CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

ATmega128 – IAR

Disclaimer

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 IAR Embedded Workbench for AVR 6.11; the specific version used for the port and all tests is Version 6.11.

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_IAR.s90`	RTOS assembly file for the ATmega128 to use with the IAR Embedded Workbench
`Abassi_IAR_MTX_IF.c`	Abassi interface functions for multithread-safe operation of the IAR DLIB for EW AVR Version >= 6.10
`Demo_3_AVRMT128_IAR.c`	Demo code that runs on the Olimex AVR-MT-128 evaluation board using the serial port
`Demo_4_AVRMT128_IAR.c`	Demo code that runs on the Olimex AVR-MT-128 evaluation board using the LCD
`Demo_4_AVRMT128_IAR.c`	Demo code that runs on the Olimex AVR-MT-128 evaluation board using the UART.
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

The tiny memory model is not supported in this port.

2 Target Set-up

Very little is needed to configure the IAR Embedded Workbench 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_IAR.s90` in the source files of the application project, and make sure the configuration settings (described in the following subsections) in the file `Abassi_ATMEGA128_IAR.s90` 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`.

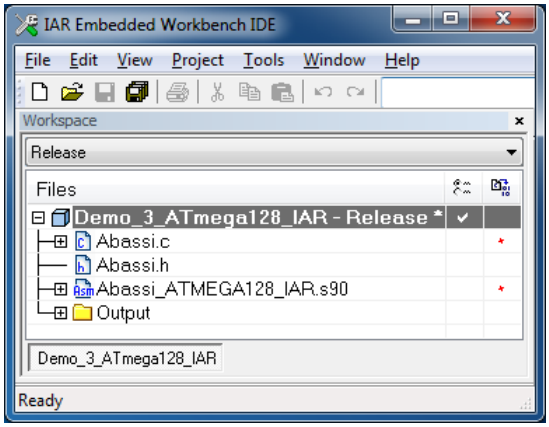

Figure 2-1 Project File List

NOTE: By default, some functions in the IAR Embedded Workbench C/C++ run-time library are not multithread-safe. As such, library functions like `printf()`, `malloc()`, or `fopen()` should be made multithread-safe through the use of a mutex. It is also advisable to use a single mutex (`G_OSmutex`) for all accesses to the non- multithread-safe modules, as some non- multithread-safe modules quite likely call other non- multithread-safe ones.

The IAR Embedded Workbench for the AVR Version 6.10 (and up) supports multithread-safe libraries as long as the `--guard_calls` option is set in the C/C++ Compiler Extra Options box, as the library supplies the RTOS interface functions for the mutexes. The libraries may need to be rebuilt to enable this feature; refer to the IAR "C/C++" User's Guide. These interface functions are supplied in the file `Abassi_IAR_MTX_IF.c`. All there is to do is add the file `Abassi_IAR_MTX_IF.c` in the application project.

