RZ/A2M Group

RZ/A2M Circular Buffer Driver

Introduction

This application note describes the operation of the software Circular Buffer 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 the OS abstraction module for memory allocation.

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
API	Application Programming Interface
ARM	Advanced RISC Machines
CPU	Central Processing Unit
FIFO	First In First Out
IDE	Integrated Development Environment
OS	Operating System

Table 1-1 List of Abbreviations and Acronyms

Contents

1. Outline of the Software Driver

The Circular Buffer implements a FIFO queue whose maximum size is defined when the buffer is created. Data can be added to the buffer until it is full, whereupon attempts to add further data will fail.

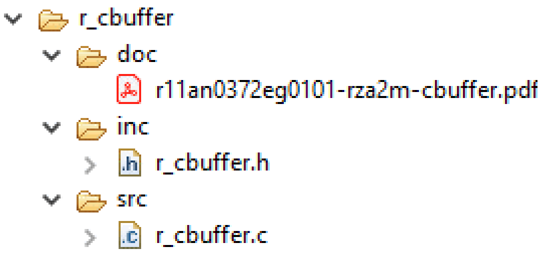
2. Description of the Software Driver

The key features of the driver include:

- Configurable buffer length on creation
- First in, first out
- Reading a single byte or a packet of bytes
- Writing a single byte or a packet of bytes
- Returning the amount of free space in the buffer
- Returning the number of bytes in the buffer
- Clearing the buffer

2.1 Structure

The Circular Buffer driver comprises a single source file and header file, plus this document.



2.2 Description of each file

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

Filename	Usage	Description
Driver API		
r_cbuffer.h	Driver header file	The API header file to include in application code
Driver Source		
r_cbuffer.c	Driver source code	Implements the driver API functions

2.3 Operation

The circular buffer allows data to written to and read from the buffer byte by byte, or in packets of bytes. As well as functions to create and destroy a circular buffer, functions are provided to clear the buffer, return the number of bytes in the buffer, and return the amount of free space in the buffer.

2.4 The Driver API

The driver provides the functions detailed below.

Type	Function	Arguments	Description	Return
PCBUFF	**cbCreate(** size_t stBufferSize **)**	 the size, in bytes, of the required buffer	Create a circular buffer of the desired size	Handle to circular buffer control structure on success; use on subsequent calls to reference this buffer. NULL on failure
void	**cbDestroy(** PCBUFF pcBuffer **)**	 handle of the buffer to destroy	Destroy a circular buffer	None
_Bool	**cbPut**(PCBUFF pcBuffer, uint8_t byData)	 handle of the circular buffer data to write to buffer	Put a byte into the buffer	**true:** byte added successfully **false:** failed (the buffer is full)
_Bool	**cbGet(** PCBUFF pcBuffer, uint8_t *pbyData **)**	 handle of the circular buffer on return, contains read data	Get a byte from the buffer	**true:** byte retrieved successfully **false:** failed (the buffer is empty)
size_t	**cbUsed(** PCBUFF pcBuffer **)**	 handle of the circular buffer	Return the number of bytes in the buffer	The number of bytes in the buffer
size_t	**cbFree(** PCBUFF pcBuffer **)**	 handle of the circular buffer	Return the number of bytes of free space in the buffer	The amount of free space in the buffer in bytes
_Bool	**cbFull(** PCBUFF pcBuffer **)**	 handle of the circular buffer	Determine if the buffer is full	**true:** the buffer is full **false:** the buffer is not full
void	**cbGetPacket(** PCBUFF pcBuffer, size_t stPacketLength, void *pDest **)**	 handle of the circular buffer number of bytes to read from the buffer pointer to the start of destination buffer	Read a packet of data without removing it from the buffer	None

