

# RZ/A1LU Group

R01AN4314EJ0200

Rev.2.00

## Touch Panel Utility

Jun 29, 2018

### Introduction

This application note describes the operation of a FreeRTOS based, embedded firmware project which provides a development platform for a Touch Panel Utility using the RIIC driver.

### Target Device

This application note is covering the usage of the touch panel utility application, which is in of itself, not device or OS specific. However, the sample project containing this application is running FreeRTOS10 and contains RZ/A1LU drivers.

### Contents

1. Specifications .....	4
2. Operation Check Conditions .....	4
3. Application Functionality .....	4
4. Software Description.....	4
4.1 Operation Outline .....	4
4.2 Inserting the application into a project .....	5
4.3 Modifying the application .....	6
4.4 Files .....	7
5. Data Structure Index .....	8
5.1 Data Structures .....	8
6. File Index.....	9
6.1 File List.....	9
7. Data Structure Documentation .....	10
7.1 LCDEVT_ENTRY Struct Reference.....	10
7.1.1 Data Fields .....	10
7.1.2 Detailed Description .....	10
7.1.3 Field Documentation .....	10
7.2 TP_TouchEvent_st Struct Reference .....	11
7.2.1 Data Fields .....	11
7.2.2 Detailed Description .....	11
7.2.3 Field Documentation .....	11
7.3 TP_TouchFinger_st Struct Reference.....	12
7.3.1 Data Fields .....	12
7.3.2 Detailed Description .....	12
7.3.3 Field Documentation .....	12
7.4 TPEVT_COORDINATES Struct Reference .....	13

7.4.1	Data Fields .....	13
7.4.2	Detailed Description .....	13
7.4.3	Field Documentation .....	13
7.5	TPEVT_ENTRY Struct Reference .....	14
7.5.1	Data Fields .....	14
7.5.2	Detailed Description .....	14
7.5.3	Field Documentation .....	14
8.	File Documentation .....	15
8.1	Icd_controller_if.h File Reference .....	15
8.1.1	Macros .....	15
8.1.2	Typedefs .....	15
8.1.3	Enumerations .....	15
8.1.4	Functions .....	15
8.1.5	Detailed Description .....	16
8.1.6	Macro Definition Documentation .....	16
8.1.7	Typedef Documentation .....	16
8.1.8	Enumeration Type Documentation .....	16
8.1.9	Function Documentation .....	16
8.2	Icd_ft5x06.h File Reference .....	20
8.2.1	Data Structures .....	20
8.2.2	Macros .....	20
8.2.3	Enumerations .....	20
8.2.4	Functions .....	20
8.2.5	Variables .....	21
8.2.6	Detailed Description .....	21
8.2.7	Macro Definition Documentation .....	21
8.2.8	Enumeration Type Documentation .....	22
8.2.9	Function Documentation .....	22
8.2.10	Variable Documentation .....	25
8.3	Icd_ft5x06_int.h File Reference .....	26
8.3.1	Macros .....	26
8.3.2	Functions .....	26
8.3.3	Detailed Description .....	26
8.3.4	Macro Definition Documentation .....	26
8.3.5	Function Documentation .....	26
8.4	tp.h File Reference .....	28
8.4.1	Data Structures .....	28
8.4.2	Macros .....	28
8.4.3	Enumerations .....	28
8.4.4	Functions .....	28

8.4.5	Variables .....	30
8.4.6	Detailed Description .....	30
8.4.7	Macro Definition Documentation.....	30
8.4.8	Enumeration Type Documentation .....	31
8.4.9	Function Documentation.....	32
8.4.10	Variable Documentation .....	37
8.5	tp_if.h File Reference .....	37
8.5.1	Data Structures .....	37
8.5.2	Macros.....	37
8.5.3	Typedefs .....	37
8.5.4	Enumerations .....	37
8.5.5	Functions .....	37
8.5.6	Detailed Description .....	38
8.5.7	Macro Definition Documentation.....	38
8.5.8	Typedef Documentation .....	39
8.5.9	Enumeration Type Documentation .....	39
8.5.10	Function Documentation.....	39

## 1. Specifications

Touch Panel utility controls a touch panel via RIIC device controller(ch1), which is implemented on RZ/A1LU.

## 2. Operation Check Conditions

To ensure the touch screen application is enabled in software, please check that:

```
/* Enable control for src/application/app_touchscreen sample application */
#define R_SELF_INSERT_APP_TOUCH_SCREEN (R_OPTION_ENABLE)
```

is present inside of “application\_cfg.h”.

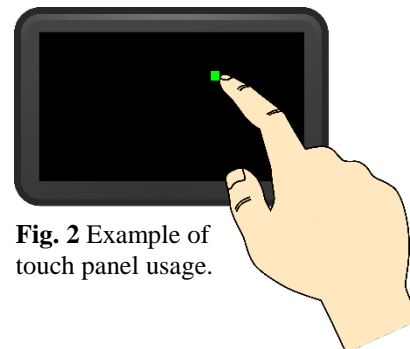
## 3. Application Functionality

The functionality of the touch panel sample application is to detect a touch event and draw a small green rectangle at the coordinates of the event, see figure 2. Additionally, the sample application will update the console to display the coordinates of the event and a categorisation of the event type.

The sample application will place the event into one of three categories:

<b>UP</b>	Finger is no longer placed on the touch panel
<b>DOWN</b>	Finger is currently placed on touch panel, but is stationary.
<b>MOVE</b>	Finger is currently placed on touch panel, but has moved.

**Fig. 3** Table of specifications



**Fig. 2** Example of touch panel usage.

The image displayed in figure 4 shows the expected console output upon detection of an event. This is displayed in the format:

**Touch: x = \$\$ , y = ££ [category]**

Where \$\$ represents the X coordinate value, ££ represents the Y coordinate value and [category] holds the event categorisation.

```
COM3 - PuTTY
RZ/A1LU Web Engine Demo Ver.3.02.0322
Copyright (C) Renesas Electronics Europe.

REE> tsdemo
Touch panel sample program start
I2c driver loaded initialising demo
Touch the LCD to display the contact point on this console

Press any key to terminate demo

Touch Demo: supporting 1 touch points
Touch: x = 246 , y = 132 [ UP ]
```

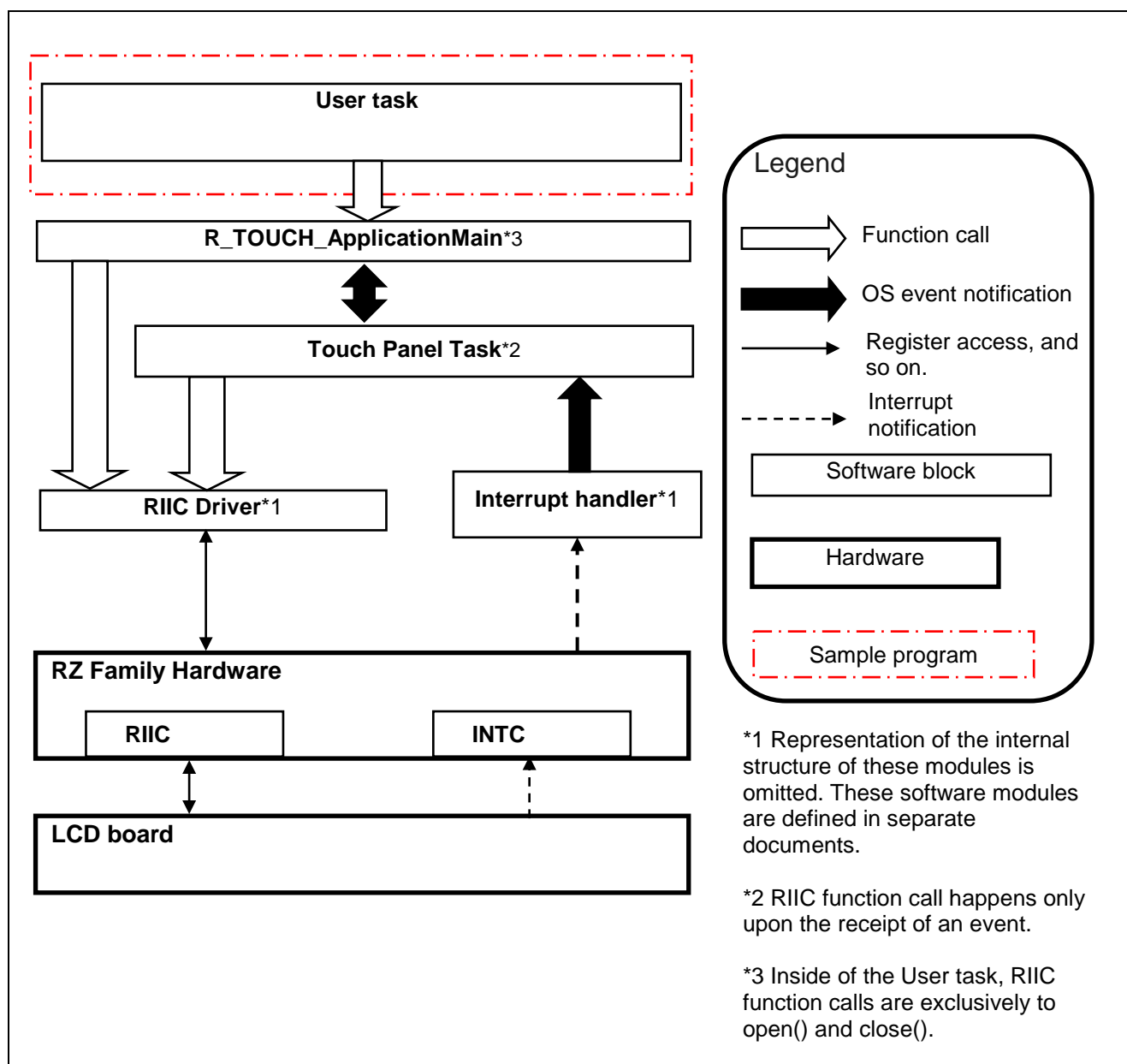
**Fig. 4** Expected console output of sample application.

