APPLICATION NOTE

AT08477: SAM4N/G51/G53 Analog-to-Digital Converter

ASF PROGRAMMERS MANUAL

SAM4N/G51/G53 Analog-to-Digital Converter

This document describes the usage of the driver for the ADC module of the SAM4N, G51, and G53 range of microcontrollers.

See also Quickstart guide for ADC driver.

Table of Contents

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, to convert analog voltages to a corresponding digital value. The ADC has up to 12-bit resolution (10 bits on SAM4N), and is capable of converting up to 800k samples per second (ksps).

3. Special Considerations

NONE.

4. Extra Information

4.1 Terms and Definitions

Term	Definition
SCR	Status Clear Register
ADTRG	ADC Trigger

5. Examples

- Simple Example
- ADC Enhanced Resolution Example
- ADC Temperature Sensor Example

6. API Overview

6.1 Variable and Type Definitions

6.1.1 Type adc_callback_t

```
typedef void(* adc_callback_t )(void)
```

6.2 Structure Definitions

6.2.1 Struct adc_config

Table 6-1. Members

Type	Name	Description
uint32_t	adc_clock	ADC Clock
bool	aste	Averaging on Single Trigger Event
uint32_t	mck	Master Clock
enum adc_resolution	resolution	Resolution
enum adc_startup_time	startup_time	Start Up Time
bool	tag	TAG of ADC_LDCR register
uint8_t	tracktim	Tracking Time = (tracktim+1) / ADC clock
uint8_t	transfer	Transfer Period
bool	useq	Use Sequence Enable

6.2.2 Struct adc_temp_sensor_config

ADC Temperature Sensor configuration structure.

Table 6-2. Members

Type	Name	Description
uint16_t	high_threshold	Temperature High Threshold
uint16_t	low_threshold	Temperature Low Threshold
enum adc_temp_cmp_mode	mode	Temperature Comparison Mode
bool	tempon	Temperature Sensor On

6.3 Macro Definitions

6.3.1 Macro ADC_WPMR_WPKEY_PASSWD

```
#define ADC_WPMR_WPKEY_PASSWD
```

Write Protect Key.

6.3.2 Macro SLEEP_MODE_ADC

```
#define SLEEP_MODE_ADC
```

6.3.3 Macro TEMP_SENSOR

```
#define TEMP_SENSOR
```

6.4 Function Definitions

6.4.1 Function adc_average_on_multi_trigger()

Set ADC averaging on several trigger events.

```
void adc_average_on_multi_trigger(
  Adc *const adc)
```

Table 6-3. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

6.4.2 Function adc_average_on_single_trigger()

Set ADC averaging on single trigger event.

```
void adc_average_on_single_trigger(
  Adc *const adc)
```

Table 6-4. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

6.4.3 Function adc_channel_disable()

Disable the specified ADC channel.

```
void adc_channel_disable(
  Adc *const adc,
  const enum adc_channel_num adc_ch)
```

Table 6-5. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

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

6.4.4 Function adc_channel_enable()

Enable the specified ADC channel.

```
void adc_channel_enable(
  Adc *const adc,
  const enum adc_channel_num adc_ch)
```

Table 6-6. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	adc_ch	Adc channel number

6.4.5 Function adc_channel_get_status()

Get the ADC channel status.

```
uint32_t adc_channel_get_status(
  Adc *const adc,
  const enum adc_channel_num adc_ch)
```

Table 6-7. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	adc_ch	ADC channel number

Table 6-8. Return Values

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

6.4.6 Function adc_channel_get_value()

Read the Converted Data of the selected channel.

```
uint32_t adc_channel_get_value(
  Adc *const adc,
  enum adc_channel_num adc_ch)
```

Table 6-9. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

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

Returns ADC converted value of the selected channel.

6.4.7 Function adc_configure_sequence()

Configure conversion sequence.

```
void adc_configure_sequence(
  Adc *const adc,
  const enum adc_channel_num ch_list,
  const uint8_t uc_num)
```

Table 6-10. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	ch_list	Channel sequence list
[in]	uc_num	Number of channels in the list

6.4.8 Function adc_disable()

Disable ADC Module.

```
void adc_disable(void)
```

6.4.9 Function adc_disable_interrupt()

Disable ADC interrupts.

```
void adc_disable_interrupt(
  Adc *const adc,
  enum adc_interrupt_source interrupt_source)
```

Table 6-11. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	interrupt_source	Interrupts to be disabled

6.4.10 Function adc_enable()

Enable ADC Module.

```
void adc_enable(void)
```

6.4.11 Function adc_enable_interrupt()

Enable ADC interrupts.

```
void adc_enable_interrupt(
  Adc *const adc,
  enum adc_interrupt_source interrupt_source)
```

Table 6-12. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	interrupt_source	Interrupts to be enabled

6.4.12 Function adc_get_comparison_mode()

Get comparison mode.

```
enum adc_cmp_mode adc_get_comparison_mode(
  Adc *const adc)
```

Table 6-13. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Table 6-14. Return Values

Return value	Description
Compare	mode value

6.4.13 Function adc_get_config_defaults()

Get the ADC default configurations. SAM4N/G51/G53 Analog-to-Digital Converter.

```
void adc_get_config_defaults(
  struct adc_config *const cfg)
```

Use to initialize the configuration structure to known default values. This function should be called at the start of any ADC initiation.

