HCC OAL for eCos® User's Guide

Version 1.00

For use with OAL for eCos® versions 2.03 and above

Date: 04-Dec-2014 13:43

Table of Contents

1 System Overview

1.1 Introduction

This guide is for those who want to use the HCC Embedded OS Abstraction Layer (OAL) for their developments in embedded systems which use the eCOS® operating system from eCosCentric Limited.

The HCC OAL is an abstraction of a Real Time Operating System (RTOS). It defines how HCC software requires an RTOS to behave and its API defines the functions it requires. Most HCC systems and modules use one or more components of the OAL.

HCC has ported its OAL to eCOS®, in the process creating "hooks" which call eCOS® functions from the HCC abstractions. Once you unzip the files from the **oal_os_ecos** package into the **oal/os** folder in the source tree, these files automatically call the correct functions.

The OAL API defines functions for handling the following elements:

- Tasks.
- Events – these are used as a signaling mechanism, both between tasks, and from asynchronous sources such as Interrupt Service Routines (ISRs) to tasks.
- Mutexes – these guarantee that, while one task is using a particular resource, no other task can pre-empt it and use the same resource.
- Interrupt Service Routines (ISRs) – in eCOS® ISRs are platform-specific.

1.2 Feature Check

The main features of the module are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It is integrated with the OAL base package.
- It provides a standard interface for HCC tasks.
- It provides a standard interface for HCC mutexes.
- It provides a standard interface for HCC events.

1.3 Packages and Documents

Packages

The table below lists the packages which you need in order to use the OAL:

Package	Description
oal_base	The OAL base package.
oal_os_ecos	The OAL for eCOS® package. Unzip the files from this package into the **oal/os** folder in the source tree.

Documents

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC OS Abstraction Layer (Base) User's Guide

This document describes the base OAL package, defining the standard functions that must be provided by an RTOS. Use this as your reference to global configuration options and the API.

HCC OAL for eCOS® User's Guide

This is this document.

2 Source File List

This section lists and describes all the source code files included in the system. These files follow HCC Embedded's standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file and PSP file.

2.1 Configuration File

The file **src/config/config_oal_os.h** contains configuration options specific to the system. Configure these as required. (Global configuration parameters are controlled by the base package's configuration file.)

2.2 Source Files

These files should only be modified by HCC.

File	Description
src/oal/os/oalp_defs.h	System defines header file.
src/oal/os/oalp_event.c	Event functions source code.
src/oal/os/oalp_event.h	Event functions header file.
src/oal/os/oalp_isr.c	ISR functions source code.
src/oal/os/oalp_isr.h	ISR functions header file.
src/oal/os/oalp_mutex.c	Mutex functions source code.
src/oal/os/oalp_mutex.h	Mutex functions header file.
src/oal/os/oalp_task.c	Task functions source code.
src/oal/os/oalp_task.h	Task functions header file.

2.3 PSP File

The file **src/psp/target/isr/psp_isr.h** provides elements the core code needs to use, depending on the hardware. Modify this file as required for your hardware.

2.4 Version File

The file **src/version/ver_oal_os.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

Note: Systemwide configuration options which allow you to disable certain functions or sets of functions are set in the base package's configuration file. See the *HCC OS Abstraction Layer (Base) User's Guide* for details.

Set the eCos® configuration options in the file **src/config/config_oal_os.h**. This section lists the available configuration options and their default values.

OAL_HIGHEST_PRIORITY, OAL_HIGH_PRIORITY, OAL_NORMAL_PRIORITY, OAL_LOW_PRIORITY, OAL_LOWEST_PRIORITY

By default these are respectively 5, 10, 15, 20, and 25.

OAL_EVENT_FLAG

The event flag to use for user tasks invoking internal functions. For example, this is used when a task calls an internal function that needs to wait for an event.

The value of this flag should be over 0x80 because lower bits might be used by internal tasks. The default is 0x100.

