Hemro; EKX Human Machine Interface (HMI)

Hemro EKX HMI

**Interface Specification**

**for Internal RS232 Communication between**

**HMI and Motor Control Board**

Document name: HEM-EKX-HMI-MotorControlInterfaceSpecification

Date: 15.11.2022

Author: Gerd Esser, Research & Development

Revision: 100-004

Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author** | **Description** |
| 100-001 | 29.05.2020 | Gerd Esser | Initial creation |
| 100-002 | 15.06.2020 | Gerd Esser | Rework according to IPM comments onto revision 001 and Telco from 08.06.2020 |
| 100-003 | 31.07.2020 | Gerd Esser | * Split status message into two chapters for HMI and MC side (4.2.2 and 4.2.3) * Extend MC status message in chapter 4.2.3 to 2 bytes payload and rework content * Extend simulation mode message in chapter 4.2.14 to 2 bytes payload size and rework content * Introduce generic assumption for timeout values in software update process (chapters 4.2.15-4.2.18) * Restrict MC reset behaviour in 4.2.19 * Adding new NACK codes in chapter 4.2.5 |
| 100-004 | 15.11.2022 | Gerd Esser | * Adding timeouts to SW update messages (4.2.15, 4.2.16, 4.2.17) * Adding description of motor config parameters in 4.2.9 * Split SAFELOCK flag into hopper and chamber flags in 4.2.3 and 4.2.14 * Insert sequence diagram for alive mechanism to 4.1.5 * New chapter 4.1.7 post initialization process |

Distribution list

|  |  |  |
| --- | --- | --- |
| **Name** | **Function** | **Company** |
| Florian Schlaadt | UX Project Leader | Ultratronik GmbH |
| Gerd Esser | UX Team Leader | Ultratronik GmbH |
| Gianni D'Ambrosio | IPM Project Leader | I.P.M. Technologies s.r.l. |
| Christoph Meier | Research & Test Laboratory Manager | HEMRO Group |
| Dr. Arnaldo Rodriguez | Head of Innovation & Technology | HEMRO Group |

Table of contents

[1 Intention 4](#_Toc119592258)

[2 System description 4](#_Toc119592259)

[3 RS232 connection 4](#_Toc119592260)

[3.1 Cable 4](#_Toc119592261)

[3.2 Communication parameters 5](#_Toc119592262)

[4 Protocol description 5](#_Toc119592263)

[4.1 Common rules 5](#_Toc119592264)

[4.1.1 Protocol data frame 5](#_Toc119592265)

[4.1.2 Transaction protocol 6](#_Toc119592266)

[4.1.3 CRC16 calculation 6](#_Toc119592267)

[4.1.4 Acknowledgement of messages 6](#_Toc119592268)

[4.1.5 Alive monitoring 7](#_Toc119592269)

[4.1.6 Message retries 8](#_Toc119592270)

[4.1.7 Post initialization process 8](#_Toc119592271)

[4.2 Message descriptions 10](#_Toc119592272)

[4.2.1 Message overview 10](#_Toc119592273)

[4.2.2 HMI Status 10](#_Toc119592274)

[4.2.3 MC Status 10](#_Toc119592275)

[4.2.4 ACK 11](#_Toc119592276)

[4.2.5 NACK 12](#_Toc119592277)

[4.2.6 Message request 12](#_Toc119592278)

[4.2.7 Motor actuation 12](#_Toc119592279)

[4.2.8 Product identification data 13](#_Toc119592280)

[4.2.9 Motor configuration 13](#_Toc119592281)

[4.2.10 Motor temperature NTC sensor 14](#_Toc119592282)

[4.2.11 MC board temperature 14](#_Toc119592283)

[4.2.12 Motor actuation info 14](#_Toc119592284)

[4.2.13 Motor DC bus voltage 15](#_Toc119592285)

[4.2.14 Simulation mode 16](#_Toc119592286)

[4.2.15 Software update start 16](#_Toc119592287)

[4.2.16 Software update data 17](#_Toc119592288)

[4.2.17 Software update finish 18](#_Toc119592289)

[4.2.18 Software update reject 18](#_Toc119592290)

[4.2.19 MC reset 19](#_Toc119592291)

