SMART ARM-based Microcontrollers

# AT12199: SAM C21 Sigma-Delta Analog-to-Digital Converter (SDADC) Driver

APPLICATION NOTE

## Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the device's SDADC functionality.

The following peripheral is used by this module:

- SDADC (Sigma-Delta Analog-to-Digital Converter)

The following devices can use this module:

- Atmel | SMART SAM C21

The outline of this documentation is as follows:

# Table of Contents

# 1. Software License

## 2. Prerequisites

There are no prerequisites for this module.

# 3. Module Overview

The Sigma-Delta Analog-to-Digital Converter (SDADC) converts analog signals to digital values. The sigma-delta architecture of the SDADC implies a filtering and a decimation of the bitstream at the output of the SDADC. The input selection is up to three input analog channels.

The SDADC provides up to 16-bit resolution at about 1000 samples per second (1KSPS) and sized 24 bits signed result to handle filtering and gain correction without overflow. The SDADC measurements can be started by either application software or an incoming event from another peripheral in the device.

The conversion is performed on a full range between 0V and the reference voltage. Both internal and external reference voltages can be selected. The reference range must be set to match the voltage of the reference used. Analog inputs between these voltages convert to values based on a linear conversion.

## 3.1. Sample Clock

A generic clock (GCLK_SDADC) is required to generate the CLK_SDADC to the SDADC module. The SDADC features a prescaler, which enables conversion at lower clock rates than the input Generic Clock to the SDADC module.

The SDADC data sampling frequency (CLK_SDADC_FS) in the SDADC module is the CLK_SDADC/4, the reduction comes from the phase generator between the prescaler and the SDADC.

OSR is the Over Sampling Ratio, which can be modified to change the output data rate. The conversion time depends on the selected OSR and the sampling frequency of the SDADC. The conversion time can be described with:

$$t_{SAMPLE} = \frac{22 + 3 \times OSR}{CLK\_SDADC\_FS}$$

1. Initialization of the SDADC (22 sigma-delta samples).
2. Filling of the decimation filter (3*OSR sigma-delta samples).

## 3.2. Gain and Offset Correction

A specific offset, gain, and shift can be applied to each source of the SDADC by performing the following operation:

$$Data = \left(Data_0 + OFFSET\right) \times \frac{GAIN}{2^{SHIFT}}$$

## 3.3. Window Monitor

The SDADC module window monitor function can be used to automatically compare the conversion result against a predefined pair of upper and lower threshold values.

## 3.4. Events

Event generation and event actions are configurable in the SDADC.

The SDADC has two actions that can be triggered upon event reception:

- Start conversion

- Conversion flush

The SDADC can generate two kinds of events:

- Window monitor
- Result ready

If the event actions are enabled in the configuration, any incoming event will trigger the action.

If the window monitor event is enabled, an event will be generated when the configured window condition is detected.

If the result ready event is enabled, an event will be generated when a conversion is completed.

# 4. Special Considerations

There are no special considerations for this module.

# 5. Extra Information

For extra information see Extra Information for SDADC Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

# 6. Examples

For a list of examples related to this driver, see Examples for SDADC Driver.

# 7. API Overview

## 7.1. Variable and Type Definitions

### 7.1.1. Type sdadc_callback_t

```
typedef void(* sdadc_callback_t )(const struct sdadc_module *const module)
```

Type of the callback functions.

## 7.2. Structure Definitions

### 7.2.1. Struct sdadc_config

Configuration structure for an SDADC instance. This structure should be initialized by the sdadc_get_config_defaults() function before being modified by the user application.

**Table 7-1 Members**

| Type | Name | Description |
|---|---|---|
| uint8_t | clock_prescaler | Clock prescaler |
| enum gclk_generator | clock_source | GCLK generator used to clock the peripheral |
| struct sdadc_correction_config | correction | Gain and offset correction configuration structure |
| enum sdadc_event_action | event_action | Event action to take on incoming event |
| bool | freerunning | Enables free running mode if true |
| enum sdadc_mux_input | mux_input | MUX input |
| bool | on_command | Enables SDADC depend on other peripheral if true |
| enum sdadc_over_sampling_ratio | osr | Over sampling ratio |
| struct sdadc_reference | reference | Voltage reference |
| bool | run_in_standby | Enables SDADC in standby sleep mode if true |
| bool | seq_enable[] | Enables positive input in the sequence if true |
| uint8_t | skip_count | Skip Count |
| struct sdadc_window_config | window | Window monitor configuration structure |

### 7.2.2. Struct sdadc_correction_config

Offset, gain, and shift correction configuration structure. Part of the sdadc_config struct will be initialized by sdadc_get_config_defaults.

