KingKong.tech
金 钢 科 技

# MPT 磁编码器数据手册 v0.3

KingKong.tech
金 钢 科 技

# MPT 磁编码器

**MPT 磁编码器超紧凑款产品，使用金钢科技图案式磁技术，在极其有限的体积内可以实现高精度的测量，达到类光电的分辨率与精度，同时拥有高强的抗环境干扰能力。**

该编码器由磁电技术驱动，且拥有独特的干扰屏蔽技术，编码器内部拥有多个高精度霍尔传感器测量转子磁环的磁场变化情况，并且拥有金钢科技提供的精密标定技术成形，每个产品出厂都拥有独一无二的磁场标定数据，提供最佳的测量精度。

独特的动、静件公差配合安装技术在简化用户安装的同时，也为测量精度保驾护航。

分离式的磁电方案拥有更强的环境承受力，如振动、灰尘、油污等，且可以运转于超高速情况中，均不会影响编码器精度与工作寿命。

超薄紧密的中空结构，更方便在各种应用场景中嵌入。

- 24 位绝对值输出
- ±0.01°绝对定位精度
- 超紧凑（径向单边 5mm）
- 内径每隔 5mm 一个型号
- 定、转子公差配合安装技术
- 磁干扰屏蔽技术
- 中空不限制安装位置
- 最高速度 8,000 rpm
- 唯一数据标定成形
- 多圈电池模式
- 多种输出接口
- 各种环境干扰抗性

# 型号

## MPT–20–30–A–24–S–N–A–M

**转子内径**

具体请查看尺寸

**定子外径**

具体请查看尺寸

**输出种类**

A — 绝对值

**输出参数**

24 — 24 位单圈

| | |
|---|---|
| 掉电后多圈清零<br>多圈仅上电有效 | 24M — 24 位单圈+普通多圈 |
| 掉电后使用电池供电<br>仍可保持多圈测量 | 24BM — 24 位单圈+电池多圈 |
| 掉电后仅允许±90°转动<br>无需电池 | 24FM — 24 位单圈+闪存多圈 |

**输出接口**

S — SSI (1) 不推荐使用

B — BISS–C 推荐使用

R — RS485 推荐使用

A — RS422

T — T485(兼容 17 位多摩川) (2) 推荐使用

T — T485(兼容 23 位多摩川) (2) 推荐使用

D — BUS（高速总线）

P — PERIOD（周期发送）

**输入电压**

N — 5V

**工作温度**

A — –40 ~ 85 ˚C

B — –40 ~ 105 ˚C

C — –40 ~ 125 ˚C

**附加**

M — 无附加

波特率 — 接口为 R、A 时

(1) SSI 协议无 CRC 校验，建议使用相同硬件的 BISS–C 代替以提高可靠性

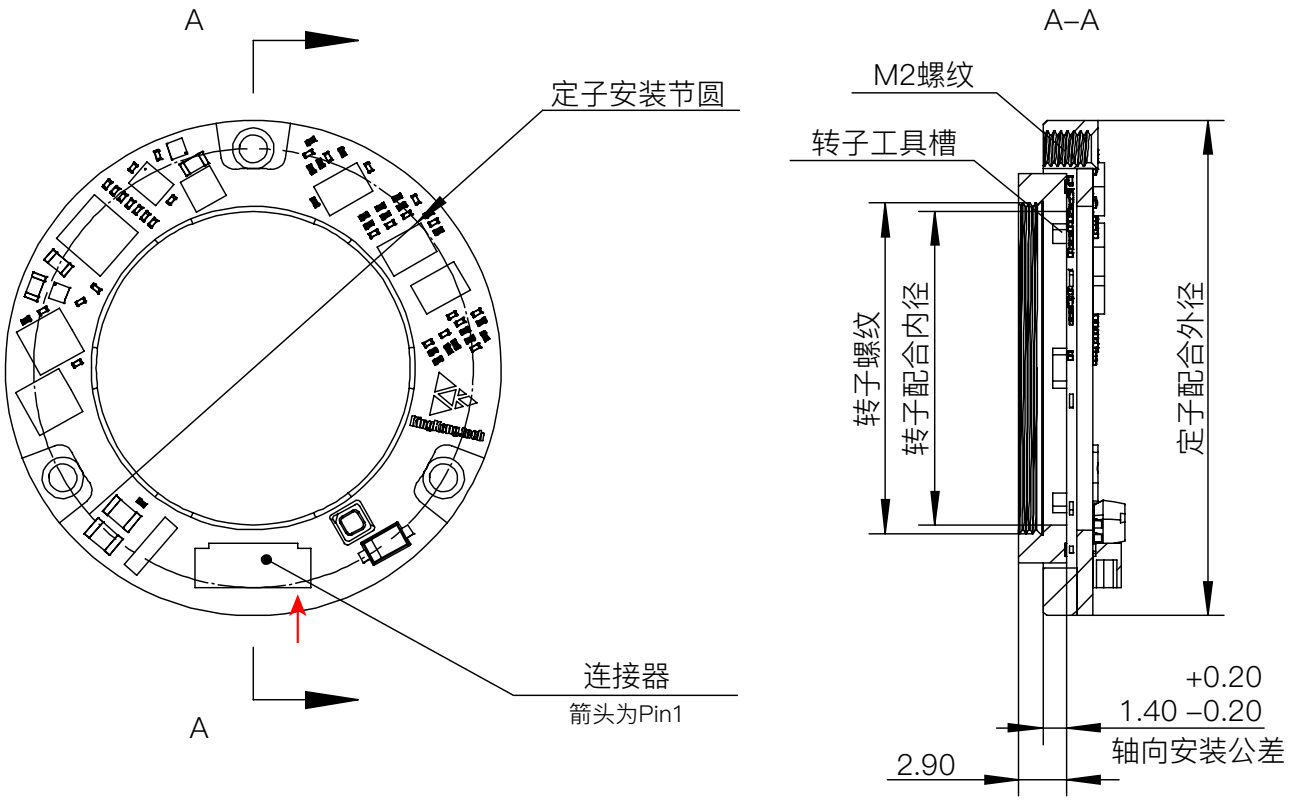
(2) 多摩川协议选择 17 位时输出参数为 17M–T、17BM–T、17FM–T，选择 23 位则为 23M–T、23BM–T、23FM–T

