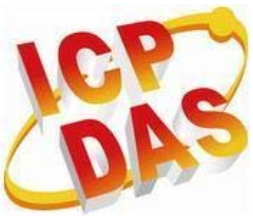
# ET-M8194H
# Motion Control Module
# User Manual
**(Version 2.0)**

## API Library




**ICP DAS CO., LTD.**

## Warranty

**All products manufactured by ICPDAS Inc. are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.**

## Warning

**ICPDAS Inc. assumes no liability for damages consequent to the use of this product. ICPDAS Inc. reserves the right to change this manual at any time without notice. The information furnished by ICPDAS Inc. is believed to be accurate and reliable. However, no responsibility is assumed by ICPDAS Inc. for its use, or for any infringements of patents or other rights of third parties resulting from its use.**

## Copyright



## Trademark

**The names used for identification only maybe registered trademarks of their respective companies.**

## License

**The user can use, modify and backup this software on a single machine. The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.**

# 1 Preface

## 1.1 Introduction

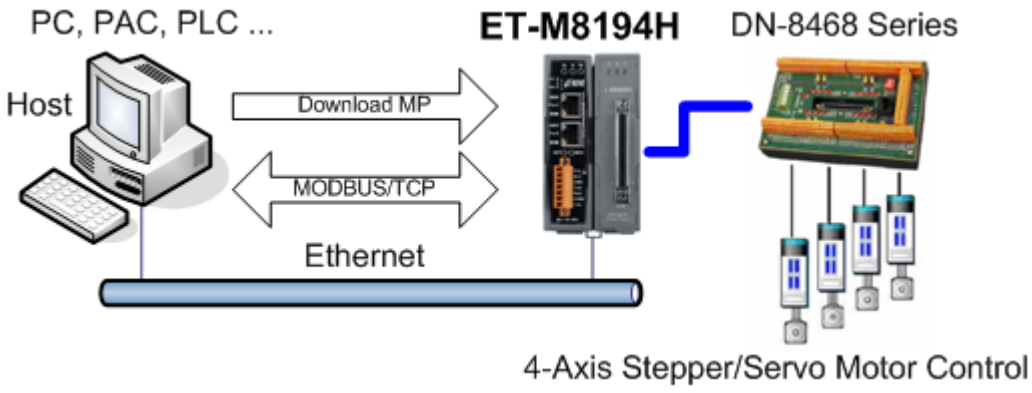
- This manual includes nine chapters and nine appendices. This chapter gives a brief description of this manual. The Chapters 2 to 7 provide the MODBUS command of the relevant ET-M8194H API; the Chapter 8 descripes why it can be a standalone controller; finally, the Chapter 9 introduces those communication functions and error codes. The Appendices include the definitions of MODBUS address, Sub Function Code of ET-M8194H API, the LED of ET-M8194H and Lock_IP feature.

- This documentation helps user control the ET-M8194H via MODBUS/TCP. Most functions are implemented with the FC=16, start_address = 8000. Other state-related functions use FC= 3, 4, 16, 1, 2, 5 and 15 to access ET-M8194H.

## 1.2 ET-M8194H Features

ET-M8194H is a product of ICP DAS to implement the Ethernet remote motion controller for the remote motion solution. It includes an i-8094H module (a 4-axis stepping/pulse-type servo motor control module). By using the intelligent function of ET-M8194H, we can do the remote motion control for various motion applications via MODBUS/TCP.

Also, ET-M8194H can be applied in many platforms with MODBUS/TCP protocol (for example: PC, PAC, PLC, HMI). Additionally, it is easy and simple to use many ET-M8194Hs at the same time to implement the multi-axis motion control by the serial connection of ET-M8194H. ICP DAS also provides EzMove Utility and API Library for users to quickly configure ET-M8194H and develop their own control applications easily.



## 1.3 Function description

- **Function_name( parameter1, parameter2 ……)**

**Description:** Explanation of this function.

**Category:** Function categories description.

**Parameters:** Definitions of the parameters and how to use them.

**Return:** The return value of this function.

**Example:** Simple example program.

**Remark:** Comments.

**MODBUS Register Table:**

For instance, the commands of MODBUS are defined as follows:

(1) **TID**: Transaction ID.
(2) **PID**: Protocol ID.
(3) **Field Length**: Length of data-field (in Byte). The data-field begins from UID and end at last Register.
(4) **UID**: Unit ID (CardNo).
(5) **FC**: Function Code.
(6) **St_Addr.**: Starting Address.
(7) **Word Count**: The length of the following Registers (in Word, 16-bit).
(8) **Byte Count**: The length of the following Registers (in Byte, 8-bit).
(9) **Register**: Content of parameter (16-bit).
  (9-1) **Sub_Function Code**: The pre-defined code for each ET-M8194H function.
  (9-2) **Axis**: The target Axis/Axes of the ET-M8194H function.

For each function, the contents of each field and the required Registers will be listed after function description. The MODBUS Registers only support 16-bit data format. Therefore, two Registers will be needed for the 32-bit value. If WORD Order is set to 0, the first register is high-word (MSW) and the second register is low-word (LSW).

For instance, calling ETM_SET_LP() to set all axes' LP to zero needs the *Sub_Function Code* 0x0A28 and value for axes to be 0x000F. The zero value has the 32-bit hexical format as 0x00000000. Therefore, the MSW and LSW are both 0x0000. The contents of MODBUS Register Table values for this calling will be listed as follows:

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 28 | *Sub_function code* |
| 1 | 00 0F | *axis (F ➔ AXIS_XYZU)* |
| 2 | 00 00 | *MSW of wdata* |
| 3 | 00 00 | *LSW of wdata* |

## 1.4 Function categories description

**RTC (Real Time Command): Functions will be buffered on DPRAM and can be executed immediately if buffer is empty or be executed later on if the internal CPU is busy doing something at that moment.**

**MP (Macro Program): MP is a set of APIs which are stored in memory. MP will be executed when MP_CALL is used to call its number.**

**ISR (Interrupt Service Routine): An ISR is similar to an MP. An ISR will be executed when MP_ISR_CALL is used to call it. But most of time, it is called when a related interrupt occurs. Therefore, some APIs can not be applied inside an ISR.**

**IT (Initial Table): All functions defined in this talbe will be executed when power on ET-M8194H.**

**Maximum number of Function Line for ISR1 ~ ISR20 and MP1 ~ MP157 are listed in the following table.**

| ISR(6) | ISR1 | ISR2 | ISR3 | ISR4 | ISR5 | ISR6 | | | |
|---|---|---|---|---|---|---|---|---|---|
| *Total:* | 8 | 8 | 8 | 8 | 8 | 8 | | | |
| ISR(9) | ISR7 | ISR8 | ISR9 | ISR10 | ISR11 | ISR12 | ISR13 | ISR14 | ISR15 |
| *Total:* | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| ISR(3) | ISR16 | ISR17 | ISR18 | | | | | | |
| *Total:* | 32 | 32 | 32 | | | | | | |
| ISR(2) | ISR19 | ISR20 | | | | | | | |
| *Total:* | 64 | 64 | | | | | | | |
| MP(40) | MP1 | ~ | MP40 | | | | | | |
| *Total:* | 8 | | 8 | | | | | | |
| MP(50) | MP41 | ~ | MP90 | | | | | | |
| *Total:* | 16 | | 16 | | | | | | |
| MP(40) | MP91 | ~ | MP130 | | | | | | |
| *Total:* | 32 | | 32 | | | | | | |
| MP(20) | MP131 | ~ | MP150 | | | | | | |
| *Total:* | 64 | | 64 | | | | | | |
| MP(5) | MP151 | MP152 | MP153 | MP154 | MP155 | | | | |
| *Total:* | 128 | 128 | 128 | 128 | 128 | | | | |
| MP(2) | MP156 | MP157 | | | | | | | |
| *Total:* | 512 | 512 | | | | | | | |

**In the following Function Table, most functions in sections 2, 3, 4, 5 and 6 could be used in i8094H_MP_CREATE (Please refer to Section 7.2.1), all values could be replaced by variables (when applied to MP or ISR).**
**b*varNo*: User-defined variables: bVAR0 ~ bVAR127 (Data type :BYTE)**
***varNo*: User-defined variables: VAR0 ~ VAR511 (Data type :long or DWORD)**

**Note: In the following sections ※ indicates functions applied to MP.**
**In the following sections Δ indicates functions applied to ISR.**
**The mark ◎ indicates the function is available in that category.**
**The mark ◎x2 means the function is available in that category with two MP function lines.**

**Table 1: Command types and command scope**

| Basic settings | | | | | |
|---|---|---|---|---|---|
| Section | Function | RTC | MP | ISR | IT |
| 2.3 | ETM_RESET_CARD | ◎ | | | |
| 2.3 | ETM_CLEAR_CARD_BUFFER | ◎ | | | |
| 2.4 | ETM_SET_PULSE_MODE | ◎ | | | ◎ |
| | ETM_MACRO_SET_PULSE_MODE | | ◎ | | ◎ |
| 2.5 | ETM_SET_MAX_V | ◎ | | | ◎ |
| | ETM_MACRO_SET_MAX_V | | ◎ | | ◎ |
| 2.6 | ETM_SET_HLMT | ◎ | | | ◎ |
| | ETM_MACRO_SET_HLMT | | ◎ | | ◎ |
| 2.7 | ETM_LIMITSTOP_MODE | ◎ | | | ◎ |
| | ETM_MACRO_LIMITSTOP_MODE | | ◎ | | ◎ |
| 2.8 | ETM_SET_NHOME | ◎ | | | ◎ |
| | ETM_MACRO_SET_NHOME | | ◎ | | ◎ |
| 2.9 | ETM_SET_HOME_EDGE | ◎ | | | ◎ |
| | ETM_MACRO_SET_HOME_EDGE | | ◎ | | ◎ |
| 2.10 | ETM_SET_SLMT | ◎ | | | ◎ |
| | ETM_MACRO_SET_SLMT | | ◎ | | ◎ |
| 2.10 | ETM_CLEAR_SLMT | ◎ | | | ◎ |
| | ETM_MACRO_CLEAR_SLMT | | ◎ | | ◎ |
| 2.11 | ETM_SET_ENCODER | ◎ | | | ◎ |
| | ETM_MACRO_SET_ENCODER | | ◎ | | ◎ |
| 2.12 | ETM_SERVO_ON | ◎ | | | ◎ |
| | ETM_MACRO_SERVO_ON | | ◎ | | ◎ |
| 2.12 | ETM_SERVO_OFF | ◎ | | | ◎ |
| | ETM_MACRO_SERVO_OFF | | ◎ | | ◎ |
| 2.13 | ETM_SET_ALARM | ◎ | | | ◎ |
| | ETM_MACRO_SET_ALARM | | ◎ | | ◎ |
| 2.14 | ETM_SET_INPOS | ◎ | | | ◎ |
| | ETM_MACRO_SET_INPOS | | ◎ | | ◎ |
| 2.15 | ETM_SET_FILTER | ◎ | | | ◎ |
| | ETM_MACRO_SET_FILTER | | ◎ | | ◎ |
| | | | | | |

| Section | Function | RTC | MP | ISR | IT |
|---|---|---|---|---|---|
| 2.16 | ETM_VRING_ENABLE | ◎ | | | ◎ |
| | ETM_MACRO_VRING_ENABLE | | ◎ | | ◎ |
| 2.16 | ETM_VRING_DISABLE | ◎ | | | ◎ |
| | ETM_MACRO_VRING_DISABLE | | ◎ | | ◎ |
| 2.17 | ETM_AVTRI_ENABLE | ◎ | | | ◎ |
| | ETM_MACRO_AVTRI_ENABLE | | ◎ | | ◎ |
| 2.17 | ETM_AVTRI_DISABLE | ◎ | | | ◎ |
| | ETM_MACRO_AVTRI_DISABLE | | ◎ | | ◎ |
| 2.18 | ETM_EXD_MP | ◎ | | | |
| 2.18 | ETM_EXD_FP | ◎ | | | |
| 2.18 | ETM_EXD_CP | ◎ | | | |
| 2.18 | ETM_EXD_DISABLE | ◎ | | | |
| 2.19 | ETM_READ_bVAR | ◎ | | | |
| 2.19 | ETM_WRITE_bVAR | ◎ | | | |
| 2.19 | ETM_READ_VAR | ◎ | | | |
| 2.19 | ETM_WRITE_VAR | ◎ | | | |
| 2.20 | ETM_READ_MD | ◎ | | | |
| 2.20 | ETM_WRITE_MD | ◎ | | | |
| **Status Reading and Setting** | | | | | |
| 3.1 | ETM_SET_LP | ◎ | | | |
| | ETM_MACRO_SET_LP | | ◎ | ◎ | |
| 3.1 | ETM_GET_LP | ◎ | | | |
| | ETM_MACRO_GET_LP | | ◎ | ◎ | |
| | ETM_GET_LP_4_AXIS | ◎ | | | |
| 3.2 | ETM_SET_EP | ◎ | | | |
| | ETM_MACRO_SET_EP | | ◎ | ◎ | |
| 3.2 | ETM_GET_EP | ◎ | | | |
| | ETM_MACRO_GET_EP | | ◎ | ◎ | |
| | ETM_GET_EP_4_AXIS | ◎ | | | |
| 3.3 | ETM_GET_CV | ◎ | | | |
| | ETM_GET_CV_4_AXIS | ◎ | | | |
| 3.4 | ETM_GET_CA | ◎ | | | |
| | ETM_GET_CA_4_AXIS | ◎ | | | |
| 3.5 | ETM_MACRO_GET_DI | | ◎ | ◎ | |
| 3.5 | ETM_GET_DI_ALL | ◎ | | | |
| | ETM_MACRO_GET_DI_ALL | | ◎ | ◎ | |
| | ETM_GET_DI_ALL_4_AXIS | ◎ | | | |
| 3.6 | ETM_MACRO_GET_ERROR | | ◎ | ◎ | |
| | ETM_GET_ERROR_STATE | ◎ | | | |
| 3.6 | ETM_GET_ERROR_CODE | ◎ | | | |
| | ETM_MACRO_GET_ERROR_CODE | | ◎ | ◎ | |
| | ETM_GET_ERROR_CODE_4_AXIS | ◎ | | | |
| 3.7 | ETM_GET_FREE_BUFFER | ◎ | | | |
| | | | | | |

| Section | Function | RTC | MP | ISR | IT |
|---|---|---|---|---|---|
| 3.8 | ETM_GET_STOP_STATUS | ◎ | | | |
| | ETM_GET_STOP_STATUS_4_AXIS | ◎ | | | |
| **FRnet Function** | | | | | |
| 4.1 | ETM_MACRO_FRNET_IN | | ◎ | ◎ | |
| | ETM_MACRO_FRNET_READ | | ◎ | ◎ | |
| | ETM_FRNET_READ_SINGLE_DIO | ◎ | | | |
| | ETM_FRNET_READ_GROUP_DIO | ◎ | | | |
| | ETM_FRNET_READ_MULTI_GROUP_DIO | ◎ | | | |
| 4.2 | ETM_ MACRO_FRNET_OUT | | ◎ | ◎ | |
| | ETM_MACRO_FRNET_WRITE | | ◎ | ◎ | |
| | ETM_FRNET_WRITE_SINGLE_DO | ◎ | | | |
| | ETM_FRNET_WRITE_GROUP_DO | ◎ | | | |
| | ETM_FRNET_WRITE_MULTI_GROUP_DO | ◎ | | | |
| 4.3 | ETM_MACRO_FRNET_WAIT | | ◎ | ◎ | |
| **Home Search** | | | | | |
| 5.1 | ETM_SET_HV | ◎ | | | |
| | ETM_MACRO_SET_HV | | ◎ | | |
| 5.2 | ETM_HOME _LIMIT | ◎ | | | |
| | ETM_MACRO_HOME _LIMIT | | ◎ | | |
| 5.3 | ETM_SET_HOME_MODE | ◎ | | | |
| | ETM_MACRO_SET_HOME_MODE | | ◎ | | |
| 5.4 | ETM_HOME _START | ◎ | | | |
| | ETM_MACRO_HOME _START | | ◎ | | |
| **Independent Axis Motion Control** | | | | | |
| 6.1.1 | ETM_NORMAL_SPEED | ◎ | | | |
| | ETM_MACRO_NORMAL_SPEED | | ◎ | ◎ | |
| 6.1.2 | ETM_SET_SV | ◎ | | | |
| | ETM_MACRO_SET_SV | | ◎ | ◎ | |
| 6.1.3 | ETM_SET_V | ◎ | | | |
| | ETM_MACRO_SET_V | | ◎ | ◎ | |
| 6.1.4 | ETM_SET_A | ◎ | | | |
| | ETM_MACRO_SET_A | | ◎ | ◎ | |
| 6.1.5 | ETM_SET_D | ◎ | | | |
| | ETM_MACRO_SET_D | | ◎ | ◎ | |
| 6.1.6 | ETM_SET_K | ◎ | | | |
| | ETM_MACRO_SET_K | | ◎ | ◎ | |
| 6.1.7 | ETM_SET_L | ◎ | | | |
| | ETM_MACRO_SET_L | | ◎ | ◎ | |
| 6.1.8 | ETM_SET_AO | ◎ | | | |
| | ETM_MACRO_SET_AO | | ◎ | ◎ | |
| 6.1.9 | ETM_FIXED_MOVE | ◎ | | | |
| | ETM_MACRO_FIXED_MOVE | | ◎ | ◎ | |
| 6.1.9 | ETM_SET_PULSE | ◎ | | | |
| | ETM_MACRO_SET_PULSE | | ◎ | ◎ | |
| 6.1.10 | ETM_CONTINUE_MOVE | ◎ | | | |
| | ETM_MACRO_CONTINUE_MOVE | | ◎ | ◎ | |
| | | | | | |

| Section | Function | RTC | MP | ISR | IT |
|---|---|---|---|---|---|
| **Interpolation Motion** | | | | | |
| 6.2.1 | ETM_AXIS_ASSIGN | ◎ | | | |
| | ETM_MACRO_AXIS_ASSIGN | | ◎ | ◎ | |
| 6.2.2 | ETM_VECTOR_SPEED | ◎ | | | |
| | ETM_MACRO_VECTOR_SPEED | | ◎ | ◎ | |
| 6.2.3 | ETM_SET_VSV | ◎ | | | |
| | ETM_MACRO_SET_VSV | | ◎ | ◎ | |
| 6.2.4 | ETM_SET_VV | ◎ | | | |
| | ETM_MACRO_SET_VV | | ◎ | ◎ | |
| 6.2.5 | ETM_SET_VA | ◎ | | | |
| | ETM_MACRO_SET_VA | | ◎ | ◎ | |
| 6.2.6 | ETM_SET_VD | ◎ | | | |
| | ETM_MACRO_SET_VD | | ◎ | ◎ | |
| 6.2.7 | ETM_SET_VK | ◎ | | | |
| | ETM_MACRO_SET_VK | | ◎ | ◎ | |
| 6.2.8 | ETM_SET_VL | ◎ | | | |
| | ETM_MACRO_SET_VL | | ◎ | ◎ | |
| 6.2.9 | ETM_SET_VAO | ◎ | | | |
| | ETM_MACRO_SET_VAO | | ◎ | ◎ | |
| 6.2.10 | ETM_LINE_2D | ◎ | | | |
| | ETM_MACRO_LINE_2D | | ◎ | ◎ | |
| 6.2.11 | ETM_LINE_3D | ◎ | | | |
| | ETM_MACRO_LINE_3D | | ◎ | ◎ | |
| 6.2.12 | ETM_ARC_CW | ◎x2 | | | |
| | ETM_MACRO_ARC_CW | | ◎x2 | ◎x2 | |
| 6.2.12 | ETM_ARC_CCW | ◎x2 | | | |
| | ETM_MACRO_ARC_CCW | | ◎x2 | ◎x2 | |
| 6.2.13 | ETM_CIRCLE_CW | ◎ | | | |
| | ETM_MACRO_CIRCLE_CW | | ◎ | ◎ | |
| 6.2.13 | ETM_CIRCLE_CCW | ◎ | | | |
| | ETM_MACRO_CIRCLE_CCW | | ◎ | ◎ | |
| **Synchronous Actions** | | | | | |
| 6.3.1 | ETM_SYNC_ACTION | ◎x2 | | | |
| | ETM_MACRO_SYNC_ACTION | | ◎x2 | ◎x2 | |
| | ETM_CLEAR_SYNC_ACTION | ◎ | | | |
| | ETM_MACRO_CLEAR_SYNC_ACTION | | ◎ | ◎ | |
| | ETM_SET_ACTIVATION_FACTORS | ◎ | | | |
| | ETM_MACRO_SET_ACTIVATION_FACTORS | | ◎ | ◎ | |
| | ETM_SET_ACTIVATION_AXIS | ◎ | | | |
| | ETM_MACRO_SET_ACTIVATION_AXIS | | ◎ | ◎ | |
| | ETM_SET_ACTION | ◎ | | | |
| | ETM_MACRO_SET_ACTION | | ◎ | ◎ | |
| 6.3.2 | ETM_SET_COMPARE | ◎ | | | |
| | ETM_MACRO_SET_COMPARE | | ◎ | ◎ | |
| 6.3.3 | ETM_GET_LATCH | ◎ | | | |
| | ETM_MACRO_GET_LATCH | | ◎ | ◎ | |
| | ETM_GET_LATCH_4_AXIS | ◎ | | | |
| 6.3.4 | ETM_SET_PRESET | ◎ | | | |
| | ETM_MACRO_SET_PRESET | | ◎ | ◎ | |

| Section | Function | RTC | MP | ISR | IT |
|---|---|---|---|---|---|
| 6.3.5 | ETM_SET_OUT | ◎ | | | |
| | ETM_MACRO_SET_OUT | | ◎ | ◎ | |
| **Enable / Disable Interrupt Function** | | | | | |
| 6.3.6 | ETM_ENABLE_INT | ◎ | | | |
| | ETM_MACRO_ENABLE_INT | | ◎ | | |
| 6.3.6 | ETM_DISABLE_INT | ◎ | | | |
| | ETM_MACRO_DISABLE_INT | | ◎ | | |
| 6.3.6 | ETM_INTFACTOR_ENABLE | ◎ | | | |
| | ETM_MACRO_INTFACTOR_ENABLE | | ◎ | ◎ | |
| 6.3.6 | ETM_INTFACTOR_DISABLE | ◎ | | | |
| | ETM_MACRO_INTFACTOR_DISABLE | | ◎ | ◎ | |
| **Read Triggered Interrupt Factors** | | | | | |
| 6.4 | ETM_GET_TRIG_INTFACTOR | ◎ | | | |
| **Macro Download status** | | | | | |
| 6.5 | ETM_GET_MP_DOWNLOAD_STATUS | ◎ | | | |
| **Read ETM_MACRO_SET_RINT Triggered Interrupt** | | | | | |
| 6.6 | ETM_GET_USER_RINT | ◎ | | | |
| **Get ET-M8194H State** | | | | | |
| 6.7 | ETM_GET_DEVICE_STATE | ◎ | | | |
| **Continuous Interpolation** | | | | | |
| 6.8.1 | ETM_RECTANGLE | ◎x4 | | | |
| | ETM_MACRO_RECTANGLE | | ◎x4 | | |
| 6.8.2 | ETM_LINE_2D_INITIAL | ◎x2 | | | |
| | ETM_MACRO_LINE_2D_INITIAL | | ◎x2 | | |
| 6.8.2 | ETM_LINE_2D_CONTINUE | ◎ | | | |
| | ETM_MACRO_LINE_2D_CONTINUE | | ◎ | | |
| 6.8.3 | ETM_LINE_3D_INITIAL | ◎x2 | | | |
| | ETM_MACRO_LINE_3D_INITIAL | | ◎x2 | | |
| 6.8.3 | ETM_LINE_3D_CONTINUE | ◎ | | | |
| | ETM_MACRO_LINE_3D_CONTINUE | | ◎ | | |
| 6.8.4 | ETM_MIX_2D_INITIAL | ◎x2 | | | |
| | ETM_MACRO_MIX_2D_INITIAL | | ◎x2 | | |
| 6.8.4 | ETM_MIX_2D_CONTINUE | ◎x2 | | | |
| | ETM_MACRO_MIX_2D_CONTINUE | | ◎x2 | | |
| 6.8.5 | ETM_CONTINUE_INTP | ◎ | | | |
| 6.8.6 | ETM_HELIX_3D | ◎x3 | | | |
| | ETM_MACRO_HELIX_3D | | ◎x3 | | |
| 6.8.7 | ETM_RATIO_INITIAL | ◎x2 | | | |
| | ETM_MACRO_RATIO_INITIAL | | ◎x2 | | |
| 6.8.7 | ETM_RATIO_2D | ◎ | | | |
| | ETM_MACRO_RATIO_2D | | ◎ | | |
| 6.8.8 | ETM_LINE_SCAN | ◎ | | | |
| 6.8.8 | ETM_LINE_SCAN_START | ◎ | | | |
| 6.8.8 | ETM_LINE_SCAN_OFFSET2 | ◎ | | | |
| 6.8.8 | ETM_GET_LINE_SCAN_DONE | ◎ | | | |
| | | | | | |

| Section | Function | RTC | MP | ISR | IT |
|---|---|---|---|---|---|
| **Other Functions** | | | | | |
| 6.9.1 | ETM_DRV_HOLD | ◎ | | | |
| | ETM_MACRO_DRV_HOLD | | ◎ | ◎ | |
| 6.9.2 | ETM_DRV_START | ◎ | | | |
| | ETM_MACRO_DRV_START | | ◎ | ◎ | |
| 6.9.3 | ETM_STOP_SLOWLY | ◎ | | | |
| | ETM_MACRO_STOP_SLOWLY | | ◎ | | |
| 6.9.3 | ETM_STOP_SUDDENLY | ◎ | | | |
| | ETM_MACRO_STOP_SUDDENLY | | ◎ | | |
| 6.9.3 | ETM_VSTOP_SLOWLY | ◎ | | | |
| | ETM_MACRO_VSTOP_SLOWLY | | ◎ | | |
| 6.9.3 | ETM_VSTOP_SUDDENLY | ◎ | | | |
| | ETM_MACRO_VSTOP_SUDDENLY | | ◎ | | |
| 6.9.4 | ETM_CLEAR_STOP | ◎ | | | |
| | ETM_MACRO_CLEAR_STOP | | ◎ | | |
| 6.9.4 | ETM_CLEAR_VSTOP | ◎ | | | |
| | ETM_MACRO_CLEAR_VSTOP | | ◎ | | |
| 6.9.5 | ETM_INTP_END | ◎ | | | |
| | ETM_MACRO_INTP_END | | ◎ | | |
| 6.9.6 | ETM_EMERGENCY_STOP | ◎ | | | |
| 6.9.7 | ETM_CLEAR_EMERGENCY_STOP | ◎ | | | |
| **Additional Functions supported by i-8094H** | | | | | |
| 7.1 | ETM_LOAD_INITIAL | ◎ | | | |
| 7.2.1 | ETM_MP_CREATE | ◎ | | | |
| | ETM_MACRO_MP_CLOSE | | ◎ | | |
| 7.2.2 | ETM_MP_CALL | ◎ | | | |
| | ETM_MACRO_MP_CALL | | ◎ | | |
| 7.3.1 | ETM_MP_ISR_CREATE | ◎ | | | |
| 7.3.2 | ETM_MACRO_MP_ISR_CLOSE | | | ◎ | |
| 7.3.3 | ETM_MP_ISR_CALL | ◎ | | | |
| 7.4.1 | ETM_MACRO_SET_VAR | | ◎ | ◎ | |
| 7.4.2 | ETM_ MACRO_SET_RVAR | | ◎ | ◎ | |
| 7.5 | ETM_ MACRO_VAR_CALCULATE | | ◎ | ◎ | |
| 7.6.1 | ETM_ MACRO_FOR | | ◎ | | |
| 7.6.2 | ETM_ MACRO_NEXT | | ◎ | | |
| 7.6.3 | ETM_ MACRO_EXIT_FOR | | ◎ | | |
| 7.7.1 | ETM_ MACRO_IF | | ◎ | ◎ | |
| 7.7.2 | ETM_ MACRO_ELSE | | ◎ | ◎ | |
| 7.7.3 | ETM_ MACRO_END_IF | | ◎ | ◎ | |
| 7.8.1 | ETM_MACRO_GOTO | | ◎ | | |
| 7.8.2 | ETM_MACRO_LABEL | | ◎ | | |
| 7.9 | ETM_ MACRO_TIMER | | ◎ | | |
| 7.10 | ETM_STOP_WAIT | ◎ | | | |
| | ETM_MACRO_STOP_WAIT | | ◎ | | |
| 7.11 | ETM_ MACRO_SET_RINT | | ◎ | ◎ | |
| 7.12 | ETM_MACRO_EXIT_MACRO | | ◎ | ◎ | |
| 7.13 | ETM_MP_TERMINATE | ◎ | | | |
| | ETM_MACRO_MP_TERMINATE | | ◎ | ◎ | |

# 2 Basic Settings

## 2.1 Axes Code Definition

**The definition of axis assignments is as follows: X=1, Y=2, Z=4, and U=8. If you assign X and Y axes simultaneously, the code will be 3. In a similar way, AXIS_YZ = 2+4 =0x6; and AXIS_XYZU = 1+2+4+8 = 0xf. You could assign single axis as well as multiple axes. Available axis codes are listed below:**

**Table 2: Axis assignments and their corresponding codes**

| Axis | X | Y | Z | U | XY | XZ | XU | YZ |
|---|---|---|---|---|---|---|---|---|
| Code | 0x1 | 0x2 | 0x4 | 0x8 | 0x3 | 0x5 | 0x9 | 0x6 |
| Variable | AXIS_X | AXIS_Y | AXIS_Z | AXIS_U | AXIS_XY | AXIS_XZ | AXIS_XU | AXIS_YZ |
| Axis | YU | ZU | XYZ | XYU | XZU | YZU | XYZU | |
| Code | 0xa | 0xc | 0x7 | 0xb | 0xd | 0xe | 0xf | |
| Variable | AXIS_YU | AXIS_ZU | AXIS_XYZ | AXIS_XYU | AXIS_XZU | AXIS_YZU | AXIS_XYZU | |

**Write the setting values into the IT parameter table without making a change of other current settings (please refer to Section 7.1), the definitions are as follow:**

**Table 3: Axis assignments and their corresponding codes for initialization table (IT)**

| Axis | X | Y | Z | U | XY | XZ | XU | YZ |
|---|---|---|---|---|---|---|---|---|
| Code | 0x11 | 0x12 | 0x14 | 0x18 | 0x13 | 0x15 | 0x19 | 0x16 |
| Variable | INITIAL_X | INITIAL_Y | INITIAL_Z | INITIAL_U | INITIAL_XY | INITIAL_XZ | INITIAL_XU | INITIAL_YZ |
| Axis | YU | ZU | XYZ | XYU | XZU | YZU | XYZU | |
| Code | 0x1a | 0x1c | 0x17 | 0x1b | 0x1d | 0x1e | 0x1f | |
| Variable | INITIAL_YU | INITIAL_ZU | INITIAL_XYZ | INITIAL_XYU | INITIAL_XZU | INITIAL_YZU | INITIAL_XYZU | |

**IT table can not be used inside Macro Program (MP or ISR).**

## 2.2 Registering mudules and getting LIB version

**BYTE i8094H_REGISTRATION(BYTE *cardNo*, BYTE *slot*)**

**Description:**

**You are required to register your i8094H before performing any operation. This function enables to register a module by doing the following steps: module registration; assigning the slot number the module being installed on; and accessing a card number.**

**Registration must be performed for each I-8094(A/H) motion control module before other functions are called. After registration, each module can be identified by its corresponding module number.**

**Category:**

**MODBUS table ; Internal function.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *slot*: | Slot number → MP-8000 : 1~7 |

**Return:**

**YES: Normal; NO: Abnormal**

**Example:**

```
//================== for MP-8000 ==================
//set each module number the same as the slot number, respectively.
//(slot1 ~ slot7)
BYTE cardNo;
BYTE slot;
int Found = 0;
for (slot = 1; slot < 8; slot++)
{
    cardNo = slot;
    if (i8094H _REGISTRATION(cardNo, slot) == YES)
     {   //slot number begins from 1.
         //if a module is found, the slot number would be registered as the slot
         number of that module.
         i8094H _RESET_CARD(cardNo);
         Found++;
    }
```

```
}
if (Found == 0)
{
      //if MP-8000 cannot find any I-8094H module,
      please add your code here to take care of the exceptional cases.
      return;
}
```

**Special Description:**

**This function only give MP-8000 or the controller which direct control the i-8094H to use. ET-M8194H does not provide the corresponding MODBUS command function. The start of the ET-M8194H automatically for registered i-8049H so the user does not need to use this function. ET-M8194H will startup by the following initialization actions:**

```
i8094H_REGISTRATION(CardNo, 0);
i8094H_RESET_CARD(CardNo);
i8094H_SET_PULSE_MODE(CardNo, AXIS_XYZU, 2);
i8094H_SET_ALARM(CardNo, AXIS_XYZU, 0, 0);              //disable Servo Alarm Input
i8094H_SET_ENCODER(CardNo, AXIS_XYZU, 0, 0, 0);         //Set Encoder Input Type
i8094H_EXD_DISABLE(CardNo, AXIS_XYZU);                  //set External Input Off
i8094H_SET_LP(CardNo, AXIS_XYZU, 0);              //set Logic position =0
i8094H_SET_EP(CardNo, AXIS_XYZU, 0);              //set Encoder position =0
i8094H_SERVO_ON(CardNo, AXIS_XYZU);               //set Servo_ON to servo motors
i8094H_CLEAR_SLMT(CardNo, AXIS_XYZU);
i8094H_SET_MAX_V(CardNo, AXIS_XYZU, 160000L);
```

**Users can use MODBUS command to check what kind of module is inserted in the controller slot. Use FC=4 to read the value of address 0x51(decimal is 81). If an i-8094H module is plugged inside the slot, value 0x44 will be returned; if an i-8094A module is inside the slot, 0x55 will be returned. In addition, UID = 01 represents the first slot. ET-M8194H only supports one slot now. If additional slot is supported in the future, users can change the UID value to 02 to read the data from the second slot.**

**MODBUS example:**

```
WORD ModID;
HANDLE h;
ETM_GET_MODULE_ID (h, 1, &ModID);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 51 | 00 01 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | *i-8094H: 0x44*<br>*i-8094A: 0x55*<br>(hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 44 |

```
WORD i8094H_GET_VERSION(void)
```

**Description:**

Read the current version of i8094H library.

**Category:**

MODBUS table; Internal function.

**Parameters:**

None

**Return:**

Version number, 0x0000 ~ 0x9999

**Example:**

```
WORD VER_No;
VER_No = i8094H_GET_VERSION();
//Read the version code of i8094Hce.dll
```

**Remark:**

If the return value is 0x0607
06: the year is 2006
07: the month is July.

**Special Description:**

This function only supports MP-8000 or other controllers which direct control the i-8094H. However, a MODBUS command is provided for checking the firmware version of the ET-M8194H controller (not a firmware version of i-8094H). Use MODBUS command FC=4 to read the address 0x52 (decimal is 82). If a value 0x02000000 is returned that means the version number is 2.00.

**The function prototype is as follows:**

```
eRET ETM_GET_ETM8194H_FIRMWARE_VERSION (HANDLE h,
     DWORD* ETM8194H_Ver);
```

**MODBUS example:**

```
DWORD ETM8194H_Ver;
HANDLE h;
ETM_GET_ETM8194H_FIRMWARE_VERSION(h, &ETM8194H_Ver);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 52 | 00 02 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | *Version* (hex) High Word | *Version* (hex) Low Word |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 04 | 04 | 02 00 | 00 00 |

**ET-M8194H also allows MODBUS commands for checking the firmware version of i-8094H. Use the FC=4 and read the address 0x54 (decimal is 84). If a value 0x02210201 is returned, the high word represents the hardware version as 2.21, the low word represents the firmware version as 2.01.**

**The function prototype is as follows:**

```
eRET ETM_GET_i8094H_FIRMWARE_VERSION (HANDLE h, DWORD*
     i8094H_Ver);
```

**MODBUS example:**

```
DWORD i8094H_Ver;
HANDLE h;
ETM_GET_i8094H_FIRMWARE_VERSION(h, &i8094H_Ver);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 54 | 00 02 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | *Version* (hex) High Word | *Version* (hex) Low Word |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 04 | 04 | 02 21 | 02 01 |

**Additional function ETM_GET_DLL_VERSION is provided to read ET-M8194H API Library version. This function does not need to send MODBUS commands. Instead, it is an internal function and can be called without connection.**

**The function prototype is as follows:**

```
WORD ETM_GET_DLL_VERSION (void);
```

**If the response WORD value is 0x0100 that means the version number is 1.00. The high byte is the major version number and the low byte is the minor version number.**

## 2.3 Resetting the Motion Module

**eRET ETM_RESET_CARD (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function enables motion module (I8094H) to restore the power-on default settings, please refer to Section 7: initial settings after resetting the module.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_RESET_CARD (h, 1);** **//Reset the module1**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 0A 0A | | *Sub_function code* | | | |
| | | | | | | | |

**eRET ETM_CLEAR_CARD_BUFFER (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**Clear all data in i-8094H command buffer.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_CLEAR_CARD_BUFFER (h, 1); //clear data buffer in module 1**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0A 0B | | *Sub_function code* | | | |
| | | | | | | | |

**Section 3.7 provides ETM_GET_FREE_BUFFER function to get the available block-number inside the command buffer. The maximum block-number is 30).**

## 2.4 Pules Output Mode Setting

**eRET ETM_SET_PULSE_MODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*)**

**※ eRET ETM_MACRO_SET_PULSE_MODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*)**

**Description:**

**This function sets the pulse output mode to be either CW/CCW or PULSE/DIR for the specific axes and their direction.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis:* | Axis or axes (Please refer to Table 2 and Table 3) |
| *nMode*: | Assigned mode (Please refer to Table 4) |

**Table 4: List of pulse output modes**

| | mode | Pulse output signa | |
|---|---|---|---|
| | | nPP | nPM |
| CW / CCW | 0 | CW(risig edge) | CCW(risig edge) |
| | 1 | CW(falling edge) | CCW(falling edge) |
| PULSE / DIR | 2 | PULSE (rising edge) | DIR (LOW:+dir/ HIGH:-dir) |
| | 3 | PULSE (falling edge) | DIR (LOW:+dir/ HIGH:-dir) |
| | 4 | PULSE (rising edge) | DIR (HIGH:+dir/ LOW:-dir) |
| | 5 | PULSE (falling edge) | DIR (HIGH:+dir/ LOW:-dir) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_PULSE_MODE is 0A 0C.**
**The Sub_function code of ETM_MACRO_SET_PULSE_MODE is 0C 0C.**

**MODBUS example:**

**ETM_SET_PULSE_MODE (h, 1, AXIS_XYZ, 2);**

**// set the pulse mode of X, Y, and Z axes to be mode 2 for module 1**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 0C/0C 0C | *Sub_function code* |
| 1 | 00 07 | *axis* |
| 2 | 00 02 | *nMode* |

**Example:**

**ETM_SET_PULSE_MODE (h, 1, AXIS_XYZ, 2);**

**// set the pulse mode of X, Y, and Z axes to be mode 2 in module 1**

**ETM_SET_PULSE_MODE(h, 1, AXIS_U, 3);**

**//set the pulse mode of U axis to be mode 3 in module 1**

**ETM_SET_PULSE_MODE (h, 1, INITIAL_XYZU, 0);**

**//set the pulse mode of X Y Z U axes to be mode 0, write into the initial parameter table(table 3) in module 1**

## 2.5 Set the Maximum Speed

**eRET ETM_SET_MAX_V (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**※ eRET ETM_MACRO_SET_MAX_V (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function sets the maximum rate for the output pulses (speed). A larger value results in a rougher resolution, and vice versa. There are 8000 speed segments available. For example, if the maximum speed is set as 8000 PPS, the resolution will be 1 PPS; if the maximum speed is set as 16000 PPS, the resolution will be 2 PPS; if the maximum speed is set as 80000 PPS, the resolution will be 10 PPS, etc. Maximum value 4,000,000 PPS means the resolution of speed will be 500 PPS. This function will change the resolution of speed to reach the desired maximum speed. Since the scale in hardware is changed, other parameters will be updated accordingly too; such as the starting speed, the acceleration, and the jerk. It is recommended to set the maximum speed value to be an integral multiplier of 8000.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo* | Module number → MP-8000 : 1~7 |
| *axis* | Axis or axes (Table 2 or Table 3)<br><br>If *axis* setting is defined in Table 2, the setting values will change current speed settings.<br><br>If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings. |
| *data* | Range of the maximum speed:<br>single axis: 8,000~4,000,000 PPS;<br>the second axis: 8,000~2,828,854 PPS;<br>the third axis: 8,000~2,309,468 PPS. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_MAX_V is 0A 0D.**
**The Sub_function code of ETM_MACRO_SET_MAX_V is 0C 0D.**

**MODBUS example:**

**ETM_SET_MAX_V (h, 1, AXIS_XY, 200000L);**
**//The maximum speed for the X and Y axes of module 1 is 200KPPS.**
**//The resolution of the speed will be 200000/8000 = 25 PPS.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 0D/0C 0D | *Sub_function code* |
| 1 | 00 03 | *axis* |
| 2 | 00 03 | *MSW of data* |
| 3 | 0D 40 | *LSW of data (200000 = 0x30D40)* |

## 2.6 Setting the Active Level of Hardware Limit Switches

**eRET ETM_SET_HLMT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nFLEdge*, BYTE *nRLEdge*)**

**※ eRET ETM_MACRO_SET_HLMT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nFLEdge*, BYTE *nRLEdge*)**

**Description:**

**This function sets the active logic level of the hardware limit switch inputs.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***nFLEdge*:** | **Active level setting for the forward limit switch.<br>0 = low active; 1 = high active** |
| ***nRLEdge*:** | **Active level setting for the reverse limit switch.<br>0 = low active; 1 = high active** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_HLMT is 0A 0E.**
**The Sub_function code of ETM_MACRO_SET_HLMT is 0C 0E.**

**MODBUS example:**

```
ETM_SET_HLMT (h, 1, AXIS_XYZU, 0, 0);
//set all the trigger levels as low-active for all limit switches on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 0E/0C 0E | *Sub_function code* |
| 1 | 00 0F | *axis* |
| 2 | 00 00 | *nFLEdge* |
| 3 | 00 00 | *nRLEdge* |

## 2.7 Setting the Motion Stop Mode when Limit Switch being turn on

**eRET ETM_LIMITSTOP_MODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*)**

**※ eRET ETM_MACRO_*LIMITSTOP*_MODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*)**

**Description:**

**This function configures the settings of motion stop mode of the axes when the corresponding limit switches being turn on.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***nMode*:** | **0: stop immediately; 1: decelerating to stop** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_LIMITSTOP_MODE is 0A 0F.**
**The Sub_function code of ETM_MACRO_LIMITSTOP_MODE is 0C 0F.**

**MODBUS example:**

```
ETM_LIMITSTOP_MODE (h, 1, AXIS_X, 0);
//set X axis to stop immediately if any limit switch on X axis is turned on.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 0F/0C 0F | *Sub_function code* |
| 1 | 00 01 | *axis* |
| 2 | 00 00 | *nMode* |
| | | |

## 2.8 Setting the Trigger Level of the NHOME Sensor

**eRET ETM_SET_NHOME (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE nNHEdge)**

**※ eRET ETM_MACRO_SET_NHOME (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nNHEdge*)**

**Description:**

This function enables to set up the trigger level of the near home sensor (NHOME).

**Category:**

MODBUS sub_function; RTC, MP and IT.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis:* | Axis or axes (Table 2 or Table 3)<br><br>If *axis* setting is defined in Table 2, the setting values will change current speed settings.<br><br>If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings. |
| *nNHEdge*: | Active level setting for for the near home sensor<br>0 = low active; 1 = high active |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_SET_NHOME is 0A 10.
The Sub_function code of ETM_MACRO_SET_NHOME is 0C 10.

**MODBUS example:**

```
ETM_SET_NHOME (h, 1, AXIS_XY, 0);
// set the trigger level of NHOME of X and Y axes on module 1 to be active low.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 10/0C 10 | *Sub_function code* |
| 1 | 00 03 | *Axis (3 ➔ AXIS_XY)* |
| 2 | 00 00 | *nNHEdge* |
| | | |

## 2.9 Setting the Trigger Level of the HOME Sensor

**eRET ETM_SET_HOME_EDGE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nHEdge*)**

**※ eRET ETM_MACRO_SET_HOME_EDGE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nHEdge*)**

**Description:**

**This function sets the trigger level of the home sensor (HOME)**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***nHEdge*:** | **Active level setting for the home sensor**<br>**0 = low active; 1 = high active** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_HOME_EDGE is 0A 11.**
**The Sub_function code of ETM_MACRO_SET_HOME_EDGE is 0C 11.**

**MODBUS example:**

```
ETM_SET_HOME_EDGE (h , 1, AXIS_XYZU, 1);
//set the trigger level as high active for all home sensors on module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 11/0C 11 | *Sub_function code* |
| 1 | 00 0F | *Axis (F ➔ AXIS_XYZU)* |
| 2 | 00 01 | *nHEdge* |
| | | |

## 2.10 Setting and Clearing the Software Limit

**eRET ETM_SET_SLMT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *dwFL*, long *dwRL*, BYTE *nType*)**

**※ eRET ETM_MACRO_SET_SLMT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *dwFL*, long *dwRL*, BYTE *nType*)**

**Description:**

**This function sets the Positive and Negative software limits.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***dwFL*:** | **Value of the forward software limit (-2,000,000,000 ~ +2,000,000,000)** |
| ***dwRL*:** | **Value of the reverse software limit (-2,000,000,000 ~ +2,000,000,000)** |
| ***nType*:** | **Position counter to be compared<br>0 = logical position counter (LP), i.e., the command position<br>1 = encoder position counter (EP), i.e., the real position** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_SLMT is 0A 12.**
**The Sub_function code of ETM_MACRO_SET_SLMT is 0C 12.**

**MODBUS example:**

**ETM_SET_SLMT (h, 1, AXIS_XYZU, 20000, -3000, 0);**

**//set the forward software limit to be 20000 and the reverse**

**// software limit to be -3000 for all axes on module 1.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 15 | 01 | 10 | 1F 40 | 00 07 | 0E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 12/0C 12 | *Sub_function code* |
| 1 | 00 0F | *axis* |
| 2 | 00 00 | *MSW of dwFL* |
| 3 | 4E 20 | *LSW of dwFL (20000 ➔ 0x4E20)* |
| 4 | FF FF | *MSW of dwRL* |
| 5 | F4 48 | *LSW of dwRL (-3000 ➔ 0xFFFFF448)* |
| 6 | 00 00 | *nType* |

**eRET ETM_CLEAR_SLMT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_CLEAR_SLMT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function clears the software limits.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_CLEAR_SLMT is 0A 13.**
**The Sub_function code of ETM_MACRO_CLEAR_SLMT is 0C 13.**

**MODBUS example:**

```
ETM_CLEAR_SLMT (h, 1, AXIS_XYZU);
//clear the software limits for all axes on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 13/0C 13 | *Sub_function code* |
| 1 | 00 0F | *Axis (F ➔ AXIS_XYZU)* |
| | | |

## 2.11 Setting the Encoder Related Parameters

**eRET ETM_SET_ENCODER (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*, BYTE *nDivision*, BYTE *nZEdge*)**

**※ eRET ETM_MACRO_SET_ENCODER (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*, BYTE *nDivision*, BYTE *nZEdge*)**

**Description:**

**This function sets the relevant parameters for encoder input.**

**Category:**

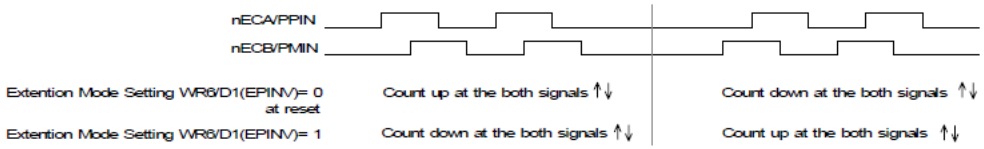
**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***nMode*:** | **Encoder input type: 0 = A quad B; 1 = up/down** |
| ***nDivision*:** | **Division setting for A quad B input signals:**<br>**0 = 1/1**<br>**1 = 1/2**<br>**2 = 1/4** |
| ***nZEdge*:** | **Sets the trigger level for the Z phase**<br>**0 = low active; 1 = high active** |

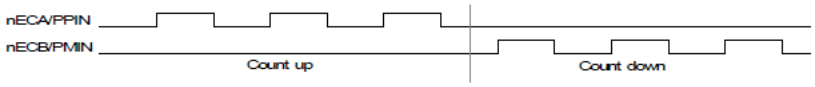
**A/B quadrature pulse input mode:**

**In this mode, when A phase is advancing with positive logical pulses, the count is incremented; and when the B-phase is advancing, the count is decremented. The count is incremented and decremented at the rising edge and falling edge of both signals. In A/B quadrature pulse input mode, the input pulses can be divided into 1/2 or 1/4.**




**Up /down pulse input mode:**

**nECA/PPIN is count up input and nECB/PMIN is count down input. The counter counts at the rising edge of the positive pulse.**

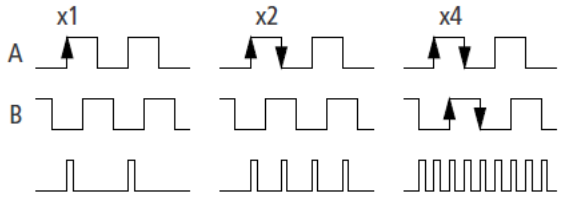



**Division setting for A quad B input signals:**

**0 - Counting positive and negative edge of channel A and B**

**1 - Counting positive and negative edge of the A channel only**

**2 - Considering the positive edge of the A channel only**




**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_ENCODER is 0A 14.**

**The Sub_function code of ETM_MACRO_SET_ENCODER is 0C 14.**

**MODBUS example:**

**ETM_SET_ENCODER (h, 1, AXIS_XYZU, 0, 0, 0);**

**//set the encoder input type as A quad B; the division setting is 1/1; and the Z phase is low active.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 14/0C 14 | *Sub_function code* |
| 1 | 00 0F | *axis (F ➔ AXIS_XYZU)* |
| 2 | 00 00 | *nMode (0 ➔ AB phase)* |
| 3 | 00 00 | *nDivision (0 ➔ 1:1)* |
| 4 | 00 00 | *nZEdge (0➔Z phase is low-active)* |
| | | |

## 2.12 Setting the Servo Driver (ON/OFF)

**eRET ETM_SERVO_ON (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_SERVO_ON (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function outputs a DO signal (ENABLE) to enable the motor driver.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SERVO_ON is 0A 15.**
**The Sub_function code of ETM_MACRO_SERVO_ON is 0C 15.**

**MODBUS example:**

```
ETM_SERVO_ON (h, 1, AXIS_XYZU);
//enables all drivers on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 15/0C 15 | *Sub_function code* |
| 1 | 00 0F | *axis (F ➔ AXIS_XYZU)* |
| | | |

**eRET ETM_SERVO_OFF (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_SERVO_OFF (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function outputs a DO signal (ENABLE) to disable the motor driver.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SERVO_OFF is 0A 16.**
**The Sub_function code of ETM_MACRO_SERVO_OFF is 0C 16.**

**MODBUS example:**

```
ETM_SERVO_OFF (h, 1, AXIS_XYZU);
//disables all drivers on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 16/0C 16 | *Sub_function code* |
| 1 | 00 0F | *axis (F ➔ AXIS_XYZU)* |
| | | |

## 2.13 Setting the SERVO ALARM Function

**eRET ETM_SET_ALARM (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*, BYTE *nAEdge*)**

**※ eRET ETM_MACRO_SET_ALARM (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*, BYTE *nAEdge*)**

**Description:**

**This function sets the ALARM input signal related parameters.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***nMode*:** | **Mode:**<br>**0 = disable ALARM function;**<br>**1 = enable ALARM function** |
| ***nAEdge*:** | **Sets the trigger level**<br>**0 = low active; 1 = high active** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_ALARM is 0A 17.**
**The Sub_function code of ETM_MACRO_SET_ALARM is 0C 17.**

**MODBUS example:**

```
ETM_SET_ALARM (h, 1, AXIS_ZU, 1, 0);
//enable the ALARM for the Z and U axes on module 1 and set them
//as low-active.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 17/0C 17 | *Sub_function code* |
| 1 | 00 0C | *axis (C ➔ AXIS_ZU)* |
| 2 | 00 01 | *nMode (1 ➔ enable)* |
| 3 | 00 00 | *nAEdge (0 ➔ low-active)* |
| | | |

## 2.14 Setting the Active Level of the In-Position Signals

**eRET ETM_SET_INPOS (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*, BYTE *nIEdge*)**

**※ eRET ETM_MACRO_SET_INPOS (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nMode*, BYTE *nIEdge*)**

**Description:**

**This function sets the INPOS input signal related parameters.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***nMode*:** | **Mode:**<br>**0 = disable INPOS input;**<br>**1 = enable INPOS input** |
| ***nIEdge*:** | **Set the trigger level**<br>**0 = low active; 1 = high active** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_INPOS is 0A 18.**
**The Sub_function code of ETM_MACRO_SET_INPOS is 0C 18.**

**MODBUS example:**

```
ETM_SET_INPOS (h, 1, AXIS_X, 1, 0);
//enable the INPOS function of the X axis on module 1 and set it to
//be low-active.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 18/0C 18 | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| 2 | 00 01 | *nMode (1 ➔ enable)* |
| 3 | 00 00 | *nIEdge (0 ➔ low-active)* |
| | | |

## 2.15 Setting the Time Constant of Digital Filter

**eRET ETM_SET_FILTER (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *FEn*, BYTE *FLn*)**

**※ eRET ETM_MACRO_SET_FILTER (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *FEn*, BYTE *FLn*)**

**Description:**

This function selects the axes and sets the time constant for digital filters of the input signals.

