

RLIN3 Module Software Integration System

User's Manual Rev 1.00

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (<http://www.renesas.com>).

NOTES FOR CMOS DEVICES

- (1) **VOLTAGE APPLICATION WAVEFORM AT INPUT PIN:** Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (MAX) and V_{IH} (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (MAX) and V_{IH} (MIN).
- (2) **HANDLING OF UNUSED INPUT PINS:** Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.
- (3) **PRECAUTION AGAINST ESD:** A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.
- (4) **STATUS BEFORE INITIALIZATION:** Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.
- (5) **POWER ON/OFF SEQUENCE:** In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.
- (6) **INPUT OF SIGNAL DURING POWER OFF STATE :** Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

INTRODUCTION

- Audience** This manual is intended for engineers who wish to understand the functions of the LIN 2.1 software driver and design and develop application systems and programs for these devices.
- Purpose** This manual is intended to give users an understanding of the functions described in the Organization below. Refer to “LIN Specification Package Revision 2.1” (In this manual called LIN 2.1 spec).
- Organization** This manual consists of the following items.
- Product overview
 - Installation
 - System build
 - Configuration
 - Driver functions

How to Read This Manual

It is assumed that the readers of this manual have general knowledge of electrical engineering, logic circuits, and microcontrollers.

To gain a general understanding of LIN 2.1 software driver functions, it is suggested to read this manual in the order that the material is presented.

- Conventions**
- Data significance: Higher digits on the left and lower digits on the right
- Active low representation: xxx (over score over pin and signal name)
- Note:** Footnote for item marked with Note in the text
- Caution:** Information requiring particular attention
- Remark:** Supplementary information
- Numeric representation: Binary ... xxxx or xxxxB
Decimal ... xxxx
Hexadecimal ... xxxxH
- Units for representing powers of 2 (address space or memory space):
K (kilo): $2^{10} = 1,024$
M (mega): $2^{20} = 1,024^2$
G (giga): $2^{30} = 1,024^3$
- Data type: Word ... 32 bits
Halfword ... 16 bits
Byte ... 8 bits

CONTENTS

INTRODUCTION	3
CHAPTER 1 PRODUCT OVERVIEW	10
1. 1 General	10
1. 2 Features	10
1. 2. 1 High portability	10
1. 2. 2 Provision of Configuration tool	10
1. 3 Types of LIN Software Driver	10
1. 4 Execution Environment	11
1. 4. 1 Target CPU	11
1. 4. 2 Memory usage	12
1. 4. 3 Hardware resources	13
1. 5 Development Environment	14
1. 6 Restrictions	15
1. 6. 1 Clock and baud rate setting	15
1. 6. 2 Restrictions regarding interrupt	15
1. 6. 3 Other restrictions	16
CHAPTER 2 INSTALLATION	17
2. 1 General	17
2. 2 Installation Steps	17
2. 1. 1 Installation of LIN Configurator	17
CHAPTER 3 SYSTEM BUILD OVERVIEW	18
3. 1 Position of LIN 2.1 Software driver	18
3. 2 Creating the LIN System	19
3. 2. 1 File generation by LIN Configurator	20
3. 2. 2 User applications	21
3. 2. 3 Build	21
CHAPTER 4 Linkage to Smart Configurator	22
4. 1 Operating procedure	23
4. 2 Configuration value	24
4. 2. 1 Configuration for LIN Configurator	24
4. 2. 2 Configuration for LIN driver	25
CHAPTER 5 HOW TO BUILD LIN APPLICATION	27
5. 1 Build driver library	28
5. 1. 1 Compiler options for library	29
5. 1. 2 Edit of confmlin_opt.h (for Master)	30
5. 1. 3 Edit of confslin_opt.h (for Slave)	31
5. 1. 4 Specify the IAR I/O header file	32
5. 2 Create LIN application code	33
5. 2. 1 Creation of development environment project file	34
5. 2. 2 Craation of channel-only source code	34
5. 2. 3 Peripheral hardware processing implementation	35
5. 2. 4 Describe compiler options (conflin_x.h)	37
5. 2. 5 Edit public constants (conflin_x.c)	38
5. 2. 6 Scheduler implementation (only master)	39
5. 2. 7 User-defined callouts implementation	40
5. 2. 8 Use Section Setting	41
CHAPTER 6 LIN CONFIGURATOR	43
6. 1 General	43
6. 1. 1 Features	43
6. 1. 2 Execution Environment	43

6. 1. 3 Output Folder	43
6. 2 File Generation Steps.....	44
6. 2. 1 Start Up of LIN Configurator	45
6. 2. 2 Start a New Configuration.....	48
6. 2. 3 Device selection.....	49
6. 2. 4 Channel configuration.....	51
6. 2. 5 Baud rate configuration.....	53
6. 2. 6 Message management	54
6. 2. 7 Schedule management.....	61
6. 2. 8 Node configuration.....	63
6. 2. 9 Other configurations.....	64
6. 2. 10 Save / Load setting file.....	65
6. 2. 11 Generate source code	66
6. 3 Error message list	67
6. 4 Warning message list	70
CHAPTER 7 LIN 2.1 SOFTWARE DRIVER OVERVIEW	71
7. 1 Signal Types (Master/Slave)	71
7. 2 Frame Format (Master/Slave)	72
7. 2. 1 Byte Field.....	73
7. 2. 2 Break Field.....	73
7. 2. 3 Frame Length	73
7. 3 Frame Transfer (Master/Slave)	74
7. 3. 1 Unconditional Frame Transfer	74
7. 3. 2 Event Trigger Frame Transfer.....	75
7. 3. 3 Sporadic Frame Transfer	78
7. 4 Response Error Notify Function (Master/Slave)	79
7. 5 Sleep and Wakeup Function (Master/Slave)	80
7. 5. 1 Sleep Function.....	80
7. 5. 2 Wake up Function.....	80
7. 6 Node Configuration Function (Master/Slave).....	81
7. 6. 1 Node Information	81
7. 6. 2 Node Configuration.....	82
7. 7 Scheduling Function (Master).....	84
7. 7. 1 Schedule Transition (l_sch_tick)	84
7. 7. 2 Schedule Switching (l_sch_set)	86
7. 8 Auto Baud Rate Detecting Function (Option) (Slave)	87
7. 9 Driver configuration	88
7. 9. 1 Slave driver configuration	88
7. 9. 2 Master driver configuration	92
CHAPTER 8 LIN 2.1 SOFTWARE DRIVER FUNCTION (SLAVE)	95
8. 1 LIN 2.1 Software Slave Driver Function List.....	95
8. 2 Data types (Slave).....	96
8. 3 Description of LIN 2.1 Software Slave Driver Function.....	97
8. 3. 1 [Slave] LIN 2.1 Software Driver and Cluster Management.....	99
8. 3. 2 [Slave] Scalar Signal Read	101
8. 3. 3 [Slave] Scalar Signal Write	107
8. 3. 4 [Slave] Byte Array Read.....	112
8. 3. 5 [Slave] Byte Array Write.....	115
8. 3. 6 [Slave] Notification	118
8. 3. 7 [Slave] Interface Management	121
8. 3. 8 [Slave] User provided call-outs	126
CHAPTER 9 LIN 2.1 SOFTWARE DRIVER FUNCTION (MASTER)	131

9. 1 LIN 2.1 Software Master Driver Function List.....	131
9. 2 Data types (Master).....	132
9. 3 Description of LIN 2.1 Software Master Driver Function	133
9. 3. 1 [Master] LIN 2.1 Software Driver and Cluster Management	135
9. 3. 2 [Master] Scalar Signal Read	137
9. 3. 3 [Master] Scalar Signal Write	143
9. 3. 4 [Master] Byte Array Read	148
9. 3. 5 [Master] Byte Array Write	151
9. 3. 6 [Master] Notification.....	154
9. 3. 7 [Master] Schedule Management.....	157
9. 3. 8 [Master] Interface Management.....	160
9. 3. 9 [Master] Node Configuration.....	166
9. 3. 10 [Master] User provided call-outs.....	173
CHAPTER 10 Example of LIN description file (LDF) description.....	178
APPENDIX REVISION HISTORY	182

LIST OF TABLES

Table 1-1. Description of LIN nodes.....	10
Table 1-2. Target CPU (slave)	11
Table 1-3. Target CPU (master).....	11
Table 1-4. Memory usage (slave)	12
Table 1-5. Memory usage (master).....	12
Table 1-6. Hardware resources list (slave)	13
Table 1-7. Hardware resources list (master).....	13
Table 1-8. List of development software	14
Table 1-9. Conditions List	15
Table 4-1. Defined value for LIN Configurator (r_rlin3_config.h).....	24
Table 4-2. Defined value for LIN driver (r_rlin3_config.h)	25
Table 5-1. Compiler Option for Library Usage List	29
Table 5-2. LIN2.1 software driver use sections (CC-RL).....	41
Table 5-3. LIN2.1 software driver use sections (IAR).....	42
Table 6-1. Operation from tree contents, buttons and Tool menu (for Master channel).....	47
Table 6-2. Operation from tree contents, buttons and Tool menu (for Slave channel).....	47
Table 6-3. Error message list.....	67
Table 6-4. Warning message list.....	70
Table 7-1. Frame Component List	72
Table 7-2. Types of LIN Frames	74
Table 7-3. Response Error List	79
Table 7-4. Node Information List.....	81
Table 7-5. Master Request Frame Format.....	82
Table 7-6. Master Request Frame Format (Read by identifier).....	82
Table 7-7. Slave Response Frame Format	83
Table 7-8. Negative Response Format	83
Table 7-9. Driver configuration of RL78/F23,F24 slave (1 of 2)	88
Table 7-10. Driver configuration of RL78/F23,F24 slave (2 of 2)	90
Table 7-11. Driver configuration of RL78/F23,F24 master (1 of 2).....	92
Table 7-12. Driver configuration of RL78/F23,F24 master (2 of 2).....	94
Table 8-1. LIN 2.1 Software Slave Driver Function	95
Table 8-2. LIN 2.1 Spec and LIN 2.1 Slave Driver Type Definition List.....	96
Table 8-3. LIN 2.1 Software Driver and Cluster Management (Slave)	99
Table 8-4. Scalar Signal Read (Slave).....	101
Table 8-5. Scalar Signal Write (Slave).....	107
Table 8-6. Byte Array Read (Slave)	112
Table 8-7. Byte Array Write (Slave)	115
Table 8-8. Notification (Slave).....	118
Table 8-9. Interface Management (Slave).....	121
Table 8-10. User provided call-outs (Slave).....	126
Table 9-1. List of LIN 2.1 Software Master Driver Function	131
Table 9-2. LIN 2.1 Spec and LIN 2.1 Master Driver Type Definition List.....	132
Table 9-3. List of LIN 2.1 Software Driver and Cluster Management (Master).....	135
Table 9-4. List of Scalar Signal Read (Master)	137
Table 9-5. Scalar Signal Write (Master)	143
Table 9-6. List of Byte Array Read (Master).....	148
Table 9-7. List of Byte Array Write (Master)	151
Table 9-8. List of Notification (Master)	154
Table 9-9. List of Schedule Management (Master)	157
Table 9-10. List of Interface Management (Master)	160

Table 9-11. List of Node Configuration (Master)	166
Table 9-12. Error Code List	168
Table 9-13. PID list setting value	170
Table 9-14. List of User provided call-outs (Master)	173

LIST OF FIGURES

Figure 3-1. System Overview.....	18
Figure 3-2. LIN System Creation Process.....	19
Figure 3-3. Relationship between User Application Program and LIN 2.1 Software Driver.....	21
Figure 4-1. Image of linkage with smart configurator	22
Figure 5-1. Flow of building LIN application	27
Figure 5-2. How to describe the include in device.h.....	32
Figure 5-3. How to build LIN application	33
Figure 5-4 Timing of interrupt and LIN frame in RL78 series.	36
Figure 5-5. Location of macro description in conflin_x.h	37
Figure 6-1. Main Screen	45
Figure 6-2. Main Screen (before selection).....	46
Figure 6-3. Device Selection window (before device selection)	49
Figure 6-4. Device Selection window (device applying multi channels)	49
Figure 6-5. Main Screen (before channel configuration)	50
Figure 6-6. Channel Configuration window	51
Figure 6-7. Main Screen (before channel contents configuration).....	51
Figure 6-8. Setting Baud Rate window.....	53
Figure 6-9. Setting Frame window	54
Figure 6-10. Setting Frame window (for each frame).....	56
Figure 6-11. Setting Signal window.....	57
Figure 6-12. Setting Signal window.....	58
Figure 6-13. Setting Event Triggered Frame Entry window	59
Figure 6-14. Setting Sporadic Frame Entry window.....	60
Figure 6-15. Setting Schedule window.....	61
Figure 6-16. Setting Schedule Entry window	62
Figure 6-17. Behavior of the schedule	62
Figure 6-18. Setting Node window	63
Figure 6-19. Setting Others window.....	64
Figure 6-20. Example of output folder.....	66
Figure 7-1. Example of Arranging Signal to Message Data	71
Figure 7-2. Frame Format.....	72
Figure 7-3. Byte Field Format	73
Figure 7-4. Break Format.....	73
Figure 7-5. Example of Data Transfer	74
Figure 7-6. Example of Transfer Data with Event Triggered Frame.....	75
Figure 7-7. Example of Transfer Data with Sporadic Frame	78
Figure 7-8. Relation between l_sch_tick Function and Frame Transfer	85
Figure 7-9. Switch of Schedule Table by l_sch_set Function.....	86
Figure 8-1. Description Format of LIN 2.1 Software Slave Driver Function.....	97
Figure 9-1. Description Format of LIN 2.1 Software Master Driver Function.....	133

CHAPTER 1 PRODUCT OVERVIEW

1.1 General

LIN 2.1 software driver realize LIN communication on many microcomputers made by Renesas Electronics. This driver provides Application Programming Interface based on LIN 2.1 specification.

LIN 2.1Spec is compatible with the following software driver for microcomputers

- 16 bit microcomputer RL78/F23 series
- 16 bit microcomputer RL78/F24 series

1.2 Features

1.2.1 High portability

Users can write LIN communication programs without having to know about the LIN's hardware dependencies or the CPU core. As a result, porting to another environment is easy.

1.2.2 Provision of Configuration tool

GUI-driven commands make it easy to select environment-based initial settings for LIN hardware and other devices to be used, as well as static generation of messages.

1.3 Types of LIN Software Driver

The following table shows the types of driver that are available in the LIN 2.1 software driver library.

Table 1-1. Description of LIN nodes

Type	Description
Hardware dependency	Different drivers are provided for each CPU core.
Master/slave function	2 types are available: one for master applications, one for slave applications.

1. 4 Execution Environment

LIN 2.1 software driver operate on target systems that are equipped with the following hardware.

1. 4. 1 Target CPU

(1) Slave

Table 1-2. Target CPU (slave)

RL78 Series	RL78/F23, RL78/F24
-------------	--------------------

(2) Master

Table 1-3. Target CPU (master)

RL78 Series	RL78/F23, RL78/F24
-------------	--------------------

1. 4. 2 Memory usage

The amount of memory required to use the LIN 2.1 software driver varies depending on the number of functions implemented by the user, the number of message buffers, peripheral I/O used, compiler, and compile options used. The amount of required memory when all functions are used is as follows.

(1) Slave

Table 1-4. Memory usage (slave)

Component	ROM Size	RAM Size
LIN 2.1 Software driver	3.5 KB (RL78/F23,F24:CC-RL) 3.6 KB (RL78/F23,F24:IAR)	28 bytes (RL78/F23,F24:CC-RL) 25 bytes (RL78/F23,F24:IAR)
stack	-	200 bytes
One unconditional frame	5 bytes	14 bytes (16 signals version) 18 bytes (32 signals version)
One signal	4 bytes (16 signals version) 5 bytes (32 signals version)	-
One event triggered frame (per relation of one unconditional frame)	2 bytes	1 byte

*: Separately, other memory capacity for transferring message is needed.

*: In RL78/F23,F24 series, these values are in case of auto baud rate mode and falling edge detection for bus wakeup.

*: These values are not considered align of structure.

*: "16 signals version" means LIN driver not using compiler option `__LIN_SIGNAL_32__`, and "32 signals version" means LIN driver using `__LIN_SIGNAL_32__`.

(2) Master

Table 1-5. Memory usage (master)

Component	ROM Size	RAM Size
LIN 2.1 software driver	4.8 KB (RL78/F23,F24:CC-RL) 4.6 KB (RL78/F23,F24:IAR)	46 bytes (RL78/F23,F24:CC-RL) 45 bytes (RL78/F23,F24:IAR)
Stack	-	150 bytes
One unconditional frame	12 bytes	11 bytes (16 signals version) 13 bytes (32 signals version)
One signal	5 bytes	-
One event triggered frame	17 bytes	-
One sporadic frame	16 bytes	-
One schedule entry	4 bytes	-

*: Separately, other memory capacity for transferring message is needed.

*: These values are not considered align of structure.

*: "16 signals version" means LIN driver not using compiler option `__LIN_SIGNAL_32__`, and "32 signals version" means LIN driver using `__LIN_SIGNAL_32__`.

1. 4. 3 Hardware resources

Each hardware resource utilized from the devices available for LIN use is shown in the tables below.

(1) Slave

Serial interface : RLIN3 channel 0 or 1

Timer Array Unit : TAU00-07, TAU10-17

External interrupt *: INTP11 (INTLIN0WUP), INTP12 (INTLIN1WUP)

* External interrupts are used only when the wake-up type is set to Falling edge detection type. Not used when the method is Dominant width detection type.

Table 1-6. Hardware resources list (slave)

Microcomputers	UART* ¹		TAU* ³	
	Channel	Interrupt request	Unit/Channel	Interrupt request
RL78/F23	RLIN3 (Channel 0)	Transmission completion interrupt : INTLIN0TRM Reception completion interrupt : INTLIN0RVC Error interrupt : INTLIN0STA External interrupt : INTLIN0WUP* ²	TAU00~07 TAU10~13	INTTM00~07 INTTM10~13
RL78/F24	RLIN3 (Channel 0)	Transmission completion interrupt : INTLIN0TRM Reception completion interrupt : INTLIN0RVC Error interrupt : INTLIN0STA External interrupt : INTLIN0WUP* ²	TAU00~07 TAU10~17	INTTM00~07 INTTM10~17
	RLIN3 (Channel 1)	Transmission completion interrupt : INTLIN1TRM Reception completion interrupt : INTLIN1RVC Error interrupt : INTLIN1STA External interrupt : INTLIN0WUP* ²		

*1 : Only one channel can be selected for the UART channel.

*2 : Only one unit or channel can be selected for the TAU unit or channel.

(2) Master

Serial interface : RLIN3 channel 0, RLIN3 channel 1

External interrupt *: INTP11 (INTLIN0WUP), INTP12 (INTLIN1WUP)

* External interrupts are used only when the wake-up type is set to Falling edge detection type. Not used when the method is Dominant width detection type.

Table 1-7. Hardware resources list (master)

Microcomputers	UART* ¹	
	Channel	Interrupt request
RL78/F23	RLIN3 (Channel 0)	Transmission completion interrupt : INTLIN0TRM Reception completion interrupt : INTLIN0RVC Error interrupt : INTLIN0STA External interrupt : INTLIN0WUP
RL78/F24	RLIN3 (Channel 0)	Transmission completion interrupt : INTLIN0TRM Reception completion interrupt : INTLIN0RVC Error interrupt : INTLIN0STA External interrupt : INTLIN0WUP
	RLIN3 (Channel 1)	Transmission completion interrupt : INTLIN1TRM Reception completion interrupt : INTLIN1RVC Error interrupt : INTLIN1STA External interrupt : INTLIN0WUP

*1 : Only one channel can be selected for the UART channel.

1. 5 Development Environment

The following environments are required for use of LIN 2.1 software driver for development of application systems.

(1) Hardware

- Host machine : IBM PC/AT™ Series
- OS : Windows 10
(Microsoft .NET Framework 3.5 is needed)

(2) Software

Table 1-8. List of development software

Microcomputer	Development environment	Compiler	Debugger
RL78/F23,F24	CS+	CC-RL	Included in CS+
	IAR	IAR compiler	Included in IAR

1. 6 Restrictions

The restriction matter this software is used is shown below.

1. 6. 1 Clock and baud rate setting

When using CPU/peripheral hardware clock (f_{CLK}) as the LIN communication clock source, use f_{CLK} and the baud rate on the following conditions.

Table 1-9. Conditions List

Type	Terms of use	
	Peripheral H/W clock (f_{CLK}),	Baud rate (BR)
Master driver	$4\text{ MHz} \leq f_{CLK} \leq 40\text{ MHz}$	9600 bps, 19200 bps
Slave driver (Auto baud rate)	$8\text{ MHz} \leq f_{CLK} \leq 40\text{ MHz}$	$2400\text{ bps} \leq BR \leq 20000\text{ bps}$
Slave driver (Fixed baud rate)	$4\text{ MHz} \leq f_{CLK} \leq 40\text{ MHz}$	9600 bps, 19200 bps

When using CPU/peripheral hardware clock (f_{CLK}) as the LIN communication clock source, use f_{MX} and f_{CLK} on the following conditions. The baud rate range is the same as the table above.

$$4\text{ MHz} \leq f_{MX} \leq 40\text{ MHz} \text{ and } f_{CLK} \geq f_{MX} \times 1.2$$

(However, $4\text{ MHz} \leq f_{MX} < 8\text{ MHz}$ is not allowed when using slave auto baud rate.)

The following LIN communication clock source frequencies can be specified by smart and LIN configurators: If other frequencies are used, the output file must be edited separately.

8, 10, 12, 16, 20, 24, 32, 40 [MHz]

1. 6. 2 Restrictions regarding interrupt

When using the interrupt used by a driver and the interrupt which the same resource is shared, in the user application at the same time, the driver is not executed normally. The interrupt used by the driver is refer to 1. 4. 3 Hardware resources. Please refer to the user's manual (interrupt factor list) of RL78/F23, F24 regarding the specification of the interrupt vector table.

1. 6. 3 Other restrictions

- In RL78/F23, F24 series driver, LIN channel selection register LCHSEL is modified in API and interrupts. However driver doesn't set original value to LCHSEL.
- Application must not to use timer channel same as used by slave driver. And application must not use the timer clock selection same as LIN slave driver.
- In examination of this driver, the values deviating LIN specification set in driver configuration were not considered. If these values are used, examination should be done in your environment.
- In RL78/F23, F24 master driver, the error caused by own hardware is notified to application using callback function `I_sys_call_fatal_error`.
- In RL78/F23, F24 master driver, wakeup baud rate can be changed by `CONFLIN_u2sBAUDRATEWKUP` in `conf/conflin_x.c` . However, use 19200[bps] (default value) as wakeup baud rate usually because driver may not be able to receive wakeup request correctly.
- In RL78/F23, F24 master driver, the source of LIN communication clock can be changed by `CONFMLIN_OPT_u1gLINMCK_CFG` of driver configuration. Set appropriately value after code generation by LIN configurator.

CHAPTER 2 INSTALLATION

2. 1 General

The LIN 2.1 software driver functions are provided in the LIN configurator.

2. 2 Installation Steps

2. 1. 1 Installation of LIN Configurator

Double-click the LINConfigurator_RL78F23_F24_E_V100.msi file (attached to this document), and obey the instructions of Wizard.

In the Smart Configurator output, the installer above is stored in the r_lin3/tool folder.

Note: .NET Framework 3.5 is necessary for using the LIN configurator.
.NET Framework 3.5 can be obtained from Microsoft's web site.

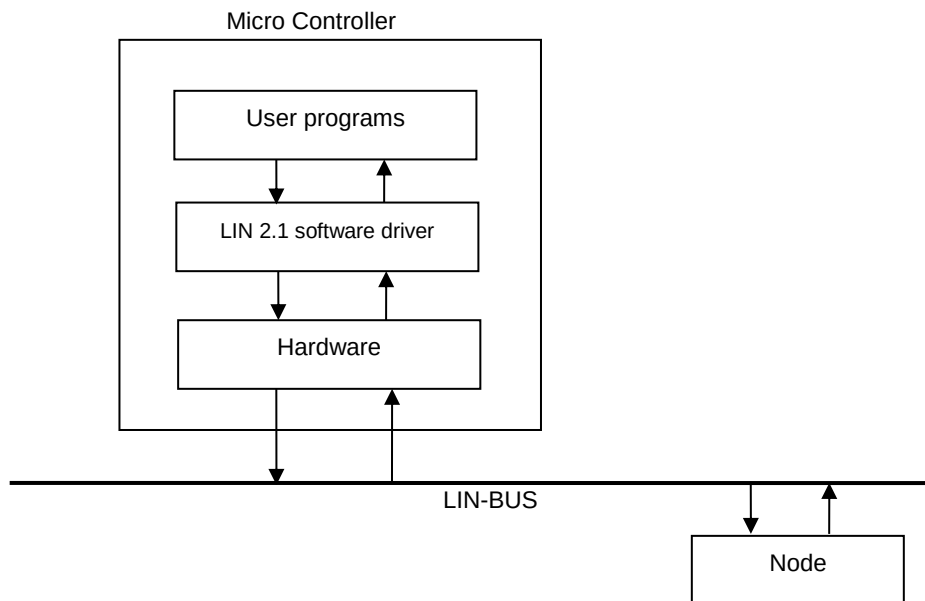
CHAPTER 3 SYSTEM BUILD OVERVIEW

3.1 Position of LIN 2.1 Software driver

In the system, the LIN 2.1 software driver is positioned between the user application and the hardware (see Figure 3-1). A user interface is provided for controlling the hardware.

The user can simply call the LIN 2.1 software driver functions in the application without having to know about controlling the hardware registers.

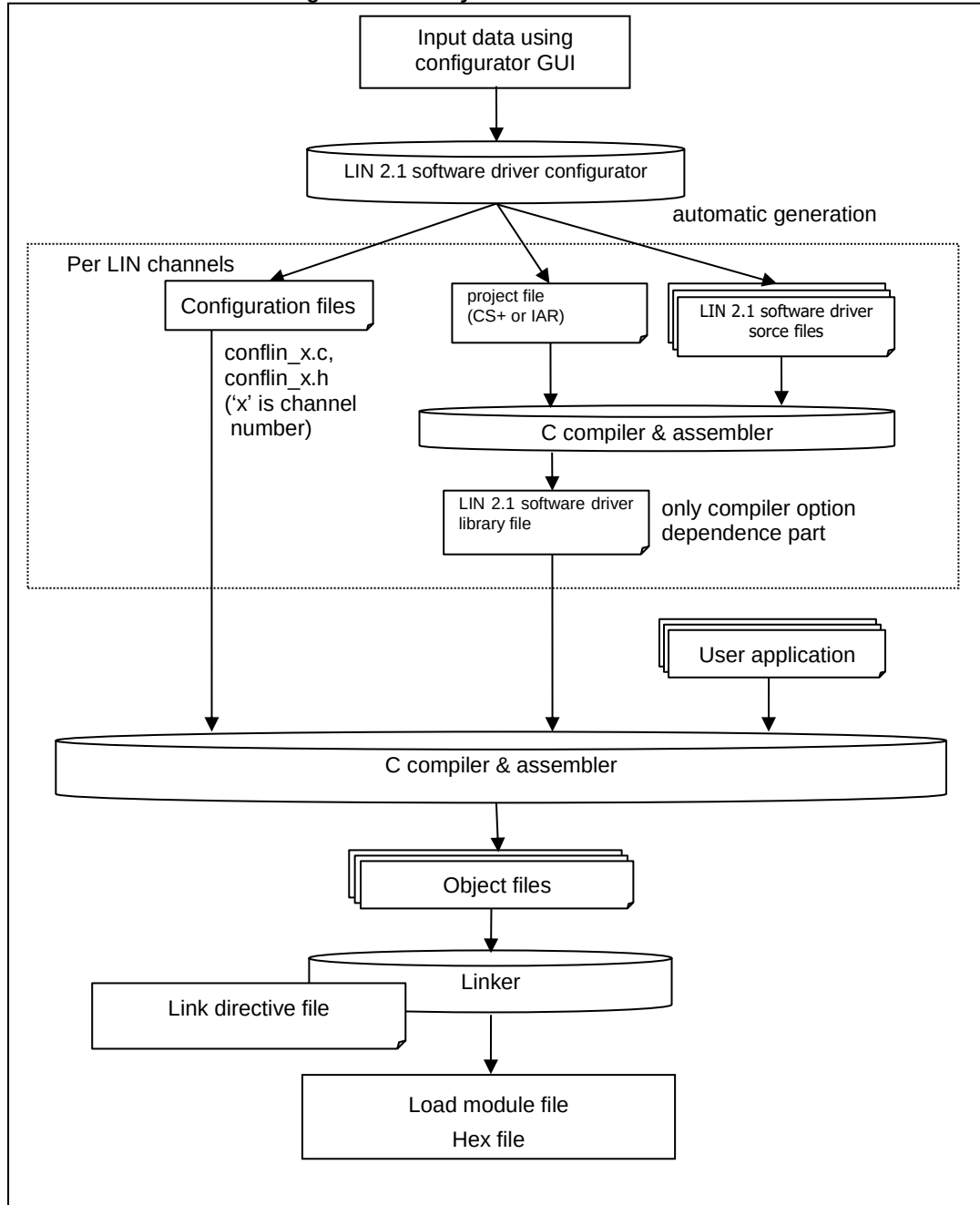
Figure 3-1. System Overview



3.2 Creating the LIN System

This is a creating the LIN system by using the LIN configurator and the LIN 2.1 software driver. Users can also edit the configuration files (conflin.c conflin.h) directly.

Figure 3-2. LIN System Creation Process



Note: The library file of source files depended on the compiler options are included in the LIN 2.1 software driver. For the compiler option of the library, see "5. 1. 1 Compiler options for library".

3. 2. 1 File generation by LIN Configurator

The LIN Configurator accepts various operating parameters such as target device, peripheral hardware clock frequency, LIN baud rate, frame sizes, schedule parameters, and node information, then generates the appropriate function library and configuration files.

For details of the configurator's setting steps, see "CHAPTER 6 LIN CONFIGURATOR".

(1) Required LIN Configurator Settings

The following list of parameters are the minimum requirements needed to be entered into the configurator in order to generate the driver library and configuration files.

- Target Device to be used (Series name and device name)
- Device's peripheral hardware clock frequency
- Channel to be used
- LIN baud rate
- Setting of signal
- Setting of frame
- Setting of schedule
- Node information

(2) LIN Configurator Output

The following files are created by the LIN Configurator.

- Configuration files
- Integrated Development Environment project files
- LIN 2.1 software driver source files (Compiler option dependence part)

(3) Generate library of compiler option dependence part

Generate the library of the compiler option dependence part after generating the file from the configurator.

Using CS+ or IAR project file, the library of the compiler option dependence part can be generated from the LIN 2.1 software driver source file with specifying an arbitrary compiler option.

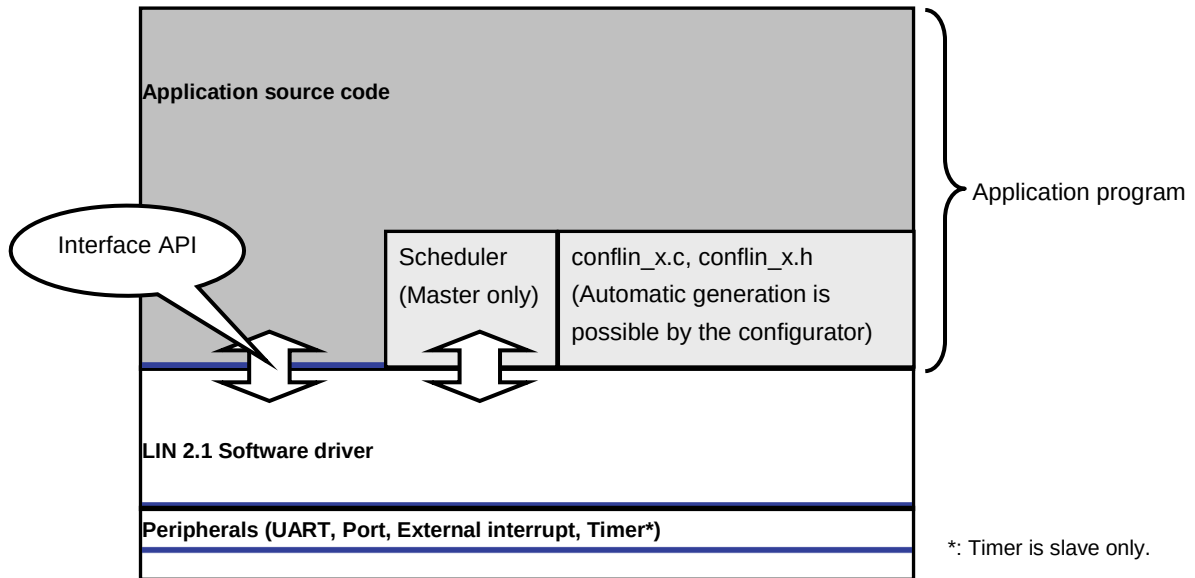
These library files of the compiler option dependence part are needed for creating the LIN system.

