Technology Specifications

MODEL: I2C

PART NO: _____

VERSION: V1.01

Approver	Check	Design

.

■ Document Title

Content

1. Specification

The technical file is suitable for Protocol Analyzer I2C V1.01.

2. Function

1. It is available for analyzing I2C signals, whose packets mainly include WRITE, READ, ADDRESS, DATA, ACK, NACK, etc..

2. The logic is clear; the encapsulation is good. It is easy to read the code. There has been adding the notes to optimize the code, which has improved the analysis speed.

3. It can count the packet.

3. Function Lifecycle

Beginning: Set the Parameters Configuration in the Bus Property dialog box.

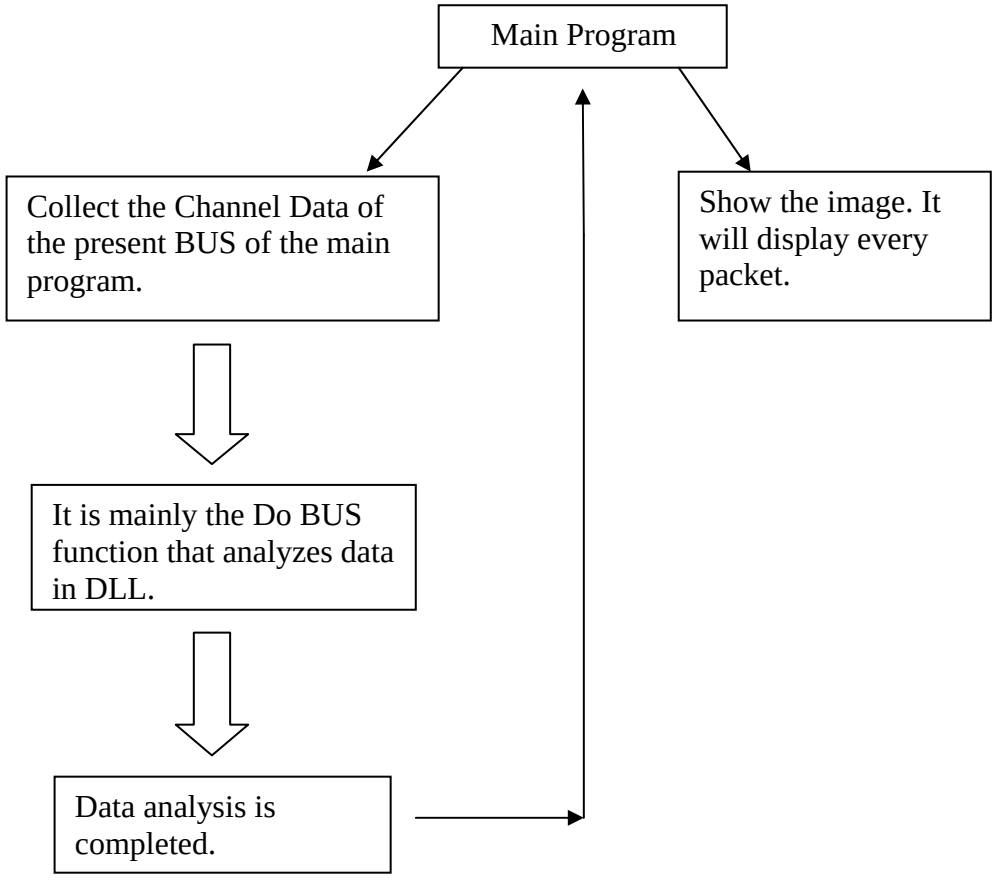
Step1 during the Settings: Set the relative module parameters in the Configuration page or set the color of the items in the Packet page.

Step2 during the Settings: Collect the user defined parameters after pressing “OK”. Call the module DLL and Do BUS function will analyze the signal.

Completion: Return to the main program which shows the analysis results with the graphic mode.

