Plug-in Developer

Plug-in overview

Write a Plug-in

The Protocol Analyzer Plug of the Logic Analyzer is a Plug-in of enhancing the Analysis Function of Protocol Analyzer of Logic Analyzer. The Protocol Analyzer Plug-in of Protocol Analyzer is supplied by way of the windows DLLs with 32 bits.

The Plug-in of Protocol Analyzer is a protocol analyzer decoder. When you start the decoder, the Protocol Analyzer Data of Logic Analyzer will be inputted to the decoder; after the decoder completes the data decoding, it will output the decoded data to the Logic Analyzer for displaying and analyzing. Our Company has designed more than seventy protocol analyzer modules for customers, such as I2C, SPI, UART, CAN, USB and SSI. You can refer to the website of our Company to learn the more detailed information. If the supplied Protocol Analyzers did not meet your needs, you can refer to this file to design the desired decoder by yourself.

All the Analysis Plug-in of Protocol Analyzer use the Microsoft Visual C++ 6.0 or higher edition to develop and design with the modes of C or C＋＋. Please don’t use the Microsoft Visual Basic, JAVE or other program languages to develop.

Tools

All the Protocol Analyzer Plug-ins use the Microsoft Visual C++ 6.0 or higher edition to develop and design with the modes of C or C＋＋. (We recommend that you use the Visual Studio 6.0. Because our program codes are developed by the Visual Studio 6.0, we can’t make sure that the program codes can be compiled normally in other higher edition.). Please don’t use the Microsoft Visual Basic, JAVE or other program languages to develop.

Content

Ⅰ API Function Introduction

This Chapter will be divided into three parts, which will introduce the API Functions of the CBUSDATA, CBUSDATAARRAY and CBPOPARAMTERS.

Part 1 Interface Function between the main program and module

The API Function which is introduced in this part is the Interface Function between the main program and module, which is very important. We can know the data communication mode between the main program and module through understanding it.

The API of this part is defined in the busdata.cpp, and the enablement system is also in the busdata.cpp file. The busdata.cpp is the main-body file of the module. We should combine the source code of the module with the API, which can help us quickly understand the use of API Function.

//==========

1. VidPid

The main program can set the Vid and Pid of the module. Vid and Pid are the identification sign of LA.

bool VidPid(
unsigned short vid,// Logic Analyzer Hardware vid
unsigned short pid // Logic Analyzer Hardware pid
);

Parameters
Vid: It is the manufacturer code of Logic Analyzer.
Pid: It is the model code of Logic Analyzer.

Return Values
When the Return Values are TRUE, the setup is successful; when the Return Values are FALSE, the setup is failed.
//==========

2. Supportcheck

Check the module whether it supports the main program. The main program will introduce a version No., and the module can analyze the version No. to decide whether it supports or not. The module needs to do another two things in this Function. The parameters of protocol analyzer are set as the default, and check the VID and PID to make sure whether the module needs to be registered in this model.

bool supportcheck
(
struct _DataBusModule *this_mod,// Module Index
int nVersion // Module Version
);

Parameters
this_mod: It is the Module Index.
nVersion: It is the supporting module version of the main program.

Return Values
When the Return Values are TRUE, the setup is successful; when the Return Values are FALSE, the setup is failed.

//==========

3. Config

Open the User Interface of Module

```
void Config(
struct _DataBusModule *this_mod
);
```

Parameters

Return Values
Without Return Values
//==========

4. Init

Init is an Initialization Function of Module, where the main program will transmit the necessary data into the module.

```
int Init(
struct _DataBusModule *this_mod,
void* BPOP
);
```

Parameters
BPOP: It is the CBPOParamters Index; the main program will transmit the necessary data of module decoding into the module by the CBPOParamters Index.

Return Values
The return value, 0, indicates that the implementation of the Function is successful.
//==========

5. DoBus

It is a Module Decoding Function; the module implements the decoding process through the DoBus and completes the decoding of the protocol analyzer.

```
bool DoBus(
struct _DataBusModule *this_mod,
void* pObject
);
```

Parameters
pObject: It is not used; the condition is always NULL.

Return Values
When the Return Values are TRUE, the setup is successful; when the Return Values are FALSE, the setup is failed.
//==========

6. CleanBus

Clean the Array data of Bus data; the Bus data consist of four arrays. This Function will clean the data of the four arrays at the same time.

```
void CleanBus(
struct _DataBusModule *this_mod
);
```

Parameters

Return Values
Without Return Values
//==========

7. Quit

It is the Quit Function of Module; it can be ignored.

```
void Quit(
struct _DataBusModule *this_mod
);
```

Parameters

Return Values
//==========

8. SetParamBuf

Copy the decoded parameter of Module to the appointed BUF; according to some Functions, the main program can get the module parameter; it also can return the parameter to the module through these Functions.

Void* SetParamBuf(
void *pbuf,
int size
);

Parameters
pbuf: It is the ectypal object memory index.
size: It is the ectypal memory size.

