HCC UDP User Guide

Version 2.90

For use with User Datagram Protocol (UDP) module versions 7.01 and above

Date: 09-Oct-2017 15:20

Table of Contents

System Overview 3
Introduction 3
Feature Check 4
Packages and Documents 5
Packages 5
Documents 5
Change History 6
Source File List 7
API Header File 7
Configuration File 7
UDP Source Files 7
Version File 7
Configuration Options 8
Application Programming Interface 9
Functions 10
udp_open 11
udp_close 12
udp_get_buf 13
udp_release_buf 14
udp_receive 15
udp_rx_ready 16
udp_send 17
Error Codes 18
Types and Definitions 19
t_ip_addr 19
t_ip_ntf 19
t_ip_port 19
t_ip_get_buf 19
t_ip_get_buf_tn 20
t_ip_opt 20
Integration 21
PSP Porting 21

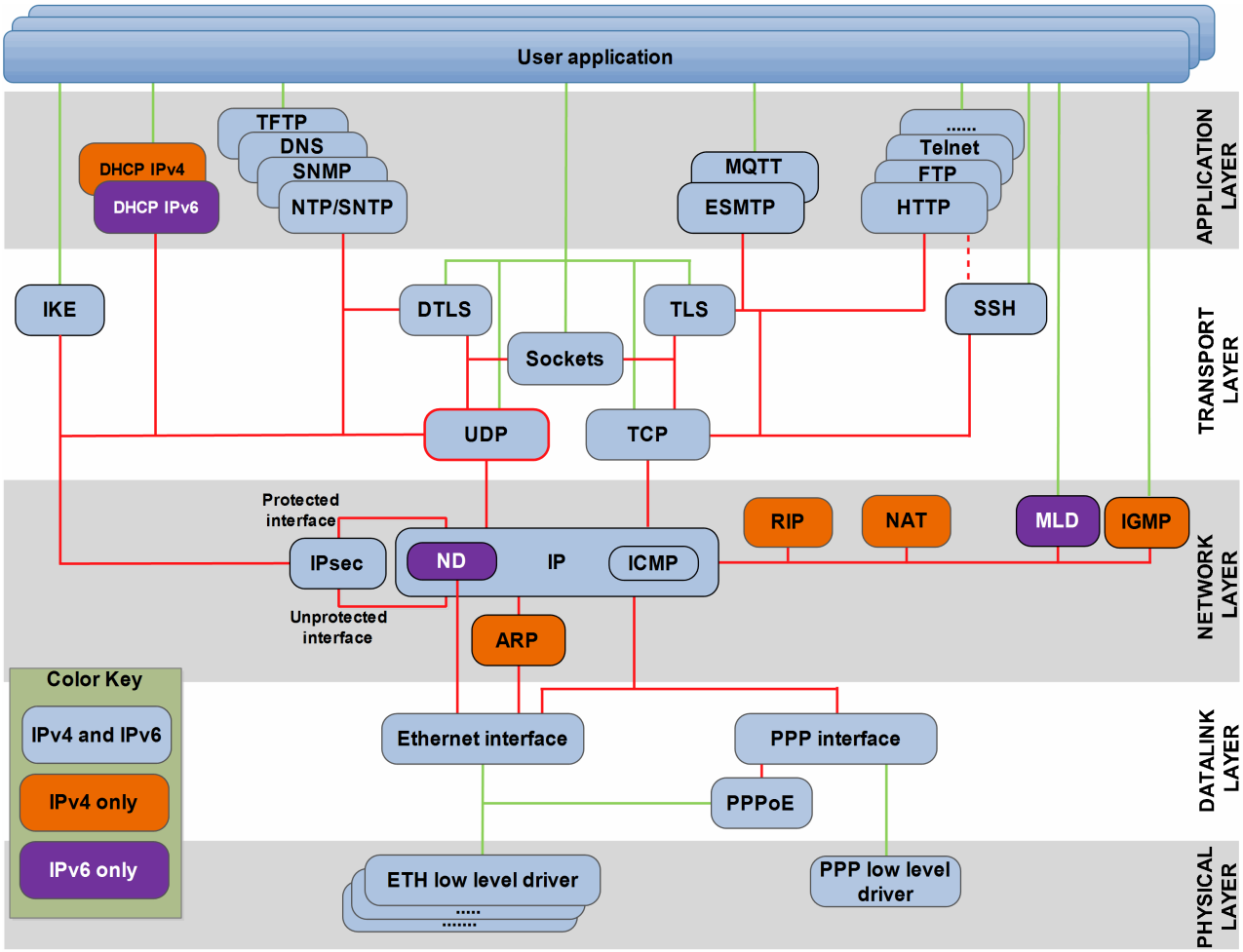
1 System Overview

1.1 Introduction

This guide is for those who want to implement the optional User Datagram Protocol (UDP) as part of HCC Embedded's MISRA-compliant TCP/IP stack. UDP allows applications to send messages called datagrams to other hosts on an Internet Protocol (IP) network without any prior communication to set up special transmission channels or data paths. UDP uses a simple transmission model with a minimum of protocol mechanism.

UDP uses no handshaking, so exposes your program to any unreliability in the underlying network protocol. It is suitable for use where error checking and correction is either unnecessary or performed in the application, avoiding the overhead of such processing at the network interface level. Real time applications often use UDP because dropping packets is preferable to waiting for delayed packets.

The UDP module is part of the HCC TCP/IP stack, as shown below, and is designed specifically for use with it. (In this diagram green lines show interfaces available to users of the stack, red lines show interfaces internal to the TCP/IP system.)



The module gives you two options:

- Employ the native UDP Application Programming Interface (API) to communicate. This manual describes this API.
- Add HCC IP Sockets and use the standard Sockets API for the communications interface.

1.2 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Complies with the HCC MISRA-compliant TCP/IP stack.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compliant with RFC 768.
- Supports zero copy send and receive.
- Provides an optional MISRA-compliant native API.
- Provides an optional Sockets API.

