A123 Systems Inc.

Energy Solutions Group

BMS & Sub-module Communication Protocol

Version 1.1

February 18, 2008

Revision History

VERSION NUMBER	AUTHOR	DATE	COMMENTS
Draft	Greg Tremelling	???	Created
Draft	Paul Blanc	February 12, 2008	Updated
1.0	Xun (Simon) Li	February 12, 2008	Formatted
1.1	Brian Cascarino	February 18, 2008	Added AUTOADDR_DONE and SUSI messages

TABLE OF CONTENTS

1. INTRODUCTION

The purpose of this document is to describe the communications method and protocols between the Battery Management System (BMS) and an array of attached Battery Modules (BM) in the A123/BAE Energy Storage System (ESS).

2. COMMUNICATIONS PHYSICAL LAYER

The communications physical layer consists of a single BMS connected to multiple BMs. The communication is via a multi-drop +5V level TTL serial UART bus. The output from the BMS is connected in common to the input pins of all attached BMs. All of the outputs from the BMs are wired-or connected together to the input of the BMS. All BM outputs have series diodes and can only pull down. (Note: To communicate without a BMS, a pull-up resistor is required on the outputs from the BMs.)

At the writing this document, latest release (V1_1C) use baud rate of 115200. In the next release, the serial baud rate will be 230400. 8 data bits, 1 stop bit, no parity are used.

See the Battery Module schematic for details and connections.

3. COMMUNICATIONS PROTOCOL

The communications between the BMS (master) and Battery Module (BM) (slaves) are of a master/slave relationship. The slave will not generate communications unless requested by the master.

Each Battery Module (BM) in a system obtains a unique address upon booting via an auto-addressing scheme. A BM can be addressed individually by address. Additionally, some commands from the BMS can be sent to a *global* address which will be received by all BMs present in a system.

3.1 General Message Formats

3.1.1 Command Packets

All command packets from the BMS to the BMs are 8 bytes long.

Byte 0, transmitted first always contains a fixed header value of 0x58.

Byte 1 always contains the address of the intended module or the Global Address value of 0xFF.

Byte 2 always contains an Op Code which identifies each command. See Op Code Summary table.

Bytes 3 through 6 are used to pass command-specific parameters. Byte 3 is typically used for an 8-bit parameter, while bytes 4 and 5 are typically used to pass a 16-bit parameter.

Byte 7 contains an 8-bit CRC value calculated from the previous 7 bytes and is used for transmission error checking. See Appendix A for a description of the CRC calculation.

BMS Command Format:

0	1	2	3	4	5	6	7
HEAD	ADDR	Op Code	Param1	Param2		padding	CRC

Command Op Code Summary:

Command	Value	
	Dec	Hex
TRIGGER	50	0x32
SET ADDRESS	60	0x3C
AUTOADDR_DONE	65	0x41
GLOBAL SNAPSHOT	70	0x46
SEND SUMMARY	80	0x50
SEND ALL VOLATAGES 1	160	0xA0
SEND ALL VOLATAGES 2	161	0xA1
SEND ALL VOLATAGES 3	162	0xA2
BALANCE TARGET	170	0xAA
SUSI	251	0xFB

3.1.2 BM Response Packets

All response packets from a BM to the BMS are 14 bytes long.

Byte 0, transmitted first always contains a fixed header value of 0x58.

Byte 1 always contains the address of the responding Module. Global commands are not responded to due to the wired-or nature of the communications bus.

Bytes 2 through 12 contain response-specific data. See the descriptions of the individual commands for details.

Byte 13 always contains an 8-bit CRC value calculated from the previous 13 bytes and is used for reception error checking in the BMS. See Appendix A for a description of the CRC calculation.

BM Response Format:

0	1	2	3	4	5	6	7
HEAD	ADDR						

8	9	10	11	12	13
					CRC

3.2 Message Definitions

3.2.1 Message –Trigger

BMS Command:

0	1	2	3	4	5	6	7
HEAD	ADDR	50	0	0	0	0	CRC

BM Response: Trigger Acknowledge

NOTE*: This response is sent only if TRIGGER_IN is not active.*

0	1	2	3	4	5	6	7
HEAD	ADDR	0x7F	0	0	0	0	0

8	9	10	11	12	13
0	0	0	0	0	CRC

The modules are assigned addresses via an auto-addressing scheme. The modules are connected in daisy chain fashion via digital TRIGGER_IN and TRIGGER_OUT lines.

This command effectively causes the addressed Battery Module to propagate its TRIGGER_IN input to its TRIGGER_OUT pin.

If TRIGGER_IN is active, the BM activates the trigger to the next BM in the chain by asserting the TRIGGER_OUT digital output pin. No Acknowledge is generated.

