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 Quick Facts & Frequently Asked Questions

Introduction

LIN is an electronic device serial bus communication network intended as a low-cost alternative to CAN, particularly when a very low cost hardware implementation is needed. LIN offers a very low pin count and reduces wiring complexity. It is also noise tolerant due to its relatively low speed, low slew rates, a multi-sampling UART and since the transceivers add a capacitive load to the bus.

Contents

1. What criteria determine the choice of using LIN as a communication BUS? What types of applications are suited for LIN?

- LIN enables you to have inexpensive remote modules that communicate over an inexpensive bus as opposed to having everything wired into one central module with a single MCU. Only a microcontroller with a common UART interface is necessary.

- The standardized approach of LIN and of the Renesas source code gives the possibility of module re-use in multiple products, thus reducing design time, inventory cost and manufacturing expense. The Renesas LIN API firmware reduces application code development effort and cost.

- The size of the network should be max 16 nodes. LIN is limited to 64 identifiers and relatively low transmission speed (<100kBits/s).

- When synchronization of slave nodes would be too expensive using a crystal oscillator. See question 'Do I need a crystal for my LIN devices?'

- LIN is often compared to I2C or a simple UART based serial protocol such as RS232. See separate tables in this document comparing LIN with I2C and Uart communication.

- See also 'Home-Appliance-Design_IATC_2006_Final.pdf' which discusses the advantages of using LIN.

2. What transfer speeds are available with LIN?

- 2 to 20 Kbit/s. The LIN spec says a maximum of 20 kBaud. This is mainly to reduce EMI as the rise and fall times are to be limited by the transceiver. Most applications use 19 200 baud, since that is a reliable baud rate most micros can generate. Some automotive users have 'standardized' on 9600 Baud for even lower EMI values. If you want to go 'non-spec', you can go as high as your transceiver and bus characteristics will allow. 100 kbaud should not be a problem to reach.

3. Do I need complete LIN specification compatibility, or do I simply need a convenient way to communicate serially between a number of devices?

- LIN is a standard, but you could use LIN in a number of ways that are not according to the official LIN specification. For instance, LIN specifies a 12V bus, a maximum of 16 devices (based on 'standard' driver characteristics) and a maximum bit rate of 20kBaud. If you need LIN spec compatibility, you have to meet these requirements. If you are designing all of the modules in your system (and simply need a convenient and inexpensive method of implementing inter-module communications), then you can use a subset of the LIN features and requirements.

4. What benefit does the R8C MCU's LIN HW block give me?

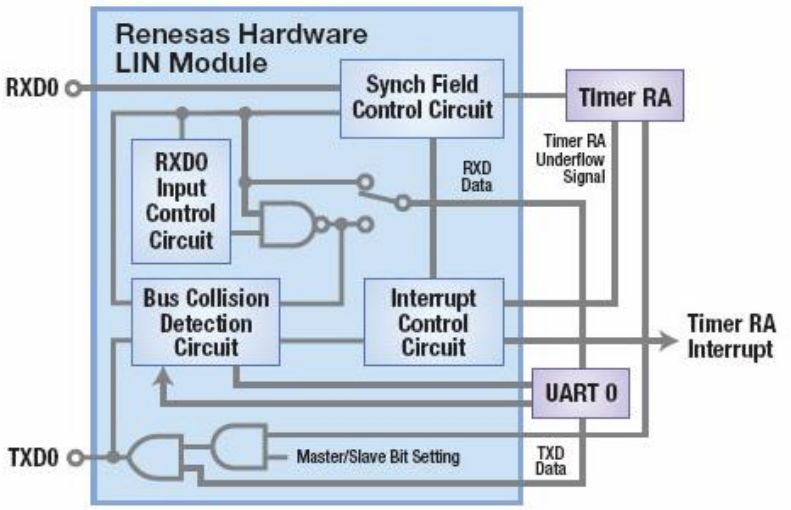

Fig. The hardware LIN block controls Timer RA to simplify Break pulse detection (Slave) and Break pulse generation (Master). It also detects bus collisions and facilitates baud rate adjustment.

