To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

R8C/2X Series

LIN System Configuration and Using the LIN API

Introduction

This document describes how to set up and use the LIN demo source code for your own LIN project and how to use the LIN API.

Contents

1. Setting up the LIN source code for your project

The LIN demonstration project for the R8C family implements all that is written in this document as an example.

The code size for a "bare bones" node runs at about 5 kB, and for a master 5.5 kB.

LIN can be mixed with CAN on with a Renesas MCU that offers CAN without fear of conflicts.

1.1 Quick summary - How to set up project

Frames and signals are defined in the files lin_dev.h & .c. Your application is written in your application source code files(s).

lin_dev.h:

Enumerate all frames and signals in lin_dev.h

lin_dev.c:

- Add frame id entries to LIN_id_table[] for which the node is to subscribe or publish data to
- Add signals to LIN_signal_table[] and define signal's size
- Using the FRAME_SIGNAL_TBL, define
    - each frames signals
    - signal location in frame

Main application file

In your main application file, use the API calls in the API quick reference section 'Data read and write functions' to send and receive data between nodes.

2. Configure your project

2.1 General source files

The C-source files

- **lin_api.c** - do not edit
- **lin_api.h** - do not edit
- **lin_low_level.c** - do not edit
- **lin_low_level.h** - do not edit
- **lin_hdw.c**
- **lin_hdw.h**

shall be copied into each project directory. Do not edit the first four files.

2.2 The dev files

- **lin_dev.h**
- **lin_dev.c**

will contain your project definitions. See next chapter.

