AN891

Interrupt-based PIC18 Master LIN Driver in C for Enhanced USART

Author: Caio Gübel
Microchip Technology Inc.

The LIN protocol was originally designed by a group of European carmakers to be used as a low-cost, short distance, low-speed network for automotive applications (see **Appendix C: "References"**).

The main characteristics of the LIN protocol are:

- Serial communication
- Single master, multiple slave concept
- Low-cost, one-wire implementation
- Speed up to 20 Kbit/s
- Self-synchronization (on the slave side)
- Ensured latency time in transmission

This application note presents a Microchip Application Maestro™ compatible interrupt driven implementation of the Master Side Driver of the LIN protocol in a PIC18F device in C language (Microchip and HI-TECH 'C' compatible), which takes advantage of the new features provided by the PIC18 Enhanced USART module.

FILES

The implementation presented in this application note is based on the LIN Specification Package Version 1.3. This specification adheres to Microchip's Application Maestro standard and contains the following files:

- **`ELINMInt.c`** – C source file, contains all functions and variables used by the LIN protocol.
- **`ELINMInt.h`** – Header file, contains constants, unions and structures definitions, function prototypes and macro definitions used by the LIN protocol.
- **`ELINMInt.def`** – Contains the definitions used to configure the LIN protocol.
- **`ELINMInt.ex.txt`** – Example of code using the driver.

MACROS

The following macros are defined in the `ELINMInt.h` file:

Name	Usage	Description	Page
`mELINMIntInitialize`	Initialization	Initializes EUSART and driver's variables	3
`mELINMIntTXBufferAvailable`	Transmission	Checks for available transmission buffers	4
`mELINMIntGetTXPointer`	Transmission	Sets a tag and returns a buffer data pointer	5
`mELINMIntSendMessage`	Transmission	Requests the transmission of a message	6
`mELINMIntTXStatus`	Transmission	Returns the status of a sent message	8
`mELINMIntMessageSent`	Transmission	Checks for end of a message transmission	9
`mELINMIntTXErrorDetected`	Transmission	Checks for transmission errors	10
`mELINMIntTXErrorTag`	Transmission	Returns the tag of a message with error	11
`mELINMIntTXErrorCode`	Transmission	Returns the error code of a transmission	12
`mELINMIntRXBufferAvailable`	Reception	Checks for available reception buffers	13
`mELINMIntReceiveMessage(tag,i,s)`	Reception	Requests a slave to send a message	14
`mELINMIntMessageReceived`	Reception	Checks for end of reception process	15
`mELINMIntGetMessageTag`	Reception	Returns the tag of a received message	16
`mELINMIntGetRXPointer`	Reception	Returns a pointer to a received message	17
`mELINMIntRXMessageAvailable`	Reception	Checks for any message arrival	18
`mELINMIntRXStatus`	Reception	Returns the status of a received message	19
`mELINMIntRXErrorDetected`	Reception	Checks for errors in reception	20
`mELINMIntRXErrorTag`	Reception	Returns the error tag	21
`mELINMIntRXErrorCode`	Reception	Returns the error code of a reception	22
`mELINMIntCheckWakeUPReceived`	Bus Control	Signals that a slave issued a wake-up	23
`mELINMIntSendWakeUPSignal`	Bus Control	Sends a wake-up signal to the slaves	24
`mELINMIntSleepTimeOut`	Bus Control	Signals when bus Idle time exceeded Sleep time-out	25

mELINMIntInitialize()

This macro initializes the driver.

Syntax

```
mELINMIntInitialize();
```

Parameters

None

Return Values

0 – Initialization OK

!= 0 – Error in initialization

Preconditions

None

Side Effects

None

Remarks

In this first version of the protocol, no error can be returned. However, in order to be compatible with future versions that may incorporate error returns, designers must include the proper test.

Example

```
if(mELINMIntInitialize())      // if an error in initialization was detected
{
                               // error handling
}
else                           // if NO error (macro returned 0)
{
                               // normal processing
}
```

mELINMIntTXBufferAvailable()

This macro checks if there is a transmission buffer available. The application must call this macro before trying to initiate any transmission.

Syntax

```
mELINMIntTXBufferAvailable();
```

Parameters

None

Return Values

1 – There is an available buffer to be used to transmit a message

0 – No buffer is currently available for transmission

Preconditions

The protocol must have been successfully initialized using the `ELINMIntInitialize(void)` function.

