**RS485 DC模式发送接收数据格式**

通讯波特率115200，校验位 None，数据位 8，停止位 1

**读数据时，发送的询问指令为（数据均为16进制）：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 |
| AA | 09 | 09 | 68 | 00 | 00 | 00 | 92 | 88 |

Byte0：同步字节，固定为AA。

Byte1：接收到的数据长度（包括CRC）。

Byte2：接收到的数据长度（包括CRC）。

Byte3：命令代码，读命令为68。

Byte4：Command Index。

Byte5：block number (start block)。

Byte6：number of blocks。（本例中读取1个block = 4 bytes）（00=1block，01=2block。。。）

Byte7：CRC-16

Byte8：CRC-16

**其返回的数据为：**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte8 | Byte9 |
| AA | 07 | 07 | 68 | 81 | 2B | 8C | AA | 0B | 0B |
| Byte10 | Byte11 | Byte12 | Byte13 | Byte14 | Byte15 | Byte16 | Byte17 |
| 68 | 9A | 01 | 02 | 03 | 04 | CC | A6 |

Byte0 – 6：Acknowledge信息

Byte7：同步字节，固定为AA。

Byte8：接收到的数据长度（包括CRC，不包括Acknowledge信息）

Byte9：接收到的数据长度（包括CRC，不包括Acknowledge信息）

Byte10：命令代码，读命令为68。

Byte11：Command Index。

Byte12：读取出的第1个字节（此例中只用到载码体的第一个BLOCK，读出的数值为01）。

Byte13：读取出的第2个字节。

Byte14：读取出的第3个字节。

Byte15：读取出的第4个字节。

Byte16：CRC-16

Byte17：CRC-16

**写数据时，发送的询问指令为（数据均为16进制）：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 |
| AA | 0D | 0D | 69 | 00 | 00 | 00 | 01 | 02 |
| Byte 9 | Byte 10 | Byte 11 | Byte 12 |
| 03 | 04 | C0 | 2A |

Byte0：同步字节，固定为AA。

Byte1：接收到的数据长度（包括CRC）。

Byte2：接收到的数据长度（包括CRC）。

Byte3：命令代码，写命令为69。

Byte4：Command Index。

Byte5：block number (start block)。

Byte6：number of blocks。（本例中读取1个block = 4 bytes）（00=1block，01=2block。。。）

Byte7：需要写入的第1个字节（此例中只用到载码体的第一个BLOCK，写入的数值为01）

Byte8：需要写入的第2个字节

Byte9：需要写入的第3个字节

Byte10：需要写入的第4个字节

Byte11：CRC-16

Byte12：CRC-16

**其返回的数据为：**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte8 | Byte9 |
| AA | 07 | 07 | 69 | 89 | BB | 19 | AA | 07 | 07 |
| Byte10 | Byte11 | Byte12 | Byte13 |
| 69 | 8A | 20 | 2B |

Byte0 – 6：Acknowledge信息

Byte7：同步字节，固定为AA。

Byte8：接收到的数据长度（包括CRC，不包括Acknowledge信息）

Byte9：接收到的数据长度（包括CRC，不包括Acknowledge信息）

Byte10：命令代码，写命令为69。

Byte11：Command Index。

Byte12：CRC-16

Byte13：CRC-16

**打开读写头指令（数据均为16进制）：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Byte 0 | Byte 1 | Byte 2 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
| AA | 07 | 07 | 48 | 00 | 99 | 3A |

**无返回数据**

**关闭读写头指令（数据均为16进制）：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Byte 0 | Byte 1 | Byte 2 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
| AA | 07 | 07 | 49 | 00 | 41 | 23 |

**无返回数据**

**Command Index说明：**

**Request**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |  |  |
|  |  |  |  |  |  |  |  | \_\_\_ | reserved | |
|  |  |  |  |  |  |  | \_\_ | \_\_\_ | 0 = Auto-tuning OFF 1 = Auto-tuning ON (volatile) | |
|  |  |  |  |  |  | \_\_ | \_\_ | \_\_\_ | 0 = “non-addr.” mode 1 = always “addressed mode” | |
|  |  |  |  |  | \_\_ | \_\_ | \_\_ | \_\_\_ | 0 = RESP. after time 1 = RESPONSE if TAG present | |
|  |  |  |  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_\_ | 0 = standard 1 = REQUEST contains TAG-UID | |
|  |  |  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_\_ | 0 = standard 1 = REQUEST contains CCC | |
|  |  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_\_ | 0 = new Inventory 1 = next (up to) 8 UID's | |
|  |  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_\_ | reserved | |

Bit 0: reserved

Bit 1: the transceiver is tuned before every TAG request (dynamic compensation of ambient conditions) – only valid for the current command (see 1.5.3.1, Parameter\_12 for permanent use)

Bit 2: switch between "non-addressed-mode" (default) and "addressed-mode" where only the TAG with the specified UID will communicate