3. 2. 2 User applications

LIN 2.1 software driver functions are used to create a complete LIN communication application. The user application program is composed of the application source code, `conflin_x.c`, and `conflin_x.h` ('x' is channel number).

The header file that is created by the configurator must be included in any file that uses a driver function.

Figure 3-3. Relationship between User Application Program and LIN 2.1 Software Driver



See "CHAPTER 5 HOW TO BUILD LIN APPLICATION" for more information.

3. 2. 3 Build

(1) Creation of Object Files

Once all the source files are created (user application and support files created by the LIN configurator), they are compiled and assembled to create re-locatable object files.

Remark See the user's manual of each development tool for details concerning the C compiler/assembler startup options and execution method.

(2) Creation of load module files

The following files are linked to create load module files.

- Object file with compiled/assembled user application
- Library file which compiled the source files generated by LIN configurator.
(The library file which reflected the compiler option)
- Link directive file
- Any other library files recommended by the C compiler package being used.

CHAPTER 4 Linkage to Smart Configurator

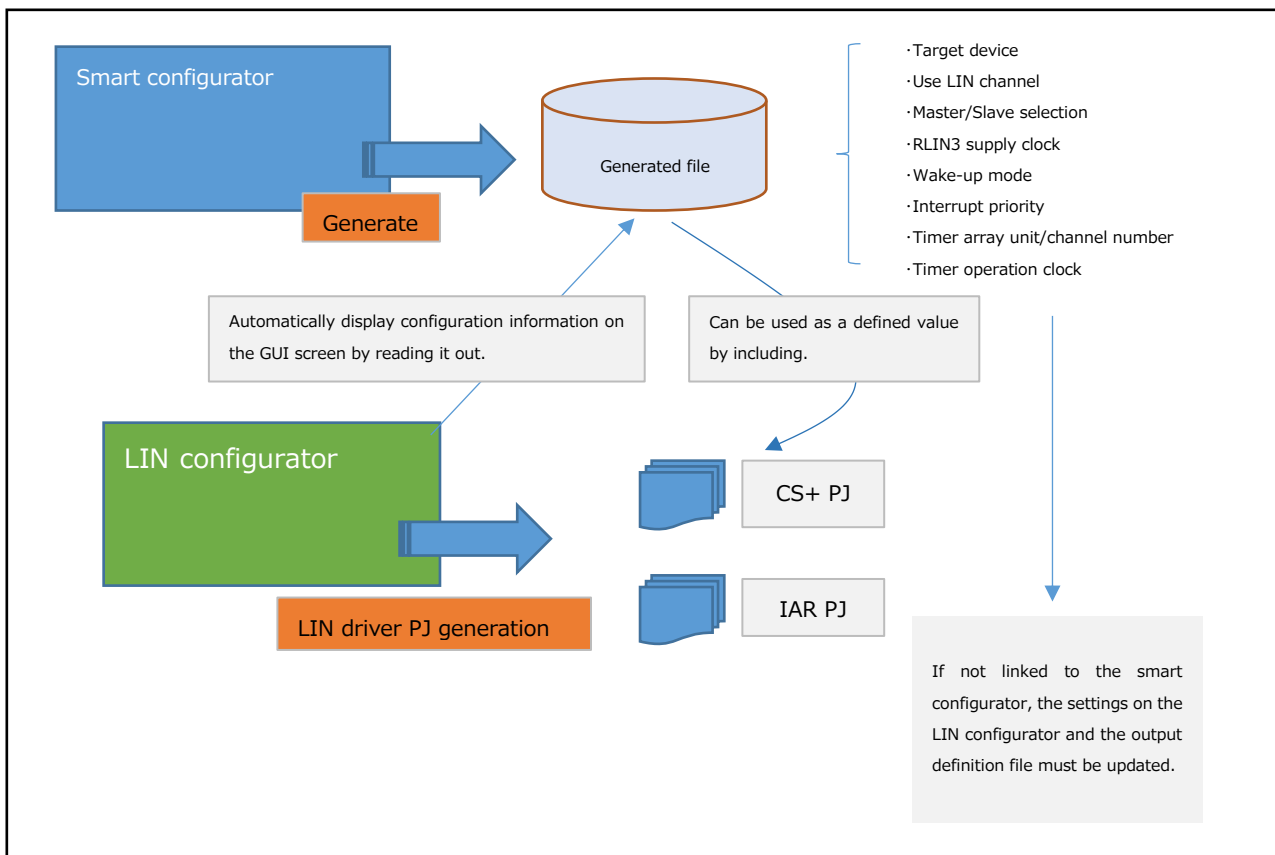
The Smart Configurator is a utility based on the concept of "free combination of software". It facilitates the integration of Renesas drivers into customer systems by importing middleware and drivers, and pin settings.

For specific devices, the LIN configurator supports the ability to read a header file (r_rlin3_config.h) containing LIN-related configuration information generated by the smart configurator.

By reading the relevant file, the LIN-related configuration information set in the smart configurator is automatically reflected on the screen of the LIN configurator.

In addition, by incorporating the relevant file into the LIN 2.1 driver generated from the LIN Configurator, some of the definition values in the conflin_x.h file will be replaced by the definition values set in the Smart Configurator.

Figure 4-1. Image of linkage with smart configurator



4. 1 Operating procedure

The following is a series of processing steps for the linkage.

For more information about operating with Smart Configurator, see the Smart Configurator User's Manual.

- (1) Set LIN-related (RLIN3) parameters on the smart configurator and generate the code.

The files that will be generated are as follows.

- | | |
|--|--|
| - r_rlin3_config.h | : File with LIN configuration value |
| - r_rlin3_m_callout.c, r_rlin3_s_callout | : User-defined callout functions |
| - conflin_x.c , conflin_x.h | : Empty file (File to replace with file generated by LIN Configurator) |

The r_rlin3_config.h file will be shown later.

- (2) Read the r_rlin3_config.h file generated in (1) from the LIN configurator.

Select "SMC Header File Open" from the File menu in the start-up screen of the LIN Configurator.

- (3) If the file is read successfully, the information set in the Smart Configurator will be automatically reflected in the 6. 2. 3 Device selection window and 6. 2. 4 Channel configuration window.

The items to be reflected are as follows.

[6. 2. 3 Device selection window]

Series Information

Device Name

Channel

[6. 2. 4 Channel configuration window]

Master/Slave

Peripheral Hardware Clock

- (4) Configure the other items on the LIN configurator and generate the code.

- (5) Replace the r_rlin3_config.h file contained in the generated folder with the file generated in 1).

This will cause some of the definition values in the confmclin_opt.h file and the confslin_opt.h file to refer to the definition values in the r_rlin3_config.h file.

4. 2 Configuration value

The `r_rlin3_config.h` file consists of definitions for the LIN configurator to read and properly configure the GUI, and definitions for the LIN driver to reference. The channels for RLIN3 and timers should be set from the Smart Configurator within the range of the number of channels provided in the target device.

4. 2. 1 Configuration for LIN Configurator

The following information will be read by the LIN configurator and displayed in the GUI.

Table 4-1. Defined value for LIN Configurator (`r_rlin3_config.h`)

Defining type	Description
RLIN3_CFG_DEVICENAME	Target device
RLIN3_CFG_USE_LIN_CH0 RLIN3_CFG_USE_LIN_CH1	LIN pin setting to be used for each LIN channel 0 and 1 LIN channel 0. 0: unused 1: P13,P14 2: P42,P43 LIN channel 1 0: unused 1: P10, P11 2: P106,P107 3: P120,P125
RLIN3_CFG_CH0_OPERATION_MODE RLIN3_CFG_CH1_OPERATION_MODE	Master/Slave selection for each LIN channel 0 and 1. 0: Master 1: Slave
RLIN3_CFG_CH0_INPUT_CLOCK RLIN3_CFG_CH1_INPUT_CLOCK	Supply clock to LIN channels 0 and 1. (*1) 0: 40 MHz 1: 32 MHz 2: 24 MHz 3: 20 MHz 4: 16 MHz 5: 12 MHz 6: 10 MHz 7: 8 MHz

Remark

- *1. Avoid settings that are inconsistent on the smart configurator.

4. 2. 2 Configuration for LIN driver

The following information is referenced by the LIN driver generated by the LIN configurator.

When building the LIN driver library, set `r_rlin3_config.h` to be included.

Table 4-2. Defined value for LIN driver (`r_rlin3_config.h`)

Defining type	Description
RLIN3_CFG_CH0_BUSWAKEUP RLIN3_CFG_CH1_BUSWAKEUP	LIN channel 0 and 1 wake-up type. 0x00: Falling edge detection type. 0x01: Dominant width detection type.
RLIN3_CFG_INTLIN0TRM_PRIORITY_LEVEL RLIN3_CFG_INTLIN1TRM_PRIORITY_LEVEL	INTLIN0TRM、INTLIN1TRM interrupt priority 0: Level 0 *Highest priority 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_INTLIN0RVC_PRIORITY_LEVEL RLIN3_CFG_INTLIN1RVC_PRIORITY_LEVEL	INTLIN0RVC、INTLIN1RVC interrupt priority 0: Level 0 *Highest priority 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_INTLIN0STA_PRIORITY_LEVEL RLIN3_CFG_INTLIN1STA_PRIORITY_LEVEL	INTLIN0STA、INTLIN1STA interrupt priority 0: Level 0 *Highest priority 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_INTLIN0WUP_PRIORITY_LEVEL RLIN3_CFG_INTLIN1WUP_PRIORITY_LEVEL	INTLIN0WUP、INTLIN1WUP interrupt priority 0: Level 0 *Highest priority 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_CH0_TAU_UNIT RLIN3_CFG_CH1_TAU_UNIT	Timer array unit number used by the LIN Slave driver. 0x00: Unit 0 0x01: Unit 1
RLIN3_CFG_CH0_TAU_CH RLIN3_CFG_CH1_TAU_CH	Channel number of the timer array unit used by the LIN Slave driver. 0x00: channel 0 0x01: channel 1 0x02: channel 2 0x03: channel 3 0x04: channel 4 0x05: channel 5 0x06: channel 6 0x07: channel 7

RLIN3_CFG_CH0_TAU_CLKSEL RLIN3_CFG_CH1_TAU_CLKSEL	The operating clock of the timer array unit used by the LIN Slave driver. 0x0000: Operating clock (CKm0) set by the timer clock selection register (TPSm). 0x8000: Operating clock (CKm1) set by the timer clock selection register (TPSm).
RLIN3_CFG_CH0_TAU_INTTMmn_PRIORITY_LEVEL RLIN3_CFG_CH1_TAU_INTTMmn_PRIORITY_LEVEL	INTTMmn interrupt priority of the timer array unit used by the LIN Slave driver. 0: Level 0 *Highest priority 1: Level 1 2: Level 2 3: Level 3
RLIN3_CFG_USE_SMC_DEFINITION	Some of the definition values in the conflin_x.h file will be the above definition values.

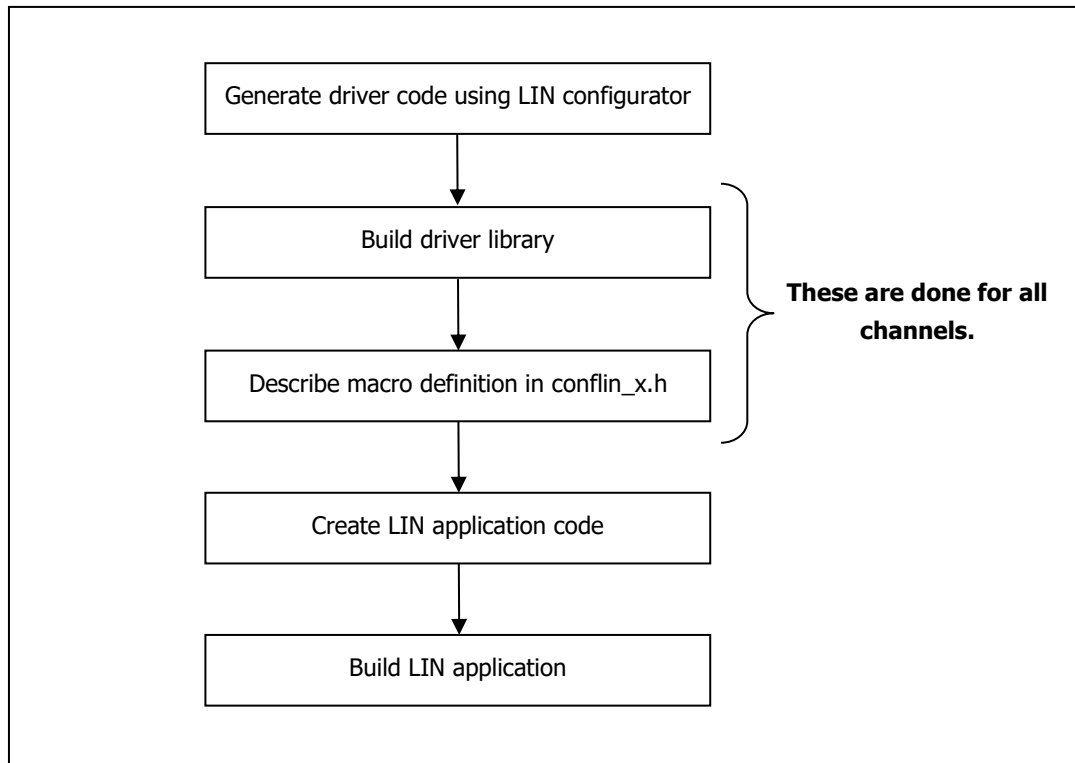
Remark

- Make sure that the LIN-related settings on the Smart Configurator do not conflict with the settings of other functions. The following modules need to be taken care of.
 - Interrupt priority for INTLIN0TRM, INTLIN0RVC, INTLIN0STA, INTLIN0WUP, INTLIN1TRM, INTLIN1RVC, INTLIN1STA, and INTLIN1WUP
 - The used unit/channel number of the timer array unit and the interrupt priority
- Set within the range of the number of units and channels that the product is equipped with.

CHAPTER 5 HOW TO BUILD LIN APPLICATION

In this section, how to build LIN application using this LIN software driver generated by LIN configurator is described.

Figure 5-1. Flow of building LIN application



The flow of building LIN application is shown above.

See "CHAPTER 6 LIN CONFIGURATOR" for more information about "Generate driver code using LIN configurator".

5.1 Build driver library

LIN driver library is built by compiling and linking LIN driver source code generated by LIN configurator. LIN driver library must be built for each channel.

The project file generated by LIN configurator is loaded by development environment and "Build" is started. LIN driver library file is created after these operations.

CS+: (Master) liblin21m_CCRL_x.lib, (Slave) liblin21s_CCRL_x.lib
IAR: (Master) liblin21m_IAR_x.a, (Slave) liblin21s_IAR_x.a
x = 0, 1 (Channel number)

Before building the library, you must make the following edits as needed:

1) Compiler options

The behavior of LIN driver can be changed by specifying compiling option not specified by LIN configurator. See readme.txt generated by LIN configurator and "5. 1. 1 Compiler options for library" for more information.

2) r_rlin3_config.h specification (when linking Smart Configurator)

The LIN configurator outputs an empty r_rlin3_config.h to the folder libsrc/conf/, assuming that it does not work with the Smart Configurator. To link with Smart Configurator, overwrite the empty r_rlin3_config.h with the Smart Configurator output file.

3) Driver configuration

The property of LIN driver can be changed by driver configuration. Change driver configuration (libsrc/conf/confslin_opt.h in slave, libsrc/conf/confmlin_opt.h in master) and build LIN driver if you need. See "5. 1. 2 Edit of confmlin_opt.h (for Master)" and "5. 1. 3 Edit of confslin_opt.h (for Slave)" for details.

4) I/O header file specification (only IAR)

The LIN driver accesses the SFR register and requires an I/O header file with the register address definition. If the development environment used is CS+, it is automatically resolved at build time, but in the case of IAR, editing is required separately. See "5. 1. 4 Specify the IAR I/O header file" for details.

5. 1. 1 Compiler options for library

Compiler options for building RL78/F23, F24 driver library are shown below.

"Mandatory setting" or "Normally setting" option are predefined in the development environment project file generated by LIN configurator.

__LIN_CH0_P1__ or __LIN_CH1_P1__ is defined as the initial setting of compiler option for channel. If you want to use port group other than port 1 when it does not work with Smart Configurator, edit the project file and change the option. If you use Smart Configurator, no editing is required since the options are automatically replaced according to the specified port.

If you need "Selectable" option or do not use "Normally setting" option, edit the project file. For the editing method, refer to the user's manual of the integrated development environment used.

If you use macro __LIN_SIGNAL_32__, you need to add it to the conflin_x.h file separately. See "5. 2. 4 Describe compiler options (conflin_x.h)" for details.

Table 5-1. Compiler Option for Library Usage List

Compiler option	Macro Name	Master	Slave
Series name	__LIN_RL78_F23_F24__	⊙	⊙
Channel	*1	⊙	⊙
Memory copy routine by assembler	__LIN_MEMCOPY_ASM__	△ *2	△ *2
Enable auto-baudrate-mode by hardware	__LIN_HW_AUTO_BR__	Not applicable	○*3 *4
The number of the signals in the frame : 32	__LIN_SIGNAL_32__	△	△

⊙ : Mandatory setting. ○ : Normally setting △ : Selectable

*1 __LIN_CH0_P1__, __LIN_CH0_P4__, __LIN_CH1_P1__, __LIN_CH1_P10__, or __LIN_CH1_P12__
Change the compiler option according to the port group you are using if not use Smart Configurator.

(1) If choice the channel 0

P1 : __LIN_CH0_P1__ (Rx port is P14. Tx port is P13)

P4 : __LIN_CH0_P4__ (Rx port is P43. Tx port is P42)

(2) If choice the channel 1

P1 : __LIN_CH1_P1__ (Rx port is P11. Tx port is P10)

P10 : __LIN_CH1_P10__ (Rx port is P107. Tx port is P106)

P12 : __LIN_CH1_P12__ (Rx port is P125. Tx port is P120)

*2 In IAR compiler, this option is available when memory model is near model.
The CC-RL compiler can be used with either the small or medium model.

You should also set the define in the Assembler Options to the option "__LIN_CHn_Pn__" and "__LIN_MEMCOPY_ASM__".

*3 If __LIN_HW_AUTO_BR__ is used, set number of bit samplings to 4 or 8.

If __LIN_HW_AUTO_BR__ is not used, set number of bit samplings to 16,

See "5. 1. 3 Edit of conflin_opt.h (for Slave)" and "7. 9. 1 Slave driver configuration" for more information.

*4 It must be configured to match the baud rate setting by the LIN configurator.

See "6. 2. 5 Baud rate configuration" for details.

5. 1. 2 Edit of confmlin_opt.h (for Master)

Typical configuration items for the RL78/F23, F24 master drivers are listed below. These cannot be configured in the LIN Configurator and Smart Configurator. Therefore, if you want to change from the default settings, the confmlin_opt.h file (storage folder libsrc/conf) must be edited directly.

See "7. 9. 2 Master driver configuration" for configuration details and other configuration items.

- | | |
|------------------------------|--|
| - CONFMLIN_OPT_u1gLINMCK_CFG | Source of LIN communication clock (f_{CLK} , f_{MX})
(Initial value: f_{CLK}) |
| - CONFMLIN_OPT_u1gBDT_CFG | Sending break delimiter length (Initial value: 2Tbit) |
| - CONFMLIN_OPT_u1gIBHS_CFG | Width between sending synch field and id field, and width
of sending response space
(Initial value: 0Tbit) |
| - CONFMLIN_OPT_u1gIBS_CFG | Width between each sending response
(Initial value: 0Tbit) |

When using Smart Configurator, the following items are set according to the definition in the output header file r_rlin3_config.h. If not using, the items will be the initial value, so edit confmlin_opt.h directly if necessary.

- | | |
|--|---|
| - CONFMLIN_OPT_u1gBUSWKUP_CFG | Bus wakeup method (Initial value: Down edge detection) |
| - CONFMLIN_OPT_u1gINTXXXPR_CFG | |
| (INTXXX: 4 types of INTLINTRM, INTLINRVC, INTLINSTA, INTP) | Priority of each interrupt
(Initial value: Level 3 (lowest)) |

5. 1. 3 Edit of confslin_opt.h (for Slave)

Typical configuration items for the RL78/F23, F24 slave drivers are listed below. These cannot be configured in the LIN Configurator and Smart Configurator. Therefore, if you want to change from the default settings, the confslin_opt.h file (storage folder libsrc/conf) must be edited directly.

If you use the fixed baud rate by removing the compilation option `__LIN_HW_AUTO_BR__`, change the number of bit sampling `CONFSLIN_OPT_u1gNSPB_NORM_CFG` to 16.

See "7. 9. 1 Slave driver configuration" for configuration details and other configuration items.

- CONFSLIN_OPT_u1gLINMCK_CFG	Source of LIN communication clock (f_{CLK} , f_{MX}) (Initial value: f_{CLK})
- CONFSLIN_OPT_u1gLPRS_NORM_CFG	Prescaler division value for LIN macro (Initial value: 1/1)
- CONFSLIN_OPT_u1gNSPB_NORM_CFG	Number of bit samplings (Initial value: 4)
- CONFSLIN_OPT_u1gRS_CFG	Response space width at response Tx (Initial value: 1Tbit)
- CONFSLIN_OPT_u1gIBS_CFG	Inter byte space width at response Tx (Initial value: 0Tbit)

When using Smart Configurator, the following items are set according to the definition in the output header file `r_rlin3_config.h`. If not using, the items will be the initial value, so edit `confslin_opt.h` directly if necessary.

- CONFSLIN_OPT_u1gBUSWKUP_CFG	Bus wakeup method (Initial value: Down edge detection)
- CONFSLIN_OPT_u1gINTXXXPR_CFG	Priority of each interrupt (INTXXX: 4 types of INTLINTRM, INTLINRVC, INTLINSTA, INTP) (Initial value: Level 0 (highest))
- CONFSLIN_OPT_u1gTMUNIT_CFG etc.	4 settings for interval timer to be used (Initial value: TAU0, Channel 0, CK01 use, Priority of timer interrupt is Level 0 (highest))

5. 1. 4 Specify the IAR I/O header file

LIN 2.1 software driver not only CC-RL compiler but also IAR compiler.

When using the project file for the IAR compiler output from the configurator, the header file with address definitions and unique function declarations for the SFR registers, extended SFR registers, interrupt vectors, etc. provided by IAR must be resolved to include.

In the case of the R7F124FPJ, the file is as follows.

- ior7f124fpj.h
- ior7f124fpj_ext.h
- intrinsics.h

These files are included in the IAR environment installation folder. It is also possible to use Smart Configurator r_bsp output files. See the r_bsp application note for details.

The same header file must be placed in the appropriate folder in the user application project folder in the user application that incorporates the LIN 2.1 software driver.

Solve the include device.h (storage folder libsrc/dev) by doing one of the following:

- a) Edit device.h and match the header file name specified by "#include" to the device.

At this time, it is described including the path "../liblin2/".

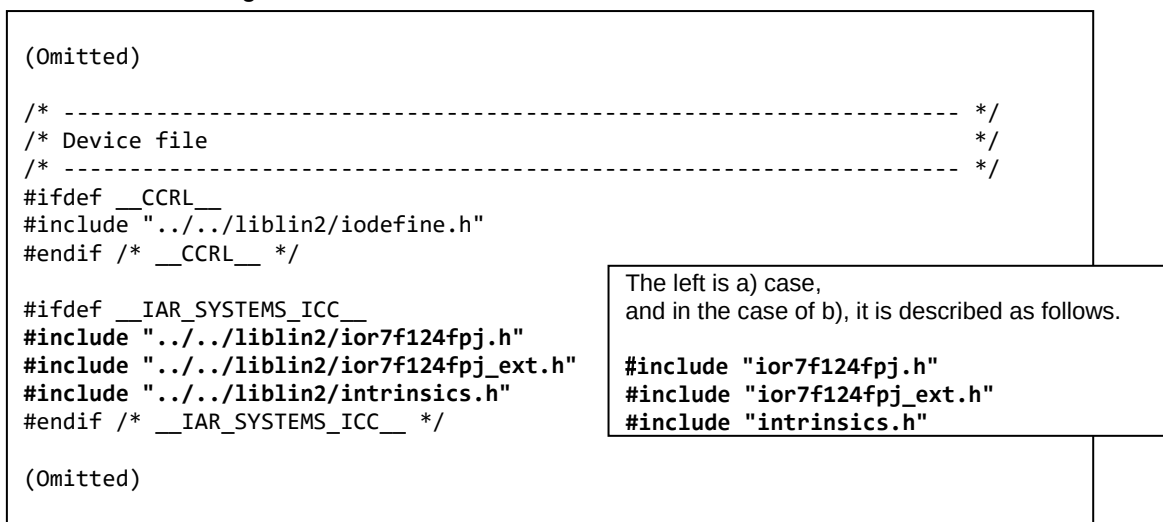
Copy the specified file to the liblin2 folder.

- b) Edit device.h and match the header file name specified by "#include" to the device.

Delete the path "../liblin2".

Uncheck "Ignore standard include directories" in the project file.

Figure 5-2. How to describe the include in device.h



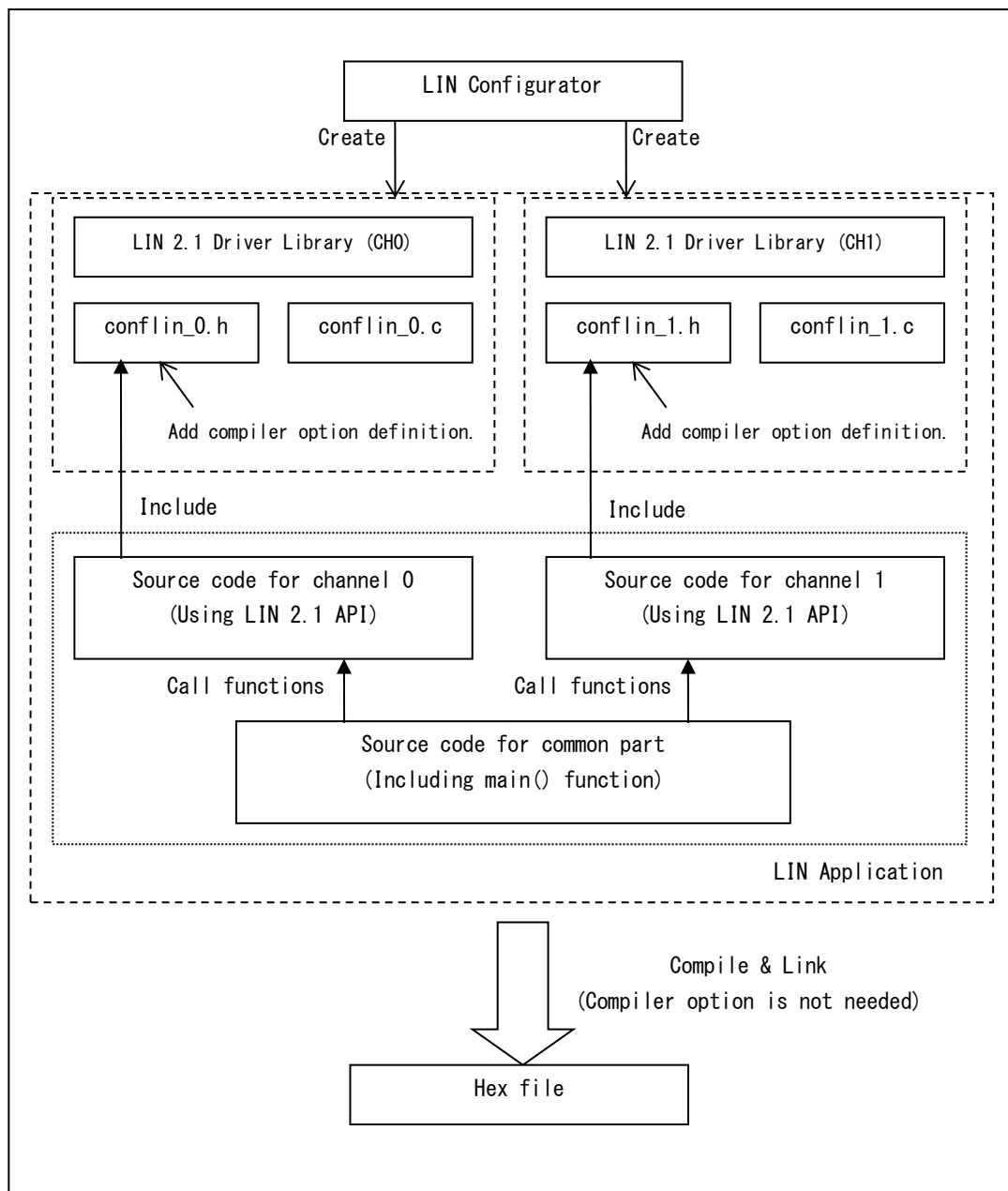
5.2 Create LIN application code

Create LIN application program using LIN 2.1 API provided by LIN 2.1 software driver.

How to build LIN application for using channel 0 and channel 1 is shown below.

Because one source code (*.c file) cannot include multiple `conflin_x.h`, the channel-only source code must be isolated when using multiple channels. If the channel used is one channel, channel-only source code is not required.

Figure 5-3. How to build LIN application



5. 2. 1 Creation of development environment project file

When creating a development environment project file for a LIN application program, add the following files for build to use the LIN driver.

- Configuration files generated by LIN Configurator: `conflin_x.c` , `conflin_x.h`
- Library files created by building LIN driver sources (See "5. 1 Build driver library")
`liblin21m_CCRL_0.lib`、`liblin21m_IAR_0.a`, etc.
- In case of IAR, link configuration file (extension `.icf`)

When linking with Smart Configurator, `conflin_x.c` , `conflin_x.h` are generated in an empty file, so overwrite it with the LIN configurator output file.

Because LIN drivers use unique sections, in the IAR environment, it is necessary to add the project's own link configuration file and write the section. In the CS+ environment, the user edits the properties of the project. See "5. 2. 8 Use Section Setting" for more information on the specified section.

Note that when using the compiler option "`__LIN_MEMCOPY_ASM__`", do not place in the far area and huge area.

5. 2. 2 Craation of channel-only source code

The function interface of the LIN2.1 software driver has a commonized function name between the master and slave, and the function name is also commonized for the channel. On the other hand, the functions of the driver library are implemented separately in master/slave and channel numbers.

For this reason, header file `conflin_x.h` must be included in order to link the commonized function interface to the function name of the driver library.

ex. Function name `I_u16_rd`

Include `conflin_0.h` (for Master) -> Replace to `ApMLin_u2gRead16bitsSig_0`

Include `conflin_1.h` (for Slave) -> Replace to `ApSLin_u2gRead16bitsSig_1`

One source code (`.c` file) can include only one `conflin_x.h`.

Therefore, if LIN application for multi channels is created, source code must be created for each channel and each source code must include `conflin_x.h`.

Use LIN 2.1 API for each channel in the channel source code.

5. 2. 3 Peripheral hardware processing implementation

The LIN driver operates on the peripheral hardware shown in "1. 4. 3 Hardware resources". Because no other peripheral hardware is configured, user code must handle the peripheral hardware when creating LIN applications.

The main peripheral hardware processing is as follows.

1) Configure clock oscillator circuit.

Clock oscillator function is not set by LIN 2.1 software driver. Application must configure this setting.

When linking to the output source of the Smart Configurator, no user code is required.

2) Configure LIN transceiver

LIN transceiver is not operated by LIN 2.1 software driver. Application must configure this setting.

When linking to the output source of Smart Configurator, the output pin can be initially set by using the port function. A user code is required to switch pin levels.

3) Standby function.

The transition to microcomputer standby mode (HALT, STOP and SNOOZE) isn't controlled by LIN 2.1 software driver. If it is needed, application must control it.

In RL78/F23,F24 series, microcomputer can transit to standby mode when the bus wakeup way is "Falling edge detection" only (This is controlled by driver configuration.). In master, microcomputer can transit to standby mode if the return value of `I_ifc_read_status()` after calling `I_ifc_goto_sleep()` is "0x3C02" (bit3-7 are indeterminate values). In slave, microcomputer must can transit to standby mode when `I_sys_call_sleep()` is called.

The notes about peripheral hardware when creating LIN applications are as follows.

- Note for using peripheral hardware.

Hardware resources used by LIN 2.1 software driver can not be used by application. These hardware resources are described in 1. 4. 3 .

- Output latch of LIN port.

LIN 2.1 software driver operates output latch of LIN TXD pin on RL78. So, if other pins in same register as TXD pin is input direction, these pin levels may be changed by LIN driver operation.

- Interrupt priority setting.

