Communication Protocol Proposal 1.0.2

**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.

Contents

[1 Introduction 1](#_Toc528743455)

[2 Communication Protocol overview 2](#_Toc528743456)

[2.1 SPI physical communication layer 2](#_Toc528743457)

[2.2 SPI communication model 3](#_Toc528743458)

[2.3 Packet structure 4](#_Toc528743459)

[2.4 Command Ids 4](#_Toc528743460)

[2.5 CRC checksum calculation 4](#_Toc528743461)

[3 Error handling 5](#_Toc528743462)

[4 Example communication 5](#_Toc528743463)

[5 Command formats 6](#_Toc528743464)

# 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.



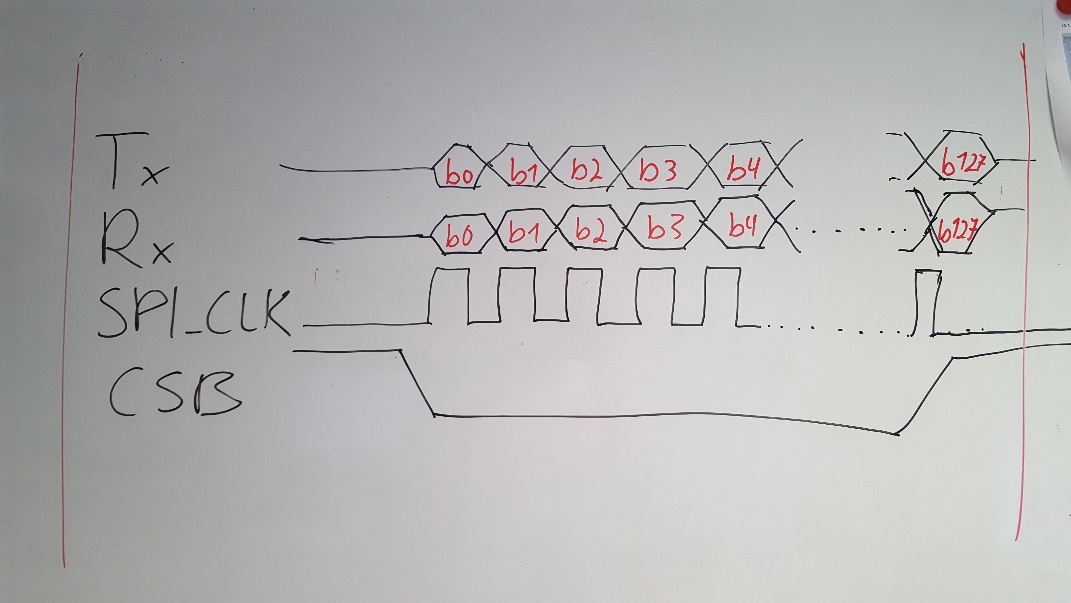
## SPI physical communication layer

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 Ids

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 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Response Code** | **Value** | **Description** |
| 0x00 | ACK | Everything is OK |
| 0x01 | NACK | General ERROR |
| <more codes> | TBD | TBD |

## 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.