CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

ATmega128 – GCC

Table of Contents

List of Figures

List of Tables

1 Introduction

This document details the port of the Abassi RTOS to the ATmega128 processor from Atmel. The software suite used for this specific port is the Atmel AVR Studio 5; the specific version used for the port and all tests is Version 5.0.1223, which bundles GCC version 4.5.1.

NOTE: This document does not cover the port for AVR devices other than ATmega128. Different documents describe the port for non-ATmega128 AVR devices.

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_ATmega128_GCC.s`	RTOS assembly file for the ATmega128 to use with the AVR-GCC toolchain
`Demo_3_AVRMT128_GCC.c`	Demo code that runs on the Olimex AVR-MT-128 evaluation board using the serial port
`Demo_3_AVRMT128_GCC.mak`	Makefile for Demo #3
`Demo_4_AVRMT128_GCC.c`	Demo code that runs on the Olimex AVR-MT-128 evaluation board using the LCD
`Demo_4_AVRMT128_GCC.mak`	Makefiel for Demo #4
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

None

2 Target Set-up

Very little is needed to configure the Atmel AVR 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_ATmega128_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_ATmega128_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`.

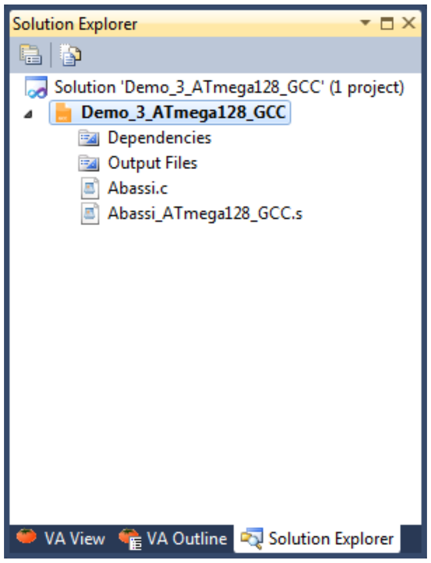


Figure 2-1 Project File List

NOTE: The GCC libraries are not multithread-safe without the use of the `-pthread` command line option. However, this option is not available in the avr-gcc bundled with Atmel AVR Studio 5. This means calls to libraries functions that are non- multithread-safe should be protected by a mutex, ideally the `G_OSmutex` 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.

2.1 Interrupt Stack Set-up

It is possible, and highly recommended to use a hybrid stack when nested interrupts are enabled 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 `ISR_STACK`, located around line 25 in the file `Abassi_ATmega128_GCC.s`. To disable this feature, set the definition of `ISR_STACK` to a value of zero. To enable it, and specify the interrupt data stack size, set the definition of `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 data stack size of 64 bytes is allocated; this is shown in the following table:

Table 2-1 Interrupt Stack enabled

```
    .equ  ISR_STACK, 64              ; 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  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 ATmega128 devices is to only allow a single interrupt to operate at any time. 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` in the `Abassi.h` file and the token `NESTED_INTS` in the `Abassi_ATmega128_GCC.s` file, around line 30, to a non-zero value, as shown in the following table:

Table 2-3 Nested Interrupts enabled

```
    .equ 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 ATmega128 interrupt controller operates) by setting both the build option `OS_NESTED_INTS` in the `Abassi.h` file and the token `NESTED_INTS` in the `Abassi_ATmega128_GCC.s` file to a zero value, as shown in the following table:

Table 2-4 Nested Interrupts disabled

```
    .equ NESTED_INTS, 1         ; 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 `NESTED_INTS` in the file `Abassi_ATmega128_GCC.s` is set to a non-zero value. If the token `NESTED_INTS` in the file `Abassi_ATmega128_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.

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 Atmel AVR Studio, an interrupt function is specified with the `ISR()` macro. 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.

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 vector index (the interrupt vector index is the interrupt vector number minus one). For example, Table 3-1 shows the code required to attach the `TIMER/COUNTER1` overflow interrupt (on a ATMEGA128-16AU) to the RTOS timer tick handler (`TIMtick`):

Table 3-1 Attaching a Function to an Interrupt

```
#include “Abassi.h”

   …
   OSstart();
   …
   OSisrInstall(14, &TIMtick);
   /* Set-up the count reload and enable SysTick interrupt */

   … /* More ISR setup */

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

The standard interrupt vector definition supplied by the file `avr/io.h` (`TIMER1_OVF_vect` in the above example) cannot be used, as they are macro definitions generating the wrapping code for interrupt handlers.

NOTE: The function to attach to an interrupt is a regular function, not one declared with the Embedded Atmel AVR Studio specific `ISR()` macro.

