AVR286: LIN Firmware Base for LIN/UART Controller

8-bit AVR® Microcontroller

Application Note

LIN Features

The LIN (Local Interconnect Network) is a serial communications protocol which efficiently supports the control of mechatronics nodes in distributed automotive applications. The main properties of the LIN bus are:

- **Single master with multiple slaves concept**
- **Low cost silicon implementation based on common UART communication**
- **Self synchronization with on-chip oscillator in slave node**
- **Deterministic signal transmission with signal propagation time computable in advance**
- **Low cost single-wire implementation**
- **Speed up to 20 Kbit/s.**

1. Atmel® LIN/UART Controller

The LIN/UART Controller is available on some AVR® micro controllers such as ATmega32/64/M1/C1 or ATtiny167.

1.1 Features

- **Hardware Implementation of LIN 2.x (LIN 1.x Compatibility)**
- **Small, CPU Efficient and Independent Master/Slave Routines Based on “LIN Work Flow Concept” of LIN 2.x Specification**
- **Automatic LIN Header Handling and Filtering of Irrelevant LIN Frames**
- **Automatic LIN Response Handling**
- **Extended LIN Error Detection and Signaling**
- **Hardware Frame Time-out Detection**
- **“Break-in-data” Support Capability**
- **Automatic Re-synchronization to Ensure Proper Frame Integrity**
- **Fully Flexible Extended Frames Support Capabilities**

1.2 Controller Overview

The LIN/UART Controller is designed to match as closely as possible the LIN software application structure. The LIN software application is developed as independent tasks, several slave tasks and one master task. The controller conforms to this partitioning. The only link between a master task and a slave task in a master node is the (interrupt) flag IDOK. For a master task, this event represents the end of this task, for the slave task it represents the beginning of this task.

When the slave task is warned of an identifier presence, it has first to analyze it to know what to do with the response. Hardware flags identify the presence of one of the specific identifiers from 60 (0x3C) up to 63 (0x3F).

8123A–AVR–03/08

2. Firmware Base

2.1 Description

This application note provides a firmware base (drivers) to use the LIN/UART Controller on the ATmega32/64/M1/C1 and ATtiny167. It assumes that readers are familiar with the device architecture and with the LIN/UART Controller. A minimum knowledge of LIN specification v1.x or v2.x (www.lin-subbus.org) is also required to understand the content of this document.

This document is written for the software developers to help in the development of their applications or their LIN stack construction with the Atmel micro controllers. The drivers are designed to simplify the programming of the LIN/UART Controller.

Some general information and tips are also discussed to help the developers to build quickly their applications. In addition, this document provides application examples to illustrate the usage of these drivers.

2.2 Limitations

- **Diagnostics frames:** handled by LIN library, software placed above the drivers,
- **Sleep & Wake-up:** features application dependent,
- **Configuration for LIN hardware test purposes:** not included.

2.3 Terminology

ID: LIN Identifier
Tx: Transmission
Rx: Reception

2.4 Coding Style

The coding style explained below are important to understand the firmware:

- **Defined constants use caps characters.**
```
#define CONF_LINBRR   25
```
- **C-macro functions use the first character as cap**
```
#define Is_lin_header_ready() ((LINSIR & (1<<LIDOK)) ? TRUE: FALSE)
```
- **Functions use small caps characters**
```
void lin_get_response (unsigned char *l_data)
```

2.5 Device-specific I/O Header File

The firmware described in this document is based upon the device-specific I/O header file provided by the compiler:

1. **AVR GCC**: "*iom64m1.h*", "*iom64c1.h*", "*iom32m1.h*" or "*iom32c1.h*",
2. **IAR**: "*iom64m1.h*", "*iom64c1.h*", "*iom32m1.h*" or "*iom32c1.h*",
3. ...

In the device-specific I/O header file, the name, the address and the content (register-bit) of all the registers are defined.

