APPLICATION NOTE

# Smart Analog IC101

## API Specification

R21AN0015EJ0100
Rev.1.00
Feb 01, 2015

## Introduction

This application note describes the specifications related to the Application Program Interface (referred to API herein) used to control Smart Analog IC101 (RAA730101).

## Contents

# 1. Specifications

## 1.1 Overview

This specification describes API functions that implement asynchronous communication control using UART with the serial array unit (SAU) in Renesas MCU's or clock-synchronous communication control using the 3-wire serial I/O mode (CSI/SPI) to control Smart Analog IC 101 (RAA730101, referred to herein as SAIC101). These API functions chiefly control the SAIC101's registers, A/D converter, and flash memory.

## 1.2 Operation Confirmation Conditions

The source codes used in these specifications have been confirmed under the following conditions.

**Table 1.1 Conditions for Confirming Operations**

| Item | Description |
|---|---|
| Evaluation boards | ・Renesas Starter Kit for RL78/L13 [R0K5010WMS900BE]<br>- Renesas Starter Kit for RL78/L13 CPU board<br>・Smart Analog IC RSK Option Evaluation Board [TSA-OP-IC101] |
| Target device | R5F10WMGAFB (RL78/L13) |
| Operating frequency | 24MHz |
| Operating voltage | 5.0V |
| Integrated Development Environment (CubeSuite+) | V2.02.00 [21 Feb 2014] |
| C Compiler (CubeSuite+) | CA78K0R<br>V4.02.00.03 [16 Jan 2014] |
| RL78/L13 Code Library (CubeSuite+) | V1.02.01.02 [11 Jun 2014] Note 1 |
| Integrated Development Environment (e2studio) | V3.0.0.22 |
| C Compiler (e2studio) | GNURL78 v14.01 |
| RL78/L13 Code Library (e2studio) | V1.02.00.03 [11 Feb 2014] Note 2 |

Note 1: The CubeSuite+ code library is included in the code generator plug-in. The environment described in this document has been confirmed with CubeSuite+ Code_Generator for RL78_78K V2.04.00.

Note 2: The e2studio code library is included with the e2studio product.

The following variable declarations are used in the API described in this document.

| | |
|---|---|
| int8_t | : signed char |
| uint8_t | : unsigned char |
| int16_t | : signed short |
| uint16_t | : unsigned short |
| int32_t | : signed long |
| uint32_t | : unsigned long |
| MD_STATUS | : unsigned short |

Note: This API includes the header file [r_cg_macrodriver.h] generated by the CubeSuite+ code generator.

## 1.3 Software Configuration

Figure 1.1 shows the software configuration of the Renesas MCU and SAIC101.

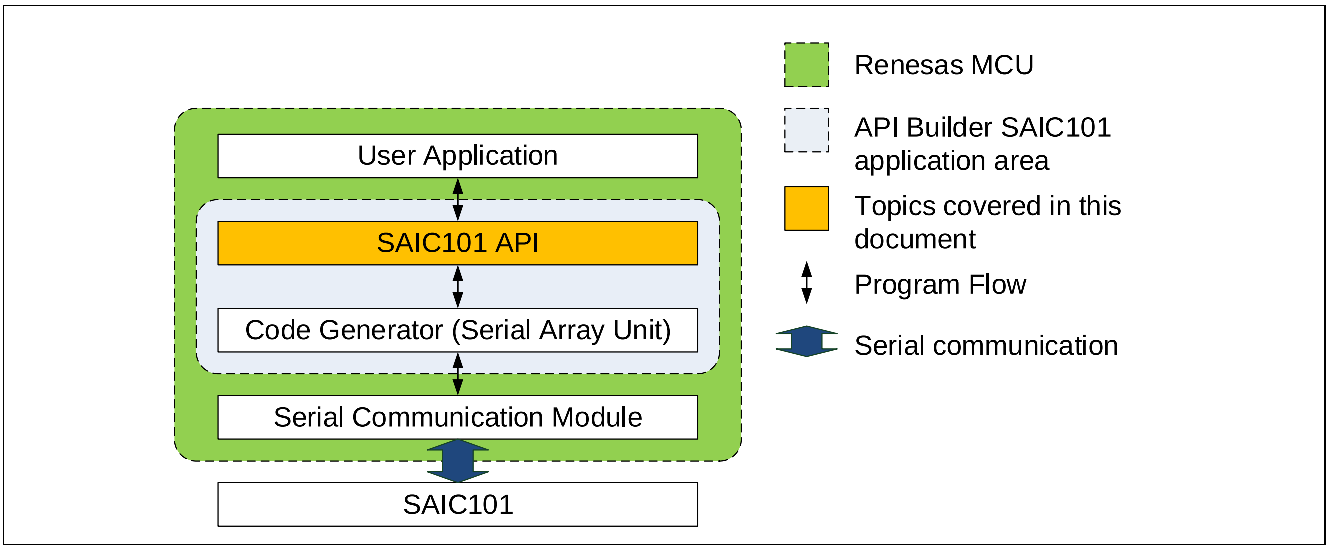

User Application
SAIC101 API
Code Generator (Serial Array Unit)
Serial Communication Module
SAIC101
Renesas MCU
API Builder SAIC101 application area
Topics covered in this document
Program Flow
Serial communication

**Figure 1.1 Software Configuration**

# 2. Cautions for API Usage

This section describes limitations and cautions related to using this API to control SAIC101. Please carefully read all the following details before using the API.

## 2.1 [UART/SPI] Cautions for Communications

### 2.1.1 Writing global variables, definitions in API User Definition Files

CPU and peripheral clock frequency (24 MHz), and Serial Array Unit (UART1) and other definitions are set as one array in the API User Definition File. When not utilizing the API Builder SAIC101 coding assistance tool, some global variables and definitions must be manually rewritten to adjust the API to the user’s development environment. See the **Smart Analog IC101 Tutorial for Sample Code Introduction and API Builder SAIC101 (RL78/L13)** (R21AN0012EJ) for details.

### 2.1.2 Number of Flash Memory Rewrites

Do not exceed the maximum number of writes (100 times) when rewriting the flash memory in a loop in the user program. For details, see the SAIC101 Data Sheet entitled, **RAA730101: 16-bit ΔΣ A/D converter IC with programmable gain instrumentation amplifier** (R02DS0014EJ).

### 2.1.3 Caution for when SBIAS is stopped

When the SENSPD bit of the power/mode control register (CHIPCNT) is set to 1, sensor power supply (SBIAS) stops operating. Note that A/D also stops when SBIAS is not operating. For details, see the SAIC101 Data Sheet entitled, **RAA730101: 16-bit ΔΣ A/D converter IC with programmable gain instrumentation amplifier** (R02DS0014EJ).

### 2.1.4 Caution for rewriting register shadow area

The register shadow area in the flash memory (address 00H to 1FH) must be written carefully because the register value is related to all operations, including communications immediately after power-on. Pay particular attention when writing to address 01H (power/mode control register (CHIPCNT)) and 1FH (startup sequence/communication control register (STARTUP)). The unintentional setting of an incorrect value may prevent normal communications. For details, see the SAIC101 Data Sheet entitled, **RAA730101: 16-bit ΔΣ A/D converter IC with programmable gain instrumentation amplifier** (R02DS0014EJ).

### 2.1.5 Caution for API function internal wait times

All API functions use software timers for processing specified wait times. Note that, in consideration of software timer accuracy, the API functions are set to a period longer than that of the expected wait time. When more accurate wait time is required, measure the time in your own development environment and modify the optimum value for the number of NOPs.

## 2.2 [UART] Cautions for Communications

### 2.2.1 Caution for state of MOSI_RX pin

Communications are disabled when a low level signal of character length x 3 or higher is input to the MOSI_RX pin in the no parity state. For details, see the SAIC101 Data Sheet entitled, **RAA730101: 16-bit ΔΣ A/D converter IC with programmable gain instrumentation amplifier** (R02DS0014EJ).

### 2.2.2 Caution for when AREG operation is stopped

In power supply configuration 1 and 2, AREG operation is stopped when the AREGPD bit of the power/mode control register (CHIPCNT) is set to 1. Note that transmission/reception operations are disabled when AREG operation is stopped. If this bit is overwritten unintentionaly, the SAIC101 must be reset. For details, see the SAIC101 Data Sheet entitled, **RAA730101: 16-bit ΔΣ A/D converter IC with programmable gain instrumentation amplifier** (R02DS0014EJ).

### 2.2.3 Startup sequence data transfer

Normally, when the CPSOR or SDCOR bit of the STARTUP register is set to 1, all 256 bytes of data in the flashmemory are transmitted to the microcontroller immediately after a power-on reset. However, this API does not support this function.

## 2.3 [SPI] Caution for Communications

### 2.3.1 Caution for SPI mode

Although SAIC101 operates in SPI modes 0 and 3, SPI mode 3 operation is only supported by SAIC300, SAIC301, SAIC500, SAIC501, and SAIC502. Make sure the SPI mode is set correctly when connecting to SAIC series other than SAIC101 in the same SPI channel. For details, see the corresponding SAIC series' Data Sheet.

### 2.3.2 Caution for MCU settings and SPI control

The combination of the MCU's CPU and peripheral clock frequency and the baud rate value and communication control method[Note1] for SPI communication may not result in the acquisition of a normal A/D-converted value. To acquire the normal A/D-converted value, make sure the total time for reading the STATUS register and polling is within one sampling period of the over-sampling rate.

When the value of input multiplexer x (x = 1 to 5) A/D conversion setting register 3 is 2 or higher, make sure the CPU,and peripheral clock frequency and SPI communication baud rate are set appropriately.

Note 1: Control using INT pin interrupt or polling control using STATUS register confirmation

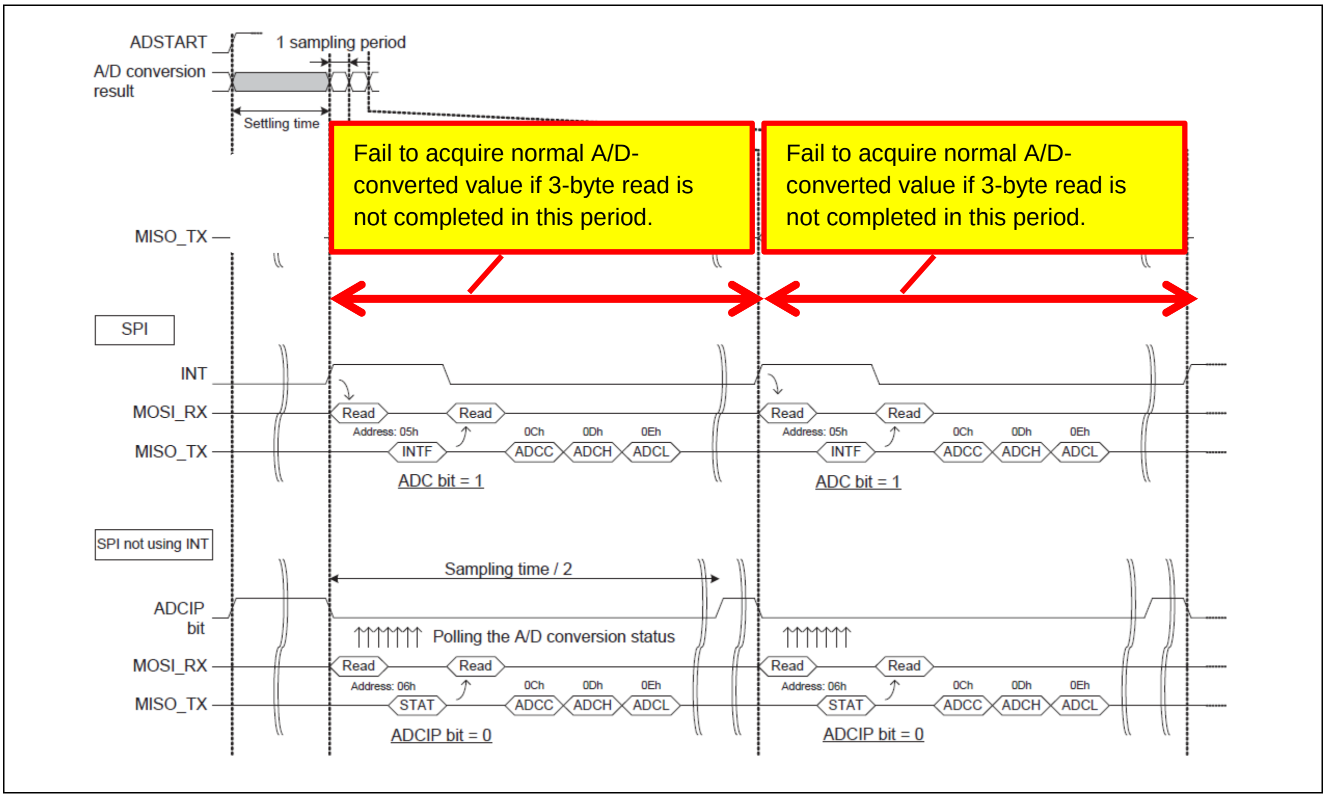


**Figure 2.1 Relationship between sampling period and 3-byte read time**

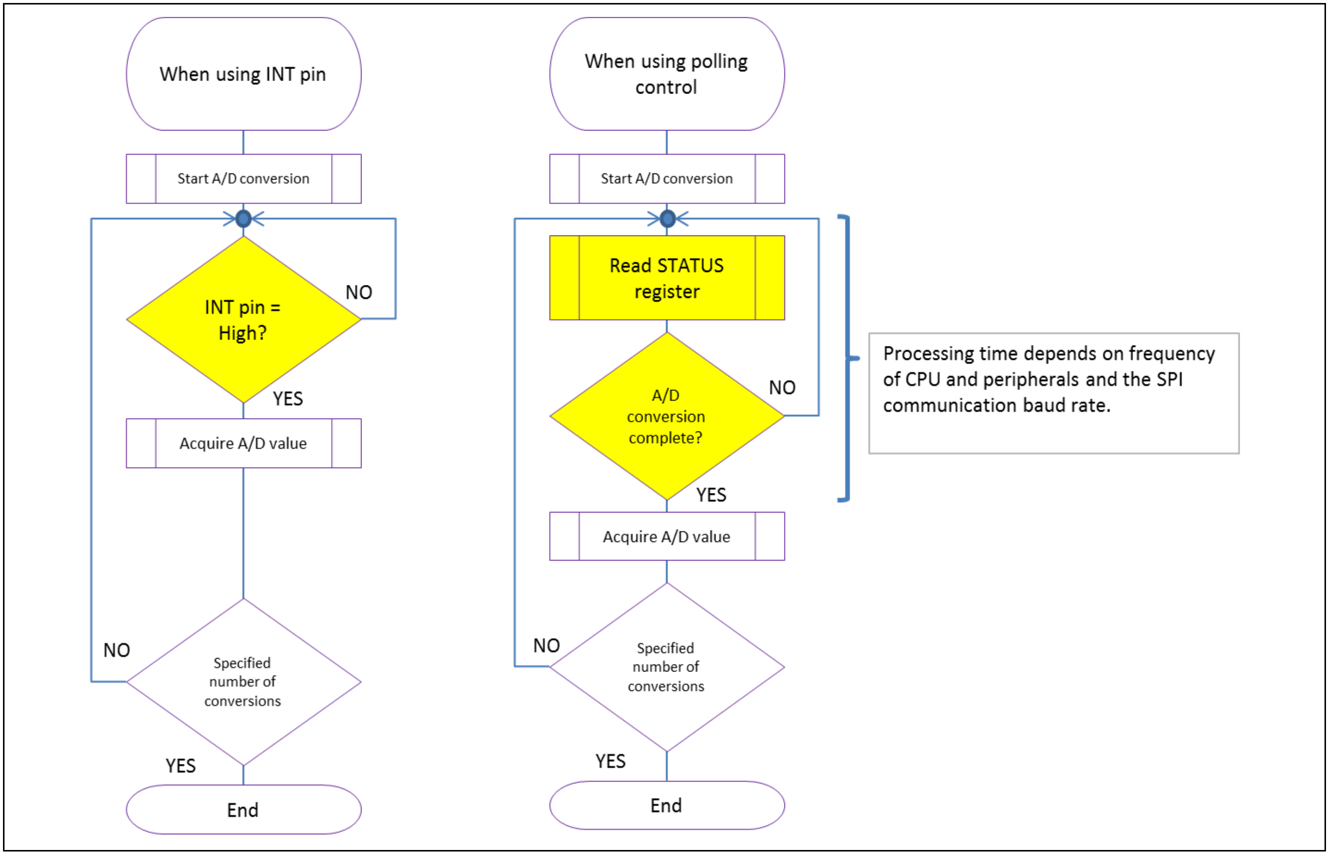

When using INT pin
Start A/D conversion
INT pin = High?
NO
YES
Acquire A/D value
Specified number of conversions
NO
YES
End
When using polling control
Start A/D conversion
Read STATUS register
A/D conversion complete?
NO
YES
Acquire A/D value
Specified number of conversions
NO
YES
End
Processing time depends on frequency of CPU and peripherals and the SPI communication baud rate.

**Figure 2.2 Comparison of INT pin and polling control methods**

# 3. API Functions

The sample code offers API functions related to communication, flash memory, A/D converter, and power supply operations for each type of UART/SPI control, as described below.

For details of the sample code, see SAIC101 Application Notes & Sample Code entitled, Smart Analog IC101: Useful Examples of SAIC101 Sample Code (R21AN0014EJ).

## 3.1 UART Control

### 3.1.1 Communications

Table 3-1 shows a list of API functions related to UART control communications.

**Table 3-1 API functions for UART-control communications**

| Function Name | Description |
|---|---|
| R_SAIC_UART_Init | Smart Analog initialization function |
| R_SAIC_UART_Reset | Smart Analog RESET function |
| R_SAIC_UART_Read | Read register bytes function for UART control |
| R_SAIC_UART_Write | Write register bytes function for UART control |
| R_SAIC_UART_WriteVerify | Write-verify register bytes function for UART control |
| R_SAIC_UART_Negotiation | Communication setting negotiation function for UART control: Automatically confirms communication settings with SAIC101, sets 250kbps/odd parity |
| R_SAIC_UART_SAIC101[Note] | SAIC 101 command function for UART control<br>Enables the following functions:<br>Burst read register<br>Burst write register |

Note R_SAIC_UART_SAIC101 includes both register access and flash memory access functions. See section 3.1.2 Flash memory for details on the flash memory access functions

### 3.1.2 Flash memory

Table 3-2 shows a list of API functions related to the UART-control flash memory.

**Table 3-2 API functions related to UART-control flash memory**

| Function Name | Description |
|---|---|
| R_SAIC_UART_SAIC101[Note] | SAIC101 command functions for UART control<br>Enables the following functions:<br>Read flash memory<br>Write flash memory (addresses 01H and 1FH are write-disabled)<br>Erase all flash memory<br>Copy flash memory shadow area register<br>Copy system settings to buffer |
| R_SAIC_UART_FLASH_WRITE_01H | Write to flash memory address 01H function for UART control.<br>Address 01H implements power supply settings. |
| R_SAIC_UART_FLASH_WRITE_1FH | Write to flash memory address 1FH function for UART control.<br>Address 1F implements startup operations settings. |

Note Note: R_SAIC_UART_SAIC101 includes both register access and flash memory accessfunctions. See section 3.1.1 Communications for details on the register access functions.

### 3.1.3 A/D Converter

Table 3-3 shows a list of API functions related to A/D converter operations for UART control.

**Table 3-3 API functions related to UART-control A/D converter operations**

| Function Name | Description |
|---|---|
| R_SAIC_UART_ADC_Start | A/D converter start process function for UART control |
| R_SAIC_UART_ADC_Stop | A/D converter stop process function for UART control |
| R_SAIC_UART_ADC_GetResult | A/D-converted value acquisition function for UART control. Acquire data from multiple channels, multiple times for each channel. |
| R_SAIC_UART_ADC_GetResult_1Shot | A/D-converted value acquisition function for UART control. Acquire data from a single channel in 1Shot mode. |
| R_SAIC_UART_ADC_GetReceive | A/D-converted value received data acquisition function for UART control. |
| R_SAIC_UART_ADC_InitRegSet | A/D converter register initial setup function for UART control |

### 3.1.4 Power supply

Table 3-4 shows a list of API functions related to power supply for UART control.

**Table 3-4 API functions related to UART-control power supply**

| Function Name | Description |
|---|---|
| R_SAIC_UART_AregOn | AREG ON setting function for UART control |
| R_SAIC_UART_AregOff | AREG OFF setting function for UART control |
| R_SAIC_UART_SbiasRegSet | SBIAS register setting function for UART control |
| R_SAIC_UART_SbiasRegGet | SBIAS register acquisition function for UART control |
| R_SAIC_UART_SleepModeOn | Sleep mode ON setting function for UART control |
| R_SAIC_UART_SleepModeOff | Sleep mode OFF setting function for UART control |

## 3.2 SPI control

### 3.2.1 Communications

Table 3-5 shows a list of API functions related to SPI-control communications.

**Table 3-5 API functions for SPI-control communications**

| Function Name | Description |
|---|---|
| R_SAIC_SPI_Init | Smart Analog initialization function |
| R_SAIC_SPI_Reset | Smart Analog reset function |
| R_SAIC_SPI_Read | Read register bytes function for SPI control |
| R_SAIC_SPI_Write | Write register bytes function for SPI control |
| R_SAIC_SPI_WriteVerify | Write-verify register bytes function for SPI control |
| R_SAIC_SPI_ReadBit | Read register bits function for SPI control |
| R_SAIC_SPI_WriteBit | Write register bits function for SPI control |
| R_SAIC_SPI_WriteVerifyBit | Write-verify register bits function for SPI control |
| R_SAIC_SPI_CSEnable | CS enable function for SPI control |
| R_SAIC_SPI_CSCheck | CS check function for SPI control |
| R_SAIC_SPI_CSDisable | CS disable function for SPI control |
| R_SAIC_SPI_SAIC101[Note] | SAIC 101 command function for SPI control<br>Enables the following functions:<br>Burst read register<br>Burst write register |

Note R_SAIC_UART_SAIC101 includes both register access and flash memory access functions. See section 3.2.2 Flash Memory for details on the flash memory access functions.

Remark SPI communication API functions, other than R_SAIC_SPI_SAIC101, can also be used for Smart Analog IC300, IC301, IC500, IC501, and IC502.

### 3.2.2 Flash Memory

Table 3-6 shows a list of API functions for SPI-control flash memory.

**Table 3-6 API functions related to SPI-control flash memory**

| Function Name | Description |
|---|---|
| R_SAIC_SPI_SAIC101[Note] | SAIC101 command function for SPI control<br>Enables the following functions:<br>Read flash memory<br>Write to flash memory (addresses 01H and 1FH are write-disabled)<br>Erase all flash memory<br>Copy flash memory shadow area registerCopy system setting to buffer |
| R_SAIC_SPI_FLASH_WRITE_01H | Write to flash memory address 01H function for SPI control<br>Address 01H implements power supply settings. |
| R_SAIC_SPI_FLASH_WRITE_1FH | Write to flash memory address 1FH function for SPI control<br>Address 1FH implements startup operation settings. |

Note R_SAIC_UART_SAIC101 includes both register access and flash memory access functions. See section 3.2.1 Communications for details on the register access functions.

### 3.2.3 A/D converter

Table 3-7 shows a list of API functions related to A/D/ converter operations for SPI control.

**Table 3-7 API functions related to SPI-control A/D converter operations**

| Function Name | Description |
|---|---|
| R_SAIC_SPI_ADC_Start | A/D conversion start process function for SPI control |
| R_SAIC_SPI_ADC_Stop | A/D conversion stop process function for SPI control |
| R_SAIC_SPI_ADC_GetResult | A/D-converted value acquisition function for SPI control. Acquire data from multiple channels, multiple times for each channel. |
| R_SAIC_SPI_ADC_GetResult_1Shot | A/D-converted value acquisition function for SPI control. Acquire data from a single channel in 1Shot mode. |
| R_SAIC_SPI_ADC_InitRegSet | A/D converter register initial setup function for SPI control |

### 3.2.4 Power supply

Table 3-8 shows a list of API functions related to power supply for SPI control.

**Table 3-8 API functions related to SPI-control power supply**

| Function Name | Description |
|---|---|
| R_SAIC_SPI_AregOn | AREG ON setting function for SPI control |
| R_SAIC_SPI_AregOff | AREG OFF setting function for SPI control |
| R_SAIC_SPI_SbiasRegSet | SBIAS register setting function for SPI control |
| R_SAIC_SPI_SbiasRegGet | SBIAS register acquisition function for SPI control |
| R_SAIC_SPI_SleepModeOn | Sleep mode ON setting function for SPI control |
| R_SAIC_SPI_SleepModeOff | Sleep mode OFF setting function for SPI control |

# 4. Common API Defintions

This section provides the common definitions used for all API functions.

## 4.1 Common API Function Return Values

Most functions in this API return status values in a common form. In the user's application, the user must judge the return value; if normal, the corresponding process is executed, if an error is returned, a correction process is executed.

**Table 4-1 [UART/SPI] Common API function return values**

| Part Name | Macro Name | Constant Value | Description |
|---|---|---|---|
| saic_status_t<br>(uint8_t) | D_SAIC_OK | 00H | Successful completion |
| | D_SAIC_ERR_PARAM | 01H | Parameter error |
| | D_SAIC_ERR_COM | 02H | Communication error (overrun error, timeout error) |
| | D_SAIC_ERR_VERIFY | 03H | Verify error |

## 4.2 Macro Declarations for User Environment-dependent Settings

This API uses macro declarations to define areas dependent on the user environment or usage conditions. Please modify each definition as needed to meet the user's development environment.

### 4.2.1 UART communications

(a) **r_sa_uart_control_register.h**

**Table 4-2 [UART] Macro declarations 1/2**

| Macro Declaration | Default Setting Value | Input Range | Description |
|---|---|---|---|
| D_DEADLOCK_CNT | 11000000L | uint32_t[Note 1] | Number of loops to judge deadlock during communication wait |

Note 1: Specify value larger than 0.

**Table 4-3 [UART] Macro declarations 2/2**

| Macro Declaration | Description |
|---|---|
| D_SAIC_FLASH_API_VALID | Enables FLASH API function. Disable when not using FLASH API to reduce RAM capacity. |
| D_UART_NEGOTIATION_250KBPS_PARITY_ODD | Enables negotiation for baudrate=250000bps, Parity=odd. Disable when not used. |
| D_UART_NEGOTIATION_250KBPS_PARITY_EVEN | Enables negotiation for baudrate=250000bps, Parity=even. Disable when not used. |
| D_UART_NEGOTIATION_250KBPS_PARITY_NONE | Enables negotiation for baudrate=250000bps, Parity=none. Disable when not used. |
| D_UART_NEGOTIATION_4800BPS_PARITY_ODD | Enables negotiation for baudrate=4800bps, Parity=odd. Disable when not used. |
| D_UART_NEGOTIATION_4800BPS_PARITY_EVEN | Enables negotiation for baudrate=4800bps, Parity=even. Disable when not used. |
| D_UART_NEGOTIATION_4800BPS_PARITY_NONE | Enables negotiation for baudrate=4800bps, Parity=none. Disable when not used. |

(b) **r_sa_uart_control_register_user.c**

**Table 4-4 [UART] Macro declarations**

| Macro Declaration | Default Setting Value | Input Range | Description |
|---|---|---|---|
| D_CPU_CLK_MHZ | 24.0F | float | Definition of CPU and peripheral clock frequency [Note 1 Note 2]<br>Use to calculate general software wait used in API. Unit:MHz |
| D_WAIT_PON_RST_TIME_MS | 4.00F | float | Definition of power-on RESET wait. [Note1]<br>Use to determine wait when power-on RESET is specified in the RESET function. Unit:ms |

Note 1: Specifiy value larger than 0.

