RX Family

ADC Module Using Firmware Integration Technology

R01AN1666EU0130
Rev. 1.30
Jun 5, 2014

Introduction

This module provides support for all features of the S12AD A/D Converter on the RX110, RX111, RX210, and the RX63N.

Target Device

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

- **RX110. RX111 Groups**
- **RX210 Group**
- **RX63N Group**

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU0100)
- Board Support Package Firmware Integration Technology Module (R01AN1685EU0250)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU0110)

Contents

1. Overview

This A/D Converter (ADC) driver supports the S12ADa peripheral on the RX63x and the S12ADb peripheral on the RX11x and RX210. The hardware functionality is detailed in Chapter 27 of the RX110 Hardware User's Manual, Chapter 30 of the RX111 Hardware User's Manual, Chapter 33 of the RX210 Hardware User's Manual, and in Chapter 42 of the RX63N Hardware User's Manual. The full functionality of the peripherals are supported by this driver. There is no dependency on any other software except for the board support package (r_bsp module).

The ADC begins conversion when it receives a trigger. When the conversion is complete, a flag is set and an interrupt issued if enabled. If the ADC is operating in a single scan mode, only one scan takes place per trigger. If the ADC is operating in a continuous mode, scans continue indefinitely after the initial trigger occurs.

The trigger source may be synchronous from an MTU peripheral, Event Link Controller (ELC), or a TPU peripheral (non-RX11x); asynchronous from an external trigger on ADTRG0, or from setting a bit in software. Note that even though a software trigger is an asynchronous action, the hardware manual reserves the term "asynchronous trigger" to refer to asynchronous external hardware triggers. Additionally, although the temperature sensor on the RX210 has its own independent trigger, the driver hides this operation and the application should be written as if the standard software trigger is being used.

The driver provides a Control() command for polled applications which checks to see if a scan completed after a trigger occurs. If interrupts are in use, the interrupt will call a Callback function specified in the Open() function. The only argument to the Callback function specifies whether the default scan has completed or a Group B scan has completed. In general, the peripheral operates on a single trigger source. However, in a group mode, two different trigger sources are used. Each group can contain one or more unique channels and may be scanned at different trigger intervals.

The majority of the driver serves to initialize the A/D peripheral and provide functions to read conversion results. Settings which are common to all channels such as conversion alignment or addition count are set in the Open() call. Specific channel enabling is done via a Control() command. Two Read() functions are provided- one which retrieves a single conversion value and another which retrieves all conversion registers whether the channel is enabled or not.

There are 13 to 21 channels available for conversion depending upen the MCU chosen. All MCUs include a Temperature and an Internal Reference Voltage sensor. Each has its own conversion register. In addition, in non-RX63x MCUs, a single channel can be designated as a "double trigger". This means the scan-complete flag/interrupt occurs on every other scan. On an odd scan, the conversion result is placed in its normal data register. On an even scan, the result is placed in a special register. Double trigger cannot be used with sensors or other channels (unless the other channels are in Group B), and a sensor can only be scanned by itself.

2. API Information

This Driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires your MCU support the following features:

- S12ADa or S12ADb peripheral

2.2 Hardware Resource Requirements

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

2.2.1 S12ADa/S12ADb

This driver makes use of all features available on these peripherals.

2.2.2 GPIO

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

2.3 Software Requirements

This driver is dependent upon the following packages:

- r_bsp

2.4 Limitations

No software limitations.

2.5 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v1.02

2.6 Header Files

Compile time configurable options are located in r_s12ad_rx\ref\r_s12ad_rx_config_reference.h. This file should be copied into the r_config subdirectory of the project and renamed to r_s12ad_rx_config.h. It is this renamed file that should be modified if needed and the original kept as a reference.

All API calls and their supporting interface definitions are located in r_s12ad_rx\r_s12ad_rx_if.h. Both this file and r_s12ad_rx_config.h should be included by the User's application.

2.7 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in *stdint.h*.

2.8 Configuration Overview

All configurable options that can be set at build time are located in the file “r_s12ad_rx_config.h”. A summary of these settings are provided in the following table:

Configuration options in *r_s12ad_rx_config.h*	
`#define ADC_CFG_PARAM_CHECKING_ENABLE 1`	If this equate is set to 1, parameter checking is included in the build. If the equate is set to 0, the parameter checking is omitted from the build. Setting this equate to BSP_CFG_PARAM_CHECKING_ENABLE utilizes the system default setting.

