



SPHERES

Expansion Port v2

Interface Control

Document

Prepared By

MIT Space Systems Lab

and

Aurora Flight Sciences Corp
Research and Development Center

Aurora Report Number: AR13-267

Inquiries to:
John F. Merk
Aurora Flight Sciences
Four Cambridge, 11th Floor
Cambridge, MA 02142

Office: 617-500-4887
Fax: 617-500-4810
Email: jmerk@aurora.aero

TABLE OF CONTENTS

A	Revision History	3
1	Introduction	4
2	Purpose	4
3	Mechanical Interface	5
4	Electrical Interface	6
4.1	Overview	6
4.2	Connectors	6
4.3	Pin Assignment	7
4.4	Power Supplies	10
4.5	Serial Communication and GPIO	11
4.6	RESET Signal	11
4.7	US/IR Bypass	12
5	Software Interface	14
5.1	GSP Expansion Port API	14
5.2	Overview	14
5.2.1	Callback Function Types	14
5.3	Configuration Structure	15
5.4	General Functions	16
5.5	UART Functions	16
5.6	GPIO Functions	18
6	Appendix A: Mechanical Interface Drawing	1
7	Appendix B: EXP_v2 Header & Code Example	1
7.1	EXP_v2 Header	1
7.2	Example GSP.c source code – UART Communication	3

A Revision History

Version	Date	Comments
0.7 (ASO)	2004-02-24	Preliminary draft
0.9 (ASO)	2004-02-25	Continued revisions before initial release: added measurements to mechanical section and part number to the power line fuses. Added block diagram of serial line. Re-arranged some sections for better flow.
0.9a (JGK)	2010-01-5	Removed global data bus section. Not functional on flight SPHERES.
2.0 beta (JM, AJB, BET, ASO)	2011-Dec-12	Major revision for the Expansion Port Version 2 (ExpV2) additional hardware. Describes new interfaces for the multiple ports and DIO now available. Explains the communications protocol between the ExpV2 and SPHERES Core.
2.1 beta (AJB)	2012-Mar-9	Updated pinout table. Corrected references to hardware handshaking (RTS/CTS). Hardware handshaking is not supported anymore.

1 Introduction

The SPHERES Satellites have been equipped with an enhanced expansion port interface that improves the functionality and utility of the original port.

This enhanced port has been permanently installed over the existing port and is comprised of four items; a PEEK Board Carrier, aluminum Mounting Plate, Expansion PCB (conformal coated), and polycarbonate Expansion PCB Cover (when port is not in use). Figure 1 shows the parts that make up the expansion port and a front view of the port with the cover removed.

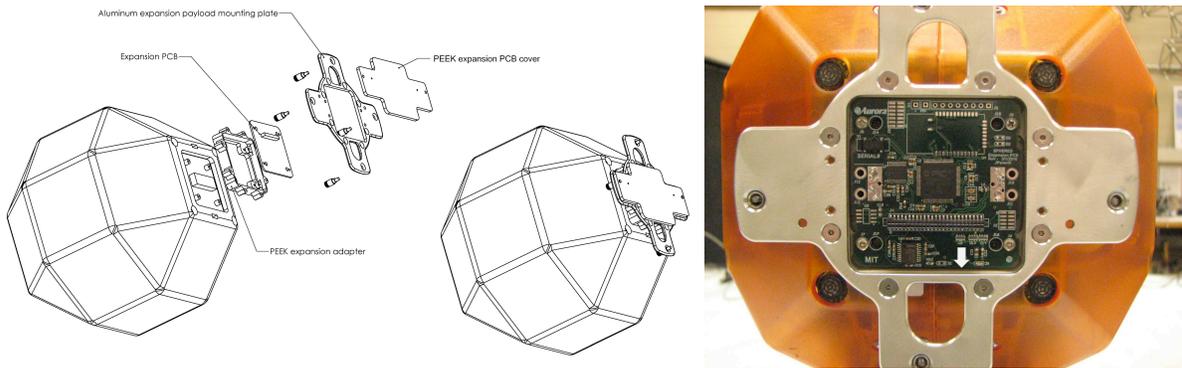


Figure 1 - SPHERES Expansion Port (left: port hardware, right: with cover removed)

2 Purpose

This document is intended to provide the guest scientist with all the information needed to interface an Expansion Item (or Payload) with the SPHERES expansion port. This document describes the power and data I/O interfaces available in the expansion port, the GSP software interface, and the structural connections to mount an expansion item to a SPHERES satellite.

3 Mechanical Interface

The expansion port has a 0.156" thick aluminum plate (referred to as the 'mounting plate') on which to mount the Expansion Item (payload). Four captive thumbscrews on the mounting plate can be used to rapidly attach and remove expansion items. When designing the expansion item interface, care must be taken to ensure proper alignment with the captive screws and expansion connector. Not all thumbscrews must be used to secure the Expansion Item, however a minimum of two is recommended.

The mounting plate, shown in Figure 2, is designed to not interfere with the existing metrology sensors on the SPHERES satellite. If possible, care should be taken to avoid the keepout zones for these sensors. Figure 6 details the keepout zone locations and section 4.7 describes the requirements levied on the Expansion Item if blocking the metrology sensors cannot be avoided.