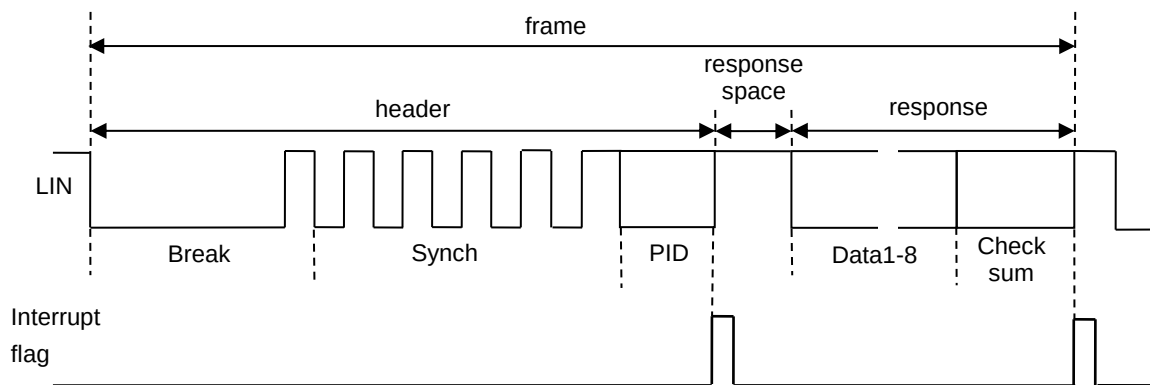
In the driver for RL78/F23 and F24, the interrupt priority is set by the driver configuration or Smart Configurator. If application uses interrupt, interrupt collision with used by LIN 2.1 software driver should be taken care.

Setting interrupt priority used by LIN slave driver to higher (used by application is set to lower) is recommended.

The LIN master driver does not use interrupts except for the driver sleep or wake up. This eliminates the need for higher priority settings.

RL78/F23, and F24 masters do not use interrupts except for driver sleep and wake-up.

Figure 5-4 Timing of interrupt and LIN frame in RL78 series.



5. 2. 4 Describe compiler options (conflin_x.h)

Describe macro definition on top of conf/conflin_x.h ('x' is channel number) generated by LIN configurator. This must be done for each channel.

The compiler options required for LIN applications are as follows. Of these, the LIN configurator automatically specifies the target microcomputer and channel. If `__LIN_SIGNAL_32__` is added when building the LIN driver library, `conflin_x.h` must be edited directly. (See "5. 1. 1 Compiler options for library".)

表 5-1 Compiler options (conflin_x.h) list

Compiler options	Macro name	RL78/F23,F24 Master	RL78/F23,F24 Slave
Series name	<code>__LIN_RL78_F23_F24__</code>	LIN configurator automatically specifies (cannot be deleted)	
Channel	<code>__LIN_CH0__</code> or <code>__LIN_CH1__</code>	LIN configurator automatically specifies (cannot be deleted) <code>conflin_0.h: __LIN_CH0__</code> <code>conflin_1.h: __LIN_CH1__</code>	
The number of the signals in the frame : 32	<code>__LIN_SIGNAL_32__</code>	Need to be added if <code>__LIN_SIGNAL_32__</code> is specified to build LIN driver library.	

The location to describe macro definition is shown below (bold part).

Figure 5-5. Location of macro description in conflin_x.h

```
(omitted)

#ifndef H_CONFLIN
#define H_CONFLIN

/* ***** */
/* [Add macro definitions for LIN application] (from here) */
/* ***** */
/* [Example] */
/* #define __LIN_CH0__ */
/* ***** */
/* [Note] */
/* Definition contents are different by each devices. */
/* For more detail, see "compile option" in readme.txt . */
/* ***** */

#define __LIN_RL78_F23_F24__
#define __LIN_CH0__
#define __LIN_SIGNAL_32__ /* example */

/* ***** */
/* [Add macro definitions for LIN application] (to here) */
/* ***** */

(omitted)
```

Contents of macro definitions are different depending on each device. The "compile option" for application building in `readme.txt` generated by LIN configuration is equal to these contents.

5.2.5 Edit public constants (conflin_x.c)

The conf/conflin_x.c ("x" is channel number) generated by the LIN Configurator contains public constants with const variables. Normally, the conflin_x.c file does not need to be edited, but it needs to be edited in the following cases.

- a) When using a LIN communication clock other than the frequency that can be specified by the LIN configurator (using other than 8, 10, 12, 16, 20, 24, 32, 40 [MHz])

When using a LIN communication clock frequency that cannot be specified by the LIN configurator, specify the frequency (resolution 10 kHz) as defined value of the CONFLIN_u2sPERICLOCK as follows.

ex. 4MHz for LIN communication clock

[Before modification] LIN communication clock 40 MHz = 10 kHz x 4000

```
#define CONFLIN_u2sPERICLOCK    (4000)
```

[After modification] LIN communication clock 4 MHz = 10 kHz x 400

```
#define CONFLIN_u2sPERICLOCK    (400)
```

- b) If the following conditions are met:

- Use slave driver and
- Use f_{MX} instead of f_{CLK} as LIN supply clock and
- Different frequency for f_{CLK} and f_{MX}

The initial setting of the LIN slave driver assumes that the supply clocks for LIN and interval timer (TAU) are both f_{CLK} and the same frequency is input.

Therefore, When using f_{MX} for LIN communication clock provides LIN with a different frequency than the TAU, it is necessary to modify the constant ConfSLin_u2gTMPERICLOCK indicating the TAU supply clock frequency (resolution 10 kHz) as follows after code generation with the LIN configurator.

ex. 8MHz for LIN communication clock, 40 MHz for TAU supply clock

[Before modification] Both LIN communication clock and TAU supply clock 8 MHz = 10 kHz x 800

```
#define CONFLIN_u2sPERICLOCK    (800)
```

```
const u2 ConfSLin_u2gPERICLOCK   = (u2)CONFLIN_u2sPERICLOCK;
```

```
const u2 ConfSLin_u2gTMPERICLOCK = (u2)CONFLIN_u2sPERICLOCK;
```

[After modification] LIN communication clock 8 MHz, TAU supply clock 40 MHz = 10 kHz x 4000

```
#define CONFLIN_u2sPERICLOCK    (800)
```

```
const u2 ConfSLin_u2gPERICLOCK   = (u2)CONFLIN_u2sPERICLOCK;
```

```
const u2 ConfSLin_u2gTMPERICLOCK = (u2)4000;
```

5. 2. 6 Scheduler implementation (only master)

When creating the LIN master application, it is necessary to implement a scheduler using a timer in the user master application.

See “7. 7 Scheduling Function (Master)” for more details regarding the implementation of a schedule table.

The program example of a scheduler setting using the timer function of the Smart Configurator is shown as follows.

- Component: Interval timer
- Configuration name: Config_TAU0_1
- Operation mode: 16 bits counter mode
- Resource: TAU0_1
- Output files: Config_TAU0_1.c, Config_TAU0_1.h, Config_TAU0_1_user.c

[Config_TAU0_1_user.c] extract

- Add conflin_0.h as the include target

```

/*****
Includes
*****/
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_TAU0_1.h"
/* Start user code for include. Do not edit comment generated here */
#include "conflin_0.h"
/* End user code. Do not edit comment generated here */

```

- Add I_sch_tick in interrupt routine

[CC-RL]

```

/*****
* Function Name: r_Config_TAU0_1_interrupt
* Description : This function is INTTM01 interrupt service routine.
* Arguments : None
* Return Value : None
*****/
static void __near r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment generated here */
    (void)I_sch_tick(LIN_CHANNEL0);
    /* End user code. Do not edit comment generated here */
}

```

[IAR]

```

#pragma vector = INTTM01_vect
__interrupt static void r_Config_TAU0_1_interrupt(void)
{
    /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment generated here */
    (void)I_sch_tick(LIN_CHANNEL0);
    /* End user code. Do not edit comment generated here */
}

```

[main.c]

- Add timer function header file Config_TAU0_1.h as the include target Include
- Add conflin_0.h as the include target
- Call the timer start function R_Config_TAU0_1_Start after LIN initialize
- Set schedule table by the l_sch_set function

```
#include "platform.h"
#include "Config_TAU0_1.h"

/* defined in configuration file.
   - Channel name   : LIN_CHANNEL0
   - Schedule name  : SCH_MAINAPL
*/
#include "conflin_0.h"

/* main function */
void main( void )
{
    /* LIN initialize or more */

    BSP_ENABLE_INTERRUPT();

    R_Config_TAU0_1_Start();           /* Start timer           */

    l_sch_set(LIN_CHANNEL0, SCH_MAINAPL, 0U); /* Set schedule of main app. */
    ApLst_vosMainApl();               /* Start main application */
}
```

5. 2. 7 User-defined callouts implementation

The LIN2.1 software driver calls to call-out functions for interrupt disable, interrupt restore, and wakeup notification, etc. Each callout function must be implemented in user code in order to link with the LIN driver library.

If the code is output by using the Smart Configurator, the following source file will be output as the required callout functions.

Source code for master driver: r_rlin3_m_callout.c

Source code for slave driver: r_rlin3_s_callout.c

See “8. 3. 8 [Slave] User provided call-outs” and “9. 3. 10 [Master] User provided call-outs” for details.

5.2.8 Use Section Setting

The LIN2.1 software driver places the code area and data area to be used in the unique section. Therefore, you need to define a section to link. Therefore, the sections must be defined in order to link.

In the case of CS+, define it in the project file. In the case of IAR, add the target section to the link configuration file (extension .icf) and link the file. For details on how to edit, refer to the user's manual of the integrated development environment to use.

(1) Use section for CC-RL

Define the following sections in the CS+ project file.

Master driver

- ROM area: LMCODE_n, LMCODE_f, LMCNST_n, LMCNST_f
- RAM area: LMDATA_n, LMDATA_f

Slave driver

- ROM area: LSCODE_n, LSCODE_f, LSCNST_n, LSCNST_f
- RAM area: LSDATA_n, LSDATA_f

Table 5-2. LIN2.1 software driver use sections (CC-RL)

Default Section Name	Description	LIN Driver Usage Section Name		Supplement
		Master	Slave	
.callt0	Section for the table to call the callt function	(unuse)	(unuse)	
.text	Section for code (allocated to the near area)	LMCODE_n	LSCODE_n	Function, Interrupt function
.textf	Section for code (allocated to the far area)	LMCODE_f	LSCODE_f	
.textf_unit64kp	Section for code (section is allocated so that the start address is an even address and the section does not exceed the (64 Kbytes - 1) boundary)	(unuse)	(unuse)	
.const	ROM data (allocated to the near area) (within the mirror area)	LMCNST_n	LSCNST_n	Constant data (Including global and static)
.constf	ROM data (allocated to the far area)	LMCNST_f	LSCNST_f	
.data	Section for near initialized data (with initial value)	(unuse)	(unuse)	
.dataf	Section for far initialized data (with initial value)	(unuse)	(unuse)	
.sdata	Section for initialized data (with initial value, variable allocated to saddr)	(unuse)	(unuse)	
.bss	Section for data area (without initial value, allocated to the near area)	LMDATA_n	LSDATA_n	Variable data without initial value (Including global and static)
.bssf	Section for data area (without initial value, allocated to the far area)	LMDATA_f	LSDATA_f	
.sbss	Section for data area (without initial value, variable allocated to saddr)	(unuse)	(unuse)	
.option_byte	Section specific for user option byte and on-chip debugging specification	(unuse)	(unuse)	
.security_id	Section specific for security ID specification	(unuse)	(unuse)	
.vect<vector table address>	Interrupt vector table	-	-	

NOTE:

When using the compiler option "`__LIN_MEMCOPY_ASM__`", do not place it in the far area.

(2) **User section for IAR**

Define the following sections in the IAR link configuration file (extension .icf).

Master driver

- ROM area: LMCODE, LMCNST
- RAM area: LMDATA

Slave driver

- ROM area: LSCODE, LSCNST
- RAM area: LSADATA

Table 5-3. LIN2.1 software driver use sections (IAR)

Description	Default Section Name	LIN Driver Usage Section Name		Supplement
		Master	Slave	
Functions, interrupts and <code>_callt</code> functions	<code>.text</code>	LMCODE	LSCODE	for the near area
	<code>.textf</code>	LMCODE	LSCODE	for the far area
Constant data	<code>.const</code>	LMCNST	LSCNST	for the near area
	<code>.constf</code>	LMCNST	LSCNST	for the far area
	<code>.consth</code>	LMCNST	LSCNST	for the huge area
Interrupt vector table	<code>.intvec</code>	–	–	–
Static and global initialized variables	<code>.data</code>	(unuse)	(unuse)	for the near area
	<code>.dataf</code>	(unuse)	(unuse)	for the far area
	<code>.hdata</code>	(unuse)	(unuse)	for the huge area
Zero-initialized static and global variables	<code>.bss</code>	LMDATA	LSADATA	for the near area
	<code>.bssf</code>	LMDATA	LSADATA	for the far area
	<code>.hbss</code>	LMDATA	LSADATA	for the huge area
Initial values for <code>.data</code> section	<code>.data_init</code>	(unuse)	(unuse)	for the near area
	<code>.dataf_init</code>	(unuse)	(unuse)	for the far area
	<code>.hdata_init</code>	(unuse)	(unuse)	for the huge area
Call table vectors generated by use of the <code>_callt</code>	<code>.callt0</code>	(unuse)	(unuse)	–
OCD option bytes	<code>.option_byte</code>	(unuse)	(unuse)	–
Security ID	<code>.security_id</code>	(unuse)	(unuse)	–

NOTE:

When using the compiler option "`__LIN_MEMCOPY_ASM__`", do not place it in the far area and huge area.

CHAPTER 6 LIN CONFIGURATOR

6.1 General

The LIN Configurator is a development tool that the user can use to select the LIN's initial values when building a system that includes LIN functions. Functions are provided to enable the user to set initial values for registers in accordance with the device to be used, and to perform static generation of messages used in the system.

6.1.1 Features

The initial values for registers corresponding to the target device can be set while selecting the device, such as entering the peripheral hardware clock value, and setting the baud rate per channel. Necessary setting for LIN communications between the setting of various frames such as unconditional frames and the event triggered frames and schedules and the settings of node information, etc. can be done at a time.

In the LIN configurator software, any LIN system configuration information that is entered by the user is saved to a project file. This enables the user to preserve any work done on a project and also recall it when needed.

6.1.2 Execution Environment

The environment needed for this tool is as follows.

- Host machine : IBM PC/AT™ Series
- OS : Windows 10
(Microsoft .NET Framework 3.5 is needed)

6.1.3 Output Folder

The folders generated from LIN configurator are as follows.

- /conf : Location where configuration files are saved.
- /liblin2 : Location where the project files of development environment are saved. It is used to generate the library from the generated source files by LIN configurator.
- /libsrc : Location for the LIN 2.1 software driver source code.

6.2 File Generation Steps

If the target device setting on the display is completed, it is possible to set it in random order. In general, it is recommended to set it in order of 4.2.1-4.2.13.

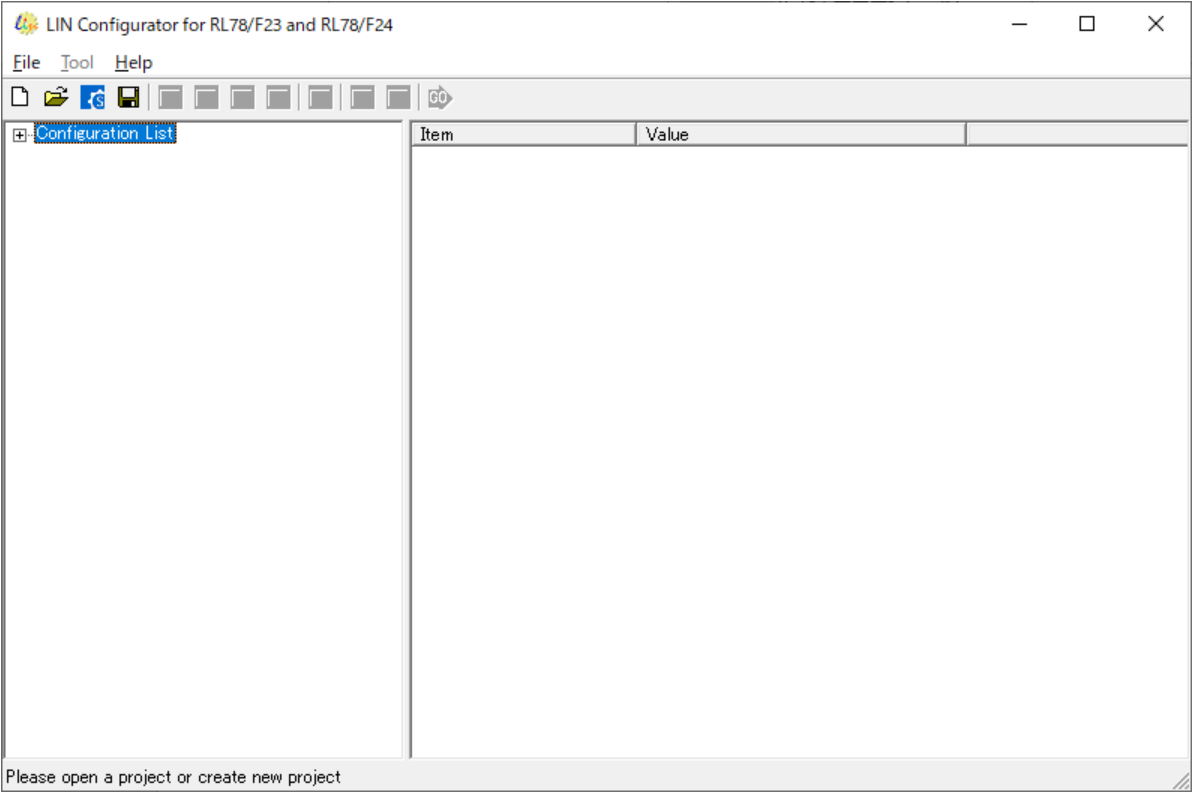
However, when input the set value, note the following points.

- Names (Frame name, Signal name, Schedule table name)
These names must be unique; they can not be the same name. It is highly recommended to specify the name according to the identifier pattern (naming convention) of the C language.
- Numerical values
All values should be defined by their decimal value.
The Message ID/Initial ID, Initial data, NAD, Supplier ID, Functional ID, and Variant may be described by their hexadecimal value.
- XML files
Do not change the XML file of the configurator attachment. There is a possibility of not operating normally.

6. 2. 1 Start Up of LIN Configurator

Main Screen shown below is opened when “LIN Configurator” in start menu is selected. Main contents are shown in left side and right side is value of contents. Contents not be set is not displayed.
(Following all screens is on Windows 10.)

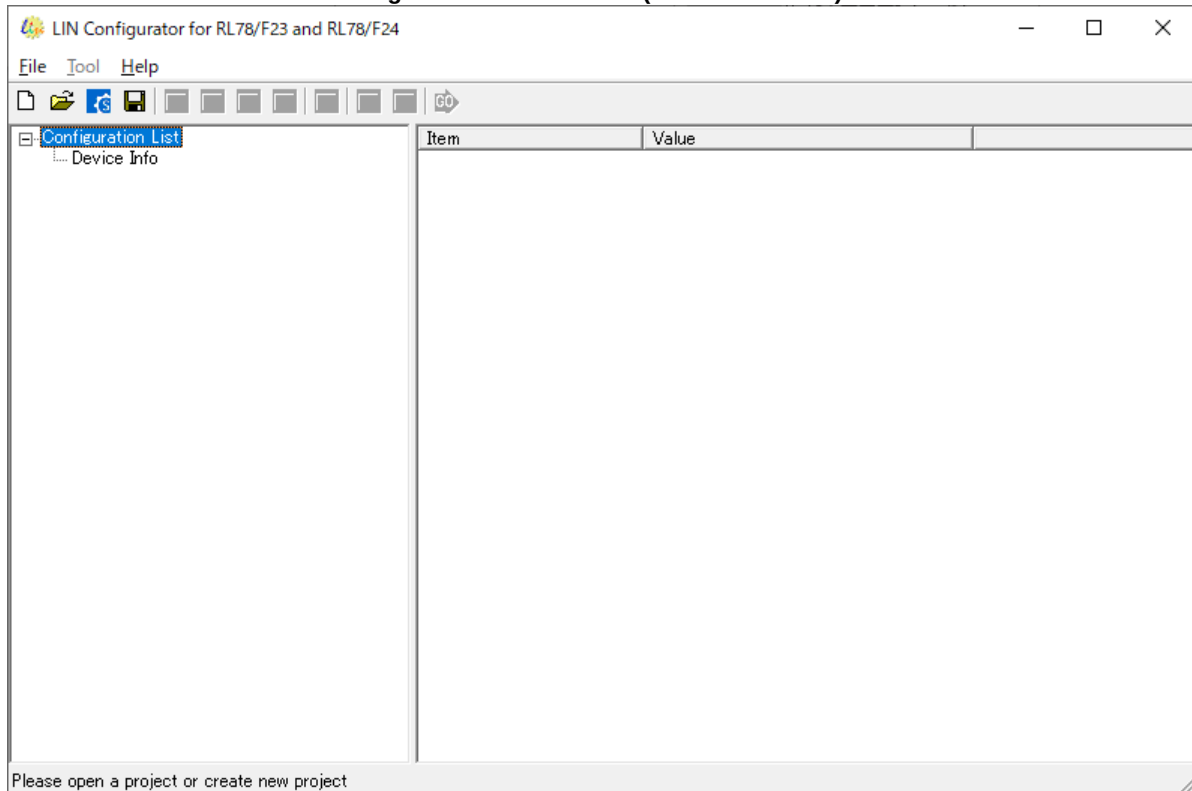
Figure 6-1. Main Screen



Contents of device and channels are hanged on “Configuration List”.

Main Screen transits to below when the '+' left of "Configuration List" is clicked.

Figure 6-2. Main Screen (before selection)



Following contents are selectable in upper part of Main Screen.

- "File"
 - "New" : Start new configuration after discarding current settings.
 - "Open" : Load setting XML file.
 - "SMC Header File Open"
 - : Reads the LIN-related definition file (header file) output from the smart configurator. (It can also be executed with the  button.)
 - "Save" : Overwrite settings to XML file saved last time.
 - "Save As" : Save settings to new XML file.
 - "Exit" : Finish LIN Configurator.
- "Tool" : See table in next page for operations from "Tool" menu.
- "Help"
 - "About Configurator" : Information of LIN Configurator is shown in dialog..

Remark

By executing "SMC Header File Open", all the information that has been set up until then will be cleared.

Selectable of buttons on upper part depends on the channel type (Master or Slave) currently selected.
See table in next page.

Table 6-1. Operation from tree contents, buttons and Tool menu (for Master channel)

Operation			Opened window
Tree contents (double clicked)	Button	Tool menu selection	
"Device Info"		Device Setup	Device Selection
"Channel*"		Channel Setup ^{*1}	Channel Configuration
"Baud rate"		Baud Rate Setup ^{*1}	Setting Baud Rate
"Unconditional frames"		Message Setup ^{*1}	Setting Frame
A frame in "Unconditional frames"			
A signal in "Unconditional frames"			
Event triggered frames			
A frame in "Event triggered frames"			
Sporadic frames			
A frame in "Sporadic frames"			
"Schedules"		Schedule Setup ^{*1}	Setting Schedule
A schedule in "Schedules"			

*1 Channel has to be selected to select this menu and button.

Table 6-2. Operation from tree contents, buttons and Tool menu (for Slave channel)

Operation			Opened window
Tree contents (double clicked)	Button	Tool menu selection	
"Device Info"		Device Setup	Device Selection
"Channel*"		Channel Setup ^{*2}	Channel Configuration
"Baud rate"		Baud Rate Setup ^{*2}	Setting Baud Rate
"Unconditional frames"		Message Setup ^{*2}	Setting Frame
A frame in "Unconditional frames"			
A signal in "Unconditional frames"			
"Event triggered frames"			
A frame in "Event triggered frames"			
Node information		Node Setup ^{*2}	Setting Node
Others		Others Setup ^{*2}	Setting Others

*2 Channel has to be selected to select this menu and button.

6. 2. 2 Start a New Configuration

There are two ways to start a new configuration.

a) When starting with a configuration of Smart Configurator

If "SMC Header File Open" of File menu is selected, the file loading dialog is displayed.

Specify the LIN-related definition file `r_rlin3_config.h` generated by the Smart Configurator.

A new configuration is created with the information set in the file as the initial state.

It can also be executed with the  button.

If the file is read successfully, the information set in the Smart Configurator will be automatically reflected on the 6. 2. 3 Device selection window and the 6. 2. 4 Channel configuration window.

The items to be reflected are as follows.

[6. 2. 3 Device selection window]

Series Information

Device Name

Channel

[6. 2. 4 Channel configuration window]

Master/Slave

Peripheral Hardware Clock

b) When creating a new configuration by LIN Configurator

If "New" of File menu is selected, 6. 2. 3 Device selection window is displayed. When the device and channel selection is complete, a new configuration will be created.

It can also be executed with the  button.

6. 2. 3 Device selection

Figure 6-3. Device Selection window (before device selection)

Microcomputer series and device are selected by this window.

If multi channels can be used , message is displayed under channel list and multi channels can be selected in channel list box.

Figure 6-4. Device Selection window (device applying multi channels)

Click “OK” button after selecting channels.

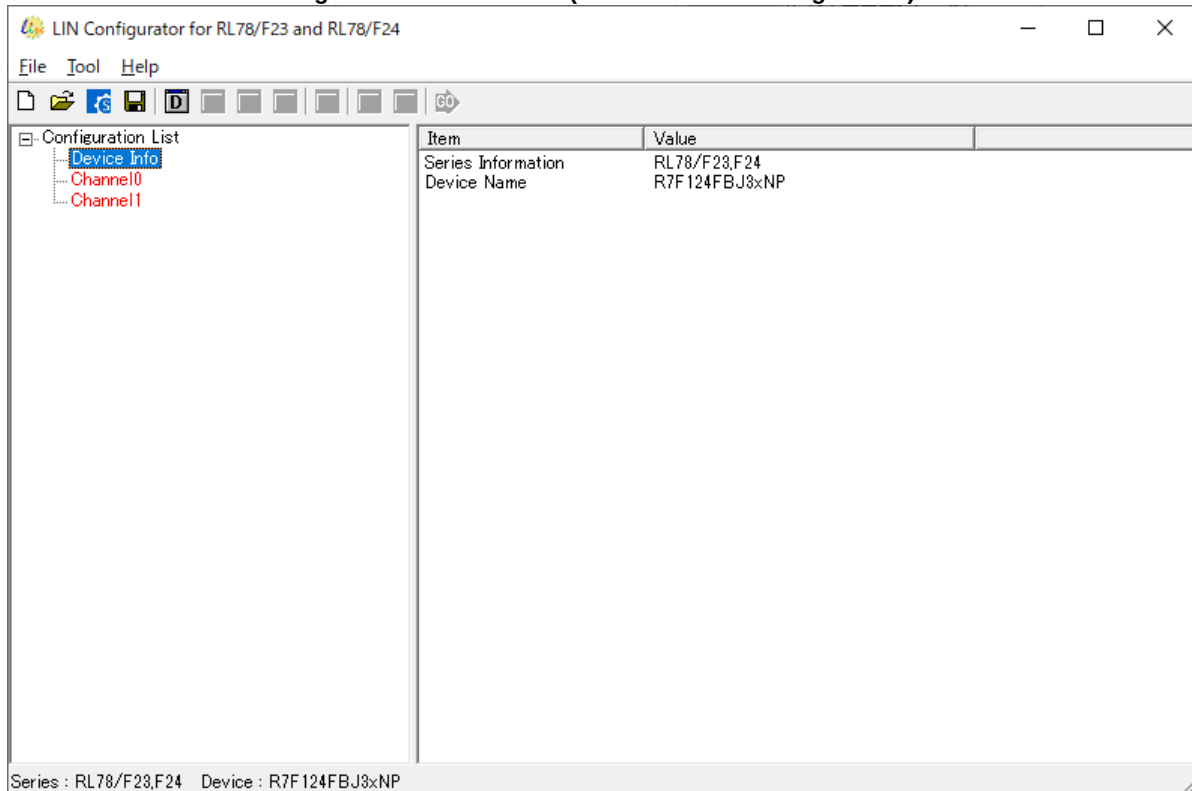
Then, all channels are hanged under “Configuration List” in Main Screen.

(The channel that is not configured is displayed in red letter.)

Remark

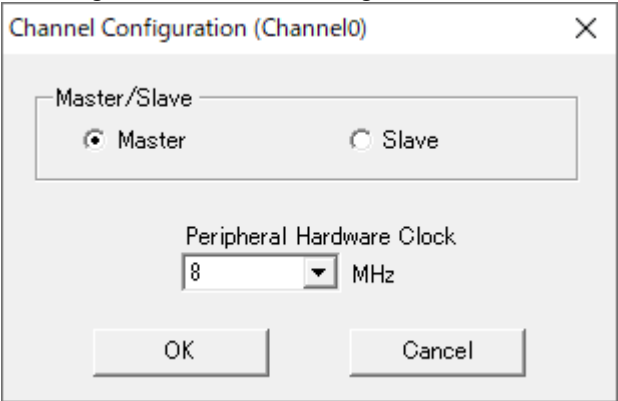
By reading the LIN-related definition file output from the smart configurator, the device and channel information set on the smart configurator will be reflected on this screen.

Figure 6-5. Main Screen (before channel configuration)



6. 2. 4 Channel configuration

Figure 6-6. Channel Configuration window

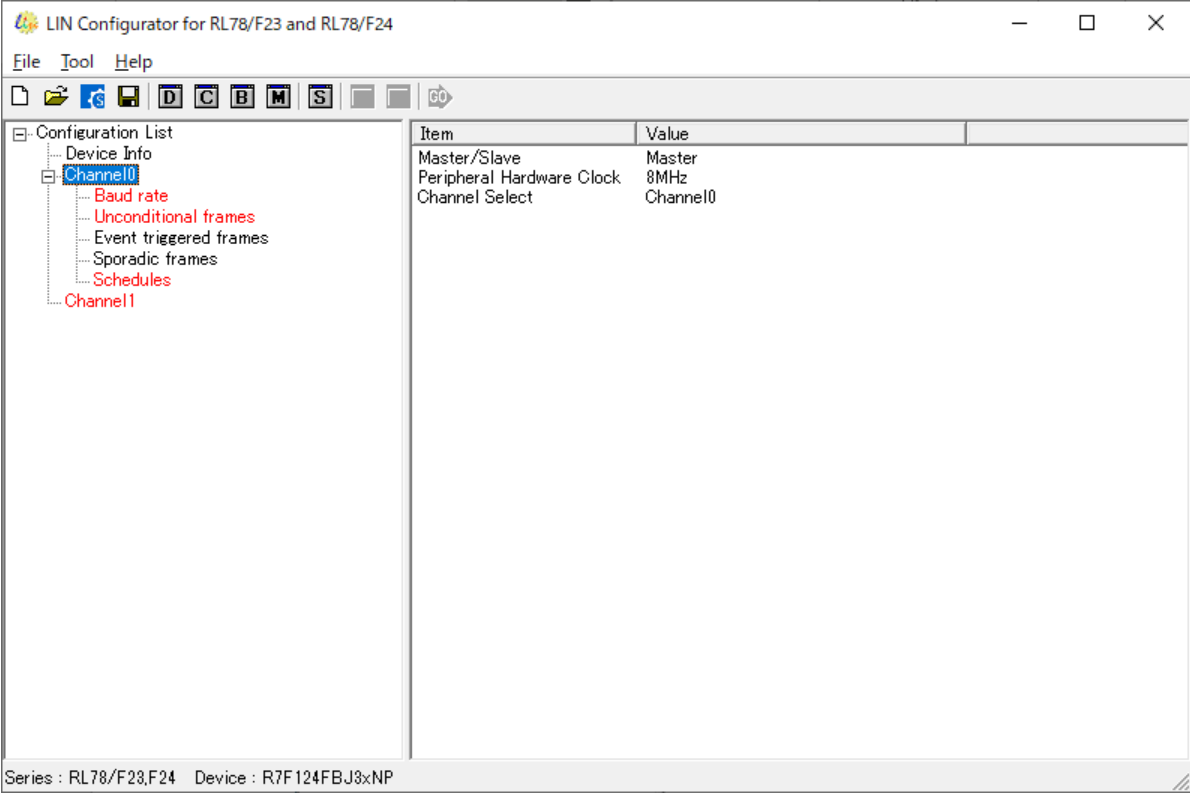


Master or slave and clock frequency used by LIN driver are selected in channel configuration window. Click “OK” button after configuration.

Remark Clock oscillator function isn’t configured in LIN driver. So, this setting should be configured application side.

Setting contents are hanged under a channel in main screen after channel configuration completion. The content that is not configured and that is required for generating source code is displayed in red letter.

Figure 6-7. Main Screen (before channel contents configuration)

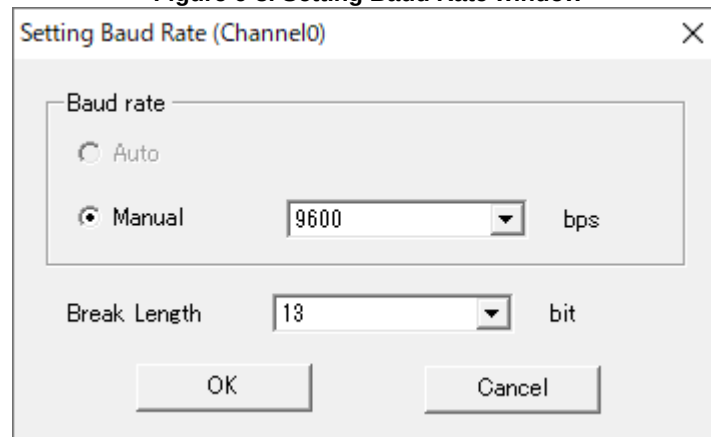


Remark

By reading the LIN-related definition file output from the Smart Configurator, the Master/Slave and input clock settings made on the Smart Configurator are reflected on this screen.