3. Definitions

First, an application will have to define the following values:

```
#define FOSC          8000          // Device frequency in KHz
#define LIN_BAUDRATE  19200         // LIN baudrate in bit/s
```

3.1 Bit Time

The bit time definition **CONF_LINBRR**, setup the baud rate for communication. This definition is controlled by **FOSC** and **LIN_BAUDRATE**. At the end, only one **CONF_LINBRR** value is retained to program the LIN baud rate register of the LIN/UART Controller.

```
#ifndef FOSC
#error  You must define FOSC

#elif   FOSC == 16000               // if device frequency = 16 MHz
#ifndef LIN_BAUDRATE
#error  You must define LIN_BAUDRATE
#elif   LIN_BAUDRATE == 19200
#define CONF_LINBRR  25             // (0x19) 19.2 kbps, error = 0.2%
#elif   LIN_BAUDRATE == 9600
#define CONF_LINBRR  51             // (0x33) 9.6 kbps, error= 0.2%
#elif   LIN_BAUDRATE == 4800
#define CONF_LINBRR  103            // (0x67) 4.8 kbps, error= 0.2%
#elif   LIN_BAUDRATE == 2400
#define CONF_LINBRR  207            // (0xCF) 2.4 kbps, error=-0.2%
#else
#error  Not available LIN_BAUDRATE value
#endif

#elif   FOSC == 8000                // if device frequency = 8 MHz
#ifndef LIN_BAUDRATE
#error  You must define LIN_BAUDRATE
#elif   LIN_BAUDRATE == 19200
#define CONF_LINBRR  12             // (0x0C) 19.2 kbps, error = 0.2%
#elif   LIN_BAUDRATE == 9600
#define CONF_LINBRR  25             // (0x19) 9.6 kbps, error= 0.2%
#elif   LIN_BAUDRATE == 4800
#define CONF_LINBRR  51             // (0x33) 4.8 kbps, error= 0.2%
#elif   LIN_BAUDRATE == 2400
#define CONF_LINBRR  103            // (0x67) 2.4 kbps, error=-0.2%

#else
#error  Not available LIN_BAUDRATE value
#endif

#else
#error  Not available FOSC value
#endif
```

This method uses the conditional compilation of the C preprocessor. This method saves time and code bytes compared to a code line in C. It is very easy to add a new set of **CONF_LINBRR** values for other **FOSC** definitions (ex: `FOSC == 4000`). The **CONF_LINBRR** value is to calculate using the following formula:

```
CONF_LINBRR = [round {(FOSC x 1000) / (32 x LIN_BAUDRATE)}]-1
```

3.2 Configuration

```
// LIN protocols
#define LIN_1X          (1<<LIN13)
#define LIN_2X          (0<<LIN13)
```

```
// LIN commands
#define LIN_CMD_MASK    ((1<<LCMD1)|(1<<LCMD0))
#define LIN_RX_HEADER   ((0<<LCMD1)|(0<<LCMD0))
#define LIN_ABORT       ((0<<LCMD1)|(0<<LCMD0))
#define LIN_TX_HEADER   ((0<<LCMD1)|(1<<LCMD0))
#define LIN_RX_RESPONSE ((1<<LCMD1)|(0<<LCMD0))
#define LIN_TX_RESPONSE ((1<<LCMD1)|(1<<LCMD0))

// LIN interrupt flags
#define LIN_ERROR        (1<<LERR )
#define LIN_IDOK         (1<<LIDOK)
#define LIN_RXOK         (1<<LRXOK)
#define LIN_TXOK         (1<<LTXOK)

// LIN ID masks
#define LIN_1X_ID_MASK  ((1<<LID3)|(1<<LID2)|(1<<LID1)|(1<<LID0))
#define LIN_1X_LEN_MASK ((1<<LID5)|(1<<LID4))
#define LIN_2X_ID_MASK  ((1<<LID5)|(1<<LID4)| LIN_1X_ID_MASK )
```

```
// Specific to ATmega32/64/M1/C1          // Specific to ATtiny167
#define LIN_PORT_IN      PIND             #define LIN_PORT_IN      PINA
#define LIN_PORT_DIR     DDRD             #define LIN_PORT_DIR     DDRA
#define LIN_PORT_OUT     PORTD            #define LIN_PORT_OUT     PORTA
#define LIN_INPUT_PIN    4                #define LIN_INPUT_PIN    0
#define LIN_OUTPUT_PIN   3                #define LIN_OUTPUT_PIN   1
```

This method of definition tolerates changes of register address and allocation of register-bit.

