APPLICATION NOTE

ABSTRACT
The single-wire LIN bus (Local Interconnect Network) is being defined as a new standard to complement high-end automotive buses like CAN with lower-cost solutions where less performance suffices. This application note describes how a LIN bus can be implemented with existing 80C51-based microcontrollers from Philips Semiconductors.

AN10115
Philips Microcontrollers in LIN Applications

Torsten Eggers, Lutz Pape, Wolfgang Schwartz

2002 February 15

Philips Semiconductors

PHILIPS

INTRODUCTION

Several network concepts are in place for the communication between the ever-increasing number of electronic modules in the car. Buses like J1850 and CAN are well established and the number of implemented nodes per car is constantly growing. Increasing complexity and the need to keep costs at a minimum brought a requirement for simpler sub-buses, which in turn may be connected via a more powerful backbone.

Driven by a consortium of European car manufacturers the open LIN ("Local Interconnect Network") bus standard is being defined for this purpose. The standard covers the transmission protocol and medium and defines tool and application programming interfaces. It guarantees the interoperability of network nodes (hardware/software) and assures a predictable EMC behaviour.

Some key features of the LIN bus concept are:

- Physical Layer
  - low cost single-wire implementation
  - enhanced ISO9141, powered from car battery
  - speed up to 20kbits/s (limited for EMC reasons)
- Data Link Layer
  - single master / multiple slaves
  - no arbitration necessary
  - interface based on standard UART/SCI (or software "bit-banging")
  - self synchronization of slaves without crystal or resonator
  - guaranteed latency times for signal transmission
- Network Layer
  - Time-triggered scheduling

Transceivers care for the signal conditioning of the physical layer and handle the 12V-signal. Their slew rate control and wave shaping mainly impact the system's EMC behaviour. These components are readily available from Philips as TJA1020T.

This application note describes how a LIN bus can be implemented with existing microcontrollers from Philips Semiconductors.

Often the LIN master controller acts as a gateway between e.g. a CAN bus and the LIN bus. The P87C591 has a PeliCAN-interface for the CAN side, while its enhanced UART can be used to support the LIN bus. The handling of the CAN interface is covered in a separate Application Note (AN00043).

MESSAGE FRAME

LIN uses a single message frame format to synchronize and address the nodes and to exchange data between them. The master defines the transmission speed and sends the header of the message frame (see figure 1). This header starts with a synch brake followed by a synch field to synchronize the LIN slaves to the master's bit rate. The ID-field is the last header block; it holds information about the sender, receiver(s) and data field length.

After synchronization all slave nodes - and the master node can be a slave, too - interpret the ID-field and react accordingly: either receive or send data or do nothing. This response consists of 2, 4 or 8 data bytes and a final checksum field.

A special identifier is used to set all nodes in a sleep mode to conserve battery power. This stops all bus activity until a wake-up signal is applied on the bus by any node.

Header and response are separated by an in-frame

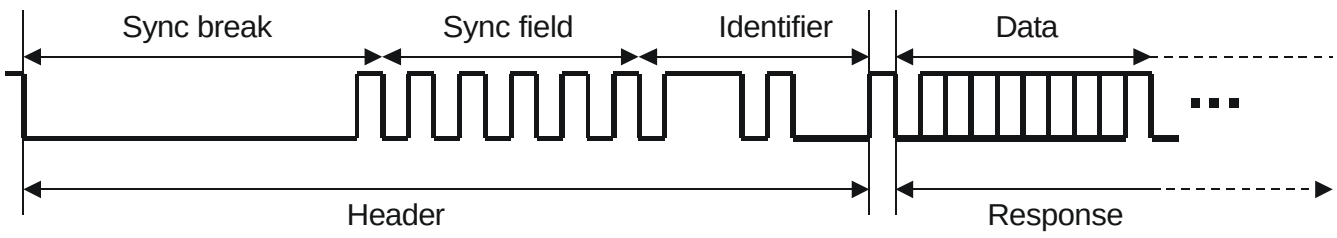

Figure 1 LIN Message Frame

response space, while message frames are separated by inter-frame spaces. The minimum length of both spaces is zero.

LIN MASTER

The master task, running on the LIN bus master node, controls all traffic on the bus:

- define transmission speed (2k ... 20kbits/s), derived from a precise reference clock
- send synch brake
- send synch field
- send ID field
- monitor and validate data bytes by checking the checksum
- request slaves to enter sleep-mode and wake them up again, when needed
- react on wake-up break from slave

In the examples below Philips' P87C591 handles these master tasks.