**Category:**

MODBUS sub_function; RTC, MP and IT.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis:* | Axis or axes (Table 2 or Table 3)<br><br>If *axis* setting is defined in Table 2, the setting values will change current speed settings.<br><br>If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings. |
| *FEn*: | Enabled filters.<br>The sum of the code numbers (0~31) are used to select input signals. |
| *FLn*: | Sets the filter time constant (0~7). |

*FEn*: Please refer to the following table.

| Code Number | Enabling filters |
|---|---|
| 0 | Disable |
| 1 | EMG, nLMTP, nLMTM, nIN0, nIN1 |
| 2 | nIN2 |
| 4 | nINPOS, nALARM |
| 8 | nEXPP, nEXPM, EXPLSN |
| 16 | nIN3 |

*FLn*: Please refer to the following table.

| Code | Removable max. noise width | Input signal delay time |
|---|---|---|
| 0 | 1.75 μSEC | 2 μSEC |
| 1 | 224 μSEC | 256 μSEC |
| 2 | 448 μSEC | 512 μSEC |
| 3 | 896 μSEC | 1.024mSEC |
| 4 | 1.792mSEC | 2.048mSEC |
| 5 | 3.584mSEC | 4.096mSEC |
| 6 | 7.168mSEC | 8.192mSEC |
| 7 | 14.336mSEC | 16.384mSEC |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_SET_FILTER is 0A 19.
The Sub_function code of ETM_MACRO_SET_FILTER is 0C 19.

**MODBUS example:**

```
ETM_SET_FILTER (h, 1, AXIS_XYZU, 21, 3);
//set the filter time constants of X, Y, Z, and U axes as 1.024mSEC.
//These filters include EMG, nLMTP, nLMTM, nIN0, nIN1, nINPOS, nALARM,
//and nIN3.
//(21 = 1+4+16) 1: EMG + nLMP + nLMPM + nIN0 + nIN1;
//4: nINPOS + nALARM;
//16: nIN3.
```

**Note: The default wiring design is: nIN0 is connected to the NEAR HOME (NHOME) sensors; nIN1 is connected to the HOME sensors; and nIN2 is connected to the index of Encoder input (Z phase).**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 19/0C 19 | *Sub_function code* |
| 1 | 00 0F | *axis (F ➔ AXIS_XYZU)* |
| 2 | 00 15 | *FEn (21 = 0x15, enable several noise filters)* |
| 3 | 00 03 | *FLn (using filter 3)* |

## 2.16 Position Counter Variable Ring

**eRET ETM_VRING_ENABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *nVRing*)**

**※ eRET ETM_MACRO_VRING_ENABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD nVRing)**

**Description:**

**This function enables the linear counter of the assigned axes as variable**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |
| ***nVRing*:** | **Maximum value of the ring counter (0 ~ +2,000,000,000)** |

**Return:**

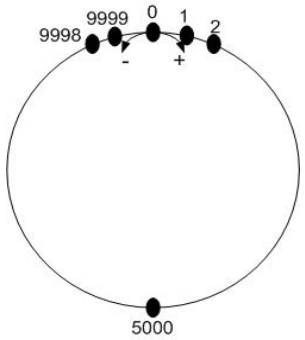
**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_VRING_ENABLE is 0A 1A.**
**The Sub_function code of ETM_MACRO_VRING_ENABLE is 0C 1A.**

**MODBUS example:**

```
ETM_VRING_ENABLE (h, 1, AXIS_X, 9999);
//set the X axis of module 1 to be a ring counter. The encoder
//values will be 0 to 9999.
```




**Max. ring encoder value = 9999**

**The encoder value ranges from 0 to 9999. When the counter value reaches 9999, one more adding pulse will cause the counter to reset to 0. When the counter value is 0, a lessening pulse will let the counter set to 9999.**

**Note: 1. This function will set the LP and EP simultaneously.**
**2. If this function is enabled, the software limit function cannot be used.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 1A/0C 1A | *Sub_function code* |
| 1 | 00 01 | *axis* |
| 2 | 00 00 | *MSW of nVRing* |
| 3 | 27 0F | *LSW of nVRing (9999 = 0x270F)* |

**eRET ETM_VRING_DISABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_VRING_DISABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function disables the variable ring counter function.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 or Table 3)**<br><br>**If *axis* setting is defined in Table 2, the setting values will change current speed settings.**<br><br>**If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_VRING_DISABLE is 0A 1B.**
**The Sub_function code of ETM_MACRO_VRING_DISABLE is 0C 1B.**

**MODBUS example:**

**ETM_VRING_DISABLE (h, 1, AXIS_X);**
**//disable the ring counter function for the X axis on module 1.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 1B/0C 1B | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| | | |

## 2.17 Triangle prevention of fixed pulse driving

**eRET ETM_AVTRI_ENABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_AVTRI_ENABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function prevents a triangle form in linear acceleration (T-curve) fixed**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis:* | Axis or axes (Table 2 or Table 3)<br><br>If *axis* setting is defined in Table 2, the setting values will change current speed settings.<br><br>If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_AVTRI_ENABLE is 0A 1C.**
**The Sub_function code of ETM_MACRO_AVTRI_ENABLE is 0C 1C.**

**MODBUS example:**

```
ETM_AVTRI_ENABLE (h, 1, AXIS_X);
//set the X axis of module 1 not to generate a triangle form in its speed profile.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 1C/0C 1C | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| | | |

**eRET ETM_AVTRI_DISABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_AVTRI_DISABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function disables the function to prevent a triangle form in linear acceleration fixed pulse driving.**

**Category:**

**MODBUS sub_function; RTC, MP and IT.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis:* | Axis or axes (Table 2 or Table 3)<br><br>If *axis* setting is defined in Table 2, the setting values will change current speed settings.<br><br>If *axis* setting is defined in Table 3, the setting values will be writted into the parameter table without changing current speed settings. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_AVTRI_DISABLE is 0A 1D.**
**The Sub_function code of ETM_MACRO_AVTRI_DISABLE is 0C 1D.**

**MODBUS example:**

```
ETM_AVTRI_DISABLE (h, 1, AXIS_X);
//enable the X axis of module 1 to generate a triangle form in its
//speed profile if the pulse number for output is too low.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 1D/0C 1D | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| | | |

## 2.18 External Pulse Input

**Cannot write settings for external input driving into the parameter Table.**

### 2.18.1 Handwheel (Manual Pulse Generator) Driving

**eRET ETM_EXD_MP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function outputs pulses according to the input pulses from a handwheel.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

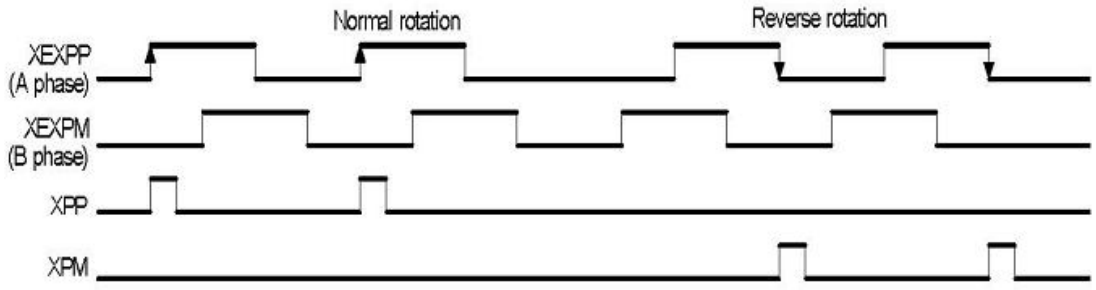
| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Table 2 )<br>The axis can be either X,Y, Z or U.** |
| ***data*:** | **Number of command pulses<br>(Range: 0 ~ +2,000,000,000).** |

**Return:**

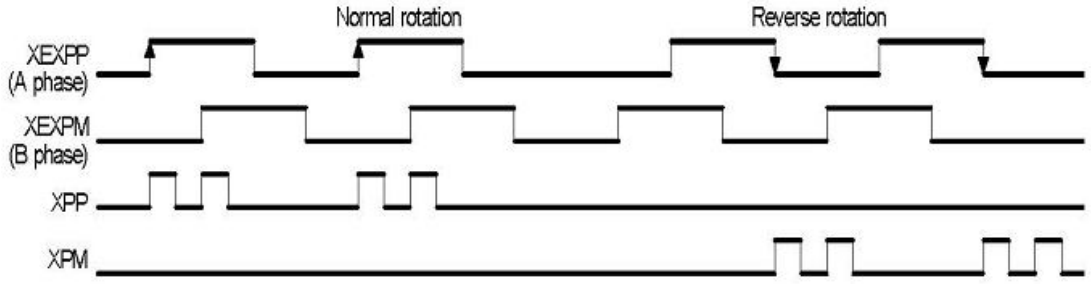
**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_EXD_MP (h, 1, AXIS_X, 1);
//Each time the handwheel inputs a pulse to the X axis
on module 1, the controller will output 1 pulse to the motor driver.
```



**ETM_EXD_MP (h, 1, AXIS_X, 2);**
**//Each time the handwheel inputs a pulse to the X axis**
**//on module 1, the controller will output 2 pulses to the motor driver.**




**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 1E | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 00 01 | *LSW of data* |

## 2.18.2 Fixed Pulse Driving Mode

**eRET ETM_EXD_FP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function outputs fixed pulses according to the trigger input (the falling edge of the nEXP+ signal) from a handwheel.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

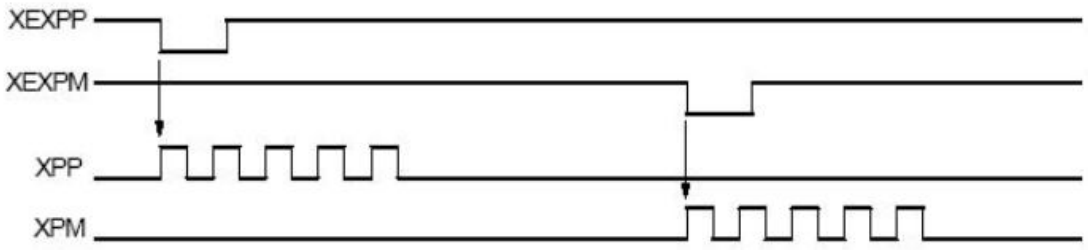
| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***data*:** | **Number of command pulses<br>(Range: 0 ~ +2,000,000,000).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_EXD_FP (h, 1, AXIS_X, 5);
//Each time the controller detects a falling edge of an XEXP+
//signal, it will output 5 pulses to the X axis.
```



**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 1F | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 00 05 | *LSW of data* |

### 2.18.3 Continuous Pulse Driving Mode

**eRET ETM_EXD_CP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**The controller will continuously output pulses in positive direction if the falling edge of nEXP+ signal from a handwheel is detected. On the contrary, it will continuously output pulses in negative direction if the falling edge of nEXPsignal is detected.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

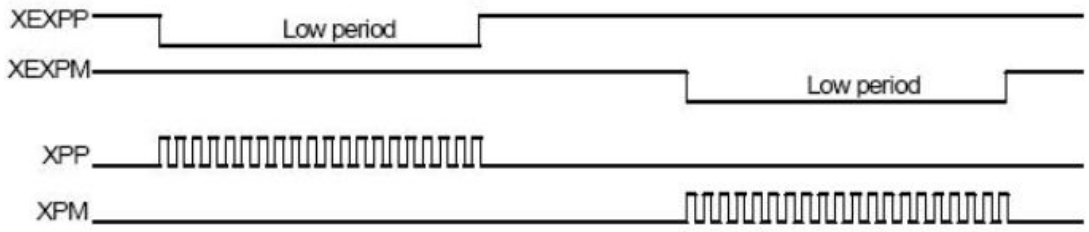
| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***data*:** | **Number of command pulses<br>(Range: 0 ~ +2,000,000,000).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_EXD_CP (h, 1, AXIS_X, 20);
//Each time the controller detects a falling edge of an XEXP+
signal, it will continuously drive X axis at the speed of 20 PPS.
```



**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 20 | *Sub_function code* |
| 1 | 00 01 | *axis* |
| 2 | 00 00 | *MSW of data* |
| 3 | 00 14 | *LSW of data (20 = 0x14)* |

## 2.18.4 Disabling the External Signal Input Functions

**eRET ETM_EXD_DISABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function turns off the external input driving control functions.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis:* | Axis or axes. Please refer to Table 2.<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_EXD_DISABLE (h, 1, AXIS_X);
//disable the external input driving control function
of X axis on module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 21 | *Sub_function code* |
| 1 | 00 01 | *axis (1 → AXIS_X)* |
| | | |

## 2.19 Read/Write User-defined Variables (VAR and bVAR)

**eRET ETM_READ_bVAR (HANDLE *h*, BYTE *cardNo*, BYTE *bvarNo*, BYTE* bVARValue)**

**Description:**

**This function read variable bVARn, it could be passed to Macro Program(MP), for detail information, please refer to Chapter 7.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***bvarNo*:** | **custom variable: bVAR0 ~ bVAR127** |
| ***bVARValue*:** | **The pointer to the memory that stores bVARn (0 ~ +255).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The corresponding holding register modbus address is as follow:**

| Macro Variable | | | |
|---|---|---|---|
| **Variable Name** | **Address** | **Type** | **Comment** |
| bVAR0 | 128 | R/W | Value of bVAR0. |
| bVAR1 | 129 | R/W | Value of bVAR1. |
| … | … | | |
| bVAR126 | 254 | R/W | Value of bVAR126. |
| bVAR127 | 255 | R/W | Value of bVAR127. |

**MODBUS example:**

**Suppose the value of bVAR100 is 100, or 0x64 .**
**The hexadecimal value of (128+100) is E4.**

```
BYTE bVARValue;
ETM_READ_bVAR (h, 1, bVAR100, &bVARValue);
//read value of VAR100 in module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 00 E4 | 00 01 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 Value (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 03 | 02 | 00 64 |

**eRET ETM_WRITE_bVAR (HANDLE *h*, BYTE *cardNo*, BYTE *bvarNo*, BYTE *bVar*)**

**Description:**

**This function write variable bVARn. bVARn could be passed to Macro Program (MP). For detail information, please refer to Chapter 8. bVAR parameter data are stored in a volatile memory.**

**Category:**

**MODBUS table, MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***bvarNo*:** | **custom variable: bVAR0 ~ bVAR127** |
| ***bVar*:** | **variable (0 ~ +255).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The corresponding holding register modbus address is as follow:**

| Macro Variable | | | |
|---|---|---|---|
| **Variable Name** | **Address** | **Type** | **Comment** |
| bVAR0 | 128 | R/W | Value of bVAR0. |
| bVAR1 | 129 | R/W | Value of bVAR1. |
| … | … | | |
| bVAR126 | 254 | R/W | Value of bVAR126. |
| bVAR127 | 255 | R/W | Value of bVAR127. |

**MODBUS example:**

- **Method 1: use Sub_Function code method：**

**ETM_WRITE_bVAR (h, 1, bVAR100, 100);**
**//write bVAR100=100 in module 1**
**The address of bVARn = 128 + n (or 0x80 + n).**
**( bVAR100 = 128 + 100 = 228 = 0xE4 )**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 23 | *Sub_function code* |
| 1 | 00 E4 | *bvarNo = n + 0x80 = 0xE4* |
| 2 | 00 64 | *bVar (100 = 0x64)* |

■ **Method 2: Set the Holding Register table：**

**All bVARn are defined in the holding register table. The index of bVARn is (128+n); therefore, the index, or address, of bVAR100 is 228 (= 0xE4).**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 00 E4 | 00 01 | 02 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 00 64 | *bVar (100 = 0x64)* |

**eRET *ETM*_READ_VAR (HANDLE *h*, BYTE *cardNo*, long *varNo*, long* *VARValue*)**

**Description:**

**This function read variable VARn, it could be passed to Macro Program (MP), for detail information, please refer to Chapter 7.4.1.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *varNo*: | custom variable: VAR0 ~ VAR511 |
| *VARValue*: | The pointer to the memory that stores VARn (-2,000,000,000 ~ +2,000,000,000). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**All VARn are defined in the holding register table. Each VARn takes two registers space. For example, VARn takes both registers indexed by (300 + 2*n) and (300 +2*n + 1). To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.**

**The Start_Address (index) = 300 + 2*100 = 500 = 0x01F4**

```
long VARValue;
ETM_READ_VAR (h, 1, VAR100, &VARValue);
//read value of VAR100 in module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 01 F4 | 00 02 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 (hex) | Reg_1 (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 03 | 04 | 00 00 | 27 10 |

**To get the ldata, use the following instructions in C language.**

```
ldata = Register[0];
ldata = ((ldata <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

**eRET ETM_WRITE_VAR (HANDLE *h*, BYTE *cardNo*, long *varNo*, long *lVar*)**

**Description:**

**This function write variable VARn. It could be passed to Macro Program (MP). For detail information, please refer to Chapter 7.4.1. VAR parameter data are stored in a volatile memory.**

**Category:**

**MODBUS table, MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *varNo*: | custom variable: VAR0 ~ VAR511 |
| *lVar*: | Value of variable (-2,000,000,000 ~ +2,000,000,000). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_WRITE_VAR (h, 1, VAR100, 10000);
//write VAR100=10,000 in module 1
```

■ **Method 1: use Sub_Function code method**

**The address of VARn = 0x7FFF0000 + n.**
**→ the address of VAR100 = 0x7FFF0000 + 0x64**
**And, 10000 = 0x2710**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 25 | *Sub_function code* |
| 1 | 7F FF | *MSW of varNo (MSW of 0x7FFF0064)* |
| 2 | 00 64 | *LSW of varNo (LSW of 0x7FFF0064)* |
| 3 | 00 00 | *MSW of lVar (MSW 0f 0x2710)* |
| 4 | 27 10 | *LSW of lVar (LSW 0f 0x2710)* |

■ **Method 2: Set the Holding Registers.**

**All VARn are defined in the holding register table. Each VARn takes two registers space. For example, VARn takes both registers indexed by (300 + 2*n) and (300 +2*n + 1). To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.**

**For VARn, the Start address = (300 + 2*n).**
**➔The Start_Address = 300 + 2*100 = 500 = 0x01F4**

**Suppose the value of VAR100 is 10000 (= 0x2710).**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 01 F4 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 00 00 | *MSW of lVar (MSW 0f 0x2710)* |
| 1 | 27 10 | *LSW of lVar (LSW 0f 0x2710)* |

## 2.20 Read/Write Data for Power outage carry-over (MD)

**eRET ETM_READ_MD (HANDLE *h*, BYTE *cardNo*, long *mdNo*, long* *ldata*, float* *fdata*)**

**Description:**

**This function reads the machine data. MD data are stored in a non-volatile memory.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***mdNo*:** | **Machine data(long): MD0 ~ MD1023<br>Machine data (float): MD1024 ~ MD2047** |
| **&*ldata*:** | **Read MD long (-2,147,483,648 ~ +2,147,483,647)** |
| **&*fdata*:** | **Read MD float (Integer number plus decimal number giving a total of six places).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**All MDn are defined in the holding register table. Each MDn takes two registers space. For example, MDn takes both registers indexed by (3000 + 2*n) and (3000 +2*n + 1). To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.**

**The MD100 Start_Address = 3000 + 2*100 = 3200 = 0x0C80**
**The MD1500 Start_Address = 3000 + 2*1500 = 6000 = 0x1770**

```
long ldata;
float fdata;
ETM_READ_MD (h, 1, MD100, &ldata, 0);
//read MD100 long data in module 1
ETM_READ_MD(h, 1, MD1500, 0, &fdata);
//read MD1500 float data in module 1
```

**ldata: the MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 0C 80 | 00 02 |

**ldata: The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 (hex) | Reg_1 (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 03 | 04 | 12 34 | 56 78 |

**fdata: the MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 17 70 | 00 02 |

**fdata: The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 (hex) | Reg_1 (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 03 | 04 | 43 A0 | D3 33 |

**To get the ldata, use the following instructions in C language.**

```
ldata = Register[0];
ldata = ((ldata <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

**eRET ETM_WRITE_MD (HANDLE *h*, BYTE *cardNo*, long *mdNo*, long *ldata*, float *fdata*)**

**Description:**

**This function writes the machine data.**

**Category:**

**MODBUS table, MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***mdNo*:** | **Machine data(long): MD0 ~ MD1023**<br>**Machine data (float): MD1024 ~ MD2047** |
| ***&ldata*:** | **Write MD long (-2,147,483,648 ~ +2,147,483,647)** |
| ***&fdata*:** | **Write MD float (Integer number plus decimal number giving a total of six places).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_WRITE_MD (h, CardNo, MD100, 0x12345678, 0);
//write MD100 long data in module 1
```

■ **Method 1: use Sub_Function code method**

**The address of MDn = 0x00000000 + n.**
**→ the address of MD100 = 0x00000000 + 0x64**

**Write MD100 = 0x12345678**
**Write MD1500 = 321.65**

ldata: the MODBUS command is listed as follows:

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 15 | 01 | 10 | 1F 40 | 00 07 | 0E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 27 | *Sub_function code* |
| 1 | 00 00 | *MSW of mdNo (= 0)* |
| 2 | 00 64 | *LSW of mdNo (n= 0x64)* |
| 3 | 12 34 | *MSW of ldata* |
| 4 | 56 78 | *LSW of ldata* |
| 5 | 00 00 | *MSW of fdata* |
| 6 | 00 00 | *LSW of fdata* |

fdata: the MODBUS command is listed as follows:

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 15 | 01 | 10 | 1F 40 | 00 07 | 0E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 27 | *Sub_function code* |
| 1 | 00 00 | *MSW of mdNo (= 0)* |
| 2 | 05 DC | *LSW of mdNo (n= 0x05DC)* |
| 3 | 00 00 | *MSW of ldata* |
| 4 | 00 00 | *LSW of ldata* |
| 5 | 43 A0 | *MSW of fdata* |
| 6 | D3 33 | *LSW of fdata* |

■ **Method 2: Set the Holding Registers.**

**All MDn are defined in the holding register table. Each MDn takes two registers space. For example, MDn takes both registers indexed by (3000 + 2*n) and (3000 +2*n + 1). To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.**

**➔The Start_Address = 3000 + 2*100 = 3200 = 0x0C80**

ldata: the MODBUS command is listed as follows:

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 0C 80 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 12 34 | *MSW of MD100* |
| 1 | 56 78 | *LSW of MD100* |

➔The Start_Address = 3000 + 2*1500 = 6000 = 0x1770

ldata: the MODBUS command is listed as follows:

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 17 70 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 43 A0 | *MSW of MD1500* |
| 1 | D3 33 | *LSW of MD1500* |

# 3 Reading and Register Setting

## 3.1 Setting and Reading Command Position

**eRET ETM_SET_LP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *wdata*)**

**※Δ eRET ETM_MACRO_SET_LP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *wdata*)**

**Description:**

**This function sets the command position counter value (logical position counter, LP).**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *wdata*: | Position command<br>(-2,000,000,000 ~ +2,000,000,000) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_LP is 0A 28.**
**The Sub_function code of ETM_MACRO_SET_LP is 0C 28.**

**MODBUS example:**

```
ETM_SET_LP (h, 1, AXIS_XYZU, 10000);
//Set the LP as 0 for X, Y, Z, and U axes in module 1
// will clear all LP counters on module 1
```

■ **Method 1: use sub_function code method**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 28/0C 28 | *Sub_function code* |
| 1 | 00 0F | *axis (F ➔ AXIS_XYZU)* |
| 2 | 00 00 | *MSW of wdata* |
| 3 | 27 10 | *LSW of wdata* |
| | | |

■ **Method 2: Set the holding registers.**

**The Start address is 90, or 0x5A, for LP_X. Since the LP needs 2 registers to hold the value, the start address setting for MODBUS request must begins from the MSW of the required LP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.**

**Method 2 allows users to set different values for 4 different axes at only one MODBUS request, which is not possible by using method 1.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 17 | 01 | 10 | 00 5A | 00 08 | 10 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 00 00 | *LP_X_MSW (address = 0x5A)* |
| 1 | 27 10 | *LP_X_LSW* |
| 2 | 00 00 | *LP_Y_MSW (address = 0x5C)* |
| 3 | 27 10 | *LP_Y_LSW* |
| 4 | 00 00 | *LP_Z_MSW (address = 0x5E)* |
| 5 | 27 10 | *LP_Z_LSW* |
| 6 | 00 00 | *LP_U_MSW (address = 0x60)* |
| 7 | 27 10 | *LP_U_LSW* |
| | | |

**eRET ETM_GET_LP(HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long* *LPValue*)**

**※Δ eRET ETM_MACRO_GET_LP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**eRET ETM_GET_LP_4_AXIS(HANDLE *h*, BYTE *cardNo*, long* *LpValueX*, long* *LpValueY*, long* *LpValueZ*, long* *LpValueU*)**

**Description:**

**This function reads the command position counter value (logical position counter, LP).**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *LPValue*: | The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000). |
| *LpValueX*: | The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of x axis. |
| *LpValueY*: | The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of y axis. |
| *LpValueZ*: | The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of z axis. |
| *LpValueU*: | The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of u axis. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

- **Method 1: Use Holding Registers to get the current values.**

**Since each LP needs 2 registers to hold the value, the start address setting for MODBUS request must begins from the MSW of the required LP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.**

**This method allows users to get more than one axis data at only one MODBUS request. Please refer to MODBUS holding registers definition table.**

```
long X_LP;
ETM_GET_LP (h, 1, AXIS_X, &X_LP);
//Reads the LP value of the X axis on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 00 5A | 00 02 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 (hex) | Reg_1 (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 03 | 04 | 00 00 | 27 10 |

**To get the X_LP, use the following instructions in C language.**

```
X_LP = Register[0];
X_LP = ((X_LP <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

- **Method 2: It is used as an MP instruction.**

**For this case, the current LP value does not actually be returned at the moment when request is sent. The getting LP instruction is desired to be executed later when an MP is called. Users can use FC = 16 to write this command into an MP. Inside this MP, there can be an ETM_MACRO_SET_RVAR() function just next to it to save the return LP value to a specified variable. This variable can be referred by other functions. Please refer to other related MP usages explanations.**

```
ETM_MACRO_GET_LP (h, 1, AXIS_X);
//Reads the LP value of the X axis on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 29 | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| | | |

## 3.2 Setting and Reading the Encoder Counter

**eRET ETM_SET_EP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *wdata*)**

**※Δ eRET ETM_MACRO_SET_EP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *wdata*)**

**Description:**

**This function sets the encoder position counter value (real position counter or EP).**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***wdata*:** | **Position command<br>(-2,000,000,000 ~ +2,000,000,000)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_EP is 0A 2A.**
**The Sub_function code of ETM_MACRO_SET_EP is 0C 2A.**

**MODBUS example:**

**ETM_SET_EP (h, 1, AXIS_XYZU, 10000);**
**//Set the EP as 10000 for X, Y, Z, and U axes of module 1.**

■ **Method 1: use Sub_Function code method**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 2A/0C 2A | *Sub_function code* |
| 1 | 00 0F | *axis (F ➔ AXIS_XYZU)* |
| 2 | 00 00 | *MSW of wdata* |
| 3 | 27 10 | *LSW of wdata* |
| | | |

■ **Method 2: Set the Holding Registers.**

**Since each EP needs 2 registers to hold the value, the start address setting for MODBUS request must begins from the MSW of the required EP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.**

**This method allows users to set more than one axis data at only one MODBUS request. Please refer to MODBUS holding registers definition table.**

**The start address is 98, or 0x62, for EP_X.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 17 | 01 | 10 | 00 62 | 00 08 | 10 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 00 00 | *LP_X_MSW (address = 0x5A)* |
| 1 | 27 10 | *LP_X_LSW* |
| 2 | 00 00 | *LP_Y_MSW (address = 0x5C)* |
| 3 | 27 10 | *LP_Y_LSW* |
| 4 | 00 00 | *LP_Z_MSW (address = 0x5E)* |
| 5 | 27 10 | *LP_Z_LSW* |
| 6 | 00 00 | *LP_U_MSW (address = 0x60)* |
| 7 | 27 10 | *LP_U_LSW* |
| | | |

**eRET ETM_GET_EP (HANDLE *h*, BYTE *cardNo*, BYTE *axis* , long* *EPValue*)**

**※Δ eRET ETM_MACRO_GET_EP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**eRET ETM_GET_EP_4_AXIS (HANDLE *h*, BYTE *cardNo*, long* *EpValueX*, long* *EpValueY*, long* *EpValueZ*, long* *EpValueU*)**

**Description:**

**This function reads the encoder position counter value (EP).**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***EPValue*:** | **The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~ +2,000,000,000).** |
| ***EpValueX:*** | **The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of x axis.** |
| ***EpValueY:*** | **The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of y axis.** |
| ***EpValueZ:*** | **The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of z axis.** |
| ***EpValueU:*** | **The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of u axis.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**Suppose the EP is 10000, or 0x2710. The Start address is 98, or 0x62. Since the EP needs 2 registers to hold the value, the start address setting for MODBUS request must begins from the MSW of the required EP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.**

- **Method 1: Use holding registers to get the current values.**

```
long X_EP;
ETM_GET_EP (h, 1, AXIS_X, &X_EP);
//reads the encoder position value (EP) of the X axis on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 00 62 | 00 02 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 (hex) | Reg_1 (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 03 | 04 | 00 00 | 27 10 |

**To get the X_EP, use the following instructions in C language.**

```
X_EP = Register[0];
X_EP = ((X_EP <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

- **Method 2: It is used for creating the content of an MP.**

**For this case, users do not actually want to get the current EP values. The getting EP will be executed only when the MP is called. Therefore, use FC = 16 to write this command inside a MP. This kind of usages often has ETM_MACRO_SET_RVAR() followed to get the return EP value. Please refer to MP related explanation literature.**

```
ETM_MACRO_GET_EP (h, 1, AXIS_X);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 2B | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| | | |

## 3.3 Reading the Current Velocity

**eRET ETM_GET_CV (HANDLE h, BYTE cardNo, BYTE axis, long* CVValue)**

**eRET ETM_GET_CV_4_AXIS (HANDLE h, BYTE cardNo, long* CvValueX, long* CvValueY, long* CvValueZ, long* CvValueU)**

**Description:**

**This function reads the current velocity value.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo***: | Module number → MP-8000 : 1~7 |
| ***axis***: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| ***CVValue:*** | The pointer to the memory that stores the current velocity |
| ***CvValueX:*** | The pointer to the memory that stores the current velocity of x axis |
| ***CvValueY:*** | The pointer to the memory that stores the current velocity of y axis |
| ***CvValueZ:*** | The pointer to the memory that stores the current velocity of z axis |
| ***CvValueU:*** | The pointer to the memory that stores the current velocity of u axis |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**Suppose the CV is 10000, or 0x2710. The Start address is 106, or 0x6A.**

```
long CVValue;
ETM_GET_CV (h, 1, AXIS_X, &CVValue);
//reads the current velocity of the X axis on module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 00 6A | 00 02 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 (hex) | Reg_1 (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 03 | 04 | 00 00 | 27 10 |

**To get the CV_X, use the following instructions in C language.**

```
CV_X = Register[0];
CV_X = ((CV_X <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

## 3.4 Reading the Current Acceleration

**eRET ETM_GET_CA (HANDLE *h*, BYTE *cardNo*, BYTE *axis* , long* *CAValue*)**

**eRET ETM_GET_CA_4_AXIS (HANDLE *h*, BYTE *cardNo*, long* *CaValueX*, long* *CaValueY*, long* *CaValueZ*, long* *CaValueU*)**

**Description:**

**This function reads the current acceleration value PPS/sec.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***CAValue:*** | **The pointer to the memory that stores the current acceleration (in PPS/Sec).** |
| ***CaValueX:*** | **The pointer to the memory that stores the current acceleration (in PPS/Sec) of x axis.** |
| ***CaValueY:*** | **The pointer to the memory that stores the current acceleration (in PPS/Sec) of y axis.** |
| ***CaValueZ:*** | **The pointer to the memory that stores the current acceleration (in PPS/Sec) of z axis.** |
| ***CaValueU:*** | **The pointer to the memory that stores the current acceleration (in PPS/Sec) of u axis.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**Suppose the CA is 10000, or 0x2710. The Start address is 114, or 0x72.**

```
long CAValue;
ETM_GET_CA (h, 1, AXIS_X, &CAValue);
//reads the current acceleration value of the X axis on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 03 | 00 72 | 00 02 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Reg_0 (hex) | Reg_1 (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 03 | 04 | 00 00 | 27 10 |

**To get the CA_X, use the following instructions in C language.**

```
CA_X = Register[0];
CA_X = ((CA_X <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

## 3.5 Reading the DI Status

**※Δ eRET ETM_MACRO_GET_DI (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE nType)**

**Description:**

**This function reads the digital input (DI) status.**

**Category:**

**MODBUS sub_function; MP, ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo***: | Module number → MP-8000 : 1~7 |
| ***axis***: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| ***nType***: | 0 → DRIVING (Check whether the axis is driving or not)<br>1 → LIMIT+ (Check whether the limit+ is engaged or not)<br>2 → LIMIT- (Check whether the limit- is engaged or not)<br>3 → EMERGENCY (Check whether EMG signal is on or not)<br>4 → ALARM (Check the ALARM input signal)<br>5 → IN1 (Check the IN1 input signal)<br>6 → IN0 (Check the IN0 input signal)<br>7 → IN3 (Check the IN3 input signal)<br>8 → INPOS (Check the INPOS input signal)<br>9 →IN2 (Check the IN2 input signal) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

- **It is used for creating the content of an MP.**

**For this case, users do not actually want to get the current DI values. The getting DI will be executed only when the MP is called. Therefore, use FC = 16 to write this command inside a MP. This kind of usage often has ETM_MACRO_SET_RVAR() followed to get the return DI value. Please refer to MP related explanation literature.**

**ETM_MACRO_GET_DI (h, 1, AXIS_X, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 2E | *Sub_function code* |
| 1 | 00 01 | *axis* |
| 2 | 00 01 | *The specified DI (LIMIT+)* |

**eRET ETM_GET_DI_ALL (HANDLE *h*, BYTE *cardNo*, BYTE *axis* , WORD* DIValue)**

**※Δ eRET ETM_MACRO_GET_DI_ALL (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**eRET ETM_GET_DI_ALL_4_AXIS (HANDLE *h*, BYTE *cardNo*, WORD* *DiValueX*, WORD* *DiValueY*, WORD* *DiValueZ*, WORD* *DiValueU*)**

**Description:**

**This function reads the digital input (DI) status.**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***DIValue:*** | **The pointer to the memory that stores the all DI Status<br>The bits in *DIValue* are defined as follows:<br>*Bit* 0 → DRIVING (Axis is driving or not)<br>*Bit* 1 → LIMIT+ (Limit+ is engaged or not)<br>*Bit* 2 → LIMIT- (Limit- is engaged or not)<br>*Bit* 3 → EMERGENCY (EMG signal)<br>*Bit* 4 → ALARM (ALARM signal)<br>*Bit* 5 → IN1 (IN1 input signal, low-active)<br>*Bit* 6 → IN0 (IN0 input signal,low-active)<br>*Bit* 7 → IN3 (IN3 input signal,low-active)<br>*Bit* 8 → INPOS (INPOS input signal)<br>*Bit* 9 →IN2 (IN2 input signal, low-active)** |
| ***DiValueX*:** | **The pointer to the memory that stores the all DI Status of x axis** |
| ***DiValueY*:** | **The pointer to the memory that stores the all DI Status of y axis** |
| ***DiValueZ*:** | **The pointer to the memory that stores the all DI Status of z axis** |
| ***DiValueU*:** | **The pointer to the memory that stores the all DI Status of u axis** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

- **Method 1: It is used for getting current status of DI.**

  **Each bit corresponds to a register. Each register's value is either 0 or 1.**

  **WORD *DIValue*;**
  **ETM_GET_DI_ALL (h, 1, AXIS_X, &DIValue);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 10 | 00 01 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | DI Status of AXIS_X. ( Reg_0 ) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 00 |

- **Method 2: When it is used inside an MP.**

**For this case, users do not actually want to get the current DI values. The getting DI will be executed only when the MP is called. Therefore, use FC = 16 to write this command inside a MP. This kind of usage often has ETM_MACRO_SET_RVAR() followed to get the return DI value. Please refer to MP-related explanation literature.**

```
ETM_MACRO_GET_DI_ALL (h, 1, AXIS_X);
//ETM_MACRO_SET_RVAR(h, 1, VAR0);  //assign this DI value to
//VAR0
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 31 | *Sub_function code* |
| 1 | 00 01 | *axis (1: AXIS_X)* |
| | | |

## 3.6 Reading and Clearing the ERROR Status

**eRET ETM_GET_ERROR_STATE (HANDLE *h*, BYTE *cardNo*, BYTE* *ErrState*)**

**※Δ eRET ETM_MACRO_GET_ERROR (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function checks whether an error occurs or not.**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***ErrState:*** | **The pointer to the memory that stores the error status:**<br><br>**1: Error(s) occured.**<br>**Please perform ETM_GET_ERROR_CODE () to get more information. If *GET_ERROR_CODE =256,* it indicates that the motion stop was due to the "STOP" command, not because of an error happened. Please refer to Section 6.5.5 and the following Example for cleaning ERROR.**<br><br>**0: No error.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

- **Method 1: It is used for getting current status of error.**

```
BYTE ErrState;
ETM_GET_ERROR_STATE (h, 1, &ErrState);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 46 | 00 01 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Error status ( Reg_0 ) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 00 |

- **Method 2: It is used for creating the content of an MP.**

**For this case, users do not actually want to get the current error status. The getting error will be executed only when the MP is called. Therefore, use FC = 16 to write this command inside a MP. This kind of usage often has ETM_MACRO_SET_RVAR() followed to get the return value. Please refer to MP related explanation literature.**

**ETM_MACRO_GET_ERROR (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0C 2F | | *Sub_function code* | | | |
| | | | | | | | |

**eRET ETM_GET_ERROR_CODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, WORD* *ErrCode*)**

**※Δ eRET ETM_MACRO_GET_ERROR_CODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**eRET ETM_GET_ERROR_CODE_4_AXIS (HANDLE *h*, BYTE *cardNo*, WORD* *ErrCodeX*, WORD* *ErrCodeY*, WORD* *ErrCodeZ*, WORD* *ErrCodeU*)**

**Description:**

**This function reads the ERROR status.**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***ErrCode:*** | **The pointer to the memory that stores the error status** |
| ***ErrCodeX:*** | **The pointer to the memory that stores the error status of x axis** |
| ***ErrCodeY:*** | **The pointer to the memory that stores the error status of y axis** |
| ***ErrCodeZ:*** | **The pointer to the memory that stores the error status of z axis** |
| ***ErrCodeU:*** | **The pointer to the memory that stores the error status of u axis** |

***ErrCode:***

**0→ No error.**

**For non-zero return value, please refer to the following table. If there are more than one error, the return value will be the sum of there error code values.**

| Erro Code | Cause of stop | Explanation |
|---|---|---|
| 1 | SOFT LIMIT+ | Occurs when the forward software limit is asserted |
| 2 | SOFT LIMIT- | Occurs when the reverse software limit is asserted |
| 4 | LIMIT+ | Occurs when the forward hardware limit is asserted |
| 8 | LIMIT- | Occurs when the reverse hardware limit is asserted |
| 16 | ALARM | Occurs when the ALARM is asserted |
| 32 | EMERGENCY | Occurs when the EMG is asserted |
| 64 | Reserved | Reserved |
| 128 | HOME | Occurs when both Z phase and HOME are asserted |
| 256 | Refer to 6.5.4 | Occurs when the EMG(software) is asserted |

**For example, a return code 48 means that ALARM and EMGERENCY occur at the same time.**

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

- **Method 1: It is used for getting current error code.**

**WORD *ErrCode*;**
**ETM_GET_ERROR_CODE (h, 1, AXIS_X, &*ErrCode*);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 14 | 00 01 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | ErrCode ( Reg_0 ) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 00 |

■ **Method 2: It is used for creating the content of an MP.**

**For this case, users do not actually want to get the current error codes. The getting error code will be executed only when the MP is called. Therefore, use FC = 16 to write this command inside a MP. This kind of usage often has ETM_MACRO_SET_RVAR() followed to get the return value. Please refer to MP related explanation literature.**

**ETM_MACRO_GET_ERROR_CODE (h, 1, AXIS_X);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 30 | *Sub_function code* |
| 1 | 00 01 | *axis (AXIS_X)* |

## 3.7 Read RTC status

**BYTE i8094H_CHECK_RTC(BYTE *cardNo*)**

**Description:**

**This function reads current RTC status.**

**Category:**

**Internal function.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**YES: The buffer is full**
**NO: The buffer is not full**

**EXAMPLE:**

```
if (i8094H_CHECK_RTC(1) == YES)
    {
      //buffer is full
      // please add your code here to take care of the exceptional cases.
}
```

**Special Description:**

**This function does not support on the ET-M8194H. Similar function can be implemented by using ETM_GET_FREE_BUFFER() function.**

**eRET ETM_GET_FREE_BUFFER(HANDLE *h*, BYTE *cardNo*, BYTE* FreeBufNum)**

**Description:**

**This function gets the block-number of available command-buffer. The maximum block-number is 30.**

**Category:**

**MODBUS table；RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *FreeBufNum*: | The pointer points to the memory that stores the available block number of i-8094H |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
BYTE FreeBufNum;
ETM_GET_FREE_BUFFER(h, 1, & FreeBufNum);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 06 | 00 01 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | *Block_Num* (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 1E |

**The retured value, 0x1E, means 30 blocks of command-buffer are available.**

## 3.8 Read stop status

**eRET ETM_GET_STOP_STATUS(HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE* *StopState*)**

**eRET ETM_*GET*_STOP_STATUS_4_AXIS(HANDLE *h*, BYTE *cardNo*, BYTE* *StopStateX*, BYTE* *StopStateY*, BYTE* *StopStateZ*, BYTE* *StopStateU*)**

**Description:**

**This function reads current stop status.**

**Category:**

**MODBUS table；RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***StopState*:** | **The pointer to the memory that stores the stop status** |
| ***StopStateX*:** | **The pointer to the memory that stores the stop status of x axis** |
| ***StopStateY*:** | **The pointer to the memory that stores the stop status of y axis** |
| ***StopStateZ*:** | **The pointer to the memory that stores the stop status of z axis** |
| ***StopStateU*:** | **The pointer to the memory that stores the stop status of u axis** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

| STOP status | Address | Remarks |
|---|---|---|
| X_STOP_STATUS | 58 (0x3A) | 1 ➔ AXIS_X has stopped<br>0 ➔ AXIS_X is moving |
| Y_STOP_STATUS | 59 (0x3B) | 1 ➔ AXIS_Y has stopped<br>0 ➔ AXIS_Y is moving |
| Z_STOP_STATUS | 60 (0x3C) | 1 ➔ AXIS_Z has stopped<br>0 ➔ AXIS_Z is moving |
| U_STOP_STATUS | 61 (0x3D) | 1 ➔ AXIS_U has stopped<br>0 ➔ AXIS_U is moving |

```
BYTE StopState;
ETM_GET_STOP_STATUS(h, 1, AXIS_X, & StopState);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 3A | 00 01 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | AXIS Stop_Status (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 01 |

**Users can get these states by polling and decide what to do next in following program.**

# 4 FRnet Functions

## 4.1 Read FRnet DIO Signals

※Δ eRET ETM_MACRO_FRNET_IN (HANDLE *h*, BYTE *cardNo*, BYTE *wGroup*)

**Description:**

This function reads the FRnet digital input signals. One group comprises 16 bits data. Maximum eight groups of DI are provided. Therefore, total 128 DI can be defined for one FRnet interface.

**Category:**

MODBUS sub_function; MP, ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *wGroup*: | DI group number is between 8~15.<br>If you enter a value between 0 to 7, you can get the output state of the external DO group. |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

Each register contains 16-bit data. Each bit corresponds to a DI channel in a group.

■ It is used for creating the content of an MP.

For this case, users do not actually want to get the current DI value. The getting will be executed only when the MP is called. Therefore, use FC = 16 to write this command inside a MP. This kind of usage often has ETM _MACRO_SET_RVAR() followed to get the return DI value. Please refer to MP related explanation literature.

ETM_MACRO_FRNET_IN (h, 1, 8);

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 32 | *Sub_function code* |
| 1 | 00 08 | *Group number (8~15)* |

## ※Δ eRET ETM_MACRO_FRNET_READ (HANDLE *h*, BYTE *cardNo*, BYTE *bGroup*, DWORD *dwChannel*)

**Description:**

Read the channel data of FRnet group.

**Category:**

MODBUS sub_function; MP, ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *wGroup*: | DI group number is between 8~15.<br>If you enter a value between 0 to 7, you can get the output state of the external DO group. |
| *dwChannel*: | The channel inside the group define the range of 0 to 15. |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

ETM_MACRO_FRNET_READ (h, 1, 8, 1);

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 34 | *Sub_function code* |
| 1 | 00 00 | *MSW of 0x0008* |
| 2 | 00 08 | *LSW of 0x0008* |
| 3 | 00 00 | *MSW of 0x0001* |
| 4 | 00 01 | *LSW of 0x0001* |

**eRET ETM_FRNET_READ_SINGLE_DIO (HANDLE *h*, BYTE *cardNo*, WORD *wGroup*, BYTE *bChannelNo*, BYTE **pbChannelStatus*)**

**Description:**

**Read the channel data of FRnet group.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***wGroup*:** | **DI group number is between 8~15.<br>If you enter a value between 0 to 7, you can get the output state of the external DO group.** |
| ***bChannelNo*:** | **The channel number inside the group define the range of 0 to 15.** |
| ***pbChannelStatus*:** | **The pointer to the memory that stores the data which you read from the channel.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**eRET ETM_FRNET_READ_GROUP_DIO (HANDLE *h*, BYTE *cardNo*, WORD *wGroup*, WORD* *pwGroupStatus*)**

**Description:**

**Read the data of FRnet group.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *wGroup*: | DI group number is between 8~15.<br>If you enter a value between 0 to 7, you can get the output state of the external DO group. |
| *pwGroupStatus* | The pointer to the memory that stores the data which you read from the group. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
WORD pwGroupStatus;
ETM_MACRO_FRNET_READ(h, 1, 8, &pwGroupStatus);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 08 | 00 01 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | *pwGroupStatus* (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 0001 |

**eRET ETM_FRNET_READ_MULTI_GROUP_DIO (HANDLE *h*, BYTE *cardNo*, WORD *wStartGroup*, BYTE *bGroupQty*, WORD* *pwInputBuffer*, BYTE *bBufferSize* )**

**Description:**

**Read the data of FRnet multi-group.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo***: | **Module number → MP-8000 : 1~7** |
| ***wStartGroup:*** | **The starting group.** |
| ***bGroupQty***: | **The group quantity.** |
| ***pwInputBuffer***: | **The pointer to the memory that store the data which you read from the group.** |
| ***bBufferSize***: | **The size of the registers(*bBufferSize**2 must >= *bGroupQty)*** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
WORD pwInputBuffer[2];
ETM_FRNET_READ_MULTI_GROUP_DIO(h, 1, 8, 2, & pwInputBuffer, 4);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 08 | 00 02 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | *pwInputBuffer [0]* (hex) | *pwInputBuffer [1]* (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 04 | 04 | 00 01 | 00 03 |

## 4.2 Write data to FRnet DO

※Δ eRET ETM_MACRO_FRNET_OUT (HANDLE *h*, BYTE *cardNo*, BYTE *bGroup*, DWORD *data*)

**Description:**

This function writes data to the FRnet digital output. One group comprises 16 bits data. Maximum eight groups of DO are provided. Therefore, total 128 DO can be defined for one FRnet interface.