4. Flow Chart

The flow chart of I2C function refers to the below:



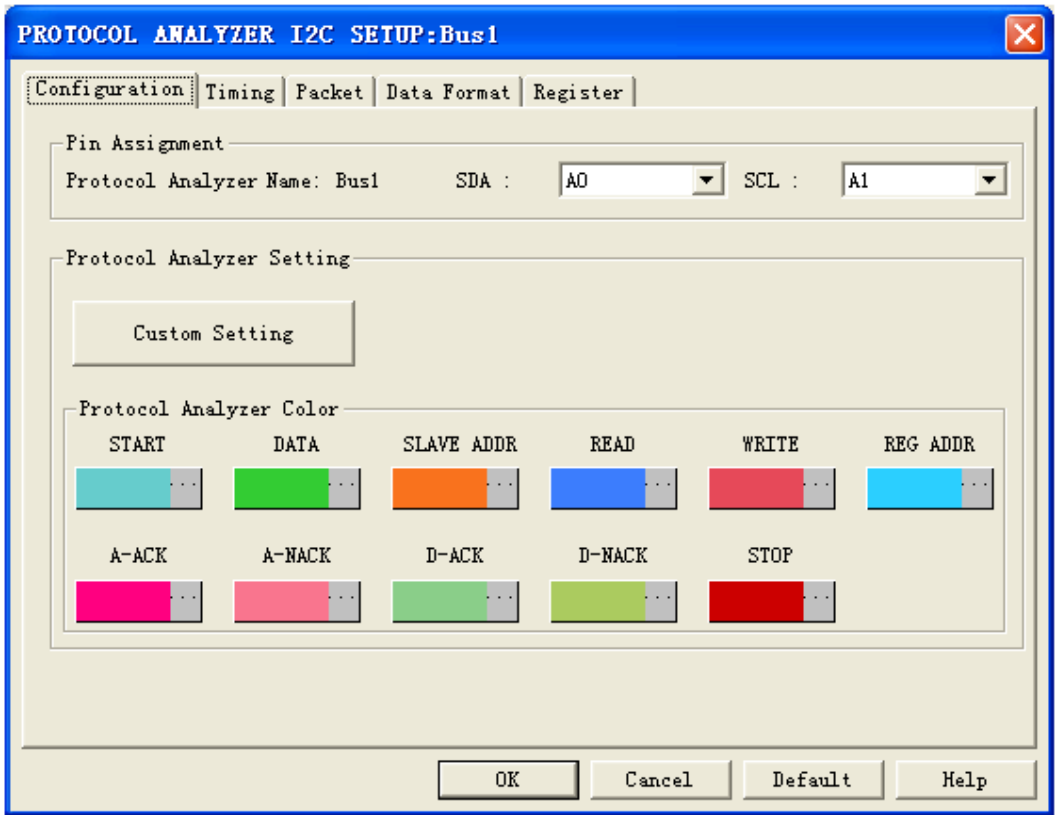
5. Design Idea

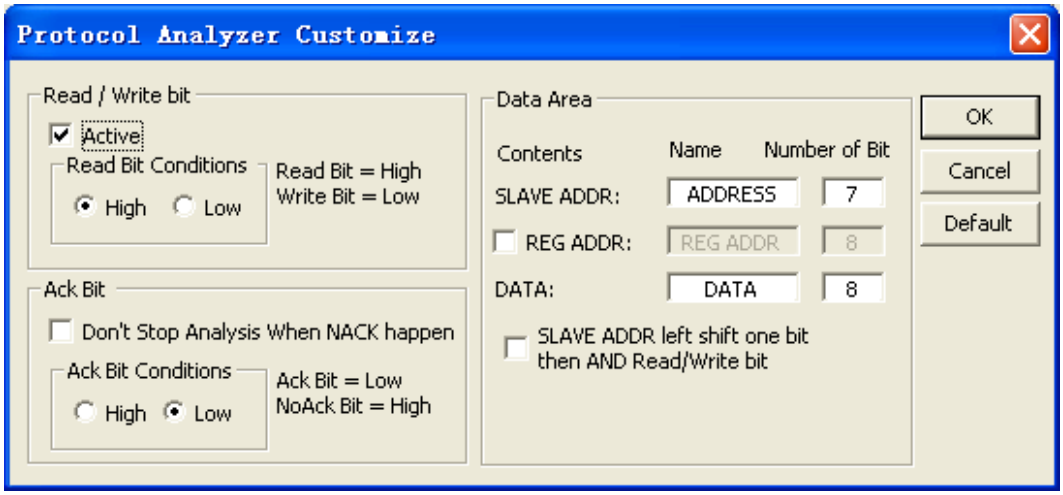
When the SDA line is at the Falling Edge and the corresponding SCL line is at the High Level, the conditions of START are satisfied. Next, it will judge the conditions to decide whether STOP is decoded. When the SDA line is at the Rising Edge and the corresponding SCL line is at the High Level, it will decode the STOP and it is ended at the Falling Edge of SCL. The bits of ADDRESS can be set according to the Configuration dialog box; the value of ADDRESS is read at the Rising Edge of SCL according to the corresponding SDA line, as well as the DATA. Then 1BIT READ/WRITE will be decoded; the value is read at the Rising Edge of SCL according to the corresponding SDA line; if the SDA line is at the High Level, it is decoded as “1” which denotes READ, otherwise it is decoded as “0” which denotes WRITE. Following is the ACK or NACK. If the previous one is READ or WRITE, it will be ACK, otherwise it is NACK. Now it is the time to decode DATA. The number of DATA is not fixed; if the abnormal conditions, such as STOP or START, do not appear, it will decode all the DATA; the condition and value of DATA are the same as that of the ADDRESS. The last is STOP.

6. Technical Tip

Nothing

7. Interface





8. Resource Explanation

Menu

…Nothing

Toolbar

…Nothing

Dialog

