SMART ARM-based Microcontrollers

AT03242: SAM D20/D21/D10/D11/DA1/L/C Analog Comparator (AC) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the device's Analog Comparator functionality, for the comparison of analog voltages against a known reference voltage to determine its relative level. The following driver API modes are covered by this manual:

- Polled APIs
- Callback APIs

The following peripherals are used by this module:

- AC (Analog Comparator)

The following devices can use this module:

- Atmel | SMART SAM D20/D21
- Atmel | SMART SAM R21
- Atmel | SMART SAM D10/D11
- Atmel | SMART SAM L21/L22
- Atmel | SMART SAM DA1
- Atmel | SMART SAM C20/C21

The outline of this documentation is as follows:

Table of Contents

1. Software License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.

4. This software may only be redistributed and used in connection with an Atmel microcontroller product.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

The Analog Comparator module provides an interface for the comparison of one or more analog voltage inputs (sourced from external or internal inputs) against a known reference voltage, to determine if the unknown voltage is higher or lower than the reference. Additionally, window functions are provided so that two comparators can be connected together to determine if an input is below, inside, above, or outside the two reference points of the window.

Each comparator requires two analog input voltages, a positive and negative channel input. The result of the comparison is a binary `true` if the comparator's positive channel input is higher than the comparator's negative input channel, and `false` if otherwise.

3.1. Driver Feature Macro Definition

Driver Feature Macro	Supported devices
FEATURE_AC_HYSTERESIS_LEVEL	SAM L21/L22/C20/C21
FEATURE_AC_SYNCBUSY_SCHEME_VERSION_2	SAM L21/L22/C20/C21
FEATURE_AC_RUN_IN_STANDY_EACH_COMPARATOR	SAM L21/L22/C20/C21
FEATURE_AC_RUN_IN_STANDY_PAIR_COMPARATOR	SAM D20/L22/D21/D10/D11/R21/DAx

Note: The specific features are only available in the driver when the selected device supports those features.

3.2. Window Comparators and Comparator Pairs

Each comparator module contains one or more comparator pairs, a set of two distinct comparators which can be used independently or linked together for Window Comparator mode. In this latter mode, the two comparator units in a comparator pair are linked together to allow the module to detect if an input voltage is below, inside, above, or outside a window set by the upper and lower threshold voltages set by the two comparators. If not required, window comparison mode can be turned off and the two comparator units can be configured and used separately.

3.3. Positive and Negative Input MUXes

Each comparator unit requires two input voltages, a positive and a negative channel (note that these names refer to the logical operation that the unit performs, and both voltages should be above GND), which are then compared with one another. Both the positive and the negative channel inputs are connected to a pair of multiplexers (MUXes), which allows one of several possible inputs to be selected for each comparator channel.

The exact channels available for each comparator differ for the positive and the negative inputs, but the same MUX choices are available for all comparator units (i.e. all positive MUXes are identical, all negative MUXes are identical). This allows the user application to select which voltages are compared to one another.

When used in window mode, both comparators in the window pair should have their positive channel input MUXes configured to the same input channel, with the negative channel input MUXes used to set the lower and upper window bounds.

3.4. Output Filtering

The output of each comparator unit can either be used directly with no filtering (giving a lower latency signal, with potentially more noise around the comparison threshold) or be passed through a multiple stage digital majority filter. Several filter lengths are available, with the longer stages producing a more stable result, at the expense of a higher latency.

When output filtering is used in single shot mode, a single trigger of the comparator will automatically perform the required number of samples to produce a correctly filtered result.

3.5. Input Hysteresis

To prevent unwanted noise around the threshold where the comparator unit's positive and negative input channels are close in voltage to one another, an optional hysteresis can be used to widen the point at which the output result flips. This mode will prevent a change in the comparison output unless the inputs cross one another beyond the hysteresis gap introduces by this mode.

3.6. Single Shot and Continuous Sampling Modes

Comparators can be configured to run in either Single Shot or Continuous sampling modes; when in Single Shot mode, the comparator will only perform a comparison (and any resulting filtering, see Output Filtering) when triggered via a software or event trigger. This mode improves the power efficiency of the system by only performing comparisons when actually required by the application.

For systems requiring a lower latency or more frequent comparisons, continuous mode will place the comparator into continuous sampling mode, which increases the module's power consumption, but decreases the latency between each comparison result by automatically performing a comparison on every cycle of the module's clock.

3.7. Events

Each comparator unit is capable of being triggered by both software and hardware triggers. Hardware input events allow for other peripherals to automatically trigger a comparison on demand - for example, a timer output event could be used to trigger comparisons at a desired regular interval.

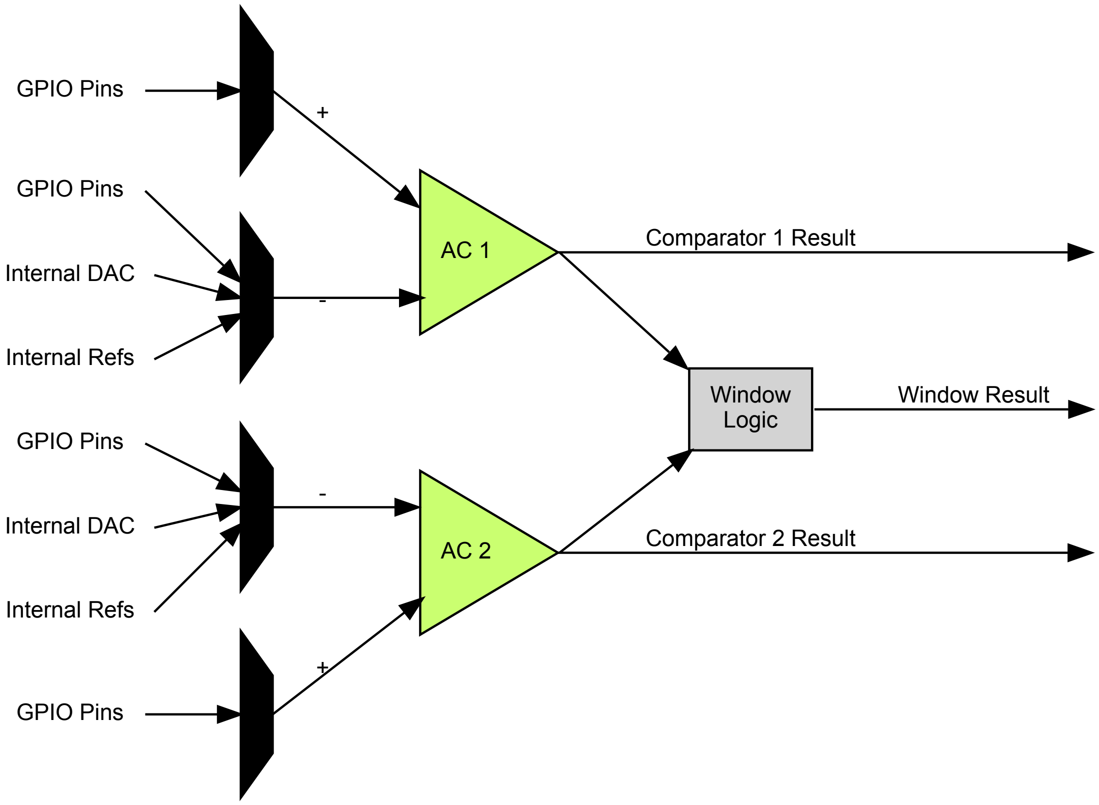
The module's output events can similarly be used to trigger other hardware modules each time a new comparison result is available. This scheme allows for reduced levels of CPU usage in an application and lowers the overall system response latency by directly triggering hardware peripherals from one another without requiring software intervention.

