YOSEMITECH

Multi - Parameter Sensor

MODBUS RTU
Programmer Manuel

Table of Contents

1 MODBUS RTU Overview

1.1 Scope

This document is about MODBUS of optical dissolved oxygen probes with software Rev1.6 or later.

1.2 MODBUS Command Structure

Data format in this document:

----Binary number – shown with suffix B. For example: 10001B

----Decimal number – without nay suffix. For example: 256

----Hexadecimal number—shown with prefix 0x. For example: 0x2A

----ASCII character or string – shown with quotation marks. For example: ” YL0114010022”

1.2.1 Command Structure

MODBUS defines a simple protocol data unit (PDU), which is transparent to communication layer.

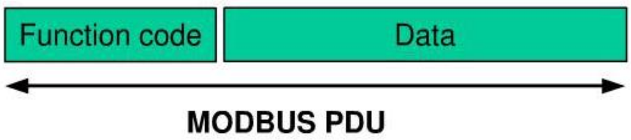

Figure 1: MODBUS Protocol Data Unit

The mapping of MODBUS protocol on a specific bus or network introduces some additional fields on the Protocol Data Unit. The client that initiates a MODBUS transaction builds the MODBUS PDU, and then adds fields in order to build the appropriate communication PDU.

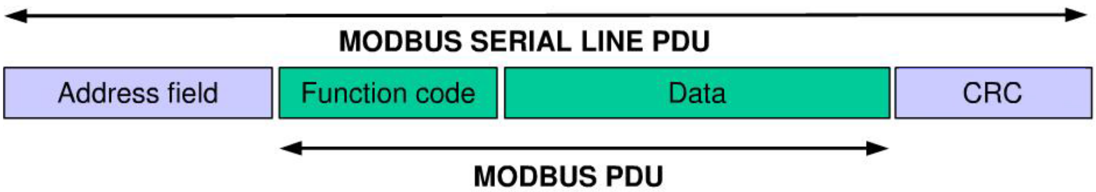

Figure 2: MODBUS Structure for Serial Communication

On a MODBUS serial bus, address field only includes addresses for slave devices.

Note:

- Slave address range for optical dissolved oxygen sensor is: 1…247
- Master device sends a “request frame” with a targeted slave address. When slave device responses, it has to put its own address in the “response frame”, so that master device knows where the response comes from.
- Function code indicates type of operations
- CRC is the result of redundancy check.

1.2.2 MODBUS RTU Transmission Mode

When devices communicate on a MODBUS using RTU (remote terminal unit) mode, each 8-bit byte contains two 4-bit hexadecimal characters. The main advantage of the RTU mode is that it has higher character density, which enables better throughput compare to ASCII mode at same baud rate. Each RTU message must be transmitted in a continuous string of characters.

RTU mode format for each byte (11 bits):

Encoding system	8 bit binary
	Each 8-bit presponseet contains 4-bit hexadecimal characters (0-9, A-F)
Bit per byte:	1 start bit
	8 data bits, least significant bit first
	No parity check
	1 stop bit
Baud rate:	9600bps

Serial transmission of characters:

Every character or byte is sent under this sequence (left to right):

Least Significant Bit (LSB)……Most Significant Bit(MSB)

Start	1	2	3	4	5	6	7	8	Stop

Figure 3: RTU Mode Bit Sequence

CRC Field Structure:

Redundancy check (CRC16)

Frame Structure:

<table>
<tr><th>Slave address</th><th>Function Code</th><th>Data</th><th colspan="2">CRC</th></tr>
<tr><td rowspan="2">1 byte</td><td rowspan="2">1 byte</td><td rowspan="2">0…252 bytes</td><td colspan="2">2 bytes</td></tr>
<tr><td>CRC Low</td><td>CRC High</td></tr>
</table>

Figure 4: RTU Message Frame Structure

Maximum size of MODBUS frame is 256 bytes.

1.2.3 MODBUS RTU Message Frame

In RTU mode, message frames need to be separated by an idle interval of at least 3.5 character lengths. In rest of this document, this idle interval is called t3.5.

