /* If operating with nested interrupts              */
#define OS_PRIO_CHANGE      0    /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN         2    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME        1    /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN      0    /* Use round-robin, value specifies period in uS    */
#define OS_RUNTIME          0    /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO      0    /* If using a fast search                           */
#define OS_STACK_START      256  /* Stack sie of the start-up / ilde functions       */
#define OS_STARVE_PRIO      0    /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX   0    /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX  0    /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX   0    /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX       0    /* If !=0 how many mailboxes                        */
#define OS_STATIC_NAME      0    /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM       0    /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK     0    /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK      0    /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND     0    /* If a task can suspend another one                */
#define OS_TIMEOUT          0    /* !=0 enables blocking timeout                     */
#define OS_TIMER_CB         0    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module           */
#define OS_TIMER_US         0    /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG     0    /* If tasks have arguments                          */
```

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

Table 10-2: Case 1 build options

```
#define OS_ALLOC_SIZE       0    /* When !=0, RTOS supplied OSalloc                  */
#define OS_COOPERATIVE      0    /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS           0    /* If event flags are supported                     */
#define OS_FCFS             0    /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK       0    /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE     0    /* Type of logging to use                           */
#define OS_MAILBOX          0    /* If mailboxes are used                            */
#define OS_MAX_PEND_RQST    32U  /* Maximum number of requests in ISRs               */
#define OS_MP_TYPE          2U   /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK     0    /* This test validates this                         */
#define OS_MTX_INVERSION    0    /* To enable protection against priority inversion  */
#define OS_N_CORE           2    /* Number of cores to handle                        */
#define OS_NAMES            0    /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS      0    /* If operating with nested interrupts              */
#define OS_PRIO_CHANGE      0    /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN         20   /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME        1    /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN      0    /* Use round-robin, value specifies period in uS    */
#define OS_RUNTIME          1    /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO      0    /* If using a fast search                           */
#define OS_STACK_START      256  /* Stack sie of the start-up / ilde functions       */
#define OS_STARVE_PRIO      0    /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX   0    /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX  0    /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX   0    /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX       0    /* If !=0 how many mailboxes                        */
#define OS_STATIC_NAME      0    /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM       5    /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK     128  /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK      5    /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND     0    /* If a task can suspend another one                */
#define OS_TIMEOUT          0    /* !=0 enables blocking timeout                     */
#define OS_TIMER_CB         0    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module           */
#define OS_TIMER_US         0    /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG     0    /* If tasks have arguments                          */
```

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

Table 10-3: Case 2 build options

```
#define OS_ALLOC_SIZE       0    /* When !=0, RTOS supplied OSalloc                    */
#define OS_COOPERATIVE      0    /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS           0    /* If event flags are supported                       */
#define OS_FCFS             1    /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK       0    /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE     0    /* Type of logging to use                             */
#define OS_MAILBOX          0    /* If mailboxes are used                              */
#define OS_MAX_PEND_RQST    32   /* Maximum number of requests in ISRs                 */
#define OS_MP_TYPE          2    /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK     0    /* This test validates this                           */
#define OS_MTX_INVERSION    1    /* To enable protection against priority inversion  */
#define OS_N_CORE           2    /* Number of cores to handle                          */
#define OS_NAMES            0    /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS      0    /* If operating with nested interrupts              */
#define OS_PRIO_CHANGE      1    /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN         20   /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME        1    /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN      0    /* Use round-robin, value specifies period in uS    */
#define OS_RUNTIME          1    /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO      0    /* If using a fast search                             */
#define OS_STACK_START      256  /* Stack sie of the start-up / ilde functions       */
#define OS_STARVE_PRIO      0    /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX   0    /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX  0    /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX   0    /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX       0    /* If !=0 how many mailboxes                          */
#define OS_STATIC_NAME      0    /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM       5    /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK     128  /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK      5    /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND     1    /* If a task can suspend another one                */
#define OS_TIMEOUT          0    /* !=0 enables blocking timeout                       */
#define OS_TIMER_CB         0    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module           */
#define OS_TIMER_US         0    /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG     0    /* If tasks have arguments                            */
```

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

Table 10-4: Case 3 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                    */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS          0     /* If event flags are supported                       */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                             */
#define OS_MAILBOX         0     /* If mailboxes are used                              */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                 */
#define OS_MP_TYPE         2     /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK    0     /* This test validates this                           */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_N_CORE          2     /* Number of cores to handle                          */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts              */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN     100000/* Use round-robin, value specifies period in uS    */
#define OS_RUNTIME         1     /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO     0     /* If using a fast search                           */
#define OS_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#define OS_STARVE_PRIO     0     /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX  0     /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX 0     /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX  0     /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX      0     /* If !=0 how many mailboxes                        */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                     */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module           */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG    0     /* If tasks have arguments                          */
```

