CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

MSP430 – GCC

Disclaimer

Table of Contents

List of Tables

1 Introduction

This document details the port of the Abassi RTOS to the MSP430 processor. The software suite used for this specific port is the "mspgcc" for MSP430; the version used for the port and all tests is Version 4.5.3.

NOTE: mspgcc does not yet support the MSP430 extended architecture, commonly known as MSP430X. This port will work on such devices, but the code size remains limited to less than 64Kbytes.

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_MSP430_GCC.s`	RTOS assembly file for the MSP430 to use with the GCC
`Demo_2X_MSP430_GCC.c`	Demo code that runs on the Olimex MSP-5438STK evaluation board using the LCD
`Demo_3_MSP430_GCC.c`	Demo code that runs on the Olimex MSP430-P1611 evaluation board using the serial port
`Demo_3X_MSP430_GCC.c`	Demo code that runs on the Olimex MSP-5438STK evaluation board using the serial port
`Demo_4X_MSP430_GCC.c`	Demo code that runs on the Olimex MSP-5438STK evaluation board using joystick and the serial port
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

None.

2 Target Set-up

Very little is needed to configure GCC for the MSP430 to use the Abassi RTOS in an application. All there is to do is to add the files `Abassi.c` and `Abassi_MSP430_GCC.s` in the source files of the application project, and make sure the configuration settings (described in the following subsections) in the file `Abassi_MSP430_GCC.s` 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`.

NOTE: The file `Abassi_MSP430_GCC.s` is the start-up file and also holds the interrupt vector table; the standard start-up in GCC should not be used. This means it is necessary to specify the option `-nostartfile` on the linker command line.

NOTE: The GCC libraries are not multithread-safe without the use of the `-pthread` command line option. However, this option is not available for GCC built to generate MSP430 code. This means calls to libraries functions that are non- multithread-safe should be protected by a mutex. These functions are typically the dynamic memory management functions, some form of the `printf` / `scanf` functions, file I/O, etc. If the GCC toolset used utilizes the `newlib` libraries from Red Hat, you need to attach Abassi mutexes to the *`x`*`_lock()` and *`x`*`_unlock()` multithread protections functions.

NOTE: If a hardware multiplier is available on the target device, the use of the multiplier must always be protected by disabling/enabling the interrupts. This is true even when the multiplier is not accessed inside an interrupt. The reason is that one or many task switches may be triggered by an interrupt. So, if the preemption of a task occurs when it is in the process of using the multiplier, and a newly running task also uses the multiplier, the multiplication result for the preempted task will be erroneous.

There is no possibility for the RTOS to protect the multiplier as the operation to perform is set when the first operand is written to the desired "operation" register; there is no way to know which of the "operation" registers was written last, specifying the type of operation, therefore the RTOS cannot protect the multiplier registers.

By default the GCC compiler generates code that protects the multiplier when using it. This means the compiler option `-mnoint-hwmul` should not be used, as this option removes the protection around the multiplier use.

2.1 Interrupt Stack Set-up

It is possible, and highly recommended, to use a hybrid stack when nested interrupts occur in an application. Using this hybrid stack, specially dedicated to the interrupts, removes the need to allocate extra room to the stack of every task in the application to handle the interrupt nesting. This feature is controlled by the value set by the definition `OS_ISR_STACK`, located around line 30 in the file `Abassi_MSP430_GCC.s`. To disable this feature, set the definition of `OS_ISR_STACK` to a value of zero. To enable it, and specify the interrupt stack size, set the definition of `OS_ISR_STACK` to the desired size in bytes (see Section 4 for information on stack sizing). As supplied in the distribution, the hybrid stack feature is enabled, and a stack size of 128 bytes is allocated; this is shown in the following table:

Table 2-1 Interrupt Stack enabled

```
.equ  OS_ISR_STACK, 128      ; If using a dedicated stack for the ISRs
                             ; 0 if not used, otherwise size of stack in bytes
```

