HPS-FT060

High performance digital 6D force torque sensor

Features

- Fast, accurate force and torque measuring on three spatial axes of XYZ
  - High resolution
  - High SNR (signal to noise ratio)
  - Dustproof and waterproof design (IP65)
  - High overload range
  - Compact design for easy integration

- Fully integrated compact structure, including:
  - High sensitivity strain gauge and structure
  - High precision ADC
  - High performance embedded microprocessor
  - Advanced embedded data processing, filtering and decoupling algorithms
  - RS-485 bus interface
  - 78(L) x 60(W) x 28.5(H) mm, 235g

Applications

- Multi-axis force/torque measurement
- Industrial robot polishing and grinding
- 3C precision assembly
- Industrial robots “Teach-In”
- Force feedback automatic control
- Collision detection
- Industrial precision cutting
- Measurement in sports medicine

Description

The HPS-FT060 is a high-performance digital six-dimensional force torque sensor that accurately measures forces and torques on three spatial axes of XYZ. The sensor structure is highly rugged and durable. The sensor’s factor-of-safety can be as high as several times during operation.

Compared to traditional strain gage structures, the HPS-FT060 achieves higher signal-to-noise ratio and sensitivity. The built-in temperature compensation algorithm greatly reduces the temperature drift introduced by the temperature changes. HPS-FT060 can measure six force/torque components of Fx, Fy, Fz, Mx, My and Mz with high precision and in real-time. The internal integrated professional compensation algorithm ensures high linearity of measurement results and extremely low cross-axis crosstalk.

Ordering Information

HPS- | F | T | 0 | 6 | 0 | - | |

- Hypersen product designator
- **FT**: 6D force torque sensor family
- Sensor diameter (Unit: mm)
- **Communication Interface:**
  None: RS-485 (Built-in amplifier)
  E: EtherCAT (External amplifier)
  L: LAN (External amplifier)

Overview

1.1 Technical specification

Table 1. Technical specification

Parameter	Values	Unit
Size	78（L）x 60（W）x 28.5（H）	mm
Weight*1	235	g
Power supply	8 ~ 30	V
Power consumption	0.5	W
Storage temperature	-40 ~ 85	℃
Operating temperature	-10 ~ 55 *2	℃
Nominal capacity	±600 (Fxy)	N
	±1000 (Fz)	N
	±15 (Mxy)	Nm
	±15 (Mz)	Nm
Precision	0.2 (Fxy)	N
	0.4 (Fz)	N
	0.002 (Mxy)	Nm
	0.002 (Mz)	Nm
Non-linearity	<0.25	%
Creep	<2	%
Crosstalk	<2 (F.S.)	%
Zero drift	0.3	%/Day
Hysteresis	0.1 (F.S.)	%
Overload*2	500	%
Output data rate*3	1300 (Max.)	Hz
Output data	Fx, Fy, Fz, Mx, My, Mz	-
Wiring definition	Red: VCC, Black: GND, White: 485A, Green: 485B	-
Communication*4	RS-485, 115200bps (Default)	-
Cable length	200	cm
Protection	IP65	-

Note：*1 Not include cable.

*2 The overload will not damage the sensor, but it may cause changes in sensor parameters.

*3 To achieve the maximum 1300Hz output data rate, it needs to set the baud rate to 1500000bps, and to send Command#5 to set the maximum sensor measurement frequency.

*4 HPS-FT060 supports seven baud rates: 9600bps, 115200bps, 128000bps, 256000bps, 460800bps, 600000bps, 750000bps, 921600bps and 1500000bps. The default baud rate is 115200bps. The Command#6 can be used to modify the communication baud rate. The modified value will be saved to the internal non-volatile memory and reloaded with each power up.