Side Effects

None

Remarks

None

Example

```
if(mELINMIntTXBufferAvailable())     // check if there is an available TX buffer
{
                                     // init transmission from this point
}
```


mELINMIntGetTXPointer()

This macro returns a pointer to the available transmission buffer.

Syntax

`mELINMIntGetTXPointer(tag);`

Parameters

tag The tag of a message. This is an identification of the message to be saved and used by the LIN protocol to inform the application of an eventual error that the transmission of a specific message suffered. In the event of an error being detected, the user can access the tag of the error message and with this tag, read the error code.

Return Values

(BYTE *) – A byte type data pointer to the transmission buffer. The application will load the message to be transmitted using this pointer.

Preconditions

1. The protocol must have been successfully initialized by the `ELINMIntInitialize(void)` function.
2. The `mELINMIntTXBufferAvailable()` macro must have been invoked with success.

Side Effects

None

Remarks

The total size of the message to be loaded must not exceed the maximum allowed size defined by `ELINMINT_MAX_MESSAGE_SIZE (ELINMInt.def).`

Example

```
pt=mELINMIntGetTXPointer(3);          // get the pointer to message #3
pt[0]=mymsg[0];                       // insert first message byte
pt[1]=mymsg[1];                       // insert second message byte
```


mELINMIntSendMessage(tag,i,s)

This macro requests the transmission of a message through LIN.

Syntax

`mELINMIntSendMessage(tag,i,s);`

Parameters

tag The tag that identifies a message, previously defined by the application when calling the `mELINMIntGetTXPointer` macro.

i The ID of the message, ranging from 0x00 to 0x3F. Bits 6 and 7 of the ID will be filled with parity bits and their original content ignored.

s The size of the message, limited to 8 for all standard messages and to `ELINMINT_MAX_MESSAGE_SIZE` in the case of an extended frame (ID = 0x3E or ID = 0x3F).

Return Values

None

Preconditions

1. The `mELINMIntTXBufferAvailable` macro must have been successfully invoked.
2. The `mELINMIntGetTXPointer` macro must have been invoked.
3. The data buffer shall have been loaded with the message pointer.

Side Effects

None

Remarks

1. Calling this macro doesn't ensure that the message will be successfully transmitted. The application must check the result of the transmissions by:
   - Waiting until the message transmission is completed (using `mELINMIntMessageSent`) and then checking if an error was detected in that message.
   - Checking if a transmission buffer is available (using `mELINMIntTXBufferAvailable`) and if an error is detected, evaluating which message (identified by its tag) presented a problem and the nature of the problem.
2. The ID = 0x3F is reserved by the LIN Consortium for future expansion (see *LIN Specification Package 1.3, Protocol Specification Chapter 3.2*) and therefore, it's use may compromise future compatibility.
3. The macro takes the size of the message and calculates the minimum and maximum frame times to be used by the underlying function. If the size is passed in a variable, the calculations are done in real-time, requiring several cycles; however, if the application always calls this macro with fixed values instead of variables, then these calculations can be made in compile time, therefore, saving both code space and processing time.

Example 1 (Fixed Size)

```
mELINMIntSendMessage(tag,myID,4);            // requests transmission, size = 4, better!
while(mELINMIntMessageSent(tag)==0)          // wait transmission to be completed
    ;
if(mELINMIntTXStatus(tag)==ELINMINT_NO_ERROR)// check transmission status
{
                                             // no error, normal processing
}
else                                         // if error detected
{
                                             // error handling
}
```

Example 2 (Variable Size)

```
mELINMIntSendMessage(tag,myID,msgSize);      // requests transmission, variable size
while(mELINMIntMessageSent(tag)==0)          // wait transmission to be completed
    ;
if(mELINMIntTXStatus(tag)==ELINMINT_NO_ERROR)// check transmission status
{
                                             // no error, normal processing
}
else                                         // if error detected
{
                                             // error handling
}
```

mELINMIntTXStatus(tag)

This macro checks the status of a message already transmitted.

Syntax

`mELINMIntTXStatus(tag);`

Parameters

tag This byte contains a message tag which is an identification of the message that was sent.

