

# SANDMAN Power Management User Manual

---



# Table of Contents

## Contents

1.	Introduction .....	5
1.1	Limitations .....	5
1.1.1	User Responsibility .....	5
1.2	Theory of Operation.....	5
2.	Specifications and Characteristics .....	6
2.1	Performance Specifications .....	6
2.2	Electrical Characteristics .....	6
2.3	Absolute Maximum Ratings .....	6
2.4	Mechanical and Pin Assignments .....	7
2.4.1	Dimensions .....	7
2.4.2	Recommended Connectors .....	7
2.4.3	Pin Assignments .....	8
3.	Hardware Integration .....	10
3.1	Power .....	10
3.1.1	Input Power .....	10
3.1.2	Output Power.....	10
3.2	Communication Interface.....	10
3.2.1	Pin 1 - TX.....	10
3.2.2	Pin 2 - RX .....	10
3.2.3	Pin 3 and Pin 9 - Ground .....	10
3.2.4	Pin 4 - ISP .....	10
3.2.5	Pin 5 - DIO1 .....	10
3.2.6	Pin 6 – DIO2 .....	10
3.2.7	Pin 7 – DIO3 .....	10
3.2.8	Pin 8 – DIO4 .....	10
3.3	Communication.....	11
3.3.1	UART.....	11
3.3.2	I2C Port .....	11
3.4	Power Status LED .....	11
4.	Software Interface .....	12
4.1	Basic Message Structure.....	12
4.1.1	Message format .....	12
4.1.2	Number Formats.....	12
4.1.3	LRC Checksum Code .....	12
4.1.4	ACK / NACK Response .....	13
5.	Input Message Details .....	15
5.1	Input Message Summary .....	15
5.1.1	PING (ID 0x00) .....	16
5.1.2	SET TIME (ID 0x01).....	17
5.1.3	GET TIME (ID 0x02) .....	17
5.1.4	SET ALARM (ID 0x03).....	18
5.1.5	GET ALARM (ID 0x04) .....	18
5.1.6	STORE DATA (ID 0x05) .....	19
5.1.7	GET DATA (ID 0x06) .....	19
5.1.8	GOTOSLEEP (ID 0x07) .....	20
5.1.9	EASY SLEEP (ID 0x08).....	20
6.	Output Message Details .....	21
6.1	Output Messages .....	21
6.1.1	PING (ID 0x00) .....	22

6.1.2	RESERVED (ID 0x01)	22
6.1.3	UNIT TIME (ID 0x02)	22
6.1.4	RESERVED (ID 0x03)	22
6.1.5	UNIT ALARM (ID 0x04)	23
6.1.6	RESERVED (ID 0x05)	23
6.1.7	GET DATA (ID 0x06)	23
6.1.8	RESERVED (ID 0x07)	24
6.1.9	RESERVED (ID 0x08)	24
7.	Software Reprogramming	25
8.	Software Release Notes	25
9.	Appendix A: TBD	26

## Release Notes

<b>Title</b>	<b>Sandman</b>		
<b>Subtitle</b>	Sandman User Manual		
<b>Type</b>	Manual		
<b>Document number</b>	UM3000		
<b>Revision Index</b>	<b>Date</b>	<b>Name</b>	<b>Status / Comments</b>
<b>Initial Release</b>	11/2012	MR	Initial release

## **IMPORTANT DISCLAIMERS**

This document and the use of any information contained therein, is subject to the acceptance of the Ryan Mechatronics terms and conditions. They can be downloaded from [www.ryanmechatronics.com](http://www.ryanmechatronics.com).

Ryan Mechatronics LLC makes no warranties based on the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice.

Ryan Mechatronics LLC assumes no liability for damages or otherwise due to use of the information in this document or application of any device described in this document.

Ryan Mechatronics LLC stresses end user compliance with all applicable laws and regulations when using devices of this nature. Use by an end user in violation of any applicable laws is automatic basis for termination of warranty, technical support and future sales.

Ryan Mechatronics LLC reserves all rights to this document and the information contained herein. Reproduction, use or disclosure to third parties without express permission is strictly prohibited.