The R8C on-chip LIN HW peripheral block utilizes UART0 and Timer RA for communication and timing. The LIN peripheral facilitates the following tasks:

1. Detecting and generating break pulses.
2. Measurement of the synch field duration (so a node can measure the master's synch pulse length and fine tune its internal baud rate generator)
3. Detect bus collisions: It can respond in a single bit time and back off without corrupting data to improve collision response. The peripheral detects bus collision and can respond in a single bit time and back off without corrupting data through the RxD/TxD comparator and thus reduce CPU overhead, especially when detecting breaks.

Comments:
- Renesas offers a LIN 2.0 compliant API for the R8C, M16C and H8/Tiny families of processors. The R8C family provides a function block we call "Hardware LIN". This is not at all a complete hardware LIN block, but rather a special, internal hardware connection of Timer RA, UART0 and some limited control hardware that simplifies the software implementation of LIN. The Renesas on-chip LIN hardware peripheral block of the R8C family lowers the processor load because of the reduction in the amount of source code needed, as certain tasks are carried out by the HW peripheral. It also reduces the number of interrupts that must be processed.

- For instance, using the R8C Hardware LIN, Timer RA is automatically set up to measure the Sync byte after a Break pulse has been detected. Then the Timer RA value is read under software control, and the new load value for the Baudrate generator is calculated. So, the HW block automatically detects a synch pulse, greatly alleviating software interrupts, but also simplifies measurement of the synch pulse length by generating a specific interrupt. When a slave processes this interrupt it reads the length of the synch pulse and can then adjust the baud rate. We call this auto synchronization. This makes it possible to eliminate the need for a slave to be clocked from a crystal and instead use the internal oscillator. See question 'Do I need a crystal for my LIN devices?'

For the M16C and H8/Tiny families of processors, the entire Sync byte measurement process is under software control.

5. Do I need a crystal for my LIN devices?

- In short: The master needs a crystal but the slaves don't.

- The LIN master and slave devices must run reasonably close to the same baud rate, since LIN is an asynchronous protocol. The LIN spec requires Master clocks to be accurate to 0.5% and the Baud rate "error" between any two LIN devices to be 2% max. Renesas LIN offers a mechanism by which slave devices can synch up to the baud rate used by the master. This is called 'Sync-Auto'. The slaves use this auto synchronization mechanism using a timer and software

able to fine tune the baud rate. Slave clocks must be within 14% of the master clock in order for this synchronization process to cut the error down to less than 1.5% error.

- For the slaves to do this a timer is set up to automatically measure the synch pulse length, and generate an interrupt using - in the case of HW LIN - 'Synch Field measurement interrupt enable' if, 'Synch-Auto' in the LIN source code is defined. When this interrupt is taken, a slave measures the synch pulse length and adjusts a baud rate ratio register.

- This eliminates the need for slaves to run off a crystal and use the MCU's internal oscillator instead.

- Currently, (Apr '07) SW Sync-Auto is only available for H8Tiny and HW Sync-Auto for R8C HW LIN devices. SW LIN for other devices pending.

6. Can I use LIN with just a UART - without a transceiver?

- In short: If the transmitted data can be fed back to the RxD pin. The MCU TxD outputs will need to be open drain – or use an open drain buffer. Once the TxD outputs are open drain, however that is accomplished, the TxD output can be connected to the RxD input on each node – and then all nodes connected together. Probably not a recommended practice, but possible.

- To be more detailed, the LIN bus uses a wired-AND signal line that is passively pulled up to 12V for the "standard" bus. The standard 12V signal level provides increased noise margin for more reliable operation. The bus signal rise and fall times are controlled to limit EMI. The LIN transceiver provides conversion from the totem-pole outputs on the MCU to the open-drain outputs required by the wired-AND bus configuration, the transceiver provides level translation from 12V to the logic levels used by the MCU I/O pins, the transceiver provides increased bus drive capability and the transceiver provides the signal slew rate limiting (rise and fall time control).

If:

- the UART TxD output (and the break timer output for a Master node) have an open-drain mode,
- the noise margin provided by a logic level bus is sufficient,
- the MCU outputs have sufficient drive capability to handle the bus load,
- controlled bus signal rise and fall times are not needed (to limit EMI),

then a LIN transceiver circuit is not needed and the LIN node MCUs could be wired directly together. The result would be a non-standard LIN bus configuration.