## 4. Software Description

This section of the application note will describe and explain the usage of the touch screen sample application.

### 4.1 Operation Outline

Figure 5 outlines the overall structure of the software modules used in this sample application and their interaction with the target hardware.



**Fig. 5** Figure Touch Panel Utility System Block Diagram

As can be seen in the figure 5 the expectation is for the user to create a task which calls the “R\_TOUCH\_ApplicationMain()” function.

The “R\_TOUCH\_ApplicationMain()” function is responsible for opening drivers and creating a “Touch Panel Task”, this task holds all subsequent responsibility for interaction with the touch panel.

## 4.2 Inserting the application into a project

It is assumed the specifications outlined in section 1 of this document have been met.

The touch panel sample application can be started by calling the “R\_TOUCH\_ApplicationMain()” function (found in the “r\_touch\_capacitive.c” file), it is expected that this will be called from inside of a user created task.

Shown below is a control flowchart of the “R\_TOUCH\_ApplicationMain()” function.

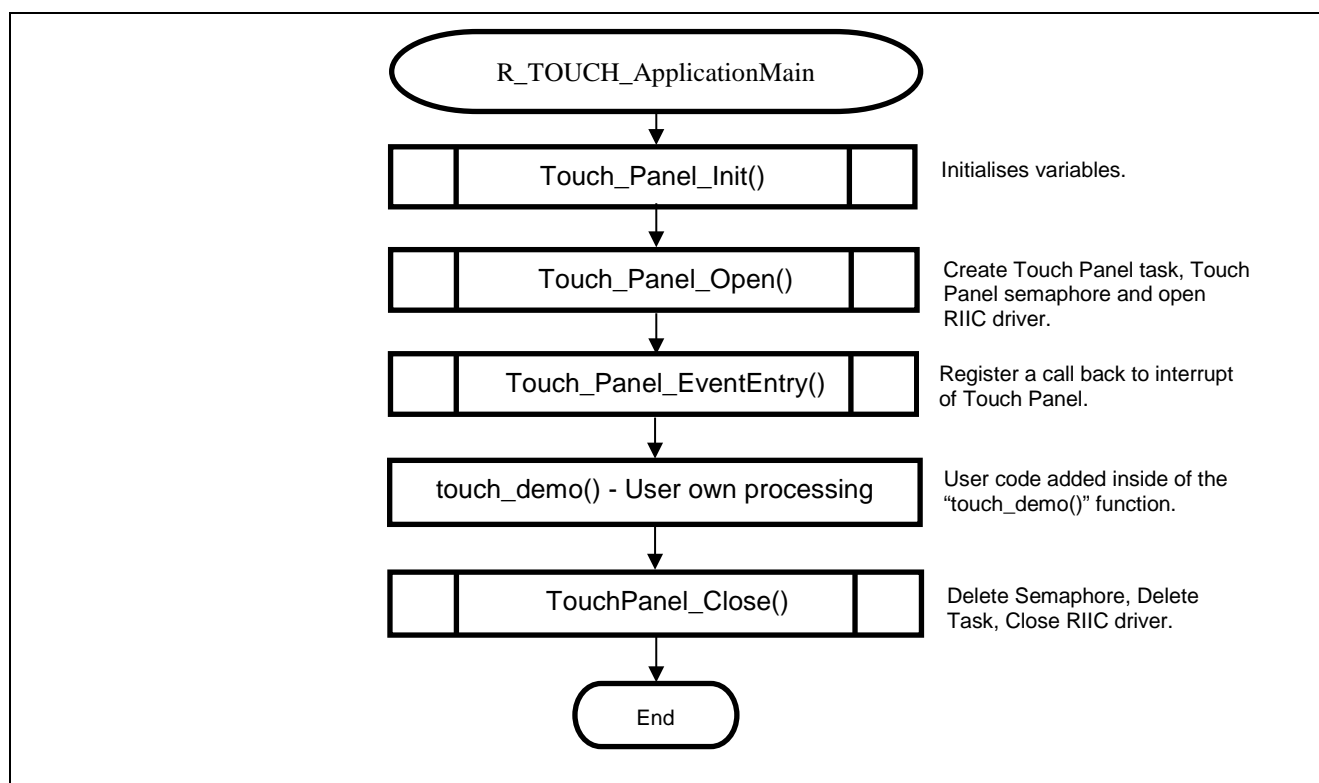


Fig. 6 Simplified Control Flow Scheme of Touch Panel utility

### 4.3 Modifying the application

As a user, there are two primary sections of code suggested for modification:

The first section of code is the “`touch_demo()`” function seen in figure 6. The currently implemented “`touch_demo()`” function is responsible for initialising the touch screen and then blocking the “User Task” seen in figure 5 until receipt of a character through the serial console.

```

static void touch_demo (void *parameters)
{
    fprintf(s_dsp_console->p_out, "Touch Demo: supporting %2-d touch points\r\n", 1);

    /* initialize screen */
    R_TOUCH_init_screen();

    /* START - User Places Concurrent Code Here */
    while (control(R_DEVLINK_FilePtrDescriptor(s_dsp_console->p_in), CTL_GET_RX_BUFFER_COUNT,
    NULL) == 0)
    {
        R_OS_TaskSleep(5);
    }
    /* END */

    /* un-initialize screen */
    R_TOUCH_uninit_screen();

    fgetc(s_dsp_console->p_in);
}
  
```

The expectation is for the user to place any operations desired to run concurrently with the “Touch Panel Task” inside of the “`touch_demo()`” function, between the “`R_TOUCH_init_screen()`” and “`R_TOUCH_uninit_screen()`” function calls.

The second section of code for user modification is the “Touch Panel Task”, which is found inside of “`tp_task.c`”. This task is where the user should insert any code related to the processing of information to and from the touch screen.

## 4.4 Files

Software	
—src	
—arm	
—FreeRTOS	
—supplierX	
—renesas	
—configuration	
application_cfg.h	- Application settings defined
—application	
—app_touchscreen	
r_drawrectangle.c	- Functions to draw on-screen rectangle
r_touch_capacitive.c	- Handles communication with capacitive controllers
—inc	
r_display_init.h	- Initialise display controller init
r_draw_rectangle.h	- Draw Rectangle header
r_image_config.h	- Defines for image correction
r_lcd_panel.h	- Defines for LCD panel type connected
_touch_capacitive.h	- Touch_capacitive header
r_vdc_portsetting.h	- VDC Port Setting header
—lcd	
r_stream2_tft_ch0.h	- LCD panel for VDC5 Channel 0 header
r_stream2_tft_clk.h	- Definitions used for LCD Clock
—video	
r_display_init.c	- Display Setting for VDC
vdc_portsetting.c	- VDC Port Setting Functions
—app_2	
—app_3...	
—compiler	
—configuration	
—drivers	
—middleware	
—touch	
—inc	
lcd_controller_if.h	- LCD Control interface header
lcd_if.h	- LCD API header
tp_if.h	- Touch Panel utility interface header
—src	
—lcd_controller	
r_lcd_controller_if.c	- LCD Control interface
—FT5x06	
lcd_ft5x06.c	- FT5x06 Control Operation
lcd_ft5x06.h	- FT5x06 Control header
lcd_ft5x06_int.c	- FT5x06 Control interrupt handler
lcd_ft5x06_int.h	- FT5x06 Control interrupt header
—touch	
tp.c	- Touch Panel utility internal operation
tp.h	- Touch Panel utility internal header
tp_if.c	- Touch Panel utility interface
tp_task.c	- Touch Panel utility task operation
tp_task.h	- Touch Panel utility task header

5. Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

LCDEVT_ENTRY .....	10
TP_TouchEvent_st .....	11
TP_TouchFinger_st .....	12
TPEVT_COORDINATES .....	13
TPEVT_ENTRY .....	14



6. File Index

6.1 File List

Here is a list of all files with brief descriptions:

lcd\_controller\_if.h (LCD Driver API header) ..... 15

lcd\_ft5x06.h (LCD Driver internal header) ..... 20

lcd\_ft5x06\_int.h (LCD Driver internal header) ..... 26

tp.h (TouchPanel Driver internal header) ..... 28

tp\_if.h (TouchPanel Driver API header) ..... 37

## 7. Data Structure Documentation

### 7.1 LCDEVT\_ENTRY Struct Reference

```
#include <lcd_ft5x06.h>
```

#### 7.1.1 Data Fields

- **LcdEvt\_EntryType mode**
- **LcdCBFunc function**
- **LcdEvt\_LockState evtlock**

---

#### 7.1.2 Detailed Description

Event entry struct

Definition at line 113 of file lcd\_ft5x06.h.

