MCU Motor Studio
Serial communication - Protocol Specification

Description

This document describes the MCU Motor Studio Serial Protocol, This Protocol defines the serial communication between a host (PC) and a Toshiba MCU with Toshiba firmware running on MCU.

Table of Contents

List of Figures

List of Tables

Conventions used in this document

Numerical Values

Hexadecimal number: 0xABC or 0h12F
Decimal number: 123 or 0d123 (explicitly indicating the decimal numbers)
Binary number: 0b111

Signals

Active low signals are indicated by a _N at the end of the signal name. Example: RESET_N.
Assertion of a signal shall mean its activation (transition to the active state). Deassertion of a signal shall mean its deactivation (transition to the inactive state).

Bus signals are indicated by [x:y] at the end of signal name. Example: DATA[3:0] indicates a four bit bus with the individual bus signals DATA[3], DATA[2], DATA[1] and DATA[0].

Registers

Register names are indicated by square brackets […]. Example: [ABCD].

Two or more of the same kind of registers, fields, and bit names are collectively referred to by using a numerical suffix n. Example: [XYZ1], [XYZ2] and [XYZ3] are collectively referred to as [XYZn].

The bit width of a register is expressed as [x:y] where x is the number of the most significant bit and y is the number of the least significant bit. Example: [XYZ][3:0] indicates a four bit-wide register named XYZ.

The configuration value of a register is expressed by either a hexadecimal number or a binary number. Example: [ABCD].EFG = 0x01 (hexadecimal), [XYZn].XY = 0b1 (binary).

The following definitions apply for Bytes and Words:

Byte	8 bits
Half Word	16 bits
Word	32 bits
Double Word	64 bits

Unless specified otherwise, registers support only word access.

Register which are indicated to be reserved must not be rewritten. The read value from reserved registers must not be used.

Properties of each bit in a register are expressed as follows:

R	Read only
W	Write only
W1C	Write 1 Clear; the corresponding bit is cleared (=0) when "1" is written to this bit.
W1S	Write 1 Set; the corresponding bit is set (=1) when "1" is written to this bit.
R/W	Read and Write are possible.
R/W0C	Read/Write 0 Clear
R/W1C	Read/Write 1 Clear
R/W1S	Read/Write 1 Set
RS/WC	Read Set/Write Clear; set after read operation, cleared after write operation.

Reading from register bits having a default value of "—" will result in an unknown value.
In case of write accesses to registers containing both read/write (R/W) and read-only (R) bits, the read-only bits shall be written with their default value. If this default is "—", follow the instructions of each register.

Reserved bits of Write-only (W) register should be written with their default value. If this default is "—", follow the instructions of each register.

1. Introduction

This document describes the Motor Control Serial Protocol. This Protocol defines the serial communication between a host and “MCU Motor Studio” Firmware version 3.10.
The host is typically a Personal Computer connected via USB to serial converter and controls the Motor MCU. The Motor Control Protocol is a strict request and response protocol. The Host sends a request, and the Toshiba MCU transmits the response. The host is not allowed to send a new request before the first response has been received (or a timeout has occurred).

The physical interface between a host and the MCU is a 4-wire connection (no use of CTS and RTS).

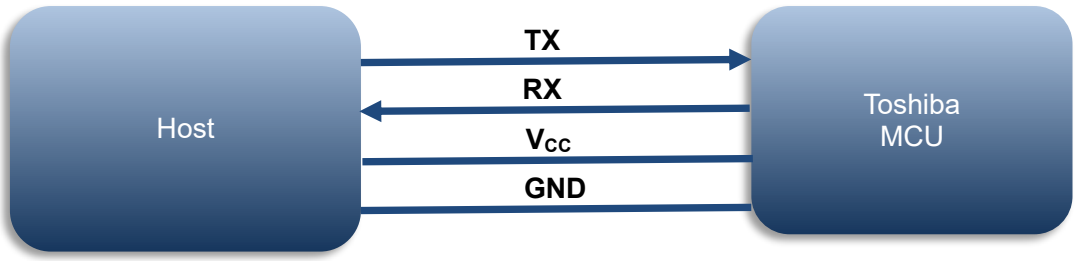

Figure 1-1: Host - MCU physical connection

2. Electrical specifications of UART interface

2.1. Description of the UART lines

Pin	Description
TX	Transmit from host to MCU
RX	Reception from MCU to host
Vcc	Voltage supply
GND	Ground

Table 2-1: UART interface pins

2.2. UART settings

Name	Setting
Baudrate	115200
Databits	8
Stopbits	1
Parity	None

Table 2-2: UART settings

3. Telegram structure

BOT (0x11)	CMD or Event	Data 0	Data 1	…	Data n	CRC8	EOT(0x13)

Table 3-1:Telegram structure

The transmission starts with a BOT (Begin Of Transmission – 0x11), followed by the command byte. After the command the payload for the command is transmitted. The amount of data for the payload is depending on the command. That means that there is not a fixed frame size. Both Toshiba MCU and host are aware of the amount of data following a certain command. After the payload a CRC8 is calculated over the command and payload data is transmitted, followed by an EOT (End of Transmission – 0x13).
The transmission frame from the Toshiba MCU is similar, just the payload doesn't reflect the request but the response.
The byte order is exactly the byte order represented in the memory of the CortexM3, so communicating via serial line is a straight forward if the orientation in the memory is known.
If the transmission to the target is disturbed (CRC check failed), the target will answer with 0xee.

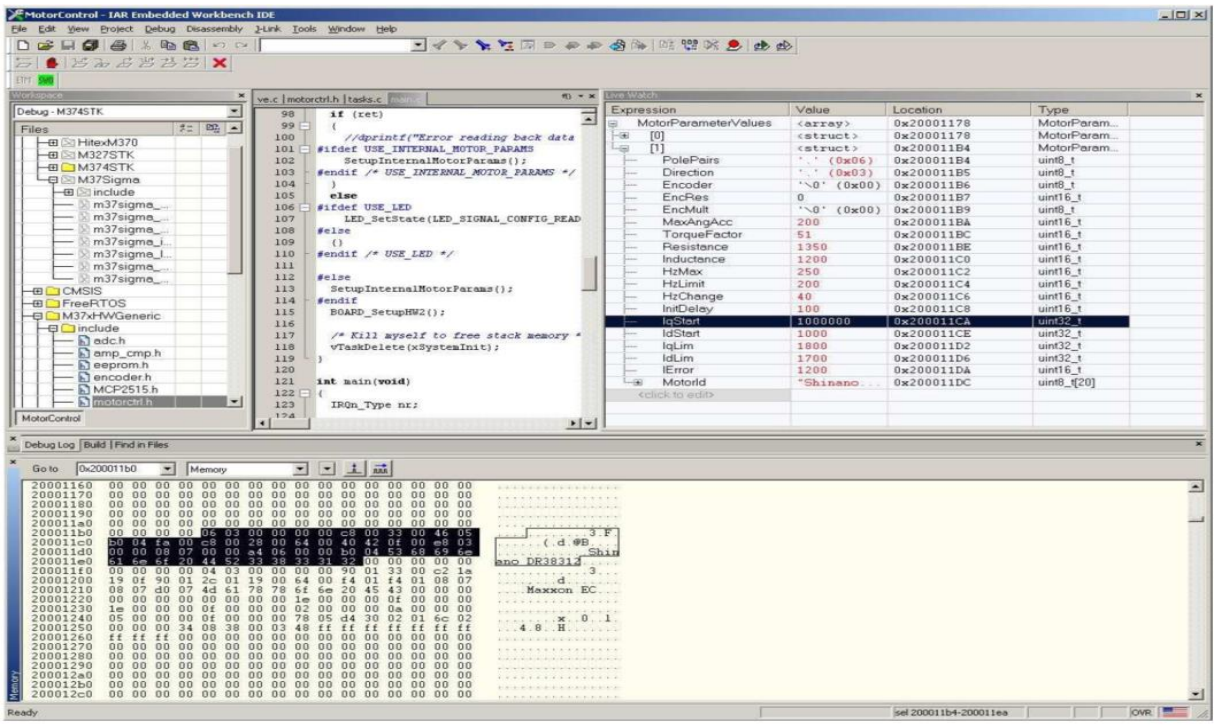

Figure 3-2: Telegram capture at the Motor Control Firmware side

As a proof this is the logged data from the serial port:

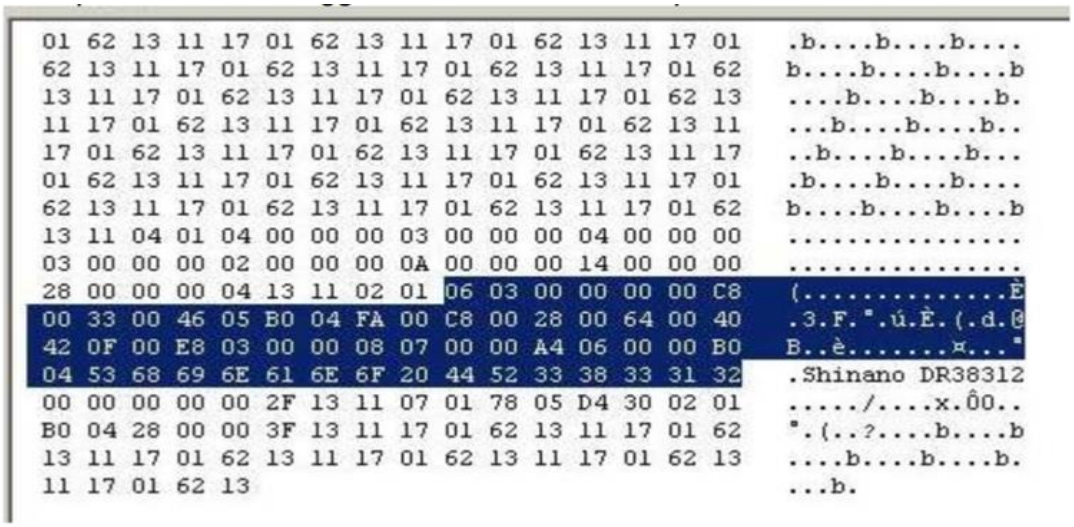

Figure 3-3: Example of a telegram, captured with IAR

So generally spoken a 32bit value of <b31-24><b23-16><b15-8><b7-0>. Will be transmitted in the following order: <b7-0><b15-8><b23-16><b31-24>.
A 16bit value <b15-8><b7-0> in the order <b7-0><b15-8>.