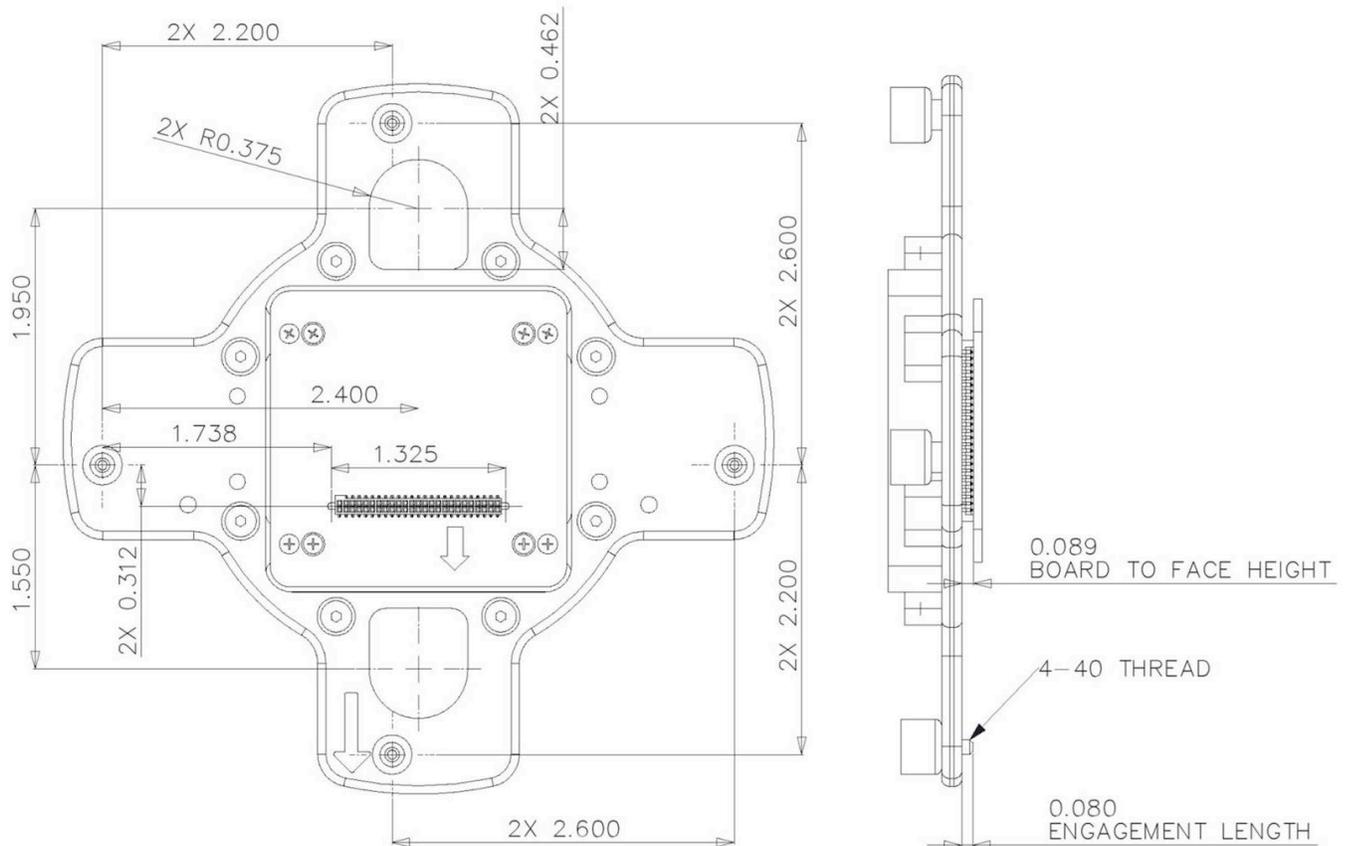


Figure 2 - Expansion Port Mechanical Interface for TFM-125-02-L-D-A Mating Connector

The electrical connection between the satellite and the Expansion Item is detailed in section 4. The location of the mating connector, relative to the mounting plate features, is shown in Figure 2.

4 Electrical Interface

4.1 Overview

The expansion port electrical interface consists of a single 50-pin connector (see Table 1) that carries control and power signals to allow an Expansion Item (payload) to access the internal electronics of the SPHERES satellite. The expansion port power and communication signals to/from the external payload include:

- $\pm 15\text{VDC}$, $+5\text{VDC}$, $+3.3\text{VDC}$ and Ground
- 2 RS-232 serial ports at EIA standard voltage levels (no hardware handshaking)
- 16 GPIO lines (independently configurable as inputs or outputs)
- Bypass US/IR signals in the event the expansion item covers the sensors on the +X face of the satellite.

4.2 Connectors

The expansion port mating connectors are manufactured by Samtec. The connectors can be obtained directly from Samtec or an authorized distributor, however to ensure available stock for many years, Aurora Flight Sciences has acquired several Expansion side (plug) connectors for use in developing expansion items. Scientists interested in creating their own expansion item can contact Aurora for further information on obtaining the necessary connector. The satellite side connector is shown in Table 1.

Table 1 - Expansion Port Connector Part Number (Satellite)

Side	Connector
Satellite	Samtec SFM-125-L2-S-D-A

The SPHERES socket connector is recessed below the expansion port cover; the Expansion Item mating connector must therefore extend out from the edge of the expansion circuit board to plug into the SPHERES socket connector. The expansion item must be designed to ensure accurate spacing between the edge of the expansion item (that secures to the port thumbscrews) and the circuit board which contains the expansion mating connector.

The distance between the edge of the mounting plate and the expansion circuit board is dependent on the selected mating connector. There are four discrete options, shown in Table 2. Both connector genders are shown in Figure 3.

Table 2 - Expansion Port Connector Part Numbers and Mating Heights (Expansion Item)

Side	Connector Part Number and Notch	Expansion Circuit Board-to-Mounting Plate Distance
Expansion Item	Samtec TFM-125-02-S-D-A	0.089"
	Samtec TFM-125-12-S-D-A	0.159"
	Samtec TFM-125-22-S-D-A	0.229"
	Samtec TFM-125-32-S-D-A	0.304"

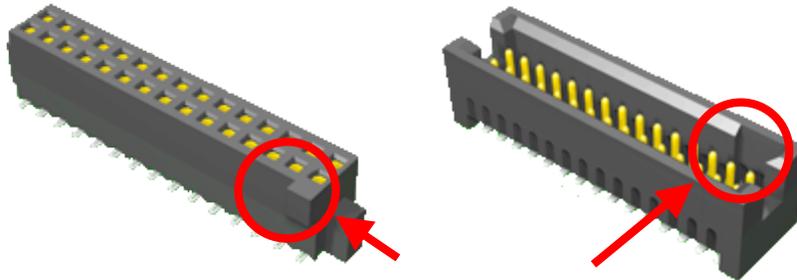


Figure 3 - Samtec Connectors (Left: Satellite Side SFM-125-L2-S-D-A , Right: Expansion Side TFM-125-*2-S-D-A)

Figure 2 shows an example of the Expansion board-to-mounting plate separation for the TFM-125-02-L-D-A connector.

The connector has an alignment pin (see Figure 3) and the connector itself is offset from the centerline of the port to ensure the expansion item cannot be inserted upside-down.

4.3 Pin Assignment

Figure 4 and Table 3 show the expansion port signals and connector pinout.