6. 2. 5 Baud rate configuration

Figure 6-8. Setting Baud Rate window



Baud rate and low level length of break field sent by this node is configured in this window.
(The low level length of break field is configured in master only.)

If channel is slave, "Auto" can be selected in baud rate configuration. Node sets baud rate automatically according to LIN header sent by master node.

Remark Compile option (ex. "`__LIN_HW_AUTO_BR__`") is required depending on the baud rate detection way. See readme.txt generated by LIN Configurator for more detail.

6. 2. 6 Message management

Figure 6-9. Setting Frame window

A message (LIN frame) is managed by this window. It's possible to choose "Sporadic Frame" only at a master channel. Frame ID number of LIN is set as "Valid/Invalid ID"(range: 0x00-0x3B). The parity is unnecessary. It's possible to choose "Invalid" only in case of a slave channel, and "0xFF" is set as frame ID number in that case. Initial data of a message is divided by a comma and it's set into "Initial Data". For example, set "0,0,0,0,0,0,0,0" when you set zero to all of data area(8byte length).

Operation of message registration is as follows. A registered message is shown to a list of left side of window.

- * Input the information of message to register to "new message registration pane".
- * Select frame type by radio button at "Select Frame", and press "Register" button.

Operation of message modification is as follows.

When a message in "Unconditional Frame" is double-clicked, "Setting Frame" window opens.

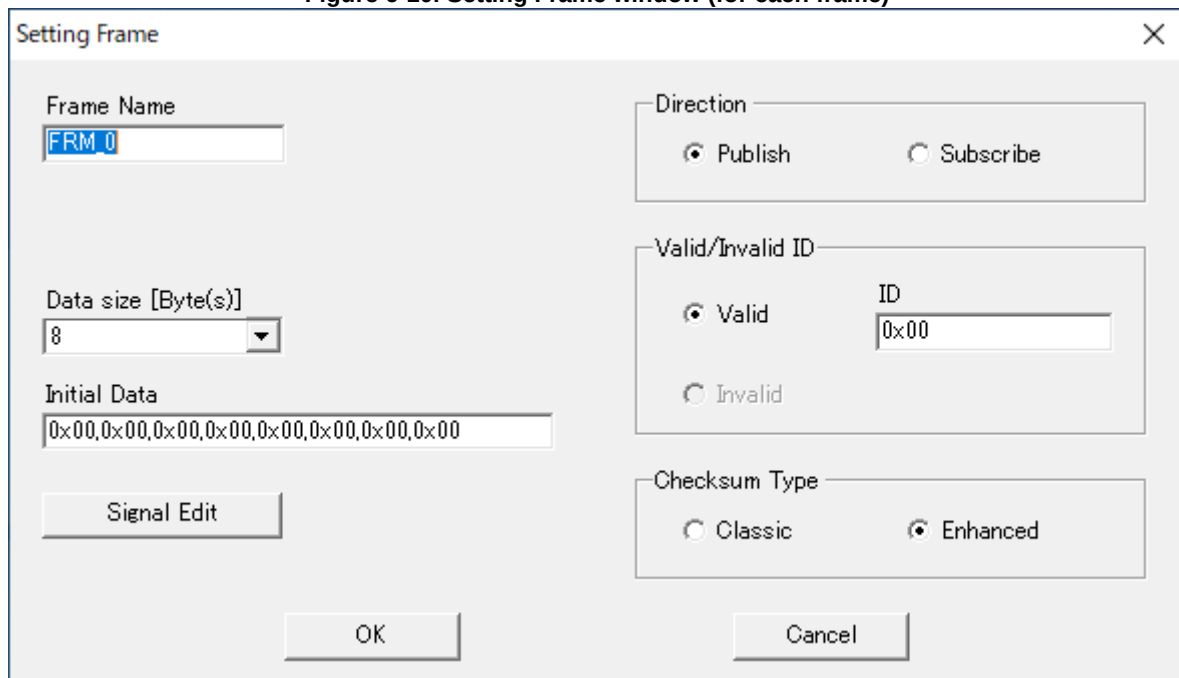
When a message in "Event Triggered Frame" is double-clicked, "Setting Event Triggered Frame Entry" window opens.

When a message in "Sporadic Frame" is double-clicked, "Setting Sporadic Frame Entry" window opens.

Remark: Length of "Frame Name" string must be up to 256. Alphabet, number and ' _ ' are available only.
However number can't be used on top.
Range of "ID" is from 0x00 to 0x3B.
Number of "Initial Data" must be the value of "Size".

(1) Unconditional frame configuration

Figure 6-10. Setting Frame window (for each frame)



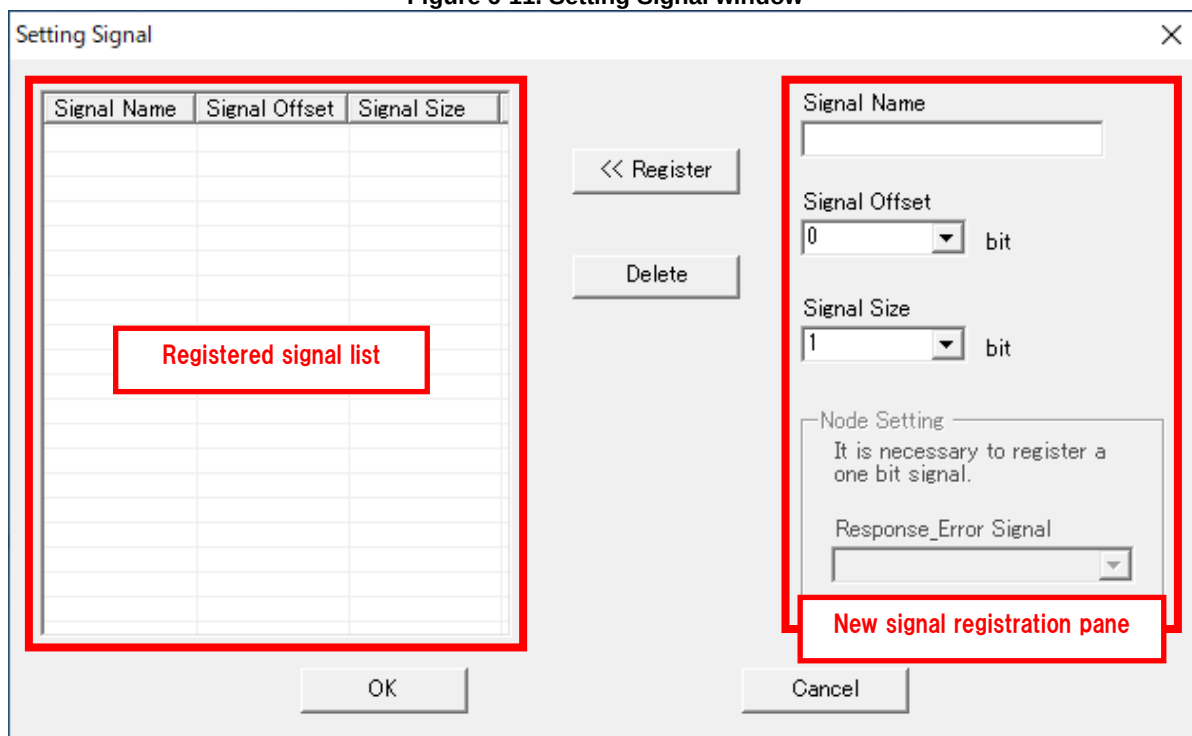
The "Setting Frame" dialog box is used for configuring frame parameters. It includes the following fields and controls:

- Frame Name:** A text box containing "FRM 0".
- Direction:** Radio buttons for "Publish" (selected) and "Subscribe".
- Data size [Byte(s)]:** A dropdown menu showing "8".
- Initial Data:** A text box containing "0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00".
- Signal Edit:** A button to open the "Setting Signal" window.
- Valid/Invalid ID:** Radio buttons for "Valid" (selected) and "Invalid". An adjacent text box shows "ID" as "0x00".
- Checksum Type:** Radio buttons for "Classic" and "Enhanced" (selected).
- OK** and **Cancel** buttons at the bottom.

When a message in "Unconditional Frame" is double-clicked, "Setting Frame" window opens. When "OK" is clicked after setting, setting is reflected to selected message. When "Signal Edit" is clicked, "Setting Signal" window opens.

(2) Signal management

Figure 6-11. Setting Signal window



A signal in the message is managed in this window.

When the item of "new signal registration pane" is set and "Register" button is clicked, a signal is added. When you choose signal from "registered signal list" and click "Delete" button, it's possible to remove information of its signal. When a message in "registered signal list" is double-clicked, "Setting Signal" window opens.

In case of a slave channel, 1 of "Response_Error_Signal" is needed about a channel. It's possible to assign only a signal of 1 bit length in the send message to "Response_Error_Signal". "Response_Error_Signal" can be selected by setting a signal of Signal Size 1 for an unconditional frame with Direction Publish.s

Remark: Length of "Signal Name" string must be up to 256. Alphabet, number and '_' are available only. However number can't be used on top.

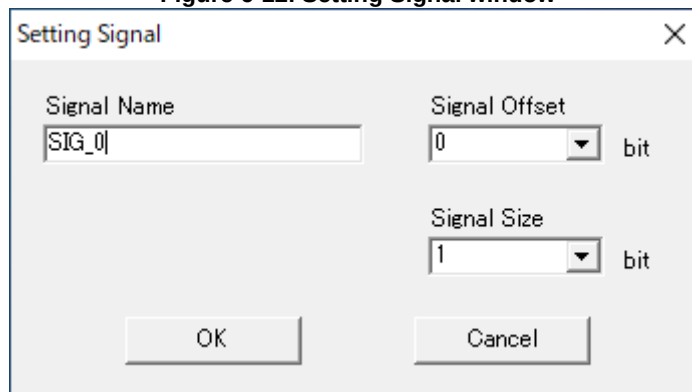
Don't overlap between the definition areas of any signals when multi signals are defined in one frame.

Don't define a signal striding 3 bytes or more.

Define as "Byte Array Signal" if signal size is 16 bit or more.

(3) Signal configuration

Figure 6-12. Setting Signal window

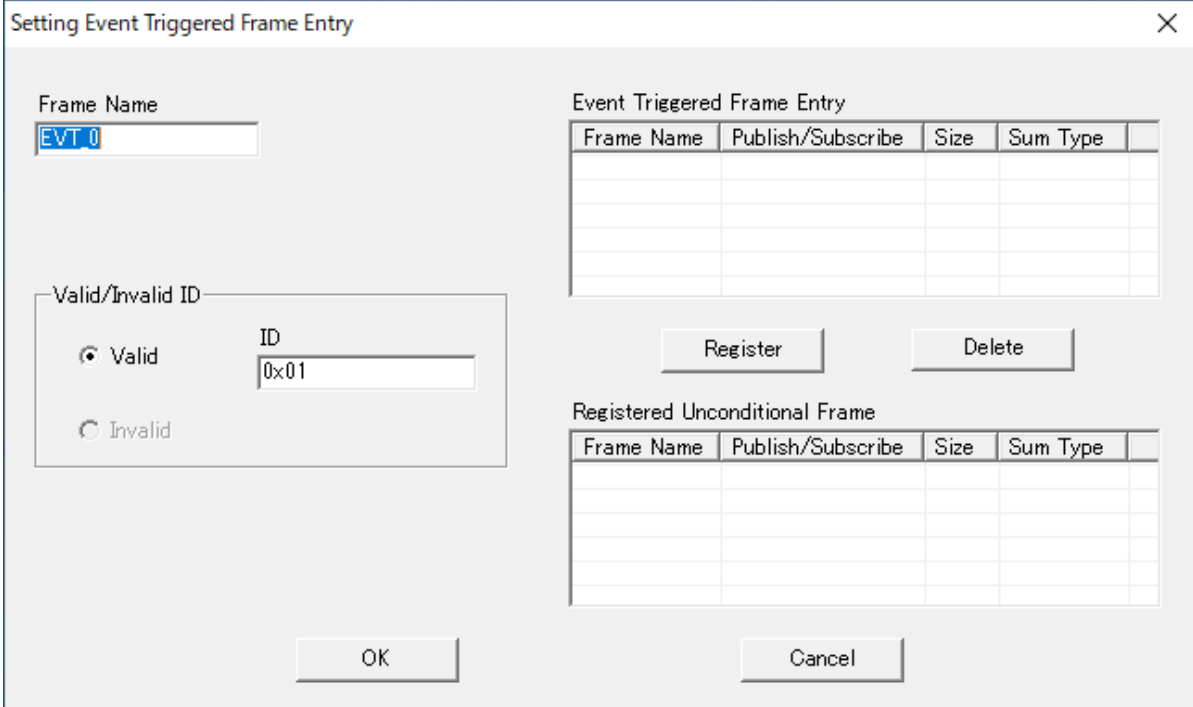


Already registered signal setting is configured in this window.

Click "OK" after configuration. Setting is reflected after clicking "OK" in "Setting Signal" window and "Setting Frame" windows.

(4) Unconditional frame configuration

Figure 6-13. Setting Event Triggered Frame Entry window



The window titled "Setting Event Triggered Frame Entry" contains the following elements:

- Frame Name:** A text field containing "EVT_0".
- Valid/Invalid ID:** A group box containing:
 - ☒ Valid
 - ☐ Invalid
 - ID:** A text field containing "0x01".
- Event Triggered Frame Entry:** A table with 5 columns: Frame Name, Publish/Subscribe, Size, Sum Type, and an empty column. It contains 5 empty rows.

Frame Name	Publish/Subscribe	Size	Sum Type	
- Buttons:** "Register" and "Delete" buttons are located below the "Event Triggered Frame Entry" table. "OK" and "Cancel" buttons are at the bottom of the window.
- Registered Unconditional Frame:** A table with 5 columns: Frame Name, Publish/Subscribe, Size, Sum Type, and an empty column. It contains 5 empty rows.

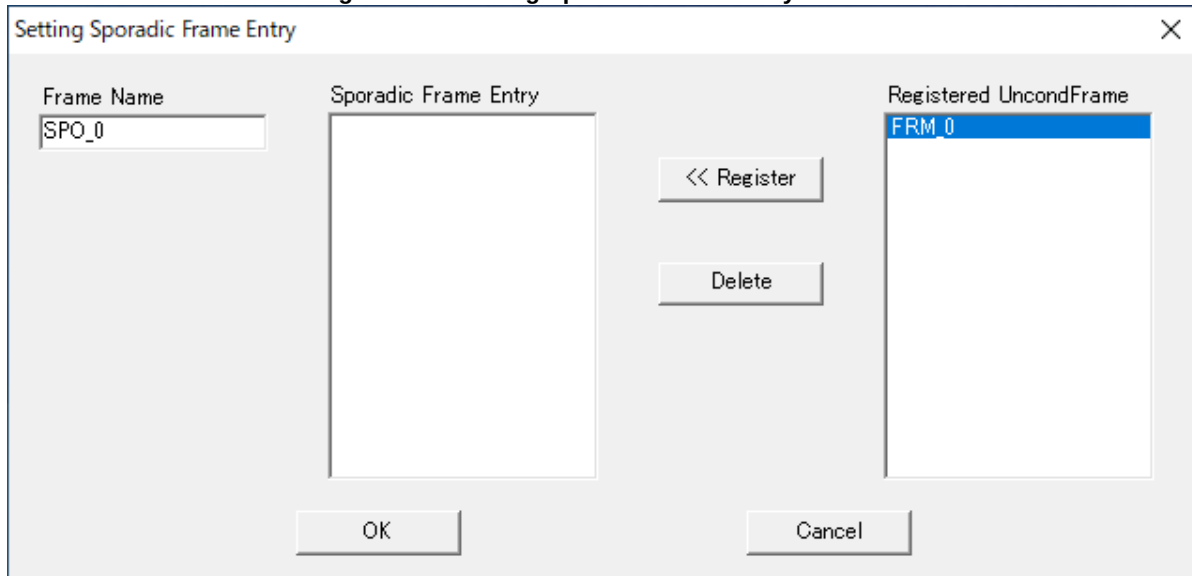
Frame Name	Publish/Subscribe	Size	Sum Type	

When a message in "Event Triggered Frame"(refer Figure 6-9) is double-clicked, this window opens.

When an unconditional frame is chosen from list of "Registered Unconditional Frame" and "Register" button is clicked, it's possible to relate to an event trigger frame. About "Valid/Invalid ID", please refer to . When "OK" button is clicked with this window and "Setting Frame" window, setting is reflected.

Remark: Length of "Frame Name" string must be up to 256. Alphabet, number and '_' are available only.
 However number can't be used on top.
 Range of "ID" is from 0x00 to 0x3B.
 Same communication direction, data size and checksum type must be used by all unconditional frames associated with one event triggered frame.

(5) Sporadic frame configuration

Figure 6-14. Setting Sporadic Frame Entry window

This window is opened when a message is double clicked in "Sporadic Frame" in registered message list.

Unconditional frame can be associated with sporadic frame by selecting unconditional frame in "Registered UncondFrame" list and clicking "Register".

Configuration is reflected after clicking "OK" and clicking "OK" in "Setting Frame" window.

Remark: Length of "Frame Name" string must be up to 256. Alphabet, number and '_' are available only.

However number can't be used on top.

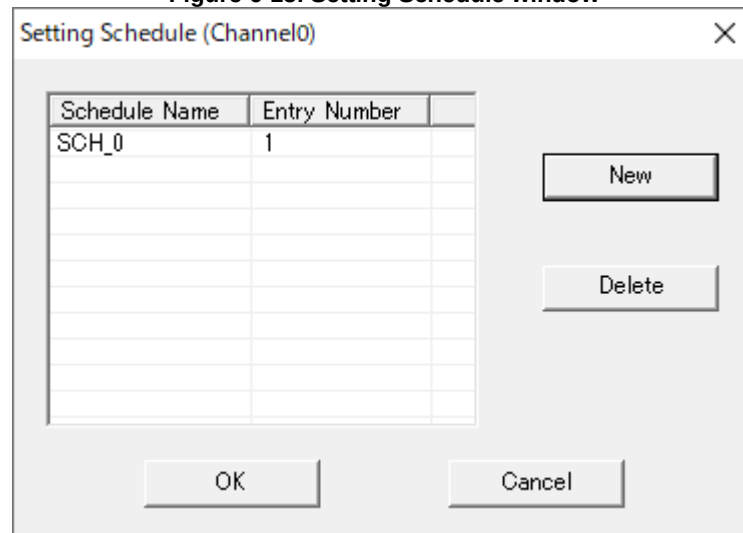
The unconditional frames in the "Registered Unconditional Frames" list are only "Publish" frames.

6. 2. 7 Schedule management

A table having communication timing of LIN frame is called Schedule Table.

(1) Schedule table management

Figure 6-15. Setting Schedule window



This window manages schedule table and this is used in master channel only.

Schedule table configuration window is opened when "New" or registered schedule is selected.

Configuration is reflected after clicking "OK".

Remark: Number of schedule table is from 1 to 255.

"OK" can be selected and LIN driver source code can be generated with no schedules.

(2) Schedule table configuration

Figure 6-16. Setting Schedule Entry window

[illegible]

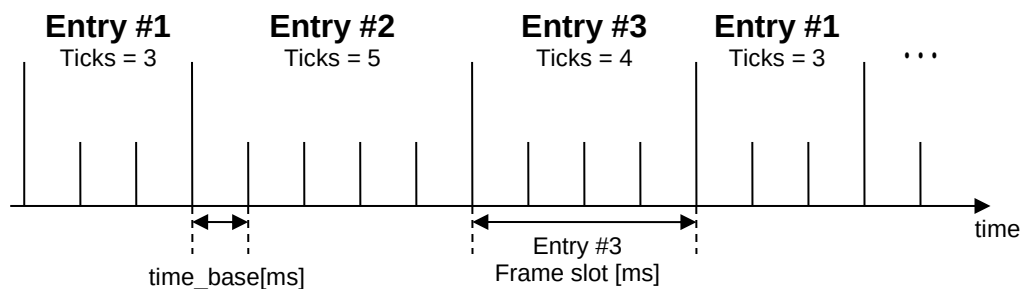
LIN header transmission schedule is configured in this window. This window is used by master channel only.

Write schedule name in "Schedule Name". Same schedule names must not be used in one channel.

Schedule entry is added when registered frame into schedule and Tick are selected and "Register" is clicked.

For example, behavior of the schedule when setting are Entry#1:Tick=3, Entry#2:Tick=5 and Entry#3:Tick=4 is indicated on below figure.

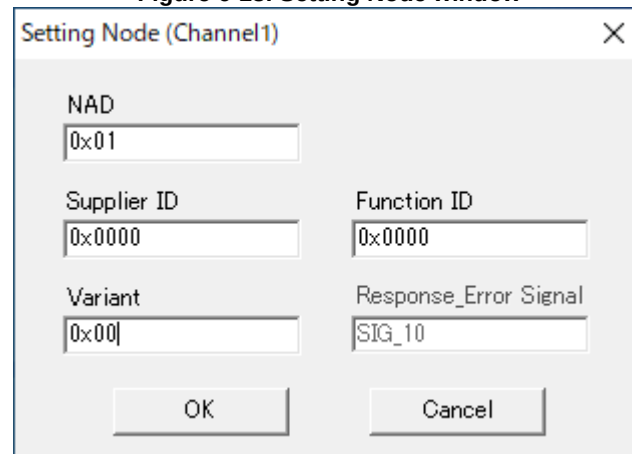
Figure 6-17. Behavior of the schedule



When the item in the schedule entry list is chosen and "Delete" button is clicked, selected information is removed. When "OK" button clicked, setting is reflected.

6. 2. 8 Node configuration

Figure 6-18. Setting Node window



The image shows a dialog box titled "Setting Node (Channel1)". It contains five input fields arranged in two columns. The first column has "NAD" with value "0x01", "Supplier ID" with value "0x0000", and "Variant" with value "0x00". The second column has "Function ID" with value "0x0000" and "Response_Error Signal" with value "SIG_10". At the bottom are "OK" and "Cancel" buttons.

Field	Value
NAD	0x01
Supplier ID	0x0000
Function ID	0x0000
Variant	0x00
Response_Error Signal	SIG_10

Node information of LIN slave is set in this window.

Node address (1 to 127) is set in "NAD".

IDs of product are set in "Supplier ID", "Function ID" and "Variant".

Signal name of "Response_Error Signal" set by "Setting Signal" window is displayed in "Response_Error Signal". This can't be changed in this window.

Setting is reflected after clicking "OK".

Remark: Range of NAD is from 1 to 255 (from 0x01 to 0xFF).

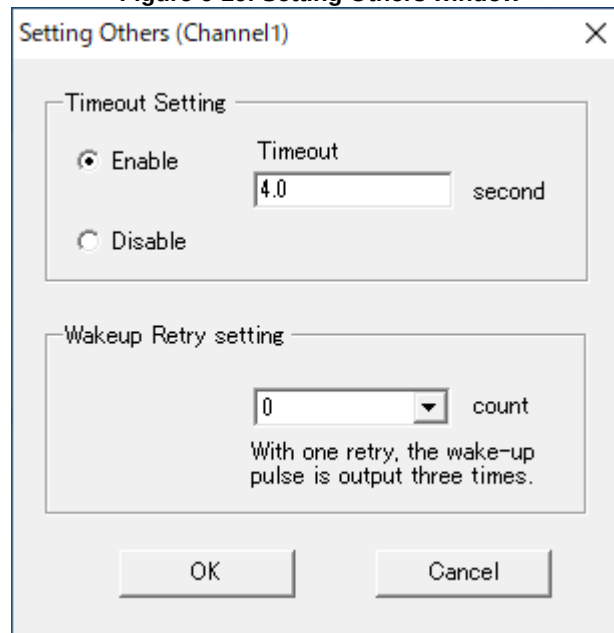
Range of Supplier ID is from 0 to 65535 (from 0x0000 to 0xFFFF).

Range of Function ID is from 0 to 65535 (from 0x0000 to 0xFFFF).

Range of Variant is from 0 to 255 (from 0x00 to 0xFF).

6. 2. 9 Other configurations

Figure 6-19. Setting Others window



Sets the bus timeout for LIN slaves and the number of retries for wake-up pulse output.

If "Enable" is selected, the time of bus timeout is set. LIN slave goes to sleep mode when LIN bus is in-active during this time.

If "Disable" is selected, LIN driver doesn't go to sleep mode.

The number of wake-up retry pulse output can be controlled and the number of 3 retry pulses can be set in the range of 0 to 80.

Setting is reflected after clicking "OK".

Remark: Range of timeout is from 0.1 to 18.0[sec].

6. 2. 10 Save / Load setting file

"Save" button in the main window upper part or "Save" menu or "Save As" menu in "File" menu are clicked, it's possible to write the setting contents of LIN Configurator to file by the XML form.

"Open" button in the upper part or "Open" menu in "File" menu are clicked, it's possible to read the setting contents from file.

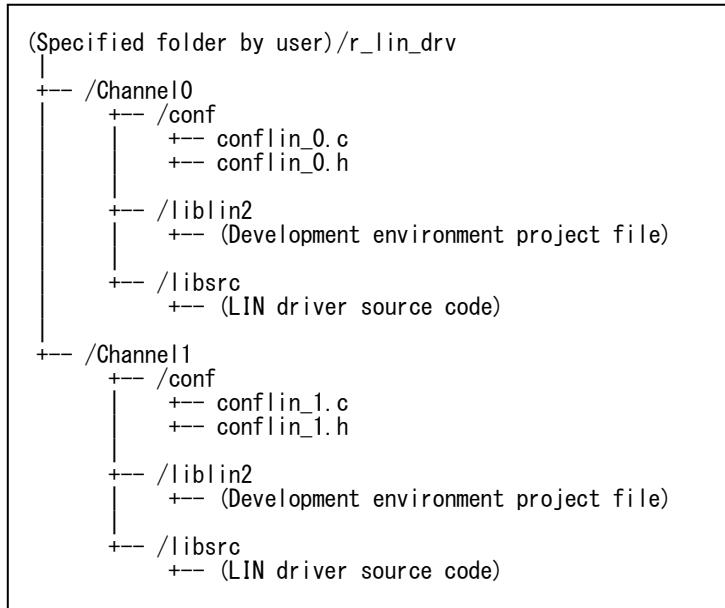
6. 2. 11 Generate source code

"Go" button in the main window upper part or "Generate Source Code" menu in the "Tool" menu is clicked, source code and development environment project of a LIN driver are generated.

An example of output destination of a file is shown in below.

(In case of channel 0 and channel 1 of R7F124FPJ3xNP.)

Figure 6-20. Example of output folder



[Note]

Configurator outputs project files for CC-RL and IAR compiler.

6. 3 Error message list

Table 6-3. Error message list

Error Message	Grounds
XXX is not specified (XXX is item name.)	OK and Register were pushed down when the value not input to item XXX.
XXX is not numerical number (XXX is item name.)	OK and Register were pushed down when things except a numeric character string had been input in the numeric input column.
XXX over the maximum limit (XXX is item name.)	OK and Register were pushed down when the numerical value more than the maximum value had been input in the numeric input column.
XXX is smaller than the minimum limit (XXX is item name.)	OK and Register were pushed down while had input for the numerical value that fell below minimum value in the numeric input column.
The input value of XXX is illegal (XXX is item name.)	OK and Register were pushed down while had input for a numeric character string that did not exist in the list in the combo box for numeric specification. (When the character string input is possible from the keyboard.)
	OK and Register were pushed down while had input for a not corresponding character string to the identifier character pattern of C language in the name input column.
Can not open the device entry file	It failed in reading the XML file of configurator.
Can not open the specified file	Failed to read the configurator project file or the LIN-related header file output by the smart configurator.
	It failed in writing the project file of configurator.
Illegal file format of the device entry file	The XML description was correct and it was not possible to interpret it by reading the XML file of configurator.
Illegal file format	The XML description was correct and it was not possible to interpret it by reading the project file of configurator.
Can not make a configuration file	It failed in the configuration file generation.
Can not make a library file	It failed in the output of the library.
Baud rate is not specified	The configuration file generation was executed with the setting on the baud rate configuration screen not done.
One or more unconditional frames are needed	The configuration file generation was executed when there is no registered unconditional frame.
Signal is not specified	The configuration file generation was executed when there is no registered signal.
Schedule is not specified (Master Only)	The configuration file generation had been executed before the setting on the schedule configuration screen was done.
Node is not found (Slave Only)	The configuration file generation had been executed before the setting on the node configuration screen was done.
Response_Error Signal is not specified (Slave Only)	The configuration file generation had been executed before Response_Error Signal was set.
The extra information is not specified (Slave Only)	The configuration file generation had been executed before the setting on external configuration screen was done.
There is a frame that Valid ID is not specified (Master Only)	The configuration file generation was executed when there is a frame which had not registered Valid ID.
Frame Name is specified redundantly	OK or Register was pushed down when the same name as another data had been input to Frame Name.
The input value of Initial Data is illegal	OK or Register was pushed down in the state that the numeric input number of Initial Data is not corresponding to specification Size.
The input value of Direction is illegal	OK was pushed down when the Size had been changed while renewing the unconditional frame related to the event triggered frame.

Error Message	Grounds
Can not register Unconditional Frame any more	New or Register was pushed down when the registration entry reached the maximum number.
Can not register Event Triggered Frame any more	
Can not register Sporadic Frame any more	
Can not register Signal any more	
Can not register Event Triggered Frame entry any more	
Can not register Sporadic Frame entry any more	
Can not register Schedule any more	
Can not register Schedule Entry any more	
Can not update the Frame Name	OK was pushed down when Frame Name had been changed while renewing the unconditional frame related to the event triggered frame or the Sporadic frame, or OK was pressed with Frame Name had been changed while renewing the unconditional frame included in the entry of the schedule table.
	OK was pushed down when Frame Name had been changed while renewing the event triggered frame included in the entry of the schedule table.
	OK was pushed down when Frame Name had been changed while renewing the Sporadic frame included in the entry of the schedule table.
Can not update the Size	OK was pushed down when Size had been changed while renewing the unconditional frame related to the event triggered frame.
Can not update the Direction	OK was pushed down when Direction had been changed while renewing the frame that corresponded to either the following. a) It is registered in the frame that relates the event triggered frame and it is master. b) Another has the unconditional frame that is registered in a related frame of the event triggered frame and related to the same event triggered frame and it is slave.
Can not update the Checksum Type	OK was pushed down when Checksum Type had been changed while renewing the unconditional frame related to the event triggered frame.
The input of Direction is illegal	Direction was made Subscribe and OK was pushed down when the signal that had been allocated in Response_Error.
The input of Event Triggered Frame Entry is illegal	Register was pushed down while had selected for an unconditional frame that the listed entry is not corresponding to Direction, Size, and Checksum Type.
	OK and Register were pushed down when there illegal data in the listed entry.
The input of Sporadic Frame Entry is illegal	OK was pushed down when there illegal data in the listed entry.
Significant digit of a Timeout is one place of decimals	OK was pushed down when the following input in Timeout from the first decimal place.
Schedule Name is specified redundantly	OK was pushed down when the same name as another data had been input to Schedule Name.
Data of Schedule Entry is not adjusted	OK and Register were pushed down when there illegal data (The mistake is found in Frame Type corresponding to Frame Name) in the listed entry.
Normal Frame is not selected	When the schedule entry of Normal Frame was registered, Register was pushed down when the object frame had not been selected.
Signal Name is specified redundantly	Register was pushed down when the same name as another data had been input to Signal Name.
The input of Signal Offset is illegal	Register was pushed down when things except the multiple of 8 had been input to Signal Offset when Signal Size exceeded 16bit.

Error Message	Grounds
The input of Signal range is illegal	Register was pushed down in the following states. a) The range of the signal is not included to a specified data size. b) The range of the signal extends over 3 bytes. c) The range of the signal comes in succession with other signals.
	OK was pushed down when there a signal not included to a specified data size.
Can not update the Signal Name	OK was pushed down when Signal Name had been changed while updating the signal registered as an error signal.
Can not update the Signal Size	OK was pushed down when Signal Size had been changed excluding 1 while updating the signal registered as an error signal.

6. 4 Warning message list

Table 6-4. Warning message list

Warning Message	Grounds
Do you want to create a new project without saving the current project?	[File]-[New] (make project) was done while not save the change.
Do you want to save the project file?	[File]-[Exit] or click close button in window was done while not save the change.
Do you want to load the project without saving the current project?	[File]-[Open] (Project read) was done while not save the change.
conflin.h already exists. Do you want to overwrite it?	The configuration file is already exists in the specified folder.

