HPS-167S

Ultra-High Performance Waterproof Time-of-Flight (ToF) Sensor

Features

- Fast and accurate distance measurement
  - Maximum measurement distance up to 18m (white target)
  - Output data rate up to 54Hz
  - Measurement results are insensitive to target color and reflectivity
  - Electrical and optical crosstalk compensation
  - -10℃~+55℃ temperature compensation
  - Built-in ambient light compensation function allows the sensor to work in high infrared background light environments

- Fully integrated miniature module
  - ABS waterproof shell + aviation aluminum heat dissipation back panel
  - 850nm infrared LED emitter
  - Emitter driver
  - Integrated optimally-designed emitting & receiving optical lens
  - Ranging sensor with advanced embedded micro controller
  - Advanced embedded data processing & filtering algorithm
  - 19200 bps RS485 bus interface
  - 53(L) x 32(W) x 26.5(H) mm, 27g
  - Compliant with latest CE, FCC, RoHS standards

Applications

- Drone (obstacle avoidance, hovering at a fixed height, soft landing)
- Robotics & AGV (obstacle detection)
- Industrial location and proximity sensing
- Security and surveillance
- 1D gesture recognition

Description

HPS-167S is a new generation of ultra-high performance waterproof time-of-flight (ToF) infrared ranging sensor, equipped with optimized transmitting and receiving optical lenses, suitable for high-precision, long-distance ranging occasions. Different from traditional technologies, its measurement accuracy is not affected by the color and reflectivity of the measured target. When the target is a white target with a reflectivity of 90%, the measurement distance of HPS-167S can reach 18 meters, setting a new milestone in the field of infrared ranging and expanding a wider range of application scenarios.

HPS-167S adopts ABS waterproof shell + hard anodized aviation aluminum heat dissipation backplane, and integrates a high-power 850nm infrared LED transmitter and a high-sensitivity photodiode receiver. In addition, the internal integrated physical infrared filter enables it to measure farther and has stronger resistance to ambient light interference.

Advanced embedded data processing & filtering algorithm realizes extremely stable and real-time measurement outputs.

AVOID EYE OR SKIN EXPOSURE TO DIRECT OR SCATTERED RADIATION

1. Overview

1.1 Technical specification

Table 1. Technical specification

Parameter	Values	Unit
Size	53（L）x 32（W）x 26.5（H）	mm
Weight	27 *1	g
Power supply	5 ~ 15	V
Maximum power consumption	1.4	W
Quiescent power consumption	0.1	W
Storage temperature	-40 ~ 85	℃
Operating temperature	-10 ~ 55 *2	℃
Infrared LED emission wavelength	850	nm
Emitting angle	±1.8	°
Maximum measuring distance	18 *3	m
Minimum measuring distance	0.08	m
Output data rate	6~54	Hz
Output data	Distance, accuracy, signal strength, ambient infrared light intensity,, temperature	-
Electrical output	Four-wire output, default line length 70cm, bare wire hot tinning or cold pressing terminal	-
Communication Interface	RS485, 19200bps	

Note:

*1 Cable not included.

*2 When performing long-distance measurements in continuous working mode (distance greater than 20 meters), HPS-167S needs a few seconds warm-up time to stabilize the output.

*3 Tested on 90% reflectance white target.

1.1 Dimensions and pin definition

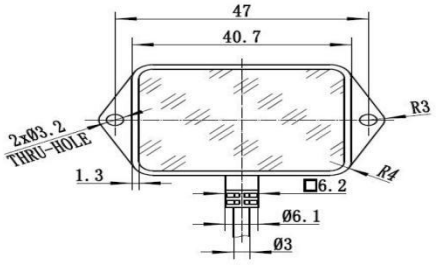


Figure 1. Front view of HPS-167S

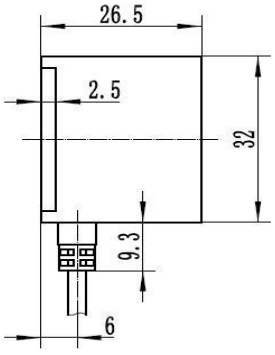


Figure 2. Side view of HPS-167S

Table 2. HPS-167S Line Sequence Definition

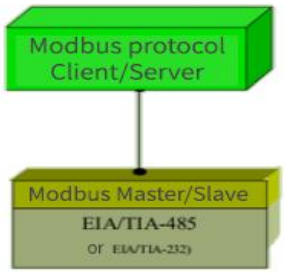
Line Color	Signal name	Signal type	Description
Black	GND	GND	Ground
White	A	Digital	RS485 bus A terminal
Green	B	Digital	RS485 bus B terminal
Red	VDD	Power	The positive pole of the power supply is connected to DC +5V~+15V

2. MODBUS_RTU Communication Protocol

2.1 Introduction

MODBUS is an application layer messaging protocol on layer 7 of the OSI model that provides client/server communication between devices connected to different types of buses or networks. It also standardizes the protocol on a serial link to exchange Modbus requests between a master and one or more slaves.

The following figure shows the general relationship of the Modbus serial communication stack corresponding to the 7-layer OSI model.



Layer	ISO/OSI Model	
7	Application Layer	Modbus Protocol
6	Presentation Layer	Null
5	Session Layer	Null
4	Transport Layer	Null
3	Network Layer	Null
2	Data Link Layer	Modbus Serial Link Protocol
1	Physical Layer	EIA/TIA-485 (or EIA/TIA-232)

Figure 3. Modbus protocol and ISO/OSI model

2.2 Modbus Data Link Layer

2.2.1 Modbus Host/Slave Protocol Principle

Modbus is a host-slave communication mode communication protocol, that is, Modbus has a host, which can connect multiple slaves (up to 32). The host is connected to the bus, and one or more slaves are connected to the same serial bus.

