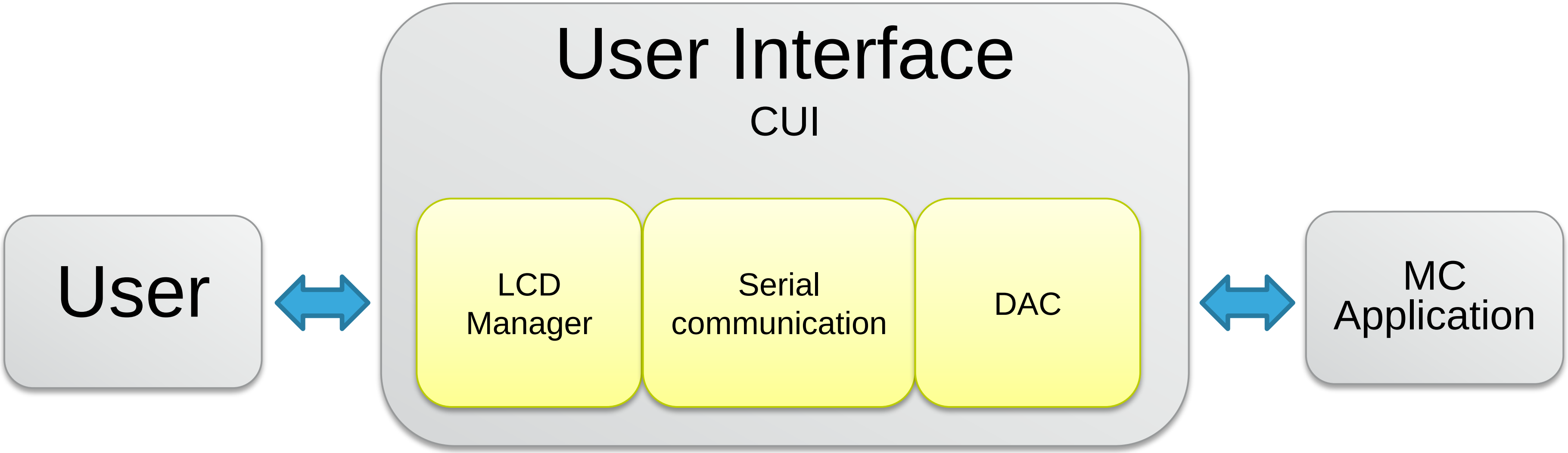
Agenda

- 2nd day – Morning
  - User interface
    - Serial Protocal
    - DAC
    - LCD
    - DAC customization
  - STM32 MC Workbench presentation in detail
- 2nd day – Afternoon
  - Quick Start
    - Config the firmware lib with Workbench
    - LCD User Interface
    - IAR IDE – MC Workspace
  - Practical hints in motor tuning:
    - Draw an arbitrary sensorless start-up waveform
    - Open loop feature
    - Faults generation
    - Motor start-up
  - Demo, Q&A

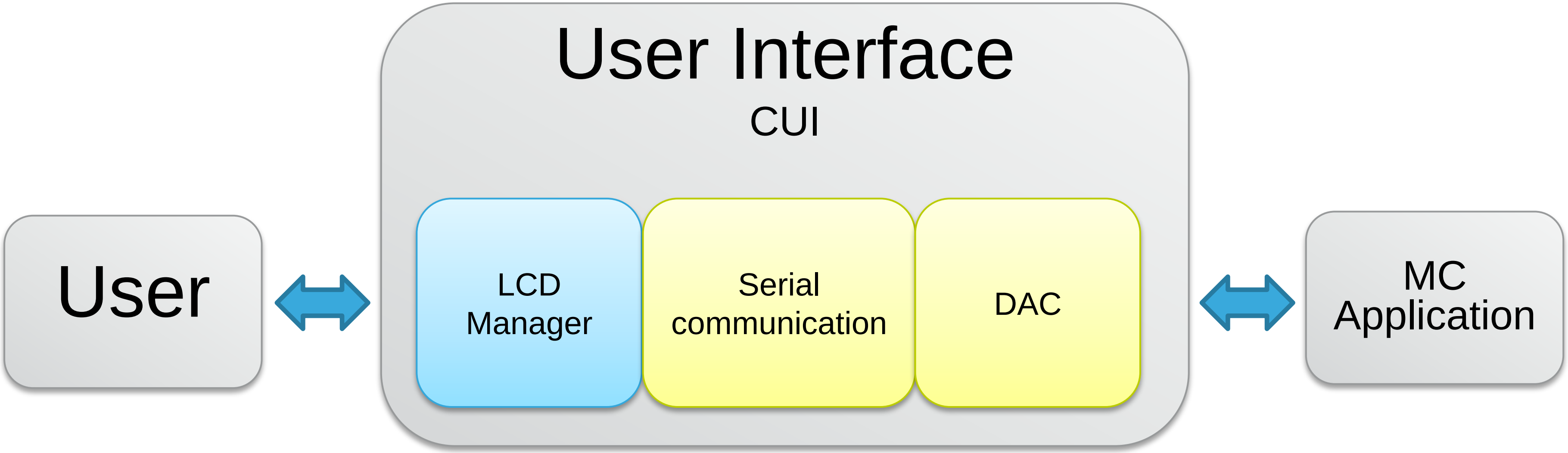
User interface architecture

- The user interface class (CUI) manages the interaction between the user and the motor control library (MC Library) through the use of the motor control application (MC Application).



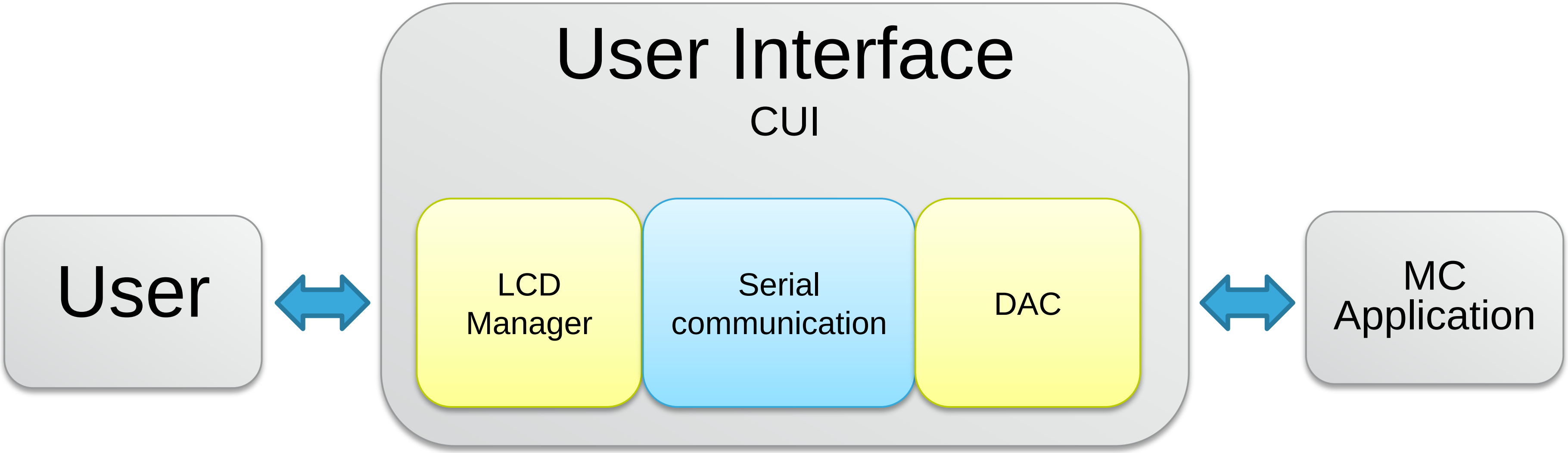
User interface architecture

- LCD Manager Class (CLCD_UI) is used to interact with the LCD color display. It has been implemented over the graphical library STMFC written in C++ language.



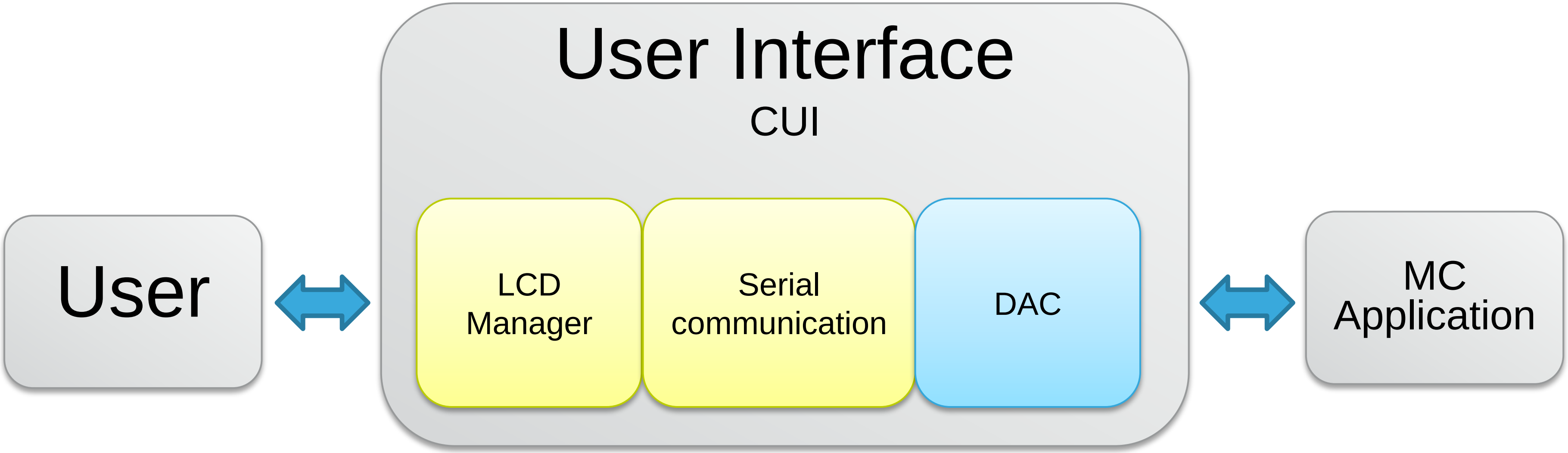
User interface architecture

- Motor control protocol (CMCP_UI) is used to manage serial communications.