NOTE: `OSisrInstall()` uses the interrupt priority index. As an example, the reset interrupt has the index of 0.

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 3-2:

Table 3-2 Invalidating an ISR handler

```
#include "Abassi.h"

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

Another example with a real interrupt initialization function is shown in Section **Error! Reference source not found.**.

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_ATmega128_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 prologues for unused interrupts will help recover a bit of code memory from that dead code.

Removing the interrupt dispatcher prologue for an unused interrupt is a one-step process. All there is to do is to remove the unused interrupt vector prologue. The 34 prologues are located at around line 230, and each one is defined as shown in the following:

Table 3-3 Interrupt Prologue

```
    ISR_PROLOGUE  XX
```

Commenting out the desired prologue removes the code:

Table 3-4 Interrupt Prologue Removal

```
;    ISR_PROLOGUE  XX
```

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 remove the corresponding interrupt prologue as explained in the previous section and use the Atmel AVR Studio interrupt macro `ISR()` to create the interrupt handler. Two following tables show how attach a fast interrupt to the `COUNTER/TIMER3` overflow (interrupt index 29) of the ATmega128:

Table 3-5 ATMEGA128-16AU TIMER/COUNTER3 Prologue removal

```
;     ISR_PROLOGUE  29
```

Table 3-6 ATMEGA128-16AU TIMER/CONTER3 Fast Interrupt

```
#include <avr/io.h>
#include <avr/interrupt.h>

…

ISR(TIMER3_OVF_vect)
{
  …             /* Code for the interrupt handler */
}
```

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

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. If the fast interrupt is coded in "C", it is not possible to make this fast interrupt use a hybrid stack. If the interrupt handler is coded in assembler, then it can be done, as shown in the Table 3-7. The example re-uses the `TIMER/COUNTER3` overflow interrupt:

Table 3-7 Fast Interrupt with Dedicated Stack

```
    …
    …

;   ISR_PROLOGUE  29                        ; Remove the ISR prologue of Timer 3

    …
    …

    .section .text, "ax"
    .balign  2
    .type __vector_29, @function            ; Overload the library interrupt vector
    .global __vector_29

__vector_29:

    push    r31                             ; 2 registers are needed to deal with
    push    r30                             ; the hybrid stack

    in      r31, SPH                        ; Save current stack pointer on the
    sts     (T3stack-1), r31                ; hybrid stack
    in      r31, SPL
    sts     (T3stack-2), r31

    ldi     r31, lo8(T3stack-3)             ; Set stack pointer to hybrid stack
    out     SPL, r31
    ldi     r31, hi8(T3stack-3)
    out     SPH, r31

                                            ; *** ADD extra register save here

    call    My_Timer3_Int                   ; OR insert ISR handler code here

                                            ; *** ADD extra register restore here

    pop     r31                             ; Back to the original stack pointer
    pop     r30
    out     SPL, r31
    out     SPH, r30

