

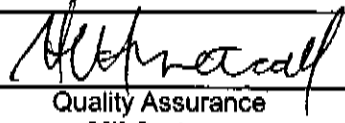


User Manual
for the
4-Channel New Generation
and
8-Channel High-Speed Serial I/O Adapters
VxWorks Software Driver

C²I² Systems Document No.	CCII/HSS8/6-MAN/002
Document Issue	1.6
Issue Date	2015-09-18
Print Date	2015-09-18
File Name	W:\HSS8\TECH\MAN\CH8MAN02.WPD
Distribution List No.	

© C²I² Systems *The copyright of this document is the property of C²I² Systems. The document is issued for the sole purpose for which it is supplied, on the express terms that it may not be copied in whole or part, used by or disclosed to others except as authorised in writing by C²I² Systems.*

Signature Sheet

Name	Signature	Date
Completed by <i>WSE Wynn</i>	 Project Engineer Board Level Products C ² Systems	2015-09-18
Accepted by <i>WSE Wynn</i>	 Project Manager Board Level Products C ² Systems	2015-09-18
Accepted by HCH METCALF	 Quality Assurance C ² Systems	2015-09-18

Amendment History

Issue	Description	Date	ECP No.
0.1	First draft.	2004-07-08	-
1.0	Baselined document.	2004-07-20	-
1.1	Added description for 4 channel version of HSS8 Adapter.	2004-11-25	CCII/HSS8/6-ECP/018
1.2	Implemented ECP, made references to adapter more generic.	2006-06-29	CCII/HSS8/6-ECP/024
1.3	Added description for the hss8Set_pci_base function.	2008-01-28	CCII/HSS8/6-ECP/034
1.4	Improve document naming consistency.	2009-08-18	CCII/HSS8/6-ECP/042
1.5	Added description of POST progress indicator.	2014-02-06	CCII/HSS8/6-ECP/057
1.6	Added procedure to configure pins as outputs on startup.	2015-09-18	CCII/HSS8/6-ECP/059

Contents

1.	Scope	1
1.1	Identification	1
1.2	System Overview	1
1.3	Document Overview	1
2.	Applicable and Reference Documents	2
2.1	Applicable Documents	2
2.2	Reference Documents	2
3.	Software Driver Distribution	3
4.	Installation Procedure	4
4.1	To Build the HSS8 VxWorks Software Driver into the VxWorks Kernel	4
4.2	To Load the HSS8 VxWorks Software Driver Separately	4
5.	Using the HSS8 VxWorks Software Driver	5
5.1	Overview of the HSS8 VxWorks Software Driver	5
5.2	Creating the Device	6
5.3	Configuring the Channels	6
5.4	Adding Call-back Functions	6
5.5	Sending and Receiving Data	8
5.6	Destroying the Device	9
5.7	Detecting an Active Clock Signal on Channels	9
5.8	Obtaining the Current Host and Firmware Version Number	9
5.9	HSS8 Built-in Tests (BIT)	9
5.10	HSS8 Power-On-Self Tests	10
5.11	Return Adapter Type	11
6.	Application Program Interface (API)	12
6.1	General Function Structure	12
6.2	High-Speed Serial VxWorks Software Driver Interface	12
6.2.1	Create Device	13
6.2.2	Destroy Device	14
6.2.3	Set Port Configuration	15
6.2.4	Get Port Configuration	16
6.2.5	Open Port	17
6.2.6	Close Port	18
6.2.7	Send Data	19
6.2.8	Add Call-back	20
6.2.9	Remove Call-back	21
6.2.10	Detecting an Active Clock Signal on Ports	22
6.2.11	Print Out Current Host Software Version Number	23
6.2.12	Print Out Current Embedded Software Version Number	24
6.2.13	HSS8 BIT Structures	25
6.2.14	Enable / Disable POST	28
6.2.15	Return POST Status	29
6.2.16	Return Adapter Type	30
6.2.17	Set PCI Base Address	31
6.2.18	Set Output Pin Values at Startup	32
6.3	Software Driver Data Structures	33
6.3.1	UART Mode	34
6.3.1.1	UART Protocol Information Structure	34
6.3.1.2	UART Protocol Information Structure Members	35

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page iv of vi

- 6.3.2 HDLC Mode 38
 - 6.3.2.1 HDLC Protocol Information Structure 38
 - 6.3.2.2 HDLC Protocol Information Structure Members 39
 - 6.3.2.3 Preamble Requirements 41
- 6.3.3 BISYNC Mode 42
 - 6.3.3.1 BISYNC Protocol Information Structure 42
 - 6.3.3.2 BISYNC Protocol Information Structure Members 43
- 6.3.4 SMC UART Mode 47
 - 6.3.4.1 SMC UART Protocol Information Structure 47
 - 6.3.4.2 SMC UART Protocol Information Structure Members 48
- 7. Getting Started 49**
- 8. Contact Details 50**
 - 8.1 Contact Person 50
 - 8.2 Physical Address 50
 - 8.3 Postal Address 50
 - 8.4 Voice and Electronic Contacts 50
 - 8.5 Product Support 50
- Annexure A 51**
 - Making Changes to sysLib.c for x86 51

Abbreviations and Acronyms

API	Application Program Interface
BCS	Block Check Sequence
BISYNC	Binary Synchronous Communication
BIT	Built-in Test
bit/s	bits per second
BRG	Baud Rate Generator
BSP	Board Support Package
CD	Carrier Detect
CRC	Cyclic Redundancy Check
CTS	Clear to Send
DLE	Data Link Escape
DPLL	Digital Phase-Locked Loop
EEPROM	Electrically Erasable Programmable Read Only Memory
FIFO	First In First Out
HDLC	High Level Data Link Control
HSS8	8-Channel High-Speed Serial
I/O	Input / Output
LED	Light Emitting Diode
MHz	MegaHertz
NRZ	Non-Return-to-Zero
NRZI	Non-Return-to-Zero-Inverted
PC	Personal Computer
PCI	Peripheral Component Interconnect
PMC	Peripheral Component Interconnect Mezzanine Card
POST	Power-On-Self Test
RAM	Random Access Memory
RTS	Request to Send
RxD	Receive Data
SBC	Single Board Computer
SCC	Serial Communications Controller
SDLC	Synchronous Data Link C
SMC	Serial Management Controller
SYNC	Synchronisation
TxD	Transmit Data
UART	Universal Asynchronous Receiver/Transmitter

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page vi of vi

1. **Scope**

1.1 Identification

This document is the user manual for the VxWorks Software Driver for the 8-Channel High-Speed Serial (HSS8) Adapter and the 4-Channel High-Speed Serial Adapter (HSS4NG). The 4-Channel Adapter is based on a stripped down HSS8 Adapter and as such this manual applies, except that only SCC Channels A - D and SMC Channels I - J will be available.

1.2 System Overview

The HSS8 Adapter provides eight channels of simultaneous, high-speed, bi-directional serial communications and an additional four channels of lower-speed serial communications (available only on the frontpanel HSS8 Adapter). The eight high-speed channels are jumper configurable (on a per channel basis) for RS-232 or RS-422/485 drivers while the lower-speed channels have RS-232 drivers only.

The HSS8 VxWorks Software Driver is a low level, device-dependant, interface for transferring data over a C²I² Systems HSS8 Adapter . The HSS8 VxWorks Software Driver binaries are provided with explicit installation instructions.

1.3 Document Overview

This document gives an overview of the HSS8 VxWorks Software Driver installation procedure and its Application Program Interface (API).

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 1 of 51

2. **Applicable and Reference Documents**

2.1 Applicable Documents

2.1.1 Motorola, *MPC8260 PowerQUICC II Family Reference Manual*, MPC8260UM/D Rev. 1, dated May 2003.

2.1.2 CCII/HSS8/6-MAN/001, *Hardware Reference Manual for the 4-Channel New Generation and 8-Channel High-Speed Serial I/O Adapters*.

2.2 Reference Documents

None.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 2 of 51

3. Software Driver Distribution

The VxWorks Software Driver distribution consists of (at least) the following files :

<p>ccHss8Lib<arch>V<version><long>.a</p>	<p>Host-architecture specific, driver object file :</p> <p><i>cc</i> - CCI Systems (Pty) Ltd</p> <p><i>Hss8Lib</i> - 8-Channel High-Speed Serial VxWorks Software Driver</p> <p><arch> - Host for which the binary is built :</p> <ul style="list-style-type: none"> • X86 • PPC <p><version> - Software version is a 3 digit integer :</p> <ul style="list-style-type: none"> • 1st digit : version number • 2nd digit : revision number • 3rd digit : beta number <p><long> - VxWorks Software Driver compiled with -mlongcall flag (only for a PPC host)</p> <p>e.g. "ccHss8LibPPCV100.a" for version 1.0.0 of the HSS8 software, built for a PPC processor.</p>
<p>ccHss8EmbV<version>.hex</p>	<p>HSS8 firmware.</p>
<p>ccHss8Flash<arch>V<version><long>.a</p>	<p>Flash update driver.</p>
<p>hss8Readme.txt</p>	<p>General information and installation notes.</p>
<p>hss8Release_emb.txt, hss8Release_host.txt</p>	<p>Release notes and revision history :</p> <p>Please check this file for information on the latest updates.</p>
<p>ccHss8HfilesV<version>.zip</p>	<p>Zip file which contains all header files that define the Application Program Interface (API) to the HSS8 VxWorks Software Driver.</p>
<p>ccHss8Test.c and ccHss8Test<arch><long>.a</p>	<p>Sample C code for accessing the HSS8 VxWorks Software Driver.</p>
<p>hss8Changes.txt</p>	<p>Changes to be made to VxWorks and Board Support Package (BSP) files.</p>
<p>hss8Flash.txt</p>	<p>Procedure for updating the firmware if required.</p>
<p>hss8Test.txt</p>	<p>Test procedure for verifying host software driver and firmware.</p>

4. **Installation Procedure**

This paragraph describes the installation procedure for the HSS8 VxWorks Software Driver. (The examples given are for a PowerPC host.)

4.1 To Build the HSS8 VxWorks Software Driver into the VxWorks Kernel

Assume the BSP directory is given as : BSP_DIR = /tornado/target/config/dy4181.

- Copy ccHss8LibPPCV<version>.a to your \$(BSP_DIR)/lib directory as ccHss8.a.
- In the Builds section of the Project Workspace, change the Kernel properties to include the ccHss8.a library file in the Macros LIBS option.
- Rebuild all VxWorks images.

4.2 To Load the HSS8 VxWorks Software Driver Separately

Note : This step is not required if the software driver was built into the BSP.

If the software driver is not built into the BSP, a user can load it separately :

- Copy ccHss8LibPPCV<version>.a to your present working directory as ccHss8.a.
- From the VxWorks shell type :
ld < ccHss8.a

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 4 of 51

5. **Using the HSS8 VxWorks Software Driver**

5.1 Overview of the HSS8 VxWorks Software Driver

The following flow chart shows the main functions of the HSS8 VxWorks Software Driver :

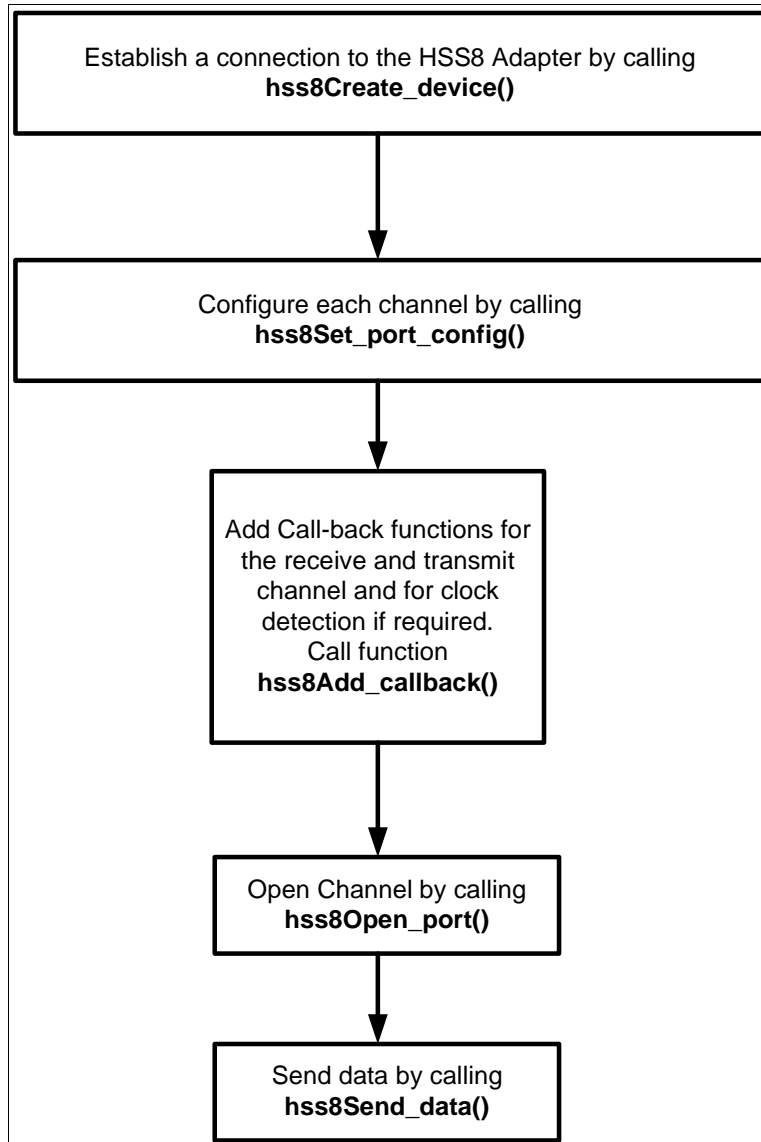


Figure 1 : Overview of HSS8 VxWorks Software Driver

5.2 Creating the Device

The HSS8 VxWorks Software Driver supports multiple HSS8 Adapters on a single host. To establish a connection and construct all the device specific structures, a user must create each of the devices separately, using the device ID to identify it.

The device ID starts at 0 and increments by 1 for each of the devices. Device 0 refers to the device in the lowest slot. The HSS8 VxWorks Software Driver can not be used until the user has created the device.

Example : For device 0 :

```
/* Create one HSS8 device */
hss8Create_device(HSS8_ARG_1, 0);
```

The device ID is used in all calls to the HSS8 VxWorks Software Driver to identify the correct device.

5.3 Configuring the Channels

The HSS8 Adapter has eight Serial Communications Controllers (SCCs) [Channels A - H] that support UART, HDLC/SDLC and BISYNC protocols, and four Serial Management Controllers (SMCs) [Channels I - L] that support only asynchronous UART.

After the HSS8 device has been created, the user must first set the default configuration for each of the Channels. To set the configuration of a Channel, a protocol-specific information structure is used. Examples of the required structure is given in ccHss8Test.c (for the UART protocol) and can be used as a starting point.

The structures allow the user to set all the protocol-specific options available on the HSS8 communication controller chip (the MPC8260 PowerQUICC II™). For available options for each of the structure fields, see [2.1.1].

Example : Set four SCC channels to UART mode, the other four SCC channels to HDLC mode and all four SMC channels to UART mode :

```
/* Set initial SCC port configuration */
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_A, anduart_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_B, anduart_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_C, anduart_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_D, anduart_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_E, andhdlc_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_F, andhdlc_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_G, andhdlc_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_H, andhdlc_info);

/* Set initial SMC port configuration */
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_I, andsmc_uart_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_J, andsmc_uart_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_K, andsmc_uart_info);
hss8Set_port_config(HSS8_ARG_3, 0, HSS8_PORT_L, andsmc_uart_info);
```

5.4 Adding Call-back Functions

The HSS8 VxWorks Software Driver notifies the user of different events by calling a user defined Call-back function. The events for which the user may specify one or more Call-back functions are :

- Transmit Begin - The software driver has accepted the data for sending.
- Transmit Done - The software driver has finished sending the data.
- Receive Done - Data has been received into a buffer specifically allocated by the software driver.
- Clock Detect - A clock signal has been detected on a specific channel.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 6 of 51

Only one Call-back function for each event is recommended. For the user to receive data, at least the Receive Done Call-back must be installed. While the Receive Done Call-back is executed, the corresponding buffer will not be accessed by the HSS8 VxWorks Software Driver. The user can process the data in the Call-back function or copy the data somewhere else for processing.

Call-back function prototype :

```
void hss8Callback_function(hss8ArgType num_args, ...);
```

Where num_args indicates the number of arguments to follow. The number of arguments differs for each call back function :

- Transmit Begin Call-back function (six arguments) :

1. dev_id - device ID.
2. port_id - port ID.
3. chan_id - channel ID.
4. user_id - user defined ID.
5. nr_bytes - length of transmitted data.
6. p_data - pointer to buffer containing the data to be send.

- Transmit Done Call-back function (eight arguments) :

1. dev_id - device ID.
2. port_id - port ID.
3. chan_id - channel ID.
4. user_id - user defined ID.
5. error - HSS8_OK (no errors)
- HSS8_TX_BUFFER_UNDERRUN_ERROR (HDLC and BISYNC only)
- HSS8_TX_CTS_LOST_ERROR (All protocols).
6. nr_bytes - length of transmitted data.
7. p_data - pointer to buffer containing the data that was send.
8. missed_tx_done - variable that is incremented when the HSS8 Adapter was not able to generate an interrupt to the host SBC.

- Receive Done Call-back function (eight arguments) :

1. dev_id - device ID.
2. port_id - port ID.
3. chan_id - channel ID.
4. user_id - user defined ID.
5. error - HSS8_OK (no errors)
- HSS8_RX_FRAMING_ERROR (UART only)
- HSS8_RX_PARITY_ERROR (UART and BISYNC only)
- HSS8_RX_CRC_ERROR (HDLC and BISYNC only)
- HSS8_RX_NONOCTET_ALIGNED_FRAME (HDLC and BISYNC only)
- HSS8_RX_ABORT_SEQUENCE (HDLC only).
6. nr_bytes - length of transmitted data.
7. p_data - pointer to buffer containing the data that was send.
8. missed_rx_done - variable that is incremented when the HSS8 Adapter was not able to generate an interrupt to the host SBC.

- Clock Detect Call-back function (three arguments) :

1. dev_id - device ID.
2. port_id - port ID.
3. user_id - user defined ID.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 7 of 51

Example : Add a Call-back function for handling receives :

```
/* Receive function prototype - this function is implemented by the user */
void Process_rx_data(hss8ArgType num_args, ...)
{
    hss8DeviceId dev_id;
    hss8PortId port_id;
    hss8ChannelId chan_id;
    hss8UserId user_id;
    hss8UINT32 error;
    hss8UINT32 nr_bytes;
    hss8UINT32 missed_rx;
    hss8BufferPtr p_data;

    /* declare a variable of type va_list */
    va_list arg_ptr;

    /* initialize the argument pointer */
    va_start(arg_ptr, num_args);

    /* retrieve each argument in the variable list */
    dev_id = va_arg(arg_ptr, hss8DeviceId);
    port_id = va_arg(arg_ptr, hss8PortId);
    chan_id = va_arg(arg_ptr, hss8ChannelId); /* always 0 for now */
    user_id = va_arg(arg_ptr, hss8UserId);
    error = va_arg(arg_ptr, hss8UINT32);
    nr_bytes = va_arg(arg_ptr, hss8UINT32);
    p_data = (hss8BufferPtr)va_arg(arg_ptr, hss8UINT32);
    missed_rx = va_arg(arg_ptr, hss8UINT32);

    /* perform clean up */
    va_end(arg_ptr);

    /* user specific code here... */
}

/* Add receive call-back */
hss8Add_callback(HSS8_ARG_4, 0, HSS8_CB_ON_RECEIVE_DONE, Process_rx_data, 0);
```

5.5 Sending and Receiving Data

To send and receive data on a specified channel, the user must first open the channel. To stop sending or receiving data from a channel, the user must close the channel.

Example : Send some data on device 0, Channel B :

```
/* Open port for sending data */
hss8Open_port(HSS8_ARG_5, 0, HSS8_PORT_B, 15, 16, 17);

/* Send some data */
hss8Send_data(HSS8_ARG_5, 0, HSS8_PORT_B, 0, 256, pBuffer256);

/* Do other stuff */
/* */

/* Close port after final usage */
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_B);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 8 of 51

5.6 Destroying the Device

When the device is no longer required it should be destroyed to free system resources.

Example : Device 0 is no longer required :

```
/* Close all ports after final usage */
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_A);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_B);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_C);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_D);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_E);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_F);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_G);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_H);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_I);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_J);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_K);
hss8Close_port(HSS8_ARG_2, 0, HSS8_PORT_L);

/* Destroy device to free resources */
hss8Destroy_device(HSS8_ARG_1, 0);
```

5.7 Detecting an Active Clock Signal on Channels

To detect when a channel's clock signal becomes active, use the following function.

Example : Detecting a clock signal on device 0 and Channel A :

```
/* Enable port to detect clock */
hss8Clock_detect(HSS8_ARG_2, 0, HSS8_PORT_A);
```

A Call-back function gets called once a clock has been detected. After this Call-back function has been serviced, the user can re-initialise the clock detection routine as shown above.

5.8 Obtaining the Current Host and Firmware Version Number

The following function prints out the current version number of the software driver and firmware software :

```
/* Print current version number */
hss8Version_print(HSS8_ARG_1, 0);
```

Note : Run hss8Create_device() first.

The following function returns the firmware version stored in the EEPROM :

```
/* Return firmware version number : version*100 + revision*10 + beta */
hss8Firmware_version(0, andversion);
```

Note : This function may be called without running hss8Create_device() first.

5.9 HSS8 Built-in Tests (BIT)

The following function displays each channel's statistics : e.g. how many bytes / packets have been accepted / rejected / sent / received and how many errors were reported.

Example : Displaying each channel's statistics for device 0 :

```
hss8Bit_report(HSS8_ARG_1, 0);
```

To clear the counters of the hss8Bit_report() function, use the function hss8Bit_clear().

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 9 of 51

Note : Instead of displaying each channel's statistics, the function `hss8Bit_getstruct()` only returns the corresponding statistics in a structure. See paragraph 6.2.13.

5.10 HSS8 Power-On-Self Tests

The Power-On-Self Tests (POST) are disabled by default, but can be selectively enabled. The following tests are defined :

- Magic number checking - checks the start of the flash for a correct magic number. This test is always performed and cannot be disabled.
- Flash CRC - checks that the checksum of the kernel in flash is correct. This test is disabled by default.
- EEPROM verification - verifies the contents of the EEPROM. If the contents is corrupt, the EEPROM is reprogrammed. This test is always performed.
- RAM databus test - checks the databus connected to the RAM for any errors. This test is disabled by default.
- RAM addressbus test - checks the addressbus connected to the RAM for any errors. This test is disabled by default.
- RAM device test - checks the whole RAM device for any errors. This test is disabled by default.

If any of the above tests fail, an error code is flashed on the LEDs D1 - D3 and the card will not boot up further. If the card is reset and the card still does not boot up, contact C²I² Systems.

The error codes are continuously flashed on the LEDs D1 - D3, followed by a short break. The number of flashes are defined as follows (see also header file `hss8Controllfc.h`) :

- `HSS_EEPROM_UPDATE` - 1 flash
- `HSS_EEPROM_ERROR` - 2 flashes
- `HSS_RAM_DATA_ERROR` - 3 flashes
- `HSS_RAM_ADDR_ERROR` - 4 flashes
- `HSS_RAM_DEVICE_ERROR` - 5 flashes
- `HSS_FLASH_MAGIC_ERROR` - 6 flashes
- `HSS_FLASH_KERNEL_CRC_ERROR` - 7 flashes
- `HSS8_SLAVE_POWERQUICC_II_FAIL` - 8 flashes

The above codes are also written to the EEPROM and may be read back with the following function :

Example : Obtain POST error status on device 0 :

```
hss8Post_status(0, &data)
```

To enable / disable selected tests, the following function may be used :

Example : Enable all RAM tests but disable Flash CRC checking on device 0 :

```
hss8Post_enable(0, HSS8_POST_RAM_DATA_ENABLE | HSS8_POST_RAM_ADDR_ENABLE |  
HSS8_POST_RAM_DEV_ENABLE);
```

Note : The function `hss8Create_device()` will return `HSS8_POWER_ON_SELFTEST_FAIL` if any of the tests fail. The function `hss8Post_status()` may then be used to determine which one of the tests has failed.

If a fast host processor is used, it is possible that the user might attempt to initialise the adapter while the POST routine is still executing. This will result in `hss8Create_device()` not executing correctly. Starting with software version 1.9.1, enabling `HSS8_POST_PROGRESS_ENABLE` will delay `hss8Create_device()` until POST has completed.

```
hss8Post_enable(0, HSS8_POST_PROGRESS_ENABLE);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 10 of 51

Note : If HSS8_POST_PROGRESS_ENABLE is enabled, host software earlier than V 1.9.1 will hang when hss8Create_device() is called.

5.11 Return Adapter Type

The following function returns the adapter type. The return value will be either 4 or 8, describing a 4-Channel (four SCCs and two SMCs available) or 8-Channel (eight SCCs and four SMCs available) adapter. When a 4-Channel adapter is present, only the following channels are valid : HSS8_PORT_[A - D] (four SCC channels) and HSS8_PORT_[I - J] (two SMC channels).

Example : Get adapter type of device 0 :

```
hss8UINT8 adapter_type;  
  
hss8Adapter_type(0, &adapter_type);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 11 of 51

6. **Application Program Interface (API)**

6.1 General Function Structure

The general function structure is as follows :

```
hss8Status hss8Function_name(hss8ArgType num_args, ...)
```

With num_args indicating the number of arguments to follow. It can be one of the following :

HSS8_ARG_1 -HSS8_ARG_15, depending on each individual function.

6.2 High-Speed Serial VxWorks Software Driver Interface

The zip file ccHss8HfilesV<version>.zip contains the following header files, which should always be included :

- hss8Defs.h
- hss8HostDriver.h
- hss8Controllfc.h

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 12 of 51

6.2.1 Create Device

Function : **hss8Create_device**

Purpose : Create and initialise the HSS8 device specific structures.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least one (dev_id).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.
- <buffer_size> - Maximum RX and TX buffer size for specific port (optional). One of HSS8_2K, HSS8_4K, HSS8_8K, HSS8_16K or HSS8_32K. If not specified, the default size is HSS8_2K.

Returns :

- HSS8_OK - On success.
- HSS8_INVALID_PARAM - Invalid <dev_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_PCI_INIT_FAIL - PCI initialisation failed.
- HSS8_MEM_ALLOC_FAILED - If HSS8 device structure could not be created in memory.
- HSS8_DEVICE_NOT_FOUND - If HSS8 device <dev_id> was not found on the PCI bus.
- HSS8_MEM_INVALID_ADDRESS - If the HSS8 device PCI address was not valid.
- HSS8_MEM_EEPROM_BUSY - If HSS8 device could not read version number from EEPROM.
- HSS8_POWER_ON_SELFTEST_FAIL - If the POST failed.

```
hss8Status hss8Create_device(hss8ArgType num_args, ...);
```

Note : This function has to be called (once per device) before any other function call to the specified device will be valid.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 13 of 51

6.2.2 Destroy Device

Function : **hss8Destroy_device**

Purpose : Destroy the HSS8 device specific structures.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least one (dev_id).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
 - <dev_id> = 0, next HSS8 device :
 - <dev_id> = 1, etc.

Returns :

- HSS8_OK - On success.
- HSS8_INVALID_PARAM - Invalid <dev_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_PCI_INIT_FAIL - PCI initialisation failed.
- HSS8_ERROR - Device has already been destroyed.

```
hss8Status hss8Destroy_device(hss8ArgType num_args, ...);
```

Note : After this function is called, no other function call to the specified device will be valid, except for hss8Create_device(..).

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 14 of 51

6.2.3 Set Port Configuration

Function : **hss8Set_port_config**

Purpose : Set port protocol and protocol configuration.

Arguments :

- | | |
|--|--|
| <p><num_args></p> <p><dev_id></p> <p><port_id></p> <p><p_info></p> | <ul style="list-style-type: none">- The number of arguments to follow. Needs to be at least three (dev_id, port_id, p_info).- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
 <dev_id> = 0,
 next HSS8 device :
 <dev_id> = 1, etc.- Port to configure.- Pointer to information struct (hss8ProtocollInfo) used for configuration. |
|--|--|

Returns :

- | | |
|---|---|
| <p>HSS8_OK</p> <p>HSS8_PCI_INIT_FAIL</p> <p>HSS8_ERROR</p> <p>HSS8_INVALID_PARAM</p> <p>HSS8_PORT_NOT_INSTALLED</p> <p>HSS8_DEVICE_BUSY</p> <p>HSS8_DEVICE_NOT_RESPONDING</p> <p>HSS8_INCORRECT_PARAM_COMBINATION</p> | <ul style="list-style-type: none">- On success.- PCI initialisation failed.- If the Tx/Rx tasks have not been destroyed.- Invalid <dev_id> or <port_id> supplied or the function has been called with the incorrect number of variables.- If the port does not exist.- If no PCI buffer is available.- If the HSS8 control block could not be accessed within a certain time.- If an incorrect parameter combination was selected in the protocol structure. |
|---|---|

```
hss8Status hss8Set_port_config(hss8ArgType num_args, ...);
```

Note: The <p_info> pointer must point to a valid hss8ProtocollInfo structure with all protocol information set as required. If only a few items need to change, the hss8Get_port_config(..) function should be used to fill in the rest of the structure.

Warning : Do not call this function while sending or receiving data as this may result in data loss.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 15 of 51

6.2.4 Get Port Configuration

Function : **hss8Get_port_config**

Purpose : Get port protocol and protocol configuration.

Arguments :

<num_args>	- The number of arguments to follow. Needs to be at least three (dev_id, port_id, p_info).
<dev_id>	- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot : <dev_id> = 0, next HSS8 device : <dev_id> = 1, etc.
<port_id>	- Port to get configuration info from.
<p_info>	- Pointer to information struct (hss8ProcollInfo) used for configuration.

Returns :

HSS8_OK	- On success.
HSS8_ERROR	- If the Tx/Rx tasks have not been destroyed.
HSS8_INVALID_PARAM	- Invalid <dev_id> or <port_id> supplied or the function has been called with the incorrect number of variables.
HSS8_PORT_NOT_INSTALLED	- If the port does not exist.
HSS8_DEVICE_BUSY	- If no PCI buffer is available.
HSS8_DEVICE_NOT_RESPONDING	- If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Get_port_config(hss8ArgType num_args, ...);
```

Note : The <p_info> pointer must point to an existing hss8ProcollInfo structure.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 16 of 51

6.2.5 Open Port

Function : **hss8Open_port**

Purpose : Open specified port for send and receive.

Arguments :

- | | |
|--|--|
| <p><num_args></p> <p><dev_id></p> <p><port_id></p> <p><tx_/rx_/clk_priority></p> <p><fp_options></p> | <ul style="list-style-type: none">- The number of arguments to follow. Needs to be at least five (dev_id, port_id, tx_priority, rx_priority, clk_priority).- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.- Port to open for send and receive.- Priority of the send, receive and clock detection task servicing this port.- Floating point enable for send, receive and clock detect task :
HSS8_TX_TASK_FP_ENABLE,
HSS8_RX_TASK_FP_ENABLE,
HSS8_CLK_TASK_FP_ENABLE. |
|--|--|

Returns :

- | | |
|--|--|
| <p>HSS8_OK</p> <p>HSS8_ERROR</p> <p>HSS8_INVALID_PARAM</p> <p>HSS8_PORT_NOT_INSTALLED</p> <p>HSS8_PORT_NOT_CONFIGURED</p> <p>HSS8_DEVICE_BUSY</p> <p>HSS8_DEVICE_NOT_RESPONDING</p> <p>HSS8_MEM_ALLOC_FAILED</p> | <ul style="list-style-type: none">- On success.- If opening of port failed.- Invalid <dev_id> or <port_id> supplied or the function has been called with the incorrect number of variables.- If the port does not exist.- If an 'Open' is attempted on a port before configuring the port.- If no PCI buffer is available.- If the HSS8 control block could not be accessed within a certain time.- If failed to create semaphore or spawn tasks. |
|--|--|

```
hss8Status hss8Open_port(hss8ArgType num_args, ...);
```

Note : This function must be called prior to attempting to send or receive on the specified port. Opening a port spawns a receive, send and clock detect task for that specific port. The priority of these tasks is specified by the user.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 17 of 51

6.2.6 Close Port

Function : **hss8Close_port**

Purpose : Close specified port for send and receive.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least two (dev_id, port_id).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.
- <port_id> - Port to close for send and receive.

Returns :

- HSS8_OK - On success.
- HSS8_ERROR - If opening of port failed or Rx/Tx tasks have not been destroyed.
- HSS8_INVALID_PARAM - Invalid <dev_id> or <port_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_PORT_NOT_INSTALLED - If the port does not exist.
- HSS8_DEVICE_BUSY - If no PCI buffer is available.
- HSS8_DEVICE_NOT_RESPONDING - If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Close_port(hss8ArgType num_args, ...);
```

Note : Closing a port a second time has no effect and still returns HSS8_OK, since the port was already successfully closed.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 18 of 51

6.2.7 Send Data

Function : **hss8Send_data**

Purpose : Send data over the specified port.

Arguments :

<num_args>

- The number of arguments to follow. Needs to be at least five (dev_id, port_id, chan_id, nr_bytes, p_data).

<dev_id>

- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :

<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.

<port_id>

- Port on which data must be sent.

<chan_id>

- Channel on which data must be sent. If a port has only one channel, <chan_id> = 0 (not currently implemented).

<nr_bytes>

- Number of bytes to send.

<p_data>

- Pointer to buffer with at least <nr_bytes> bytes of data.

Returns :

HSS8_OK

- On success.

HSS8_INVALID_PARAM

- Invalid <dev_id>, <port_id>, <chan_id>, <nr_bytes> or <p_data> supplied or the function has been called with the incorrect number of variables.

HSS8_PORT_NOT_INSTALLED

- If the port does not exist.

HSS8_PORT_NOT_OPEN

- If the port is not yet open.

HSS8_DEVICE_BUSY

- If no PCI buffer is available.

HSS8_DEVICE_NOT_RESPONDING

- If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Send_data(hss8ArgType num_args, ...);
```

Note : The port must be opened before attempting to send data over it.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 19 of 51

6.2.8 Add Call-back

Function : **hss8Add_callback**

Purpose : Add a user defined Call-back routine.

Arguments :

- | | |
|------------|---|
| <num_args> | - The number of arguments to follow. Needs to be at least four (dev_id, cb_type, callback, user_id). |
| <dev_id> | - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc. |
| <cb_type> | - Call-back type, one of : HSS8_CB_ON_SEND_BEGIN, HSS8_CB_ON_SEND_DONE, HSS8_CB_ON_RECEIVE_DONE, HSS8_CB_ON_CLOCK_DETECT. |
| <callback> | - User function. |
| <user_id> | - User identifier. This identifier will be passed to the Call-back function when it is called. |

Returns :

- | | |
|-----------------------|---|
| HSS8_OK | - On success. |
| HSS8_INVALID_PARAM | - Invalid <dev_id> supplied or the function has been called with the incorrect number of variables. |
| HSS8_MEM_ALLOC_FAILED | - If HSS8 Call-back node could not be created in memory. |

```
hss8Status hss8Add_callback(hss8ArgType num_args, ...);
```

Four Call-backs are provided for user notification from the software driver :

- HSS8_CB_ON_SEND_BEGIN : This Call-back will be called as soon as the data has been handed over to the software driver for sending.
- HSS8_CB_ON_SEND_DONE : This Call-back will be called when all the data for a given send has been sent by the software driver.
- HSS8_CB_ON_RECEIVE_DONE : This Call-back will be called when a block of data has been received by the software driver. The user must add at least one of these Call-backs to receive data.
- HSS8_CB_ON_CLOCK_DETECT : This Call-back will be called when a clock signal has been detected on a port and only if the port has been instructed to detect a clock signal, e.g. calling the function hss8Clock_detect().

Only one Call-back for each above type per device is recommended. The Call-back function receives the port id, such that the user can distinguish which port triggered the Call-back. More than one Call-back function may be used, in which case the Call-backs will be called in the sequence they were added.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 20 of 51

6.2.9 Remove Call-back

Function :

hss8Remove_callback

Purpose :

Remove a user defined Call-back routine.

Arguments :

<num_args>

- The number of arguments to follow. Needs to be at least four (dev_id, cb_type, callback, user_id).

<dev_id>

- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :

<dev_id> = 0,
next HSS8 device :

<dev_id> = 1, etc.

<cb_type>

- Call-back type, one of : HSS8_CB_ON_SEND_BEGIN,
HSS8_CB_ON_SEND_DONE,
HSS8_CB_ON_RECEIVE_DONE,
HSS8_CB_ON_CLOCK_DETECT.

<callback>

- User function to remove.

<user_id>

- User identifier. This identifier must be the same as the one passed to hss8Add_callback.

Returns :

HSS8_OK

- On success.

HSS8_INVALID_PARAM

- Invalid <dev_id> supplied or the function has been called with the incorrect number of variables.

```
hss8Status hss8Remove_callback(hss8ArgType num_args, ...);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 21 of 51

6.2.10 Detecting an Active Clock Signal on Ports

Function : **hss8Clock_detect**

Purpose : Set up a port to detect when a clock signal becomes active.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least two (dev_id, port_id).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
 - <dev_id> = 0, next HSS8 device :
 - <dev_id> = 1, etc.
- <port_id> - Port on which to detect clock signal.

Returns :

- HSS8_OK - On success.
- HSS8_INVALID_PARAM - Invalid <dev_id> or <port_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_PORT_NOT_INSTALLED - If the port does not exist.
- HSS8_DEVICE_BUSY - If no PCI buffer is available.
- HSS8_DEVICE_NOT_RESPONDING - If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Clock_detect(hss8ArgType num_args, ...);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 22 of 51

6.2.11 Print Out Current Host Software Version Number

Function : **hss8Version_print**

Purpose : To obtain the current version number of the software driver and firmware software. This function essentially calls `hss8Bit_getstruct(..)` and prints out the contents of the `hss8BoardBitInfo` struct. See paragraph 6.2.13.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least one (`dev_id`).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
 - <dev_id> = 0, next HSS8 device :
 - <dev_id> = 1, etc.

Returns :

- HSS8_OK - On success.
- HSS8_INVALID_PARAM - Invalid <dev_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_DEVICE_BUSY - If no PCI buffer is available.
- HSS8_DEVICE_NOT_RESPONDING - If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Version_print(hss8ArgType num_args, ...);
```

Note : This function can only be used after a call to `hss8Create_device(..)`.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 23 of 51

6.2.12 Print Out Current Embedded Software Version Number

Function : **hss8Firmware_version**

Purpose : Return the firmware version number stored in the EEPROM.

Arguments :

- <dev_id>
- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.
- <version>
- Return value for version number :
version*100 + revision*10 + beta.

Returns :

- HSS8_OK
- On success.
- HSS8_INVALID_PARAM
- Invalid <dev_id> supplied.
- HSS8_DEVICE_NOT_FOUND
- If HSS8 device <dev_id> was not found on the PCI bus.
- HSS8_MEM_INVALID_ADDRESS
- If the HSS8 device PCI address was not valid.
- HSS8_MEM_EEPROM_BUSY
- If the HSS8 device could not read version number from EEPROM.

```
hss8Status hss8Firmware_version(hss8DeviceId dev_id, hss8UINT32 *version);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 24 of 51

6.2.13 HSS8 BIT Structures

The following structures define the HSS8 BIT variables (defined in hss8Controllfc.h) :

BIT Structures :

```
struct hss8BoardBitInfoStruct
{
    hss8UINT32 board_number;
    hss8UINT32 firmware_version;
    hss8UINT32 firmware_revision;
    hss8UINT32 firmware_beta;
    char firmware_creation_date[30];
};
typedef struct hss8BoardBitInfoStruct hss8BoardBitInfo;

struct hss8SendBitInfoStruct
{
    hss8Count nr_accepted;
    hss8Count nr_rejected;
    hss8Count nr_errors;
    hss8Count nr_sent;
    hss8Count nr_bytes_accepted;
    hss8Count nr_bytes_rejected;
    hss8Count nr_bytes_sent;
};
typedef struct hss8SendBitInfoStruct hss8SendBitInfo;

struct hss8ReceiveBitInfoStruct
{
    hss8Count nr_buffers_busy;
    hss8Count nr_received;
    hss8Count nr_bytes_received;
    hss8Count nr_errors;
};
typedef struct hss8ReceiveBitInfoStruct hss8ReceiveBitInfo;
```

Main BIT Structure :

```
struct hss8BitInfoStruct
{
    hss8BoardBitInfo board_bit;
    hss8SendBitInfo tx_scc_bit[HSS8_HW_NR_SCC];
    hss8ReceiveBitInfo rx_scc_bit[HSS8_HW_NR_SCC];
    hss8SendBitInfo tx_smc_bit[HSS8_HW_NR_SMC];
    hss8ReceiveBitInfo rx_smc_bit[HSS8_HW_NR_SMC];
};
typedef struct hss8BitInfoStruct hss8BitInfo;
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 25 of 51

Three functions give access to the HSS8 BIT structures :

Function : **hss8Bit_getstruct**

Purpose : To obtain the latest BIT variables.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least two (dev_id, bit_info).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.
- <bit_info> - Pointer to hss8BitInfo struct.

Returns :

- HSS8_OK - On success.
- HSS8_INVALID_PARAM - Invalid <dev_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_DEVICE_BUSY - If no PCI buffer is available.
- HSS8_DEVICE_NOT_RESPONDING - If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Bit_getstruct(hss8ArgType num_args, ...);
```

Function : **hss8Bit_report**

Purpose : To display each port's statistics.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least one (dev_id).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.

Returns :

- HSS8_OK - On success.
- HSS8_INVALID_PARAM - Invalid <dev_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_DEVICE_BUSY - If no PCI buffer is available.
- HSS8_DEVICE_NOT_RESPONDING - If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Bit_report(hss8ArgType num_args, ...);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 26 of 51

Function : **hss8Bit_clear**

Purpose : To clear each port's counters.

Arguments :

- <num_args> - The number of arguments to follow. Needs to be at least one (dev_id).
- <dev_id> - Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
 - <dev_id> = 0, next HSS8 device :
 - <dev_id> = 1, etc.

Returns :

- HSS8_OK - On success.
- HSS8_INVALID_PARAM - Invalid <dev_id> supplied or the function has been called with the incorrect number of variables.
- HSS8_DEVICE_BUSY - If no PCI buffer is available.
- HSS8_DEVICE_NOT_RESPONDING - If the HSS8 control block could not be accessed within a certain time.

```
hss8Status hss8Bit_clear(hss8ArgType num_args, ...);
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 27 of 51

6.2.14 Enable / Disable POST

Function :

hss8Post_enable

Purpose :

Enable / disable selected POSTs.

Arguments :

<dev_id>

- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.

<test_enable>

- One or all of the following ORed together :
HSS8_POST_RAM_DATA_ENABLE,
HSS8_POST_RAM_ADDR_ENABLE,
HSS8_POST_RAM_DEV_ENABLE,
HSS8_POST_KERNEL_CRC_ENABLE.

Returns :

HSS8_OK

- On success.

HSS8_INVALID_PARAM

- Invalid <dev_id> supplied.

HSS8_DEVICE_NOT_FOUND

- If HSS8 device <dev_id> was not found on the PCI bus.

HSS8_MEM_INVALID_ADDRESS

- If the HSS8 device PCI address was not valid.

HSS8_MEM_EEPROM_BUSY

- If the HSS8 device could not write to EEPROM.

```
hss8Status hss8Post_enable(hss8DeviceId dev_id, hss8UINT8 test_enable);
```

Note : This function may be called without calling hss8Create_device(..) first.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 28 of 51

6.2.15 Return POST Status

Function :

hss8Post_status

Purpose :

Return POST status or error code.

Arguments :

<dev_id>

- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.

<status>

- Byte returning the error code of the POST. One of :
 - HSS8_OK -On success.
 - HSS8_EEPROM_UPDATE -EEPROM was corrupt and was reprogrammed.
 - HSS8_EEPROM_ERROR -EEPROM read / write error.
 - HSS8_RAM_DATA_ERROR -RAM databus error.
 - HSS8_RAM_ADDR_ERROR -RAM addressbus error.
 - HSS8_RAM_DEVICE_ERROR -RAM device error.
 - HSS8_FLASH_MAGIC_ERROR -Flash magic number corrupt.
 - HSS8_FLASH_KERNEL_CRC_ERROR -Flash CRC error.
 - HSS8_SLAVE_POWERQUICC_II_FAIL -Second PowerQUICC II processor failed to start up.

Returns :

HSS8_OK

- On success.

HSS8_INVALID_PARAM

- Invalid <dev_id> supplied.

HSS8_DEVICE_NOT_FOUND

- If HSS8 device <dev_id> was not found on the PCI bus.

HSS8_MEM_INVALID_ADDRESS

- If the HSS8 device PCI address was not valid.

HSS8_MEM_EEPROM_BUSY

- If the HSS8 device could not read status from EEPROM.

```
hss8Status hss8Post_status(hss8DeviceId dev_id, hss8UINT8 *status);
```

Note : This function may be called without calling hss8Create_device(..) first.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 29 of 51

6.2.16 Return Adapter Type

Function :

hss8Adapter_type

Purpose :

Return adapter type : either 4-Channel or 8-Channel.

Arguments :

<dev_id>

- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :
<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.
- Byte returning the adapter type : either 4 or 8.

<adapter_type>

Returns :

HSS8_OK

- On success.

HSS8_INVALID_PARAM

- Invalid <dev_id> supplied.

HSS8_DEVICE_NOT_FOUND

- If HSS8 device <dev_id> was not found on the PCI bus.

HSS8_MEM_INVALID_ADDRESS

- If the HSS8 device PCI address was not valid.

HSS8_MEM_EEPROM_BUSY

- If the HSS8 device could not read status from EEPROM.

```
hss8Status hss8Adapter_type(hss8DeviceId dev_id, hss8UINT8 *adapter_type);
```

Note : This function may be called without calling hss8Create_device(..) first.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 30 of 51

6.2.17 Set PCI Base Address

Function :

hss8Set_pci_base

Purpose :

This function is used to set the PCI base address used by the adapter's firmware for the PCI reverse mapping (adapter to host). A default value of 0x00000000 is assumed by the adapter at start-up.

Arguments :

<dev_id>

- Device ID on the PCI bus. The HSS8 device in the lowest PCI slot :

<dev_id> = 0,
next HSS8 device :
<dev_id> = 1, etc.

<addr>

- A 32-bit value to be used as the new PCI base address.

Returns :

HSS8_OK

- On success.

HSS8_INVALID_PARAM

- Invalid <dev_id> supplied.

HSS8_DEVICE_BUSY

- If no PCI buffer is available.

HSS8_ERROR

- If firmware did not respond to configuration change.

```
hss8Status hss8Set_pci_base(hss8DeviceId dev_id, hss8UINT32 addr);
```

Note : This function can only be used after a call to hss8Create_device(..).

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 31 of 51

6.2.18 Set Output Pin Values at Startup

Function :

hss8Set_init_config

Purpose :

The standard behaviour is that all line drivers are disabled (lines are tristate) until the channel is initialised. In some cases it may be desirable to force the lines to a known state as soon as the adapter is powered up.

This function stores a value in the adapter EEPROM which is used to selectively enable the line drivers at startup if required.

The default line idle state is a logic one.

Arguments :

<num_args>

- The number of arguments to follow. Needs to be at least one (dev_id).

<dev_id>

- Device ID on the PCI bus.

<enable mask>

- 0x00 to 0xFF, where 0x01 enables Channel A, 0x02 enables Channel B, etc.

Returns :

HSS8_OK

- On success.

HSS8_INVALID_PARAM

- Invalid <dev_id> supplied.

HSS8_DEVICE_BUSY

- If no PCI buffer is available.

HSS8_MEM_EEPROM_BUSY

- EEPROM error.

HSS8_ERROR

- If firmware did not respond to configuration change.

```
hss8Status hss8Set_init_config(hss8ArgType num_args, ...);
```

Example :

From the VxWorks command line :

hss8Set_init_config(2,0,0x01) will program the adapter so that the line drivers for Channel A is enabled at startup. A reboot is required to verify correct behaviour.

hss8Set_init_config(2,0,0) will program the adapter to default behaviour.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 32 of 51

6.3 Software Driver Data Structures

Each protocol defines a protocol information structure used to configure a port with protocol specific options. This paragraph details the information structures used by each protocol and explains the use and limitations of every structure member.

hss8ProtocolInfo Structure :

```
struct hss8ProtocolInfoStruct
{
    hss8UINT32 protocol_id;
    union
    {
        /* SCC info */
        hss8UartInfo uart;
        hss8HdlcInfo hdlc;
        hss8BisyncInfo bisync;
        /* SMC info */
        hss8SmcUartInfo smc_uart;
    } info;
};
typedef struct hss8ProtocolInfoStruct hss8ProtocolInfo;
```

protocol_id :

```
HSS8_PROTOCOL_UART
HSS8_PROTOCOL_HDLC
HSS8_PROTOCOL_BISYNC
HSS8_PROTOCOL_SMC_UART
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 33 of 51

6.3.1 UART Mode

This protocol may only be used with the eight SCC channels : Channels A - H.

6.3.1.1 UART Protocol Information Structure

The following structure is defined in the file hss8Controllfc.h and is given here in abbreviated format (i.e. reserved and obsolete members are not shown). Always use the structure as defined in hss8Controllfc.h.

```
struct hss8UartInfoStruct
{
    hss8UINT32 baud_rate;
    hss8UINT32 flow_control;
    hss8UINT32 stop_bits;
    hss8UINT32 data_bits;
    hss8UINT32 uart_mode;
    hss8UINT32 freeze_tx;
    hss8UINT32 rx_zero_stop_bits;
    hss8UINT32 sync_mode;
    hss8UINT32 disable_rx_while_tx;
    hss8UINT32 parity_enable;
    hss8UINT32 rx_parity;
    hss8UINT32 tx_parity;
    hss8UINT32 diag_mode;
    hss8UINT32 max_receive_bytes;
    hss8UINT32 max_idl;
    hss8UINT32 brkcr;
    hss8UINT32 parec;
    hss8UINT32 frmec;
    hss8UINT32 noseq;
    hss8UINT32 brkec;
    hss8UINT32 uaddr1;
    hss8UINT32 uaddr2;
    hss8UINT32 toseq;
    hss8UINT32 cc[8];
    hss8UINT32 rccm;
    hss8UINT32 clock_source;
};
typedef struct hss8UartInfoStruct hss8UartInfo;
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 34 of 51

6.3.1.2 UART Protocol Information Structure Members

Name	Options	Description	
baud_rate	<p>1 200 - 1 Mbit/s (RS-232) 1 200 - 16 Mbit/s (RS-422/485)</p> <p>Any values permissible.</p> <p>The equation to calculate the actual baud rate for asynchronous UART is as follows :</p> $\text{Actual baud rate} = 100 \text{ MHz} / 16 / \text{ROUND}(100 \text{ MHz} / 16 / \text{Desired baud rate})$ <p>The equation to calculate the actual baud rate for synchronous UART is as follows :</p> $\text{Actual baud rate} = 100 \text{ MHz} / \text{ROUND}(100 \text{ MHz} / \text{Desired baud rate})$ <p>Where ROUND() implies that the result is rounded to the nearest integer.</p>	<p>Used to specify a single baud rate for both transmitter and receiver.</p> <p>Units in bit/s.</p>	
clock_source	HSS8_CLOCK_DEFAULT	<p>HSS8_CLOCK_DEFAULT connects Baud Rate Generator (BRGs) [1 - 4] to Channels [A - D] and Channels [E - H].</p> <p>For synchronous UART : When transmit clock is set to HSS8_CLOCK_BRG[1 - 4], then receive clock is still set to HSS8_CLOCK_EXT[1 - 4] for Channel [A - D] and [E - H].</p> <p>For asynchronous UART : Transmit and receive clocks can be set to one of HSS8_CLOCK_BRG[1 - 4] or HSS8_CLOCK_EXT[1 - 4].</p> <p>Note : There are four BRGs and four clock input pins per PowerQUICC II processor.</p>	
	<p>HSS8_CLOCK_BRG1 HSS8_CLOCK_BRG2 HSS8_CLOCK_BRG3 HSS8_CLOCK_BRG4</p>		<p>BRGs [1-4]. BRG1 for Channels [A and E] BRG2 for Channels [B and F] BRG3 for Channels [C and G] BRG4 for Channels [D and H]</p>
	<p>HSS8_CLOCK_EXT1 HSS8_CLOCK_EXT2 HSS8_CLOCK_EXT3 HSS8_CLOCK_EXT4</p>		<p>External Clocks connected on CLK_IN Pins.</p> <p>Note : HSS8_CLOCK_EXT[1-2] can only be used for Channels [A and B] and [E and F], while HSS8_CLOCK_EXT[3-4] can only be used for Channels [C and D] and [G and H].</p>
flow_control	<p>HSS8_UART_FLOW_NORMAL HSS8_UART_FLOW_ASYNC</p>	Normal or asynchronous flow control.	
stop_bits	<p>HSS8_UART_STOP_BITS_ONE HSS8_UART_STOP_BITS_TWO</p>	Number of full stop bits.	
data_bits	<p>HSS8_UART_DATA_BITS_5 HSS8_UART_DATA_BITS_6 HSS8_UART_DATA_BITS_7 HSS8_UART_DATA_BITS_8 HSS8_UART_DATA_BITS_9 HSS8_UART_DATA_BITS_10 HSS8_UART_DATA_BITS_11 HSS8_UART_DATA_BITS_12 HSS8_UART_DATA_BITS_13 HSS8_UART_DATA_BITS_14</p>	Number of data bits. Note only channels [I - L] (i.e. the SMC channels) support nine or more data bits.	
uart_mode	<p>HSS8_UART_MODE_NORMAL HSS8_UART_MODE_MAN_MM HSS8_UART_MODE_AUTO_MM</p>	Select UART mode : Normal, manual multidrop or automatic multidrop mode.	

Name	Options		Description
freeze_tx	HSS8_UART_FREEZE_TX_NORMAL HSS8_UART_FREEZE_TX_FREEZE		Pause (freeze) transmission. Transmission continues when set back to normal.
rx_zero_stop_bits	HSS8_UART_RX_ZERO_STOP_BITS_NORMAL HSS8_UART_RX_ZERO_STOP_BITS_NONE		If set to none, the receiver receives data without stop bits.
sync_mode	HSS8_UART_SYNC_MODE_ASYNC HSS8_UART_SYNC_MODE_SYNC		Select asynchronous (normal) or synchronous mode.
disable_rx_while_tx	HSS8_UART_DISABLE_RX_WHILE_TX_NORMAL HSS8_UART_DISABLE_RX_WHILE_TX_DISABLE		Enable (normal) or disable receiver while transmitting. Used in multidrop mode to prevent reception of own messages.
parity_enable	HSS8_UART_PARITY_NO_PARITY HSS8_UART_PARITY_ENABLE		Enable or disable parity checking.
rx_parity, tx_parity	HSS8_UART_PARITY_ODD HSS8_UART_PARITY_LOW HSS8_UART_PARITY_EVEN HSS8_UART_PARITY_HIGH		Receive and transmit parity. Parity will only be checked if parity is enabled.
diag_mode	HSS8_DIAG_NORMAL	Normal operation. Use this for external loopback .	Set diagnostic mode. External loopback - RS-422/485 : Connect TxD+ to RxD+, TxD- to RxD-, (CLK_OUT+ to CLK_IN+ and CLK_OUT- to CLK_IN- for synchronous mode). External loopback - RS-232 : Connect TxD to RxD, (CLK_OUT to CLK_IN for synchronous mode) and RTS to CTS and CD.
	HSS8_DIAG_LOOPBACK	Internal loopback : TxD and RxD are connected internally. The value on RxD, CTS and CD is ignored. The transmitter and receiver share the same clock source.	
	HSS8_DIAG_ECHO	The transmitter automatically resends received data bit-by-bit.	
	HSS8_DIAG_LOOPBACK_EC HO	Loopback and echo operation occur simultaneously.	
max_receive_bytes	1 to 2 048 (default) or up to 32 kBytes, depending on how many bytes have been allocated to the RX and TX buffers (See function hss8Create_device(..)).		Maximum number of bytes that may be copied into a buffer.
max_idl	0 to 2 048 (default) or up to 32 kBytes, depending on how many bytes have been allocated to the RX and TX buffers (See function hss8Create_device(..)).		Maximum idle characters. When a character is received, the receiver begins counting idle characters. If max_idl idle characters are received before the next data character, an idle timeout occurs and the buffer is closed. Thus, max_idl offers a way to demarcate frames. To disable the feature, clear max_idl. The bit length of an idle character is calculated as follows : 1 + data length (5-9) + 1 (if parity is used) + number of stop bits (1-2). For 8 data bits, no parity, and 1 stop bit, the character length is 10 bits.
brkcr	0 - 2 048		Number of break characters sent by transmitter. For 8 data bits, no parity, 1 stop bit, and 1 start bit, each break character consists of 10 zero bits.
parec	0 - 65 535		Number of received parity errors.

Name	Options	Description
frmec	0 - 65 535	Number of received characters with framing errors.
nosec	0 - 65 535	Number of received characters with noise errors.
brkec	0 - 65 535	Number of break conditions on the signal.
uaddr1, uaddr2	0x0000 - 0x00FF	Address in multidrop mode. Only the lower 8 bits are used so the upper 8 bits should be cleared.
toseq	0x0000 - 0x00FF	Transmit out of sequence character (e.g. XON, XOFF).
cc[8]	0b00-----cccccccc - Valid entry. 0b10-----cccccccc - Entry not valid and is not used.	Control character 1 to 8. These characters can be used to delimit received messages. ----- (6 bits) - Reserved. Initialise to zero. cccccccc (8 bits) - Defines control characters to be compared to the incoming character.
rccm	0b11-----00000000 - Ignore these bits when comparing incoming character. 0b11-----11111111 - Enable comparing the incoming character to cc[n].	Receive control character mask. A one enables comparison and a zero masks it.

6.3.2 HDLC Mode

This protocol may only be used with the eight SCC channels : Channels A - H.

6.3.2.1 HDLC Protocol Information Structure

The following structure is defined in the file hss8Controllfc.h and is given here in abbreviated format (i.e. reserved and obsolete members are not shown). Always use the structure as defined in hss8Controllfc.h.

```
struct hss8HdlcInfoStruct
{
    hss8UINT32 baud_rate;
    hss8UINT32 crc_mode;
    hss8UINT32 diag_mode;
    hss8UINT32 max_receive_bytes;
    hss8UINT32 max_frame_bytes;
    hss8UINT32 address_mask;
    hss8UINT32 address1;
    hss8UINT32 address2;
    hss8UINT32 address3;
    hss8UINT32 address4;
    hss8UINT32 nr_flags_between_frames;
    hss8UINT32 retransmit_enabled;
    hss8UINT32 flag_sharing_enabled;
    hss8UINT32 rx_disabled_during_tx;
    hss8UINT32 bus_mode;
    hss8UINT32 bus_mode_rts;
    hss8UINT32 multiple_tx_frames;
    hss8UINT32 encoding_method;
    hss8UINT32 preamble_length;
    hss8UINT32 preamble_pattern;
    hss8UINT32 send_idles_or_flags;
    hss8UINT32 clock_source;
};
typedef struct hss8HdlcInfoStruct hss8HdlcInfo;
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 38 of 51

6.3.2.2 HDLC Protocol Information Structure Members

Name	Options	Description	
baud_rate	<p>1 200 - 1 Mbit/s (RS-232) 1 200 - 16 Mbit/s (RS-422/485)</p> <p>Any values permissible.</p> <p>The equation to calculate the actual baud rate for FM0/1, Manchester and Diff. Manchester is as follows :</p> $\text{Actual baud rate} = 100 \text{ MHz} / 16 / \text{ROUND}(100 \text{ MHz} / 16 / \text{Desired baud rate})$ <p>The equation to calculate the actual baud rate for NRZ/NRZI is as follows :</p> $\text{Actual baud rate} = 100 \text{ MHz} / \text{ROUND}(100 \text{ MHz} / \text{Desired baud rate})$ <p>Where ROUND() implies that the result is rounded to the nearest integer.</p>	<p>Used to specify a single baud rate for both transmitter and receiver.</p> <p>Units in bit/s.</p>	
clock_source	HSS8_CLOCK_DEFAULT	<p>HSS8_CLOCK_DEFAULT connects BRGs [1 - 4] to Channels [A - D] and Channels [E - H].</p> <p>For NRZ/NRZI : When transmit clock is set to HSS8_CLOCK_BRG[1 - 4], then receive clock is still set to HSS8_CLOCK_EXT[1 - 4] for Channels [A - D] and [E - H].</p> <p>For FM0/1, Manchester and Diff. Manchester : Transmit and receive clocks can be set to one of HSS8_CLOCK_BRG[1 - 4] or HSS8_CLOCK_EXT[1 - 4].</p> <p>Note : There are four BRGs and four clock input pins per PowerQUICC II processor.</p>	
	HSS8_CLOCK_BRG1 HSS8_CLOCK_BRG2 HSS8_CLOCK_BRG3 HSS8_CLOCK_BRG4		BRGs [1-4]. BRG1 for Channels [A and E] BRG2 for Channels [B and F] BRG3 for Channels [C and G] BRG4 for Channels [D and H]
	HSS8_CLOCK_EXT1 HSS8_CLOCK_EXT2 HSS8_CLOCK_EXT3 HSS8_CLOCK_EXT4		External Clocks connected on CLK_IN Pins. Note : HSS8_CLOCK_EXT[1 - 2] can only be used for Channels [A and B] and [E and F], while HSS8_CLOCK_EXT[3 - 4] can only be used for Channels [C and D] and [G and H].
crc_mode	HSS8_HDLC_CRC_MODE_16_BIT HSS8_HDLC_CRC_MODE_32_BIT	HDLC CRC mode.	

Name	Options		Description
diag_mode	HSS8_DIAG_NORMAL	Normal operation. Use this for external loopback .	Set diagnostic mode. External loopback - RS-422/485 : Connect TxD+ to RxD+, TxD- to RxD-, (CLK_OUT+ to CLK_IN+ and CLK_OUT- to CLK_IN- for synchronous mode). External loopback - RS-232 : Connect TxD to RxD, (CLK_OUT to CLK_IN for synchronous mode) and RTS to CTS and CD. Set diagnostic mode. For synchronous mode : see encoding_method .
	HSS8_DIAG_LOOPBACK	Internal loopback : TxD and RxD are connected internally. The value on RxD, CTS and CD is ignored. The transmitter and receiver share the same clock source.	
	HSS8_DIAG_ECHO	The transmitter automatically resends received data bit-by-bit.	
	HSS8_DIAG_LOOPBACK_ECHO	Loopback and echo operation occur simultaneously.	
max_receive_bytes	1 to (2 048 - CRC bytes (2 or 4)) (default) or up to (32 kBytes - CRC bytes (2 or 4)), depending on how many bytes have been allocated to the RX and TX buffers (See function hss8Create_device(..)).		Maximum number of bytes to receive before closing buffer. Set equal to max_frame_bytes.
max_frame_bytes	1 to 2 048 (default) or up to 32 kBytes, depending on how many bytes have been allocated to the RX and TX buffers (See function hss8Create_device(..)).		Maximum number of bytes per frame. Set equal to the number of data bytes plus the number of CRC bytes (either two or four) per frame.
address_mask	0x0000 - 0xFFFF		HDLC address mask. A one enables comparison and a zero masks it.
address1, address2, address3, address4	0x0000 - 0xFFFF		Four address registers for address recognition. The SCC reads the frame address from the HDLC receiver, compares it with the address registers, and masks the result with address_mask. For example, to recognize a frame that begins 0x7E (flag), 0x68, 0xAA, using 16-bit address recognition, the address registers should contain 0xAA68 and address_mask should contain 0xFFFF. For 8-bit addresses, clear the eight high-order address bits.
nr_flags_between_frames	0 - 15		Minimum number of flags between or before frames.
retransmit_enabled	TRUE FALSE		Enable re-transmit.
flag_sharing_enabled	TRUE FALSE		Enable flag sharing.
rx_disabled_during_tx	TRUE FALSE		Disable receive during transmit.
bus_mode	TRUE FALSE		Enable bus mode.
bus_mode_rts	TRUE FALSE		Enable special RTS operation in HDLC bus mode.

Name	Options	Description
multiple_tx_frames	TRUE FALSE	Enable multiple frames in transmit FIFO.
encoding_method	HSS8_HDLC_ENCODING_METHOD_NRZ HSS8_HDLC_ENCODING_METHOD_NRZI_MARK HSS8_HDLC_ENCODING_METHOD_NRZI_SPACE HSS8_HDLC_ENCODING_METHOD_FM0 HSS8_HDLC_ENCODING_METHOD_FM1 HSS8_HDLC_ENCODING_METHOD_MANCHESTER HSS8_HDLC_ENCODING_METHOD_DIFF_MANCHESTER	RX / TX encoding method. NRZ and NRZI use no DPLL. FM0/1, Manchester and Diff_Manchester use the DPLL for clock recovery. The clock rate is 16x when the DPLL is used.
preamble_length	HSS8_DPLL_PREAMBLE_LENGTH_0 HSS8_DPLL_PREAMBLE_LENGTH_8 HSS8_DPLL_PREAMBLE_LENGTH_16 HSS8_DPLL_PREAMBLE_LENGTH_32 HSS8_DPLL_PREAMBLE_LENGTH_48 HSS8_DPLL_PREAMBLE_LENGTH_64 HSS8_DPLL_PREAMBLE_LENGTH_128	Determines the length of the preamble pattern.
preamble_pattern	HSS8_DPLL_PREAMBLE_PATTERN_00 HSS8_DPLL_PREAMBLE_PATTERN_10 HSS8_DPLL_PREAMBLE_PATTERN_01 HSS8_DPLL_PREAMBLE_PATTERN_11	Determines what bit pattern precedes each TX frame.
send_idles_or_flags	HSS8_HDLC_SEND_IDLES HSS8_HDLC_SEND_FLAGS_SYNC	Send either idles or flags/syncs between frames as defined by the protocol. For HDLC the flag is defined as 0x7E. NRZI encoding methods may only be used with flags/syncs.

6.3.2.3 Preamble Requirements

Decoding Method	Preamble Pattern	Minimum Preamble Length Required
NRZI Mark	All zeros	8-bit
NRZI Space	All ones	8-bit
FM0	All ones	8-bit
FM1	All zeros	8-bit
Manchester	101010...10	8-bit
Differential Manchester	All ones	8-bit

6.3.3 BISYNC Mode

This protocol may only be used with the eight SCC channels : Channels A - H.

6.3.3.1 BISYNC Protocol Information Structure

The following structure is defined in the file hss8Controllfc.h and is given here in abbreviated format (i.e. reserved and obsolete members are not shown). Always use the structure as defined in hss8Controllfc.h.

```
struct hss8BisyncInfoStruct
{
    hss8UINT32 baud_rate;
    hss8UINT32 clock_source;
    hss8UINT32 max_receive_bytes;
    hss8UINT32 min_no_sync_pairs;
    hss8UINT32 crc_select;
    hss8UINT32 receive_bcs;
    hss8UINT32 rx_transparant_mode;
    hss8UINT32 reverse_data;
    hss8UINT32 disable_rx_while_tx;
    hss8UINT32 rx_parity;
    hss8UINT32 tx_parity;
    hss8UINT32 diag_mode;
    hss8UINT32 crcc;
    hss8UINT32 prcrc;
    hss8UINT32 ptcrc;
    hss8UINT32 parec;
    hss8UINT32 bsync;
    hss8UINT32 bdle;
    hss8UINT32 cc[8];
    hss8UINT32 rccm;
    hss8UINT32 sync;
    hss8UINT32 syn_length;
    hss8UINT32 send_idles_or_flags;
};
typedef struct hss8BisyncInfoStruct hss8BisyncInfo;
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 42 of 51

6.3.3.2 BISYNC Protocol Information Structure Members

Name	Options	Description
baud_rate	<p>1 200 - 1 Mbit/s (RS-232) 1 200 - 16 Mbit/s (RS-422/485)</p> <p>Any values permissible.</p> <p>The equation to calculate the actual baud rate for BISYNC is as follows :</p> $\text{Actual baud rate} = 100 \text{ MHz} / \text{ROUND}(100 \text{ MHz} / \text{Desired baud rate})$ <p>Where ROUND() implies that the result is rounded to the nearest integer.</p>	<p>Used to specify a single baud rate for both transmitter and receiver.</p> <p>Units in bit/s.</p>
clock_source	HSS8_CLOCK_DEFAULT	<p>HSS8_CLOCK_DEFAULT connects BRGs [1 - 4] to Channels [A - D] and Channels [E - H].</p>
	<p>HSS8_CLOCK_BRG1 HSS8_CLOCK_BRG2 HSS8_CLOCK_BRG3 HSS8_CLOCK_BRG4</p>	<p>BRGs [1 - 4]. BRG1 for Channels [A and E] BRG2 for Channels [B and F] BRG3 for Channels [C and G] BRG4 for Channels [D and H]</p>
	<p>HSS8_CLOCK_EXT1 HSS8_CLOCK_EXT2 HSS8_CLOCK_EXT3 HSS8_CLOCK_EXT4</p>	<p>External Clocks connected on CLK_IN Pins.</p> <p>Note : HSS8_CLOCK_EXT[1 - 2] can only be used for Channels [A and B] and [E and F], while HSS8_CLOCK_EXT[3 - 4] can only be used for Channels [C and D] and [G and H].</p>
max_receive_bytes	1 to (2 048 - 2 CRC bytes) (default) or up to (32 kBytes - 2 CRC bytes), depending on how many bytes have been allocated to the RX and TX buffers (See function hss8Create_device(..)).	Maximum number of bytes to receive before closing buffer.
min_no_sync_pairs	0b0000 (0 pairs) - 0b1111 (16 pairs)	Minimum number of SYN1-SYN2 pairs sent between or before messages. The entire pair is always sent, regardless of the syn_length variable.
crc_select	<p>HSS8_BISYNC_CRC_MODE_16 HSS8_BISYNC_CRC_MODE_LRC</p>	<p>CRC selection.</p> <p>1. CRC16 (X16 + X15 + X2 + 1) : Initialise prcrc and ptcrc to all zeros or all ones.</p> <p>2. LRC (sum check) : For even LRC, initialise prcrc and ptcrc to zeros, for odd LRC initialise to ones.</p>
receive_bcs	TRUE FALSE	Enable Receive Block Check Sequence (BCS).

Name	Options		Description
rx_transparent_mode	TRUE FALSE		Enable Receiver transparent mode. FALSE : Normal receiver mode with SYNC stripping and control character recognition. TRUE : Transparent receiver mode. SYNC's, DLE's and control characters are recognised only after the leading DLE character. The receiver calculates the CRC16 sequence even if it is programmed to LRC while in transparent mode. Initialize prcrc to the CRC16 preset value before setting rx_transparent_mode .
reverse_data	TRUE FALSE		Enable Reverse data.
disable_rx_while_tx	TRUE FALSE		Disable receiver while sending.
rx_parity tx_parity	HSS8_BISYNC_PARITY_ODD HSS8_BISYNC_PARITY_LOW HSS8_BISYNC_PARITY_EVEN HSS8_BISYNC_PARITY_HIGH		Receive and transmit parity. Parity is ignored unless crcc_select = LRC.
diag_mode	HSS8_DIAG_NORMAL	Normal operation. Use this for external loopback.	Set diagnostic mode. External loopback - RS-422/485 : Connect TxD+ to RxD+, TxD- to RxD-, CLK_OUT+ to CLK_IN+ and CLK_OUT- to CLK_IN-. External loopback - RS-232 : Connect TxD to RxD, CLK_OUT to CLK_IN and RTS to CTS and CD.
	HSS8_DIAG_LOOPBACK	Internal loopback : TxD and RxD are connected internally. The value on RxD, CTS and CD is ignored. The transmitter and receiver share the same clock source.	
	HSS8_DIAG_ECHO	The transmitter automatically resends received data bit-by-bit.	
	HSS8_DIAG_LOOPBACK_ECHO	Loopback and echo operation occur simultaneously.	
crcc	0		CRC constant value.
prcrc ptcrc	0x0000 or 0xFFFF		Preset receiver / transmitter CRC16 / LRC. These values should be preset to all ones or zeros, depending on the BCS used.
parec	0 - 65 535		Number of received parity errors.

Name	Options	Description
bsync	0bv0000000ssssssss	<p>BISYNC SYNC register. Contains the value of the SYNC character stripped from incoming data on receive once the receiver synchronizes to the data using the SYN1- SYN2 pair.</p> <p>v - If v = 1 and the receiver is not in hunt mode when a SYNC character is received, this character is discarded.</p> <p>ssssssss (8 bits) - SYNC character. When using 7-bit characters with parity, the parity bit should be included in the SYNC register value.</p>
bdle	0bv0000000ddddddd	<p>BISYNC DLE register. In transparent mode, the receiver discards any DLE character received.</p> <p>v - If v = 1 and the receiver is not in hunt mode when a DLE character is received, this character is discarded.</p> <p>ddddddd (8 bits) - DLE character. This character tells the receiver that the next character is text.</p>
cc[8]	<p>0b0bh-----ccccccc - Valid entry. 0b1bh-----ccccccc - Entry not valid and is not used.</p>	<p>Control characters 1 to 8.</p> <p>----- (5 bits) - Reserved. Initialise to zero.</p> <p>b - Block check sequence expected. A maskable interrupt is generated after the buffer is closed.</p> <p>b = 0 : The character is written into the receive buffer and the buffer is closed.</p> <p>b = 1 : The character is written into the receive buffer. The receiver waits for 1 LRC or 2 CRC bytes and then closes the buffer.</p> <p>h - Enables hunt mode when the current buffer is closed.</p> <p>h = 0 : The BISYNC controller maintains character synchronisation after closing the buffer.</p> <p>h = 1 : The BISYNC controller enters hunt mode after closing the buffer. When b = 1, the controller enters hunt mode after receiving LRC or CRC.</p> <p>ccccccc (8 bits) - Defines control characters to be compared to the incoming character. When using 7-bit characters with parity, include the parity bit in the character value.</p>

Name	Options	Description
rccm	0b11-----00000000 - Ignore these bits when comparing incoming character. 0b11-----11111111 - Enable comparing the incoming character to cc[n].	Receive control character mask. A one enables comparison and a zero masks it.
sync	0xssss (2 bytes)	SYNC character : Should be programmed with the sync pattern.
syn_length	HSS8_BISYNC_SYNL_8 HSS8_BISYNC_SYNL_16	HSS8_BISYNC_SYNL_8 : Should be chosen to implement mono-sync protocol. The receiver synchronizes on an 8-bit sync pattern in sync . HSS8_BISYNC_SYNL_16 : The receiver synchronizes on a 16-bit sync pattern stored in sync .
send_idles_or_flags	HSS8_BISYNC_SEND_IDLEES HSS8_BISYNC_SEND_FLAGS_SYNCES	Send either idles or flags/syncs between frames as defined by the protocol. The flag character is equal to sync.

6.3.4 SMC UART Mode

This protocol may only be used with the four SMC channels : Channels I - L.

6.3.4.1 SMC UART Protocol Information Structure

The following structure is defined in the file hss8Controllfc.h and is given here in abbreviated format (i.e. reserved and obsolete members are not shown). Always use the structure as defined in hss8Controllfc.h.

```
struct hss8SmcUartInfoStruct
{
    hss8UINT32 max_receive_bytes;
    hss8UINT32 max_idl;
    hss8UINT32 data_bits;
    hss8UINT32 stop_bits;
    hss8UINT32 parity_enable;
    hss8UINT32 parity_mode;
    hss8UINT32 diag_mode;
    hss8UINT32 baud_rate;
};
typedef struct hss8SmcUartInfoStruct hss8SmcUartInfo;
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 47 of 51

6.3.4.2 SMC UART Protocol Information Structure Members

Name	Options		Description
baud_rate	1 200 - 115.2 kbit/s (RS-232/RS-422/485) Any values permissible. The equation to calculate the actual baud rate for the SMC UART is as follows : $\text{Actual baud rate} = 100 \text{ MHz} / 16 / \text{ROUND}(100 \text{ MHz} / 16 / \text{Desired baud rate})$ Where ROUND() implies that the result is rounded to the nearest integer.		Used to specify a single baud rate for both transmitter and receiver. Units in bit/s.
stop_bits	HSS8_UART_STOP_BITS_ONE HSS8_UART_STOP_BITS_TWO		Number of full stop bits.
data_bits	HSS8_UART_DATA_BITS_5 HSS8_UART_DATA_BITS_6 HSS8_UART_DATA_BITS_7 HSS8_UART_DATA_BITS_8 HSS8_UART_DATA_BITS_9 HSS8_UART_DATA_BITS_10 HSS8_UART_DATA_BITS_11 HSS8_UART_DATA_BITS_12 HSS8_UART_DATA_BITS_13 HSS8_UART_DATA_BITS_14		Number of data bits. Note only Channels I - H (i.e. the SMC channels) support nine or more data bits.
parity_enable	HSS8_UART_PARITY_NO_PARITY HSS8_UART_PARITY_ENABLE		Enable or disable parity checking.
parity_mode	HSS8_UART_SMC_PARITY_ODD HSS8_UART_SMC_PARITY_EVEN		Receive and transmit parity. Parity will only be checked if parity is enabled.
diag_mode	HSS8_DIAG_NORMAL	Normal operation. Use this for external loopback .	Set diagnostic mode. External loopback - RS-422/485 : Connect TxD+ to RxD+ and TxD- to RxD-. External loopback - RS-232 : Connect TxD to RxD.
	HSS8_DIAG_LOOPBACK	Internal loopback : TxD and RxD are connected internally. The value on RxD is ignored.	
	HSS8_DIAG_ECHO	The transmitter automatically resends received data bit-by-bit.	
	HSS8_DIAG_LOOPBACK_ECHO	Loopback and echo operation occur simultaneously.	
max_receive_bytes	1 to 2 048 (default) or up to 32 kBytes, depending on how many bytes have been allocated to the RX and TX buffers (See function hss8Create_device(..)).		Maximum number of bytes that may be copied into a buffer.
max_idle	0 to 2 048 (default) or up to 32 kBytes, depending on how many bytes have been allocated to the RX and TX buffers (See function hss8Create_device(..)).		Maximum idle characters. When a character is received, the receiver begins counting idle characters. If max_idle idle characters are received before the next data character, an idle timeout occurs and the buffer is closed. Thus, max_idle offers a way to demarcate frames. To disable the feature, clear max_idle. The bit length of an idle character is calculated as follows : $1 + \text{data length (5-14)} + 1 \text{ (if parity is used)} + \text{number of stop bits (1 - 2)}$ For 8 data bits, no parity, and 1 stop bit, the character length is 10 bits.

7. **Getting Started**

After installing the HSS8 VxWorks Software Driver according to Paragraph 4, test the host software driver following the test procedure given in hss8Test.txt.

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 49 of 51

8. **Contact Details**

8.1 Contact Person

Direct all correspondence and / or support queries to the Project Manager at C²I² Systems.

8.2 Physical Address

C²I² Systems
Real-Time House
Greenford Office Estate
Punters Way
Kenilworth
Cape Town
7708
South Africa

8.3 Postal Address

C²I² Systems
P.O. Box 171
Rondebosch
7701
South Africa

8.4 Voice and Electronic Contacts

Tel : (+27) (0)21 683 5490
Fax : (+27) (0)21 683 5435
Email : info@ccii.co.za
Email : support@ccii.co.za
WWW : <http://www.ccii.co.za/>

8.5 Product Support

Support on C²I² Systems products is available telephonically between Monday and Friday from 09:00 to 17:00 CAT. Central African Time (CAT = GMT + 2).

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 50 of 51

Annexure A

Making Changes to sysLib.c for x86

The PCI free memory space may need to be defined in the memory descriptor table. Consult the relevant reference manual and obtain the upper address of the PCI memory. Allocate at least 5 MBytes of memory per HSS8 Adapter. Subtract that amount from the upper address of the PCI memory, and use this value as the base of the PCI memory space.

Note : If there are other devices on the PCI bus, it may be necessary to allocate more memory.

Example : For 2 HSS8 Adapters, allocate 10 MBytes of memory. If the upper address of the PCI memory space is defined as 0xFFFF0000, then subtracting 10 MBytes gives a base address of :

0xFFFF0000 - 0xA00000 = 0xFF500000.

In the PC 386/486/Pentium/Pentiumpro system-dependent library (sysLib.c), code (**shown in bold text**) needs to be added to the memory descriptor table, sysPhysMemDesc[] :

```
#ifndef CPU_PCI_MEM_ADRS
#define CPU_PCI_MEM_ADRS      0xFF500000  /* base of PCI MEM addr */
#endif

PHYS_MEM_DESC sysPhysMemDesc [] =
{
    /* adrs and length parameters must be page-aligned (multiples of 4KB/4MB) */
    #if(VM_PAGE_SIZE == PAGE_SIZE_4KB)

        /* lower memory */
        ...
        /* video ram, etc */
        ...
        /* upper memory for OS */
        ...
        /* upper memory for Application */
        ...
        /* PCI I/O space */
        {
            (void *) CPU_PCI_MEM_ADRS,
            (void *) CPU_PCI_MEM_ADRS,
            (0xA00000),
            VM_STATE_MASK_VALID | VM_STATE_MASK_WRITABLE |
            VM_STATE_MASK_CACHEABLE, VM_STATE_VALID | VM_STATE_WRITABLE |
            VM_STATE_CACHEABLE_NOT
        },

        /* entries for dynamic mappings - create sufficient entries */
        DUMMY_MMU_ENTRY,
        DUMMY_MMU_ENTRY,
        DUMMY_MMU_ENTRY,
        ...
        ...

    #else
        ...
    #endif
}
```

CCII/HSS8/6-MAN/002	2015-09-18	Issue 1.6
CH8MAN02.WPD		Page 51 of 51