

---

## AT11481: ADC Configurations with Examples

---

### APPLICATION NOTE

### Introduction

---

This application note describes the following features of the Analog to Digital converter available on the Atmel® | SMART SAM D microcontrollers.

1. Differential mode.
2. Hardware Averaging.
3. Oversampling.
4. Window monitor.
5. Gain and Offset Correction.
6. Temperature sensor.

For detailed description of ADC module, refer respective SAM D Complete datasheet. For demonstration purpose a SAM D21 Xplained pro kit is used.

This firmware project associated with this application note is available in the ASF ([Atmel® Software Framework](#)). This firmware project contains the configuration examples for the above features.

## Table of Contents

---

Introduction.....	1
1. ADC Features.....	4
2. Abbreviations.....	5
3. Pre-requisties.....	6
4. Setup.....	7
4.1. Hardware Setup.....	7
4.2. Software Setup.....	7
4.2.1. Terminal Configuration.....	8
5. ADC Module.....	9
5.1. Overview.....	9
5.2. ADC Example Project.....	10
6. ADC Feature Demonstration.....	11
6.1. ADC Driver.....	11
6.1.1. ADC Driver Functions used in this Application Note.....	11
6.1.2. ADC Driver Data Structures used in this Application Note.....	11
7. Differential Mode.....	12
7.1. Differential Mode Configuration.....	12
8. Hardware Averaging.....	14
8.1. Hardware Averaging Mode Configuration.....	14
9. Oversampling and Decimation.....	16
9.1. Oversampling Mode Configuration.....	16
10. Window Monitoring.....	18
10.1. Window Mode Configuration.....	18
11. Gain and Offset Correction using ADC Calibration.....	20
11.1. ADC Calibration Configuration.....	20
12. Temperature Sensor.....	22
12.1. ADC Temperature Sensor Configuration.....	22
13. SAM D21 ADC Usage Notes.....	24
13.1. ADC Input Signal Range.....	24
13.1.1. Example in Differential Mode.....	24
13.2. Maximum Source Impedance.....	25
13.2.1. Source Impedance Calculation.....	25
13.3. Sampling Time.....	26

14. Best Practices to Improve Accuracy.....28

15. Revision History.....29

## 1. ADC Features

- 8-, 10-, or 12-bit resolution
- Up to 350,000 samples per second (350ksps)
- Differential and single-ended inputs
  - Up to 32 analog inputs
  - 25 positive and 10 negative, including internal and external
- Five internal inputs
  - Band gap
  - Temperature sensor
  - DAC
  - Scaled core supply
  - Scaled I/O supply
- 1/2x to 16x gain
- Single, continuous, and pin-scan conversion options
- Windowing monitor with selectable channel
- Conversion range:
  - $V_{REF}$  [1v to VDDANA - 0.6V]
  - $ADCx * GAIN$  [0V to  $-V_{REF}$ ]
- Built-in internal reference and external reference options
  - Four bits for reference selection
- Event-triggered conversion for accurate timing (one event input)
- Optional DMA transfer of conversion result
- Hardware gain and offset compensation
- Averaging and oversampling with decimation to support, up to 16-bit result
- Selectable sampling time

## 2. Abbreviations

ADC	Analog to Digital Converter
DMA	Direct Memory Access
EDBG	Embedded Debugger
SWD	Serial Wire Debug
VDDANA	Analog Supply
VDDIO	I/O Supply
VIN	Input Voltage
VPIN	ADC pin voltage
VREF	ADC reference voltage

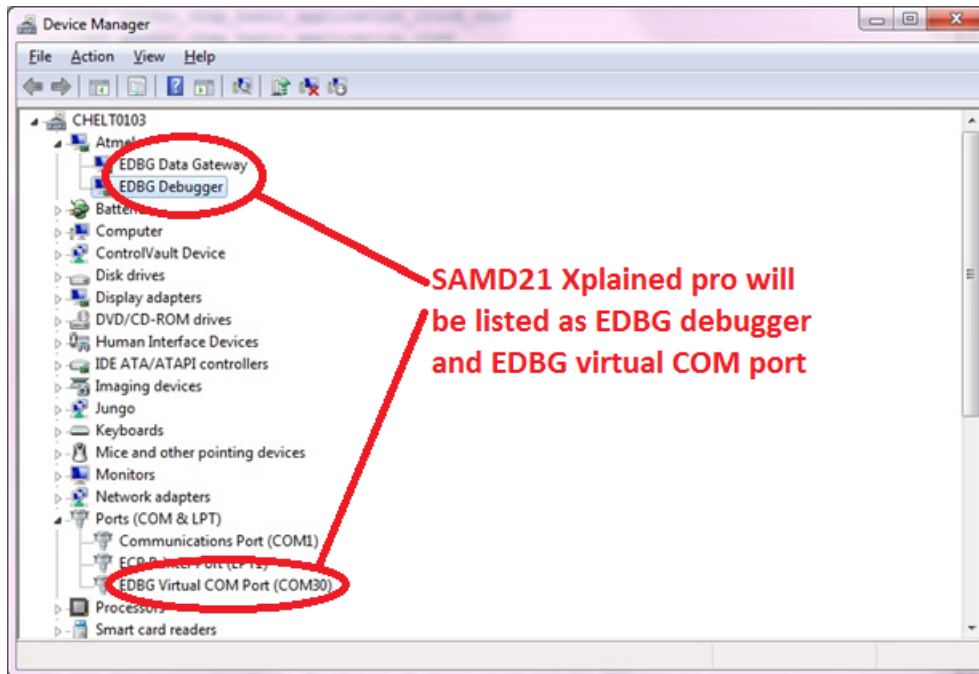
### 3. Pre-requisties

The solutions discussed in this document requires:

- Atmel Studio 6.2 or later
- ASF version 3.27 or later
- Atmel Studio Project “SAMD21\_ADC\_Examples” (Available with ASF 3.27)
- SAM D21 Xplained Pro kit



Figure 4-2. Successful EDBG Driver Installation



The example project in this application note uses the virtual com port functionality offered by the kit to send the results to terminal at 115200 baud rate.

