Freescale Semiconductor
Application Note

Document Number: AN3031
Rev. 1, 04/2010

Temperature Sensor for the HCS08 Microcontroller Family

by: Donnie Garcia
MCD Applications, Austin, Texas

Rafael Peralez
RTAC Americas, Guadalajara, Mexico

1 Introduction

This document explains how to use the temperature sensor available in the analog-to-digital converter (S08ADCV1) peripheral of the HCS08 family of devices. Devices in this family use the high performance HCS08 core and are ideal for a range of applications such as:

- Small appliances
- Security systems
- Handheld devices
- Control systems

This document provides information on how to perform a basic temperature reading, shows example code that demonstrates a temperature reading, and examines methods for optimizing the accuracy of this temperature sensor.

Contents

freescale™ semiconductor

The outstanding HCS08 ADC peripheral contains numerous features:

- Linear successive approximation algorithm with 10-bit resolution
- Up to 28 analog inputs
- Output formatted in 10- or 8-bit right-justified format
- Single or continuous conversion (automatic return to idle after single conversion)
- Configurable sample time and conversion speed/power
- Conversion complete flag and interrupt
- Input clock selectable from up to four sources
- Operation in wait or stop modes for lower noise operation
- Asynchronous clock source for lower noise operation
- Selectable asynchronous hardware conversion trigger
- Automatic compare with interrupt for less-than, greater-than, or equal-to programmable value

The automatic compare with interrupt and operation in low-power modes features add unique functionality to this ADC peripheral. ADC also contains an on-chip temperature sensor connected to one of the ADC channel inputs. This allows the MCU to monitor the board temperature and take action such as:

- Enter a low power mode to reduce excess battery drain at high temperatures
- Adjust calibration tables for temperature dependant sensors
- Shut down system loading to prevent damage to mechanical components

2 Basic Temperature Sensor Reading

The SO8ADC module has a P-N transistor junction with temperature dependent properties acting as an embedded temperature sensor. The voltage across this junction rises or lowers with temperature allowing silicon to act as a temperature sensor. Figure 1 shows the typical ADC readings of the temperature sensor output across a range of temperatures.

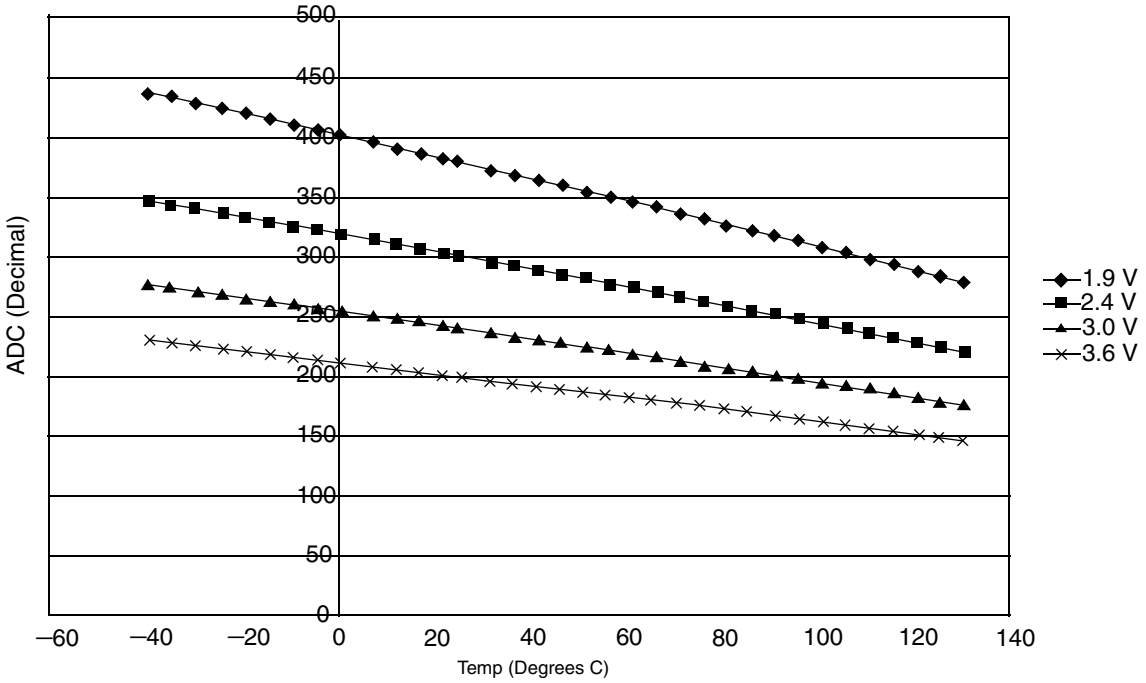


Figure 1. Typical ADC Temperature Readings

The graph shows that the temperature sensor output is linear and dependant on V_{DD}. The temperature sensor output voltage is highest at cold temperatures and lowest at hot temperatures. For V_{DD} = 3 V, the readings range from 277d at –40°C down to 176d at 130°C. An approximate transfer function demonstrated in the following sections represents this behavior.