1.2 Dimensions and wiring definition

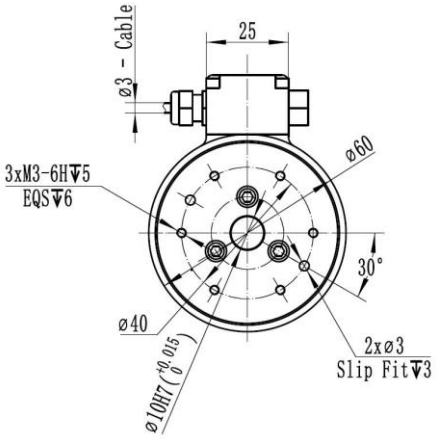

Figure 1. Top view

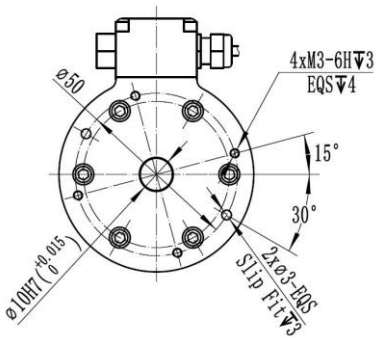

Figure 2. Bottom view

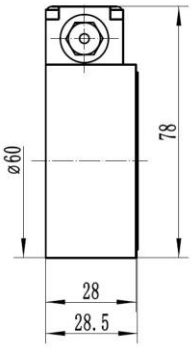

Figure 3. Side view

Table 2. HPS-FT060 wiring definition

Cable color	Signal name	Signal type	Description
Red	Positive	VCC	To be connected to DC +8V~+30V
Black	Negative	GND	Ground
White	RS-485A	Digital	To be connected to RS-485 transceiver A (+)
Green	RS-485B	Digital	To be connected to RS-485 transceiver B (-)

Control interface

2.1 RS-485 bus communication protocol

HPS-FT060 uses RS-485 bus interface to communicate with the host. The default communication baud rate is 115200bps. Users can modify the baud rate by sending Command#6. HPS-FT060 supports seven baud rates: 9600bps, 115200bps, 128000bps, 256000bps, 460800bps, 600000bps, 750000bps, 921600bps and 1500000bps.

Table 3. RS-485 parameters

Default baud rate	115200bps
Date bit	8bits
Parity	Even parity
Stop bit	1bit
CRC verify	CRC16-CCITT

2.1.1 Data frame format

Table 4. Sensor data frame format

Frame header		Data length	Device address	Reserved byte	Command	Data	CRC LSB	CRC MSB	Frame end	
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6~N	ByteN+1	ByteN+2	ByteN+3	ByteN+4
0xF6	0x6F	1 byte	0x00 (Default)	0x00	1 byte	0~255 byte	1 byte	1 byte	0x6F	0xF6

2.1.2 Command list

Table 5. Sensor command list

Command	Command byte	Description
Read device ID number	0x01	Read device ID No.: 0x46FE:
Continuous measurement	0x02	Continuously output measuring data
Stop continuous measuring	0x03	Stop continuous measuring
Single measurement	0x04	Single output measuring data
Set measuring frequency	0x05	Sensor supports five grades (0~4) measuring frequencies, users can change it according to different applications
Set baud rate	0x06	Set baud rates to: 9600bps, 115200bps, 128000bps, 256000bps, 460800bps, 614400bps or 921600bps
Reset user settings	0x07	Reset user setting saved in flash memory, such as: measuring frequency, baud rate, device address
Rest factory settings	0x08	Reset factory settings
Save user settings	0x09	Save user settings to flash memory
Read version information	0x0A	Read sensor version information
Reset zero-point	0x0B	Reset sensor's zero point
Reset device address	0x0D	Reset device address to default value "0x00"
Set device address	0x0E	Set device address

Command#1 Read device ID number

This command is used to read device ID number (0x46FE)

Table 6. Read device ID number command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x01	0xF0	0x3E	0x6F	0xF6

Returned data:

Table 7. Returned data of read device ID number command

Byte number	Data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x05	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x01	Command byte
6	0xFE	Device ID No. LSB
7	0x46	Device ID No. MSB
8	0xF0	CRC16-CCITT LSB
9	0x3E	CRC16-CCITT MSB
10	0x6F	Frame end first byte
11	0xF6	Frame end second byte

