RX Family

ELC Module Using Firmware Integration Technology

Introduction

This application note describes the Renesas ELC module which uses Firmware Integration Technology (FIT). This module uses ELC to create links between other modules. In this document, this module is referred to as the ELC FIT module.

Target Devices

- RX113 Group
- RX130 Group, RX140 Group
- RX230 Group, RX231 Group, RX23W Group
- RX65N Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “5.2 Operation Confirmation Environment".

Related Documents

- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)

Contents

1. Overview

The ELC FIT module provides settings that allow the event link signals output by the various modules to be transmitted to other modules.

1.1 ELC FIT Module

The ELC FIT module can be used by being implemented in a project as an API. See section 2.12, Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the ELC FIT Module

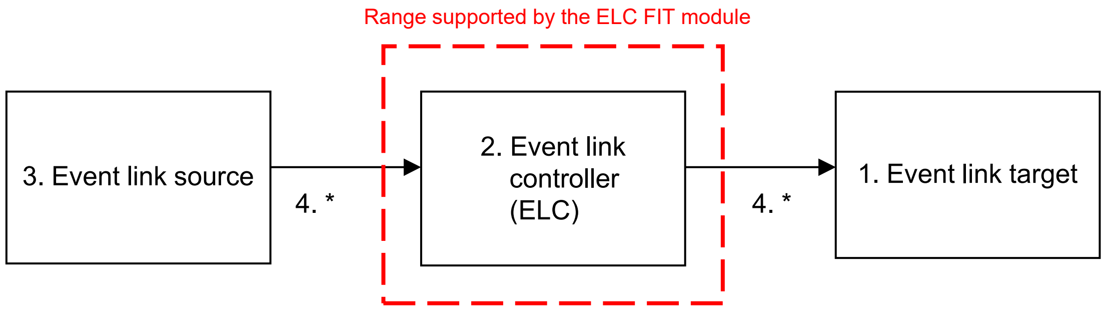
When used, the ELC FIT module is initialized and operated using the following procedure.

Step 1: Initialize the event link target module.
Step 2: Set up the event link from the event link source module to the event link target module.
Step 3: Initialize and start the event link source module.*1
Step 4: When an event signal is output from the event link source module to the event link target module, the operation set up in advance starts.

Note 1. When either an RTC or LVD is used as the event link source, that RTC or LVD should be set up first and then the ELC should be set up (step 2).

The ELC FIT module supports the setting up of an event link between the event link source module and the event link target module in step 2. Note that the user must perform the individual settings required for steps 1 and 3 separately.

Figure 1.1 presents an overview of the ELC and the setup procedure.



* Event signal output

ELC FIT module overview and setup procedure

Step 1. Event link target setup
First, the event link target is set up. When a port is set as the event link, the PODR and PDR registers for the corresponding port must be set.
Step 2. Event link controller (ELC) setup
This sets up the event link between the event link target and the event link source.
This module supports this event link setup operation.
Step 3. Event link source setup
Sets up and starts the event link source.
Step 4. Event signal output
The event link signal is output from the event link source to the ELC. The event link signal it transmitted to the event link target and the operations set up in advance start.

Figure 1.1 ELC FIT Module Overview

1.3 API Overview

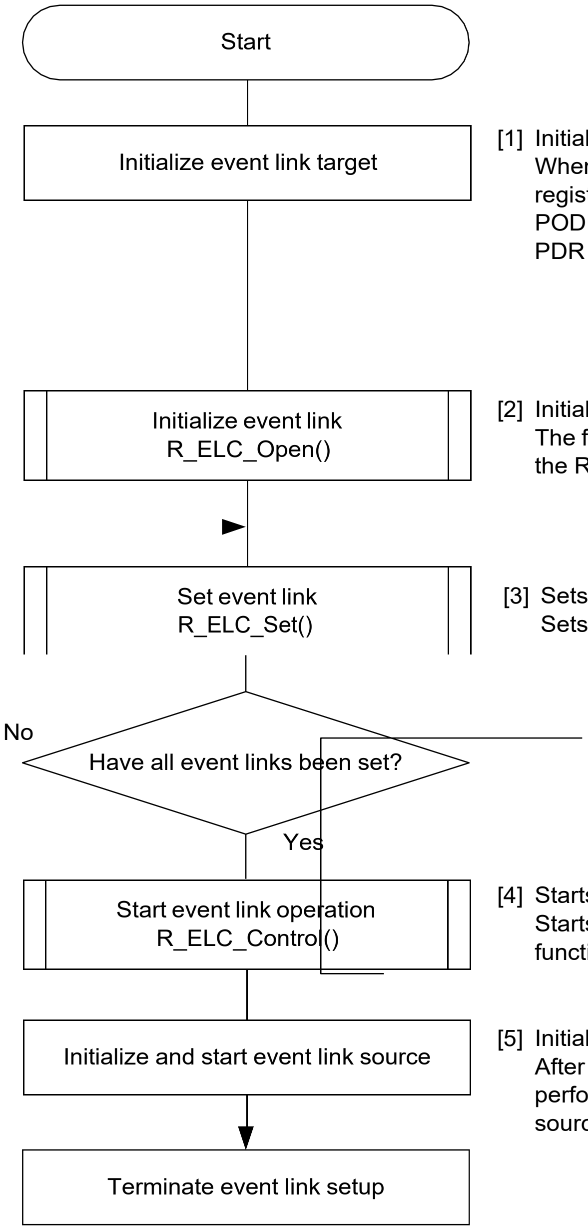
Table 1.1 lists the API functions included in this module. Also, section 2.8, Code Size, lists the size of the code sections used by this module.

Table 1.1 API Functions

Function	Function Description
R_ELC_Open	ELC module initialization
R_ELC_Set	Connects the event link source event signal and the event link target module and sets up the operations performed when an event occurs.
R_ELC_Control	Performs ELC module control. • Event link start/stop • Clear event link settings • Generate software events • Write a port buffer • Read a port buffer
R_ELC_Close	Stop the ELC module
R_ELC_GetVersion	Return the ELC FIT module version number.

1.4 Processing Example

Figure 1.2 shows an example of processing.



[1] Initializes the event link target.
When an event link is being set up for a port, the follow registers for the corresponding port are set.
PODR register: Sets the initial value for ports set to output.
PDR register: Sets the corresponding ports to input or output.

[2] Initializes the event link.
The following R_ELC_Set() function can be called by calling the R_ELC_Open() function.

[3] Sets the event link.
Sets the event link source and the event link target.

[4] Starts the event link operation.
Starts operation of the event link set with the R_ELC_Set() function by calling the R_ELC_Control() function.

[5] Initializes and starts the event link source.
After this, the operation set with the R_ELC_Set() function is performed when an event signal is output from the event link source.

Figure 1.2 Processing Example of the ELC FIT Module

1.5 State Transition Diagram

Figure 1.3 shows the state transition diagram for this module.