Return Values

The error code, defined according to the following table:

#define	Definition
ELINMINT_NO_ERROR	No error was detected
ELINMINT_THMIN_ERROR	Header time too short
ELINMINT_THMAX_ERROR	Header time too long
ELINMINT_TFMIN_ERROR	Frame time too short
ELINMINT_TFMAX_ERROR	Frame time too long
ELINMINT_CHECKSUM_ERROR	Checksum incorrect
ELINMINT_DATA_ERROR	Received and transmitted bytes don't match
ELINMINT_FRAMING_ERROR	Framing error

Preconditions

The `mELINMIntSendMessage` macro must have been invoked.

The message transmission was completed, checked by `mELINMIntMessageSent`.

Side Effects

None

Remarks

This macro returns the result of the transmission of a specific message identified by *tag*.

Example

```
mELINMIntSendMessage(9,0x04,2);             // send a message
while(mELINMIntMessageSent(9)==0)           // wait transmission message #9 completed
    ;
if(mELINMIntTXStatus(9)==ELINMINT_NO_ERROR)    // check transmission status
{
                                               // no error, normal processing
}
else                                           // if error detected
{
                                               // error handling
}
```

mELINMIntMessageSent(tag)

This macro checks if a message identified by *tag* was already sent.

Syntax

```
mELINMIntMessageSent(tag);
```

Parameters

tag This byte contains a message tag which is an identification of the message, which the driver can use to track a specific message.

Return Values

1 – Message already sent

0 – Message not yet sent

Preconditions

The `mELINMIntSendMessage` macro must have been invoked.

Side Effects

None

Remarks

This macro flags when a specific message transmission is complete. However, it doesn't ensure that the transmission was successful. The application must check it using `mELINMIntTXErrorDetected`.

Example

```
mELINMIntSendMessage(9,0x04,2);         // send a message
while(mELINMIntMessageSent(9)==0)       // wait transmission message #9 completed
    ;
```

mELINMIntTXErrorDetected()

This macro flags if an error was detected in the transmission of a message.

Syntax

`mELINMIntTXErrorDetected();`

Parameters

None

Return Values

1 – Error detected

0 – No Error detected

Preconditions

Called after detecting that a message was transmitted either by the `mELINMIntMessageSent` macro or by the `mELINMIntTXBufferAvailable` macro.

Side Effects

None

Remarks

None

Example 1

```
mELINMIntSendMessage(9,0x04,2);             // send a message
while(mELINMIntMessageSent(9)==0)           // wait transmission #9 to complete
    ;
if(mELINMIntTXErrorDetected())              // check if an TX error was detected
{
   errTx=mELINMIntTXErrorTag();             // if detected let's find the message
                                            // that caused the error
                                            // error handling
}
```

Example 2

```
mELINMIntSendMessage(9,0x04,2);             // send a message
while(mELINMIntTXBufferAvailable())         // wait for an available buffer
    ;
if(mELINMIntTXErrorDetected())              // check if an TX error was detected
{
   errTx=mELINMIntTXErrorTag();             // if detected let's find the message
                                            // that caused the error
                                            // error handling

}
```

`mELINMIntTXErrorTag()`

This macro returns the tag of the message that presented an error.

Syntax

```
mELINMIntTXErrorTag();
```

Parameters

None

Return Values

A byte with the tag of the error message.

Preconditions

Error has been previously detected by `mELINMIntTXErrorDetected()`.

Side Effects

None

Remarks

None

Example

```
errorTag=mELINMIntTXErrorTag();   // read the tag of the message that had an error
```

mELINMIntTXErrorCode(tag)

This macro returns in one byte the error associated with a message.

Syntax

```
mELINMIntTXErrorCode(tag);
```

Parameters

tag The identification of the message where an error was detected.

Return Values

The error code, defined according to the following table:

#define	Definition
ELINMINT_NO_ERROR	No error was detected
ELINMINT_THMIN_ERROR	Header time too short
ELINMINT_THMAX_ERROR	Header time too long
ELINMINT_TFMIN_ERROR	Frame time too short
ELINMINT_TFMAX_ERROR	Frame time too long
ELINMINT_CHECKSUM_ERROR	Checksum incorrect
ELINMINT_DATA_ERROR	Received and transmitted bytes don't match
ELINMINT_FRAMING_ERROR	Framing error

Preconditions

1. Error detected by `mELINMIntTXErrorDetected`.
2. Tag of the related message read by `mELINMIntTXErrorTag`.

Side Effects

None

Remarks

None

Example

```
if(mELINMIntTXErrorDetected())                 // check if an TX error was detected
{
   errorTag=mELINMIntTXErrorTag();             // get the tag of the message
   errorCode=mELINMIntTXErrorCode(errorTag);   // find the error code
                                               // error handling
}
```


`mELINMIntRXBufferAvailable()`

This macro flags if there is a reception buffer available.

