CODE TIME TECHNOLOGIES

# Abassi RTOS

## Porting Document

### 80251 – Keil Compiler

## Disclaimer

# Table of Contents

**1 INTRODUCTION ..... 6**

1.1 DISTRIBUTION CONTENTS ..... 6
1.2 LIMITATIONS ..... 6
1.3 NOTES ..... 7

**2 TARGET SET-UP ..... 8**

2.1 ABOUT THE STACK ..... 10
2.2 START251.A51 MODIFICATIONS ..... 11
2.3 DATA ZEROING / DATA MODELS ..... 11
*2.3.1 XTiny data model ..... 12*
*2.3.2 XSmall data model ..... 12*
*2.3.3 Large data model ..... 13*
*2.3.4 Zeroing changes ..... 13*

**3 INTERRUPTS ..... 14**

3.1 INTERRUPT INSTALLER ..... 15
3.2 2 OR 4 BYTES INTERRUPT CONTEXT ..... 16
3.3 HYBRID INTERRUPT STACK ..... 16
3.4 FAST INTERRUPTS ..... 17

**4 STACK USAGE ..... 18**

**5 SEARCH SET-UP ..... 19**

**6 CHIP SUPPORT ..... 21**

6.1 TIMERS / COUNTERS ..... 21
*6.1.1 TimerInit() ..... 22*
*6.1.2 TIM80251_RLD() ..... 24*
6.2 SERIAL PORT ..... 25
*6.2.1 SerialInit() ..... 26*

**7 MEASUREMENTS ..... 28**

7.1 MEMORY ..... 28
7.2 LATENCY ..... 32

**8 APPENDIX A: BUILD OPTIONS FOR CODE SIZE ..... 41**

8.1 CASE 0: MINIMUM BUILD ..... 41
8.2 CASE 1: + RUNTIME SERVICE CREATION / STATIC MEMORY ..... 42
8.3 CASE 2: + MULTIPLE TASKS AT SAME PRIORITY ..... 43
8.4 CASE 3: + PRIORITY CHANGE / PRIORITY INHERITANCE / FCFS / TASK SUSPEND ..... 44
8.5 CASE 4: + TIMER & TIMEOUT / TIMER CALL BACK / ROUND ROBIN ..... 45
8.6 CASE 5: + EVENTS / MAILBOXES ..... 46
8.7 CASE 6: FULL FEATURE BUILD (NO NAMES) ..... 47
8.8 CASE 7: FULL FEATURE BUILD (NO NAMES / NO RUNTIME CREATION) ..... 48
8.9 CASE 8: FULL BUILD ADDING THE OPTIONAL TIMER SERVICES ..... 49

# List of Figures

FIGURE 2-1 LARGE DATA AND LARGE CODE SET-UP ..... 8
FIGURE 2-2 REENTRANCE SELECTION ..... 9
FIGURE 7-1 MEMORY MEASUREMENT CODE OPTIMIZATION SETTINGS ..... 28
FIGURE 7-2 LATENCY MEASUREMENT CODE OPTIMIZATION SETTINGS ..... 32

# List of Tables

TABLE 1-1 DISTRIBUTION ..... 6
TABLE 2-1 OS_ROM_MODEL ..... 9
TABLE 2-2 EFFECT OF OS_MIN_STACK_USE ON LARGE DATA MODEL ..... 10
TABLE 2-3 START251.A51 ORIGINAL LINES ..... 11
TABLE 2-4 START251.A51 MODIFIED LINES ..... 11
TABLE 2-5 START-UP DATA SECTION ZEROING ..... 12
TABLE 3-1 PRE-ISR DISPATCHER CODE ..... 14
TABLE 3-2 DISTRIBUTION INTERRUPTS ..... 15
TABLE 3-3 INVALIDATING AN ISR HANDLER ..... 15
TABLE 3-4 INTERRUPT SET-UP ..... 16
TABLE 3-5 OS_ISR_STACK ..... 16
TABLE 4-1 CONTEXT SAVE STACK REQUIREMENTS ..... 18
TABLE 5-1 SEARCH ALGORITHM CYCLE COUNT ..... 19
TABLE 6-1 TIMER #0 USED BY THE RTOS ..... 21
TABLE 6-2 TIMER #1 USED BY THE RTOS ..... 21
TABLE 6-3 TIMER #2 USED BY THE RTOS ..... 21
TABLE 6-4 SERIAL PORT SET-UP EXAMPLE ..... 25
TABLE 7-1 "C" CODE MEMORY USAGE (XTINY DATA MODEL) ..... 29
TABLE 7-2 "C" CODE MEMORY USAGE (XSMALL DATA MODEL) ..... 30
TABLE 7-3 "C" CODE MEMORY USAGE (LARGE DATA MODEL) ..... 31
TABLE 7-4 ASSEMBLY CODE MEMORY USAGE ..... 31
TABLE 7-5 MEASUREMENT WITHOUT TASK SWITCH ..... 33
TABLE 7-6 MEASUREMENT WITHOUT BLOCKING ..... 33
TABLE 7-7 MEASUREMENT WITH TASK SWITCH ..... 33
TABLE 7-8 MEASUREMENT WITH TASK UNBLOCKING ..... 34
TABLE 7-9 LATENCY MEASUREMENTS (XTINY DATA / LARGE CODE) ..... 35
TABLE 7-10 LATENCY MEASUREMENTS (XSMALL DATA / LARGE CODE) ..... 36
TABLE 7-11 LATENCY MEASUREMENTS (LARGE DATA / LARGE CODE) ..... 37
TABLE 7-12 LATENCY MEASUREMENTS (XTINY DATA / HUGE CODE) ..... 38
TABLE 7-13 LATENCY MEASUREMENTS (XSMALL DATA / HUGE CODE) ..... 39
TABLE 7-14 LATENCY MEASUREMENTS (LARGE DATA / HUGE CODE) ..... 40
TABLE 8-1: CASE 0 BUILD OPTIONS ..... 41
TABLE 8-2: CASE 1 BUILD OPTIONS ..... 42
TABLE 8-3: CASE 2 BUILD OPTIONS ..... 43
TABLE 8-4: CASE 3 BUILD OPTIONS ..... 44
TABLE 8-5: CASE 4 BUILD OPTIONS ..... 45
TABLE 8-6: CASE 5 BUILD OPTIONS ..... 46
TABLE 8-7: CASE 6 BUILD OPTIONS ..... 47
TABLE 8-8: CASE 7 BUILD OPTIONS ..... 48
TABLE 8-9: CASE 8 BUILD OPTIONS ..... 49

# 1 Introduction

This document details the port of the Abassi RTOS on the 80251 family of microcontrollers. The software suite used for this specific port is the Keil C251 compiler, assembler, linker, and simulator suite, better known as µVision V5.50.

## 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_80251_KEIL.a51` | RTOS assembly file for the 80251 with Keil compiler |
| `sio80251.h` | Include file for the serial port driver |
| `sio80251.c` | Source file for the serial port driver |
| `tim80251.h` | Include file for the timer driver |
| `tim80251.c` | Source file for the timer driver |
| `START251.A51` | Modified standard start-up code |

## 1.2 Limitations

There are a few limitations when using the Abassi RTOS built with the Keil compiler suite for the 80251.

Only the Large and Huge ROM code models are supported. Abassi cannot fit in any of the other code models, as under some build configurations the Abassi kernel single function is larger than 2Kbytes. Also, only the XTiny, XSmall, and Large data memory models are supported, as the amount of data needed in most multi-tasking applications exceeds the amount of the internal RAM available on most 80251 devices; by default, the XTiny and XSmall data memory models use the internal data memory. Finally, the code must be generated as re-entrant.

