AUTHOR

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Department** |  | **Signature** |  | **Print Name** |  | **Date** |
| R&D |  |  |  | Ken Woodland |  |  |

SIGNATURES

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Department** |  | **Signature** |  | **Print Name** |  | **Date** |
| R&D |  |  |  |  |  |  |

Revision History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rev.** |  | **Date** |  | **Initials** |  | **Description** |
| 0.1 |  | 20-Sep-2006 |  | UH |  | Initial draft |
| 0.2 |  | 01-Nov-2006 |  | UH |  | Minor changes to align with tests |
| 0.3 |  | 15-Feb-2007 |  | KW |  | Only R&D Signature, fixed references |
| 0.4 |  | 24-Jul-2007 |  | MM |  | Modified section 2.1.f.3, reference to NAK bit unclear |
| 0.5 |  | 01-Jan-2008 |  | DT |  | Added POWERMINI and Fast MDU Accessory IDs |
| 0.6 |  | 23-Apr-2008 |  | KW |  | Added Software Update Start command, updated Program Page |
| 0.7 |  | 20-Jan-2009 |  | KW |  | Updated response times |
| 0.8 |  | 06-Apr-2009 |  | KW |  | Reference DYONICS II EIP System Specification |
| A |  | 16-Oct-2009 |  | DT |  | Initial Release |
| A.1 |  | 25-Mar-2019 |  | KW |  | Changed FAST MDU to Reliant MDU  Added Motor Table transfer  Added Serial Number transfer |
| A.2 |  | 12-Jun-2019 |  | KW |  | Created table with more detail for Motor Table transfer  Clarify Section 1.1 |
| A.3 |  | 17-Jun-2019 |  | KW |  | Changed Serial Number Command Request from 15 to 10 because hardware startup can generate a spurious 0xFF |
| A.4 |  | 26-Jun-2019 |  | KW |  | Moved the Motor Table and Serial Number special commands to the Reliant RS485 protocol specification |
| A.5 |  | 18-Jul-2019 |  | KW |  | Max Response Times for Accessory Specific Commands are 3ms and 8ms for Footswitch Commands 3-5 |
| A.6 |  | 10-Sep-2019 |  | KW |  | Corrected Program Page Command Timeout from 400ms to 500ms to reflect code |
| B |  | 24-Sep-2019 |  | DAT |  | Updated Revision to B |

References

1. 15000291 – DYONICS II System Specification
2. 15000694 – DYONICS II EIP System Specification

Glossary

ACK – Acknowledgement

CRC – Cyclic Redundancy Check

ECC – Error-Correcting Code

MCU – Microcontroller

MDU – Motor Drive Unit

NAK – Negative Acknowledgement

Table of Contents

[1 Overview 3](#_Toc19031383)

[1.1 Physical Layer 3](#_Toc19031384)

[2 Protocol Elements 3](#_Toc19031385)

[2.1 Communication 3](#_Toc19031386)

[2.2 Command requests and responses 3](#_Toc19031387)

[2.2.1 Accessory ID Request 4](#_Toc19031388)

[2.2.2 Software Update Start Request 4](#_Toc19031389)

[2.2.3 Program Page Command Request 4](#_Toc19031390)

[2.2.4 Reset Command Request 5](#_Toc19031391)

[2.2.5 CRC Byte 5](#_Toc19031392)

# Overview

This document describes the protocol used for communication between the DYONICS II and its accessories on the RS485 bus.

## Physical Layer

1. The physical layer used for the RS485 communication is an industry standard half-duplex RS485 differential bus. Although RS485 communication can support a single master transceiver and multiple simultaneous slave transceivers, this protocol only supports a single master transceiver paired with single slave transceiver.
2. Pin-outs, connectors, and cables are specified in the DYONICS II System Specification or the DYONICS II EIP System Specification.

