ElecKits UHF RFID Reader Module

AS3992 Protocol

Support Uart and USB version

ElecKits Technologies, Inc.
http://rfid.eleckits.com

After connecting the USB version reader to the computer it is automatically installed as a HID (Human Interface Device). The HID protocol defines different reports. Every report has its own report ID, length and a definition if it is an IN- or OUT-report. A report starts with a report ID.

1. Command-Frame

As mentioned before a frame starts with the report ID. The report ID is also called command in this documentation. The second byte is the length of the frame (the ID and the length bytes are included in the length).

Byte 1	Byte 2	Variable length
Report ID	Frame Length	Payload

Table 1: Command frame

2. Error Byte

Some commands from the controller to the host include an error byte:

0x00	No Error
0x80-0xFF	Look at the EPC Specification for more information
0xFF	No response from the Tag (time out)

Table 2: Error Byte

If the tag does not respond, the cause might be that the tag is no longer in the field or a communication error. For more information please refer to the EPC specification [EPC 2005] or Annex A below.

Error-Code Support	**Error Code**	**Error-Code Name**	**No reply error from Tag**
Error-specific	1000 0000	Other error	Catch-all for errors not covered by other codes
	1000 0011	Memory overrun or unsupported PC value	The specified memory location does not exist or the PC value is not supported by the tag
	1000 0100	Memory locked	The specified memory location is locked and/or perm locked and is either not writeable or not readable
	1000 1011	Insufficient power	The tag has insufficient power to perform the memory-write operation

Non-specific	1000 1111	Non-specific error	The tag does not support error-specific codes
no Reply	1111 1111	No reply error from Tag	The Tag has not replied to the reader command

Table 3: Annex A Error Codes

3. Memory Bank

Some commands contain the tag's memory bank:

Memory Bank	Value	Implementation state for Firmware
MEM_RES	0x00	Implemented since 0.0.6
MEM_EPC	0x01	Implemented
MEM_TID	0x02	Implemented since 0.0.6
MEM_USER	0x03	Implemented since 0.0.6

Table 4: Memory Bank

4. Reports(Commands)

Only the following reports are implemented. There are enough other values for additional new reports (value 0 is reserved and the maximum value is 0xFF). OUT stands for a report from the host to the controller and IN means a report from the controller to the host.

Report	value		Report	value
OUT_FIRM_HARDW_ID	0x10		OUT_KILL_TAG	0x3D
IN_FIRM_HARDW_ID	0x11		IN_KILL_TAG	0x3E
OUT_DEVICE_INFO	0x12		OUT_INVENTORY_6B_ID	0x3F
IN_DEVICE_INFO	0x13		IN_INVENTORY_6B_ID	0x40
OUT_CPU_RESET	0x16		OUT_CHANGE_FREQ	0x41
OUT_ANTENNA_POWER	0x18		IN_CHANGE_FREQ	0x42
OUT_WRITE_REG	0x1A		OUT_INVENTORY_RSSI	0x43
IN_WRITE_REG	0x1B		IN_INVENTORY_RSSI	0x44
OUT_READ_REG	0x1C		OUT_NXP_COMMAND	0x45
IN_READ_REG	0x1D		IN_NXP_COMMAND	0x46
OUT_INVENTORY	0x31		OUT_WRITE_TO_TAG_6B_ID	0x47
IN_INVENTORY	0x32		IN_WRITE_TO_TAG_6B_ID	0x48
OUT_SELECT_TAG	0x33		OUT_READ_FROM_TAG_6B_ID	0x49
IN_SELECT_TAG	0x34		IN_READ_FROM_TAG_6B_ID	0x50
OUT_WRITE_TO_TAG	0x35		OUT_FIRM_PROGRAM_ID	0x55
IN_WRITE_TO_TAG	0x36		IN_FIRM_PROGRAM_ID	0x56
OUT_READ_FROM_TAG	0x37		OUT_REGS_COMPLETE_ID	0x57

IN_READ_FROM_TAG	0x38		IN_REGS_COMPLETE_ID	0x58
OUT_LOCK_UNLOCK	0x3B		OUT_GEN2_SETTINGS_ID	0x59
IN_LOCK_UNLOCK	0x3C		IN_GEN2_SETTINGS_ID	0x5a

Table 5: Report List

It is advised to send the reports always with the maximal report length of 64 bytes. Most reports are already defined in the descriptor with the maximal length. The others may change in future. Windows truncates longer reports and discards shorter reports!

5. Reader-Oriented Commands

This command does not start a communication with the tags. They are usable for configuring the microcontroller and/or the AS399X and for getting (some) information.

Serial communication port (UART) configuration (initialization):

Parameter	Configuration
Port	COMx (this is serial communication port connecting to RFID reader)
Baud Rate	115200
Check bit	NONE
Data bits	8
Stop bits	1

Table 6: Initialization

Power-on reset or click reset switch S1, the serial port debugging tool show (see Figure 1):

Hello 20110324 World

INTVCO_lwm

Lwm_as399xInitialize () returned 0000

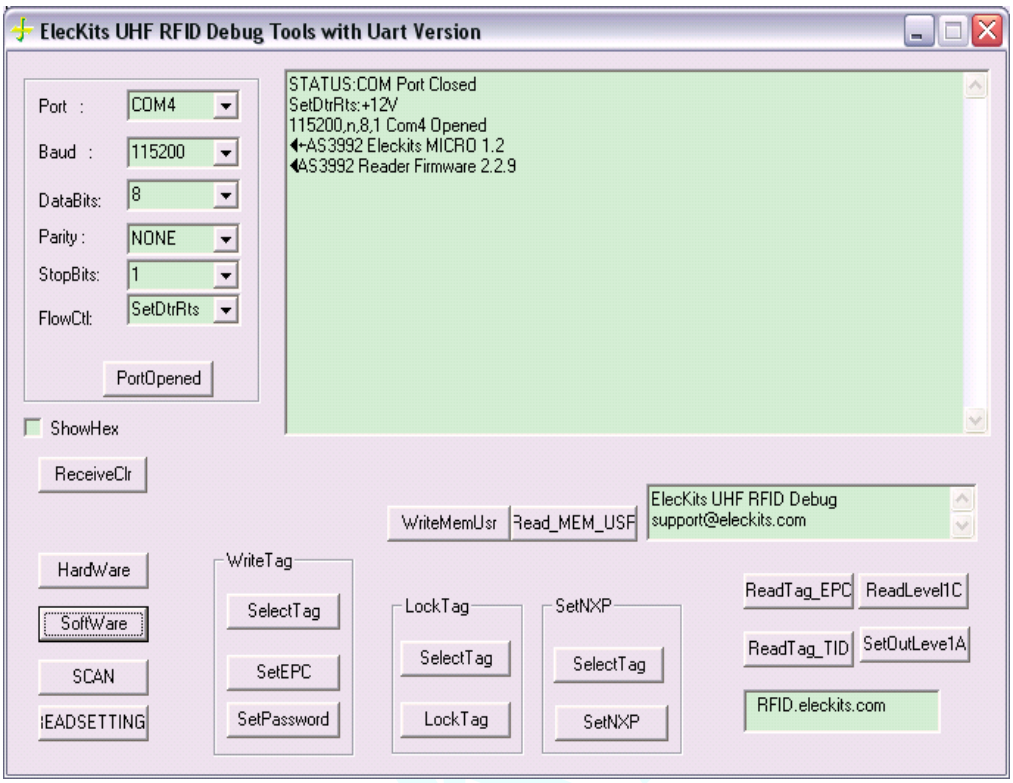

Figure 1.

5.1 Command Send Firm-/Hardware ID

This command is used to read out the reader's firmware and hardware ID. The command sent from the host to the microcontroller looks the following way:

Byte 0/ID	Byte 1	Byte 2
0x10	FRAME Length	Firm-/Hardware

Table 7: Command frame: Send Firm-/Hardware ID from host.

With byte 3 the host can select the ID which the microcontroller shall return:

value	ID
0x00	Firmware
0x01	Hardware

Table 8: Firmware-/Hardware ID byte (Byte 2).

The command sent to the host has the following form:

Byte 0/ID	Byte 1	Variable length but maximum 64 bytes
0x11	Frame Length	string

Table 9: Command frame: Send Firm-/Hardware ID from microcontroller.

Ec. Software version indentify command

Send: 10 03 00

Receive: 11 23 41 53 33 39 39 31 20 4D 69 6E 69 20 52 65 61 64 65 72 20 46 69 72 6D 77 61 72 65 20 31 2E 35 2E 31 (select the Show Hex)
Or receive: 0x11_AS3991 Mini Reader Firmware 1.5.1 (not select the Show Hex)

To achieve through the software key on the serial port debugging tool.

Hardware version indentify command
Send: 10 03 01
Receive: 11 22 41 53 33 39 39 31 20 52 4F 47 45 52 20 52 65 61 64 65 72 20 48 61 72 64 77 61 72 65 20 31 2E 32
Or receive: 0x11_AS3991 ROGER Reader Hardware 1.2

To achieve through the hardware key on the serial port debugging tool.

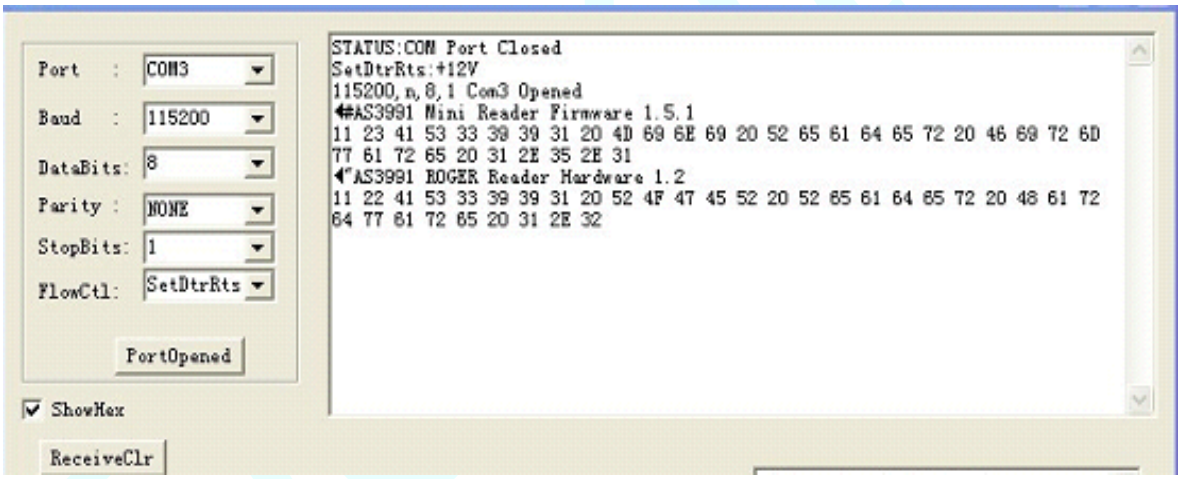

Figure 2.