Contrary to most other ports, for the 80251 `main()`, or the Adam&Eve task, must have stack room allocated to it when the Large data memory model is selected. The reason is the Adam&Eve task cannot use the default "C" start-up stack, which is the internal stack, as the internal stack is used by all tasks. See Section 2.1 for more information about this limitation.

The interrupt hybrid stack feature is supported by this port, but only for the XTiny and XSmall data models. In the case of the Large data model, the internal stack acts exactly as a hybrid stack from the point of view of the task stack usage, therefore the hybrid stack is not made available for the Large data model as it would be redundant.

## 1.3 Notes

Unless the external memory is physically located at base address 0x010000, the use of the Large data model (for both the large and huge code models) should be avoided, as it is the data model that creates the largest code footprint and also delivers the slowest execution. This is already stated in the Keil C251 user guide.

Depending on the RTOS configuration, one or more compiler warnings could be issued when the selected data model is XSmall or Large. The warning(s) is/are one of the followings:

```
Abassi.c (xxx): warning C153: "!=": different spaced pointers (far, xdata)
```

```
Abassi.c (xxx): warning C153: "!=": different spaced pointers (far, edata)
```

The assembly code generated that triggers these warnings has been verified and the Abassi code will operate without problems for all supported data and code models.

# 2 Target Set-up

Very little is needed to set-up the Keil build environment to use the Abassi RTOS in an application. The first thing to do is to set the build to generate code for the desired data model. In the *"Options for Target"* select the *"Target"* tab and set the Memory Model to *"XTiny: near vars & const, ptr-2"*, or to *"XSmall: near vars, far const, ptr-4"*, or to "*Large: xdata vars, far const, ptr-4"*. Then, set the Code ROM Size to either *"Large: 64K program"* or to *"Huge: 64K functions, 16M prog"*. When selecting the XTiny or XSmall data models, the external RAM must be specified in the bottom part of the window in the *"External Memory"* sub-window. A snapshot of the window is shown in Figure 2-1:

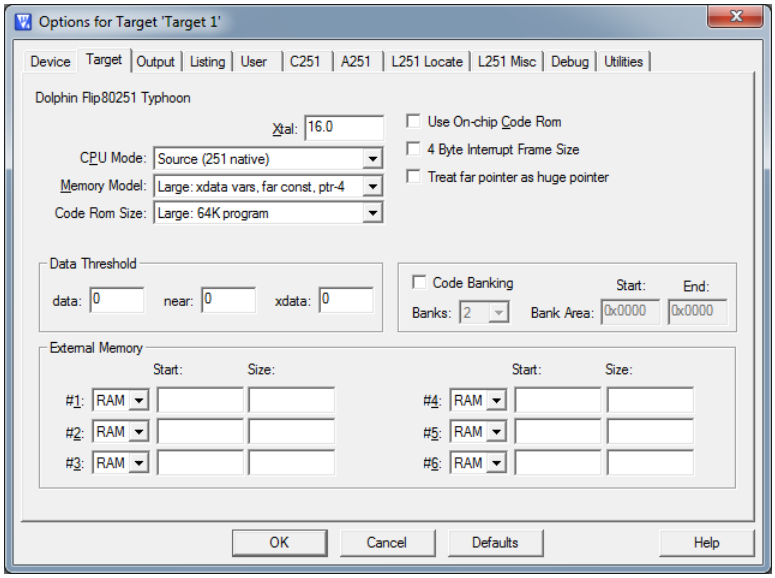


**Figure 2-1 Large Data and Large code set-up**

The second set-up is to disable the data overlay feature of the linker by informing the compiler to generate re-entrant code. If the data overlay is not disabled for an application using the Abassi RTOS, the linker could make multiple parts of the code reuse areas of the data space, and, as the linker is not aware of the multiple tasks in the application, there are possible run-time conflicts where one task cold reuse some of the data area of another one. In the *"Options for Target"* select the *"C251"* tab and select the *"Generate reentrant functions"* button. A snapshot of the window is shown in Figure 2-2 below:

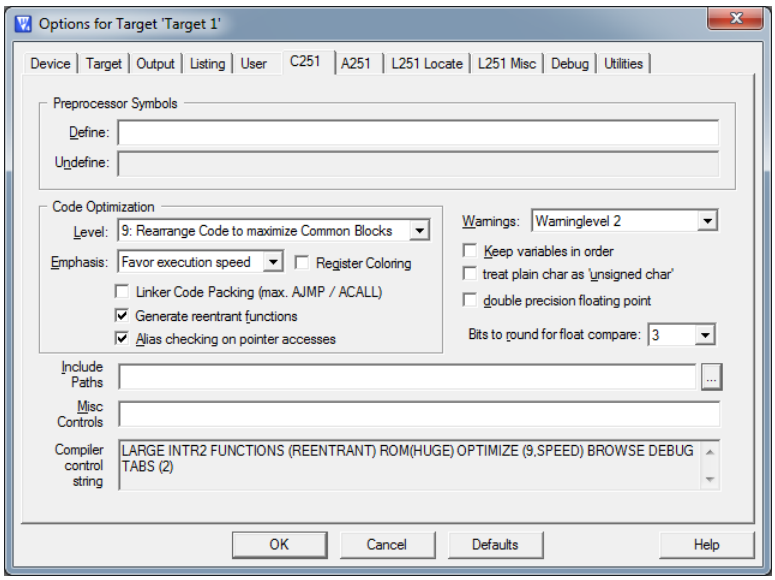


**Figure 2-2 Reentrance selection**

The assembly support file must also be configured for the data model and the code ROM model that is selected for the compiler. In the file `Abassi_80251_KEIL.a51`, at around line 25, there is a definition for `OS_CODE_MODEL` and `OS_DATA_MODEL`, as shown in the following table:

**Table 2-1 `OS_ROM_MODEL`**

```
OS_CODE_MODEL   EQU   0           ; Code model    0:Large  /  1:Huge
OS_DATA_MODEL   EQU   0           ; Data model    0:XTiny  /  1:XSmall  /  2:Large
```

Set the value of `OS_CODE_MODEL` to 0 if the Large Code ROM model is used, or set it to 1 if the Huge Code ROM model is used. If the wrong code model is selected, link-time errors will be issued because the function names will not match Keil's code model naming conventions.

Set the value of `OS_DATA_MODEL` to 0 if the XTiny data model is used, set it to 1 if the XSmall data model is used, or set it to 2 if the Large data model is used. Contrary to the code ROM model, if the wrong data model is selected, link-time errors may not be issued.

NOTE: Some functions in the Keil C runtime library are not multithread-safe. If these functions are only used in one task, then there will be no problems. But if they are used by more than one task, they need to be protected by an Abassi mutex. The preferred way is to re-use the `G_OSmutex` for all non-multithread-safe function, as this will avoid deadlocks.

A final set-up is needed to match the interrupt context save configuration used by Abassi interrupt dispatcher to match the set-up of the device according to the flash off-chip configuration. Refer to section 3.2 for this.