…IDD_IIC_SETUP, IDD_ IIC _SETUP_SI, IDD_ IIC _SETUP_TR,
IDD_ IIC _CHECK_ANSWERCODE, IDD_ IIC _CHECK_ANSWERCODE_SI,
IDD_ IIC _CHECK_ANSWERCODE_TR, IDD_PACKETITEM,
IDD_PACKETITEM_SI, IDD_PACKETITEM_TR, IDD_IIC_SETUP_CUSTOM ,
IDD_ IIC _SETUPP_CUSTOM_SI,
IDD_ IIC _SETUPP_CUSTOM_TR, IDD_IIC_RULECUSTOMIZE,
IDD_IIC_RULECUSTOMIZE_SI, IDD_IIC_RULECUSTOMIZE_TR,
IDD_IIC_TIME, IDD_IIC_TIME_SI, IDD_IIC_TIME_TR

Resourch ID

…IDC_CURSOR_HAND, IDB_PLUGIN_DATA, IDB_BITMAP_TIME

9. Application Class

class CCheckAnswerPage2: public CPropertyPage//Register Page Class
class CIICSetupPage1: public CPropertyPage//Configuration page Class
class CPacketItemPage: public CPropertyPage//Package Page Class
class CIICSetupSheet: public CpropertySheet//Sheet Dialg Class
class CEditIicBitCount: public CEdit//Control the Input of editor dialog box letters of Custom Setting
class CEditIicComponentName: public CEdit//Control the Input of editor dialog box letters of Custom Setting

10. Corresponding Arithmetic

The Arithmetic of String Calling When Switching the Language

```
CString MessageStringTable(UINT ID)
{

    CString str;
```

```
    switch(pBPOParameters->m_nLanguage)
    {
    case 1:
        str.LoadString(ID+1);      //SI
        break;
    case 2:
        str.LoadString(ID+2); //TR
        break;
    case 3:
        str.LoadString(ID); //En
        break;
    default:
        str.LoadString(ID); //En
        break;
    }
    return str;
}
```