In Modbus, the host can actively communicate, and other slaves can only respond to the host but cannot actively send data to the communication bus, and the slaves cannot communicate with each other. This method stipulates the communication order and other relationships during the communication process, avoiding the generation of communication conflicts when multiple devices are working at the same time.

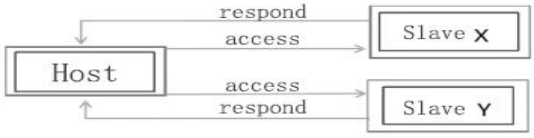


Figure 4. Host/Slave Communication

In Modbus, the host can send requests to the slave in two modes:

- Unicast mode: The host station accesses a slave station through a specific address (device address or sensor ID). After the slave station receives and processes the request, the slave station returns a message (a 'response') to the slave station.
  Each slave must have a unique address or ID (1 to 32) so that it can be addressed independently from other slaves.

- Broadcast mode: The host sends requests to all slaves.
  For the request sent by the host in broadcast mode, the slave does not respond. Therefore, broadcast requests are generally used for write commands. All devices must accept the write function of broadcast mode. Address 0x00 represents the broadcast address.

2.2.2 Modbus serial transmission mode

Modbus has two transmission modes: RTU mode and ASCII mode. It determines how information is packaged into messages and decoded.
This product uses the RTU mode.

RTU mode:
The main advantage of this mode is higher data density, which allows higher throughput than ASCII mode at the same baud rate.
Each message must be transmitted as a continuous stream of characters.
The format of each byte (11 bits) in RTU mode is:

Table 3. RTU mode bit sequence

Start Bit	Data Bit								Parity Bit	Stop Bit
1 bit	1	2	3	4	5	6	7	8	1 bit	1 bit

Note: Data bits: Least significant bit is sent first (from left to right).
Parity bit: Even parity is required and is the default mode, other modes (odd parity, no parity) can also be used.

2.2.3 Modbus RTU message frame description

Table 4. RTU mode message frame

Address field	Function code	Data	CRC
1 byte	1 byte	0~252	2 bytes

Address field: generally refers to the device code, here it refers to the sensor ID or device address. The host has no address field, but each slave has an independent address field. Through the address field, the host can communicate data with multiple slaves and avoid other devices from responding incorrectly to communications that do not belong to the device.

Modbus messages are constructed by the sending device as frames with known start and end markers. This allows the device to receive a new frame at the beginning of a message and know when the message ends. Incomplete messages must be detected and the error flag must be set as a result.

In RTU mode, message frames are separated by idle intervals of at least 3.5 character times.

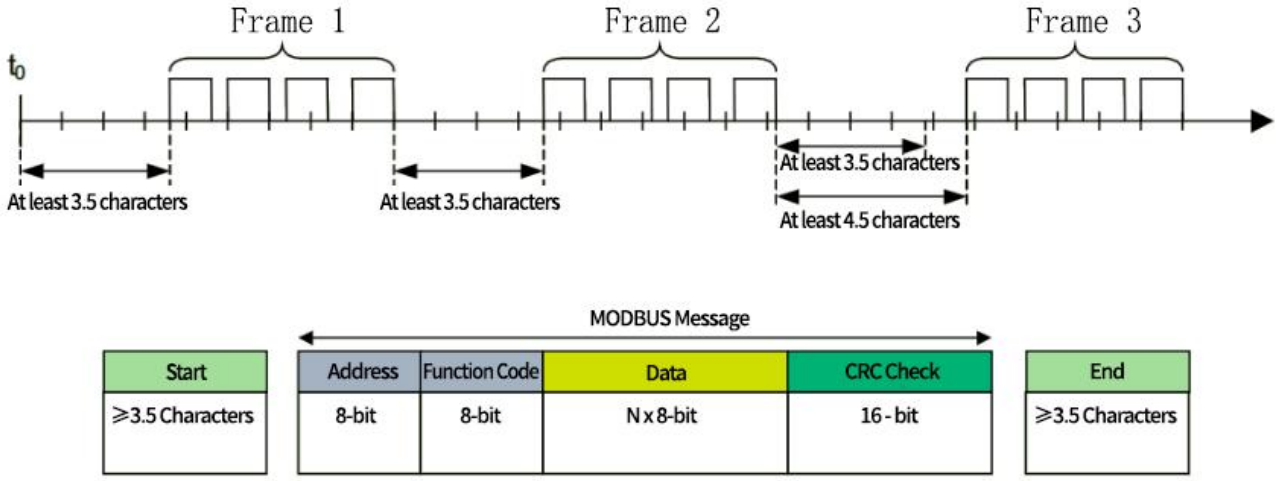


Figure 5. RTU Message Frame

The entire message frame must be sent as a continuous stream of characters.

If the idle interval between two characters is greater than 1.5 character times, the message frame is considered incomplete and should be discarded by the receiving node.