# 尺寸

| 系列 | 尺寸 | 内转子<br>螺纹 | 转子<br>配合内径(H7) | 定子外径<br>(h7) | 定子安装<br>节圆 | 连接器<br>型号 | 整体厚度 |
|---|---|---|---|---|---|---|---|
| **10** | MPT-10-20 | M10x0.4mm | 9 | 20 | 16.6 | SM06B-XSRS-ETB | 6.7 |
| **15** | MPT-13-25 | M13x0.4mm | 12 | 25 | 21.6 | SM06B-XSRS-ETB | |
| | MPT-14-25 | M14x0.4mm | 13 | | | SM06B-XSRS-ETB | |
| | MPT-15-25 | M15x0.4mm | 14 | | | SM06B-XSRS-ETB | |
| **20** | MPT-20-30 | M20x0.4mm | 19 | 30 | 26.6 | SM08-SURS-TF | |
| **25** | MPT-25-35 | M25x0.4mm | 24 | 35 | 31.6 | SM08-SURS-TF | |
| **30** | MPT-30-40 | M30x0.4mm | 29 | 40 | 36.6 | SM08-SURS-TF | |
| **35** | MPT-35-45 | M35x0.4mm | 34 | 45 | 41.6 | SM08-SURS-TF | |

1. 下载 3D 模型：https://kingkong.tech/encoder/MPT

# 图纸




定子安装时，可以使用 M1.6 螺丝从 M2 螺纹孔中穿过，也可以使用 M2 螺纹配合 M2 螺丝来做锁紧

转子安装时，使用专用工具拧紧，具体可参见《MPT 设计安装建议》。

# 电气连接

## 连接器

| | 线对板连接器 |
|---|---|
| **型号** | 参见尺寸 |
| **种类** | 线对板型 |
| **连接线** | 4PxAWG32 铁氟龙双绞线 |

## 引脚

| 引脚 | 颜色 | S | B | A | R | T | D | P |
|---|---|---|---|---|---|---|---|---|
| | | SSI | BISS-C | RS422 | RS485 | T485 | BUS | PERIOD |
| 1 | 红 | +5V | | | | | | |
| 2 | 黑 | 0V（GND） | | | | | | |
| 3 | 蓝 | Clock + | MA + | RX + | A | A | A | – |
| 4 | 绿 | Clock – | MA – | RX – | B | B | B | – |
| 5 | 黄 | Data + | SLO + | TX + | – | – | – | TX + |
| 6 | 橙 | Data – | SLO – | TX – | – | – | – | TX – |
| 7 | 白 | 电池 + | 电池 + | 电池 + | 电池 + | 电池 + | 电池 + | 电池 + |
| 8 | 灰 | 电池 – | 电池 – | 电池 – | 电池 – | 电池 – | 电池 – | 电池 – |

* 电池+/–仅出现在电池多圈（BM）版本中

* 电池–于编码器内部与 GND 相连

* 屏蔽层在驱动端建议接地

# 参数规格

## 系统参数

| | |
|---|---|
| 安装方式 | 轴向中空 |
| 精度 | ±0.01° |

## 电气参数

| | |
|---|---|
| 电源 | 4.5 ~ 5.5 V |
| 电池 | 2.7 ~ 3.6 V |
| 启动时间 | 15 ms |
| 连接方式 | 线对板连接器 |
| 电流 | ≈ 100 mA |
| 低功耗电流 | ≈ 6 μA (电池电压 3.6V，于低功耗静态不动检测模式下) |
| 静电保护 | HBM, max. ±2 kV<br>CDM, max. ±1 kV |

## 机械参数

| | |
|---|---|
| 转子托架 | 不锈钢 |
| 定子托架 | 铝合金 |

## 环境参数

| | |
|---|---|
| 工作温度 | –40 ~ 85 ℃ / –40 ~ 105 ℃ / –40 ~ 125 ℃ |

# 参数详解

## 最大转速

采取非接触式结构，转子与定子间无摩擦存在，最大转速可以极高，更高转速应用环境。

## 环境干扰

磁编码器的测量原理可以使得其对振动拥有一定的抗性，并且不会对编码器本体造成致命性的损伤，其对油污、灰尘等非导磁性物体，均拥有强抗干扰能力，不会对测量造成影响。

## 外部磁场干扰

MPT 编码器拥有独特的抗电磁干扰技术，电磁屏蔽技术可以对由环境（如电机）中传播过来的电磁干扰进行屏蔽，且磁环的磁场强度一般远高于外界的磁场干扰，两者配合可以稳定地抵抗外部干扰。

## 精度

在加工过程中，磁环的充磁过程拥有一致性低的问题。而金钢科技每个编码器均经过出厂标定，其内部拥有组合中对应磁环的唯一数据信息。

# 绝对值系列

## 输出格式

| | |
|---|---|
| **单圈** | |
| 输出角度数据 | |
| **多圈** | |
| 输出圈数与角度数据 | 断电后无法保持圈数，再次上电后圈数重新从零计数 |
| **电池多圈** | |
| 输出圈数与角度数据 | 断电后使用电池供电，编码器进入低功耗模式，此时仍可对电机的转动进行测量，再次上电后仍可读出有效圈数 |

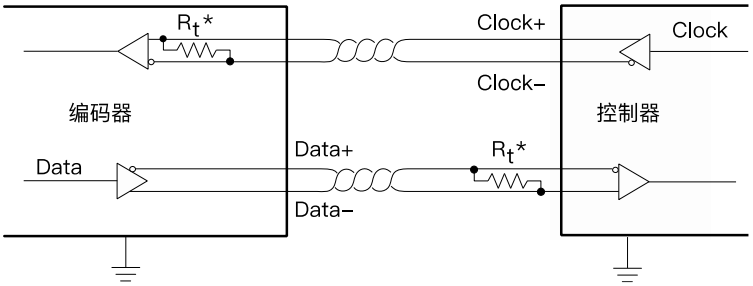
## 绝对值参数

| | |
|---|---|
| **单圈位数** | 24 bit |
| **最大转速** | 8,000 rpm |
| **更新频率** | 50 kHz |
| **重复精度** | 0 ~ ±1 bit（根据位数选择变化） |
| **输出接口** | SSI、BISS-C、RS485、RS422、T485、BUS、PERIOD |

