PM / PB MODBUS

Communication Protocol

VERSION 1.0

2007/3/29

1. Introduction:

PM / PB Microprocessor Bargraphic Display Scaling Meter is using MODBUS communication Protocol in which one computer can connected with 1~255 units of power meter.

MODBUS has two transmission modes: RTU Mode and ASCII Mode which are compatible with both PM/PB Meter. The communication is Half-Duplex. There are 9 selective transmission rates which are 600 / 1200 / 2400 / 4800 / 9600 / 14400 / 19200 / 28800 / 57600 (bps).

2. RTU (HEX) Mode

2.1 Basic structure of commands (16 Hexadecimal)

START	ADDRESS	FUNCTION	DATA	CRC	END
T1~4	8Bit	8Bit	n x 8Bit	8Bit	T1~4

(1) START ：at least 4 data bits without transmitting data.
(2) ADDRESS ：read or control the address of meter.
(address range: 1~255).
(3) FUNCTION：03H: read the data of Meter.
06H: write out the data into Meter.
(4) DATA ：Include address of the register and the number of words for read.
(5) CRC CHECK：16bit CRC，the details of calculation showing in folwoing
(6) END ：at least 4 data bits without transmitting data.

2.2 Bit Per Byte

4 modes:

Start Bit	Data Bit	Parity	Stop
1	8	None	2
1	8	Even	1
1	8	Odd	1
1	8	None	1

2.3 CRC Algorithm

Two types of CRC algorithm: **logistic** and **checksum.** CRC column is a two -16 Hexadecimal bytes, calculating from address to data end. If the CRC value from PC calculation is different from received value, then the data is an error .

(1) Logistic algorithm:

Procedures:

(1). Install a one -16 Hexadecimal bytes into FFFF (Hex), defining as CRC register.

(2).Let the CRC low 8 bytes and the 1st byte of Message to be excluded or Exclusive OR，then install the result into CRC register.

(3) Move 1 byte of the CRC register to right hand side, then set zero as the highest byte of CRC register, comparing to the removed byte and defining as SLSB.

(4) If SLSB=0, repeat procedure 3. If SLSB=1，let CRC register and parameter A001 (Hex) to be excluded, then put into CRC register.

(5) Repeat procedure 3 & 4, until complete the 8 bytes.

(6) Repeat procedure 2-5, until all Bytes to be completed.

(7) The calculated value required to be exchanged by the high/low bits.

Address	Function	Data Count	Data	Data	Data	Data	CRC Lo	CRC Hi

```
unsigned int addCRC( unsigned CRC,unsigned char b )
{
  unsigned char i,bh,bl;
  bh=CRC/256;
  bl=CRC%256;
  bl^=b;
  CRC=bh*256+bl;
  for(i=0;i<8;i++)
      if(CRC&0x0001) CRC=(CRC/2)^0xA001;
      else           CRC=CRC/2;
  return CRC;
}

unsigned char Check_CRC(void)
{ unsigned int i,CRC;
  CRC=addCRC(0xFFFF,ID);
  for(i=1;i<(TXD_CNT-2);i++) CRC=addCRC(CRC,TXDB[i]);
  return CRC;
}
```

(2) Checksum

```
static const unsigned int CRCtbl[ 256 ] = {
0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040    };
unsigned addCRC( unsigned CRC,char b )
{
    return ( CRC >> 8 ) ^ CRCtbl[ ( CRC & 0xFF ) ^ b ];
}
```

2.4 Read the Register of Meter (Function Code＝03 Hex)

Request：

Slave Address	**1 Byte**	**1~255**
Function code	**1 Byte**	**03**
Start Address	**2 Byte**	**0x0000~0xFFFF**
Quantity of Register	**1 Byte**	**1~128**
CRC Check	**2 Byte**	

Response：

Slave Address	**1 Byte**	**1~255**
Function code	**1 Byte**	**03**
Byte Count	**1 Byte**	**2 x N**
Register Value	**N x 2 Byte**	
CRC Check	**2 Byte**	

Error:

Slave	**1 Byte**	**1~255**
Error code	**1 Byte**	**0x83**
Exception code	**1 Byte**	**01 or 02 or 03**
CRC Check	**2 Byte**	

Exception code：01：Error Function

02：Error data Address

03：Error Data Value

2.5 Write out Meter Register (Function Code＝06 Hex)

Request：

Slave Address	**1 Byte**	**1~255**
Function code	**1 Byte**	**06**
Register Address	**2 Byte**	**0x0000~0xFFFF**
Register Value	**2 Byte**	**0x0000~0xFFFF**
CRC Check	**2 Byte**	

Response：

Slave Address	**1 Byte**	**1~255**
Function code	**1 Byte**	**06**
Register Address	**2 Byte**	**0x0000~0xFFFF**
Register Value	**2 Byte**	**0x0000~0xFFFF**
CRC Check	**2 Byte**	