`// 1.8V <= AVcc0 <= 2.7V`
`#define ADC_CFG_PGA_GAIN 0`
 	
`// 2.7V <= AVcc0 <= 3.6V`
`//#define ADC_CFG_PGA_GAIN 1`
 	
`// 3.6V <= AVcc0 <= 5.5V`
`//#define ADC_CFG_PGA_GAIN 2`
 	
`// 4.5V <= AVcc0 <= 5.5V`
`//#define ADC_CFG_PGA_GAIN 3`
	This equate is for the Temperature Sensor gain amplifier on the RX210. The default is a value of 0 which is good for all target voltages. For best temperature resolution, the voltage range which most accurately reflects the AVcc0 should have its #define uncommented.

2.9 API Data Structures

The API data structures are located in the file “r_s12ad_rx_if.h” and discussed in Section 3.

2.10 Adding Driver to Your Project

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

1. Add the r_s12ad_rx and r_config folders to your project.
2. Add a project include path for the “r_s12ad_rx” directory.
3. Add a project include path for the “r_s12ad_rx\src” directory.
4. Add a project include path for the “r_config” directory.
5. Open "r_config\r_s12ad_rx_config.h" file and configure the driver for your project.
6. Add a #include for r_s12ad_rx_if.h to any source files that need to use the API functions.

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_ADC_Open()	Applies power to the A/D peripheral (and TEMPS peripheral on RX210 and RX63x), sets the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors. Optionally takes a callback function pointer for notifying the user at interrupt level whenever a scan has completed.
R_ADC_Control()	Provides commands for enabling channels and sensors, and for runtime operations. These include enabling/disabling trigger sources and interrupts, initiating a software trigger, and checking for scan completion.
R_ADC_Read()	Reads conversion results from a single channel, sensor, double trigger, or self-diagnosis register.
R_ADC_ReadAll()	Reads conversion results from all potential channel sources, enabled or not.
R_ADC_Close()	Ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.
R_ADC_GetVersion()	Returns at runtime the driver version number.

3.2 Return Values

These are the different error codes API functions can return. The enum is found in r_rs12ad_rx_if.h along with the API function declarations.

```
typedef enum e_adc_err     // ADC API error codes
{
    ADC_SUCCESS = 0,
    ADC_ERR_AD_LOCKED,     // Open() call is in progress elsewhere
    ADC_ERR_AD_NOT_CLOSED, // peripheral still running in another mode
    ADC_ERR_MISSING_PTR,   // missing required pointer argument
    ADC_ERR_INVALID_ARG,   // argument is not valid for parameter
    ADC_ERR_ILLEGAL_ARG,   // argument is illegal for mode
    ADC_ERR_SCAN_NOT_DONE  // default, Group A, or Group B scan not done
} adc_err_t;
```

3.3 R_ADC_Open()

This function applies power to the A/D peripheral, sets the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors. If interrupt priority is non-zero, the function takes a callback function pointer for notifying the user at interrupt level whenever a scan has completed.

Format

```
adc_err_t  R_ADC_Open(adc_mode_t const    mode,
                      adc_cfg_t * const   p_cfg,
                      void                (* const  p_callback)(void *p_args));
```

Parameters

mode

Operational mode (see enumeration below)

p_cfg

Pointer to configuration structure (see below)

p_callback

Optional pointer to function called from interrupt when a scan completes

The mode indicates whether channels or a sensor is to be scanned, and the type of scan to be performed. For ADC_MODE_SS_MULTI_CH_GROUPED_DBLTRIG_A, only one channel can be in Group A. The following enum lists possible operational modes (MCU dependent):

```
typedef enum e_adc_mode
{
    ADC_MODE_SS_TEMPERATURE,        // single scan temperature sensor
    ADC_MODE_SS_INT_REF_VOLT,       // single scan internal ref voltage sensor
    ADC_MODE_SS_ONE_CH,             // single scan one channel
    ADC_MODE_SS_MULTI_CH,           // 1 trigger source, scan multiple channels
    ADC_MODE_CONT_ONE_CH,           // continuous scan one channel
    ADC_MODE_CONT_MULTI_CH,         // continuous scan multiple channels

    ADC_MODE_SS_ONE_CH_DBLTRIG,     // on even trigs save to ADDBLDR
    ADC_MODE_SS_MULTI_CH_GROUPED,   // 2 trigger sources, scan multiple channels
    ADC_MODE_SS_MULTI_CH_GROUPED_DBLTRIG_A,

    ADC_MODE_MAX
} adc_mode_t;
```