1.3 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module, and also optional modules that may interact with this module, depending on your system's design:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
mip_udp	The HCC UDP package, described in this document, is an optional extension to the HCC TCP/IP suite. No additional packages beyond those included in the base system are required.
mip_base	The TCP/IP Dual Stack base package. This is required.
ip_udp_test	The HCC UDP Test package, a suite of reference code for exercising the UDP interface to the network stack. This provides a test suite for using both the native UDP interface and the Sockets interface.
ip_socket	The HCC Sockets package, an optional extension to the UDP and TCP packages. This enables a standard sockets interface to be used for accessing TCP and UDP ports. If this package is not used, you must use the HCC native UDP API described in this manual for accessing UDP ports.

Documents

For an overview of the HCC TCP/IP stack 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 TCP/IP Dual Stack System User Guide

This is the core document that describes the complete TCP/IP stack, of which UDP is an optional additional component. It covers both IPv4 and IPv6 systems.

HCC UDP User Guide

This is this document.

HCC TCP/IP Sockets Interface User Guide

This document is the API guide for the HCC Sockets package. It describes every function available to you once the system is installed.

1.4 Change History

This section describes past changes to this manual.

- To download earlier manuals, see TCP/IP PDFs.
- For the history of changes made to the package code itself, see History: mip_udp.

The current version of this manual is 2.90. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
2.90	2017-10-09	7.01	Added UDP_ENABLE_SEND_PORT_UNREACHABLE option. Modified **udp_open()**. Improved functions' error reporting tables. Added functions to *PSP Porting*.
2.80	2017-09-05	6.02	Changed *Packages* list.
2.70	2017-06-20	6.02	New *Change History* format.
2.60	2017-03-28	6.02	Updated network diagram.
2.50	2017-01-16	6.01	Updated network diagram.
2.40	2016-03-29	6.01	Added *Integration* section.
2.30	2015-09-04	5.05	Expanded *Introduction*.
2.20	2015-03-31	5.03	Added software change history to manual.
2.10	2014-08-22	5.03	Reorganized *System Overview* section.
2.00	2014-05-15	5.02	First online version.

2 Source File List

The following sections describe 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.