#### 4.2.1. Terminal Configuration

SAM D21 Xplained pro provides virtual COM port functionality. This COM port is used to send the output to terminal window. Any terminal window available for Windows® such as TeraTerm can be used. Hyper Terminal for Windows is used in this application note to display the output.

Table 4-1. Terminal Window Configuration

Parameter	Value
Data Bits	8
Baud	115200
Parity	None
Stop Bits	1
Flow Control	None



## 5. ADC Module

### 5.1. Overview

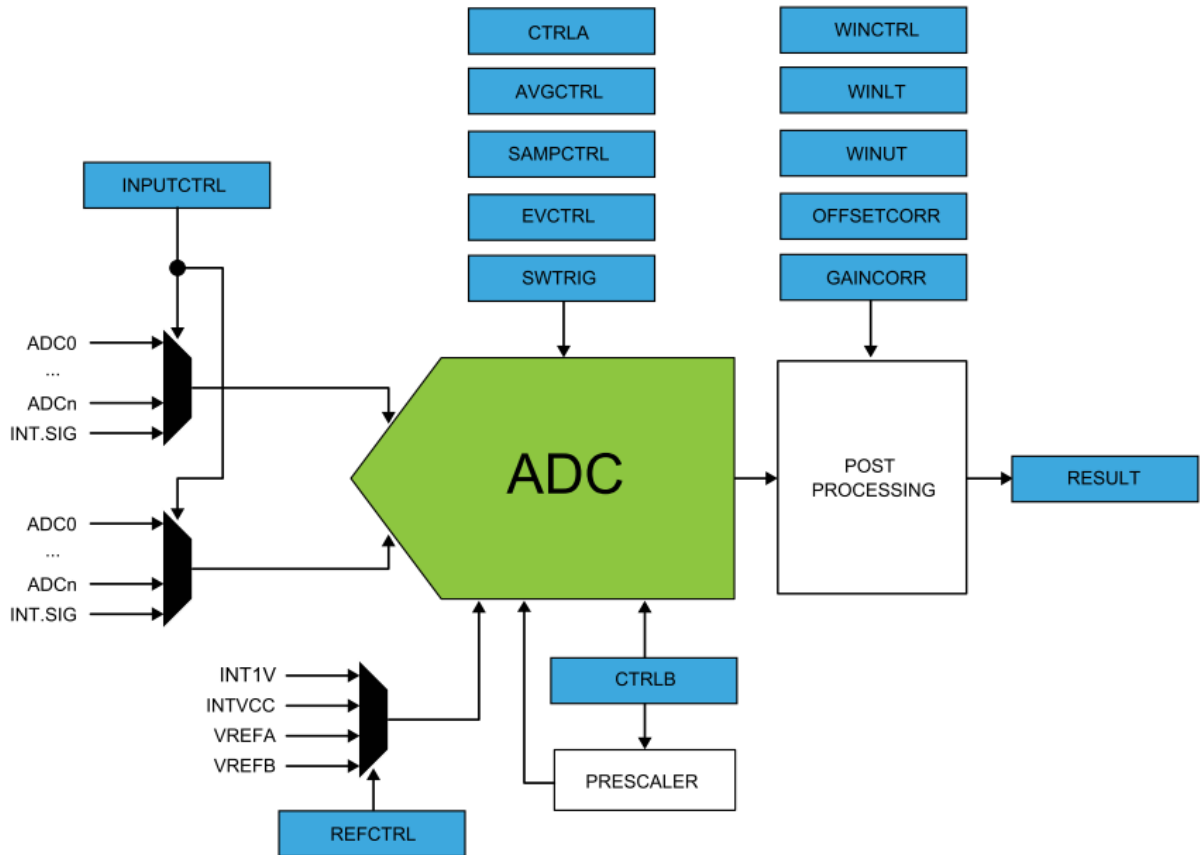
The ADC module has 12-bit resolution, and is capable of converting up to 350kps. The input selection is flexible, and both differential and single-ended measurements can be performed. An optional gain stage is available to increase the dynamic range. In addition, several internal signal inputs are available. The ADC can provide both signed and unsigned results.

ADC measurements can be started by either application software or an incoming event from another peripheral in the device. ADC measurements can be started with predictable timing, and without software intervention. Both internal and external reference voltages can be used.

An integrated temperature sensor is available for use with the ADC. The bandgap voltage as well as the scaled I/O and core voltages can also be measured by the ADC. The ADC has a compare function for accurate monitoring of user-defined thresholds with minimum software intervention.

The ADC may be configured for 8-, 10-, or 12-bit results. ADC conversion results are provided left- or right-adjusted, which eases calculation when the result is represented as a signed value. It is possible to use DMA to move ADC results directly to memory or peripherals when conversions are done.

Figure 5-1. ADC Block Diagram



The ADC has an oversampling with decimation option that can extend the resolution to 16-bits.

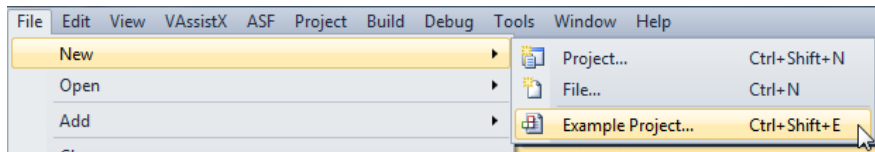
## 5.2. ADC Example Project

The SAMD21\_ADC\_Examples code associated with this application note is available in ASF (Section : Pre-requisites) with Atmel Studio.

To load the SAMD21\_ADC\_Examples code in the Atmel Studio,

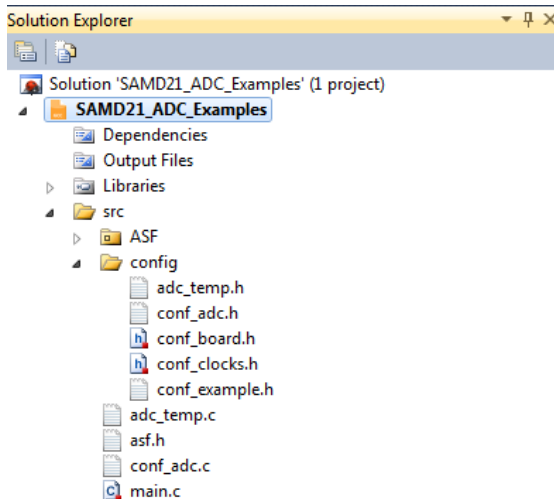
1. Go to **File> New** and click on **Example Project...**. The shortcut key is (CTRL + Shift + E):

**Figure 5-2. Creating Example Project in Atmel Studio**



2. **New Example Project from ASF or Extensions** dialog box will appear. Type **D21** in the search box. It will list the **SAMD21\_ADC\_Examples** project solution available in the ASF:
3. Select the project and click **OK**.
4. After clicking **OK**, the **SAMD21\_ADC\_Examples** project is loaded in the Atmel Studio as shown in following figure .