The temperature sensor reading most accurately represents the temperature of the die and, due to proximity, the leads and the PCB board connected to it. For many applications, the desired temperature may differ from the die's temperature. For example, the die temperature can differ from the temperature of a room's air flow. Using calibration methods can make approximations of this delta.

Some conditions will require special considerations, such as if the PCB board and the parameter that the application measures have different temperatures. These scenarios have many different solutions. Resolve this scenario by thermally connecting the IC to the target with epoxy or placing the IC as near as possible to the monitored element.

Calibration aids in modifying the issue of temperature offset. Correct calibration will ensure that the target element's temperature, as opposed to the die, will correlate to the resulting temperature sensor voltage readings. In this way, you can understand and account for the offset due to location. The following sections further discuss calibration benefits.

2.1 Using Typical Parameters Provided by the Data Sheet

Use the typical parameters provided by the HCS08 data sheet to perform a temperature reading. An approximate transfer function describes the temperature sensor.

$$Temp = 25 - \left(\frac{V_{TEMP} - V_{TEMP25}}{m}\right) \quad \textit{Eqn. 1}$$

Where:

V_{TEMP} is the voltage of the temperature sensor channel at the ambient temperature
V_{TEMP25} is the voltage of the temperature sensor channel at 25°C and V_{DD} = 3 V
m is the hot or cold voltage versus temperature slope in V/°C

The parameter *m* is different for the hot or the cold slope of the equation.

- Hot slope parameter *m* applies to readings greater than 25°C
- Cold slope parameter *m* applies to readings less than 25°C

In the electrical characteristics section of some data sheets, typical parameters are provided for V_{TEMP25} and *m* (both hot and cold slope). These parameters perform a temperature reading according to the flowchart shown in Figure 2. For this example, parameters from the MC9S08QG8 data sheet were used.

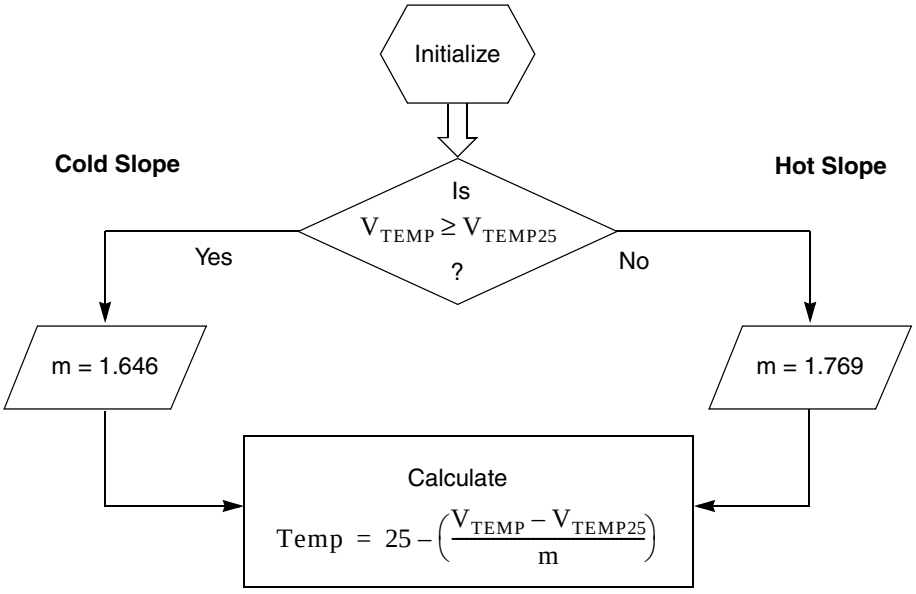


Figure 2. Temperature Reading Flowchart

As shown in this implementation, software initializes the ADC; next, an ADC reading of the temperature sensor is completed. Based on this reading, a decision is made to use the cold or the hot slope parameter. Finally, the calculation is performed.