Return Values

//==========

9. GetParamBuf

It is not used.

void* GetParamBuf(
int &size
);

Parameters

Return Values
//==========

10. SetBusTriParamBuf

It is not used.

void SetBusTriParamBuf(
void *pbuf,
int size
);

Parameters

Return Values

//==========

11. GetBusTriParamBuf

It is not used.

```
void* GetBusTriParamBuf(
int &size
);
```

Parameters

Return Values

//==========

12. GetUnitCount

Get the total Basic Data Unit of Module (For example: We take the Start and Address as the Basic Data Units in the Protocol Analyzer I2C.)

```
int
GetUnitCount();
```

Parameters

Return Values

The Return Value must be more than zero.

//==========

13. GetUnit

Get the Data of the appointed Data Unit; each Data Unit consists of four data.

```
void GetUnit
```

(
int nIndex,
int &nFlag,
COLORREF &clr,
char *name,
unsigned int &bitcount
);

Parameters
nIndex: It is the Index value of Data Unit Array; it is an input parameter.
nFlag: It is the FLAG of Data Unit; it is an output parameter.
clr: It is the color of Data Unit; it is an output parameter.
name: It is the string of Data Unit; it is an output parameter.

Return Values
Without Return Values
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

14. CheckSerial

SN Check

bool CheckSerial
(
char *sn
);

Parameters
sn: It is the SN string of Logic Analyzer Hardware; it is an input parameter.

Return Values
When the Return Values are TRUE, the implementation is successful; when the Return Values are FALSE, the implementation is failed.
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

15. PlugEnableUse

If it is necessary to register, this Function will check whether the module has been registered or not.

bool PlugEnableUse();

Parameters

Return Values
If the Return Values are TRUE, it indicates that the module has been registered or can be used for free; if the Return Values are FALSE, it means that the module needs to register; it isn't registered at present.
//==========

16. LeastSingalNum

Get the least channel number for decoding module; the main program can confirm whether the channel number meets the requirement of decoding.

int LeastSingalNum();

Parameters

Return Values
The Return Value must be more than zero.
//==========

17. BusListUnitShow

Get whether the appointed Data Unit needs to display in the Packet list.

bool BusListUnitShow
(
short nFlag, Input Parameters
);

Parameters
nFlag: FLAG It is the FLAG of Packet Unit.

Return Values
When the Return Values are TRUE, the Data Unit will be displayed in the Packet list; when the Return Values are FALSE, the Data Unit won't be displayed in the Packet list.
//==========

18. BusListUnitName

Get the string of the appointed Data Unit.

```
bool BusListUnitName
(
short nFlag,
char *name
);
```

Parameters
nFlag: It is the FLAG of Data Unit; it is an input parameter.
name: It is the string of Data Unit; it is an output parameter.

Return Values
When the Return Values are TRUE, the implementation of the Function is successful; when the Return Values are FALSE, the implementation of the Function is failed.
//==========

19. BusListUnitClr

Get the color of the appointed Data Unit.

```
bool BusListUnitClr
(
short nFlag,
COLORREF &clr
);
```

Parameters
nFlag: It is the FLAG of Data Unit; it is an input parameter.
clr name: It is the color of Data Unit; it is an output parameter.

Return Values
When the Return Values are TRUE, the implementation of the Function is successful; when the Return Value are FALSE, the implementation of the Function is failed.
//==========

20. BusListPackStart

Check whether the appointed Data Unit is the start unit of the Packet.

```
bool BusListPackStart(
short nFlag
);
```

Parameters
nFlag: It is the FLAG of Data Unit; it is an input parameter.

Return Values
When the Return Values are TRUE, it is the start unit; when the Return Values are FALSE, it isn't the start unit.
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

21. BusListPackEnd

Check whether the appointed Data Unit is the end unit of the Packet.

```
bool BusListPackEnd(
short nFlag
);
```

Parameters
nFlag: It is the FLAG of Data Unit; it is an input parameter.

Return Values
When the Return Values are TRUE, it is the end unit; when the Return Values are FALSE, it isn't the end unit.

//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

22. BusListDesShow

Whether to show the description of the Packet or not.

```
bool BusListDesShow();
```

Parameters

Return Values
When the Return Values are TRUE, it shows the description of the Packet; when the Return Values are FALSE, it doesn't show the description of the Packet.
//==========

23. BusListDesFlag

Get the descriptive string according to the Flag.

```
bool BusListDesFlag(
short nFlag,
char *name
);
```

Parameters
nFlag: It is the Flag of the Packet. According to the different Packets, we can get the relative and descriptive string; it is an input parameter.
name: It is the descriptive string; it is an output parameter.