**Category:**

MODBUS table, MODBUS sub_function; MP, ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *bGroup*: | Group number is between 0~7. Eight or above is not defined. |
| *data:* | 0x00000000 ~ 0x0000ffff, can also use VARn to set. |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

- Method 1: It is used for setting FRnet DO values by some true values. Each register contains 16-bit data. Each bit corresponds to a DO of group. Please note that FRnet DO values can be read back.

```
ETM_MACRO_FRNET_OUT (h, 1, 0, 0x0000FFFF);
ETM_MACRO_FRNET_OUT (h, 1, 1, 0x0000AAAA);
ETM_MACRO_FRNET_OUT (h, 1, 2, 0x00001100);
ETM_MACRO_FRNET_OUT (h, 1, 3, 0x00000011);
```

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 00 00 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | FF FF | *DO setting (for address = 0)* |
| 1 | AA AA | *DO setting (for address = 1)* |
| 2 | 11 00 | *DO setting (for address = 2)* |
| 3 | 00 11 | *DO setting (for address = 3)* |
| | | |

- **Method 2: It can be used for setting FRnet DO values by true values or by variables.**

**The DO is defined as a DWORD because it can be a variable or a true value. When a VARn is used instead of the true value, the MSW will not be zero. When users desire to use a true value to set the DO, please keep the MSW to be zero, and let the LSW part contain the DO value.**

```
ETM_MACRO_FRNET_OUT (h, 1, 0, 0x0000FFFF);
// on module 1, set group number RA=0, output data is 0x0000ffff
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 33 | *Sub_function code* |
| 1 | 00 00 | *Group number (0~7)* |
| 2 | 00 00 | *MSW of DO* |
| 3 | FF FF | *LSW of DO* |
| | | |

```
ETM_MACRO_FRNET_OUT (h, 1, bVAR0, VAR1);
// on module 1, set group number RA= bVAR0, output data is VAR1
```

**Note: The address of bVAR0 is 0x80 ( = 0x80+n); and the start address of VAR1 is 0x012E ( = 300 + 2*n).**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 33 | *Sub_function code* |
| 1 | 00 80 | *bVAR0* |
| 2 | 00 00 | *MSW of VAR1* |
| 3 | 01 2E | *LSW of VAR1* |
| | | |

## ※Δ eRET ETM_MACRO_FRNET_WRITE (HANDLE *h*, BYTE *cardNo*, BYTE *bDoGroup*, DWORD *dwChannel*, DWORD *dwOutput*)

**Description:**

**Write data to the channel of FRnet DO group.**

**Category:**

**MODBUS sub_function; MP, ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *bDoGroup*: | Group number is between 0~7. Eight or above is not defined. |
| *dwChannel*: | The channel of the group is defined 0~15. |
| *dwOutput*: | The written data. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_FRNET_WRITE(h, 1, 1, 1, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 15 | 01 | 10 | 1F 40 | 00 07 | 0E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 35 | *Sub_function code* |
| 1 | 00 00 | *MSW of 0x0001* |
| 2 | 00 01 | *LSW of 0x0001* |
| 3 | 00 00 | *MSW of 0x0001* |
| 4 | 00 01 | *LSW of 0x0001* |
| 5 | 00 00 | *MSW of 0x0001* |
| 6 | 00 01 | *LSW of 0x0001* |

**eRET ETM_FRNET_WRITE_SINGLE_DO (HANDLE *h*, BYTE *cardNo*, BYTE *bGroup*, DWORD *dwChannelNo*, BYTE *bOutput*)**

**Description:**

**Write data to the channel of FRnet DO group.**

**Category:**

**MODBUS table；RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***bGroup*:** | **Group number is between 0~7. Eight or above is not defined.** |
| ***dwChannelNo:*** | **The channel of the group is defined 0~15.** |
| ***bOutput*:** | **The written data.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_FRNET_WRITE_SINGLE_DO (h, 1, *bGroup*, *dwChannelNo*, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Value (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 05 | (bGroup * 16 + dwChannelNo) | 00 00(off); FF 00(on) |

**eRET** ***ETM*_FRNET_WRITE_GROUP_DO (HANDLE *h*, BYTE *cardNo*, BYTE *bGroup*, DWORD *dwdata*)**

**Description:**

**Write data to FRnet DO group.**

**Category:**

**MODBUS table；RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***bGroup*:** | **Group number is between 0~7. Eight or above is not defined.** |
| ***dwdata*:** | **0x00000000 ~ 0x0000ffff, can also use VARn to set.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_FRNET_WRITE_GROUP_DO(h, 1, 0, 0x0000FFFF); // on module 1, set group number RA=0, output data is 0x0000ffff**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 00 00 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 00 00 | *MSW of 0xFFFF* |
| 1 | FF FF | *LSW of 0xFFFF* |

**eRET ETM_FRNET_WRITE_MULTI_GROUP_DO (HANDLE *h*, BYTE *cardNo*, WORD *wStartGroup*, BYTE *bGroupQty*, WORD* *pwOutBuffer*, BYTE *bBufferSize*)**

**Description:**

**Write data to FRnet DO multi-group.**

**Category:**

**MODBUS table；RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***wStartGroup:*** | **The starting group.** |
| ***bGroupQty*:** | **The group quantity.** |
| ***pwOutBuffer*:** | **The pointer points to the memory that store the data of group which you want to write.** |
| ***bBufferSize*:** | **The size of the registers(*bBufferSize**2 >= *bGroupQty)*** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
WORD pwOutBuffer[4];
pwOutBuffer[0]=0xFFFF;
pwOutBuffer[1]=0xAAAA;
pwOutBuffer[2]=0x1100;
pwOutBuffer[3]=0x0011;
ETM_FRNET_WRITE_MULTI_GROUP_DO(h, 1, 0, 4, & pwOutBuffer, 8);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 00 00 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | FF FF | *DO setting (for address = 0)* |
| 1 | AA AA | *DO setting (for address = 1)* |
| 2 | 11 00 | *DO setting (for address = 2)* |
| 3 | 00 11 | *DO setting (for address = 3)* |

## 4.3 FRnet WAIT

※Δ eRET ETM_MACRO_FRNET_WAIT (HANDLE *h*, BYTE *cardNo*, BYTE *bDiGroup*, DWORD *dwChannel*, DWORD *dwDiInput*, DWORD *dwTimeout*)

**Description:**

Wait until FRnet DI turned ON.

**Category:**

MODBUS sub_function; MP, ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *bDiGroup*: | Group number, 8~15 |
| *dwChannel*: | The channel of the group is defined 0~15. |
| *dwDiInput:* | 1:ON 0:OFF |
| *dwTimeout*: | The waiting time(ms) 0: Represent the waiting time is infinite until ON. n: Represent only wait n(ms). |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

ETM_MACRO_FRNET_WAIT(h, 1, 8, 1, 1, 1000);

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 17 | 01 | 10 | 1F 40 | 00 08 | 10 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 36 | *Sub_function code* |
| 1 | 00 08 | *bDiGroup* |
| 2 | 00 00 | *MSW of 0x0001* |
| 3 | 00 01 | *LSW of 0x0001* |
| 4 | 00 00 | *MSW of 0x0001* |
| 5 | 00 01 | *LSW of 0x0001* |
| 6 | 00 00 | *MSW of 0x03E8* |
| 7 | 03 E8 | *LSW of 0x03E8* |

# 5 Auto Homing Search

**i-8094H module provides automatic home search function. After the configuration of proper settigs, the whole process for Homing searching can function automatically. The main steps are as bellows:**

- **Near-home sensor searching (NHOME) under high-speed motion.**
- **Home sensor serachig under low-speed motion.**
- **Servo motor Z-Phase searching under low-speed motion.**
- **Offset movement to the origin of the working area under high-speed motion.**

**A few steps could be skipped by adjusting settings accordingly to meet customer's actual needs. This operation could be performed automatically, therefore economize on CPU resource and reduce programming efforts.**

## 5.1 Set Up Homing Speed

**eRET ETM_SET_HV (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**※ eRET ETM_MACRO_SET_HV (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function sets the homing speed.**

**Category:**

**MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***data*:** | **Homing speed (Vmin~Vmax PPS).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_HV is 0A 3C.**
**The Sub_function code of ETM_MACRO_SET_HV is 0C 3C.**

**MODBUS example:**

**ETM_SET_HV (h, 1, AXIS_X, 500);**
**//set the homing speed of the X axis on module 1 to be 500 PPS.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 3C/0C 3C | *Sub_function code* |
| 1 | 00 01 | *axis* |
| 2 | 00 00 | *MSW of data* |
| 3 | 01 F4 | *LSW of data (500 = 0x1F4)* |
| | | |

## 5.2 Using an Limit Switch as the Home sensor

**eRET ETM_HOME_LIMIT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nType*)**

**※ eRET ETM_MACRO_HOME_LIMIT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nType*)**

**Description:**

**This function sets the Limit Switch to be used as the HOME sensor.**

**Category:**

**MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***nType*:** | **0: Disable the The LIMIT SWITCH function;<br>1: Use the LIMIT SWITCH as the HOME sensor.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_HOME_LIMIT is 0A 3D.**
**The Sub_function code of ETM_MACRO_HOME_LIMIT is 0C 3D.**

**MODBUS example:**

```
ETM_HOME_LIMIT (h, 1, AXIS_X, 0);
// Do not use the Limit Switch as the HOME sensor.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 3D/0C 3D | *Sub_function code* |
| 1 | 00 01 | *axis* |
| 2 | 00 00 | *nType* |
| | | |

## 5.3 Setting the Homing Mode

**eRET ETM_SET_HOME_MODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nStep1*, BYTE *nStep2*, BYTE *nStep3*, BYTE *nStep4* , DWORD *data*)**

**※ eRET ETM_MACRO_SET_HOME_MODE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nStep1*, BYTE *nStep2*, BYTE *nStep3*, BYTE *nStep4* , DWORD *data*)**

**Description:**

**This function sets the homing method and other related parameters.**

**Category:**

**MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***nStep1*:** | **0: Step 1 will not be executed.<br>1: Moves in the positive direction.<br>2: Moves in the negative direction.** |
| ***nStep2*:** | **0: Step 2 will not be executed.<br>1: Moves in a positive direction.<br>2: Moves in a negative direction.** |
| ***nStep3*:** | **0: Step 3 will not be executed.<br>1: Moves in a positive direction.<br>2: Moves in a negative direction.** |
| ***nStep4*:** | **0: Step 4 will not be executed.<br>1: Positive correction.<br>2: Negative correction.** |
| ***data*:** | **Offset value (0 ~ +2,000,000,000)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**The Four Steps Required for Automatic Homing**

| Step | Action | Speed | Sensor |
|---|---|---|---|
| 1 | Searching for the Near Home sensor | V | NHOME (IN0) |
| 2 | Searching for the HOME sensor | HV | HOME (IN1) |
| 3 | Searching for the encoder Z-phasesignal | HV | Z-Phase (IN2) |
| 4 | Moves to the specified position | V | |

**Remark:**

**The Sub_function code of ETM_SET_HOME_MODE is 0A 3E.**
**The Sub_function code of ETM_MACRO_SET_HOME_MODE is 0C 3E.**

**MODBUS example:**

**ETM_SET_HOME_MODE (h, 1, 0x1, 2, 2, 1, 1, 3500);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 17 | 01 | 10 | 1F 40 | 00 08 | 10 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 3E/0C 3E | *Sub_function code* |
| 1 | 00 01 | *axis* |
| 2 | 00 02 | *nStep1* |
| 3 | 00 02 | *nStep2* |
| 4 | 00 01 | *nStep3* |
| 5 | 00 01 | *nStep4* |
| 6 | 00 00 | *MSW of data* |
| 7 | 0D AC | *LSW of data (3500 = 0xDAC)* |
| | | |

## 5.4 Starting the Homing Sequence

eRET ETM_HOME_START (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)

※ eRET ETM_MACRO_HOME_START (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)

**Description:**

This function starts the home search of assigned axes.

**Category:**

MODBUS sub_function; RTC and MP.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_HOME_START is 0A 3F.
The Sub_function code of ETM_MACRO_HOME_START is 0C 3F.

**MODBUS example:**

ETM_HOME_START (h, 1, AXIS_X);
//start the automatic homing sequence for the X axis on module 1.

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 3F/0C 3F | *Sub_function code* |
| 1 | 00 01 | *axis (AXIS_X)* |
| | | |

## 5.5 Wait for the completion of the Homing Sequence

**BYTE i8094H_HOME_WAIT(BYTE *cardNo*, BYTE *axis*)**

**Special Description:**

**This function does not support on the ET-M8194H, related features can use ETM_STOP_WAIT() function in 7.10 to replace.**

**Related example:**

```
//Use the following functions to set the homing mode of the X axis.
ETM_SET_V(h, 1, 0x1, 20000);
ETM_SET_HV(h, 1, 0x1, 500);
ETM_SET_HOME_MODE(h, 1, 0x1, 2, 2, 1, 1, 3500);
ETM_HOME _START(h, 1, 0x1); //start auto-homing.
ETM_STOP_WAIT (h, 1, AXIS_X);
```

| Step | Input Signal | Direction | Speed |
|---|---|---|---|
| 1 | Near HOME (IN0) is active | - | 20000 PPS (V) |
| 2 | HOME (IN1) is active | - | 500 PPS (HV) |
| 3 | Z-phase (IN2) is active | + | 500 PPS (HV) |
| 4 | No sensor is required. Move 3500 pulses along the X axis. | + | 20000 PPS (V) |

# 6 Basic Motion Control

## 6.1 Independent Motion Control for each axis

- **Multiple axes could move at the same time.**
- **The motion of each axis can be started independently.**
- **Each axis is moving independently.**
- **Each axis can receive commandeds to change motion, such as changing the number of pulses or the speed.**
- **Each axis can receive commandeds to stop slowly or suddenly to meet the specific requirements.**
- **Independent axis motion can work with interpolation or synchrous action to perform more complicated and versatile motion.**

### 6.1.1 Setting the Acceleration/Deceleration Mode

**eRET ETM_NORMAL_SPEED (HANDLE *h*, BYTE *cardNo*, BYTE *axis* , BYTE *nMode*)**

**※Δ eRET ETM_MACRO_NORMAL_SPEED (HANDLE *h*, BYTE *cardNo*, BYTE *axis* , BYTE *nMode*)**

**Description:**

**This function sets the speed mode.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *nMode*: | 0 → Symmetric T-curve (Please set SV, V, A, and AO).<br>1 → Symmetric S-curve (Please set SV, V, K, and AO).<br>2 → Asymmetric T-curve (Please set SV, V, A, D, and AO).<br>3 → Asymmetric S-curve (Please set SV, V, K, L, and AO). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**Please refer the configurations of speed-related parameters.**
**The Sub_function code of ETM_NORMAL_SPEED is 0A 46.**
**The Sub_function code of ETM_MACRO_NORMAL_SPEED is 0C 46.**

**MODBUS example:**

```
ETM_NORMAL_SPEED (h, 1, AXIS_XYZU, 0);
//use a symmetric T-curve for all axes on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 46/0C 46 | *Sub_function code* |
| 1 | 00 0F | *axis (AXIS_XYZU)* |
| 2 | 00 00 | *nMode* |

**Related example:**

```
BYTE cardNo=1; //select module 1.
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 20000);
//set the max speed of XYZU axes to be 20K PPS.

//=========================================================
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU, 0);
//use a symmetric T- curve  for all axes on module 1.
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
ETM_SET_A(h, cardNo, AXIS_XYZU, 1000);
//set the acceleration of all axes on module 1 to be 1000 PPS/Sec.
ETM_SET_SV(h, cardNo, AXIS_XYZU, 2000);
//set the start speed of all axes on module 1 to be 2000 PPS.
ETM_SET_AO(h, cardNo, AXIS_XYZU, 9);
//set the number of remaining offset pulses for all axes to be 9 pulses.
ETM_FIXED_MOVE(h, cardNo, AXIS_XYZU, 10000);
//move all axes on module 1 to be 10000 pulses.

//=========================================================
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU,1);
//use a symmetric S-curve for all axes on module 1.
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
ETM_SET_K(h, cardNo, AXIS_XYZU, 500);
//set the acceleration rate of all axes on module 1 to be 500 PPS/sec^2.
ETM_SET_SV(h, cardNo, AXIS_XYZU, 200);
//set the start speed of all axes on module 1 to be 200 PPS.
ETM_SET_AO(h, cardNo, AXIS_XYZU, 9);
//set the number of remaining offset pulses to be 9 pulses for all axes.
ETM_FIXED_MOVE(h, cardNo, AXIS_XYZU, -10000);
//move all axes on module 1 to be 10000 pulses in reverse direction.

//=========================================================
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU,2);
```

```
//use an asymmetric T-curve for all axes on module 1.
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
ETM_SET_A(h, cardNo, AXIS_XYZU, 1000 );
//set the acceleration of all axes on module 1 to be 1000 PPS/sec.
ETM_SET_D(h, cardNo, AXIS_XYZU, 500);
//set the deceleration of all axes on module 1 to be 500 PPS.
ETM_SET_SV(h, cardNo, AXIS_XYZU, 200);
//set the start speed of all axes on module 1 to 200 PPS.
ETM_SET_AO(h, cardNo, AXIS_XYZU, 9);
//set the number of remaining offset pulses to be 9 pulses for all axes.
ETM_FIXED_MOVE(h, cardNo, axis, 10000);
//move all axes on module 1 to be 10000 pulses.

//=========================================================
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU,3);
//use an asymmetric S-curve for all axes on module 1.
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
ETM_SET_K(h, cardNo, AXIS_XYZU, 500);
//set the acceleration rate of all axes on module 1 to be 500 PPS/sec^2.
ETM_SET_L(h, cardNo, AXIS_XYZU, 300);
//set the deceleration rate of all axes on module 1 to be 300 PPS/sec^2.
ETM_SET_SV(h, cardNo, AXIS_XYZU, 200);
//set the start speed of all axes on module 1 to be 200 PPS.
ETM_SET_AO(h, cardNo, AXIS_XYZU, 9);
//set the number of remaining offset pulses to be 9 pulses for all axes.
ETM_FIXED_MOVE(h, cardNo, AXIS_XYZU, 10000);
//move all axes on module 1 to be 10000 pulses.
```

### 6.1.2 Setting the Starting Speed

**eRET ETM_SET_SV (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_SV (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function sets the initial speed for the assigned axes.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | Set up speed(Please refer to 2.5 for maximm speed)PPS. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_SV is 0A 47.**
**The Sub_function code of ETM_MACRO_SET_SV is 0C 47.**

**MODBUS example:**

```
ETM_SET_SV (h, 1, AXIS_X, 1000);
//set the starting speed for the X axis on module 1 to 1000 PPS.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 47/0C 47 | *Sub_function code* |
| 1 | 00 01 | *axis (1 = AXIS_X)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 03 E8 | *LSW of data (1000 = 0x3E8)* |

### 6.1.3 Setting the Desirted Speed

**eRET ETM_SET_V (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD data)**

**※Δ eRET ETM_MACRO_SET_V (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function sets the desired speed for the assigned axes.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | Set up speed(Please refer to 2.5 for maximm speed)PPS. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_V is 0A 48.**
**The Sub_function code of ETM_MACRO_SET_V is 0C 48.**

**MODBUS example:**

```
ETM_SET_V (h, 1, AXIS_X, 120000L);
//set the speed for the X axis on module 1 to 120000 PPS.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 48/0C 48 | *Sub_function code* |
| 1 | 00 01 | *axis (1 = AXIS_X)* |
| 2 | 00 01 | *MSW of data* |
| 3 | D4 C0 | *LSW of data (120000 = 0x1D4C0)* |

### 6.1.4 Setting the Acceleration

**eRET ETM_SET_A (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD data)**

**※Δ eRET ETM_MACRO_SET_A (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function sets the acceleration value for the assigned axes.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | The acceleration value (PPS/sec).<br>This value is related to the maximum speed value defined by ETM_SET_MAX_V() function. The maximum available acceleration value is MAX_V * 125. The minimum acceleration value is MAX_V ÷ 64, and all other acceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_A is 0A 49.**
**The Sub_function code of ETM_MACRO_SET_A is 0C 49.**

**MODBUS example:**

```
ETM_SET_A (h, 1, AXIS_X, 100000L);
//set the acceleration value of the X axis on module 1 to 100K PPS/Sec.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 49/0C 49 | *Sub_function code* |
| 1 | 00 01 | *axis (1 = AXIS_X)* |
| 2 | 00 01 | *MSW of data* |
| 3 | 86 A0 | *LSW of data (100000 = 0x186A0)* |

### 6.1.5 Setting the Deceleration

**eRET ETM_SET_D (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_D (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**This function sets the deceleration value for the assigned axes.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***data*:** | **The deceleration value (PPS/sec).<br>This value is related to the maximum speed value defined by ETM_SET_MAX_V () function. The maximum available deceleration value is MAX_V * 125. The minimum deceleration value is MAX_V ÷ 64, and all other deceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_D is 0A 4A.**
**The Sub_function code of ETM_MACRO_SET_D is 0C 4A.**

**MODBUS example:**

```
ETM_SET_D (h, 1, AXIS_X, 100000L);
//set the deceleration value of the X axis on module 1 to 100K PPS/sec.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 4A/0C 4A | *Sub_function code* |
| 1 | 00 01 | *axis (1 = AXIS_X)* |
| 2 | 00 01 | *MSW of data* |
| 3 | 86 A0 | *LSW of data (100000 = 0x186A0)* |

### 6.1.6 Setting the Acceleration Rate

**eRET ETM_SET_K (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_K (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**The function sets the acceleration rate (i.e., Jerk) value for the assigned axes.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis*:** | **Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).** |
| ***data*:** | **The acceleration rate (jerk) value(PPS/ $sec^2$).<br>This value is related to the maximum speed value defined by ETM_SET_MAX_V () function.<br>The minimum acceleration rate is MAX_V * 0.0119211;<br>The maximum acceleration rate value is 2,000,000,000.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_K is 0A 4B.**
**The Sub_function code of ETM_MACRO_SET_K is 0C 4B.**

**MODBUS example:**

```
ETM_SET_K (h, 1, AXIS_X, 10000);
//set the acceleration rate value of the X axis on module 1 to
//1,000*10 (= 10,000) PPS/Sec^2.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 4B/0C 4B | *Sub_function code* |
| 1 | 00 01 | *axis (1 = AXIS_X)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 27 10 | *LSW of data (10000 = 0x2710)* |

### 6.1.7 Setting the Deceleration Rate

**eRET ETM_SET_L (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_L (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

**The function sets the deceleration rate (i.e., Jerk) value for the assigned axes.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | The deceleration rate (jerk) value(PPS/ $Sec^2$).<br>This value is related to the maximum speed value defined by ETM_SET_MAX_V () function.<br>The minimum deceleration rate is MAX_V * 0.0119211;<br>The maximum deceleration rate value is 2,000,000,000 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_L is 0A 4C.**
**The Sub_function code of ETM_MACRO_SET_L is 0C 4C.**

**MODBUS example:**

```
ETM_SET_L (h, 1, AXIS_X, 10000);
//set the deceleration rate value of the X axis on module 1 to
//1,000*10 (= 10,000) PPS/Sec^2.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 4C/0C 4C | *Sub_function code* |
| 1 | 00 01 | *axis (1 = AXIS_X)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 27 10 | *LSW of data (10000 = 0x2710)* |

## 6.1.8 Setting the Value of the Remaining Offset of Pulses

**eRET ETM_SET_AO (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *data*)**

**※Δ eRET ETM_MACRO_SET_AO (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *data*)**

**Description:**

**This function sets the number of remaining offset pulses for the assigned axes. Please refer to the figure below for a definition of the remaining offset pulse value.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | The number of remaining offset pulses. (-32,768 ~ +32,767). |

**Return:**

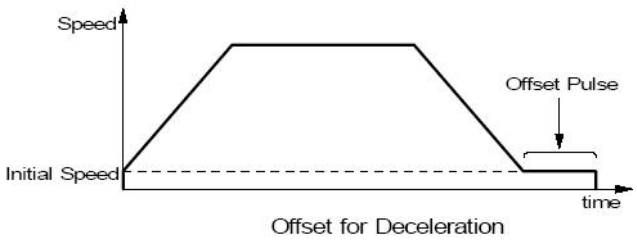
**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_AO is 0A 4D.**
**The Sub_function code of ETM_MACRO_SET_AO is 0C 4D.**

**MODBUS example:**

```
ETM_SET_AO (h, 1, AXIS_X, 200);
//set the number of remaining offset pulses for the X axis on
//module 1 to 200 pulses.
```



Offset for Deceleration

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 4D/0C 4D | *Sub_function code* |
| 1 | 00 01 | *axis (1 = AXIS_X)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 00 C8 | *LSW of data (200 = 0xC8)* |

### 6.1.9 Fixed Pulse Output

**eRET ETM_FIXED_MOVE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *data*)**

**※Δ eRET ETM_MACRO_FIXED_MOVE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *data*)**

**Description:**

Command a point-to-point motion for Fixed Position movement.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | Pulses (-2,000,000,000 ~ +2,000,000,000). |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_FIXED_MOVE is 0A 4E.
The Sub_function code of ETM_MACRO_FIXED_MOVE is 0C 4E.

**MODBUS example:**

```
ETM_FIXED_MOVE (h, cardNo, AXIS_XYZU, 10000);
// AXIS_XYZU move 10000 Pulses
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 4E/0C 4E | *Sub_function code* |
| 1 | 00 0F | *axis (0xF = AXIS_XYZU)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 27 10 | *LSW of data (10000 = 0x2710)* |

**Related example:**

```
BYTE cardNo=1; //select module 1
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 20000);
//set the max. velocity of all axes on module 1 to be 20K PPS
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU, 0);
//set the speed profile of all axes on module 1 to be symmetric T-curve
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS
ETM_SET_A(h, cardNo, AXIS_XYZU,1000);
//set the acceleration value of all axes on module 1 to be 1000 PPS/S
ETM_SET_SV(h, cardNo, AXIS_XYZU, 2000);
//set the start velocity of all axes on module 1 to be 2000 PPS
ETM_SET_AO(h, cardNo, AXIS_XYZU, 9);
//set the remaining offset pulses to be 9 PPS
ETM_FIXED_MOVE(h, cardNo, AXIS_XYZU, 10000);
// move 10000 Pulses for each axis on module 1
```

**eRET ETM_SET_PULSE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_PULSE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, DWORD *data*)**

**Description:**

This command enables to make a change of pulse during outputing fixed pulses on each axis (but not for change of the direction).

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | Pulses (0 ~ +2,000,000,000). |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_SET_PULSE is 0A 4F.
The Sub_function code of ETM_MACRO_SET_PULSE is 0C 4F.

**MODBUS example:**

```
ETM_SET_PULSE (h, cardNo, AXIS_XYZU, 9000);
//Change the total pulses to 9000 pulses during fixed pulse moving.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 4F/0C 4F | *Sub_function code* |
| 1 | 00 0F | *axis (0xF = AXIS_XYZU)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 23 28 | *LSW of data (9000 = 0x2328)* |

**Related example:**

```
BYTE cardNo=1; //select module 1
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 20000);
//set the max velocity of all axes on module 1 to be 20K PPS
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU, 0);
//set the speed profile of all axes on module 1 to be symmetric T-curve
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS
ETM_SET_A(h, cardNo, AXIS_XYZU,1000);
//set the acceleration value of all axes on module 1 to be 1000 PPS/S
ETM_SET_SV(h, cardNo, AXIS_XYZU, 2000);
//set the start velocity of all axes on module 1 to be 2000 PPS
ETM_SET_AO(h, cardNo, AXIS_XYZU, 9);
//set the remaining offset pulses to be 9 PPS
ETM_FIXED_MOVE(h, cardNo, AXIS_XYZU, 10000);
// move 10000 Pulses for each axis on module 1
ETM_SET_PULSE(h, cardNo, AXIS_XYZU, 9000);
//Set pulse as 9000 Pulse.
```

### 6.1.10 Continue Pulse Output

**eRET ETM_CONTINUE_MOVE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long data)**

**※Δ eRET ETM_MACRO_CONTINUE_MOVE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *data*)**

**Description:**

**This function is continuous motion command for several independent axes.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *data*: | The specified speed (positive value for CW motion; negative value for CCW motion) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the errors.**

**Remark:**

**The Sub_function code of ETM_CONTINUE_MOVE is 0A 50.**
**The Sub_function code of ETM_MACRO_CONTINUE_MOVE is 0C 50.**

**MODBUS example:**

```
ETM_CONTINUE_MOVE (h, cardNo, AXIS_XYZU, 1000);
//continue to move at the speed of 1K PPS
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 50/0C 50 | *Sub_function code* |
| 1 | 00 0F | *axis (0xF = AXIS_XYZU)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 03 E8 | *LSW of data (1000 = 0x3E8)* |

**Related example:**

```
BYTE cardNo=1; //select module 1
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 20000);
//set the maximum speed of all axes on module 1 to 20K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU, 0);
//set the speed profile for all axes as a symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to 2000 PPS.
ETM_SET_A(h, cardNo, AXIS_XYZU,1000);
//set the acceleration value of all axes to 1000 PPS/S.
ETM_SET_SV(h, cardNo, AXIS_XYZU, 2000);
//set the start velocity of all axes to 2000 PPS
ETM_CONTINUE_MOVE(h, cardNo, AXIS_XYZU, 1000);
//move all axes on module 1 at a speed of 1000 PPS.
```

## 6.2 Interpolation Commands

### 6.2.1 Assigning the Axes for Interpolation

**eRET ETM_AXIS_ASSIGN (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*)**

**※Δ eRET ETM_MACRO_AXIS_ASSIGN (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*)**

**Description:**

This function assigns the axes to be used for interpolation. Either two or three axes can be assigned by using this function. Interpolation commands will refer to the assigned axes to construct a working coordinate system. The X axis does not necessarily have to be the first axis. However, it is easier to use the X axis as the first axis, the Y axis as the second axis, and the Z axis as the third axis.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (Please refer to Table 2)<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis; can be either X, Y, Z, or U (1 or 2 or 4 or 8). |
| *axis3*: | The third axis; can be either X, Y, Z, or U (1 or 2 or 4 or 8). Without 3rd axis the value is 0. |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_AXIS_ASSIGN is 0A 5A.
The Sub_function code of ETM_MACRO_AXIS_ASSIGN is 0C 5A.

**MODBUS example:**

```
ETM_AXIS_ASSIGN (h, 1, AXIS_X, AXIS_Y, 0);
//set the X axis of module 1 as the first axis and the Y axis as the second
//axis.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 5A/0C 5A | *Sub_function code* |
| 1 | 00 01 | *axis1 (axis1 = AXIS_X)* |
| 2 | 00 02 | *axis2 (axis1 = AXIS_Y)* |
| 3 | 00 00 | *Axis3 (not defined)* |

## 6.2.2 Setting the Speed and Acc/Dec Mode

**eRET ETM_VECTOR_SPEED (HANDLE *h*, BYTE *cardNo*, BYTE *nMode*)**

**※Δ eRET ETM_MACRO_VECTOR_SPEED (HANDLE *h*, BYTE *cardNo*, BYTE *nMode*)**

**Description:**

**This function sets vector acceleration or deceleration mode.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***nMode*:** | **0 → 2-axis linear or circular motion at a constant vector speed (Set VV and VSV; and VV=VSV).**<br>**1 → 2-axis linear motion using a symmetric T-curve velocity profile (set VSV, VV, VA, and VAO)**<br>**2 → 2-axis linear motion using a symmetric S-curve velocity profile(set VSV, VV, VK, and VAO)**<br>**3 → 2-axis linear motion using an asymmetric T-curve velocity profile (set VSV, VV, VA, VD, and VAO)**<br>**4 → 2-axis linear motion using an asymmetric S-curve velocity profile (set VSV, VV, VK, VL, and VAO)**<br>**5 → 2-axis circular motion using a symmetric T-curve velocity profile (set VSV, VV, VA, and VAO)**<br>**6 → 2-axis circular motion using an asymmetric T-curve velocity rofile (set VSV, VV, VA, VD, and VAO)**<br>**7 → 3-axis linear motion at a constant vector speed set VV and VSV; and VV=VSV)**<br>**8 → 3-axis linear motion at using a symmetric T-curve velocity rofile (set VSV, VV, VA, and VAO)**<br>**9 → 3-axis linear motion using a symmetric S-curve velocity profile (set VSV, VV, VK, and VAO)**<br>**10 → 3-axis linear motion using an asymmetric T-curve velocity profile (set VSV, VV, VA, VD, and VAO)**<br>**11 → 3-axis linear motion using an asymmetric S-curve velocity profile (set VSV, VV, VK, VL, and VAO)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**Please refer the configurations of speed-related parameters.**
**The Sub_function code of ETM_VECTOR_SPEED is 0A 5B.**
**The Sub_function code of ETM_MACRO_VECTOR_SPEED is 0C 5B.**

**MODBUS example:**

```
ETM_VECTOR_SPEED (h, cardNo, 0);
//set the module to perform 2-axis linear or circular motion
//at a constant vector speed.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 5B/0C 5B | *Sub_function code* |
| 1 | 00 00 | *nMode* |
| | | |

**Related example:**

```
BYTE cardNo=1; //select module 1.
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 20000);
//set the maximum speed of all axes to 20K PPS.

//================================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
ETM_VECTOR_SPEED(h, cardNo, 0);
//set module 1 to perform 2-axis linear or circular motion
ETM_SET_VV(h, cardNo, 1000);
//set the vector speed to 1000 PPS.
ETM_LINE_2D(h, cardNo, 12000, 10000);
//execute the 2-axis linear interpolation motion.

//================================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
ETM_VECTOR_SPEED(h, cardNo, 1);
//set module 1 to perform 2-axis linear motion using a symmetric
//T-curve velocity profile (VSV、VV、VA、VAO).
ETM_SET_VSV(h, cardNo, 500); //set the starting vector speed to 500 PPS.
```

```
ETM_SET_VV(h, cardNo, 2000); //set the vector speed to 2000 PPS.
ETM_SET_VA(h, cardNo, 1000);
//set the vector acceleration to 1000 PPS/sec.
ETM_LINE_2D(h, cardNo, 20000, 10000);
//execute the 2-axis linear interpolation motion.

//=============================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
ETM_VECTOR_SPEED(h, cardNo, 2);
//2-axis linear motion using a symmetric S-curve velocity profile(VSV、VV、
//VK、AO).
ETM_SET_VSV(h, cardNo, 200); //set the starting vector speed to 200 PPS.
ETM_SET_VV(h, cardNo, 2000); //set the vector speed to 2000 PPS.
ETM_SET_VK(h, cardNo, 500); //set the acceleration rate to 500 PPS/Sec.
ETM_SET_VAO(h, cardNo, 20);
//set the value of remaining offset pulses to 20.
ETM_LINE_2D(h, cardNo, 10000, 10000);
//execute the 2-axis linear interpolation motion.

//=============================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
ETM_VECTOR_SPEED(h, cardNo, 3);
//2-axis linear motion using an asymmetric T-curve velocity profile(VSV、
//VV、VA、VD、VAO).
ETM_SET_VSV(h, cardNo, 100); //set the start vector speed to 100 PPS.
ETM_SET_VV(h, cardNo, 2000); //set the vector speed to 2000 PPS.
ETM_SET_VA(h, cardNo, 1000);
//set the vector acceleration to 1000 PPS/sec.
ETM_SET_VD(h, cardNo, 500); //set the vector deceleration to 500 PPS/sec.
ETM_SET_VAO(h, cardNo, 20);
//set the value of remaining offset pulses to 20.
ETM_LINE_2D(h, cardNo, 10000, 5000);
//execute the 2-axis linear interpolation motion.

//=============================================
long fp1=4000;
long fp2=10000;
unsigned short  sv=200;
unsigned short  v=2000;
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 8000);
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
ETM_VECTOR_SPEED(h, cardNo, 4);
//2-axis linear motion using an asymmetric S-curve velocity profile(VSV、
```

```
//VV、VK、VL、VAO).
ETM_SET_VSV(h, cardNo, sv); //set the starting velocity to sv PPS.
ETM_SET_VV(h, cardNo, v); //set the vector speed to v PPS.
ETM_SET_VK(h, cardNo, 500); //set the acceleration rate to 500 PPS/Sec^2.
ETM_SET_VL(h, cardNo, 300); //set the deceleration rate to 300 PPS/Sec^2.
ETM_SET_VAO(h, cardNo, 20);
//set the value of remaining offset pulses to 20.
ETM_LINE_2D(h, cardNo, fp1, fp2); //execute the 2-axis linear motion.

//=============================================
long fp1=11000;
long fp2=9000;
long c1=10000;
long c2=0;
unsigned short  sv=100;
unsigned short  v=3000;
unsigned long  a=5000;
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 8000);
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
ETM_VECTOR_SPEED(h, cardNo, 5);
//2-axis circular motion using a symmetric T-curve velocity profile(VSV、
// VV、VA、VAO).
ETM_SET_VSV(h, cardNo, sv); //set the starting vector speed to sv PPS.
ETM_SET_VV(h, cardNo, v); //set vector speed to v PPS.
ETM_SET_VA(h, cardNo, a); //set the vector acceleration to a PPS/sec.
ETM_SET_VAO(h, cardNo, 0);
//set the value of remaining offset pulses to 0 Pulse.
ETM_ARC_CW(h, cardNo, c1,c2, fp1, fp2);
//execute the 2-axis CW circular motion.

//=============================================
long c1=300;
long c2=0;
unsigned short sv=100;
unsigned short v=3000;
unsigned long a=125;
unsigned long d=12;
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 8000);
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
ETM_VECTOR_SPEED(h, cardNo, 6);
//2-axis circular motion using an asymmetric T-curve velocity
//profile(VSV、VV、VA、VD、VAO).
ETM_SET_VSV(h, cardNo, sv); //set the starting vector speed to sv PPS.
ETM_SET_VV(h, cardNo, v); //set vector speed to v PPS.
```

```
ETM_SET_VA(h, cardNo, a); //set acceleration to a PPS/Sec.
ETM_SET_VD(h, cardNo, d); //set the deceleration to d PPS/Sec.
ETM_SET_VAO(h, cardNo, 0); //set the value of remaining offset pulses to 0.
ETM_CIRCLE_CW(h, cardNo, c1, c2);
//execute the 2-axis CW circular motion.

//=============================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, AXIS_Z);
// set axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
ETM_VECTOR_SPEED(h, cardNo, 7);
//3-axis linear motion at a constant vector speed (VSV=VV).
ETM_SET_VSV(h, cardNo, 1000); //set the start speed to 1000 PPS.
ETM_SET_VV(h, cardNo, 1000); //set the constant speed to 1000 PPS.
ETM_LINE_3D(h, cardNo, 10000, 10000,10000);
//execute the 3-axis linear motion.

//=============================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, AXIS_Z);
// set axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z-axis.
ETM_VECTOR_SPEED(h, cardNo, 8);
//3-axis linear motion using a symmetric T-curve velocity profile(VSV、VV、
//VA、VAO).
ETM_SET_VSV(h, cardNo, 100); //set the starting speed to 100 PPS.
ETM_SET_VV(h, cardNo, 3000); //set the vector speed to 3000 PPS.
ETM_SET_VA(h, cardNo, 500); //set the vector acceleration to 500 PPS/Sec.
ETM_SET_VAO(h, cardNo, 20);
//set the value of remaining offset pulses to 20.
ETM_LINE_3D(h, cardNo, 10000, 1000,20000);
//execute the 3-axis linear motion

//=============================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, AXIS_Z);
// set the axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
ETM_VECTOR_SPEED(h, cardNo, 9);
//3-axis linear motion using a symmetric S-curve velocity profile(VSV、VV、
//VK、VAO).
ETM_SET_VSV(h, cardNo, 100); //set the starting speed to 100 PPS.
ETM_SET_VV(h, cardNo, 3000); //set the vector speed to 3000 PPS.
ETM_SET_VK(h, cardNo, 500);
//set the vector acceleration rate to 500 PPS/sec^2.
ETM_SET_VAO(h, cardNo, 20);
//set the value of remaining offset pulses to 20.
ETM_LINE_3D(h, cardNo, 10000, 1000,1000);
//execute the 3-axis linear motion.
```

```
//=============================================
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, AXIS_Z);
// set the axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
ETM_VECTOR_SPEED(h, cardNo, 10);
//set the module 1 to perform 3-axis linear motion
//using an asymmetric T-curve speed profile(VSV、VV、VA、VD、VAO).
ETM_SET_VSV(h ,cardNo, 100); //set the starting speed to 100 PPS.
ETM_SET_VV(h ,cardNo, 2000); //set the vector speed as 2000 PPS.
ETM_SET_VA(h, cardNo, 1000);
//set the vector acceleration to 1000 PPS/sec.
ETM_SET_VD(h, cardNo, 500); //set the vector deceleration to 500 PPS/sec.
ETM_SET_VAO(h, cardNo, 20);
//set the value of remaining offset pulses to 20.
ETM_LINE_3D(h, cardNo, 10000, 1000,1000);
//execute the 3-axis linear motion.

//=============================================
long fp1=4000;
long fp2=10000;
long fp3=20000;
unsigned short sv=200;
unsigned short v=2000;
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 8000);
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, AXIS_Z);
// set axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
ETM_VECTOR_SPEED(h, cardNo, 11);
//3-axis linear motion using an asymmetric S-curve velocity profile(VSV、
//VV、VK、VL、VAO).
ETM_SET_VSV(h, cardNo, sv); //set the starting speed to sv PPS.
ETM_SET_VV(h, cardNo, v); //set the vector speed to v PPS.
ETM_SET_VK(h, cardNo, 500);
//set the vector acceleration rate to 500 PPS/sec^2.
ETM_SET_VL(h, cardNo, 300);
//set the vector deceleration rate to 300 PPS/sec^2.
ETM_SET_VAO(h, cardNo, 20);
//set the value of remaining offset pulses to 20.
ETM_LINE_3D(h, cardNo, fp1, fp2, fp3);
//execute the 3-axis linear motion.
```

### 6.2.3 Setting the Vector Starting Speed

**eRET ETM_SET_VSV (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_VSV (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**Description:**

**This function sets the starting speed of the principle axis (axis 1) for the interpolation motion.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***data*:** | **The vector starting speed value (in PPS) (For maximum value please refer to section 2.5).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_VSV is 0A 5C.**
**The Sub_function code of ETM_MACRO_SET_VSV is 0C 5C.**

**MODBUS example:**

**ETM_SET_VSV (h, 1, 1000);**
**//set the starting speed of the axis 1 for the interpolation motion**
**//on module 1 to 1000 PPS.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 0A 5C/0C 5C | | *Sub_function code* | | | |
| 1 | | 00 00 | | *MSW of data* | | | |
| 2 | | 03 E8 | | *LSW of data (1000 = 0x3E8)* | | | |

### 6.2.4 Setting the Vector Speed

**eRET ETM_SET_VV (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_VV (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**Description:**

**This function sets the vector speed of the interpolation motion. Users do not need to assign any axes on this function. The speed setting will take effect on the current working coordinate system which is defined by the ETM_AXIS_ASSIGN() function.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *data*: | The vector speed value (in PPS) (For maximum value please refer to section 2.5). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_VV is 0A 5D.**
**The Sub_function code of ETM_MACRO_SET_VV is 0C 5D.**

**MODBUS example:**

**ETM_SET_VV (h, 1, 120000L);**
**//set the vector speed of the interpolation on module 1 to 120000 PPS.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 5D/0C 5D | *Sub_function code* |
| 1 | 00 01 | *MSW of data* |
| 2 | D4 C0 | *LSW of data (120000 = 0x1D4C0)* |

### 6.2.5 Setting the Vector Acceleration

**eRET ETM_SET_VA (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_VA (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**Description:**

**This function sets the vector acceleration of the interpolation motion. Users do not need to assign any axes on this function. The speed setting will take effect on the current working coordinate system which is defined by the ETM_AXIS_ASSIGN() function.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***data*:** | **The vector acceleration value (in PPS/sec). This value is related to the maximum speed value defined by ETM_SET_MAX_V () function. The maximum available acceleration value is MAX_V * 125. The minimum acceleration value is MAX_V ÷ 64, and all other acceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_VA is 0A 5E.**
**The Sub_function code of ETM_MACRO_SET_VA is 0C 5E.**

**MODBUS example:**

```
ETM_SET_VA (h, 1, 100000L);
//set the vector acceleration of the interpolation motion
//on module 1 to 100K PPS/sec.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 5E/0C 5E | *Sub_function code* |
| 1 | 00 01 | *MSW of data* |
| 2 | 86 A0 | *LSW of data (100000 = 0x186A0)* |

### 6.2.6 Setting the Vector Deceleration Value

**eRET ETM_SET_VD (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_VD (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**Description:**

**This function sets the vector deceleration of the interpolation motion.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***data*:** | **The vector deceleration value (in PPS/sec). This value is related to the maximum speed value defined by ETM_SET_MAX_V () function. The maximum available acceleration value is MAX_V * 125. The minimum acceleration value is MAX_V ÷ 64, and all other acceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_VD is 0A 5F.**
**The Sub_function code of ETM_MACRO_SET_VD is 0C 5F.**

**MODBUS example:**

```
ETM_SET_VD (h, 1, 100000L);
//set the vector deceleration value of interpolation motion
//on module 1 to 100K PPS/Sec.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 5F/0C 5F | *Sub_function code* |
| 1 | 00 01 | *MSW of data* |
| 2 | 86 A0 | *LSW of data (100000 = 0x186A0)* |

### 6.2.7 Setting the Vector Acceleration Rate

**eRET ETM_SET_VK (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_VK (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**Description:**

Set the acceleration rate (jerk) value for interpolation motion.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *data*: | The acceleration rate (jerk) value(PPS/sec$^2$).<br>This value is related to the maximum speed value defined by ETM_SET_MAX_V () function.<br>The minimum acceleration rate is MAX_V * 0.0119211;<br>The maximum acceleration rate value is 2,000,000,000. |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_SET_VK is 0A 60.
The Sub_function code of ETM_MACRO_SET_VK is 0C 60.

**MODBUS example:**

```
ETM_SET_VK (h, 1, 10000);
//set the acceleration rate of the interpolation motion on module 1
// to 10,000 PPS/sec^2.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 60/0C 60 | *Sub_function code* |
| 1 | 00 00 | *MSW of data* |
| 2 | 27 10 | *LSW of data (10000 = 0x2710)* |

### 6.2.8 Setting the Vector Deceleration Rate

**eRET ETM_SET_VL (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**※Δ eRET ETM_MACRO_SET_VL (HANDLE *h*, BYTE *cardNo*, DWORD *data*)**

**Description:**

Set the deceleration rate of the interpolation motion.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *data*: | The deceleration rate (jerk) value(PPS/sec$^2$).<br>This value is related to the maximum speed value defined by ETM_SET_MAX_V () function.<br>The minimum acceleration rate is MAX_V * 0.0119211;<br>The maximum acceleration rate value is 2,000,000,000. |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_SET_VL is 0A 61.
The Sub_function code of ETM_MACRO_SET_VL is 0C 61.

**MODBUS example:**

```
ETM_SET_VL (h, 1, 10000);
//set the deceleration rate of the interpolation on module 1 to 10,000
//PPS/sec^2.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 61/0C 61 | *Sub_function code* |
| 1 | 00 00 | *MSW of data* |
| 2 | 27 10 | *LSW of data (10000 = 0x2710)* |

## 6.2.9 Setting the Number of Remaining Offset Pulses

**eRET ETM_SET_VAO (HANDLE *h*, BYTE *cardNo*, long *data*)**

**※Δ eRET ETM_MACRO_SET_VAO (HANDLE *h*, BYTE *cardNo*, long *data*)**

**Description:**

**Setting this value will cause the motion control chip to start deceleration earlier. The remaining offset pulses will be completed at low speed to allow the controller to stop immediately when it reaches the offset pulse value. Please refer to the figure below for more information.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***data*:** | **The number of remaining offset pulses. (-32,768 ~ +32,767).** |

**Return:**

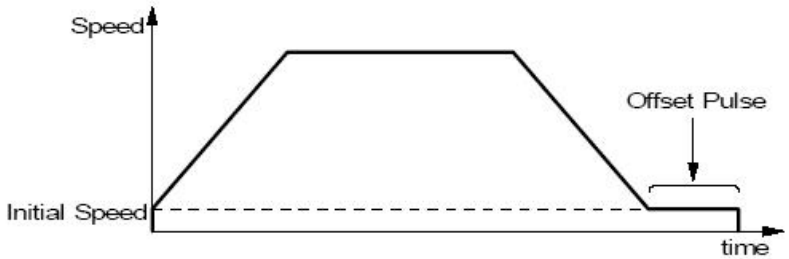
**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_VAO is 0A 62.**
**The Sub_function code of ETM_MACRO_SET_VAO is 0C 62.**

**MODBUS example:**

```
ETM_SET_VAO (h, 1, 200);
//set the number of remaining offset pulse value on module 1 to 200.
```



Offset for Deceleration

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 62/0C 62 | *Sub_function code* |
| 1 | 00 00 | *MSW of data* |
| 2 | 00 C8 | *LSW of data (200 = 0xC8)* |

### 6.2.10 2-Axis Interpolation Motion

**eRET ETM_LINE_2D (HANDLE *h*, BYTE *cardNo*, long *fp1*, long *fp2*)**

**※Δ eRET ETM_MACRO_LINE_2D (HANDLE *h*, BYTE *cardNo*, long *fp1*, long *fp2*)**

**Description:**

**This function executes a 2-axis linear interpolation motion.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***fp1*:** | **The displacement of the axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)** |
| ***fp2*:** | **The displacement of the axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)** |

**Return:**

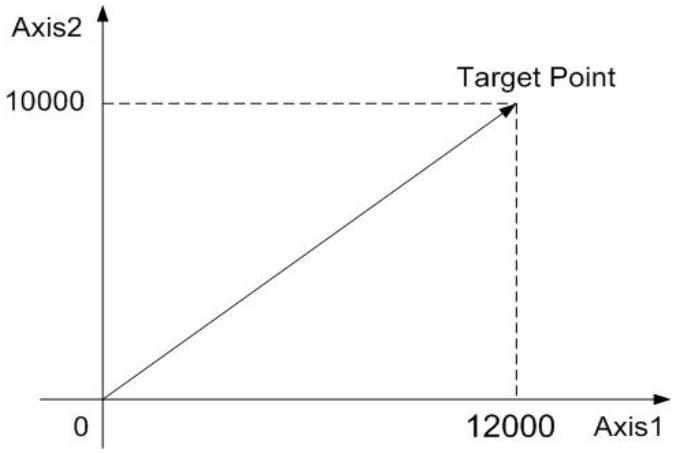
**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_LINE_2D is 0A 63.**
**The Sub_function code of ETM_MACRO_LINE_2D is 0C 63.**

**MODBUS example:**

```
ETM_LINE_2D (h, 1, 12000, 10000);
//execute the 2-axis linear interpolation motion on module 1.
```




**2-axis linear interpolation motion**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 63/0C 63 | *Sub_function code* |
| 1 | 00 00 | *MSW of fp1* |
| 2 | 2E E0 | *LSW of fp1 (12000 = 0x2EE0)* |
| 3 | 00 00 | *MSW of fp2* |
| 4 | 27 10 | *LSW of fp2 (10000 = 0x2710)* |

### 6.2.11 3-Axis Interpolation Motion

**eRET ETM_LINE_3D (HANDLE *h*, BYTE *cardNo*, long *fp1*, long *fp2*, long *fp3*)**

**※Δ eRET *ETM*_MACRO_LINE_3D (HANDLE *h*, BYTE *cardNo*, long *fp1*, long *fp2*, long *fp3*)**

**Description:**

**This function executes a 3-axis linear interpolation motion.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***fp1*:** | **The displacement of the first axis (axis 1) in Pulses(-2,000,000,000 ~ +2,000,000,000)** |
| ***fp2*:** | **The displacement of the second axis (axis 2) in Pulses(-2,000,000,000 ~ +2,000,000,000)** |
| ***fp3*:** | **The displacement of the third axis (axis 3) in Pulses(-2,000,000,000 ~ +2,000,000,000)** |

**Return:**

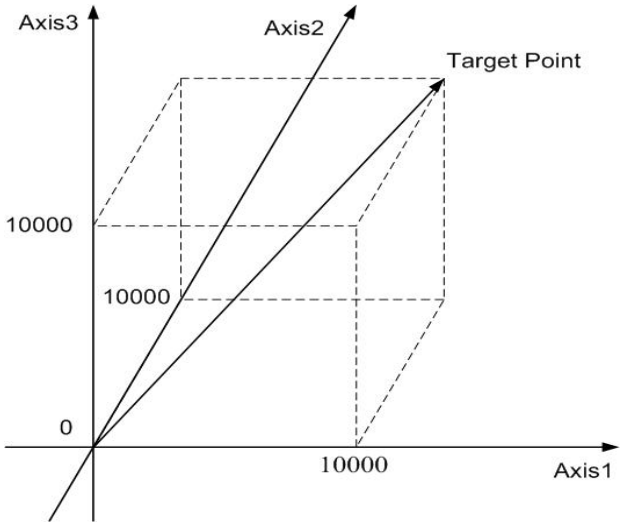
**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the errors.**

**Remark:**

**The Sub_function code of ETM_LINE_3D is 0A 64.**
**The Sub_function code of ETM_MACRO_LINE_3D is 0C 64.**

**MODBUS example:**

```
ETM_LINE_3D (h, 1, 10000, 10000, 10000);
//execute the 3-axis linear interpolation motion on module 1.
```




**3-axis linear interpolation mo**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 15 | 01 | 10 | 1F 40 | 00 07 | 0E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 64/0C 64 | *Sub_function code* |
| 1 | 00 00 | *MSW of fp1* |
| 2 | 27 10 | *LSW of fp1 (10000 = 0x2710)* |
| 3 | 00 00 | *MSW of fp2* |
| 4 | 27 10 | *LSW of fp2 (10000 = 0x2710)* |
| 5 | 00 00 | *MSW of fp3* |
| 6 | 27 10 | *LSW of fp3 (10000 = 0x2710)* |

### 6.2.12 2-Axis Circular Interpolation Motion (an Arc)

**eRET ETM_ARC_CW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)**

**※Δ eRET ETM_MACRO_ARC_CW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)**

**Description:**

**This function executes a 2-axis circular interpolation motion in a clockwise (CW) direction.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***cp1*:** | **The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).** |
| ***cp2*:** | **The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).** |
| ***fp1*:** | **The displacement of the axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)** |
| ***fp2*:** | **The displacement of the axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)** |

**Return:**

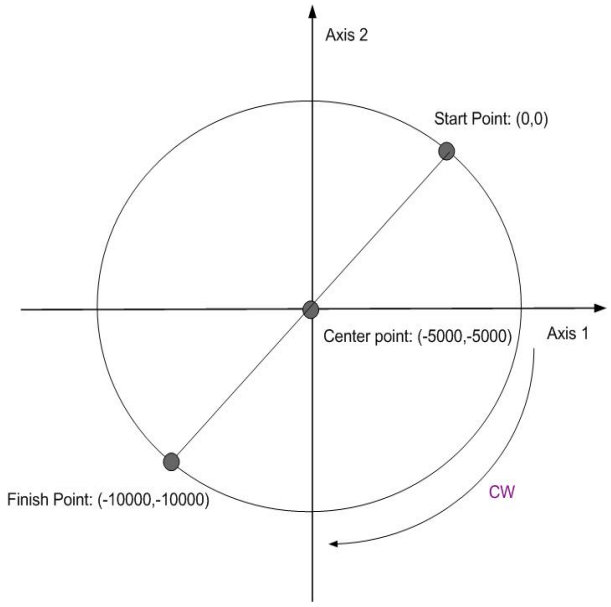
**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the errors.**

**Remark:**

**The Sub_function code of ETM_ARC_CW is 0A 65.**
**The Sub_function code of ETM_MACRO_ARC_CW is 0C 65.**

**MODBUS example:**

```
ETM_ARC_CW (h, 1, -5000, -5000, -10000, -10000);
// Issues a command to perform a circular motion (an arc)
in a CW direction. Please refer to the following figure.
```




**2-axis circular motion in a CW direction**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 19 | 01 | 10 | 1F 40 | 00 09 | 12 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 65/0C 65 | *Sub_function code* |
| 1 | FF FF | *MSW of cp1* |
| 2 | EC 78 | *LSW of cp1 (-5000 = 0xFFFFEC78)* |
| 3 | FF FF | *MSW of cp2* |
| 4 | EC 78 | *LSW of cp2 (-5000 = 0xFFFFEC78)* |
| 5 | FF FF | *MSW of fp1* |
| 6 | D8 F0 | *LSW of fp1 (-10000 = 0xFFFFD8F0)* |
| 7 | FF FF | *MSW of fp2* |
| 8 | D8 F0 | *LSW of fp2 (-10000 = 0xFFFFD8F0)* |

**eRET ETM_ARC_CCW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)**

**※Δ eRET ETM_MACRO_ARC_CCW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)**

**Description:**

**This function executes a 2-axis circular interpolation motion in a counter-clockwise (CCW) direction.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***cp1*:** | **The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).** |
| ***cp2*:** | **The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).** |
| ***fp1*:** | **The displacement of the axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)** |
| ***fp2*:** | **The displacement of the axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)** |

**Return:**

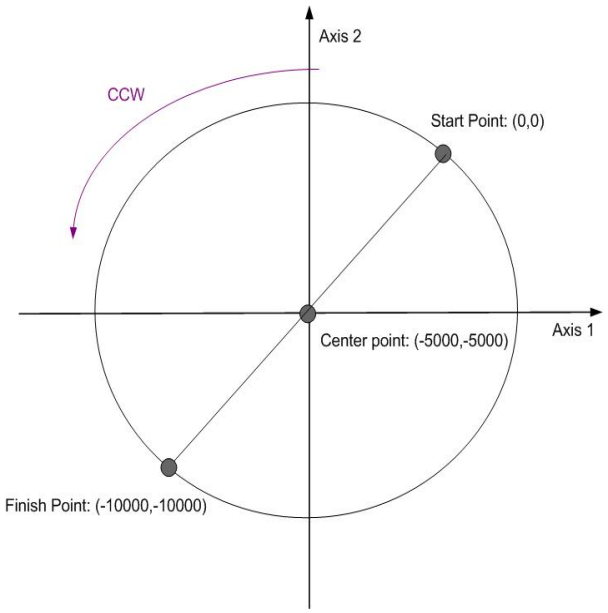
**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the errors.**

**Remark:**

**The Sub_function code of ETM_ARC_CCW is 0A 67.**
**The Sub_function code of ETM_MACRO_ARC_CCW is 0C 67.**

**MODBUS example:**

**ETM_ARC_CCW (h, 1, -5000, -5000, -10000, -10000);**
**// Issues a command to perform a circular motion (an arc) in a CCW direction. Please refer to the following figure.**




**2-axis circular motion in a CW direction**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 19 | 01 | 10 | 1F 40 | 00 09 | 12 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 67/0C 67 | *Sub_function code* |
| 1 | FF FF | *MSW of cp1* |
| 2 | EC 78 | *LSW of cp1 (-5000 = 0xFFFFEC78)* |
| 3 | FF FF | *MSW of cp2* |
| 4 | EC 78 | *LSW of cp2 (-5000 = 0xFFFFEC78)* |
| 5 | FF FF | *MSW of fp1* |
| 6 | D8 F0 | *LSW of fp1 (-10000 = 0xFFFFD8F0)* |
| 7 | FF FF | *MSW of fp2* |
| 8 | D8 F0 | *LSW of fp2 (-10000 = 0xFFFFD8F0)* |

### 6.2.13 2-Axis Circular Interpolation Motion (an Complete Circle)

| |
|---|
| eRET ETM_CIRCLE_CW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*) |
| ※Δ eRET ETM_MACRO_CIRCLE_CW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*) |

**Description:**

This function executes a 2-axis circular interpolation motion in a clockwise (CW) direction.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *cp1*: | The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000). |
| *cp2*: | The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000). |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)
Use the ETM_GET_ERROR_CODE() function to identify the errors.