2.1 API Header File

The file **src/api/api_ip_udp.h** must be included by any application using the system. It includes all that is required to access the system. For details of the API functions, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_ip_udp.h** contains the configurable system parameters. Configure these as required. For details of the options, see Configuration Options.

2.3 UDP Source Files

There are two files in the directory **src/ip/stack/udp**. **These files should only be modified by HCC**.

File	Description
udp.c	Implements the UDP protocol.
udp.h	UDP protocol header file.

2.4 Version File

The file **src/version/ver_ip_udp.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

Set the system configuration options in the file **src/config/config_ip_udp.h**.

UDP_SERVER_PORTS

The maximum number of ports that can be opened for listening with the **udp_open()** function. The default setting is 4.

UDP_ENABLE_SEND_PORT_UNREACHABLE

Keep the default of 1 to enable sending of an "ICMP destination unreachable" message if the received message's destination port is not open. Set the value to 0 to disable this.

UDP_RX_CHECKSUM_ENABLE

The default of 1 enables checksum verification for RX packets. Set the value to 0 to disable this.

UDP_TX_CHECKSUM_ENABLE

The default of 1 enables checksum verification for RX packets. Set the value to 0 to disable this.

4 Application Programming Interface

This section describes all the native UDP Application Programming Interface (API) functions. It includes all the functions that are available to an application program.

Note:

- UDP ports can be accessed by using either the native UDP API or the TCP/IP Sockets API. Only use one of these APIs for any individual host system UDP port number.
- The two APIs can be used in parallel so long as they use different UDP ports.

4.1 Functions

The functions are the following:

Function	Description
udp_open()	Opens a UDP server port.
udp_close()	Closes a UDP port.
udp_get_buf()	Gets a UDP protocol buffer from the IP buffer pool.
udp_release_buf()	Releases an IP buffer.
udp_receive()	Retrieves the data on a port that was opened earlier by using **udp_open()**.
udp_rx_ready()	Checks whether there is data available on an opened UDP port.
udp_send()	Sends UDP data to a remote host.

udp_open

Use this function to open a UDP server port.

You can provide a callback function that will be called when an open UDP port either receives data or is closed.

Format

```
t_ip_ret udp_open (
    const uint16_t      port_num,
    t_ip_addr * const   p_ip_addr,
    t_ip_ntf * const    p_ntf,
    t_udp_hdl * const   p_udp_hdl )
```

Arguments

Argument	Description	Type
port_num	The number of the UDP port to open.	uint16_t
p_ip_addr	A pointer to the IP address, or NULL if all IP addresses are accepted.	t_ip_addr *
p_ntf	A pointer to the notification structure.	t_ip_ntf *
p_udp_hdl	Where to store the handle of the allocated UDP port, if the port is opened successfully.	t_udp_hdl *

Return Values

Code	Description
IP_SUCCESS	Successful execution.
IP_ERR_PORT_OPENED	The port is already open.
IP_ERR_NO_MORE_ENTRY	No more entries available (UDP_SERVER_PORTS exceeded).

udp_close

Use this function to close a UDP port.

Format

```
t_ip_ret udp_close ( const t_udp_hdl udp_hdl )
```

Arguments

Argument	Description	Type
udp_hdl	The handle of the UDP port.	t_udp_hdl

Return Values

Code	Description
IP_SUCCESS	Successful execution.
Values returned by function **udp_check_hdl()**.	See the main list of IP Error Codes in the *TCP User Guide*.
Else	See Error Codes.

udp_get_buf

Use this function to get a UDP protocol buffer from the IP buffer pool.

Check the return code to ensure that a buffer was successfully allocated.

In case no buffer is available immediately, you can:

- Specify a timeout to wait for a buffer if none is available. The function waits for this period of time and, if a buffer becomes available during this period, the call returns successfully.
- Specify a notification function to be called if the call fails but a buffer becomes available later. If the timeout expires and you have specified a notification function, the call returns successfully. If a buffer becomes available later, the notification function is called to notify you that you can try requesting a buffer again.