A sample structure for the p_cfg parameter is as follows:

```
typedef struct st_adc_cfg
{
    adc_add_t       add_cnt;
    adc_align_t     alignment;       // ignored if addition used
    adc_clear_t     clearing;
    adc_speed_t     conv_speed;
    adc_trig_t      trigger;         // default and Group A trigger source
    adc_trig_t      trigger_groupb;  // valid only for group modes
    uint8_t         priority;        // for S12ADIO int; 1=lo 15=hi 0=off/polled
    uint8_t         priority_groupb; // GBADI interrupt priority; 0-15
} adc_cfg_t;
```

The .trigger and .trigger_groupb fields use an enum for trigger sources (contents MCU dependent):

```
typedef enum e_adc_trig         // trigger sources
{
```

```
    ADC_TRIG_NONE_GROUPB = 0,
    ADC_TRIG_ASYNC_ADTRG0 = 0,  // ext asynchronous trigger; not for Group modes
                                //  nor double trigger modes
    ADC_TRIG_SYNC_TRG0AN = 1,
    ADC_TRIG_SYNC_TRG0BN = 2,
    ADC_TRIG_SYNC_TRGAN = 3,
    ADC_TRIG_SYNC_TRG0EN = 4,
    ADC_TRIG_SYNC_TRG0FN = 5,
    ADC_TRIG_SYNC_TRG4AN = 6,
    ADC_TRIG_SYNC_TRG4BN = 7,
    ADC_TRIG_SYNC_TRG4ABN = 8,
    ADC_TRIG_SYNC_ELC = 9,
    ADC_TRIG_PLACEHOLDER = 10,  // for RX210 temp sensor, use ADC_TRIG_SOFTWARE
    ADC_TRIG_SYNC_TRGAN1 = 11,
    ADC_TRIG_SYNC_TRG4ABN1 = 12,
    ADC_TRIG_HW_MAX,
    ADC_TRIG_SOFTWARE = 16      // software trigger; not for Group modes
                                //  nor double trigger modes
} adc_trig_t;
```

The .priority and .groupb_priority fields use interrupt priority levels ranging from 1 (lowest) to 15 (highest). If the priority is set to 0, the corresponding interrupt is disabled and the application must poll for the scan to complete (use Control() function).

The .add_cnt field uses the enum which follows. The number of samples added applies to all channels and sensors (except for Temperature sensor on RX210). An individual channel may opt out of any addition using the Control() function.

```
typedef enum e_adc_add
{
    ADC_ADD_OFF=0,                  // addition functionality is disabled
    ADC_ADD_TWO_SAMPLES=1,
    ADC_ADD_THREE_SAMPLES=2,
    ADC_ADD_FOUR_SAMPLES=3,
    ADC_ADD_MAX
} adc_add_t;
```

The .alignment, .clearing, and .conversion fields use the enumerations specified below. If addition is enabled, the .alignment field is ignored.

```
typedef enum e_adc_align
{
    ADC_ALIGN_RIGHT = 0x0000,
    ADC_ALIGN_LEFT  = 0x8000
} adc_align_t;

typedef enum e_adc_clear
{
    ADC_NO_CLEAR_AFTER_READ = 0x0000,
    ADC_CLEAR_AFTER_READ    = 0x0020
} adc_clear_t;

// For RX63x
typedef enum e_adc_speed
{
    ADC_CONVERT_SPEED_PCLK_DIV8 = 0x00,
    ADC_CONVERT_SPEED_PCLK_DIV4 = 0x01,
    ADC_CONVERT_SPEED_PCLK_DIV2 = 0x02,
    ADC_CONVERT_SPEED_PCLK = 0x03,
    ADC_CONVERT_SPEED_MAX
```

```
} adc_speed_t;

// For RX11x
typedef enum e_adc_speed
{
    ADC_CONVERT_SPEED_NORM = 0x0000,
    ADC_CONVERT_SPEED_HI   = 0x0400
} adc_speed_t;
```

Return Values

ADC_SUCCESS:	*Successful*
ADC_ERR_AD_LOCKED:	*Open() call is in progress elsewhere*
ADC_ERR_AD_NOT_CLOSED:	*Peripheral is still running in another mode; Perform R_ADC_Close() first*
ADC_ERR_INVALID_ARG:	*mode or element of p_cfg structure has invalid value*
ADC_ERR_ILLEGAL_ARG:	*an argument is illegal based upon mode*
ADC_ERR_MISSING_PTR:	*p_cfg pointer is FIT_NO_PTR/NULL*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Applies power to the A/D peripheral, sets the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors. Setting the interrupt priority to 0 indicates that the application will poll until scanning completes (see Control()). With a non-zero interrupt priority (interrupt usage), a callback function is required. This function is called by the interrupts whenever a scan has completed with an argument indicating which scan completed.