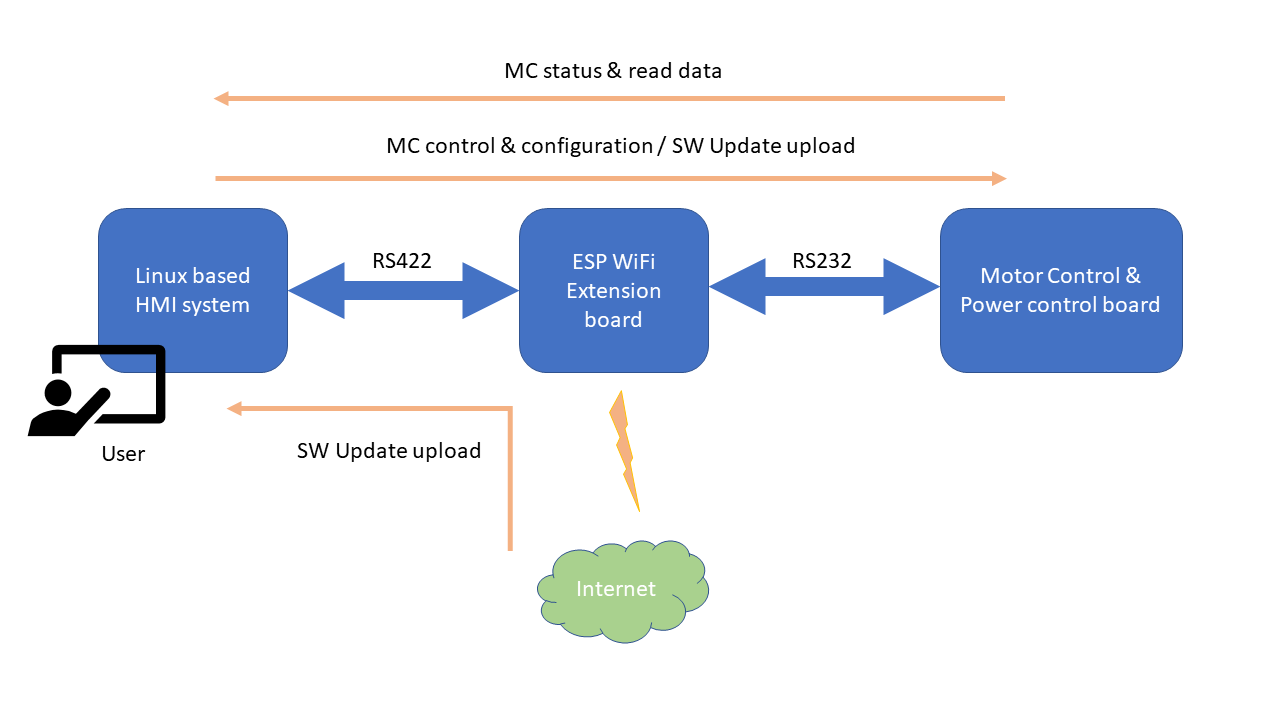
[5 Glossary 20](#_Toc119592292)

# Intention

This document describes the interface between the Hemro EKX HMI subsystem and Motor Control subsystem on data protocol layer within Hemro EKX project. This is to harmonize the integration of subsystems, which will be manufactured by different suppliers: Ultratronik GmbH for HMI, IPM-Technologies for Motor Control and Hemro as system responsible.

# System description

The relevant system architecture related to the Motor Control is shown in the following figure.



Motor Control is connected to ESP / WiFi extension board of the system by a RS232 interface. But all signals will simply be passed through ESP / WiFi extension board to the logic system master, which is the Linux based HMI board. Therefore, protocol implementations according to this document should be present on HMI board and Motor Control board.

Overall functional requirements due to HMI / Motor Control interface will be

* Power supply for HMI and subsequent systems (not scope of this document)
* Motor actuation from HMI side
* Status and sensor data transmission from Motor Control (motor status, motor current, board temperature, hopper status etc.)
* Software update capability of Motor Control
* Light weight simulation capability of error status for system test / self-test
* Alive monitoring

# RS232 connection

ESP and Motor Control boards are interconnected through a RS232 line.

## Cable

The cable is a 6-wire connection. The pin assignment is as follows:

|  |  |
| --- | --- |
| **Pin #** | **Description** |
| 1 | RX (Motor Control to ESP) |
| 2 | TX (ESP to Motor Control) |
| 3 | +24V (Power supply for ESP, HMI and subsequent boards) |
| 4 |
| 5 | GND |
| 6 |

## Communication parameters

For RS232 communication line between ESP and Motor Control board, following parameters shall be applied:

* 115200 Baud
* 8n1 (8 bits of data, no parity and one stop bit)
* Full duplex (RX and TX line)
* Asynchronous transmission (no additional serial clock line)
* No handshake

On HMI side a virtual COM port will be established and connection parameters can be independently selected from physical layer.

