CODE TIME TECHNOLOGIES

Abassi RTOS

Porting Document

8051/8052 – Keil Compiler

Table of Contents

List of Figures

List of Tables

1 Introduction

This document details the port of the Abassi RTOS on the 8051/8052 family of microcontrollers. The software suite used for this specific port is the Keil C51 compiler, assembler, linker, and simulator suite, better known as µVision V4.10.

The 8051/8052 family of microcontrollers possesses a CPU architecture that is not very friendly to code generated by compilers; it is the same for applications using multitasking. It has a small internal stack, the external memory accesses are inefficient and the CPU register count is minimal. One should not be surprised to notice the RTOS code size is quite large compared to other processors and the task switching time is in the thousand of cycles instead of the hundreds.

Because the stack size is not large enough for multitasking, the large model is the only target model supported by the RTOS. In this model of the Keil compiler, the processor stack is only used to hold the return addresses for function calls; the function arguments and local variables are all located on an external stack. For the RTOS, each task possesses its own external stack and to mitigate internal stack overflows, the internal stack contents is copied back and forth to the external stack upon task switching. Also, the interrupt context save temporarily uses a small portion of the internal stack, but the whole interrupt context save lands on the external stack before the interrupt handler is called.

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_8051_Keil.a51`	RTOS assembly file for the 8051/8052 with Keil compiler
`sio8051.h`	Include file for the serial port driver
`sio8051.c`	Source file for the serial port driver
`tim8051.h`	Include file for the timer driver
`tim8051.c`	Source file for the timer driver

1.2 Limitations

There are a few limitations when using the RTOS built with the Keil compiler suite for the 8051/8052.

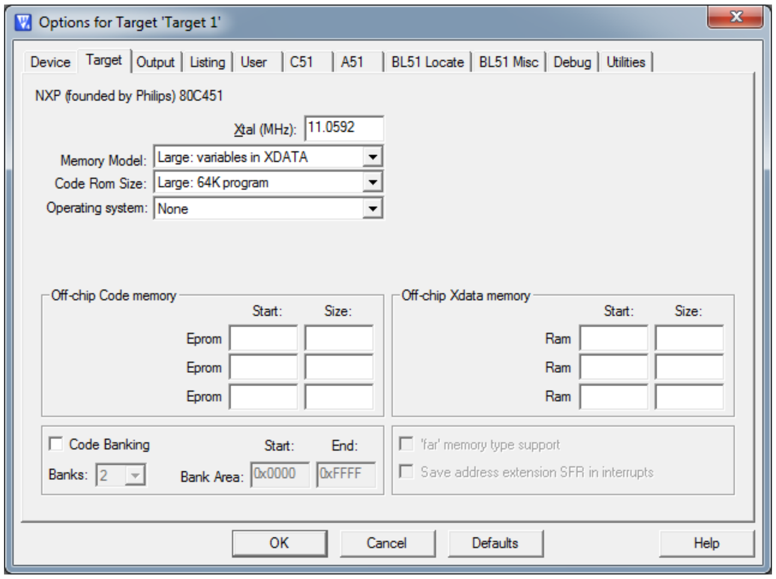
The Idle Task must be created: this is controlled with the build option `OS_IDLE_TASK`, which cannot be set to 0. Error trapping was added to detect if that condition is not fulfilled for this port.

The way the individual task stack frame for the C51 was constructed, `main()` must have some information about the external stack, which sadly the compiler does not make available: the lower memory location of the external stack is needed, but not available. To overcome this limitation, the build option `OS_AE_STACK_SIZE` is used to define how much free space on the external stack is available. This option is not part of the standard RTOS build options; instead it is only declared inside the file `Abassi.h`. As supplied, the RTOS code sets this value in `Abassi.h` to 512 bytes; modify the value of this token according to your application (make sure the definition that is modified is enclosed between both `#ifdef __KEIL__` and `#ifdef __C51__` declarations).

The static descriptors macros (`SEM_STATIC(), MTX_STATIC(), TSK_STATIC()` and `MBX_STATIC()`) are not available for this port.

2 Target Set-up

Very little is needed to set-up the Keil build environment to use the RTOS in an application. The first thing to do is to set the build to generate code for the large model. In the “Options for Target” select the “Target” tab and set the memory model to “Large variables in XDATA”. A snapshot of the window is shown in Figure 2-1 below:



Options for Target 'Target 1'

NXP (founded by Philips) 80C451

Xtal (MHz): 11.0592

Memory Model: Large: variables in XDATA

Code Rom Size: Large: 64K program

Operating system: None

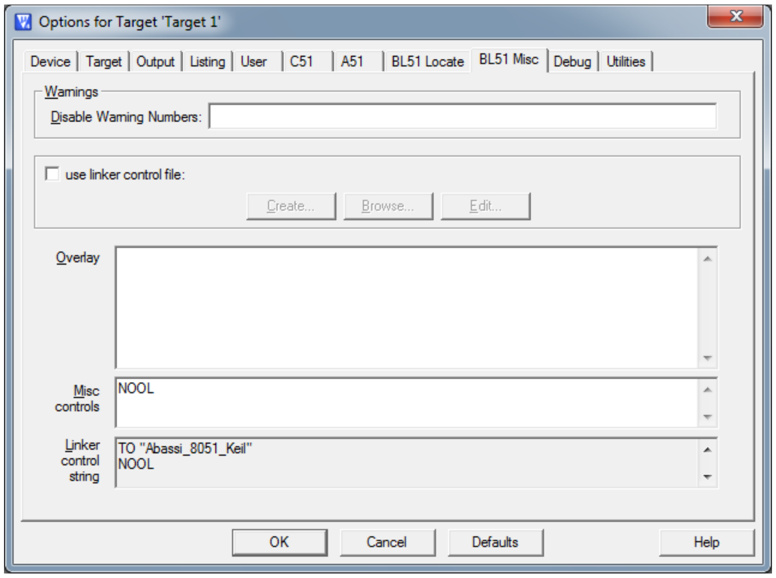
Code Banking: ☐ Banks: 2 Bank Area: Start: 0x0000 End: 0xFFFF

