AN-1138
APPLICATION NOTE

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

LINB DLL Programmer´s Guide

by Holger Grothe

INTRODUCTION

This application note describes the library functions available in the LINBWSD.dll library. These functions can be used to create an USB-to-LIN downloader for integrated battery sensor devices. LINBWSD.dll uses Protocol 6 for Flash/EE memory programming via LIN. Protocol 6 is explained in detail in Application Note AN-946. LINBWSD.dll can be used for LIN programming with the following IBS devices:

- ADuC7032-8L
- ADuC7033
- ADuC7036DCPZ
- ADuC7039

DISCLAIMER

All LINBWSD library code provided by Analog Devices, Inc., including this file, is provided as is without warranty of any kind, either expressed or implied. Users assume all risk from the use of this code. It is the responsibility of the user integrating this code into an application to ensure that the resulting application performs as required and is safe.

LIN DOWNLOADER

Figure 1. LIN Downloader

TABLE OF CONTENTS

REVISION HISTORY

1/12—Revision 0: Initial Revision

HARDWARE SETUP

DOWNLOADER JUMPER SETTINGS

The downloader board allows two interfaces to be connected, LIN and I²C. The LIN functionality of the interface is determined by the on-board jumper connections, J5 and J6, as shown in Figure 2.

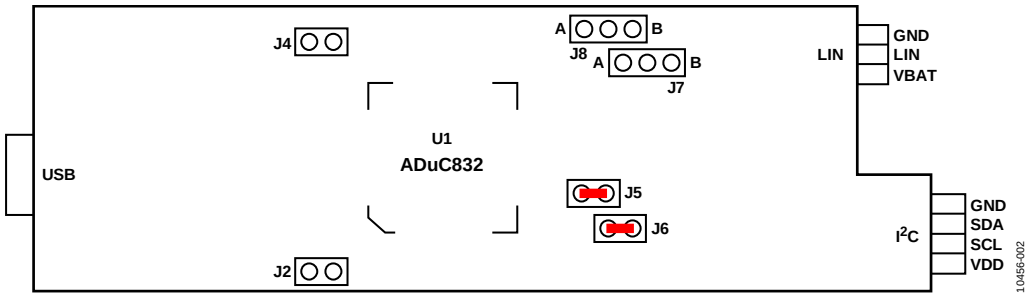

Figure 2. LIN Downloader Jumper Configuration

DOWNLOAD SEQUENCE

LINBWSD.dll contains a security sequencer to prevent corruption of the Flash. The normal download sequence using the DLL functions follows:

- OpenDongle();
- GetDongleStatus();
- ReadDongleIdent();
- AssignNAD();
  - If no default, NAD should be used
- InitLINkernel();
- ImportHexData();
- GetImportInfo();
- DoPageErase();
  - Number of pages get by GetImportInfo();
- DoDataWrite();
  - Number of bytes get by GetImportInfo();
- DoWriteCRC();
- DoResetKernel();
  - Used if necessary
- CloseDongle();
  - If no more CPUs to be programmed
- ReadByIdentifier();

GetDLLInfo

Getting the DLL version information.

void **GetDLLInfo** (*char *pcDLLInfo, BYTE *pusStringLen*)

Parameters

Table 1.

Parameter	Description
* *pcDLLInfo*	Pointer to a char buffer that contains the version information from the DLL
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data

Return Value

There is no return value.

Remark

It is necessary that an application close the opened USB port before closing the application. If an error occurs, the DLL opens a MessageBox.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;


pMessage = strMessage.GetBuffer(100);
GetDLLInfo(pMessage,&usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
MessageBox(strMessage);
```

OpenDongle

Opening USB port and looking for LIN-Dongle.

BYTE **OpenDongle** (char *pcOpenPortError, BYTE *pusStringLen)

Parameters

Table 2.

Parameter	Description
**pcOpenPortError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data

Return Value

The return values for OpenDongle are as follows:

- 0 = no error.
- 1 = error during opening of USB port or no LIN-Dongle connected.

Remark

If the application tries to open an already open port, the DLL ignores the command.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;

pMessage = strMessage.GetBuffer(100);
usResult = OpenDongle(pMessage,&usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0)            // Error
{
       MessageBox(strMessage);
       return;
}
```

CloseDongle

Close USB port.

BYTE **CloseDongle()**

Parameters

There are no parameters.

Return Value

The return values for CloseDongle are as follows:

- 0 = no error.
- 1 = error during closing of USB port.

Remark

It is necessary that the application close the open USB port before closing the application. If an error occurs, the DLL opens a MessageBox.

Example

```
BYTE usResult;

usResult = closeDongle();
```

GetDongleStatus

Starting dongle firmware, and testing hardware and software version from dongle.