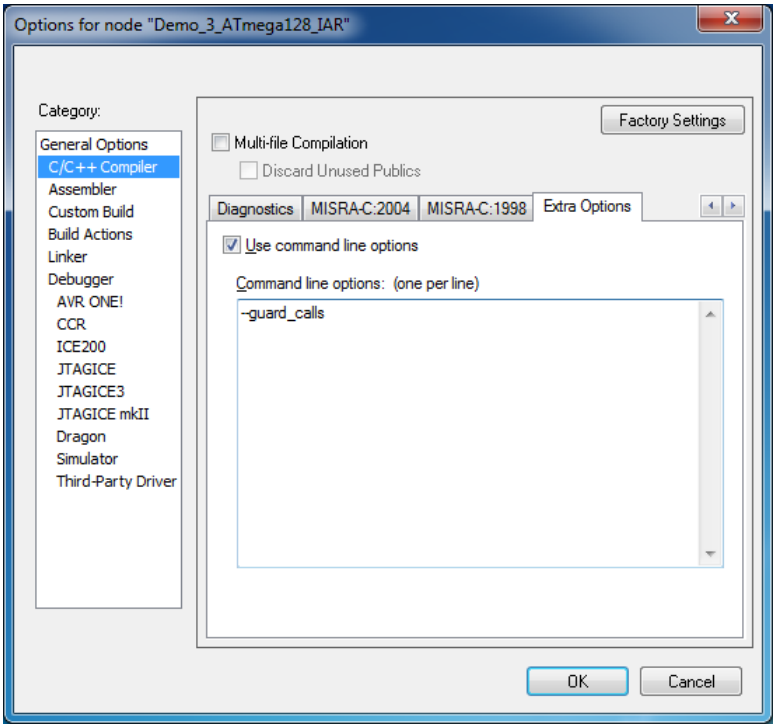

Figure 2-2 Run-time Library Configuration

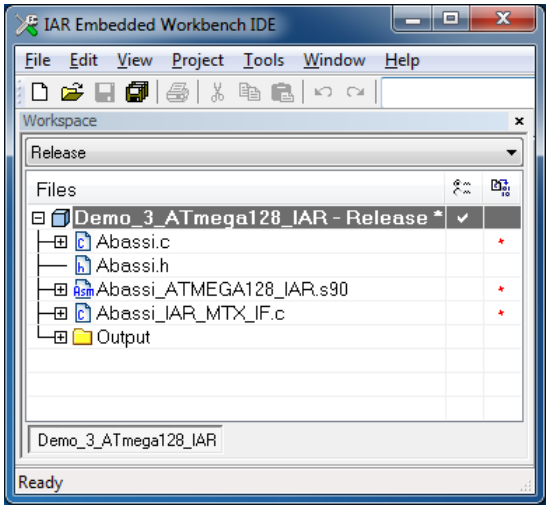

Figure 2-3 Thread-safe Project File List

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, and by the definition `OS_CODE_ISR_STACK`, located around line 35, in the file `Abassi_ATMEGA128_IAR.s90`. To disable this feature, set the definition of `OS_ISR_STACK` to a value of zero. To enable it, and specify the interrupt data stack size, set the definition of `OS_ISR_STACK` to the desired size in bytes. To set the code stack size, set the definition of `OS_CODE_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, a data stack size of 64 bytes is allocated, and the code stack size is set to the application code stack size; this is shown in the following table:

Table 2-1 Interrupt Stack enabled

```
#ifndef OS_ISR_STACK
OS_ISR_STACK       EQU 64            ; If using a dedicated stack for the ISRs
#endif                               ; 0 if not used, otherwise size of stack in bytes

#ifndef OS_CODE_ISR_STACK            ; Code stack size for ISR. If not defined, re-use
OS_CODE_ISR_STACK  EQU OS_CODE_STACK ; the value set for the function code stack size
#endif
```

Table 2-2 Interrupt Stack disabled

```
#ifndef OS_ISR_STACK
OS_ISR_STACK       EQU 0             ; If using a dedicated stack for the ISRs
#endif                               ; 0 if not used, otherwise size of stack in bytes


#ifndef OS_CODE_ISR_STACK            ; Code stack size for ISR. If not defined, re-use
OS_CODE_ISR_STACK  EQU OS_CODE_STACK ; the value set for the function code stack size
#endif
```

Alternatively, it is possible to overload the `OS_STACK_SIZE` and `OS_CODE_ISR_STACK` values set in `Abassi_ATMEGA128_IAR.s90` by using the assembler command line option `-D` and specifying the desired hybrid stack sizes. In the following example, the ISR data stack size is set to 128 bytes and the ISR code stack size is set to 8 bytes:

Table 2-3 Command line set of `OS_ISR_STACK`

```
aavr … -DOS_ISR_STACK=128 -DOS_CODE_ISR_STACK=8  …
```

The interrupt stack size can also be set through the GUI, in the "*Assembler / Preprocessor*" menu, as shown in the following figure:

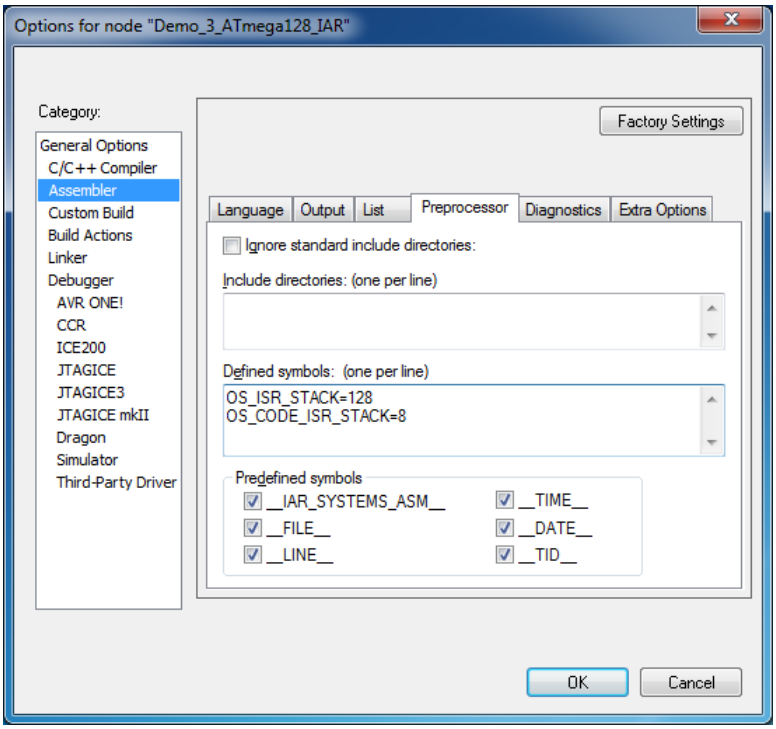

Figure 2-4 GUI set of `OS_ISR_STACK` and `OS_CODE_ISR_STACK`

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 `OS_NESTED_INTS` in the `Abassi_ATMEGA128_IAR.s90` file, around line 40, to a non-zero value, as shown in the following table:

Table 2-4 Nested Interrupts enabled

```
#ifndef OS_NESTED_INTS
OS_NESTED_INTS     EQU     1    ; To allow interrupt nesting, set to non zero
#endif                          ; 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 `OS_NESTED_INTS` in the `Abassi_ATMEGA128_IAR.s90` file to a zero value, as shown in the following table:

Table 2-5 Nested Interrupts disabled

```
#ifndef OS_NESTED_INTS
OS_NESTED_INTS     EQU     0    ; To allow interrupt nesting, set to non-zero
#endif                          ; To not allow interrupt nesting, set to zero
```

Alternatively, it is possible to overload the `OS_NESTED_INTS` value set in `Abassi_ATMEGA128_IAR.s90` by using the assembler command line option `-D` and specifying the setting for the nesting of the interrupts. Even though the token name is identical to the Abassi build option, a definition passed to the compiler does not get propagated to the assembler, so the assembler option `-D` must also be used. The following example shows the activation of the nesting for the interrupts:

Table 2-6 Command line set of `OS_NESTED_INTS`

```
aavr  … -DOS_NESTED_INTS=1  …
```

The control of the interrupt nesting can also be set through the GUI, in the "*Assembler / Preprocessor*" menu, as shown in the following figure:

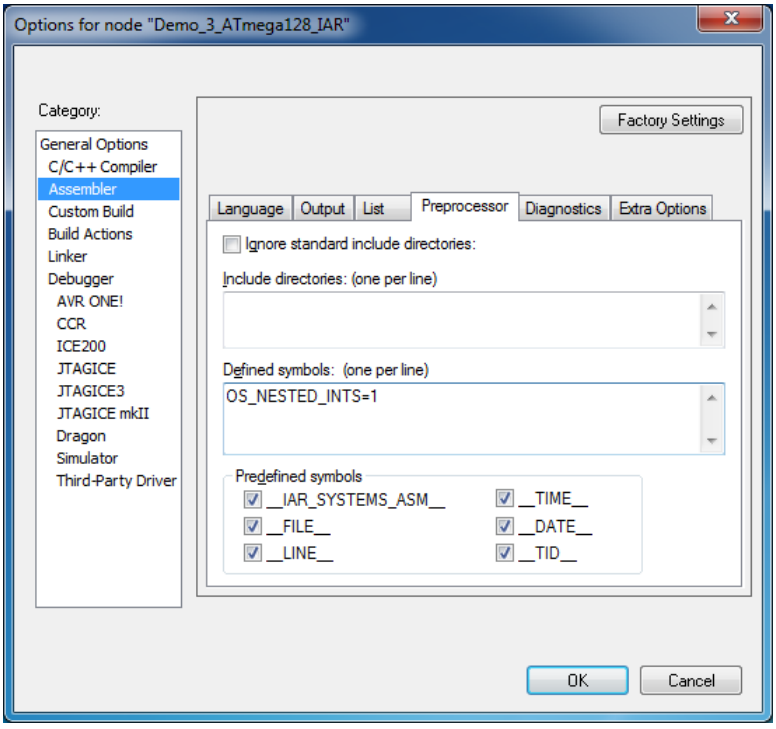

Figure 2-5 GUI set of `OS_NESTED_INTS`

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_IAR.s90` is set to a non-zero value. If the token `NESTED_INTS` in the file `Abassi_ATMEGA128_IAR.s90` 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 Compiler Options

This subsection describes the configuration changes to make in the `Abassi_ATMEGA128_IAR.s90` when the application is built enabling some compiler options.

2.3.1 Option -lock_regs

The "C" compiler can be configured to not use a set of registers, with the option `-lock_regs`. This affects the implementation of the assembly-coded functions in the file `Abassi_ATMEGA128_IAR.s90`. If this option is set to a non-zero value in the "*C/C++ / Code / Number of registers to lock for global variables*" menu in the compiler configuration menu, then the token `OS_OPT__lock_regs`, defined in the file `Abassi_ATMEGA128_IAR.s90` around line 50, must be set to the same value specified in the "*C/C++ / Code / Number of registers to lock for global variables*" menu, as shown in the following table:

Table 2-7 `4` registers set global