Command#2 Single measurement

This command will activate sensor and take a single measurement

Table 8. Single measurement command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x04	0x18	0x8C	0x6F	0xF6

Command#3 Continuous measurement

This command will activate sensor and take a continuous measurement

Table 9. Continuous measurement command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x02	0xDE	0xEC	0x6F	0xF6

Returned data of single/continuous measurement commands:

Table 10. Returned data of single /continuous measurement command

Byte number	Data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x1B	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x04/0x02	Command byte
6~9	……	Fx integer complement value (LSB to MSB), value / 1000 = Fx (Unit：N)
10~13	……	Fy integer complement value (LSB to MSB), value / 1000 = Fy (Unit：N)
14~17	……	Fz integer complement value (LSB to MSB), value / 1000 = Fz (Unit：N)
18~21	……	Mx integer complement value (LSB to MSB), value / 1000 = Mx (Unit：Nm)
22~25	……	My integer complement value (LSB to MSB), value / 1000 = My (Unit：Nm)
26~29	……	Mz integer complement value (LSB to MSB), value / 1000 = Mz (Unit：Nm)
30	……	CRC16-CCITT LSB
31	……	CRC16-CCITT MSB
32	0x6F	Frame end first byte
33	0xF6	Frame end second byte

Data decoding example:

Following is a frame of continuous measurement data:

F6 6F 1B 00 00 02 16 FF FF FF 01 FA FF FF EF 02 00 00 06 00 00 00 0A 00 00 00 0F 00 00 00 6F 58 6F F6

Decoding:

0xF6: Header
0x6F: Header
0x1B: Data length
0x00: Device address
0x00: Reserved byte
0x02: Continuous measurement command byte
0x16 0xFF 0xFF 0xFF: Fx data 0xFFFFFF16 → -234, Fx = -0.234N
0x01 0xFA 0xFF 0xFF: Fy data 0xFFFFFA01 → -1535, Fy = -1.535N
0xEF 0x02 0x00 0x00: Fz data 0x000002EF → 751, Fz = 0.751N
0x06 0x00 0x00 0x00: Mx data 0x00000006 → 6, Mx = 0.006Nm
0x0A 0x00 0x00 0x00: My data 0x0000000A → 10, My = 0.010Nm
0x0F 0x00 0x00 0x00 Mz data 0x0000000F → 15, Mz = 0.015Nm
0x6F: CRC16-CCITT LSB
0x58: CRC16-CCITT MSB
0x6F: Frame end
0xF6: Frame end

Note: After the sensor is powered on, it needs to be warmed up for a period of time to stabilize the output. It is recommended to operate the sensor at the highest measurement frequency for 10~20 minutes, so that the sensor is thermally balanced, and then send a reset zero-point command before use.

Command#4 Stop continuous measurement

This command can be used to stop the continuous measurement

Table 11. Stop continuous measurement command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x03	0xFF	0xFC	0x6F	0xF6

Returned data：

Table 12. Returned data of stopping continuous measurement

Byte number	Data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x03	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	0xB2	CRC16-CCITT LSB
8	0xC1	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Note: If the sensor is currently working under continuous measurement mode, before sending the stop measurement command it is necessary to send 50 bytes of 0x00 to occupy the RS-485 bus. After receiving those bytes, the sensor will temporarily release the RS-485 bus.

The host can send stop measurement command 200ms after sending 50 bytes of 0x00 data, then the sensor will return an ACK data packet indicating that the continuous measurement has been stopped correctly.

If the host has not sent a stop measurement command within 250ms after sending 50 bytes of 0x00 data, the sensor will automatically restart the continuous measurement. If the sensor is currently working under continuous measurement mode, it needs to be stopped before sending any other commands.