Note 2: Specify setting value for the MCU CPU clock.

### 4.2.2 SPI communications

(a) **r_sa_spi_control_register.h**

**Table 4-5 [SPI] Macro declarations 1/2**

| Macro Declaration | Default Setting Value | Input Range | Description |
|---|---|---|---|
| D_DEADLOCK_CNT | 11000000L | uint32_t | Number of loops to judge deadlock during communication wait [Note1] |
| D_SPI_OPERATION | D_SPI_USE_INTERRUPT | D_SPI_USE_INTERRUPT<br>Or<br>D_SPI_REGISTER_POLLING | Definition for selecting use of communication module interrupt or no use (use polling) |

Note 1: Specify value larger than 0.

**Table 4-6 [SPI] Macro declarations 2/2**

| Macro Declaration | Description |
|---|---|
| D_SAIC_FLASH_API_VALID | Enables FLASH API function. Disable when not using FLASH API to reduce RAM capacity. |

(b) **r_sa_spi_control_register_user.c**

**Table 4-7 [SPI] Macro declarations**

| Macro Declaration | Default Setting Value | Input Range | Description |
|---|---|---|---|
| D_CPU_CLK_MHZ | 24.0F | float | Definition of CPU and peripheral clock frequency [Note1] [Note2]<br>Use to calculate general software wait used in API. Unit:MHz |
| D_WAIT_PON_RST_TIME_MS | 4.00F | float | Definition of power-on RESET wait [Note1]<br>Use to determine wait when power-on RESET is specified in the RESET function. Unit:ms |
| D_WAIT_HARD_RESET_TIME_MS | 0.01F | float | Definition of hard RESET wait [Note1]<br>Use to determine wait when hard RESET is specified in the RESET function. Unit:ms |

Note1: Specify value larger than 0.

Note 2: Specify setting value for the MCU CPU clock.

## 4.3 Macro Declarations

This section describes the macro declarations defined in the API.

(a) **r_sa_uart_control_register.h**

### Table 4-8 [UART] Macro declarations

| Macro Declaration | Value | Description |
|---|---|---|
| D_UART_USE_INTERRUPT | 1U | Definition of communication module interrupt use |
| D_UART_REGISTER_POLLING | 2U | Definition for not using communication module interrupt (use polling) |
| D_UART_OPERATION | D_UART_USE_INTERRUPT | Definition for selecting use/not use (use polling) communication module interrupt |

(b) **r_sa_spi_control_register.h**

### Table 4-9 [SPI] Macro declarations

| Macro Declaration | Value | Description |
|---|---|---|
| D_SPI_USE_INTERRUPT | 1U | Definition of communication module interrupt use |
| D_SPI_REGISTER_POLLING | 2U | Definition for not using communication module interrupt (use polling) |

(c) **r_sa_uart_control_register.c**

### Table 4-10 [UART] Macro declarations

| Macro Declaration | Value | Description |
|---|---|---|
| D_READ_MAX_SIZE | 256U | Definition of maximum receive data size |
| D_COMMAND_LENGTH | 3U | Definition of number of bytes in UART response packet |
| D_BURST_MAX_SIZE | 16U | Definition of SAIC101 burst read/write maximum data size |
| D_REGISTER_MAX_ADDRESS | 1FH | Definition of SAIC101 register maximum address |
| D_FLASH_MAX_ADDRESS | FFH | Definition of SAIC101 flash memory maximum address |
| D_AD_DATA_LENGTH | 3U | Definition of SAIC101 A/D conversion data register size |
| D_FLASH_READ_MAX_SIZE | 256U | Definition of SAIC101 flash memory read maximum data size |
| D_AUTO_ADC_MAX_RECEIVE_SIZE | 10U | Definition of UART auto A/D-converted value reception buffer size |
| D_AUTO_ADC_REPLAY_MAX | 5U | Definition for number of replays |

(d) **r_sa_spi_control_register.c**

### Table 4-11 [SPI] Macro declarations

| Macro Declaration | Value | Description |
|---|---|---|
| D_BURST_MAX_SIZE | 16U | Definition of SAIC101 burst read/write maximum data size |
| D_COMMAND_LENGTH | 2U | Definition of number of command bytes for: burst read register, burst write register, read flash, and write flash commands |
| D_FLASH_READ_MAX_SIZE | 256U | Definition of SAIC101 flash memory read maximum data size |
| D_REGISTER_MAX_ADDRESS | 1FH | Definition of SAIC101 register maximum address |
| D_FLASH_MAX_ADDRESS | FFH | Definition of SAIC101 flash memory maximum address |

## 4.4 Type Declarations

This section describes the unique type declarations (typedefs) defined in the API.

(a) **r_sa_uart_control_register.h / r_sa_spi_control_register.h**

### Table 4-12 [UART/SPI] Type declaration

| Type | Definition | Description |
|---|---|---|
| saic_status_t | uint8_t | Return value type common to all API functions |

(b) **r_sa_spi_control_register.c**

### Table 4-13 [UART/SPI] Type declaration

| Type | Definition | Description |
|---|---|---|
| saic_func_bitRead_t | saic_status_t (* )(uint8_t saic_num, uint8_t *red_bit) | Read register bits function pointer |

## 4.5 Enumerations Requiring User Modification

The enumerations shown in Table 4-14 Table 4-15 are used with the constants described in section 4.9. The enumerations need to be specified by the user according to the constant settings. For details, see section 4.9.

(a) **r_sa_uart_control_register.h**

### Table 4-14 [UART] Enumuration for specifying global variable to store serial module information

| Part name | Default Definition | Description |
|---|---|---|
| e_uart_ch_t | E_UART0 | Define enumerations according to the number of arrays defined in g_uart_serial_data_tbl.<br>Enumeration names are used in g_uart_saic_data_tbl.<br>Any names can be for default definitions as long as they correspond to the following examples: g_uart_serial_data_tbl or g_uart_saic_data_tbl. |
| | E_UART1 | |
| | E_UART2 | |
| | E_UART3 | |
| | E_UART4 | |
| | E_UART5 | |
| | E_UART6 | |
| | E_UART_MAX | Determine maximum value of g_uart_serial_data_tbl[] |

(b) **r_sa_spi_control_register.h**

### Table 4-15 [SPI] Enumuration for specifying global variable to store serial module information

| Part name | Default Definition | Description |
|---|---|---|
| e_csi_ch_t | E_CSI00 | Define enumerations according to the number of arrays defined in g_spi_serial_data_tbl.<br>Enumeration names are used in g_spi_saic_data_tbl.<br>Any names can be for default definitions as long as they correspond to the following examples: g_spi_serial_data_tbl or g_spi_saic_data_tbl. |
| | E_CSI01 | |
| | E_CSI10 | |
| | E_CSI11 | |
| | E_CSI20 | |
| | E_CSI21 | |
| | E_CSI30 | |
| | E_CSI31 | |
| | E_CSI_MAX | Determine maximum value of g_spi_serial_data_tbl[] |

## 4.6 Enumerations

This section describes the enumerations declarations defined in the API.

(a) **r_sa_uart_control_register.h / r_sa_spi_control_register.h**

### Table 4-16 [UART/SPI] Enumeration for specifying SAIC type (part name)

| Part name | Macro Name | Description |
|---|---|---|
| e_saic_type_t | E_SAIC300 | Specifies SAIC300 |
| | E_SAIC301 | Specifies SAIC301 |
| | E_SAIC500 | Specifies SAIC500 |
| | E_SAIC501 | Specifies SAIC501 |
| | E_SAIC502 | Specifies SAIC502 |
| | E_SAIC101 | Specifies SAIC101 |
| | E_IC_TYPE_MAX | Determine maximum value |

### Table 4-17 [UART/SPI] Enumeration for specifying RESET process

| Part name | Macro Name | Description |
|---|---|---|
| e_reset_process_t | E_SAIC_EXTERNAL_RESET | External RESET (resets RESET pin from L→H) |
| | E_SAIC_SPI_INTERNAL_RESET | Internal RESET (transmits RESET/RESET release command in SPI) |
| | E_SAIC_UART_INTERNAL_RESET | Internal RESET (transmits RESET/RESET release command in UART) |
| | E_SAIC_POWERON_RESET | POWER_ON RESET (waits for operations after power is turned on) |
| | E_SAIC_RESET_MAX | Determine maximum value |

### Table 4-18 [UART/SPI] Enumeration for specifying SAIC101 unique commands

| Part name | Macro Name | Description |
|---|---|---|
| e_saic101_func_t | E_REGISTER_READ | Read register |
| | E_REGISTER_WRITE | Write register |
| | E_REGISTER_BURST_READ | Burst read register |
| | E_REGISTER_BURST_WRITE | Burst write register |
| | E_REGISTER_ALL_WRITE_FROM_FLASH | Copy flash shadow area to register |
| | E_BUFFER_REFRESH | Copy flash system setting to buffer |
| | E_FLASH_READ | Read flash memory |
| | E_FLASH_READ_1 | First read of command from flash memory |
| | E_FLASH_READ_2 | Second read of command from flash memory |
| | E_FLASH_WRITE | Write to flash memory |
| | E_FLASH_ALL_ERASE | Erase all of flash memory |
| | E_FLASH_WRITE_VERIFY | Verify-write to flash memory |

### Table 4-19 [UART/SPI] Enumeration for specifying A/D converter input mode

| Part name | Macro Name | Description |
|---|---|---|
| e_adc_mode_t | E_ADC_DIFF | Differential input mode |
| | E_ADC_SINGLE | Single-ended input mode |

### Table 4-20 [UART/SPI] Enumeration for specifying PGIA gain settings

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_adc_gain_t | E_ADC_GAIN_1_1_1 | 00H | GSET1 = x1, GSET2 = x1, Total = x 1 |
| | E_ADC_GAIN_2_1_2 | 04H | GSET1 = x2, GSET2 = x1, Total = x 2 |
| | E_ADC_GAIN_3_1_3 | 08H | GSET1 = x3, GSET2 = x1, Total = x 3 |
| | E_ADC_GAIN_4_1_4 | 0CH | GSET1 = x4, GSET2 = x1, Total = x 4 |
| | E_ADC_GAIN_8_1_8 | 10H | GSET1 = x8, GSET2 = x1, Total = x 8 |
| | E_ADC_GAIN_1_2_2 | 01H | GSET1 = x1, GSET2 = x2, Total = x 2 |
| | E_ADC_GAIN_2_2_4 | 05H | GSET1 = x2, GSET2 = x2, Total = x 4 |
| | E_ADC_GAIN_3_2_6 | 09H | GSET1 = x3, GSET2 = x2, Total = x 6 |
| | E_ADC_GAIN_4_2_8 | 0DH | GSET1 = x4, GSET2 = x2, Total = x 8 |
| | E_ADC_GAIN_8_2_16 | 11H | GSET1 = x8, GSET2 = x2, Total = x16 |
| | E_ADC_GAIN_1_4_4 | 02H | GSET1 = x1, GSET2 = x4, Total = x 4 |
| | E_ADC_GAIN_2_4_8 | 06H | GSET1 = x2, GSET2 = x4, Total = x 8 |
| | E_ADC_GAIN_3_4_12 | 0AH | GSET1 = x3, GSET2 = x4, Total = x12 |
| | E_ADC_GAIN_4_4_16 | 0EH | GSET1 = x4, GSET2 = x4, Total = x16 |
| | E_ADC_GAIN_8_4_32 | 12H | GSET1 = x8, GSET2 = x4, Total = x32 |
| | E_ADC_GAIN_1_8_8 | 03H | GSET1 = x1, GSET2 = x8, Total = x 8 |
| | E_ADC_GAIN_2_8_16 | 07H | GSET1 = x2, GSET2 = x8, Total = x16 |
| | E_ADC_GAIN_3_8_24 | 0BH | GSET1 = x3, GSET2 = x8, Total = x24 |
| | E_ADC_GAIN_4_8_32 | 0FH | GSET1 = x4, GSET2 = x8, Total = x32 |

### Table 4-21 [UART/SPI] Enumeration for specifying over-sampling rate

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_adc_osr_t | E_ADC_OSR_64 | 00H | 15625.000 [sps] |
| | E_ADC_OSR_128 | 01H | 7812.500 [sps] |
| | E_ADC_OSR_256 | 02H | 3906.250 [sps] |
| | E_ADC_OSR_512 | 03H | 1953.125 [sps] |
| | E_ADC_OSR_1024 | 04H | 976.563 [sps] |
| | E_ADC_OSR_2048 | 05H | 488.281 [sps] |
| | E_ADC_OSR_MAX | 06H | Determine maximum value |

### Table 4-22 [UART/SPI] Enumeration for specifying DC offset

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_adc_offset_t | E_ADC_OFFSET_164p06 | 1FH | 164.06/GSET1 [mV] |
| | E_ADC_OFFSET_153p13 | 1EH | 153.13/GSET1 [mV] |
| | E_ADC_OFFSET_142p19 | 1DH | 142.19/GSET1 [mV] |
| | E_ADC_OFFSET_131p25 | 1CH | 131.25/GSET1 [mV] |
| | E_ADC_OFFSET_120p31 | 1BH | 120.31/GSET1 [mV] |
| | E_ADC_OFFSET_109p38 | 1AH | 109.38/GSET1 [mV] |
| | E_ADC_OFFSET_98p44 | 19H | 98.44/GSET1 [mV] |
| | E_ADC_OFFSET_87p50 | 18H | 87.50/GSET1 [mV] |
| | E_ADC_OFFSET_76p56 | 17H | 76.56/GSET1 [mV] |
| | E_ADC_OFFSET_65p63 | 16H | 65.63/GSET1 [mV] |
| | E_ADC_OFFSET_54p69 | 15H | 54.69/GSET1 [mV] |
| | E_ADC_OFFSET_43p75 | 14H | 43.75/GSET1 [mV] |
| | E_ADC_OFFSET_32p81 | 13H | 32.81/GSET1 [mV] |
| | E_ADC_OFFSET_21p88 | 12H | 21.88/GSET1 [mV] |
| | E_ADC_OFFSET_10p94 | 11H | 10.94/GSET1 [mV] |
| | E_ADC_OFFSET_0p00 | 10H | 0.00/GSET1 [mV] |
| | E_ADC_OFFSET_M10p94 | 0FH | -10.94/GSET1 [mV] |
| | E_ADC_OFFSET_M21p88 | 0EH | -21.88/GSET1 [mV] |
| | E_ADC_OFFSET_M32p81 | 0DH | -32.81/GSET1 [mV] |
| | E_ADC_OFFSET_M43p75 | 0CH | -43.75/GSET1 [mV] |
| | E_ADC_OFFSET_M54p69 | 0BH | -54.69/GSET1 [mV] |
| | E_ADC_OFFSET_M65p63 | 0AH | -65.63/GSET1 [mV] |
| | E_ADC_OFFSET_M76p56 | 09H | -76.56/GSET1 [mV] |
| | E_ADC_OFFSET_M87p50 | 08H | -87.50/GSET1 [mV] |
| | E_ADC_OFFSET_M98p44 | 07H | -98.44/GSET1 [mV] |
| | E_ADC_OFFSET_M109p38 | 06H | -109.38/GSET1 [mV] |
| | E_ADC_OFFSET_M120p31 | 05H | -120.31/GSET1 [mV] |
| | E_ADC_OFFSET_M131p25 | 04H | -131.25/GSET1 [mV] |
| | E_ADC_OFFSET_M142p19 | 03H | -142.19/GSET1 [mV] |
| | E_ADC_OFFSET_M153p13 | 02H | -153.13/GSET1 [mV] |
| | E_ADC_OFFSET_M164p06 | 01H | -164.06/GSET1 [mV] |
| | E_ADC_OFFSET_M175p00 | 00H | -175.00/GSET1 [mV] |

### Table 4-23 [UART/SPI] Enumeration for specifying A/D converter channel number

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_adc_ch_t | E_ADC_CH1 | 00H | Ch1 |
| | E_ADC_CH2 | 01H | Ch2 |
| | E_ADC_CH3 | 02H | Ch3 |
| | E_ADC_CH4 | 03H | Ch4 |
| | E_ADC_CH5 | 04H | Ch5 (temperature sensor) |

### Table 4-24 [UART/SPI] Enumeration for setting A/D converter ON/OFF

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_adc_onoff_t | E_ADC_OFF | 01H | Turn A/D converter OFF |
| | E_ADC_ON | 00H | Turn A/D converter ON |

### Table 4-25 [UART / SPI] Enumeration for setting ADSTART bit

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_adc_start_t | E_ADC_STOP | 00H | Stop A/D converter |
| | E_ADC_START | 01H | Start A/D converter |

### Table 4-26 [UART / SPI] Enumeration for specifying SBIAS output power setting

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_sbias_t | E_ADC_SBIAS_0p0 | FFH | OFF (SENSEPD=1) |
| | E_ADC_SBIAS_1p2 | 00H | 1.2V |
| | E_ADC_SBIAS_1p3 | 01H | 1.3V |
| | E_ADC_SBIAS_1p4 | 02H | 1.4V |
| | E_ADC_SBIAS_1p5 | 03H | 1.5V |
| | E_ADC_SBIAS_1p6 | 04H | 1.6V |
| | E_ADC_SBIAS_1p7 | 05H | 1.7V |
| | E_ADC_SBIAS_1p8 | 06H | 1.8V |
| | E_ADC_SBIAS_1p9 | 07H | 1.9V |
| | E_ADC_SBIAS_2p0 | 08H | 2.0V |
| | E_ADC_SBIAS_2p1 | 09H | 2.1V |
| | E_ADC_SBIAS_2p2 | 0AH | 2.2V |
| | E_ADC_SBIAS_MAX | 0BH | Determine maximum value |

(b) **r_sa_uart_control_register.h**

### Table 4-27 [UART] Enumeration for specifying receive packet judgement

| Part name | Macro Name | Description |
|---|---|---|
| get_response_state_t | E_GET_HEADER | Receive header |
| | E_HEADER_DECODE | Decode header |
| | E_TYPE1 | Process TYPE1 response |
| | E_TYPE2 | Process TYPE2 response |
| | E_TYPE3 | Process TYPE3 response |
| | E_END | End |

### Table 4-28 [UART] UART communication settings

| Part name | Macro Name | Description |
|---|---|---|
| e_uart_setting_t | E_UART_4800bps_None | Baud rate=4800 bps, Parity = None |
| | E_UART_4800bps_Odd | Baud rate=4800 bps, Parity = Odd |
| | E_UART_4800bps_Even | Baud rate=4800 bps, Parity = Even |
| | E_UART_250kbps_None | Baud rate=250000 bps, Parity = None |
| | E_UART_250kbps_Odd | Baud rate=250000 bps, Parity = Odd |
| | E_UART_250kbps_Even | Baud rate=250000 bps, Parity = Even |

(c) **r_sa_spi_control_register.c**

**Table 4-29 [C source file definition, SPI] Enumeration for specifying ReadWrite**

| Part name | Macro Name | Value | Description |
|---|---|---|---|
| e_rw_t | E_READ | 00H | For read |
| | E_WRITE | 01H | For write |

## 4.7 Structures

This section describes the structure declarations defined in the API.

### (a) r_sa_uart_control_register.h / r_sa_spi_control_register.h

**Table 4-30 [UART/SPI] Structure for A/D-converted information storage variable**

| | | | |
|---|---|---|---|
| **Structure name** | saic101_adc_t | | |
| **Description** | SAIC101 A/D converted information storage variable structures | | |
| Member variable | **Type** | **Name** | **Description** |
| | e_adc_onoff_t | onoff | Enumeration variable for specifying A/D converter ON/OFF |
| | e_adc_mode_t | input_mode | Enumeration variable for specifying A/D converter input mode |
| | e_adc_offset_t | offset | Enumeration variable for specifying DC offset |
| | e_adc_osr_t | over_sampling_rate | Enumeration variable for specifying over sampling rate |
| | e_adc_gain_t | gain | Enumeration variable for specifying PGIA gain setting |
| | uint8_t | count | A/D conversion count specifying for one AUTOSCAN cycle |

**Table 4-31 [UART / SPI] Structure for byte manipulation functions**

| | | | |
|---|---|---|---|
| **Structure name** | saic_data_t | | |
| **Description** | Structure for byte manipulation functions | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | uint8_t | address | SAIC control register address |
| | uint8_t | data | SAIC control register data |

**Table 4-32 [UART/SPI] Structure for bit manipulation functions**

| | | | |
|---|---|---|---|
| **Structure name** | saic_data_bit_t | | |
| **Description** | Structure for bit manipulation functions | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | uint8_t | address | SAIC control register address |
| | uint8_t | bit_number | SAIC control register bit number (0 to7) |
| | uint8_t | data | SAIC control register bit data (0/1) |

### (b) r_sa_uart_control_register.h

**Table 4-33 [UART] Structure for SAIC information storage**

| | | | |
|---|---|---|---|
| **Structure name** | uart_saic_t | | |
| **Description** | Structure for SAIC information storage variables (SA type, UART channel) | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | e_uart_ch_t | uart_ch | UART ch number |
| | e_saic_type_t | sa_type | SAIC part name |

### Table 4-34 [UART] Structure for serial information storage (when using interrupt)

| | | | |
|---|---|---|---|
| **Structure name** | uart_serial_t | | |
| **Description** | Structure for serial information storage variables (stores pointers for start, stop, transmit, receive and other functions) | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | void (* )(void) | UART_Start | UARTxx_Start function pointer |
| | void (* )(void) | UART_Stop | UARTxx_Stop function pointer |
| | MD_STATUS (*)(uint8_t * const rx_buf, uint16_t rx_num) | UART_Receive | UARTxx_Receive function pointer |
| | MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num) | UART_Send | UARTxx_Send function pointer |
| | uint8_t (*)(uint8_t *packet_data, uint8_t rx_buffer[], uint16_t read_pos) | UART_GetHeader | UARTxx_GetHeader function pointer |
| | uint8_t (*)(uint16_t rx_cnt) | UART_Getdata | UARTxx_Getdata function pointer |
| | void (*)(uint8_t setting) | UART_SettingChange | UARTxx_SettingChange function pointer |

Note: xx indicates the serial channel number used for communication with SAIC.

### Table 4-35 [UART] Structure for RESET information storage

| | | | |
|---|---|---|---|
| **Structure name** | uart_reset_t | | |
| **Description** | SAIC RESET information variables (RESET process, address of port connected to RESET pin and corresponding bit number, number of NOP executions for wait, uart_saic_t variable number | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | e_reset_process_t | process | Specifies SAIC RESET method |
| | uint8_t * | p_reset_addr | Address of port register connected to RESET pin. |
| | uint8_t | reset_bit_num | Bit number of port register connected to RESET pin. |
| | uint32_t | nop_cnt | NOP command count for wait |
| | uint8_t | uart_saic_num | Buffer number of SA storage variables *Use for RESET and other commands in UART communications. Not specified for common RESET pin, etc. |

### (c) r_sa_spi_control_register.h

### Table 4-36 [SPI] Structure for SAIC information storage

| | | | |
|---|---|---|---|
| **Structure name** | spi_saic_t | | |
| **Description** | Structure for SAIC information storage variables (SA type, CSI ch, port address connected to CS port and corresponding bit number, address for port of connected INT pin (if used) and corresponding bit number) | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | e_csi_ch_t | csi_ch | CSI channel number |
| | e_saic_type_t | sa_type | SAIC type (part name) |
| | uint8_t * | p_cs_addr | Address of port register connected to CS pin |
| | uint8_t | cs_bit_num | Bit number of port register connected to CS pin |
| | uint8_t * | p_int_addr | Address of port register connected to INT pin |
| | uint8_t | int_bit_num | Bit number of port register connected to INT pin |

Note: xx indicates the serial channel number used for communication with SAIC.

### Table 4-37 [SPI] Structure for serial information storage (when using interrupt)

| | | | |
|---|---|---|---|
| **Structure name** | spi_serial_t | | |
| **Description** | Structure for serial information storage variables (stores pointers for start, stop, and R/W execution functions) | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | void (* )(void) | CSI_Start | CSIxx_Start function pointer |
| | void (* )(void) | CSI_Stop | CSIxx_Stop function pointer |
| | MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf) | CSI_Send_Receive | CSIxx_Send_Receive function pointer |

Note: xx indicates the serial channel number used for communication with SAIC.

### Table 4-38 [SPI] Structure for serial information storage (when using polling, not interrupt)

| | | | |
|---|---|---|---|
| **Structure name** | spi_serial_t | | |
| **Description** | Structure for serial information storage variables (registers flags used for start, stop, and other processes) | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | uint8_t * | p_trans_reg_addr | Address for transmit data register corresponding to CSI channel |
| | uint8_t * | p_recv_reg_addr | Address for receive data register corresponding to CSI channel |
| | uint8_t * | p_csiif_addr | Address for interrupt request flag register corresponding to CSI channel |
| | uint8_t | csiif_bit_num | Bit number for interrupt request flag register corresponding to CSI channel |
| | uint16_t * | p_ssr_addr | Address for SSR register corresponding to CSI channel |
| | uint16_t * | p_sir_addr | Address for SIR register corresponding to CSI channel |
| | void (* )(void) | CSI_Start | CSIxx_Start function pointer |
| | void (* )(void) | CSI_Stop | CSIxx_Stop function pointer |

### Table 4-39 [SPI] Structure for RESET information storage

| | | | |
|---|---|---|---|
| **Structure name** | spi_reset_t | | |
| **Description** | SAIC RESET information variables (RESET process, address of port connected to RESET pin and corresponding bit number, number of NOP executions for wait, spi_saic_t variable number | | |
| **Member variable** | **Type** | **Name** | **Description** |
| | e_reset_process_t | process | Specifies SAIC RESET method |
| | uint8_t * | p_reset_addr | Address of port register connected to RESET pin |
| | uint8_t | reset_bit_num | Bit number of port register connected to RESET pin. |
| | uint32_t | nop_cnt | NOP command count for wait |
| | uint8_t | spi_saic_num | Buffer number of SA storage variables *Use for RESET and other commands in SPI communications. Not specified for common RESET pin, etc. |

## 4.8 Unions

This section describes the union declarations defined in the API.

