APPLICATION NOTE

RX Family

LVD Module Using Firmware Integration Technology

R01AN1726EU0150
Rev. 1.50
Jul 9, 2015

Introduction

The RX family MCUs supported by this application note have a two channel Low Voltage Detection (LVD) circuit that can be used to monitor the VCC and/or an external voltage level. This document outlines the FIT-compliant API that can be used to configure and utilize the LVD module.

Target Device

The following is a list of devices that are currently supported by this API:

- **RX110, RX111, RX113 Groups**
- **RX210 Group**
- **RX231 Group**
- **RX63N Group**
- **RX64M, RX71M Groups**

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

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833)
- Board Support Package Firmware Integration Technology Module (R01AN1685)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723)
- Adding Firmware Integration Technology Modules to CubeSuite+ Projects (R01AN1826)

Contents

1. Overview

The RX family MCUs supported by this application note have two Low Voltage Detection circuit channels (Channel 1 and Channel 2) which can be configured to monitor VCC and/or the external voltage level on the CMPA2 pin. The external voltage monitoring is not available for RX63N, 64M, or 71M.

The channels can be configured to detect different voltage levels specific to each of the supported MCUs. The circuit can be configured to detect when the monitored voltage goes above or below a certain voltage or for both of these conditions. It is important to note that the circuit is not level sensitive. If a channel is configured to detect a voltage falling below 2.00v on VCC, but the current VCC is already at 1.80v, then the circuit will not detect this condition. On detecting the trigger condition, the channels can be configured to generate a RESET, an NMI, or an interrupt request. RX63N only supports RESET and NMI. For more operational details, refer to the relevant RX Hardware Manual.

1.1 API Overview

The following functions are included in this API:

Function	**Description**
R_LVD_Open()	Opens and configures the LVD channel
R_LVD_Close()	Closes the LVD channel
R_LVD_Control()	Change LVD Voltage Detection Level, clears LVD status, and returns the current detection state of the specified LVD channel based on the command passed.
R_LVD_GetVersion()	Returns at runtime the driver version number.

2. API Information

This driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires an RX MCU with following features:

- LVD module

2.2 Hardware Resource Requirements

This section details the hardware peripherals that this driver requires. Unless explicitly stated, these resources must be reserved for the driver and the user cannot use them.

2.2.1 LVD Registers

This module's API functions have the ability to write all LVD circuit registers. The function of this driver cannot be guaranteed if the user modifies these registers outside of the APIs provided with this application note.

2.3 Software Requirements

This driver requires the following FIT compliant packages:

Renesas Board Support Package (r_bsp) v2.90.

2.4 Limitations

This driver does not support the following features:

ELC linking

Digital filtering

2.5 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain version 2.02

2.6 Header Files

All API calls and their supporting interface definitions are located in r_lvd_rx_if.h. Compile time configurable options are located in r_lvd_rx_config.h. The r_lvd_rx_if.h file must be included in the user's application code.

2.7 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in *stdint.h*.

2.8 Configuration Overview

All configurable options that can be set at build time are located in the file “r_lvd_rx_config.h”. A summary of these settings are provided in the following table:

Configuration options in *r_lvd_rx_config.h*	
`#define LVD_CFG_PARAM_CHECKING_ENABLE`	Specifies whether parameter checking is to be done for each function call.

`#define LVD_CFG_INTERRUPT_PRIORITY_CHANNEL_1`
`#define LVD_CFG_INTERRUPT_PRIORITY_CHANNEL_2`
	Specifies the interrupt priority for the LVD channels

`#define LVD_CFG_STABILIZATION_CHANNEL_1`
`#define LVD_CFG_STABILIZATION_CHANNEL_2`
	Specify when LVD Stabilization should occur.
	Set to 0 for stabilization after VCC>Vdet
	Set to 1 for stabilization after assertion of LVD RESET

`#define LVD_CFG_VDET2_VCC_CMPA2`	Specify whether to monitor VCC or CMPA2 pin. Not available for all MCUs. Set to 0 for VCC monitoring Set to 1 for CMPA2 voltage monitoring

`#define LVD_CFG_CHANNEL_1_USED`
`#define LVD_CFG_CHANNEL_2_USED`
	Specify if a channel is used or not
	Set to 1 to enable code for that channel

2.9 Code Size

The code size is for the RX64M and RX71M, and is based on optimization level 2 and optimization type size, for the RXC toolchain in Section 2.5. The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options set in the module configuration header file.