Return Values
When the Return Values are TRUE, the implementation of the Function is successful; when the Return Values are FALSE, the implementation of the Function is failed.
//==========

24. Reserve0

The Return Value of Reserve8 is TRUE denotes that it supports the Memoey Analyzer function. At that time, if the Return Value of Reserve9 is FALSE, this Function will return the Start Flag; if the Return Value of Reserve9 is TRUE, it will return the total number of the Start Flag.

```
int Reserve0(void)
{
    return LABEL_START;
}
```

25. Reserve1

The Return Value of Reserve8 is TRUE denotes that it supports the Memoey Analyzer

function.. At that time, if the Return Value of Reserve9 is FALSE, this Function will return the Address Flag; if the Return Value of Reserve9 is TRUE, it will return the total number of the Address Flag.

```
int Reserve1(void)
{
    return LABEL_ADDRESS;
}
```

26. Reserve2

The Return Value of Reserve8 is TRUE denotes that it supports the Memoey Analyzer function. At that time, if the Return Value of Reserve9 is FALSE, this Function will return the Read Flag; if the Return Value of Reserve9 is TRUE, it will return the total number of the Read Flag.

```
int Reserve2(void)
{
    return LABEL_READ;
}
```

27. Reserve3

The Return Value of Reserve8 is TRUE denotes that it supports the Memoey Analyzer function. At that time, if the Return Value of Reserve9 is FALSE, this Function will return the Write Flag; if the Return Value of Reserve9 is TRUE, it will return the total number of the Write Flag.

```
int Reserve3(void)
{
    return LABEL_WRITE;
}
```

28. Reserve4

The Return Value of Reserve8 is TRUE denotes that it supports the Memoey Analyzer function. At that time, if the Return Value of Reserve9 is FALSE, this Function will return the Data Flag; if the Return Value of Reserve9 is TRUE, it will return the total number of the Data Flag.

```
int Reserve4(void)
```

```
{
    return LABEL_DATA;
}
```

29. Reserve5

It is used to control the function of the Memory Analyzer, and the Function will return the bit length of the Address Flag.

```
int Reserve5(int value)
{
    int length=8; // address
    return length;
}
```

30. Reserve6

It is used to control the display of Serial Number of Data; this Function will return the Data Flag.

```
int Reserve6(int value)
{
        Return LABEL_DATA;
}
```

31. Reserve7

It is used to control the function of the Packet Filter; this Function will return the total number of ToolBar; ”0” indicates there is not a ToolBar.

```
int Reserve7(int value)
{
         return 1;
}
```

32. Reserve8

It is used to control the function of the Memory Analyzer. TRUE indicates that the function of the Memory Analyzer is supported; FALSE indicates that the function of the Memory Analyzer is not supported.

```
int Reserve8(void)
{
        return true;
}
```

33. Reserve9

It is used to control the function of the Memory Analyzer. As there are some special conditions in some modules, for example, there are many Start and Address, if the Return Value of this Function is TRUE, it indicates that the main program support the special conditions of the function of the Memory Analyzer; if the Return Value of this Function is FALSE, it indicates that the main program support the normal conditions of the function of the Memory Analyzer.

```
int Reserve9(void)
{
        return true;
}
```

34. Reserve10

It is used to control the function of the Memory Analyzer. If the Return Value of Reserve9 is TRUE; it denotes the special condition and returning the relative parameters of the special conditions.

```
void* Reserve10(void* pdata)
{       pMSParameters = (CMSParamters*)pdata;
        //START 1
        pMSParameters->m_objMSStartAry[0]=LABEL_START;
        //ADDRESS 2
        pMSParameters->m_objMSAddressAry[0]=LABEL_ADDRESS;
        pMSParameters->m_objMSAddressAry[1]=LABEL_REGADDR;
        // READ 1
        pMSParameters->m_objMSReadAry[0]=LABEL_READ;
```

```
        //WRITE 1
        pMSParameters->m_objMSWriteAry[0]=LABEL_WRITE;
        //DATA
        pMSParameters->m_objMSDataAry[0]=LABEL_DATA;

        pMSParameters->m_nReserve1=41;
        if( iicparamdata.m_bSpecflag8 )
        pMSParameters->m_nReserve2=2;
        else
          pMSParameters->m_nReserve2=1;
        pdata=(void*)pMSParameters;

        return pdata;
}
```

35. Reserve12

It is used to control the SDK function.

```
void* Reserve12(void* pdata)
{
          void *pKey = (void*)szKey;
          return pKey;
}
//==========================================
```

36. Reserve13

The Function can be used to store the number of the parameters which is displayed by way of Data Format of the Protocol Analyzer.

```
void* Reserve13 (void* pdata)
{
}
//==========================================
```

37. Reserve14

It is used to control the function of the user-defined Data Format of Protocol Analyzer; this Function can store the Flag whose Data Format needs to be changed.

```
void* Reserve14(void* pdata)
{
}
//==========================================
```

Part 2 API Functions, which supply the parameters for BUS data creation, BUS data operation and Bus data conversion.

The main functions of API Function, which is introduced in this part, are supplying the parameters for BUS data creation, BUS data operation and Bus data conversion. The API in this part is very important, because the final aim of the module decoding is to set up a data array to store the BUS decoding according to the protocol, and the data array is the final data that we require, then the data will be sent back to the main program.

The definition of API in this part is in the class CbusDataArray. The class packs four arrays that consist of the stored BUS data, and we only need to put the decoded result into it. API offers the integrated Function to the operation of BUS data array, and we will use it in the process of decoding continually.