(a) **r_sa_uart_control_register.h / r_sa_spi_control_register.h**

### Table 4-40 [UART/SPI] Unions for SAIC101 register information storage variables

| | | | | |
|---|---|---|---|---|
| **Structure Name** | uni_reg_t | | | |
| **Description** | For acquiring or for updating register specific bit value | | | |
| **Member variable** | **Type** | **No. of bits** | **Name** | **Description** |
| | uint8_t | 8 | BYTE | 8-bit access |
| | CHIPCNT_BIT | 1 (lower) | slp | Sleep |
| | | 1 | senspd | SENSPD |
| | | 2 | - | Unused |
| | | 1 | aregpd | AREGPD |
| | | 1 | psthru | PSTHRU |
| | | 2 (upper) | - | Unused |
| | VSBIAS_BIT | 4 | sbias | SBIAS |
| | | 4 | - | Unused |
| | INTFLAG_BIT | 1 | fr | FR |
| | | 1 | fw | FW |
| | | 1 | fae | FAE |
| | | 1 | raw | RAW |
| | | 1 | adc | ADC |
| | | 3 | - | Unused |
| | STATUS_BIT | 1 | - | Unused |
| | | 1 | fwip | FWIP |
| | | 1 | faeip | FAEIP |
| | | 1 | rawip | RAWIP |
| | | 1 | adcip | ADCIP |
| | | 3 | - | Unused |
| | ADCCNT_BIT | 5 | ainxadc | OnOff |
| | | 2 | - | Unused |
| | | 1 | adstart | ADSTART |
| | CHxCNT1_BIT | 5 | offset | offset |
| | | 2 | - | Unused |
| | | 1 | input_mode | AINSEL |
| | CHxCNT2_BIT | 5 | gain | GAIN |
| | | 3 | osr | OSR |
| | CHxCNT3_BIT | 5 | low | Lower 5 bits |
| | | 3 | high | Upper 3 bits |
| | STARTUP_BIT | 1 | cpsor | CPSOR |
| | | 1 | sdcor | SDCOR |
| | | 2 | - | Unused |
| | | 1 | tglsm | TGLSM |
| | | 3 | - | Unused |

### Table 4-41 [UART/SPI] Union for UART communication functions (word access)

| **Structure Name** | uni_adc_reg_t | | | |
|---|---|---|---|---|
| **Description** | For word access | | | |
| **Member variable** | **Type** | **No. of bits** | **Name** | **Description** |
| | uint16_t | 16 | WORD | 16-bit access |
| | BYTE | 8 | low | Lower 8-bit access |
| | | 8 | high | Upper 8-bit access |

### Table 4-42 [UART/SPI] Union for A/D converter functions

| **Structure Name** | uni_adcc_t | | | |
|---|---|---|---|---|
| **Description** | Bit field union for SAIC101 ADCC information storage variables | | | |
| **Member variable** | **Type** | **No. of bits** | **Name** | **Description** |
| | uint8_t | 8 | BYTE | ADCC register value |
| | BIT | 4 (lower) | sum | Checksum value |
| | | 1 | overflow | Overflow flag for A/D conversion results |
| | | 3 (upper) | ch | Channel number for conversion results |

(b) **r_sa_uart_control_register.h**

### Table 4-43 [UART] Union for UART communication function (receive packet data)

| **Structure Name** | rcv_packet_t | | | |
|---|---|---|---|---|
| **Description** | For analyzing received data packets | | | |
| **Member variable** | **Type** | **No. of bits** | **Name** | **Description** |
| | uint8_t | 8 | BYTE | 8-bit access |
| | HEADER | 5 (lower) | data | Data |
| | | 3 (upper) | type | type |
| | TYPE1 | 5 | length | length |
| | | 3 | type | type |
| | TYPE2 | 4 | check_sum | Checksum value |
| | | 1 | overflow | Overflow flag for A/D conversion results |
| | | 3 | ad_ch | Channel number for conversion results |
| | TYPE3 | 1 | f0 | Flash Read ready |
| | | 1 | f1 | Flash Write (FW flag) |
| | | 1 | f2 | Flash All Erase (FAE flag) |
| | | 1 | f3 | Register All Write (RAW flag) |
| | | 1 | f4 | A parity error occurred at the last reception (PE flag) |
| | | 3 | type | type |

(c) **r_sa_uart_control_register.c / r_sa_spi_control_register.c**

**Table 4-44 [UART/SPI] Union for UART communication functions (BYTE access)**

| | | | | |
|---|---|---|---|---|
| **Structure name** | uni_sum_t | | | |
| **Description** | For byte access | | | |
| **Member variable** | **Type** | **No. of bits** | **Name** | **Description** |
| | uint8_t | 8 | BYTE | 8-bit access |
| | BIT | 4 (lower) | low | Lower 4 bits |
| | | 4 (upper) | high | Upper 4 bits |

## 4.9 Global constants for user environment-dependent settings

This API can be customized to fit the hardware configuration (interface such as serial module or I/O ports) of the user's environment by changing the constants described in this section.

### 4.9.1 Global constants for user environment-dependent settings in UART communications

(a) **Global constant to store serial module information (UART)**

**Constant name** g_uart_serial_data_tbl

**Declaration** const uart_serial_t g_uart_serial_data_tbl[] = { *constant value* }

**Description** g_uart_serial_data_tbl is the uart_serial_t structure array and registers the functions for serial communications. The ENUM e_uart_ch_t for specifying the global variable to store serial module information must be specified according to the array's index number.

uart_serial_t structure setting

| **Type** | **User Setting** | **Example** |
|---|---|---|
| void (* )(void) | Define the UART start function name generated by the code generator | R_UART1_Start |
| void (* )(void) | Define the UART stop function name generated by the code generator | R_UART1_Stop |
| MD_STATUS (*)(uint8_t * const rx_buf, uint16_t rx_num) | Define the UART receive function name generated by the code generator | R_UART1_Receive |
| MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num) | Describe the UART transmissoin function name generated by the code generator | R_UART1_Send |
| uint8_t (*)(uint8_t *packet_data, uint8_t rx_buffer[], uint16_t read_pos) | Define the header data acquisition function name included in the sample code | R_UART1_GetHeader |
| uint8_t (*)(uint16_t rx_cnt) | Define the number of received data packets check function included in the sample code | R_UART1_Getdata |
| void (*)(uint8_t setting) | Define the name of the UART setting change function included in the sample code | R_UART1_SettingChange |

### (b) Global constant to store SAIC information (UART)

**Constant name** g_uart_saic_data_tbl

**Declaration** const uart_saic_t g_uart_saic_data_tbl[] = { *constant value* } ;

**Description** g_uart_saic_data_tbl is the uart_saic_t structure array and specifies the type of communication with the connected SAIC. This constant associates ENUM e_uart_ch_t for specifying the global variable to store serial module information and ENUM e_saic_type_t for specifying the SAIC type (part name), and the array's index number serves as the SAIC number.

uart_saic_t structure setting

| Type | User Setting | Example |
|---|---|---|
| e_uart_ch_t | Define the element number corresopnding to the ENUM e_uart_ch_t for specifying the global variable to store serial module information | E_UART1 |
| e_saic_type_t | Define the element number corresopnding to the ENUM e_saic_type_t for specifying the SAIC type.(part name) | E_SAIC101 |

### (c) Global constant to store RESET information (UART)

**Constant name** g_uart_reset_data_tbl

**Declaration** const uart_saic_t g_uart_saic_data_tbl[] = { *constant value* } ;

**Description** g_uart_reset_data_tbl is the uart_reset_t structure array and specifies the "RESET by:" for each connected SAIC. When connecting more than one SAIC, normally the user needs to define the same number of arrays as there are connected SAICs, but the user needs to define only one array for the power-on IC.

uart_reset_t structure setting

| Type | User Setting | Example |
|---|---|---|
| e_reset_process_t | Define the SAIC "Reset by:" | E_SAIC_POWERON_RESET |
| uint8_t * | When using an external reset for "Reset by:", define the address of the port register connected to the RESET pin. Specify NULL in all other cases. | NULL |
| uint8_t | When using an external reset for "Reset by:", define the bit number of the port register connected to the RESET pin. Specify 0 in all other cases. | 0U |
| uint32_t | Number of NOP() executions for wait | D_PON_RST_NOP_CNT |
| uint8_t | This is the SAIC number when using an internal "Reset by:".<br>Although this API can only be used with an internal "Reset by:", the user needs to specify the SAIC number. (Please use with API Builder SAIC101.) | 0U |

### (d) Declaration examples

- Case 1

  — SAIC101 is connected to UART1. "RESET by:" is set to power-on reset.

```
r_sa_uart_control_register.h
typedef enum
{
    E_SAIC300 = 0x00U,                          /* SAIC300      */
    E_SAIC301,                                  /* SAIC301      */
    E_SAIC500,                                  /* SAIC500      */
    E_SAIC501,                                  /* SAIC501      */
    E_SAIC502,                                  /* SAIC502      */
    E_SAIC101,                                  /* SAIC101      */
    E_IC_TYPE_MAX,                              /* Determines maximum value */
} e_saic_type_t;

typedef enum
{
    E_UART0 = 0x00U,                    /* UART0 */
    E_UART1,                            /* UART1 */
    E_UART2,                            /* UART2 */
    E_UART3,                            /* UART3 */
    E_UART_MAX,                         /* Determine maximum value */
} e_uart_ch_t;

r_sa_uart_control_register_user.c
const uart_serial_t g_uart_serial_data_tbl[] =
{
    { NULL,          NULL,         NULL,            NULL,         NULL,              NULL,            NULL,                  },
    { R_UART1_Start, R_UART1_Stop, R_UART1_Receive, R_UART1_Send, R_UART1_GetHeader, R_UART1_Getdata, R_UART1_SettingChange, },
};

const uart_saic_t g_uart_saic_data_tbl[] =
{
//  { UART_ch,    sa_type,       }, /* format                */
    { E_UART1,    E_SAIC101,     }, /* SAIC information when SAIC number = 0 */
};

const uart_reset_t g_uart_reset_data_tbl[] =
{
    //process,                    Port address, Bit num, nop_cnt,                uart_saic_t number, }, /* format            */
    { E_SAIC_POWERON_RESET,       NULL,          0U,      D_PON_RST_NOP_CNT,       0U,              }, /* First RESET by: */
};
```

- Case 2
  — Four SAIC101s are connected to UART0, 1, 2, and 3. “RESET by:” is set to power-on reset.

```
r_sa_uart_control_register.h
typedef enum
{
    E_SAIC300 = 0x00U,                          /* SAIC300      */
    E_SAIC301,                                  /* SAIC301      */
    E_SAIC500,                                  /* SAIC500      */
    E_SAIC501,                                  /* SAIC501      */
    E_SAIC502,                                  /* SAIC502      */
    E_SAIC101,                                  /* SAIC101      */
    E_IC_TYPE_MAX,                              /* Determine maximum value */
} e_saic_type_t;

typedef enum
{
    E_UART0 = 0x00U,                    /* UART0 */
    E_UART1,                            /* UART1 */
    E_UART2,                            /* UART2 */
    E_UART3,                            /* UART3 */
    E_UART_MAX,                         /* Determine maximum value */
} e_uart_ch_t;

r_sa_uart_control_register_user.c
const uart_serial_t g_uart_serial_data_tbl[] =
{
    { R_UART0_Start, R_UART0_Stop, R_UART0_Receive, R_UART0_Send, R_UART0_GetHeader, R_UART0_Getdata, R_UART0_SettingChange, },
    { R_UART1_Start, R_UART1_Stop, R_UART1_Receive, R_UART1_Send, R_UART1_GetHeader, R_UART1_Getdata, R_UART1_SettingChange, },
    { R_UART2_Start, R_UART2_Stop, R_UART2_Receive, R_UART2_Send, R_UART2_GetHeader, R_UART2_Getdata, R_UART2_SettingChange, },
    { R_UART3_Start, R_UART3_Stop, R_UART3_Receive, R_UART3_Send, R_UART3_GetHeader, R_UART3_Getdata, R_UART3_SettingChange, },
};

const uart_saic_t g_uart_saic_data_tbl[] =
{
//  { UART_ch,    sa_type,       }, /* format                */
    { E_UART0,    E_SAIC101,     }, /* SAIC information when SAIC number = 0 */
    { E_UART1,    E_SAIC101,     }, /* SAIC information when SAIC number = 1 */
    { E_UART2,    E_SAIC101,     }, /* SAIC information when SAIC number = 2 */
    { E_UART3,    E_SAIC101,     }, /* SAIC information when SAIC number = 3 */
};


const uart_reset_t g_uart_reset_data_tbl[] =
{
    //process,                    Port address, Bit num, nop_cnt,                uart_saic_t number, }, /* format           */
    { E_SAIC_POWERON_RESET,       NULL,         0U,      D_PON_RST_NOP_CNT,      3U,             }, /* First RESET by: */
};
```

### 4.9.2 Global constants for user environment-dependent settings when using SPI

#### (a) Global constant to store serial module information (SPI)

**Constant name** g_spi_serial_data_tbl

**Declaration** const spi_serial_t g_spi_serial_data_tbl[] = { *constant value* }

**Description** g_spi_serial_data_tbl is the spi_serial_t structure array and registers the functions/registers for serial communications. The content of the spi_serial_t type changes according to the value of the macro declaration D_SPI_OPERATION for user environment-dependent settings. The ENUM e_csi_ch_t for specifying the global variable to store serial module information must be specified according to the array's index number.

spi_serial_t structure setting

When using the interrupt function

| Type | User Setting | Example |
|---|---|---|
| void (* )(void) | Define the SPI start function name generated by the code generator | R_CSI10_Start |
| void (* )(void) | Define the SPI stop function name generated by the code generator | R_CSI10_Stop |
| MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf) | Define the SPI transmit/receive function name generated by the code generator | R_CSI10_Send_Receive |

When using register polling

| Type | User Setting | Example |
|---|---|---|
| uint8_t * | Define the address of the transmission data register corresponding to the CSI channel | &SIO10 |
| uint8_t * | Define the address of the receive data register corresponding to the CSI channel | &SIO10 |
| uint8_t * | Define the address of the interrupt request flag register corresponding to the CSI channel | &IF1L |
| uint8_t | Define the bit number of the interrupt request flag registercorresponding to the CSI channel | 1U |
| uint16_t * | Define the address of the SMR register corresponding to the CSI channel | (uint16_t *)&SSR02 |
| uint16_t * | Define the address of the SIR register corresponding to the CSI channel | (uint16_t *)&SIR02 |
| void (* )(void) | Define the name of the SPI start function generated in the code generator | R_CSI10_MaskStart |
| void (* )(void) | Define the name of the SPI stop function generated in the code generator | R_CSI10_Stop |

#### (b) Global constant to store SAIC information (SPI)

**Constant name** g_spi_saic_data_tbl

**Declaration** const spi_saic_t g_spi_saic_data_tbl[] = { *constant value* } ;

**Description** g_spi_saic_data_tbl is the spi_saic_t structure array and specifies the method of communication with the connected SAIC. This constant associates ENUM spi_saic_t for specifying the global variable to store serial module information and ENUM e_saic_type_ to specifying the SAIC type (part name), and the array's index number serves as the SAIC number.

spi_saic_t structure setting

| Type | User Setting | Example |
|---|---|---|
| e_csi_ch_t | Define the CSI channel number. | E_CSI10 |
| e_saic_type_t | Define the SAIC type (part name) | E_SAIC101 |
| uint8_t * | Define the address of the port register connected to the CS pin. | &P0 |
| uint8_t | Define the bit number of the port register connected to the CS pin. | 6U |
| uint8_t * | Define the address of the port register connected to the INT pin. | &P0 |
| uint8_t | Define the bit number of the port register connected to the INT pin. | 7U |

### (c) Global constant to store RESET information (SPI)

**Constant name** g_spi_reset_data_tbl

**Declaration** const spi_reset_t g_spi_reset_data_tbl[] = { constant value } ;

**Description** g_spi_reset_data_tbl is the spi_reset_t structure array and specifies the “RESET by:” for each connected SAIC. When connecting more than one SAIC, normally the user needs to define the same number of arrays as there are connected SAICs, but the user needs to define only one array for the power-on IC.

uart_reset_t structure setting

| Type | User Setting | Example |
|---|---|---|
| e_reset_process_t | Define the SAIC “Reset by:”. | E_SAIC_POWERON_RESET |
| uint8_t * | When using an external reset for “Reset by:”, define the address of the port register connected to the RESET pin. Specify NULL in all other cases. | NULL |
| uint8_t | When using an external reset for “Reset by:”, define the bit number of the port register connected to the RESET pin. Specify 0 in all other cases. | 0U |
| uint32_t | Number of NOP() executions for wait | D_PON_RST_NOP_CNT |
| uint8_t | This is the SAIC number when using an internal “Reset by:”.<br>Although this API can only be used with an internal “Reset by:”, the user needs to specify the SAIC number. (Please use with API Builder SAIC101.) | 0U |

### (d) Declaration examples

- Case 1
  - Using SPI interrupt
  - SAIC101 is connected to CSI10, CS_B pin to port P06, and INT pin to P07. “RESET by:” is set to power-on reset.

```
r_sa_spi_control_register.h
typedef enum
{
    E_SAIC300 = 0x00U,                          /* SAIC300      */
    E_SAIC301,                                  /* SAIC301      */
    E_SAIC500,                                  /* SAIC500      */
    E_SAIC501,                                  /* SAIC501      */
    E_SAIC502,                                  /* SAIC502      */
    E_SAIC101,                                  /* SAIC101      */
    E_IC_TYPE_MAX,                              /* Determine maximum value */
} e_saic_type_t;

typedef enum
{
    E_CSI00 = 0x00U,                    /* CSI00 */
    E_CSI10,                             /* CSI10 */
    E_CSI_MAX                            /* Determine maximum value */
} e_csi_ch_t;

r_sa_spi_control_register_user.c
const spi_saic_t g_spi_saic_data_tbl[] =
{
  { E_CSI10,   E_SAIC101,     &P0,       6U,         &P0,        7U,          }, /* SAIC information when SAIC number = 0 */
};

const spi_serial_t g_spi_serial_data_tbl[] =
{
//  { CSI_Start,     CSI_Stop,     CSI_Send_Receive,     }, /* format  */
    { NULL,          NULL,         NULL,                 }, /* CSI00   */
    { R_CSI10_Start, R_CSI10_Stop, R_CSI10_Send_Receive, }, /* CSI10   */
};

const uart_reset_t g_uart_reset_data_tbl[] =
{
    //process,                    Port address, Bit num, nop_cnt,                spi_saic_t number, }, /* format            */
    { E_SAIC_POWERON_RESET,       NULL,         0U,      D_PON_RST_NOP_CNT,      0U,              }, /* First RESET by: */
};
```

- Case 2
  - Using SPI register polling
  - SAIC101 is connected to CSI10, CS_B pin to port P16, and INT pin to P17.
  - SAIC500 is connected to CSI10, CS_B pin to port P14, and RESET pin to P40.
  - SAIC501 is connected to CSI00, CS_B pin to port P20, and RESET pin to P41.
  - First RESET by: is specified as external RESET by pin P40.
  - Second RESET by: is specified as external RESET by pin P41.
  - Third RESET by: is specified as power-on reset.

```
r_sa_spi_control_register.h
typedef enum
{
    E_SAIC300 = 0x00U,                          /* SAIC300      */
    E_SAIC301,                                  /* SAIC301      */
    E_SAIC500,                                  /* SAIC500      */
    E_SAIC501,                                  /* SAIC501      */
    E_SAIC502,                                  /* SAIC502      */
    E_SAIC101,                                  /* SAIC101      */
    E_IC_TYPE_MAX,                              /* Determine maximum value */
} e_saic_type_t;

typedef enum
{
    E_CSI00 = 0x00U,                    /* CSI00 */
    E_CSI10,                             /* CSI10 */
    E_CSI_MAX                            /* Determine maximum value */
} e_csi_ch_t;

r_sa_spi_control_register_user.c
const spi_saic_t g_spi_saic_data_tbl[] =
{
//  { csi_ch,    sa_type,       p_cs_addr, cs_bit_num, p_int_addr, int_bit_num, }, /* format                */
    { E_CSI10,   E_SAIC101,     &P1,       6U,        &P1,        7U,          }, /* SAIC information when SAIC number = 0 */
    { E_CSI10,   E_SAIC500,     &P1,       4U,        NULL,       0U,          }, /* SAIC information when SAIC number = 1 */
    { E_CSI00,   E_SAIC501,     &P2,       0U,        NULL,       0U,          }, /* SAIC information when SAIC number = 2 */
};

const spi_serial_t g_spi_serial_data_tbl[] =
{
    { &SIO00, &SIO00, &IF0H, 5U, (uint16_t *)&SSR00, (uint16_t *)&SIR00, R_CSI00_MaskStart, R_CSI00_Stop, }, /* CSI00   */
    { &SIO10, &SIO10, &IF1L, 1U, (uint16_t *)&SSR02, (uint16_t *)&SIR02, R_CSI10_MaskStart, R_CSI10_Stop, }, /* CSI10   */
};

const spi_reset_t g_spi_reset_data_tbl[] =
{
    //process,                    Port address, Bit num, nop_cnt,                spi_saic_t number, }, /* format            */
    { E_SAIC_EXTERNAL_RESET,      &P4,          0U,      D_HARD_RESET_NOP_CNT1,  1U,             }, /* First RESET by: */
    { E_SAIC_EXTERNAL_RESET,      &P4,          1U,      D_HARD_RESET_NOP_CNT2,  2U,             }, /* Second RESET by: */
    { E_SAIC_POWERON_RESET,       NULL,         0U,      D_PON_RST_NOP_CNT,      0U,             }, /* Third RESET by: */
};
```

## 4.10 Global Constants

This section describes the global constants defined in the API.

### 4.10.1 UART Control

Table 4-45 Global constants to define UART communications used with SAIC

| Part name | Global constant name | Description |
|---|---|---|
| const uint8_t | g_uart_saic_data_tbl_size | Number of elements of global constant to store SAIC information |
| const uint8_t | g_uart_reset_data_tbl_size | Number of elements of global constant to store RESET information |

Table 4-46 Global constants to define SAIC101 flash memory write data example

| Part name | Global constant name | Description |
|---|---|---|
| const uint8_t | g_uart_flash_data_tbl_size | Number of elements of g_uart_smartanalog_flash_data |
| const saic_data_t | g_uart_smartanalog_flash_data[] | Initialization example of write data to register shadow area in flash memory |

Table 4-47 Global constants to define CPU wait time

| Part name | Global constant name | Description |
|---|---|---|
| const uint16_t | g_uart_4800bps_half_bit_time | Counter value of number of loops for time equivalent to half the 1-bit width time (about 105 us) at 4800bps.<br>Calculated from the value defined in “D_UART_4800BPS_HALF_BIT”Note |
| const uint16_t | g_uart_250kbps_half_bit_time | Counter value of number of loops for time equivalent to half the 1-bit width time (about 2 us) at 250kbps.<br>Calculated from the value defined in “D_UART_250kbps_HALF_BIT”Note |
| const uint16_t | g_uart_response_step | Counter value of number of loops for time for calculating the number of steps for functions waiting for a response<br>Calculated from the value defined in “D_UART_10MS” Note |
| const uint32_t | g_uart_saic_auto_adc_timeout | Counter value of number of loops for settling time when OSR = 2048 (8.192 ms)<br>Calculated from the value defined in “D_UART_3US_NOP_CNT Note |
| const uint16_t | g_uart_5ms_nop_cnt | Counter value of number of loops for SBIAS to stop and then wait for stabilization after transfer to sleep mode (at least 5 ms)<br>Calculated from the value defined in “D_UART_5MS” Note |
| const uint16_t | g_uart_1800us_nop_cnt | Counter value of number of loops for AREG to start and then wait for stabilization (at least 1800 us)<br>Calculated from the value defined in “D_UART_1800US”Note |
| const uint16_t | g_uart_270us_nop_cnt | Counter value of number of loops for return from sleep mode and then wait for stabilization (at least 270 us)<br>Calculated from the value defined in “D_UART_270US”Note |
| const uint16_t | g_uart_250us_nop_cnt | Counter value of number of loops for SBIAS to start and then wait for stabilization (at least 250 us)<br>Calculated from the value defined in “D_UART_250US” Note |

Note: Calculate approximate number NOP command executions from reference frequency in “D_CPU_CLK_MHZ”.

Table 4-48 Private global constant for bit operations

| Part name | Global constant name | Description |
|---|---|---|
| const uint8_t | gs_bit_tbl[] | Power-of-two data table constant. Stores data corresponding to index. |

### 4.10.2 SPI Control

**Table 4-49 Global constants to define SPI communications used with SAIC**

| Part name | Global constant name | Description |
|---|---|---|
| const uint8_t | g_spi_saic_data_tbl_size | Number of elements of global constant to store SAIC information |
| const uint8_t | g_spi_reset_data_tbl_size | Number of elements of global constant to store RESET information |

**Table 4-50 Global constants to define SAIC101 flash memory write data example**

| Part name | Global constant name | Description |
|---|---|---|
| const uint8_t | g_spi_flash_data_tbl_size | Number of elements of g_spi_smartanalog_flash_data |
| const saic_data_t | g_spi_smartanalog_flash_data[] | Initialization example of write data to register shadow area in flash memory |

**Table 4-51 Global constants to define CPU wait time**

| Part name | Global constant name | Description |
|---|---|---|
| const uint16_t | g_spi_5ms_nop_cnt | Counter value of number of loops for SBIAS to stop and then wait for stabilization after transfer to sleep mode (at least 5 ms)<br>Calculated from the value defined in “D_SPI_5MS” Note |
| const uint16_t | g_spi_1800us_nop_cnt | Counter value of number of loops for AREG to start and then wait for stabilization (at least 1800 us)<br>Calculated from the value defined in “D_SPI_1800US”Note |
| const uint16_t | g_spi_820us_nop_cnt | Counter value of number of loops for return from sleep mode and then wait for stabilization (at least 820 us)<br>Calculated from the value defined in “D_SPI_820US”Note |
| const uint16_t | g_spi_250us_nop_cnt | Counter value of number of loops for SBIAS to start and then wait for stabilization (at least 250 us)<br>Calculated from the value defined in “D_SPI_250US”Note |
| const uint16_t | g_spi_3us_nop_cnt | Counter value of number of loops for Flash (Burst) Read command delay time (at least 3 us)<br>Calculated from the value defined in “D_SPI_3US_NOP_CNT”Note |

Note: Calculate approximate number NOP command executions from reference frequency in “D_CPU_CLK_MHZ”.

**Table 4-52 Private Global constant for bit operations**

| Part name | Global constant name | Description |
|---|---|---|
| const uint8_t | gs_bit_tbl[] | Power-of-two data table constants. Stores data corresponding to index. |

## 4.11 Global Variables

This section describes the global variables defined in the API.