11. Key Code

Module Decoding Function

```
bool AnalyIICData()
{

unsigned short RAISING_EDGE = 0, FALLING_EDGE = 1;
unsigned int nSize = pBPOParameters->m_nRamsize;;
unsigned int nUnknowSize = pBPOParameters->GetUnknowSize();
unsigned int pos = nUnknowSize;
busdata.Clear();
CBPOSubItem* pSubItemA = NULL;
CBPOSubItem* pSubItemB = NULL;
CBPOSubItem* pTemItem = NULL;

if(pBPOParameters->m_bDsdp)
{
if(pos < pBPOParameters->m_nDs)
{
    pos = pBPOParameters->m_nDs;
}
nSize = pBPOParameters->m_nDp;
}
m_nSize = nSize;
int subcount = 0;
short    nFlag=LABEL_UNKNOW;
int count=0;
Add(0,0,nFlag,Getclr(nFlag));

subcount = pBPOParameters->m_pSubItemAry->GetSize();
if(subcount <=1)    return false;
for(int c = 0; c < subcount; c++)
{
```

```
pTemItem = (CBPOSubItem*)pBPOParameters->m_pSubItemAry->GetAt(c);
if(pTemItem->m_nItemChannel == iicparamdata.SdaNameIndex)
{
      pSubItemA = pTemItem;
}
if(pTemItem->m_nItemChannel == iicparamdata.SclNameIndex)
{
      pSubItemB = pTemItem;
}
}
if(pSubItemA == NULL||pSubItemB == NULL)
{
pSubItemA = (CBPOSubItem*)pBPOParameters->m_pSubItemAry->GetAt(0);
iicparamdata.SdaNameIndex = pSubItemA->m_nItemChannel;

pSubItemB = (CBPOSubItem*)pBPOParameters->m_pSubItemAry->GetAt(1);
iicparamdata.SclNameIndex = pSubItemB->m_nItemChannel;
}


int    i=0,nIndex=0;
int    nAryIndex=0;
int    temp=0;
int    address_bits=iicparamdata.AddressBitCount;
int data_bits=iicparamdata.DataBitCount;
BOOL add_shift = iicparamdata.AddressLeftShift;
BOOL bNackDontCare = iicparamdata.NackDontCare;
BOOL bReadWriteActive = iicparamdata.ReadWriteActive;
int Read_Level = iicparamdata.ReadLevel;
int Write_Level = !Read_Level;
int Ack_Level = iicparamdata.AckLevel;
__int64 AddressData=0;
__int64 data=0;
int    nAddress=0;

if(pos > nSize)return false;
//-----------------------------------------------------------------s
unsigned int unknown_pos = pos;
unsigned int sda_array_size = pSubItemA->GetAryCount();
unsigned int unknown_sda_index = pSubItemA->GetAryIndexByPos(unknown_pos);
unsigned int sda_index = 0;
if( (unknown_sda_index % 2 == 1) && (unknown_sda_index + 1 < sda_array_size) )
sda_index = unknown_sda_index + 1;
else
sda_index = unknown_sda_index;

unsigned int sda_start_pos = pSubItemA->GetPosByAryIndex(sda_index);
unsigned int sda_edge_pos = sda_start_pos;
if(sda_index + 1 < sda_array_size)
```

```
        sda_index = sda_index + 1;


        unsigned int scl_array_size = pSubItemB->GetAryCount();
        unsigned int unknown_scl_index = pSubItemB->GetAryIndexByPos(unknown_pos);
        unsigned int scl_index = 0;
        if( (unknown_scl_index % 2 == 0) && (unknown_scl_index + 1 < scl_array_size) )
        scl_index = unknown_scl_index + 1;
        else
        scl_index = unknown_scl_index;

        unsigned int scl_start_pos = pSubItemB->GetPosByAryIndex(scl_index);
        unsigned int scl_edge_pos = scl_start_pos;
        if(scl_index + 1 < scl_array_size)
        scl_index = scl_index + 1;

        bool bfirst = false;
        bool bdataend = false;
        unsigned int datapos = 0;
        int sda_edge_type = -1 , scl_edge_type = -1;
        bool timebool=true;
        bool SLAVEADDRbool=false;
    __int64 SLAVEADDRData=0;
    unsigned int SLAVEADDRstartpos=0;
        bool writebool=false;// display Reg addr under the status of WRITE
        for(;pos < nSize;)
        {
        sda_edge_pos = pSubItemA->GetPosByAryIndex(sda_index);
        scl_edge_pos = pSubItemB->GetPosByAryIndex(scl_index);
        pos = min(sda_edge_pos,scl_edge_pos);