CHAPTER 7 LIN 2.1 SOFTWARE DRIVER OVERVIEW

7. 1 Signal Types (Master/Slave)

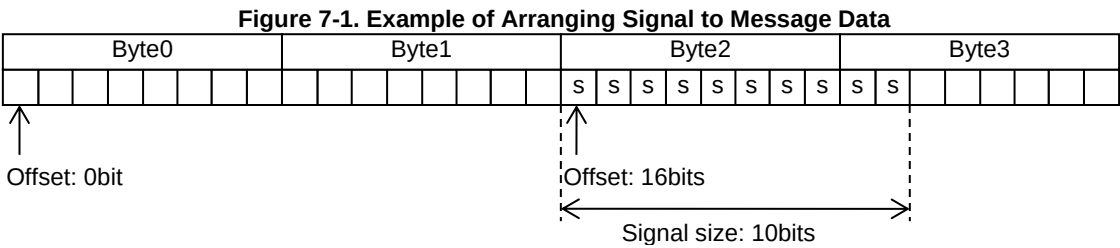
The LIN 2.1 software driver can treat two kinds of signals of the scalar signal from 1 to 16 bits and the byte array signal from 1 to 8 bytes. The signal is defined by editing conflin_x.c.

More than 2 signals can be defined in an arbitrary place in one frame. However, the scalar signal that exceeds over 2 bytes boundaries cannot be defined.

The access from the user application to the signal is possible with a call of the LIN 2.1 API.

E.g. when contain 10 bit signal “S” in 4 bytes long frame. (Message buffer 0)

Parameter:
Frame size : 4 bytes
Signal initial value: 0x03FF
Offset : 16 bits
Signal size : 10 bits



7.2 Frame Format (Master/Slave)

The frame format in the LIN communication is as follows.

Figure 7-2. Frame Format

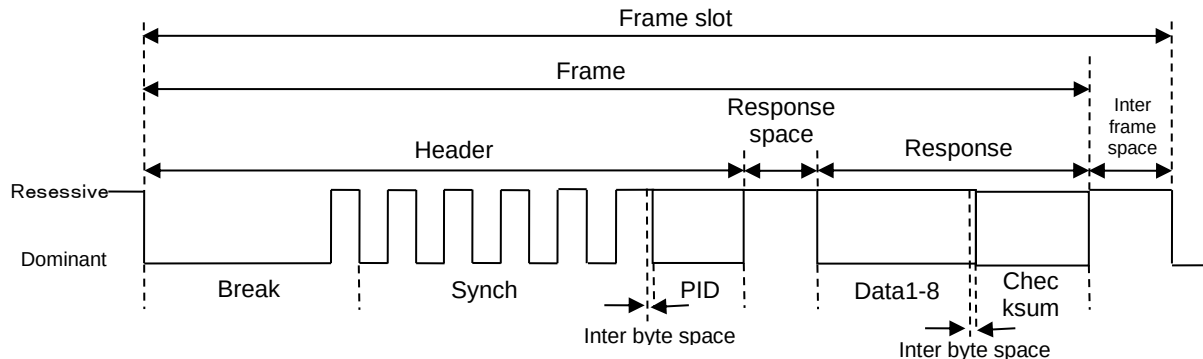
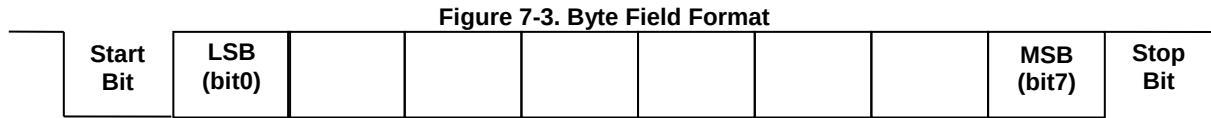


Table 7-1. Frame Component List

Term	Designation
Frame slot	Transfer time allocated in frame forwarding once of management by master.
Frame	Unit of LIN communication.
Header	Transmission data from master from Break to PID.
Response space	Time from header transmission completion to response transmission beginning.
Response	Data sent from the slave or master according to PID included in header.
Inter Frame space	It is time after it ends (forwarding completion of checksum) of one transfer until the next transfer beginning (negative edge generation of Break). It should be over 0.
Inter byte space	It is time from stop bit to following start bit. It should be over 0.
Break	Over 13 bits low pulse. If it is over 11 bits in the slave, it is considered Break.
Synch	Data of 0x55. Slave use for baud rate detection.
PID	Frame ID with parity to identify frame. In 2 high order bits are parity, and in 6 low order bits are frame ID.
Data1-8	Data from 1 to 8 bytes. The initial data value is specified with the configuration file.
Checksum	Inverse value of 8 bit checksum with carry over. The classic checksum calculates by using all data, and the enhanced checksum calculates by using all PID and data. In sporadic frame, the classic checksum must be used.

7.2.1 Byte Field

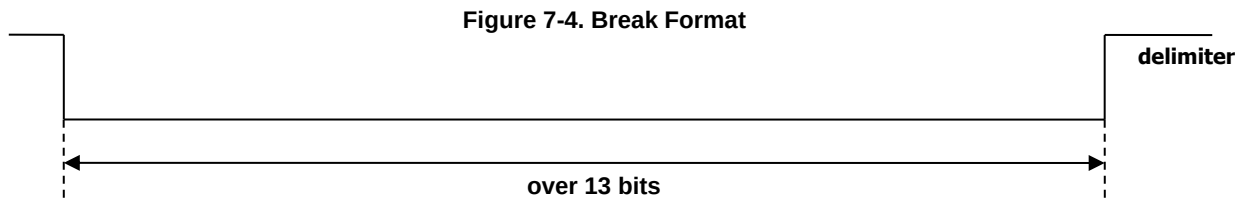
Byte field format excluding Break is as follows.



7.2.2 Break Field

The Break field is composed of over 13 bits low pulse and over 1 bit high pulse (delimiter).

For the LIN 2.1 slave, over 11 bits low pulse is recognized as Break.



Note:

- It is possible to set it from 13 to 20 bits in LIN 2.1 master driver.
- The maximum value of the low pulse depends on the maximum value of the header (1.4 times minimum value).

7.2.3 Frame Length

A nominal value at the frame transfer time is corresponding to the forwarded number of bits (value that does not contain response space and inter byte space).

The calculation formula of frame length is shown as follows.

$$\begin{aligned}
 T_{Header_Nominal} &= 34 \times T_{Bit} \\
 T_{Response_Nominal} &= 10 \times (N_{Data} + 1) \times T_{Bit} \\
 T_{Frame_Nominal} &= T_{Header_Nominal} + T_{Response_Nominal} \\
 * : T_{Bit} &= 1 \text{ bit time, } N_{Data} = \text{number of data bytes}
 \end{aligned}$$

In LIN 2.1 Spec, the error margin of 40% is permissible to usual transfer time.

The calculation formula of maximum frame length is shown as follows.

$$\begin{aligned}
 T_{Header_Max} &= 1.4 \times T_{Header_Nominal} \\
 T_{Response_Max} &= 1.4 \times T_{Response_Nominal} \\
 T_{Frame_Max} &= T_{Header_Max} + T_{Response_Max}
 \end{aligned}$$

7.3 Frame Transfer (Master/Slave)

The list of various LIN frame transfer types as defined by the LIN 2.1 Spec is shown below.

Table 7-2. Types of LIN Frames

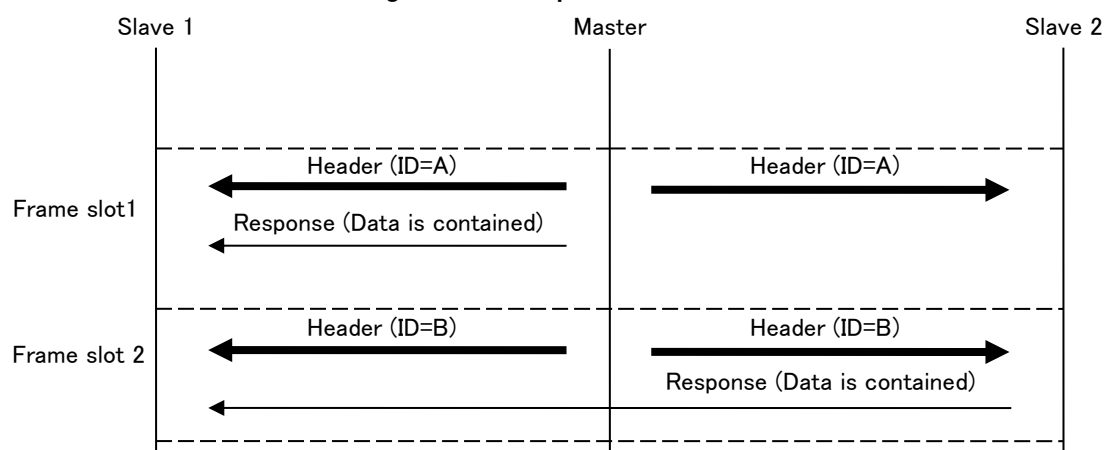
Frame name	Description
Unconditional frame	The frame using normal transfer.
Event triggered frame	The frames receive from slave that updated the signal.
Sporadic frame	The frame that treats information outside schedule.
Diagnostic frame	The frame to transfer specific information for node configuration. There is master request frame and slave response frame.

Examples of the different types of frame transfers are shown below.

7.3.1 Unconditional Frame Transfer

The response is transmitted from the master or slave node of the object by transmitting the header from the master. The master or slave node to be received receives the transmitted response.

Figure 7-5. Example of Data Transfer

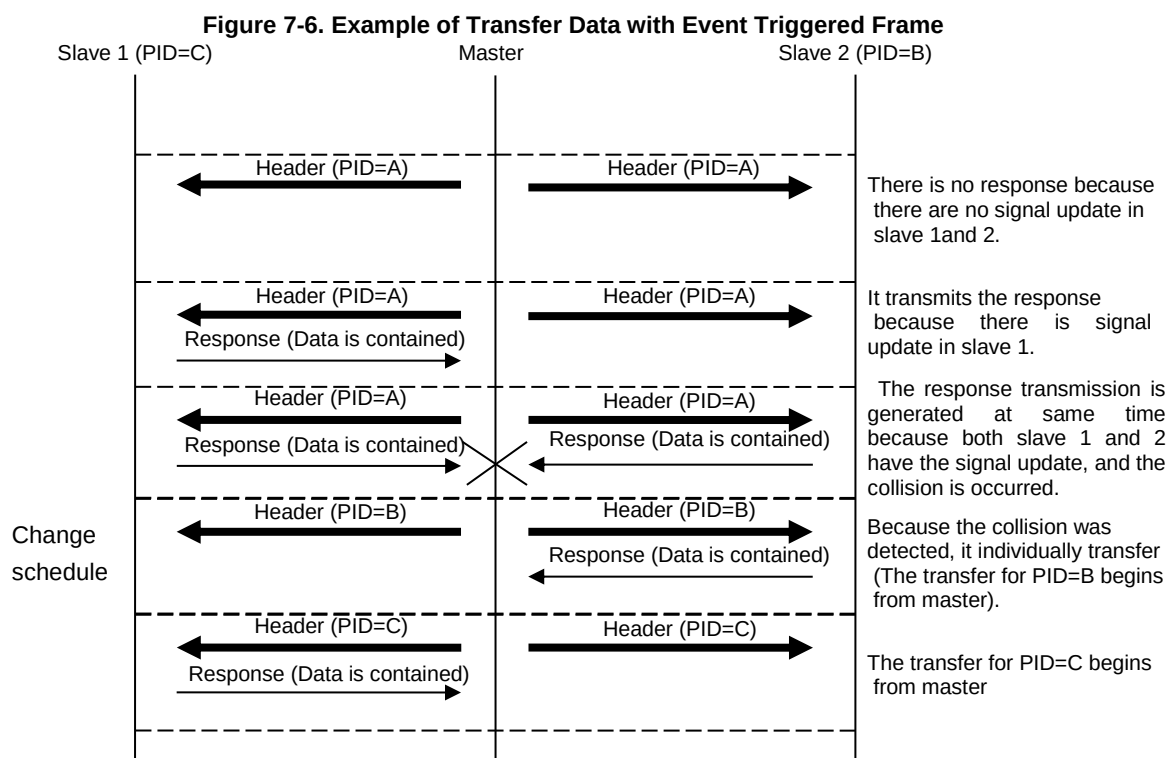


7.3.2 Event Trigger Frame Transfer

The Master receives the response of the related unconditional frame by using the event triggered frame (the transmission of the response is not supported). When the header of the event triggered frame is received, the slave transmits a response only in the case of having a signal update.

When any slave does not have a signal update, the response is empty. Also, when more than 2 nodes transmit a response at the same time, then a collision has occurred. In this case, the master mode should avoid the collision.

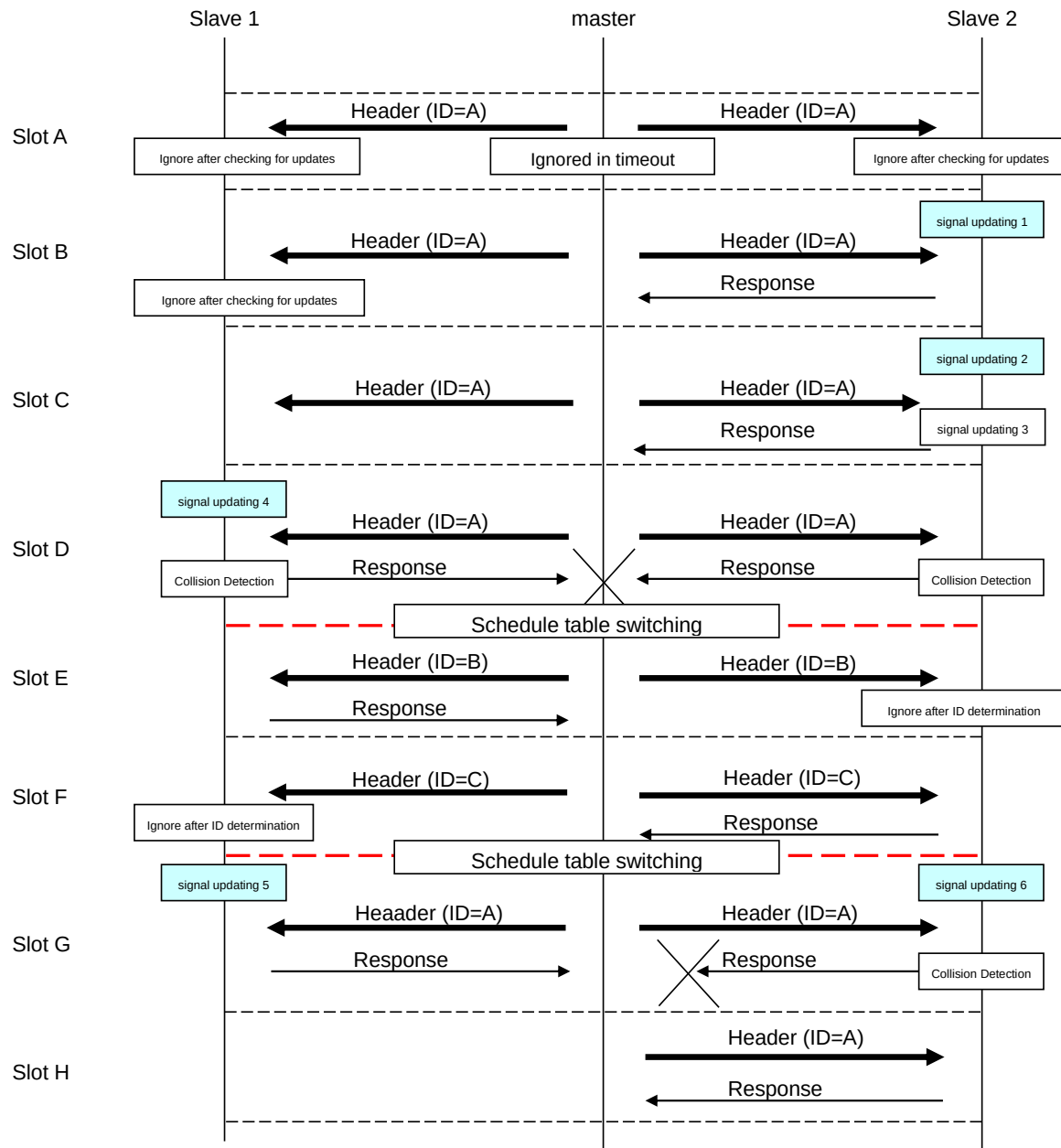
The event triggered frame is PID=A, and relating unconditional frame is PID=B, and PID=C. The following communications are executed.



Note:

1. When the error is detected by the event triggered frame reception, it is considered collision. It does not become the receive completion and an error.
2. Event triggered frame ID should be from 0x00 to 0x3b (does not contain parity).
3. Do not allocate the signal in first 1 byte of the response, because related unconditional frame's PID is allocated here. The signal can be allocated in 7 bytes except 1 head byte.
4. The unconditional frame related to the event triggered frame should be made the same checksum model.
5. The data length of the unconditional frame related to the event triggered frame should be made the same length.
6. The unconditional frame related to the event triggered frame must be sent from a different node.
7. The event triggered frame and the unconditional frame related to it cannot exist in the same schedule table.

The LIN 2.1 software driver performs a collision avoidance adjustment on the master to resolve the collision.



Slot A: Master asks both slaves (ID=A[0x00-0x3B], ID for event trigger), no response

Slot B: Master inquires both slaves (ID=A[0x00-0x3B], ID for event trigger), Slave 2 sends data to master (update value 1)

Slot C: Master inquires both slaves (ID=A[0x00-0x3B], ID for event trigger), slave 2 sends data to master (update value 2), when it coincides with update

Slot D: When the master makes a query to both slaves (ID=A[0x00-0x3B], ID for event trigger), both slaves send data to the master, both slaves are in a collision state, and the master can detect the collision.

Automatically switch to conflict resolution schedule.

Slot E: After collision detection, slave 1 sends data from slave 1 to master (ID=B[0x00-0x3B], for

unconditional frame) (update value 4)

Slot F: After collision detection, slave 2 sends data (update value 3) to the master (ID=C[0x00-0x3B], for unconditional frame).

After all collisions are resolved, the schedule is automatically switched back to the original schedule.

Slot G: The master sends a query to both slaves (ID=A[0x00-0x3B], ID for event trigger). Both slaves send data to the master, but although an error is detected inside Slave 2, the updated value 5 is sent as it is on the LIN line, and the master interprets this as a normal data transmission from Slave 1 rather than a collision detection. (*1: See frame example)

Slot H: Slave 2 sends data (update value 6) to the master (ID=C[0x00-0x3B], for unconditional frame). Since no data was sent last time due to an error, the updated value 6 is sent.

Reference

Example of a frame when the master cannot detect a collision

ID	Data	Sum	&	ID	Data	Sum	=	ID	Data	Sum
0x80	0x00	0x7F		0xc1	0xbe	0x7F		0x80	0x00	0x7F

Sum is a traditional checksum.

Remark:

1. If an error is detected in the transmission/reception of an event trigger frame, it is treated as a collision and no error is generated. It also does not mean that the transmission was successful.
2. The event trigger frame ID must be 0x0 to 0x3B (not including parity).
3. Since the first data byte stores the PID of the associated unconditional frame, a maximum of 7 bytes, excluding the first byte, can be allocated as the data area. Do not assign any signal to the first byte.
4. The checksum model of the unconditional frame associated with the event trigger should be the same (i.e. LIN1.3). The checksum model of the unconditional frame associated with the event trigger must be the same (i.e. LIN1.3 compliant slave nodes cannot be mixed with LIN 2.1 compliant slave nodes).
5. The data length of the unconditional frame associated with the event trigger should be the same.
6. All unconditional frames associated with an event trigger must be issued from different slave nodes.
7. The LIN 2.1 master driver will switch to the collision resolution schedule table when it detects a collision of event trigger frames.
8. In the LIN 2.1 master driver, collision resolution for event trigger frames is done using unconditional frames.
9. If an error occurs in the unconditional frame during collision resolution, it is ignored.

7.3.3 Sporadic Frame Transfer

The Master node transfers the response of the related unconditional frame by using the sporadic frame.

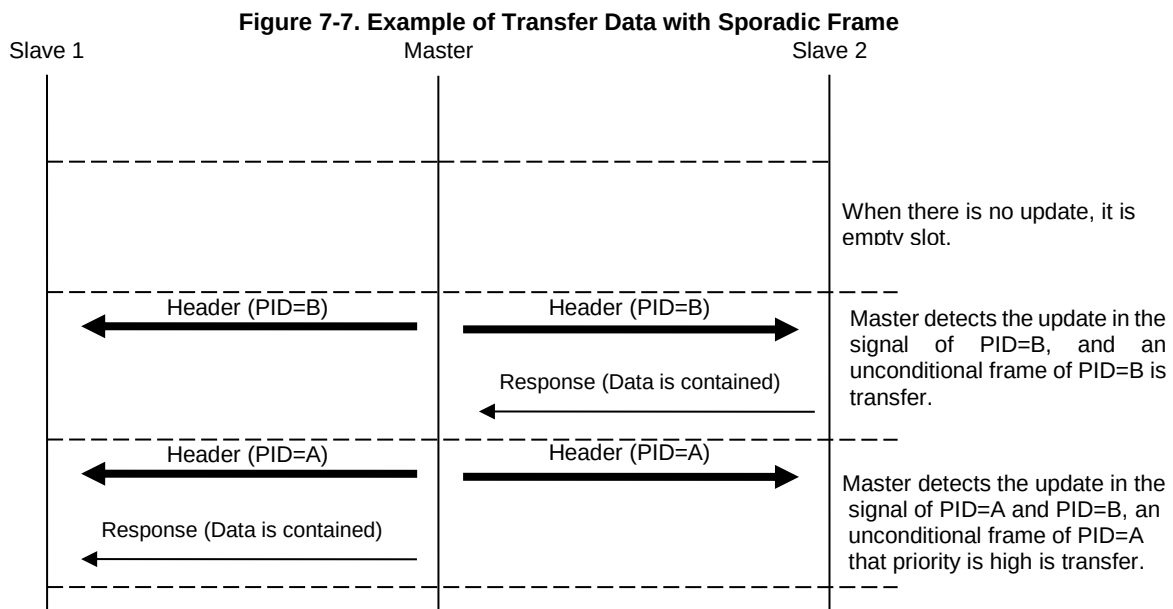
An unconditional frame is transmitted only when there is a signal update (the Master transmits the header of an unconditional frame; the slave transfers it according to the header).

When there is a signal update in over 2 relating unconditional frame, the highest priority unconditional frame is transferred.

The update of the signal in the sporadic frame is following processing.

- The signal writing the unconditional frame of the direction of the transfer is transmits.
- To receive the unconditional frame of the direction of the transfer is transmits related in the event triggered frame as an event triggered frame, the collision is detected.

The unconditional frame related in sporadic frame is PID=A (Transmit), and PID=B (reception). (The priority of A is high). The following communications are executed.



Note:

1. When there is no signal update in all relating unconditional frame, the bus becomes silent.
2. There is no limitation in the unconditional frame related to the sporadic frame. The different checksum model, the different frame length, and the frame as which allotted node is the same can be related to the same sporadic frame.

7.4 Response Error Notify Function (Master/Slave)

Error information in the slave node can be read by the master node by allocating a specific 1 bit signal for the response error.

In the slave node, when the error is detected in the response fields except the event triggered frame, "1" is set in the response error signal.

The signal is maintained as a "1" until it is sent to the master node. After the slave node has transmitted the signal to master node, the response error signal is cleared to "0".

The master node can confirm error information on the slave node by the following information.

Table 7-3. Response Error List

Master Receive Value	Designation
Response_Error = 0 (False)	Normal performance
Response_Error = 1 (True)	Temporary error
No response	Fatal error

Note:

1. The response error signal should be 1 bit size. When it is not 1 bit signal, correct information cannot be read.
2. Be allocating it when not using it by the user application.
3. Do not allocate in the event triggered frame.
4. The initial value of this signal should be defined as "0".

7.5 Sleep and Wakeup Function (Master/Slave)

7.5.1 Sleep Function

The slave node transitions to a sleep mode when the LIN 2.1 slave driver receives a go-to-sleep command or the bus is inactive for over 4 seconds.

After it transitions to sleep mode, it becomes wake up pulse waiting state.

When the LIN 2.1 master driver normally issues a go-to-sleep command, it transitions to a sleep mode. After it transitions to a sleep mode, it becomes a wake up waiting state, and the frame is not transferred.

A go-to-sleep command is transmitted as master request frame of the node configuration function. It is possible to transmit by allocating master request frame in the schedule slot, and call the `I_ifc_goto_sleep` function.

On RL78/F23,F24 driver, LIN macro(RLIN3) needs clock in sleep mode if the wake up method is dominant width.

Note: Set time with the configuration file. It is also possible to set the inactive bus detection to off.

And, timer interrupts are used in the process of determining bus off. This can extend the transition time to sleep mode due to conflicts with interrupts used by user applications.

7.5.2 Wake up Function

When a low edge is detected during sleep mode or the `I_ifc_wake_up` function is issued, the slave node will wake up.

- At the detection of a low edge
 - The slave node waiting Break and Synch fields.
 - As for master node, the restart of the schedule becomes possible by the `I_sch_tick` function.
- When the `I_ifc_wake_up` function is issued
 - Transmits a 260 us wake-up pulse to the LIN bus.
5 Tbit at 19200 bps .
 - If a break is not received within 150 ms after the wake-up pulse is sent, another wake-up pulse is sent. 3 wake-up pulses are sent, and a fourth wake-up pulse is sent 1.5 sec later. 3 wake-up pulses are considered a block, and the block transmission is repeated for a user-defined number of times. After that, if the bus is inactive for more than 4 seconds, it will enter the sleep mode.

The master node can resume the schedule by using the `I_sch_tick` function.

Note:

1. The LIN 2.1 software driver does not do the wait processing from the wake up to the frame transfer provided with LIN 2.1 Spec restart. After issuing the `I_ifc_wake_up` or low edge detection, restart the schedule waiting arbitrary time at the application level.
2. If the wake up is done from sleep mode, it enters the state immediately before changing sleep mode. Therefore, when the `I_sch_tick` function is called, the frame is not necessarily transferred. After wake up, the LIN 2.1 software driver is recommended to be initialized.

7. 6 Node Configuration Function (Master/Slave)

A slave node connected to a LIN cluster can be configured by using the LIN 2.1 software driver's node configuration function.

The configuration command can be transmitted to the slave node by allocating the master request frame and the slave response frame in the schedule table slot, and calling the `Id_assign_frame_id_range` function and the `Id_read_by_id` function.

7. 6. 1 Node Information

It is necessary to define node information on NAD, product ID, and response error signal in the slave node. It is defined in the configuration file.

Table 7-4. Node Information List

Node Information	Designation	Size
Initial NAD	Node unique number in LIN cluster.	1 byte
Supplier ID	ID allocated from LIN Consortium.	2 bytes
Function ID	Allocated ID of each function	2 bytes
Variant ID	Allocated ID of each product version in function not changed.	1 byte
Response error signal	Signal that treats error information in the slave node.	1 byte

7.6.2 Node Configuration

For this software driver, it corresponds to a part of the node configuration function with the master node. The following requests can be transmitted from the LIN 2.1 master driver.

- Assign frame identifier range
- Read by identifier

- Assign Frame Identifier range

Modifies or disables the PIDs of up to four message frames of nodes with matching NADs.

However, frames with IDs of 60~63 (0x3C~0x3F) cannot be changed.

The start index specifies the first frame to which the PID is assigned. The order of the frames depends on the slave node. The first index in the list starts at 0.

The setting values for the PID list are as follows

The communication format of the master request frame is as follows

Table 7-5. Master Request Frame Format

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb7	Start Index	PID (Index)	PID (Index +1)	PID (Index +2)	PID (Index +3)

The response from the slave is sent only if the NAD matches.

If all the assignments are not executed, a negative response will be returned.

Also, the slave node does not verify the specified PID, so the master node must set the correct PID.

- Note:
- It does not correspond to the slave response frame transfer at the normal termination.
 - Protected ID should specify ID with parity.

- Read By Identifier

The request is issued by the following formats by the issue of the `Id_read_by_id` function from the master node. Afterwards, information according to ID (D1:identifier) can be obtained from the slave node by the slave response frame.

The format of master request frame is as follows.

Table 7-6. Master Request Frame Format (Read by identifier)

NAD	PCI	SID	D1	D2	D3	D4	D5
NAD	0x06	0xb2	Identifier	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB

Only when NAD, supplier ID, and message ID are corresponding, PID is allocated to message ID. However, if wild card (0x7F) is specified for NAD, when supplier ID and message ID is corresponding, processing is executed.

Moreover, when it is not agree, it does not correspond to the slave response request from the master node.

The format of slave response frame is as follows.

Table 7-7. Slave Response Frame Format

ID	NAD	PCI	RSID	D1	D2	D3	D4	D5
0	NAD	0x06	0xf2	Supplier ID LSB	Supplier ID MSB	Function ID LSB	Function ID MSB	Variant

When the slave is not supporting the request from master, the negative response is returned.

Table 7-8. Negative Response Format

ID	NAD	PCI	RSID	D1	D2	D3	D4	D5
1 - 255	NAD	0x03	0x7f	Requested SID(=0xb2)	Error code (= 0x12)	0xff	0xff	0xff

Note:

For the LIN 2.1 slave driver, only identifier=0 (product ID reading) corresponds. The negative response is returned to other identifier.

7.7 Scheduling Function (Master)

It is possible to switch and transition of the schedule table defined with the configuration file by calling the `l_sch_tick` function and the `l_sch_set` function prepared as a scheduling function.

It is necessary to implement the scheduler by using a timer and the forementioned functions in the user application of a master node. (Refer to "5.2.6 Scheduler implementation (only master)".)

- About timing parameters for slave diagnostics

LIN 2.1 Spec specifies the following parameters as timing requirements for slave diagnostics.

P2 : Time between when the slave node receives the last diagnostic request and then can provide data for response

STmin : Minimum time required for the slave node to receive the next diagnostic request for a diagnostic request or to prepare to send the next diagnostic response for a diagnostic response

P2* : Time between the slave node sending a negative response and being able to provide data for the next response

Lin master/slave drivers do not implement these parameters.

The above time will be adjusted by the master driver's schedule.

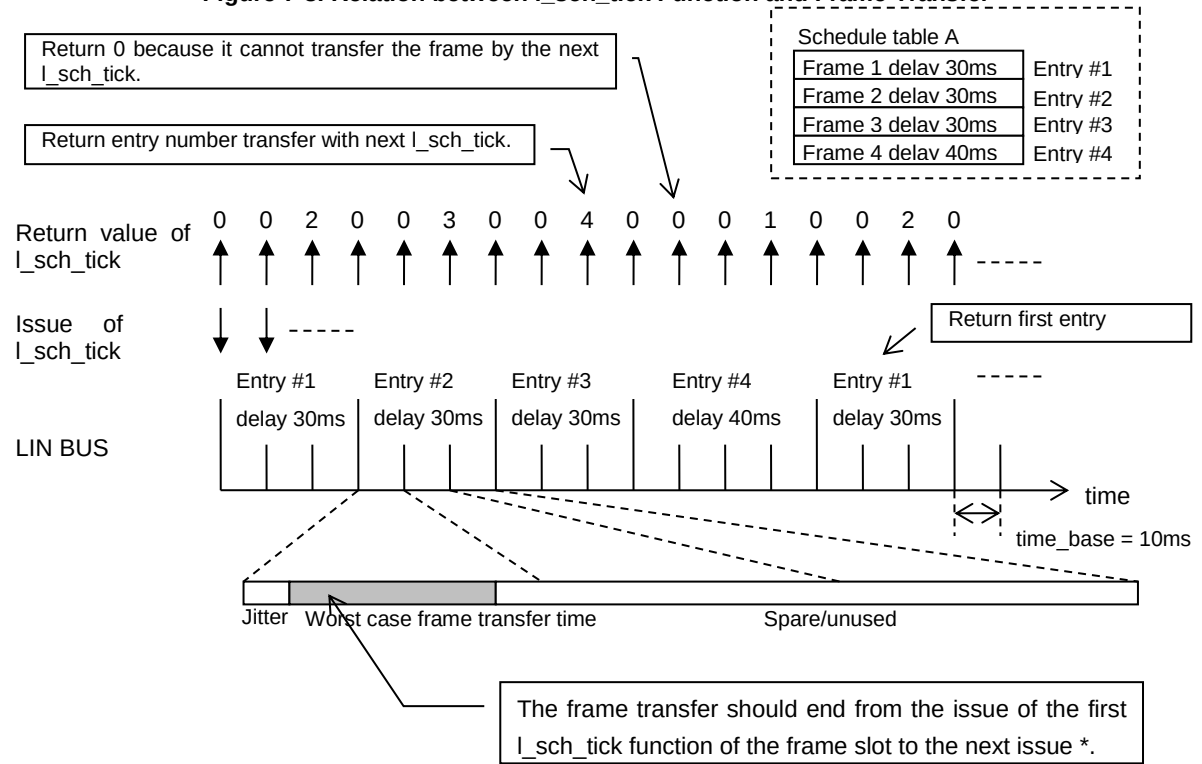
7.7.1 Schedule Transition (`l_sch_tick`)

The schedule can change by issuing the `l_sch_tick` function. When the `l_sch_tick` function is called and a current frame slot is entries in the schedule table ends, the frame transfer of the following entry begins (When the ending frame slot is the last entry of the schedule table, the frame transfer of the first entry of the schedule table begins). When the frame slot does not end, the frame transfer does not begin.