## 2.1 About the stack

When an application is configured for the XTiny or XSmall data model, the context switch of the RTOS operates the same way as all other processor ports. What happens during a context switch for these two data models involves saving the stack pointer in the current task descriptor and restoring it from the descriptor of next task to run. But when the Large data model is selected, there is an important restriction that does not allow the swapping of the stack pointers. The Large data model (when not using the `near` or `far` declarations) handles, by default, the external memory between addresses 0x010000 and 0x01FFFF. The stack pointer is a 16-bit register, therefore it is only able to access memory below address 0x010000. So only the internal memory can be used for the stack, and as the internal memory is too small to hold all the tasks stacks, another approach was required.

For the Large data model, when a context switch occurs, it is necessary to copy the 80251 internal stack into the currently running task's stack and copy the contents from the next to run task's stack back to the internal stack. This means the size of the internal stack, defined in `START251.A51` (look for the token `STACKSIZE`) must be set to a value larger or equal to the largest stack size of all the tasks in the application; it may be larger to handle the interrupt nesting stack requirements as these are not needed to be held in the task stacks. The stack contents copying is not a lightweight operation: simulated on an Intel 8xC251TA, each byte to copy from the internal stack requires on average 14.5 cycles and each byte to copy to the internal stack requires 14 cycles.

As each task must hold its own copy of the internal stack contents, Adam & Eve cannot use the default start stack for itself, as this stack is the internal data area reserved for the stack. To overcome this limitation, the build option `OX_AE_STACK_SIZE` is used to define how much stack room needs to be allocated to Adam&Eve. This option is not part of the standard Abassi build options, but is instead only declared inside the file `Abassi.h`. As supplied, this value is set to 512 bytes, meaning the stack reserved for Adam&Eve has a dimension of 512 bytes[1]; modify the value of this token according to your application (make sure the definition that is modified is right after a `#elif defined(__C251__)` declaration).

NOTE: If the XTiny or the XSmall data models are selected, set the value of `OX_AE_STACK_SIZE` to 0 otherwise the memory reserved for the Adam&Eve stack is simply wasted, as it is not used with these two data models.

The Abassi RTOS allows shrinkage of the amount of stack used by the kernel by setting the build option `OS_MIN_STACK_USE` to a non-zero value. For the Large data model, it is strongly advised to set this build option to non-zero, otherwise the context save and restore (copying) of the stack used by the kernel could represent a large portion of the CPU requirements. For example, a simulation the 8xC251TA part for the search algorithm (table look-up, 1 priority level difference, very small RTOS configuration) delivers the following cycle requirements, and the total stack usage when a context switch occurs:

**Table 2-2 Effect of OS_MIN_STACK_USE on Large Data model**

| `OS_MIN_STACK_USE` **setting** | **Cycles** | **Stack Use** |
|---|---|---|
| Equal to 0 | 4130 | 46 bytes |
| Non-equal to 0 | 3164 | 20 bytes |

[1] 512 bytes is the default value assigned to `STACKSIZE` in the distribution file `START251.A51`.

When the Abassi RTOS is configured for the full features set, the local variables that land on the stack require 62 bytes when the Abassi RTOS is not set for minimum stack usage. When is it set to minimize the stack usage, the local variables stack requirement is reduced to 4 bytes. In the case of the Large data model, it was observed that the CPU impact of not having the kernel local variables on the stack is greatly offset by the saving achieved in the context switch.

NOTE: It was observed that setting the build option `OS_MIN_STACK_USE` to a non-zero value for the XTiny and XSmall data model reduces the code size generated by the compiler and it speeds-up the operation of the RTOS. Therefore the build option `OS_MIN_STACK_USE` should always be set to a non-zero value, no matter the data memory model selected.

## 2.2 START251.A51 Modifications

Abassi needs to be aware of the start address of the internal stack. Keil uses a standard start code that is supplied in the file `START251.A51`, in which the internal stack area is declared but its starting address is not made public. All there is to do is to add a declaration to allow the Abassi RTOS to know where the base of the internal stack is located. You will find the following declaration in the standard `START251.A51` file:

**Table 2-3 START251.A51 original lines**

```
?STACK          SEGMENT   EDATA

                RSEG      ?STACK
                DS        STACKSIZE
```

Add these two lines as indicated:

**Table 2-4 START251.A51 modified lines**

```
?STACK          SEGMENT   EDATA

                RSEG      ?STACK
                PUBLIC    START?STACK
START?STACK:
                DS      STACKSIZE
```

And that's it. The distribution code contains a copy of `START251.A51` file properly modified. If an unmodified standard `START251.A51` file is used, a link-time error will occur as the public symbol `START?STACK` will be missing.

## 2.3 Data Zeroing / Data models

The standard start-up file could be configured to zero the data upon start-up. In the distribution, this feature was not enabled, as the size and addresses of the data RAM are target dependent. As the data is not zeroed, Abassi is configured by default to initialize the internal variables that must be set to zero at startup. The zeroing of the data adds over 130 bytes of code when all the RTOS features are enabled. It is advised to turn on the data zeroing of the data in order to reduce the code size. Table 2-5 lists the available data space. Setting the `?DATALEN` to non-zero will zero the specified number of bytes in the associated section upon at start-up.

**Table 2-5 Start-up data section zeroing**

| Start | Length | Description | Maximun value |
|---|---|---|---|
| n/a | EDATALEN | Internal data | 0FFFFH |
| XDATASTART | XDATALEN | External data @ address 010000H | 0FFFFH |
| HDATASTART | HDATALEN | External data (anywhere) | 100000H |

How the different sections are used according to the data memory model is a bit confusing. In short, the following sub-sections describe what happens with the target RAM set-up. The explanation does not take in account the modification Keil's `HOLD()` declaration modifies. The RAM target set-up is at the bottom of the *"Options for Target"* / *"Target"* tab window. Except for the Large data model, where the memory area between 0x010000 and 0x01FFFF is declared internal by default, all other memory areas must be declared in the *"External Memory"* sub-window of the *"Target"* window (see Section 2).

### 2.3.1 XTiny data model

The XTiny model uses the `EDATA` section for all the non-far declared variables; it is able to access more than 64K of data, but the variables outside the lower 64K (addresses 0x000000 to 0x00FFFF) must be declared `far`. The difference between the XTiny and XSmall data models is that the XTiny data model uses 16 bit data pointers for all variables except the ones declared `far`, and then 24 bit pointers are used. Internally, the compiler and linker never assign more than 64K to the `EDATA` section; all memory defined with addresses between 0x000000 and 0x00FFFF are assigned to the `EDATA` section. For the XTiny data model, the `HDATA` section (that inherits all data memory) overlaps the `EDATA` sections.

As the source code for Abassi is multi-platform, it does not use the `near` or `far` keywords. As such, all variables of Abassi are located in the `EDATA` section, including the task stacks, meaning there are no issues and no required special care to apply when using the XTiny model.

There is some confusion on why choosing the XTiny versus the XSmall data model. The answer is quite simple: in the XTiny data model, if pointers or addresses of variables declared `far` are used as regular pointers, then the upper 8 bits of the 24 bit address are ignored. This means, for example, it is not possible to use any of the "C" standard library functions with a `far` pointer as a function argument; e.g. `strcmp()` or `strcpy()`. The XSmall data model does not have the mismatch issue with the `near` and `far` addresses.

### 2.3.2 XSmall data model

The XSmall model uses the `EDATA` section for all the non-far declared variables; it is able to access more than 64K of data, but the variables outside the lower 64K (addresses 0x000000 to 0x00FFFF) must be declared `far`. The difference between the XTiny and XSmall data models is that the XSmall data model uses 24 bit data pointers for all variables except the ones declared `near`, and then 16 bit pointers are used when appropriate. Internally, the compiler and linker never assign more than 64K to the `EDATA` section; all memory defined with addresses between 0x000000 and 0x00FFFF are assigned to the `EDATA` section. For the XSmall data model, the `HDATA` section (that inherits all data memory) overlaps the `EDATA` sections.