# Protocol description

## Common rules

### Protocol data frame

For communication a common data frame format will be used which is protected by CRC16.

Little endian format will commonly be used within the data frame.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Preamble** | **Message Type** | **Transaction ID** | **Payload Length** | **Payload** | **CRC16** |
| **Byte [0:1]** | **Byte [2]** | **Byte [3]** | **Byte [4:5]** | **Byte [6:n-2]** | **Byte [n-1:n]** |
| 0xa55a | 0-255 | 0-255 | 0-512 | Type dependent (size 0-512 bytes) | CRC value |

The fixed preamble bytes will enable each communication party to re-synchronize to the start point of a data frame. The data stream shall be first scanned for a first coming up of the preamble bytes to start a data frame processing.

After reading the preamble bytes, message type, transaction ID and payload length word (implying length is in valid range) with subsequent payload data bytes of given length together with the final CRC16 word shall be read.

A complete data frame shall be read within 500ms.

Timeout condition shall be applied if

* Preamble has been received (start time of data frame receipt) and
* 500ms timeout has been reached and
* Data frame parser is waiting for payload length, type, content or CRC16

Special timeouts may be applicable (esp. see SW update).

Invalid data frames are identified by

* Payload length out of range (>512 bytes) or
* Data frame bytes received including CRC16 bytes and
* CRC16 check is false

Detecting an invalid data frame or timeout will abort current data frame processing. Received data so far will be ignored. Data stream analysation will be restarted with the next received byte by looking for the next preamble.

All data frames passing the above conditions are considered as valid data frames / messages. However, its contents (message type, transaction ID, payload bytes) have to be processed with range checks and semantic checks.

### Transaction protocol

Each communication party can start a transaction at any time regarding timing constraints in order to prevent bus or receiver overload. A transaction is identified by a single message emitted by the sender and will be answered by the receiver with ACK/NACK. The emitter will use a unique (!) transaction ID in terms of a counter, which is incremented in ranges 0-255 for each new transaction.

ACK messages will mirror the emitter transaction ID to enable the emitter to close the current transaction and increment its transaction ID counter.

NACK messages will mirror the emitter transaction ID. The transaction is in pending state so the emitter shall reuse the transaction ID to repeat the message or close the transaction to start a new one.

HMI and Motor Control use their own transaction ID scopes when starting transactions.

One emitter shall keep only one transaction in progress at the same time.

As both communication parties can initiate transactions asynchronously, send and receive transactions may cross over.

### CRC16 calculation

CRC16 calculation will be applied according to following parameters

* CRC-CCITT (CRC-16)
* Polynom 0x1021
* Seed 0xFFFF
* Calculated over complete data including preamble
* Appended as CRC16 with LSB/MSB

Complete data frame bytes including preamble bytes have to be considered for calculation.

### Acknowledgement of messages

Invalid or timeout data frames will not be answered anyhow. The sender instead will detect a timeout condition upon missing acknowledgement and will resend the message.

All valid data frames will be answered by an acknowledge (ACK) or a not acknowledge (NACK) message. ACK or NACK messages itself shall never be acknowledged.

Each ACK or NACK message shall contain the transaction ID the message it responds to.

Reception of ACK for currently opened transactions will close the transaction within the originator. ACK / NACK for unknown transactions will silently be ignored.

ACK or NACK message type ID shall not be used for message requests.

### Alive monitoring

To prove each communication party’s alive status, each communication party sends an alive status message to each other each 1sec. Alive status and status message types are synonyms.

As soon as any alive status message is received, its internal alive status of the opposite party switches to true. If no valid alive status message is received within 5secs, the alive status falls back to false.

The opposite alive status shall be mirrored in the ALIVE bit of the own status message.