Copyright © 2009 - 2013, Ryan Mechatronics LLC

# 1. Introduction

The Sandman is a power management and data storage electronic board that allows you to effectively turn electronic hardware off for a specified amount of time, then have it turn back on. During the power down phase, current draw to the Sandman board is approximately 100 micro-amps. Sandman can control power to systems from 3.3V up to 20V while allowing up to 1.8 amps of current.

Up to 1k of data can be stored in nonvolatile memory on Sandman before turning power off, allowing your system to save system status and other information for use after power has returned.

The communication interface is via a 3.3V level UART with simple command structure.

The Sandman is intended for use in remote sensing applications that require strict power management to operate effectively.

## 1.1 Limitations

The unit, like any IMU / AHRS, can be pushed beyond the limits of its ability to sense any of the measurements it needs to operate correctly. The following list includes results that are known to occur if operation exceeds the limits listed later in this document.

### 1.1.1 User Responsibility

Accidental programming of an incorrect wake-up can result in a system that goes to sleep...and stays asleep. [Please be careful and test your system thoroughly before deployment.](#)

## 1.2 Theory of Operation

The Sandman integrates low resistance power circuitry, a low power real time clock unit and a CPU capable of deep power down modes in a single package that allows users to:

- Control the voltage supply to a separate system
- Store data to nonvolatile flash
- Turn off power to another system or the host system with automatic wake up

## 2. Specifications and Characteristics

Presented in this section are the sensor and system specifications for the Sandman. All parameters specified are @ VDD = 5.0 V and Ta = 25°C unless otherwise noted.

### 2.1 Performance Specifications

Characteristics	Conditions	Min	Typical	Max	Units
<b>Memory</b>					
Size	NVM storage available for user	1024	1024	4096	bytes
<b>Communication</b>					
UART characteristics	115,200 baud, 8-N-1	-	-	-	-

- Specifications are subject to change at any time without notice

### 2.2 Electrical Characteristics

Characteristics	Conditions	Min	Typical	Max	Units
<b>Power</b>					
Supply Voltage Range	V <sub>dd</sub> Referenced to GND	3.3	5	20	V
Current, maximum pass thru to load	Maximum to load on output	0	-	1.8	A
Support circuitry, normal operation	System operating and awake		4.5		mA
Support circuitry in deep sleep	System in deep sleep, load output turned off		105		uA