To the software key, the codes are as follows:

```
void CGpsDlg::OnButtonSoftware()
{
    // TODO: Add your control notification handler code here
    BYTE buf[] = {0x10,0x03,0x00};
    CByteArray hexdata;
      this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
      this->m_CommCtrl.SetOutput(COleVariant(hexdata));
}
```

To the hardware key, the codes are as follows:

```
void CGpsDlg::OnButtonHardware()
{
    char Send_str[]={0X10,0X03,0X01};
        CByteArray hexdata;       //buffer how to into cbytearray
this->m_Function.ByteToByteArray((BYTE*)Send_str,sizeof(Send_str),hexdata);
        this->m_CommCtrl.SetOutput(COleVariant(hexdata));
}
```

5.2 Command Antenna Power

To change the antenna power uses this command.

Command from host:

Byte 0/ID	Byte 1	Byte 2
0x18	Frame length	Antenna power

Table 10: Command frame: Antenna Power from host

With byte 3 the host can select the antenna output power:

Value	Antenna Power
0x00	Power OFF
0x01 - 0xFE	Reserved to change the output level in later versions.
0xFF	Power ON

Table 11: Antenna Power Byte

Response from microcontroller:

Byte 0/ID	Byte 1	Byte 2
0x19	Frame length	Rfu

Table 12: Command frame: Write Register from controller

Byte 2 is reserved for further use.

Ec. Send: 18 03 00

Receive: 19 03 00

Now the microcontroller can not communicate with the tags.

This command acts as a switch of antenna control.

5.3 Command Write Register

The write register command can be used to directly manipulate the AS399X

registers.

Command from host:

Byte 0/ID	Byte 1	Byte 2	Byte 3 (Byte 4 & 5 optional)
0x1A	Frame length	Register Address	Data

Table 13: Command frame: Write register from host.

If data is longer than one byte the data is written into one of the 3 bytes deep register.
Response from microcontroller:

Byte 0/ID	Byte 1	Byte 2
0x1B	Frame length	Error byte

Table 14: Command frame: Write Register from controller

The controller sends back an error byte if anything goes wrong.

Ec. Send: 1A 04 08 00

Receive: 1B 03 00 No error

RX Wait Time (08)

Bit	Signal Name	Function	Comments
B7	Rxw7	RX wait time	Defines the time during which the RX input is ignored. It starts from the end of TX. RX wait range is 6.4µs to 1632µs (1..255), Step size 6.4µs, 00: receiver enabled immediately after TX. ISO 1800-6C(Gen2) Gen2: T1min=11.28µs…262us, ISO 1800 - 6A: 150…1150µs ISO 1800 - 6B: 85…460µs
B6	Rxw6		
B5	Rxw5		
B4	Rxw4		
B3	Rxw3		
B2	Rxw2		
B1	Rxw1		
B0	Rxw0		

1. Defines the time after TX when the RX input is disregarded.

Notes:

1. Preset at por=H or EN=L and at each write to 'Protocol control' register
2. Gen2: 07(44.8µs < 54.25µs…84.5µs – LF:160kHz)

5.4 Command Read Register

To read directly an AS399X register, use this command.

Command from host:

Byte 0/ID	Byte 1	Byte 2
0x1C	Frame length	Register Address

Table 15: Command frame: Read register from host.

If one of the 3 bytes deep registers is selected the controller sends back 3 bytes data.
Response from microcontroller:

Byte 0/ID	Byte 1	Byte 2 (Byte 3 & 4 optional)
0x1D	Frame length	Data

Table 16: Command frame: Read register from microcontroller.

Ec. Read the RX Wait Time register,

Send: 1C 03 08 Receive: 1D 06 00 00 00 00
Write to it,
Send: 1A 04 08 07 Receive: 1B 03 00
Read again,
Send: 1C 03 08 Receive: 1D 06 07 00 00 00

5.5 Command Change Frequency

To change the frequency of the reader and to get the reflected power or the RSSI value of the channel, use this command.

Command from host:

Byte 0/ID	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
0x41	Frame length	MASK	Freq low Byte	freq mid Byte	freq high Byte	RSSI level

Table 17: Command frame: Read register from host.

With the mask byte, it is possible to select either the RSSI value that is scanned with no carrier (LBT) or the reflected power that is received with activated carrier.
Mask 0x00: No specific value; - measurement skipped no valid dates in response
Mask 0x01: RSSI scan
Mask 0x02: reflected power scan
Mask 0x04: turn hop mode on; - add the frequency to the List
Mask 0x08: turn hop mode off clear the List
Mask 0x10: set LBT parameters

Byte 0/ID	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0x41	Frame	0x10	listening	listening	max	max	idle	idle

	length		Time low	Time high	SendingTime low	SendingTime high	Time low	Time high

Table 18: Command frame.

The frequency is transmitted in kHz: that means 868000 means 868 MHz.
With the RSSI value in dBm you can define the LBT border which is used in the hop mode.

Response from microcontroller:

Byte 0/ID	Byte 1	Byte 2	Byte 3
0x42	Frame Length	Data X	Data Y

Table 19: Command frame: Get data after set frequency

You can change the frequency of the reader using this command. The frequency profiles for different countries are shown in the following table:

Profile	Start freq [khz]	End freq [khz]	Increment [khz]	RSSI Threshold [dBm]	Listen Time [ms]	Idle Time [ms]	Max. Allocation [ms]
Europe	865,700 0x0d35a4	867,500 0x0d3cac	600 0x0258	-40 0xd8	1	0	10000
Japan	952,400	952,600	200	-87	10	100	4000
USA	902,750 0x0dc65e	927,250 0x0e2612	500 0x01f4	-40 0xd8	1	0	400
China 920.625	920,625	924,375	750	-40	1	0	10000
China 840.125	840,125	844,875	250	-40	1	0	10000
Korea	917,300	920,300	600	-40	1	0	10000

Table 20: Frequency Profiles

Specific European frequency:

865700 khz	866300 khz	866900 khz	867500 khz
0x0d35a4	0x0d37fc	0x0d3a54	0x0d3cac

The existing frequency profiles are defined in a CSV (Comma Separated Values) file within the installation directory (e.g. C:\Program Files\AS399x Reader Suite). The “profiles.csv” file can be modified if the user may want to add a new frequency profile by adding a new line.

The syntax for adding a new profile is defined as follows:

Name,StartFreq,StopFreq,Increment,dBm,listenTime,IdleTime,maxAllocationTime

For example the European frequency profile is defined as:

Europe,865.7,867.5,0.6,-40,1,0,10000

Besides changing the frequency profile the user may change each frequency related parameter individually.

Ec. Send: 41 08 08 AC 3C 0D D8 01

Receive: 42 40 FE FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Value	Meaning
0x41	Command Frequency change ID
0x08	Frame Length
0x08	Turn hop mode off clear the List
0xAC 0x3C 0x0D	The frequency of the reader you want,867.5Mhz
0xD8	RSSI Threshold,-40 dBm
0x01	Profile No.

Notes: After this command is sent, the reader will work at the frequency you set. The reader does not work in the hop mode, it works at a fixed frequency.

Send: 41 08 04 54 3A 0D D8 01

Receive: 42 40 FC FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Value	Meaning
0x41	Command Frequency change ID
0x08	Frame Length
0x04	turn hop mode on; - add the frequency to the List
0x54 0x3A 0x0D	The frequency of the reader you want to add to the list,866.9Mhz
0xD8	RSSI Level,-40 dBm
0x01	Profile
0x42	Answering sends 0x41
0xFE	Answering sends 0x08, hop mode off
0XFC	Answering sends 0x04, hop mode on

Notes: After this command is sent, the reader will work in the hop mode, it works at the frequency that hops between 867.5Mhz and 866.9Mhz. You can

continue to add new frequency to the list accordingly.

Use the serial port debugging tool:

Click ReadSetting button, check box ShowHex, pop AdvanceReaderSettings dialog box, select Europe in the profile list box, click the OK button, see below:

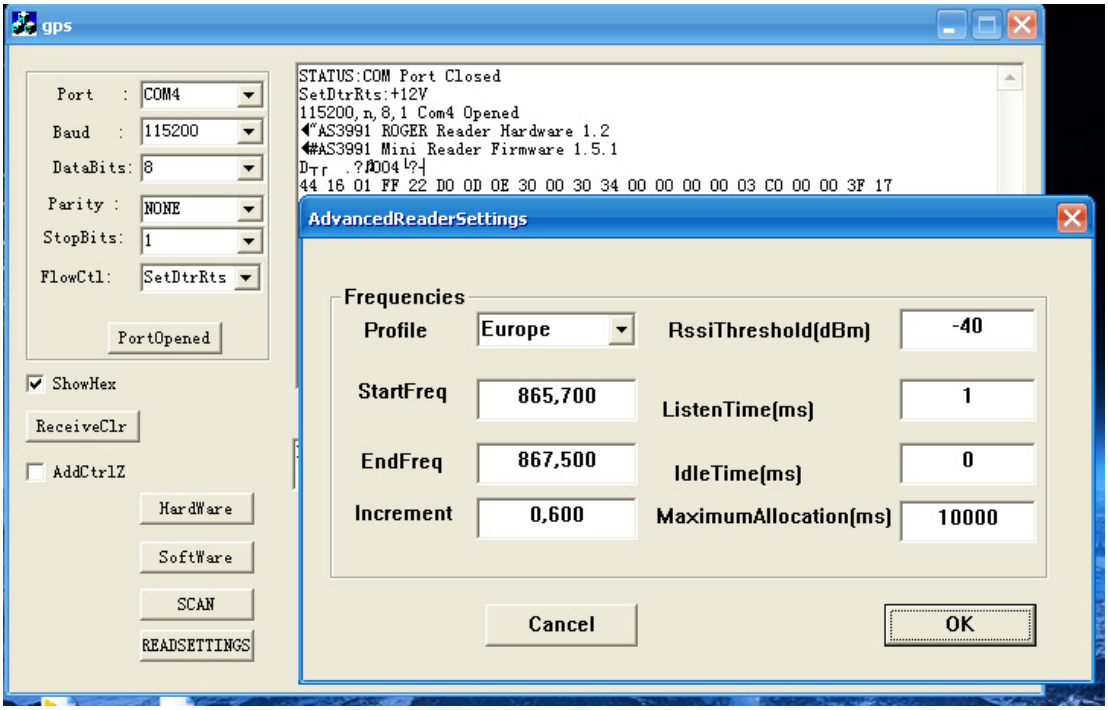

Figure 3.