10.5 Case 4: + Events / Mailboxes

Table 10-5: Case 4 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS          0     /* If event flags are supported                      */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                            */
#define OS_MAILBOX         0     /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                */
#define OS_MP_TYPE         2     /* SMP vs BMP and load balancing selection           */
#define OS_MTX_DEADLOCK    0     /* This test validates this                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion   */
#define OS_N_CORE          2     /* Number of cores to handle                         */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN     100000/* Use round-robin, value specifies period in uS     */
#define OS_RUNTIME         1     /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO     0     /* If using a fast search                            */
#define OS_STACK_START     256   /* Stack sie of the start-up / ilde functions        */
#define OS_STARVE_PRIO     0     /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX  0     /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX 0     /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX  0     /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX      0     /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                 */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module            */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG    0     /* If tasks have arguments                           */
```

10.6 Case 5: Full feature Build (no names)

Table 10-6: Case 5 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                    */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS          1     /* If event flags are supported                       */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                             */
#define OS_MAILBOX         1     /* If mailboxes are used                              */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                 */
#define OS_MP_TYPE         2     /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK    0     /* This test validates this                           */
#define OS_N_CORE          2     /* Number of cores to handle                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts              */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN     -100000 /* Use round-robin, value specifies period in uS  */
#define OS_RUNTIME         1     /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO     0     /* If using a fast search                           */
#define OS_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#define OS_STARVE_PRIO     -3    /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX  -10   /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX -100  /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX  100   /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX      2     /* If !=0 how many mailboxes                        */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                     */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module           */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG    1     /* If tasks have arguments                          */
```

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

Table 10-7: Case 6 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                    */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative  */
#define OS_EVENTS          1     /* If event flags are supported                       */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore    */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size         */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                             */
#define OS_MAILBOX         1     /* If mailboxes are used                              */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                 */
#define OS_MP_TYPE         2     /* SMP vs BMP and load balancing selection          */
#define OS_MTX_DEADLOCK    0     /* This test validates this                           */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion  */
#define OS_N_CORE          2     /* Number of cores to handle                          */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes   */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts              */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time    */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0   */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority    */
#define OS_ROUND_ROBIN     -100000 /* Use round-robin, value specifies period in uS  */
#define OS_RUNTIME         0     /* If create Task / Semaphore / Mailbox at run time */
#define OS_SEARCH_ALGO     0     /* If using a fast search                           */
#define OS_STACK_START     256   /* Stack sie of the start-up / ilde functions       */
#define OS_STARVE_PRIO     -3    /* Priority threshold for starving protection       */
#define OS_STARVE_RUN_MAX  -10   /* Maximum Timer Tick for starving protection       */
#define OS_STARVE_WAIT_MAX -100  /* Maximum time on hold for starving protection     */
#define OS_STATIC_BUF_MBX  0     /* when OS_STATIC_MBOX != 0, # of buffer element    */
#define OS_STATIC_MBX      0     /* If !=0 how many mailboxes                        */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char  */
#define OS_STATIC_SEM      0     /* If !=0 how many semaphores and mutexes           */
#define OS_STATIC_STACK    0     /* if !=0 number of bytes for all stacks            */
#define OS_STATIC_TASK     0     /* If !=0 how many tasks (excluding A&E and Idle)   */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                     */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period              */
#define OS_TIMER_SRV       0     /* !=0 includes the timer services module           */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS   */
#define OS_USE_TASK_ARG    1     /* If tasks have arguments                          */
```

10.8 Case 7: Full build adding the optional timer services

Table 10-8: Case 7 build options