Syntax

`mELINMIntRXBufferAvailable();`

Parameters

None

Return Values

0 – No buffer available

1 – Buffer available

Preconditions

The protocol must have been successfully initialized by the `ELINMIntInitialize(void)` function.

Side Effects

None

Remarks

None

Example

```
if (mELINMIntRXBufferAvailable())       // if there is a reception buffer available
   mELINMIntReceiveMessage(5,0x01,2);   // request data: tag #5, ID=0x01 and Size=2
```

`mELINMIntReceiveMessage(tag,i,s)`

This macro requests a message to be sent from a slave.

Syntax

```
mELINMIntReceiveMessage(tag,i,s);
```

Parameters

tag The tag defined by the application to be associated with the incoming message requested.

i The ID of the requested message.

s The size of the message in bytes.

Return Values

None

Preconditions

Reception buffer available (detected by `mELINMIntRXBufferAvailable`).

Side Effects

None

Remarks

The request of reception, like in the transmission, doesn't ensure that the received message is correct; therefore, the application must check the results.

Example

```
mELINMIntReceiveMessage(5,0x01,2);   // request data: tag #5, ID=0x01 and Size=2
```

mELINMIntMessageReceived(tag)

This macro checks if a message was received.

Syntax

`mELINMIntMessageReceived(tag);`

Parameters

tag The received message identification.

Return Values

1 – Message received

0 – Message not yet received

Preconditions

Reception of a message previously requested by the `mELINMIntReceiveMessage` macro.

Side Effects

None

Remarks

This macro detects the reception of a message, but doesn’t ensure its correctness; therefore, the application must check it.

Example

```
mELINMIntReceiveMessage(7,0x01,2);      // request data:tag=5, ID=0x01, size=2
while(mELINMIntMessageReceived(7)==0)   // wait this message to be received
    ;
```

mELINMIntGetMessageTag()

This macro returns the tag (identification) of a message that was received.

Syntax

`mELINMIntGetMessageTag();`

Parameters

None

Return Values

The tag (identification) of the received message.

Preconditions

1. The reception of a message must have already been requested using the `mELINMIntReceiveMessage` macro.
2. The message reception Acknowledged by the `mELINMIntMessageReceived()` macro.
3. No error detected as per the `mELINMIntRXErrorDetected()` macro (returning 0).

Side Effects

None

Remarks

None

Example

```
if(mELINMIntRXErrorDetected()==0)    // check if an reception error was detected
{                                    // if no error then
   Tag=mELINMIntGetMessageTag();     // get the tag of the message received
   pt=mELINMIntGetRXPointer(Tag);    // get the data pointer of the message

}
```

mELINMIntGetRXPointer(tag)

This macro returns a pointer to the received message.

Syntax

`mELINMIntGetRXPointer(tag);`

Parameters

tag The tag identifying the message.

Return Values

(BYTE *) – A byte type data pointer to the reception buffer. The application can read the message using this pointer.

Preconditions

1. Reception of a message detected by the `mELINMIntMessageReceived` macro.
2. No error detected by `mELINMIntRXErrorDetected` (return 0).
3. Tag of the message read using the `mELINMIntGetMessageTag` macro.

Side Effects

None

Remarks

None

Example

```
Tag=mELINMIntGetMessageTag();        // get the tag of the message received
pt=mELINMIntGetRXPointer(Tag);       // get the data pointer
```

mELINMIntRXMessageAvailable()

This macro checks for the reception of a message.

Syntax

```
mELINMIntRXMessageAvailable();
```

Parameters

None

Return Values

1 – Message Received

0 – No Message received

Preconditions

1. Protocol initialized – `ELINMIntInitialize()` macro invoked.
2. Message reception requested – `LINMIntReceiveMessage` macro invoked.

Side Effects

None

Remarks

Because the reception of a message is Acknowledged even when an error has occurred, the application will always check the integrity of the received message.

Example

```
if(mELINMIntRXMessageAvailable())      // check for received message
{                                      //

}
```

mELINMIntRXStatus(tag)

This macro checks the status of a received message.