4. C-macros

C-macros are commonly used in C to define small snippets of code. During the preprocessing phase, each C-macro call is replaced, in-line, by the corresponding C-macro definition. If the C-macro has parameters, they are substituted into the C-macro body during expansion; thus, a C-macro can mimic a C-function. The usual reason for doing this is to avoid the overhead of a function call in simple cases, where the code is lightweight enough that function call overhead has a significant impact on performance.

4.1 C-macros for LIN 1.x

```
// Use LIN 1.x protocol
#define Lin_1x_set_type()    ( LINCR|= LIN_1X )
#define Lin_1x_enable()      ( LINCR = LIN_1X|(1<<LENA)|(0<<LCMD2) )
#define Lin_1x_set_id(id)    { LINIDR &= ~LIN_1X_ID_MASK;     \
                               LINIDR |= id & LIN_1X_ID_MASK;}
#define Lin_1x_set_len(len)  { LINIDR &= ~LIN_1X_LEN_MASK;                  \
                               LINIDR |= (((len+4)<<2) & LIN_1X_LEN_MASK;}
#define Lin_get_len()        ( LINDLR & (0x0F << LRXDL0) )

// Lin_set_rx_len(len) - Automatic setting in LIN 1.x for response
// Lin_set_tx_len(len) - Automatic setting in LIN 1.x for response
```

4.2 C-macros for LIN 2.x

```
// Use LIN 2.x protocol
#define Lin_2x_set_type()    ( LINCR|= LIN_2X )
#define Lin_2x_enable()      ( LINCR = LIN_2X|(1<<LENA)|(0<<LCMD2) )
#define Lin_2x_set_id(id)    { LINIDR &= ~LIN_2X_ID_MASK;     \
                               LINIDR |= id & LIN_2X_ID_MASK;}
// Lin_2x_set_len(len) - Not available in LIN 2.1
// Lin_get_len() ------- Not available in LIN 2.1

#define Lin_set_rx_len(len)  ( LINDLR = len & (0x0F << LRXDL0) )
#define Lin_set_tx_len(len)  ( LINDLR = len << LTXDL0 )
```

4.3 Shared C-macros

```
// Use of any protocol
#define Lin_set_type(ltype)(( ltype == LIN_2X ) ?                       \
                             Lin_2x_set_type(): Lin_1x_set_type() )
#define Lin_get_type()      ( LINCR & (1<<LIN1X) )
#define Lin_set_len(len)    ( LINDLR = (len<<LTXDL0)|(len &(0x0F<<LRXDL0) )

// Miscellaneous C-macros
#define Lin_get_error_status()( LINERR )
#define Lin_set_baudrate(br)                                        \
            { LINBRRH = (unsigned char)(((unsigned short)br)>>8);\
              LINBRRL = (unsigned char) ((unsigned short)br);     }
#define Lin_sw_reset()      ( LINCR = 1<<LSWRES )
#define Lin_full_reset()    { Lin_sw_reset(); Lin_clear_enable_it(); \
                             LINBRRL = 0x00; LINBRRH = 0x00;       }

// Interrupt handling
#define Lin_get_it()                                                    \
            ( LINSIR & ((1<<LERR)|(1<<LIDOK)|(1<<LTXOK)|(1<<LRXOK)) )
#define Lin_set_enable_it()                                             \
            ( LINENIR = (1<<LENERR)|(1<<LENIDOK)|(1<<LENTXOK)|(1<<LENRXOK) )
#define Lin_clear_enable_it()                                           \
            ( LINENIR = (0<<LENERR)|(0<<LENIDOK)|(0<<LENTXOK)|(0<<LENRXOK) )
#define Lin_clear_err_it()    ( LINSIR = 1<<LERR  )
#define Lin_clear_idok_it()   ( LINSIR = 1<<LIDOK )
#define Lin_clear_txok_it()   ( LINSIR = 1<<LTXOK )
#define Lin_clear_rxok_it()   ( LINSIR = 1<<LRXOK )

// LIN commands
#define Lin_abort()     (LINCR &= ~LIN_CMD_MASK)
#define Lin_rx_header()(LINCR &= ~LIN_CMD_MASK)
#define Lin_tx_header(){LINCR &= ~LIN_CMD_MASK; LINCR|= LIN_TX_HEADER  ;}
#define Lin_rx_response(){LINCR &= ~LIN_CMD_MASK; LINCR|= LIN_RX_RESPONSE;}
#define Lin_tx_response(){LINCR &= ~LIN_CMD_MASK; LINCR|= LIN_TX_RESPONSE;}

// LIN checking
#define Is_lin_header_ready()      ( (LINSIR & (1<<LIDOK)) ? 1 : 0 )
#define Is_lin_rx_response_ready()( (LINSIR & (1<<LRXOK)) ? 1 : 0 )
#define Is_lin_tx_response_ready()( (LINSIR & (1<<LTXOK)) ? 1 : 0 )

// ID & data handling
#define Lin_get_id()          ( LINIDR & LIN_2X_ID_MASK)
#define Lin_clear_index()     ( LINSEL = (0<<LAINC)|(0x00<<LINDX0) )
#define Lin_get_data()        ( LINDAT )
#define Lin_set_data(data)    ( LINDAT = data )
```

