Communication Protocol Proposal 1.0.3

**Version History:**

1.0.0 – First issue by Joonatan Renel  
1.0.1 – Added chip select pin to slave. Added physical signal description.  
1.0.2 – Added response code descriptions.  
1.0.3 – Added command descriptions

Contents

[1 Introduction 1](#_Toc529258390)

[2 Communication Protocol overview 2](#_Toc529258391)

[2.1 SPI physical communication layer 2](#_Toc529258392)

[2.2 SPI communication model 3](#_Toc529258393)

[2.3 Packet structure 4](#_Toc529258394)

[2.4 Command Descriptions 4](#_Toc529258395)

[2.4.1 Response Codes 5](#_Toc529258396)

[2.5 CRC checksum calculation 5](#_Toc529258397)

[3 Error handling 6](#_Toc529258398)

[4 Example communication 6](#_Toc529258399)

[5 Command formats 7](#_Toc529258400)

# Introduction

This document outlines a general low-level inter-processor communication design that will be be used to drive a stepper motor controller. This is a draft and details are subject to change.

# Communication Protocol overview

Communication between master and slave shall be realised over as a classical 3-wire SPI solution. The master shall send packets with a known interval. Since the master device controls the SPI clock, then the master also dictates when the slave can give a response.



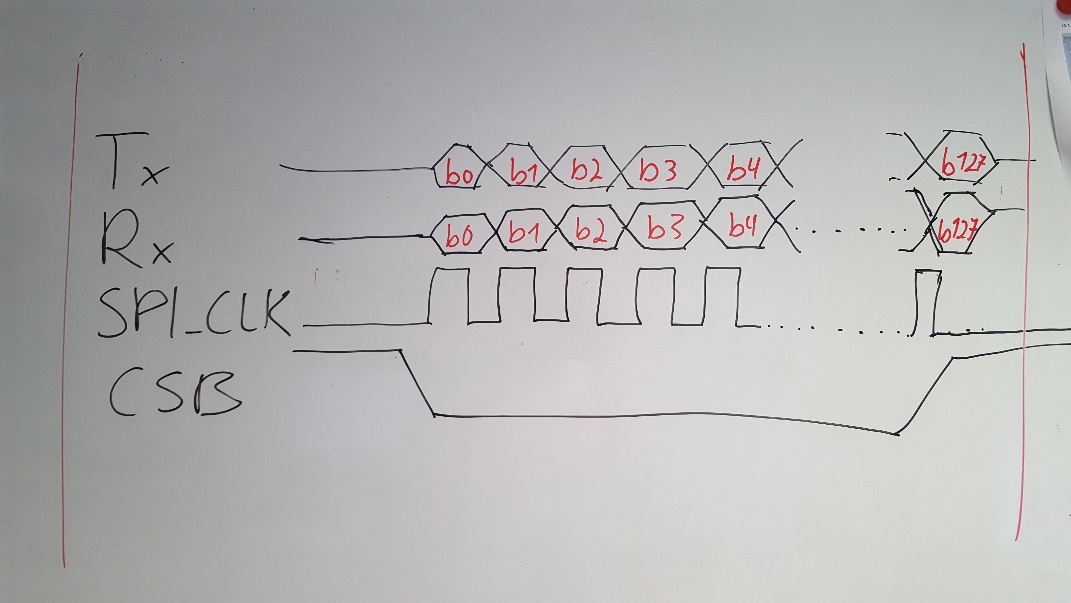
## SPI physical communication layer

The SPI interface is realized over 4 wires between the slave and master.

|  |  |
| --- | --- |
| **Pin** | **Function** |
| Tx | Transmit line (from master’s perspective) |
| Rx | Receive line (from master’s perspective) |
| CLK (SPI Clock | SPI clock driven by master. |
| CS (CSB) | Chip select, driven by master |
|  |  |

SPI communication shall begin with the master driving the CS pin low. This will be followed by a 128-byte long data packet. The chip select pin will help to synchronize communication and recover from potential data corruption.

**Example of low level data communication:**



## SPI communication model

SPI messages shall be exchanged at fixed intervals. Currently the planned interval for data packets is 50 milliseconds. This corresponds to a data rate of about 20 packets (command->response) per second.

The master sends commands to the slave device and receives a response to the previous command simultaneously. The slave has 40 milliseconds to create an answer for the previous command. The data packets contain a header and a CRC checksum at the end. The packets are all 128 bytes long. Empty byte slots at the end are padded with 0xff characters.