☐ 'far' memory type support

☐ Save address extension SFR in interrupts

Figure 2-1 Large Model set-up

The second set-up is to disable the data overlay feature of the linker. If the data overlay is not disabled for the RTOS, the linker will reuse part of the data space and as the linker is not aware of the multiple tasks in the application, there are run-time conflicts where one task may reuse the data space of another one. In the "Options for Target" select the "BL51 Misc" tab and add "NOOL" in the "Misc Control" window. ". A snapshot of the window is shown in Figure 2-2 below:



Misc controls: NOOL

Linker control string:

```
TO "Abassi_8051_Keil"
NOOL
```

Figure 2-2 Linker Overlay disabling

2.1 STARTUP.A51 Modifications

Keil uses a standard start code that is supplied in the file `STARTUP.A51`. The first thing to change in this file is to set-up the large model environment:

Table 2-1 STARTUP.A51 set-up #1

```
XBPSTACK        EQU     1       ; set to 1 if large reentrant is used.
```

The second change is to add a declaration to allow the RTOS to know where the base of the internal stack is located. You will find the following declaration in `STARTUP.A51`:

Table 2-2 STARTUP.A51 original lines

```
?STACK          SEGMENT   IDATA

                RSEG    ?STACK
                DS      1
```

Add these two lines as indicated:

Table 2-3 STARTUP.A51 modified lines

```
?STACK          SEGMENT   IDATA

                RSEG    ?STACK
               PUBLIC   START?STACK
START?STACK:
                DS      1
```

And that's it.

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 by adding the postfix "`interrupt K using N`". But for all 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`[1]. 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 8051 compatible device has 5 sources of interrupts and the 8052 has 6. 1 extra source has been provided as an example when more interrupts need to be handled. This is provided because today's 8051 comes in so many variants and most of the modern ones add one or more extra interrupts sources over the legacy device.

The dispatcher always uses the same piece of code, replicated for each interrupt vector entry. The following snippet of code extracted from the file `Abassi_8051_Keil.a51`, around line 75, shows the code for the extra interrupt:

Table 3-1 Pre-ISR dispatcher code

```
CSEG    AT 0x0033
push    b
mov     b, #(0x06*TBL_SCALE)
ljmp    ISRdispatch
```

The first statement, `CSEG AT 0x0033`, informs the linker to locate the code at the physical address `0x0033`. If the new interrupt to add uses a different vector, simply replace `0x0033` with the appropriate address. The second statement is the 8051 op-code to preserve the B register. After that, the B register is loaded with the value that corresponds to the interrupt number used by the interrupt installer (`ISRinstall()`). In the above example the interrupt number is `0x06`. Finally, the dispatcher is entered through the `ljmp` statement.

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

1. Set `OS_N_INTERRUPTS` to the correct value;
2. Duplicate the pre-ISR dispatch code as many times as needed;
3. For each duplicated code, set the correct address in the CSEG statement;
4. For each duplicated code, 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).`

[1] When modifying the build option `OS_N_INTERRUPTS` for this port, make sure the modified value is enclosed between both `#ifdef __KEIL__` and `#ifdef __C51__`.

Table 3-2 Distribution Interrupts

Interrupt Number	Interrupt Vector Address	Description
0	`0x0003`	External 0
1	`0x000B`	Timer 0
2	`0x0013`	External 1
3	`0x001B`	Timer 1
4	`0x0023`	Serial Port
5	`0x002B`	Timer 2 (8052)
6	`0x0033`	Extra 0

3.1 Interrupt Installer

Attaching a function to an interrupt is quite straight forward. All there is to do is use the 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. But as a function operating within an interrupt context, it must be re-entrant, therefore the function must be declared with the “`reentrant`” postfix.

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 `ISRintall()` 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 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 5.1.

3.2 Regular Interrupts

The regular interrupts have been implemented to minimize the impact on the internal stack (which stands at 96 bytes or less). When an interrupt is handled by the dispatcher, the whole internal stack contents in use is transferred to the current task’s stack. That frees up the internal stack such that a high level of function nesting is possible in an interrupt.

3.3 Fast Interrupts

Fast interrupts are interrupts that never use components of the RTOS. As they don't use any kernel resources, it is possible to reduce the overhead of the dispatcher and these interrupts are assumed to be small, therefore it was decided to not transfer the contents of the internal stack to the current task's stack, as it is done with regular interrupts. To enable fast interrupts[2], set the following statement in the file `Abassi_8051_Keil.a51`, located around line 30:

Table 3-4 Enabling Fast Interrupts

```
FAST_INTS     EQU     1
```

If fast interrupts are not required then set the following statement instead:

Table 3-5 Disabling Fast Interrupts

```
FAST_INTS     EQU     0
```

Fast interrupts are only supported for the original 6 sources of interrupts. When fast interrupts are enabled, any interrupt in the original 6 which is set to a priority of 1 is a fast interrupt. Any additional interrupts (above the original 6), or one of the original interrupts at priority 0 are regular interrupts, are allowed to use RTOS components.