SSI

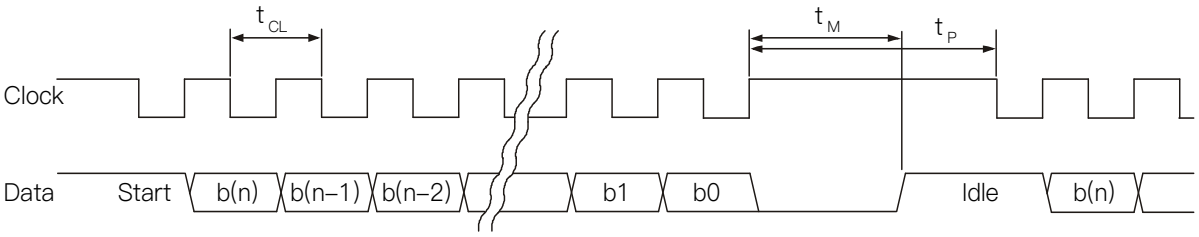
## SSI 协议接口

电气连接图：




该接口使用四线制，分别为 Clock 正反相与 Data 正反相。且 Clock 的终端电阻已经被集成进了编码器内部，用户需在控制器侧 Data 端接终端电阻或其它阻抗匹配方案。

时序图：




该协议使用 Clock 来同步获取时钟序列，当第一个下降沿到来时，系统则会锁存当前的数据，以及从 MSB 开始于每一个 Clock 的上升沿写入数据至 Data 线，而在控制器端，则会在 Clock 的下降沿来读取 Data 线上的数据，往复至到 LSB 被控制器读取到。

传输完成后当 $t_M$ 传输时间结束后，Data 线会处理高电平，Clock 信号必须保持高电平状态直到下一次读取被允许，也就是 $t_P$ 时间后。$t_{CL}$ 必须小于 $t_M$，而且在任意读取操作进行时，都可以使时间超过 $t_M$ 从而终断读取。

SSI

时序相关：

| 参数 | 符号 | 最小值 | 典型值 | 最大值 |
|---|---|---|---|---|
| 时钟频率 | $t_{CL}$ | 400 ns | | 14 µs |
| 时钟频率 | $t_{CL}$ | 110 kHz | | 1.5 MHz(1) |
| 传输超时 | $t_M$ | | 10 µs | |
| 暂停时间 | $t_P$ | 20 µs | | |

(1) 如果 Clock 在第一个低电平处可保持 500ns，后面的时钟频率则最高可达 10MHz

数据格式：

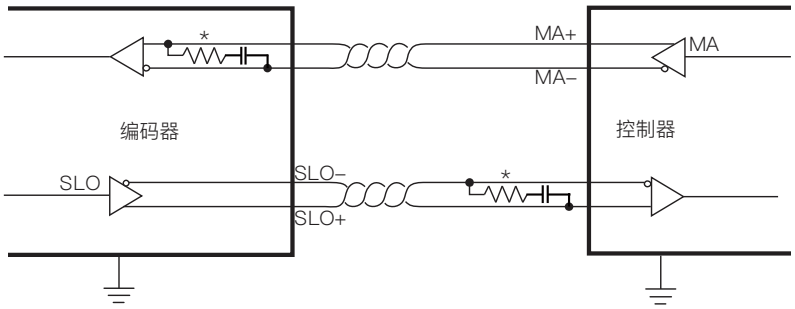
| 位 | b(23 + X) : b(8 + X) | b(7 + X) : b8 | b7 | b6 | b5 : b0 |
|---|---|---|---|---|---|
| 长度 | 16 bits | X bits | 1bit | 1bit | 6 bits |
| 数据 | 多圈计数 (1) | 单圈角度 | 错误位 | 警告位 | 状态位 |

(1) 多圈计数仅存在于多圈、电池多圈版本

状态位的具体描述，请参见后文状态位章节。

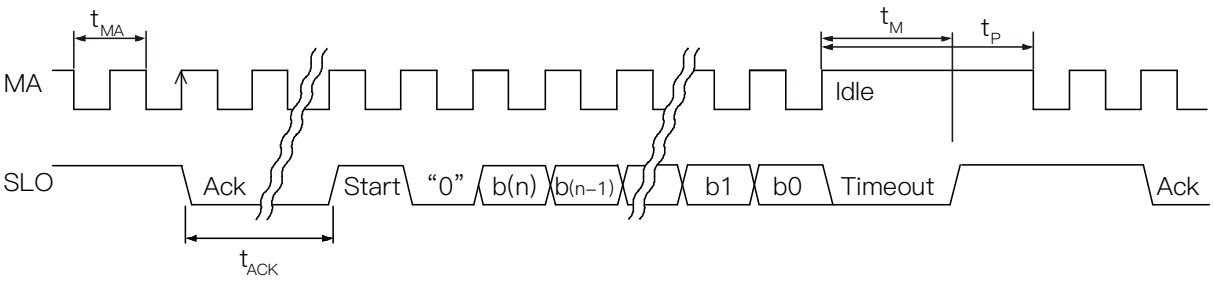
## BISS-C 协议接口

BiSS-C 电气连接图：




该接口使用四线制，分别为 MA 正反向与 SLO 正反向。且 MA 的终端电阻已经被集成进了编码器内部，用户需在控制器侧 SLO 端接终端电阻或其它阻抗匹配方案。

