

DPT Magnetic Dual Encoder DataSheet v0.6



DPT Magnetic Dual Encoder



DPT dual magnetic encoder is specifically designed for integrated robot actuators. It adopts KingKong Technology's patterned magnetic technology, enabling dual-axis measurement within a compact structure. Each axis can achieve a resolution and accuracy approaching that of optical encoders, while maintaining strong resistance to environmental interference.

The DPT is based on magnetolectric sensing technology and features proprietary interference shielding. Multiple high-precision magnetic field sensors are integrated inside the encoder to detect variations in the magnetic field of the rotor magnet ring. Combined with KingKong Technology's precision calibration process, each unit is individually calibrated at the factory with unique magnetic field data to ensure optimal measurement accuracy.

The unique rotor-stator tolerance-matching mounting method simplifies installation while ensuring measurement precision.

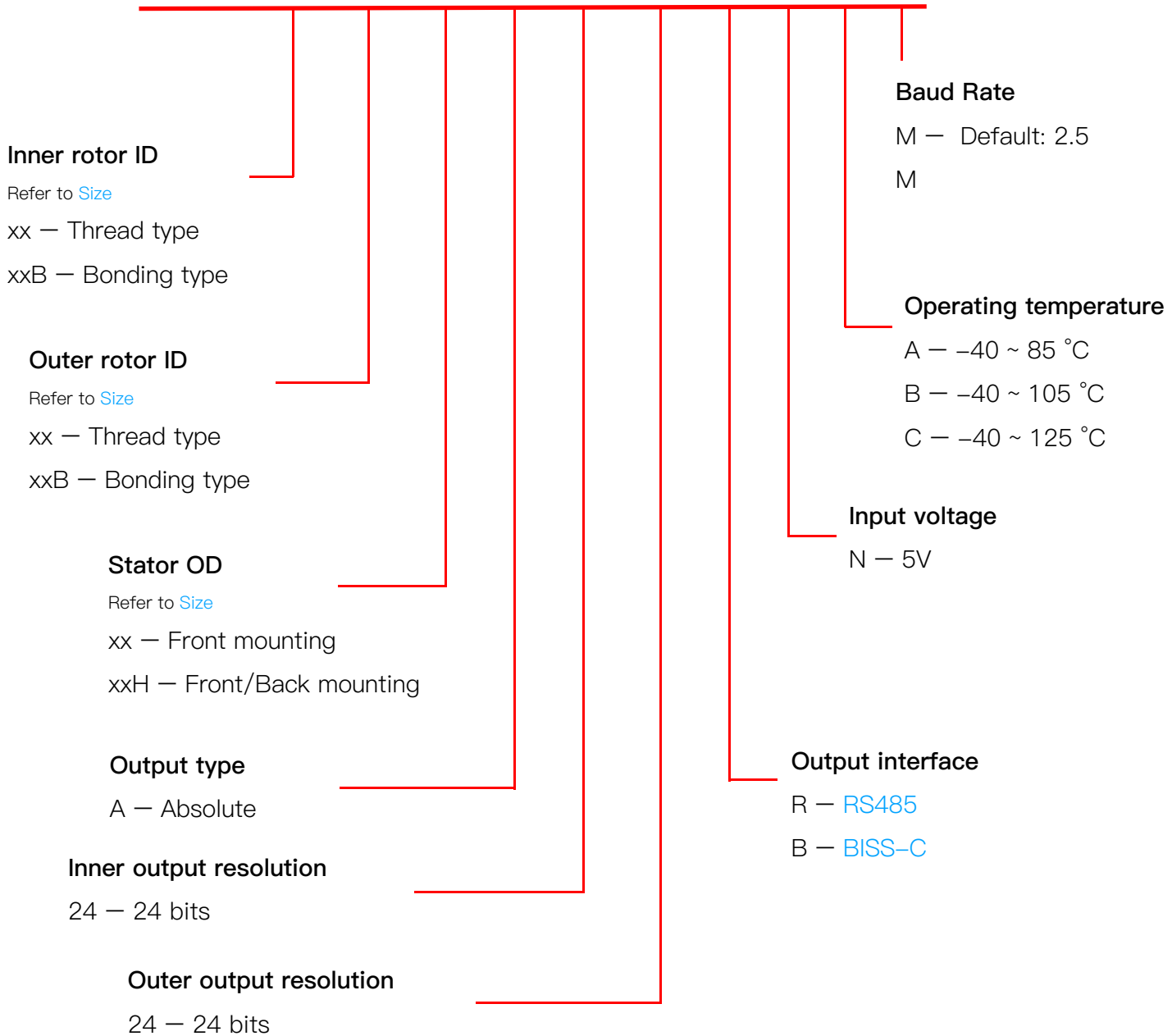
The separated magnetolectric design allows the encoder to effectively resist external disturbances such as vibration, dust, and oil contamination. Even under ultra-high-speed operation, accuracy and service life remain unaffected.