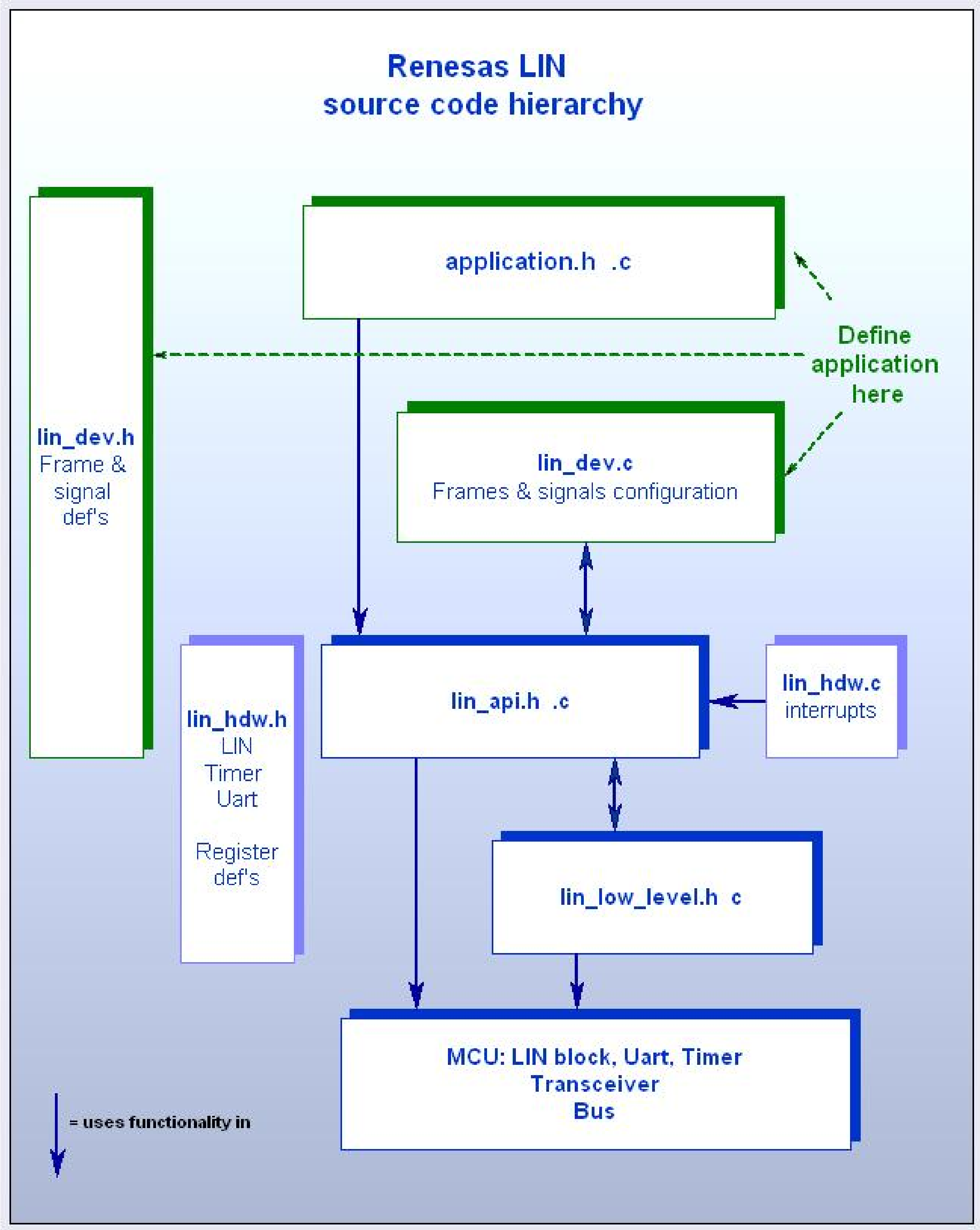

Figure. *The files that constitute the LIN source code. There is a master and two slave demo using this exact design.*

3. lin_dev.h & .c

The files **lin_dev.h** & **.c** need to be set up for your LIN application by editing them before copying them into the project directories. They should be completed first for the LIN system master node, and then copied into the project directories for each slave and edited down to configure each particular slave node. Alternatively, you can take the sample project and modify all **lin_dev.h** & **.c** files in place, but you will repeat practically all changes - twice for a constellation of one master and one slave.

The master *must* know every message frame and signal in the system, while each slave only needs to know those message frames and signals which it must process. Therefore, data structures and tables for the slave will be a subset of the master's. Slave nodes must know nothing about the schedule table `LIN_SCHEDULE_data[]`, so this may be deleted from the slave lin_dev files.

When signal and frame names are deleted from the enumerations in **lin_dev.h** for a slave node, it is totally irrelevant if the same signal or frame name is enumerated as a different value on different slave nodes or the master node. These enumerations are specific for each node, and serve as index values into the local `LIN_signal_table[]` and `LIN_id_table[]`. It is important that the enumerations in **lin_dev.h** for each individual node have a one-to-one correspondence to the signals and message frames configured in the corresponding tables for each node in **lin_dev.c**.

The files contain other defines. If the defines are commented out, the associated functionality will not be included in the compiled object file. There is a comment for each define explaining the functionality. There are also a few defines in **lin_hdw.h** that tailor the software to the functionality of the MCU and/or the application hardware.

3.1 Determining your signals

The first step in designing a LIN system is to identify all of the information that must be transferred from one device (master or slave) to another device on the LIN cluster. Then this information must be assigned to 'signals' (bit, byte, word, etc.). Each of these signals must be named. The signal names are defined by enumeration in the file **lin_dev.h**. For example;

```
typedef enum signal_names

{

   LIN_SIG_NODE_A_AD,

   LIN_SIG_NODE_A_SWITCH,

   LIN_SIG_NODE_A_COUNTER,

   LIN_SIG_CMD_SWITCH

} signal_name_type;
```

In addition a signal data buffer must be defined for each signal in **lin_dev.c**. The structure `LIN_signal_table[]` must be defined in **lin_dev.c**. This signal table lists all of the signals 'known' to the node (master knows all) in the same order they were enumerated in **lin_dev.h**.

3.2 Determining frames

All signals must be assigned to frames. Multiple signals may be packed into a single message frame, but all signals packed into one message frame must be published by a single node.