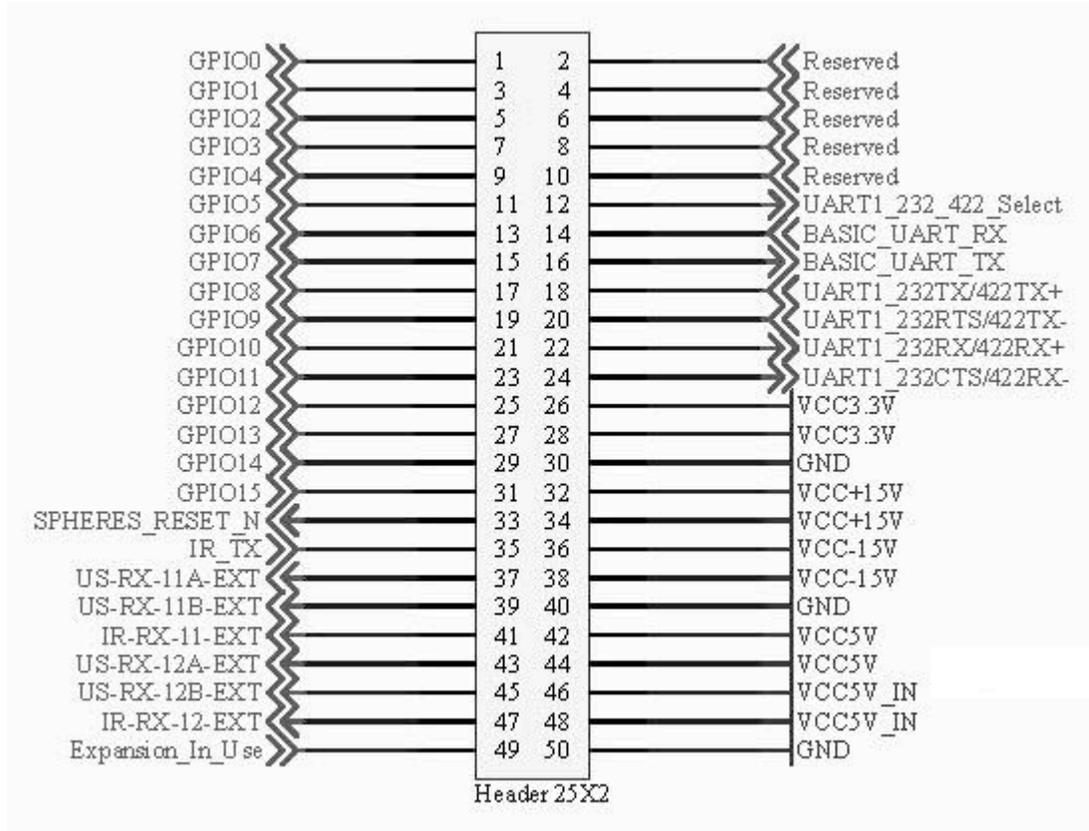


Figure 4 – Expansion Port Pin Assignment

Table 3 - Expansion Port Connector Pinout

SIGNAL NAME	PIN#	SYMBOL	I/O	DESCRIPTION
VCC3.3V	26,28	VCC3.3V	0	Regulated +3.3V power supply to payload. Current from this supply plus the VCC5V supply is limited to 0.5A total
GND	30,40,50	GND	-	System Ground. All GND pins are tied together at the expansion board
VCC+15V	32,34	VCC+15V	0	Unregulated +15V power supply to payload. A 0.5A fuse prevents damaging internal SPHERES components. This supply is shared with internal SPHERES avionics
VCC-15V	36,38	VCC-15V	0	Unregulated -15V power supply to payload. A 0.5A fuse prevents damaging internal SPHERES components. This supply is shared with internal SPHERES avionics
VCC5V	42,44	VCC5V	0	Expansion board processor power in. Pins 42,44,46,48 must be tied together on the payload
VCC5V_IN	46,48	VCC5V_IN	1	Unregulated +5V power supply to payload. A 0.5A fuse prevents damaging internal SPHERES components. This power is shared with internal SPHERES avionics, and also powers the +3.3V regulator.
GPIO0	1	GPIO[15:0]	I/O	Bits 0-15, 16-bit General Purpose I/O. All bits are independently settable as either Inputs or Outputs and are addressed through the satellite UART. The GPIO HIGH voltage level is 3.3VDC
GPIO1	3			
GPIO2	5			

SIGNAL NAME	PIN#	SYMBOL	I/O	DESCRIPTION
GPIO3	7			
GPIO4	9			
GPIO5	11			
GPIO6	13			
GPIO7	15			
GPIO8	17			
GPIO9	19			
GPIO10	21			
GPIO11	23			
GPIO12	25			
GPIO13	27			
GPIO14	29			
GPIO15	31			
SPHERES_RESET_N	33		0	RESET OUT to Expansion Item. The signal is negative logic (active low); it is triggered by the watchdog or the reset button on the satellite.
IR_TX	35		0	A command from the satellite to the Expansion Item to transmit IR (used when the expansion item covers the US/IR of the satellite).
US-RX-11A-EXT	37		I	Ultrasound and infrared signals received by the Expansion Item to be used instead of the internal US/IR receivers in the satellite. These signals are only used when EXPANSION_IN_USE is HI.
US-RX-11B-EXT	39		I	
IR-RX-11-EXT	41		I	
US-RX-12A-EXT	43		I	
US-RX-12B-EXT	45		I	
IR-RX-12-EXT	47		I	
EXPANSION_IN_USE	49		I	
UART1_232_422_SELECT	12		I	UART1 RS232/422 Select line. The Expansion Item must set this pin LOW for RS232
UART0_RX	14		I	RS-232 compliant UART0 Receive. Connect to Expansion Item transmit.
UART0_TX	16		0	RS-232 compliant UART0 Transmit. Connect to Expansion Item receive
UART1_232TX/422TX+	18		0	RS-232 compliant UART1 Transmit/Receive signals. UART1 can be configured for full-duplex RS-422. The UART1_232_422_SELECT line determines operation of port.
UART1_422TX-	20		I	
UART1_232RX/422RX+	22		I	
UART1_422RX-	24		0	

SIGNAL NAME	PIN#	SYMBOL	I/O	DESCRIPTION
DO NOT USE (Reserved for Future Expansion)	2,4,6,8,10	DNC	-	These pins currently have no function on the expansion board. To ensure compatibility with future expansion board revisions, leave these pins unconnected.

4.4 Power Supplies

The expansion port provides four power supplies and one common ground. The +5VDC and ± 15 VDC supplies come directly from the satellite, and are individually fused inside the satellite, upstream of the expansion port. The fuses have a non-trivial / non-constant resistance, so they must be treated as unregulated supplies. The +3.3VDC is regulated down from the +5VDC on the expansion port PCB via a low-dropout linear regulator. This is the only regulated supply, however the available current is shared with the +5VDC supply.

The ± 15 VDC can be regulated down on the expansion item to provide other voltages as needed; the -15VDC should not be used for voltages below -12VDC. The internal SPHERES supply ratings are given in Table 4.