Syntax

`mELINMIntRXStatus(tag);`

Parameters

tag The identification of the message.

Return Values

The error code, defined according to the following table:

#define	Definition
ELINMINT_NO_ERROR	No error was detected
ELINMINT_THMIN_ERROR	Header time too short
ELINMINT_THMAX_ERROR	Header time too long
ELINMINT_TFMIN_ERROR	Frame time too short
ELINMINT_TFMAX_ERROR	Frame time too long
ELINMINT_CHECKSUM_ERROR	Checksum incorrect
ELINMINT_DATA_ERROR	Received and transmitted bytes don't match
ELINMINT_FRAMING_ERROR	Framing error

Preconditions

1. Protocol initialized – `ELINMIntInitialize()` macro invoked.
2. Message reception requested using `mELINMIntReceiveMessage`.
3. Reception completed, checked with `mELINMIntMessageReceived`.

Side Effects

None

Remarks

None

Example

```
if(mELINMIntRXMessageReceived(tag))      // check for received message
{                                        //
    status=mELINMIntRXStatus(tag);
    if(status)                           // if error handle it
    {

    }
    else                                 // otherwise read it
    {

    }
}
```

`mELINMIntRXErrorDetected()`

This macro checks for reception errors.

Syntax

```
mELINMIntRXErrorDetected();
```

Parameters

None

Return Values

1 – Error detect

0 – No error detect

Preconditions

1. Protocol initialized – `ELINMIntInitialize()` macro invoked.
2. Message reception requested – `LINMIntReceiveMessage` macro invoked.
3. Reception of message Acknowledged – `mELINMIntMessageReceived`.

Side Effects

None

Remarks

Because the reception of a message is Acknowledged even when an error has occurred, the application will always check the integrity of the received message with this macro. In case of an error, the `mELINMIntRXErrorTag` and the `mELINMIntRXErrorCode` macros are called to identify the exact nature of the problem.

Example

```
if(mELINMIntRXErrorDetected())        // check if an RX error was detected
{
                                      // error handling
}
```

mELINMIntRXErrorTag()

This macro returns the tag of the message that presented an error.

Syntax

`mELINMIntRXErrorCode();`

Parameters

None

Return Values

tag The identification of the error message that was received.

Preconditions

An error detected by the `mELINMIntRXErrorDetected` macro.

Side Effects

None

Remarks

None

Example

```
if(mELINMIntRXErrorDetected())          // check if an RX error was detected
{
   ErrorTag=mELINMIntRXErrorTag();       // find the tag of the message with error
                                        //
}
```

mELINMIntRXErrorCode(tag)

This macro returns the code identifying the error detected in the reception of the message identified by *tag*.

Syntax

```
mELINMIntRXErrorCode(tag);
```

Parameters

tag The identification of the error message received.

Return Values

The error code, defined according to the following table:

#define	Definition
ELINMINT_NO_ERROR	No error was detected
ELINMINT_THMIN_ERROR	Header time too short
ELINMINT_THMAX_ERROR	Header time too long
ELINMINT_TFMIN_ERROR	Frame time too short
ELINMINT_TFMAX_ERROR	Frame time too long
ELINMINT_CHECKSUM_ERROR	Checksum incorrect
ELINMINT_DATA_ERROR	Received and transmitted bytes don't match
ELINMINT_FRAMING_ERROR	Framing error

Preconditions

1. Error detected by the `mELINMIntRXErrorDetected` macro.
2. The tag of the error read with the `mELINMIntRXErrorTag` macro.

Side Effects

None

Remarks

None

Example

```
ErrorCode=mELINMIntRXErrorCode(ErrorTag);// read the error code for the given tag
```

`mELINMIntCheckWakeUPReceived()`

This macro flags the reception of a wake-up signal from the slave.

Syntax

`mELINMIntCheckWakeUPReceived();`

Parameters

None

Return Values

1 – Wake-up received

0 – No wake-up received

Preconditions

Protocol initialized by the `ELINMIntInitialize` function.

Side Effects

None

Remarks

1. This macro flags the reception of a wake-up (reception of 0x80 from a slave). According to the specifications (*LIN Specification Package 1.3, Twudel Parameter*), the Master must wait at least a 4-bit time before starting communication. Therefore, once the application detects the wake-up signal, it must allow this minimum time before starting the communication process.
2. This signal is kept active until a message transmission or reception is completed.