When transfer of the next entry begins (When a current frame slot ends) by issue of the `l_sch_tick` function, the entry number is returned. When the frame transfer cannot begin by issue of `l_sch_tick` function, "0" is returned. (Refer to Figure 7-8.)

When schedule table A composed of 4 entries is set, the relation between the issue the `l_sch_tick` function and frame transfer is shown below.

Figure 7-8. Relation between l_sch_tick Function and Frame Transfer



(*): The l_sch_tick call transferring the frame has not allowed

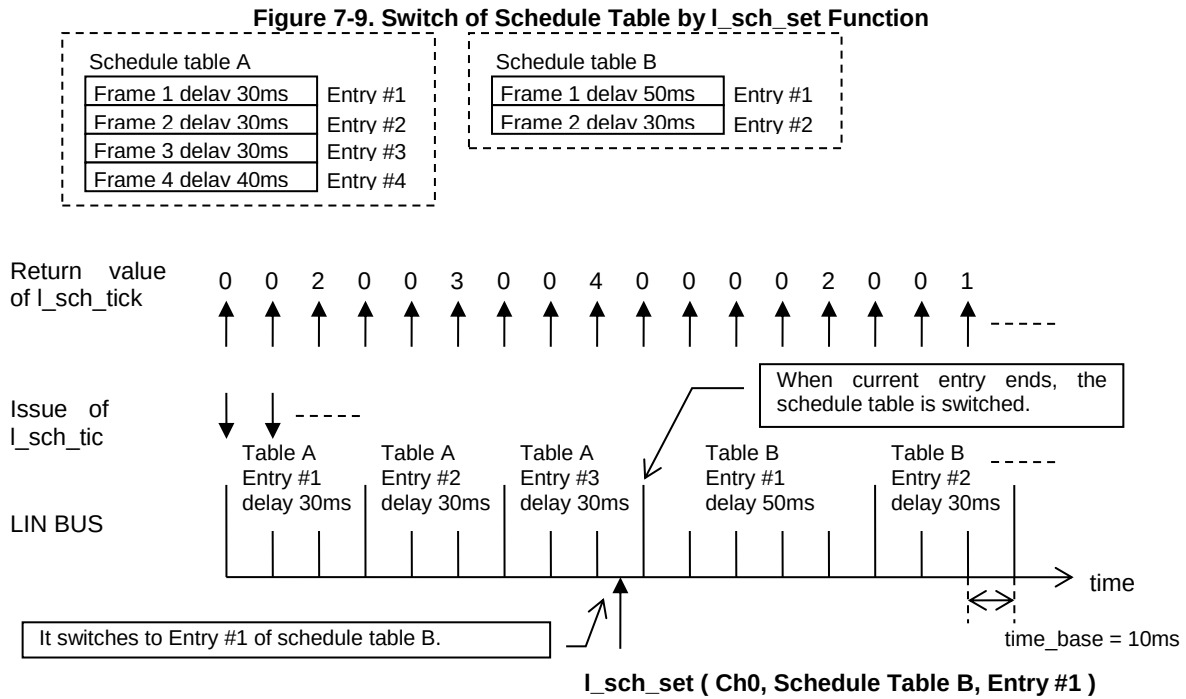
Complement: - 4 frame slots are entered in schedule table A.
 - Time_base is the maximum permissible time of the frame transfer.
 - Set the scheduler that time_base is must become long time than frame transfer time.

7.7.2 Schedule Switching (I_sch_set)

It is possible to switch to the schedule table specified that the I_sch_set function is issued from the schedule table under the operation. Moreover, the entry of the switched schedule table can be specified.

When the frame slot of the current schedule table entry ends, the schedule table is switched.

The I_sch_set function is issued, and the processing switched from schedule table A to schedule table B is shown below.



7.8 Auto Baud Rate Detecting Function (Option) (Slave)

For the LIN 2.1 slave driver, the automatic baud rate detection from 2400 to 20000 bps is done by measuring the width of the Break and Synch fields.

7.9 Driver configuration

7.9.1 Slave driver configuration

On RL78/F23,F24 slave driver, user can modify behavior of LIN driver using driver configuration.

Driver configuration is described in part “Driver configuration” in libsrc/conf/confslin_opt.h file.

Table 7-9. Driver configuration of RL78/F23,F24 slave (1 of 2)

Configuration points ("CONFSLIN_OPT" is omitted)	Contents	Setting values	Meaning of setting values
u1gLPRS_NORM_CFG	Prescaler division value for LIN macro. *1,*4	u1gLPRS_NODIV	1/1
		u1gLPRS_DIV2P1	1/2
		u1gLPRS_DIV2P2	1/4
		u1gLPRS_DIV2P3	1/8
		u1gLPRS_DIV2P4	1/16
		u1gLPRS_DIV2P5	1/32
		u1gLPRS_DIV2P6	1/64
		u1gLPRS_DIV2P7	1/128
u1gBUSWKUP_CFG *6	The wakeup method of LIN driver. If edge is selected, LIN macro does not need clock in sleep mode.	u1gBUSWKUP_EDGE	The method of wakeup is down edge detection.
		u1gBUSWKUP_WIDTH	The method of wakeup is dominant width detection.
u1gNSPB_NORM_CFG	Number of bit samplings. *4	u1gNSPB_4SMPL	4(only for auto baud rate mode)
		u1gNSPB_8SMPL	8(only for auto baud rate mode)
		u1gNSPB_16SMPL	16(only for fixed baud rate mode)
u1gLINMCK_CFG	Source of LIN communication clock. *5	u1gLINMCK_FCLK	f _{CLK}
		u1gLINMCK_FMX	f _{MX} *2
u1gINTLINTRMPR_CFG *6	Priority of LIN transmission completion interrupt.	u1gINTPR_LV0	Level 0 (Highest)
		u1gINTPR_LV1	Level 1
		u1gINTPR_LV2	Level 2
		u1gINTPR_LV3	Level 3 (Lowest)
u1gINTLINRVCP_CFG *6	Priority of LIN reception completion interrupt.	Same as u1gINTLINTRMPR_CFG.	-
u1gINTLINSTAPR_CFG *6	Priority of LIN status interrupt.	Same as u1gINTLINTRMPR_CFG.	-
u1gTMUNIT_CFG *6	Unit of interval timer(TAU). *3	u1gTMUNIT_UNIT0	Unit 0
		u1gTMUNIT_UNIT1	Unit 1
u1gTMCH_CFG *6	Channel of interval timer(TAU). *3	u1gTMCH_CH0	Channel 0
		u1gTMCH_CH1	Channel 1
		u1gTMCH_CH2	Channel 2
		u1gTMCH_CH3	Channel 3
		u1gTMCH_CH4	Channel 4
		u1gTMCH_CH5	Channel 5
		u1gTMCH_CH6	Channel 6
		u1gTMCH_CH7	Channel 7
u1gINTTMPR_CFG *6	Priority of interval timer(TAU) interrupt.	Same as u1gINTLINTRMPR_CFG.	-
u1gINTPPR_CFG *6	Priority of external interrupt.	Same as u1gINTLINTRMPR_CFG.	-

*1 : In auto baud rate mode, this value is not used. LIN driver set automatically as prescaler clock is between 8[MHz] and 12[MHz].

*2 : The setting value in LIN configurator is f_{MX} clock value. In this case, following modification is needed because the source of interval timer(TAU) is constantly f_{CLK} .

(conflin.c)

[Before modification]

```
const u2 ConfSLin_u2gTMPERICLOCK = (u2)CONFLIN_u2sPERICLOCK;
```

[After modification] (In case of f_{CLK} is 8[MHz])

```
const u2 ConfSLin_u2gTMPERICLOCK = (u2)800;
```

*3 : Please set the unit or a channel to be equipped with in the device of a used microcomputer.

Availability isn't checked in the driver.

*4 : Following calculation is done in LIN driver to obtain the Baud Rate pre-scaler register value (LBRP).

$$(LBRP \text{ value}) = \{ (LIN \text{ communication clock frequency}[Hz]) \times (Pre\text{-}scaler \text{ division value}) \} / \{ (Number \text{ of bit samplings}) \times (LIN \text{ communication baud rate}[bps]) \}$$

Calculation is done in integer value (decimal will be omitted). Above (LBRP value) must not be equal or less than zero.

*5 : There is a notice regarding the LIN communication clock source in LIN/UART module (RLIN3) of RL78/F23, F24. For details, please refer to the user's manual. The slave driver includes the following restriction for by this case.

- When setting f_{MX} (u1gLINMCK_FMX) as type of a LIN communication clock source (u1gINTLINTRMPR_CFG) and setting "Not detect time out error" (u1gTER_DISABLE) as "Time out error detection" (u1gTER_CFG), please set f_{CLK} clock more than 1.2 times of LIN communication clock source.

*6 : Configurable by the smart configurator.

Table 7-10. Driver configuration of RL78/F23,F24 slave (2 of 2)

Configuration points ("CONFSLIN_OPT" is omitted)	Contents	Setting values	Meaning of setting values
u1gLRDNFS_NORM_CFG	Switch of noise filter for LIN communication Rx.	u1gLRDNFS_USE	Use
		u1gLRDNFS_NOUSE	Not use
u1gLRDNFS_WKUP_CFG	Switch of noise filter for LIN sleep Rx.	u1gLRDNFS_USE	Use
		u1gLRDNFS_NOUSE	Not use
u1gBLT_CFG	Minimum dominant width of break field detection.	u1gBLT_SHORT	[Auto baud rate mode] 10[Tbit] [Fixed baud rate mode] 9.5[Tbit]
		u1gBLT_LONG	[Auto baud rate mode] 11[Tbit] [Fixed baud rate mode] 10.5[Tbit]
u1gRS_CFG	Response space width at response Tx.	u1gRS_0BIT	0[Tbit]
		u1gRS_1BIT	1[Tbit]
		u1gRS_2BIT	2[Tbit]
		u1gRS_3BIT	3[Tbit]
		u1gRS_4BIT	4[Tbit]
		u1gRS_5BIT	5[Tbit]
		u1gRS_6BIT	6[Tbit]
		u1gRS_7BIT	7[Tbit]
u1gIBS_CFG	Inter byte space width at response Tx.	u1gIBS_0BIT	0[Tbit]
		u1gIBS_1BIT	1[Tbit]
		u1gIBS_2BIT	2[Tbit]
		u1gIBS_3BIT	3[Tbit]
u1gWUTL_CFG	Tx wakeup pulse width.	u1gWUTL_1BIT	1[Tbit]
		u1gWUTL_2BIT	2[Tbit]
		u1gWUTL_3BIT	3[Tbit]
		u1gWUTL_4BIT	4[Tbit]
		u1gWUTL_5BIT	5[Tbit]
		u1gWUTL_6BIT	6[Tbit]
		u1gWUTL_7BIT	7[Tbit]
		u1gWUTL_8BIT	8[Tbit]
		u1gWUTL_9BIT	9[Tbit]
		u1gWUTL_10BIT	10[Tbit]
		u1gWUTL_11BIT	11[Tbit]
		u1gWUTL_12BIT	12[Tbit]
		u1gWUTL_13BIT	13[Tbit]
		u1gWUTL_14BIT	14[Tbit]
		u1gWUTL_15BIT	15[Tbit]
		u1gWUTL_16BIT	16[Tbit]
u1gBERE_CFG	Switch of bit error detection.	u1gBERE_DISABLE	Not detect
		u1gBERE_ENABLE	Detect
u1gTER_CFG	Switch of timeout error detection. *1	u1gTER_DISABLE	Not detect
		u1gTER_ENABLE	Detect (can't be used in auto baud rate mode)
u1gFERE_CFG	Switch of framing error detection.	u1gFERE_DISABLE	Not detect
		u1gFERE_ENABLE	Detect
u1gSFERE_CFG	Switch of synch	u1gSFERE_DISABLE	Not detect

	field error detection.	u1gSFERE_ENABLE	Detect
u1gIPERE_CFG	Switch of IP parity error detection.	u1gIPERE_DISABLE	Not detect
		u1gIPERE_ENABLE	Detect
u1gLTES_CFG	Switch of the type of timeout error detection. If timeout error detection is not used, this value is ignored.	u1gLTES_FRAMETO	Frame timeout
		u1gLTES_RESPTO	Response timeout
u4gTRWTCNTMAX_CFG	Maximum count value of waiting the transition of RLIN3. If RLIN3 does not transit next state within this value, LIN driver will stop LIN communication.	Greater than or equal to 1	-
u2gTMCLKSEL_CFG	Clock selection of interval timer (TAU).	u2gTMCLKSEL_SELO	Clock selection 0
		u2gTMCLKSEL_SEL1	Clock selection 1
u2gTMCLKDIV_CFG	Division value of interval timer (TAU).	u2gTMCLKDIV_NODIV	1/1
		u2gTMCLKDIV_DIV2	1/2
		u2gTMCLKDIV_DIV2P2	1/4
		u2gTMCLKDIV_DIV2P3	1/8
		u2gTMCLKDIV_DIV2P4	1/16
		u2gTMCLKDIV_DIV2P5	1/32
		u2gTMCLKDIV_DIV2P6	1/64
		u2gTMCLKDIV_DIV2P7	1/128
		u2gTMCLKDIV_DIV2P8	1/256
		u2gTMCLKDIV_DIV2P9	1/512
		u2gTMCLKDIV_DIV2P10	1/1024
		u2gTMCLKDIV_DIV2P11	1/2048
		u2gTMCLKDIV_DIV2P12	1/4096
		u2gTMCLKDIV_DIV2P13	1/8192
		u2gTMCLKDIV_DIV2P14	1/16384
		u2gTMCLKDIV_DIV2P15	1/32768

*1 : Please refer to *5 of "Table 6-9. Driver configuration of RL78/F23,F24 slave (1 of 2)".

7.9.2 Master driver configuration

On RL78/F23,F24 master driver, user can modify behavior of LIN driver using driver configuration. Driver configuration is described in part “Driver configuration” in libsrc/conf/confmlin_opt.h file.

Table 7-11. Driver configuration of RL78/F23,F24 master (1 of 2)

Configuration points ("CONFMLIN_OPT" is omitted)	Contents	Setting values	Meaning of setting values
u1gBUSWKUP_CFG *3	The wakeup method of LIN driver. If edge is selected, LIN macro does not need clock in sleep mode.	u1gBUSWKUP_EDGE	The method of wakeup is down edge detection. (Operation clock isn't supplied to LIN macros when LIN driver is in sleep mode.)
		u1gBUSWKUP_WIDTH	The method of wakeup is dominant width detection.
u1gLINMCK_CFG	Source of LIN communication clock. *2	u1gLINMCK_FCLK	f _{CLK}
		u1gLINMCK_FMX	f _{MX}
u1gBDT_CFG	Sending break delimiter length.	u1gBDT_1BIT	1 [bit]
		u1gBDT_2BIT	2 [bit]
		u1gBDT_3BIT	3 [bit]
		u1gBDT_4BIT	4 [bit]
u1gIBHS_CFG	Width between sending synch field and id field, and width of sending response space. *1	u1gIBHS_0BIT	0 [bit]
		u1gIBHS_1BIT	1 [bit]
		u1gIBHS_2BIT	2 [bit]
		u1gIBHS_3BIT	3 [bit]
		u1gIBHS_4BIT	4 [bit]
		u1gIBHS_5BIT	5 [bit]
		u1gIBHS_6BIT	6 [bit]
		u1gIBHS_7BIT	7 [bit]
u1gIBS_CFG	Width between each sending response. *1	u1gIBS_0BIT	0 [bit]
		u1gIBS_1BIT	1 [bit]
		u1gIBS_2BIT	2 [bit]
		u1gIBS_3BIT	3 [bit]
u1gINTLINRMPR_CFG *3	Priority of LIN transmission completion interruption. This is used at sending go-to-sleep command and wakeup request only.	u1gINTPR_LV0	Level 0 (Highest)
		u1gINTPR_LV1	Level 1
		u1gINTPR_LV2	Level 2
		u1gINTPR_LV3	Level 3 (Lowest)
u1gINTLINRVCPR_CFG *3	Priority of LIN reception completion interruption. This is used at only receiving wakeup request when bus wakeup way is dominant width detection on LIN bus.	Same as u1gINTLINRMPR_CFG.	—
u1gINTLINSTAPR_CFG *3	Priority of LIN status interruption. This is used at only sending go-to-sleep or wakeup request.	Same as u1gINTLINRMPR_CFG.	—

u1gINTPPR_CFG *3	Priority of external interruption. This is used at only receiving wakeup request when bus wakeup way is dominant width detection on LIN bus.	Same as u1gINTLINTRMPR_CFG.	—
------------------	---	-----------------------------	---

*1: Timeout error is detected when whole time of 1 frame exceeds following time and frame timeout error is enabled. Set these values as not exceeding.

[Frame is classic checksum] : $49 + (\text{number of response bytes} + 1) \times 14$ [bit]

[Frame is enhanced checksum] : $48 + (\text{number of response bytes} + 1) \times 14$ [bit]

Timeout error is detected when whole time of response exceeds following time and response timeout error is enables. Set these values as not exceeding.

$(\text{number of response bytes} + 1) \times 14$ [bit]

*2 : There is a notice regarding the LIN communication clock source in LIN/UART module (RLIN3) of RL78/F23,F24. For details refer to the user's manual for RL78/F23 and F24. The master driver includes the following restriction for by this case.

- When setting f_{MX} (u1gLINMCK_FMX) as type of a LIN communication clock source (u1gINTLINTRMPR_CFG) and setting "Not detect time out error" (u1gTER_DISABLE) as "Time out error detection" (u1gTER_CFG), please set f_{CLK} clock more than 1.2 times of LIN communication clock source.

*3 : Configurable by the smart configurator.

Table 7-12. Driver configuration of RL78/F23,F24 master (2 of 2)

Configuration points ("CONFMLIN_OPT" is omitted)	Contents	Setting values	Meaning of setting values
u1gLRDNFS_NORM_CFG	Selection of using LIN rx noise filter.	u1gLRDNFS_USE	Use
		u1gLRDNFS_NOUSE	Not use
u1gWUTL_CFG	Width of sending wakeup request. *1	u1gWUTL_1BIT	1 [bit]
		u1gWUTL_2BIT	2 [bit]
		u1gWUTL_3BIT	3 [bit]
		u1gWUTL_4BIT	4 [bit]
		u1gWUTL_5BIT	5 [bit]
		u1gWUTL_6BIT	6 [bit]
		u1gWUTL_7BIT	7 [bit]
		u1gWUTL_8BIT	8 [bit]
		u1gWUTL_9BIT	9 [bit]
		u1gWUTL_10BIT	10 [bit]
		u1gWUTL_11BIT	11 [bit]
		u1gWUTL_12BIT	12 [bit]
		u1gWUTL_13BIT	13 [bit]
		u1gWUTL_14BIT	14 [bit]
		u1gWUTL_15BIT	15 [bit]
		u1gWUTL_16BIT	16 [bit]
u1gBERE_CFG	Selection of using bit error detection.	u1gBERE_DISABLE	Not detect
		u1gBERE_ENABLE	Detect
u1gPBERE_CFG	Selection of using physical bus error detection.	u1gBERE_DISABLE	Not detect
		u1gBERE_ENABLE	Detect
u1gTER_CFG	Selection of using timeout error detection.	u1gTER_DISABLE	Not detect
		u1gTER_ENABLE	Detect
u1gFEER_CFG	Selection of using framing error detection.	u1gFERE_DISABLE	Not detect
		u1gFERE_ENABLE	Detect
u1gLTES_CFG	Type of timeout error. This is no influence if timeout error is not enabled. *2	u1gLTES_FRAMETO	Frame timeout
		u1gLTES_RESPTO	Response timeout
u4gTRWTCNTMAX_CFG	Waiting time (software counter) of RLIN3 transition. If RLIN3 doesn't complete transition exceeding this value, LIN driver notifies error to application.	1 or more.	–
u1gRXD_PU_CFG	Selection of using internal pull up of LIN rx pin.	u1gRXD_PU_DISABLE	Not use
		u1gRXD_PU_ENABLE	Use
u1gRXD_PITHL_CFG	Input level of LIN rx pin.	u1gRXD_PITHL_03VDD	0.3 EV _{DD}
		u1gRXD_PITHL_05VDD	0.5 EV _{DD}

*1 : Don't set 6 bit or less because wakeup request low width may not satisfy LIN 2.1 specification.

*2 : Please refer to *2 of "Table 6-11. Driver configuration of RL78/F23, F24 master (1 of 2)".

CHAPTER 8 LIN 2.1 SOFTWARE DRIVER FUNCTION (SLAVE)

8.1 LIN 2.1 Software Slave Driver Function List

LIN 2.1 Software driver function list is as follows.

Table 8-1. LIN 2.1 Software Slave Driver Function

Category	Function Name	Description
LIN 2.1 Software Driver and Cluster Management	<code>l_sys_init</code>	System initialization
Scalar Signal Read	<code>l_bool_rd</code>	Scalar signal read process of 1 bit
	<code>l_u8_rd</code>	Scalar signal read process of 8 bits
	<code>l_u16_rd</code>	Scalar signal read process of 16 bits
Scalar Signal Write	<code>l_bool_wr</code>	Scalar signal write process of 1 bit
	<code>l_u8_wr</code>	Scalar signal write process of 8 bits
	<code>l_u16_wr</code>	Scalar signal write process of 16 bits
Byte Array Read	<code>l_bytes_rd</code>	Byte array read process
Byte Array Write	<code>l_bytes_wr</code>	Byte array write process
Notification	<code>l_flg_tst</code>	Signal update flag confirmation
	<code>l_flg_clr</code>	Signal update flag clear
Interface Management	<code>l_ifc_init</code>	Interface initialization
	<code>l_ifc_wake_up</code>	Wake up issue
	<code>l_ifc_read_status</code>	Status reading
User provided call-outs	<code>l_sys_irq_disable</code>	Interrupt prohibition setting
	<code>l_sys_irq_restore</code>	Interrupt reintegration setting
	<code>l_sys_call_sleep</code>	Sleep state transition
	<code>l_sys_call_wake_up</code>	Wake up start

8.2 Data types (Slave)

The type defined by LIN 2.1 API and the type that the LIN 2.1 slave driver uses are the following definitions.

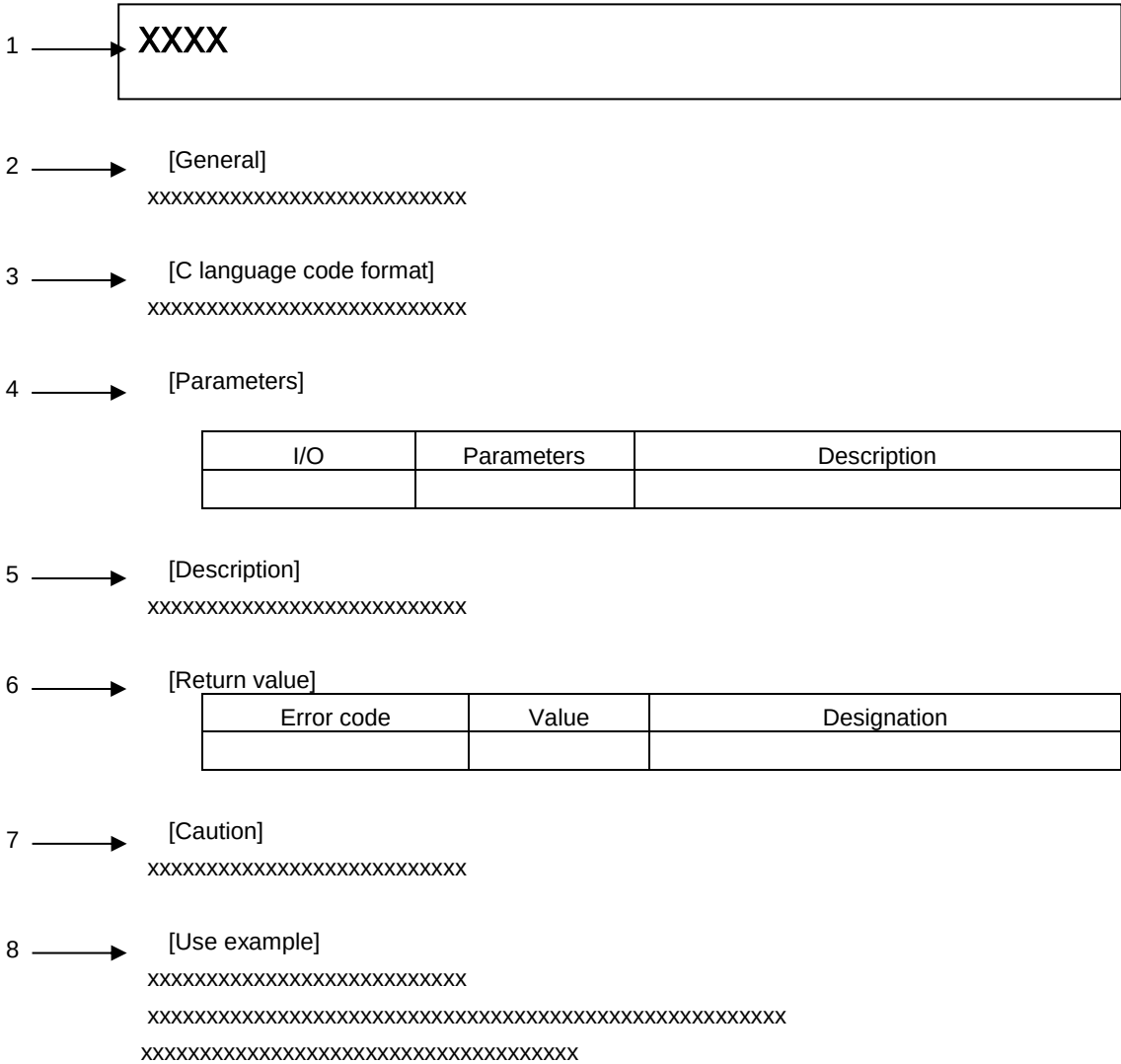
Table 8-2. LIN 2.1 Spec and LIN 2.1 Slave Driver Type Definition List

Type of LIN 2.1 Spec	Type of LIN 2.1 Driver
<code>l_bool</code>	unsigned char
<code>l_u8</code>	unsigned char
<code>l_u16</code>	unsigned short
<code>l_signal_handle</code>	unsigned char
<code>l_flag_handle</code>	unsigned char
<code>l_irqmask</code>	unsigned char
<code>l_ifc_handle</code>	unsigned char

8. 3 Description of LIN 2.1 Software Slave Driver Function

It explains the LIN 2.1 software slave driver function according to the following forms.

Figure 8-1. Description Format of LIN 2.1 Software Slave Driver Function



1. Name

This indicates the name of the LIN 2.1 software driver function.

2. [General]

This indicates each LIN 2.1 software driver function's general functions.

3. [C language code format]

This indicates the code format used to issue LIN 2.1 software driver function is C language.

4. [Parameters]

LIN 2.1 software driver function parameters are indicated in the following format.

I/O	Parameters	Description
A	B	C

A: Parameter I/O classification

I ... Input parameter

O ... Output parameter

B: Parameter type and name

C: Description of parameter

5. [Description]

This describes the functions of each LIN 2.1 software driver function.

6. [Return value]

This indicates return value of each LIN 2.1 software driver function. The format is as follows.

Error code	Value	Designation
A	B	C

A: The name when return value is error code.

B: Range of return value.

C: Description of return value.

7. [Caution]

This indicates cautions concerning LIN 2.1 software driver function. In particular, device-dependent cautions are explained.

8. [Use example]

This provides use examples for specific LIN 2.1 software driver functions.

8. 3. 1 [Slave] LIN 2.1 Software Driver and Cluster Management

The LIN 2.1 software driver function is described from next page.

Table 8-3. LIN 2.1 Software Driver and Cluster Management (Slave)

Function Name	Description
<code>l_sys_init</code>	System initialization

I_sys_init

[General]

The LIN 2.1 software driver is initialized.

[C language code format]

```
I_bool I_sys_init(void)
```

[Parameters]

None

[Description]

This function initializes the LIN 2.1 software driver system. It is necessary to call it before all function is called.

[Return value]

Error code	Value	Designation
L_SUCCESS	0x00	Initialize is succeed

* For LIN software driver, only L_SUCCESS is returned.

Recommend the return value check for safety.

[Use example]

```
/* after start up finished */
if ( I_sys_init() ){
    /* init error */
    ;
}
else{
    /* after this, other function can be called. */
}
```

8. 3. 2 [Slave] Scalar Signal Read

The functions of scalar signal read are described from next page.

Table 8-4. Scalar Signal Read (Slave)

Function Name	Description
I_bool_rd	Scalar signal read process of 1 bit
I_u8_rd	Scalar signal read process of 8 bits
I_u16_rd	Scalar signal read process of 16 bits

I_bool_rd

[General]

The 1 bit scalar signal is read.

[C language code format]

```
I_bool I_bool_rd(I_signal_handle sss)
```

[Parameters]

I/O	Parameter	Description
I	I_signal_handle sss	Signal name (macro value)

[Description]

This function reads the signal data in the message buffer that related to the specified signal name. It is possible to read it asynchronously with the LIN communication.

[Return value]

Error code	Value	Designation
-	0x00,0x01	Signal data

[Caution]

- Do not read the signal defined by the sizes other than 1 bit.
- Do not specify "sss" excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the transmission.

[Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

- ```
- Signal name : WINDOW_SWITCH
- Size : 1 bit
```

```
*/
```

```
if(I_bool_rd(WINDOW_SWITCH))
{
 "Processing for window switch ON"
}
else
{
 "Processing for window switch OFF"
}
```

## I\_u8\_rd

### [General]

The scalar signal from 1 to 8 bits is read.

### [C language code format]

```
I_u8 I_u8_rd(I_signal_handle sss)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |

### [Description]

This function reads the signal data in the message buffer that related to the specified signal name. It is possible to read it asynchronously with the LIN communication.

The current signal data is read regardless of the presence of new data.

### [Return value]

| Error code | Value  | Designation |
|------------|--------|-------------|
| -          | 0-0xFF | Signal data |

### [Caution]

- Do not read the signal defined by a size that is larger than 8 bits.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_STATUS

- Size : 8 bits

\*/

#define ST\_BUSY (0x00)

#define ST\_IDLE (0x01)

#define ST\_NG (0x02)

switch\_status = l\_u8\_rd(WINDOW\_STATUS);

if( switch\_status == ST\_BUSY )

{

    "Processing for BUSY state"

}

else if( switch\_status == ST\_NG )

{

    "Processing for NG state"

}

## I\_u16\_rd

### [General]

The scalar signal from 1 to 16 bits is read.

### [C language code format]

```
I_u16 I_u16_rd(I_signal_handle sss)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |

### [Description]

This function reads the signal data in the message buffer that related to the specified signal name. It is possible to read it asynchronously with the LIN communication.

The current signal data is read regardless of the presence of new data.

### [Return value]

| Error code | Value    | Designation |
|------------|----------|-------------|
| -          | 0-0xFFFF | Signal data |

### [Caution]

- Do not read the signal defined by the sizes larger than 16 bits.
- Do not specify "sss" excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the transmission.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_STATUS

- Size : 16 bits

\*/

#define ST\_BUSY 0x0000

#define ST\_IDLE 0x0001

l\_u16 switch\_status;

switch\_status = l\_u16\_rd(WINDOW\_STATUS);

switch( switch\_status )

{

case ST\_BUSY:

/\* Processing for BUSY state \*/

break;

case ST\_IDLE:

/\* Processing for IDLE state \*/

break;

default:

/\* Processing for error state \*/

break;

}

### 8. 3. 3 [Slave] Scalar Signal Write

The functions of scalar signal write are described from next page.

**Table 8-5. Scalar Signal Write (Slave)**

| Function Name | Description                            |
|---------------|----------------------------------------|
| I_bool_wr     | Scalar signal write process of 1 bit   |
| I_u8_wr       | Scalar signal write process of 8 bits  |
| I_u16_wr      | Scalar signal write process of 16 bits |

## I\_bool\_wr

### [General]

The 1 bit scalar signal is written.

### [C language code format]

```
void I_bool_wr(I_signal_handle sss, I_bool v)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |
| I   | I_bool v            | Writing data              |

### [Description]

This function writes the data specified by “v” in the message buffer that related to the specified signal name.  
After writing, the update flag of the corresponding signal is set.

### [Return value]

None

### [Caution]

- Do not write the signal defined by the sizes larger than 1 bit.
- Do not specify the signal in the message buffer that setting as the reception.
- Do not specify “sss” excluding the defied signal.

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW, WINDOW\_SWITCH
- Size : 1 bit

\*/

```
#define PUSH (0x01)
```

```
#define OPEN (0x00)
```

/\* Switch push is detected \*/

```
if(I_bool_rd(WINDOW_SWITCH) == PUSH)
```

```
{
```

```
 /* 1 bit signal OPEN is written */
```

```
 I_bool_wr(WINDOW, OPEN);
```

```
}
```

## I\_u8\_wr

### [General]

The scalar signal from 1 to 8 bits is written.

### [C language code format]

```
void I_u8_wr(l_signal_handle sss, I_u8 v)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |
| I   | I_u8 v              | Writing data              |

### [Description]

This function writes the data specified by “v” in the message buffer that related to the specified signal name.

