Modbus Protocol
On MD Series Controllers

MD. Co., Ltd

1. Introduction

Communication protocol(RS485, Modbus) on the BLDC/DC motor controller.

■ MD series BLDC motor controller(OP->Option)

Name	Volt	Curr-ent(A)	RS 485	TTL 232	CAN	MOD-BUS	PULSE	RC	CLUTCH	POW_SW
MD50	12~24VDC	3								
MD50C	**12~24VDC**	**3.5**	○			○				
PNT50	12~24VDC	3x2	○							
MD100	12~24VDC	7								
MD200	12~48VDC	10	○							○
MD400	12~48VDC	20	○	○		○	○	○	○	○
MD750	24~72 VDC	30	○	○	○	○	○	○	○	○
MD750T	12~48 VDC	30x2	○	○	○			○	○	○
MD1K	12~48 VDC	50	○	○	○	○	○	○	○	○
MD2K	24~48 VDC	100	○	○	○	○		○	○	○
MDA200	110~220VAC	1.5	○(OP)	○						
MDA400	110~220 VAC	2.5	○(OP)	○	○(OP)					
MDA400C	110~220 VAC	2.5	○	○		○	○	○		
MDA1K	110~220 VAC	5	○	○	○	○	○	○	○	
MDA1KH	380~460 VAC	2.5	○							
MDA2K	110~220 VAC	10	○	○	○	○		○		

- PULSE : speed control by 0~500kpps pulse input
- RC : Wireless RC input
- ENC : Encoder input available(A,B,Z)
- CAN : CAN communication, extended mode only
- TTL232 : TTL level RS232 (G, Rx, Tx, 5VDC)
- CLUTCH : To drive electric brake, or clutch(G, Vpp)
- RS485 : RS485 communication(G, 485+, 485-)
- POW_SW : Power switch
- MODBUS : Modbus communication available
- Baudrate is 19200bps but MDA200, MDA400 only 9600bps

2. Contents

2.1 Function Code

Types	**Content(Hex)**	**Remark**
Bit read	1	On/Off data reading
Word read	3	Word(two byte) data reading
Bit write	6	On/Off data writing
Word write	6	Word data writing
Bit write2	15	Serial bit wriring
Word write2	16(0x10)	Serial word data writing

# We just use Word read/write and Word write2 function

2.2 PID(Parameter IDentification Number, same with ADRESS)

-R : Read only

-W : Write parameter

-R/W : Read and write available

-0xfe(254) : ID ALL(broadcasting ID)

2.3 Word Read(R1)

Request	
Slave Addr	Controller ID
Func code	3
Start Addr Hi	PID_H
Start Addr Lo	PID_L
Quantity of inputs Hi	0
Quantity of inputs Lo	1
CRC Lo	
CRC Hi	

Response	
Slave Addr	Controller ID
Func code	3
Byte Count	2
Data Hi	DH
Data Lo	DL
CRC Lo	
CRC Hi	

Received adress(PID) = (PID_L | (int)PID_H<<8);

Data = (DL | (int)DH<<8);

2.4 Word Write (W1)

Request	
Slave Addr	ID
Func code	6
Start Addr Hi	PID_H
Start Addr Lo	PID_L
Force Data Hi	DH
Force Data Lo	DL
CRC Lo	
CRC Hi	

Response	
Slave Addr	ID
Func code	6
Start Addr Hi	PID_H
Start Addr Lo	PID_L
Force Data Hi	DH
Force Data Lo	DL
CRC Lo	
CRC Hi	

2.5 Word Read2(2words, R2)

Request	
Slave Addr	Controller ID
Func code	3
Start Addr Hi	PID_H
Start Addr Lo	PID_L
Quantity of inputs Hi	0x00
Quantity of inputs Lo	0x02
CRC Lo	
CRC Hi	

Response	
Slave Addr	Controller ID
Func code	3
Byte Count	0x04
Data Hi	D1H
Data Lo	D1L
Data Hi	D2H
Data Lo	D2L
CRC Lo	
CRC Hi	

```
Received adress(PID) = (PID_L | (int)PID_H<<8);
Data1 = (D1L | (int)D1H<<8);
Data2 = D2L | (int)D2H<<8;
Data = D1L | (long)D1H<<8 | (long)D2L<<16 | (long)D2H<<24;
```

2.6 Word Write2(2 words data, W2)

Request	
Slave Addr	ID
Func code	16
Start Addr Hi	PID_H
Start Addr Lo	PID_L
Number of Register Hi	0
Number of Register Lo	2
Byte Count	4
Data Hi	D1H
Data Lo	D1L
Data Hi	D2H
Data Lo	D2L
CRC Lo	
CRC Hi	

Response	
Slave Addr	ID
Func code	16
Start Addr Hi	PID_H
Start Addr Lo	PID_L
Number of Register Hi	0
Number of Register Lo	2
CRC Lo	
CRC Hi	

Data1 = D1L | (int)D1H<<8;

Data2 = D2L | (int)D2H<<8;

Data(Long) = D1L | (long)DlH<<8 | (long)D2L<<16 | (long)D2H<<24;