Note: The connection of events between modules requires the use of the SAM Event System Driver (EVENTS) to route output event of one module to the input event of another. For more information on event routing, refer to the event driver documentation.

3.8. Physical Connection

Physically, the modules are interconnected within the device as shown in Figure 3-1 Physical Connection on page 7.

Figure 3-1 Physical Connection



4. Special Considerations

The number of comparator pairs (and, thus, window comparators) within a single hardware instance of the Analog Comparator module is device-specific. Some devices will contain a single comparator pair, while others may have two pairs; refer to your device specific datasheet for details.

5. Extra Information

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

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

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

7. API Overview

7.1. Variable and Type Definitions

7.1.1. Type ac_callback_t

```
typedef void(* ac_callback_t )(struct ac_module *const module_inst)
```

Type definition for a AC module callback function.

7.2. Structure Definitions

7.2.1. Struct ac_chan_config

Configuration structure for a comparator channel, to configure the input and output settings of the comparator.

Table 7-1 Members

Type	Name	Description
bool	enable_hysteresis	When true, hysteresis mode is enabled on the comparator inputs
enum ac_chan_filter	filter	Filtering mode for the comparator output, when the comparator is used in a supported mode
enum ac_hysteresis_level	hysteresis_level	Hysteresis level of the comparator channel
enum ac_chan_interrupt_selection	interrupt_selection	Interrupt criteria for the comparator channel, to select the condition that will trigger a callback
enum ac_chan_neg_mux	negative_input	Input multiplexer selection for the comparator's negative input pin. Any internal reference source, such as a bandgap reference voltage or the DAC, must be configured and enabled prior to its use as a comparator input.
enum ac_chan_output	output_mode	Output mode of the comparator, whether it should be available for internal use, or asynchronously/ synchronously linked to a general-purpose input/ output (GPIO) pin
enum ac_chan_pos_mux	positive_input	Input multiplexer selection for the comparator's positive input pin
bool	run_in_standby	If true, the comparator will continue to sample during sleep mode when triggered

Type	Name	Description
enum ac_chan_sample_mode	sample_mode	Sampling mode of the comparator channel
uint8_t	vcc_scale_factor	Scaled VCC voltage division factor for the channel, when a comparator pin is connected to the V_{CC} voltage scalar input. The formular is: Vscale = Vdd * vcc_scale_factor / 64. If the V_{CC} voltage scalar is not selected as a comparator channel pin's input, this value will be ignored.

7.2.2. Struct ac_config

Configuration structure for a comparator channel, to configure the input and output settings of the comparator.

Table 7-2 Members

Type	Name	Description
enum gclk_generator	source_generator	Source generator for AC GCLK

7.2.3. Struct ac_events

Event flags for the Analog Comparator module. This is used to enable and disable events via ac_enable_events() and ac_disable_events().

Table 7-3 Members

Type	Name	Description
bool	generate_event_on_state[]	If `true`, an event will be generated when a comparator state changes
bool	generate_event_on_window[]	If `true`, an event will be generated when a comparator window state changes
bool	on_event_sample[]	If `true`, a comparator will be sampled each time an event is received

7.2.4. Struct ac_module

AC 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 ac_win_config

Table 7-4 Members

Type	Name	Description
enum ac_win_interrupt_selection	interrupt_selection	Interrupt criteria for the comparator window channel, to select the condition that will trigger a callback

7.3. Macro Definitions

7.3.1. Driver Feature Definition

Define AC driver feature set according to different device family.

7.3.1.1. Macro FEATURE_AC_HYSTERESIS_LEVEL

```
#define FEATURE_AC_HYSTERESIS_LEVEL
```

Setting of hysteresis level

7.3.1.2. Macro FEATURE_AC_SYNCBUSY_SCHEME_VERSION_2

```
#define FEATURE_AC_SYNCBUSY_SCHEME_VERSION_2
```

SYNCBUSY scheme version 2

7.3.1.3. Macro FEATURE_AC_RUN_IN_STANDY_EACH_COMPARATOR

```
#define FEATURE_AC_RUN_IN_STANDY_EACH_COMPARATOR
```

Run in standby feature for each comparator

7.3.2. AC Window Channel Status Flags

AC window channel status flags, returned by ac_win_get_status().

7.3.2.1. Macro AC_WIN_STATUS_UNKNOWN

```
#define AC_WIN_STATUS_UNKNOWN
```

Unknown output state; the comparator window channel was not ready.

7.3.2.2. Macro AC_WIN_STATUS_ABOVE

```
#define AC_WIN_STATUS_ABOVE
```

Window Comparator's input voltage is above the window

7.3.2.3. Macro AC_WIN_STATUS_INSIDE

```
#define AC_WIN_STATUS_INSIDE
```

Window Comparator's input voltage is inside the window

7.3.2.4. Macro AC_WIN_STATUS_BELOW

```
#define AC_WIN_STATUS_BELOW
```

Window Comparator's input voltage is below the window

7.3.2.5. Macro AC_WIN_STATUS_INTERRUPT_SET

```
#define AC_WIN_STATUS_INTERRUPT_SET
```

This state reflects the window interrupt flag. When the interrupt flag should be set is configured in ac_win_set_config(). This state needs to be cleared by the of ac_win_clear_status().

7.3.3. AC Channel Status Flags

AC channel status flags, returned by ac_chan_get_status().

7.3.3.1. Macro AC_CHAN_STATUS_UNKNOWN

```
#define AC_CHAN_STATUS_UNKNOWN
```

Unknown output state; the comparator channel was not ready.

7.3.3.2. Macro AC_CHAN_STATUS_NEG_ABOVE_POS

```
#define AC_CHAN_STATUS_NEG_ABOVE_POS
```

Comparator's negative input pin is higher in voltage than the positive input pin.

7.3.3.3. Macro AC_CHAN_STATUS_POS_ABOVE_NEG

```
#define AC_CHAN_STATUS_POS_ABOVE_NEG
```

Comparator's positive input pin is higher in voltage than the negative input pin.

7.3.3.4. Macro AC_CHAN_STATUS_INTERRUPT_SET

```
#define AC_CHAN_STATUS_INTERRUPT_SET
```

This state reflects the channel interrupt flag. When the interrupt flag should be set is configured in ac_chan_set_config(). This state needs to be cleared by the of ac_chan_clear_status().

7.4. Function Definitions

7.4.1. Configuration and Initialization

7.4.1.1. Function ac_reset()

Resets and disables the Analog Comparator driver.

```
enum status_code ac_reset(
        struct ac_module *const module_inst)
```

Resets and disables the Analog Comparator driver, resets the internal states and registers of the hardware module to their power-on defaults.

Table 7-5 Parameters

Data direction	Parameter name	Description
[out]	module_inst	Pointer to the AC software instance struct

7.4.1.2. Function ac_init()

Initializes and configures the Analog Comparator driver.

```
enum status_code ac_init(
        struct ac_module *const module_inst,
        Ac *const hw,
        struct ac_config *const config)
```

Initializes the Analog Comparator driver, configuring it to the user supplied configuration parameters, ready for use. This function should be called before enabling the Analog Comparator.

Note: Once called the Analog Comparator will not be running; to start the Analog Comparator call ac_enable() after configuring the module.

Table 7-6 Parameters

Data direction	Parameter name	Description
[out]	module_inst	Pointer to the AC software instance struct
[in]	hw	Pointer to the AC module instance
[in]	config	Pointer to the config struct, created by the user application

7.4.1.3. Function ac_is_syncing()

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

```
bool ac_is_syncing(
        struct ac_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 7-7 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Pointer to the AC software instance struct

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

Table 7-8 Return Values

Return value	Description
false	If the module has completed synchronization
ture	If the module synchronization is ongoing

7.4.1.4. Function ac_get_config_defaults()

Initializes all members of an Analog Comparator configuration structure to safe defaults.

```
void ac_get_config_defaults(
        struct ac_config *const config)
```

Initializes all members of a given Analog Comparator configuration structure to safe known default values. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- All comparator pairs disabled during sleep mode (if has this feature)
- Generator 0 is the default GCLK generator

Table 7-9 Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to initialize to default values

7.4.1.5. Function ac_enable()

Enables an Analog Comparator that was previously configured.

```
void ac_enable(
        struct ac_module *const module_inst)
```

Enables an Analog Comparator that was previously configured via a call to ac_init().

Table 7-10 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral

7.4.1.6. Function ac_disable()

Disables an Analog Comparator that was previously enabled.

```
void ac_disable(
        struct ac_module *const module_inst)
```

Disables an Analog Comparator that was previously started via a call to ac_enable().

Table 7-11 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral

7.4.1.7. Function ac_enable_events()

Enables an Analog Comparator event input or output.

```
void ac_enable_events(
        struct ac_module *const module_inst,
        struct ac_events *const events)
```

Enables one or more input or output events to or from the Analog Comparator module. See ac_events for a list of events this module supports.

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

Table 7-12 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	events	Struct containing flags of events to enable

7.4.1.8. Function ac_disable_events()

Disables an Analog Comparator event input or output.

```
void ac_disable_events(
        struct ac_module *const module_inst,
        struct ac_events *const events)
```

Disables one or more input or output events to or from the Analog Comparator module. See ac_events for a list of events this module supports.

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

Table 7-13 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	events	Struct containing flags of events to disable

7.4.2. Channel Configuration and Initialization

7.4.2.1. Function ac_chan_get_config_defaults()

Initializes all members of an Analog Comparator channel configuration structure to safe defaults.

```
void ac_chan_get_config_defaults(
        struct ac_chan_config *const config)
```

Initializes all members of an Analog Comparator channel configuration structure to safe defaults. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- Continuous sampling mode
- Majority of five sample output filter
- Comparator disabled during sleep mode (if has this feature)
- Hysteresis enabled on the input pins
- Hysteresis level of 50mV if having this feature
- Internal comparator output mode
- Comparator pin multiplexer 0 selected as the positive input
- Scaled V_{CC} voltage selected as the negative input

- V_{CC} voltage scaler set for a division factor of two
- Channel interrupt set to occur when the compare threshold is passed

Table 7-14 Parameters

Data direction	Parameter name	Description
[out]	config	Channel configuration structure to initialize to default values

7.4.2.2. Function ac_chan_set_config()

Writes an Analog Comparator channel configuration to the hardware module.

```
enum status_code ac_chan_set_config(
        struct ac_module *const module_inst,
        const enum ac_chan_channel channel,
        struct ac_chan_config *const config)
```

Writes a given Analog Comparator channel configuration to the hardware module.

Table 7-15 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	channel	Analog Comparator channel to configure
[in]	config	Pointer to the channel configuration struct

7.4.2.3. Function ac_chan_enable()

Enables an Analog Comparator channel that was previously configured.

```
void ac_chan_enable(
        struct ac_module *const module_inst,
        const enum ac_chan_channel channel)
```

Enables an Analog Comparator channel that was previously configured via a call to ac_chan_set_config().

Table 7-16 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	channel	Comparator channel to enable

7.4.2.4. Function ac_chan_disable()

Disables an Analog Comparator channel that was previously enabled.

```
void ac_chan_disable(
        struct ac_module *const module_inst,
        const enum ac_chan_channel channel)
```

Stops an Analog Comparator channel that was previously started via a call to ac_chan_enable().

Table 7-17 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	channel	Comparator channel to disable

7.4.3. Channel Control

7.4.3.1. Function ac_chan_trigger_single_shot()

Triggers a comparison on a comparator that is configured in single shot mode.

```
void ac_chan_trigger_single_shot(
        struct ac_module *const module_inst,
        const enum ac_chan_channel channel)
```

Triggers a single conversion on a comparator configured to compare on demand (single shot mode) rather than continuously.

Table 7-18 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	channel	Comparator channel to trigger

7.4.3.2. Function ac_chan_is_ready()

Determines if a given comparator channel is ready for comparisons.

```
bool ac_chan_is_ready(
        struct ac_module *const module_inst,
        const enum ac_chan_channel channel)
```

Checks a comparator channel to see if the comparator is currently ready to begin comparisons.

Table 7-19 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	channel	Comparator channel to test

Returns

Comparator channel readiness state.

7.4.3.3. Function ac_chan_get_status()

Determines the output state of a comparator channel.

```
uint8_t ac_chan_get_status(
        struct ac_module *const module_inst,
        const enum ac_chan_channel channel)
```

Retrieves the last comparison value (after filtering) of a given comparator. If the comparator was not ready at the time of the check, the comparison result will be indicated as being unknown.

Table 7-20 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	channel	Comparator channel to test

Returns
Bit mask of comparator channel status flags.

7.4.3.4. Function ac_chan_clear_status()

Clears an interrupt status flag.

```
void ac_chan_clear_status(
        struct ac_module *const module_inst,
        const enum ac_chan_channel channel)
```

This function is used to clear the AC_CHAN_STATUS_INTERRUPT_SET flag it will clear the flag for the channel indicated by the channel argument.

Table 7-21 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	channel	Comparator channel to clear

7.4.4. Window Mode Configuration and Initialization

7.4.4.1. Function ac_win_get_config_defaults()

Initializes an Analog Comparator window configuration structure to defaults.

```
void ac_win_get_config_defaults(
        struct ac_win_config *const config)
```

Initializes a given Analog Comparator channel configuration structure to a set of known default values. This function should be called if window interrupts are needed and before ac_win_set_config().

The default configuration is as follows:

- Channel interrupt set to occur when the measurement is above the window

Table 7-22 Parameters

Data direction	Parameter name	Description
[out]	config	Window configuration structure to initialize to default values

7.4.4.2. Function ac_win_set_config()

Function used to setup interrupt selection of a window.

```
enum status_code ac_win_set_config(
        struct ac_module *const module_inst,
        enum ac_win_channel const win_channel,
        struct ac_win_config *const config)
```

This function is used to setup when an interrupt should occur for a given window.

Note: This must be done before enabling the channel.

Table 7-23 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Pointer to software instance struct
[in]	win_channel	Window channel to setup
[in]	config	Configuration for the given window channel

Table 7-24 Return Values

Return value	Description
STATUS_OK	Function exited successful
STATUS_ERR_INVALID_ARG	win_channel argument incorrect

7.4.4.3. Function ac_win_enable()

Enables an Analog Comparator window channel that was previously configured.

```
enum status_code ac_win_enable(
        struct ac_module *const module_inst,
        const enum ac_win_channel win_channel)
```

Enables and starts an Analog Comparator window channel.

Note: The comparator channels used by the window channel must be configured and enabled before calling this function. The two comparator channels forming each window comparator pair must have identical configurations other than the negative pin multiplexer setting.

Table 7-25 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	win_channel	Comparator window channel to enable

Returns
Status of the window enable procedure.

Table 7-26 Return Values

Return value	Description
STATUS_OK	The window comparator was enabled
STATUS_ERR_IO	One or both comparators in the window comparator pair is disabled
STATUS_ERR_BAD_FORMAT	The comparator channels in the window pair were not configured correctly

7.4.4.4. Function ac_win_disable()

Disables an Analog Comparator window channel that was previously enabled.

```
void ac_win_disable(
        struct ac_module *const module_inst,
        const enum ac_win_channel win_channel)
```

Stops an Analog Comparator window channel that was previously started via a call to ac_win_enable().

Table 7-27 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	win_channel	Comparator window channel to disable

7.4.5. Window Mode Control

7.4.5.1. Function ac_win_is_ready()

Determines if a given Window Comparator is ready for comparisons.

```
bool ac_win_is_ready(
        struct ac_module *const module_inst,
        const enum ac_win_channel win_channel)
```

Checks a Window Comparator to see if the both comparators used for window detection is currently ready to begin comparisons.

Table 7-28 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	win_channel	Window Comparator channel to test

Returns

Window Comparator channel readiness state.

7.4.5.2. Function ac_win_get_status()

Determines the state of a specified Window Comparator.

```
uint8_t ac_win_get_status(
        struct ac_module *const module_inst,
        const enum ac_win_channel win_channel)
```

Retrieves the current window detection state, indicating what the input signal is currently comparing to relative to the window boundaries.

Table 7-29 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	win_channel	Comparator Window channel to test

Returns
Bit mask of Analog Comparator window channel status flags.

7.4.5.3. Function ac_win_clear_status()

Clears an interrupt status flag.

```
void ac_win_clear_status(
        struct ac_module *const module_inst,
        const enum ac_win_channel win_channel)
```

This function is used to clear the AC_WIN_STATUS_INTERRUPT_SET flag it will clear the flag for the channel indicated by the win_channel argument.

Table 7-30 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Analog Comparator peripheral
[in]	win_channel	Window channel to clear

7.5. Enumeration Definitions

7.5.1. Enum ac_callback

Enum for possible callback types for the AC module.

Table 7-31 Members

Enum value	Description
AC_CALLBACK_COMPARATOR_0	Callback for comparator 0
AC_CALLBACK_COMPARATOR_1	Callback for comparator 1
AC_CALLBACK_WINDOW_0	Callback for window 0

7.5.2. Enum ac_chan_channel

Enum for the possible comparator channels.

Table 7-32 Members

Enum value	Description
AC_CHAN_CHANNEL_0	Comparator channel 0 (Pair 0, Comparator 0)
AC_CHAN_CHANNEL_1	Comparator channel 1 (Pair 0, Comparator 1)
AC_CHAN_CHANNEL_2	Comparator channel 2 (Pair 1, Comparator 0)
AC_CHAN_CHANNEL_3	Comparator channel 3 (Pair 1, Comparator 1)

7.5.3. Enum ac_chan_filter

Enum for the possible channel output filtering configurations of an Analog Comparator channel.

Table 7-33 Members

Enum value	Description
AC_CHAN_FILTER_NONE	No output filtering is performed on the comparator channel
AC_CHAN_FILTER_MAJORITY_3	Comparator channel output is passed through a Majority-of-Three filter
AC_CHAN_FILTER_MAJORITY_5	Comparator channel output is passed through a Majority-of-Five filter

7.5.4. Enum ac_chan_interrupt_selection

This enum is used to select when a channel interrupt should occur.

Table 7-34 Members

Enum value	Description
AC_CHAN_INTERRUPT_SELECTION_TOGGLE	An interrupt will be generated when the comparator level is passed
AC_CHAN_INTERRUPT_SELECTION_RISING	An interrupt will be generated when the measurement goes above the compare level
AC_CHAN_INTERRUPT_SELECTION_FALLING	An interrupt will be generated when the measurement goes below the compare level
AC_CHAN_INTERRUPT_SELECTION_END_OF_COMPARE	An interrupt will be generated when a new measurement is complete. Interrupts will only be generated in single shot mode. This state needs to be cleared by the use of ac_chan_cleare_status()

7.5.5. Enum ac_chan_neg_mux

Enum for the possible channel negative pin input of an Analog Comparator channel.

Table 7-35 Members

Enum value	Description
AC_CHAN_NEG_MUX_PIN0	Negative comparator input is connected to physical AC input pin 0
AC_CHAN_NEG_MUX_PIN1	Negative comparator input is connected to physical AC input pin 1
AC_CHAN_NEG_MUX_PIN2	Negative comparator input is connected to physical AC input pin 2
AC_CHAN_NEG_MUX_PIN3	Negative comparator input is connected to physical AC input pin 3
AC_CHAN_NEG_MUX_GND	Negative comparator input is connected to the internal ground plane
AC_CHAN_NEG_MUX_SCALED_VCC	Negative comparator input is connected to the channel's internal V_{CC} plane voltage scalar
AC_CHAN_NEG_MUX_BANDGAP	Negative comparator input is connected to the internal band gap voltage reference
AC_CHAN_NEG_MUX_DAC0	For SAM D20/D21/D10/D11/R21/DA1: Negative comparator input is connected to the channel's internal DAC channel 0 output. For SAM L21/C20/C21: Negative comparator input is connected to the channel's internal DAC channel 0 output for Comparator 0 or OPAMP output for Comparator 1.

7.5.6. Enum ac_chan_output

Enum for the possible channel GPIO output routing configurations of an Analog Comparator channel.

Table 7-36 Members

Enum value	Description
AC_CHAN_OUTPUT_INTERNAL	Comparator channel output is not routed to a physical GPIO pin, and is used internally only
AC_CHAN_OUTPUT_ASYNCRONOUS	Comparator channel output is routed to its matching physical GPIO pin, via an asynchronous path
AC_CHAN_OUTPUT_SYNCHRONOUS	Comparator channel output is routed to its matching physical GPIO pin, via a synchronous path

7.5.7. Enum ac_chan_pos_mux

Enum for the possible channel positive pin input of an Analog Comparator channel.

Table 7-37 Members

Enum value	Description
AC_CHAN_POS_MUX_PIN0	Positive comparator input is connected to physical AC input pin 0
AC_CHAN_POS_MUX_PIN1	Positive comparator input is connected to physical AC input pin 1
AC_CHAN_POS_MUX_PIN2	Positive comparator input is connected to physical AC input pin 2
AC_CHAN_POS_MUX_PIN3	Positive comparator input is connected to physical AC input pin 3

7.5.8. Enum ac_chan_sample_mode

Enum for the possible channel sampling modes of an Analog Comparator channel.

Table 7-38 Members

Enum value	Description
AC_CHAN_MODE_CONTINUOUS	Continuous sampling mode; when the channel is enabled the comparator output is available for reading at any time
AC_CHAN_MODE_SINGLE_SHOT	Single shot mode; when used the comparator channel must be triggered to perform a comparison before reading the result

7.5.9. Enum ac_hysteresis_level

Enum for possible hysteresis level types for AC module.

Table 7-39 Members

Enum value	Description
AC_HYSTERESIS_LEVEL_50	Hysteresis level of 50mV
AC_HYSTERESIS_LEVEL_70	Hysteresis level of 70mV
AC_HYSTERESIS_LEVEL_90	Hysteresis level of 90mV
AC_HYSTERESIS_LEVEL_110	Hysteresis level of 110mV

7.5.10. Enum ac_win_channel

Enum for the possible window comparator channels.

Table 7-40 Members

Enum value	Description
AC_WIN_CHANNEL_0	Window channel 0 (Pair 0, Comparators 0 and 1)
AC_WIN_CHANNEL_1	Window channel 1 (Pair 1, Comparators 2 and 3)

7.5.11. Enum ac_win_interrupt_selection

This enum is used to select when a window interrupt should occur.

Table 7-41 Members

Enum value	Description
AC_WIN_INTERRUPT_SELECTION_ABOVE	Interrupt is generated when the compare value goes above the window
AC_WIN_INTERRUPT_SELECTION_INSIDE	Interrupt is generated when the compare value goes inside the window
AC_WIN_INTERRUPT_SELECTION_BELOW	Interrupt is generated when the compare value goes below the window
AC_WIN_INTERRUPT_SELECTION_OUTSIDE	Interrupt is generated when the compare value goes outside the window

8. Extra Information for AC Driver

8.1. Acronyms

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

Acronym	Description
AC	Analog Comparator
DAC	Digital-to-Analog Converter
MUX	Multiplexer

8.2. Dependencies

This driver has the following dependencies:

- System Pin Multiplexer Driver

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 AC Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM Analog Comparator (AC) 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 AC - Basic
- Quick Start Guide for AC - Callback

9.1. Quick Start Guide for AC - Basic

In this use case, the Analog Comparator module is configured for:

- Comparator peripheral in manually triggered (e.g. "Single Shot" mode)
- One comparator channel connected to input MUX pin 0 and compared to a scaled $V_{CC}/2$ voltage

This use case sets up the Analog Comparator to compare an input voltage fed into a GPIO pin of the device against a scaled voltage of the microcontroller's V_{CC} power rail. The comparisons are made on-demand in single-shot mode, and the result stored into a local variable which is then output to the board LED to visually show the comparison state.

9.1.1. Setup

9.1.1.1. Prerequisites

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

9.1.1.2. Code

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

```
/* AC module software instance (must not go out of scope while in use) */
static struct ac_module ac_instance;

/* Comparator channel that will be used */
#define AC_COMPARATOR_CHANNEL   AC_CHAN_CHANNEL_0

void configure_ac(void)
{
    /* Create a new configuration structure for the Analog Comparator
settings
     * and fill with the default module settings. */
    struct ac_config config_ac;
    ac_get_config_defaults(&config_ac);

    /* Alter any Analog Comparator configuration settings here if required
*/

    /* Initialize and enable the Analog Comparator with the user settings
*/
    ac_init(&ac_instance, AC, &config_ac);
}

void configure_ac_channel(void)
{
    /* Create a new configuration structure for the Analog Comparator
channel
```

```
     * settings and fill with the default module channel settings. */
    struct ac_chan_config ac_chan_conf;
    ac_chan_get_config_defaults(&ac_chan_conf);

    /* Set the Analog Comparator channel configuration settings */
    ac_chan_conf.sample_mode      = AC_CHAN_MODE_SINGLE_SHOT;
    ac_chan_conf.positive_input   = AC_CHAN_POS_MUX_PIN0;
    ac_chan_conf.negative_input   = AC_CHAN_NEG_MUX_SCALED_VCC;
    ac_chan_conf.vcc_scale_factor = 32;

    /* Set up a pin as an AC channel input */
    struct system_pinmux_config ac0_pin_conf;
    system_pinmux_get_config_defaults(&ac0_pin_conf);
    ac0_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    ac0_pin_conf.mux_position = CONF_AC_MUX;
    system_pinmux_pin_set_config(CONF_AC_PIN, &ac0_pin_conf);

    /* Initialize and enable the Analog Comparator channel with the user
     * settings */
    ac_chan_set_config(&ac_instance, AC_COMPARATOR_CHANNEL, &ac_chan_conf);
    ac_chan_enable(&ac_instance, AC_COMPARATOR_CHANNEL);
}
```

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

```
system_init();
configure_ac();
configure_ac_channel();
ac_enable(&ac_instance);
```

9.1.1.3. Workflow

1. Create an AC device instance struct, which will be associated with an Analog Comparator peripheral hardware instance.

   ```
   static struct ac_module ac_instance;
   ```

   Note: Device instance structures shall never go out of scope when in use.

2. Define a macro to select the comparator channel that will be sampled, for convenience.

   ```
   #define AC_COMPARATOR_CHANNEL   AC_CHAN_CHANNEL_0
   ```

3. Create a new function `configure_ac()`, which will be used to configure the overall Analog Comparator peripheral.

   ```
   void configure_ac(void)
   ```

4. Create an Analog Comparator peripheral configuration structure that will be filled out to set the module configuration.

   ```
   struct ac_config config_ac;
   ```

5. Fill the Analog Comparator peripheral configuration structure with the default module configuration values.

   ```
   ac_get_config_defaults(&config_ac);
   ```

6. Initialize the Analog Comparator peripheral and associate it with the software instance structure that was defined previously.

   ```
   ac_init(&ac_instance, AC, &config_ac);
   ```

7. Create a new function `configure_ac_channel()`, which will be used to configure the overall Analog Comparator peripheral.

   ```
   void configure_ac_channel(void)
   ```

8. Create an Analog Comparator channel configuration structure that will be filled out to set the channel configuration.

   ```
   struct ac_chan_config ac_chan_conf;
   ```

9. Fill the Analog Comparator channel configuration structure with the default channel configuration values.

   ```
   ac_chan_get_config_defaults(&ac_chan_conf);
   ```

10. Alter the channel configuration parameters to set the channel to one-shot mode, with the correct negative and positive MUX selections and the desired voltage scaler.

    ```
    ac_chan_conf.sample_mode      = AC_CHAN_MODE_SINGLE_SHOT;
    ac_chan_conf.positive_input   = AC_CHAN_POS_MUX_PIN0;
    ac_chan_conf.negative_input   = AC_CHAN_NEG_MUX_SCALED_VCC;
    ac_chan_conf.vcc_scale_factor = 32;
    ```

    Note: The voltage scalar formula is documented in description for ac_chan_config::vcc_scale_factor.

11. Configure the physical pin that will be routed to the AC module channel 0.

    ```
    struct system_pinmux_config ac0_pin_conf;
    system_pinmux_get_config_defaults(&ac0_pin_conf);
    ac0_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    ac0_pin_conf.mux_position = CONF_AC_MUX;
    system_pinmux_pin_set_config(CONF_AC_PIN, &ac0_pin_conf);
    ```

12. Initialize the Analog Comparator channel and configure it with the desired settings.

    ```
    ac_chan_set_config(&ac_instance, AC_COMPARATOR_CHANNEL, &ac_chan_conf);
    ```

13. Enable the now initialized Analog Comparator channel.

    ```
    ac_chan_enable(&ac_instance, AC_COMPARATOR_CHANNEL);
    ```

14. Enable the now initialized Analog Comparator peripheral.

    ```
    ac_enable(&ac_instance);
    ```

9.1.2. Implementation

9.1.2.1. Code

Copy-paste the following code to your user application:

```
ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);

uint8_t last_comparison = AC_CHAN_STATUS_UNKNOWN;

while (true) {
    if (ac_chan_is_ready(&ac_instance, AC_COMPARATOR_CHANNEL)) {
        do {
            last_comparison = ac_chan_get_status(&ac_instance,
                    AC_COMPARATOR_CHANNEL);
        } while (last_comparison & AC_CHAN_STATUS_UNKNOWN);

        port_pin_set_output_level(LED_0_PIN,
                (last_comparison & AC_CHAN_STATUS_NEG_ABOVE_POS));
```

```
        ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);
    }
}
```

9.1.2.2. Workflow

1. Trigger the first comparison on the comparator channel.

   ```
   ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);
   ```

2. Create a local variable to maintain the current comparator state. Since no comparison has taken place, it is initialized to AC_CHAN_STATUS_UNKNOWN.

   ```
   uint8_t last_comparison = AC_CHAN_STATUS_UNKNOWN;
   ```

3. Make the application loop infinitely, while performing triggered comparisons.

   ```
   while (true) {
   ```

4. Check if the comparator is ready for the last triggered comparison result to be read.

   ```
   if (ac_chan_is_ready(&ac_instance, AC_COMPARATOR_CHANNEL)) {
   ```

5. Read the comparator output state into the local variable for application use, re-trying until the comparison state is ready.

   ```
   do {
       last_comparison = ac_chan_get_status(&ac_instance,
               AC_COMPARATOR_CHANNEL);
   } while (last_comparison & AC_CHAN_STATUS_UNKNOWN);
   ```

6. Set the board LED state to mirror the last comparison state.

   ```
   port_pin_set_output_level(LED_0_PIN,
           (last_comparison & AC_CHAN_STATUS_NEG_ABOVE_POS));
   ```

7. Trigger the next conversion on the Analog Comparator channel.

   ```
   ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);
   ```

9.2. Quick Start Guide for AC - Callback

In this use case, the Analog Comparator module is configured for:

- Comparator peripheral in manually triggered (e.g. "Single Shot" mode)
- One comparator channel connected to input MUX pin 0 and compared to a scaled V_{CC}/2 voltage

This use case sets up the Analog Comparator to compare an input voltage fed into a GPIO pin of the device against a scaled voltage of the microcontroller's V_{CC} power rail. The comparisons are made on-demand in single-shot mode, and the result stored into a local variable which is then output to the board LED to visually show the comparison state.

9.2.1. Setup

9.2.1.1. Prerequisites

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

9.2.1.2. Code

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

```
/* AC module software instance (must not go out of scope while in use). */
static struct ac_module ac_instance;

/* Comparator channel that will be used. */
#define AC_COMPARATOR_CHANNEL   AC_CHAN_CHANNEL_0

void configure_ac(void)
{
    /* Create a new configuration structure for the Analog Comparator
settings
     * and fill with the default module settings. */
    struct ac_config config_ac;
    ac_get_config_defaults(&config_ac);

    /* Alter any Analog Comparator configuration settings here if
required. */

    /* Initialize and enable the Analog Comparator with the user settings.
*/
    ac_init(&ac_instance, AC, &config_ac);
}

void configure_ac_channel(void)
{
    /* Create a new configuration structure for the Analog Comparator
channel
     * settings and fill with the default module channel settings. */
    struct ac_chan_config config_ac_chan;
    ac_chan_get_config_defaults(&config_ac_chan);

    /* Set the Analog Comparator channel configuration settings. */
    config_ac_chan.sample_mode         = AC_CHAN_MODE_SINGLE_SHOT;
    config_ac_chan.positive_input      = AC_CHAN_POS_MUX_PIN0;
    config_ac_chan.negative_input      = AC_CHAN_NEG_MUX_SCALED_VCC;
    config_ac_chan.vcc_scale_factor    = 32;
    config_ac_chan.interrupt_selection =
AC_CHAN_INTERRUPT_SELECTION_END_OF_COMPARE;

    /* Set up a pin as an AC channel input. */
    struct system_pinmux_config ac0_pin_conf;
    system_pinmux_get_config_defaults(&ac0_pin_conf);
    ac0_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    ac0_pin_conf.mux_position = CONF_AC_MUX;
    system_pinmux_pin_set_config(CONF_AC_PIN, &ac0_pin_conf);

    /* Initialize and enable the Analog Comparator channel with the user
     * settings. */
    ac_chan_set_config(&ac_instance, AC_COMPARATOR_CHANNEL,
&config_ac_chan);
    ac_chan_enable(&ac_instance, AC_COMPARATOR_CHANNEL);
}

void callback_function_ac(struct ac_module *const module_inst)
{
    callback_status = true;
}

void configure_ac_callback(void)
{
    ac_register_callback(&ac_instance, callback_function_ac,
AC_CALLBACK_COMPARATOR_0);
    ac_enable_callback(&ac_instance, AC_CALLBACK_COMPARATOR_0);
}
```

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

```
system_init();
configure_ac();
configure_ac_channel();
configure_ac_callback();