**Remark:**

The Sub_function code of ETM_CIRCLE_CW is 0A 69.
The Sub_function code of ETM_MACRO_CIRCLE_CW is 0C 69.

**MODBUS example:**

```
ETM_CIRCLE_CW (h, 1, 0, 10000);
// execute a circular motion (a complete circle) in a CW direction
// on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 69/0C 69 | *Sub_function code* |
| 1 | 00 00 | *MSW of cp1* |
| 2 | 00 00 | *LSW of cp1 (0 = 0x0)* |
| 3 | 00 00 | *MSW of cp2* |
| 4 | 27 10 | *LSW of cp2 (10000 = 0x2710)* |

**eRET ETM_*CIRCLE*_CCW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*)**

**※Δ eRET *ETM*_MACRO_CIRCLE_CCW (HANDLE *h*, BYTE *cardNo*, long *cp1*, long *cp2*)**

**Description:**

**This function executes a 2-axis circular interpolation motion in a counter-clockwise (CCW) direction.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***cp1*:** | **The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).** |
| ***cp2*:** | **The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the errors.**

**Remark:**

**The Sub_function code of ETM_CIRCLE_CCW is 0A 6A.**
**The Sub_function code of ETM_MACRO_CIRCLE_CCW is 0C 6A.**

**MODBUS example:**

```
ETM_CIRCLE_CCW (h, 1, 0, 10000);
// execute a circular motion (a complete circle) in a CCW direction
//on module 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 6A/0C 6A | *Sub_function code* |
| 1 | 00 00 | *MSW of cp1* |
| 2 | 00 00 | *LSW of cp1 (0 = 0x0)* |
| 3 | 00 00 | *MSW of cp2* |
| 4 | 27 10 | *LSW of cp2 (10000 = 0x2710)* |

## 6.3 Synchronous Actions

### 6.3.1 Setting the Synchronous Action

**eRET ETM_SYNC_ACTION (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, DWORD *nSYNC*, BYTE *nDRV*, BYTE *nLATCH*, BYTE *nPRESET*, BYTE *nOUT*, BYTE *nINT*, BYTE *isrNoX*, BYTE *isrNoY*, BYTE *isrNoZ*, BYTE *isrNoU*)**

**※Δ eRET ETM_MACRO_SYNC_ACTION (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, DWORD *nSYNC*, BYTE *nDRV*, BYTE *nLATCH*, BYTE *nPRESET*, BYTE *nOUT*, BYTE *nINT*, BYTE *isrNoX*, BYTE *isrNoY*, BYTE *isrNoZ*, BYTE *isrNoU*)**

**Description:**

**This function sets the activation factors for synchronous action. (It's total-hardware-solution, will not consume any MP-8000 system resources).**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | This is the monitored axis. It will be checked by hardware. The axis can be either X,Y, Z or U (1 or 2 or 4 or 8) (Please refer to Table 2). |
| *axis2*: | This defined the other axes (or axis) that will take action when one of the activation factors occurs. |

***nSYNC*:** **It defines the activation factors. Multiple activation factors can be defined at the same time. Available active factors are listed in the following table.**

| Value | Event | Explanation |
|---|---|---|
| 0x00000000 | | Disable the synchronous action |
| 0x00000001 | P ≥ C+ | The logical/real position counter value exceeded the COMP+ register value.<br>Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000002 | P<C+ | The logical/real position counter value became less than the COMP+ register value.<br>Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000004 | P<C- | The logical/real position counter value became less than the COMP- register value.<br>Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000008 | P ≥ C- | The logical/real position counter value exceeded the COMP- register value.<br>Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000010 | D-STA | Driving started. |
| 0x00000020 | D-END | Driving terminated. |
| 0x00000040 | IN3 ↑ | The nIN3 signal rose from the Low to the High level. |
| 0x00000080 | IN3 ↓ | The nIN3 signal fell from the High to the Low level. |

**example: Choose P ≥ C+ and IN3 ↑ (0x00000001 + 0x00000040 = 0x00000041)**

***nDRV*:** **It defines the actions that are related with axial driving. Available actions are listed in the following table. Only one driving action can be chosen.**

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable driving action. |
| 1 | FDRV+ | Activates fixed pulse driving in the + direction. It must set the *nPRESET* value to be "OPSET" which indicates that ETM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, ETM_SET_PRESET(), is necessary. However, this command does not take effect if the assigned axes, *axis2*, are moving. |
| 2 | FDRV- | Activates fixed pulse driving in the - direction. It must set the *nPRESET* value to be "OPSET" which indicates that ETM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, ETM _SET_PRESET(), is necessary. However, this command does not take effect if the assigned axes, *axis2*, are moving. |
| 3 | CDRV+ | Activates continuous pulse driving in the + direction. However, this command does not take effect if the assigned axes, *axis2*, are moving |
| 4 | CDRV- | Activates continuous pulse driving in the - direction. However, this command does not take effect if the assigned axes, *axis2*, are moving. |
| 5 | SSTOP | Stop driving in deceleration |
| 6 | ISTOP | Stop driving immediately |

***nLATCH***: **It defines the actions that is related of latching position. Available actions are listed in the following table. Only one of these actions can be chosen**

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable position latch function |
| 1 | LPSAV | Saves the current logical position counter value (LP) in the synchronous buffer register (BR). [LP → LATCH] |
| 2 | EPSAV | Saves the current real position counter value (EP) in the synchronous buffer register (BR). [EP → LATCH] |

**After the event is occoured, the ETM_GET_LATCH() function can be use to get the latched value, please refer to section 6.3.3.**

***nPRESET***: **It defines the actions that is related of latching position. Available actions are listed in the following table. Only one of these actions can be chosen.**

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable setting function |
| 1 | LPSET | Indicates that a new value for the logical position (LP) will be set. The new value will be set by ETM_SET_PRESET() function. [LP ← PRESET] |
| 2 | EPSET | Indicates that a new value for the real position (EP) will be set. The new value will be set by ETM_SET_PRESET() function. [EP ← PRESET] |
| 3 | OPSET | Indicates that a new offset value (P) for the fixed pulse driving will be set. The new value will be set by ETM_SET_PRESET() function. [P ← PRESET] This setting is invalid to the Finish-Point of the axes in CONTINUE_MOVE. |
| 4 | VLSET | Indicates that a new speed value (V) will be set. The new value will be set by ETM_SET_PRESET() function. [V ← PRESET] |

**Must be used with ETM_SET_PRESET together, please refer to section 6.3.4.**

***nOUT***: **setting trigger output, as the following table:**

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable trigger output |
| 1 | OUT | Enable trigger output |

**Must be used with ETM_SET_OUT together, please refer to section 6.3.5.**

***nINT*:** **setting interrupt function, as the following table:**

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable interrupt function |
| 1 | INT | Enable interrupt function |

**1. It will generate a interrupt for the synchronous action of axis2, please write the corresponding number: *isrNoX* 、*isrNoY* 、*isrNoZ* 、*isrNoU***

**2. Must be used with ETM_ENABLE_INT together, please refer to section 6.3.6.**

***isrNoX*:** **ISR1 ~ ISR20 : Specify the interrupt number of X-axis.**
**0: disable**
***isrNoY*:** **ISR1 ~ ISR20 : Specify the interrupt number of Y-axis.**
**0: disable**
***isrNoZ*:** **ISR1 ~ ISR20 : Specify the interrupt number of Z-axis.**
**0: disable**
***isrNoU*:** **ISR1 ~ ISR20 : Specify the interrupt number of U-axis.**
**0: disable**

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SYNC_ACTION is 0A 6E.**
**The Sub_function code of ETM_MACRO_SYNC_ACTION is 0C 6E.**

**MODBUS example:**

**ETM_SYNC_ACTION (h, cardNo, AXIS_U, AXIS_U, 0X00000040, 0, 2, 4, 0, 0, 0, 0, 0, 0);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 23 | 01 | 10 | 1F 40 | 00 0E | 1C |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 6E/0C 6E | *Sub_function code* |
| 1 | 00 08 | *axis1 (8 = AXIS_U)* |
| 2 | 00 08 | *axis2* |
| 3 | 00 00 | *MSW of nSYNC* |
| 4 | 00 40 | *LSW of nSYNC (0x40 ➔ rising edge of IN3 will trigger the action)* |
| 5 | 00 00 | *nDRV (0 ➔ no driving control)* |
| 6 | 00 02 | *nLATCH (2 ➔ will latch EP value)* |
| 7 | 00 04 | *nPRESET (4 ➔ set new velocity when SYNC condition is true)* |
| 8 | 00 00 | *nOUT (0 ➔ no pulse out)* |
| 9 | 00 00 | *nINT (0 ➔ disable)* |
| 10 | 00 00 | *isrNoX (It can be any value since nINT is disable.)* |
| 11 | 00 00 | *isrNoY (It can be any value since nINT is disable.)* |
| 12 | 00 00 | *isrNoZ (It can be any value since nINT is disable.)* |
| 13 | 00 00 | *isrNoU (It can be any value since nINT is disable.)* |

**Related example:**

```
//Example1: When the U axis received IN3 positive edge trigger signal, it
//will change the LATCH encoder value.
 ETM_SYNC_ACTION(h, cardNo, AXIS_U, AXIS_U, 0X00000040, 0, 2, 4, 0, 0,
0, 0, 0, 0);
ETM_SET_MAX_V(h, cardNo, AXIS_U, 5000);
//Set the maximum speed of u axis on module 1 to 5K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_U, 0);
// set the speed profile for u axis as a symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_U, 2000);
// set the speed of u axis on module 1 to 2000 PPS
ETM_SET_A(h, cardNo, AXIS_U, 100000);
// set the acceleration value of u axis to 100K PPS/sec.
ETM_SET_SV(h ,cardNo, AXIS_U, 100);
//set the start velocity of u axis to  100 PPS
ETM_FIXED_MOVE(h, cardNo, AXIS_U, 10000);
```

```
//move 10000 Pulses for u axis on module 1
ETM_SET_PRESET(h cardNo, AXIS_U, 100);
// set the new speed of u axis on module 1 to 100 PPS
ETM_STOP_WAIT(h, cardNo, AXIS_U);
long Vsb = ETM_GET_LATCH(h, cardNo, AXIS_U);


//Example2: When the EP value of u axis exceeds COMP+(5000), it will start
//the Y-axis moving 2,000 PPS.
 ETM_SYNC_ACTION(h, cardNo, AXIS_U, AXIS_Y, 0X00000001, 1, 0, 3, 0, 0,
0, 0, 0, 0);
ETM_SET_COMPARE(h, cardNo, AXIS_U, 0, 1, 5000);
//Set the value of COMP+ is 5000, source reference the EP of U axis.
ETM_SET_MAX_V(h, cardNo, AXIS_YU, 9000);
// Set the maximum speed of YU axes on module 1 to 9K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_YU, 0);
// set the speed profile for YU axes as symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_YU, 3000);
// set the speed of YU axes on module 1 to 3000 PPS.
ETM_SET_A(h, cardNo, AXIS_YU, 200000);
// set the acceleration value of YU axes to 200K PPS/sec.
ETM_SET_SV(h, cardNo, AXIS_YU, 200);
// set the start velocity of yu axes to 200 PPS
ETM_FIXED_MOVE(h, cardNo, AXIS_U, 10000);
// move 10000 Pulses for U axis on module 1
ETM_SET_PRESET(h, cardNo, AXIS_Y, 2000);
// Set the Y axis moving 2000 PPS。


//Example3: When the LP value of X axis exceeds COMP+(200), it will
//start the nout output 5V of Y axis.
ETM_SYNC_ACTION(h, cardNo, AXIS_X, AXIS_Y, 0X00000001, 0, 0, 0, 1, 0, 0,
0, 0, 0);
ETM_SET_COMPARE(h, cardNo, AXIS_X, 0, 0, 200);
//Set the value of COMP+ is 200, source reference the LP of X axis.
ETM_SET_MAX_V(h, cardNo, AXIS_X, 5000);
// Set the maximum speed of X axis on module 1 to 5K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_X, 0);
// set the speed profile for X axis as a symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_X, 2000);
// set the speed of X axis on module 1 to 2000 PPS.
ETM_SET_A(h, cardNo, AXIS_X, 100000);
// set the acceleration value of X axis to 100K PPS/s.
ETM_SET_SV(h, cardNo, AXIS_X, 100);
// set the start velocity of X axis to 100 PPS
ETM_FIXED_MOVE(h, cardNo, AXIS_X, 500);
```

```
// move 500 Pulses for X axis on module 1
ETM_SET_OUT(h, cardNo, AXIS_Y, 1, 0);
// Set the Y axis high level output 5V.


//Example4: When the LP value of X axis exceeds COMP+(5000), it will
emergency stop and generate an interrupt to call ISR1.
ETM_SYNC_ACTION(h, cardNo, AXIS_X, AXIS_X, 0X00000001, 6, 0, 0, 0, 1,
ISR1, 0, 0, 0);
ETM_ENABLE_INT(h, cardNo);  //Enable interrupt
ETM_SET_COMPARE(h, cardNo, AXIS_X, 0, 0, 5000);
// Set the value of COMP+ is 5000, source reference the LP of X axis.
ETM_SET_MAX_V(h, cardNo, AXIS_XY, 5000);
// Set the maximum speed of XY axes on module 1 to 5K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_XY, 0);
// set the speed profile for XY axes as symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_XY, 2000);
// set the speed of XY axes on module 1 to 2000 PPS.
ETM_SET_A(h, cardNo, AXIS_XY, 100000);
// set the acceleration value of XY axes to 100K PPS/sec.
ETM_SET_SV(h, cardNo, AXIS_XY, 100);
// set the start velocity of XY axes  to 100 PPS.
ETM_FIXED_MOVE(h, cardNo, AXIS_X, 10000);
// move 10K Pulses for X axis on module 1


ISR1:
ETM_MP_ISR_CREATE(h, cardNo, ISR1); // Create ISR1
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_Y, 1000);
//move 1000 Pulses  for y axis on module 1
ETM_MACRO_MP_ISR_CLOSE(h, cardNo); // End ISR1


//Example5: Use the for loop to iterate, when the LP value of X axis
// exceeds COMP+(38000), it will generate an interrupt to call ISR1, and
// determine whether receive IN3 signal in the ISR1, if receive then
// emergency stop.
ETM_MP_CREATE(h, cardNo, MP59); //Create a macro program MP59.
ETM_MACRO_SET_MAX_V(h, cardNo, AXIS_XY, 5000);
// Set the maximum speed of XY axes on module 1 to 5K PPS.
ETM_MACRO_NORMAL_SPEED(h, cardNo, AXIS_XY, 0);
// set the speed profile for XY axes as symmetric T-curve.
ETM_MACRO_SET_V(h, cardNo, AXIS_XY, 2000);
// set the speed of XY axes  on module 1 to 2000 PPS.
ETM_ MACRO_SET_A(h, cardNo, AXIS_XY, 100000);
// set the acceleration value of XY axes to 100K PPS/S.
```

```
ETM_ MACRO_SET_SV(h, cardNo, AXIS_XY, 100);
// set the start velocity of XY axes to 100 PPS.
ETM_MACRO_FOR(h, cardNo, 100); //Set the times of for loop is 100.
ETM_ MACRO_SET_LP(h, cardNo, AXIS_XY, 0);
//Set the LP value of XYaxes to 0
ETM_MACRO_SET_FILTER(h, cardNo, AXIS_X, 16, 1); //Set the filter of IN3
ETM_MACRO_CLEAR_SYNC_ACTION(h, cardNo, AXIS_X);
//Clear all the sync_action conditions of X axis.
ETM_MACRO_SYNC_ACTION(h, cardNo, AXIS_X, AXIS_X, 0X00000001, 0, 0,
0, 0, 1, ISR1, 0, 0, 0);
// Set the LP value of X axis exceeds COMP+, it will generate an interrupt to
// call ISR1.
ETM_MACRO_ENABLE_INT(h, cardNo);  //Enable interrupt
ETM_MACRO_SET_COMPARE(h, cardNo, AXIS_X, 0, 0, 38000);
//Set the value of COMP+ is 38000, source reference the LP of X axis.
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_X, 42000);
// move 42K Pulses for X axis on module 1
ETM_MACRO_STOP_WAIT(h, cardNo, AXIS_X); //Wait until X axis stop.
ETM_MACRO_NEXT(h, cardNo); //Match with for loop.
ETM_MACRO_MP_CLOSE(h, cardNo); // End a macro program MP59


ISR1:
ETM_MP_ISR_CREATE(h, cardNo, ISR1); //Create ISR1
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_Y, 1000);
// move 1000 Pulses for Y axis on module 1.
ETM_MACRO_SYNC_ACTION(h, cardNo, AXIS_X, AXIS_X, 0X00000040, 6, 0,
0, 0, 0, 0, 0, 0); //Set the X axis to receive the IN3 positive edge trigger
//signals from low level to high level, then emergency stop.
ETM_MACRO_MP_ISR_CLOSE(h, cardNo); // End ISR1
```

**eRET ETM_CLEAR_SYNC_ACTION (HANDLE *h*, BYTE *cardNo*, BYTE *triggerAxis*)**

**※Δ eRET ETM_MACRO_CLEAR_SYNC_ACTION (HANDLE *h*, BYTE *cardNo*, BYTE *triggerAxis*)**

**Description:**

Clear all the synchronization conditions of the triggered axis.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *triggerAxis* | The axis number set by ETM_SYNC_ACTION (Please refer to Table 2) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_CLEAR_SYNC_ACTION is 0A 6A.
The Sub_function code of ETM_MACRO_CLEAR_SYNC_ACTION is 0C 6A.

**MODBUS example:**

```
ETM_CLEAR_SYNC_ACTION (h, 1, AXIS_X);
// Clear all the sync_action conditions of X axis.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 95/0C 95 | *Sub_function code* |
| 1 | 00 01 | *TriggerAxis* |

**eRET ETM_SET_ACTIVATION_FACTORS(HANDLE *h*, BYTE *cardNo*, BYTE *triggerAxis*, DWORD *nSYNC*)**

**※Δ eRET ETM_MACRO_SET_ACTIVATION_FACTORS (HANDLE *h*, BYTE *cardNo*, BYTE *triggerAxis*, DWORD *nSYNC*)**

**Description:**

Set the synchronization conditions of triggered axis.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *triggerAxis* | Set the axis number(Please refer to Table 2) |

*nSYNC*: It defines the activation factors. Multiple activation factors can be defined at the same time. Available active factors are listed in the following table.

| Value | Event | Explanation |
|---|---|---|
| 0x00000000 | | Disable the synchronous action |
| 0x00000001 | P≥C+ | The logical/real position counter value exceeded the COMP+ register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000002 | P<C+ | The logical/real position counter value became less than the COMP+ register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000004 | P<C- | The logical/real position counter value became less than the COMP- register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000008 | P≥C- | The logical/real position counter value exceeded the COMP- register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2). |
| 0x00000010 | D-STA | Driving started. |
| 0x00000020 | D-END | Driving terminated. |
| 0x00000040 | IN3↑ | The nIN3 signal rose from the Low to the High level. |
| 0x00000080 | IN3↓ | The nIN3 signal fell from the High to the Low level. |

example: Choose P≥C+ and IN3↑ (0x00000001 + 0x00000040 = 0x00000041)

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_ACTIVATION_FACTORS is 0A 6A.**
**The Sub_function code of ETM_MACRO_SET_ACTIVATION_FACTORS is 0C 6A.**

**MODBUS example:**

**ETM_SET_ACTIVATION_FACTORS (h, 1, AXIS_X, 0X00000001);**
**// Set the synchronization conditions of x axis is P≥C+.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 96/0C 96 | *Sub_function code* |
| 1 | 00 01 | *TriggerAxis* |
| 2 | 00 00 | *MSW of nSYNC* |
| 3 | 00 01 | *LSW of nSYNC* |

**eRET ETM_SET_ACTIVATION_AXIS(HANDLE *h*, BYTE *cardNo*, BYTE *triggerAxis*, BYTE *activationAxis*)**

**※Δ eRET ETM_MACRO_SET_ACTIVATION_FACTORS (HANDLE *h*, BYTE *cardNo*, BYTE *triggerAxis*, DWORD *nSYNC*)**

**Description:**

Set the synchronization axis.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *triggerAxis* | Set the axis number(Please refer to Table 2) |
| *activationAxis*: | Set the synchronization axis number(Please refer to Table 2) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_SET_ACTIVATION_AXIS is 0A 97.

The Sub_function code of ETM_MACRO_SET_ACTIVATION_AXIS is 0C 97.

**MODBUS example:**

```
ETM_SET_ACTIVATION_AXIS(h, 1, AXIS_X, AXIS_Y);
// Set the synchronization axis of x axis is y axis.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 97/0C 97 | *Sub_function code* |
| 1 | 00 01 | *TriggerAxis(AXIS_X)* |
| 2 | 00 02 | *ActivationAxis(AXIS_Y)* |

**eRET ETM_SET_ACTION(HANDLE *h*, BYTE *cardNo*, BYTE *activationAxis*, BYTE *nDRV*, BYTE *nLATCH*, BYTE *nPRESET*, BYTE *nOUT*, BYTE *nINT*, BYTE *isrNoX*, BYTE *isrNoY*, BYTE *isrNoZ*, BYTE *isrNoU*)**

**※Δ eRET ETM_MACRO_SET_ACTION (HANDLE *h*, BYTE *cardNo*, BYTE *activationAxis*, BYTE *nDRV*, BYTE *nLATCH*, BYTE *nPRESET*, BYTE *nOUT*, BYTE *nINT*, BYTE *isrNoX*, BYTE *isrNoY*, BYTE *isrNoZ*, BYTE *isrNoU*)**

**Description:**

Set the action of synchronization motion.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *activationAxis*: | Set the synchronization axis number(Please refer to Table 2) |

*nDRV*: It defines the actions that are related with axial driving. Available actions are listed in the following table. Only one driving action can be chosen.

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable driving action. |
| 1 | FDRV+ | Activates fixed pulse driving in the + direction. It must set the *nPRESET* value to be "OPSET" which indicates that ETM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, ETM_SET_PRESET(), is necessary. However, this command does not take effect if the assigned axes, *axis2*, are moving. |
| 2 | FDRV- | Activates fixed pulse driving in the - direction. It must set the *nPRESET* value to be "OPSET" which indicates that ETM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, ETM _SET_PRESET(), is necessary. However, this command does not take effect if the assigned axes, *axis2*, are moving. |
| 3 | CDRV+ | Activates continuous pulse driving in the + direction. However, this command does not take effect if the assigned axes, *axis2*, are moving |

| | | |
|---|---|---|
| 4 | CDRV- | Activates continuous pulse driving in the - direction. However, this command does not take effect if the assigned axes, *axis2*, are moving. |
| 5 | SSTOP | Stop driving in deceleration |
| 6 | ISTOP | Stop driving immediately |

***nLATCH***: It defines the actions that is related of latching position. Available actions are listed in the following table. Only one of these actions can be chosen

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable position latch function |
| 1 | LPSAV | Saves the current logical position counter value (LP) in the synchronous buffer register (BR). [LP → LATCH] |
| 2 | EPSAV | Saves the current real position counter value (EP) in the synchronous buffer register (BR). [EP → LATCH] |

**After the event is occoured, the ETM_GET_LATCH() function can be use to get the latched value, please refer to section 6.3.3.**

***nPRESET***: It defines the actions that is related of latching position. Available actions are listed in the following table. Only one of these actions can be chosen.

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable setting function |
| 1 | LPSET | Indicates that a new value for the logical position (LP) will be set. The new value will be set by ETM_SET_PRESET() function. [LP ← PRESET] |
| 2 | EPSET | Indicates that a new value for the real position (EP) will be set. The new value will be set by ETM_SET_PRESET() function. [EP ← PRESET] |
| 3 | OPSET | Indicates that a new offset value (P) for the fixed pulse driving will be set. The new value will be set by ETM_SET_PRESET() function. [P ← PRESET] This setting is invalid to the Finish-Point of the axes in CONTINUE_MOVE. |
| 4 | VLSET | Indicates that a new speed value (V) will be set. The new value will be set by ETM_SET_PRESET() function. [V ← PRESET] |

**Must be used with ETM_SET_PRESET together, please refer to section 6.3.4.**

***nOUT***: setting trigger output, as the following table:

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable trigger output |
| 1 | OUT | Enable trigger output |

**Must be used with ETM_SET_OUT together, please refer to section 6.3.5.**

***nINT*:** **setting interrupt function, as the following table:**

| Value | Symbol | Explanation |
|---|---|---|
| 0 | | Disable interrupt function |
| 1 | INT | Enable interrupt function |

**1. It will generate a interrupt for the synchronous action of axis2, please write the corresponding number: *isrNoX* 、*isrNoY* 、*isrNoZ* 、*isrNoU***

**2. Must be used with ETM_ENABLE_INT together, please refer to section 6.3.6.**

***isrNoX*:** **ISR1 ~ ISR20 : Specify the interrupt number of X-axis.**
**0: disable**
***isrNoY*:** **ISR1 ~ ISR20 : Specify the interrupt number of Y-axis.**
**0: disable**
***isrNoZ*:** **ISR1 ~ ISR20 : Specify the interrupt number of Z-axis.**
**0: disable**
***isrNoU*:** **ISR1 ~ ISR20 : Specify the interrupt number of U-axis.**
**0: disable**

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_ACTION is 0A 98.**
**The Sub_function code of ETM_MACRO_SET_ACTION is 0C 98.**

**MODBUS example:**

```
ETM_SET_ACTION (h, 1, AXIS_Y, 0, 2, 4, 0, 0, 0, 0, 0, 0);
// Set the action of synchronization motion is when the IN3 positive
//edge trigger signals received, it will change the velocity and the
//encoder of LATCH.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 1D | 01 | 10 | 1F 40 | 00 0B | 16 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 98/0C 98 | *Sub_function code* |
| 1 | 00 02 | *ActivationAxis(AXIS_Y)* |
| 2 | 00 00 | *nDRV (0 ➔ no driving control)* |
| 3 | 00 02 | *nLATCH (2 ➔ will latch EP value)* |
| 4 | 00 04 | *nPRESET (4 ➔ set new velocity when SYNC condition is true)* |
| 5 | 00 00 | *nOUT (0 ➔ no pulse out)* |
| 6 | 00 00 | *nINT (0 ➔ disable)* |
| 7 | 00 00 | *isrNoX (It can be any value since nINT is disable.)* |
| 8 | 00 00 | *isrNoY (It can be any value since nINT is disable.)* |
| 9 | 00 00 | *isrNoZ (It can be any value since nINT is disable.)* |
| 10 | 00 00 | *isrNoU (It can be any value since nINT is disable.)* |

**Related example:**

```
//Example1: Set two groups synchronization motion, when the LP value of x
//axis exceeds COMP+(30000), the synchronization motion axis z-axis
//generate an interrupt to call ISR1, and when the LP value of y axis
//exceeds COMP+(35000), the synchronization motion axis u-axis generate
//an interrupt to call ISR2.
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 5000);
// Set the maximum speed of xyzu axes on module 1 to 5K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU, 0);
// set the speed profile for xyzu axes as symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
// set the speed of xyzu axes on module 1 to 2000 PPS.
ETM_SET_A(h, cardNo, AXIS_XYZU, 100000);
// set the acceleration value of xyzu axes to 100K PPS/sec.
ETM_SET_SV(h, cardNo, AXIS_XYZU, 100);
// set the start velocity of xyzu axes to 100 PPS.
ETM_SET_ACTIVATION_FACTORS(h, cardNo, AXIS_XY, 1);
// Set the synchronization conditions of xy axes is P≥C+.
ETM_SET_ACTIVATION_AXIS(h, 1, AXIS_X, AXIS_Z);
// Set the synchronization axis of x axis is z axis.
ETM_SET_ACTIVATION_AXIS(h, 1, AXIS_Y, AXIS_U);
```

```
// Set the synchronization axis of y axis is u axis.
ETM_SET_ACTION (h, 1, AXIS_Z, 0, 0, 0, 0, 1, 0, 0, ISR1, 0);
// Set the action of synchronization motion axis z-axis is to generate an
// interrupt to call ISR1.
ETM_SET_ACTION (h, 1, AXIS_U, 0, 0, 0, 0, 1, 0, 0, 0, ISR2);
// Set the action of synchronization motion axis u-axis is to generate an
// interrupt to call ISR2.
ETM_ENABLE_INT(h, cardNo);  //Enable interrupt
ETM_SET_COMPARE (h, cardNo, AXIS_X, 0, 0, 30000);
//Set the value of COMP+ is 30000, source reference the LP of x axis.
ETM_SET_COMPARE (h, cardNo, AXIS_Y, 0, 0, 35000);
//Set the value of COMP+ is 35000, source reference the LP of y axis.
ETM_FIXED_MOVE(h, cardNo, AXIS_XY, 42000);
// move 42K Pulses for xy axes on module 1

ISR1:
ETM_MP_ISR_CREATE(h, cardNo, ISR1); // Create ISR1
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_Z, 10000);
// move 10K Pulses for z axis on module 1
ETM_MACRO_MP_ISR_CLOSE(h, cardNo); // End ISR1

ISR2:
ETM_MP_ISR_CREATE(h, cardNo, ISR2); // Create ISR2
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_U, 10000);
// move 10K Pulses for u axis on module 1
ETM_MACRO_MP_ISR_CLOSE(h, cardNo); // End ISR2
```

### 6.3.2 Setting the COMPARE Value

**eRET ETM_SET_COMPARE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nSELECT*, BYTE *nTYPE*, long *data*)**

**※Δ eRET ETM_MACRO_SET_COMPARE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nSELECT*, BYTE *nTYPE*, long *data*)**

**Description:**

This function sets the values of COMPARE registers. Howerer, it will disable the functions of software limits.

**Category:**

MODBUS sub_function; RTC, MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *nSELECT*: | 0 → C+<br>1 → C- |
| *nTYPE*: | 0 → Position(P) = LP<br>1 → Position(P) = EP |
| *data*: | Set the COMPARE value<br>(-2,000,000,000 ~ +2,000,000,000). |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_SET_COMPARE is 0A 70.
The Sub_function code of ETM_MACRO_SET_COMPARE is 0C 70.

**MODBUS example:**

```
ETM_SET_COMPARE (h, cardNo, AXIS_U, 0, 1, 5000);
//Set the comparison function for U-Axis.
//Set the compared source to be EP; and the COMP+ value to 5000.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 13 | 01 | 10 | 1F 40 | 00 06 | 0C |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 70/0C 70 | *Sub_function code* |
| 1 | 00 08 | *Axis (8 ➔ AXIS_U)* |
| 2 | 00 00 | *nSELECT* |
| 3 | 00 01 | *nTYPE* |
| 4 | 00 00 | *MSW of data* |
| 5 | 13 88 | *LSW of data (5000 = 0x1388)* |

### 6.3.3 Get the LATCH Value

```
eRET  ETM_GET_LATCH (HANDLE h, BYTE cardNo, BYTE axis, long* LatchValue)

※Δ eRET  ETM_MACRO_GET_LATCH (HANDLE h, BYTE cardNo, BYTE axis)

eRET  ETM_GET_LATCH_4_AXIS (HANDLE h, BYTE cardNo, long* LatchValueX, long* LatchValueY, long* LatchValueZ, long* LatchValueU)
```

**Description:**

**This function gets the values from the LATCH register.**

**Category:**

**MODBUS table, MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |
| *LatchValue*: | The pointer points to the memory that stores the value of LATCH.<br>(-2,000,000,000 ~ +2,000,000,000). |
| *LatchValueX:* | The pointer points to the memory that stores the value of LATCH.<br>(-2,000,000,000 ~ +2,000,000,000) of x axis. |
| *LatchValueY:* | The pointer points to the memory that stores the value of LATCH.<br>(-2,000,000,000 ~ +2,000,000,000) of y axis. |
| *LatchValueZ:* | The pointer points to the memory that stores the value of LATCH.<br>(-2,000,000,000 ~ +2,000,000,000) of z axis. |
| *LatchValueU:* | The pointer points to the memory that stores the value of LATCH.<br>(-2,000,000,000 ~ +2,000,000,000) of u axis. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

■ **Method 1: It can get latched values directly.**

**The latched values are always long type values. Therefore, polling the latched values must start at the MSW of each axis's MSW.**

| STOP status | Address | Remarks |
|---|---|---|
| X_LATCH_MSW | 62 (0x3E) | MSW of X_LATCH |
| X_LATCH_LSW | 63 (0x3F) | LSW of X_LATCH |
| Y_LATCH_MSW | 64 (0x40) | MSW of Y_LATCH |
| Y_LATCH_LSW | 65 (0x41) | LSW of Y_LATCH |
| Z_LATCH_MSW | 66 (0x42) | MSW of Z_LATCH |
| Z_LATCH_LSW | 67 (0x43) | LSW of Z_LATCH |
| U_LATCH_MSW | 68 (0x44) | MSW of U_LATCH |
| U_LATCH_LSW | 69 (0x45) | LSW of U_LATCH |

```
long LATCH_Y;
ETM_GET_LATCH (h, 1, AXIS_Y, &LATCH_Y);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 | 04 | 00 40 | 00 02 |

**The response via MODBUS may be:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | *MSW of Y_LATCH* ( Reg_0 ) | *LSW of Y_LATCH* ( Reg_1 ) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 07 | 01 | 04 | 04 | 00 00 | 03 E8 |

```
LATCH_Y = Register[0];
LATCH_Y = (long) ( ( (LATCH_Y << 16) & 0xffff0000) | (Register[1] & 0xffff) );
```

■ **Method 2: It can be used inside a MP program.**

**For this case, users do not actually want to get the current latched values. The getting latched values will be executed only when the MP is called. Therefore, use FC = 16 to write this command inside a MP. This kind of usage often has ETM_MACRO_SET_RVAR() followed to save the return latched value. Please refer to MP related explanation literature.**

```
ETM_MACRO_GET_LATCH (h, 1, AXIS_Y);
//Get the latched value which is from Y-axis of card 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C 71 | *Sub_function code* |
| 1 | 00 02 | *axis* |
| | | |

### 6.3.4 Set the PRESET data for synchronous acton

**eRET ETM_SET_PRESET (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *data*)**

**※Δ eRET ETM_MACRO_SET_PRESET (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, long *data*)**

**Description:**

**This function sets the PRESET value for synchronous action. Each synchronous action axis could not be set individually.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | The number of synchronization motion axis, this axis must be assined as the axis2 parameter of ETM_SYNC_ACTION( ). |
| *data*: | LP: (-2,000,000,000 ~ +2,000,000,000)<br>EP: (-2,000,000,000 ~ +2,000,000,000)<br>P : (-2,000,000,000 ~ +2,000,000,000)<br>V : Please refer to section 2.5. If there are more than two synchronous action axes, please set ETM_SET_MAX_V to be same value. |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_PRESET is 0A 72.**
**The Sub_function code of ETM_MACRO_SET_PRESET is 0C 72.**

**MODBUS example:**

**If the SYNC action is set to change velocity, then following statement will change the velocity of AXIS_U to 100 PPS when the condition is true.**

```
ETM_SET_PRESET (h, cardNo, AXIS_U, 100);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 72/0C 72 | *Sub_function code* |
| 1 | 00 08 | *Axis (8 ➔ AXIS_U)* |
| 2 | 00 00 | *MSW of data* |
| 3 | 00 64 | *LSW of data (100 = 0x64)* |

### 6.3.5 Set the OUT Data

**eRET ETM_SET_OUT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *outEdge*, BYTE *PulseWidth*)**

**※Δ eRET ETM_MACRO_SET_OUT (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *outEdge*, BYTE *PulseWidth*)**

**Description:**

**This function configures the output pulse settings.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2), currently only support Axes X & Y |
| *outEdge*: | Output active logic:<br>0 = low active; 1 = high active |
| *PulseWidth*: | Output pulse width<br>0 = 10 uSec<br>1 = 20 uSec<br>2 = 100 uSec<br>3 = 200 uSec<br>4 = 1,000 uSec<br>5 = 2,000 uSec<br>6 = 10,000 uSec<br>7 = 20,000 uSec |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_SET_OUT is 0A 73.**
**The Sub_function code of ETM_MACRO_SET_OUT is 0C 73.**

**MODBUS example:**

**If the SYNC action enables the digital OUT function, then following statement can define the waveform of digital output that includes the level and the pulse width.**

```
ETM_SET_OUT (h, 1, AXIS_U, 1, 0);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 73/0C 73 | *Sub_function code* |
| 1 | 00 08 | *Axis (8 ➔ AXIS_U)* |
| 2 | 00 01 | *outEdge* |
| 3 | 00 00 | *PulseWidth* |

## 6.3.6 Interrupt functions of motion module (i-8094H)

**eRET ETM_ENABLE_INT (HANDLE *h*, BYTE *cardNo*)**

**※ eRET ETM_MACRO_ENABLE_INT (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function enables the interrupt function of the motion chip inside the i8094H.**

**Category:**

**MODBUS sub_function; RTC, MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_ENABLE_INT is 0A AA.**
**The Sub_function code of ETM_MACRO_ENABLE_INT is 0C AA.**

**MODBUS example:**

**ETM_ENABLE_INT (h, 1);**
**//Enable the interrupt function for module 1.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0A AA/0C AA | | *Sub_function code* | | | |
| | | | | | | | |

**eRET ETM_DISABLE_INT (HANDLE *h*, BYTE *cardNo*)**

**※ eRET ETM_MACRO_DISABLE_INT (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function disables the interrupt function of the motion chip inside the i8094H.**

**Category:**

**MODBUS sub_function; RTC, MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_DISABLE_INT is 0A AB.**
**The Sub_function code of ETM_MACRO_DISABLE_INT is 0C AB.**

**MODBUS example:**

**ETM_DISABLE_INT (h, 1);**
**//Disable the interrupt function for module 1.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 0A AB/0C AB | | *Sub_function code* | | | |
| | | | | | | | |

**eRET ETM_INTFACTOR_ENABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nINT*, BYTE *isrNo*)**

**※Δ eRET ETM_*MACRO*_INTFACTOR_ENABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nINT*, BYTE *isrNo*)**

**Description:**

This function sets the condition factor of interrupt of the motion chip.

**Category:**

MODBUS sub_function; RTC, MP, ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |

*nINT*: condition factors of interrupt, please refer to the following table

**Table 5: Motion chip interrupt factor table**

| nINT | Symbol | Description |
|---|---|---|
| 0 | PULSE | Interupt occurs when pulse is up |
| 1 | P>=C- | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register. Please refer to ETM_SET_COMPARE() (6.3.2) |
| 2 | P<C- | Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register Please refer to ETM_SET_COMPARE() (6.3.2) |
| 3 | P<C+ | Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register. Please refer to ETM_SET_COMPARE() (6.3.2) |
| 4 | P>=C+ | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register. Please refer to ETM_SET_COMPARE() (6.3.2) |
| 5 | C-END | Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output |
| 6 | C-STA | Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output |
| 7 | D-END | Interrupt occurs when the driving is finished |

*isrNo*: ISR1 ~ ISR20 : Specify the number of interrupt service.

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**This function is conflict with ETM_SET_SLMT(2.10).**
**The Sub_function code of ETM_INTFACTOR_ENABLE is 0A AC.**
**The Sub_function code of ETM_MACRO_INTFACTOR_ENABLE is 0C AC.**

**MODBUS example:**

**This function assigns an ISRn to process an interrupt that happens at a specified axis.**

```
ETM_INTFACTOR_ENABLE (h, 1, AXIS_X, 4, ISR1);//When the position counter of x axis exceeds C+, then call ISR1.
```

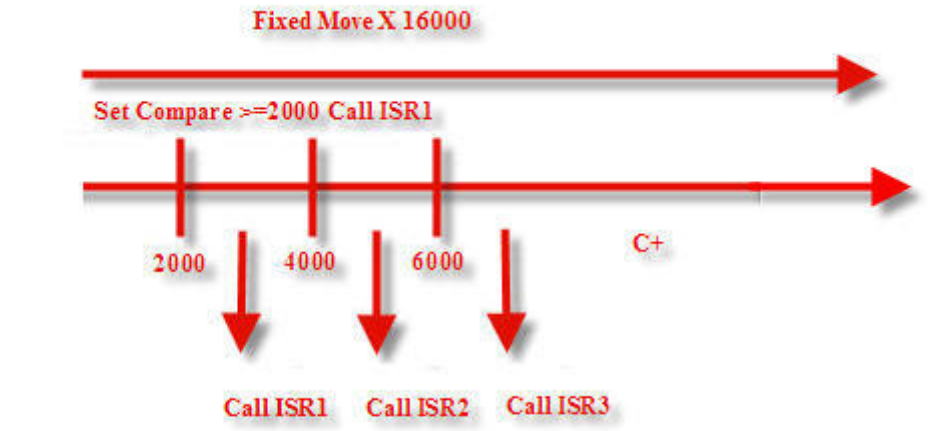
**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0F | 01 | 10 | 1F 40 | 00 04 | 08 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A AC/0C AC | *Sub_function code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| 2 | 00 04 | *nINT* |
| 3 | 00 01 | *isrNo (ISR1 = 1)* |

**Related example:**

**//Example1: Using call the ISR to set multi-group compare.**




ISR1:
Set Compare >=4000 Call ISR2
Fixed Move Y 1000

ISR2:
Set Compare >=6000 Call ISR3
Fixed Move Z 1000

ISR3:
Fixed Move U 1000

```
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 5000);
// Set the maximum  speed of xyzu axes on module 1 to 5K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_XYZU, 0);
// set the speed profile for xyzu axes as symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_XYZU, 2000);
// set the speed of xyzu axes on module 1 to 2000 PPS.
ETM_SET_A(h, cardNo, AXIS_XYZU, 100000);
// set the acceleration value of xyzu axes to 100K PPS/S.
ETM_SET_SV(h, cardNo, AXIS_XYZU, 100);
// set the start velocity of xyzu axes to 100 PPS.
ETM_SET_COMPARE(h, cardNo, AXIS_X, 0, 0, 2000);
//Set the value of COMP+ is 2000, source reference the LP of x axis.
ETM_INTFACTOR_ENABLE (h, 1, AXIS_X, 4, ISR1);
//When the LP value of x axis exceeds C+(2000), then call ISR1.
ETM_ENABLE_INT (h, 1);    //Enable interrupt
ETM_FIXED_MOVE(h, cardNo, AXIS_X, 16000);
// move 16K Pulses for x axis on module 1

ISR1:
ETM_MP_ISR_CREATE(h, cardNo, ISR1); //Create ISR1
ETM_MACRO_SET_COMPARE(h, cardNo, AXIS_X, 0, 0, 4000);
//Set the value of COMP+ is 4000, source reference the LP of x axis.
ETM_MACRO_INTFACTOR_ENABLE (h, 1, AXIS_X, 4, ISR2);
```

```
//When the LP value of x axis exceeds C+(4000), then call ISR2.
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_Y, 1000);
// move 1000 Pulses for y axis on module 1
ETM_MACRO_MP_ISR_CLOSE(h, cardNo); // End ISR1

ISR2:
ETM_MP_ISR_CREATE(h, cardNo, ISR2); //Create ISR2
ETM_MACRO_SET_COMPARE(h, cardNo, AXIS_X, 0, 0, 6000);
//Set the value of COMP+ is 6000, source reference the LP of x axis.
ETM_MACRO_INTFACTOR_ENABLE (h, 1, AXIS_X, 4, ISR3);
//When the LP value of x axis exceeds C+(6000), then call ISR3.
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_Z, 1000);
// move 1000 Pulses for z axis on module 1
ETM_MACRO_MP_ISR_CLOSE(h, cardNo); // End ISR2

ISR3:
ETM_MP_ISR_CREATE(h, cardNo, ISR3); //Create ISR3
ETM_MACRO_FIXED_MOVE(h, cardNo, AXIS_U, 1000);
// move 1000 Pulses for u axis on module 1
ETM_MACRO_MP_ISR_CLOSE(h, cardNo); // End ISR3
```

**eRET ETM_INTFACTOR_DISABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nINT*)**

**※Δ eRET ETM_MACRO_INTFACTOR_DISABLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *nINT*)**

**Description:**

**This function disables the condition factor of interrupt of the motion chip.**

**Category:**

**MODBUS sub_function; RTC, MP, ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |

***nINT*:** condition factors of interrupt, please refer to the following table

| Number | Symbol | Description |
|---|---|---|
| 0 | PULSE | Interupt occurs when pulse is up |
| 1 | P>=C- | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register. |
| 2 | P<C- | Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register |
| 3 | P<C+ | Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register. |
| 4 | P>=C+ | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register. |
| 5 | C-END | Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output |
| 6 | C-STA | Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output |
| 7 | D-END | Interrupt occurs when the driving is finished |
| 10 | DISABLE ALL | Disable all the trigger conditions interrupt factor |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_INTFACTOR_DISABLE is 0A AD.**
**The Sub_function code of ETM_MACRO_INTFACTOR_DISABLE is 0C AD.**

**MODBUS example:**

```
ETM_INTFACTOR_DISABLE (h, 1, AXIS_XYZU, 4);
// Disable the interrupt factors of Module 1
// 4 : Interrupt occurs when the value of logic / real position counter
// is larger than or equal to that of COMP+ register.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A AD/0C AD | *Sub_function code* |
| 1 | 00 0F | *axis (0xF ➔ AXIS_XYZU)* |
| 2 | 00 04 | *nINT* |
| | | |

**Related example:**

```
ETM_INTFACTOR_DISABLE(h, 1, AXIS_XYZU, 1);
ETM_INTFACTOR_DISABLE(h, 1, AXIS_XYZU, 2);
ETM_INTFACTOR_DISABLE(h, 1, AXIS_XYZU, 3);
ETM_INTFACTOR_DISABLE(h, 1, AXIS_XYZU, 4);
// Disable the interrupt factors of Module 1
```

## 6.4 Read triggered interrupt factors

**eRET ETM_GET_TRIG_INTFACTOR (HANDLE *h*, BYTE *cardNo*, BYTE *nINT*, BYTE **pAxis*);**

**Description:**

**This function reads which axis has triggered an interrupt for the set interrupt factor since the last ETM_INTFACTOR_ENABLE function call. The function ETM_INTFACTOR_ENABLE enables the motion chip interrupt and clears the interrupt status for the respective interrupt factor.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *nINT:* | Consult Table 6 |
| *pAxis:* | Consult Table 2<br>Example:<br>0x0F: All 4 axis have triggered<br>0x02: only the Y axis has triggered<br>0x0B: axis XYU have triggered |

**The Modbus function code 4 register table is as follows.**

**Table 6: Motion chip interrupt factor table with respective Modbus FC 4 register addresses**

| Modbus register address FC4 | *nINT* | Symbol | Description |
|---|---|---|---|
| 130 | 0 | PULSE | Interupt occurs when pulse is up |
| 131 | 1 | P>=C- | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register. |
| 132 | 2 | P<C- | Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register |
| 133 | 3 | P<C+ | Interrupt occurs when the value of logical / real position counter is smaller than that of |

| | | | |
|---|---|---|---|
| | | | COMP+ register. |
| 134 | 4 | P>=C+ | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register. |
| 135 | 5 | C-END | Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output |
| 136 | 6 | C-STA | Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output |
| 137 | 7 | D-END | Interrupt occurs when the driving is finished |
| 138 | 8 | HMEND | Automatic home search terminated |
| 139 | 9 | SYNC | Synchronous action was activated |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
BYTE bAxis;
ETM_GET_TRIG_INTFACTOR (h, 1, 0, & bAxis);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 (Card No.) | 04 | 00 82 (130) (Pulse output) | 00 01 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Registers (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 09 (Axis XY triggered) |

## 6.5 Macro download status

