CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

PIC32 – MPLAB/GCC Suite

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

This document details the port of the Abassi RTOS to the Microchip PIC32 family of microcontrollers. The software suite used for this specific port is the MPLAB IDE using the GCC tool suite; the version used for the port and all tests is Version 8.83.00.00.

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_PIC32_MPLAB.s`	RTOS assembly file for the PIC32 with MPLAB/GCC tool suite
`Demo_0_PIC32_MPLAB.c`	Demo code that runs on both the PIC32 Starter Kit and the Olimex PIC32-WEB boards
`AbassiDemo.h`	Build option settings for the demo code

1.2 Limitations

The Abassi RTOS port on the PIC32 does not support Fast Interrupts.

2 Target Set-up

Very little is needed to set-up the MPLAB development environment in order to use the RTOS in an application. All there is to do is add the files `Abassi.c` and `Abassi_PIC32_MPLAB.s` in the source files of the application project, the file `Abassi.h` in the header files of the application project, and make sure the three configuration settings in the file `Abassi_PIC32_MPLAB.s` (`OS_USE_MIPS16E` as described in Section 2.1 , `OS_ISR_STACK` described in Section 2.2, and `OS_ISR_CLR_CUTOFF` as described in Section 3.1.2) are set according to the needs of the application. As well, update the include file path with the location of `Abassi.h`.

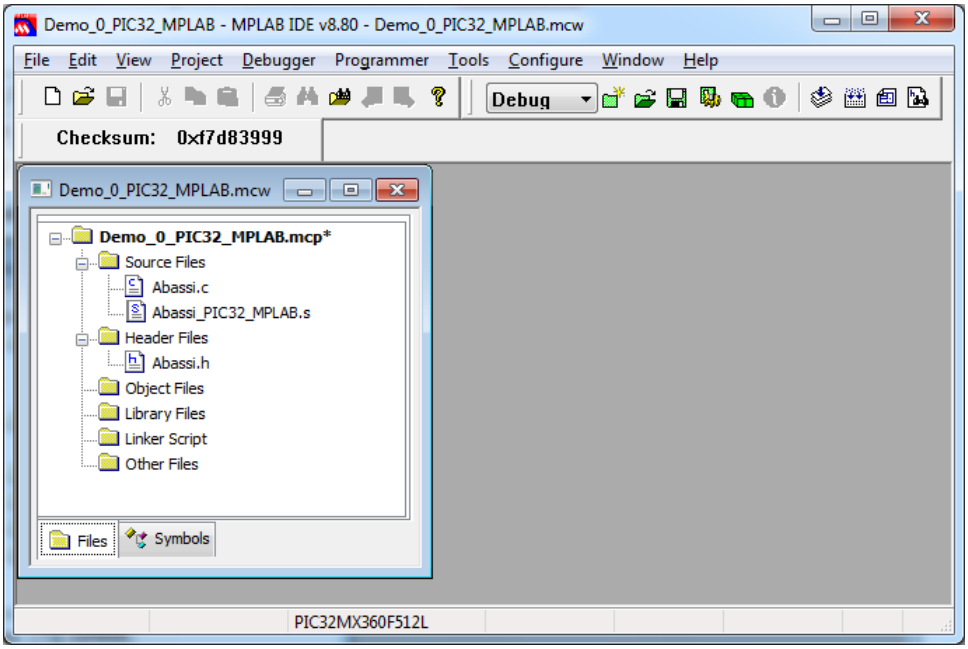

Figure 2-1 Project File List

2.1 MIPS16e Set-up

The processor used on the PIC32 family of devices is the MIPS32 M4K processor, and this processor architecture can decode two different sets of instructions: the native 32 bit (MIPS32) or the 16 bits (MIPS16e) code compression instruction set. The MIPS16e instruction set is selected in MPLAB under the "Build Options / MPLAB PIC32 C Compiler / General" window by selecting "Generate 16-bit code". A snapshot of the window is shown in Figure 2-2 below:

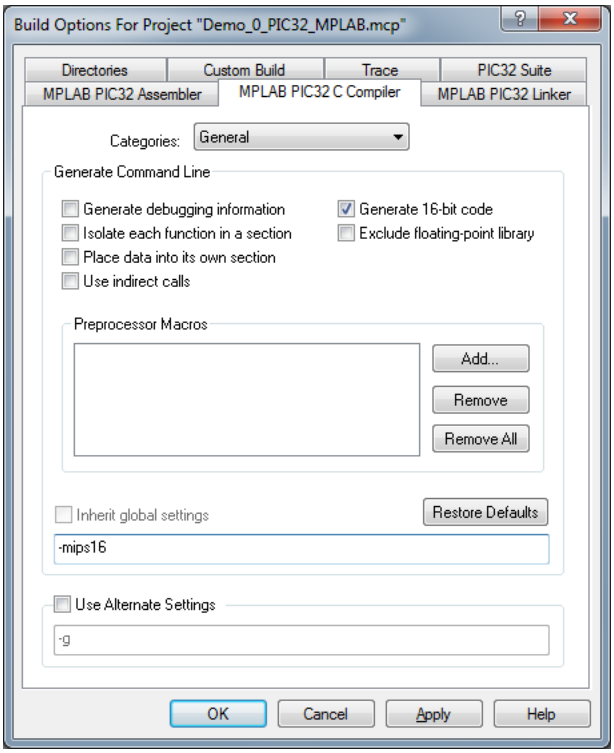

Figure 2-2 MIPS16e Code Generation Selection

If the compiler is used through the command line, the MIPS16e instruction set is selected with the option `-mips16`, as shown below:

Table 2-1 Command line setting for MIPS16e instruction set

```
pic32-gcc  … -mips16 …
```

The MIPS16e libraries are selected in MPLAB under the "Build Options / MPLAB PIC32 Linker / Library Selection" window by selecting "Generate 16-bit code". A snapshot of the window is shown in Figure 2-3 below:

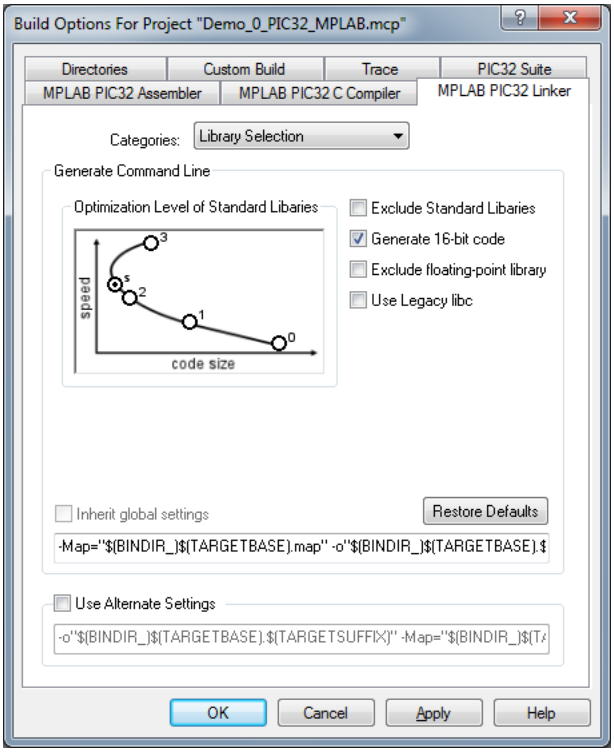

Figure 2-3 MIPS16e Library Selection

If the linker is used through the command line with `pic32-gcc`, the MIPS16e instruction set libraries are selected with the option `-mips16`, as shown below:

Table 2-2 Command line for MIPS16e libraries

```
pic32-gcc  … -mips16 …
```

When code is generated with the MIPS16e instruction set, the number of registers used by the CPU is reduced and the versatility of the function calling convention is reduced. This means it becomes possible to achieve some CPU cycle count savings when performing task switching and handling interrupts when the MIPS16e instruction set is selected, as the number of registers to preserve and restore is reduced compared to the native MIPS32 register set.

NOTE: This implies that all application code, without exception, including the libraries linked with the application, uses the MIPS16e instruction set and only that instruction set.

The following examples show the few lines around line number 30 in the file `Abassi_PIC32_MPLAB.s`, and they illustrate how to set-up the build for either a 32 bit application or a MIPS16e application:

Table 2-3 Build for 32 bit application

```
.ifndef OS_USE_MIPS16E
   .equ  OS_USE_MIPS16E, 0            # Set to 0 for 32 bit (MIPS32) application
 .endif                               # Set to != 0 for 16 bits MIPS16e application
```

Table 2-4 Build for 16 bit application

```
 .ifndef OS_USE_MIPS16E
   .equ  OS_USE_MIPS16E, 1            # Set to 0 for 32 bit (MIPS32) application
 .endif                               # Set to != 0 for 16 bits MIPS16e application
```

Alternatively, it is possible to overload the `OS_USE_MIPS16E` value set in `Abassi_PIC32_MPLAB.s` by using the assembler command line option `--defsym`[1] and specifying the desired instruction set. In the following example, the MIPS16e instruction set is selected:

Table 2-5 Command line set of `OS_USE_MIPS16E`

```
pic32-as  … --defsym=OS_USE_MIPS16E=1  …
```

Or using the compiler:

Table 2-6 Command line set of `OS_USE_MIPS16E`

```
pic32-gcc  … -Wa,--defsym=OS_USE_MIPS16E=1  …
```

[1] The manual and error message of the assembler indicates the syntax being `--defsym sym=value` but in reality, the syntax is `--defsym=sym=value`.