Command#5 Set measuring frequency command

This command can be used to set the continuous measurement frequency (Five grades: 0~4)

Table 13. Set measuring frequency command

Header		Data length	Device address	Reserved byte	Command	Frequency	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
0xF6	0x6F	0x04	0x00	0x00	0x05	0x00~0x04	……	……	0x6F	0xF6

Returned data:

Table 14. Returned data of set measuring frequency command

Byte number	Data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x05	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	0x14	CRC16-CCITT LSB
8	0x6B	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Command #6 Set communication baud rate

This command can be used to set the RS-485 communication baud rate. The sensor supports seven baud rates: 9600bps, 115200bps, 128000bps, 256000bps, 460800bps, 614400bps and 921600bps. The factory default baud rate is 115200bps. The command takes effect immediately, and the client needs to switch to the new baud rate accordingly, otherwise, client will receive wrong returned data.

Table15. Set communication baud rate command

Header		Data length	Device address	Reserved byte	Command	Baud rate	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte6~9	Byte10	Byte11	Byte12	Byte13
0xF6	0x6F	0x07	0x00	0x00	0x06	……	……	……	0x6F	0xF6

Table16. Values of byte 6~11 correspond to different baud rates

Baud rate	Values of byte 6~9	Values of byte 10~11
9600 bps	0x80, 0x25, 0x00, 0x00	0x45, 0x8C
115200 bps	0x00, 0xC2, 0x01, 0x00	0xED, 0x47
128000 bps	0x00, 0xF4, 0x01, 0x00	0xE8, 0x30
256000 bps	0x00, 0xE8, 0x03, 0x00	0x88, 0x60
460800 bps	0x00, 0x08, 0x07, 0x00	0x7D, 0x0C
600000 bps	0xC0, 0x27, 0x09, 0x00	0x21, 0x36
750000 bps	0xB0, 0x71, 0x0B, 0x00	0x58, 0xFE
921600 bps	0x00, 0x10, 0x0E, 0x00	0x27, 0x5C
1500000 bps	0x60, 0xE3, 0x16, 0x00	0x2D, 0x35

Returned data：

Table17. Returned data of set communication baud rate command

Byte number	data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x07	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x06	Command byte
6	……	Baud rate first byte
7	……	Baud rate second byte
8	……	Baud rate third byte
9	……	Baud rate fourth byte
10	……	CRC16-CCITT LSB
11	……	CRC16-CCITT MSB
12	0x6F	Frame end first byte
13	0xF6	Frame end second byte

Command#7 Zero-reset

This command can be used to reset the zero output of the sensor. After the sensor is installed, send this command to reset the zero output.

Table 18. Zero-reset command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x0B	0xF7	0x7D	0x6F	0xF6

Returned data：

Table 19. Returned data of zero-reset command

Byte number	data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	0x00	Device address

4	0x00	Reserved byte
5	0x0B	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	0x1B	CRC16-CCITT LSB
8	0x48	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Command#8 Modify device address

This command can be used to modify the sensor's device address. This feature allows multiple sensor devices working on a same RS-485 bus under single measurement mode.

Table 20. Modify device address command

Frame header		Data length	Device address	Reserved byte	Command	New Device address	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
0xF6	0x6F	0x04	0x00	0x00	0x0E	……	……	……	0x6F	0xF6

Returned data：

Table 21. Returned of modify device address command

Byte number	Data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	……	New device address
4	0x00	Reserved byte
5	0x0E	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	……	CRC16-CCITT LSB
8	……	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Command#9 Reset device address

This command can be used to reset the device address to the factory default value “0x00”.

Table22. Reset device address command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x0D	0x31	0x1D	0x6F	0xF6

Returned data：

Table23. Returned data of reset device address command

Byte number	data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte

2	0x04	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x0D	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	0xBD	CRC16-CCITT LSB
8	0xE2	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Command#10 Reset user setting

This command can be used to reset user saved settings from internal flash memory. After the parameter recovery is completed, note that the device address and baud rate may be different from the current settings.

Table 24. Reset user setting command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x07	0x7B	0xBC	0x6F	0xF6

Returned data：

Table 25. Returned data of resetting user setting command

Byte number	Data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x07	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	0x76	CRC16-CCITT LSB
8	0x0D	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Command#11 Reset factory settings

This command can be used to reset factory settings from internal flash memory, including: measurement frequency, communication baud rate, device address.

