傲蓝

单圈绝对值编码器 A40S06

接口

接口	RS485 / RS422 / CANbus
通讯协议	MODBUS RTU(RS485) / SSI(RS422) / CAN2.0
诊断	内存、接口自诊断
编程功能	节点号，波特率，总圈数，分辨率，预设(原始点)，计数方向，温度
传输速率	RS485: 9600~115200bps RS422: 1Mbps CANbus: 1Mbps
接口响应时间	<1ms

输出

输出驱动器	RS422/RS485/CAN总线差分

电气数据

电源电压	DC5V 或 DC10 ~ 24 V（电源符合EN 50178）
电流消耗	< 100 mA
功耗	≤ 1W
启动时间	< 100ms
反极性保护	是
短路保护	是
EMC:发射干扰	EN 61000-6-4
EMC:抗干扰	EN 61000-6-2
MTTF	50000小时，在40 ℃下

传感器

技术	磁电
单圈分辨率	32768cpr
精度(INL)	±0.1°
码制	二进制码

环境规格

外壳防护等级 (轴)	IP57
外壳防护等级 (外壳)	IP65
工作温度	-40°C～+85°C
存储温度	-40°C～+105°C
湿度	98%相对湿度，无凝结状态

机械数据

连接帽材料	不锈钢 V4A (1.4404; 316L)
外壳材料	45#钢
外壳涂层	喷涂漆层
法兰形式	夹紧法兰, ø40 mm
法兰材料	铝合金
轴的类型	实心轴，单面平坦型轴，长度 = 15 mm
轴的直径	ø 6 mm
转子转动惯量	≤ 15 gcm²
摩擦力矩	≤ 1 Ncm @ 20 °C
最高机械速度	≤ 6000 rpm
耐冲击性	≤ 100g (6毫秒半弦波, EN 60068-2-27)
耐振动性	≤ 10g (10 Hz ~ 200 Hz, EN 60068-2-6)
长度	55 mm
重量	300 g

电气连接

连接方向	径向
连接类型	1 × 防水接头

产品寿命周期信息

产品寿命周期信息	现有产品

信号芯线定义(RS485 RTU/ RS422 SSI):

PCB序号	芯线颜色	标签	说明
1	红色	VDD	供电电源
2	黄色	485A/SSI DATA+/CANH	RS485A/SSI数据+/CAN_H
3	黄白	485B/SSI DATA-/CANL	RS485B/SSI数据-/CAN_L
4	绿色	NA/SSI CLK+	未用/SSI时钟+
5	绿白	NA/SSI CLK-	未用/SSI时钟-
6	黑白	STATUS	设置编码器状态(1--Ready,0--Busy)
7	红白	SETPOS	设置原始位置/恢复出厂设置
8	黑色	GND	接地

SETPOS 是外部复用输入引线，具备两种功能：

1. 恢复出厂默认设置：

编码器上电前，预先将 SETPOS 与 GND 短接，上电 3 秒后，将 SETPOS 与 GND 断开并悬空。

2.设置原始位置：

编码器通电时，将 SETPOS 与 GND 短接 1 秒后，将 SETPOS 与 GND 断开并悬空。

订货选择信息

订货代码	描述
A40S06R	RS485接口/RTU协议
A40S06S	RS422接口/SSI协议
A40S06C	CANbus接口/CAN2.0b协议

RS485 MODBUS RTU 接口通讯方式（主动/被动模式）

主动模式只适合单机运行模式
被动模式适合单机/联网模式
出厂默认通讯参数为:
被动等待命令模式
从地址：0x01
数据地址：41800
波特率：115200/8/N/1

I)编码器位置数据请求命令：

1. 主控端发送 MODBUS RTU 命令帧：
发送数据(HEX)：0x01 0x03 0xA3 0x48 0x00 0x02 0x66 0x59
其中：
0x01: 从设备地址(出厂默认为0x01,可设置范围0x01~0xFE)
0x03: 读寄存器功能
0xA3 0x48: 数据寄存器首地址(出厂默认为41800,可设置范围40000~49999)
0x00 0x02：读取数据字个数(2个16 bits数据，分别对应圈数值和角度值)
0x66 0x59: CRC 校验