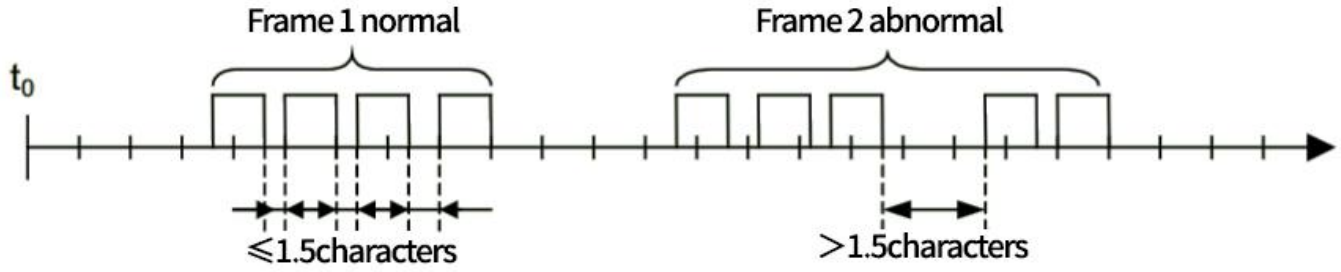


Figure 6. RTU Message Frame 2

Note:

The implementation of the RTU receive driver implies a lot of interrupt management due to the timing of t1.5 and t3.5. At high communication rates, this leads to a heavy CPU burden. Therefore, when the communication rate is equal to or lower than 19200 Bps, these two timings must be strictly followed; for baud rates greater than 19200 Bps, two fixed values of timing should be used: the recommended inter-character timeout (t1.5) is 750μs, and the inter-frame timeout (t1.5) is 1.750ms.

2.2.4 Function code description

The meaning of the function code is to indicate the function or purpose of the message frame.

There are three main types of function codes: public function codes, user-defined function codes, and reserved function codes.

This product only uses public function codes.

Figure 5. Public Function Codes

Function code	Name	Description
0x03	Read Holding Registers	This function code reads the contents of a continuous block of holding registers.
0x06	Writing a single register	This function code writes a single holding register
0x10	Writing multiple registers	This function code writes a continuous block of registers (1 to about 120 registers)

2.2.4.1 (0x03) Reading the saved register

The host sends a request:

Address field	Function code	Initial address	Number of registers	CRC Check
1 byte	1 byte	2 bytes	2 bytes	2 bytes
1 to 32	0x03	0x0000 to 0xFFFF	1 to 125 (0x7D)	

Slave response:

Address field	Function code	Number of bytes	Register Value	CRC Check
1 byte	1byte	1byte	N X 2 bytes	2 byte
1 to 32	0x03	2 X N		

N = Number of registers

Slave error response:

Address field	Error code	Exception code	CRC Check
1 byte	1 byte	1 byte	2 byte
1 to 32	0x83	01 or 02 or 03 or 04	

For details, please see: Command# Abnormal response code

For example: Function code 03 indicates that the host requires to read the values of several registers of the slave. After receiving the communication frame, the slave performs a series of processing and returns the same function code to the host, indicating that the slave responds to the function.

In addition, if there is an error in the data frame, such as incorrect format, missing transmission, incorrect verification, etc., when the slave returns the function code, it will add 128 (0x80 | 0x03) to the original function code to indicate which function code is wrong. After receiving it, the host can make an error judgment based on the function code returned at this time. Therefore, the function code not only expresses the function of the frame, but also indicates whether the communication is wrong.

2.2.4.2 (0x06) Write a single register

The host sends a request:

Address field	Function code	Register Address	Register Value	CRC Check
1 byte	1 byte	2 bytes	2 bytes	2 bytes
1 to 32	0x06	0x0000 to 0xFFFF	0x0000 to 0xFFFF	

Slave response:

Address field	Function code	Register Address	Register Value	CRC Check
1 byte	1 byte	2 byte	2 byte	2 byte
1 to 32	0x06	0x0000 to 0xFFFF	0x0000 to 0xFFFF	

N = Number of registers

Slave error response:

Address field	Error code	Exception code	CRC Check
1 byte	1 byte	1 byte	2 bytes
1 to 32	0x86	01 or 02 or 03 or 04	

For details, please see: Command# Abnormal response code

2.2.4.3 (0x10) Writing multiple registers

The host sends a request:

Address field	Function code	Initial address	Number of registers	Number of bytes	Register Value	CRC Check
1 byte	1 byte	2 bytes	2 bytes	1 byte	N X 2 bytes	2 bytes
1 to 32	0x10	0x0000 to 0xFFFF	0x0001 to 0x0078	2 X N	Value	

N = Number of registers

Slave response:

Address field	Function code	Initial address	Number of registers	CRC Check
1 byte	1 byte	2 bytes	2 bytes	2 bytes
1 to 32	0x10	0x0000 to 0xFFFF	1 to 123（0x7B）	

N = Number of registers

Slave error response:

Address field	Error code	Exception code	CRC Check
1 byte	1 byte	1 byte	2 bytes
1 to 32	0x90	01 or 02 or 03 or 04	

For details, please see: Command# Abnormal response code

2.2.5 CRC Check

In RTU mode, it contains an error check field based on the cyclic redundancy check (CRC) algorithm that is performed on the entire message content. The CRC field checks the contents of the entire message. This check is performed regardless of whether the message has a parity check or not.

The CRC consists of a 16-bit value composed of two 8-bit bytes.

The CRC field is appended as the last field of the message. After calculation, the low byte is appended first, followed by the high byte.

The CRC value attached to the message is calculated by the sending device. The receiving device recalculates the CRC value when receiving the message and compares the calculated result with the actual received CRC value. If the two values are not equal, it is an error.

CRC calculation:

Start by preloading all 1s into a 16-bit register. Then perform subsequent calculations on the consecutive 8-bit sub-bytes in the message. Only the 8 data bits in the character participate in the CRC calculation, and the start bit, stop bit and check bit do not participate in the CRC calculation.

During the CRC generation process, each 8-bit character is XORed with the value in the register. The result is then shifted by 1 bit in the direction of the least significant bit (LSB), and the most significant bit (MSB) position is filled with zero. The LSB is then extracted and checked: if the LSB is 1, the value in the register is XORed with a fixed preset value; if the LSB is 0, no XOR operation is performed.

This process will be repeated until 8 shifts are performed. After the last (8th) shift and related operations are completed, the next 8-bit byte is XORed with the current value of the register, and then repeated 8 times as described above. The final value of the register obtained after all sub-bytes in the message are calculated is the CRC.