---

#### 7.1.3 Field Documentation

(1) **LcdEvt\_LockState evtlock**

Event lock state

Definition at line 116 of file lcd\_ft5x06.h.

(2) **LcdCBFunc function**

Definition at line 115 of file lcd\_ft5x06.h.

(3) **LcdEvt\_EntryType mode**

The type of touch panel event entry

Definition at line 114 of file lcd\_ft5x06.h.

---

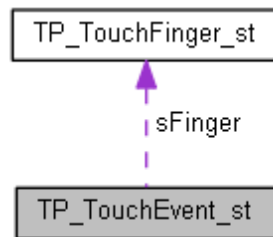
(4) **The documentation for this struct was generated from the following file:**

- **lcd\_ft5x06.h**

## 7.2 TP\_TouchEvent\_st Struct Reference

```
#include <tp_if.h>
```

Collaboration diagram for TP\_TouchEvent\_st:



### 7.2.1 Data Fields

- `TP_TouchFinger_st sFinger [TP_TOUCHNUM_MAX]`

### 7.2.2 Detailed Description

Definition at line 71 of file `tp_if.h`.

### 7.2.3 Field Documentation

- (1) `TP_TouchFinger_st sFinger[TP_TOUCHNUM_MAX]`

Definition at line 72 of file `tp_if.h`.

- (2) The documentation for this struct was generated from the following file:

- `tp_if.h`

### 7.3 TP\_TouchFinger\_st Struct Reference

```
#include <tp_if.h>
```

#### 7.3.1 Data Fields

- **TpEvt\_EntryType** eState
- **uint16\_t** unPosX
- **uint16\_t** unPosY

---

#### 7.3.2 Detailed Description

Definition at line 65 of file tp\_if.h.

---

#### 7.3.3 Field Documentation

(1) **TpEvt\_EntryType** eState

Definition at line 66 of file tp\_if.h.

(2) **uint16\_t** unPosX

Definition at line 67 of file tp\_if.h.

(3) **uint16\_t** unPosY

Definition at line 68 of file tp\_if.h.

---

(4) The documentation for this struct was generated from the following file:

- **tp\_if.h**

## 7.4 TPEVT\_COORDINATES Struct Reference

```
#include <tp.h>
```

### 7.4.1 Data Fields

- `int32_t x`
- `int32_t y`

---

### 7.4.2 Detailed Description

Coordinate structure

Definition at line 115 of file `tp.h`.

---

### 7.4.3 Field Documentation

(1) `int32_t x`

x-coordinate [pixel]

Definition at line 116 of file `tp.h`.

(2) `int32_t y`

y-coordinate [pixel]

Definition at line 117 of file `tp.h`.

---

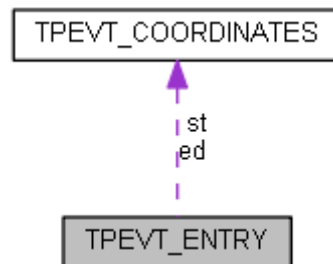
(3) The documentation for this struct was generated from the following file:

- `tp.h`

## 7.5 TPEVT\_ENTRY Struct Reference

```
#include <tp.h>
```

Collaboration diagram for TPEVT\_ENTRY:



### 7.5.1 Data Fields

- **TpEvt\_EntryType mode**
- **TPEVT\_COORDINATES st**
- **TPEVT\_COORDINATES ed**
- **void(\* function)(int\_t, TP\_TouchEvent\_st \*)**
- **TpEvt\_LockState evtlock**

### 7.5.2 Detailed Description

Event entry struct

Definition at line 121 of file tp.h.

### 7.5.3 Field Documentation

(1) **TPEVT\_COORDINATES ed**

The lower-right coordinates of the rectangular area in which touch event can be received. [pixel]

Definition at line 124 of file tp.h.

(2) **TpEvt\_LockState evtlock**

Event lock state

Definition at line 126 of file tp.h.

(3) **void(\* function)(int\_t, TP\_TouchEvent\_st \*)**

Event notification callback function pointer

Definition at line 125 of file tp.h.

(4) **TpEvt\_EntryType mode**

The type of touch panel event entry

Definition at line 122 of file tp.h.

(5) **TPEVT\_COORDINATES st**

The upper-left coordinates of the rectangular area in which touch event can be received. [pixel]

Definition at line 123 of file tp.h.

(6) **The documentation for this struct was generated from the following file:**

- **tp.h**

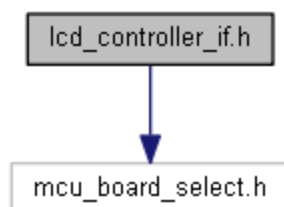
## 8. File Documentation

### 8.1 lcd\_controller\_if.h File Reference

LCD Driver API header.

```
#include "mcu_board_select.h"
```

Include dependency graph for lcd\_controller\_if.h:



#### 8.1.1 Macros

- `#define LCD_SLAVE_ADDRESS (0x38 << 1)`

#### 8.1.2 Typedefs

- `typedef void(* LcdCBFunc) (void *)`

#### 8.1.3 Enumerations

- `enum LcdEvt_EntryType { LCDEVT_ENTRY_NONE = 0x0000, LCDEVT_ENTRY_TP = 0x0001, LCDEVT_ENTRY_ALL = 0x0001 }`

#### 8.1.4 Functions

- `void R_LCD_Init (void)`  
*Sets the LCD board initialization counter (nLcdInitCnt) to 0.*
- `int_t R_LCD_Open (const uint32_t unIrqLv, const int16_t nTskPri, const uint32_t unTskStk)`  
*Opens a communication environment with the LCD board.*  
*This function enables the user to perform multiple open operations.*
- `int_t R_LCD_Close (void)`  
*Closes a communication environment with the LCD board.*  
*When LCD\_Open is used to perform multiple open operations, this function must be called the same number of times.*
- `uint8_t R_LCD_WriteCmd (const uint16_t unDevAddr, const uint8_t uCmd, const uint8_t uData, const uint32_t unSize)`
- `uint8_t R_LCD_ReadCmd (const uint16_t unDevAddr, const uint8_t uCmd, uint8_t *puData, const uint32_t unSize)`  
*Receives data from the LCD board via the RIIC.*
- `int_t R_LCD_EventEntry (const LcdEvt_EntryType eType, const LcdCBFunc function)`  
*Registers an LCD board event.*
- `int_t R_LCD_EventErase (const int_t nId)`  
*Removes an LCD board event.*
- `int_t R_LCD_StartInt (const LcdEvt_EntryType eType)`  
*Removes masking of specified interrupt type.*
- `int_t R_LCD_Restart (void)`  
*Reset LCD board.*
- `void R_LCD_ReadVersion (uint8_t *puData)`
- `void R_LCD_SetBacklight (const uint8_t uLevel)`  
*Set bright level of backlight.*
- `void R_LCD_SetBuzzer (const uint8_t uScale)`

*Set scale of buzzer.*

### 8.1.5 Detailed Description

LCD Driver API header.

Rev: 30 Date:: 2016-12-21 12:02:44 +0900#

### 8.1.6 Macro Definition Documentation

#### (1) **#define LCD\_SLAVE\_ADDRESS (0x38 << 1)**

LCD slave address

Definition at line 48 of file lcd\_controller\_if.h.

### 8.1.7 Typedef Documentation

#### (1) **typedef void(\* LcdCBFunc) (void \*)**

Definition at line 41 of file lcd\_controller\_if.h.

### 8.1.8 Enumeration Type Documentation

#### (1) **enum LcdEvt\_EntryType**

The type of touch panel event entry

##### (a) **Enumerator:**

LCDEVT_ENTR Y_NONE	None
LCDEVT_ENTR Y_TP	None
LCDEVT_ENTR Y_ALL	All

Definition at line 57 of file lcd\_controller\_if.h.

```

57      {
58      LCDEVT_ENTRY_NONE = 0x0000,
59      LCDEVT_ENTRY_TP   = 0x0001,
61      LCDEVT_ENTRY_ALL  = 0x0001
62 } LcdEvt_EntryType ;

```

### 8.1.9 Function Documentation

#### (1) **int\_t R\_LCD\_Close (void )**

Closes a communication environment with the LCD board.

When LCD\_Open is used to perform multiple open operations, this function must be called the same number of times.

##### (a) **Return values:**

NONE	
------	--



- (2) **int\_t R\_LCD\_EventEntry (const LcdEvt\_EntryType eType, const LcdCBFunc function)**

Registers an LCD board event.