Example

```
if(mELINMIntCheckWakeUPReceived())   // once received the wake-up from a slave
{                                    // process it to detect what happened

}
```

mELINMIntSendWakeUPSignal()

This macro sends a wake-up signal to the slaves.

Syntax

```
mELINMIntSendWakeUPSignal();
```

Parameters

None

Return Values

None

Preconditions

Protocol initialized by the `ELINMIntInitialize` function.

Side Effects

None

Remarks

This macro sends a wake-up signal (0x80). According to the LIN specifications (*LIN Specification Package 1.3, Twudel Parameter*), the Master must wait at least a 4-bit time before starting communication. Therefore, once the application sends the wake-up signal, it must allow this minimum time before starting the communication process.

Example

```
mELINMIntSendWakeUPSignal();
```

`mELINMIntSleepTimeOut()`

This macro detects the time-out of the bus.

Syntax

`mELINMIntSleepTimeOut();`

Parameters

None

Return Values

1 – Time-out

0 – No time-out

Preconditions

None

Side Effects

None

Remarks

The time-out condition is detected when no bus activity is observed for a time interval larger than 25000 bits. In this case a flag is set and the condition can be detected by calling this macro and if necessary, a wake-up signal may be issued (`mELINMIntSendWakeUPSignal`).

Example

```
if(mELINMIntSleepTimeOut())       // if an sleep time-out is detected
{                                 //

}
```

DRIVER USAGE

There are two ways to add the driver to a project:

1. Through Application Maestro™ Software:
   Select the module, adjust the parameters and select the directory location of the files that are going to be copied. Please refer to the Application Maestro documentation of this module for further explanation.

   After this, the designer must include the `ELINMInt.c` file in the project (inside MPLAB® IDE, as a C18 C Compiler source code file) and copy the following include file in all source code files accessing the protocol:

   ```
   #include "ELINMInt.h"
   ```

2. Manually:
   To add the driver into the project, do the following:
   a) Copy all three files in the source code directory of the project.
   b) Include the `ELINMInt.c` file in the project (inside MPLAB® IDE, as a C18 C Compiler source code file).
   c) Copy the following include file in all source code files accessing the protocol:

   ```
   include "ELINMInt.h"
   ```

   d) Adjust the parameters of the driver. These parameters are located inside the `ELINMInt.def` file and are described in **Appendix B: "ELINMInt.def Parameter Setup"**.

Note:	This first version of the protocol supports only one communication buffer, therefore, the use of tags wouldn't be necessary. Future versions will implement multiple buffers (queuing), therefore, the application shall always send and receive the messages with the proper tag assignment.