(a) **r_sa_uart_control_register.c**

**Table 4-53 [UART] Global variables**

| Part name | Global Variable Name | Description |
|---|---|---|
| static uint8_t | gs_adc_1shot | ADC 1Shot acquisition flag: 1 = ADC 1Shot acquired, 0 = Other |
| static uint16_t | gs_uart_half_bit_time | Time equivalent to half the 1-bit width |
| static uint16_t | gs_uart_negotiation_timeout | Communication negotiation timeout value (depends on the baud rate) |

# 5. Communication-related Definitions

## 5.1 API Function Specifications

### 5.1.1 [UART/SPI] Smart Analog initialization function

void R_SAIC_UART_Init(void)

void R_SAIC_SPI_Init(void)

| | |
|---|---|
| Outline | Smart Analog initialization function |
| Header | -For UART<br>r_sa_uart_control_register.h<br>-For SPI<br>r_sa_spi_control_register.h |
| Argument | None |
| Global Variable | - For UART<br>gs_uart_half_bit_time:<br>Stop bit completion counter<br>g_uart_4800bps_half_bit_time:<br>Time equivalent to half the 1-bit width at 4800bps<br>gs_uart_negotiation_timeout:<br>Communication negotiation timeout value<br>g_uart_reset_data_tbl_size:<br>Number of stored RESET information entries<br>- For SPI<br>g_spi_reset_data_tbl_size:<br>Number of stored RESET information entries |
| Return Value | None |
| Description | 1. Initialization<br>-For UART<br>1.1. Initialize the counter for UART stop bit completion<br>1.2. Initialize the UART communication negotiation timeout value.<br>- For SPI<br>1.1. Calls CS enable function for each channel and selects all SAIC.<br>1.2. Calls CS disable function for each channel and unselects all SAIC.<br>2. Continuously calls Smart Analog RESET function for the specified number of elements. |

### 5.1.2 [UART/SPI] Smart Analog RESET function

void R_SAIC_UART_Reset(uint8_t reset_num)

void R_SAIC_SPI_Reset(uint8_t reset_num)

| | |
|---|---|
| Outline | Smart Analog RESET function |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | reset_num:<br>Array number of global variables to store RESET information. |
| Global Variable | - For UART<br>g_uart_reset_data_tbl[]:<br>Global variables to store RESET information<br>- For SPI<br>g_spi_reset_data_tbl[]:<br>Global variables to store RESET information |
| Return Value | None |
| Description | Call internal function for RESET processing corresponding to the specified RESET process. |

### 5.1.3 [UART/SPI] Read register bytes function

saic_status_t R_SAIC_UART_Read(uint8_t saic_num, saic_data_t *data, uint8_t num)

saic_status_t R_SAIC_SPI_Read(uint8_t saic_num, saic_data_t *data, uint8_t num)

Outline
Read register bytes function

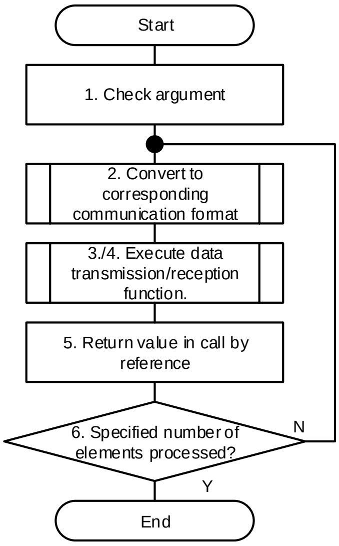
Header
- For UART
  r_sa_uart_control_register.h
- For SPI
  r_sa_spi_control_register.h

Argument
saic_num:
  SAIC number
*data:
  Data buffer pointer
num:
  Number of data units

Global Variable
- For UART
  g_uart_saic_data_tbl_size:
    Number of stored SAIC information entries
  gs_uart_half_bit_time:
    Stop bit completion counter
- For SPI
  g_spi_saic_data_tbl_size:
    Number of stored SAIC information entries

Return Value
saic_status_t:
  D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
Error processing branches have been omitted from flowchart.


Start
1. Check argument
2. Convert to corresponding communication format
3./4. Execute data transmission/reception function.
5. Return value in call by reference
6. Specified number of elements processed?
N
Y
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Converts to corresponding communication format based on specified address.
3. Executes data transmission/reception execution function.
4. If data transmission/reception execution function fails, returns corresponding error and goes to end.
5. Stores read value in argument *data->data.
6. Repeats steps 2 to 5 for the specified number of elements.

-

### 5.1.4 [UART/SPI] Write register bytes function

saic_status_t R_SAIC_UART_Write(uint8_t saic_num, saic_data_t *data, uint8_t num)

saic_status_t R_SAIC_SPI_Write(uint8_t saic_num, saic_data_t *data, uint8_t num)

Outline
Write register bytes function

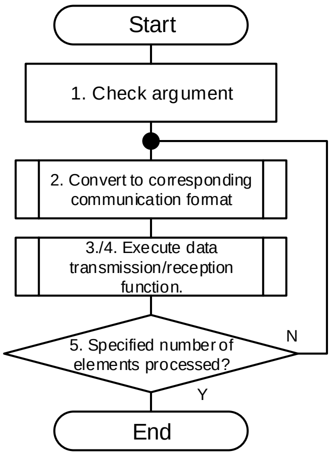
Header
- For UART
  r_sa_uart_control_register.h
- For SPI
  r_sa_spi_control_register.h

Argument
saic_num:
  SAIC number
*data:
  Data buffer pointer
num:
  Number of data units

Global Variable
- For UART
  g_uart_saic_data_tbl_size:
    Number of stored SAIC information entries
  gs_uart_half_bit_time:
    Stop bit completion counter
- For SPI
  g_spi_saic_data_tbl_size:
    Number of stored SAIC information entries

Return Value
saic_status_t:
  D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
Error processing branches have been omitted from flowchart.


Start
1. Check argument
2. Convert to corresponding communication format
3./4. Execute data transmission/reception function.
5. Specified number of elements processed?
N
Y
End

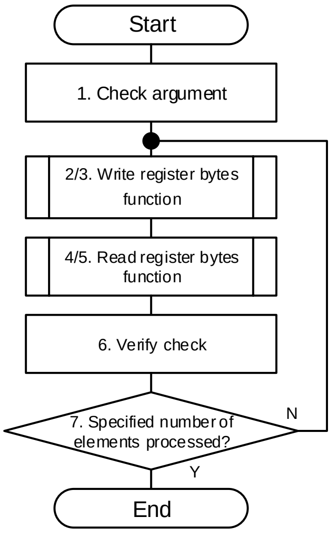
1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Converts to corresponding communication format based on specified address and data.
3. Executes data transmission/reception function.
4. If data transmission/reception function fails, returns corresponding error and goes to end.
5. Repeats steps 2 to 4 for the specified number of elements.

### 5.1.5 [UART/SPI] Write-verify register bytes function

saic_status_t R_SAIC_UART_WriteVerify
(uint8_t saic_num, saic_data_t *data, uint8_t num, uint8_t *err_index)

saic_status_t R_SAIC_SPI_WriteVerify
(uint8_t saic_num, saic_data_t *data, uint8_t num, uint8_t *err_index)

| | |
|---|---|
| Outline | Write-verify register bytes function |
| Header | - For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*data:<br>Data buffer pointer<br>num:<br>Number of data units<br>*err_index:<br>Index value of the saic_data array that caused an error |
| Global Variable | - For UART<br>g_uart_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>- For SPI<br>g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY or D_SAIC_ERR_PARAM |
| Description | Error processing branches have been omitted from flowchart. |


Start
1. Check argument
2/3. Write register bytes function
4/5. Read register bytes function
6. Verify check
7. Specified number of elements processed?
N
Y
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Executes write register bytes function to specified address, then write data.
3. If write register bytes function fails, returns corresponding error.
4. Executes read register bytes function to specified address, then reads data.
5. If function fails, returns corresponding error.
6. If write and read value of specified address do not match, returns unmatched element number D_SAIC_ERR_VERIFY, and goes to end.
7. Repeats steps 2 to 6 for the specified number of elements.

### 5.1.6 [UART/SPI] SAIC101 command function

saic_status_t R_SAIC_UART_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

saic_status_t R_SAIC_SPI_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

Outline
SAIC101 command function
Process branches to one of the following based on e_func argument: register burst-read/write function/flash memory function.

Header
- For UART
  r_sa_uart_control_register.h
- For SPI
  r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
e_func:
Specify SAIC101 functions
*data:
Data buffer pointer
length:
Data length

Global Variable
- For UART
  g_uart_saic_data_tbl_size:
  Number of stored SAIC information entries
  g_uart_saic_data_tbl[]:
  Global variable to store SAIC information
- For SPI
  g_spi_saic_data_tbl_size:
  Number of stored SAIC information entries
  g_spi_saic_data_tbl[]:
  Global variable to store SAIC information

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY

Description
1. If argument error occurs, returns D_SAIC_ERR_COM and goes to end.
2. Calls internal functions corresponding to the specified processes.
3. If internal function fails, returns corresponding error and goes to end.

### 5.1.7 [UART] Communication setting negotiation function

saic_status_t R_SAIC_UART_Negotiation(uint8_t saic_num, uint8_t saic_uartcnt_set)

Outline
Communication setting negotiation function
Automatically recognizes communication setting with SAIC101, sets, 250kbps/parity odd.

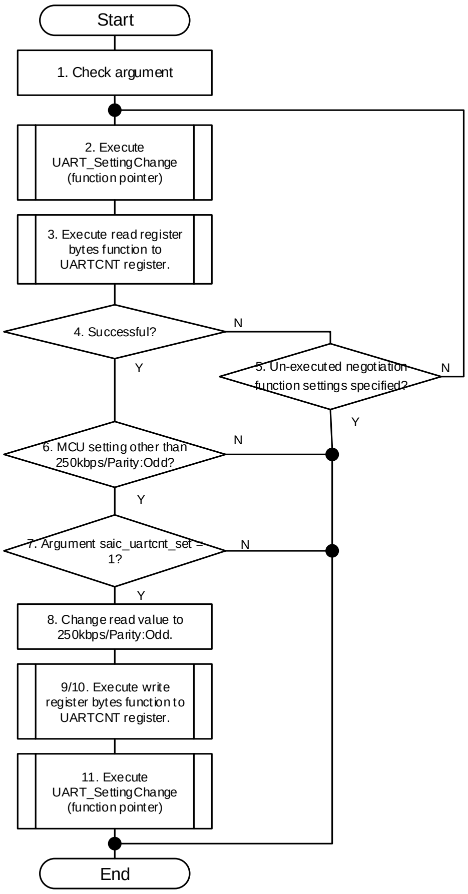
Header
r_sa_uart_control_register.h

Argument
saic_num:
SAIC number
saic_uartcnt_set:
SAIC UART communication setting change availability judgment flag (0: No change, 1: Changed)

Global Variable
g_uart_saic_data_tbl_size:
Number of stored SAIC information entries
gs_uart_half_bit_time:
Stop bit completion counter
gs_uart_negotiation_timeout:
UART communication negotiation timeout value
g_uart_saic_data_tbl[]:
Global variable to store SAIC information
g_uart_serial_data_tbl:
UART table
g_uart_4800bps_half_bit_time:
Time equivalent to half the 1-bit width at 4800bps
g_uart_250kbps_half_bit_time:
Time equivalent to half the 1-bit width at 250kbps (approx. 2 us)
g_uart_response_step:
Time for calculating the number of steps for functions waiting for a response

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
Error processing branches have been omitted from flowchart.


Start
1. Check argument
2. Execute UART_SettingChange (function pointer)
3. Execute read register bytes function to UARTCNT register.
4. Successful?
N
Y
5. Un-executed negotiation function settings specified?
N
Y
6. MCU setting other than 250kbps/Parity:Odd?
N
Y
7. Argument saic_uartcnt_set = 1?
N
Y
8. Change read value to 250kbps/Parity:Odd.
9/10. Execute write register bytes function to UARTCNT register.
11. Execute UART_SettingChange (function pointer)
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end. If function pointer is NULL, returns D_SAIC_ERR_COM and goes to end.
2. Executes UART_SettingChange (function pointer), and changes MCU's UART communication setting.
3. Executes read register bytes function to UARTCNT register.
4. Determine negotiation timeout.
5. Goes to step 2 if is a declaration is defined for setting un-executed negotiation function processes.
6. If successful and MCU communications are set to 250kbps/Parity:Odd, processing is completed.
7. If argument saic_uartcnt_set is not 1, processing is completed.
8. Changes UART setting from value read from UARTCNT register to 250kbps/Parity:Odd.
9. Executes write register bytes function to UARTCNT register.
10. If function fails, returns corresponding error and goes to end.
11. Executes UART_SettingChange (function pointer), and changes MCU's UART setting to 250kbps/Parity:Odd.

### 5.1.8 [SPI] Read register bits function

saic_status_t R_SAIC_SPI_ReadBit
(uint8_t saic_num, saic_data_bit_t *data, uint8_t num, uint8_t *err_index)

Outline
Read register bits function

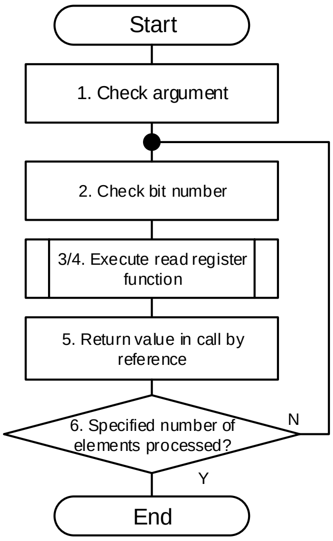
Header
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
*data:
Data buffer pointer
num:
Number of data units
*err_index:
Index value of the saic_ data array that caused an error

Global Variable
g_spi_saic_data_tbl_size:
Number of stored SAIC information entries
gs_bit_tbl[]:
Power-of-two data table constant

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
Error processing branches have been omitted from flowchart.

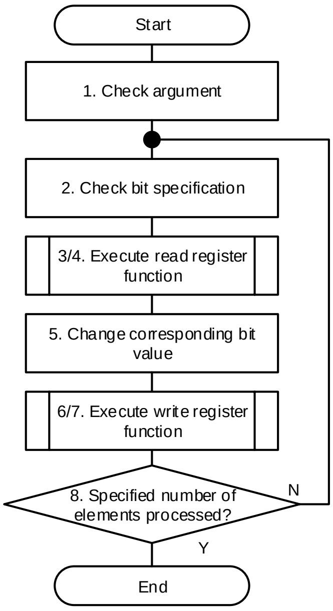

Start
1. Check argument
2. Check bit number
3/4. Execute read register function
5. Return value in call by reference
6. Specified number of elements processed?
N
Y
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. If bit number specification error occurs, returns D_SAIC_ERR_PARAM and goes to end.
3. Executes read register function.
4. If read register function fails, returns corresponding error and goes to end.
5. Stores value of read bit in argument *data->bit_data.
6. Repeats steps 2 to 5 for the specified number of elements

### 5.1.9 [SPI] Write register bits function

saic_status_t R_SAIC_SPI_WriteBit
(uint8_t saic_num, saic_data_bit_t *data, uint8_t num, uint8_t *err_index)

| | |
|---|---|
| Outline | Write register bits function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*data:<br>Data buffer pointer<br>num:<br>Number of data units<br>*err_index:<br>Index value of the saic_ data array that caused an error |
| Global Variable | g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | Error processing branches have been omitted from flowchart. |

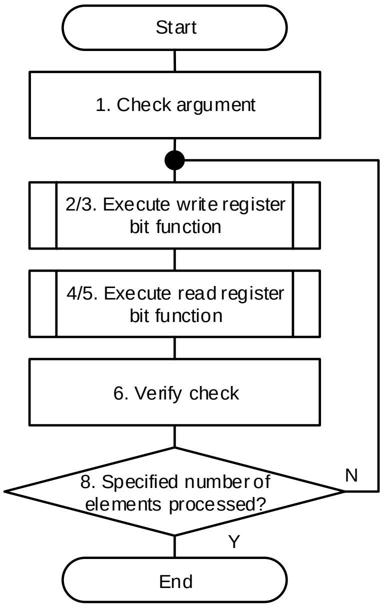

Start
1. Check argument
2. Check bit specification
3/4. Execute read register function
5. Change corresponding bit value
6/7. Execute write register function
8. Specified number of elements processed?
N
Y
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. If bit number or write data specification error occurs, returns D_SAIC_ERR_PARAM and goes to end.
3. Executes read register function.
4. If read register function fails, returns corresponding error and goes to end.
5. Changes corresponding bit value.
6. Executes write register function.
7. If write register function fails, returns corresponding error and goes to end.
8. Repeats steps 2 to 7 for the specified number of elements.

### 5.1.10 [SPI] Write-verify register bits function

saic_status_t R_SAIC_SPI_WriteVerifyBit(uint8_t saic_num, saic_data_bit_t *data,
uint8_t num, uint8_t *err_index)

| | |
|---|---|
| Outline | Write-verify register bits function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*data:<br>Data buffer pointer<br>num:<br>Number of data units<br>*err_index:<br>Index value of the saic_ data array that caused an error |
| Global Variable | g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY |
| Description | Error processing branches have been omitted from flowchart. |

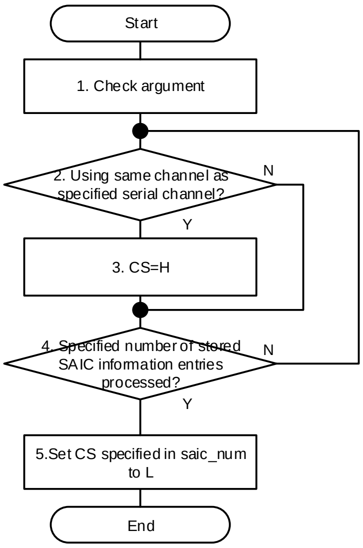

Start
1. Check argument
2/3. Execute write register bit function
4/5. Execute read register bit function
6. Verify check
8. Specified number of elements processed?
N
Y
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Executes write register bit function to specified address.
3. If write register bit function fails, returns corresponding error and goes to end.
4. Executes read register bit function to specified address.
5. If read register bit function fails, returns corresponding error and goes to end.
6. If write and read value of specified address do not match, returns unmatched element number D_SAIC_ERR_VERIFY, and goes to end.
7. Repeats steps 2 to 6 for the specified number of elements.

### 5.1.11 [SPI] CS enable function

void R_SAIC_SPI_CSEnable(uint8_t saic_num)

| | |
|---|---|
| Outline | CS enable function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number |
| Global Variable | g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | None |
| Description | Error processing branches have been omitted from flowchart. |



Start
1. Check argument
2. Using same channel as specified serial channel?
N
Y
3. CS=H
4. Specified number of stored SAIC information entries processed?
N
Y
5.Set CS specified in saic_num to L
End

1. If argument error occurs and function pointer is NULL, process goes to end.
2. Determines if the same channel is used as the serial channel specified in saic_num.
3. Sets CS pin on the SAIC using the same channel to H (not selected).
4. Repeats steps 2 and 3 for the number of SAIC information storage entries.
5. Sets CS pin specified in saic_num to L (selected).

### 5.1.12 [SPI] CS check function

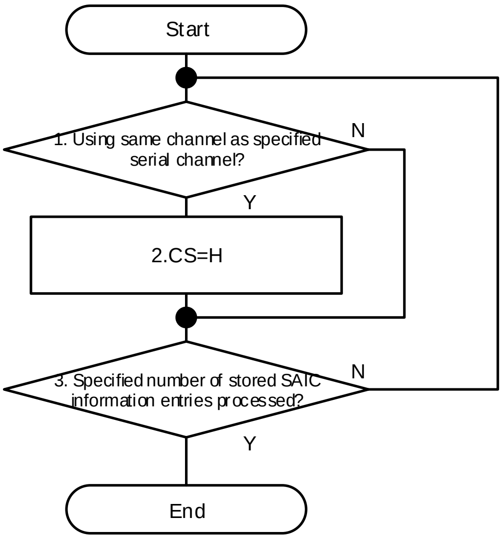
uint8_t R_SAIC_SPI_CSCheck(uint8_t saic_num)

| | |
|---|---|
| Outline | CS check function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number |
| Global Variable | g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br><br>g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | uint8_t:<br>0 = enabled, 1 = disabled |
| Description | 1. If argument error occurs or the function pointer stored in the global variable to store SAIC information, returns 1 and goes to end.<br>2. If the SAIC CS pin specified in saic_num is L (selected), returns 0; if H (not selected), 1. |

### 5.1.13 [SPI] CS disable function

void R_SAIC_SPI_CSDisable(e_csi_ch_t cs_ch)

| | |
|---|---|
| Outline | CS disable function |
| Header | r_sa_spi_control_register.h |
| Argument | cs_ch:<br>Corresponding CSI channel number |
| Global Variable | g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | None |
| Description | Error processing branches have been omitted from flowchart. |


Start
1. Using same channel as specified serial channel?
N
Y
2.CS=H
3. Specified number of stored SAIC information entries processed?
N
Y
End

1. Determines if the same channel is used as the specified serial channel.
2. Sets the SAIC CS pin using the same channel to H (not selected).
3. Repeats steps 1 and 2 for the number of stored SAIC information entries.

## 5.2 Internal Function Specifications

### 5.2.1 [UART/SPI] Smart Analog external RESET function

static void r_saic_uart_external_reset(uint8_t reset_num)

static void r_saic_spi_external_reset(uint8_t reset_num)

| | |
|---|---|
| Outline | Smart Analog external RESET function |
| Header | - For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | reset_num:<br>Array number of global variable to store RESET information |
| Global Variable | - For UART<br>g_uart_reset_data_tbl[]:<br>Global variable to store RESET information<br>gs_bit_tbl[]:<br>Power-of-two data table constant<br>- For SPI<br>g_spi_reset_data_tbl[]:<br>Global variable to store RESET information<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | None |
| Description | 1. Sets port output connected to RESET pin to L, generates external RESET.<br>2. Executes NOP execution function, waits for RESET time.<br>3. Sets port output connected to RESET pin to H, generates external RESET. |

### 5.2.2 [UART/SPI] Smart Analog internal RESET function

static void r_saic_uart_internal_reset(uint8_t reset_num)

static void r_saic_spi_internal_reset(uint8_t reset_num)

| | |
|---|---|
| Outline | Smart Analog internal RESET function |
| Header | - For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | reset_num:<br>Array number of global variable to store RESET information |
| Global Variable | - For UART<br>g_uart_reset_data_tbl[]:<br>Global variable to store RESET information<br>g_uart_saic_data_tbl[]:<br>Global variable to store SAIC information<br>- For SPI<br>g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information<br>g_spi_reset_data_tbl[]:<br>Global variable to store RESET information |
| Return Value | None |
| Description | 1. Generates internal RESET to write 1 to RESET control register corresponding to specified SAIC.<br>2. Cancels internal RESET to write 0 to RESET control register corresponding to specified SAIC. |

### 5.2.3 [UART/SPI] Smart Analog power-on RESET wait function

static void r_saic_uart_poweron_reset(uint8_t reset_num)

static void r_saic_spi_poweron_reset(uint8_t reset_num)

| | |
|---|---|
| Outline | Smart Analog power-on RESET wait function |
| Header | - For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | reset_num:<br>Array number of global variable to store RESET information |
| Global Variable | - For UART<br>g_uart_reset_data_tbl[]:<br>Global variable to store RESET information<br>- For SPI<br>g_spi_reset_data_tbl[]:<br>Global variable to store RESET information |
| Return Value | None |
| Description | Calls NOP execution function, passes number of NOP executions specified in the global variable to store RESET information by value, and waits for RESET time. |

### 5.2.4 [UART/SPI] NOP execution function

static void r_nop_wait(uint32_t nop_cnt)

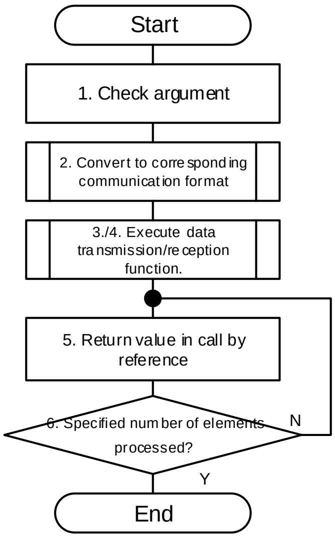
| | |
|---|---|
| Outline | NOP execution function. Executes the NOP() command for the number of times indicated in the argument of nop_cnt. |
| Header | - For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | nop_cnt:<br>Number of NOP executions |
| Global Variable | None |
| Return Value | None |
| Description | Repeats NOP() commands for the number of times specified in nop_cnt. |

### 5.2.5 [UART/SPI] Burst read function

static saic_status_t r_saic_uart_burst_read(uint8_t saic_num, saic_data_t *data, uint8_t length)

static saic_status_t r_saic_spi_burst_read(uint8_t saic_num, saic_data_t *data, uint8_t length)

| | |
|---|---|
| Outline | Burst read function. Issues [Register Burst Read] commands. |
| Header | - For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*data:<br>Data buffer pointer<br>length:<br>Data length |
| Global Variable | - For UART<br>gs_uart_half_bit_time:<br>Stop bit completion counter |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | Error processing branches have been omitted from flowchart. |


Start
1. Check argument
2. Convert to corresponding communication format
3./4. Execute data transmission/reception function.
5. Return value in call by reference
6. Specified number of elements processed?
N
Y
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Converts to communication format that corresponds to specified address.
3. Executes transmission/receive function.
4. If transmission/receive function fails, returns corresponding error and goes to end.
5. Stores read value data in argument *data->data.
6. Repeats step 5 and increments address for the specified number of elements.

### 5.2.6 [UART/SPI] Burst write function

static saic_status_t r_saic_uart_burst_write
(uint8_t saic_num, saic_data_t *data, uint8_t length)

static saic_status_t r_saic_spi_burst_write
(uint8_t saic_num, saic_data_t *data, uint8_t length)

| | |
|---|---|
| Outline | Burst write function. Issues [Register Burst Write] commands. |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*data:<br>Data buffer pointer<br>length:<br>Data length |
| Global Variable | -For UART<br>gs_uart_half_bit_time:<br>Stop bit completion counter |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | 1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.<br>2. Converts to communication format that corresponds to specified address and data.<br>3. Executes transmission/receive function.<br>4. If transmission/reception function fails, returns corresponding error and goes to end. |

### 5.2.7 [UART/SPI] SAIC101 dedicated communication command format conversion function

static void r_saic_uart_format_conv_saic101(uint8_t *address, e_saic101_func_t e_func)

static void r_saic_spi_format_conv_saic101(uint8_t *address, e_saic101_func_t e_func)

| | |
|---|---|
| Outline | SAIC101 dedicated communication command format conversion function |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | e_func:<br>Specify SAIC101 functions<br>*address:<br>Target register address |
| Global Variable | None |
| Return Value | None |
| Description | Converts to communication format that corresponds to SAIC101 specified command. |

### 5.2.8 [SPI] SPI format conversion function

static void r_saic_spi_format_conv(uint8_t saic_num, uint8_t *address, e_rw_t rw)

| | |
|---|---|
| Outline | SPI format conversion function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*address:<br>Target register address<br>rw:<br>Read/write specification |
| Global Variable | g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information |
| Return Value | None |
| Description | Converts to communication format that corresponds to SAIC type (part name) and specified command. |

### 5.2.9 [SPI] Overrun error check function

static uint8_t r_saic_spi_overrun_err_check(uint8_t saic_num)

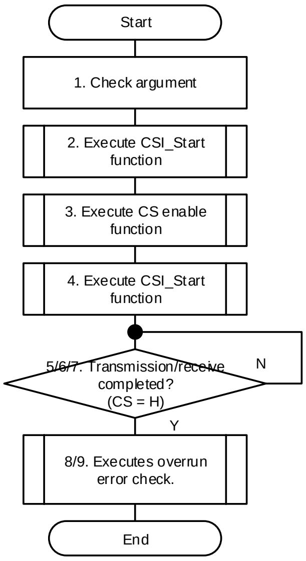
| | |
|---|---|
| Outline | Overrun error check function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number |
| Global Variable | g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information<br>g_csi_overrun_flag:<br>SPI overrun flag variable<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | uint8_t:<br>0 OK or 1 overrun error |
| Description | 1. Executes bit check of corresponding channels specified by the SPI overrun flag variable<br>2. Returns 0 if overrun has not occurred; clears bit and returns 1 if overrun has occurred. |

### 5.2.10 [SPI] Data transmission/reception execution function

static saic_status_t r_saic_spi_write_read
(uint8_t saic_num, uint8_t tx_buffer[], uint8_t rx_buffer[], uint16_t length)

| | |
|---|---|
| Outline | Data transmission/reception execution function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>tx_buffer[]:<br>Transmit buffer<br>rx_buffer[]:<br>Receive buffer<br>length:<br>Data length |
| Global Variable | g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information<br>g_spi_serial_data_tbl[]:<br>Global variable to store serial module information<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | saic_status_t<br>D_SAIC_OK or D_SAIC_ERR_COM |
| Description | Error processing branches have been omitted from flowchart. |

Process when using interrupts

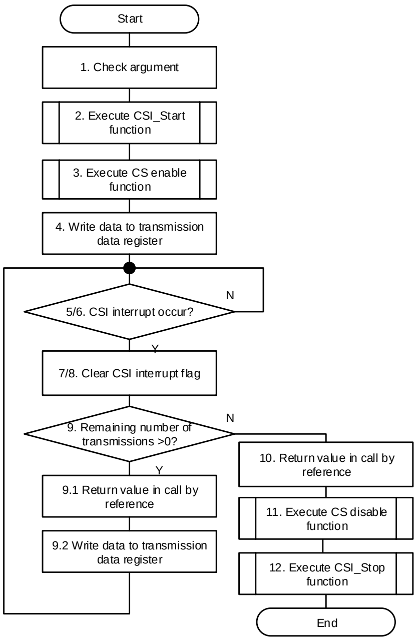

Start
1. Check argument
2. Execute CSI_Start function
3. Execute CS enable function
4. Execute CSI_Start function
5/6/7. Transmission/receive completed? (CS = H)
N
Y
8/9. Executes overrun error check.
End

1. If address and function pointer are NULL, returns D_SAIC_ERR_COM and goes to end.
2. Executes CSI_Start function and enables CSI communication.
3. Executes CS enable function and activates CS on the target SAIC.
4. Executes CSI_Start function and starts transmission/reception.
5. Continues loop until transmission/receive is completed.
6. CSIxx transmission/reception interrupt sets CS = H and triggers CSI_Stop function.
7. If deadlock occurs, processes error, returns D_SAIC_ERR_COM and goes to end.