ROM and RAM code sizes				
	One Channel		Two Channels	
	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX110	ROM: 781 bytes	ROM: 663 bytes	ROM: 1193 bytes	ROM: 1065 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes
RX111	ROM: 781 bytes	ROM: 663 bytes	ROM: 1193 bytes	ROM: 1065 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes
RX113	ROM: 781 bytes	ROM: 663 bytes	ROM: 1193 bytes	ROM: 1065 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes
RX210	ROM: 786 bytes	ROM: 669 bytes	ROM: 1194 bytes	ROM: 1067 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes

RX231	ROM: 777 bytes	ROM: 663 bytes	ROM: 1189 bytes	ROM: 1065 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes
RX63N	ROM: 664 bytes	ROM: 557 bytes	ROM: 945 bytes	ROM: 826 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes
RX64M	ROM: 681 bytes	ROM: 557 bytes	ROM: 965 bytes	ROM: 831 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes
RX71M	ROM: 681 bytes	ROM: 557 bytes	ROM: 965 bytes	ROM: 831 bytes
	RAM: 17 bytes	RAM: 17 bytes	RAM: 18 bytes	RAM: 18 bytes

Table 1: ROM and RAM code size

2.10 API Data Structures

This module uses the following data structures. They are defined in r_lvd_rx_if.h file.

The following structure is used with the API R_LVD_Open() to initialize a channel and start voltage monitoring.

```
typedef struct _lvd_config
{
    lvd_action_t         e_action;
    lvd_voltage_level_t  e_voltage_level;
    lvd_trigger_t        e_trigger;
}lvd_config_t;
```

The next structure is used with R_LVD_Control() to update the voltage level of a channel that is already initialized.

```
typedef struct st_lvd_lvl_cfg
{
    lvd_channel_t        e_channel;
    lvd_voltage_level_t  e_voltage_lvl;
} lvd_lvl_cfg_t;
```

This structure is used with R_LVD_Control() to get and clear the status of a channel.

```
typedef struct st_lvd_status
{
    lvd_channel_t        e_channel;
} lvd_status_t;
```

2.11 API Enums

This module uses the following enumerations. They are defined in r_lvd_rx_if.h file.

This enumeration defines the channel numbers.

```
typedef enum _lvd_channel_t
{
    LVD_CHANNEL_1 = 1,
    LVD_CHANNEL_2,
    LVD_CHANNEL_INVALID
} lvd_channel_t;
```

