AVR322: LIN v1.3 Protocol Implementation on Atmel AVR Microcontrollers

AVR® Microcontrollers

Application Note

1. Description

The Controller Area Network (CAN) has seen enormous success in automotive body and powertrain control systems. However, there is a change in emphasis arising in the automotive industry in which CAN is seen as too powerful and expensive for simple digital sensor/actuator body control applications. The emerging protocol Local Interconnect Network (LIN – Specification 1.3) has been proposed for such applications. LIN is a simple protocol whose specification has been aimed at low-end microcontrollers that have a USART or USI peripheral. The LIN protocol is introduced in this application note, along with its implementation on Atmel Automotive AVR microcontroller. USART implementations are described for LIN Master and Slave drivers, whose C-source code is available from Atmel.

1.1 In-Vehicle Network Applications

Today, electronic components and systems account for over 20% of the cost of a high end passenger car. In-vehicle networks are already successfully applied to body and powertrain control systems. This figure is expected to grow significantly over the next ten years with the introduction increasingly more complex control systems such as Drive-by-Wire and multimedia systems giving access to the internet. As far as in-vehicle control networking technology is concerned, an emerging growth area is in the adoption of a low cost body control sub-bus that complements CAN.

Many body control functions are often simple digital on/off operations such as activating lights, wipers, windows etc. These are considered soft requirement real time systems that do not necessarily need the response that can be provided by CAN. Additionally, there is also the desire in the industry to integrate a network node into a sensor or actuator. LIN is currently the candidate technology for these applications since it is a complementary technology to CAN but at a much lower cost. Table 1 briefly compares CAN and LIN protocols.

7548A–AVR–12/05

Table 1-1. A Comparison of the Main Features of CAN and LIN

	CAN	**LIN**
Affiliation	Robert Bosch® GmbH (www.can.bosch.com) CAN in Automation (www.can-cia.com)	The LIN Consortium (www.linsubbus.org)
Maximum baud rate	1Mbaud - High Speed CAN ISO-11898	20Kbaud Enhanced ISO-9141 (ISO-K)
Network Topology	Bus	Bus
Network Access Method	Non-Destructive Bit-wise Arbitration	Master initiates transmission, therefore no arbitration necessary
Number of nodes	Usually limited by physical layer or higher layer protocol to 128 or 64	16 nodes (1 Master, 15 Slave)
Number of wires	2	1

1.2 LIN Applications

The LIN protocol was suggested in 1999 as a low cost sub-bus system to complement CAN in applications such as:

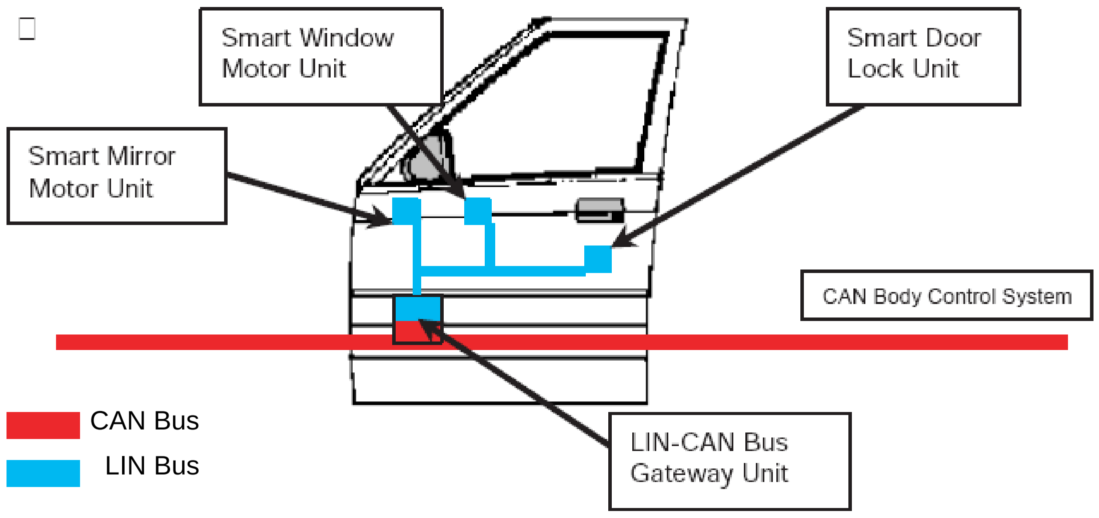
- Vehicle Roof (Rain Sensor, Light Sensor, Light Control, Sun Roof)
- Vehicle Doors (Mirror, Central Locking, Mirror Switch, Window Lift)
- Engine (Sensors, Small Motors, Steering Wheel, Cruise Control Switches, Wiper, Turn Signal, Radio, Climate Control)
- Seat (Seat Position Motors, Seat heater, Occupancy Sensor)

The main features of LIN are:

- Based on USART or USI
- Physical Layer, enhanced ISO-9141
- Baud rate up to 20Kbaud
- 16 LIN ID, 8 Data Bytes
- Master/Slave communication

The use of LIN gives increases the number of vehicle body control network architecture as shown in Figure 1-1. Figure 1-1 shows an example LIN sub-bus for control of a car door electrical sub-system. It can be seen that the main body control CAN network interfaces with the rest of the door system via a LIN-CAN Bus Gateway Unit. A single LIN signal line is used to connect the LIN-CAN Bus Gateway Unit to the Smart Door Lock Unit, Smart Mirror Motor Unit and Smart Window Motor Unit. This would result in a total of three signal wires, which is at least five less than the traditional implementation. Therefore the cost saving enjoyed comes from the reduction in wiring offset by the additional cost of the three additional low-end microcontrollers.