4.4 Remark on C-macro

If a long C-macro (several lines of C-code) is too often used, it penalizes the size of generated code (each C-macro call is replaced, in-line, by the corresponding C-macro definition). Then, it will be appropriate to encapsulate it in a C-function.

5. C-functions

5.1 LIN/UART Controller Initialization

This function initializes the LIN controller and, if needed, the LIN interrupts.

Two arguments are passed to the function:

1. '`unsigned char l_type`': By construction, '`l_type`' is either **`LIN_1X`** or **`LIN_2X`**
2. '`unsigned long b_rate`': LIN Baud Rate Register (LINBRR) value

The function returns:

'`unsigned char`':	== 0 : Initialization failed, LIN type is not in accordance
	!= 0 : Initialization performed

Warning: none

5.1.1 Prototype

```
extern unsigned char lin_init (unsigned char l_type, unsigned long b_rate);
```

5.1.2 Function

```
unsigned char lin_init (unsigned char l_type, unsigned long b_rate) {

    // Pull-up on TxLIN & RxLIN (one by one to use bit-addressing)
    LIN_PORT_DIR &= ~(1<<LIN_INPUT_PIN );
    LIN_PORT_DIR &= ~(1<<LIN_OUTPUT_PIN);
    LIN_PORT_OUT |=  (1<<LIN_INPUT_PIN );
    LIN_PORT_OUT |=  (1<<LIN_OUTPUT_PIN);

    Lin_full_reset();
    Lin_set_baudrate(b_rate);

    if (l_type == LIN_1X) {
        Lin_1x_enable();
    } else if (l_type == LIN_2X) {
        Lin_2x_enable();
    } else {
        return 0;
    }
    // If LIN is interrupt driven, enable the 2 following lines
    // Lin_set_enable_it();
    // asm ("sei");

    return 1;
}
```

5.1.3 Notes & Remarks

The LIN/UART Controller allows user customization of the protocol.

- It is possible to remove the detection of the CHECKSUM field in received frames, the transmission of the CHECKSUM field or the frame time-out.
- If these features are insufficient, the LIN/UART Controller can also be initialized as a pure UART to allow maximum user freedom. It is then possible to keep the automatic features to manage the header and switch to "manual" mode (UART mode) to manage the response.
- A system mixing reception and transmission in a same response is also possible.

5.2 LIN Header Transmission

This function commands the sending of the LIN header, MASTER task of MASTER node.

Three arguments are passed to the function:

1. '`unsigned char l_type`': By construction, '`l_type`' is either '**`LIN_1X`**' or '**`LIN_2X`**'
2. '`unsigned char l_id`': LIN identifier value. In case of '**`LIN_1X`**', the coded length is transported into the LIN identifier.
3. '`unsigned char l_len`': True length (not coded), number of data bytes transported in the response. This information is not used in '**`LIN_1X`**' because it is coded in '`l_id`'.

