CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

ARM Cortex-A9 – CCS

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

This document details the port of the Abassi RTOS to the ARM Cortex-A9 processor, commonly known as Arm9. The port is also valid for the ARMv4, ARMv5, ARMv6 and ARMv7 core architectures. The software suite used for this specific port is the Code Composer Studio from Texas Instruments (abbreviated CCS); the version used for the port and all tests is Version 5.2.0.00069.

1.1 Distribution Contents

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

Table 1-1 Distribution

File Name	Description
`Abassi.h`	Include file for the RTOS
`Abassi.c`	RTOS "C" source file
`Abassi_CORTEXA9_CCS.s`	RTOS assembly file for the ARM Cortex-A9 to use with the Code Composer Studio
`Demo_3_PANDA_A9_CCS.c`	Demo code that runs on the Pandaboard ES evaluation board
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

The RTOS uses `SWI` (software interrupts) numbers 0, 1 and 2. A hook is made available for the application to use the `SWI`, as long as the numbers used are not 0, 1 or 2.

Fast Interrupts (FIQ) are not handled by the RTOS, as they are left untouched by the RTOS to fulfill their intended purpose of interrupts not requiring kernel accesses. 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 (General Interrupt Controller) does not allow nesting of the interrupts (except FIQ nesting the IRQ).

Some linker issues have been found when the test suite was run. One should avoid setting the optimization level to 4 as it was found a few time that calls to the library function `strcpy()` was replaced by faulty code. When the optimization is set to 4, it enables the linker to inline function code that is called once, or when in-lining produces smaller code than a function call.

1.3 Features

All tasks run in User mode.

The assembly file was coded using only ARMv4 non-superseded instructions.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

The assembly file does not use the BL or BLX instruction when branching/calling an external module. This was done to first allow the assembly file to access the whole 4 Gigabytes address space and, second, to be usable with ARMv4 devices.

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

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

All Code Composer application binary interfaces (tiabi, ti_arm9_abi, and eabi) are supported in the assembly file.

2 Target Set-up

Very little is needed to configure the Code Composer Studio development environment to use the Abassi RTOS in an application. All there is to do is to add the files `Abassi.c` and `Abassi_CORTEXA9_CCS.s` in the source files of the application project, and make sure the configuration settings in the file `Abassi_CORTEXA9_CCS.s` (the 6 stack size definitions, as described in Section 2.1, `OS_HANDLE_PSR_Q` as described in Section 2.2, and the VFP setting, as described in Section 2.3) are set according to the needs of the application. As well, update the include file path in the C/C++ compiler preprocessor options with the location of `Abassi.h`. There is no need to include a start-up file, as `Abassi_CORTEXA9_CCS.s` is the start-up file.

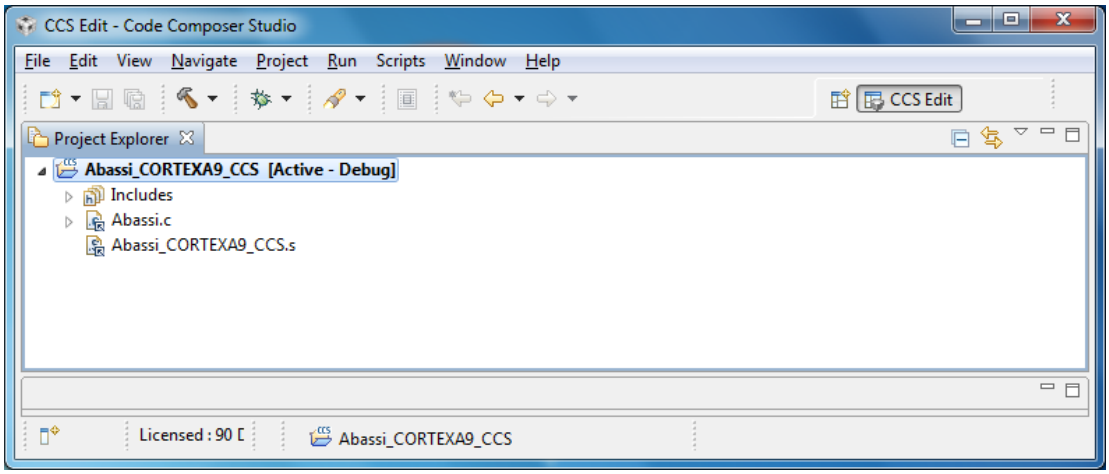

Figure 2-1 Project File List

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

2.1 Stacks Set-up

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

Table 2-1 Stack Size Tokens

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

The User / System mode stack size is defined with the linker line option `-stack_size`. Or through the GUI in the *"Properties"* menu *"Build / ARM Linker / Basic Options / Set C system stack size (--stack_size, -stack)"*.