   Executes CS = H and CSI_STOP functionand stops transmission/reception.

   Executes CS disable function and deactivates CS on target SAIC.
8. Executes overrun error check function.
9. If overrun flag is set, returns D_SAIC_ERR_COM and goes to end.

Process when not using interrupt (use polling)


Start
1. Check argument
2. Execute CSI_Start function
3. Execute CS enable function
4. Write data to transmission data register
N
5/6. CSI interrupt occur?
Y
7/8. Clear CSI interrupt flag
N
9. Remaining number of transmissions >0?
Y
10. Return value in call by reference
9.1 Return value in call by reference
11. Execute CS disable function
9.2 Write data to transmission data register
12. Execute CSI_Stop function
End

1. If address and function pointer are NULL, returns D_SAIC_ERR_COM and goes to end.
2. Executes CSI_Start function.
3. Executes CS enable function.
4. Writes data to transmission data register.
5. Continues loop until CSI interrupt is generated.
6. If deadlock occurs, processes error, returns D_SAIC_ERR_COM, and goes to end.
7. Clears CSI interrupt flag.
8. If communication error occurs, returns D_SAIC_ERR_COM and goes to end.
9. If remaining number of transmissions is >0, repeats steps 5 to 9.

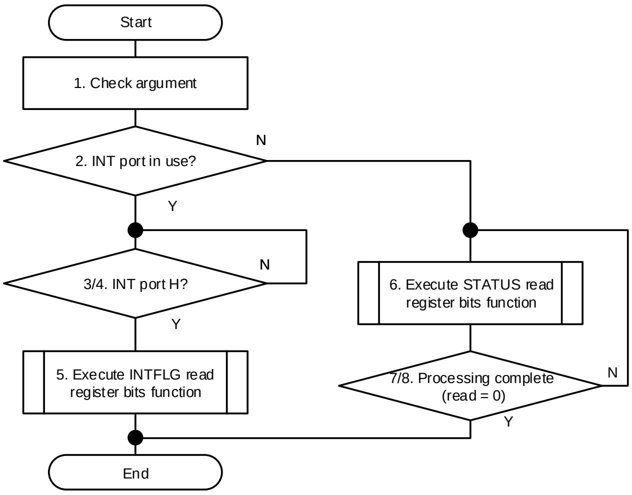
   9.1 Stores receive data in argument rx_buffer[].

   9.2 Writes data to transmission data register.
10. Stores receive data in argument rx_buffer[].
11. Executes CS disable function.
12. Executes CSI_Stop function.

### 5.2.11 [SPI] Polling monitoring function

static saic_status_t r_saic101_spi_polling
(uint8_t saic_num, saic_func_bitRead_t p_func_int, saic_func_bitRead_t p_func_status)

| | |
|---|---|
| Outline | Polling monitoring function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>p_func_int:<br>INTFLG read register bits function<br>p_func_status:<br>STATUS read register bits function |
| Global Variable | g_spi_saic_data_tbl[]:<br>Global variable to store SAIC information<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | saic_status_t<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | Error processing branches have been omitted from flowchart. |


Start
1. Check argument
2. INT port in use?
N
Y
3/4. INT port H?
N
Y
5. Execute INTFLG read register bits function
6. Execute STATUS read register bits function
7/8. Processing complete (read = 0)
N
Y
End

1. If INT port is not used (polling) and p_func_status=NULL, returns D_SAIC_ERR_COM and leaves loop.
2. Determines if INT port is used.
3. If INT port is used, continues loop until INT port goes to H.
4. If deadlock occurs during loop, returns D_SAIC_ERR_COM and leaves loop.
5. Execute INTFLG read register bits function.
6. If INT port is not used (polling), executes STATUS read register bits function.
7. Continues loop until processing is completed (read value = 0).
8. If deadlock occurs during loop, returns D_SAIC_ERR_COM and leaves loop.

### 5.2.12 [UART] Command transmission & response reception function

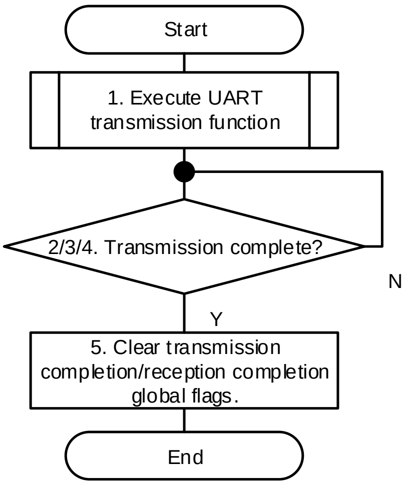
```
static saic_status_t r_saic_uart_write_read
    (uint8_t saic_num, uint8_t tx_buffer[], uint8_t rx_buffer[], uint16_t tx_length, uint16_t * p_rx_length)
```

| | |
|---|---|
| Outline | Command transmission & response reception function |
| Header | r_sa_uart_control_register.h |
| Argument | saic_num:<br>SAIC number<br>tx_buffer[]:<br>Transmit buffer<br>rx_buffer[]:<br>Receive buffer<br>tx_length:<br>Transmission size<br>p_rx_length:<br>Reception size |
| Global Variable | g_uart_saic_data_tbl[]:<br>Global variable to store SAIC information<br>gs_bit_tbl[]:<br>Power-of-two data table constant<br>gs_adc_1shot:<br>ADC 1Shot acquire flag: 1 = ADC 1Shot acquired, 0 = Other<br>g_uart_rx_end_flag:<br>UART reception completion flag variable<br>g_uart_serial_data_tbl:<br>UART table |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM |
| Description | 1. If function pointer is NULL, returns D_SAIC_ERR_COM and goes to end.<br>2. Executes UART reception function and sets reception wait status.<br>3. Executes UART communication start function.<br>4. Executes command transmission function, sends transmit buffer data according to transmission size.<br>5. If command transmission function fails, sets corresponding error.<br>6. If command transmission function is successful, executes UART receive data packet analysis function.<br>7. Stops UART<br>8. Clears ADC 1Shot flag. |

### 5.2.13 [UART] Command transmission function

static saic_status_t r_saic_uart_send_command
(uint8_t saic_num, uint8_t tx_buffer[], uint16_t tx_length)

| | |
|---|---|
| Outline | Command transmission function (enabled only when using interrupts) |
| Header | r_sa_uart_control_register.h |
| Argument | saic_num:<br>SAIC number<br>tx_buffer[]:<br>Transmit buffer<br>tx_length:<br>Transmission size |
| Global Variable | g_uart_saic_data_tbl[]:<br>Global variable to store SAIC information<br>g_uart_serial_data_tbl:<br>UART table<br>gs_bit_tbl[]:<br>Power-of-two data table constant<br>g_uart_tx_end_flag:<br>UART transmission completion flag variable<br>g_uart_rx_end_flag:<br>UART reception completion flag variable<br>gs_uart_negotiation_timeout:<br>Communication negotiation timeout value |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM |
| Description | Error processing branches have been omitted from flowchart. |


Start
1. Execute UART transmission function
2/3/4. Transmission complete?
N
Y
5. Clear transmission completion/reception completion global flags.
End

1. Executes UART transmission function.
2. If negotiation timeout occurs, returns D_SAIC_ERR_COM and leaves loop after UART stops
3. If deadlock occurs, returns D_SAIC_ERR_COM and leaves loop.
4. Continues loop until transmission is completed.
5. Clears transmission completion/reception completion global flags.

### 5.2.14 [UART] UART Receive data packet analysis function

static saic_status_t r_saic_uart_get_response
(uint8_t saic_num, uint8_t rx_buffer[], uint16_t * p_rx_length)

Outline UART receive data packet analysis function. Determines type of response from received header and receives a one-response message (enabled only when using interrupts).

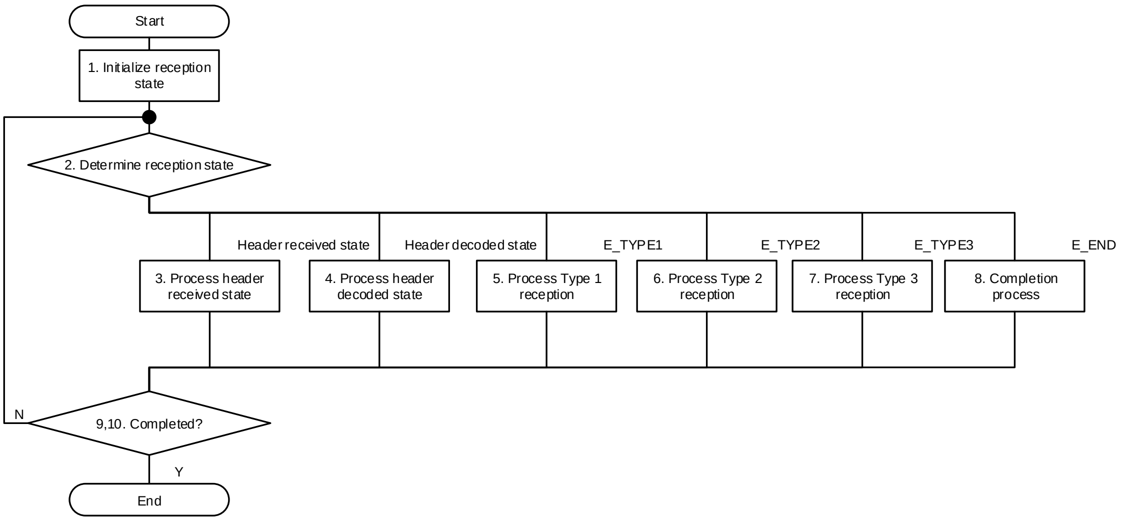
Header r_sa_uart_control_register.h

Argument saic_num:
SAIC number
rx_buffer[]:
Receive buffer
p_rx_length:
Reception size

Global Variable g_uart_saic_data_tbl[]:
Global variable to store SAIC information
g_uart_serial_data_tbl:
UART table
gs_bit_tbl[]:
Power-of-two data table constant
gs_adc_1shot:
ADC 1Shot acquire flag: 1 = ADC 1Shot acquired, 0 = Other
g_uart_rx_end_flag:
UART reception completion flag variable
gs_uart_negotiation_timeout:
Communication negotiation timeout value

Return Value saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM

Description Error processing branches have been omitted from flowchart.


Start
1. Initialize reception state
2. Determine reception state
Header received state
Header decoded state
E_TYPE1
E_TYPE2
E_TYPE3
E_END
3. Process header received state
4. Process header decoded state
5. Process Type 1 reception
6. Process Type 2 reception
7. Process Type 3 reception
8. Completion process
N
9,10. Completed?
Y
End

1. Sets reception state to header received state.
2. Determines reception state.
3. Header received state processing
If more than one packet is received, sets reception state to header decoded state.
4. Header decoded state processing
Analyzes header from receieved datat and sets reception state to Type 1, 2 or 3.
5. Type 1 reception processing
Sets reception state to END state when Type 1 format reception is completed.
6. Type 2 reception processing
Sets reception state to END state when Type 2 format reception is completed.
7. Type 3 reception processing
Sets reception state to END state when Type 3 format reception is completed.
8. Completion processing
Sets completion flag.
9. Continues loop until reception is completed (completion flag is set).
10. If negotiation timeout or deadlock occurs, returns D_SAIC_ERR_COM and goes to end.

# 6. Flash Memory Control Function Definitions

## 6.1 API Function Specifications

### 6.1.1 [UART/SPI] SAIC101 command function

saic_status_t R_SAIC_UART_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

saic_status_t R_SAIC_SPI_SAIC101
(uint8_t saic_num, e_saic101_func_t e_func, saic_data_t *data, uint16_t length)

Outline
SAIC101 command function. Process branches to one of the following based on e_func argument: register burst-read/write function/flash memory function.

Header
-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
e_func:
Specify SAIC101 functions
*data:
Data buffer pointer
length:
Data length

Global Variable
-For UART
g_uart_saic_data_tbl_size:
Number of stored SAIC information entries
g_uart_saic_data_tbl[]:
Global variable to store SAIC information
- For SPI
g_spi_saic_data_tbl_size:
Number of stored SAIC information entries
g_spi_saic_data_tbl[]:
Global variable to store SAIC information

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM or D_SAIC_ERR_VERIFY

Description
1. If argument error occurs, returns D_SAIC_ERR_COM and goes to end.
2. Calls internal functions corresponding to the specified processes.
3. If internal function fails, returns corresponding error and goes to end.

## 6.2 Internal Function Specifications

### 6.2.1 [UART/SPI] Read flash data function

static saic_status_t r_saic_uart_flash_read(uint8_t saic_num, saic_data_t *data, uint16_t length)

static saic_status_t r_saic_spi_flash_read(uint8_t saic_num, saic_data_t *data, uint16_t length)

Outline
Read flash data function. Issues [Flash (Burst) Read] commands.

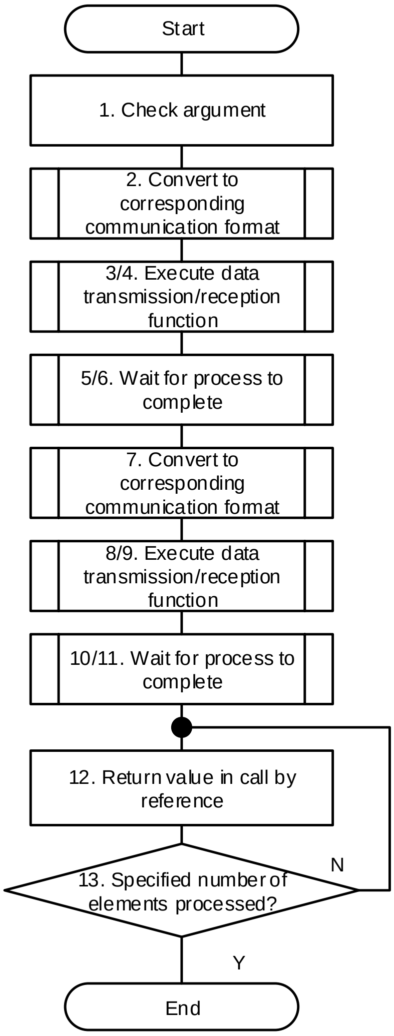
Header
-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
*data:
Data buffer pointer
length:
Data length

Global Variable
-For UART
g_uart_saic_data_tbl[]:
Global variable to store SAIC information
gs_bit_tbl[]:
Power-of-two data table constant
g_uart_rx_end_flag:
UART reception completion flag variable
gs_uart_half_bit_time:
Stop bit completion counter
g_uart_serial_data_tbl:
UART table
- For SPI
g_spi_saic_data_tbl[]:
Global variable to store SAIC information
g_spi_3us_nop_cnt:
Counter value of number of loops for delay time between Flash (Burst) Read commands (at least 3us)

Return Value
saic_status_t
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
Error processing branches have been omitted from flowchart.


Start
1. Check argument
2. Convert to corresponding communication format
3/4. Execute data transmission/reception function
5/6. Wait for process to complete
7. Convert to corresponding communication format
8/9. Execute data transmission/reception function
10/11. Wait for process to complete
12. Return value in call by reference
13. Specified number of elements processed?
N
Y
End

1. If error occurs, returns corresponding error and goes to end.

- For UART

If argument error occurs, returns SAIC_ERR_PARAM.

If function pointer is NULL, returns D_SAIC_ERR_COM.

- For SPI

If argument error occurs, returns SAIC_ERR_PARAM.

2. Converts to communication format (2-byte) based on specified address.

3. Executes data transmission/reception function, transmits/receives command 1.

-For UART

Executes command  transmission & response reception function.

-For SPI

Executes data transmission/reception execution function.

4. If data transmission/reception execution function fails, returns corresponding error and goes to end.

5. Waits for completion of data transmission/reception execution function processing.

- For UART

Executes UART receive data packet analysis function.

- For SPI

When INT pin is used, waits for INT interrupt.

When INT is not used (polling is used), waits at least 3us.

6. If the wait process fails before processing is completed, returns corresponding error and goes to end.

7. Converts to corresponding communication format

8. Executes data transmission/reception function, transmits/receives command 2.

-For UART

Executes command transmission & response reception function.

-For SPI

Executes data transmission/reception execution function.

9. If data transmission/reception function fails, returns corresponding error and goes to end.

10.  Waits for Flash (Burst) Read process to complete.

-For UART

Executes UART receive data packet analysis function.

11. If the wait process fails before processing is completed, returns corresponding error and goes to end.

12. Stores address in argument *data->address and read value in argument *data->data.

13. Repeats step 12 for the specified number of elements in length argument while incrementing the address.

### 6.2.2 [UART/SPI] Write flash data function

static saic_status_t r_saic_uart_flash_write(uint8_t saic_num, saic_data_t *data, uint16_t num)

static saic_status_t r_saic_spi_flash_write(uint8_t saic_num, saic_data_t *data, uint16_t num)

Outline

Write flash data function. All address other than 01H and 1FH can be written to. Issues [Flash Write] commands.

Header

-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument

saic_num:
SAIC number
*data:
Data buffer pointer
num:
Number of data units

Global Variable

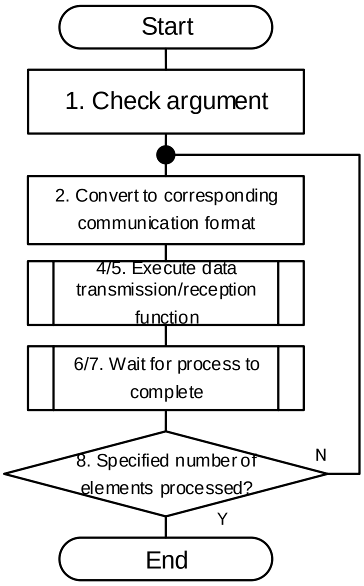
- For UART
g_uart_saic_data_tbl[]:
Global variable to store SAIC information
gs_bit_tbl[]:
Power-of-two data table constant
g_uart_rx_end_flag:
UART reception completion flag variable
gs_uart_half_bit_time:
Stop bit completion counter
g_uart_serial_data_tbl:
UART table
- For SPI
None

Return Value

saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description

Error processing branches have been omitted from flowchart.


Start
1. Check argument
2. Convert to corresponding communication format
4/5. Execute data transmission/reception function
6/7. Wait for process to complete
8. Specified number of elements processed?
N
Y
End

1. If error occurs, returns corresponding error and goes to end.
- For SPI
If argument error occurs, returns SAIC_ERR_PARAM.
- For UART
If argument error occurs, returns SAIC_ERR_PARAM.
If function pointer is NULL, returns D_SAIC_ERR_COM.
2. If specified address is 0x01H or 0x1F, returns D_SAIC_ERR_PARAM and goes to end.
3. Converts to communication format based on specified address and data.
4. Executes data transmission/reception execution function
-For SPI
Executes data transmission/reception execution function.
-For UART
Executes command transmission & response reception function.
5. If transmission/reception function fails, returns corresponding error and goes to end.
6. Waits for completion of Flash Write processing.
- For SPI control: executes polling monitoring function, and either waits for INT interrupt or for FWIP flag olling.
- For UART
Executes UART receive data packet analysis function.
7. If the wait process fails before processing is completed, returns corresponding error and goes to end.
8. Repeats steps 2 to 7 for the specified number of elements.

### 6.2.3 [UART/SPI] Erase all flash data function

static saic_status_t r_saic_uart_flash_all_erase(uint8_t saic_num)

static saic_status_t r_saic_spi_flash_all_erase(uint8_t saic_num)

Outline

Erase all flash data function. Issues [Flash All Erase] commands.

Header

-For UART
  r_sa_uart_control_register.h
- For SPI
  r_sa_spi_control_register.h

Argument

saic_num:
  SAIC number

Global Variable

-FOR UART
  g_uart_saic_data_tbl[]:
    Global variable to store SAIC information
  gs_bit_tbl[]:
    Power-of-two data table constant
  g_uart_rx_end_flag:
    UART reception completion flag variable
  gs_uart_half_bit_time:
    Stop bit completion counter
  g_uart_serial_data_tbl:
    UART table
- For SPI
  None

Return Value

saic_status_t:
  D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description

1. If error occurs, returns corresponding error and goes to end.
  -For UART
    If function pointer is NULL, returns D_SAIC_ERR_COM.
2. Generates Flash All Erase command (2 bytes)
3. Executes data transmission/reception function, transmits command.
  -For UART
    Executes command transmission & response reception function.
    Transmits/receives 2 bytes.
  -For SPI
    Executes data transmission/reception execution function.
    Transmits/receives second byte after completion of first byte transmission/reception.
4. If data transmission/reception function fails, returns corresponding error and goes to end.
5. Waits for completion of Flash All Erase processing.
  - For UART
  Executes UART receive data packet analysis function.
  - For SPI
  Executes polling monitoring function, and either waits for INT interrupt or for FAEIP flag polling.
6. If the wait process fails before processing is completed, returns corresponding error and goes to end.

### 6.2.4 [UART/SPI] Write-verify flash memory data function

saic_status_t r_saic_uart_flash_write_verify(uint8_t saic_num, saic_data_t data[], uint16_t num)

saic_status_t r_saic_spi_flash_write_verify(uint8_t saic_num, saic_data_t data[], uint16_t num)

Outline
Write-verify flash memory data function. Executes erase all data process in function when necessary.

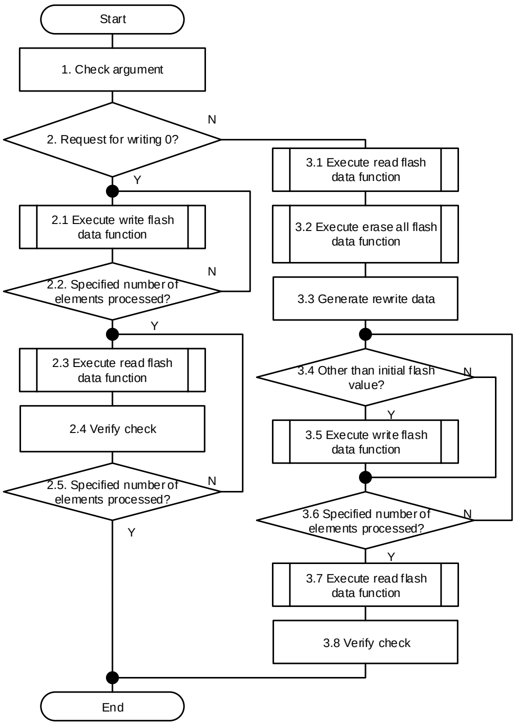
Header
-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
data[]:
Data buffer pointer
num:
Number of data units

Global Variable
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM or D_SAIC_ERR_VERIFY

Description
Error processing branches have been omitted from flowchart.


Start
1. Check argument
2. Request for writing 0?
N
Y
2.1 Execute write flash data function
2.2. Specified number of elements processed?
N
Y
2.3 Execute read flash data function
2.4 Verify check
2.5. Specified number of elements processed?
N
Y
3.1 Execute read flash data function
3.2 Execute erase all flash data function
3.3 Generate rewrite data
3.4 Other than initial flash value?
N
Y
3.5 Execute write flash data function
3.6 Specified number of elements processed?
N
Y
3.7 Execute read flash data function
3.8 Verify check
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Checks for a request for writing a value other than 0.
- Request for writing 0 only:
2.1 Executes write flash data function corresponding to the specified address.
If write flash data function fails, returns corresponding error and goes to end.
2.2 Executes step 2.1 for the specified number of elements by argument num.
2.3 Executes read flash data function to address written to in previous steps.
If read flash data function fails, returns corresponding error and goes to end.
2.4 Executes verify check; if verify error occurs, returns D_SAIC_ERR_VERIFY.
2.5 Repeats steps 2.3 to 2.4 for the specified number of elements in argument num.
- Request for writing 1:
3.1 Executes read flash data function, reads and stores all flash memory data.
If read flash data function fails, returns corresponding error and goes to end.
3.2 Executes erase all flash data function, erases all flash memory data.
If erase all flash data function fails, returns corresponding error and goes to end.
3.3 Writes data only to the target write addresses from all data read from the flash memory.
3.4 If write data is initial flash value 0xFF, goes to step 3.6.
3.5 Executes write flash data function to corresponding specified address.
If write flash data function fails, returns corresponding error and goes to end.
3.6 Repeats steps 3.4 and 3.5 for all addresses.
3.7 Executes read flash data function; reads and stores all memory data.
If read flash data function fails, returns corresponding error and goes to end.
3.8 Verify checks write and read data.
If verify error occurs, returns D_SAIC_ERR_VERIFY and goes to end.

### 6.2.5 [UART/SPI] Flash shadow area copy function

static saic_status_t r_saic_uart_all_flash_to_reg(uint8_t saic_num)

static saic_status_t r_saic_spi_all_flash_to_reg(uint8_t saic_num)

Outline

Flash shadow area copy function. Copies flash shadow area to register. Issues [Register All Write from Flash] commands.

Header

-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument

saic_num:
SAIC number

Global Variable

- For UART
g_uart_saic_data_tbl[]:
Global variable to store SAIC information
gs_bit_tbl[]:
Power-of-two data table constant
g_uart_rx_end_flag:
UART reception completion flag variable
gs_uart_half_bit_time:
Stop bit completion counter
g_uart_serial_data_tbl:
UART table
- For SPI
None

Return Value

saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description

1. If error occurs, returns corresponding error and goes to end.
-For UART
If function pointer is NULL, returns D_SAIC_ERR_COM.
2. Generates Register All Write from Flash command.
3. Executes data transmission/receive execution function, transmits command.
- For UART: executes command transmission & response reception function
- For SPI: executes data transmission/reception execution function.
4. If transmission/reception function fails, returns corresponding error and goes to end.
5. Waits for completion of Register All Write from Flash processing.
-For UART
Executes UART receive data packet analysis function.
- For SPI
Executes polling monitoring function, and either waits for INT interrupt or for RAWIP flag polling.
6. If the wait process fails before processing is completed, returns an error and goes to end.