Codes are as follows:

```
void CGpsDlg::OnButtonReadsettings()
{
    // TODO: Add your control notification handler code here
    if(this->m_ReadSettingsDlg.DoModal()==IDOK)
    {
        //UINT freqs[50];//usa frequency numbers
          UINT start,end,increment,freq;
    //    unsigned char mode =8;
          m_mode = 8;
          start = this->m_ReadSettingsDlg.m_StartFreq;//KHZ
          end = this->m_ReadSettingsDlg.m_EndFreq;
          increment = this->m_ReadSettingsDlg.m_Increment;

          this->m_freqs_size = 0;
          for ( freq = start; freq <= end; freq += increment)
          {
```

```
                m_freqs[m_freqs_size++] = freq;
            }
            this->SetTimer(1,1000,NULL);//start time1
        }//
    }
    void CGpsDlg::OnTimer(UINT nIDEvent)
    {
        // TODO: Add your message handler code here and/or call default
        if(1==nIDEvent)
          {
            unsigned char rssi =this->m_ReadSettingsDlg.m_RssiThreshold;
             unsigned char profile = this->m_ReadSettingsDlg.m_profile;
                //for(int i=0;i<size;i++)//
                //{
            static int size = 0;
            if(size < this->m_freqs_size)
            {
                this->setFrequency(m_freqs[size],this->m_mode,rssi,profile);
                this->m_mode = 4;
                size++;
            }
            else
            {
                size =0;
                //shold kill the timer
                KillTimer(1);
            }
        }
    }
    void CGpsDlg::setFrequency( UINT frequencyKHz, unsigned char mode, unsigned char rssi,
    unsigned char profile)
    {
        BYTE buf[64];//QByteArray buf;
    //    buf.resize(64);
        buf[0] = OUT_CHANGE_FREQ;//0x41
        buf[1] = sizeof(buf);//7;
        buf[2] = mode;
        buf[3] = frequencyKHz & 0x000000FF;
        buf[4] = (frequencyKHz & 0x0000FF00) >> 8;
        buf[5] = (frequencyKHz & 0x00FF0000) >> 16;
        buf[6] = rssi;
        buf[7] = profile;
        CByteArray hexdata;
```

```
    this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
    this->m_CommCtrl.SetOutput(COleVariant(hexdata));
}
```

The serial port debugging tool receives:

42 40 FE FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

42 40 FC FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

42 40 FC FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

42 40 FC FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

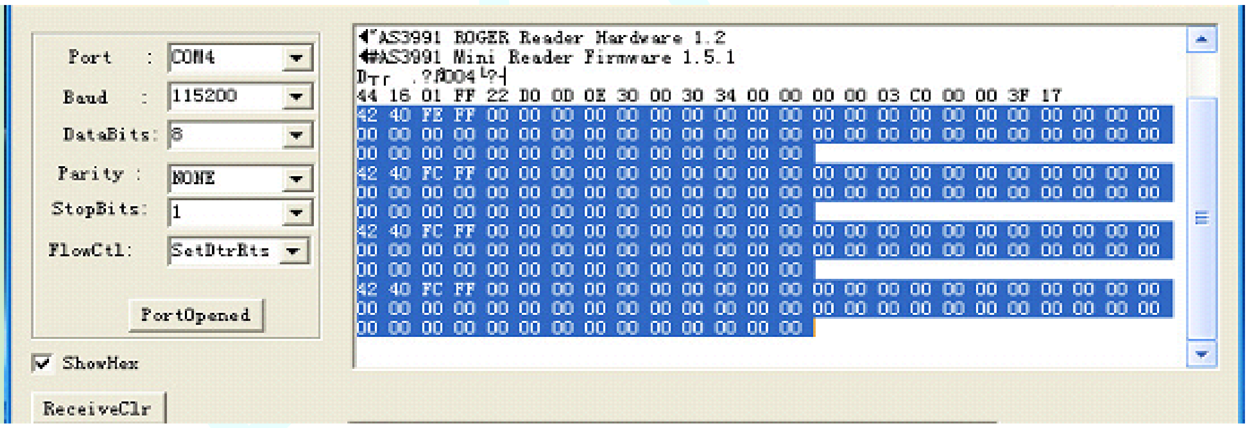

Figure 4.

There are a total of 4 responses to the European frequency corresponding to its four frequencies. There are a total of 50 responses to the USA frequency and so on.

5.6 Set GEN2 Parameters

To access GEN2 specific parameters use this report:

Command from host:

Byte 0/ID	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
0x59	Frame length	linkfrequency set	linkfrequency	miller set	miller setting
Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
session set	session	trext set	trext	qbegin set	qbegin

Table 21: Command frame:

The “set” bytes define if the subsequent byte should be set in the reader firmware. The answer to this command returns all the actually set values. This allows reading out the values without actually changing anything.

The parameters are:

·**linkfrequency:** 0=40kHz,3=80kHZ,6=160kHz,8=213kHz, 9=256 kHz, 12=320kHz, 15=640kHz

· **miller:** 0=FM0, 2=Miller2, 2=Miller4, 3=Miller8

· **session:** 0=S0, 1= S1, 2=S2, 3=S3, 4=S4

· **trext:** 1 use long pilot tone, 0: don’t use

· **qbegin:** Start value for q when doing inventory rounds. The first round will have 2^q slots

The controller sends back following frame:

Byte 0/ID	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
0x5A	Frame length	0	linkfrequency	0	miller setting
Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
0	session	0	trext	0	qbegin

Table 22: Command frame: Lock Tag Memory from controller

5.7 Register Bulk

Gets the complete register list in one big bulk.

Command from host:

Byte 0/ID	Byte 1

0x57	Frame length

Table 23: Command frame:

Response from device:

Byte 0/ID	Byte 1	Byte 2	…	Byte 21	Byte 22	Byte 23	…	Byte 42
0x58	Frame length	register 0		register 0x12-1	register 0x12-2	register 0x13		register 0x1e

Table 24: Complete Register List (Bulk)

Ec. Send: 57 02

Receive: 58 2D 02 06 F0 62 35 05 00 07 07 01 08 02 00 37 0B 10 98 02 0C 40 00 38 83 84 0A 06 3F 20 06 41 E4 46 18 01 00 87 00 00 00 00 00 00 00

It gets the value in each register of AS388X.

6. Transponder Oriented Commands

Transponder oriented commands are commands which force the microcontroller to communicate with the tags. Therefore the antenna power output must be enabled and there must be at least one tag in the field.

6.1 Command Inventory

To start an inventory round to find new tags use this command. To get tag information of all found tags also use this command with the next tag information flag set.

Command from host:

Byte 0/ID	Byte 1	Byte 2
0x31	Frame Length	Start inventory / Next tag information

Table 25: Command frame: Inventory from host

With byte 2 the host can select whether it wants to start a new round or if it wants to get information about the next tag in the microcontroller list.

Value	Start/Next
0x01	Start inventory round
0x02	Next Tag information (should not be sent anymore since v1.3.0)

Table 26: Start/Next byte (Byte 2).

Response from microcontroller:

Byte 0/ID	Byte 1	Byte 2	Byte 3	Byte 4-Byte xx	Byte xx+1..Byte 63
0x32	Frame length	Number of found tags	Length of EPC byte	EPC 1…x	rfu

Table 27: Command frame: Inventory from microcontroller on Firmware.

With byte 2 the controller reports how many tags are found by the inventory command. After sending the first inventory with the next flag set, the controller sends back only the count of the leftover tags. This is used to inform the host how often he has to call the inventory command with the next flag set until he has the tag information of all found tags. But the tag information is still in the microcontroller's tag list. No tag information is deleted. The complete report length is 64 bytes and needs to be taken into account in the Host Software.

Ec. Send: 31 03 01

Receive: 32 12 01 0E 30 00 01 02 03 04 05 06 07 08 09 10 6A 0F

Value	Meaning
0x32	Answering sends 0x31
0x12	Frame length,0x12=18
0x01	Representatives one label
0x0E	Length of EPC byte
0X30 0X00	Reserved
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x10 0x6A 0x0F	RFID tag ID. Different tag has different value.

6.2 Command Inventory with RSSI

To start an inventory round to find new tags use this command. This command must be executed first. Reading or writing to the tag will not work without this command executed first. To get tag information of all found tags also use this command with the next tag information flag set.

Command from host:

Byte 0/ID	Byte 1	Byte 2
0x43	Frame Length	Start inventory / Next tag information

Table 28: Command frame: Inventory from host

With byte 3 the host can select whether it wants to start a new round or if it wants to get information about the next tag in the microcontroller list.

value	Start/next
0x01	Start inventory round
0x02	Next tag information

Table 29: Start/Next byte (Byte 2).

Response from microcontroller:

Byte 0/ID	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5-Byte xx	Byte xx+1..Byte 63
0x44	Frame length	Number of found tags	RSSI	Length of EPC byte	EPC 1…x	rfu

Table 30: Command frame: Inventory from microcontroller.

With byte 2 the controller reports how many tags are found by the inventory command. After sending the first inventory with the next flag set, the controller sends back only the count of the leftover tags. This is used to inform the host how often he has to call the inventory command with the next flag set until he has the tag information of all found tags. But the tag information is still in the microcontroller's tag list. No tag information is deleted. The complete report length is 64 bytes and needs to be taken into account in the Host Software.

Ec. Send: 43 03 01

Receive: 44 16 01 9E AC 3C 0D 0E 30 00 01 02 03 04 05 06 07 08 09 10 6A 0F

Value	Meaning
0x44	Answering sends 0x43
0x16	FRAME length
0x01	Representatives one label
0x9E	Means Q value, I value; Signal strength of signal Q =(0x9E>>4)*2=18 Signal strength of signal I =(0x9E&0x0F)*2=28
0xAC 0x3C 0x0D	Frequency 866900kz，can be obtained by calculation 0D<<16 | 3C<<8 | AC = 0x0D3CAC=867500kz=867.5M=865.7+0.6+0.6+0.6M; This is European standard frequency; For example, the European frequency profile is define as: Europe,865.7,867.5,0.6,-40,1,0,10000
0x0E	Length of EPC byte
0x30 0x00	Reserved
0x01 0x02	

0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x10 0x6A 0x0F	RFID tag ID. Different tag has different value.

Click the SCAN button on the serial port debugging tool, see below:

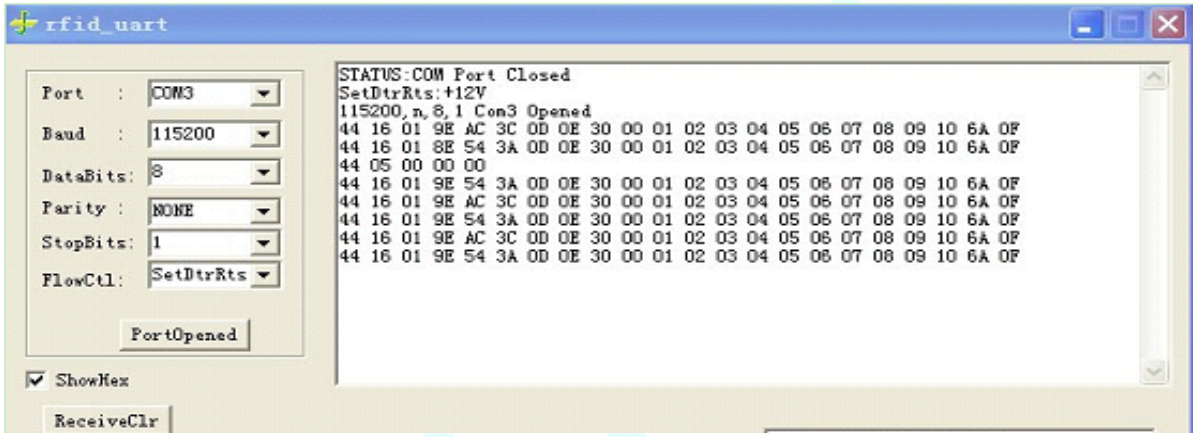

Figure 5.

Codes are as follows:

```
void CGpsDlg::OnButtonScan()
{
    // TODO: Add your control notification handler code here
    BYTE buf[] ={0X43,0X04,0X01,0Xcd};
    CByteArray hexdata;
      this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
      this->m_CommCtrl.SetOutput(COleVariant(hexdata));
}
```

6.3 Command Select or Isolate Tag

To communicate with one tag the host must isolate one of the found tags. The host needs to send always all EPC bytes to the controller regardless how long

the EPC mask is specified in order to ensure the USB protocol. The complete report length is 64 bytes and needs to be taken into account in the Host Software.
This command is needed for a read or write operation in case there are several Tags found.