NOTE: Using a RTOS component in a fast interrupt is almost guaranteed to crash the application. Again, never use a RTOS component inside a fast interrupt.

To enable fast interrupts for interrupts above the original 6, please contact us.

3.4 Nested Interrupts

When fast interrupts are enable, interrupt nesting occurs (priority 1 interrupts can interrupt a priority 0 interrupt). But as this nesting does not involve preempting any of the kernel components, the build option `OS_NESTED_INTS` can be set to 0; that is if no extra (compared to the original 6) interrupts is set to a priority of 1. If one or more extra interrupt sources is set to a priority of 1, and any of these interrupts use one or more RTOS components, then component preemption can happen. Setting `OS_NESTING_INTS` to non-zero when the fast interrupts are enabled will not create any problems; the side effect is a very small extra cycle of CPU needed when using some RTOS components in a regular interrupt.

Considering the above description, the RTOS build option `OS_NESTED_INTS` is always overloaded and forced to non-zero when the RTOS is built for this port. If you are sure that RTOS component preemption in interrupts will never happen, all there is to do is to set it to zero by modifying the statement `#define OS_NESTED_INTS 1` in the section for the KEIL complier and the 8051 target processor in the file `Abassi.h`, as indicated in Footnote 1.

[2] The distribution code has fast interrupts disabled by default.

4 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 second column is when `OS_SEARCH_FAST` is set to zero, meaning 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 4 (Keil compiler `int` are 16 bits, so 2^4), meaning a 16 bit look-up table further searched through successive approximation.

Table 4-1 Search Algorithm Cycle Count

Priority	Linear search	Look-up	Approximation
1	2826	3149	3968
2	2937	3260	3975
3	3048	3371	4080
4	3159	3482	3989
5	3270	3593	4094
6	3381	3704	4101
7	3492	3815	4206
8	3603	3116	4017
9	3714	3227	4122
10	3825	3338	4129
11	3936	3449	4234
12	4047	3560	4143
13	4158	3671	4248
14	4269	3782	4255
15	4380	3893	4360
16	4491	3194	3953
17	4602	3305	4058
18	4713	3416	4065
19	4824	3527	4170
20	4935	3638	4079
21	5046	3749	4184
22	5157	3860	4191
23	5268	3971	4296
24	5379	3272	4107

The third option, when `OS_SEARCH_FAST` is set to 4, never achieves a lower CPU usage than when `OS_SEARCH_FAST` is set to zero or 1. This is understandable as the 8051 does not possess a barrel shifter for variable shift. When `OS_SEARCH_FAST` is set to zero each extra priority level to traverse requires exactly 111 CPU cycle. When `OS_SEARCH_FAST` is set to 1 each extra priority level to traverse also requires exactly 111 CPU cycle except when the priority level is an exact multiple of 8; then there is a sharp reduction of CPU usage. Overall, setting `OS_SEARCH_FAST` to 1 adds an extra 323 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 78 cycles needed but without the 8 times 111 cycles accumulation.

What does this mean? Using 20 or 30 task on the 8051 may be an exception due to the limited code and data memory space, so one could assume the number of tasks will remain small. If that is the case, then `OS_SEARCH_FAST` should be set to 0. If an application is created with 20 or 30 tasks, then setting `OS_SEARCH_FAST` to 1 may be better choice.

5 Chip Support

There are a multitude of variants for the MCS-51, so the chip support that is offered with the Abassi RTOS is limited to the peripherals of the original 8051/8052; that is 2 (8051) or 3 (8052) timers and a single serial port. 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 8051/8052 and they are made available.

5.1 Timers / Counters

The timer driver offers a simple way to program each of the basic timers available in the 8051/8052 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 its own automatic reload capabilities.

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 5-1 and Table 5-2 below. The period (second argument) must be the token `OS_TIMER_US` remapped with the macro `TIM8051_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 5-1 Timer #0 used by the RTOS

```
ISRinstall(1, &HWItim8051_0);
TimerInit(0, TIM8051_RLD(OS_TIMER_US), 1, &TimTick, 0);
```

Or if timer #1 is preferred:

Table 5-2 Timer #1 used by the RTOS

```
ISRinstall(3, &HWItim8051_1);
TimerInit(1, TIM8051_RLD(OS_TIMER_US), 1, &TimTick, 0);
```

Or if timer #2, when on a 8052 compatible device, is preferred:

Table 5-3 Timer #2 used by the RTOS

```
ISRinstall(5, &HWItim8051_2);
TimerInit(2, TIM8051_RLD(OS_TIMER_US), 1, &TimTick, 0);
```

5.1.1 TimerInit()

Synopsis

```
#include "tim8051.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 8051/8052 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 8051/8052 port only.

Arguments

`TimNmb`	Timer to program. Value must be 0 or 1 for 8051 compatible devices, and 0, 1, or 2 for 8052 compatible devices.
`Period`	Desired period (when `OneShot == 0`) or desired elapsed time (when `OneShot != 0`). Specified in timer ticks count.
`Prio`	Interrupt priority. Value must be 0 or 1.
`CallBack`	Function to call when the timer interrupt occurs. `NULL` indicates to not call a function after the interrupt.
`OneShot`	When zero, program the timer to generate a single interrupt upon completion. 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

`TIM8051_RLD()` (Section 5.1.2)

`SerialInit()` (Section 5.2.1)

5.1.2 TIM8051_RLD()

Synopsis