The ultra-thin design with a hollow shaft makes it easy to integrate into a wide range of applications.

- Dual 24-bit absolute output
- Dual $\pm 0.01^\circ$ accuracy
- Ultra-compact (Ring width 7mm)
- Models for every 5mm on diameter
- Stator and rotor tolerance fit mounting
- Magnetic interference shielding
- Hollow structure
- Permissible speed 8,000 rpm
- Unique data calibration
- Various output interfaces
- Resistance to various environmental disturbances

Models

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



Size

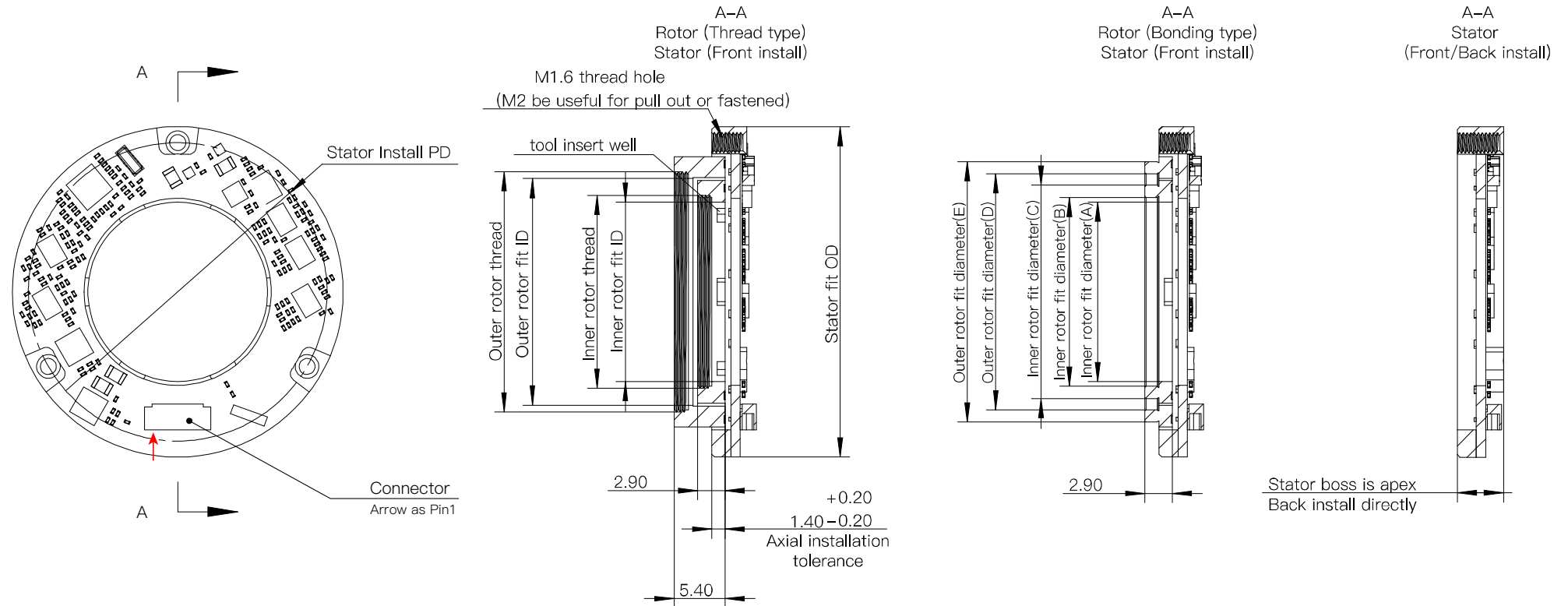
Series	Size	Rotor type	Stator type	Inner rotor thread	Inner rotor height	Inner rotor ID (H7) ⁽³⁾			Outer rotor thread	Outer rotor height	Outer rotor OD(H7) ⁽³⁾		Stator mounting PD	Stator fit OD(h7)	Stator back install directly ⁽²⁾	Connector model
						A	B	C			D	E				
10-15-25	DPT-10-15-25	Thread	Front	M10x0.4mm	2.9	9			M15x0.4mm	5.4	14		21.6	25	-	X0800WRS-06HF-LPV01
	DPT-10B-15B-	Bonding	Front	-	2.9	9	10	12.6	-	2.9	15	17.6			-	
	DPT-10-15-25H	Thread	Front/Back	M10x0.4mm	2.9	9			M15x0.4mm	5.4	14				✓	
	DPT-10B-15B-	Bonding	Front/Back	-	2.9	9	10	12.6	-	2.9	15	17.6			✓	
15-20-30	DPT-15-20-30	Thread	Front	M15x0.4mm	2.9	14			M20x0.4mm	5.4	19		26.6	30	-	X0800WRS-06HF-LPV01
	DPT-15B-20B-	Bonding	Front	-	2.9	14	15	17.6	-	2.9	20	22.6			-	