W:Word write, R:Word read,

W2:Word write2, R2:Word read2, X : don't care

PID/ Addr	**Type**	**PID Name/Remark**	**Contents of data bytes(Range) DL(Low byte), DH(High byte)**	**Variable type Default value**
1	R	PID_VERSION	VER 1.3-> DL = 13 Read : ID, 3, 0, 1, 0, 1, CRCL, CRCH Response : ID, 3, 2, 0, DL, CRCL, CRCH	BYTE
3	W	PID_DEFAULT_SET Default setting	DL = 0x55(CHECK) ID, 6, 0, 3, 0, DL, CRCL, CRCH	BYTE 0x55
5	W	PID_TQ_OFF Stop naturally	Stop motor naturally, data don't care(x). ID, 6, 0, 5, x, x, CRCL, CRCH	BYTE
6	W	PID_BRAKE Erectric brake	Stop motor urgently(electric braking mode) Data don't care	BYTE
10	W	PID_COMMAND	DL = Command	BYTE
		CMD_TQ_OFF	2 : Tq-off, motor free state	
		CMD_BRAKE	4 : Erectric brake	
		CMD_MAIN.. BC_ON	5 : PID_MAIN_DATA broadcasting ON	
		CMD_MAIN_BC_OFF	6 : broadcasing OFF	
		CMD_ALARM_RESET	8 : Reset alarm	
		CMD_POSI_RESET	10 : Position reset(set position to zero)	
		CMD_MONITOR_BC_ON	11 : PID_MONITOR broadcasting ON	
		CMD_MONITOR_BC_OFF	12 : Broadcasting off	
		CMD_IO_MONITOR_ON	13 : PID_IO_MONITOR BC ON	
		CMD_IO_MONITOR_OFF	14 : PID_IO_MONITOR BC OFF	
		CMD_FAN_ON	15 : Fan ON(motor cooling fan)	
		CMD_FAN_OFF	16 : Fan OFF	
		CMD_CLUTCH_ON	17 : Mechanical brake(clutch) ON	
		CMD_CLUTCH_OFF	18 : Mechanical breka OFF	
		CMD_TAR_VEL_OFF	20 : Erase target vel, set by PID_TAR_VEL	
		CMD_SLOW_START_OFF	21 : Erase target slow/start value	
		CMD_SLOW_DOWN_OFF	22 : Erase target slow/down vaule	
		CMD_CAN_RESEND_ON,	23 : Send CAN data to RS485 serial port.	
		CMD_CAN_RESEND_OFF	24 : Turn off resending of CAN data	
		CMD_MAX_TQ_OFF	25 : Erase target limit load(max. current)	
		CMD_ENC_OFF	26 : Cancel the use of encoder sensor.	
		CMD_LOW_SPEED_LIMIT_OF	27 : Cancel the set of low speed limit.	
		F, ... HIGH..	28 : Cancel the set of high speed limit.	
		CMD_SPEED_LI._OFF	29 : Cancel the set of low/high speed limits.	
		CMD_CURVE.. OFF	31 : Cancel set of curve fitting func.	

		CMD_STEP.. OFF CMD_UICOM_OFF CMD_UICOM_ON CMD_MAX_RP._OFF CMD_HALL_TY..OFF CMD_MAIN..BC_ON2 CMD_MAIN..BC_OFF2 CMD_MONIT..BC_ON2 CMD_MONIT..BC_OFF2 CMD_IO_MONIT..BC_ON2 CMD_IO_MONIT..BC_OFF2	32 : Cancel step input mode 44 : I/O control(ctrl 11pin cnt) available 45 : I/O control disable(when comm. is used) 46 : Cancel max. speed set by DIP SW 47 : Cancel set of motor hall type 50 : PID_MAIN_DATA, BC ON for 2nd motor 51 : PID_MAIN_DATA, BC OFF for 2nd motor 52 : PID_MONITOR, BC ON for 2nd motor 53 : PID_MONITOR, BC OFF for 2nd motor 54 : PID_IO_MONITOR, BC ON for 2nd motor 55 : PID_IO_MONITOR, BC OFF for 2nd motor ID, 6, 0, 3, 0, DL, CRCL, CRCH	
12	W	PID_ALARM_RESET Reset alarm	DL : Data don't care(x) ID, 6, 0, 12, 0, DL, CRCL, CRCH	-
13	W	PID_POSI_RESET Reset position, POSI.->0	Data don't care(x) ID, 6, 0, 13, 0, DL, CRCL, CRCH	-
16	R/W	PID_INV_SIGN_CMD Inverse of moving direction Used to the left, right, two wheel driving system.	DL : 1 or 0 If PC sending data is X->Controller gets -X 1 : Inverse of reference direction 0 : Don't use inverse sign(normal command) Read : ID, 0x03, 0, 16, 0, 1, CRCL, CRCH Write : ID, 0x06, 0, 16, 0, DL, CRCL, CRCH	BYTE 0
17	R/W	PID_USE_LIMIT_SW Safety limit switch func.	DL : 1 or 0 1 : CTRL connector on, 6,7 is used as limit switchs(DIR, START/STOP) 0 : Cancel limit switch function. Read : ID, 3, 0, 17, 0, 1, CRCL, CRCH Write : ID, 6, 0, 17, 0, DL, CRCL, CRCH	BYTE 1

<table>
<tr><th>PID/
Addr</th><th>Type</th><th>PID Name/Remark</th><th>Contents of data bytes(Range)
DL(Low byte), DH(High byte)</th><th>Variable type
Default value</th></tr>
<tr><td>19</td><td>R/W</td><td>PID_INV_ALARM
Inverse the sign of alarm signal(output)</td><td>DL : 1 or 0
1 : Inverse the alarm signal on/off status
0 : Normal signal
Read : ID, 3, 0, 19, 0, 1, CRCL, CRCH
Write : ID, 6, 0, 19, 0, DL, CRCL, CRCH</td><td>BYTE
0</td></tr>
<tr><td>21</td><td>R/W</td><td>PID_HALL_TYPE
Set the poles of BLDC motor</td><td>DL : 0~
Set the number of poles on motor<table><tr><td>Pole</td><td>4</td><td>8</td><td>10</td><td>12</td><td>2</td><td>6</td></tr><tr><td>No.</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr></table>and over the no. 5 must be set by comm. input then, the pole number is following,
poles of motor = DL * 2
Read : ID, 3, 0, 21, 0, 1, CRCL, CRCH
Write : ID, 6, 0, 21, 0, DL, CRCL, CRCH</td><td>BYTE
0</td></tr>
<tr><td>24</td><td>R/W</td><td>PID_STOP_STATUS
Set the stop status</td><td>DL : 0~3
0 : If speed input is zero, don't control(free)
1 : Control zero speed(like a servo lock)
2 : Electric brake mode
3 : Free(tq-off)
Read : ID, 3, 0, 24, 0, 1, CRCL, CRCH
Write : ID, 6, 0, 24, 0, DL, CRCL, CRCH</td><td>BYTE
0</td></tr>
<tr><td>25</td><td>R/W</td><td>PID_INPUT_TYPE
Set the user input type</td><td>DL : 0~5
0 : Analog input(0~5V) or PWM(>5Khz)
1 : Joystick(0~2.5~5), center(2.5V)
5 : Step(1~7 step input)
Set the speed steps up to 7 step.
The step input number is set by following
BIT0 : INT_SPEED, CTRL No. 2
BIT1 : RUN/BRAKE, CTRL No. 7
BIT2 : START/STOP, CTRL No. 8
StepInputNumber = BIT0 + BIT1x2 + BIT2x4
Read : ID, 3, 0, 25, 0, 1, CRCL, CRCH
Write : ID, 6, 0, 25, 0, DL, CRCL, CRCH</td><td>BYTE
0</td></tr>
</table>