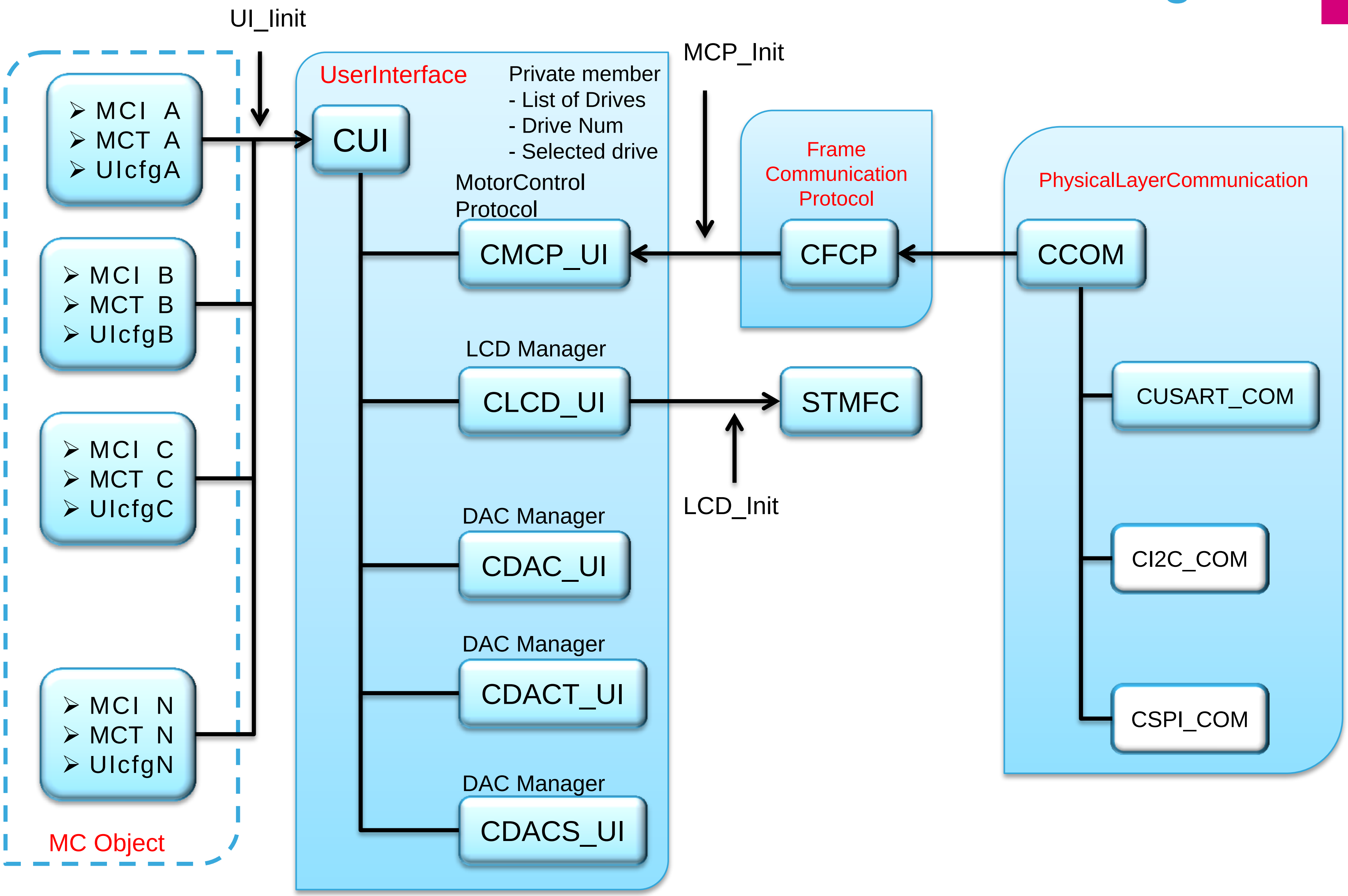


User interface architecture

- DAC manager (CDAC_UI) is used to manage the DAC outputs.

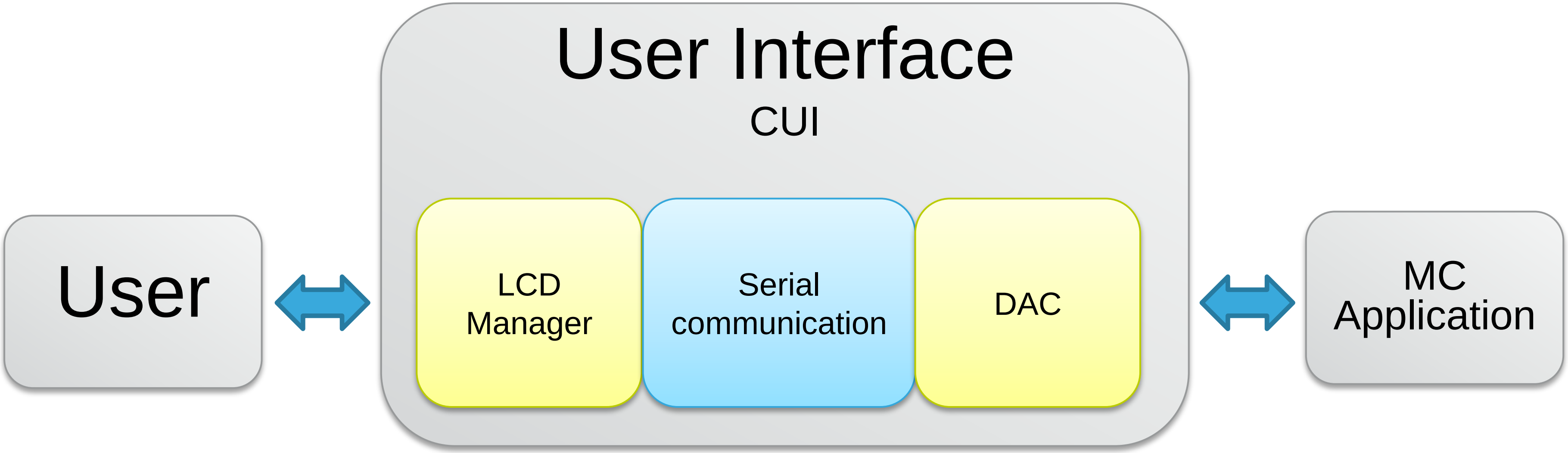


User interface block diagram



User interface architecture

- Motor control protocol (CMCP_UI) is used to manage serial communications.