Floating-point math facilitates the temperature calculations. The software files accompanying this application note show both a floating point and a non-floating point implementation of this flowchart using C code. Both implementations have benefits and drawbacks.

3 Optimizing the Temperature Sensor

You can receive the most accuracy from the temperature sensor in many ways.

1. Analog-to-digital Configuration

   Configure the analog-to-digital for long sample time and a maximum of 1MHz ADC CLK. Use a MCU low power mode to do an analog-to-digital reading. Wait or preferably stop mode reduces the effect of internal MCU noise on the temperature sensor reading.

2. Averaging ADC readings as demonstrated in the Processor Expert example.

   Averaging is the most basic of digital filtering techniques and can reduce the effect system noise on ADC readings. This smooths the temperature sensor input and increases the effective resolution of the analog-to-digital converter.

3. Determine a current reading of V_{DD} by using the bandgap voltage to calculate V_{DD}.

   Using a current value of V_{DD} more accurately represents V_{TEMP25} and V_{TEMP}. This leads to a better result for the approximate transfer function.

4. A floating-point implementation results in more accurate math when using the approximate transfer function if you can spare the code space.

3.1 Calibration

Along with the methods listed, there are other ways to improve accuracy. Ideally, the final application would use the data shown in Figure 1. You could store this data in the MCU as a look-up table. With this method, each ADC reading of the temperature sensor would correlate to a temperature. For example, the data in the graph shows that at 3 V and 0°C the ADC will read 254d. At a constant voltage, you can use the data shown in the graph to get the best temperature accuracy. Unfortunately, this method requires a great deal of effort. You will have to correlate the final application at many temperatures, and this takes time. Also, each of the test points must reside in code space. Although this calibration results in the best results, it is very costly.

A very suitable alternative is to calibrate at a smaller sample of test points. Calibrating at 25°C results in a typical temp sensor Accuracy of ±4.5°C. This method involves determining the V_{TEMP25} parameter from the approximate transfer function (Equation 1) and using this parameter in the temperature calculation. The unit that created the data from Figure 1 V_{TEMP25} (3 V) is 0.703125V. Using this parameter as V_{TEMP25} results in more accurate temperature readings.

Calibrating at three points, –40, 25, and 105°C results in a typical temp sensor accuracy of ±2.5°C. This method requires that you calculate the cold and hot slope (m) for the approximate transfer function. Use the data from Graph 1 to solve for m(hot) and m(cold) as shown here:

$$m(cold) = \frac{-(V_{TEMP40} - V_{TEMP25})}{(-40) - 25} \qquad m(cold) = 0.001668$$

$$m(hot) = \frac{(V_{TEMP105} - V_{TEMP25})}{105 - 25} \qquad m(hot) = 0.001758$$

These calculated parameters differ slightly from the typical parameters provided by the data sheet and result in a more accurate temperature sensor reading.

3.2 Digital Filtering

The handling of the ADC readings generated by the temperature sensor is a very important part of generating an accurate temperature reading. In the calibration methods discussed, the calculated V_{TEMP} readings are dependant on the ADC readings. Along with averaging, you can apply other digital filtering methods to the ADC readings to make improvements to the data sample. Implement a simple software filter to reduce the affect of jumpy ADC readings. Use a digital filter to apply weighting to each of the temp sensor readings. For example, divide each reading by two and add the previous output divided by 2 to make the current output. The result would be a weighted average that places equal weight on the present reading and the contributions of all the previous readings. In pseudo C code, this is implemented using shifts to do the divides: `Output = Current_Reading » 1 + Output » 1`

Sharp change in temperature sensor readings smooths with this method. Using digital filtering reduces the impact of erroneous temperature readings. This is one implementation of a digital filter. In other implementations, you can change the parameters to set the weight of the current reading to a different value.

Additional software could look for erroneous temperature sensor ADC readings. If previous readings differ greatly from the current reading, you could ignore the current reading as bad data.

4 Processor Expert Floating-Point Implementation

This implementation demonstrates a quick and straightforward process to determine a temperature reading. For this example, the DEMOQG8 board and Processor Expert complete a temperature reading. The CodeWarrior project, QG8_Floating-Point_Temp, demonstrates this implementation. When using the method from QG8_Floating-Point_Temp code example, the expected accuracy of the temperature sensor is typically ±8°C. In this example, use Processor Expert to initialize and complete the ADC reading for this project. Figure 3 shows the ADC bean parameters.