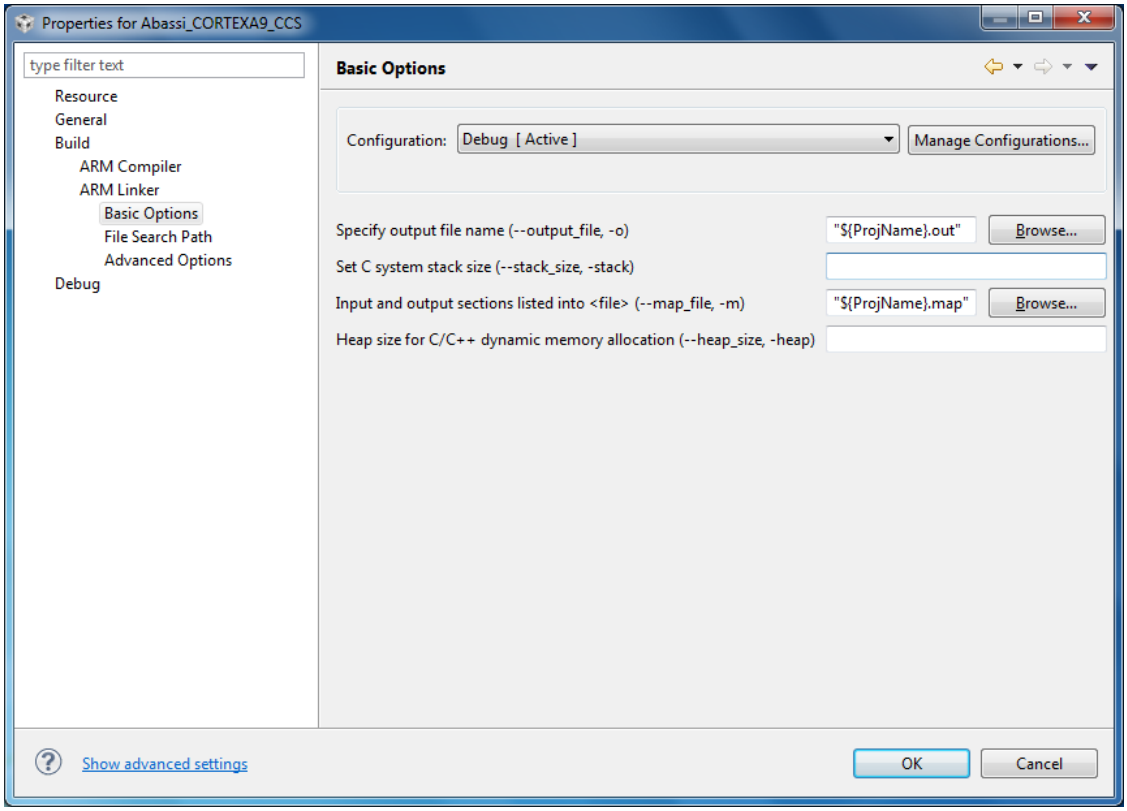

Figure 2-2 GUI set of System Stack Size

The other stack sizes are individuality controlled by the value set by the definition `OS_XXX_STACK_SIZE`, located between lines 30 and 50 in the file `Abassi_CORTEXA9_CCS.s`. To not reserve a stack, set the definition of `OS_XXX_STACK_SIZE` to a value of zero. To specify the stack size, set the definition of `OS_XXX_STACK_SIZE` to the desired size in bytes (see Section 4 for information on stack sizing). As supplied in the distribution, the Supervisor and IRQ stacks are set to a size of 512 bytes each, and the FIQ, Undefined and Abort stacks are all set to a size of 64 bytes each.

To modify the size of a stack, all there is to do is to change the numerical value associated to the token; taking the IRQ stack for example, setting its stack size to 1024 bytes is shown in the following table:

Table 2-2 `OS_IRQ_STACK_SIZE` modification

```
  .if !($$defined(OS_IRQ_STACK_SIZE))
OS_IRQ_STACK_SIZE   .equ 1024
  .endif
```

Alternatively, it is possible to overload the `OS_XXX_STACK_SIZE` value set in `Abassi_CORTEXA9_CCS.s` by using the assembler command line option `-asm_define` and specifying the desired hybrid stack size, as shown in the following example, where the Supervisor stack size is set to 1024 bytes:

Table 2-3 Command line set of `OS_SUPER_STACK_SIZE`

```
cl470  … -asm_define=OS_SUPER_STACK_SIZE=1024  …
```