//==========

1. RemoveBusPosByAry

The Pos Index of the Bus Data Storage Array can be used to delete the Pos of BUS unit data

Bool RemoveBusPosByAry
(
int nAryIndex
);

Parameters

nAryIndex: It is the index of Bus Data Storage Array, and users input the appointed parameter.

Return Values

When the Return Values are TRUE, the data is deleted successfully; When the Return Values are FALSE, the data fails to be deleted.

//==========

2. RemoveBusDataByAry

The Data Index of the Bus Data Storage Array can be used to delete the Data of BUS unit data

Bool RemoveBusDataByAry
(
int nAryIndex
);

Parameters

nAryIndex: It is the index of Bus Data Storage Array; and users input the appointed parameter.

Return Values

When the Return Values are TRUE, the data is deleted successfully; When the Return Values are FALSE, the data fails to be deleted.

//==========

3. RemoveBusFlagByAry

The Flag Index of the Bus Data Storage Array can be used to delete the Flag of BUS unit data

Bool RemoveBusFlagByAry(
int nAryIndex
);

Parameters
nAryIndex: It is the index of Bus Data Storage Array , and users input the appointed parameter.

Return Values
When the Return Values are TRUE, the data is deleted successfully; When the Return Values are FALSE, the data fails to be deleted.
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

4. RemoveBusClrByAry

The CIr Index of the Bus Data Storage Array can be used to delete the CIr of BUS unit data

Bool RemoveBusClrByAry(
int nAryIndex
);

Parameters
nAryIndex: It is the index of Bus Data Storage Array , and users input the appointed parameter.

Return Values
When the Return Values are TRUE, the data is deleted successfully; When the Return Values are FALSE, the data fails to be deleted.
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

5. Overflow

It is the tip of Overflow information which is used to debug.

Bool Overflow();

Parameters

Return Values

Return to TRUE
//==========

6. Add

Add a new unit at the end of the BUS Data Array.

Int Add(
unsigned int nAddress,
__int64 nValue,
short nFlag,
COLORREF clr);

Parameters
nAddress: The start address of the data unit; users input the appointed Parameter.
nValue: The value of the data unit; users input the appointed Parameter.
nFlag: The FLAG of the data unit; users input the appointed Parameter.
clr: The color of the data unit; users input the appointed Parameter.

Return Values
Return a number more than 0, which is used to show the index value of the new data unit.
//==========

7. Delete

Delete the data unit （Address, Data, Flag, Clr）according to the index value of the BUS data array.

Bool Delete(
int nAryPos
);

Parameters
The index value of the data array

Return Values
When the Return Values are TRUE, the data is deleted successfully; When the Return Values are FALSE, the data fails to be deleted.

//==========

8. Read

Read the value of the data unit according to the index value of the BUS data array.

Bool Read(

int nAryPos,
unsigned int &nPos,
__int64 &nData,
short &nFlag,
COLORREF &clr);

Parameters
nAryPos: The index value of the BUS data array; users input the appointed parameter.
nPos: The starting Address of the data unit that the index value indicates; it is an output parameter.
nData: The Data of the data unit that the index value indicates; output a parameter for users.
nFlag: The Flag of the data unit that the index value indicates; output a parameter for users.
clr: The color value of the data unit that the index value indicates; output a parameter for users.

Return Values
When the Return Values are TRUE, the data is read successfully; When the Return Values are FALSE, the data fails to be read.
//==========

9. Modify

Modify the value of the data unit by the index value of the BUS data array.

Int Modify(
int nAryIndex,
unsigned int nAddress,
__int64 nValue,
short nFlag,
COLORREF clr);

Parameters
nAryIndex: The index value of the BUS data array; users input the appointed parameter.
nAddress: The starting Address of the data unit that the index value indicates; users input the appointed parameter.
nValue: The Data of the data unit that the index value indicates; users input the appointed parameter.
nFlag: The Flag of the data unit that the index value indicates; users input the appointed parameter.
Clr: The color value of the data unit that the index value indicates; users input the appointed parameter.

Return Values
When the Return Values are TRUE, the data is modified successfully; When the Return Values are FALSE, the data is modified unsuccessfully.
//==========

10. Clear

Clear the data of the BUS data array.

Bool Clear();

Parameters

Return Values
When the Return Values are TRUE, the data is cleared successfully; When the Return Values are FALSE, the data is cleared unsuccessfully.
//==========

11. GetAryCount

Get the unit count of the BUS data array.

Int GetAryCount();

Parameters

Return Values
Return a value more than 0.
//==========

12. BinarySearch

Binary Search means searching the address belongs to which unit range of the BUS data array.

Int BinarySearch(
unsigned int nValue,
int nLow,
int nHigh,
int nLimit);

Parameters
nValue: The address value; users input the appointed parameter.
nLow: The index value of the min. BUS unit which belongs to the current search range.

nHigh: The index value of the max. BUS unit which belongs to the current search range.
nLimit: The index value of the max. BUS unit of the search range.

