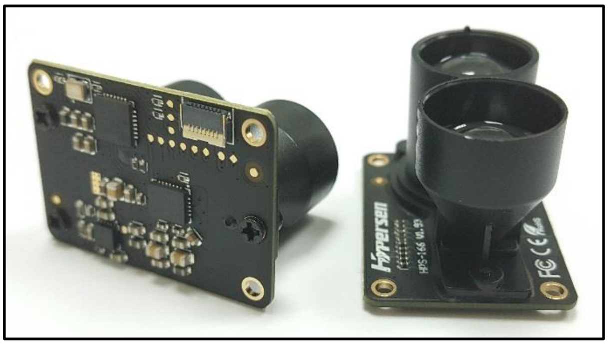
HPS-166

High Performance Time-of-Flight (ToF) Sensor



Features

- Fast, accurate distance ranging
  - Measures absolute range up to 32m (white target) with accuracy indication
  - Output data rate up to 54Hz
  - Measuring result is not sensitive to the target color and reflectivity
  - Embedded electrical & optical cross-talk compensation
  - -10℃~+55℃ temperature compensation
  - Ambient light compensation enables accurate measurement in high infrared ambient light levels
- Fully integrated miniature module
  - 850nm infrared LED emitter
  - Emitter driver
  - Integrated optimally-designed emitting & receiving optical lens
  - Ranging sensor with advanced embedded micro controller
  - Advanced embedded data processing & filtering algorithm
  - 34(W) x 24(H) x 22(D) mm, 7g
- Eye safety
  - Compliant with latest Photobiological Safety of Lamps and Lamp Systems Standard IEC62471(Class 0), CE, FCC, RoHS

Applications

- Drones (collision avoidance, soft-landing)
- Robotics & AGV (obstacle detection)
- Industrial location and proximity sensing
- Security and surveillance
- 1D gesture recognition

Description

The HPS-166 is a new generation Time-of-Flight (ToF) infrared-ranging module with optimally-designed emitting & receiving optical lens, suitable for precise, long-distance measurements. It provides accurate distance measurement whatever the target color and reflectivity unlike conventional technologies. HPS-166 can measure absolute distances up to 32m on a white target, setting a new benchmark in ranging performance levels, opening the door to various new applications.

The HPS-166 integrates a high-power 850nm infrared LED and a high-sensitivity PD (photodiode) coupled with internal physical infrared filters, enables longer ranging distance and higher immunity to ambient light.

Advanced embedded data processing & filtering algorithm realizes extremely stable and real-time measurement outputs.

CE FC RoHS

Overview

1.1 Technical specification

Table 1. Technical specification

Parameter	Values	Unit
Size	34(L) x 24(W) x 22(H) *	mm
Weight	7 *	g
Power supply	4 ~ 6	V
Maximum power consumption	1.3	W
Quiescent power consumption	0.1	W
Storage temperature	-40 ~ 85	℃
Operating temperature	-10 ~ 55	℃
Infrared LED emitter	850	nm
Emitting angle	±1.8	°
Maximum measuring distance	32 **	m
Minimum measuring distance	0.08	m
Output data rate	3 ~ 54	Hz
Output data	Distance, accuracy, signal strength, ambient light level, temperature	-
Connector	0.5mm-pitch, 8-pin, FPC connector, top and bottom double contacts	-
Interface	TTL UART, 115200bps, 8 data bits, no parity, 1 stop bit	