User

User Interface

CUI

LCD Manager

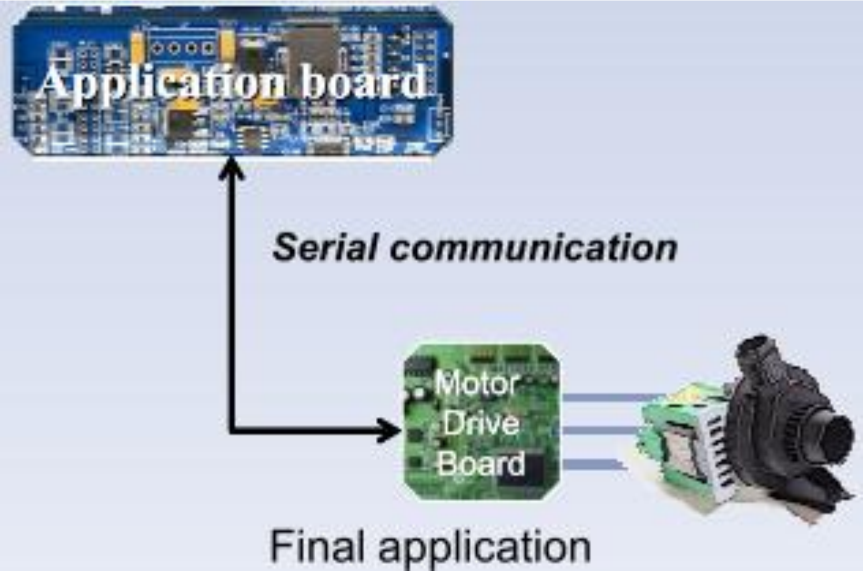
Serial communication

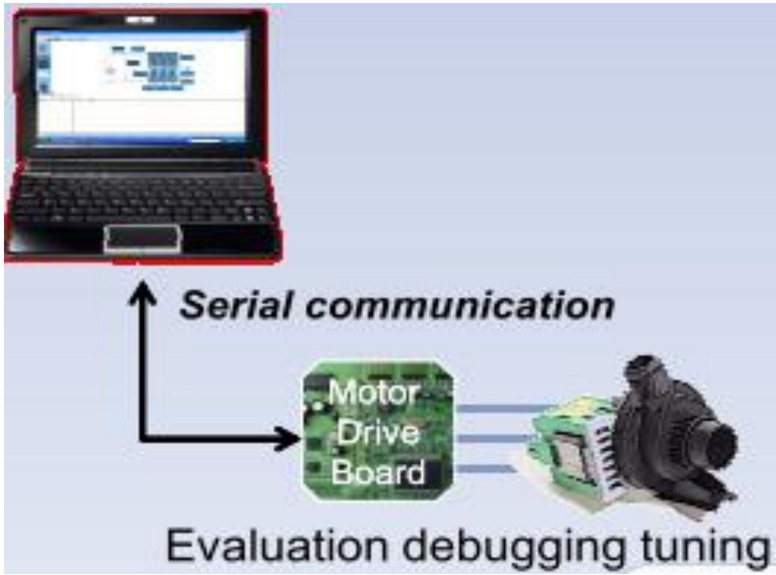
DAC

MC Application

Serial communication

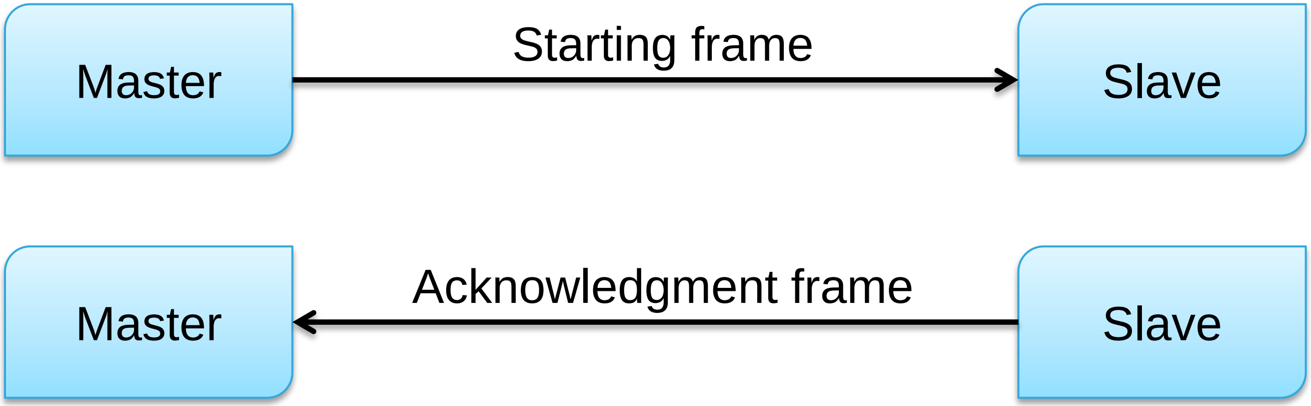
- Usually in the market, the applications that require an electrical motor to be driven should have the electronics split in two parts: “Application board” and “Motor Drive board”.
- To proper drive the system the “Application board” require a method to send command to the “Motor Drive board” and get feedback from it. Usually this is performed using a serial communication.
- Serial communication can be useful also during the evaluation, debugging and tuning of motor control HW/SW.





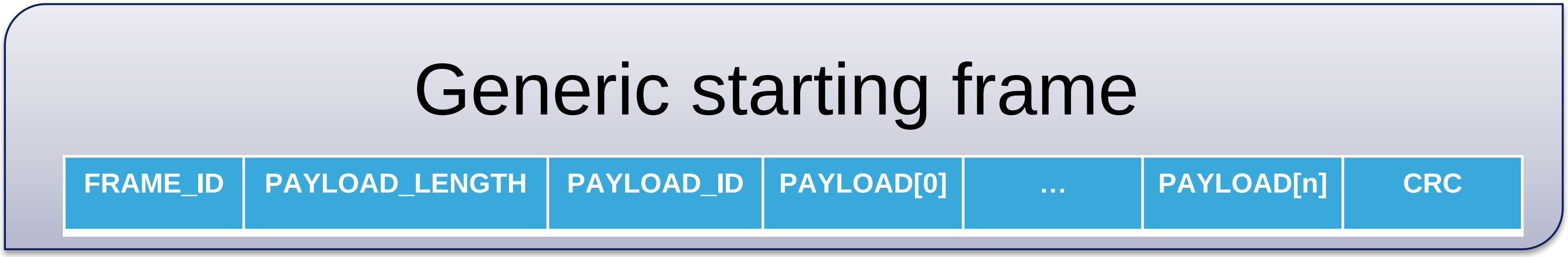
Master slave architecture

- The implemented communication protocol is based on a “master-slave” architecture in which the motor control firmware running on STM32 microcontroller is the “slave”.
- At any time the “master”, usually a PC or another microcontroller present on a “master” board, can start the communication sending to the “slave” the first communication frame. The “slave” answer to this frame with the acknowledge frame.



Starting frame

- A generic starting frame is composed by:
  - Frame_ID, it is a byte that define which is the “type” of the starting frame
  - Payload_Length, it is the total number of byte that actually compose the frame payload,
  - Payload_ID, it is the first byte of payload and contains the identifier of payload, it can be missed if not required by this “type” of frame,
  - Payload[x], it is the remaining payload content, it can be missed if not required by this “type” of frame,
  - CRC, it is a byte used for cyclic redundancy check.



Generic starting frame

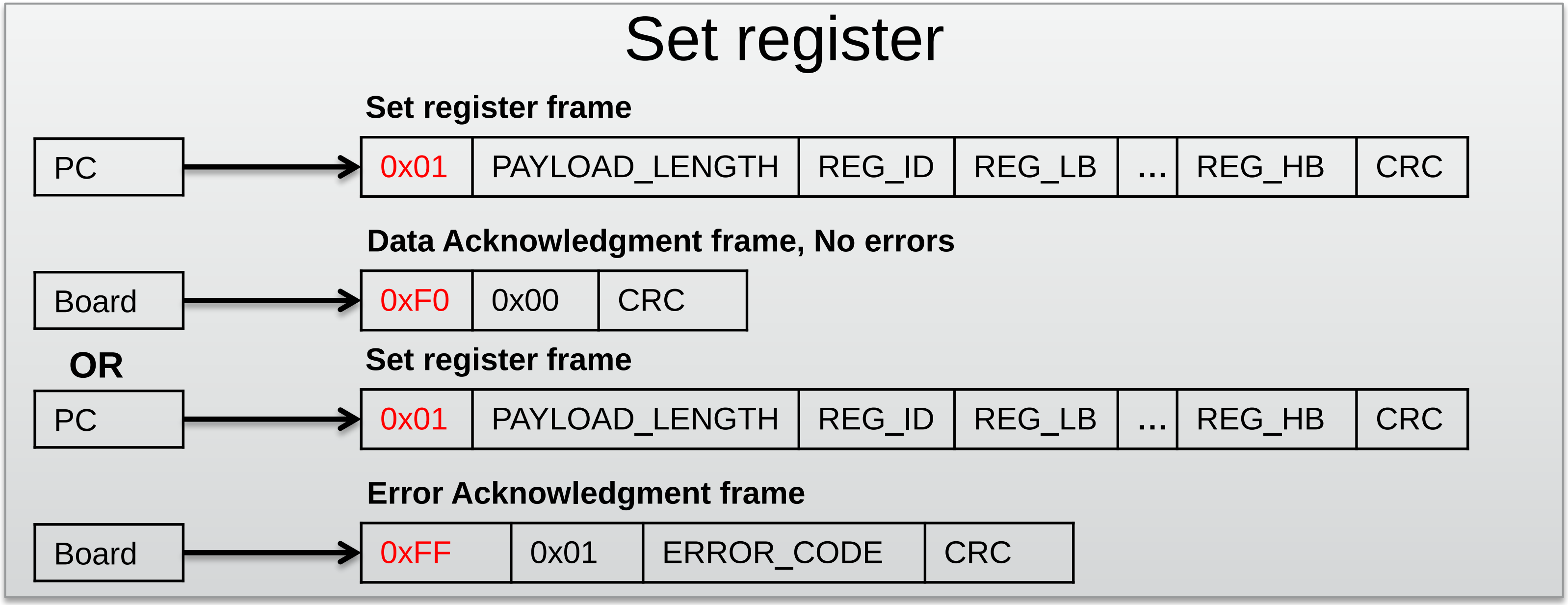
FRAME_ID	PAYLOAD_LENGTH	PAYLOAD_ID	PAYLOAD[0]	…	PAYLOAD[n]	CRC

Frame IDs

Frame_ID	Description
MSB 3 bit	Motor selection ID
0x01(LSB 5 bit)	**Set register frame**. It is used to write a value into arelevant motor control variable. See Set register frame.
0x02	**Get register frame**. It is used to read a value from a relevant motor control variable. See Get register frame.
0x03	**Execute command frame**. It is used to send a command to the motor control object. See Execute command frame.
0x06	**Get board info**. It is used to retrieve the information about the firmware currently running on the microcontroller.
0x07	**Exec ramp**. It is used to execute a speed ramp. See Execute ramp frame.
0x08	**Get revup data**. It is used to retrieve the revup parameters. See Get revup data frame.
0x09	**Set revup data**. It is used to set the revup parameters. See Set revup data frame.
0x0A	**Set current references**. It is used to set current reference. See Set current references frame.

Set register frame

- The set register frame is sent by the master to write a value into a relevant motor control variable.
  - Payload length depends on reg_id.
  - Reg id indicates the register to be updated.
  - Remaining payload contains the value to be updated starting from least significant byte to most significant byte.