PID/ Addr	**Type**	**PID Name/Remark**	**Contents of data bytes(Range)** **DL(Low byte), DH(High byte)**	**Variable type** **Default value**
34	R	PID_CTRL_STATUS Read control status	DL : STATUS BIT0 : ALARM(1-> alarm status, 0->normal) BIT1 : CTRL_FAIL, Speed control fail BIT2 : OVER_VOLT, Over voltage BIT3 : OVER_TEMP, Over temperature BIT4 : OVER_LOAD, Overload BIT5 : HALL_FAIL, Hall sensor or encoder fail BIT6 : INV_VEL, Motor speed inversed BIT7 : STALL, motor not moved Read : ID, 3, 0, 34, 0, 1, CRCL, CRCH Response : ID, 3, 2, 0, DL, CRCL, CRCH	BIT 0
36	R/W	PID_START_INV_SIGN START/STOP signal inverse	DL : 1 or 0 1 : Inverse START/STOP signal(HIGH->ON) 0 : Normal signal(LOW->ON) Read : ID, 3, 0, 36, 0, 1, CRCL, CRCH Write : ID, 6, 0, 36, 0, DL, CRCL, CRCH	BYTE 0
36	R/W	PID_START_INV_SIGN START/STOP signal inverse	DL : 1 or 0 1 : Inverse START/STOP signal(HIGH->ON) 0 : Normal signal(LOW->ON) Read : ID, 3, 0, 36, 0, 1, CRCL, CRCH Write : ID, 6, 0, 36, 0, DL, CRCL, CRCH	BYTE 0
37	R/W	PID_RUN_INV_SIGN RUN/BRAKE signal inverse	DL : 1 or 0 1 : Inverse RUN/BRAKE signal(HIGH->ON) 0 : Normal signal(LOW->ON) Read : ID, 3, 0, 37, 0, 1, CRCL, CRCH Write : ID, 6, 0, 37, 0, DL, CRCL, CRCH	BYTE 0
40	R/W	PID_LIMIT_STOP_COND Set the stop condition when limit switch is ON	DL : 1 or 0 Stop status when limit switch is ON(CW/CCW and STAR/STOP) 1 : BRAKE(electric braking) 0 : Natural stop(Free, TqOff) Read : ID, 3, 0, 40, 0, 1, CRCL, CRCH Write : ID, 6, 0, 40, 0, DL, CRCL, CRCH	BYTE 0

PID/ Addr	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
44	R/W	PID_TQ_CTRL Current(torque) control	DL : 1 or 0 1 : Current control(Torque) 0 : not current control(normal control) Read : ID, 3, 0, 44, 0, 1, CRCL, CRCH Write : ID, 6, 0, 44, 0, DL, CRCL, CRCH	BYTE 0
45	R/W	PID_BLUETOOTH When use bluetooth Set(send) this PID	DL : 1 or 0 1 : Bluetooth commucation, 0 : not used Read : ID, 3, 0, 40, 0, 1, CRCL, CRCH Write : ID, 6, 0, 40, 0, DL, CRCL, CRCH	BYTE 0

■ User input type(PID_INPUT_TYPE, CTRL Pin no. 10 or PULSE_IN, Pin no. 2)

MODE	Contents/Connector	Range			비고(digit)
		Input	Speed(position)	Center	
0	Analog voltage/CTRL	0~5VDC	0~max.	-	Default setting
	PWM/CTRL	Duty cycle	0~max.	-	More than 5Khz
1	Joystick voltage/CTRL	0~5VDC	-max.~+max.	2.5VDC	deadzone:2~3VDC
2	Pulse/PULSE_IN	5~500kpps	0~max.		
3	RC(pulse)(>250Hz) /PULSE_IN	1.05~1.95ms	min-center-max	1.5ms	deadzone:1.4~1.6ms
5	Step(digital input)/CTRL BIT0:INT_SPEED BIT1:RUN/BRAKE BIT2:START/STOP	0~7	Set value	-	0 : Stop the motor 1~7 : drive motor by set value.

■ Step input(7steps for speed setting)