OAL_ISR_COUNT

The maximum number of interrupt sources. The default is 4.

OAL_TICK_RATE_NS

In some eCos® configurations the tick rate is not defined in ns. In these cases, set this option as follows to represent the tick rate in ns:

```
#define  OAL_TICK_RATE_NS ( CYGNUM_HAL_RTC_NUMERATOR / CYGNUM_HAL_RTC_DENOMINATOR )
```

4 Implementation Notes

The RTOS elements are implemented as follows.

Events

There are no rules governing events.

Mutexes

There are no rules governing mutexes.

Tasks

The configuration option OAL_TASK_COUNT sets the maximum number of tasks in HCC modules. Set this to the number of tasks required in all HCC modules.

ISRs

The combination of OAL and platform ISR is used.

Set the configuration option OAL_ISR_COUNT to the number of ISRs required in HCC modules.

The platform ISR is always present for compatibility reasons. It is required for processors where shared interrupts are used; in this case the ISR ID (passed in the *oal_isr_dsc_t* structure) consists of a major and a minor number. The major number is the interrupt vector number identified by CYGNUM_HAL_INTERRUPT_* and the minor number identifies the relevant shared vector.

The file **psp/target/isr/psp_isr.h** must always be present and should contain the following definitions (the example below shows a system where shared interrupts are used):

```
/*
    ** ISR_ID      - ID generated in the config file
    ** ISR_POS_ID  - ID generated by oal_isr_install
    ** ISR_MAJOR   - to decode major ISR number from ISR_ID or ISR_POS_ID
    ** ISR_MINOR   - to decode minor ISR number from ISR_ID or ISR_POS_ID
    ** ISR_POS     - to decode OAL internal position from ISR_POS_ID
*/
#define ISR_ID(major,minor)           (((major)<<8)|(minor))
#define ISR_POS_ID(pos,major,minor)   (((pos)<<16)|((major)<<8)|(minor))
#define ISR_MAJOR(id)                 (((id)>>8)&0xFF)
#define ISR_MINOR(id)                 ((id)&0xFF)
#define ISR_POS(id)                   (((id)>>16)&0xFF)
```

Ticks

There are no rules governing ticks. For configurations where the tick rate is not defined in ns, see the configuration option OAL_TICK_RATE_NS.

5 Functions

The functions in this section must be present.

If the processor does not have shared interrupts, these functions can be replaced by macros as follows:

```
#define psp_isr_install(isr_dsc)    OAL_ERROR
#define psp_isr_delete(id,empty)    OAL_ERROR
#define psp_isr_handler(id)
```

5.1 psp_isr_handler

This creates the main ISR handler called by OAL for all shared interrupts. This should call the appropiate sub-interrupt, based on the *isr_id*.

Format

```
void psp_isr_handler (uint32_t isr_id);
```

Arguments

Parameter	Description	Type
isr_id	The ISR ID containing the major and minor numbers.	uint32_t

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Operation failed.

5.2 psp_isr_install

This function makes a record of the interrupt entry for the specific shared interrupt.

Format

```
int psp_isr_install (const oal_isr_dsc_t * isr_dsc);
```

Arguments

Parameter	Description	Type
isr_dsc	The ISR descriptor structure (see the file **oalp_isr.c**).	oal_isr_dsc_t *

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Function called with a vector that is not shared (or not handled).
OAL_ERR_RESOURCE	System cannot handle any more interrupts.

5.3 psp_isr_delete

This function deletes one shared interrupt.

Format

```
int psp_isr_delete (
    uint32_t    isr_id,
    uint8_t *   empty)
```

Arguments

Parameter	Description	Type
isr_id	The ISR ID containing the major and minor numbers.	uint32_t
empty	On return, this is set to 1 if there are no more sub-interrupts available for the main interrupt.	uint8_t *

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Function called with a vector that is not shared (or not handled).