HPS-166S-L

超高性能防水飞行时间(ToF)传感器

产品特性

- 快速、精确测距
  - 最远测量距离可达 25m (白色目标)
  - 输出数据速率高达 54Hz
  - 测量结果对目标的颜色和反射率不敏感
  - 电气和光学串扰补偿
  - -10℃~+55℃ 温度补偿
  - 内置环境光补偿功能使得传感器能在高红外背景光环境下工作
- 完全集成的微型模块，包含:
  - ABS 防水外壳+航空铝散热背板
  - 850nm 红外 VCSEL 发射器
  - 发射器驱动电路
  - 优化设计的发射和接收光学镜头
  - 高性能嵌入式微处理器
  - 先进的嵌入式数据处理与滤波算法
  - 19200 bps RS485 总线接口
  - 53(长) x 32(宽) x 26.5(高) mm，27g
  - 符合最新的 CE, FCC 及 RoHS 标准

产品应用

- 无人机 (避障、定高悬停、软着陆)
- 机器人 & AGV 自动导航机器人 (障碍物检测)
- 工业定位和接近传感器
- 安全和监控
- 1D 动作识别

产品描述

HPS-166S-L 是新一代超高性能防水飞行时间(ToF)红外测距传感器，配有优化设计的发射和接收光学镜头，适用于高精度、长距离的测距场合。区别于传统的技术，它的测量精度不受测量目标颜色和反射率的影响。目标为 90%反射率白色目标时，HPS-166S-L 的测量距离可以达到 25 米，在红外测距领域设定了一个新的里程碑，拓展了更加广阔的应用场景。

HPS-166S-L 采用 ABS 防水外壳+硬质阳极氧化航空铝散热背板，内部集成 1 个大功率 850nm 红外 VCSEL 发射器和 1 个高灵敏度光电二极管接收器，加上内部集成的物理红外滤光片，使其测距离更远，抗环境光干扰能力更强。

内部集成先进的嵌入式数据处理和滤波算法，实现了非常稳定和实时的测量结果输出。

订购信息

HPS - 1 6 x S - L

- 海伯森产品系列标识
- **16x**: ToF 测距/避障传感器系列
- **无**: 3.3V UART 接口，无防水外壳
- **U**: 3.3V UART 接口，有防水外壳
- **C**: 1Mbps CAN 总线接口，有防水外壳
- **S**: RS485 总线接口，有防水外壳
- **无**: 850nm LED(发光二极管)发射管
- **L**: 850nm VCSEL(垂直腔面发射激光)发射管

CE FC RoHS

AVOID EYE OR SKIN EXPOSURE TO DIRECT OR SCATTERED RADIATION

1.概述

1.1 技术规格

表 1. 技术规格

参数	值	单位
尺寸	53（长）x 32（宽）x 26.5（高）	mm
重量	27 *1	g
供电	5 ~ 15	V
最大功耗	1.4	W
静态功耗	0.1	W
存储温度	-40 ~ 85	℃
工作温度	-30 ~ 55 *2	℃
红外 VCSEL 发射波长	850	nm
发射角度	±1.8	°
最大测量距离	25 *3	m
最小测量距离	0.08	m
输出数据频率	6~54	Hz
输出数据	距离，精度指示，信号强度，环境红外光强度，温度	-
电气输出	四线输出，线长默认 70cm，裸线烫锡或者冷压端子	-
通讯接口	RS485, 19200bps	

注： *1 不含电缆。

*2 在连续工作模式下进行远距离测量时（距离大于 20 米），HPS-166S-L 需要一定的预热时间以稳定输出。

*3 测试基于 90%反射率的白色目标。

1.2 外形尺寸及针脚定义

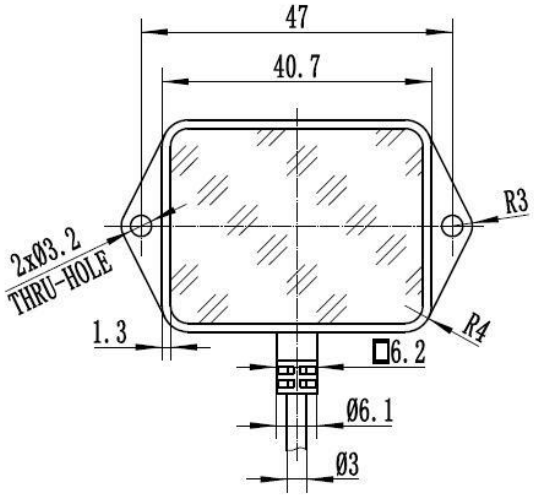


图 1. HPS-166S-L 前视图

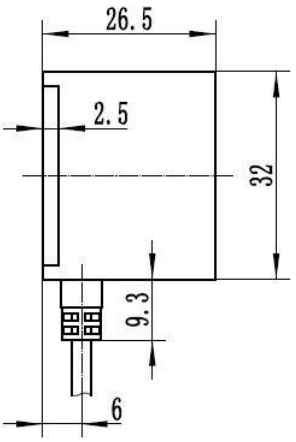


图 2. HPS-166S-L 侧视图

表 2. HPS-166S-L 线序定义

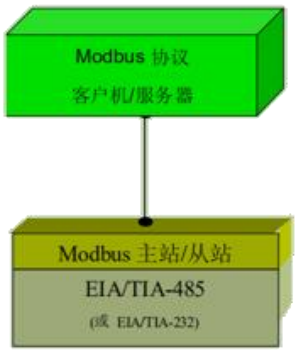
线色	信号名称	信号种类	描述
黑色	GND	GND	电源地
白色	A	Digital	RS485 总线 A 端
绿色	B	Digital	RS485 总线 B 端
红色	VDD	Power	电源正极，连接到 DC +5V~+15V