```
#ifndef OS_OPT__lock_regs
OS_OPT__lock_regs  EQU     4    ; Set to zero if the "C" option --lock_regs is not used
#endif                          ; Set to N if the "C" option --lock_regs is used
```

If the compiler is not configured to reserve any registers as global variables, but can use them as any other registers, leave the token `OS_OPT__lock_regs` set to a zero value, as originally supplied in the distribution.

Table 2-8 No registers set as a global

```
#ifndef OS_OPT__lock_regs
OS_OPT__lock_regs  EQU     0    ; Set to zero if the "C" option --lock_regs is not used
#endif                          ; Set to N if the "C" option --lock_regs is used
```

Alternatively, it is possible to overload the `OS_OPT__lock_regs` value set in `Abassi_ATMEGA128_IAR.s90` by using the assembler command line option `-D` and specifying the required usage of the `r4` register. In the following example, the register is reserved as a global register:

Table 2-9 Command line set of `OS_OPT__lock_regs`

```
aavr  … -DOS_OPT__lock_regs=4  …
```

The usage of reserved registers can also be set through the GUI, in the "*Assembler / Preprocessor*" menu, as shown in the following figure:

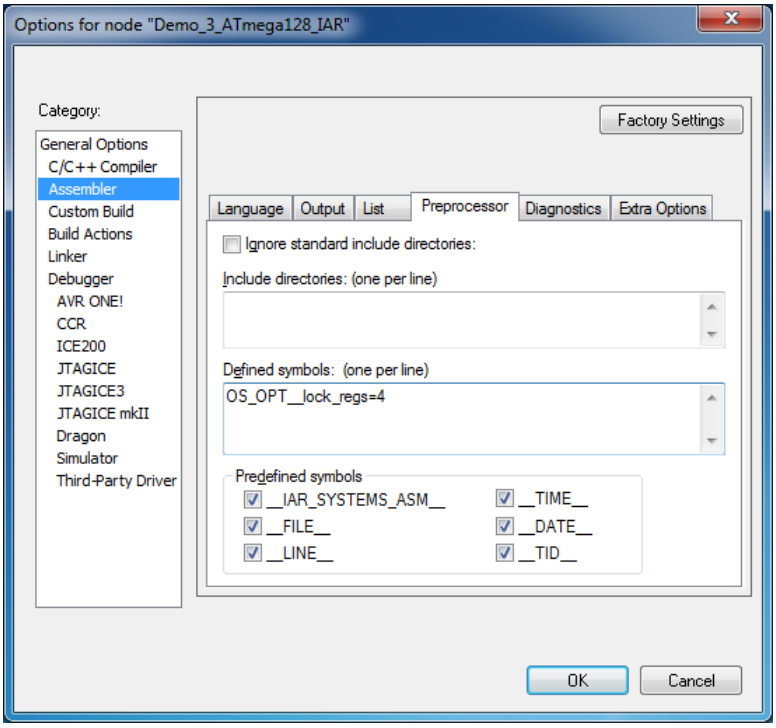

Figure 2-6 GUI set of `OS_OPT__lock_regs`

2.3.2 Option –zero_register

The "C" compiler can be configured to force the register `r15` to always hold a zero value, with the option `-zero_register`. Nothing special needs to be done when this compiler option is set, as the code in the file `Abassi_ATMEGA128_IAR.s90` was designed and implemented to handle this case.

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 IAR Embedded Workbench, 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.

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        */
```