**Figure 5-3. Solution Explorer View of SAM D21\_ADC\_Examples Project**



5. The `conf_example.h` in **SAMD21\_ADC\_Examples** project contains macros to enable/disable specific ADC configuration. The example code supports only one configuration at a time and it is mandatory to enable only one configuration at a time.

### **Enable/Disable ADC Configurations in `conf_example.h`**

```
#define ENABLE      1
#define DISABLE    0

/* Macro Definitions for ADC Configuration */
/* Please define any one of the below ADC mode */
#define ADC_MODE_DIFFERENTIAL    DISABLE
#define ADC_MODE_HW_AVERAGING    DISABLE
#define ADC_MODE_OVERSAMPLING    DISABLE
#define ADC_MODE_WINDOW          DISABLE
#define ADC_MODE_CALIBRATION     DISABLE
#define ADC_MODE_TEMP_SENSOR     ENABLE
```

6. After enabling the required configuration in `conf_example.h` file, compile the project by selecting **Build solution** option in the **build** menu.

## 6. ADC Feature Demonstration

### 6.1. ADC Driver

This application note uses (ASF) ADC driver's functions and data structures. This driver contains all the essential functions to configure, enable/disable, start, and read the ADC modules.

For more information on SAM D21 ADC driver functions and data structures, refer AT03243: SAM Analog to Digital Converter Driver available at [http://www.atmel.com/images/atmel-42109-sam-analog-to-digital-converter-adc-driver\\_applicationnote\\_at03243.pdf](http://www.atmel.com/images/atmel-42109-sam-analog-to-digital-converter-adc-driver_applicationnote_at03243.pdf).

#### 6.1.1. ADC Driver Functions used in this Application Note

Function name	Description
<code>adc_get_config_defaults</code>	Reads ADC default configuration
<code>adc_init</code>	Generic ADC initialization function used to configure the ADC in required mode of operation by passing appropriate configuration parameters using <code>adc_config</code> data structure
<code>adc_enable</code>	Enables ADC module
<code>adc_disable</code>	Disables ADC module
<code>adc_start_conversion</code>	Starts a new ADC conversion
<code>adc_get_status</code>	Read ADC status
<code>adc_read</code>	Reads ADC results
<code>adc_start_read_result</code>	Starts a new conversion and returns results upon conversion complete

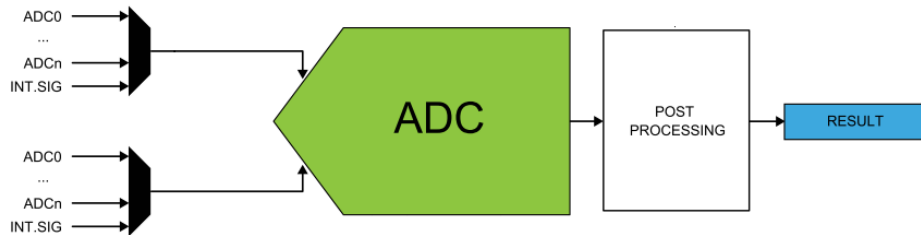
#### 6.1.2. ADC Driver Data Structures used in this Application Note

Data structure name	Description
<code>adc_config</code>	ADC configuration data structure which contains configurable ADC parameters
<code>adc_module</code>	ADC module instance

## 7. Differential Mode

Differential mode of ADC will measure the voltage difference between ADC input pins (the plus and minus input), differential mode should be used when the positive input may follow negative input creating negative results.

Figure 7-1. Differential Mode



### 7.1. Differential Mode Configuration

Differential mode configuration requires setting DIFFMODE bit in ADC's CTRLB register, selecting of positive (PA02) and negative (PA04) inputs for the ADC, selecting voltage reference and clock source for the ADC. The following code example shows all these configuration.

```
void configure_adc_differential(void)
{
    struct adc_config conf_adc;

    adc_get_config_defaults(&conf_adc);

    conf_adc.clock_source = GCLK_GENERATOR_1;
    conf_adc.reference = ADC_REFERENCE_INTVCC1;
    conf_adc.clock_prescaler = ADC_CTRLB_PRESCALER_DIV16;
    conf_adc.differential_mode = true;
    conf_adc.positive_input = ADC_POSITIVE_INPUT_PIN0;
    conf_adc.negative_input = ADC_NEGATIVE_INPUT_PIN4;

    adc_init(&adc_instance, ADC, &conf_adc);

    adc_enable(&adc_instance);
}
```

The following code configures the ADC in differential mode, starts ADC conversion and sends the ADC result to terminal window.

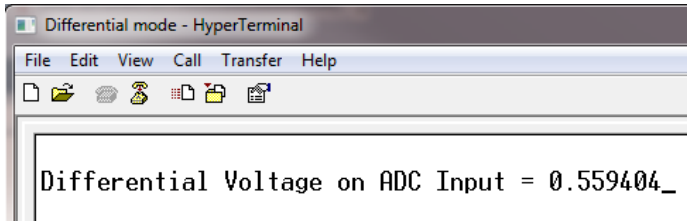
```
void adc_differential(void)
{
    int16_t raw_result_signed;
    configure_adc_differential();
    raw_result = adc_start_read_result();
    raw_result_signed = (int16_t)raw_result;
    result = ((float)raw_result_signed * (float)ADC_REFERENCE_INTVCC1_VALUE) /
    (float)ADC_11BIT_FULL_SCALE_VALUE;

    printf("\r\nDifferential Voltage on ADC Input = %f", result);
    adc_disable(&adc_instance);
}
```