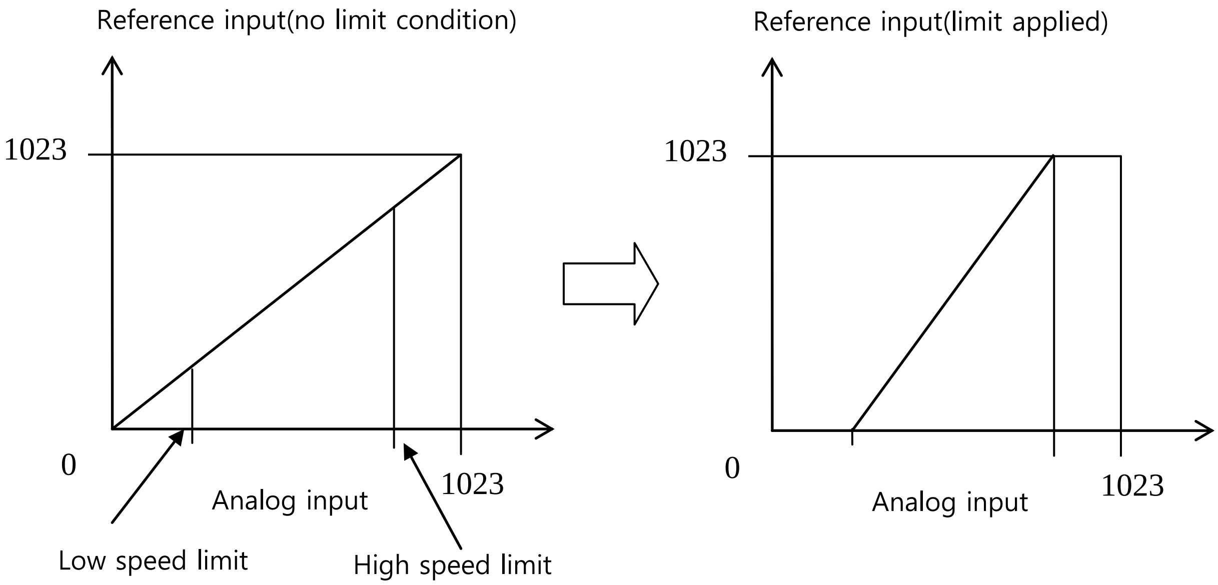
STEP INPUT(CTRL connector)				Default setting(%)
No.	INT_SPEED	RUN/BRKAKE	START/STOP	Percentage of max. speed
0	OFF	OFF	OFF	0(stop condition)
1	ON	OFF	OFF	14
2	OFF	ON	OFF	28
3	ON	ON	OFF	42
4	OFF	OFF	ON	57
5	ON	OFF	ON	71
6	OFF	ON	ON	85
7	ON	ON	ON	100

PID/ Addr	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
48	R	PID_DI Digital input(ON/OFF)	DL : Digital input of CTRL connector BIT0 : INT_SPEED(Speed input by internal VR) BIT1 : ALARM_RESET input BIT2 : DIR(CW/CCW) direction input BIT3 : RUN/BRAKE input BIT4 : START/STOP input BIT5 : ENC_B(Encoder signal input) BIT6 : ENC_A User use the limit SW when control by comm. Then DIR, START/STOP signal is read by this PID Read : ID, 3, 0, 48, 0, 1, CRCL, CRCH Response : ID, 3, 2, 0, DL, CRCL, CRCH	BIT
49	R	PID_IN_POSITION_OK Position control result	DL : 1 or 0 IN_POSITION(PID 171) Piosition control result on success or not. If motor position control error is in IN_POSITION, then the result is 1, else 0 Read : ID, 3, 0, 49, 0, 1, CRCL, CRCH Response : ID, 3, 2, 0, DL, CRCL, CRCH	BYTE

PID/ Addr	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
100	R/W	PID_START_STOP If there is a target speed(by PID155) then control like as START/STOP, DIR I/O act.	DL : 0,1,2 0 : STOP, 1 : CCW direction driving 2 : CW direction driving Refer to PID_COM_TAR_SPEED(180) Target speed is set by PID_COM_TAR_SPEED(180) Read : ID, 3, 0, 100, 0, 1, CRCL, CRCH Write : ID, 6, 0, 100, 0, DL, CRCL, CRCH	BYTE 0
130	R/W	PID_VEL_CMD Velocity command (unit : rpm)	Data(speed, rpm) = DH*256 + DL Speed>0, CCW direction from the shaft Speed<0, CW direction Read : ID, 3, 0, 130, 0, 1, CRCL, CRCH Write : ID, 6, 0, 130, DH, DL, CRCL, CRCH	INT
133	W	PID_ID ID setting	DL = WRITE_CHK_BYTE(0xaa) DH = ID of controller to set(1~253) Write : ID, 6, 0, 133, DH, DL, CRCL, CRCH	BYTE 1
134	**W**	**PID_OPEN_VEL_CMD** **Open-loop control**	**DL, DH: Open-loop velocity** **Open loop output = DH*256 + DL** **Range : -1023~1023** **Write : ID, 6, 0, 134, DH, DL, CRCL, CRCH**	**INT**
135	W	PID_BAUDRATE Baudrate setting(RS485)	DL = WRITE_CHK_BYTE(0xaa) DH = Baudrate(1~5) 1 : 9600bps, 2 : 19200bps 3 : 38400bps, 4 : 57600bps, 5 : 115200bps Write : ID, 6, 0, 135, DH, DL, CRCL, CRCH	BYTE 2
138	**R**	**PID_RPM_DATA** **Motor speed**	**DL, DH: Motor speed(rpm)** **Open loop output = DH*256 + DL** **Range : -5,000~5,000** **Read : ID, 3, 0, 138, 0, 1, CRCL, CRCH** **Response : ID, 3, 2, DH, DL, CRCL, CRCH**	**INT**
139	R	PID_TQ_DATA Torque(current) control	Return current(unit 0.1A), 10->1A, 15->1.5A Current = (DL | DH<<8)/10 Read : ID, 3, 0, 139, 0, 1, CRCL, CRCH	INT
140	W	PID_TQ_CMD Torque(current) control	Target current = DH*256 + DL Range : -1023~1023 Write : ID, 6, 0, 140, DH, DL, CRCL, CRCH	INT

