RX Series

r_switches

FIT Sample Code
Rev.1.50
February 15, 2013

Overview

This module allows you to easily have callback functions for when a switch is pressed on your board. Detection can be done by polling or by using the IRQ functionality of the switch pins. If polling is used then the user can use the debounce mechanisms built into the code.

Supported Boards

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

- **RSKRX610**
- **RSK+RX62N**
- **RSKRX62T**
- **RDKRX62N**
- **RSKRX630**
- **RSKRX63N**
- **RDKRX63N**
- **RSKRX210**
- **RSKRX111**

Contents

1. API Information

This Middleware API follows the Renesas API naming standards.

1.1 Hardware Requirements

This middleware requires your MCU support the following features if IRQ detection is enabled:

- Trigger an interrupt based on voltage on pin or change of voltage on pin.

1.2 Hardware Resource Requirements

None.

1.3 Software Requirements

This middleware is dependent upon the following packages:

- Renesas Board Support Package (r_bsp) v2.10 or higher

1.4 Supported Toolchains

This middleware is tested and working with the following toolchains:

- Renesas RX Toolchain v1.02

1.5 Header Files

All API calls are accessed by including a single file *r_switches_if.h* which is supplied with this middleware's project code.

1.6 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*.

1.7 Configuration Overview

All configuration is done through the header file *r_switches_config.h*.

Configuration Options in *r_switches_config.h*	
SWITCHES_CFG_DETECTION_MODE	This macro sets whether interrupts (IRQ) or polling is used for detecting switch presses. The benefit of using interrupts is that no extra processing is used for polling and the use of a system timer tick is not a requirement. The downside of using interrupts is that callback functions are called from within an interrupt so if your ISR is long then it can degrade the real-time response of your system. The benefit of polling is that functions can be called at the application level and debouncing is supported. The downside to polling is that your system must call the R_SWITCHES_Update() on a regular basis which requires extra processing. 0 = Use interrupts 1 = Use polling

SW1_CALLBACK_FUNCTION **SW2_CALLBACK_FUNCTION** … **SWn_CALLBACK_FUNCTION**	The definition for these macros should be the name of the function that you want called when a switch is pressed. It is very important that the user recognize that this function will be called from the interrupt service routine. This means that code inside of the function should be kept short to ensure it does not hold up the rest of the system. This function's prototype should look like: void sw1_callback(void * pdata); When the callback is called, a pointer to a sw_callback_arg_t structure will be passed as the argument and cast as (void *). Please see r_switches_if.h for more information on the sw_callback_arg_t structure. Please note that this structure will only be valid during the callback function. If you need the contents of this structure after the callback then store this data in your application. Example: If SWITCHES_CFG_SW1_CALLBACK_FUNCTION is defined to be sw1_callback then the sw1_callback function will be called when switch 1 is pressed. If you do not wish to have a callback for a switch then comment out its macro below.

Table 1 : Configuration Options

1.8 API Data Structures

This section details the data structures that are used with the middleware's API functions.

1.8.1 Callback Function Argument

This structure is what is passed to the user's callback function when a switch press has occurred.

```
/* Data Structure #1 */
typedef struct
{
    uint32_t switch_number;     //Which switch was pressed
} sw_callback_arg_t;
```

1.9 Return Values

None.

1.10 Adding Middleware to Your Project

Follow the steps below to add the middleware's code to your project.

1. Copy the 'r_switches' directory to your project directory.
2. Add src\r_switches.c to your project.
3. Add an include path to the 'r_switches' directory.
4. Add an include path to the 'r_switches\src' directory.
5. Copy r_switches_config_reference.h from 'ref' directory to your desired location and rename to r_switches_config.h.
6. Configure middleware through r_switches_config.h.
7. Add a #include for r_switches_if.h to files that need to use this package.

2. API Functions

2.1 Summary

The following functions are included in this API:

Function	**Description**
R_SWITCHES_Init()	Initialize r_switches data structures and pins.
R_SWITCHES_GetVersion()	Returns the version of this module.
R_SWITCHES_Update()	If using polling detection, then this function updates the debounce data and calls the user callback functions when needed.