- If a subset of the LIN transceiver functionality is needed, some simple interface circuit can be built using discrete components. The component cost of such a discrete solution lies in the range of $0.30 to $0.35. This has been done in a number of designs, but lately commercial LIN interface chips have been used instead. LIN interface IC's can now be purchased for less than $0.45 in under 100K quantities. When you consider the total cost of building a discrete interface circuit (component cost, inventory cost, component placement cost, testing cost and PCB space used) and the more limited functionality of the discrete solution, it is hard to beat the price of a commercial LIN interface IC.

7. What is the minimum subset of API calls I need if I just want to use unconditional frames to get data across a network, i.e. what API calls must I learn first?

- For the R8C/23 Starter Kit there is a demo that only uses Unconditional frames. This is suitable to start with to become familiar with LIN and Renesas' code. This code is also the most tested by the tech. support department in the U.S. Read 'Renesas LIN R8C configuration and API' for directions how to create your own project. There is also some information on what API calls are the most common, or basic. Upon request, there is a full set of source code that includes code for event triggered, sporadic, and diagnostic frames.

8. What times the device publishing data for a particular frame?

- No timer 'runs' the transmission of a slave. It is the receive interrupt LIN_Rcv_ISR that starts the node up for every received byte and calls API l_ifc_rx(). The node checks its state and whether all conditions are met for it to publish data.

9. Do I need a dedicated timer interrupt for each byte going out?

- An interrupt is generated each time the transmit shift register is empty and the interrupt routine LIN_Xmt_ISR trips, which in thurn calls the API l_ifc_tx() to send the next byte, if one remains to be sent.

10. Do my non-LIN application interrupts need to be disabled when a LIN frame is being processed?

- Definitely not. The LIN API does occasionally disable all interrupts when something critical is being done that the application may not disturb, but this is infrequent and only for very short periods. The application never needs to do anything along these lines.

-To be a bit more exact, interrupts are turned off when the LIN state variable is being updated in lin_change_bus_status(), which aswell as updates the state variable also updates some 5 MCU registers, or when a node updates its signal data field in the LIN_signal_table (see e.g. l_bytes_wr()).

11. What is the maximum time I can block a pending LIN service interrupt from being serviced if my application needs to tend to a more important interrupt? (I realize that it depends on the baud rate, but is there some general rule I could use?)

- As you said, this is highly dependent upon the baud rate. A master node is more tolerant of delay than a slave node. A slave node that doesn't use Auto Synch is more tolerant of delay than one that does. An R8C slave is more tolerant of delay than an M16C or an H8/Tiny, in that order, because of differences in how the break pulse is detected. In principle, the limiting time for a slave node has to do with break detection. For that reason, a slave application should not block LIN API interrupts for more than 2 bit periods. The R8C can probably push that number out to 3 bit periods. A master node must only worry about not missing received data bytes. For that reason, a master application should not block LIN API interrupts for more than 8 bit periods.

12. What physical aspects affect how far I can run the bus?

- As with any transmission line, the most obvious limiting factor is that the bus must settle out in less than half a bit time. For the LIN bus, the critical factor is the transition from 0 to 1, since the bus is passively pulled up. The single wire LIN bus does not have a controlled impedance, nor is it terminated in the traditional sense. For short bus lengths, these factors really don't matter. For longer buses, they may due to reflection propagation time.

- The pull-up resistance on the bus and the total bus capacitance, both driven by the transceiver circuit design, will determine the bus settling time. This settling time may not be longer than half a bit period, at most. If the bus gets long enough, it is possible that the settling time will be extended by signal reflections due to abrupt impedance changes, but this should not play a role at 19200 Baud unless the bus length gets out to maybe 300 m. The main thing is to control the total bus capacitance – node capacitance plus wiring capacitance. With 'standard' LIN transceiver circuits and wiring capacitance of 50 pF/m, you should be able to get out to 100 m at 19200 Baud. Beyond that, you will need to control the capacitance by reducing the number of nodes, reducing the node capacitance and reducing the cable capacitance. And you might have to drop the bit rate to 9600 baud.

- Repeaters are not for anything close to 'standard' LIN. The LIN bus is realized as a single wire, bidirectional bus. The bus logic state 1 is indistinguishable from the bus idle state. Any node on any bus segment must be able to detect a collision with any other node on any other bus segment. This is not an easy task. A solution would be to two or more LIN clusters be used in situations where a single LIN bus cannot be run far enough to reach every desired node. These LIN clusters could be joined by nodes that function on both clusters – a dual Slave node or a Slave/Master node. This cluster interface node could be programmed to relay specific signal data from one cluster to the other. This would be much more practical than trying to implement a true LIN bus repeater.

