Motor Control Protocol

PowerPC PMSM single/dual FOC – Serial Communication

Contents

List of tables

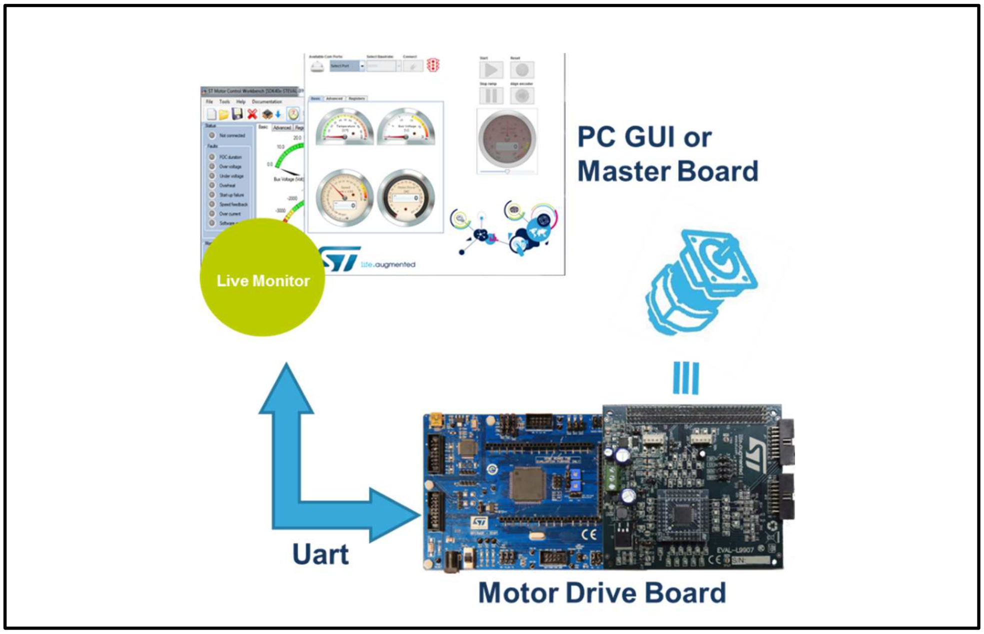
List of figures

1 Serial communication class overview

Applications on the market, that require an electrical motor to be driven, usually have the electronics split in two parts: application board and motor drive board.

To drive the system correctly, the application board requires a method to send a command to the motor drive board and get a feedback. This is usually performed using a serial communication. See *Figure 1: "Serial communication in motor control application"*.

Figure 1: Serial communication in motor control application



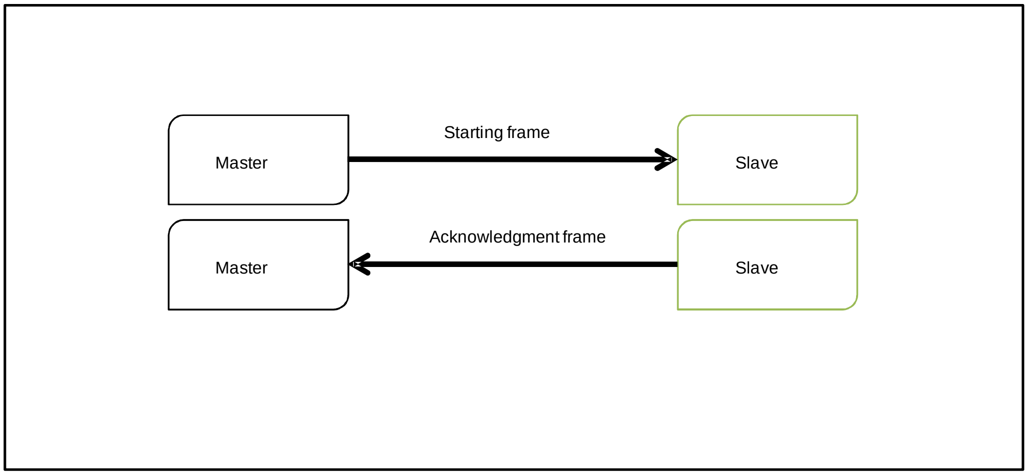
To target this kind of application, a dedicated serial communication protocol has been developed for real-time data exchange. The aim of this protocol is to implement the feature requested by motor control related applications. The implemented protocol is called motor control protocol (MCP).