Each of the message frames must be named. The frame names are defined by enumeration in **lin_dev.h**, and a frame signal table (`FRAME_SIGNAL_TBL`) must be defined in **lin_dev.c** for each frame.

3.3 Setting up the ID table for each node

The structure `LIN_id_table[]` contains all of the information necessary to process the transmission or reception of a message frame. There is one entry in the id-table for each message frame that is known to the node (the master knows all) in the same order in which they were enumerated. This table may be in ROM for slave nodes that do not use dynamic node configuration.

Node Address, NAD, in master and node id tables need to match.

The 3rd item the table, `message_id`, only has a meaning when used together with dynamic node configuration. In systems that use dynamic node configuration, LIN_id_table[] for slave nodes is placed in RAM by removing the C keyword 'const' from the structure. The table can then be configured by the master at start up (or later.) Dynamic node configuration is not supported in the demo version, but its meaning in the id-table is shown in the figure below.

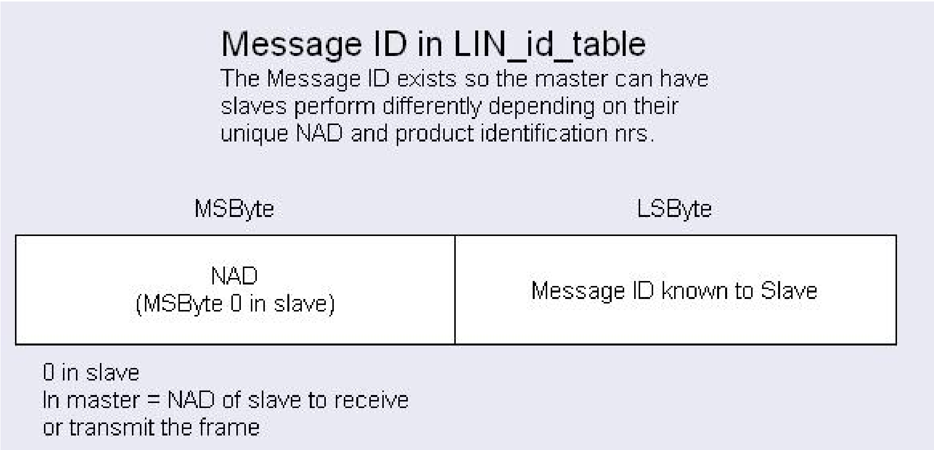

Figure: Message ID's meaning in the id-table.

3.4 The master node's schedule table

The message frames must be arranged in a schedule table. Schedule table names are enumerated in the file **lin_dev.h** and the structure `LIN_schedule_table[]` is defined and initialized in **lin_dev.c** and provides the data needed by the master node API software to make use of the various schedule tables.

Multiple schedule tables may be defined when the LIN cluster has more than one mode in which it must operate*. A LIN cluster should normally have only one operational schedule table, but may have one or more special schedule tables to perform unusual operations, such as dynamic node configuration or to support service technician diagnostics.

If the 'operational' schedule table contains Event Triggered message frames, there must be a collision resolution schedule table for each Event Triggered message frame. These special schedule tables contain the Unconditional message frames that access the same message data that would be sent by each individual slave that might have responded to the Event Triggered message frame ID.

*A LIN cluster that might operate in several different configurations could be configured with two 'operational' schedule tables and the application software could choose which schedule table to activate once the cluster configuration is known. Each of these schedule tables must be named.

3.5 Special features and functionality

The definitions in 'Constant and Macro Definitions' in **lin_dev.h** are used to control conditional compiling to include (or not) code for specific features or functionality. If a feature or functionality is to be used, the #define statement must be uncommented otherwise left out. <u>Start with the demo code and leave these as they are until you want to add or subtract something</u>

USE_POWER_CONTROL – This define is used by both Master and Slave nodes to enable the API functions `l_ifc_goto_sleep()` and `l_ifc_wake_up()`. For Master nodes, the Break detect input pin will not be used by the LIN interface if this define is commented out.

