AVR308: Software LIN Slave

8-bit AVR® Microcontroller

Application Note

Features

- **Low-cost, No External Components**
- **Compatible with LIN Protocol Specification Version 1.0**
- **Size-efficient Code**

Introduction

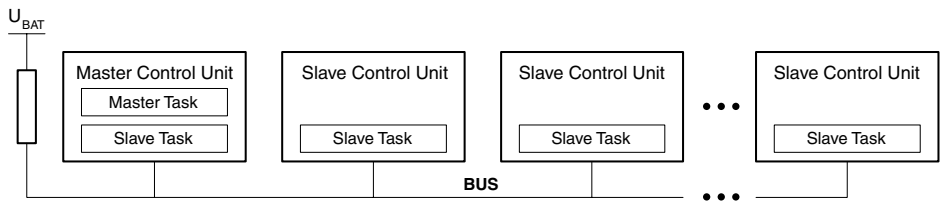
This application note shows how to implement a LIN (Local Interconnect Network) slave task in an 8-bit RISC AVR microcontroller without the need for any external components.

The LIN protocol is a serial communications protocol which efficiently supports control of mechatronic nodes in a distributed network. This makes the protocol ideal for use in automotive applications. A LIN network consist of a single Master and a set of Slave nodes. This application note show how to implement the protocol for the physical layer and the data link layer according to the two lowest levels of the ISO/OSI reference model. This provides the foundation for message transmissions between nodes in the network. The physical layer of the ISO/OSI model only provides a raw bit-stream between two nodes in the network. The data link layer makes the physical layer reliable by adding error detection and control. It also adds the means to activate, maintain, and deactivate the link. The higher levels of the ISO/OSI reference model is beyond the scope of the LIN protocol and thus not explained here. For now, this is up to the user to specify and implement.

The LIN protocol differs from the CAN protocol in the sense that it is below the performance and scope of the CAN. Its strongpoint is that it is provides a simple, low-cost solution for applications not requiring the power of the CAN protocol. The main properties of the LIN bus include:

- Single Master, multiple Slaves.
- Low-cost silicon implementation.
- Self synchronization without quartz or ceramic resonators in the Slave nodes.
- Guarantee of latency time for signal transmission.
- Single-wire implementation.
- Speeds up to 20 Kbits/s.

Figure 1. LIN Network Topology



Rev. 1637B–AVR–05/02

LIN Protocol Concepts

Single Master Multiple Slaves

The LIN protocol does not use bus arbitration. A single Master is responsible for initializing all message transfers. All slaves can respond to the Master or any other node in the network but only after being addressed and given permission by the Master.

Variable Length of Data Frames

Included in the identifier field are two bits indicating the length of the message field (see Table 1). This adds flexibility and reduces overhead bytes when only a limited amount of data is required.

Multi-cast Reception

When a message frame is transmitted from the Master or a Slave task, all connected nodes in the network can read the message. Depending on the identifier byte, the receiving nodes decides if any action is to be initiated or not. For example, a single “CLOSE ALL” command from the Master could be accepted from all nodes and could in the case of a car security system close all windows and doors.

Time Synchronization without Need for Quartz or Ceramic Resonators in the Slave Nodes

After the Synch Break, a Synch Field is transmitted from the Master, this field makes it possible for all the Slaves to synchronize to the master clock. Such synch fields are placed at the beginning of every message frame. The accuracy of the receiving Slaves need only be good enough to keep synchronization through the entire message frame. This feature allows the slave to run on an internal RC Oscillator thus reducing the overall system cost.

Data Checksum Security and Error Detection

The data in the message frame uses an inverted modulo256-checksum with the carry of the MSB added to the LSB for error detection. In addition, the identifier byte uses a XOR algorithm for error detection.

Detection of Defect Nodes in the Network

The Master task is responsible for initiating the transmission of message frames and thus has the responsibility of requesting information and checking that all nodes are alive and working correctly.

Minimum Cost Solution

Due to the simplicity of the protocol, a slave task complying to the LIN standard can be built using a minimum of external components and does not put heavy constraints on the accuracy of the Oscillator in the Slave nodes.

Signal Transmission

The LIN bus consists of a single channel that carries both data and synchronization information. The physical medium is a single wire connected to V_{CC} via a pull-up resistor (see Figure 1). The idle state on the bus is high or “recessive” and the active state is low or “dominant”. In automotive applications V_{CC} will typically be the positive battery node.