2.2 R_SWITCHES_Init

Initializes port pins and sets up debounce data (when polling is enabled).

Format

```
void R_SWITCHES_Init(uint32_t detection_hz, uint32_t debounce_counts);
```

Parameters

detection_hz

The times per second that the user will call R_SWITCHES_Update(). This argument is deprecated and is no longer used.

debounce_counts

The number of times to check the port value before accepting the change. The slower the rate at which R_SWITCHES_Update() will likely lower this number.

Return Values

None.

Properties

Prototyped in file “r_switches_if.h”

Description

Initializes everything needed to start up switch detection. If interrupt detection is used then none of the input arguments are used. If polling is used then *debouce_counts* will set how many times switch must be read as active before calling the user callback function. For example, if *debounce_counts* is set to 5 then the user callback function will be called after R_SWITCHES_Update() has been called 5 times if the switch is read as active each time. R_SWITCHES_Update() will update the counters associated with each switch. If the switch is read as active then that switches counter is incremented until it is equal to *debounce_counts* at which point the switch is marked as pressed and the user callback function is called. If R_SWITCHES_Update() is called and the switch is not active then its counter is decremented until it reaches 0.

Reentrant

Yes.

Example

```
/* Polling detection being used.
   R_SWITCHES_Update() will be called at 100Hz or every 10ms. Since we are
   sending in 5 debounce counts this means that switch will have to be read
   as active at least 5 times (50ms) in order for the switch to be marked as
   pressed and to have the user callback function called. */
R_SWITCHES_Init(100, 5);
```

2.3 R_SWITCHES_GetVersion

Returns the version of the code.

Format

```
uint32_t R_SWITCHES_GetVersion(void);
```

Parameters

None.

Return Values

Four byte value that represents the version of the code.

Properties

Prototyped in file "r_switches_if.h"

Description

Returns an encoded version of the middleware. The version number is encoded where the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number. For example, Version 4.25 would be returned as 0x00040019.

Reentrant

Yes.

Example

```
/* Get version of r_switches code and make sure it's greater than v2.08 */
uint32_t current_version;

current_version = R_SWITCHES_GetVersion();

if (current_version <= 0x00020008)
{
    printf("Need at least v2.08 of r_switches code to run this application!");
}
```

Special Notes:

This function is specified to be an inline function in *r_switches.c.*

2.4 R_SWITCHES_Update

Reads switches and updates debounce data (when polling detection mode is enabled).

Format

```
    void R_SWITCHES_Update(void);
```

Parameters

None.

Return Values

None.

Properties

Prototyped in file “r_switches_if.h”

Description

Reads each switch to determine if it is currently active or not. Using this information the function updates the debounce data for each switch. If the switch has been active for the number of debounce counts as sent in to the R_SWITCHES_Init() function then the user callback function will be called. The user callback function will only be called once per button press event. After a switch is deemed as pressed, it must then go through the debounce process to return to inactive before it can return to the active state again and cause another callback. For example, if the user input 5 as the *debounce_counts parameter to the R_SWITCHES_Init()* function then the user callback will not be called until the switch has been read as active at least 5 times. After the switch has been deemed as pressed, the switch will have to be read as inactive at least 5 times before it will be deemed as released. When the switch is marked as released it can start the process of being marked as pressed again.

Reentrant

Yes.

Example

```
/* Read switches.
   This function should be called on a regular basis (e.g. from system timer
   tick). */
R_SWITCHES_Update();
```

Special Notes:

If interrupt detection is enabled then this function is still defined but it will simply return when called.

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	Mar.13.12	—	First edition issued
1.50	Feb.15.13	—	• Updated API Information section. • Added API Summary subsection • Add R_SWITCHES_Update() function to API list • Updated R_SWITCHES_Init() API description • Added Callback Function Argument subsection

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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord 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 one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

RENESAS

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0