- Specifications are subject to change at any time without notice

### 2.3 Absolute Maximum Ratings

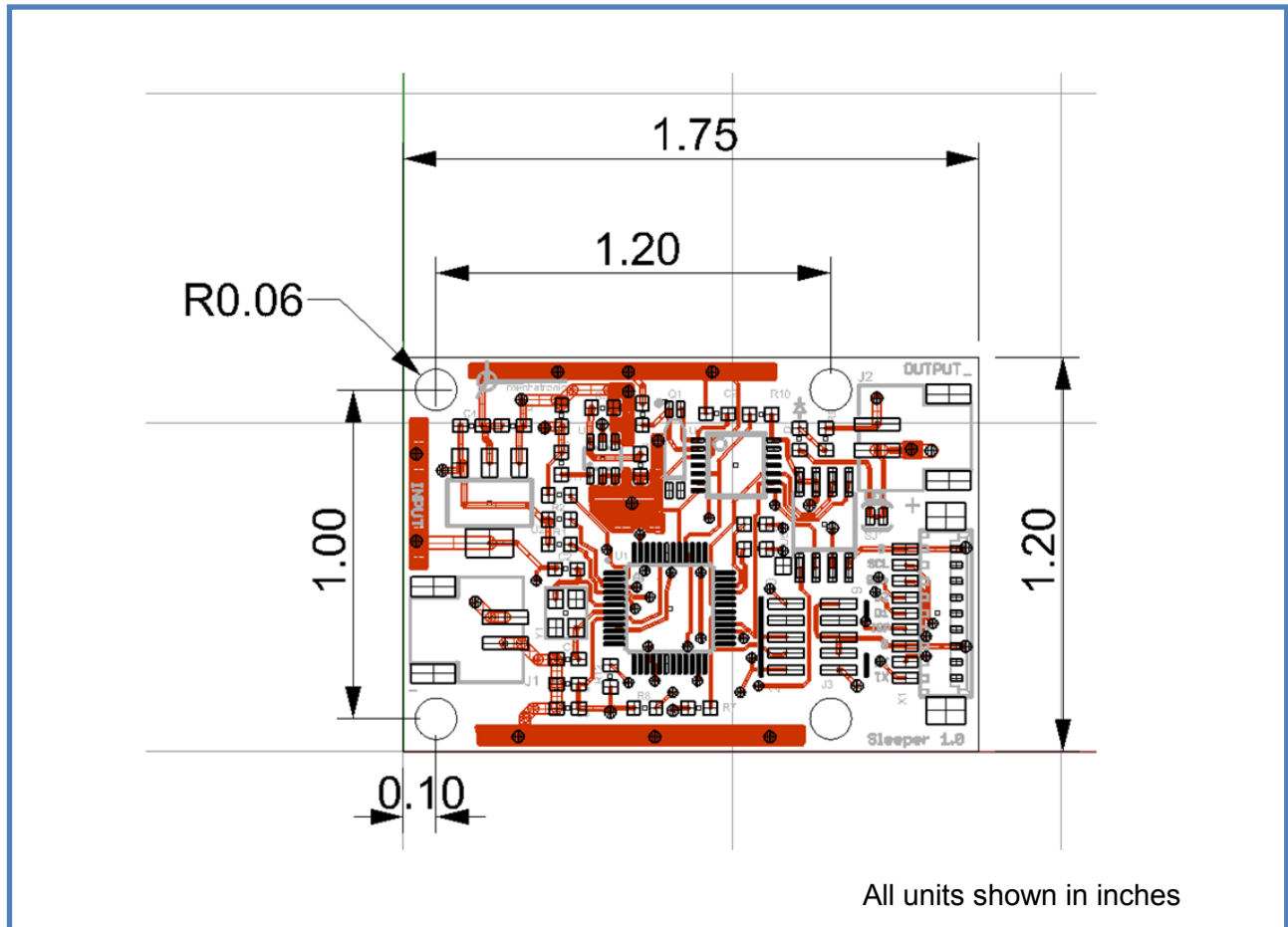
Parameter	Rating
V <sub>dd</sub>	-0.3V to +20V
Operating Temperature Range	-40°C to +85°C
Storage Temperature Range	-55°C to +125°C

- Specifications are subject to change at any time without notice

Stresses above those listed under the Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at or near these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect device reliability.

## 2.4 Mechanical and Pin Assignments

### 2.4.1 Dimensions



### 2.4.2 Recommended Connectors

#### 2.4.2.1 Power In / Out

The Input and Output power connectors are identical. They are JST connector part S2B-PH-SM4-TB(LF)(SN) (Digikey P/N: 455-1749-1-ND).

There are many mating options for this connector, but an easy method is to purchase pre-crimped / built mating cables. An example of this is Pololu item 1116

<http://www.pololu.com/catalog/product/1116> which is a connector and 14cm wire length. For reference, a picture of this connector is shown here:





### 2.4.2.2 Communications Connector (X1)

The communications connector has a set of connections for discrete I/O and communication. This connector is a 9 pin Molex socket. The mating connector for this is Digi-Key part number WM1727-ND, but this is just the connector housing. Pre-crimped wires can be found at Newark, part number 06-66-0013. For reference, a picture of this connector is shown here:



### 2.4.3 Pin Assignments

Table 1 – Power In/Out Connector Pin Assignments

Pin #	Pin Name	I/O	Pin Connection Required for Typical Operation?	Description
1	Vin	N/A	X	Or Vout if output connector
2	GND	N/A	X	

**Table 2 – Communication (X1) Pin Assignments**

Pin #	Pin Name	I/O	Pin Connection Required for Typical Operation?	Description
1	TX	O		UART com from Sandman 3.3V level
2	RX	I	X	UART com to Sandman 3.3V level
3	GND	N/A	X	Ground
4	ISP	I		Hold low upon power for reprogramming of unit (not standard)
5	DIO1	I		TBD
6	DIO2	I		TBD
7	SDA/DIO3	I		TBD
8	SCL/DIO4	I		TBD
9	GND	N/A		Ground (redundant to pin 3)

## 3. Hardware Integration

Presented in this section are selected hardware interface comments to help ease integration of the unit in the end user system.