APPENDIX A: CODE EXAMPLE

```
#include "ELINMInt.h"                     // .H file - to be added in all
// source files that reference            // LIN routines or constants
void InterruptVectorHigh(void);           // prototype of the routines used in this test
void InterruptVectorLow(void);
void InterruptHandler(void);
void main(void);

BYTE my_msg[8];                           // message buffer, for tests

/*********************************************************************
 * Function:        void Main(void)
 *
 * PreCondition:
 *
 * Input:
 *
 * Output:
 *
 * Side Effects:
 *
 * Stack Requirements:
 *
 * Overview:Main function of the LIN Master Test Firmware
 *
 ********************************************************************/

void main(void)
{
 char leds;
 unsigned int mydelay;
 BYTE *pt;
 BYTE ErrorCode;
 BYTE ErrorTag;
 BYTE Tag;
 BYTE rxtag;

 mydelay=2000;                        // initialize the delay variable
 TRISC=0x9F;                          // init serial pins - TX and control (LIN driver)
 PORTCbits.RC5=0;                     // negative edge (initial) pulse in the control
 if(mELINMIntInitialize()==0)         // initialize Enhanced USART and LIN
 ErrorCode=mELINMIntInitErrorCode();  // if an error was detected return the Initialization
                                      // error code
 T0CON=0xC8;                          // initialize TIMER0
 INTCON_TMR0IE=1;                     // enable timer0 int.
 INTCON_PEIE=1;                       // enable ints.
 PORTCbits.RC5=1;                     // positive edge pulse in the control pin (LIN Driver)
 INTCON_GIE=1;
 while(mydelay--)                     // initial delay -> necessary to MCP201.
                                      // (Data Sheet - param. Tcsor)
```

```
 ;
 while(1)                              // run forever
 {
    Tag=0;                             // init Tag
    mydelay=600;
    while(mydelay--)                   // give a delay between messages, to make scope
        ;                              // visalization easier
//*******************************************************
// First receive a single message using fixed value tag
// checking for the reception of that specific message
//*******************************************************

  while(mELINMIntRXBufferAvailable()==0)     // if there is no RX buffer available wait
    ;
  mELINMIntReceiveMessage(5,0x01,2);         // request data using tag=5 (message number),
                                             // ID=0x01, size=2
  while(mELINMIntMessageReceived(5)==0)      // wait until the message is received
    ;
  if((ErrorCode=mELINMIntRXStatus(5)))       // check if an RX error was detected
  {
                                             // error handling - to be added at this point by
   leds++;                                   // application
  }
  else                                       // otherwise (no error)
  {
   pt=mELINMIntGetRXPointer(5);              // get the data pointer
   my_msg[0]=*pt;                            // read the message
   pt++;
   my_msg[1]=*pt;
                                             // received message handling - to be added at this
                                             // point by the application

  }                                          // else

  mydelay=600;
  while(mydelay--)                           // give another delay
   ;

//*******************************************************
// Send a single message using fixed value tag
// checking for the transmission of that specific message
//*******************************************************

  while(mELINMIntTXBufferAvailable()==0)     // Wait TX buffer available
   ;
  pt= mELINMIntGetTXPointer(3);              // get available pointer and tag it's message as 3
  *pt=my_msg[0];                             // insert data
  pt++;
  *pt=0;
  mELINMIntSendMessage(3,0x04,2);            // send message
  while(mELINMIntMessageSent(3)==0)          // wait until transmission message #3 completes
   ;                                         // the application
  if((ErrorCode=mELINMIntTXStatus(3)))       // check if an TX error was detected (!=0)
  {
                                             // error handling - to be added at this point by
                                             // the application
  leds++;                                    // application
  }

  mydelay=600;
  while(mydelay--)                           // give another delay
  ;
```

```
//****************************************************
// Check for Sleep Time-Out and wake-up if necessary
//****************************************************

  if(mELINMIntSleepTimeOut())                // if timeout set (more than 25000 bit-time
  {                                          // of silence in the BUS)
   mELINMIntSendWakeUPSignal();              // send wake-up signal to the slaves
                                             // add application code here
  }

//****************************************************
// Check for Wake-Up Signal sent by slave
//****************************************************

  if(mELINMIntCheckWakeUPReceived())     // check for Wake-Up signals received
  {
   if(mELINMIntSleepTimeOut())           // if timeout already set (more than 25000 bit-time)
   {                                     // of silence in the BUS

   }
   else                                  // if no timeout something unexpected happened, process
   {

   }
  }
 } // while (1)
} // void main(void)


//**************************************************************************
// High priority interrupt vector
#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh(void)
{
 _asm
 bra InterruptHandler                    // jump to interrupt routine
 _endasm
}
#pragma code InterruptVectorHigh = 0x08
void InterruptVectorLow(void)
{
 _asm
 bra InterruptHandler                    // jump to interrupt routine
 _endasm

}

/*********************************************************************
 * Function:    void InterruptHandler(void)
 *
 * PreCondition:
 *
 * Input:
 *
 * Output:
 *
 * Side Effects:
 *
 * Stack Requirements:
 *
 * Overview:High priority interrupt routine
 *
```

```
 *****************************************************************/
#pragma code
#pragma interrupt InterruptHandler
void InterruptHandler(void)
{
 if(INTCON_TMR0IF)
  ELINMIntHandler();                    // process LIN int. based protocol
 TMR0L|=0x80;                           // timer0 int. every 128 counts - 128 instructions
                                        // time as timer0 is not using any prescaler (1:1)
 INTCON_TMR0IF=0;                       // reset int0 flag

}
```

APPENDIX B: ELINMInt.def PARAMETER SETUP

The ELINMInt.def file has many parameters, some of which must be adjusted by the designer. The parameters that require adjustment are:

CLOCK_FREQ

Description

This is the main clock frequency used by the microcontroller when running the driver.