```
#include "tim8051.h"

int TIM8051_RLD(long TimeUS);
```

Description

`TIM8051_RLD()` is a utility that converts a time from microseconds into a number of timer ticks.

Availability

Keil 8051/8052 port only.

Arguments

`TimeUS`	Time to convert into number of timer ticks. 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 5.1.1)

5.2 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 5-4 Serial Port set-up example

```
ISRinstall(4, &HWIsio8051);
SioInit(19200, 1, 1, 0);
```

5.2.1 SerialInit()

Synopsis

```
#include "sio8051.h"

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

Description

`SerialInit()` is a utility that programs the serial port (UART) of the 8051/8052.

Availability

Keil 8051/8052 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. Zero is polling; non-zero is ISR.
`TimNmb`	Timer to use for the serial port. Value must be 0 or 1 for the 8051, and 0, 1, or 2 for the 8052.
`Prio`	Interrupt priority. Value must be 0 or 1.

Returns

`void`

Component type

Function

Options

Including the file `sio8051.c` in the build makes the standard `getchar()`, `putchar()` and not so standard `GetKey()` functions available.

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.

See also

`TimerInit()` (Section 5.1.1)

6 Measurements

This section gives an overview of the memory usage and latency when the RTOS is used on the 8051/8052. The CPU cycles are not the clock cycles; on the original Intel 8051/8052, they are full instruction cycles that require 12 transitions of the processor clock each. On many variants of the 8051/8052 less than 12 transitions are needed for a CPU cycle.

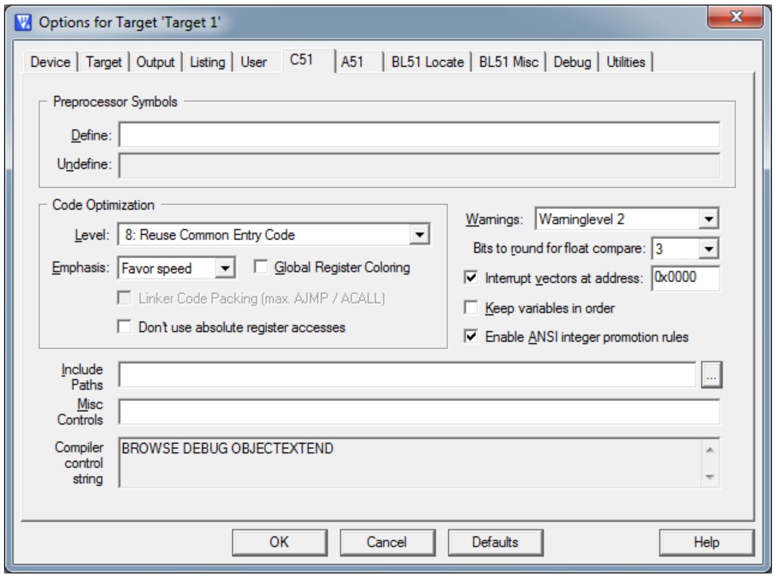
6.1 Memory

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 50 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: 8 (Reuse Common Entry Code)
2. Emphasis: Favor size



Define:

Undefine:

Level: 8: Reuse Common Entry Code

Emphasis: Favor speed

☐ Global Register Coloring

☐ Linker Code Packing (max. AJMP / ACALL)

☐ Don't use absolute register accesses

Warnings: Warninglevel 2

Bits to round for float compare: 3

☑ Interrupt vectors at address: 0x0000

☐ Keep variables in order

☑ Enable ANSI integer promotion rules

Include Paths

Misc Controls

Compiler control string: BROWSE DEBUG OBJECTEXTEND

Figure 6-1 Memory Measurement Code Optimization Settings

Table 6-1 “C” Code Memory Usage

Description	Code Size
Minimal Build	`N/A`
+ Runtime service creation / static memory	`< 3250 bytes`
+ Multiple tasks at same priority	`< 4000 bytes`
+ Runtime priority change + Mutex priority inheritance + FCFS + Task suspension	`< 6050 bytes`
+ Timer & timeout + Timer call back + Round robin	`< 8100 bytes`
+ Events + Mailbox	`< 11000 bytes`
Full Feature Build (no names)	`< 12350 bytes`

Table 6-2 Assembly Code Memory Usage

Description	Code Size
Build without Fast interrupts	`553 bytes`
Build with Fast interrupts	`647 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.

