RZ/A2M Group

RZ/A2M SCIFA Driver

Introduction

This application note describes the operation of the software SCIFA 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 depends on:

- Middleware:
  - Renesas OS Abstraction (FreeRTOS, or OSless version, see R11AN0309EG).
- Drivers
  - STDIO
  - INTC Driver
  - CPG Driver
  - GPIO Driver
  - CBuffer Driver
  - DMAC Driver (optional)

Referenced Documents

Document Type	Document Name	Document No.
User's Manual	RZ/A2M Hardware Manual	R01UH0746EJ
Application Note	RZ/A2M SCIFA using DMAC Application Note	R11AN0358EG
Application Note	OS Abstraction Middleware	R11AN0309EG
Application Note	RZ/A2M Smart Configurator User's Guide: e² studio	R20AN0583EJ

List of Abbreviations and Acronyms

Abbreviation	Full Form
ANSI	American National Standards Institute
API	Application Programming Interface
ARM	Advanced RISC Machine
CPG	Clock Pulse Generator
CPU	Central Processing Unit
DMAC	Direct Memory Access Controller
FIFO	First In First Out
GPIO	General Purpose Input/Output
HLD	High Layer Driver
IDE	Integrated Development Environment
INTC	INTerrupt Controller
LLD	Low Layer Driver
MCU	Microcontroller Unit
MODEM	MOdulate DEModulate
OS	Operating System
RISC	Reduced Instruction Set Computer
RTX	Short for CMSIS-RTOS Keil RTX real-time operating system
RX	Receive
RXI	Receive FIFO data full Interrupt
SCIFA	Serial Communications Interface with FIFO
STDIO	Standard Input/Output
TX	Transmit
TXI	Transmit data empty Interrupt

Table 1-1 List of Abbreviations and Acronyms

Contents

1. Outline of SCIFA Driver

The MCU provides the ‘Serial Communications Interface with FIFO’ (SCIFA) peripheral. The peripheral has five channels that support both asynchronous and clock synchronous serial communication. The SCIFA makes use of a 16-stage FIFO buffers for both transmission and reception to allow for efficient high-speed continuous communication.

For further information regarding the hardware specifics of the SCIFA peripheral, please refer to the hardware manual (R01UH0746EJ).

For further information regarding using the DMAC features with a SCIFA peripheral, please refer to the RZ/A2M SCIFA using DMAC Application Note (R11AN0358EG).

2. Description of the Software Driver

The key features of the driver include selectable:

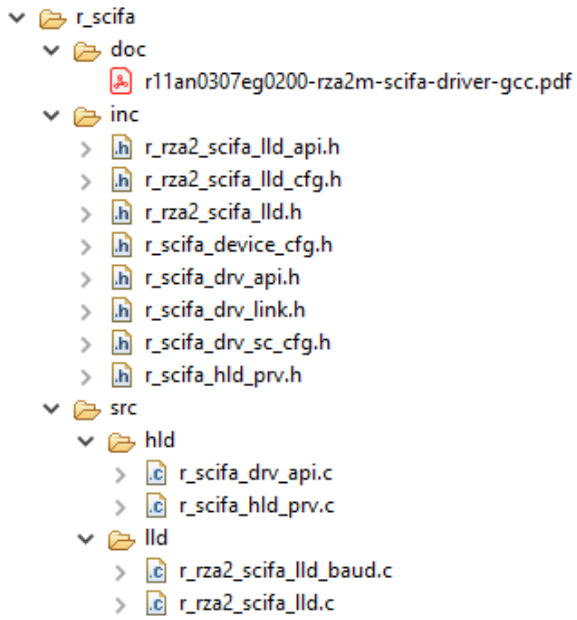
- Channels
- Baud rates
- Data bits
- Stop bits
- Parity
- Transmission mode: asynchronous or clock synchronous
- Data order
- Loopback mode

The extended features include:

- Configurable transmit FIFO trigger
- Configurable receive FIFO trigger
- MODEM mode
- Configure external clocks
- Noise cancellation
- DMAC transfers