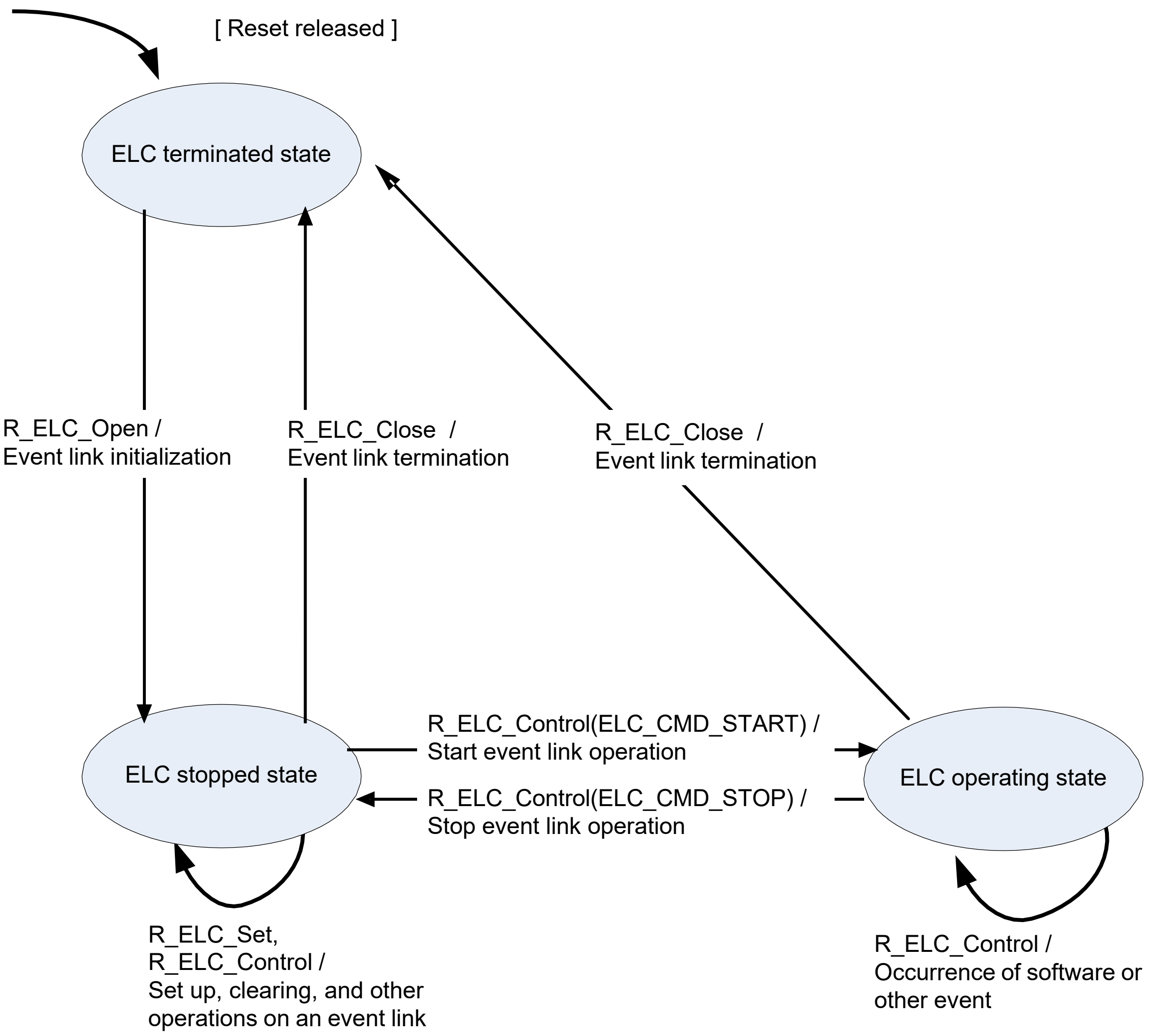


Figure 1.3 ELC FIT Module State Transition Diagram

2. API Information

The sample code provided with this application note has been tested under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- Event link controller (ELC)

2.2 Software Requirements

This driver is dependent upon the following FIT module.

- Renesas Board Support Package (r_bsp) Rev.5.20 or higher

2.3 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 5.2 Operation Confirmation Environment.

2.4 Interrupt Vector

To enable the ELC interrupt, use the R_ELC_Set function to specify the ELC interrupt as an event signal for event linking and set the interrupt priority level to a value other than 0.

Table 2.1 lists the Interrupt Vector Used in the ELC FIT Module.

Table 2.1 Interrupt Vector Used in the ELC FIT Module

Device	Interrupt Vector
RX113, RX130, RX140	ELSR8I interrupt (vector no.: 80) ELSR18I interrupt (vector no.: 106)
RX230, RX231, RX23W	ELSR8I interrupt (vector no.: 80) ELSR18I interrupt (vector no.: 106) ELSR19I interrupt (vector no.: 107)
RX65N	ELSR18I interrupt (vector no.: 193) (1) ELSR19I interrupt (vector no.: 194) (1)

Note 1. The interrupt vector numbers for software configurable interrupt B shown here are the default values specified in the board support package FIT module (BSP module).

2.5 Header Files

All API calls and their supporting interface definitions are located in r_elc_rx_if.h.

2.6 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7 Configuration Overview

The configuration option settings of this module are located in r_elc _rx_config.h. The option names and setting values are listed in the table below:

Configuration options in r_elc _rx_config.h	
Definition	**Description**
#define ELC_CFG_PARAM_CHECKING_ENABLE Note: The default value becomes the value of "BSP_CFG_PARAM_CHECKING_ENABLE defined in the file r_bsp_config.h.	Selects whether or not parameter checking is included in the code. 0: Parameter checking is omitted from the code at build time. 1: Parameter checking is included in the code at build time. The code size can be reduced by omitting parameter checking from the code at build time.

2.8 Code Size

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7 Configuration Overview.

The values in the table below are confirmed under the following conditions.

Module Revision: r_elc_rx rev2.00

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00

(The option of "-lang = c99" is added to the default settings of the integrated development environment.)

GCC for Renesas RX 8.3.0.201904

(The option of "-std=gnu99" is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.14.01

(The default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Memory Usage							
		Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
Device	Category	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX113	ROM	1,295 bytes	1,081 bytes	2,544 bytes	2,152 bytes	2,277 bytes	1,981 bytes
	RAM	20 bytes		20 bytes		18 bytes	
	Maximum stack usage*1	96 bytes		-		120 bytes	
RX130	ROM	1,300 bytes	1,087 bytes	2,504 bytes	2,120 bytes	2,284 bytes	1,989 bytes
	RAM	20 bytes		20 bytes		18 bytes	
	Maximum stack usage*1	96 bytes		-		120 bytes	
RX140	ROM	1,286 bytes	1,083 bytes	2,512 bytes	2,128 bytes	2,268 bytes	1,971 bytes
	RAM	20 bytes		20 bytes		18 bytes	
	Maximum stack usage*1	96 bytes		-		120 bytes	
RX230 RX231	ROM	1,696 bytes	1,482 bytes	3,400 bytes	2,992 bytes	3,088 bytes	2,736 bytes
	RAM	28 bytes		28 bytes		23 bytes	
	Maximum stack usage*1	96 bytes		-		120 bytes	
RX65N	ROM	1,670 bytes	1,480 bytes	3,408 bytes	3,056 bytes	3,082 bytes	2,777 bytes
	RAM	20 bytes		20 bytes		18 bytes	
	Maximum stack usage*1	108 bytes		-		128 bytes	

Note 1. The maximum stack sizes listed are for the case when interrupt processing is included in the API functions.

2.9 Parameters

This section describes the parameter structure used by the API functions in this module. The structure is located in r_elc_rx_if.h as are the prototype declarations of API functions.

[Event link source setup structure]

```
typedef struct elc_event_signal_s
{
  elc_eventlink_signal_t      event_signal;                    /* Event signal */
  elc_port_trigger_select_t   event_signal_input_port_edge;    /* Input edge selection */
  elc_single_port_select_t    event_signal_single_port;        /* Single port selection */
  uint_8                      event_signal_port_group_bit;     /* Port group specification
                                                                  selection */
} elc_event_signal_t;
```

[Event link target setup structure]

```
typedef struct elc_link_module_s
{
  elc_ module_t               link_module;                  /* Peripheral module to be linked */
  elc timer operation select tlink module timer operation;  /* Timer operation selection */
  elc_port_level_select_t     link module output port level; /* Output port level selection */
  elc_single_port_select_t    link_module_single_port;      /* Single port selection */
  uint8_t                     link_module_port_group_bit;   /* Pin selection for port group
                                                               specification */
  elc_port_buffer_select_t    link_module_port_buffer;      /* Port buffer overwrite selection */
  uint8_t                     link_module_interrupt_level;  /* ELC interrupt priority level */
  elc_interrupt_set _t        link_module_callbackfunc;     /* ELC interrupt callback function */;
}elc_link_module_t;
```

[Port buffer access structure]

```
typedef struct elc_pdbf_access_s
{
  elc_portbuffer_t          select_group;       /* Port buffer group selection */
  uint8_t                   value;              /* Port buffer write value or read value */;
}elc_pdbf_access_t;
```

2.10 Return Values

This section describes return values of API functions. This enumeration is located in r_elc_rx_if.h as are the prototype declarations of API functions.