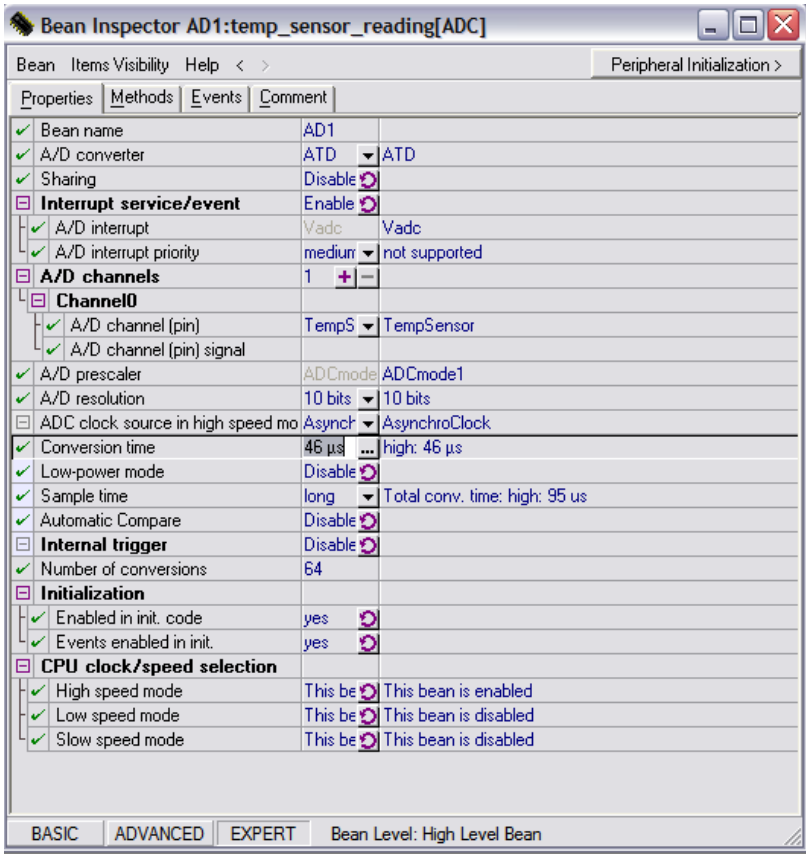


Figure 3. ADC Bean Parameters

The Bean Inspector view displays the ADC configuration necessary to perform a temperature reading. The bean allows the Temp Sensor to be selected as one of the channels. Processor Expert initializes the ADC for long sample time and 10 bits of resolution. The selected conversion time configures an ADC input clock of 1 MHz or lower. As shown in Figure 3, select the asynchronous clock as the source clock input to the ADC and the longest conversion time, 46 µs. This configuration sets an ADCCLK prescaler of 8, creating an ADCCLK below 1 MHz. Later, we will discuss the importance for generating an accurate reading from the temperature sensor with these parameters.

4.1 Averaging ADC Readings

The Processor Expert ADC Bean easily allows for the averaging of analog conversions. As shown in Figure 3, selecting the number of conversions enables averaging. For this example, we set 64 conversions. Averaging is an excellent way to filter some of the errors involved in using the ADC. The averaging of ADC readings can filter noise caused by external system activity. Averaging results in a steady reading that more accurately represents the temperature. The most basic digital filtering done on the temperature sensor readings involves averaging results.

4.2 Using the Approximate Transfer Function

In the Processor Expert Basic_Temp example code, floating-point math is enabled. This allows you to perform the mathematical equations shown in the C code below. This code represents the previous flowchart (see Figure 2).

```
Vtemp = Vtemp * 0.0029296875            ; //Convert the ADC reading into voltage
if (Vtemp => .7012){                    ; //Check for Hot or Cold Slope

Temp1 = 25 - ((Vtemp - .7012)/.001646)  ; //Cold Slope)
}else {

Temp1 = 25 - ((Vtemp-.7012)/.001749)    ; //Hot Slope)
```

In this floating-point implementation, the compiler uses 32-bit IEEE floating-point support selected during the creation of this project. The generated assembly code manages the floating-point math necessary to perform the computations. Files created with the project provide all of the subroutines for floating-point computations. Using the floating-point support provided by CodeWarrior simplifies the use of the approximate transfer function.

5 Fixed-Point Approximation

Use a fixed-point method to reduce code size with a small degradation in accuracy. The following example explains how to use a fixed-point method to create a temperature reading. See the example provided in the Fixed_Point_Basic_Temp folder associated with this application note.

