ANIMATICS SmartMotor™ RS232/RS485 Communications Protocol

document date 03/01/02 Jason Hyatt

Introduction

This is a brief summary of SmartMotor™ communication protocol for users writing their own software to communicate with the motors. Please contact Animatics with any questions.

Overview

The basic protocol is:

Control characters have high order bit set
Commands are ASCII characters, high order bit is not set
Commands must be terminated by either a carriage return, space, or line feed.
An occurrence of a terminator character terminates the command being parsed (ie no white space within a command: P=100 is ok, P = 100 is not.
The terminator completes the parsing of ASCII commands. ASCII commands are not executed until the terminator character is received.
Multiple terminators are considered as null commands and ignored (ie white space is permitted between commands, and would take up bandwidth)

Control characters are non-ASCII characters with high order bit set and may be:
- motor addressing bytes hex 80 to hex F4 for motor addresses 0 to 116
  (ie hex 80 plus motor address, or, decimal 128 plus motor address)
- a few binary format commands, which are always followed by four characters of binary data. The binary data characters are not interpreted as control characters
- hex FF which is used to terminate the end of a downloading a program following the LOAD command

Commands for the command stream versus user program execution:

Some commands only have meaning when executed by a user program within the motor. These are:
- User program “pause” commands:
  - WAIT, TWAIT
- User program execution flow commands:
  - IF, ELSE, ELSEIF ENDIF
  - WHILE LOOP
  - SWITCH ENDSW CASE BREAK
  - Cnnn (program label) GOTO GOSUB

PRINT and PRINT1 are intended for use within the user program.

Some commands only have meaning when received in the command stream and do not have meaning within the motor. These are:

Control characters
UPLOAD UP RCKS LOAD

SmartMotor™ Interface Program (SMI)

The SMI application provided by Animatics uses some shortcuts for convenience that may be confusing when comparing the protocol to what appears on the SMI terminal window.

On the terminal window, the line entered is not actually sent to the motor until return is entered. Since space is a valid terminator, multiple commands may be sent from one line. The motor does not treat a space differently than a return.

On the terminal window, numerals preceding the commands are translated into motor address non-ASCII control characters.

SMIEngine COM Interface DLL

For users writing their own application software to talk to the motors, included with the SMI application software is the SMIEngine. The SMIEngine is a Microsoft Windows DLL that provide functions conforming to the COM interface that users may access in their language that supports the COM interface (VC++, VB, Borland C++, etc.).

These functions provides services such as opening the serial port, addressing the motors on an RS232 daisy chain, sending commands to the motors, streaming coordinated motion position and time command data to the motors, compiling a user program.

More information regarding these is available through SMI Help.

Simple Example

Sample Character stream and descriptions:

In this example there are two motors. Both are initialized with the same acceleration and velocity, each is given a different destination position, and both are commanded to “Go” with a single command.

The commands are:
<80>A=152<CR>
V=9900<SP>
<81>P=152<LF>
<82>P=85<CR>
<80>G<CR>

hex	ASCII	comment
80		control character, motor executes following commands regardless of its address
	A	ASCII motor command, all motors execute because of preceeding hex 80
	=	
	1	
	5	
	2	
0D	CR	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A
	V	ASCII motor command, all motors execute because of earlier hex 80
	=	
	9	
	9	
	0	
	0	
20	SP	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A
81		control character, motor 1 executes following commands, other motors do not
	P	ASCII motor command, motor 1 executes, other motors do not
	=	
	1	
	5	
	2	
0A	LF	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A
82		control character, motor 2 executes following commands, other motors do not
	P	ASCII motor command, motor 2 executes, other motors do not
	=	
	8	
	5	
0D	CR	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A
80		control character, motor executes following commands regardless of its address
	G	ASCII motor command, all motors execute
0D	CR	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A

Example Using Binary Format Commands

Sample Character stream and descriptions:

In this example binary format commands are used. Binary format commands allow achieving higher bandwidth when the values sent to the motor would be longer when expressed in ASCII decimal digits. See below for the list of binary format commands.

This is the same as the previous example, using binary format commands.