```
#define OS_ALLOC_SIZE      0     /* When !=0, RTOS supplied OSalloc                   */
#define OS_COOPERATIVE     0     /* When 0: pre-emptive, when non-zero: cooperative   */
#define OS_EVENTS          1     /* If event flags are supported                      */
#define OS_FCFS            1     /* Allow the use of 1st come 1st serve semaphore     */
#define OS_IDLE_STACK      0     /* If IdleTask supplied & if so, stack size          */
#define OS_LOGGING_TYPE    0     /* Type of logging to use                            */
#define OS_MAILBOX         1     /* If mailboxes are used                             */
#define OS_MAX_PEND_RQST   32    /* Maximum number of requests in ISRs                */
#define OS_MP_TYPE         2     /* SMP vs BMP and load balancing selection           */
#define OS_MTX_DEADLOCK    0     /* This test validates this                          */
#define OS_MTX_INVERSION   1     /* To enable protection against priority inversion   */
#define OS_N_CORE          2     /* Number of cores to handle                         */
#define OS_NAMES           0     /* != 0 when named Tasks / Semaphores / Mailboxes    */
#define OS_NESTED_INTS     0     /* If operating with nested interrupts               */
#define OS_PRIO_CHANGE     1     /* If a task priority can be changed at run time     */
#define OS_PRIO_MIN        20    /* Max priority, Idle = OS_PRIO_MIN, AdameEve = 0    */
#define OS_PRIO_SAME       1     /* Support multiple tasks with the same priority     */
#define OS_ROUND_ROBIN     -100000 /* Use round-robin, value specifies period in uS   */
#define OS_RUNTIME         0     /* If create Task / Semaphore / Mailbox at run time  */
#define OS_SEARCH_ALGO     0     /* If using a fast search                            */
#define OS_STACK_START     256   /* Stack sie of the start-up / ilde functions        */
#define OS_STARVE_PRIO     -3    /* Priority threshold for starving protection        */
#define OS_STARVE_RUN_MAX  -10   /* Maximum Timer Tick for starving protection        */
#define OS_STARVE_WAIT_MAX -100  /* Maximum time on hold for starving protection      */
#define OS_STATIC_BUF_MBX  100   /* when OS_STATIC_MBOX != 0, # of buffer element     */
#define OS_STATIC_MBX      2     /* If !=0 how many mailboxes                         */
#define OS_STATIC_NAME     0     /* If named mailboxes/semaphore/task, size in char   */
#define OS_STATIC_SEM      5     /* If !=0 how many semaphores and mutexes            */
#define OS_STATIC_STACK    128   /* if !=0 number of bytes for all stacks             */
#define OS_STATIC_TASK     5     /* If !=0 how many tasks (excluding A&E and Idle)    */
#define OS_TASK_SUSPEND    1     /* If a task can suspend another one                 */
#define OS_TIMEOUT         1     /* !=0 enables blocking timeout                      */
#define OS_TIMER_CB        10    /* !=0 gives the timer callback period               */
#define OS_TIMER_SRV       1     /* !=0 includes the timer services module            */
#define OS_TIMER_US        50000 /* !=0 enables timer & specifies the period in uS    */
#define OS_USE_TASK_ARG    1     /* If tasks have arguments                           */
```

11 References

[R1] mAbassi RTOS – User Guide, available at http://www.code-time.com
[R2] Abassi RTOS – User Guide, available at http://www.code-time.com
[R3] mAbassi – ARMv8 Caches (GCC), available at http://www.code-time.com
[R4] Abasi RTOS – Library Reentrance Protection, available at http://www.code-time.com