APPLICATION NOTE

AT06860: SAM3/4S/4C Analog-to-digital Converter (ADC)

ASF PROGRAMMERS MANUAL

SAM3/4S/4C Analog-to-digital Converter (ADC)

This document describes the usage of the driver for the ADC module of the SAM3 and SAM4 range of microcontrollers.

- Prerequisites
- Module Overview
- Examples
- API Overview
- Special Considerations
- Extra Information

Table of Contents

SAM3/4S/4C Analog-to-digital Converter (ADC) 1

Software License 4

1. Prerequisites 5

2. Module Overview 6

3. Special Considerations 7

4. Extra Information 8

4.1. Terms and Definitions 8

5. Examples 9

6. API Overview 10

6.1. Macro Definitions 10
6.1.1. Macro ADC_FREQ_MAX 10
6.1.2. Macro ADC_FREQ_MIN 10
6.1.3. Macro ADC_STARTUP_FAST 10
6.1.4. Macro ADC_STARTUP_NORM 10
6.2. Function Definitions 10
6.2.1. Function adc_check() 10
6.2.2. Function adc_configure_sequence() 10
6.2.3. Function adc_configure_timing() 11
6.2.4. Function adc_configure_trigger() 11
6.2.5. Function adc_disable_all_channel() 12
6.2.6. Function adc_disable_anch() 12
6.2.7. Function adc_disable_channel() 12
6.2.8. Function adc_disable_channel_differential_input() 13
6.2.9. Function adc_disable_channel_input_offset() 13
6.2.10. Function adc_disable_interrupt() 13
6.2.11. Function adc_disable_tag() 14
6.2.12. Function adc_enable_all_channel() 14
6.2.13. Function adc_enable_anch() 14
6.2.14. Function adc_enable_channel() 14
6.2.15. Function adc_enable_channel_differential_input() 15
6.2.16. Function adc_enable_channel_input_offset() 15
6.2.17. Function adc_enable_interrupt() 15
6.2.18. Function adc_enable_tag() 16
6.2.19. Function adc_get_actual_adc_clock() 16
6.2.20. Function adc_get_channel_status() 16
6.2.21. Function adc_get_channel_value() 17
6.2.22. Function adc_get_comparison_mode() 17
6.2.23. Function adc_get_interrupt_mask() 17
6.2.24. Function adc_get_latest_value() 18
6.2.25. Function adc_get_overrun_status() 18
6.2.26. Function adc_get_pdc_base() 18
6.2.27. Function adc_get_status() 19
6.2.28. Function adc_get_tag() 19
6.2.29. Function adc_get_writeprotect_status() 19
6.2.30. Function adc_init() 20
6.2.31. Function adc_set_bias_current() 20
6.2.32. Function adc_set_channel_input_gain() 21
6.2.33. Function adc_set_comparison_channel() 21
6.2.34. Function adc_set_comparison_filter() 22
6.2.35. Function adc_set_comparison_mode() 22
6.2.36. Function adc_set_comparison_window() 22

6.2.37. Function adc_set_resolution() 22
6.2.38. Function adc_set_writeprotect() 23
6.2.39. Function adc_start() 23
6.2.40. Function adc_start_sequencer() 23
6.2.41. Function adc_stop() 24
6.2.42. Function adc_stop_sequencer() 24
6.3. Enumeration Definitions 24
6.3.1. Enum adc_gainvalue_t 24
6.3.2. Enum adc_trigger_t 24

7. Quickstart guide for SAM ADC driver 26
7.1. Basic Use Case 26
7.1.1. Prerequisites 26
7.2. Setup Steps 26
7.2.1. Example Code 26
7.2.2. Workflow 27
7.3. Usage Steps 28
7.3.1. Example Code 28
7.3.2. Workflow 28
7.4. Advanced Use Cases 28
7.5. Use case #1 28
7.5.1. Setup Steps 29
7.5.2. Usage Steps 30

8. ADC Enhanced Resolution Example - Potentiometer 31
8.1. Purpose 31
8.2. Requirements 31
8.3. Description 31
8.4. Usage 31

9. Simple Example 32
9.1. Purpose 32
9.2. Requirements 32
9.3. Usage 32

10. ADC Threshold Wakeup Example 34
10.1. Purpose 34
10.2. Requirements 34
10.3. Description 34
10.4. Usage 34

Index 36

Document Revision History 37

Software License

1. Prerequisites

There are no prerequisites for this module.

2. Module Overview

This driver provides an interface for the Analog-to-Digital conversion functions on the device. This is designed to convert analog input voltages to corresponding digital values.

The ADC has up to 12-bit resolution, and is capable of converting up to 1 million samples per second (ksps).

See Quickstart guide for SAM ADC driver.

3. Special Considerations

NONE.

4. Extra Information

Some of the functions described below in the API section, have slightly different call and return parameters due to variations in the functionality of the various devices. Check for notes to this effect.

4.1 Terms and Definitions

Term	Definition
PDC	Peripheral DMA Control Register
ADTRG	ADC Trigger

5. Examples

Note

The examples listed below present a simple user interface, and messages, to a USB serial port. This port can be connected to by using a terminal emulator (such as Hyperterm, Terraterm, or PuTTY). The correct port can be identified by starting 'Device Manager' and expanding the tab labelled "Ports (COM and LPT)."

- Simple Example
- ADC Enhanced Resolution Example - Potentiometer
- ADC Threshold Wakeup Example

See also Quickstart guide for SAM ADC driver.

6. API Overview

6.1 Macro Definitions

6.1.1 Macro ADC_FREQ_MAX

```
#define ADC_FREQ_MAX
```

The maximum ADC clock frequency. The value of this is dependent on the device in use. e.g. for SAM4C it is 20000000, otherwise 16000000.

6.1.2 Macro ADC_FREQ_MIN

```
#define ADC_FREQ_MIN
```

The min ADC clock freq definition, set to 1000000.

6.1.3 Macro ADC_STARTUP_FAST

```
#define ADC_STARTUP_FAST
```

6.1.4 Macro ADC_STARTUP_NORM

```
#define ADC_STARTUP_NORM
```

6.2 Function Definitions

6.2.1 Function adc_check()

Check ADC configurations. When called this routine tests the validity of the values in the p_adc structure, and reports errors if they are not correct.

```
void adc_check(
  Adc * p_adc,
  const uint32_t ul_mck)
```

Table 6-1. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	ul_mck	Main clock of the device (in Hz)

6.2.2 Function adc_configure_sequence()

Configure conversion sequence.

```
void adc_configure_sequence(
  Adc * p_adc,
  const enum adc_channel_num_t ch_list,
  const uint8_t uc_num)
```

Table 6-2. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	ch_list	Channel sequence list
[in]	uc_num	Number of channels in the list

Note Check the device datasheet for adc sequencer support.

6.2.3 Function adc_configure_timing()

Configure ADC timing.

```
void adc_configure_timing(
  Adc * p_adc,
  const uint8_t uc_tracking,
  const enum adc_settling_time_t settling,
  const uint8_t uc_transfer)
```

Table 6-3. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	uc_tracking	ADC tracking time = uc_tracking / ADC clock
[in]	uc_settling	Analog settling time = (uc_settling + 1) / ADC clock
[in]	uc_transfer	Data transfer time = (uc_transfer * 2 + 3) / ADC clock

Note This API applies to SAM3S, SAM4S, and SAM3XA only.

6.2.4 Function adc_configure_trigger()

Configure conversion trigger and free run mode.

```
void adc_configure_trigger(
  Adc * p_adc,
  const enum adc_trigger_t trigger,
  const uint8_t uc_freerun)
```

Table 6-4. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	trigger	Conversion trigger
[in]	uc_freerun	ADC_MR_FREERUN_ON enables freerun mode, ADC_MR_FREERUN_OFF disables freerun mode.

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.5 Function adc_disable_all_channel()

Disable all ADC channels.

```
void adc_disable_all_channel(
  Adc * p_adc)
```

Table 6-5. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

6.2.6 Function adc_disable_anch()

Disable analog change.

```
void adc_disable_anch(
  Adc * p_adc)
```

Note DIFF0, GAIN0, and OFF0 are used for all channels.

Table 6-6. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

6.2.7 Function adc_disable_channel()

Disable the specified ADC channel.

```
void adc_disable_channel(
  Adc * p_adc,
  const enum adc_channel_num_t adc_ch)
```

Table 6-7. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Data direction	Parameter name	Description
[in]	adc_ch	ADC channel number

6.2.8 Function adc_disable_channel_differential_input()

Disable differential input for the specified channel.

```
void adc_disable_channel_differential_input(
  Adc * p_adc,
  const enum adc_channel_num_t channel)
```

Table 6-8. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	channel	ADC channel number

Note This function is only supported by the SAM3S, SAM3XA, and SAM4S devices.

6.2.9 Function adc_disable_channel_input_offset()

Disable analog signal offset for the specified channel.

```
void adc_disable_channel_input_offset(
  Adc * p_adc,
  const enum adc_channel_num_t channel)
```

Table 6-9. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	channel	ADC channel number

Note This function is only supported by the SAM3S, SAM3XA, and SAM4S devices.

6.2.10 Function adc_disable_interrupt()

Disable ADC interrupts.

```
void adc_disable_interrupt(
  Adc * p_adc,
  const uint32_t ul_source)
```

Table 6-10. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Data direction	Parameter name	Description
[in]	ul_source	Interrupts to be disabled

6.2.11 Function adc_disable_tag()

Disable TAG option.

```
void adc_disable_tag(
  Adc * p_adc)
```

Table 6-11. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.12 Function adc_enable_all_channel()

Enable all ADC channels.

```
void adc_enable_all_channel(
  Adc * p_adc)
```

Table 6-12. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

6.2.13 Function adc_enable_anch()

Enable analog change.

```
void adc_enable_anch(
  Adc * p_adc)
```

Note It allows different analog settings for each channel.

Table 6-13. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

6.2.14 Function adc_enable_channel()

Enable the specified ADC channel.

```
void adc_enable_channel(
  Adc * p_adc,
  const enum adc_channel_num_t adc_ch)
```

Table 6-14. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	adc_ch	ADC channel number

6.2.15 Function adc_enable_channel_differential_input()

Enable differential input for the specified channel.

```
void adc_enable_channel_differential_input(
  Adc * p_adc,
  const enum adc_channel_num_t channel)
```

Table 6-15. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	channel	ADC channel number

Note This function is only supported by the SAM3S, SAM3XA, and SAM4S devices.

6.2.16 Function adc_enable_channel_input_offset()

Enable analog signal offset for the specified channel.

```
void adc_enable_channel_input_offset(
  Adc * p_adc,
  const enum adc_channel_num_t channel)
```

Table 6-16. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	channel	ADC channel number

Note This function is only supported by the SAM3S, SAM3XA, and SAM4S devices.

6.2.17 Function adc_enable_interrupt()

Enable ADC interrupts.

```
void adc_enable_interrupt(
  Adc * p_adc,
  const uint32_t ul_source)
```

Table 6-17. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	ul_source	Interrupts to be enabled

6.2.18 Function adc_enable_tag()

Enable TAG option so that the number of the last converted channel can be indicated.

```
void adc_enable_tag(
  Adc * p_adc)
```

Table 6-18. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Note

This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.19 Function adc_get_actual_adc_clock()

Return the actual ADC clock.

```
uint32_t adc_get_actual_adc_clock(
  const Adc * p_adc,
  const uint32_t ul_mck)
```

Table 6-19. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	ul_mck	Main clock of the device (in Hz)

Returns

The actual ADC clock (in Hz).

6.2.20 Function adc_get_channel_status()

Read the ADC channel status.

```
uint32_t adc_get_channel_status(
  const Adc * p_adc,
  const enum adc_channel_num_t adc_ch)
```

Table 6-20. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	adc_ch	ADC channel number

Table 6-21. Return Values

Return value	Description
1	if channel is enabled
0	if channel is disabled

6.2.21 Function adc_get_channel_value()

Read the ADC result data of the specified channel.

```
uint32_t adc_get_channel_value(
  const Adc * p_adc,
  const enum adc_channel_num_t adc_ch)
```

Table 6-22. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	adc_ch	ADC channel number

Returns

ADC value of the specified channel.

6.2.22 Function adc_get_comparison_mode()

Get comparison mode.

```
uint32_t adc_get_comparison_mode(
  const Adc * p_adc)
```

Table 6-23. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Table 6-24. Return Values

Return value	Description
Compare	mode value

Note

This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.23 Function adc_get_interrupt_mask()

Read ADC interrupt mask.

```
uint32_t adc_get_interrupt_mask(
  const Adc * p_adc)
```

Table 6-25. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Returns

The interrupt mask value.

6.2.24 Function adc_get_latest_value()

Read the last ADC result data.

```
uint32_t adc_get_latest_value(
  const Adc * p_adc)
```

Table 6-26. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Returns

ADC latest value.

6.2.25 Function adc_get_overrun_status()

Get ADC interrupt and overrun error status.

```
uint32_t adc_get_overrun_status(
  const Adc * p_adc)
```

Table 6-27. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Returns

ADC status structure.

6.2.26 Function adc_get_pdc_base()

Get PDC registers base address.

```
Pdc * adc_get_pdc_base(
  const Adc * p_adc)
```

Table 6-28. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Returns ADC PDC register base address.

6.2.27 Function adc_get_status()

Get ADC interrupt and overrun error status.

```
uint32_t adc_get_status(
  const Adc * p_adc)
```

Table 6-29. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Returns ADC status structure.

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.28 Function adc_get_tag()

Indicate the last converted channel.

```
enum adc_channel_num_t adc_get_tag(
  const Adc * p_adc)
```

Note If TAG option is NOT enabled before, an incorrect channel number is returned.

Table 6-30. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Returns The last converted channel number.

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.29 Function adc_get_writeprotect_status()

Indicate write protect status.

```
uint32_t adc_get_writeprotect_status(
  const Adc * p_adc)
```

Table 6-31. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Returns 0 if the peripheral is not protected, or 16-bit write protect violation Status.

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.30 Function adc_init()

Initialize the given ADC with the specified ADC clock and startup time.

```
uint32_t adc_init(
  Adc * p_adc,
  const uint32_t ul_mck,
  const uint32_t ul_adc_clock,
  const enum adc_startup_time startup)
```

Table 6-32. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	ul_mck	Main clock of the device (value in Hz)
[in]	ul_adc_clock	Analog-to-Digital conversion clock (value in Hz)
[in]	uc_startup	ADC start up time. Refer to the product datasheet for details.

Returns 0 on success.

Note Present for sam3s, sam3n, sam3xa, sam4s, and sam4c devices. Refer to the product datasheet for details.

6.2.31 Function adc_set_bias_current()

Adapt performance versus power consumption.

```
void adc_set_bias_current(
  Adc * p_adc,
  const uint8_t uc_ibctl)
```

Note Refer to adc characteristics in the product datasheet for more details.

Table 6-33. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	uc_ibctl	ADC Bias current control

Note This API applies to SAM3S, SAM4S, and SAM3XA only.

6.2.32 Function adc_set_channel_input_gain()

Configure input gain for the specified channel.

```
void adc_set_channel_input_gain(
  Adc * p_adc,
  const enum adc_channel_num_t channel,
  const enum adc_gainvalue_t uc_gain)
```

Table 6-34. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	channel	ADC channel number
[in]	gain	Gain value for the input

Note This function is only supported by the SAM3S, SAM3XA, and SAM4S devices.

6.2.33 Function adc_set_comparison_channel()

Configure comparison selected channel.

```
void adc_set_comparison_channel(
  Adc * p_adc,
  const enum adc_channel_num_t channel)
```

Table 6-35. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	channel	ADC channel number

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.34 Function adc_set_comparison_filter()

```
void adc_set_comparison_filter(
  Adc * p_adc,
  uint8_t filter)
```

6.2.35 Function adc_set_comparison_mode()

Configure comparison mode.

```
void adc_set_comparison_mode(
  Adc * p_adc,
  const uint8_t uc_mode)
```

Table 6-36. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	uc_mode	ADC comparison mode

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.36 Function adc_set_comparison_window()

Configure ADC compare window.

```
void adc_set_comparison_window(
  Adc * p_adc,
  const uint16_t us_low_threshold,
  const uint16_t us_high_threshold)
```

Table 6-37. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	us_low_threshold	Low threshold of compare window
[in]	us_high_threshold	High threshold of compare window

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.37 Function adc_set_resolution()

Configure the conversion resolution.

```
void adc_set_resolution(
  Adc * p_adc,
  const enum adc_resolution_t resolution)
```

Table 6-38. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	resolution	ADC resolution

6.2.38 Function adc_set_writeprotect()

Enable or disable write protection of ADC registers.

```
void adc_set_writeprotect(
  Adc * p_adc,
  const uint32_t ul_enable)
```

Table 6-39. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance
[in]	ul_enable	1 to enable, 0 to disable

Note

This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.39 Function adc_start()

Start analog-to-digital conversion.

```
void adc_start(
  Adc * p_adc)
```

Note

If one of the hardware events is selected as the ADC trigger, this function can NOT start analog to digital conversion.

Table 6-40. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

6.2.40 Function adc_start_sequencer()

Enable conversion sequencer.

```
void adc_start_sequencer(
  Adc * p_adc)
```

Table 6-41. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.2.41 Function adc_stop()

Stop analog-to-digital conversion.

```
void adc_stop(
  Adc * p_adc)
```

Table 6-42. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

6.2.42 Function adc_stop_sequencer()

Disable conversion sequencer.

```
void adc_stop_sequencer(
  Adc * p_adc)
```

Table 6-43. Parameters

Data direction	Parameter name	Description
[in]	p_adc	Pointer to an ADC instance

Note This API applies to SAM3S, SAM4S, SAM3N, SAM3XA, and SAM4C only.

6.3 Enumeration Definitions

6.3.1 Enum adc_gainvalue_t

Table 6-44. Members

Enum value	Description
ADC_GAINVALUE_0	
ADC_GAINVALUE_1	
ADC_GAINVALUE_2	
ADC_GAINVALUE_3	

6.3.2 Enum adc_trigger_t

Table 6-45. Members

Enum value	Description
ADC_TRIG_SW	

Enum value	Description
ADC_TRIG_EXT	
ADC_TRIG_TIO_CH_0	
ADC_TRIG_TIO_CH_1	
ADC_TRIG_TIO_CH_2	

7. Quickstart guide for SAM ADC driver

This is the quickstart guide for the SAM ADC driver, with step-by-step instructions on how to configure and use the driver in a selection of use cases.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, e.g., the main application function.

7.1 Basic Use Case

In this basic use case, the ADC module and single channel are configured for:

- 12-bit, unsigned conversions
- Internal bandgap as 3.3V reference
- ADC clock rate of at most 6.4MHz and maximum sample rate is 1MHz
- Software triggering of conversions
- Interrupt-based conversion handling
- Single channel measurement
- ADC_CHANNEL_5 as positive input

7.1.1 Prerequisites

1. System Clock Management (Sysclock).
2. Power Management Controller (PMC).

7.2 Setup Steps

7.2.1 Example Code

Add to application C-file:

```
void ADC_IrqHandler(void)
{
        // Check the ADC conversion status
        if ((adc_get_status(ADC) & ADC_ISR_DRDY) == ADC_ISR_DRDY)
        {
                // Get latest digital data value from ADC.
                uint32_t result = adc_get_latest_value(ADC);
        }
}

void adc_setup(void)
{
        adc_init(ADC, sysclk_get_main_hz(), ADC_CLOCK, 8);

        adc_configure_timing(ADC, 0, ADC_SETTLING_TIME_3, 1);

        adc_set_resolution(ADC, ADC_MR_LOWRES_BITS_12);

        adc_enable_channel(ADC, ADC_CHANNEL_5);

        adc_set_comparison_channel(ADC, ADC_CHANNEL_5);
        adc_set_comparison_mode(ADC, ADC_EMR_CMPMODE_IN);
        adc_set_comparison_window(ADC, us_high_threshold, us_low_threshold);
        adc_enable_interrupt(ADC, ADC_IER_DRDY);

        adc_configure_trigger(ADC, ADC_TRIG_SW, 0);
}
```

```
void adc_setup(void)
{
        adc_init(ADC, sysclk_get_main_hz(), ADC_CLOCK, 8);

        adc_configure_timing(ADC, 0, ADC_SETTLING_TIME_3, 1);

        adc_set_resolution(ADC, ADC_MR_LOWRES_BITS_12);

        adc_enable_channel(ADC, ADC_CHANNEL_5);

        adc_set_comparison_channel(ADC, ADC_CHANNEL_5);
        adc_set_comparison_mode(ADC, ADC_EMR_CMPMODE_IN);
        adc_set_comparison_window(ADC, us_high_threshold, us_low_threshold);

        adc_enable_interrupt(ADC, ADC_IER_DRDY);

        adc_configure_trigger(ADC, ADC_TRIG_SW, 0);
}
```

7.2.2 Workflow

1. Define the interrupt service handler in the application:

```
void ADC_IrqHandler(void)
{
        // Check the ADC conversion status
        if ((adc_get_status(ADC) & ADC_ISR_DRDY) == ADC_ISR_DRDY)
        {
                // Get latest digital data value from ADC.
                uint32_t result = adc_get_latest_value(ADC);
        }
}
```

Note

Get ADC status and check if the conversion is finished. If done, read the last ADC result data.

2. Initialize the given ADC with the specified ADC clock and startup time:

```
adc_init(ADC, sysclk_get_main_hz(), ADC_CLOCK, 8);
```

Note

The ADC clock range is between master clock / 2 and master clock / 512. The function sysclk_get_main_hz() is used to get the master clock frequency while ADC_CLOCK gives the ADC clock frequency.

3. Configure ADC timing:

```
adc_configure_timing(ADC, 0, ADC_SETTLING_TIME_3, 1);
```

Note Tracking Time = (0 + 1) * ADC Clock period Settling Time = ADC_SETTLING_TIME_3 * ADC Clock period Transfer Time = (1 * 2 + 3) * ADC Clock period

4. Set the ADC resolution.

```
adc_set_resolution(ADC, ADC_MR_LOWRES_BITS_12);
```

Note The resolution value can be set to 10 bits or 12 bits.

5. Enable the specified ADC channel:

```
adc_enable_channel(ADC, ADC_CHANNEL_5);
```

6. Enable ADC interrupts:

```
adc_enable_interrupt(ADC, ADC_IER_DRDY);
```

7. Configure software conversion trigger:

```
adc_configure_trigger(ADC, ADC_TRIG_SW, 0);
```

7.3 Usage Steps

7.3.1 Example Code

Add to, e.g., main loop in application C-file:

```
adc_start(ADC);
```

7.3.2 Workflow

1. Start ADC conversion on channel:

```
adc_start(ADC);
```

7.4 Advanced Use Cases

For more advanced use of the ADC driver, see the following use cases:

- Use case #1 : 12-bits unsigned, comparison event happen and interrupt driven

7.5 Use case #1

In this use case the ADC module and one channel are configured for:

- 12-bit, unsigned conversions
- Internal bandgap as 3.3V re/func1ference
- ADC clock rate of at most 6.4MHz and maximum sample rate is 1MHz

- Software triggering of conversions
- Comparison event happen and interrupt handling
- Single channel measurement
- ADC_CHANNEL_5 as positive input

7.5.1 Setup Steps

7.5.1.1 Example Code

Add to application C-file:

```
void adc_setup(void)
{
        adc_init(ADC, sysclk_get_main_hz(), ADC_CLOCK, 8);

        adc_configure_timing(ADC, 0, ADC_SETTLING_TIME_3, 1);

        adc_set_resolution(ADC, ADC_MR_LOWRES_BITS_12);

        adc_enable_channel(ADC, ADC_CHANNEL_5);

        adc_set_comparison_channel(ADC, ADC_CHANNEL_5);
        adc_set_comparison_mode(ADC, ADC_EMR_CMPMODE_IN);
        adc_set_comparison_window(ADC, us_high_threshold, us_low_threshold);

        adc_enable_interrupt(ADC, ADC_IER_DRDY);

        adc_configure_trigger(ADC, ADC_TRIG_SW, 0);
}
```

7.5.1.2 Workflow

1. Define the interrupt service handler in the application:

```
void ADC_IrqHandler(void)
{
        // Check the ADC conversion status
        if ((adc_get_status(ADC) & ADC_ISR_DRDY) == ADC_ISR_DRDY)
        {
                // Get latest digital data value from ADC.
                uint32_t result = adc_get_latest_value(ADC);
        }
}
```

The above code gets the ADC status and checks if a comparison event has occurred. If it has then read the ADC channel value and comparison mode.

2. Initialize the given ADC with the specified ADC clock and startup time:

```
adc_init(ADC, sysclk_get_main_hz(), ADC_CLOCK, 8);
```

The ADC clock range is between master clock/2 and master clock/512. The function sysclk_get_main_hz() is used to get the master clock frequency while ADC_CLOCK gives the ADC clock frequency.

3. Configure ADC timing:

```
adc_configure_timing(ADC, 0, ADC_SETTLING_TIME_3, 1);
```

Settling Time = ADC_SETTLING_TIME_3 * ADC Clock period Transfer Time = (1 * 2 + 3) * ADC Clock period

4. Set the ADC resolution.

```
adc_set_resolution(ADC, ADC_MR_LOWRES_BITS_12);
```

5. Enable the specified ADC channel:

```
adc_enable_channel(ADC, ADC_CHANNEL_5);
```

6. Set the comparison ADC channel, mode, and window:

```
adc_set_comparison_channel(ADC, ADC_CHANNEL_5);
adc_set_comparison_mode(ADC, ADC_EMR_CMPMODE_IN);
adc_set_comparison_window(ADC, us_high_threshold, us_low_threshold);
```

The high and low threshold of comparison window can be set by the user. An event will be generated whenever the converted data is in the comparison window.

7. Enable ADC interrupts:

```
adc_enable_interrupt(ADC, ADC_IER_DRDY);
```

8. Configure software conversion trigger:

```
adc_configure_trigger(ADC, ADC_TRIG_SW, 0);
```

7.5.2 Usage Steps

7.5.2.1 Example Code

Add to, e.g., main loop in application C-file: TBD

```
adc_start(ADC);
```

7.5.2.2 Workflow

1. Start ADC conversion on the configured channels:

```
adc_start(ADC);
```

8. ADC Enhanced Resolution Example - Potentiometer

8.1 Purpose

This example demonstrates how to use the enhanced resolution mode of the microcontroller to sample analog voltages.

8.2 Requirements

This example can be used on SAM4C-EK boards. Refer to the list of available kits at http://www.atmel.com

8.3 Description

The aim of this example is to demonstrate the enhanced resolution mode of the microcontroller. To use this feature, the ADC channel connected to the potentiometer should be enabled. Users can select different resolution modes from the configuration menu of the example.

8.4 Usage

1. Build the program and download it into the evaluation board.

2. On the computer, open and configure a terminal application (e.g., HyperTerminal on Microsoft Windows) with these settings:

   - 115200 bauds
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control

3. In the terminal window, the following text should appear (values depend on the board and the chip used):

```
-- ADC Enhanced Resolution Examplexxx --
-- xxxxxx-xx
-- Compiled: xxx xx xxxx xx:xx:xx --
=========================================================
Menu: press a key to change the resolution mode.
---------------------------------------------------------
-- n: Normal Resolution Mode--
-- e: Enhanced Resolution Mode--
-- q: Quit Configuration--
```

4. The application will output the raw ADC result and the current voltage of potentiometer on the terminal.

9. Simple Example

ADC Example from adc_examples

This application demonstrates the use of many of the ADCs modes. e.g.:

- With/without PDC
- Several sources of trigger (Software, ADTRG, Timer, etc.)
- Gain and offset selection
- Use of the sequencer

Users can select the different modes from the configuration menu.

9.1 Purpose

To provide a demonstration of the various ADC/ADC12B modes.

9.2 Requirements

This example can be used with SAM evaluation kits, such as SAM4S_EK, SAM4C_EK , and other evaluations kits. Refer to the list of available kits at http://www.atmel.com

Note ADVREF must be set to 3300mv in order to enable full scale measurement of the potentiometer. Refer to the board schematics for advref jumper configuration.

We use one push button for ADTRG, so connect ADTRG to relavent button pin

Note On the SAM3S8 channel 15 is used for the TempSensor.

9.3 Usage

1. Build the program and download it into the evaluation board.
2. On the computer, open and configure a terminal application (e.g., HyperTerminal on Microsoft Windows) with these settings:
   - 115200 bauds
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. In the terminal window, the following text should appear (values depend on the board and the chip used):

```
    -- ADC Example xxx --
    -- xxxxxx-xx
    -- Compiled: xxx xx xxxx xx:xx:xx --
    =========================================================
    Menu: press a key to change the configuration.
    ---------------------------------------------------------
    [X] 0: Set ADC trigger mode: Software.
```

```
[ ] 1: Set ADC trigger mode: ADTRG.
[ ] 2: Set ADC trigger mode: Timer TIOA.
[ ] 3: Set ADC trigger mode: PWM Event Line.
[ ] 4: Set ADC trigger mode: Free run mode.
[E] T: Enable/Disable to transfer with PDC.
[D] S: Enable/Disable to use user sequence mode.
[D] P: Enable/Disable ADC power save mode.
[D] G: Enable/Disable to set gain=2 for potentiometer channel.
[D] O: Enable/Disable offset for potentiometer channel.
    Q: Quit configuration and start ADC.
=========================================================
```

The application will send converted values to the serial port and display a menu for users to set the different modes.

10. ADC Threshold Wakeup Example

10.1 Purpose

This example demonstrates how to use ADC with threshold wakeup.

10.2 Requirements

This example can be used with SAM evaluation kits, such as SAM4C-EK, SAM4S-EK and others. Refer to the list of available kits at http://www.atmel.com

Note

ADVREF must be set to 3300mv in order to enable full scale measurement of the potentiometer. Refer to the board schematics for advref jumper configuration.

10.3 Description

This example uses TIOA0 as an external trigger instead of a software trigger of ADC conversion. The TIOA0 is a 1ms period, i.e. 1kHz, square wave. The rising edge during each period triggers the ADC to begin a conversion on the given channel, which is connected to the potentiometer. This example shows a menu as below upon running:

```
-- Menu Choices for this example--
-- 0: Display voltage on potentiometer.--
-- 1: Modify low threshold.--
-- 2: Modify high threshold.--
-- 3: Choose comparison mode.--
-- i: Display ADC information.--
-- m: Display this main menu.--
-- c: Set Auto Calibration Mode. --
-- s: Enter sleep mode.--
```

With the user interface, comparison window and mode can be set. The ADC supports four comparison events as follows:

- Lower than the low threshold
- Higher than the high threshold
- In the comparison window
- Out of the comparison window

If the target recieves an 'S' or 's' from user's input, the core falls into sleep mode thanks to the __WFI.

Changing the position of the potentiometer, and thus the input to the ADC, will bring the voltage within the preset thresholds. This will cause the device to be woken, and ADC sampling to begin.

10.4 Usage

1. Build the program and download it into the evaluation board.
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:
    - 115200 bauds
    - 8 bits of data
    - No parity

- 1 stop bit
- No flow control

3. In the terminal window, the following text should appear (values depend on the board and the chip used):

```
-- ADC Threshold Wakeup Example xxx --
-- xxxxxx-xx
-- Compiled: xxx xx xxxx xx:xx:xx --
-- Menu Choices for this example--
-- 0: Display voltage on potentiometer.--
-- 1: Modify low threshold.--
-- 2: Modify high threshold.--
-- 3: Choose comparison mode.--
-- i: Display ADC information.--
-- m: Display this main menu.--
-- c: Set Auto Calibration Mode. --
-- s: Enter sleep mode.--
```

4. Input the command according to the menu.

Index

E

Enumeration Definitions
- adc_gainvalue_t, 24
- adc_trigger_t, 24

F

Function Definitions
- adc_check, 10
- adc_configure_sequence, 10
- adc_configure_timing, 11
- adc_configure_trigger, 11
- adc_disable_all_channel, 12
- adc_disable_anch, 12
- adc_disable_channel, 12
- adc_disable_channel_differential_input, 13
- adc_disable_channel_input_offset, 13
- adc_disable_interrupt, 13
- adc_disable_tag, 14
- adc_enable_all_channel, 14
- adc_enable_anch, 14
- adc_enable_channel, 14
- adc_enable_channel_differential_input, 15
- adc_enable_channel_input_offset, 15
- adc_enable_interrupt, 15
- adc_enable_tag, 16
- adc_get_actual_adc_clock, 16
- adc_get_channel_status, 16
- adc_get_channel_value, 17
- adc_get_comparison_mode, 17
- adc_get_interrupt_mask, 17
- adc_get_latest_value, 18
- adc_get_overrun_status, 18
- adc_get_pdc_base, 18
- adc_get_status, 19
- adc_get_tag, 19
- adc_get_writeprotect_status, 19
- adc_init, 20
- adc_set_bias_current, 20
- adc_set_channel_input_gain, 21
- adc_set_comparison_channel, 21
- adc_set_comparison_filter, 22
- adc_set_comparison_mode, 22
- adc_set_comparison_window, 22
- adc_set_resolution, 22
- adc_set_writeprotect, 23
- adc_start, 23
- adc_start_sequencer, 23
- adc_stop, 24
- adc_stop_sequencer, 24

M

Macro Definitions
- ADC_FREQ_MAX, 10
- ADC_FREQ_MIN, 10
- ADC_STARTUP_FAST, 10
- ADC_STARTUP_NORM, 10

Document Revision History

Doc. Rev.	Date	Comments
42298A	05/2014	Initial document release

Atmel | Enabling Unlimited Possibilities®

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

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.