Protocol specification for SCHUNK Five Finger Hand

by:

FZI

FZI Forschungszentrum Informatik

Stiftung des Bürgerlichen Rechts

Haid-und-Neue-Str.- 10-17

76131 Karlsruhe

S5FH – Protocol specification

This document gives an overview of the software protocol used to control the SCHUNK Five Finger Hand. We provide this information "as is" and in the hope it will be usefull to you, however the protocol might change or behave differently so we are not liable for any damage resulting from using this information. Please always monitor the currents when developing software for the five finger hand. The protocol information is based on the Firmware v1.1 provided by MeCoVis.

Version History

version	date	author	changes	reviewed
0.1	10.02.14	Georg Heppner	Created initial version	Lars Pfotzer
0.2	19.03.14	Georg Heppner	Reviewed data and filled in missing commands	Lars Pfotzer
0.3	19.03.14	Georg Heppner	Minor changes	
0.4	19.03.14	Arne Rönnau	Changes picture & minor changes	
1.0	30.09.14	Georg Heppner	Finalized the document for release	

Interface Settings

Default settings used by the controller:

- Baudrate: 921600
- Parity : None
- Data Bytes: 8
- Stop Bits : 1

The RS485 serial interface is used as a full duplex point to point connection.

Data Alignment

All packets are send with little endian encoding.
Note: All values given in this specification (except raw data examples) are written in standard hexadecimal or decimal format, see the raw byte examples to check what you are sending.

Packet Structure

The S5FH is controlled via RS485 serial interface. Packets consist of raw data, synchronization bytes, an address and checksum information. Each packet starts with two synchronization bytes SYNC1 **(0x4C)** and SYNC2 **(0xAA)**. The following index should be continuously incremented by the master to map incoming responses to send requests. The address byte encodes the meaning of the following data, length gives the number of data packets. The checksums CHECK1 and CHECK2 verify the data integrity of the raw data fields. CHECK1 is the byte sum of all data bytes. CHECK2 the xor of all data bytes.

Sync1	Sync2								
Index	Add	Length							
Data	Data	Data	Data	Data	Data	Data	Data	Data	Data
…	…	…	…	…	…	…	…	…	…
Check1	Check2								

Checksum Calculation:

```
foreach (byte value in packet.Data)
 {
    checkSum1 += value;
    checkSum2 ^= value;
 }
```

Address Constants

Address	Name	Description
0x00	GetControlFeedback	Request the current position and current (of a specific channel) to be sent
0x01	SetControlCommand	Set the target position of a specific channel
0x02	GetControlFeedback AllChannels	Requests the current position and current of all channels to be sent
0x03	SetControlCommand AllChannels	Sets the target position of all channels simultaneously
0x04	GetPositionSetting	Request the slave to transmit the current position controller settings
0x05	SetPositionSetting	Set the position controller settings
0x06	GetCurrentSettings	Request the slave to transmit the current motor current controller settings
0x07	SetCurrentSettings	Set the motor current controller settings
0x08	GetControllerState	Unknown, but probably returns the state of the main controller
0x09	SetControllerState	Activate, deactivate and reset the main controller circuit
0x0A	GetEncoderValues	Request the slave to transmit the current encoder settings
0x0B	SetEncoderValues	Set the encoder settings to given values
0x0C	GetFirmwareInfo	Request the slave to transmit firmware information

Abbreviations

In the following sections some values will be represented by placeholders:

- 0xNN = Index (counted up by the master)
- 0xSS = Checksum value (calculated out of raw data)
- 0xTT = Value calculated from channel

Channel Selection

The address field is used for both, encoding the meaning of a command and its recipient. The channel number (the number of the motor to control) is given in the upper nibble of the address byte.

Address = Address Constant | (channel << 4)

Data Length

Although the length field should specify the number of bytes following, the package length of commands sent to the controller (PC -> hand) are always 40 Byte. Returning packages always have a length of 64 Bytes and are padded with zeros.

SetControllerState - S5FH ControllerState

To send a control command a number of controller state variables have to be sent. The controller state enables control of the motors on a very low level. This command is explained first as it has to be used to enable or disable the controllers. The data structure is as follows:

0x4C	0xAA								
Index	0x09	0x0C							
pwm_fault		pwm_otw		pwm_reset		pwm_active		pos_ctrl	
cur_ctrl									
Check1	Check2								

- **pwm_fault**
  0x001F → Reset of controller faults (only used internally)
- **pwm_otw**
  0x001F → Reset of controller faults from temperature warnings (over temperature waring)
- **pwm_reset**
  Reset pwm generation in the motor controller. 0X0200 is used as a special value to turn on the 12V supply driver.
  0x0200 → Enable +12V supply driver
- **pwm_active**
  Currently unused, should be activated like pwm_reset.
- 0X0200 is used as a special value to turn on the 12V supply driver.
  0x0200 → Enable +12V supply driver
- **pos_ctrl**
  0x0001 → Enable position controller
- **cur_ctrl**
  0x0001 → Enable current controller
-

Note: Remember that all values have to be transmitted as little endian (intel architecture)
(Example a PWM Fault Value of 0x001F will be transmitted as:
0x4C 0xAA 0xNN 0x09 0x0C 0x1F 0x00 .....)