2. 主控端接收来自编码器的数据帧：
接收数据(HEX)：0x01 0x03 0x04 0x07 0x08 0x09 0x0A 0xFC 0xD2
其中：
0x01: 从设备地址
0x03: 读寄存器功能
0x04: 寄存器字节数量
0x07 0x08: 圈数值 = 0x0708 = 1800 (max.4095)
0x09 0x0A: 角度值 = 0x090A = 2314 (max.16383)
0xFC 0xD2: CRC 校验

II)编码器内部温度读出命令：

1. 主控端发送 MODBUS RTU 命令帧：
发送数据(HEX)：0x01 0x03 0xA3 0x4A 0x00 0x01 0x87 0x98
其中：
0x01: 从设备地址
0x03: 读寄存器功能
0xA3 0x4A: 温度值寄存器地址(即数据首地址+2)
0x00 0x01：读取数据字个数(1个16 bits数据，对应温度值)
0x87 0x98: CRC 校验

2. 主控端接收来自编码器的数据帧：
接收数据(HEX)：0x01 0x03 0x02 0x00 0x35 0x78 0x53
其中：

0x01: 从设备地址
0x03: 读寄存器功能
0x02: 寄存器字节数量
0x00 0x30: 温度值 = 0x0035 = 53(℃)
0x78 0x53: CRC 校验

III)编码器内部参数修改命令：

以下命令只适合单机设置时运行，并且默认处于上电锁定状态，为防止意外修改参数，须先执行解锁命令才能进行后续各项参数的设置。

参数设置解锁命令：
发送命令(HEX)：0x55 0xAA 0xF7 0x7F 0x0D 0x0A
接收数据(ASCII): 如设置成功板载指示灯会闪烁 1 次。

一、参数读取命令：
发送命令(HEX)：0x55 0xA0 0xXX 0xXX 0x0D 0x0A (0xXX 表示可以是任意数，下同)
接收数据(ASCII): 依次收到可记圈数和分辨率、波特率、子地址、数据地址、计数方向、圈数预设值、角度预设值、主动模式、时间间隔、圈数间隔、角度间隔，如设置成功板载指示灯会闪烁 1 次。

二、波特率设置命令：
发送命令(HEX)：0x55 0xA1 Baudrate 0xXX 0x0D 0x0A
Baudrate(Byte)取值范围为 0~4，分别对应波特率:
0--9600bps, 1--19200bps, 2--38400bps, 3--57600bps, 4--115200bps
接收数据(ASCII): 收到设定的波特率值，如设置成功板载指示灯会闪烁 1 次。

三、子地址设置命令：
发送命令(HEX)：0x55 0xA2 Node_address 0xXX 0x0D 0x0A
Node_address(Byte)取值范围为 1~254
接收数据(ASCII): 收到设定的子地址值，如设置成功板载指示灯会闪烁 1 次。

四、数据地址设置命令：
发送命令(HEX)：0x55 0xA3 Modbus_ADDH Modbus_ADDL 0x0D 0x0A
Modbus_ADD (Int)取值范围为 00001~49999
接收数据(ASCII): 收到设定的数据地址值，如设置成功板载指示灯会闪烁 1 次。

五、计数方向设置命令：
发送命令(HEX)：0x55 0xA4 Direction 0xXX 0x0D 0x0A
Direction (Byte)取值范围为 0 或 1，分别对应计数方向: 0--CW, 1--CCW
接收数据(ASCII): 收到设定的计数方向，如设置成功板载指示灯会闪烁 1 次。