Table 2-2 Interrupt Stack disabled

```
.equ  OS_ISR_STACK, 0        ; If using a dedicated stack for the ISRs
                             ; 0 if not used, otherwise size of stack in bytes
```

2.2 Interrupt Nesting

The normal operation of the interrupt controller on the MSP430 family is to allow a single interrupt to operate anytime. This means when the processor is servicing an interrupt, any new interrupts, even if their priority is higher than the serviced interrupt level, remain pending until the processor finishes servicing the current interrupt. The interrupt dispatcher allows the nesting of interrupts; this means an interrupt of any priority can interrupt the processing of an interrupt currently being handled. Nested interrupts are enabled by setting both the build option `OS_NESTED_INTS` and the token `OS_NESTED_INTS` in the `Abassi_MSP430_GCC.s` file, around line 30, to a non-zero value, as shown in the following table:

Table 2-3 Nested Interrupts enabled

```
.equ  OS_NESTED_INTS, 1      ; To allow interrupt nesting, set to non zero
                             ; To not allow interrupt nesting, set to zero
```

Interrupt nesting is disabled (in other words, the interrupts operate exactly as the MSP430 interrupt controller operates) by setting both the build option `OS_NESTED_INTS` and the token `OS_NESTED_INTS` to a zero value, as shown in the following table:

Table 2-4 Nested Interrupts disabled

```
.equ  OS_NESTED_INTS, 0      ; To allow interrupt nesting, set to non zero
                             ; To not allow interrupt nesting, set to zero
```

NOTE: The build option `OS_NESTED_INTS` must be set to a non-zero value when the token `OS_NESTED_INTS` in the file `Abassi_MSP430_GCC.s` is set to a non-zero value. If the token `OS_NESTED_INTS` in the file `Abassi_MSP430_GCC.s` is set to a zero value, and the build option `OS_NESTED_INTS` is non-zero, the application will properly operate, but with a tiny bit less real-time efficiency when kernel requests are performed during an interrupt.

2.3 Oscillator control bits propagation

In the MSP430 status register, there are 3 bits that control the oscillators on the device. If any of these bits is modified after the interrupts are enabled in the application, the change must be propagated across all tasks and interrupts. This feature is controlled by the value set in the definition `OS_HANDLE_OSC`, located around line 30 in the file `Abassi_MSP430_GCC.s`. To disable this feature, set the definition of the token `OS_HANDLE_OSC` to a value of zero. To enable it, set the definition of `OS_HANDLE_OSC` to a non-zero value. As supplied in the distribution, the oscillator control bits propagation is disabled; this is shown in the following table:

Table 2-5 Oscillator bits not propagated

```
.equ  OS_HANDLE_OSC, 0        ; Set to non-zero to propagate oscillator control bits
                              ; in SR from ISR to the background / tasks
```

Table 2-6 Oscillator bits propagated

```
.equ  OS_HANDLE_OSC,1         ; Set to non-zero to propagate oscillator control bits
                              ; in SR from ISR to the background / tasks
```

2.4 Interrupt vector table

There are three different flavors for the MSP430 interrupt table: some devices have a table capable of handling up to 16 interrupts sources, others have room for 32 interrupt sources, and others can deal with 64 interrupt sources. Abassi can support all three, but it must be configured to the correct size in order to optimize the code footprint and properly map the interrupt priority to the interrupt vector table entry. The information must be set in the file `Abassi_MSP430_GCC.s` around line 30; the token `OS_INT_VECT_SIZE` must be set to 16, for a 16 entry table, to 32, for a 32 entry table, or set to 64 for a 64 entry table:

Table 2-7 Interrupt vector with 16 entries

```
.equ  OS_INT_VECT_SIZE, 16    ; Number of interrupts in the interrupt vector table
                              ; Should be either 16 / 32 /64
```

Table 2-8 Interrupt vector with 32 entries

```
.equ  OS_INT_VECT_SIZE, 32    ; Number of interrupts in the interrupt vector table
                              ; Should be either 16 / 32 /64
```