Reentrant

No. There is only one S12AD peripheral and therefore this function should only be called once before a call to Close().

Example

```
adc_cfg_t   config;

    /* INITIALIZE FOR SINGLE SCAN OF TEMPERATURE SENSOR
     *  - use software trigger to start scan; poll for completion
     *  - don't do any summing of conversion values
     *  - keep the data registers aligned right, and clear after reading
     *  - use normal speed conversion
     */
    config.trigger = ADC_TRIG_SOFTWARE;
    config.priority = 0;                // denotes polling!
    config.add_cnt = ADC_ADD_OFF;
    config.alignment = ADC_ALIGN_RIGHT;
    config.clearing = ADC_CLEAR_AFTER_READ;
    config.conv_speed = ADC_CONVERT_NORM_SPEED;

    R_ADC_Open(ADC_MODE_SS_TEMPERATURE, &config, FIT_NO_FUNC);
```

Special Notes:

The application must complete MPC and PORT initialization prior to calling Open(). Note that the application cannot use Port 4 for output if an analog channel is used for input on Port 4 or Port E. The following is a sample initialization for an RSKRX111 Rev 1 board:

```
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);
```

```
    MPC.P40PFS.BYTE = 0x80;     // Pin Func Select P40 AN000
    PORT4.PDR.BIT.B0 = 0;       // set AN pin direction to input (dflt)
    PORT4.PMR.BIT.B0 = 1;       // set AN pin mode to peripheral

    MPC.PB0PFS.BIT.PSEL = 0x09; // Pin Func Select PB0 ADTRIG0; SW3 on RSKRX111
    PORTB.PDR.BIT.B0 = 0;       // set ADTRIG0 pin direction to input (dflt)
    PORTB.PMR.BIT.B0 = 1;       // set ADTRIG0 pin mode to peripheral

    R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);
```

The application must set the A/D conversion clock (PCKD) prior to calling Open().

If interrupts are in use, a callback function is required which takes a single argument. This is a pointer to a structure which is cast to a void pointer (provides consistency with other FIT module callback functions). The structure is as follows:

```
typedef struct st_adc_cb_args      // callback arguments
{
    adc_cb_evt_t   event;
} adc_cb_args_t;

typedef enum e_adc_cb_evt          // callback function events
{
    ADC_EVT_SCAN_COMPLETE,         // normal/Group A scan complete
    ADC_EVT_GROUPB_SCAN_COMPLETE   // Group B scan complete
} adc_cb_evt_t;
```

An example template for a callback function is provided here:

```
void MyCallback(void *p_args)
{
adc_cb_args_t    *args;


    args = (adc_cb_args_t *)p_args;

    if (args->event == ADC_EVT_SCAN_COMPLETE)
    {
        // Read results here
        nop();
    }
    else if (args->event == ADC_EVT_GROUPB_SCAN_COMPLETE)
    {
        // Read Group B results here
        nop();
    }
}
```

3.4 R_ADC_Control()

This function provides commands for enabling channels and sensors and for runtime operations. These include enabling/disabling trigger sources and interrupts, initiating a software trigger, and checking for scan completion.

Format

```
adc_err_t   R_ADC_Control(adc_cmd_t const    cmd,
                          void * const       p_cfg);
```

Parameters

cmd

Command to run (see enumeration below)

p_cfg

Pointer to optional configuration structure (see below). FIT_NO_PTR/NULL for most commands.