As the source code for Abassi is multi-platform, it does not use the `near` or the `far` keywords. As such, all variables of Abassi are located in the `EDATA` section, including the task stacks, meaning there are no issues and no special care to apply when using the XSmall model.

See the previous section for an explanation on the reason to select the XSmall data model over the XTiny model.

### 2.3.3 Large data model

The Large model uses the `XDATA` section for all the non-`far` / non-`near` declared variables; it is able to access more than 64K of data, but the variables outside the basic 64K (addresses 0x010000 to 0x01FFFF) must be declared either `near` (for the addresses between 0x000000 and 0x00FFFF) or declared `far` (for the whole 16M data address space). The difference between the Large model and the two other ones is the non-`near` / non-`far` declared variables are accessed through the `DPTR` and Keil assigns a value of 1 to the `DPXL` (upper byte of the `DTPR`). In the Large data model, the `XDATA` section cannot be modified; it is always at base address 0x010000 and its length is always 0x10000.

For the Large memory model, there cannot be any issues with the location of the task stacks, as the stack pointer never directly uses these memory areas.

### 2.3.4 Zeroing changes

Once the `START251.A51` is modified to zero the data RAM, then, in the file `Abassi.h`, set the value of the build token `OX_BSS_ZEROED` to 1 (make sure the definition that is modified is right after a `#elif defined(__C251__)` declaration). This later change will remove the code that zeroes Abassi's internal variable at start-up.

# 3 Interrupts

The Abassi RTOS needs to be aware when kernel requests are performed within or outside an interrupt context. Normally an interrupt function is specified in Keil's compiler by adding the postfix "`interrupt K using N`". But for all regular interrupts (this does not apply to the fast interrupts), 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 be aware 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 context save and restore needed for an interrupt.

Out of the box there are provisions for 7 sources of interrupts, as specified by the build option `OS_N_INTERRUPTS` defined in the file `Abassi.h`. 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. The basic 80251 compatible device has 7 sources of interrupts. 1 extra source has been provided as an example when more interrupts need to be handled. This is provided because today's 80251 comes in many variants, and most of the modern ones add one or more extra interrupt source over the legacy device.

The following snippet of code, extracted from the file `Abassi_80251_Keil.a51`, around line 135, shows the code used for the extra interrupt:

**Table 3-1 Pre-ISR dispatcher code**

```
                                    ; ----------------------------------------
ORG    0FF:003BH                    ; For 80251 variants with extra interrupts
ISR_HNDL   7                        ; Add as many of these as it is needed
                                    ; Change the ORG address to the required value
                                    ; Set the 7 for #, and add anothr ISR_JUMP
…
ISR_JUMP   7
```

The first statement, `ORG 0FF:003BH`, informs the linker to locate the code at the physical address `0xFF003B`. If the new interrupt to add uses a different vector, simply replace `0FF:003BH` with the appropriate address. The second statement is the macro `ISR_HNDL` that sets-up the initial part of the pre-ISR dispatch. A second macro is also needed, `ISR_JUMP`, and that one is for the second part of the pre-ISR dispatch; there must always be a pair of `ISR_HNDL` and `ISR_JUMP` for every interrupt number added (non-fast interrupts). In the above example the interrupt number is `7`.

If more than one extra interrupt needs to be added, then all there is to do is to follow these steps:

1. Define and set `OS_N_INTERRUPTS` to the correct value;
2. Insert the two pre-ISR dispatch macros as many times as needed;
3. For each duplicated macros, set the correct address in the `ORG` statement;
4. For each duplicated macro pairs, assign a unique interrupt number (<`OS_N_INTERRUPTS`).

Remember that interrupt numbers start with an index of zero, so that setting `OS_N_INTERRUPTS` to a given value allows for interrupt numbers in the range 0…(`OS_N_INTERRUPTS-1`).

**Table 3-2 Distribution Interrupts**

| Interrupt Number | Interrupt Vector Address | Description |
|---|---|---|
| 0 | 0xFF0003 | External 0 |
| 1 | 0xFF000B | Timer / Counter 0 |
| 2 | 0xFF0013 | External 1 |
| 3 | 0xFF001B | Timer / Counter 1 |
| 4 | 0xFF0023 | Serial Port |
| 5 | 0xFF002B | Timer / Counter 2 |
| 6 | 0xFF0033 | Programmable Counter Array (PCA) |

## 3.1 Interrupt Installer

Attaching a function to an interrupt is quite straightforward. All there is to do is use Abassi's component `ISRinstall()` to specify the interrupt number and the function to be attached to that interrupt number:

```
#include "Abassi.h"

  …
  ISRinstall(Number, &ISRfct);
  Set-up the interrupt
```

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

NOTE: It is a regular function, not one declared with the Keil specific "`interrupt K using N`" postfix statement.

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 `ISRinstall()` component before calling `OSstart()`, this attachment will be removed by `OSstart()`, so `ISRinstall()` must 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 should be set back to `OSinvalidISR()`. This is shown in the next table:

**Table 3-3 Invalidating an ISR handler**

```
#include "Abassi.h"

  …
  ISRinstall(Number, &OSinvalidISR);
  …
```

A better example with a real interrupt initialization function is shown in Section 6.1.

## 3.2 2 or 4 bytes interrupt context

The 80251 can enter an interrupt in two different ways: either pushing 2 bytes or pushing 4 bytes on the internal stack. This configuration is part of the Flash off-chip configuration (bit `INTR` in the `CONFIG1` register), which is not accessible to the application. In `START251.A51`, Keil allows the building process to put the contents of the off-chip configuration registers at addresses `0xFFFFF8` and `0xFFFFF9`, but Abassi does not rely on these values. Instead, the assembly file must be configured to handle interrupts with 2 or 4 bytes. To set-up the interrupt dispatcher to match the number of bytes saved by the processor upon responding to an interrupt, set the following statement in the file `Abassi_80251_Keil.a51`, located around line 30:

**Table 3-4 Interrupt set-up**

```
OS_INTR_TYPE      EQU     2          ; Setting of the INTR bit in CONFIG1 register
                                     ; CONFIG1 is not user accessible, use build time
                                     ; 2: 2 bytes / 4: 4 bytes
```

To use a 2 byte interrupt context save, set `OS_INTR_TYPE` to a value of 2 as shown above. For a 4 byte interrupt context save, set `OS_INTR_TYPE` to a value of 4.

## 3.3 Hybrid Interrupt Stack

It is possible with the XTiny and the XSmall data models, 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 tasks in the application to handle the interrupt nesting (the 80251 can nest up to 4 interrupts). This feature is controlled by the value set by the definition `OS_ISR_STACK`, located around line 35 in the file `Abassi_80251_KEIL.a51`. To disable this feature, set the definition of `OS_ISR_STACK` to a value of zero. To enable it, and specify the hybrid interrupt stack size in bytes, 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 size of 256 bytes is allocated; this is shown in the following table:

**Table 3-5 `OS_ISR_STACK`**

```
OS_ISR_STACK   EQU   256            ; If using a dedicated stack for the nested ISRs
                                    ; 0 if not used, otherwise size of stack in bytes
                                    ; This option is ignored for the Large data model
```

NOTE: The setting of `OS_ISR_STACK` is ignored when the application is configured for the Large data model.