[Error structure]

```
typedef enum
{
    ELC_SUCCESS,               /* Normal termination */
    ELC_ERR_LOCK_FUNC,         /* ELC already opened */
    ELC_ERR_INVALID_ARG        /* Illegal argument */
} elc_err_t;
```

2.11 Callback Functions

In this module, the callback function specified by the user is called when the ELC interrupt occurs.

The callback function is set up by storing the address of the callback function in the link_module_callbackfunc structure member (see 2.9 Parameters). When the callback function is called, the variable which stores the constant listed in Table 2.2 is passed as the argument.

The argument is passed as void type. Thus the argument of the callback function is cast to a void pointer. See examples below as reference.

When using a value in the callback function, type cast the value.

Table 2.2 Callback Function Parameters (enum elc_icu_t)

Constant Definition	Description
ELC_EVT_ICU1	Callback function called from interrupt handling for ELC interrupt 1
ELC_EVT_ICU2	Callback function called from interrupt handling for ELC interrupt 2*1
ELC_EVT_ICU_LPT	Callback function called from interrupt handling for the dedicated LPT ELC interrupt. *2

Note 1. Not available for RX113 Group, RX130 Group and RX140 Group.
Note 2. Not available for RX65N Group.

```
Sample callback function:
void my_elc_callback(void * pdata)
{
  elc_icu_t elc_icu_number;
  elc_icu_number = *(( elc_icu_t *)pdata); //cast pointer to elc_icu_t
  …
}
```

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

(5) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” for details.

3. API Functions

3.1 R_ELC_Open()

This function initializes the ELC FIT module and transitions the module from the ELC terminated state to the ELC stopped state. This function must be called before calling any other API functions.

Format

```
elc_err_t
   R_ELC_Open(void
)
```

Parameters

None

Return Values

ELC_SUCCESS /* Normal completion */
ELC_ERR_LOCK_FUNC /* The ELC was already open */

Properties

The declaration is located in r_elc_rx_if.h.

Description

Initializes an event link. Also, if the ELC interrupt is used, it sets the priority level of that interrupt.

Example

```
volatile elc_err_t ret;

ret = R_ELC_Open();
if( ELC_SUCCESS != ret)
{
  /* Error handling is performed if a failure to initialize occurs. */
}
```

Special Notes:

When this function is called, all of the content set by the R_ELC_Set() function and R_ELC_Control() function is cleared.

3.2 R_ELC_Set()

When this module is in the ELC stopped state, this function sets the event link source and event link target.

Format

```
elc_err_t R_ELC_Set(
   elc_event_signal_t  * const p_elc_event_signal
                                 /* Pointer to an link source setup structure */
   elc_link_module_t  * const p_elc_module
                                 /* Pointer to an link target setup structure */
)
```

Parameters

*elc_event_signal *p_elc_event_signal*

Pointer to an event link source setup structure.

Table 3.1 lists the content set in the event link source setup structure.

Table 3.1 Content Set in the Event Link Source Setup Structure (*p_elc_event_signal)

Constant Definition	Description
event_signal	Sets the event link source event signal. See Table 5.1 and Table 5.2 for the event signal definitions.
event_signal_input_port_edge	Specifies the valid edge for the single port and the input port group. See Table 5.7 for the valid edge definitions. This is valid when either a single port or an input port group is selected for the event signal.
event_signal_single_port	Specifies the pins allocated to the single port. See Table 5.5 for the single port definitions. This is valid when a single port is selected for an event signal.
event_signal_port_group_bit	Specifies, with 8 bits, the pins allocated as port group. Pins specified as 1 are allocated as a port group. This is valid when an input port group is selected for the event signal.

*elc_link_module_t *p_elc_module*

Pointer to an event link target setup structure.

Table 3.2 lists the content set in the event link target setup structure.

Table 3.2 Content Set in the Event Link Target Setup Structure (*p_elc_module)

Constant Definition	Description
link_module	Specifies the peripheral module to link. See Table 5.4 for the definitions of the peripheral modules that may be linked.
link_module_timer_operation	Specifies the timer operation when an event signal is input. See Table 5.8 for the definitions of the timer operations. This is valid when MTU, TMR, or CMT is specified as peripheral module to be linked.
link_module_output_port_level	Specifies the port output operation when an event signal is input. See Table 5.6 for the definitions of the port output operations. This is valid when either a single port or an output port group is selected for the peripheral module to be linked.
link_module_single_port	Specifies the pins allocated to the single port. See Table 5.5 for the single port definitions. This is valid when a single port is selected for the peripheral module to be linked.
link_module_port_group_bit	Specifies, with 8 bits, the pins allocated as port group. Pins specified as 1 are allocated as a port group. This is valid when either an input port group or an output port group is selected for the peripheral module to be linked.
link_module_port_buffer	Specifies write enable/disable for the port buffer. See Table 5.9 for the definitions of the write enable/disable settings. This is valid when an input port group is selected for the peripheral module to be linked.
link_module_interrupt_level	Specifies the interrupt priority level when interrupts are used. This is valid when interrupts are selected for the peripheral module to be linked.
link_module_callbackfunc	Specifies the callback function to be called when an interrupt occurs. This is valid when interrupts are selected for the peripheral module to be linked.

Return Values

ELC_SUCCESS   */* Normal completion */*
ELC_ERR_INVALID_ARG   */* Illegal argument */*

Properties

The declaration is located in r_elc_rx_if.h.

Description

This function sets up an event link. The event link source and event link target are specified as arguments.

Example

Example 1 Event link source: MTU and event link target: DA

This section presents an example in which the MTU is set up as the event link source and the DA is set up as the event link target.

[Event link source settings]

- event_signal;
  Specifies the event link source event signal. In example 1, MTU1 compare match 1A is specified as the event signal.

[Event link target settings]

- link_module
  Specifies the event link target. In example 1, DA0 is specified.

The source code for example 1 is shown below.

```
volatile elc_err_t     ret;
elc_event_signal_t         event_signal_info;
elc_link_module_t          event_module_info;

ret = R_ELC_Open();                                          /* Initializes the event link. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure to initialize occurs. */
  }
}

/* Link source settings */
event_signal_info.event_signal = ELC_MTU1_CMP1A;             /* Specifies MTU1 compare match 1A
                                                             as the link source event signal. */

/* Link target settings */
event_module_info.link_module = ELC_DA0;                     /* Specifies DA0 as the link target.*/

ret = R_ELC_Set(&event_signal_info, &event_module_info);     /* Creates an event link between the
                                                                link source and the link target.*/
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure in the event link settings occurs. */
  }
}
```

Example 2 Event link source: single port and event link target: port group

This section presents an example in which the single port is set up as the event link source and the port group is set up as the event link target.

[Event link source settings]

- event_signal;
  Specifies the event link source event signal. In example 2, an event signal consisting of input edge detection for single input port 2 is specified.

- event_signal_input_port_edge;
  Specifies input edge detection. In example 2, falling edge detection is specified.

- event_signal_single_port;
  Specifies which port is used as a single port. In example 2, PE3 is specified.

[Event link target settings]

- link_module
  Specifies the event link target. In example 2, output port group 1 (port B) is specified.

- link_module_output_port_level
  Specifies the operation when a port output is performed. In example 2, toggle output from the specified port is specified.

- link_module_port_group_bit
  Specifies which pins are used for the port specified as the port group. In example 2, PB0 to PB3 are specified.

- link_module_port_buffer
  Specifies whether writing to the PDBF register is enabled or disabled. In example 2, write enabled is specified.