六、圈数预设置命令：
发送命令(HEX)：0x55 0xA5 Preset_Turns_H Preset_Turns_L 0x0D 0x0A
Preset_Turns (Int)取值范围为 0~Max.Turns -1
接收数据(ASCII): 收到设定的圈数预设值，如设置成功板载指示灯会闪烁 1 次。

七、角度预设置命令：
发送命令(HEX)：0x55 0xA6 Preset_Angle_H Preset_Angle_L 0x0D 0x0A
Preset_Angle (Int)取值范围为 0~Max.Angle -1
接收数据(ASCII): 收到设定的角度预设值，如设置成功板载指示灯会闪烁 1 次。

八、软加载预设值命令：
发送命令(HEX)：0x55 0xA7 0xXX 0xXX 0x0D 0x0A
接收数据(ASCII): 无，如设置成功板载指示灯会闪烁 1 次。

九、编码器主动发送数据启动命令：
发送命令(HEX)：0x55 0xB0 Node_address OTP_Send 0x0D 0x0A
Node_address 为需要启动的编码器
OTP_Send 默认值为 0xFF,如果被设置为特定值 **0x95**,则此编码器被设置为强主动模式，一上电即开始主动发送数据（可通过 SETPOS 引线恢复出厂默认设置）
接收数据(HEX): 收到 MOBUS RTU 响应数据帧（具体可参照前页描述），如发送成功板载指示灯会闪烁 1 次。该命令执行后，编码器即进入主动连续发送数据状态，若要转入被动模式，需重新上电。

十、编码器主动发送模式设置命令：
发送命令(HEX)：0x55 0xB1 Master_Mode 0xXX 0x0D 0x0A
Master_Mode(Byte)取值范围为 0 或 1，分别对应两种主动模式: 0--按照时间间隔, 1--按照空间间隔
接收数据(ASCII): 收到设定的主动模式，如设置成功板载指示灯会闪烁 1 次。

十一、 主动发送时间间隔设置命令：
发送命令(HEX)：0x55 0xB2 Interval_TH Interval_TL 0x0D 0x0A
Interval_T (int)取值范围为 0~65534，单位为毫秒
接收数据(ASCII): 收到设定的时间间隔值，如设置成功板载指示灯会闪烁 1 次。

十二、 主动发送空间间隔圈数设置命令：
发送命令(HEX)：0x55 0xB3 Interval_RH Interval_RL 0x0D 0x0A
Interval_R (int)取值范围为 0~4094，单位为圈
接收数据(ASCII): 收到设定的圈数间隔值，如设置成功板载指示灯会闪烁 1 次。

十三、 主动发送空间间隔角度设置命令：
发送命令(HEX)：0x55 0xB4 Interval_AH Interval_AL 0x0D 0x0A
Interval_A (int)取值范围为 0~16382，单位为最小分辨率
接收数据(ASCII): 收到设定的角度间隔值，如设置成功板载指示灯会闪烁 1 次。

****

CRC 生成函数代码示例（MODBUS RTU 模式）

```
//unsigned char *puchMsg ; /* message to calculate CRC upon */
//unsigned int usDataLen ; /* quantity of bytes in message */

unsigned int CRC16(unsigned char *puchMsg, unsigned int usDataLen)
{
    unsigned char uchCRCHi = 0xFF ; /* high CRC byte initialized */
    unsigned char uchCRCLo = 0xFF ; /* low CRC byte initialized */
```

```
    unsigned char uIndex ; /* will index into CRC lookup table*/
    while (usDataLen--) /* pass through message buffer*/
    {
        uIndex = uchCRCHi ^ *puchMsg++ ; /* calculate the CRC*/
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex} ;
        uchCRCLo = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

High Order Byte Table

```
/* 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
} ;
```

Low Order Byte Table