The selection of the instruction set for `Abassi_PIC32_MPLAB.s` can also be controlled through the GUI, in the "*MPLAB PIC32 Assembler / Use Alternate Settings*" menu, as shown in the following figure:

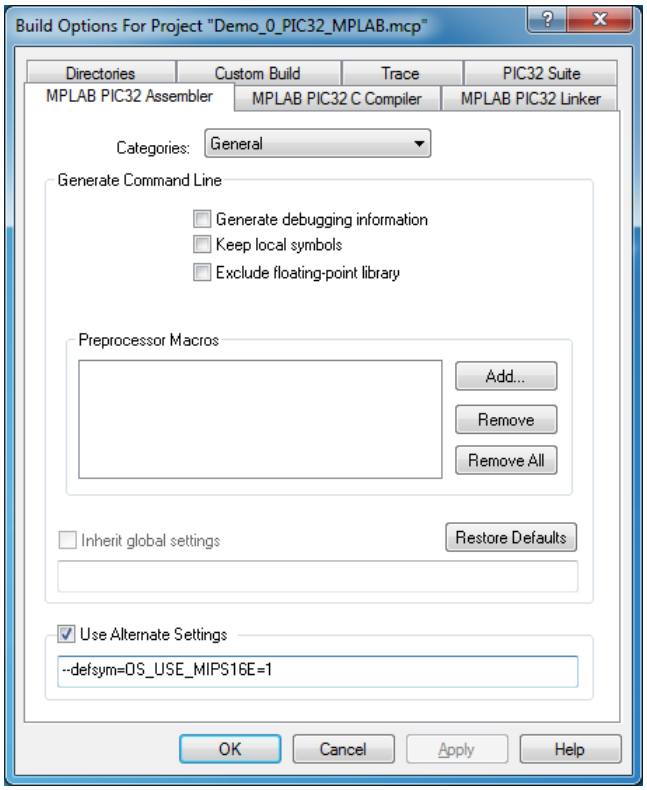

Figure 2-4 GUI set of `OS_USE_MIPS16E`

By examining the assembly code in the `Abassi_PIC32_MPLAB.s` file, it is obvious all of it has been coded with the 32 bit instructions. The CPU can intermix both instruction types in the same application, and as it is impossible to access the `CP0` registers using the MIPS16e instruction set, it is necessary to utilize the 32 bit instructions to handle the interrupts and to control the enabling/disabling of interrupts. The build flag `OS_USE_MIPS16E` controls the set of registers that are preserved during a context switch and in the interrupt dispatcher. The code, as distributed, sets the flag `OS_USE_MIPS16E` to 0, configuring the build for 32 bit applications. When `Abassi_PIC32_MPLAB.s` is built for a 32 bit application, it can be used with an application using either the 32 bit or MIPS16e instruction set; but when it is built for a MIPS16e, it will only work for MIPS16e built applications; in this case, the presence of 32 bit instructions in the applications will provoke an application crash.

The interrupt auto-clearing feature also needs to be set-up according to the needs of the application; see Section 3.1.2.

2.2 Interrupt Stack Set-up

It is possible, and is highly recommended, to use a hybrid stack when nested interrupts occur in an application. Using this hybrid stack, specially dedicated to the interrupts, removes the need to allocate extra room to the stack of every task in the application to handle the interrupt nesting. This feature is controlled by the value set by the definition `OS_ISR_STACK`, located around line 30 in the file `Abassi_PIC32_MPLAB.s`. To disable this feature, set the definition of `OS_ISR_STACK` to a value of zero. To enable it, and specify the interrupt stack size, set the definition of `OS_ISR_STACK` to the desired size in bytes (see Section 4 for information on stack sizing). As supplied in the distribution, the hybrid stack feature is enabled, and a stack size of 128 bytes is allocated; this is shown in the following table:

Table 2-7 Interrupt Stack enabled

```
.ifndef OS_ISR_STACK
  .equ   OS_ISR_STACK, 1024          # If using a dedicated stack for the nested ISRs
.endif                               # 0 if not used, otherwise size of stack in bytes
```

Table 2-8 Interrupt Stack disable

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

Alternatively, it is possible to overload the `OS_ISR_STACK` value set in `Abassi_PIC32_MPLAB.s` by using the assembler command line option `--defsym`[2] and specifying the desired hybrid stack setting. In the following example, the hybrid stack is set to a size of 512:

Table 2-9 Command line set of `OS_ISR_STACK`

```
pic32-as  … --defsym=OS_ISR_STACK=512  …
```

Or using the compiler:

Table 2-10 Command line set of `OS_ISR_STACK`

```
pic32-gcc  … -Wa,--defsym=OS_ISR_STACK=512  …
```

[2] The manual and error message of the assembler indicates the syntax being `-defsym sym=value` but in reality, the syntax is `-defsym=sym=value`.

The sizing of the hybrid stack used by `Abassi_PIC32_MPLAB.s` can also be controlled through the GUI, in the "*MPLAB PIC32 Assembler / Use Alternate Settings*" menu, as shown in the following figure:

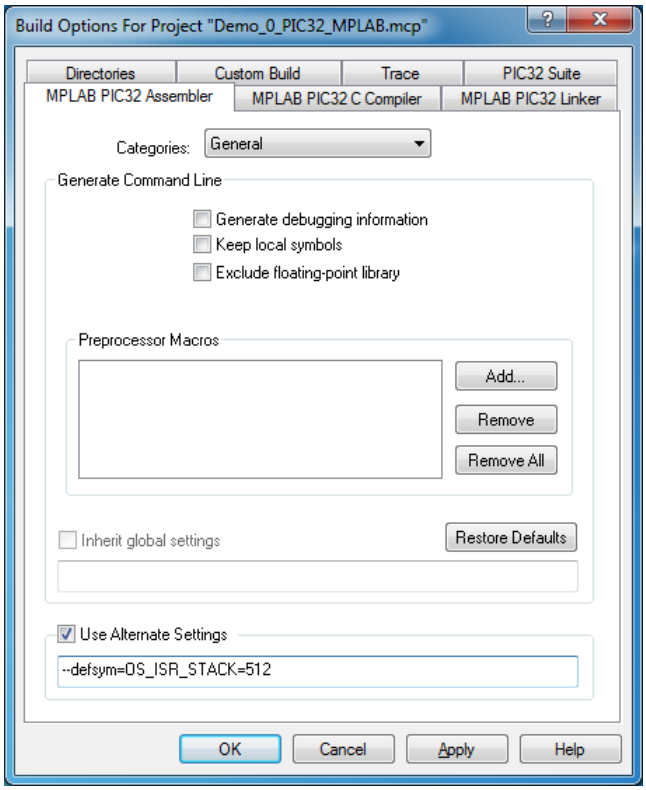

Figure 2-5 GUI set of `OS_ISR_STACK`

3 Interrupts

The Abassi RTOS needs to be aware when kernel requests are performed inside or outside an interrupt context. Normally, when coding with MPLAB, an interrupt function is specified with the "`#pragma interrupt …`" directive. But for all interrupt sources, the Abassi RTOS provides an interrupt dispatcher, which allows it to be interrupt-aware. This dispatcher achieves two goals. First, the kernel uses it to know if a request to it 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 interrupts.

Out of the box there are provisions for 64 sources of interrupts, as specified by the build option `OS_N_INTERRUPTS`, defined in the file `Abassi.h`[3]. This build option is not part of the generic RTOS build options as it is processor-specific, therefore its value is set according to the specific compiler and processor port. PIC32 family devices have the capability to support 96 sources of interrupts that are remapped to the 64 interrupt vectors.

The PIC32 interrupt controller can operate either in multi-vector mode or in single-vector mode. The Abassi RTOS interrupt dispatcher requires the interrupt controller to operate in single-vector mode and that the general register set, NOT the shadow register set, is used during interrupts. All this set-up is done inside the RTOS by the initialization function `OSstart()`. Nothing extra about the interrupts should be set by the application; everything interrupt related must use the RTOS interrupt installer. The only other set-up an application needs to do for the ISR is to set the priority of the interrupt and enable the interrupt on the peripheral, and if the auto-clearing feature (Section 3.1.2) does not apply to the interrupt vector number or if it is disabled, to clear the interrupt request in the interrupt function. Obviously, configuring the peripheral is also needed.

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 component `OSisrInstall()` to specify the interrupt vector number and the function to be attached to that interrupt vector number:

Table 3-1 Attaching a Function to an Interrupt

```
#include "Abassi.h"

  …
  OSstart();
  …
  OSisrInstall(Number, &ISRfct);
  Set-up the interrupt source
  Enable the interrupt source

  … /* More ISR setup */

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

The function to attach to the interrupt, `ISRfct()` in the above example, is and must always be a regular function.

NOTE: It is a regular function, not one declared with the MPLAB specific "`#pragma interrupt`" prefix statement.

[3] The build option `OS_N_INTERRUPTS` for this port should never be modified.

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 vector 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 the next table:

Table 3-2 Invalidating an ISR handler

```
#include "Abassi.h"

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

One must remember the component `OSisrInstall()` uses the interrupt vector number, NOT the interrupt request number. On the PIC32, multiple interrupt request numbers can be mapped to a single interrupt vector. When this happens, the interrupt function must determine what interrupt request number has triggered the interrupt. The data sheet of each device provides a mapping table for the interrupt request numbers and the interrupt vector numbers in the section "Interrupt Controller". Alternatively, one can use the MPLAB development suite definitions for the interrupt vector number and these include files are located in the device specific definition file, named similar to the part number. For example, for a PIC32MX360F512L and PICMX795F512L devices, the definition file names are `p32mx360f512l.h` and `p32mx795f512l.h` respectively. These include files are in the "...`\MPLAB C32 Suite\pic32mx\include\proc`" folder of the MPLAB IDE install location.

3.1.2 Interrupt Request Auto-Clearing

Examining the pairing of the interrupt request numbers and the interrupt vector numbers in the datasheets table, it is obvious that the pairing is a perfect match up to some interrupt request number. After that cutoff, either an interrupt vector number becomes paired to multiple interrupt request numbers, or single pairings do not have the same values anymore. The following table lists the lowest cutoff value of the interrupt vector number when the pairing does not match anymore for the device family (at the time of writing this document):

Table 3-3 Interrupt Pairing Cutoff Values

Device Family	Cut-Off
PIC32MX1XX	5
PIC32MX2XX	5
PIC32MX3XX	23
PIC32MX4XX	23
PIC32MX5XX	23
PIC32MX6XX	23
PIC32MX7XX	23