Error:

Slave Address	**1 Byte**	**1~255**
Error code	**1 Byte**	**0x86**
Exception code	**1 Byte**	**01 or 02 or 03**
CRC Check	**2 Byte**	

Exception code：01：Error Function

02：Error Data Address

03：Error Data Value

3. ASCII MODE

3.1 Basic structure of command (16 Hexadecimal)

START	ADDRESS	FUNCTION	DATA	LRC CHECK	END
1CHAR	2CHAR	2CHAR	nCHAR	2CHAR	2CHAR CR LF

(1) .START ： Fixed as :"(3AH)。

(2) .ADDRESS ： Read or control the address of Meter(Address rang: 1~255).

(3) .FUNCTION："03": Read the data of Meter.

"06": Write out the data into Meter.

(4).DATA ：Include the address of the register and the number of word for read.

(5).LRC CHECK：8 bit LRC， the detail of calculation is shown in following chapter.

(6).END ：CR(0DH), LF(0AH)。

3.2 Bit Per Byte

9 Types:

	Data Bit	Parity	Stop
1	8	None	1
1	7	None	2
1	7	Odd	1
1	7	Even	1
1	8	None	2
1	8	Odd	1
1	8	Even	1
1	7	Odd	2
1	7	Even	2

3.3 LRC algorithm

The common LRC algorithm is Logical algorithm in which LRC column is a one-16 Hexadecimal, calculating from address to data end. If the LRC value from PC calculation is different from received value, then the data is an error.

Procedures：

(1). Install a one-8 bytes register into 00(Hex), defining as LRC register.

(2).The high bytes value of the Message to be transmitted from ASCII to Binary, then moving 4 bits to left hand side and adding low byte value from ASCII to Binary.

(3). Repeat procedure 2, until 3 bits are completed.

(4). The calculated LRC value requires of having complement 2 for transforming and installing into Message.

START	ADDRESS	FUNCTION	DATA	LRC	END
1CHAR	2CHAR	2CHAR	nCHAR	2CHAR	2CHAR CR
0	1,2	3,4	5,6,7,8	9,10	11,12

```
Unsigned char CharToBinary(char Code)
{   Unsigned char b=0xff;
    if(Code>='0' && Code<='9') b=Code-'0';      // 0-9
    else
    if(Code>='A' && Code<='F') b=Code-'A'+10; // A-F
    return b;
}

unsigned char Check_LRC(void)
{
    unsigned char i, LRC=0;
    for(i=1;i<=7;i+=2)
    LRC+=( CharToBinary(RXDB[i])<<4 + CharToBinary (RXDB[i+1]) );
    LRC =0x00-LRC;
    return LRC;
}
```

3.4 Read the register of Meter. (Function Code＝"03" ASCII)

Request：

Start	**1 Byte**	**":"**
Slave Address	**2 Byte**	**"01"~"FF"**
Function code	**2 Byte**	**"03"**
Start Address	**4 Byte**	**"0000"~"FFFF"**
Quantity of Register	**4 Byte**	**"0001"~"0040"**
LRC Check	**2 Byte**	**"XX"**
End	**2 Byte**	**CR(0x0D),LF(0xOA)**

Response：

Start	**1 Byte**	**":"**
Slave Address	**2 Byte**	**"01"~"FF"**
Function code	**2 Byte**	**"03"**
Byte Count	**2 Byte**	**"02"~"80"**
Register Value	**N x 4 Byte**	**"0000"~"FFFF"**
LRC Check	**2 Byte**	**"XX"**
End	**2 Byte**	**CR(0x0D),LF(0xOA)**

Error:

Start	**1 Byte**	**":"**
Slave Address	**2 Byte**	**"01"~"FF"**
Error code	**2 Byte**	**"83"**
Exception code	**2 Byte**	**"01" or "02" or "03"**
LRC Check	**2 Byte**	**"XX"**
End	**2 Byte**	**CR(0x0D),LF(0xOA)**

Exception code：**"**01**"**：error Function, **"**02**"**：error Data Address, **"**03**"**：error Data value.