```
/* Table of CRC values for low-order 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
} ;
```

CAN 总线通讯方式（CAN2.0b）：

出厂默认通讯参数为:
设备 NodeID：0x03
从模式标准帧：id = 0x580+NodeID
波特率：1Mbps

要求主控端标准帧 id = 0x80(广播)或 0x600+NodeID(单点)

一、编码器设置解锁命令:
除编码器位置数据读出命令外，其他设置命令都需经此解锁命令后才可执行，编码器每次上电后处于默认锁定状态。

1. 主控端发送 CAN 命令帧：

```
cansndbuf[0] = 0x55;          //解锁命令标志 1
cansndbuf[1] = 0xAA;          //解锁命令标志 2
cansndbuf[2] = 0xF7;          //解锁命令标志 3
cansndbuf[3] = 0x7F;          //解锁命令标志 4
cansndbuf[4] = 0x55;          //解锁命令标志 4
cansndbuf[5] = 0x0D;          //解锁命令标志 5
cansndbuf[6] = 0x0A;          //解锁命令标志 6
cansndbuf[7] = NodeID;        // NodeID 为该编码器从设备地址
CAN1_Send_Msg(cansndbuf,8);
```

2. 主控端接收编码器返回的 CAN 应答帧：

```
unsigned char getdata[2];

CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
if((RxMessage.StdId==NodeID)&&(RxMessage.DLC==2)) //NodeID 是为该编码器设置的从设备地址
   {
      getdata[0] = RxMessage.Data[0];  // getdata[0]应该等于 0xAA
      getdata[1] = RxMessage.Data[1];  // getdata[1]应该等于 NodeID
   }
//并且编码器从端 LED 闪烁 3 次，表示解锁成功
```

二、编码器从地址设置命令(注：此命令只可单机运行)：

1. 主控端发送 CAN 命令帧：

```
cansndbuf[0] = 0xA0;          //从设备地址设置命令标志
cansndbuf[1] = NodeID;        //将该编码器的从设备地址设置为 NodeID，允许设置范围 0x00~0x7F
CAN1_Send_Msg(cansndbuf,2);
```

2. 主控端接收编码器返回的 CAN 应答帧：

```
unsigned char getdata[2];

CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
if((RxMessage.StdId==NodeID)&&(RxMessage.DLC==2)) // NodeID 是为该编码器设置的从设备地址
   {
      getdata[0] = RxMessage.Data[0];  // getdata[0]应该等于 0xA0
      getdata[1] = RxMessage.Data[1];  // getdata[1]应该等于 NodeID
   }
//并且编码器从端 LED 闪烁 3 次，表示从地址设置成功
```

三、编码器位置数据读出命令(仅此命令不受设置锁定命令影响，任何时候都可以有效执行)：

1. 主控端发送 CAN 命令帧：

主控端标准帧 ID 设置为 0x80 或 0x600+NodeID，发送一帧空数据即可

2. 主控端接收编码器发出的 CAN 位置数据帧：

```
unsigned int Turns,Angle;

CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
if((RxMessage.StdId==NodeID)&&(RxMessage.DLC==4))
   {
      Turns = RxMessage.Data[0];
      Turns = (Turns << 8)|RxMessage.Data[1];    //对于单圈型编码器，圈数读数恒为 0
      Angle = RxMessage.Data[2];
      Angle = (Angle << 8)|RxMessage.Data[3];
   }
```

四、编码器内部参数读出命令：

1. 主控端发送 CAN 命令帧：

```
cansndbuf[0] = 0xA2;    //参数读出命令标志
cansndbuf[1] = NodeID;// NodeID 为该编码器从设备地址
CAN1_Send_Msg(cansndbuf,2);
```

2. 主控端接收编码器发出的 CAN 参数数据帧：

```
unsigned char    NodeID,Baudrate,Direction;
unsigned int PreTurns,PreAngle;

CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
if((RxMessage.StdId== NodeID)&&(RxMessage.DLC==7))
   {
      NodeID = RxMessage.Data[0];
      Baudrate = RxMessage.Data[1];
      Direction = RxMessage.Data[2];
      PreTurns = RxMessage.Data[3];
```