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

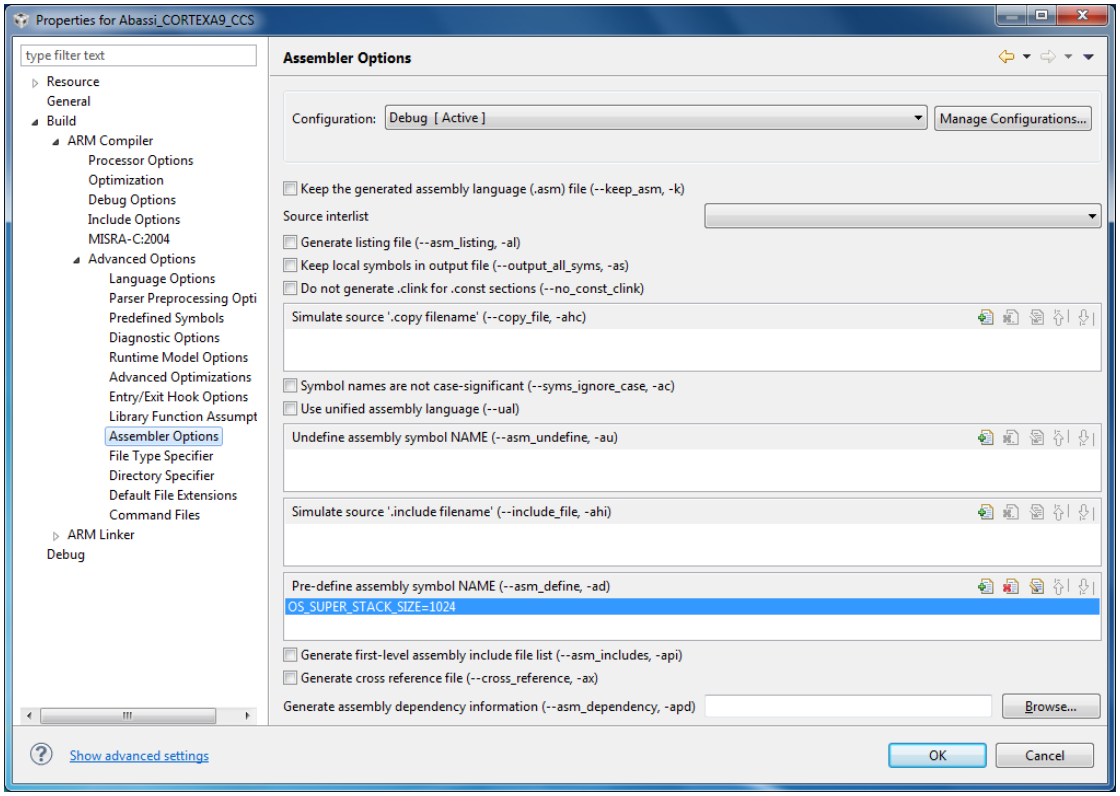

Figure 2-3 GUI set of `OS_SUPER_STACK_SIZE`

2.2 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 it needs extra processing during a context switch to do so; instead, it is propagated across all tasks. This choice was made because most applications do not care about the value of this bit.

If this bit is relevant for an application, even in a single task, then it must be kept locally in each task. To keep the meaning of the saturation bit localized, the token `OS_HANDLE_PSR_Q` must be set to a non-zero value; to disable it, it must be set to a zero value. This is located at around line 25 in the file `Abassi_CORTEXA9_CCS.s`. The distribution code disables the localization of the Q bit, setting the token `OS_HANDLE_PSR_Q` to zero, as shown in the following table:

Table 2-4 Saturation Bit configuration

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

Alternatively, it is possible to overload the `OS_HANDLE_PSR_Q` value set in `Abassi_CORTEXA9_CCS.s` by using the assembler command line option `-asm_define` and specifying the desired setting with the following:

Table 2-5 Command line set of Saturation Bit configuration

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

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

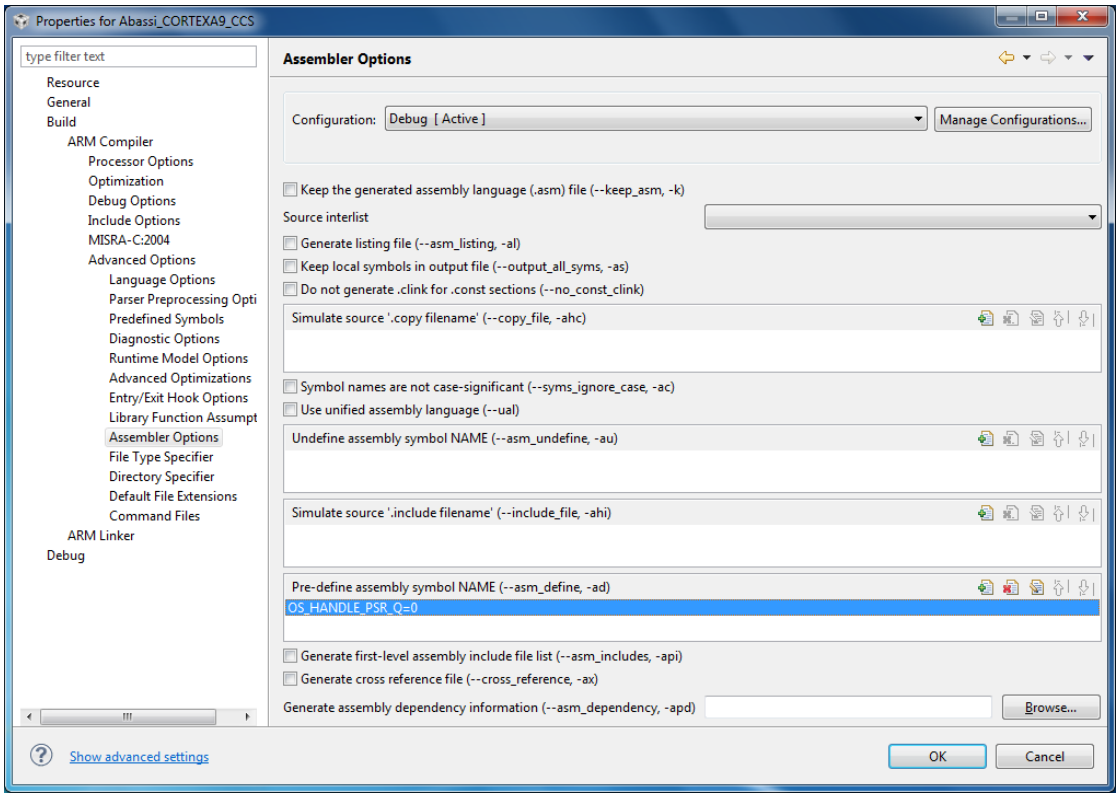

Figure 2-4 GUI set of Saturation Bit configuration

NOTE: The saturation bit is not supported by the ARMv4 architecture. Enabling `OS_HANDLE_PSR_Q` has no negative effect, aside from making the context switch take a bit more CPU for no reason.

2.3 VFP / NEON set-up

The assembly file `Abassi_CORTEXA9_CCS.s`, depending on its configuration, handles four different types of VFP. They are:

- No VPU coprocessor
- VPUv3FPU
- VPUv3D16
- NEON

The file `Abassi_CORTEXA9_CCS.s` is aware of the presence of a VFP, and the type of VFP, when the assembler command line option `-float_support` is used, or when set through the GUI, in the "*Build / ARM Compiler / Processor Options Options*" menu, as shown in the following figure:

--FIGURE—

Table 2-6 Command line enabling of the VFPv3

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

Table 2-7 Command line enabling of the VFPv3D16

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

Table 2-8 Command line enabling of the NEON

```
Cl470 … --neon …
```

3 Interrupts

The Abassi RTOS needs to be aware when kernel requests are performed inside or outside an interrupt context. For all IRQ sources, the Abassi RTOS provides an interrupt dispatcher, which allows it to be interrupt-aware. This dispatcher achieves two goals. First, the kernel uses it to know if a request occurs within an interrupt context or not. Second, using this dispatcher reduces the code size, as all interrupts share the same code for the decision making of entering the kernel or not at the end of the interrupt.

The distribution makes provision for 256 sources of interrupts, as specified by the token `OS_N_INTERRUPTS` in the file `Abassi_CortexA9_CCS`.s, and the internal default value used by `Abassi.c`. Even though the Generic Interrupt Controller (GIC) peripheral supports a maximum of 1244 interrupts, it was decided to set the distribution maximum value to 256 as this seems to be a typical maximum supported by the different devices on the market.

3.1 Interrupt Handling

3.1.1 Interrupt Table Size

Most devices do not require all 256 interrupts, as they typically only handle between 64 and 128 sources of interrupts; or some devices require more than 256. The interrupt table can be easily reduced to recover code space, and at the same time recover the same amount of data memory. All there is to do is define the build option `OS_N_INTERRUPTS` to the desired value. This can be done by using the compiler command line option `-D` and specifying the desired setting with the following:

Table 3-1 Command line set the interrupt table size

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

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

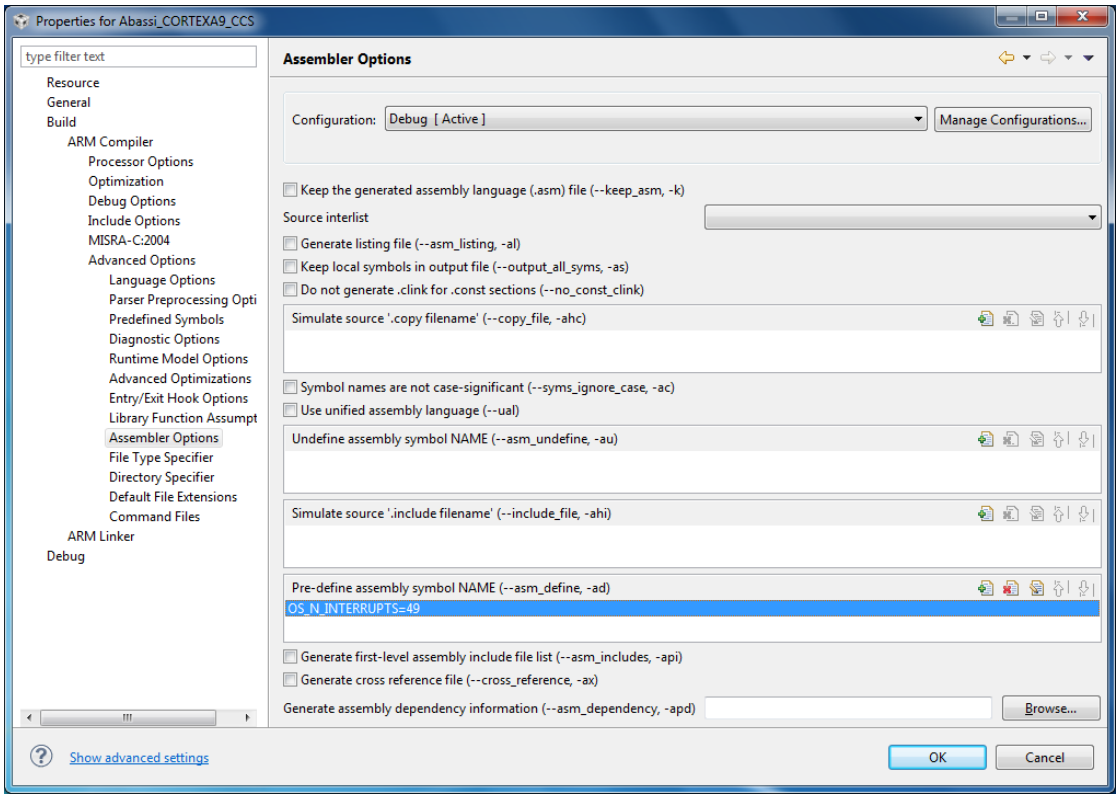

Figure 3-1 GUI set of `OS_N_INTERRUPTS`

3.1.2 Interrupt Installer

Attaching a function to a regular interrupt is quite straightforward. All there is to do is use the RTOS component `OSisrInstall()` to specify the interrupt number and the function to be attached to that interrupt number. For example, Table 3-2 shows the code required to attach the private timer interrupt on an OMAP4460 (ID #29) to the RTOS timer tick handler (`TIMtick`):

Table 3-2 Attaching a Function to an Interrupt

```
#include "Abassi.h"

  …
  OSstart();
  …
  OSisrInstall(29, &TIMtick);
  /* Set-up the count reload and enable SysTick interrupt */

  … /* More ISR setup */

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

At start-up, once `OSstart()` has been called, all `OS_N_INTERRUPTS` interrupt handler functions are set to a "do nothing" function, named `OSinvalidISR()`. If an interrupt function is attached to an interrupt number using the `OSisrInstall()` component before calling `OSstart()`, this attachment will be removed by `OSstart()`, so `OSisrInstall()` should never be used before `OSstart()` has ran. When an interrupt handler is removed, it is very important and necessary to first disable the interrupt source, then the handling function can be set back to `OSinvalidISR()`. This is shown in Table 3-3:

Table 3-3 Invalidating an ISR handler

```
#include "Abassi.h"