SYNC_AUTO_MODE – This define is never required for a Master node, but may be uncommented for Slave nodes. If "SYNC_AUTO_MODE" is defined, the width of the Sync pulse will be measured and the Baud rate of the Slave node will be adjusted accordingly to "fine tune" it to the Baud rate generated by the Master node. The LIN 2.0 Specification requires that the maximum Baud rate error between two LIN devices not be greater than 2%. The specified accuracy for a Master node is 0.5%. The required accuracy for a Slave node that communicates only with the Master node is 1.5%. The required accuracy for each of two Slave nodes that communicate with each other is 1%. The Renesas MCU's can generate 19200 Baud with an accuracy of 0.16%. If the total maximum error of the main CPU clocks on the Master and Slave nodes does not exceed 0.68% (6800PPM), it is not necessary to define "SYNC_AUTO_MODE" for Slave nodes.

AUTO_CALCULATION – This define is required only if it is desired to calculate the values for the Baud rate generator and break detector under program control. Otherwise, the default values calculated by hand and entered in "lin_hdw.c" will be used.

ENABLE_ODD_SIGNAL_WIDTHS – This define allows the processing of Boolean signals or signals with a bit width not evenly divisible by 8. If this define is commented out, only signals with an 8-bit or 16-bit width may be transferred over the LIN interface.

ENABLE_BYTE_ARRAYS – This define enables the use of the LIN API function calls "l_bytes_rd()" and "l_bytes_wr()", which read and write multi-byte arrays to LIN signals. If this define is commented out, the maximum width of a signal is limited to 16 bits.

EXTENDED_ERROR_STATS – This define enables the use of error counters to allow tracking of specific errors and the number of them that occur. The standard LIN status message simply indicates that an error occurred. The specific source of the error is not indicated, nor is any count of total errors maintained. This define enables 4 8-bit counters to tally specific transmit errors and 3 8-bit counters to tally specific receive errors.

LIN_SPEED – Defines the LIN bus Baud rate in bits per second.

LIN_TIME_BASE – Defines the basic interval in milliseconds used by the Master node schedule tick timer. The actual slot time allocated by the Master node may be some multiple of this time to allow "tailoring" of the slot time to the actual data length required by any given message frame.

LIN_NUM_RETRIES – Defines the number of times the Master node will resend an Unconditional message frame that generated a transfer error, before giving up and sending the next frame in the schedule table. This define must always be set for Master nodes. Legal values are 0 through 255.

LIN – This define for "LIN" must remain **as is** for both Master and Slave nodes.

Device node names Each device in the system must be assigned a unique name. These names are "enumerated" to allow them to be matched using simple numerical compares as opposed to using string searches. The node names used are completely left up to the system designer. The names are used only within a node and are never transferred across the network. The numbers assigned are also completely left up to the system designer. The numbers assigned must not be sequential. The only restriction on these names and the numbers assigned is that each must be unique.

LIN_MY_NODE_NAME – This define is required for all nodes. The assigned name must agree with one of the names defined above.

LIN_MASTER_NODE – Some name (defined in "lin_dev.h") must be given to the Master node. For a Slave node, this name should be "LIN_MST_OFF".

If desired, it is also possible to define names for each of the LIN Message Frame ID's used in the system. However, this is almost a waste of effort. The only place these ID's are actually used is in the "LIN_id_table[]" in "lin_dev.c". It is just as easy to list the ID's directly in the appropriate field of this structure, although using a defined mnemonic might make it easier to "read" this table visually.

4. Customizing lin_hdw.h/c

lin_hdw.h

Set `LIN_TXCVR_ENABLE` to the port which you wish to use to control this faction.

lin_hdw.c

Leave these as they are unless you change the baud rate or the system clock.

`LIN_wake_up_period` – The value to be loaded into the Break timer prescaler register to generate a 10ms "tick" to the Break timer main count register. If the Break timer architecture used for the LIN device does not break up the timer into two sections (prescaler and count), this value should be set to 0.