PID/ Addr	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
143	R	PID_VOLT_IN Supply voltage	Return supply voltage Unit of DC power : 0.1V(245 -> 24.5V) Unit of AC power : 1V(internal DC voltage) Voltage = DL | (DH<<8) Read : ID, 3, 0, 143, 0, 1, CRCL, CRCH Response : ID, 3, 2, DH, DL, CRCL, CRCH	INT
153	R/W	PID_SLOW_START Set the SlowStart variable (not use internal volume)	Value of SlowStart(0~1023->0~15s delay) Applied when the reference speed increased SlowStart = DH*256 + DL Write : ID, 6, 0, 153, DH, DL, CRCL, CRCH	INT 0
154	R/W	PID_SLOW_DOWN Set the SlowDown variable	Value of SlowDown(0~15s) Applied when the ref. speed decreased. SlowDown = DH*256 + DL Read : ID, 3, 0, 154, 0, 1, CRCL, CRCH Write : ID, 6, 0, 154, DH, DL, CRCL, CRCH	INT 0
155	R/W	PID_TAR_VEL Use fixed set speed, no use internal/external volume(SPEED_IN)	Replace internal/external volume, SPEED_IN TAR_VEL = Data, 0~MaxRPM(about 5000rpm) START/STOP, RUN/BRAKE, DIR is used with that TAR_VEL Target speed(rpm) = DH*256 + DL Read : ID, 3, 0, 155, 0, 1, CRCL, CRCH Write : ID, 6, 0, 155, DH, DL, CRCL, CRCH	INT

PID	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
156	R/W	PID_ENC_PPR Set encoder pulse	When the encoder pulse is set. The controller uses encoder signals as a speed sensor. ENC_PULSE = DH*256 + DL Read : ID, 3, 0, 156, 0, 1, CRCL, CRCH Write : ID, 6, 0, 156, DH, DL, CRCL, CRCH	INT 0
157	R/W	PID_LOW_SPEED_LIMIT Set the low value of analog input(lower then this value is same as zero input)	Setting range is 0~512 If the input is lower than LOW_LIMIT This input is treated as a zero LOW_LIMIT = DH*256 + DL Read : ID, 3, 0, 157, 0, 1, CRCL, CRCH Write : ID, 6, 0, 157, DH, DL, CRCL, CRCH	INT 0
158	R/W	PID_HIGH_SPEED_LIMIT Set the higher value of analog input	Setting range is 512~1023 Higher than this HIGH_LIMIT is same to HIGH_LIMIT = DH*256 + DL Read : ID, 3, 0, 158, 0, 1, CRCL, CRCH Write : ID, 6, 0, 158, DH, DL, CRCL, CRCH	INT 0



Reference input vs. input voltage by SPEED_LIMIT

PID/ Addr	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
167	R/W	PID_PV_GAIN P-gain of position control	PV_GAIN : Position P gain RefSpeed = PV_GAIN*Position Error PV_GAIN = DH*256 + DL Read : ID, 3, 0, 167, 0, 1, CRCL, CRCH Write : ID, 6, 0, 167, DH, DL, CRCL, CRCH	INT
168	R/W	PID_P_GAIN P-gain for velocity control	P_GAIN : Proportional gain for speed control P_GAIN = DH*256 + DL Read : ID, 3, 0, 168, 0, 1, CRCL, CRCH Write : ID, 6, 0, 168, DH, DL, CRCL, CRCH	INT
169	R/W	PID_I_GAIN I-gain for velocity control	I_GAIN : Integral gain for velocity control I_GAIN = DH*256 + DL Read : ID, 3, 0, 169, 0, 1, CRCL, CRCH Write : ID, 6, 0, 169, DH, DL, CRCL, CRCH	INT
180	**C/R**	**PID_COM_TAR_SPEED** **Refer to PID_START_STOP, 100**	**TAR_SPEED : Target speed(rpm)** **TAR_SPEED(rpm) = DH*256 + DL** **Read : ID, 3, 0, 180, 0, 1, CRCL, CRCH** **Write : ID, 6, 0, 180, DH, DL, CRCL, CRCH**	**WORD**
181	R/W	PID_MAX_LOAD Set the max. current	MAX_LOAD : Max. current available. MAX_LOAD(0.1A) = DH*256 + DL Read : ID, 3, 0, 181, 0, 1, CRCL, CRCH Write : ID, 6, 0, 181, DH, DL, CRCL, CRCH	INT
183	**R/W**	**PID_FUNC_CMD_TYPE** **If the function type is set(more than 1) then the motor used by start/stop only.**	**DL, DH : Special function type** **Refer to the function below explanation** **FUNC_TYPE = DL** **DH is ignored(don't care)** **0 : NONE(normal driving)** **1 : SPEED** **2 : POSITION** **3 : SPEED_MONENTARY** **4 : SPEED_ADD** **Refet to the below explanation(page 17)** **Read : ID, 3, 0, 183, 0, 1, CRCL, CRCH** **Write : ID, 6, 0, 183, DH, DL, CRCL, CRCH**	**WORD**

PID/ Addr	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
184	**W**	**PID_FUNC_CMD If function type is, then user can use start/stop signal and this PID also.**	**D1, D2 : Run the function set by PID 183 Like as control input signal START/STOP CMD = (D1 | D2<<8) 0 : OFF, more than 1 : ON Write : ID, 6, 0,184, DH, DL, CRCL, CRCH**	**WORD**
211	R/W	PID_MAX_LOAD Set the max. current	MAX_LOAD : Max. current available. MAX_LOAD(0.1A) = DH*256 + DL Read : ID, 3, 0, 211, 0, 1, CRCL, CRCH Write : ID, 6, 0, 211, DH, DL, CRCL, CRCH	INT
221	R/W	PID_MAX_RPM Set max. speed(rpm)	MAX_SPEED(rpm) : max. control speed(rpm) MAX_SPEED(rpm) = DH*256 + DL Read : ID, 3, 0, 221, 0, 1, CRCL, CRCH Write : ID, 6, 0, 221, DH, DL, CRCL, CRCH	INT
222	R2/W2	PID_SPEED_LIMIT Refer to above fig. (Low/high speed limit)	Limit speed input(analog input) L_LIMIT : low speed limit(0~512). L_LIMIT = D1H*256 + D1L H_LIMIT : high speed limit(512~1023) H_LIMIT = D2H*256 + D2L R2: ID, 3, 0, 222, 0, 2, CRCL, CRCH W2 :ID,16,0,222,D1H,D1L,D2H,D2L,CRCL,CRCH	INT INT
239	**R2/W2**	**PID_FUNC_SPEED With target speed the motor run for a run time(delay)**	**Data : 4bytes Target speed(rpm) = D1H*256 + D1L Delay(run time, 0.1s unit), 10->1s = D2H*256 + D2L R2: ID, 3, 0, 239, 0, 2, CRCL, CRCH ID,16,0,239,D1H,D1L,D2H,D2L,CRCL,CRCH**	**INT INT**
243	W2	PID_POSI_CMD Move to target positon	TAR_POSI : Target position TAR_POSI = D1L | D1H<<8 | D2L<<16 | D2H<<24 ID,16,0,243,D1H,D1L,D2H,D2L,CRCL,CRCH	LONG
244	W2	PID_INC_POSI_CMD Move to target position	INC_TAR_POSI : Incremental target position Real target posi. = Current posi. + Incremental target position. INC_TAR_POSI = D1L | D1H<<8 | D2L<<16 | D2H<<24 ID,16,0,244,D1H,D1L,D2H,D2L,CRCL,CRCH	LONG

