

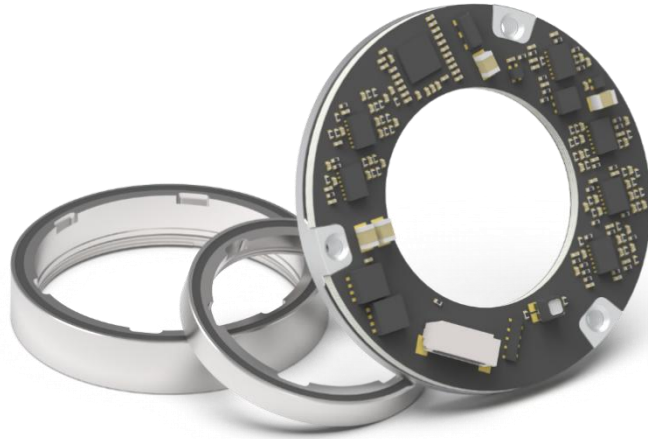


KingKong.tech
金 钢 科 技

DPT 磁双编码器数据手册 v0.6



DPT 磁双编码器



DPT 磁双编码器为机器人一体化关节专用，使用金钢科技图案式磁技术，在有限的体积内可以达到两个轴的测量，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。

- 双 24 位绝对值输出
- 双 $\pm 0.01^\circ$ 绝对定位精度
- 超紧凑（径向单边 7 mm）
- 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
- 一对一标定
- 多种输出接口
- 各种环境干扰抗性

型号

DPT-10-15-25-A-24-24-R-N-A-M

内转子内径

具体请查看[尺寸](#)

xx — 螺纹款

xxB — 粘胶款

外转子内径

具体请查看[尺寸](#)

xx — 螺纹款

xxB — 粘胶款

定子外径

具体请查看[尺寸](#)

xx — 正装款

xxH — 正反装款

输出种类

A — 绝对值

内编码器输出参数

24 — 24 位

外编码器输出参数

24 — 24 位

波特率

M — 默认 2.5M

V — 5M

工作温度

A — -40 ~ 85 °C

B — -40 ~ 105 °C

C — -40 ~ 125 °C

输入电压

N — 5V

输出接口

R — [RS485](#)

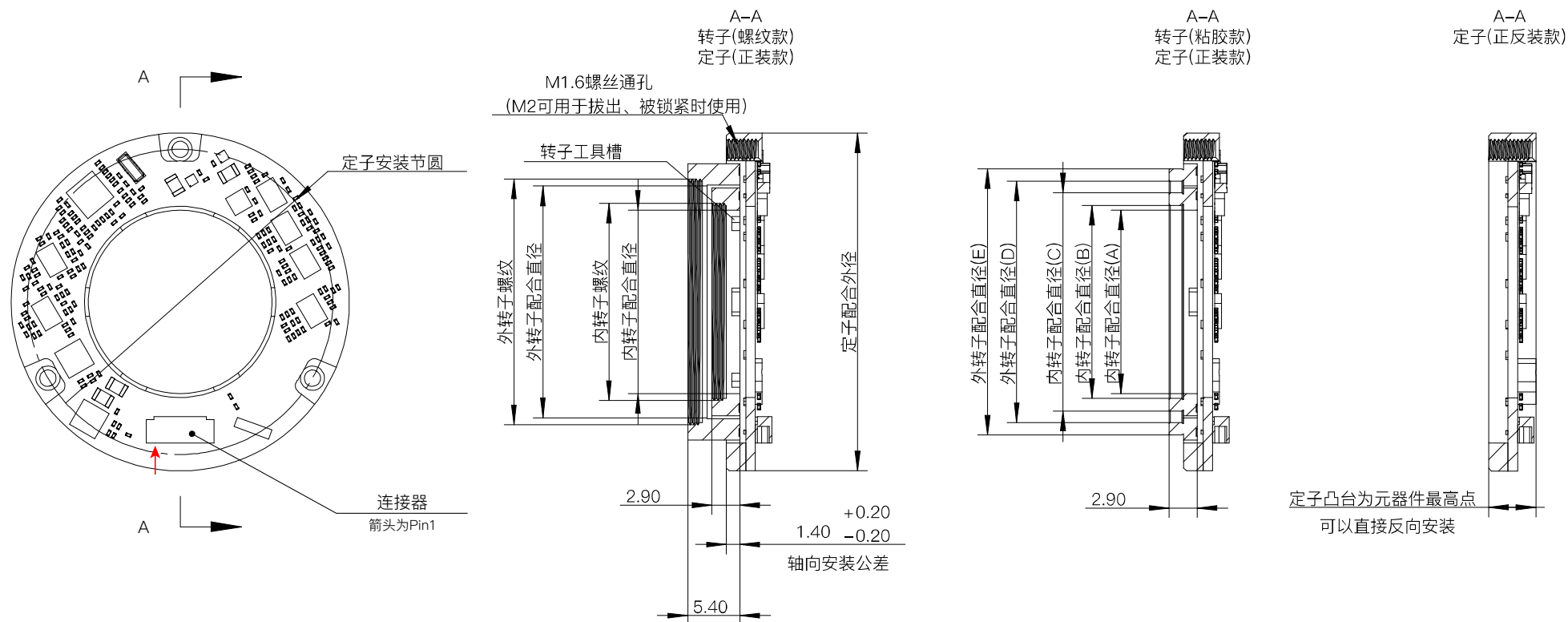
B — [BISS-C](#)