It is always necessary to clear the interrupt request by clearing the associated bit in the `IFS0` or `IFS1` register of the interrupt controller; the bit position and register is a function of the interrupt request number. To simplify the implementation of applications using the RTOS, the ISR dispatcher can deal with clearing the request bit by itself, but unfortunately, all the ISR dispatcher can be made aware of is the interrupt vector number. As previously indicated, there is a one to one mapping between the interrupt request number and the interrupt vector number up to a cutoff value. In the file `Abassi_PIC32_MPLAB.s`, at around line number 40, there is a definition that controls if the auto-clearing of interrupt requests is performed by the ISR dispatcher, and what is the interrupt vector number when the ISR dispatcher must stop performing the auto-clearing:

Table 3-4 ISR Auto-clearing Control

```
.ifndef OS_ISR_CLR_CUTOFF
  .equ  OS_ISR_CLR_CUTOFF, 23    # IRQ vector value at which the ISR dispatcher stops
.endif                           # auto-clearing the ISR. Set to 0 to disable
```

In the above table the definition is set to 23, which is the cutoff value for the 3XX, 4XX, 5XX, 6XX and 7XX device families, and this is the value set in the distribution code.

To completely disable the auto-clearing feature, set the definition to a value of zero as shown below:

Table 3-5 ISR Auto-clearing Disabling

```
.ifndef OS_ISR_CLR_CUTOFF
  .equ  OS_ISR_CLR_CUTOFF, 0     # IRQ vector value at which the ISR dispatcher stops
.endif                           # auto-clearing the ISR. Set to 0 to disable
```

Alternatively, it is possible to overload the value of `OS_ISR_CLR_CUTOFF` set in `Abassi_PIC32_MPLAB.s` by using the assembler command line option `--defsym`[4] and specifying the desired hybrid stack setting. In the following example, the cutoff value set to 12:

Table 3-6 Command line set of ISR Auto-clearing

```
pic32-as  … --defsym=OS_ISR_CLR_CUTOFF=12  …
```

Or using the compiler

Table 3-7 Command line set of ISR Auto-clearing

```
pic32-gcc  … -Wa,--defsym=OS_ISR_CLR_CUTOFF=12 …
```

[4] The manual and error message of the assembler indicates the syntax being `-defsym sym=value` but in reality, the syntax is `-defsym=sym=value`.

The selection of the ISR auto-clearing cutoff can also be controlled through the GUI, in the "*MPLAB PIC32 Assembler / Use Alternate Settings*" menu, as shown in the following figure:

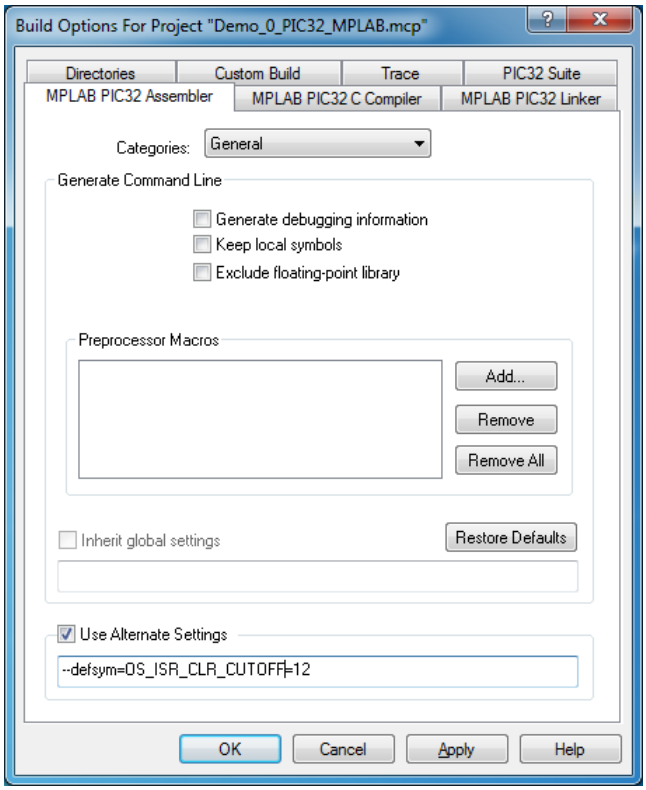

Figure 3-1 GUI set of ISR Auto-clearing

NOTE: Never set the value of `ISR_CLR_CUTOFF` to a value greater than the lowest interrupt vector number where the perfect pairing stops existing.

Here are two examples to show the advantage of the auto-clearing feature. The first example is the code implementing the ISR function for timer #1 with the auto-clearing enabled (or set to a value greater than 4). The timer #1 interrupt vector number is 4 (same as the interrupt request number):

Table 3-8 ISR function with auto-clearing

```
void TimerISR_Auto(void)
{
   g_Count++;                          /* Increment the global count variable    */
   return;
}
```

The next code example is with the auto-clearing feature disabled (or with a value set to 4 or less):

Table 3-9 ISR function without auto-clearing

```
void TimerISR_Auto(void)
{
   mT1ClearIntFlag();               /* Clear the interrupt request           */
   g_Count++;                       /* Increment the global count variable   */
   return;
}
```

One obvious advantage of the auto-clearing feature is that this timer interrupt function can be detached from one timer and attached to another one without modifying the code (as long as both interrupt vector number are below the cutoff value specified in the `ISR_CLR_CUTOFF` definition). In the case of the function without the auto-clearing, the "`mT1ClearIntFlag();`" must be modified when it is attached to a timer different than timer #1. For example, if this interrupt function is attached to timer #3, the statement must then be "`mT3ClearIntFlag();`"

3.2 Regular Interrupts

Regular interrupts are interrupts that can use all RTOS component. The PIC32 port of the RTOS only uses regular interrupts, so there is no ambiguity when using interrupts on a PIC32 device.