### 3.1 Power

#### 3.1.1 Input Power

The module operates off the input power voltage. On board circuitry operates at 3.3V levels. However, up to 20V may be applied at this connector.

The system draws very little power; typically it will require less than 5 mA of supply for operation when not in sleep mode.

#### 3.1.2 Output Power

The module outputs the input power voltage when the unit is programmed to allow this output. When the unit goes to sleep, this output power is turned off.

### 3.2 Communication Interface

The module utilizes the X1 pins to command the unit and receive status. These pins are described here.

#### 3.2.1 Pin 1 - TX

Output data from the module.

#### 3.2.2 Pin 2 - RX

Commands to the module

#### 3.2.3 Pin 3 and Pin 9 - Ground

Ground pins for signal use.

#### 3.2.4 Pin 4 - ISP

In system programming pin. Pull low upon boot to load new code via UART.

#### 3.2.5 Pin 5 - DIO1

TBD

#### 3.2.6 Pin 6 – DIO2

TBD

#### 3.2.7 Pin 7 – DIO3

TBD

#### 3.2.8 Pin 8 – DIO4

TBD

### **3.3 Communication**

Sandman has two means of communication - a standard UART and an I2C port.

Currently, the I2C port is inactive, but may be active as a slave device in future software releases.

#### **3.3.1 UART**

The UART is a 3.3V level interface. The unit does not use hardware handshaking. It is always 8 bits, no parity, and one stop bit (8-N-1). Do NOT interface with a standard RS-232 port, as the voltages on that port will damage the unit. An external adapter that uses 3.3V to convert to RS-232 levels can be powered from the onboard 3.3V regulator.

Standard operation is at 115k baud.

#### **3.3.2 I2C Port**

The I2C bus will be configured as a master device. Currently, it is not implemented, but future software revisions may include this.

### **3.4 Power Status LED**

There is one (1) informational LED on the unit. This LED is for test only, and can be disconnected by cutting the LED trace solder jumper. The LED pulls an extra 0.5 mA at 5V during normal operation, so cutting this trace is good practice for energy saving purposes.

## 4. Software Interface

The hardware com interfaces have been described already. Details on software setup and communication are presented here.

### 4.1 Basic Message Structure

Input and Output messages from the unit are identical. Both include header and checksum and other information to protect data integrity and allow easier decoding by the end user.

#### 4.1.1 Message format

Both input and output messages have a defined structure that consists of the following:

- (2) header bytes (0xAE 0xAE)
- (1) Message ID byte
- (xxx) Payload bytes (varies with message)
- (1) Checksum byte

The checksum byte is a LRC checksum calculated for the entire message, including header bytes, length, id, and data bytes. Details and code for calculating this checksum are provided later in this section.

#### 4.1.2 Number Formats

Much of the data messages use single bytes and unsigned integers, which are typically easy to understand. A number format list is presented here for clarity on number formats however.

All floating point values are transmitted in IEEE754 single precision.

**Table 3 - Number Formats**

Abbreviation	Type	Size (bytes)	Comment	Min/Max
U1	Unsigned char	1		0 ... 255
I1	Signed char	1	2's complement	-128 ... 127
X1	Bitfield	1		n/a
U2	Unsigned short integer	2		0 ... 65535
I2	Signed short integer	2	2's complement	-32768 ... 32767
X2	Bitfield	2		n/a
U4	Unsigned long	4		0...4,294,967,295
L4	Signed long	4	2's complement	-2,147,483,648 ... 2,147,483,647
R4	IEEE 754 Single Precision	4		$-1 \cdot 2^{127} \dots 2^{127}$
CH	ASCII encoded	1		

#### 4.1.3 LRC Checksum Code

The checksum calculated for outgoing messages is an 8 bit Longitudinal Redundancy Check (LRC) code. C code to compute the entire checksum is shown below.

```

unsigned char calculateLRC(const unsigned char *buf, unsigned int n)
{
    unsigned char checksum = 0;
    while(n>0){
        checksum += *buf++;
        n--;
    }
    return ((char) -checksum );
}

```

#### 4.1.4 ACK / NACK Response

A properly formatted message that is accepted by the module will be responded to with the following ACK or NACK:

##### 4.1.4.1 ACK Response

The acknowledge response will have the form:

0x21

This is the hex code for an ASCII “!” character.