PID/ Addr	Type	PID Name/Remark	Contents of data bytes(Range) DL(Low byte), DH(High byte)	Variable type Default value
250	**R2/W2**	**PID_FUNC_POSI** **With target speed the motor move to the target position**	**Data : 4bytes** **Target speed(rpm) = D1H*256 + D1L** **Target position(motor poles*3/rev)** **= D2H*256 + D2L** **R2: ID, 3, 0, 250, 0, 2, CRCL, CRCH** **ID,16,0,250,D1H,D1L,D2H,D2L,CRCL,CRCH**	**INT** **INT**

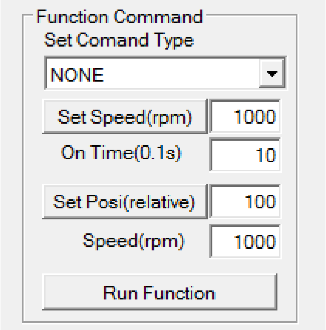
■ Function types(PID_FUNC_CMD_TYPE)

START/STOP : control input signal of controller, refer to each controller spec.

Related PID(PID_FUNC_CMD_TYPE, PID_FUNC_CMD, PID_FUNC_SPEED, PID_FUNC_POSI)

Special function types are 4 kinds, following

TYPE_SPEED, TYPE_SPEED_MOMENTARY, TYPE_SPEED_ADD, TYPE_POSI.



Function Command

Set Comand Type: NONE

Set Speed(rpm): 1000

On Time(0.1s): 10

Set Posi(relative): 100

Speed(rpm): 1000

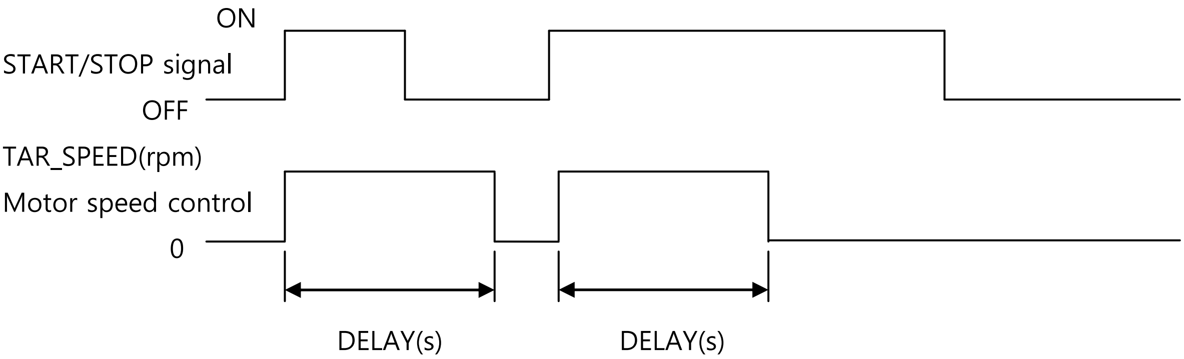
Run Function

refer to the control panel of MDAS

- Function type, SPEED(1)

Motor moves when the signal of START/STOP is ON from OFF set by PID_FUNC_SPEED(target speed, and on time(delay))

X-axis is time(s)



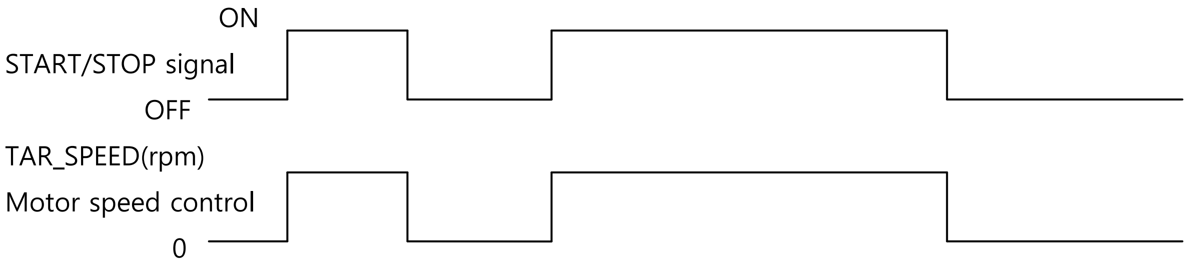
Here, the target speed and delay is set by PID_FUNC_SPEED(target speed(TAR_SPEED), and delay(DELAY))

And the START/STOP signal is raplaced by PID_FUNC_CMD(184) when user want to use comm. only

- Function type, SPEED_MOMENTARY(3)

Motor moves when the signal of START/STOP is ON from OFF set by PID_FUNC_SPEED(target speed, and on time(delay))

Refer to following diagram.