Example *cmd* values are as follows:

```
typedef enum e_adc_cmd
{
    // Commands for special hardware configurations
    ADC_CMD_SET_DDA_STATE_CNT,     // RX210 only; cannot use with sensors
    ADC_CMD_SET_SAMPLE_STATE_CNT,

    // Commands to enable channels or sensors
    ADC_CMD_ENABLE_CHANS,          // enables chans and A&B INT if priority != 0
    ADC_CMD_ENABLE_TEMP_SENSOR,    // enables sensor and INT if priority != 0
    ADC_CMD_ENABLE_VOLT_SENSOR,    // enables sensor and INT if priority != 0

    // Commands to enable hardware triggers or cause software trigger
    ADC_CMD_ENABLE_TRIG,           // ADCSR.TRGE=1 for sync/async trigs
    ADC_CMD_SCAN_NOW,              // software trigger start scan

    // Commands to poll for scan completion
    ADC_CMD_CHECK_SCAN_DONE,        // for Normal scan
    ADC_CMD_CHECK_SCAN_DONE_GROUPA, // non-63x
    ADC_CMD_CHECK_SCAN_DONE_GROUPB, // non-63x

    // Advanced control commands
    ADC_CMD_DISABLE_TRIG,          // ADCSR.TRGE=0 for sync/async trigs
    ADC_CMD_DISABLE_INT,           // interrupt disable; ADCSR.ADIE=0
    ADC_CMD_ENABLE_INT,            // interrupt enable;  ADCSR.ADIE=1
    ADC_CMD_DISABLE_INT_GROUPB,    // interrupt disable; ADCSR.GBADIE=0
    ADC_CMD_ENABLE_INT_GROUPB,     // interrupt enable;  ADCSR.GBADIE=1
    ADC_CMD_MAX
} adc_cmd_t;
```

The commands ADC_CMD_SET_DDA_STATE_CNT, ADC_CMD_SET_SAMPLE_STATE_CNT and ADC_CMD_ENABLE_CHANS require arguments. The argument structures for these commands are shown below. All other commands should use FIT_NO_PTR/NULL for the *p_cfg* argument.

For ADC_CMD_SET_DDA_STATE_CNT:

```
typedef enum e_adc_charge
{
    ADC_DDA_DISCHARGE = 0x00,
    ADC_DDA_PRECHARGE = 0x01
```

```
} adc_charge_t;

typedef struct st_adc_dda           // Disconnection Detection Assist
{
    adc_charge_t    method;
    uint8_t         num_states;     // 1-255
} adc_dda_t;
```

Example structure for non-63x ADC_CMD_SET_SAMPLE_STATE_CNT:

```
typedef enum e_adc_sst_reg          // sample state registers
{
    ADC_SST_CH0 = 0,
    ADC_SST_CH1,
    ADC_SST_CH2,
    ADC_SST_CH3,
    ADC_SST_CH4,
    ADC_SST_CH5,                    // not on RX110/111
    ADC_SST_CH6,
    ADC_SST_CH7,                    // not on RX110/111
    ADC_SST_CH8_TO_15,
    ADC_SST_TEMPERATURE,
    ADC_SST_VOLTAGE,
    ADC_SST_REG_MAX
} adc_sst_reg_t;

typedef struct st_adc_time
{
    adc_sst_reg_t   reg_id;
    uint8_t         num_states;     // default=20; ch8-15 use the same value
} adc_time_t;
```

Example structure for ADC_CMD_ENABLE_CHANS:

```
typedef enum e_adc_diag             // Self-Diagnosis Channel
{
    ADC_DIAG_OFF = 0x00,
    ADC_DIAG_0_VOLT = 0x01,
    ADC_DIAG_HALF_VREFH0 = 0x2,
    ADC_DIAG_VREFH0 = 0x3,
    ADC_DIAG_ROTATE_VOLTS = 0x4,
    ADC_DIAG_MAX
} adc_diag_t;

typedef struct st_adc_ch_cfg          // bit 0 is ch0; bit 15 is ch15
{
    uint32_t      chan_mask;          // channels/bits 0-15
    uint32_t      chan_mask_groupb;   // valid for group modes
    uint32_t      add_mask;           // valid if add enabled in Open()
    adc_diag_t    diag_method;        // self-diagnosis virtual channel
    uint8_t       sample_hold_mask;   // channels/bits 0-2
    uint8_t       sample_hold_states; // 4-255; min .4us
} adc_ch_cfg_t;
```

Return Values

ADC_SUCCESS:	*Successful*
ADC_ERR_MISSING_PTR:	*p_cfg pointer is FIT_NO_PTR/NULL when required as an argument*

ADC_ERR_INVALID_ARG: *cmd or element of p_cfg structure has invalid value.*
ADC_ERR_ILLEGAL_ARG: *cmd is illegal based upon mode*
ADC_ERR_SCAN_NOT_DONE: *The requested scan has not completed*

Properties
Prototyped in file "r_s12ad_rx_if.h"

Description
Provides commands for enabling channels and sensors and for runtime operations. These include enabling/disabling trigger sources and interrupts, initiating a software trigger, and checking for scan completion.