Note: * Without lens cover

** Tested on 90% reflectance white target

1.2 Mechanical drawing & device pinout

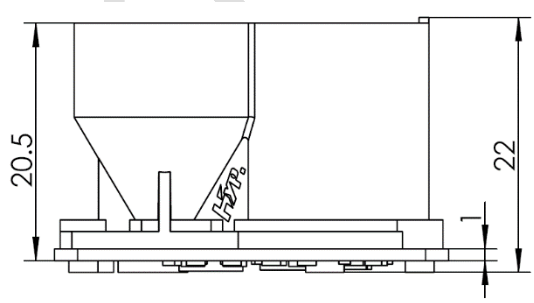


Figure 1. Front view of HPS-166

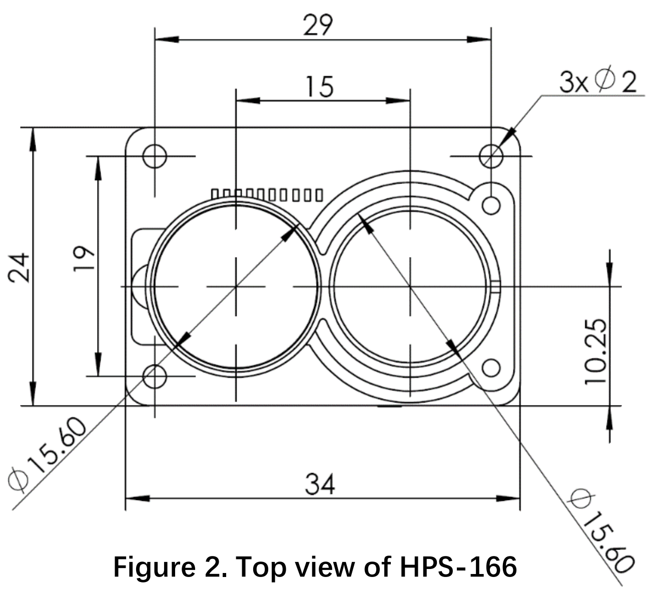


Figure 2. Top view of HPS-166

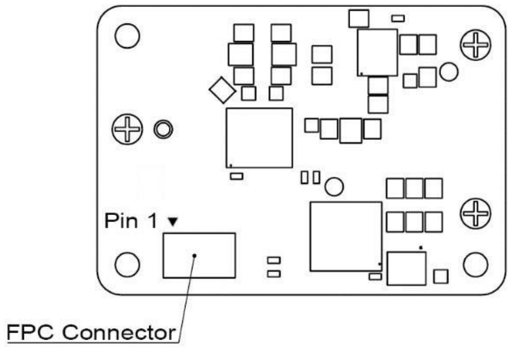


Figure 3. Bottom view of HPS-166

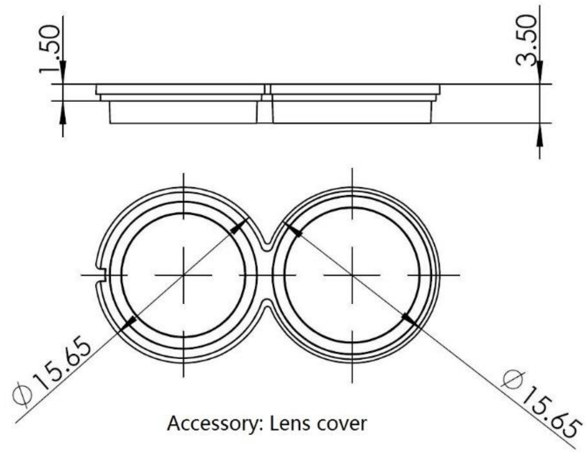


Figure 4. Front and bottom view of lens cover

Table 2. HPS-166 pin description

Pin number	Signal name	Signal type	Description
1	TXD	Digital output	UART TTL output
2	RXD	Digital input	UART TTL input
3	RST	Digital input	Reset signal input, active low
4	INT	Digital output	Interrupt signal output (pulse width: 100us)
5	GND	GND	Ground
6	GND	GND	Ground
7	VDD	Power	Supply, to be connected to main supply (typical +5V)
8	VDD	Power	Supply, to be connected to main supply (typical +5V)

All pins are compliant with IEC61000-4-2 ESD Immunity Test values presented in Table 3

Table 3. ESD performances

Parameter	Conditions
Air Discharge	+/- 8kV
Direct Contact	+/- 4kV
Indirect Contact HCP	+/- 4kV
Indirect Contact VCP	+/- 4kV