When using fixed-point operations, keep in mind the precision used for all the operations. A number can represent different things for our application; for instance, the number 1000 in our application can mean 100.0, 10.00, 1.000, 0.1000, or even smaller numbers. At every step of our application, we decide the precision used for each number, so we know where the decimal point is for the variables.

Remember the important step of adding comments to each operation to simplify the understanding of what we do, for either us or other people that can use our code. The amount of code size and execution speed saved in these implementations is significant. This allows this implementation to help give a much better performance of a design working with an 8-bit machine.

5.1 Calculating V_{DD}

As described previously, you used this equation to complete a temperature reading:

$$\text{Temp} = 25 - \left(\frac{V_{TEMP} - V_{TEMP25}}{m}\right)$$

The value of V_{TEMP25} given in the data sheet is also the typical value calculated for V_{DD} = 3 V. If the system voltage can vary or it is not set to 3 V, then one important step in calculating temperature is determining the value of V_{DD}. Determine the value of V_{TEMP25} by using the value of V_{DD}.

Knowing that the value of V_{DD} equals the maximum return value for the ATD, with a 10-bit resolution ATD conversion like the one possibly used for the MC9S08QG8, 0x3FF hexadecimal (1023 decimal) will represent the V_{DD}. We can easily determine the value of the supply voltage with the following equations:

$$V_{DD} = 1023 = ADCR_{V_{DD}}$$
$$V_{BG} = 1.2volts = ADCR_{BG} \qquad \textit{Eqn. 2}$$

Where:

- $ADCR_{BG}$ results from the ATD conversion of the Bandgap channel stored in the ADC result register.
- $ADCR_{VDD}$ is the analog to digital conversion of V_{DD}.

Because we can make a conversion for the Bandgap channel and get a value for $ADCR_{BG}$, we can solve with the following expression with a one-variable equation:

$$\frac{V_{DD}}{V_{BG}} = \frac{ADCR_{V_{DD}}}{ADCR_{BG}}$$

$$V_{DD} = \frac{ADCR_{V_{DD}} \times V_{BG}}{ADCR_{BG}} \qquad \textit{Eqn. 3}$$

$$V_{DD} = \frac{1023 \times 1.2volts}{ADCR_{BG}}$$

We can determine the precision needed here because we want a fixed-point representation of the value. The floating-point variables represent a value in a predefined format by an IEEE standard where we have a sign bit, eight bits for the exponent and 23 bits for the fraction, but at the end is a 32 bit (or 64 bit) variable with a special interpretation. The standard was intended to represent values from $\pm 2^{-126}$ to $(2\text{-}2^{-23}) \times 2^{127}$ (approximately from $10^{-44.85}$ to $10^{38.53}$) when using 32-bit format. However, our application has a very limited range of possible values (and many of the applications we see in embedded systems are the same). In cases like this application note, we can choose the way to represent the value in a fixed-point way that helps the MCU to perform all the operations in an easier, smaller, and faster way by having a couple of digits after the decimal point for most of the operations and in each of the operations.

The first approach has a representation of the supply voltage with one value after the decimal point with the intention of adding precision to the final result. The easiest way is to have the original value multiplied times 10. From Equation 3, we can tell that:

$$V_{DD} \times 10 = \frac{(1023 \times 1.2)}{ADCR_{BG}} \times 10$$

The value of 1023 × 1.2 × 10 will always stay fixed; this code's operation will be something like:

$$V_{DD_{CODE}} = \frac{12276}{ADCR_{BG}} \qquad \textit{Eqn. 4}$$

The division then performs with 16-bit fixed-point operations that are much cheaper and faster than floating-point operations. This gives a fixed-point result with one decimal value interpretation and the rest as the integer part. For instance, for a conversion value of $ADCR_{BG}$ = 409 we have the following result:

$$V_{DD} \times 10 = \frac{1023 \times 1.2}{ADCR_{BG}} \times 10 = \frac{12276}{409} = 30$$

$$V_{DD} = \frac{30.0}{10} = 3.00 \text{ volts}$$

In the code, V_{DD} will have a calculated value of 30, and we know that the interpretation for that value is one integer and one decimal value (3.0 in our case). The same kind of equations can lead to calculate a representation of in an equivalent ADC conversion value and use it directly in the original equation. It is possible to make the conversion in the other way where the result from the Temperature Channel would convert to a voltage value and then perform the operation. They have similar steps in either case. At the beginning of the execution, we can calculate all the values needed for the conversion. In addition, every time a new conversion finishes after this, it will perform the conversion to a temperature value faster. In the software files, the equations meant to determine the ATD conversion value for 0.7012 volts (which is the typical V_{TEMP25}). To simplify the writing of further expressions we will use:

$$V_{DD_{CONV}} = 10 \times V_{DD}$$

Because the example uses everything in values equivalent to ATD conversion values, we have to find a representation of V_{TEMP25}. Knowing the value of V_{DD} from the previous equations, we can tell that:

$$\frac{V_{DD}}{V_{TEMP25}} = \frac{ADCR_{V_{DD}}}{ADCR_{TEMP25}}$$

$$ADCR_{TEMP25} = \frac{ADCR_{V_{DD}} \times V_{TEMP25}}{V_{DD}}$$

Replacing V_{DD} with ($10 \times V_{DD}$, the value in our code) and replacing with the typical values, we have the following equation:

$$ADCR_{TEMP25} = \frac{ADCR_{V_{DD}} \times V_{TEMP25} \times 10}{V_{DD_{CONV}}}$$

$$ADCR_{TEMP25} = \frac{1023 \times 0.7012 \times 10}{V_{DD_{CONV}}}$$

Again, the divided value is fixed and the code with the result of the operation (approximately 7173) replaces it. For the previous example where V_{DD} = 3 V and replacing V_{DD} with V_{DDCONV} = 30, we have:

$$ADCR_{TEMP25} = \frac{7173}{V_{DD_{CONV}}}$$

$$ADCR_{TEMP25} = \frac{7173}{30} \qquad \textit{Eqn. 5}$$

$$ADCR_{TEMP25} \approx 239$$

The equivalency between V_{TEMP25} and its conversion to a digital value as $ADCR_{TEMP25}$ helps make a direct subtraction between the result of the Temperature Sensor channel conversion and $ADCR_{TEMP25}$. From the original equation, we need a useful value for m to perform the operations directly with fixed-point. Because of the precision used for each of the operations and the small value of m, (either 1.769 or 1.646 millivolts) we need to have a big multiplier.

If we perform the operations without any multiplier, we will have:

$$\frac{V_{DD}}{m} = \frac{ADCR_{V_{DD}}}{ADCR_m}$$

$$ADCR_m = \frac{ADCR_{V_{DD}} \times m}{V_{DD}} = \frac{ADCR_{V_{DD}} \times m \times 10}{V_{DD_{CONV}}} \qquad \textit{Eqn. 6}$$

$$ADCR_m = \frac{1023 \times 0.001646 \times 10}{V_{DD_{CONV}}}$$

However, if we keep these values, we will see a smaller result of the multiplication than the minor value allowed for VDD. This means that a fixed-point approximation is impossible if we do not multiply for a fixed value to make the previous division possible. Because of the small value, we multiply everything by 100(or 10 or 1,000 because the idea is to use a value that helps to improve the precision with 16-bit operations). The result will be:

$$ADCR_{100m} = \frac{ADCR_{V_{DD}} \times m \times 100}{V_{DD}} = \frac{ADCR_{V_{DD}} \times m \times 10 \times 100}{V_{DD_{CONV}}}$$

$$ADCR_{100m} = \frac{1023 \times 0.001646 \times 10 \times 100}{V_{DD_{CONV}}} \approx \frac{1684}{V_{DD_{CONV}}} \leftrightarrow -40°C < Temp < 25°C \qquad \textit{Eqn. 7}$$

$$ADCR_{100m} = \frac{1023 \times 0.001769 \times 10 \times 100}{V_{DD_{CONV}}} \approx \frac{1810}{V_{DD_{CONV}}} \leftrightarrow 25°C < Temp < 85°C$$

After this point, we have a representation of each of the values needed for the conversion. In order to know where the decimal point for our operations lies, recall if those values were previously multiplied:

$$Temp = 25 - \frac{Temp - Temp25}{m}$$

$$Temp = 25 - \frac{ADCR_T - ADCR_{TEMP25}}{ADCR_m}$$

Where $ADCR_T$ = analog to digital conversion of the Temperature sensor channel

If we replace $ADCR_m$ with $ADCR_{100m}$ ($100 \times ADCR_m$), we have to multiply all the expression times 100 to keep the same units. If we do that, we will have the following result:

$$Temp = 25 - \left(\frac{ADCR_T - ADCR_{TEMP25}}{ADCR_{100m}}\right) \times 100 \qquad \textit{Eqn. 8}$$

Here is a full example using all the previous equations in the way possibly written into the code. The only value we have is the Bandgap conversion which is 380. Calculate the value of the supply voltage as the first step. Knowing that the Bandgap voltage is typically 1.2 volts, we can say that:

From Equation 4, we have;

$CalcV_{DD}$ = 12276/ADCR;

$CalcV_{DD}$ = 32; /* we know that this means 3.2 volts */

After this, we have to determine the value of $ADCR_{Temp25}$. From Equation 5, we have that

$ADCR_{Temp25}$ = 7173/$CalcV_{DD}$;

$ADCR_{Temp25}$ = 224/* this value is the direct equivalency to its ATD conversion in our system*/

As the final step, calculate the two possible used temperature slopes. From Equation 7,

TempSlopeCold = 1684/$CalcV_{DD}$;

TempSlopeCold = 52/* this value is times 100*/

TempSlopeHigh = 1810/$CalcV_{DD}$;

TempSlopeHigh = 56/* this value is times 100 */

Calculate all these values at the start of the code. After this, we only need to make the conversion of the Temperature Sensor channel in the MC9S08QG8 and then apply it to the formula. For two values in the conversion, ADCR = 235 and ADCR = 210 we will have the following results:

	Fixed-point Equation	Floating-point Equation
Example 1: ADCR = 235	$Temp = 25 - \frac{(235 - 224) \times 100}{52}$ $Temp = 25 - 20$ $Temp = 5°C$	$V_{TEMP} = 235 \times 0.003125 = 0.7343$ $Temp = 25 - \frac{0.7343 - 0.7012}{0.001646}$ $Temp = 25 - 20.155 = 4.845°C$
Example 2: ADCR = 210	$Temp = 25 - \frac{(210 - 224) \times 100}{56}$ $Temp = 25 - (-25)$ $Temp = 50°C$	$V_{TEMP} = 210 \times 0.003125 = 0.6562$ $Temp = 25 - \frac{0.6562 - 0.7012}{0.001769}$ $Temp = 25 - (-25.4) = 50.4°C$

In this example, V_{DD} = 3.2 volts for the V_{TEMP} conversion

In this example, we see that the result with fixed-point and floating-point is almost the same. Because we determine the precision in each operation, we can make it with determine positions for each operation and can establish the accuracy of the result. The good thing about this implementation is the smaller code size and a faster execution time because everything uses fixed-point and has 16-bit operations.

The bandgap channel (channel 27 of the ADC peripheral in the MC9S08QG8) has a typical value of 1.2 V. We can start an ADC conversion for the bandgap channel and use the result (ADCR) in Equation 9 to determine the value of V_{DD}.

$$V_{DD} = \frac{1023 \times 1.2 \text{ Volts}}{ADCR} \qquad \textit{Eqn. 9}$$

5.2 Fixed-Point Calculations

To use fixed point, multiply V_{DD} (calculated in Equation 9) by 10. Knowing the supply voltage value ($CalcV_{DD}$), make an equivalency of the value of V_{TEMP25} ($V_{ADCTEMP25}$). Using this method, you can also calculate the value to use for slope (m). See Equation 10.

$$\text{CalcVDD} = V_{DD} \times 10 \qquad \textit{Eqn. 10}$$

$V_{TEMP25} = 0.7012$ Volts

$$V_{ADCTEMP25} = \frac{(1023 \times .7012 \text{ Volts})}{\text{VDD}} = \frac{(1023 \times 7.012 \text{ Volts})}{\text{CalcVDD}} \approx \frac{7173}{\text{CalcVDD}}$$

$$m_{ADC} = \frac{(1000 \times m) \times 1023}{\text{CalCVDD}}$$

$$m_{ADC} = \frac{1.769 \times 1.023}{\text{CalcVDD}} = \frac{1809}{\text{CalcVDD}} \leftrightarrow \text{Temp} \geq 25°C$$

$$m_{ADC} = \frac{1.646 \times 1.023}{\text{CalcVDD}} = \frac{1685}{\text{CalcVDD}} \leftrightarrow \text{Temp} < 25°C$$

After calculating the V_{TEMP25} value in an ADC conversion ($V_{ADCTEMP25}$), start a conversion in the temperature sensor channel and use Equation 11 to approximate the temperature. Use fixed-point values to perform all the needed operations using the result of the ADC conversion (ADCR).