After an Open() call, the following sequence of Control() commands should be issued:

1) Optionally issue commands for special hardware configuration (ADC_CMD_SET_DDA_STATE_CNT or ADC_CMD_SET_SAMPLE_STATE_CNT).
2) Issue command to enable channels or sensors (ADC_CMD_ENABLE_CHANS, ADC_CMD_ENABLE_TEMP_SENSOR, or ADC_CMD_ENABLE_VOLT_SENSOR).
3) Issue command to enable hardware triggers or cause a software trigger (ADC_CMD_TRIG_ENABLE or ADC_CMD_SOFTWARE_TRIG_START_SCAN).
4) If polling instead of using interrupts, check for completion of scan (ADC_CMD_CHECK_SCAN_DONE, ADC_CMD_CHECK_SCAN_DONE_GROUPA or ADC_CMD_CHECK_SCAN_DONE_GROUPB).
5) When the scan completes, the converted data is accessed using the Read() or ReadAll() functions at the application or interrupt level.

Reentrant
Yes, but in general should not be used as such. A valid case for re-entrancy would be in group mode with one task performing a ADC_CMD_CHECK_SCAN_DONE_GROUPA and another performing a ADC_CMD_CHECK_SCAN_DONE_GROUPB. Other commands such as enabling/disabling triggers or interrupts would affect operations being performed by another task and re-entrancy of this kind should be avoided.

Example 1: Single Channel Polling

```
uint16_t       data;
adc_cfg_t      config;
adc_ch_cfg_t   ch_cfg;

/* OPEN ADC */

// Open ADC for software trigger, single scan of one channel, and polling
config.trigger = ADC_TRIG_SOFTWARE;
config.priority = 0;                              // denotes polling
config.add_cnt = ADC_ADD_OFF;
config.alignment = ADC_ALIGN_RIGHT;
config.clearing = ADC_CLEAR_AFTER_READ_ON;
config.conv_speed = ADC_CONVERT_SPEED_NORM;     // RX11x
err = R_ADC_Open(ADC_MODE_SS_ONE_CH, &config, NULL);

/* ENABLE CHANNELS */

// Specify and enable potentiometer channel
// Channel 4 on RSKRX111 Rev 0
// Channel 0 on RSKRX111 Rev 1
// Channel 0 on RSKRX210
```

```
// Channel 2 on RDKRX63N
ch_cfg.chan_mask = ADC_MASK_CH0;
err = R_ADC_Control(ADC_CMD_ENABLE_CHANS, &ch_cfg);

// Repeatedly trigger, poll for completion, and read result
while(1)
{
    /* CAUSE SOFTWARE TRIGGER */
    err = R_ADC_Control(ADC_CMD_SCAN_NOW, NULL);

    /* WAIT FOR SCAN TO COMPLETE */
    while (R_ADC_Control(ADC_CMD_CHECK_SCAN_DONE,NULL) == ADC_ERR_SCAN_NOT_DONE)
        nop();

    /* READ RESULT */
    err = R_ADC_Read(ADC_REG_CH0, &data);
}
```

Example 2: Temperature Sensor Polling and Set Sample State Count

```
uint16_t       data;
adc_cfg_t      config;
adc_time_t     sst;           // sample state

/* OPEN ADC */

// Open ADC for software trigger, single scan of temperature sensor, and polling
config.trigger = ADC_TRIG_SOFTWARE;
config.priority = 0;                          // denotes polling
config.add_cnt = ADC_ADD_OFF;
config.alignment = ADC_ALIGN_RIGHT;
config.clearing = ADC_CLEAR_AFTER_READ_ON;
config.conv_speed = ADC_CONVERT_SPEED_NORM;    // RX11x
R_ADC_Open(ADC_MODE_SS_TEMPERATURE, &config, NULL);


/* DO SPECIAL HARDWARE CONFIGURATION */

// Set number of sampling states
// Minimum sampling time: RX11x 5us
// For PCLKD=50Mhz, 1 state = 1/50MHz = 20ns. 5us/20ns = 250 states
sst.reg_id = ADC_SST_TEMPERATURE;
sst.num_states = 250;
adc_err = R_ADC_Control(ADC_CMD_SET_SAMPLE_STATE_CNT, &sst);

/* ENABLE TEMPERATURE SENSOR */
R_ADC_Control(ADC_CMD_ENABLE_TEMP_SENSOR, NULL);
// insert 80us stabilization delay here on RX210 for first conversion

/* CAUSE SOFTWARE TRIGGER */
R_ADC_Control(ADC_CMD_SCAN_NOW, NULL);

/* WAIT FOR SCAN TO COMPLETE */
while (R_ADC_Control(ADC_CMD_CHECK_SCAN_DONE, NULL) == ADC_ERR_SCAN_NOT_DONE)
    nop();

/* READ RESULT */
R_ADC_Read(ADC_REG_TEMP, &data);
// NOTE: Do not use first result on RX11x
```