This enumeration defines the monitored voltage levels. Note that voltage level settings differ based on the MCU, the channel, and the external pin.

```
typedef enum
{
#if ((BSP_MCU_RX111 == 1) || (BSP_MCU_RX110 == 1) || (BSP_MCU_RX113 == 1))
    LVD_VOLTAGE_CH2_MIN   = 0,
    LVD_VOLTAGE_CH2_2_90v = 0,
    LVD_VOLTAGE_CH2_2_60v,
    LVD_VOLTAGE_CH2_2_00v,
    LVD_VOLTAGE_CH2_1_80v,
    LVD_VOLTAGE_CH2_MAX = LVD_VOLTAGE_CH2_1_80v,

    LVD_VOLTAGE_CH1_MIN   = 4,
    LVD_VOLTAGE_CH1_3_10v = 4,
    LVD_VOLTAGE_CH1_3_00v,
    LVD_VOLTAGE_CH1_2_90v,
    LVD_VOLTAGE_CH1_2_79v,
    LVD_VOLTAGE_CH1_2_68v,
    LVD_VOLTAGE_CH1_2_58v,
    LVD_VOLTAGE_CH1_2_48v,
    LVD_VOLTAGE_CH1_2_06v,
    LVD_VOLTAGE_CH1_1_96v,
    LVD_VOLTAGE_CH1_1_86v,
    LVD_VOLTAGE_CH1_MAX = LVD_VOLTAGE_CH1_1_86v,
    LVD_VOLTAGE_INVALID,
#endif

#if (BSP_MCU_RX210 == 1)
    LVD_VOLTAGE_CH1_MIN  = 0,
    LVD_VOLTAGE_CH1_4_15 = 0,
    LVD_VOLTAGE_CH1_4_00,
    LVD_VOLTAGE_CH1_3_85,
    LVD_VOLTAGE_CH1_3_70,
    LVD_VOLTAGE_CH1_3_55,
    LVD_VOLTAGE_CH1_3_40,
    LVD_VOLTAGE_CH1_3_25,
    LVD_VOLTAGE_CH1_3_10,
    LVD_VOLTAGE_CH1_2_95,
    LVD_VOLTAGE_CH1_2_80,
    LVD_VOLTAGE_CH1_2_65,
    LVD_VOLTAGE_CH1_2_50,
    LVD_VOLTAGE_CH1_2_35,
    LVD_VOLTAGE_CH1_2_20,
    LVD_VOLTAGE_CH1_2_05,
    LVD_VOLTAGE_CH1_1_90,
    LVD_VOLTAGE_CH1_MAX = LVD_VOLTAGE_CH1_1_90,
#if (LVD_CFG_VDET2_VCC_CMPA2 == 0)
    LVD_VOLTAGE_CH2_MIN  = 0,
    LVD_VOLTAGE_CH2_4_15 = 0,
    LVD_VOLTAGE_CH2_4_00,
    LVD_VOLTAGE_CH2_3_85,
    LVD_VOLTAGE_CH2_3_70,
    LVD_VOLTAGE_CH2_3_55,
    LVD_VOLTAGE_CH2_3_40,
    LVD_VOLTAGE_CH2_3_25,
    LVD_VOLTAGE_CH2_3_10,
    LVD_VOLTAGE_CH2_2_95,
    LVD_VOLTAGE_CH2_2_80,
    LVD_VOLTAGE_CH2_2_65,
    LVD_VOLTAGE_CH2_2_50,
    LVD_VOLTAGE_CH2_2_35,
    LVD_VOLTAGE_CH2_2_20,
    LVD_VOLTAGE_CH2_2_05,
    LVD_VOLTAGE_CH2_1_90,
    LVD_VOLTAGE_CH2_MAX = LVD_VOLTAGE_CH2_1_90,
#else
    LVD_VOLTAGE_CH2_MIN   = 1,
    LVD_VOLTAGE_CH2_1_33  = 1,
    LVD_VOLTAGE_CH2_MAX   = LVD_VOLTAGE_CH2_1_33,
#endif
    LVD_VOLTAGE_INVALID,
#endif

#if (BSP_MCU_RX231 == 1)
    LVD_VOLTAGE_CH1_MIN  = 0,
    LVD_VOLTAGE_CH1_4_29 = 0,
    LVD_VOLTAGE_CH1_4_14,
    LVD_VOLTAGE_CH1_4_02,
    LVD_VOLTAGE_CH1_3_84,
    LVD_VOLTAGE_CH1_3_10,
    LVD_VOLTAGE_CH1_3_00,
    LVD_VOLTAGE_CH1_2_90,
    LVD_VOLTAGE_CH1_2_79,
    LVD_VOLTAGE_CH1_2_68,
    LVD_VOLTAGE_CH1_2_58,
    LVD_VOLTAGE_CH1_2_48,
    LVD_VOLTAGE_CH1_2_20,
    LVD_VOLTAGE_CH1_1_96,
    LVD_VOLTAGE_CH1_1_86,
    LVD_VOLTAGE_CH1_MAX = LVD_VOLTAGE_CH1_1_86,
#if (LVD_CFG_VDET2_VCC_CMPA2 == 0)
    LVD_VOLTAGE_CH2_MIN  = 0,
    LVD_VOLTAGE_CH2_4_29 = 0,
    LVD_VOLTAGE_CH2_4_14,
    LVD_VOLTAGE_CH2_4_02,
    LVD_VOLTAGE_CH2_3_84,
    LVD_VOLTAGE_CH2_MAX = LVD_VOLTAGE_CH2_3_84,
#endif
    LVD_VOLTAGE_INVALID,
#endif

#if (BSP_MCU_RX63N == 1)
    LVD_VOLTAGE_CH1_MIN  = 10,
    LVD_VOLTAGE_CH1_2_95 = 10,
    LVD_VOLTAGE_CH1_MAX  = LVD_VOLTAGE_CH1_2_95,
    LVD_VOLTAGE_CH2_MIN  = 10,
    LVD_VOLTAGE_CH2_2_95 = 10,
    LVD_VOLTAGE_CH2_MAX  = LVD_VOLTAGE_CH2_2_95,
    LVD_VOLTAGE_INVALID,
#endif

#if ((BSP_MCU_RX64M == 1) || (BSP_MCU_RX71M == 1))
    LVD_VOLTAGE_CH2_MIN  = 9,
    LVD_VOLTAGE_CH2_2_99 = 9,
    LVD_VOLTAGE_CH2_2_92 = 10,
    LVD_VOLTAGE_CH2_2_85 = 11,
    LVD_VOLTAGE_CH2_MAX  = LVD_VOLTAGE_CH2_2_85,

    LVD_VOLTAGE_CH1_MIN  = 9,
    LVD_VOLTAGE_CH1_2_99 = 9,
    LVD_VOLTAGE_CH1_2_92 = 10,
    LVD_VOLTAGE_CH1_2_85 = 11,
    LVD_VOLTAGE_CH1_MAX  = LVD_VOLTAGE_CH1_2_85,
    LVD_VOLTAGE_INVALID,
#endif
} lvd_voltage_level_t;
```