### 6.2.6 [UART/SPI] Flash system setting copy function

static saic_status_t r_saic_uart_buffer_refresh(uint8_t saic_num)

static saic_status_t r_saic_spi_buffer_refresh(uint8_t saic_num)

Outline
Flash system setting copy function. Copies flash system setting to the buffer. Issues [Buffer Refresh] commands.

Header
-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number

Global Variable
- For UART
g_uart_saic_data_tbl[]:
Global variable to store SAIC information
gs_bit_tbl[]:
Power-of-two data table constant
g_uart_rx_end_flag:
UART reception completion flag variable
gs_uart_half_bit_time:
Stop bit completion counter
g_uart_serial_data_tbl:
UART table
- For SPI
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
1. If error occurs returns corresponding error and goes to end.
-For UART
If function pointer is NULL, returns D_SAIC_ERR_COM.
2. Generates Buffer Refresh command.
3. Executes data transmission/receive execution function, transmits command.
- For UART: executes command transmission & response reception function
- For SPI: executes data transmission/reception execution function.
4. If transmission/reception function fails, returns corresponding error and goes to end.
5. Waits for completion of Buffer Refresh processing.
-For UART
Executes UART receive data packet analysis function.
- For SPI
Executes polling monitoring function, and either waits for INT interrupt or for RAWIP flag polling.
6. If the wait process fails before processing is completed, returns corresponding error and goes to end.

### 6.2.7 [SPI] INTFLAG register FR bit acquisition function

static saic_status_t r_saic_spi_flash_read_fr_bit(uint8_t saic_num, uint8_t *fr_bit)

| | |
|---|---|
| Outline | INTFLAG register FR bit acquisition function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*fr_bit:<br>FR bit return value |
| Global Variable | None |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | 1. Executes read register bytes function to INTFLAG register and reads data.<br>2. Returns FR bit value of read value to argument pointer in call by reference. |

### 6.2.8 [SPI] INTFLAG register FW bit acquisition function

static saic_status_t r_saic_spi_flash_read_fw_bit(uint8_t saic_num, uint8_t *fw_bit)

| | |
|---|---|
| Outline | INTFLAG register FW bit acquisition function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*fw_bit:<br>FW bit return value |
| Global Variable | None |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | 1. Executes read register bytes function to INTFLAG register and reads data.<br>2. Returns FW bit value of read value to argument pointer in call by reference. |

### 6.2.9 [SPI] INTFLAG register FAE bit acquisition function

static saic_status_t r_saic_spi_flash_read_fae_bit(uint8_t saic_num, uint8_t *fae_bit)

| | |
|---|---|
| Outline | INTFLAG register FAE bit acquisition function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*fae_bit:<br>FAE bit return value |
| Global Variable | None |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | 1. Executes read register bytes function to INTFLAG register and reads data.<br>2. Returns FAE bit value of read value to argument pointer in call by reference. |

### 6.2.10 [SPI] INTFLAG register RAW bit acquisition function

static saic_status_t r_saic_spi_flash_read_raw_bit(uint8_t saic_num, uint8_t *raw_bit)

| | |
|---|---|
| Outline | INTFLAG register RAW bit acquisition function |
| Header | r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>*raw_bit:<br>RAW bit return value |
| Global Variable | None |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | 1. Executes read register bytes function to INTFLAG register and reads data.<br>2. Returns RAW bit value of read value to argument pointer in call by reference. |

### 6.2.11 [SPI] STATUS register FWIP bit acquisition function

static saic_status_t r_saic_spi_flash_read_fwip_bit(uint8_t saic_num, uint8_t *fwip_bit)

Outline
STATUS register FWIP bit acquisition function

Header
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
*fwip_bit:
FWIP bit return value

Global Variable
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
1. Executes read register bytes function to STATUS register and reads data.
2. Returns FWIP bit value of read value to argument pointer in call by reference.

### 6.2.12 [SPI] STATUS register FAEIP bit acquisition function

static saic_status_t r_saic_spi_flash_read_faeip_bit(uint8_t saic_num, uint8_t *faeip_bit)

Outline
STATUS register FAEIP bit acquisition function

Header
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
*faeip_bit:
FAEIP bit return value

Global Variable
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
1. Executes read register bytes function to STATUS register and reads data.
2. Returns FAEIP bit value of read value to argument pointer in call by reference.

### 6.2.13 [SPI] STATUS register RAWIP bit acquisition function

static saic_status_t r_saic_spi_flash_read_rawip_bit(uint8_t saic_num, uint8_t *rawip_bit)

Outline
STATUS register RAWIP bit acquisition function

Header
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
*rawip_bit:
RAWIP bit return value

Global Variable
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description
1. Executes read register bytes function to STATUS register and reads data.
2. Returns RAWIP bit value of read value to argument pointer in call by reference.

## 6.3 System Function Specifications

### 6.3.1 [UART/SPI] Write flash data address 01H function

saic_status_t R_SAIC_UART_FLASH_WRITE_01H(uint8_t saic_num, uint8_t data)

saic_status_t R_SAIC_SPI_FLASH_WRITE_01H(uint8_t saic_num, uint8_t data)

| | |
|---|---|
| Outline | Write flash data address 01H function. Implements settings for power supply. |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>data:<br>Write data |
| Global Variable | -For UART<br>g_uart_saic_data_tbl[]:<br>Global variable to store SAIC information<br>gs_bit_tbl[]:<br>Power-of-two data table constant<br>g_uart_rx_end_flag:<br>UART reception completion flag variable<br>gs_uart_half_bit_time:<br>Stop bit completion counter<br>g_uart_serial_data_tbl:<br>UART table<br>- For SPI<br>None |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM |
| Description | 1. If error occurs, returns corresponding error and goes to end.<br>-For UART<br>If function pointer is NULL, returns D_SAIC_ERR_COM.<br>2. If write data specifies setting 1 to a bit at address 01H which cannot be rewritten (bits 7, 6, 3, and 2), returns D_SAIC_ERR_PARAM and goes to end.<br>3. Converts to communication format based on specified address and write data.<br>4. Executes data transmission/reception function.<br>- For UART<br>Executes command transmission & response reception function.<br>- For SPI<br>Executes data transmission/reception execution function.<br><br>5. If data transmission/reception function fails, returns corresponding error and goes to end.<br>6. Waits for completion of flash write processing.<br>- For SPI: Executes polling monitoring function; waits for INT interrupt or FWIP flag polling.<br>- For UART: Executes UART receive data packet analysis function.<br>7. If the wait process fails before processing is completed, returns corresponding error and goes to end. |

### 6.3.2 [UART/SPI] Write flash data address 1FH function

saic_status_t R_SAIC_UART_FLASH_WRITE_1FH(uint8_t saic_num, uint8_t data)

saic_status_t R_SAIC_SPI_FLASH_WRITE_1FH(uint8_t saic_num, uint8_t data)

Outline

Write flash data address 1FH function. Implements settings for start-up operations.

Header

-For UART

r_sa_uart_control_register.h

- For SPI

r_sa_spi_control_register.h

Argument

saic_num:

SAIC number

data:

Write data

Global Variable

-For UART

g_uart_saic_data_tbl[]:

Global variable to store SAIC information

gs_bit_tbl[]:

Power-of-two data table constant

g_uart_rx_end_flag:

UART reception completion flag variable

gs_uart_half_bit_time:

Stop bit completion counter

g_uart_serial_data_tbl:

UART table

- For SPI

None

Return Value

saic_status_t:

D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM

Description

1. If error occurs, returns corresponding error and goes to end.
 - For UART
  If function pointer is NULL, returns D_SAIC_ERR_COM.
2. If write data specifies setting 1 to a bit at address 1FH which cannot be rewritten (bits 7, 6, 5, 3, and 2), returns D_SAIC_ERR_PARAM and goes to end.
3. Converts to communication format based on specified address and write data.
4. Executes data transmission/reception function.
 - For UART
  Executes command transmission & response reception function.
 - For SPI
  Executes data transmission/reception execution function.
5. If data transmission/reception function fails, returns corresponding error and goes to end.
6. Waits for completion of flash write processing.
 - For SPI: Executes polling monitoring function; waits for INT interrupt or FWIP flag polling.
 - For UART: Executes UART receive data packet analysis function.
7. If the wait process fails before processing is completed, returns corresponding error and goes to end.

# 7. ADC Definitions

## 7.1 API Function Specifications

### 7.1.1 [UART/SPI] A/D conversion start process function

saic_status_t R_SAIC_UART_ADC_Start(uint8_t saic_num)

saic_status_t R_SAIC_SPI_ADC_Start(uint8_t saic_num)

| | |
|---|---|
| Outline | A/D conversion start process function |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number |
| Global Variable | -For UART<br>g_uart_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>- For SPI<br>g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM |
| Description | 1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.<br>2. Generates saic_data_t variable array in function and assigns it as address variable of ADCCNT register.<br>3. Executes read register bytes function.<br>4. If read register bytes function fails, returns corresponding error and goes to end.<br>5. Assigns 1 to ADSTART bit of read value and executes write register bytes function.<br>6. If write register bytes function fails, returns corresponding error and goes to end. |

### 7.1.2 [UART/SPI] A/D conversion stop process function

saic_status_t R_SAIC_UART_ADC_Stop(uint8_t saic_num)

saic_status_t R_SAIC_SPI_ADC_Stop(uint8_t saic_num)

Outline A/D conversion stop process function

Header -For UART

r_sa_uart_control_register.h

- For SPI

r_sa_spi_control_register.h

Argument saic_num:

SAIC number

Global Variable - For UART

g_uart_saic_data_tbl_size:

Number of stored SAIC information entries

gs_uart_half_bit_time:

Stop bit completion counter

- For SPI

g_spi_saic_data_tbl_size:

Number of stored SAIC information entries

Return Value saic_status_t:

D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description 1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.

2. Generates saic_data_t variable array in function and assigns it as address variable of ADCCNT register.

3. Executes read register bytes function.

4. If read register bytes function fails, returns corresponding error and goes to end.

5. Assigns 0 to ADSTART bit of read value and executes write register bytes function.

- For UART

Reads the written address twice in the read register bytes function.

If read register bytes function fails the second time, returns corresponding error and goes to end.

If ADSTART bit of the second read value is 1, returns D_SAIC_ERR_COM and goes to end.

6. If write register bytes function fails, returns corresponding error and goes to end.

### 7.1.3 [UART/SPI] A/D converter register initial setup function

saic_status_t R_SAIC_UART_ADC_InitRegSet(uint8_t saic_num, saic101_adc_t adc_setting[])

saic_status_t R_SAIC_SPI_ADC_InitRegSet(uint8_t saic_num, saic101_adc_t adc_setting[])

Outline
A/D converter register initial setup function

Header
-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
adc_setting[]:
ADC setting information storage structure

Global Variable
- For UART
g_uart_saic_data_tbl_size:
Number of stored SAIC information entries
gs_bit_tbl[]:
Power-of-two data table constant
- For SPI
g_spi_saic_data_tbl_size:
Number of stored SAIC information entries
gs_bit_tbl[]:
Power-of-two data table constant

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description
1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Converts register setting values based on adc_setting argument.
3. Executes burst write function.
4. If function fails, returns corresponding error and goes to end.

### 7.1.4 [UART/SPI] A/D-converted value acquisition function (for multiple channels, multiple times for each channel)

```
saic_status_t R_SAIC_UART_ADC_GetResult
    (uint8_t saic_num, uni_adcc_t adcc[], uint16_t adc_value[], uint16_t count)

saic_status_t R_SAIC_SPI_ADC_GetResult
    (uint8_t saic_num, uni_adcc_t adcc[], uint16_t adc_value[], uint16_t count)
```

Outline
A/D-converter value acquisition function. Acquires data from multiple channels, multiple times for each channel.

Header
-For UART

r_sa_uart_control_register.h

- For SPI

r_sa_spi_control_register.h

Argument
saic_num:
SAIC number

adcc[]:
ADCC register value storage buffer. Requires storage area equivalent to number of A/D conversions.

adc_value[]:
ADC value storage buffer. Requires storage area equivalent to number of A/D conversions

count:
Total number of A/D conversions. (total number of valid A/D conversions for all channels).

Global Variable
-For UART

g_uart_saic_data_tbl_size:
Number of stored SAIC information entries

g_uart_saic_data_tbl[]:
Global variable to store SAIC information

gs_bit_tbl[]:
Power-of-two data table constant

g_uart_rx_end_flag:
UART reception completion flag variable

g_uart_serial_data_tbl:
Global variable to store serial module information.

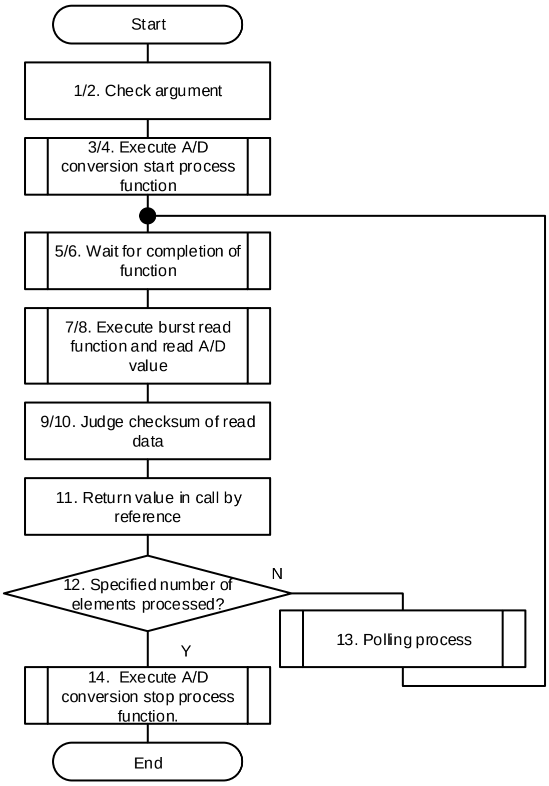
For SPI

g_spi_saic_data_tbl_size:
Number of stored SAIC information entries

g_spi_saic_data_tbl[]:
Global variable to store SAIC information

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description
Error processing branches have been omitted from flowchart.


Start
1/2. Check argument
3/4. Execute A/D conversion start process function
5/6. Wait for completion of function
7/8. Execute burst read function and read A/D value
9/10. Judge checksum of read data
11. Return value in call by reference
12. Specified number of elements processed?
N
Y
13. Polling process
14. Execute A/D conversion stop process function.
End

1. If error occurs, returns corresponding error and goes to end.
  -For SPI
    If argument error occurs, returns D_SAIC_ERR_PARAM.
  - For UART
    If argument error occurs, returns D_SAIC_ERR_PARAM.
      If function pointer is NULL, returns D_SAIC_ERR_COM.
2. If number of A/D in argument =0, returns SAIC_ERR_PARAM.
3. Executes A/D conversion start process function
  - For SPI: Execute A/D conversion start process function
  - For UART: Executes read register function and reads ADCCNT register.
      Assigns 1 to the ADSTART bit of the read value, then executes the write register function.
      Executes UART receive data packet analysis function (r_saic_uart_get_response) to retrieve response from write register.
4. If A/D conversion start process function fails, returns corresponding error and goes to end.
5. Waits for completion of A/D conversion start process function.
  - For SPI: If INT pin is used, waits for INT interrupt; if not (polling is used), ADCIP flag polling.
6. If polling monitoring function fails, returns corresponding error and goes to end.
  - SPI: If function fails, returns corresponding error and goes to end.
7. Executes read A/D value process, reads A/D-converted value.
  - For SPI: Executes burst read function and reads A/D value.
  - For UART: (to read A/D value) Executes UART receive data packet analysis function.
8. If read A/D value process fails, returns corresponding error and goes to end.
9. Judges checksum of read data.
10. If checksum error occurs, returns D_SAIC_ERR_COM.
11. Assigns adcc[] argument as the ADCC register value, and adc_value[] argument as the ADC value.
12. Repeats for the specified number of elements in the count argument.
13. Polling until completion of A/D conversion.
  - For SPI: If INT pin is used, goes to step 5.
    If INT pin is not used (polling is used), executes internal function read ADCIP bit function. Continues until ADCIP bit is 1. After function completes successfully, goes to step 5.
    If read ADCIP bits function fails or generates a deadlock, returns corresponding error and goes to end.
14. Executes A/Dconversion stop process function.

### 7.1.5 [UART/SPI] A/D-converted value acquisition function (for a single channel in 1Shot mode)

saic_status_t R_SAIC_UART_ADC_GetResult_1Shot
(uint8_t saic_num, e_adc_ch_t ch, uint16_t *adc_value)

saic_status_t R_SAIC_SPI_ADC_GetResult_1Shot
(uint8_t saic_num, e_adc_ch_t ch, uint16_t *adc_value)

| | |
|---|---|
| Outline | A/D-converted value acquisition function for a single channel, acquire data only once. |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>ch:<br>Target channel number for A/D conversion<br>*adc_value:<br>ADC value storage buffer |
| Global Variable | -For UART<br>g_uart_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>gs_bit_tbl[]:<br>Power-of-two data table constant<br>gs_adc_1shot:<br>ADC 1Shot acquisition flag: 1 = ADC 1Shot acquired, 0 = Other<br>- For SPI<br>g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>gs_bit_tbl[]:<br>Power-of-two data table constant |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM |
| Description | 1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.<br>2. Generates ADCCNT register setting data based on argument channel.<br>3. Executes write register bytes function, only enables input multiplexer.<br>4. If write register bytes function fails, returns corresponding error and goes to end.<br>5. Generates CHxCNT3 setting data based on argument channel.<br>6. Executes write register bytes function, sets number of A/D conversions for specified channel to 1.<br>7. If write register bytes function fails, returns corresponding error and goes to end.<br>8. A/D conversion and reception processing<br>- For SPI: Executes A/D conversion start process function<br>If A/D conversion start process function fails, returns corresponding error and goes to end.<br>Waits for completion of A/D conversion start process function.<br>If INT pin is used, waits for INT interrupt; if not (polling is used), ADCIP flag polling.<br>If polling fails while waiting for process to complete, returns corresponding error and goes to end.<br>Executes burst read function and reads A/D-converted value.<br>If burst read function fails, returns corresponding error and goes to end.<br>- For UART: Executes read register function to ADCCNT register and reads data.<br>Assigns 1 to ADSTART bit of read value.<br>Assigns 1 to gs_adc_1shot variable.<br>Executes command transmission & response reception function.<br>9. Judges checksum of read data.<br>10. If checksum error occurs, returns D_SAIC_ERR_COM and goes to end.<br>11. Assigns A/D conversion results to *adc_value argument. |

### 7.1.6 [UART] A/D-converted value received data acquisition function

saic_status_t R_SAIC_UART_ADC_GetReceive
(uint8_t saic_num, uni_adcc_t *adcc, uint16_t *adc_value)

Outline: A/D-converterd value received data acquisition function

Header: r_sa_uart_control_register.h

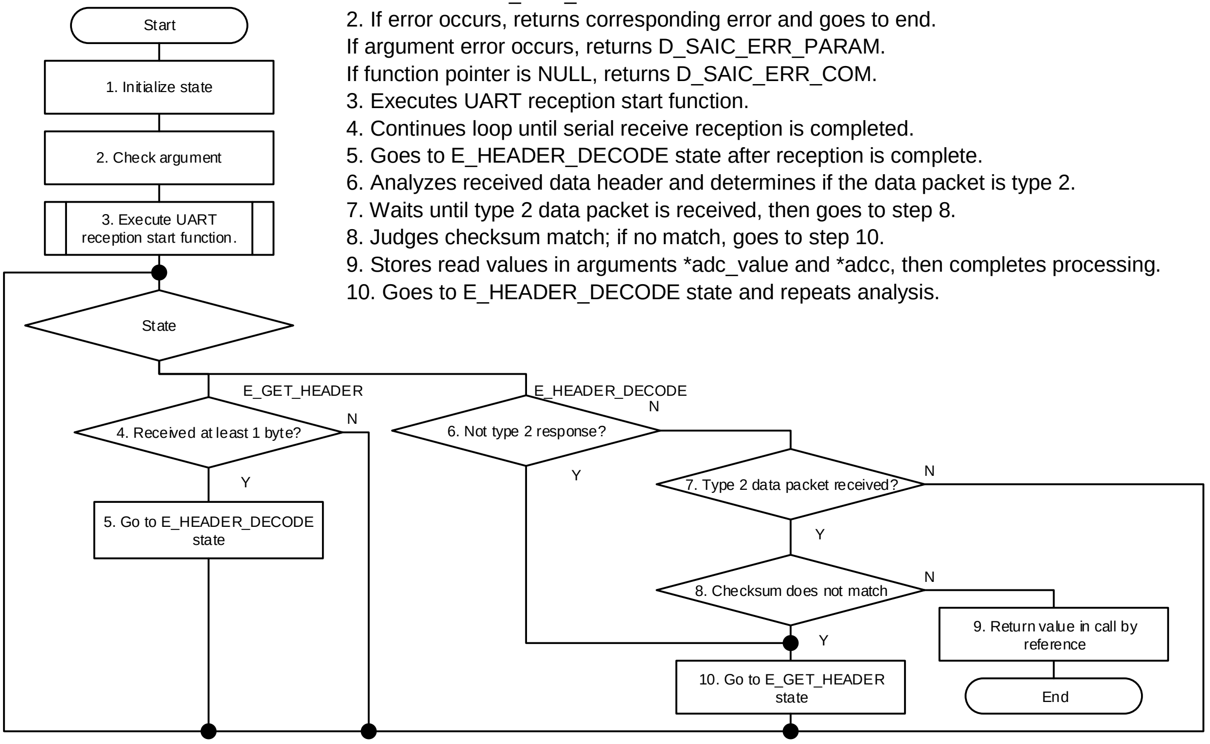
Argument:
saic_num:
SAIC number
*adcc:
ADCC register value storage buffer
*adc_value:
ADC value storage buffer

Global Variable:
g_uart_saic_data_tbl_size:
Number of stored SAIC information entries
g_uart_saic_data_tbl[]:
Global variable to store SAIC information
gs_bit_tbl[]:
Power-of-two data table constant
g_uart_saic_auto_adc_timeout:
Settling time value (8.192 ms) when OSR=2048
g_uart_rx_end_flag:
UART reception completion flag variable
g_uart_serial_data_tbl:
Global variable to store serial module information

Return Value:
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description: Error processing branches have been omitted from flowchart.

1. Sets state to E_GET_HEADER.
2. If error occurs, returns corresponding error and goes to end.
If argument error occurs, returns D_SAIC_ERR_PARAM.
If function pointer is NULL, returns D_SAIC_ERR_COM.
3. Executes UART reception start function.
4. Continues loop until serial receive reception is completed.
5. Goes to E_HEADER_DECODE state after reception is complete.
6. Analyzes received data header and determines if the data packet is type 2.
7. Waits until type 2 data packet is received, then goes to step 8.
8. Judges checksum match; if no match, goes to step 10.
9. Stores read values in arguments *adc_value and *adcc, then completes processing.
10. Goes to E_HEADER_DECODE state and repeats analysis.


Start
1. Initialize state
2. Check argument
3. Execute UART reception start function.
State
E_GET_HEADER
E_HEADER_DECODE
4. Received at least 1 byte?
N
Y
5. Go to E_HEADER_DECODE state
6. Not type 2 response?
N
Y
7. Type 2 data packet received?
N
Y
8. Checksum does not match
N
Y
9. Return value in call by reference
End
10. Go to E_GET_HEADER state

## 7.2 Internal Function Specifications

### 7.2.1 [UART/SPI] A/D-converted value checksum value judgement function

static saic_status_t r_saic_uart_adc_checksum(uint8_t adc_data[])

static saic_status_t r_saic_spi_adc_checksum(saic_data_t adc_data[])

Outline
A/D-converted value checksum value judgement function

Argument
adc_data[]:
First 3-bytes ADC address

Global Variable
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM

Description
1. Assigns three bytes to all unions from the saic_data argument.
2. Calculates checksum from ADC value.
3. Compares with checksum value of ADCC register.
4. If checksum value matches, returns D_SAIC_OK; if not, returns D_SAIC_ERR_COM and goes to end.

### 7.2.2 [SPI] INTFLAG register ADC bit acquisition function

static saic_status_t r_saic_spi_adc_read_adc_bit(uint8_t saic_num, uint8_t *adc_bit)

Outline
INTFLAG register ADC bit acquisition function

Header
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
*adc_bit:
ADC bit return value

Global Variable
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description
1. Executes read register bytes function to INTFLAG register and reads data.
2. Returns ADC bit value of read value to argument pointer in call by reference.

### 7.2.3 [SPI] STATUS register ADCIP bit acquisition function

static saic_status_t r_saic_spi_adc_read_adcip_bit(uint8_t saic_num, uint8_t *adcip_bit)

Outline
STATUS register ADCIP bit acquisition function

Header
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number
*adcip_bit:
ADCIP bit return value

Global Variable
None

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description
1. Executes read register bytes function to STATUS register and reads data.
2. Returns ADCIP bit value of read value to argument pointer in call by reference.