This function returns:

'`unsigned char`':	== 0 : Initialization failed, LIN type is not in accordance != 0 : Header transmission of the header on-going

Warning: none

5.2.1 Prototype

```
extern unsigned char lin_tx_header \
       (unsigned char l_type, unsigned char l_id, unsigned char l_len);
```

5.2.2 Function

```
unsigned char lin_tx_header \
       (unsigned char l_type, unsigned char l_id, unsigned char l_len) {

     Lin_abort();   // Useful if controller is still in 'lin_tx_response'
                    // or in 'lin_rx_response' mode. Note that when these
                    // modes are running, writing in LIN registers is
                    // disabled and the ID cannot be set in the controller.
                    // (c.f. "Break-in-Data" behavior")

     if (l_type == LIN_1X) {
          Lin_1x_set_id(l_id);
          Lin_1x_set_len(l_len);
     } else if (l_type == LIN_2X) {
          Lin_2x_set_id(l_id);      // No length transported in LIN 2.X
     } else {
          return 0;
     }

     Lin_tx_header();              // Set command
     return 1;
}
```

5.2.3 Notes & Remarks

It will be essential to verify that there will be no error during the transmission.

5.3 LIN Response Reception

This function commands the reception of a LIN response, SLAVE task of MASTER or SLAVE node.

Two arguments are passed to the function:

1. '`unsigned char l_type`': By construction, '`l_type`' is either '**`LIN_1X`**' or '**`LIN_2X`**'
2. '`unsigned char l_len`': True length (not coded), number of data bytes expected in the response. This information is not used in '**`LIN_1X`**' because it is coded in '`l_id`'.

This function returns:

'`unsigned char`':	== 0 : Initialization failed, LIN type is not in accordance
	!= 0 : Response reception on-going

Warning: At the end of the reception, the LIN/UART Controller data buffer will need to be emptied (read) (c.f. '`Lin_get_response()`' function)

5.3.1 Prototype

```
extern unsigned char lin_rx_response \
       (unsigned char l_type, unsigned char l_len);
```

5.3.2 Function

```
unsigned char lin_rx_response \
       (unsigned char l_type, unsigned char l_len) {

     if (l_type == LIN_1X) {
          Lin_1x_set_type();          // Change is necessary
     } else if (l_type == LIN_2X) {
          Lin_2x_set_type();          // Change is necessary
          Lin_set_rx_len(l_len);
     } else {
          return 0;
     }

     Lin_rx_response();               // Set command
     return 1;
}
```

5.3.3 Notes & Remarks

It will be essential to verify that there will be no error during the reception.

5.4 LIN Response Transmission

This function commands the sending of a LIN response, SLAVE task of MASTER or SLAVE node.

Three arguments are passed to the function:

1. '`unsigned char l_type`': By construction, '`l_type`' is either '**`LIN_1X`**' or '**`LIN_2X`**'
2. '`unsigned char *l_data`': Internal SRAM array pointer. This array contains the data bytes to transmit.
3. '`unsigned char l_len`': True length (not coded), number of data bytes transported in the response. This information is not used in '**`LIN_1X`**' because it is coded in '`l_id`'.

This function returns:

'`unsigned char`':	== 0 : Initialization failed, LIN type is not in accordance
	!= 0 : Response transmission on-going

Warning: none

5.4.1 Prototype

```
extern unsigned char lin_tx_response \
       (unsigned char l_type, unsigned char *l_data, unsigned char l_len);
```

5.4.2 Function

```
unsigned char lin_tx_response \
       (unsigned char l_type, unsigned char *l_data, unsigned char l_len) {

unsigned char i;

     if (l_type == LIN_1X) {
          Lin_1x_set_type();          // Change is necessary
     } else if (l_type == LIN_2X) {
          Lin_2x_set_type();          // Change is necessary
          Lin_set_tx_len(l_len);
     } else {
          return 0;
     }

     Lin_clear_index();               // Data processing
     for (i = 0; i < l_len; i++) {
          Lin_set_data(*l_data++);
     }

     Lin_tx_response();               // Set command
     return 1;
}
```

5.4.3 Notes & Remarks

It will be essential to verify that there will be no error during the transmission.

5.5 Read Data Received

This function reads (empties) the reception data buffer when a LIN response had been received. This function is additional of the '`lin_rx_response()`' function.

Only one argument is passed to the function:

1. '`unsigned char *l_data`': Internal SRAM array pointer. This array will contain the data bytes received.

This function returns nothing.