The correct sequence to operate that command is shown below:

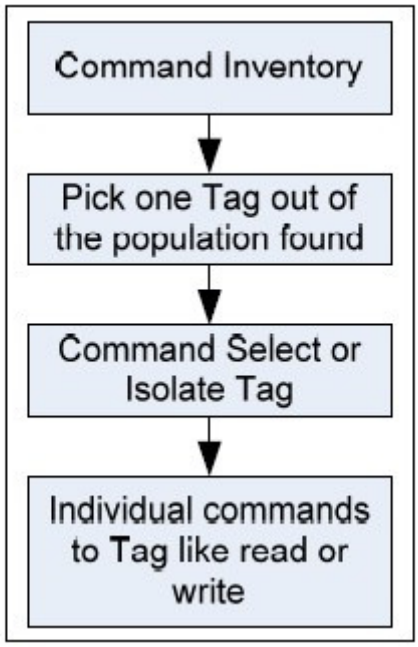

Figure 6: Read or write operation in case of several tags

Command from host:

Byte 0	Byte 1	Byte 2	Byte 3	Byte n+4	Byte n+5…byte 63
Report ID 0x33	Frame length	Length of EPC mask	EPC byte 0	EPC byte n	ruf

Table 31: Command frame: Select tag from host

The tag information is used to select one tag.
If the access password is zero, the controller did not access the Tag.

Response from microcontroller:

Byte0/ID	Byte1	Byte2
0x34	Frame length	Error byte

Table 32: Command frame: No access to tag

The error byte Information is described in Annex A

Ec. Send: 33 0F 0C 01 02 03 04 05 06 07 08 09 10 6A 0F

Receive: 34 03 00

Find the tag - 01 02 03 04 05 06 07 08 09 10 6A 0F

If receive: 34 03 09

Not find the tag.

Click the SelectTag button on the serial port debugging tool, pop SelectTag dialog box, see below:

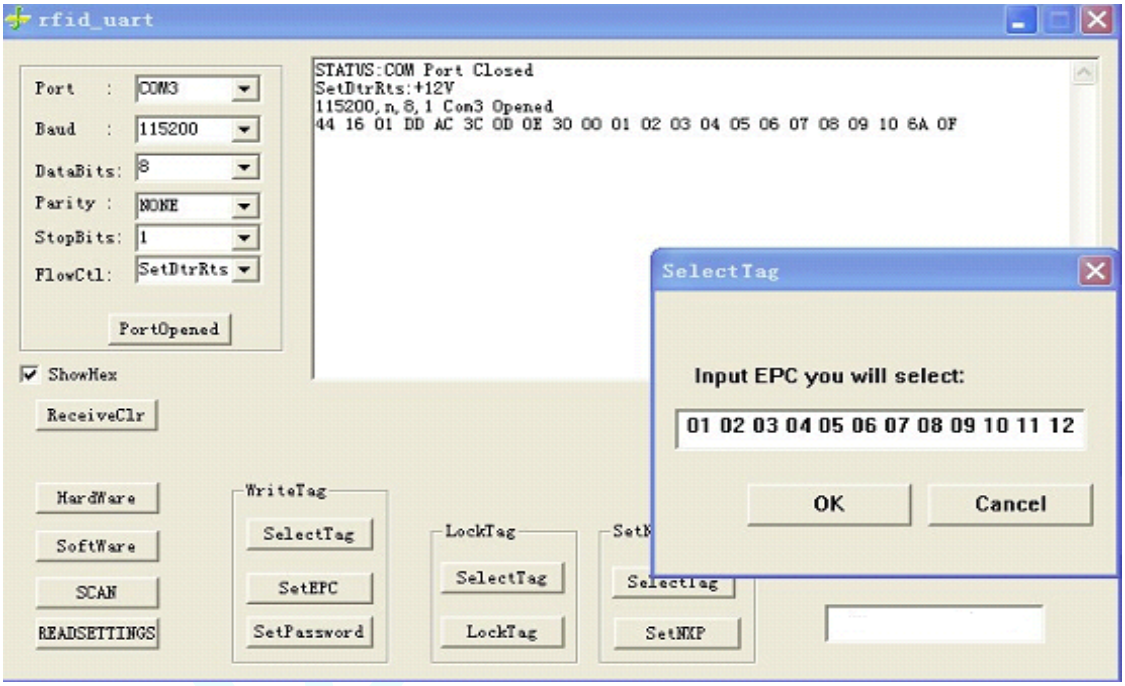

Figure 7.

Then input EPC you will select, click the OK button.

Codes are as follows:

```
void CGpsDlg::OnBUTTONSelectTag()
{
    // TODO: Add your control notification handler code here
    // TRACE("Select Tag\n");
    if(this->m_CSelectTagDlg.DoModal()==IDOK)
    {
        CByteArray hexdata;
        int len = this->m_Function.String2Hex(m_CSelectTagDlg.m_EditSelectEPC,hexdata);
        if(len ==12)
        {
            BYTE buf[64];
```

```
                buf[0] = OUT_SELCET_TAG;        // Report ID;0x33 lwm
                buf[1] = sizeof(buf);
                buf[2] =len;            //// EPC size
                for(int n=0;n<hexdata.GetSize();n++)
                {
                    buf[n+3] = hexdata.GetAt(n);
                }
                //      CByteArray hexdata;
                this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
                this->m_CommCtrl.SetOutput(COleVariant(hexdata));
            }
            else
                MessageBox("EPClength!=12");
        }
    }
```

6.4 Command Write to Tag/Set EPC AccessPassword

6.4.1 Command Write to Tag

To write some information into the tag this function is used.

Command from host:

Byte 0	Byte1	Byte2	Byte3	Byte[4] — Byte[7]	Byte 8	Byte[9]— Byte[2*n+9]	Byte[2*n+10] …Byte63
Report ID 0x35	Frame Length	Memory bank	Tag memory Address(in words)	Access Password (4 bytes Long)	Data length n in words	Data[2×n]	rfu

Table 33: Command frame: Write to Tag from host

Response from microcontroller:

Byte0/ID	Byte1	Byte2	Byte3
0x36	Frame length	Error byte	Number of words written

Table 34: Command frame: Write to Tag from controller

The controller sends back an error byte if anything goes wrong. The error byte Information is described in Annex A.

Ec. Send: **35 15 01 02 00 00 00 00 06 01 02 03 04 05 06 07 08 09 10 11 12**

Receive: **36 04 00 06** No error.

Value	Meaning
0x15	Frame length,0x15=21
0x01	MEM_EPC,see table 4
0x02	EPC memory space,see table 41
0x00 0x00 0x00 0x00	Access password,default:00 00 00 00
0x06	Written words,6*2=12 byte
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x10 0x11 0x12	The numbers written into the tag
0x36	Answering sends 0x35
0x04	Frame length
0x00	No error
0x06	Written words,6*2=12 byte

Notes: Before you write, you should find/select the tag that you want to write first. When you find it, you can change the tag number. Find the tag:

01 02 03 04 05 06 07 08 09 10 6A 0F

Click the SetEPC button, see below:

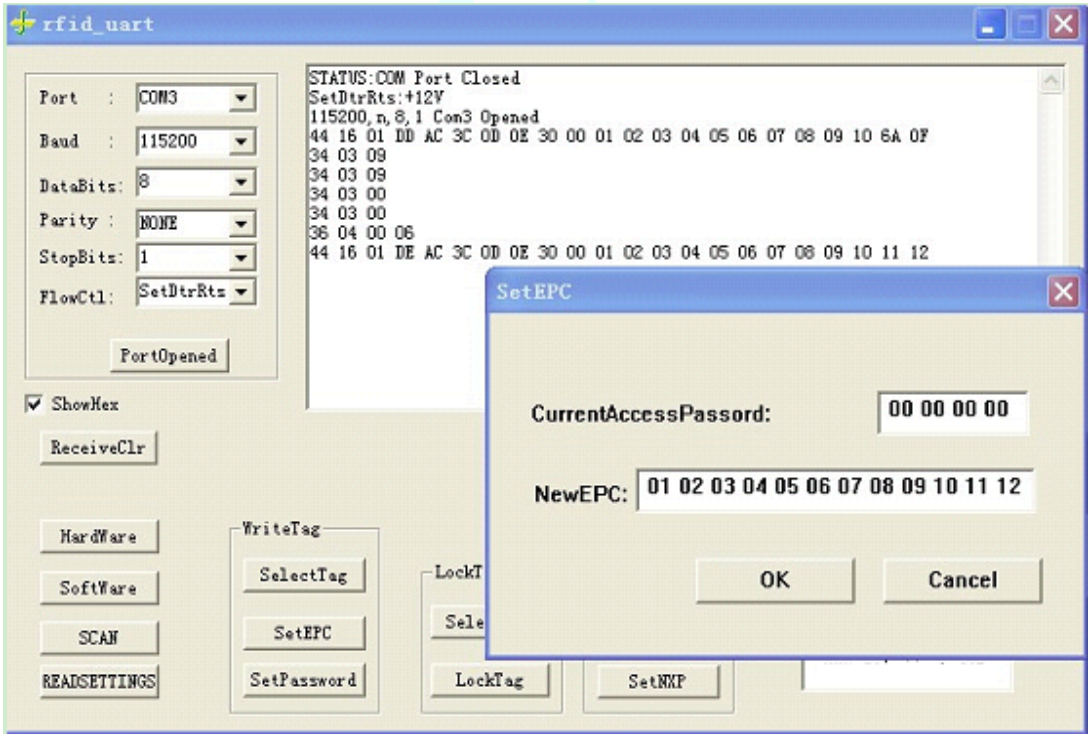

Figure 8.

Codes are as follows:

```
void CGpsDlg::OnBUTTONSetEPC()
{
    // TODO: Add your control notification handler code here
    TRACE("SetEPC\n");
    if(this->m_CSetEPCDlg.DoModal()==IDOK)
    {
        CByteArray hexdata,hexdata_password,hexdata_epc;
        int len;
        len=
this->m_Function.String2Hex(m_CSetEPCDlg.m_EditCAPassword,hexdata_password);
        if(len != 4)
        {
            MessageBox("PasswordLength!=4!","error");
            return ;
        }
        len = this->m_Function.String2Hex(m_CSetEPCDlg.m_EditNewEPC,hexdata_epc);
        if(len !=12)
        {
            MessageBox("NewEPCLength!=12!","error");
            return ;
        }
        BYTE buf[64];
        buf[0]= OUT_WRITE_TO_TAG; // Report ID;lwm OUT_WRITE_TO_TAG    = 0x35
        buf[1] = sizeof(buf);
        buf[2] = MEM_EPC;                   // Bank: EPC        MEM_EPC =0X01; LWM
        buf[3] = MEMADR_EPC;                //Address          #define          MEMADR_EPC
0x02
        for(int n=0;n<hexdata_password.GetSize();n++)
        {
            buf[n+4] = hexdata_password.GetAt(n);
        }
        buf[8] = hexdata_epc.GetSize()/2;//getlength in words
        for(int m=0;m<hexdata_epc.GetSize();m++)
        {
            buf[m+9] = hexdata_epc.GetAt(m);
        }
        this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
        this->m_CommCtrl.SetOutput(COleVariant(hexdata));
    }
}
```

After this command, if you write with no error, the EPC has been changed. The new EPC is as follows:

01 02 03 04 05 06 07 08 09 10 11 12

6.4.2 Set EPC AccessPassword

You can set EPC access password using this command.Do follow these steps:

Find the fag number Command Inventory → Pick one Tag out of the population found →Command Select or Isolate Tag → Individual commands to Tag like read or write, set password

When you use command select tag and the tag you want to set is present,

Send: 35 0D 00 02 00 00 00 00 02 11 22 33 44

Receive: 36 04 00 02 No error.