The following table enumerates the data memory required by the RTOS itself, according to the build options:

Table 6-3 Data Memory Usage

Description	Data Size
`Always`	`23 + (OS_MAX_PEND_POST * 9) + (OS_PRIO_MIN * 3) bytes`
`OS_LOGGING_TYPE != 0`	`+ 2  + (OS_MAX_PEND_POST * 2) bytes`
`OS_NAMED_MBX != 0`	`+ 3  + "Semaphore Size" bytes`
`OS_NAMED_SEM != 0`	`+ 3  + "Semaphore Size" bytes`
`OS_NAMED_TASK != 0`	`+ 12 + "Semaphore Size" bytes`
`OS_NAMED_SEM || OS_NAMED_MBX || OS_NAMED_TASK`	`+ 4 bytes`
`OS_ROUND_ROBIN != 0`	`+ 2 bytes`
`OS_TIMEOUT !+ 0`	`+ 5 bytes`
`OS_TIMR_US != 0`	`+ 2 bytes`
`OS_N_INTERRUPTS != 0`	`+ (OS_N_INTERRUPTS * 3) bytes`
`OS_STARVE_WAIT_MAX != 0`	`+ 7 bytes`
`OS_ALLOC_SIZE != 0`	`+ 2 + OS_ALLOC_SIZE bytes`
`OS_STATIC_MBX && OS_RUNTIME_MBX`	`+ 4 + ("Mailbox Size" * OS_STATIC_MBX) ((OS_STATIC_MBX+OS_STATIC_BUF_MAX) * 3) bytes`
`OS_STATIC_SEM && OS_RUNTIME_SEM`	`+ 2 + ("Semaphore Size" * OS_STATIC_SEM) bytes`
`OS_STATIC_TASK != 0`	`+ 2 + ("Task Size" * (OS_STATIC_TASK + 1 + (OS_IDLE_STACK != 0) + OS_AE_STACK_SIZE)) bytes`
`OS_STATIC_NAME != 0`	`+ 2 + OS_STATIC_NAME bytes`
`OS_STATIC_STACK != 0`	`+ 5 + OS_IDLE_STACK bytes`
`OS_SEARCH_FAST == 1`	`+ (OS_PRIO_MIN+7)>>3 bytes`
`OS_SEARCH_FAST == 4`	`+ (OS_PRIO_MIN+15)>>4 bytes`
`OS_IDLE_STACK && !OS_RUNTIME_TASK`	`+ 3 + "Task Size" + OS_IDLE_STACK bytes`
`OS_TIMER_CB != 0`	`+ 2 bytes`
`OS_LOGGING_TYPE == 1`	`+ many strings`

The data memory usage by the services offered by the RTOS has been broken down by descriptor. The first entry in the table is the memory required by the descriptor when none of the optional features of the RTOS are included in the build. Then each entry, until the last one, indicates the number of bytes to add to the descriptor when a feature is included in the build. Finally, the last entry gives the maximum size of the descriptor, when all features are enabled, excluding the data required for the descriptor naming.

Table 6-4 Task Descriptor Memory Usage

Description	Data Size
Always	14 bytes + "Semaphore Size"
OS_TASK_SUSPEND != 0	+ 4 bytes
OS_SAME_PRIO != 0	+ 3 bytes
OS_TIMEOUT != 0	+ 10 bytes
OS_ROUND_ROBIN != 0	+ 2 bytes
OS_ROUND_ROBIN < 0	+ 2 bytes
OS_NAMED_TASK != 0	+ 6 bytes
OS_MTX_INVERSION != 0	+ 2 bytes
OS_TSK_SYSPEND || OS_MTX_DEADLOCK || OS_MTX_INVERSION	+ 3 bytes
OS_USE_TASK_ARG != 0	+ 3 bytes
OS_EVENT != 0	+ 8 bytes
OS_MAILBOX != 0	+ 10 bytes
OS_STARVE_WAIT_MAX != 0	+ 18 bytes
OS_STARVE_WAIT_MAX < 0	+ 2 bytes
OS_STARVE_RUN_MAX < 0	+ 2 bytes
OS_STARVE_PRIO < 0	+ 2 bytes
Largest size / no naming	100 bytes

Table 6-5 Semaphore / Mutex Descriptor Memory Usage

Description	Data Size
Always	5 bytes
OS_IS_FCFS != 0	+ 2 bytes
OS_TSK_SYSPEND || OS_MTX_DEADLOCK || OS_MTX_INVERSION	+ 6 bytes
OS_MTX_INVERSION < 0	+ 2 bytes
OS_NAMED_SEMA != 0	+ 6 bytes
Largest size / no naming	15 bytes

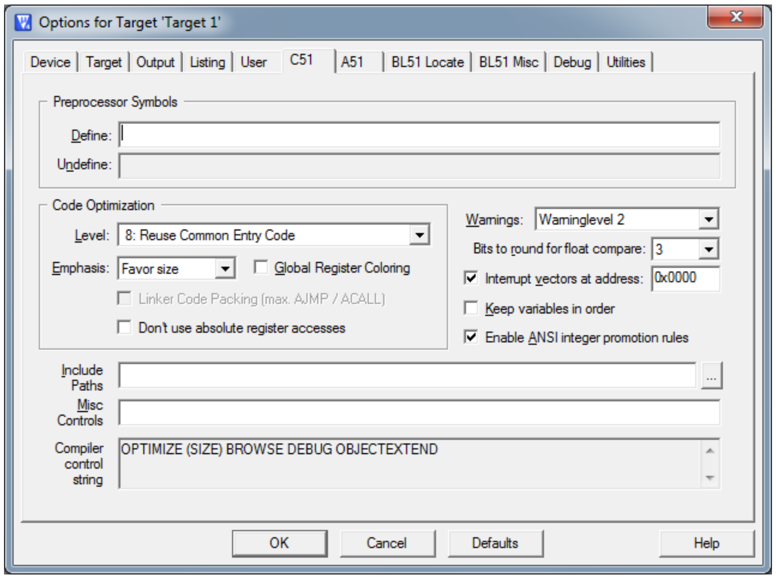
Table 6-6 Mailbox Descriptor Memory Usage