##### 4.1.4.2 NACK Response

The non-acknowledge response will have the form:

0x63

This is the hex code for an ASCII “?” character.

##### 4.1.4.3 Useful Code Structures

The following structures are useful for the setting and reading of data from the module.

##### 4.1.4.4 TIME Structure

The following C structure is used for both setting and reading time from the unit. This is 27 bytes.

```

typedef struct
{
    unsigned char hun_sec; //Hundreds of seconds
    unsigned char tenth_sec; //Tenths of seconds
    unsigned char sec; //Seconds
    unsigned char ten_sec; //Tens of seconds
    unsigned char min; //Minutes
    unsigned char ten_min; //Tens of minutes
    unsigned char st; //Oscillator start bit
    unsigned char hour; //Hours
    unsigned char ten_hr; //Tens of hours (use with 24 hr time)
    unsigned char AMPM; //10 hour AM/PM
    unsigned char TIMEFORMAT; //If set, 24 hr time. If cleared, 12 hour time
    unsigned char CALSGN; //Calibration sign
    unsigned char day; //Day

```

```

unsigned char VBATEN; //Vbat enable
unsigned char VBAT; //Vbat switched
unsigned char OSCON; //Oscillator on flag
unsigned char date; //date
unsigned char ten_date; //Tens of date
unsigned char month; //Month
unsigned char ten_month; //Tens of month
unsigned char lp; //leap year
unsigned char year; //year
unsigned char ten_year; //Tens of year
unsigned char CTRL_REG; //control register
unsigned char CALIBRATION; //Should be zero if you ever set this
unsigned char WATCHDOG; //Watchdog register
unsigned char EVENT; //Event detect (unused)
} TimeKeeper_RTC_TypeDef;

```

#### 4.1.4.5 ALARM Structure

The following C structure is used for both setting and reading alarms on the unit. It is 16 bytes long.

```

typedef struct
{
    unsigned char sec; //Seconds
    unsigned char ten_sec; //Tens of seconds
    unsigned char min; //Minutes
    unsigned char ten_min; //Tens of minutes
    unsigned char hour; //Hours
    unsigned char ten_hr; //Tens of hours (use with 24 hr time)
    unsigned char AMPM; //10 hour AM/PM
    unsigned char TIMEFORMAT; //If set, 24 hr time. If cleared, 12 hour time
    unsigned char day; //Day
    unsigned char ALMxIF; //Alarm Interrupt flag bit (must be cleared by software)
    unsigned char ALMxCx; //Alarm match conditions. 000=seconds, 001 = min, 010 = hrs, 111
= sec, min, hr, day, date, and month
    unsigned char ALMxPIN; //Alarm Output Pin config bit. 0 = IRQ pin. 1 = WDO pin.
    unsigned char date; //date
    unsigned char ten_date; //Tens of date
    unsigned char month; //Month
    unsigned char ten_month; //Tens of month
} TimeKeeper_ALARM_TypeDef;

```

## 5. Input Message Details

Shown in this section are specific input message requirements and details

### 5.1 Input Message Summary

The output messages from the unit are shown in this summary table:

**Table 4. Sandman Message Input**

Name	ID	Description
<b>Ping</b>	0x00	Communication check
<b>Set Time</b>	0x01	Sets time on unit. Time is lost if power to the module is removed (i.e. no battery backup)
<b>Get Time</b>	0x02	Retrieves the time from the unit
<b>Set Alarm</b>	0x03	Sets alarm on unit.
<b>Get Alarm</b>	0x04	Retrieves the alarm currently set on the unit
<b>Store Data</b>	0x05	Stores data to NVM on the module in preparation of power down / sleep
<b>Get Data</b>	0x06	Retrieves data from NVM
<b>GoToSleep</b>	0x07	Goes to sleep, powers down. Will not engage if alarm time is prior to current time
<b>EasySleep_10min</b>	0x08	Go to sleep for 10 minutes from current time. Will occur immediately upon receipt.
<b>EasySleep_1hour</b>	0x09	1 hour from current time
<b>EasySleep_3 hour</b>	0x0A	3 hours from current time
<b>EasySleep_12hour</b>	0x0B	12 hours from current time
<b>EasySleep_24hour</b>	0x0C	1 day from current time



