RX111, RX210, RX631, RX63N Group

FIT SCI Asynchronous Mode Module

R01AN1667EU0200
Rev. 2.00
May 29, 2013

Introduction

This module provides UART channel support for all channels of the RX111, RX210 and RX63N SCI peripheral. Channels may be configured on an individual basis, with disabled channels allocating no resources. The code is re-entrant and as such is suitable for inclusion with RTOSs.

Target Device

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

- **RX111**
- **RX210**
- **RX63N**

Contents

1. Overview

This SCI Asynchronous Mode driver supports the SCIe and SCIf peripherals on the RX111, and the SCIc and SCId peripherals on the RX210 and RX63N. The hardware functionality is detailed in Chapter 26 of the RX111 Hardware User's Manual, Chapter 28 of the RX210 User's Manual, and Chapter 35 of the RX63N User's Manual. All basic UART functionality is supported by this driver. Additionally, the driver supports the following features:

- noise cancellation.
- outputting baud clock on the SCK pin.
- one-way flow control of either CTS or RTS.

Features not supported by this driver are:

- extended mode (channel 12)
- multiprocessor mode (all channels).
- event linking (channel 5).
- DTC data transfer.

This is a multi-channel driver, and it supports all channels present on the peripheral. Specific channels can be excluded via compile-time equates to reduce driver RAM usage and code size if desired. These equates are specified in "r_sci_ascync_rx_config.h".

An individual channel is initialized in the application by calling R_SCI_Open(). This function applies power to the peripheral and initializes UART-type settings. A handle is returned from this function to uniquely identify the channel. The handle references an internal driver structure that maintains pointers to the channel's register set, queue structures, and other critical information. It is also used an argument for the other API functions.

This driver is interrupt-driven and non-blocking. As such, circular buffers are used to queue incoming and outgoing data. The size of these buffers can also be set using equates in "r_sci_ascync_rx_config.h". Interrupts supported by this driver are TXI, TEI, RXI, and ERI (Group12 on RX63N).

The TXI interrupt occurs when a byte in the TDR register has been shifted into the TSR register for transmit on the TXDn pin. During this interrupt, the next byte in the transmit circular buffer (which is loaded by R_SCI_Send()) is placed into the TDR register to be ready for transmit. The TEI interrupt occurs only after the last bit of the last byte from the transmit queue has been shifted out of the TSR register. If a callback function is provided in the R_SCI_Open() call, it is called here with a TEI event passed to it.

Support for TEI interrupts may be removed from the driver via a setting in "r_sci_async_rx_config.h". If it is to be included, it must also be enabled for a channel though an R_SCI_Control() command.

The RXI interrupt occurs each time the RDR register has shifted in a byte. This byte is loaded into the receive circular buffer during the interrupt for access later via an R_SCI_Receive() call at the application level. If a callback function is provided, it is called with a receive event and the received byte. If the receive queue is full, it is called with a queue full event and the received/unstored byte.

The ERI/Group12 interrupt occurs when a framing, overrun, or parity error is detected by the receiver hardware. If a callback function is provided, the interrupt determines which error occurred and notifies the application of the event and provides the contents of the RDR register. Whether a callback function is provided or not, the interrupt clears the error condition by writing "0" to the SSR error flags repeatedly until the condition clears. If the error condition is not cleared, no more data can be received on that channel.

2. API Information

This Driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires your MCU support the following features:

- SCI peripheral

2.2 Hardware Resource Requirements

This section details the hardware peripherals that this driver requires. Unless explicitly stated, these resources must be reserved for the driver and the user cannot use them.

2.2.1 SCI

This driver makes use of the SCI peripheral. Individual channels may be omitted by this driver by disabling them in the r_sci_async_config.h file.

2.2.2 GPIO

This driver utilizes port pins corresponding to each individual channel. These pins may not be used for GPIO.

2.3 Software Requirements

This driver is dependent upon the following packages:

- r_bsp
- r_byteq

2.4 Limitations

No software limitations.