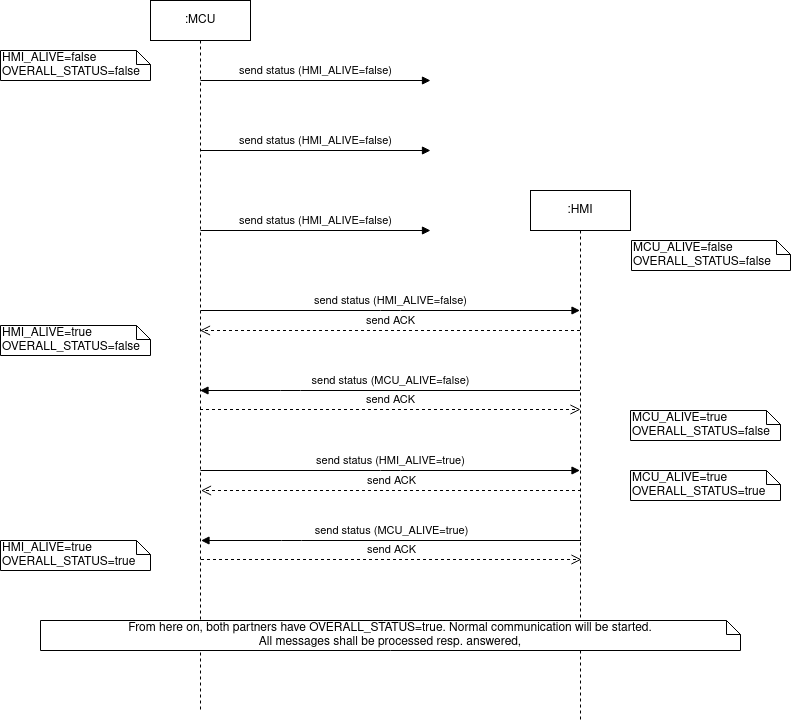
If no ACK / NACK is received by the originator upon sending alive status message, this message shall not be repeated.

Each communication party shall generate an overall communication alive status consisting of the own detected alive status of the opposite party and the content of ALIVE bit in its received status message.

No messages other than alive status shall be emitted by a sender if overall communication alive status is false.

If opposite alive status is false, all messages other than alive status shall be ignored and not answered by ACK/NACK.

Alive status of the communication counterpart simply relies on alive message exchange. Failed message transactions anyhow won't influence that state.



### Message retries

Data frames which are not answered by ACK (instead by NACK or by timeout) can (not mandatory) be repeated by the sender up to a maximum of 2 times. When reaching the maximum message repeat number without success the transaction is declared as failed.

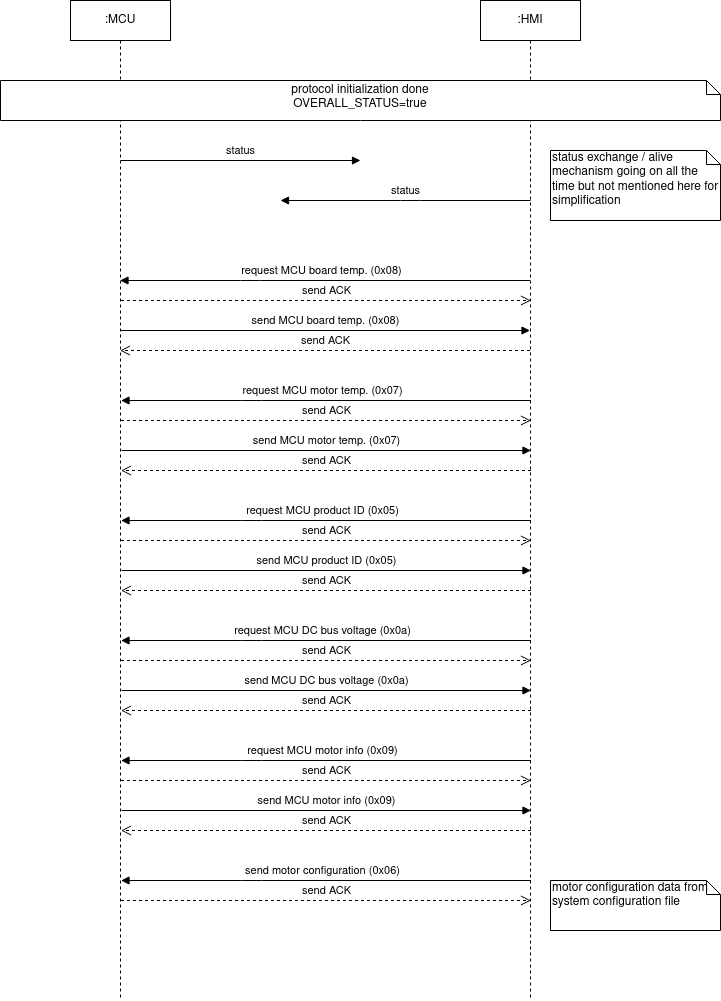
Transaction ID for repeated messages shall not be incremented.

Each communication party has to take proper actions to process failed transactions.

Cyclic messages esp. alive status need not to be repeated because its content will automatically be updated by the next time schedule.