The LIN protocol does not define an acknowledgment procedure for the Slave tasks. The Master task uses its own slave task to verify that the sent message frame is identical with the one received by this slave task. If any discrepancy is detected the message frame can be retransmitted.

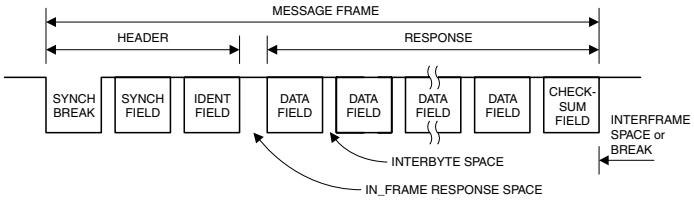
The data rate for transmitted data is limited to 20 Kbits/s due to EMI (Electro Magnetic Interference) requirements for a single wire transmission medium.

Message Frames

All information transmitted on the LIN bus is formatted as Message Frames. As shown in Figure 2, a Message Frame consists of the following fields:

- Synch Break
- Synch Field
- Identifier Field
- Data Field
- Checksum Field

Figure 2. LIN Message Frame

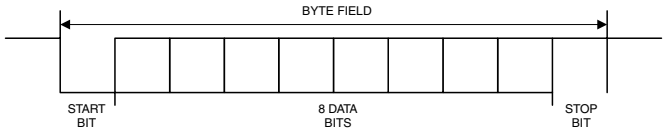


The message frame consists of two parts: The “header” sent by the Master and the “response” which can be used by both Master and Slave tasks.

Byte Fields

The Byte Field format, shown in Figure 3, is identical to the commonly used UART serial data format with 8N1 coding. This means that each Byte Field contains eight data bits, no parity bits, and one stop-bit. Every Byte Field has the length of 10 bit-times (Tbit). As shown in the figure, the start bit marks the beginning of the Byte Field and is “dominant” while the stop-bit is “recessive”. The eight data bits can be either “dominant” or “recessive”.

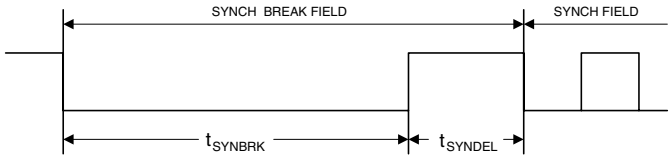
Figure 3. LIN Byte Field



Synch Break

The Synchronization Break marks the beginning of a Message Frame. This field is always sent by the Master task, and provide a means for the Slave task to prepare for the synchronization field. The Synch Break consists of two different parts: a dominant (low) level that should be minimum 13 bit-times (Tbit) which is followed by a recessive (high) period that should be in the range one to four Tbit. The second field is necessary to detect the dominant (low) level of the start bit of the following Synch Field.

Figure 4. Synch Break Field

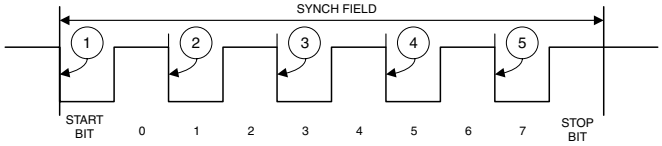


The length of the first field is chosen to distinguish between the Synch Break and the maximum possible allowed sequence of dominant bits within a data frame. For example, a data field with all “0”s should not be mistaken for a Synch Break field.

Synch Field

The Synch Field contains the signalling required for the slave to synchronize with the master clock. The Synch Field is a Byte Field containing the data “0x55” giving the waveform shown in Figure 5.

Figure 5. Synch Field



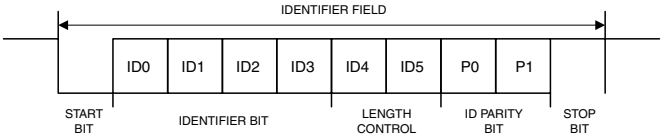
As illustrated, the Synch Field is characterized by five falling edges (recessive to dominant edges)

These edges are used during synchronization to tune the slave node transmission and reception speed to match the master node as explained later in this application note.

Identifier Field

The Identifier Field contains information about contents and length of a message. As shown in Figure 6, this field is divided into three sections: identifier bits (four bits), length control bits (two bits) and parity bits (two bits). This divides the set of 64 identifiers into four subsets of 16 identifiers each.