2.5 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v1.02

2.6 Header Files

All API calls and their supporting interface definitions are located in r_sci_async_if.h. Compile time configurable options are located in r_sci_async_config.h. Both of these files should be included by the User's application.

2.7 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*.

2.8 Configuration Overview

All configurable options that can be set at build time are located in the file “r_sci_async_config.h”. A summary of these settings are provided in the following table:

Configuration options in *r_sci_async_config.h*	
`#define SCI_CFG_PARAM_CHECKING_ENABLE 1`	If this equate is set to 1, parameter checking is included in the build. If the equate is set to 0, the parameter checking is omitted from the build. Setting this equate to BSP_CFG_PARAM_CHECKING_ENABLE utilizes the system default setting.

`#define SCI_CFG_CH0_INCLUDED 0`
`#define SCI_CFG_CH1_INCLUDED 1`
`#define SCI_CFG_CH2_INCLUDED 0`
`#define SCI_CFG_CH3_INCLUDED 0`
`#define SCI_CFG_CH4_INCLUDED 0`
`#define SCI_CFG_CH5_INCLUDED 1`
`#define SCI_CFG_CH6_INCLUDED 0`
`#define SCI_CFG_CH7_INCLUDED 0`
`#define SCI_CFG_CH8_INCLUDED 0`
`#define SCI_CFG_CH9_INCLUDED 0`
`#define SCI_CFG_CH10_INCLUDED 0`
`#define SCI_CFG_CH11_INCLUDED 0`
`#define SCI_CFG_CH12_INCLUDED 1`
	Each channel has associated with it transmit and receive buffers, counters, interrupts, and other program and RAM resources. Setting a #define to 1 allocates resources for that channel.

`#define SCI_CFG_CH0_TX_BUFSIZ 0`
`#define SCI_CFG_CH1_TX_BUFSIZ 80`
`#define SCI_CFG_CH2_TX_BUFSIZ 0`
`#define SCI_CFG_CH3_TX_BUFSIZ 0`
`#define SCI_CFG_CH4_TX_BUFSIZ 0`
`#define SCI_CFG_CH5_TX_BUFSIZ 80`
`#define SCI_CFG_CH6_TX_BUFSIZ 0`
`#define SCI_CFG_CH7_TX_BUFSIZ 0`
`#define SCI_CFG_CH8_TX_BUFSIZ 0`
`#define SCI_CFG_CH9_TX_BUFSIZ 0`
`#define SCI_CFG_CH10_TX_BUFSIZ 0`
`#define SCI_CFG_CH11_TX_BUFSIZ 0`
`#define SCI_CFG_CH12_TX_BUFSIZ 80`
	These #defines specify the size of the buffer to use for the transmit queue on each channel. If the corresponding SCI_CHn_INCLUDED is not set to 1, the buffer is not allocated.

`#define SCI_CFG_CH0_RX_BUFSIZ 0`
`#define SCI_CFG_CH1_RX_BUFSIZ 80`
`#define SCI_CFG_CH2_RX_BUFSIZ 0`
`#define SCI_CFG_CH3_RX_BUFSIZ 0`
`#define SCI_CFG_CH4_RX_BUFSIZ 0`
`#define SCI_CFG_CH5_RX_BUFSIZ 80`
`#define SCI_CFG_CH6_RX_BUFSIZ 0`
`#define SCI_CFG_CH7_RX_BUFSIZ 0`
`#define SCI_CFG_CH8_RX_BUFSIZ 0`
`#define SCI_CFG_CH9_RX_BUFSIZ 0`
`#define SCI_CFG_CH10_RX_BUFSIZ 0`
`#define SCI_CFG_CH11_RX_BUFSIZ 0`
`#define SCI_CFG_CH12_RX_BUFSIZ 80`
	These #defines specify the size of the buffer to use for the receive queue on each channel. If the corresponding SCI_CHn_INCLUDED is not set to 1, the buffer is not allocated.