Set register

Set register frame

PC →

0x01	PAYLOAD_LENGTH	REG_ID	REG_LB	…	REG_HB	CRC

Data Acknowledgment frame, No errors

Board →

0xF0	0x00	CRC

OR

Set register frame

PC →

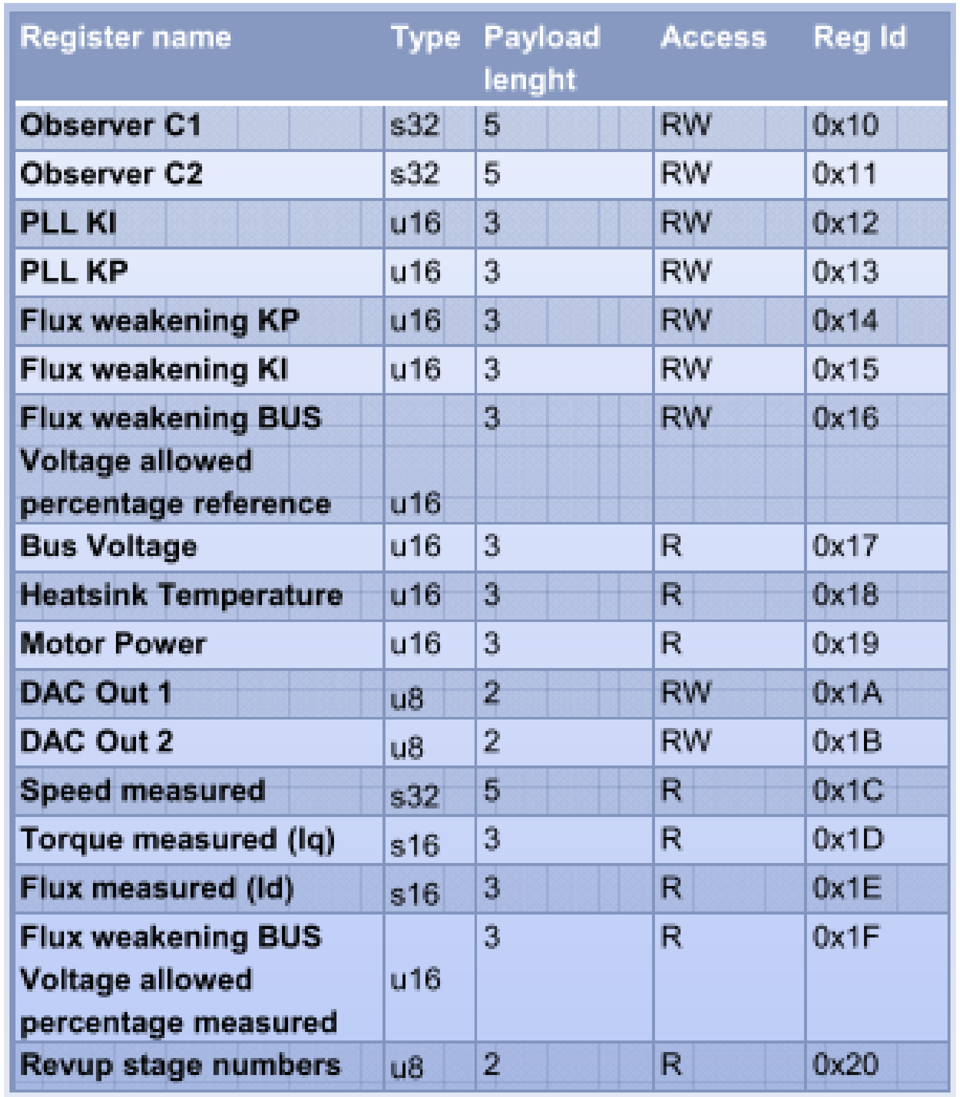
0x01	PAYLOAD_LENGTH	REG_ID	REG_LB	…	REG_HB	CRC

Error Acknowledgment frame

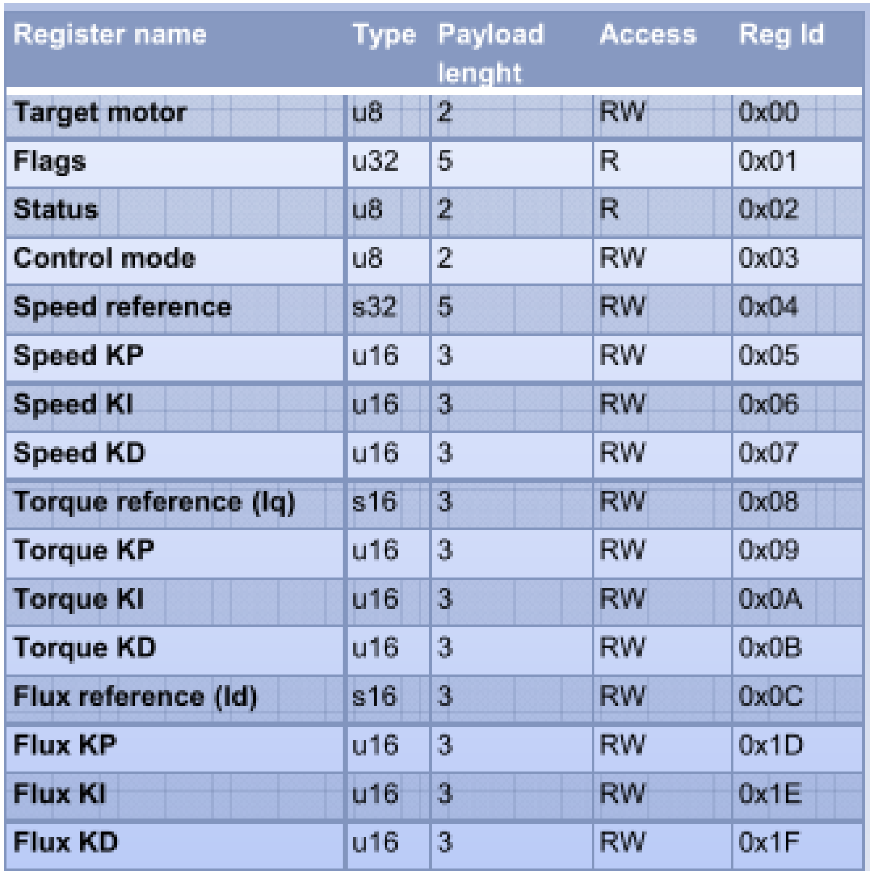
Board →

0xFF	0x01	ERROR_CODE	CRC

Motor control registers



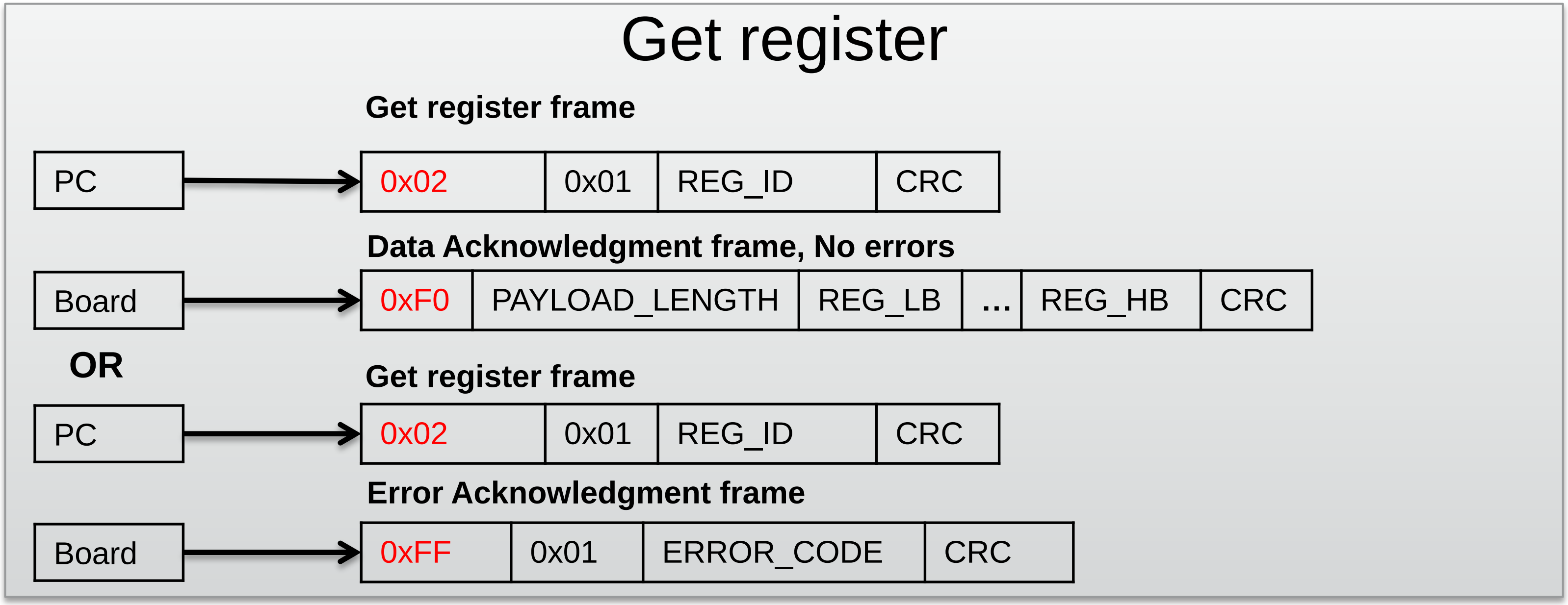
Register name	Type	Payload lenght	Access	Reg Id
Observer C1	s32	5	RW	0x10
Observer C2	s32	5	RW	0x11
PLL KI	u16	3	RW	0x12
PLL KP	u16	3	RW	0x13
Flux weakening KP	u16	3	RW	0x14
Flux weakening KI	u16	3	RW	0x15
Flux weakening BUS Voltage allowed percentage reference	u16	3	RW	0x16
Bus Voltage	u16	3	R	0x17
Heatsink Temperature	u16	3	R	0x18
Motor Power	u16	3	R	0x19
DAC Out 1	u8	2	RW	0x1A
DAC Out 2	u8	2	RW	0x1B
Speed measured	s32	5	R	0x1C
Torque measured (Iq)	s16	3	R	0x1D
Flux measured (Id)	s16	3	R	0x1E
Flux weakening BUS Voltage allowed percentage measured	u16	3	R	0x1F
Revup stage numbers	u8	2	R	0x20



Register name	Type	Payload lenght	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	RW	0x04
Speed KP	u16	3	RW	0x05
Speed KI	u16	3	RW	0x06
Speed KD	u16	3	RW	0x07
Torque reference (Iq)	s16	3	RW	0x08
Torque KP	u16	3	RW	0x09
Torque KI	u16	3	RW	0x0A
Torque KD	u16	3	RW	0x0B
Flux reference (Id)	s16	3	RW	0x0C
Flux KP	u16	3	RW	0x1D
Flux KI	u16	3	RW	0x1E
Flux KD	u16	3	RW	0x1F

Get register frame

- The get register frame is sent by the master to read a value from a relevant motor control variable.
  - Payload length is always 1.
  - Reg Id indicates the register to be queried.
  - 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.
    - Error Acknowledgment frame, if the operation has not been successfully completed by the firmware. The payload of this Error Acknowledgment frame is always 1.