Format

```
t_ip_ret udp_get_buf (
    const t_ip_route_hdl      route_hdl,
    const uint16_t            req_len,
    t_ip_get_buf_tn * const   p_get_buf_tn,
    t_ip_get_buf * const      p_get_buf )
```

Arguments

Argument	Description	Type
route_hdl	The handle of the route for which the buffer will be used.	t_ip_route_hdl
req_len	The requested length of the buffer.	uint16_t
p_get_buf_tn	A pointer to the structure holding the timeout and/or notification.	t_ip_get_buf_tn *
p_get_buf	Where to write the properties of the buffer.	t_ip_get_buf *

Return Values

Code	Description
IP_SUCCESS	Successful execution.
Else	See Error Codes.

udp_release_buf

Use this function to release an IP buffer.

The buffer is released back to the IP buffer pool.

> **Notes:** This function only works for buffers allocated with **udp_get_buf()** or buffer pointers returned by **udp_receive()**.

Format

```
void udp_release_buf ( uint8_t * const p_buf )
```

Arguments

Argument	Description	Type
p_buf	A pointer to the buffer.	uint8_t *

Return Values

None.

udp_receive

Use this function to retrieve the data on a port that was opened earlier by using **udp_open()**.

Notes: After the received data is processed, the buffer must be released by using **udp_release_buf()**. Alternatively, you may reuse the buffer in a subsequent **udp_send()** call.

Format

```
t_ip_ret udp_receive (
    const t_udp_hdl          udp_hdl,
    uint32_t                 timeout,
    t_ip_route_hdl * const   p_route_hdl,
    t_ip_port * const        p_ip_port,
    uint8_t * * const        pp_buf,
    uint16_t * const         p_buf_len )
```

Arguments

Argument	Description	Type
udp_hdl	The handle of the UDP port to read from.	t_udp_hdl
timeout	The time to wait for a packet.	uint32_t
p_route_hdl	Where to write the route handle (optional).	t_ip_route_hdl *
p_ip_port	The IP address and UDP port of the sender.	t_ip_port *
pp_buf	Where to write the pointer of the received data.	uint8_t * *
p_buf_len	Where to write the received data length.	uint16_t *

Return Values

Code	Description
IP_SUCCESS	Successful execution: data is available and is the last IP fragment.
IP_MORE_DATA	Data is available and is not the last IP fragment.
IP_ERR_NO_DATA	There is no data available.
IP_ERR_INVALID_HDL	The handle is invalid.
Values from **udp_check_hdl()**, **ip_clr_get_event()** and **ipbuf_queue_get()**.	See the main list of IP Error Codes in the *TCP User Guide*.

udp_rx_ready

Use this function to check whether there is data available on an opened UDP port.

Format

```
t_ip_ret udp_rx_ready (
    const t_udp_hdl    udp_hdl,
    uint32_t * const   p_len )
```

Arguments

Argument	Description	Type
udp_hdl	The handle of the UDP port.	t_udp_hdl
p_len	Where to write the number of bytes waiting to be read.	uint32_t *

Return Values

Code	Description
IP_SUCCESS	Successful execution.
IP_ERR_NO_DATA	No data is available.
Values returned by function **udp_check_hdl()**.	See the main list of IP Error Codes in the *TCP User Guide*.

udp_send

Use this function to send UDP data to a remote host.

Note:

- *p_buf* must point to a buffer previously allocated with **udp_get_buf()**.
- If the function returns an error, you must release the buffer by using **udp_release_buf()**.

Format

```
t_ip_ret udp_send (
    const t_ip_route_hdl      route_hdl,
    const uint16_t            src_port_num,
    const t_ip_addr * const   p_dst_ip_addr,
    const uint16_t            dst_port_num,
    uint8_t * const           p_buf,
    const uint16_t            buf_len,
    const t_ip_opt * const    p_ip_opt )
```

Arguments

Argument	Description	Type
route_hdl	The route handle.	t_ip_route_hdl
src_port_num	The source port number.	uint16_t
p_dst_ip_addr	A pointer to the destination IP address.	t_ip_addr *
dst_port_num	The destination port number.	uint16_t
p_buf	A pointer to the buffer to send.	uint8_t *
buf_len	The length of the data.	uint16_t
p_ip_opt	A pointer to the IP options (default values are used if this is set to NULL).	t_ip_opt *

Return Values

Code	Description
IP_SUCCESS	Successful execution.
Values returned by **ip_send_pkt()**.	See the main list of IP Error Codes in the *TCP User Guide*.