`#define SCI_CFG_TEI_INCLUDED 0`	Setting this #define to 1 causes the Transmit Buffer Empty interrupt code to be included. An R_SCI_Control() command is used to enable TEI interrupts for a particular channel. This interrupt only occurs when the last bit of the last byte of data has been sent and the transmitter has become idle. The interrupt calls the user's callback function (specified in R_SCI_Open()) and passes it an

	SCI_EVT_TEI event. A common use of this feature is to disable an external transceiver to save power. It would then be up to the application to re-enable the transceiver before sending again.

2.9 API Data Structures

The API data structures are located in the file “r_sci_async_rx_if.h” and discussed in Section 3.

2.10 Adding Driver to Your Project

Follow the steps below to add the driver’s code to your project:

1. Add the r_sci_async_rx and r_config folders to your project.
2. Add a project include path for the “r_sci_async_rx” directory.
3. Add a project include path for the “r_sci_async_rx\src” directory.
4. Add a project include path for the “r_config” directory.
5. Open "r_config\r_sci_async_rx_config.h" file and configure the driver for your project.
6. Add a #include for “r_sci_async_rx_if.h” to any source files that need to use the API functions.

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_SCI_Open()	Applies power to the SCI channel, initializes the associated registers, enables interrupts, and provides the channel handle for use with other API functions. Takes a callback function pointer for notifying the user at interrupt level whenever a receiver error or other interrupt events have occurred.
R_SCI_Close()	Removes power to the SCI channel and disables the associated interrupts.
R_SCI_Send()	Queues data and initiates transmit if transmitter is not in use.
R_SCI_Receive()	Fetches data from a queue which is filled by RXI interrupts.
R_SCI_Control()	Handles special hardware or software operations for the SCI channel.
R_SCI_GetVersion()	Returns at runtime the driver version number.

3.2 Return Values

This shows the different values API functions can return. This enum is found in r_sci_multi_ch_if.h along with the API function declarations.

```
typedef enum e_sci_err      // SCI API error codes
{
    SCI_SUCCESS=0,
    SCI_ERR_BAD_CHAN,       // non-existent channel number
    SCI_ERR_OMITTED_CHAN,   // SCI_CHx_INCLUDED is 0 in config.h
    SCI_ERR_CH_NOT_CLOSED,  // channel still running in another mode
    SCI_ERR_BAD_MODE,       // unsupported or incorrect mode for channel
    SCI_ERR_QUEUE_UNAVAILABLE,  // can't open tx or rx queue or both
    SCI_ERR_INVALID_ARG,    // argument is not valid for parameter
    SCI_ERR_INSUFFICIENT_SPACE, // not enough space in transmit queue
    SCI_ERR_INSUFFICIENT_DATA,  // not enough data in receive queue
    SCI_ERR_NULL_PTR        // received null ptr; missing required argument
} sci_err_t;
```

3.3 R_SCI_Open()

This function applies power to the SCI channel, initializes the associated registers, enables interrupts, and provides the channel handle for use with other API functions.

Format

```
sci_err_t  R_SCI_Open(uint8_t  const        chan,
                      sci_mode_t  const     mode,
                      void * const          p_cfg,
                      void                  (* const p_callback)(void *p_args),
                      sci_hdl_t * const     p_hdl);
```

Parameters

chan
Channel to initialize; 1, 5, or 12

mode
Operational mode (see enumeration below)

p_cfg
Pointer to configuration structure (see below) specific to mode, casted to void *

p_callback
Pointer to function called from interrupt when an RXI or receiver error is detected or for transmit end (TEI) condition

p_hdl
Pointer to a handle for channel (value set here)

At present, this driver only supports the Asynchronous mode of the SCI peripheral. An enumeration is provided for future expansion.

```
typedef enum e_sci_mode     // SCI operational modes
{
    SCI_MODE_OFF=0,         // channel not in use
    SCI_MODE_ASYNC          // Asynchronous
} sci_mode_t;
```