Table 26. Reset factory setting command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x08	0x94	0x4D	0x6F	0xF6

Returned data：

Table 27. Returned data of reset factory setting command

Byte number	data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x08	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	0x48	CRC16-CCITT LSB
8	0x1D	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Command #12 Save user setting

This command can be used to save the user settings into internal flash memory. The settings will be automatically reloaded with each power up.

Table28. Save user setting command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x09	0xB5	0x5D	0x6F	0xF6

Returned data：

Table29. Returned data of save user setting command

Byte number	data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x09	Command byte
6	0x01	ACK byte, 0x01: Succeed; 0x00: Fail
7	0x79	CRC16-CCITT LSB
8	0x2E	CRC16-CCITT MSB
9	0x6F	Frame end first byte
10	0xF6	Frame end second byte

Command#13 Read sensor version information

This command can be used to read sensor version information

Table 30. Read sensor version information command

Frame header		Data length	Device address	Reserved byte	Command	CRC LSB	CRC MSB	Frame end	
Byte 0	Byte 1	Byte2	Byte3	Byte4	Byte3	Byte4	Byte5	Byte6	Byte7
0xF6	0x6F	0x03	0x00	0x00	0x0A	0xD6	0x6D	0x6F	0xF6

Returned data：

Table31. Returned data of saving user setting command

Byte number	data	Description
0	0xF6	Header first byte
1	0x6F	Header second byte
2	0x04	Data length
3	0x00	Device address
4	0x00	Reserved byte
5	0x0A	Command byte
6	……	Firmware generated time，Year
7	……	Firmware generated time，Month
8	……	Firmware generated time，Day
9	……	Major version number
10	……	Minor version number
11	……	Modify version number
12	……	CRC16-CCITT LSB
13	……	CRC16-CCITT MSB
14	0x6F	Frame end first byte
15	0xF6	Frame end second byte

Revision history

Table32. Specification revision history

Date	Revision	Description
2018/07/08	1.0	Initial version
2018/08/28	1.1	Added 750000 bps and 1500000bps baud rates. Updated the output data rate from 300Hz to 1300Hz.

Appendix

CRC16-CCITTC-language Implementations

Implementation 1：

```
#################################################################
#include<stdio.h>
/**
Flash Space: Small
Calculation Speed: Slow
*/
/*Function Name：          crc_cal_by_bit          //Calculate CRC by byte
  Function Parameters：   unsigned char* ptr      //Pointer of the data buffer
                          unsigned char len       //data length
  Return Value：          unsigned int
  Polynomial:             CRC-CCITT 0x1021
*/
unsigned int crc_cal_by_bit(unsigned char* ptr, unsigned char len)
{
    #define CRC_CCITT   0x1021
    unsigned int crc = 0xffff;

    while(len-- != 0)
    {
        for(unsigned char i = 0x80; i != 0; i /= 2)
        {
            crc *= 2;
            if((crc&0x10000) !=0)
                crc ^= 0x11021;
            if((*ptr&i) != 0)
                crc ^= CRC_CCITT;
        }
        ptr++;
    }
    return crc;
}
#################################################################
```

Implementation 2：

```
##################################################################
#include<stdio.h>
/**
Flash Space: Medium
Calculation Speed: Medium
*/
/* Function Name：          crc_cal_by_halfbyte      //Calculate CRC by half byte
    Function Parameters：    unsigned char* ptr       // Pointer of data buffer
                             unsigned char len        //data length
    Return Value：           unsigned int
    Polynomial:              CRC-CCITT   0x1021
*/
unsigned int crc_cal_by_halfbyte(unsigned char* ptr, unsigned char len)
{
    unsigned short crc = 0xffff;

    while(len-- != 0)
    {
        unsigned char high = (unsigned char)(crc/4096);
        crc <<= 4;
        crc ^= crc_ta_4[high^(*ptr/16)];
        high = (unsigned char)(crc/4096);
        crc <<= 4;
        crc ^= crc_ta_4[high^(*ptr&0x0f)];
        ptr++;
    }
    return crc;
}

unsigned int crc_ta_4[16]={ /* CRC half byteTable */
    0x0000,0x1021,0x2042,0x3063,0x4084,0x50a5,0x60c6,0x70e7,
    0x8108,0x9129,0xa14a,0xb16b,0xc18c,0xd1ad,0xe1ce,0xf1ef,
};
##################################################################
```