Alternatively, instead of using a hard coded number, the standard definition supplied by the file `iom128.h` can be used. These definitions are set to the vector table offset, specified in bytes; since `OSisrIntall()` uses the interrupt vector index value, these definitions must be divided by 4, as shown in Table 3-2.

Table 3-2 Attaching a Function to an Interrupt

```
#include "Abassi.h"
#include <ATmega128.h>

  …
  OSstart();
  …
  OSisrInstall(TIMER0_OVF_vect/4, &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 Embedded Workbench specific `__interrupt` prefix statement.

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 3-3:

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_ATMEGA128_IAR.s90`, 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 190, 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_TBL  XX, INTXX_handler
```

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

```
   ISR_TBL  XX, INT_NO_handler
```

The second step is to create the do-nothing interrupt handler. This step only needs to be performed once, as the same do-nothing handler should be re-used for all unused interrupts. The do-noting 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 `INT02_handler` label; this should be around line 350 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

```
INTXX_handler:
    ISR_PROLOGUE   XX
```

Commenting out the `ISR_PROLOGUE` line for the unused interrupt will remove the prologue code. It is not necessary to remove the label.

Table 3-8 Interrupt dispatcher prologue removal

```
INTXX_handler:
;;; 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 set the address of the interrupt function in the corresponding entry in the interrupt vector table that is used by the ATmega128 processor. The beginning of the interrupt vector table is located in the file `Abassi_ATMEGA128_IAR.s90` around line 190, at the label `VectTbl`. For example, on a ATMEGA128F1611 device, `TIMERA` is set to the priority 6. This is the entry in the table for `TIMERA` in the distribution file:

Table 3-9 ATMEGA128-16AU TIMER/COUNTER1 Regular Interrupt

```
    ISR_TBL    14, INT14_handler
```

To attach a fast interrupt handler to the TIMER/COUNTER1 overflow interrupt, assuming the name of the interrupt function to attach is TIMER1_handler(), replace the previous line with that shown in Table 3-10:

Table 3-10 ATMEGA128-16AU TIMER/CONTER1 Fast Interrupt

```
    EXTERN  TIMER1_handler
    ISR_TBL   14, TIMER1_handler
```

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 IAR 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 also 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 almost identical to what is done in the regular interrupt dispatcher. Reusing the example of the TIMER/COUNTER1 on a ATMEGA128-16AU device, this would look something like:

Table 3-11 Fast Interrupt with Dedicated Stack

```
    …
    …

    ISR_TBL  14, TIMER1_preHandler

    …
    …

    RSEG    CODE:CODE:ROOT(1)

    EXTERN  TIMER1_handler

TIMER1_preHandler:

    st      -Y, r31                         ; 2 regs needed to set up ISR stack
    st      -Y, r30

    sts     DataStackISR-1, Y1              ; We need to start using the ISR stack
    sts     DataStackISR-2, Y0              ; Memo current Y on the Data ISR stack
    ldi     Y0, LOW(T1_DataStackISR-2)      ; Set-up Y to use the Data ISR stack
    ldi     Y1, (T1_DataStackISR-2) >> 8
    in      r31, SPH                        ; Memo current code stack and set-up to
    st      -Y, r31                         ; start using the code ISR stack
    in      r31, SPL                        ; Interrupt disable in here, so no issues
    st      -Y, r31
    ldi     r31, LOW(T1_CodeStackISR)
    out     SPL, r31
    ldi     r31, (T1_CodeStackISR)>>8
    out     SPH, r31

    in      r30, SREG                       ; A little bit more context save
```

```
    st      -Y, r30                         ; And we need to also preserved SREG
    st      -Y, r23
    st      -Y, r22
    st      -Y, r21
    st      -Y, r20
    st      -Y, r19
    st      -Y, r18
    st      -Y, r17
    st      -Y, r16
    st      -Y, r3
    st      -Y, r2
    st      -Y, r1
    st      -Y, r0