**Table 7-1. Connection Details for Differential Mode Example**

Signal name	Pin	Extension header
Positive Input	Pin3 (PA02 – AIN[0])	EXT3
Negative Input	Pin17 (PA04 – AIN[4])	EXT1

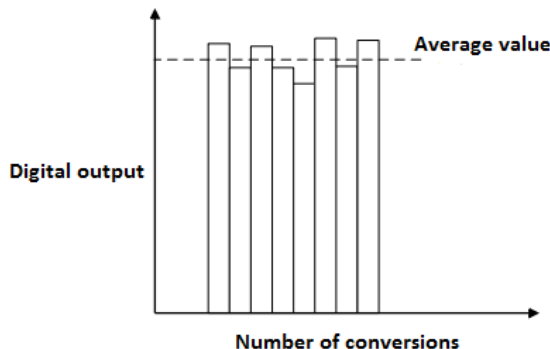
**Figure 7-2. Differential Mode Example Output**



## 8. Hardware Averaging

Averaging is a feature that increases the sample accuracy, at the cost of reduced sample rate. This feature is suitable while operating in noisy conditions. Generally, averaging is performed by accumulating 'm' samples and divide the result by 'm' in the software. The hardware averaging features performs the averaging in hardware without intervention of CPU, reducing the CPU overhead.

Figure 8-1. Hardware Averaging



### 8.1. Hardware Averaging Mode Configuration

Average mode configuration requires setting of sample count and division coefficient in AVGCTRL register, selecting of positive (PA02) and negative (GND) input pins for the ADC, selecting 16-bit Resolution for averaging output, selecting voltage reference and clock source for the ADC.

The following code example performs all these configuration.

```
void configure_adc_averaging(void)
{
    struct adc_config conf_adc;

    adc_get_config_defaults(&conf_adc);

    conf_adc.clock_source = GCLK_GENERATOR_1;
    conf_adc.reference = ADC_REFERENCE_INTVCC1;
    conf_adc.clock_prescaler = ADC_CTRLB_PRESCALER_DIV16;
    conf_adc.positive_input = ADC_POSITIVE_INPUT_PIN0;
    conf_adc.negative_input = ADC_NEGATIVE_INPUT_GND;
    conf_adc.resolution = ADC_RESOLUTION_CUSTOM;
    conf_adc.accumulate_samples = ADC_ACCUMULATE_SAMPLES_1024;
    conf_adc.divide_result = ADC_DIVIDE_RESULT_16;

    adc_init(&adc_instance, ADC, &conf_adc);

    adc_enable(&adc_instance);
}
```

The following code configures the ADC in hardware averaging mode which will accumulate 1024 samples before taking the average, starts ADC conversion and sends the ADC result to terminal window.

```
void adc_hardware_averaging(void)
{
    configure_adc_averaging();

    raw_result = adc_start_read_result();

    result = ((float)raw_result * (float)ADC_REFERENCE_INTVCC1_VALUE) /
    (float)ADC_12BIT_FULL_SCALE_VALUE;

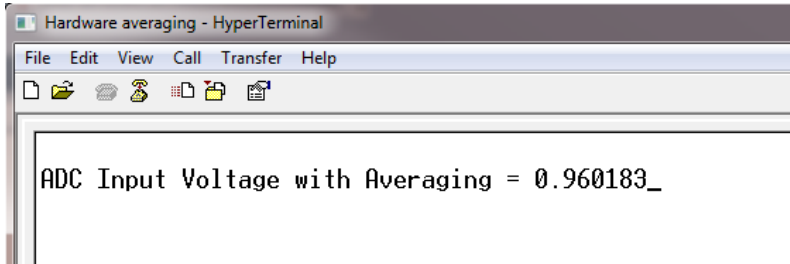
    printf("\n\rADC Input Voltage with Averaging = %f", result);
}
```

```
    adc_disable(&adc_instance);  
}
```

**Table 8-1. Connection Details for Hardware Averaging Mode Example**

Signal name	Pin	Extension header
Positive Input	Pin3 (PA02 – AIN[0])	EXT3
Negative Input	Pin2 (GND)	EXT1

**Figure 8-2. Hardware Averaging Example Output**



ADC input 0 = .96V

## 9. Oversampling and Decimation

The theory behind 'Oversampling' is rather complex, but using the method is fairly easy. The technique requires a higher number of samples. These extra samples can be achieved by oversampling the signal. For each additional bit of resolution 'n', the signal must be oversampled four times. Which frequency to sample the input signal with is given by following equation. To derive the best possible representation of analog input signal, it is necessary to oversample the signal. A larger amount of samples will provide a better representation of the input signal, when averaged.

$$f_{oversample} = 4^n * f_{nyquist}$$

By using oversampling, the ADC resolution can be increased from 12 bits to up to 16 bits. To increase the resolution by n bits, 4 n samples must be accumulated. The result must then be shifted right by n bits. This right shift is a combination of the automatic right shift and the value written to AVGCTRL.ADJRES. To obtain the correct resolution, the ADJRES must be configured as described in the following table. This method results in n bit extra LSB resolution.

**Table 9-1. Configuration Required for Oversampling and Decimation**

Result resolution	Number of samples to average	AVGCTRL.SAMPL ENUM[3:0]	Number of automatic right shifts	AVGCTRL.ADJRES[2:0]
13 bits	4 <sup>1</sup> =4	0x02	0	0x01
14 bits	4 <sup>2</sup> =16	0x04	0	0x02
15 bits	4 <sup>3</sup> =64	0x06	2	0x01
16 bits	4 <sup>4</sup> =256	0x08	4	0x0

### 9.1. Oversampling Mode Configuration

Oversampling mode configuration requires setting of sample count in AVGCTRL register and selecting number of right shifts to adjust resolution to 16-bit in AVGCTRL register, selecting of positive (PA02) and negative (GND) input pins for the ADC, selecting 16-bit resolution for averaging output, selecting voltage reference and clock source for the ADC. The following code performs all the mentioned configuration.