## 3.4 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. If an Abassi component is used in a fast interrupt, the application could misbehave. To set-up a fast interrupt, all there is to do is to comment or remove the two pre-dispatcher macros (`ISR_HNDL` and `ISR_JUMP`) related to the interrupt number to be set as a fast interrupt. This is the snippet of code shown in Table 3-1. The Keil interrupt function to attach to this fast interrupt must be specified by adding the postfix "`interrupt K using N`". Because the Abassi ISR dispatcher is not involved in a 80251 fast interrupt, attaching the interrupt function handler with the component `ISRinstall()` becomes useless.

NOTE: A fast interrupt handler is NOT a regular function, as such, it must be declared with the Keil specific "`interrupt K using N`" postfix statement. Abassi's interrupt dispatcher must not handle fast interrupt.

# 4 Stack Usage

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. For the 80251, the context save contents of a blocked or pre-empted task is completely different from the one used in an interrupt. The following table lists the number of bytes required by each type of context save operation. The size of the interrupt context save does not include the 2 or 4 bytes the processor pushes on the internal stack.

**Table 4-1 Context Save Stack Requirements**

| Description | Context save |
|---|---|
| Blocked/Preempted task context save (excluding the internal stack) (Large Model) | 8 bytes |
| Blocked/Preempted task context save (excluding the internal stack) (Huge Model) | 9 bytes |
| Interrupt dispatcher context save (IRQ stack) (2 bytes interrupt) | 39 bytes |
| Interrupt dispatcher context save (IRQ stack) (4 bytes interrupt) | 41 bytes |

As explained in Section 2.1, the build option `OS_MIN_STACK_USE` should be set to a non-zero value to minimize the amount of stack the kernel uses.

# 5 Search Set-up

The Abassi RTOS build option OS_SEARCH_ALGO 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 second column is when OS_SEARCH_ALGO is set to zero, meaning simple array traversing. The third column, named Look-up, is when OS_SEARCH_ALGO is set to 1, which uses an 8 bit look-up table. Finally, the last column is when OS_SEARCH_ALGO is set to 4 as Keil compiler int are 16 bits, meaning a 16 bit look-up table further searched through successive approximation. The 80251's instruction set is different from the 8051; even though the accumulator is an 8 bit register, the 80251 supports almost all arithmetic operations on 16 bits.

The numbers provided were obtained using the Keil simulator with an Intel 8xC251TA part. The build was set to the XTiny data model and the Large code model, and the option OS_MIN_STACK_USE was set to a non-zero value. It is assumed the other data and code models will deliver the same relative performance across the range of search algorithm. 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 array based search, the number of CPU cycles is constant at 1260 cycles. One must remember the following numbers are dependent on the data and code model. The relative values between the 3 tests are more important that their values as such.

**Table 5-1 Search Algorithm Cycle Count**

| Priority | Linear search | Look-up | Approximation |
|---|---|---|---|
| 1 | 1246 | 1418 | 1830 |
| 2 | 1296 | 1466 | 1842 |
| 3 | 1346 | 1514 | 1890 |
| 4 | 1396 | 1562 | 1866 |
| 5 | 1446 | 1610 | 1914 |
| 6 | 1496 | 1658 | 1926 |
| 7 | 1546 | 1706 | 1974 |
| 8 | 1596 | 1422 | 1914 |
| 10 | 1646 | 1470 | 1962 |
| 11 | 1696 | 1518 | 1974 |
| 12 | 1746 | 1566 | 2022 |

When OS_SEARCH_ALGO is set to 0, each extra priority level to traverse requires 50 CPU cycles. When OS_SEARCH_ALGO is set to 1, each extra priority level to traverse requires 48 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_ALGO to 1 adds around 170 cycles of CPU for the search compared to setting OS_SEARCH_ALGO to zero. But when the next ready to run priority is less than 8, 16, … then there is an about the same number cycles needed, but without the 8 times 48 cycle accumulation. Finally, the third option, when OS_SEARCH_ALGO is set to 3, delivers a quasi-constant CPU usage (around 1930), as the algorithm utilizes a successive approximation search technique; as the 80251 do not have a barrel shifter, the resulting CPU variation is around +/- 100 cycles.

The first observation, looking at this table, is that the first option, when OS_SEARCH_ALGO is set to 0, is the most CPU efficient when the priority span is less than 8. For more than 8 priority spans, the second option (when OS_SEARCH_ALGO is set to 1) becomes more CPU efficient than the first option, but only for the priority span greater than 7. The third option (when OS_SEARCH_ALGO is set to 5) is always less efficient than the second option; therefore the third option should never be selected.

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

There are a multitude of variants for the 80251, so the chip support that is offered with the Abassi RTOS is limited to the peripherals of the original 80251; that is: 3 timers and a single serial port, excluding the PCA (Programmable Counter Array). This is not a full Board Support Package (BSP); it is only a few functions that have been used to port the RTOS on the 80251, and they are made available in the code distribution.

## 6.1 Timers / Counters

The timer driver offers a simple way to program each of the basic timers /counters available in the 80251 microcontroller to generate a periodic interrupt. Special care was taken to compensate the value reloaded in the timer to take into account the time elapsed between the triggering of the interrupt and the set-up for timer 0 and 1. Timer 2 has an automatic reload capability.

For the timer used by the RTOS when either build option `OS_TIMEOUT` or `OS_ROUND_ROBIN` are non-zero, the timer can be set-up with the "C" expression in Table 6-1 and Table 6-2 below. The period (second argument) must be the token `OS_TIMER_US` remapped with the macro `TIM80251_RLD()` as this token is what the RTOS was configured with when built, and the callback function (third argument) must be `TimTick()`, which is the RTOS timer internal maintenance function.

This example installs the interrupt vector and programs timer #0 as the RTOS timer:

**Table 6-1 Timer #0 used by the RTOS**

```
ISRinstall(1, &HWItim80251_0);
TimerInit(0, TIM80251_RLD(OS_TIMER_US), 1, &TimTick, 0);
```

Or if timer #1 is preferred:

**Table 6-2 Timer #1 used by the RTOS**

```
ISRinstall(3, &HWItim80251_1);
TimerInit(1, TIM80251_RLD(OS_TIMER_US), 1, &TimTick, 0);
```

Or if timer #2 is preferred:

**Table 6-3 Timer #2 used by the RTOS**

```
ISRinstall(5, &HWItim80251_2);
TimerInit(2, TIM80251_RLD(OS_TIMER_US), 1, &TimTick, 0);
```

### 6.1.1 TimerInit()

**Synopsis**

```
#include "tim80251.h"

void TimerInit(int TimNmb, int Period, int Prio, void(*Callback)(void),
               int OneShot);
```

**Description**

`TimerInit()` is a utility that programs a timer of a 80251 compatible device to generate either a periodic interrupt or a one time interrupt. The function allows the attachment of a function to call when the interrupt occurs.

**Availability**

Keil 80251 port only.

**Arguments**

| | |
|---|---|
| `TimNmb` | Timer to program.<br>Value must be 0, 1, or 2. |
| `Period` | Desired period (when `OneShot == 0`) or desired elapsed time (when `OneShot != 0`).<br>Specified in timer ticks count. |
| `Prio` | Interrupt priority.<br>Value must be 0 or 1 or 2 or 3. If not, only the 2 LSbits are used. |
| `CallBack` | Function to call when the timer interrupt occurs.<br>`NULL` indicates to not call a function after the interrupt. |
| `OneShot` | When zero, program the timer to generate a single interrupt upon completion.<br>When non-zero, program the timer to generate a periodic interrupt. |

**Returns**

`void`

**Component type**

Function

**Options**

**Notes**

When a timer is used for the serial port, do not program it with `TimerInit()`; `SerialInit()` takes care of initializing the timer it uses.