尺寸

系列	尺寸	转子款式	定子款式	内转子螺纹	内转子高度	内转子配合直径 (H7、h7)			外转子螺纹	外转子高度	外转子配合直径 (H7)		定子安装节圆	定子配合 外径(h7)	定子可以直接反装 ⁽²⁾	连接器 型号
						A	B	C			D	E				
10	DPT-10-15-25	螺纹款	正装款	M10x0.4mm	2.9	9			M15x0.4mm	5.4	14		21.6	25	-	X0800WRS-06HF-LPV01
	DPT-10B-15B-25	粘胶款	正装款	-		9	10	12.6	-	2.9	15	17.6			-	
	DPT-10-15-25H	螺纹款	正反装款	M10x0.4mm		9			M15x0.4mm	5.4	14				✓	
	DPT-10B-15B-25H	粘胶款	正反装款	-		9	10	12.6	-	2.9	15	17.6			✓	
15	DPT-15-20-30	螺纹款	正装款	M15x0.4mm		14			M20x0.4mm	5.4	19		26.6	30	-	
	DPT-15B-20B-30	粘胶款	正装款	-		14	15	17.6	-	2.9	20	22.6			-	
	DPT-15-20-30H	螺纹款	正反装款	M15x0.4mm		14			M20x0.4mm	5.4	19				✓	
	DPT-15B-20B-30H	粘胶款	正反装款	-		14	15	17.6	-	2.9	20	22.6			✓	
20	DPT-20-25-35	螺纹款	正装款	M20x0.4mm		19			M25x0.4mm	5.4	24		31.6	35	-	
	DPT-20B-25B-35	粘胶款	正装款	-		19	20	22.6	-	2.9	25	27.6			-	
	DPT-20-25-35H	螺纹款	正反装款	M20x0.4mm		19			M25x0.4mm	5.4	24				✓	
	DPT-20B-25B-35H	粘胶款	正反装款	-		19	20	22.6	-	2.9	25	27.6			✓	
25	DPT-25-30-40	螺纹款	正装款	M25x0.4mm		24			M30x0.4mm	5.4	29		36.6	40	-	
	DPT-25B-30B-40	粘胶款	正装款	-		24	25	27.6	-	2.9	30	32.6			-	
	DPT-25-30-40H	螺纹款	正反装款	M25x0.4mm		24			M30x0.4mm	5.4	29				✓	
	DPT-25B-30B-40H	粘胶款	正反装款	-		24	25	27.6	-	2.9	30	32.6			✓	
30	DPT-30-35-45	螺纹款	正装款	M30x0.4mm		29			M35x0.4mm	5.4	34		41.6	45	-	
	DPT-30B-35B-45	粘胶款	正装款	-		29	30	32.6	-	2.9	35	37.6			-	
	DPT-30-35-45H	螺纹款	正反装款	M30x0.4mm		29			M35x0.4mm	5.4	34				✓	
	DPT-30B-35B-45H	粘胶款	正反装款	-		29	30	32.6	-	2.9	35	37.6			✓	