Activating all channels:
When activating all channels (i.e all fingers) the packet should be send incrementally in three messages leaving 2ms delay between each packet. While the first contains pwm_fault and pwm_otw, the second contains additionally pwm_reset and pwm_active, the last contains additionally position and current control. The exact messages send would be:

```
0x4C 0xAA 0xNN 0x09 0x0C 0x1F 0x00 0x1F 0x00 0x00 0x00  0x00 0x00  0x00 0x00  0x00 0x00 0xSS 0xSS
<2ms delay>
0x4C 0xAA 0xNN 0x09 0x0C 0x1F 0x00 0x1F 0x00 0x00 0x02  0x00 0x02  0x00 0x00  0x00 0x00 0xSS 0xSS
<2ms delay>
0x4C 0xAA 0xNN 0x09 0x0C 0x1F 0x00 0x1F 0x00 0x00 0x02  0x00 0x02  0x01 0x00  0x01 0x00 0xSS 0xSS
<2ms delay>
```

NOTE: The 2ms delay are used to give the supply drivers time to settle.

Activating individual channels
When activating individual channels one packet can be send (instead of the three individual ones). If any finger was already activated before, the control values stay the same, except for pwm_reset and pwm_active which are activated by an additional Bit-Flag

```
enableMask = (1 << channel)
pwm_reset  = (0x0200 | (enableMask & 0x01FF));
pwm_active = (0x0200 | (enableMask & 0x01FF));
```

This will result in the following packet:

```
0x4C 0xAA 0xNN 0x09 0x0C 0x1F 0x00 0x1F 0x00 0xBB 0x0T  0xTT 0x0B  0x01 0x00  0x01 0x00 0xCC 0xSS
```

Deactivating all channels
For deactivation a ControllerStatemessage with just *pwm_fault* and *pwm_otw* is send (setting the controllers to 0):

```
0x4C 0xAA 0xNN 0x09 0x0C 0x1F 0x00 0x1F 0x00 0x00 0x00  0x00 0x00  0x00 0x00  0x00 0x00 0xCC 0xSS
```

Deactivating individual channels
Deactivating individual channels is done in the same way as activating them but setting the enableMask bit to 0 instead of 1.

NOTE: At activation of channels we currently use a two step approach where we first reset all the faults by writing to pwm_fault and pwm_otw and activate the driver units by writing the bitmask to pwm_reset and pwm_active. pwm_reset is an low active signal resetting the pwm generators. After a very short delay we activate the position and current controller. If the position and current controllers are activated together with the pwm_reset a jumping behavior of the fingers may occur possible due to internal integration of values.

GetControlFeedback

0x4C	0xAA								
Index	0xT0	Length							
0x00	0x00	0x00	0x00	….	...	0x00	0x00	0x00	0x00
Check1	Check2								

To request control feedback the GetControlFeedback command is send together with the channel to be requested shifted to the upper nibble of the address byte:

Address = 0x00 | (channel << 4)

A GetControlFeedback for channel 6 would therefore result in the message:

```
0x4C 0xAA 0xNN 0x60 0x28 0x00 0x00  ....  0x00 0x00  0x00 0x00 0xSS 0xSS
```

The control feedback consists of:

- 32bit signed number (int) for the position
- 16bit signed number (short) for the motor current.

SetControlCommand

Sync1	Sync2								
Index	0x01	0x04							
Target Position									
Check1	Check2								

The control command consists of a single 32 bit signed integer indicating the target position of the controller given in encoder ticks.
A command setting the Goal of the controllers to the value 3523 would look like this:

```
0x4C 0xAA 0xNN 0x01 0x04 0xC3 0x0D 0x00 0x00 0xCC 0xCC
```

GetControlFeedbackAllChannels

0x4C	0xAA								
Index	0x02	Length							
0x00	0x00	0x00	0x00	….	...	0x00	0x00	0x00	0x00
Check1	Check2								

This command requests the feedback for all channels at once. The Feedback will first contain all the positions (32bit) and all the currents afterward:

0x4C	0xAA								
Index	0x02	0x40							
Pos 0				Pos 1				Pos 2	
Pos 2		Pos 3				Pos 4			
Pos 5				Pos 6				Pos 7	
Pos 7		Pos 8				Current 0		Current 1	
Current 2		Current 3		Current 4		Current 5		Current 6	
Current 7		Current 8							
Check1	Check2								

The control feedback consists of:

- 32bit signed number (int) for the position
- 16bit signed number (short) for the motor current.