When the CRC is appended to the message, the low byte is appended first, followed by the high byte. A detailed example of CRC generation is included in the appendix.

The process of generating CRC is:

1. Load a 16-bit register with hexadecimal FFFF (all 1s) and call it the CRC register.
2. XOR the first 8-bit byte of the message with the low byte of the 16-bit CRC register and place the result in the CRC register.
3. Shift the CRC register right by 1 bit (toward the LSB), fill the MSB with zeros, extract and detect the LSB.
4. (If the LSB is 0): Repeat step 3 (another shift).

(If the LSB is 1): XOR the CRC register with the polynomial value 0xA001 (1010 0000 0000 0001).

5. Repeat steps 3 and 4 until 8 shifts are completed. When this operation is completed, the complete operation on the 8-bit byte is completed.
6. Repeat steps 2 to 5 for the next byte in the message, and continue this operation until all messages are processed.
7. The final content of the CRC register is the CRC value.
8. When placing the CRC value in the message, the high and low bytes must be swapped as described below.

Place CRC in the message

When the 16-bit CRC (2 8-bit bytes) is transmitted in a message, the low-order byte is sent first, followed by the high-order byte.

For example, if the CRC value is hexadecimal 1241 (0001 0010 0100 0001):

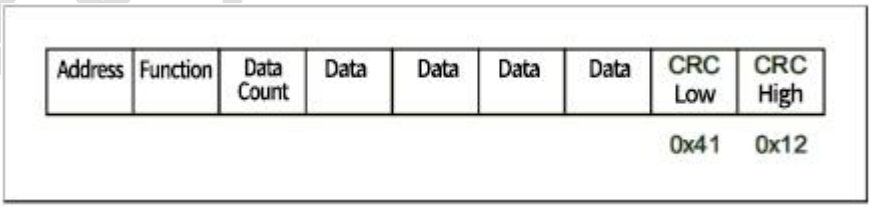


Figure 7. CRC byte sequence

2.3 Communication parameter configuration

The device interface is RS485. It uses MODBUS protocol-RTU mode. (8 data bits, the least significant bit is sent first)

Table 6. Master/Slave Parameter Configuration

Baud rate	19200bps
Start bit	1bit
Data bits	8bits

Stop bits	1bit
Parity bit	Even parity
CRC Check	ModbusCRC16

The baud rate can also be 57600bps, 115200bps, and 230400bps.

2.4 Communication Model

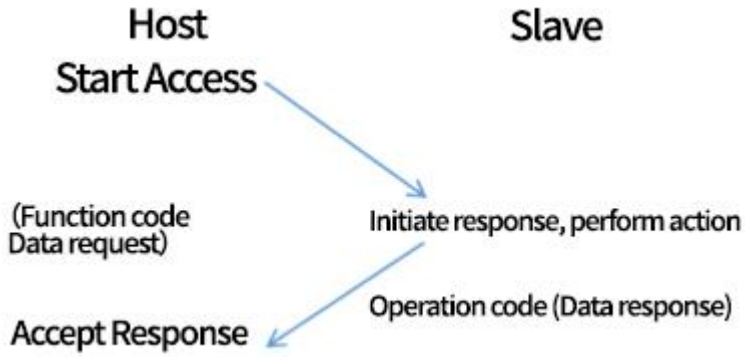


Figure 8. Communication model

2.5 Data frame format

Table 7. MODBUS data frame format

Device ID	Function code	Data	CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4
1 byte	1 byte	0 ~ 256 bytes	1 byte	1 byte

When the host sends a command to the sensor, the correct sensor ID must be set, otherwise the sensor will not respond; after receiving the correct command, the sensor will perform the corresponding operation and return the result of the operation. When the communication is normal, each command sent by the host will have return data.

3. Command List

Table 8. Modbus_RS485 command list

Command	Device Address	Function code	Initial address	Description
Read RS485 version information	Device ID (1~32)	0x03	0x01	Get the software version information of the RS485 interface firmware
Get RS485 configuration information			0x02	Get the configuration information of the RS485 interface firmware
Read sensor details			0x03	Read sensor details
Get the temperature of the sensor (AFE)			0x04	Get the current temperature of the sensor analog front end
Single measurement			0x08	Will start the sensor for a single

				measurement
Restore the RS485 interface firmware to factory settings			0x09	Restore all settings of the RS485 interface to factory settings
Sensor warm-up time		0x06	0x0A	Will set the sensor warm-up time
Set the sensor ID			0x10	The sensor ID on the RS485 bus can be modified
Set RS485 communication baud rate			0x11	Used to set the RS485 communication baud rate
Set the measurement mode after the sensor is powered on			0x12	Set whether to automatically enter the continuous measurement mode after power on
Set the 485 bus terminal resistance			0x13	Set whether to automatically start the bus terminal resistance after power on
Set the measurement distance deviation compensation value			0x14	Can be used to compensate for inherent measurement deviations of sensors
Load sensor settings			0x15	The settings saved in the sensor will be loaded.
Set the sensor output filter sensitivity			0x16	Set the output filter sensitivity inside the sensor
Set sensor reconnection time		0x10	0x20	Set sensor reconnection time

Command #1 Get the version information of the RS485 interface firmware

This command can be used to obtain the software version information of the RS485 interface firmware.

Table 9. Commands for obtaining the version information of the RS485 interface firmware

Device ID	Function code	Initial address		Number of registers		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x01	0x00	0x03	0x54	0x0B

Note: The device ID used here is 0x01.