    call     TIMER1_handler                 ; Enter the interrupt handler


    ld      r0,  Y+                         ; Context restore
    ld      r1,  Y+
    ld      r2,  Y+
    ld      r3,  Y+
    ld      r16, Y+
    ld      r17, Y+
    ld      r18, Y+
    ld      r19, Y+
    ld      r20, Y+
    ld      r21, Y+
    ld      r22, Y+
    ld      r23, Y+
    ld      r30, Y+
    out     SREG, r30

    ld      r30, Y+                         ; Read original code stack pointer
    out     SPL, r30                        ; and put it back in SP
    ld      r30, Y+
    out     SPH, r18
    ld      r30, Y+                         ; Read original data stack pointer
    ld      r31, Y
    movw    Y1:Y0,r30:r31                   ; Put it back in

    ld      r30, Y+
    ld      r31, Y+

    reti                                    ; exit the ISR

    …
    …

    RSEG    RSEG  NEAR_N:DATA(1)

    DS8     T1_Data_stack_size              ; Room for the fast interrupt data stack
T1_ DataStackISR:
    DS8    (T1_Data_stack_size)-1           ; Room for the fast interrupt code stack
T1_ CodeStackISR:
    DS8     1
```

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 (global interrupt enable) bit 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 ATmega128 interrupt controller only allows the servicing of a single interrupt at any time. If nesting is desired on the fast interrupts with the hybrid stack, extreme care must be taken, as a critical region exists when the code stack is set back to its pre-interrupt value.

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 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	`18 bytes`[1]
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 `OS_ISR_STACK` token in the file `Abassi_ATMEGA128_IAR.s90` is set to a non-zero value (Section 2.1).

[1] Subtract N bytes when the `OS_OPT__lock_regs` tokens is set to a non-zero value of N.

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 401 cycles.

Table 5-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	431	494	662
2	440	505	668
3	451	516	689
4	462	527	680
5	473	538	808
6	484	549	707
7	495	560	728
8	506	498	704
9	517	507	725
10	528	518	731
11	539	529	752
12	550	540	743
13	561	551	764
14	572	562	770
15	583	573	791
16	594	511	653
17	605	520	674
18	616	531	680
19	627	542	701
20	638	553	692
21	649	564	713
22	660	575	719
23	671	586	740
24	682	524	716

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 11 CPU cycles. When `OS_SEARCH_FAST` is set to 1, each extra priority level to traverse also requires exactly 11 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 65 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 13 cycles needed, but without the 8 times 11 cycles accumulation.