### Post initialization process

The post initialization process describes the data exchange which follows up when communication becomes alive (overall status true). It contains a bunch of messages to synchronize HMI and MCU. If alive status will be lost and re-established again, also the post initialization process will be repeated.



## Message descriptions

### Message overview

Following message types are applicable:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Message Type** | **Implemented in receiver** | **Read / Write on MC** | **Message request applicable for** | **Description** |
| 0x00 | MC | R | HMI | MC Status |
| 0x00 | HMI | n/a | MC | HMI Status |
| 0x01 | HMI / MC | n/a | n/a | ACK |
| 0x02 | HMI / MC | n/a | n/a | NACK |
| 0x03 | HMI / MC | n/a | n/a | Message request |
| 0x04 | MC | W | n/a | Motor actuation |
| 0x05 | MC | R | HMI | Product identification data |
| 0x06 | HMI / MC | R / W | HMI | Motor configuration |
| 0x07 | HMI | R | HMI | Motor temperature NTC sensor |
| 0x08 | HMI | R | HMI | MC board temperature |
| 0x09 | HMI | R | HMI | Motor actuation info |
| 0x0a | HMI | R | HMI | Motor DC bus voltage |
| 0x0b | MC | W | n/a | Simulate alarm code |
| 0x0c | MC | W | n/a | SW update start |
| 0x0d | MC | W | n/a | SW update data |
| 0x0e | MC | W | n/a | SW update finish |
| 0x0f | MC | W | n/a | SW update reject |
| 0x10 | MC | W | n/a | MC reset |

HMI commonly supports only read access to its data.

### HMI Status

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** |
| 0x00 | 1 | Status |

**Description**:

Commits the current status of the HMI as soon as any of the status bits change. At least a status has to be sent cyclically each 1sec for alive monitoring.

Status byte bit coding is as follows:

|  |  |  |
| --- | --- | --- |
| **ESP Status Bit** | **Ident.** | **Description** |
| 0 | ALIVE | Alive status of MC (0=Not Alive, 1=Alive) |
| 1-7 |  | Reserved |

### MC Status

|  |  |  |  |
| --- | --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** | |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** | **Byte [7]** |
| 0x00 | 2 | System status | Fault status |

**Description**:

Commits the current status of the sender as soon as any of the status bits change. At least a status has to be sent cyclically each 1sec for alive monitoring.

System status byte bit coding is as follows:

|  |  |  |
| --- | --- | --- |
| **MC System Status Bit** | **Ident.** | **Description** |
| 0 | ALIVE | Alive status of ESP  (0=Not Alive, 1=Alive) |
| 1 | MOT\_RUN | Motor running  (0=Not Running, 1=Running) |
| 2 | SIMU | Simulation mode active  (0=Not Active, 1=Active) |
| 3 | HOPPER\_LOCK | Hopper switch dismounted -> motor is locked and cannot be started  (0=unlocked 1=locked) |
| 4 | FAN\_RUN | Fan running (only if function will be implemented to board)  (0=Not Running, 1=Running) |
| 5 | CHAMBER\_LOCK | Grinder chamber open -> motor is locked and cannot be started  (0=unlocked 1=locked) |
| 6-7 |  | Reserved |

Fault status byte bit coding is as follows (common bit meaning: 0=no fault, 1=fault):

|  |  |  |
| --- | --- | --- |
| **MC Fault Status Bit** | **Ident.** | **Description** |
| 0 | BRD\_OVERTEMP | Board overtemperature T > 100°C |
| 1 | DC\_OVERCURR | DC Overcurrent  (Range 6-10A) |
| 2 | AC\_OVERVOLT | AC power overvoltage (IN > 242VAC) |
| 3 | AC\_UNDERVOLT | AC power undervoltage (IN < 198VAC) |
| 4 | HALL\_SENSOR | Motor Hall sensor failure |
| 5 | CURR\_SENSOR | Current sensor failure |
| 6 | ROTOR\_LOCKED | Rotor locked (esp. mechanical blockage) |
| 7 |  | Reserved |

Status information from MC will be used by HMI to create user information on the display or for logging purposes. Also, software lock states will be derived from it to present a well reactive user interface. However, HMI reactions are not reliable in terms of functional safety, i.e. motor start operations should be expected and handled accordingly by MC although hopper status forbids such like.

### ACK

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **n/a** |
| 0x01 | 0 |  |

**Description**:

Acknowledge shall be used to confirm the positive reception and processing of any message other than ACK/NACK itself.

Data frame transaction ID shall contain the transaction ID of the message, which shall be acknowledged.