Return data:

Table 10. Return data of the command to obtain the version information of the RS485 interface firmware

Byte number	Name	Date	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x03	Function code
2	Date Length	0x06	Number of bytes = number of registers*2
3	Year	.....	Production date of the adapter plate
4	Month	.....	
5	Day	.....	
6	Major version	.....	Major version number
7	Minor version	.....	From the version number
8	Rev	.....	Amendment No
9	CRC MSB	.....	ModbusCRC16 Check high byte

10	CRC LSB	.....	ModbusCRC16 Check low byte

Command #2 Get the configuration information of the RS485 interface firmware

This command is to obtain the configuration information of the RS485 interface firmware.

Table 11. Commands for obtaining RS485 interface firmware configuration information

Device ID	Function code	Initial address		Number of registers		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x02	0x00	0x04	0xE5	0xC9

Note: The device ID used here is 0x01.

Return data:

Table 12. Command return data for obtaining RS485 interface firmware configuration information

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x03	Function code
2	Date Length	0x08	Number of bytes = number of registers*2
3	Resistance	.....	Terminal resistance selection, 0x00 means not enabled; 0xFF means enabled
4	Auto_out	.....	Whether to enter the continuous output mode after initialization, 0x00 means not enabled; 0xFF means enabled
5	Preheat time	.....	Warm-up time, unit: seconds
6	Connect_time	.....	Time required for RS485 to resume communication under continuous output, unit: milliseconds
7		.....	
8		.....	
9		.....	
10	Baud rate	.....	RS485 baud rate, default: 19200
11	CRC MSB	.....	ModbusCRC16 check high byte
12	CRC LSB	.....	Modbus CRC16 check high byte

Command #3 Read sensor details

This command will read the detailed information of the sensor.

Table 13. Commands for reading detailed information of sensors

Device ID	Function code	Initial address		Number of registers		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x03	0x00	0x0B	0xF4	0x0D

Note: The device ID used here is 0x01.

Return data:

Table 14. Return data for the command to read detailed information of a sensor

Byte	Name	Data	Description

number			
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x03	Function code
2	Date Length	0x16	Number of bytes = number of registers*2
3	ACK Byte	0xB0	The response byte returned by the sensor
4~19	UUID	.....	Universally unique identifier of the sensor
20	Year	.....	Year
21	Month	.....	Month
22	Day	.....	Day
23	Major version	.....	The version number of the sensor
24	Minor version	.....	
25	CRC MSB	.....	ModbusCRC16 check high byte
26	CRC LSB	.....	ModbusCRC16 check low byte

The following is an example of the parsing of the returned sensor detailed information data:
Return data packet: 0x01 0x03 0x16 0xB0 0x00 0x39 0x00 0x25 0x42 0x34 0x57 0x14 0x20 0x34 0x38 0x35 0x01 0x06 0x07 0x05 0x13 0x09 0x17 0x03 0x09 0x1A 0xDC

Parse:
0x01: Device ID
0x03: function code
0x16: Number of bytes = number of registers*2
0xB0: Response Byte
0x00 0x39 0x00 0x25 0x42 0x34 0x57 0x14 0x20 0x34 0x38 0x35 0x01 0x06 0x07 0x05: Universally unique identifier
0x13 0x09 0x17: 19/09/27
0x03 0x09: Ver. 2.1
0x1A 0xDC: ModBus CRC16 check

Command #4 Gets the temperature of the sensor analog front end (AFE)

This command can get the current temperature of the sensor analog front end. (Unit: Fahrenheit).

Table 15. Get Sensor Analog Front End (AFE) Temperature Commands

Device ID	Function code	Initial address		Number of registers		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x04	0x00	0x01	0xC5	0xCB

Note: The device ID used here is 0x01.

Return data:

Table 16. Get Sensor Analog Front End (AFE) Temperature Command Return Data

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x03	Function code
2	Date Length	0x02	Number of bytes = number of registers*2
3	AFE_temp MSB	.....	Get the high byte of the AFE temperature value
4	AFE_temp LSB	.....	Get the low byte of the AFE temperature value
5	CRC MSB	.....	ModbusCRC16 check high byte
6	CRC LSB	.....	ModbusCRC16 check low byte

Command #5 Single measurement

This command starts the sensor taking a single measurement.

Table 17. Single measurement commands

Device ID	Function code	Initial address		Number of registers		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x08	0x00	0x04	0xC5	0xCB

Note: The device ID used here is 0x01.

Continuous measurement: sending and receiving a single measurement command multiple times.The maximum data output rate in continuous measurement mode is 54 Hz. If a sensor failure occurs during continuous measurement, an abnormal data packet will be returned (the values of Distance, Ambient, and Precision are all "0xFF" and the value of Magnitude is "0x00").

Return data:

Table 18. Return data of single measurement command

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x03	Function code
2	Date Length	0x08	Number of bytes = number of registers*2
3	Distance MSB	.....	Measuring distance, unit: mm
4	Distance LSB	.....	
5	Magnitude MSB	.....	Receive reflected light signal strength
6	Magnitude LSB	.....	

7	Magnitude Exp	.....	Received signal strength index byte
8	Ambient ADC	.....	Relative ambient infrared light intensity
9	Precision MSB	.....	Accuracy indicator value, the smaller the value, the higher the measurement accuracy
10	Precision LSB	.....	
11	CRC MSB	.....	ModbusCRC16 check high byte
12	CRC LSB	.....	ModbusCRC16 check low byte

Note: When the sensor is out of range or the reflected signal strength is too low, an over-range indication of 65.53 meters will be output.