# 8. Power Supply Definitions

## 8.1 API Function Specifications

### 8.1.1 [UART/SPI] AREG ON setting function

saic_status_t R_SAIC_UART_AregOn(uint8_t saic_num)

saic_status_t R_SAIC_SPI_AregOn(uint8_t saic_num)

| | |
|---|---|
| Outline | AREG ON setting function |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number |
| Global Variable | - For UART<br>g_uart_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_uart_1800us_nop_cnt:<br>Counter value for AREG operation stabilization wait (at least 1800 us)<br>- For SPI<br>g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_spi_1800us_nop_cnt:<br>Counter value for AREG operation stabilization wait (at least 1800 us) |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM |
| Description | 1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.<br>2. Executes read register bytes function to specified address and reads data from CHIPCNT register.<br>3. If read fails, returns corresponding error and goes to end.<br>4. Generates transmission data from value read from CHIPCNT register.<br>First command: sets AREGPD bit to 0, SLP bit to 1<br>Second command: sets AREGPD bit to 0, SLP bit to 1 (same as first command)<br>Third command: sets SLP bit to 0<br>5. Executes write register bytes function and writes data.<br>6. If write fails, returns corresponding error and goes to end.<br>7. Executes NOP command for number of times indicated by counter value for AREG operation stabilization wait, waits at least 1800 us. |

### 8.1.2 [UART/SPI] AREG OFF setting function

saic_status_t R_SAIC_UART_AregOff(uint8_t saic_num)

saic_status_t R_SAIC_SPI_AregOff(uint8_t saic_num)

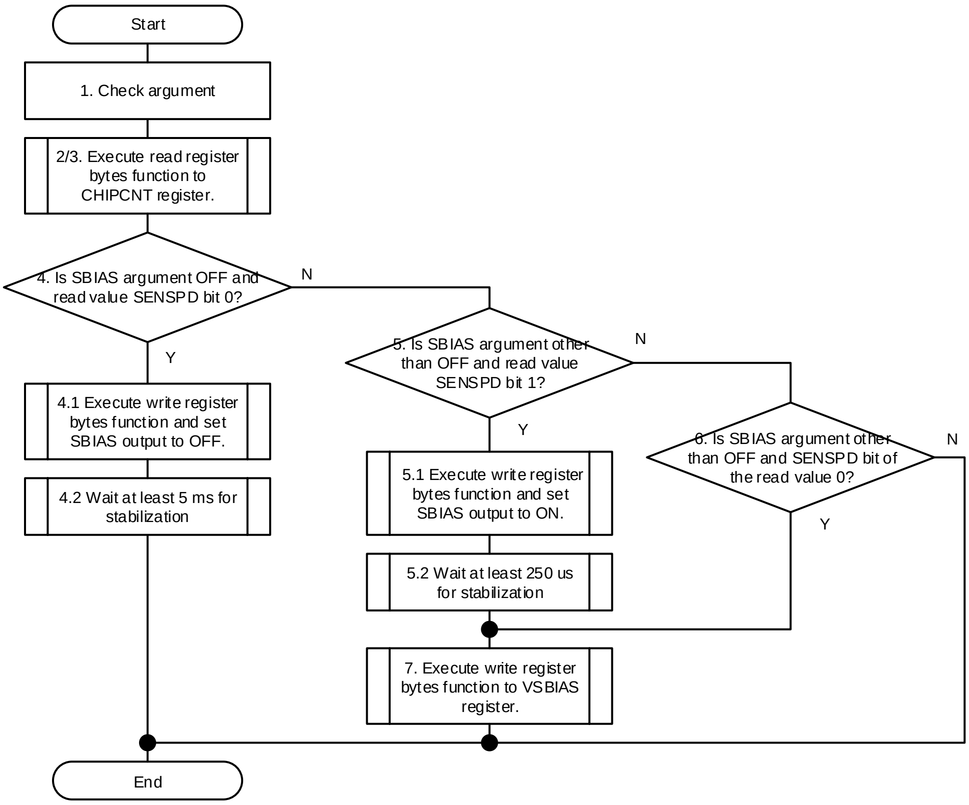
| | |
|---|---|
| Outline | AREG OFF setting function |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number |
| Global Variable | - For UART<br>g_uart_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>- For SPI<br>g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM |
| Description | 1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.<br>2. Executes read register bytes function to specified address and reads data from CHIPCNT register.<br>3. If read fails, returns corresponding error and goes to end.<br>4. Generates transmission data from value read from CHIPCNT register.<br>First command: sets AREGPD bit to 1, SLP bit to 1<br>Second command: sets AREGPD bit to 1, SLP bit to 1 (same as first command)<br>Third command: sets SLP bit to 0<br>5. Executes write register bytes function and writes data.<br>6. If write fails, returns corresponding error and goes to end. |

### 8.1.3 [UART/SPI] SBIAS register setting function

saic_status_t R_SAIC_UART_SbiasRegSet(uint8_t saic_num, e_sbias_t sbias)

saic_status_t R_SAIC_SPI_SbiasRegSet(uint8_t saic_num, e_sbias_t sbias)

| | |
|---|---|
| Outline | SBIAS register setting function |
| Header | - For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number<br>sbias:<br>SBIAS output voltage setting |
| Global Variable | - For UART<br>g_uart_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_uart_5ms_nop_cnt:<br>Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms)<br>g_uart_250us_nop_cnt:<br>Counter value for SBIAS operation stabilization wait (at least 250 us)<br>- For SPI<br>g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_spi_5ms_nop_cnt:<br>Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms)<br>g_spi_250us_nop_cnt:<br>Counter value for SBIAS operation stabilization wait (at least 250 us) |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM |
| Description | Error processing branches have been omitted from flowchart. |


Start
1. Check argument
2/3. Execute read register bytes function to CHIPCNT register.
4. Is SBIAS argument OFF and read value SENSPD bit 0?
N
Y
4.1 Execute write register bytes function and set SBIAS output to OFF.
4.2 Wait at least 5 ms for stabilization
5. Is SBIAS argument other than OFF and read value SENSPD bit 1?
N
Y
5.1 Execute write register bytes function and set SBIAS output to ON.
5.2 Wait at least 250 us for stabilization
6. Is SBIAS argument other than OFF and SENSPD bit of the read value 0?
N
Y
7. Execute write register bytes function to VSBIAS register.
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.

2. Executes read register bytes function to specified address and reads data from CHIPCNT register.

3. If read fails, returns corresponding error and goes to end.

4. If sbias argument is E_ADC_SBIAS_0p0 and the SENSPD bit read is 0:

4.1 Assigns 1 to SENSPD bit of the read value and executes write register function. Stops SBIAS operations.

4.2 Executes NOP command for number of times indicated by counter value for SBIAS stop and sleep mode operation stabilization wait, waits at least 5 ms, then goes to end.

5. If sbias argument is not E_ADC_SBIAS_0p0 and SENSPD bit of the read value is 1:

5.1 Assigns 0 to SENSPD bit of the read value and executes write register function. SBIAS normal operations.

5.2 Executes NOP command for number of times indicated by counter value for operation stabilization after SBIAS is set to ON. Waits at least 250 us, then goes to step 7.

6. If sbias argument is not E_ADC_SBIAS_0p0 and SENSPD bit of the read value is 0: goes to step 7.

In all other cases, goes to end.

7. Executes write register bytes function and writes sbias argument data to VSBIAS register.

### 8.1.4 [UART/SPI] SBIAS register acquisition function

saic_status_t R_SAIC_UART_SbiasRegGet(uint8_t saic_num, e_sbias_t *sbias)

saic_status_t R_SAIC_SPI_SbiasRegGet(uint8_t saic_num, e_sbias_t *sbias)

Outline
SBIAS register acquisition function

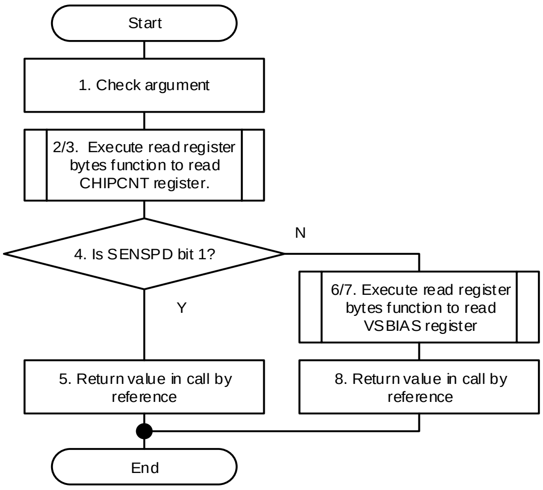
Header
- For UART
  r_sa_uart_control_register.h
- For SPI
  r_sa_spi_control_register.h

Argument
saic_num:
  SAIC number
*sbias:
  Return value for SBIAS output voltage setting

Global Variable
-For UART
  g_uart_saic_data_tbl_size:
    Number of stored SAIC information entries
- For SPI
  g_spi_saic_data_tbl_size:
    Number of stored SAIC information entries

Return Value
saic_status_t:
  D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description
Error processing branches have been omitted from flowchart.


Start
1. Check argument
2/3. Execute read register bytes function to read CHIPCNT register.
4. Is SENSPD bit 1?
N
Y
6/7. Execute read register bytes function to read VSBIAS register
5. Return value in call by reference
8. Return value in call by reference
End

1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Executes read register bytes function to specified address and reads data from CHIPCNT register.
3. If read fails, returns corresponding error and goes to end.
4. Judges SENSPD bit value.
5. If SENSPD bit value is 1, assigns E_ADC_SBIAS_0p0 to *sbias argument.
6. If SENSPD bit value is 0, executes read register bytes function and reads VSBIAS register data.
7. If read fails, returns corresponding error and goes to end.
8. Assigns read value to *sbias argument.

### 8.1.5 [UART/SPI] Sleep mode ON setting function

saic_status_t R_SAIC_UART_SleepModeOn(uint8_t saic_num)

saic_status_t R_SAIC_SPI_SleepModeOn(uint8_t saic_num)

| | |
|---|---|
| Outline | Sleep mode ON setting function |
| Header | -For UART<br>r_sa_uart_control_register.h<br>- For SPI<br>r_sa_spi_control_register.h |
| Argument | saic_num:<br>SAIC number |
| Global Variable | -For UART<br>g_uart_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_uart_5ms_nop_cnt:<br>Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms)<br>- For SPI<br>g_spi_saic_data_tbl_size:<br>Number of stored SAIC information entries<br>g_spi_5ms_nop_cnt:<br>Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms) |
| Return Value | saic_status_t:<br>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM |
| Description | 1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.<br>2. Executes read register bytes function to specified address and reads data from CHIPCNT register.<br>3. If read fails, returns corresponding error and goes to end.<br>4. Sets 1 to SLP bit of CHIPCNT register read in step 2.<br>5. Executes write register bytes function and writes to CHIPCNT register.<br>6. Waits at least 5 ms for stabilization. |

### 8.1.6 [UART/SPI] Sleep mode OFF setting function

saic_status_t R_SAIC_UART_SleepModeOff(uint8_t saic_num)

saic_status_t R_SAIC_SPI_SleepModeOff(uint8_t saic_num)

Outline
Sleep mode OFF setting function

Header
-For UART
r_sa_uart_control_register.h
- For SPI
r_sa_spi_control_register.h

Argument
saic_num:
SAIC number

Global Variable
-For UART
g_uart_saic_data_tbl_size:
Number of stored SAIC information entries
g_uart_270us_nop_cnt:
Wait for operation stabilization after exiting sleep mode (at least 270 us)
- For SPI
g_spi_saic_data_tbl_size:
Number of stored SAIC information entries
g_spi_820us_nop_cnt:
Wait for operation stabilization after exiting sleep mode (at least 820 us)

Return Value
saic_status_t:
D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM

Description
1. If argument error occurs, returns D_SAIC_ERR_PARAM and goes to end.
2. Executes read register bytes function to specified address and reads data from CHIPCNT register.
3. If read fails, returns corresponding error and goes to end.
4. Sets 0 to SLP bit of CHIPCNT register read in step 2.
5. Executes write register bytes function and writes to CHIPCNT register.
6. Waits for stabilization.
-For UART: at least 270 us
- For SPI: at least 820 us

# 9. Power Supply Settings

## 9.1 Power Supply Configurations

### 9.1.1 List of configurations

SAIC101 power-supply voltage supports three power supply configurations: 5.0V system power supply, 3.0V system power supply, and 3.0V supplied from the connected MCU.

However, applied voltage of 4.5 V (4.6 V for Configuration 2) to 5.5V is required when rewriting the SAIC101's built-in flash memory (flash memory programming). Table 9.1 shows the list of power supply configurations supported by this API. The term "normal operations" indicates operations not using flash memory programming.

**Table 9.1 Power supply configurations**

| Power Supply Configuration | Power-supply Voltage during operations | | Changes to register setting for normal operations | For Flash Memory Programming | |
|---|---|---|---|---|---|
| | SAIC101 | MCU | | Power supply configuration changes | Register setting changes |
| Configuration 1 | 3.3 to 5.5V<br>(same electric potential) | | Not required | Not required | Not required |
| Configuration 2 | 3.3～5.5V | 3.0V | Not required | **Required** | **Required** |
| Configuration 3 | 2.7 to 3.6V<br>(same electric potential) | | **Required** | **Required** | **Required** |

### 9.1.2 Power supply configuration 1 (normal operations)

Figure 9-1 shows the power supply configuration when SAIC101 and the MCU both operate in a range of 3.3V to 5.5V and have the same electric potential (as in Configuration 1 in Table 9.1).

- Power supply configuration for normal operations

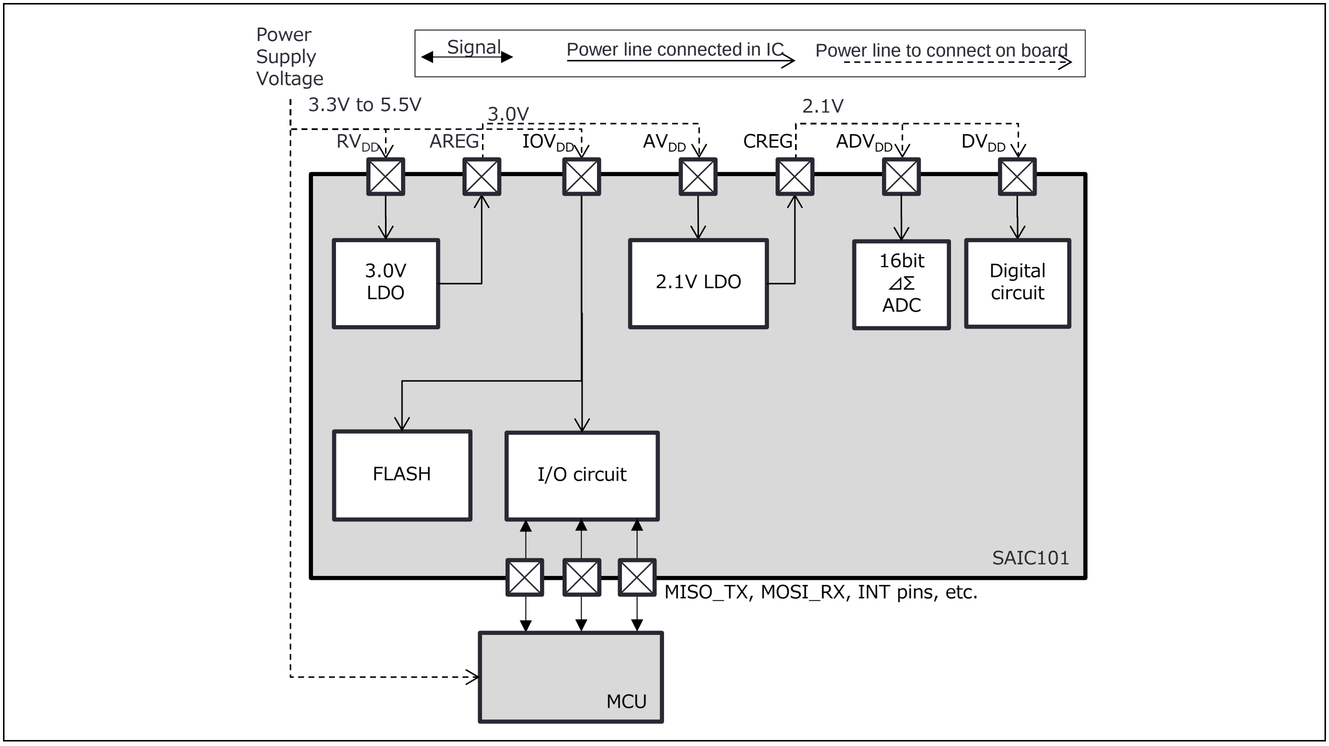


**Figure 9-1 Power supply configuration for normal operations**

- Software settings for normal operations: not required

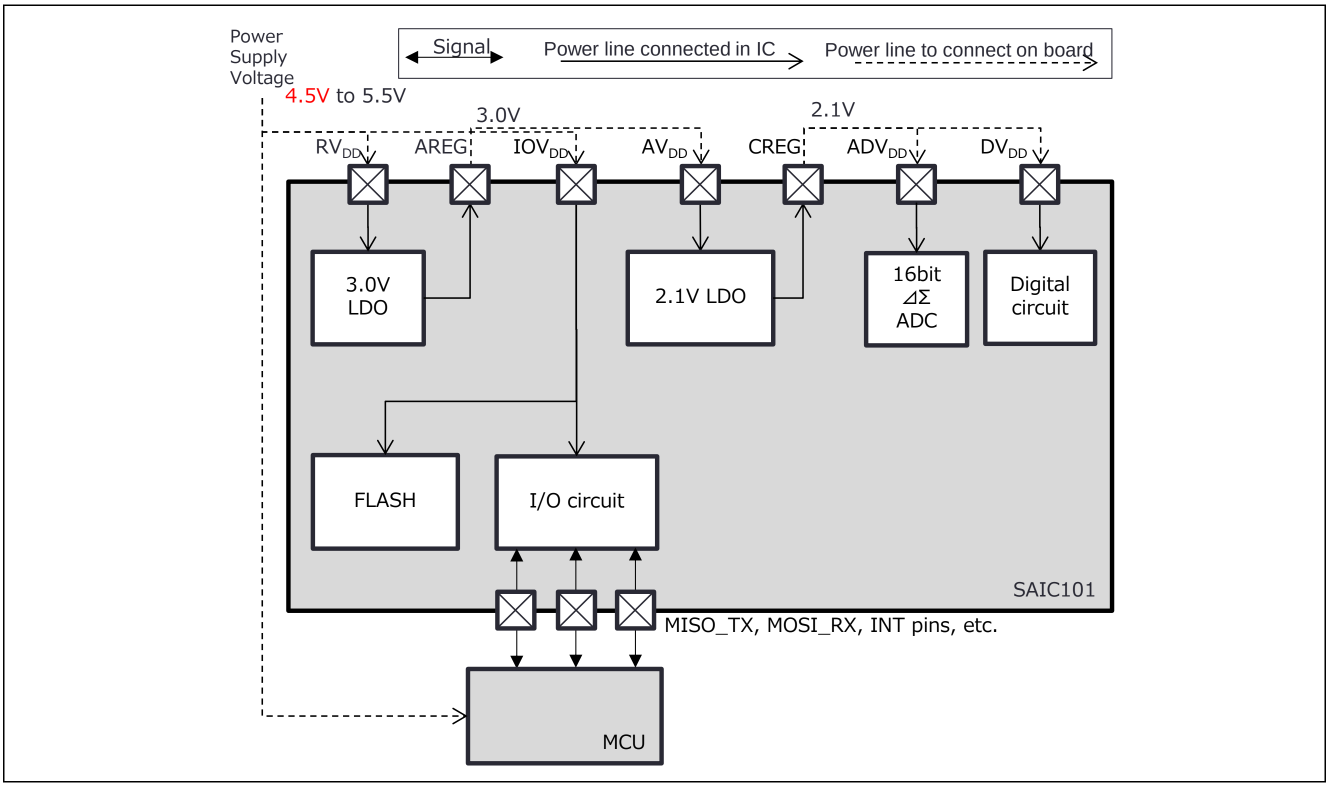
### 9.1.3 Power supply configuration 1 (for flash programming)

- **Figure 9-2 Power supply configuration for flash programming**

shows the power supply configuration when SAIC101 and the MCU both operate in a range of 3.3V to 5.5V and have the same electric potential (as in Configuration 1 in Table 9.1).

- Power supply configuration changes for flash programming

Apply 4.5V to5.5V when power supply is less than 4.5V.



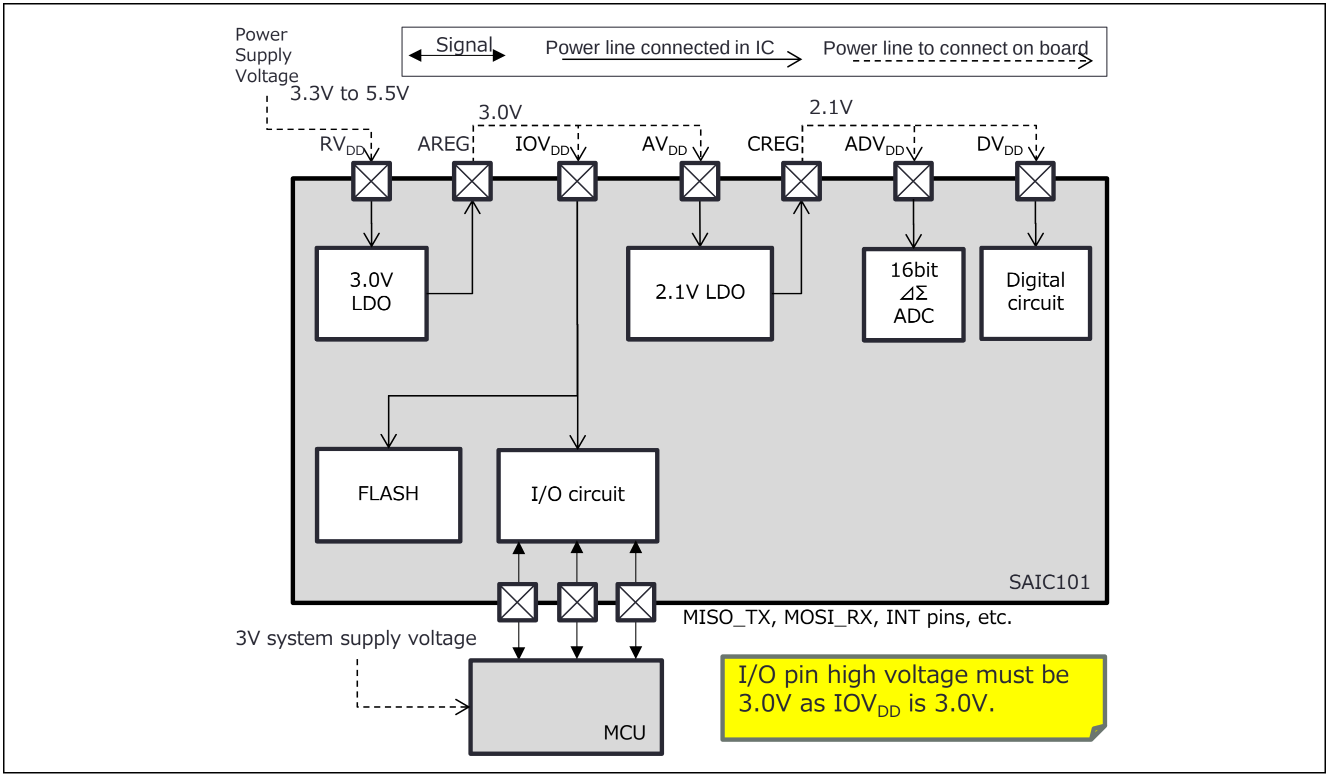
- **Figure 9-2 Power supply configuration for flash programming**

- Software settings for flash programming: not required

### 9.1.4 Power supply configuration 2 (normal operations)

Figre 9-3 shows the power supply configuration when SAIC101 operates in a range of 3.3V to 5.5V, the MCU operates at 3.0V with an electric potential difference (as in Configuration 2 in Table 9.1).

- Power supply configuration for normal operations



**Figre 9-3 Power supply configuration for normal operations**

- Software settings for normal operations: not required

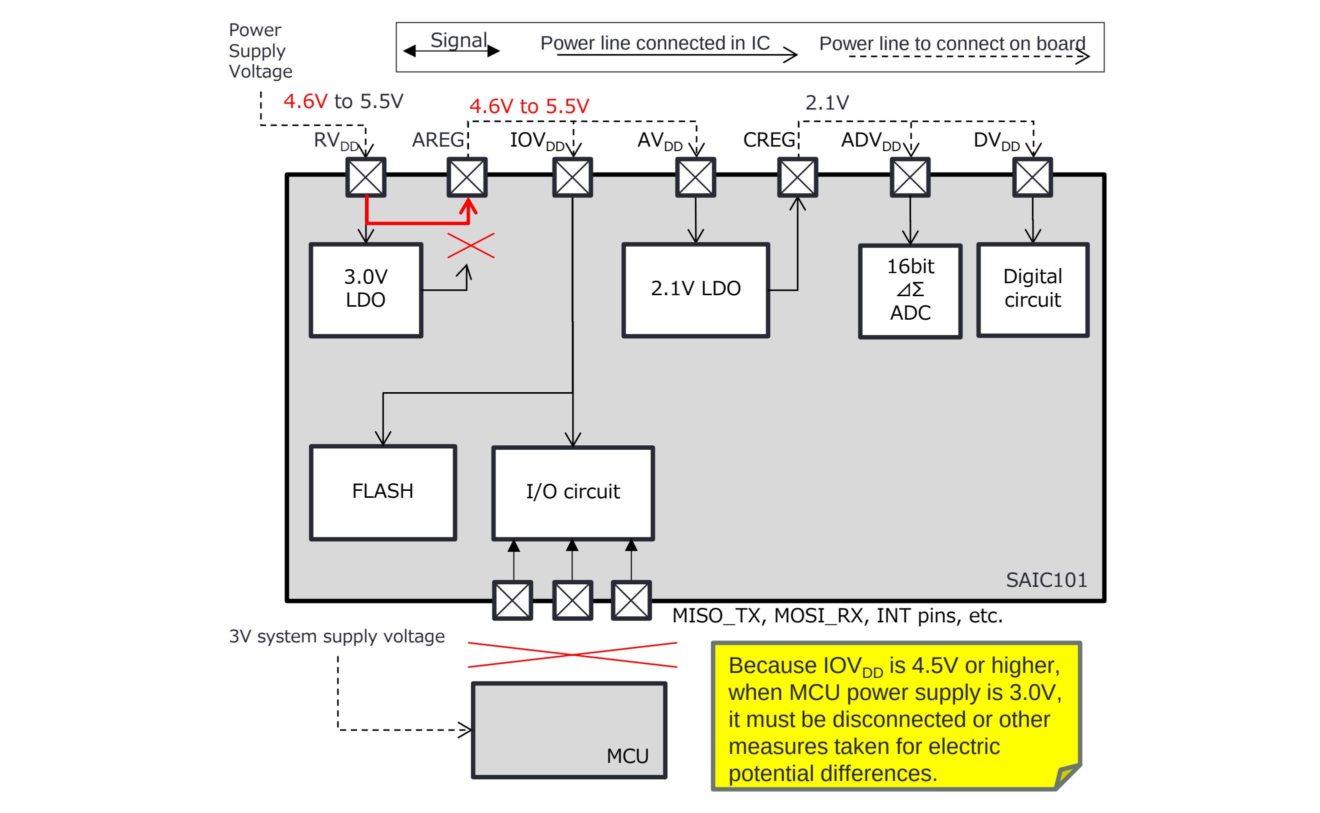
### 9.1.5 Power Supply Configuration 2 (for flash programming)

Figure 9-4 shows the power supply configuration when SAIC101 operates in a range of 3.3V to 5.5V, the MCU operates at 3.0V with an electric potential difference (as in Configuration 2 in Table 9.1).

- Power supply configuration changes for flash programming

Apply 4.6V to 5.5V when SAIC101 power supply is less than 4.6V.

When the MCU withstand voltage is less than 4.6V, either disconnect the SAIC101 power supply, or take other electric potential difference measures.



Because $IOV_{DD}$ is 4.5V or higher, when MCU power supply is 3.0V, it must be disconnected or other measures taken for electric potential differences.

**Figure 9-4 Power supply configuration for flash programming**

- Software settings for flash programming

Set the PSTHRU bit of the CHIPCNT register to 1U and connect the $RV_{DD}$ pin and AREG pin in the IC.

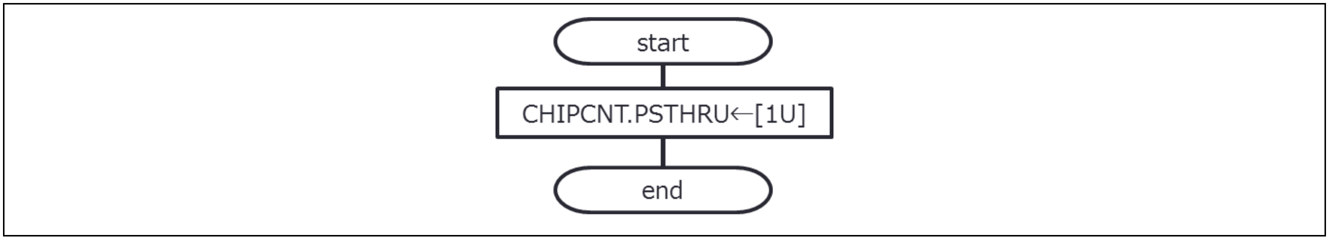


**Figure 9-5 Software settings for flash programming**

— How to set PSTHRU bit

Set the PSTHRU bit of the CHIPCNT register to 1U by software.