2.MODBUS_RTU 通讯协议

2.1 简介

MODBUS 是 OSI 模型第 7 层上的应用层报文传输协议，它在连接至不同类型总线或网络的设备之间提供客户机/服务器通信。它还将串行链路上的协议标准化，以便在一个主站和一个或多个从站之间交换 Modbus 请求。

下图给出了 Modbus 串行通信栈对应于 7 层 OSI 模型的一般关系。



层	ISO/OSI 模型	
7	应用层	Modbus协议
6	表示层	空
5	会话层	空
4	传输层	空
3	网络层	空
2	数据链路层	Modbus 串行链路协议
1	物理层	EIA/TIA-485 (或 EIA/TIA-232)

图 3. Modbus 协议和 ISO/OSI 模型

2.2 Modbus 数据链路层

2.2.1 Modbus 主站/从站协议原理

Modbus 是一种主从通讯模式的通讯协议，也就是说，Modbus 有一个主机，可以连接多个从机（最多 32 个），主机连接于总线，一个或多个从机连接于同一个串行总线上。

Modbus 中主机可以主动进行通讯，其他从机只能对主机进行响应而不能主动发送数据到通讯总线中，而且从机之间不能互相通信。这种方法规定了通讯过程中的通讯次序等关系，避免了多个设备同时工作的情况下通讯冲突的产生。

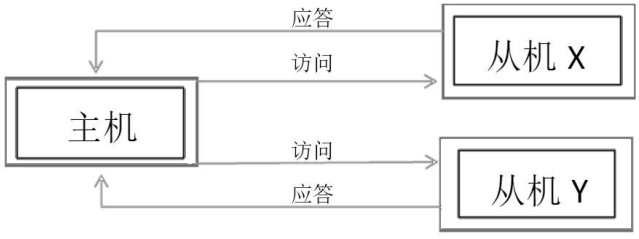


图 4. 主机/从机通讯

Modbus 中，主站可以通过两种模式向从机发送请求：

- 单播模式：主站通过特定的地址（设备地址或者传感器 ID）访问某个从站，从站接收到并处理完请求后，从站向从站返回一个报文(一个'应答')。
  每个从站必须有唯一的地址或者 ID (1 到 32)，这样才能区别于其它从站被独立的寻址。

- 广播模式：主站向所有的从站发送请求。
  对于主机用广播模式下发送的请求，从机没有应答返回。所以广播请求一般用于写命令。所有设备必须接受广播模式的写功能。地址 0x00 表示广播地址。

2.2.2 modbus 串行传输模式

Modbus 有两种传输模式 RTU 模式和 ASCII 模式。它确定了信息如何打包为报文和解码。
本产品使用的是 RTU 模式。

RTU 模式：
这种模式的主要优点是较高的数据密度，在相同的波特率下比 ASCII 模式有更高的吞吐率。
每个报文必须以连续的字符流传送。

RTU 模式下每个字节（11 位）格式：

表 3. RTU 模式位序列

起始位	数据位								奇偶校验位	停止位
1 位	1	2	3	4	5	6	7	8	1 位	1 位

注：数据位：首先发送最低有效位（从左到右）。
奇偶校验位：偶校验是要求的，也是默认模式。

2.2.3 Modbus RTU 报文帧描述

表 4. RTU 模式的报文帧

地址域	功能码	数据	CRC
1 字节	1 字节	0~252 字节	2 字节

地址域：一般指的是设备码，在这里是指传感器 ID 或者设备地址。主机没有地址域，而各个从机都有独立的地址域，通过地址域，主机可以和多个从机进行数据通讯而避免了其他设备错误响应不属于该设备的通讯。

由发送设备将 Modbus 报文构造为带有已知起始和结束标记的帧。这使设备可以在报文的开始接收新帧，并且知道何时报文结束。不完整的报文必须能够被检测到而错误标志必须作为结果被设置。
在 RTU 模式，报文帧由时长至少为 3.5 个字符时间的空闲间隔区分。