Figure 6. Identifier Field



The LIN protocol defines this as follows:

Table 1. Number of Data Fields in a Data Frame

ID5	ID4	N_{DATA} (Number of Data Fields)
0	0	2
0	1	2
1	0	4
1	1	8

As shown in Table 1 there are two groups with two data fields, one group with four data fields, and one group with eight data fields. Note that the Identifier Field does not indicate the destination of the message but describes the contents of the message frame. It is up to all the Receiving Slave tasks to decide if they should act upon the received message or not. The last two bits of the Identifier Field contains parity information. LIN uses a mixed-parity algorithm that ensures that the Identifier Field never will consist of an all “recessive” or “dominant” pattern. Note that the parity check only detects errors, it does not correct them.

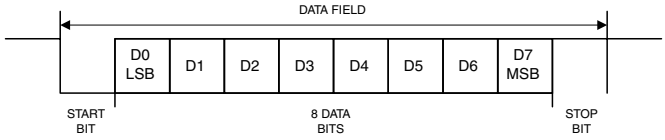
The parity check bits are calculated by the following mixed-parity algorithm:

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$
$$\overline{P1} = ID1 \oplus ID3 \oplus ID4 \oplus ID5$$

Data Field

The data frame contains from two to eight Data Fields containing eight bits of data each. Transmission is done with LSB first. The data fields are 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.

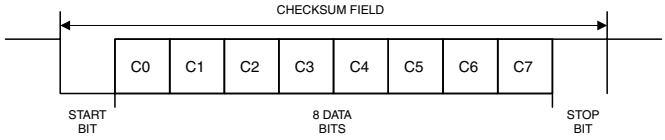
Figure 7. Data Field



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 (data frame not including the identifier). This 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”.

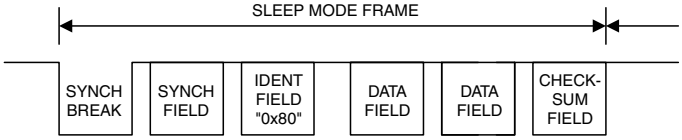
Figure 8. Checksum Field



Sleep Mode Frame

The frame structure for the Sleep Mode Frame is identical to an ordinary Message Frame and is distinguished by the identifier byte “0x80”. The contents of the data fields are not specified and could be used to distribute system parameters for the sleep mode.

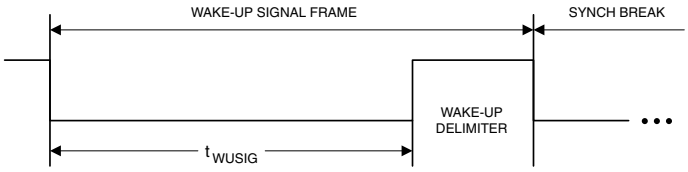
Figure 9. Sleep Mode Frame



Wake-up Signal

The sleep mode can be terminated by any connected slave task by sending a wake-up signal. The wake-up signal is only allowed when the bus is in sleep mode and a node-internal request for wake-up is pending. The wake-up signal is the character “0x80”.

Figure 10. Wake-up Signal Frame



Depending on whether the Slave task issuing this wake-up is in synch with the Master task or not, the transmission speed of the transmitted character may be faster or slower than the Receiving Master. For this reason “0x80” could be received as either a “0xC0”, “0x80” or “0x00”. All will be regarded as a valid wake-up condition byte by the attached

slave nodes which will wake up and wait for a Sync Break Field from the Master. If no such field is detected within a timeframe of 128 Tbit, a new wake-up signal is issued. This is repeated three times. It there is still no response from the Master, the Slave with the pending wake-up request will wait at least 15,000 Tbit before trying again. The wake-up delimiter is in the current version of the LIN protocol specified to be in the range four to 64 Tbit.

Error Handling

The LIN protocol includes error detection on both identifier field and data fields. No error correction is specified nor is there any way for the slaves to automatically transmitting information on a bad transmission. The Master is responsible for polling the slaves for this information.

The LIN protocol does not define a procedure for acknowledgment for a correctly received message. The Master control unit compares the message sent by its Master task with the one received by its own Slave task. If these match, a correct transmission is assumed. In case of an inconsistency, the Master task can retransmit the message.

If an inconsistency is detected by a slave, this information will be saved and provided on request from the Master. This diagnostics information can be transmitted as part of a regular data frame.