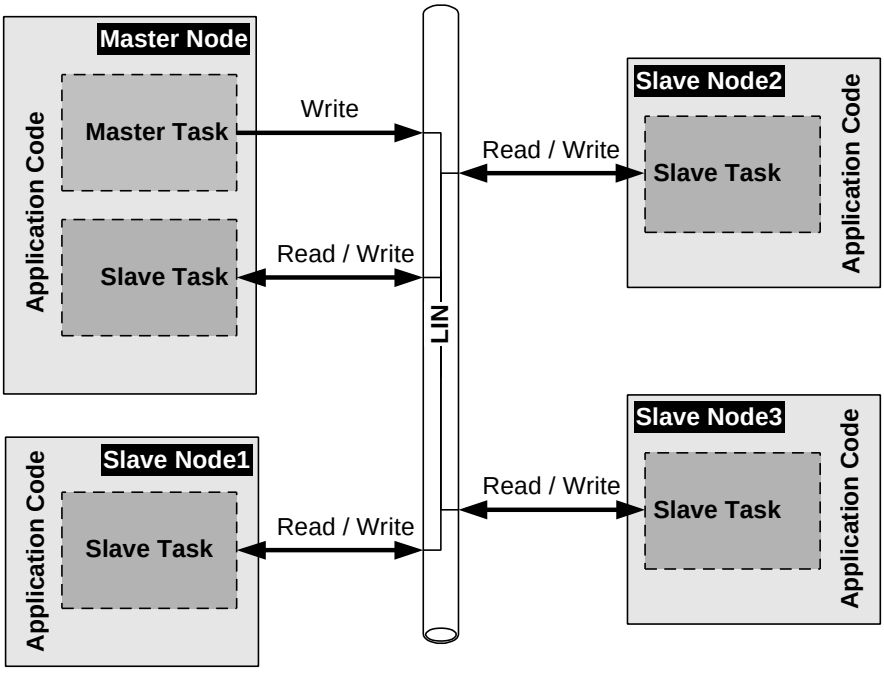

Figure 2 LIN Bus Nodes

LIN SLAVES

Slave tasks run on up to 16 slave nodes connected to a LIN bus. The master node can be a slave, too. The slave task cares for the following:

- Wait for synch brake
- Synchronize on synch field
- interpret identifier and act accordingly:
  - do nothing
  - receive data
  - send data
- check/send checksum

The Philips 51LPC76x family of microcontrollers is used in the examples.

EXAMPLES

A sample LIN system was implemented with a P87C591 on the master node and P87LPC76x on several slave nodes. Following is a description of the use of the on-chip peripherals to perform the LIN master and slave tasks.

To fulfill all timing restrictions of the current LIN specification, all nodes have to be equipped with a precise oscillator reference, i.e. a crystal on the master node (P87C591) and a crystal or ceramic resonator on the slave nodes (P87LPC76x). The '591 has a dedicated Baud-rate generator on-chip, while for the '76x timer T1 is used to generate the Baud-rates.

Both families of microcontrollers have an enhanced UART to support the serial LIN transmission.

Timer T0 is used for various time-out detections.

Because the LIN synch-break is longer than the breaks common UARTs can handle, the slaves additionally use other on-chip blocks like the analog comparator or external interrupts for break-detection.

SOFTWARE

The software routines require less than 1Kbytes of program memory (master/slave tasks in '591 and slave tasks in '76x) and less than 30 bytes of RAM.

During the initialization phase, the LIN transceiver is configured and the protocol handler variables are preset.

The interface between the application program and the LIN driver is implemented with so-called callback functions. These functions have to be provided by the application program and are called from the driver.

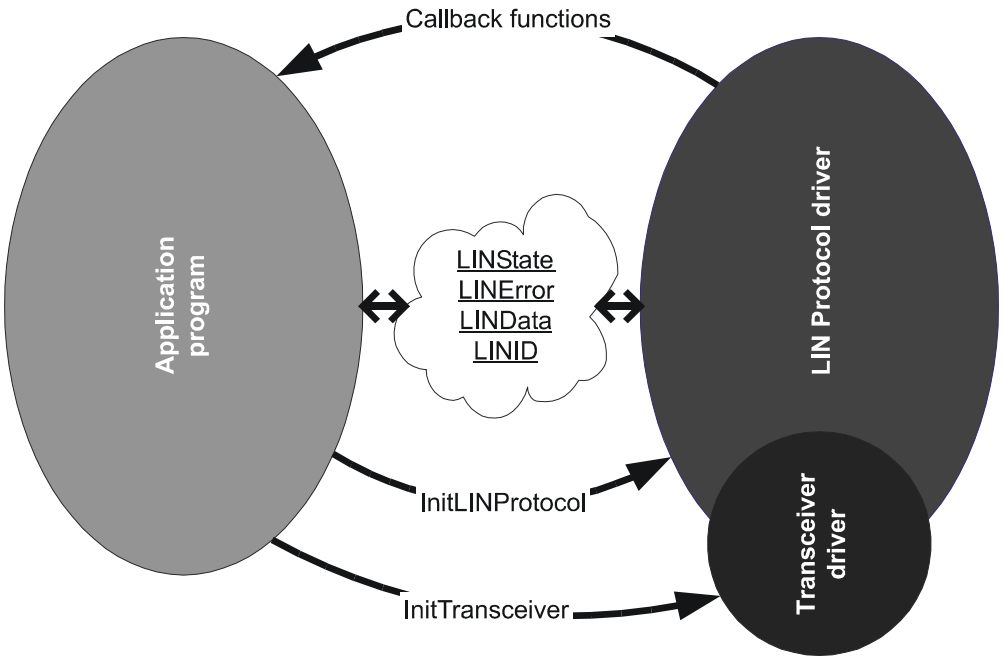

Figure 2 Interface Application Program - LIN Drivers

Three basic tasks need to be performed by the nodes: the main task reflecting the specific function of the node and two tasks to control the power consumption by putting the node to sleep and wake it up again. In the example these functions are called

- `LINServiceRoutine`
- `LINSetToSleep`
- `LINWakeup`

The functions use the parameters `LINState`, `LINError`, `LINData` and `LINID` to pass information between the routines.

