Smart Analog IC101

Useful Examples of SAIC101 Sample Code

R21AN0014EJ0100
Rev.1.00
Nov 01, 2014

Introduction

This application note describes general examples for a sample code using API functions to control Smart Analog IC101 (RAA730101).

Note: Smart Analog IC101 is referred to as "SAIC101" throughout this document.

Target Device

Smart Analog IC 101 (part name: RAA730101), RL78/L13 (part name: R5F10WMGAFB)

Contents

1. Overview

This application note describes the following typical usage examples for the sample code using API functions for controlling the Smart Analog IC101 ("SAIC101" herein): register operations (read, write)Note, flash memory operations (read, write verify), and A/D convertor control. The document also describes an application example using the sensor mounted on the Smart Analog IC RSK Option Evaluation Board TSA-OP-IC101, which has an onboard Smart Analog IC and can be used with the Renesas Starter Kit.

Note: Register read/write processes using SPI communications can be used for other Smart Analog devices as well.

2. Operation Confirmation Conditions

Operations for these usage examples have been confirmed under the following conditions.

Table 2-1 Operation Confirmation Conditions

Item	Description
Evaluation board	• Renesas Starter Kit for RL78/L13 [R0K5010WMS900BE] — Renesas Starter Kit for RL78/L13 CPU Board Abbreviation: RSK CPU Board — Renesas Starter Kit LCD Application Board V2 Abbreviation: LCD Extension Board • Smart Analog IC RSK Option Evaluation Board [TSA-OP-IC101] Abbreviation: TSA-OP-IC101
Target device	R5F10WMGAFB (RL78/L13)
Operating frequency	24MHz
Operating voltage	5.0V
Integrated Development Environment (CubeSuite+)	V2.02.00 [21 Feb 2014]
C Compiler (CubeSuite+)	CA78K0R V4.02.00.03 [16 Jan 2014]
Integrated Development Environment (e2studio)	V3.0.0.22
C Compiler (e2studio)	GNURL78 v14.01

3. Usage Example Instructions

3.1 Register Operations

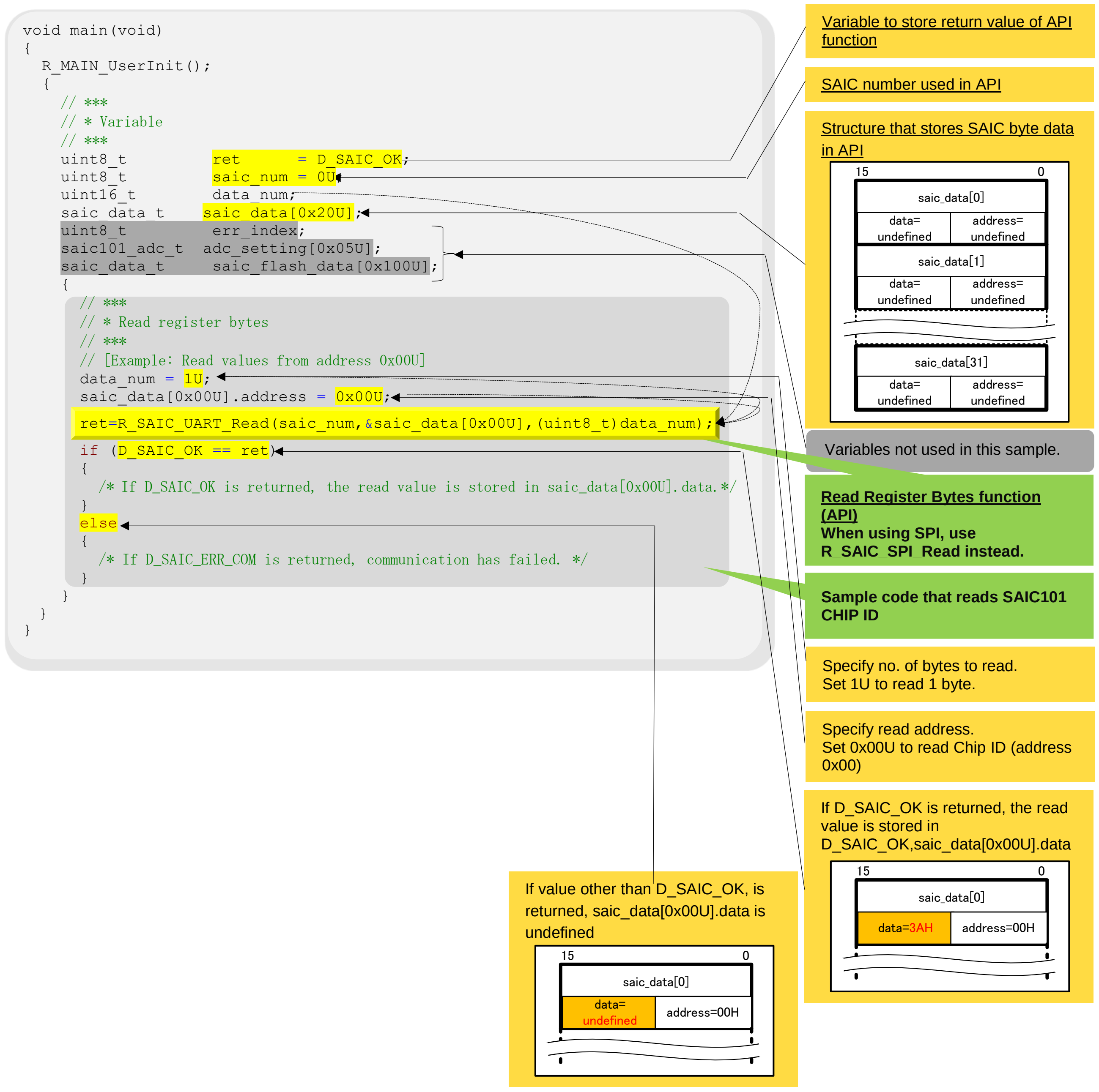
3.1.1 Read Register Bytes (SPI/UART)

This usage example reads bytes of data from SAIC101^Note^ registers.