The following #defines indicate some configurable options for Asynchronous Mode. These values correspond to bit definitions in the SMR register.

```
#define SCI_CLK_INT         0x00    // use internal clock for baud generation
#define SCI_CLK_EXT_8X      0x03    // use external clock 8x baud rate
#define SCI_CLK_EXT_16X     0x02    // use external clock 16x baud rate
#define SCI_DATA_7BIT       0x40
#define SCI_DATA_8BIT       0x00
#define SCI_PARITY_ON       0x20
#define SCI_PARITY_OFF      0x00
#define SCI_ODD_PARITY      0x10
#define SCI_EVEN_PARITY     0x00
#define SCI_STOPBITS_2      0x08
#define SCI_STOPBITS_1      0x00
```

The complete runtime configurable options for Asynchronous mode are declared in the structure below. This structure is cast into a void pointer to allow for potential alternate mode configurations in the future.

```
typedef struct st_sci_uart
{
    uint32_t    baud_rate;      // ie 9600, 19200, 115200 (with internal clock)
    uint8_t     clk_src;        // use SCI_CLK_INT/EXT8/EXT16
    uint8_t     data_size;      // use SCI_DATA_nBIT
    uint8_t     parity_en;      // use SCI_PARITY_ON/OFF
    uint8_t     parity_type;    // use SCI_ODD/EVEN_PARITY
    uint8_t     stop_bits;      // use SCI_STOPBITS_1/2
    uint8_t     int_priority;   // txi,tei,rxi,eri INT priority; 1=low, 15=high
} sci_uart_t;
```

Return Values

SCI_SUCCESS:	*Successful; channel initialized*
SCI_ERR_BAD_CHAN:	*Channel number is invalid for part*
SCI_ERR_OMITTED_CHAN:	*Corresponding SCI_CHx_INCLUDED is 0*
SCI_ERR_CH_NOT_CLOSED:	*Channel currently in operation; Perform R_SCI_Close() first*
SCI_ERR_BAD_MODE:	*Specified mode not currently supported*
SCI_ERR_NULL_PTR:	*p_cfg pointer is NULL*
SCI_ERR_INVALID_ARG:	*An element of the p_cfg structure contains an invalid value.*
SCI_ERR_QUEUE_UNAVAILABLE:	*Cannot open transmit or receive queue or both*

Properties

Prototyped in file “r_sci_async_if.h”

Description

Initializes an SCI channel for a particular mode and provides a Handle in **pHdl* for use with other API functions. TXI, RXI, and ERI interrupts are enabled. See Control() for enabling TEI interrupts.

Reentrant

Function is re-entrant for different channels.

Example

```
    sci_uart_t  config;
    sci_hdl_t   Console;
    sci_err_t   err;

    config.baud_rate = 115200;
    config.clk_src = SCI_CLK_INT;
    config.data_size = SCI_DATA_8BIT;
    config.parity_en = SCI_PARITY_OFF;
    config.parity_type = SCI_EVEN_PARITY;   // ignored because parity is disabled
    config.stop_bits = SCI_STOPBITS_1;
    config.int_priority = 2;                // 1=lowest, 15=highest

    err = R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, (void *)&config, MyCallback, &Console);
```

Special Notes:

If an internal clock is used, there should be 1 bit time delay after calling R_SCI_Open() (or more specifically, after setting the BRR register within this function) and performing communications on the channel to allow time for the clock to settle. Often the application startup processing consumes this time and no extra delay is required. The driver uses an algorithm for calculating the optimum values for BRR, SEMR.ABCS, and SMR.CKS using BSP_PCLKB_HZ as defined in mcu_info.h of the board support package. This however does not guarantee a low bit error rate for all peripheral clock/baud rate combinations.