Table 4 - Power Supplies available on SPHERES Expansion Port

Supply	Signal Name	Regulation	Current @ Exp Port	Comments
+3.3VDC	VCC3.3V	Regulated	0.5A	Current shared with VCC5V
+5VDC	VCC5V	Unregulated	0.5A	Current shared with VCC3.3V
+15VDC	VCC+15V	Unregulated	0.5A	
-15VDC	VCC-15V	Unregulated	0.5A	Do not regulate below -12VDC

The available power at the Expansion Port has been rated to prevent overload of the supplies since all of them are used internally by the satellite. All the internal fuses are PolySwitch (PPTC) resettable surface mount components (Tyco Electronics/Raychem Circuit Protection part # miniSMDC050). The PPTC devices trigger on temperature rise in the device, therefore the expansion item must account for further temperature increase if it creates heat. The fuse self-resets once its temperature returns to normal.

4.5 Serial Communication and GPIO

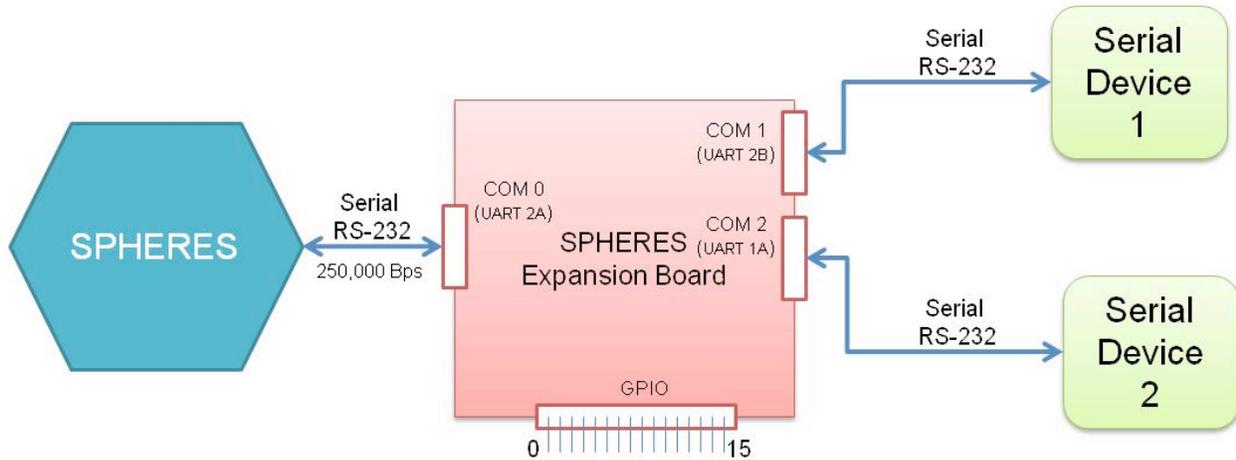


Figure 5 – Expansion Port Communication and I/O

All communication to/from the satellite to the expansion port printed circuit board is routed through a single serial port with a default baud rate of 250Kbps (refer to Figure 2). The I/O configuration settings are programmed by the satellite on power-up to the default settings shown in Table 2. These settings can be changed by the SPHERES satellite. Once the expansion port I/O is configured, data through the expansion port to the Guest Scientist API is seamless, the satellite can communicate directly with the attached payload(s).

Table 5 – UART & GPIO Default Settings

Setting	Default Value
SPHERES Port Baud Rate	250,000 bps
Serial Device 1 Baud Rate	115,200 bps
Serial Device 2 Baud Rate	115,200 bps
Data Bits	8 bits
Parity	No Parity
Stop Bits	1
GPIO Pin I/O Status	Outputs
GPIO ODC Status	Disabled

4.6 RESET Signal

The expansion port outputs the general SPHERES Reset signal to command the expansion item to reset when the SPHERES does. The **SPHERES_RESET_N** (pin 33) is a negative logic (active low) signal that is on for at least 100ms when active. The signal is triggered by either the user pressing the reset button (it remains low as long as the reset button is being pressed), or by the watchdog timing out (which creates a 100ms pulse).

4.7 US/IR Bypass

The SPHERES Satellites metrology use a combination of infrared (IR) output pulses and Ultrasound (US) input pulses to determine its position and attitude within the fixed reference frame provided by the SPHERES beacons. There are four US receivers and two IR transceivers on each face of the satellite, including the +X face with the expansion port. The mounting plate is already sized to avoid these sensors. If possible, care should be taken to prevent covering these keep-out areas with the expansion item.

The four US sensors are in a square pattern with the centers at a distance of 2.19" from the center of the expansion port. The two IR sensors are behind the two mounting plate cutouts, at a distance of 1.7" from the center of the expansion port.