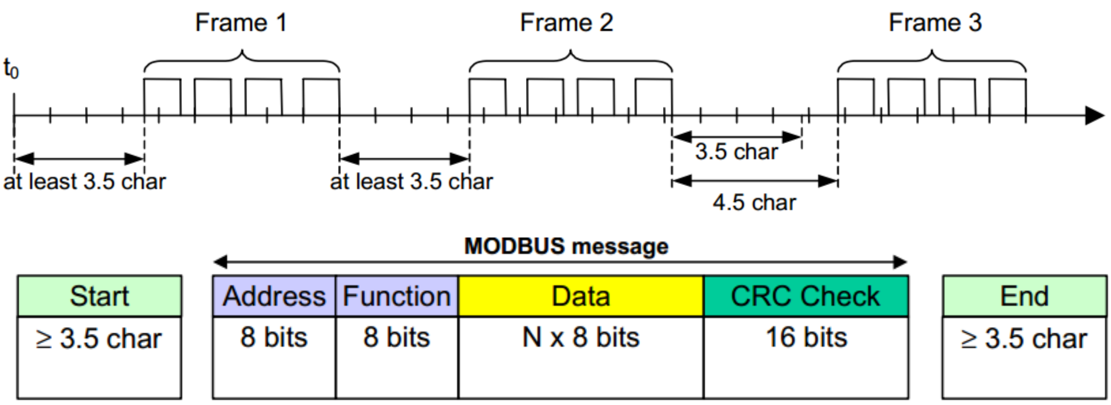

Figure 5: RTU Message Frame

Entire message frame must be sent as continuous stream of characters.

If idle time between two characters is longer than 1.5 characters, the message frame will be considered incomplete, and will be discarded by receiving side.

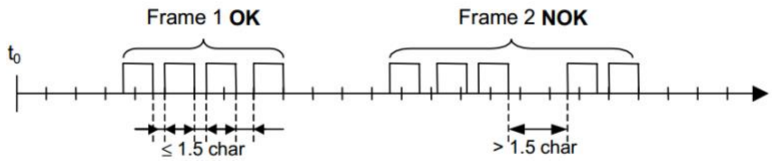

Figure 6: Frame transmission

1.2.4 MODBUS RTU CRC Check

In RTU mode, the error checking field is based on a cyclical redundant checking (CRC) method. The CRC field checks entire content of MODBUS message, regardless of the existence of parity check bit. CRC16 checking method is utilized. CRC result is a 16-bit value with two 8-bit bytes, low order 8-bit byte first followed by high order 8-bit byte.

1.3 MODBUS RTU for YOSEMITECH's Multi - Parameter Sensor

Based on standard MODBUS definition, message frame starts with t3.5 idle interval, and similarly, ends with t3.5 idle interval. Device address and Function code are both 8-bit byte. Data character string has n*8 bits, it contains information about register start/end address and number of registers for read/write operation. CRC field is 16 bit in length.

	Start	Device address	Function code	Data	CRC		End
Value	Idle for 3.5 character length	1-247	Comply with MODBUS function code format	Comply with MODBUS data format	CRC Low	CRC High	Idle for 3.5 character length
Length (byte)	3.5	1	1	n	1	1	3.5

Figure 7: Message frame structure for Yosemitech's MODBUS

1.4 MODBUS RTU Function Code for YOSEMITECH's Multi - Parameter Sensor

YOSEMITECH's Multi-Parameter sensor has two MODBUS function codes:

0x03: Read registers 0x10： Write registers

1.4.1 MODBUS Function Code 0x03: Read Registers

This function code is to read a block of continuous registers from a remote device. Request PDU defines start address and number of registers for the read operation. Register addressing starts from 0. Therefore, addresses for register 1-16 are 0-15. Data for each register in Response message have two bytes. For each register data, first byte is for high bits, and second byte for low bits.

Request Frame:

Function code	1 Byte	0x03
Start address	2 Bytes	0x0000….0xfffff
Number of registers	2 Bytes	1…125

Figure 8: Request frame for read registers

Response Frame:

Function code	1 byte	0x03
Number of byte	1 byte	N×2
Register data	N×2 bytes	

N = number of registers

Figure 9: Response frame for read registers

Below is an example of Request and Response frames (Read register 108-110. Register 108 is read only with 2-byte value of 0X022B. Registers 109-110 have values of 0X0000 and 0X0064).