时序图：




该协议使用 MA 来同步获取时钟序列，MA 线空闲时为高电平，当第一个下降沿到来时，系统则会锁存当前的数据，通信将于第一个下降沿开始，编码器将会于第二个 MA 上升沿时配置 SLO 为低电平，从"0"后 MSB 开始于每一个 MA 的上升沿写入数据至 SLO 线，而在控制器端，则会在 MA 的下降沿来读取 SLO 线上的数据，往复至到 LSB 被控制器读取到。

时序参数：

| 参数 | 符号 | 最小值 | 典型值 | 最大值 |
|---|---|---|---|---|
| 时钟频率 | tMA | 400 ns | | 14 µs |
| 时钟频率 | f | 120 kHz | | 2.5 MHz(1) |
| ACK 长度 | tACK | | 5 bits | |
| 传输超时 | tM | | 10 µs | |
| 暂停时间 | tP | 20 µs | | |

(1) 用户如果拥有相位补偿技术来补偿差分转换之间的延时，频率则最高可达 10MHz

传输完成后当 $t_M$ 传输时间结束后，SLO 线会处理高电平，MA 信号必须保持高电平状态直到下一次读取被允许，也就是 $t_P$ 时间后。$t_{CL}$ 必须小于 $t_M$，而且在任意读取操作进行时，都可以使时间超过 $t_M$ 从而终断读取。

数据格式：

| 位 | b(24 + X) : b(9 + X) | b(8 + X) : b8 | b7 | b6 | b5 : b0 |
|---|---|---|---|---|---|
| 长度 | 16 bits | X bits | 1bit | 1bit | 6 bits |
| 数据 | 多圈计数(1) | 单圈角度 | 错误位(2) | 警告位(3) | CRC(4) |

(1) 多圈计数仅存在于多圈、电池多圈版本

(2) 错误位为低电平有效，该位仅会在电池多圈版本内可能为低电平。当为 1 时，表示电池相关状态位正常，多圈数据可信；当为 0 时，表示状态位中的电池低压、电池中断错误触发，解决方法请参见状态位章节。

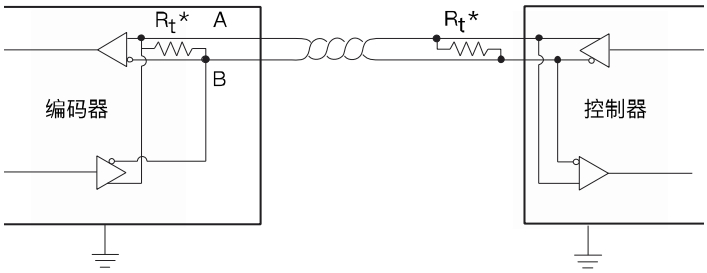
(3) 警告位为低电平有效。当为 1 时，表示无任何错误、警告触发；当为 0 时，表示至少有一项错误、警告触发。

(4) CRC 多项式为 x6 + x1 + 1（即 0x43），根据 BISS-C 协议要求，计算出的 CRC 会取反后再发送。附录 CRC-6 计算给出了可直接移植的计算代码，方便参考。

状态位的具体描述，请参见后文状态位章节。

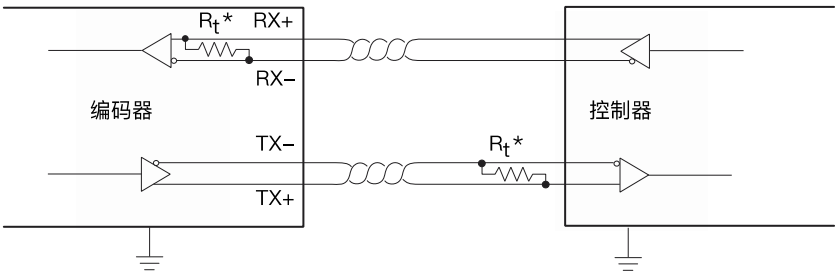
## RS485/RS422 协议接口

RS485 电气连接图：




该接口为二线制，其主要是差分的 A、B 相，两根线的终端都需要来并接终端电阻，编码器端的终端电阻已经集成进了编码器内部，用户需在控制器侧 A、B 接终端电阻或其它阻抗匹配方案。

RS422 电气连接图：




该接口使用四线制，分别为 TX 差分输出与 RX 的差分输入。且 RX 的终端电阻已经被集成进了编码器内部，用户需在控制器侧 RX 端接终端电阻或其它阻抗匹配方案。

RS485 与 RS422 的底层协议均为 UART，由于该协议没有时钟线，所以编码器与控制器必须工作在约定好的相同频率、数据格式下，才能完成数据的传输。

协议配置：

| 字符长度 | 奇偶校验 | 停止位 | 流控制 | 字节顺序 |
|---|---|---|---|---|
| 8 bit | 无 | 1 | 无 | LSB 优先 |

支持的波特率（若未在附加中标出，默认及推荐 B）：

| 代号 | A | B |
|---|---|---|
| 波特率（Mbps） | 0.1152 | 2.5 |

指令与数据：

| 指令 | 介绍 | | 输出参数 | N | 返回数据（N 个字节） | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 |
| 0x30 | 设置 | 零位(1) | – | 2 | C | CRC | | | | | | |
| 0x31 | 获取 | 位置 | 24 | 4 | A2 | A1 | A0 | CRC | | | | |
| | | | 24M<br>24BM<br>24FM | 6 | M1 | M0 | A2 | A1 | A0 | CRC | | |
| 0x64 | 获取 | 位置+状态 | 24 | 5 | A2 | A1 | A0 | S | CRC | | | |
| | | | 24M<br>24BM<br>24FM | 7 | M1 | M0 | A2 | A1 | A0 | S | CRC | |
| 0x74 | 获取 | 位置+温度(2) | 24 | 6 | A2 | A1 | A0 | T1 | T0 | CRC | | |
| | | | 24M<br>24BM<br>24FM | 8 | M1 | M0 | A2 | A1 | A0 | T1 | T0 | CRC |