**Table 7-2 Members**

| Type | Name | Description |
|---|---|---|
| uint16_t | gain_correction | Gain correction |
| int32_t | offset_correction | Offset correction |
| uint8_t | shift_correction | Shift correction |

### 7.2.3. Struct sdadc_events

Event flags for the SDADC module. This is used to enable and disable events via sdadc_enable_events() and sdadc_disable_events().

**Table 7-3 Members**

| Type | Name | Description |
|---|---|---|
| bool | generate_event_on_conversion_done | Enable event generation on conversion done |
| bool | generate_event_on_window_monitor | Enable event generation on window monitor |

### 7.2.4. Struct sdadc_module

SDADC software instance structure, used to retain software state information of an associated hardware module instance.

**Note:** The fields of this structure should not be altered by the user application; they are reserved for module-internal use only.

### 7.2.5. Struct sdadc_reference

Reference configuration structure.

**Table 7-4 Members**

| Type | Name | Description |
|---|---|---|
| bool | on_ref_buffer | Reference buffer turning switch |
| enum sdadc_reference_select | ref_range | Reference voltage range |
| enum sdadc_reference_select | ref_sel | Reference voltage selection |

### 7.2.6. Struct sdadc_window_config

Window monitor configuration structure.

**Table 7-5 Members**

| Type | Name | Description |
|---|---|---|
| int32_t | window_lower_value | Lower window value |
| enum sdadc_window_mode | window_mode | Selected window mode |
| int32_t | window_upper_value | Upper window value |

## 7.3. Macro Definitions

### 7.3.1. Module Status Flags

SDADC status flags, returned by sdadc_get_status() and cleared by sdadc_clear_status().

#### 7.3.1.1. Macro SDADC_STATUS_RESULT_READY

```
#define SDADC_STATUS_RESULT_READY
```

SDADC result ready.

#### 7.3.1.2. Macro SDADC_STATUS_OVERRUN

```
#define SDADC_STATUS_OVERRUN
```

SDADC result overwritten before read.

#### 7.3.1.3. Macro SDADC_STATUS_WINDOW

```
#define SDADC_STATUS_WINDOW
```

Window monitor match.

## 7.4. Function Definitions

### 7.4.1. Driver Initialization and Configuration

#### 7.4.1.1. Function sdadc_init()

Initializes the SDADC.

```
enum status_code sdadc_init(
        struct sdadc_module *const module_inst,
        Sdadc * hw,
        struct sdadc_config * config)
```

Initializes the SDADC device struct and the hardware module based on the given configuration struct values.

**Table 7-6 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[out]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | hw | Pointer to the SDADC module instance |
| **[in]** | config | Pointer to the configuration struct |

**Returns**
Status of the initialization procedure.

**Table 7-7 Return Values**

| Return value | Description |
|---|---|
| STATUS_OK | The initialization was successful |
| STATUS_ERR_INVALID_ARG | Invalid argument(s) were provided |
| STATUS_BUSY | The module is busy with a reset operation |
| STATUS_ERR_DENIED | The module is enabled |

#### 7.4.1.2. Function sdadc_get_config_defaults()

Initializes an SDADC configuration structure to defaults.

```
void sdadc_get_config_defaults(
        struct sdadc_config *const config)
```

Initializes a given SDADC configuration struct to a set of known default values. This function should be called on any new instance of the configuration struct before being modified by the user application.