Table 2-9 Interrupt vector with 64 entries

```
.equ  OS_INT_VECT_SIZE, 64    ; Number of interrupts in the interrupt vector table
                              ; Should be either 16 / 32 /64
```

3 Interrupts

The Abassi RTOS needs to be aware when kernel requests are performed inside or outside an interrupt context. Normally, when coding with the GCC, an interrupt function is specified with the `interrupt` directive. But for all interrupt sources (except for the reset), 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 15 sources of interrupts, as specified by the build option `OS_N_INTERRUPTS`, defined in the file `Abassi.h`, and the token `OS_INT_VECT_SIZE`, in the file `Abassi_MSP430_GCC.s`. If the target device uses a 32 or 64 entries interrupt vector, consult Section 2.4 to understand how to set Abassi to support the larger interrupt vector table.

3.1 Interrupt Handling

3.1.1 Interrupt Installer

Attaching a function to an interrupt is quite straightforward. All there is to do is use the RTOS component `OSisrInstall()` to specify the interrupt priority and the function to be attached to that interrupt priority. For example, Table 3-1 shows the code required to attach the `TIMERA` interrupt (on a MSP430F1611) to the RTOS timer tick handler (`TIMtick`):

Table 3-1 Attaching a Function to an Interrupt

```
#include "Abassi.h"

  …
  OSstart();
  …
  OSisrInstall(54, &TIMtick);
  /* Set-up the count reload and enable SysTick interrupt */

  … /* More ISR setup */

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

Alternatively, instead of using a hard coded number, the standard definition supplied by the file `msp430.h` can be used. These definitions are set to the vector table index, specified in bytes; since `OSisrIntall()` uses the priority value, these definitions must be divided by 2, as shown in the Table 3-2:

Table 3-2 Attaching a Function to an Interrupt

```
#include "Abassi.h"
#include <msp430.h>

  …
  OSstart();
  …
  OSisrInstall(TIMERA0_VECTOR/2, &TIMtick);
  /* Set-up the count reload and enable SysTick interrupt */

  … /* More ISR setup */

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

NOTE: The function to attach to an interrupt is a regular function, not one declared with the GCC specific `interrupt` prefix statement.

NOTE: `OSisrInstall()` uses the interrupt priority number. As an example, the non-maskable interrupt has a priority of 14 when the device uses a table of 16 interrupt, and a value of 30 when the device uses a table of 32 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.

3.2 Unused Interrupts

The assembly file `Abassi_MSP430_GCC.s`, as supplied in the distribution, includes the prologue code for the interrupt dispatcher for all sources of interrupts. If the code memory space is becoming a bit short, removing the prologue for unused interrupts will help recover memory from that dead code.

Removing the interrupt dispatcher prologue for an unused interrupt is a three-step process. First, the unused interrupt vector must be replaced in the interrupt vector table. This table is located at around line 150, at the label `VectTbl`, and each interrupt entry is defined as shown in the following:

Table 3-4 Entry in the interrupt vector table

```
    ISR_VECTOR   XX                     ; Priority XX interrupt
```

The desired table entry must be attached to a do-nothing interrupt handler; it is preferable to attach a do-nothing interrupt handler in case of spurious interrupts. To attach the do-nothing interrupt handler, replace the desired vector table entry by the following:

Table 3-5 Unused interrupt vector table

```
    .word   INT_NO_handler              ; Priority XX interrupt
```

The second step is to create the do-nothing interrupt handler. This step only need to be performed once, as the same do-nothing handler should be re-used for all unused interrupts. The do-nothing interrupt handler code must be located in the `ISR_CODE` section. Therefore, insert the following code right after the definition of the `ISR_PROLOGUE` macro, right before the `ISR_HANDLR 0` statement; this should be around line 230 in the file:

Table 3-6 Do-nothing interrupt handler

```
INT_NO_handler:                         ; Entry point of the do-nothing ISR handler
    reti                                ; Return from the interrupt
```

The last step is to remove the unused interrupt dispatcher prologue code. Each interrupt has an interrupt dispatcher prologue, where the prologue is always defined as follows:

Table 3-7 Interrupt dispatcher prologue

```
    ISR_HANDLER   XX
```

Deleting the `ISR_HANDLER` line for the unused interrupt will remove the prologue code.