        if (pos>=nSize)      return true;//2008/1/2 modify the bug which is that the data display will be
        over the range when refreshing the DP
        if(pos == sda_edge_pos)
        {
              if(sda_index < sda_array_size)
                    sda_index = sda_index + 1;
        }
        if(pos == scl_edge_pos)
        {
              if(scl_index < scl_array_size)
                    scl_index = scl_index + 1;
        }
//-------------------------------------------------------------------------------e
        sda_edge_type = pSubItemA->GetEdgeType(pos);
        scl_edge_type = pSubItemB->GetEdgeType(pos);
        if(nFlag!=LABEL_START && sda_edge_type == FALLING_EDGE)
        {
              if(pSubItemB->GetPerSignal(pos) == 1)
              {
```

```
            if(pSubItemB->GetPerSignal(pos-1) == 1 )
            {
                  if(TRUE == DecodeStartTime(pos,pSubItemB))//START TIMING
                  {
                        nFlag=LABEL_START;
                        nAryIndex=Add(pos,nFlag,nFlag,Getclr(nFlag));
                        count=0;
                        timebool=true;
                        continue;
                  }
            }
      }
}
if(nFlag!=LABEL_STOP && sda_edge_type == RAISING_EDGE)
{
      if(pSubItemB->GetPerSignal(pos) == 1)
      {
            if(pSubItemB->GetPerSignal(pos-1) == 1)
            {
                  if( TRUE == DecodeStopTime(pos,pSubItemB) )//STOP TIMING
                  {
                        if( SLAVEADDRbool )
                        {
                              int
SLAVEADDRindex=busdata.GetBusAryByPos(SLAVEADDRstartpos);

busdata.Modify(SLAVEADDRindex,SLAVEADDRstartpos,SLAVEADDRData,LABEL_REGA
DDR,Getclr(LABEL_REGADDR));
                              SLAVEADDRbool=false;
                              SLAVEADDRData=0;
                              if( busdata.GetAryCount() > SLAVEADDRindex+1)
                              {
                                    if( busdata.GetBusFlagByAry(SLAVEADDRindex+1) ==
LABEL_DATAACK)

busdata.Modify(SLAVEADDRindex+1,busdata.GetBusPosByAry(SLAVEADDRindex+1),0,LA
BEL_ADDRACK,Getclr(LABEL_ADDRACK));
                                    else

busdata.Modify(SLAVEADDRindex+1,busdata.GetBusPosByAry(SLAVEADDRindex+1),0,LA
BEL_ADDRNACK,Getclr(LABEL_ADDRNACK));
                              }
                        }
                        nFlag=LABEL_STOP;
                        nAryIndex=Add(pos,nFlag,nFlag,Getclr(nFlag));
                        count=0;
                        //have no stop end pos
                        unsigned int idex=pSubItemB->GetAryIndexByPos(pos);
                        if((idex >= scl_array_size - 1))
```

```
                    {
                        return true;
                    }
                    //------
                    continue;
                }
                else
                {
                    nFlag=LABEL_UNKNOW;
                    nAryIndex=Add(pos,nFlag,nFlag,Getclr(nFlag));
                }
            }
        }
}
//stop end pos
if(nFlag==LABEL_STOP && scl_edge_type == FALLING_EDGE)
{
    nFlag=LABEL_UNKNOW;
    nAryIndex=Add(pos,nFlag,nFlag,Getclr(nFlag));
    count=0;
    continue;
}
// when decoding the SDA signal which is the last level, the decoding will be ended
unsigned int tmindex=pSubItemB->GetAryIndexByPos(pos);
if((tmindex >= scl_array_size - 1) && (nFlag != LABEL_DATAACK) && (nFlag !=
LABEL_DATANACK))
{
    //the end data less
    unsigned int tmpos;