This enumeration defines the actions. Note that IRQ action is not available for RX63N.

```
typedef enum
{
    LVD_ACTION_RESET = 0,
    LVD_ACTION_NMI,
#if ((BSP_MCU_RX111 == 1) || (BSP_MCU_RX110 == 1) || (BSP_MCU_RX113 == 1) || \
      (BSP_MCU_RX210 == 1) || (BSP_MCU_RX231 == 1))
    LVD_ACTION_IRQ,
#endif
    LVD_ACTION_POLL,
    LVD_ACTION_INVALID
} lvd_action_t;
```

This enumeration defines the commands for the R_LVD_Control() API.

```
typedef enum
{
    LVD_CMD_LEVEL_SET,
    LVD_CMD_STATUS_GET,
    LVD_CMD_STATUS_CLEAR,
    LVD_CMD_INVALID
}lvd_cmd_t;
```

2.12 Return Values

This module returns the following error values. They are defined as enumerations in r_lvd_rx_if.h file.

```
typedef enum _lvd_err
{
    LVD_SUCCESS,
    LVD_ERR_VDET,                               //Illegal value attempted for Vdet level
    LVD_ERR_NOT_INITIALIZED,                    //Channel has not been opened yet
    LVD_ERR_ILL_REINITIALIZATION,               //Attempt to open channel again without
                                                //first closing
    LVD_ERR_ILL_PARAM,                          //Illegal argument not specified in enum
    LVD_ERR_VCC_BELOW_AND_NOT_CROSSED = 0x10,  //VCC/CMPA2 is below Vdet and not crossed
                                                //yet
    LVD_ERR_VCC_BELOW_AND_CROSSED     = 0x11,  //VCC/CMPA2 is below Vdet and has crossed
                                                //at least once
    LVD_ERR_VCC_ABOVE_AND_NOT_CROSSED = 0x12,  //VCC/CMPA2 is above Vdet and not crossed
                                                //yet
    LVD_ERR_VCC_ABOVE_AND_CROSSED     = 0x13,  //VCC/CMPA2 is above Vdet and has crossed
```

```
                                               //at least once
    LVD_ERR_DISABLED                           //LVD Channel is disabled
}lvd_err_t;
```

2.13 Adding Module to Your Project

The driver must be added to an existing e2Studio project. It is best to use the e2Studio FIT plugin to add the driver to your project as that will automatically update the include file paths for you. Alternatively, the driver can be imported from the archive that accompanies this application note and manually added by following these steps:

2.13.1 Manual Installation

1. This application note is distributed with a zip file package that includes the FIT LVD support module in its own folder r_lvd_rx.
2. Unzip the package into the location of your choice.
3. In a file browser window, browse to the directory where you unzipped the distribution package and locate the r_lvd_rx folder.
4. Open your e2Studio workspace.
5. In the e2Studio project explorer window, select the project that you want to add the LVD module to.
6. Drag and drop the r_lvd_rx folder from the browser window (or copy/paste) into your e2Studio project at the top level of the project.
7. Update the source search/include paths for your project by adding the paths to the module files:
    a. Navigate to the "Add directory path" control:
        i. 'project name'->properties->C/C++ Build->Settings->Compiler->Source -Add (green + icon)
    b. Add the following paths:
        i. "${workspace_loc:/${ProjName}/r_lvd_rx}"
        ii. "${workspace_loc:/${ProjName}/r_lvd_rx/src}"
8. Configure the driver for your application.
    a. Locate the r_lvd_rx_config_reference.h file in the r_lvd_rx/ref/ source folder in your project and copy it to your project's r_config folder.
    b. Change the name of the copy in the r_config folder to r_lvd_rx_config.h

2.13.2 All Installations

Make the required configuration settings by editing the copied r_lvd_rx_config.h file. See Configuration Overview.

The LVD module uses the r_bsp package for certain MCU information and support functions. The r_bsp package is easily configured through the platform.h header file which is located in the r_bsp folder. To configure the r_bsp package, open up platform.h and uncomment the #include for the board you are using. Only one board may be uncommented. Be sure that any other board #includes are commented out.

3. API Functions

3.1 Summary

The following functions are included in this API:

Function	**Description**
R_LVD_Open()	Opens and configures the LVD channel
R_LVD_Close()	Closes the LVD channel
R_LVD_Control()	Change LVD Voltage Detection Level, clears LVD status, and returns the current detection state of the specified LVD channel based on the command passed.
R_LVD_GetVersion()	Returns at runtime the driver version number.

3.2 R_LVD_Open()

This function initializes and enables the specified LVD channel. This function takes arguments that are used to configure the exception type (NMI, IRQ, RESET) or polling, the monitoring voltage level, the callback function and the detection type.

Format

```
   lvd_err_t R_LVD_Open(lvd_channel_t     e_channel,
                        lvd_config_t      *p_cfg,
                        void              (*pcallback)(void *));
```

Parameters

e_channel

Enumerated channel number to be initialized

p_cfg

Specifies the configuration parameters for the channel

p_callback

Function to be called from the LVD NMI or IRQ. This can be set to a NULL if the e_action argument is RESET or polled mode.

Return Values

LVD_SUCCESS	*Successful*
LVD_ERR_VDET	*Illegal value attempted for Vdet level*
LVD_ERR_ILL_REINITIALIZATION	*Channel is opened again without first closing*
LVD_ERR_ILL_PARAM	*Illegal argument*

Properties

Prototyped in file “r_lvd_rx_if.h”

Description

This function initializes and enables the specified LVD channel. The configuration parameters are enumerated in the r_lvd_rx_if.h file. Voltage levels are different for each supported MCU. Further, they depend on the channel and the external pin configuration. The channel can be configured to generate reset, NMI, and IRQ exceptions. For exception handling, a valid callback function must be provided. In case of polling or reset, callback function is not needed.
.

Reentrant

No.

Example

```
   lvd_err_t      err;
   lvd_config_t   channel1_cfg;

   channel1_cfg.e_action = LVD_ACTION_RESET;
   channel1_cfg.e_trigger = LVD_TRIGGER_FALL;
   channel1_cfg.e_voltage_level = LVD_VOLTAGE_CH1_2_06v;
   err = R_LVD_Open(LVD_CHANNEL_1, &channel1_cfg, NULL);
```

Special Notes

The Hardware Manual specifies that the voltage detection levels of the two channels should not overlap. Overlapping will cause only either one of the LVD channels circuits to generate the event. This API does not check for the overlapping condition; it is up to the user to prevent configuring the two channels this way.