(1) 设置零位需连续间隔发送指令“0x31”、发送命令“0x30”共 10 次，方能设置成功；当指令“0x30”返回计数值为 10 时，触发设置零位

(2) 温度信息为芯片的结温

上表中，字母对应数据为：

| M | A | C | S | T | CRC |
|---|---|---|---|---|---|
| 多圈数据 | 单圈角度 | 清零读数值 | 状态 | 温度(2) | CRC 校验(1) |

状态位的具体描述，请参见后文状态位章节。

(1) CRC 字节（CRC 多项式为 $x^8+x^7+x^4+x^2+x^1+1$，计算方法见附录 CRC–8 表($x^8+x^7+x^4+x^2+x^1+1$)）

(2) 温度计算方法为：int16_t temp = T0 | T1 << 8; float tempFloat = temp / 10.f; 单位°C

示例（24BM）：

如发送 0x31，得到 uint8_t Buffer[7];

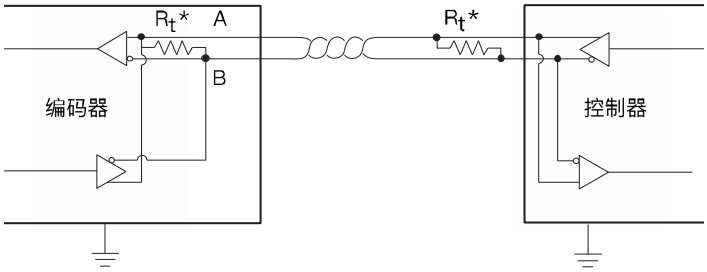
那使用时：

```
uint16_t multi = Buffer[0] << 8 | Buffer[1];
uint32_t angle = Buffer[2] << 16 | Buffer[3] << 8 | Buffer[4];
float angleFloat = angle / (float)(1 << 24) * 360;
```

## T485 协议接口

*本接口兼容多摩川协议

RS485 电气连接图：




该接口为二线制，其主要是差分的 A、B 相，两根线的终端都需要来并接终端电阻，编码器端的终端电阻已经集成进了编码器内部，用户需在控制器侧端接终端电阻或其它阻抗匹配方案。

T485 底层基于 RS485，拥有一定的通信协议。该接口接收 1Byte 的操作请求，根据请求数据返回对应的编码器数据，并在数据尾加上 CRC–8 以做校验。

协议配置：

| 字符长度 | 奇偶校验 | 停止位 | 流控制 | 字节顺序 |
|---|---|---|---|---|
| 8 bit | 无 | 1 | 无 | LSB 优先 |

操作请求数据：

| 数据位 | b7 ~ b3 | b2 | b1 | b0 |
|---|---|---|---|---|
| 数据内容 | 操作类型 | 0 | 1 | 0 |

返回数据：

| 字节 | B0 | B1 | B(2 ~ n) | B(n + 1) |
|---|---|---|---|---|
| 数据内容 | 操作请求(1) | 状态 | 返回数据 | CRC (2) |

(1) 返回的操作请求与发送的操作请求相同

(2) CRC 字节（CRC 多项式为 x8 + 1，计算方法见附录 CRC–8 计算）

B1，状态格式：

| 数据位 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|---|
| 数据内容 | 0 | 通信错误 | 编码器错误 | 0 | 0 | 0 | 0 | 0 |

B(2 ~ n)，操作类型与对应返回数据（A 为角度，M 为多圈，E 为错误）：

| 操作类型 | | | | | 介绍 | n | 返回数据 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b7 | b6 | b5 | b4 | b3 | | | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 |
| 0 | 0 | 0 | 0 | 0 | 获取角度 | 4 | A0 | A1 | A2 | | | | | |
| 1 | 0 | 0 | 0 | 1 | 获取多圈 | 4 | M0 | M1 | M2 | | | | | |
| 0 | 0 | 0 | 1 | 1 | 获取所有 | 9 | A0 | A1 | A2 | 17 | M0 | M1 | M2 | E |
| 1 | 1 | 0 | 0 | 0 | 重置角度(3) | 4 | A0 | A1 | A2 | | | | | |
| 0 | 1 | 1 | 0 | 0 | 重置多圈(4) | 4 | A0 | A1 | A2 | | | | | |

(1) An、Mn 为左对齐，即若 A 为 17 位数据，则 A2 的高 7 位为 0

(2) 当操作类型未出现于 B（2 ~ n）的表中，会触发通信错误，其返回数据与获取角度的返回数据相同

(3) 重置角度需连续发送 10 次方能生效

(4) 重置多圈需连续发送 10 次方能生效

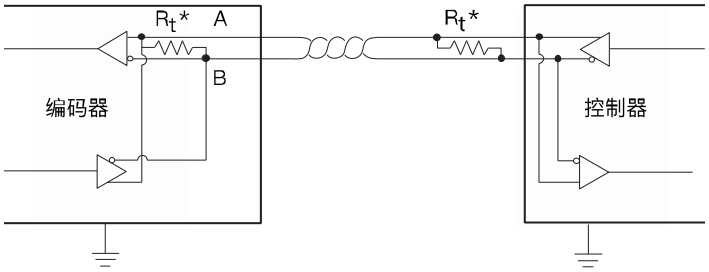
E，错误位（参考状态位章节，T485 中仅提示错误异常）：

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|
| 电池中断 | 电池低压 | 0 | 0 | 0 | 0 | 0 | 0 |

与 LED 相关的指示，请参考状态位章节。

# BUS(高速总线)协议接口

BUS 电气连接图：




该接口为二线制，其主要是差分的 A、B 相，两根线的终端都需要来并接终端电阻，编码器端的终端电阻已经集成进了编码器内部，用户需在控制器侧端接终端电阻。

该协议电平基于 485 电平逻辑，数据基于 UART 格式，工作频率为 2.5Mbps。该接口接收 1Byte 的操作请求，根据请求数据返回对应的编码器数据，并在数据尾加上 CRC–8（$x^8+1$）以做校验。

版本选择：

| 单圈位数 | 多圈位数 | 型号 |
|---|---|---|
| XX | 无 | XX–D |
| | 16 | XXM–D |

协议配置：

| 字符长度 | 奇偶校验 | 停止位 | 流控制 | 字节顺序 |
|---|---|---|---|---|
| 8 bit | 无 | 1 | 无 | LSB 优先 |