Example 3: Grouped Channels (unavailable RX63N) with Interrupt Triggers, Double Trigger on Group A, and Addition

```
adc_cfg_t      config;
adc_ch_cfg_t   ch_cfg;

/* INITIALIZE MTU HERE (USED FOR TRIGGER SOURCES) */

/* OPEN ADC */

/* INITIALIZE ADC FOR GROUP SCANNING WITH DOUBLE TRIGGER
 *  - use synchronous trigger TRGAN to start Group A scan; int priority 4
 *  - use synchronous trigger TRG0EN to start Group B scan; int priority 5
 *  - allow each channel to be scanned four times and added before continuing
 *     can shift result right 4x to get average
 *     2x because LSB is bit 2; 2x more to divide by 4
 *  - clear regs after reading and use normal speed conversion
 */
config.trigger = ADC_TRIG_SYNC_TRGAN;
config.priority = 4;
config.trigger_groupb = ADC_TRIG_SYNC_TRG0EN;
config.priority_groupb = 5;
config.add_cnt = ADC_ADD_FOUR_SAMPLES;
// .alignment field is ignored due to addition
config.clearing = ADC_CLEAR_AFTER_READ_ON;
config.conv_speed = ADC_CONVERT_SPEED_NORM;     // RX11x
R_ADC_Open(ADC_MODE_SS_MULTI_CH_GROUPED_DBLTRIG_A, &config, MyCallback);

/* ENABLE CHANNELS */

// Can only have one chan for double triggering, and is only channel in Group A
// Have channel 8 as Group A, have 2, 3, and 9 as Group B
// Perform addition on all channels except 9
ch_cfg.chan_mask = ADC_MASK_CH8;
ch_cfg.chan_mask_groupb = ADC_MASK_CH2 | ADC_MASK_CH3 | ADC_MASK_CH9;
ch_cfg.add_mask = ADC_MASK_CH8 | ADC_MASK_CH2 | ADC_MASK_CH3;
R_ADC_Control(ADC_CMD_ENABLE_CHANS, &ch_cfg);

/* ENABLE TRIGGERS */
R_ADC_Control(ADC_CMD_ENABLE_TRIG, NULL);

/* SCAN COMPLETE WHEN INTERRUPT OCCURS */



/* The callback is called twice from interrupt level- once after each
 *  group scan completes. The order depends upon the trigger order.
 */
void MyCallback(void *p_args)
{
adc_cb_args_t  *args;
uint16_t        dbltrg,data2,data3,data8,data9;


    args = (adc_cb_args_t *)p_args;

    /* READ RESULTS */
```

```
    if (args->event == ADC_EVT_SCAN_COMPLETE)
    {
        // From S12ADIO interrupt, Group A scan complete, read registers
        R_ADC_Read(ADC_REG_CH8, &data8);
        R_ADC_Read(ADC_REG_DBLTRIG, &dbltrig);
    }
    else if (args->event == ADC_EVT_SCAN_COMPLETE_GROUPB)
    {
        // From GBADI interrupt, Group B scan complete, read registers
        R_ADC_Read(ADC_REG_CH2, &data2);
        R_ADC_Read(ADC_REG_CH3, &data3);
        R_ADC_Read(ADC_REG_CH9, &data9);
    }

    // process data, or set flag for application level to do so
}
```

Special Notes:

The application shall ensure that 1us has elapsed after Open() has completed (or more specifically, after the peripheral is brought out of module stop state within Open()) before beginning A/D conversions.

The application should ignore the first temperature or internal reference voltage sensor conversion made, and ensure 5 ms stabilization time has passed before starting the second conversion.

Channel masks are checked against the pin package specified by the equate BSP_CFG_MCU_PART_PACKAGE in r_bsp_config.h for available A/D channels.

3.5 R_ADC_Read()

This function reads conversion results from a single channel, sensor, double trigger, or self-diagnosis register.

Format

```
adc_err_t  R_ADC_Read(adc_reg_t const       reg_id,
                      uint16_t * const      p_data);
```

Parameters

reg_id
Id for the register to read (see enum below)
p_data
Pointer to variable to load value into.