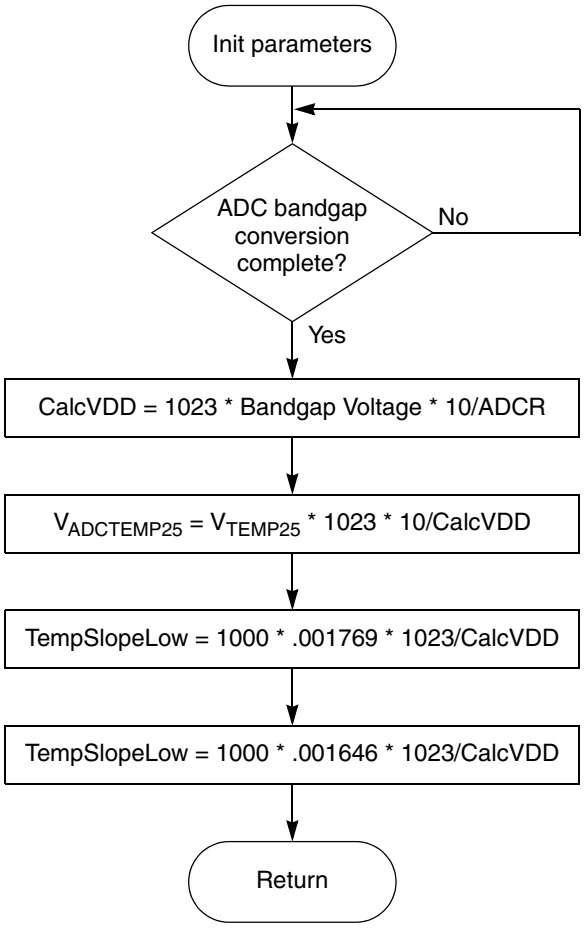
$$\text{Temperature} = 25 - \left[\frac{\text{ADCR} - V_{ADCTEMP25}}{\left(m_{ADC} / 1000 \right)} \right] \qquad \textit{Eqn. 11}$$

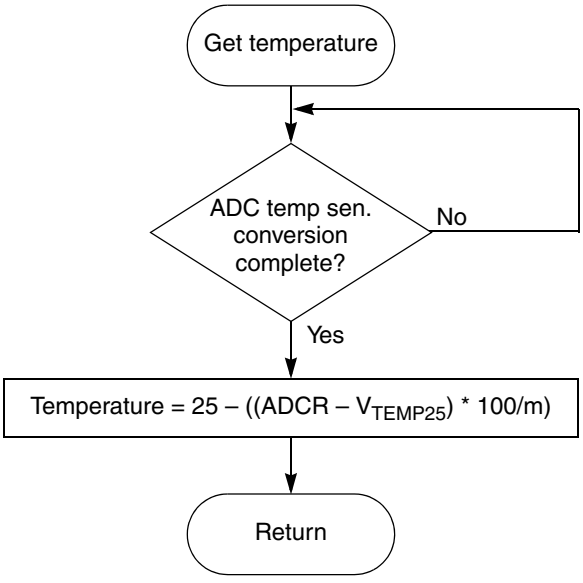
Remember the real value multiplied by 1000 equals the value used for the temperature slope; also, you used a V_{DD} multiplied by 10 in all your calculations. You must multiply the subtraction by 100 to have the same order in the final result. This gives the final equation (Equation 12).

$$\text{Temperature} = 25 - \left[\frac{(\text{ADCR} - V_{ADCTEMP25}) \times 100}{(m_{ADC})} \right] \qquad \textit{Eqn. 12}$$

Also, the VDD is calculated with only one decimal value and can add errors for some scenarios of the calculation. For instance, if VDD = 2.97 volts, the calculated VDD will be 2.9 and all the results carry the same error. Fix this and add more precision by having two decimal digits; it is up to the user to determine the way to obtain the best results according to the application. In this case, the finished project is an example on how to implement this.

5.3 Initialization Flowcharts





	Uncalibrated	Calibrated (3 Points)	Uncalibrated	Calibrated (3 Points)
	Floating-Point		Fixed-Point	
Code Size	2295bytes[1]		698 bytes[1]	
# of cycles	4636[1]			
Typical Accuracy	± 8degs C	± 2.5degs C	±18degs C	±12degs C
Ease of implementation	Easiest	Difficult	Easy	Most Difficult

[1] Using calibrated parameters to calculate temperature does not affect code size or execution times.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3031
Rev. 1
04/2010

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
semiconductor