Value	Meaning
0x0D	Frame length
0x00	MEM_RES,see table 4
0x02	EPC memory space,see table 41
0x00 0x00 0x00 0x00	CurrentAccess password,default:00 00 00 00
0x02	Written words,2*2=4 byte
0x11 0x22 0x33 0x44	New password

Click the SetPassword button, see below:

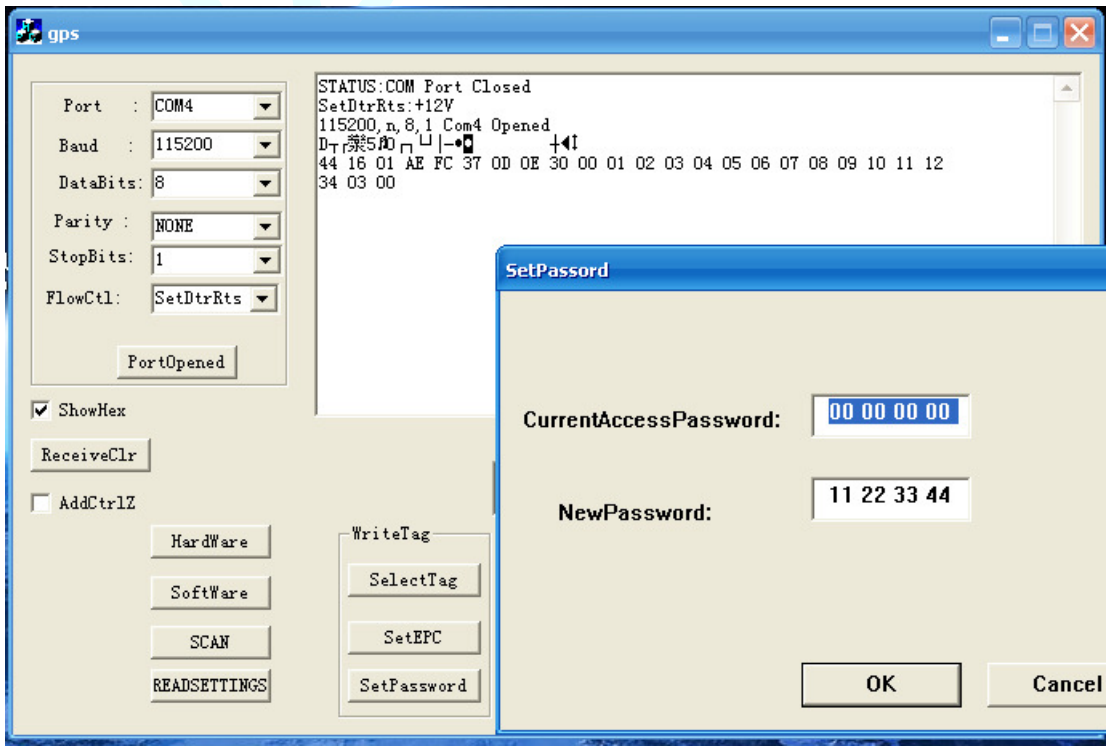

Figure 9.

Codes are as follows:

```
void CGpsDlg::OnBUTTONSetPassword()
{
    // TODO: Add your control notification handler code here
    TRACE("Set password\n");
    if(this->m_CSetPasswordDlg.DoModal()==IDOK)
    {
        CByteArray hexdata,hexdata_Curpassword,hexdata_newpassword;
        int len;
        len=
this->m_Function.String2Hex(m_CSetPasswordDlg.m_EditCuAPassword,hexdata_Curpassword);
        if(len != 4)
        {
            MessageBox("PasswordLength!=4!","error");
            return ;
        }
        len                                                                    =
this->m_Function.String2Hex(m_CSetPasswordDlg.m_EditNewPassword,hexdata_newpassword)
;
        if(len != 4)
        {
            MessageBox("PasswordLength!=4!","error");
            return ;
        }
        BYTE buf[64];
        buf[0] = OUT_WRITE_TO_TAG; // Report ID;lwm OUT_WRITE_TO_TAG   = 0x35,
        buf[1] = sizeof(buf);
        buf[2] = MEM_RES;    // MEM_RES 0X00
        BYTE wordaddr = 0;
        wordaddr += 2;
        buf[3] = wordaddr;
        for(int n=0;n<hexdata_Curpassword.GetSize();n++)
        {
            buf[n+4] = hexdata_Curpassword.GetAt(n);
        }
        buf[8]=hexdata_newpassword.GetSize()/2;//getlength in words
          for(int m=0;m<hexdata_newpassword.GetSize();m++)
        {
            buf[m+9] = hexdata_newpassword.GetAt(m);
        }
        this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
```

```
            this->m_CommCtrl.SetOutput(COleVariant(hexdata));
        }
    }
```

Click the OK button, it receives: 36 04 00 02

6.4.3 Write to MEM_USER

First select the tag you want to read, codes are as follows:

```
void Crfid_uartDlg::OnBUTTONWriteMemUsr()
{
    // TODO: Add your control notification handler code here
    TRACE("OnButtonWrite_Mem_USR\n");
    if(this->m_CWriteMemUsrDlg.DoModal() == IDOK)
    {
       CByteArray hexdata,hexdata_MemUsrAdr,hexdata_data;
          int len;
          len                                                                    =
this->m_Function.String2Hex(this->m_CWriteMemUsrDlg.m_MemUsrAdr,hexdata_MemUsrAdr
);
          if(len != 1)
          {
               MessageBox("MemUsrAdrLength!=1!","error");
               return ;
          }
          len                                                                    =
this->m_Function.String2Hex(this->m_CWriteMemUsrDlg.m_EditMemUsrData,hexdata_data);
          if(len !=16)
          {
               MessageBox("MemUsrDataLength!=16!","error");
               return ;
          }
          BYTE buf[64];
          buf[0] = OUT_WRITE_TO_TAG; // Report ID;lwm OUT_WRITE_TO_TAG   = 0x35,
          buf[1] = sizeof(buf);
          buf[2] = MEM_USER;                // Bank: 0X03;
          buf[3] = hexdata_MemUsrAdr.GetAt(0);
          //for(int n=0;n<hexdata_password.GetSize();n++)
    //    {
          //     buf[n+4] = hexdata_password.GetAt(n);
    //    }
          buf[4] = 0x00;        // Password, not used
```

```
            buf[5] = 0x00;        // Password, not used
            buf[6] = 0x00;        // Password, not used
            buf[7] = 0x00;        // Password, not used
            buf[8] = hexdata_data.GetSize()/2;//getlength in words
            for(int m=0;m<hexdata_data.GetSize();m++)
            {
                buf[m+9] = hexdata_data.GetAt(m);
            }
            this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
            this->m_CommCtrl.SetOutput(COleVariant(hexdata));
        }
    }
```

Byte 3, namely the address, you can enter 00 or other.

Notes: the address should be entered **in words**, and the memory size is 64 byte.

Then enter the data: e0 e1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff

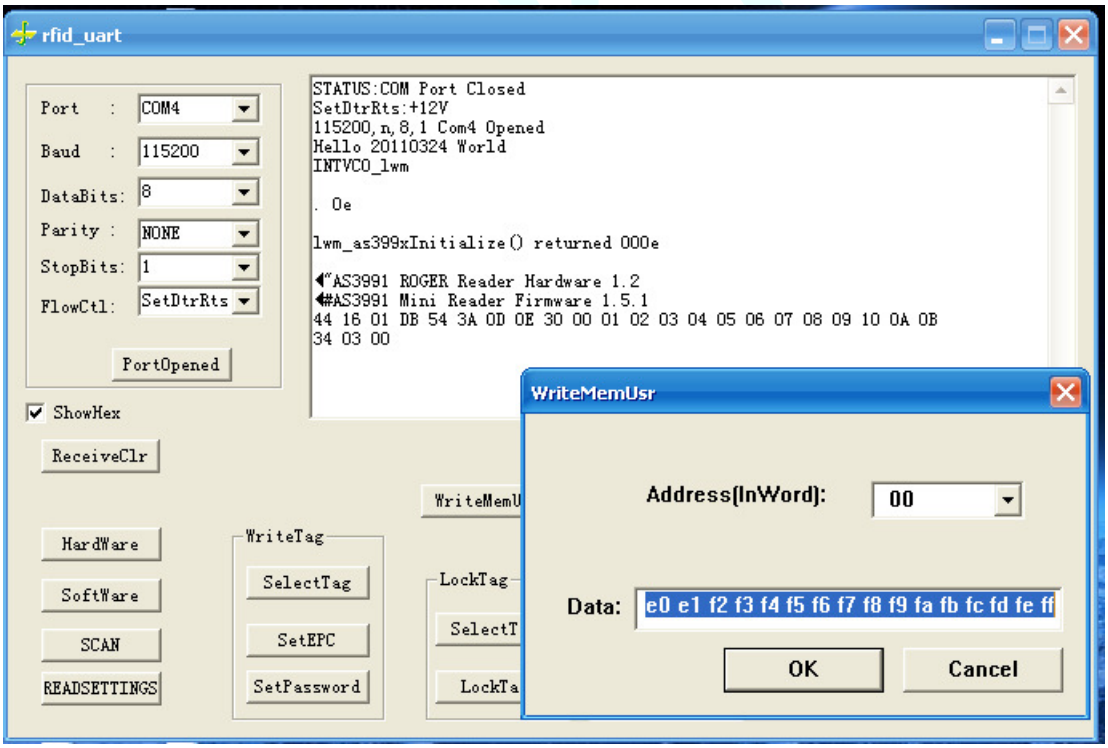

Figure 10.

Click the OK button, receive: 36 04 00 08

It means you have written 16-byte data from the address 00.

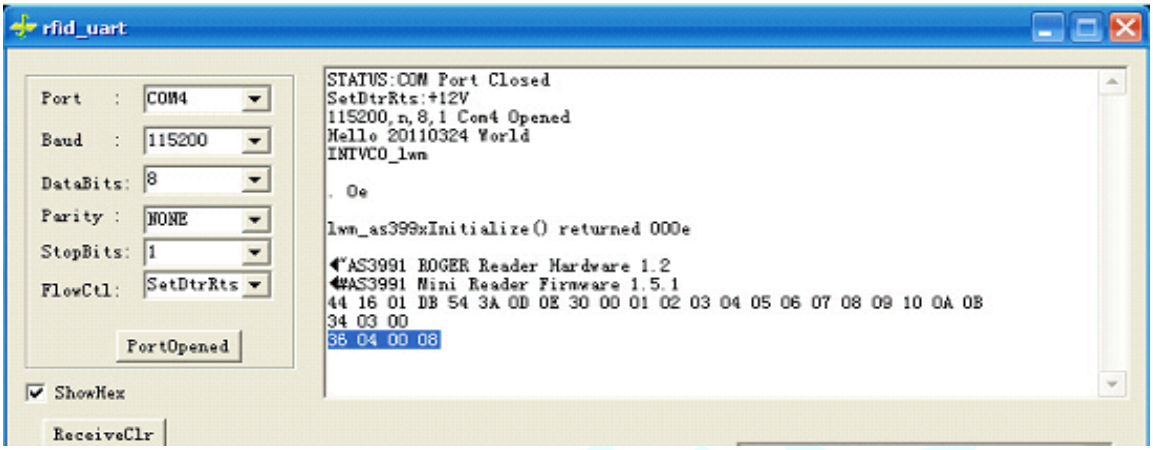

Figure 11.