The default configuration is as follows:

- GCLK generator 0 (GCLK main) clock source
- Positive reference 1
- Div 2 clock prescaler
- Over Sampling Ratio is 64
- Skip 0 samples
- MUX input on SDADC AIN1
- All events (input and generation) disabled
- Free running disabled
- Run in standby disabled
- On command disabled
- Disable all positive input in sequence
- Window monitor disabled
- No gain/offset/shift correction

**Table 7-8 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[out]** | config | Pointer to configuration struct to initialize to default values |

### 7.4.2. Status Management

#### 7.4.2.1. Function sdadc_get_status()

Retrieves the current module status.

```
uint32_t sdadc_get_status(
        struct sdadc_module *const module_inst)
```

Retrieves the status of the module, giving overall state information.

**Table 7-9 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |

**Returns**
Bitmask of `SDADC_STATUS_*` flags.

**Table 7-10 Return Values**

| Return value | Description |
|---|---|
| SDADC_STATUS_RESULT_READY | SDADC result is ready to be read |
| SDADC_STATUS_WINDOW | SDADC has detected a value inside the set window range |
| SDADC_STATUS_OVERRUN | SDADC result has overrun |

#### 7.4.2.2. Function sdadc_clear_status()

Clears a module status flag.

```
void sdadc_clear_status(
        struct sdadc_module *const module_inst,
        const uint32_t status_flags)
```

Clears the given status flag of the module.

**Table 7-11 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | status_flags | Bitmask of `SDADC_STATUS_*` flags to clear |

#### 7.4.2.3. Function sdadc_get_sequence_status()

Get a module sequence flag.

```
bool sdadc_get_sequence_status(
        struct sdadc_module *const module_inst,
        uint8_t * seq_state)
```

Get the given status flag of the module.

**Table 7-12 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[out]** | seq_state | Identifies the last conversion done in the sequence |

**Returns**
Status of the SDADC sequence conversion.

**Table 7-13 Return Values**

| Return value | Description |
|---|---|
| true | When the sequence start |
| false | When the last conversion in a sequence is done |

### 7.4.3. Enable, Disable, and Reset SDADC Module, Start Conversion and Read Result

#### 7.4.3.1. Function sdadc_is_syncing()

Determines if the hardware module(s) are currently synchronizing to the bus.

```
bool sdadc_is_syncing(
        struct sdadc_module *const module_inst)
```

Checks to see if the underlying hardware peripheral module(s) are currently synchronizing across multiple clock domains to the hardware bus. This function can be used to delay further operations on a module until such time that it is ready, to prevent blocking delays for synchronization in the user application.

**Table 7-14 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |

**Returns**
Synchronization status of the underlying hardware module(s).

**Table 7-15 Return Values**

| Return value | Description |
|---|---|
| true | If the module synchronization is ongoing |
| false | If the module has completed synchronization |

#### 7.4.3.2. Function sdadc_enable()

Enables the SDADC module.

```
enum status_code sdadc_enable(
        struct sdadc_module *const module_inst)
```

Enables an SDADC module that has previously been configured. If any internal reference is selected it will be enabled.

**Table 7-16 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |

#### 7.4.3.3. Function sdadc_disable()

Disables the SDADC module.

```
enum status_code sdadc_disable(
        struct sdadc_module *const module_inst)
```

Disables an SDADC module that was previously enabled.

**Table 7-17 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |

#### 7.4.3.4. Function sdadc_reset()

Resets the SDADC module.

```
enum status_code sdadc_reset(
        struct sdadc_module *const module_inst)
```

Resets an SDADC module, clearing all module state, and registers to their default values.

**Table 7-18 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |

#### 7.4.3.5. Function sdadc_enable_events()

Enables an SDADC event input or output.

```
void sdadc_enable_events(
        struct sdadc_module *const module_inst,
        struct sdadc_events *const events)
```

Enables one or more input or output events to or from the SDADC module. See sdadc_events for a list of events this module supports.

**Note:** Events cannot be altered while the module is enabled.

**Table 7-19 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Software instance for the SDADC peripheral |
| **[in]** | events | Struct containing flags of events to enable |

#### 7.4.3.6. Function sdadc_disable_events()

Disables an SDADC event input or output.

```
void sdadc_disable_events(
        struct sdadc_module *const module_inst,
        struct sdadc_events *const events)
```

Disables one or more input or output events to or from the SDADC module. See sdadc_events for a list of events this module supports.