**See also**

`TIM80251_RLD()` (Section 6.1.2)

`SerialInit()` (Section 6.2.1)

### 6.1.2 TIM80251_RLD()

**Synopsis**

```
#include "tim80251.h"

int TIM80251_RLD(long TimeUS);
```

**Description**

`TIM80251_RLD()` is a utility that converts a time from microseconds into a number of timer ticks.

**Availability**

Keil 80251 port only.

**Arguments**

| | |
|---|---|
| `TimeUS` | Time to convert into number of timer ticks.<br>Specified in μs units. |

**Returns**

| | |
|---|---|
| `int` | Number of timer ticks. |

**Component type**

Macro definition

**Options**

**Notes**

The processor clock must be specified with the token `OS_CPU_FREQ`; the value indicates the processor clock in Hz. This token is defined in `Abassi.h` but can be deleted and added on the compiler command line instead.

**See also**

`TimerInit()` (Section 6.1.1)

## 6.2 Serial Port

As with the timers, the serial port driver provides a simple way to program the serial port. Using the driver allows using the serial port in polling mode or in interrupt mode. When the interrupt mode is selected, an internal circular buffer holds the characters to transmit and the newly received characters. The user does not need to add or set-up anything, except installing the interrupt function handler. The programming of the serial port is always set to 8 data bit, 1 stop bit and no parity.

This example installs the interrupt vector, set the interrupt to priority 0, programs the serial port to use timer 1, sets the baud rate to 19200 bps, and selects interrupt mode.

**Table 6-4 Serial Port set-up example**

```
ISRinstall(4, &HWIsio80251);
SioInit(19200, 1, 1, 0);
```

### 6.2.1 SerialInit()

**Synopsis**

```
#include "sio80251.h"

void SerialInit(int BaudRate, int UseISR, int TimNmb, int Prio);
```

**Description**

`SerialInit()` is a utility that programs the serial port (UART) of the 80251.

**Availability**

Keil 80251 port only.

**Arguments**

| | |
|---|---|
| `BaudRate` | Baud rate to set the serial port to. |
| `UseISR` | Boolean indicating if the serial port operates in polling mode or interrupt mode.<br>Zero is polling; non-zero is ISR. |
| `TimNmb` | Timer to use for the serial port.<br>Value must be 0, 1, or 2. |
| `Prio` | Interrupt priority.<br>Value must be 0 or 1 or 2 or 3. If not, only the 2 LSbits are used |

**Returns**

`void`

**Component type**

Function

**Options**

Including the file `sio80251.c` in the build adds Keil's required `___getchar()`, `___putchar()` I/O function and not so standard `GetKey()` function.

**Notes**

The processor clock must be specified with the token `OS_CPU_FREQ`; the value indicates the processor clock in Hz. This token is defined in `Abassi.h` but can be deleted and added on the compiler command line instead.

Baud rate choices are limited to those that satisfy (`OS_CPU_FREQ/(192*BaudRate)`) such that it yields an integer value (or very close).

When a timer is used for the serial port, it should never be programmed with `TimerInit()`.

When the serial port operates in interrupt mode, an internal circular buffer is used. If the token `OS_SIO_BUF_SIZE` is not defined, a buffer of 16 entries is used. If `OS_SIO_BUF_SIZE` is defined, it must be a power of 2 value.

When using interrupts for the serial port, when the transmit internal buffer is full, it possible to configure the serial port driver to wait for room in the buffer or to drop the request. By default, it is set to wait for room in the buffer. To change this to dropping the request, set the exported variable `G_SerialWait4room` to a value of 0. The data type of `SerialWait4room` is declared in `sio80251.h`.

**See also**

`TimerInit()` (Section 6.1.1)

# 7 Measurements

This section gives an overview of the memory usage and latency when the RTOS is used on the 80251. The CPU cycles are the processor clock cycles; not the "x12" used by the original Intel 8051/8052, where a full instruction cycles requires 12 transitions of the processor clock.

## 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.

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 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 measurement are:

1. Level: 9 (Rearrange Code to maximize Common Blocks)
2. Emphasis: Favor code size

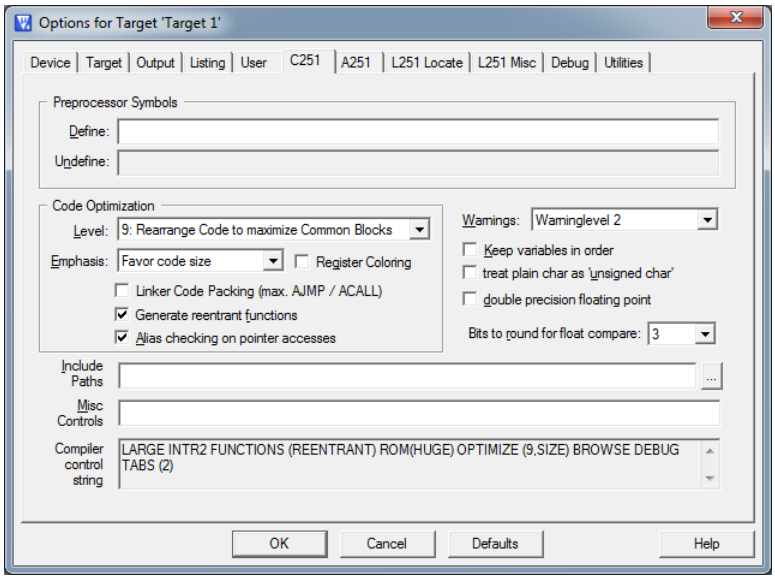


**Figure 7-1 Memory Measurement Code Optimization Settings**

The build option `OS_MIN_STACK_USE` was set to a non-zero value (minimizing the amount of stack used by the kernel) as it is the most real-time efficient for the Large data model and, at the same time, it also delivers the smallest code size for the XTiny and XSmall data models. Also, the data was not zeroed at start-up; some code space saving will be achieved by zeroing the data upon start-up (see Section 2.2).

**Table 7-1 "C" Code Memory Usage (XTiny data model)**

| Description | Large code Size | Huge code size |
|---|---|---|
| Minimal Build | < 1150 bytes | < 1175 bytes |
| + Runtime service creation / static memory | < 1425 bytes | < 1500 bytes |
| + Multiple tasks at same priority | < 1650 bytes | < 1700 bytes |
| + Runtime priority change<br>+ Mutex priority inheritance<br>+ FCFS<br>+ Task suspension | < 2650 bytes | < 2800 bytes |
| + Timer & timeout<br>+ Timer call back<br>+ Round robin | < 3775 bytes | < 3825 bytes |
| + Events<br>+ Mailbox | < 5025 bytes | < 5100 bytes |
| Full Feature Build (no names) | < 6050 bytes | < 6150 bytes |
| Full Feature Build (no names / no runtime creation) | < 5600 bytes | < 5625 bytes |
| Full Feature Build (no names / no runtime creation)<br>+ Timer services module | < 6325 bytes | < 6350 bytes |