The source code for example 2 is shown below.

```
volatile elc_err_t     ret;
elc_event_signal_t         event_signal_info;
elc_link_module_t          event_module_info;

ret = R_ELC_Open();                                  /* Initializes the event link. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure to initialize occurs. */
  }
}

/* Link source settings */
event_signal_info.event_signal = ELC_PORT_PSP2;      /* Specifies single input port 2 input edge
                                                        detection event signal as the link source
                                                        event signal. */
event_signal_info.event_signal_input_port_edge = ELC_EDGE_FALLING;  /* Specifies falling edge
                                                                       detection. */
event_signal_info.event_signal_single_port = ELC_PSB_PE3;    /* Specifies PE3. */

/* Link target settings */
event_module_info.link_module = ELC_OUT_PGR1;        /* Specifies output port group (port B)
                                                        as the link target. */
event_module_info.link_module_output_port_level = ELC_PORT_TOGGLE;  /* Specifies toggle output. */
event_module_info.link_module_port_group_bit = 0x0F; /* Specifies PB0 to PB3 as the port group. */
ret = R_ELC_Set(&event_signal_info, &event_module_info); /* Creates an event link between the link
                                                            source and the link target. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure in the event link settings occurs. */
  }
}
```

Example 3 Event link source: port group and event link target: MTU

This section presents an example in which the port group 1 is set up as the event link source and the MTU is set up as the event link target.

[Event link source settings]

- event_signal;
  Specifies the event link source event signal. In example 3, an event signal consisting of input edge detection for input port group 1 (port B) is specified.

- event_signal_input_port_edge;
  Specifies input edge detection. In example 3, falling edge detection is specified.

- link_module_port_group_bit
  Specifies which pins are used for the port specified as the port group. In example 3, PB4 to PB7 are specified.

[Event link target settings]

- link_module
  Specifies the event link target. In example 3, MTU1 is specified.

- link_module_timer_operation
  Specifies timer operation for the event link target. In example 3, input capture is specified.

The source code for example 3 is shown below.

```
volatile elc_err_t      ret;
elc_event_signal_t          event_signal_info;
elc_link_module_t           event_module_info;

ret = R_ELC_Open();                             /* Initializes the event link. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure to initialize occurs. */
  }
}

/* Link source settings *
event_signal_info.event_signal = ELC_PORT_PGR1; /* Specifies input port group 1 (port B) input
                                                   edge detection event signal as the link source
                                                   event signal. */
event_signal_info.event_signal_input_port_edge = ELC_EDGE_FALLING;  /* Specifies falling edge
                                                                       detection. */
event_signal_info.event_signal_port_group_bit = 0xF0;/* Specifies PB4 to PB7 as the port group. */

/* Link target settings */
event_module_info.link_module = ELC_MTU1;            /* Specifies MTU1 as the link target. */
event_module_info.link_module_timer_operation = ELC_TIMER_INPUT_CAPTURE;    /* Specifies input
                                                                               capture. */

ret = R ELC Set(&event signal info, &event module info);     /* Creates an event link between the
                                                                link source and the link target. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure in the event link settings occurs. */
  }
}
```

Example 4 Event link source: single port and event link target: ELC interrupt

This section presents an example in which the single port is set up as the event link source and the ELC interrupt is set up as the event link target.

[Event link source settings]

- event_signal;
  Specifies the event link source event signal. In example 4, an event signal consisting of input edge detection for single input port 1 is specified.

- event_signal_input_port_edge;
  Specifies input edge detection. In example 4, falling edge detection is specified.

- event_signal_single_port;
  Specifies which port is used as a single port. In example 4, port B3 is specified.

[Event link target settings]

- link_module
  Specifies the event link target. In example 4, interrupt 1 is specified.

- link_module_callbackfunc
  Registers the callback function to be called when an interrupt occurs.

The source code for example 4 is shown below.

```
volatile elc_err_t     ret;
elc_event_signal_t          event_signal_info;
elc_link_module_t           event_module_info;

ret = R_ELC_Open();                         /* Initializes the event link. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure to initialize occurs. */
  }
}

/* Link source settings */
event_signal_info.event_signal = ELC_PORT_PSP1;      /* Specifies single input port 1 input edge
                                                        detection event signal as the link source
                                                        event signal. */

event_signal_info.event_signal_input_port_edge = ELC_EDGE_FALLING;  /* Specifies falling edge
                                                                       detection. */
event_signal_info.event_signal_single_port = ELC_PSB_PE3;     /* Specifies port E3. */

/* Link target settings */
event_module_info.link_module = ELC_ ICU1;           /* Specifies ELC interrupt 1 as the link
                                                        target. */
event_module_info.link_module_interrupt_level = 3;   /* Sets the interrupt priority level to 3. */
event_module_info.link_module_callbackfunc = &elc_icu1_callbackfunc;    /* Registers a callback
                                                                           function. */

ret = R ELC Set(&event signal info, &event module info);     /* Creates an event link between the
                                                                link source and the link target.*/
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure in the event link settings occurs. */
  }
}

void elc_icu1_callbackfunc(void *pdata)
{
  /* User processing when an ELC interrupt occurs. */
}
```

Special Notes:

- This function should be called when the ELC is in the stopped state.
- The event link signals and link target peripheral modules that can be used differ with the device used.
- To start event link operation, set this module to the ELC operating state with the R_ELC_Control() function (ELC_CMD_START) described later in this document.
- See section 1.5, State Transition Diagram, for details on the ELC FIT module states.
- When an output port group is selected as the link target and bit rotate output is selected as the port group operation, an initial value must be written to the port buffers in advance.
  See section 4.4, Case C Setup Example, for the setup procedure.

3.3 R_ELC_Control()

This function transitions this module to the ELC operating state, clears the event link settings, and generates port buffer accesses and ELC software events.

Format

```
elc_err_t R_ELC_Control(
    const elc_eventlink_cmd_t    command /* Command specification */
    void                         *pdata
                             /* Value that corresponds to the specified command.*/
)
```

Parameters

const elc_eventlink_cmd_t *command*

Specifies the command

Table 3.3 lists the commands that can be specified.

Table 3.3 Commands

Command Definition	Command Description
ELC_CMD_START	Transitions to the ELC operating state.
ELC_CMD_STOP	Transitions to the ELC stopped state.
ELC_CMD_CLEAR_EVENTLINK	Clears the specified event link.
ELC_CMD_WRITE_PORTBUFFER	Writes a value to a port buffer.
ELC_CMD_READ_PORTBUFFER	Reads a value from a port buffer.
ELC_CMD_SOFTWARE_EVENT	Generates a software event signal.

void **pdata*

Used as the pointer to the arguments for each command.

The void pointer set to the argument is converted to the appropriate type according to the command used.

Table 3.4 lists the pointer settings for each command.

Table 3.4 Pointer Settings Corresponding to Each Command

Command Definition	Type Assigned to *pdata	Value Assigned to *pdata
ELC_CMD_START	—	Not used. Must be set to a FIT_NO_PTR.
ELC_CMD_STOP	—	Not used. Must be set to a FIT_NO_PTR.
ELC_CMD_CLEAR_EVENTLINK	elc_link_module_t*	Pointer variable set to the event link target peripheral module to be cleared. See Table 5.4 for the definitions of the peripheral modules specified.
ELC_CMD_WRITE_PORTBUFFER	elc_pdbf_access_t*	Pointer variable set to the port buffer to be accessed and the write value. See Table 5.11 for the definitions of the port buffers specified.
ELC_CMD_READ_PORTBUFFER	elc_pdbf_access_t*	Pointer variable set to the port buffer to be accessed.
ELC_CMD_SOFTWARE_EVENT	—	Not used. Must be set to a FIT_NO_PTR.

Return Values

ELC_SUCCESS */* Normal completion */*
ELC_ERR_INVALID_ARG */* Illegal argument */*

Properties

The declaration is located in r_elc_rx_if.h.