Warning: none

5.5.1 Prototype

```
extern void lin_get_response (unsigned char *l_data);
```

5.5.2 Function

```
void lin_get_response (unsigned char *l_data) {

unsigned char i, l_len;

     l_len = Lin_get_len();
     Lin_clear_index();
     for (i = 0; i < l_len; i++) {
          (*l_data++) = Lin_get_data();
     }
}
```

5.5.3 Notes & Remarks

(none)

6. Firmware Packaging

Two files are delivered.

6.1 ‘lin_drv.h’ File

“lin_drv.h” file includes the “config.h” file where are defined the main clock frequency, the LIN baudrate used and the device-specific I/O header file.

In “lin_drv.h” file, we will find:

1. Definitions:
   - ‘Bit Time’ definitions,
   - ‘Configuration’ definitions.
2. C-macros:
   - ‘LIN 1.x’ C-macros,
   - ‘LIN 2.x’ C-macros,
   - Shared C-macros.
3. Prototypes of functions:
   - ‘lin_init()’
   - ‘lin_tx_header()’,
   - ‘lin_rx_response()’,
   - ‘lin_tx_response()’,
   - ‘lin_get_response()’.

6.2 ‘lin_drv.c’ File

“lin_drv.c” file includes the “lin_drv.h” file.

In “lin_drv.c” file, we will only find the C-functions:

1. ‘lin_init()’
2. ‘lin_tx_header()’,
3. ‘lin_rx_response()’,
4. ‘lin_tx_response()’,
5. ‘lin_get_response()’.

7. LIN Communication Management

This paragraph shows a LIN communication management based on a polling method. Management based on interrupts needs to build its own LIN message object descriptor. But many items discussed below can be reused if interrupt method is preferred.

7.1 Using LIN Status & Interrupt Register

To manage LIN communication, only reading LIN Status & Interrupt Register (LINSIR) is necessay. Note that this register houses all the interrupt sources. Look at this register:

7	6	5	4	3	2	1	0	
LIDST2	LIDST1	LIDST0	LBUSY	LERR	LIDOK	LTXOK	LRXOK	LINSIR

There can be only four events:

1. **ID transmission completed / ID received**: When the ID flag rises (LIDOK), this means two things (both are important):
   – The LIN header phase has been performed, the header transmission was completed (MASTER node) or an header was received (SLAVE node),
   – The LIN response phase must take place as soon as possible if the node is concerned with the received LIN ID.
2. **Response reception completed**: When the Rx flag rises (LRXOK), this means that the LIN response reception is now completed and there was no detected error.
3. **Response transmission completed**: When the Tx flag rises (LTXOK), this means that the LIN response transmission is now completed and there was no detected error.
4. **Error**: If the error flag rises (LERR), this root cause is contained in the LIN Error Register (LINERR). It is important to note when an error rises. The meaning of the error often depends on it.