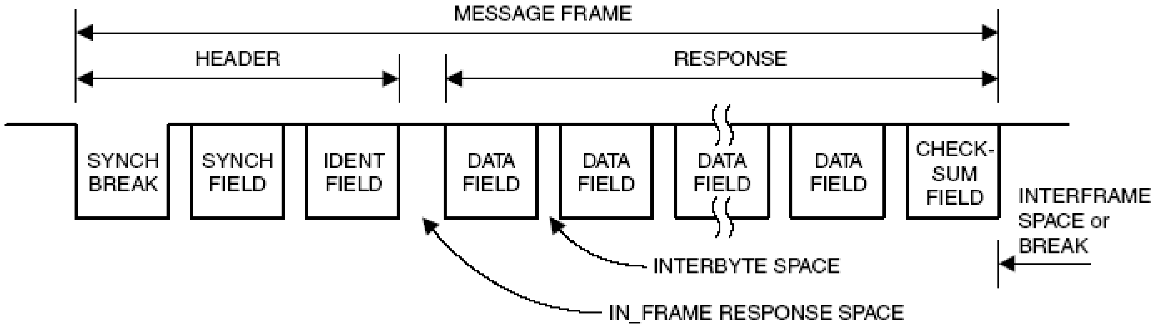
Figure 1-1. Typical Use of a Combination of LIN Sub-bus Along with the CAN bus in the Localized Area of Car Door Control.



1.3 LIN Protocol v1.3

A typical LIN frame is shown in Figure 1-2 and consists of the following components:

Figure 1-2. A Typical LIN Frame Consisting of Synch Break, Synch Field, Identifier, Data Field and Checksum.



Each byte of data follows a standard USART format of 10 bits (i.e. start bit, eight data bits, stop bit)

Sync Break (LIN Master Task)
- At least 13 bits -Signifies start of frame

Sync Field (LIN Master Task)
- 1 bit - Synch Delimiter
- Hex 55 (8 bit data, start & stop bits) - Allows slaves to synchronize

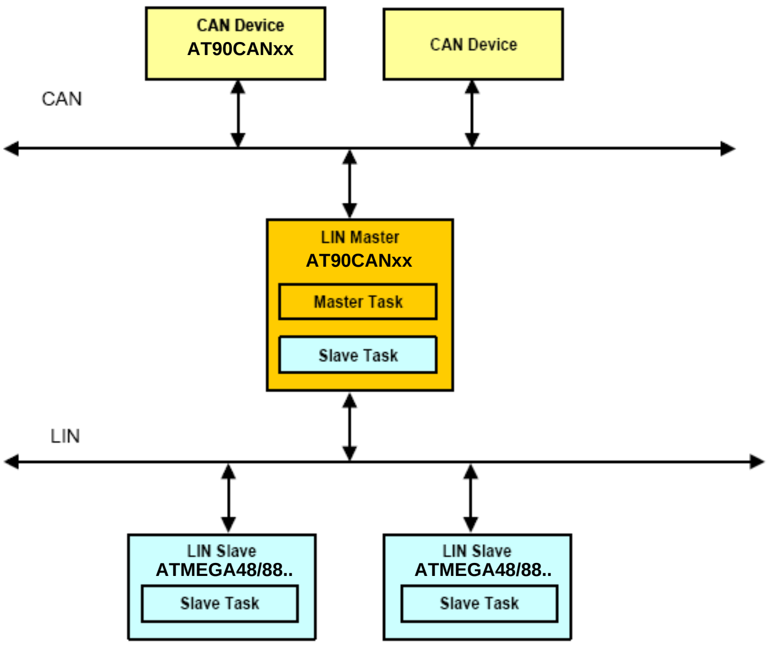
Indentifier Field (LIN Master Task)
- 8 bit data, start & stop bits
- Bits 0 to 3 - LIN ID
- Bits 4 to 5 - Length Control
- Bits 6 to 7 – Parity

Data Field (LIN Slave Task)
- 2, 4 or 8 bytes (each 8 bit data, start & stop bits)

Checksum 8 bits (LIN Slave Task)

The communication concept consists in one Master and multiple Slave tasks as shown in Figure 1-2 and Figure 1-3. The Master Task transmits the Message Header which consists of the Synch Break, Synch Field and Identifier Field. The Slave Task transmits Message Response which consists of the Data Field and Checksum Field. The Master Task is always transmitted by the Master Node, whilst the Slave Task can be transmitted by any of the nodes on the LIN subnet. The Master Node transmits the Slave Task when it is necessary to send data and receives via the Slave Task when it is necessary to receive data.

Figure 1-3. Typical Application of an AVR Microcontroller in a LIN Sub-bus System



1.4 Message Header

1.4.1 Synch Break

The Synch Break signifies the beginning of a LIN frame and is always generated by the Master Task. The Synch Break tells the Slave Task to prepare for the Synch Field. The Synch Break is twofold :

- a dominant (low) level that should be a minimum of 13 bit-times (Tbit, based on Master time base) which is followed by
- one recessive (high) period that should be in the range one to four Tbit (Synch Delimiter).