Description	Data Size
Always	(12 + (1 + BufferSize) * 4) + "Semaphore Size" bytes
OS_NAMED_MBX != 0	+ 6 bytes
Largest size / no naming	(27 + ((1 + BufferSize) * 4)) bytes

6.2 Latency

Latency of operations have been measured on a legacy 8052 40 pin DIP based platform using a gated frequency/counter test gear, and confirmed with the Keil simulator. The code optimization settings of the compiler that were used for the latency measurements are:

1.	Level:	8 (Reuse Common Entry Code)
2.	Emphasis:	Favor speed


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C51 | A51 | BL51 Locate | BL51 Misc | Debug | Utilities
Preprocessor Symbols
Define:
Undefine:
Code Optimization
Include Paths
Misc Controls
OK
Cancel
Defaults
Help

Level: 8: Reuse Common Entry Code
Emphasis: Favor size
☐ Global Register Coloring
☐ Linker Code Packing (max. AJMP / ACALL)
☐ Don't use absolute register accesses
Warnings: Warninglevel 2
Bits to round for float compare: 3
☑ Interrupt vectors at address: 0x0000
☐ Keep variables in order
☑ Enable ANSI integer promotion rules
Compiler control string: OPTIMIZE (SIZE) BROWSE DEBUG OBJECTEXTEND

Figure 6-2 Latency Measurement Code Optimization Settings

There are 4 types of latency that are measured, and these 4 measurements are expected to give a 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 4 measurements are performed on a semaphore, the event flags of a task and finally a mailbox. The first 2 latency measurements use the component in a manner where it could unblock a higher priority task blocked on the service. In one case, no task is blocked, in the other a higher priority task is blocked on it. The third measurement involves the opposite, which is a task grabbing the service without getting blocked. Finally, the reaction to unblocking a 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. This means:

Table 6-7 Measurement without Task Switch

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

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

Table 6-8 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 third measurements counts the number of CPU cycles elapsed starting right before the component is used until it is back from the component. This means:

Table 6-9 Measurement without Task Switch

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

The fourth set of measurements counts the number of CPU cycles elapsed starting right at the beginning of the interrupt until the task that was blocked is back from the component used that blocked the task. It is the same as the second set of measurement, except the CPU cycle counting is started at the beginning of the interrupt code, in the processor interrupt vector table. The interrupt handler, attached with `ISRinstall()`, is simply a two line function that uses the appropriate RTOS component with a return.

The following table lists the results obtained, where the cycle count is 1/12 of the CPU clock, as a single CPU cycle requires 12 clock transitions on the legacy 8052 microcontroller.

The interrupt overhead 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 `ISRinstall()`. The interrupt trigger was timer #1.

Table 6-10 Latency Measurements

Description	Minimal Features	Full Features
Semaphore posting no task switch	1262	1895
Semaphore posting with task switch	2308	3764
Semaphore waiting no blocking	1258	1898
Semaphore posting in ISR with task switch	3210	4681
Event setting no task switch	n/a	1955
Event setting with task switch	n/a	4121
Event getting no blocking	n/a	2296
Event setting in ISR with task switch	n/a	5038
Mailbox writing no task switch	n/a	2576
Mailbox writing with task switch	n/a	5478
Mailbox reading no blocking	n/a	2846
Mailbox writing in ISR with task switch	n/a	6430
Interrupt overhead (Build with no Fast Interrupts)	223	223
Interrupt overhead (Build with Fast Interrupts)	256	256
Fast Interrupt overhead	92	92

7 Appendix A: Build Options for Code Size

7.1 Case 0: Minimum build

Table 7-1: Case 0 build options

```
#define OS_ALLOC_SIZE       0     /* When !=0, RTOS supplied OSalloc                 */
#define OS_EVENTS           0     /* If event flags are supported                    */
#define OS_IDLE_STACK       0     /* If IsleTask 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_POST    32    /* Maximum number of sempahores posted in ISRs     */
#define OS_MTX_DEADLOCK     0     /* This test validates this                        */
#define OS_MTX_INVERSION    0     /* To enable protection against priority inversion */
#define OS_NAMED_MBX        0     /* Use named Mailboxes                             */
#define OS_NAMED_SEM        0     /* Use named Semaphores                            */
#define OS_NAMED_TASK       0     /* Use named Tasks                                 */
#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        0     /* Support multiple tasks with the same priority   */
#define OS_ROUND_ROBIN      0     /* Use round-robin, value specifies period in uS   */
#define OS_RUNTIME_MBX      0     /* If can create mailboxes at run time             */
#define OS_RUNTIME_SEM      0     /* If can create semaphores at run time            */
#define OS_RUNTIME_TASK     0     /* If can create tasks at run time                 */
#define OS_SEARCH_FAST      0     /* If using a fast search                          */
#define OS_SEMA_FCFS        0     /* Allow the use of 1st come 1st serve semaphore   */
#define OS_STARVE_PRIO      0     /* Priority threshold for starving protection      */
#define OS_STARVE_RUN_MAX   0     /* Maximum Timer Tick for starving protection      */
#define OS_STARVE_WAIT_MAX  0     /* Maximum time on hold for starving protection    */
#define OS_STATIC_BUF_MBX   0     /* when OS_STATIC_MBOX != 0, # of buffer element   */
#define OS_STATIC_MBX       0     /* If !=0 how many mailboxes                       */
#define OS_STATIC_NAME      0     /* If named mailboxes/semaphore/task, size in char */
#define OS_STATIC_SEM       0     /* If !=0 how many semaphores and mutexes          */
#define OS_STATIC_STACK     0     /* if !=0 number of bytes for all stacks           */
#define OS_STATIC_TASK      0     /* If !=0 how many tasks (excluding A&E and Idle)  */
#define OS_TASK_SUSPEND     0     /* If a task can suspend another one               */
#define OS_TIMEOUT          0     /* !=0 enables blocking timeout                    */
#define OS_TIMER_CB         0     /* !=0 gives the timer callback period             */
#define OS_TIMER_US         0     /* !=0 enables timer & specifies the period in uS  */
#define OS_USE_TASK_ARG     0     /* If tasks have arguments                         */
```