2.1 Structure

The SCIFA 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_scifa_drv_api.h	Application	The only API header file to include in application code.
High Layer Driver (HLD) Source		
r_scifa_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_scifa_drv_api.c	Private (HLD only)	High Layer Driver (HLD) source code enabling the driver API functions.
r_scifa_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 Layer API		
r_xxxx_scifa_lld.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_xxxx_scifa_lld_api.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).
r_xxxx_scifa_lld_cfg.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_scifa_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 config file "r_xxxx_scifa_lld.h".
r_scifa_device_cfg.h	Should be included in "r_scifa_drv_api.h"	Header file intended as an abstraction between low and high layer. This header will include the device specific config file "r_xxxx_scifa_lld_cfg.h".
Low Layer Driver (LLD) Source		
r_xxxx_scifa_lld.c	Private (LLD only)	(Where "xxxx" is a device and board specific identification). Provides the definitions for the Low Layer Driver interface.
r_xxxx_scifa_lld_baud.c	Private (LLD only)	Low Layer Driver function definitions for setting baud rate.
Smart Configurator		
r_scifa_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 not recommended to mix both access methods.

The API functions can be seen in the table below:

Return	Function	Arguments	Description	Return
int_t	**scifa_hld_open(** st_stream_ptr_t p_stream **)**	[in] driver handle.	Driver initialisation interface is mapped to open function called directly using the st_r_driver_t SCIFA driver handle g_scifa_driver: i.e. **g_scifa_driver.open()**	**DRV_SUCCESS** Open Success **DRV_ERROR** Open Error
void	**scifa_hld_close(** st_stream_ptr_t p_stream **)**	[in] driver handle.	Driver close interface is mapped to close function called directly using the st_r_driver_t SCIFA driver structure g_scifa_driver: i.e. **g_scifa_driver.close()**	**None**
int_t	**scifa_hld_read(** st_stream_ptr_t p_stream, uint8_t *p_buffer, uint32_t count **)**	[in] driver handle [out] buffer for returned data [in] size of buffer.	Driver close interface is mapped to read function called directly using the st_r_driver_t SCIFA driver structure g_scifa_driver: i.e. **g_scifa_driver.read()**	**Number of Bytes Read** **DRV_ERROR** Read Error
int_t	**scifa_hld_write(** st_stream_ptr_t p_stream, uint8_t *p_buffer, uint32_t count **)**	[in] driver handle [in] buffer containing data to send [in] size of data to send.	Driver write interface is mapped to write function called directly using the st_r_driver_t SCIFA driver structure g_scifa_driver: i.e. **g_scifa_driver.write()**	**DRV_SUCCESS** Write Success **DRV_ERROR** Write Error
int_t	**scifa_hld_control(** st_stream_ptr_t p_stream**,** uint32_t ctl_code**,** void* p_ctl_struct **)**	[in] driver handle [in] type of control function to use [in/out] dependent on control function.	Driver control interface function. Maps to ANSI library low level control function. Called directly using the st_r_driver_t SCIFA driver structure g_scifa_driver: i.e. **g_scifa_driver.control()**	**DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error

Return	Function	Arguments	Description	Return
int_t	**scifa_hld_get_version(** st_stream_ptr_t stream_ptr**,** st_ver_info_t *p_ver_info **)**	[in] driver handle [out] receives version information.	Driver get_version interface function Maps to extended non-ANSI library low level get_version function. Called directly using the st_r_driver_t SCIFA driver structure g_scifa_driver: i.e. **g_scifa_driver.get_version()**	**DRV_SUCCESS** Operation Success

2.4 Control Codes

The table below lists the command codes available for use with the Control function on the Open channel.

2.4.1 CTL_SCIFA_SET_CONFIGURATION

Structure	Options	Description
scifa_config_t	Allocate and fill before calling control command.	Configures the opened channel. **Returns**: **DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error
scifa_sync_mode_t sync_mode;	SCIFA_MODE_ASYNC SCIFA_MODE_SYNC	Asynchronous Synchronous
uint32_t baud_rate;	e.g. 9600, 19200, 115200	Required baud rate
scifa_clk_enable_t clk_enable;	SCIFA_CLK_SRC_INT_SCK_IN SCIFA_CLK_SRC_INT_SCK_OUT SCIFA_CLK_SRC_EXT_SCK_IN	use internal clock & set SCK as input use internal clock & set SCK as output use external clock & set SCK as input
scifa_base_clk_sel_t clk_base;	SCIFA_CLK_16X SCIFA_CLK_8X	Use 16x mode clock Use 8x mode clock (Asynchronous mode only.)
scifa_size_t data_size;	SCIFA_DATA_8BIT SCIFA_DATA_7BIT	8 data bits 7 data bits
scifa_parity_en_t parity_en;	SCIFA_PARITY_OFF SCIFA_PARITY_ON	No parity Parity in use