The length of the Synch Field has been chosen to be at least 13 Tbit so that LIN Slave Tasks can distinguish between a valid Synch Break and the maximum possible allowed sequence of dominant bit within a data frame (i.e. 9 Tbit). The Slave device detects the Synch Break by receiving 11 Tbit at dominant level (Tbit, based upon the slave's local time base).

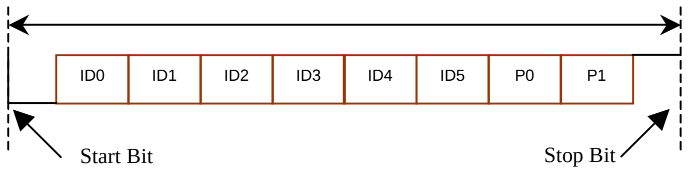
1.4.2 Synch Field

The Synch Field contains the signalling required for the slaves to synchronize with the Master's clock. The Synch Field is a byte containing the data "0x55" giving a waveform with five falling edges and five rising edges. These edges can be used during synchronization to tune the slave node transmission and reception speed to match the Master node. Therefore the bit time is found by measuring the time of either 5 rising edges or 5 falling edges and dividing by eight (logical shift by 3 bits).

1.4.3 Identifier Field

The Identifier Field contains information about the contents and length of a message. This field is divided into three sections as shown below in Figure 1-4; identifier bits (four bits), length control bits (two bits) and parity bits (two bits).

Figure 1-4. Identifier Field



The Identifier bits (ID0 to ID3) represent the identifier of the node who is to respond in the Message Response part of the frame. Thus there can be up to 16 nodes on a LIN subnet; one master and up to 15 slaves. The Identifier Field therefore describes the content of the message and not the destination.

Table 1-2. Number Bytes in Data Field as Dictated by Length Control for LIN v1.3

ID5	ID4	NData (no. of bytes)
0	0	2
0	1	2
1	0	4
1	1	8

The final two bits in the Identifier Field are parity bits which are defined by a mixed parity algorithm. This ensures that the Identifier Field will never consist of an all Dominant or all Recessive bit pattern. It must be noted that the parity check only detects errors, it does not correct them.

The parity check bits are calculated by the following mixed parity algorithm:

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$

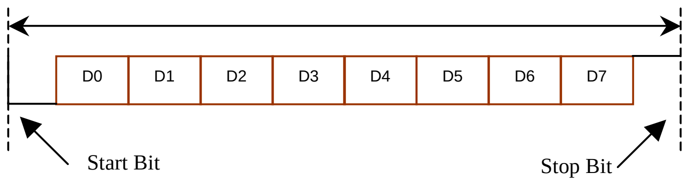
$$P1 = ID1 \oplus ID3 \oplus ID4 \oplus ID5$$

1.5 Message Response

1.5.1 Data Field

The Data Field contains two, four or eight bytes of data, each consisting of one stop bit, eight data bits and one stop bit. Transmission is done with the LSB first. The Data Field is written by the responding slave task. Since there is no bus arbitration, only one slave task should be allowed to respond to each identifier. All other slave tasks are limited to reading the response and act accordingly. The format of a single byte within the Data Field is shown in Figure 1-5.

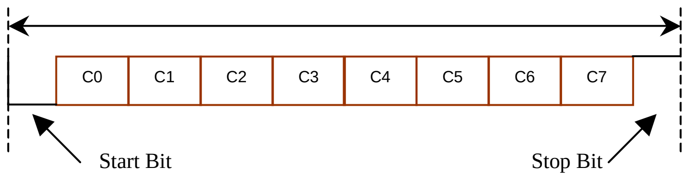
Figure 1-5. A Single Byte of the Data Field



1.5.2 Checksum Field

The last field in the Message Frame is the Checksum Field. This byte contains the inverted modulo-256 sum of all data bytes within the Data Field. The sum is calculated by doing an “Add with carry” on all data bytes and then inverting the answer. The properties of the inverted modulo-256 sum are such that if this number is added to the sum of all data bytes, the result will be “0xFF”. The format of the Checksum Field is shown in Figure 1-6.

Figure 1-6. Checksum Field



It is important that the carry bit is added to the LSB during the addition of each byte in the “Add with Carry” operation so that:

0xFF + 0x01 = 0x01

1.5.3 Error Detection

The following methods of error detection are outlined in the LIN specification v1.3:

Bit Error detection is concerned with any unit that is sending a bit on the LIN bus. A Bit Error occurs when a monitored bit value is different to the bit value sent. This is the responsibility of either the *Master* or *Slave* node that has transmitted the data.

Checksum Error detection of the Data Field as described previously. This is the responsibility of either the *Master* or *Slave* node that has received the data within the Data Field.

Identifier Parity Error detection for the Identifier Field as described previously. This is the responsibility of the *Slave* node receiving the Message Header.

Slave Not Responding Error detection is concerned with ascertaining when a Slave's Message Response does not respond to a Master's Message Header. This is defined in the LIN specification v1.3 as frame that does not complete its transmission within T_{FRAME_MAX} after frame initiation (i.e. start of Synch Break). This works out to be:

– Data Field 2 Bytes = 91 Tbit
– Data Field 4 Bytes = 119 Tbit
– Data Field 8 Bytes = 174 Tbit

This is the responsibility of the *Master* node or any *Slave* node that is expecting to receive that particular Message Response.