After writing, the update flag of the corresponding signal is set.

### [Return value]

None

### [Caution]

- Do not write the signal defined by the sizes larger than 8 bits.
- Do not specify the signal in the message buffer that setting as the reception.
- Do not specify “sss” excluding the defied signal.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW, WINDOW\_SWITCH

- Size : 8 bits

\*/

#define OPEN (0x00)

#define CLOSE(0x01)

#define SLEEP (0x02)

#define UP (0x00)

#define DOWN (0x01)

switch(l\_u8\_rd(WINDOW\_SWITCH))

{

case UP:

l\_u8\_wr( WINDOW, OPEN );

break;

case DOWN:

l\_u8\_wr( WINDOW, CLOSE );

break;

default:

l\_u8\_wr( WINDOW, SLEEP);

break;

}

## I\_u16\_wr

### [General]

The scalar signal from 1 to 16 bits is written.

### [C language code format]

```
void I_u16_wr(I_signal_handle sss, I_u16 v)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |
| I   | I_u16 v             | Writing data              |

### [Description]

This function writes the data specified by “v” in the message buffer that related to the specified signal name.  
After writing, the update flag of the corresponding signal is set.

### [Return value]

None

### [Caution]

- Do not write the signal defined by the sizes larger than 16 bits.
- Do not specify the signal in the message buffer that setting as the reception.
- Do not specify “sss” excluding the defied signal.

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : VOLUME

- Size : 12 bits

\*/

```
I_u16 vol;
```

```
/* The volume level is acquired */
```

```
vol =GetVolLevel();
```

```
/* Set in signal */
```

```
I_u16_wr(VOLUME, vol);
```

### 8. 3. 4 [Slave] Byte Array Read

The functions of byte array read are described from next page.

**Table 8-6. Byte Array Read (Slave)**

| Function Name | Description             |
|---------------|-------------------------|
| l_bytes_rd    | Byte array read process |

## I\_bytes\_rd

### [General]

The byte array data is read.

### [C language code format]

```
void I_bytes_rd(I_signal_handle sss, I_u8 start, I_u8 count, I_u8* const data)
```

### [Parameters]

| I/O | Parameter           | Description                     |
|-----|---------------------|---------------------------------|
| I   | I_signal_handle sss | Signal name (macro value)       |
| I   | I_u8 start          | Reading beginning byte number   |
| I   | I_u8 count          | Reading number of bytes         |
| O   | I_u8* const data    | Stored location of reading data |

### [Description]

This function reads byte data for a number of counts from the start byte in the message buffer related to the specified signal name.

For instance, for the message of 7 bytes longs (number is 0 to 6) in byte array, to read the data from 3rd to 4th, 3 is set in start and 2 is set in count. In this case, the 3rd value is written in data [0] and the 4th value is written in data [1].

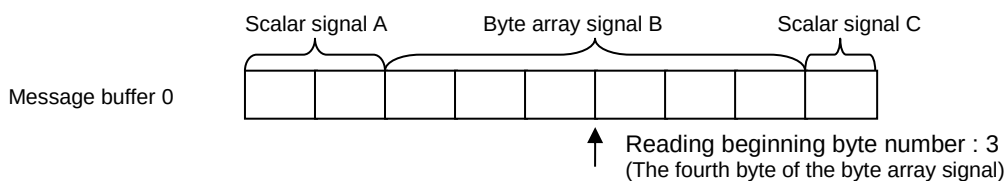
### [Return value]

None

### [Caution]

- Do not specification that the signal size becomes outside the range by combining start and count.  
A correct value cannot be read.
- Only the signal defined as byte array can be reading (Signal that offset and size become multiple of 8).  
When specifying by other definitions, a correct value cannot be read.
- Do not specify the signal in the message buffer that setting as the transmission.
- Reading beginning byte number is not number from head of message buffer, but number from head of signal name. Refer to the following example.

Ex. When you want to read it from the fourth byte of byte array



[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_TEST

- Size : 4 bytes

\*/

l\_u8 data[4];

/\* the 4 bytes data stored in data array \*/

l\_bytes\_rd(WINDOW\_TEST, 0, 4, data)

/\* the 2 bytes data from 2nd to 3rd stored in data array\*/

l\_bytes\_rd(WINDOW\_TEST, 2, 2, data);

### 8. 3. 5 [Slave] Byte Array Write

The functions of byte array write are described from next page.

**Table 8-7. Byte Array Write (Slave)**

| Function Name           | Description              |
|-------------------------|--------------------------|
| <code>I_bytes_wr</code> | Byte array write process |

## l\_bytes\_wr

### [General]

The byte array data is written.

### [C language code format]

```
void l_bytes_wr(l_signal_handle sss, l_u8 start, l_u8 count, const l_u8* const data)
```

### [Parameters]

| I/O | Parameter                           | Description                     |
|-----|-------------------------------------|---------------------------------|
| I   | <code>l_signal_handle sss</code>    | Signal name (macro value)       |
| I   | <code>l_u8 start</code>             | Writing beginning byte number   |
| I   | <code>l_u8 count</code>             | Writing number of bytes         |
| I   | <code>const l_u8* const data</code> | Stored location of writing data |

### [Description]

This function writes byte data for a number of counts from the start byte in the message buffer related to the specified signal name.

For instance, for the message of 7 bytes longs (number is 0 to 6) in byte array, to write the data from 3rd to 4th, 3 is set in start and 2 is set in count. In this case, the 3rd value is read from data [0] and the 4th value is read from data [1].

After writing, the update flag of the corresponding signal is set.

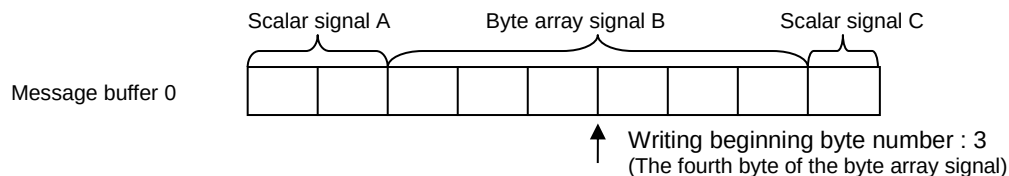
### [Return value]

None

### [Caution]

- Do not specification that the signal size becomes outside the range by combining start and count.  
A correct value cannot be read.
- Only the signal defined as byte array can be reading (Signal that offset and size become multiple of 8).  
When specifying by other definitions, a correct value cannot be read.
- Do not specify the signal in the message buffer that setting as the reception.
- Writing beginning byte number is not number from head of message buffer, but number from head of signal name. Refer to the following example.

Ex. When you want to write it from the fourth byte of byte array



[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_TEST

- Size : 4 bytes

\*/

const l\_u8 data[4] = {0x00, 0x01, 0x02, 0x03};

/\* the 4 bytes data of data array is written \*/

l\_bytes\_wr(WINDOW\_TEST, 0, 4, data);

/\* the 2 bytes data is written in WINDOW\_TEST signal \*/

l\_bytes\_wr(WINDOW\_TEST, 2, 2, data);

### 8. 3. 6 [Slave] Notification

The functions of notification are described from next page.

**Table 8-8. Notification (Slave)**

| Function Name | Description                     |
|---------------|---------------------------------|
| I_flg_tst     | Signal update flag confirmation |
| I_flg_clr     | Signal update flag clear        |

## I\_flg\_tst

### [General]

The value of signal update flag is read.

### [C language code format]

```
I_bool I_flg_tst(I_flag_handle fff)
```

### [Parameters]

| I/O | Parameter         | Description                     |
|-----|-------------------|---------------------------------|
| I   | I_flag_handle fff | Flag name (same as signal name) |

### [Description]

This function reads the signal update flag related to the specified flag name.

Timing that the flag updated is as follows.

- When the reception successful completion.
- When call the scalar signal writing function and byte array writing function.

### [Return value]

| Error code | Value | Designation |
|------------|-------|-------------|
| -          | 0x00  | Not update  |
| -          | 0x01  | Update      |

### [Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

- ```
- Signal name      : WINDOW_STATUS
- Size             : 1 byte
```

```
*/
```

```
I_bool status;
```

```
if( I_flg_tst( WINDOW_STATUS ) )
```

```
{
```

```
    /* processing when WINDOW_STATUS is updated is described */
```

```
    status = I_bool_rd( WINDOW_STATUS );
```

```
}
```

I_flg_clr

[General]

The signal update flag is cleared by 0.

[C language code format]

```
void I_flg_clr(I_flg_handle fff)
```

[Parameters]

I/O	Parameter	Description
I	I_flg_handle fff	Flag name (same as macro value and signal name)

[Description]

This function clear the signal update flag related to the specified flag name.

Timing that the flag is cleared is as follows.

- When issue this function
- When the transmission is successful completed.

[Return value]

None

[Use example]

/* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW_TEST
- Size : 1 byte

*/

```
I_u8 status;
```

```
if( I_flg_tst(WINDOW_TEST) )  
{  
    status = I_u8_rd(WINDOW_TEST);
```

```
    /* the update flag is cleared */
```

```
    I_flg_clr(WINDOW_TEST);
```

```
}
```

8. 3. 7 [Slave] Interface Management

The functions of interface management are described from next page.

Table 8-9. Interface Management (Slave)

Function Name	Description
<code>l_ifc_init</code>	Interface initialization
<code>l_ifc_wake_up</code>	Wake up issue
<code>l_ifc_read_status</code>	Status reading

I_ifc_init

[General]

The LIN interface is initialized.

[C language code format]

```
I_bool I_ifc_init (I_ifc_handle iii)
```

[Parameters]

I/O	Parameter	Description
I	I_ifc_handle iii	Interface name (channel number)

[Description]

This function executes the initialization that relates to the specified LIN interface and the LIN communication in the specified interface is enabled.

The permission processing of the LIN transfer is executed.

[Return value]

Error code	Value	Designation
L_FAIL	0xFF	Failed
L_SUCCESS	0x00	Succeeded

[Caution]

- Use this function after issuing the I_sys_init function.
- Issue this function before using the LIN 2.1 software driver functions other than I_sys_init function.
- Do not specify the interface name excluding "LIN_CHANNEL0" if don't use multi-channel function.

[Use example]

/* beforehand, it changes to the following settings with the configuration file.

- Channel name: LIN_CHANNEL0

*/

```
if( I_sys_init( ) )
{
    /* error processing is described */
}
else
{
    /* interface initialization */
    if( I_ifc_init( LIN_CHANNEL0 ) )
    {
        /* error processing is described */
    }
}
```

I_ifc_wake_up

[General]

Wake up is issued.

[C language code format]

```
void I_ifc_wake_up(I_ifc_handle iii)
```

[Parameters]

I/O	Parameter	Description
I	I_ifc_handle iii	Interface name (channel number)

[Description]

This function transitions the specified LIN interface from sleep mode to wake-up and sends a 260µs wake-up Low pulse to the LIN bus.

After three wake-up pulses, the fourth wake-up pulse is sent after 1.5 sec. Three wake-up pulses are a block, and this is repeated a user-defined number of times. Three transmissions are a block, and this is repeated a user-defined number of times. After that, if the bus is inactive for more than 4 seconds, it will enter the sleep mode.

[Return value]

None

[Caution]

- It is possible to issue it only for sleep mode.
- Do not specify the interface name excluding "LIN_CHANNEL0" if don't use multi-channel function.

[Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

- Channel name: LIN_CHANNEL0
- Status mask bit: LD_MASK_SLEEP

```
*/
```

```
/* when sleep mode is detected, wake up immediately */
```

```
/*sleep detection */
```

```
if( (I_ifc_read_status(LIN_CHANNEL0) & LD_MASK_SLEEP) == LD_MASK_SLEEP)
{
    I_ifc_wake_up();
}
```

I_ifc_read_status

[General]

LIN software driver's various statuses are read.

[C language code format]

```
I_u16 I_ifc_read_status(I_ifc_handle iii)
```

[Parameters]

I/O	Parameter	Description
I	I_ifc_handle iii	Interface name (channel number)

[Description]

This function reads status information in the specified LIN interface.

[Return value]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit0
Last frame PID								0	Save configuration	Event triggered frame collision	Bus activity	Goto sleep	Overrun	Successful transfer	Error in response

Last frame PID : PID of the last communication frame is shown.

Save configuration : Not supported.

Event triggered frame collision: Not a relevant bit in the slave driver.

Bus activity : When a rising or falling edge of the bus is detected, it is set.

Goto sleep : When go-to-sleep-command is normally transmitted, it is set.

Overrun : When over 2 transfers are executed before I_ifc_read_status is called, it is set.

Successful transfer : When the transfer I succeeded, it is set.

Error in response : When an illegal transfer is detected while response, it is set.

[Caution]

- Use this function after issuing the I_ifc_init function.
- After this function or I_ifc_init function is issued, status cleared by 0.
- Status is an accumulation flag. When two or more transfer is done from the call of previous this function, status is overwritten.
- Do not specify the interface name excluding "LIN_CHANNEL0" if don't use multi-channel function.

[Use example]

/* beforehand, it changes to the following settings with the configuration file.

- Channel name: LIN_CHANNEL0

- Status mask bit: LD_MASK_ERROR_IN_RESPONSE, LD_MASK_SUCCESSFUL_TRANSFER

*/

```
status = l_ifc_read_status( LIN_CHANNEL0 );
```

```
mask = LD_MASK_ERROR_IN_RESPONSE | LD_MASK_SUCCESSFUL_TRANSFER;
```

```
if( status & mask ) == mask )
```

```
{
```

```
    /* Detection error and succeeded, in this case, it is though that the transfer is intermittently done. */
```

```
}
```

8.3.8 [Slave] User provided call-outs

The functions of call-outs are described from next page.

Table 8-10. User provided call-outs (Slave)

Function Name	Description
I_sys_irq_disable	Interrupt prohibition setting
I_sys_irq_restore	Interrupt reintegration setting
I_sys_call_sleep	Sleep state transition
I_sys_call_wake_up	Wake up start

I_sys_irq_disable

[General]

The interrupt is prohibited.

[C language code format]

```
I_irqmask I_sys_irq_disable(void)
```

[Parameters]

None

[Description]

This function is a call-outs function to prohibit interrupt.

DI is executed, and current interrupt is returned. The return value is the PSW value for the CC-RL edition, and the value returned by the IAR intrinsic function `__get_interrupt_state` for the IAR edition.

[Return value]

I/O	Value	Designation
-	[CC-RL] Interrupt status bit of PSW [IAR] Global interrupt state	-

[Caution]

- Because of this is call-outs function, it is possible to change, but you don't need to modify the Smart Configurator code. (See "5. 2. 7 User-defined callouts implementation")

[Use example]

- It is automatically called. It is not necessary to use it in the application.

I_sys_irq_restore

[General]

It returns the state of the interrupt.

[C language code format]

```
void I_sys_irq_restore(I_irqmask previous)
```

[Parameters]

I/O	Parameter	Description
I	I_irqmask previous	Return value of last I_sys_irq_disable(). ([CC-RL] Interrupt status bit of PSW, [IAR] Global interrupt state)

[Description]

This function is a call-outs function to return the interrupt state.

In default, it set to EI when "previous" is interrupt enable ("0x80"), and besides, it set to DI for the CC-RL edition. For the IAR edition, it restores the interrupt state specified by "previous".

[Return value]

None

[Caution]

- Because of this is call-outs function, it is possible to change, but you don't need to modify the Smart Configurator code. (See "5. 2. 7 User-defined callouts implementation")

[Use example]

- It is automatically called. It is not necessary to use it in the application.

I_sys_call_sleep

[General]

It is called when the state changed to sleep mode.

It is original enhanced function of this driver.

[C language code format]

```
void I_sys_call_sleep(I_ifc_handle iii)
```

[Parameters]

I/O	Parameter	Description
I	I_ifc_handle iii	Interface name (channel number)

[Description]

This function is call-out function when the sleep request is received or it change to sleep mode by the timeout.

[Return value]

None

[Caution]

- There is not change to sleep mode immediately after the first byte of the data field of sleep command is received. It is not change to sleep mode without receive checksum.
- This function need not necessarily be mounted. When not using it, make it empty. In the code provided by the Smart Configurator, it is an empty function. (See "5. 2. 7 User-defined callouts implementation")

[Use example]

- It is automatically called. It is not necessary to use it in the application.

I_sys_call_wake_up

[General]

It is called immediately after reception of wake up.

It is original enhanced function of this driver.

[C language code format]

```
void I_sys_call_wake_up(I_ifc_handle iii)
```

[Parameters]

I/O	Parameter	Description
I	I_ifc_handle iii	Interface name (channel number)

[Description]

This function is call-out function when the LIN 2.1 software driver receives wake up.

[Return value]

None

[Caution]

- This function need not necessarily be mounted. When not using it, make it empty. In the code provided by the smart configurator, it is an empty function. (See "5. 2. 7 User-defined callouts implementation")

[Use example]

- It is automatically called. It is not necessary to use it in the application.

CHAPTER 9 LIN 2.1 SOFTWARE DRIVER FUNCTION (MASTER)

9.1 LIN 2.1 Software Master Driver Function List

LIN 2.1 Software Driver Function List is as follows.

Table 9-1. List of LIN 2.1 Software Master Driver Function

Category	Function Name	Description
LIN 2.1 Software Driver and Cluster Management	<code>l_sys_init</code>	System initialization
Scalar Signal Read	<code>l_bool_rd</code>	Scalar signal read process of 1 bit
	<code>l_u8_rd</code>	Scalar signal read process of 8 bits
	<code>l_u16_rd</code>	Scalar signal read process of 16 bits
Scalar Signal Write	<code>l_bool_wr</code>	Scalar signal write process of 1 bit
	<code>l_u8_wr</code>	Scalar signal write process of 8 bits
	<code>l_u16_wr</code>	Scalar signal write process of 16 bits
Byte Array Read	<code>l_bytes_rd</code>	Byte array read process
Byte Array Write	<code>l_bytes_wr</code>	Byte array write process
Notification	<code>l_flg_tst</code>	Signal update flag confirmation
	<code>l_flg_clr</code>	Signal update flag clear
Interface Management	<code>l_ifc_init</code>	Interface initialization
	<code>l_ifc_goto_sleep</code>	go-to-sleep-command issue
	<code>l_ifc_wake_up</code>	Wake up issue
	<code>l_ifc_read_status</code>	Status reading
Schedule Management	<code>l_sch_tick</code>	Schedule control
	<code>l_sch_set</code>	Schedule setting
Node Configuration	<code>ld_is_ready</code>	Node configuration ready
	<code>ld_check_response</code>	Node configuration check
	<code>ld_assign_frame_id_range</code>	PID allocation
	<code>ld_read_by_id</code>	ID reading
User provided call-outs	<code>l_sys_irq_disable</code>	Interrupt prohibition setting
	<code>l_sys_irq_restore</code>	Interrupt reintegration setting
	<code>l_sys_call_wake_up</code>	Wake up start
	<code>l_sys_call_fatal_error</code>	Resolve fatal error

9.2 Data types (Master)

The type defined by LIN 2.1 spec and the type that the LIN 2.1 master driver uses are the following definitions.

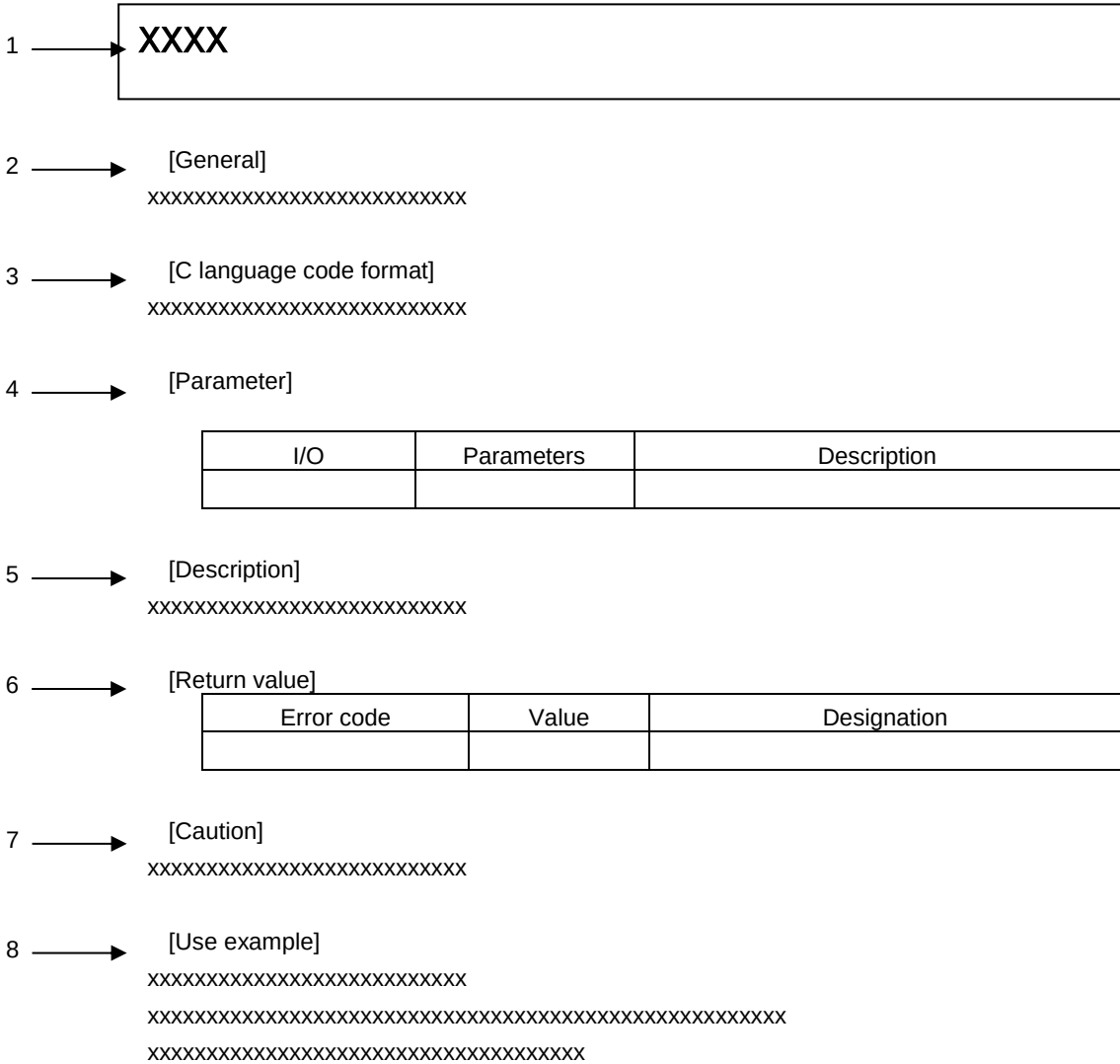
Table 9-2. LIN 2.1 Spec and LIN 2.1 Master Driver Type Definition List

Type of LIN 2.1 Spec	Type of LIN 2.1 Driver
<code>l_bool</code>	unsigned char
<code>l_u8</code>	unsigned char
<code>l_u16</code>	unsigned short
<code>l_signal_handle</code>	unsigned char
<code>l_flag_handle</code>	unsigned char
<code>l_irqmask</code>	unsigned char
<code>l_ifc_handle</code>	unsigned char
<code>l_ioctl_op</code>	unsigned char
<code>l_schedule_handle</code>	unsigned char

9. 3 Description of LIN 2.1 Software Master Driver Function

It explains the LIN 2.1 software master driver function according to the following forms.

Figure 9-1. Description Format of LIN 2.1 Software Master Driver Function



1. Name

This indicates the name of the LIN 2.1 software driver function.

2. [General]

This indicates each LIN 2.1 software driver function's general functions.

3. [C language code format]

This indicates the code format used to issue LIN 2.1 software driver function is C language.

4. [Parameter]

LIN 2.1 software driver function parameters are indicated in the following format.

I/O	Parameters	Description
A	B	C

A: Parameter I/O classification

I ... Input parameter

O ... Output parameter

B: Parameter type and name

C: Description of parameter

5. [Description]

This describes the functions of each LIN 2.1 software driver function.

6. [Return value]

This indicates return value of each LIN 2.1 software driver function. The format is as follows.

Error code	Value	Designation
A	B	C

A: The name when return value is error code.

B: Range of return value.

C: Description of return value.

7. [Caution]

This indicates cautions concerning LIN 2.1 software driver function. In particular, device-dependent cautions are explained.

8. [Use example]

This provides use examples for specific LIN 2.1 software driver functions.

9. 3. 1 [Master] LIN 2.1 Software Driver and Cluster Management

The function of LIN 2.1 Software Driver and Cluster Management is described from next page.

Table 9-3. List of LIN 2.1 Software Driver and Cluster Management (Master)

Function Name	Description
I_sys_init	System initialization

I_sys_init

[General]

The LIN 2.1 software driver is initialized.

[C language code format]

```
I_bool I_sys_init(void)
```

[Parameters]

None

[Description]

This function initializes the LIN 2.1 software driver system. It is necessary to call it before all function is called.

[Return value]

Error code	Value	Designation
L_SUCCESS	0x00	Initialize is succeed

*: For LIN software driver, only L_SUCCESS is returned.

Recommend the return value checked for safety.

[Use example]

```
/* After start up finished */
if ( I_sys_init() ){
    /* init error */
    ;
}
else{
    /* After this, other function can be called. */
}
```

9. 3. 2 [Master] Scalar Signal Read

The functions of scalar signal read are described from next page.

Table 9-4. List of Scalar Signal Read (Master)

Function Name	Description
I_bool_rd	Scalar signal read process of 1 bit
I_u8_rd	Scalar signal read process of 8 bits
I_u16_rd	Scalar signal read process of 16 bits

I_bool_rd

[General]

The 1 bit scalar signal is read.

[C language code format]

```
I_bool I_bool_rd(I_signal_handle sss)
```

[Parameters]

I/O	Parameter	Description
I	I_signal_handle sss	Signal name (macro value)

[Description]

This function reads the signal data in the message buffer that related to the specified signal name. It is possible to read it asynchronously with the LIN communication.

[Return value]

Error code	Value	Designation
-	0x00,0x01	Signal data

[Caution]

- Do not read the signal defined by the sizes other than 1 bit.
- Do not specify "sss" excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the transmission.

[Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

- ```
- Signal name : WINDOW_SWITCH
- Size : 1 bit
```

```
*/
```

```
if(I_bool_rd(WINDOW_SWITCH))
{
 "Processing for window switch ON"
}
else
{
 "Processing for window switch OFF"
}
```

## I\_u8\_rd

### [General]

The scalar signal from 1 to 8 bits is read.

### [C language code format]

```
I_u8 I_u8_rd(I_signal_handle sss)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |

### [Description]

This function reads the signal data in the message buffer that related to the specified signal name. It is possible to read it asynchronously with the LIN communication.

The current signal data is read regardless of the presence of new data.

### [Return value]

| Error code | Value  | Designation |
|------------|--------|-------------|
| -          | 0-0xFF | Signal data |

### [Caution]

- Do not read the signal defined by a size that is larger than 8 bits.
- Do not specify "sss" excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the transmission.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_STATUS

- Size : 8 bits

\*/

#define ST\_BUSY (0x00)

#define ST\_IDLE (0x01)

#define ST\_NG (0x02)

switch\_status = l\_u8\_rd(WINDOW\_STATUS);

if( switch\_status == ST\_BUSY )

{

    "Processing for BUSY state"

}

else if( switch\_status == ST\_NG )

{

    "Processing for NG state"

}

## I\_u16\_rd

### [General]

The scalar signal from 1 to 16 bits is read.

### [C language code format]

```
I_u16 I_u16_rd(I_signal_handle sss)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |

### [Description]

This function reads the signal data in the message buffer that related to the specified signal name. It is possible to read it asynchronously with the LIN communication.

The current signal data is read regardless of the presence of new data.

### [Return value]

| Error code | Value    | Designation |
|------------|----------|-------------|
| -          | 0-0xFFFF | Signal data |

### [Caution]

- Do not read the signal defined by the sizes larger than 16 bits.
- Do not specify "sss" excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the transmission.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_STATUS

- Size : 16 bits

\*/

#define ST\_BUSY (0x0000)

#define ST\_IDLE (0x0001)

l\_u16 switch\_status;

switch\_status = l\_u16\_rd(WINDOW\_STATUS);

switch( switch\_status )

{

case ST\_BUSY:

/\* Processing for BUSY state \*/

break;

case ST\_IDLE:

/\* Processing for IDLE state \*/

break;

default:

/\* Processing for error state \*/

break;

}

### 9. 3. 3 [Master] Scalar Signal Write

The functions of scalar signal write are described from next page.

**Table 9-5. Scalar Signal Write (Master)**

| Function Name | Description                            |
|---------------|----------------------------------------|
| I_bool_wr     | Scalar signal write process of 1 bit   |
| I_u8_wr       | Scalar signal write process of 8 bits  |
| I_u16_wr      | Scalar signal write process of 16 bits |

## I\_bool\_wr

### [General]

The 1 bit scalar signal is written.

### [C language code format]

```
void I_bool_wr(I_signal_handle sss, I_bool v)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |
| I   | I_bool v            | Writing data              |

### [Description]

This function writes the data specified by “v” in the message buffer that related to the specified signal name. After writing, the update flag of the corresponding signal is set.

### [Return value]

None

### [Caution]

- Do not write the signal defined by the sizes larger than 1 bit.
- Do not specify “sss” excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the reception.

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW, WINDOW\_SWITCH

- Size : 1 bit

\*/

```
#define PUSH (1)
```

```
#define OPEN (1)
```

```
/* Switch push is detected */
```

```
if(I_bool_rd(WINDOW_SWITCH) == PUSH)
```

```
{
```

```
 /* 1 bit signal OPEN is written */
```

```
 I_bool_wr(WINDOW, OPEN);
```

```
}
```

## I\_u8\_wr

### [General]

The scalar signal from 1 to 8 bits is written.