7.2 Case 1: + Runtime service creation / static memory

Table 7-2: Case 1 build options

```
#define OS_ALLOC_SIZE       0     /* When !=0, RTOS supplied OSalloc               */
#define OS_EVENTS           0     /* If event flags are supported                  */
#define OS_IDLE_STACK       0     /* If IsleTask 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_POST    32    /* Maximum number of sempahores posted in ISRs   */
#define OS_MTX_DEADLOCK     0     /* This test validates this                      */
#define OS_MTX_INVERSION    0     /* To enable protection against priority inversion */
#define OS_NAMED_MBX        0     /* Use named Mailboxes                           */
#define OS_NAMED_SEM        0     /* Use named Semaphores                          */
#define OS_NAMED_TASK       0     /* Use named Tasks                               */
#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        0     /* Support multiple tasks with the same priority */
#define OS_ROUND_ROBIN      0     /* Use round-robin, value specifies period in uS */
#define OS_RUNTIME_MBX      0     /* If can create mailboxes at run time           */
#define OS_RUNTIME_SEM      1     /* If can create semaphores at run time          */
#define OS_RUNTIME_TASK     1     /* If can create tasks at run time               */
#define OS_SEARCH_FAST      0     /* If using a fast search                        */
#define OS_SEMA_FCFS        0     /* Allow the use of 1st come 1st serve semaphore */
#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     4096  /* if !=0 number of bytes for all stacks         */
#define OS_STATIC_TASK      5     /* If !=0 how many tasks (excluding A&E and Idle) */
#define OS_TASK_SUSPEND     0     /* If a task can suspend another one             */
#define OS_TIMEOUT          0     /* !=0 enables blocking timeout                  */
#define OS_TIMER_CB         0     /* !=0 gives the timer callback period           */
#define OS_TIMER_US         0     /* !=0 enables timer & specifies the period in uS */
#define OS_USE_TASK_ARG     0     /* If tasks have arguments                       */
```

7.3 Case 2: + Multiple tasks at same priority

Table 7-3: Case 2 build options

```
#define OS_ALLOC_SIZE       0     /* When !=0, RTOS supplied OSalloc                 */
#define OS_EVENTS           0     /* If event flags are supported                    */
#define OS_IDLE_STACK       0     /* If IsleTask 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_POST    32    /* Maximum number of sempahores posted in ISRs     */
#define OS_MTX_DEADLOCK     0     /* This test validates this                        */
#define OS_MTX_INVERSION    0     /* To enable protection against priority inversion */
#define OS_NAMED_MBX        0     /* Use named Mailboxes                             */
#define OS_NAMED_SEM        0     /* Use named Semaphores                            */
#define OS_NAMED_TASK       0     /* Use named Tasks                                 */
#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_MBX      0     /* If can create mailboxes at run time             */
#define OS_RUNTIME_SEM      1     /* If can create semaphores at run time            */
#define OS_RUNTIME_TASK     1     /* If can create tasks at run time                 */
#define OS_SEARCH_FAST      0     /* If using a fast search                          */
#define OS_SEMA_FCFS        0     /* Allow the use of 1st come 1st serve semaphore   */
#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     4096  /* if !=0 number of bytes for all stacks           */
#define OS_STATIC_TASK      5     /* If !=0 how many tasks (excluding A&E and Idle)  */
#define OS_TASK_SUSPEND     0     /* If a task can suspend another one               */
#define OS_TIMEOUT          0     /* !=0 enables blocking timeout                    */
#define OS_TIMER_CB         0     /* !=0 gives the timer callback period             */
#define OS_TIMER_US         0     /* !=0 enables timer & specifies the period in uS  */
#define OS_USE_TASK_ARG     0     /* If tasks have arguments                         */
```

7.4 Case 3: + Priority change / Priority inheritance / FCFS / Task suspend

Table 7-4: Case 3 build options

```
#define OS_ALLOC_SIZE       0     /* When !=0, RTOS supplied OSalloc                */
#define OS_EVENTS           0     /* If event flags are supported                   */
#define OS_IDLE_STACK       0     /* If IsleTask 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_POST    32    /* Maximum number of sempahores posted in ISRs    */
#define OS_MTX_DEADLOCK     0     /* This test validates this                       */
#define OS_MTX_INVERSION    1     /* To enable protection against priority inversion */
#define OS_NAMED_MBX        0     /* Use named Mailboxes                            */
#define OS_NAMED_SEM        0     /* Use named Semaphores                           */
#define OS_NAMED_TASK       0     /* Use named Tasks                                */
#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_MBX      0     /* If can create mailboxes at run time            */
#define OS_RUNTIME_SEM      1     /* If can create semaphores at run time           */
#define OS_RUNTIME_TASK     1     /* If can create tasks at run time                */
#define OS_SEARCH_FAST      0     /* If using a fast search                         */
#define OS_SEMA_FCFS        1     /* Allow the use of 1st come 1st serve semaphore  */
#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     4096  /* 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_US         0     /* !=0 enables timer & specifies the period in uS */
#define OS_USE_TASK_ARG     0     /* If tasks have arguments                        */
```