**Table 7-2 "C" Code Memory Usage (XSmall data model)**

| Description | Large code Size | Huge code size |
|---|---|---|
| Minimal Build | < 1325 bytes | < 1350 bytes |
| + Runtime service creation / static memory | < 1700 bytes | < 1750 bytes |
| + Multiple tasks at same priority | < 2025 bytes | < 2075 bytes |
| + Runtime priority change<br>+ Mutex priority inheritance<br>+ FCFS<br>+ Task suspension | < 3300 bytes | < 3350 bytes |
| + Timer & timeout<br>+ Timer call back<br>+ Round robin | < 4525 bytes | < 4575 bytes |
| + Events<br>+ Mailbox | < 5925 bytes | < 5975 bytes |
| Full Feature Build (no names) | < 7125 bytes | < 7175 bytes |
| Full Feature Build (no names / no runtime creation) | < 6425 bytes | < 6500 bytes |
| Full Feature Build (no names / no runtime creation)<br>+ Timer services module | < 7300 bytes | < 7375 bytes |

**Table 7-3 "C" Code Memory Usage (Large data model)**

| Description | Large code Size | Huge code size |
|---|---|---|
| Minimal Build | < 1875 bytes | < 1900 bytes |
| + Runtime service creation / static memory | < 2325 bytes | < 2350 bytes |
| + Multiple tasks at same priority | < 2800 bytes | < 2800 bytes |
| + Runtime priority change<br>+ Mutex priority inheritance<br>+ FCFS<br>+ Task suspension | < 4525 bytes | < 4550 bytes |
| + Timer & timeout<br>+ Timer call back<br>+ Round robin | < 6250 bytes | < 6250 bytes |
| + Events<br>+ Mailbox | < 7925 bytes | < 7925 bytes |
| Full Feature Build (no names) | < 9500 bytes | < 9475 bytes |
| Full Feature Build (no names / no runtime creation) | < 8800 bytes | < 8725 bytes |
| Full Feature Build (no names / no runtime creation)<br>+ Timer services module | < 9925 bytes | < 9925 bytes |

**Table 7-4 Assembly Code Memory Usage**

| Code | Data | Description | Code Size |
|---|---|---|---|
| Large | XTiny | 2 bytes ISR | 216 bytes |
| Large | XTiny | 2 Bytes ISR + Hybrid stack | 291 bytes |
| Large | XSmall | 2 bytes ISR | 257 bytes |
| Large | XSmall | 2 Bytes ISR + Hybrid stack | 332 bytes |
| Large | Large | 2 bytes ISR | 364 bytes |
| Huge | XTiny | 2 bytes ISR | 235 bytes |
| Huge | XTiny | 2 Bytes ISR + Hybrid stack | 314 bytes |
| Huge | XSmall | 2 bytes ISR | 284 bytes |
| Huge | XSmall | 2 Bytes ISR + Hybrid stack | 363 bytes |
| Huge | Large | 2 bytes ISR | 394 bytes |
| all | all | 4 Bytes ISR | +25 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 website.

## 7.2 Latency

Latency of operations has been simulated using Intel's 8xC251TA part using Keil's µVision simulator. The 8xC251TA part was retained, because it is the reference part. This means that when manufacturers create improved versions of the 80251, they always refer to the Intel part to as the benchmark to show the performance improvement.

The code optimization settings of the compiler that were used for the latency measurements are:

1. Level: 9 (Rearrange Code to maximize Common Blocks)
2. Emphasis: Favor execution speed

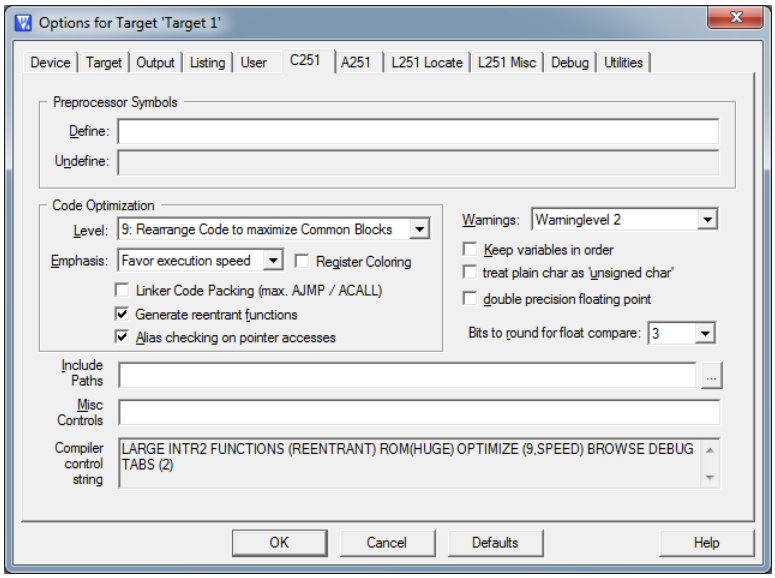


**Figure 7-2 Latency Measurement Code Optimization Settings**

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 7-5 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 7-6 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 7-7 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 7-8 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 7-9 through Table 7-14 list the results obtained, where the cycle count is measured using simulator with an Intel 8xC251TA part, with the code located in RAM. As was the case for the memory measurements, these numbers were obtained with a beta release of this RTOS port. It is possible the released version of the port may have slightly different numbers. The measurements were performed with the hybrid stack disabled and the two byte interrupt context save.

The interrupt latency is the number of cycles elapsed when the interrupt trigger occurred and the ISR function handler is entered after dispatch. 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 Timer / Counter #1 is used to trigger 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” handler function in the `OSisrInstall()`. The interrupt trigger was timer #1. 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, but it includes the call set-up. This measurement is for the best case of task switching where the stack contains only the return address (2 bytes for the large Code ROM model and 3 bytes for the Huge Code ROM model). Interrupt measurement were performed using the 80251 configured with a 2 byte context save.

**Table 7-9 Latency Measurements (XTiny Data / Large Code)**

| Description | Minimal Features | Full Features |
|---|---|---|
| Semaphore posting no task switch | 570 (572) | 868 (872) |
| Semaphore waiting no blocking | 594 (596) | 938 (942) |
| Semaphore posting with task switch | 924 (1084) | 1616 (1756) |
| Semaphore waiting with blocking | 1672 (2744) | 1838 (1794) |
| Semaphore posting in ISR with task switch | 1896 (2046) | 2900 (3022) |
| Event setting no task switch | n/a | 864 (868) |
| Event getting no blocking | n/a | 1120 (1124) |
| Event setting with task switch | n/a | 1780 (1920) |
| Event getting with blocking | n/a | 1972 (1928) |
| Event setting in ISR with task switch | n/a | 2780 (2904) |
| Mailbox writing no task switch | n/a | 1094 (1098) |
| Mailbox reading no blocking | n/a | 1250 (1254) |
| Mailbox writing with task switch | n/a | 2026 (2166) |
| Mailbox reading with blocking | n/a | 2114 (2070) |
| Mailbox writing in ISR with task switch | n/a | 2998 (3114) |
| Interrupt dispatcher latency | 232 | 232 |
| Interrupt overhead entering the kernel | 972 (962) | 1284 (1266) |
| Interrupt overhead NOT entering the kernel | 462 | 462 |
| Context switch | 112 | 112 |