(a) **Parameters:**

in	<i>eType</i>	Specified Interrupt type
in	<i>function</i>	Call-back function

(b) **Return values:**

0- (LCDEVT_ENTRY _MAX-1)	registration value
-1	event registration failure

- (3) **int\_t R\_LCD\_EventErase (const int\_t nId)**

Removes an LCD board event.

(a) **Parameters:**

in	<i>nId</i>	Event ID
----	------------	----------

(b) **Return values:**

NONE	
------	--

- (4) **void R\_LCD\_Init (void )**

Sets the LCD board initialization counter (nLcdInitCnt) to 0.

R\_LCD\_Init

(a) **Return values:**

NONE	
------	--

- (5) **int\_t R\_LCD\_Open (const uint32\_t unIrqLv, const int16\_t nTskPri, const uint32\_t unTskStk)**

Opens a communication environment with the LCD board.

This function enables the user to perform multiple open operations.

(a) **Parameters:**

in	<i>unIrqLv</i>	IRQ interrupt priority (0 to 255) Sets the GIC interrupt priority
in	<i>nTskPri</i>	Task Priority Sets the value of osPriority type.
in	<i>unTskStk</i>	Not Used.

(b) **Return values:**

0	Normal end
-1	Open error

- (6) **uint8\_t R\_LCD\_ReadCmd (const uint16\_t unDevAddr, const uint8\_t uCmd, uint8\_t \* puData, const uint32\_t unSize)**

Receives data from the LCD board via the RIIC.

(a) **Parameters:**

in	<i>unDevAddr</i>	LCD Device Address
in	<i>uCmd</i>	Not Used
in	<i>*puData</i>	Receive data buffer pointer
out	<i>unSize</i>	Receive Data Length

(b) **Return values:**

0	normal end
-1	data send processing error

- (7) **void R\_LCD\_ReadVersion (uint8\_t \* puData)**

(a) **Parameters:**

out	<i>*puData</i>	: pointer to receive buffer
-----	----------------	-----------------------------

(b) **Return values:**

0	
---	--

- (8) **int\_t R\_LCD\_Restart (void )**

Reset LCD board.

(a) **Return values:**

0	
---	--

- (9) **void R\_LCD\_SetBacklight (const uint8\_t uLevel)**

Set bright level of backlight.

(a) **Parameters:**

in	<i>uLevel</i>	bright level
----	---------------	--------------

(b) **Return values:**

None.	
-------	--

- (10) **void R\_LCD\_SetBuzzer (const uint8\_t uScale)**

Set scale of buzzer.

(a) **Parameters:**

in	<i>uScale</i>	scale
----	---------------	-------

(b) **Return values:**

<i>None.</i>	
--------------	--

(11) **int\_t R\_LCD\_StartInt (const LcdEvt\_EntryType eType)**

Removes masking of specified interrupt type.

(a) **Parameters:**

in	<i>eType</i>	Not Used
----	--------------	----------

(b) **Return values:**

<i>0</i>	event successfully removed
<i>-1</i>	event removal failure

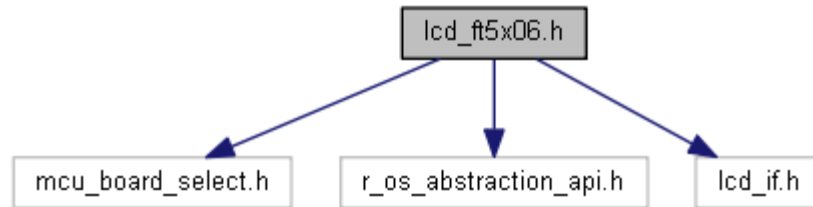
(12) **uint8\_t R\_LCD\_WriteCmd (const uint16\_t unDevAddr, const uint8\_t uCmd, const uint8\_t uData, const uint32\_t unSize)**

## 8.2 lcd\_ft5x06.h File Reference

LCD Driver internal header.

```
#include "mcu_board_select.h"
#include "r_os_abstraction_api.h"
#include "lcd_if.h"
```

Include dependency graph for lcd\_ft5x06.h:



### 8.2.1 Data Structures

- struct **LCDEVT\_ENTRY**

### 8.2.2 Macros

- #define **DBG\_LEVEL\_OT** (-1) /\* onetime debug \*/
- #define **DBG\_LEVEL\_DEF** (0) /\* default \*/
- #define **DBG\_LEVEL\_ERR** (1) /\* error \*/
- #define **DBG\_LEVEL\_MSG** (2) /\* message \*/
- #define **DBG\_LEVEL\_LOG** (3) /\* log \*/
- #define **DBG\_LEVEL\_DBG** (4) /\* debug \*/
- #define **DBG\_LEVEL** (**DBG\_LEVEL\_ERR**)
- #define **DBG\_printf\_OT** printf
- #define **DBG\_printf\_DEF** printf
- #define **DBG\_printf\_ERR** printf
- #define **DBG\_printf\_MSG** 1 ? (int32\_t) 0 : printf
- #define **DBG\_printf\_LOG** 1 ? (int32\_t) 0 : printf
- #define **DBG\_printf\_DBG** 1 ? (int32\_t) 0 : printf
- #define **SCOPE\_STATIC** static
- #define **LCDEVT\_ENTRY\_MAX** (1)

### 8.2.3 Enumerations

- enum **LcdEvt\_LockState** { **LCD\_EVT\_UNLOCK** = 0, **LCD\_EVT\_LOCK** }

### 8.2.4 Functions

- int\_t **LCD\_Ft5x06\_Open** (const uint32\_t unIrqLv, int16\_t nTskPri, uint32\_t unTskStk)  
*Opens the communication environment with the FT5x06.*
- int\_t **LCD\_Ft5x06\_Close** (void)  
*Closes the communication environment with the FT5x06.*
- uint8\_t **LCD\_Ft5x06\_WriteCmd** (const uint16\_t unDevAddr, const uint8\_t uData, const uint32\_t unSize)  
*Sends data to the FT5x06 via the RIIC DeviceController ch1.*
- uint8\_t **LCD\_Ft5x06\_ReadCmd** (const uint16\_t unDevAddr, uint8\_t \*puData, const uint32\_t unSize)  
*Reads data from the FT5x06 via the RIIC DeviceController ch1.*
- int\_t **LCD\_Ft5x06\_EventEntry** (const **LcdEvt\_EntryType** eType, const **LcdCBFunc** function)  
*Registers in the event management structure a call-back function linked to an interrupt from the FT5x06. After registration finishes, the LCD interrupt is enabled and the event ID is sent as a return value.*
- int\_t **LCD\_Ft5x06\_EventErase** (const int\_t nId)

*Removes the registration information for the specified event ID from the event management structure.*

- `int_t LCD_Ft5x06_StartInt` (const `LcdEvt_EntryType` eType)  
*Removes masking of specified interrupt type.*
- `LCDEVT_ENTRY * LCD_Ft5x06_GetEventTable` (const `int_t` nId)  
*Get assigned callback event.*
- `int32_t LCD_Ft5x06_SendEvtMsg` (const `uint32_t` unEvtFlg)  
*Send event message to synchronism.*
- `int32_t LCD_Ft5x06_WaitEvtMsg` (void)  
*Wait event message to synchronism.*
- `void LCD_Ft5x06_ClearEvtMsg` (const `uint32_t` unEvtFlg)  
*Clear assigned event flag.*

### 8.2.5 Variables

- `int32_t sLcdSemIdAcc`

### 8.2.6 Detailed Description

LCD Driver internal header.

Rev: 30 Date:: 2016-12-21 12:02:44 +0900#

### 8.2.7 Macro Definition Documentation

- (1) **#define DBG\_LEVEL (DBG\_LEVEL\_ERR)**

Definition at line 56 of file `lcd_ft5x06.h`.

- (2) **#define DBG\_LEVEL\_DBG (4) /\* debug \*/**

Definition at line 54 of file `lcd_ft5x06.h`.

- (3) **#define DBG\_LEVEL\_DEF (0) /\* default \*/**

Definition at line 50 of file `lcd_ft5x06.h`.

- (4) **#define DBG\_LEVEL\_ERR (1) /\* error \*/**

Definition at line 51 of file `lcd_ft5x06.h`.

- (5) **#define DBG\_LEVEL\_LOG (3) /\* log \*/**

Definition at line 53 of file `lcd_ft5x06.h`.

- (6) **#define DBG\_LEVEL\_MSG (2) /\* message \*/**

Definition at line 52 of file `lcd_ft5x06.h`.

- (7) **#define DBG\_LEVEL\_OT (-1) /\* onetime debug \*/**

Definition at line 49 of file `lcd_ft5x06.h`.

- (8) **#define DBG\_printf\_DBG 1 ? (int32\_t) 0 : printf**

Definition at line 85 of file `lcd_ft5x06.h`.

(9) **#define DBG\_printf\_DEF printf**

Definition at line 63 of file lcd\_ft5x06.h.

(10) **#define DBG\_printf\_ERR printf**

Definition at line 68 of file lcd\_ft5x06.h.