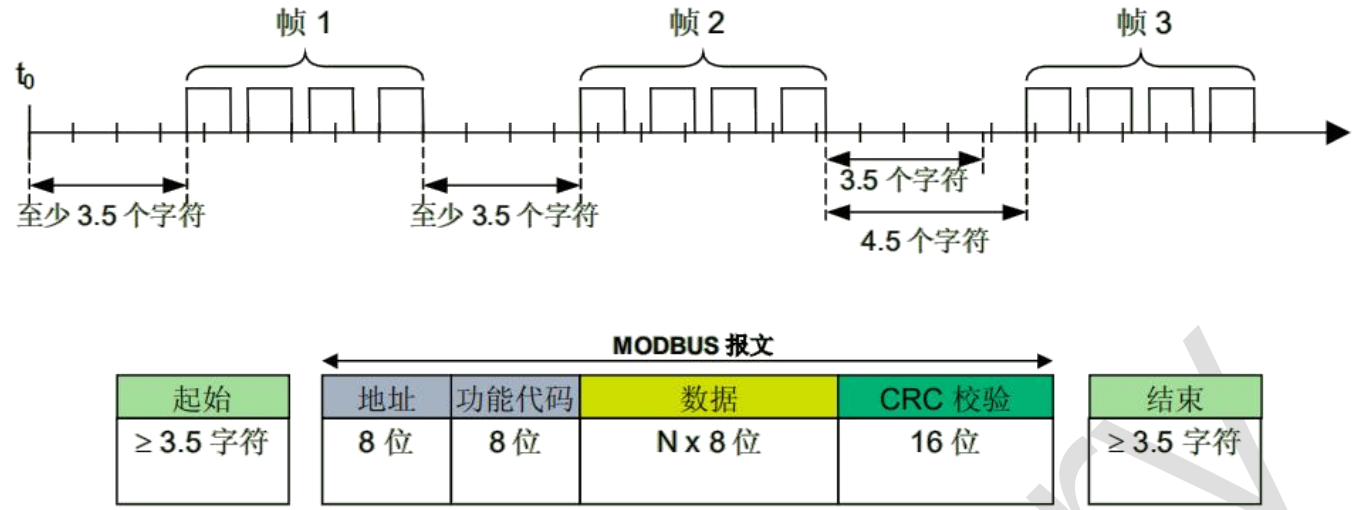


图 5. RTU 报文帧

整个报文帧必须以连续的字符流发送。
如果两个字符之间的空闲间隔大于 1.5 个字符时间，则报文帧被认为不完整应该被接收节点丢弃。

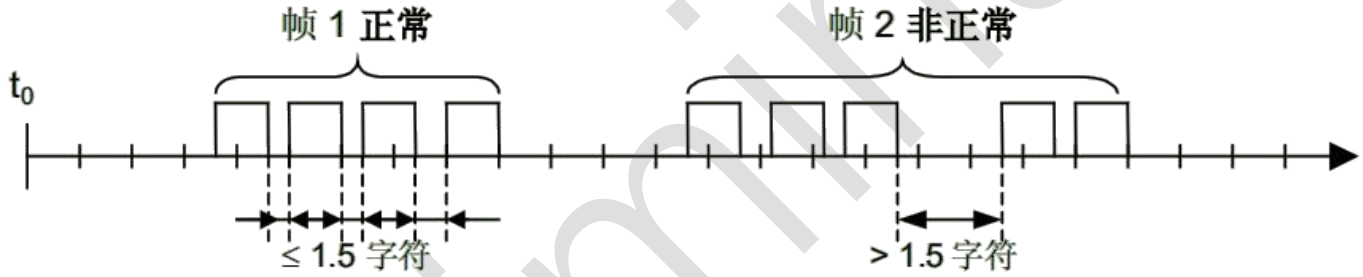


图 6. RTU 报文帧 2

注:

RTU 接收驱动程序的实现，由于 t1.5 和 t3.5 的定时，隐含着大量的对中断的管理。在高通信速率下，这导致 CPU 负担加重。因此，在通信速率等于或低于 19200 Bps 时，这两个定时必须严格遵守；对于波特率大于 19200 Bps 的情形，应该使用 2 个定时的固定值：建议的字符间超时时间(t1.5)750μs，帧间的超时时间(t1.5)为 1.750ms。

2.2.4 功能码描述

功能码的含义是表明该报文帧的功能或目的。
功能码分类主要是有三种：公共功能码、用户定义功能码、保留功能码。
其中本产品只用公共功能码。

表 5. 公共功能码

功能码	名称	描述
0x03	读保存寄存器	该功能码读取保持寄存器连续块的内容
0x06	写单个寄存器	该功能码写单个保持寄存器
0x10	写多个寄存器	该功能码写连续寄存器块(1 至约 120 个寄存器)

2.2.4.1 （0x03）读保存寄存器

主机发送请求：

地址域	功能码	起始地址	寄存器数量	CRC 校验
1 字节	1 字节	2 字节	2 字节	2 字节
1 至 32	0x03	0x0000 至 0xFFFF	1 至 125（0x7D）	

从机响应：

地址域	功能码	字节数	寄存器值	CRC 校验
1 字节	1 字节	1 字节	N X 2 个字节	2 字节
1 至 32	0x03	2 X N		

N = 寄存器的数量

从机错误响应：

地址域	差错码	异常码	CRC 校验
1 字节	1 字节	1 字节	2 字节
1 至 32	0x83	01 或 02 或 03 或 04	

详细请看：命令# 异常响应码

例如：功能码 03 表明主机要求读取从机若干个寄存器的数值，从机接收到通讯帧后进行一系列的处理，返回给主机相同的功能码，表明从机对该功能进行了响应。

此外如果数据帧存在错误，例如：格式不正确、漏发、校验不正确等，从机在返回功能码时，会在原有的功能码的基础上加 128（0x80 | 0x03），来表示是哪一段功能码出错，主机在接收到后就能根据此时返回的功能码来进行错误判断。因此功能码既有表达该帧功能的作用，又有表示该次通讯是否错误的功能。

2.2.4.2 （0x06）写单个寄存器

主机发送请求：

地址域	功能码	寄存器地址	寄存器值	CRC 校验
1 字节	1 字节	2 字节	2 字节	2 字节
1 至 32	0x06	0x0000 至 0xFFFF	0x0000 至 0xFFFF	