Physical Bus Error detection is carried out by the Master if no valid message can be generated, i.e. bus shorted to ground. This is the responsibility of both *Master* and *Slave* nodes.

Inconsistent Synch Field Error detection is concerned with ascertaining when the edges of the Synch Field go outside of tolerance. This is the responsibility *Slave* nodes only.

There are also error situations that are not covered by the LIN specification such as reception of the incorrect number of data bytes in the Data Field. Too few data bytes received should result in a Slave Not Responding Error. Too many data bytes should result in a Checksum Error.

1.5.4 Error Signalling And Recovery Time

The direct signalling of errors is not possible due to the master concept. Errors should be locally detected and provided in the form of diagnostics on request.

1.5.5 Fault Confinement

The LIN specification specifies that the Master node shall handle most (if not all) of error detection, error recovery and diagnostics. Fault confinement depends upon the system requirements and therefore only basic support is specified. Therefore much of the fault confinement needs to be specified by the user particularly for their applications needs.

1.5.5.1 Master Control Unit Fault Confinement

The Master shall detect the following error situations:

- Bit Error or Identifier Parity Error in the Master Task
- Slave Task in Master Control Unit -> Slave Not Responding Error and Checksum Error.

The Master Control Unit Fault Confinement is not specified in LIN specification v1.3. However a recommended practice is outlined. The recommended practice states that a *Master Task* sending data must detect a Bit Error in the Synch or Identifier Fields. The Master Control Unit should keep track of any transmission error by incrementing the ***MasterTransmitErrorCounter***. It is increased by 8 each time a bit in the Synch or Identifier Fields are locally corrupted. It is decreased by 1 (not less than 0) each time both Fields are read back correctly. If the counter is incremented beyond **C_MASTER_TRANSMIT_ERROR_THRESHOLD** (recommended default value = 64) it is assumed a massive disturbance has occurred on the LIN bus and an error handling procedure should be taken in the application code level.

The *Message Response (i.e. Data and Checksum Fields)* in the Master Control Unit sending data will detect Bit Errors. The *Slave Task* in the Master Control Unit receiving data will detect either Slave Not Responding Error or a Checksum Error. When a Bit Error occurs, ***MasterTransmitErrorCounter*** is incremented as described previously. When Slave Not Responding Error or Checksum Error is detected when expecting or receiving data, ***MasterReceiveErrorCounter*** is increased by 8. It is decreased by 1 (not less than 0), each time the slave response is received correctly with no errors. If ***MasterReceiveErrorCounter*** is incremented beyond **C_MASTER_RECEIVE_ERROR_THRESHOLD** (recommended default value = 64), it is assumed that the addressed slave has malfunctioned and an error handling procedure should take place in the application code level.

When a Slave Not Responding Error (i.e. detected by the Master Control Unit) occurs, the LIN specification v1.3 does not specify whether or not retransmissions should take place. This is a decision that should take place within the application code level.

1.5.5.2 Slave Control Unit Fault Confinement

The Slave node shall detect the following error situations:

- Bit Error detection when transmitting data via Slave Task.
- Checksum Error when receiving data via the Slave Task.
- Identifier Parity Error when receiving the Message Header from the Master node.

In addition to these errors, two other errors should be detected but are dependent upon the application of the specific LIN drivers:

- Slave Not Responding Error should be detected when a Slave node is expecting data to come from another Slave node.
- Inconsistent Synch Field Error should be detected when the received Synch Field is out of tolerance.

The Slave Control Unit Fault Confinement is not specified in LIN specification v1.3. However a recommended practice is outlined which is similar to the Master Control Unit's method. It is not outlined to the same extent as the Master Control Unit. However, a suggested method is proposed here, based on that of the Master Control Unit.

The suggested practice is that a *Message Response* from a Slave Control Unit sending data must detect a Bit Error in the Data or Checksum Fields. The Slave Control Unit should keep track of any transmission error by incrementing the ***SlaveTransmitErrorCounter***. It is increased by 8 each time a bit in the Data or Checksum Fields are locally corrupted. It is decreased by 1 (not less than 0) each time both Fields are read back correctly. If the counter is incremented beyond ***C_SLAVE_TRANSMIT_ERROR_THRESHOLD*** (recommended default value = 64) it is assumed a massive disturbance has occurred on the LIN bus and an error handling procedure should be taken in the application code level.

The Slave Control Unit receiving Message Response data will detect an Identifier Parity Error, Slave Not Responding Error or a Checksum Error. When Slave Not Responding Error or Checksum Error is detected when expecting or receiving data, ***SlaveReceiveErrorCounter*** is increased by 8. It is decreased by 1 (not less than 0), each time the slave response is received correctly with no errors. If ***SlaveReceiveErrorCounter*** is incremented beyond **C_SLAVE_RECEIVE_ERROR_THRESHOLD** (recommended default value = 64), it is assumed that the addressed slave has malfunctioned and an error handling procedure should take place in the application code level.

1.5.6 Reserved Identifiers

The Reserved Identifiers are reserved for specific features as outlined below:

Command Frame Identifier: Two COMMAND Frame Identifiers are reserved to broadcast general command requests for service purposes from the Master to the Slaves. Each frame has 8 bytes distinguished by the following reserved Identifiers; '0x3C' Master Request Frame and '0x7D' Slave Response Frame. The Master Request Frame is used to send commands and data to the slave nodes. The Slave Response Frame triggers on slave node (addressed by a prior download frame) to send data to the Master node.