RX63N only supports reset and NMI exceptions. External voltage pin is not available in RX63N.

3.3 R_LVD_Close()

This function closes the specified LVD channel.

Format

```
   lvd_err_t        R_LVD_Close(lvd_channel_t e_channel);
```

Parameters

e_channel

Enumerated channel number to be closed

Return Values

Return Values

LVD_SUCCESS	*Successful*
LVD_ERR_ILL_PARAM	*Illegal argument*

Properties

Prototyped in file “r_lvd_rx_if.h”

Description

This function closes and disables the specified LVD channel. Voltage monitoring for the specified channel is stopped.
.

Reentrant

No.

Example

```
lvd_err_t err;

err =  R_LVD_Close(LVD_CHANNEL_1);
```

Special Notes

If the channel was configured for an NMI interrupt, closing the channel will prevent the peripheral from sending out an interrupt request to the MCU and thus prevent NMI interrupts. However, the NMI enable bit itself will not be cleared because once it is set this bit cannot be cleared.

3.4 R_LVD_Control()

This function acts as a routing function that calls the appropriate sub-function and routes the arguments based on the command passed to it. The commands supported by this version of the API allows for voltage detection level modification, clearing the detection status of either channel and checking the detection status of either channel.

Format

```
lvd_err_t        R_LVD_Control(lvd_cmd_t const e_cmd, void * param);
```

Parameters

e_cmd

Enumerated command number to be executed

param

Pointer to the structure with any required data to complete execution of the command.

Return Values

LVD_SUCCESS	*Successful*
LVD_ERR_VDET	*Illegal value attempted for Vdet level*
LVD_ERR_NOT_INITIALIZED	*Channel has not been opened yet*
LVD_ERR_ILL_PARAM	*Illegal argument*
LVD_ERR_VCC_BELOW_AND_NOT_CROSSED	*VCC/CMPA2 is below Vdet and not crossed yet*
LVD_ERR_VCC_BELOW_AND_CROSSED	*VCC/CMPA2 is below Vdet and has crossed at least once*
LVD_ERR_VCC_ABOVE_AND_NOT_CROSSED	*VCC/CMPA2 is above Vdet and not crossed yet*
LVD_ERR_VCC_ABOVE_AND_CROSSED	*VCC/CMPA2 is above Vdet and has crossed at least once*
LVD_ERR_DISABLED	*LVD Channel is disabled*

Properties

Prototyped in file “r_lvd_rx_if.h”

Description

Depending on the command argument, one of the following internal actions are executed.

3.4.1 LVD_CMD_LEVEL_SET

This command can be used to modify the detection voltage level on any of the supported LVD channels. Before changing the levels, the channel interrupts and LVD monitoring will be temporarily disabled, detection level changed, and interrupts enabled.

Special Notes

This command clears the detection status bit for the selected channel. Before executing this command, the channel should be opened by using the R_LVD_Open() function.

3.4.2 LVD_CMD_STATUS_GET

This function returns the status of the voltage detection circuit for the specified channel.

Special Notes

Before executing this command, the channel should be opened by using the R_LVD_Open() function

3.4.3 LVD_CMD_STATUS_CLEAR

This command clears the detection status for the specified channel.

Special Notes

The status will be set when the conditions for detection circuit that was configured are met. It is important that the status of the LVD circuits are cleared by calling this function once the application has serviced the condition. If this function is not called after the LVD event has occurred, further events cannot be detected until the status is cleared. Before executing this command, the channel should be opened by using the R_LVD_Open() function.

Reentrant
No.

Example
```
lvd_lvl_cfg_t     lvd_level_config;
lvd_level_config.e_channel = LVD_CHANNEL_2;
lvd_level_config.e_voltage_lvl = LVD_VOLTAGE_CHANNEL2_2_00v;
R_LVD_Control(LVD_CMD_LEVEL_SET, &lvd_level_config);        // Change the voltage level

lvd_status_t      status_get
status_get.e_channel = LVD_CHANNEL_1;
stat_1 = R_LVD_Control(LVD_CMD_STATUS_GET, &status_get);    // Get the channel status

lvd_status_t      status_clear;
status_clear.e_channel = LVD_CHANNEL_2;
R_LVD_Control(LVD_CMD_STATUS_CLEAR,&status_clear);          // Clear the Status bit
```

Special Notes
For special notes on each command refer to the sub-sections above.