Return Values
Return a number more than 0, which is an index value of the BUS unit.
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

13. GetBusAryByPos

Get the index value whose address belongs to the unit of the BUS data array according to the address.

Int GetBusAryByPos(
unsigned int nPos
);

Parameters
nPos: Address data (0～Ramsize); users input the appointed parameter.

Return Values
Return a number more than 0, which is an index value of the BUS unit.
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

14. GetBusPerPos

Get the starting address of the BUS unit that the address belongs to according to the address.

unsigned int GetBusPerPos(
unsigned int nPos
);

Parameters
nPos: Address; users input the appointed parameter.

Return Values
Return a value between 0 and Ramsize, which is the starting address of the BUS unit.
//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

15. GetBusPerData

Get the Data of the BUS unit that the address belongs to according to the address.

__int64 GetBusPerData(
unsigned int nPos
);

Parameters
nPos: Address; users input the appointed parameter.

Return Values
Return the Data of the BUS unit.
//===

16. GetBusPerFlag

Get the Flag of the BUS unit that the address belongs to according to the address.

Short GetBusPerFlag(
unsigned int nPos
);

Parameters
nPos: Address; users input the appointed parameter.

Return Values
Return the Flag of the BUS unit.
//===

17. GetBusPerClr

Get the Color of the BUS unit that the address belongs to according to the address.

COLORREF GetBusPerClr(
unsigned int nPos
);

Parameters
nPos: Address; users input the appointed parameter.

Return Values
Return the Color of the BUS unit.
//===

18. GetBusPosByAry

Get the starting address of the data unit according to the index value of the BUS unit.

unsigned int GetBusPosByAry(
int nAryIndex
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.

Return Values
Return the starting address of the data unit.
//==========

19. GetBusDataByAry

Get the Data of the data unit according to the index value of the BUS unit.

__int64 GetBusDataByAry(
int nAryIndex
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.

Return Values
Return the Data of the data unit.
//==========

20. GetBusFlagByAry

Get the Flag of the data unit according to the index value of the BUS unit.

Short GetBusFlagByAry(
int nAryIndex
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.

Return Values
Return the Flag of the data unit.
//==========

21. GetBusClrByAry

Get the Color of the data unit according to the index value of the BUS unit.

COLORREF GetBusClrByAry(
int nAryIndex
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.

Return Values
Return the Color of the data unit.
//==========

22. SetBusAryPos

Modify the starting address of the data unit according to the index value of the BUS unit.

Bool SetBusAryPos(
int nAryIndex,
unsigned int nPos
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.
nPos: The starting address of the data unit; users input the appointed parameter.

Return Values
When the Return Values are TRUE, the data is modified successfully; When the Return Values are FALSE, the data is modified unsuccessfully.
//==========

23. SetBusAryData

Modify the Data of the data unit according to the index value of the BUS unit.

Bool SetBusAryData(
int nAryIndex,
__int64 nData
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.
nData: The Data of the data unit; users input the appointed parameter.

Return Values
When the Return Values are TRUE, the data is modified successfully; When the Return Values are FALSE, the data is modified unsuccessfully.
//==========

24. SetBusAryFlag

Modify the Flag of the data unit according to the index value of the BUS unit.
Bool SetBusAryFlag(
int nAryIndex,
short nFlag
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.
nFlag: The Flag of the data unit; users input the appointed parameter.

Return Values
When the Return Values are TRUE, the data is modified successfully; When the Return Values are FALSE, the data is modified unsuccessfully.
//==========

25. SetBusAryClr

Modify the Color of the data unit according to the index value of the BUS unit.

Bool SetBusAryClr(
int nAryIndex,
COLORREF clr
);

Parameters
nAryIndex: The index value of the BUS unit; users input the appointed parameter.
clr: The Color of the data unit; users input the appointed parameter.

Return Values
When the Return Values are TRUE, the data is modified successfully; When the Return Values are FALSE, the data is modified unsuccessfully
//==========

26. GetBusPosArySize

Get the size of the Address array of the BUS data array.

Int GetBusPosArySize();

Parameters

Return Values
Return a number more than 0, which is the size of the Address array.
//==========

27. GetBusDataArySize

Get the size of the Data array of the BUS data array

Int GetBusDataArySize();

Parameters

Return Values
Return a number more than 0, which is the size of the Data array.
//==========

28. GetBusFlagArySize

Get the size of the Flag array of the BUS data array.

Int GetBusFlagArySize();

Parameters

Return Values
Return a number more than 0, which is the size of the Flag array.
//==========

29. GetBusClrArySize

Get the size of the Color array of the BUS data array.

Int GetBusClrArySize();

Parameters

Return Values
Return a number more than 0, which is the size of the Color array.
//==========

30. MatchLastFlag

//Confirm whether the input FLAG is the same as the FLAG of the final data unit in the array in order to confirm whether the same data unit appears continuously.

```
Bool MatchLastFlag(
short nFlag
);
```

Parameters

nFlag: The Flag of the data unit.