If an external clock is used, the Pin Function Select and port pins must be initialized first. The following is an example initialization for RX111 channel 1:

```
    MPC.P17PFS.BYTE = 0x0A;     // Pin Func Select P17 SCK1
    PORT1.PDR.BIT.B7 = 0;       // set SCK pin direction to input (dflt)
    PORT1.PMR.BIT.B7 = 1;       // set SCK pin mode to peripheral
```

The callback function used by this driver should have a single argument. This is a pointer to a structure which is cast to a void pointer (provides consistency with other FIT module callback functions). The structure is as follows:

```
typedef struct st_sci_cb_args   // callback arguments
{
    sci_hdl_t       hdl;
    sci_cb_evt_t    event;
    uint8_t         byte;   // byte read when error occurred (unused for TEI)
} sci_cb_args_t;
```

The “hdl” argument is the handle for the channel. The possible events passed in are in the following enumeration:

```
typedef enum e_sci_cb_evt   // callback function events
{
    SCI_EVT_TEI,            // TEI interrupt occurred; transmitter is idle
    SCI_EVT_RX_CHAR,        // received a character; already placed in queue
    SCI_EVT_RXBUF_OVFL,     // rx queue is full; can't save character in queue
    SCI_EVT_OVFL_ERR,       // receiver hardware overrun error occurred
    SCI_EVT_FRAMING_ERR,    // receiver hardware framing error occurred
    SCI_EVT_PARITY_ERR      // receiver hardware parity error occurred
} sci_cb_evt_t;
```

All events except for SCI_EVT_TEI include a data byte from the receiver. An example template for a callback function is provided here:

```
void MyCallback(void *pArgs)
{
sci_cb_args_t    *args;


    args = (sci_cb_args_t *)pArgs;

    if (args->event == SCI_EVT_RX_CHAR)
    {
        // from RXI interrupt; character placed in queue is in args->byte
        nop();
    }
#if SCI_TEI_INCLUDED
    else if (args->event == SCI_EVT_TEI)
    {
        // from TEI interrupt; transmitter is idle
        // possibly disable external transceiver here
        nop();
    }
#endif
    else if (args->event == SCI_EVT_RXBUF_OVFL)
    {
        // from RXI interrupt; receive queue is full
        // unsaved char is in args->byte
        // will need to increase buffer size or reduce baud rate
        nop();
    }
    else if (args->event == SCI_EVT_OVFL_ERR)
    {
        // from ERI interrupt; receiver overflow error occurred
        // error char is in args->byte
        // error condition is cleared in ERI routine
        nop();
    }
    else if (args->event == SCI_EVT_FRAMING_ERR)
    {
        // from ERI interrupt; receiver framing error occurred
        // error char is in args->byte; if = 0, received BREAK condition
        // error condition is cleared in ERI routine
        nop();
    }
    else if (args->event == SCI_EVT_PARITY_ERR)
    {
        // from ERI interrupt; receiver parity error occurred
        // error char is in args->byte
        // error condition is cleared in ERI routine
```

```
        nop();
    }
}
```

3.4 R_SCI_Close()

This function removes power to the SCI channel and disables the associated interrupts.

Format
sci_err_t R_SCI_Close(sci_hdl_t const hdl);

Parameters
hdl
Handle for channel

Return Values
SCI_SUCCESS: *Successful; channel closed*
SCI_ERR_NULL_PTR: *hdl is NULL*

Properties
Prototyped in file “r_sci_async_if.h”

Description
Disables the SCI channel designated by the handle. Does not free any resources but saves power and allows the corresponding channel to be re-opened later, potentially with a different configuration.

Reentrant
Function is re-entrant for different channels.

Example
```
sci_hdl_t   Console;
  :
err = R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, (void *)&config, MyCallback, &Console);
  :
err = R_SCI_Close(Console);
```

Special Notes:
This function will abort any transmission or reception that may be in progress.

3.5 R_SCI_Send()

This function buffers data and initiates transmit if transmitter is not in use.