Description

Performs the operation specified by the command. The following commands can be specified.

- Start event link
  Transitions the event link to the operating state. Only *ELC_SUCCESS* is returned.

```
R_ELC_Control(ELC_CMD_START, FIT_NO_PTR); /* Transitions the event link to the operating state. */
```

- Stop event link
  Transitions the event link to the stopped state. Only *ELC_SUCCESS* is returned.

```
R_ELC_Control(ELC_CMD_STOP, FIT_NO_PTR);  /* Transitions the event link to the stopped state. */
```

- Clear event link settings
  Clears an event link set up with the R_ELC_Set() function.

```
volatile elc_err_t      ret;
elc_link_module_t       elc_clear_module = ELC_ICU1;      /* Selects ICU1 as the event link target
                                                             to be cleared. */

ret = R_ELC_Control(ELC_CMD_CLEAR_EVENTLINK, &elc_clear_module); /* Clears the ICU1 event link
                                                                    settings. */
```

- Write port buffer
  Writes the specified value to the port buffer.

```
volatile elc_err_t      ret;
elc_pdbf_access_t       pdbf_access;

pdbf_access.select_group = ELC_PORT_GROUP1;      /* Selects port group 1. */
pdbf_access.value = 0x0F;                        /* Sets up the write value for the port buffer.*/
ret = R_ELC_Control(ELC_CMD_WRITE_PORTBUFFER, &pdbf_access); /* Writes the value to the port
                                                                buffer. */
```

- Read port buffer
  Reads the value from the port buffer.
  The value read is stored in the value element of the elc_pdbf_access_t structure passed as an argument.
  Only use this value after confirming that the return value from the R_ELC_Control() function was *ELC_SUCCESS*.

```
volatile elc_err_t      ret;
uint8_t read_pdbf_value;
elc_pdbf_access_t       pdbf_access;

pdbf_access.select_group = ELC_PORT_GROUP1;                     /* Selects port group 1. */
ret = R_ELC_Control(ELC_CMD_READ_PORTBUFFER, &pdbf_access);     /* Reads from the port buffer. */
if( ELC_SUCCESS == ret ){                                    /* Did the port buffer read succeed? */
  read_pdbf_value = pdbf_access.value;                       /* Get value read from the port buffer.*/
}
```

- Generate software event
  Software events can be generated.
  When a software event is to be generated, first set the link source to software event with the R_ELC_Set() function. Only *ELC_SUCCESS* is returned.

```
R_ELC_Control(ELC_CMD_SOFTWARE_EVENT, FIT_NO_PTR);
```

Example

```
volatile elc_err_t     ret;
elc_event_signal_t         event_signal_info;
elc_link_module_t          event_module_info;
elc_module_t               elc_clear_module
uint8_t                    pipr;


ret = R_ELC_Open();                          /* Initializes the event link. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure to initialize occurs. */
  }
}
event_signal_info.event_signal = ELC_ELC_SEG;        /* Specifies software event as the link
                                                        source event signal. */
event_module_info.link_module = ELC_ICU1;            /* Specifies ELC interrupt 1 as the link
                                                        target. */
event_module_info.link_module_interrupt_level = 3;   /* Sets the interrupt priority level to 3. */
event_module_info.link_module_callbackfunc = &elc_icu1_callbackfunc;    /* Registers a callback
                                                                           function. */
ret = R_ELC_Set(&event_signal_info, &event_module_info);     /* Creates an event link between the
                                                                link source and the link target. */
if( ELC_SUCCESS != ret)
{
  while(1)
  {
    /* Error handling is performed if a failure in the event link settings occurs. */
  }
}

R_ELC_Control(ELC_CMD_START,FIT_NO_PTR);         /* Transitions the ELC to the operating state. */

R_ELC_Control(ELC_CMD_SOFTWARE_EVENT, FIT_NO_PTR);  /* Generates a software event. */

elc_clear_module = ELC_ICU1;                     /* Selects ELC interrupt 1 for the event link
                                                    target to be cleared. */
R_ELC_Control(ELC_CMD_CLEAR_EVENTLINK, &elc_clear_module);  /* Clears the event link setting for
                                                               ELC interrupt 1. */

R_ELC_Control(ELC_CMD_STOP, FIT_NO_PTR);         /* Transitions the ELC to the stopped state. */
```

Special Notes:

- If event link start is specified as the command, call this function if the ELC is in the stopped state.
- If event link stop is specified as the command, call this function if the ELC is in the operating state.
- If software event is specified as the command, call this function if the ELC is in the operating state.
- See section 1.5, State Transition Diagram, for details on the ELC FIT module states.

3.4 R_ELC_Close()

Sets the ELC to the terminated state.

Format

```
elc_err_t
    R_ELC_Close(void
)
```

Parameters

None

Return Values

ELC_SUCCESS */* Normal completion */*

Properties

The declaration is located in r_elc_rx_if.h.

Description

Closes the ELC module.

Example

```
R_ELC_Close();                          /* Terminates operation of the set event link. */
```

Special Notes:

None

3.5 R_ELC_GetVersion()

Returns the version number of the API.

Format

```
uint32_t
   R_ELC_GetVersion(void
)
```

Parameters

None

Return Values

Version Number

Properties

The declaration is located in r_elc_rx_if.h.

Description

Returns the version number of this API.

Special Notes:

None

4. Setup Procedure Examples

4.1 Setup Procedure

The ELC setup procedure is shown below.

Step 1. Set up the module used as the event link target.
If an output port group bit rotate operation is selected as the event link target, the port buffer is also set up.
If an RTC or LVD is used as the event link source, that RTC or LVD is also set up.
Step 2. Set up the ELC.
Step 3. Set up the module used as the event link target.
(This step is omitted if an RTC or LVD is used.)
Step 4. Start the module used as the event link source.

The following part of this section presents the setup procedures for three cases, A to C, using the ELC FIT module.

Case A: When a module other than an RTC or LVD is used as the event link source.
Case B: When an RTC or LVD module is used as the event link source.
Case C: When an output port group bit rotate operation is selected as the event link target.

4.2 Case A Setup Example

The settings for case A are performed in the sequence of first setting up the event link target module, then setting up the ELC, and then setting up the event link source module.

This section presents an ELC setup example under the following conditions.

- Target device: RX231 Group
- Event link source: MTU1 compare match 1A event signal
- Event link target: S12AD (Scan started by an event link signal from the ELC)