3.3 Fast Interrupts

Fast interrupts are supported on this port. A fast interrupt is an interrupt that never uses any component from Abassi and as the name says, is desired to operate as fast as possible. To set-up a fast interrupt, all there is to do is to set the address of the interrupt function in the corresponding entry in the interrupt vector table that is used by the MSP430 processor. The beginning of the interrupt vector table is located in the file `Abassi_MSP430_GCC.s` around line 150, at the label `VectTbl`. For example, on a MSP430F1611 device, `TIMERA` is set to the priority 6. This is the entry in the table for `TIMERA` in the distribution file:

Table 3-8 MSP430F1611 TIMERA Regular Interrupt

```
    ISR_VECTOR 6                        ; Priority  6 interrupt
```

To attach a fast interrupt handler to the `TIMERA`, assuming the name of the interrupt function to attach is `TIMERA_handler()`, replace the previous line with that shown in the Table 3-9:

Table 3-9 MSP430F1611 TIMERA Fast Interrupt

```
    .word   TIMERA_handler              ; Priority  6 interrupt
```

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

NOTE: If an Abassi component is used inside a fast interrupt, the application will misbehave.

NOTE: Fast interrupt handlers must use the GCC keyword `interrupt`, unless `reti` is used.

Even if the hybrid interrupt stack feature is enabled (see Section 2.1), fast interrupts will not use that stack. This translates into the need to reserve room on all task stacks for the possible nesting of fast interrupts. To make the fast interrupts use a hybrid interrupt stack, a prologue and epilogue must be used around the call to the interrupt handler. The prologue and epilogue code to add is identical to what is done in the regular interrupt dispatcher.

If the extra stack room required on all tasks is a burden on the data space, another way to reduce the impact on the task stacks is to dedicate individual stacks to each one of the fast interrupts. Reusing the example of the `TIMERA` on a MSP430F1611 device, this would look something like:

Table 3-10 Fast Interrupt with Dedicated Stack

```
    …
    …

    .word     TIMERA_preHandler               ; Priority  6 interrupt

    …
    …

    .section .text, "ax", @progbits

    .global   TIMERA_handler

TIMERA_preHandler:
    mov.w   r1, #(TIMERA_stack-2)           ; Memo current sp on the hybrid stack
    mov.w   #(TIMERA_stack-2), r1           ; Set sp to the new stack
    push.w  r15                             ; Context save on the hybrid stack
    push.w  r14
    push.w  r13
    push.w  r12

    call    TIMERA_handler                  ; Enter the interrupt handler

    pop.w   r12                             ; Context restore
    pop.w   r13
    pop.w   r14
    pop.w   r15
    pop.w   r1                              ; Recover original sp
    reti                                    ; Exit from the interrupt

    …
    …

    .section  .bss
    .balign   2
    .space   2*((TIMERA_stack_size+1)/2)     ; Room for the fast interrupt stack
TIMERA_stack:

    …
```

The same code, with unique labels, must be repeated for each of the fast interrupts. As the use of the hybrid stack creates the prologue-epilogue for the interrupt context, the function called must be a regular "C" function, not one declared with the `interrupt` directive. If the GIE bit (global interrupt enable) in the status register is not set in the interrupt function, and the nesting of interrupts is not allowed (Section 2.2), then the same hybrid stack memory can be re-used, as, by default, the MSP430 interrupt controller only allows the servicing of a single interrupt at any time.

3.4 Nested Interrupts

The interrupt dispatcher allows the nesting of interrupts; nested interrupt means an interrupt of any priority will interrupt the processing of an interrupt currently being serviced. Refer to section 2.2 for information on how to enable or disable interrupt nesting.

The Abassi RTOS kernel never disables interrupts, but there are a few very small regions within the interrupt dispatcher where interrupts are temporarily disabled when nesting is enabled (a total of between 10 to 20 instructions).