After this, read the MEM_USER, you can see the data written before.

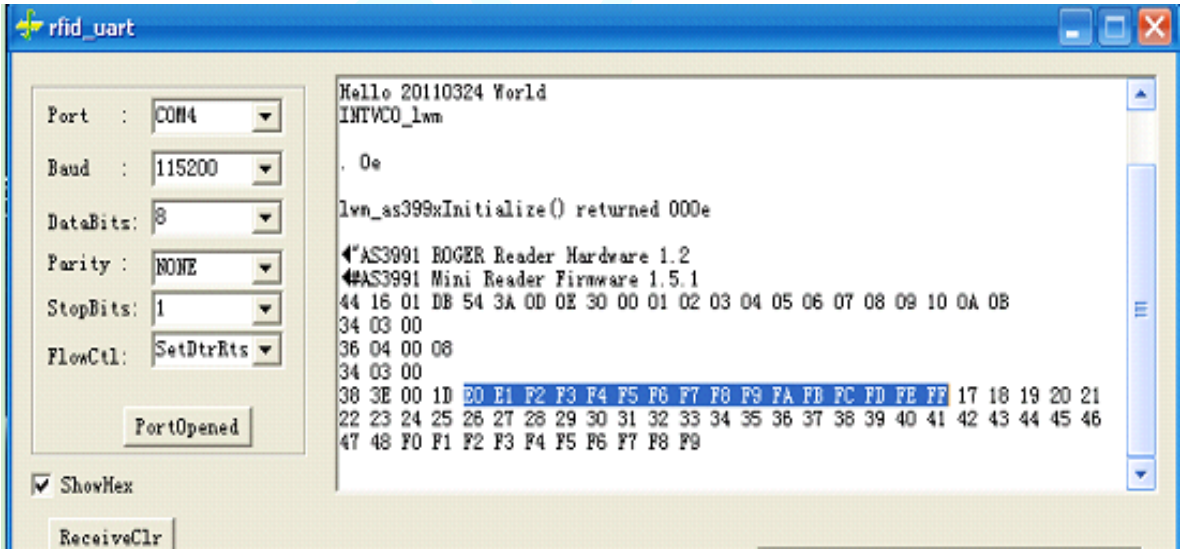

Figure 12.

6.5 Command Read from Tag Implemented

This command can be used to read data from the tag.

Command from host:

Byte 0/ID	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5,6,7,8
0x37	Frame length	Memory bank	Tag memory Address(in words)	Data length in words	ruf

Table 35: Command frame: Read from Tag from host

Command from controller:

Byte 0/ID	Byte 1	Byte 2	Byte 3	Variable
0x38	Frame length	Error byte	Data length in words	Data[n]

Table 36: Command frame: Read from Tag from controller

The error byte Information is described in Annex A.

Ec. Send: 37 05 01 02 06

Receive: 38 10 00 06 01 02 03 04 05 06 07 08 09 10 11 12

Read out the information of the tag you want. But you should select the tag first.

Now present: EPC-01 02 03 04 05 06 07 08 09 10 0A 0B

You have selected the tag successfully, now you can read it, see below:

6.5.1 Read MEM_TID

Codes are as follows:

```
void Crfid_uartDlg::OnBUTTONReadTag()
{
    // TODO: Add your control notification handler code here
    TRACE("OnButtonReadTag\n");
    BYTE buf[9];
    buf[0] = OUT_READ_FROM_TAG; // Report ID; 0x37
    buf[1] = sizeof(buf);
    buf[2] = MEM_TID; //0x02
    unsigned short address = 0;
    buf[3] = address;
    unsigned char count = 0;
    buf[4] = count;                    // Data Length
    buf[5] = 0x00;        // Password, not used
```

```
    buf[6] = 0x00;      // Password, not used
    buf[7] = 0x00;      // Password, not used
    buf[8] = 0x00;      // Password, not used
    CByteArray hexdata;
    this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
    this->m_CommCtrl.SetOutput(COleVariant(hexdata));
}
```

Receive: 38 0C 83 04 E2 00 60 03 00 BF 81 3C

Value	Meaning
0x0C	Frame length
0x83	The specified memory location does not exist or the PC value is not supported by the tag
0x04	length in words for E2 00 60 03 00 BF 81 3C
0xE2	EPC_TID_CLASS_ID_2
00 6	manufacturer,NXP
0 03	model
0x00 0xBF 0x81 0x3C	Serial Number
model	**Description**
001	UCODE EPC G2
003	UCODE G2XM
004	UCODE G2XL

The serial port debugging tool is as follows:

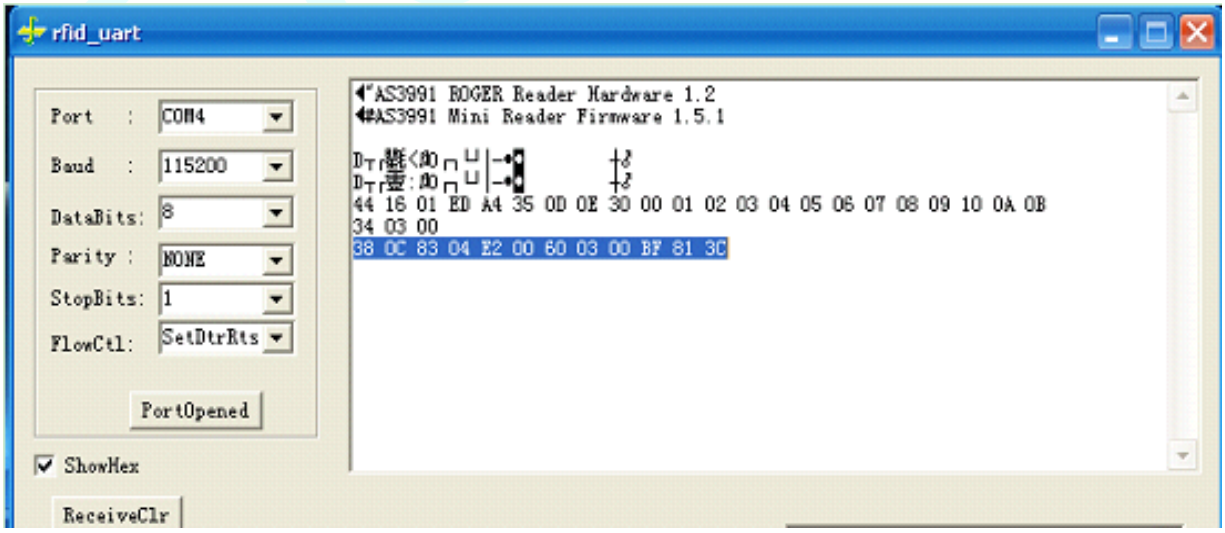

Figure 13.

6.5.2 Read MEM_EPC

Codes are as follows:

```
void Crfid_uartDlg::OnBUTTONReadTagEPC() //READ MEM_EPC
{
    // TODO: Add your control notification handler code here
    TRACE("OnButtonReadTag_MemEPC\n");
    BYTE buf[9];
    buf[0] = OUT_READ_FROM_TAG; // Report ID; 0x37
     buf[1] = sizeof(buf);
    buf[2] =   MEM_EPC; // 0x01
    unsigned short address = 0;
    buf[3] = address;
    unsigned char count = 0;
    buf[4] = count;                    // Data Length
    buf[5] = 0x00;       // Password, not used
    buf[6] = 0x00;       // Password, not used
    buf[7] = 0x00;       // Password, not used
    buf[8] = 0x00;       // Password, not used
    CByteArray hexdata;
    this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
    this->m_CommCtrl.SetOutput(COleVariant(hexdata));
}
```

Receive: 38 26 83 11 4E 08 30 00 01 02 03 04 05 06 07 08 09 10 0A 0B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Value	Meaning
0x38	Answering sends 0x37
0x26	Frame length
0x83	the specified memory location does not exist or the PC value is not supported by the tag;
0x11	Length in words for the datas behind
0x4E 0x08 0x30 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x10 0x0A 0x0B 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	The datas read out

	00	01	02	03	04	05	06	07
1	4e	08	30	00	01	02	03	04
2	05	06	07	08	09	10	0a	0b
3	00	00	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00
5	00	00						

The serial port debugging tool is as follows:

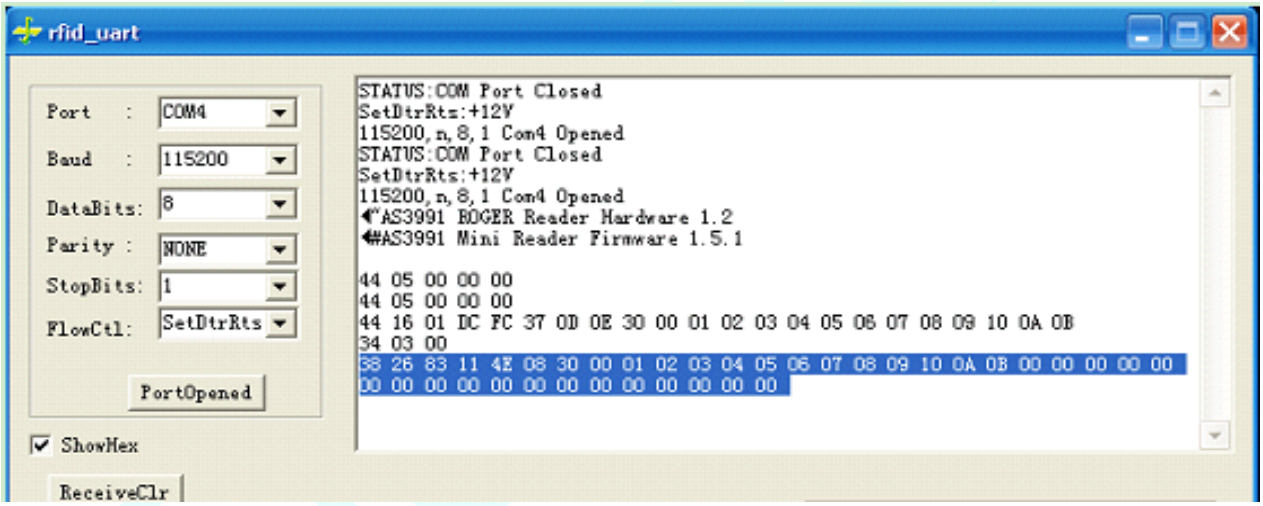

Figure 14.