# Protocol Elements

## Communication

1. 8 bit bytes are sent over the bus using no parity and one stop bit (8N1) at either 19200 baud (for footswitch accessories) or 57600 baud (for all other accessories).
2. The master initiates all communication by sending command requests. The only time a slave transmits on the bus is in response to a master command.
3. Command requests, except for the Program Page command, are one byte.
4. All slave responses are two bytes, unless otherwise specified.
5. Slave response times are defined under Max Response Times in section 2.2 for each command.
6. Forward error correction is used on both command requests and responses, to allow a receiver to perform one-bit error correction and two bit error detection. Command requests use (7,4) Hamming ECC encoding, while command responses use (15,11) Hamming ECC encoding.
7. For 8 bit command requests, bits 3-0 contain the actual command, thus there are 16 possible commands. Due to the possibility of a glitch on a serial line creating a false byte with value 0xff, the last command 0xf should not be used. Thus there are 15 usable commands.
   1. For ECC calculations, bits 3-0 represent logical data bits D4 – D1 respectively. Bits 4-6 contain ECC bits E1, E2 and E4 respectively, while bit 7 is the even parity bit over bits 6-0. Note that this parity bit is needed to detect two bit errors in software, thus the hardware parity at the RS485 level is not used.
8. For 16 bit command responses, bits 9-0 contain the actual response, whose format is determined by the specific request. Bit 10 is the ACK/NAK bit. Bits 11 through 14 are ECC bits E1, E2, E4 and E8 respectively for the data in bits 0-10, while bit 15 is the even parity bit over bits 14-0.
9. When a device gets a valid request, it should return a valid response with the ACK/NAK bit set to 1. When a device gets an erroneous request, it should return a response with the ACK/NAK bit set to 0, bits 9-0 cleared to zero, and the parity and ECC bits set accordingly.

## Command requests and responses

There are some generic commands that every RS485 accessory device must support. The following table lists the generic and reserved command requests:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command Request** | **Value** | **Encoded byte sent by Master: P E4 E2 E1 <Cmd>** | **Response bits 9-0** | **Max Response Times**  **(footswitch)** |
| Software Version | 0 | 0000 0000 (0x00) | Bits 9-8: Major (x.y.z format)  Bits 7-4: Minor  Bits 3-0: Build | 3ms |
| Accessory ID | 1 | 1011 0001 (0xB1) | Accessory ID (see 2.2.1 below) | 3ms |
| Accessory Specific Command | 2 | 1101 0010 (0xD2) | Accessory Specific Response | 3ms |
| Accessory Specific Command | 3 | 0110 0011 (0x63) | Accessory Specific Response | 3ms (8ms) |
| Accessory Specific Command | 4 | 1110 0100 (0xE4) | Accessory Specific Response | 3ms (8ms) |
| Accessory Specific Command | 5 | 0101 0101 (0x55) | Accessory Specific Response | 3ms (8ms) |
| Accessory Specific Command | 6 | 0011 0110 (0x36) | Accessory Specific Response | 3ms (No Response) |
| Accessory Specific Command | 7 | 1000 0111 (0x87) | Accessory Specific Response | 3ms (No Response) |
| Accessory Specific Command | 8 | 0111 1000 (0x78) | Accessory Specific Response | 3ms (No Response) |
| Accessory Specific Command | 9 | 1100 1001 (0xC9) | Accessory Specific Response | 3ms (No Response) |
| Special 1 | 10 | 1010 1010 (0xAA) | Special 1 | Special 1 (No Response) |
| Software Update Start | 11 | 0001 1011 (0x1B) | Bits 9-0: Zero (see 2.2.2 below) | 3ms (No Response) |
| Program Page | 12 | 1001 1100 (0x9C) | Bits 9-0: Zero (see 2.2.3 below) | 500ms |
| Reset | 13 | 0010 1101 (0x2D) | Bits 9-0: Zero (see 2.2.4 below) | 3ms (No Response) |
| Special 1 | 14 | 0100 1110 (0x4E) | Special 1 | Special 1 (No Response) |
| Reserved 2 | 15 | 1111 1111 (0xFF) | Reserved 2 |  |

1Special – Non-standard response data and timing defined in the accessory specific RS485 protocol.

2Reserved – Do not use, response to this request will be a NAK (Bits 15-0: Zero).

### Accessory ID Request

The following table shows the list of accessory device IDs that identify the type of device (footswitch etc.) These IDs shall be allocated by Smith & Nephew, and are sent in response to the Accessory ID request.

|  |  |
| --- | --- |
| **Accessory Device Type** | **Accessory Device ID** |
| Footswitch | 1 |
| POWERMINI MDU with Hand Controls | 2 |
| Reliant MDU with Hand Controls | 3 |
| Reliant MDU without Hand Controls | 4 |
| POWERMINI MDU without Hand Controls | 5 |

### Software Update Start Request

The accessory will stop processing that may get in the way of the software update that is about to take place. The only commands that still need to be responded to are Program Page and Reset. No other commands are to be sent during the software update process until the next power cycle or a Reset command is sent.