从机响应：

地址域	功能码	寄存器地址	寄存器值	CRC 校验
1 字节	1 字节	2 字节	2 字节	2 字节
1 至 32	0x06	0x0000 至 0xFFFF	0x0000 至 0xFFFF	

N = 寄存器的数量

从机错误响应：

地址域	差错码	异常码	CRC 校验
1 字节	1 字节	1 字节	2 字节
1 至 32	0x86	01 或 02 或 03 或 04	

详细请看：命令# 异常响应码

2.2.4.3 （0x10）写多个寄存器

主机发送请求：

地址域	功能码	起始地址	寄存器数量	字节数	寄存器值	CRC 校验
1 字节	1 字节	2 字节	2 字节	1 字节	N X 2 个字节	2 字节
1 至 32	0x10	0x0000 至 0xFFFF	0x0001 至 0x0078	2 X N	值	

N = 寄存器的数量

从机响应：

地址域	功能码	起始地址	寄存器数量	CRC 校验
1 字节	1 字节	2 字节	2 字节	2 字节
1 至 32	0x10	0x0000 至 0xFFFF	1 至 123（0x7B）	

N = 寄存器的数量

从机错误响应：

地址域	差错码	异常码	CRC 校验
1 字节	1 字节	1 字节	2 字节
1 至 32	0x90	01 或 02 或 03 或 04	

详细请看：命令# 异常响应码

2.2.5 CRC 校验

在 RTU 模式包含一个对全部报文内容执行的，基于循环冗余校验（CRC - Cyclical Redundancy Checking）算法的错误检验域。CRC 域检验整个报文的内容。不管报文有无奇偶校验，均执行此检验。

CRC 包含由两个 8 位字节组成的一个 16 位值。

CRC 域作为报文的最后的域附加在报文之后。计算后，首先附加低字节，然后是高字节。

附加在报文后面的 CRC 的值由发送设备计算。接收设备在接收报文时重新计算 CRC 的值，并将计算结果于实际接收到的 CRC 值相比较。如果两个值不相等，则为错误。

CRC 计算：

开始对一个 16 位寄存器预装全 1。 然后将报文中的连续的 8 位子节对其进行后续的计算。只有字符中的 8 个数据位参与生成 CRC 的运算，起始位，停止位和校验位不参与 CRC 计算。

CRC 的生成过程中，每个 8 位字符与寄存器中的值异或。然后结果向最低有效位(LSB)方向移动(Shift) 1 位，而最高有效位(MSB)位置充零。然后提取并检查 LSB：如果 LSB 为 1，则寄存器中的值与一个固定的预置值异或；如果 LSB 为 0，则不进行异或操作。

这个过程将重复直到执行完 8 次移位。完成最后一次（第 8 次）移位及相关操作后，下一个 8 位字节与寄存器的当前值异或，然后又同上面描述过的一样重复 8 次。当所有报文中子节都运算之后得到的寄存器忠的最终值，就是 CRC。

当 CRC 附加在报文之后时，首先附加低字节，然后是高字节。在附录含有 CRC 生成的详细示例。

生成 CRC 的过程为：

1、将一个 16 位寄存器装入十六进制 FFFF (全 1)，将之称作 CRC 寄存器。

2、将报文的第一个 8 位字节与 16 位 CRC 寄存器的低字节异或，结果置于 CRC 寄存器。

3、将 CRC 寄存器右移 1 位(向 LSB 方向)，MSB 充零，提取并检测 LSB。

4、(如果 LSB 为 0)：重复步骤 3 (另一次移位)。

(如果 LSB 为 1)：对 CRC 寄存器异或多项式值 0xA001 (1010 0000 0000 0001)。

5、重复步骤 3 和 4，直到完成 8 次移位。当做完此操作后，将完成对 8 位字节的完整操作。

6、对报文中的下一个字节重复步骤 2 到 5，继续此操作直至所有报文被处理完毕。

7、CRC 寄存器中的最终内容为 CRC 值。

8、当放置 CRC 值于报文时，如下面描述的那样，高低字节必须交换。

将 CRC 放置于报文

当 16 位 CRC (2 个 8 位字节)在报文中传送时，低位字节首先发送，然后是高位字节。

例如，如果 CRC 值为十六进制 1241 (0001 0010 0100 0001):

图 7. CRC 字节序列

2.3 通讯参数配置

设备接口方式为 RS485。采用 MODBUS 协议-RTU 模式。（8 数据位，首先发送最低有效位）

表 6. 主/从机参数配置

波特率	19200bps
起始位	1bit
数据位	8bits
停止位	1bit
奇偶校验位	偶校验
CRC 校验	ModbusCRC16

波特率还可以有 57600bps、115200bps、230400bps。

2.4 通讯模型

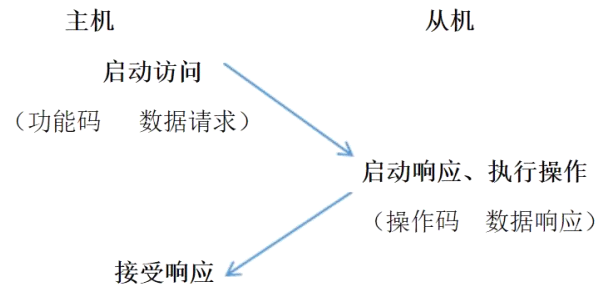


图 8．通讯模型

2.5 数据帧格式

表 7. MODBUS 数据帧格式

设备 ID	功能码	数据	CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4
1 字节	1 字节	0 ~ 256 字节	1 字节	1 字节