4. Serial Command / Event Description

4.1. Supported Serial Commands / Events

The following table lists all supported serial requests / responses:
Note: The payload depends on the transmit direction.

Command / Event	ID	Notes
RPC_StartMotor	0x00	-
RPC_StopMotor	0x01	-
RPC_GetMotorState	0x02	-
RPC_StoreParameters	0x03	-
RPC_ClearParameters	0x04	-
RPC_ConfigDSOLog	0x05	-
RPC_GetDSOLogData	0x06	-
RPC_ConfigureHsDSO	0x07	-
RPC_DoTurn	0x08	Only supported if turn control is enabled.
RPC_GetOneMotorParameter	0x09	-
RPC_SetOneMotorParameter	0x0A	-
RPC_GetMotorParameters	0x0B	-
RPC_SetMotorParameters	0x0C	-
RPC_DoLinearMotion	0x0D	Only supported if motion control is enabled.
RPC_GetAbsolutePosition	0x0E	Only supported if motion control is enabled.
RPC_AbortLinearMotion	0X0F	Only supported if motion control is enabled.
RPC_StartMotorTorqueCtrl	0x10	-
RPC_GetMotorControlMethod	0x11	-
RPC_GetEncoderCounter	0x12	-
RPC_GetFWVersion	0x14	-
RPC_SetDemoState	0x21	Toshiba has developed several demo systems. This command is used for such a system, but can also be used for customer projects.
RPC_GetDemoState	0x22	Toshiba has developed several demo systems. This command is used for such a system, but can also be used for customer projects.
RPC_GetExtendedMotorState	0x23	-

Table 4-1: List of available commands/events

4.2. Detailed Command / Event Description

4.2.1. CMD : (0x00) RPC_StartMotor

This command starts the VE and the selected Motor. The default control is by **Speed**. Control by **Torque** is also supported, different start command is to be used. The motor will be spun-up until the set rotational speed is achieved and then it will be maintained.
Payload Data:

Request Payload Data:

uint8_t motor_nr; /* Motor number */

Response Payload Data:

uint8_t dummy; /* Dummy value - due to protocol specification */

Example:

	BOT	**CMD**	**MOTOR_NR**	**CRC**	**EOT**
Request:	0x11	0x00	0x01	0x07	0x13

	BOT	**CMD**	**DMY**	**CRC**	**EOT**
Response:	0x11	0x00	0x00	0x00	0x13

4.2.2. CMD : (0x00) RPC_StopMotor

This command stops the VE and the Motor.
Payload Data:

Request Payload Data:

uint8_t motor_nr; /* Motor number */

Response Payload Data:

uint8_t dummy; /* Dummy value - due to protocol specification */

Example:

	BOT	**CMD**	**MOTOR_NR**	**CRC**	**EOT**
Request:	0x11	0x01	0x01	0x00	0x13

	BOT	**CMD**	**DMY**	**CRC**	**EOT**
Response:	0x11	0x01	0x00	0x07	0x13

4.2.3. CMD : (0x02) RPC_GetMotorState

This command gets the actual status of the motor. Legacy function kept for backward compatibility.

Payload Data:

Request Payload Data:

uint8_t motor_nr; /* Motor number */

Response Payload Data:

int32_t ActualSpeed; /* [RPM] - Actual Rotation speed of Motor - negative is CCW*/

int32_t TargetSpeed; /* [RPM] - Target Rotation speed of Motor - negative is CCW*/

```
uint32_t Current;          /* [mA] - Current of motor */
uint32_t Torque;           /* [Ncm] - Torque of Motor */
uint32_t Timestamp;        /* [systick]- systick counter of system */
```

Example:

	BOT CMD MOTOR_NR CRC EOT
Request:	0x11 0x02 0x01 0x09 0x13
	BOT CMD ACTUAL SPEED TARGET SPEED CURRENT
Response:	0x11 0x02 0xf1 0x01 0x00 0x00 0xf4 0x01 0x00 0x00 0x3c 0x00 0x00 0x00
	TORQUE TIMESTAMP CRC EOT
	0xcc 0x00 0x00 0x00 0xea 0x21 0x01 0x00 0x43 0x13

- Actual Speed: 0x000001f1 = 497 [RPM]
- Target Speed: 0x000001f4 = 500 [RPM]
- Current : 0x0000003c = 60 [mA]
- Torque : 0x000000cc = 204 [Ncm]
- Timestamp : 0x000121ea = 74218 [systicks]

4.2.4. CMD : (0x03) RPC_StoreParameters

This command stores the actual motor parameters to the EEProm / Flash.

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;   /* Motor number */
```

Response Payload Data:

```
uint8_t dummy;      /* Dummy value - due to protocol specification */
```

Example:

	BOT CMD MOTOR_NR CRC EOT
Request:	0x11 0x03 0x01 0x0e 0x13
	BOT CMD DMY CRC EOT
Response:	0x11 0x03 0x00 0x09 0x13

4.2.5. CMD : (0x04) RPC_ClearParameters

This command clears the EEProm / Flash content.

Payload Data:

Request Payload Data:

```
uint8_t dummy;          /* Dummy value - due to protocol specification */
```

Response Payload Data:

```
uint8_t dummy;          /* Dummy value - due to protocol specification */
```

Example:

	BOT	CMD	DMY	CRC	EOT
Request:	0x11	0x04	0x00	0x1c	0x13
Response:	0x11	0x04	0x00	0x1c	0x13

4.2.6. CMD : (0x05) RPC_ConfigDSOLog

This command configures and starts the DSO snapshot. After finishing the logging the data can be read out using command 0x06 - RPC_GetDSOLogData.

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;          /* Motor number */
uint32_t selected_values;  /* Bitfield with DSO_Selection enumerated values
                              for selecting the signals to be logged */
uint32_t trigger_on_value; /* Bitfield with DSO_Selection enumerated values
                              for selecting the signals to trigger on */
uint8_t trigger_mode;      /* Bitfield with TriggerMode enumerated values to
                              select the trigger mode of the DSO */
uint16_t trigger_level;    /* Trigger level for the DSO */
uint8_t spread_factor;     /* Factor of which the logging is spread */
```

Response Payload Data:

```
uint16_t logsteps;         /* Number of data-sets that will be logged */
uint32_t selected_values            / <Bitfield of DSO Parameters to log>
                                     trigger_on_value:
                                      Va = (1<< 0), a-phase voltage
                                      Vb = (1<< 1), b-phase voltage
                                      Vc = (1<< 2), c-phase voltage
                                      Valpha = (1<< 3), alpha-axis voltage
                                      Vbeta = (1<< 4), beta-axis voltage
                                      Id = (1<< 5), d-axis current
                                      Id_Ref = (1<< 6), d-axis reference current
                                      Iq = (1<< 7), q-axis current
                                      Iq_Ref = (1<< 8), q-axis reference current
                                      Vd = (1<< 9), d-axis voltage
                                      Vq = (1<<10), q-axis voltage
                                      Vdi = (1<<11), d-axis integral term
                                      Vqi = (1<<12), q-axis integral term
                                      Theta = (1<<13), Phase 0
                                      Omega = (1<<14), Rotation speed
                                      Omega_Calc = (1<<15), Calculated Rotation speed
                                      SIN_Theta = (1<<16), Sine Value at 0
                                      COS_Theta = (1<<17), Cosine Value at 0
                                      Sector = (1<<18), Sector information
                                      VDC = (1<<19), Motor supply voltage
                                      Ia = (1<<20), a-phase current
                                      Ib = (1<<21), b-phase current
                                      Ic = (1<<22), c-phase current
                                      Ialpha = (1<<23), alpha-axis current
                                      Ibeta = (1<<24), beta-axis current
                                      VE_Stage = (1<<25), stage of Vector Engine
                                      User 1 = (1<<26), User Variable
                                      User 2 = (1<<27), User Variable
```

User 3 = (1<<28), User Variable
User 4 = (1<<29), User Variable
User 5 = (1<<30), User Variable
User 6 = (1<<31),

Example:

BOT CMD NR SELECTED TRIGGER MODE LEVEL
Request: 0x11 0x05 0x01 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x00 0x00
SPRE CRC EOT
0x00 0xf0 0x13

BOT CMD LOGS CRC EOT
Response: 0x11 0x05 0x14 0x00 0x42 0x13

Request:

- Selected: 0x00000007 -> Va/Vb/Vc
- Trigger: 0x00000000 -> no trigger
- Mode: 0x03 -> don't care (as no trigger)
- Level: 0x0000 -> don't care (as no trigger)
- Spread: 0x00 -> log every value

Response:

- Logsteps: 0x14 -> 14 sets of 16 logged values

4.2.7. CMD: (0x06) RPC_GetDSOLogData

Payload Data:

Request Payload Data:
```
uint8_t dummy;          /* Dummy value - due to protocol specification */
```
Response Payload Data:
```
int16_t value[16];      /* Log data of selected values - up to 16 values
                           have to be read until return value is 0 */
```

There are different answers of this command:
0- success – data is available in the answer.
-1 - transmission error
-2 - not ready – wait.
This happens if the sampling of data has not finished till now either because ofa big spread factor or because the trigger condition has not been valid till now.