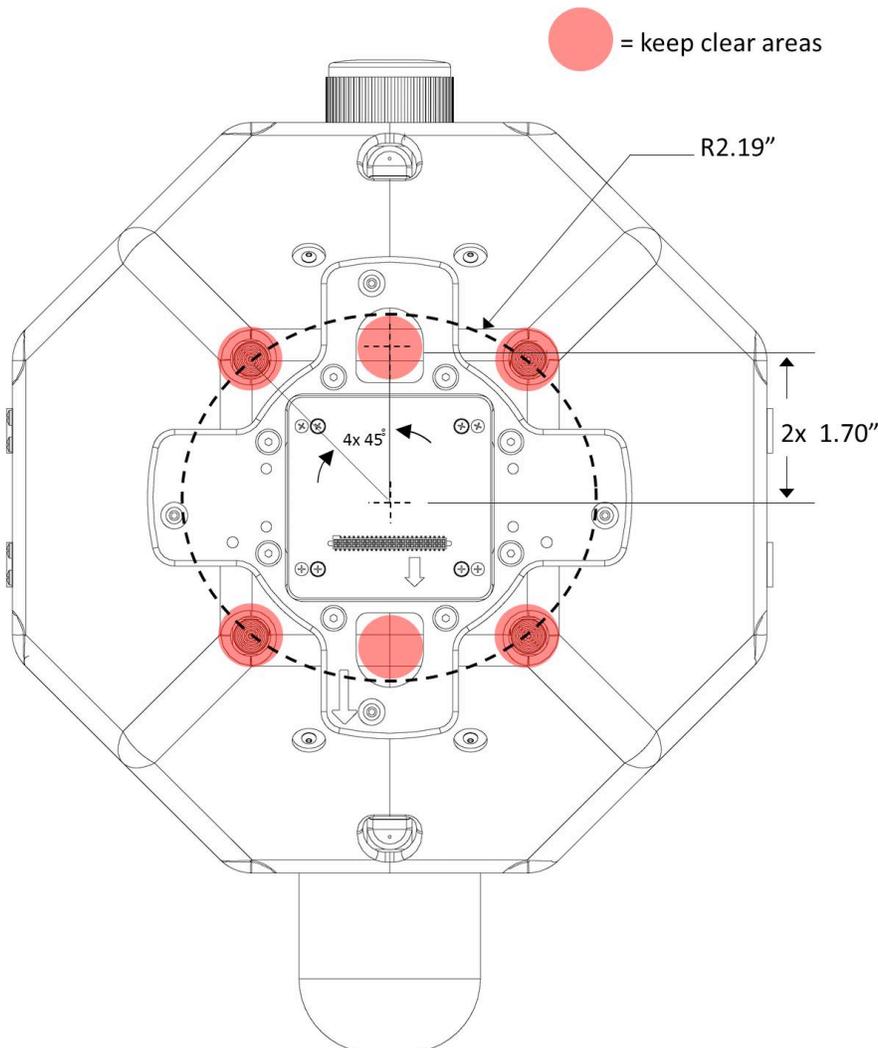


Figure 6 - US / IR Keepout Zones

When the expansion item must cover the ultrasound/infrared sensors of the +X face of the SPHERES satellite, the expansion item should replace those signals with metrology sensors on its own hardware, or at least indicate to the satellite that the sensors are covered. To this purpose the **EXPANSION_IN_USE** signal is present at pin 49 of the

expansion board. This is a positive logic signal which tells the DSP and other hardware in the satellite whether the US/IR sensors are being covered or not.

If the signal is driven high by the attached device (active) the internal SPHERES hardware will bypass via a multiplexer the signals from the internal metrology boards and instead send to the DSP the metrology signals present on the expansion board. If the expansion item does not replace the signals correctly, then the lines should be tied to either VCC5V or Ground as follows:

Table 6 - US / IR Inactive Connections

SIGNAL	PIN	INACTIVE
US-RX	37,39,43,45	GND
IR-RX	41,47	VCC5V (+5VDC)

If the expansion item does not cover the US/IR sensors, the **EXPANSION_IN_USE** input should be left as a no-connect or tied to GND.

The IR_TX signal (pin 35) can always be used, regardless of the state of the **EXPANSION_IN_USE** pin. This signal indicates that an IR signal should be transmitted. In the case where the expansion item covers the US/IR sensors, it should also provide replacement IR transmitters and use this signal to trigger them. This signal is active low. The US/IR bypass circuitry in the satellite is shown in Figure 7.

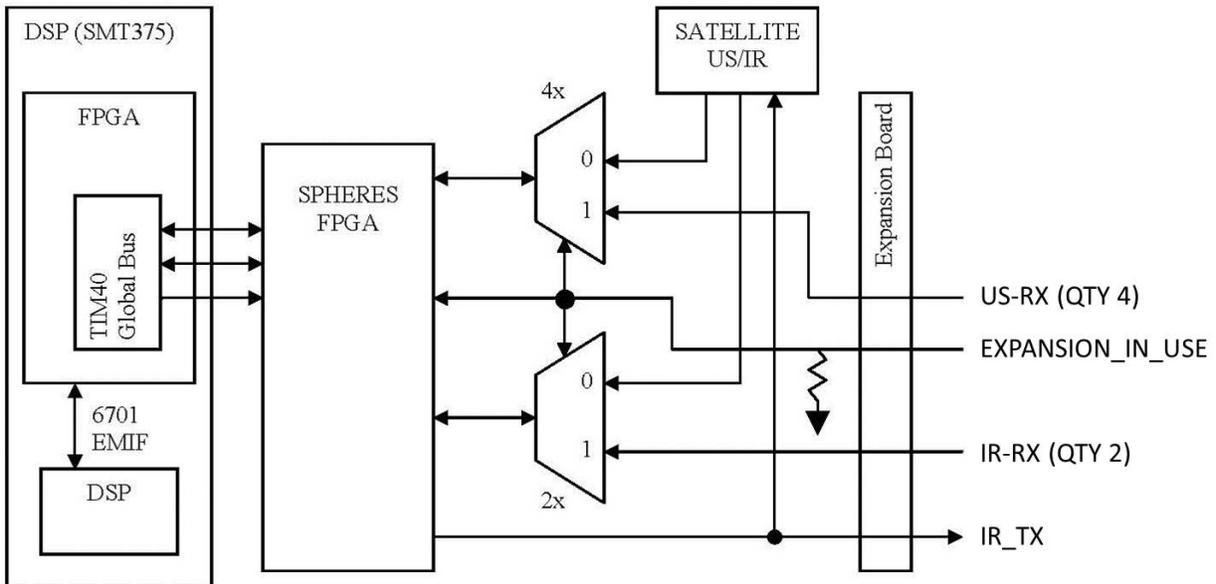


Figure 7 - US/IR Bypass Signals Block Diagram

5 Software Interface

5.1 GSP Expansion Port API

5.2 Overview

The Spheres Expansion Port software provides support for communicating with two serial ports and 16 General Purpose I/O (GPIO) pins on the SPHERES Expansion board. The GSP Interface with the expansion port allows the SPHERES satellite to read and write to a serial device and set the baud rate. The SPHERES satellite will also be able to configure the GPIO pins to be either input or output pins and will be able to read or set the pin values.

The Expansion Port interface is asynchronous between the expansion board and the SPHERES DSP. To handle this asynchronicity the user will write callback functions that are executed whenever the SPHERES satellite receives data from the expansion board. Additionally, the expansion board configuration settings are stored locally in a configuration structure for synchronous access by the user as needed.

The GSP Interface functions and type definitions are described in their relevant sections. There are 5 sections:

- **Callback Function Definitions**
 - Type definition information
- **Configuration Structure Definition**
- **General Function Definitions**
 - Initialization, reset, and general configuration query functions
 - Configuration queries are handled by the user-defined callback
- **UART Function Definitions**
 - Setting configuration and the sending data
 - Receiving data is handled by the user-defined callback
- **GPIO Function Definitions**
 - Setting configuration and GPIO pin setting/writing.
 - Receiving pin values is handled by the user defined callback

5.2.1 Callback Function Types

There are three types of callback functions available to the user, for the three different types of data received by the satellite from the expansion port. The three functions are the UART callback, the GPIO callback, and the configuration callback. The user will define the callback function in either `gsp.c` or a user defined source file and then register the function with the system. To register a callback function, call the relevant registration function and provide it the address of, or pointer to, the user-defined callback function. The callback registration functions are defined at the end of their relevant sections.