13. Can a LIN slave be made to wake up from Wait mode when LIN activity occurs on UART0 and resume communication?

With "peripheral function clock does not stop in wait mode" set on e.g. an R8C, any peripheral interrupt can wake up the MCU. They do need to disable other interrupts that otherwise will wake the MCU right before entering Wait mode and reenabling them upon returning from the LIN receive interrupt.

The time from Wait Mode to Interrupt Routine Execution (40 cpu cycles + 30 us) must be taken into account, but according to the LIN spec there is a 40% additional space allocated compared to the whole transmission time for slave responses.

If you are running the peripheral clock at normal speed, you shouldn't have to do anything, except the first thing the Master should do upon waking up itself is send a break pulse to start the first message. In the slave, Timer RA will be

looking for a break pulse. Since Timer RA is operating at normal speed, it will detect this break pulse and trigger the LIN_Break_ISR. This ISR will do virtually nothing (set the LIN interface into the Sync Receive State) and return, to let the CPU pick up after the NOP's. The wakeup time (150 us – or three bit periods) will be long enough that the LIN interface will enter the Sync Receive State too late to process the sync byte (so the first message will be lost), but other than that, the LIN interface will come back up as if nothing had happened. So just halt the CPU and wait for the next break to come in. This is valid only if the frequency of the peripheral clock remains unchanged. The wakeup interrupt will come from LIN_Break_ISR, not LIN_Rcv_ISR, not that that should make any difference.

In the RTA LIN demo version 1.19 and later, a Node A switch puts the demo to sleep.

If you want to save extra power by slowing the CPU clock down, then here is an outline of what needs to be done.

1) When the application is preparing to enter WAIT mode:

```
// First need to disable all other interrupt sources; INT0, Timer RB ...
// Disable LIN
l_ifc_disconnect();

S0RIC = 0;     // disable serial port interrupts
LINCR = 0;     // revert hardware into normal uart mode
//U0CR -> set RI, RE bits
//S0RIC -> set a valid (non-zero) interrupt level to re-enable interrupts
//set global C-variable processor_sleeping = TRUE

// Put CPU to sleep
//
//Add code or function to slow cpu clock down here…
//
_asm("BCLR 1, 01B7H");  // CPU rewrite mode disabled
_asm("FSET I");         // Enable interrupt
_asm("WAIT");
_asm("NOP");
_asm("NOP");   //Pick up somewhere here after sleep…
_asm("NOP");
_asm("NOP");
//
//Add code or function to reset cpu clock…
//
_asm("NOP");
_asm("NOP");
_asm("NOP");
_asm("NOP");

// Re-enable LIN, diagnostic config info has been retained in RAM
l_sys_init();
l_ifc_init();
l_ifc_connect();
// re-enable other interrupt sources, INT0, Timer RB, do other reinitializations as needed.
ProcessorIsSleeping=FALSE;
```

2) The LIN receive ISR needs this added:

```
LIN_Rcv_ISR()
{
if   (processor_sleeping ==TRUE)
    return;    // Pick up from within the NOPs
else
    l_ifc_rx();
}
```

The CPU will freeze after the WAIT instruction is executed. One or more of the following NOP's will be fetched, and perhaps even executed, but somewhere in among those NOP's, the CPU will stop execution. When an interrupt occurs, the interrupt will execute using whatever clock configuration was set before the CPU was halted and return. Then, the CPU will pick up where it left off in the NOP's. The global flag 'processor_sleeping' is to prevent the "l_ifc_rx()" function from executing. It won't have any valid data to operate on and it will take a lot of CPU cycles before it returns. By using the global flag, the ISR will return at once and get the CPU started again.

Some numbers for indication (not validated): With the Xin clock enabled and running at normal speed, the peripheral clock enabled at normal speed and the CPU halted, I would expect the current draw to be at most 1 mA. Using the internal oscillator at 125 kHz, the Xin clock disabled, the peripheral clock enabled at 125 kHz and the CPU halted, I would expect your current draw to be somewhere between 25 uA and 75 uA.