If TRIGGER_IN is not active, the BM de-activates the trigger to the next BM in the chain by de-asserting the TRIGGER_OUT digital output pin. Additionally, the BM will send an Acknowledge packet so that the BMS knows that this address was properly obtained. The Acknowledge packet contains the current address and the fixed Acknowledge code 0x7F.

3.2.2 Message – Global Set Address

BMS Command:

0	1	2	3	4	5	6	7
HEAD	0xFF	60	Addr	Start Up Code		Start Up Mode	CRC

BM Response: Acknowledge Address

0	1	2	3	4	5	6	7
HEAD	ADDR	0x7F	0	0	0	0	0

8	9	10	11	12	13
0	0	0	0	0	CRC

This global broadcast command causes the one Battery Module with an active TRIGGER_IN input to capture the address for itself. An Acknowledge packet is sent back to the BMS containing the captured address and the fixed Acknowledge code 0x7F. The Start Up Mode indicates what mode the module should start up in (Boot, Prime, or Reset Only).

```
//------------------------------------------------------------------------------
// Enumeration of module startup modes
//------------------------------------------------------------------------------
typedef enum
{
       MODULE_MODE_BOOT=1,
       MODULE_MODE_PRIME=2,
       MODULE_MODE_RESETONLY=3
} MODULE_MODE_T;
```

3.2.3 Message – Auto Addressing Done

BMS Command:

0	1	2	3	4	5	6	7
HEAD	0xFF	0x41	0	0	0	0	CRC

BM Response:

NO RESPONSE

This global broadcast command causes all Battery Modules to exit auto-addressing mode after they have received their address.

3.2.4 Message – Global Snapshot

BMS Command:

0	1	2	3	4	5	6	7
HEAD	0xFF	70	0	Ess Current*		0	CRC

BM Response:

NO RESPONSE

This command causes all Battery Modules to store an immediate snapshot of all 12 cell voltages in a memory buffer. These cell voltages can subsequently be read by the BMS by issuing three SendAllVoltages commands. Note that 0xFF is the Global Address value.

ESS Current is the total current through the entire ESS system and is passed to the BMs within this command packet. It is intended for future use and is presently not implemented.

3.2.5 Message – Send All Voltages 1,2,3

BMS Command:

0	1	2	3	4	5	6	7
HEAD	ADDR	16x	0x00	0x00	0x00	0x00	CRC

BM Response:

0	1	2	3	4	5	6	7
HEAD	ADDR	Cell A Voltage		Cell B Voltage		Cell C Voltage	

8	9	10	11	12	13
Cell D Voltage		0x00	0x00	0x00	CRC

The Op Code can be one of three values, indicating which group of 4 cells will be returned.

OpCode Value		
160	Cell A	Cell 1
	Cell B	Cell 2
	Cell C	Cell 3
	Cell D	Cell 4
161	Cell A	Cell 5
	Cell B	Cell 6
	Cell C	Cell 7
	Cell D	Cell 8
162	Cell A	Cell 9
	Cell B	Cell 10
	Cell C	Cell 11
	Cell D	Cell 12

Once a Global Snapshot command has been executed, the contents of the snapshot memory buffer can be read by issuing 3 SendAllVoltages commands.

Since only 4 cell voltages can be returned in any one Response packet, three commands are required to retrieve all 12 cell voltages. The OpCode determine which 4 cells are returned by each SendAllVoltages command.

All voltages are expressed in units of mV.

3.2.6 Message – Balance Target

<u>BMS Command</u>:

0	1	2	3	4	5	6	7
HEAD	ADDR	170	0	Balance Target		0	CRC

<u>BM Response</u>:

NOTE: A response is only returned if a specific BM address was used. If the Global address (0xFF) was used, there is no module response.

0	1	2	3	4	5	6	7
HEAD	ADDR	Balance Target		0	0	0	0

8	9	10	11	12	13
0	0	0	0	0	CRC

Cell voltage balancing is a continuous operation in the modules. The target voltage sent from the BMS is immediately used as the new balancing voltage. To disable the balancing function a target voltage of 4096 is sent to the module(s).

If BalanceTarget is sent to a specifically addressed Module, a response packet will be retuned including the new balance target voltage. If BalanceTarget is sent as a global broadcast, no response will be returned.

Balance Target voltage is expressed in units of mV.

3.2.7 Message – Send Summary

<u>BMS Command</u>:

0	1	2	3	4	5	6	7
HEAD	ADDR	80	0	Ess Current*		0	CRC

* ESS Current is the total current through the entire ESS system and is passed to the BMs within this command packet. It is intended for future use and is presently not implemented.