### [C language code format]

```
void I_u8_wr(l_signal_handle sss, I_u8 v)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |
| I   | I_u8 v              | Writing data              |

### [Description]

This function writes the data specified by “v” in the message buffer that related to the specified signal name. After writing, the update flag of the corresponding signal is set.

### [Return value]

None

### [Caution]

- Do not write the signal defined by the sizes larger than 8 bits.
- Do not specify “sss” excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the reception.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW, WINDOW\_SWITCH

- Size : 8 bits

\*/

#define OPEN (0x00)

#define CLOSE (0x01)

#define SLEEP (0x02)

#define UP (0x00)

#define DOWN (0x01)

switch(l\_u8\_rd(WINDOW\_SWITCH))

{

case UP:

l\_u8\_wr( WINDOW, OPEN );

break;

case DOWN:

l\_u8\_wr( WINDOW, CLOSE );

break;

default:

l\_u8\_wr( WINDOW, SLEEP);

break;

}

## I\_u16\_wr

### [General]

The scalar signal from 1 to 16 bits is written.

### [C language code format]

```
void I_u16_wr(I_signal_handle sss, I_u16 v)
```

### [Parameters]

| I/O | Parameter           | Description               |
|-----|---------------------|---------------------------|
| I   | I_signal_handle sss | Signal name (macro value) |
| I   | I_u16 v             | Writing data              |

### [Description]

This function writes the data specified by “v” in the message buffer that related to the specified signal name. After writing, the update flag of the corresponding signal is set.

### [Return value]

None

### [Caution]

- Do not write the signal defined by the sizes larger than 16 bits.
- Do not specify “sss” excluding the defied signal.
- Do not specify the signal in the message buffer that setting as the reception.

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : VOLUME

- Size : 12 bits

\*/

```
I_u16 vol;
```

```
/* The volume level is acquired */
```

```
vol = GetVolLevel();
```

```
/* Set in signal */
```

```
I_u16_wr(VOLUME, vol);
```

### 9. 3. 4 [Master] Byte Array Read

The functions of byte array read are described from next page.

**Table 9-6. List of Byte Array Read (Master)**

| Function Name | Description             |
|---------------|-------------------------|
| l_bytes_rd    | Byte array read process |

## I\_bytes\_rd

### [General]

The byte array data is read.

### [C language code format]

```
void I_bytes_rd(I_signal_handle sss, I_u8 start, I_u8 count, I_u8* const data)
```

### [Parameters]

| I/O | Parameter           | Description                     |
|-----|---------------------|---------------------------------|
| I   | I_signal_handle sss | Signal name (macro value)       |
| I   | I_u8 start          | Reading beginning byte number   |
| I   | I_u8 count          | Reading number of bytes         |
| O   | I_u8* const data    | Stored location of reading data |

### [Description]

This function reads byte data for a number of counts from the start byte in the message buffer related to the specified signal name.

For instance, for the message of 7 bytes longs (number is 0 to 6) in byte array, to read the data from 3rd to 4th, 3 is set in start and 2 is set in count. In this case, the 3rd value is written in data [0] and the 4th value is written in data [1].

The current data in message buffer is read regardless of the presence of new data.

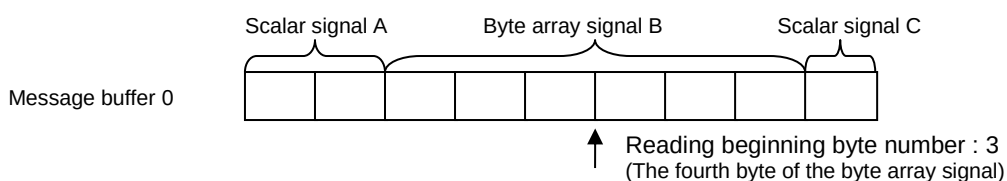
### [Return value]

None

### [Caution]

- Do not specification that the signal size becomes outside the range by combining start and count.  
A correct value cannot be read.
- Only the signal defined as byte array can be reading (Signal that offset and size become multiple of 8).  
When specifying by other definitions, a correct value cannot be read.
- Do not specify the signal in the message buffer that setting as the transmission.
- Reading beginning byte number is not number from head of message buffer, but number from head of signal name. Refer to the following example.

Ex. When you want to read it from the fourth byte of byte array



[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_TEST

- Size : 4 bytes

\*/

l\_u8 data[4];

/\* the 4 bytes data stored in data array \*/

l\_bytes\_rd(WINDOW\_TEST, 0, 4, data);

/\* the 2 bytes data from 2nd to 3rd stored in data array\*/

l\_bytes\_rd(WINDOW\_TEST, 2, 2, data);

### 9. 3. 5 [Master] Byte Array Write

The functions of scalar signal write are described from next page.

**Table 9-7. List of Byte Array Write (Master)**

| Function Name | Description              |
|---------------|--------------------------|
| l_bytes_wr    | Byte array write process |

## I\_bytes\_wr

### [General]

The byte array data is written.

### [C language code format]

```
void I_bytes_wr(I_signal_handle sss, I_u8 start, I_u8 count, const I_u8* const data)
```

### [Parameters]

| I/O | Parameter              | Description                     |
|-----|------------------------|---------------------------------|
| I   | I_signal_handle sss    | Signal name (macro value)       |
| I   | I_u8 start             | Writing beginning byte number   |
| I   | I_u8 count             | Writing number of bytes         |
| I   | const I_u8* const data | Stored location of writing data |

### [Description]

This function writes byte data for a number of counts from the start byte in the message buffer related to the specified signal name.

For instance, for the message of 7 bytes longs (number is 0 to 6) in byte array, to write the data from 3rd to 4th, 3 is set in start and 2 is set in count. In this case, the 3rd value is read from data [0] and the 4th value is read from data [1].

After writing, the update flag of the corresponding signal is set.

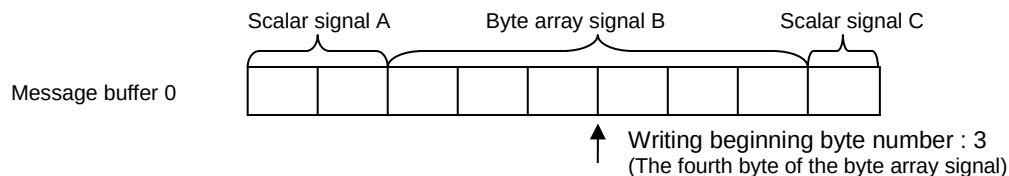
### [Return value]

None

### [Caution]

- Do not specification that the signal size becomes outside the range by combining start and count.  
A correct value cannot be read.
- Only the signal defined as byte array can be reading (Signal that offset and size become multiple of 8).  
When specifying by other definitions, a correct value cannot be read.
- Do not specify the signal in the message buffer that setting as the reception.
- Writing beginning byte number is not number from head of message buffer, but number from head of signal name. Refer to the following example.

Ex. When you want to write it from the fourth byte of byte array



[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_TEST

- Size : 4 bytes

\*/

const l\_u8 data[4] = {0x00, 0x01, 0x02, 0x03};

/\* the 4 bytes data of data array is written \*/

l\_bytes\_wr(WINDOW\_TEST, 0, 4, data);

/\* the 2 bytes data is written in WINDOW\_TEST signal \*/

l\_bytes\_wr(WINDOW\_TEST, 2, 2, data);

### 9.3.6 [Master] Notification

The functions of notification are described from next page.

**Table 9-8. List of Notification (Master)**

| Function Name | Outline                         |
|---------------|---------------------------------|
| l_flg_tst     | Signal update flag confirmation |
| l_flg_clr     | Signal update flag clear        |

## I\_flg\_tst

### [General]

The value of signal update flag is read.

### [C language code format]

```
I_bool I_flg_tst(I_flag_handle fff)
```

### [Parameters]

| I/O | Parameter         | Description                     |
|-----|-------------------|---------------------------------|
| I   | I_flag_handle fff | Flag name (same as signal name) |

### [Description]

This function reads the signal update flag related to the specified flag name.

Timing that the flag updated is as follows.

- When the signal data is normally received from the LIN bus, and the I\_sch\_tick function is called.
- When the scalar signal writing function and byte array writing function are called, and data is written in the signal.

### [Return value]

| Error code | Value | Designation |
|------------|-------|-------------|
| -          | 0x00  | Not update  |
| -          | 0x01  | Update      |

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_STATUS

- Size : 1 byte

\*/

```
I_bool status;
```

```
if(I_flg_tst(WINDOW_STATUS))
```

```
{
```

```
 /* processing when WINDOW_STATUS is updated is described */
```

```
 status = I_bool_rd(WINDOW_STATUS);
```

```
}
```

## I\_flg\_clr

### [General]

The signal update flag is cleared by 0.

### [C language code format]

```
void I_flg_clr(I_flg_handle fff)
```

### [Parameters]

| I/O | Parameter        | Description                                     |
|-----|------------------|-------------------------------------------------|
| I   | I_flg_handle fff | Flag name (same as macro value and signal name) |

### [Description]

This function clear the signal update flag related to the specified flag name.

Timing that the flag is cleared is as follows.

- When issue this function
- When the transmission for LIN bus is successful completed, and the I\_sch\_tick function is called.

### [Return value]

None

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Signal name : WINDOW\_TEST
- Size : 1 byte

\*/

```
I_u8 status;
```

```
if(I_flg_tst(WINDOW_TEST))
{
 status = I_u8_rd(WINDOW_TEST);
```

```
 /* the update flag is cleared */
```

```
 I_flg_clr(WINDOW_TEST);
```

```
}
```

### 9. 3. 7 [Master] Schedule Management

The functions of schedule management are described from next page.

**Table 9-9. List of Schedule Management (Master)**

| Function Name | Description      |
|---------------|------------------|
| I_sch_tick    | Schedule control |
| I_sch_set     | Schedule setting |

## I\_sch\_tick

### [General]

The schedule management is done by calling this function at constant intervals.

### [C language code format]

```
I_u8 I_sch_tick(I_ifc_handle iii)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

It is checked whether transfer the next entry of the current schedule table that is specified LIN interface can begin. And if it is possible, transfer the table entry begin.

When transfer the next entry of the schedule table can begin, the number is returned by next calling of this function. If transfer of the next entry cannot begin, return 0.

### [Return value]

| Error code | Value                                                | Designation                                                                                                  |
|------------|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| -          | 0x00                                                 | By next calling, transfer the next entry of the schedule table does not begin.                               |
| -          | 0x01 to 0xFF (less than the number of maximum entry) | By next calling, transfer the next entry of the schedule table is beginning. The value is that entry number. |

### [Caution]

- The signal update when go-to-sleep-command is transmitted has not synchronized with this function.  
When the transmission of go-to-sleep-command is completed, the signal is updated. (Even if this function is not called, it shifts to sleep mode when the transmission of go-to-sleep-command is normally succeed)  
After a frame transport, signals are updated at the time of first calling I\_sch\_tick.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if you don't use multi-channel function.
- Do not call this function when the LIN 2.1 software driver is sleep mode. If it is called, "1" is set in "Error in none active" of a detailed status bit.

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Channel name: LIN\_CHANNEL0

\*/

/\* Timer interrupt function (time\_base = 20ms) \*/

void main\_application\_20ms( void )

{

/\* schedule update in each 20 ms \*/

I\_sch\_tick( LIN\_CHANNEL0 );

}

## l\_sch\_set

### [General]

The schedule table is set.

### [C language code format]

```
void l_sch_set(l_ifc_handle iii, l_schedule_handle schedule, l_u8 entry)
```

### [Parameters]

| I/O | Parameter                  | Description                     |
|-----|----------------------------|---------------------------------|
| I   | l_ifc_handle iii           | Interface name (channel number) |
| I   | l_schedule_handle schedule | Schedule table name             |
| I   | l_u8 entry                 | Schedule entry number           |

### [Description]

The schedule table in the specified LIN interface is switched to the schedule table specified in “schedule”.  
 After calling this function, transfer of the entry specified with “entry” is begins on timing (when call l\_sch\_tick function one or more times) that the next entry of the current schedule table is transfer.  
 When 0 or 1 is set in “entry”, transfer the first entry of “schedule” begins.

### [Return value]

None

### [Caution]

- The entry number of schedule table is from 1 to n (n is number of entry in schedule tables).  
 Note the different from schedule table structure array element number (it is from 0 to n-1).
- Do not specify “schedule” and “entry” that does not exist in the schedule table or the entry number.
- Do not specify the interface name excluding “LIN\_CHANNEL0” if you don’t use multi-channel function.

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Channel name : LIN\_CHANNEL0
- Schedule table name : SCH\_TABLE0

\*/

/\* Timer interrupt function (time\_base = 20ms)\*/

```
void main_application_20ms(void)
{
 /* When the 3rd of the current schedule table entry is begun,
 switch to the entry1 of schedule table “SCH_TABLE0”.
 */
 if(l_sch_tick (LIN_CHANNEL0) == 3)
 {
 l_sch_set(LIN_CHANNEL0, SCH_TABLE0, 1);
 }
}
```

### 9.3.8 [Master] Interface Management

The functions of interface management are described from next page.

**Table 9-10. List of Interface Management (Master)**

| Function Name     | Description               |
|-------------------|---------------------------|
| I_ifc_init        | Interface initialization  |
| I_ifc_goto_sleep  | go-to-sleep-command issue |
| I_ifc_wake_up     | Wake up issue             |
| I_ifc_read_status | Status reading            |

## I\_ifc\_init

### [General]

The LIN interface is initialized.

### [C language code format]

```
I_bool I_ifc_init (I_ifc_handle iii)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

This function performs the initialization process related to the specified LIN interface to enable LIN communication.

This function executes the permission process for LIN transmission/reception.

### [Return value]

| Error code | Value | Designation |
|------------|-------|-------------|
| L_FAIL     | 0xFF  | Failed      |
| L_SUCCESS  | 0x00  | Succeeded   |

### [Caution]

- Use this function after issuing the I\_sys\_init function.
- Issue this function before using the LIN 2.1 software driver functions other than I\_sys\_init function.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if you don't use multi-channel function.

### [Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Channel name: LIN\_CHANNEL0

\*/

```
if(I_sys_init())
{
 /* error processing is described */
}
else
{
 /* interface initialization */
 if(I_ifc_init(LIN_CHANNEL0))
 {
 /* error processing is described */
 }
}
```

## I\_ifc\_goto\_sleep

### [General]

Go-to-sleep-command is issued.

### [C language code format]

```
void I_ifc_goto_sleep(I_ifc_handle iii)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

This function transmits go-to-sleep-command to the specified LIN bus. When go-to-sleep-command transmit succeed, "Goto sleep" of the status bit (refer to I\_ifc\_read\_status) is set, and it shifts to sleep mode. When failing in the transmission of go-to-sleep-command, neither the set of "Goto sleep" nor the shift to sleep mode are done.

Transmit go-to-sleep-command by this function call, at least one entry of master request frame should exist in the set schedule table.

### [Return value]

None

### [Caution]

- Go-to-sleep-command is issued as one of the diagnostic commands. Therefore, before this function is called, confirm the Id\_is\_ready function is called and LD\_SERVICE\_IDLE is returned. When LD\_SERVICE\_BUSY is returned and this function called, the function call is neglected.
- When the entry of master request frame does not exist in the set schedule table, go-to-sleep-command is not issued. In that case, call the I\_sch\_set function after calling this function, and switch to the schedule table where the entry of master request frame exists.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if don't use multi-channel function.

### [Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

```
- Channel name: LIN_CHANNEL0
```

```
*/
```

```
/* after it is confirmed that diagnostic command can be acceptance, go-to-sleep-command transmission */
```

```
if(Id_is_ready (LIN_CHANNEL0) == LD_SERVICE_IDLE)
```

```
{
```

```
 I_ifc_goto_sleep(LIN_CHANNEL0);
```

```
}
```

## I\_ifc\_wake\_up

### [General]

Wake up is issued.

### [C language code format]

```
void I_ifc_wake_up(I_ifc_handle iii)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

This function transition the specified LIN interface from sleep mode to wake up mode and transmits the wake up low pulse from 250 us to 5 ms to the LIN bus.

### [Return value]

None

### [Caution]

- It is possible to issue it only for sleep mode.
- After calling this function, wait processing is necessary till frame transfer beginning.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if don't use multi-channel function.

### [Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

- Channel name: LIN\_CHANNEL0
- Status mask bit: LD\_MASK\_SLEEP

```
*/
```

```
/* when sleep mode is detected, wake up immediately */
```

```
/*sleep detection */
```

```
if((I_ifc_read_status(LIN_CHANNEL0) & LD_MASK_SLEEP) == LD_MASK_SLEEP)
{
 I_ifc_wake_up();
}
```

## I\_ifc\_read\_status

### [General]

LIN software driver's various statuses are read.

### [C language code format]

```
I_u16 I_ifc_read_status(I_ifc_handle iii)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

This function reads status information in the specified LIN interface.

| 15             | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6                  | 5                               | 4            | 3          | 2       | 1                   | Bit0              |
|----------------|----|----|----|----|----|---|---|---|--------------------|---------------------------------|--------------|------------|---------|---------------------|-------------------|
| Last frame PID |    |    |    |    |    |   |   | 0 | Save configuration | Event triggered frame collision | Bus activity | Goto sleep | Overrun | Successful transfer | Error in response |

Last frame ID : PID of the last communication frame is shown.

Save configuration : Not supported

Event triggered frame collision : Set during the execution of collision resolution of the event trigger frame

Bus activity : When a rising or falling edge of the bus is detected, it is set.

Goto sleep : When go-to-sleep-command is normally transmitted, it is set.

Overrun : When over 2 transfers are executed before I\_ifc\_read\_status is called, it is set.

Successful transfer : When the transfer I succeeded, it is set.

Error in response : When an illegal transfer is detected while response, it is set.

### [Caution]

- Use this function after issuing the I\_ifc\_init function.
- After this function or I\_ifc\_init function is issued, status cleared by 0.
- Status is an accumulation flag. When two or more transfer is done from the call of previous this function, status is overwritten.
- When this function is called continuously with no transfer in the bus, the 2nd call returns 0.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if don't use multi-channel function.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Channel name: LIN\_CHANNEL0

- Status mask bit: LD\_MASK\_ERROR\_IN\_RESPONSE, LD\_MASK\_SUCCESSFUL\_TRANSFER

\*/

```
status = l_ifc_read_status(LIN_CHANNEL0);
```

```
mask = LD_MASK_ERROR_IN_RESPONSE | LD_MASK_SUCCESSFUL_TRANSFER;
```

```
if(status & mask) == mask)
```

```
{
```

```
 /* Detection error and succeeded, in this case, it is though that the transfer is intermittently done. */
```

```
}
```

### 9.3.9 [Master] Node Configuration

The functions of node configuration are described from next page.

**Table 9-11. List of Node Configuration (Master)**

| Function Name            | Description              |
|--------------------------|--------------------------|
| Id_is_ready              | Node configuration ready |
| Id_check_response        | Node configuration check |
| Id_assign_frame_id_range | PID allocation           |
| Id_read_by_id            | ID reading               |

## Id\_is\_ready

### [General]

The reception state of the diagnostic command is checked.

### [C language code format]

```
I_u8 Id_is_ready (I_ifc_handle iii) ;
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

This function returns LD\_SERVICE\_IDLE when a diagnostic command (Id\_assign\_frame\_id\_range, Id\_read\_by\_id, I\_ifc\_goto\_sleep) for the specified LIN interface is executable. Also, when this function returns LD\_SERVICE\_IDLE, the result of the previous diagnostic command takes effect.

No other node configuration function should be called until this function returns LD\_SERVICE\_IDLE.

### [Return value]

| Error code          | Value | Designation                                                                         |
|---------------------|-------|-------------------------------------------------------------------------------------|
| LD_SERVICE_IDLE     | 0     | Diagnostic commands can be accepted.                                                |
| LD_SERVICE_BUSY     | 1     | From service start to completion of master request transmission.                    |
| LD_REQUEST_FINISHED | 2     | From master request transmission completion to slave response reception completion. |
| LD_SERVICE_ERROR    | 3     | An error occurs on the bus.                                                         |

### [Caution]

- Use this function after issuing the I\_sys\_init function.
- To transmit go-to-sleep-command as master request frame, the I\_ifc\_goto\_sleep function is included in the diagnostic command.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if don't use multi-channel function.

### [Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

```
- Channel name: LIN_CHANNEL0
```

```
*/
```

```
/* Id_assign_frame_id is transmission after it is confirmed that diagnostic command can be acceptance. */
```

```
if(Id_is_ready (LIN_CHANNEL0) == LD_SERVICE_IDLE)
```

```
{
```

```
 Id_assign_frame_id_range (LIN_CHANNEL0, u1tNad, u1tStartIndex, (u1*)&pids[0]);
```

```
}
```

## ld\_check\_response

### [General]

The result of diagnostic command is checked.

### [C language code format]

```
void ld_check_response(l_ifc_handle iii, l_u8* RSID, l_u8* error_code)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | l_ifc_handle iii | Interface name (channel number) |
| O   | l_u8* RSID       | RSID received at the last time  |
| O   | l_u8* error_code | Detailed information in error   |

### [Description]

This function returns the result of the last diagnostic command that supports the slave response frame issued for the specified LIN interface with RSID and error\_code. The error code list is as follows. The error\_code is effective when the return value of the function is LD\_ERR\_NEGRESPONSE.

**Table 9-12. Error Code List**

| Macro Name         | Value | Designation                                |
|--------------------|-------|--------------------------------------------|
| LD_ERR_NOERROR     | 0x00  | No error                                   |
| LD_ERR_NEGRESPONSE | 0x01  | Slave response frame is negative response  |
| LD_ERR_SRFERROR    | 0x02  | PDU in the slave response frame is illegal |

Result of transmit the diagnostic command, even if this function is called when the slave node does not respond (when you do not return the response) , the value is set in RSID and error\_code. In this case, set value is RSID and error\_code set at the last time. Moreover, the return value is same with last issued one.

### [Caution]

- Use this function after issuing the l\_sys\_init function.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if don't use multi-channel function.
- In this version, the diagnostic commands that support slave response frames are ld\_read\_by\_id and ld\_assign\_frame\_id\_range. Refer to the RSID and error\_code values only for diagnostic result commands that do not support slave response frames (l\_ifc\_goto\_sleep).

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Channel name: LIN\_CHANNEL0

\*/

l\_u8 RSID;

l\_u8 error\_code;

/\* after confirmed that diagnostic command can be an acceptance, the result of diagnostic command is acquired.

\*/

if( ld\_is\_ready( LIN\_CHANNEL0 ) == LD\_SERVICE\_IDLE )

{

ld\_check\_response( LIN\_CHANNEL0, &RSID, &error\_code );

}

## Id\_assign\_frame\_id\_range

### [General]

PID is allocated in the message buffer of the slave node.

### [C language code format]

```
void Id_assign_frame_id_range (I_ifc_handle iii,I_u8 NAD,I_u8 start_index,const I_u8* const PIDs)
```

### [Parameters]

| I/O | Parameter        | Description                                   |
|-----|------------------|-----------------------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number)               |
| I   | I_u8 NAD         | NAD value                                     |
| I   | I_u8 start_index | Index value to start rewriting the frame list |
| I   | I_u8* PIDs       | Pointer to PID list                           |

### [Description]

Modifies or disables the PIDs of up to four message frames of nodes with matching NADs.

However, frames with IDs of 60~63 (0x3C~0x3F) cannot be changed.

The start index specifies the first frame to which the PID is assigned. The order of the frames depends on the slave node. The first index in the list starts from 0.

The set values for the PID list are as follows

**Table 9-13. PID list setting value**

| Value                    | Operation                              |
|--------------------------|----------------------------------------|
| Valid ID value (0 to 59) | Change to the specified PID            |
| 0x00                     | Unassign                               |
| 0xFF                     | Maintain previous value without change |

### [Return value]

None

### [Caution]

- Before this function is called, confirm LD\_SERVICE\_IDLE is returned by calling Id\_is\_ready function. If this function is called when something except LD\_SERVICE\_IDLE is returned, the function call will be ignored.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if don't use multi-channel function.

[Use example]

/\* beforehand, it changes to the following settings with the configuration file.

- Channel name: LIN\_CHANNEL0

\*/

/\* after confirmed that diagnostic command can be an acceptance, Id\_assign\_frame\_id\_range is transmitted \*/

if( Id\_is\_ready ( LIN\_CHANNEL0 ) == LD\_SERVICE\_IDLE )

{

    Id\_assign\_frame\_id\_range ( LIN\_CHANNEL0, u1tNad, u1tStartIndex, (u1\*)&pids[0]);

}

## Id\_read\_by\_id

### [General]

Various ID is read from the slave node.

### [C language code format]

```
void Id_read_by_id(l_ifc_handle iii, l_u8 NAD, l_u16 supplier_id, l_u16 function_id, l_u8 id, l_u8* const data)
```

### [Parameters]

| I/O | Parameter         | Description                     |
|-----|-------------------|---------------------------------|
| I   | l_ifc_handle iii  | Interface name (channel number) |
| I   | l_u8 NAD          | NAD value                       |
| I   | l_u16 supplier_id | Supplier ID value               |
| I   | l_u16 function_id | Function ID value               |
| I   | l_u8 id           | ID value                        |
| O   | l_u8* const data  | Pointer of writing data         |

### [Description]

The property specified with "id" of slave node to which NAD, supplier\_id, and function\_id are corresponding is set in RAM area specified with data. When slave node is not supported the specified "id", the negative response is set.

The relation of "writing data", id and negative response is described in CHAPTER 5 .

### [Return value]

None

### [Caution]

- Before this function is called, confirm LD\_SERVICE\_IDLE is returned by calling Id\_is\_ready function. If this function is called when anything other than LD\_SERVICE\_IDLE is returned, the function call will be ignored.
- Do not specify the interface name excluding "LIN\_CHANNEL0" if don't use multi-channel function.

### [Use example]

```
/* beforehand, it changes to the following settings with the configuration file.
```

```
- Channel name: LIN_CHANNEL0
```

```
*/
```

```
/* after confirmed that diagnostic command can be an acceptance, Id_read_by_id is transmitted. */
```

```
if(Id_is_ready (LIN_CHANNEL0) == LD_SERVICE_IDLE)
```

```
{
```

```
 Id_assign_frame_id_range (LIN_CHANNEL0, 0, 0xA6, 0xE7, 0xA8, 0xE9);
```

```
}
```

### 9. 3. 10 [Master] User provided call-outs

The functions of call-out are described from next page.

**Table 9-14. List of User provided call-outs (Master)**

| Function Name                       | Description                     |
|-------------------------------------|---------------------------------|
| <code>l_sys_irq_disable</code>      | Interrupt prohibition setting   |
| <code>l_sys_irq_restore</code>      | Interrupt reintegration setting |
| <code>l_sys_call_wake_up</code>     | Wake up start                   |
| <code>l_sys_call_fatal_error</code> | Resolve fatal error             |

## I\_sys\_irq\_disable

### [General]

The interrupt is prohibited.

### [C language code format]

```
I_irqmask I_sys_irq_disable(void)
```

### [Parameters]

None

### [Description]

This function is a call-outs function to prohibit interrupt.

DI is executed, and current interrupt is returned. The return value is the PSW value for the CC-RL edition, and the value returned by the IAR intrinsic function `__get_interrupt_state` for the IAR edition.

### [Return value]

| I/O | Value                                                                     | Designation |
|-----|---------------------------------------------------------------------------|-------------|
| -   | [CC-RL]<br>Interrupt status bit of PSW<br>[IAR]<br>Global interrupt state | -           |

### [Caution]

- Because of this is call-outs function, it is possible to change, but you don't need to modify the Smart Configurator code. (See "5. 2. 7 User-defined callouts implementation")

### [Use example]

- It is automatically called. Don't modify from the provided code in sample.

## I\_sys\_irq\_restore

### [General]

It returns the state of the interrupt.

### [C language code format]

```
void I_sys_irq_restore(I_irqmask previous)
```

### [Parameters]

| I/O | Parameter          | Description                                                                                                        |
|-----|--------------------|--------------------------------------------------------------------------------------------------------------------|
| I   | I_irqmask previous | Return value of last I_sys_irq_disable().<br>[CC-RL] Interrupt status bit of PSW,<br>[IAR] Global interrupt state) |

### [Description]

This function is a call-outs function to return the interrupt state.

In default, it set to EI when "previous" is interrupt enable ("0x80"), and besides, it set to DI for the CC-RL edition. For the IAR edition, it restores the interrupt state specified by "previous".

### [Return value]

None

### [Caution]

- Because of this is call-outs function, it is possible to change, but you don't need to modify the Smart Configurator code. (See "5. 2. 7 User-defined callouts implementation")

### [Use example]

- It is automatically called. Don't modify the provided code in sample.

## **I\_sys\_call\_wake\_up**

### [General]

It is called immediately after reception of wake up.

It is original enhanced function of this driver.

### [C language code format]

```
void I_sys_call_wake_up(I_ifc_handle iii)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

This function is call-out function when the LIN 2.1 software driver receives wake up.

### [Return value]

None

### [Caution]

- This function need not necessarily be mounted. When not using it, make it empty. In the code provided by the Smart Configurator, it is an empty function. (See "5. 2. 7 User-defined callouts implementation")

### [Use example]

- It is automatically called. It is not necessary to use it in the application.

## I\_sys\_call\_fatal\_error

### [General]

It is called by LIN driver when an error related to hardware of this node is detected.

It is original enhanced function of this driver.

### [C language code format]

```
void I_sys_call_fatal_error(I_ifc_handle iii)
```

### [Parameters]

| I/O | Parameter        | Description                     |
|-----|------------------|---------------------------------|
| I   | I_ifc_handle iii | Interface name (channel number) |

### [Description]

This function is called out from the LIN 2.1 software driver to notify user when LIN driver detects an error can't be resolved by driver.

At this calling, LIN driver is disconnected from LIN bus and RAM is initialized.

Application should call I\_ifc\_init API in this function in order to re-connect to bus. If I\_ifc\_init is failed, application should retry.

### [Return value]

None

### [Caution]

The code for this function provided by the Smart Configurator does not take any action if the I\_ifc\_init function fails, so add a retry if necessary. (See "5. 2. 7 User-defined callouts implementation")

### [Use example]

- It is automatically called. It is not necessary to use it in the application.

## CHAPTER 10 Example of LIN description file (LDF) description

The LIN Description File (LDF) is a standard format for defining LIN networks.

LDF defines information necessary for LIN communication, such as global definitions, node definitions, signal definitions, frame definitions, and schedule definitions.

An example of LDF description is shown below.

In this example, a description example is shown for each category, but in reality, they are all defined in a single file.

| Category                                                                              | Example description                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Global information definition<br>(LIN protocol version, communication speed, etc.)    | <pre>LIN_description_file;<br/>LIN_protocol_version = "2.1";<br/>LIN_language_version = "2.1";<br/>LIN_speed = 9.6 kbps;</pre>                                                                                                                                                                                                                                                                                                                                                  |
| Node definition<br>(node name, time base, etc.)                                       | <pre>Nodes {<br/>    Master: LIN_Master, 10 ms, 0.2 ms ;<br/>    Slaves: LIN_Slave ;<br/>}</pre>                                                                                                                                                                                                                                                                                                                                                                                |
| Signal definition<br>(signal name, size, initial value, sending/receiving node, etc.) | <pre>Signals {<br/>    Sample_PublishSig1: 10, 0, LIN_Master, LIN_Slave ;<br/>    Sample_PublishSig2: 3, 0, LIN_Master, LIN_Slave ;<br/>    Sample_PublishSig3: 3, 0, LIN_Master, LIN_Slave ;<br/>    Sample_SubscribeSig1: 10, 0, LIN_Slave,<br/>        LIN_Master ;<br/>    Sample_SubscribeSig2: 3, 0, LIN_Slave,<br/>        LIN_Master ;<br/>    Sample_SubscribeSig3: 2, 0, LIN_Slave,<br/>        LIN_Master ;<br/>    RsErr: 1, 0, LIN_Slave, LIN_Master ;<br/>}</pre> |

| Category                                                      | Example description                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Diagnostic signal definition                                  | <pre>Diagnostic_signals {<br/>  MasterReqB0: 8, 0 ;<br/>  MasterReqB1: 8, 0 ;<br/>  MasterReqB2: 8, 0 ;<br/>  MasterReqB3: 8, 0 ;<br/>  MasterReqB4: 8, 0 ;<br/>  MasterReqB5: 8, 0 ;<br/>  MasterReqB6: 8, 0 ;<br/>  MasterReqB7: 8, 0 ;<br/>  SlaveRespB0: 8, 0 ;<br/>  SlaveRespB1: 8, 0 ;<br/>  SlaveRespB2: 8, 0 ;<br/>  SlaveRespB3: 8, 0 ;<br/>  SlaveRespB4: 8, 0 ;<br/>  SlaveRespB5: 8, 0 ;<br/>  SlaveRespB6: 8, 0 ;<br/>  SlaveRespB7: 8, 0 ;<br/>}</pre> |
| Frame definition<br>(Frame name, ID, size, signal name, etc.) | <pre>Frames {<br/>  Sample_Frame1: 10, LIN_Master, 8 {<br/>    Sample_PublishSig1, 0 ;<br/>    Sample_PublishSig2, 16 ;<br/>    Sample_PublishSig3, 24 ;<br/>  }<br/>  Sample_Frame2: 20, LIN_Slave, 8 {<br/>    Sample_SubscribeSig1, 0 ;<br/>    Sample_SubscribeSig2, 10 ;<br/>    Sample_SubscribeSig3, 16 ;<br/>    RsErr, 63 ;<br/>  }<br/>}</pre>                                                                                                              |

| Category                                                                                 | Example description                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Diagnostic Frame Definition                                                              | <pre> Diagnostic_frames {   MasterReq: 0x3c {     MasterReqB0, 0 ;     MasterReqB1, 8 ;     MasterReqB2, 16 ;     MasterReqB3, 24 ;     MasterReqB4, 32 ;     MasterReqB5, 40 ;     MasterReqB6, 48 ;     MasterReqB7, 56 ;   }   SlaveResp: 0x3d {     SlaveRespB0, 0 ;     SlaveRespB1, 8 ;     SlaveRespB2, 16 ;     SlaveRespB3, 24 ;     SlaveRespB4, 32 ;     SlaveRespB5, 40 ;     SlaveRespB6, 48 ;     SlaveRespB7, 56 ;   } } </pre> |
| Node attribute definition<br>(LIN protocol version, product identification number, etc.) | <pre> Node_attributes {   LIN_Slave{     LIN_protocol = "2.1" ;     configured_NAD = 0xA ;     initial_NAD = 0xA ;     product_id = 0x6E, 0x3039, 0 ;     response_error = RsErr ;     P2_min = 50 ms ;     ST_min = 0 ms ;     N_As_timeout = 1000 ms ;     N_Cr_timeout = 1000 ms ;     configurable_frames {       Sample_Frame1 ;       Sample_Frame2 ;     }   } } </pre>                                                                 |

| Category                                                                                    | Example description                                                                                                                                                                                                   |
|---------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Schedule table definition<br>(schedule table name, frame name, transmission interval, etc.) | <pre>Schedule_tables {<br/>  Master_Req {<br/>    MasterReq delay 20 ms ;<br/>  }<br/>  Slave_Resp {<br/>    SlaveResp delay 20 ms ;<br/>  }<br/>  Normal_1 {<br/>    Sample_Frame1 delay 20 ms ;<br/>  }<br/>}</pre> |

**APPENDIX REVISION HISTORY**

| Edition  | Description                                                             | Chapter |
|----------|-------------------------------------------------------------------------|---------|
| Rev.1.00 | Created based on LIN 2.0, 2.1 SOFTWARE DRIVER User's Manual<br>Rev.4.80 | All     |

---

Preliminary User's Manual

Publication Date: April 20, 2022 Rev 1.00

Published by: Renesas Electronics Corporation

---

# LIN 2.1 SOFTWARE DRIVER

## for RL78/F23, F24