Request Frame		Response Frame	
Data format	Hexadecimal	Data Format	Hexadecimal
Function code	0x03	Function code	0x03
Start address(high bits)	0x00	Number of bytes	0x06
Start address (low bits)	0x6B	Register value (high bits, 108)	0x02
Number of registers (high bits)	0x00	Register value (low bits, 108)	0x2B
Number of registers (low bits)	0x03	Register value (high bits, 109)	0x00
		Register value (low bits, 109)	0x00
		Register value (high bits, 110)	0x00
		Register value (low bits, 110)	0x64

Figure 10: Example of request and response frame for read operation

1.4.2 MODBUS Function Code 0x10: Write Registers

This function code is to write a block of continuous registers at a remote device. Request frame contains register data. Each register data have two character bytes. Response frame contains function code, start address, and number of registers that completed write operation.

Request Frame:

Function code	1 byte	0x10
Start address	2 bytes	0x0000….0xffff
Number of registers	2 bytes	0x0001….0x0078
Number of bytes	1 byte	N×2
Register data	N×2 bytes	value

N = number of registers

Figure 11: Request frame for write operation

Response Frame

Function Code	1 byte	0x10
Start address	2 bytes	0x0000….0xffff
Number of registers	2 bytes	1…123(0x7B)

Figure 12: Response frame for write operation

Below is an example of Request frame and Response frame (write 0x000A and 0x0102 to two registers starting from address 2):

Request Frame		Response Frame	
Data Format	Hexadecimal	Data Format	Hexadecimal
Function code	0x10	Function code	0x10
Start address (high bits)	0x00	Start address (high bits)	0x00
Start address (low bits)	0x01	Start address (low bits)	0x01
Number of registers (high bits)	0x00	Number of registers (high bits)	0x00
Number of registers (low bits)	0x02	Number of registers (low bits)	0x02
Number of bytes	0x04		
Register value (high bits)	0x00		
Register value (low bits)	0x0A		
Register value (high bits)	0x01		
Register value (low bits)	0x02		

Figure 13: Example of Request frame and response frame for write operation

1.5 Data formats in Multi - Parameter sensor

1.5.1 Floating-point number

Definition: floating point number, comply with IEEE754 (single precision)

Note	Sign	Exponent	Fraction	Total
bit	31	30…23	22…0	32
Exponent deviation	127			

Figure 14: Single floating point number definition (4 bytes, 2 MODBIS registers)

Example: Convert decimal number 17.625 to binary number

Step 1: Convert decimal number 17.625 to a floating point number with binary format

First, convert integer to binary

$17_{decimal} = 16 + 1 = 1\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$

Thus, integer 17 in binary format is 10001B

Then convert decimal part to binary

$0.625_{decimal} = 0.5 + 0.125 = 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

Thus, 0.625 in binary format is 0.101B

Combine above together, 17.625 in binary format is 10001.010B

Step 2: Calculate exponent

Left shift the binary number 10001.010B until only bit left before the decimal point --- 10001.101B = 1.0001101 B× 2^4, so exponent value is 4. By adding 127, we have 131, which is 10000011B in binary format

Step 3: Get fraction

Fraction is simply the number after decimal point. Thus from 1.0001101B, fraction number is 0001101B. IMPORTANT NOTE about the 23 bit fraction number: the first bit which on the left side of decimal point is hidden bit and does not need to be compiled.

Step 4: Sign definition

Sign bit is 0 if the number is positive. Sign is 1 if the number is negative. For 17.625, sign bit is 0.

Step 5: Convert to floating point number

1 Sign bit + 8-bit exponent + 23-bit fraction

0 10000011 00011010000000000000000B

(Corresponding hexadecimal number is 0x418D0000)

Sample code:

1. If your compiler has similar library functions, it can be called directly. For example if C language is used, we can directly call memcpy() function in C library to convert floating point number. Sample code:

```
float   floatdata;//floating point data to be converted
void*   outdata;
memcpy(outdata,&floatdata,4);
```

If floatdata=17.625,

In little-endian storage mode after the function is called:

Value at address of outdata is 0x00

Value at address of (outdata+1) is 0x00

Value at address of (outdata+2) is 0x8D

Value at address of (outdata+3) is 0x41

In big-endian storage mode after the function is called:

Value at address of outdata is 0x41

Value at address of (outdata+1) is 0x8D

Value at address of (outdata+2) is 0x00

Value at address of (outdata+3) is 0x00

2. If your complier doesn’t have the conversion function, then the following function can be used:

```
void memcpy(void *dest,void *src,int n)
{
    char *pd = (char *)dest;
    char *ps = (char *)src;
    for(int i=0;i<n;i++)    *pd++ = *ps++;
}
```

Then you can get same result by calling this function memcpy(outdata,&floatdata,4);

Example: Convert binary floating point number 0100 0010 0111 1011 0110 0110 0110 0110B to a decimal number

Step 1: Separate this binary number 0100 0010 0111 1011 0110 0110 0110 0110B and get values of Sign , exponent and fraction.

0 10000100 11110110110011001100110B

1 Sign bit 8-bit exponent 23-bit fraction

Sign bit(s): 0

Exponent(E):10000100B=$1\times2^7+0\times2^6+0\times2^5+0\times2^4+0\times2^3+1\times2^2+0\times2^1+0\times2^0$

=128+0+0+0+0+4+0+0=132

Fraction(M)：11110110110011001100110B =8087142

Step 2: Calculate decimal value

$D = (-1)^S \times (1.0+M/2^{23}) \times 2^{E-127}$

$= (-1)^0 \times (1.0+8087142/2^{23}) \times 2^{132-127}$

$= 1 \times 1.964062452316284 \times 32$

$= 62.85$

Reference code:

```
float floatTOdecimal(long int byte0, long int byte1, long int byte2, long int byte3)
{
        long int realbyte0,realbyte1,realbyte2,realbyte3;
        char S;
        long int     E,M;
        float D;
        realbyte0 = byte3;
        realbyte1 = byte2;
        realbyte2 = byte1;
        realbyte3 = byte0;

        if((realbyte0&0x80)==0)
        {
            S = 0; //Positive
         }
         else
        {
            S = 1; //Negative
        }
        E = ((realbyte0<<1)|(realbyte1&0x80)>>7)-127;
        M = ((realbyte1&0x7f) << 16) | (realbyte2<< 8)| realbyte3;
        D = pow(-1,S)*(1.0 + M/pow(2,23))* pow(2,E);
        return D;
}
```

Note:

- Function parameters byte0, byte1, byte2 and byte3 represent the 4 sections of a binary floating number.
- Return value is value of decimal number after conversion

For example when a command is sent to a sensor to get temperature value, response frame from the sensor will have measured temperature. If the values are 4 byte floating point number 0x00,0x00,0x8d,0x41, then the following function can be used to get temperature in decimal value:

```
float temperature = floatTOdecimal( 0x00, 0x00, 0x8d, 0x41);
```

and temperature = 17.625.

1.5.2 Characters

Definition: Character is shown by ASCII code.

Example: String “YL” could be shown by corresponding ASCII codes (refer to ASCII character chart)

“Y” is 0x59

“L” is 0x4C

2 MODBUS RTU Commands for Multi - Parameter Sensor

2.1 Overview

In order to communicate with Multi-Parameter via MODBUS RTU, master terminal software will be needed. MODBUS RTU is an open standard. There are free commercial software tools available. For applications described in this document, MODBUS register address starts from 1. However, slave address in MODBUS protocol starts from 0, and usually master software compiles addresses. For example, register address 2090 will be compiled by master software as address 2089.

2.2 Command Description

2.2.1 Get measurement data

Function: Get each measurement data for the sensor.

Get measurement data can be get via MODBUS register starting from 0x2601:

Start address	Number of registers	Register 1~16	MODBUS Function code
0x2601	0x10	All Senser value	0x03

Below is an example of request and response frames for sending a get measurement data command with slave address 0x01.

Table 1 Get the measurement data request frame

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x03	0x26	**0x01**	0x00	**0x10**	

Table 2 Get the measurement data response frame

Definition	Slave id	Function	Count	Value	CRC
Byte	0	1	2	3 ~ 38	39 ~ 40
Value	0x01	0x03	**0x20**	Value	

Value is as follow:

Byte	3 ~ 6	7 ~ 10	11 ~ 14	15 ~ 18	19 ~ 22	23 ~ 26	27 ~ 30	31 ~ 34
Value	DO (mg / L)	Tur	CT	pH	Temp	Orp	Chl	BGA

If the probe is not exist , the corresponding value is 0.