Format

```
sci_err_t   R_SCI_Send(sci_hdl_t const    hdl,
                       uint8_t            *p_src,
                       uint16_t const     length);
```

Parameters

hdl
Handle for channel
p_src
Pointer to data to transmit
length
Number of bytes to send

Return Values

SCI_SUCCESS:	*Requested number of bytes was sent/loaded into queue*
SCI_ERR_NULL_PTR:	*hdl value is NULL*
SCI_ERR_BAD_MODE:	*Channel mode not currently supported*
SCI_ERR_INSUFFICIENT_SPACE:	*Insufficient space in queue to load all data*

Properties

Prototyped in file “r_sci_async_if.h”

Description

Places data into transmit queue for sending on an SCI channel referenced by the handle. Transmit starts immediately if another transmission is not already in progress. All transmissions are handled at the interrupt level.

Reentrant

Function is re-entrant for different channels.

Example

```
#define STRING1     “Test String”
sci_hdl_t Console;
sci_err_t err;
  :
err = R_SCI_Send(Console, STRING1, sizeof(STRING1));
```

Special Notes:

None.

3.6 R_SCI_Receive()

This function fetches data from the receive queue filled by RXI interrupts.

Format

```
sci_err_t   R_SCI_Receive(sci_hdl_t const   hdl,
                          uint8_t           *p_dst,
                          uint16_t const    length);
```

Parameters

hdl
Handle for channel
p_dst
Pointer to buffer to load data into
length
Number of bytes to read

Return Values

SCI_SUCCESS:	*Requested number of bytes were loaded into pDst*
SCI_ERR_NULL_PTR:	*hdl value is NULL*
SCI_ERR_BAD_MODE:	*Channel mode not currently supported*
SCI_ERR_INSUFFICIENT_DATA:	*Insufficient data in receive queue to fetch all data*

Properties

Prototyped in file “r_sci_async_if.h”

Description

Gets data received on an SCI channel referenced by the handle from its receive queue. Function does not block if the requested number of bytes is not available. If any errors occurred during reception by hardware, they are handled by the callback function specified in R_SCI_Open() and no corresponding error code is provided here.

Reentrant

Function is re-entrant for different channels.

Example

```
    sci_hdl_t Console;
    sci_err_t err;
    uint8_t   byte;

    // do initialization here

    /* echo characters */
    while (1)
    {
        while (R_SCI_Receive(Console, &byte, 1) != SCI_SUCCESS)
            ;

        R_SCI_Send(Console, &byte, 1);
    }
```

Special Notes:

See the “Special Notes” section for R_SCI_Open() for hardware receiver error reporting with a callback function.

3.7 R_SCI_Control()

This function handles special hardware and software operations for the SCI channel.

Format

```
sci_err_t  R_SCI_Control(sci_hdl_t const     hdl,
                         sci_cmd_t const     cmd,
                         void                *p_args);
```

Parameters

hdl
Handle for channel
cmd
Command to run (see enumeration below)
p_args
Pointer to argument(s) specific to command, casted to void *

The valid *cmd* values are as follows:

```
typedef enum e_sci_cmd          // SCI Control() commands
{
    SCI_CMD_EN_NOISE_CANCEL,    // enable noise cancellation
    SCI_CMD_EN_CTS_IN,          // enable cts input (default rts output)
    SCI_CMD_EN_TEI,             // enable tei interrupts
    SCI_CMD_OUTPUT_BAUD_CLK,    // output baud clock on the SCK pin
    SCI_CMD_START_BIT_EDGE,     // detect start bit as falling edge of RXDn pin
                                // (default detect as low level on RXDn pin)
    SCI_CMD_GENERATE_BREAK,     // generate a break condition and flush queue
    SCI_CMD_TX_Q_FLUSH,         // flush transmit queue
    SCI_CMD_RX_Q_FLUSH,         // flush receive queue
    // the following use p_args
    SCI_CMD_TX_Q_BYTES_FREE,    // get count of unused transmit queue bytes
    SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, // get num bytes ready for reading
    SCI_CMD_CHANGE_BAUD         // change baud rate
} sci_cmd_t;
```

None of the commands require arguments except for SCI_CMD_CHANGE_BAUD, SCI_CMD_TX_Q_BYTES_FREE, and SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ. The argument structure for the change-baud command is as follows:

```
typedef struct st_sci_baud
{
    uint32_t    pclk;           // peripheral clock speed; ie 24000000 (24 MHz)
    uint32_t    rate;           // ie 9600, 19200, 115200 (valid for internal clock)
} sci_baud_t;
```

The other two commands simply require a pointer to a uint16_t variable to load the byte count into (see example).