MCP makes it possible to send commands such as start/stop motor and set the target speed to the PowerPC FOC motor control firmware, and also to tune in real-time relevant control variables such as PI coefficients. It is also possible to monitor relevant quantities, such as the speed of the motor or the bus voltage present in the board related to the controlled system.

The implemented communication protocol is based on a master-slave architecture in which the motor control firmware, running on a PowerPC microcontroller, is the slave.

The master, usually a PC or another microcontroller present on a master board, can start the communication at any time by sending the first communication frame to the slave. The slave answers this frame with the acknowledge frame. See *Figure 2: "Master-slave communication architecture"*.

Figure 2: Master-slave communication architecture



The implemented MCP is based on the physical layer that uses the USART communication.

A generic starting frame (*Table 1: "Generic starting frame"*) is composed of:

- **FRAME_START:** this byte defines the type of starting frame. The least significant 5 bits indicate the frame identifier. The most significant 3 bits indicate the motor selection. See *Table 2: "FRAME_START byte"*.
- **PAYLOAD_LENGTH:** the total number of bytes that compose the frame payload.
- **PAYLOAD_ID:** first byte of the payload that contains the identifier of payload. Not necessary if not required by this type of frame.
- **PAYLOAD[X]:** the remaining payload content. Not necessary if not required by this type of frame.
- **CRC:** byte used for cyclic redundancy check.

The minimum frame length is 3: **FRAME_START, PAYLOAD_LENGTH = 0, CRC**.

The CRC byte is computed as follows:

$$Total = (unsigned16bit)(FRAME\_START + PAYLOAD\_LENGTH + PAYLOAD\_ID + \sum_{i=0}^{n} PAYLOAD[i])$$

$$CRC = (unsigned8bit)\big(HighByte(Total) + LowByte(Total)\big)$$

Table 1: Generic starting frame

FRAME_START	PAYLOAD_LENGTH	PAYLOAD_ID	PAYLOAD[0]	...	PAYLOAD[n]	CRC

Table 3: "Starting frame codes" shows the list of possible starting frames.

Table 2: FRAME_START byte

FRAME_START	Motor bits			FRAME_ID				
	7	6	5	4	3	2	1	0

Table 3: FRAME_START Motor bits

Motor bits	Description
000	The command is applied to the last motor selected
001	The command is applied to motor 1; motor 1 is selected from now on
010	The command is applied to motor 2; motor 2 is selected from now on (this can be accepted only in dual drive, currently it's not supported in the MC Library)