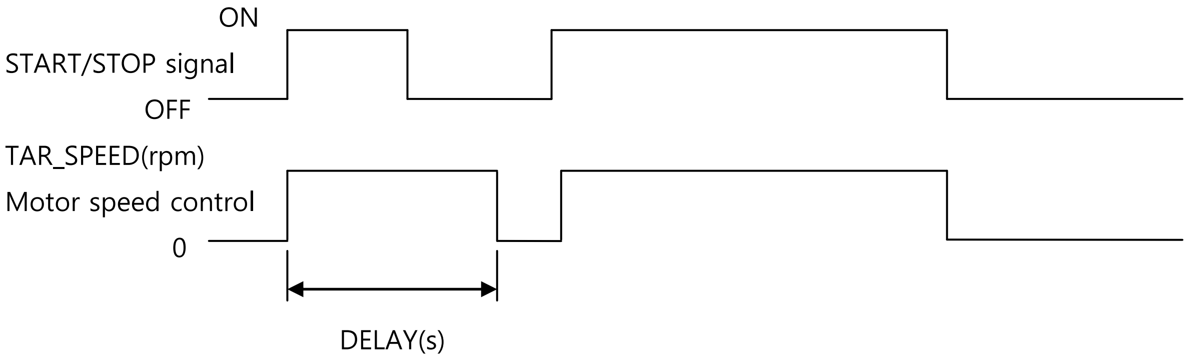


Here, the target speed only applied, on time is ignored.

- Function type, SPEED_ADD(4)

Motor moves when the signal of START/STOP is ON from OFF set by PID_FUNC_SPEED(target speed, and on time(delay))

Refer to following diagram.

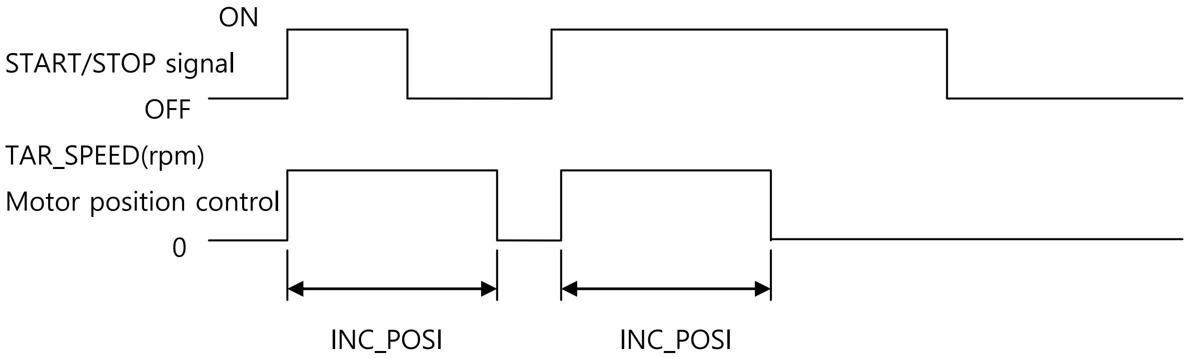


If the signal of START/STOP is ON then motor run continuously regardless of delay(DELAY)

- Function type, POSI(2)

Motor moves to the target position when the signal of START/STOP is ON from OFF

X-axis is motor position.



Here, the target speed(TAR_SPEED) and incremental position(INC_POSI) are set by PID_FUNC_POSI(250)

2.4 PID HEADER

```
// General ID definition
#define ID_ALL                          0xfe
#define ID_WRITE_CHK                    0xaa
#define ID_DEFALUT_CHK                  0x55                // Default setting(write)
#define ID_DEVELOPER_CHK                0x77

////////////////////////////////////////////////////////
// Command : RMID, TMID, ID, PID, Data number, Data.., CHK
//////////////////////////////////////////////
// PID one-byte data : PID 0~127
#define PID_DEFAULT_SET                 3
#define PID_REQ_PID_DATA                4
#define PID_TQ_OFF                      5
#define PID_BRAKE                       6
#define PID_ACK                         7

/////////////////////////////////////////
#define PID_COMMAND                     10

#define CMD_TQ_OFF                      2
#define CMD_BRAKE                       4
#define CMD_MAIN_DATA_BC_ON             5
#define CMD_MAIN_DATA_BC_OFF            6
#define CMD_ALARM_RESET                 8
#define CMD_POSI_RESET                  10
#define CMD_MONITOR_BC_ON               11
#define CMD_MONITOR_BC_OFF              12
#define CMD_IO_MONITOR_BC_ON            13
#define CMD_IO_MONITOR_BC_OFF           14
#define CMD_FAN_ON                      15
#define CMD_FAN_OFF                     16
#define CMD_CLUTCH_ON                   17
#define CMD_CLUTCH_OFF                  18
#define CMD_TAR_VEL_OFF                 20
#define CMD_SLOW_START_OFF              21
#define CMD_SLOW_DOWN_OFF               22
#define CMD_CAN_RESEND_ON               23
```

```
#define CMD_CAN_RESEND_OFF              24
#define CMD_MAX_LOAD_OFF                25
#define CMD_ENC_PPR_OFF                 26
#define CMD_LOW_SPEED_LIMIT_OFF         27
#define CMD_HIGH_SPEED_LIMIT_OFF        28



#define PID_ALARM_RESET                 12
#define PID_POSI_RESET                  13
#define PID_MAIN_BC_STATUS              14
#define PID_MONITOR_BC_STATUS           15
#define PID_INV_SIGN_CMD                16
#define PID_USE_LIMIT_SW                17
#define PID_INV_ALARM                   18
#define PID_HALL_TYPE                   19
#define PID_INPUT_TYPE                  25
#define PID_PRESET_SAVE                 30
#define PID_PRESET_RECALL               31

// PID two-byte data : PID 128 ~ 192
#define PID_VEL_CMD                     130
#define PID_VEL_CMD2                    131
#define PID_ID                          133
#define PID_OPEN_VEL_CMD                134
#define PID_BAUD_RATE                   135        // 9600, 19200, 38400, 57600 , 115200
#define PID_ECAN_BITRATE                137                   // 50K,100K,250K,500K,1M
#define PID_INT_RPM_DATA                138
#define PID_TQ_DATA                     139

#define PID_VOLT_IN                     143
#define PID_CCW_PHASE_OFFSET            146
#define PID_CW_PHASE_OFFSET             147

// 0 no return, 1:Monitor, 2:Ack return
#define PID_RETURN_TYPE                 149
#define RETURN_TYPE_MONITOR              1
#define RETURN_TYPE_ACK                  2
#define RETURN_TYPE_IO_MONITOR           3
```

```
#define PID_TQ_PO                         150
#define PID_OVER_MODULATION               152
#define PID_SLOW_START                    153
#define PID_SLOW_DOWN                     154
#define PID_TAR_VEL                       155
#define PID_ENC_PPR                       156
#define PID_LOW_SPEED_LIMIT               157
#define PID_HIGH_SPEED_LIMIT              158
#define PID_SLOW_START_DOWN               159
#define PID_DEAD_ZONE                     162
#define PID_READ_ADDR                     163
#define PID_REQ_PID_DATA2                 164


// PID N-byte data : PID 193 ~ 240
#define PID_MAIN_DATA                     193
#define PID_IO_MONITOR                    194
#define PID_MONITOR                       196
#define PID_POSI_DATA                     197
#define PID_RPM_DATA                      198
#define PID_VEL_GAIN                      202
#define PID_VEL_GAIN2                     203
#define PID_TYPE                          205
#define PID_MAX_LOAD                      211
#defien PID_POSI_SET                      217
#define PID_POSI_VEL_CMD                  219
#define PID_MAX_RPM                       221
#define PID_SPEED_LIMIT                   222
#define PID_STEP_INPUT                    225
#define PID_CURVE_PT                      226
#define PID_PRESET_DATA                   227
#define PID_VEL_GAIN_W                    231
#define PID_TIME                          234
#define PID_CAN_RESEND                    238
#define PID_PHASE_OFFSET                  241
```