Connections

In theory, the total number of nodes are unlimited. Realistically, the number will be limited by noise, delay, and electrical loads on the bus. The total number of nodes in a subnetwork should not exceed 16. In principle, the protocol supports 63 nodes. The LIN protocol also specifies that the accumulated “galvanic” wire should not exceed 40 meters. The bus termination is specified as 1 for the Master node and 20 - 47kΩ for the Slave nodes.

Message Transfer

Message filtering is based upon the whole identifier. All slaves can read and act upon a message but only one node is allowed to respond to a transmitted identifier.

A message is valid if there are no errors detected through the entire message frame. If a message is corrupted, it is regarded as not transmitted by both the Slave task and the Master task.

Fault Confinement

The Slave task should be able to identify the following error situations:

- A “bit error” in a data or checksum field while reading back its own transmission.
- An “Identifier-Parity-Error” or a “Checksum-Error” while reading from the bus.
- A “Slave-Not-Responding-Error” is detected while reading from the bus.
- An “Inconsistent-Synch-Byte-Error” is detected when the edges of the Synch Field are not detected within the given tolerance.

Oscillator Tolerance

On-chip RC Oscillators typically have a large tolerance range (e.g., -50% to +100%) due to process parameters during production. The LIN protocol specifies that the frequency state of the slave when it has lost synchronization should be within ±15% compared to the Master. Within a short time frame, the tolerance should be better than ±2%. These requirements should be valid over the complete voltage and temperature range of the device.

Software Implementation

The implementation of the different sections of the program is described in the following sections. The program contains seven different routines.

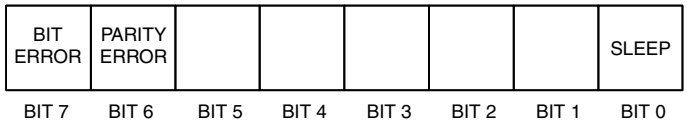
Table 2. Implemented Function

Function	Description	Called by
main	Main Program	Reset
ext_int0	Interrupt Handler for External Interrupt 0	Falling Edge of PIND2
check_parity	Counts the Number of Dominant Bits in the Passed Variable	Ext_int0
tim0_ovf	Called when Timer/Counter0 Overflows when Counting Bittime	Timer/Counter0 Hardware
putchar	Transmits Data via the LIN Bus	Ext_int0
delay	General Delay Routine	Ext_int0
wakeup	Sends a wake-up Signal to the Master	User

main

In this example the main program does nothing but an eternal loop. This is where the user's application code should be placed. Note that the AT90S1200 does not have any SRAM and uses a three-level hardware stack instead of a conventional SRAM Stack. If more than three levels are used, the first value of the Stack is overwritten and lost. This causes unpredictable program behavior. When an external interrupt occurs (due to bus activity) the interrupt routine is called. When receiving data, the interrupt routine calls the check_parity routine and when transmitting data the putchar routine is called. Thus two stack levels are used by the interrupt routine. It is therefore important that the user program does not use more than one stack level, giving a total of three stack levels.

Figure 11. Status Byte



When a bit error or parity check error is detected, the corresponding bits in the Status Register (r20) is set. See Figure 11 for the bit positions. These flags must be deleted manually after a read. The main program should continuously poll the Sleep Flag. When this flag goes high the AT90S1200 can be put in idle mode by the following instructions:

```
ldi   r23,(1<<SE)+(1<<ISC01)
out   MCUCR,r23              ;select idle mode and enable sleep
sleep                        ;wakeup on falling edge INT0
```

When any bus activity is detected, the device will wake up and execute the ext_int0 routine. 181 words are used by the LIN interface so the user program size can be up to 331 words (662 bytes). This gives a total of 512 words (1 KB).

External Interrupt 0: ext_int0

This routine contains all the functionality of the LIN slave interface. External interrupt 0 is triggered by a high-to-low edge on pin 2, Port D. The interrupt routine is triggered by the starting edge of the Synch Break, for every edge in the Sync Byte, and by the start byte of the identifier. To keep track of what to expect on the data bus, three modes are used as defined by the variable “mode”:

- mode1: The program expects to receive a Synch Break.
- mode2: The program expects to receive the Synch Byte.
- mode4: The program expects to receive the identifier.