3.5 R_LVD_GetVersion()

This function returns the driver version number at runtime.

Format

```
uint32_t  R_LVD_GetVersion(void);
```

Parameters

None.

Return Values

Version number.

Properties

Prototyped in file "r_lvd_rx_if.h"

Description

Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number. For example, version 4.25 is represented as 0x00040019.

Reentrant

No.

Example

```
uint32_t   version;

version = R_LVD_GetVersion();
```

4. Usage Examples

4.1 Configuring LVD Channel 1 for RESET

The following example shows how to use the API to configure channel 1 to generate a reset (LVD_ACTION_RESET) when VCC falls below (LVD_TRIGGER_FALL) 2.06 volts (LVD_VOLTAGE_CHANNEL1_2_06v). The callback argument can be a NULL in this case since the MCU will reset when the detection condition is met.

Note that if the LVD is configured **after** VCC has fallen below the detection voltage, then the LVD will not detect this and thus the reset will not be generated.

```
void main(void)
{
  lvd_err_t       err;
  lvd_config_t    channel1_cfg;

  channel1_cfg.e_action = LVD_ACTION_RESET;
  channel1_cfg.e_trigger = LVD_TRIGGER_FALL;
  channel1_cfg.e_voltage_level =LVD_VOLTAGE_CHANNEL1_2_06v;
  err = R_LVD_Open(LVD_CHANNEL_1, &channel1_cfg, NULL);
}
```

4.2 Configuring LVD Channel 2 for CMPA2 Monitoring

When configuring channel 2 to monitor the voltage on the CMPA2 pin instead of VCC, set the below macro to (1) in r_lvd_rx_config.h.

```
#define LVD_CFG_VDET2_VCC_CMPA2           (1)
```

The following example shows how to set up channel 2 to monitor CMPA2 for 2.60v (LVD_VOLTAGE_CHANNEL2_2_60v) and to generate an interrupt (LVD_ACTION_IRQ) if the voltage crosses higher or lower (LVD_TRIGGER_BOTH). The interrupt will call the callback function (lvd2_isr_cb). This function has to be provided by the user.

Note that in the ISR, the status is cleared by R_LVD_Control() with the LVD_CMD_STATUS_CLEAR argument. This is because if the status is not cleared, subsequent LVD events on that channel will not trigger an interrupt.

The argument passed to the callback function is of type *lvd_int_cb_args_t*. The API populates the "vector" field of this argument with the channel number allowing the callback function to determine the vector source. This feature may be useful if the same callback function is used for both channels.

```
void main(void)
{
  lvd_err_t       err;
  lvd_config_t    channel2_cfg;

  channel2_cfg.e_action = LVD_ACTION_IRQ;
  channel2_cfg.e_trigger = LVD_TRIGGER_BOTH;
  channel2_cfg.e_voltage_level =LVD_VOLTAGE_CHANNEL2_2_60v;
  err = R_LVD_Open(LVD_CHANNEL_2, &channel2_cfg, lvd2_isr_cb);
}


void lvd2_isr_cb(void *pArgs)
{
  lvd_status_t          status_clear;
  lvd_int_cb_args_t     cb_args;

  status_clear.e_channel = LVD_CHANNEL_2;
  R_LVD_Control(LVD_CMD_STATUS_CLEAR,&status_clear);

  cb_args = pArgs;
  if (cb_arg->vector == BSP_INT_SRC_LVD2)
  {
     /* User code */
  }
}
```