In conclusion, turning off Xin and slowing everything down internally will save you somewhere around 600 uA or 700 uA. But it will make the wakeup more complicated to achieve.

14. Do Renesas' MCUs have autobaud detect? If so, is this a hardware of software feature?

The LIN protocol does not support a true "Auto Baud" function. The LIN cluster nominal Baudrate is a parameter (LIN_speed) that is defined in the LIN Description File (or in our "lin_dev.h" file). The allowed LIN Baudrates range from 1kBaud to 20 kBaud, and the LIN specification requires that the Master node generate this Baudrate with an accuracy of 0.5%. Slave devices must generate this Baudrate with an accuracy of 1.5%. The LIN specification provides for a Sync byte, which Slave nodes can measure to "fine tune" their local Baudrate to meet this 1.5% requirement. The LIN specification states that a Slave node should be able to sync up to the Master node even if the Slave's clock oscillator is off by as much as 14%. This allows a Slave device to use a relatively inaccurate on-chip oscillator.

The LIN Specification has a section called the LIN Node Capability Language Specification. This section defines a BNF syntax that can be used in a file that describes the capabilities a given node can support. Mostly, this applies to Slave nodes. One of the syntax elements this section defines is "bitrate". The allowed options are "automatic min max", "select (from list)" and "bitrate (fixed)". This sounds like a Slave node can be set up to function in an Auto Baud mode. But the LIN Specification itself makes no provision for performing Auto Baud.

Auto Baud could presumably be implemented as follows:

- The Break pulse (generated by the Master node) is low for 13 bit periods minimum.
- The Sync byte is 0x55 – plus start and stop bits.
- Prior to enabling the LIN interface, the application could monitor the LIN bus to detect a valid Break pulse, starting at 1kBaud and working up to 20 kBaud.
- When a potentially valid Break pulse is detected, one or more bit periods of the following Sync byte could be measured to get close to the Master's Baudrate.

The LIN interface could then be enabled at this Baudrate, and the Auto Sync function would "pull in" the Baudrate to the required accuracy.

Implementing this "feature" would mean that the Master node would have to delay any dynamic configuration of the Slave nodes until enough "dummy" frames had been sent to allow the Slave nodes to perform Auto Baud. With that restriction, I think the above procedure would work to provide an Auto Baud capability. But this is not part of the LIN Specification, nor is it a function provided by the Renesas LIN API.

15. How good is LIN at noise rejection?

LIN noise rejection is mostly a function of the LIN transceiver characteristics, aided to some extent by the sampling method employed in the MCU's UART. A LIN spec compliant transceiver chip has 4.8 volts of noise margin at the logic 0 (dominant) and logic 1 (recessive) states, and 2.1 volts of hysteresis about the switching threshold. Since the LIN Baud rates are limited, the LIN transceiver chips can have a slow response time, which tends to "absorb" high-speed transients. The LIN bus can have up to 10 nF total bus capacitance, which also acts to filter high-speed transients. The net effect of all of this is that the LIN transceiver receive output signal is very clean. At most, you might see an occasional noise pulse get through to the MCU's RXD input while the bus is transitioning. The LIN bus signal is deliberately shaped with slow transition times in order to limit EMI. This extends the "hazard" period when a noise pulse might slip through. And that is where the UART's sampling characteristics come into play.

Since the LIN UART operates in the asynchronous mode, it is not susceptible to false clocks at the transition point. The LIN RXD signal is sampled near the middle of the expected bit period to determine the bus state. Most Renesas UARTs (including the R8C family) are “oversampled”, meaning that several samples are taken close to the middle point to provide greater immunity to high-speed transients.

All of these factors, taken together, provide a bus system that is highly immune to noise. That is a primary reason why the LIN bus has found so much acceptance in the commercial/industrial market.

16. How big is the Renesas LIN stack?

There is no one answer to this question. The size of the code depends on the LIN functionality used by the device. For the R8C, a minimal LIN Slave device can be as small as 2402 bytes. A Slave device that implements absolutely every option available in LIN uses 6392 bytes. A fairly normal Slave device uses about 4200 bytes. Added to this are maybe 175 bytes of ROM constants. RAM use is maybe 150 bytes. These last two numbers depend mostly on how many signals and message frames the device is configured to process. There is a document that explains all of this in exhaustive detail titled “LIN API Memory Usage.doc” available on request to Technical Support, or from the CAN\LIN download website.