	DPT-15-20-30H	Thread	Front/Back	M15x0.4mm	2.9	14			M20x0.4mm	5.4	19				✓	
	DPT-15B-20B-	Bonding	Front/Back	-	2.9	14	15	17.6	-	2.9	20	22.6			✓	
20-25-35	DPT-20-25-35	Thread	Front	M20x0.4mm	2.9	19			M25x0.4mm	5.4	24		31.6	35	-	X0800WRS-06HF-LPV01
	DPT-20B-25B-	Bonding	Front	-	2.9	19	20	22.6	-	2.9	25	27.6			-	
	DPT-20-25-35H	Thread	Front/Back	M20x0.4mm	2.9	19			M25x0.4mm	5.4	24				✓	
	DPT-20B-25B-	Bonding	Front/Back	-	2.9	19	20	22.6	-	2.9	25	27.6			✓	
25-30-40	DPT-25-30-40	Thread	Front	M25x0.4mm	2.9	24			M30x0.4mm	5.4	29		36.6	40	-	X0800WRS-06HF-LPV01
	DPT-25B-30B-	Bonding	Front	-	2.9	24	25	27.6	-	2.9	30	32.6			-	
	DPT-25-30-40H	Thread	Front/Back	M25x0.4mm	2.9	24			M30x0.4mm	5.4	29				✓	
	DPT-25B-30B-	Bonding	Front/Back	-	2.9	24	25	27.6	-	2.9	30	32.6			✓	
30-35-45	DPT-30-35-45	Thread	Front	M30x0.4mm	2.9	29			M35x0.4mm	5.4	34		41.6	45	-	X0800WRS-06HF-LPV01
	DPT-30B-35B-	Bonding	Front	-	2.9	29	30	32.6	-	2.9	35	37.6			-	
	DPT-30-35-45H	Thread	Front/Back	M30x0.4mm	2.9	29			M35x0.4mm	5.4	34				✓	
	DPT-30B-35B-	Bonding	Front/Back	-	2.9	29	30	32.6	-	2.9	35	37.6			✓	

1. Download 3D models: <https://kingkong.tech/encoder/DPT>

2. If front install needs to back install, must align boss to shield manually.

3. ID of bonding type, You can choose either one.

Drawings



The "red arrow" in the figure indicates Pin 1. For details, please refer to Page 5.