Get register

Get register frame

PC →

0x02	0x01	REG_ID	CRC

Data Acknowledgment frame, No errors

Board →

0xF0	PAYLOAD_LENGTH	REG_LB	…	REG_HB	CRC

OR

Get register frame

PC →

0x02	0x01	REG_ID	CRC

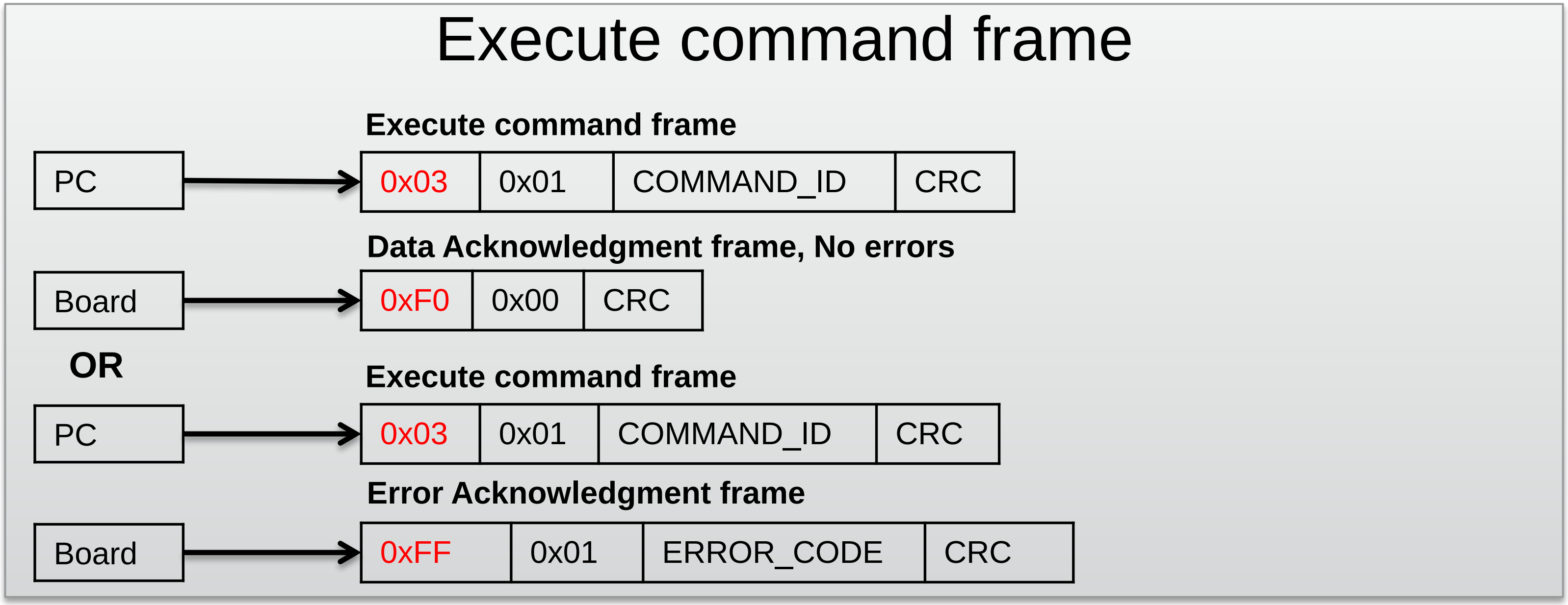
Error Acknowledgment frame

Board →

0xFF	0x01	ERROR_CODE	CRC

Execute command frame

- The execute command frame is sent by the master to the motor control firmware to request the execution of a specific command.
  - Payload length is always 1.
  - Command Id indicates the requested command.



Execute command frame

PC → Execute command frame

0x03	0x01	COMMAND_ID	CRC

Board → Data Acknowledgment frame, No errors

0xF0	0x00	CRC

OR

PC → Execute command frame

0x03	0x01	COMMAND_ID	CRC

Board → Error Acknowledgment frame

0xFF	0x01	ERROR_CODE	CRC

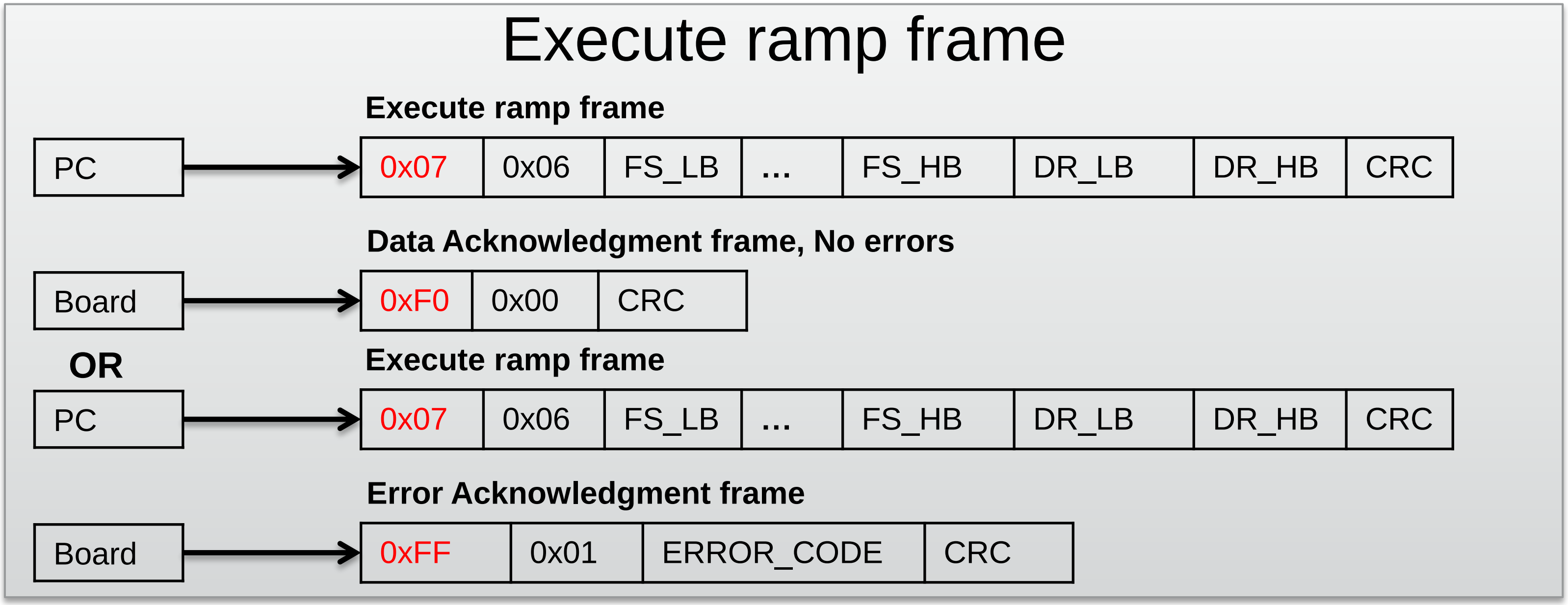
List of commands

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 speed ramp that is currently executed.
Start/Stop	0x06	Indicates the user request to start the motor if the motor is still or stop the motor is 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.

Execute ramp frame

- The execute ramp frame is sent by the master to the motor control firmware to request the execution of a speed ramps.

- A speed ramps start always from the actual motor speed, and is defined by a duration and final speed.
  - Payload length is always 6.
  - FS_LB, FS_.., FS_HB represent the final speed expressed in rpm respectively from least significant byte to the most significant byte.
  - DR_LB and DR_HB represent the duration expressed in milliseconds respectively least significant byte and most significant byte.