17. Is the Renesas LIN stack fully validated?

The LIN Consortium does not provide a formal process that can be used to validate our LIN stack separately from a specific LIN product. The LIN Consortium has defined a LIN Protocol Conformance Test that can be used to validate performance of a LIN product. There are three test houses in Germany that are certified to test LIN products. I don’t know if any of our customers have ever had their products tested for conformance.

In house, the Renesas LIN API has been thoroughly tested. 13 different LIN devices were built for each of the 3 processor families. These devices were assembled into 4 different LIN clusters. The LIN clusters were then tested with every possible combination of devices. This process was repeated for revision 1.2 of the API. The API originally was written in 2004. It was used the first time in a product in that year. It has been used since then in dozens of products, mostly commercial/industrial. It has been used in automotive products, but the ratio of commercial/industrial products to automotive products is probably at least 3 to 1.

18. Have any customers delivered an automotive product using our stack?

There is no one point within Renesas where LIN products are tracked. Assembling that information would require an extensive survey of Renesas sales offices, rep sales offices and even our customers. Mostly, we interface with customers only when they are starting their first LIN project, and we do not keep track of how many different products a customer subsequently develops with Renesas LIN.

19. Comparison between LIN and I2C/CAN/regular UART

19.1 I2C vs. LIN

- I2C is often compared to LIN. If one considers the 7 layer OSI network model, I2C provides only Layer 1 functionality, the Physical Layer, while LIN provides the functionality for all layers up through Layer 4, the Network Layer. An even greater advantage of LIN over I2C is the provision of functionality up through Layer 6, the Presentation Layer, since the LIN spec defines a standard data interface to the application (l_u8_rd(), l_u8_wr(), etc.). With LIN, the application must only read and write data to the interface using the standard API function calls and signal update flags.

- To summarize, the complete network management is handled by the LIN API. This provides a simple and reliable transport method that can be used by the system designer at virtually zero development cost. With I2C, all of the network management must be implemented in the application code. At the Physical Layer, I2C provides substantially higher data rates, but is more limited than LIN in bus length and the number of nodes that can be attached to the cluster.

- EMI. Using a transceiver normally increases the node’s noise immunity.

- Only if someone needs high data rates, is communicating between a limited number of devices physically located within feet of each other, is not worried about EMI emissions and doesn’t mind implementing the “network management” as part of his application software, I2C is probably the better choice. If one or more of these conditions is not true, LIN may be the better choice.

	I2C	LIN
Nr bus lines	2	1
Max. capacitive load Max. bus length & no. nodes	0.4 nF Both bus lines are close together - higher cap.	10 nF Bus line separate from ground – less cap.
Bus synchronization	Stable oscillator needed for all nodes	sync pulse - enables auto synch
Number of protocol layers	Renesas only layer 1 - physical Layer	Functionality up through layer 6 with its API interface - l_u8_rd(), l_u8_wr()…
Comm. Speed	400 kbps	20 kbps

19.2 CAN vs. LIN

	CAN	LIN
Nr bus lines	2	1
Bus length, no. of nodes	1000 m, 100+	40 m, ~16
Bus clock synchronization tolerance	Clock stability +- 0.5 % Crystal needed	Up to +- 14% possible sync pulse meas. w. auto sync software
Number of protocol layers	Layer s 1 & 2 - Physical l and MAC (Data ink) Layer Renesas has API to read and write at Transport layer.	Functionality up to presentation layer (6) with its API interface l_u8_rd(), l_u8_wr()…
Comm. Speed	1000 kbps	20 kbps

19.3 UART vs. LIN

	UART	LIN
Nr bus lines	2	1
Bus length limitation	50 ft.	40 m.
Number of nodes	2	~16
Bus clock synchronization tolerance	+-5 %	+- 14%. Sync pulse, xtal not needed.
Number of protocol layers	Renesas only layer 1 - Physical layer	Functionality up to presentation layer (6) with its API interface l_u8_rd(), l_u8_wr()…
Comm. speed	1000 kbps	20 kbps

20. 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

		Description	
Rev.	**Date**	**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 06 2008	—	Disclaimer was changed.
1.03	Dec 01 2008	—	Third edition after formal review at RTA AE.

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.