RZ/A2M Group

RZ/A2M OSTM Driver

Introduction

This application note describes the operation of the software OSTM 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 e² studio and to be equipped with an RZ/A2M CPU Board.

Target Device

RZ/A2M Group

Driver Dependencies

This driver depends on:

- Drivers
    - STDIO
    - INTC (Interrupt Controller)
    - CPG (Clock Pulse Generator)
    - STB (Standby Module)

Referenced Documents

Document Type	Document Name	Document No.
User's Manual	RZ/A2M Hardware Manual	R01UH0746EJ
Application Note	RZ/A2M Smart Configurator User's Guide: e² studio	R20AN0583EJ
Application Note	OS Abstraction Middleware	R11AN0309EG

List of Abbreviations and Acronyms

Abbreviation	Full Form
ANSI	American National Standards Institute
API	Application Programming Interface
ARM	Advanced RISC Machines
CPG	Clock Pulse Generator
CPU	Central Processing Unit
HLD	High Layer Driver
IDE	Integrated Development Environment
INTC	Interrupt Controller
LLD	Low Layer Driver
OS	Operating System
OSTM	Operating System Timer Module
STB	Standby
STDIO	Standard Input/Output

Table 1-1 List of Abbreviations and Acronyms

Contents

1. Outline of Software Driver

The OSTM (Operating System Timer Module) driver controls the 3 timer channels provided by the RZ/A2M MPU.

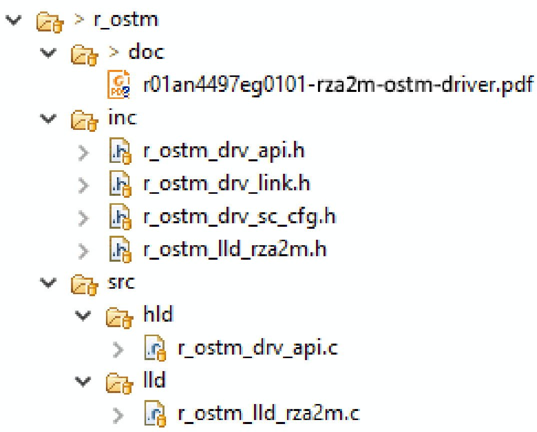
2. Description of the Software Driver

The key features of the driver include:

- Setting a timer into free-running mode for generating interrupts at non-fixed intervals
- Setting a timer into interval timer mode for generating interrupt requests at a fixed interval

2.1 Structure

The OSTM driver is split into two parts: the High Layer Driver (HLD) and the Low Layer Driver (LLD). The HLD includes platform independent features of the driver, implemented via the STDIO standard functions. The LLD includes all the hardware specific functions.



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_ostm_drv_api.h	Application	The only API header file to include in application code
High Layer Driver (HLD) Source		
r_ostm_hld_prv.h	Private (HLD only)	Private header file intended ONLY for use in High Layer Driver (HLD) source. NOT for application or Low Layer Driver (LLD) use
r_ostm_drv_api.c	Private (HLD only)	High Layer Driver (HLD) source code enabling the driver API functions
r_ostm_hld_prv.c	Private (HLD only)	High Layer Driver (HLD) private source code enabling the functionality of the driver, abstracted from the low level access
High Layer to Low Level API		
r _ostm_lld_xxxx.h	Private (HLD/LLD only)	Low Layer Driver (LLD) header file (where "xxxx" is a device and board-specific identification). Intended ONLY to provide access for High Layer Driver (HLD) to required Low Layer Driver functions (LLD). Not for use in application, not to define any device specific enumerations or structures
r_ostm_lld_cfg_xxxx.h	Private (HLD/LLD only)	Low Layer Driver (LLD) header file (where "xxxx" is a device and board-specific identification). Intended for definitions of device specific settings (in the form of enumerations and structures). No LLD functions to be defined in this file
Abstraction Link between High and Low Layer Drivers (HLD/LLD Link)		
r_ostm_drv_link.h	Private (HLD/LLD only)	Header file intended as an abstraction between low and high layer. This header will include the device specific configuration file "r_ostm_lld_xxxx.h"
r_ostm_device_cfg.h	Should be included in "r_ostm_drv_api.h"	Header file intended as an abstraction between low and high layer. This header will include the device specific configuration file "r_ostm_lld_cfg_xxxx.h"
Low Layer Driver (LLD) Source		
r_ostm_lld_xxxx.c	Private (LLD only)	(Where "xxxx" is a device and board specific identification). Provides the definitions for the Low Layer Driver interface.
Smart Configurator		
r_ostm_drv_sc_cfg.h	Private (HLD/LLD only)	This file is intended to be used by Smart Configurator to pass setup information to the driver. This is not for application use

2.3 Driver API

The driver can be either used through STDIO or through direct access. It is recommended not to mix both access methods.