### NACK

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** |
| 0x02 | 1 | Reason for NACK |

**Description**:

Not-Acknowledge shall be used to signal the failed processing of any message other than ACK/NACK itself.

Illegal or timed-out data frames shall not be answered by a NACK message.

Data frame transaction ID shall contain the transaction ID of the message, which shall not be acknowledged.

Reason gives a more detailed hint, why the processing failed:

|  |  |
| --- | --- |
| **Reason Value** | **Description** |
| 0 | Unknown reason |
| 1 | Payload data out of range |
| 2 | Invalid CRC16 of message |
| 3 | Invalid message type (out of range) |
| 4 | Invalid payload length (according to specified message type) |
| 5 | Message type not supported |
| 6 | SW update not started |
| 7 | SW update wrong chunk sequence |
| 8 | SW update illegal binary size |
| 9 | SW update data chunk processing failed |
| 10 | SW update binary invalid |
| 11 | SW update not possible |
| 12 | Motor start rejected / not possible (see status) |
| 13 | Illegal motor configuration |

### Message request

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** |
| 0x03 | 1 | Request message type |

**Description**:

A certain message type is requested by the sender as read data.

Note that not all message types are supported to be sent by the receiver. If the receiver doesn’t support the message request, NACK (Reason 5, Not supported) will be returned.

If the message request is supported, receiver returns ACK followed by a new receiver transaction, containing the requested message.

### Motor actuation

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** |
| 0x04 | 1 | Actuation type |

**Description**:

Motor actuation command is used to start and stop the motor.

If motor cannot be started, NACK with Reason 12 will be returned.

|  |  |
| --- | --- |
| **Actuation type**  **(Byte 6)** | **Description** |
| 0 | Stop motor |
| 1 | Start motor |

### Product identification data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** | | | |
| **Byte [2]** | **Byte [4:5]** | **Byte [6:25]** | **Byte [26:45]** | **Byte [46:65]** | **Byte [66:85]** |
| 0x05 | 80 | Product ID | S/N | HW version | SW version |

**Description**:

Motor Control board tells HMI its product identification. This command will be sent by motor control upon request by HMI.

Each payload data element is a max. 20 characters ASCII string type (including terminating null byte):

* Product ID
* Device serial number
* Hardware version number
* Software version number

The string format is supplier dependent. It is used as-is to be displayed on HMI and to be logged in persistent data files. This data shall inform any service technician about system configuration and ensure correct spare part exchange.

Hardware and software version numbers shall enable any evaluating software to calculate upward or downward version compatibilities, i.e. a new version shall be an increment anyhow.

### Motor configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** | | | |
| **Byte [2]** | **Byte [4:5]** | **Byte [6:9]** | **Byte [10:13]** | **Byte [14:17]** | **Byte [18:21]** |
| 0x06 | 16 | Max motor speed [rpm] | Nominal motor speed [rpm] | Acceleration time [ms] | Deceleration time [ms] |

**Description**:

This data will be sent by HMI to configure motor control board. This message will be sent once whenever overall alive status is detected as true, assuming that previous false alive status was result of a MC reset. It will also be the case, that motor configuration will be sent in front of any grind to configure the motor speed for a better grind result.

Upon send by HMI the data is used to configure the MC. Upon requested by HMI this will be a read operation of the configuration stored in MC.

Maximum motor speed is the motor speed which shall never be overshoot to protect hardware and/or grinding result quality. This value limits the max. values of nopminal motor speed, acceleration and deceleration time.

Range: 500-1500 [rpm], default 1500 [rpm].

Nominal motor speed is the motor speed, which shall usually be achieved by the motor driver for the current run. The real motor speed might always slightly differ according to changes of the grinding resistance and the regulator algorithm.

Range: 0-max. motor speed [rpm]

Acceleration time is the time in which the motor accelerates from 0 to nominal motor speed [ms].

Range: 100-value of max. motor speed [ms]

Deceleration time is the time in which the motor decelerates from nominal motor speed to full stop [ms].

Range: 100-value of max. motor speed [ms]

If MC isn’t able to set the motor configuration (esp. one or several values are out of range), NACK with reason 13 will be returned and the payload shall not be used. MCU shall not cut off out-of-range values against the min/max boundaries by itself.

### Motor temperature NTC sensor

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** |
| 0x07 | 1 | Motor temperature [°C + 50°C] |

**Description**:

This message shall be sent by MC upon each change of motor temperature (full degree). Updates shall not happen more than once per second. To allow negative temperatures, the measured value will be transmitted with a positive offset of 50°C.

Valid temperature range is -50 to 204°C.