(11) **#define DBG\_printf\_LOG 1 ? (int32\_t) 0 : printf**

Definition at line 80 of file lcd\_ft5x06.h.

(12) **#define DBG\_printf\_MSG 1 ? (int32\_t) 0 : printf**

Definition at line 75 of file lcd\_ft5x06.h.

(13) **#define DBG\_printf\_OT printf**

Definition at line 58 of file lcd\_ft5x06.h.

(14) **#define LCDEVT\_ENTRY\_MAX (1)**

The max number of event entry

Definition at line 96 of file lcd\_ft5x06.h.

(15) **#define SCOPE\_STATIC static**

Definition at line 92 of file lcd\_ft5x06.h.

## 8.2.8 Enumeration Type Documentation

(1) **enum LcdEvt\_LockState**

Touch panel event lock state

(a) **Enumerator:**

LCD_EVT_UNLOCK	Unlocked
LCD_EVT_LOCK	Locked

Definition at line 103 of file lcd\_ft5x06.h.

```

103     {
104         LCD_EVT_UNLOCK = 0,
105         LCD_EVT_LOCK
106     } LcdEvt_LockState ;

```

## 8.2.9 Function Documentation

(1) **void LCD\_Ft5x06\_ClearEvtMsg (const uint32\_t unEvtFLg)**

Clear assigned event flag.

(a) **Parameters:**

in	<i>unEvtFlg</i>	: event flag
----	-----------------	--------------

(b) **Return values:**

<i>None.</i>	
--------------	--

(2) **int\_t LCD\_Ft5x06\_Close (void )**

Closes the communication environment with the FT5x06.

(a) **Return values:**

<i>NONE</i>	
-------------	--

(3) **int\_t LCD\_Ft5x06\_EventEntry (const LcdEvt\_EntryType *eType*, const LcdCBFunc *function*)**

Registers in the event management structure a call-back function linked to an interrupt from the FT5x06. After registration finishes, the LCD interrupt is enabled and the event ID is sent as a return value.

(a) **Parameters:**

in	<i>eType</i>	Specified Interrupt type
in	<i>function</i>	Call-back function

(b) **Return values:**

<i>0</i>	to (LCDEVT_ENTRY_MAX - 1)
<i>-1</i>	event registration failure

(4) **int\_t LCD\_Ft5x06\_EventErase (const int\_t *nId*)**

Removes the registration information for the specified event ID from the event management structure.

(a) **Parameters:**

in	<i>nId</i>	Event ID return value of LCD_EventEntry function.
----	------------	--

(b) **Return values:**

<i>NONE</i>	
-------------	--

(5) **LCDEVT\_ENTRY\* LCD\_Ft5x06\_GetEventTable (const int\_t *nId*)**

Get assigned callback event.

(a) **Parameters:**

in	<i>nId</i>	event ID
----	------------	----------

(b) **Return values:**

<i>LCDEVT_ENTRY</i>	pointer to event.
---------------------	-------------------

- (6) **int\_t LCD\_Ft5x06\_Open (const uint32\_t *unIrqLv*, int16\_t *nTskPri*, uint32\_t *unTskStk*)**

Opens the communication environment with the FT5x06.

(a) **Parameters:**

in	<i>unIrqLv</i>	IRQ interrupt priority (0 to 255) Sets the GIC interrupt priority
in	<i>nTskPri</i>	Task Priority Sets the value of osPriority type.
in	<i>unTskStk</i>	Not Used.

(b) **Return values:**

0	Normal end
-1	failure to open

- (7) **uint8\_t LCD\_Ft5x06\_ReadCmd (const uint16\_t *unDevAddr*, uint8\_t \* *puData*, const uint32\_t *unSize*)**

Reads data from the FT5x06 via the RIIC DeviceController ch1.

(a) **Parameters:**

in	<i>unDevAddr</i>	LCD Device Address
in	<i>*puData</i>	Receive data buffer pointer
out	<i>unSize</i>	Receive Data Length

(b) **Return values:**

0	normal end
-1	data receive error

- (8) **int32\_t LCD\_Ft5x06\_SendEvtMsg (const uint32\_t *unEvtFlg*)**

Send event message to synchronism.

(a) **Parameters:**

in	<i>unEvtFlg</i>	event flag
----	-----------------	------------

(b) **Return values:**

0	Operation successful.
-1	Error occurred.



(9) **int\_t LCD\_Ft5x06\_StartInt (const LcdEvt\_EntryType eType)**

Removes masking of specified interrupt type.

(a) **Parameters:**

in	<i>eType</i>	Specified interrupt type
----	--------------	--------------------------

(b) **Return values:**

0	Always, normal end
---	--------------------

(10) **int32\_t LCD\_Ft5x06\_WaitEvtMsg (void )**

Wait event message to synchronism.

(a) **Return values:**

0	Event flag list.
-1	: Error occurred.

(11) **uint8\_t LCD\_Ft5x06\_WriteCmd (const uint16\_t unDevAddr, const uint8\_t uData, const uint32\_t unSize)**

Sends data to the FT5x06 via the RIIC DeviceController ch1.

(a) **Parameters:**

in	<i>unDevAddr</i>	LCD Device Address
in	<i>uData</i>	Send Data
in	<i>unSize</i>	Send Data Length

(b) **Return values:**

0	normal end
-1	data send processing error

## 8.2.10 Variable Documentation

(1) **int32\_t sLcdSemIdAcc**

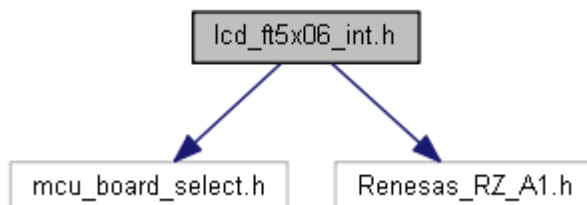
### 8.3 lcd\_ft5x06\_int.h File Reference

LCD Driver internal header for interrupt.

```
#include "mcu_board_select.h"
```

```
#include "Renesas_RZ_A1.h"
```

Include dependency graph for lcd\_ft5x06\_int.h:



#### 8.3.1 Macros

- `#define LCD_FT5x06_INT_NUM (IRQ3_IRQn)`

#### 8.3.2 Functions

- `int_t LCD_Ft5x06_Int_Open (const uint32_t unIrqLv)`  
*Open LCD interrupt.*
- `int_t LCD_Ft5x06_Int_Close (void)`  
*Close LCD interrupt.*
- `int_t LCD_Ft5x06_Int_Start (void)`  
*Enable interrupt of assigned type.*

#### 8.3.3 Detailed Description

LCD Driver internal header for interrupt.

Rev: 30 Date:: 2016-12-21 12:02:44 +0900#

#### 8.3.4 Macro Definition Documentation

- (1) `#define LCD_FT5x06_INT_NUM (IRQ3_IRQn)`

Definition at line 48 of file lcd\_ft5x06\_int.h.

#### 8.3.5 Function Documentation

- (1) `int_t LCD_Ft5x06_Int_Close (void )`

Close LCD interrupt.

- (a) **Return values:**

0	Operation Successful
-1	Error occurred

- (2) `int_t LCD_Ft5x06_Int_Open (const uint32_t unIrqLv)`

Open LCD interrupt.

(a) **Parameters:**

<i>unIrqLv</i>	IRQ interrupt level
----------------	---------------------

(b) **Return values:**

0	Operation Successful
-1	Error occurred

(3) **int\_t LCD\_Ft5x06\_Int\_Start (void )**

Enable interrupt of assigned type.

(a) **Return values:**

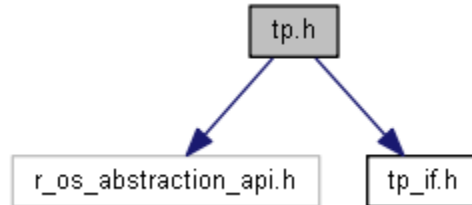
0	Operation Successful
---	----------------------

## 8.4 tp.h File Reference

TouchPanel Driver internal header.

```
#include "r_os_abstraction_api.h"
#include "tp_if.h"
```

Include dependency graph for tp.h:



### 8.4.1 Data Structures

- struct **TPEVT\_COORDINATES**
- struct **TPEVT\_ENTRY**

### 8.4.2 Macros

- `#define DBG_LEVEL_OT (-1)` /\* onetime debug \*/
- `#define DBG_LEVEL_DEF (0)` /\* default \*/
- `#define DBG_LEVEL_ERR (1)` /\* error \*/
- `#define DBG_LEVEL_MSG (2)` /\* message \*/
- `#define DBG_LEVEL_LOG (3)` /\* log \*/
- `#define DBG_LEVEL_DBG (4)` /\* debug \*/
- `#define DBG_LEVEL (DBG_LEVEL_ERR)`
- `#define DBG_printf_OT printf`
- `#define DBG_printf_DEF printf`
- `#define DBG_printf_ERR printf`
- `#define DBG_printf_MSG 1 ? (int32_t) 0 : printf`
- `#define DBG_printf_LOG 1 ? (int32_t) 0 : printf`
- `#define DBG_printf_DBG 1 ? (int32_t) 0 : printf`
- `#define SCOPE_STATIC static`
- `#define TPEVT_ENTRY_MAX (16)`
- `#define TP_EVTFLG_NONE (0x00000000)`
- `#define TP_EVTFLG_PENIRQ (0x00000001)` /\*! Touch Panel event flag, pen interrupt \*/
- `#define TP_EVTFLG_EXIT (0x00000080)` /\*! Touch Panel event flag, exit and delete task \*/
- `#define TP_EVTFLG_ALL (TP_EVTFLG_PENIRQ | TP_EVTFLG_EXIT)`