What does this mean? Using 8 tasks or more on the ATmega128 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 more than 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 IAR Embedded Workbench. 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: High
2. Optimize for: Size
3. Always do cross call optimization Disabled

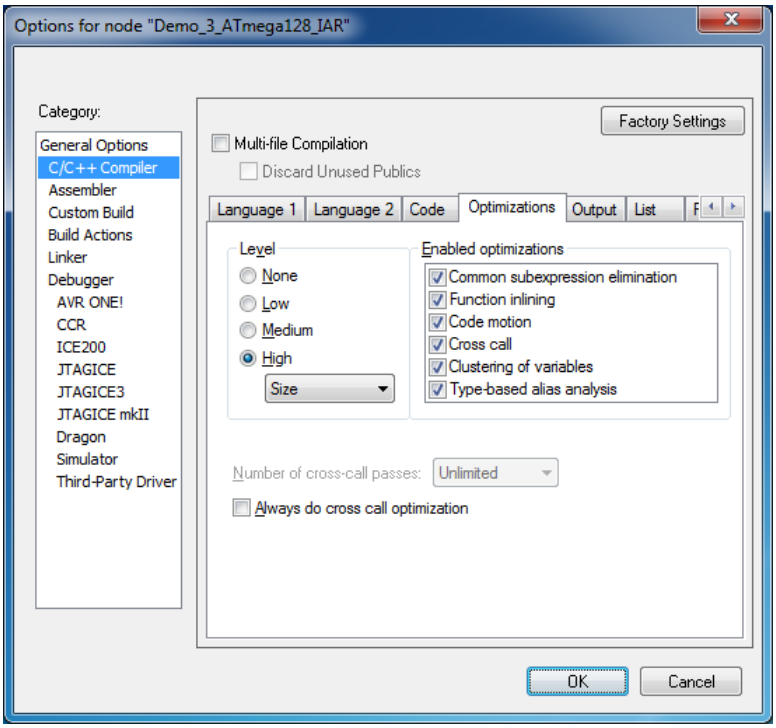

Figure 7-1 Memory Measurement Code Optimization Settings

Table 7-1 "C" Code Memory Usage

Description	Code Size
Minimal Build	< 950 bytes
+ Runtime service creation / static memory	< 1225 bytes
+ Multiple tasks at same priority	< 1350 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 2025 bytes
+ Timer & timeout + Timer call back + Round robin	< 3625 bytes
+ Events + Mailbox	< 4375 bytes
Full Feature Build (no names)	< 4050 bytes
Full Feature Build (no names / no runtime creation)	< 3775 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 4550 bytes

Table 7-2 Assembly Code Memory Usage

Description	Size
ASM code	294 bytes
Vector Table (per interrupt)	+ 4 bytes
Interrupt prologue (per interrupt)	+ 8 bytes
Hybrid Stack Enabled	+42 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 `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 `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: High
2. Optimize for: Speed
3. Always do cross call optimization Disabled

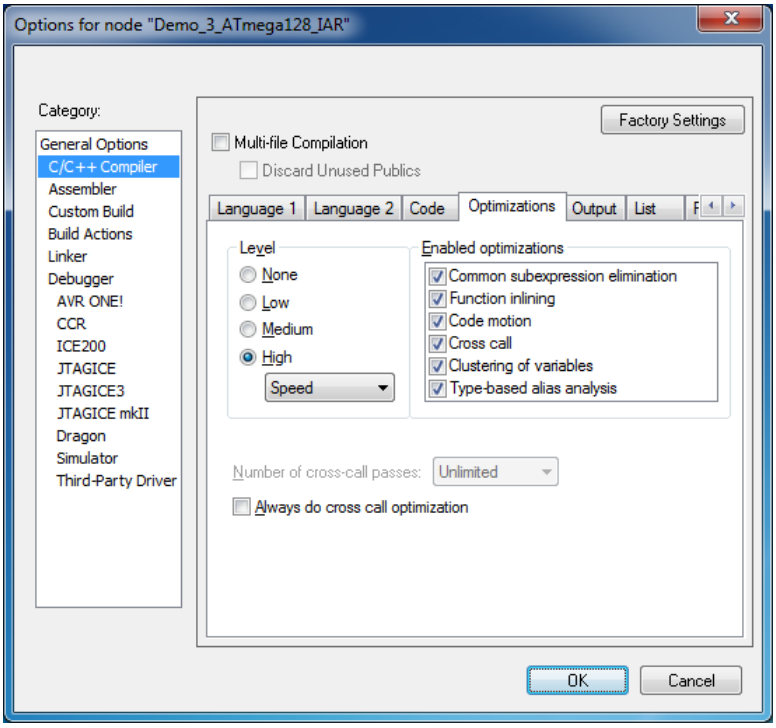

Figure 7-2 Latency Measurement Code Optimization Settings

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	208 (207)	308 (308)
Semaphore waiting no blocking	215 (214)	325 (324)
Semaphore posting with task switch	374 (413)	579 (607)
Semaphore waiting with blocking	395 (378)	607 (587)
Semaphore posting in ISR with task switch	603 (634)	839 (862)
Event setting no task switch	n/a	306 (305)
Event getting no blocking	n/a	367 (366)
Event setting with task switch	n/a	616 (643)
Event getting with blocking	n/a	647 (627)
Event setting in ISR with task switch	n/a	878 (900)
Mailbox writing no task switch	n/a	390 (389)
Mailbox reading no blocking	n/a	397 (397)
Mailbox writing with task switch	n/a	671 (698)
Mailbox reading with blocking	n/a	697 (675)
Mailbox writing in ISR with task switch	n/a	937 (958)
Interrupt Latency	60	60
Interrupt overhead entering the kernel	229 (221)	260 (255)
Interrupt overhead NOT entering the kernel	104	104
Context switch	108	108

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                          */
```