Bit 3: Response after TimeOut (default) or not until the command was executed

Bit 4: 1 = address TAG with UID (addressed mode) included in request

Bit 5: 1 = address TAG with custom specific command (CCC) included in request

Bit 6: only if CC=Inventory:

0 = start new inventory-round, 1 = proceed with last inventory-round, transfer next UID's

Bit 7: reserved

**Acknowledge / Response / Error-Response**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |  |  |
|  |  |  |  |  |  |  |  | \_\_ | 00 = no\_ACK 01 = ACK | |
|  |  |  |  |  |  |  |  | \_\_ | 10 = Response 11 = Error-Response | |
|  |  |  |  |  |  | \_\_ | \_\_ | \_\_ | 0 = Single-TAG-Inventory 1 = Multi-TAG-Inventory | |
|  |  |  |  |  | \_\_ | \_\_ | \_\_ | \_\_ | 0 = no TAG present 1 = TAG\_present | |
|  |  |  |  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | 0 = no\_presence\_change 1 = TAG\_presence\_changed | |
|  |  |  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | 0 = Transmitter\_tuned 1 = Transmitter\_detuned | |
|  |  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | 0 = Inventory complete 1 = Inventory not complete,  more UID's available  0 = PW transfer failed 1 = PW transferred correctly | |
|  | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | \_\_ | 0 = Transceiver parameterized 1 = PARAM\_needed | |

Bit 0, Bit 1:

*no\_ACK(00)*: telegram fault, command not executed

*ACK (01)*: command was received correctly and will be executed

*Response (10)*: command executed correctly

Error-Response (11): fault during last command, see Error-Code\_1, Error-Code\_2 and Error-Code\_3

Bit 2: only at "INVENTORY": multiple UID's may follow

Bit 3: *TAG\_present*: 0 = no TAG in the air interface / 1 = TAG recognized in the air interface

Bit 4: *TAG\_Presence*: 0 = no change of TAG-Status / 1 = change of TAG status (more or less or other TAG’s in the air interface)

Bit 5: 0 = transmitter tuned correctly, 1 = transmitter reached tuning range limits

Bit 6:0 =Inventory-Response is complete, 0 < amount of TAG’s <= 8

1 = Inventory-Response is not complete, amount of TAG’s > 8,

next inventory necessary for transfering the remaining UID's (Multi-TAG-mode only)

1 = Password transferred correctly (only at Single-TAG-mode INVENTORY-Cmd and EM4233)

Bit 7: *PARAM\_needed*: Transceiver needs parameters, e.g. after power cycle or reset

**CRC计算说明：**

### CRC-16 Calculation, C-SourceCode

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* sample code \*/

/\* CRC-calculation \*/

/\* based on the ISO/IEC 15693 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#define POLYNOMIAL 0x8408 // x^16 + x^12 + x^5 + 1

#define PRESET\_VALUE 0xFFFF

#define NUMBER\_OF\_BYTES 4 // Example: 4 data bytes

void main()

{

unsigned int current\_crc\_value;

unsigned char array\_of\_databytes[NUMBER\_OF\_BYTES] = {0xAA, 0x06, 0x49, 0x00};

int number\_of\_databytes = NUMBER\_OF\_BYTES;

int calculate\_or\_check\_crc;

int i, j;

number\_of\_databytes = NUMBER\_OF\_BYTES;

current\_crc\_value = PRESET\_VALUE;

for (i = 0; i < number\_of\_databytes; i++)

{

current\_crc\_value = current\_crc\_value ^ ((unsigned int)array\_of\_databytes[i]);

for (j = 0; j < 8; j++)

{

if (current\_crc\_value & 0x0001)

{

current\_crc\_value = (current\_crc\_value >> 1) ^ POLYNOMIAL;

}

else

{

current\_crc\_value = (current\_crc\_value >> 1);

}

}

}

current\_crc\_value = ~current\_crc\_value;

printf ("Generated CRC is '%04X'\n", current\_crc\_value);

printf ("The Least Significant Byte (transmitted first) is: '%02X'\n",

current\_crc\_value & 0xFF);

printf( "The Most Significant Byte (transmitted second) is: '%02X'\n",

(current\_crc\_value >> 8) & 0xFF);

**C#写法：**

//CRC-CCITT Linux Kernel算法,初始值0xFFFF,LSB first

public static byte[] CRC(byte[] data,int len)

{

int crc =0xFFFF;

int poly = 0x8408;

byte[] crcLH=new byte[2];

for (int i = 0; i < len; i++)

{

crc = crc ^ (int)data[i];

for (int j = 0; j < 8; j++)

{

if ((crc & 0x0001) == 1)

{

crc = (crc >> 1) ^ poly;

}

else

{

crc = (crc >> 1);

}

}

}

crc = ~crc;

crcLH[0] = Convert.ToByte(crc & 0xff);

crcLH[1] = Convert.ToByte((crc >> 8) & 0xff);

return crcLH;

}