If D7 of the 1st data byte of the Data Field is zero, then the frames usage is reserved for definition by the LIN consortium, otherwise the usage is free for custom definition.

Sleep Mode Command: The Sleep Mode Command is a broadcast message to put all slave nodes into sleep mode. The Sleep Mode Command is a Command Frame from the Master with the 1st byte in the Data Field as 0x00.

Extended Frames: A frame with Identifier Field as '0xFE' is reserved for the user defined message formats. A frame with Identifier Field as '0xBF' is reserved for future definition by the LIN consortium.

1.6 Atmel AVR Microcontrollers

The next parts of this document describes the current LIN v1.3 implementation for ATMEL AVR microcontrollers. Both Master and Slave nodes are supported. Below is the MCU list wich are currently supported.

Table 1-3. Currrently Supported AVR microcontrollers

MCU	PERIPHERAL USAGE		NOTE
ATMEGA48/88/168	USART_0		
AT90CAN128	USART_0	USART_1	Dual LIN available
ATTINY45/85	USI		LIN Slave Only

Moreover, any AVR device with at least one free USART or USI may be supported after minor source code modification (include files, register definition, timer usage,...). The drivers presented in this application note are concerned with implementation on the USART device of the AVR microcontrollers only.

1.6.1 Oscillator Considerations

For LIN applications, the internal RC oscillator is an important feature. This means that an external crystal is not required and therefore leads to cost saving. The accuracy of the internal oscillator of the AVR microcontrollers is +/-1% which is adequate for USART LIN Slave implementations (LIN Specification required accuracy is +/-1.5% after re-synchronisation). However, for a USART LIN Master, an external crystal oscillator must be used since at least +/-0.5% accuracy is required.

Further details can be found in the AVR datasheets available on the Atmel website. All software driver examples use values for timing for an 8MHz oscillator. More information on the internal RC oscillator runtime calibration can be found on the Atmel website (AVR054).

1.6.2 File list overview

Below is the main file list for the typical delivered software package:

- LIN_DRV_USART.C: driver source file for USART implementation
- LIN_DRV_USART.H: driver header file for USART implementation
-
- LIN_LIB.C: library source file
- LIN_LIB.H: library header file
-
- CONFIG.H: general configuration file (baud rate and system clock selection)

For a Slave LIN implementation (no header transmission node)

- SLAVE_LIN.C: "slave manager" file
- SLAVE_LIN.H: "slave manager" header file & option selection
-

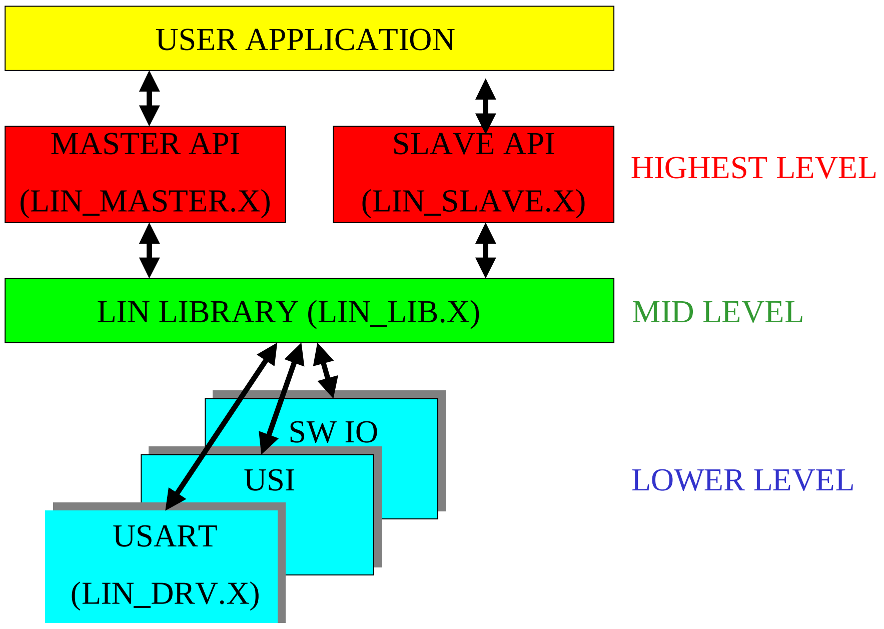
Or for Master LIN implementation (header transmission capability)

- MASTER_LIN.C: "master manager" file
- MASTER_LIN.H: "master manager" file

1.6.3 Software Implementation

In the delivered package, three software layers have been defined:

Figure 1-7. Software Layers



1.6.3.1 *LOW LEVEL DRIVERS "LIN_DRV.x" (mandatory)*

They are usually dedicated to a particular peripheral, i.e. USART, USI, … They are not meant to be called directly from by the user application. Instead they are called by the LIN library (intermediate level). The driver layer is in charge of controlling the chosen "LIN" peripheral, updating the LIN controller status through its internal Finite State Machine and reports errors/bus events to the upper layer. Main tasks are:

- SynchBreak detection
- SynchroByte decoding / Clock Synchronization (slave)
- Identifier decoding, dlc extraction, parity error detection
- Frame / Bit error detection
- Data reception and transmission
- Checksum generation / verification
- Update LIN controller status
- Time-out detection

1.6.3.2 *LIN LIBRARY "LIN_LIB.X" (mandatory)*

The LIN library functions and variables can be called / handled directly from the user application. It is possible to use also the proposed API instead. The library is implementation independent i.e. it is the same for USART, USI,…Main tasks are:

- LIN controller initialisation (baud rate,…)
- Wakeup / sleep mode handler
- User command " interpreter "
- Controller status

1.6.3.3 *LIN API "SLAVE / MASTER_LIN.X "(optional)*

Moreover, to make easier user's job a LIN API has been added into the software package. This layer is optional and could be removed or replaced by any other higher layer. Since LIN is most of the time implemented as a Time Trigger protocol (scheduler), related TT functions have been added:

- Data buffer handling (retrieve, copy data to/from application buffer)
- LIN identifier matching
- Error counters
- Sleep mode control (No Bus activity, Sleep Command)
- Ensure that the LIN slave controller is always ready to accept a new frame

1.6.3.4 *LIN FRAMES TYPES*

Two kind of frames have been considered:

- "REMOTE_DATA_FRAME" is the name for frame whose header is sent by the unique network master node and whose response is transmitted by the current slave node. It is a data request frame. In that case, Master node expects a response from a particular slave connected to the network. The unique addressed slave node will reply with a prepared response.

- "STANDARD_DATA_FRAME" is the name for frame whose header is transmitted by the master and response is transmitted by another external node (master node or another slave node). The current slave node will just capture the response data.

Note that a "REMOTE_DATA_FRAME" is seen as a "STANDARD_DATA_FRAME" for a slave node, which is not addressed by the current transaction.

1.7 LIN Library application

1.7.1 Board declaration

According to the selected MCU, different files are included. The file “BOARD_EXAMPLE.H” has to be modified or replaced by the user.

1.7.2 Library Options

To reduce code size, following option can be enable/disable in “MASTER_LIN.H” and “SLAVE_LIN.H” header files :

- #define _TIMOUT_DETECTION

If defined, timeout detection is active for frame header and response. Time Out detection function needs one timer. In the delivered package, timer 2 is used. “TIMER2_LIB.C” and “TIMER2OVF_ISR.C” files should be added to the project.

- #define _SLEEP_TIMOUT_DETECTION

If enabled, sleep mode is activated after transmitted a sleep message frame. WakeUp frame can be sent and/or received. To correctly use this function, the tick function has to be called at TBit rate through an available timer. In the delivered package, “TIMER1OVF_ISR.C” file shows an example. Time before sleep time out detection is adjusted by ‘SLEEP_TIMOUT’ value in “MASTER_LIN.H” or “SLAVE_LIN.H” file.

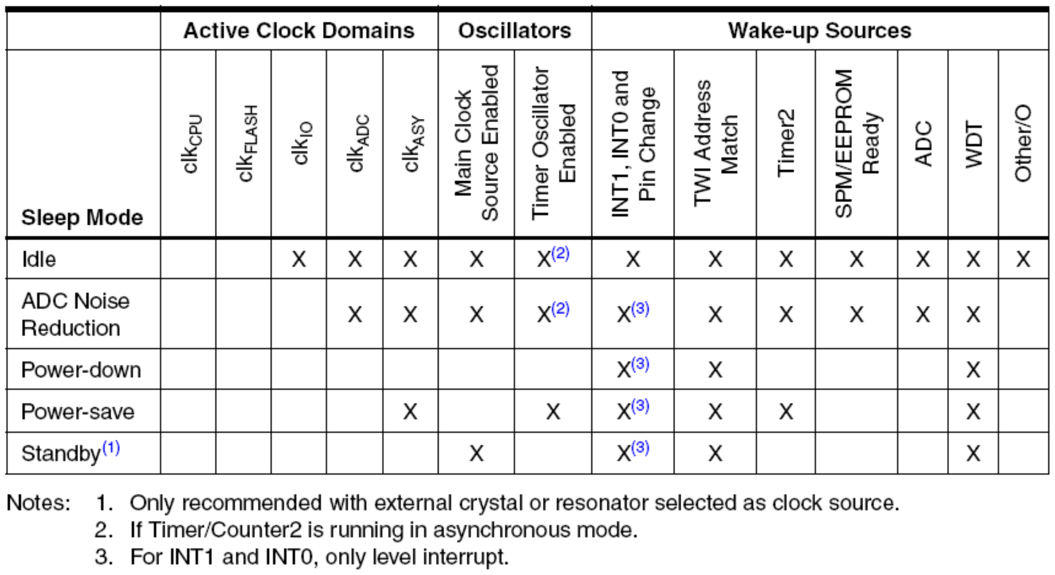
When a sleep message frame is sent or no LIN activity is detected on the LIN network, the corresponding flag LINSleepFlag or NoBusActivityFlag is switched to ‘1’. Then the user is free to select appropriate sleep mode in the main function.

With AVR microcontroller, five sleep modes are available (see datasheet for more information):

- Idle Mode
- ADC Noise reduction.
- Power Down Mode
- Power Save Mode.
- Standby Mode.

See below active clock domains and wake up sources in the different sleep modes.