Note: After RESET release, the PSTHRU bit can be set until the ADSTART bit of the ADCCNT register is set to 1U. After the ADSTART bit has been set to 1U, it is fixed at 0U until the next RESET occurs.

### 9.1.6 Power supply configuration 3 (normal operations)

Figure 9-6 shows the power supply configuration when SAIC101 and the MCU operate in a range of 2.7V to 3.6V with the same electric potential (as in Configuration 3 in Table 9.1).

- Power supply configuration for normal operations

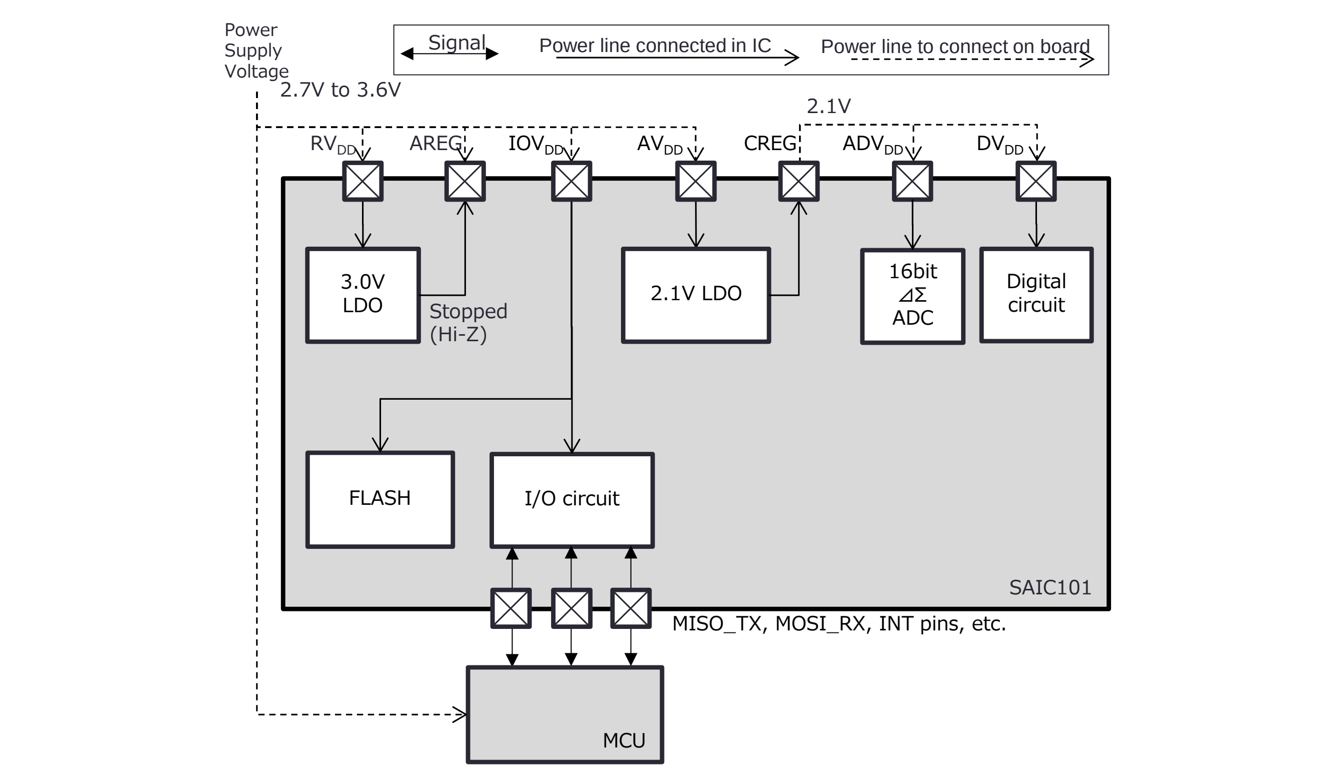


**Figure 9-6 Power supply configuration for normal operations**

- Software settings for normal operations

Set the AREGPD bit of the CHIPCNT register to 1U, and stop AREG operations.

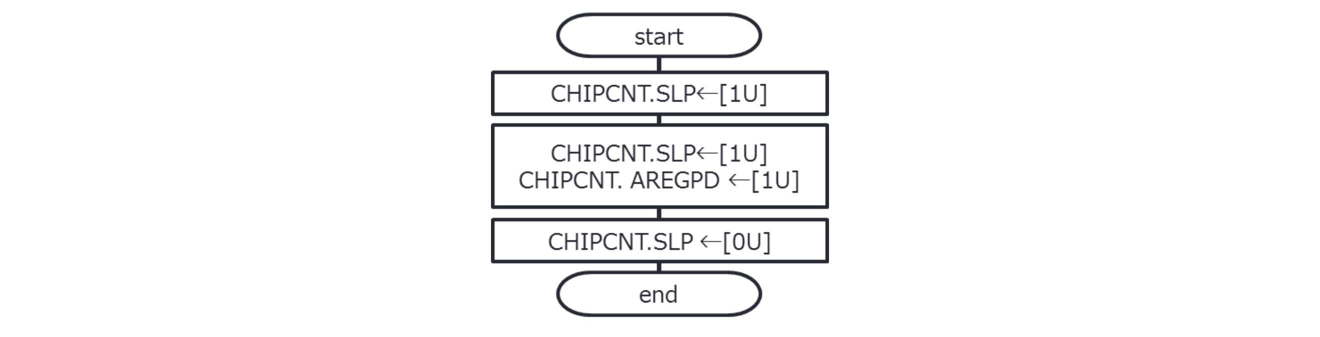


**Figure 9-7 Software settings for normal operations**

— How to stop AREG

A protection function is in place to prevent incorrect writing of bits. To set AREGPD bit of CHIPCNT register to 1U, first set SLP bit to 1U, then write the SLP bit 1U and AREGPD bit 1U simultaneously.

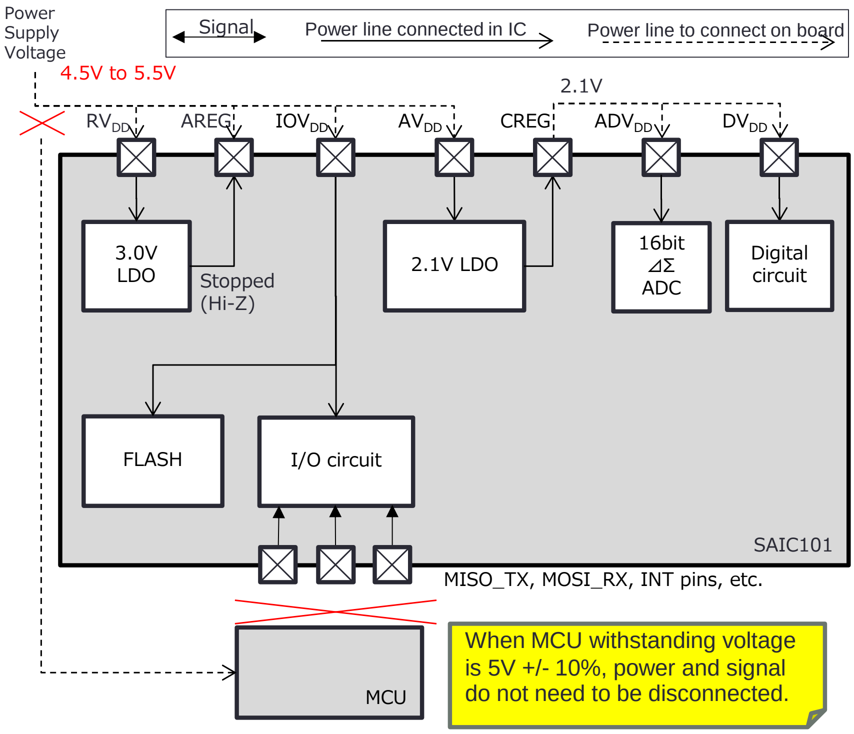
### 9.1.7 Power Supply Configuration 3 (flash programming)

Figure 9-8 shows the power supply configuration when SAIC101 and the MCU operate in a range of 2.7V to 3.6V with the same electric potential (as in Configuration 3 in Table 9.1).

- Power supply configuration changes for flash programming

Apply 4.5V to 5.5V when SAIC101 power supply is less than 4.5V.

When the MCU withstand voltage is less than 4.5V, either disconnect the SAIC101 power supply, or take other electric potential difference measures.



When MCU withstanding voltage is 5V +/- 10%, power and signal do not need to be disconnected.

**Figure 9-8 Power supply configuration for flash programming**

- Software settings for flash programming

Set the AREGPD bit of the CHIPCNT register to 1U and stop AREG operations.

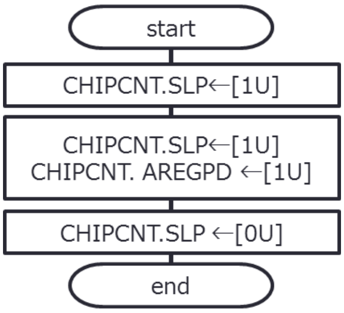


**Figure 9-9 Software settings for flash programming**

— How to stop AREG

A protection function is in place to prevent incorrect writing of bits. To set AREGPD bit of CHIPCNT register to 1U, first set SLP bit to 1U, then write the SLP bit 1U and AREGPD bit 1U simultaneously.

## 9.2 Power-saving Function

### 9.2.1 Power-saving mode

SAIC101 power consumption can be reduced by setting the SENSPD and SLP bits of the CHIPCNT register. Table 9.2 shows the restrictions that apply when using power-saving settings.

**Table 9.2 Operating modes**

| **Operating Modes** | **Register Settings** | | **A/D Conversion** availability | **R/W to FLASH** availability | **Effect on Functions** |
|---|---|---|---|---|---|
| | **SLP bit** | **SENSPD bit** | | | |
| Normal operations | 0 | 0 | Yes | Yes | - |
| SBIAS Operations stopped | 0 | 1 | **No** | Yes | • Stops AFE VREF<br>• Stops A/D converter<br>• Stops D/A converter<br>• Stops internal bias voltage (VBIAS)<br>• Stops sensor power supply (SBIAS)<br>• Stops programmable gain instrumentation amplifier (PGIA) |
| Sleep mode | 1 | Don't care | **No** | **No** | • Stops internal reference voltage generator (VREF)<br>High-precision BGR[Note], analog circuit reference voltage generator (AFE VREF)<br>• Stops A/D converter<br>• Stops D/A converter<br>• Stops internal bias voltage (VBIAS)<br>• Stops sensor power supply (SBIAS)<br>• Stops programmable gain instrumentation amplifier (PGIA)<br>• Limits CREG output current to 2mA and switches to lower-power BGR<br>• Stops oscillation circuit for internal system clock (OSC)[Note] |

Note: Stopped only during SPI mode.

### 9.2.2 Control module

Figure 9-10 shows the SLP and SENSPD bit control module layout.

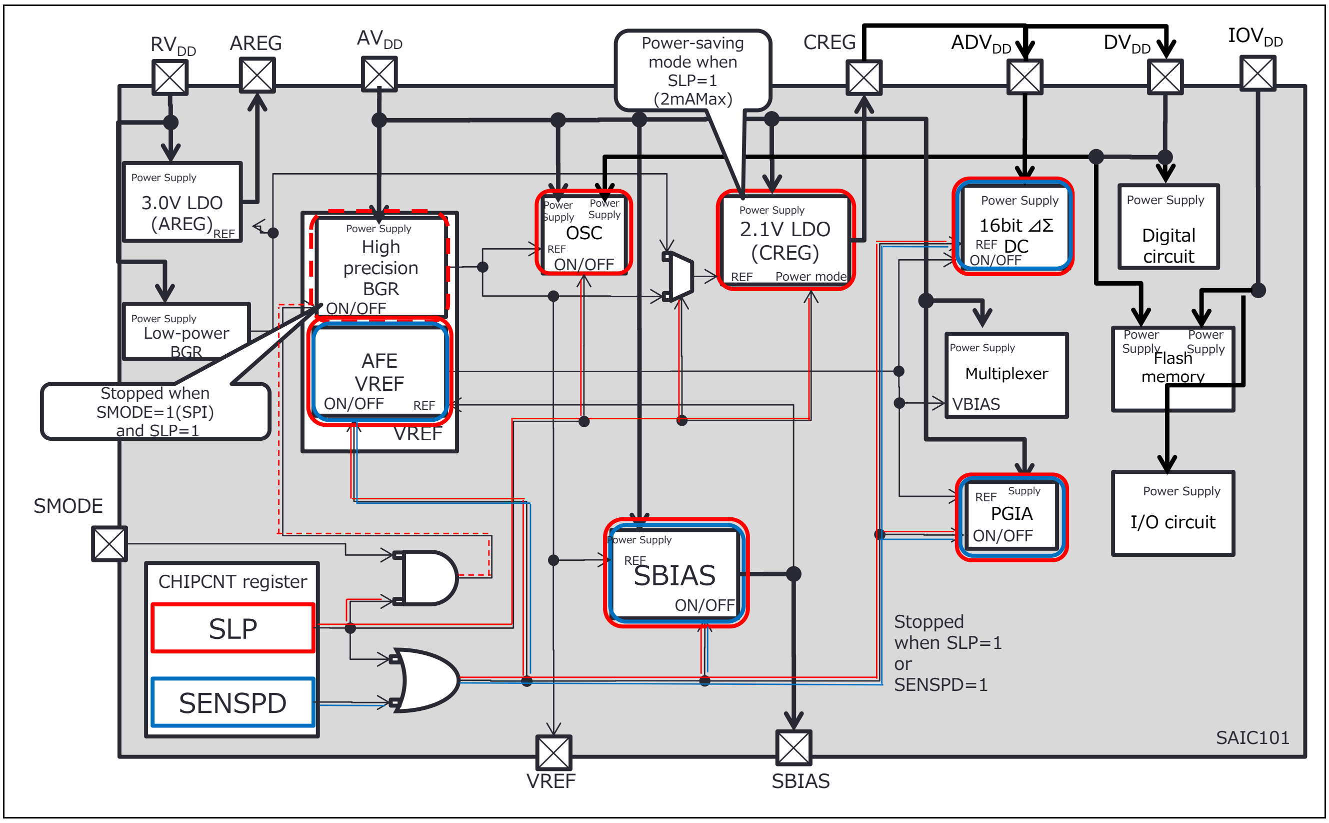

RVDD
AREG
AVDD
Power-saving mode when SLP=1 (2mAMax)
CREG
ADVDD
DVDD
IOVDD
Power Supply 3.0V LDO (AREG) REF
Power Supply Low-power BGR
Stopped when SMODE=1(SPI) and SLP=1
Power Supply High precision BGR ON/OFF
AFE VREF ON/OFF REF
VREF
Power Supply Power Supply OSC REF ON/OFF
Power Supply 2.1V LDO (CREG) REF Power mode
Power Supply 16bit ΔΣ REF DC ON/OFF
Power Supply Digital circuit
Power Supply Multiplexer VBIAS
Power Supply Power Supply Flash memory
REF Supply PGIA ON/OFF
Power Supply I/O circuit
SMODE
CHIPCNT register
SLP
SENSPD
Power Supply REF SBIAS ON/OFF
Stopped when SLP=1 or SENSPD=1
SAIC101
VREF
SBIAS

**Figure 9-10 Control Module**

### 9.2.3 How to stop/restart SBIAS operations

- How to stop SBIAS operations

Implement the following settings by software: set SLP bit of CHIPCNT register to 0U and SENSPD bit to 1U.

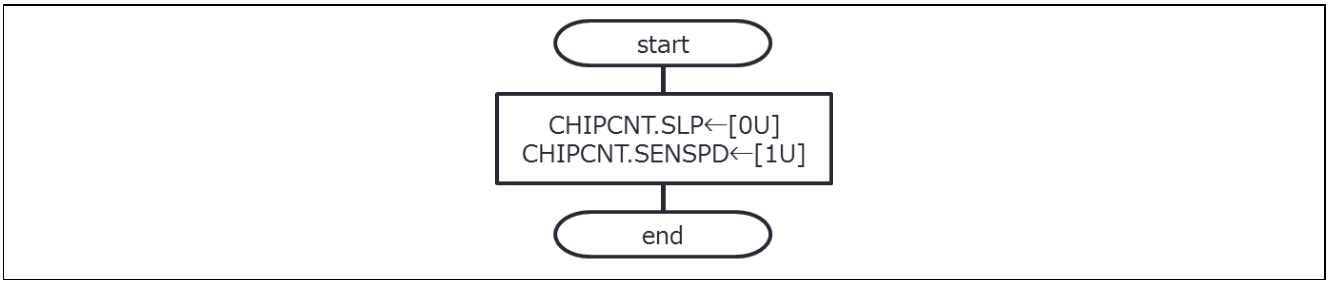


**Figure 9-11 SBIAS operations stop routine**

- Transition timing

| Module | Item | max | Unit | Conditions |
|---|---|---|---|---|
| SBIAS | Turn off time | (5) | ms | $V_{OUT}$ < 10% |

Note: Values indicated in parentheses are target design values.

- How to restart SBIAS operations

Implement the following settings by software: set SLP bit of CHIPCNT register to 0U and SENSPD bit to 0U

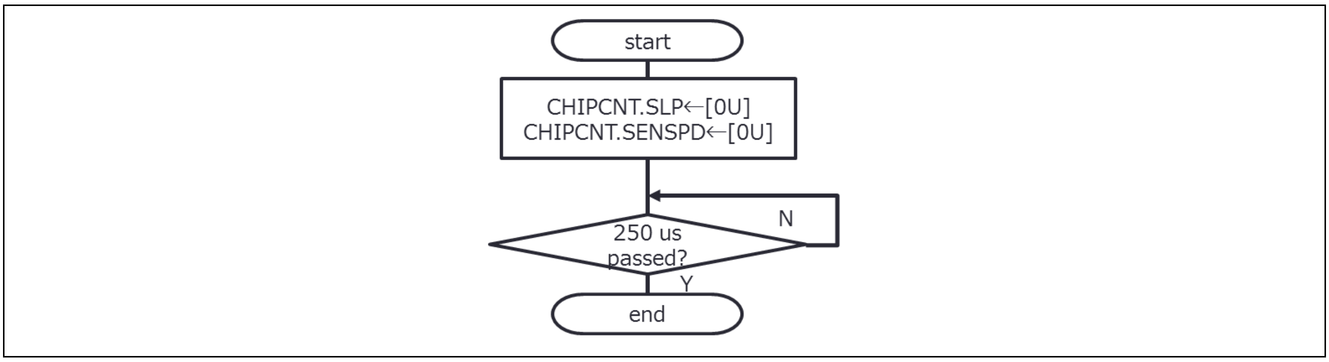


**Figure 9-12 SBIAS operations restart routine**

- Transition timing

| Module | Item | max | Unit | Conditions |
|---|---|---|---|---|
| SBIAS | Turn on time | (250) | µs | $V_{OUT}$ > 90% |

Note: Values indicated in parentheses are target design values.

### 9.2.4 How to transition to sleep mode

- How to transition to sleep mode

Implement the following setting by software: set SLP bit of CHIPCNT register to 1U

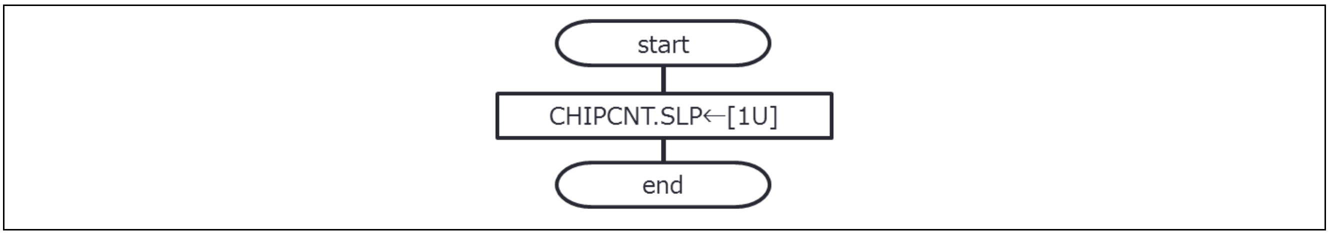


**Figure 9-13 Sleep mode ON setting routine**

- Transition timing

| Module | Item | max | Unit | Conditions |
|---|---|---|---|---|
| SBIAS | Turn off time | (5) | ms | $V_{OUT}$ < 10% |
| CREG | Mode switch time 1 | (400) | µs | Normal operations → wait mode |
| VREF | - | - | µs | Stops only in SPI mode |

Note: Values indicated in parentheses are target design values.

- How to return from sleep mode

Implement the following setting by software: set SLP bit of CHIPCNT register to 0U

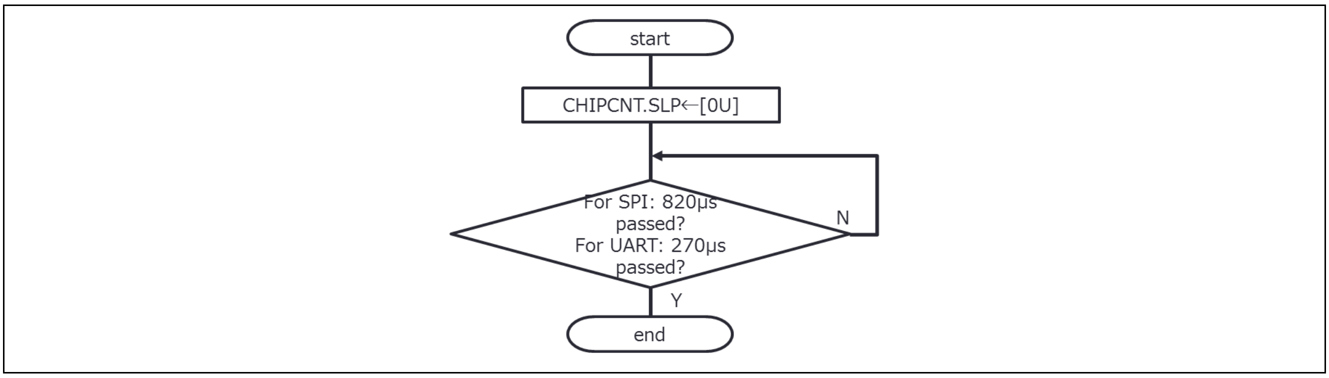


**Figure 9-14 Sleep mode OFF setting routine**

- Transition timing

| Module | Item | max | Unit | Conditions |
|---|---|---|---|---|
| SBIAS | Turn on time | (250) | µs | $V_{OUT}$ > 90% |
| CREG | Mode switch time 2 | (150) | µs | Wait mode → normal operations<br>Not available when VREF is turned on |
| VREF | Turn on time | (550) | µs | SPI mode only |
| All (SPI) | Wake up time | (820) | µs | Duration between sleep mode cleared (SLP = 1U→0U) and A/D conversion ready |
| All (UART) | Wake up time | (270) | µs | |

Note: Values indicated in parentheses are target design values.

### 9.2.5 How to stop/restart AREG operations

- How to stop AREG operations

Implement the following setting by software: set AREGPD bit of CHIPCNT register to 1U

Note: To set AREGPD bit to 1U, first set SLP bit to 1U, then write the SLP bit 1U and AREGPD bit 1U simultaneously.

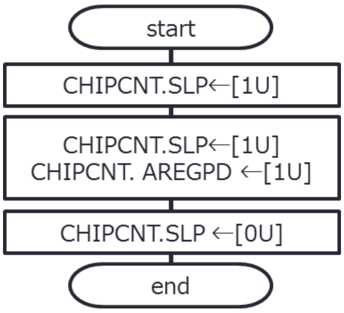


**Figure 9-15 AREG operation stop routine**

- How to restart AREG operations

Implement the following setting by software: set AREGPD bit of CHIPCNT register to 0U

Note: To set AREGPD bit to 0U, first set SLP bit to 1U, then write the SLP bit 1U and AREGPD bit 0U simultaneously.

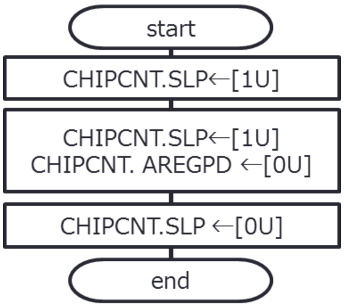


**Figure 9-16 AREG operation restart routine**

- Transition timing

| Module | Item | max | Unit | Conditions |
|---|---|---|---|---|
| AREG | Turn on time | (1800) | µs | $RV_{DD}$ ≥ 3.3 V,<br>$V_{OUT}$ > 90%,<br>$I_{OUT}$ = 0 mA |

Note: Values indicated in parentheses are target design values.

## 9.3 SAIC Startup (power-on) Sequence

The SAIC101 startup sequence varies based on the register shadow settings. In UART mode, 256-byte transmission[Note 1] and A/D results transmission are conducted automatically. Table 9.3 shows the startup sequence according to various settings.

**Table 9.3 Startup sequences according to settings**

| **CPSOR**[Note 2]<br>**Copy register shadow**<br>**0 = None**<br>**1 = copy** | **SDCOR** [Note2]<br>**256-byte transmission**<br>**0 = None**<br>**1 = transmission** | **SLP** [Note2]<br>**Sleep mode control**<br>**0 = operating**<br>**1 = sleep** | **SENSPD**[Note2]<br>**SBIAS operation control**<br>**0 = operating**<br>**1 = stopped** | **ADSTART** [Note2]<br>**AD conversion operation control**<br>**0 = stop**<br>**1 = start** | **SMODE pin**<br>**Serial mode selection**<br>**L = UART**<br>**H = SPI** | Operations after SAIC101 Startup<br>**(Copy register shadow operation abbreviated)** |
|---|---|---|---|---|---|---|
| 0 | - | - | - | - | - | - |
| 1 | 0 | 0 | 0 | 0 | - | - |
| | | | | 1 | L | A/D conversion operating<br>A/D results atuo transfer |
| | | | | | H | A/D conversion operating |
| | | | 1 | - | - | SBIAS operation stopped |
| | | 1 | - | - | - | Sleep mode |
| | 1[Note1] | 0 | 0 | 0 | L | 256-byte auto-transmission[Note1] |
| | | | | | H | - |
| | | | | 1 | L | 256-byte auto-transmission[Note1]<br>A/D conversion operating<br>A/D results auto transfer |
| | | | | | H | A/D conversion operating |
| | | | 1 | - | L | 256-byte auto-transmission[Note1]<br>SBIAS operations stopped |
| | | | | | H | SBIAS operation stopped |
| | | 1 | - | - | L | 256-byte auto-transmission[Note1]<br>Sleep mode |
| | | | | | H | Sleep mode |

Note 1: This API does not support 256-byte transmissions (CPSOR bit of STARTUP register set to 1U and SDCOR bit to 1U)

Note 2: CPSOR, SDCOR, SLP, SENSPD, and ADSTART indicate register shadow value bits that correspond to flash memory areas.

## Website and Support

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

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

## Revision History

| | | Description | |
|---|---|---|---|
| **Rev.** | **Date** | **Page** | **Summary** |
| Rev.1.00 | **Feb 01,** 2015 | --- | First edition issued |

## 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 document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — 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 a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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.**
2801 Scott Boulevard Santa Clara, CA 95050-2549, 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.**
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

**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: +852-2265-6688, Fax: +852 2886-9022/9044

**Renesas Electronics Taiwan Co., Ltd.**
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

**Renesas Electronics Singapore Pte. Ltd.**
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

**Renesas Electronics Malaysia Sdn.Bhd.**
Unit 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.**
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0