The API functions can be seen in the table below:

Return Type	Function	Description	Arguments	Return
int_t	**ostm_hld_open(***st_stream_ptr_t* p_stream**)**	Driver initialisation interface is mapped to open function called directly using the st_r_driver_t OSTM driver handle g_ostm_driver: i.e. **g_ostm_driver.open()**	[in] **p_stream** driver handle	>0: the handle to the driver **DRV_ERROR** Open failed
void	**ostm_hld_close(** *st_stream_ptr_t* p_stream**)**	Driver close interface is mapped to close function. Called directly using the st_r_driver_t OSTM driver structure g_ostm_driver: i.e. **g_ostm_driver.close()**	[in] **p_stream** driver handle	**None**
int_t	**ostm_hld_control(** *st_stream_ptr_t* p_stream**,** *uint32_t* ctl_code**,** *void ** p_ctl_struct**)**	Driver control interface function. Maps to ANSI library low level control function. Called directly using the st_r_driver_t OSTM driver structure g_ostm_driver: i.e. **g_ostm_driver.control()**	[in] **p_stream** driver handle. [in] **ctl_code** the type of control function to use. [in/out] **p_ctl_struct** Required parameter is dependent upon the control function.	**DRV_SUCCESS** Operation succeeded **DRV_ERROR** Operation failed
int_t	**ostm_get_version(** *st_stream_ptr_t* p_stream**,** *st_ver_info_ptr_t* p_ver_info**)**	Driver get_version interface function. Maps to extended non-ANSI library low level get_version function. Called directly using the st_r_driver_t OSTM driver structure g_ostm_driver: i.e. **g_ostm_driver.get_version ()**	[in] **p_stream** Handle to the (pre-opened) channel. [out] **p_ver_info** Pointer to a version information structure.	**DRV_SUCCESS** Operation succeeded

These High Layer functions can be accessed either executed directly or through STDIO.

3. Accessing the Driver

3.1 STDIO

The API can be accessed through the ANSI ‘C’ library <stdio.h>. The following table details the operation of each function:

Operation	Return	Function Details
open	gs_stdio_handle, unique handle to driver	open(DEVICE_IDENTIFIER "ostm", O_RDWR);
close	DRV_SUCCESS successful operation, or driver specific error	close(gs_stdio_handle);
read	DRV_ERROR (read is not implemented in this OSTM driver)	read(gs_stdio_handle, buffer, buffer_length)
write	DRV_ERROR (write is not implemented in this OSTM driver)	write(gs_stdio_handle, buffer, data_length)
control	DRV_SUCCESS control was process, or driver specific error	control(gs_stdio_handle, CTRL, &struct);
get_version	DRV_SUCCESS drv_info was updated, or DRV_ERROR drv_info was not updated	get_version(DEVICE_IDENTIFIER "ostm", &drv_info);