Structure		Options	Description
scifa_parity_t	parity_type;	SCIFA_NONE_PARITY SCIFA_EVEN_PARITY SCIFA_ODD_PARITY	No parity Even parity Odd parity
scifa_stop_t	stop_bits;	SCIFA_STOPBITS_1 SCIFA_STOPBITS_2	1 stop bit 2 stop bits
scifa_nfen_t	noise_canceller;	SCIFA_NOISE_CANCEL_DISABLE SCIFA_NOISE_CANCEL_ENSABLE	Noise cancellation disabled Noise cancellation enabled
scifa_dir_t	data_direction;	SCIFA_LSB_FIRST SCIFA_MSB_FIRST	Lowest significant bit first Highest significant bit first
scifa_loop_t	loopback;	SCIFA_LOOPBACK_DISABLE SCIFA_LOOPBACK_ENABLE	Loopback disabled Loopback enabled
scifa_mce_t	modem_control;	SCIFA_MODEM_CONTROL_DISABLE SCIFA_MODEM_CONTROL_ENABLE	Modem not used Modem controls enabled
scifa_rstrg_t	rts_trigger;	SCIFA_RTS_ACTIVE_TRIGGER_1 SCIFA_RTS_ACTIVE_TRIGGER_4 SCIFA_RTS_ACTIVE_TRIGGER_6 SCIFA_RTS_ACTIVE_TRIGGER_8 SCIFA_RTS_ACTIVE_TRIGGER_10 SCIFA_RTS_ACTIVE_TRIGGER_12 SCIFA_RTS_ACTIVE_TRIGGER_14 SCIFA_RTS_ACTIVE_TRIGGER_15	RTS# Output Active Trigger Number Select When the number of entries in the reception FIFO meets or exceeds the specified trigger number, the RTS# signal is raised.
uint8_t	tftc;	0-15	Transmit data trigger number is 0, 1, 2 … 15.
uint8_t	rftc;	1-16	Receive data trigger number is 1, 2 … 15, 16.
scifa_sptr_init_t	txd_init_value;	SCIFA_SPTR_INIT_LOW SCIFA_SPTR_INIT_HIGH	Initial TXD set low Initial TXD set high
scifa_sptr_init_t	sck_init_value;	SCIFA_SPTR_INIT_LOW SCIFA_SPTR_INIT_HIGH	Initial SCK set low Initial SCK set high
scifa_sptr_init_t	cts_init_value;	SCIFA_SPTR_INIT_LOW SCIFA_SPTR_INIT_HIGH	Initial CTS set low Initial CTS set high

Structure	Options	Description
scifa_sptr_init_t rts_init_value;	SCIFA_SPTR_INIT_LOW SCIFA_SPTR_INIT_HIGH	Initial RTS set low Initial RTS set high
scifa_tx_dt_mode_t tx_dt_mode;	SCIFA_TX_MODE_OFF SCIFA_TX_POLLING_MODE SCIFA_TX_INTERRUPT_MODE SCIFA_TX_DMA_MODE	not in use polling mode interrupt mode DMA mode
scifa_rx_dt_mode_t rx_dt_mode;	SCIFA_RX_MODE_OFF SCIFA_RX_POLLING_MODE SCIFA_RX_INTERRUPT_MODE SCIFA_RX_DMA_MODE	not in use polling mode interrupt mode DMA mode
uint8_t eri_bri_priority;	0-31	interrupt priority level of ERI_BRI
uint8_t rxi_priority;	0-31	interrupt priority level of RXI
uint8_t txi_priority;	0-31	interrupt priority level of TXI
uint8_t tei_dri_priority;	0-31	interrupt priority level of TEI_DRI
void (*write_complete_callback)(void)		Callback function for DMA write complete. [Note DMAC driver version 2.0 or greater is required]
void (*read_complete_callback)(void)		Callback function for DMA read complete. [Note DMAC driver version 2.0 or greater is required]
char *dma_tx		Pointer to description string of DMA module configuration to use for transmit, for example “scifa_tx”. [Note DMAC driver version 2.0 or greater is required]
char *dma_rx		Pointer to description string of DMA module configuration to use for receive, for example “scifa_rx”. [Note DMAC driver version 2.0 or greater is required]
scifa_read_blocking_mode_t read_blocking_mode	SCIFA_READ_BLOCKING_MODE_ENABLE SCIFA_READ_BLOCKING_MODE_DISABLE	Read blocks until at least one character has been received Read returns 0 if there is no data in the receive buffer