For example, DO value is 7.90 mg/L:

Byte	3	4	5	6
Value	CD	CC	FC	40

2.2.2 Get SN

Function: Get sensor’s serial number (SN). Each sensor probe has a unique SN.

Serial Number can be read from 6 continuous MODBUS registers starting from address 0x1400

Start Address	Number of registers	Register 1-6	MODBUS Function code
0x1400	0x06	SN	0x03

The frame structure is as follow:

Table 3 Get SN request frame

Definition	Slave id	Function	Start address		Quantity		CRC
byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x03	0x14	**0x00**	0x00	**0x06**	

Table 4 Get SN response frame

Definition	Slave id	Function	Count	Value	CRC
Byte	0	1	2	3 ~ 14	15 ~ 16
Value	0x01	0x03	**0x0C**	“YL0114010022”	

Note: SN value is in ASCII code as below:

Byte	4	5	6	7	8	9	10	11	12	13	14	15
Value	0x59	0x4C	0x30	0x31	0x31	0x34	0x30	0x31	0x30	0x30	0x32	0x32

2.2.3 Error Flag

Function: Ask for sensor error, default value 00 means correct.

Error Flag can be read via MODBUS registers address 0x0800

Start Address	Number of registers	Register 1	MODBUS Function code
0x0800	0x01	Error Flag	0x03

The frame structure is as follow:

Table 5 Get Error Flag request frame

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x03	0x08	**0x00**	0x00	**0x01**	

Table 6 Get Error Flag response frame

Definition	Slave id	Function	Count	Value		CRC
Byte	0	1	2	3	4	5 ~ 6
Value	0x01	0x03	**0x02**	Flag		

Error	none	Brush problem	Humidity communication	12V voltage problem	Sensor is in water
Flag	0x00	0x01	0x02	0x04	0x10

2.2.4 Get Brush Interval of time

Function: Get the interval of time for brush between each rotation, default time is 30 min.

The interval of time can be read from 1 MODBUS registers starting from address 0x0E00.

Start address	Number of registers	Register 1	MODBUS function code
0x0E00	0x01	Interval of time(min)	0x03

The frame structure is as follow:

Table 7 Request frame to get brush Interval of time

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x03	0x0E	**0x00**	0x00	**0x01**	

Table 8 Response frame to get brush Interval of time

Definition	Slave id	Function	Count	Value		CRC
Byte	0	1	2	3	4	5 ~ 6
Value	0x01	0x03	**0x02**	1E	00	

The rotation time is (00,1E) = 30min.

2.2.5 Set Brush Interval of time

Function: Set the interval of time for brush between each rotation, the unit is minute.

The interval of time can be set at 1 MODBUS registers starting from address 0x0E00.

Start address	Number of registers	Register 1	MODBUS function code
0x0E00	0x01	Interval of time(min)	0x10

Below is an example of request and response frames for setting brush interval of time, assuming slave address is 0x01, time is 10min.

Table 9 Set brush Interval of time request frame

Definition	Slave id	Function	Start address		Quantity		Count	Value		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6	7	8	9 ~ 10
Value	0x01	0x10	0x0E	0x00	0x00	0x01	0x02	0x0a	0x00	

Table 10 Set brush Interval of time response frame

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x10	0x0E	**0x00**	0x00	0x01	

2.2.6 Set the time

Function: Set the Sensor real time.

The real time can be set at 1 MODBUS registers starting from address 0x1300.

Start address	Number of registers	Register 1-4	MODBUS function code
0x1300	0x04	Real time	0x10

The frame structure is as follow:

Table 11 Set the time request frame

Definition	Slave id	Function	Start address		Quantity		Count	Value	CRC
Byte	0	1	2 ~ 3		4 ~ 5		6	7 ~ 14	15 ~ 16
Value	0x01	0x10	0x13	0x00	0x00	0x04	0x08	Time	

Table 12 Set the time response frame

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x10	0x13	0x00	0x00	0x04	

Time Format structure:

byte	7	8	9	10	11	12	13	14
Meaning	Second	Minute	Hour	Day	Month	00	Year	00

Time is displayed in BCD type, for example, 0x59 means 59 seconds

2.2.7 Get the time

Function: Get the Sensor real time.