Implementation 3：

```
#####################################################################
#include<stdio.h>
/**
Flash Space: Large
Calculation Speed: Fast
*/
/* Function Name：          crc_cal_by_byte          //Calculate CRC by byte
    Function Parameters：   unsigned char* ptr      //Pointer of data buffer
                            unsigned char len       //Data length
    Return Value：          unsigned int
    Polynomial:             CRC-CCITT 0x1021
*/
unsigned int crc_ta_8[256]={ /* CRCbyteTable */
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
```

```
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

unsigned int crc_cal_by_byte(unsigned char* ptr, unsigned char len)
{
    unsigned short crc = 0xffff;

    while(len-- != 0)
    {
        unsigned int high = (unsigned int)(crc/256);
        crc <<= 8;
        crc ^= crc_ta_8[high^*ptr];
        ptr++;
    }

    return crc;
}
```

##

Testing code：

##

```
void main()
{
    unsigned char sample_data[] = {0x01, 0x01, 0x01, 0x06, 0xd9, 0xfc, 0x8c, 0x02, 0x01, 0x00, 0x01};//Result should be: 0x9b94
    unsigned char data1[] = {0x63};//Result should be: 0xbd35
    unsigned char data2[] = {0x8c};//Result should be: 0xb1f4
    unsigned char data3[] = {0x7d};//Result should be: 0x4eca
    unsigned char data4[] = {0xaa, 0xbb,0xcc};//Result should be: 0x6cf6
    unsigned char data5[] = {0x00,0x00,0xaa, 0xbb, 0xcc};//Result should be: 0xb166
    unsigned short r1 = 0, r2=0, r3=0, r4=0, r5=0, r_sample_data;

    //Implementation 1
    r1 = crc_cal_by_byte(data1, 1);
    r2 = crc_cal_by_byte(data2, 1);
    r3 = crc_cal_by_byte(data3, 1);
    r4 = crc_cal_by_byte(data4, 3);
    r5 = crc_cal_by_byte(data5, 5);
    r_sample_data = crc_cal_by_byte(sample_data, 11);
    printf("Implementation_1: r1= %x, r2=%x, r3=%x, r4=%x, r5=%x, r_sample_data=%x\n", r1, r2, r3, r4, r5,r_sample_data);
    r1=r2=r3=r4=r5=0;
```

```
    //Implementation 2
    r1 = crc_cal_by_bit(data1, 1);
    r2 = crc_cal_by_bit(data2, 1);
    r3 = crc_cal_by_bit(data3, 1);
    r4 = crc_cal_by_bit(data4, 3);
    r5 = crc_cal_by_bit(data5, 5);
    r_sample_data = crc_cal_by_bit(sample_data, 11);
    printf("Implementation_2: r1= %x, r2=%x, r3=%x, r4=%x, r5=%x, r_sample_data=%x\n", r1, r2, r3, r4,
r5,r_sample_data);
    r1=r2=r3=r4=r5=0;

    //Implementation 3
    r1 = crc_cal_by_halfbyte(data1, 1);
    r2 = crc_cal_by_halfbyte(data2, 1);
    r3 = crc_cal_by_halfbyte(data3, 1);
    r4 = crc_cal_by_halfbyte(data4, 3);
    r5 = crc_cal_by_halfbyte(data5, 5);
    r_sample_data = crc_cal_by_halfbyte(sample_data, 11);
    printf("Implementation_3: r1= %x, r2=%x, r3=%x, r4=%x, r5=%x, r_sample_data=%x\n", r1, r2, r3, r4,
r5,r_sample_data);
    r1=r2=r3=r4=r5=0;
}
################################################################
```


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.