Minimum 4000000L

Maximum 40000000L

Remarks

None

Example

```
#define    CLOCK_FREQ16000000L              // define the main clock as 16MHz
```

ELINMINT_BAUD

Description

This is the baud rate to be used in the bus by the LIN driver.

Minimum 1000L

Maximum 20000L

Remarks

None

Example

```
#define    ELINMINT_BAUD19200L              // Baud Rate adjusted to 19200 Baud
```

ELINMINT_MAX_MESSAGE_SIZE

Description

This is the maximum size of a message either to be transmitted or received.

Minimum 2

Maximum 255

Remarks

None

Example

```
#define    ELINMINT_MAX_MESSAGE_SIZE16      // maximum size 16 bytes
```

ELINMINT_INTERRUPT_PERIOD

Description

This is the interrupt period of the interrupt routine (in microseconds) used by the application.

Minimum The minimum period of this timer-based interrupt must be larger than the sum of the total time used by the LIN interrupt handler, plus the interrupt latency and any other task run by the application in the interrupt. The minimum time required by the LIN interrupt handler is calculated inside (ELINMInt.H) as follows:

```
// number of instructions run by the protocol during an interrupt

#define     ELINMINT_NINST_HANDLER_MIN112L

// here the necessary time to run the protocol during an int. is calculated as a
// function of the uC's clock frequency (CLOCK_FREQ) and the number of inst. of the
// LIN int. handler

#define ELINMINT_INT_HANDLER_TIME((1000000L*(4*(ELINMINT_NINST_HANDLER_MIN+5)))/CLOCK_FREQ)

// after that the processing time of the protocol is compared with the interrupt.
// execution time and if smaller ( interrupt period > processing time)
// then set an error message.

#if ELINMINT_INT_HANDLER_TIME>ELINMINT_INTERRUPT_PERIOD
    #error  "LIN TIMING NOT VIABLE - INTERRUPT PERIOD TOO SMALL !!"
#endif
```

A safe approach is to assume that the handler is going to take about ELINMINT_NINST_HANDLER_MIN instructions, multiply by a safety margin value (e.g., 1.2) and calculate the time spent in microseconds (multiplying the result by (4/CLOCK_FREQ)).

If the interrupt process time becomes larger than the interrupt period, an error message will be issued and the compiling process will fail.

Maximum 8*(ELINMINT_BAUD)

Remarks

As a rule of thumb, a good interrupt period should be smaller than 3-bits time (3/ELINMINT_BAUD) because this time may affect the interbyte delay.

Example

```
#define         ELINMINT_INTERRUPT_PERIOD       128L     // 128usec interrupt rate
```


ELINMINT_INTERBYTE_SPACE

Description

This is the delay time added between the transmission of two bytes in a message. This delay time is automatically calculated based on the baud rate and the interruption period.

Minimum 0

Maximum

Remarks

In some systems, the slave requires a delay slightly larger than the one provided. In these cases, the designer may have to increase the size of the delay. It is recommended to increase this value in steps of one, as an excessive increment in the delays may lead to errors, like excessive header time or excessive frame time.

Example

With ‘`1`’ added to the original Interbyte space.

```
#define ELINMINT_INTERBYTE_SPACE ((ELINMINT_INTERBYTE_MIN+ELINMINT_INTERBYTE_MAX)/2 +1)
```

APPENDIX C: REFERENCES

- LIN Consortium (http://www.lin-subbus.de)
  The authoritative reference, the LIN Consortium provides all standards and specifications necessary to understand and implement LIN-based communication systems.
- Microchip (http://www.microchip.com)
  Microchip provides extensive support to LIN:
  - Hardware:
    Transceivers, LIN Enabled Microcontrollers, Development Boards
  - Software:
    Application Notes, Code Examples and Templates

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003 . The Company’s quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Atlanta
3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888
Fax: 949-263-1338

Phoenix
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966
Fax: 480-792-4338

San Jose
1300 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-215-1444

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Unit 706B
Wan Tai Bei Hai Bldg.
No. 6 Chaoyangmen Bei Str.
Beijing, 100027, China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu
Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen
Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde
Room 401, Hongjian Building
No. 2 Fengxiangnan Road, Ronggui Town
Shunde City, Guangdong 528303, China
Tel: 86-765-8395507 Fax: 86-765-8395571

China - Qingdao
Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan
Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy
Via Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands
P. A. De Biesbosch 14
NL-5152 SC Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820

11/24/03