The following is an example of parsing the measurement return data:

Return data packet: 0x01 0x03 0x08 0x08 0x23 0xDC 0xB2 0x07 0x01 0x00 0x00 0xFD 0x41

Parse:

0x01: Device ID

0x03: function code

0x08: Number of bytes = number of registers*2

Measuring distance Distance = (0x08 * 256 + 0x23) / 1000.0f = 2.083 (Unit: m)

Received signal strength Magnitude = ((0xDC * 256 + 0xB2) << 0x07) / 10000.0f = 723.1744

Ambient infrared light intensity Ambient ADC = 1

Accuracy indication Precision = (0x00 * 256) + 0x00 = 0

0xFD 0x41: ModBus CRC16 check

Command #6 Restore RS485 interface firmware to factory settings

This command can restore all settings in the sensor RS485 interface firmware to factory settings. After executing this command, the sensor ID number on the bus will be reset to "0x01".

Table 19. Restore RS485 interface firmware factory settings command

Device ID	Function code	Initial address		Number of registers		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x09	0x00	0x01	0x54	0x08

Note: The device ID used here is 0x01.

Return data:

Table 20. Return data of the command to restore the RS485 interface firmware to factory settings

Byte number	Name	Date	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x03	Function code
2	Date Length	0x02	Number of bytes = number of registers*2
3	.....	0x00	.....
4	Success Flag	0x01	If 0x01 is displayed, it means the operation is successful, otherwise an error code will be returned.
5	CRC MSB	0x79	ModbusCRC16 check high byte
6	CRC LSB	0x84	ModbusCRC16 check low byte

Command #7 Set the sensor warm-up time

The default startup warm-up time of the sensor is 5 seconds. During the warm-up time, the sensor will not output measurement data. This command can modify the warm-up time after the sensor is powered on. The set value after executing this command will be automatically written into the FLASH inside the sensor and will not be lost when the power is off.

Table 21. Commands for setting the sensor warm-up time

Device ID	Function code	Register address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x0A	0x00	0x05	0x69	0xCB

Note: The device ID used here is 0x01.

Return data:

Table 22. Return data for setting the sensor startup warm-up time

Byte number	Name	Data	Description
0	Device ID	0x01	Unique device identification ID
1	Function Code	0x06	function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x0A	Low byte of the register address
4	Register data MSB	0x00	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	0x05	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	0x69	ModbusCRC16 check high byte
7	CRC LSB	0xCB	ModbusCRC16 check low byte

Command #8 Set sensor ID

This command can modify the sensor's ID number on the RS485 bus (the factory default value is "0x01"). This command will take effect immediately after setting, and the set ID number will be automatically written into the FLASH inside the sensor and will not be lost when the power is off.

Table 23. Set Sensor ID Command

Device ID	Function code	Register address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x10	.....	.....	.....	.....

Note: The device ID used is 0x01.。

Return data:

Table 24. Set Sensor ID Command Return Data

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x06	Function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x10	Low byte of register address

4	Register data MSB	.....	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	.....	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	.....	ModbusCRC16 check high byte
7	CRC LSB	.....	ModbusCRC16 check low byte

Command #9 Set RS485 communication baud rate

This command can be used to set the RS485 communication baud rate.

Table 25. Commands for setting the communication baud rate

Device ID	Function code	Register Address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x11	0x00	0x01	0x18	0x0F

Note: The device ID is 0x01. The default baud rate is 19200bps, where 01 represents 19200, 02 represents 57600, 03 represents 115200, 04 represents 230400, and the others are all 19200bps.

Return data:

Table 26. Return data of the command to set the communication baud rate

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x06	Function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x11	Low byte of register address
4	Register data MSB	0x00	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	0x01	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	0x18	ModbusCRC16 check high byte
7	CRC LSB	0x0F	ModbusCRC16 check low byte

Command #10 Sets the measurement mode after the sensor is powered on

This command can set whether the sensor automatically enters the continuous measurement mode after power is turned on. The factory default is enabled. After this command is set, it will take effect the next time the power is turned on. The set value will be automatically written to the FLASH inside the sensor and will not be lost when the power is turned off.

Table 27. Commands to set the measurement mode after the sensor is powered on

Device ID	Function code	Register address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x12	0x00	0x00	0x29	0xCF

Note: The device ID used here is 0x01, where 0x00 means not enabled and 0xFF means enabled.

Return data:

Table 28. Return data of the command to set the communication baud rate

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x06	Function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x12	Low byte of the register address
4	Register data MSB	0x00	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	0x00	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	0x29	ModbusCRC16 check high byte
7	CRC LSB	0xCF	ModbusCRC16 check low byte

Command #11 Set RS485 bus terminal resistance

This command can set whether to automatically start the RS485 bus terminal resistance after the sensor is powered on. The factory default is enabled. After this command is set, it will take effect at the next power-on. The set value will be automatically written to the FLASH inside the sensor and will not be lost when the power is off.

Table 29. Commands for setting RS485 bus terminal resistance

Device ID	Function code	Register Address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0xRegister Address13	0x00	0xFF	0x38	0x4F

Note: The device ID used here is 0x01, where 0x00 means not enabled and 0xFF means enabled.

Return data:

Table 30. Return data of command to set RS485 bus terminal resistance

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x06	Function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x13	Low byte of register address
4	Register data MSB	0x00	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	0xFF	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	0x38	ModbusCRC16 check high byte
7	CRC LSB	0x4F	ModbusCRC16 check low byte

Command #12 Set the measurement distance deviation compensation value

This command can be used to compensate for the inherent measurement deviation of the sensor.

Table 31. Commands for setting the measurement distance deviation compensation value

Device ID	Function code	Register Address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x14	.....	.....	.....	.....

Note: The device ID used here is 0x01.