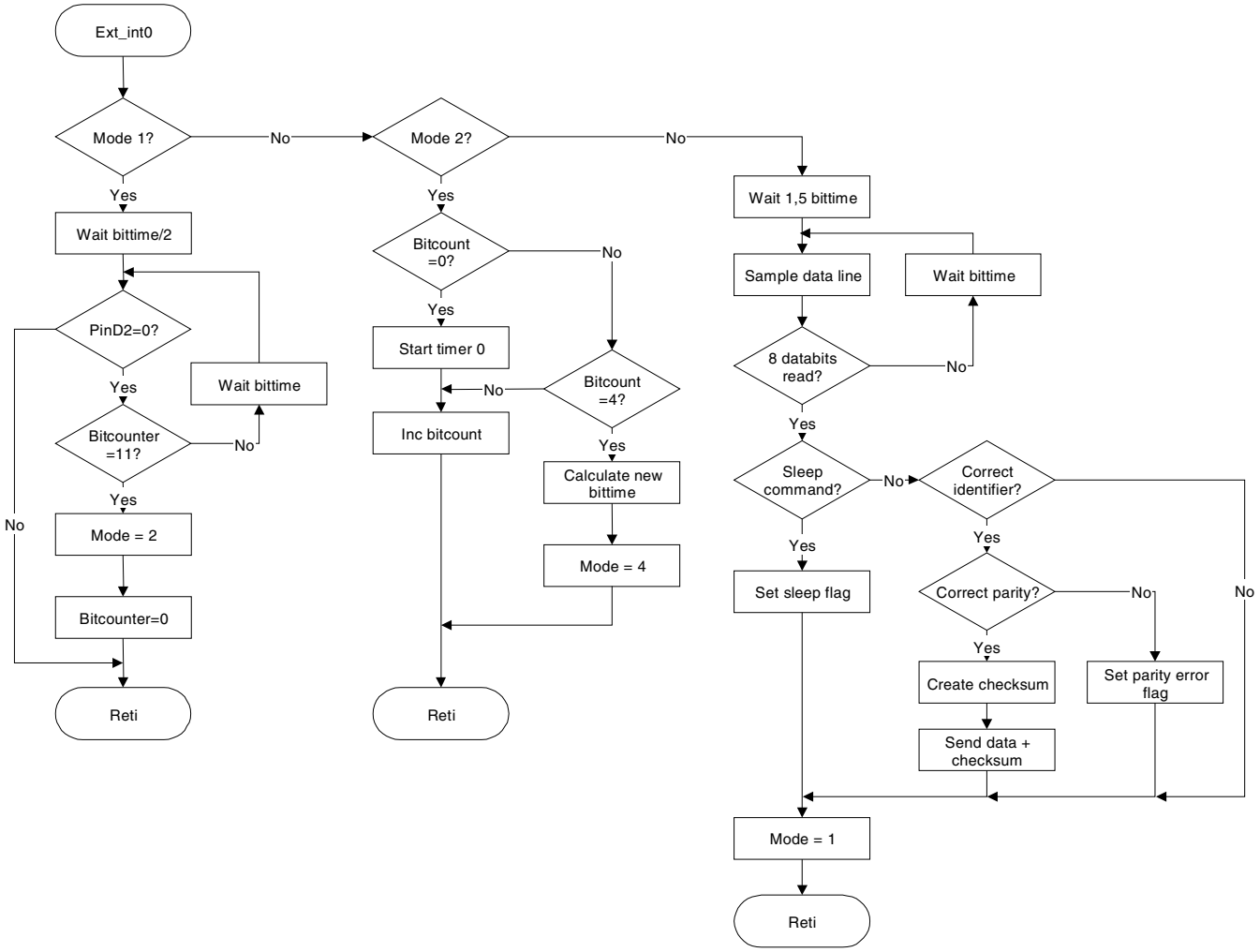
Figure 12 shows the flowchart diagram of the ext_int0 routine.

In mode1 the program looks for 11 succeeding recessive bits. The bit length is assumed to be the same as the last transmission. If 11 succeeding recessive bits are found, the mode variable is set to two.

When the first edge is detected in mode2 the Timer is started. Then the program count the number of high to low edges, and when five edges are detected (eight bits) the Timer is stopped (see Figure 5). The bit width is found by dividing the timer value by eight.

Mode4 reads the transmitted identifier. First the identifier is checked to see if there was a sleep command. If a sleep command was not received, the identifier is compared to the internal masks to see if it should be responded to. If the identifier corresponds with a mask, parity is checked. Then the checksum is created and data and checksum is sent.