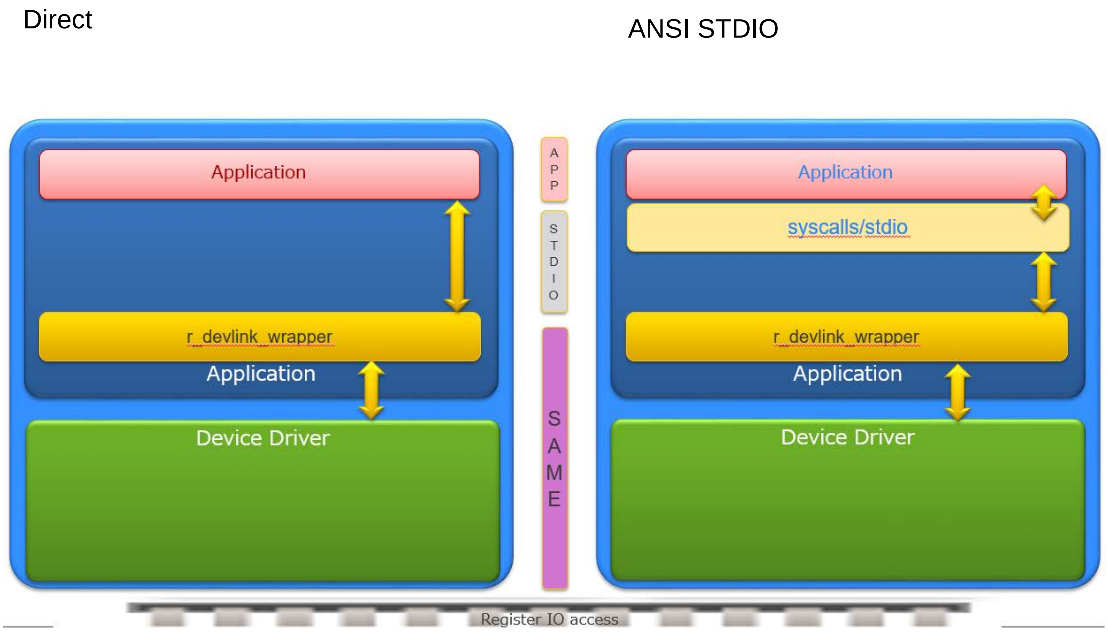
3.2 Direct

The following table shows the available direct functions.

Operation	Return	Function details
open	gs_direct_handle unique handle to driver	direct_open("ostm", 0);
close	DRV_SUCCESS successful operation, or driver specific error	direct_close(gs_direct_handle);
read	DRV_ERROR (read is not implemented in this OSTM driver)	direct_read(gs_direct_handle, buff, data_length);
write	DRV_ERROR (write not implemented in this OSTM driver)	direct_write(gs_direct_handle, buff, data_length);
control	DRV_SUCCESS control was processed, or driver specific error	direct_control(gs_direct_handle, CTRL, &struct);
get_version	DRV_SUCCESS drv_info was updated, or DRV_ERROR drv_info was not updated	direct_get_version("ostm", &drv_info);

3.3 Comparison

The diagram below illustrates the difference between the direct and ANSI STDIO methods.



4. Example of Use

This section gives simple examples for opening the driver, starting a timer, stopping a timer, reconfiguring a timer, closing the driver, and finally getting the driver version.

4.1 Open

```
int_t gs_ostm_handle;
char_t *drv_name = "\\\\.\\ostm";

/* Note that the text "\\\\.\\" in the drive name signifies to the STDIO
interface that the handle is to a peripheral and is not an access to a
standard file-based structure */


gs_ostm_handle = open(drv_name, O_RDWR);
```

4.2 Control – Start Timer

```
e_stb_module_t module;
int_t result;

module = MODULE_JCU;

result = control(gs_ostm_handle, CTRL_OSTM_START_TIMER, (void *) &module);
```

4.3 Control – Stop Timer

```
result = control(gs_ostm_handle, CTRL_OSTM_STOP_TIMER, (void *) &module);
```

4.4 Control – Reconfigure

```
result = control(gs_ostm_handle, CTRL_OSTM_RECONFIGURE, (void *) &module);
```

4.5 Write

The stdio write() function is not supported by the OSTM device driver.

4.6 Read

The stdio read() function is not supported by the OSTM device driver.

4.7 Close

```
close(gs_ostm_handle);
```

4.8 Get Version

```
st_ver_info_t info;
result = get_version(gs_ostm_handle, &info);
```

5. OS Support

Operating system support for this driver is available using the OS abstraction module. For more details, please refer to the OS abstraction module application note (R11AN0309EG).

6. How to Import the Driver

6.1 e² studio

Please refer to the RZ/A2M Smart Configurator User's Guide: e² studio (R20AN0583EJ) for details on how to import drivers into projects in e2 studio using the Smart Configurator tool.

6.2 For Projects created outside e² studio

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

1) Copy the driver to the location in the source tree that you require for your project.
2) Add the link to where you copied your driver to the compiler.

Other required drivers, e.g. r_cbuffer, must be imported similarly.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sept 18, 2018	All	Created document.
1.01	May 08, 2019	10	Added SC Import Details. Removed LLD API.

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

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

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

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

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

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

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

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

Corporate Headquarters

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

Trademarks

Contact information

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