主机在向传感器发送命令时，必须设置正确的传感器 ID 号，否则传感器将不会响应；传感器在接收到正确的命令后会执行相应的操作，并返回操作的结果。通讯正常的情况下，主机发送的每个命令都会有返回数据。

3. 命令列表

表 8. Modbus_RS485 命令列表

命令	设备地址	功能码	起始地址	描述
读取 RS485 的版本信息	设备 ID (1~32)	0x03	0x01	获取 RS485 接口固件的软件版本信息
获取 RS485 配置信息			0x02	获取 RS485 的接口固件的配置信息
读取传感器的详细信息			0x03	将读取传感器的详细信息
获取传感器（AFE）的温度			0x04	获取传感器模拟前端的当前温度
单次测量			0x08	将启动传感器进行一次单次测量
恢复 RS485 接口固件出厂设置			0x09	将 RS485 接口的所有设定恢复到出厂设定值
传感器的预热时间		0x06	0x0A	将设置传感器的预热时间
设置传感器 ID 号			0x10	可修改传感器在 RS485 总线上的 ID 号
设置 RS485 通讯波特率			0x11	用来设置 RS485 的通讯波特率
设定传感器通电后的测量模式			0x12	设定通电后是否自动进入连续测量模式
设定 485 总线终端电阻			0x13	设定通电后是否自动启动总线终端电阻
设置测量距离偏差补偿值			0x14	可用来补偿传感器的固有测量偏差
加载传感器的设置参数			0x15	将加载传感器内部保存的设置参数
设置传感器输出滤波器灵敏度			0x16	设置传感器内部的输出滤波器灵敏度
设置传感器重连时间		0x10	0x20	设置传感器重连时间

命令#1 获取 RS485 接口固件的版本信息

该命令可获取 RS485 接口固件的软件版本信息。

表 9. 获取 RS485 接口固件的版本信息命令

设备 ID	功能码	起始地址		寄存器数量		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x01	0x00	0x03	0x54	0x0B

注：这里统一用设备 ID：0x01。

返回数据：

表 10. 获取 RS485 接口固件的版本信息命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x03	功能码
2	Date Length	0x06	字节数 = 寄存器数量*2
3	Year	.....	转接板的生产日期
4	Month	.....	
5	Day	.....	
6	Major version	.....	主版本号
7	Minor version	.....	从版本号
8	Rev	.....	修订号
9	CRC MSB	.....	ModbusCRC16 校验高字节
10	CRC LSB	.....	ModbusCRC16 校验低字节

命令#2 获取 RS485 的接口固件的配置信息

此命令就是获取 RS485 的接口固件的配置信息。

表 11. 获取 RS485 的接口固件的配置信息的命令

设备 ID	功能码	起始地址		寄存器数量		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x02	0x00	0x04	0xE5	0xC9

注：这里统一用设备 ID：0x01。

返回数据：

表 12. 获取 RS485 的接口固件的配置信息的命令返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x03	功能码
2	Date Length	0x08	字节数 = 寄存器数量*2
3	Resistance	.....	终端电阻选择，0x00 表示不启用；0xFF 表示启用
4	Auto_out	.....	初始化后是否进入连续输出模式，0x00 表示不启用；0xFF 表示启用
5	Preheat time	.....	预热时间，单位：秒
6	Connect_time	.....	连续输出下，RS485 恢复通讯所需要的的时间,单位：毫秒
7		.....	
8		.....	
9		.....	
10	Baud rate	.....	RS485 波特率，默认：19200
11	CRC MSB	.....	ModbusCRC16 校验高字节
12	CRC LSB	.....	ModbusCRC16 校验低字节

命令#3 读取传感器的详细信息

此命令将读取传感器的详细信息。

表 13. 读取传感器的详细信息的命令

设备 ID	功能码	起始地址		寄存器数量		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x03	0x00	0x0B	0xF4	0x0D

注：这里统一用设备 ID：0x01。

返回数据：

表 14. 读取传感器的详细信息的命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x03	功能码
2	Date Length	0x16	字节数 = 寄存器数量*2
3	ACK Byte	0xB0	传感器返回的应答字节
4~19	UUID	.....	传感器的通用唯一识别码
20	Year	.....	年
21	Month	.....	月
22	Day	.....	日
23	Major version	.....	传感器的版本号
24	Minor version	.....	
25	CRC MSB	.....	ModbusCRC16 校验高字节
26	CRC LSB	.....	ModbusCRC16 校验低字节

以下是返回的传感器详细信息数据的解析例：

返回数据包：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

解析：

0x01: 设备的 ID 号

0x03: 功能码

0x16: 字节数 = 寄存器数量*2

0xB0: 应答字节

0x00 0x39 0x00 0x25 0x42 0x34 0x57 0x14 0x20 0x34 0x38 0x35 0x01 0x06 0x07 0x05: 通用唯一识别码

0x13 0x09 0x17: 19/09/27

0x03 0x09: Ver. 2.1

0x1A 0xDC: ModBus CRC16 校验

命令#4 获取传感器模拟前端（AFE）的温度

此命令可以获取传感器模拟前端的当前温度（单位：华氏度）。

表 15. 获取传感器模拟前端（AFE）的温度命令

设备 ID	功能码	起始地址		寄存器数量		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x04	0x00	0x01	0xC5	0xCB