In this example, the MTU FIT module Rev. 1.20 and the S12AD FIT module Rev. 2.11 are used.

```
#include "r_elc_rx_if.h"
#include "r_s12ad_rx_if.h"
#include "r_mtu_rx_if.h"

void main(void);
void adc_int_callback(void *p_args);

void main()
{
  mtu_timer_chnl_settings_t  my_timer_cfg;
  mtu_err_t  mtu_result;

  elc_event_signal_t event_signal_info;
  elc_link_module_t event_module_info;
  elc_err_t elc_result;

  adc_cfg_t my_adc_cfg;
  adc_ch_cfg_t my_adc_ch_cfg;
  adc_err_t  adc_result;

  /* Event link target (S12AD) settings */
  my_adc_cfg.conv_speed = ADC_CONVERT_SPEED_DEFAULT;
  my_adc_cfg.alignment = ADC_ALIGN_RIGHT;
  my_adc_cfg.add_cnt = ADC_ADD_OFF;
  my_adc_cfg.clearing = ADC_CLEAR_AFTER_READ_OFF;
  my_adc_cfg.trigger = ADC_TRIG_SYNC_ELCTRG0N_OR_ELCTRG1N;   /* Specifies event input from the ELC
                                                                as the A/D conversion trigger. */
  my_adc_cfg.priority = 3;
  adc_result = R_ADC_Open(0, ADC_MODE_SS_ONE_CH, &my_adc_cfg, &adc_int_callback);

  my_adc_ch_cfg.chan_mask = ADC_MASK_CH0;
  my_adc_ch_cfg.chan_mask_groupb = ADC_MASK_GROUPB_OFF;
  my_adc_ch_cfg.priority_groupa = ADC_GRPA_PRIORITY_OFF;
  my_adc_ch_cfg.diag_method = ADC_DIAG_OFF;
  my_adc_ch_cfg.add_mask = 0;
  my_adc_ch_cfg.signal_elc = ADC_ELC_ALL_SCANS_DONE;
  adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_CHANS, &my_adc_ch_cfg);
  adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_TRIG, NULL);
```

```
    /* ELC settings */
    elc_result = R_ELC_Open();
    event_signal_info.event_signal = ELC_MTU1_CMP1A;
    event_module_info.link_module = ELC_S12AD;
    elc_result = R_ELC_Set(&event_signal_info, &event_module_info);
    /* When using multiple ELC settings, the R_ELC_Set() function should be called before calling
       the R_ELC_Control() function. */
    elc_result = R_ELC_Control(ELC_CMD_START, FIT_NO_PTR);

    /* Event link source (MTU1) settings */
    my_timer_cfg.clock_src.source          = MTU_CLK_SRC_INTERNAL;
    my_timer_cfg.clock_src.clock_edge      = MTU_CLK_RISING_EDGE;
    my_timer_cfg.clear_src                 = MTU_CLR_TIMER_A;
    my_timer_cfg.timer_a.actions.do_action = MTU_ACTION_REPEAT;
    my_timer_cfg.timer_a.freq              = 1000; //1KHz
    my_timer_cfg.timer_b.actions.do_action = MTU_ACTION_NONE;
    my_timer_cfg.timer_c.actions.do_action = MTU_ACTION_NONE;
    my_timer_cfg.timer_d.actions.do_action = MTU_ACTION_NONE;
    mtu_result = R_MTU_Timer_Open(MTU_CHANNEL_1, &my_timer_cfg, FIT_NO_FUNC);
    /* Starts operation of the event link source (MTU). */
    mtu_result = R_MTU_Control(MTU_CHANNEL_1, MTU_CMD_START, FIT_NO_PTR);

    while(1)
    {
      /* Main loop */
    }
}

void adc_int_callback(void *p_args)
{
    /* A/D conversion completing interrupt handling */
}
```

4.3 Case B Setup Example

In case B, the event link source is set up before setting up the ELC. The sample code for the case where the RTC (periodic event signal) is the event link source and the S12AD (scan started by a trigger from the ELC) is the event link target.

This section presents an ELC setup example under the following conditions.

- Target device: RX231 Group
- Event link source: RTC period (1 second)
- Event link target: S12AD (Scan started by an event link signal from the ELC)

In this example, the RTC FIT module Rev. 2.41 and the S12AD FIT module Rev. 2.11 are used.

```
#include "r_elc_rx_if.h"
#include "r_rtc_rx_if.h"
#include "r_s12ad_rx_if.h"

void main(void);
void adc_int_callback(void *p_args);
void rtc_int_callback(void *p_args);

void main()
{
  adc_cfg_t my_adc_cfg;
  adc_ch_cfg_t my_adc_ch_cfg;
  adc_err_t  adc_result;

  elc_event_signal_t event_signal_info;
  elc_link_module_t event_module_info;
  elc_err_t elc_result;

  rtc_init_t rtc_init;
  rtc_err_t rtc_result;

  /* Set the current date & time to be Aug 31, 2015 (Monday) 11:59:20pm */
  tm_t init_time =
  {
    20, //Second
    59, //Minutes
    23, //Hours
    31, //Day of month
    (8-1), //Month
    115, //Years since 1900
    1, //Day of week
    0, //
    0, //Daylight savings disabled
  };
```

```
    /* Event link source (RTC) settings */
    rtc_init.output_freq = RTC_OUTPUT_OFF;
    rtc_init.periodic_freq = RTC_PERIODIC_1_HZ;
    rtc_init.periodic_priority = 1;
    rtc_init.set_time = true;
    rtc_init.p_callback = rtc_int_callback;
    rtc_result = R_RTC_Open(&rtc_init, &init_time);

    /* Event link target (S12AD) settings */
    my_adc_cfg.conv_speed = ADC_CONVERT_SPEED_DEFAULT;
    my_adc_cfg.alignment = ADC_ALIGN_RIGHT;
    my_adc_cfg.add_cnt = ADC_ADD_OFF;
    my_adc_cfg.clearing = ADC_CLEAR_AFTER_READ_OFF;
    my_adc_cfg.trigger = ADC_TRIG_SYNC_ELCTRG0N_OR_ELCTRG1N;/* Specifies event input from the
                                                            ELC as the A/D conversion trigger. */
    my_adc_cfg.priority = 3;
    adc_result = R_ADC_Open(0, ADC_MODE_SS_ONE_CH, &my_adc_cfg, &adc_int_callback);

    my_adc_ch_cfg.chan_mask = ADC_MASK_CH0;
    my_adc_ch_cfg.chan_mask_groupb = ADC_MASK_GROUPB_OFF;
    my_adc_ch_cfg.priority_groupa = ADC_GRPA_PRIORITY_OFF;
    my_adc_ch_cfg.diag_method = ADC_DIAG_OFF;
    my_adc_ch_cfg.add_mask = 0;
    my_adc_ch_cfg.signal_elc = ADC_ELC_ALL_SCANS_DONE;
    adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_CHANS, &my_adc_ch_cfg);
    adc_result = R_ADC_Control(0, ADC_CMD_ENABLE_TRIG, NULL);

    /* ELC settings */
    elc_result = R_ELC_Open();
    event_signal_info.event_signal = ELC_RTC_PRD;
    event_module_info.link_module = ELC_S12AD;
    elc_result = R_ELC_Set(&event_signal_info, &event_module_info);
    /* When using multiple ELC settings, the R_ELC_Set() function should be called before calling
       the R_ELC_Control() function.*/
    elc_result = R_ELC_Control(ELC_CMD_START, FIT_NO_PTR);

    while(1)
    {
      /* Perform an A/D conversion at each period set up in the RTC. */
    }
}

void adc_int_callback(void *p_args)
{
    /* A/D conversion completing interrupt handling */
}

void rtc_int_callback(void *p_args)
{
    /* No processing required. */
}
```

4.4 Case C Setup Example

In case C, the initial value for the output port group is set before setting up the ELC. The sample code, which sets the event link source to be software events and sets up bit rotate operation for output port group 1, which is the event link target is shown below.

```
#include "r_elc_rx_if.h"

void main(void);

void main()
{
  elc_event_signal_t   event;
  elc_link_module_t    link;
  elc_pdbf_access_t    pdbf;
  elc_err_t        elc_err;

  PORTB.PDR.BYTE = 0x0F;      /* Sets the port group 1 (PORTB) pins to output. */
  PORTB.PODR.BYTE = 0x00;     /* Sets the port group 1 (PORTB) pins to low. */

  /* ELC settings */
  elc_err = R_ELC_Open();

  event.event_signal = ELC_ELC_SEG;   /* Sets software triggers to be the event link source. */
  link.link_module = ELC_OUT_PGR1;       /* Sets output port group 1 as the event link target. */
  link.link_module_output_port_level = ELC_PORT_ROTATE;     /* Rotate output */
  link.link_module_port_group_bit = (uint8_t)0x0F;   /* Sets the data to be rotated
                                                        in PB3to PB0.*/

  /* Sets rotate output as the initial value in the PDBF1 register before setting up the
     event link. */
  pdbf.select_group = ELC_PORT_GROUP1;
  pdbf.value = 0x08;
  elc_err = R_ELC_Control( ELC_CMD_WRITE_PORTBUFFER, &pdbf );

  elc err = R ELC Set( &event, &link );                  /* Sets up the event link. */
  elc_err = R_ELC_Control( ELC_CMD_START, FIT_NO_PTR );  /* Transitions the module to the ELC
                                                            operating state. */

  while(1)
  {
    R_ELC_Control( ELC_CMD_SOFTWARE_EVENT, FIT_NO_PTR );
    /* Each time a software trigger occurs, the value set in the PDBF1 register is rotated across
       PB3 to PB0, from MSB to LSB. */
  }
}
```

5. Appendices

5.1 Definitions

The table below lists the definitions used as the arguments to each function.