7.2 LIN Slave Example

```
//------------------------------------------------------------------------
// T I T L E : lin_slave_example.c
//------------------------------------------------------------------------

//----- I N C L U D E S
#include "config.h"
#include "lin_drv.h"

//----- D E C L A R A T I O N S
#define   LIN_ID_0   0x12
#define   LIN_ID_1   0x13

#define   LEN_0      1
#define   LEN_1      2

unsigned char lin_motor_contol[LEN_0];
unsigned char lin_node_status[LEN_1];

volatile unsigned char lin_slave_err_nb = 0;

//----- F U N C T I O N S

//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_id_task function of "lin_slave_example.c"
```

```
//
// The LIN ID received must be checked and compared to
//     the ones that the node must manage.
//
void lin_id_task (void) {
     switch (Lin_get_id()) {
          case LIN_ID_0:
               lin_rx_response(LIN_2X, LEN_0);
               break;
          case LIN_ID_1:
               lin_tx_response(LIN_2X, lin_node_status, LEN_1);
               break;
          default:
               // ID: absent/refused
               break;
     }
}


//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_rx_task function of "lin_slave_example.c"
//
//     - Save response data
//     - Update application signals.
//
void lin_rx_task (void) {

unsigned char l_temp;

     lin_get_response (lin_motor_contol);

     // Update of the application signals
     l_temp = lin_motor_contol[0] & 0x03;
     if((l_temp == 0x01) || (l_temp == 0x03)) {
          PORTB |= 1<<PORT0;
     } else {
          PORTB &= ~(1<<PORT0);
     if(l_temp == 0x02) {
          PORTB |= 1<<PORT1;
     } else {
          PORTB &= ~(1<<PORT1);
     }
}


//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_tx_task function of "lin_slave_example.c"
//
//     - Update buffer array with application signals
//       to be ready for the next transmission
//
void lin_tx_task (void) {

     // Update of the application signals
     lin_node_status[0] = PINB & 0x03;
     lin_node_status[1] = lin_slave_err_nb;
}


//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_err_task function of "lin_slave_example.c"
```

```
//
//    - If LIN error, increment the node error number
//
void lin_err_task (void) {

     // Global variable update
     lin_slave_err_nb++;
}


//. . . M A I N ( ) . . . . . . . . . . . . . . . . .
//
// main function of "lin_slave_example.c"
//
//     In a 'no end' loop, do:
//       - if LIN_ID_0 (Rx response 1 Byte )
//            . PORT-B.0 = DC motor command -> clockwise
//            . PORT-B.1 = DC motor command -> counterclockwise
//       - if LIN_ID_1 (Tx response 2 bytes)
//            . Byte[0] = motor status
//            . Byte[1] = node error number
//
int main (void) {

     // Port B setting for motor control
     DDRB |= 1<<PORTB0; DDRB |= 1<<PORTB1;
     PORTB &= ~(1<<PORTB0); PORTB &= ~(1<<PORTB1);

     // LIN Initialization
     lin_init((unsigned char)LIN_2X, ((unsigned short)CONF_LINBRR));

     // No End Loop
     while (1) {
          switch (Lin_get_it()) {
               case LIN_IDOK:
                    lin_id_task();
                    Lin_clear_idok_it();
                    break;
               case LIN_RXOK:
                    lin_rx_task();
                    Lin_clear_rxok_it();
                    break;
               case LIN_TXOK:
                    lin_tx_task();
                    Lin_clear_txok_it();
                    break;
               case LIN_ERROR:
                    lin_err_task();
                    Lin_clear_err_it();
                    break;
               default:
                    break;
          }
     }
return 0;
}
```

Program size (compiler option: -Os): **Code** = 816 Flash bytes, **Data** = 4 RAM bytes.

7.3 LIN Master Example

A real time task must be added to send the headers. For instance the Timer0 will be used to send them.