### 5.1.1 PING (ID 0x00)

Message		Name			
Description		<b>PING</b>			
ID		<b>0x00</b>			
Length		<b>1</b>			
Type		Input Message			
Comment		<b>Pings the unit. Unit responds with ping output message, BIT message, and current time. Useful for software protocol test.</b>			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x00	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
<b>0</b>	U1	-	N/A	-	Value not used, but required for valid message

This message forces a ping output message (0x00) as a response regardless of the package data.

### 5.1.2 SET TIME (ID 0x01)

Message		Name			
Description		Set Time			
ID		0x01			
Length		31			
Type		Input Message			
Comment		Sets the unit time.			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x01	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	Time Structure	-	Current Time	-	Current time from the unit.

This message sets the time on the unit. This is NOT held in memory if main power is removed to the module (no battery backup). It is maintained and incremented in deep sleep however.

### 5.1.3 GET TIME (ID 0x02)

Message		Name			
Description		Get Time			
ID		0x02			
Length		5			
Type		Input Message			
Comment		Gets the unit time.			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x02	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	U1	-	N/A	-	Requests output message 0x02 from module

This message requests time from the unit via message 0x02.

### 5.1.4 SET ALARM (ID 0x03)

Message		Name			
Description		Set Alarm			
ID		0x03			
Length		21			
Type		Input Message			
Comment		Sets the unit alarm			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x03	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	Alarm Structure	-	Alarm	-	Desired alarm time for wakeup
16	U1	-	Alarm Type	-	Alarm type to wake up on: <ul style="list-style-type: none"> <li>• Seconds match</li> <li>• Minutes match</li> <li>• Hours match</li> <li>• Full date match</li> </ul>

This message sets the alarm on the unit. This is NOT held in memory if main power is removed to the module (no battery backup). It is maintained and incremented in deep sleep however.

The alarm type sets what aspect of the alarm triggers a wake up.

Follow this up with the GoToSleep message to power down.

### 5.1.5 GET ALARM (ID 0x04)

Message		Name			
Description		Get Alarm			
ID		0x04			
Length		5			
Type		Input Message			
Comment		Gets the unit time.			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x02	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	U1	-	N/A	-	Requests output message 0x04 from module

This message requests alarm from the unit via message 0x04.

## 5.1.6 STORE DATA (ID 0x05)

Message		Name			
Description		Store Data			
ID		0x05			
Length		Varies			
Type		Input Message			
Comment		Stores the data for retention during power down			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x05	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	U2	-	Bytes to write	-	Number of bytes to write (maximum 4096)
2,3,4....4097	U1	-	Data to store	-	Data bytes are stored up to length sent

This message stores a stream of data to the unit to be saved during power down. All other data on the unit is erased / overwritten by this message.

## 5.1.7 GET DATA (ID 0x06)

Message		Name			
Description		Get Data			
ID		0x06			
Length		5			
Type		Input Message			
Comment		Requests data retrieval			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x06	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	U2	-	Bytes to read	-	Number of bytes to read via output message 0x06

This message requests stored data from the unit via message 0x06.

### 5.1.8 GOTOSLEEP (ID 0x07)

Message			Name		
Description			Go To Sleep		
ID			0x07		
Length			5		
Type			Input Message		
Comment			Goes to sleep, wakes on alarm		
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x07	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	U2	-	Sleep pause	-	Milliseconds to delay before going to sleep

This message puts the unit to sleep after the number of milliseconds in the payload.