Table 5.1 Event Link Signal Definitions (1/3)

Definition	Description
ELC_MTU0_CMP0A	MTU0: compare match 0A event signal
ELC_MTU0_CMP0B	MTU0: compare match 0B event signal
ELC_MTU0_CMP0C	MTU0: compare match 0C event signal
ELC_MTU0_CMP0D	MTU0: compare match 0D event signal
ELC_MTU0_CMP0E	MTU0: compare match 0E event signal
ELC_MTU0_CMP0F	MTU0: compare match 0F event signal
ELC_MTU0_OVF	MTU0: overflow event signal
ELC_MTU1_CMP1A	MTU1: compare match 1A event signal
ELC_MTU1_CMP1B	MTU1: compare match 1B event signal
ELC_MTU1_OVF	MTU1: overflow event signal
ELC_MTU1_UDF	MTU1: underflow event signal
ELC_MTU2_CMP2A	MTU2: compare match 2A event signal
ELC_MTU2_CMP2B	MTU2: compare match 2B event signal
ELC_MTU2_OVF	MTU2: overflow event signal
ELC_MTU2_UDF	MTU2: underflow event signal
ELC_MTU3_CMP3A	MTU3: compare match 3A event signal
ELC_MTU3_CMP3B	MTU3: compare match 3B event signal
ELC_MTU3_CMP3C	MTU3: compare match 3C event signal
ELC_MTU3_CMP3D	MTU3: compare match 3D event signal
ELC_MTU3_OVF	MTU3: overflow event signal
ELC_MTU4_CMP4A	MTU4: compare match 4A event signal
ELC_MTU4_CMP4B	MTU4: compare match 4B event signal
ELC_MTU4_CMP4C	MTU4: compare match 4C event signal
ELC_MTU4_CMP4D	MTU4: compare match 4D event signal
ELC_MTU4_OVF	MTU4: overflow event signal
ELC_MTU4_UDF	MTU4: underflow event signal
ELC_CMT_CMP1	CMT1: compare match 1 event signal
ELC_TMR0_CMPA0	TMR0: compare match A0 event signal
ELC_TMR0_CMPB0	TMR0: compare match B0 event signal
ELC_TMR0_OVF	TMR0: overflow event signal
ELC_TMR1_CMPA1	TMR1: compare match A1 event signal
ELC_TMR1_CMPB1	TMR1: compare match B1 event signal
ELC_TMR1_OVF	TMR1: overflow event signal
ELC_TMR2_CMPA2	TMR2: compare match A2 event signal
ELC_TMR2_CMPB2	TMR2: compare match B2 event signal
ELC_TMR2_OVF	TMR2: overflow event signal
ELC_TMR3_CMPA3	TMR3: compare match A3 event signal
ELC_TMR3_CMPB3	TMR3: compare match B3 event signal
ELC_TMR3_OVF	TMR3: overflow event signal
ELC_RTC_PRD	RTC: periodic event signal*1
ELC_IWDT_UDF	IWDT: Underflow refresh error event signal
ELC_LPT_CMP0	LPT: compare match 0
ELC_LPT_CMP1	LPT: compare match 1

Note 1. When this event signal is used, the setup procedure sequence differs from that for other event signals. See case B in section 4, Setup Procedure Examples for details.

Table 5.38 Event Link Signal Definitions (38

Defined Name	Description
ELC_S12AD_WMELC	S12AD: Comparison condition met
ELC_S12AD_WUMELC	S12AD: Comparison condition not met
ELC_SCI5_ER5	SCI5: Error (reception error, error signal detected) event signal
ELC_SCI5_RX5	SCI5: Receive data full event signal
ELC_SCI5_TX5	SCI5: Transmit data empty event signal
ELC_SCI5_TE5	SCI5: Transmit complete event signal
ELC_RIIC0_ER0	RIIC0: Communication error, event occurrence signal
ELC_RIIC0_RX0	RIIC0: Receive data full event signal
ELC_RIIC0_TX0	RIIC0: Transmit data empty event signal
ELC_RIIC0_TE0	RIIC0: Transmit terminated event signal
ELC_RSPI0_ER0	RSPI0: Error (mode fault, overrun, underrun, or parity error) event signal
ELC_RSPI0_IDLE	RSPI0: Idle event signal
ELC_RSPI0_RX0	RSPI0: Receive data full event signal
ELC_RSPI0_TX0	RSPI0: Transmit data empty event signal
ELC_RSPI0_TE0	RSPI0: Transmit complete event signal
ELC_S12AD_S12AD0	S12AD: A/D conversion complete event signal
ELC_CMPB_CMPB0	Comparator B0: Comparison result change
ELC_CMPB_CMPB0_CMPB1	Comparator B0/B1 common comparison result change
ELC_LVD1_LVD1	LVD1: Voltage detection event signal*[1]
ELC_LVD2_LVD2	LVD2: Voltage detection event signal*[1]
ELC_DMAC0_DMAC0	DMAC0: Transfer complete event signal
ELC_DMAC1_DMAC1	DMAC1: Transfer complete event signal
ELC_DMAC2_DMAC2	DMAC2: Transfer complete event signal
ELC_DMAC3_DMAC3	DMAC3: Transfer complete event signal
ELC_DTC_DTC	DTC: Transfer complete event signal
ELC_CGC_OSTD	Clock generator circuit: Input edge detection event signal
ELC_PORT_PGR1	Input port group 1: input edge detection event signal
ELC_PORT_PGR2	Input port group 2: input edge detection event signal
ELC_PORT_PSP0	Single input port 0: input edge detection event signal
ELC_PORT_PSP1	Single input port 1: input edge detection event signal
ELC_PORT_PSP2	Single input port 2: input edge detection event signal
ELC_PORT_PSP3	Single input port 3: input edge detection event signal
ELC_ELC_SEG	Software event
ELC_DOC_DOPCF	DOC: Data calculation result signal

Note 1. When this event signal is used, the setup procedure sequence differs from that for other event signals. See case B in section 4, Setup Procedure Examples for details.

Table 5.39 Event Link Signal Definitions (39

Defined Name	Description
ELC_S12AD_S12AD1	S12AD1: A/D conversion complete event signal
ELC_CMT_CMPW	CMTW: channel0: compare match signal
ELC_TPU0_CMPA	TPU0: compare match A event signal
ELC_TPU0_CMPB	TPU0: compare match B event signal
ELC_TPU0_CMPC	TPU0: compare match C event signal
ELC_TPU0_CMPD	TPU0: compare match D event signal
ELC_TPU0_OVF	TPU0: overflow event signal
ELC_TPU1_CMPA	TPU1: compare match A event signal
ELC_TPU1_CMPB	TPU1: compare match B event signal
ELC_TPU1_OVF	TPU1: overflow event signal
ELC_TPU1_UDF	TPU1: underflow event signal
ELC_TPU2_CMPA	TPU2: compare match A event signal
ELC_TPU2_CMPB	TPU2: compare match B event signal
ELC_TPU2_OVF	TPU2: overflow event signal
ELC_TPU2_UDF	TPU2: underflow event signal
ELC_TPU3_CMPA	TPU3: compare match A event signal
ELC_TPU3_CMPB	TPU3: compare match B event signal
ELC_TPU3_CMPC	TPU3: compare match C event signal
ELC_TPU3_CMPD	TPU3: compare match D event signal
ELC_TPU3_OVF	TPU3: overflow event signal