The following enum should be used for the reg_id parameter (contents MCU dependent):

```
typedef enum e_adc_reg
{
    ADC_REG_CH0 = 0,
    ADC_REG_CH1 = 1,
    ADC_REG_CH2 = 2,
    ADC_REG_CH3 = 3,
    ADC_REG_CH4 = 4,
    ADC_REG_CH5 = 5,       // Not on RX110/111
    ADC_REG_CH6 = 6,
    ADC_REG_CH7 = 7,       // Not on RX110/111
    ADC_REG_CH8 = 8,
    ADC_REG_CH9 = 9,
    ADC_REG_CH10 = 10,
    ADC_REG_CH11 = 11,
    ADC_REG_CH12 = 12,
    ADC_REG_CH13 = 13,
    ADC_REG_CH14 = 14,
    ADC_REG_CH15 = 15,
    ADC_REG_CH16 = 16,     // RX63x only
    ADC_REG_CH17 = 17,     // RX63x only
    ADC_REG_CH18 = 18,     // RX63x only
    ADC_REG_CH19 = 19,     // RX63x only
    ADC_REG_CH20 = 20,     // RX63x only
    ADC_REG_TEMP,
    ADC_REG_VOLT,
    ADC_REG_DBLTRIG,       // Not on RX63x
    ADC_REG_SELF_DIAG,     // RX210 only
    ADC_REG_MAX
} adc_reg_t;
```

Return Values

ADC_SUCCESS:	*Success*
ADC_ERR_INVALID_ARG:	*reg_id contains an invalid value.*
ADC_ERR_MISSING _PTR:	*p_data is FIT_NO_PTR/NULL*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Reads conversion results from a single channel, sensor, double trigger, or self-diagnosis register.

Reentrant
Yes.

Example
```
uint16_t data;
     :
    /* Read channel 0 */
    R_ADC_Read(ADC_CH0_REG, &data);       // conversion value placed in “data”
```

Special Notes:
None.

3.6 R_ADC_ReadAll()

This function reads conversion results from all potential sources, enabled or not.

Format

adc_err_t R_ADC_ReadAll(adc_data_t * const p_data);

Parameters

p_data

Pointer to structure to load register values into.

An example structure for the register values is as follows:

```
typedef struct st_adc_data
{
    uint16_t    chan[ADC_REG_ARRAY_MAX];  // chans/indexes used depends upon MCU
    uint16_t    dbltrig;                  // Not on RX63x
    uint16_t    self_diag;                // RX210 only
} adc_data_t;
```

Return Values

ADC_SUCCESS:	*Success*
ADC_ERR_MISSING_PTR:	*p_data is FIT_NO_PTR/NULL*

Properties

Prototyped in file “r_s12ad_rx_if.h”

Description

Reads conversion results from all potential sources, enabled or not. This function is convenient for getting a snapshot of all values at any point in time.

Reentrant

Yes.

Example

```
adc_data_t data;
     :
    /* Read all channel registers available on hardware */
    R_ADC_ReadAll(&data);     // “data” loaded with all conversion reg values
```

Special Notes:

None.

3.7 R_ADC_Close()

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

Format
void R_ADC_Close(void);

Parameters
none

Return Values
None

Properties
Prototyped in file “r_s12ad_rx_if.h”

Description
Ends any scan in progress, disables interrupts, and removes power to the A/D peripheral. Allows call to Open() to be performed again. This is necessary when changing configurations, such as switching between sensor scanning and channel scanning.

Reentrant
No. There is only one S12AD peripheral and therefore this function should only be called once after a call to Open().

Example
```
 :
err = R_ADC_Open(ADC_MODE_SS_MULTI_CH_GROUPED, &config, MyCallback);
 :
R_ADC_Close();
```

Special Notes:
This function will abort any scan that may be in progress.

3.8 R_ADC_GetVersion()

This function returns the driver version number at runtime.

Format
uint32_t R_ADC_GetVersion(void)

Parameters
None

Return Values
Version number.

Properties
Prototyped in file “r_s12ad_rx_if.h”

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

Reentrant
Yes

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

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

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Nov.15.2013	—	First edition issued
1.20	Apr.21.2014	1,3	Added mention of support for RX110/63x.
		11,12	Added interface for RX210 Sample&Hold, Self-Diagnosis, and Disconnect Detection Assist (DDA)
1.30	Jun.05.2014	—	Fixed bug in code that eliminated channels 8-15.

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 product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; 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 (artificial life 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 malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or 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 Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. 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 this 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.

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

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

RENESAS

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

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

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

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

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

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

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

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

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

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

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

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

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

Colophon 4.0