BYTE **GetDongleStatus** (*char *pcOpenPortError, BYTE *pusStringLen*)

Parameters

Table 3.

Parameter	Description
**pcOpenPortError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data

Return Value

The return values for GetDongleStatus are as follows:

- 0 = no error.
- 1 = error during dongle initialization or wrong dongle.

Remark

There is no remark.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;

pMessage = strMessage.GetBuffer(100);
usResult = GetDongleStatus(pMessage,&usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0) // Error
{
        MessageBox(strMessage);
        return;
}
```

DoReadDongleIdent

Reading dongle firmware information.

BYTE **DoReadDongleIdent** *(1, char *pcDongleIdent, BYTE *pusStringLen*)

Parameters

Table 4.

Parameter	Description
1	First parameter must be 1 and should not be changed
**pcDongleIdent*	Pointer to a char buffer that contains the hardware and software information in ASCII from the dongle
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data

Return Value

The return values for DoReadDongleIdent are as follows:

- 0 = no error.
- 1 = error during getting dongle information.

Remark

There is no remark.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;

pMessage = strMessage.GetBuffer(100);
usResult = DoReadDongleIdent(1,pMessage,&usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0)
{
        MessageBox(strMessage);
        return
}
```

InitLinKernel

Connecting to kernel and getting its identification.

BYTE **InitLinKernel** (char **pcLinInitError, BYTE *pusStringLen, BYTE *pusCPU_Ident)*

Parameters

Table 5.

Parameter	Description
**pcLinInitError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data
**pusCPU_Ident*	Pointer to a variable of type BYTE that receives the CPU variant

Return Value

The return values for InitLinKernel are as follows:

- 0 = no error.
- 1 = error during initialization or no LIN-Dongle connected.
- 2 = detecting wrong CPU signature.

Remark

If GetDongleStatus() failed, the application should not use this function.

If the application does not use the default NAD, the AssignNAD() should call before InitLinKernel is called.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
BYTE usCPU_Ident;
CString strMessage;

pMessage = strMessage.GetBuffer(100);
usResult = InitLinKernel(pMessage,&usStrLen,&usCPU_Ident);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0) /          / Error
{
        MessageBox(strMessage);
        return;
}
```

ImportHexData

Importing the file that should be programmed.

BYTE **ImportHexData** (*char *cHexFilePath, char *pcImportError, BYTE *pusStringLen*)

Parameters

Table 6.

Parameter	Description
**cHexFilePath*	The file path to the hex file
**pcImportError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data

Return Value

The return values for ImportHexData are as follows:

- 0 = no error.
- 1 = error during import.

Remark

The application should use GetImportInfo() to get information about the imported data for erase, program, and verify.

Example

```
char *pMessage;
char szFileName[1024];
BYTE usStrLen;
BYTE usResult;
CString strMessage;


pMessage = strMessage.GetBuffer(100);
usResult = ImportHexData(szFileName ,pMessage, &usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0)            // Error
{
       MessageBox(strMessage);
       return;
}
```

GetImportInfo

Getting information about the imported file.

BYTE **GetImportInfo** (&*ulEndAdr, &ulHexBytes, &ucMaxPages, &ulAdrOffset*)

Parameters

Table 7.

Parameter	Description
&*ulEndAdr*	Variable for the highest address in hex file
&*ulHexBytes*	Variable for the number of bytes imported
&*ucMaxPages*	Variable for the number of pages to erase
&*ulAdrOffset*	Offset for programming (not necessary at the moment)

Return Value

There is no return value.

Remark

There is no remark.

Example

```
char *pMessage;
char szFileName[1024];
BYTE usStrLen;
BYTE usResult;
CString strMessage;

unsigned long ulEndAdr, ulHexBytes, ulAdrOffset;
BYTE ucMaxPages;


pMessage = strMessage.GetBuffer(100);
usResult = ImportHexData(szFileName ,pMessage, &usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0) // Error
{
        MessageBox(strMessage);
        return;
}
else
{
        GetImportInfo(&ulEndAdr, &ulHexBytes, &ucMaxPages, &ulAdrOffset);
}
```

DoPageErase

Erase one single page.

BYTE **DoPageErase** (*char *pcPageEraseError, BYTE *pusStringLen, BYTE *pusErasedPageNumber*)

Parameters

Table 8.

Parameter	Description
**pcPageEraseError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data
**pusErasedPageNumber*	Pointer to a variable of type BYTE that receives the number of pages that are erased

Return Value

The return values for DoPageErase are as follows:

- 0 = no error.
- 1 = error during page erase.
- 2 = application tries to erase more pages than necessary.

Remark

usErasedPageNumber is always 1.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;


BYTE ucPage = 0;


do
{
        pMessage = strMessage.GetBuffer(100);
        usResult = DoPageErase(pMessage, &usStrLen, &usErasedPageNumber);
        strMessage.ReleaseBuffer();
        if(usResult == 1)
        {
                strMessage.SetAt(usStrLen,'\0');
                MessageBox(strMessage);
                return;
        }
        if(usResult == 2)
        {
                MessageBox(„No more pages to erase“);
                ucPage = ucMaxPages;
        }
        ucPage ++;
}
while(ucPage<ucMaxPages);
// ucMaxPage contains the value get by GetImportInfo()
```

DoDataWrite

Writing data to flash, and verifying the written data.