7.5 Case 4: + Timer & timeout / Timer call back / Round robin

Table 7-5: Case 4 build options

```
#define OS_ALLOC_SIZE       0     /* When !=0, RTOS supplied OSalloc                 */
#define OS_EVENTS           0     /* If event flags are supported                    */
#define OS_IDLE_STACK       0     /* If IsleTask 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_POST    32    /* Maximum number of sempahores posted in ISRs     */
#define OS_MTX_DEADLOCK     0     /* This test validates this                        */
#define OS_MTX_INVERSION    1     /* To enable protection against priority inversion */
#define OS_NAMED_MBX        0     /* Use named Mailboxes                             */
#define OS_NAMED_SEM        0     /* Use named Semaphores                            */
#define OS_NAMED_TASK       0     /* Use named Tasks                                 */
#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_MBX      0     /* If can create mailboxes at run time             */
#define OS_RUNTIME_SEM      1     /* If can create semaphores at run time            */
#define OS_RUNTIME_TASK     1     /* If can create tasks at run time                 */
#define OS_SEARCH_FAST      0     /* If using a fast search                          */
#define OS_SEMA_FCFS        1     /* Allow the use of 1st come 1st serve semaphore   */
#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     4096  /* 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_US         50000 /* !=0 enables timer & specifies the period in uS  */
#define OS_USE_TASK_ARG     0     /* If tasks have arguments                         */
```

7.6 Case 5: + Events / Mailboxes

Table 7-6: Case 5 build options

```
#define OS_ALLOC_SIZE       0     /* When !=0, RTOS supplied OSalloc                */
#define OS_EVENTS           1     /* If event flags are supported                   */
#define OS_IDLE_STACK       0     /* If IsleTask 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_POST    32    /* Maximum number of sempahores posted in ISRs    */
#define OS_MTX_DEADLOCK     0     /* This test validates this                       */
#define OS_MTX_INVERSION    1     /* To enable protection against priority inversion */
#define OS_NAMED_MBX        0     /* Use named Mailboxes                            */
#define OS_NAMED_SEM        0     /* Use named Semaphores                           */
#define OS_NAMED_TASK       0     /* Use named Tasks                                */
#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_MBX      1     /* If can create mailboxes at run time            */
#define OS_RUNTIME_SEM      1     /* If can create semaphores at run time           */
#define OS_RUNTIME_TASK     1     /* If can create tasks at run time                */
#define OS_SEARCH_FAST      0     /* If using a fast search                         */
#define OS_SEMA_FCFS        1     /* Allow the use of 1st come 1st serve semaphore  */
#define OS_STARVE_PRIO      0     /* Priority threshold for starving protection     */
#define OS_STARVE_RUN_MAX   0     /* Maximum Timer Tick for starving protection     */
#define OS_STARVE_WAIT_MAX  0     /* Maximum time on hold for starving protection   */
#define OS_STATIC_BUF_MBX   100   /* when OS_STATIC_MBOX != 0, # of buffer element  */
#define OS_STATIC_MBX       2     /* If !=0 how many mailboxes                      */
#define OS_STATIC_NAME      0     /* If named mailboxes/semaphore/task, size in char */
#define OS_STATIC_SEM       5     /* If !=0 how many semaphores and mutexes         */
#define OS_STATIC_STACK     4096  /* 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_US         50000 /* !=0 enables timer & specifies the period in uS */
#define OS_USE_TASK_ARG     0     /* If tasks have arguments                        */
```

7.7 Case 6: Full feature Build (no names)

Table 7-7: Case 6 build options

```
#define OS_ALLOC_SIZE       0     /* When !=0, RTOS supplied OSalloc                */
#define OS_EVENTS           1     /* If event flags are supported                   */
#define OS_IDLE_STACK       0     /* If IsleTask 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_POST    32    /* Maximum number of sempahores posted in ISRs    */
#define OS_MTX_DEADLOCK     0     /* This test validates this                       */
#define OS_MTX_INVERSION    1     /* To enable protection against priority inversion */
#define OS_NAMED_MBX        0     /* Use named Mailboxes                            */
#define OS_NAMED_SEM        0     /* Use named Semaphores                           */
#define OS_NAMED_TASK       0     /* Use named Tasks                                */
#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_MBX      1     /* If can create mailboxes at run time            */
#define OS_RUNTIME_SEM      1     /* If can create semaphores at run time           */
#define OS_RUNTIME_TASK     1     /* If can create tasks at run time                */
#define OS_SEARCH_FAST      0     /* If using a fast search                         */
#define OS_SEMA_FCFS        1     /* Allow the use of 1st come 1st serve semaphore  */
#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     4096  /* 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_US         50000 /* !=0 enables timer & specifies the period in uS */
#define OS_USE_TASK_ARG     1     /* If tasks have arguments                        */
```