操作请求数据：

| 数据位 | b7 | b6 ~ b5 | b4 ~ b0 |
|---|---|---|---|
| 数据内容 | 奇校验 | 操作类型 | 设备地址 |

返回数据：

| 字节 | B0 | B(1 ~ n) | B(n + 1) |
|---|---|---|---|
| 数据内容 | 操作请求 | 返回数据 | CRC |

B(1 ~ n)，操作类型与对应返回数据（M 为多圈，A 为角度，C 为设置计数，S 为状态）：

| 操作类型 | | 介绍 | 型号版本 | | n | 返回数据 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| b6 | b5 | | 单圈位数 | 多圈位数 | | B1 | B2 | B3 | B4 | B5 | B6 |
| 0 | 0 | 获取信息 | ≤16 | 无 | 3 | S | A0 | A1 | | | |
| | | | ≤16 | 16 | 5 | S | A0 | A1 | M0 | M1 | |
| | | | >16 | 无 | 4 | S | A0 | A1 | A2 | | |
| | | | >16 | 16 | 6 | S | A0 | A1 | A2 | M0 | M1 |
| 0 | 1 | 设置零位 | | | 2 | S | C | | | | |
| 1 | 0 | 设置地址 | | | 2 | S | C | | | | |

| 操作类型 | | 介绍 | n | 返回数据 | | |
|---|---|---|---|---|---|---|
| b6 | b5 | | | B1 | B2 | B3 |
| 0 | 0 | 获取信息 | 3 | S | A0 | A1 |
| 0 | 1 | 设置零位 | 2 | S | C | |
| 1 | 0 | 设置地址 | 2 | S | C | |

状态位的具体描述，请参见后文状态位章节。

注：

1. 当操作类型未出现于 B（1 ~ n）的表中，如 0b11，将没有任何返回响应
2. C 为当即编码器设置时返回的连续计数，当返回 10 时，表示将会立即执行设置（该过程最大耗时约 50ms，在此期间编码器不会响应任何命令）
3. 出厂默认地址为 0x1F，即 0b11111

**设置零位操作：**

为了保证设置零位不被误操作，需要交替连续发送操作类型 00、01、00、01、00、01、.....、00、01 共十组（每个指令发送完后必须等待编码器回复完毕后才能再发下个指令），方能设置成功，可根据 01 指令返回的 C 值来判断还需发送的次数。

注：

1. 当 01 的上一条指令不是 00 时，不满足序列起始要求，C 值为 0
2. 当 01 的上上一条指令不是 01 时，不满足序列要求，C 值为 1，且从此开始计数

**设置 ID 操作：**

为了保证设置 ID 不被误操作，需要使用一定序列的地址值来确定进入设置 ID 功能后，再进行 ID 配置，如下发送为设置 ID 中的地址值（为上次返回 C 值的 2 倍）：

| 地址值 | X | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | Y |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 无 | 无 | 无 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

注：

1. 其中 X 值为任意值，Y 值为实际要配置的地址值
2. 序列中的前三组，均无任何响应返回，此举为了防止总线工作状态下误触发，由于该指令权限超越于 ID 限制，有可能会引起误发，导致总线集体响应的情况
3. 当有其它指令插入时，下一个设置指令返回的 C 值为 1，且从此开始计数
4. 当发送的地址值与序列不匹配时，返回的 C 值为 1，且从此开始计数

**总线设备：**

总线上所有设备都需要做到当收到的 ID 不是自己的设备 ID 时，需要进入一定时间的“睡眠”，即对总线上任何指令都不予响应，这样才能防止应答涟漪的出现。

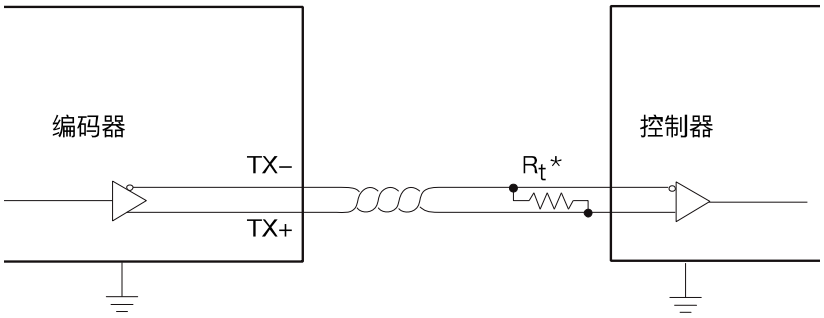
而为了方便扩展，该睡眠时间为 $T_{SUSPEND}$。，其计算方法如下表：

| 单圈位数 | 多圈位数 | 总线睡眠字节数 $B_{SUSPEND}$ | 总线睡眠时间 $T_{SUSPEND}$（us） |
|---|---|---|---|
| XX | YY | ceil(XX/8) + YY/8 + 4 | $B_{SUSPEND}$ * (1 + 8 + 1) / 2.5 |

例：16M1-D 型号的 $T_{SUSPEND}$ 为(ceil(16 / 8) + 8 / 8 + 4) * (1 + 8 + 1) / 2.5 = 28us

## PERIOD 周期发送协议接口

PERIOD 接口电气图：




该接口为二线制，其主要是差分的 A、B 相，两根线的终端都需要来并接终端电阻，用户需在控制器侧端接终端电阻

该协议基于 RS422 协议，唯一区别是其主动以 1ms 的间隔通过 TX 向外发送数据，不会对 RX 线上的数据进行响应，编码器内部会周期性触发指令**“d”(0x64)**以发送对应指令的数据，请参考 RS485/RS422 协议接口。

## 状态位

在 SSI/BISS-C/RS485/RS422 协议里，其使用的状态位是一致的，当有警示或错误发生时，输出中的警告位或错误位置位，然后可以通过状态位明确了解当前的置位原因。