**Note:** Events cannot be altered while the module is enabled.

**Table 7-20 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Software instance for the SDADC peripheral |
| **[in]** | events | Struct containing flags of events to disable |

#### 7.4.3.7. Function sdadc_start_conversion()

Starts an SDADC conversion.

```
void sdadc_start_conversion(
        struct sdadc_module *const module_inst)
```

Starts a new SDADC conversion.

**Table 7-21 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |

#### 7.4.3.8. Function sdadc_read()

Reads the SDADC result.

```
enum status_code sdadc_read(
        struct sdadc_module *const module_inst,
        int32_t * result)
```

Reads the result from an SDADC conversion that was previously started.

**Table 7-22 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[out]** | result | Pointer to store the result value in |

**Returns**
Status of the SDADC read request.

**Table 7-23 Return Values**

| Return value | Description |
|---|---|
| STATUS_OK | The result was retrieved successfully |
| STATUS_BUSY | A conversion result was not ready |
| STATUS_ERR_OVERFLOW | The result register has been overwritten by the SDADC module before the result was read by the software |

### 7.4.4. Runtime Changes of SDADC Module

#### 7.4.4.1. Function sdadc_flush()

Flushes the SDADC pipeline.

```
void sdadc_flush(
        struct sdadc_module *const module_inst)
```

Flushes the pipeline and restart the SDADC clock on the next peripheral clock edge. All conversions in progress will be lost. When flush is complete, the module will resume where it left off.

**Table 7-24 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |

#### 7.4.4.2. Function sdadc_set_window_mode()

Sets the SDADC window mode.

```
void sdadc_set_window_mode(
        struct sdadc_module *const module_inst,
        const enum sdadc_window_mode window_mode,
        const int16_t window_lower_value,
        const int16_t window_upper_value)
```

Sets the SDADC window mode to a given mode and value range.

**Table 7-25 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | window_mode | Window monitor mode to set |
| **[in]** | window_lower_value | Lower window monitor threshold value |
| **[in]** | window_upper_value | Upper window monitor threshold value |

#### 7.4.4.3. Function sdadc_set_mux_input()

Sets MUX SDADC input pin.

```
void sdadc_set_mux_input(
        struct sdadc_module *const module_inst,
        const enum sdadc_mux_input mux_input)
```

Sets the MUX SDADC input pin selection.

**Table 7-26 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | mux_input | MUX input pin |

### 7.4.5. Enable and Disable Interrupts

#### 7.4.5.1. Function sdadc_enable_interrupt()

Enable interrupt.

```
void sdadc_enable_interrupt(
        struct sdadc_module *const module_inst,
        enum sdadc_interrupt_flag interrupt)
```

Enable the given interrupt request from the SDADC module.

**Table 7-27 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | interrupt | Interrupt to enable |

#### 7.4.5.2. Function sdadc_disable_interrupt()

Disable interrupt.

```
void sdadc_disable_interrupt(
        struct sdadc_module *const module_inst,
        enum sdadc_interrupt_flag interrupt)
```

Disable the given interrupt request from the SDADC module.

**Table 7-28 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | interrupt | Interrupt to disable |

### 7.4.6. Callback Management

#### 7.4.6.1. Function sdadc_register_callback()

Registers a callback.

```
void sdadc_register_callback(
        struct sdadc_module *const module,
        sdadc_callback_t callback_func,
        enum sdadc_callback callback_type)
```

Registers a callback function which is implemented by the user.

**Note:** The callback must be enabled by for the interrupt handler to call it when the condition for the callback is met.

**Table 7-29 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module | Pointer to SDADC software instance struct |
| **[in]** | callback_func | Pointer to callback function |
| **[in]** | callback_type | Callback type given by an enum |

#### 7.4.6.2. Function sdadc_unregister_callback()

Unregisters a callback.

```
void sdadc_unregister_callback(
        struct sdadc_module * module,
        enum sdadc_callback callback_type)
```

Unregisters a callback function which is implemented by the user.

**Table 7-30 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module | Pointer to SDADC software instance struct |
| **[in]** | callback_type | Callback type given by an enum |

#### 7.4.6.3. Function sdadc_enable_callback()

Enables callback.

```
void sdadc_enable_callback(
        struct sdadc_module *const module,
        enum sdadc_callback callback_type)
```

Enables the callback function registered by sdadc_register_callback. The callback function will be called from the interrupt handler when the conditions for the callback type are met.

**Table 7-31 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module | Pointer to SDADC software instance struct |
| **[in]** | callback_type | Callback type given by an enum |

**Returns**
Status of the operation.

**Table 7-32 Return Values**

| Return value | Description |
|---|---|
| STATUS_OK | If operation was completed |
| STATUS_ERR_INVALID | If operation was not completed, due to invalid callback_type |

#### 7.4.6.4. Function sdadc_disable_callback()

Disables callback.

```
void sdadc_disable_callback(
        struct sdadc_module *const module,
        enum sdadc_callback callback_type)
```

Disables the callback function registered by the sdadc_register_callback.

**Table 7-33 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module | Pointer to SDADC software instance struct |
| **[in]** | callback_type | Callback type given by an enum |

**Returns**
Status of the operation.

**Table 7-34 Return Values**

| Return value | Description |
|---|---|
| STATUS_OK | If operation was completed |
| STATUS_ERR_INVALID | If operation was not completed, due to invalid callback_type |

### 7.4.7. Job Management

#### 7.4.7.1. Function sdadc_read_buffer_job()

Read multiple samples from SDADC.

```
enum status_code sdadc_read_buffer_job(
        struct sdadc_module *const module_inst,
        int32_t * buffer,
        uint16_t samples)
```

Read `samples` from the SDADC into the `buffer`. If there is no hardware trigger defined (event action) the driver will retrigger the SDADC conversion whenever a conversion is complete until `samples` has been acquired. To avoid jitter in the sampling frequency using an event trigger is advised.

**Table 7-35 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | samples | Number of samples to acquire |
| **[out]** | buffer | Buffer to store the SDADC samples |

**Returns**
Status of the job start.

**Table 7-36 Return Values**

| Return value | Description |
|---|---|
| STATUS_OK | The conversion job was started successfully and is in progress |
| STATUS_BUSY | The SDADC is already busy with another job |

#### 7.4.7.2. Function sdadc_get_job_status()

Gets the status of a job.

```
enum status_code sdadc_get_job_status(
        struct sdadc_module * module_inst,
        enum sdadc_job_type type)
```

Gets the status of an ongoing or the last job.

**Table 7-37 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | type | Type of job to abort |

**Returns**
Status of the job.

#### 7.4.7.3. Function sdadc_abort_job()

Aborts an ongoing job.

```
void sdadc_abort_job(
        struct sdadc_module * module_inst,
        enum sdadc_job_type type)
```

Aborts an ongoing job with given type.

**Table 7-38 Parameters**

| Data direction | Parameter name | Description |
|---|---|---|
| **[in]** | module_inst | Pointer to the SDADC software instance struct |
| **[in]** | type | Type of job to abort |

## 7.5. Enumeration Definitions

### 7.5.1. Enum sdadc_callback

Callback types for SDADC callback driver.

**Table 7-39 Members**

| Enum value | Description |
|---|---|
| SDADC_CALLBACK_READ_BUFFER | Callback for buffer received |
| SDADC_CALLBACK_WINDOW | Callback when window is hit |
| SDADC_CALLBACK_ERROR | Callback for error |

### 7.5.2. Enum sdadc_event_action

Enum for the possible actions to take on an incoming event.

**Table 7-40 Members**

| Enum value | Description |
|---|---|
| SDADC_EVENT_ACTION_DISABLED | Event action disabled |
| SDADC_EVENT_ACTION_FLUSH_START_CONV | Flush SDADC and start conversion |
| SDADC_EVENT_ACTION_START_CONV | Start conversion |

### 7.5.3. Enum sdadc_interrupt_flag

Enum for the possible SDADC interrupt flags.