Return Values

When the Return Values are TRUE, the input flag is the same as the flag of the final data unit of the current array; when the Return Values are FALSE, the input flag is different from the flag of the final data unit of the current array.

//＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

Part 3 API Functions, which provide the functions for the operation and the conversion of channel data of the Logic Analyzer

The main function of API Function which is introduced in this part is to provide the functions for the operation and the conversion of channel data for the Logic Analyzer. This part of API is very important, you can flexibly operate each data of the channel through understanding this part; the channel data is the basis of BUS data. The operation of channel data is also the basis of decoding BUS data. Only each data of the channel is flexibly operated, and then it is helpful for quick decoding BUS data.

The API is defined in the class CBPOSubItem and the class CBPOParamters. CBPOSubItem packs the stored array of channel data and the API Function of array operation. CBPOParamters packs the arrays of all the channel objects and a little API Functions. But the CBPOParamters packs other data, which should be understood carefully.

//==========

1. GetAryCount

Get the unit count of channel data array.

Int GetAryCount();

Parameters

Return Values
The Return Value must be more than zero; it is the unit count of the channel data array.
//==========

2. GetFirstData

Get the index, the signal and the total length of its data unit according to address.

unsigned int GetFirstData(
unsigned int nPos,
int &nArrayPos,
bool &bSignal,
unsigned int &nTotal
);

Parameters
nPos: It is the address.
nArrayPos: It is the index of data unit of address.
bSignal： It is the signal value of this unit. The TRUE value indicates the high level; the FALSE value indicates the low level.
nTotal: It is the total length of this unit.

Return Values
The Return Value is the length from the input address to the next data unit, which is more than zero.
//==========

3. GetNextData

Get the signal value and total length of the next data unit of the appointed index.

unsigned int GetNextData(
int nArray,
bool &bSignal
);

Parameters
nArray: It is the index value of data unit.
bSignal: It is the signal value of the next unit of the appointed index. The TRUE value indicates the high level; the FALSE value indicates the low level.

Return Values
Return the total length of the next data unit of the appointed index, which is more than zero.
//==========

4. GetPreData

Get the signal value and total length of the preceding data unit of the appointed index.

unsigned int GetPreData(
int nArray,
bool &bSignal
);

Parameters
nArray: It is the index value of data unit.
bSignal: It is the signal value of the preceding unit of the appointed index. The TRUE value indicates the high level; the FALSE value indicates the low level.

Return Values
Return the total length of the preceding data unit of the appointed index, which is more than zero.
//==========

5. GetCurData

Get the signal value and total length of the current data unit of the appointed index.

unsigned int GetCurData(
int nArray,
bool &bSignal
);

Parameters
nArray: It is the index value of data unit.
bSignal: It is the signal value of this unit of the appointed index. The TRUE value indicates the high level; the FALSE value indicates the low level.

Return Values
Return the total length of the data unit, which is more than zero.
//==========

6. GetAryIndexByPos

Get the index value of its data unit according to address.

```
int GetAryIndexByPos(
unsigned int nPos
);
```

Parameters
nPos: It is the address.

Return Values
Return the unit index of the input address.
//==========

7. GetPosByAryIndex

Get the start address of the data unit according to the index value of data unit.

```
unsigned int GetPosByAryIndex(
int nArrayIndex
);
```

Parameters
nArrayIndex: It is the index value of data unit.

Return Values
Return the start address of the data unit.
//==========

8. GetPosByPos

Get the start address of its data unit according to address.

```
unsigned int GetPosByPos(
unsigned int nPos
);
```

Parameters
nPos: It is the address.

Return Values
Return the start address of the unit of the input address.
//==========

9. GetPerData

Read the signal value of the address according to address.

bool GetPerData(
unsigned int nPos,
bool &bSignal
);

Parameters
nPos: It is the address.
bSignal: It is the signal value of address. The TRUE value indicates the high level; the FALSE value indicates the low level.

Return Values
When the Return Values are TRUE, the implementation of the Function is successful; when the Return Values are FALSE, the implementation of the Function is failed.
//==========

10. GetPerSignal

Read the signal value of the address according to address.

int GetPerSignal(
unsigned int nPos
);

Parameters
pos: It is the address.

Return Values
When the Return Value is“1”, the signal value is high level; when the return value is “0”, the signal value is low level.

//==========

11. GetEdgeType

Make sure whether the address is the signal edge or not according to the input address.

int GetEdgeType(
unsigned int pos
);

Parameters
pos: It is the address.

Return Values
When the Return Value is“0”, it is the RISING edge; when the return value is “1”, it is the FALLING edge; when the Return Value is “-1”, it is not the edge.
//==========

12. RisingEdge

Make sure whether the input address is the RISING edge or not.

```
bool RisingEdge(
unsigned int pos
);
```

Parameters
pos: It is the address.

Return Values
When the Return Value is TRUE, it is the RISING edge; when the Return Value is FALSE, it isn’t the RISING edge.
//==========

13. FallingEdge

Make sure whether the input address is the FALLING edge or not.

```
bool FallingEdge(
unsigned int pos
);
```

Parameters
pos: It is the address.