**Communication example:**



**Communication intervals:**



|  |  |
| --- | --- |
| **Characteristic** |  |
| Bit rate (clock frequency) | 100 kHz |
| Baud rate (bytes per second) | 12500 |
| Packet interval | 50 ms |
| Packet length | 128 bytes |
| Time for transmitting 1 packet | 10ms |
|  |  |

## Packet structure

Each packet has a fixed length of 128 bytes, but this does not all have to contain meaningful data.

|  |  |  |
| --- | --- | --- |
| **Packet contents** | **Bytes** | **Comment** |
| message header bytes | 0-1 | 0xFFFE - Signifies start of a packet |
| Packet length | 2-3 | Total length of packet N |
| Command ID | 4 | Command code (in case of slave response this is the command  that it is answering to) |
| Subcommand ID | 5 | Subcommand code (depends on cmd id) |
| Response code | 6 | Ack/Nack (used for response, always 0x00 when master) |
| Data bytes | 7 - (N - 3) | These contain data, arguments relevant to command. |
| CRC checksum | (N-2) – (N – 1) | 2-byte CRC checksum |
| Padding | (N) - 127 | End of message is padded with 0xffs. |

## Command Descriptions

This is a very preliminary list and just contains a proposal of possible commands. It is bound to change in the future.

|  |  |  |
| --- | --- | --- |
| **Command** | **Command ID** | **Description** |
| No command | 0x00 | Idle command, just for keeping alive (not sure if necessary) |
| Report Status | 0x01 | Motor controller will report status of itself and motors. |
| Set motor speed | 0x02 | Used to control motor speed, data part contains desired speed |
| <other commands> |  | TBD |
|  |  |  |

### Report Status command

This command is sent periodically. It is the default command. The response contains information about each stepper motor, their speed etc... This command is also preliminary and can be expanded in the future.

**Command format**:

|  |  |  |
| --- | --- | --- |
| **CMD id** | **Sub** | **Resp** |
| 0x01 | 0x00 | 0x00 |

**Response format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Stepper 0 | | | | | Stepper 1..3 |
| **CMD id** | **Sub** | **Resp** | **data 0** | **data 1** | **data 2** | **data 3** | **data 4** | .... |
| 0x01 | 0x00 | 0x00 | Step | Interval | | Speed (RPM) | | .... |

### Set Motor speed command

This command is used to set the stepper motor speed in RPM. The command can set the speed of all 4 stepper motors within a single command. The sub command contains the flags for motors that are accessed with this command. The speed for each motor is described in 8 data bytes. If the corresponding motor flag is not set, then the motor speed shall be ignored. For example if it is desired to set a speed for only M2, then only the flag for M2 should be set. The speed for other motors can be 0xFFFF or 0x0000, it will not be changed anyway in that case.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sub cmd:** |  |  |  |  |  |  |  |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| RES | | | | M3 | M2 | M1 | M0 |

**Command format:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **cmd\_id** | **sub** | **Resp** | **data 0** | **data 1** | **data 2** | **data 3** | **data 4** | **data 5** | **data 6** | **data 7** |
| 0x02 | m flags | 0x00 | M0 | | M1 | | M2 | | M3 | |

The response to the command shall report the result of the operation and the current speed of all stepper motors.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Response:** |  |  |  |  |  |  |  |  |  |  |
| cmd\_id | sub | code | data 0 | data 1 | data 2 | data 3 | data 4 | data 5 | data 6 | data 7 |
| 0x02 | flags | 0x00 | M0 speed | | M1 speed | | M2 speed | | M3 speed | |

### Response Codes

|  |  |  |  |
| --- | --- | --- | --- |
| **Response Code** |  | **Value** | **Description** |
| 0x00 |  | ACK | Everything is OK |
| 0x01 |  | NACK | General ERROR |
| 0x02 |  | Unknown Command |  |
| 0x03 |  | Incorrect Format |  |
| 0x04 |  | Checksum Error |  |

## CRC checksum calculation

The SPI message shall contain a 2-byte CRC checksum code at the end. This is calculated over the entire message using a 16-bit CRC-CCITT algorithm. The checksum can be used to detect possible communication errors.

# Error handling

The response packet will contain an ACK/NACK field that can be used to report that an error occurred when processing the previous message. If the slave reports a NACK, then it will be up to the master to decide how to proceed.

Also it is recommended to have a single separate wire between the steppermotor controller and the master device. This should be used to report a major error case, so that a failure on the controller can be transmitted independently of the SPI communication protocol. The logical line should use inverted logic (Output is High = No Error), so that it is possible to detect even cases where the stepper motor controller loses power.

# Example communication

**Example packet:**



This is an example packet of a set motor state command. Note that the command format itself is yet to be defined. In the example Bytes 7-13 should contain arguments for the set motor command (speed, direction, stepping mode, etc).

**Example communication sequence:**



# Command formats

<To be defined>

Here we will define command formats once the low-level protocol has been agreed on.