### 8.4.3 Enumerations

- enum **TpEvt\_LockState** { **TP\_EVT\_UNLOCK** = 0, **TP\_EVT\_LOCK** }

### 8.4.4 Functions

- void **TP\_Init** (void)  
*Initializes internal variables of the touch panel driver.*
  - *Securing of touch panel event entry area*
  - *Setting of internal variable `nEvtEntryId` to -1*
  - *Setting of internal variable `TpEvtLockInf` to **TP\_EVT\_UNLOCK***
- int\_t **TP\_Open** (const int\_t nWidth, const int\_t nHeight, const uint32\_t unIrqLv, const int16\_t nTskPri, const uint32\_t unTskStk)

*Opens the touch panel driver.*

- *Setting the LCD size in the driver's variables ScreenWidth and ScreenHeight*
- *Generation of touch panel task synchronization semaphore*
- *Generation of touch panel task*
- *Setting of task priority of touch panel task*
- *Opening of communication environment with LCD board*
- *Registration of call-back event when touch panel interrupt occurs in LCD event.*

• **int\_t TP\_Close** (void)

*Closes the touch panel driver.*

- *Removal of call-back event when touch panel interrupt occurs in LCD event*
- *Removal of all touch panel event registrations by the user*
- *Removal of touch panel task*
- *Removal of semaphore for synchronization with the touch panel task.*

• **int\_t TP\_EventEntry** (const **TpEvt\_EntryType** eMode, const int32\_t nPosX, const int32\_t nPosY, const int32\_t nWidth, const int32\_t nHeight, const **TpCBFunc** function)

*Registers in the event table a call-back function linked to a touch panel interrupt.*

*After registration finishes, the event ID is sent as a return value.*

.

• **int\_t TP\_EventErase** (const int\_t nId)

*Removes an event from the call-back event table of the touch panel driver.*

- *Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)*
- *Disabling of event associated with event ID (TPEVT\_ENTRY\_NON)*

• **int\_t TP\_ChangeEventEntry** (const int\_t nId, const int32\_t nPosX, const int32\_t nPosY, const int32\_t nWidth, const int32\_t nHeight)

*The rectangular area to which the event ID specified by the 1st argument (nId) is registered is changed to the rectangular area specified by the 2nd to 5th arguments.*

- *Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)*
- *Event ID checking (unregistered ID or removed ID)*
- *Registration of event in area of specified ID in touch panel event table.*

• **int\_t TP\_EventLockAll** (void)

*Locks all registered touch panel call-back events.*

*Calls the function described in TP\_EventLock, to set all events to the locked state (TP\_EVT\_LOCK).*

• **int\_t TP\_EventUnlockAll** (void)

*Unlocks all registered touch panel call-back events.*

*Calls the function described in TP\_EventUnlock, to set all events to the unlocked state (TP\_EVT\_UNLOCK).*

• **int\_t TP\_EventLock** (const int\_t nId)

*Locks the touch panel call-back event specified by the 1st argument (nId).*

- *Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)*
- *Setting the event specified by the event ID to the locked state (TP\_EVT\_LOCK) in the touch panel event table.*

• **int\_t TP\_EventUnlock** (const int\_t nId)

*Unlocks the touch panel call-back event specified by the 1st argument (nId).*

- *Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)*
- *Setting the event specified by the event ID to the unlocked state (TP\_EVT\_UNLOCK) in the touch panel event table.*

• **TPEVT\_ENTRY \* TP\_GetEventTable** (const int\_t nId)

*Acquires from the touch panel driver call-back event table the pointer address at which the event ID event information is registered.*

• **TpEvt\_LockState TP\_GetEventLockInf** (void)

*Acquires the lock state of the touch panel call-back event.*

- void **TP\_GetScreenSize** (int\_t \*pnWidth, int\_t \*pnHeight)  
*Acquires the screen size of the LCD panel.*
- int32\_t **TP\_SendEvtMsg** (const uint32\_t unEvtFlg)  
*Sends a synchronization event message.*
- int32\_t **TP\_WaitEvtMsg** (void)  
*Waits to receive a synchronization event message.*
- void **TP\_ClearEvtMsg** (const uint32\_t unEvtFlg)  
*Clears the specified event flag.*

#### 8.4.5 Variables

- os\_task\_t \* **p\_os\_task**

#### 8.4.6 Detailed Description

TouchPanel Driver internal header.

Rev: 30 Date:: 2016-12-21 12:02:44 +0900#

#### 8.4.7 Macro Definition Documentation

- (1) **#define DBG\_LEVEL (DBG\_LEVEL\_ERR)**

Definition at line 53 of file tp.h.

- (2) **#define DBG\_LEVEL\_DBG (4) /\* debug \*/**

Definition at line 51 of file tp.h.

- (3) **#define DBG\_LEVEL\_DEF (0) /\* default \*/**

Definition at line 47 of file tp.h.

- (4) **#define DBG\_LEVEL\_ERR (1) /\* error \*/**

Definition at line 48 of file tp.h.

- (5) **#define DBG\_LEVEL\_LOG (3) /\* log \*/**

Definition at line 50 of file tp.h.

- (6) **#define DBG\_LEVEL\_MSG (2) /\* message \*/**

Definition at line 49 of file tp.h.

- (7) **#define DBG\_LEVEL\_OT (-1) /\* onetime debug \*/**

Definition at line 46 of file tp.h.

- (8) **#define DBG\_printf\_DBG 1 ? (int32\_t) 0 : printf**

Definition at line 82 of file tp.h.

- (9) **#define DBG\_printf\_DEF printf**

Definition at line 60 of file tp.h.

(10) **#define DBG\_printf\_ERR printf**

Definition at line 65 of file tp.h.

(11) **#define DBG\_printf\_LOG 1 ? (int32\_t) 0 : printf**

Definition at line 77 of file tp.h.

(12) **#define DBG\_printf\_MSG 1 ? (int32\_t) 0 : printf**

Definition at line 72 of file tp.h.

(13) **#define DBG\_printf\_OT printf**

Definition at line 55 of file tp.h.

(14) **#define SCOPE\_STATIC static**

Definition at line 89 of file tp.h.

(15) **#define TP\_EVTFLG\_ALL (TP\_EVTFLG\_PENIRQ | TP\_EVTFLG\_EXIT)**

Definition at line 98 of file tp.h.

(16) **#define TP\_EVTFLG\_EXIT (0x00000080) /\*! Touch Panel event flag, exit and delete task \*/**

Definition at line 97 of file tp.h.

(17) **#define TP\_EVTFLG\_NONE (0x00000000)**

Definition at line 95 of file tp.h.

(18) **#define TP\_EVTFLG\_PENIRQ (0x00000001) /\*! Touch Panel event flag, pen interrupt \*/**

Definition at line 96 of file tp.h.

(19) **#define TPEVT\_ENTRY\_MAX (16)**

The max number of event entry

Definition at line 93 of file tp.h.

#### 8.4.8 Enumeration Type Documentation

(1) **enum TpEvt\_LockState**

Touch panel event lock state

(a) **Enumerator:**

TP_EVT_UNLOCK	Unlocked
TP_EVT_LOCK	Locked

--	--

Definition at line 105 of file tp.h.

```

105      {
106      TP_EVT_UNLOCK = 0,
107      TP_EVT_LOCK
108  } TpEvt_LockState ;

```

#### 8.4.9 Function Documentation

- (1) **int\_t TP\_ChangeEventEntry (const int\_t *nId*, const int32\_t *nPosX*, const int32\_t *nPosY*, const int32\_t *nWidth*, const int32\_t *nHeight*)**

The rectangular area to which the event ID specified by the 1st argument (*nId*) is registered is changed to the rectangular area specified by the 2nd to 5th arguments.

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
- Event ID checking (unregistered ID or removed ID)
- Registration of event in area of specified ID in touch panel event table.

(a) **Parameters:**

in	<i>nId</i>	event ID
in	<i>nPosX</i>	X-coordinate of LCD area
in	<i>nPosY</i>	Y-coordinate of LCD area
in	<i>nWidth</i>	width of LCD area
in	<i>nHeight</i>	height of LCD area

(b) **Return values:**

0	Operation successful.
-1	Error occurred.

- (2) **void TP\_ClearEvtMsg (const uint32\_t *unEvtFlg*)**

Clears the specified event flag.

(a) **Parameters:**

in	<i>unEvtFlg</i>	event flag
----	-----------------	------------

(b) **Return values:**

None.	
-------	--

- (3) **int\_t TP\_Close (void )**

Closes the touch panel driver.