`LIN_break_prescale` – The value to be loaded into the Break timer prescaler register to generate "ticks" at the frequency required for Break pulse generation or detection. This count (plus 1) multiplied by "LIN_break_count" (plus 1) multiplied by the period of the clock used to drive the timers should equal 13 pit periods (Master node) or 11 bit periods (Slave node) of the Baud rate selected for use in the LIN cluster.

`LIN_break_count` – The number of "ticks" of the Break timer required to generate the break pulse (Master node, 13 bit periods) or to detect a break pulse (Slave node, 11 bit periods).

`LIN_smr_value` – The value loaded into the UART mode register to set the prescaler select bits for the Baud rate generator. Normally, for higher Baud rates, no prescaler will be required.

`LIN_brr_value` – The default divider value for Baud rate generator required to generate the Baud rate defined for "LIN_SPEED".

5. Using the LIN API

5.1 Start up API calls

To run your application the LIN API is provided according to the standard set out in LIN 2.0.

See the demonstration project for your R8C. Care must be taken during device initialization that none of the I/O ports used for the LIN interface are disturbed in any way.

To start, three LIN API functions calls must be included in the application for both master and slave nodes. Their task is to initialize the LIN interface and to prepare it for operation.

- `l_sys_init()`
- `l_ifc_init(LIN)`
- `l_ifc_connect(LIN)`

These three calls can be done directly as shown with no delays or other code intervening. These lines should be placed early in the device initialization as in the demonstration project.

For master nodes, two more LIN API call must be made.

- `l_sch_set(LIN, OP_SCHED, 0)` to select the active schedule table and to start LIN operation.

In the source code, `LIN` is the interface number defined as 0 in lin_dev.h. `OP_SCHED` is the name of the schedule table to use, defined by enumeration in lin_dev.h and 0 is the initial entry in this schedule table to be used.

5.2 Interrupt API calls

The message frame interval varies according to the length of the message and is configured as a multiple of the tick interval.

- `l_sch_tick()` has to be called every 1ms or so by the schedule timer interrupt to clock the start of the next message frame. A dedicated mcu timer configures this frequency. Master only.

The interrupt vector for the timer chosen as the LIN break timer, and the transmit and receive interrupt vectors for the UART chosen for the LIN interface, must be set to call LIN API functions.

- The break timer interrupt must call `l_ifc_aux()`
- The UART receive interrupt must call `l_ifc_rx()`
- The UART transmit interrupt must call `l_ifc_tx()`

If the processor offers only a single UART interrupt to be used for both transmit and receive functions, this interrupt must call `l_ifc_rx()`. The code for this function will determine if the TX function needs to be called and branch to that function independently.

For master nodes, the interrupt vector for the timer chosen as the schedule timer must be set to call `l_sch_tick()`, but only if the define statement for `DEDICATED_SCHEDULE_TIMER` is uncommented in **lin_hdw.h**. These interrupt vectors are set in **sect30.inc**.

The file **lin_hdw.c** contains the interrupt vectors to make the calls to the appropriate LIN API functions. The names used for this purpose must be the names used in the interrupt vector table in **sect30.inc**. For master nodes where the define for `DEDICATED_SCHEDULE_TIMER` is commented out, the user application is responsible for calling `l_sch_tick()` every `LIN_TIME_BASE` milliseconds.

5.3 Data Transfer API calls

Data transfer between the LIN API and the application is accomplished through the signals.

5.3.1 Read LIN signal data

When data is received over the LIN bus it is compared to the data already present in the signal buffer. If the data has changed, the new data is copied into the signal buffer and the signal's update flag is set. The application should poll the update flags of all received signals by using the LIN API function `l_flg_tst()`. When an updated signal value is detected, the application can read out the new data by using one of the LIN API data read function calls

- `l_bool_rd()`
- `l_u8_rd()`
- `l_u16_rd()`
- `l_bytes_rd()`

The signal's update flag will be cleared when the data has been read.