SAIC101 API read register bytes function [R_SAIC_SPI_Read] (for SPI) or [R_SAIC_UART_Read] (for UART) is used to read values from the SAIC101 CHIPID register (address 0x00). The CHIPID register is reserved for reading the stored SAIC101 chip ID; it reads 0x3A (a fixed value). When 0x3A is read, this serves as a confirmation that the serial communication connection and read register process function are operating correctly.

Note: SPI communications described here can be used for other Smart Analog devices as well.

- Sample Code (for UART)



```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * Variable
    // ***
    uint8_t         ret      = D_SAIC_OK;
    uint8_t         saic_num = 0U;
    uint16_t        data_num;
    saic_data_t    saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t  adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * Read register bytes
      // ***
      // [Example: Read values from address 0x00U]
      data_num = 1U;
      saic_data[0x00U].address = 0x00U;
      ret=R_SAIC_UART_Read(saic_num,&saic_data[0x00U],(uint8_t)data_num);
      if (D_SAIC_OK == ret)
      {
        /* If D_SAIC_OK is returned, the read value is stored in saic_data[0x00U].data.*/
      }
      else
      {
        /* If D_SAIC_ERR_COM is returned, communication has failed. */
      }
    }
  }
}
```

Variable to store return value of API function

SAIC number used in API

Structure that stores SAIC byte data in API

Variables not used in this sample.

Read Register Bytes function (API)
When using SPI, use R_SAIC_SPI_Read instead.

Sample code that reads SAIC101 CHIP ID

Specify no. of bytes to read.
Set 1U to read 1 byte.

Specify read address.
Set 0x00U to read Chip ID (address 0x00)

If D_SAIC_OK is returned, the read value is stored in D_SAIC_OK,saic_data[0x00U].data

If value other than D_SAIC_OK, is returned, saic_data[0x00U].data is undefined

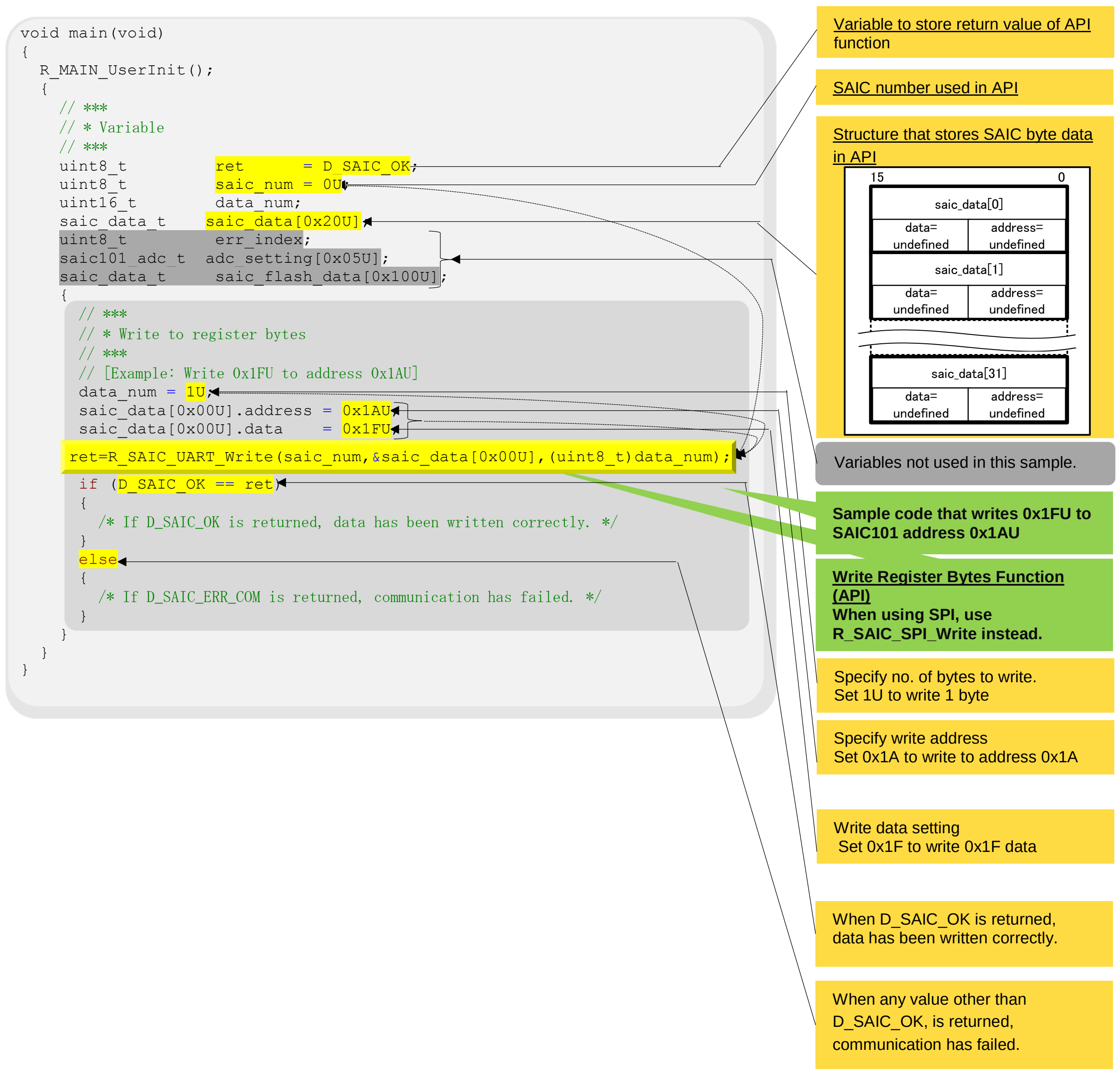
3.1.2 Write Register Bytes (SPI/UART)

This usage example writes bytes of data to SAIC101$^{\text{Note}}$registers.

SAIC101 API write register bytes function [R_SAIC_SPI_Write] (for SPI) or [R_SAIC_UART_Write] (for UART) is used to write 0x1F to the SAIC101 CH4CNT2 register (address 0x1A), a writable register. The lower 5 bits of the CH4CNT2 register store the DC offset value for channel 4 of the SAIC101 input multiplexer.

Note: The SPI used in this example can be used for other Smart Analog devices as well.

- Sample Code (for UART)



```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * Variable
    // ***
    uint8_t          ret       = D_SAIC_OK;
    uint8_t          saic_num  = 0U;
    uint16_t         data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t          err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t      saic_flash_data[0x100U];
    {
      // ***
      // * Write to register bytes
      // ***
      // [Example: Write 0x1FU to address 0x1AU]
      data_num = 1U;
      saic_data[0x00U].address  = 0x1AU;
      saic_data[0x00U].data     = 0x1FU;
     ret=R_SAIC_UART_Write(saic_num,&saic_data[0x00U],(uint8_t)data_num);
      if (D_SAIC_OK == ret)
      {
        /* If D_SAIC_OK is returned, data has been written correctly. */
      }
      else
      {
        /* If D_SAIC_ERR_COM is returned, communication has failed. */
      }
    }
  }
}
```

Variable to store return value of API function
SAIC number used in API
Structure that stores SAIC byte data in API
15
0
saic_data[0]
data= undefined
address= undefined
saic_data[1]
data= undefined
address= undefined
saic_data[31]
data= undefined
address= undefined
Variables not used in this sample.
Sample code that writes 0x1FU to SAIC101 address 0x1AU
Write Register Bytes Function (API)
When using SPI, use R_SAIC_SPI_Write instead.
Specify no. of bytes to write. Set 1U to write 1 byte
Specify write address Set 0x1A to write to address 0x1A
Write data setting Set 0x1F to write 0x1F data
When D_SAIC_OK is returned, data has been written correctly.
When any value other than D_SAIC_OK, is returned, communication has failed.

3.2 Flash Memory Operations

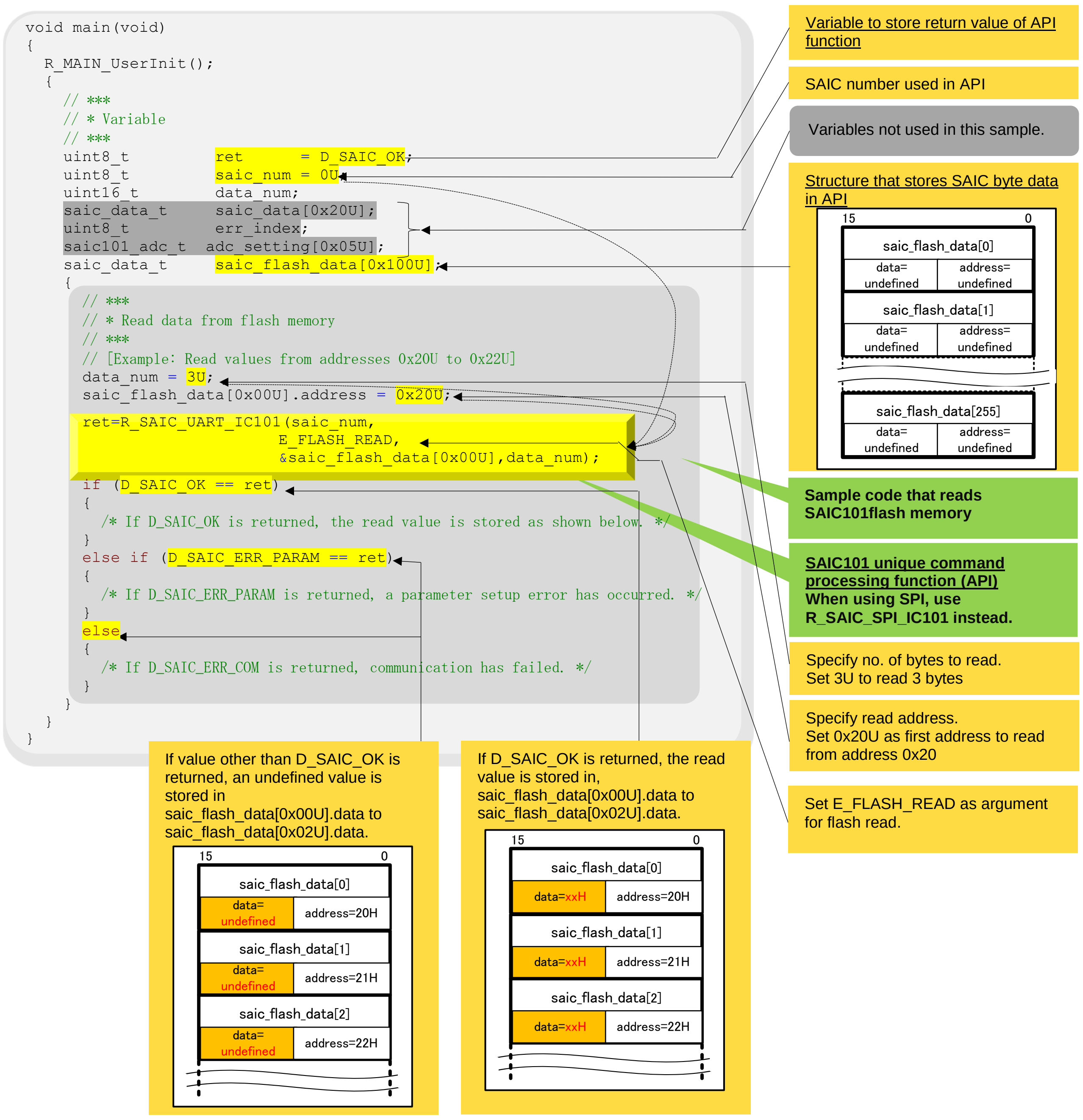
3.2.1 Read Flash Memory Data (SPI/UART)

This usage example reads bytes of data from the SAIC101 flash memory.

In this example, the SAIC101 API’s unique command processing function [R_SAIC_SPI_IC101] (for SPI) or [R_SAIC_UART_IC101] (for UART) reads values from addresses 0x20 to 0x22 in the user area of the SAIC101 flash memory.

Caution: This program is limited to a maximum size of 256 bytes for one flash memory read operation. In addition, when reading flash memory data of over 32 bytes in UART communications, set the size of data to be read to 32 bytes. For other limitations, please refer to Section 13. Flash Memory in the latest SAIC101 Data Sheet (R02DS0014E).

- Sample code (for UART)



```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * Variable
    // ***
    uint8_t         ret       = D_SAIC_OK;
    uint8_t         saic_num  = 0U;
    uint16_t        data_num;
    saic_data_t     saic_data[0x20U];
    uint8_t         err_index;
    saic101_adc_t  adc_setting[0x05U];
    saic_data_t     saic_flash_data[0x100U];
    {
      // ***
      // * Read data from flash memory
      // ***
      // [Example: Read values from addresses 0x20U to 0x22U]
      data_num = 3U;
      saic_flash_data[0x00U].address = 0x20U;

      ret=R_SAIC_UART_IC101(saic_num,
                            E_FLASH_READ,
                            &saic_flash_data[0x00U],data_num);
      if (D_SAIC_OK == ret)
      {
        /* If D_SAIC_OK is returned, the read value is stored as shown below. */
      }
      else if (D_SAIC_ERR_PARAM == ret)
      {
        /* If D_SAIC_ERR_PARAM is returned, a parameter setup error has occurred. */
      }
      else
      {
        /* If D_SAIC_ERR_COM is returned, communication has failed. */
      }
    }
  }
}
```

Variable to store return value of API function
SAIC number used in API
Variables not used in this sample.
Structure that stores SAIC byte data in API
15 0
saic_flash_data[0]
data= undefined
address= undefined
saic_flash_data[1]
data= undefined
address= undefined
saic_flash_data[255]
data= undefined
address= undefined
Sample code that reads SAIC101flash memory
SAIC101 unique command processing function (API)
When using SPI, use R_SAIC_SPI_IC101 instead.
Specify no. of bytes to read.
Set 3U to read 3 bytes
Specify read address.
Set 0x20U as first address to read from address 0x20
Set E_FLASH_READ as argument for flash read.
If value other than D_SAIC_OK is returned, an undefined value is stored in saic_flash_data[0x00U].data to saic_flash_data[0x02U].data.
15 0
saic_flash_data[0]
data= undefined
address=20H
saic_flash_data[1]
data= undefined
address=21H
saic_flash_data[2]
data= undefined
address=22H
If D_SAIC_OK is returned, the read value is stored in, saic_flash_data[0x00U].data to saic_flash_data[0x02U].data.
15 0
saic_flash_data[0]
data=xxH
address=20H
saic_flash_data[1]
data=xxH
address=21H
saic_flash_data[2]
data=xxH
address=22H

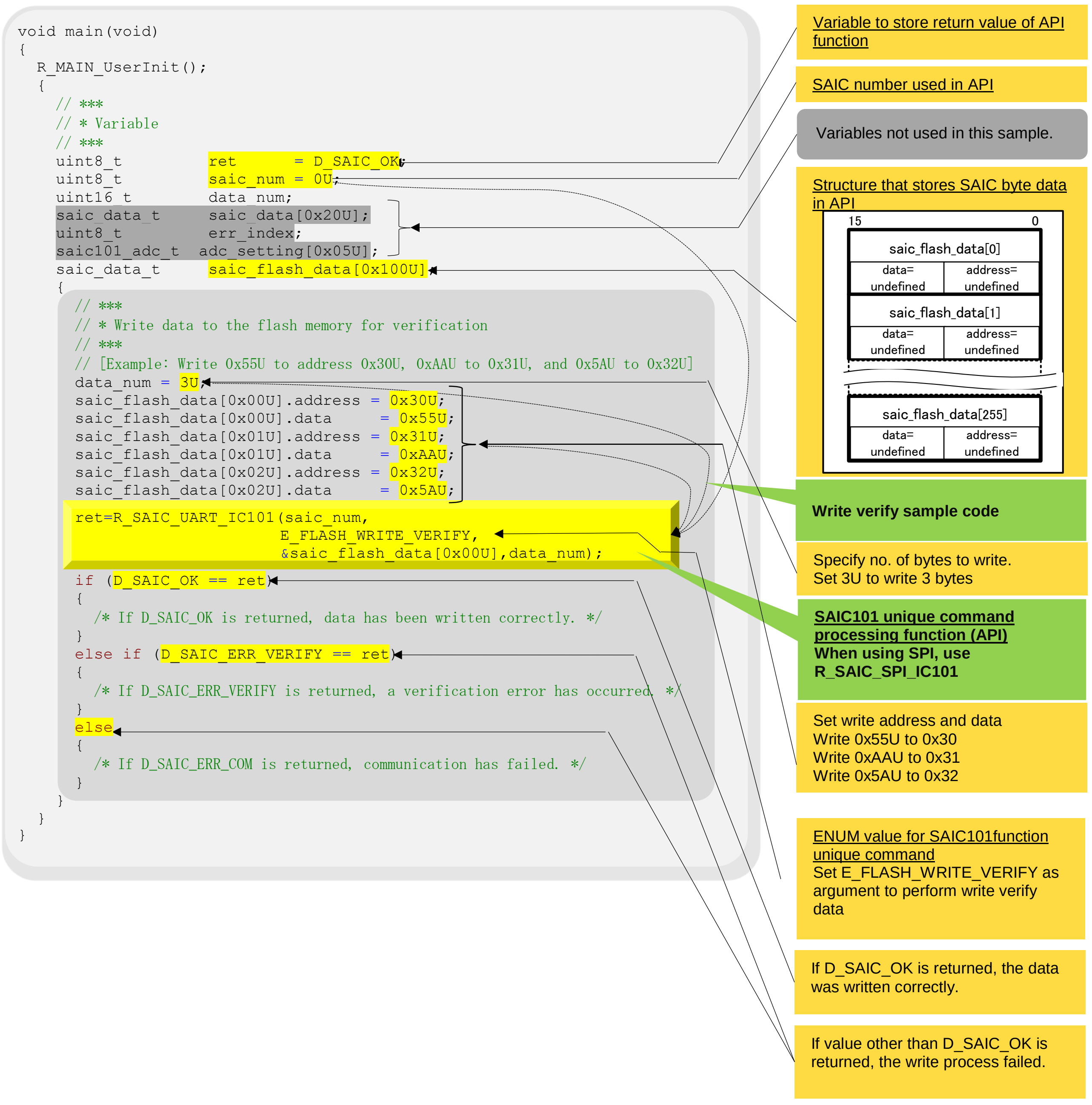
3.2.2 Write Verify Flash Memory Data

This usage example writes data to the SAIC101 flash memory and verifies the data.

SAIC101 operations are unaffected even when values in the flash memory users area (addresses 0x20 to 0xFF) are rewritten. This example uses SAIC101 unique command processing function [R_SAIC_UART_IC101] (for SPI) or [R_SAIC_UART_IC101](for UART) to write the value 0x55 to address 0x30, 0xAA to address 0x31, and 0x5A to address 0x32.

Caution: The SAIC101 API function used in this usage example writes data in single bytes. Therefore, the user must specify an address each time data is written to the SAIC data storage structure, the second argument. When writing any value other than 0x00, all data must be erased before the rewrite. A programming window period is established from startup after a power-on reset until the first A/D conversion starts. Flash memory programming is only valid during this period. For other limitations, please refer to **Section 13. Flash Memory** in the latest SAIC101 Data Sheet (R02DS0014E).

- Sample code (for UART)



```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * Variable
    // ***
    uint8_t          ret        = D_SAIC_OK;
    uint8_t          saic_num = 0U;
    uint16_t         data_num;
    saic_data_t      saic_data[0x20U];
    uint8_t          err_index;
    saic101_adc_t  adc_setting[0x05U];
    saic_data_t      saic_flash_data[0x100U];
    {
      // ***
      // * Write data to the flash memory for verification
      // ***
      // [Example: Write 0x55U to address 0x30U, 0xAAU to 0x31U, and 0x5AU to 0x32U]
      data_num = 3U;
      saic_flash_data[0x00U].address = 0x30U;
      saic_flash_data[0x00U].data     = 0x55U;
      saic_flash_data[0x01U].address = 0x31U;
      saic_flash_data[0x01U].data     = 0xAAU;
      saic_flash_data[0x02U].address = 0x32U;
      saic_flash_data[0x02U].data     = 0x5AU;
      ret=R_SAIC_UART_IC101(saic_num,
                            E_FLASH_WRITE_VERIFY,
                            &saic_flash_data[0x00U],data_num);
      if (D_SAIC_OK == ret)
      {
        /* If D_SAIC_OK is returned, data has been written correctly. */
      }
      else if (D_SAIC_ERR_VERIFY == ret)
      {
        /* If D_SAIC_ERR_VERIFY is returned, a verification error has occurred. */
      }
      else
      {
        /* If D_SAIC_ERR_COM is returned, communication has failed. */
      }
    }
  }
}
```

Variable to store return value of API function

SAIC number used in API

Variables not used in this sample.

Structure that stores SAIC byte data in API

Write verify sample code

Specify no. of bytes to write.
Set 3U to write 3 bytes

SAIC101 unique command processing function (API)
When using SPI, use R_SAIC_SPI_IC101

Set write address and data
Write 0x55U to 0x30
Write 0xAAU to 0x31
Write 0x5AU to 0x32

ENUM value for SAIC101function unique command
Set E_FLASH_WRITE_VERIFY as argument to perform write verify data

If D_SAIC_OK is returned, the data was written correctly.

If value other than D_SAIC_OK is returned, the write process failed.

3.3 A/D Converter Control

3.3.1 A/D-converted Value Acquire (one channel, one sampling: “1 shot”) (SPI/UART)

This usage example uses the SAIC101 A/D converter to convert and acquire data from one channel of the SAIC101 input multiplexer.

The example uses the following three functions: SAIC101 API A/D converter registers initial setup function [R_SAIC_SPI_ADC_InitRegSet] (for SPI) or [R_SAIC_UART_ADC_InitRegSet] (for UART); SBIAS register setting function [R_SAIC_SPI_SbiasRegSet] (for SPI) or [R_SAIC_UART_SbiasRegSet] (for UART); and A/D-converted value acquire function (1 shot) [R_SAIC_SPI_ADC_GetResult_1Shot] (for SPI) or [R_SAIC_UART_ADC_GetResult_1Shot] (for UART). Call the A/D-converted value acquire function (1 shot) one time to acquire the A/D-converted value of one sampling a one channel.

Caution: The A/D converter must be setup using the A/D converter registers initial setup function of the SAIC101 API before calling the A/D-converted value acquire function. The saic101_adc_t array of the 5 channels is required to set the A/D converter.

- Sample code (for UART)

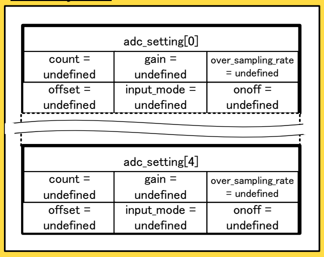
```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * Variable
    // ***
    uint8_t          ret       = D_SAIC_OK;
    uint8_t          saic_num = 0U;
    uint16_t         data_num;
    saic_data_t      saic_data[0x20U];
    uint8_t          err_index;
    saic101_adc_t   adc_setting[0x05U];
    saic_data_t      saic_flash_data[0x100U];
    {
      // ***
      // * Initial setup of A/D converter registers
      // ***
      // [Example: Disable SAIC101 input multiplexer channels 1 and 3 to 5, and enable channel
2. Set as follows.]
      uint8_t    count;
      /* Initialize all values set to channels */
      for (count=0U; count<5U; count++)
      {
        adc_setting[count].onoff                = E_ADC_OFF;
        adc_setting[count].input_mode          = E_ADC_DIFF;
        adc_setting[count].offset               = E_ADC_OFFSET_0p00;
        adc_setting[count].over_sampling_rate = E_ADC_OSR_256;
        adc_setting[count].gain                  = E_ADC_GAIN_1_1_1;
        adc_setting[count].count                 = 0x01U;
      }
      /* Set up channel 2 separately */
      adc_setting[E_ADC_CH2].gain             = E_ADC_GAIN_1_4_4;
      adc_setting[E_ADC_CH2].offset           = E_ADC_OFFSET_M153p13;
      ret = R_SAIC_UART_ADC_InitRegSet(saic_num, adc_setting);
```

Variable to store return value of API function

SAIC number used in API

Variables not used in this sample.

Variables stored in ADC information used by API



Sample code for [one channel (ch2) x 1 shot] for acquiring A/D converter value

Initial settings for variables stored in ADC information used by API:
See link for details.
- Disable A/D conversion
- Set to differential input mode
- Set DC offset to 0mV
- Set oversampling ratio to 256
- Set gain to x1
- Set number of A/D conversions to 1

Ch2 settings for variables stored in ADC information used in API:
- Set DC offset to -153.13/GSET1 [mV]
- Set gain to 1x4=4

A/D converter registers initial setup function (API)
When using SPI, use R_SAIC_SPI_ADC_InitRegSet

```
        if  (D_SAIC_OK == ret)
        {
            /* If D_SAIC_OK is returned, processing has finished correctly. */
            // ***
            // * Specify SBIAS register settings
            // ***
            // [Example: Set the SBIAS output value to 1.2 V]
            ret = R_SAIC_UART_SbiasRegSet(saic_num, E_ADC_SBIAS_1p2);
            if  (D_SAIC_OK == ret)
            {
                /* If D_SAIC_OK is returned, processing has finished correctly.
                // ***
                // * Acquire the A/D-converted value (1shot)
                // ***
                // [Example: Acquire the value A/D-converted on SAIC101 input multiplexer channel 2]
                uint16_t      ad_data_1shot;                  /* A/D-converted value */

                ret=R_SAIC_UART_ADC_GetResult_1Shot(saic_num,
                                                    E_ADC_CH2,
                                                    &ad_data_1shot);
                if  (D_SAIC_OK == ret)
                {
                    /* If D_SAIC_OK is returned, the A/D-converted value is stored in ad_data_1shot. */
                }
                else
                {
                    /* If D_SAIC_ERR_COM is returned, communication has failed. */
                }
            }
            else
            {
                /* If D_SAIC_ERR_COM is returned, communication has failed. */
            }
        }
        else
        {
            /* If D_SAIC_ERR_COM is returned, communication has failed. */
        }
    }
  }
}
```

If process is successful, SAIC101 A/D registers (CHxCNT1、CHxCNT2、CHxCNT3 *x=1 to 5) are overwritten.

ENUM value for setting SBIAS voltage
Set to 1.2V

If process is successful, 1.2V setting is stored in SBIAS register.

SBIAS register setting function (API)
When using SPI, use R_SAIC_SPI_SbiasRegSet instead.

A/D-converted value acquire function (1 shot) (API)
When using SPI, use R_SAIC_SPI_ADC_GetResult_1Shot instead.

ENUM value for setting CH number to be measured
Set ch2

If process is successful, A/D converted value is stored.

15	0
ad_data_1shot = A/D converted value	

If return value is not D_SAIC_OK, the sampling failed. A/D converted value is undefined.

15	0
ad_data_1shot = undefined	

If R_SAIC_UART_SbiasRegSet is returned (not D_SAIC_OK), SBIAS register settings are undefined.

If R_SAIC_UART_ADC_InitRegSet is returned (not D_SAIC_OK), A/D register settings are undefined.

3.3.2 A/D-converted Value Acquire (multi-channel, continuous sampling) (SPI/UART)

This usage example uses the SAIC101 A/D converter to convert and acquire data from multiple channels of the SAIC101 input multiplexer.

The example uses the following three functions: SAIC101 API A/D converter registers initial setup function [R_SAIC_SPI_ADC_InitRegSet] (for SPI) or [R_SAIC_UART_ADC_InitRegSet] (for UART); SBIAS register setting function [R_SAIC_SPI_SbiasRegSet] (for SPI) or [R_SAIC_UART_SbiasRegSet] (for UART); and A/D-converted value acquire function [R_SAIC_SPI_ADC_GetResult] (for SPI) or [R_SAIC_UART_ADC_GetResult] (for UART). Call the A/D-converted value acquire function one time to acquire the A/D-converted value for the number of times set in A/D conversion setting register 3of each channel.

When executing this sample code, the converted value is acquired by performing A/D conversion twice on all channels of the SAIC101 input multiplexer.

Caution: The A/D converter must be setup using the A/D converter registers initial setup function of the SAIC101 API before calling the A/D-converted value acquire function. The saic101_adc_t array of the 5 channels is required to set the A/D converter. The user should prepare the array size of the second and third arguments of the A/D-converted value acquire function to be equal to the total number of times the channels are processed.

- Sample code (for UART)

```
void main(void)
{
  R_MAIN_UserInit();
  {
    // ***
    // * Variable
    // ***
    uint8_t          ret        = D_SAIC_OK;
    uint8_t          saic_num = 0U;
    uint16_t         data_num;
    saic_data_t      saic_data[0x20U];
    uint8_t          err_index;
    saic101_adc_t  adc_setting[0x05U];
    saic_data_t      saic_flash_data[0x100U];
    {
      // ***
      // * Initial setup of A/D converter registers
      // ***
      // [Example: Enable SAIC101 input multiplexer channels 1 to 5. Set as follows.]
      uint16_t       ad_data[2U * 5U];
      uni_adcc_t     ad_adcc[2U * 5U];

      uint8_t        count;
      for (count=0U; count<5U; count++)
      {
        adc_setting[count].onoff                    = E_ADC_ON;
        adc_setting[count].input_mode             = E_ADC_SINGLE;
        adc_setting[count].offset                   = E_ADC_OFFSET_0p00;
        adc_setting[count].over_sampling_rate = E_ADC_OSR_128;
        adc_setting[count].gain                     = E_ADC_GAIN_1_1_1;
        adc_setting[count].count                    = 0x02U;
      }
      /* Initial setup of A/D converter registers */
      ret = R_SAIC_UART_ADC_InitRegSet(saic_num, adc_setting);
```

Variable to store return value of API function

SAIC number used in API

Variables not used in this sample

Variables stored in ADC information used by API

adc_setting[0]

count = undefined	gain = undefined	over_sampling_rate = undefined
offset = undefined	input_mode = undefined	onoff = undefined

adc_setting[4]

count = undefined	gain = undefined	over_sampling_rate = undefined
offset = undefined	input_mode = undefined	onoff = undefined

Sample code for continuous sampling (2 times) of A/D-converted value acquire channels

Variables to store A/D-converted value
5 channels x 2 times

Variables to store ADCC register value
5 channels x 2 times

Settings for variables stored in ADC information used by API:

- Enable A/D conversion
- Single-end input mode
- DC offset: 0 mV
- Oversampling ratio: 128
- Gain: x1
- Number of A/D conversions: 2

A/D converter registers initial setup function (API)
When using SPI, use R_SAIC_SPI_ADC_InitRegSet

```
      if  (D_SAIC_OK == ret)
      {
        /* If D_SAIC_OK is returned, processing has finished correctly. */
        // ***
        // * Specify SBIAS register settings
        // ***
        // [Example: Set the SBIAS output value to 1.2 V]
        ret = R_SAIC_UART_SbiasRegSet(saic_num, E_ADC_SBIAS_1p2);

        if  (D_SAIC_OK == ret)
        {
            /* If D_SAIC_OK is returned, processing has finished correctly. */
            // ***
            // * Acquire the A/D-converted value
            // ***
            /* Acquire the A/D-converted value */
            ret=R_SAIC_UART_ADC_GetResult(saic_num,ad_adcc,ad_data,(5U*2U));
            if ((D_SAIC_OK == ret) &&
             (ad_adcc[0x00U].BIT.ch==1U)&&(ad_adcc[0x01U].BIT.ch==1U)&&
             (ad_adcc[0x02U].BIT.ch==2U)&&(ad_adcc[0x03U].BIT.ch==2U)&&
             (ad_adcc[0x04U].BIT.ch==3U)&&(ad_adcc[0x05U].BIT.ch==3U)&&
             (ad_adcc[0x06U].BIT.ch==4U)&&(ad_adcc[0x07U].BIT.ch==4U)&&
             (ad_adcc[0x08U].BIT.ch==5U)&&(ad_adcc[0x09U].BIT.ch==5U))
            {
              /* If D_SAIC_OK is returned, the A/D-converted value is stored in ad_data,
and the ADCC register value is stored in ad_adcc. */
            }
            else
            {
              /* If D_SAIC_ERR_COM is returned, communication has failed. */
            }
        }
        else
        {
          /* If D_SAIC_ERR_COM is returned, communication has failed. */
        }
      }
      else
      {
        /* If D_SAIC_ERR_COM is returned, communication has failed. */
      }
    }
  }
}
```

If process is successful, SAIC101A/D control registers (CHxCNT1, CHxCNT2, CHxCNT3 ∗x=1 to 5) are overwritten.

ENUM value for setting SBIAS voltage
Set to 1.2V

If process is successful, 1.2V setting is stored in SBIAS register.

SBIAS register setting function (API)
When using SPI, use R_SAIC_SPI_SbiasRegSet instead.

A/D-converted value acquire function (API)
When using SPI, use R_SAIC_SPI_ADC_GetResult instead.

If successful, A/D-converted values are stored in ad_data[0] to ad_data[9].

15	0
ad_data[0] = ch1 1st A/D converted value	
ad_data[1] = ch1 2nd A/D converted value	
ad_data[2] = ch2 1st A/D converted value	
ad_data[3] = ch2 2nd A/D converted value	
ad_data[4] = ch3 1st A/D converted value	
ad_data[5] = ch3 2nd A/D converted value	
ad_data[6] = ch4 1st A/D converted value	
ad_data[7] = ch4 2nd A/D converted value	
ad_data[8] = ch5 1st A/D converted value	
ad_data[9] = ch5 2nd A/D converted value	

If successful, ADCC register values are stored in ad_adcc[0] to ad_adcc[9].

15	0
ad_adcc[0] = ch1 1st ADCC register value	
ad_adcc[1] = ch1 2nd ADCC register value	
ad_adcc[2] = ch2 1st ADCC register value	
ad_adcc[3] = ch2 2nd ADCC register value	
ad_adcc[4] = ch3 1st ADCC register value	
ad_adcc[5] = ch3 2nd ADCC register value	
ad_adcc[6] = ch4 1st ADCC register value	
ad_adcc[7] = ch4 2nd ADCC register value	
ad_adcc[8] = ch5 1st ADCC register value	
ad_adcc[9] = ch5 2nd ADCC register value	

If the return value is anything other than D_SAIC_OK, the values of ad_adcc[0] to ad_adcc[9] and ad_data[0] to ad_data[9] are undefined.

If R_SAIC_UART_SbiasRegSet is returned (not D_SAIC_OK), SBIAS register settings are undefined.

If R_SAIC_UART_ADC_InitRegSet is returned (not D_SAIC_OK), A/D register settings are undefined.

3.4 Application Example for Smart Analog IC RSK Option Evaluation Board

3.4.1 Thermistor Control

The following example demonstrates how to measure temperature using the thermistor on the Smart Analog IC RSK Option Evaluation Board (TSA-OP-IC101) by connecting the Renesas Starter Kit for RL78/L13 to the TSA-OP-IC101 board integrating a Smart Analog IC.

The thermistor, an onboard sensor, is attached to channel 2 of the SAIC101 input multiplexer on the TSA-OP-IC101 board. When the sample code is executed, the A/D conversion results of the thermistor output are displayed on the LCD panel enabling confirmation of A/D converter operations. In this example, the A/D-converted value decreases as the thermistor temperature increases. This effect can be visually confirmed by holding the thermistor between two fingers. As your fingers heat up the thermistor, the value on the LCD panel decreases. The range of A/D-converted values is between -32768 and 32767 as the differential input is 16 bits.

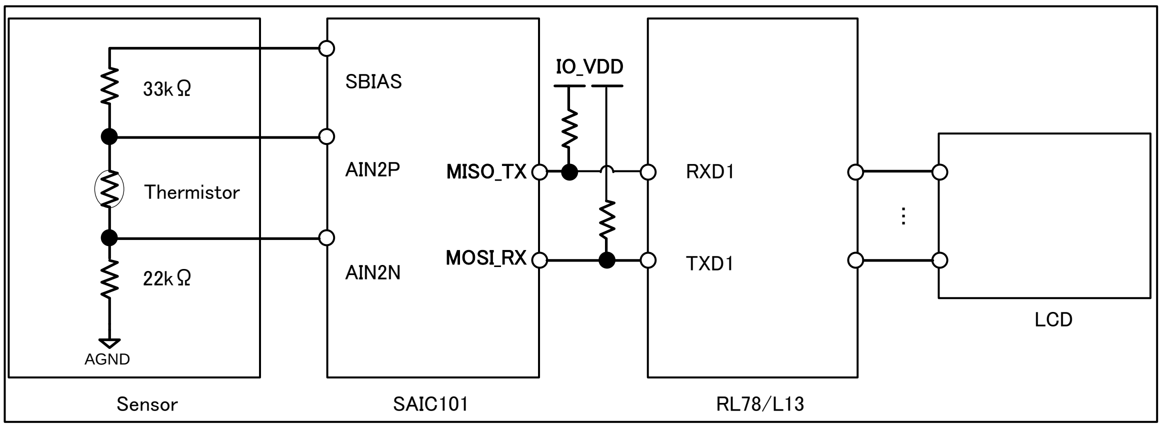


Figure 3-1 Block Diagram

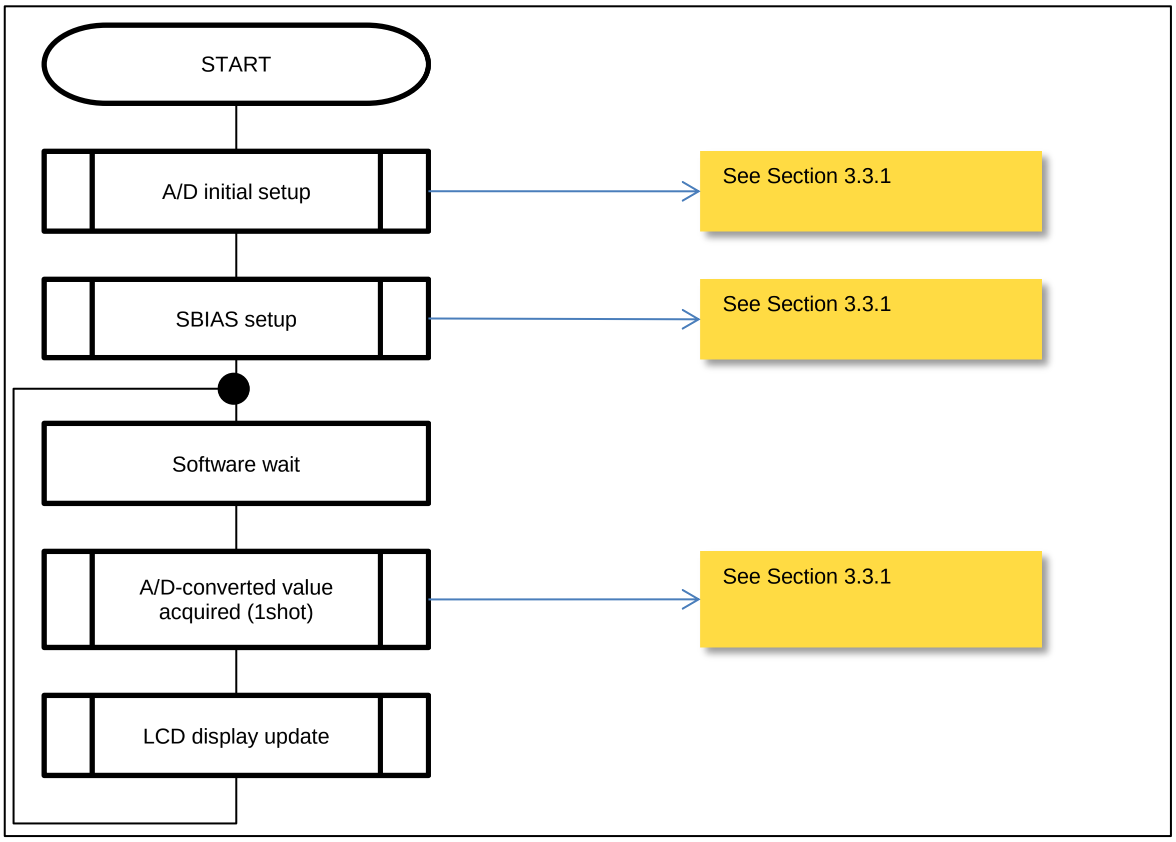


Figure 3-2 Flowchart

Website and Support

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

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

Revision History

Rev.	Date	Description	
		Page	Summary
Rev.1.00	Nov 01, 2014	---	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

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

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

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

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

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

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

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

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141