3.5 Write out the register of Meter. (Function Code＝"06" ASCII)

Request：

Start	**1 Byte**	**":"**
Slave Address	**2 Byte**	**"01"~"FF"**
Function code	**2 Byte**	**"06"**
Register Address	**4 Byte**	**"0000"~"FFFF"**
Register Value	**4 Byte**	**"0000"~"FFFF"**
LRC Check	**2 Byte**	**"XX"**
End	**2 Byte**	**CR(0x0D),LF(0xOA)**

Response：

Start	**1 Byte**	**":"**
Slave Address	**2 Byte**	**"01"~"FF"**
Function code	**2 Byte**	**"06"**
Register Address	**4 Byte**	**"0000"~"FFFF"**
Register Value	**4 Byte**	**"0000"~"FFFF"**
LRC Check	**2 Byte**	**"XX"**
End	**2 Byte**	**CR(0x0D),LF(0xOA)**

Error:

Start	**1 Byte**	**":"**
Slave Address	**2 Byte**	**"01"~"FF"**
Error code	**2 Byte**	**"86"**
Exception code	**2 Byte**	**"01" or "02" or "03"**
LRC Check	**2 Byte**	**"XX"**
End	**2 Byte**	**CR(0x0D),LF(0xOA)**

Exception code：**"**01**"**：Error Function, **"**02**"**：Error Data Address, **"**03**"**：Error Data value.

Panel Meter ModBus Command List

Address	Function	Authority	Address	Function	Authority	Address	Function	Authority
0	VerCode	Read Only						
1	Sp1	R/W	46	Enb6	R/W	160	DspData2	Read Only
2	Sp2	R/W	47	Enb7	R/W	161	Ch2Dot	R/W
3	Sp3	R/W	48	Enb8	R/W	162	Ch2Sch	R/W
4	Sp4	R/W	49	Alr1	R/W	163	Ch2Scl	R/W
5	Sp5	R/W	50	Alr2	R/W	164	Ch2Isel	R/W
6	Sp6	R/W	51	Alr3	R/W	165	Ch2Schi	R/W
7	Sp7	R/W	52	Alr4	R/W	166	Ch2Scli	R/W
8	Sp8	R/W	53	Alr5	R/W	167	Ch2Out	R/W
9	Hon1	R/W	54	Alr6	R/W	168	Ch2Sel	R/W
10	Hon2	R/W	55	Alr7	R/W	169	Ch2Lp01	R/W
11	Hon3	R/W	56	Alr8	R/W	170	Ch2Lp02	R/W
12	Hon4	R/W	57	Nonusing		171	Ch2Lp03	R/W
13	Hon5	R/W	58	Nonusing		172	Ch2Lp04	R/W
14	Hon6	R/W	59	Nonusing		173	Ch2Lp05	R/W
15	Hon7	R/W	60	DspData1	Read Only	174	Ch2Lp06	R/W
16	Hon8	R/W	61	Ch1Dot	R/W	175	Ch2Lp07	R/W
17	Hof1	R/W	62	Ch1Sch	R/W	176	Ch2Lp08	R/W
18	Hof2	R/W	63	Ch1Scl	R/W	177	Ch2Lp09	R/W
19	Hof3	R/W	64	Ch1Isel	R/W	178	Ch2Lp10	R/W
20	Hof4	R/W	65	Ch1Schi	R/W	179	Ch2Lp11	R/W
21	Hof5	R/W	66	Ch1Scli	R/W	180	Ch2Lp12	R/W
22	Hof6	R/W	67	Ch1Out	R/W	181	Ch2Lp13	R/W
23	Hof7	R/W	68	Ch1Sel	R/W	182	Ch2Lp14	R/W
24	Hof8	R/W	69	Ch1Lp01	R/W	183	Ch2Lp15	R/W
25	Don1	R/W	70	Ch1Lp02	R/W	184	Ch2Lp16	R/W
26	Don2	R/W	71	Ch1Lp03	R/W	185	Ch2Lp17	R/W
27	Don3	R/W	72	Ch1Lp04	R/W	186	Ch2Lp18	R/W
28	Don4	R/W	73	Ch1Lp05	R/W	187	Ch2Lp19	R/W
29	Don5	R/W	74	Ch1Lp06	R/W	188	Ch2Lp20	R/W
30	Don6	R/W	75	Ch1Lp07	R/W			
31	Don7	R/W	76	Ch1Lp08	R/W			
32	Don8	R/W	77	Ch1Lp09	R/W			
33	Dof1	R/W	78	Ch1Lp10	R/W			
34	Dof2	R/W	79	Ch1Lp11	R/W			
35	Dof3	R/W	80	Ch1Lp12	R/W			
36	Dof4	R/W	81	Ch1Lp13	R/W			
37	Dof5	R/W	82	Ch1Lp14	R/W			
38	Dof6	R/W	83	Ch1Lp15	R/W			
39	Dof7	R/W	84	Ch1Lp16	R/W			
40	Dof8	R/W	85	Ch1Lp17	R/W			
41	Enb1	R/W	86	Ch1Lp18	R/W			
42	Enb2	R/W	87	Ch1Lp19	R/W			
43	Enb3	R/W	88	Ch1Lp20	R/W			
44	Enb4	R/W	89	Nonusing				
45	Enb5	R/W	90	Nonusing				