LINServiceRoutine

This is the main routine, which realizes the behaviour of the node. This function is called whenever a protocol event has occurred, e.g. a break was detected, a LIN-ID was received or a bus error happened. Information about the event is returned in two variables, "LinState" and "LinError". Based on the contents of these variables the application program performs the function of the node.

LINSetToSleep

When the car engine is not running and power is supplied from the battery, the power consumption must be minimized. All nodes connected to the LIN bus can be put in a power-saving sleep mode.

Philips microcontrollers P87C591 and P87LPC76x support this function with two different modes. In Power-Down mode, the oscillator is stopped and no on-chip peripherals are running. This is the most effective power saving mode. It can only be ended by an external interrupt or a chip reset.

In Idle mode, the CPU is stopped and only some peripherals like timers keep working. Normal operation is resumed after any (enabled) interrupt request. Even if the microcontroller in a LIN-node has to be kept active, power on the LIN bus can be conserved by putting the LIN transceiver into Sleep-Mode.

LINWakeup

There are two ways to wake up a LIN node from Sleep mode. Either a local event wakes it up or a wake up request was sent by any other node via the

LIN bus. A local event could be a pushed button or an exceeded temperature limit. If a local event requires an action of the master, any node can issue a wake up request on the LIN bus.

The follow code example[1] illustrates how the callback function `LINServiceRoutine` can be implemented.

```
#include <RelPc764.h>
#INCLUDE "TJA1020.H"
      //assuming the header file
      //is located in the same di-
      //rectory as this file
#include "LINCmpSl.h"
      //assuming the header file
      //is located in the same di-
      //rectory as this file
#include "Tempsense.h"
      //assuming there is tempera-
      //ture sensor in this appli-
      //cation. The corresponding
      //code has to be written by
      //the user.

//LINServiceRoutine

void LINServiceRoutine()
{
 if (LinError==0)
      //check on LinError
   {          //no LINError

//###### LINFrameOk ################
      //the node receives data
   if (LINFrameOk)
      //A valid LinFrame availabe?
        {      //Now test the LINID
//### ID0 ###
         if (LinID==0)
            {  //react on LinID=0
             if (LinDataPtr[0]>0)
                LED1=0;   ;LED1 on
             else
                LED1=1;   ;LED1 off
            } //LINID = 0
```

```
//### ID3 ###
         IF (LINID==3)
            {  //REACT ON LINID=3
             IF (LINDATAPTR[0]>0)
                LED2=0;   ;LED2 ON
             ELSE
                LED2=1;   ;LED2 OFF
            } //LINID = 3
         } // LINFRAMEOK
//###### LINID RECEIVED ############
       //the node transmits data
      else if(LinIDReceived)
       //test if a valid ID has been
       //received
             {
              if (LinID==10)
                 { // react on ID=10
      LinDataPtr[0]=TemperaturBuffer;
       //write data to LinData
      TI=1
       // initiates sending
                 } //LINID=10
             } //LinIDReceived
   } //LinError = 0
 return;
}
```

REFERENCES

For further details, please refer to the following publications:

- LIN (general, specification)
  - http://www.lin-subbus.org/
  - LIN Specification Revision 1.2
- AN00043; P8xC591 Microcontroller in CAN Applications
  - www.semiconductors.philips.com/acrobat/applicationnotes/an00043.pdf
- Philips Data Sheets
  - www.semiconductors.philips.com
- LIN transceiver
  - TJA1020 LIN Transceiver
  - www.semiconductors.philips.com/pip/tja1020t/
- Microcontrollers
  - P87C591 CAN Microcontroller
  - P87LPC76x Microcontroller Family
  - www.semiconductors.philips.com/products/standard/microcontrollers/products/selguides/

[1] Raisonance C-compiler syntax.

Philips Microcontrollers in LIN Applications

AN10115

Revision history: **2002-02-15** **Revision 1**

Definitions

Short-form specification – The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information, see the relevant datasheet or data handbook.

Limiting values definition – Limiting values given are in accordance with the Absolute Maximum Rating System (IEC134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information – Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support – These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes – Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Contact information

For additional information please visit
http://www.semiconductors.philips.com. Fax: **+31 40 27 24825**

For sales offices addresses send e-mail to:
sales.addresses@www.semiconductors.philips.com

Date of release: 09-02
Document order number: 9397 750 10414

Let's make things better.

Philips Semiconductors

PHILIPS