4.3 Checking LVD Channel Status

Once the LVD channel is configured, it is possible to check the status of the channel by using the LVD_CMD_STATUS_GET argument in the R_LVD_Control() function. Below is an example of how to determine the LVD Channel status for channel 2.

```
void main(void)
{
  lvd_err_t       err;
  lvd_err_t       status;
  lvd_config_t    channel2_cfg;
  lvd_status_t    status_get;

  channel2_cfg.e_action = LVD_ACTION_NMI;
  channel2_cfg.e_trigger = LVD_TRIGGER_BOTH;
  channel2_cfg.e_voltage_level =LVD_VOLTAGE_CHANNEL2_2_60v;
  err = R_LVD_Open(LVD_CHANNEL_2, &channel2_cfg, lvd2_isr_cb);

  status_get.e_channel = LVD_CHANNEL_2;
  status = R_LVD_Control(LVD_CMD_STATUS_GET, &status_get);
}
```

4.4 Changing Voltage Detection Level

Once the LVD channel is configured, it is possible to modify the voltage detection level of the channel by using the LVD_CMD_LEVEL_SET argument in the R_LVD_Control() function. Below is an example of how to change the detection level on the LVD Channel 2 from an initial configuration of 2.60v to 2.00v.

```
void main(void)
{
  lvd_err_t       err;
  lvd_config_t    channel2_cfg;
  lvd_lvl_cfg_t   lvd_level_config;

  channel2_cfg.e_action = LVD_ACTION_NMI;
  channel2_cfg.e_trigger = LVD_TRIGGER_BOTH;
  channel2_cfg.e_voltage_level =LVD_VOLTAGE_CHANNEL2_2_60v;
  err = R_LVD_Open(LVD_CHANNEL_2, &channel2_cfg, lvd2_isr_cb);

  lvd_level_config.e_channel = LVD_CHANNEL_2;
  lvd_level_config.e_voltage_lvl = LVD_VOLTAGE_CHANNEL2_2_00v;
  err = R_LVD_Control(LVD_CMD_LEVEL_SET, &lvd_level_config);
}
```

5. Demo Project

Demo projects are complete stand-alone programs. They include function main() that utilize the module and its dependent modules (e.g. r_bsp). This FIT module has the following demo projects.

5.1 lvd_demo_rskrx231

The demos shows how LVD channel 1 triggers for a falling voltage level. When the user lowers the MCU power voltage from 3.3V to below 3.10V, the NMI triggers the user callback to signify a low voltage.

Power the board with E1 debugger. Use an adjustable power supply to power the MCU by UC_VCC. To do this, remove the zero ohms resistor between pins 1 and 2 of J8, populate J8 with 2-pin header (if not populated) and connect the adjustable power supply to J8 pin 2. Make sure not to exceed allowable voltage level for the MCU. When finished be sure to jumper J8 pins 1 and 2 for normal operation.

Run the code. A green LED (LED0) should light up showing the demo is running. Gradually lower the voltage to below 3.10V. When the LVD triggers for the falling voltage, the yellow LED (LED1) should light up.

5.2 lvd_demo_rskrx64m, lvd_demo_rskrx71m

The demos shows how LVD channel 1 triggers for a falling voltage level. When the user lowers the MCU power voltage from 3.3V to below 2.92V, the NMI triggers the user callback to signify a low voltage.

Power the board with E1 debugger. Use an adjustable power supply to power the MCU by UC_VCC. To do this, remove the zero ohms resistor between pins 1 and 2 of J21, populate J21 with 2-pin header (if not populated) and connect the adjustable power supply to J21 pin 2. Make sure not to exceed allowable voltage level for the MCU. When finished be sure to jumper J21 pins 1 and 2 for normal operation.

Run the code. A green LED (LED0) should light up showing the demo is running. Gradually lower the voltage to below 2.92 V. When the LVD triggers for the falling voltage, the yellow LED (LED1) should light up.

5.3 Adding the Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select File > Import > General > Existing Projects into Workspace, then click "Next". From the Import Projects dialog, choose the "Select archive file" radio button. "Browse" to the FITDemos subdirectory, select the desired demo zip file, then click "Finish".

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	July 15.13	—	First edition issued.
1.20	Apr 17, 14	Various	Added support for RX110, RX210, and RX63N.
1.30	Dec 22, 2014	Various	Support for RX113. Code size section added, and doc. properties adjusted. Demo added.
1.40	Mar 18, 2015	Various	Added support for RX64M and RX71M, with demos.
1.50	Jul 9, 2015	1,5,8,21	Added support for RX231

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0