```
  eRET ETM_GET_MP_DOWNLOAD_STATUS(HANDLE h, BYTE cardNo,
        MpDownloadInfo *pInfo);

typedef struct
{
     WORD MpNo;
     WORD IsrNo;
     WORD wCmd;
     WORD wLineNo;
     WORD wErrCode;
}MpDownloadInfo;
```

**Description:**

**This function confirms whether the previous MP or ISR macro download procedure was successful. It indicate which Macro type has been downloaded (MP or ISR) and its number. In case of a download error the type of error, the command and Macro line number will be indicated.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***pInfo:*** | ***MpNo*: indicates the MP Macro number (1-MP1, 2 - MP2, ...). If no MP Macro has been downloaded MpNo will be zero (MpNo=0).**<br><br>***IsrNo*: indicates the latest ISR Macro number downloaded(1-ISR1, 2 -ISR2, ...) . If no ISR Macro has been downloaded IsrNo will be zero (IsrNo=0).**<br><br>***wCmd*: code of the command in the Macro program which caused the error (see command code chapter 0).**<br><br>***wLineNo*: the line number in the Macro program. The first line has the number zero.**<br><br>***wErrCode*: error code describing the cause of macro download failure:**<br>**– *0x00*: no error occurred. Download was** |

| | |
|---|---|
| | **succesful.**<br>– ***0x01*: number of commands lines exceeds the quantity of line provide by the corresponding Macro table.**<br><br>**<u>IF command errors:</u>**<br>– **0x02: Number of nested MP_IF statements exceeds the limit of six inner nested MP_IFs.**<br>– ***0x03*: MP_ELSE command outside MP_IF block**<br>– ***0x04*: MP_END_IF command outside MP_IF block. The MP_END_IF statement has not got a corresponding MP_IF statement.**<br>– ***0x05*: MP_IF block has not been closed by MP_END_IF command.**<br>– ***0x06*: Wrong MP_IF conditional operator.**<br><br>**<u>FOR loop command errors:</u>**<br>– **0x07: Number of nested MP_FOR exceeds the limit.**<br>– ***0x08*: MP_EXIT_FOR command outside MP_FOR loop. The corresponding MP_FOR command is missing.**<br>– ***0x09*: MP_NEXT command outside MP_FOR block. The corresponding MP_FOR command is missing.**<br>– ***0x0A*: MP_FOR block has not been closed with a MP_NEXT command.**<br><br>**<u>GOTO command errors:</u>**<br>– ***0x0B*: Label number is being used more than once.**<br>– ***0x0C*: MP_LABEL inside a MP_IF block. Jumping into a MP_IF block is prohibited.**<br>– ***0x0D*: MP_LABEL inside a MP_FOR block. Jumping into a MP_FOR block is prohibited.**<br>– **0x0E: MP_GOTO command has NO corresponding MP_LABEL command.**<br><br>– ***0x0F*: MACRO (MP or ISR )command not defined: Command will not be written to the FRAM.**<br>– ***0x10*: Wrong MP_VAR_CALCULATE operator.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
MpDownloadInfo MpInfo;
ETM_GET_MP_DOWNLOAD_STATUS(h, 1, &MpInfo);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 (Card No.) | 04 | 00 6E (110) | 00 05 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Registers (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 04 | 0A | See next table |

| Register [] | Value (hex) | Remarks |
|---|---|---|
| 0 | 00 02 | *MpNo:* MP number (e.g. MP2) |
| 1 | 00 00 | *IsrNo:* ISR number (no ISR Macro has been downloded) |
| 2 | 0A D6 | *wCmd:* command code (here MP_NEXT) |
| 3 | 00 09 | *wLineNo*: command line within Macro table (line number 9) |
| 4 | 00 09 | *wErrCode:* error code describing the kind of error (here: MP_NEXT command outside MP_FOR block. The corresponding MP_FOR command is missing.) |

## 6.6 Read ETM_MACRO_SET_RINT triggered interrupt

**The MP Macro command ETM_MACRO_SET_RINT interrupts the host controller. As soon as the ETM_MACRO_SET_RINT command inside a MP Macro program is being executed the i8094H module interrupts the ET-M8194H device and informing that the command has been executed.**

---

**eRET ETM_GET_USER_RINT (HANDLE *h*, BYTE *cardNo*, BYTE* *bState*);**

**Description:**

**This function asks the ET-M8194H whether a ETM_MACRO_SET_RINT command has been executed. The interrupt flag will be cleared after the reading operation.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *bState:* | 1: an ETM_MACRO_SET_RINT interrupt occurred<br>0: no interrupt occurred since the last call |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
BYTE bState;
ETM_GET_USER_RINT (h, 1, & bState);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 (Card No.) | 04 | 00 48 (72) | 00 01 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Register (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 01 ETM_MACRO_SET_RINT interrupt occurred |

## 6.7 Get ET-M8194H state

**The motion unit basically consists of two modules:**

- **ET-M8194H**
- **i8094H**

**• The ET-M8194H module (Figure 1) in slot 0 is the unit which is mainly responsible for the Modbus communication between the Master and Slave. The main task of the module is to verify and filter every command arriving at its TCP/IP port according to its current state and consequently decide whether to transfer the command to the i8094H or if the command is not supported by the respective state ignore the command and respond to the Master with a Modbus exception code.**

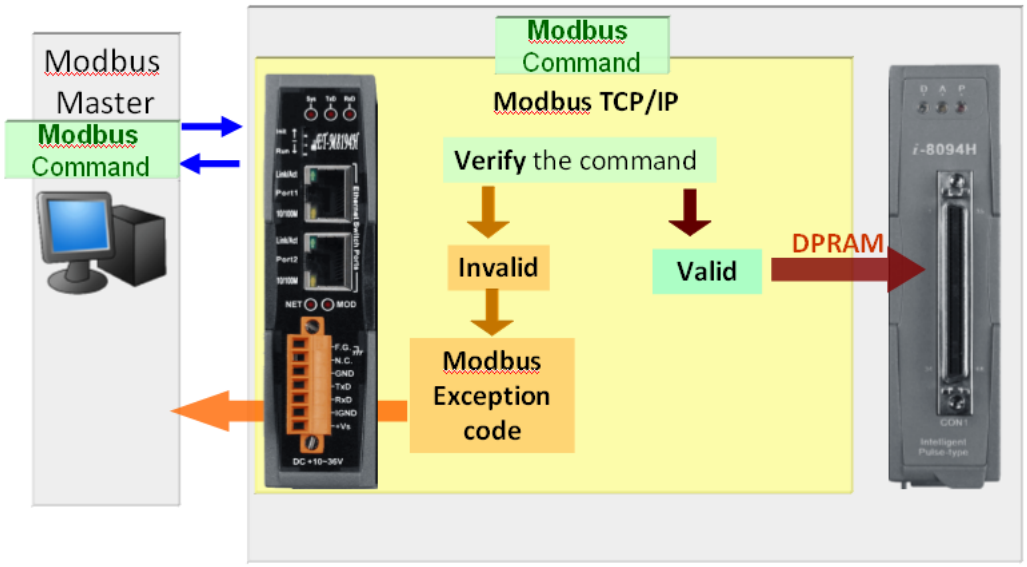



**Figure 1: ET-M8194H module in slot 0 of the motion device**

**The module can be in one of the following three states:**

- **Ready state**
- **MP Macro download state**
- **ISR Macro download state**

**When the ET-M8194H (module in the first slot is responsible for the communication part) is in a certain state certain commands can not be processed by the module and therefore the module responds with a Modbus exception code when a command is being sent by the Master which is not being supported by the current module state. Table 1 describes for each command the valid scope. When the module is in the ready state all RTC commands marked in Table 1 are valid commands.**

**For example, the ET-M8194H is in the MP Macro download state. In this state only commands which are marked with MP in Table 1 are valid commands. As soon as a non MP command is being sent (e.g. MP_CREATE) the module will ignore this**

**command and return a Modbus exception code. The ETM_GET_DEVICE_STATE allows the user to determine the reason for the Modbus exception code by reading the current module state.**

**• The i8094H module (Figure 2) in slot 1 is mainly responsible for executing motion commands and storing the Macro tables in its non-volatile memory. The motion control chip and the non-volatile memory for storing Macro tables are part of the i8094H module. Furthermore it has a DPRAM (volatile FIFO buffer) which stores up to 29 commands for further processing.**

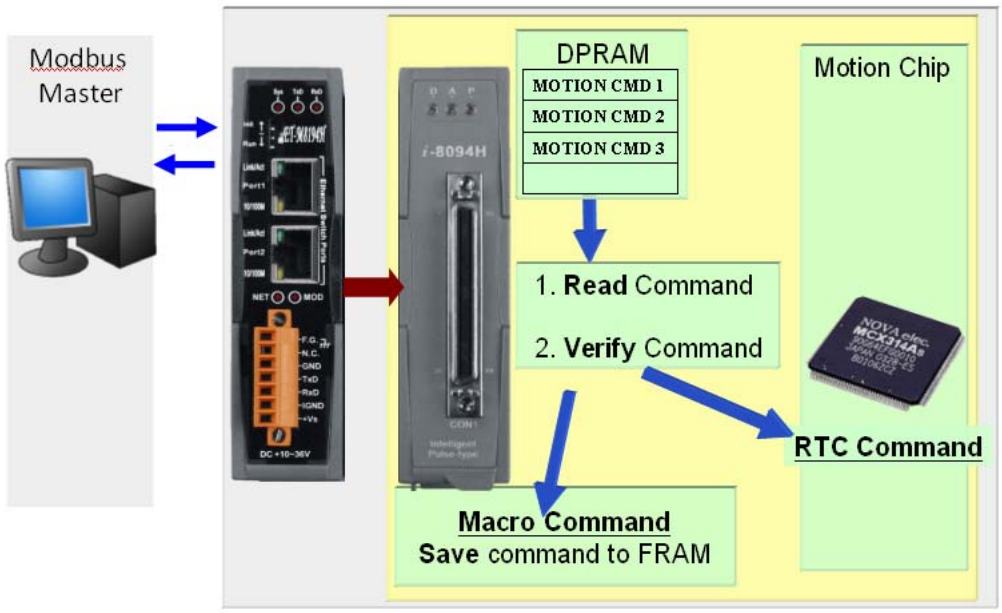



**Figure 2: i8094H situated in slot 1 of the motion device**

```
  eRET  ETM_GET_DEVICE_STATE (HANDLE h, BYTE cardNo, EtmState
        *pState);

typedef struct
{
     WORD Etm8194hState;
     WORD I8094hState;
     WORD IllegalFuncErr;
}EtmState;
```

**Description:**

**This function returns the current state of the ET-M8194H and of the i8094H module and furthermore the cause of the last Modbus exception code.**

**Category:**

**MODBUS table; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***pState:*** | ***Etm8194hState* (Table 7)**<br>**ETM_READY:**<br>**only RTC commands can be sent.**<br><br>**ETM_MP_DOWNLOAD:**<br>**only MP Macro commands are accepted.**<br><br>**ETM_ISR_DOWNLOAD:**<br>**only ISR Macro commands are valid.**<br><br>***I8094hState* (Table 8)**<br>**i8094H_READY:**<br>**The i8094H is ready for receiving RTC commands.**<br><br>**i8094H_INITIALIZATION:**<br>**executing initialization table (IT commands).**<br><br>**i8094H_MACRO_DOWNLOAD:**<br>**i8094h is in the MACRO download mode.**<br><br>**i8094H_MP_EXECUTION:**<br>**MP Macro table is being executed.** |

| | |
|---|---|
| | **i8094H_ISR_EXECUTION:**<br>**Interrupt MACRO (ISR) program is being executed.**<br><br>**i8094H_ERR_FIRMWARE:**<br>**Module exited the DPRAM scan engine, Module needs to be rebooted.**<br><br>**i8094H_FIRMWARE_BOOTING:**<br>**I8094H firmware has not completed the booting process.**<br><br>***IllegalFuncErr* (Table 9)**<br>**ERR_NO_ERROR:**<br>**No error occurred.**<br><br>**ERR_FC_NOT_SUPPORTED:**<br>**Function code not supported.**<br><br>**ERR_EXCEED_MP_MEMORY:**<br>**Macro program exceeds the number of command lines reserved for the selected table.**<br><br>**ERR_MP_INSIDE_ISR:**<br>**A MP command is being used inside ISR Macro.**<br><br>**ERR_ISR_INSIDE_MP:**<br>**A ISR command is being used inside MP Macro.**<br><br>**ERR_RTC_INSIDE_MP_OR_ISR:**<br>**A RTC command is being used inside a MP or ISR Macro.**<br><br>**ERR_MP_OR_ISR_AS_RTC:**<br>**A MP or ISR Macro command is being used outside a Macro program.** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
EtmState etmState;
ETM_GET_DEVICE_STATE (h, 1, & etmState);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01 (Card No.) | 04 | 00 64 (100) | 00 03 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Registers (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 04 | 0A | See next table |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 00 01 | *Etm8194hState:* ETM_MP_DOWNLOAD |
| 1 | 00 03 | *I8094hState :* i8094H_MACRO_DOWNLOAD |
| 2 | 00 06 | *IllegalFuncErr:* ERR_RTC_INSIDE_MP_OR_ISR |

**Table 7: ET-M8194H states**

| States of the ET-M8194H module | Number (hex) | Description |
|---|---|---|
| ETM_READY | 0x00 | only RTC commands can be sent |
| ETM_MP_DOWNLOAD | 0x01 | only MP Macro commands are accepted |
| ETM_ISR_DOWNLOAD | 0x02 | only ISR Macro commands are valid |

**Table 8: i8094h states**

| Different states of the i8094H | Number (hex) | Description |
|---|---|---|
| i8094H_READY | 0x00 | The i8094H is ready for receiving RTC commands |
| i8094H_INITIALIZATION | 0x01 | executing initialization table (IT commands) |
| i8094H_MACRO_DOWNLOAD | 0x03 | i8094h is in the MACRO download mode |
| i8094H_MP_EXECUTION | 0x05 | MP Macro table is being executed |
| i8094H_ISR_EXECUTION | 0x07 | Interrupt MACRO (ISR) program is being executed |
| | | |
| i8094H_ERR_FIRMWARE | 0x10 | I8094H firmware experienced an error and has to be power off/on |
| i8094H_FIRMWARE_BOOTING | 0xFF | I8094H firmware has not completed the booting process |

**Table 9: Causes of Modbus exception response**

| Illegal function errors | Number (hex) | Description |
|---|---|---|
| ERR_NO_ERROR | 0x00 | No error occurred |
| ERR_FC_NOT_SUPPORTED | 0x01 | Function code not supported |
| ERR_EXCEED_MP_MEMORY | 0x03 | Macro program exceeds the number of command lines reserved for the selected table |
| ERR_MP_INSIDE_ISR | 0x04 | A MP command is being used inside ISR Macro |
| ERR_ISR_INSIDE_MP | 0x05 | A ISR command is being used inside MP Macro |
| ERR_RTC_INSIDE_MP_OR_ISR | 0x06 | A RTC command is being used inside a MP or ISR Macro |
| ERR_MP_OR_ISR_AS_RTC | 0x07 | A MP or ISR Macro command is being used outside a Macro program |

## 6.8 Continuous Interpolation

### 6.8.1 2-Axis Rectangular Motion

eRET ETM_RECTANGLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *nAcc*, BYTE *Sp*, BYTE *nDir*, long *Lp*, long *Wp*, long *Rp*, DWORD *RSV*, DWORD *RV*, DWORD *RA*, DWORD *RD*)

※ eRET ETM_MACRO_RECTANGLE (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *nAcc*, BYTE *Sp*, BYTE *nDir*, long *Lp*, long *Wp*, long *Rp*, DWORD *RSV*, DWORD *RV*, DWORD *RA*, DWORD *RD*)

**Description:**

Continuous interpolation will be performed to create a rectangular motion, which is formed by 4 lines and 4 arcs. The length of each side can be changed. The radius of each arc is of the same value and it can also be changed. The deceleration point will be calculated automatically. This is a command that appears in various motion applications. **(It will not consume any system resources of MP-8000).**

**Category:**

MODBUS sub_function; RTC and MP.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (axis 1). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis (axis 2). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *nAcc*: | 0 → constant vector speed interpolation mode<br>1 → symmetric T-curve Acc/Dec interpolation mod |
| *Sp*: | Start point 0 ~ 7. (Sp0 ~ Sp7 are defined in the following. |
| *nDir*: | Direction of movement<br>0: CCW; 1: CW |
| *Lp*: | Length in Pulses (1 ~ 2,000,000,00) |
| *Wp*: | Width in Pulses (1 ~ 2,000,000,00) |
| *Rp*: | Radius of each in pulses (1 ~ 2,000,000,00) |
| *RSV*: | Starting speed (in PPS) |
| *RV*: | Vector speed (in PPS) |
| *RA*: | Acceleration (PPS/sec) |
| *RD*: | Deceleration of the last segment (in PPS/sec) |

**Return:**

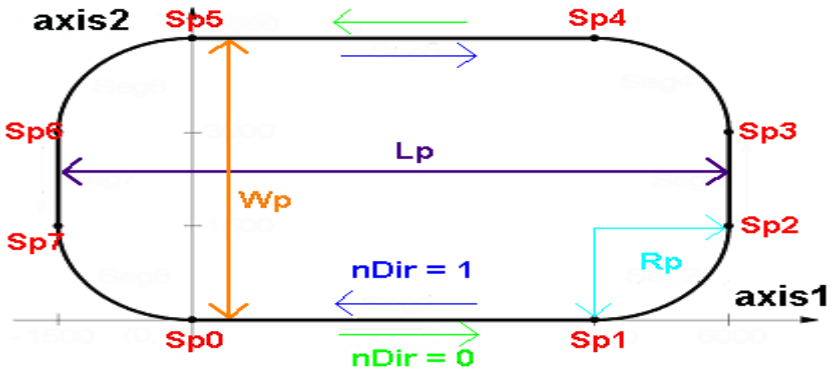
**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the error.**

**Remark:**

**The Sub_function code of ETM_RECTANGLE is 0A 78.**
**The Sub_function code of ETM_MACRO_RECTANGLE is 0C 78.**

**MODBUS example:**

**ETM_RECTANGLE (h, 1, AXIS_X, AXIS_Y, 1, 0, 0, 20000, 10000, 1000, 1000, 10000, 5000, 5000);**
**//execute a rectangular motion on XY plane, will auto-calculate**
**//deceleration point.**




**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 2F | 01 | 10 | 1F 40 | 00 14 | 28 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 78/0C 78 | *Sub_funciton code* |
| 1 | 00 01 | *axis1 (1 ➔ AXIS_X)* |
| 2 | 00 02 | *axis2 (2 ➔ AXIS_Y)* |
| 3 | 00 01 | *nAcc (velocity profile)* |
| 4 | 00 00 | *Sp (choose the start point from 0~7 )* |
| 5 | 00 00 | *nDir (set the direction to be CCW)* |
| 6 | 00 00 | *MSW of Lp (length of the rectangle)* |
| 7 | 4E 20 | *LSW of Lp (20000 = 0x4E20)* |
| 8 | 00 00 | *MSW of Wp (width of the rectangle)* |
| 9 | 27 10 | *LSW of Wp (10000 = 0x2710)* |
| 10 | 00 00 | *MSW of Rp (set the corner radius value)* |
| 11 | 03 E8 | *LSW of Rp (1000 = 0x3E8)* |
| 12 | 00 00 | *MSW of RSV (define start velocity)* |

| | | |
|---|---|---|
| **13** | **03 E8** | ***LSW of RSV (1000 = 0x3E8)*** |
| **14** | **00 00** | ***MSW of RV (define velocity)*** |
| **15** | **27 10** | ***LSW of RV (10000 = 0x2710)*** |
| **16** | **00 00** | ***MSW of RA (define acc value)*** |
| **17** | **13 88** | ***LSW of RA (5000 = 1388)*** |
| **18** | **00 00** | ***MSW of RD (define dec value)*** |
| **19** | **13 88** | ***LSW of RD (5000 = 1388)*** |

### 6.8.2 2-Axis Continuous Linear Interpolation

**eRET ETM_LINE_2D_INITIAL (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, DWORD *VSV* , DWORD *VV* , DWORD *VA*)**

**※ eRET ETM_MACRO_LINE_2D_INITIAL (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, DWORD *VSV* , DWORD *VV* , DWORD *VA*)**

**Description:**

**This function sets the necessary parameters for a 2-axis continuous linear interpolation using symmetric T-curve speed profile.**

**Category:**

**MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (axis 1). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis (axis 2). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *VSV*: | Starting speed (in PPS) |
| *VV*: | Vector speed (in PPS) |
| *VA*: | Vector acceleration (PPS/sec) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the error.**

**Remark:**

**The Sub_function code of ETM_LINE_2D_INITIAL is 0A 7C.**
**The Sub_function code of ETM_MACRO_LINE_2D_INITIAL is 0C 7C.**

**MODBUS example:**

```
ETM_LINE_2D_INITIAL (h, 1, AXIS_X, AXIS_Y, 300, 18000, 500000);
//This function should be defined before the ETM_LINE_2D_CONTINUE()
//function is used. Please refer to the example of this function.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 19 | 01 | 10 | 1F 40 | 00 09 | 12 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 7C/0C 7C | *Sub_funciton code* |
| 1 | 00 01 | *axis1 (1 ➔ AXIS_X)* |
| 2 | 00 02 | *axis2 (2 ➔ AXIS_Y)* |
| 3 | 00 00 | *MSW of VSV* |
| 4 | 01 2C | *LSW of VSV (300 = 0x12C)* |
| 5 | 00 00 | *MSW of VV* |
| 6 | 46 50 | *LSW of VV (18000 = 0x4650)* |
| 7 | 00 07 | *MSW of VA* |
| 8 | A1 20 | *LSW of VA (500000 = 0x0007A120)* |

**eRET ETM_LINE_2D_CONTINUE (HANDLE *h*, BYTE *cardNo*, BYTE *nType*, long *fp1*, long *fp2*)**

**※ eRET ETM_MACRO_LINE_2D_CONTINUE (HANDLE *h*, BYTE *cardNo*, BYTE *nType*, long *fp1*, long *fp2*)**

**Description:**

**This function executes a 2-axis continuous linear interpolation. (It will not consume any system resources of MP-8000).**

**Category:**

**MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***nType*:** | **0 → 2-axis linear continuous interpolation<br>1 → end of 2-axis linear continuous interpolation** |
| ***fp1*:** | **The assigned number of pulses for the axis 1 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)** |
| ***fp2*:** | **The assigned number of pulses for the axis 2 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the error.**

**Remark:**

**The Sub_function code of ETM_LINE_2D_CONTINUE is 0A 7E.**
**The Sub_function code of ETM_MACRO_LINE_2D_CONTINUE is 0C 7E.**

**MODBUS example:**

```
ETM_LINE_2D_CONTINUE (h, 1, 0, 100, 100);
//execute X, Y 2-axis linear continuous interpolation on module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 13 | 01 | 10 | 1F 40 | 00 06 | 0C |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 7E/0C 7E | *Sub_funciton code* |
| 1 | 00 00 | *nType* |
| 2 | 00 00 | *MSW of fp1* |
| 3 | 00 64 | *LSW of fp1 (100 = 0x64)* |
| 4 | 00 00 | *MSW of fp2* |
| 5 | 00 64 | *LSW of fp2* |
| | | |

**Related example:**

```
BYTE cardNo=1;
unsigned short sv=300; //Set the vector start velocity is 300 PPS。
unsigned short v=18000; //Set the vector velocity is 18000 PPS。
unsigned long a=500000; //Set the vector acceleration is 500K PPS/s。
unsigned short loop1;
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 160000L);
// Set the maximum  speed of xyzu axes on module 1 to 160K PPS.
ETM_LINE_2D_INITIAL(h, cardNo, AXIS_X, AXIS_Y, sv, v, a);
for (loop1 = 0; loop1 < 10000; loop1++)
{
        ETM_LINE_2D_CONTINUE (h, cardNo, 0, 100, 100);
        ETM_LINE_2D_CONTINUE (h, cardNo, 0, -100, -100);
        //execute X, Y 2-axis linear continuous interpolation on module 1
}
ETM_LINE_2D_CONTINUE (h, cardNo, 1, 100, 100);
//end X, Y 2-axis linear continuous interpolation on module 1
```

### 6.8.3 3-Axis Continuous Linear Interpolation

eRET ETM_LINE_3D_INITIAL (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*, DWORD *VSV* , DWORD *VV* , DWORD *VA*)

※ eRET ETM_MACRO_LINE_3D_INITIAL (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*, DWORD *VSV* , DWORD *VV* , DWORD *VA*)

**Description:**

This function sets the necessary parameters for a 3-axis continuous linear interpolation using symmetric T-curve speed profile.

**Category:**

MODBUS sub_function; RTC and MP.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (axis 1). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis (axis 2). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis3*: | The third axis (axis 3). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *VSV*: | Starting speed (in PPS) |
| *VV*: | Vector speed (in PPS) |
| *VA*: | Vector acceleration (PPS/sec) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_LINE_3D_INITIAL is 0A 7F.
The Sub_function code of ETM_MACRO_LINE_3D_INITIAL is 0C 7F.

**MODBUS example:**

```
ETM_LINE_3D_INITIAL (h, 1, AXIS_X, AXIS_Y, AXIS_Z, 300, 18000, 500000);
//This function should be defined before the ETM_LINE_3D_CONTINUE()
//function is used. Please refer to the example of this function.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 1B | 01 | 10 | 1F 40 | 00 0A | 14 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 7F/0C 7F | *Sub_funciton code* |
| 1 | 00 01 | *axis1 (1 ➔ AXIS_X)* |
| 2 | 00 02 | *axis2 (2 ➔ AXIS_Y)* |
| 3 | 00 04 | *axis3 (4 ➔ AXIS_Z)* |
| 4 | 00 00 | *MSW of VSV* |
| 5 | 01 2C | *LSW of VSV (300 = 0x12C)* |
| 6 | 00 00 | *MSW of VV* |
| 7 | 46 50 | *LSW of VV (18000 = 0x4650)* |
| 8 | 00 07 | *MSW of VA* |
| 9 | A1 20 | *LSW of VA (500000 = 0x7A120)* |

**eRET ETM_LINE_3D_CONTINUE (HANDLE *h*, BYTE *cardNo*, BYTE *nType*, long *fp1*, long *fp2*, long *fp3*)**

**※ eRET ETM_MACRO_LINE_3D_CONTINUE (HANDLE *h*, BYTE *cardNo*, BYTE *nType*, long *fp1*, long *fp2*, long *fp3*)**

**Description:**

**This function executes a 3-axis continuous linear interpolation. (It will not consume any system resources of MP-8000).**

**Category:**

**MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***nType*:** | **0 → 3-axis linear continuous interpolation<br>1 → end of 3-axis linear continuous interpolation** |
| ***fp1*:** | **The assigned number of pulses for the axis 1 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)** |
| ***fp2*:** | **The assigned number of pulses for the axis 2 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)** |
| ***fp3*:** | **The assigned number of pulses for the axis 3 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the error.**

**Remark:**

**The Sub_function code of ETM_LINE_3D_CONTINUE is 0A 81.**
**The Sub_function code of ETM_MACRO_LINE_3D_CONTINUE is 0C 81.**

**MODBUS example:**

```
ETM_LINE_3D_CONTINUE (h, 1, 0, 100, 100, 100);
//execute X, Y, Z 3-axis linear continuous interpolation on module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 17 | 01 | 10 | 1F 40 | 00 08 | 10 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 81/0C 81 | *Sub_funciton code* |
| 1 | 00 00 | *nType* |
| 2 | 00 00 | *MSW of fp1* |
| 3 | 00 64 | *LSW of fp1 (100 = 0x64)* |
| 4 | 00 00 | *MSW of fp2* |
| 5 | 00 64 | *LSW of fp2* |
| 6 | 00 00 | *MSW of fp3* |
| 7 | 00 64 | *LSW of fp3* |
| | | |

**Related example:**

```
BYTE cardNo=1;
unsigned short sv=300; //Set the vector start velocity is 300 PPS。
unsigned short v=18000; //Set the vector velocity is 18000 PPS。
unsigned long a=500000; //Set the vector acceleration is 500K PPS/s。
unsigned short loop1;
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 160000L);
// Set the maximum speed of xyzu axes on module 1 to 160K PPS.
ETM_LINE_3D_INITIAL(h, cardNo, AXIS_X, AXIS_Y, AXIS_Z, sv, v, a);
for (loop1 = 0; loop1 < 10000; loop1++)
{
        ETM_LINE_3D_CONTINUE(h, cardNo, 0, 100, 100, 100);
        ETM_LINE_3D_CONTINUE(h, cardNo, 0, -100, -100, -100);
        //execute X, Y, Z 3-axis linear continuous interpolation on module 1
}
ETM_LINE_3D_CONTINUE (h, 1, 0, 100, 100, 100);
//end X, Y, Z 3-axis linear continuous interpolation on module 1
```

### 6.8.4 Mixed Linear and Circular 2-axis motions in Continuous Interpolation

```
eRET  ETM_MIX_2D_INITIAL (HANDLE h, BYTE cardNo, BYTE axis1,
      BYTE axis2, BYTE nAcc, DWORD VSV , DWORD VV , DWORD VA)

※ eRET  ETM_MACRO_MIX_2D_INITIAL (HANDLE h, BYTE cardNo, BYTE
      axis1, BYTE axis2, BYTE nAcc, DWORD VSV , DWORD VV ,
      DWORD VA)
```

**Description:**

This function does the initial settings for mixed linear and circular 2-axis motions in continuous interpolation.

**Category:**

MODBUS sub_function; RTC and MP.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (axis 1). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis (axis 2). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *nAcc*: | 0 → constant speed (VV)<br>1 → symmetric T-curve Acc/Dec (VSV、VV、VA) |
| *VSV*: | Starting speed (in PPS) |
| *VV*: | Vector speed (in PPS) |
| *VA*: | Vector acceleration (PPS/Sec) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_MIX_2D_INITIAL is 0A 82.
The Sub_function code of ETM_MACRO_MIX_2D_INITIAL is 0C 82.

**MODBUS example:**

```
ETM_MIX_2D_INITIAL (h, 1, 1, 2, 0, 300, 18000, 500000);
//This function should be defined before the ETM_MIX_2D_CONTINUE()
//function is used. Please refer to the example of this function.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 1B | 01 | 10 | 1F 40 | 00 0A | 14 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 82/0C B2 | *Sub_funciton code* |
| 1 | 00 01 | *axis1 (1 ➔ AXIS_X)* |
| 2 | 00 02 | *axis2 (2 ➔ AXIS_Y)* |
| 3 | 00 00 | *nAcc* |
| 4 | 00 00 | *MSW of VSV* |
| 5 | 01 2C | *LSW of VSV (300 = 0x12C)* |
| 6 | 00 00 | *MSW of VV* |
| 7 | 46 50 | *LSW of VV (18000 = 0x4650)* |
| 8 | 00 07 | *MSW of VA* |
| 9 | A1 20 | *LSW of VA (500000 = 0x7A120)* |

eRET ETM_MIX_2D_CONTINUE (HANDLE *h*, BYTE *cardNo*, BYTE *nAcc*, BYTE *nType*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

※ eRET ETM_MACRO_MIX_2D_CONTINUE (HANDLE *h*, BYTE *cardNo*, BYTE *nAcc*, BYTE *nType*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

**Description:**

This function executes mixed linear and circular 2-axis motion in continuous interpolation. (It will not consume any system resources of MP-8000).

**Category:**

MODBUS sub_function; RTC and MP.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *nAcc*: | 0 → continuous interpolation.<br>1 → it is the last command of this continuous interpolation.<br>In Acc/Dec mode, it will perform a deceleration stop. In constant speed mode, it will directly stop rather than decelerate. |
| *nType*: | 1 → ETM_LINE_2D(BYTE *cardNo*, long *fp1*, long*fp2*)<br><br>2 → ETM_ARC_CW(BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)<br><br>3 → ETM_ARC_CCW(BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)<br><br>4 → ETM_CIRCLE_CW(BYTE *cardNo*, long *cp1*, long *cp2*)<br><br>5 → ETM_CIRCLE_CCW(BYTE *cardNo*, long *cp1*, long *cp2*) |
| *cp1*: | It assigns the center point data at axis 1 (-2,000,000,000 ~ +2,000,000,000) |
| *cp2*: | It assigns the center point data at axis 2 (-2,000,000,000 ~ +2,000,000,000) |
| *fp1*: | It assigned number of pulses for axis 1 (-2,000,000,000 ~ +2,000,000,000) |
| *fp2*: | It assigned number of pulses for axis 2 (-2,000,000,000 ~ +2,000,000,000) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the error.**

**Remark:**

**The Sub_function code of ETM_MIX_2D_CONTINUE is 0A 84.**
**The Sub_function code of ETM_MACRO_MIX_2D_CONTINUE is 0C 84.**

**MODBUS example:**

**ETM_MIX_2D_CONTINUE (h, 1, 0, 1, 0, 0, 100, 100);**
**//execute X, Y, 2-axis motions in continuous interpolation on module 1**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 1D | 01 | 10 | 1F 40 | 00 0B | 16 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 84/0C 84 | *Sub_funciton code* |
| 1 | 00 00 | *nAcc* |
| 2 | 00 01 | *nType* |
| 3 | 00 00 | *MSW of cp1* |
| 4 | 00 00 | *LSW of cp1 (for linear motion, set 0)* |
| 5 | 00 00 | *MSW of cp2* |
| 6 | 00 00 | *LSW of cp2 (for linear motion, set 0)* |
| 7 | 00 00 | *MSW of fp1* |
| 8 | 00 64 | *LSW of fp1 (100 = 0x64)* |
| 9 | 00 00 | *MSW of fp2* |
| 10 | 00 64 | *LSW of fp2 (100 = 0x64)* |

### 6.8.5 Multi-Segment Continuous Interpolation(Using Array)

```
eRET ETM_CONTINUE_INTP(HANDLE h, BYTE cardNo, BYTE axis1,
     BYTE axis2, BYTE axis3, BYTE nAcc, DWORD VSV, DWORD VV,
     DWORD VA, DWORD VD)
```

**Description:**

This function executes multi-segment continuous interpolation, please write the motion data arrays first. (It will not consume any system resources of MP-8000).

**Category:**

MODBUS sub_function; RTC.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (axis 1). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis (axis 2). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis3*: | The third axis (axis 3). Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *nAcc*: | 0 → constant speed (VV)<br>1 → asymmetric T-curve Acc/Dec (VSV、VV、VA、VD) |
| *VSV*: | Starting speed (in PPS) |
| *VV*: | Vector speed (in PPS) |
| *VA*: | Vector acceleration (PPS/Sec) |
| *VD*: | Vector deceleration (PPS/Sec) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)
Use the ETM_GET_ERROR_CODE() function to identify the error.

**MODBUS example:**

```
ETM_CONTINUE_INTP (h, 1, 1, 2, 4, 1, 300, 18000, 500000, 200);
//execute X, Y, Z 3-axis multi-segment continuous interpolation
// on module 1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 21 | 01 | 10 | 1F 40 | 00 0D | 1A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 86 | *Sub_function code* |
| 1 | 00 01 | *axis1* |
| 2 | 00 02 | *axis2* |
| 3 | 00 04 | *axis3* |
| 4 | 00 01 | *nAcc* |
| 5 | 00 00 | *MSW of VSV* |
| 6 | 01 2C | *LSW of VSV (300 = 0x12C)* |
| 7 | 00 00 | *MSW of VV* |
| 8 | 46 50 | *LSW of VV (18000 = 0x4650)* |
| 9 | 00 07 | *MSW of VA* |
| 10 | A1 20 | *LSW of VA (500000 = 0x7A120)* |
| 11 | 00 00 | *MSW of VD* |
| 12 | 00 64 | *LSW of VD (200 = 0xC8)* |

**Special Description:**

**This similar motion can be done by continuously using ETM_MACRO_MIXED_2D_INITIAL( ) and ETM_MACRO_MIXED_2D_CONTINUE( ) inside an MP.**

**void i8094H_CONTINUE_INTP_ARRAY(BYTE *cardNo*, BYTE *nType[ ]*, long *cp1[ ]*, long *cp2[ ]*, long *fp1[ ]*, long *fp2[ ]*, long *fp3[ ]*)**

**Description:**

**Multi-segment continuous interpolation motion data arrays.**

**Category:**

**MODBUS table ; Internal function.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *nType[ ]*: | Maximum segment: 100(0 ~ 99). It contains the interpolation commands defined as follows.<br><br>1 →i8094H_LINE_2D(BYTE *cardNo*, long *fp1*, long *fp2*);<br><br>2 →i8094H_ARC_CW(BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*);<br><br>3 →i8094H_ARC_CCW(BYTE *cardNo*, long *cp1*, long *cp2*, long *fp1*, long *fp2*);<br><br>4 →i8094H_CIRCLE_CW(BYTE *cardNo*, long *cp1*, long *cp2*);<br><br>5 →i8094H_CIRCLE_CCW(BYTE *cardNo*, long *cp1*, long *cp2*);<br><br>6 →i8094H_LINE_3D(BYTE *cardNo*, long *fp1*, long *fp2*, long *fp3*);<br><br>7 → It indicates the end of continuous interpolation. |
| *cp1[ ]*: | It contains a list of segment center point data at axis 1. (-2,000,000,000 ~ +2,000,000,000) |
| *cp2[ ]*: | It contains a list of segment center point data at axis 2. (-2,000,000,000 ~ +2,000,000,000) |
| *fp1[ ]*: | This array contains a list of segment end point data at axis 1. (-2,000,000,000 ~ +2,000,000,000) |
| *fp2[ ]*: | This array contains a list of segment end point data at axis 2. (-2,000,000,000 ~ +2,000,000,000) |
| *fp3[ ]*: | This array contains a list of segment end point data at axis 3. (-2,000,000,000 ~ +2,000,000,000)<br>(Note: The 2-axis and 3-axis motion commands can not be mixed together when applying commands. Please fill 0 for the cell values in the array if these cells are not used.) |

**Return:**

**None**

**Special Description:**

**This function only allow to be used on MP-8000 or the controller which direct control the i-8094H; it can not be used in ET-M8194H. This similar motion can be done by continuously using ETM_MACRO_MIXED_2D_INITIAL() and ETM_MACRO_MIXED_2D_CONTINUE( ) inside an MP.**

### 6.8.6 3-Axis Helical Motion

eRET ETM_HELIX_3D (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*, BYTE nDir, DWORD *VV* , long *cp1*, long *cp2*, long *cycle*, long *pitch*)

※ eRET ETM_MACRO_HELIX_3D (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*, BYTE *nDir*, DWORD *VV* , long *cp1*, long *cp2*, long *cycle*, long *pitch*)

**Description:**

This function performs a 3-axis helical motion. However, it is a software macro-function; therefore, it requires CPU resource to run this function. (But It will not consume any system resources of MP-8000).

**Category:**

MODBUS sub_function; RTC and MP.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (axis 1). Please refer to Table 2. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis (axis 2). Please refer to Table 2. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis3*: | The third axis (axis 3). Please refer to Table 2. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *nDir*: | 0 → Move in a CW direction.<br>1 → Move in a CCW direction. |
| *VV*: | Vector speed (in PPS) |
| *cp1*: | The value of center at axis 1 (-2,000,000,000 ~ +2,000,000,000) |
| *cp2*: | The value of center at axis 2 (-2,000,000,000 ~ +2,000,000,000) |
| *cycle*: | Number of cycle (-2,000,000,000 ~ +2,000,000,000) |
| *pitch*: | Pitch per revolution (the advanced distance for each revolution) (-2,000,000,000 ~ +2,000,000,000) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)
Use the ETM_GET_ERROR_CODE() function to identify the error.

**Remark:**

The Sub_function code of ETM_HELIX_3D is 0A 88.
The Sub_function code of ETM_MACRO_HELIX_3D is 0C 88.

**MODBUS example:**

```
ETM_HELIX_3D (h, 1, AXIS_Y, AXIS_Z, AXIS_U, 1, 50000, 0, 25000, 50, 3600);
//the circular motion is on YZ plane, and the linear motion is along the U axis.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 25 | 01 | 10 | 1F 40 | 00 0F | 1E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 88/0C 88 | *Sub_funciton code* |
| 1 | 00 02 | *axis1 (2 ➔ AXIS_Y)* |
| 2 | 00 04 | *axis2 (4 ➔ AXIS_Z)* |
| 3 | 00 08 | *axis3 (8 ➔ AXIS_U)* |
| 4 | 00 01 | *nDir* |
| 5 | 00 00 | *MSW of VV* |
| 6 | C3 50 | *LSW of VV (50000 = 0xC350)* |
| 7 | 00 00 | *MSW of cp1* |
| 8 | 00 00 | *LSW of cp1 (0 = 0x0)* |
| 9 | 00 00 | *MSW of cp2* |
| 10 | 61 A8 | *LSW of cp2 (25000 = 0x61A8)* |
| 11 | 00 00 | *MSW of cycle* |
| 12 | 00 32 | *LSW of cycle (50 ➔ 0x32)* |
| 13 | 00 00 | *MSW of pitch* |
| 14 | 0E 10 | *LSW of pitch (3600 ➔ 0xE10)* |

**Related example :**

```
BYTE cardNo=1; //select module 1.

//================================================
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU,160000L);
//set maximum speed for all axes to 160K PPS.
long v=50000; //set vector speed to 50K PPS.
ETM_HELIX_3D(h, cardNo, AXIS_Y, AXIS_Z, AXIS_X, 1, v, 0, 1000, 5, -2000);
//the circular motion is on YZ plane, and the linear motion is
//along the X axis.

//================================================
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU,160000L);
//set the maximum    speed for all axes to 160K PPS.
long v=100000; //set vector speed to 100K PPS.
ETM_HELIX_3D(h, cardNo, AXIS_Y, AXIS_Z, AXIS_U, 1, v, 0, 25000, 50, 3600);
//the circular motion is on YZ plane, and the linear motion is along
//the U axis.
```

### 6.8.7 2-Axis Ration Motion

**eRET ETM_RATIO_INITIAL (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, DWORD *SV* , DWORD *V* , DWORD *A*, BYTE *CMX*, BYTE *CDV*)**

**※ eRET ETM_MACRO_RATIO_INITIAL (HANDLE *h*, BYTE *cardNo*, BYTE *axis1*, BYTE *axis2*, DWORD *SV* , DWORD *V* , DWORD *A*, BYTE *CMX*, BYTE *CDV*)**

**Description:**

This function sets the Initial values for ratio motion (motion in ratio) using a symmetric T-curve speed profile.

**Category:**

MODBUS sub_function; RTC and MP.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis1*: | The first axis (axis 1). Please refer to Table 2. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *axis2*: | The second axis (axis 2). Please refer to Table 2. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |
| *SV*: | Set the value for the starting speed ( in PPS) |
| *V*: | Set the value for the vector speed (in PPS) |
| *A*: | Set the acceleration value (in PPS/sec) |
| *CMX*: | Set the molecule value between the two assigned axes(1 ~+127) |
| *CDV*: | Set the denominator value between the two assigned axes(1 ~ +127) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**Remark:**

The Sub_function code of ETM_RATIO_INITIAL is 0A 8B.
The Sub_function code of ETM_MACRO_RATIO_INITIAL is 0C 8B.

**MODBUS example:**

```
ETM_RATIO_INITIAL (h, 1, AXIS_U, AXIS_X, 300, 18000, 500000, 9, 25);
//Initial setting forETM_RATIO_2D(…) function.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 1D | 01 | 10 | 1F 40 | 00 0B | 16 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 8B/0C 8B | *Sub_funciton code* |
| 1 | 00 08 | *axis1 (2 ➔ AXIS_U)* |
| 2 | 00 01 | *axis2 (4 ➔ AXIS_X)* |
| 3 | 00 00 | *MSW of SV* |
| 4 | 01 2C | *LSW of SV (300 = 0x12C)* |
| 5 | 00 00 | *MSW of V* |
| 6 | 46 50 | *LSW of V (18000 = 0x4650)* |
| 7 | 00 07 | *MSW of A* |
| 8 | A1 20 | *LSW of A (500000 = 0x7A120)* |
| 9 | 00 09 | *CMX* |
| 10 | 00 19 | *CDV (25 = 0x19)* |

**eRET ETM_RATIO_2D (HANDLE *h*, BYTE *cardNo*, BYTE *nType*, long *data*, BYTE *nDir*)**

**※ eRET ETM_MACRO_RATIO_2D (HANDLE *h*, BYTE *cardNo*, BYTE *nType*, long *data*, BYTE *nDir*)**

**Description:**

**This function performs a two-axis ratio motion. (It will not consume any system resources of MP-8000).**

**Category:**

**MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***nType*:** | **0 → Perform the ratio motion.<br>1 → Declare the end of ratio motion.** |
| ***data*:** | **The pulse number of axis1.<br>(-2,000,000,000 ~ +2,000,000,000)** |
| ***nDir*:** | **Direction of the second axis.<br>0: CW; 1: CCW** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**
**Use the ETM_GET_ERROR_CODE() function to identify the error.**

**Remark:**

**The Sub_function code of ETM_RATIO_2D is 0A 8D.**
**The Sub_function code of ETM_MACRO_RATIO_2D is 0C 8D.**

**MODBUS example:**

```
ETM_RATIO_2D (h, 1, 0, 3600, 0);
//perform the ratio motion in the CW direction.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 8D/0C 8D | *Sub_funciton code* |
| 1 | 00 00 | *nType* |
| 2 | 00 00 | *MSW of data* |
| 3 | 0E 10 | *LSW of data (3600 = 0xE10)* |
| 4 | 00 00 | *nDir* |

**Related example:**

```
BYTE cardNo=1; //select module 1.
unsigned short sv=300; //set starting speed to 300 PPS.
unsigned short v=18000; //set vector speed to 18000 PPS.
unsigned long a=500000; //set acceleration value to 500K PPS/Sec.
unsigned short loop1;
unsigned short loop2;
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU,160000L);
//set maximum speed value to 160000 PPS.
ETM_RATIO_INITIAL(h, cardNo, AXIS_U, AXIS_X, sv, v, a, 9, 25);
//assign U axis as the axis 1 and X axis as the axis 2.
for (loop2 = 0; loop2 < 5; loop2++)
{
        for (loop1 = 0; loop1 < 5; loop1++)
        {
                ETM_RATIO_2D(h, cardNo, 0, 3600, 0);
                //perform the ratio motion in the CW direction.
                ETM_RATIO_2D(h, cardNo, 0, 3600, 1);
                //perform the ratio motion in the CCW direction.
        }
        ETM_RATIO_2D(h, cardNo, 0, 7200, 0);
        ETM_RATIO_2D(h, cardNo, 0, 3600, 1);
}
ETM_RATIO_2D(h, cardNo, 1, 7200, 0);
//End the ratio motion.
```

### 6.8.8 Synchronous Line Scan Motion

**eRET ETM_LINE_SCAN (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *Type*, BYTE *outEdge*, BYTE *PulseWidth*, long *Pitch*)**

**Description:**

**Equal distance Line Scan trigger out: Max speed < 100KHz (Pulse Width 10uS). Unequal distance Line Scan trigger out : Max speed < 18KHz.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis number: equal distance X or Y (1,2) unequal distance X(1). |
| *Type*: | 0 = equal distance motion (please refer to C+，LP)<br>1 = unequal distance motion (please refer to C+，LP)<br>2 = equal distance motion (please refer to C+，EP)<br>3 = unequal distance motion (please refer to C+，EP) |
| *outEdge*: | Output active logic:<br>0 = low active (0V); 1 = high active (5V) |
| *PulseWidth*: | Output pulse width<br>0 = 10 uSec<br>1 = 20 uSec<br>2 = 100 uSec<br>3 = 200 uSec<br>4 = 1,000 uSec<br>5 = 2,000 uSec<br>6 = 10,000 uSec<br>7 = 20,000 uSec |
| *pitch*: | Specify interval pulses<br>(-2,000,000,000 ~ +2,000,000,000) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_LINE_SCAN (h, 1, AXIS_X, 0, 0, 0, -39);
//Sets the equal distance Line Scan trigger on axis-X.
//One trigger will be sent out every 40 (=39+1) pulses.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 15 | 01 | 10 | 1F 40 | 00 07 | 0E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 90 | *Sub_funciton code* |
| 1 | 00 01 | *axis (1 ➔ AXIS_X)* |
| 2 | 00 00 | *Type (even distance trigger)* |
| 3 | 00 00 | *outEdge (high level pulse)* |
| 4 | 00 00 | *PulseWidth (0➔ 10 uSec)* |
| 5 | FF FF | *MSW of Pitch* |
| 6 | FF D9 | *LSW of Pitch (-39 = 0xFFFFFFD9)* |

**Related example:**

```
ETM_SET_MAX_V(h, CardNo, AXIS_XY, 4000000);
//set the maximum speed of X and Y axes to 4M PPS.
ETM_NORMAL_SPEED(h, CardNo, AXIS_XY, 0);
//set the driving mode to be symmetric T-curve.
ETM_SET_V(h, CardNo, AXIS_XY, 2000000);
//set the speed of X and Y axes to 2M PPS.
ETM_SET_A(h, CardNo, AXIS_XY, 20000000);
//set the acceleration of X and Y axes to 2M PPS/S.
ETM_SET_SV(h, CardNo, AXIS_XY, 2000000);
//set the starting speed of X and Y axes to 2M PPS.
ETM_LINE_SCAN(h, CardNo, AXIS_X, 0, 0, 0, -39);
//Sets the equal distance Line Scan trigger on axis-X.
//One trigger will be sent out every 40 (=39+1) pulses.
ETM_LINE_SCAN(h, CardNo, AXIS_Y, 0, 0, 0, -79);
//Sets the equal distance Line Scan trigger on axis-Y.
//One trigger will be sent out every 80 (=79+1) pulses.
ETM_LINE_SCAN_START(h, CardNo, AXIS_XY, 0, -4000000);
//Towards the negative direction move 4000K PPS.
```

**eRET ETM_LINE_SCAN_START (HANDLE *h*, BYTE *cardNo*, BYTE *axis*, BYTE *Type*, long *Position*)**

**Description:**

**Enable Line Scan trigger out motion.**

**When performing unequal distance motion, please note (there is no restriction for equal distance motion):**

**a. All ISR will be stopped.**

**b. Do not execute the following commands:**

**ETM_READ_bVAR**
**ETM_READ_VAR**
**ETM_GET_LP**
**ETM_GET_EP**
**ETM_GET_CV**
**ETM_GET_CA**
**ETM_GET_DI**
**ETM_GET_ERROR**
**ETM_GET_ERROR_CODE**
**ETM_GET_LATCH**
**ETM_STOP_WAIT**
**ETM_CLEAR_STOP**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis number:<br>fixed offset displacement X (1)or Y (2) or XY (3); or<br>unequal offset distance X(1). |
| *Type*: | 0 = equal distance motion (according to LP value)<br>1 = unequal distance motion (according to LP value). Need to call ETM_LINE_SCAN_OFFSET2 to set comparing position.<br>2 = equal distance motion (according to EP value)<br>3 = unequal distance motion (according to EP value). Need to call ETM_LINE_SCAN_OFFSET2 to set comparing position. |
| *Position*: | Total pulses, moving distance<br>(-2,000,000,000 ~ +2,000,000,000) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_LINE_SCAN_START (h, 1, AXIS_XY, 0, -4000000);**
**//Towards the negative direction move 4000K PPS.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 91 | *Sub_funciton code* |
| 1 | 00 03 | *axis (3 ➔ AXIS_XY)* |
| 2 | 00 00 | *Type* |
| 3 | FF C2 | *MSW of Position* |
| 4 | F7 00 | *LSW of Position (-4000000 = 0xFFC2F700)* |
| | | |

**eRET ETM_LINE_SCAN_OFFSET2 (HANDLE *h*, BYTE *cardNo*, DWORD Start, WORD Param_Len, char Offset[])**

**Description:**

Set different offset pulses. The array, *Offset[ ]*, contains the offset value for position comparing and trigger output. (the actual trigger position = previous_postion + offset). This function helps to set or modify the elements of *Offset[ ].* The parameter, *Start,* defines the beginning index of offset array that will be changed. The maximum length of the unequal distance interval offsets is 7000.