```
    PreTurns = (PreTurns << 8)|RxMessage.Data[4]; //对于单圈型圈数读数恒为 0
    PreAngle = RxMessage.Data[5];
    PreAngle = (PreAngle << 8)|RxMessage.Data[6];
  }
//并且编码器从端 LED 闪烁 3 次，表示参数读出成功
```

五、编码器内部参数组一设置命令：

1. 主控端发送 CAN 命令帧：

```
cansndbuf[0] = 0xA3;          //第一组参数设置命令标志
cansndbuf[1] = NodeID;        // NodeID 指定需要设置参数的编码器
cansndbuf[2] = Baudrate;      //设置范围 0x01~0x12,对应 18 个不同波特率（附表 1）
cansndbuf[3] = Direction;     //旋转方向  0x00--CW , 0x01—CCW
cansndbuf[4] = PreTurns_H; //圈数预设值高字节(对于单圈型设置为0)
cansndbuf[5] = PreTurns_L; //圈数预设值低字节(对于单圈型设置为0)
cansndbuf[6] = PreAngle_H; //角度预设值高字节
cansndbuf[7] = PreAngle_L; //角度预设值低字节
CAN1_Send_Msg(cansndbuf,8);
```

2. 主控端接收编码器返回的 CAN 应答帧：

```
unsigned char getdata[2];

CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
if((RxMessage.StdId==NodeID)&&(RxMessage.DLC==2)) //NodeID 是为该编码器设置的从地址
  {
    getdata[0] = RxMessage.Data[0];  // getdata[0]应该等于 0xA3
    getdata[1] = RxMessage.Data[1];  // getdata[1]应该等于 NodeID
  }
//并且编码器从端 LED 闪烁 3 次，表示参数设置成功
```

六、设置参考位置命令(与外部 SETPOS 引脚功能相同)：

1. 主控端发送 CAN 命令帧：

```
cansndbuf[0] = 0xA4;          //设置参考位置命令标志
cansndbuf[1] = NodeID;        //将该地址编码器的当前位置，设置为 PreTurns &PreAngle 中的值
CAN1_Send_Msg(cansndbuf,2);
```

2. 主控端接收编码器返回的 CAN 应答帧：

```
unsigned char getdata[2];

CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);
if((RxMessage.StdId==NodeID)&&(RxMessage.DLC==2)) /
  {
    getdata[0] = RxMessage.Data[0];  // getdata[0]应该等于 0xA4
    getdata[1] = RxMessage.Data[1];  // getdata[1]应该等于 NodeID
  }
//并且编码器从端 LED 闪烁 3 次，表示参考位置设置成功
```

附表 1：（波特率代号对应值）

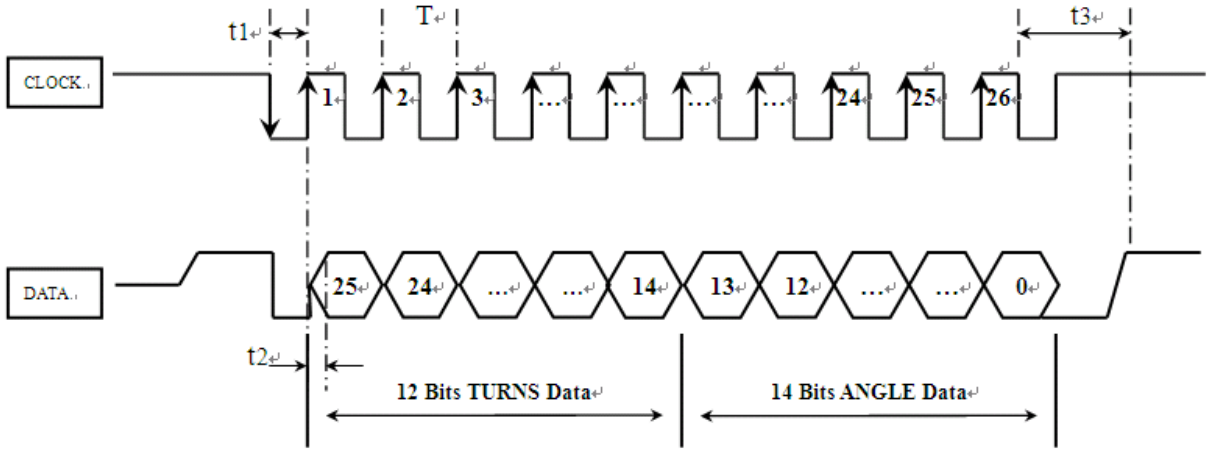
设置值	对应值	设置值	对应值	设置值	对应值
1	1M	7	300K	13	90K
2	900K	8	250K	14	80K
3	800K	9	200K	15	60K
4	600K	10	150K	16	50K
5	500K	11	125K	17	40K
6	400K	12	100K	18	30K

基于 STM32 CAN 时钟 36MHz 的内部时序参数配置参考:

```
    SJW          BS1          BS2          Prescaler
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      6          // 1M
CAN_SJW_1tq  CAN_BS1_4tq  CAN_BS2_3tq      5          // 900K
CAN_SJW_1tq  CAN_BS1_5tq  CAN_BS2_3tq      5          // 800K
CAN_SJW_1tq  CAN_BS1_6tq  CAN_BS2_3tq      6          // 600K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      12         // 500K
CAN_SJW_1tq  CAN_BS1_5tq  CAN_BS2_3tq      10         // 400K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      20         // 300K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      24         // 250K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      30         // 200K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      40         // 150K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      48         // 125K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      60         // 100K
CAN_SJW_1tq  CAN_BS1_4tq  CAN_BS2_3tq      50         // 90K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      75         // 80K
CAN_SJW_1tq  CAN_BS1_6tq  CAN_BS2_3tq      60         // 60K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      120        // 50K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      150        // 40K
CAN_SJW_1tq  CAN_BS1_6tq  CAN_BS2_3tq      120        // 30K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      300        // 20K
CAN_SJW_1tq  CAN_BS1_3tq  CAN_BS2_2tq      600        // 10K
CAN_SJW_2tq  CAN_BS1_6tq  CAN_BS2_4tq      600        // 5K
CAN_SJW_2tq  CAN_BS1_6tq  CAN_BS2_4tq      1000       // 3K
CAN_SJW_2tq  CAN_BS1_10tq CAN_BS2_6tq      1000       // 2K
```

RS422 SSI 接口通讯方式

SSI 时序图及示例代码



Time Description

t1 > 0.45us

t2 < 0.40us

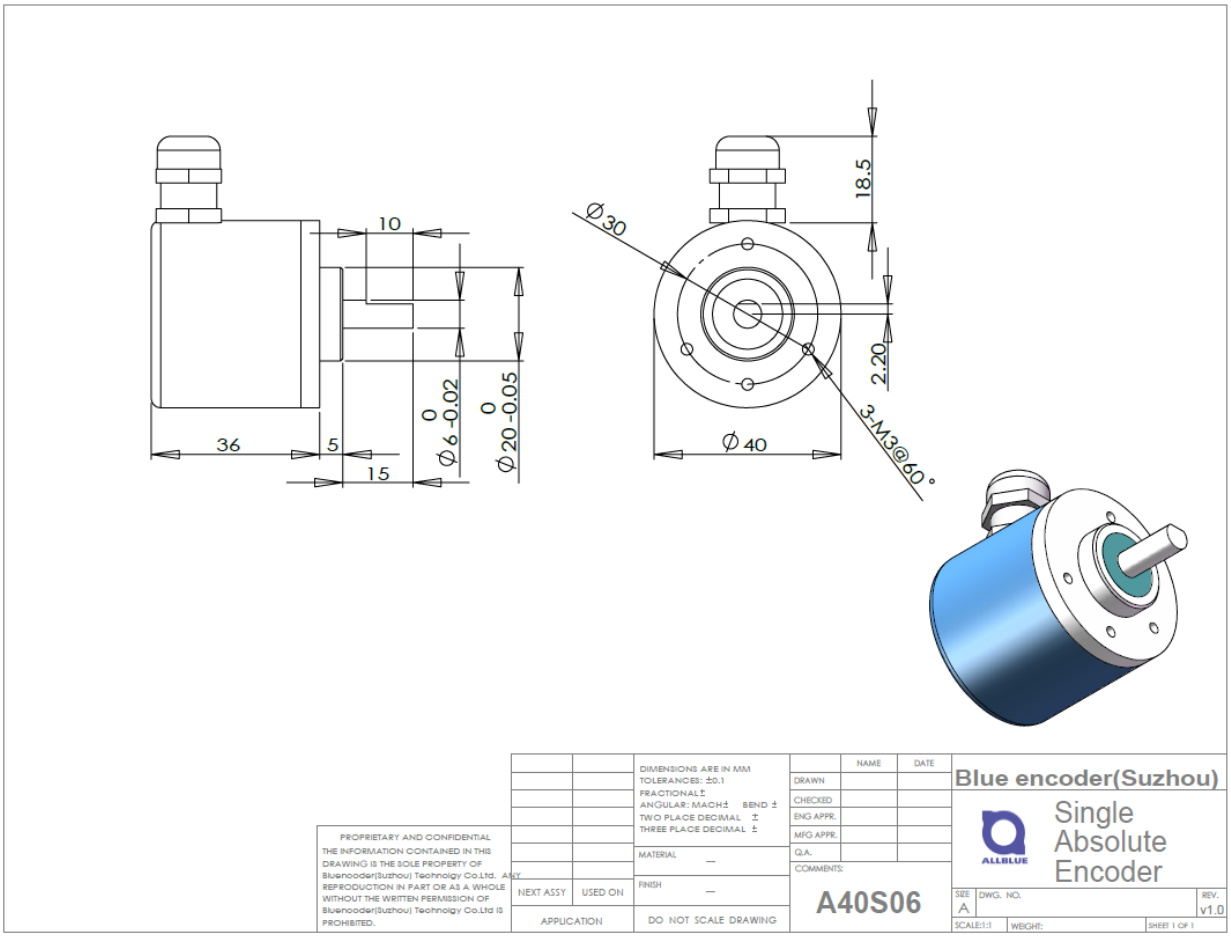
t3 = 12us ~ 30us

T = 1us ~ 11us

C Code Example

