RZ/A2M Group

RZ/A2M INTC Driver

R01AN4500EG0100
Rev.1.0
Sept 19, 2018

Introduction

This application note describes the operation of the software INTC Driver for the RZ/A2 device on the RZ/A2M CPU Board.

It provides a comprehensive overview of the driver. For further details please refer to the software driver itself.

The user is assumed to have knowledge of e^2 studio and to be equipped with an RZ/A2M CPU Board.

Target Device

RZ/A2M Group

Driver Dependencies

This driver has no other driver dependencies.

Referenced Documents

Document Type	Document Name	Document No.
User's Manual	RZ/A2M Hardware Manual	R01UH0746EJ

List of Abbreviations and Acronyms

Abbreviation	Full Form
API	Application Programming Interface
ARM	Advanced RISC Machines
CPU	Central Processing Unit
HLD	High Layer Driver
IDE	Integrated Development Environment
INTC	Interrupt Controller
LLD	Low Layer Driver

Table 1-1 List of Abbreviations and Acronyms

Contents

1. Outline of Software Driver

The INTC (Interrupt Controller) driver is an abstraction layer between the application and the hardware. It provides functions for enabling and disabling interrupts, setting callback functions, interrupt priorities, etc.

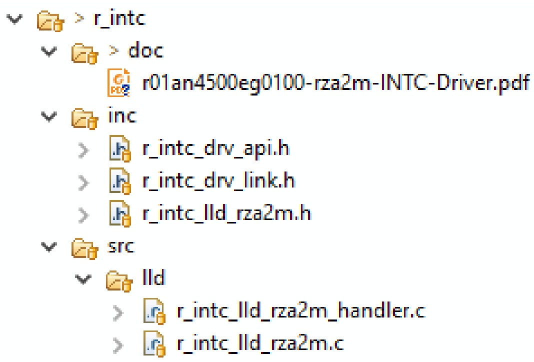
2. Description of the Software Driver

The key features of the driver include:

- Enabling and disabling interrupts
- Setting interrupt detection methods (such as level or edge triggered)
- Setting interrupt priorities
- Setting interrupt callback functions
- Masking interrupts on or off

2.1 Structure

Unlike many of the other drivers, the INTC driver currently consists of a single layer: the Low Layer Driver (LLD). This includes all the hardware specific functions and provides the API to the application.



2.2 Description of each file

Each file's description can be seen in the following table.

Filename	Usage	Description
Application-Facing Driver API		
r_intc_drv_api.h	Application	** FOR FUTURE USE **
Low Layer API		
r_intc_lld_xxxx.h	API header file	Low Layer Driver (LLD) header file (where "xxxx" is a device and board-specific identification). This is the header file to include in application code.
Abstraction Link between High and Low Layer Drivers (HLD/LLD Link)		
r_intc_drv_link.h	Private (HLD/LLD only)	** FOR FUTURE USE **
Low Layer Driver (LLD) Source		
r_intc_lld_xxxx.c	Private (LLD only)	(Where "xxxx" is a device and board specific identification). Provides the definitions for the Low Layer Driver interface.
r_intc_lld_xxxx_handler.c	Private (LLD only)	Contains Low Layer Driver interrupt handler routines

2.3 Low Layer Driver

The Low Layer Driver provides the functions to configure the hardware.