Return Values

SCI_SUCCESS:	*Successful; channel initialized*
SCI_ERR_NULL_PTR:	*hdl or p_args pointer is NULL*
SCI_ERR_BAD_MODE:	*Channel mode not currently supported*
SCI_ERR_INVALID_ARG:	*The cmd or p_args contains an invalid value.*

Properties

Prototyped in file “r_sci_async_if.h”

Description

This function is used for configuring “non-standard” UART hardware features as well as performing driver software maintenance functions.

By default, the SCI hardware outputs a low level on the RTSn#CTSn# pin when the receiver is available for receiving data. Here the pin functions as an RTS output signal. By issuing an SCI_CMD_EN_CTS_IN, the pin accepts CTS input signals. In this case, when the RTSn#CTSn# pin is high, the transmitter is disabled until the line transitions low again. If the transmitter is in process of sending a byte when the line goes high, it completes transmission of the byte before halting.

Reentrant
Function is re-entrant for different channels.

Example
```
sci_hdl_t   Console;
sci_baud_t  baud;
sci_err_t   err;
uint16_t    cnt;
  :
R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, (void *)&config, MyCallback, &Console);
R_SCI_Control(Console, SCI_CMD_EN_NOISE_CANCEL, NULL);
R_SCI_Control(Console, SCI_CMD_EN_TEI, NULL);
  :
/* get amount of space left in transmit queue */
R_SCI_Control(Console, SCI_CMD_TX_Q_BYTES_FREE, (void *)&cnt);
  :
/* get number of bytes sitting in receive queue waiting to be read */
R_SCI_Control(Console, SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, (void *)&cnt);
  :
/* reset baud rate due to low power mode clock switching */
baud.pclk = 8000000;      // 8MHz
baud.rate = 19200;
R_SCI_Control(Console, SCI_CMD_CHANGE_BAUD, (void *)&baud);
```

Special Notes:
If an internal clock is used, there should be 1 bit time delay after issuing an SCI_CMD_CHANGE_BAUD command (or more specifically, after setting the BRR register within this function) and performing communications on the channel to allow time for the clock to settle. The driver uses an algorithm for calculating the optimum values for BRR, SEMR.ABCS, and SMR.CKS. This however does not guarantee a low bit error rate for all peripheral clock/baud rate combinations.

If the command SCI_CMD_EN_CTS_IN is to be used, the Pin Function Select and port pins must be configured first. The following is an example initialization for RX111 channel 1:

```
MPC.P14PFS.BYTE = 0x0B;     // Pin Func Select P14 CTS
PORT1.PDR.BIT.B4 = 0;       // set CTS/RTS pin direction to input (dflt)
PORT1.PMR.BIT.B4 = 1;       // set CTS/RTS pin mode to peripheral
```

If the command SCI_CMD_OUTPUT_BAUD_CLK is to be used, the Pin Function Select and port pins must be configured first. The following is an example initialization for RX111 channel 1:

```
MPC.P17PFS.BYTE = 0x0A;     // Pin Func Select P17 SCK1
PORT1.PDR.BIT.B7 = 1;       // set SCK pin direction to output
PORT1.PMR.BIT.B7 = 1;       // set SCK pin mode to peripheral
```

3.8 R_SCI_GetVersion()

This function returns the driver version number at runtime.

Format
uint32_t R_SCI_GetVersion(void)

Parameters
None

Return Values
Version number.

Properties
Prototyped in file “r_sci_async_if.h”

Description
Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number.

Reentrant
Yes

Example
```
uint32_t   version;
  :
version = R_SCI_GetVersion();
```

Special Notes:
This function is inlined using the “#pragma inline” directive

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
2.00	29-May-13	—	Initial Release.

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.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of 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. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may 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.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

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