**Category:**

MODBUS sub_function; RTC.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *Start*: | The starting index in the Offset array for entering offset values (0 ~ 6999). |
| *Param_Len*: | The number of element in *Offset[]*. |
| *Offset[]*: | The distance offset value (-128 ~ +127). |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

```
char Offset[10];

Offset[0] =  1;
Offset[1] =  -2;
Offset[2] =  1;
Offset[3] =  2;
Offset[4] =  -1;
Offset[5] =  1;
Offset[6] =  0;
Offset[7] =  1;
Offset[8] =  0;
Offset[9] =  0;
ETM_LINE_SCAN_OFFSET2 (h, 1, 0, sizeof(Offset), Offset);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 21 | 01 | 10 | 1F 40 | 00 0D | 1A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 93 | *Sub_funciton code* |
| 1 | 00 00 | *MSW 0F Start* |
| 2 | 00 00 | *LSW 0F Start* |
| 3 | 00 01 | *Offset[0]* |
| 4 | 00 FE | *Offset[1] (-2 = 0xFE )* |
| 5 | 00 01 | *Offset[2]* |
| 6 | 00 02 | *Offset[3]* |
| 7 | 00 FF | *Offset[4] (-1 = 0xFF )* |
| 8 | 00 01 | *Offset[5]* |
| 9 | 00 00 | *Offset[6]* |
| 10 | 00 01 | *Offset[7]* |
| 11 | 00 00 | *Offset[8]* |
| 12 | 00 00 | *Offset[9]* |
| | | |

**eRET ETM_GET_LINE_SCAN_DONE (HANDLE *h*, BYTE *cardNo*, BYTE* *LScanState*)**

**Description:**

**After calling ETM_LINE_SCAN_START, the ETM_GET_LINE_SCAN_DONE function will determine if the line scan procedure has finished. The line scan done flag will be reset after calling ETM_LINE_SCAN_START.**

**Category:**

**MODBUS table；RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *LScanState*: | 1: line scan process has been completed<br>0: no interrupt occurred |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
BYTE bState;
ETM_GET_LINE_SCAN_DONE (h, 1, & bState);
```

**MODBUS request:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 06 | 01<br>(Card No.) | 04 | 00 49<br>(73) | 00 01 |

**MODBUS response:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | Byte Count (hex) | Register (hex) |
|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 05 | 01 | 04 | 02 | 00 01<br>line scan operation has ended |

## 6.9 Other Functions

### 6.9.1 Holding the Driving Command

**eRET ETM_DRV_HOLD (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※Δ eRET ETM_MACRO_DRV_HOLD (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This command is usually used when users desire to start multi-axis driving simultaneously. When this command is issued, users may write other driving commands to the control card. All the driving commands will be held after ETM_DRV_HOLD() is issued, and these commands will be started once the ETM_DRV_START() is issued. However, if in driving, this command will not cause the driving to be stopped. But the next command will be held.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or Axes. Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_DRV_HOLD is 0A B4.**
**The Sub_function code of ETM_MACRO_DRV_HOLD is 0C B4.**

**MODBUS example:**

```
ETM_DRV_HOLD (h, 1, AXIS_XYU);
//hold the driving command to XYU
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A B4/0C B4 | *Sub_funciton code* |
| 1 | 00 0B | *axis (B ➔ AXIS_XYU)* |
| | | |

**Related example:**

```
BYTE cardNo=1; //select card 1.
ETM_DRV_HOLD(h, cardNo, AXIS_XYU);
//hold the driving command to XYU
ETM_SET_MAX_V(h, cardNo, AXIS_U, 10000);
//set the maximum speed of U-axis to be 10K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_U, 0);
//set the driving mode to be symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_U, 2000);
//set the speed of U-axis to 2,000 PPS.
ETM_SET_A(h, cardNo, AXIS_U,1000);
//set the acceleration of U-axis to 1,000 PPS/S.
ETM_SET_SV(h, cardNo, AXIS_U, 2000);
//set the starting speed to 2,000 PPS.
ETM_SET_AO(h, cardNo, AXIS_U, 9); // set the AO to 9 Pulses.
ETM_SET_MAX_V(h, cardNo, AXIS_XY, 20000);
//set the maximum speed of X and Y axes to 20K PPS.
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X-axis as the axis 1 and Y-axis as the axis 2 for a 2-axis
// interpolation.
ETM_VECTOR_SPEED(h, cardNo, 0);
//set constant speed motion.  Therefore, VSV=VV. Only VV is required.
ETM_SET_VV(h, cardNo, 5000);
//set the vector speed for card 1 to 5,000  PPS.
ETM_FIXED_MOVE(h, cardNo, AXIS_U, 5000);
//command U-axis to move  5,000 Pulse.  This command is be held.
ETM_LINE_2D(h, cardNo, 12000, 10000);
//command a linear interpolation    motion on the XY planes. It is held, too.
ETM_DRV_START(h, cardNo, AXIS_XYU);
//release the holding status. X,Y , and U axes will start to move
// simultaneously.
```

## 6.9.2 Release the Holding Status and Start the Driving

**eRET ETM_DRV_START (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※Δ eRET ETM_MACRO_DRV_START (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This command releases the holding status, and start the driving of the assigned axes immediately.**

**Category:**

**MODBUS sub_function; RTC, MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or Axes. Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_DRV_START is 0A B5.**
**The Sub_function code of ETM_MACRO_DRV_START is 0C B5.**

**MODBUS example:**

```
ETM_DRV_START (h, 1, AXIS_XYU);
//release the holding status. X,Y , and U axes will start to move
// simultaneously.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A B5/0C B5 | *Sub_funciton code* |
| 1 | 00 0B | *axis (B → AXIS_XYU)* |
| | | |

**Related example:**

```
BYTE cardNo=1; //select card 1.
ETM_DRV_HOLD(h, cardNo, AXIS_XYU);
//hold the driving command to XYU
ETM_SET_MAX_V(h, cardNo, AXIS_U, 10000);
//set the maximum speed of U-axis to be 10K PPS.
ETM_NORMAL_SPEED(h, cardNo, AXIS_U, 0);
//set the driving mode to be symmetric T-curve.
ETM_SET_V(h, cardNo, AXIS_U, 2000);
//set the speed of U-axis to 2,000 PPS.
ETM_SET_A(h, cardNo, AXIS_U,1000);
//set the acceleration of U-axis to 1,000 PPS/S.
ETM_SET_SV(h, cardNo, AXIS_U, 2000);
//set the starting speed to 2,000 PPS.
ETM_SET_AO(h, cardNo, AXIS_U, 9); // set the AO to 9 Pulses.
ETM_SET_MAX_V(h, cardNo, AXIS_XY, 20000);
//set the maximum speed of X and Y axes to 20K PPS.
ETM_AXIS_ASSIGN(h, cardNo, AXIS_X, AXIS_Y, 0);
//set the X-axis as the axis 1 and Y-axis as the axis 2 for a 2-axis
// interpolation.
ETM_VECTOR_SPEED(h, cardNo, 0);
//set constant speed motion.  Therefore, VSV=VV. Only VV is required.
ETM_SET_VV(h, cardNo, 5000);
//set the vector speed for card 1 to 5,000  PPS.
ETM_FIXED_MOVE(h, cardNo, AXIS_U, 5000);
//command U-axis to move  5,000 Pulse.  This command is be held.
ETM_LINE_2D(h, cardNo, 12000, 10000);
//command a linear interpolation    motion on the XY planes. It is held, too.
ETM_DRV_START(h, cardNo, AXIS_XYU);
//release the holding status. X,Y , and U axes will start to move
// simultaneously.
```

### 6.9.3 Stopping the Axes

**eRET ETM_STOP_SLOWLY (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_STOP_SLOWLY (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function commands to decelerate and finally stops the assigned axes slowly.**

**Category:**

**MODBUS sub_function; RTC, and MP**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or Axes. Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_STOP_SLOWLY is 0A B7.**
**The Sub_function code of ETM_MACRO_STOP_SLOWLY is 0C B7.**

**MODBUS example:**

**ETM_STOP_SLOWLY (h, 1, AXIS_XY);**
**//decelerate and stop the X and Y axes**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A B7/0C B7 | *Sub_funciton code* |
| 1 | 00 03 | *axis (3 → AXIS_XY)* |
| | | |

**eRET ETM_STOP_SUDDENLY (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_STOP_SUDDENLY (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**This function commands to immediately stop the assigned axes.**

**Category:**

**MODBUS sub_function; RTC, and MP**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis*: | Axis or Axes. Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_STOP_SUDDENLY is 0A B8.**
**The Sub_function code of ETM_MACRO_STOP_SUDDENLY is 0C B8.**

**MODBUS example:**

**ETM_STOP_SUDDENLY (h, 1, AXIS_ZU);**
**//immediately stop the Z and U axes.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A B8/0C B8 | *Sub_funciton code* |
| 1 | 00 0C | *axis (C → AXIS_ZU)* |
| | | |

**eRET ETM_VSTOP_SLOWLY (HANDLE *h*, BYTE *cardNo*)**

**※ eRET ETM_MACRO_VSTOP_SLOWLY (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function commands to stop interpolation motion of the assigned module in a decelerating way.**

**Category:**

**MODBUS sub_function; RTC, and MP**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_VSTOP_SLOWLY is 0A B9.**
**The Sub_function code of ETM_MACRO_VSTOP_SLOWLY is 0C B9.**

**MODBUS example:**

**ETM_VSTOP_SLOWLY (h, 1);**
**//stop the interpolation of card 1 in a decelerating way.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0A B9/0C B9 | | *Sub_funciton code* | | | |
| | | | | | | | |

**eRET ETM_VSTOP_SUDDENLY (HANDLE *h*, BYTE *cardNo*)**

**※ eRET ETM_MACRO_VSTOP_SUDDENLY (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function commands to stop interpolation motion of the assigned module immediately.**

**Category:**

**MODBUS sub_function; RTC, and MP**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_VSTOP_SUDDENLY is 0A BA.**
**The Sub_function code of ETM_MACRO_VSTOP_SUDDENLY is 0C BA.**

**MODBUS example:**

**ETM_VSTOP_SUDDENLY (h, 1);**
**// stop the interpolation of card 1 immediately.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A BA/0C BA | *Sub_funciton code* |
| | | |

### 6.9.4 Clear the Stop Status

**eRET ETM_CLEAR_STOP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_CLEAR_STOP (HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**After using anyone of the stop functions mentioned in section 6.9.3, please solve the malfunction, and then issue this function to clear the stop status.**

**Category:**

**MODBUS sub_function; RTC, and MP**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***axis:*** | **Axis or Axes. Please refer to Table 2.<br>The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_CLEAR_STOP is 0A BB.**
**The Sub_function code of ETM_MACRO_CLEAR_STOP is 0C BB.**

**MODBUS example:**

```
ETM_CLEAR_STOP (h, 1, AXIS_ZU);
//clear the error status of zu axes.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A BB/0C BB | *Sub_funciton code* |
| 1 | 00 0C | *axis (C → AXIS_ZU)* |
| | | |

**eRET ETM_CLEAR_VSTOP (HANDLE *h*, BYTE *cardNo*)**

**※ eRET ETM_MACRO_CLEAR_VSTOP (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**After using anyone of the stop functions mentioned in section 6.9.3, please solve the malfunction, and then issue this function to clear the stop status.**

**Category:**

**MODBUS sub_function; RTC, and MP**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_CLEAR_VSTOP is 0A 09.**
**The Sub_function code of ETM_MACRO_CLEAR_VSTOP is 0C 09.**

**MODBUS example:**

**ETM_CLEAR_VSTOP (h, 1);**
**//clear the error status of card 1.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0A 09/0C 09 | | *Sub_funciton code* | | | |
| | | | | | | | |

### 6.9.5 End of Interpolation

**eRET ETM_INTP_END (HANDLE *h*, BYTE *cardNo*, BYTE *type*)**

**※ eRET ETM_MACRO_INTP_END (HANDLE *h*, BYTE *cardNo*, BYTE *type*)**

**Description:**

- **If the current motion status is running an interpolation motion and you would like to issue a single axis motion or change the coordinate definition, you should call this function before the new command is issued.**

- **You can redefine the MAX_V for each axis. In this way, you do not have to execute ETM_INTP_END() function.**

**Category:**

**MODBUS sub_function; RTC, and MP**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *type*: | 0 → 2-axis interpolation<br>1 → 3-axis interpolation |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_INTP_END is 0A BC.**
**The Sub_function code of ETM_MACRO_INTP_END is 0C BC.**

**MODBUS example:**

```
ETM_INTP_END (h, 1, 0);
//declear the end of a 2-axis interpolation on card        1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A BC/0C BC | *Sub_funciton code* |
| 1 | 00 00 | *type (0 → 2-axis interpolation)* |

### 6.9.6 Emergency Stop

**eRET ETM_EMERGENCY_STOP (HANDLE *h*, BYTE *cardNo*, BYTE *stopType*)**

**Description:**

**Stop all the axes and clear all data in the i-8094H command buffer.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***stopType*:** | **Stop type:**<br>**0: immediately stop; 1: deceleration stop** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_EMERGENCY_STOP (h, 1, 0);**

**// Stop all the axes immediately and clear all data in the i-8094H command //buffer.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 04 | *Sub_function code* |
| 1 | 00 00 | *stopType* |
| | | |

### 6.9.7 Clear the Emergency Stop Status

**eRET ETM_CLEAR_EMERGENCY_STOP (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**After using anyone of the stop functions mentioned in section 6.9.6, please solve the malfunction, and then issue this function to clear the stop status.**

**Category:**

**MODBUS sub_function; RTC.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
ETM_CLEAR_EMERGENCY_STOP (h, 1);
//clear the error status of card 1.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 05 | *Sub_function code* |
| | | |

# 7 Initial Parameter Table

**The initial table stores the basic settings of the motion control chip in a non-volatile memory. The initialization table includes the following setting (Figure 3):**

- **pulse output signal**
- **encoder setting**
- **hardware trigger level**
- **software limits**

| Function | Parameter | X-Axis | Y-Axis | Z-Axis | U-Axis |
|---|---|---|---|---|---|
| Pulse Output Signal | Pulse Output Mode | 0 | 0 | 0 | 0 |
| Max Pulse Output Rate | Data (8000 to 4,000,000 PPS) | 8000 | 8000 | 8000 | 8000 |
| Hardware Limit Switch (HLMT) | Active Level (forward) | Low Active | Low Active | Low Active | Low Active |
| | Active Level (reverse) | Low Active | Low Active | Low Active | Low Active |
| Hardware Limit Stop Mode | Stop Mode | Abrupt Stop | Abrupt Stop | Abrupt Stop | Abrupt Stop |
| Near Home Sensor | Trigger Level | Low Active | Low Active | Low Active | Low Active |
| Home Sensor | Trigger Level | Low Active | Low Active | Low Active | Low Active |
| Software Limit | Enable Software Limit | Disable | Disable | Disable | Disable |
| | Software Limit (forward) | 100000 | 100000 | 100000 | 100000 |
| | Software Limit (reverse) | -100000 | -100000 | -100000 | -100000 |
| | Position Counter Type | Logic Pos | Logic Pos | Logic Pos | Logic Pos |
| Set Encoder Parameters | Encoder Input Type | A Quad B | A Quad B | A Quad B | A Quad B |
| | A Quad B Input Signal Division | 1/1 | 1/1 | 1/1 | 1/1 |
| | Trigger Level for Z Phase | Low Active | Low Active | Low Active | Low Active |
| Servo Driver Setting | On/Off | Off | Off | Off | Off |
| Servo Alarm Setting | Enable Servo Alarm | Disable | Disable | Disable | Disable |
| | Trigger Level | Low Active | Low Active | Low Active | Low Active |
| In-Position Signal | Enable In-Position Input | Disable | Disable | Disable | Disable |
| | Trigger Level | Low Active | Low Active | Low Active | Low Active |
| Digital Filter | Input Ports | 1 | 1 | 1 | 1 |
| | Filter Time Constant | 2 | 2 | 2 | 2 |
| Variable Ring Position Counter | Enable Variable Ring Counter | Disable | Disable | Disable | Disable |
| | Maximum Value | 10000 | 10000 | 10000 | 10000 |
| Triangle Driving Profile Prevention | Enable Triangle Prevention | Disable | Disable | Disable | Disable |
| | | | | | |

**Figure 3: Initial table**

**The factory default settings of the initial table is as follows:**

```
ETM_SET_PULSE_MODE(h, cardNo, AXIS_XYZU, 0);
ETM_SET_MAX_V(h, cardNo, AXIS_XYZU, 2000000L);
ETM_SET_HLMT(h, cardNo, AXIS_XYZU, 0, 0);
ETM_LIMITSTOP_MODE(h, cardNo, AXIS_XYZU, 0);
ETM_SET_NHOME(h, cardNo, AXIS_XYZU, 0);
```

**ETM_SET_HOME_EDGE(h, *cardNo*, AXIS_XYZU, 0);**
**ETM_CLEAR_SLMT(h, *cardNo*, AXIS_XYZU);**
**ETM_SET_ENCODER(h, *cardNo*, AXIS_XYZU, 0, 0, 0);**
**ETM_SERVO_OFF(h, *cardNo*, AXIS_XYZU);**
**ETM_SET_ALARM(h, *cardNo*, AXIS_XYZU, 0, 0);**
**ETM_SET_INPOS(h, *cardNo*, AXIS_XYZU, 0, 0);**
**ETM_SET_FILTER(h, *cardNo*, AXIS_XYZU, 0, 0);**
**ETM_VRING_DISABLE(h, *cardNo*, AXIS_XYZU);**
**ETM_AVTRI_DISABLE(h, *cardNo*, AXIS_XYZU);**
**ETM_EXD_DISABLE(h, *cardNo*, AXIS_XYZU);**

**Modify the default initial table by calling the above function with the required settings. The settings may also be changed after start up or later on during the control operations. Always the last setting will be retained. Set the initialization table according to the specific environment in which the ET-M8194H is being used by calling the above functions.**

## 7.1 Calling the Initial Table

**eRET ETM_LOAD_INITIAL (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**In order to initialize the basic hardware and software settings of the ET-M8194, it is required to call the ETM_LOAD_INITIAL function. Directly after power on, the initial table is being automatically called by executing the ETM_LOAD_INITIAL function. The Modbus master is also allowed to call this function at any time.**

**Category:**

**MODBUS sub_function; RTC**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_LOAD_INITIAL (h, 1);**

**// load the initial setting values of the parameter table into i8094H**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0A C8 | | *Sub_funciton code* | | | |
| | | | | | | | |

**Related example:**

```
ETM_SET_PULSE_MODE(h, 1, INITIAL_XYZU, 0);
//set the pulse mode of X, Y, Z, and U axes as 0, write into the parameter
//table in module 1.
ETM_SET_MAX_V(h, 1, INITIAL_XY, 200000L);
//The maximum speed for the X and Y axes of module 1 is 200KPPS.
//Write into the parameter table.
ETM_SET_HLMT(h, 1, INITIAL_XYZU, 0, 0);
//set all the trigger levels as low-active for all limit switches
//on module 1. Write into the parameter table.
ETM_LIMITSTOP_MODE(h, 1, INITIAL_X, 0);
//set X axis to stop immediately if any limit switch on X axis is triggered
//on module 1. Write into the parameter table.
ETM_SET_NHOME(h, 1, INITIAL_XY, 0);
//set the trigger level of NHOME of X and Y axes on module 1 to be
//active low. Write into the parameter table.
ETM_ LOAD_INITIAL(h, 1);
// load the initial setting values of the parameter table into i8094H
```

## 7.2 Macro Programming

**Macro programs are written to the nonvolatile memory of the i8094H module. The nonvolatile memory can hold Macro program of different sizes. Two types of Macro tables are provided:**

- **MP tables: These tables hold normal motion control Macro commands. MP commands in these tables are being called by the command ETM_MP_CALL or ETM_MACRO_MP_CALL. The ETM_MP_CALL command can be directly called by the host controller, and the ETM_MACRO_MP_CALL is called within a MP table.**
- **ISR tables: ISR in these tables will be called in case of an interrupt generated by the motion control chip. Not all Macro commands can be added to the ISR table in order to guaranty that the execution of ISR is finished in a short time interval.**

**The maximum of command lines provided by each Macro table is shown in the following figure (Figure 4).**

| Macro Type | Number of Macros | Number of Command Lines | Macro Names | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Interrupt Service Routine (ISR) | 6 | 8 | ISR1 | ISR2 | ISR3 | ISR4 | ISR5 | ISR6 | | | |
| | 9 | 16 | ISR7 | ISR8 | ISR9 | ISR10 | ISR11 | ISR12 | ISR13 | ISR14 | ISR15 |
| | 3 | 32 | ISR16 | ISR17 | ISR18 | | | | | | |
| | 2 | 64 | ISR19 | ISR20 | | | | | | | |
| | | | | | | | | | | | |
| Motion Program (MP) | 40 | 8 | MP1 | ~ | MP40 | | | | | | |
| | 50 | 16 | MP41 | ~ | MP90 | | | | | | |
| | 40 | 32 | MP91 | ~ | MP130 | | | | | | |
| | 20 | 64 | MP131 | ~ | MP150 | | | | | | |
| | 5 | 128 | MP151 | MP152 | MP153 | MP154 | MP155 | | | | |
| | 2 | 512 | MP156 | MP157 | | | | | | | |

**Figure 4: Macro table definitions**

### 7.2.1 Create and Close MP Macro Program Codes

**eRET ETM_MP_CREATE(HANDLE *h*, BYTE *cardNo*, BYTE *mp_No*)**

**Description**:

**Each Macro program MP has to start with an ETM_MP_CREATE command and the end of the MP has to be indicated with an ETM_MACRO_MP_CLOSE command.**

**Category:**

**MODBUS sub_function; RTC**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***mpNo:*** | **The following variables are supported:<br>Macro Program number (MP1 ~ MP157)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MP_CREATE (h, 1, MP21);**
**//Write Macro Program into i8094H.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A C9 | *Sub_funciton code* |
| 1 | 00 15 | *mpNo (MP21 = 21 = 0x15)* |

**Related example:**

```
ETM_MP_CREATE(h, 1, MP21); //Write #21 Macro Program into i8094H.
//=========================================================
//The following functions will not be executed, but will be written into
//i8094H for further execution.
ETM_MACRO_SET_MAX_V(h, 1, AXIS_XYZU, 20000);
//The maximum speed of all axes is 20K PPS.
ETM_MACRO_NORMAL_SPEED(h, 1, AXIS_XYZU, 0);
//Set symmetric T-curve for XYZU axes on module.
ETM_MACRO_SET_V(h, 1, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
ETM_MACRO_SET_A(h, 1, AXIS_XYZU, 1000);
//set the acceleration of XYZU axes to be 1000 PPS/sec
ETM_MACRO_SET_SV(h, 1, AXIS_XYZU, 2000);
//set the start speed of XYZU axes to be 2000 PPS.
ETM_MACRO_SET_AO(h ,1, AXIS_XYZU, 9);
//set the number of remaining offset pulses for XYZU axes to be 9 pulses.
ETM_MACRO_FIXED_MOVE(h, 1, AXIS_XYZU, 10000);
//move XYZU axes to be 10000 pulses.
ETM_MACRO_MP_CLOSE(h, 1);
// module 1 finish, write Macro Program    into i8094H
//=========================================================
```

## ※ eRET ETM_MACRO_MP_CLOSE (HANDLE *h*, BYTE *cardNo*)

**Description:**

Indicate the end of a MP Macro program. This command is only used together with ETM_MP_CREATE to define an MP program. Otherwise, it will be ignored.

**Category:**

MODBUS sub_function; MP

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

ETM_MACRO_MP_CLOSE (h, 1);
// module 1 finish, write Macro Program into i8094H

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 0C CA | | *Sub_funciton code* | | | |
| | | | | | | | |

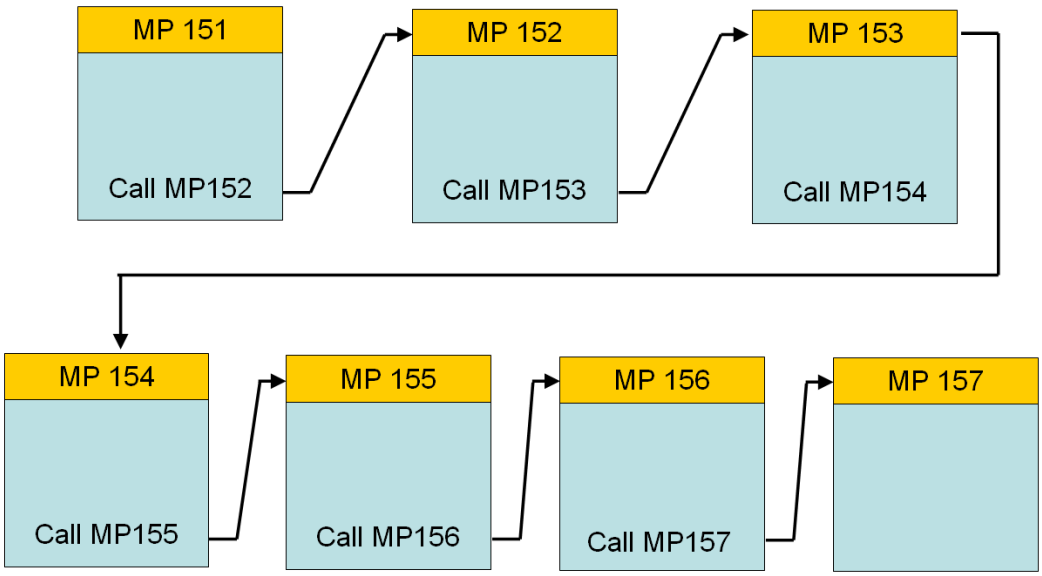
### 7.2.2 Execute MP Macro program

**eRET ETM_MP_CALL (HANDLE *h*, BYTE *cardNo*, BYTE *mpNo*)**

**※ eRET ETM_MACRO_MP_CALL (HANDLE *h*, BYTE *cardNo*, BYTE *mpNo*)**

**Description:**

**Calls and execute a specific Macro Program stored in the non-volatile memory of the i8094H. Within an MP program, up to 6 nested MP_CALL function calls are allowed. For MP, the maximum layers could be up to 7. For ISR, only one layer.**




**Category:**

**MODBUS table, MODBUS sub_function; RTC and MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***mpNo:*** | **The following variables are supported:<br>Macro Program number (MP1 ~ MP157)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_MP_CALL is 0A CB.**
**The Sub_function code of ETM_MACRO_MP_CALL is 0C CB.**

**MODBUS example:**

```
//The true value of MPn = n; therefore, MP21 = 21 = 0x15.
ETM_MP_CALL (h, 1, MP21); // Execute the MP21 on i8094H
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 0A CB/0C CB | | *Sub_funciton code* | | | |
| 1 | | 00 15 | | *mpNo (MP21 = 21 = 0x15)* | | | |

**The other method to call MPn is writing the MPn to the pre-defined holding register 8.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 00 08 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 15 | | *mpNo (MP21 = 21 = 0x15)* | | | |
| | | | | | | | |

**Related example:**

```
ETM_MP_CREATE(h, 1, MP21); //Write #21 Macro Program into i8094H.
//=========================================================
//The following functions will not be executed, but will be written into
//i8094H for further execution.
ETM_MACRO_SET_MAX_V(h, 1, AXIS_XYZU, 20000);
//The maximum speedof the axis is 20K PPS.
ETM_MACRO_NORMAL_SPEED(h, 1, AXIS_XYZU, 0);
//Set symmetric T-curve for XYZU axes on module.
ETM_MACRO_SET_V(h, 1, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
ETM_MACRO_SET_A(h, 1, AXIS_XYZU, 1000);
//set the acceleration of XYZU axes to be 1000 PPS/sec.
ETM_MACRO_SET_SV(h, 1, AXIS_XYZU, 2000);
//set the start speed of XYZU axes to be 2000 PPS.
ETM_MACRO_SET_AO(h, 1, AXIS_XYZU, 9);
//set the number of remaining offset pulses for XYZU axes to be 9 pulses.
```

```
ETM_MACRO_FIXED_MOVE(h, 1, AXIS_XYZU, 10000);
//move XYZU axes to be 10000 pulses.
ETM_MACRO_MP_CLOSE(h, 1);
// module 1 finish, write Macro Program into i8094H
//=========================================================
ETM_MP_CALL(h, 1, MP21); // Execute the MP21 on i8094H
```

## 7.3 Create ISR Macro Program Codes

### 7.3.1 Start of ISR Macro program

**eRET ETM_MP_ISR_CREATE(HANDLE *h*, BYTE *cardNo*, BYTE *isr_No*)**

**Description:**

**ISR Macro table are being called by hardware interrupts generated by the motion control chip (e.g. exceeding harware or software limit, driving has finished or when a compare condition has been met). To ensure that the API in an ISR are executed completely in a short time, no nested Macro calls, MP_FOR loop and MP_STOP_WAIT are allowed inside an ISR.**

**Each ISR program has to be created by calling an ETM_MP_ISR_CREATE command and ended by an ETM_MACRO_MP_ISR_CLOSE command.**

**Category:**

**MODBUS sub_function; RTC**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *isr_No:* | The following variables are supported:<br>ISR Macro Program number (ISR1 ~ ISR20) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MP_ISR_CREATE (h, 1, ISR1);**
**//Write ISR Macro program into i8094H.**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0ACD | *Sub_funciton code* |
| 1 | 00 01 | *isr_No (ISR1 = 1 = 0x01)* |

### 7.3.2 End of ISR Macro program

**Δ eRET ETM_MACRO_MP_ISR_CLOSE (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**Indicate the end of a ISR program definition. This command is only valid after ETM_MP_ISR_CREATE is executed beforehand. Otherwise, it will be ignored.**

**Category:**

**MODBUS sub_function; ISR**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_MP_ISR_CLOSE (h, 1);**
**// finish the writing of ISR Macro Program for i8094H in slot 1**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 0C CE | | *Sub_funciton code* | | | |
| | | | | | | | |

### 7.3.3 Execute ISR Maro Program (ISR)

**eRET ETM_MP_ISR_CALL (HANDLE *h*, BYTE *cardNo*, BYTE *isrNo*)**

**Description:**

**This function is used for calling an ISR Macro program. Usually, ISR Macro tables are called by hardware interrupts. Use this function can call an ISR directly. No nested ISR calls are allowed.**

**Category:**

**MODBUS table, MODBUS sub_function; ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *isrNo*: | The following variables are supported:<br>ISR Macro Program number (ISR1 ~ ISR20) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
//The true value of ISRn = n; therefore, ISR1 = 1 = 0x1.
ETM_MP_ISR_CALL (h, 1, ISR1);   // Execute the ISR1
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A CF | *Sub_funciton code* |
| 1 | 00 01 | *isrNo (ISR1 = 1 = 0x1)* |

## 7.4 User Defined Variables

### 7.4.1 Assign Macro variable a value

※Δ eRET ETM_MACRO_SET_VAR (HANDLE *h*, BYTE *cardNo*, long *varNo*, long *data*)

**Description:**

Assign a Macro variable VARn (n = 0,1 , 2, ...) an integer value. It is also possible to assign the variable a value from another variable. This API provides functions similar to the following statments:
VARn = VARm
VARn = Value

**Category:**

MODBUS table, MODBUS sub_function; MP and ISR.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *varNo:* | The following variables are supported:<br>MACRO variable (VAR0 ~ VAR511) |
| *data:* | The following inputs are supported:<br>Value (-2,000,000,000 ~ +2,000,000,000) or<br>MACRO variable (VAR0 ~ VAR511) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

```
//The address of VARn = 0x7FFF0000 + n; therefore, the address of VAR1 is
// 0x7FFF0001.
ETM_MACRO_SET_VAR(h, 1, VAR1, 100); // VAR1 = 100
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 11 | 01 | 10 | 1F 40 | 00 05 | 0A |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C D2 | *Sub_funciton code* |
| 1 | 7F FF | *MSW address of varNo* |
| 2 | 00 01 | *LSW address of varNo* |
| 3 | 00 00 | *MSW of data* |
| 4 | 00 64 | *LSW of data (100 = 0x64)* |

### 7.4.2 Get command return value

**※Δ eRET ETM_MACRO_SET_RVAR (HANDLE *h*, BYTE *cardNo*, long *varNo*)**

**Description:**

**Assign the return value of the preceding function to a Macro variable VARn (n = 0, 1, 2, ...),**
**where VARn = return value of preceding function (the value will be automatically casted to (long).**

**The command reads the return values of the following commands:**

- **ETM_MACRO_GET_LP**
- **ETM_ MACRO_GET_EP**
- **ETM_ MACRO_GET_DI**
- **ETM_ MACRO_GET_ERROR**
- **ETM_ MACRO_GET_ERROR_CODE**
- **ETM_ MACRO_FRNET_IN**
- **ETM_ MACRO_GET_LATCH**
- **ETM_ MACRO_GET_DI_ALL**

**Category:**

**MODBUS sub_function; MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *varNo:* | The following variables are supported:<br>MACRO variable (VAR0 ~ VAR511) |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

```
// The address of VARn = 0x7FFF0000 + n; therefore, the address of VAR5
// is 0x7FFF0005.
ETM_MACRO_SET_RVAR(h, 1, VAR5);
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C D3 | *Sub_funciton code* |
| 1 | 7F FF | *MSW address of varNo* |
| 2 | 00 05 | *LSW address of varNo* |
| | | |

**Related example:**

```
//The following commands show how to use Macro commands to store the
//logic position in a variable.
ETM_MP_CREATE(h, 1, MP1); //Write #1 Macro Program into i8094H.
ETM_MACRO_GET_LP(h, 1, AXIS_X);
//Reads the LP value of the X axis on module 1.
ETM_MACRO_SET_RVAR(h, 1, VAR5);
//Assign the return value of ETM_MACRO_GET_LP to VAR5
//( VAR5 = LP value)
ETM_MACRO_MP_CLOSE(h, 1);
// module 1 finish, write Macro Program   into i8094H
```

## 7.5 Simple calculations

**※Δ eRET ETM_MACRO_VAR_CALCULATE (HANDLE *h*, BYTE *cardNo*, long *varNo*, BYTE *Operator*, long *varNo1*, long *varNo2*)**

**Description:**

**Basic two parameter (value) calculations:**

| | | | |
|---|---|---|---|
| ***varNo* +** | ***varNo1*** | **=** | ***varNo2*** |
| ***varNo* -** | ***varNo1*** | **=** | ***varNo2*** |
| ***varNo* *** | ***varNo1*** | **=** | ***varNo2*** |
| ***varNo* /** | ***varNo1*** | **=** | ***varNo2*** |
| ***varNo*&** | ***varNo1*** | **=** | ***varNo2*** |
| ***varNo* \|** | ***varNo1*** | **=** | ***varNo2*** |

**Category:**

**MODBUS sub_function; MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***varNo*:** | **The first operand for this calculation. The following operand inputs are supported:<br>Value (-2,000,000,000 ~ +2,000,000,000) or<br>MACRO variable (VAR0 ~ VAR511)** |
| ***Operator*:** | **'+' addition<br>'−' subtraction<br>'∗' multiplication<br>'/' division(rounding down to the nearest integer)<br>'&' AND operator<br>'\|' OR operator** |
| ***varNo1*:** | **The second operand of the calculation. The following operand inputs are supported:<br>Value (-2,000,000,000 ~ +2,000,000,000) or<br>MACRO variable (VAR0 ~ VAR511)** |
| ***varNo2*:** | **MACRO variable (VAR0 ~ VAR511) which stores the result of the calculation** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**The address of VARn = 0x7FFF0000 + n;**
**For example the address of VAR2 = 0x7FFF0002.**

**The ASCII code of operator:**

| Operator | ASCII (dec) | ASCII (hex) |
|---|---|---|
| '+' | 43 | 2B |
| '-' | 45 | 2D |
| '*' | 42 | 2A |
| '/' | 47 | 2F |
| '&' | 38 | 26 |
| '\|' | 124 | 7C |

```
ETM_MACRO_VAR_CALCULATE (h, 1, VAR1, '+', VAR2, VAR3);
//VAR1 + VAR2 = VAR3
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 17 | 01 | 10 | 1F 40 | 00 08 | 10 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C D4 | *Sub_funciton code* |
| 1 | 7F FF | *MSW address of varNo* |
| 2 | 00 01 | *LSW address of varNo* |
| 3 | 00 2B | *Operator : ASCII code of '+'* |
| 4 | 7F FF | *MSW address of varNo1* |
| 5 | 00 02 | *LSW address of varNo1* |
| 6 | 7F FF | *MSW address of varNo2* |
| 7 | 00 03 | *LSW address of varNo2* |
| | | |

**Related example:**

```
ETM_MP_CREATE(h, 1, MP100); // Write #100 Macro Program into i8094H.
//=========================================================
//The following functions will not be executed, but will be written into
// i8094H for further execution.
ETM_MACRO_SET_VAR(h, 1, VAR1, 100); //VAR1 = 100
ETM_MACRO_SET_VAR (h, 1, VAR2, 200); //VAR2 = 200
ETM_MACRO_SET_VAR (h, 1, VAR10, 10); //VAR10 = 10
//-----------------------------------------------------------------------------------------------
ETM_MACRO_FRNET_IN(h, 1, 8);
ETM_MACRO_SET_RVAR(h, 1, VAR6);
//VAR6 = current input of RA8 on   module 1
//-----------------------------------------------------------------------------------------------
ETM_MACRO_GET_LP(h, 1, AXIS_X);
//Reads the LP value of the X axis on module 1.
ETM_MACRO_SET_RVAR(h, 1, VAR5);
//VAR5 = LP value of the X axis on module 1.
//-----------------------------------------------------------------------------------------------
ETM_MACRO_VAR_CALCULATE(h, 1, VAR1, '+', VAR2, VAR3);
```

```
//VAR1 + VAR2 = VAR3
ETM_MACRO_VAR_CALCULATE (h, 1, VAR3, '-', VAR1, VAR3);
//VAR3 – VAR1 = VAR3
ETM_MACRO_VAR_CALCULATE (h, 1, VAR3, '*', VAR10, VAR2);
//VAR3 x VAR10 = VAR2
ETM_MACRO_VAR_CALCULATE (h, 1, VAR3, '/', VAR2, VAR1);
//VAR3 / VAR2 = VAR1
//Results: VAR1 = 10; VAR2 = 200; VAR3 = 2,000; VAR10 = 10
ETM_MACRO_MP_CLOSE(h, 1);
// module 1 finish, write #100 Macro Program into i8094H
//=========================================================
```

## 7.6 Command Loop (FOR~NEXT)

### 7.6.1 Beginning of FOR loop block

※ **eRET ETM_MACRO_FOR (HANDLE *h*, BYTE *cardNo*, long *varNo*)**

**Description:**

**The *varNo* specifies how many times the code between the ETM_MACRO_FOR command and the ETM_MACRO_NEXT command will be executed. The *varNo* parameter will be decreased until it reaches zero, and then the loop will be stopped. If the *varNo* is assigned to be a Macro VARn, this value of VARn represents the loop cycles and will be changed each time a loop is finished. An ETM_MACRO_EXIT_FOR command can abort the current loop immediately if it is executed inside the loop. Up to six nested loops are supported: one outer loop and six inner loops.**

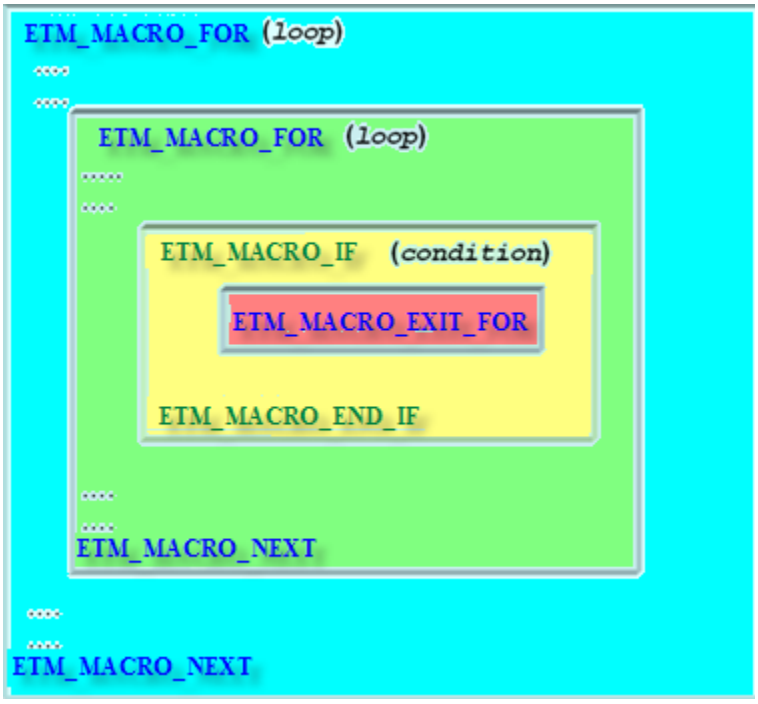



**Figure 5: Basic loop application**

**Category:**

**MODBUS sub_function; MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *lValue:* | Specifies the cycle number. The following inputs are supported:<br>Value (-2,000,000,000 ~ +2,000,000,000) or<br>MACRO variable (VAR0 ~ VAR511) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

// The address of VARn = 0x7FFF0000 + n;
For example, the address of VAR1 = 0x7FFF0001.
ETM_MACRO_FOR (h, 1, VAR1);

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C D5 | *Sub_funciton code* |
| 1 | 7F FF | *MSW address of lValue* |
| 2 | 00 01 | *LSW address of lValue* |
| | | |

**Related example:**

```
ETM_MP_CREATE(h, 1, MP100); // Write #100 Macro Program into i8094H.
//=========================================================
// The following functions will not be executed, but will be written into
// i8094H for further execution.
ETM_MACRO_SET_MAX_V(h, 1, AXIS_X, 20000);
//The maximum speed of the axis X is 20K PPS.
ETM_MACRO_NORMAL_SPEED(h, 1, AXIS_X, 0);
//Set symmetric T-curve for axis X
ETM_MACRO_SET_V(h, 1, AXIS_X, 2000);
//set the speed of axis X to be 2000 PPS.
ETM_MACRO_SET_A(h, 1, AXIS_X, 1000);
//set the acceleration of axis X to be 1000 PPS/Sec
ETM_MACRO_SET_SV(h, 1, AXIS_X, 2000);
//set the start speed of axis X to be 2000 PPS.
ETM_MACRO_SET_AO(h, 1, AXIS_X, 0);
//set the number of remaining offset pulses for axis X to be 0 PPS.
//-------------------------------------------------------------------------------------------
ETM_MACRO_SET_VAR(h, 1, VAR1, 100); //VAR1 = 100。
```

```
ETM_MACRO_FOR(h, 1, VAR1);
//enable axis X to move back and forward 1,000 Pulse, loop for 100 times.
//or input the instant operating value ETM_MACRO_FOR(h, 1, 100).
ETM_MACRO_FIXED_MOVE(h, 1, AXIS_X, 1000);
//move axis X to be 1000 pulses.
ETM_MACRO_FIXED_MOVE(h, 1, AXIS_X, -1000);
//move axis X to be -1000 pulses.
ETM_MACRO_NEXT(h, 1);
ETM_MACRO_MP_CLOSE(h, 1);
// module 1 finish, Macro Program is written into i8094H
//=========================================================
ETM_MP_CALL(h, 1, MP100);
// call module 1 i8094H, execute Macro Program #100.
```

### 7.6.2 End of FOR loop block

※ **eRET ETM_MACRO_NEXT (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function indicates the end of the ETM_MACRO_FOR loop. Every loop has to start with ETM_MACRO_FOR statement and end with ETM_MACRO_NEXT statement.**

**Category:**

**MODBUS sub_function; MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_NEXT (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| **0** | | **0C D6** | | ***Sub_funciton code*** | | | |
| | | | | | | | |

### 7.6.3 Exiting a FOR loop block

※ **eRET ETM_MACRO_EXIT_FOR(HANDLE *h*, BYTE *cardNo*)**

**Description:**

**An ETM_MACRO_EXIT_FOR command can abort the current loop immediately if it is executed inside the loop.**

**Category:**

**MODBUS sub_function; MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_EXIT_FOR (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| 0 | | 0C DE | | *Sub_funciton code* | | | |
| | | | | | | | |

## 7.7 Condition Command (IF~ELSE)

### 7.7.1 IF condition

※Δ eRET ETM_MACRO_IF (HANDLE *h*, BYTE *cardNo*, long *varNo*, WORD *Operator*, long *varNo1*)

**Description:**

**A conditional checking ETM_MACRO_IF has to end the whole block with an ETM_MACRO_END_IF statement. One ETM_MACRO_ELSE statement can be added inside the conditional block and will be executed if the ETM_MACRO_IF statement turns out to be false. Up to 6 nested IF statements are supported (one outer and six inner IF conditions).**

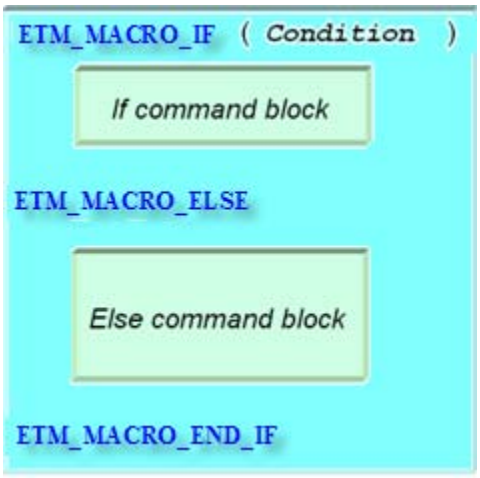



**Figure 6: ETM_MACRO_IF application structure**

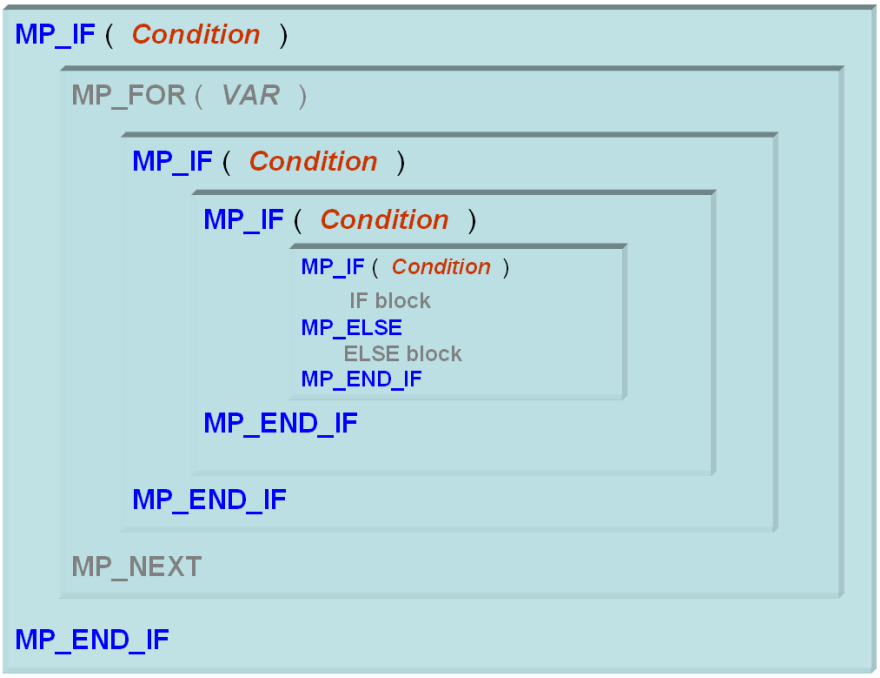



**Figure 7: Nested IF statements**

**Category:**

**MODBUS sub_function; MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *varNo*: | The first variable of the comparison expressions. Can either be a variable of long type (range: -2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511) |
| *Operator*: | Comparison operators:<br>string constant Description<br>"<" LT Less than (ASCII code = 0x003C)<br>">" GT Greater than (ASCII code = 0x003E)<br>"<=" LE Less than or equal to (ASCII code = 0x3D3C)<br>">=" GE Greater than or equal to (ASCII code = 0x3D3E)<br>"==" EQ Equal to (ASCII code = 0x3D3D)<br>"!=" NE Not equal to (ASCII code = 0x3D21) |

| | |
|---|---|
| *varNo1*: | **The second variable of the comparison expressions. Can either be a variable of long type (range: -2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**The ASCII code of operator.**

| Operator | ASCII (dec) | ASCII (hex) |
|---|---|---|
| < | 60 | 003C |
| <= | 15676 | 3D3C |
| > | 62 | 003E |
| >= | 15678 | 3D3E |
| == | 15677 | 3D3D |
| = | 61 | 003D |
| != | 15649 | 3D21 |
| ! | 33 | 0021 |

```
ETM_MACRO_IF (h, 1, VAR1, ”<=”, VAR2);
//The double quotes (“”) is required for Operator parameter.
```

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 13 | 01 | 10 | 1F 40 | 00 06 | 0C |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C D7 | *Sub_funciton code* |
| 1 | 7F FF | *MSW address of varNo* |
| 2 | 00 01 | *LSW address of varNo* |
| 3 | 3C 3D | *Operator (0x3C3D ➔ “<=”)* |
| 4 | 7F FF | *MSW address of varNo1* |
| 5 | 00 02 | *LSW address of varNo1* |

**Related example:**

```
ETM_MP_CREATE(h, 1, MP100); // Write #100 Macro Program into i8094H.
//=========================================================
// The following functions will not be executed, but will be written into
// i8094H for further execution.
ETM_MACRO_SET_MAX_V(h, 1, AXIS_X, 20000);
//The maximum speed of the axis X is 20K PPS.
ETM_MACRO_NORMAL_SPEED(h, 1, AXIS_X, 0);
```

```
//Set symmetric T-curve for axis X.
ETM_MACRO_SET_V(h, 1, AXIS_X, 2000);
//set the speed of axis X to be 2000 PPS.
ETM_MACRO_SET_A(h, 1, AXIS_X, 1000);
//set the acceleration of axis X to   be 1000 PPS/sec
ETM_MACRO_SET_SV(h, 1, AXIS_X, 2000);
//set the start speed of axis X to be 2000 PPS.
ETM_MACRO_SET_AO(h, 1, AXIS_X, 0);
//set the number of remaining offset pulses for axis X to be 0 PPS.
//-------------------------------------------------------------------------------------------------
ETM_MACRO_SET_VAR(h, 1, VAR1, 100); //VAR1 = 100。
ETM_MACRO_SET_VAR(h, 1, VAR2, 200); //VAR2 = 200。
ETM_MACRO_IF(h, 1, VAR1, "<", VAR2);
ETM_MACRO_FIXED_MOVE(h, 1, AXIS_X, 1000);
//command to move axis X for 1000 pulses.
ETM_MACRO_ELSE(h, 1);
ETM_MACRO_FIXED_MOVE(h, 1, AXIS_X, -1000);
// command to move axis X for -1000 pulses.
ETM_MACRO_END_IF(h, 1);
ETM_MACRO_MP_CLOSE(h, 1);
// module 1, Macro Program #100 is written into i8094H
//=========================================================
ETM_MP_CALL(h, 1, MP100);
// execute  Macro Program #100 of module 1.
```

### 7.7.2 ELSE statement

**※Δ eRET ETM_MACRO_ELSE (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**The block under ETM_MACRO_ELSE will be executed if the conditional statement of ETM_MACRO_IF is false; otherwise, it will be ignored.**

**Category:**

**MODBUS sub_function; MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_ELSE (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0C D8 | | *Sub_funciton code* | | | |
| | | | | | | | |

### 7.7.3 End of IF statement

**※Δ eRET ETM_MACRO_END_IF (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function indicates the end of a conditional block. A conditional block always begins with an ETM_MACRO_IF statement and ends with this function.**

**Category:**

**MODBUS sub_function; MP and ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_END_IF (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0C D9 | | *Sub_funciton code* | | | |
| | | | | | | | |

## 7.8 Jump Commands (GOTO-LABEL)

### 7.8.1 GOTO statement

※ eRET ETM_MACRO_GOTO(HANDLE *h*, BYTE *cardNo*, BYTE *lableNo*)

**Description:**

**The ETM_MACRO_GOTO Macro statement is a label based statement that transfers the control directly to the labeled location in the Macro program. The goto statement is very useful while working with nested loops. It enables to leave all nested nested loops with one command by jumping from the inner loop to an ETM_MP_LABEL position ouside the nested loop.**

**Limitations:**

- **A Maximum of four ETM_MACRO_LABEL position can be defined.**
- **It is NOT allowed to jump into a loop block. Therefore, the ETM_MACRO_LABEL statement can not be located inside a loop.**
- **Jumping into an MP_IF block is prohibited. ETM_MACRO_LABEL command is not allowed to be located inside an ETM_MACRO_IF block.**
- **You can only jump within a Macro table and it is not allowed to jump from one Macro table (e.g. MP1) to another Macro table (e.g. MP2).**
- **Can not be used inside an ISR Macro table.**


**Figure 8: ETM_MACRO_GOTO application**

**Category:**

**MODBUS sub_function; only inside MP Macro table allowed.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |
| ***labelNo:*** | **Only four labels are supported for each MP Macro table: 0, 1, 2, 3 (LABEL0, LABEL1, LABEL2, LABEL3)** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_GOTO (h, 1, 3);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C DF | *Sub_funciton code* |
| 1 | 00 03 | *labelNo (3 →0x03)* |

### 7.8.2 LABEL statement

※ eRET ETM_MACRO_LABEL(HANDLE *h*, BYTE *cardNo*, BYTE *lableNo*)

**Description:**

This function indicates the target position of a jump statement. The ETM_MACRO_LABEL can be positioned before or after ETM_MACRO_GOTO Macro statement. In a Macro only one ETM_MACRO_LABEL with the same Label number can be used but more than one ETM_MACRO_GOTO with the same Label number is allowed.

**Note:**

ETM_MACRO_LABEL command is not allowed inside a MP_IF or MP_FOR block.

**Category:**

MODBUS sub_function; only inside MP Macro table allowed.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *labelNo:* | Only four labels are supported for each MP Macro table: 0, 1, 2, 3 (LABEL0, LABEL1, LABEL2, LABEL3) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

ETM_MACRO_LABEL (h, 1, 3);

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |
| Register[] | | Value (hex) | | Remarks | | | |
| 0 | | 0C E1 | | *Sub_funciton code* | | | |
| 1 | | 00 03 | | *labelNo (3 →0x03)* | | | |

## 7.9 TIMER

※ eRET ETM_MACRO_TIMER (HANDLE *h*, BYTE *cardNo*, long *varNo*)

**Description:**

Delay the execution of the following Macro commands. The execution of the following Macro commands is held until the specified time has elapsed.