**Table 7-10 Latency Measurements (XSmall Data / Large Code)**

| Description | Minimal Features | Full Features |
|---|---|---|
| Semaphore posting no task switch | 664 (664) | 1002 (1006) |
| Semaphore waiting no blocking | 668 (668) | 1058 (1062) |
| Semaphore posting with task switch | 1236 (1512) | 2128 (2342) |
| Semaphore waiting with blocking | 2094 (2078) | 2354 (2308) |
| Semaphore posting in ISR with task switch | 2602 (2862) | 3558 (3742) |
| Event setting no task switch | n/a | 1004 (1008) |
| Event getting no blocking | n/a | 1286 (1290) |
| Event setting with task switch | n/a | 2322 (2536) |
| Event getting with blocking | n/a | 2516 (2470) |
| Event setting in ISR with task switch | n/a | 3466 (3658) |
| Mailbox writing no task switch | n/a | 1342 (1346) |
| Mailbox reading no blocking | n/a | 1542 (1546) |
| Mailbox writing with task switch | n/a | 2700 (2914) |
| Mailbox reading with blocking | n/a | 2738 (2692) |
| Mailbox writing in ISR with task switch | n/a | 3820 (3992) |
| Interrupt dispatcher latency | 254 | 254 |
| Interrupt overhead entering the kernel | 1366 (1350) | 1430 (1400) |
| Interrupt overhead NOT entering the kernel | 496 | 496 |
| Context switch | 190 | 190 |

**Table 7-11 Latency Measurements (Large Data / Large Code)**

| Description | Minimal Features | Full Features |
|---|---|---|
| Semaphore posting no task switch | 970 (970) | 1336 (1336) |
| Semaphore waiting no blocking | 968 (968) | 1408 (1408) |
| Semaphore posting with task switch | 2338 (2770) | 3428 (3732) |
| Semaphore waiting with blocking | 3628 (3582) | 3846 (3708) |
| Semaphore posting in ISR with task switch | 4480 (4880) | 5628 (5904) |
| Event setting no task switch | n/a | 1354 (1354) |
| Event getting no blocking | n/a | 1748 (1748) |
| Event setting with task switch | n/a | 3686 (3990) |
| Event getting with blocking | n/a | 4072 (3934) |
| Event setting in ISR with task switch | n/a | 5582 (5858) |
| Mailbox writing no task switch | n/a | 1720 (1720) |
| Mailbox reading no blocking | n/a | 1946 (1946) |
| Mailbox writing with task switch | n/a | 4340 (4644) |
| Mailbox reading with blocking | n/a | 4538 (4400) |
| Mailbox writing in ISR with task switch | n/a | 6160 (6382) |
| Interrupt dispatcher latency | 246 | 246 |
| Interrupt overhead entering the kernel | 2142 (2110) | 2200 (2172) |
| Interrupt overhead NOT entering the kernel | 484 | 484 |
| Context switch | 382 | 382 |

**Table 7-12 Latency Measurements (XTiny Data / Huge Code)**

| Description | Minimal Features | Full Features |
|---|---|---|
| Semaphore posting no task switch | 584 (584) | 874 (868) |
| Semaphore waiting no blocking | 606 (606) | 952 (946) |
| Semaphore posting with task switch | 948 (1110) | 1636 (1766) |
| Semaphore waiting with blocking | 1704 (1672) | 1862 (1818) |
| Semaphore posting in ISR with task switch | 2302 (2452) | 3018 (3140) |
| Event setting no task switch | n/a | 878 (872) |
| Event getting no blocking | n/a | 1136 (1130) |
| Event setting with task switch | n/a | 1810 (1936) |
| Event getting with blocking | n/a | 1996 (1952) |
| Event setting in ISR with task switch | n/a | 2862 (2976) |
| Mailbox writing no task switch | n/a | 1110 (1104) |
| Mailbox reading no blocking | n/a | 1278 (1272) |
| Mailbox writing with task switch | n/a | 2050 (2180) |
| Mailbox reading with blocking | n/a | 2150 (2106) |
| Mailbox writing in ISR with task switch | n/a | 3070 (3188) |
| Interrupt dispatcher latency | 262 | 262 |
| Interrupt overhead entering the kernel | 1354 (1342) | 1382 (1374) |
| Interrupt overhead NOT entering the kernel | 498 | 498 |
| Context switch | 122 | 122 |

**Table 7-13 Latency Measurements (XSmall Data / Huge Code)**

| Description | Minimal Features | Full Features |
|---|---|---|
| Semaphore posting no task switch | 672 (672) | 1008 (1002) |
| Semaphore waiting no blocking | 678 (678) | 1072 (1066) |
| Semaphore posting with task switch | 1254 (1530) | 2152 (2348) |
| Semaphore waiting with blocking | 2160 (2110) | 2384 (2326) |
| Semaphore posting in ISR with task switch | 2740 (3000) | 3690 (3868) |
| Event setting no task switch | n/a | 1018 (1012) |
| Event getting no blocking | n/a | 1302 (1296) |
| Event setting with task switch | n/a | 2356 (2548) |
| Event getting with blocking | n/a | 2546 (2488) |
| Event setting in ISR with task switch | n/a | 3562 (3736) |
| Mailbox writing no task switch | n/a | 1366 (1356) |
| Mailbox reading no blocking | n/a | 1578 (1568) |
| Mailbox writing with task switch | n/a | 2742 (2934) |
| Mailbox reading with blocking | n/a | 2780 (2722) |
| Mailbox writing in ISR with task switch | n/a | 3908 (4078) |
| Interrupt dispatcher latency | 290 | 290 |
| Interrupt overhead entering the kernel | 1486 (1470) | 1538 (1520) |
| Interrupt overhead NOT entering the kernel | 534 | 534 |
| Context switch | 198 | 198 |

**Table 7-14 Latency Measurements (Large Data / Huge Code)**

| Description | Minimal Features | Full Features |
|---|---|---|
| Semaphore posting no task switch | 978 (978) | 1328 (1328) |
| Semaphore waiting no blocking | 978 (978) | 1408 (1408) |
| Semaphore posting with task switch | 2442 (2874) | 3512 (3820) |
| Semaphore waiting with blocking | 3736 (3690) | 3948 (3824) |
| Semaphore posting in ISR with task switch | 4788 (5136) | 5872 (6152) |
| Event setting no task switch | n/a | 1352 (1352) |
| Event getting no blocking | n/a | 1750 (1750) |
| Event setting with task switch | n/a | 3778 (4086) |
| Event getting with blocking | n/a | 4174 (4050) |
| Event setting in ISR with task switch | n/a | 5758 (6038) |
| Mailbox writing no task switch | n/a | 1726 (1726) |
| Mailbox reading no blocking | n/a | 1964 (1964) |
| Mailbox writing with task switch | n/a | 4480 (4788) |
| Mailbox reading with blocking | n/a | 4694 (4570) |
| Mailbox writing in ISR with task switch | n/a | 6342 (6622) |
| Interrupt dispatcher latency | 278 | 278 |
| Interrupt overhead entering the kernel | 2346 (2262) | 2360 (2332) |
| Interrupt overhead NOT entering the kernel | 514 | 514 |
| Context switch | 448 | 448 |

# 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_MIN_STACK_USE    1
#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, Adam&Eve = 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_MIN_STACK_USE    1
#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, Adam&Eve = 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_MIN_STACK_USE    1
#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, Adam&Eve = 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_MIN_STACK_USE    1
#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, Adam&Eve = 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_MIN_STACK_USE   1
#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, Adam&Eve = 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_MIN_STACK_USE   1
#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, Adam&Eve = 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_MIN_STACK_USE   1
#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, Adam&Eve = 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_MIN_STACK_USE   1
#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, Adam&Eve = 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_MIN_STACK_USE   1
#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, Adam&Eve = 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                            */
```