5.3.2 Write LIN signal data

When the application has data that it wishes to transmit, the data is written to the appropriate signal using one of the LIN API data write function calls, such as

- `l_bool_wr()`
- `l_u8_wr()`
- `l_u16_wr()`
- `l_bytes_wr()`

These functions set the signal's associated update flag so the data will be sent the next time the associated message frame is sent over the bus. After the data has been sent, the signal's update flag is cleared. The application may check whether the signal has been transmitted (whether the flag is still set) by calling the API function `l_flg_tst()`.

5.4 x API quick reference

See also the LIN 2.0 specification, yours to download from Lin Consortium Web Site.

The above sections have described everything that the application must do to interface to the LIN bus via the LIN API. However, the LIN 2.0 Specification does define a number of other API function calls that may be used by the application in special circumstances to implement out-of-the-ordinary functionality such as dynamic node configuration. The LIN API function calls are listed by categories below. The descriptions and use of the function calls are explained in the LIN Specification.

Basic startup and initialization API function calls required for every LIN device:

- `l_sys_init()`
- `l_ifc_init()`
- `l_ifc_connect()`
- `l_sch_set()` – master only

API function calls required for every LIN device, but used as interrupt service routines only:

- `l_ifc_rx()`
- `l_ifc_tx()`
- `l_ifc_aux()`
- `l_sch_tick()` - master only

Data read and write functions:

- `l_bool_rd()`
- `l_u8_rd()`
- `l_u16_rd()`
- `l_bytes_rd()`
- `l_bool_wr()`
- `l_u8_wr()`
- `l_u16_wr()`
- `l_bytes_wr()`

Signal update status functions:

- `l_flg_tst()`
- `l_flg_clr()`

System flags

In addition to the signal update flags provided for each signal (and accessed using the defined signal names), the LIN API provides the following system flags:

- `LIN_TX_END_FLG`

- `LIN_RX_END_FLG`
- `LIN_AWAKE_FLG`
- `LIN_DIAG_RESP_FLG`

The `LIN_TX_END_FLG` indicates the end of a data transmission. This flag must be cleared by the application. The `LIN_RX_END_FLG` indicates the end of a data reception. This flag must be cleared by the application. There is no real defined use for either of these flags, and the application may ignore them entirely. The `LIN_AWAKE_FLG` indicates (when set) that the LIN device has not been placed in sleep mode.

Interface control

Other interface control function calls (not usually required by the application):

- `l_ifc_disconnect()`
- `l_ifc_goto_sleep()`
- `l_ifc_wakeup()`
- `l_ifc_read_status()`
- `l_sys_irq_disable()`
- `l_sys_irq_restore()`

HW configuration

- `l_ifc_ioctl()`

Nothing has been defined for this function to do in the Renesas sample code.

In the UNIX world, IOCTL calls perform actions such as setting baud rate, flow control, parity for a serial interface etc. Actions as are required for LIN are carried out by the LIN API function `l_ifc_init()`. The LIN 2.0 specification does not list any specific actions that this function call is to perform and there really doesn't appear to be any necessary action left over for the LIN API function `l_ifc_ioctl()` to perform – at least for the various Renesas families of MCU's. Perhaps some specific LIN transceiver chip might require some sort of special handling that might be implemented in this function call. If such a requirement does exist for some specific user application hardware, the user must add the necessary code to implement the required actions in **lin_api.c**.

6. Porting to your own hardware

6.1 Summary

Porting the code to your own hardware, using the same MCU, means reassigning port inputs and outputs according to your own board design. The external ports used by the demo drive, and that need to be changed are

- NSLP on the transceiver
- LEDs
- LCD
- Switches
- Varistor for reading A-D value

Below are shown the files board design dependencies. (Also shown are the files' MCU dependency).