ac_enable(&ac_instance);
```

9.2.1.3. Workflow

1. Create an AC device instance struct, which will be associated with an Analog Comparator peripheral hardware instance.

   ```
   static struct ac_module ac_instance;
   ```

   Note: Device instance structures shall never go out of scope when in use.

2. Define a macro to select the comparator channel that will be sampled, for convenience.

   ```
   #define AC_COMPARATOR_CHANNEL   AC_CHAN_CHANNEL_0
   ```

3. Create a new function `configure_ac()`, which will be used to configure the overall Analog Comparator peripheral.

   ```
   void configure_ac(void)
   {
   ```

4. Create an Analog Comparator peripheral configuration structure that will be filled out to set the module configuration.

   ```
   struct ac_config config_ac;
   ```

5. Fill the Analog Comparator peripheral configuration structure with the default module configuration values.

   ```
   ac_get_config_defaults(&config_ac);
   ```

6. Initialize the Analog Comparator peripheral and associate it with the software instance structure that was defined previously.

   ```
   ac_init(&ac_instance, AC, &config_ac);
   ```

7. Create a new function `configure_ac_channel()`, which will be used to configure the overall Analog Comparator peripheral.

   ```
   void configure_ac_channel(void)
   {
   ```

8. Create an Analog Comparator channel configuration structure that will be filled out to set the channel configuration.

   ```
   struct ac_chan_config config_ac_chan;
   ```

9. Fill the Analog Comparator channel configuration structure with the default channel configuration values.

   ```
   ac_chan_get_config_defaults(&config_ac_chan);
   ```

10. Alter the channel configuration parameters to set the channel to one-shot mode, with the correct negative and positive MUX selections and the desired voltage scaler.
    Note: The voltage scalar formula is documented in description for ac_chan_config::vcc_scale_factor.
11. Select when the interrupt should occur. In this case an interrupt will occur at every finished conversion.

    ```
    config_ac_chan.sample_mode           = AC_CHAN_MODE_SINGLE_SHOT;
    config_ac_chan.positive_input        = AC_CHAN_POS_MUX_PIN0;
    config_ac_chan.negative_input        = AC_CHAN_NEG_MUX_SCALED_VCC;
    config_ac_chan.vcc_scale_factor      = 32;
    config_ac_chan.interrupt_selection =
    AC_CHAN_INTERRUPT_SELECTION_END_OF_COMPARE;
    ```

12. Configure the physical pin that will be routed to the AC module channel 0.

    ```
    struct system_pinmux_config ac0_pin_conf;
    system_pinmux_get_config_defaults(&ac0_pin_conf);
    ac0_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    ac0_pin_conf.mux_position = CONF_AC_MUX;
    system_pinmux_pin_set_config(CONF_AC_PIN, &ac0_pin_conf);
    ```

13. Initialize the Analog Comparator channel and configure it with the desired settings.

    ```
    ac_chan_set_config(&ac_instance, AC_COMPARATOR_CHANNEL,
    &config_ac_chan);
    ```

14. Enable the initialized Analog Comparator channel.

    ```
    ac_chan_enable(&ac_instance, AC_COMPARATOR_CHANNEL);
    ```

15. Create a new callback function.

    ```
    void callback_function_ac(struct ac_module *const module_inst)
    {
        callback_status = true;
    }
    ```

16. Create a callback status software flag.

    ```
    bool volatile callback_status = false;
    ```

17. Let the callback function set the calback_status flag to true.

    ```
    callback_status = true;
    ```

18. Create a new function `configure_ac_callback()`, which will be used to configure the callbacks.

    ```
    void configure_ac_callback(void)
    {
        ac_register_callback(&ac_instance, callback_function_ac,
    AC_CALLBACK_COMPARATOR_0);
        ac_enable_callback(&ac_instance, AC_CALLBACK_COMPARATOR_0);
    }
    ```

19. Register callback function.

    ```
    ac_register_callback(&ac_instance, callback_function_ac,
    AC_CALLBACK_COMPARATOR_0);
    ```

20. Enable the callbacks.

    ```
    ac_enable_callback(&ac_instance, AC_CALLBACK_COMPARATOR_0);
    ```

21. Enable the now initialized Analog Comparator peripheral.

    ```
    ac_enable(&ac_instance);
    ```

    Note: This should not be done until after the AC is setup and ready to be used.

9.2.2. Implementation

9.2.2.1. Code

Copy-paste the following code to your user application:

```
ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);

uint8_t last_comparison = AC_CHAN_STATUS_UNKNOWN;
port_pin_set_output_level(LED_0_PIN, true);
while (true) {
    if (callback_status == true) {
        do
        {
            last_comparison = ac_chan_get_status(&ac_instance,
                    AC_COMPARATOR_CHANNEL);
        } while (last_comparison & AC_CHAN_STATUS_UNKNOWN);
        port_pin_set_output_level(LED_0_PIN,
                (last_comparison & AC_CHAN_STATUS_NEG_ABOVE_POS));
        callback_status = false;
        ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);
    }
}
```

9.2.2.2. Workflow

1. Trigger the first comparison on the comparator channel.

   ```
   ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);
   ```

2. Create a local variable to maintain the current comparator state. Since no comparison has taken place, it is initialized to AC_CHAN_STATUS_UNKNOWN.

   ```
   uint8_t last_comparison = AC_CHAN_STATUS_UNKNOWN;
   ```

3. Make the application loop infinitely, while performing triggered comparisons.

   ```
   while (true) {
   ```

4. Check if a new comparison is complete.

   ```
   if (callback_status == true) {
   ```

5. Check if the comparator is ready for the last triggered comparison result to be read.

   ```
   do
   {
       last_comparison = ac_chan_get_status(&ac_instance,
               AC_COMPARATOR_CHANNEL);
   } while (last_comparison & AC_CHAN_STATUS_UNKNOWN);
   ```

6. Read the comparator output state into the local variable for application use, re-trying until the comparison state is ready.

   ```
   do
   {
       last_comparison = ac_chan_get_status(&ac_instance,
   ```

```
            AC_COMPARATOR_CHANNEL);
} while (last_comparison & AC_CHAN_STATUS_UNKNOWN);
```

7. Set the board LED state to mirror the last comparison state.

```
port_pin_set_output_level(LED_0_PIN,
        (last_comparison & AC_CHAN_STATUS_NEG_ABOVE_POS));
```

8. After the interrupt is handled, set the software callback flag to false.

```
callback_status = false;
```

9. Trigger the next conversion on the Analog Comparator channel.

```
ac_chan_trigger_single_shot(&ac_instance, AC_COMPARATOR_CHANNEL);
```

10. Document Revision History

Doc. Rev.	Date	Comments
42106F	12/2015	Fixed typos and legal disclaimer
42106E	08/2015	Added support for SAM L21, SAM C20/C21, and SAM DA1
42106D	12/2014	Added support for SAM R21 and SAM D10/D11
42106C	01/2014	Added support for SAM D21
42106B	06/2013	Added additional documentation on the event system. Corrected documentation typos.
42106A	06/2013	Initial 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.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.