**Category:**

MODBUS sub_function; for MP Macro table.

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *varNo:* | Time in milliseconds. The following inputs are supported:<br>Value (0 ~ +2,000,000,000) ms, or<br>MACRO variables (VAR0 ~ VAR511) |

**Return:**

0: Success; Others: Fail (Please refer to chapter 9.2)

**MODBUS example:**

ETM_MACRO_TIMER (h, 1, 200);

//delay the execution of the function for 200ms.

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0D | 01 | 10 | 1F 40 | 00 03 | 06 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0C DA | *Sub_funciton code* |
| 1 | 00 00 | *MSW of varNo* |
| 2 | 00 C8 | *LSW of varNo (200 = 0xC8)* |

## 7.10 Wait until motion command has been executed (for MP only)

**eRET ETM_STOP_WAIT(HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**※ eRET ETM_MACRO_STOP_WAIT(HANDLE *h*, BYTE *cardNo*, BYTE *axis*)**

**Description:**

**The command ensures that the motion command for the assigned axis is completed before the next macro command is being executed. The command waits until the motion control chip has finished outputting pulses for the designated axis then continues to process the following command.**

**Category:**

**MODBUS sub_function; RTC, MP.**

**Parameters:**

| Paramters | Description |
|---|---|
| *cardNo*: | Module number → MP-8000 : 1~7 |
| *axis:* | Axis or axes (Please refer to Table 2)<br>The axis can be either X,Y, Z or U (1 or 2 or 4 or 8). |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**Remark:**

**The Sub_function code of ETM_STOP_WAIT is 0A DB.**
**The Sub_function code of ETM_MACRO_STOP_WAIT is 0C DB.**

**MODBUS example:**

**ETM_STOP_WAIT (h, 1, *AXIS_Y*);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 0B | 01 | 10 | 1F 40 | 00 02 | 04 |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A DB/0C DB | *Sub_funciton code* |
| 1 | 00 02 | *axis (2 → AXIS_Y)* |

## 7.11 User-defined RINT

**※Δ eRET ETM_MACRO_SET_RINT (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**This function causes the i8094H module to send a hardware interrupt with the flag 0x04 to the host controller (ET-M8194H).**

**Category:**

**MODBUS sub_function; MP ans ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_SET_RINT (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| **0** | | **0C DC** | | ***Sub_funciton code*** | | | |
| | | | | | | | |

## 7.12 Exit Macro Table

**※Δ eRET ETM_MACRO_EXIT_MACRO (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**The execution of a MP or ISR Macro table will be stopped and the table will be exited as soon as the Macro interpreter encounters this command. If a nested Macro call is being executed, this command causes the command interpreter to jump to calling Macro layer one level up (Figure 9). If the top level Macro layer encounters ETM_MACRO_EXIT_MACRO, the execution of Macro commands terminates all together. Only in case of a new harware interrupt occurs, the relevant ISR Macro will be executed. This command behaves similarly to the ETM_MACRO_MP_CLOSE command except that it can be called at any place inside a Macro table, whereas the ETM_MACRO_MP_CLOSE command has to be placed at the end of a Macro table.**

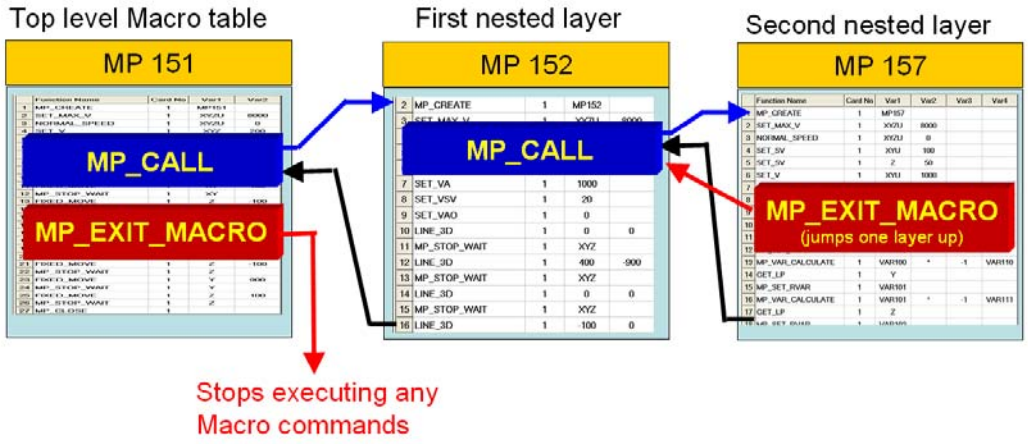



**Figure 9: Exiting Macro tables**

**Category:**

**MODBUS sub_function; MP ans ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MACRO_EXIT_MACRO (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| **0** | | **0C DD** | | ***Sub_funciton code*** | | | |
| | | | | | | | |

## 7.13 Terminate all Macro executions

**eRET ETM_MP_TERMINATE (HANDLE *h*, BYTE *cardNo*)**

**※Δ eRET ETM_MACRO_MP_TERMINATE (HANDLE *h*, BYTE *cardNo*)**

**Description:**

**All currently running MP or ISR Macro program will be terminated immediately. After all the Macro programs have been terminated successfully, the command automatically resets itself so that new Macro programs can be executed. This command is intended for emergent situation where any Macro execution needs to be stopped immediately. This command can be called both inside a Macro program or outside a Macro program, for example, by the host controller or a remote HMI controller.**

**Warning:**

**This command does not stop the motion chip from continuing executing the motion command received prior to the ETM_MP_TERMINATE command. In order to stop the motion chip from outputting pulse signals, one of the following commands has to be called:**

- **ETM_STOP_SUDDENLY**
- **ETM_STOP_SLOWLY**
- **ETM_VSTOP_SUDDENLY**
- **ETM_VSTOP_SLOWLY**

**Therefore, if it required stopping motion and Macro commands from execution then one of the above four STOP commands as well as the ETM_MP_TERMINATE has to be called.**

**Category:**

**MODBUS sub_function; RTC, MP, ISR.**

**Parameters:**

| Paramters | Description |
|---|---|
| ***cardNo*:** | **Module number → MP-8000 : 1~7** |

**Return:**

**0: Success; Others: Fail (Please refer to chapter 9.2)**

**MODBUS example:**

**ETM_MP_TERMINATE (h, 1);**

**The MODBUS command is listed as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 09 | 01 | 10 | 1F 40 | 00 01 | 02 |
| **Register[]** | | **Value (hex)** | | **Remarks** | | | |
| **0** | | **0A E2/0C E2** | | ***Sub_funciton code*** | | | |
| | | | | | | | |

# 8 Stand Alone Controller

**The ET-8194H can operate as a standalone controller without being connected to a Modbus master. Right after power on, the initialization and macro execution sequences are as follows:**

- **<u>Step 1:</u> ET-M8194H initializes the i8094H according to the Initial Table setting. Hardware is immediately configured**
- **<u>Step 2:</u> Power-On Macro is being executed**
- **<u>Step 3:</u> Power-On Macro calls the main motion MACRO**

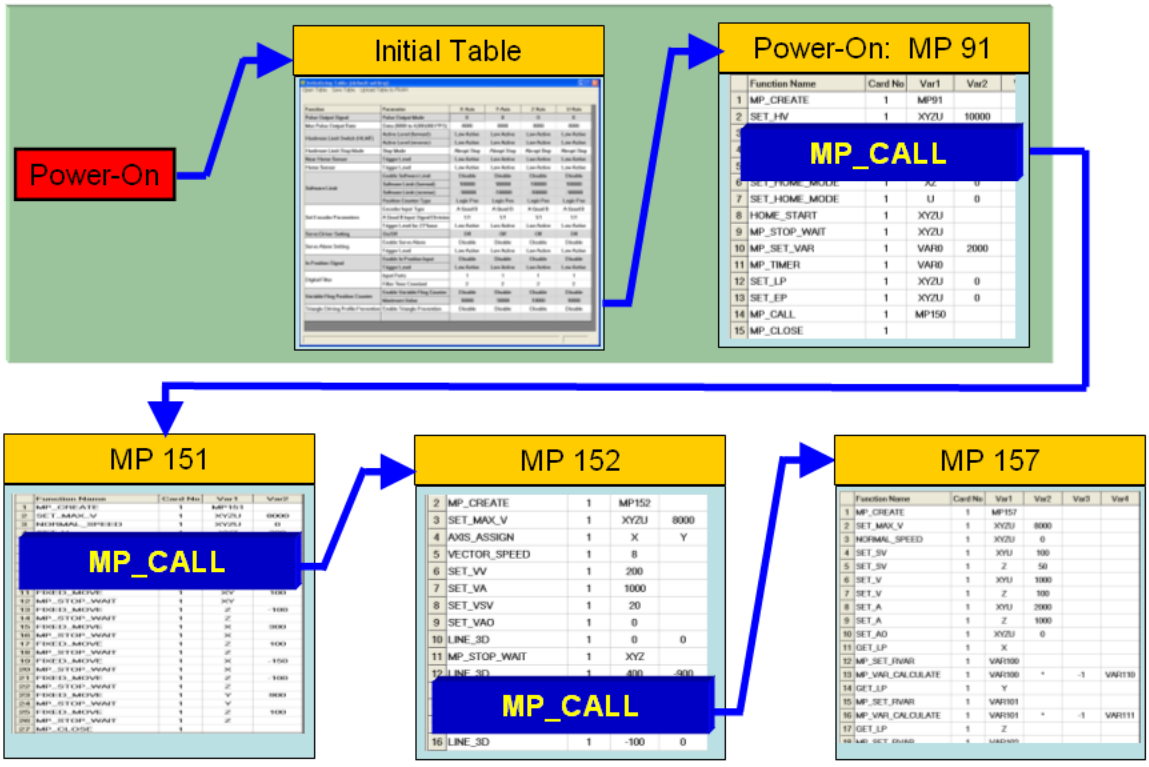



**Figure 10: Power on procedure with three nested Macro calls**

**In order for the ET-M8194H to act as a stand alone controller, it is necessary to assign an MP Macro program (Power-On Macro program) which has to be called direct after initialization. In a stand alone mode, the Power-On Macro program acts as the top level main Macro program. From this top level program table, other MP Macro program tables can be called. Nested Macro calls are supported, too. The configuration procedures are as**

follows.

1. Set the initialization table. (chapter 7)

2. Write the macro program for the power-on macro table and if required for the nested macro program. (chapter 8)

3. Assign the variable MD1001 the MP macro table number (MP1-MP157) which acts as a Power-ON Macro (chapter 2.20). If a value outside the valid range (1 -157) is used then no Power-On Macro will be called and the device does not operate as a stand alone controller. In this case ET-M8194H needs to be connected to a Modbus master to receive the necessary commands via communication.

**MODBUS example:**

Set Macro table MP150 as the Power-On Macro table. Since the function is defined as follows:

eRET ETM_WRITE_MD (HANDLE *h*, BYTE *cardNo*, long *mdNo*, long *ldata*, float *fdata*)

**Paramters:**
**cardNo = 1 (ET-M8194H has only got one slot)**
**mdNo = MD1001 (Value = 3000 + 2*1001 = 5002 = 0x138A)**
**ldata = MP150 (Value = 150 = 0x96)**
**fdata = 0 ( this parameter is only valid for the range MD1024 – MD2047)**

**//The following function call sets MP150 as the Power-On Macro**
**ETM_WRITE_MD ( h, 1, MD1001, MP150 , 0);**

**The MODBUS command is as follows:**

| TID (hex) | PID (hex) | Field Length (hex) | UID (hex) | FC (hex) | St_Addr. (hex) | Word Count (hex) | Byte Count (hex) |
|---|---|---|---|---|---|---|---|
| 00 01 | 00 00 | 00 15 | 01 | 10 | 1F 40 | 00 07 | 0E |

| Register[] | Value (hex) | Remarks |
|---|---|---|
| 0 | 0A 27 | *Sub_funciton code* |
| 1 | 00 00 | *MSW of mdNo (= 0)* |
| 2 | 13 8A | *LSW of mdNo (n= 0x138A)* |
| 3 | 00 00 | *MSW of ldata* |
| 4 | 00 96 | *LSW of ldata* |
| 5 | 00 00 | *MSW of fdata* |
| 6 | 00 00 | *LSW of fdata* |

# 9 Communication Functions and Error Code

## 9.1 Communication Functions:

**The functions help to establish the Ethernet connection between the PC and the ET-M8194H modules. With these functions, the MODBUS command can be sent out manually. The MODBUS command on this connection can be monitored, as well.**

**1. HANDLE ETM_CONNECT(char* *ip*, DWOAD *dwTimeout*)**
**Function: Establish the Ethernet connection between PC and ET-M8194H. The connection is required before senting out any MODBUS command. The ET-M8194H allows 29 Clients to connect to it simultaneously. The returned handle stands for the conection to every ET-M8194H, and this handle is needed to control the ET-M8194H.**

**Parameters: ip: The IP adress of specific ET-M8194H**
**dwTimeout: Set the timeout(unit: ms)**

**Return: If the return value is INVALID_HANDLE_VALUE represent fail, others success.**

**Example: [VC++]**

```
#include <winsock2.h>
#include "ET_M8194H_API.h"
Handle hETM1, hETM2;
int iRet;
hETM1 = ETM_Connect("192.168.0.1", 1000);
//The connection to 1st ET-M8194H
hETM2 = ETM_Connect("192.168.0.2", 1000);
//The connection to 2nd ET-M8194H
iRet = ETM_RESET_CARD(hETM1, 1); //Reset the 1st ET-M8194H
iRet = ETM_RESET_CARD(hETM2, 1); // Reset the 2nd ET-M8194H
```

**2. eTCP_CONNECT ETM_RECONNECT(HANDLE *h*)**
**Function: Used to re-establish the Ethernet connection between PC and ET-M8194H.**

**Return: 1: Connected; 0: Disconnected**

**3. eRET ETM_DISCONNECT (HANDLE *h*)**
**Function:** Disconnect the established Ethernet connection.

**Return:** 0: Success; Others: Fail (Please refer to chapter 9.2)

**4. eTCP_CONNECT ETM_CONNECTION_STATE (HANDLE *h*)**
**Function:** Shows the status of Ethernet connection.

**Return:** 1:Connected; 0:Disconnected

**5. eRET ETM_Set_DebugMode (HANDLE *h*, BYTE *DebugMode*)**
**Function:** This function enables/disable to display the MODBUD command. This feature helps user to debug the functions based on MODBUS Function Code 03, 04 ans 16.

**Parameter:** DebugMode：1: Enable Debug Message, 0: Disable Debug Message

**Return:** 0: Success; Others: Fail (Please refer to chapter 9.2)

**6. eRET ETM_MODBUS_CMD (HANDLE *h*, char* *Request*, int *Req_Len*, char* *Response*, int* *Res_Len*)**
**Function:** Send out the MODBUS command without related message.

**Parameters:** Request: The sent-out MODBUS command
Req_Len: The length of sent-out MODBUS command
Response: The MODBUS response from ET-M8194H
Res_Len: The length of MODBUS response from ET-M8194H

**Return:** 0: Success; Others: Fail (Please refer to chapter 9.2)

**7. eRET ETM_MODBUS_CMD_DEBUG (HANDLE *h*, char* *Request*, int *Req_Len*, char* *Response*, int* *Res_Len*)：**

**Function:** Send out the MODBUS command with the related message.

**Parameters:** Request : The sent-out MODBUS command
Req_Len: The length of sent-out MODBUS command
Response: The MODBUS response from ET-M8194H
Res_Len: The length of MODBUS response from ET-M8194H

**Return:** 0: Success; Others: Fail (Please refer to chapter 9.2)

**8. eRET ETM_SET_WORD_ORDER (HANDLE *h*, BYTE bWordOrder)：**

**Function:** Used to set the order of the MSW and LSW.

**Parameters:** bWordOrder : 0: MSW at the first Register ; 1: MSW at the second Register

**Return:** 0: Success; Others: Fail (Please refer to chapter 9.2)

**9. eRET ETM_GET_WORD_ORDER (HANDLE *h*, BYTE* pbWordOrder)：**

**Function:** Used to get the order of the MSW and LSW.

**Parameters:** pbWordOrder : 0: MSW at the first Register ; 1: MSW at the second Register

**Return:** 0: Success; Others: Fail (Please refer to chapter 9.2)

## 9.2 ErrorCode

**API Library communications return value (eRET) is defined and described as follows:**

**(1) ETM_SUCCESS 0x00**
**=> Successful communication via MODBUS.**

**(2) MB_ILLEGAL_FUNCTION_CODE 0x01**
**=> Error in the Function_Code of MODBUS.**

**(3) MB_ILLEGAL_DATA_ADDRESS 0x02**
**=> Invalid starting-address of MODBUS command. For instance, there are the two Registers are be to accessed ( LP_X, LP_Y … ). If WORD Order is set to be 1, then the starting-address is the Low_Word address.**

**(4) MB_ILLEGAL_DATA_VALUE 0x03**
**=> Invalid data of MODBUS command, such as the number of parameters, the target-axis, the addresses of MD, VAR bVAR.**

**(5) MB_SLAVE_DEVICE_FAILURE 0x04**
**=> Indicates that no i-809H is resident on ET-M8194H.**

**(6) MB_SLAVE_DEVICE_BUSY 0x06**
**=> Indicates the DPRAM-Buffer on i-8094H is occupied; no more command can be forwarded through DPRAM-Buffer.**

**(7) WRONG_CARD_NO 0x0B**
**=> Invalid UID ( the CardNo parameter ).**

**(8) ETM_SOCKET_ERR -1**
**=> Error in communication, please check the Ethernet connection.**

**(9) ETM_WRITE_DENY -2**
**=> Indicates the writing command is denied after enabling the writing-protection feature.**

**(10) ETM_PAR_ERROR -3**
**=> Indicates parameter value range error.**

**(11) ETM_UNDEFINED_ERROR -4**
**=> Indicates undefined error.**

**(12) ETM_VALID_HANDLE 7**

**=> Indicates a valid handle**

**(13) ETM_ INVALID_HANDLE -1**

**=> Indicates an invalid handle**

# Appendix

## A. MODBUS Input Registers

Use Modbus function code 4 to read the input register table. The table addresses defined below is zero-based. For some PLC or HMI, the addresses maybe show on their documents as 3xxxx, x is digital number. For example, 30001 to 39999. '3' represents the table is defined for Input registers; and xxxx is the address of a register.

In PLC, the first input register defined at address 30001 is the same as the first register at address 0 shown in the following table. Most PLC systems which addresses do not start from zero can be referred as one-based system. The following tables will only show zero-based addresses. When PLC or HMI users configure their registers, they have to consider register number mapping problem carefully.

When a value needs two registers to store, the order of MSW and LSW can be determined by setting a flag (WORD Order). WORD Order is defined at address 14 (zero-based) in holding register table. If WORD Order is set to 0, the first register (lower address) is high-word (**MSW**) and the second register is low-word (**LSW**).

## A.1 Read FRnet DI/O

**Appendix table 1: FRnet DI/O (PLC address: 30001 to 30016)**

| Read FRnet DIO | | | | |
|---|---|---|---|---|
| **Variable Name** | | **Address (zero based)** | **Type** | **Comment** |
| FRnet DO | Group 0 | 0 | R | Read the FRnet DO group 0 (16-bit) |
| | Group 1 | 1 | R | Read the FRnet DO group 1 (16-bit) |
| | Group 2 | 2 | R | Read the FRnet DO group 2 (16-bit) |
| | Group 3 | 3 | R | Read the FRnet DO group 3 (16-bit) |
| | Group 4 | 4 | R | Read the FRnet DO group 4 (16-bit) |
| | Group 5 | 5 | R | Read the FRnet DO group 5 (16-bit) |
| | Group 6 | 6 | R | Read the FRnet DO group 6 (16-bit) |
| | Group 7 | 7 | R | Read the FRnet DO group 7 (16-bit) |
| FRnet DI | Group 8 | 8 | R | Read the FRnet DI group 8 (16-bit) |
| | Group 9 | 9 | R | Read the FRnet DI group 9 (16-bit) |
| | Group 10 | 10 | R | Read the FRnet DI group 10 (16-bit) |
| | Group 11 | 11 | R | Read the FRnet DI group 11 (16-bit) |
| | Group 12 | 12 | R | Read the FRnet DI group 12 (16-bit) |
| | Group 13 | 13 | R | Read the FRnet DI group 13 (16-bit) |
| | Group 14 | 14 | R | Read the FRnet DI group 14 (16-bit) |
| | Group 15 | 15 | R | Read the FRnet DI group 15 (16-bit) |

## A.2 Read DI Status of Daughterboard

**Appendix table 2: DI Status of Daughterboard (PLC address: 30017 to 30020)**

| Read DI Status (of Daughterboard) | | | | |
|---|---|---|---|---|
| **Variable Name** | | **Address (zero based)** | **Type** | **Comment** |
| DI ALL | X-Axis | 16 | R | All DI status of AXIS-X,<br>If the individual DI is needed, read X_DI_0 ~ X_DI_9. |
| | Y-Axis | 17 | R | All DI status of AXIS-Y,<br>If the individual DI is needed, please read Y_DI_0 ~ Y_DI_9. |
| | Z-Axis | 18 | R | All DI status of AXIS-Z,<br>If the individual DI is needed, please read Z_DI_0 ~ Z_DI_9. |
| | U-Axis | 19 | R | All DI status of AXIS-Z,<br>If the individual DI is needed, please read U_DI_0 ~ U_DI_9. |

## A.3 Read Error Code

**Appendix table 3: Error Code (PLC address: 30021 to 30026)**

| Read Error Code | | | | |
|---|---|---|---|---|
| **Variable Name** | | **Address (zero based)** | **Type** | **Comment** |
| ERROR CODE | X-Axis | 20 | R | The error occurred on AXIS-X. Please refer to ETM_GET_ERROR_CODE() for detailed information. |
| | Y-Axis | 21 | R | The error occurred on AXIS-Y. Please refer to ETM_GET_ERROR_CODE() for detailed information. |
| | Z-Axis | 22 | R | The error occurred on AXIS-Z. Please refer to ETM_GET_ERROR_CODE() for detailed information. |
| | U-Axis | 23 | R | The error occurred on AXIS-U. Please refer to ETM_GET_ERROR_CODE() for detailed information. |
| | | | | |
| MP Call number | | 24 | R | Current running MP or ISR number |
| EMERGENCY_STOP state | | 25 | R | Shows the current emergency state<br>1- emergency stop enabled<br>0 – no emergency disabled |

## A.4 Read Logic and Encoder Position, Acceleration, Velocity

**Appendix table 4: Logic and Encoder Position, Acceleration, Velocity (PLC address: 30027 to 30058)**

| Motion Status | | | | |
|---|---|---|---|---|
| **Variable Name** | | **Address (zero based)** | **Type** | **Comment** |
| Logic Position | X | 26-27 | R | Logical position of X-axis. |
| | Y | 28-29 | R | Logical position of Y-axis. |
| | Z | 30-31 | R | Logical position of Z-axis. |
| | U | 32-33 | R | Logical position of U-axis. |
| Encoder Position | X | 34-35 | R | Encoder feedback position of X-axis. It takes two registers. |
| | Y | 36-37 | R | Encoder feedback position of Y-axis. It takes two registers. |
| | Z | 38-39 | R | Encoder feedback position of Z-axis. It takes two registers. |
| | U | 40-41 | R | Encoder feedback position of U-axis. It takes two registers. |
| Current velocity | X | 42-43 | R | Current velocity of X-axis. It takes two registers. |
| | Y | 44-45 | R | Current velocity of Y-axis. It takes two registers. |
| | Z | 46-47 | R | Current velocity of Z-axis. It takes two registers. |
| | U | 48-49 | R | Current velocity of U-axis. It takes two registers. |
| Current Acceleration | X | 50-51 | R | Current acceleration of X-axis. It takes two registers. |
| | Y | 52-53 | R | Current acceleration of Y-axis. It takes two registers. |
| | Z | 54-55 | R | Current acceleration of Z-axis. It takes two registers. |
| | U | 56-57 | R | Current acceleration of U-axis. It takes two registers. |

## A.5 Read Stop Status

**Appendix table 5: Stop Status (PLC address: 30059 to 30062)**

| Variable Name | | Address (zero based) | Type | Comment |
|---|---|---|---|---|
| Stop Status | X | 58 | R | 1:stop, 0: moving |
| | Y | 59 | R | 1:stop, 0: moving |
| | Z | 60 | R | 1:stop, 0: moving |
| | U | 61 | R | 1:stop, 0: moving |

## A.6 Read Latch

**Appendix table 6: Read Latch (PLC address: 30063 to 30070)**

| Variable Name | | Address (zero based) | Type | Comment |
|---|---|---|---|---|
| Latch | X | 62-63 | R | Value in X-axis latch register long type |
| | Y | 64-65 | R | Value in Y-axis latch register long type |
| | Z | 66-67 | R | Value in Z-axis latch register long type |
| | U | 68-69 | R | Value in U-axis latch register long type |

## A.7 Read Error State and Free Buffer Size

**Appendix table 7: Error State and Free Buffer Size (PLC address: 30071 to 30072)**

| Variable Name | Address (zero based) | Type | Comment |
|---|---|---|---|
| ERROR_STATE | 70 | R | 1: error occurred; 0: no error |
| FREE_BUFFER_SIZE | 71 | R | Get the available block-numbers in Buffer. The maximum block is 30. |
| | | | |

## A.8 Read i8094H Interrupt

**Appendix table 8: i8094H Interrupt (PLC address: 30073 to 30081)**

| I8094H Interrupt ET-M8194H (Return Interrupt RINT) | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero based)** | **Type** | **Comment** |
| RINT_STATE_ALL | 72 | R | Read the RINT state. For detailed, please refer to the descriptions of **ETM_RINT_WAIT()** and **ETM_RINT_ENABLE()**. |
| Line_Scan_Completed | 73 | R | Bit0 of RINT_STATE |
| MP_Completed | 74 | R | Bit1 of RINT_STATE |
| User-Defined_RINT | 75 | R | Bit2 of RINT_STATE |
| Continuous_Inp_Interrupt | 76 | R | Bit3 of RINT_STATE |
| .....Undefined | 77 | -- | 0 |
| .....Undefined | 78 | -- | 0 |
| Axes_Error | 79 | R | Bit6 of RINT_STATE |
| Module_Error | 80 | R | Bit7 of RINT_STATE |

## A.9 Read Firmware Version

**Appendix table 9: Firmware Version (PLC address: 30082 to 30087)**

| Others | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero based)** | **Type** | **Comment** |
| I8094H Module ID | 81 | R | i8094H：0x**44**<br>i8094A：0x**55** |
| ET-M8194H<br>Firmware Version | 82-83 | R | 0x**0100**0000 means<br>Ver：1.00 |
| I8094H<br>Firmware Version | 84-85 | R | 0x02210201<br>▪ High Word for PCB version<br>▪ Low WORD represents the i8094H firmware Version:<br>// Example:<br>▪ High WORD<br>o PCB Version 2.21<br>▪ Low WORD<br>o 0201->02 major version, 01-> minor version number |
| TCN | 86 | R | **T**otal Modbus Client **C**onnection **N**umber |

## A.10 Read ET-M8194H and i8094H State

**Appendix table 10: ET-M8194H and i8094H State (PLC address: 30101 to 30103)**

| Current state of the ET-M8194H and i8094H | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero baed)** | **Type** | **Comment** |
| ET-M8194H state | 100 | R | The current state of the ET-M8194H module:<br>0- ready for receiving new RTC and Macro commands<br>1- ETM is in an MACRO download state (MPn)<br>2- ETM is in an MACRO download state (ISRn) |
| i8094H state | 101 | R | The current state of the i8094H module:<br>3- ready for receiving new RTC commands<br>4- Calling the initialization table<br>5- Initialization finished<br>6- downloading MACRO commands (MP or ISR commands) to the FRAM or is waiting for the next MACRO command<br>7- Macro download finished<br>8- MACRO program (MP) is being executed<br>9- MACRO program (MP) execution finished<br>10- Interrupt MACRO (ISR) program is being executed<br>11- Interrupt MACRO (ISR) execution finished<br>255- firmware is booting |
| Illegal Modbus function | 102 | R | Cause of the ILLIGAL FUNCTION Modbus exception response:<br>0- no error<br>1- function code not supported<br>2- Parameter value in NOT within defined range<br>3- No such command |

| | | | |
|---|---|---|---|
| | | | 4- More than the avalaible command lines are used for the specific MACRO program (MPn or ISRn)<br>5- A MP command is being used inside ISR MACRO.<br>6- A ISR command is being used inside MP MACRO<br>7- A RTC command is being used inside a MP or ISR MACRO<br>8- A MP or ISR MACRO command is being used outside a MACRO program as real time command<br>**0xB1** - The max number of nested MP_IF commands has been exceeded<br>**0xB2** - MP_ELSE command outside MP_IF block<br>**0xB3** - MP_END_IF command outside MP_IF block<br>**0xB4** - MP_IF block has not been closed by MP_END_IF command<br>**0xB5** - Number of nested MP_FOR exceeds the limit<br>**0xB6** -MP_EXIT_FOR command outside MP_FOR loop<br>**0xB7** - MP_NEXT command outside MP_FOR block<br>**0xB8** - MP_FOR block has not been closed with a MP_NEXT command<br>**0xB9** - Label number is being used more than once<br>**0xBA** - MP_LABEL inside a MP_IF block. Jumping into a MP_IF block is prohibited<br>**0xBB** - MP_LABEL inside a MP_FOR block. Jumping into a MP_FOR block is prohibited<br>**0xBC** - MP_GOTO command has NO corresponding MP_LABEL command<br>**0xC0** - A DWORD is required but only one register has been received<br>**0xC1** -DPRAM of i8094H is full<br>**0xC2** -Module not in slot<br>**0xC3** - A Macro program is currently being executed inside the i8094H |

## A.11 Macro Program Download Error Messages

**Appendix table 11: Program Download Error Message (PLC address: 30111 to 30115)**

| Macro program download error message | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero based)** | **Type** | **Comment** |
| Macro MP number (MPxx) | 110 | R | The last downloaded Macro MP program number since ET-M8194H power on. |
| Macro ISR number (ISRxx) | 111 | R | The last downloaded Macro ISR program number since ET-M8194H power on. |
| Command type | 112 | R | The number identifying the command. See sub function code. The register following the 8000 sub function register identifies the command. |
| Line number | 113 | R | The line number of the Macro program at which the error occurred |
| Error type | 114 | R | Describes the type of error |

## A.12 Macro Program Execution Error Messages

**Appendix table 12: Macro Program Execution Error Messages (PLC address: 30116 to 30120)**

| Macro program execution error message | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero based)** | **Type** | **Comment** |
| Macro MP number (MPxx) | 115 | R | The last downloaded Macro MP program number since ET-M8194H power on. |
| Macro ISR number (ISRxx) | 116 | R | The last downloaded Macro ISR program number since ET-M8194H power on. |
| Command type | 117 | R | The number identifying the command at which the error occurred. See sub function code. The register following the 8000 sub function register identifies the command. |
| Line number | 118 | R | The line number of the Macro program at which the error occurred |
| Error type | 119 | R | Describes the type of error |

## A.13 Motion Chip Triggered Interrupt

**Appendix table 13: Motion Chip Triggered Interrupt (PLC address: 30131 to 30140)**

| Motion chip triggered interrupt | | | | |
|---|---|---|---|---|
| ***nINT*** | **Symbol** | **Address (zero based)** | **Type** | **Description** |
| 0 | PULSE | 130 | R | Interupt occurs when pulse is up. |
| 1 | P>=C- | 131 | R | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register. |
| 2 | P<C- | 132 | R | Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register. |
| 3 | P<C+ | 133 | R | Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register. |
| 4 | P>=C+ | 134 | R | Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register. |
| 5 | C-END | 135 | R | Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output |
| 6 | C-STA | 136 | R | Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output |
| 7 | D-END | 137 | R | Interrupt occurs when the driving is finished |
| 8 | HMEND | 138 | R | Automatic home search terminated |
| 9 | SYNC | 139 | R | Synchronous action was activated |

# B. Holding Registers

Use Modbus function code 6 or 16 to write to the holding register(s) and function code 3 to read the registers. The table addresses defined below is zero-based. For some PLC or HMI, the addresses maybe show on their documents as 4xxxx, x is digital number. For example, 40001 to 49999. '4' represents the table is defined for holding registers; and xxxx is the address of a register.

In PLC, the first holding register defined at address 40001 is the same as the first register at address 0 shown in the following table. Most PLC systems which addresses do not start from zero can be referred as one-based system. The following tables will only show zero-based addresses. When PLC or HMI users configure their registers, they have to consider register number mapping problem carefully.

When a parameters needs two registers to store, such as DWORD, long, float, the order of MSW and LSW can be determined by setting a flag (WORD Order). WORD Order is defined at address 14 (zero-based) in holding register table. If WORD Order is set to 0, the first register (lower address) is high-word (**MSW**) and the second register is low-word (**LSW**).

R/W – read/ write

## B.1 FRnet DO

**Appendix table 14: FRnet DO (PLC address: 40001 to 40008)**

| FRnet DO | | | | |
|---|---|---|---|---|
| **Variable Name** | | **Address (zero base)** | **Type** | **Comment** |
| FRnet DO | Group 0 | 0 | R/W | Set/Read the FRnet DO group 0 |
| | Group 1 | 1 | R/W | Set/Read the FRnet DO group 1 |
| | Group 2 | 2 | R/W | Set/Read the FRnet DO group 2 |
| | Group 3 | 3 | R/W | Set/Read the FRnet DO group 3 |
| | Group 4 | 4 | R/W | Set/Read the FRnet DO group 4 |
| | Group 5 | 5 | R/W | Set/Read the FRnet DO group 5 |
| | Group 6 | 6 | R/W | Set/Read the FRnet DO group 6 |
| | Group 7 | 7 | R/W | Set/Read the FRnet DO group 7 |

## B.2 Macro Call, FRnet Event and WORD Order Setting

**Appendix table 15: Macro Call, FRnet Event and WORD Order Setting (PLC address: 40009 to 40015)**

| Variable Name | Address (zero based) | Type | Comment |
|---|---|---|---|
| CALL_MPn | 8 | R/W | This is a simple method to call MP. Write the MPn at this address, the corresponding MPn will be called.<br>n=1-157 |
| FRNet Event | 9 | -- | Enable FRnet DI to generate events<br>0 – disable FRnet event trigger<br>1 – enable FRnet event trigger |
| Reserved | 10-13 | -- | Not defined |
| WORD Order | 14 | R/W | This setting is relevant for DWORD, long, float variables:<br><br>0: Big endian WORD (reverse type). First register represents the most significant word the second register represents the least significant word<br><br>1: Small Endian WORD<br>Most significant word is the second register |

## B.3 IP Address Setting

**Appendix table 16: IP Address Setting (PLC address: 40021 to 40031)**

| IP Address Setting | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero based)** | **Type** | **Comment** |
| Lock_IP | 20 ~ 23 | R/W | Lock/Read_Locked TCP Client IP |
| UnLock_IP | 24 | W | Unlock TCP Client IP |
| ET-M8194H_IP | 25~26 | R/W | Set ET-M8194H IP |
| ET-M8194H_Mask | 27~28 | R/W | Set ET-M8194H Mask |
| ET-M8194H_Gateway | 29~30 | R/W | Set ET-M8194H Gateway |

## B.4 Read/ Write Logic and Encoder Position, Acceleration, Velocity

**Appendix table 17: Logic and Encoder Position, Acceleration, Velocity (PLC address: 40091 to 40122)**

| Motion Status | | | | |
|---|---|---|---|---|
| **Variable Name** | | **Address (zero based)** | **Type** | **Comment** |
| Logic Position (LP) | X-Axis | 90-91 | R/W | Logical position of X-axis. |
| | Y-Axis | 92-93 | R/W | Logical position of Y-axis. |
| | Z-Axis | 94-95 | R/W | Logical position of Z-axis. |
| | U-Axis | 96-97 | R/W | Logical position of U-axis. |
| Encoder Position (EP) | X-Axis | 98-99 | R/W | Encoder feedback position of X-axis. |
| | Y-Axis | 100-101 | R/W | Encoder feedback position of Y-axis. |
| | Z-Axis | 102-103 | R/W | Encoder feedback position of Z-axis. |
| | U-Axis | 104-105 | R/W | Encoder feedback position of U-axis. |
| Current Velocity (CV) | X-Axis | 106-107 | R | Current velocity of X-axis. |
| | Y-Axis | 108-109 | R | Current velocity of Y-axis. |
| | Z-Axis | 110-111 | R | Current velocity of Z-axis. |
| | U-Axis | 112-113 | R | Current velocity of U-axis. |
| Current Acceleration (CA) | X-Axis | 114-115 | R | Current acceleration of X-axis. |
| | Y-Axis | 116-117 | R | Current acceleration of Y-axis. |
| | Z-Axis | 118-119 | R | Current acceleration of Z-axis. |
| | U-Axis | 120-121 | R | Current acceleration of U-axis. |

## B.5 Read/ Write VAR Variables

**Appendix table 18: VAR Variables (PLC address: 40301 to 41324)**

| Macro Variables | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero based)** | **Type** | **Comment** |
| VAR0 | 300-301 | R/W | Value of VAR0. |
| VAR1 | 302-303 | R/W | Value of VAR1. |
| … | … | | VARn = 300+2*n to 300+2*n+1 |
| VAR510 | 1320-1321 | R/W | Value of VAR510. |
| VAR511 | 1322-1323 | R/W | Value of VAR511. |

## B.6 Sub Function Code Definition

**Appendix table 19: Sub Function Code Definition (PLC address: 48001 to 48101)**

| Motion Commands<br>Sub Function Code 16 | | | |
|---|---|---|---|
| **Variable Name** | **Address (zero based)** | **Type** | **Comment** |
| Sub_Function_Code | 8000 | W | The mapping of sub function code can refer to table in **Appendix C**. |
| Reg1 | 8001 | W | The definitions of parameters depend on the sub functions. Please refer to the table in **Appendix C**. |
| Reg2 | 8002 | W | Users can use the sub function call method to implement most of the motion functions, especially for setting functions. |
| Reg3 | 8003 | W | For getting real-time information from modules, these tables in **Appendix A and B** are valuable. |
| Reg4 | 8004 | W | Users can refer to examples in the functions description above to know how to set the parameters. |
| … | | | The basic rules of parameter definitions are :<br>BYTE ➔ one register (low-byte)<br>WORD ➔ one register<br>DWORD ➔ two registers<br>long ➔ two registers<br>float ➔ two registers |
| Reg100 | 8100 | W | |

# C. Sub Function Code mapping table

Please use the FC=16 to call the sub-functions; the related starting address is 8000 (zero-based). The following table lists the reference-section, Sub_Function name and required-registers of the related sub-function-code.

Some code-fields are filled with xxxxxx; this means no sub-function-code is involved. Most of these functions are belong to the Getting command; the FC=3 or FC=4 will be used to get the needed information. Please refer to the MODBUS table in the Appendix A and B.

**Appendix table 20: Sub Function Code mapping table**

| Section | Sub_Function | Code (Hex) | Code (Dec) | No. of registers |
|---|---|---|---|---|
| | | reg(0) | reg(0) | |
| 2.2 | i8094H_REGISTRATION | xxxxxx | xxxxx | |
| 2.2 | i8094H_GET_VERSION | xxxxxx | xxxxx | |
| 2.3 | ETM_RESET_CARD | 0x0A0A | 2570 | 1 |
| 2.3 | ETM_CLEAR_CARD_BUFFER | 0x0A0B | 2571 | 1 |
| 2.4 | ETM_SET_PULSE_MODE | 0x0A0C | 2572 | 3 |
| 2.4 | ETM_MACRO_SET_PULSE_MODE | 0x0C0C | 3084 | 3 |
| 2.5 | ETM_SET_MAX_V | 0x0A0D | 2573 | 4 |
| 2.5 | ETM_MACRO_SET_MAX_V | 0x0C0D | 3085 | 4 |
| 2.6 | ETM_SET_HLMT | 0x0A0E | 2574 | 4 |
| 2.6 | ETM_MACRO_SET_HLMT | 0x0C0E | 3086 | 4 |
| 2.7 | ETM_LIMITSTOP_MODE | 0x0A0F | 2575 | 3 |
| 2.7 | ETM_MACRO_LIMITSTOP_MODE | 0x0C0F | 3087 | 3 |
| 2.8 | ETM_SET_NHOME | 0x0A10 | 2576 | 3 |
| 2.8 | ETM_MACRO_SET_NHOME | 0x0C10 | 3088 | 3 |
| 2.9 | ETM_SET_HOME_EDGE | 0x0A11 | 2577 | 3 |
| 2.9 | ETM_MACRO_SET_HOME_EDGE | 0x0C11 | 3089 | 3 |
| 2.10 | ETM_SET_SLMT | 0x0A12 | 2578 | 7 |
| 2.10 | ETM_MACRO_SET_SLMT | 0x0C12 | 3090 | 7 |
| 2.10 | ETM_CLEAR_SLMT | 0x0A13 | 2579 | 2 |
| 2.10 | ETM_MACRO_CLEAR_SLMT | 0x0C13 | 3091 | 2 |
| 2.11 | ETM_SET_ENCODER | 0x0A14 | 2580 | 5 |
| 2.11 | ETM_MACRO_SET_ENCODER | 0x0C14 | 3092 | 5 |

| Section | Sub_Function | Code (Hex) | Code (Dec) | No. of registers |
|---|---|---|---|---|
| | | reg(0) | reg(0) | |
| 2.12 | ETM_SERVO_ON | 0x0A15 | 2581 | 2 |
| 2.12 | ETM_MACRO_SERVO_ON | 0x0C15 | 3093 | 2 |
| 2.12 | ETM_SERVO_OFF | 0x0A16 | 2582 | 2 |
| 2.12 | ETM_MACRO_SERVO_OFF | 0x0C16 | 3094 | 2 |
| 2.13 | ETM_SET_ALARM | 0x0A17 | 2583 | 4 |
| 2.13 | ETM_MACRO_SET_ALARM | 0x0C17 | 3095 | 4 |
| 2.14 | ETM_SET_INPOS | 0x0A18 | 2584 | 4 |
| 2.14 | ETM_MACRO_SET_INPOS | 0x0C18 | 3096 | 4 |
| 2.15 | ETM_SET_FILTER | 0x0A19 | 2585 | 4 |
| 2.15 | ETM_MACRO_SET_FILTER | 0x0C19 | 3097 | 4 |
| 2.16 | ETM_VRING_ENABLE | 0x0A1A | 2586 | 4 |
| 2.16 | ETM_MACRO_VRING_ENABLE | 0x0C1A | 3098 | 4 |
| 2.16 | ETM_VRING_DISABLE | 0x0A1B | 2587 | 2 |
| 2.16 | ETM_MACRO_VRING_DISABLE | 0x0C1B | 3099 | 2 |
| 2.17 | ETM_AVTRI_ENABLE | 0x0A1C | 2588 | 2 |
| 2.17 | ETM_MACRO_AVTRI_ENABLE | 0x0C1C | 3100 | 2 |
| 2.17 | ETM_AVTRI_DISABLE | 0x0A1D | 2589 | 2 |
| 2.17 | ETM_MACRO_AVTRI_DISABLE | 0x0C1D | 3101 | 2 |
| 2.18 | ETM_EXD_MP | 0x0A1E | 2590 | 4 |
| 2.18 | ETM_EXD_FP | 0x0A1F | 2591 | 4 |
| 2.18 | ETM_EXD_CP | 0x0A20 | 2592 | 4 |
| 2.18 | ETM_EXD_DISABLE | 0x0A21 | 2593 | 2 |
| 2.19 | ETM_READ_bVAR | xxxxxx | xxxxx | |
| 2.19 | ETM_WRITE_bVAR | 0x0A23 | 2595 | 3 |
| 2.19 | ETM_READ_VAR | xxxxxx | xxxxx | |
| 2.19 | ETM_WRITE_VAR | 0x0A25 | 2597 | 5 |
| 2.20 | ETM_READ_MD | xxxxxx | xxxxx | |
| 2.20 | ETM_WRITE_MD | 0x0A27 | 2599 | 7 |
| | | | | |
| 3.1 | ETM_SET_LP | 0x0A28 | 2600 | 4 |
| 3.1 | ETM_MACRO_SET_LP | 0x0C28 | 3112 | 4 |
| 3.1 | ETM_GET_LP | xxxxxx | xxxxx | |
| 3.1 | ETM_MACRO_GET_LP | 0x0C29 | 3113 | 2 |
| 3.1 | ETM_GET_LP_4_AXIS | xxxxxx | xxxxx | |

| Section | Sub_Function | Code (Hex) | Code (Dec) | No. of registers |
|---|---|---|---|---|
| | | reg(0) | reg(0) | |
| 3.2 | ETM_SET_EP | 0x0A2A | 2602 | 4 |
| 3.2 | ETM_MACRO_SET_EP | 0x0C2A | 3114 | 4 |
| 3.2 | ETM_GET_EP | xxxxxx | xxxxx | |
| 3.2 | ETM_MACRO_GET_EP | 0x0C2B | 3115 | 2 |
| 3.2 | ETM_GET_EP_4_AXIS | xxxxxx | xxxxx | |
| 3.3 | ETM_GET_CV | xxxxxx | xxxxx | |
| 3.3 | ETM_GET_CV_4_AXIS | xxxxxx | xxxxx | |
| 3.4 | ETM_GET_CA | xxxxxx | xxxxx | |
| 3.4 | ETM_GET_CA_4_AXIS | xxxxxx | xxxxx | |
| 3.5 | ETM_MACRO_GET_DI | 0x0C2E | 3118 | 3 |
| 3.5 | ETM_GET_DI_ALL | xxxxxx | xxxxx | |
| 3.5 | ETM_MACRO_GET_DI_ALL | 0x0C31 | 3121 | 2 |
| 3.5 | ETM_GET_DI_ALL_4_AXIS | xxxxxx | xxxxx | |
| 3.6 | ETM_MACRO_GET_ERROR | 0x0C2F | 3119 | 1 |
| 3.6 | ETM_GET_ERROR_STATE | xxxxxx | xxxxx | |
| 3.6 | ETM_GET_ERROR_CODE | xxxxxx | xxxxx | |
| 3.6 | ETM_MACRO_GET_ERROR_CODE | 0x0C30 | 3120 | 2 |
| 3.6 | ETM_GET_ERROR_CODE_4_AXIS | xxxxxx | xxxxx | |
| 3.7 | i8094H_CHECK_RTC | xxxxxx | xxxxx | |
| 3.8 | ETM_GET_STOP_STATUS | xxxxxx | xxxxx | |
| 3.8 | ETM_GET_STOP_STATUS_4_AXIS | xxxxxx | xxxxx | |
| | | | | |
| 4.1 | ETM_MACRO_FRNET_IN | 0x0C32 | 3122 | 2 |
| 4.1 | ETM_MACRO_FRNET_READ | 0x0C34 | 3124 | 5 |
| 4.1 | ETM_FRNET_READ_SINGLE_DIO | xxxxxx | xxxxx | |
| 4.1 | ETM_FRNET_READ_GROUP_DIO | xxxxxx | xxxxx | |
| 4.1 | ETM_FRNET_READ_MULTI_GROUP_DIO | xxxxxx | xxxxx | |
| 4.2 | ETM_MACRO_FRNET_OUT | 0x0C33 | 3123 | 4 |
| 4.2 | ETM_MACRO_FRNET_WRITE | 0x0C35 | 3125 | 7 |
| 4.2 | ETM_FRNET_WRITE_SINGLE_DO | xxxxxx | xxxxx | |
| 4.2 | ETM_FRNET_WRITE_GROUP_DO | xxxxxx | xxxxx | |
| 4.2 | ETM_FRNET_WRITE_MULTI_GROUP_DO | xxxxxx | xxxxx | |

| Section | Sub_Function | Code (Hex) | Code (Dec) | No. of registers |
|---|---|---|---|---|
| | | reg(0) | reg(0) | |
| 4.3 | ETM_MACRO_FRNET_WAIT | 0x0C36 | 3126 | 7 |
| | | | | |
| 5.1 | ETM_SET_HV | 0x0A3C | 2620 | 4 |
| 5.1 | ETM_MACRO_SET_HV | 0x0C3C | 3132 | 4 |
| 5.2 | ETM_HOME _LIMIT | 0x0A3D | 2621 | 3 |
| 5.2 | ETM_MACRO_HOME _LIMIT | 0x0C3D | 3133 | 3 |
| 5.3 | ETM_SET_HOME_MODE | 0x0A3E | 2622 | 8 |
| 5.3 | ETM_MACRO_SET_HOME_MODE | 0x0C3E | 3134 | 8 |
| 5.4 | ETM_HOME _START | 0x0A3F | 2623 | 2 |
| 5.4 | ETM_MACRO_HOME _START | 0x0C3F | 3135 | 2 |
| 5.5 | i8094H_HOME _WAIT | xxxxxx | xxxxx | |
| | | | | |
| 6.1.1 | ETM_NORMAL_SPEED | 0x0A46 | 2630 | 3 |
| 6.1.1 | ETM_MACRO_NORMAL_SPEED | 0x0C46 | 3142 | 3 |
| 6.1.2 | ETM_SET_SV | 0x0A47 | 2631 | 4 |
| 6.1.2 | ETM_MACRO_SET_SV | 0x0C47 | 3143 | 4 |
| 6.1.3 | ETM_SET_V | 0x0A48 | 2632 | 4 |
| 6.1.3 | ETM_MACRO_SET_V | 0x0C48 | 3144 | 4 |
| 6.1.4 | ETM_SET_A | 0x0A49 | 2633 | 4 |
| 6.1.4 | ETM_MACRO_SET_A | 0x0C49 | 3145 | 4 |
| 6.1.5 | ETM_SET_D | 0x0A4A | 2634 | 4 |
| 6.1.5 | ETM_MACRO_SET_D | 0x0C4A | 3146 | 4 |
| 6.1.6 | ETM_SET_K | 0x0A4B | 2635 | 4 |
| 6.1.6 | ETM_MACRO_SET_K | 0x0C4B | 3147 | 4 |
| 6.1.7 | ETM_SET_L | 0x0A4C | 2636 | 4 |
| 6.1.7 | ETM_MACRO_SET_L | 0x0C4C | 3148 | 4 |
| 6.1.8 | ETM_SET_AO | 0x0A4D | 2637 | 4 |
| 6.1.8 | ETM_MACRO_SET_AO | 0x0C4D | 3149 | 4 |
| 6.1.9 | ETM_FIXED_MOVE | 0x0A4E | 2638 | 4 |
| 6.1.9 | ETM_MACRO_FIXED_MOVE | 0x0C4E | 3150 | 4 |
| 6.1.9 | ETM_SET_PULSE | 0x0A4F | 2639 | 4 |
| 6.1.9 | ETM_MACRO_SET_PULSE | 0x0C4F | 3151 | 4 |
| | | | | |