Structure	Options	Description
scifa_write_blocking_mode_t write_blocking_mode	SCIFA_WRITE_BLOCKING_MODE_ENABLE SCIFA_WRITE_BLOCKING_MODE_DISABLE	Write blocks until all of the supplied data has been buffered Write buffers as much data as it can, and returns the number of characters buffered
uint16_t receive_buffer_length	~16-32,768	The size of the software receive buffer
uint16_t transmit_buffer_length	~16-32,768	The size of the software transmit buffer

2.4.2 CTL_SCIFA_GET_CONFIGURATION

Structure	Options	Description
scifa_config_t Elements as CTL_SCIFA_SET_CONFIGURATION.	Allocate a new structure before calling; on a successful return, the structure will contain the configuration data.	Obtains the configuration data for the channel. **Returns**: **DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error

2.4.3 CTL_SCIFA_DMA_READ_ABORT

Structure	Options	Description
NULL	None	Aborts an in-progress DMA read operation. [Note DMAC driver version 2.0 or greater is required] **Returns**: **DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error

2.4.4 CTL_SCIFA_DMA_WRITE_ABORT

Structure	Options	Description
NULL	None	Aborts an in-progress DMA write operation. [Note DMAC driver version 2.0 or greater is required] **Returns**: **DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error

2.4.5 CTL_SCIFA_READ_NEXT

Structure	Options	Description
st_scifa_dma_rw_t	Allocate and fill before calling control command.	Configures a **DMA** read to be automatically executed as soon as the current one completes. [Note DMAC driver version 2.0 or greater is required] **Returns**: **DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error
`char *buffer`	Allocate buffer before calling; on a successful return, the buffer will contain the received data.	Buffer for received data.
`uint32_t length`		Maximum length of received data.

2.4.6 CTL_SCIFA_WRITE_NEXT

Structure	Options	Description
st_scifa_dma_rw_t	Allocate and fill before calling control command.	Configures a **DMA** write to be automatically executed as soon as the current one completes. [Note DMAC driver version 2.0 or greater is required] **Returns**: **DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error
`char *buffer`		Data to be transmitted.
`uint32_t length`		Length of data to transmit.

2.4.7 CTL_SCIFA_FLUSH_OUTPUT

Structure	Options	Description
`NULL`	None	Waits until the entire content of the software transmit buffer have been transferred to the transmit FIFO. **Returns**: **DRV_SUCCESS** Operation Success **DRV_ERROR** Operation Error

3. Accessing the Driver

3.1 STDIO

The API can be accessed through the ANSI ‘C’ Library <stdio.h>. This is the preferred method of accessing the driver. The following table details the operation of each function:

Operation	Return	Function Details
open	stdio_handle, unique handle to driver	open (DEVICE_IDENTIFIER "scifa0", O_RDWR);
close	DRV_SUCCESS successful operation, or driver specific error	close (stdio_handle);
read	Number of characters read, -1 on error	read (stdio_handle, buff, data_length);
write	Number of characters written, -1 on error	write (stdio_handle, buff, data_length);
control	DRV_SUCCESS control was process, or driver specific error	control (stdio_handle, CTRL, &struct);
get_version	DRV_SUCCESS drv_info was updated, or DRV_ERROR drv_info was not updated	get_version ("scifa0", &drv_info);