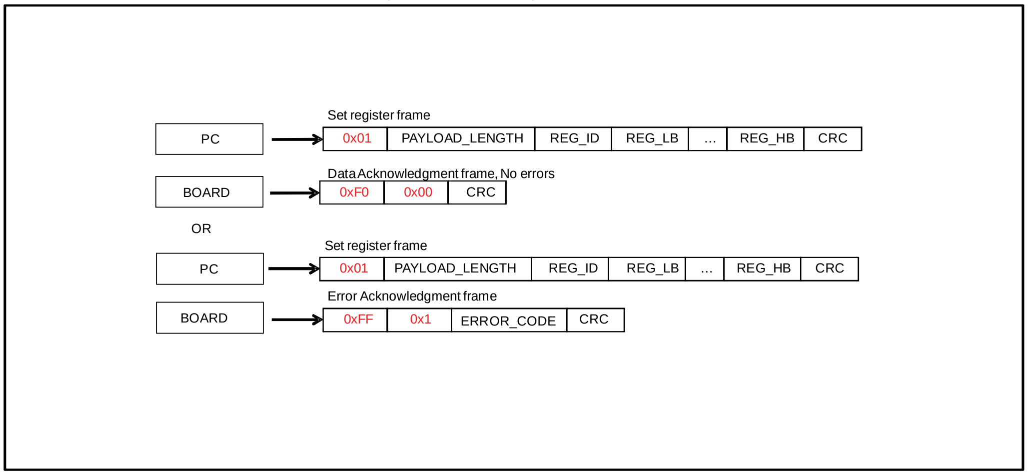
Table 4: Starting frame codes

FRAME_ID	Description
0x01	**Set register frame.** It is used to write a value into a relevant motor control variable. See *Section 1.1: "Set register frame".*
0x02	**Get register frame.** It is used to read a value from a relevant motor control variable. See *Section 1.2: "Get register frame".*
0x03	**Execute command frame.** It is used to send a command to the motor control object. See *Section 1.3: "Execute command frame".*
0x06	**Get board info.** It is used to retrieve information about the firmware currently running on the microcontroller. **Payload length is 0.**
0x07	**Exec ramp.** It is used to execute a speed ramp. See *Section 1.4: "Execute ramp frame"*.
0x08	**Get revup data.** It is used to retrieve the revup parameters. See *Section 1.5: "Get revup data frame"*.
0x09	**Set revup data.** It is used to set the revup parameters. See *Section 1.6: "Set revup data frame"*.
0x0A	**Set current references.** It is used to set the current reference. See *Section 1.7: "Set current references frame"*

1.1 Set register frame

The set register frame (*Figure 3: "Set register frame"*) is sent by the master to write a value into a relevant motor control variable.

Figure 3: Set register frame



The payload length depends on REG_ID (See *Table 5: "List of error codes"*).

REG_ID indicates the register to be updated.

The remaining payload contains the value to be updated, starting from the least significant byte to the most significant byte.

The Acknowledgment frame can be of two types:

- Data Acknowledgment frame, if the operation has been successfully completed. The payload of this Data Acknowledgment frame is zero.
- Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1. The list of error codes is shown in *Table 5: "List of error codes"*.

Table 5: List of error codes

Error code	Description
0x01	BAD Frame ID. The Frame ID has not been recognized by the firmware.
0x02	Write on read-only. The master wants to write on a read-only register.
0x03	Read not allowed. The value cannot be read.
0x04	Bad target drive. The target motor is not supported by the firmware.
0x05	Out of range. The value used in the frame is outside the range expected by the firmware.
0x07	Bad command ID. The command ID has not been recognized.
0x08	Overrun error. The frame has not been received correctly because the transmission speed is too fast.
0x09	Timeout error. The frame has not been received correctly and a timeout occurs. This kind of error usually occurs when the frame is not correct or is not correctly recognized by the firmware.

Error code	Description
0x0A	Bad CRC. The computed CRC is not equal to the received CRC byte.
0x0B	Bad target drive. The target motor is not supported by the firmware.

Table 6: "List of relevant motor control registers" indicates the following for each of the relevant motor control registers:

- Type (u8 8-bit unsigned, u16 16-bit unsigned, u32 32-bit unsigned, s16 16-bit signed, s32 32-bit signed)
- Payload length in Set register frame
- allowed access (R read, W write)
- Reg Id