Table 1-4. Clock domains and Wake-up sources in different sleep modes



	Active Clock Domains					Oscillators		Wake-up Sources						
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/O
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	X	
Power-down								X(3)	X				X	
Power-save					X		X	X(3)	X	X			X	
Standby(1)						X		X(3)	X				X	

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

- #define _COUNTING_ERRORS_ENABLE

If defined, counting errors are updated every time a LIN error is detected.

One other option available only in “SLAVE_LIN.H” is :

- #define _RUN_TIME_RC_CALIBRATION_ENABLE

If defined, internal RC oscillator run time calibration is enabled. One capture input is necessary for this feature (refer to RUN_TIME_RC_CALIBRATION Application Note on the Atmel website : AVR054).

1.7.3 Library Configuration

Most important LIN options are configured in “CONFIG.H” file :

- MCU selection, can be either :
  – AT90CAN128 (Two USARTs)
  – ATMEGA48
  – ATMEGA88
  – ATMEGA168

- “FOSC” is the nominal system frequency :
  – It has to be defined in kHz. Currently only “8000” is supported. Two sources are available:

- Internal RC Oscillator

Only slave nodes can run on the internal RC oscillator (8.0MHz). A runtime RC oscillator calibration routine is available to compensate temperature / power voltage variation (refer to application note AVR054 for more information).

- External crystal Oscillator

Both slave and master nodes can use an external crystal for a stable clock. An 8MHz external crystal is recommended for the master node to ensure a maximum frequency variation of +/-0.5%.

- “LIN_BAUDRATE”
  - The LIN baud rate in baud. Only most popular LIN baud rates are supported (19200/9600/4800/2400). For other baud rate #define values have to be entered manually by the user in “LIN_BDR.H” for his particular needs.
- LIN_CONFIG
  - LIN controller mode :
- 1 : Slave controller.
- 0 : Master controller.
- USART selection :
  - “USE_UART0” has to be defined if USART0 of ATMEGA48/88/168 and AT90CAN128 is used as LIN port.
  - “USE_UART1” has to be defined if USART1 of AT90CAN128 is used as LIN port.

1.7.4 How to set up my own LIN Master

Several steps have to be followed to get a working LIN master :

- LIN Library option definition. Check the “MASTER_LIN.H” file.
- LIN Library configuration. Set the “CONFIG.H” file.
- Data buffer declaration and initialization.

Set all buffer needed for the LIN application.

```
U8 Buf_SET_SLAVE [8];
...
```

- LIN frame declaration and initialization.

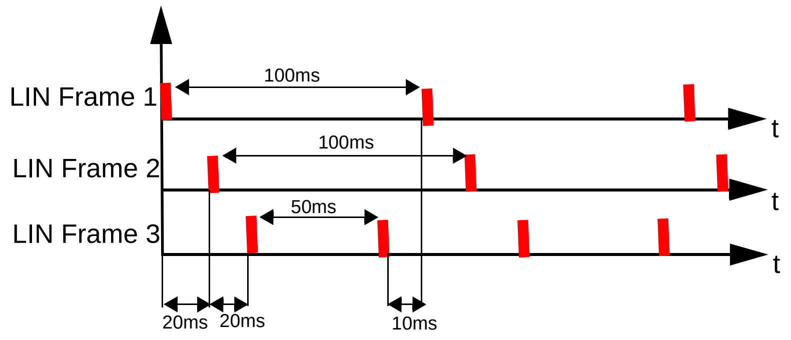
```
t_frame MESS_SET_SLAVE;

MESS_SET_SLAVE.frame_id    = SET_SLAVE_FRAME_ID ;
MESS_SET_SLAVE.frame_size  = SET_SLAVE_FRAME_DLC ;
MESS_SET_SLAVE.frame_type  = SET_SLAVE_FRAME_TYPE ;
MESS_SET_SLAVE.frame_delay  = 1500 ;  // delay in tick
MESS_SET_SLAVE.frame_data  = Buf_SET_SLAVE;
```

- Scheduler Timing definition.

All timings have to be well known. For example, 3 LIN frames have to be sent :

Figure 1-8. LIN Frame Timings



LIN frames have to be initialized as following :

- LIN Frame 1 delay = 20ms. (delay_tick = 20ms*19200baud = 384)
- LIN Frame 2 delay = 20ms.
- LIN Farme 3 delay = 50ms.
- LIN Frame 3 delay = 10ms.

- "my_schedule" object initialization.

"my schedule" object needs all LIN frames used and the number of LIN frame.

```
my_schedule.frame_message[0] = MESS_SET_SLAVE;
my_schedule.number_of_frame = number_of_frame;
```

To modify the maximum number of frame allowed in the software, change the NB_FRAME value in "MASTER_LIN.H".

- LIN initialization.

The function LIN_init() has to be executed in the main function after I/O ports configuration : CONFIG_IO_PORTS(). To finish enable interrupts : SREG |= 0x80;

- The tick function.

The tick function has to be called regularly in the main function or at TBIT rate through an available timer for example. Currently, the LIN_TIMER_init function initialize the timer 1 to be reloaded at each overflow. The "timer1ovfl_isr.c" file shows a possible use of the timer 1 to call the tick function.

- Data processing by user application.

All configuration and initialization are done. The LIN master is ready to be used.