Else the program looks like the LIN slave example.

```
//----------------------------------------------------------------------------
// T I T L E : lin_master_example.c
//----------------------------------------------------------------------------

//----- I N C L U D E S
#include "config.h"
#include "lin_drv.h"
#include <avr/interrupt.h>       // Use AVR-GCC library

//----- D E C L A R A T I O N S
#define   LIN_ID_0  0x12
#define   LIN_ID_1  0x13

#define   LEN_0     1
#define   LEN_1     2

#define   MAGIC_NUMBER   77

unsigned char lin_motor_command[LEN_0];
unsigned char lin_slave_node_status[LEN_1];

volatile unsigned char lin_master_err_nb = 0;

volatile unsigned char rtc_tics = 0;

//----- F U N C T I O N S

//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_id_task function of "lin_master_example.c"
//
// The LIN ID received must be checked and compared to
//     the ones that the node must manage.
//
void lin_id_task (void) {
     switch (Lin_get_id()) {
          case LIN_ID_0:
               lin_tx_response(LIN_2X, lin_motor_command, LEN_0);
               break;
          case LIN_ID_1:
               lin_rx_response(LIN_2X, LEN_1);
               break;
          default:
               // ID: absent/refused
               break;
     }
}


//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_rx_task function of "lin_master_example.c"
//
//     - Save response data
//     - Update application signals.
//
void lin_rx_task (void) {
        lin_get_response (lin_slave_node_status);

        // If command OK and no error on slave node
        //      - PORTB[6] = 0
        //      - else PORTB[6] = 1
        if (lin_slave_node_status[0] != lin_motor_command[0]) {
            PORTB |= 1<<PORT6;
        } else if (lin_slave_node_status[1] != 0){
            PORTB |= 1<<PORT6;
        } else {
            PORTB &= ~(1<<PORT6);
        }
}


//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_tx_task function of "lin_master_example.c"
//
//      - Update buffer array with application signals
//        to be ready for the next transmission
//
void lin_tx_task (void) {

        // Update of the application signals (active-low switches)
        lin_motor_command[0] = (~PINB) & 0x03;
}


//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// lin_err_task function of "lin_master_example.c"
//
//      - If LIN error, increment the node error number
//
void lin_err_task (void) {

        // Global variable update
        lin_master_err_nb++;
}


//. . . . . . . . . . . . . . . . . . . . . . . . .
//
// OCR0A interrupt service routine of "lin_master_example.c"
//
//      AVR-GCC declaration (default):
//           ISR(TIMER0_COMPA_vect)
//      IAR declaration:
//           #pragma vector TIMER0_COMPA_vect
//           __interrupt void timer0_compa_vect(void)
//
//      The appropriate LIN header is sent following the rtc_tic value
//
ISR(TIMER0_COMPA_vect) {

        rtc_tics++;

        if((rtc_tics & 0x01) == 0x01) {
            lin_tx_header((unsigned char)LIN_2X, (unsigned char)LIN_ID_0, 0);
        } else {
            lin_tx_header((unsigned char)LIN_2X, (unsigned char)LIN_ID_1, 0);
        }
}
```

```
//. . . M A I N ( ) . . . . . . . . . . . . . . . .
//
// main function of "lin_master_example.c"
//
//     In a 'no end' loop, do:
//       - if LIN_ID_0 (Tx response 1 Byte)
//             . PORT-B.0 = DC motor command -> clockwise
//             . PORT-B.1 = DC motor command -> counterclockwise
//       - if LIN_ID_1 (Rx response 2 bytes)
//             . Byte[0] = motor status
//             . Byte[1] = slave node error number
//     Timer0 ensures that each header is sent in 20ms intervals.
//
int main (void) {

     // Port B setting for motor control
     // PORTB0 & PORTB1 switches active low with internal pull-up
     DDRB &= ~(1<<PORTB0); DDRB &= ~(1<<PORTB1);
     PORTB |= 1<<PORTB0; PORTB |= 1<<PORTB1;

     // Port B setting for motor status flag
     DDRB |= 1<<PORTB6; PORTB &= ~(1<<PORTB6);

     // LIN initialization
     lin_init((unsigned char)LIN_2X, ((unsigned short)CONF_LINBRR));

     // Timer0 & INT initialization (no output)
          // Timer0 Reset
          TIMSK0 = 0; TCCR0B = 0; TCCR0A = 0; TCNT0 = 0; OCR0A = 0;
          TIFR0 = ((1<<OCF0A) | (1<<TOV0));
          // No output, mode 2, 10 ms interrupt
          TCCR0A = 0x02; OCR0A = MAGIC_NUMBER; TCCR0B = 0x05;
          // Timer0 compare A interrupt enable
          TIMSK0 |= 1<<OCIE0A; asm ("sei");

     // No End Loop
     while (1) {
          switch (Lin_get_it()) {
               case LIN_IDOK:
                    lin_id_task();
                    Lin_clear_idok_it();
                    break;
               case LIN_RXOK:
                    lin_rx_task();
                    Lin_clear_rxok_it();
                    break;
               case LIN_TXOK:
                    lin_tx_task();
                    Lin_clear_txok_it();
                    break;
               case LIN_ERROR:
                    lin_err_task();
                    Lin_clear_err_it();
                    break;
               default:
                    break;
          }
     }
return 0;
}
```

Program size (compiler option: -Os): **Code** = 946 Flash bytes, **Data** = 5 RAM bytes.

7.4 Configuration Header File

```
//-----------------------------------------------------------------------
// config.h for "lin_slave_example.c" or "lin_master_example.c"
//-----------------------------------------------------------------------

//----- I N C L U D E S
#include <avr/io.h>                 // Use AVR-GCC library

//----- D E C L A R A T I O N S
     // Use an external crystal oscillator for "lin_master_example.c"
#define  FOSC          8000         // In KHz
#define  LIN_BAUDRATE  19200        // In bit/s
```

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature