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 .................................................................................................................. 5
   1.1 Overview ..................................................................................................................... 5
   1.2 Operation Confirmation Conditions ......................................................................... 5
   1.3 Software Configuration ............................................................................................. 6

2. Cautions for API Usage .................................................................................................... 7
   2.1 [UART/SPI] Cautions for Communications ............................................................. 7
      2.1.1 Writing global variables, definitions in API User Definition Files ..................... 7
      2.1.2 Number of Flash Memory Rewrites ................................................................... 7
      2.1.3 Caution for when SBIAS is stopped ................................................................... 7
      2.1.4 Caution for rewriting register shadow area ....................................................... 7
      2.1.5 Caution for API function internal wait times ..................................................... 7
   2.2 [UART] Cautions for Communications .................................................................... 8
      2.2.1 Caution for state of MOSI_RX pin ................................................................. 8
      2.2.2 Caution for when AREG operation is stopped ................................................. 8
      2.2.3 Startup sequence data transfer ......................................................................... 8
   2.3 [SPI] Cautions for Communications ......................................................................... 8
      2.3.1 Caution for SPI mode ......................................................................................... 8
      2.3.2 Caution for MCU settings and SPI control ...................................................... 8

3. API Functions .................................................................................................................. 10
   3.1 UART Control ............................................................................................................. 10
      3.1.1 Communications ............................................................................................... 10
      3.1.2 Flash memory ..................................................................................................... 10
      3.1.3 A/D Converter .................................................................................................... 11
      3.1.4 Power supply ...................................................................................................... 11
   3.2 SPI Control ................................................................................................................ 12
      3.2.1 Communications ............................................................................................... 12
      3.2.2 Flash Memory .................................................................................................... 12
      3.2.3 A/D converter .................................................................................................... 13
      3.2.4 Power supply ...................................................................................................... 13

4. Common API Definitions ................................................................................................. 14
4.1 Common API Function Return Values .......................................................... 14
4.2 Macro Declarations for User Environment-dependent Settings ......................... 14
  4.2.1 UART communications ........................................................................... 14
  4.2.2 SPI communications ............................................................................ 16
4.3 Macro Declarations ....................................................................................... 17
4.4 Type Declarations ......................................................................................... 18
4.5 Enumerations Requiring User Modification .................................................... 19
4.6 Enumerations ............................................................................................... 20
4.7 Structures ..................................................................................................... 25
4.8 Unions ........................................................................................................... 28
4.9 Global constants for user environment-dependent settings ............................... 30
  4.9.1 Global constants for user environment-dependent settings in UART communications ..... 30
  4.9.2 Global constants for user environment-dependent settings when using SPI ........... 34
4.10 Global Constants .......................................................................................... 37
  4.10.1 UART Control......................................................................................... 37
  4.10.2 SPI Control............................................................................................ 38
4.11 Global Variables .......................................................................................... 38

5. Communication-related Definitions .................................................................. 39
5.1 API Function Specifications .......................................................................... 39
  5.1.1 [UART/SPI] Smart Analog initialization function ....................................... 39
  5.1.2 [UART/SPI] Smart Analog RESET function .............................................. 40
  5.1.3 [UART/SPI] Read register bytes function .................................................. 41
  5.1.4 [UART/SPI] Write register bytes function .................................................. 42
  5.1.5 [UART/SPI] Write-verify register bytes function ........................................ 43
  5.1.6 [UART/SPI] SAIC101 command function .................................................. 44
  5.1.7 [UART] Communication setting negotiation function ............................... 45
  5.1.8 [SPI] Read register bits function ............................................................. 46
  5.1.9 [SPI] Write register bits function ............................................................. 47
  5.1.10 [SPI] Write-verify register bits function .................................................... 48
  5.1.11 [SPI] CS enable function ....................................................................... 49
  5.1.12 [SPI] CS check function ....................................................................... 49
  5.1.13 [SPI] CS disable function ..................................................................... 50
5.2 Internal Function Specifications ..................................................................... 51
  5.2.1 [UART/SPI] Smart Analog external RESET function .................................. 51
  5.2.2 [UART/SPI] Smart Analog internal RESET function ................................... 51
  5.2.3 [UART/SPI] Smart Analog power-on RESET wait function ....................... 52
  5.2.4 [UART/SPI] NOP execution function ....................................................... 52
  5.2.5 [UART/SPI] Burst read function ............................................................... 53
  5.2.6 [UART/SPI] Burst write function ............................................................... 54
  5.2.7 [UART/SPI] SAIC101 dedicated communication command format conversion function... 54
6. Flash Memory Control Function Definitions

6.1 API Function Specifications

6.1.1 [UART/SPI] SAIC101 command function

6.2 Internal Function Specifications

6.2.1 [UART/SPI] Read flash data function

6.2.2 [UART/SPI] Write flash data function

6.2.3 [UART/SPI] Erase all flash data function

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

6.2.5 [UART/SPI] Flash shadow area copy function

6.2.6 [UART/SPI] Flash system setting copy function

6.2.7 [SPI] INTFLAG register FR bit acquisition function

6.2.8 [SPI] INTFLAG register FW bit acquisition function

6.2.9 [SPI] INTFLAG register FAE bit acquisition function

6.2.10 [SPI] INTFLAG register RAW bit acquisition function

6.2.11 [SPI] STATUS register FWIP bit acquisition function

6.2.12 [SPI] STATUS register FAEIP bit acquisition function

6.2.13 [SPI] STATUS register RAWIP bit acquisition function

6.3 System Function Specifications

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

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

7. ADC Definitions

7.1 API Function Specifications

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

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

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

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

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

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

7.2 Internal Function Specifications

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

7.2.2 [SPI] INTFLAG register ADC bit acquisition function

7.2.3 [SPI] STATUS register ADCIP bit acquisition function
8. Power Supply Definitions ........................................................................................................... 83
  8.1 API Function Specifications.................................................................................................. 83
      8.1.1 [UART/SPI] AREG ON setting function ................................................................... 83
      8.1.2 [UART/SPI] AREG OFF setting function .................................................................... 84
      8.1.3 [UART/SPI] SBIAS register setting function ............................................................... 85
      8.1.4 [UART/SPI] SBIAS register acquisition function ......................................................... 87
      8.1.5 [UART/SPI] Sleep mode ON setting function ............................................................. 88
      8.1.6 [UART/SPI] Sleep mode OFF setting function ............................................................ 89

9. Power Supply Settings ............................................................................................................ 90
  9.1 Power Supply Configurations ............................................................................................... 90
      9.1.1 List of configurations .................................................................................................... 90
      9.1.2 Power supply configuration 1 (normal operations) ...................................................... 90
      9.1.3 Power supply configuration 1 (for flash programming) ............................................... 91
      9.1.4 Power supply configuration 2 (normal operations) ...................................................... 92
      9.1.5 Power Supply Configuration 2 (for flash programming) ............................................ 93
      9.1.6 Power supply configuration 3 (normal operations) ..................................................... 94
      9.1.7 Power Supply Configuration 3 (flash programming) .................................................. 95

  9.2 Power-saving Function ......................................................................................................... 96
      9.2.1 Power-saving mode ....................................................................................................... 96
      9.2.2 Control module ............................................................................................................ 97
      9.2.3 How to stop/restart SBIAS operations ....................................................................... 98
      9.2.4 How to transition to sleep mode ................................................................................. 99
      9.2.5 How to stop/restart AREG operations ..................................................................... 100

  9.3 SAIC Startup (power-on) Sequence .................................................................................... 101
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

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation boards</td>
<td>• Renesas Starter Kit for RL78/L13 [R0K5010WMS900BE]</td>
</tr>
<tr>
<td></td>
<td>- Renesas Starter Kit for RL78/L13 CPU board</td>
</tr>
<tr>
<td></td>
<td>• Smart Analog IC RSK Option Evaluation Board [TSA-OP-IC101]</td>
</tr>
<tr>
<td>Target device</td>
<td>R5F10WMGAFB (RL78/L13)</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>24MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>5.0V</td>
</tr>
<tr>
<td>Integrated Development Environment</td>
<td>V2.02.00 [21 Feb 2014]</td>
</tr>
<tr>
<td>Integrated Development Environment</td>
<td>CA78K0R V4.02.00.03 [16 Jan 2014]</td>
</tr>
<tr>
<td>C Compiler (CubeSuite+)</td>
<td>V1.02.01.02 [11 Jun 2014]</td>
</tr>
<tr>
<td>RL78/L13 Code Library (CubeSuite+)</td>
<td>V3.0.0.22</td>
</tr>
<tr>
<td>C Compiler (e2studio)</td>
<td>GNUURL78 v14.01</td>
</tr>
<tr>
<td>RL78/L13 Code Library (e2studio)</td>
<td>V1.02.00.03 [11 Feb 2014]</td>
</tr>
</tbody>
</table>

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.

```c
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.

---

**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 unintentionally, 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\textsuperscript{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, 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

Fail to acquire normal A/D-converted value if 3-byte read is not completed in this period.

Figure 2.2  Comparison of INT pin and polling control methods

When using INT pin

- Start A/D conversion
- INT pin = High?
  - YES: Acquire A/D value
  - NO: Specified number of conversions
- End

When using polling control

- Start A/D conversion
- Read STATUS register
- A/D conversion complete?
  - YES: Acquire A/D value
  - NO: Specified number of conversions
- End

Processing time depends on frequency of CPU and peripherals and the SPI communication baud rate.
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**

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

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**

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

Note: R_SAIC_UART_SAIC101 includes both register access and flash memory access functions. 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

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

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

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_SAIC_UART_AregOn</td>
<td>AREG ON setting function for UART control</td>
</tr>
<tr>
<td>R_SAIC_UART_AregOff</td>
<td>AREG OFF setting function for UART control</td>
</tr>
<tr>
<td>R_SAIC_UART_SbiasRegSet</td>
<td>SBIAS register setting function for UART control</td>
</tr>
<tr>
<td>R_SAIC_UART_SbiasRegGet</td>
<td>SBIAS register acquisition function for UART control</td>
</tr>
<tr>
<td>R_SAIC_UART_SleepModeOn</td>
<td>Sleep mode ON setting function for UART control</td>
</tr>
<tr>
<td>R_SAIC_UART_SleepModeOff</td>
<td>Sleep mode OFF setting function for UART control</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_SAIC_SPI_Init</td>
<td>Smart Analog initialization function</td>
</tr>
<tr>
<td>R_SAIC_SPI_Reset</td>
<td>Smart Analog reset function</td>
</tr>
<tr>
<td>R_SAIC_SPI_Read</td>
<td>Read register bytes function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_Write</td>
<td>Write register bytes function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_WriteVerify</td>
<td>Write-verify register bytes function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_ReadBit</td>
<td>Read register bits function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_WriteBit</td>
<td>Write register bits function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_WriteVerifyBit</td>
<td>Write-verify register bits function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_CSEnable</td>
<td>CS enable function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_CSCheck</td>
<td>CS check function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_CSDisable</td>
<td>CS disable function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_SAIC101</td>
<td>SAIC 101 command function for SPI control</td>
</tr>
</tbody>
</table>

Table 3-5 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**

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

Table 3-6 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**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_SAIC_SPI_ADC_Start</td>
<td>A/D conversion start process function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_ADC_Stop</td>
<td>A/D conversion stop process function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_ADC_GetResult</td>
<td>A/D-converted value acquisition function for SPI control. Acquire data from multiple channels, multiple times for each channel.</td>
</tr>
<tr>
<td>R_SAIC_SPI_ADC_GetResult_1Shot</td>
<td>A/D-converted value acquisition function for SPI control. Acquire data from a single channel in 1Shot mode.</td>
</tr>
<tr>
<td>R_SAIC_SPI_ADC_Init</td>
<td>A/D converter register initial setup function for SPI control</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_SAIC_SPI_AregOn</td>
<td>AREG ON setting function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_AregOff</td>
<td>AREG OFF setting function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_SbiasRegSet</td>
<td>SBIAS register setting function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_SbiasRegGet</td>
<td>SBIAS register acquisition function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_SleepModeOn</td>
<td>Sleep mode ON setting function for SPI control</td>
</tr>
<tr>
<td>R_SAIC_SPI_SleepModeOff</td>
<td>Sleep mode OFF setting function for SPI control</td>
</tr>
</tbody>
</table>
4. Common API Definitions

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

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Macro Name</th>
<th>Constant Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saic_status_t</td>
<td>D_SAIC_OK</td>
<td>00H</td>
<td>Successful completion</td>
</tr>
<tr>
<td></td>
<td>D_SAIC_ERR_PARAM</td>
<td>01H</td>
<td>Parameter error</td>
</tr>
<tr>
<td></td>
<td>D_SAIC_ERR_COM</td>
<td>02H</td>
<td>Communication error (overrun error, timeout error)</td>
</tr>
<tr>
<td></td>
<td>D_SAIC_ERR_VERIFY</td>
<td>03H</td>
<td>Verify error</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th>Default Setting Value</th>
<th>Input Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_DEADLOCK_CNT</td>
<td>11000000L</td>
<td>uint32_t ^32</td>
<td>Number of loops to judge deadlock during communication wait</td>
</tr>
</tbody>
</table>

Note 1: Specify value larger than 0.

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

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_SAIC_FLASH_API_VALID</td>
<td>Enables FLASH API function. Disable when not using FLASH API to reduce RAM capacity.</td>
</tr>
<tr>
<td>D_UART_NEGOTIATION_250KBPS_PARITY_ODD</td>
<td>Enables negotiation for baudrate=250000bps, Parity=odd. Disable when not used.</td>
</tr>
<tr>
<td>D_UART_NEGOTIATION_250KBPS_PARITY_EVEN</td>
<td>Enables negotiation for baudrate=250000bps, Parity=even. Disable when not used.</td>
</tr>
<tr>
<td>D_UART_NEGOTIATION_250KBPS_PARITY_NONE</td>
<td>Enables negotiation for baudrate=250000bps, Parity=none. Disable when not used.</td>
</tr>
<tr>
<td>D_UART_NEGOTIATION_4800BPS_PARITY_ODD</td>
<td>Enables negotiation for baudrate=4800bps, Parity=odd. Disable when not used.</td>
</tr>
<tr>
<td>D_UART_NEGOTIATION_4800BPS_PARITY_EVEN</td>
<td>Enables negotiation for baudrate=4800bps, Parity=even. Disable when not used.</td>
</tr>
<tr>
<td>D_UART_NEGOTIATION_4800BPS_PARITY_NONE</td>
<td>Enables negotiation for baudrate=4800bps, Parity=none. Disable when not used.</td>
</tr>
</tbody>
</table>
Table 4-4  [UART] Macro declarations

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th>Default Setting Value</th>
<th>Input Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_CPU_CLK_MHZ</td>
<td>24.0F</td>
<td>float</td>
<td>Definition of CPU and peripheral clock frequency. Use to calculate general software wait used in API. Unit: MHz</td>
</tr>
<tr>
<td>D_WAIT_PON_RST_TIME_MS</td>
<td>4.00F</td>
<td>float</td>
<td>Definition of power-on RESET wait. Use to determine wait when power-on RESET is specified in the RESET function. Unit: ms</td>
</tr>
</tbody>
</table>

Note 1: Specify 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

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th>Default Setting Value</th>
<th>Input Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_DEADLOCK_CNT</td>
<td>11000000L</td>
<td>uint32_t</td>
<td>Number of loops to judge deadlock during communication wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note1: Specify value larger than 0.</td>
</tr>
<tr>
<td>D_SPI_OPERATION</td>
<td></td>
<td>D_SPI_USE_INTERRUPT</td>
<td>Definition for selecting use of communication module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or</td>
<td>interrupt or no use (use polling)</td>
</tr>
</tbody>
</table>

(b) r_sa_spi_control_register_user.c

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

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_SAIC_FLASH_API_VALID</td>
<td></td>
<td>Enables FLASH API function. Disable when not using FLASH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>API to reduce RAM capacity.</td>
</tr>
</tbody>
</table>

Table 4-7 [SPI] Macro declarations

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

Note1: Specify value larger than 0.
Note2: Specify setting value for the MCU CPU clock.
4.3 Macro Declarations

This section describes the macro declarations defined in the API.

(a)  
Table 4-8  [UART] Macro declarations

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_UART_USE_INTERRUPT</td>
<td>1U</td>
<td>Definition of communication module interrupt use</td>
</tr>
<tr>
<td>D_UART REGISTER_POLLING</td>
<td>2U</td>
<td>Definition for not using communication module interrupt (use polling)</td>
</tr>
<tr>
<td>D_UART_OPERATION</td>
<td>D_UART_USE_INTERRUPT</td>
<td>Definition for selecting use/not use (use polling) communication module interrupt</td>
</tr>
</tbody>
</table>

(b)  
Table 4-9  [SPI] Macro declarations

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_SPI_USE_INTERRUPT</td>
<td>1U</td>
<td>Definition of communication module interrupt use</td>
</tr>
<tr>
<td>D_SPI REGISTER POLLING</td>
<td>2U</td>
<td>Definition for not using communication module interrupt (use polling)</td>
</tr>
</tbody>
</table>

(c)  
Table 4-10  [UART] Macro declarations

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

(d)  
Table 4-11  [SPI] Macro declarations

<table>
<thead>
<tr>
<th>Macro Declaration</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_BURST_MAX_SIZE</td>
<td>16U</td>
<td>Definition of SAIC101 burst read/write maximum data size</td>
</tr>
<tr>
<td>D_COMMAND_LENGTH</td>
<td>2U</td>
<td>Definition of number of command bytes for: burst read register, burst write register, read flash, and write flash commands</td>
</tr>
<tr>
<td>D_FLASH_READ_MAX_SIZE</td>
<td>256U</td>
<td>Definition of SAIC101 flash memory read maximum data size</td>
</tr>
<tr>
<td>D_REGISTER_MAX_ADDRESS</td>
<td>1FH</td>
<td>Definition of SAIC101 register maximum address</td>
</tr>
<tr>
<td>D_FLASH_MAX_ADDRESS</td>
<td>FFH</td>
<td>Definition of SAIC101 flash memory maximum address</td>
</tr>
</tbody>
</table>
4.4 Type Declarations

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

(a) \texttt{r\_sa\_uart\_control\_register.h} / \texttt{r\_sa\_spi\_control\_register.h}

<table>
<thead>
<tr>
<th>Table 4-12 [UART/SPI] Type declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>saic_status_t</td>
</tr>
</tbody>
</table>

(b) \texttt{r\_sa\_spi\_control\_register.c}

<table>
<thead>
<tr>
<th>Table 4-13 [UART/SPI] Type declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>saic_func_bitRead_t</td>
</tr>
</tbody>
</table>
4.5 Enumerations Requiring User Modification

The enumerations shown in Table 4-14 and 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) \texttt{r\_sa\_uart\_control\_register.h}

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Default Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_uart_ch_t</td>
<td>E_UART0</td>
<td>Define enumerations according to the number of arrays defined in g_uart_serial_data_tbl.</td>
</tr>
<tr>
<td></td>
<td>E_UART1</td>
<td><strong>Enumerations</strong> names are used in g_uart_saic_data_tbl.</td>
</tr>
<tr>
<td></td>
<td>E_UART2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_UART3</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>E_UART4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_UART5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_UART6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_UART_MAX</td>
<td>Determine maximum value of g_uart_serial_data_tbl[].</td>
</tr>
</tbody>
</table>

(b) \texttt{r\_sa\_spi\_control\_register.h}

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Default Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_csi_ch_t</td>
<td>E_CSI00</td>
<td>Define enumerations according to the number of arrays defined in g_spi_serial_data_tbl.</td>
</tr>
<tr>
<td></td>
<td>E_CSI01</td>
<td><strong>Enumeration</strong> names are used in g_spi_saic_data_tbl.</td>
</tr>
<tr>
<td></td>
<td>E_CSI10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_CSI11</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>E_CSI20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_CSI21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_CSI30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_CSI31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E_CSI_MAX</td>
<td>Determine maximum value of g_spi_serial_data_tbl[].</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Table 4-16</th>
<th>[UART/SPI] Enumeration for specifying SAIC type (part name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part name</td>
<td>Macro Name</td>
</tr>
<tr>
<td>e_saic_type_t</td>
<td>E_SAIC300</td>
</tr>
<tr>
<td></td>
<td>E_SAIC301</td>
</tr>
<tr>
<td></td>
<td>E_SAIC500</td>
</tr>
<tr>
<td></td>
<td>E_SAIC501</td>
</tr>
<tr>
<td></td>
<td>E_SAIC502</td>
</tr>
<tr>
<td></td>
<td>E_SAIC101</td>
</tr>
<tr>
<td></td>
<td>E_IC_TYPE_MAX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4-17</th>
<th>[UART/SPI] Enumeration for specifying RESET process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part name</td>
<td>Macro Name</td>
</tr>
<tr>
<td>e_reset_process_t</td>
<td>E_SAIC_EXTERNAL_RESET</td>
</tr>
<tr>
<td></td>
<td>E_SAIC_SPI_INTERNAL_RESET</td>
</tr>
<tr>
<td></td>
<td>E_SAIC_UART_INTERNAL_RESET</td>
</tr>
<tr>
<td></td>
<td>E_SAIC_POWERON_RESET</td>
</tr>
<tr>
<td></td>
<td>E_SAIC_RESET_MAX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4-18</th>
<th>[UART/SPI] Enumeration for specifying SAIC101 unique commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part name</td>
<td>Macro Name</td>
</tr>
<tr>
<td>e_saic101_func_t</td>
<td>E_REGISTER_READ</td>
</tr>
<tr>
<td></td>
<td>E_REGISTER_WRITE</td>
</tr>
<tr>
<td></td>
<td>E_REGISTER_BURST_READ</td>
</tr>
<tr>
<td></td>
<td>E_REGISTER_BURST_WRITE</td>
</tr>
<tr>
<td></td>
<td>E_REGISTER_ALL_WRITE_FROM_FLASH</td>
</tr>
<tr>
<td></td>
<td>E_BUFFER_REFRESH</td>
</tr>
<tr>
<td></td>
<td>E_FLASH_READ</td>
</tr>
<tr>
<td></td>
<td>E_FLASH_READ_1</td>
</tr>
<tr>
<td></td>
<td>E_FLASH_READ_2</td>
</tr>
<tr>
<td></td>
<td>E_FLASH_WRITE</td>
</tr>
<tr>
<td></td>
<td>E_FLASH_ALL_ERASE</td>
</tr>
<tr>
<td></td>
<td>E_FLASH_WRITE_VERIFY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4-19</th>
<th>[UART/SPI] Enumeration for specifying A/D converter input mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part name</td>
<td>Macro Name</td>
</tr>
<tr>
<td>e_adc_mode_t</td>
<td>E_ADC_DIFF</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SINGLE</td>
</tr>
</tbody>
</table>
### Table 4-20  [UART/SPI] Enumeration for specifying PGIA gain settings

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

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_adc_osr_t</td>
<td>E_ADC_OSR_64</td>
<td>00H</td>
<td>15625.000 [sps]</td>
</tr>
<tr>
<td></td>
<td>E_ADC_OSR_128</td>
<td>01H</td>
<td>7812.500 [sps]</td>
</tr>
<tr>
<td></td>
<td>E_ADC_OSR_256</td>
<td>02H</td>
<td>3906.250 [sps]</td>
</tr>
<tr>
<td></td>
<td>E_ADC_OSR_512</td>
<td>03H</td>
<td>1953.125 [sps]</td>
</tr>
<tr>
<td></td>
<td>E_ADC_OSR_1024</td>
<td>04H</td>
<td>976.563 [sps]</td>
</tr>
<tr>
<td></td>
<td>E_ADC_OSR_2048</td>
<td>05H</td>
<td>488.281 [sps]</td>
</tr>
<tr>
<td></td>
<td>E_ADC_OSR_MAX</td>
<td>06H</td>
<td>Determine maximum value</td>
</tr>
</tbody>
</table>
### Table 4-22 [UART/SPI] Enumeration for specifying DC offset

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

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_adc_ch_t</td>
<td>E_ADC_CH1</td>
<td>00H</td>
<td>Ch1</td>
</tr>
<tr>
<td></td>
<td>E_ADC_CH2</td>
<td>01H</td>
<td>Ch2</td>
</tr>
<tr>
<td></td>
<td>E_ADC_CH3</td>
<td>02H</td>
<td>Ch3</td>
</tr>
<tr>
<td></td>
<td>E_ADC_CH4</td>
<td>03H</td>
<td>Ch4</td>
</tr>
<tr>
<td></td>
<td>E_ADC_CH5</td>
<td>04H</td>
<td>Ch5 (temperature sensor)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_adc_onoff_t</td>
<td>E_ADC_OFF</td>
<td>01H</td>
<td>Turn A/D converter OFF</td>
</tr>
<tr>
<td></td>
<td>E_ADC_ON</td>
<td>00H</td>
<td>Turn A/D converter ON</td>
</tr>
</tbody>
</table>

Feb 01, 2015
### Table 4-25  [UART / SPI] Enumeration for setting ADSTART bit

<table>
<thead>
<tr>
<th>Part name</th>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_adc_start_t</td>
<td>E_ADC_STOP</td>
<td>00H</td>
<td>Stop A/D converter</td>
</tr>
<tr>
<td></td>
<td>E_ADC_START</td>
<td>01H</td>
<td>Start A/D converter</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>E_ADC_SBIAS_0p0</th>
<th>FFH</th>
<th>OFF (SENSEP=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_sbias_t</td>
<td>E_ADC_SBIAS_1p2</td>
<td>00H</td>
<td>1.2V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_1p3</td>
<td>01H</td>
<td>1.3V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_1p4</td>
<td>02H</td>
<td>1.4V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_1p5</td>
<td>03H</td>
<td>1.5V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_1p6</td>
<td>04H</td>
<td>1.6V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_1p7</td>
<td>05H</td>
<td>1.7V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_1p8</td>
<td>06H</td>
<td>1.8V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_1p9</td>
<td>07H</td>
<td>1.9V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_2p0</td>
<td>08H</td>
<td>2.0V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_2p1</td>
<td>09H</td>
<td>2.1V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_2p2</td>
<td>0AH</td>
<td>2.2V</td>
</tr>
<tr>
<td></td>
<td>E_ADC_SBIAS_MAX</td>
<td>0BH</td>
<td>Determine maximum value</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_response_state_t</td>
<td></td>
</tr>
<tr>
<td>E_GET_HEADER</td>
<td>Receive header</td>
</tr>
<tr>
<td>E_HEADER_Decode</td>
<td>Decode header</td>
</tr>
<tr>
<td>E_TYPE1</td>
<td>Process TYPE1 response</td>
</tr>
<tr>
<td>E_TYPE2</td>
<td>Process TYPE2 response</td>
</tr>
<tr>
<td>E_TYPE3</td>
<td>Process TYPE3 response</td>
</tr>
<tr>
<td>E_END</td>
<td>End</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_uart_setting_t</td>
<td></td>
</tr>
<tr>
<td>E_UART_4800bps_None</td>
<td>Baud rate=4800 bps, Parity = None</td>
</tr>
<tr>
<td>E_UART_4800bps_Odd</td>
<td>Baud rate=4800 bps, Parity = Odd</td>
</tr>
<tr>
<td>E_UART_4800bps_Even</td>
<td>Baud rate=4800 bps, Parity = Even</td>
</tr>
<tr>
<td>E_UART_250kbps_None</td>
<td>Baud rate=250000 bps, Parity = None</td>
</tr>
<tr>
<td>E_UART_250kbps_Odd</td>
<td>Baud rate=250000 bps, Parity = Odd</td>
</tr>
<tr>
<td>E_UART_250kbps_Even</td>
<td>Baud rate=250000 bps, Parity = Even</td>
</tr>
</tbody>
</table>
Table 4-29  [C source file definition, SPI] Enumeration for specifying ReadWrite

<table>
<thead>
<tr>
<th>Part name</th>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_rw_t</td>
<td>E_READ</td>
<td>00H</td>
<td>For read</td>
</tr>
<tr>
<td></td>
<td>E_WRITE</td>
<td>01H</td>
<td>For write</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saic101_adc_t</td>
<td><code>onoff</code></td>
<td>Enumeration variable for specifying A/D converter ON/OFF</td>
</tr>
<tr>
<td></td>
<td><code>input_mode</code></td>
<td>Enumeration variable for specifying A/D converter input mode</td>
</tr>
<tr>
<td></td>
<td><code>offset</code></td>
<td>Enumeration variable for specifying DC offset</td>
</tr>
<tr>
<td></td>
<td><code>over_sampling_rate</code></td>
<td>Enumeration variable for specifying over sampling rate</td>
</tr>
<tr>
<td></td>
<td><code>gain</code></td>
<td>Enumeration variable for specifying PGIA gain setting</td>
</tr>
<tr>
<td><code>count</code></td>
<td>A/D conversion count specifying for one AUTOSCAN cycle</td>
<td></td>
</tr>
</tbody>
</table>

(b) `r_sa_uart_control_register.h`

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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saic_data_t</td>
<td><code>address</code></td>
<td>SAIC control register address</td>
</tr>
<tr>
<td></td>
<td><code>data</code></td>
<td>SAIC control register data</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saic_data_bit_t</td>
<td><code>address</code></td>
<td>SAIC control register address</td>
</tr>
<tr>
<td></td>
<td><code>bit_number</code></td>
<td>SAIC control register bit number (0 to7)</td>
</tr>
<tr>
<td></td>
<td><code>data</code></td>
<td>SAIC control register bit data (0/1)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_saic_t</td>
<td><code>uart_ch</code></td>
<td>UART ch number</td>
</tr>
<tr>
<td></td>
<td><code>sa_type</code></td>
<td>SAIC part name</td>
</tr>
</tbody>
</table>
### Table 4-34  [UART] Structure for serial information storage (when using interrupt)

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_serial_t</td>
<td>Structure for serial information storage variables (stores pointers for start, stop, transmit, receive and other functions)</td>
<td>void (*)(void)</td>
<td>UART_Start</td>
<td>UARTxx_Start function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void (*)(void)</td>
<td>UART_Stop</td>
<td>UARTxx_Stop function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_STATUS(*)(uint8_t *, const rx_buf, uint16_t tx_num)</td>
<td>UART_Receive</td>
<td>UARTxx_Receive function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_STATUS(*)(uint8_t *, const tx_buf, uint16_t tx_num)</td>
<td>UART_Send</td>
<td>UARTxx_Send function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t (*)(uint8_t *, packet_data, uint8_t rx_buffer[], uint16_t read_pos)</td>
<td>UART_GetHeader</td>
<td>UARTxx_GetHeader function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t (*)(uint16_t tx_cnt)</td>
<td>UART_Getdata</td>
<td>UARTxx_Getdata function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void (*)(uint8_t setting)</td>
<td>UART_SettingChange</td>
<td>UARTxx_SettingChange function pointer</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_reset_t</td>
<td>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</td>
<td>e_reset_process_t</td>
<td>process</td>
<td>Specifies SAIC RESET method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t *</td>
<td>p_reset_addr</td>
<td>Address of port register connected to RESET pin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>reset_bit_num</td>
<td>Bit number of port register connected to RESET pin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint32_t</td>
<td>nop_cnt</td>
<td>NOP command count for wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>uart_saic_num</td>
<td>Buffer number of SA storage variables *Use for RESET and other commands in UART communications. Not specified for common RESET pin, etc.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_saic_t</td>
<td>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)</td>
<td>e_csi_ch</td>
<td>csi_ch</td>
<td>CSI channel number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e_saic_type_t</td>
<td>sa_type</td>
<td>SAIC type (part name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t *</td>
<td>p_cs_addr</td>
<td>Address of port register connected to CS pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>cs_bit_num</td>
<td>Bit number of port register connected to CS pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t *</td>
<td>p_int_addr</td>
<td>Address of port register connected to INT pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>int_bit_num</td>
<td>Bit number of port register connected to INT pin</td>
</tr>
</tbody>
</table>

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

(c)  r_sa_spi_control_register.h

(c)  r_sa_spi_control_register.h

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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_saic_t</td>
<td>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)</td>
<td>e_csi_ch</td>
<td>csi_ch</td>
<td>CSI channel number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e_saic_type_t</td>
<td>sa_type</td>
<td>SAIC type (part name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t *</td>
<td>p_cs_addr</td>
<td>Address of port register connected to CS pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>cs_bit_num</td>
<td>Bit number of port register connected to CS pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t *</td>
<td>p_int_addr</td>
<td>Address of port register connected to INT pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>int_bit_num</td>
<td>Bit number of port register connected to INT pin</td>
</tr>
</tbody>
</table>

Note: xx indicates the serial channel number used for communication with SAIC.
Table 4-37  [SPI] Structure for serial information storage (when using interrupt)

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_serial_t</td>
<td>Structure for serial information storage variables (stores pointers for start, stop, and R/W execution functions)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>void (*) (void)</td>
<td>CSI_Start</td>
<td>CSIxx_Start function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void (*) (void)</td>
<td>CSI_Stop</td>
<td>CSIxx_Stop function pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_STATUS (*)(uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf)</td>
<td>CSI_Send_Receive</td>
<td>CSIxx_Send_Receive function pointer</td>
</tr>
</tbody>
</table>

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)

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

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

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_reset_t</td>
<td>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)</td>
<td>e_reset_process_t</td>
<td>process</td>
<td>Specifies SAIC RESET method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t *</td>
<td>p_reset_addr</td>
<td>Address of port register connected to RESET pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>reset_bit_num</td>
<td>Bit number of port register connected to RESET pin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint32_t</td>
<td>nop_cnt</td>
<td>NOP command count for wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8_t</td>
<td>spi_saic_num</td>
<td>Buffer number of SA storage variables *Use for RESET and other commands in SPI communications. Not specified for common RESET pin, etc.</td>
</tr>
</tbody>
</table>
4.8 Unions

This section describes the union declarations defined in the API.

(a) \textit{r\_sa\_uart\_control\_register.h / r\_sa\_spi\_control\_register.h}

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

<table>
<thead>
<tr>
<th>Structure Name</th>
<th>uni_reg_t</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>No. of bits</td>
<td>Name</td>
</tr>
<tr>
<td>uint8_t</td>
<td>8</td>
<td>BYTE</td>
</tr>
<tr>
<td>CHIPCNT_BIT</td>
<td>1 (lower)</td>
<td>slp</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>senspd</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>aregp</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>psthr</td>
</tr>
<tr>
<td></td>
<td>2 (upper)</td>
<td>-</td>
</tr>
<tr>
<td>VSBIAS_BIT</td>
<td>4</td>
<td>sbias</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>INTFLAG_BIT</td>
<td>1</td>
<td>fr</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>fw</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>fae</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>raw</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>adc</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>STATUS_BIT</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>fwp</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>faeip</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>rawip</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>adcip</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>ADC_CNT_BIT</td>
<td>5</td>
<td>axincdc</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>adstart</td>
</tr>
<tr>
<td>CHxCNT1_BIT</td>
<td>5</td>
<td>offset</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>input_mode</td>
</tr>
<tr>
<td>CHxCNT2_BIT</td>
<td>5</td>
<td>gain</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>osr</td>
</tr>
<tr>
<td>CHxCNT3_BIT</td>
<td>5</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>high</td>
</tr>
<tr>
<td>STARTUP_BIT</td>
<td>1</td>
<td>cpsor</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>sdcor</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>tglsm</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 4-41 [UART/SPI] Union for UART communication functions (word access)

<table>
<thead>
<tr>
<th>Structure Name</th>
<th>uni_adc_reg_t</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>For word access</td>
</tr>
<tr>
<td><strong>Member variable</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>No. of bits</strong></td>
</tr>
<tr>
<td>uint16_t</td>
<td>16</td>
</tr>
<tr>
<td>BYTE</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Structure Name</th>
<th>uni_adcc_t</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Bit field union for SAIC101 ADCC information storage variables</td>
</tr>
<tr>
<td><strong>Member variable</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>No. of bits</strong></td>
</tr>
<tr>
<td>uint8_t</td>
<td>8</td>
</tr>
<tr>
<td>BIT</td>
<td>4 (lower)</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3 (upper)</td>
</tr>
</tbody>
</table>

(b) r_sa_uart_control_register.h

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

<table>
<thead>
<tr>
<th>Structure Name</th>
<th>rcv_packet_t</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>For analyzing received data packets</td>
</tr>
<tr>
<td><strong>Member variable</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>No. of bits</strong></td>
</tr>
<tr>
<td>uint8_t</td>
<td>8</td>
</tr>
<tr>
<td>HEADER</td>
<td>5 (lower)</td>
</tr>
<tr>
<td></td>
<td>3 (upper)</td>
</tr>
<tr>
<td>TYPE1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>TYPE2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>TYPE3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Table 4-44  [UART/SPI] Union for UART communication functions (BYTE access)

<table>
<thead>
<tr>
<th>Structure name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uni_sum_t</td>
<td>For byte access</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member variable</th>
<th>Type</th>
<th>No. of bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uint8_t</td>
<td>8</td>
<td>BYTE</td>
<td>8-bit access</td>
</tr>
<tr>
<td>BIT</td>
<td>4 (lower)</td>
<td>low</td>
<td>Lower 4 bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 (upper)</td>
<td>high</td>
<td>Upper 4 bits</td>
<td></td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_uart_serial_data_tbl</td>
<td>const uart_serial_t g_uart_serial_data_tbl[] = { constant value }</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uart_serial_t structure setting</th>
<th>Type</th>
<th>User Setting</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>void (*)</td>
<td>(void)</td>
<td>Define the UART start function name generated by the code generator R_UART1_Start</td>
</tr>
<tr>
<td></td>
<td>void (*)</td>
<td>(void)</td>
<td>Define the UART stop function name generated by the code generator R_UART1_Stop</td>
</tr>
<tr>
<td></td>
<td>MD_STATUS (*)</td>
<td>(const uint8_t * const rx_buf, uint16_t rx_num)</td>
<td>Define the UART receive function name generated by the code generator R_UART1_Receive</td>
</tr>
<tr>
<td></td>
<td>MD_STATUS (*)</td>
<td>(const uint8_t * const tx_buf, uint16_t tx_num)</td>
<td>Describe the UART transmission function name generated by the code generator R_UART1_Send</td>
</tr>
<tr>
<td></td>
<td>uint8_t (*)</td>
<td>(uint8_t *packet_data, uint8_t *rx_buffer[], uint16_t read_pos)</td>
<td>Define the header data acquisition function name included in the sample code R_UART1_GetHeader</td>
</tr>
<tr>
<td></td>
<td>uint8_t (*)</td>
<td>(uint16_t rx_cnt)</td>
<td>Define the number of received data packets check function included in the sample code R_UART1_Getdata</td>
</tr>
<tr>
<td></td>
<td>void (*)</td>
<td>(uint8_t *setting)</td>
<td>Define the name of the UART setting change function included in the sample code R_UART1_SettingChange</td>
</tr>
</tbody>
</table>
(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.

<table>
<thead>
<tr>
<th>Type</th>
<th>User Setting</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>e_uart_ch_t</code></td>
<td>Define the element number corresponding to the <code>ENUM e_uart_ch_t</code> for specifying the global variable to store serial module information</td>
<td><code>E_UART1</code></td>
</tr>
<tr>
<td><code>e_saic_type_t</code></td>
<td>Define the element number corresponding to the <code>ENUM e_saic_type_t</code> for specifying the SAIC type (part name)</td>
<td><code>E_SAIC101</code></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Type</th>
<th>User Setting</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_reset_process_t</td>
<td>Define the SAIC “Reset by:”</td>
<td>E_SAIC_POWERON_RESET</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>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.</td>
<td>NULL</td>
</tr>
<tr>
<td>uint8_t</td>
<td>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.</td>
<td>0U</td>
</tr>
<tr>
<td>uint32_t</td>
<td>Number of NOP() executions for wait</td>
<td>D_PON_RST_NOP_CNT</td>
</tr>
<tr>
<td>uint8_t</td>
<td>This is the SAIC number when using an internal “Reset by:”. 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.)</td>
<td>0U</td>
</tr>
</tbody>
</table>

Declaration examples

Case 1

SAIC101 is connected to UART1. “RESET by:” is set to power-on reset.

```c
r_sa_uart_control_register.h
typedef enum
{
  E_SAIC300 = 0x00U, /* SAIC300 */
  E_SAIC301, /* SAIC301 */
  E_SAIC500, /* SAIC500 */
  E_SAIC501, /* SAIC501 */
  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.h
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[] =
{
  { E_SAIC101, /* SAIC information when SAIC number = 0 */
  },
};

const uart_reset_t g_uart_reset_data_tbl[] =
{
  { E_SAIC_POWERON_RESET, NULL, 0U, D_PON_RST_NOP_CNT, 0U, /* First RESET by: */
  },
};
```

R21AN0015EJ0100 Rev.1.00
Feb 01, 2015
● Case 2

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

```c
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;
```

```c
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;
```

```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, },
};
```

```c
const uart_saic_t g_uart_saic_data_tbl[] =
{ 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 */
};
```

```c
const uart_reset_t g_uart_reset_data_tbl[] =
{ E_SAIC_POWERON_RESET, NULL, 0U, D_PON_RST_NOP_CNT, /* 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

<table>
<thead>
<tr>
<th>Type</th>
<th>User Setting</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>void (*) (void)</td>
<td>Define the SPI start function name generated by the code generator</td>
<td>R_CSI10_Start</td>
</tr>
<tr>
<td>void (*) (void)</td>
<td>Define the SPI stop function name generated by the code generator</td>
<td>R_CSI10_Stop</td>
</tr>
<tr>
<td>MD_STATUS (*) (uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf)</td>
<td>Define the SPI transmit/receive function name generated by the code generator</td>
<td>R_CSI10_Send_Receive</td>
</tr>
</tbody>
</table>

#### When using register polling

<table>
<thead>
<tr>
<th>Type</th>
<th>User Setting</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t *</td>
<td>Define the address of the transmission data register corresponding to the CSI channel</td>
<td>&amp;SIO10</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>Define the address of the receive data register corresponding to the CSI channel</td>
<td>&amp;SIO10</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>Define the address of the interrupt request flag register corresponding to the CSI channel</td>
<td>&amp;IF1L</td>
</tr>
<tr>
<td>uint8_t</td>
<td>Define the bit number of the interrupt request flag register corresponding to the CSI channel</td>
<td>1U</td>
</tr>
<tr>
<td>uint16_t *</td>
<td>Define the address of the SMR register corresponding to the CSI channel</td>
<td>(uint16_t *)&amp;SSR02</td>
</tr>
<tr>
<td>uint16_t *</td>
<td>Define the address of the SIR register corresponding to the CSI channel</td>
<td>(uint16_t *)&amp;SIR02</td>
</tr>
<tr>
<td>void (*) (void)</td>
<td>Define the name of the SPI start function generated in the code generator</td>
<td>R_CSI10_MaskStart</td>
</tr>
<tr>
<td>void (*) (void)</td>
<td>Define the name of the SPI stop function generated in the code generator</td>
<td>R_CSI10_Stop</td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>Type</th>
<th>User Setting</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_csi_ch_t</td>
<td>Define the CSI channel number.</td>
<td>E_CSI10</td>
</tr>
<tr>
<td>e_saic_type_t</td>
<td>Define the SAIC type (part name)</td>
<td>E_SAIC101</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>Define the address of the port register connected to the CS pin.</td>
<td>&amp;P0</td>
</tr>
<tr>
<td>uint8_t</td>
<td>Define the bit number of the port register connected to the CS pin.</td>
<td>6U</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>Define the address of the port register connected to the INT pin.</td>
<td>&amp;P0</td>
</tr>
<tr>
<td>uint8_t</td>
<td>Define the bit number of the port register connected to the INT pin.</td>
<td>7U</td>
</tr>
</tbody>
</table>
(c) Global constant to store RESET information (SPI)

Constant name  g_spi_reset_data_tbl
Declaration  
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.

<table>
<thead>
<tr>
<th>Type</th>
<th>User Setting</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_reset_process_t</td>
<td>Define the SAIC “Reset by:”.</td>
<td>E_SAIC_POWERON_RESET</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>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.</td>
<td>NULL</td>
</tr>
<tr>
<td>uint8_t t</td>
<td>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.</td>
<td>0U</td>
</tr>
<tr>
<td>uint32_t</td>
<td>Number of NOP() executions for wait</td>
<td>D_PON_RST_NOP_CNT</td>
</tr>
<tr>
<td>uint8_t t</td>
<td>This is the SAIC number when using an internal “Reset by:”. 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.)</td>
<td>0U</td>
</tr>
</tbody>
</table>

(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.

  ```c
  typedef enum {
    E_SAIC300 = 0x00U, /* SAIC300 */
    E_SAIC301, /* SAIC301 */
    E_SAIC302, /* SAIC302 */
    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.h
  const spi_saic_t g_spi_saic_data_tbl[ ] = {
    /* SAIC information when SAIC number = 0 */
    { E_CSI10,   E_SAIC101, 0x00, 0x06, Epiry, 0x07, }, /* CSI10 */
  };

  const spi_serial_t g_spi_serial_data_tbl[ ] = {
    /* format */
    { NULL, 0x00U, 0x00U, 0x00U, 0x00U, 0x00U, }, /* CSI10 */
  };

  const uart_reset_t g_uart_reset_data_tbl[ ] = {
    /* format */
    { E_SAIC_POWERON_RESET, NULL, 0x00U, D_PON_RST_NOP_CNT, 0x00U, }, /* 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.

```c
const spi_saic_t g_spi_saic_data_tbl[] =
{ /* format */
  { E_CSI10, E_SAIC101, &P1, 6U, &P1, 7U, }, /* SAIC101 information when SAIC number = 0 */
  { E_CSI10, E_SAIC500, &P1, 4U, NULL, 0U, }, /* SAIC500 information when SAIC number = 1 */
  { E_CSI00, E_SAIC501, &P2, 0U, NULL, 0U, }, /* SAIC501 information when SAIC number = 2 */
};
```

```c
const spi_serial_t g_spi_serial_data_tbl[] =
{ /* format */
  { &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 */
};
```

```c
const spi_reset_t g_spi_reset_data_tbl[] =
{ /* 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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint8_t</td>
<td>g_uart_saic_data_tbl_size</td>
<td>Number of elements of global constant to store SAIC information</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>g_uart_reset_data_tbl_size</td>
<td>Number of elements of global constant to store RESET information</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint8_t</td>
<td>g_uart_flash_data_tbl_size</td>
<td>Number of elements of g_uart_smartanalog_flash_data</td>
</tr>
<tr>
<td>const saic_data_t</td>
<td>g_uart_smartanalog_flash_data[]</td>
<td>Initialization example of write data to register shadow area in flash memory</td>
</tr>
</tbody>
</table>

Table 4-47 Global constants to define CPU wait time

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint16_t</td>
<td>g_uart_4800bps_half_bit_time</td>
<td>Counter value of number of loops for time equivalent to half the 1-bit width time (about 105 us) at 4800bps. Calculated from the value defined in &quot;D_UART_4800BPS_HALF_BIT&quot; Note</td>
</tr>
<tr>
<td>const uint16_t</td>
<td>g_uart_250kbps_half_bit_time</td>
<td>Counter value of number of loops for time equivalent to half the 1-bit width time (about 2 us) at 250kbps. Calculated from the value defined in &quot;D_UART_250kbps_HALF_BIT&quot; Note</td>
</tr>
<tr>
<td>const uint16_t</td>
<td>g_uart_response_step</td>
<td>Counter value of number of loops for time for calculating the number of steps for functions waiting for a response. Calculated from the value defined in &quot;D_UART_10MS&quot; Note</td>
</tr>
<tr>
<td>const uint32_t</td>
<td>g_uart_saic_auto_adc_timeout</td>
<td>Counter value of number of loops for settling time when OSR = 2048 (8.192 ms) Calculated from the value defined in &quot;D_UART_3US_NOP_CNT&quot; Note</td>
</tr>
<tr>
<td>const uint16_t</td>
<td>g_uart_5ms_NOP_cnt</td>
<td>Counter value of number of loops for SBIAS to stop and then wait for stabilization after transfer to sleep mode (at least 5 ms) Calculated from the value defined in &quot;D_UART_5MS&quot; Note</td>
</tr>
<tr>
<td>const uint16_t</td>
<td>g_uart_1800us_NOP_cnt</td>
<td>Counter value of number of loops for AREG to start and then wait for stabilization (at least 1800 us) Calculated from the value defined in &quot;D_UART_1800US&quot; Note</td>
</tr>
<tr>
<td>const uint16_t</td>
<td>g_uart_270us_NOP_cnt</td>
<td>Counter value of number of loops for return from sleep mode and then wait for stabilization (at least 270 us) Calculated from the value defined in &quot;D_UART_270US&quot; Note</td>
</tr>
<tr>
<td>const uint16_t</td>
<td>g_uart_250us_NOP_cnt</td>
<td>Counter value of number of loops for SBIAS to start and then wait for stabilization (at least 250 us) Calculated from the value defined in &quot;D_UART_250US&quot; Note</td>
</tr>
</tbody>
</table>

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

Table 4-48 Private global constant for bit operations

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint8_t</td>
<td>gs_bit_tbl[]</td>
<td>Power-of-two data table constant. Stores data corresponding to index.</td>
</tr>
</tbody>
</table>
4.10.2 SPI Control

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint8_t</td>
<td>g_spi_saic_data_tbl_size</td>
<td>Number of elements of global constant to store SAIC information</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>g_spi_reset_data_tbl_size</td>
<td>Number of elements of global constant to store RESET information</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint8_t</td>
<td>g_spi_flash_data_tbl_size</td>
<td>Number of elements of g_spi_smartanalog_flash_data</td>
</tr>
<tr>
<td>const saic_data_t</td>
<td>g_spi_smartanalog_flash_data[]</td>
<td>Initialization example of write data to register shadow area in flash memory</td>
</tr>
</tbody>
</table>

Table 4-51 Global constants to define CPU wait time

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

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

Table 4-52 Private Global constant for bit operations

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global constant name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint8_t</td>
<td>gs_bit_tbl[]</td>
<td>Power-of-two data table constants. Stores data corresponding to index.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Part name</th>
<th>Global Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static uint8_t</td>
<td>gs_adc_1shot</td>
<td>ADC 1Shot acquisition flag: 1 = ADC 1Shot acquired, 0 = Other</td>
</tr>
<tr>
<td>static uint16_t</td>
<td>gs_uart_half_bit_time</td>
<td>Time equivalent to half the 1-bit width</td>
</tr>
<tr>
<td>static uint16_t</td>
<td>gs_uart_negotiation_timeout</td>
<td>Communication negotiation timeout value (depends on the baud rate)</td>
</tr>
</tbody>
</table>
5. Communication-related Definitions

5.1 API Function Specifications

5.1.1 [UART/SPI] Smart Analog initialization function

```c
void R_SAIC_UART_Init(void)
void R_SAIC_SPI_Init(void)
```

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

<table>
<thead>
<tr>
<th>Outline</th>
<th>Read register bytes function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>- For UART</td>
</tr>
<tr>
<td></td>
<td>r_sa_uart_control_register.h</td>
</tr>
<tr>
<td></td>
<td>- For SPI</td>
</tr>
<tr>
<td></td>
<td>r_sa_spi_control_register.h</td>
</tr>
<tr>
<td>Argument</td>
<td>saic_num:</td>
</tr>
<tr>
<td></td>
<td>SAIC number</td>
</tr>
<tr>
<td></td>
<td>*data:</td>
</tr>
<tr>
<td></td>
<td>Data buffer pointer</td>
</tr>
<tr>
<td></td>
<td>num:</td>
</tr>
<tr>
<td></td>
<td>Number of data units</td>
</tr>
<tr>
<td>Global Variable</td>
<td>- For UART</td>
</tr>
<tr>
<td></td>
<td>g_uart_saic_data_tbl_size:</td>
</tr>
<tr>
<td></td>
<td>Number of stored SAIC information entries</td>
</tr>
<tr>
<td></td>
<td>g_uart_half_bit_time:</td>
</tr>
<tr>
<td></td>
<td>Stop bit completion counter</td>
</tr>
<tr>
<td></td>
<td>- For SPI</td>
</tr>
<tr>
<td></td>
<td>g_spi_saic_data_tbl_size:</td>
</tr>
<tr>
<td></td>
<td>Number of stored SAIC information entries</td>
</tr>
<tr>
<td>Return Value</td>
<td>saic_status_t:</td>
</tr>
<tr>
<td></td>
<td>D_SAIC_OK or D_SAIC_ERR_PARAM or D_SAIC_ERR_COM</td>
</tr>
<tr>
<td>Description</td>
<td>Error processing branches have been omitted from flowchart.</td>
</tr>
</tbody>
</table>

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.

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
(int8_t saic_num, saic_data_t *data, uint8_t num, uint8_t *err_index)

saic_status_t R_SAIC_SPI_WriteVerify
(int8_t saic_num, saic_data_t *data, uint8_t num, uint8_t *err_index)

Outline
Write-verify 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
*err_index:
  Index value of the saic_data array that caused an error

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_VERIFY or D_SAIC_ERR_PARAM

Description
Error processing branches have been omitted from flowchart.

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.

---

**Flowchart:**

1. Check argument
2. Execute UART_SettingChange (function pointer)
3. Execute read register bytes function to UARTCNT register
4. **Successful?**
   - **Yes:**
     - Un-executed negotiation function settings specified?
     - **No:**
       - If argument saic_uartcnt_set is not 1, processing is completed.
       - Changes UART setting from value read from UARTCNT register to 250kbps/Parity:Odd.
6. MCU setting other than 250kbps/Parity:Odd?
   - **Yes:**
     - Executes write register bytes function to UARTCNT register
     - If function fails, returns corresponding error and goes to end.
     - Executes UART_SettingChange (function pointer), and changes MCU’s UART setting to 250kbps/Parity:Odd.
   - **No:**
     - MCU setting other than 250kbps/Parity:Odd?
     - **Yes:**
       - Executes UART_SettingChange (function pointer), and changes MCU’s UART communication setting.
     - Argument saic_uartcnt_set is not 1, processing is completed.
     - Changes UART setting from value read from UARTCNT register to 250kbps/Parity:Odd.
     - **No:**
       - Determines negotiation timeout.
     - Goes to step 2 if it 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

```c
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.

- 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

---

**Flowchart Diagram**

```
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
   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
   Y
End
```
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:
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.

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:
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

Return Value
saic_status_t:
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.

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:
SAIC number

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

Return Value
None

Description
Error processing branches have been omitted from flowchart.

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

Global Variable
g_spi_saic_data_tbl_size:
Number of stored SAIC information entries

g_spi_saic_data_tbl[]:
Global variable to store SAIC information

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

Return Value
uint8_t:
0 = enabled, 1 = disabled

Description
1. If argument error occurs and function pointer is NULL, process goes to end.
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:
Corresponding CSI channel number

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

Return Value
None

Description
Error processing branches have been omitted from flowchart.

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.

Start

1. Using same channel as specified serial channel?

N

2. CS = H

Y

3. Specified number of stored SAIC information entries processed?

N

Y

End
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)

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

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)

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

<table>
<thead>
<tr>
<th>Outline</th>
<th>Smart Analog power-on RESET wait function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>- For UART</td>
</tr>
<tr>
<td></td>
<td>r_sa_uart_control_register.h</td>
</tr>
<tr>
<td></td>
<td>- For SPI</td>
</tr>
<tr>
<td></td>
<td>r_sa_spi_control_register.h</td>
</tr>
<tr>
<td>Argument</td>
<td>reset_num:</td>
</tr>
<tr>
<td></td>
<td>Array number of global variable to store RESET information</td>
</tr>
<tr>
<td>Global Variable</td>
<td>- For UART</td>
</tr>
<tr>
<td></td>
<td>g_uart_reset_data_tbl[]:</td>
</tr>
<tr>
<td></td>
<td>Global variable to store RESET information</td>
</tr>
<tr>
<td></td>
<td>- For SPI</td>
</tr>
<tr>
<td></td>
<td>g_spi_reset_data_tbl[]:</td>
</tr>
<tr>
<td></td>
<td>Global variable to store RESET information</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>

5.2.4 [UART/SPI] NOP execution function

static void r_nop_wait(uint32_t nop_cnt)

<table>
<thead>
<tr>
<th>Outline</th>
<th>NOP execution function. Executes the NOP() command for the number of times indicated in the argument of nop_cnt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>- For UART</td>
</tr>
<tr>
<td></td>
<td>r_sa_uart_control_register.h</td>
</tr>
<tr>
<td></td>
<td>- For SPI</td>
</tr>
<tr>
<td></td>
<td>r_sa_spi_control_register.h</td>
</tr>
<tr>
<td>Argument</td>
<td>nop_cnt:</td>
</tr>
<tr>
<td></td>
<td>Number of NOP executions</td>
</tr>
<tr>
<td>Global Variable</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Repeats NOP() commands for the number of times specified in nop_cnt.</td>
</tr>
</tbody>
</table>
### 5.2.5 [UART/SPI] Burst read function

```c
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  
  - 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  
  - `gs_uart_half_bit_time`: Stop bit completion counter

**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**
     - 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.

   - **Y**

End
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
  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
  gs_uart_half_bit_time:
    Stop bit completion counter

Return Value
saic_status_t:
  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.
2. Converts to communication format that corresponds to specified address and data.
3. Executes transmission/receive function.
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
  r_sa_uart_control_register.h
- For SPI
  r_sa_spi_control_register.h

Argument
e_func:
  Specify SAIC101 functions
"address:
  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`:
  - SAIC number
- `*address`:
  - Target register address
- `rw`:
  - Read/write specification

**Global Variable**
- `g_spi_saic_data_tbl[]`:
  - 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`:
  - SAIC number

**Global Variable**
- `g_spi_saic_data_tbl[]`:
  - Global variable to store SAIC information
- `g_csi_overrun_flag`:
  - SPI overrun flag variable
- `gs_bit_tbl[]`:
  - Power-of-two data table constant

**Return Value**
- `uint8_t`:
  - 0 OK or 1 overrun error

**Description**
1. Executes bit check of corresponding channels specified by the SPI overrun flag variable
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:
    SAIC number
    tx_buffer[]:
    Transmit buffer
    rx_buffer[]:
    Receive buffer
    length:
    Data length
Global Variable
    g_spi_saic_data_tbl[]:
    Global variable to store SAIC information
    g_spi_serial_data_tbl[]:
    Global variable to store serial module information
    gs_bit_tbl[]:
    Power-of-two data table constant
Return Value
    saic_status_t
    D_SAIC_OK or D_SAIC_ERR_COM
Description  Error processing branches have been omitted from flowchart.
Process when using interrupts

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 function and 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)

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

```c
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:** SAIC number
- **p_func_int:** INTFLG read register bits function
- **p_func_status:** STATUS read register bits function

**Global Variable**
- **g_spi_saic_data_tbl[]:** Global variable to store SAIC information
- **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.

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:
           SAIC number
  tx_buffer[]:
           Transmit buffer
  rx_buffer[]:
           Receive buffer
  tx_length:
           Transmission size
  p_rx_length:
           Reception size

Global Variable  g_uart_saic_data_tbl[]:
                 Global variable to store SAIC information
  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
  g_uart_serial_data_tbl:
                 UART table

Return Value  saic_status_t:
               D_SAIC_OK or D_SAIC_ERR_COM

Description  1. If function pointer is NULL, returns D_SAIC_ERR_COM and goes to end.
  2. Executes UART reception function and sets reception wait status.
  3. Executes UART communication start function.
  4. Executes command transmission function, sends transmit buffer data according to transmission size.
  5. If command transmission function fails, sets corresponding error.
  6. If command transmission function is successful, executes UART receive data packet analysis function.
  7. Stops UART
  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:
           SAIC number
           tx_buffer[]:
           Transmit buffer
           tx_length:
           Transmission 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
                      g_uart_tx_end_flag:
                      UART transmission completion flag variable
                      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.

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.

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 received data 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.

Start

1. Initialize reception state

2. Determine reception state

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

End

Completed? N Y

N

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 received data 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

```c
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

<table>
<thead>
<tr>
<th>SAIC101 command function. Process branches to one of the following based on e_func argument: register burst-read/write function/flash memory function.</th>
</tr>
</thead>
</table>

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)

<table>
<thead>
<tr>
<th>Outline</th>
<th>Read flash data function. Issues [Flash (Burst) Read] commands.</th>
</tr>
</thead>
</table>
| 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. |
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 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
g_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.

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 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.
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_sai_uart_control_register.h
- For SPI
  r_sai_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
  g_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.

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

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

```c
static saic_status_t r_saic_spi_flash_read_fr_bit(uint8_t saic_num, uint8_t *fr_bit)
```

<table>
<thead>
<tr>
<th>Outline</th>
<th>INTFLAG register FR bit acquisition function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td><code>r_sa_spi_control_register.h</code></td>
</tr>
</tbody>
</table>
| Argument              | saic_num:
|                       | SAIC number                                 |
|                       | *fr_bit:
|                       | FR 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 INTFLAG register and reads data. |
|                       | 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

```c
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:
SAIC number
*fw_bit:
FW 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 INTFLAG register and reads data.
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

```c
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:
SAIC number
*fae_bit:
FAE 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 INTFLAG register and reads data.
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

```c
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:
SAIC number
*raw_bit:
RAW 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 INTFLAG register and reads data.
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

```c
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

```c
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

```c
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.
### System Function Specifications

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

```c
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)
```

<table>
<thead>
<tr>
<th>Outline</th>
<th>Write flash data address 01H function. Implements settings for power supply.</th>
</tr>
</thead>
</table>
| 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 01H which cannot be rewritten (bits 7, 6, 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 function.  
|            |     - For SPI  
|            |         Executes data transmission/reception execution function.  
|            | 5. If data transmission/reception function fails, returns corresponding error and goes to end.  
|            |     - 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.
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_vars:

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
  g_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.
   - 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**

```c
saic_status_t R_SAIC_UART_ADC_Start(uint8_t saic_num)

saic_status_t R_SAIC_SPI_ADC_Start(uint8_t saic_num)
```

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

```c
saic_status_t R_SAIC_UART_ADC_Stop(uint8_t saic_num)
saic_status_t R_SAIC_SPI_ADC_Stop(uint8_t saic_num)
```

<table>
<thead>
<tr>
<th>Outline</th>
<th>A/D conversion stop process function</th>
</tr>
</thead>
</table>
| 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**

```c
(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**

```c
(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
  ```c
  r_sa_uart_control_register.h
  ```
- For SPI
  ```c
  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
  ```c
  g_uart_saic_data_tbl_size:
  ```
  Number of stored SAIC information entries
  ```c
  g_uart_saic_data_tbl[]:
  ```
  Global variable to store SAIC information
  ```c
  gs_bit_tbl[]:
  ```
  Power-of-two data table constant
  ```c
  g_uart_rx_end_flag:
  ```
  UART reception completion flag variable
  ```c
  g_uart_serial_data_tbl:
  ```
  Global variable to store serial module information.

- For SPI
  ```c
  g_spi_saic_data_tbl_size:
  ```
  Number of stored SAIC information entries
  ```c
  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.
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.

2. If number of A/D in argument =0, returns SAIC_ERR_PARAM.
3. Executes A/D conversion start function
   - For SPI: Execute A/D conversion start 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 function fails, returns corresponding error and goes to end.
5. Waits for completion of A/D conversion start function.
   - For SPI: If INT pin is used, waits for INT interrupt; if not (polling is used), ADCIP flag polling.
   - SPI: If function fails, returns corresponding error and goes to end.
6. If polling monitoring function fails, returns corresponding error and goes to end.
   - For SPI: If function fails, returns corresponding error and goes to end.
   - 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.
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.
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
   r_sa_uart_control_register.h
- For SPI
   r_sa_spi_control_register.h

Argument
saic_num:
   SAIC number
ch:
   Target channel number for A/D conversion
*adc_value:
   ADC value storage buffer

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
   gs_adc_1shot:
      ADC 1Shot acquisition flag: 1 = ADC 1Shot acquired, 0 = Other
- 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. Generates ADCCNT register setting data based on argument channel.
3. Executes write register bytes function, only enables input multiplexer.
4. If write register bytes function fails, returns corresponding error and goes to end.
5. Generates CHxCNT3 setting data based on argument channel.
6. Executes write register bytes function, sets number of A/D conversions for specified channel to 1.
7. If write register bytes function fails, returns corresponding error and goes to end.
8. A/D conversion and reception processing
   - For SPI: Executes A/D conversion start process function
     If A/D conversion start process function fails, returns corresponding error and goes to end.
     Waits for completion of A/D conversion start process function.
     If INT pin is used, waits for INT interrupt; if not (polling is used), ADCIP flag polling.
     If polling fails while waiting for process to complete, returns corresponding error and goes to end.
     Executes burst read function and reads A/D-converted value.
     If burst read function fails, returns corresponding error and goes to end.
   - For UART: Executes read register function to ADCCNT register and reads data.
     Assigns 1 to ADSTART bit of read value.
     Assigns 1 to gs_adc_1shot variable.
     Executes command transmission & response reception function.
10. If checksum error occurs, returns D_SAIC_ERR_COM and goes to end.
11. Assigns A/D conversion results to *adc_value argument.
### 7.1.6 [UART] A/D-converted value received data acquisition function

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

**Outline**  
A/D-converted 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:  
Setting 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_DECODER state and repeats analysis.

---

**Flowchart**

1. Initialize state
2. Check argument
3. Execute UART reception start function.
4. Received at least 1 byte?
   - Y: Go to E_HEADER_DECODER state
   - N: Not type 2 response?
     - Y: Type 2 data packet received?
       - Y: Checksum does not match?
         - N: Return value in call by reference
         - Y: Go to E_GET_HEADER state
       - N: Go to E_HEADER_DECODER state
     - N: Go to E_GET_HEADER state
     - Y: Go to E_HEADER_DECODER state
   - N: Go to E_GET_HEADER state

---

**Legend**

- Start
- End
- State
- N
- Y

---

R21AN0015EJ0100 Rev.1.00  
Page 81 of 105  
Feb 01, 2015
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
  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_1800us_nop_cnt:
    Counter value for AREG operation stabilization wait (at least 1800 us)
- For SPI
  g_spi_saic_data_tbl_size:
    Number of stored SAIC information entries
  g_spi_1800us_nop_cnt:
    Counter value for AREG operation stabilization wait (at least 1800 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. Generates transmission data from value read from CHIPCNT register.
   First command: sets AREGPD bit to 0, SLP bit to 1
   Second command: sets AREGPD bit to 0, SLP bit to 1 (same as first command)
   Third command: sets SLP bit to 0
5. Executes write register bytes function and writes data.
6. If write fails, returns corresponding error and goes to end.
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

```c
saic_status_t R_SAIC_UART_AregOff(uint8_t saic_num)

saic_status_t R_SAIC_SPI_AregOff(uint8_t saic_num)
```

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

<table>
<thead>
<tr>
<th>Outline</th>
<th>SBIAS register setting function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- For UART</td>
</tr>
<tr>
<td></td>
<td>r_sa_uart_control_register.h</td>
</tr>
<tr>
<td></td>
<td>- For SPI</td>
</tr>
<tr>
<td></td>
<td>r_sa_spi_control_register.h</td>
</tr>
<tr>
<td>Argument</td>
<td>saic_num:</td>
</tr>
<tr>
<td></td>
<td>SAIC number</td>
</tr>
<tr>
<td></td>
<td>sbias:</td>
</tr>
<tr>
<td></td>
<td>SBIAS output voltage setting</td>
</tr>
<tr>
<td>Global Variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- For UART</td>
</tr>
<tr>
<td></td>
<td>g_uart_saic_data_tbl_size:</td>
</tr>
<tr>
<td></td>
<td>Number of stored SAIC information entries</td>
</tr>
<tr>
<td></td>
<td>g_uart_5ms_nop_cnt:</td>
</tr>
<tr>
<td></td>
<td>Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms)</td>
</tr>
<tr>
<td></td>
<td>g_uart_250us_nop_cnt:</td>
</tr>
<tr>
<td></td>
<td>Counter value for SBIAS operation stabilization wait (at least 250 us)</td>
</tr>
<tr>
<td></td>
<td>- For SPI</td>
</tr>
<tr>
<td></td>
<td>g_spi_saic_data_tbl_size:</td>
</tr>
<tr>
<td></td>
<td>Number of stored SAIC information entries</td>
</tr>
<tr>
<td></td>
<td>g_spi_5ms_nop_cnt:</td>
</tr>
<tr>
<td></td>
<td>Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms)</td>
</tr>
<tr>
<td></td>
<td>g_spi_250us_nop_cnt:</td>
</tr>
<tr>
<td></td>
<td>Counter value for SBIAS operation stabilization wait (at least 250 us)</td>
</tr>
<tr>
<td>Return Value</td>
<td>saic_status_t:</td>
</tr>
<tr>
<td></td>
<td>D_SAIC_OK or D_SAIC_ERR_COM or D_SAIC_ERR_PARAM</td>
</tr>
</tbody>
</table>

Description

Error processing branches have been omitted from flowchart.
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.

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
  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_5ms_nop_cnt:
    Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms)
- For SPI
  g_spi_saic_data_tbl_size:
    Number of stored SAIC information entries
  g_spi_5ms_nop_cnt:
    Counter value for SBIAS stop and sleep mode operation stabilization wait (at least 5 ms)

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 1 to SLP bit of CHIPCNT register read in step 2.
5. Executes write register bytes function and writes to CHIPCNT register.
6. Waits at least 5 ms for stabilization.
## 8.1.6 [UART/SPI] Sleep mode OFF setting function

```c
saic_status_t R_SAIC_UART_SleepModeOff(uint8_t saic_num)
```

```c
saic_status_t R_SAIC_SPI_SleepModeOff(uint8_t saic_num)
```

<table>
<thead>
<tr>
<th>Outline</th>
<th>Sleep mode OFF setting function</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Power Supply Configuration</th>
<th>Power supply Voltage during operations</th>
<th>Changes to register setting for normal operations</th>
<th>For Flash Memory Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SAIC101</td>
<td>MCU</td>
<td>Power supply configuration changes</td>
</tr>
<tr>
<td>Configuration 1</td>
<td>3.3 to 5.5V (same electric potential)</td>
<td>Not required</td>
<td>Not required</td>
</tr>
<tr>
<td>Configuration 2</td>
<td>3.3~5.5V</td>
<td>3.0V</td>
<td>Not required</td>
</tr>
<tr>
<td>Configuration 3</td>
<td>2.7 to 3.6V (same electric potential)</td>
<td>Required</td>
<td>Required</td>
</tr>
</tbody>
</table>

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

- 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 to 5.5V when power supply is less than 4.5V.

- Software settings for flash programming: not required
9.1.4 Power supply configuration 2 (normal operations)

Figure 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

![Diagram of power supply configuration for normal operations]

Figure 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.

- Software settings for flash programming
  
  Set the PSTHRU bit of the CHIPCNT register to 1U and connect the RVDD pin and AREG pin in the IC.

---

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

---

**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

![Power supply configuration for normal operations diagram]

- Software settings for normal operations
Set the AREGPD bit of the CHIPCNT register to 1U, and stop AREG operations.

![Software settings for normal operations diagram]

— 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.

![Figure 9-8 Power supply configuration for flash programming](image)

- 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](image)

---

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

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

Note: Stopped only during SPI mode.
9.2.2 Control module

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

```
start
CHIPCNT.SLP ← [0U]
CHIPCNT.SENSPD ← [1U]
end
```

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

- Transition timing

<table>
<thead>
<tr>
<th>Module</th>
<th>Item</th>
<th>max</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIAS</td>
<td>Turn off time</td>
<td>(5)</td>
<td>ms</td>
<td>$V_{OUT} &lt; 10%$</td>
</tr>
</tbody>
</table>

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.

```
start
CHIPCNT.SLP ← [0U]
CHIPCNT.SENSPD ← [0U]
N
250 us passed?
Y
end
```

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

- Transition timing

<table>
<thead>
<tr>
<th>Module</th>
<th>Item</th>
<th>max</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIAS</td>
<td>Turn on time</td>
<td>(250)</td>
<td>$\mu$s</td>
<td>$V_{OUT} &gt; 90%$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Item</th>
<th>max</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIAS</td>
<td>Turn off time</td>
<td>(5)</td>
<td>ms</td>
<td>$V_{OUT} &lt; 10%$</td>
</tr>
<tr>
<td>CREG</td>
<td>Mode switch time 1</td>
<td>(400)</td>
<td>$\mu$s</td>
<td>Normal operations $\rightarrow$ wait mode</td>
</tr>
<tr>
<td>VREF</td>
<td>-</td>
<td>-</td>
<td>$\mu$s</td>
<td>Stops only in SPI mode</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Item</th>
<th>max</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIAS</td>
<td>Turn on time</td>
<td>(250)</td>
<td>$\mu$s</td>
<td>$V_{OUT} &gt; 90%$</td>
</tr>
<tr>
<td>CREG</td>
<td>Mode switch time 2</td>
<td>(150)</td>
<td>$\mu$s</td>
<td>Wait mode $\rightarrow$ normal operations Not available when VREF is turned on</td>
</tr>
<tr>
<td>VREF</td>
<td>Turn on time</td>
<td>(550)</td>
<td>$\mu$s</td>
<td>SPI mode only</td>
</tr>
<tr>
<td>All (SPI)</td>
<td>Wake up time</td>
<td>(820)</td>
<td>$\mu$s</td>
<td>Duration between sleep mode cleared (SLP = 1U$\rightarrow$0U) and A/D conversion ready</td>
</tr>
</tbody>
</table>

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](image)

- 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](image)

- Transition timing

<table>
<thead>
<tr>
<th>Module</th>
<th>Item</th>
<th>max</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREG</td>
<td>Turn on time</td>
<td>(1800)</td>
<td>μs</td>
<td>(RV_{DD} \geq 3.3) V, (V_{OUT} &gt; 90%), (I_{OUT} = 0) mA</td>
</tr>
</tbody>
</table>

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 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

<table>
<thead>
<tr>
<th>CPSOR(^{\text{Note 2}})</th>
<th>SDCOR(^{\text{Note 2}})</th>
<th>SLP(^{\text{Note 2}})</th>
<th>SENSPD(^{\text{Note 2}})</th>
<th>ADSTART(^{\text{Note 2}})</th>
<th>SMODE pin</th>
<th>Operations after SAIC101 Startup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy register shadow</td>
<td>256-byte transmission</td>
<td>Sleep mode control</td>
<td>SBIAS operation control</td>
<td>AD conversion operation</td>
<td>Serial mode selection</td>
<td>(Copy register shadow operation abbreviated)</td>
</tr>
<tr>
<td>0 = None</td>
<td>0 = None</td>
<td>0 = operating</td>
<td>0 = operating</td>
<td>0 = stop</td>
<td>L = UART, H = SPI</td>
<td></td>
</tr>
<tr>
<td>1 = copy</td>
<td>1 = transmisson</td>
<td>1 = sleep</td>
<td>1 = stopped</td>
<td>1 = start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>L</td>
<td></td>
<td>A/D conversion operating</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>A/D results atuo transfer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>H</td>
<td>A/D conversion operating</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SBIAS operation stopped</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>-</td>
<td></td>
<td></td>
<td>Sleep mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>L</td>
<td>256-byte auto-transmission</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>L</td>
<td></td>
<td>256-byte auto-transmission</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>L</td>
<td></td>
<td>A/D conversion operating</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>A/D results auto transfer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>-</td>
<td></td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>256-byte auto-transmission</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SBIAS operation stopped</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>-</td>
<td></td>
<td></td>
<td>SBIAS operation stopped</td>
</tr>
</tbody>
</table>

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/

All trademarks and registered trademarks are the property of their respective owners.
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev.1.00</td>
<td>Feb 01, 2015</td>
<td>---</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
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.
Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the products quality grade, as indicated below.

- **Standard**: Computers, office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

- **High Quality**: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (not only the support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunction under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging-degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final product or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas products or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in the document. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

Refer to “http://www.renesas.com/” for the latest and detailed information.

Renesas Electronics Corporation

http://www.renesas.com

SALES OFFICES

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95054-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Nepean, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-6220

Renesas Electronics Europe Limited
Dukes Meadow, Billbrook Road, Bourne End, Buckinghamshire, SL8 5FH, UK
Tel: +44-1628-595-100, Fax: +44-1628-595-930

Renesas Electronics Europe GmbH
Arcadiaoffice 10, 40472 Düsseldorf, Germany
Tel: +49-211-6003-0, Fax: +49-211-6003-1327

Renesas Electronics (China) Ltd.
Room 1709, Quantum Plaza, No.67 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, CEFII Tower, 555 Kangqiao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-5228-0988, Fax: +86-21-5226-0999

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: +852-2265-6888, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F., No.362, Fu Shing North Road, 1, Tadji, 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886-2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit B09-02 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: +60-3-7955-9330, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Tahan-ro, Gangnam-gu, Seoul, 135-920, Korea
Tel: +82-2-558-3137, Fax: +82-2-558-3141

© 2014 Renesas Electronics Corporation. All rights reserved.

Colophon 4.0