    nFlag=LABEL_UNKNOW;
    tmpos = pSubItemB->GetPosByAryIndex(scl_array_size - 1);
    Add(tmpos,0,nFlag,Getclr(nFlag));
    return true;
}

if(scl_edge_type != RAISING_EDGE)continue;
int sda_value = pSubItemA->GetPerSignal(pos);
nFlag=busdata.GetBusFlagByAry(nAryIndex);
if(LABEL_START==nFlag && timebool == TRUE)
{
    if(count==0)nAddress=pos;
    if( TRUE == DecodeDataTime(pos,pSubItemB,pSubItemA) )
    {
        if( count < address_bits )
        {
            AddressData += ( sda_value * (int)pow(2,address_bits - count -1));
            count++;
        }
        if( count >= address_bits )
```

```
            {
                if(add_shift == TRUE)
                    AddressData <<= 1;
                nFlag=LABEL_ADDRESS;
                nIndex=Add(nAddress,AddressData,nFlag,Getclr(nFlag));
                nAryIndex=nIndex;
                AddressData=0;
                count=0;
                timebool=true;
            }
            continue;
        }
        else
        {
            timebool=false;
            nFlag=LABEL_UNKNOW;
            Add(nAddress,0,nFlag,Getclr(nFlag));
        }
}
if(bReadWriteActive && LABEL_ADDRESS==nFlag && timebool == TRUE )
{
        if(add_shift)
        {
            __int64 temp=busdata.GetBusPerData(nAddress);
            temp = temp | (__int64)sda_value;
            busdata.SetBusAryData(nIndex, temp);
        }
        if( TRUE == DecodeDataTime(pos,pSubItemB,pSubItemA) )
        {
            if(sda_value == Read_Level)
            {
                nFlag=LABEL_READ;
                nAryIndex=Add(pos,1,nFlag,Getclr(nFlag));
                data=0;
                count = 0;
            }
            else
            {
                nFlag=LABEL_WRITE;
                nAryIndex=Add(pos,0,nFlag,Getclr(nFlag));
                data=0;
                count = 0;
                if(iicparamdata.m_bSpecflag8)
                    writebool=true;
            }
            timebool=true;
            continue;
        }
        else
```

```
        {
            timebool=false;
            nFlag=LABEL_UNKNOW;
            Add(pos,0,nFlag,Getclr(nFlag));
        }
}
//Address Ack
bool bCondition = false;
if(bReadWriteActive == TRUE)
        bCondition = ( nFlag == LABEL_READ || nFlag == LABEL_WRITE );
else
        bCondition = ( nFlag == LABEL_ADDRESS );

if(bCondition == true && timebool == TRUE)
{
        if( TRUE == DecodeDataTime(pos,pSubItemB,pSubItemA) )
        {
            if(sda_value==Ack_Level)
            {
                nFlag=LABEL_ADDRACK;
                nAryIndex=Add(pos,0,nFlag,Getclr(nFlag));
                temp=sda_value;
                data=0;
                count = 0;
            }
            else
            {
                nFlag=LABEL_ADDRNACK;
                nAryIndex=Add(pos,0,nFlag,Getclr(nFlag));
                temp=sda_value;
                data=0;
                count = 0;
            }
            timebool=true;
            continue;
        }
        else
        {
            timebool=false;
            nFlag=LABEL_UNKNOW;
            Add(pos,0,nFlag,Getclr(nFlag));
        }
}

if( (LABEL_ADDRACK==nFlag || LABEL_ADDRNACK==nFlag ||
        LABEL_DATAACK==nFlag || LABEL_DATANACK==nFlag) && timebool == TRUE )
{
       if(count==0)nAddress=pos;
        if(TRUE == DecodeDataTime(pos,pSubItemB,pSubItemA))
```

```
    {
        if(temp == Ack_Level || bNackDontCare)
        {
            if(count==0)
            {
                nAddress=pos;
                bdataend = true;