In this example there are two motors. Both are initialized with the same acceleration and velocity, each is given a different destination position, and both are commanded to "Go" with a single command.

hex	ASCII	comment
80		control character, motor executes following commands regardless of its address
FC		binary format motor command for ASCII command A= four bytes following binary format motor command are interpreted as binary data and not as a control control character
00		most significant byte of 32 bit hex value 00 00 00 98 = decimal 152
00		
00		
98		least significant byte of 32 bit hex value 00 00 00 98 = decimal 152 not interpreted as a control byte because within 4 bytes of earlier hex FC binary format command
0D	CR	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A
FD		binary format motor command for ASCII command V= four bytes following binary format motor command are interpreted as binary data and not as a control control character
00		most significant byte of 32 bit hex value 00 00 26 AC = decimal 9900
00		
26		
AC		least significant byte of 32 bit hex value 00 00 26 AC = decimal 9900 not interpreted as a control character
20	SP	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A
81		control character, motor 1 executes following commands, other motors do not
FE		binary format motor command for ASCII command P= four bytes following binary format motor command are interpreted as binary data and not as a control control character
00		most significant byte of 32 bit hex value 00 00 00 98 = decimal 152
00		
00		
98		least significant byte of 32 bit hex value 00 00 00 98 = decimal 152
0A	LF	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A
82		control character, motor 2 executes following commands, other motors do not
FE		binary format motor command for ASCII command P= four bytes following binary format motor command are interpreted as binary data and not as a control control character
00		most significant byte of 32 bit hex value 00 00 00 55 = decimal 85
00		
00		
55		least significant byte of 32 bit hex value 00 00 00 55 = decimal 85
0D	CR	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A

80		control character, motor executes following commands regardless of its address
	G	ASCII motor command, all motors execute
0D	CR	command terminator, any of: CR hex 0D, space hex 20, LF hex 0A

Communication Protocol States

The following paragraphs describe some relevant states the motor may be in regarding the communications protocol.

SmartMotor™ RS232/RS485 communications have several states:

Command Stream (channel is opened as a command channel):

Non-ASCII bit 8 set control bytes: 1xxx xxxx, hex 80 to hex FF
hex 80 to hex F4 specify motor addresses 0 to 116
hex F5 to hex FE are binary format commands

ASCII characters are motor commands, hex 00 to hex 7F

Each motor command is terminated by an end of line character, which may be either:
space (hex 20),
carriage return (hex 0D), or
new line or line feed (hex 0A)

Command Stream (channel is opened as a command channel) and LOAD command has been sent and has not yet terminated:

All bytes are interpreted as a user program download, and stored in the non-volatile (EEPROM) memory

The user program has a header, motor commands, EEPROM addresses relating to user program labels (Cnnn) and user program flow control statements (IF, WHILE, etc.).

Two bytes of hex FF terminate the user program LOAD state.

Command Stream (channel is opened as a command channel) and a binary format command control character has been sent (hex F5 to hex FE):

The four characters following the binary format command control character are binary data and are NOT interpreted either as a motor address control character, another binary format command control character, nor a

Using RS485 Channel 1

To use Channel 1as a host channel, it must be opened with the open channel command OCHN(…), and the F=4 command to direct report command responses emitted by the motor to channel 1. These commands may be issued from the RS232 channel or from an EEPROM program.

For example:

OCHN(RS4,1,N,19200,1,8,C)<space or cr> opens
- RS485
- channel 1
- No parity
- 19200 baud
- 1 stop bit
- 8 data bits
- interpreted as a command stream

F=4
- motor responses emitted on channel 1

Address Control Commands

An end of line character (space, carriage return, line feed) must follow a command or address control character immediately preceding an address control character.

A motor is in either an addressed state or de-addressed state.
When addressed, it executes commands. When de-addressed it ignores commands.
It may be addressed or de-addressed separately on each of RS232 channel 0, or RS485 channel 1.
On power-up/reset the motor is in an addressed state on both channel 0 and channel 1.

A motor becomes addressed on a channel by receiving on that channel:
- a global address byte x80 which addresses all motors
- an address byte = x80 plus the address of the motor.
  - For example: hex 81 = decimal 129 addresses motor with address 1

Once a motor has been addressed, it stays addressed until it receives the address of a different motor. Then it becomes de-addressed.

A motor's address is set by the SADDRn command. For example, if a program in the EEPROM runs on power up or reset, and executes SADDR1<space or cr>, the motor will have address 1.

Binary Format Commands

Position, Velocity, and Acceleration commands may be transmitted to the motor in binary format. In this format:

P=	is replaced by hexadecimal byte	xFE
V=	is replaced by hexadecimal byte	xFD
A=	is replaced by hexadecimal byte	xFC
Coordinated Motion Mode MD Time Data		xFB
Coordinated Motion Mode MD Position Data		xFA
Reserved		xF9
Reserved		xF8
Reserved		xF7
Reserved		xF6
Reserved		xF5

and the ASCII decimal digits of the commanded value are replaced by the big-endian four byte binary integer, followed.

The binary format command requires an end-of-line delimiter (carriage return, space, or line feed).

The binary format command may be immediately preceeded by an address control character.

For Example:

ASCII command byte stream	Binary command byte stream
[x81]P=1000000<space>	[x81][xFE][x00][x0F][x42][x40][x20]

Command RCS
Report Checksum

Returns the current 8 bit checksum of all bytes addressed to this specific motor (even while SLEEPing) since prior RCS or reset was issued.

An address control byte addressing the motor is included in the checksum. An address control byte addressing a different motor (de-addressing the motor) is not included in the checksum.

This checksum includes the RCS1 characters and the delimeter character.

EXAMPLE FOR MOTORS 1 AND 2:

(<space> following commands sent)

sent to motors	received from motors	
[x81]RCS1<sp>	<accumulated checksum>	clear prior checksum
[x82]2RCS1<sp>	<accumulated checksum>	clear prior checksum
[x81]RCS1<sp>	186<cr>	
[x82]RCS1<sp>	187<cr>	(redundant to show values)
[x81]P=100<sp>		set motor 1 new position
[x82]P=200<sp>		set motor 2 new position
[x80]G<sp>		both motors go begin trajectory
[x81]RCS1<sp>	96<cr>	verify motor 1 checksum position and go
[x82]RCS1<sp>	99<cr>	verify motor 2 checksum position and go

INPUT CHARACTER STREAM	DEC	HEX	CONT 1 HEX SUM	CONT 2 HEX SUM	1BYTE DEC CHECK-SUM	comment
			00 00	00 00		
x81	129	81	00 81			address motor 1, de-address 2
R	82	52	00 d3			
C	67	43	01 16			
S	83	53	01 69			
1	49	31	01 9a			
<space>	32	20	01 ba		186	
			00 00			motor 1 checksum cleared
x82	130	82		00 82		address 2, de-address 1
R	82	52		00 d4		
C	67	43		00 17		
S	83	53		01 69		
1	49	31		01 9a		
<space>	32	20		01 bb	187	

00 00	motor 2 checksum cleared

x81	129	81	00 81			address 2, de-address 1
P	80	50	00 d1			
=	61	3d	01 0e			
1	49	31	01 3f			
0	48	30	01 6f			
0	48	30	01 9f			
<space>	32	20	01 bf			
x82	130	82		00 82		
P	80	50		00 d2		
=	61	3d		01 0f		
2	50	32		01 41		
0	48	30		01 71		
0	48	30		01 a1		
<space>	32	20		01 c1		
x80	128	80	02 3f	02 41		address both 1 and 2
G		47	02 86	02 88		both are adding to checksum
<space>		20	02 a6	02 a8		
x81	129	81	03 27			address 1, de-address 2
R	82	52	03 79			
C	67	43	03 bc			
S	83	53	04 0f			
1	49	31	04 40			
<space>	32	20	04 60		96	
			00 00			
x82	130	82		03 2a		
R	82	52		03 7c		
C	67	43		03 bf		
S	83	53		04 12		
1	49	31		04 43		
<space>	32	20		04 63	99	
				00 00		