Table 5.4 Event Link Target Peripheral Module Definitions

Defined Name	Description
ELC_MTU0	MTU0
ELC_MTU1	MTU1
ELC_MTU2	MTU2
ELC_MTU3	MTU3
ELC_MTU4	MTU4
ELC_CMT1	CMT1
ELC_ICU_LPT	ELC interrupt (LPT only)
ELC_TMR0	TMR0
ELC_TMR1	TMR1
ELC_TMR2	TMR2
ELC_TMR3	TMR3
ELC_CTSU	CTSU
ELC_S12AD	S12AD
ELC_DA0	DA0
ELC_ICU1	ELC interrupt 1
ELC_ICU2	ELC interrupt 2
ELC_OUT_PGR1	Output port group 1
ELC_OUT_PGR2	Output port group 2
ELC_IN_PGR1	Input port group 1
ELC_IN_PGR2	Input port group 2
ELC_PSP0	Single port 0
ELC_PSP1	Single port 1
ELC_PSP2	Single port 2
ELC_PSP3	Single port 3
ELC_CGC_LOCO	Clock generator circuit (clock source switched to LOCO)
ELC_POE	POE
ELC_CMTW0	CMTW0
ELC_TPU0	TPU0
ELC_TPU1	TPU1
ELC_TPU2	TPU2
ELC_TPU3	TPU3
ELC_S12AD1	S12AD1

Table 5.5 Event Connection Port Selection Definitions

Defined Name	Description
ELC_PSB_PB0	Selects port B0 as the single port
ELC_PSB_PB1	Selects port B1 as the single port
ELC_PSB_PB2	Selects port B2 as the single port
ELC_PSB_PB3	Selects port B3 as the single port
ELC_PSB_PB4	Selects port B4 as the single port
ELC_PSB_PB5	Selects port B5 as the single port
ELC_PSB_PB6	Selects port B6 as the single port
ELC_PSB_PB7	Selects port B7 as the single port
ELC_PSB_PE0	Selects port E0 as the single port
ELC_PSB_PE1	Selects port E1 as the single port
ELC_PSB_PE2	Selects port E2 as the single port
ELC_PSB_PE3	Selects port E3 as the single port
ELC_PSB_PE4	Selects port E4 as the single port
ELC_PSB_PE5	Selects port E5 as the single port
ELC_PSB_PE6	Selects port E6 as the single port
ELC_PSB_PE7	Selects port E7 as the single port

Table 5.6 Single Port/Port Group Operation by Event Link Signal Selection Definitions

Defined Name	Description
ELC_PORT_LOW	Low-level output from specified port
ELC_PORT_HIGH	High-level output from specified port
ELC_PORT_TOGGLE	Toggle output from specified port
ELC_PORT_BUFFER	Port buffer value output from specified port*1
ELC_PORT_ROTATE	Bit rotate output from specified port*1*2

Note 1. This may only be selected when output port group operation is selected. Do not select this when single port output is used.

Note 2. An initial value must be written in advance to the port buffers when output port group is selected as the event link target peripheral module and bit rotate output is selected as the port group output. See section 4.4, Case C Setup Example.

Table 5.7 External Input Signal Edge Selection Definitions

Defined Name	Description
ELC_EDGE_RISING	Detect rising edge on the external input signal
ELC_EDGE_FALLING	Detect falling edge on the external input signal
ELC_EDGE_RISING_AND_FALLING	Detect both rising and falling edges on the external input signal

Table 5.8 Timer Operation by Event Link Signal Selection Definitions

Defined Name	Description
ELC_TIMER_START	Timer start
ELC_TIMER_RESTART	Timer restart
ELC_TIMER_INPUT_CAPTURE	Input capture
ELC_TIMER_DISABLED	Event disabled

Table 5.9 Port Buffer Write Enable/Disable Setting Definitions

Defined Name	Description
ELC_PDBF_OVERWRITE_ENABLE	Enable port buffer write
ELC_PDBF_OVERWRITE_DISABLE	Disable port buffer write

Table 5.10 Definitions for the Commands Used with the Control Function

Defined Name	Description
ELC_CMD_START	Transitions to the ELC operating state
ELC_CMD_STOP	Transitions to the ELC stopped state
ELC_CMD_CLEAR_EVENTLINK	Clears the event link settings for the specified module
ELC_CMD_WRITE_PORTBUFFER	Writes a value to the port buffer
ELC_CMD_READ_PORTBUFFER	Reads a value from the port buffer
ELC_CMD_SOFTWARE_EVENT	Generates a software event signal.

Table 5.11 Port Group Selection Definitions

Defined Name	Description
ELC_PORT_GROUP1	Selects port group 1
ELC_PORT_GROUP2	Selects port group 2

5.2 Operation Confirmation Environment

This section describes operation confirmation environment for the ELC FIT module.

Table 5.12 Operation Confirmation Environment (Rev. 2.01)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2021-07 IAR Embedded Workbench for Renesas 4.20.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202004 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.20.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.01
Board used	Target board for RX140 (product No.: RTK5RX140xxxxxxxxx)

Table 5.13 Operation Confirmation Environment (Rev. 2.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.7.0 IAR Embedded Workbench for Renesas 4.14.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.201904 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.00
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxxx) Renesas Starter Kit for RX130 (product No.: RTK5005130xxxxxxxx) Renesas Starter Kit for RX231 (product No.: R0K505231xxxxxx) Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxxxx)

Table 5.14 Operation Confirmation Environment (Rev. 1.21)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.21

Table 5.15 Operation Confirmation Environment (Rev. 1.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.20
Board used	Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE)

5.3 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file "platform.h".

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"

- Using e^2 studio:

  Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note "Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_elc_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul. 20, 2016	—	First edition issued
1.10	Oct. 01. 2016	1,8,10,33, 35,36	Added support for RX65N
1.20	July. 24, 2017	—	Added support for RX130-512KB and RX65N-2MB.
		7	2.6 Interrupt Vector: Added.
		11	2.12 Adding the FIT Module to Your Project: Revised.
		39	5.2 Operation Confirmation Environment: Added.
		40	5.3 Troubleshooting: Added.
1.21	Apr. 01, 2019	—	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
		3	Changed 1.1 ELC FIT Module.
		5	Moved 1.3 API Overview.
		8	Moved 2.5 Header Files. Moved 2.6 Integer Types.
		9	Changed 2.8 Code Size.
		10	Changed 2.9 Parameters. Changed 2.10 Return Values.
		11	Changed 2.11 Callback Functions. Changed 2.12 Adding the FIT Module to Your Project.
		40	5.2 Operation Confirmation Environment: Added table for Rev.1.21.
2.00	Jun. 10, 2020	—	Added support for RX23W Modified comment of API function to Doxygen style. Update the following compilers - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX.
		1	Added Target Compilers.
		1	Related Documents: Deleted the following documents Firmware Integration Technology User's Manual (R01AN1833) RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723) RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)
		8	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		8	2.4 Interrupt Vector: RX23W added. Table 2.1 Interrupt Vector Used in the ELC FIT Module
		10	Changed 2.8 Code Size.
		13	Changed 2.12 Adding the FIT Module to Your Project.
		14..29	Deleted the Reentrant for each API in 3. API Functions.
		42	5.2 Operation Confirmation Environment: Added table for Rev.2.00.
		43	5.3 Troubleshooting: Changed.

Rev.	Date	Description	
		Page	Summary
2.00	Jun. 10, 2020	Program	Fixed the following. [Target device] All devices. [Description] Changed processing so that there is a register that may be accessed from multiple peripheral functions at the same time, and the atomicity of writing to that register can be ensured.
2.01	Jul. 31. 2021	—	Added support for RX140
		8	2.4 Interrupt Vector: RX140 added. Table 2.1 Interrupt Vector Used in the ELC FIT Module
		12	Changed 2.11 Callback Functions.
		37	LPT compare match 1 added. Table 5.15 Event Link Signal Definitions (1/3)

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can 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 must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. 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. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. 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, for example, 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.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.