1. 下载 3D 模型: <https://kingkong.tech/encoder/DPT>
2. 正装款如果需要反装的话, 需要与定子上的凸台进行结构上的对齐
3. 粘胶款配合直径任选其一即可

图纸



上图“红色箭头”代表引脚 1，详见第 5 页

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《DPT 设计安装建议》

安装完成后必须进行标定，相关标定软件、标定方法及图纸请联系金钢公司

电气连接

连接器

	线对板连接器
型号	X0800WRS-06HF-LPV01
种类	线对板型
连接线	3PxAWG32 铁氟龙双绞线

引脚

引脚	颜色	R	B
		RS485	BISS-C
1	橙	—	SLO —
2	黄	—	SLO +
3	绿	B	MA —
4	蓝	A	MA +
5	黑	0V (GND)	
6	红	+5V	

参数规格

系统参数

安装方式	轴向中空
分辨率	24bit
精度	$\pm 0.01^\circ$ (与装配有关)

电气参数

电源	4.5 ~ 5.5 V
启动时间	15 ms
连接方式	线对板连接器
电流	≈ 140 mA
静电保护	HBM, max. ± 2 kV CDM, max. ± 1 kV

机械参数

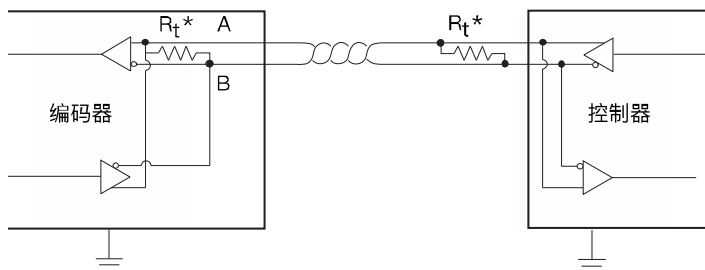
转子托架	不锈钢
定子托架	铝合金

环境参数

工作温度	$-40 \sim 85^\circ\text{C}$ / $-40 \sim 105^\circ\text{C}$ / $-40 \sim 125^\circ\text{C}$
------	---

RS485 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。

RS485 底层协议为 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：

代号	M	V
波特率 (Mbps)	2.5	5

时序图：



(1) 编码器会在红外点处锁存当前角度数据

指令与数据:

指令	介绍		N	返回数据 (N 个字节)							
				B0	B1	B2	B3	B4	B5	B6	B7
0x29	设置	内零位 ⁽¹⁾	2	C	CRC						
0x30	设置	外零位 ⁽²⁾	2	C	CRC						
0x31	获取	内角度	4	A0	A1	A2	CRC				
0x32	获取	外角度	4	B0	B1	B2	CRC				
0x33	获取	内角度 外角度	7	A0	A1	A2	B0	B1	B2	CRC	
0x41	获取	内角度 状态信息	5	A0	A1	A2	S	CRC			
0x42	获取	外角度 状态信息	5	B0	B1	B2	S	CRC			
0x43	获取	内角度 外角度 状态信息	8	A0	A1	A2	B0	B1	B2	S	CRC

上表中, 字母对应数据为:

A	B	C	S	CRC
内编码器角度	外编码器角度	清零读数值	状态	CRC 校验

(1) 设置零位需连续间隔发送命令 0x31、发送命令 0x29 共 10 次, 方能设置成功; 当返回计数值为 10 时, 触发设置零位

(2) 设置零位需连续间隔发送命令 0x32、发送命令 0x30 共 10 次, 方能设置成功; 当返回计数值为 10 时, 触发设置零位

(3) 数字越大, 字节顺序越高

(4) 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\(\$x^8+x^7+x^4+x^2+x^1+1\$ \)](#))

示例:

如发送 0x33, 得到 uint8_t Buffer[7];

那使用时:

uint32_t angleInner = Buffer[2] << 16 | buffer[1] << 8 | buffer[0];