Table 6: List of relevant motor control registers

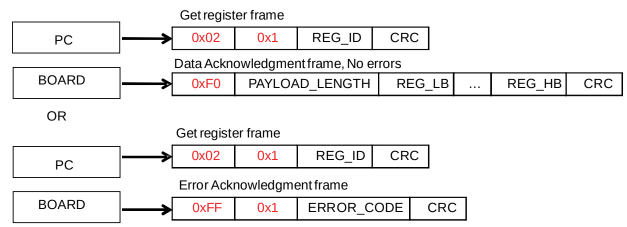
Register name	Type	Payload length	Access	REG_ID
Target motor	u8	2	RW	0x00
Flags	u32	5	R	0x01
Status	u8	2	R	0x02
Control mode	u8	2	RW	0x03
Speed reference	s32	5	R	0x04
Speed KP	u16	3	RW	0x05
Speed KI	u16	3	RW	0x06
Speed KD	u16	3	RW	0x07
Torque reference (I_q)	s16	3	RW	0x08
Torque KP	u16	3	RW	0x09
Torque KI	u16	3	RW	0x0A
Torque KD	u16	3	RW	0x0B
Flux reference (I_d)	s16	3	RW	0x0C
Flux KP	u16	3	RW	0x1D
Flux KI	u16	3	RW	0x1E
Flux KD	u16	3	RW	0x1F
Observer C1	s16	3	RW	0x10
Observer C2	s16	3	RW	0x11
Cordic Observer C1	s16	3	RW	0x12
Cordic Observer C2	s16	3	RW	0x13
PLL KI	u16	3	RW	0x14
PLL KP	u16	3	RW	0x15
Flux weakening KP	u16	3	RW	0x16
Flux weakening KI	u16	3	RW	0x17

Register name	Type	Payload length	Access	REG_ID
Flux weakening BUS Voltage allowed percentage reference	u16	3	RW	0x18
Bus Voltage	u16	3	R	0x19
Heatsink temperature	u16	3	R	0x1A
Motor power	u16	3	R	0x1B
DAC Out 1	u8	2	RW	0x1C
DAC Out 2	u8	2	RW	0x1D
Speed measured	s32	5	R	0x1E
Torque measured (I_q)	s16	3	R	0x1F
Flux measured (I_d)	s16	3	R	0x20
Flux weakening BUS Voltage allowed percentage measured	u16	3	R	0x21
Revup stage numbers	u8	2	R	0x22
Maximum application speed	u32	5	R	0x3F
Minimum application speed	u32	5	R	0x40
Iq reference in speed mode	s16	3	W	0x41
Expected BEMF level (PLL)	s16	3	R	0x42
Observed BEMF level (PLL)	s16	3	R	0x43
Expected BEMF level (CORDIC)	s16	3	R	0x44
Observed BEMF level (CORDIC)	s16	3	R	0x45
Feedforward (1Q)	s32	5	RW	0x46
Feedforward (1D)	s32	5	RW	0x47
Feedforward (2)	s32	5	RW	0x48
Feedforward (VQ)	s16	3	R	0x49
Feedforward (VD)	s16	3	R	0x4A
Feedforward (VQ PI out)	s16	3	R	0x4B
Feedforward (VD PI out)	s16	3	R	0x4C
Ramp final speed	s32	5	RW	0x5B
Ramp duration	u16	3	RW	0x5C

1.2 Get register frame

The get register frame (*Figure 4: "Get register frame"*) is sent by the master to read a value from a relevant motor control variable.

Figure 4: Get register frame



Payload length is always 1.

REG_ID indicates the register to be queried (See *Table 6: "List of relevant motor control registers "*).

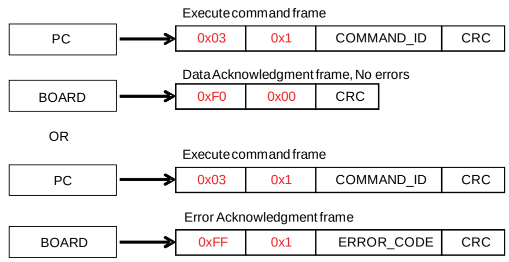
The Acknowledgment frame can be of two types:

- Data Acknowledgment frame, if the operation has been successfully completed. In this case, the returned value is embedded in the Data Acknowledgment frame. The size of the payload depends on **REG_ID** and is equal to the Payload length present in *Table 6: "List of relevant motor control registers"* minus 1. The value is returned starting from the least significant byte to the most significant byte.
- Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1. The list of error codes is shown in *Table 5: "List of error codes"*.

1.3 Execute command frame

The execute command frame (*Figure 5: "Execute command frame"*) is sent by the master to the motor control firmware to request the execution of a specific command.