<u>BM Response</u>:

0	1	2	3	4	5	6	7
HEAD	ADDR	Min Cell Voltage		Max Cell Voltage		Average Cell Voltage	

8	9	10	11	12	13
Min/Max Locations	Temperatures			Status	CRC

Format of Response packet:

```
unsigned char header;             // Response header byte
unsigned char address;            // Battery Module address
unsigned int  cellMinV;           // Voltage of the minimum cell in mV
unsigned int  cellMaxV;           // Voltage of the maximum cell in mV
unsigned int  cellAvgV;           // Average voltage of all 12 cells in mV
unsigned char minCell : 4;        // Location of minimum cell (1..12)
unsigned char maxCell : 4;        // Location of maximum cell (1..12)
unsigned long Temperature1 : 12;  // Temperature at sensor #1 in 0.1C units
unsigned long Temperature2 : 12;   // Temperature at sensor #2 in 0.1C units
unsigned char status;             // Battery Module status byte
unsigned char crc;                // 8-bit CRC of previous bytes in this packet
```

This command will cause a return of summary information to the BMS including, minimum cell voltage and location, maximum cell voltage and location, average cell voltage, temperature of both thermal sensors, and module status.

The C format of this structure is illustrated above. Note the use of bit-fields to pack data into fewer bytes. For example, byte 8 contains the location of the minimum cell in the lower nibble and the location of the maximum cell in the upper nibble. Bytes 9 through 11 contain the temperature of sensor 1 in the lower 12-bits and the temperature of sensor 2 in the upper 12-bits.

The bit assignments of the Status byte are as follows:

7	6	5	4	3	2	1	0
					Limits Error	Sum Error	OVP Error

Bit 0	OVP Error	Hardware over voltage protection asserted
Bit 1	Sum Error	Mismatch between sum of cells and total voltage
Bit 2	Limits Error	One or more cell voltages are beyond limits

3.2.8 Message – SUSI

BMS Command:

0	1	2	3	4	5	6	7
HEAD	ADDR (SUSI Target ID)	0xFB	SUSI Host ID	SUSI Command Code	SUSI Cmd Address 2 (MSB24)	SUSI Cmd Address 3	SUSI Cmd Adderss 4 (LSB)

8	9	10	11	13
SUSI Cmd Data 1 (MSB)	SUSI Cmd Data 2	SUSI Cmd Data 3	SUSI Cmd Data 4 (LSB)	CRC

BM Response:

0	1	2	3	4	5	6	7
HEAD	ADDR (SUSI Target ID)	0xFB	SUSI Host ID	SUSI Acknowledge Code	SUSI Response Data 1 (MSB32)	SUSI Response Data 2	SUSI Response Data 3

8	9	10	11	13
SUSI Response Data 4 (LSB)	SUSI Echo Cmd Addr 2	SUSI Echo Cmd Addr 3	SUSI Echo Cmd Addr 4	CRC

This conveys a SUSI command from the BMS to one or all Battery Modules. In the case of a broadcast packet, no ACK is returned. Bytes 3 through 11 are C24 SUSI Packet data (see Table 3 of SUSI in CAN V1.0 – BAE Systems Document Number 362A6788). SUSI command packets may only be sent from the BMS to the Battery Modules, which may return SUSI ACK packets to the BMS.

Appendix A

CRC Calculation

The CRC used in both Command and Response packets is an 8-bit CRC which is commonly used by Dallas Semiconductor 1-Wire products. The CRC is appended to the end of each message packet and is calculated from the previous bytes in the packet. Upon reception, the CRC of all but the last byte in the packet can be calculated and compared to the CRC in the last byte. A useful property of the CRCs is that the CRC calculated from all bytes in a packet, including the received CRC byte, will always be equal to zero if no errors were detected.