### 5.1.9 EASY SLEEP (ID 0x08)

Message			Name		
Description			Easy Sleep		
ID			0x08		
Length			5		
Type			Input Message		
Comment			Easy method for setting the unit to sleep		
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x08	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	U1	-	Type of Easy Sleep	-	Easy sleep

This message puts the unit to sleep immediately for the specified amount of time.

Type of Easy Sleep:

- 0 = 10 seconds
- 1 = 1 minute
- 2 = 1 hour
- 3 = 3 hour
- 4 = 12 hour
- 5 = 24 hour

## 6. Output Message Details

Shown in this section are the specific message descriptions for output messages from the unit.

### 6.1 Output Messages

The output messages from the unit are shown in this summary table:

**Table 5. Sandman Message Output**

Name	ID	Length	Description
Ping	0	7	"I'm alive" message
Reserved	1	-	-
UNIT TIME	2	31	Time in module
Reserved	3	-	-
UNIT ALARM	4	21	Alarm in module
Reserved	5	-	-
GET DATA	6	Varies	Retrieves stored data from module
Reserved	7	-	-
Reserved	8	-	-
ACK	-	1	0x21 ("!")
NACK	-	1	0x63 ("?")

Note: The length shown in this section is the package / payload length for that message. It does not include the header characters (0xAEAE), length byte, device ID byte, message ID byte, or checksum byte.

The messages available with content are follow the same format as the input messages with respect to header, id, payload and checksum. The only messages that do not abide by this are the ACK and NACK responses.

### 6.1.1 PING (ID 0x00)

Message		Name		
Description		PING		
ID		0x00		
Length		7		
Type		Output Message		
Comment		Response to ping request		
Message Structure	Header	Message ID	Payload	Checksum
	0xAE 0xAE	0x00	0xMM 0xNN 0xPP	XSUM

This message is in response to the 0x00 ping request.

The three bytes sent down (0xMM, 0xNN, and 0xPP) represent the software version on board. Major.Minor.Revision = MM.NN.PP.

### 6.1.2 RESERVED (ID 0x01)

Unused.

### 6.1.3 UNIT TIME (ID 0x02)

Message		Name			
Description		Unit Time			
ID		0x02			
Length		31			
Type		Output Message			
Comment		Time currently residing in the module			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x02	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	Time Structure	-	Current Time	-	Current time from the unit.

This message is a response to the request time message.

### 6.1.4 RESERVED (ID 0x03)

Unused.

### 6.1.5 UNIT ALARM (ID 0x04)

Message		Name			
Description		Unit Alarm			
ID		0x04			
Length		21			
Type		Output Message			
Comment		Gets the unit time.			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x04	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	Alarm Structure	-	Alarm	-	Desired alarm time for wakeup
16	U1	-	Alarm Type	-	Alarm type to wake up on: <ul style="list-style-type: none"> <li>• Seconds match</li> <li>• Minutes match</li> <li>• Hours match</li> <li>• Full date match</li> </ul>

This message is a response to the request alarm message.

### 6.1.6 RESERVED (ID 0x05)

Unused.

### 6.1.7 GET DATA (ID 0x06)

Message		Name			
Description		Get Data			
ID		0x06			
Length		Varies			
Type		Output Message			
Comment		Data stored in NVM			
Message Structure	Header	Message ID	Payload	Checksum	
	0xAE 0xAE	0x06	See Below	XSUM	
Byte offset	Number format	Scaling	Name	Units	Description
0	U2	-	Bytes being recovered	-	Number of bytes to recover (maximum 4096)
2,3,4....4097	U1	-	Data to store	-	Data bytes are stored up to length sent

This message is a response to request for stored data from the unit via message 0x06.



### **6.1.8 RESERVED (ID 0x07)**

Unused.

### **6.1.9 RESERVED (ID 0x08)**

Unused.

## 7. Software Reprogramming

The Sandman board can be reprogrammed in the field if necessary. In order to accomplish this, the user will have to pull down the ISP pin upon powering up. This puts the processor into boot loader mode, and new firmware can be uploaded via the serial port. Please contact us for more details.

## 8. Software Release Notes

Software version descriptions can be found here.

**Table 6 – Software Revision**

Major	Minor	Build	Description
1	1	1	Initial release

## 9. Appendix A: TBD

Placeholder