6.5.3 Read MEM_USER

Codes are as follows:

```
void Crfid_uartDlg::OnBUTTONReadMemUsr()
{
    // TODO: Add your control notification handler code here
    TRACE("OnButtonReadTag_Mem_USR\n");
    if(this->m_CReadMemUsrDlg.DoModal()==IDOK)
    {
```

```
    BYTE buf[9];
    buf[0] = OUT_READ_FROM_TAG; // Report ID; 0x37
    buf[1] = sizeof(buf);
    buf[2] = MEM_USER;//        = 0x03,
    CByteArray hexdata;
    int len;
        len                                                                    =
this->m_Function.String2Hex(this->m_CReadMemUsrDlg.m_EditMemUsrAdr,hexdata);
        if(len != 1)
        {
            MessageBox("MemoryUsrAddressLength!=1!","error");
            return ;
        }
    unsigned short address;
     address = hexdata.GetAt(0);
    buf[3] = address;      //In words of the memory
    unsigned char count = 0;
    buf[4] = count;                  // Data Length
    buf[5] = 0x00;        // Password, not used
    buf[6] = 0x00;        // Password, not used
    buf[7] = 0x00;        // Password, not used
    buf[8] = 0x00;        // Password, not used
    this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
    this->m_CommCtrl.SetOutput(COleVariant(hexdata));
    }
}
```

Enter the 00 into the MEM_USR_address dialog box, click the OK button,

Receive: 38 3E 00 1D 11 22 33 44 55 66
77 88 09 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56
57 58

Value	Meaning
0x38	Answering sends 0x37
0x3E	Frame length
0x00	No error
0x1D	Length in words for the datas behind
0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x09 0x10 0x11 0x12	The datas read out

0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x30 0x31 0x32 0x33 0x34 0x35 0x36	
0x37 0x38 0x39 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58	

Select the tag, then click Read_MEM_USR button ,enter 1D(MEM_USR_address), see below:

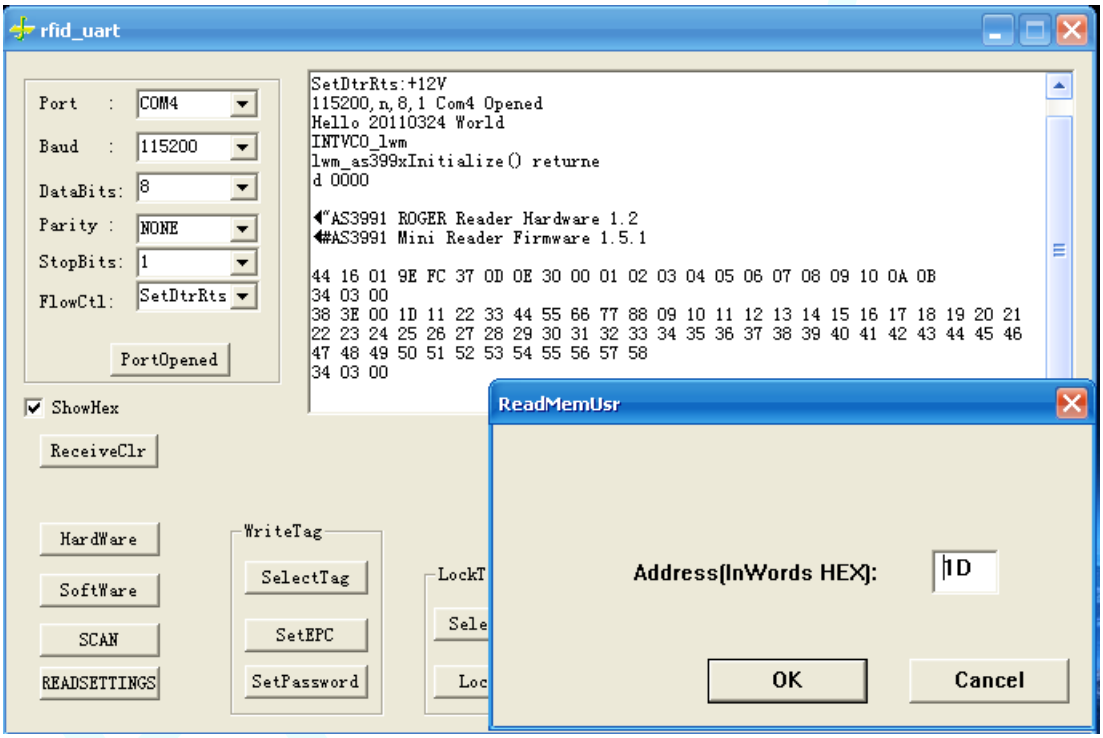

Figure15.

Receive: 38 0A 83 03 59 60 61 62 63 64

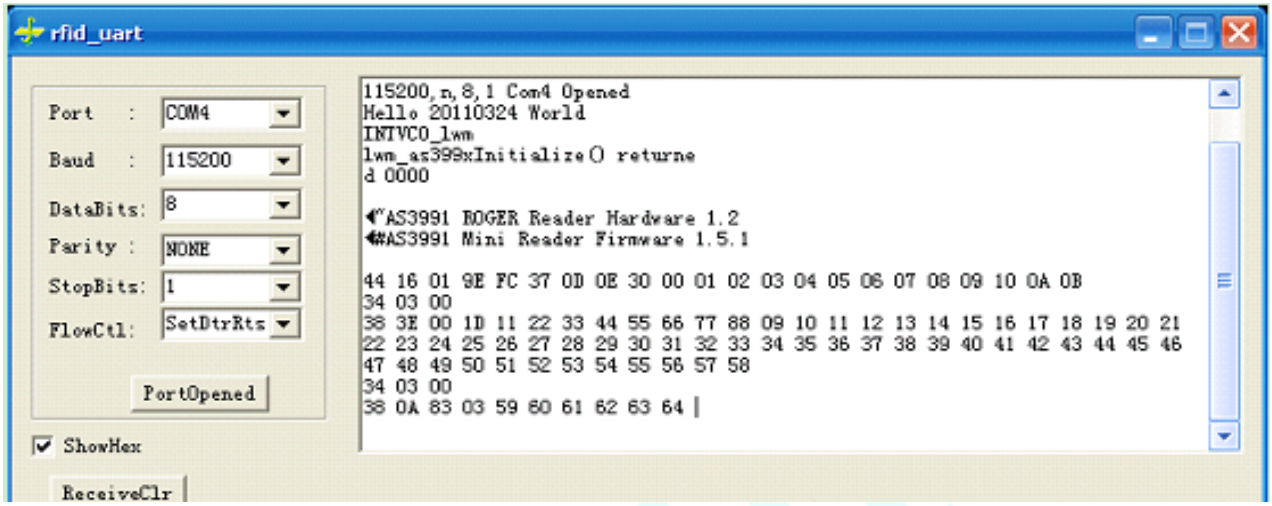

Figure 16.

User memory size: 03+1D=0x20=32words length = 64 bytes length,

The memory is:

11 22 33 44 55 66 77 88
09 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64

6.6 Command Lock Tag Memory

To lock a tag's memory address use this command.

Command from host:

Byte 0/ID	Byte1	Byte2	Byte3	Byte[4]…byte[7]
0x3B	Frame Length	Lock/unlock	Memory space	Access Password(4 bytes long)

Table 37: Command frame: Lock Tag Memory from host

Byte 2 is used to define the locking status:

value	Description
0x00	Unlock
0x01	Lock
0x02	Permalock
0x03	Lock&Permalock

Table 38: Locking/Unlocking Byte

To select the memory space, which should be locked, Byte 3 is used:

Value	**Memory space**
0x00	Kill password
0x01	Access password
0x02	EPC
0x03	TID
0x04	User

Table 39: Lock Memory Space

The controller sends back following frame:

Byte 0	Byte1	Byte 2
Report ID 0x3C	Frame length	Error byte

Table 40: Command frame: Lock Tag Memory from controller

The error byte Information is described in Annex A

You should first select the tag whose memory you want to lock, then use this command.

Ec. Send: 3B 08 01 02 11 22 33 44

Receive: 3C 04 00 00 No error.

Error message such as: 3C 04 09 00

Notes: After you have locked the tag memory, the tag can not be written or read.

Click the LockTag button, pop LockTag dialog box, enter current password, chose Lock option, see below:

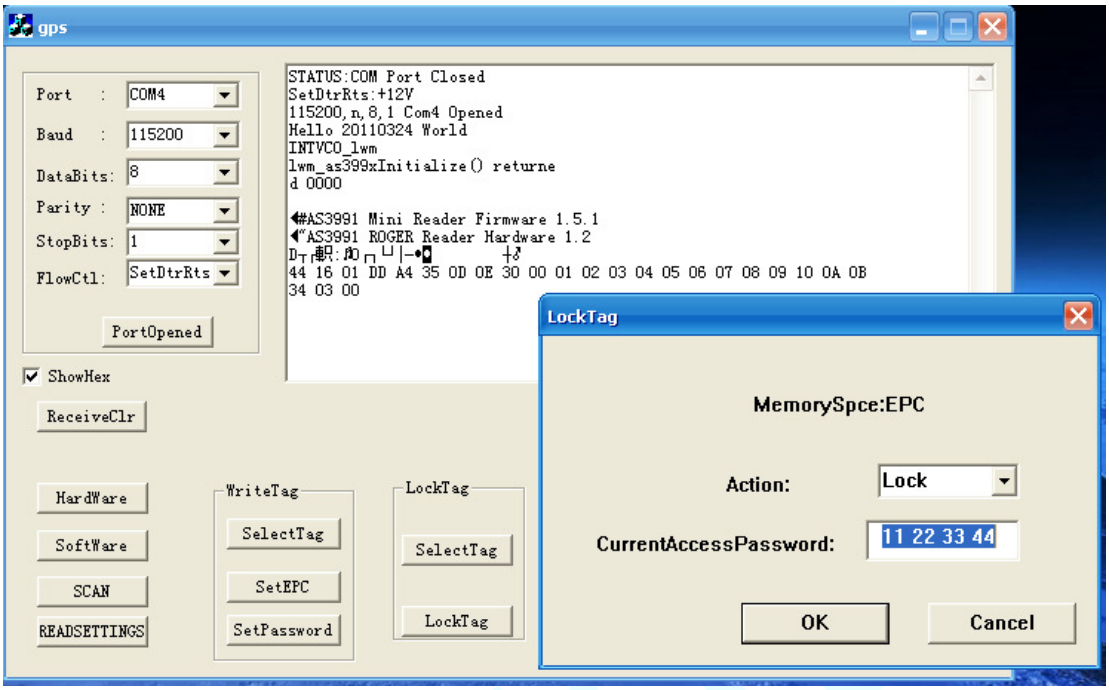

Figure 17.

Codes are as follows:

```
 void CGpsDlg::OnBUTTONLockTag()
 {
     // TODO: Add your control notification handler code here
     TRACE("LockTag\n");
     if(this->m_CLockTagDlg.DoModal()==IDOK)
     {
         int
LockMode=this->m_CLockTagDlg.m_LockStatus;//m_ComboLockStatus.GetCurSel();
         CByteArray hexdata,hexdata_Curpassword;
         int len;
         len=
this->m_Function.String2Hex(m_CLockTagDlg.m_EditCurAPassword,hexdata_Curpassword);
         if(len != 4)
         {
             MessageBox("PasswordLength!=4!","error");
             return ;
         }
          BYTE buf[64];
         buf[0] = OUT_LOCK_UNLOCK;// 0x3b
         buf[1] = sizeof(buf);
         buf[2] = LockMode;
         buf[3] = 0x02;//EPC memory space
```

```
            for(int n=0;n<hexdata_Curpassword.GetSize();n++)
            {
                buf[n+4] = hexdata_Curpassword.GetAt(n);
            }
            this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
            this->m_CommCtrl.SetOutput(COleVariant(hexdata));
        }
    }
```

6.7 Command Kill Tag

To kill a tag this command must be used.