Example:

BOT CMD PAY CRC EOT
Request: 0x11 0x06 0x00 0x12 0x13

BOT CMD VALUE[0] VALUE[1] VALUE[2] VALUE[3] VALUE[4] VALUE[5]
Response: 0x11 0x06 0x9d 0x24 0x90 0x20 0x62 0x1b 0x99 0x24 0xd4 0x20 0x66 0x1b
VALUE[6] VALUE[7] VALUE[8] VALUE[9] VALUE[10] VALUE[11] VALUE[12]
0x94 0x24 0x19 0x21 0x6b 0x1b 0x8f 0x24 0x5d 0x21 0x70 0x1b 0x88 0x24
VALUE[13] VALUE[14] VALUE[15] CRC EOT
0xa1 0x21 0x77 0x1b 0x7f 0x24 0x88 0x13

If (as in the former example Va/Vb/Vc) three signals are selected the values 0,3,6,9,12,15 belong to the first selected value.
1,4,7 … to the second …. and so on.

4.2.8. CMD: (0x07) RPC_ConfigureHsDSO

This command configures the HsDSO.

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;           /* Motor number */
uint8_t enable;                     /* enable the HsDSO */
uint32_t selected_values;   /* Bitfield with DSO_Selection enumerated values
                                    for selecting the signals to be logged:
                                     Va = (1<< 0), a-phase voltage
                                     Vb = (1<< 1), b-phase voltage
                                     Vc = (1<< 2), c-phase voltage
                                     Valpha = (1<< 3), alpha-axis voltage
                                     Vbeta = (1<< 4), beta-axis voltage
                                     Id = (1<< 5), d-axis current
                                     Id_Ref = (1<< 6), d-axis reference current
                                     Iq = (1<< 7), q-axis current
                                     Iq_Ref = (1<< 8), q-axis reference current
                                     Vd = (1<< 9), d-axis voltage
                                     Vq = (1<<10), q-axis voltage
                                     Vdi = (1<<11), d-axis integral term
                                     Vqi = (1<<12), q-axis integral term
                                     Theta = (1<<13), Phase 0
                                     Omega = (1<<14), Rotation speed
                                     Omega_Calc = (1<<15), Calculated Rotation speed
                                     SIN_Theta = (1<<16), Sine Value at 0
                                     COS_Theta = (1<<17), Cosine Value at 0
                                     Sector = (1<<18), Sector information
                                     VDC = (1<<19), Motor supply voltage
                                     Ia = (1<<20), a-phase current
                                     Ib = (1<<21), b-phase current
                                     Ic = (1<<22), c-phase current
                                     Ialpha = (1<<23), alpha-axis current
                                     Ibeta = (1<<24), beta-axis current
                                     VE_Stage = (1<<25), stage of Vector Engine
                                     User 1 = (1<<26), User Variable
                                     User 2 = (1<<27), User Variable
                                     User 3 = (1<<28), User Variable
                                     User 4 = (1<<29), User Variable
                                     User 5 = (1<<30), User Variable
                                     User 6 = (1<<31), User Variable
uint8_t spread_factor;              /* Factor of which the logging is spread */
```

Response Payload Data:

```
uint8_t dummy;              /* Dummy value - due to protocol specification */
```

The HsDSO will transmit the recorded signals on a second serial channel using the FTDI C232HDEDHSP-0 (USB to UART Serial Cable).
The configuration for the FTDI Adapter is: **5 MBit/s, 8N1, NO flow control**.

The serial transmission order is the following:

Byte 0:	<BOT> (0x11)
Byte 1:	CRC8 over Bytes 2 till ByteX - CRC starting from 0x00
Byte 2:	Nr_of_missed_values
Byte 3:	timestamp (lower-part)

Byte 4: timestamp (upper-part)
Byte 5 –
Byte X: 16 Bit values of capture in the order of selection

X = (3+(number_of_selected_values-nr_of_missed_values)*2)

Example:

BOT CMD NR ENA SELECTED SPRE CRC EOT
Request: 0x11 0x07 0x01 0x01 0x07 0x00 0x00 0x00 0x00 0xcc 0x13

BOT CMD DMY CRC EOT
Response : 0x11 0x07 0x00 0x15 0x13

Request:
- Selected: 0x00000007 -> Va/Vb/Vc
- Spread: 0x00 -> log every value

4.2.9. CMD: (0x08) RPC_DoTurn

This command starts the Turn Control task.

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;       /* Motor number */
int16_t turns;          /* Number of turns to perform */
uint16_t max_speed;     /* Maximum rpm while turning */
```

Response Payload Data:

```
uint8_t dummy;          /* Dummy value - due to protocol specification */
```

Example:

BOT CMD NR TURNS MAXSPEED CRC EOT
Request: 0x11 0x08 0x010xf4 0x01 0xe8 0x03 0xa9 0x13

BOT CMD PAY CRC EOT
Response: 0x11 0x08 0x00 0x38 0x13

Request:
- Turns: 0x01f4 -> 500
- Max.Speed: 0x03e8 -> 1000

4.2.10. CMD: (0x09) RPC_GetOneMotorParameter

This command reads out one parameter.

Payload Data:

```
struct get_entry_req
        uint8_t id;
        uint8_t offset;
struct get_entry_res
        int32_t value;
        int8_t unit;
```

Request Payload Data:

```
uint8_t motor_nr;           /* Motor number */
struct get_entry_q entry;   /* parameter request */
```

Response Payload Data:

```
struct get_entry_a entry;           /* parameter value */
```

Example:

	BOT CMD NR ID OFF CRC EOT
Request:	0x11 0x09 0x01 0xe1 0x00 0x07 0x13
	BOT CMD ENTRY CRC EOT
Response:	0x11 0x09 0x00 0x00 0x00 0x00 0x00 0x7b 0x13

Request:

- ID: 0xe1 -> MMS_ErrorState
- Offset: 0x00 -> 0

Response:

- Value: 0x00000000 -> 0 -> no error
- Unit: 0x0 -> 0 -> n/a

Description of ID, offset, value and unit follow later.

4.2.11. CMD: (0x0A) RPC_SetOneMotorParameter

This command sets one parameter in the firmware.

Payload Data:

```
struct set_entry
        uint8_t id;
        int8_t unit;
        int32_t value;
```

Request Payload Data:

```
uint8_t motor_nr;    /* Motor number */
struct set_entry entry;
```

Response Payload Data:

```
uint8_t dummy;              /* Dummy value - due to protocol
                               specification */
```

Example:

	BOT CMD NR ID UNIT VALUE CRC EOT
Request:	0x11 0x0a 0x01 0xe5 0x00 0xd0 0x07 0x00 0x00 0x7b 0x13
	BOT CMD PAY CRC EOT
Response:	0x11 0x0a 0x00 0x36 0x13

Request:

- ID: 0xe5 -> MMS_Speed
- Unit: 0x00 -> 0 -> not applicable
- Value: 0x000007d0 -> 2000 -> 2000 [rpm]

Description of ID, offset, value and unit follow later.

4.2.12. CMD: (0x0B) RPC_GetMotorParameters

This command reads out a set of parameters from the firmware.
The number of used parameters can determine by PAR: (84) MPC_ParameterTransferCount.
At the moment the count is set to 8 inside of the firmware.