**Table 7-41 Members**

| Enum value | Description |
|---|---|
| SDADC_INTERRUPT_RESULT_READY | SDADC result ready |
| SDADC_INTERRUPT_OVERRUN | SDADC result overwritten before read |
| SDADC_INTERRUPT_WINDOW | Window monitor match |

### 7.5.4. Enum sdadc_job_type

Enum for the possible types of SDADC asynchronous jobs that may be issued to the driver.

**Table 7-42 Members**

| Enum value | Description |
|---|---|
| SDADC_JOB_READ_BUFFER | Asynchronous SDADC read into a user provided buffer |

### 7.5.5. Enum sdadc_mux_input

Enum for the possible MUX input selections for the SDADC.

**Table 7-43 Members**

| Enum value | Description |
|---|---|
| SDADC_MUX_INPUT_AIN0 | Select SDADC AINN0 and AINP0 pins |
| SDADC_MUX_INPUT_AIN1 | Select SDADC AINN1 and AINP1 pins |
| SDADC_MUX_INPUT_AIN2 | Select SDADC AINN2 and AINP2 pins |

### 7.5.6. Enum sdadc_over_sampling_ratio

Enum for the over sampling ratio, which change the output data rate.

**Table 7-44 Members**

| Enum value | Description |
|---|---|
| SDADC_OVER_SAMPLING_RATIO64 | SDADC over Sampling Ratio is 64 |
| SDADC_OVER_SAMPLING_RATIO128 | SDADC over Sampling Ratio is 128 |
| SDADC_OVER_SAMPLING_RATIO256 | SDADC over Sampling Ratio is 256 |
| SDADC_OVER_SAMPLING_RATIO512 | SDADC over Sampling Ratio is 512 |
| SDADC_OVER_SAMPLING_RATIO1024 | SDADC over Sampling Ratio is 1024 |

### 7.5.7. Enum sdadc_reference_range

Enum for the matched voltage range of the SDADC reference used.

**Table 7-45 Members**

| Enum value | Description |
|---|---|
| SDADC_REFRANGE_0 | Vref < 1.4V |
| SDADC_REFRANGE_1 | 1.4V < Vref < 2.4V |
| SDADC_REFRANGE_2 | 2.4V < Vref < 3.6V |
| SDADC_REFRANGE_3 | Vref > 3.6V |

### 7.5.8. Enum sdadc_reference_select

Enum for the possible reference voltages for the SDADC.

**Table 7-46 Members**

| Enum value | Description |
|---|---|
| SDADC_REFERENCE_INTREF | Internal Bandgap Reference |
| SDADC_REFERENCE_AREFB | External reference B |
| SDADC_REFERENCE_DACOUT | DACOUT |
| SDADC_REFERENCE_INTVCC | VDDANA |

### 7.5.9. Enum sdadc_window_mode

Enum for the possible window monitor modes for the SDADC.

**Table 7-47 Members**

| Enum value | Description |
|---|---|
| SDADC_WINDOW_MODE_DISABLE | No window mode |
| SDADC_WINDOW_MODE_ABOVE | RESULT > WINLT |
| SDADC_WINDOW_MODE_BELOW | RESULT < WINUT |
| SDADC_WINDOW_MODE_INSIDE | WINLT < RESULT < WINUT |
| SDADC_WINDOW_MODE_OUTSIDE | !(WINLT < RESULT < WINUT) |

# 8. Extra Information for SDADC Driver

## 8.1. Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

| Acronym | Description |
|---|---|
| SDADC | Sigma-Delta Analog-to-Digital Converter |
| OSR | Over Sampling Ratio |

## 8.2. Dependencies

This driver has no dependencies.

## 8.3. Errata

There are no errata related to this driver.

## 8.4. Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

| Changelog |
|---|
| Initial Release |

# 9. Examples for SDADC Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM Sigma-Delta Analog-to-Digital Converter (SDADC) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for SDADC - Basic
- Quick Start Guide for SDADC - Callback

## 9.1. Quick Start Guide for SDADC - Basic

In this use case, the SDADC will be configured with the following settings:

- GCLK generator 0 (GCLK main) clock source
- Internal bandgap reference 1V
- Div 2 clock prescaler
- Over Sampling Ratio is 64
- Skip 2 samples
- MUX input on SDADC AIN1
- All events (input and generation) disabled
- Free running disabled
- Run in standby disabled
- On command disabled
- Disable all positive input in sequence
- Window monitor disabled
- No gain/offset/shift correction

### 9.1.1. Setup

#### 9.1.1.1. Prerequisites

There are no special setup requirements for this use-case.

