HCC OAL for ChibiOS User Guide

Version 1.10

For use with OAL for ChibiOS versions 1.01 and above

Exported on 06/07/2018

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

1.1 Introduction

This guide is for those who want to use HCC Embedded's OS Abstraction Layer (OAL) for their developments in embedded systems that use the ChibiOS operating system.

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 Application Programming Interface (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 ChibiOS , in the process creating "hooks" which call ChibiOS functions from the HCC abstractions. Once you unzip the files from the **oal_os_chibios** 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 ChibiOS ISRs are platform-specific.

1.2 Feature Check

The main features of the module are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Integrated with the HCC OS Abstraction Layer (OAL).
- Allows all HCC middleware to run with the ChibiOS RTOS.

1.3 Packages and Documents

Packages

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

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

Documents

For an overview of HCC RTOS software, see Product Information on the main HCC website.

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 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 ChibiOS User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

- To view or download manuals, see OAL PDFs.
- For the history of changes made to the package code itself, see History: oal_os_chibios.

The current version of this manual is 1.10. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.10	2018-06-07	1.01	Corrected text on OAL_EVENT_FLAG configuration option. Added note to *Platform Support Package (PSP) Files*.
1.00	2018-02-12	1.01	First online version.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded 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 files.

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 are in the directory **src/oal/os**. **These files should only be modified by HCC**.

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

2.3 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.

2.4 Platform Support Package (PSP) Files

These files in the directory **src/psp/target/isr** provide functions and elements the core code needs to use, depending on the hardware. Modify these files as required for your hardware.

Note:

- These are PSP implementations for the specific microcontroller and board; you may need to modify these to work with a different microcontroller and/or development board. See PSP Porting for details.
- In the package these files are offset to avoid overwriting an existing implementation. Copy them to the root **hcc** directory for use.

File	Description
psp_isr.c	ISR functions source code.
psp_isr.h	ISR functions header file.
psp_isr_fn.s	ISR functions Visual Studio file.

3 Configuration Options

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

> **Note:** Set systemwide configuration options in the base package's configuration file; these allow you to disable certain functions or sets of functions. See the HCC OS Abstraction Layer (Base) User Guide for details.

OAL_HIGHEST_PRIORITY

The highest priority. By default this is `HIGHPRIO`.

OAL_HIGH_PRIORITY

High priority. By default this is `( NORMALPRIO + ( HIGHPRIO - NORMALPRIO ) / 2 )`.

OAL_NORMAL_PRIORITY

Normal priority. By default this is `NORMALPRIO`.

OAL_LOW_PRIORITY

Low priority. By default this is `( LOWPRIO + ( NORMALPRIO - LOWPRIO ) / 2 )`.

OAL_LOWEST_PRIORITY

The lowest priority. By default this is `LOWPRIO`.

OAL_EVENT_FLAG

OAL_EVENT_FLAG's usage depends on the type of event system an RTOS uses. There are two types:

- Event groups are supported independently of everything else in the system. In this case OAL_EVENT_FLAG does not matter.
- Each event group is directly controlled by a specific task. In this case all HCC stack internal events use the OAL_EVENT_FLAG as the event flag to set on the tasks event group. None of the tasks invoking HCC API calls should use OAL_EVENT_FLAG for signalling an event.

The default is 0x100.

OAL_ISR_COUNT

The maximum number of interrupt sources. The default is 2.

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

There are no rules governing tasks.

ISRs

The platform ISR is used. The implementation depends on the target microcontroller.

The configuration option OAL_ISR_COUNT defines the number of interrupts supported in HCC modules. All ISR handlers require **CH_IRQ_PROLOGUE()** at the beginning and **CH_IRQ_EPILOGUE()** at the end.

The implementation depends on the target microcontroller. The principle is to pre-define <OAL_ISR_COUNT> ISR routines; these save the context, call the real ISR based on the description passed in *oal_isr_dsc,* then restore the context. This way ISR handlers can be dynamically assigned to the wrapper functions.

Ticks

There are no rules governing ticks.

5 PSP Porting

These functions are provided by the Platform Support Package (PSP) to perform various tasks. They are designed for a specific microcontroller and development board. You may need to port them to work with your hardware solution; they are designed to make porting easy.

The package includes samples in the **psp_isr.c** file.

Function	Description
psp_isr_install()	Initializes the ISR.
psp_isr_delete()	Deletes the ISR, releasing the associated resources.
psp_isr_enable()	Enables the ISR.
psp_isr_disable()	Disables the ISR.
psp_int_enable()	Enables global interrupts.
psp_int_disable()	Disables global interrupts.

These functions are described in the following sections.

5.1 psp_isr_install

This function is provided by the PSP to initialize the ISR.

Format

```
int psp_isr_install (
    const oal_isr_dsc_t *   isr_dsc,
    oal_isr_id_t *          isr_id )
```

Arguments

Argument	Description	Type
isr_dsc	The ISR descriptor.	oal_isr_dsc_t *
isr_id	The ISR ID.	oal_isr_id_t *

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Operation failed.

5.2 psp_isr_delete

This function is provided by the PSP to delete the ISR, releasing the associated resources.

Format

```
int psp_isr_delete ( oal_isr_id_t isr_id )
```

Arguments

Argument	Description	Type
isr_id	The ISR ID.	oal_isr_id_t

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Operation failed.

5.3 psp_isr_enable

This function is provided by the PSP to enable the ISR.

Format

```
int psp_isr_enable ( oal_isr_id_t isr_id )
```

Arguments

Argument	Description	Type
isr_id	The ISR ID.	oal_isr_id_t

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Operation failed.

5.4 psp_isr_disable

This function is provided by the PSP to disable the ISR.

Format

```
int psp_isr_disable ( oal_isr_id_t isr_id )
```

Arguments

Argument	Description	Type
isr_id	The ISR ID.	oal_isr_id_t

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Operation failed.

5.5 psp_int_enable

This function is provided by the PSP to enable global interrupts.

Format

```
int psp_int_enable ( void )
```

Arguments

None.

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Operation failed.

5.6 psp_int_disable

This function is provided by the PSP to disable global interrupts.

Format

```
int psp_int_disable ( void )
```

Arguments

None.

Return Values

Return value	Description
OAL_SUCCESS	Successful execution.
OAL_ERROR	Operation failed.