### Program Page Command Request

The accessory will attempt to program a 128 byte page of program memory on its MCU when it receives the program page command request. The format of the Program Page command is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Program Page (0x9C) | MS Byte of 128 byte Page # | LS Byte of 128 byte Page # | 128 Bytes of Flash Data | CRC Byte |

The 128 byte Page # is used to figure out what 128 byte offset to use in the MCUs program flash. Page #0 will start at program flash offset 0x0000, Page #1 is at offset 0x0080, Page #2 is at offset 0x0100, etc.

The total size of Program Page Command is 132 bytes.

The CRC Byte is calculated as an 8 bit CRC of all the data sent including the Program Page command byte. See the CRC Byte description in 2.2.5 below for details on calculating the CRC Byte

The ACK bit in the response to the program page commands will indicate the CRC state, as well as whether the actual flash write succeeded (after any retries if necessary).

The master can send the RESET command after all pages have been sent. See the RESET command description in 2.2.4 below for more details.

The accessory device should not change the contents of any other non-volatile memories as a result of this command.

This command may only be sent after a successful Software Update Start Request.

### Reset Command Request

Sending the Reset command will cause the MCU on the accessory device to reset. This is typically used after re-programming the device. The response bytes should be sent by the accessory just prior to the actual MCU reset. The accessory is not required to respond to any command requests until at least 4 seconds after the response to this command is received.

This command may only be sent after a successful Software Update Start Request.

### CRC Byte

The CRC Byte is calculated as an 8 bit CRC of all the data sent using the following table and sample code:

const SnByte pbCrcTable[] = // SnByte is 8-bit unsigned

{

/\*0\*/ 0, 94,188,226, 97, 63,221,131,194,156,126, 32,163,253, 31, 65,

/\*1\*/ 157,195, 33,127,252,162, 64, 30, 95, 1,227,189, 62, 96,130,220,

/\*2\*/ 35,125,159,193, 66, 28,254,160,225,191, 93, 3,128,222, 60, 98,

/\*3\*/ 190,224, 2, 92,223,129, 99, 61,124, 34,192,158, 29, 67,161,255,

/\*4\*/ 70, 24,250,164, 39,121,155,197,132,218, 56,102,229,187, 89, 7,

/\*5\*/ 219,133,103, 57,186,228, 6, 88, 25, 71,165,251,120, 38,196,154,

/\*6\*/ 101, 59,217,135, 4, 90,184,230,167,249, 27, 69,198,152,122, 36,

/\*7\*/ 248,166, 68, 26,153,199, 37,123, 58,100,134,216, 91, 5,231,185,

/\*8\*/ 140,210, 48,110,237,179, 81, 15, 78, 16,242,172, 47,113,147,205,

/\*9\*/ 17, 79,173,243,112, 46,204,146,211,141,111, 49,178,236, 14, 80,

/\*a\*/ 175,241, 19, 77,206,144,114, 44,109, 51,209,143, 12, 82,176,238,

/\*b\*/ 50,108,142,208, 83, 13,239,177,240,174, 76, 18,145,207, 45,115,

/\*c\*/ 202,148,118, 40,171,245, 23, 73, 8, 86,180,234,105, 55,213,139,

/\*d\*/ 87, 9,235,181, 54,104,138,212,149,203, 41,119,244,170, 72, 22,

/\*e\*/ 233,183, 85, 11,136,214, 52,106, 43,117,151,201, 74, 20,246,168,

/\*f\*/ 116, 42,200,150, 21, 75,169,247,182,232, 10, 84,215,137,107, 53

// 0 1 2 3 4 5 6 7 8 9 a b c d e f

};

SnByte CrcData(SnByte \*pbData, SnQByte qBytes)

{

SnByte bCrc = 0;

while (qBytes-- > 0) {

bCrc = pbCrcTable[bCrc ^ \*pbData++];

}

return bCrc;

}