#### 9.1.1.2. Code

Add to the main application source file, outside of any functions:

```
struct sdadc_module sdadc_instance;
```

Copy-paste the following setup code to your user application:

```
void configure_sdadc(void)
{
    struct sdadc_config config_sdadc;
    sdadc_get_config_defaults(&config_sdadc);

    sdadc_init(&sdadc_instance, SDADC, &config_sdadc);

    sdadc_enable(&sdadc_instance);
}
```

Add to user application initialization (typically the start of `main()`):

```
configure_sdadc();
```

#### 9.1.1.3. Workflow

1. Create a module software instance structure for the SDADC module to store the SDADC driver state while it is in use.

   ```
   struct sdadc_module sdadc_instance;
   ```

   **Note:** This should never go out of scope as long as the module is in use. In most cases, this should be global.

2. Configure the SDADC module.
   - Create a SDADC module configuration struct, which can be filled out to adjust the configuration of a physical SDADC peripheral.

     ```
     struct sdadc_config config_sdadc;
     ```

   - Initialize the SDADC configuration struct with the module's default values.

     ```
     sdadc_get_config_defaults(&config_sdadc);
     ```

     **Note:** This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.
   - Set SDADC configurations.

     ```
     sdadc_init(&sdadc_instance, SDADC, &config_sdadc);
     ```

   - Enable the SDADC module so that conversions can be made.

     ```
     sdadc_enable(&sdadc_instance);
     ```

### 9.1.2. Use Case

#### 9.1.2.1. Code

Copy-paste the following code to your user application:

```
sdadc_start_conversion(&sdadc_instance);

int32_t result;

do {
    /* Wait for conversion to be done and read out result */
} while (sdadc_read(&sdadc_instance, &result) == STATUS_BUSY);

while (1) {
    /* Infinite loop */
}
```

#### 9.1.2.2. Workflow

1. Start conversion.

   ```
   sdadc_start_conversion(&sdadc_instance);
   ```

2. Wait until conversion is done and read result.

   ```
   int32_t result;

   do {
   ```

```
    /* Wait for conversion to be done and read out result */
} while (sdadc_read(&sdadc_instance, &result) == STATUS_BUSY);
```

3. Enter an infinite loop once the conversion is complete.

```
while (1) {
    /* Infinite loop */
}
```

## 9.2. Quick Start Guide for SDADC - Callback

In this use case, the SDADC will convert 128 samples using interrupt driven conversion. When all samples have been sampled, a callback will be called that signals the main application that conversion is compete.

The SDADC will be set up as follows:

- GCLK generator 0 (GCLK main) clock source
- Internal bandgap reference 1V
- Div 2 clock prescaler
- Over Sampling Ratio is 1024
- Skip 2 samples
- MUX input on SDADC AIN2
- All events (input and generation) disabled
- Free running disabled
- Run in standby disabled
- On command disabled
- Disable all positive input in sequence
- Window monitor disabled
- No gain/offset/shift correction

### 9.2.1. Setup

#### 9.2.1.1. Prerequisites

There are no special setup requirements for this use-case.

#### 9.2.1.2. Code

Add to the main application source file, outside of any functions:

```
struct sdadc_module sdadc_instance;
```

```
#define SDADC_SAMPLES 128
int32_t sdadc_result_buffer[SDADC_SAMPLES];
```

Callback function:

```
volatile bool sdadc_read_done = false;

void sdadc_complete_callback(
        const struct sdadc_module *const module)
{
    sdadc_read_done = true;
}
```