File	Board design dependency	Processor dependency

lin_dev.c *user generate from template*	*Independent*	*Independent*
lin_dev.h *user generate from template*	*Independent*	*Independent*
lin_hdw.c *user edited*	*Dependent*	*Dependent*
lin_hdw.h *user edited*	*Dependent*	*Dependent*
lin_api.c	*Independent*	*Processor family dependent*
lin_api.h	*Independent*	*Independent*
lin_low_level.c	*Independent*	*Processor family dependent*
lin_low_level.h	*Independent*	*Independent*
sfr_xxx.h *Special function registers*	*Independent*	*Processor dependent*
ncrt0.a3, sect30.inc *user edited*	*Independent*	*Processor dependent*

Dependencies when porting the LIN source code to user's own design. Also shown are the processor dependency of the files

To port a LIN device from one Renesas MCU to another, use **lin_dev.c**, **lin_dev.h**, **lin_api.h** and **lin_low_level.h** without change. Replace the existing **lin_api.c**, **lin_low_level.c** and **sfr_xxx.h** files with files appropriate to the new processor, no editing required (permitted!). Finally, replace the existing **lin_hdw.c**, **lin_hdw.h**, **ncrt0.a30** and **sect30.inc** files with template files appropriate to the new processor and edit these files as required by the new hardware design. The template files provided in the sample codes are well commented to make this task as easy as possible.

6.2 lin_dev.c and h

The **lin_dev.c** and **lin_dev.h** files are completely independent of MCU and board design and are completely the responsibility of the customer – most specifically the system designer of the LIN cluster.

As far as user edits are concerned (the hardware design dependent part), as much as possible is concentrated in the processor dependent template files **lin_hdw.***. These files are all MCU family specific, meaning a user will have to select the appropriate file set for the processor being used as the start point for his design (or port to a new processor). But then these files will have to be edited for hardware design variables such as MCU resources used, clock speed, whether or not the LIN transceiver uses an enable pin, etc. The MCU startup files **ncrt0.a30** and **sect30.inc**. are also MCU specific although they should essentially be able to be used as are.

6.3 lin_api, lin_low_level and sfr files

lin_api.*, **lin_low_level.*** and **sfr_xxx.h** are MCU family dependent and require no user edits. The user should be strongly discouraged from editing any of these files. Absolutely nothing in these files is either LIN device dependent or hardware design dependent. **lin_api.h** and **lin_low_level.h** are identical for all MCU families and types, but the **sfr_xxx.h** files are completely different for each MCU family and may even differ for specific MCU's within a family. The **lin_api.c** and **lin_low_level.c** files are different for each MCU family, primarily dependent upon the byte order used by the MCU (big-endian vs. little-endian), and on whether or not the MCU offers the Hardware LIN block or not. <u>To the greatest extent practical, the processor dependent code is gathered in **lin_low_level.c**</u>. The original intent was to make **lin_api.c** identical for all MCU families, and the only truly significant code difference is in the processing of the break timer interrupt in l_ifc_aux(). But as there are differences in both files, **lin_api.c** and **lin_low_level.c** must be selected for the MCU family used. The vast majority of code in these files IS identical, so the customer can be assured that a LIN device that runs on one of our processors will run in an identical manner on another of our processors as over 90% of the code that remains is identical.

Appendix A. Third party solutions

A.1 LIN/RS-232 Converter

The LIN/RS-232 Converter allows a personal computer to act as a diagnostic analyzer during development and testing of an automotive ECU (electronic control unit) that supports the LIN communications protocol.

More information: For more information, visit the Silicon Engines Company Website.

7. More information

Documents and training

- Doc. REU05B0069. Renesas LIN Basics and Overview.
- Doc. REU05B0070. Customer frequently asked questions.
- Doc. REU05B0078. Renesas LIN R8C Demo and Quick Start. 'QSG' for setting up our RSK boards and the LIN demo. Explains how the demo is built regarding frames & signals.
- Doc. REU05B0079. Renesas LIN configuration and API. How to set up a project and use the LIN API.

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	July 6 2007	—	First edition published to RTA LIN download page.
1.01	Aug 24 2007	—	Second edition after formal review at RTA AE.
1.02	Oct 02 2008	14	Disclaimer changed
1.03	Dec 01 2008	13	More information changed