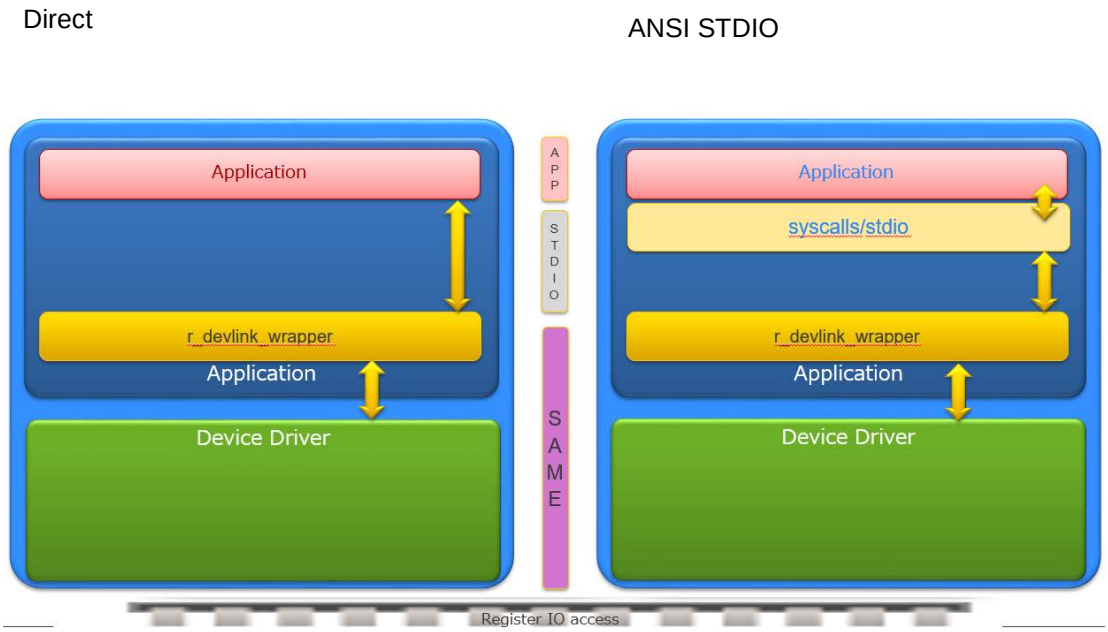
3.2 Direct

The following table shows the available direct functions.

Operation	Return	Function details
open	direct_handle unique handle to driver	direct_open ("scifa0", 0);
close	DRV_SUCCESS successful operation, or driver specific error	direct_close (direct_handle);
read	Number of characters read, -1 on error	direct_read (direct_handle, buff, data_length);
write	Number of characters written, -1 on error	direct_write (direct_handle, buff, data_length);
control	DRV_SUCCESS control was process, or driver specific error	direct_control (direct_handle, CTRL, &struct);
get_version	DRV_SUCCESS drv_info was updated, or DRV_ERROR drv_info was not updated	direct_get_version ("scifa0", &drv_info);

3.3 Comparison

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



4. Example of Use

This section describes a simple example of opening the driver, configuring the driver, transmitting and receiving data and closing a driver.

Note the minimum heap size will be H'256 bytes.

4.1 Preparation

```
#include <stdio.h>
#include <unistd.h>
#include "r_typedefs.h"
#include "r_scifa_drv_api.h"
#include "compiler_settings.h"

int_t result;
int_t scifa_handle;
uint8_t ch0_drv_name[] = “\\\\.\\scifa0";

/* 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 */
```