Figure 5: Execute command frame



Payload length is always 1.

COMMAND_ID indicates the requested command (See *Table 7: "List of commands"*).

The Acknowledgment frame can be of two types:

- Data Acknowledgment frame, if the operation has been successfully completed. In this case, the returned value embedded in the Data Acknowledgment frame is an echo of the same Command Id. The size of payload is always 1.
- Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1. The list of error codes is shown in *Table 5: "List of error codes"*.

Table 7: "List of commands" indicates the list of commands:

Table 7: List of commands

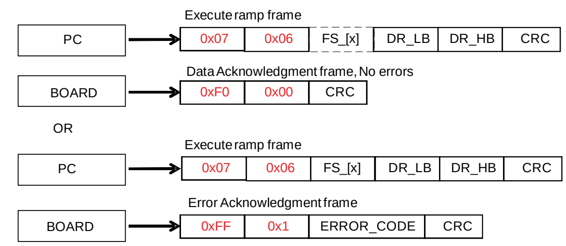
Command	Command ID	Description
Start Motor	**0x01**	Indicates the user request to start the motor regardless the state of the motor.
Stop Motor	**0x02**	Indicates the user request to stop the motor regardless the state of the motor.
Stop Ramp	**0x03**	Indicates the user request to stop the execution of the speed ramp that is currently executed
Start/Stop	**0x06**	Indicates the user request to start the motor if the motor is still, or to stop the motor if it runs.
Fault Ack	**0x07**	Communicates the user acknowledges of the occurred fault conditions.
Encoder Align	**0x08**	Indicates the user request to perform the encoder alignment procedure.

1.4 Execute ramp frame

The execute ramp frame (*Figure 6: "Execute ramp frame"*) is sent by the master to the motor control firmware, to request the execution of a speed ramp.

A speed ramp always starts from the current motor speed, and is defined by a duration and a final speed. See *Figure 7: "Speed ramp"*.

Figure 6: Execute ramp frame



Payload length is always 6.

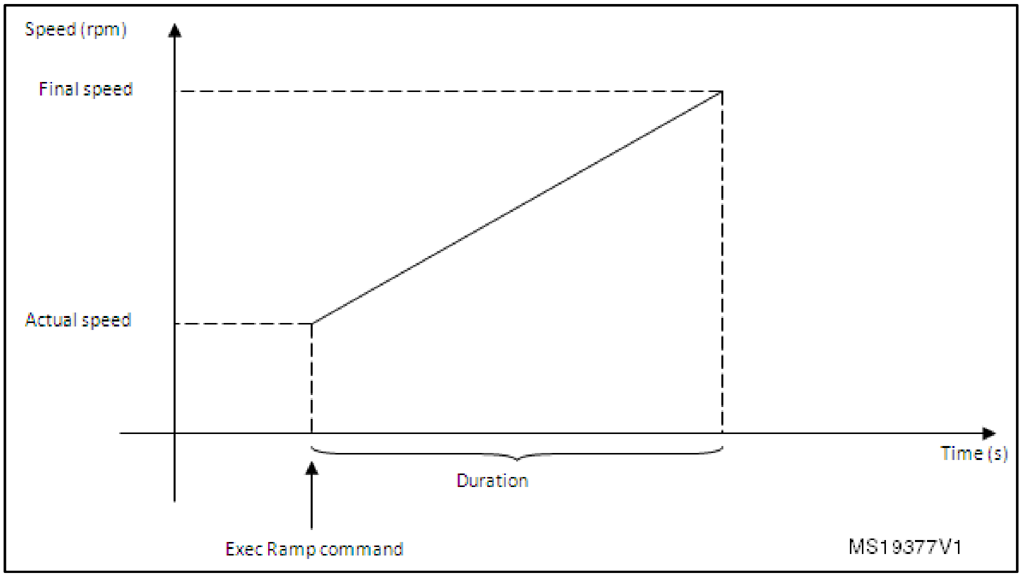
The four bytes FS[x] represent the final speed expressed in rpm least significant byte and most significant byte.