BYTE **DoDataWrite** (*char *pcWriteDataError, BYTE *pusStringLen, BYTE *pusDataLen*)

Parameters

Table 9.

Parameter	Description
**pcWriteDataError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data
**pusDataLen*	Pointer to a variable of type BYTE that receives the number of bytes that are written

Return Value

The return values for DoDataWrite are as follows:

- 0 = no error.
- 1 = error during data write.

Remark

If not all pages are erased, the application cannot write data.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
BYTE usDataLen;
CString strMessage;

unsigned long ulWrittenHexBytes = 0;

do
{
        pMessage = strMessage.GetBuffer(100);
        usResult = DoDataWrite(pMessage, &usStrLen, &usDataLen);
        strMessage.ReleaseBuffer();
        if(usResult == 1)
        {
                strMessage.SetAt(usStrLen,'\0');
                MessageBox(strMessage);
                return;
        }
        ulWrittenHexBytes += (unsigned long)usDataLen;
}
while(ulWrittenHexBytes < ulHexBytes);
// ulHexBytes contains the value get by GetImportInfo()
```

DoWriteCRC

Writing checksum to 0x80014.

BYTE **DoWriteCRC** (*char *pcWriteCRCError, BYTE *pusStringLen, usCRC_Val*)

Parameters

Table 10.

Parameter	Description
**pcWriteCRCError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data
usCRC_Val	0 = writing calculated value from hex import 1 = writing value for code development 2 = writing value from imported hex file

Return Value

The return values for DoWriteCRC are as follows:

- 0 = no error.
- 1 = error during writing checksum.

Remark

If program and verify are not successful, the DLL blocks this function.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;


pMessage = strMessage.GetBuffer(100);
usResult = DoWriteCRC(pMessage,&usStrLen, 1); // code development
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0)            // Error
{
       MessageBox(strMessage);
       return;
}
```

DoResetKernel

Reset of the CPU.

BYTE **DoResetKernel** (*char *pcResetKernelError, BYTE *pusStringLen*)

Parameters

Table 11.

Parameter	Description
**pcResetKernelError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data

Return Value

The return values for DoResetKernel are as follows:

- 0 = no error.
- 1 = error during reset of kernel.

Remark

There is no remark.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;


pMessage = strMessage.GetBuffer(100);
usResult = DoResetKernel(pMessage,&usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0) // Error
{
        MessageBox(strMessage);
        return;
}
```

AssignNAD

Assigning a new NAD to the kernel.

BYTE **AssignNAD** (*char *pcAssignNADError, BYTE *pusStringLen, BYTE usNewNAD*)

Parameters

Table 12.

Parameter	Description
**pcAssignNADError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE that receives the length of the ASCII data
usNewNAD	New NAD

Return Value

The return values for AssignNAD are as follows:

- 0 = no error.
- 1 = error during assignment of NAD.

Remark

There is no remark.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;


pMessage = strMessage.GetBuffer(100);
usResult = AssignNAD(pMessage,&usStrLen);
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0)            // Error
{
       MessageBox(strMessage);
       return;
}
```

ReadByIdentifier

Reading the identifiers.

BYTE **ReadByIdentifier** (*char *pcRBIError, BYTE *pusStringLen, BYTE usIdentifier*)

Parameters

Table 13.

Parameter	Description
**pcRBIError*	Pointer to a char buffer that contains ASCII data
**pusStringLen*	Pointer to a variable of type BYTE which receives the length of the ASCII data
usIdentifier	0x0
	0x32
	0x33
	0x34

Return Value

The return values for ReadByIdentifier are as follows:

- 0 = no error.
- 1 = error during read by identifier.

Remark

There is no remark.

Example

```
char *pMessage;
BYTE usStrLen;
BYTE usResult;
CString strMessage;


pMessage = strMessage.GetBuffer(100);
usResult = ReadByIdentifier(pMessage,&usStrLen,0); // reading 0x0
strMessage.ReleaseBuffer();
strMessage.SetAt(usStrLen,'\0');
if(usResult != 0)            // Error
{
       MessageBox(strMessage);
       return;
}
```

NOTES

NOTES

AN10456-0-1/12(0)