4.2 Open

```
scifa_handle = open(ch0_drv_name, O_RDWR);
```

4.3 Get Version

```
st_ver_info_t info;

result = get_version(ch0_drv_name, &info);
```

4.4 Control – Set Configuration Settings

```
scifa_config_t set_cfg;

set_cfg.sync_mode = SCIFA_MODE_ASYNC;
set_cfg.baud_rate = 115200u;
set_cfg.clk_enable = SCIFA_CLK_SRC_INT_SCK_IN;
set_cfg.clk_base = SCIFA_CLK_16X;
set_cfg.data_size = SCIFA_DATA_8BIT;
set_cfg.parity_en = SCIFA_PARITY_ON;
set_cfg.parity_type = SCIFA_EVEN_PARITY;
set_cfg.stop_bits = SCIFA_STOPBITS_1;
set_cfg.noise_canceller = SCIFA_NOISE_CANCEL_DISABLE;
set_cfg.data_direction = SCIFA_LSB_FIRST;
set_cfg.loopback = SCIFA_LOOPBACK_DISABLE;
set_cfg.modem_control = SCIFA_MODEM_CONTROL_DISABLE;
set_cfg.rts_trigger = SCIFA_RTS_ACTIVE_TRIGGER_15;
set_cfg.tftc = 0u;
set_cfg.rftc = 0u;
set_cfg.txd_init_value = SCIFA_SPTR_INIT_HIGH;
set_cfg.sck_init_value = SCIFA_SPTR_INIT_HIGH;
set_cfg.cts_init_value = SCIFA_SPTR_INIT_HIGH;
set_cfg.rts_init_value = SCIFA_SPTR_INIT_HIGH;
set_cfg.tx_dt_mode = SCIFA_TX_INTERRUPT_MODE;
set_cfg.rx_dt_mode = SCIFA_RX_INTERRUPT_MODE;
set_cfg.eri_bri_priority = 9u;
set_cfg.rxi_priority = 10u;
set_cfg.txi_priority = 10u;
```

```
set_cfg.tei_dri_priority = 9u;
set_cfg.write_complete_callback = NULL;
set_cfg.read_complete_callback = NULL;
set_cfg.dma_rx = NULL;
set_cfg.dma_tx = NULL;
set_cfg.read_blocking_mode = SCIFA_READ_BLOCKING_MODE_ENABLE;
set_cfg.write_blocking_mode = SCIFA_WRITE_BLOCKING_MODE_ENABLE;
set_cfg.receive_buffer_length = 1024u;
set_cfg.transmit_buffer_length = 4096u;

result = control(scifa_handle, CTL_SCIFA_SET_CONFIGURATION, &set_cfg);
```

4.5 Control – Get Configuration Settings

```
scifa_config_t get_cfg;

result = control(scifa_handle, CTL_SCIFA_GET_CONFIGURATION, &get_cfg);
```

4.6 Write

```
uint8_t data_out[] = "Data to send\r\n";

result = write(scifa_handle, data_out, sizeof(data_out));
```

4.7 Read

```
uint8_t data_in[100];

result = read(scifa_handle, data_in, sizeof(data_in));
```

4.8 Close

```
close(scifa_handle);
```

5. Transmit and Receive Buffers

When the driver is in SCIFA_TX_INTERRUPT_MODE, data sent to the driver using the write() function is transferred to the driver's software transmit buffer. The application program can then continue processing while the driver transmits the data from the software buffer in the background using interrupts to transfer data from the buffer to the transmit FIFO as required.

The write() function can be configured to block (SCIFA_WRITE_BLOCKING_MODE_ENABLE), or not block (SCIFA_WRITE_BLOCKING_MODE_DISABLE). In blocking mode, if write() is called with more data than there is space for in the transmit buffer, then the function will wait until all the data supplied has been transferred to the transmit buffer. In non-blocking mode with more data than will fit into the transmit buffer, write() will fill the transmit buffer and then return the number of characters successfully transferred to the buffer.

When the driver is in SCIFA_RX_INTERRUPT_MODE, data received by the driver will be placed into the driver's software receive buffer under interrupts, allowing the application to continue processing while data is received in the background. The application can then call read() to retrieve received data from the buffer when it is required.

The read() function also supports a blocking mode (SCIFA_READ_BLOCKING_MODE_ENABLE) and a non-blocking mode (SCIFA_READ_BLOCKING_MODE_DISABLE). In blocking mode the read() function will not return until at least one character has been received. In non-blocking mode, read() will return a length of 0 if no data has been received.

The sizes of both software buffers are configurable, either using Smart Configurator, or by calling the control() function with the CTL_SCIFA_SET_CONFIGURATION control code during program execution.

6. 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).

7. How to Import the Driver

7.1 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.

7.2 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

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sept 25, 2018	All	Created document.
1.01	Apr.03.19	1	Driver Dependencies updated to include CBuffer
		9	Section 2.4 Low Layer Driver API removed (internal use only)
		15	Section 4 Updated sample code to match driver V.1.01
		16	Section 6 Clarified destination include path
		All	Template update to 2019
1.02	Apr.04.19	1	Added DMAC dependency
		4	Section 1 Added cross-reference to DMAC Application Note
		11	Section 4 Updated sample code to match driver V.1.02
1.03	Apr.24.19	8	Added Control commands.
		17	Added Smart Configurator reference.
2.00	Jun.12.19	5	Updated driver structure diagram
		6	Changed file name
		13	Added CTL_SCIFA_FLUSH_OUTPUT control code
		10, 11, 18	Updated configuration structure
		19	Added transmit and receive buffer description

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.

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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

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

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/.