Command from host:

Byte 0/ID	Byte 1	Byte 2 & Byte 3	Byte 4 & Byte 5	Byte 6
0x3D	Frame length	Kill Password[0…15]	Kill Password[16…32]	ruf

Table 41: Command frame Kill Tag from host

The host has to know or read the kill password and send it to the controller.
The 3 lower bits of the rfu/recom represents the rfu value
The 3 lower Bits of the high nibble represents the recom value.

Response from device:

Byte 0/ID	Byte 1	Byte 2
0x3E	Frame length	Error byte

Table 42: Command frame Kill Tag from controller

The error byte Information is described in Annex A

Notes: This command makes the label permanently disabled and protecting the privacy of its own. If you do not want to use a product or find security privacy issues, you can use the kill command to stop the chip function which prevents reading the chip illegally and improves data security. The inactivated label will be guaranteed to be inactivated in any case, it does not produce modulated signals to activate RF.

6.8 NXP User Command set

To lock a tag's memory address use this command.

Command from host:

Byte0/ID	Byte1	Byte2	Byte3	Byte[4]…byte[7]
0x45	Frame length	command	Bit status infomation	Access password (4 bytes long)

Table 43: Command frame: NXP user Commands

Byte 3 is used to define the NXP Command:

value	Description
0x01	EAS Command Bit set/reset
0x02	Read Protect Bit set/reset
0x04	EAS Alarm execute
0x08	Calibrate execute

Table 44: NXP Command Byte

To define the action of the command, Byte 2 is used. Byte 3 defines the status of the appropriate Bit in EAS command and Read protect. In case the value of Byte 3 is 0x01, the correct Bit will be set. Byte three will have no function in EAS Alarm and Calibrate.

The controller sends back following frame:

Byte 0	Byte 1	Byte2
Id 0x46	Frame length	Error byte

Table 45: Command frame: Lock Tag Memory from controller

The error byte Information is described in Annex A

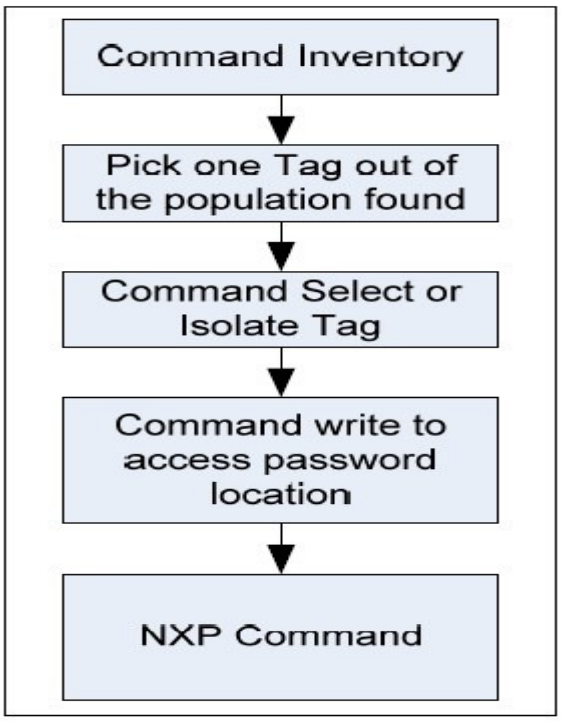

Figure18. NXP user command – flow chart

You should select the tag you want and set the access password first, then use this command.

Ec. Send: 45 08 02 01 11 22 33 44

Receive: 46 05 00 00 00 No error.

Value	Meaning
0x02	Read Protect Bit set/reset
0x01	Set the bit
0x11 0x22 0x33 0x44	Current password

Notes: The effect of this command is a switch of read protect. After set the read protect bit, the correct label number can not be read.

Now send: 43 03 01

Receive: 44 16 01 CD AC 3C 0D 0E 30 00 00 00 00 00 00 00 00 00 00 00 00 00

This shows that the EPC(01 02 03 04 05 06 07 08 09 10 0A 0B) has been read-protected successfully.So the tag number found is: 00 00 00 00 00 00 00 00 00 00 00 00

Click the SetNXP button, see below:

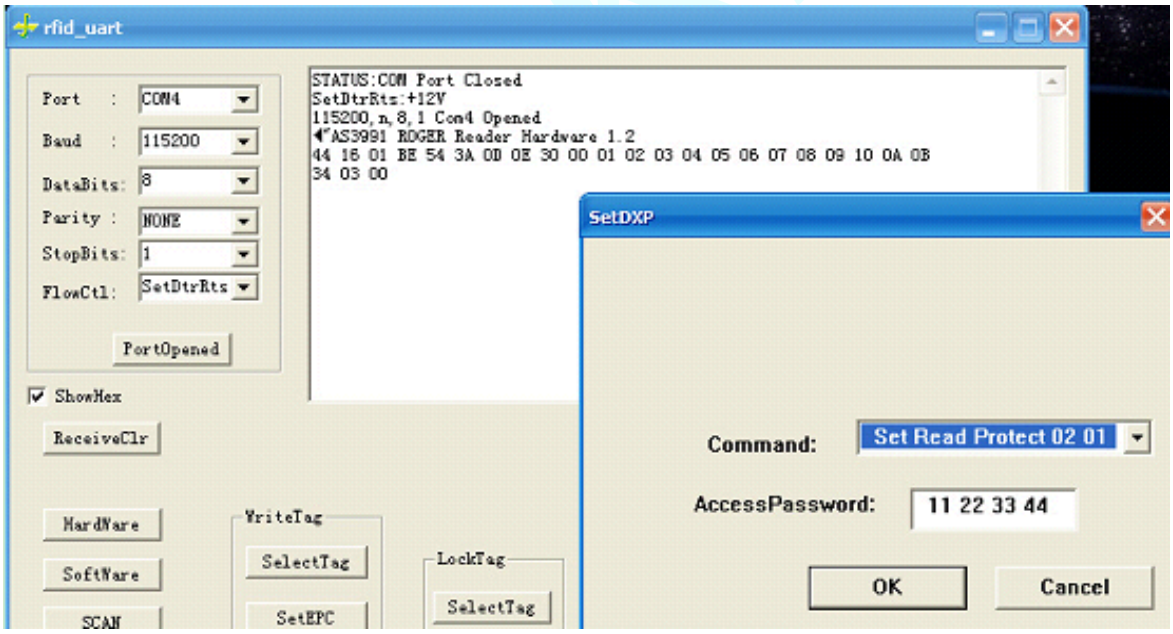

Figure 19.

Codes are as follows:

```
void Crfid_uartDlg::OnBUTTONSetNXP()
{
    // TODO: Add your control notification handler code here
    TRACE("SetNXP\n");
    if(this->m_CSetNXPDlg.DoModal()==IDOK)
```

```
    {
        CByteArray hexdata,hexdata_AccessPassword,hexdata_NXPCommand;
         int len;
        len=
this->m_Function.String2Hex(m_CSetNXPDlg.m_EditAccessPassword,hexdata_AccessPassword
);
        if(len != 4)
        {
            MessageBox("PasswordLength!=4!","error");
            return ;
        }
        len
=this->m_Function.String2Hex(m_CSetNXPDlg.m_NXPCommand,hexdata_NXPCommand);
        if(len != 2)
        {
            MessageBox("NXPCommandLength!=2!","error");
            return ;
        }
        BYTE buf[64];
            buf[0] = OUT_NXP_COMMAND;//OUT_NXP_COMMAND   = 0x45,
            buf[1] = sizeof(buf);
            buf[2] = hexdata_NXPCommand.GetAt(0);
            buf[3] = hexdata_NXPCommand.GetAt(1);
            for(int n=0;n<hexdata_AccessPassword.GetSize();n++)
            {
                buf[n+4] = hexdata_AccessPassword.GetAt(n);
            }

        this->m_Function.ByteToByteArray(buf,sizeof(buf),hexdata);
        this->m_CommCtrl.SetOutput(COleVariant(hexdata));
    }
 }
```

You can cancel the read protect, firstly select the tag (EPC:00 00 00 00 00 00 00 00 00 00 00 00), then

Send: 45 08 02 00 11 22 33 44

Receive: 46 05 00 00 00

Value	Meaning
0x02	Read Protect Bit set/reset
0x00	Reset the bit
0x11 0x22 0x33 0x44	Current password

Now the read protect is cancelled, click the SCAN button, the serial port debugging tool receives: 44 16 01 AE FC 37 0D 0E 30 00 01 02 03 04 05 06 07 08 09 10 0A 0B

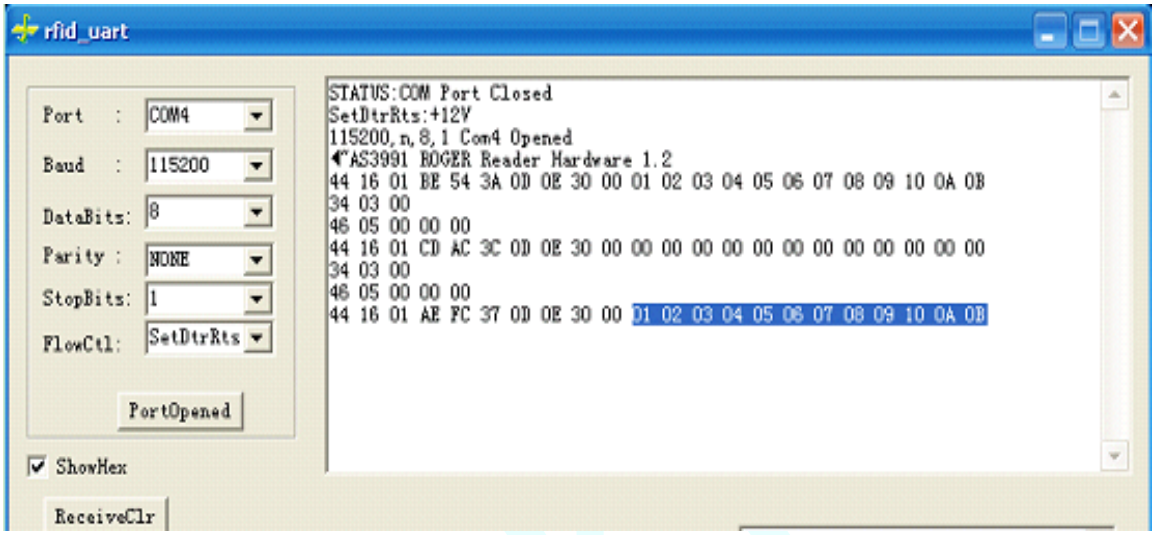

Figure 20.

This document support Roger,Mirco and Colt product

ElecKits Technoliges, Inc.
Tel： +86-0-18052482750
E-mail： sales@eleckits.com
WEB ： http://rfid.eleckits.com
Store ： http://www.eleckits.com