Execute ramp frame

Execute ramp frame

PC →

0x07	0x06	FS_LB	…	FS_HB	DR_LB	DR_HB	CRC

Data Acknowledgment frame, No errors

Board →

0xF0	0x00	CRC

OR

Execute ramp frame

PC →

0x07	0x06	FS_LB	…	FS_HB	DR_LB	DR_HB	CRC

Error Acknowledgment frame

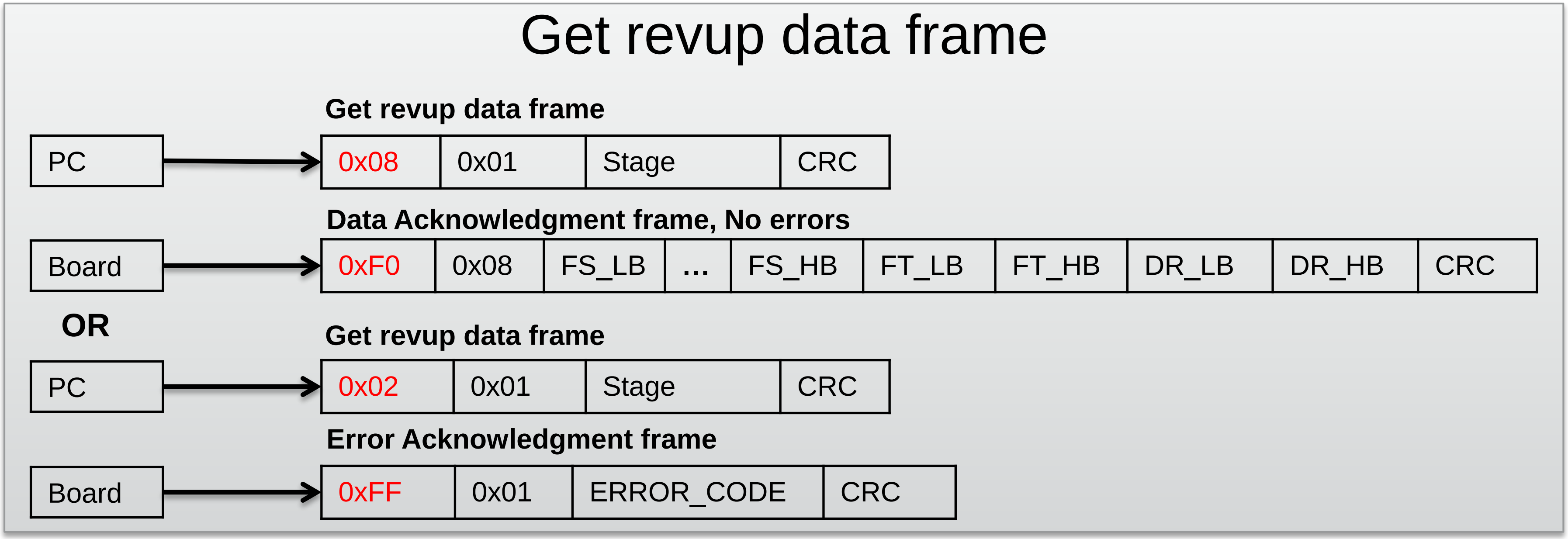
Board →

0xFF	0x01	ERROR_CODE	CRC

Get revup data frame 1/2

- The get rev-up data frame is sent by the master to retrieve the current rev-up parameters.
  - The master indicates the requested stage parameter sending the stage number in the starting frame payload. So 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 pay load size of this Data Acknowledgment frame is always 8.
    - FS_LB, FS_.., FS_HB represent the final speed of the selected stage expressed in rpm respectively from least significant byte to most significant byte.
    - FT_LB and FT_HB represent the final torque of the selected stage expressed in digit respectively least significant byte and most significant byte.
    - DR_LB and DR_HB represent the duration of the selected stage expressed in milliseconds respectively least significant byte and 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.

Get revup data frame 2/2



Get revup data frame

Get revup data frame

PC →

0x08	0x01	Stage	CRC

Data Acknowledgment frame, No errors

Board →

0xF0	0x08	FS_LB	…	FS_HB	FT_LB	FT_HB	DR_LB	DR_HB	CRC

OR

Get revup data frame

PC →

0x02	0x01	Stage	CRC

Error Acknowledgment frame

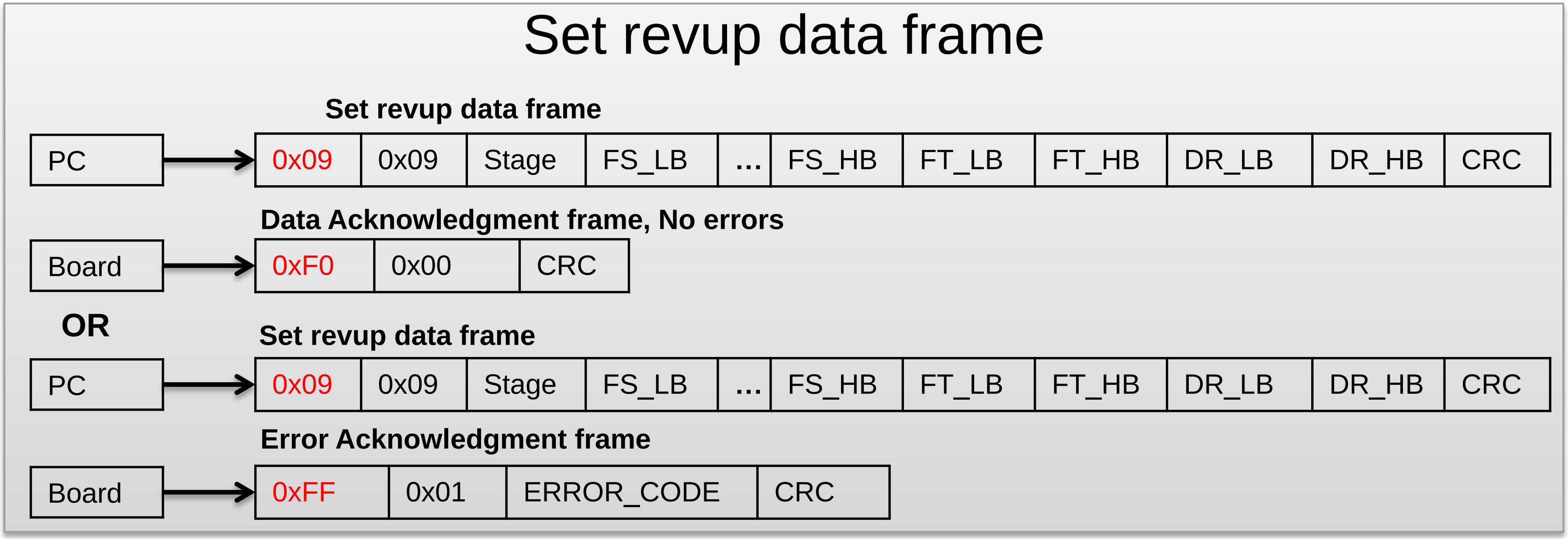
Board →

0xFF	0x01	ERROR_CODE	CRC

Set revup data frame 1/2

- The set rev-up data frame is sent by the master to modify the rev-up parameters.
  - The master sends the requested stage parameter.
  - The payload length is always 7.
  - Stage is the revup stage that will be modified.
  - FS_LB, FS_.., and FS_HB is the requested new final speed of the selected stage expressed in rpm respectively from least significant byte to most significant byte.
  - FT_LB and FT_HB is the requested new final torque of the selected stage expressed in digit respectively least significant byte and most significant byte.
  - DR_LB and DR_HB is the requested new duration of the selected stage expressed in DR_LB and DR_HB is the requested new duration of the selected stage expressed in milliseconds respectively least significant byte and most significant byte.

Set revup data frame 2/2



Set revup data frame

Set revup data frame

PC →

0x09	0x09	Stage	FS_LB	…	FS_HB	FT_LB	FT_HB	DR_LB	DR_HB	CRC

Data Acknowledgment frame, No errors

Board →

0xF0	0x00	CRC

OR

Set revup data frame

PC →

0x09	0x09	Stage	FS_LB	…	FS_HB	FT_LB	FT_HB	DR_LB	DR_HB	CRC

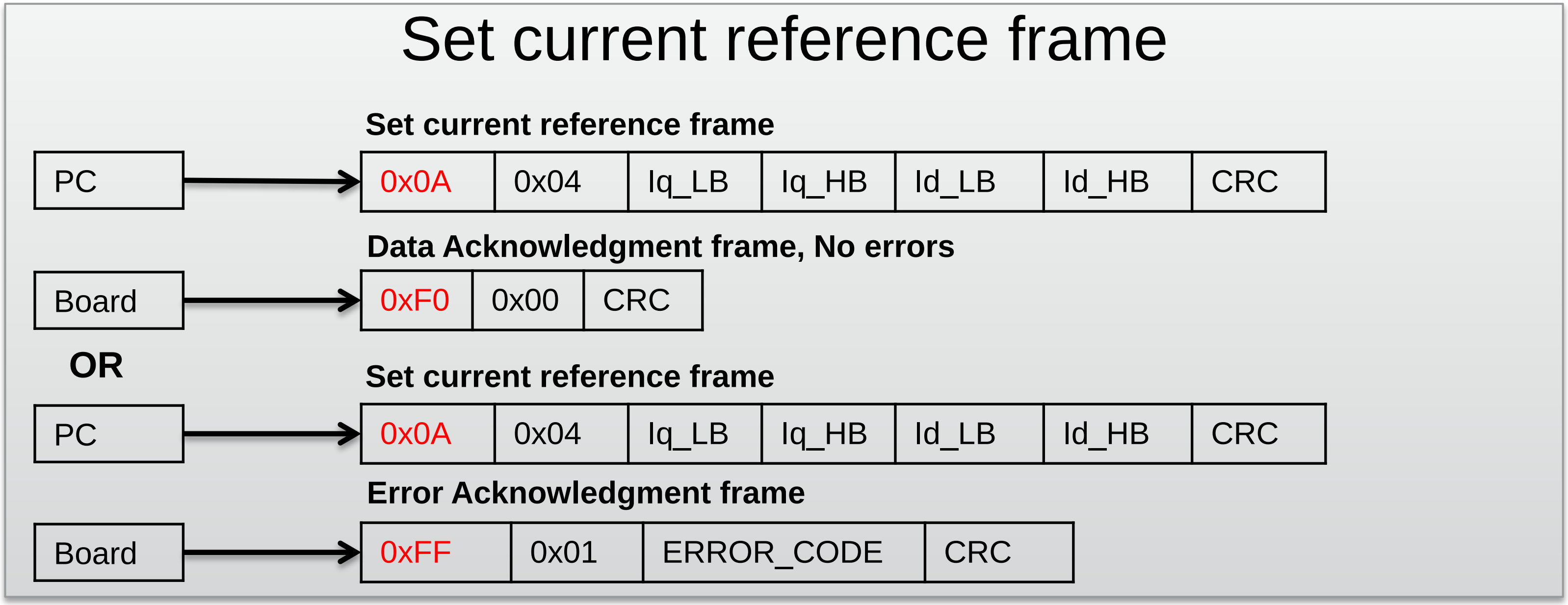
Error Acknowledgment frame

Board →

0xFF	0x01	ERROR_CODE	CRC

Set current reference frame

- The set current references frame is sent by the master to modify the current references Iq, Id.
  - The master sends the requested current references.
  - The payload length is always 4.
  - Iq_LB and Iq_HB is the requested new Iq reference expressed in digit respectively least significant byte and most significant byte.
  - Id_LB and Id_HB is the requested new Id reference expressed in digit respectively least significant byte and most significant byte.