  …
  /* Disable the interrupt source */
  OSisrInstall(Number, &OSinvalidISR);
  …
```

When an application needs to disable/enable the interrupts, the RTOS supplied functions `OSdint()` and `OSeint()` should be used.

The interrupt number indicated by Generic Interrupt Controller (GIC) interrupt is acknowledged by the ISR dispatcher, but the dispatcher does not remove the request by a peripheral if the peripheral generates a level interrupt instead of a pulse.

3.2 Fast Interrupts

Fast interrupts are supported on this port as the FIQ interrupts. The ISR dispatcher is designed to only handle the IRQ interrupts.

3.3 Nested Interrupts

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

4 Stack Usage

The RTOS uses the tasks' stack for two purposes. When a task is blocked, or ready to run but not running, the stack holds the register context that was preserved when the task got blocked or preempted. Also, when an interrupt occurs, the register context of the running task must be preserved in order for the operations performed during the interrupt to not corrupt the contents of the registers used by the task when it got interrupted. For the Cortex-A9, the context save contents of a blocked or pre-empted task is different from the one used in an interrupt. The following table lists the number of bytes required by each type of context save operation:

Table 4-1 Context Save Stack Requirements

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

When sizing the stack to allocate to a task, there are three factors to take in account. The first factor is simply that every task in the application needs at least the area to preserve the task context when it is preempted or blocked. Second, one must also take into account the stack needs interrupts have. Finally, add to all this the stack required by the code implementing the task operation.

NOTE: The ARM Cortex-A9 processor needs alignment on 8 bytes for some instructions accessing memory. When stack memory is allocated, Abassi guarantees the alignment. This said, when sizing `OS_STATIC_STACK` or `OS_ALLOC_SIZE`, make sure to take in account that all allocation performed through these memory pools are by block size multiple of 8 bytes.

5 Search Set-up

The Abassi RTOS build option `OS_SEARCH_FAST` offers three different algorithms to quickly determine the next running task upon task blocking. The following table shows the measurements obtained for the number of CPU cycles required when a task at priority 0 is blocked, and the next running task is at the specified priority. The number of cycles includes everything, not just the search cycle count. The number of cycles was measured using Code Composer's Arm9e cycle accurate simulator. The second column is when `OS_SEARCH_FAST` is set to zero, meaning a simple array traversing. The third column, labeled Look-up, is when `OS_SEARCH_FAST` is set to 1, which uses an 8 bit look-up table. Finally, the last column is when `OS_SEARCH_FAST` is set to 5 (CCS/Cortex-A9 `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 High optimization (`-O3`) / Optimize for speed (`-mf5`) without debugging information. The RTOS build options were set to the minimum feature set, except for option `OS_PRIO_CHANGE` set to non-zero. The presence of this extra feature provokes a small mismatch between the result for a difference of priority of 1, with `OS_SEARCH_FAST` set to zero, and the latency results in Section 7.2.

When the build option `OS_SEARCH_ALGO` is set to a negative value, indicating to use a 2-dimensional linked list search technique instead of the search array, the number of CPU cycles is constant at 190 cycles.

Table 5-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	196	223	266
2	201	228	266
3	207	233	267
4	213	238	266
5	219	243	267
6	225	248	267
7	231	253	268
8	237	222	266
9	243	228	267
10	249	233	267
11	255	238	268
12	261	243	267
13	267	248	268
14	273	253	268
15	279	258	269
16	285	231	266
17	291	237	267
18	297	242	267
19	303	247	268
20	309	252	267
21	315	257	268