In this mode, ADC is configured to resolution of 16-bits using oversampling to increase the accuracy. To perform the same ADC accumulates 256 samples and then store the right shifted ADC value by 4 automatically into RESULT register.

```
void configure_adc_sampling(void)
{
    struct adc_config conf_adc;

    adc_get_config_defaults(&conf_adc);

    conf_adc.clock_source = GCLK_GENERATOR_1;
    conf_adc.reference = ADC_REFERENCE_INTVCC1;
    conf_adc.clock_prescaler = ADC_CTRLB_PRESCALER_DIV16;
    conf_adc.positive_input = ADC_POSITIVE_INPUT_PIN0;
    conf_adc.negative_input = ADC_NEGATIVE_INPUT_GND;
    conf_adc.resolution = ADC_RESOLUTION_16BIT;
    conf_adc.reference_compensation_enable = true;

    adc_init(&adc_instance, ADC, &conf_adc);

    adc_enable(&adc_instance);
}
```



The following code configures the ADC in Oversampling mode, starts ADC conversion, and sends the ADC result to terminal window.

```
void adc_oversampling(void)
{
    configure_adc();

    raw_result = adc_start_read_result();

    result = ((float)raw_result * (float)ADC_REFERENCE_INTVCC1_VALUE)/
        (float)ADC_12BIT_FULL_SCALE_VALUE;

    printf("\n\rADC Input Voltage before Oversampling = %f", result);

    adc_disable(&adc_instance);

    configure_adc_sampling();

    raw_result = adc_start_read_result();

    result = ((float)raw_result * (float)ADC_REFERENCE_INTVCC1_VALUE)/
        (float)ADC_16BIT_FULL_SCALE_VALUE;

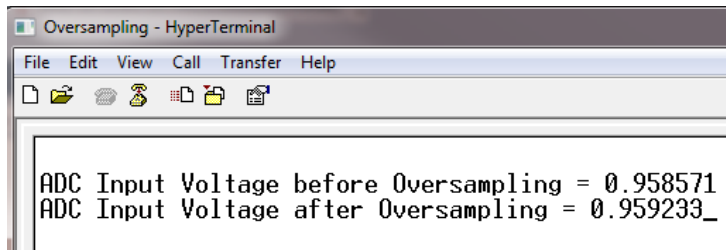
    printf("\n\rADC Input Voltage after Oversampling = %f", result);

    adc_disable(&adc_instance);
}
```

**Table 9-2. Connection Details for Hardware Averaging Mode Example**

Signal name	Pin	Extension header
Positive Input	Pin3 (PA02 – AIN[0])	EXT3
Negative Input	Pin2 (GND)	EXT3

**Figure 9-1. Oversampling Example Output**



ADC input 0 = .96V

## 10. Window Monitoring

The window monitor allows the conversion result to be compared to some predefined threshold limits.

### 10.1. Window Mode Configuration

Supported modes are selected by writing the Window Monitor Mode bit group in the Window Monitor Control register (WINCTRL.WINMODE[2:0]). Thresholds are given by writing the Window Monitor Lower Threshold register (WINLT) and Window Monitor Upper Threshold register (WINUT). If differential input is selected, the WINLT and WINUT are evaluated as signed values. Otherwise, they are evaluated as unsigned values. Another important point is that the significant WINLT and WINUT bits are given by the precision selected in the Conversion Result Resolution bit group in the Control B register (CTRLB.RESSEL). If 8-bit mode is selected, only the eight lower bits is considered. In addition, in differential mode, the eighth bit is considered as the sign bit even if the ninth bit is zero. The INTFLAG.WINMON interrupt flag is set if the conversion result matches the window monitor condition.

```
void configure_adc_window_monitor(void)
{
    struct adc_config conf_adc;

    adc_get_config_defaults(&conf_adc);

    conf_adc.clock_source = GCLK_GENERATOR_1;
    conf_adc.reference = ADC_REFERENCE_INTVCC1;
    conf_adc.clock_prescaler = ADC_CTRLB_PRESCALER_DIV16;
    conf_adc.positive_input = ADC_POSITIVE_INPUT_PIN0;
    conf_adc.negative_input = ADC_NEGATIVE_INPUT_GND;
    conf_adc.window.window_mode = ADC_WINDOW_MODE_ABOVE_LOWER;
    conf_adc.window.window_lower_value = ADC_WINDOW_LOWER_THRESHOLD_VALUE;

    adc_init(&adc_instance, ADC, &conf_adc);

    adc_enable(&adc_instance);
}
```

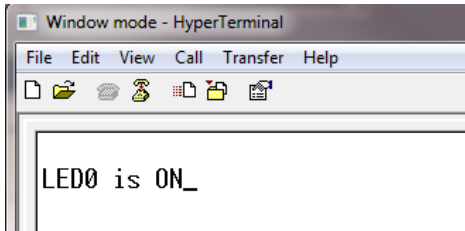
In this mode, ADC is configured in window monitor feature. To trigger the flag if the ADC input voltage is higher than programmed threshold voltage (.75V), more than .75 voltage is applied on ADC input 0, the LED0 available on the SAM D21 Xplained board will glow else it is in OFF condition, this status is sent to console as well.

```
/* This function helps to configure window monitor mode of ADC */
void adc_window_monitor(void)
{
    configure_adc_window_monitor();
    adc_start_conversion(&adc_instance);
    while((adc_get_status(&adc_instance) & ADC_STATUS_RESULT_READY) != 1);
    status = adc_get_status(&adc_instance);
    adc_read(&adc_instance, &raw_result);
    if (status & ADC_STATUS_WINDOW){
        port_pin_set_output_level(LED0_PIN, LOW);
        printf("\n\rLED0 is ON");
    }
    else{
        port_pin_set_output_level(LED0_PIN, HIGH);
        printf("\n\rLED0 is OFF");
    }
}
```

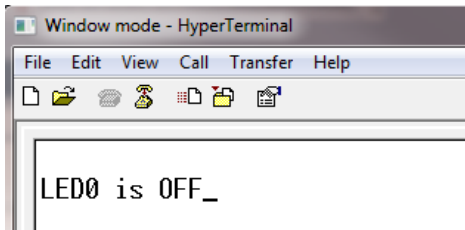
**Table 10-1. Connection Details for Window Monitor Mode Example**

Signal name	Pin	Extension header
Positive Input	Pin3 (PA02 – AIN[0])	EXT3
Negative Input	Pin2 (GND)	EXT3

**Figure 10-1. Window Mode Example Output**



ADC input 0 > .75



ADC input 0 < .75

## 11. Gain and Offset Correction using ADC Calibration

Gain and offset correction feature using ADC calibration improves the accuracy of the ADC results.