错误/警告位于各接口中的位置：

| | 错误位 | 警告位 |
|---|---|---|
| **SSI** | b7 | b6 |
| **BISS-C** | b13 | b12 |
| **RS485/RS422** | b7 | b6 |
| **BUS** | b7 | b6 |
| **PERIOD** | b7 | b6 |

状态位：

| **位置** | 仅存在于电池多圈版本中 | | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|
| | b5 | b4 | | | | |
| **描述** | 电池中断 | 电池低压 | 磁场过强 | 磁场过弱 | 温度超出范围 | 速度过高 |
| **LED 闪烁** | 有 | 有 | 无 | 无 | 无 | 无 |

当警告位为 1 时，数据仍有效，此时 LED 状态灯为黄色，但是状态位中的部分参数已接近极限值，可通过状态位查看；当错误位为 1 时，数据已无效，此时 LED 状态灯为红色，可通过状态位查看具体情况；正常工作时，LED 状态灯为绿色。

LED 闪烁会以 1s 为周期亮、灭来提示用户相应的错误/警告问题的发生。

电池相关状态位：

| | b5 | b4 |
|---|---|---|
| | 电池中断 | 电池低压 |
| **警告位为 1** | – | 电池电压低于 2.9V |
| **错误位为 1** | 编码器断电期间电池断开或电压过低，导致多圈计数中断，从而多圈数据不可信，出现该错误则多圈清零 | 电池电压低于 2.7V |
| **解决** | 检查电池安装、电压，重新上电编码器则清零 | 替换编码器电池 |

# 附录

## CRC-8 表($x^8+x^7+x^4+x^2+x^1+1$)

```
//poly = x^8+x^7+x^4+x^2+x^1+1
uint8_t crcTable [256] = {
    0x00, 0x97, 0xB9, 0x2E, 0xE5, 0x72, 0x5C, 0xCB, 0x5D, 0xCA, 0xE4, 0x73, 0xB8, 0x2F, 0x01, 0x96,0xBA, 0x2D, 0x03, 0x94,
0x5F, 0xC8, 0xE6, 0x71, 0xE7, 0x70, 0x5E, 0xC9, 0x02, 0x95, 0xBB, 0x2C,0xE3, 0x74, 0x5A, 0xCD, 0x06, 0x91, 0xBF, 0x28, 0xBE,
0x29, 0x07, 0x90, 0x5B, 0xCC, 0xE2, 0x75,0x59, 0xCE, 0xE0, 0x77, 0xBC, 0x2B, 0x05, 0x92, 0x04, 0x93, 0xBD, 0x2A, 0xE1, 0x76,
0x58, 0xCF,0x51, 0xC6, 0xE8, 0x7F, 0xB4, 0x23, 0x0D, 0x9A, 0x0C, 0x9B, 0xB5, 0x22, 0xE9, 0x7E, 0x50, 0xC7,0xEB, 0x7C, 0x52,
0xC5, 0x0E, 0x99, 0xB7, 0x20, 0xB6, 0x21, 0x0F, 0x98, 0x53, 0xC4, 0xEA, 0x7D,0xB2, 0x25, 0x0B, 0x9C, 0x57, 0xC0, 0xEE, 0x79,
0xEF, 0x78, 0x56, 0xC1, 0x0A, 0x9D, 0xB3, 0x24,0x08, 0x9F, 0xB1, 0x26, 0xED, 0x7A, 0x54, 0xC3, 0x55, 0xC2, 0xEC, 0x7B, 0xB0,
0x27, 0x09, 0x9E,0xA2, 0x35, 0x1B, 0x8C, 0x47, 0xD0, 0xFE, 0x69, 0xFF, 0x68, 0x46, 0xD1, 0x1A, 0x8D, 0xA3, 0x34,0x18, 0x8F,
0xA1, 0x36, 0xFD, 0x6A, 0x44, 0xD3, 0x45, 0xD2, 0xFC, 0x6B, 0xA0, 0x37, 0x19, 0x8E,0x41, 0xD6, 0xF8, 0x6F, 0xA4, 0x33, 0x1D,
0x8A, 0x1C, 0x8B, 0xA5, 0x32, 0xF9, 0x6E, 0x40, 0xD7,0xFB, 0x6C, 0x42, 0xD5, 0x1E, 0x89, 0xA7, 0x30, 0xA6, 0x31, 0x1F, 0x88,
0x43, 0xD4, 0xFA, 0x6D,0xF3, 0x64, 0x4A, 0xDD, 0x16, 0x81, 0xAF, 0x38, 0xAE, 0x39, 0x17, 0x80, 0x4B, 0xDC, 0xF2, 0x65,0x49,
0xDE, 0xF0, 0x67, 0xAC, 0x3B, 0x15, 0x82, 0x14, 0x83, 0xAD, 0x3A, 0xF1, 0x66, 0x48, 0xDF,0x10, 0x87, 0xA9, 0x3E, 0xF5, 0x62,
0x4C, 0xDB, 0x4D, 0xDA, 0xF4, 0x63, 0xA8, 0x3F, 0x11, 0x86, 0xAA, 0x3D, 0x13, 0x84, 0x4F, 0xD8, 0xF6, 0x61, 0xF7, 0x60, 0x4E,
0xD9, 0x12, 0x85, 0xAB, 0x3C
};

uint8_t calcCRC(uint8_t * buffer, uint8_t length){
    uint8_t temp = *buffer++;
    while(--length){
        temp = *buffer++ ^ crcTable[temp];
    }

    return crcTable[temp];
}
```

## CRC-8 计算($x^8+1$)

```
//poly = x^8+1
//该多项式查表值与查表结果相同

uint8_t calcCRC(uint8_t * buffer, uint8_t length){
    uint8_t temp = *buffer++;
    while(--length){
        temp = *buffer++ ^ temp;
    }

    return temp;
}
```