                datapos = pos;
            }
            if( iicparamdata.m_bSpecflag8 && writebool )
            {
                SLAVEADDRstartpos=nAddress;
                // judge whether there is a STOP at the end
                bool stopbool=false;
                int bitcount=0;
                int SDAindex=pSubItemA->GetAryIndexByPos(SLAVEADDRstartpos);
                while( 1 )
                {
                    unsigned int curpos=pSubItemA->GetPosByAryIndex(SDAindex);
                    if( curpos >= nSize)
                        break;
                    if(pSubItemB->GetPerSignal(curpos) == 1 &&
pSubItemA->GetEdgeType(curpos) == RAISING_EDGE)
                    {
                        if( TRUE == DecodeStopTime(curpos,pSubItemB) )//STOP
TIMING
                        {
                            stopbool=true;
                            break;
                        }
                    }
                    SDAindex++;
                }

                if( stopbool )
                    bitcount=iicparamdata.m_nSpecNumber8;
                else
                {
                    bitcount=data_bits;
                    SLAVEADDRbool=false;
                }
                if( count < bitcount )
                {
                    SLAVEADDRData += ( sda_value * (int)pow(2,bitcount - count -1));
                    count++;
                }
                if( count >= bitcount    )
                {
                    nFlag=LABEL_DATA;
```

```
nAryIndex=Add(SLAVEADDRstartpos,SLAVEADDRData,nFlag,Getclr(nFlag));

                        count=0;
                        bdataend = false;
                        timebool=true;
                        writebool=false;
                        SLAVEADDRbool=true;
                    }
                }
                else
                {
                    if( count < data_bits )
                    {
                        data += ( sda_value * (int)pow(2,data_bits - count -1));
                        count++;
                    }
                    if( count >= data_bits )
                    {
                        nFlag=LABEL_DATA;
                        nAryIndex=Add(nAddress,data,nFlag,Getclr(nFlag));

                        data=0;
                        count=0;
                        bdataend = false;
                        timebool=true;
                    }
                }
            }
            continue;
        }
        else
        {
            timebool=false;
            nFlag=LABEL_UNKNOW;
            Add(nAddress,0,nFlag,Getclr(nFlag));
        }
}

if(LABEL_DATA==nFlag && timebool == TRUE )
{
    if(TRUE == DecodeDataTime(pos,pSubItemB,pSubItemA))
    {
        if(sda_value==Ack_Level)
        {
            nFlag=LABEL_DATAACK;
            nAryIndex=Add(pos,0,nFlag,Getclr(nFlag));
            data=0;
            count = 0;
```

```
            }
            else
            {
                nFlag=LABEL_DATANACK;
                nAryIndex=Add(pos,0,nFlag,Getclr(nFlag));
                data=0;
                count = 0;
            }
            timebool=true;
            continue;
        }
        else
        {
            timebool=false;
            nFlag=LABEL_UNKNOW;
            Add(pos,0,nFlag,Getclr(nFlag));
        }
    }
    }

    if(bdataend)
    {
    //the end data less
    nFlag=LABEL_UNKNOW;
    if(nAddress < nSize)
    {
        Add(nAddress,0,nFlag,Getclr(nFlag));
    }
    }

    return true;
}
```