Copy-paste the following setup code to your user application:

```
void configure_sdadc(void)
{
    struct sdadc_config config_sdadc;
    sdadc_get_config_defaults(&config_sdadc);

    config_sdadc.clock_prescaler = 2;
    config_sdadc.reference.ref_sel = SDADC_REFERENCE_INTREF;
    config_sdadc.osr             = SDADC_OVER_SAMPLING_RATIO1024;
    config_sdadc.mux_input       = SDADC_MUX_INPUT_AIN2;

    sdadc_init(&sdadc_instance, SDADC, &config_sdadc);

    sdadc_enable(&sdadc_instance);
}

void configure_sdadc_callbacks(void)
{
    sdadc_register_callback(&sdadc_instance,
            sdadc_complete_callback, SDADC_CALLBACK_READ_BUFFER);
    sdadc_enable_callback(&sdadc_instance, SDADC_CALLBACK_READ_BUFFER);
}
```

Add to user application initialization (typically the start of `main()`):

```
configure_sdadc();
configure_sdadc_callbacks();
```

#### 9.2.1.3. Workflow

1. Create a module software instance structure for the SDADC module to store the SDADC driver state while it is in use.

   ```
   struct sdadc_module sdadc_instance;
   ```

   **Note:** This should never go out of scope as long as the module is in use. In most cases, this should be global.

2. Create a buffer for the SDADC samples to be stored in by the driver asynchronously.

   ```
   #define SDADC_SAMPLES 128
   int32_t sdadc_result_buffer[SDADC_SAMPLES];
   ```

3. Create a callback function that will be called each time the SDADC completes an asynchronous read job.

   ```
   volatile bool sdadc_read_done = false;

   void sdadc_complete_callback(
           const struct sdadc_module *const module)
   {
       sdadc_read_done = true;
   }
   ```

4. Configure the SDADC module.
   - Create a SDADC module configuration struct, which can be filled out to adjust the configuration of a physical SDADC peripheral.

     ```
     struct sdadc_config config_sdadc;
     ```

- Initialize the SDADC configuration struct with the module's default values.

```
sdadc_get_config_defaults(&config_sdadc);
```

**Note:** This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

- Change the SDADC module configuration to suit the application.

```
config_sdadc.clock_prescaler = 2;
config_sdadc.reference.ref_sel = SDADC_REFERENCE_INTREF;
config_sdadc.osr             = SDADC_OVER_SAMPLING_RATIO1024;
config_sdadc.mux_input       = SDADC_MUX_INPUT_AIN2;
```

- Set SDADC configurations.

```
sdadc_init(&sdadc_instance, SDADC, &config_sdadc);
```

- Enable the SDADC module so that conversions can be made.

```
sdadc_enable(&sdadc_instance);
```

5. Register and enable the SDADC read buffer complete callback handler.
   - Register the user-provided read buffer complete callback function with the driver, so that it will be run when an asynchronous buffer read job completes.

```
sdadc_register_callback(&sdadc_instance,
        sdadc_complete_callback, SDADC_CALLBACK_READ_BUFFER);
```

   - Enable the read buffer complete callback so that it will generate callbacks.

```
sdadc_enable_callback(&sdadc_instance,
SDADC_CALLBACK_READ_BUFFER);
```

### 9.2.2. Use Case

#### 9.2.2.1. Code

Copy-paste the following code to your user application:

```
system_interrupt_enable_global();

sdadc_read_buffer_job(&sdadc_instance, sdadc_result_buffer, SDADC_SAMPLES);

while (sdadc_read_done == false) {
    /* Wait for asynchronous SDADC read to complete */
}

while (1) {
    /* Infinite loop */
}
```

#### 9.2.2.2. Workflow

- Enable global interrupts, so that callbacks can be generated by the driver.

```
system_interrupt_enable_global();
```

- Start an asynchronous SDADC conversion, to store SDADC samples into the global buffer and generate a callback when complete.

```
sdadc_read_buffer_job(&sdadc_instance, sdadc_result_buffer,
SDADC_SAMPLES);
```

- Wait until the asynchronous conversion is complete.

```
while (sdadc_read_done == false) {
    /* Wait for asynchronous SDADC read to complete */
}
```

- Enter an infinite loop once the conversion is complete.

```
while (1) {
    /* Infinite loop */
}
```

# 10. Document Revision History

| Doc. Rev. | Date | Comments |
|---|---|---|
| 42496A | 09/2015 | Initial document release |

**Atmel Corporation** 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**