```
#define databit 26
uint32 Getdata;
uint16 TURNS,ANGLE;
void main(void)
{
   uint i;
   SSI_CLOCK = 1;              //SSI clock wire
   Delay_us(1);                //Delay 1us
   SSI_CLOCK = 0;              //Latch data into output shift register
   Delay_us(1);                //Delay 1us
   for(i=0;i<databit;i++)      //Get all of data from encoder
   {
        Getdata <<= 1;         //Shift one higher bit left
        SSI_CLOCK =0;
        SSI_CLOCK =1;      //Get one bit of data at clock rising edge
        Getdata |= SSI_DATA; //Read one bit from SSI data wire
   }
   TURNS = Getdata >> 14; //Get 12 bit Turns
   ANGLE = Getdata & 0x3fff; //Get 14 bit Angle
   Delay_us(30);               //Delay 30us
}
```

产品尺寸图：



联系方式：

苏州傲蓝电子科技有限公司
地址：中国浙江省海宁市双联路 128 号 1 栋 5 楼东
电话：+86 573 80771560/80771561；+86 13901668071
公司网址：www.bluencoder.com； 电子邮箱：newmoon002@163.com

所有尺寸均为毫米，此信息和图纸的目的旨在于一般性的介绍，请参阅“下载”部分技术图纸进行了解，