3.3 Fast Interrupts

Fast interrupts are not supported on the PIC32 port of the RTOS.

3.4 Nested Interrupts

The PIC32 interrupt controller allows nesting of interrupts; this means an interrupt of higher priority will interrupt the processing of an interrupt of lower priority. Individual interrupt sources (interrupt request numbers) can be set to one of 8 levels, where level 0 is the lowest (interrupts set to the priority level 0 are disabled) and 7 is the highest. This implies that the RTOS build option `OS_NESTED_INTS` must be set to a non-zero value. The exception to this is if all enabled interrupts in an application are all set, without exception, to the same priority; then interrupt nesting will not occur. In that case, and only that case, can the build option `OS_NESTED_INTS` be set to zero. As this latter case is quite unlikely, the build option `OS_NESTED_INTS` is always overloaded when compiling the RTOS for the PIC32. If the latter condition is guaranteed, the overloading located after the pre-processor directive can be modified. The code affected in `Abassi.h` is shown in Table 3-10 below and the line to modify is the one with `#define OX_NESTED_INTS 1`:

Table 3-10 Removing interrupt nesting

```
#elif defined(__GNUCC__) && defined(__PIC32MX_)
    …
  #define OX_NESTED_INTS 0 /* The PIC32 has 8 interrupt levels             */
```

Or if the build option `OS_NESTED_INTS` is desired to be propagated:

Table 3-11 Propagating interrupt nesting

```
#elif defined(__GNUCC__) && defined(__PIC32MX_)
    …
  #define OX_NESTED_INTS OS_NESTED_INTS
```

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 PIC32, 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 (MIPS32)	`64 bytes`
Blocked/Preempted task context save (MIPS16e)	`16 bytes`
Interrupt context save (MIPS32)	`112 bytes`
Interrupt context save (MIPS16e)	`68 bytes`

When sizing the stack to allocate to a task, there are three factors to take in account. The first factor is simply that every task in the application needs at least the area to preserve the task context when it is preempted or blocked. Second, one must take into account how many levels of nested interrupts exist in the application. As a worst case, all levels of interrupts may occur and becoming fully nested. So, if N levels of interrupts are used in the application, provision should be made to hold N times the size of an ISR context save on each task stack, plus any added stack used by the interrupt handler functions. Finally, add to all this the stack required by the code implementing the task operation.

NOTE: The MIPS32 processor needs alignment on 8 words for some instructions accessing memory. When stack memory is allocated, Abassi guarantees the alignment. This said, when sizing `OS_STATIC_STACK` or `OS_ALLOC_SIZE`, make sure to take in account that all allocation performed through these memory pools are by block size multiple of 8 bytes.

If the hybrid interrupt stack (see Section 2.2) 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 file `Abassi_PIC32_MPLAB.s` is set to a non-zero value (Section 2.2).

5 Search Set-up

The Abassi RTOS build option `OS_SEARCH_FAST` offers three different algorithms to quickly determine the next running task upon task blocking. The following table shows the measurements obtained for the number of CPU cycles required when a task at priority 0 is blocked, and the next running task is at the specified priority. The number of cycles includes everything, not just the search cycle count. The number of cycles was measured using the timer #4-5 pair set to use the system clock with a pre-scale of 1; this means the timer pair counter increments once every CPU cycle. The second column is when `OS_SEARCH_FAST` is set to zero, meaning a simple array traversing. The third column, named Look-up, is when `OS_SEARCH_FAST` is set to 1, which uses an 8 bit look-up table. Finally, the last column is when `OS_SEARCH_FAST` is set to 5 (MPLAB/PIC32 `int` are 32 bits, so 2^5), meaning a 32 bit look-up table, further searched through successive approximation. The compiler optimization for this measurement was set to Level 3 (`-O3`) with 32 bit instructions. The build options were set to the minimum feature set except for the 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-dimentsion linked list search technique instead of the search array, the number of CPU is constant at 213 cycles.

Table 5-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	210	253	288
2	220	256	288
3	224	260	290
4	228	264	288
5	232	268	290
6	236	272	290
7	240	276	292
8	244	250	288
9	248	264	290
10	252	267	290
11	256	271	292
12	260	275	290
13	264	279	292
14	268	283	292
15	272	287	294
16	276	257	286
17	280	271	288
18	284	274	288
19	288	278	290
20	292	282	288
21	296	286	290
22	300	290	290
23	304	294	292
24	308	264	288

When `OS_SEARCH_FAST` is set to 0, each extra priority level to traverse requires exactly 4 CPU cycles. When `OS_SEARCH_FAST` is set to 1, each extra priority level to traverse requires exactly 4 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 36 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 11 cycles needed but without the 8 times 7 cycle accumulation. Finally, the third option, when `OS_SEARCH_FAST` is set to 5, delivers a quasi-constant CPU usage as the algorithm utilizes a successive approximation search technique.