Return Type	Function	Description	Arguments	Return
e_r_drv_intc_err_t	**R_INTC_Init**(void)	Initialise the INTC driver	None	**INTC_SUCCESS**
e_r_drv_intc_err_t	**R_INTC_SetNMIConfig(**const st_r_drv_nmi_cfg_t * p_nmi_config)	Set NMI configuration	**p_nmi_config:** [in] NMI configuration (rising or falling edge)	**INTC_SUCCESS**
e_r_drv_intc_err_t	**R_INTC_GetNMIConfig(**st_r_drv_nmi_cfg_t * p_nmi_config)	Set NMI configuration	**p_nmi_config:** [out] NMI configuration (rising or falling edge)	**INTC_SUCCESS**
e_r_drv_intc_err_t	**R_IRQ_Init(**const st_r_drv_irq_cfg_t * p_irq_config**)**	Initialise IRQs (IRQ0 – IRQ7) setting interrupt detection methods	**p_irq_config:** [in] Interrupt detection methods	**INTC_SUCCESS**
e_r_drv_intc_err_t	**R_TINT_Init(**const st_r_drv_tint_cfg_t * p_tint_config)	Initialise TINTs (TINT0 – TINT31) setting interrupt detection methods	**p_tint_config:** [in] Interrupt detection methods	**INTC_SUCCESS**
e_r_drv_intc_err_t	**R_INTC_RegistIntFunc(**e_r_drv_intc_intid_t int_id, void (* func) (uint32_t int_sense))	Register interrupt handler function	**int_id:** [in] interrupt ID (0 – 511) **int_sense:** [in] interrupt handler function	**INTC_SUCCESS or INTC_ERR_INVALID**
e_r_drv_intc_err_t	**R_INTC_GetIntFunc(**e_r_drv_intc_intid_t int_id, void (* func) (uint32_t int_sense)**)**	Get the address of the interrupt handler function	**int_id:** [in] interrupt ID (0 – 511) **int_sense:** [out] interrupt handler function	**INTC_SUCCESS or INTC_ERR_INVALID**
e_r_drv_intc_err_t	**R_INTC_Enable(**e_r_drv_intc_intid_t int_id**)**	Enable an interrupt	**int_id:** [in] interrupt ID (0 – 511)	**INTC_SUCCESS or INTC_ERR_INVALID**
e_r_drv_intc_err_t	**R_INTC_Disable(**e_r_drv_intc_intid_t int_id**)**	Disable an interrupt	**int_id:** [in] interrupt ID (0 – 511)	**INTC_SUCCESS or INTC_ERR_INVALID**
e_r_drv_intc_err_t	**R_INTC_SetPriority(**e_r_drv_intc_intid_t int_id, e_r_drv_intc_priority_t priority**)**	Set an interrupt priority	**int_id:** [in] interrupt ID (0 – 511) **priority:** [in] interrupt priority	**INTC_SUCCESS, INTC_ERR_INVALID ID or INTC_ERR_INVALID ID_PRIORITY**
e_r_drv_intc_err_t	**R_INTC_GetPriority(**e_r_drv_intc_intid_t int_id, e_r_drv_intc_priority_t *priority**)**	Get an interrupt priority	**int_id:** [in] interrupt ID (0 – 511) **priority:** [out] interrupt priority	**INTC_SUCCESS, INTC_ERR_INVALID ID, INTC_ERR_INVALID ID_PRIORITY or INTC_ERR_INVALID ID**

e_r_drv_intc_err_t	**R_INTC_SetMaskLevel(**e_r_drv_intc_priority_t mask_level**)**	Sets the interrupt mask level	**mask_level:** [in] Interrupt mask level (0 - 31)	**INTC_SUCCESS or INTC_ERR_INVALID_PRIORITY**
e_r_drv_intc_err_t	**R_INTC_GetMaskLevel(**e_r_drv_intc_priority_t *mask_level**)**	Gets the interrupt mask level	**mask_level:** [out] Interrupt mask level (0 - 31)	**INTC_SUCCESS**
e_r_drv_intc_err_t	**R_IRQ_SetSense(**e_r_drv_irq_num_t irq_num, e_r_drv_irq_sense_t sense**)**	Set IRQ pin interrupt detection method	**num:** [in] IRQ pin number **sense:** [in] interrupt detection method	**INTC_SUCCESS, INTC_ERR_INVALID_NUM or INTC_ERR_INVALID_SENSE**
e_r_drv_intc_err_t	**R_IRQ_GetSense(**e_r_drv_irq_num_t irq_num, e_r_drv_irq_sense_t *sense**)**	Get IRQ pin interrupt detection method	**num:** [in] IRQ pin number **sense:** [out] interrupt detection method	**INTC_SUCCESS or INTC_ERR_INVALID_NUM**
e_r_drv_intc_err_t	**R_INTC_GetPendingStatus(**e_r_drv_intc_intid_t int_id, e_r_drv_intc_pending_t *pending_status**)**	Get interrupt pending state	**int_id:** [in] interrupt ID (0 – 511) **pending_status:** [out] the pending state	**INTC_SUCCESS or INTC_ERR_INVALID_ID**
e_r_drv_intc_err_t	**R_TINT_SetSense(**e_r_drv_tint_num_t tint_num, e_r_drv_tint_sense_t sense**)**	Set the pin interrupt detection method	**tint_num:** [in] TINT pin number **sense:** [in] TINT interrupt detection method	**INTC_SUCCESS, INTC_ERR_INVALID_NUM or INTC_ERR_INVALID_SENSE**
e_r_drv_intc_err_t	**R_INTC_SetIrqMask(**e_r_drv_irq_mask_t mask**)**	Set IRQ mask	**mask:** [in] interrupt mask (on or off)	**INTC_SUCCESS or INTC_ERR_INVALID**
e_r_drv_intc_err_t	**R_INTC_GetIrqMask(**e_r_drv_irq_mask_t *mask**)**	Get IRQ mask	**mask:** [in] interrupt mask (on or off)	**INTC_SUCCESS**
uint32_t	**R_INTC_GetVersion(**st_drv_info_t *pinfo**)**	Get Low Layer Driver version information	**pinfo**: [out] pointer to version information structure	**DRV_SUCCESS**