DR_LB and DR_HB represent the duration expressed in milliseconds, respectively least significant byte and most significant byte.

The Acknowledgment frame can be of two types:

- Data Acknowledgment frame, if the operation has been successfully completed. The payload of this Data Acknowledgment frame will be zero.
- Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1. The list of error codes is shown in *Table 5: "List of error codes"*.

Figure 7: Speed ramp

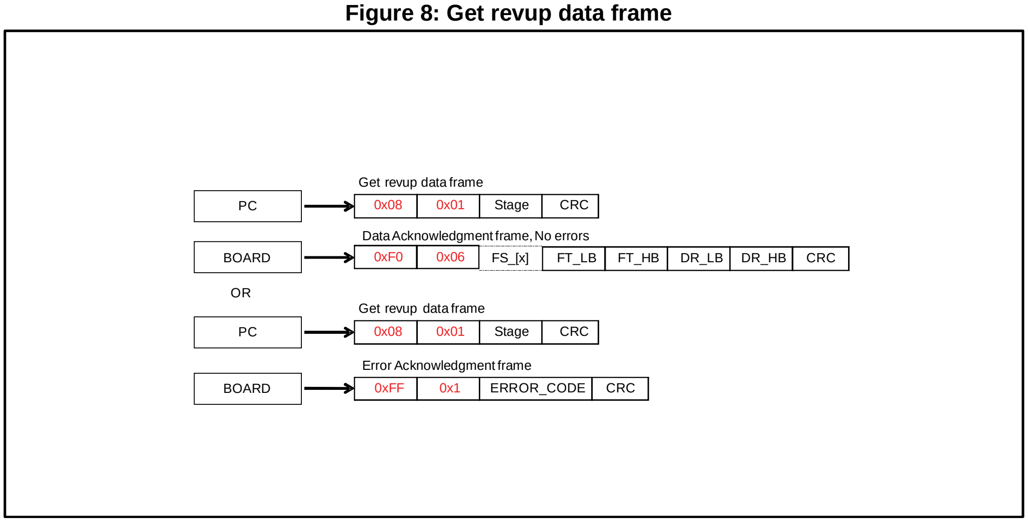


1.5 Get revup data frame

The get revup data frame (*Figure 8: "Get revup data frame"*) is sent by the master to retrieve the current revup parameters.

Revup sequence is a set of commands performed by the motor control firmware to drive the motor from zero speed up to run condition. It is mandatory for a sensor less configuration. The sequence is split into several stages; a duration, final speed and final torque (actually I_q reference) can be set up for each stage. See *Figure 9: "Revup sequence"*.

Figure 8: Get revup data frame



The master indicates the requested stage parameter sending the stage number in the starting frame payload.

Payload length is always 1.

The Acknowledgment frame can be of two types:

- Data Acknowledgment frame, if the operation has been successfully completed. In this case, the returned values are embedded in the Data Acknowledgment frame. The payload size of this Data Acknowledgment frame is always 8.

The four bytes FS[x] represent the final speed of the selected stage expressed in rpm, from the least significant byte to the most significant byte.

FT_LB and FT_HB represent the final torque of the selected stage expressed in digit, respectively the least significant byte and the most significant byte.