Return data:

Table 32. Return data of command to set measurement distance deviation compensation value

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x06	Function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x14	Low byte of register address
4	Register data MSB	.....	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	.....	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	.....	ModbusCRC16 check high byte
7	CRC LSB	.....	ModbusCRC16 check low byte

Example:

real distance：200 mm, sensor measuring distance: 215 mm

Distance deviation = 200 – 215 = -15 = 0xFFF1 (deviation high byte = 0xFF, deviation low byte = 0xF1)

Note: Due to the performance differences of individual sensors, this command can be used to compensate for small-scale measurement deviations to achieve higher measurement accuracy. The deviation value set by calling this command will be automatically saved to the sensor's Flash memory and automatically loaded each time the sensor is powered on.

Command #13 Load sensor settings

Executing this command will load the setting parameters saved inside the sensor. After the setting parameters are loaded, they will not be lost when the power is turned off.

Table 33. Load sensor setting parameter commands

Device ID	Function code	Register Address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x15	.....	.....	.....	.....

Note: The device ID used here is 0x01, where 0x00 is the user-set parameter and 0xFF is the factory-set parameter.

Return data:

Table 34. Return data for the load sensor setting parameter command

Byte	Name	Data	Description

number			
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x06	Function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x15	Low byte of register address
4	Register data MSB	.....	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	.....	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	.....	ModbusCRC16 check high byte
7	CRC LSB	.....	ModbusCRC16 check low byte

Command #14 Set the sensor output filter sensitivity

This command can set the output filter sensitivity inside the sensor. Increasing this value can improve the stability of the output data but sacrifice some sensitivity; reducing this value will increase the sensitivity of the output data to distance changes, but will reduce the stability of some output data. The factory default value of the filter sensitivity is "0x0000", and the recommended setting value range is within ±100.

Table 35. Set Sensor Output Filter Sensitivity Commands

Device ID	Function code	Register Address		Register Value		CRC high byte	CRC low byte
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x16	.....	.....	.....	.....

Note: The device ID used here is 0x01.

Return data:

Table 36. Set Sensor Output Filter Sensitivity Command Return Data

Byte number	Name	Data	Description
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x06	Function code
2	Address MSB	0x00	High byte of register address
3	Address LSB	0x16	Low byte of register address
4	Register data MSB	.....	If the setting is successful, the value you entered will be returned, the high byte of the register value
5	Register data LSB	.....	If the setting is successful, the value you entered will be returned, the low byte of the register value
6	CRC MSB	.....	ModbusCRC16 check high byte
7	CRC LSB	.....	ModbusCRC16 check low byte

Command #15 Set sensor reconnection time

This command automatically enters reconnection mode and sends out abnormal data packets if no sensor data is received within the specified time (in milliseconds, an integer multiple of 18) in continuous output mode.

Table 37. Set sensor reconnection time command

Device ID	Function code	Initial address		Number of registers		Number of bytes
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6
0x01	0x10	0x00	0x20	0x00	0x02	0x04

Register Value				**CRC high byte**	**CRC low byte**
Byte7	Byte8	Byte9	Byte10	Byte11	Byte12
.....	.....	.....	.....	.....	.....

Note: The device ID used here is 0x01.

Return data:

Table 38. Response data of the command to set sensor reconnection time

Byte number	**Name**	**Data**	**Description**
0	Device ID	0x01	Unique sensor ID
1	Function Code	0x10	Function code
2	Address MSB	0x00	High byte of the starting address
3	Address LSB	0x20	Low byte of the starting address
4	Register number MSB	0x00	High byte of register number
5	Register number LSB	0x02	Low byte of register number
6	CRC MSB	0x40	ModbusCRC16 check high byte
7	CRC LSB	0x02	ModbusCRC16 check low byte

Command# Abnormal response code

When the host device sends a request to the slave device, the host expects a normal response. One of four possible events occurs from the master query:

- If the slave device receives the request without communication errors and can process the query normally, the server device returns a normal response.
- If the slave does not receive the request due to a communication error, then no response can be returned. The client program will eventually handle the request with a timeout condition.
- If the slave receives the request but detects a communication error (parity, LRC, CRC, ...), no response can be returned. The client program will eventually handle the request with a timeout condition.
- If a slave receives a request without a communication error, but cannot process the request (for example, if the request is to read an output or register that does not exist), the server shall return an exception response informing the user of the nature of the error.

Table 39. Exception codes

Modbus exception code		
Code	**Name**	**Description**
01	**Illegal function code**	The function code received in the query is not an allowable operation for the slave. This may be because the function code is only applicable to new devices and is not implementable in the selected unit. It also indicates that the server (or slave) processed this request in an incorrect state, for example because it is unconfigured and requires register values to be returned.
02	**Illegal start address /register address**	The start address received in the query is not an admissible address for the slave. In particular, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 will succeed, a request with offset 96 and length 5 will generate exception code 02
03	**Illegal register count**	The value included in the query is not permissible for the slave. This value indicates a fault in the remaining structure of the composite request, for example the implied length is incorrect. It does not mean that the data item submitted for storage in the register has a value not expected by the application program, because the MODBUS protocol is not aware of the significance of any particular value of any particular register.
04	**Device operation failed**	An unrecoverable error occurred while the slave was trying to perform a requested operation.

Note: Each function will be given a corresponding function code, starting address and register quantity. For details, see the command list;

The exception code 01/02/03 is returned, indicating that the corresponding address or value does not exist in the slave device.

The returned exception code 04 indicates that an unknown exception or error occurred when the slave was performing the operation.

Table 40. Function abnormal code return package

Device ID	Exception code	CRC high byte	CRC low byte
Byte0	Byte2	Byte3	Byte4
0x01	0x01/02/03/04	.....	.....

If the host sends a single measurement command to the slave, such as: 01 03 00 08 00 04 C5 C8

If the start address does not exist in the slave device, the slave will return an exception response with an exception code (02), which indicates an illegal data address of the slave.