| Section | Sub_Function | Code (Hex) | Code (Dec) | No. of registers |
|---|---|---|---|---|
| | | reg(0) | reg(0) | |
| 6.1.10 | ETM_CONTINUE_MOVE | 0x0A50 | 2640 | 4 |
| 6.1.10 | ETM_MACRO_CONTINUE_MOVE | 0x0C50 | 3152 | 4 |
| | | | | |
| 6.2.1 | ETM_AXIS_ASSIGN | 0x0A5A | 2650 | 4 |
| 6.2.1 | ETM_MACRO_AXIS_ASSIGN | 0x0C5A | 3162 | 4 |
| 6.2.2 | ETM_VECTOR_SPEED | 0x0A5B | 2651 | 2 |
| 6.2.2 | ETM_MACRO_VECTOR_SPEED | 0x0C5B | 3163 | 2 |
| 6.2.3 | ETM_SET_VSV | 0x0A5C | 2652 | 3 |
| 6.2.3 | ETM_MACRO_SET_VSV | 0x0C5C | 3164 | 3 |
| 6.2.4 | ETM_SET_VV | 0x0A5D | 2653 | 3 |
| 6.2.4 | ETM_MACRO_SET_VV | 0x0C5D | 3165 | 3 |
| 6.2.5 | ETM_SET_VA | 0x0A5E | 2654 | 3 |
| 6.2.5 | ETM_MACRO_SET_VA | 0x0C5E | 3166 | 3 |
| 6.2.6 | ETM_SET_VD | 0x0A5F | 2655 | 3 |
| 6.2.6 | ETM_MACRO_SET_VD | 0x0C5F | 3167 | 3 |
| 6.2.7 | ETM_SET_VK | 0x0A60 | 2656 | 3 |
| 6.2.7 | ETM_MACRO_SET_VK | 0x0C60 | 3168 | 3 |
| 6.2.8 | ETM_SET_VL | 0x0A61 | 2657 | 3 |
| 6.2.8 | ETM_MACRO_SET_VL | 0x0C61 | 3169 | 3 |
| 6.2.9 | ETM_SET_VAO | 0x0A62 | 2658 | 3 |
| 6.2.9 | ETM_MACRO_SET_VAO | 0x0C62 | 3170 | 3 |
| 6.2.10 | ETM_LINE_2D | 0x0A63 | 2659 | 5 |
| 6.2.10 | ETM_MACRO_LINE_2D | 0x0C63 | 3171 | 5 |
| 6.2.11 | ETM_LINE_3D | 0x0A64 | 2660 | 7 |
| 6.2.11 | ETM_MACRO_LINE_3D | 0x0C64 | 3172 | 7 |
| 6.2.12 | ETM_ARC_CW | 0x0A65 | 2661 | 9 |
| 6.2.12 | ETM_MACRO_ARC_CW | 0x0C65 | 3173 | 9 |
| 6.2.12 | ETM_ARC_CCW | 0x0A67 | 2663 | 9 |
| 6.2.12 | ETM_MACRO_ARC_CCW | 0x0C67 | 3175 | 9 |
| 6.2.13 | ETM_CIRCLE_CW | 0x0A69 | 2665 | 5 |
| 6.2.13 | ETM_MACRO_CIRCLE_CW | 0x0C69 | 3177 | 5 |
| 6.2.13 | ETM_CIRCLE_CCW | 0x0A6A | 2666 | 5 |
| 6.2.13 | ETM_MACRO_CIRCLE_CCW | 0x0C6A | 3178 | 5 |

| Section | Sub_Function | Code (Hex) reg(0) | Code (Dec) reg(0) | No. of registers |
|---|---|---|---|---|
| 6.3.1 | ETM_SYNC_ACTION | 0x0A6E | 2670 | 14 |
| 6.3.1 | ETM_MACRO_SYNC_ACTION | 0x0C6E | 2670 | 14 |
| 6.3.1 | ETM_CLEAR_SYNC_ACTION | 0x0A95 | 2709 | 2 |
| 6.3.1 | ETM_MACRO_CLEAR_SYNC_ACTION | 0x0C95 | 3221 | 2 |
| 6.3.1 | ETM_SET_ACTIVATION_FACTORS | 0x0A96 | 2710 | 4 |
| 6.3.1 | ETM_MACRO_SET_ACTIVATION_FACTORS | 0x0C96 | 3222 | 4 |
| 6.3.1 | ETM_SET_ACTIVATION_AXIS | 0x0A97 | 2711 | 3 |
| 6.3.1 | ETM_MACRO_SET_ACTIVATION_AXIS | 0x0C97 | 3223 | 3 |
| 6.3.1 | ETM_SET_ACTION | 0x0A98 | 2712 | 11 |
| 6.3.1 | ETM_MACRO_SET_ACTION | 0x0C98 | 3224 | 11 |
| 6.3.2 | ETM_SET_COMPARE | 0x0A70 | 2672 | 6 |
| 6.3.2 | ETM_MACRO_SET_COMPARE | 0x0C70 | 3184 | 6 |
| 6.3.3 | ETM_GET_LATCH | xxxxxx | xxxxx | |
| 6.3.3 | ETM_MACRO_GET_LATCH | 0x0C71 | 3185 | 2 |
| 6.3.4 | ETM_SET_PRESET | 0x0A72 | 2674 | 4 |
| 6.3.4 | ETM_MACRO_SET_PRESET | 0x0C72 | 3186 | 4 |
| 6.3.5 | ETM_SET_OUT | 0x0A73 | 2675 | 4 |
| 6.3.5 | ETM_MACRO_SET_OUT | 0x0C73 | 3187 | 4 |
| 6.3.6 | ETM_ENABLE_INT | 0x0AAA | 2730 | 1 |
| 6.3.6 | ETM_MACRO_ENABLE_INT | 0x0CAA | 3242 | 1 |
| 6.3.6 | ETM_DISABLE_INT | 0x0AAB | 2731 | 1 |
| 6.3.6 | ETM_MACRO_DISABLE_INT | 0x0CAB | 3243 | 1 |
| 6.3.6 | ETM_INTFACTOR_ENABLE | 0x0AAC | 2732 | 4 |
| 6.3.6 | ETM_MACRO_INTFACTOR_ENABLE | 0x0CAC | 3244 | 4 |
| 6.3.6 | ETM_INTFACTOR_DISABLE | 0x0AAD | 2733 | 3 |
| 6.3.6 | ETM_MACRO_INTFACTOR_DISABLE | 0x0CAD | 3245 | 3 |
| | | | | |
| 6.4 | ETM_GET_TRIG_INTFACTOR | xxxxxx | xxxxx | |
| | | | | |
| 6.5 | ETM_GET_MP_DOWNLOAD_STATUS | xxxxxx | xxxxx | |
| | | | | |
| 6.6 | ETM_GET_USER_RINT | xxxxxx | xxxxx | |
| | | | | |
| 6.7 | ETM_GET_DEVICE_STATE | xxxxxx | xxxxx | |

| Section | Sub_Function | code (H)<br>reg(0) | code (D)<br>reg(0) | No. of registers |
|---|---|---|---|---|
| | | | | |
| 6.8.1 | ETM_RECTANGLE | 0x0A78 | 2680 | 20 |
| 6.8.1 | ETM_MACRO_RECTANGLE | 0x0C78 | 3192 | 20 |
| 6.8.2 | ETM_LINE_2D_INITIAL | 0x0A7C | 2684 | 9 |
| 6.8.2 | ETM_MACRO_LINE_2D_INITIAL | 0x0C7C | 3196 | 9 |
| 6.8.2 | ETM_LINE_2D_CONTINUE | 0x0A7E | 2686 | 6 |
| 6.8.2 | ETM_MACRO_LINE_2D_CONTINUE | 0x0C7E | 3198 | 6 |
| 6.8.3 | ETM_LINE_3D_INITIAL | 0x0A7F | 2687 | 10 |
| 6.8.3 | ETM_MACRO_LINE_3D_INITIAL | 0x0C7F | 3199 | 10 |
| 6.8.3 | ETM_LINE_3D_CONTINUE | 0x0A81 | 2689 | 8 |
| 6.8.3 | ETM_MACRO_LINE_3D_CONTINUE | 0x0C81 | 3201 | 8 |
| 6.8.4 | ETM_MIX_2D_INITIAL | 0x0A82 | 2690 | 10 |
| 6.8.4 | ETM_MACRO_MIX_2D_INITIAL | 0x0C82 | 3202 | 10 |
| 6.8.4 | ETM_MIX_2D_CONTINUE | 0x0A84 | 2692 | 11 |
| 6.8.4 | ETM_MACRO_MIX_2D_CONTINUE | 0x0C84 | 3204 | 11 |
| 6.8.5 | ETM_CONTINUE_INTP | 0x0A86 | 2694 | 13 |
| 6.8.5 | i8094H_CONTINUE_INTP_ARRAY | xxxxxx | xxxxx | |
| 6.8.6 | ETM_HELIX_3D | 0x0A88 | 2696 | 15 |
| 6.8.6 | ETM_MACRO_HELIX_3D | 0x0C88 | 3208 | 15 |
| 6.8.7 | ETM_RATIO_INITIAL | 0x0A8B | 2699 | 11 |
| 6.8.7 | ETM_MACRO_RATIO_INITIAL | 0x0C8B | 3211 | 11 |
| 6.8.7 | ETM_RATIO_2D | 0x0A8D | 2701 | 5 |
| 6.8.7 | ETM_MACRO_RATIO_2D | 0x0C8D | 3213 | 5 |
| 6.8.8 | ETM_LINE_SCAN | 0x0A90 | 2704 | 7 |
| 6.8.8 | ETM_LINE_SCAN_START | 0x0A91 | 2705 | 5 |
| 6.8.8 | ETM_LINE_SCAN_OFFSET2 | 0x0A93 | 2707 | 3+Param_Lengh |
| 6.8.8 | ETM_GET_LINE_SCAN_DONE | xxxxxx | xxxxx | |
| | | | | |
| | | | | |
| 6.9.1 | ETM_DRV_HOLD | 0x0AB4 | 2740 | 2 |
| 6.9.1 | ETM_MACRO_DRV_HOLD | 0x0CB4 | 3252 | 2 |
| 6.9.2 | ETM_DRV_START | 0x0AB5 | 2741 | 2 |
| 6.9.2 | ETM_MACRO_DRV_START | 0x0CB5 | 3253 | 2 |

| Section | Sub_Function | Code (Hex) | Code (Dec) | No. of registers |
|---|---|---|---|---|
| | | reg(0) | reg(0) | |
| 6.9.3 | ETM_STOP_SLOWLY | 0x0AB7 | 2743 | 2 |
| 6.9.3 | ETM_MACRO_STOP_SLOWLY | 0x0CB7 | 3255 | 2 |
| 6.9.3 | ETM_STOP_SUDDENLY | 0x0AB8 | 2744 | 2 |
| 6.9.3 | ETM_MACRO_STOP_SUDDENLY | 0x0CB8 | 3256 | 2 |
| 6.9.3 | ETM_VSTOP_SLOWLY | 0x0AB9 | 2745 | 1 |
| 6.9.3 | ETM_MACRO_VSTOP_SLOWLY | 0x0CB9 | 3257 | 1 |
| 6.9.3 | ETM_VSTOP_SUDDENLY | 0x0ABA | 2746 | 1 |
| 6.9.3 | ETM_MACRO_VSTOP_SUDDENLY | 0x0CBA | 3258 | 1 |
| 6.9.4 | ETM_CLEAR_STOP | 0x0ABB | 2747 | 2 |
| 6.9.4 | ETM_MACRO_CLEAR_STOP | 0x0CBB | 3259 | 2 |
| 6.9.4 | ETM_CLEAR_VSTOP | 0x0A09 | 2569 | 1 |
| 6.9.4 | ETM_MACRO_CLEAR_VSTOP | 0x0C09 | 3081 | 1 |
| 6.9.5 | ETM_INTP_END | 0x0ABC | 2748 | 2 |
| 6.9.5 | ETM_MACRO_INTP_END | 0x0CBC | 3260 | 2 |
| 6.9.6 | ETM_EMERGENCY_STOP | 0x0A04 | 2564 | 2 |
| 6.9.7 | ETM_CLEAR_EMERGENCY_STOP | 0x0A05 | 2565 | 1 |
| | | | | |
| 7.1 | ETM_LOAD_INITIAL | 0x0AC8 | 2760 | 1 |
| 7.2.1 | ETM_MP_CREATE | 0x0AC9 | 2761 | 2 |
| 7.2.1 | ETM_MACRO_MP_CLOSE | 0x0CCA | 3274 | 1 |
| 7.2.2 | ETM_MP_CALL | 0x0ACB | 2763 | 2 |
| 7.2.2 | ETM_MACRO_MP_CALL | 0x0CCB | 3275 | 2 |
| 7.3.1 | ETM_MP_ISR_CREATE | 0x0ACD | 2765 | 2 |
| 7.3.2 | ETM_MACRO_MP_ISR_CLOSE | 0x0CCE | 3278 | 1 |
| 7.3.3 | ETM_MP_ISR_CALL | 0x0ACF | 2767 | 2 |
| 7.4.1 | ETM_MACRO_SET_VAR | 0x0CD2 | 3282 | 5 |
| 7.4.2 | ETM_MACRO_SET_RVAR | 0x0CD3 | 3283 | 3 |
| 7.5 | ETM_MACRO_VAR_CALCULATE | 0x0CD4 | 3284 | 8 |
| 7.6.1 | ETM_MACRO_FOR | 0x0CD5 | 3285 | 3 |
| 7.6.2 | ETM_MACRO_NEXT | 0x0CD6 | 3286 | 1 |
| 7.6.3 | ETM_MACRO_EXIT_FOR | 0x0CDE | 3294 | 1 |
| 7.7.1 | ETM_MACRO_IF | 0x0CD7 | 3287 | 6 |
| 7.7.2 | ETM_MACRO_ELSE | 0x0CD8 | 3288 | 1 |
| 7.7.3 | ETM_MACRO_END _IF | 0x0CD9 | 3289 | 1 |
| 7.8.1 | ETM_MACRO_GOTO | 0x0CDF | 3295 | 2 |

| Section | Sub_Function | Code (Hex) | Code (Dec) | No. of registers |
|---|---|---|---|---|
| | | reg(0) | reg(0) | |
| 7.8.2 | ETM_MACRO_LABEL | 0x0CE1 | 3297 | 2 |
| 7.9 | ETM _MACRO_TIMER | 0x0CDA | 3290 | 3 |
| 7.10 | ETM_STOP_WAIT | 0x0ADB | 2779 | 2 |
| 7.10 | ETM_MACRO_STOP_WAIT | 0x0CDB | 3291 | 2 |
| 7.11 | ETM_MACRO_SET_RINT | 0x0CDC | 3292 | 1 |
| 7.12 | ETM_MACRO_EXIT_MACRO | 0x0CDD | 3293 | 1 |
| 7.13 | ETM_MP_TERMINATE | 0x0AE2 | 2786 | 1 |
| 7.13 | ETM_MACRO_MP_TERMINATE | 0x0CE2 | 3298 | 1 |

# D. MODBUS Coil Table

The coil table can be accessed with the following Modbus functions:
- Read Coils (FC 01)
- Write Single Coil (FC 05)
- Write Multiple Coils (FC 15)

The table addresses defined below is zero-based. For some PLC or HMI, the addresses maybe show on their documents as 0xxxx, x is digital number. For example, 00001 to 09999. The first '0' represents the table is defined for coils; and the following xxxx is the address of a coil.

In PLC, the first coil defined at address 00001 is the same as the first register at address 0 shown in the following table. Most PLC systems which addresses do not start from zero can be referred as one-based system. The following tables will only show zero-based addresses. When PLC or HMI users configure their coil numbers, they have to consider coil number mapping problem carefully.

## D.1 FRnet Digital Output

The starting address is 0 (zero-based). Please use FC=05, 15 to write the FRnet DO status:

(1) FC=05. It can be used to write single point FRnet DO (0~127).

(2) FC=15. It can be used to write multi-point FRnet DO. These DO do not have to be limited to the same group. For example: the user can choose to write DO points from 5~45 (total 41 points).

**Appendix table 21: FRnet Digital Output (PLC addresses: 00001 to 00128)**

| FRnet Digital Output | | | | |
|---|---|---|---|---|
| Group | Channel | Address (zero-based) | Type | Comment |
| 0 | 0 | 0 | R/W | |
| | 1 | 1 | R/W | |
| | 2 | 2 | R/W | |
| | 3 | 3 | R/W | |
| | 4 | 4 | R/W | |
| | 5 | 5 | R/W | |
| | 6 | 6 | R/W | |
| | 7 | 7 | R/W | |
| | 8 | 8 | R/W | |
| | 9 | 9 | R/W | |
| | 10 | 10 | R/W | |
| | 11 | 11 | R/W | |
| | 12 | 12 | R/W | |
| | 13 | 13 | R/W | |
| | 14 | 14 | R/W | |
| | 15 | 15 | R/W | |
| 1 | 0 | 16 | R/W | |
| | 1 | 17 | R/W | |
| | 2 | 18 | R/W | |
| | 3 | 19 | R/W | |
| | 4 | 20 | R/W | |
| | 5 | 21 | R/W | |
| | 6 | 22 | R/W | |
| | 7 | 23 | R/W | |

| | | | | |
|---|---|---|---|---|
| | 8 | 24 | R/W | |
| | 9 | 25 | R/W | |
| | 10 | 26 | R/W | |
| | 11 | 27 | R/W | |
| | 12 | 28 | R/W | |
| | 13 | 29 | R/W | |
| | 14 | 30 | R/W | |
| | 15 | 31 | R/W | |
| 2 | 0 | 32 | R/W | |
| | 1 | 33 | R/W | |
| | 2 | 34 | R/W | |
| | 3 | 35 | R/W | |
| | 4 | 36 | R/W | |
| | 5 | 37 | R/W | |
| | 6 | 38 | R/W | |
| | 7 | 39 | R/W | |
| | 8 | 40 | R/W | |
| | 9 | 41 | R/W | |
| | 10 | 42 | R/W | |
| | 11 | 43 | R/W | |
| | 12 | 44 | R/W | |
| | 13 | 45 | R/W | |
| | 14 | 46 | R/W | |
| | 15 | 47 | R/W | |
| 3 | 0 | 48 | R/W | |
| | 1 | 49 | R/W | |
| | 2 | 50 | R/W | |
| | 3 | 51 | R/W | |
| | 4 | 52 | R/W | |
| | 5 | 53 | R/W | |
| | 6 | 54 | R/W | |
| | 7 | 55 | R/W | |
| | 8 | 56 | R/W | |
| | 9 | 57 | R/W | |
| | 10 | 58 | R/W | |
| | 11 | 59 | R/W | |
| | 12 | 60 | R/W | |

| | | | | |
|---|---|---|---|---|
| | 13 | 61 | R/W | |
| | 14 | 62 | R/W | |
| | 15 | 63 | R/W | |
| 4 | 0 | 64 | R/W | |
| | 1 | 65 | R/W | |
| | 2 | 66 | R/W | |
| | 3 | 67 | R/W | |
| | 4 | 68 | R/W | |
| | 5 | 69 | R/W | |
| | 6 | 70 | R/W | |
| | 7 | 71 | R/W | |
| | 8 | 72 | R/W | |
| | 9 | 73 | R/W | |
| | 10 | 74 | R/W | |
| | 11 | 75 | R/W | |
| | 12 | 76 | R/W | |
| | 13 | 77 | R/W | |
| | 14 | 78 | R/W | |
| | 15 | 79 | R/W | |
| 5 | 0 | 80 | R/W | |
| | 1 | 81 | R/W | |
| | 2 | 82 | R/W | |
| | 3 | 83 | R/W | |
| | 4 | 84 | R/W | |
| | 5 | 85 | R/W | |
| | 6 | 86 | R/W | |
| | 7 | 87 | R/W | |
| | 8 | 88 | R/W | |
| | 9 | 89 | R/W | |
| | 10 | 90 | R/W | |
| | 11 | 91 | R/W | |
| | 12 | 92 | R/W | |
| | 13 | 93 | R/W | |
| | 14 | 94 | R/W | |
| | 15 | 95 | R/W | |
| 6 | 0 | 96 | R/W | |
| | 1 | 97 | R/W | |

| | | | | |
|---|---|---|---|---|
| | 2 | 98 | R/W | |
| | 3 | 99 | R/W | |
| | 4 | 100 | R/W | |
| | 5 | 101 | R/W | |
| | 6 | 102 | R/W | |
| | 7 | 103 | R/W | |
| | 8 | 104 | R/W | |
| | 9 | 105 | R/W | |
| | 10 | 106 | R/W | |
| | 11 | 107 | R/W | |
| | 12 | 108 | R/W | |
| | 13 | 109 | R/W | |
| | 14 | 110 | R/W | |
| | 15 | 111 | R/W | |
| 7 | 0 | 112 | R/W | |
| | 1 | 113 | R/W | |
| | 2 | 114 | R/W | |
| | 3 | 115 | R/W | |
| | 4 | 116 | R/W | |
| | 5 | 117 | R/W | |
| | 6 | 118 | R/W | |
| | 7 | 119 | R/W | |
| | 8 | 120 | R/W | |
| | 9 | 121 | R/W | |
| | 10 | 122 | R/W | |
| | 11 | 123 | R/W | |
| | 12 | 124 | R/W | |
| | 13 | 125 | R/W | |
| | 14 | 126 | R/W | |
| | 15 | 127 | R/W | |

## D.2 Servo On/Off

**Appendix table 22: Servo On/Off (PLC addresses: 00129 to 00132)**

| Servo State | | | |
|---|---|---|---|
| Axis | Address (zero based) | Type | Comment |
| X | 128 | R | 1: Servo on<br>0: Servo off |
| Y | 129 | R | 1: Servo on<br>0: Servo off |
| Z | 130 | R | 1: Servo on<br>0: Servo off |
| U | 131 | R | 1: Servo on<br>0: Servo off |

# E. MODBUS Discrete Input Table

The discrete input table can be accessed by the following Modbus functions:
– Read discrete input (FC 02)

The table addresses defined below is zero-based. For some PLC or HMI, the addresses maybe show on their documents as 1xxxx, x is digital number. For example, 10001 to 19999. The first '1' represents the table is defined for discrete input; and the following xxxx is the address of a discrete input.

In PLC, the first discrete input defined at address 10001 is the same as the first register at address 0 shown in the following table. Most PLC systems which addresses do not start from zero can be referred as one-based system. The following tables will only show zero-based addresses. When PLC or HMI users configure their discrete input numbers, they have to consider discrete input number mapping problem carefully.

## E.1 FRnet Digital Input

**Appendix table 23: FRnet Digital Inputs (PLC addresses: 10001 to 10128)**

| FRnet Digital Input | | | | |
|---|---|---|---|---|
| Group | Channel | Address (zero based) | Type | Comment |
| 8 | 0 | 0 | R | |
| | 1 | 1 | R | |
| | 2 | 2 | R | |
| | 3 | 3 | R | |
| | 4 | 4 | R | |
| | 5 | 5 | R | |
| | 6 | 6 | R | |
| | 7 | 7 | R | |
| | 8 | 8 | R | |
| | 9 | 9 | R | |
| | 10 | 10 | R | |
| | 11 | 11 | R | |
| | 12 | 12 | R | |
| | 13 | 13 | R | |
| | 14 | 14 | R | |
| | 15 | 15 | R | |
| 9 | 0 | 16 | R | |
| | 1 | 17 | R | |
| | 2 | 18 | R | |
| | 3 | 19 | R | |
| | 4 | 20 | R | |
| | 5 | 21 | R | |
| | 6 | 22 | R | |
| | 7 | 23 | R | |
| | 8 | 24 | R | |
| | 9 | 25 | R | |
| | 10 | 26 | R | |
| | 11 | 27 | R | |
| | 12 | 28 | R | |

| | | | | |
|---|---|---|---|---|
| | 13 | 29 | R | |
| | 14 | 30 | R | |
| | 15 | 31 | R | |
| 10 | 0 | 32 | R | |
| | 1 | 33 | R | |
| | 2 | 34 | R | |
| | 3 | 35 | R | |
| | 4 | 36 | R | |
| | 5 | 37 | R | |
| | 6 | 38 | R | |
| | 7 | 39 | R | |
| | 8 | 40 | R | |
| | 9 | 41 | R | |
| | 10 | 42 | R | |
| | 11 | 43 | R | |
| | 12 | 44 | R | |
| | 13 | 45 | R | |
| | 14 | 46 | R | |
| | 15 | 47 | R | |
| 11 | 0 | 48 | R | |
| | 1 | 49 | R | |
| | 2 | 50 | R | |
| | 3 | 51 | R | |
| | 4 | 52 | R | |
| | 5 | 53 | R | |
| | 6 | 54 | R | |
| | 7 | 55 | R | |
| | 8 | 56 | R | |
| | 9 | 57 | R | |
| | 10 | 58 | R | |
| | 11 | 59 | R | |
| | 12 | 60 | R | |
| | 13 | 61 | R | |
| | 14 | 62 | R | |
| | 15 | 63 | R | |

| | | | | |
|---|---|---|---|---|
| 12 | 0 | 64 | R | |
| | 1 | 65 | R | |
| | 2 | 66 | R | |
| | 3 | 67 | R | |
| | 4 | 68 | R | |
| | 5 | 69 | R | |
| | 6 | 70 | R | |
| | 7 | 71 | R | |
| | 8 | 72 | R | |
| | 9 | 73 | R | |
| | 10 | 74 | R | |
| | 11 | 75 | R | |
| | 12 | 76 | R | |
| | 13 | 77 | R | |
| | 14 | 78 | R | |
| | 15 | 79 | R | |
| 13 | 0 | 80 | R | |
| | 1 | 81 | R | |
| | 2 | 82 | R | |
| | 3 | 83 | R | |
| | 4 | 84 | R | |
| | 5 | 85 | R | |
| | 6 | 86 | R | |
| | 7 | 87 | R | |
| | 8 | 88 | R | |
| | 9 | 89 | R | |
| | 10 | 90 | R | |
| | 11 | 91 | R | |
| | 12 | 92 | R | |
| | 13 | 93 | R | |
| | 14 | 94 | R | |
| | 15 | 95 | R | |
| 14 | 0 | 96 | R | |
| | 1 | 97 | R | |
| | 2 | 98 | R | |

| | | | | |
|---|---|---|---|---|
| | 3 | 99 | R | |
| | 4 | 100 | R | |
| | 5 | 101 | R | |
| | 6 | 102 | R | |
| | 7 | 103 | R | |
| | 8 | 104 | R | |
| | 9 | 105 | R | |
| | 10 | 106 | R | |
| | 11 | 107 | R | |
| | 12 | 108 | R | |
| | 13 | 109 | R | |
| | 14 | 110 | R | |
| | 15 | 111 | R | |
| 15 | 0 | 112 | R | |
| | 1 | 113 | R | |
| | 2 | 114 | R | |
| | 3 | 115 | R | |
| | 4 | 116 | R | |
| | 5 | 117 | R | |
| | 6 | 118 | R | |
| | 7 | 119 | R | |
| | 8 | 120 | R | |
| | 9 | 121 | R | |
| | 10 | 122 | R | |
| | 11 | 123 | R | |
| | 12 | 124 | R | |
| | 13 | 125 | R | |
| | 14 | 126 | R | |
| | 15 | 127 | R | |

## E.2 DI or Status of Control Board

**Appendix table 24: DI or Status of Control Board (PLC addresses: 10129 to 10192)**

| Read DI or Status of Control Board | | | | |
|---|---|---|---|---|
| Axis | DI | Address (zero based) | Type | Comment |
| X | DRIVING | 128 | R | The driving state of AXIS-X:<br>1: driving, 0: stop |
| | LIMIT+ | 129 | R | The hardware limit (LMT+) state of AXIS-X:<br>0: off, 1: on |
| | LIMIT- | 130 | R | The hardware limit (LMT-) state of AXIS-X:<br>0: off, 1: on |
| | EMERGENCY | 131 | R | The emergency (EMG) state of AXIS-X:<br>0: off, 1: on |
| | ALARM | 132 | R | The ALARM state of AXIS-X:<br>(please enable Alarm with ETM_SET_ALARM()<br>0: off, 1: on |
| | HOME | 133 | R | The HOME/IN1 state of AXIS-X:<br>0: on, 1: off |
| | NHOME | 134 | R | The NHOME/IN0 state of AXIS-X:<br>0: on, 1: off |
| | IN3 | 135 | R | The IN3 state of AXIS-X:<br>0: on, 1: off |
| | INPOS | 136 | R | The INPOS or Servo Ready state of AXIS-X:<br>0: on, 1: off |
| | INDEX | 137 | R | The Z-phase/IN2 state of AXIS-X:<br>0: on, 1: off |
| | | 138 | | undefined |
| | | 139 | | |
| | | 140 | | |
| | | 141 | | |
| | | 142 | | |
| | | 143 | | |
| Y | DRIVING | 144 | R | The driving state of AXIS-Y:<br>1: driving, 0: stop |
| | LIMIT+ | 145 | R | The hardware limit (LMT+) state of AXIS-Y:<br>0: off, 1: on |
| | LIMIT- | 146 | R | The hardware limit (LMT-) state of AXIS-Y:<br>0: off, 1: on |
| | EMERGENCY | 147 | R | The emergency (EMG) state of AXIS-Y:<br>0: off, 1: on |

| | | | | |
|---|---|---|---|---|
| | ALARM | 148 | R | The ALARM state of AXIS- Y:<br>(please enable Alarm with ETM_SET_ALARM()<br>0: off, 1: on |
| | HOME | 149 | R | The HOME/IN1 state of AXIS- Y:<br>0: on, 1: off |
| | NHOME | 150 | R | The NHOME/IN0 state of AXIS- Y:<br>0: on, 1: off |
| | IN3 | 151 | R | The IN3 state of AXIS- Y:<br>0: on, 1: off |
| | INPOS | 152 | R | The INPOS or Servo Ready state of AXIS-X:<br>0: on, 1: off |
| | INDEX | 153 | R | The Z-phase/IN2 state of AXIS- Y:<br>0: on, 1: off |
| | | 154 | | undefined |
| | | 155 | | |
| | | 156 | | |
| | | 157 | | |
| | | 158 | | |
| | | 159 | | |
| Z | DRIVING | 160 | R | The driving state of AXIS-Z:<br>1: driving, 0: stop |
| | LIMIT+ | 161 | R | The hardware limit (LMT+) state of AXIS-Z:<br>0: off, 1: on |
| | LIMIT- | 162 | R | The hardware limit (LMT-) state of AXIS-Z:<br>0: off, 1: on |
| | EMERGENCY | 163 | R | The emergency (EMG) state of AXIS-Z:<br>0: off, 1: on |
| | ALARM | 164 | R | The ALARM state of AXIS-Z:<br>(please enable Alarm with ETM_SET_ALARM()<br>0: off, 1: on |
| | HOME | 165 | R | The HOME/IN1 state of AXIS-Z:<br>0: on, 1: off |
| | NHOME | 166 | R | The NHOME/IN0 state of AXIS-Z:<br>0: on, 1: off |
| | IN3 | 167 | R | The IN3 state of AXIS-Z:<br>0: on, 1: off |
| | INPOS | 168 | R | The INPOS or Servo Ready state of AXIS-Z:<br>0: on, 1: off |
| | INDEX | 169 | R | The Z-phase/IN2 state of AXIS-Z:<br>0: on, 1: off |
| | | 170 | | undefined |

| | | | | |
|---|---|---|---|---|
| | | 171 | | |
| | | 172 | | |
| | | 173 | | |
| | | 174 | | |
| | | 175 | | |
| U | DRIVING | 176 | R | The driving state of AXIS-U:<br>1: driving, 0: stop |
| | LIMIT+ | 177 | R | The hardware limit (LMT+) state of AXIS-U:<br>0: off, 1: on |
| | LIMIT- | 178 | R | The hardware limit (LMT-) state of AXIS-U:<br>0: off, 1: on |
| | EMERGENCY | 179 | R | The emergency (EMG) state of AXIS-U:<br>0: off, 1: on |
| | ALARM | 180 | R | The ALARM state of AXIS-U:<br>(please enable Alarm with ETM_SET_ALARM()<br>0: off, 1: on |
| | HOME | 181 | R | The HOME/IN1 state of AXIS-U:<br>0: on, 1: off |
| | NHOME | 182 | R | The NHOME/IN0 state of AXIS-U:<br>0: on, 1: off |
| | IN3 | 183 | R | The IN3 state of AXIS-U:<br>0: on, 1: off |
| | INPOS | 184 | R | The INPOS or Servo Ready state of AXIS-U:<br>0: on, 1: off |
| | INDEX | 185 | R | The Z-phase/IN2 state of AXIS-U:<br>0: on, 1: off |
| | | 186 | | undefined |
| | | 187 | | |
| | | 188 | | |
| | | 189 | | |
| | | 190 | | |
| | | 191 | | |
| | | | | |

## E.3 Error Stop States

**Appendix table 25: Error Stop States (PLC addresses: 10193 to 10256)**

| Read Motion Error States | | | | |
|---|---|---|---|---|
| Axis | Cause of stop | Modbus Address | Type | Comment |
| X | SOFT LIMIT+ | 192 | R | Forward **software** limit of AXIS-X has been encountered |
| | SOFT LIMIT- | 193 | R | Reverse **software** limit of AXIS-X has been encountered |
| | LIMIT+ | 194 | R | Forward **hardware** limit of AXIS-X has been encountered |
| | LIMIT- | 195 | R | Reverse **hardware** limit of AXIS-X has been encountered |
| | ALARM | 196 | R | The ALARM state of AXIS-X has been activated |
| | EMERGENCY | 197 | R | The emergency state of AXIS-X has been activated |
| | Reserved | 198 | R | Not defined |
| | HOME | 199 | R | Z phase and HOME position of AXIS-X have been reached |
| | Stop Command | 200 | R | Motion has been stopped by<br>STOP_SLOWLY<br>STOP_SUDDENLY<br>VSTOP_SLOWLY<br>VSTOP_SUDDENLY |
| | | 201 | | undefined |
| | | 202 | | |
| | | 203 | | |
| | | 204 | | |
| | | 205 | | |
| | | 206 | | |
| | | 207 | | |
| Y | SOFT LIMIT+ | 208 | R | Forward **software** limit of AXIS-Y has been encountered |
| | SOFT LIMIT- | 209 | R | Reverse **software** limit of AXIS-Y has been encountered |
| | LIMIT+ | 210 | R | Forward **hardware** limit of AXIS-Y has been encountered |

| | | | | |
|---|---|---|---|---|
| | LIMIT- | 211 | R | Reverse **hardware** limit of AXIS-Y has been encountered |
| | ALARM | 212 | R | The ALARM state of AXIS-Y has been activated |
| | EMERGENCY | 213 | R | The emergency state of AXIS-Y has been activated |
| | Reserved | 214 | R | Not defined |
| | HOME | 215 | R | Z phase and HOME position of AXIS-Y have been reached |
| | Stop Command | 216 | R | Motion has been stopped by<br>STOP_SLOWLY<br>STOP_SUDDENLY<br>VSTOP_SLOWLY<br>VSTOP_SUDDENLY |
| | | 217 | | undefined |
| | | 218 | | |
| | | 219 | | |
| | | 220 | | |
| | | 221 | | |
| | | 222 | | |
| | | 223 | | |
| Z | SOFT LIMIT+ | 224 | R | Forward **software** limit of AXIS-Z has been encountered |
| | SOFT LIMIT- | 225 | R | Reverse **software** limit of AXIS-Z has been encountered |
| | LIMIT+ | 226 | R | Forward **hardware** limit of AXIS-Z has been encountered |
| | LIMIT- | 227 | R | Reverse **hardware** limit of AXIS-Z has been encountered |
| | ALARM | 228 | R | The ALARM state of AXIS-Z has been activated |
| | EMERGENCY | 229 | R | The emergency state of AXIS-Z has been activated |
| | Reserved | 230 | R | Not defined |
| | HOME | 231 | R | Z phase and HOME position of AXIS-Z have been reached |
| | Stop Command | 232 | R | Motion has been stopped by<br>STOP_SLOWLY<br>STOP_SUDDENLY<br>VSTOP_SLOWLY<br>VSTOP_SUDDENLY |

| | | | | |
|---|---|---|---|---|
| | | 233 | | undefined |
| | | 234 | | |
| | | 235 | | |
| | | 236 | | |
| | | 237 | | |
| | | 238 | | |
| | | 239 | | |
| U | SOFT LIMIT+ | 240 | R | Forward **software** limit of AXIS-U has been encountered |
| | SOFT LIMIT- | 241 | R | Reverse **software** limit of AXIS-U has been encountered |
| | LIMIT+ | 242 | R | Forward **hardware** limit of AXIS-U has been encountered |
| | LIMIT- | 243 | R | Reverse **hardware** limit of AXIS-U has been encountered |
| | ALARM | 244 | R | The ALARM state of AXIS-U has been activated |
| | EMERGENCY | 245 | R | The emergency state of AXIS-U has been activated |
| | Reserved | 246 | R | Not defined |
| | HOME | 247 | R | Z phase and HOME position of AXIS-U have been reached |
| | Stop Command | 248 | R | Motion has been stopped by<br>STOP_SLOWLY<br>STOP_SUDDENLY<br>VSTOP_SLOWLY<br>VSTOP_SUDDENLY |
| | | 249 | | undefined |
| | | 250 | | |
| | | 251 | | |
| | | 252 | | |
| | | 253 | | |
| | | 254 | | |
| | | 255 | | |

# F. ET-M8194H LED Description


**LED description:**

| LED | Status | Description |
|---|---|---|
| **Sys** | On | Device is switched on and firmware is running. |
| | Flashing | Device is switched on and firmware is not running. |
| | Off | Device is switched off. |
| **Tx** | Flashing | Data is transmitted by the ET-M8194H via RS-232. |
| | Off | No data is sent by the ET-M8194H via RS-232. |
| **Rx** | Flashing | The device is receiving data via RS-232. |
| | Off | No data is being received. |
| **NET** | On | Device is connected to Ethernet. |
| | Flashing | Data is being transmitted via Ethernet. |
| | Off | Device is not connected to the Ethernet. |
| **MOD** | On | Module i-8094H is plugged into ET-M8194H device. |
| | Flashing | A module different than i-8094H is plugged into ET-M8194H device. |
| | Off | No module is plugged into the ET-M8194H device. |

About the LED description on the i-8094H module: P is power indicator, A is FRnet indicator, and D is pulse output indicator. Other detals, including connectors and wiring daughter board definition please refer to the Quick – Start manual for the i-8094H.

## G. Lock_IP Setting

**Users can manage the Ethernet connection with Lock_IP settings. Three restrictions can be applied to ET-M8194H by Lock_IP settings:**

**1. [ Unique IP ] :**

**If only IP = 192.168.0.100 is allowed, assign the identical value to both IPStart and IPEnd.**

```
char IPStart[20] = “192.168.0.100”;
char IPEnd[20] = “192.168.0.100”;
ETM_Lock_IP(IPStart, IPEnd);
```

**2. [ Contiguous IP (less then 256 IPs) ] :**

**If the IP connections from 192.168.0.100 to 192.168.0.200 are allowed, assign the IPStart and IPEnd parameters as follows:**

```
char IPStart[20] = “192.168.0.100”;
char IPEnd[20] = “192.168.0.200”;
ETM_Lock_IP(IPStart, IPEnd);
```

**3. [ IP Segment ] :**

**(A) If the connections from 192.168.0.0 to 192.168.0.255 are allowed (i.e. connections from 192.168.0.xxx are allowed), assign the IPStart and IPEnd parameters as follows:**

```
char IPStart[20] = “192.168.0.1”;
char IPEnd[20] = “255.255.255.0”;       // MASK
ETM_Lock_IP(IPStart, IPEnd);
```

**(B) If the connections from 192.168.0.xxx to 192.168.1.xxx are allowed, assign the IPStart and IPEnd parameters as follows:**

```
char IPStart[20] = “192.168.0.100”;
char IPEnd[20] = “255.255.254.0”;  // MASK
ETM_Lock_IP(IPStart, IPEnd);
```

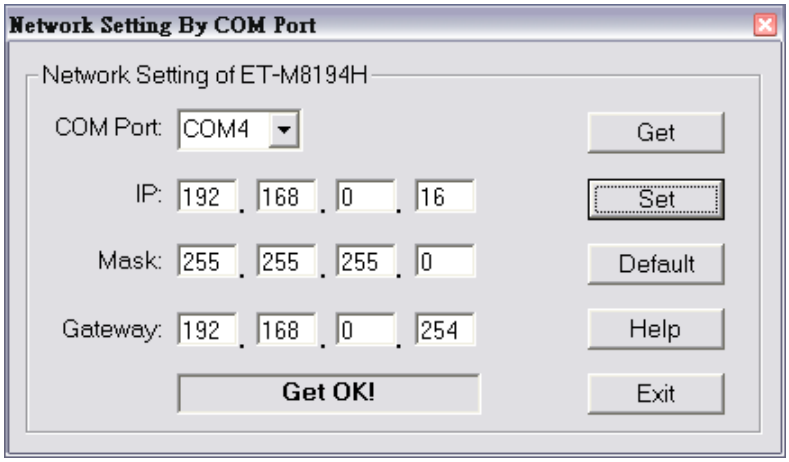
**(C) If the IPEnd is set to “255.255.255.255”, only the IP that assigned to IPStart is allowed.**

**[ Note ]**
**After rebooting ET-M8194H, the configuration of “Lock IP” will be reset, too. Please configure with ETM_Lock_IP() again to enable Lock_IP feature.**

# H. IP Configuration

## Method 1 – Setting via RS-232

**Execute the EzMove Utility and start the "Network Settings By COM Port" dialog. ([Setting] - [ET-M8194H Setting] – [By COM Port] – [Network])**



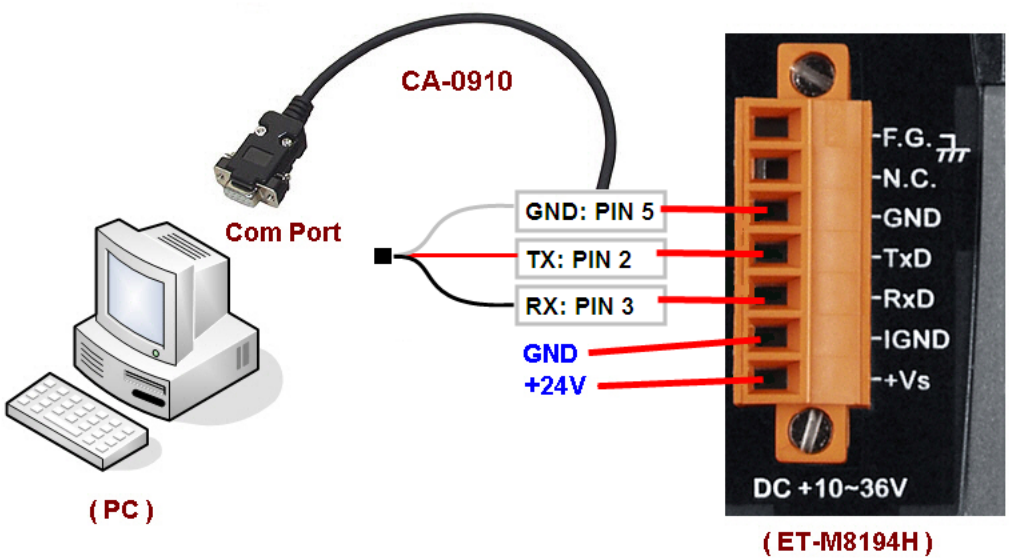

**Click Default to show the default IP configuration of ET-M8194H:**

**IP: 192.168.0.16**
**Mask: 255.255.255.0**
**Gateway: 192.168.0.254**

**The procedures to read/modify the IP configuration are:**

**1. Shut down the ET-M8194H.**

**2. Connect to ET-M8194H with the associated RS-232 cable (CA-0910). The Tx, Rx and GND pins of CA-0910 are connected to the Rx, Tx and GND ports of ET-M8194H. Please connect the other end (9-pin, D-sub connector) to the normal COM port of desktop/laptop.**




**3. Set the DIP-switch to "Init", then power up the ET-M8194H.**

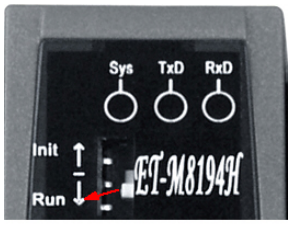

( Dip Switch -- Init )

**4. Choose the COM Port, and then click the Get button to read back the IP configuration.**

**5. Fill the settigs in IP, Mask and Gateway fields, and then click the Set button to change the IP configuration.**

**6. Click the Default button and then click the Set button to restore the default IP configuration.**

**7. Shut down the ET-M8194H, and set the DIP-switch to "Run".**




( Dip Switch -- Run )

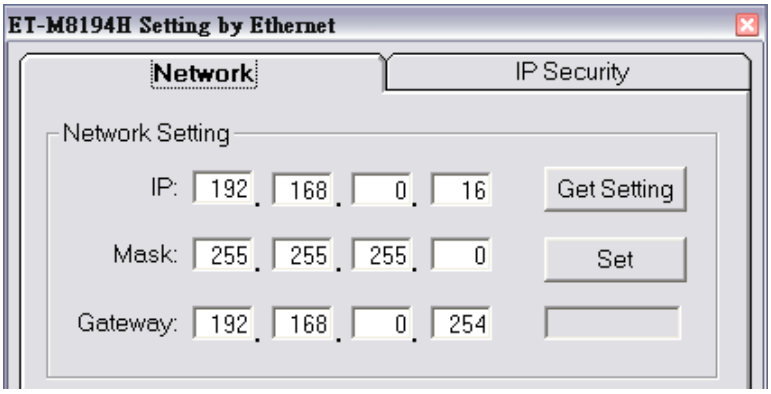
**8. Power up the ET-M8194H.**

## Method 2 – Setting via Ethernet

**Execute the EzMove Utility and connect to ET-M8194H with Ethernet. Start the "ET-M8194H Settinh by Ethernet" dialog. ([Setting] – [ET-M8194H Setting] – [By Ethernet] – [Network Tab])**

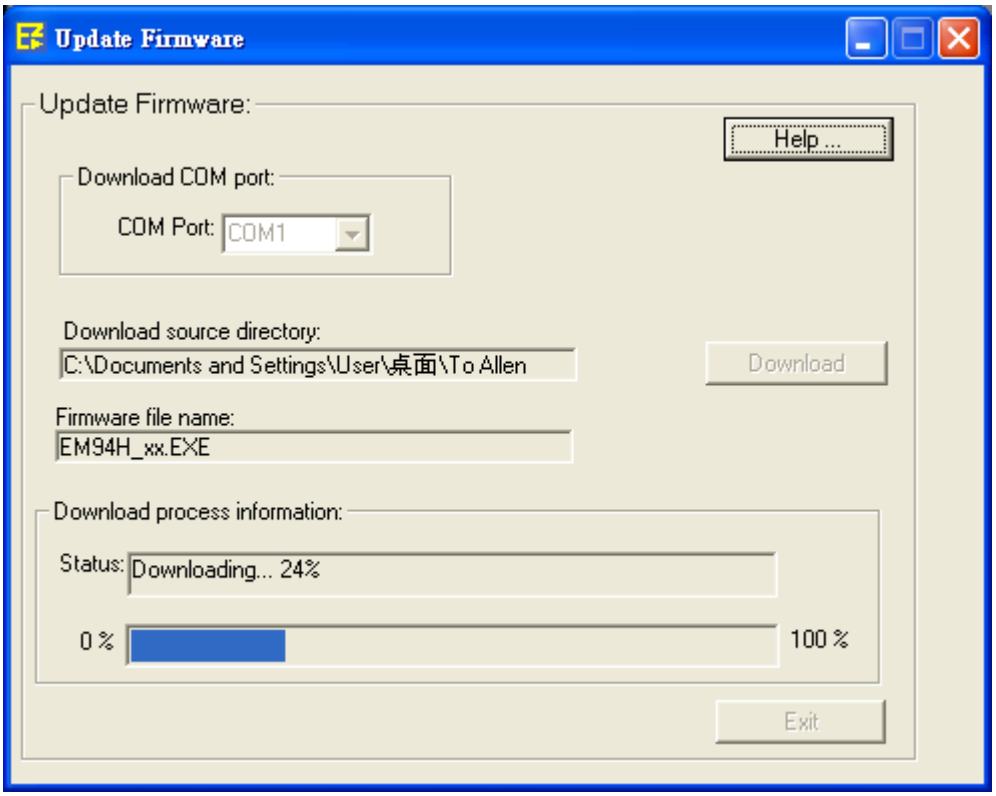
**Get Setting : read the current IP configuration from ET-M8194H.**

**Set : Fill the new settigs in IP, Mask and Gateway fields. Click this button to start IP configuration. The configuration will be completed in 10 seconds, and the EzMove Utility will re-connect to ET-M814H with updated IP address.**



# I. Update Firmware

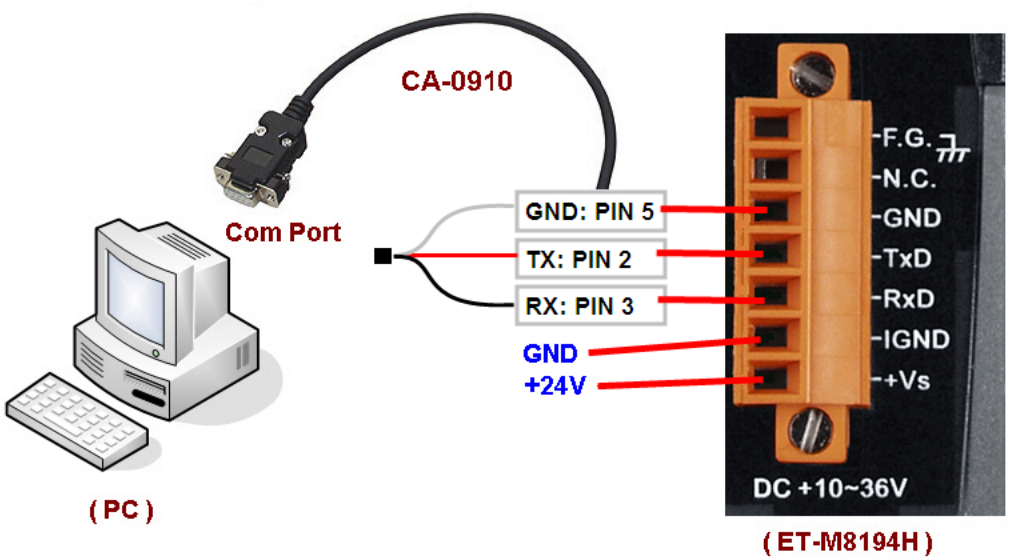
**Execute the *EzMove_Utility.exe* and open the "Update Firmware" dialog from the menu-bar: [Setting] – [ET-M8194H Setting] – [By COM Port] – [Update Firmware].**




**Please follow the following procedures to update firmware:**

1. **Shut down the ET-M8194H.**

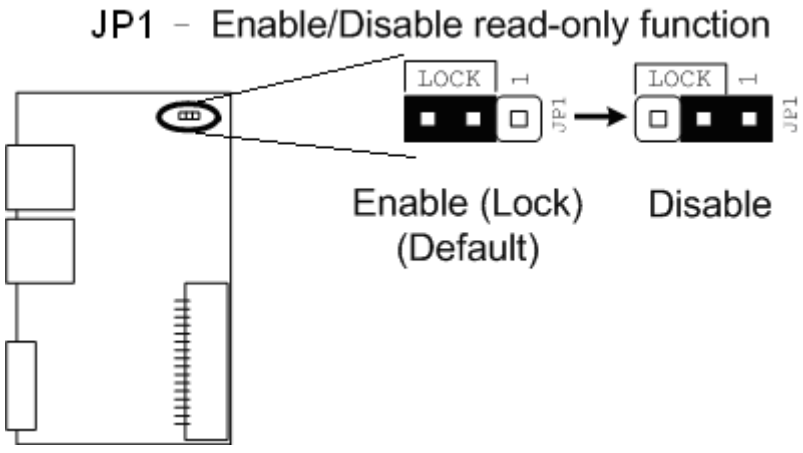
2. **Connect to ET-M8194H with the associated RS-232 cable (CA-0910). The Tx, Rx and GND pins of CA-0910 are connected to the Rx, Tx and GND ports of ET-M8194H. Please connect the other end (9-pin, D-sub connector) to the normal COM port of desktop/laptop.**




**3. Set the DIP-switch to "Init".**


( Dip Switch -- Init )

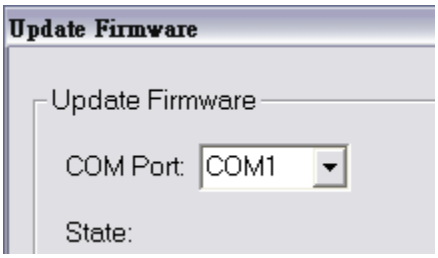
**4. Change the setting of JP1 to *disable* ead-only function.**




**5. Power up the ET-M8194H.**

6. Delete the autoexec.bat and EM94H_XX.EXE in the installed folder of EzMove (\ICPDAS\ET_M8194H\EzMove_Utility). Copy the new autoexec.bat, ET_M8194H_API.dll and the EM94H_XX.EXE into the same folder.

7. Choose the COM Port that is connected to ET-M8194H (through CA-0903 cable).

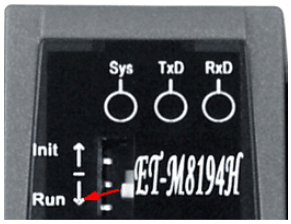



8. Click Dowload button to start firmware-updating procedure.

9. After finishing firmware-updating procedure, shut down the ET-M8194H.

10. Restore the setting of JP1 to *enable* ead-only function.

11. Set the DIP-switch to “Run”.




( Dip Switch -- Run )

12. Power up the ET-M8194H.

# J. History of Versions

| Version | Author | Date | Description of changes |
|---|---|---|---|
| v1.0 | Edward | 13-JAN-2008 | The First Version |
| v2.0 | Allen | 13-NOV-2012 | The Second Version |