A value of 255 signals any invalid data condition (temperature out of range, sensor broken aso.).

Example: Measured temperature is 36°C -> send value will be 86 (36+50).

### MC board temperature

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** |
| 0x08 | 1 | MC board temperature [°C + 50°C] |

**Description**:

This message shall be sent by MC upon each change of the MC board temperature (full degree). Updates shall not happen more than once per second. To allow negative temperatures, the measured value will be transmitted with a positive offset of 50°C.

Valid temperature range is -50 to 204°C.

A value of 255 signals any invalid data condition (temperature out of range, sensor broken aso.).

Example: Measured temperature is 36°C -> send value will be 86 (36+50).

### Motor actuation info

|  |  |  |  |
| --- | --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** | |
| **Byte [2]** | **Byte [4:5]** | **Byte [6:7]** | Byte [8:9] |
| 0x09 | 4 | Motor current [mA] | Motor speed [rpm] |

**Description**:

MC tells the actual motor current and motor speed to HMI to enable evaluation of any motor problems and for logging purposes. A higher current can indicate any motor blockage, a lower current any running out of beans.

This message shall be sent automatically by MC as long as the motor is running with a cycle of 100ms. HMI can request that data at any time.

### Motor DC bus voltage

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6:7]** |
| 0x0a | 2 | DC bus voltage [V] |

**Description**:

This message shall be sent by MC upon each change of DC bus voltage (full Volt). Updates shall not happen more than once per second.

A value of 0xffff signals any invalid data condition (esp. voltage out of range, sensor defect aso.).

### Simulation mode

|  |  |  |  |
| --- | --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** | |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** | **Byte [7]** |
| 0x0b | 2 | Simulated system code | Simulated fault code |

**Description**:

This message will be sent by HMI to activate or deactivate the simulation mode. If any of the bits will be set, the appropriate bits in status message will return the requested status. Additionally, the simulation bit in status message will be set.

A value of null in both bytes will stop the simulation.

|  |  |  |
| --- | --- | --- |
| **Simulated system code bit** | **Ident.** | **Description** |
| 0-2 |  | Reserved |
| 3 | HOPPER\_LOCK | Hopper switch dismounted -> motor is locked and cannot be started (0=unlocked 1=locked) |
| 4 | FAN\_RUN | Fan running (only if function will be implemented to board)  (0=Not Running, 1=Running) |
| 5 | CHAMBER\_LOCK | Grinder chamber open -> motor is locked and cannot be started (0=unlocked 1=locked) |
| 6-7 |  | Reserved |

Simulated fault code byte bit coding is as follows (common bit meaning: 0=no fault, 1=fault):

|  |  |  |
| --- | --- | --- |
| **Simulated fault code bit** | **Ident.** | **Description** |
| 0 | BRD\_OVERTEMP | Board overtemperature T > 100°C |
| 1 | DC\_OVERCURR | DC Overcurrent  Out of range 6-10A |
| 2 | AC\_OVERVOLT | AC power overvoltage (IN > 242VAC) |
| 3 | AC\_UNDERVOLT | AC power undervoltage (IN < 198VAC) |
| 4 | HALL\_SENSOR | Hall sensor failure |
| 5 | CURR\_SENSOR | Current sensor failure |
| 6 | ROTOR\_LOCKED | Rotor locked |
| 7 |  | Reserved |

### Software update start

|  |  |  |  |
| --- | --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** | |
| **Byte [2]** | **Byte [4:5]** | **Byte [6:9]** | **Byte [10:13]** |
| 0x0c | 8 | Number of chunks | Total binary size |

**Description**:

This message shall be sent by HMI to indicate start of software update and to enable MC to prepare for it.

The total amount of transmitted data will be indicated with the total binary size. After this message a sequence of data chunks will be sent with software data messages, containing the binary data. The number of chunks depends on HMI defined chunk size (usually 128 bytes, but may vary). The last chunk will have an individual chunk size to meet the total binary size.

To complete software update a final software update finish message.

The update is stored in MC cache area and will be executed (copied to application area) upon next boot – this must not be done automatically. HMI will send a reset message after upload completion, but the due time depends on complete system update process to enable a consistent system boot up of all components. If any later component update failed, also the software update can be rejected by clearing the cache area.

Within software update phase no other messages will be sent except alive status. To start the software update, motor control board shall be in a quiescence phase.

The receiver shall NACK the message with reason 11 if software update isn’t possible (not in quiescence; update has already been started aso.).