- Removal of call-back event when touch panel interrupt occurs in LCD event
- Removal of all touch panel event registrations by the user
- Removal of touch panel task
- Removal of semaphore for synchronization with the touch panel task.



(a) **Return values:**

0	Operation successful.
-1	Error occurred.

- (4) **int\_t TP\_EventEntry (const TpEvt\_EntryType eMode, const int32\_t nPosX, const int32\_t nPosY, const int32\_t nWidth, const int32\_t nHeight, const TpCBFunc function)**

Registers in the event table a call-back function linked to a touch panel interrupt.

After registration finishes, the event ID is sent as a return value.

- Searching for a free area in the touch panel event table (Up to 16 touch panel events can be registered, and error processing occurs if no free area is available.)

• Making “specified touch action,” “X coordinate of specified area,” “Y coordinate of specified area,” “width of specified area,”

“height of specified area,” “specified call-back function” settings for the touch panel event table free area.

Note: When “X coordinate of specified area,” “Y coordinate of specified area,” “width of specified area,” and “height of specified area” are registered in the touch panel event table, the following processing is performed to register the result as a rectangular area:

st.x (X coordinate of area start position) <- “X coordinate of specified area”

st.y (Y coordinate of area start position) <- “Y coordinate of specified area”

ed.x (X coordinate of area end position) <- (“X coordinate of specified area” - “width of specified area”)

ed.y (Y coordinate of area end position) <- (“Y coordinate of specified area” - “height of specified area”)

(a) **Parameters:**

in	<i>eMode</i>	event type
in	<i>nPosX</i>	X-coordinate of LCD area
in	<i>nPosY</i>	Y-coordinate of LCD area
in	<i>nWidth</i>	width of LCD area
in	<i>nHeight</i>	height of LCD area
in	<i>function</i>	callback function

(b) **Return values:**

0	to (TPEVT_ENTRY_MAX-1)
-1	Error occurred.

- (5) **int\_t TP\_EventErase (const int\_t nId)**

Removes an event from the call-back event table of the touch panel driver.

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
- Disabling of event associated with event ID (TPEVT\_ENTRY\_NON)

(a) **Parameters:**

in	<i>nId</i>	event ID
----	------------	----------

(b) **Return values:**

0	Operation successful.
-1	Error occurred.

(6) **int\_t TP\_EventLock (const int\_t nId)**

Locks the touch panel call-back event specified by the 1st argument (nId).

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
- Setting the event specified by the event ID to the locked state (TP\_EVT\_LOCK) in the touch panel event table.

(a) **Parameters:**

in	nId	event ID
----	-----	----------

(b) **Return values:**

0	Operation successful.
-1	Error occurred.

(7) **int\_t TP\_EventLockAll (void )**

Locks all registered touch panel call-back events.

Calls the function described in TP\_EventLock, to set all events to the locked state (TP\_EVT\_LOCK).

(a) **Return values:**

0	Operation successful.
-1	Error occurred.

(8) **int\_t TP\_EventUnlock (const int\_t nId)**

Unlocks the touch panel call-back event specified by the 1st argument (nId).

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
- Setting the event specified by the event ID to the unlocked state (TP\_EVT\_UNLOCK) in the touch panel event table.

(a) **Parameters:**

in	nId	event ID
----	-----	----------

(b) **Return values:**

0	Operation successful.
-1	Error occurred.

(9) **int\_t TP\_EventUnlockAll (void )**

Unlocks all registered touch panel call-back events.

Calls the function described in TP\_EventUnlock, to set all events to the unlocked state (TP\_EVT\_UNLOCK).

(a) **Return values:**

0	Operation successful.
-1	Error occurred.

(10) **TpEvt\_LockState TP\_GetEventLockInf (void )**

Acquires the lock state of the touch panel call-back event.

(a) **Return values:**

TP_EVT_LOCK	In locked state
TP_EVT_UNLOCK	In unlocked state.

(11) **TPEVT\_ENTRY\* TP\_GetEventTable (const int\_t nId)**

Acquires from the touch panel driver call-back event table the pointer address at which the event ID event information is registered.

(a) **Parameters:**

in	nId	event ID
----	-----	----------

(b) **Return values:**

TPEVT_ENTRY	pointer to event
-------------	------------------

(12) **void TP\_GetScreenSize (int\_t \* pnWidth, int\_t \* pnHeight)**

Acquires the screen size of the LCD panel.

(a) **Parameters:**

out	*pnWidth	pointer to width value
out	*pnHeight	pointer to height value

(b) **Return values:**

None	
------	--

(13) **void TP\_Init (void )**

Initializes internal variables of the touch panel driver.

- Securing of touch panel event entry area
- Setting of internal variable nEvtEntryId to -1
- Setting of internal variable TpEvtLockInf to TP\_EVT\_UNLOCK

(a) **Return values:**

None.	
-------	--

(14) **int\_t TP\_Open (const int\_t *nWidth*, const int\_t *nHeight*, const uint32\_t *unIrqLv*, const int16\_t *nTskPri*, const uint32\_t *unTskStk*)**

Opens the touch panel driver.

- Setting the LCD size in the driver's variables ScreenWidth and ScreenHeight
- Generation of touch panel task synchronization semaphore
- Generation of touch panel task
- Setting of task priority of touch panel task
- Opening of communication environment with LCD board
- Registration of call-back event when touch panel interrupt occurs in LCD event.

(a) **Parameters:**

in	<i>nWidth</i>	screen width
in	<i>nHeight</i>	screen height
in	<i>unIrqLv</i>	IRQ interrupt level
in	<i>nTskPri</i>	task priority
in	<i>unTskStk</i>	task stack size

(b) **Return values:**

0	Operation successful.
-1	Error occurred.

(15) **int32\_t TP\_SendEvtMsg (const uint32\_t *unEvtFlg*)**

Sends a synchronization event message.

(a) **Parameters:**

in	<i>unEvtFlg</i>	event flag
----	-----------------	------------

(b) **Return values:**

0	Operation successful.
-1	Error occurred.

(16) **int32\_t TP\_WaitEvtMsg (void )**

Waits to receive a synchronization event message.

(a) **Return values:**

<i>TP_EVTFLG_NO NE</i>	No event flags
<i>TP_EVTFLG_PEN IRQ</i>	Interrupt pending
<i>TP_EVTFLG_EXI T</i>	End task
<i>TP_EVTFLG_ALL</i>	Both Interrupt pending and exit flag.

-I

Error occurred.

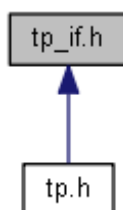
#### 8.4.10 Variable Documentation

(1) `os_task_t* p_os_task`

### 8.5 tp\_if.h File Reference

TouchPanel Driver API header.

This graph shows which files directly or indirectly include this file:



#### 8.5.1 Data Structures

- struct **TP\_TouchFinger\_st**
- struct **TP\_TouchEvent\_st**

#### 8.5.2 Macros

- `#define TP_TOUCHNUM_MAX` (2)

#### 8.5.3 Typedefs

- typedef void(\* **TpCBFunc**) (int\_t, **TP\_TouchEvent\_st** \*)

#### 8.5.4 Enumerations

- enum **TpEvt\_EntryType** { **TPEVT\_ENTRY\_NONE** = 0x0000, **TPEVT\_ENTRY\_UP** = 0x0001, **TPEVT\_ENTRY\_DOWN** = 0x0002, **TPEVT\_ENTRY\_MOVE** = 0x0004, **TPEVT\_ENTRY\_ALL** = 0x0007, **TPEVT\_ENTRY\_UNKNOWN** = 0x8000 }

#### 8.5.5 Functions

- void **TouchPanel\_Init** (void)  
*Initializes the touch panel driver by calling the TP\_Init.*
- int\_t **TouchPanel\_Open** (const int\_t nWidth, const int\_t nHeight, const uint32\_t nIrqLv, const int16\_t nTskPri, const uint32\_t nTskStk)  
*Generates and initializes a touch panel task by calling the TP\_Open.  
Do not call this function during touch panel utility has been opened.*
- int\_t **TouchPanel\_Close** (void)  
*Touch Panel utility close function.*
- int\_t **TouchPanel\_EventEntry** (const **TpEvt\_EntryType** eMode, const int32\_t nPosX, const int32\_t nPosY, const int32\_t nWidth, const int32\_t nHeight, const **TpCBFunc** function)  
*Registers a call-back function linked to the LCD area where a touch panel event occurs in the touch panel event management structure.  
Calls the function described in TP\_EventEntry, to perform the following processing:*
  - Searching for a free area in the touch panel event table (Up to 16 touch panel events can be registered, and error processing occurs if no free area is available.)
  - Making “specified touch action,” “X coordinate of specified area,” “Y coordinate of specified area,”

“width of specified area,” “height of specified area,” “specified call-back function”

settings for the touch panel event table free area.