Payload Data:

```
struct get_entry_q
        uint8_t id;
        uint8_t offset;
struct get_entry_a
        int32_t value;
        int8_t unit;
```

Request Payload Data:

```
uint8_t motor_nr;           /* Motor number */
uint8_t entries;            /* Number of entries */
struct get_entry_q entry[X];
```

Response Payload Data:

```
uint8_t error;
uint8_t offset;
struct get_entry_a entry[X];
```

Example:

BOT CMD NR ENT ID0 OFF0 ID1 OFF1 ID2 OFF2 ID3 OFF3 ID4 OFF4
Request: 0x11 0x0b 0x01 0x08 0x10 0x00 0x11 0x00 0x12 0x00 0x13 0x00 0x14 0x00
ID5 OFF5 ID6 OFF6 ID7 OFF7 CRC EOT
0x15 0x00 0x16 0x00 0x17 0x00 0xd0 0x13

BOT CMD ERR OFF VALUE0 UNT0 VALUE1 UNT1
Response: 0x11 0x0b 0x00 0x00 0x04 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x00
VALUE2 UNT2 VALUE3 UNT3 VALUE4
0x2c 0x01 0x00 0x00 0x00 0x22 0x00 0x00 0x00 0xfd 0xee 0x02 0x00 0x00
UNT4 VALUE5 UNT5 VALUE6 UNT6 VALUE7
0xfd 0x1a 0x04 0x00 0x00 0xfa 0x5e 0x01 0x00 0x00 0x00 0x19 0x00 0x00
UNT7 CRC EOT
0x00 0x00 0x61 0x13

Request:

- Entries: 0x08 -> 8 values will follow
- ID0: 0x10 -> MPC_PolePairs
- Offset0: 0x00 -> not used
- ID1: 0x11 -> MPC_Direction
- Offset1: 0x00 -> not used
- ID2: 0x12 -> MPC_MaxAngAcc
- Offset2: 0x00 -> not used
- ID3: 0x13 -> MPC_TorqueFactor
- Offset3: 0x00 -> not used
- ID4: 0x14 -> MPC_Resistance
- Offset4: 0x00 -> not used
- ID5: 0x15 -> MPC_Inductance
- Offset5: 0x00 -> not used
- ID6: 0x16 -> MPC_SpeedLimit
- Offset6: 0x00 -> not used
- ID7: 0x17 -> MPC_SpeedChange

- Offset7: 0x00 -> not used

Response:

- Error: 0x00 -> no error
- Offset: 0x00 -> not used
- Value0: 0x00000004 -> 4 -> 4 Polepairs
- Unit0: 0x00 -> 0 -> not applicable
- Value1: 0x00000002 -> 2 -> CW & CCW
- Unit1: 0x00 -> 0 -> not applicable
- Value2: 0x0000012c -> 300 -> 300
- Unit2: 0x00 -> 0 -> rad/s²
- Value3: 0x00000022 -> 34 -> 34
- Unit3: 0xfd -> -3 -> mNm/A
- Value4: 0x000002ee -> 750 -> 750
- Unit4: 0xfd -> -3 -> mΩ
- Value5: 0x0000041a -> 1050 -> 1050
- Unit5: 0xfa -> -6 -> µH
- Value6: 0x0000015e -> 350 -> 350
- Unit6: 0x00 -> 0 -> Hz
- Value7: 0x00000019 -> 25 -> 25
- Unit7: 0x00 -> 0 -> Hz

Description of ID, offset, value and unit follow later.

4.2.13. CMD: (0x0C) RPC_SetMotorParameters

This command writes a set of parameters to the firmware.
The number of used parameters can determine by PAR: (84) MPC_ParameterTransferCount.
At the moment the count is set to 8 inside of the firmware.

Payload Data:

```
struct set_entry
uint8_t id;
int8_t unit;
int32_t value;
```

Request Payload Data:

```
uint8_t motor_nr;          /* Motor number */
uint8_t entries;
struct set_entry entry[X];
Answer Payload Data:
uint8_t error;
uint8_t offset;
```

Response Payload Data:

```
uint8_t error;
uint8_t offset;
```

Example:

	BOT CMD NR ENTR ID0 UNT0 VALUE0 UNT1 VALUE1
Request:	0x11 0x0c 0x01 0x01 0x19 0xfd 0xd0 0x07 0x00 0x00 0x00 0x00 0x00 0x00
	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
	CRC EOT
	0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xfa 0x13

BOT CMD ERR OFF CRC EOT
Response: 0x11 0x0c 0x00 0x00 0xfc 0x13

Request:

- Entries: 0x01 -> One value to be transmitted
- ID0: 0x19 -> MPC_IqStart
- Unit: 0xfd -> -3 -> mA
- Value: 0x000007d0 -> 2000 -> 2000

Description of ID, offset, value and unit follow later.

4.2.14. CMD: (0x0D) RPC_DoLinearMotion

This command is to turn the motor to a new absolute position with a specific rotation speed, when motion control is enabled.

Payload Data:

Request Payload Data:

```
uint8_t   motor_nr;          /*!< Motor number: */
uint32_t abs_postion;        /*!< New absolute position in terms of encoder pulses */
uint16_t max_speed;          /*!< Maximum rpm while turning */
```

Response Payload Data:

```
uint8_t   error;             /*!< Error value   */
```

BOT CMD MOTOR_NR ABS_POS MAX SPEED CRC EOT
Request: 0x110x0d 0x00 0x64 0x00 0x00 0x00 0x64 0x00 0x08 0x13

BOT CMD ERR CRC EOT
Response: 0x11 0x0d 0x00 0x23 0x13

4.2.15. CMD: (0x0E) RPC_GetAbsolutePosition

This command is for getting the absolute position in number of encoder pulses of a motor, when used with motion control. But keep in mind that this is not the encoder value.

Payload Data:

Request Payload Data:

```
uint8_t   motor_nr;          /*!< Motor number: */
```

Response Payload Data:

```
uint32_t current_position;   /*!< Current absolute position in number of encoder pulses */
```

BOT CMD MOTOR_NR CRC EOT
Request: 0x11 0x0E 0x00 0x2A 0x13

BOT CMD CUR_POS CRC EOT
Response: 0x11 0x0E 0x00 0x00 0x00 0x00 0xc4 0x13

4.2.16. CMD: (0x0F) RPC_AbortLinearMotion

This command is to abort the linear motion for a specific motor, which means that the target speed for that motor is set to zero. It will stop the current command execution and allow new linear motion or other command to be processed. The vector engine is kept active and the motor is still powered.

Payload Data:

Request Payload Data:

```
uint8_t   motor_nr;          /*!< Motor number: */
```

Response Payload Data:

```
uint8_t   error;             /*!< Error value   */
```

	BOT CMD MOTOR_NR CRC EOT
Request:	0x11 0x0f 0x00 0x2d 0x13
	BOT CMD ERR CRC EOT
Response:	0x11 0x0f 0x00 0x2d 0x13

4.2.17. CMD : (0x10) RPC_StartMotorTorqueCtrl

This command starts the VE and the Motor with **Torque** control. Instead of controlling the motor with constant speed, this command is used to drive the motor maintaining constant torque. The speed parameter has to be set and indicates the maximal allowed speed. The motor can be stopped with command: CMD : (0x00) RPC_StopMotor

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;    /* Motor number */
```

Response Payload Data:

```
uint8_t dummy;               /* Dummy value - due to protocol specification */
```

Example:

	BOT CMD MOTOR_NR CRC EOT
Request:	0x11 0x10 0x01 0x77 0x13
	BOT CMD DMY CRC EOT
Response:	0x11 0x10 0x00 0x70 0x13

4.2.18. CMD: (0x11) RPC_GetMotorControlMethod

This command retrieves the currently used control method of the selected motor, either by speed or by torque. None is returned if the motor is idle.

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;    /* Motor number */
```

Response Payload Data:

```
uint8_t method;          /* Current Control method */
```

Example:

	BOT	CMD	MOTOR_NR	CRC	EOT
Request:	0x11	0x11	0x02	0x79	0x13

	BOT	CMD	STATE	CRC	EOT
Response:	0x11	0x11	0x01	0x70	0x13

Request:

- Motor channel: 0x2

Response:

- Status: 0x01 -> Motor is controlled by Speed

Possible control methods:
MOTORCONTROL_CONTROL_BY_SPEED = (1<<0) : 0x01 - Motor is controlled by Speed (default)
MOTORCONTROL_CONTROL_BY_TORQUE = (1<<1) : 0x02 - Motor is controlled by Torque

4.2.19. CMD: (0x12) RPC_SetMotorControlMethod

This command set the control method of the selected motor to one of by speed or by torque.