The type definitions for these callback functions are below. The type definitions define the required arguments and the return type of the callback function. The user-defined callback function must comply with this format.

Type	Callback Function Type	Argument Types	Argument Names	Description
Void	EXPv2_UART_CBK	unsigned char	<i>channel</i>	<p>The user-defined function that will be executed every time the expansion port receives serial data.</p> <p>The function is provided with:</p> <ul style="list-style-type: none"> The source of the serial data A pointer to the data The length of the data (in bytes)
		unsigned char*	<i>data</i>	
		unsigned int	<i>len</i>	
Void	EXPv2_GPIO_CBK	unsigned char	<i>pin</i>	<p>The user-defined function that will be executed every time the expansion port receives GPIO data.</p> <p>The function is provided with:</p> <ul style="list-style-type: none"> The GPIO pin number The value of the GPIO pin
		unsigned char	<i>value</i>	
Void	EXP_v2_CFG_CBK	cfgStruct	<i>cfg</i>	<p>The user-defined function that will be executed every time the expansion port receives configuration data.</p> <p>The function is provided with:</p> <ul style="list-style-type: none"> The updated configuration structure The received data packet (contains what configuration parameter was changed)
		unsigned char*	<i>dataPacket</i>	

5.3 Configuration Structure

The configuration structure stores the configuration of the UART, GPIO and SPI modes of operation. At present, the SPI mode is not yet supported. Its final implementation will mirror that of the UART and GPIO functionality.

Type Name	Element Types	Element Names	Description
Void <i>cfgStruct</i>	unsigned int [3]	<i>uart_cfg</i>	The baud rate for each channel is stored in the configuration structure Channels 0. SPHERES Port 1. Serial Device 1 2. Serial Device 2
	unsigned short [3]	<i>gpio_cfg</i>	Each unsigned short is a string of 16 bits corresponding to the 16 GPIO pins. 0. TRIS bits a. 1 – Input b. 0 – Output 1. Open Drain Collector a. 0 – Disabled b. 1 – Enabled 2. SPI CS bits (Currently Unsupported)
	unsigned int [6]	<i>spi_cfg</i>	Currently Unsupported

5.4 General Functions

Type	Function Name	Argument Types	Argument Names	Description
void	<i>expv2_reset</i>	void	<i>void</i>	Reset the configuration structure and the local GPIO register.
void	<i>expv2_init</i>	void	<i>void</i>	Initialize the configuration structure to zeros. Initialize the callback function registers to NULL.
void	<i>expv2_cfg_cbk_register</i>	EXPv2_CFG_CBK	<i>callbackFunction</i>	Register a user-defined callback function with the expansion port. Every time configuration data is received from the expansion board, the callback function is executed and given the configuration data from the expansion board and the updated configuration structure. (See Type Definitions for callback function definition)
cfgStruct	<i>expv2_cfg_get</i>	void	<i>void</i>	Return the local version of the configuration structure. This may be outdated.

5.5 UART Functions

Type	Function Name	Argument Types	Argument Names	Description
void	<i>expv2_uart_send</i>	unsigned char	<i>channel</i>	Send a data packet through the UART on a particular channel.

Type	Function Name	Argument Types	Argument Names	Description
		unsigned char	<i>len</i>	The channels/recipients are 1. Serial Device 1 2. Serial Device 2
		unsigned char*	<i>data</i>	The max packet length is 255 bytes.
void	expv2_uart_baud_set	unsigned char	<i>channel</i>	Set the baud rate for a particular channel. 3. SPHERES Port 4. Serial Device 1 5. Serial Device 2
		unsigned int	<i>baud</i>	The supported baud rates for all devices are (in bps): 115,200 (Serial Device Default) 57,600, 38,400, 19,200, 9,600, 2,400 SPHERES Port additionally supports 250 kbps as default.
void	expv2_uart_baud_get	unsigned char	<i>channel</i>	Request the baud rate for a particular channel. 0 = SPHERES Port 1 = Serial Device 1 2 = Serial Device 2 Expansion Port responds with 5 byte packet: Byte 1: Channel (unsigned char) Byte 2-5: Baud rate (unsigned int) <i>The response invokes the user-defined "EXPv2_CFG_CBK" configuration callback function.</i>
unsigned int	expv2_uart_baud_read	unsigned char	<i>channel</i>	Return the local value for the baud rate of a particular UART channel. 0 = SPHERES Port 1 = Serial Device 1 2 = Serial Device 2
void	expv2_uart_cfg_set	unsigned char	<i>channel</i>	Configure a Serial Connection Channel 0 = SPHERES Port 1 = Serial Device 1 2 = Serial Device 2
		unsigned char	<i>ctrl</i>	<i>Control Mode</i> : 3 bits Bit 1: stop 0 = 1 stop bit 1 = 2 stop bits