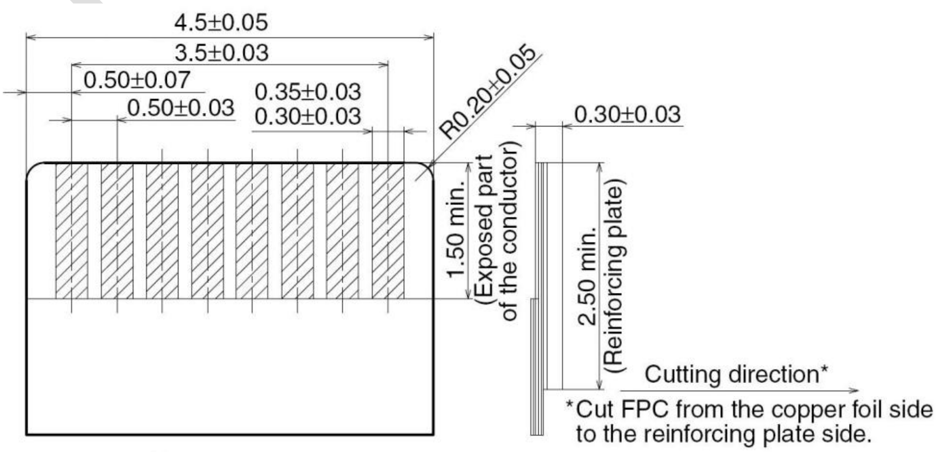


Figure 5. Recommend FPC/FFC dimensions

1.3 Absolute maximum ratings

Table 4. HPS-166 pin description

Parameter	Min.	Typ.	Max.	Unit
VDD	-0.3	-	6.5	V
RXD, RST	-0.3	-	5.6	V

Note: Stresses above those listed in Table 4. may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of the specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

1.4 Recommended operating conditions

Table 5. HPS-166 pin description

Parameter	Min.	Typ.	Max.	Unit
VDD	4	5	6	V
RXD, RST	2.8	3.3	3.6	V

1.5 Application schematic

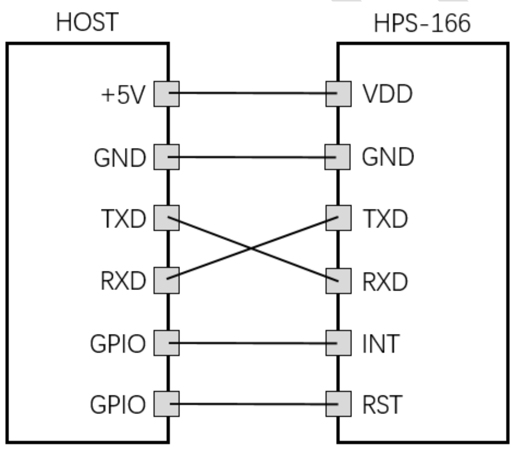


Figure. 6 Application schematic of the HPS-166

Control interface

2.1 TTL UART serial interface

HPS-166 has an TTL UART interface and can communicate with any host that has an TTL UART interface. The logical level corresponds 3.3V powered logics.

Table 6. UART properties

Baud rate	115200bps
Start bit	1bit
Data bit	8bits
Parity bit	0bit
Stop bit	1bit

2.2 Communication protocols

After the sensor is powered up, system automatically performs the initialization procedures and the serial interface will output “Hypersen” if the initialization succeeded. A start byte “0x0A” is used to indicate the start of each command and returned data frame. Each HPS-166 has its universally unique identifier (UUID), which can be read out by sending a command from the host.