Set current reference frame

Set current reference frame

PC →

0x0A	0x04	Iq_LB	Iq_HB	Id_LB	Id_HB	CRC

Data Acknowledgment frame, No errors

Board →

0xF0	0x00	CRC

OR

Set current reference frame

PC →

0x0A	0x04	Iq_LB	Iq_HB	Id_LB	Id_HB	CRC

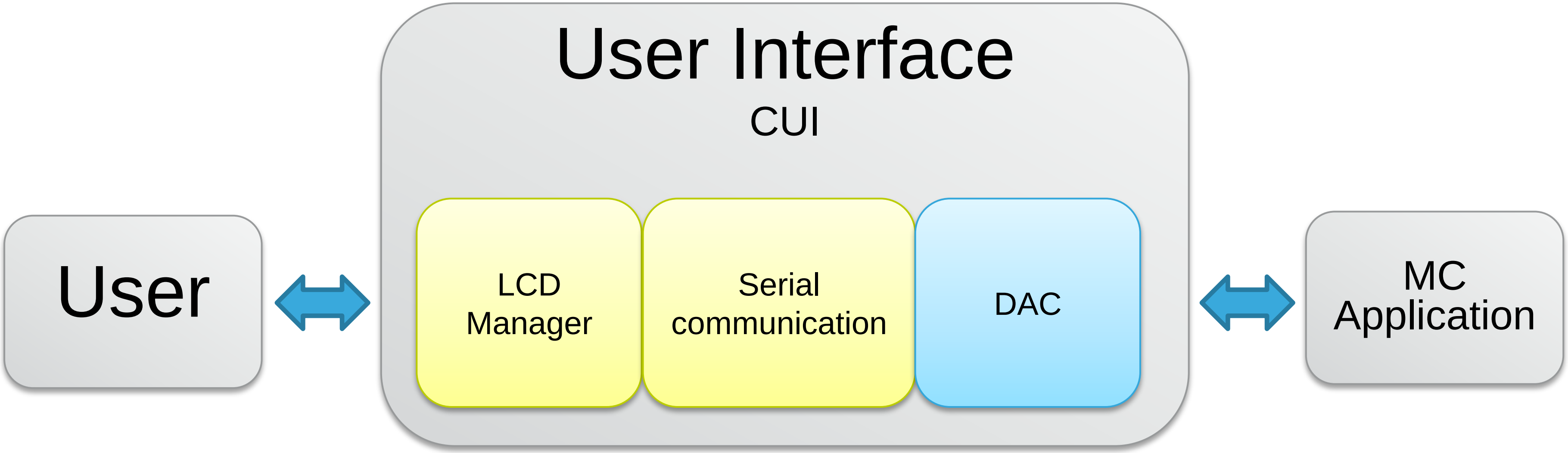
Error Acknowledgment frame

Board →

0xFF	0x01	ERROR_CODE	CRC

User interface architecture

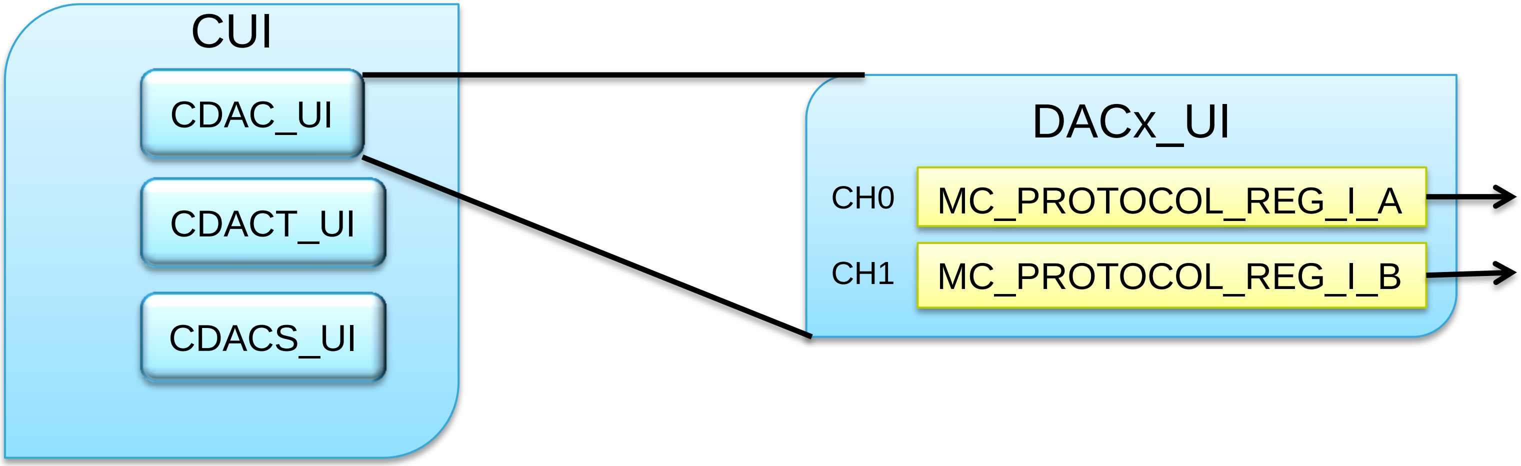
- DAC manager (CDAC_UI) is used to manage the DAC outputs.



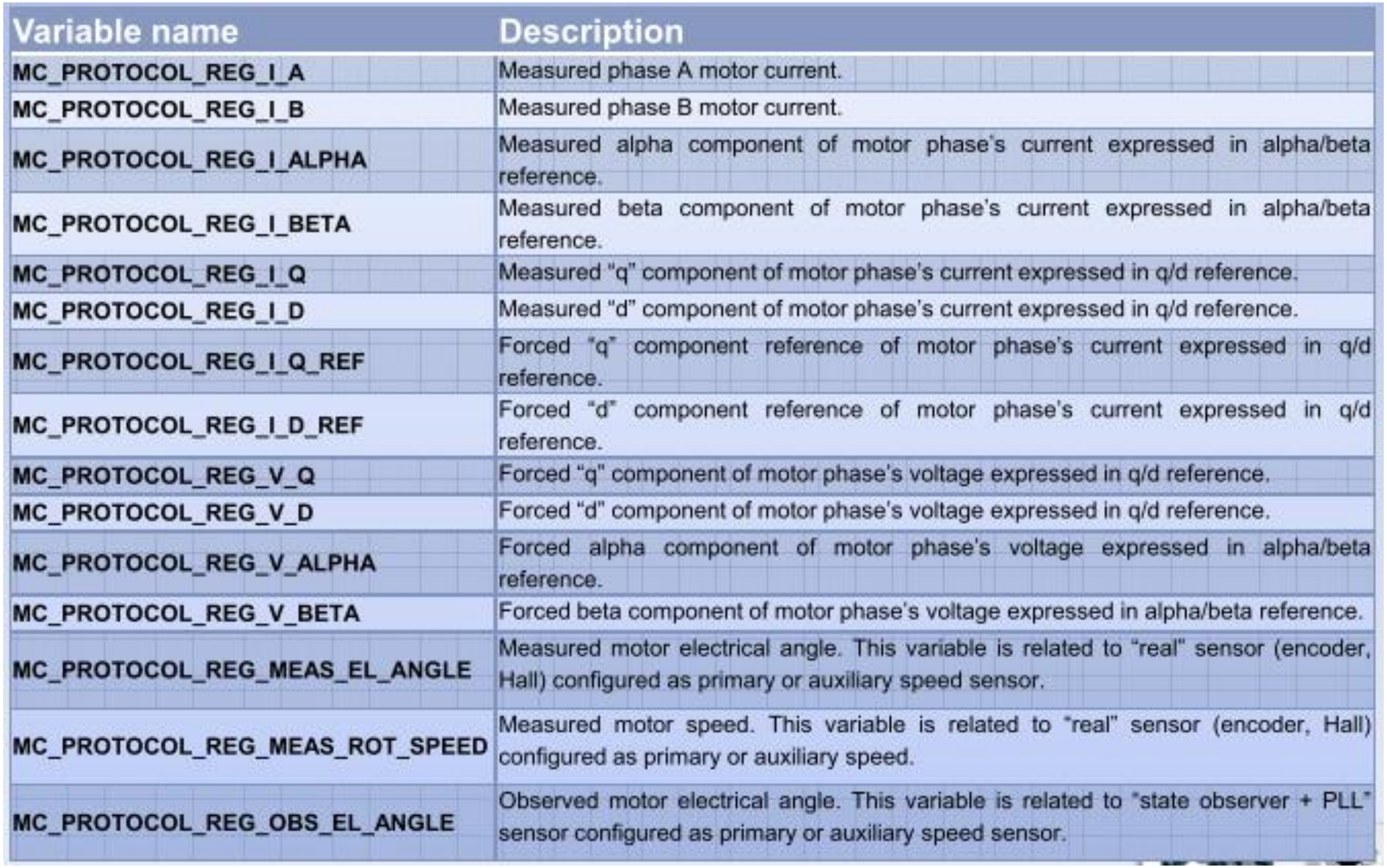
DAC manager class (CDACx_UI)

- There are three derivatives of CUI that implements DAC managements:
    - DAC_UI (DAC_UI): DAC peripheral is using as output.
    - DACRCTIMER_UI (DACT_UI): General purpose timer is used ad output together with a RC filter.
    - DACSPI_UI (DACS_UI): SPI peripheral is used as output. The data can be d codified by an oscilloscope for instance.