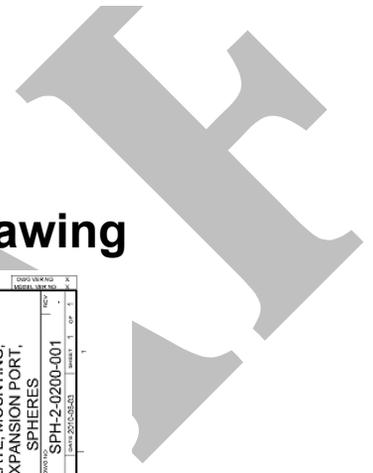
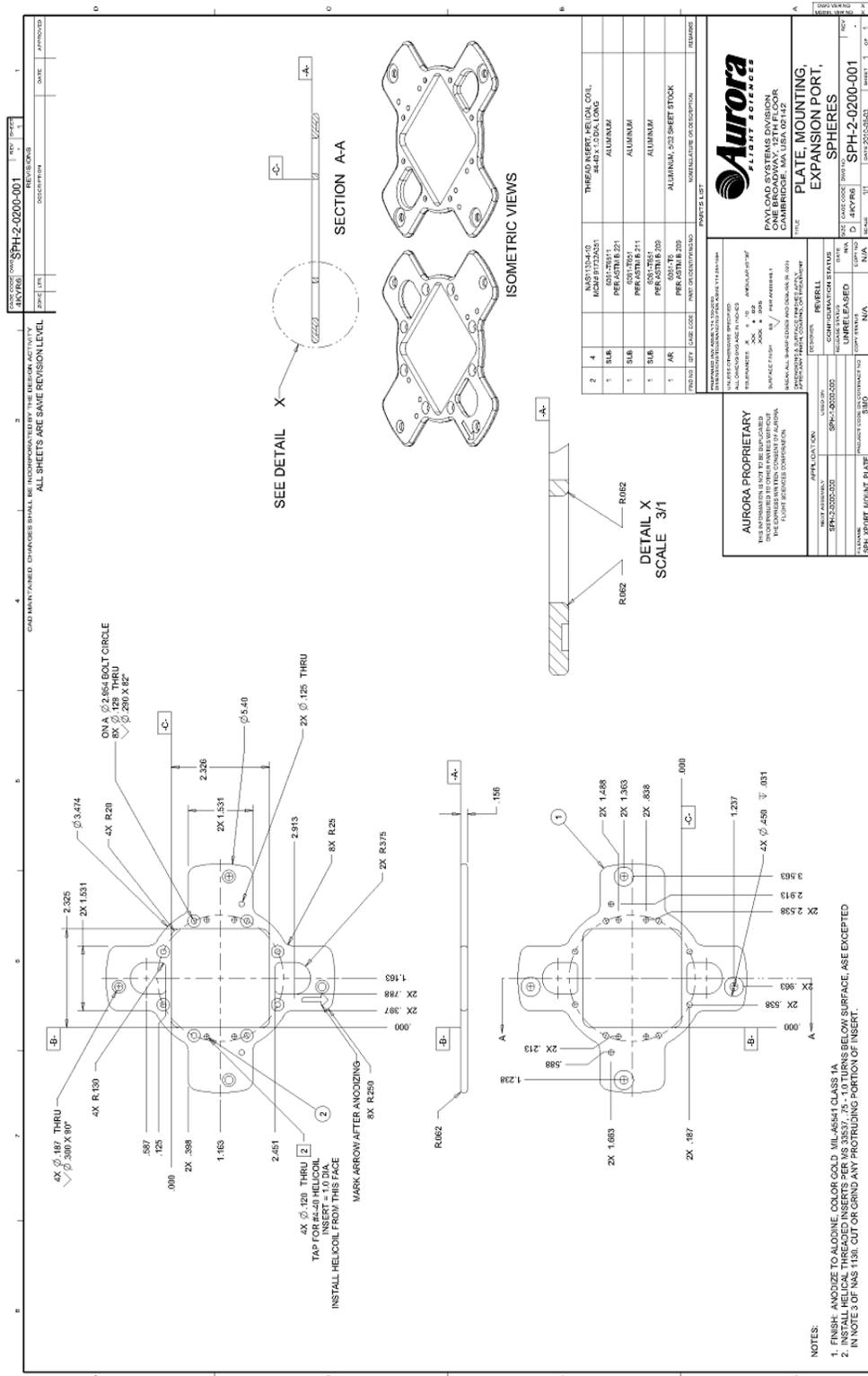
Type	Function Name	Argument Types	Argument Names	Description
		unsigned char	<i>handshaking</i>	Bits 2-3: parity 00 = 8 bit, no parity 01 = 8 bit even data 10 = 8 bit odd data 11 = 9 bit no parity <i>Handshaking</i> Not supported <i>Baud Rate</i> Desired baud rate <i>Enable</i> : 3 bits Bit 1 : UART bus Bit 2 : UART RX Bit 3 : UART TX 0 = Disabled 1 = Enabled
		unsigned int	<i>baud</i>	
		unsigned char	<i>enable</i>	
void	expv2_uart_cbk_register	EXPv2_UART_CBK	<i>callbackFunction</i>	Register a user-defined callback function with the expansion port. When a complete data packet is received the callback function is executed and given the packet, the packet source, and the packet length. (See Type Definitions for callback function definition)

5.6 GPIO Functions

Type	Function Name	Argument Types	Argument Names	Description
void	expv2_gpio_set	unsigned char	<i>pin</i>	Set a GPIO pin value to be on (1) or off (0)
		unsigned char	<i>value</i>	
void	expv2_gpio_get	unsigned char	<i>pin</i>	Get the value of a GPIO pin. <i>Pin and value are given to the user defined "EXPv2_GPIO_CBK" callback function.</i>
void	expv2_gpio_cfg_set	unsigned short	<i>cfg</i>	Set the GPIO TRIS register bits. Input is a 16 bit string with: 0 = output bit 1 = input bit <i>ex: config = 0100001000000011</i> <i>(in hex: 0x4203)</i> <i>for pin 0,1,9,14,15 as input, and the rest as output</i>
void	expv2_gpio_ODC_set	unsigned short	<i>ODC</i>	Enable or disable the Open Drain Collector mode for a GPIO pin. (ODC means that an output pin will only pull down, not pull up) 0 = disabled 1 = enabled

Type	Function Name	Argument Types	Argument Names	Description
void	expv2_gpio_cfg_get	void	void	<p>Get the GPIO configuration.</p> <p><i>The current configuration structure and the response packet are given to the user defined "EXPv2_CFG_CBK" callback function.</i></p> <p>Response packet contains 6 bytes: -From 0xFF, Cmd 0x05 -TRIS [uint16] = pin is input or output -ODC [uint16] = OCD setting for each pin -CS [uint16] = pin selected for use as an SPI CS (<i>Not Currently Supported</i>)</p>
void	expv2_gpio_cfg_read	unsigned short*	TRIS	<p>Write the current local values of the TRIS/ODC/CS bits from the configuration structure into the locations defined by the respective pointers.</p>
		unsigned short*	ODC	
		unsigned short*	CS	
void	expv2_gpio_cbk_register	EXPV2_GPIO_CBK	callbackFunction	<p>Register a user-defined callback function with the expansion port. Every time a GPIO bit value is read, the callback function is executed and given the pin number and its value. (See Type Definitions for callback function definition)</p>