注：这里统一用设备 ID：0x01。

返回数据：

表 16. 获取传感器模拟前端（AFE）的温度命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x03	功能码
2	Date Length	0x02	字节数 = 寄存器数量*2
3	AFE_temp MSB	.....	获取 AFE 的温度值高字节
4	AFE_temp LSB	.....	获取 AFE 的温度值低字节
5	CRC MSB	.....	ModbusCRC16 校验高字节
6	CRC LSB	.....	ModbusCRC16 校验低字节

命令#5 单次测量

此命令将启动传感器进行一次单次测量。

表 17. 单次测量的命令

设备 ID	功能码	起始地址		寄存器数量		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x08	0x00	0x04	0xC5	0xCB

注：这里统一用设备 ID：0x01。

连续测量：就是多次发送单次测量命令，并接收。在连续测量模式下的数据输出速率最高为 54Hz，如果连续测量期间传感器发生故障，则会返回异常数据包（Distance、Ambient、Precision 的值全为"0xFF"，Magnitude 的值为"0x00"）。

返回数据：

表 18. 单次测量的命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x03	功能码
2	Date Length	0x08	字节数 = 寄存器数量*2
3	Distance MSB	.....	测量距离，单位：毫米
4	Distance LSB	.....	
5	Magnitude MSB	.....	接收反射光信号强度
6	Magnitude LSB	.....	
7	Magnitude Exp	.....	接收信号强度指数字节
8	Ambient ADC	.....	相对环境红外光强度
9	Precision MSB	.....	精度指示值，值越小代表测量精度越高
10	Precision LSB	.....	

11	CRC MSB	.....	ModbusCRC16 校验高字节
12	CRC LSB	.....	ModbusCRC16 校验低字节

注: 当传感器超量程或反射信号强度过低时会输出 65.53 米的超量程指示。

以下是测量返回数据的解析例：

返回数据包: 0x01 0x03 0x08 0x08 0x23 0xDC 0xB2 0x07 0x01 0x00 0x00 0xFD 0x41

解析：

0x01: 设备的 ID 号

0x03: 功能码

0x08: 字节数 = 寄存器数量*2

测量距离 Distance = (0x08 * 256 + 0x23) / 1000.0f = 2.083 (单位: 米)

接收信号强度 Magnitude = ((0xDC * 256 + 0xB2) << 0x07) / 10000.0f = 723.1744

环境红外光强度 Ambient ADC = 1

精度指示 Precision = (0x00 * 256) + 0x00 = 0

0xFD 0x41: ModBus CRC16 校验

命令#6 恢复 RS485 接口固件出厂设置

此命令可将传感器 RS485 接口固件中的所有设定恢复到出厂设定值。执行该命令后总线上的传感器 ID 号将被重置为“0x01”。

表 19. 恢复 RS485 接口固件出厂设置命令

设备 ID	功能码	起始地址		寄存器数量		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x03	0x00	0x09	0x00	0x01	0x54	0x08

注：这里统一用设备 ID：0x01。

返回数据：

表 20. 恢复 RS485 接口固件出厂设置命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x03	功能码
2	Date Length	0x02	字节数 = 寄存器数量*2
3	.....	0x00	.....
4	Success Flag	0x01	如果显示 0x01，表示操作成功，错误会返回错误码
5	CRC MSB	0x79	ModbusCRC16 校验高字节
6	CRC LSB	0x84	ModbusCRC16 校验低字节

命令#7 设定传感器的预热时间

传感器出厂默认的启动预热时间为 5 秒，预热时间内传感器将不会输出测量数据。此命令可修改传感器通电后的预热时间，执行该命令后的设定值将被自动写入传感器内部的 FLASH 中，断电不会丢失。

表 21. 设定传感器的启动预热时间的命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x0A	0x00	0x05	0x69	0xCB

注：这里统一用设备 ID：0x01。

返回数据：

表 22. 设定传感器的启动预热时间的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的设备识别 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x0A	寄存器地址的低字节
4	Register data MSB	0x00	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	0x05	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	0x69	ModbusCRC16 校验高字节
7	CRC LSB	0xCB	ModbusCRC16 校验低字节

命令#8 设置传感器 ID 号

该命令可修改传感器在 RS485 总线上的 ID 号（出厂默认值为"0x01"）。此命令设置后将立即生效，设定的 ID 号会被自动写入传感器内部的 FLASH 中，断电不会丢失。

表 23. 设置传感器 ID 命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x10	.....	.....	.....	.....

注：这里统一用设备 ID：0x01。

返回数据：

表 24. 设置传感器 ID 命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x10	寄存器地址的低字节
4	Register data MSB	.....	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	.....	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	.....	ModbusCRC16 校验高字节
7	CRC LSB	.....	ModbusCRC16 校验低字节

命令#9 设置 RS485 通讯波特率

此命令可用来设置 RS485 的通讯波特率。

表 25. 设置通讯波特率的命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x11	0x00	0x01	0x18	0x0F

注：这里统一用设备 ID：0x01。默认波特率为 19200bps，其中 01 代表 19200； 02 代表 57600； 03 代表 115200； 04 代表 230400；其他的都为 19200bps。

返回数据：

表 26. 设置通讯波特率的命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x11	寄存器地址的低字节
4	Register data MSB	0x00	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	0x01	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	0x18	ModbusCRC16 校验高字节
7	CRC LSB	0x0F	ModbusCRC16 校验低字节