*Note: If events occur simultaneously in multiple registered areas that overlap, the associated call-back functions are executed in order, starting with the one with the lowest event ID.*

- **int\_t TouchPanel\_EventErase** (const int\_t nId)  
Removes registration information for the specified event ID from the touch panel event management structure.  
Calls the function described in *TP\_EventErase*, to perform the following processing:
  - Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
 Disabling of event associated with event ID.
- **int\_t TouchPanel\_ChangeEventEntry** (const int\_t nId, const int32\_t nPosX, const int32\_t nPosY, const int32\_t nWidth, const int32\_t nHeight)  
Changes the LCD area of the specified event ID.  
Calls the function described in *TP\_ChangeEventEntry*, to perform the following processing:
  - Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
  - Event ID checking (unregistered ID or removed ID)
 Registration of event in area of specified ID in touch panel event table.
- **int\_t TouchPanel\_EventLockAll** (void)  
Locks processing of all touch panel events.  
Calls the function described in *TP\_EventLockAll*, to perform the following processing:  
Setting all events in the touch panel event table to the locked state  
.
- **int\_t TouchPanel\_EventUnlockAll** (void)  
Unlocks processing of all touch panel events.  
Calls the function described in *TP\_EventUnlockAll*, to perform the following processing:  
Setting all events in the touch panel event table to the unlocked state.
- **int\_t TouchPanel\_EventLock** (const int\_t nId)  
Locks processing of the touch panel event specified by the event ID.  
Calls the function described in *TP\_EventLock*, to perform the following processing:
  - Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
 Setting the event specified by the event ID to the locked state in the touch panel event table.
- **int\_t TouchPanel\_EventUnlock** (const int\_t nId)  
Unlocks processing of the touch panel event specified by the event ID.  
Calls the function described in *TP\_EventUnlock*, to perform the following processing:
  - Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
 Setting the event specified by the event ID to the unlocked state in the touch panel event table.

### 8.5.6 Detailed Description

TouchPanel Driver API header.

Rev: 30 Date:: 2016-12-21 12:02:44 +0900

### 8.5.7 Macro Definition Documentation

(1) **#define TP\_TOUCHNUM\_MAX** (2)

Definition at line 44 of file *tp\_if.h*.

### 8.5.8 Typedef Documentation

- (1) **typedef void(\* TpCBFunc) (int\_t, TP\_TouchEvent\_st \*)**

Definition at line 75 of file tp\_if.h.

### 8.5.9 Enumeration Type Documentation

- (1) **enum TpEvt\_EntryType**

The type of touch panel event entry

- (a) **Enumerator:**

TPEVT_ENTRY_NONE	None
TPEVT_ENTRY_UP	Up
TPEVT_ENTRY_DOWN	Down
TPEVT_ENTRY_MOVE	Move
TPEVT_ENTRY_ALL	All
TPEVT_ENTRY_UNKNOWN	internal event state

Definition at line 50 of file tp\_if.h.

```

50      {
51      TPEVT_ENTRY_NONE    = 0x0000,
52      TPEVT_ENTRY_UP     = 0x0001,
53      TPEVT_ENTRY_DOWN   = 0x0002,
54      TPEVT_ENTRY_MOVE   = 0x0004,
56      TPEVT_ENTRY_ALL    = 0x0007,
58      TPEVT_ENTRY_UNKNOWN = 0x8000
59 } TpEvt_EntryType ;

```

### 8.5.10 Function Documentation

- (1) **int\_t TouchPanel\_ChangeEventEntry (const int\_t nId, const int32\_t nPosX, const int32\_t nPosY, const int32\_t nWidth, const int32\_t nHeight)**

Changes the LCD area of the specified event ID.

Calls the function described in TP\_ChangeEventEntry, to perform the following processing:

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)
- Event ID checking (unregistered ID or removed ID)

Registration of event in area of specified ID in touch panel event table.

(a) **Parameters:**

in	<i>nId</i>	Event ID
in	<i>nPosX</i>	X coordinate of area after change
in	<i>nPosY</i>	Y coordinate of area after change
in	<i>nWidth</i>	Width of area after change
in	<i>nHeight</i>	Height of area after change

(b) **Return values:**

0	normal end
-1	LCD area change failure

(2) **int\_t TouchPanel\_Close (void )**

Touch Panel utility close function.

(a) **Return values:**

NONE	
------	--

(3) **int\_t TouchPanel\_EventEntry (const TpEvt\_EntryType eMode, const int32\_t nPosX, const int32\_t nPosY, const int32\_t nWidth, const int32\_t nHeight, const TpCBFunc function)**

Registers a call-back function linked to the LCD area where a touch panel event occurs in the touch panel event management structure.

Calls the function described in TP\_EventEntry, to perform the following processing:

- Searching for a free area in the touch panel event table (Up to 16 touch panel events can be registered, and error processing occurs if no free area is available.)
- Making “specified touch action,” “X coordinate of specified area,” “Y coordinate of specified area,” “width of specified area,” “height of specified area,” “specified call-back function” settings for the touch panel event table free area.

Note: If events occur simultaneously in multiple registered areas that overlap, the associated call-back functions are executed in order, starting with the one with the lowest event ID.

(a) **Parameters:**

in	<i>eMode</i>	Specified touch action
in	<i>nPosX</i>	X coordinate of specified area
in	<i>nPosY</i>	Y coordinate of specified area
in	<i>nWidth</i>	width of specified area
in	<i>nHeight</i>	height of specified area
in	<i>function</i>	Specified call-back function

(b) **Return values:**

Success	event ID of 0 to (TPEVT_ENTRY_MAX -1) if successful
---------	---



<i>Fail</i>	returns -1
-------------	------------

(4) **int\_t TouchPanel\_EventErase (const int\_t nId)**

Removes registration information for the specified event ID from the touch panel event management structure.

Calls the function described in TP\_EventErase, to perform the following processing:

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)

Disabling of event associated with event ID.

(a) **Parameters:**

in	<i>nId</i>	Event ID
----	------------	----------

(b) **Return values:**

0	normal end
-1	event removal failure

(5) **int\_t TouchPanel\_EventLock (const int\_t nId)**

Locks processing of the touch panel event specified by the event ID.

Calls the function described in TP\_EventLock, to perform the following processing:

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)

Setting the event specified by the event ID to the locked state in the touch panel event table.

(a) **Parameters:**

in	<i>nId</i>	Event ID
----	------------	----------

(b) **Return values:**

0	normal end
-1	event removal failure

(6) **int\_t TouchPanel\_EventLockAll (void )**

Locks processing of all touch panel events.

Calls the function described in TP\_EventLockAll, to perform the following processing:

Setting all events in the touch panel event table to the locked state

(a) **Return values:**

0	normal end
-1	touch panel event locking failure

(7) **int\_t TouchPanel\_EventUnlock (const int\_t nId)**

Unlocks processing of the touch panel event specified by the event ID.

Calls the function described in TP\_EventUnlock, to perform the following processing:

- Event ID checking (within range of 0 to TPEVT\_ENTRY\_MAX)

Setting the event specified by the event ID to the unlocked state in the touch panel event table.

(a) **Parameters:**

in	<i>nId</i>	Event ID
----	------------	----------

(b) **Return values:**

0	normal end
-1	event removal failure

(8) **int\_t TouchPanel\_EventUnlockAll (void )**

Unlocks processing of all touch panel events.

Calls the function described in TP\_EventUnlockAll, to perform the following processing:

Setting all events in the touch panel event table to the unlocked state.

(a) **Return values:**

0	normal end
-1	touch panel event unlocking failure

(9) **void TouchPanel\_Init (void )**

Initializes the touch panel driver by calling the TP\_Init.

(a) **Return values:**

NONE	
------	--

(10) **int\_t TouchPanel\_Open (const int\_t nWidth, const int\_t nHeight, const uint32\_t unIrqLv, const int16\_t nTskPri, const uint32\_t unTskStk)**

Generates and initializes a touch panel task by calling the TP\_Open.

Do not call this function during touch panel utility has been opened.

(a) **Parameters:**

in	<i>nWidth</i>	LCD width
in	<i>nHeight</i>	LCD height
in	<i>unIrqLv</i>	IRQ interrupt priority (0 to 255), sets the GIC interrupt priority
in	<i>nTskPri</i>	Task priority, sets the values of the osPriority type
in	<i>unTskStk</i>	unTskStk, not used.

(b) **Return values:**

NONE	
------	--

## Website and Support

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/contact/>

All trademarks and registered trademarks are the property of their respective owners.

**Revision History**

Rev.	Date			
		Page	Description	Remark
1.00	Jun 29, 2018	-	First Edition issued	-
2.00	Jun 29, 2018	1	Introduction Corrected the wording.	-
		4	1. Specifications Corrected the wording.	-

## 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. 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 Microprocessing unit or Microcontroller unit products 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.

## 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.
- (Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
- (Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)



### 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.**

1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.  
Tel: +1-408-432-8888, Fax: +1-408-434-5351

#### **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: +44-1628-651-700, Fax: +44-1628-651-804

#### **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, 200333 P. R. China  
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, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India  
Tel: +91-80-67208700, Fax: +91-80-67208777

#### **Renesas Electronics Korea Co., Ltd.**

17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea  
Tel: +82-2-558-3737, Fax: +82-2-558-5338