SetControlCommandAllChannels

0x4C	0xAA								
Index	0x03	0x28							
Pos 0				Pos 1				Pos 2	
Pos 2		Pos 3				Pos 4			
Pos 5				Pos 6				Pos 7	
Pos 7		Pos 8				Pos 9			
Check1	Check2								

This commands sends the target positions of all channels at once. As a result the same response as for the command GetControllerFeedbackAllChannels is sent. All positions are 32bit signed numbers (int).

GetPositionSettings

0x4C	0xAA								
Index	0xT4	Length							
0x00	0x00	0x00	0x00	….	...	0x00	0x00	0x00	0x00
Check1	Check2								

To read out the current position settings the same Syntax as GetControlFeedback is used but with the address value of GetPositionSetting (0x04). The response will contain the settings in the same order as the SetPositionSettings packet.

SetPositionSetting

0x4C	0xAA								
Index	0xT5	0x28							
wmn				wmx				dwmx	
dwmx		ky				dt			
imn				imx				kp	
kp		ki				kd			
Check1	Check2								

All values are given as a 4 byte single precision float value. The meaning of these values can be taken from the MeCoVis S5FH controllers user guide.

The position controller evaluates the position difference to generate a current target for the cascaded current controller. The error signal is generated in ticks. The controlled unit is [mA]

- *wmn :* Reference signal minimum value
  Minimum allowed position input [encoder ticks]
- *wmx :* Reference signal maximum value
  Maximum allowed position input [encoder ticks]

- *dwmx :* Reference signal delta maximum threshold
  Maximum Allowed tick difference -> Max Speed of the finger in [ticks]
- *ky :* Measurement scaling
  Feedback scaling. Set to 1 to work in ticks (should not be changed)
- *dt :* Time base of controller
  Time base of the position controller (should not be changed)
- *imn :* Integral windup minimum value
  Minimum allowed value for the integrator [mA*tick/S]
- *imx :* Integral windup maximum value
  Maximum allowed value for the integrator [mA*tick/S]
- *kp :* Proportional gain
- *ki :* Integral gain
- *kd :* Differential gain

GetCurrentSettings

0x4C	0xAA								
Index	0xT6	Length							
0x00	0x00	0x00	0x00	….	...	0x00	0x00	0x00	0x00
Check1	Check2								

To read out the current motor-current settings the same Syntax as GetControlFeedback is used but with the address value of GetCurrentSetting (0x06).The response will contain the settings in the same order as the SetCurrentSettings packet.

SetCurrentSettings

0x4C	0xAA								
Index	0xT7	0x28							
wmn				wmx				ky	
ky		dt				imn			
imx				kp				ki	
ki		umn				umx			
Check1	Check2								

All values are given as a 4 byte single precision float value. The meaning of these values can be taken from the Mecovis S5FH controllers user guide.

The current controller uses the current control values generated by the position controller to generate a pwm signal for the driver units. The Error signal is calculated in [mA] the controlled unit are duty cycles

- *wmn :* Reference signal minimum value (reference is generated by position controller)
  Minimum allowed current value [mA]
- *wmx :* Reference signal maximum value (reference is generated by position controller) Maximum allowed current value [mA]
- *ky :* measurement scaling
  Feedback scaling. Do not change to work in [mA]

- *dt :* time base of controller
  Should not be changed. The current controller needs to work faster than the position controller
- *imn :* Integral windup minimum value
  Minimum allowed value for the integrator [1/(mA*s)]
- *imx :* Integral windup maximum value
  Maximum allowed value for the integrator [1/(mA*s)]
- *kp :* proportional gain
- *ki :* Integral gain
- *umn :* Output limiter min
  The output is the duty cycle of the driver units -> -255 - 0
- *umx :* Output limiter max
  The output is the duty cycle of the driver units -> 0-255

GetControllerState

0x4C	0xAA								
Index	0x08	Length							
0x00	0x00	0x00	0x00	….	...	0x00	0x00	0x00	0x00
Check1	Check2								

GetEncoderValues

0x4C	0xAA								
Index	0x0A	Length							
0x00	0x00	0x00	0x00	….	...	0x00	0x00	0x00	0x00
Check1	Check2								

Requests the current encoder settings (scaling of the encoders)

SetEncoderValues

0x4C	0xAA								
Index	0x0B	0x28							
Encoder 0				Encoder 1				Encoder 2	
Encoder 2		Encoder 3				Encoder 4			
Encoder 5				Encoder 6				Encoder 7	
Encoder 7		Encoder 8				Encoder 9			
Check1	Check2								

Sets the encoder values (scaling) to given values for each of the motors.
Usually this should not be necessary.

GetFirmwareInfo

0x4C	0xAA		
Index	0x0C	Length	
0x00	0x00	0x00	0x00
Check1	Check2		

Requests the slave to send firmware information.

Firmware information consists of:

- 4 byte char values in UTF8 encoding (always S5FH)
- 16 bit (short) major version number
- 16 bit (short) minor version number
- 48 byte string in in UTF8 encoding