The following C-code excerpts provide details of the CRC calculation.

```
/******************************************************************************
CRC-8 NOTES:
        The CRC-8 is an implementation of the Dallas 1-Wire 8-bit CRC
        Dallas CRC-8 polynomial is x^8 + x^5 + x^4 + 1
        Represented as 0x31
        Initial value used by Dallas CRC-8 is 0x00
        Data received in True (non-inverted) form
        This implementation uses a 256-byte look-up table for high speed

        The same function can be used to check an existing message containing a CRC.
        To add a CRC to an outgoing message:,
                Calculate the CRC using only the bytes of the message,
                then append the CRC byte to the end.
        To check an incoming message:
                Calculate a CRC using all the message bytes, including the received CRC.
                An intact incoming message will result in a 0x00 CRC calculation.
                A non-zero CRC indicates a received data error.

        See Maxim/Dallas AP Note 27:
                "Understanding and Using Cyclic Redundancy Checks with
                Dallas Semiconductor iButton Products"

        Author: Paul LeBlanc, LeBlanc Laboratories, Inc.
******************************************************************************/

//-----------------------------------------------------------------------------
// CRC-8 LUT data from the Dallas App Note. For use in CRC-8 calculations.
//-----------------------------------------------------------------------------
static const unsigned char crc8LUT[256] =
        {
                0x00, 0x5e, 0xbc, 0xe2, 0x61, 0x3f, 0xdd, 0x83,
                0xc2, 0x9c, 0x7e, 0x20, 0xa3, 0xfd, 0x1f, 0x41,
                0x9d, 0xc3, 0x21, 0x7f, 0xfc, 0xa2, 0x40, 0x1e,
                0x5f, 0x01, 0xe3, 0xbd, 0x3e, 0x60, 0x82, 0xdc,
                0x23, 0x7d, 0x9f, 0xc1, 0x42, 0x1c, 0xfe, 0xa0,
                0xe1, 0xbf, 0x5d, 0x03, 0x80, 0xde, 0x3c, 0x62,
                0xbe, 0xe0, 0x02, 0x5c, 0xdf, 0x81, 0x63, 0x3d,
                0x7c, 0x22, 0xc0, 0x9e, 0x1d, 0x43, 0xa1, 0xff,
                0x46, 0x18, 0xfa, 0xa4, 0x27, 0x79, 0x9b, 0xc5,
                0x84, 0xda, 0x38, 0x66, 0xe5, 0xbb, 0x59, 0x07,
                0xdb, 0x85, 0x67, 0x39, 0xba, 0xe4, 0x06, 0x58,
                0x19, 0x47, 0xa5, 0xfb, 0x78, 0x26, 0xc4, 0x9a,
                0x65, 0x3b, 0xd9, 0x87, 0x04, 0x5a, 0xb8, 0xe6,
                0xa7, 0xf9, 0x1b, 0x45, 0xc6, 0x98, 0x7a, 0x24,
                0xf8, 0xa6, 0x44, 0x1a, 0x99, 0xc7, 0x25, 0x7b,
                0x3a, 0x64, 0x86, 0xd8, 0x5b, 0x05, 0xe7, 0xb9,
                0x8c, 0xd2, 0x30, 0x6e, 0xed, 0xb3, 0x51, 0x0f,
                0x4e, 0x10, 0xf2, 0xac, 0x2f, 0x71, 0x93, 0xcd,
                0x11, 0x4f, 0xad, 0xf3, 0x70, 0x2e, 0xcc, 0x92,
                0xd3, 0x8d, 0x6f, 0x31, 0xb2, 0xec, 0x0e, 0x50,
                0xaf, 0xf1, 0x13, 0x4d, 0xce, 0x90, 0x72, 0x2c,
                0x6d, 0x33, 0xd1, 0x8f, 0x0c, 0x52, 0xb0, 0xee,
                0x32, 0x6c, 0x8e, 0xd0, 0x53, 0x0d, 0xef, 0xb1,
                0xf0, 0xae, 0x4c, 0x12, 0x91, 0xcf, 0x2d, 0x73,
                0xca, 0x94, 0x76, 0x28, 0xab, 0xf5, 0x17, 0x49,
                0x08, 0x56, 0xb4, 0xea, 0x69, 0x37, 0xd5, 0x8b,
                0x57, 0x09, 0xeb, 0xb5, 0x36, 0x68, 0x8a, 0xd4,
                0x95, 0xcb, 0x29, 0x77, 0xf4, 0xaa, 0x48, 0x16,
                0xe9, 0xb7, 0x55, 0x0b, 0x88, 0xd6, 0x34, 0x6a,
                0x2b, 0x75, 0x97, 0xc9, 0x4a, 0x14, 0xf6, 0xa8,
                0x74, 0x2a, 0xc8, 0x96, 0x15, 0x4b, 0xa9, 0xf7,
                0xb6, 0xe8, 0x0a, 0x54, 0xd7, 0x89, 0x6b, 0x35,
        };

//================================================================================
// Function:     crcCalcCRC8
//
// Purpose:      Calculates an 8-bit CRC of an array of specified length
// Returns:      CRC-8 of 'len' bytes of array 'msg'
// Arguments:    *msg     - pointer to an array containing the data to be used
//               len      - number of data bytes to use in CRC calculation
// Globals:      crc8LUT  - look up table of basic CRC values
// Remarks:      Implements the CRC used by Dallas 1-Wire devices
//               Ensure that the array contains at least 'len' bytes
// Example:      checkCRC = crcCalcCRC8(myMessage, sizeof(myMessage));
//================================================================================
unsigned char crcCalcCRC8(unsigned char * msg, unsigned int len)
{
  unsigned char crc;
  unsigned int i;

  crc = 0x00;            // initial remainder
  for(i=0; i<len; i++)
  {
        crc = crc8LUT[(msg[i] ^ crc) & 0xFF];
  }
  return crc;
}// crcCalcCRC8()
```

//____