    pop     r30
    pop     r31

    reti


    …

    .section .bss
    .space    T3_STACK_SIZE                 ; Reserve room in bss for the hybrid stack
T3stack:
```

The same code, with unique labels, must be repeated for each of the fast interrupts. If the interrupt handler modifies more registers than `r30` & `r31` (don’t forget the status register), every modified register must be preserved after the set-up of the hybrid stack and before the tear down of the hybrid stack. Also, if the interrupts are re-enabled to allow interrupt nesting, the interrupts must remain disabled until the hybrid stack is completely set-up and must be disabled before the teardown of the hybrid stack, as the manipulation of the stack registers (`SPL` and `SPH`) creates a critical region.

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[1], 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.

[1] The way GCC uses the stack, in most of the code generated by GCC, there are regions where interrupts are disabled for 3 instructions (3 CPU cycles); the same applies for Abassi context switch.

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 ATmega128, 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	`19 bytes`
Interrupt context save (no Hybrid stack)	`17 bytes`
Interrupt context save (Hybrid stack)	`17 bytes`

The numbers for the interrupt dispatcher context save include the 2 bytes the processor pushes on the stack when it enters the interrupt servicing.

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.

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_ATmega128_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 `TIMER/COUNTER3` 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 (ATmega128 `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 Level High / Speed optimization. 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 433 cycles.

Table 5-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	426	488	622
2	439	496	627
3	447	504	638
4	455	512	637
5	463	520	648
6	471	528	653
7	479	536	664
8	487	492	657
9	495	505	668
10	503	513	673
11	511	521	684
12	519	529	683
13	527	537	694
14	535	545	699
15	543	553	710
16	551	503	636
17	559	516	647
18	567	524	652
19	575	532	663
20	583	540	662
21	591	548	673
22	599	556	678
23	607	564	689
24	615	514	682

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 ATMEGA128 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 8 CPU cycles. When `OS_SEARCH_FAST` is set to 1, each extra priority level to traverse also requires exactly 8 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 57 extra cycles of CPU for the search compared to setting `OS_SEARCH_FAST` to zero. But when the next ready to run priority is less than 8, 16, 24, … then there are some extra cycles needed, but without the 8 times 8 cycles accumulation.

What does this mean? Using more that 6 to 8 tasks on the ATMEGA128 may be an exception due to the limited data 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 more than 6 to 8 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.

7 Measurements

This section gives an overview of the memory requirements and the CPU latency encountered when the RTOS is used on the ATMEGA128 and compiled with the Atmel AVR Studio 5. 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.

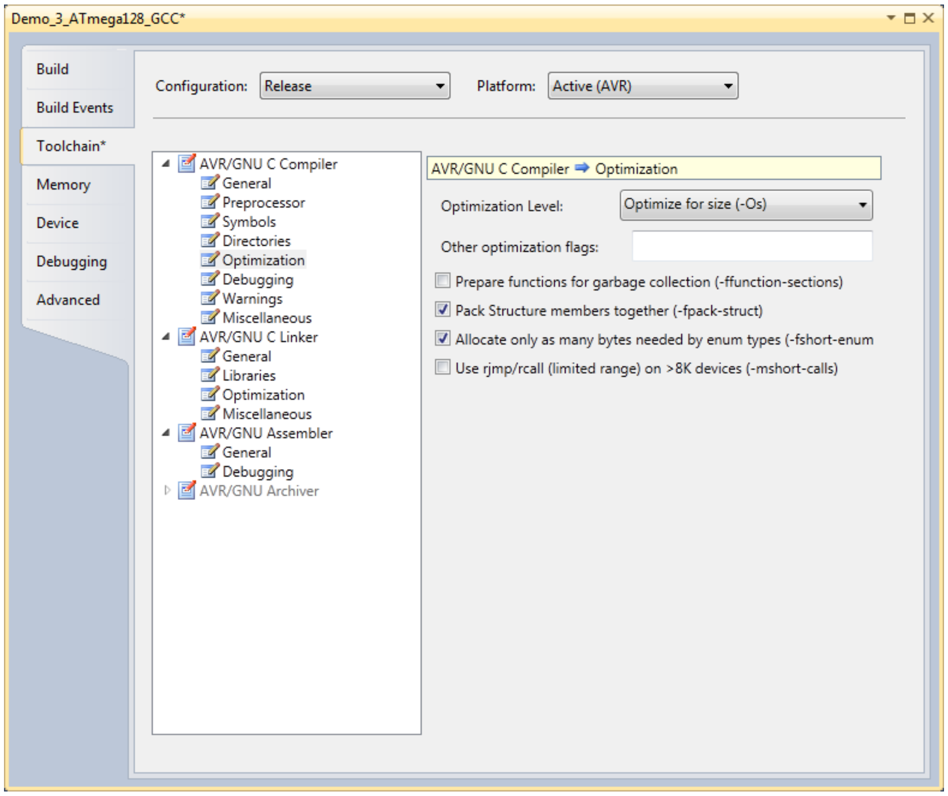
7.1 Memory

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 used for the memory measurements are:

1.	Optimization Level:	Optimize for size (-Os)
2.	Debug Level:	None



Configuration: Release
Platform: Active (AVR)

Optimization Level: Optimize for size (-Os)
Other optimization flags:

☐ Prepare functions for garbage collection (-ffunction-sections)
☑ Pack Structure members together (-fpack-struct)
☑ Allocate only as many bytes needed by enum types (-fshort-enum
☐ Use rjmp/rcall (limited range) on >8K devices (-mshort-calls)

Figure 7-1 Memory Measurement Code Optimization Settings

Table 7-1 "C" Code Memory Usage

Description	Code Size
Minimal Build	< 1100 bytes
+ Runtime service creation / static memory	< 1475 bytes
+ Multiple tasks at same priority	< 1800 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 3000 bytes
+ Timer & timeout + Timer call back + Round robin	< 3750 bytes
+ Events + Mailbox	< 5175 bytes
Full Feature Build (no names)	< 6275 bytes
Full Feature Build (no names / no runtime creation)	< 5450 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 5850 bytes

Table 7-2 Assembly Code Memory Usage

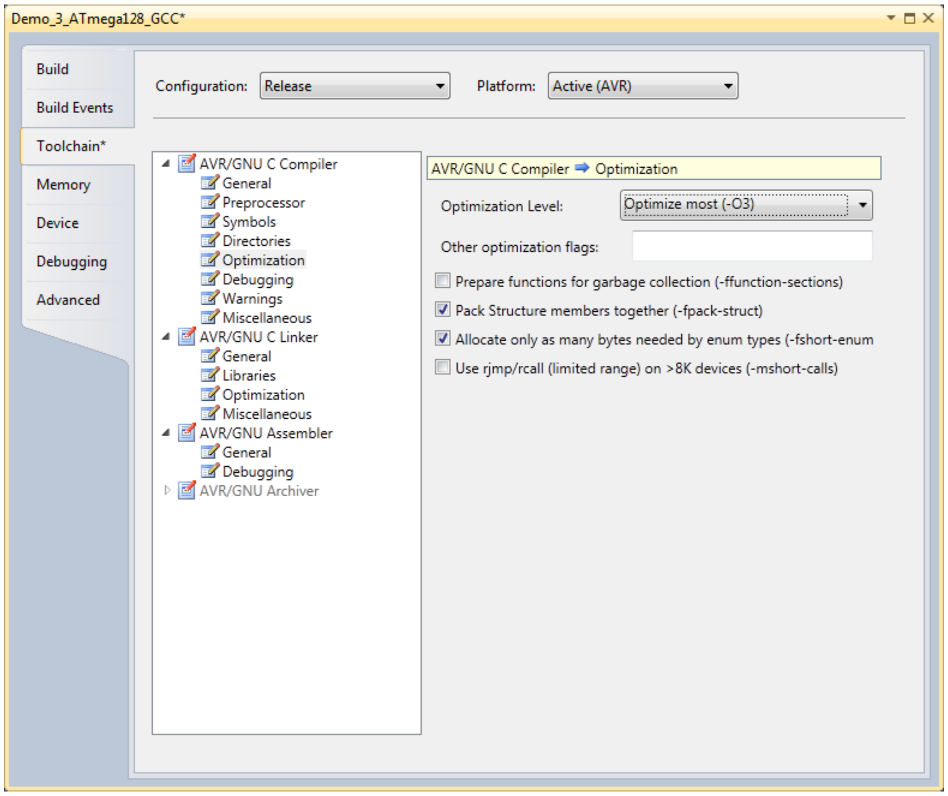
Description	Size
ASM code	296 bytes
Vector Table (per interrupt)	+ 4 bytes
Interrupt prologue (per interrupt)	+ 8 bytes
Hybrid Stack Enabled	+28 bytes
Nested interrupts Enabled	+14 bytes

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 the Code Time Technologies website.

7.2 Latency

Latency of operations has been measured on an Olimex Evaluation board populated with a 16 MHz ATMEGA128-16AU device. All measurements have been performed on the real platform, using the timer peripheral `TIMER/COUNTER3` 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 `TIMER/COUNTER3` counter value. This instrumentation can add up to 5 or 6 cycles to the measurements. The code optimization settings used for the latency measurements are:

1.	Optimization Level:	Optimize most (-O3)
2.	Debug Level:	None



Configuration: Release
Platform: Active (AVR)
Optimization Level: Optimize most (-O3)
Other optimization flags:
[ ] Prepare functions for garbage collection (-ffunction-sections)
[✓] Pack Structure members together (-fpack-struct)
[✓] Allocate only as many bytes needed by enum types (-fshort-enum
[ ] Use rjmp/rcall (limited range) on >8K devices (-mshort-calls)

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.

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 ATMEGA128. 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	229 (243)	399 (449)
Semaphore waiting no blocking	220 (236)	415 (476)
Semaphore posting with task switch	396 (436)	767 (867)
Semaphore waiting with blocking	395 (401)	800 (862)
Semaphore posting in ISR with task switch	651 (699)	1071 (1177)
Event setting no task switch	n/a	395 (448)
Event getting no blocking	n/a	449 (510)
Event setting with task switch	n/a	831 (916)
Event getting with blocking	n/a	852 (914)
Event setting in ISR with task switch	n/a	1134 (1225)
Mailbox writing no task switch	n/a	510 (568)
Mailbox reading no blocking	n/a	502 (559)
Mailbox writing with task switch	n/a	882 (987)
Mailbox reading with blocking	n/a	921 (984)
Mailbox writing in ISR with task switch	n/a	1196 (1308)
Interrupt Latency	71	71
Interrupt overhead entering the kernel	255 (263)	304 (310)
Interrupt overhead NOT entering the kernel	105	105
Context switch	112	112

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                          */
```