2.5 Sample program.

```
typedef unsigned char BYTE;
typedef unsigned int WORD;
// crc-16 is based on the polynomial x^16+x^15+x^2+1
WORD Crc16Table[256] = {
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040
};
```

```
////////////////// ModBus CRC16
WORD Crc16(WORD crc, BYTE *buf, int len)
{
        int i;
        WORD tmp;

        for(i=0; i<len; i++) {
                tmp = crc ^ (0x00ff & *(char *)buf++);
                crc = (crc>>8) ^ Crc16Table[tmp & 0xff];
        }
        return crc;
}

// MODBUS protocol.
int IsCRC16ChkOK(BYTE byIn[], int nPacketSize)
{
        WORD wCRC, wInCRC;

        wInCRC = Crc16(0xffff, byIn, nPacketSize-2);
        wCRC = byIn[nPacketSize-2] | (int)byIn[nPacketSize-1]<<8;
        if(wCRC==wInCRC) return 1;
        else return 0;
}

// Make interger from two bytes
short Byte2Int(BYTE byLow, BYTE byHigh)
{
    return (byLow | (short)byHigh<<8);
}

// Make long type data from four bytes
int Byte2LInt(BYTE byData1, BYTE byData2, BYTE byData3, BYTE byData4)
{
    return((int)byData1 | (int)byData2<<8 | (int)byData3<<16 | (int)byData4<<24);
}
```

```
typedef struct {
          BYTE byLow;
         BYTE byHigh;
} IByte;

typedef struct {
          BYTE byData1;
         BYTE byData2;
         BYTE byData3;
         BYTE byData4;
} LByte;

// Get the low and high byte from interger
IByte Int2Byte(short nIn)
{
          IByte Ret;

          Ret.byLow = nIn & 0xff;
          Ret.byHigh = nIn>>8 & 0xff;
          return Ret;
}
```

```
int MDPutWordData(BYTE byFuncCode, int nData, BYTE byPort)
{
        BYTE byArray[MAX_PACKET_SIZE];
        IByte iData;
        WORD wCRC;

        byArray[0] = DEF_ID;
        byArray[1] = byFuncCode;
        byArray[2] = 2;
        iData = Int2Byte(nData);
        byArray[3] = iData.byHigh;
        byArray[4] = iData.byLow;

        wCRC = Crc16(0xffff, byArray, 5);
        iData = Int2Byte(wCRC);
        byArray[5] = iData.byLow;
        byArray[6] = iData.byHigh;

        PutArray2Buf(byArray, 7, byPort);
        return 1;
}

int MDPutLongData(BYTE byFuncCode, long lData, BYTE byPort)
{
        BYTE byArray[MAX_PACKET_SIZE];
        IByte iData;
        LByte Data;
        WORD wCRC;

        byArray[0] = DEF_ID;
        byArray[1] = byFuncCode;
        byArray[2] = 4;

        Data = Long2Byte(lData);
        byArray[3] = Data.byData2;
        byArray[4] = Data.byData1;
        byArray[5] = Data.byData4;
        byArray[6] = Data.byData3;
        wCRC = Crc16(0xffff, byArray, 7);
        iData = Int2Byte(wCRC);
        byArray[7] = iData.byLow;
        byArray[8] = iData.byHigh;

        PutArray2Buf(byArray, 9, byPort);
        return 1;
}

int Response2Write(BYTE byIn[], int nSize, BYTE byPort)
{
        BYTE byArray[MAX_PACKET_SIZE], i;
        IByte iData;
        WORD wCRC;

        for(i=0; i<nSize; i++) byArray[i] = byIn[i];
        wCRC = Crc16(0xffff, byArray, nSize);
        iData = Int2Byte(wCRC);
        byArray[nSize] = iData.byLow;
        byArray[nSize+1] = iData.byHigh;

        PutArray2Buf(byArray, nSize+2, byPort);
        return 1;
}
```

3 History

VERSION	DATE	CONTENTS
V1.0	3/1/2017	First born
V1.1	11/8/2017	FUNC mode added(PID 239, 250, 183, 184)
V1.1a	12/8/2017	PID added(48, 49)

- The end -