The kernel is never entered as long as interrupt nesting is occurring. In all interrupt functions, when a RTOS component that needs to access some kernel functionality is used, the request(s) is/are put in a queue. Only once the interrupt nesting is over (i.e. when only a single interrupt context remains) is the kernel entered at the end of the interrupt, when the queue contains one or more requests, and when the kernel is not already active. This means that only the interrupt handler function operates in an interrupt context, and only the time the interrupt function is using the CPU are other interrupts of equal or lower level blocked by the interrupt controller.

4 Stack Usage

The RTOS uses the tasks' stack for two purposes. When a task is blocked or ready to run but not running, the stack holds the register context that was preserved when the task got blocked or preempted. Also, when an interrupt occurs, the register context of the running task must be preserved in order for the operations performed during the interrupt to not corrupt the contents of the registers used by the task when it got interrupted. For the MSP430, 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	`16 bytes`
Interrupt dispatcher context save	`14 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 take into account how many levels of nested interrupts exist in the application. As a worst case, all levels of interrupts may occur and becoming fully nested. So, if N levels of interrupts are used in the application, provision should be made to hold N times the size of an ISR context save on each task stack, plus any added stack used by the interrupt handler functions. Finally, add to all this the stack required by the code implementing the task operation.

NOTE: The MSP430 processor needs alignment on 2 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 2 bytes.

If the hybrid interrupt stack (see Section 2.1) is enabled, then the above description changes: it is only necessary to reserve room on task stacks for a single interrupt context save and not the worst-case nesting. With the hybrid stack enabled, the second, third, and so on interrupts use the stack dedicated to the interrupts. The hybrid stack is enabled when the `ISR_STACK` token in the file `Abassi_MSP430_GCC.s` is set to a non-zero value (Section 2.1).

5 Search Set-up

The Abassi RTOS build option `OS_SEARCH_FAST` offers four 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 `TIMERA` peripheral, which was set to increment the counter once every CPU cycle. 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 4 (MSP430 `int` are 16 bits, so 2^4), meaning a 16 bit look-up table, further searched through successive approximation. The compiler optimization for this measurement was set to `-O3`. 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 is constant at 293 cycles.

Table 5-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	292	337	436
2	301	343	441
3	307	349	451
4	313	355	451
5	319	361	461
6	325	367	466
7	331	373	476
8	337	337	471
9	343	346	481
10	349	352	486
11	355	358	496
12	361	364	496
13	367	370	506
14	373	376	511
15	379	382	521
16	385	346	446
17	391	355	456
18	397	361	461
19	403	367	471
20	409	373	471
21	415	379	481
22	421	385	486
23	427	391	496
24	433	355	491

The third option, when `OS_SEARCH_FAST` is set to 4, never achieves a lower CPU usage than when `OS_SEARCH_FAST` is set to zero or 1. This is understandable, as the MSP430 does not possess a barrel shifter for variable shift. When `OS_SEARCH_FAST` is set to zero, 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 also requires exactly 6 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 an extra 42 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 an extra 9 cycles needed, but without the 8 times 16 cycles accumulation.

What does this mean? Using 20 or 30 tasks on the MSP430 may be an exception due to the limited memory space, so one could assume the number of tasks will remain small. If that is the case, then `OS_SEARCH_FAST` should be set to 0. If an application is created with 20 or 30 tasks, then setting `OS_SEARCH_FAST` to 1 may be better choice.

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, but 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.

6 Chip Support

No chip support is provided with the distribution code because the MSP430Ware software library is available from Texas Instruments and it includes a high level API for all the peripherals available on the MSP430 devices. Even though the primary target for the MSP430Ware software library is with Code Composer Studio as the GUI, a standalone version is also available.

7 Measurements

This section gives an overview of the memory requirements and the CPU latency encountered when the RTOS is used on the MSP430 and compiled with GCC. The CPU cycles are exactly the CPU clock cycles, not a conversion from a duration measured on an oscilloscope then converted to a number of cycles.