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;    /* Motor number */
uint8_t method;          /* Requested control method */
```

Response Payload Data:

```
uint8_t error;           /* Error value */
```

Example:

	BOT	CMD	MOTOR_NR	METHOD	CRC	EOT
Request:	0x11	0x12	0x02	0x02	0x59	0x13

	BOT	CMD	STATE	CRC	EOT
Response:	0x11	0x12	0x00	0x7E	0x13

Request:

- Motor channel: 0x2
- Control method: 0x2 -> Motor will be controlled by Torque

Response:

- Status: 0x00 -> New method set
  0x01 -> Invalid parameter

Possible control methods:
MOTORCONTROL_CONTROL_BY_SPEED = (1<<0) : 0x01 - Motor is controlled by Speed (default)
MOTORCONTROL_CONTROL_BY_TORQUE = (1<<1) : 0x02 - Motor is controlled by Torque

4.2.20. CMD: (0x13) RPC_GetEncoderCounter

This command retrieves the current encoder value, which can be used for calculation of the motor relative position based on the encoder resolution and the revolution direction.

Payload Data:

Request Payload Data:

```
uint8_t encoder_nr;        /* Encoder number */
```

Response Payload Data:

```
int32_t counterValue;      /* Encoder Counter Value */
```

Example:

	BOT CMD ENCODER_NR CRC EOT
Request:	0x11 0x13 0x01 0x7E 0x13
	BOT CMD ENCODER_CTR_VAL CRC EOT
Response:	0x11 0x13 0x01 0x00 0x20 0xA9 0xB3 0x13

Request:

- Encoder channel: 0x1

Response:

- Counter value: 0x01

4.2.21. CMD: (0x14) RPC_GetFWVersion

This command is for downward compatibility with MotorMind 1.x versions.
MotorMind 1.x will not be able to work with the actual firmware, but will be able to give an appropriate error message.

Payload Data:

Request Payload Data:

```
uint8_t dummy;             /* Dummy value - due to protocol specification */
```

Response Payload Data:

```
uint8_t fw_version[2];     /* NONE - Version information [Major][Minor] */
```

4.2.22. CMD: (0x21) RPC_SetDemoState

This command is used to set state of a demo. It is only supported if the firmware has the demo feature enabled (FEATURE_HAS_DEMO). The state is represented by its integer ID, which is demo application specific.

Payload Data:

Request Payload Data:

```
uint8_t new_state;          /*!< New state */
```

Response Payload Data:

```
uint8_t   error;            /*!< Error value   */
```

Example:

	BOT CMD STATE CRC EOT
Request:	0x11 0x21 0x03 0xee 0x13
	BOT CMD ERR CRC EOT
Response:	0x11 0x21 0x00 0xe7 0x13

4.2.23. CMD: (0x22) RPC_GetDemoState

This command is used to retrieve the current state of a demo. It is only supported if the firmware has the demo feature enabled (FEATURE_HAS_DEMO). The state is represented by its integer ID, which is demo application specific.

Payload Data:

Request Payload Data:

```
uint8_t dummy;          /* Dummy value - due to protocol specification */
```

Response Payload Data:

```
uint8_t curr_state;     /* Current State */
```

Example:

	BOT CMD DMY CRC EOT
Request:	0x11 0x22 0xab 0xb6 0x13
	BOT CMD STATE CRC EOT
Response:	0x11 0x22 0x03 0xe7 0x13

4.2.24. CMD : (0x23) RPC_GetExtendedMotorState

This command gets the actual extended status of the motor. It is replacing the legacy RPC_GetMotorState, which is kept for backward compatibility.

Payload Data:

Request Payload Data:

```
uint8_t motor_nr;                    /* Motor number */
```

Response Payload Data:

```
int32_t   ActualControlledParameter;   /*!< [RPM] or [Ncm]   - Actual Rotation Speed or Torque */
int32_t   TargetControlledParameter;   /*!< [RPM] or [Ncm]   - Target Rotation Speed of Torque */
uint32_t  Current;                     /*!< [mA] - Current used to drive the motor */
uint32_t  NonControlParameter;         /*!< [Ncm] or [RPM] - Current Torque or Speed of the Motor */
uint32_t  Timestamp;                   /*!< [systicks] - Systick counter of system */
uint8_t   ControlMethod;               /*!< [NONE] - Idle, Control by Speed, Control by Torque */
```

Example:

	BOT CMD MOTOR_NR CRC EOT
Request:	0x11 0x23 0x00 0xE9 0x13
	BOT CMD ACTUAL SPEED TARGET SPEED CURRENT
Response:	0x11 0x23 0x00 0x00 0x01 0xf2 0x00 0x00 0x01 0xf4 0x00 0x00 0x02 0x8a
	TORQUE TIMESTAMP CONTROL_METHOD CRC EOT
	0x00 0x00 0x00 0xcb 0x00 0x00 0x25 0xd1 0x01 0x87 0x13

- Actual Speed: 0x000001F2 = 498 [RPM]
- Target Speed: 0x000001F4 = 500 [RPM]
- Current : 0x0000028A = 650 [mA]
- Torque : 0x000000CB = 203 [Ncm]
- Control method: 0x01 = Idle (no control)
- Timestamp : 0x000025d1 = 9681 [systicks]

5. Parameter Description

5.1. Supported Parameters

The MCU firmware supports a various parameter for configurations of the system and the motor.
These parameters are categorized as follows:

- Motor Parameters
- Encoder Parameters
- PI Control Parameter
- System Settings
- Board Settings

Most of the parameter values are read-writeable. If some are read or write only this will be explicitly stated in the description.

The Board configuration values are read only.

The System Settings, including a list of currently supported set of Firmware features, and the current Board(s) configuration are channel independent. These can be retrieved via the RPM_GetOneParameter command with MotorID parameter set to of -1 (0xFF).

5.1.1. PAR: (0x10) MPC_PolePairs

Pole pairs of the motor.

Value:
1 – 100 /* Number of pole pairs of the motor */

Unit:
Not applicable

5.1.2. PAR: (0x11) MPC_Direction

Turning direction of the motor.

Value:
```
MOTOR_CW_ONLY    = 0,   /* Motor can turn clock-wise only */
MOTOR_CCW_ONLY   = 1,   /* Motor can turn counter-clock-wise only */
MOTOR_CW_CCW     = 2,   /* Motor can turn CW and CCW */
```

Unit:
Not applicable

5.1.3. PAR: (0x12) MPC_MaxAngAcc

Maximum angular acceleration.

Value:
1 – 10000

Unit:
```
RAD_BY_S2        = 0,   /* acceleration is given as rad/s² */
HZ_BY_S          = 1,   /* acceleration is given as Hz/s    */
RPM_BY_S         = 2,   /* acceleration is given as RPM/s   */
RPM_BY_MIN       = 3,   /* acceleration is given as RMP/min */
```

5.1.4. PAR: (0x13) MPC_TorqueFactor

Torque factor of the motor – only used for the statistics graphs.

Value:
0 - 10000

Unit:

MNM_BY_A	= -3,	/* torque factor given as mNm/A */
NM_BY_A	= 0,	/* torque factor given as Nm/A */
MNM_BY_MA	= 0,	/* torque factor given as mNm/mA */

5.1.5. PAR: (0x14) MPC_Resistance

Resistance of a single phase.

Value:
0 - 10000

Unit:

OHM	= 0,	/* resistance given as Ω */
MOHM	= -3,	/* resistance given as mΩ */
UOHM	= -6,	/* resistance given as uΩ */

5.1.6. PAR: (0x15) MPC_Inductance

Inductance of a single phase.

Value:
0 - 10000

Unit:

MH	= -3,	/* inductance given in mH */
UH	= -6,	/* inductance given in uH */
NH	= -9,	/* inductance given in nH */

5.1.7. PAR: (0x16) MPC_SpeedLimit

Maximum speed of the motor.

Value:
1 - 200000

Unit:

HZ	= 0,	/* Speed value is given in Hz */
RPM	= 1,	/* Speed value is given in RPM */

5.1.8. PAR: (0x17) MPC_SpeedChange

Speed changing from forced mode to field oriented mode.

Value:
1 – SpeedLimit+1

Unit:

HZ	= 0,	/* Speed value is given in Hz */
RPM	= 1,	/* Speed value is given in RPM */

5.1.9. PAR: (0x18) MPC_PositionDelay

Time for reaching the initial position before starting to turn.

Value:
0 – 10000 /* positioning delay */

Unit:
Not applicable [ms]

5.1.10. PAR: (0x19) MPC_IqStart

Iq start current.
Limited by Iq Limit.

Value:
1 - 10000

Unit:
AMPERE = 0, /* current given in A */
MAMPERE = -3, /* current given in mA */

5.1.11. PAR: (0x1A) MPC_IdStart

Id start current.
Limited by Id Limit.

Value:
1 - 10000

Unit:
AMPERE = 0, /* current given in A */
MAMPERE = -3, /* current given in mA */

5.1.12. PAR: (0x1B) MPC_IqLimit

Iq limit current.
Maximum current allowed in FOC Mode.

Value:
1 - 10000

Unit:
AMPERE = 0, /* current given in A */
MAMPERE = -3, /* current given in mA */

5.1.13. PAR: (0x1C) MPC_IdLimit

Id limit current.
Maximum current allowed in FOC Mode.

Value:
1 – 10000

Unit:
AMPERE = 0, /* current given in A */
MAMPERE = -3, /* current given in mA */

5.1.14. PAR: (0x1D) MPC_MotorId

Name of the motor for identification.

Value:
unit8_t name[20] /* character field of the motor name */

Unit:
Not applicable

5.1.15. PAR: (0x30) MPC_Encoder

Type of encoder.

Value:

ENCODER_NO_ENCODER	= 0,	/* Motor has NO Encoder - use SW emulation for event counting */
ENCODER_HALL_UVW	= 1,	/* Motor is using 3 Hall Sensors Phase U,V,W*/
ENCODER_HALL_UV	= 2,	/* Motor is using 2 Hall Sensors Phase U & V*/
ENCODER_INCENC_ABZ	= 3,	/* Motor is using encoder inputs A,B,Z */
ENCODER_INCENC_AB	= 4,	/* Motor is using encoder inputs A,B */
ENCODER_SINGLE_PULSE	= 5,	/* Motor is using single pulse */
ENCODER_AMS_AS5145H	= 6,	/* Motor is using AMS - AS5145H */

Unit:
Not applicable

5.1.16. PAR: (0x31) MPC_EncoderUsage

The way the encoder will be used.

Value:

ENCODER_USAGE_NONE	=0,	/* Encoder is not used */
ENCODER_USAGE_SPEED	=1,	/* Encoder is used for speed control */
ENCODER_USAGE_EVENT	=2,	/* Encoder is used for event counting */

Unit:
Not applicable

5.1.17. PAR: (0x32) MPC_EncoderResolution

Resolution of an incremental encoder.
In case of Hall Sensors this value will be calculated internally.

Value:
0 – 20000 /* Incremental encoder resolution by revelation */

Unit:
Not applicable

5.1.18. PAR: (0x33) MPC_ EncoderMinPos

Minimal allowed absolute position in the linear motion control in terms of encoder pulses.

Value:
0 – 4294967295 /* Minimal pulses count */

Unit:
Not applicable

5.1.19. PAR: (0x34) MPC_ EncoderStartPos

Absolute start position in the linear motion control in terms of encoder pulses. Will be correlated to the current value of the line encoder

Value:
0 – 4294967295 /* Start pulses count */

Unit:
Not applicable

5.1.20. PAR: (0x35) MPC_ EncoderMaxPos

Maximal allowed absolute position in the linear motion control in terms of encoder pulses.

Value:
0 – 4294967295 /* Maximal pulses count */

Unit:
Not applicable

5.1.21. PAR: (0x36) MPC_EncoderLMApproach

Absolute approach speed in the linear motion control.

Value:
0 – 65535 /* Speed for approaching the final position in RPM */

Unit:
RPM

5.1.22. PAR: (0x37) MPC_ EncoderGearFactor

Reducing factor of the gear attached to the motor, e.g. 1:xxx.

Value:
0 – 65535 /* Reducing gear factor */

Unit:
Not applicable

5.1.23. PAR: (0x40) MPC_PIdKi

PI current controller – integral constant of Id.

Value:
0 – 65535 /* PI integral constant Id */

Unit:
Not applicable [V/As]

5.1.24. PAR: (0x41) MPC_PIdKp

PI current controller – proportional constant of Id.

Value:
0 – 65535 /* PI proportional constant Id */

Unit:
Not applicable [V/A]

5.1.25. PAR: (0x42) MPC_PiIqKi

PI current controller – integral constant of Iq.

Value:
0 – 65535 /* PI integral constant Iq */

Unit:
Not applicable [V/As]

5.1.26. PAR: (0x43) MPC_PiIqKp

PI current controller – proportional constant of Iq.

Value:
0 – 65535 /* PI proportional constant Id */

Unit:
Not applicable [V/A]

5.1.27. PAR: (0x44) MPC_PositionKi

PI position estimator – integral constant of the position estimator.

Value:
0 – 65535 /* Position Estimator integral constant */

Unit:
Not applicable [Hz/Vs]

5.1.28. PAR: (0x45) MPC_PositionKp

PI position estimator – proportional constant of the position estimator.

Value:
0 – 65535 /* Position Estimator proportional constant */

Unit:
Not applicable [Hz/V]

5.1.29. PAR: (0x46) MPC_SpeedKi

PI speed controller – integral constant of the speed controller.

Value:
0 – 65535 /* Speed controller integral constant */

Unit:
Not applicable [mA/Hz*s]

5.1.30. PAR: (0x47) MPC_SpeedKp

PI speed controller – proportional constant of the speed controller.

Value:
0 – 65535 /* Speed controller proportional constant */

Unit:
Not applicable [mA/Hz]

5.1.31. PAR: (0x50) MPC_PwmFrequency

PWM Frequency of the system.

Value:
0 – 50000 /* PWM frequency */

Unit:
Not applicable [Hz]

5.1.32. PAR: (0x51) MPC_ShutdownMode

The way the motor will be shutdown.

Value:
SHUTDOWN_DISABLE_OUTPUT = 0, /* Shutdown motor by switching off signals */
SHUTDOWN_SHORT_BRAKE = 1, /* Shutdown motor by short lines */
SHUTDOWN_GENTLE = 2, /* Shutdown motor by speeding down to 0 */

Unit:
Not applicable

5.1.33. PAR: (0x52) MPC_Braketime

When selected SHUTDOWN_SHORT_BRAKE the time of executing the brake can be selected using this parameter.

Value:
0 – 10000 /* Time for braking before entering stop mode*/

Unit:
Not applicable [ms]

5.1.34. PAR: (0x53) MPC_Brakepercentage

When selected SHUTDOWN_SHORT_BRAKE as Shutdown Mode this value gives the percentage of time the drivers are short cut.

Value:
0 – 100 /* Percentage of short brake time when using Short brake mode */

Unit:
Not applicable [%]

5.1.35. PAR: (0x54) MPC_RestartMode

When a field stall is detected this parameter selects how the system will react on this.

Value:
SWITCH_OFF_MOTOR = 0, /* Switch off motor */
RESTART_MOTOR = 1, /* Restart motor */

Unit:
Not applicable

5.1.36. PAR: (0x55) MPC_StalldetectValue

The value needed to detect a stalling field when changing from Forced to FOC Mode.

Value:

0 – 4095	/* Value for stall detection routine */

Unit:

Not applicable	[mV/s]

5.1.37. PAR: (0x56) MPC_Overtemperature

When reaching this temperature value the motor will be switched off.

Value:

-40 – 150	/* Value for over temperature detection */

Unit:

Not applicable	[ºC]

5.1.38. PAR: (0x57) MPC_CanId

When using the CAN protocol this value will be used as the CAN ID of the system.

Value:

0 – 15	/* CAN ID used for CAN Communication Protocol */

Unit:

Not applicable

5.1.39. PAR: (0x58) MPC_ExternalSpeedControl

When using external speed control in the firmware this value will select in which way the external speed control will be used.

Value:

EXTERNAL_SPEED_NONE	= 0,	/*No External Speed Control */
EXTERNAL_SPEED_ADC	= 1,	/*External Speed Control via ADC */
EXTERNAL_SPEED_PWM	= 2,	/*External Speed Control via PWM */
EXTERNAL_SPEED_SERVO_NEGMAX_TO_MAX	= 3,	/*External Speed Control via Servo Signal */
EXTERNAL_SPEED_SERVO_0_TO_MAX	= 4,	/*External Speed Control via Servo Signal */

Unit:

Not applicable

5.1.40. PAR: (0x59) MPC_SwOvervoltage

When VDC is exceeding this value the motor will be switched off.

Value:

0 – 65535	/* SW over voltage detect value */

Unit:

Not applicable	[V]

5.1.41. PAR: (0x5a) MPC_SwUndervoltage

When VDC is going below this value the motor will be switched off.

Value:
0 – 65535 /* SW under voltage detect value */

Unit:
Not applicable [V]

5.1.42. PAR: (0x5B) MPC_SwOvercurrent

If the measured (SW) current exceeding this value the motor will be switched off.

Value:
0 – 65535 /* SW over current detect value */

Unit:
Not applicable [mA]

5.1.43. PAR: (0x5C) MPC_SpeedReduction

Then the current exceeds this percentage of the Iq Limit value the target speed of the motor is reduced to keep the current below this percentage of Iq Limit.
Then the load disappear and the current lower again the original target speed is restored.

Value:
0 – 100 /* Percentage of Limit Current to use Speed Reduction */

Unit:
Not applicable [%]

5.1.44. PAR: (0x70) MPC_Deadtime

Dead Time of the FETs / IGBTs – in steps of 100 ns.

Value:
0 – 5000 /* Deadtime of the drivers */

Unit:
Not applicable [ns]

5.1.45. PAR: (0x71) MPC_BootstrapDelay

Time the lower side is activated to load the bootstrap circuit.

Value:
0 – 1000 /* Time for bootstrap */

Unit:
Not applicable [ms]

5.1.46. PAR: (0x72) MPC_SensitivityCurrentMeasure

Sensitivity of the current measurement.

Value:
0 – 10000 /* Sensitivity of current measurement */

Unit:
MV_BY_A = -3, /* mV/A */

UV_BY_A = -6, /* µV/A */

5.1.47. PAR: (0x73) MPC_SensitivityVoltageMeasure

Sensitivity of the voltage measurement.

Value:
0 – 10000 /* Sensitivity of voltage measurement */

Unit:
MV_BY_V = -3, /* mV/V */

5.1.48. PAR: (0x74) MPC_GainCurrentMeasure

Value:

For M374Baseboard:
0: Gain 1
1: Gain 2
2: Gain 4
3: Gain 5
4: Gain 8
5: Gain 10
6: Gain 16
7: Gain 32

For M37Sigma
0: Gain 1

For Hitex M370:
0: Gain 1.5
1: Gain 2.5
2: Gain 3
3: Gain 3.5
4: Gain 4
5: Gain 6
6: Gain 8
7: Gain 10

Unit:
Not applicable [ms]

5.1.49. PAR: (0x75) MPC_MeasurementType

Measurement type of the current.

Value:
CURRENT_SHUNT_1 = 1, /* Channel is using 1-shunt solution */
CURRENT_SHUNT_3 = 3, /* Channel is using 3-shunt solution */
CURRENT_SENSOR_2= 2, /* Channel is using 2 sensor solution */

Unit:
Not applicable

5.1.50. PAR: (0x76) MPC_SensorDirection

When using current sensors for measurement the orientation can be inverted by software.

Value:

```
CURRENT_SENSOR_NORMAL= 0,       /* Channel is using normal orientation */
CURRENT_SENSOR_INVERTED=1,      /* Channel is using inverted orientation */
```

Unit:

Not applicable

5.1.51. PAR: (0x77) MPC_PolL

Polarisation of the Low-Side FETs / IGBTs.

Value:

```
POLARISATION_LOW = 0,       /* Driving the FETs by low signal */
POLARISATION_HIGH= 1,       /* Driving the FETs by high signal */
```

Unit:

Not applicable

5.1.52. PAR: (0x78) MPC_PolH

Polarisation of the High-Side FETs / IGBTs.

Value:

```
POLARISATION_LOW = 0,       /* Driving the FETs by low signal */
POLARISATION_HIGH= 1,       /* Driving the FETs by high signal */
```

Unit:

Not applicable

5.1.53. PAR: (0x90) MPC_FirmwareFeatures

Information about compiled in features of the firmware.
This parameter is read only.
Please use Motor ID = -1.

Value:

```
FEATURE_HAS_DSO                                 = (1<< 0),
FEATURE_HAS_LOAD_STATISTIC                      = (1<< 1),
FEATURE_HAS_TURN_CONTROL                        = (1<< 2),
FEATURE_HAS_TEMPERATURE_CONTROL                 = (1<< 3),
FEATURE_HAS_HSDSO                               = (1<< 4),
FEATURE_HAS_RW_BOARD_SETTINGS                   = (1<< 5),
FEATURE_HAS_EXTERNAL_SPEED_CONTROL              = (1<< 6),
FEATURE_HAS_SW_OVER_UNDER_VOLTAGE_DETECTION = (1<< 7),
FEATURE_HAS_MOTOR_DISCONNECT_DETECTION      = (1<< 8),
FEATURE_HAS_SW_OVERCURRENT_DETECTION        = (1<< 9),
FEATURE_HAS_LOAD_DEPENDANT_SPEED_REDUCTION = (1<<10),
FEATURE_HAS_CAN                                 = (1<<11),
FEATURE_HAS_STALL_DETECT                        = (1<<12),
FEATURE_HAS_MOTION_CONTROL                       = (1<<13),
FEATURE_HAS_GET_DC_LINK_VOLTAGE                 = (1<<14),
FEATURE_HAS_GET_MOTOR_STAGE                     = (1<<15),
FEATURE_HAS_DEMO                                = (1<<16),
FEATURE_HAS_TORQUE_CONTROL                      = (1<<17),
```

Unit:

Not applicable

5.1.54. PAR: (0x91) MPC_BoardRevision

Information about the board revision.
This parameter is read only.
Please use Motor ID = -1.

Value:
0-255 /* HW Revision of the Board */

Unit:
Not applicable

5.1.55. PAR: (0x92) MPC_Channels

Information about the available motor channels.
This parameter is read only.
Please use Motor ID = -1.

Value:
VE_CHANNEL_0 = (1<<0), /* Vector Engine Channel 0 is available */
VE_CHANNEL_1 = (1<<1), /* Vector Engine Channel 1 is available */
VE_CHANNEL_2 = (1<<2), /* Vector Engine Channel 2 is available */

Unit:
Not applicable

5.1.56. PAR: (0x93) MPC_DsoSize

Information about the available DSO storage size.
This parameter is read only.
Please use Motor ID = -1.

Value:
0-65535 /* Size of DSO storage area */

Unit:
Not applicable

5.1.57. PAR: (0x94) MPC_ParameterTransferCount

Information about the number of parameters used by:
CMD: (0x0b) RPC_GetMotorParameters
CMD: (0x0c) RPC_SetMotorParameters
This parameter is read only.
Please use Motor ID = -1.

Value:
1-255 /* Number of max parameters for RPC_ Get/Set MotorParameters */

Unit:
Not applicable

5.1.58. PAR: (0x95) MPC_BoardName

Name of the CPU board.
This parameter is read only.
Please use Motor ID = -1.

Value:
unit8_t name[20] /* character field of the board name */

Unit:
Not applicable

5.1.59. PAR: (0x96) MPC_BoardNamePWR0

Name of the Power board on channel 0.
This parameter is read only.
Please use Motor ID = -1.

Value:
unit8_t name[20] /*character field of the power board name on channel 0 */

Unit:
Not applicable

5.1.60. PAR: (0x97) MPC_BoardNamePWR1

Name of the Power board on channel 1.
This parameter is read only.
Please use Motor ID = -1.

Value:
unit8_t name[20] /* character field of the power board name on channel 1 */

Unit:
Not applicable

5.1.61. PAR: (0x98) MPC_BoardNamePWR2

Name of the Power board on channel 2 if supported.
This parameter is read only.
Please use Motor ID = -1.

Value:
unit8_t name[20] /* character field of the power board name on channel 2 */

Unit:
Not applicable

5.1.62. PAR: (0xB0) MSS_FirmwareVersion

Version information of the firmware.
This parameter is read only.
Please use Motor ID = -1.

Value:
uint8_t byte[4] /* Byte[1] – Major Byte[0] - Minor */

Unit:
Not applicable

5.1.63. PAR: (0xB1) MSS_ResetStatus

Reset status of the board.
After a reset this parameter can be read out as 1 for one time.
This indicated that the board has been reset since the last request.
This indicates that e.g. the motor parameters might not be valid anymore as they have been exchanged with the ones compiled in or set in the EEProm.
This parameter is read only.
Please use Motor ID = -1.

Value:
0/1 /* indicator that MCU has come out of reset */

Unit:
Not applicable

5.1.64. PAR: (0xb2) MSS_SystemLoad

System Load percentage.
This parameter is read only.
Please use Motor ID = -1.

Value:
0-100 /* CPU load percentage */

Unit:
Not applicable [%]

5.1.65. PAR: (0xe0) MMS_Temperature

Temperature of the temperature sensor.
This parameter is read only.
Please use Motor ID = -1.

Value:
-40-150 /* Temperature of the temperature sensor */

Unit:
Not applicable [ºC]

5.1.66. PAR: (0xe1) MMS_ErrorState

Error state of the motor.
This parameter is read only.

Value:

```
VE_ERROR_NONE          = 0,       /* No error */
VE_OVERTEMPERATURE     =(1<<0), /* Over temperature detected */
VE_EMERGENCY           =(1<<1), /* Emergency Signal Detected */
VE_OVERVOLTAGE         =(1<<2), /* Overvoltage Signal Detected */
VE_NOMOTOR             =(1<<3), /* Motor Disconnection detected */
VE_SWUNDERVOLTAGE      =(1<<4), /* Undervoltage detected (SW) */
VE_SWOVERVOLTAGE       =(1<<5), /* Overvoltage   detected (SW) */
VE_SWOVERCURRENT       =(1<<6), /* Overcurrent   detected (SW) */
VE_WRONGDIRECTION      =(1<<7), /* Wrong Motor direction requested */
VE_SPEEDREDUCTION      =(1<<8), /* Target speed reduced due to load */
VE_STALLDETECTED       =(1<<9), /* Field stall detected */
```

Unit:
Not applicable

5.1.67. PAR: (0xe2) MMS_DcLinkVoltage

DC Link voltage of the motor in 0.1V resolution. A typical DC voltage of 24V will be encoded as 240 and so on.
This parameter is read only.

Value:

0-65535 /* Measured DC Link voltage with 0.1V resolution */

Unit:
Not applicable [V]

5.1.68. PAR: (0xe3) MMS_MotorStage

Actual stage of the motor.
This parameter is read only.

Value:

Stage_Stop	= 0,	/* control loop is off, no actions pending */
Stage_Bootstrap	= 1,	/* bootstrap IGBTs */
Stage_Brake	= 2,	/* shortbrake Motor */
Stage_ZeroCurrentMeasure	= 3,	/* measure zero current adc value */
Stage_Initposition	= 4,	/* control loop is off, first signals out and in */
Stage_Force	= 5,	/* forced commutation */
Stage_FOC	= 6,	/* Field Oriented Commutation */
Stage_Emergency	= 7,	/* emergency state */

Unit:
Not applicable [V]

5.1.69. PAR: (0xE4) MMS_TurnNumber

Actual turn number of the motor.
This parameter is read only.

Value:
−2147483648- 2147483647 /* Actual turn number */

Unit:
Not applicable

5.1.70. PAR: (0xe5) MMS_Speed

Set the target speed of the motor.
This parameter is write only.

Value:
-SpeedLimit - +SpeedLimit /* Rotation speed of the Motor */

Unit:
Not applicable [rpm]

5.1.71. PAR: (0xe6) MMS_Torque

Set the torque of the motor.
This parameter is write only.

Value:
0 - TorqueLimit /* Torque of the Motor */

Unit:
Not applicable [mNm]

6. CRC8 Calculation

Bytefield for CRC8 calculation