The default configuration is as follows:

- 10-bit resolution
- ADC clock frequency is 6MHz
- Start Up Time is 64 periods ADC clock
- Tracking Time is 3 periods of ADC clock
- Transfer Period field shall be programmed with the value 2 as stated in the datasheet

- The controller converts channels in a simple numeric order
- Appends the channel number to the conversion result in ADC_LCDR register
- Only a Single Trigger is required to get an averaged value

Table 6-15. Parameters

Data direction	Parameter name	Description
[out]	cfg	Pointer to configuration structure to be initialized

Note The structure adc_config is defined as follows:

```
struct adc_config {
    enum adc_resolution resolution;
    uint32_t mck;
    uint32_t adc_clock;
    enum adc_startup_time startup_time;
    uint8_t tracktim;
    uint8_t transfer;
    bool useq;
    bool tag;
    bool aste;
};
```

6.4.14 Function adc_get_interrupt_mask()

Get ADC interrupt mask.

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

Table 6-16. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Returns The interrupt mask value.

6.4.15 Function adc_get_interrupt_status()

Get ADC interrupt status.

```
uint32_t adc_get_interrupt_status(
  Adc *const adc)
```

Table 6-17. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Returns The interrupt status value.

6.4.16 Function adc_get_latest_chan_num()

Get the Last Converted Channel Number.

```
uint32_t adc_get_latest_chan_num(
  Adc *const adc)
```

Table 6-18. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Returns The number of the channel that performed the most recent conversion.

6.4.17 Function adc_get_latest_value()

Get the Last Data Converted.

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

Table 6-19. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Returns ADC latest converted value.

6.4.18 Function adc_get_overrun_status()

Get ADC overrun error status.

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

Table 6-20. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Returns ADC overrun error status.

6.4.19 Function adc_get_pdc_base()

Get PDC registers base address.

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

Table 6-21. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Returns ADC PDC register base address.

6.4.20 Function adc_get_writeprotect_status()

Indicate write protect status.

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

Table 6-22. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

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

6.4.21 Function adc_init()

Initialize the ADC Module.

```
enum status_code adc_init(
  Adc *const adc,
  struct adc_config *const config)
```

Table 6-23. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	config	Configuration for the ADC

Table 6-24. Return Values

Return value	Description
STATUS_OK	Initialization is finished
STATUS_ERR_BUSY	Initialization failed

6.4.22 Function adc_set_callback()

Set callback for ADC.

```
void adc_set_callback(
  Adc *const adc,
  enum adc_interrupt_source source,
  adc_callback_t callback,
  uint8_t irq_level)
```

Table 6-25. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	source	Interrupt source
[in]	callback	Callback function pointer
[in]	irq_level	Interrupt level

6.4.23 Function adc_set_comparison_mode()

Configure comparison mode.

```
void adc_set_comparison_mode(
  Adc *const adc,
  const enum adc_cmp_mode mode,
  const enum adc_channel_num channel,
  uint8_t cmp_filter)
```

Table 6-26. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	mode	Comparison mode
[in]	channel	Comparison Selected Channel
[in]	cmp_filter	Compare Event Filtering

6.4.24 Function adc_set_comparison_window()

Configure ADC compare window.

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

Table 6-27. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	us_low_threshold	Low threshold of compare window
[in]	us_high_threshold	High threshold of compare window

6.4.25 Function adc_set_power_mode()

Configure ADC power mode.

```
void adc_set_power_mode(
  Adc *const adc,
  const enum adc_power_mode mode)
```

Table 6-28. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	mode	ADC power mode value

6.4.26 Function adc_set_resolution()

Configure conversion resolution.

```
void adc_set_resolution(
  Adc *const adc,
  const enum adc_resolution res)
```

Table 6-29. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	res	Conversion resolution

6.4.27 Function adc_set_trigger()

Configure conversion trigger and free run mode.

```
void adc_set_trigger(
  Adc *const adc,
  const enum adc_trigger trigger)
```

Table 6-30. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	trigger	Conversion trigger

6.4.28 Function adc_set_writeprotect()

Enable or disable write protection of ADC registers.

```
void adc_set_writeprotect(
  Adc *const adc,
```

```
  const bool is_enable)
```

Table 6-31. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	is_enable	1 to enable, 0 to disable

6.4.29 Function adc_start_calibration()

Launch an automatic calibration of the ADC on next sequence.

```
enum status_code adc_start_calibration(
  Adc *const adc)
```

Table 6-32. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

Table 6-33. Return Values

Return value	Description
STATUS_OK	An automatic calibration is launched
STATUS_ERR_BUSY	Automatic calibration can not be launched because the ADC is in freerun mode.

6.4.30 Function adc_start_software_conversion()

Start analog-to-digital conversion.

```
void adc_start_software_conversion(
  Adc *const adc)
```

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

Table 6-34. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC

6.4.31 Function adc_temp_sensor_get_config_defaults()

Get the ADC Temperature Sensor default configurations.

```
void adc_temp_sensor_get_config_defaults(
  struct adc_temp_sensor_config *const cfg)
```

Use to initialize the configuration structure to known default values.

The default configuration is as follows:

- Generates an event when the converted data is in the comparison window
- The window range is 0xFF ~ 0xFFF

Table 6-35. Parameters

Data direction	Parameter name	Description
[in]	cfg	Pointer to temperature sensor configuration structure to be initiated.

6.4.32 Function adc_temp_sensor_set_config()

Configure the ADC temperature sensor.

```
void adc_temp_sensor_set_config(
  Adc *const adc,
  struct adc_temp_sensor_config * config)
```

Table 6-36. Parameters

Data direction	Parameter name	Description
[in]	adc	Base address of the ADC
[in]	config	Configuration for the ADC temperature sensor

6.5 Enumeration Definitions

6.5.1 Enum adc_channel_num

Definitions for ADC channel number

Table 6-37. Members

Enum value	Description
ADC_CHANNEL_0	
ADC_CHANNEL_1	
ADC_CHANNEL_2	
ADC_CHANNEL_3	
ADC_CHANNEL_4	
ADC_CHANNEL_5	
ADC_CHANNEL_6	
ADC_CHANNEL_7	
ADC_TEMPERATURE_SENSOR	
ADC_CHANNEL_ALL	

6.5.2 Enum adc_cmp_mode

Definitions for Comparison Mode

Table 6-38. Members

Enum value	Description
ADC_CMP_MODE_0	
ADC_CMP_MODE_1	
ADC_CMP_MODE_2	
ADC_CMP_MODE_3	

6.5.3 Enum adc_interrupt_source

ADC interrupt source type

Table 6-39. Members

Enum value	Description
ADC_INTERRUPT_EOC_0	
ADC_INTERRUPT_EOC_1	
ADC_INTERRUPT_EOC_2	
ADC_INTERRUPT_EOC_3	
ADC_INTERRUPT_EOC_4	
ADC_INTERRUPT_EOC_5	
ADC_INTERRUPT_EOC_6	
ADC_INTERRUPT_EOC_7	
ADC_INTERRUPT_TEMP_CHANGE	
ADC_INTERRUPT_END_CAL	
ADC_INTERRUPT_DATA_READY	
ADC_INTERRUPT_OVERRUN_ERROR	
ADC_INTERRUPT_COMP_ERROR	
ADC_INTERRUPT_END_RXBUF	
ADC_INTERRUPT_RXBUF_FULL	
ADC_INTERRUPT_ALL	

6.5.4 Enum adc_power_mode

Definitions for ADC power mode

Table 6-40. Members

Enum value	Description
ADC_POWER_MODE_0	
ADC_POWER_MODE_1	

6.5.5 Enum adc_refer_voltage_source

Definitions for Reference Voltage Selection

Table 6-41. Members

Enum value	Description
ADC_REFER_VOL_EXTERNAL	
ADC_REFER_VOL_STUCK_AT_MIN	
ADC_REFER_VOL_VDDANA	
ADC_REFER_VOL_IRVS	

6.5.6 Enum adc_resolution

Definitions for ADC resolution

Table 6-42. Members

Enum value	Description
ADC_8_BITS	
ADC_10_BITS	
ADC_11_BITS	
ADC_12_BITS	

6.5.7 Enum adc_startup_time

Definitions for ADC Start Up Time

Table 6-43. Members

Enum value	Description
ADC_STARTUP_TIME_0	
ADC_STARTUP_TIME_1	
ADC_STARTUP_TIME_2	
ADC_STARTUP_TIME_3	
ADC_STARTUP_TIME_4	
ADC_STARTUP_TIME_5	
ADC_STARTUP_TIME_6	
ADC_STARTUP_TIME_7	
ADC_STARTUP_TIME_8	
ADC_STARTUP_TIME_9	
ADC_STARTUP_TIME_10	
ADC_STARTUP_TIME_11	
ADC_STARTUP_TIME_12	
ADC_STARTUP_TIME_13	
ADC_STARTUP_TIME_14	
ADC_STARTUP_TIME_15	

6.5.8 Enum adc_temp_cmp_mode

Definitions for Temperature Comparison Mode

Table 6-44. Members

Enum value	Description
ADC_TEMP_CMP_MODE_0	
ADC_TEMP_CMP_MODE_1	
ADC_TEMP_CMP_MODE_2	
ADC_TEMP_CMP_MODE_3	

6.5.9 Enum adc_trigger

Definitions for ADC trigger

Table 6-45. Members

Enum value	Description
ADC_TRIG_SW	
ADC_TRIG_EXT	
ADC_TRIG_TIO_CH_0	
ADC_TRIG_TIO_CH_1	
ADC_TRIG_TIO_CH_2	
ADC_TRIG_FREERUN	

7. Quickstart guide for ADC driver

This is the quickstart guide for the "ADC2 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:

- 10-bit resolution
- ADC clock frequency is 6MHz
- Start Up Time is 64 periods ADC clock
- Tracking Time is three periods of ADC clock
- Transfer Period field shall be programmed with the value 2 as stated in the datasheet
- The controller converts channels in a simple numeric order
- Appends the channel number to the conversion result in AFE_LDCR register
- Single Trigger is optional to get an averaged value
- Software triggering of conversions
- Single channel measurement
- ADC_CHANNEL_1 of ADC as 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:

```
adc_enable();

adc_get_config_defaults(&adc_cfg);

adc_init(ADC, &adc_cfg);

adc_set_trigger(ADC, ADC_TRIG_SW);

adc_channel_enable(ADC, ADC_CHANNEL_1);

adc_start_software_conversion(ADC);

while (adc_get_interrupt_status(ADC) & (1 << ADC_CHANNEL_1));

uint32_t result = adc_channel_get_value(ADC, ADC_CHANNEL_1);
```

7.2.2 Workflow

1. Enable ADC Module:

```
adc_enable();
```

2. Get the ADC default configurations:

```
adc_get_config_defaults(&adc_cfg);
```

3. Initialize the ADC Module:

```
adc_init(ADC, &adc_cfg);
```

4. Configure conversion trigger and free run mode:

```
adc_set_trigger(ADC, ADC_TRIG_SW);
```

5. Enable Channel:

```
adc_channel_enable(ADC, ADC_CHANNEL_1);
```

7.3 Usage Steps

7.3.1 Example Code

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

```
adc_start_software_conversion(ADC);

while (adc_get_interrupt_status(ADC) & (1 << ADC_CHANNEL_1));

uint32_t result = adc_channel_get_value(ADC, ADC_CHANNEL_1);
```

7.3.2 Workflow

1. Start ADC conversion on channel:

```
adc_start_software_conversion(ADC);
```

2. Wait for the conversion over:

```
while (adc_get_interrupt_status(ADC) & (1 << ADC_CHANNEL_1));
```

3. Get the conversion result:

```
uint32_t result = adc_channel_get_value(ADC, ADC_CHANNEL_1);
```

8. ADC Enhanced Resolution Example

This example demonstrates how to use the enhanced resolution feature.

8.1 Purpose

To demonstrate how to use the enhanced resolution feature.

8.2 Requirements

This example can be used with SAM evaluation kits, such as SAM4N-XPRO and SAM5G-XPRO. Refer to the list of available kits at http://www.atmel.com

8.3 Description - Enhanced Resolution Example

The ADC normally operates in 8-bit or 10-bit resolution mode. However by interpolating multiple samples 11-bit and 12-bit resolution can be achieved. This is refered to as enhanced resolution mode. For 11-bit mode, four samples are used, which gives an effective sample rate of 1/4 of the actual sample frequency. For 12-bit mode, 16 samples are used, giving an effective sample rate of 1/16 of the actual sample frequency. This arrangement allows conversion speed to be traded for better accuracy.

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 Example xxx --
-- xxxxxx-xx
-- Compiled: xxx xx xxxx xx:xx:xx --
```

As readings are taken the following text is printed:

```
-- Voltage is: xxxxmv
```

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 Temperature Sensor Example

10.1 Purpose

This example demonstrates how to use the temperature sensor feature of the microcontroller.

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

10.3 Description

This example demonstrates how to use the temperature sensor feature of the device.

The temperature sensor provides an output voltage (VT) that is proportional to the absolute temperature (PTAT). The relationship between measured voltage and actual temperature could be found in the Electrical Characteristics section of the datasheet.

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 Temperature Sensor Example xxx --
*      -- xxxxxx-xx
*      -- Compiled: xxx xx xxxx xx:xx:xx --
*
```

4. The application will output current Celsius temperature on the terminal.

Index

Document Revision History

Doc. Rev.	Date	Comments
42300A	05/2014	Initial document release

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