For stator mounting, either M1.6 screws can be passed through the M2 threaded holes, or M2 screws can be used with the M2 threads for fastening.

For rotor mounting, a dedicated tool should be used for tightening. For details, please refer to the DPT Design and Mounting Guidelines.

Calibration must be performed after mounting. For calibration software, procedures, and drawings, please contact KingKong Technology.

Electrical connections

Connectors

	Wire-to-Board Connector
Model	X0800WRS-06HF-LPV01
Type	Wire to board
Wire	3PxAWG32P TFE Twisted Pair Cable

Pin Definition

Pin	Color	R	B
		RS485	BISS-C
1	Orange	-	SLO -
2	Yellow	-	SLO +
3	Green	B	MA -
4	Blue	A	MA +
5	Black	0V (GND)	
6	Red	+5V	

Specifications

System Parameters

Mounting	Axial hollow
Resolution	24bit
Accuracy	$\pm 0.01^\circ$ (assembly dependent)

Electrical Parameters

Supply Voltage	4.5 ~ 5.5 V
Startup Time	15 ms
Connection	Wire-to-Board Connector
Current Consumption	≈ 140 mA
ESD Protection	HBM, max. ± 2 kV CDM, max. ± 1 kV

Mechanical Parameters

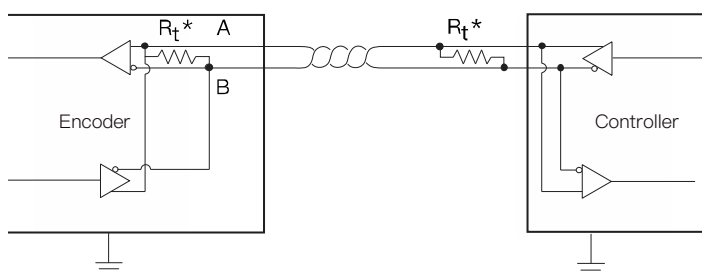
Rotor Carrier	Stainless steel
Stator Carrier	Aluminum Alloy

Environmental Parameters

Operating temperature	-40 ~ 85 °C -40 ~ 105 °C -40 ~ 125 °C
-----------------------	---

RS485 interface

RS485 Electrical connection diagram:



The interface uses a two-wire configuration with differential signals (A and B).

Termination resistors are required at both ends of the bus. The encoder side already has an integrated termination resistor, so the user should implement termination on the controller side (A and B), or apply an appropriate impedance matching scheme.

The underlying protocol of RS485 is based on UART. Since there is no clock line in this protocol, both the encoder and the controller must operate at the same predefined baud rate and data format to ensure reliable data communication.

Protocol Configuration:

Data Bits	Parity	Stop Bits	Flow Control	Byte Order
8	None	1	None	LSB first

Supported Baud Rate (if not noted in addition, B is default and recommended):

Code	M	V
Baud Rate (Mbps)	2.5	5

Timing diagram:



(1) The encoder latches the current angle at the red dot.

Commands and data:

Com- mand	Description		N	Data (N Bytes)								
				B0	B1	B2	B3	B4	B5	B6	B7	
0x29	set	Inner Zero ⁽¹⁾	2	C	CRC							
0x30	set	Outer Zero ⁽²⁾	2	C	CRC							
0x31	get	Inner Angle	4	A0	A1	A2	CRC					
0x32	get	Outer Angle	4	B0	B1	B2	CRC					
0x33	get	Inner Angle Outer Angle	7	A0	A1	A2	B0	B1	B2	CRC		
0x41	get	Inner Angle Status	5	A0	A1	A2	S	CRC				
0x42	get	Outer Angle Status	5	B0	B1	B2	S	CRC				
0x43	get	Inner Angle Outer Angle Status	8	A0	A1	A2	B0	B1	B2	S	CRC	

Above letters in table represent:

A	B	C	S	CRC
Inner Angle	Outer Angle	Zero-setting Value	Status	CRC Checksum

- (1) To set the inner zero position, send command 0x31 and 0x29 alternately at intervals for a total of 10 times. The zero-setting will be successful only after this sequence is completed. When the returned count reaches 10, the zero position is triggered..
- (2) To set the outer zero position, send command 0x32 and 0x30 alternately at intervals for a total of 10 times. The zero-setting will be successful only after this sequence is completed. When the returned count reaches 10, the zero position is triggered.
- (3) The larger the number, the higher the byte order..
- (4) CRC byte (CRC polynomial: $x^8 + x^7 + x^4 + x^2 + x^1 + 1$; calculation method is provided in the appendix). [CRC-8 table](#) ($x^8 + x^7 + x^4 + x^2 + x^1 + 1$)

Example code:

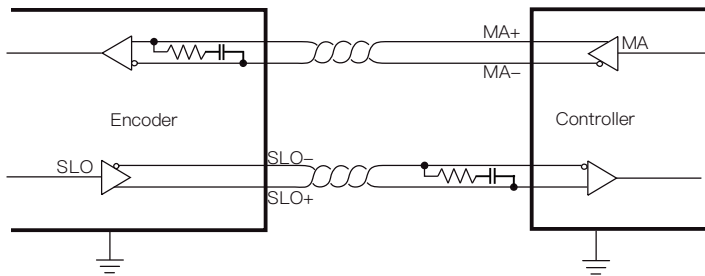
If used 0x33, get uint8_t Buffer[7];

In use:

```
uint32_t angleInner = Buffer[2] << 16 | buffer[1] << 8 | buffer[0];
uint32_t angleOuter = Buffer[5] << 16 | buffer[4] << 8 | buffer[3];
float angleInnerFloat = angleInner / (float)(1 << 24) * 360;
float angleOuterFloat = angleOuter / (float)(1 << 24) * 360;
```

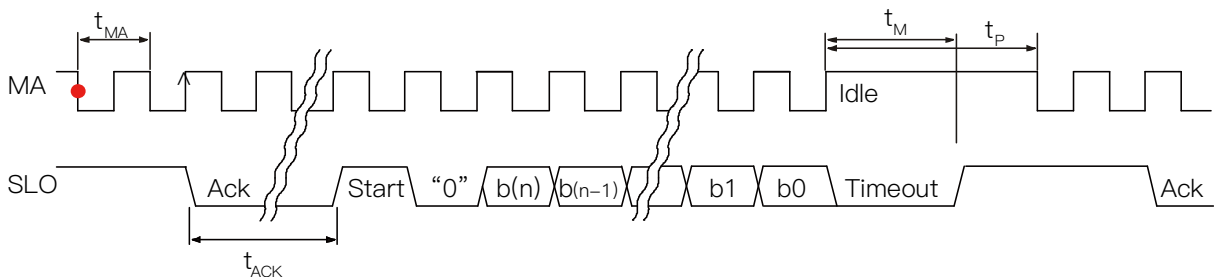
BISS-C interface

BiSS-C electrical connection diagram:



This interface uses a four-wire configuration, consisting of MA+ / MA- and SLO+ / SLO-. The termination resistor for MA is already integrated inside the encoder, and the user needs to add a termination resistor at the SLO side of the controller or implement another impedance matching scheme.

Timing diagram:



(1) The angle is latched at the red point.

The protocol uses the MA line to synchronize and acquire the clock sequence. When the MA line is idle, it remains at a high level. On the first falling edge, the system latches the current data, and communication begins.

The encoder sets the SLO line to low on the second rising edge of MA. Starting with the MSB following the start bit ("0"), data is output onto the SLO line on each rising edge of MA. On the controller side, data on the SLO line is read on each falling edge of MA, continuing until the LSB is received.

Timing parameters:

Parameters	Symbol	Min	Typical	Max
Clock period	t_{MA}	400 ns		14 μ s
Clock frequency	f	120 kHz		2.5 MHz
ACK length	t_{ACK}		5 bits	
Transmission timeout	t_M		10 μ s	

Pause time	t_P	20 μ s		
------------	-------	------------	--	--

After the transmission is completed and the timeout period t_M has elapsed, the SLO line is driven high. The MA signal shall remain high until the next read operation is permitted, i.e., after the pause time t_P . The t_{CL} shall be less than t_M . During any read operation, if the duration exceeds t_m , the read operation will be terminated.