```
uint8_table[256] ={
   0x00,  0x07,  0x0e,  0x09,  0x1c,  0x1b,  0x12,  0x15,  0x38,  0x3f,  0x36,  0x31,  0x24,
0x23,  0x2a,  0x2d,
   0x70,  0x77,  0x7e,  0x79,  0x6c,  0x6b,  0x62,  0x65,  0x48,  0x4f,  0x46,  0x41,  0x54,
0x53,  0x5a,  0x5d,
   0xe0,  0xe7,  0xee,  0xe9,  0xfc,  0xfb,  0xf2,  0xf5,  0xd8,  0xdf,  0xd6,  0xd1,  0xc4,
0xc3,  0xca,  0xcd,
   0x90,  0x97,  0x9e,  0x99,  0x8c,  0x8b,  0x82,  0x85,  0xa8,  0xaf,  0xa6,  0xa1,  0xb4,
0xb3,  0xba,  0xbd,
   0xc7,  0xc0,  0xc9,  0xce,  0xdb,  0xdc,  0xd5,  0xd2,  0xff,  0xf8,  0xf1,  0xf6,  0xe3,
0xe4,  0xed,  0xea,
   0xb7,  0xb0,  0xb9,  0xbe,  0xab,  0xac,  0xa5,  0xa2,  0x8f,  0x88,  0x81,  0x86,  0x93,
0x94,  0x9d,  0x9a,
   0x27,  0x20,  0x29,  0x2e,  0x3b,  0x3c,  0x35,  0x32,  0x1f,  0x18,  0x11,  0x16,  0x03,
0x04,  0x0d,  0x0a,
   0x57,  0x50,  0x59,  0x5e,  0x4b,  0x4c,  0x45,  0x42,  0x6f,  0x68,  0x61,  0x66,  0x73,
0x74,  0x7d,  0x7a,
   0x89,  0x8e,  0x87,  0x80,  0x95,  0x92,  0x9b,  0x9c,  0xb1,  0xb6,  0xbf,  0xb8,  0xad,
0xaa,  0xa3,  0xa4,
   0xf9,  0xfe,  0xf7,  0xf0,  0xe5,  0xe2,  0xeb,  0xec,  0xc1,  0xc6,  0xcf,  0xc8,  0xdd,
0xda,  0xd3,  0xd4,
   0x69,  0x6e,  0x67,  0x60,  0x75,  0x72,  0x7b,  0x7c,  0x51,  0x56,  0x5f,  0x58,  0x4d,
0x4a,  0x43,  0x44,
   0x19,  0x1e,  0x17,  0x10,  0x05,  0x02,  0x0b,  0x0c,  0x21,  0x26,  0x2f,  0x28,  0x3d,
0x3a,  0x33,  0x34,
   0x4e,  0x49,  0x40,  0x47,  0x52,  0x55,  0x5c,  0x5b,  0x76,  0x71,  0x78,  0x7f,  0x6a,
0x6d,  0x64,  0x63,
   0x3e,  0x39,  0x30,  0x37,  0x22,  0x25,  0x2c,  0x2b,  0x06,  0x01,  0x08,  0x0f,  0x1a,
0x1d,  0x14,  0x13,
   0xae,  0xa9,  0xa0,  0xa7,  0xb2,  0xb5,  0xbc,  0xbb,  0x96,  0x91,  0x98,  0x9f,  0x8a,
0x8d,  0x84,  0x83,
   0xde,  0xd9,  0xd0,  0xd7,  0xc2,  0xc5,  0xcc,  0xcb,  0xe6,  0xe1,  0xe8,  0xef,  0xfa,
0xfd,  0xf4,  0xf3 };
```

This crc8 has been generated by pycrc v0.7.8 – the start value for the polynomial is 0x00:

```
 * Generated on Tue Jul 12 15:41:04 2011,
 * by pycrc v0.7.8, http://www.tty1.net/pycrc/
 * using the configuration:
 * Width         = 8
 * Poly          = 0x07
 * XorIn         = 0x00
 * ReflectIn     = False
 * XorOut        = 0x00
 * ReflectOut    = False
 * Algorithm     = table-driven
```

The CRC is calculated by using the command value as starting value and calculating over the payload.

7. References

[1] “TMPM4K Group (1) Datasheet”, Revision 3.1, 07/2018, Toshiba Electronic Devices & Storage Corporation

[2] “Toshiba Motor Studio Firmware User’s Manual, Revision 1.0.0, April 2022, Toshiba Electronic Devices & Storage Corporation.

8. Revision History

Revision	Date	Changes
1.0.0	2022/04/29	Baselined Version
1.1.0	2022/10/21	Introduction section updated

Table 8-1: Revision History

Trademarks

- FreeRTOS™ is a trademark of Amazon Web Services, inc in the US and/or elsewhere. All rights reserved.
- Microsoft® and Windows® are either registered trademarks Microsoft Corporation in the United States and/or elsewhere. All rights reserved.
- Arm® , Cortex® ,Cortex®-M3, Cortex®-M4,Keil® and µVision® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
- Click boards™ is a trademark of MIKROELEKTRONIKA. All rights reserved.
- FTDI may be registered trademarks of “Future Technology Devices International Limited”. All rights reserved.
- IAR Systems® and IAR Embedded Workbench® are registered trademarks are owned by IAR Systems. All rights reserved
- SEGGER and J-Link are trademarks or registered trademarks of SEGGER Microcontroller GmbH & Co. KG. All rights reserved.

Other Company names, product names and service names mentioned herein may be trademarks of their respective companies.

RESTRICTIONS ON PRODUCT USE

Toshiba Corporation and its subsidiaries and affiliates are collectively referred to as “TOSHIBA”.
Hardware, software and systems described in this document are collectively referred to as “Product”.

- TOSHIBA reserves the right to make changes to the information in this document and related Product without notice.
- This document and any information herein may not be reproduced without prior written permission from TOSHIBA. Even with TOSHIBA's written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though TOSHIBA works continually to improve Product's quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant TOSHIBA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the "TOSHIBA Semiconductor Reliability Handbook" and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **TOSHIBA ASSUMES NO LIABILITY FOR CUSTOMERS' PRODUCT DESIGN OR APPLICATIONS.**
- **PRODUCT IS NEITHER INTENDED NOR WARRANTED FOR USE IN EQUIPMENTS OR SYSTEMS THAT REQUIRE EXTRAORDINARILY HIGH LEVELS OF QUALITY AND/OR RELIABILITY, AND/OR A MALFUNCTION OR FAILURE OF WHICH MAY CAUSE LOSS OF HUMAN LIFE, BODILY INJURY, SERIOUS PROPERTY DAMAGE AND/OR SERIOUS PUBLIC IMPACT ("UNINTENDED USE").** Except for specific applications as expressly stated in this document, Unintended Use includes, without limitation, equipment used in nuclear facilities, equipment used in the aerospace industry, lifesaving and/or life supporting medical equipment, equipment used for automobiles, trains, ships and other transportation, traffic signaling equipment, equipment used to control combustions or explosions, safety devices, elevators and escalators, and devices related to power plant. **IF YOU USE PRODUCT FOR UNINTENDED USE, TOSHIBA ASSUMES NO LIABILITY FOR PRODUCT.** For details, please contact your TOSHIBA sales representative or contact us via our website.
- Do not disassemble, analyze, reverse-engineer, alter, modify, translate or copy Product, whether in whole or in part.
- Product shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable laws or regulations.
- The information contained herein is presented only as guidance for Product use. No responsibility is assumed by TOSHIBA for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.
- **ABSENT A WRITTEN SIGNED AGREEMENT, EXCEPT AS PROVIDED IN THE RELEVANT TERMS AND CONDITIONS OF SALE FOR PRODUCT, AND TO THE MAXIMUM EXTENT ALLOWABLE BY LAW, TOSHIBA (1) ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION, INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES OR LOSS, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, LOSS OF OPPORTUNITIES, BUSINESS INTERRUPTION AND LOSS OF DATA, AND (2) DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES AND CONDITIONS RELATED TO SALE, USE OF PRODUCT, OR INFORMATION, INCLUDING WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OF INFORMATION, OR NONINFRINGEMENT.**
- Do not use or otherwise make available Product or related software or technology for any military purposes, including without limitation, for the design, development, use, stockpiling or manufacturing of nuclear, chemical, or biological weapons or missile technology products (mass destruction weapons). Product and related software and technology may be controlled under the applicable export laws and regulations including, without limitation, the Japanese Foreign Exchange and Foreign Trade Law and the U.S. Export Administration Regulations. Export and re-export of Product or related software or technology are strictly prohibited except in compliance with all applicable export laws and regulations.
- Please contact your TOSHIBA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. **TOSHIBA ASSUMES NO LIABILITY FOR DAMAGES OR LOSSES OCCURRING AS A RESULT OF NONCOMPLIANCE WITH APPLICABLE LAWS AND REGULATIONS.**