# CRC-6 计算

```
#define DATA_TOTAL_BIT_LENGTH 47


//poly = x^6+x^1+1
uint8_t tableCRC6[64] = {
0x00, 0x03, 0x06, 0x05, 0x0C, 0x0F, 0x0A, 0x09, 0x18, 0x1B, 0x1E, 0x1D, 0x14, 0x17, 0x12, 0x11, 0x30, 0x33, 0x36, 0x35, 0x3C, 0x3F, 0x3A, 0x39, 0x28, 0x2B, 0x2E, 0x2D, 0x24, 0x27, 0x22, 0x21, 0x23,
0x20, 0x25, 0x26, 0x2F, 0x2C, 0x29, 0x2A, 0x3B, 0x38, 0x3D, 0x3E, 0x37, 0x34, 0x31, 0x32, 0x13, 0x10, 0x15, 0x16, 0x1F, 0x1C, 0x19, 0x1A, 0x0B, 0x08, 0x0D, 0x0E, 0x07, 0x04, 0x01, 0x02
};

uint8_t calcBissCCRC(uint8_t buffer[]){
	#define CRC_BIT_LENGTH 6
	#define DATA_CRC_MASK ((1 << CRC_BIT_LENGTH) - 1)
	#define DATA_WITHOUT_CRC_BIT_LENGTH (DATA_TOTAL_BIT_LENGTH - CRC_BIT_LENGTH)
	#define TOP_BYTE_BITLENGTH (DATA_WITHOUT_CRC_BIT_LENGTH % CRC_BIT_LENGTH)
	#if TOP_BYTE_BITLENGTH == 0
		#undef TOP_BYTE_BITLENGTH
		#define TOP_BYTE_BITLENGTH CRC_BIT_LENGTH
	#endif

	uint32_t firstWord = __REV(*(uint32_t *) buffer);
	#if DATA_WITHOUT_CRC_BIT_LENGTH > 32
	uint32_t secondWord = __REV(*(uint32_t *) (buffer + 4));
	#endif

	uint8_t crc = tableCRC6[firstWord >> (32 - TOP_BYTE_BITLENGTH)];

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 1)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (firstWord >> (32 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 2)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (firstWord >> (32 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 3)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (firstWord >> (32 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 4)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (firstWord >> (32 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 5)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		#if 32 - CURRENT_CRC_BIT_LENGTH >= 0
			crc = tableCRC6[crc ^ (firstWord >> (32 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
		#elif 32 - CURRENT_CRC_BIT_LENGTH > -CRC_BIT_LENGTH
			crc = tableCRC6[crc ^ (((firstWord << -(32 - CURRENT_CRC_BIT_LENGTH)) & DATA_CRC_MASK) | (secondWord >> (64 - CURRENT_CRC_BIT_LENGTH)))];
		#else
			crc = tableCRC6[crc ^ (secondWord >> (64 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
		#endif
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 6)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (secondWord >> (64 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 7)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (secondWord >> (64 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 8)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (secondWord >> (64 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#undef CURRENT_CRC_BIT_LENGTH
	#define CURRENT_CRC_BIT_LENGTH (TOP_BYTE_BITLENGTH + CRC_BIT_LENGTH * 9)
	#if DATA_WITHOUT_CRC_BIT_LENGTH - CURRENT_CRC_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ (secondWord >> (64 - CURRENT_CRC_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	#if 32 - DATA_TOTAL_BIT_LENGTH >= 0
		crc = tableCRC6[crc ^ DATA_CRC_MASK ^ (firstWord >> (32 - DATA_TOTAL_BIT_LENGTH) & DATA_CRC_MASK)];
	#elif 32 - CURRENT_CRC_BIT_LENGTH > -CRC_BIT_LENGTH
		crc = tableCRC6[crc ^ DATA_CRC_MASK ^ (((firstWord << -(32 - DATA_TOTAL_BIT_LENGTH)) & DATA_CRC_MASK) | (secondWord >> (64 - DATA_TOTAL_BIT_LENGTH)))];
	#else
		crc = tableCRC6[crc ^ DATA_CRC_MASK ^ (secondWord >> (64 - DATA_TOTAL_BIT_LENGTH) & DATA_CRC_MASK)];
	#endif

	return crc;
}
```

## 使用说明：

本程序可应用于 ARM 系列 MCU，可为您通过编译器生成最快速度的 CRC-6 校验
仅需将 DATA_TOTAL_BIT_LENGTH 修改为对应型号的数值

如：17M 则为 47 ， 16 则为 30

## 使用注意：

在调用时，对 buffer 使用了 32 位读指令，要求 buffer 为 4 字节对齐（如果未对齐，有的内核版本不支持；即使支持未对齐读数，内核也消耗了多余的拼接时间）。

而对于 BISS-C 的接收来说，接收的第一个字节将会是带有 ACK 的占位字节，位置数据是从第二个字节开始的，所以我们要从位置数据开始的地址开始计算 CRC，并要求该地址为 4 字节对齐模式，才能达到快速读取数据并计算 CRC 的初衷。

如调用案例：

```
struct{
        uint8_t notUsedForAlignment[3];                 //仅为地址对齐
        uint8_t placeholder;                            //BISS-C 的第一个固定占位字节 0x82
        uint8_t buffer[8] __attribute__ ((aligned(4)));     //4 节字对齐的 buffer，方便后面快速 CRC
} receiveBuffer;

//配置 SPI 与 DMA
//使用&receiveBuffer.placeholder 作为接收地址
//……

//计算 CRC
//使用已经 4 字对齐的 receiveBuffer.buffer 来计算
uint8_t crc = calcBissCCRC(receiveBuffer.buffer);

//crc 结果等于 0，表示校验通过
if ( crc != 0 ){
        //crc 校验失败
}
```

# 版本历史

| 时间 | 版本 | 修改内容 |
| --- | --- | --- |
| 2023/09/28 | V0.1 | 初始化版本 |
| 2024/01/05 | V0.2 | 添加更多尺寸 |
| 2024/05/01 | V0.3 | 添加标准图纸<br>添加线束颜色 |

KingKong.tech
金 钢 科 技

北京金钢科技有限公司

北京市昌平区科技园区永安路 26 号 712 室

Website: https://kingkong.tech

Email: contact@kingkong.tech

Tel: 010-80111669