Figure 12. Dataflow of ext_int0 Interrupt Handler Routine



check_parity

This routine counts the number of recessive bits in the counter variable. If the number is even, the LSB of the bitcount variable is 0. If the number is odd, the LSB of the bitcount variable is 1. This is used to check the counter variable for odd or even parity.

Timer0 Overflow: tim0_ovf

Timer/Counter0 used in this application is an 8-bit counter. To be able to count eight bit-time, this routine is called when Timer0 overflows, incrementing the “counter” variable. By combining the “counter” value, and the Timer/Counter1 value, the result gives a 16-bit number indicating the time for eight bits. Dividing this by eight gives the bittime. The AVR microcontroller runs on a chosen frequency of 1 MHz and the counter increments every clock cycle. Thus, the bittime is given directly in microseconds.

putchar

The putchar routine is used for sending data via the LIN bus. When sending a “0”, PIND2 is set as output with a “0” in the Port Register. When sending a “1”, PIND2 is set as input and the LIN bus’ pull-up resistor pulls the level to logical “1”. This acts like an open collector output. The data to be sent should be placed in adjacent registers in ascending order (e.g., data1 to data*n* is placed in register r*x* to r*x+n*). The Slave sends either two, four, or eight data bytes decided by bit four and five in the Identifier Field. The constant data_addr in the assembly file holds the register number of the first data to send.

delay

The delay routine gives a delay of [temp2] cycles ± 1 cycle. This routine is used to create the required delay between the sampling of bits.

wake-up

To generate a wake-up call on the bus this routine may be called from the main program. The routine is not used by the ext_int0 routine.

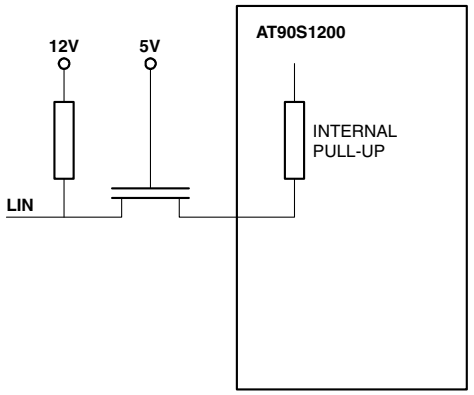
Hardware Considerations

The LIN standard specifies an operating voltage range for the “dominant” voltage between 9V and 18V (40V while not operating). This application is constructed for a LIN voltage of 5V. Since the AVR microcontroller used in this application requires a stable 5V supply, the most cost-effective and easiest solution is to let the LIN bus dominant voltage be 5V. If this is not acceptable there are some hardware solutions to interface the 5V LIN to a 9 - 18V LIN.

Method 1

By using a single FET the 5V and the 9 - 18V parts may be connected (see Figure 13). Here 12V and 5V parts are connected using a single FET. When either the 5V side or the 12V side of the FET are pulled low, the corresponding side will be pulled sufficient low to detect a recessive bit.

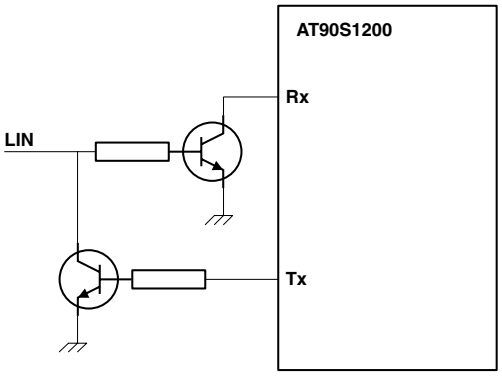
Figure 13. 5V to 12V interfacing using single FET



Method 2

Using two bipolar transistors (in the same housing) may be a more cost-effective solution (see Figure 14). These transistors are obtainable complete with base resistors and are less expensive than FET transistors.

Figure 14. 5V to 12V Interfacing using Bipolar Transistors



If this approach is selected, the program has to be slightly rewritten. The following should be altered:

- The putchar routine must output data to the Tx pin instead of to PIND2 (Rx pin).
- Data must be written to the Tx pin as normal, not as done when using “open collector” outputs.
- The Tx pin must be initialized as output (DDRD, pinTx = 1)
- The data written and received must be inverted (high level out gives low level to the bus and high level on the bus gives low level in).

Ref: LIN Protocol Specification Revision 1.0.

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

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 Tsimhatsui
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
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131
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
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
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

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

1637B–AVR–05/02 0M