The return package is 01 83 02 C0 F1

Packaging Information

Table 41. Packaging specifications

Model	HPS-167S
Size	53（L）x 32（W）x 26.5（H）
Weight	27g/piece (without cable)
Tray	20 pieces (5*4) per plate
Outer case	4 trays/box (80 pieces)

Revision History

Table 42. Specification Revision History

Date	Revision	Description
2020/06/10	1.0	Initial version.
2020/06/11	1.1	Added C language implementation example of CRC16-CCITT
2020/06/19	1.2	Errata, the sensor with RS485 interface has been changed to a waterproof housing version
2020/09/27	2.1	RS485 communication protocol changed to ModBus protocol
2022/08/03	2.2	Modified the range information

Appendix

C language implementation of MODBU-CRC16/*

Byte Lookup

*/

```
/* Table of CRC values for high–order byte */
static unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,
0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00,
0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,
0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81,
0x40
} ;
```

```
/* CRC value of the low byte */
static char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7,
0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB,
0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE,
0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2,
0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32,
0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E,
0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B,
0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27,
0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1,
0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD,
0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8,
0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4,
0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94,
0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59,
0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D,
0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83,
0x41, 0x81, 0x80,
0x40
};
#################################################################
```

Implementation 1:

```
##################################################################
/*
The CRC check result is that the high byte comes first and the low byte comes last, which is consistent
with the result of the official tool modbus pull.*/
unsigned short CRC16( puchMsg , usDataLen)     //The function returns the CRC as unsigned short.
unsigned char *puchMsg;                        //Message used to calculate CRC
unsigned short usDataLen;                          //Number of bytes in the message
{
    unsigned char uchCRCHi = 0xFF;                 // CRC high byte initialization
    unsigned char uchCRCLo = 0xFF;             //CRC low byte initialization
    unsigned uIndex;                           //CRC Lookup Table Index
    while (usDataLen--)                        //Complete the entire message buffer     {
        uIndex = uchCRCLo ^ *puchMsg++;        //Calculating CRC
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex];
        uchCRCHi = auchCRCLo[uIndex];
    }
    return (uchCRCHi << 8 | uchCRCLo);
}

##################################################################
```

Implementation 2:

```
##################################################################
/*
The CRC check result is the inversion of high and low bytes
And in the message, the low byte must be placed in front and the high byte in the back.
*/
unsigned short crc16(unsigned char* puchMsg, unsigned char usDataLen)
{
    unsigned char uchCRCHi = 0xFF;      //CRC high byte initialization
    unsigned char uchCRCLo = 0xFF;    //CRC low byte initialization
    unsigned short   uIndex;                //CRC Lookup Table Index
    while (usDataLen--)
    {
        uIndex = uchCRCHi ^ *puchMsg++;        //Calculating CRC
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];
        uchCRCLo = auchCRCLo[uIndex];
    }
    return (((unsigned short)(uchCRCHi) << 8) | uchCRCLo);
}

##################################################################
```

Test code:

```
#################################################################
void main()
{
    unsigned char sample_data[] = { 0x01, 0x01, 0x01, 0x06, 0xd9, 0xfc, 0x8c, 0x02, 0x01, 0x00, 0x01 };
    unsigned char data1[] = { 0x63 };
    unsigned char data2[] = { 0x8c };
    unsigned char data3[] = { 0x7d };
    unsigned char data4[] = { 0xaa, 0xbb,0xcc };
    unsigned char data5[] = { 0x00,0x00,0xaa, 0xbb, 0xcc };
    unsigned char data6[] = { 0x01,0x03,0x00, 0x03, 0x00,0x0B};
    unsigned short r1 = 0, r2 = 0, r3 = 0, r4 = 0, r5 = 0, r6 = 0, r_sample_data;

    //Implementation_1
    r1 = crc16(data1, 1);
    r2 = crc16(data2, 1);
    r3 = crc16(data3, 1);
    r4 = crc16(data4, 3);
    r5 = crc16(data5, 5);
    r6 = crc16(data6, 6);
    r_sample_data = crc16(sample_data, 11);
    printf("Implementation_3: r1= %x, r2=%x, r3=%x, r4=%x, r5=%x,   r6=%x, r_sample_data=%x\n", r1, r2, r3, r4, r5,
r6 , r_sample_data);
    r1 = r2 = r3 = r4 = r5 = r6 = 0;


    //Implementation_2
    r1 = CRC16(data1, 1);
    r2 = CRC16(data2, 1);
    r3 = CRC16(data3, 1);
    r4 = CRC16(data4, 3);
    r5 = CRC16(data5, 5);
    r6 = CRC16(data6, 6);
    r_sample_data = CRC16(sample_data, 11);
    printf("Implementation_3: r1= %x, r2=%x, r3=%x, r4=%x, r5=%x,   r6=%x, r_sample_data=%x\n", r1, r2, r3, r4, r5,
r6, r_sample_data);
    r1 = r2 = r3 = r4 = r5 = r6 = 0;
}
/*
Implementation_1:  r1= FF69,  r2=BEE5,  r3=7F61,  r4=2345,  r5=7685,  r6=F40D,  r_sample_data=02F1
Implementation_2:  r1= 69FF,  r2=E5BE,  r3=617F,  r4=4523,  r5=8576,  r6=0DF4,  r_sample_data=F102
*/
#################################################################
```

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Hypersen Technologies Co., Ltd. reserve the right to make changes, corrections, enhancements, modifications, and improvements to Hypersen products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on Hypersen products before placing orders. Hypersen products are sold pursuant to Hypersen's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of Hypersen products and Hypersen assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by Hypersen herein.

Resale of Hypersen products with provisions different from the information set forth herein shall void any warranty granted by Hypersen for such product.

Hypersen and the Hypersen logo are trademarks of Hypersen. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.