Data format:

Bit	b55 : b32	b31 : b8	b7	b6	b5 : b0
Length	24 bits	24 bits	1bit	1bit	6 bits
Data	Inner Angle	Outer Angle	Error ⁽¹⁾	Warning ⁽²⁾	CRC ⁽³⁾

(1) The error bit is active low.

(2) The warning bit is active low. A value of 1 indicates that no error or warning is triggered; a value of 0 indicates that at least one error or warning condition is present.

(3) The CRC polynomial is $x^6 + x^1 + 1$ (i.e. 0x43), According to the BISS-C protocol requirements, the calculated CRC value will be inverted before send. Appendix [CRC-6 calculations](#) provides directly portable calculation codes for reference.

For a detailed description of the status bit, refer to the [Status](#) section.

Status Bits

The error bit, warning bit, and status bits are used to indicate the current operating state of the encoder. Among them, the functions of the error bit and warning bit are consistent across all interfaces.

Status bits:

Position	b7	b6	b5	b4	b3	b2	b1	b0
Function	Error		Status bits					
Description	Error bit	–	Outer ring angle error	Inner ring angle error	–	–	–	–

(1) In the RS485 protocol, a value of “1” indicates an error.

(2) In the BiSS-C protocol, a value of “0” indicates an error, and no detailed status bit information is provided.

The LED changes color according to the error bit status. When the error bit is set, the data is invalid, and the LED indicator turns red; the specific condition can be identified via the status bits. During normal operation, the LED indicator remains green.

Appendix

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

```
//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 calculation

```

#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)];
#elseif 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)];
#elseif 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;
}

```

Instruction:

This program can be applied to ARM series MCU, and can generate the fastest CRC-6 check through the compiler, just modify DATA_TOTAL_BIT_LENGTH to the value of the corresponding model.

For example: 17M model to 47, 16 model to 30

Caution:

When called this function, a 32-bit read command is used for the buffer, requiring the buffer to be 4-byte aligned (some core versions don't support unaligned data read; Even with support for unaligned reads, the kernel consumes more concatenation time).

For the reception of BISS-C, the first byte received will be a placeholder byte with ACK, and the position data starts from the second byte, to read data and calculate CRC quickly, it is necessary to calculate CRC from the address where the position data starts, and require the address to be 4-byte alignment data.

For example:

```
struct{
    uint8_t notUsedForAlignment[3];           //Only align data
    uint8_t placeholder;                    //The first placeholder byte of BISS-C is 0x82
    uint8_t buffer[8] __attribute__((aligned(4))); //Buffer of 4 bytes data align, for fast CRC
} receiveBuffer;

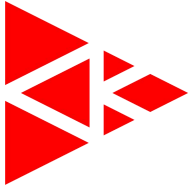
//Configure SPI and DMA
//Use &receiveBuffer.placeholder as the receiving address
//.....

//CRC calculate
// Calculated using the receiveBuffer.buffer which is already 4-byte aligned
uint8_t crc = calcBissCCRC(receiveBuffer.buffer);

//The CRC result is equal to 0, indicating that the verification is passed
if ( crc != 0 ){
    //crc validation faile
}
```

Revision history

Date	Version	Description
2023/09/01	V0.1	Initial release
2024/03/18	V0.2	Updated status bits Added BiSS-C protocol
2024/05/01	V0.3	Added standard drawings Added cable color definition
2024/12/08	V0.4	Added bonded rotor version Added inverted stator mounting option revised connector definition (Pin 1 on the left)
2024/12/22	V0.5	Added angle latch position indication
2026/06/01	V0.6	Added speed description Added baud rate options Refined protocol descriptions



KingKong.tech

Beijing KingKong Technology Co., Ltd.

Address: B2, No. 82 Shuangying West Road, Changping District, Beijing, China

Website: <https://kingkong.tech>

Email: contact@kingkong.tech

Tel: +86010-80111669