The first observation, when looking at this table, is that the third option, when OS_SEARCH_FAST is set to 5, is always less CPU efficient than the second option, the one when OS_SEARCH_FAST is set to 1. So the build option OS_SEARCH_FAST should never be set to 5, as it is the least efficient method. The other observation is that the first option (OS_SEARCH_FAST set to 0) delivers better CPU performance than the second option (OS_SEARCH_FAST set to 1) when the search spans less than around 10 priority levels. So if an application has tasks spanning less than around 10 priority levels, the build option OS_SEARCH_FAST should be set to 0; for all other cases, the build option OS_SEARCH_FAST should be set to 1.

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

6 Chip Support

No chip support is provided with the distribution code because the MPLAB development suite offers an extensive peripheral library that supports everything needed to use the PIC32 peripherals.

7 Measurements

This section gives an overview of the memory requirements and the CPU latency encountered when the RTOS is used on the PIC32 and compiled with MPLAB/GCC. The CPU cycles are exactly the CPU clock cycles, as the processor executes one instruction at every clock transition.

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 memory usage is shown for both types of instructions: the native 32 bit MIPS32 instruction set and the 16 bit (MIPS16e) instruction set (the 16 bit instructions are generated by the compiler by specifying the option `-mips16`).

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

1. Generate Debugging Information: Disable[5]
2. Optimization Level: -Os (Smallest code size)
3. Unroll Loops: Disabled
4. Omit Frame Pointer: Disabled
5. Pre-Optimization Inst Scheduling: (Default for Optimization Level)
6. Post-Optimization Inst Scheduling: (Default for Optimization Level)

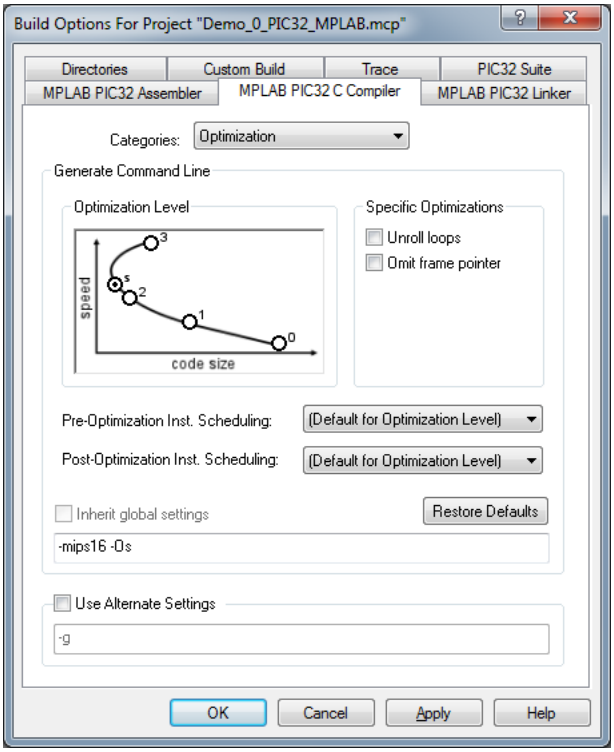

Figure 7-1 Memory Measurement Code Optimization Settings

[5] The debugging option was turned off as the debugging sometimes restricts the optimizer.

Table 7-1 "C" Code Memory Usage

Description	MIPS32	MIPS16e
Minimal Build	< 1200 bytes	< 825 bytes
+ Runtime service creation / static memory	< 1625 bytes	< 1050 bytes
+ Multiple tasks at same priority	< 1725 bytes	< 1150 bytes
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	< 2350 bytes	< 1725 bytes
+ Timer & timeout + Timer call back + Round robin	< 3150 bytes	< 2100 bytes
+ Events + Mailbox	< 4225 bytes	< 3000 bytes
Full Feature Build (no names)	< 5075 bytes	< 3550 bytes
Full Feature Build (no name / no run time creation)	< 4425 bytes	< 3175 bytes
Full Feature Build (no names / no runtime creation) + Timer services module	< 5075 bytes	< 3400 bytes

Table 7-2 Assembly Code Memory Usage

Description		Auto-Clear	ISR Stack
Build for a MIPS32 application	580 bytes	+ 36 bytes	+ 32 bytes
Build for a MIPS16e application	440 bytes	+ 36 bytes	+ 32 bytes

There are two aspects when describing the data memory usage by the RTOS. First, the RTOS needs its own data memory to operate, and second, most of the services offered by the RTOS require data memory for each instance of the service. As the build options affect either the kernel memory needs or the service descriptors (or both), an interactive calculator has been made available on Code Time Technologies web site.