命令#10 设定传感器通电后的测量模式

此命令可设定传感器通电后是否自动进入连续测量模式，出厂默认为启用状态，此命令设置后将在下次通电时生效，设定值会被自动写入传感器内部的 FLASH 中，断电不会丢失。

表 27. 设定传感器通电后的测量模式命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x12	0x00	0x00	0x29	0xCF

注：这里统一用设备 ID：0x01。其中 0x00：是不启用； 0xFF:是启用。

返回数据：

表 28. 设置通讯波特率的命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x12	寄存器地址的低字节
4	Register data MSB	0x00	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	0x00	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	0x29	ModbusCRC16 校验高字节
7	CRC LSB	0xCF	ModbusCRC16 校验低字节

命令#11 设定 RS485 总线终端电阻

此命令可设定传感器通电后是否自动启动 RS485 总线终端电阻，出厂默认为启用状态，此命令设置后将在下次通电时生效，设定值会被自动写入传感器内部的 FLASH 中，断电不会丢失。

表 29. 设定 RS485 总线终端电阻命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x13	0x00	0xFF	0x38	0x4F

注：这里统一用设备 ID：0x01。其中 0x00：是不启用； 0xFF:是启用。

返回数据：

表 30. 设定 RS485 总线终端电阻命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x13	寄存器地址的低字节
4	Register data MSB	0x00	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	0xFF	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	0x38	ModbusCRC16 校验高字节
7	CRC LSB	0x4F	ModbusCRC16 校验低字节

命令#12 设置测量距离偏差补偿值

此命令可用来补偿传感器的固有测量偏差。

表 31. 设置测量距离偏差补偿值命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x14	.....	.....	.....	.....

注：这里统一用设备 ID：0x01。

返回数据：

表 32. 设置测量距离偏差补偿值命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x14	寄存器地址的低字节
4	Register data MSB	.....	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	.....	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	.....	ModbusCRC16 校验高字节
7	CRC LSB	.....	ModbusCRC16 校验低字节

例子：

真实距离：200 毫米，传感器测量距离：215 毫米

距离偏差值 = 200 – 215 = -15 = 0xFFF1 (偏差值高字节 = 0xFF，偏差值低字节 = 0xF1)

注：由于传感器个体的性能差异，该命令可用来补偿小范围的测量偏差以实现更高的测量精度。调用该命令设定的偏差值会自动被保存到传感器的 Flash 存储器中并在每次传感器上电时自动加载。

命令#13 加载传感器的设置参数

执行此命令将加载传感器内部保存的设置参数，设置参数加载后断电不会丢失。

表 33. 加载传感器设置参数命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x15	.....	.....	.....	.....

注：这里统一用设备 ID：0x01。其中 0x00：是用户设置参数； 0xFF:是出厂设置参数。

返回数据：

表 34. 加载传感器设置参数命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x15	寄存器地址的低字节
4	Register data MSB	.....	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	.....	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	.....	ModbusCRC16 校验高字节
7	CRC LSB	.....	ModbusCRC16 校验低字节

命令#14 设置传感器输出滤波器灵敏度

此命令可设置传感器内部的输出滤波器灵敏度，增大该值能提高输出数据的稳定性但会牺牲一些灵敏度；减小该值则会提高输出数据对于距离变化的灵敏度，但会降低一些输出数据的稳定性。滤波器灵敏度的出厂默认值为“0x0000”，建议的设定值范围为±100 以内。

表 35. 设置传感器输出滤波器灵敏度命令

设备 ID	功能码	寄存器地址		寄存器值		CRC 高字节	CRC 低字节
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x01	0x06	0x00	0x16	.....	.....	.....	.....

注：这里统一用设备 ID：0x01。

返回数据：

表 36. 设置传感器输出滤波器灵敏度命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x06	功能码
2	Address MSB	0x00	寄存器地址的高字节
3	Address LSB	0x16	寄存器地址的低字节
4	Register data MSB	.....	设置成功会返回你输入的值，寄存器值的高字节
5	Register data LSB	.....	设置成功会返回你输入的值，寄存器值的低字节
6	CRC MSB	.....	ModbusCRC16 校验高字节
7	CRC LSB	.....	ModbusCRC16 校验低字节

命令#15 设置传感器重连时间

此命令在连续输出模式下，指定时间内（单位毫秒，为 18 的整数倍）没有收到传感器数据，自动进入重连模式，并发出异常数据包。

表 37. 设置传感器重连时间命令

设备 ID	功能码	起始地址		寄存器数量		字节数
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6
0x01	0x10	0x00	0x20	0x00	0x02	0x04

寄存器值				CRC 高字节	CRC 低字节
Byte7	Byte8	Byte9	Byte10	Byte11	Byte12
.....	.....	.....	.....	.....	.....