### 11.1. ADC Calibration Configuration

Inherent gain and offset errors affect the absolute accuracy of the ADC. The offset error is defined as the deviation of the actual ADC's transfer function from an ideal straight line at zero input voltage. The offset error cancellation is handled by the Offset Correction register (OFFSETCORR). The offset correction value is subtracted from the converted data before writing the Result register (RESULT). The gain error is defined as the deviation of the last output step's midpoint from the ideal straight line, after compensating for offset error. The gain error cancellation is handled by the Gain Correction register (GAINCORR). To correct these two errors, the Digital Correction Logic Enabled bit in the Control B register (CTRLB.CORREN) must be written to one.

The offset and gain error compensation results are both calculated according to:

$$\text{Result} = (\text{Conversion value} - \text{OFFSETCORR}) * \text{GAINCORR}$$

In this mode, ADC is calibrated to correct the Offset and Gain Errors by enabling Digital Correction Logic bit in CTRLB register. In this example, the user has the option to enable/disable the correction through serial console.

The offset calibration is done by applying 0V on ADC input PA02 and the result is offset error which is loaded in the correction register. The gain calibration is done by measuring fixed voltage (1.55V) on ADC input PA02 with 1.65V as ADC reference and load the gain error in the correction register.

```
void adc_calibration(void)
{
    uint8_t key;

    printf("\x0C\n\r-- Start of ADC Calibration Example --\n\r");

    configure_adc();

    printf("Commands:\n\r");
    printf("- key 'c' to enable correction\n\r");
    printf("- key 'd' to disable correction\n\r");

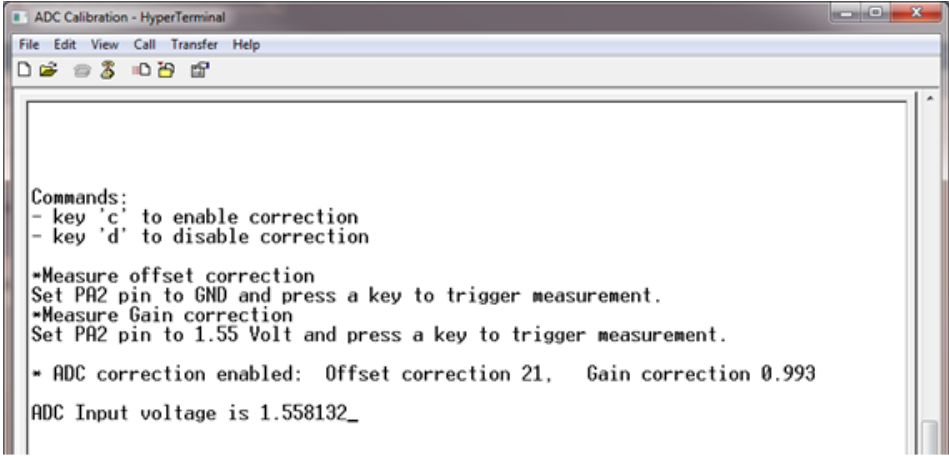
    key = getchar();

    if (key == 'c') {
        adc_correction_start();
    }
    if (key == 'd') {
        adc_correction_stop();
    }

    raw_result = adc_start_read_result();
    result = ((float)raw_result * (float)ADC_REFERENCE_INTVCC1_VALUE) /
    (float)ADC_12BIT_FULL_SCALE_VALUE;

    printf("\n\rADC Input voltage is %f", result);
    printf("\n\r-- End of ADC Calibration Example --\n\r");
}
```

Figure 11-1. ADC Calibration Example Output



```
ADC Calibration - HyperTerminal
File Edit View Call Transfer Help
Commands:
- key 'c' to enable correction
- key 'd' to disable correction

*Measure offset correction
Set PA2 pin to GND and press a key to trigger measurement.
*Measure Gain correction
Set PA2 pin to 1.55 Volt and press a key to trigger measurement.

* ADC correction enabled: Offset correction 21, Gain correction 0.993
ADC Input voltage is 1.558132_
```

## 12. Temperature Sensor

The internal temperature sensor is used to measure the core temperature. The sensor will output a linear voltage proportional to the temperature.

### 12.1. ADC Temperature Sensor Configuration

In this mode, ADC is configured to read internal sensor as positive ADC input and internal ground as negative ADC input. The temperature calculation is performed by reading the Temperature Log Row Content from NVM and current ADC measurement as explained in Equation 1 and Equation 1b in Temperature Sensor Characteristics in the section Analog characteristics of SAM D21 Complete datasheet.

```
void configure_adc_temp(void)
{
    struct adc_config conf_adc;

    adc_get_config_defaults(&conf_adc);

    conf_adc.clock_source = GCLK_GENERATOR_1;
    conf_adc.clock_prescaler = ADC_CLOCK_PRESCALER_DIV16;
    conf_adc.reference = ADC_REFERENCE_INT1V;
    conf_adc.positive_input = ADC_POSITIVE_INPUT_TEMP;
    conf_adc.negative_input = ADC_NEGATIVE_INPUT_GND;
    conf_adc.sample_length = ADC_TEMP_SAMPLE_LENGTH;

    adc_init(&adc_instance, ADC, &conf_adc);

    ADC->AVGCTRL.reg = ADC_AVGCTRL_ADJRES(2) | ADC_AVGCTRL_SAMPLENUM_4;

    adc_enable(&adc_instance);
}

/* This function return the Fine temperature value after done with
calculation using Equation1 and Equation 1b as mentioned in data sheet
on 36.9.8 Temperature Sensor Characteristics */

float calculate_temperature(uint16_t raw_code)
{
    float VADC;      /* Voltage calculation using ADC result for Coarse Temp calculation */
    float VADCM;     /* Voltage calculation using ADC result for Fine Temp calculation. */
    float INT1VM;    /* Voltage calculation for reality INT1V value during the ADC conversion
*/

    VADC = ((float)raw_code * INT1V_VALUE_FLOAT)/ADC_12BIT_FULL_SCALE_VALUE_FLOAT;

    /* Coarse Temp Calculation by assume INT1V=1V for this ADC conversion */
    coarse_temp = tempR + (((tempH - tempR)/(VADCH - VADCR)) * (VADC - VADCR));

    /* Calculation to find the real INT1V value during the ADC conversion */
    INT1VM = INT1VR + (((INT1VH - INT1VR) * (coarse_temp - tempR))/(tempH - tempR));

    VADCM = ((float)raw_code * INT1VM)/ADC_12BIT_FULL_SCALE_VALUE_FLOAT;

    /* Fine Temp Calculation by replace INT1V=1V by INT1V = INT1Vm for ADC conversion */
    fine_temp = tempR + (((tempH - tempR)/(VADCH - VADCR)) * (VADCM - VADCR));

    return fine_temp;
}
```