4.2 Error Codes

If a function executes successfully, it returns with IP_SUCCESS, a value of zero. The following table shows the meaning of the UDP error codes.

> **Note:** Also check error code values in the base system by using the *HCC TCP/IP Dual Stack System User Guide*.

Return Value	Value	Description
IP_SUCCESS	0	Successful execution.
IP_MORE_DATA	1	There is more data pending.
IP_ERR_NO_DATA	10	No data available.
IP_ERR_NO_MORE_ENTRY	11	More tasks are trying to access the stack than the system is configured for. IP_MAX_TASK has been exceeded.
IP_ERR_INVALID_HDL	15	An invalid handle has been used in the requested operation.
IP_ERR_PORT_OPENED	27	The requested port is already open.

All return codes are defined in the TCP/IP base system in the file **api_ip.h** and described in the base system module.

4.3 Types and Definitions

t_ip_addr

The *t_ip_addr* structure stores IPv4 and IPv6 addresses in big-endian mode:

Element	Type	Description
ipa_address[IP_ADDR__MAX_LENGTH]	uint8_t	The IP address.
ipa_version	t_ip_ver	The IP address version, either IPV_IP_V4 or IPV_IP_V6.

t_ip_ntf

t_ip_ntf is the IP notification descriptor structure:

Element	Type	Description	Notes
ntf_fn	t_ip_ntf_fn	The notification function to call.	User variable.
ntf_param	uint32_t	The parameter to send with the notification.	User variable.
p_ntf_next	struct s_ip_ntf *	A pointer to the next entry.	For internal use only.

t_ip_port

t_ip_port is the IP port descriptor structure:

Element	Type	Description
ipp_ip_addr	uint32_t	The IP address of the port.
ipp_port	uint16_t	The port number.

t_ip_get_buf

t_ip_get_buf is the get buffer structure. It contains information on the buffer obtained by using **udp_get_buf()**. It has these two elements:

Element	Type	Description
p_igb_buf	uint8_t *	A pointer to the buffer.
igb_buf_len	uint16_t	The length of the allocated buffer.

t_ip_get_buf_tn

t_ip_get_buf_tn is the IP pool get buffer notification structure. It is used to tell **udp_get_buf()** the requested timeout and/or notification, in case the requested buffer is not available. It has the following elements:

Element	Type	Description
igb_timeout	uint32_t	This timeout tells **udp_get_buf()** how long to wait if the requested buffer is not available. Valid values are IP_WAIT_NONE, IP_WAIT_FOREVER, or the time in milliseconds. This member is only valid if the IP stack is used with an OS. If it is used without an OS, all functions are non-blocking.
p_igb_ntf	t_ip_ntf	A pointer to a **t_ip_ntf** structure that contains the pointer to the notification function and the parameter passed in the notification. The notification function is called if the buffer is not available when a **udp_get_buf()** function fails and, after returning, a buffer becomes free.

t_ip_opt

t_ip_opt is the IP packet options structure:

Element	Type	Description
ipo_ttl	uint8_t	The IP header's TTL field.
ipo_tos	uint8_t	The IP header's TOS field.

5 Integration

This section describes all aspects of the UDP module that require integration with your target project. This includes porting and configuration of external resources.

5.1 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer. For full details of its functions and macros, see the *HCC Base Platform Support Package User Guide*.

The module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_memcmp()	psp_base	psp_string	Compares two blocks of memory.
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.

The module makes use of the following standard PSP macros:

Macro	Package	Element	Description
PSP_RD_BE16	psp_base	psp_endianness	Reads a 16 bit value stored as big-endian from a memory location.
PSP_WR_BE16	psp_base	psp_endianness	Writes a 16 bit value to be stored as big-endian to a memory location.