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 run-time 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 for the memory measurements used the option `-Os`.

Table 7-1 "C" Code Memory Usage

Description	Code Size
Minimal Build	< 725 bytes
+ Runtime service creation / static memory	< 1025 bytes
+ Multiple tasks at same priority	< 1125 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 1650 bytes
+ Timer & timeout + Timer call back + Round robin	< 2225 bytes
+ Events + Mailbox	< 3025 bytes
Full Feature Build (no names)	< 3600 bytes
Full Feature Build (no names / no runtime creation)	< 3125 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 3600 bytes

Table 7-2 Assembly Code Memory Usage

Description	Size
Assembly code size	158 bytes
Vector Table (per interrupt)	2 bytes
Interrupt prologue (per interrupt)	8 bytes
Start-up code (replacement for `crt0.s`)	66 bytes
Hybrid Stack Enabled	+10 bytes
Nested interrupts Enabled	+ 8 bytes
Oscillator bits preservation Enabled	+14 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 Code Time Technologies web site.

7.2 Latency

Latency of operations has been measured on an Olimex Evaluation board populated with an 8 MHz MSP430F1611 device. All measurements have been performed on the real platform, using the timer peripheral `TIMERA` set-up to be clocked at the same rate as the CPU. This means the interrupt latency measurements had to be instrumented to read the `TIMERA` counter value. This instrumentation can add up to 5 or 6 cycles to the measurements. The code optimization settings for the latency measurements used the option `-Os`.

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 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 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 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
    …
}
```

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 7-7 lists the results obtained, where the cycle count is measured using the `TIMERA` peripheral on the MSP430. This timer increments its counter by 1 at every CPU cycle. As was the case for the memory measurements, these numbers were obtained with a beta release of the RTOS. It is possible the released version of the RTOS may have slightly different numbers.

The interrupt latency is the number of cycles elapsed when the interrupt trigger occurred and the ISR function handler is entered. This includes the number of cycles used by the processor to set-up the interrupt stack and branch to the address specified in the interrupt vector table. For this measurement, the MSP30 `TIMERA` is used to trigger the interrupt and measure the elapsed time.

The interrupt overhead without entering the kernel is the measurement of the number of CPU cycles used between the entry point in the interrupt vector and the return from interrupt, with a "do nothing" function in the `OSisrInstall()`. The interrupt overhead when entering the kernel is calculated using the results from the third and fifth tests. Finally, the OS context switch is the measurement of the number of CPU cycles it takes to perform a task switch, without involving the wrap-around code of the synchronization component.

The hybrid interrupt stack feature was not enabled, neither was the oscillator bit preservation, nor the interrupt nesting, 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	170 (170)	258 (269)
Semaphore waiting no blocking	169 (169)	281 (295)
Semaphore posting with task switch	286 (286)	457 (486)
Semaphore waiting with blocking	273 (273)	498 (501)
Semaphore posting in ISR with task switch	473 (473)	658 (696)
Event setting no task switch	n/a	247 (259)
Event getting no blocking	n/a	306 (317)
Event setting with task switch	n/a	486 (516)
Event getting with blocking	n/a	528 (534)
Event setting in ISR with task switch	n/a	687 (725)
Mailbox writing no task switch	n/a	328 (341)
Mailbox reading no blocking	n/a	329 (341)
Mailbox writing with task switch	n/a	516 (547)
Mailbox reading with blocking	n/a	574 (578)
Mailbox writing in ISR with task switch	n/a	713 (752)
Interrupt Latency	31	31
Interrupt overhead entering the kernel	187 (187)	201 (210)
Interrupt overhead NOT entering the kernel	59	59
Context switch	68	68

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_US         0    /* !=0 enables timer & specifies the period in uS    */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module            */
#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_US         0    /* !=0 enables timer & specifies the period in uS    */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module            */
#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_US         0    /* !=0 enables timer & specifies the period in uS    */
#define OS_TIMER_SRV        0    /* !=0 includes the timer services module            */
#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          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     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  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    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                          */
```