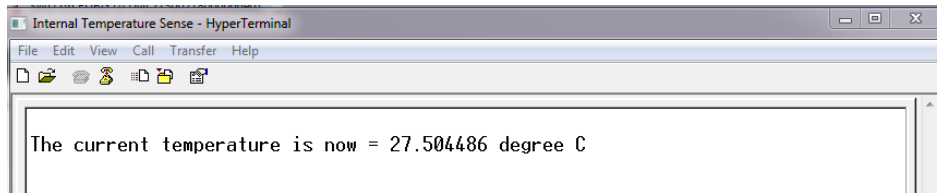
The following example code will read internal temperature and send it to terminal

```
void adc_temp_sensor(void)
{
    float temp;

    system_voltage_reference_enable(SYSTEM_VOLTAGE_REFERENCE_TEMPSENSE);
}
```

```
configure_adc_temp();  
load_calibration_data();  
raw_result = adc_start_read_result();  
temp = calculate_temperature(raw_result);  
printf("\nThe current temperature is = %f degree Celsius", temp);  
}
```

**Figure 12-1. Internal Temperature Sense Example Output**



## 13. SAM D21 ADC Usage Notes

### 13.1. ADC Input Signal Range

This section explains how to calculate the range on ADC input signal due to electrical parameters.

Several parameters restrict the ADC input voltage scale:

- $GND - 0.3V < VPIN < VDD + 0.3V$
- $1V < VREF < VDD - 0.6V$
- Input common mode voltage in single ended mode:
  - $VREF/4 - 0.3 * VDDANA - 0.1V < VCM\_IN < 0.7 * VDDANA + VREF/4 - 0.75V$
- Input common mode voltage in differential mode:
  - If  $|VIN| > VREF/4$  then  $VREF/4 - 0.05 * VDDANA - 0.1V < VCM\_IN < 0.95 * VDDANA + VREF/4 - 0.75V$
  - If  $|VIN| < VREF/4$  then  $0.2 * VDDANA - 0.1V < VCM\_IN < 1.2 * VDDANA - 0.75V$

#### 13.1.1. Example in Differential Mode

Example with  $VREF = 2V$ , gain = x2,  $VDDANA = VDDIO = VDD = 3V$ .

The pin supports without damage a voltage of:  $-0.3V < VPIN < 3.3V$ . But in differential mode, the ADC does not support a negative voltage on input pins. Limits are:  $0 < VPIN < VDD - 0.6V$ .

The input common voltage is:

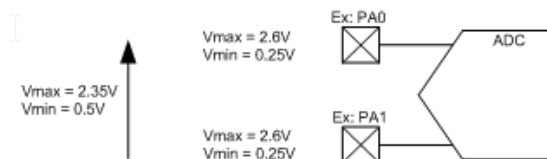
$$0.25V < VCM\_IN < 2.6V \text{ if } |VIN| > VREF/4$$

$$0.5V < VCM\_IN < 2.31V \text{ if } |VIN| < VREF/4$$

It corresponds to the maximum value of  $(VPOS + VNEG)/2$ .

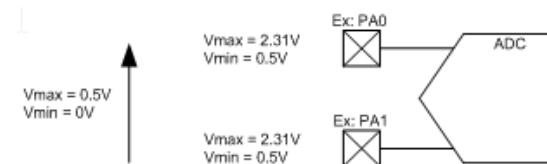
The gain x2 acts on VREF and replaces the voltage reference by  $VREF/2$ . The gain does not have an impact on the input voltage limits.

**Figure 13-1.  $|VIN| > VREF/4$  (0.5V)**



In the preceding figure, when input is greater than  $VREF/4$  (0.5V in preceding example), minimum differential voltage should be  $VREF/4$  (0.5V) and maximum is  $Vmax - Vmin$  ( $2.6V - 0.25V = 2.35V$ )

**Figure 13-2. If  $|VIN| < VREF/4$  (0.5V)**



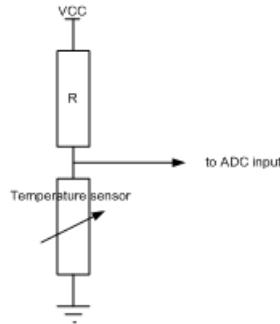
In the preceding figure, when input is lesser than  $VREF/4$ , minimum allowed differential voltage is 0 and maximum is  $VREF/4$ .



## 13.2. Maximum Source Impedance

When a source is connected to the ADC input, it is important to check if the serial equivalent resistance is in accordance with ADC specification. For example, if a temperature sensor is connected to ADC input, the maximum throughput rate is linked to the temperature sensor resistor value.

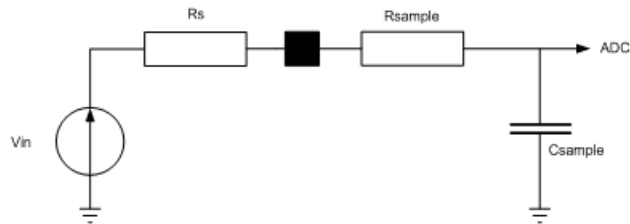
Figure 13-3. Temperature Sensor Resistor ( $R_s = R$ )



### 13.2.1. Source Impedance Calculation

The equivalent circuit from the ADC input side is a generator with a serial output resistor  $R_s$ .