The real time can be get at 1 MODBUS registers starting from address 0x1300.

Start address	Number of registers	Register 1-4	MODBUS function code
0x1300	0x04	Real time	0x03

Below is an example of request and response frames for getting real time from a device with slave address 0x01, assuming returned time is 16.04.26 13:05:17.

Table 13 Read the time request frame

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x03	0x13	**0x00**	0x00	**0x04**	

Table 14 Read the time response frame

Definition	Slave id	Function	Count	Value	CRC
Byte	0	1	2	3 ~ 10	11 ~ 12
Value	0x01	0x03	**0x08**	Time	

Byte	3	4	5	6	7	8	9	10
Meaning	0x17	0x05	0x13	0x26	0x04	0x00	0x16	00

2.2.8 Set Slave Device ID

Function: Set MODBUS slave address to a sensor probe. Range of address is 1~247.

Sensor probe slave address can be set via MODBUS register 0x3000:

Start address	Number of registers	Register 1	MODBUS Function code
0x3000	0x01	New Slave address	0x10

Below is an example of request and response frames for setting slave device ID command. Old slave address is 0x01, new address is 0x14.

Table 15 Set Slave Device ID request frame

Definition	Slave id	Function	Start address		Quantity		Count	Value		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6	7	8	9 ~ 10
Value	0x01	0x10	0x30	0x00	0x00	**0x01**	**0x02**	0x14	0	

Table 16 Set Slave Device ID response frame

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x10	0x30	0x00	0x00	**0x01**	

2.2.9 Get the supply voltage

Function: Get the current supply voltage value, in unit of V.

Supply voltage can be get via MODBUS register 0x1E00:

Start address	Number of registers	Register 1	MODBUS Function code
0x1E00	0x01	the supply voltage	0x03

The frame structure is as follow, assuming slave address is 0x01, the supply voltage is 12.31V:

Table 17 Get the power supply voltage request frame

Definition	Slave id	Function	Start address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x03	0x1E	**0x00**	0x00	**0x02**	

Table 18 Get the power supply voltage response frame

Definition	Slave id	Function	Count	Value				CRC
Byte	0	1	2	3	4	5	6	7 ~ 8
Value	0x01	0x03	**0x04**	0xC3	0xF5	0x44	0x41	

2.2.10 Active Brush

Function: Make brush rotate. It is strongly recommend that brush should be active once power on.

MODBUS register 0x2F00 is used.

Start address	Number of registers	MODBUS function code
0x2F00	0x00	0x10

Below is an example of request and response frames for a device with slave address 0x01 to active brush.

Table 19 Active brush request frame

Definition	Slave id	Function	Start address		Quantity		Count	CRC
Byte	0	1	2 ~ 3		4 ~ 5		6	9 ~ 10
Value	0x01	0x10	0x2F	0x00	0x00	0x00	0x00	

Table 20 Active brush response frame

Definition	Slave id	Function	Address		Quantity		CRC
Byte	0	1	2 ~ 3		4 ~ 5		6 ~ 7
Value	0x01	0x10	0x2F	0x00	0x00	0x00	

3 Procedure to get all value

When you need to get the value of one or more probes, send the corresponding frame command in turn.

Process **Corresponding frame command**

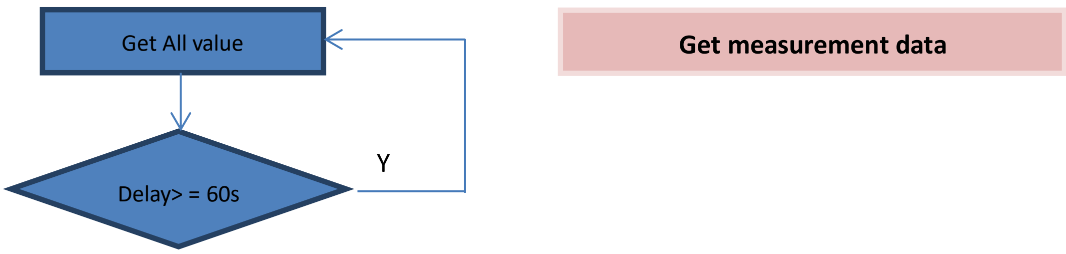

Figure 15 Get the flowchart for each measured value