1.7.5 How to setup my own LIN Slave

Several steps have to be followed to get a working LIN slave :

- LIN Library option definition. Check “SLAVE_LIN.H” file.
- LIN Library configuration. Set the “CONFIG.H” file.
- Data buffer declaration and initialization.

Set all buffer needed for the LIN application.

- LIN frame declaration and initialization.

```
t_frame MESS_SET_SLAVE;

MESS_SET_SLAVE.frame_id    = SET_SLAVE_FRAME_ID ;
MESS_SET_SLAVE.frame_size  = SET_SLAVE_FRAME_DLC ;
MESS_SET_SLAVE.frame_type  = SET_SLAVE_FRAME_TYPE ;
MESS_SET_SLAVE.frame_data  = Buf_SET_SLAVE;
```

- “my_schedule” object initialization.

“my schedule” object needs all LIN frames used and the number of LIN frame.

```
my_schedule.frame_message[0] = MESS_SET_SLAVE;
my_schedule.number_of_frame = number_of_frame;
```

To modify the maximum number of frame allowed in the software, change the NB_FRAME value in SLAVE_LIN.H”.

- LIN initialization.

The function LIN_init() has to be executed in the main function after I/O ports configuration : CONFIG_IO_PORTS(). To finish enable interrupts : SREG |= 0x80;

- The tick function.

The tick function has to be called regularly in the main function or at TBIT rate through an available timer for example. Currently, the LIN_TIMER_init function initialize the timer 1 to be reloaded at each overflow. The “timer1ovfl_isr.c” file shows a possible use of the timer 1 to call the tick function.

- Data processing by user application.

All configuration and initialization are done. The LIN slave is ready to be used.

1.7.6 LIN Library Performance

1.7.6.1 LIN Master

- Resource Usage

1 Serial peripheral (USART).

1 Timer reserved for time out detection.

1 Timer for periodic calls to tick function (optional).

- Code size and RAM usage

All results are given with all options enabled (Sleep timeout detection, timeout detection, and counting errors). Two types of results are given without optimization and with IAR® Compiler with maximum optimization for size.

Without optimization

Total code size : 4 kBytes.

Total data size : 648 Bytes (With Heap and Stack).

Table 1-5. Results without optimization

Library	CODE (Bytes)	DATA (Bytes)
Lin_drv_usart	1246	18
Lin_lib	974	0
Master_lin	862	46
Main	154	8

With maximum size optimization

Total code size : 3.1 kBytes.

Total data size : 648 Bytes (With Heap and Stack).

Table 1-6. Results with optimization

Library	CODE (Bytes)	DATA (Bytes)
Lin_drv_usart	810	18
Lin_lib	734	0
Master_lin	692	46
Main	154	8

- CPU Load

In the worst case (no delay between two frames), the CPU load is about 5% at 19200 baud. For a LIN baud rate of 2400 Baud the CPU load drops to 0,6%

1.7.6.2 LIN Slave

- Resource Usage

1 Serial peripheral (USART).

1 Timer reserved for time out detection.

1 Timer for periodic calls to tick function.

1 Input capture module if runtime RC oscillator calibration is enable (refer to the application note AVR054).

- Code size and RAM usage

All results are given with all options enabled (Sleep timeout detection, timeout detection, RC Oscillator calibration...). Two types of results are given without optimization and with IAR Compiler with maximum optimization for size.

Without optimization

Total code size : 4 kBytes.

Total data size : 648 Bytes (With Heap and Stack).

Table 1-7. Results without optimization

Library	CODE (Bytes)	DATA (Bytes)
Lin_drv_usart	1246	18
Lin_lib	974	0
Slave_lin	862	46
Main	154	8

With maximum size optimization

Total code size : 3.1 kBytes.

Total data size : 648 Bytes (With Heap and Stack).

Table 1-8. Results with optimization

Library	CODE (Bytes)	DATA (Bytes)
Lin_drv_usart	810	18
Lin_lib	734	0
Slave_lin	692	46
Main	154	8

- CPU Load

In the worst case (no delay between two frames), the CPU load is about 5% at 19200 baud. For a LIN baud rate of 2400 Baud the CPU load drops to 0,6%.

1.7.7 How to modify the code for an other AVR target

Some code modification are necessary to use an other AVR microcontroller. Only two files have to be modifie. They are :

- config.h
- lin_drv_usart.h

In “config.h” file, define the AVR controller used.

```
#define NAME_UC_USED
```

Next, include headers associated with the microcontroller and the board and define the UART used.

```
#ifdef NAME_UC_USED
#include <iomNAME_UC.h>
#include “..\board.h”
#define USE_UART0/1
#endif
```

In “lin_drv_usart.c” define interruption vector used according to the UART used (fo microcontroller with more than one USART).

```
#ifdef NAME_UC_USED
#pragma vector = USART0/1_RX_vect
#endif
```

Enable USART transmission and reception in the “Lin_hw_init()” function.

```
#ifdef NAME_UC_USED
#define UCSR0/1B = (1<<RXEN0/1 ) | (1<<TXEN0/1) | (1<<RXCIE0/1) ;
```

1.8 Conclusion

This application note provides a software solution to implement LIN protocole v1.3 on Atmel AVR microcontrollers. All source code are available on the Atmel web site.

Resources necessary fits with most of the AVR microcontrollers. The memory usage and the CPU load have ben kept low enough for developpers to implement other functions.

Atmel Corporation

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

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

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

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

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

Printed on recycled paper.