6 Appendix A: Mechanical Interface Drawing



7 Appendix B: EXP_v2 Header & Code Example

7.1 EXP_v2 Header

```
/*
 * EXP_v2.h
 *
 * AJB 2011/12/06
 * ASO 2010/12/14
 *
 */

#ifndef __SPHERES_EXP_V2__
#define __SPHERES_EXP_V2__

/* Configuration Structure */
typedef struct {
    unsigned int uart_cfg[3];
        // [0-2]=baud rate
    unsigned short gpio_cfg[3];
        // 1=I/O bits,2=GPIO open drain config,3=SPI CS bits config
    unsigned int spi_cfg[6]; // SPI not supported
        // 0=Chan#,1=Flags,2=baud Rate Divisor,3=Mode,4="In Use" status,5=CS pin# in use
} cfgStruct;

/* CALLBACK FUNCTION TYPE DEFINITIONS */
/*
EXPv2_UART_CBK

Callback type for functions that will receive multiple bytes of
data passed as a buffer (unsigned char*). The user is responsible for copying data
out of the buffer so it can be reused to receive more information.
Arguments:
channel: device that is sourcing the data
buffer pointer
length of data in buffer
*/
typedef void (*EXPv2_UART_CBK)(unsigned char,unsigned char*,unsigned int);

/*
EXPv2_GPIO_CBK

Callback type for functions that will receive a single byte of data
passed by value. This type can be used for returning status bytes
from things like a DIO. Returned data type could be changed to support
a wider size like short or int.
Arguments:
pinId
value
*/
typedef void (*EXPv2_GPIO_CBK)(unsigned char,unsigned char);

/*
EXPv2_CFG_CBK

Callback type for functions that will receive a pointer to the current
configuration struct and a pointer to the recieved message (which includes the command, thus
defining where the information is inside the data packet). */
typedef void (*EXPv2_CFG_CBK)(cfgStruct *cfg, unsigned char *dataPacket);

/*
 * Function Prototypes
 */

//////////* GENERAL FUNCTIONS *//////////
/* Calls expv2_reset
```

```

Then initializes the default byte, uart and config callbacks
*/
void expv2_init();

/* Reset the configuration variables.
This only changes the SPHERES side content.
This does not change the actual configuration on the expansion board.
*/
void expv2_reset();

/* void expv2_cfg_cbk_register
Register a callback function for when Configuration data is recieved. This occurs for the
following events:
1.) UART Baud Rate Requests
2.) GPIO Configuration Requests
The function must take an configuration structure (cfgStruct) pointer as input. */
void expv2_cfg_cbk_register(EXPv2_CFG_CBK callbackFunction);

/* Returns the configurations structure with its current values, these may be out of date. */
cfgStruct expv2_cfg_get();

//////////* GPIO FUNCTIONS *//////////
/* set a GPIO pin on or off */
void expv2_gpio_set(unsigned char pin, unsigned char value);

/* Request the value of a GPIO pin */
void expv2_gpio_get(unsigned char pin);

/* set GPIO tris (I/O) registers
- cfg [uint16] = TRIS register bits for GPIOs 0-15
0 = output
1 = input
*/
void expv2_gpio_cfg_set(unsigned short cfg);

/* set Open Drain Collector mode for GPIO pins
- ODC [uint16] = enable or dissable ODC operation for a GPIO pin
(ODC means that an output pin will only pull down, not pull up)
0 = disabled
1 = enabled
*/
void expv2_gpio_ODC_set(unsigned short ODC);

/* Request the GPIO configuration */
void expv2_gpio_cfg_get();

/* read configuration of the GPIO
- Send nothing
RETURNS:
- From 0xFF, Cmd 0x5
- TRIS(I/O) [uint16] = pin is in or out
- ODC [uint16] = OCD setting for each pin
- CS [uint16] = pin selected for use as an SPI CS
*/
void expv2_gpio_cfg_read(unsigned short *TRIS, unsigned short *ODC, unsigned short *CS);

/* Allows the user to register their own callback function for when GPIO pins are read through
the expansion port. */
void expv2_gpio_cbk_register(EXPv2_GPIO_CBK callbackFunction);

//////////* UART FUNCTIONS *//////////
/* Send data out through the UART */
void expv2_uart_send(unsigned char channel, unsigned char len, unsigned char *data);

/* Set the baud rate of the UART
Channel:
0 - SPHERES Port
1 - Serial Device 1

```

```

    2 - Serial Device 2 */
void expv2_uart_baud_set(unsigned char channel, unsigned int baud);

/* Request the baud rate of the UART
Channel:
0 - SPHERES Port
1 - Serial Device 1
2 - Serial Device 2 */
void expv2_uart_baud_get(unsigned char channel);

/* unsigned int expv2_uart_baud_read
Return the baud rate of a particular UART channel
0 - SPHERES Port
1 - Serial Device 1
2 - Serial Device 2 */
unsigned int expv2_uart_baud_read(unsigned char channel);

/* void expv2_uart_cfg_set
configure expansion port
- channel
- control mode [uint8] = 3 bits: stop & parity
    [0] = stop (0=1 stop bit, 1=2 stop bits)
    [1-2] = parity bits (00=8 bit, no parity; 01= 8 bit data even; 10=8 bit odd; 11= 9 bit
no parity)
- handshaking [uint8] = 1 bit: 0=no HW flow control; 1 = yes HW flow control
- baud rate [uint32] = actual integer value of desired baud rate
- enable [uint8] = enable/disable transmit/receive
    [0] = UART bus enabled
    [1] = UART RX enabled
    [2] = UART TX enabled */
void expv2_uart_cfg_set(unsigned char channel, unsigned char ctrl, unsigned char handshaking,
unsigned int baud, unsigned char enable);

/* Register a callback function for a particular UART channel. This function will execute
everytime a comm packet is recieved by the SPHERES from the expansion port.
Channel: 0 - SPHERES Port - DON'T USE
1 - Serial Device 1
2 - Serial Device 2 */
void expv2_uart_cbk_register(unsigned char channel, EXPv2_UART_CBK callbackFunction);

///////////* RX FUNCTIONS *///////////
/* Process data from the Expansion Port PIC
Implements callbacks for GPIO get and UART Rx */
void expv2_process_rx_packet(default_comm_packet packet);

#endif

```

7.2 Example GSP.c source code – UART Communication

In gsp.c...

```

#include "exp_v2.h"

unsigned char localBuffer[255];

void userSerialReadFunction(unsigned char channel, unsigned char* data, unsigned int len)
{
    // User Code in here is executed when serial data is recieved...
    /* Copy the provided data into the users local buffer */
    memcpy(localBuffer, data, len);
}

gspInitTest( ... ){
    /* Register the user callback function... */
    expv2_uart_cbk_register(EXPv2_CH1_HWID, &userSerialReadFunction);

    /* Set baud rate for serial device 1 to 57.6kbps */
    expv2_uart_baud_set(EXPv2_CH1_HWID, 57600);
}

```

```
}  
    ...  
gspControl( ... ){  
    ...  
    /* Send Serial Data to device 1 */  
    unsigned char* data[10]={0,1,2,3,4,5,6,7,8,9};  
    expv2_uart_send(EXPv2_CH1_HWID, 10, data);  
    ...  
}
```

DRAFT