Return Values
When the Return Value is TRUE, it is the FALLING edge; when the Return Value is FALSE, it isn’t the FALLING edge.
//==========

14. BinarySearch

Search the unit index of the channel array according to the input address.

```
int BinarySearch(
```

unsigned int nValue,
int nLow,
int nHigh,
int nLimit
);

Parameters
nValue: It is the address.
nLow: It is the min. value of the current search range.
nHigh: It is the max. value of the current search range.
nLimit: It is the max. value of the search range.

Return Values
Return the input address to search the unit index of the channel array.
//==========

15. FreeArray

It is not used.

void FreeArray();

Parameters

Return Values
//==========

16. GetUnknowSize

Get the size of the unknow waveform.

int GetUnknowSize();

Parameters

Return Values
Return the size of the unknow waveform.
//==========

Ⅱ Definition Rule of the Bus Packet Flag

It is known that every Bus Packet consists of 4 data, which are ADDRESS, DATA, FLAG, CLR（COLOR）. Thereinto, Flag is a short data, and the Flag in a Bus must be unique. So the Flag is the only ID of the Packet.

The Bus Packet Flag is divided into 5 segments, and every segment has different meaning. When designing the Bus Packet Flag, the designer should define the Packet FLAG according to the actual requirements.

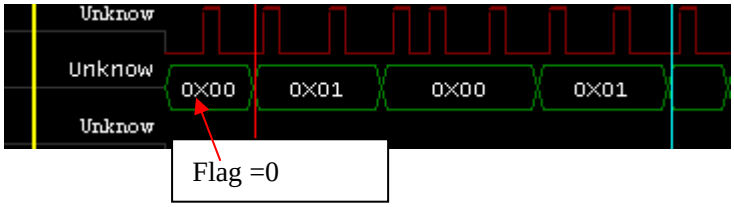
Segment 1: Common Segment (0~99)

Descriptions：

0: Bus Data (Binary, Decimal, Hexadecimal, ASCII)

1: Protocol Analyzer Packet Unknow

2: The Bus Packet doesn't display the data.



Segment 2: Bus Packet Strings（1000-1999）

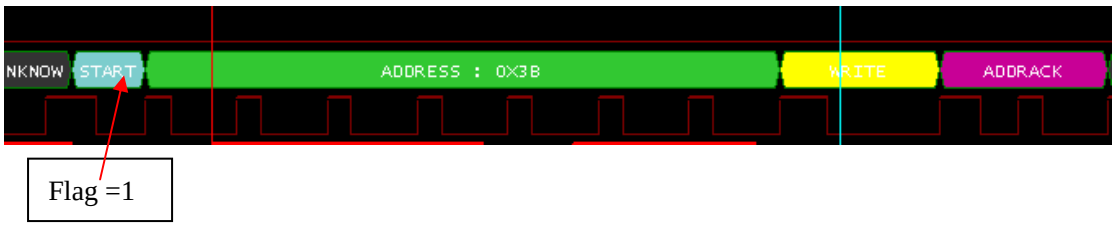
Description: Bus Packet only displays the string.

Segment 3: Bus Packet Strings（2000-2999）

Description: Bus Packet displays the string and data.

Segment 4: Protocol Analyzer Packet Strings（3000-3899）

Description: Protocol Analyzer Packet displays the string.

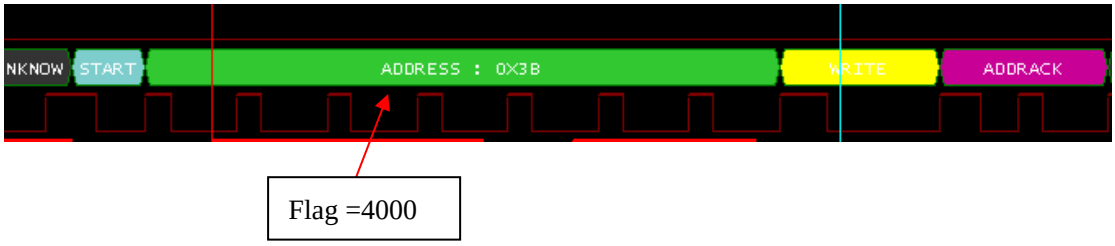


Segment 5: Protocol Analyzer Packet Strings（3900-3999）

Description: The Error Packet of the Protocol Analyzer

Segment 6: Protocol Analyzer Packet Strings（4000-4999）

Description: Protocol Analyzer Packet displays the string and data.



Ⅲ Formation of Channel Data and Bus Data

(I) The channel data is the most basic data; how many channels the logic analyzer has will bring about the same amount channel data.

1. Ramsize:

The data length of every channel is related to RAM setting. For example, if the RAM is 2K, each channel can bring about 2048 signal data. That is to say, Ramsize is 2048.