- For each DAC class are defined the number of channels (actually the DAC channels defined is two) and the DAC variables. The DAC variables are pre defined motor control variables or user defined variables that can be put in out by DAC objects. DAC variables can be any value MC_PROTOCOL_REG_xxx exported by UserInterfaceClass.h.

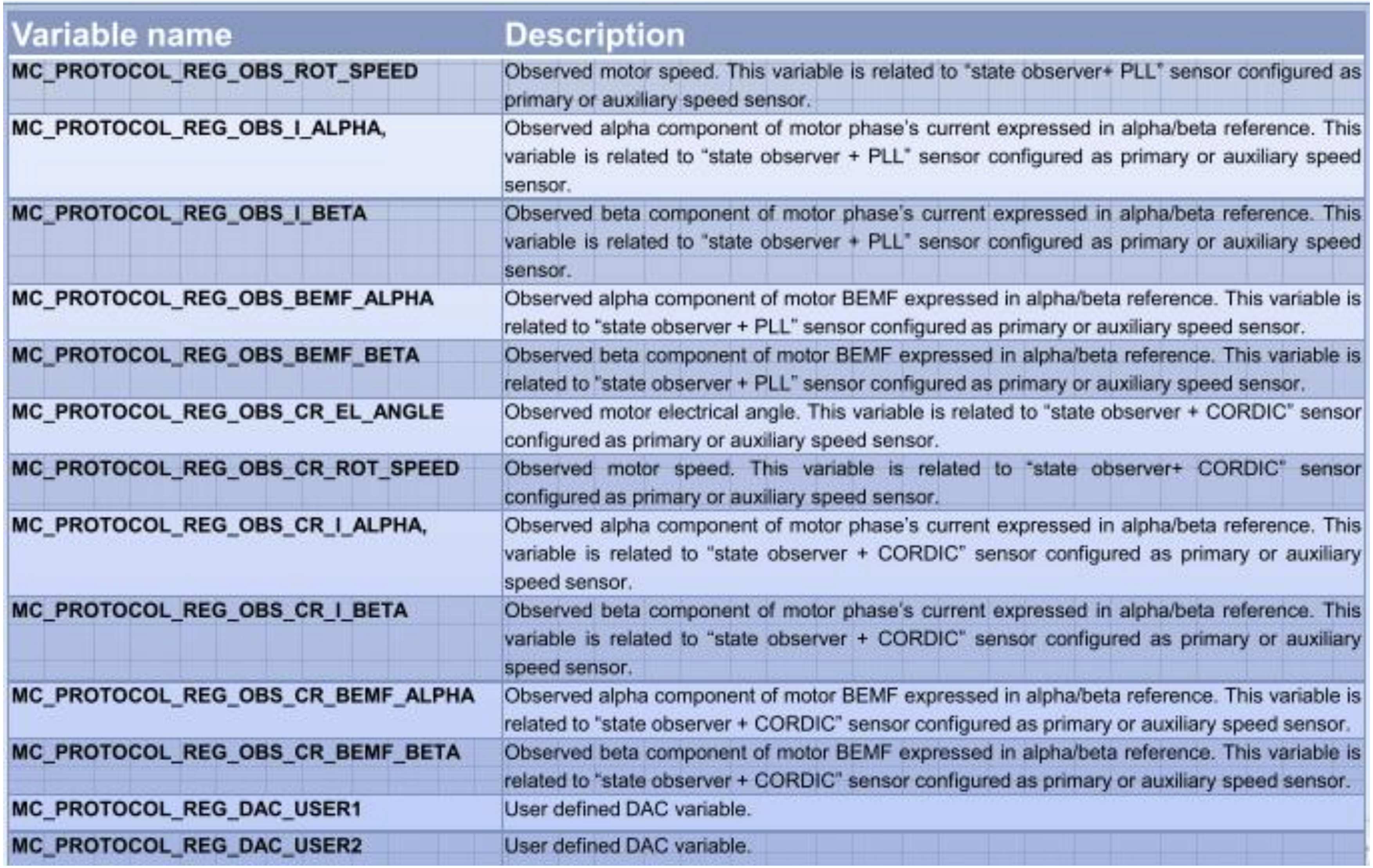


DAC variables 1/2



Variable name	Description
MC_PROTOCOL_REG_I_A	Measured phase A motor current.
MC_PROTOCOL_REG_I_B	Measured phase B motor current.
MC_PROTOCOL_REG_I_ALPHA	Measured alpha component of motor phase's current expressed in alpha/beta reference.
MC_PROTOCOL_REG_I_BETA	Measured beta component of motor phase's current expressed in alpha/beta reference.
MC_PROTOCOL_REG_I_Q	Measured "q" component of motor phase's current expressed in q/d reference.
MC_PROTOCOL_REG_I_D	Measured "d" component of motor phase's current expressed in q/d reference.
MC_PROTOCOL_REG_I_Q_REF	Forced "q" component reference of motor phase's current expressed in q/d reference.
MC_PROTOCOL_REG_I_D_REF	Forced "d" component reference of motor phase's current expressed in q/d reference.
MC_PROTOCOL_REG_V_Q	Forced "q" component of motor phase's voltage expressed in q/d reference.
MC_PROTOCOL_REG_V_D	Forced "d" component of motor phase's voltage expressed in q/d reference.
MC_PROTOCOL_REG_V_ALPHA	Forced alpha component of motor phase's voltage expressed in alpha/beta reference.
MC_PROTOCOL_REG_V_BETA	Forced beta component of motor phase's voltage expressed in alpha/beta reference.
MC_PROTOCOL_REG_MEAS_EL_ANGLE	Measured motor electrical angle. This variable is related to "real" sensor (encoder, Hall) configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_MEAS_ROT_SPEED	Measured motor speed. This variable is related to "real" sensor (encoder, Hall) configured as primary or auxiliary speed.
MC_PROTOCOL_REG_OBS_EL_ANGLE	Observed motor electrical angle. This variable is related to "state observer + PLL" sensor configured as primary or auxiliary speed sensor.

DAC variables 2/2



Variable name	Description
MC_PROTOCOL_REG_OBS_ROT_SPEED	Observed motor speed. This variable is related to “state observer+ PLL” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_I_ALPHA,	Observed alpha component of motor phase's current expressed in alpha/beta reference. This variable is related to “state observer + PLL” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_I_BETA	Observed beta component of motor phase's current expressed in alpha/beta reference. This variable is related to “state observer + PLL” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_BEMF_ALPHA	Observed alpha component of motor BEMF expressed in alpha/beta reference. This variable is related to “state observer + PLL” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_BEMF_BETA	Observed beta component of motor BEMF expressed in alpha/beta reference. This variable is related to “state observer + PLL” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_CR_EL_ANGLE	Observed motor electrical angle. This variable is related to “state observer + CORDIC” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_CR_ROT_SPEED	Observed motor speed. This variable is related to “state observer+ CORDIC” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_CR_I_ALPHA,	Observed alpha component of motor phase's current expressed in alpha/beta reference. This variable is related to “state observer + CORDIC” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_CR_I_BETA	Observed beta component of motor phase's current expressed in alpha/beta reference. This variable is related to “state observer + CORDIC” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_CR_BEMF_ALPHA	Observed alpha component of motor BEMF expressed in alpha/beta reference. This variable is related to “state observer + CORDIC” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_OBS_CR_BEMF_BETA	Observed beta component of motor BEMF expressed in alpha/beta reference. This variable is related to “state observer + CORDIC” sensor configured as primary or auxiliary speed sensor.
MC_PROTOCOL_REG_DAC_USER1	User defined DAC variable.
MC_PROTOCOL_REG_DAC_USER2	User defined DAC variable.

DAC customization Set default variables

- Obtain DAC object:

```
#include “UITask.h”
CUI oDAC  = GetDAC();
```

- Configure the required DAC variables in the DAC channel using the UI_DACChannelConfig method:

```
UI_DACChannelConfig(oDAC, DAC_CH0, MC_PROTOCOL_REG_IA);
```

DAC customization using user defined variables

- Obtain DAC object:

```
#include “UITask.h”
CUI oDAC  = GetDAC();
```

- Call the UI_DACSetUserChannelValue method of CUI object passing the value (hUser1 in the following example).

```
User0:  UI_DACSetUserChannelValue(oDAC,0,hUserVariable1);
User1:  UI_DACSetUserChannelValue(oDAC,1,hUserVariable2);
```