3. Example of Use

This section gives simple examples for initialising the driver, registering an interrupt callback function, setting interrupt priority, enabling an interrupt, disabling an interrupt, and finally getting the driver version.

3.1 Initialise Driver

```
e_r_drv_intc_err_t result;

result = R_INTC_Init();
```

3.2 Registering an Interrupt Callback Function

```
void MyInterruptHandler(uint32_t int_sense)
{
    /* interrupt handling code */
}

result = R_INTC_RegistIntFunc(INTC_ID_DMAC30_DMAERR0, MyInterruptHandler);
```

3.3 Setting Interrupt Priority

```
result = R_INTC_SetPriority(INTC_ID_DMAC30_DMAERR0, 28);
```

3.4 Enabling an Interrupt

```
result = R_INTC_Enable(INTC_ID_DMAC31_DMAINT9);
```

3.5 Disabling an Interrupt

```
result = R_INTC_Disable(INTC_ID_DMAC31_DMAINT9);
```

3.6 Get Version

```
st_ver_info_t info;
uint32_t get_version_result;

get_version_result = R_INTC_GetVersion(&Info);
```

4. OS Support

This driver supports any OS.

5. How to Import the Driver

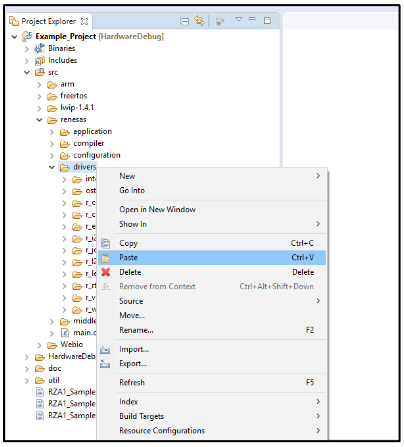
This section describes how to import the driver into your project. Generally, there are two steps in any IDE:

1) Copy the software driver to the location in the source tree that you require for your project.
2) Add the include path of the driver to the compiler.

5.1 e^2 studio

To import the driver into your project please follow the instructions below.

1) In Windows Explorer, right-click on the r_intc folder, and click **Copy**.
2) In e^2 studio Project Explorer view, select the folder where you wish the driver project to be located; right-click and click **Paste**.
3) Right-click on the parent project folder (in this case 'Example_Project') and click **Properties ...**
4) In 'C/C++ Build → Settings → Cross ARM Compiler → Includes', add the include folder of the newly added driver, e.g. '${ProjDirPath}\src\renesas\drivers\r_intc\inc'



Website and Support

Renesas Electronics website
https://www.renesas.com/

Inquiries
https://www.renesas.com/contact/

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sept 19, 2018	All	Created document.

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.

   ¾ 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.

   ¾ 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.

   ¾ 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.

   ¾ 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.

   ¾ 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.

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

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

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

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

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

Colophon 7.2