Command #1: *Acquire the sensor information*

Table 7. Acquire the sensor information command

Start byte	Command field	Data field						CRC MSB	CRC LSB
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x0A	0x2E	0x00	0x00	0x00	0x00	0x00	0x00	0xFC	0x3C

Returned data:

Table 8. Returned data of acquire the sensor information command

Byte No.	Name	Value	Description
0	Start byte	0x0A	Start byte of the returned data frame
1	Data length	0x18	Length of the data field, it does not include byte No.0 and byte No.1
2	ACK byte	0xB0	Acknowledgment byte
3~18	UUID	……	Universally unique identifier of device
19	Year	……	Production date
20	Month	……	
21	Day	……	
22	Major version	……	Device version
23	Minor version	……	
24	CRC MSB	……	CRC values of current data frame
25	CRC LSB	……	

The following is an example of the returned sensor information data:

0x0A 0x18 0xB0 0x52 0x13 0x29 0x8C 0xC7 0xE0 0xE5 0x11 0x8D 0x2B 0xB9 0x57 0x2C 0xF3 0xAD 0x25 0x10 0x0A 0x08 0x01 0x09 0x22 0xE9

Decoding:

0x0A: Start byte

0x18: Data length (24 byte data)

0xB0: Acknowledge

0x52 0x13 0x29 0x8C 0xC7 0xE0 0xE5 0x11 0x8D 0x2B 0xB9 0x57 0x2C 0xF3 0xAD 0x25: UUID

0x10 0x0A 0x08: 16/10/08

0x01 0x09: Ver. 1.9

0x22 0xE9: CRC16-CCITT MSB and LSB byte

Command #2: *Continuous ranging*

Table 9. Continuous ranging command

Start byte	Command field	Data field						CRC MSB	CRC LSB
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x0A	0x24	0x00	0x00	0x00	0x00	0x00	0x00	0x0F	0x72

Command #3: *Single ranging*

Table 10. Single ranging command

Start byte	Command field	Data field						CRC MSB	CRC LSB
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x0A	0x22	0x00	0x00	0x00	0x00	0x00	0x00	0xAE	0x57

Returned data:

Table 11. Returned data of ranging results

Byte No.	**Name**	**Value**	**Description**
0	Start byte	0x0A	Start byte of the returned data frame
1	Data length	0x0D	Length of the data field, it does not include byte No.0 and byte No.1
2~4	Reserved	……	Reserved
5	Distance MSB	……	Measured distance, unit: mm
6	Distance LSB	……	
7	Magnitude MSB	……	Received signal magnitude
8	Magnitude LSB	……	
9	Magnitude Exp.	……	
10	Ambient ADC	……	Relative ambient IR intensity
11	Precision MSB	……	Precision indication, small values correspond to a small measurement errors
12	Precision LSB	……	
13	CRC MSB	……	CRC values of current data frame
14	CRC LSB	……	

The following is an example of the returned ranging data:

0x0A 0x0D 0x01 0x01 0x01 0x06 0xD9 0xFC 0x8C 0x02 0x01 0x00 0x01 0x9B 0x94

Decoding:

0x0A: Start byte

0x0D: Data length (13 byte data)

Distance = (0x06 * 256 + 0xD9) / 1000.0f = 1.753 (unit: m)

Magnitude = ((0xFC * 256 + 0x8C) << 0x02) / 1000.0f = 258.608

Ambient ADC = 1

Precision = (0x00 * 256) + 0x01 = 1

0x9B 0x94: CRC16-CCITT MSB and LSB byte

Command #4: *Stop ranging*

Table 12. Stop ranging command

Start byte	Command field	Data field						CRC MSB	CRC LSB
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x0A	0x30	0x01	0x00	0x00	0x00	0x00	0x00	0xBC	0x6F

Returned data:

Table 13. Returned data of stop ranging command

Byte No.	Name	Value	Description
0	Start byte	0x0A	Start byte of the returned data frame
1	Data length	0x03	Length of the data field, it does not include byte No.0 and byte No.1
2	ACK	……	0x01: Succeed; 0x00: Fail
3	CRC MSB	……	CRC values of current data frame
4	CRC LSB	……	

Command #5: *Set offset compensation value*

Table 14. Set offset compensation value command

Start byte	Command field	Data field						CRC MSB	CRC LSB
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x0A	0x38	0x1A	Offset MSB	Offset LSB	0x00	0x00	0x00	Calculated CRC MSB	Calculated CRC LSB

Offset = Actual distance – Sensor measured distance, unit: mm

Example:

Actual distance: 200mm, sensor measured distance: 215mm

Offset = 200 – 215 = -15 = 0xFFF1 (Offset MSB = 0xFF, Offset LSB = 0xF1)

Note: Due to the individual deviation of sensor performances, this command can be used to compensate the small offset deviation to achieve higher ranging precision. The offset values will be automatically saved to flash memory and reloaded with each power up.