The receiver shall NACK the message with reason 8 if receiver isn’t capable to cache the expected binary data size.

ACK will be replied when preparation of software update process has been completely done and receiver is able receive the first data chunk. Therefor this message might have an individual longer timeout (value to be defined in integration phase).

Timeout for message response shall be less than 950ms.

### Software update data

|  |  |  |  |
| --- | --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** | |
| **Byte [2]** | **Byte [4:5]** | **Byte [6:9]** | **Byte [10:n]** |
| 0x0d | 5-132 | Chunk number | Binary chunk data |

**Description**:

This message shall be sent by HMI to transmit a single binary data chunk in software update process.

The chunk number is a consecutive number starting with zero to indicate the correct sequence of data chunks without missing any package. The receiver has to validate the consecutive increment regarding that data chunks may be repeated in case of transmission errors.

Each chunk message has an individual transaction ID.

The nominal chunk data size will be 128 bytes. The final chunk can have less data to meet the overall binary data size.

The receiver shall NACK the message with reason 6 if software update has not been started previously.

The receiver shall NACK the message with reason 7 if the chunk number validation fails or overshoots the given maximum from start message.

The receiver shall NACK the message with reason 8 if the sum of received chunk data overshoots the given maximum binary data size from start message.

The receiver shall NACK the message with reason 9 if chunk data can’t be processed (esp. stored into cache area).

ACK will be replied when processing of data has been completely done and receiver is able receive the next data chunk. Therefor this message might have an individual longer timeout (value to be defined in integration phase).

Timeout for message response shall be less than 450ms.

### Software update finish

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **Byte [6]** |
| 0x0e | 1 | Transmit success |

**Description**:

This message shall be sent by HMI to signal finish of software update uploading.

Transmit success signals the success (0x1) to the receiver. In case of error (value is 0x0) the receiver has to take appropriate action to ensure, that received data won’t be processed anyhow (internally reject software data update).

This message may occur out of chunk order with transmit success 0x0 (error) if the sender detects any transmit problems during data upload.

The receiver shall NACK the message with reason 6 if software update has not been started previously.

The receiver shall NACK the message with reason 7 if the chunk number of the previously received data chunks doesn’t meet the given maximum from start message.

The receiver shall NACK the message with reason 8 if the sum of received chunk data doesn’t meet the given maximum binary data size from start message.

The receiver shall NACK the message with reason 10 if the final validation of the received binary data failed (esp. internal CRC check of the cached data failed).

ACK will be replied when validation of data has been completely done and receiver is able to activate the binary upload with the next reboot. Therefor this message might have an individual longer timeout (value to be defined in integration phase).

Upon receipt of this finish message the receiver shall not perform an automatic reboot.

Timeout for message response shall be less than 950ms.

### Software update reject

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **n/a** |
| 0x0f | 0 |  |

**Description**:

This message shall be sent by HMI to reject a previously uploaded software update.

Software update of EKX system is split among several components which are updated step by step. As long as not all of these components are updated, no single update shall be considered as active to prevent an inconsistent system state. Therefore, if one later component update will fail, it must be ensured to redraw the already performed upload of binary data.

This message shall enable the MC to clear the received data or disable it to prevent any usage on next reboot.

ACK to this message is expected after finish of this function. As esp. clearing internal data cache may take a while, ACK of this message will have an individual longer timeout (value to be defined in integration phase).

### MC reset

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Payload Length** | **Payload** |
| **Byte [2]** | **Byte [4:5]** | **n/a** |
| 0x10 | 0 |  |

**Description**:

HMI requests a (soft) reset on motor control board.

This might be requested by HMI in certain situations, i.e. after successful sending of SW update data, communication problems aso.

MC shall answer the message with ACK before going into reset.

Resetting the MCU doesn’t affect the power supply for the ESP and HMI subsystem.

# Glossary

|  |  |
| --- | --- |
| **Term** | **Description** |
| ACK | Acknowledge |
| ASCII | American Standard Code for Information Interchange |
| CRC | Cyclic redundancy check |
| DC | Direct current |
| EKX | Current Hemro coffee grinder project |
| ESP | MCU ESP32 based board |
| HEM | Hemro resp Hemro Group |
| HMI | Human Machine Interface (the Linux based display system on EKX) |
| HW | Hardware |
| ID | Identifier / Identification |
| IPM | I.P.M. Technologies s.r.l. |
| MC | Motor control / motor control board |
| MCU | Micro controller unit |
| NACK | Not acknowledge |
| RS232 | Serial line communication standard |
| SW | Software |
| UX | Ultratronik |