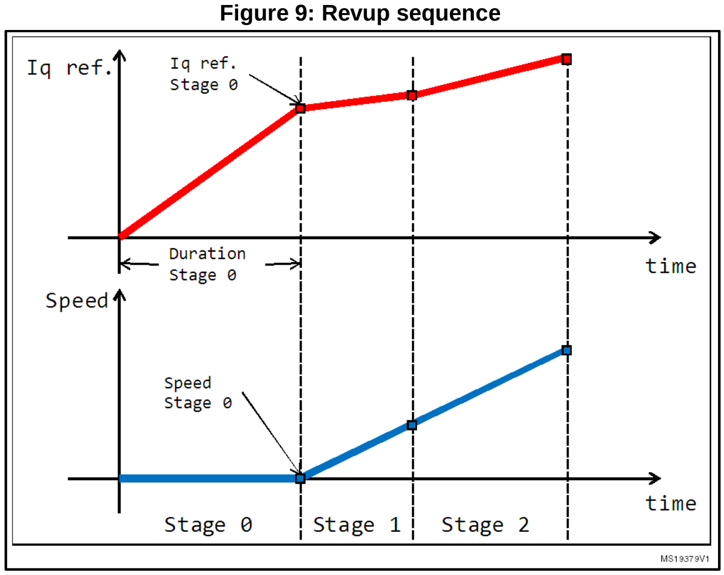
To convert current expressed in Amps to current expressed in digit, use the formula:

Current(digit)=[Current(Amp)×65536×R_Shunt×A_OP]/V_(DD Micro)

DR_LB and DR_HB represent the duration of the selected stage expressed in milliseconds, respectively the least significant byte and the most significant byte.

- Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1. The list of error codes is shown in *Table 5: "List of error codes"*.

Figure 9: Revup sequence

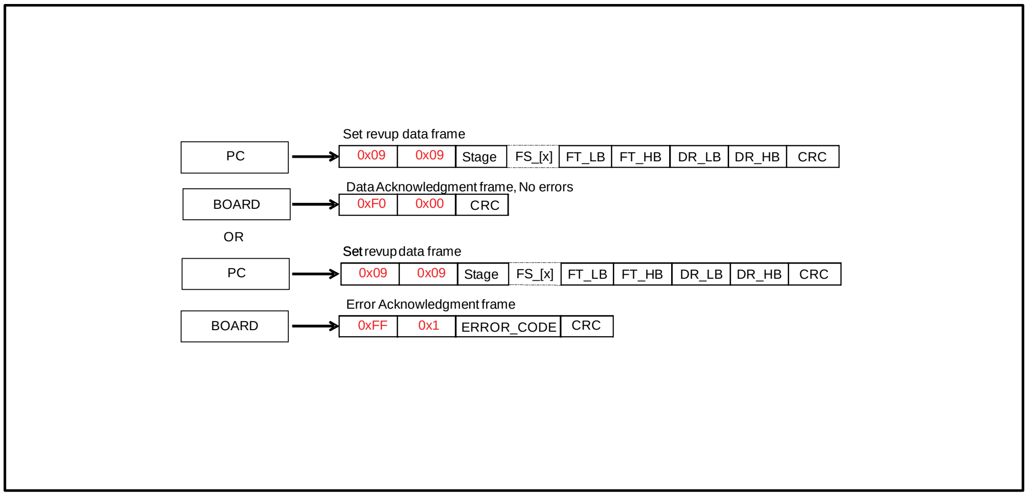


1.6 Set revup data frame

The set revup data frame (*Figure 10: "Set revup data frame"*) is sent by the master to modify the revup parameters.

Revup sequence is a set of commands performed by the motor control firmware to drive the motor from zero speed up to run condition. It is mandatory for a sensor less configuration. The sequence is split into several stages. For each stage, a duration, final speed and final torque (actually I_q reference) can be set up. See *Figure 9: "Revup sequence"*.

Figure 10: Set revup data frame



The Master sends the requested stage parameter.

The payload length is always 9.

Stage is the revup stage that will be modified.

The four bytes FS[x] is the requested new final speed of the selected stage expressed in rpm, from the least significant byte to the most significant byte.

FT_LB and FT_HB are the requested new final torque of the selected stage expressed in digit, respectively the least significant byte and the most significant byte.

To convert current expressed in Amps to current expressed in digit, it is possible to use the formula:

Current(digit) = [Current(Amp) * 65536 * Rshunt * Aop] / Vdd micro.

DR_LB and DR_HB is the requested new duration of the selected stage expressed in milliseconds, respectively the least significant byte and the most significant byte.