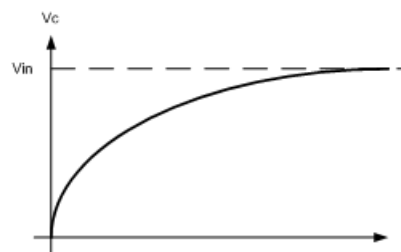
Figure 13-4. The Sample and Hold circuit of the ADC is composed of a RC



The capacitor charge follows the equation:

$$V_c = V_{in} * \left( 1 - \exp \left( \frac{-t}{((R_s + R_{sample}) * C_{sample})} \right) \right)$$

Figure 13-5. VC Charge Characteristics



The minimum sampling and hold time corresponds to the charging time of  $C_{in}$  to reach  $V_{in}$  minus less than one step. This step depends on the ADC accuracy. With 12-bit ADC, a step is  $\left(\frac{V_{in}}{2^{12}}\right)$ . The datasheet formula uses a step of  $\left(\frac{V_{in}}{2^{13}}\right)$  to have more accuracy.

$$V_c > V_{in} - \left(\frac{V_{in}}{2^{13}}\right)$$

$$V_{in} * \left(1 - \exp\left(\frac{-t}{((R_s + R_{sample}) * C_{sample})}\right)\right) > V_{in} - \left(\frac{V_{in}}{2^{13}}\right)$$

$$V_{in} * \left(1 - \exp\left(\frac{-t}{((R_s + R_{sample}) * C_{sample})}\right)\right) > V_{in} * \left(1 - \left(\frac{1}{2^{13}}\right)\right)$$

$$\left(1 - \exp\left(\frac{-t}{((R_s + R_{sample}) * C_{sample})}\right)\right) > 1 - \left(\frac{1}{2^{13}}\right)$$

$$\left(\frac{1}{2^{13}}\right) > \exp\left(\frac{-T_s}{((R_s + R_{sample}) * C_{sample})}\right)$$

$$\left(\frac{T_s}{C_{sample}}\right) > (R_s + R_{sample}) * \ln\left(2^{13}\right) \ggg R_s + R_{sample} < \frac{T_s}{C_{sample} * \ln\left(2^{13}\right)}$$

With:

$R_s$  = external resistor (from application side)

$R_{sample}$  (see datasheet)

$C_{sample}$  (see datasheet)

$T_s$  = sampling time

### 13.3. Sampling Time

The SAM D21 ADC proposes `SAMPLEN[5:0]` bits to adjust the sampling time. The sampling time is equal to  $ADC/2$  frequency by default. `SAMPLEN[5:0]` contains the number of additional half cycles to add.

Using a high frequency for ADC reduces the conversion time. The drawback is reduction of the ADC sampling time. The register allows to increase the sampling time without modifying conversion time

#### **SAMPCTRL register**

**Name:** SAMPCTRL  
**Offset:** 0x03  
**Reset:** 0x00  
**Property:** Write-Protected

Bit	7	6	5	4	3	2	1	0
			SAMPLEN[5:0]					
Access	R	R	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

- Bits 7:6 – Reserved**  
 These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written. These bits will always return zero when read.
- Bits 5:0 – SAMPLEN[5:0]: Sampling Time Length**  
 These bits control the ADC sampling time in number of half CLK<sub>ADC</sub> cycles, depending of the prescaler value, thus controlling the ADC input impedance. Sampling time is set according to the equation:

$$\text{Sampling time} = (\text{SAMPLEN} + 1) \cdot \left( \frac{\text{CLK}_{\text{ADC}}}{2} \right)$$

## 14. Best Practices to Improve Accuracy

The accuracy of ADC depends on the quality of the input signals and power supplies.

The following points should be considered to improve the accuracy of the ADC measurements:

- Understand the ADC, its features, and how they are intended to be used
- Understand the application requirements
- It is important to pay attention while designing the analog signal paths like analog reference and analog power supply. The VDDANA is supply voltage to analog modules such as comparator, ADC.
- Filtering should be used if the analog power supply is connected to digital power supply
- The reference voltage can be more immune to noise by connecting a capacitor between reference pin and ground
- Design shorter analog signal paths whenever possible. It is also important that the impedance on the PCB tracks for the ADC channels are not high which will result in longer charge period of the capacitor for an ADC measurement.
- Make sure analog tracks run over the analog ground plane
- Avoid analog signal path close to a digital signal path with high switching noise such as communication lines and clock signals
- Consider decoupling of the analog signal between signal input and ground for single-ended inputs
- For differential signals the decoupling must be between the positive and negative input. The decoupling capacitor value depends on the input signal. If the signals are switching fast, the decoupling capacitor must be lower.
- Ensure that the source impedance is not very high compared to the sampling rate. If source impedance is too high, the internal sampling capacitor will not be charged to the desired level and the result will not be accurate.
- Try to toggle as few pins as possible while the ADC is converting, to avoid switching noise internally and on the power supply. Especially, the ADC is more sensitive to switching the I/O pins that are powered by the analog power supply.
- Switch off the unused peripherals to eliminate any noise from unused peripherals
- Apply offset and gain calibration to the measurement to improve the accuracy
- Wait until the ADC, reference or sources to stabilize before sampling, as some sources (for example, band gap) need time to stabilize when they are selected as ADC input channel
- Use over-sampling to increase resolution and eliminate random noise
- In free running mode, select the channel before starting the first conversion
- Discard the first conversion result whenever there is a change in ADC configuration like voltage reference / ADC channel change
- When switching to a differential channel (with gain settings), the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. It is better to discard the first sample result.
- The linear interpolation methods such as one point (offset) calibration and two point (offset and gain) calibration method can be used based on the application needs

## 15. Revision History

Doc Rev.	Date	Comments
42645B	08/2016	The first "Window Mode Example Output" image has been corrected
42645A	01/2016	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](http://www.atmel.com)

© 2016 Atmel Corporation. / Rev.: Atmel-42645B-ADC-Configurations-with-Examples\_AT11481\_Application Note-08/2016

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo and others are the registered trademarks or trademarks of ARM Ltd. Windows® is a registered trademark of Microsoft Corporation in U.S. and or other countries. Other terms and product names may be trademarks of others.

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.