2. Ary:

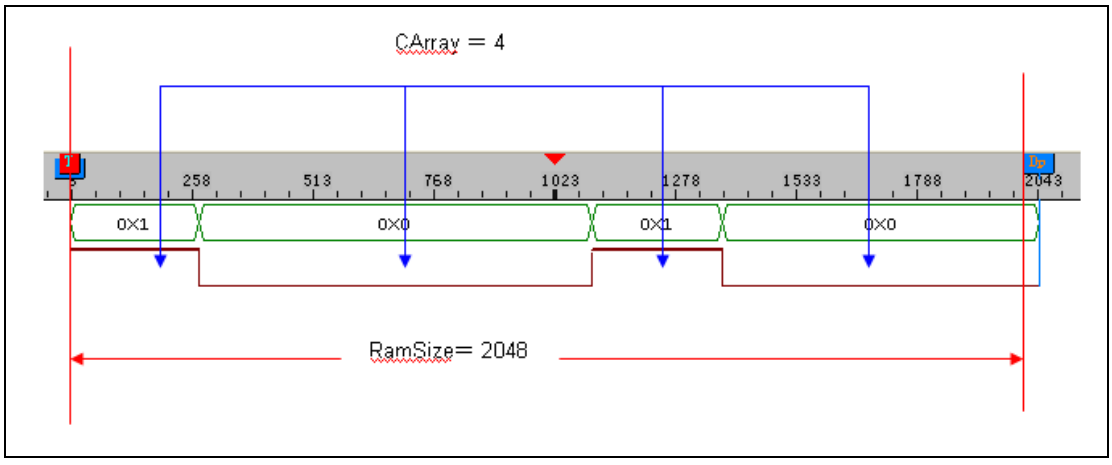
We learn the Ramsize of the data of each channel, but the software makes the conversion of the data when the data is dealt with by the software. So we know that the Ramsize data of each channel consists of high or low waveform data. However, there is too much repetitive data. It is unnecessary for us to develop too many RAM for main program. We only need to register the conversion dot of the waveform so that we can save a lot of RAM.

We will allocate one CArray to each channel to store those data. And Ary means any unit in CArray.

3. Pos:

Pos is relative to the Ramsize, that is to say, Pos means any data in Ramsize.

The relations of the above three are specified by the below image:



Ramsize Size = 2048, CArray Size ＝ 4

Class CBPOSubItem packs the CArray data of each channel and supplies the conversion Function between CArray data (Ary) and Ramsize (Pos).

(II) The Bus Data is gained by the operation of channels, and the Bus can be divided into two styles, Bus and Protocol Analyzer.

1. Ramsize:

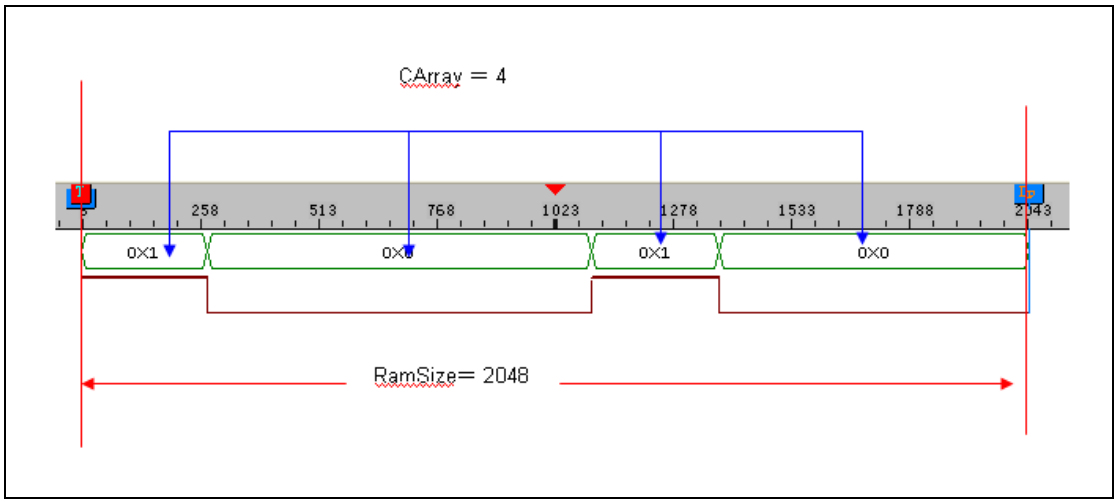
The data length of every channel is related to RAM setting. For example, if the RAM

is 2K, each channel can bring about 2048 signal data. That is to say, Ramsize is 2048.

2. Ary:
The basic unit of each Bus is Packet, and every Packet consists of 4 data, Address, Data, Flag and Clr. We allocate 4 CArrays to each Bus to store Address, Data, Flag and Clr. The 4 Arys keep the same size, increasing or decreasing synchronously. Ary means any unit data in CArray. If we want to get the data of one Packet, we must get 4 values of the same Ary in the 4 CArray.

3. Pos:
Pos is relative to the Ramsize, that is to say, Pos means any data in Ramsize.

The relations of the above three are specified by the below image:



Ramsize Size = 2048, CArray Size ＝ 4

Class CBusDataArray packs every Bus data, and supplies the conversion Function between CArray data (Ary) and Ramsize (Pos).