注：这里统一用设备 ID：0x01。

返回数据：

表 38. 设置传感器重连时间命令的返回数据

字节号	名称	数据	描述
0	Device ID	0x01	唯一的传感器 ID
1	Function Code	0x10	功能码
2	Address MSB	0x00	起始地址的高字节
3	Address LSB	0x20	起始地址的低字节
4	Register number MSB	0x00	寄存器数量的高字节
5	Register number LSB	0x02	寄存器数量的低字节
6	CRC MSB	0x40	ModbusCRC16 校验高字节
7	CRC LSB	0x02	ModbusCRC16 校验低字节

命令# 异常响应码

当主机设备向从站设备发送请求时，主机希望一个正常响应。

从主站询问中出现下列四种可能事件之一:

- 如果从站设备接收到无通信错误的请求，并且可以正常地处理询问，那么服务器设备将返回一个正常响应。
- 如果由于通信错误，从站没有接收到请求，那么不能返回响应。客户机程序将最终处理请求的超时状态。
- 如果从站接收到请求，但是检测到一个通信错误（奇偶校验、LRC、CRC、...），那么不能返回响应。客户机程序将最终处理请求的超时状态。
- 如果从站接收到无通信错误的请求，但不能处理这个请求（例如，如果请求读一个不存在的输出或寄存器），服务器将返回一个异常响应，通知用户错误的本质特性。

表 39. 异常码

Modbus 异常码		
代码	**名称**	**含义**
01	**非法功能码**	对于从站来说，询问中接收到的功能码是不可允许的操作。这也许是因为功能码仅仅适用于新设备而在被选单元中是不可实现的。同时，还指出服务器(或从站)在错误状态中处理这种请求，例如：因为它是未配置的，并且要求返回寄存器值。
02	**非法起始地址 /寄存器地址**	对于从站来说，询问中接收到的起始地址是不可允许的地址。特别是，参考号和传输长度的组合是无效的。对于带有 100 个寄存器的控制器来说，带有偏移量 96 和长度 4 的请求会成功，带有偏移量 96 和长度 5 的请求将产生异常码 02
03	**非法寄存器数量**	对于从站来说，询问中包括的值是不可允许的值。这个值指示了组合请求剩余结构中的故障，例如：隐含长度是不正确的。并不意味着，因为 MODBUS 协议不知道任何特殊寄存器的任何特殊值的重要意义，寄存器中被提交存储的数据项有一个应用程序期望之外的值。
04	**设备操作失败**	当从站正在设法执行请求的操作时，产生不可重新获得的差错。

注：每个功能都会给予相对应的功能码，起始地址和寄存器数量，详细情况看命令列表；
返回异常码 01/02/03，说明从站设备中不存在相对应的地址或者值。
返回异常码 04，说明从站在执行操作时发生未知的异常或者错误。

表 40. 功能异常码返回包

设备 ID	异常码	CRC 高字节	CRC 低字节
Byte0	Byte2	Byte3	Byte4
0x01	0x01/02/03/04	.....	.....

如果主机向从机发送一个单次测量的命令，如： 01 03 00 08 00 04 C5 C8

如果在从站设备中不存在起始地址，那么从站将返回异常码(02)的异常响应。这就说明从站的非法数据地址。

返回包是 01 83 02 C0 F1

包装信息

表 41. 包装规格

型号	HPS-166S-L
传感器尺寸	53（长）x 32（宽）x 26.5（高）
重量	27 克 / 个 （不含电缆）
托盘	20 个（5*4）每盘
外箱	4 盘 / 箱（80 个）

修订历史记录

表 42. 规格书修订历史记录

Date	Revision	Description
2020/06/10	1.0	初始版本。
2020/06/11	1.1	加入 CRC16-CCITT 的 C 语言实现示例
2020/06/19	1.2	勘误，RS485 接口的传感器改成有防水外壳版本
2020/09/27	2.1	RS485 通讯协议更改为 ModBus 协议
2022/08/03	2.2	修改了量程信息
2023/11/01	2.3	修改了奇偶校验位描述

附录

MODBU-CRC16 的 C 语言实现

```
/*
字节查表法
*/
##################################################################
/* 高位字节的 CRC 值 */
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
```

```
} ;
/* 低位字节的 CRC 值 */
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
};
#################################################################
```

实现 1:

```
#################################################################
/*
CRC校验结果是高字节在前，低字节在后，与官方工具modbus pull结果一致
*/
unsigned short CRC16( puchMsg , usDataLen)    //函数以 unsigned short 类型返回 CRC
unsigned char *puchMsg;                       //用于计算 CRC 的报文
unsigned short usDataLen;                         //报文中的字节数
{
    unsigned char uchCRCHi = 0xFF;                // CRC 的高字节初始化
    unsigned char uchCRCLo = 0xFF;            //CRC 的低字节初始化
    unsigned uIndex;                          // CRC 查询表索引
    while (usDataLen--)                       //完成整个报文缓冲区
    {
        uIndex = uchCRCLo ^ *puchMsg++;       //计算 CRC
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex];
        uchCRCHi = auchCRCLo[uIndex];
    }
    return (uchCRCHi << 8 | uchCRCLo);
}

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

实现 2:

```
#################################################################
/*
CRC校验结果是高低字节反转
而且在报文中必须把低字节放在前面，且高字节放在后面。
*/
unsigned short crc16(unsigned char* puchMsg, unsigned char usDataLen)
{
    unsigned char uchCRCHi = 0xFF;      //CRC 的高字节初始化
    unsigned char uchCRCLo = 0xFF;    //CRC 的低字节初始化
    unsigned short   uIndex;              //CRC 查询表索引
    while (usDataLen--)
    {
        uIndex = uchCRCHi ^ *puchMsg++;       //计算CRC
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];
        uchCRCLo = auchCRCLo[uIndex];
    }
    return (((unsigned short)(uchCRCHi) << 8) | uchCRCLo);
}

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

测试代码:

```
#####################################################################
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.