Returned data:

Table 15. Returned data of set offset compensation value command

Byte No.	Name	Value	Description
0	Start byte	0x0A	Start byte of the returned data frame
1	Data length	0x03	Length of the data field, it does not include byte No.0 and byte No.1
2	ACK	……	0x01: Succeed; 0x00: Fail
3	CRC MSB	……	CRC values of current data frame
4	CRC LSB	……	

Command #6: *Load configuration profiles*

Table 16. Load configuration profiles command

Start byte	Command field	Data field						CRC MSB	CRC LSB
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x0A	0x30	0x00: User profile 0xFF: Factory profile	0x00	0x00	0x00	0x00	0x00	Calculated CRC MSB	Calculated CRC LSB

Returned data:

Table 17. Returned data of load configuration profiles command

Byte No.	Name	Value	Description
0	Start byte	0x0A	Start byte of the returned data frame
1	Data length	0x03	Length of the data field, it does not include byte No.0 and byte No.1
2	ACK	……	0x01: Succeed; 0x00: Fail
3	CRC MSB	……	CRC values of current data frame
4	CRC LSB	……	

Command #7: *Output filter adjustment*

Table 18. Output filter adjustment command

Start byte	Command field	Data field						CRC MSB	CRC LSB
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
0x0A	0x3D	0xAA	Filter MSB	Filter LSB	0x00	0x00	0x00	Calculated CRC MSB	Calculated CRC LSB

Filter default value: 0x0000

Example:

Decrease the output stability by 2350 units -> Filter value = -2350 = 0xF6D2 (Filter MSB = 0xF6, Filter LSB=0xD2)

Increase the output stability by 2350 units -> Filter value = 2350 = 0x092E (Filter MSB = 0x09, Filter LSB=0x2E)

Returned data:

Table 19. Returned data of output filter adjustment command

Byte No.	Name	Value	Description
0	Start byte	0x0A	Start byte of the returned data frame
1	Data length	0x03	Length of the data field, it does not include byte No.0 and byte No.1
2	ACK	……	0x01: Succeed; 0x00: Fail
3	CRC MSB	……	CRC values of current data frame
4	CRC LSB	……	

Package information

Table 20. Package details

Model No.	HPS-166
Module dimensions	Sensor: 34(W) x 24(H) x 23.5(D) mm (With lens cover)
Weight	7.9g / pcs (With lens cover)
Tray	Modules of 50pcs. (10*5) per tray
Outer box	4 trays per box (module 200pcs)

Revision history

Table 21. Document revision history

Date	Revision	Description
23-December-2016	1.0	Initial release.
01-January-2017	1.1	Add description of recommend FPC/FFC dimensions.
10-January-2017	1.2	Modify the Output data rate: 3-54 Hz.
05-Feburary-2017	1.3	Modify the device pinout of V1.91 hardware. Add command #5 ~ #7.

Appendix

CRC16-CCITT C-language Implementations

Implementation 1:

```
####################################################################
#include<stdio.h>
/**
Flash Space: Small
Calculation Speed: Slow
*/
/*Function Name：        crc_cal_by_bit         //Calculate CRC by bit
  Function Parameters：  unsigned char* ptr     //Pointer of data buffer
                         unsigned char len      //Length of data
  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      //Length of data
   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 byte table */
    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 half byte
   Function Parameters:   unsigned char* ptr    //Pointer of data buffer
                          unsigned char len     //Length of data
   Return Value:          unsigned int
   Polynomial:            CRC-CCITT 0x1021
*/
unsigned int crc_ta_8[256]={ /* CRC byte table */
    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;
}
##################################################################
```