Type	Function	Arguments	Description	Return
void	**cbCheckOut(** PCBUFF pcBuffer, size_t stPacketLength **)**	 handle of the circular buffer number of bytes to remove from the buffer	Delete a packet of data from the buffer (following a call to the **cbGetPacket()** function)	None
void	**cbPutPacket(** PCBUFF pcBuffer, size_t stPacketLength, void *pSrc **)**	 handle of the circular buffer number of bytes to add to the buffer pointer to the start of the data buffer	Write a packet of data to the buffer without updating the buffer's input pointer	None
void	**cbCheckIn(** PCBUFF pcBuffer, size_t stPacketLength **)**	 handle of the circular buffer number of bytes written to the buffer	Update the buffer's input pointer following a call to **cbPutPacket()**	None
size_t	**cbLinOut(** PCBUFF pcBuffer **)**	 handle of the circular buffer	Returns the number of bytes between the buffer output index and the input index, or the output index and the top of the buffer	The number of bytes
size_t	**cbLinIn(** PCBUFF pcBuffer **)**	 handle of the circular buffer	Returns the number of bytes between the buffer input index and the output index, or the input index and the top of the buffer	The number of bytes
void *	**cbInPointer(** PCBUFF pcBuffer **)**	 handle of the circular buffer	Returns a pointer to the input of the buffer	Pointer to the next input address
void *	**cbOutPointer(** PCBUFF pcBuffer **)**	 handle of the circular buffer	Returns a pointer to the output of the buffer	Pointer to the next output address

3. Example of Use

This section gives simple examples for creating a circular buffer, writing a byte, reading a byte, writing a packet, reading a packet, clearing the buffer, and finally destroying the buffer.

3.1 Packet Operations

The function to write a packet of data to the buffer does not itself check that there is enough space in the buffer for the specified packet size, so it is necessary to call **cbFree()** first to ensure that there is enough space in the buffer. It also does not update the buffer input pointer following the write, so **cbCheckIn()** must be called following the write to do this.

Similarly, **cbUsed()** should be called before the packet read function **cbGetPacket()** to verify that there is sufficient data in the buffer. Then **cbCheckOut()** should be called to update the buffer output pointer.

See the examples below for further information.

3.2 Create Circular Buffer

```
PCBUFF pcBuffer;
size_t stBufferSize = 1000;

/* create a 1,000 byte circular buffer */
pcBuffer = cbCreate(stBufferSize);
```

3.3 Write a Byte

```
_Bool success;
uint8_t byData = 42;

/* write a single byte to the circular buffer */
success = cbPut(pcBuffer, byData);
```

3.4 Read a Byte

```
/* read a single byte from the circular buffer */
success = cbGet(pcBuffer, &byData);
```

3.5 Write a Packet

```
size_t stPacketLength = 100;
char write_buffer[100];

/* if there’s enough free space then write a packet to the buffer */
if (cbFree(pcBuffer) >= stPacketLength)
{
    cbPutPacket(pcBuffer, stPacketLength, (void *) write_buffer);
    cbCheckIn(pcBuffer, stPacketLength);
}
```

3.6 Read a Packet

```
char read_buffer[100];

/* if there’s enough data in the buffer then read a packet */
if (cbUsed(pcBuffer) >= stPacketLength)
{
    cbGetPacket(pcBuffer, stPacketLength, (void *) read_buffer);
    cbCheckOut(pcBuffer, stPacketLength);
}
```

3.7 Clear the Buffer

```
/* delete everything from the buffer */
cbClear(pcBuffer);
```

3.8 Destroy the Buffer

```
cbDestroy(pcBuffer);
```

4. 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).

5. How to Import the Driver

5.1 e^2 studio

Please refer to the RZ/A2M Smart Configurator User's Guide: e^2 studio R20AN0583EJ for details on how to import drivers into projects in e2 studio using the Smart Configurator tool.

5.2 For Projects created outside e^2 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_cpg, must be imported similarly.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.19.19	All	Created document.
1.01	May.08.19	8	Added Import Details

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/.