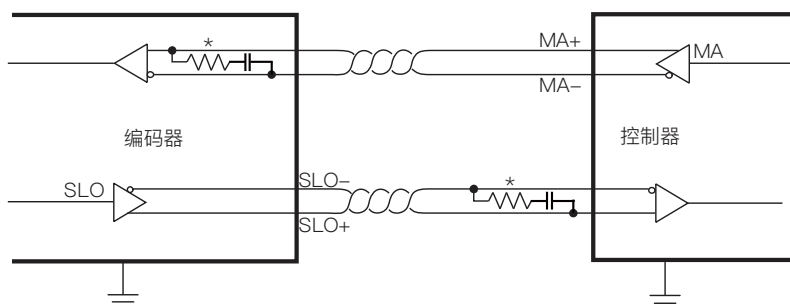
uint32_t anngleOuter = Buffer[5] << 16 | buffer[4] << 8 | buffer[3];

float angleInnerFloat = angleInner / (float)(1 << 24) * 360;

float angleOuterFloat = angleOuter / (float)(1 << 24) * 360;

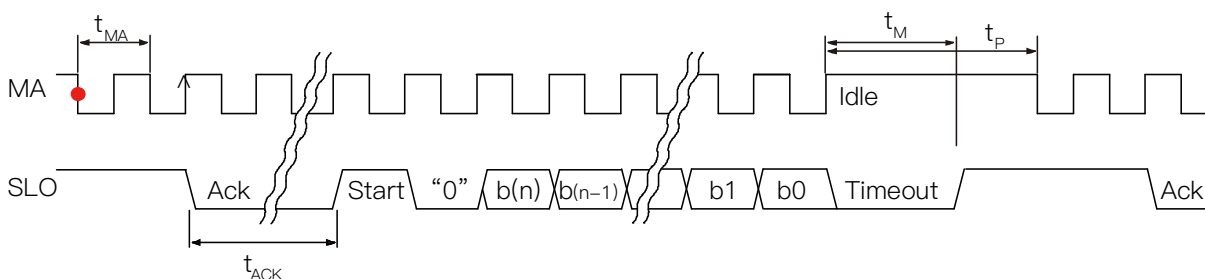
BISS-C 协议接口

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:



(1) 编码器会在红外点处锁存住当前角度数据

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		

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	b55 : b32	b31 : b8	b7	b6	b5 : b0
长度	24 bits	24 bits	1 bit	1 bit	6 bits
数据	内圈角度	外圈角度	错误位 ⁽¹⁾	警告位 ⁽²⁾	CRC ⁽³⁾

(1) 错误位为低电平有效。

(2) 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。

(3) CRC 多项式为 $x^6 + x^1 + 1$ （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。

关于错误位与警告位的具体描述，请参见后文[状态位](#)章节。

状态位

错误位，警告位及状态位信息用于表征当前编码器的工作状态，其中错误位与警告位在所有接口中的表示功能是一致的。

错误位、警告位、状态位：

位置	b7	b6	b5	b4	b3	b2	b1	b0
功能	错误指示		状态位					
描述	错误位	—	外圈角度错误	内圈角度错误	—	—	—	—

(1) RS485 协议置位为“1”报错

(2) BISS-C 协议置位为“0”报错，并且无具体状态位信息

伴随着错误位的信息的不同 LED 会发生颜色变化。当错误位置位时，数据已无效，此时 LED 状态灯为红色，具体情况可通过状态位查看；正常工作时，LED 状态灯为绿色。

附录

CRC-8 表($x^8 + x^7 + x^4 + x^2 + x + 1$)

//poly = $x^8 + x^7 + x^4 + x^2 + x + 1$

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-6 计算

```
#define DATA_TOTAL_BIT_LENGTH 47

//poly = x6+x1+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 calcBissCRC(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：24-24 为 56

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 字节对齐的 buffer，方便后面快速 CRC
} receiveBuffer;

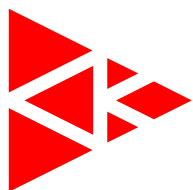
//配置 SPI 与 DMA
//使用&receiveBuffer.placeholder 作为接收地址
//.....

//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
2023/09/01	V0.1	初始化版本
2024/03/18	V0.2	更新状态位 添加 BISS-C 协议
2024/05/01	V0.3	添加标准图纸 添加线束颜色
2024/12/08	V0.4	添加转子粘胶款 添加定子反装款 修改连接器为左 1 为 Pin1
2024/12/22	V0.5	添加角度锁存位置指示
2026/03/31	V0.6	添加转速说明 添加波特率类型 完善各协议说明



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双营西路 82 号院 2 号楼 3 层 301 室

Website: <https://kingkong.tech>

Email: contact@kingkong.tech

Tel: 010-80111669