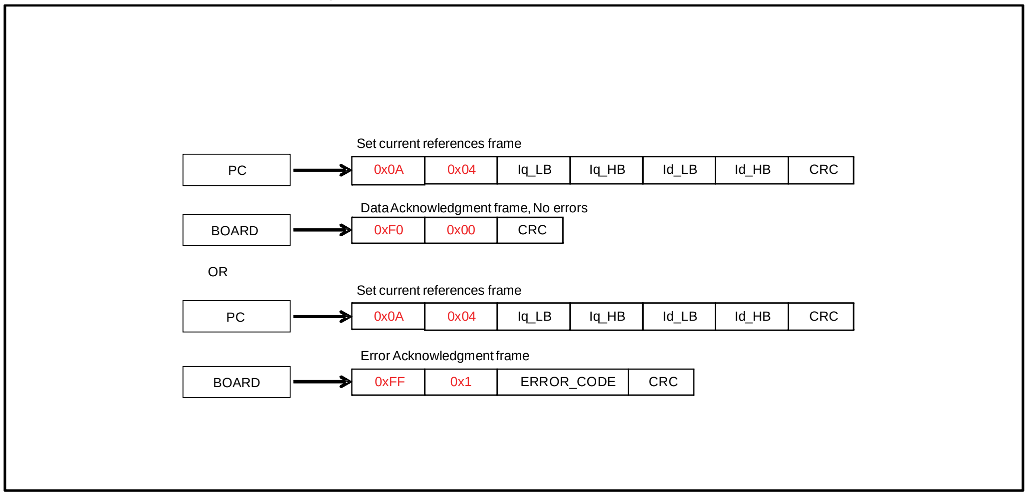
The Acknowledgment frame can be of two types:

- Data Acknowledgment frame, if the operation has been successfully completed. The payload of this Data Acknowledgment frame will be zero.
- Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1. The list of error codes is shown in *Table 5: "List of error codes"*.

1.7 Set current references frame

The set current references frame (*Figure 11: "Set current reference frame"*) is sent by the Master to modify the current references I_q, I_d.

Figure 11: Set current reference frame



The Master sends the requested current references.

The payload length is always 4.

Iq_LB and Iq_HB are the requested new I_q references expressed in digit, respectively the least significant byte and the most significant byte.

Id_LB and Id_HB are the requested new I_d reference expressed in digit, respectively the least significant byte and the most significant byte.

To convert current expressed in Amps to current expressed in digit, it is possible to use the formula:

Current(digit)=[Current(Amp)×65536×R_Shunt×A_OP]/Vdd micro)

The Acknowledgment frame can be of two types:

- Data Acknowledgment frame, if the operation has been successfully completed. The payload of this Data Acknowledgment frame will be zero.
- Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1. The list of error codes is shown in *Table 5: "List of error codes"*.

2 Document conventions

Table 8: List of abbreviations

Abbreviation	Definition
AC	Alternate Current
API	Application Programming Interface
B-EMF	Back Electromotive Force
CC-RAM	Core Coupled Memory Random Access Memory
CORDIC	COordinate Rotation DIgital Computer
DAC	Digital to Analog Converter
DC	Direct Current
FOC	Field Oriented Control
GUI	Graphical User Interface
I-PMSM	Internal Permanent Magnet Synchronous Motor
IC	Integrated Circuit
ICS	Isolated Current Sensor
IDE	Integrated Development Environment
MC	Motor Control
MCI	Motor Control Interface
MCP	Motor Control Protocol
MCT	Motor Control Tuning
MTPA	Maximum Torque Per Ampere
PGA	Programmable Gain Amplifier
PID controller	Proportional-Integral-Derivative controller
PLL	Phase-Locked Loop
PMSM	Permanent Magnet Synchronous Motor
PPC	PowerPC
SDK	Software Development Kit
SM-PMSM	Surface Mounted Permanent Magnet Synchronous Motor
SV PWM	Space Vector Pulse-Width Modulation
UI	User Interface