22	321	262	268
23	327	267	269
24	333	240	267

When `OS_SEARCH_FAST` is set to 0, each extra priority level to traverse requires exactly 6 CPU cycles. When `OS_SEARCH_FAST` is set to 1, each extra priority level to traverse requires exactly 5 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 25 cycles of CPU for the search, compared to setting `OS_SEARCH_FAST` to zero. But when the next ready to run priority is less than 8, 16, 24, … then there is around 5 cycles needed, but without the 8 times 5 cycle accumulation. Finally, the third option, when `OS_SEARCH_FAST` is set to 5, delivers a quasi-perfectly constant CPU usage, as the algorithm utilizes a successive approximation search technique (when the delta is 32 or more, the CPU cycle count is 273 +/- 2, for 64 or more, it is 282 +/- 2).

It is easy to observe, when looking at this table, that when an application has tasks spanning less than 7 or 8 priority levels, then the first option (`OS_SEARCH_FAST` set to 0) delivers the best performance. When an application has tasks spanning less than around 24 to 29[1] priority levels, then the second option (`OS_SEARCH_FAST` set to 1) delivers overall the best performance. The third option (`OS_SEARCH_FAST` set to 5) becomes interesting only when an application has tasks spanning a very large number of priority levels, e.g. 50 or 60.

Setting the build option `OS_SEARCH_ALGO` to a non-negative value minimizes the time needed to change the state of a task from blocked to ready to run, and not the time needed to find the next running task upon blocking/suspending of the running task. If the application needs are such that the critical real-time requirement is to get the next running task up and running as fast as possible, then set the build option `OS_SEARCH_ALGO` to a negative value.

[1] At 29 priority changes, the CPU cycle count is 265

6 Chip Support

No chip support is provided with the distribution.

7 Measurements

This section gives an overview of the memory requirements and the CPU latency encountered when the RTOS is used on the Arm9e and compiled with Code Composer Studio.

7.1 Memory

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

The code size numbers are expressed with “less than” as they have been rounded up to multiples of 25 for the “C” code. These numbers were obtained using the beta release of the RTOS and may change. One should interpret these numbers as the “very likely” numbers for the released version of the RTOS.

The code memory required by the RTOS includes the "C" code and assembly language code used by the RTOS. The code optimization settings of the compiler that were used for the memory measurements are:

1. Debugging model: Off[2]
2. Optimization level: 3[3]
3. Optimize for speed: 0
4. Instruction size 16
5. Target 5e

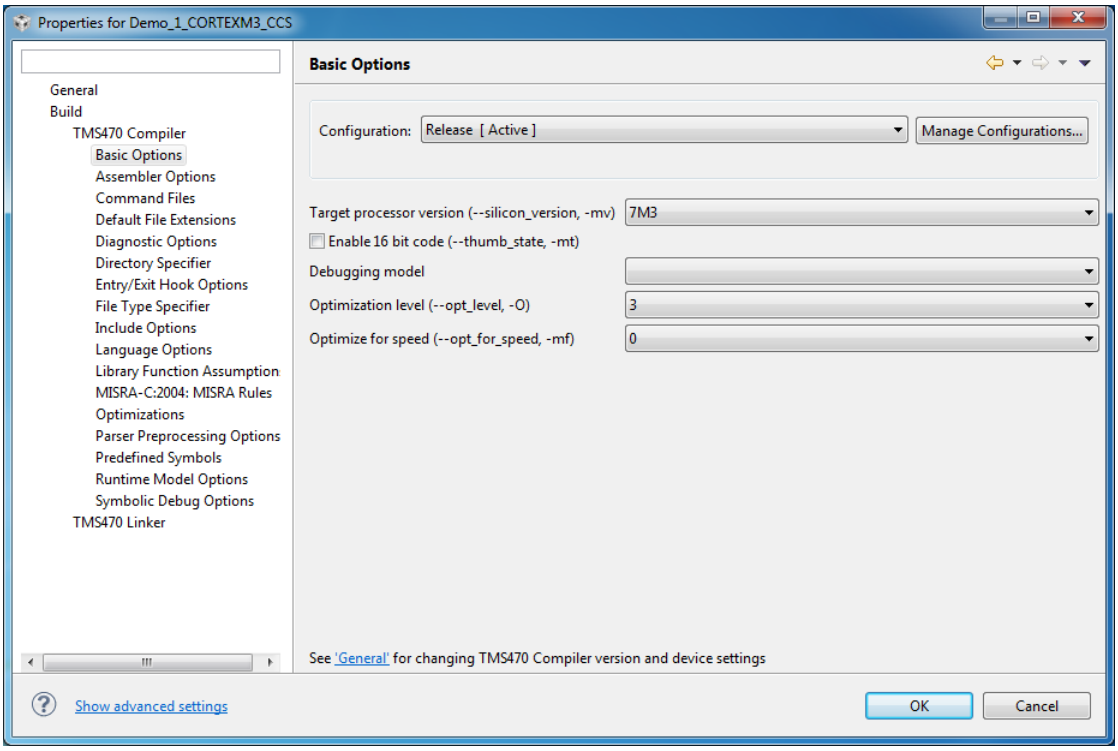

Figure 7-1 Memory Measurement Code Optimization Settings

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

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

Table 7-1 "C" Code Memory Usage

Description	Code Size
Minimal Build	< 775 bytes
+ Runtime service creation / static memory	< 1025 bytes
+ Multiple tasks at same priority	< 1125 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 1550 bytes
+ Timer & timeout + Timer call back + Round robin	< 2075 bytes
+ Events + Mailbox	< 2700 bytes
Full Feature Build (no names)	< 3225 bytes
Full Feature Build (no name / no runtime creation)	< 2825 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 3200 bytes

Table 7-2 Assembly Code Memory Usage

Description	Size
Assembly code size	804 bytes
VFPv3	+128 bytes
VFPv3D16	+116 bytes
Saturation Bit Enabled	+36 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.

7.2 Latency

Latency of operations has been measured using the cycle accurate simulator of Code Composer. The simulator was used instead of a real device as the development boards that were used during the development all used caches. The CPU required for any module becomes then dependent on the cache type, cache size and the type of code. Instead, using the simulator delivers an exact cycle count without the impact of a cache. Because the simulator was used, the measurements of latency involving interrupts were not performed. The code optimization settings that were used for the latency measurements are:

1. Debugging model: Off[4]
2. Optimization level: 3
3. Optimize for speed: 5
4. Instruction size 32
5. Target 5e
6. FPU None

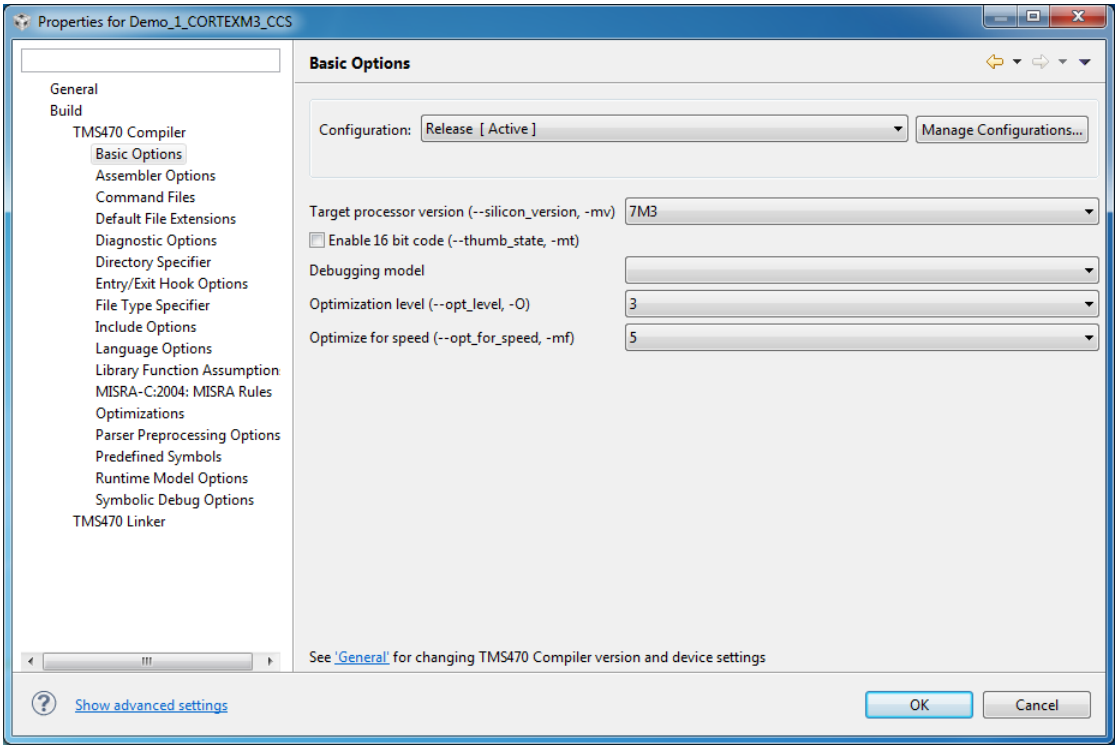

Figure 7-2 Latency Measurement Code Optimization Settings

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

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

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

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

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

Table 7-3 Measurement without Task Switch

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

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

Table 7-4 Measurement without Blocking

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

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

Table 7-5 Measurement with Task Switch

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

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

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

Table 7-6 Measurement with Task unblocking

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

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

Table 7-7 lists the results obtained, where the cycle count is measured using the number of CPU cycle measure with Code Composer cycle accurate simulator on the Cortex-A9.

The saturation bit (controlled through `OS_HANDLE_PSR_Q`) was not enabled in any of these tests.

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 7-7 Latency Measurements

Description	Minimal Features	Full Features
Semaphore posting no task switch	108 (108)	162 (162)
Semaphore waiting no blocking	118 (114)	175 (175)
Semaphore posting with task switch	160 (180)	270 (282)
Semaphore waiting with blocking	179 (172)	305 (295)
Semaphore posting in ISR with task switch	n/a (n/a)	n/a (n/a)
Event setting no task switch	n/a	159 (159)
Event getting no blocking	n/a	186 (186)
Event setting with task switch	n/a	278 (290)
Event getting with blocking	n/a	317 (307)
Event setting in ISR with task switch	n/a	n/a (n/a)
Mailbox writing no task switch	n/a	204 (204)
Mailbox reading no blocking	n/a	208 (208)
Mailbox writing with task switch	n/a	323 (335)
Mailbox reading with blocking	n/a	352 (342)
Mailbox writing in ISR with task switch	n/a	n/a (n/a)
Interrupt Latency	n/a	n/a
Interrupt overhead entering the kernel	n/a (n/a)	n/a (n/a)
Interrupt overhead NOT entering the kernel	n/a	n/a
Context switch	28	29

8 Appendix A: Build Options for Code Size

8.1 Case 0: Minimum build

Table 8-1: Case 0 build options

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

8.2 Case 1: + Runtime service creation / static memory

Table 8-2: Case 1 build options

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

8.3 Case 2: + Multiple tasks at same priority

Table 8-3: Case 2 build options

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

8.4 Case 3: + Priority change / Priority inheritance / FCFS / Task suspend

Table 8-4: Case 3 build options

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

8.5 Case 4: + Timer & timeout / Timer call back / Round robin

Table 8-5: Case 4 build options

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

8.6 Case 5: + Events / Mailboxes

Table 8-6: Case 5 build options

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

8.7 Case 6: Full feature Build (no names)

Table 8-7: Case 6 build options

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

8.8 Case 7: Full feature Build (no names / no runtime creation)

Table 8-8: Case 7 build options

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

8.9 Case 8: Full build adding the optional timer services

Table 8-9: Case 8 build options

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