7.2 Latency

Latency of operations has been measured on an Olimex PIC32-WEB development board populated with an 80 MHz PIC32MX460F512L and also on a Microchip PIC32 Starter Kit populated with an 80 MHz PIC32MX360F512L. The code optimization settings that were used for the latency measurements are:

1. Generate Debugging Information: Disable[6]
2. Optimization Level: -O3 (Fastest code)
3. Unroll Loops: Disabled
4. Omit Frame Pointer: Disabled
5. Pre-Optimization Inst Scheduling: (Default for Optimization Level)
6. Pots-Optimization Inst Scheduling: (Default for Optimization Level)

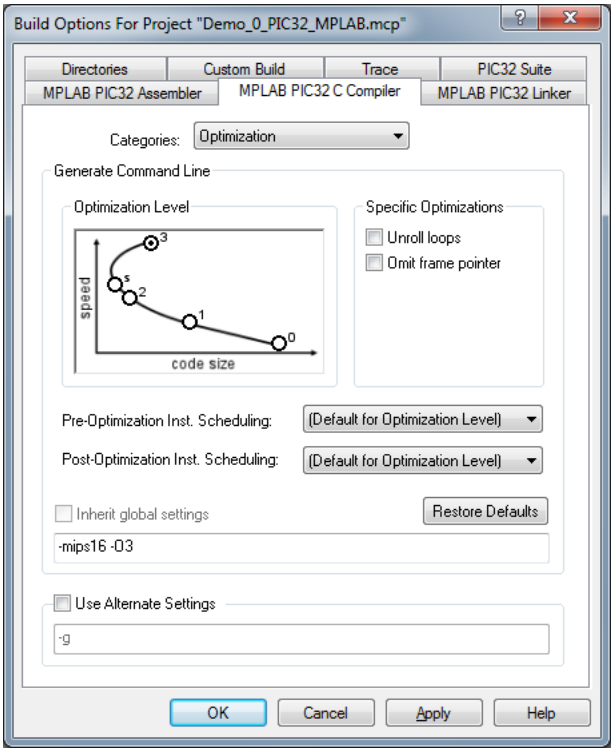

Figure 7-2 Latency Measurement Code Optimization Settings

The processor is configured for maximum performance (cache & wait state settings) using the peripheral library module `SYSTEMConfigPerformance()`.

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.

[6] The debugging option was turned off as the debugging sometimes restricts the optimizer.

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

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

Table 7-3 Measurement without Task Switch

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

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

Table 7-4 Measurement without Blocking

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

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

Table 7-5 Measurement with Task Switch

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

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

The 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 `OSisrIinstall()`, is simply a two line function that uses the appropriate RTOS component followed by a return.

The following two tables list the results obtained, where the cycle count is measured using the timer pair #4-5 with no pre-scaling and with the oscillator post-scaling for the peripheral clock set to 1. This means the timer pair #4-5 counter increments by one at every processor clock transition. As 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. But for this measurement, the timer pair #2-3 is used to trigger the interrupt and measure the elapsed time. The latency measurement includes the cycles required to acknowledge the interrupt.

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 trigger was the timer pair #2-3. The interrupt auto-clearing feature is enabled for the ISR tests but the interrupt stack is disabled. 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

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 for 32 bits instructions

Description	Minimal Features	Full Features
Semaphore posting no task switch	117 (112)	154 (159)

Semaphore waiting no blocking	121 (119)	167 (168)
Semaphore posting with task switch	183 (200)	277 (306)
Semaphore waiting with blocking	195 (190)	317 (307)
Semaphore posting in ISR with task switch	384 (412)	486 (514)
Event setting no task switch	n/a	131 (143)
Event getting no blocking	n/a	171 (172)
Event setting with task switch	n/a	280 (312)
Event getting with blocking	n/a	330 (324)
Event setting in ISR with task switch	n/a	493 (522)
Mailbox writing no task switch	n/a	198 (202)
Mailbox reading no blocking	n/a	215 (217)
Mailbox writing with task switch	n/a	315 (344)
Mailbox reading with blocking	n/a	377 (368)
Mailbox writing in ISR with task switch	n/a	530 (557)
Interrupt Latency	74	74
Interrupt overhead entering the kernel[7]	201 (212)	209 (208)
Interrupt overhead not entering the kernel	137	137
Context switch	44	44

[7] Due to the cache, this number varies +/- 5 cycles

Table 7-8 Latency Measurements for MIPS16e Instructions

Description	Minimal Features	Full Features
Semaphore posting no task switch	145 (154)	248 (256)
Semaphore waiting no blocking	154 (163)	277 (287)
Semaphore posting with task switch	225 (254)	429 (465)
Semaphore waiting with blocking	259 (257)	490 (482)
Semaphore posting in ISR with task switch	460 (495)	684 (716)
Event setting no task switch	n/a	248 (260)
Event getting no blocking	n/a	278 (288)
Event setting with task switch	n/a	465 (505)
Event getting with blocking	n/a	511 (503)
Event setting in ISR with task switch	n/a	713 (751)
Mailbox writing no task switch	n/a	321 (328)
Mailbox reading no blocking	n/a	337 (347)
Mailbox writing with task switch	n/a	482 (520)
Mailbox reading with blocking	n/a	566 (556)
Mailbox writing in ISR with task switch	n/a	745 (779)
Interrupt Latency	63	63
Interrupt overhead entering the kernel[8]	235 (241)	255 (251)
Interrupt overhead not entering the kernel	115	115
Context switch	29	29

[8] Due to the cache, this number varies +/- 4 cycles

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                          */
```