

# Initialization/Algorithm Design and DC Voltage Regulation

## Application Note

Adam Eshkanian  
Friday, March 30 2007

### **Executive Summary:**

This application note explains how to initialize/calibrate a sensor when the device is first turned on, and also how to design algorithms to properly analyze the output of the sensor. How to convert a DC voltage into a desired DC voltage will also be briefly discussed.

### **Key Words:**

- PIC
- While Loop
- Function
- Library
- Voltage Regulator
- Linear Regulator
- Switching Regulator

## Introduction:

Many electrical devices today have a sensor as one of its components. These sensors can be used for applications such as temperature sensing (thermometer) or acceleration sensing (accelerometer). In order to properly use these devices they must be initialized to cancel out unwanted “noise” that may throw off calculations. Most devices that use a sensor also do analysis of their outputs, which requires algorithm design to perform specific calculations. These initialization and algorithms are executed using programmable microcontrollers.

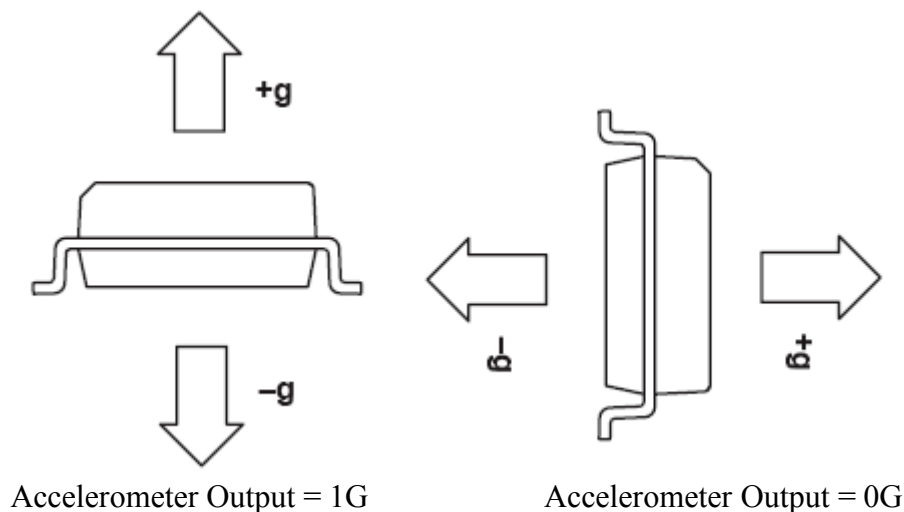
Also, all electrical devices are required to operate at very specific voltages. This usually can not be achieved by hooking batteries together and connecting them directly to the circuit. DC voltage regulators are commonly used to solve this problem.

## Using a PIC:

There is no sensor that can just be hooked up directly to a display and turned on and the correct output is shown. The output needs to be processed, and using a PIC is a great way to do this. A PIC is a microcontroller that can be programmed by a computer program such as Microchip MPLab to perform certain functions and calculations.

## Initialization:

To further explain why a sensor needs to be initialized a single axial accelerometer will be used as an example. Many companies use these accelerometers to measure vibrations of machinery or damaging shipping vibrations. When the axis of this accelerometer is faced perpendicular to the Earth it will give an output that represents the Earth’s gravitational pull, but if it is parallel to the Earth the output will represent no acceleration no matter the orientation of the accelerometer. The diagram below illustrates this problem:



1G = 1X the Earth's gravitational pull.

It is recommended that the initialization process is its own separate function that is called upon before any other code is executed. A function is simply a chunk of code that can be executed when a user tells it to. The function for the initialization process should return the value of the offset that is calculated. To start off the initialization a while loop should be used to sample the sensor output many times to get an accurate offset value. A while loop is just a statement that tells a program to run lines of code a certain amount of times. This is what the while loop will look like:

```
while (init_count < 1000)    //sample 1000 times to get accurate reading
{
    init_count++;           //adds 1 to init_count every time it is executed
}
}
```

Init\_count is the variable that defines the amount of times the code will loop. Next the code should be inserted to average the values output by the sensor. The loop will now look like this:

```
while (init_count < 1000)    //sample 1000 times to get accurate reading
{
    init_count++;           //adds 1 to init_count every time it is executed
    ConvertADC12();        //initializes AD converter (covered in another
                           application note)
    while(BusyADC12());    //read AD converter (covered in another
                           application note).
    voltage_from_sensor = ReadADC12(0)*5/(4096); //converts reading to
                                                voltage
    voltage_from_sensor_temp =
    voltage_from_sensor_temp+voltage_from_sensor; //sums initial
                                                voltage samples
    avg_volts_from_sensor= voltage_from_sensor_temp/init_count;
                                                //averages voltage samples
}
}
```

Now using the average DC voltage level that has just been calculated, the offset value will be calculated using this line after the loop:

```
offset = avg_volts_from_sensor -
reference voltage from sensor (usually the output that represents 0);
                                                //finds offset value
```

This offset value will now be used in the algorithms portion of the application note.

## Algorithm Design:

All different sensing devices require different algorithms depending on the type of analysis necessary. Because of this an accelerometer that needs to calculate RMS (root mean square) value of its output subtracting any DC offset values that it senses will be used as an example. The best way to design an algorithm is to first decide what the device needs to accomplish then write it down on a piece of paper. The next step would be to convert that algorithm so it can be coded using a language such as C or C++. The equation for calculating the RMS value is shown below:

$$x_{\text{rms}} = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \dots + x_n^2}{n}}$$

This equation squares an input, averages it, and then takes the square root of that average. The variable “n” represents the number of values being averaged. The next step is to convert this into an algorithm that can be coded. The first line will subtract any DC levels from the output of the accelerometer. To calculate the DC level use function described in Initialization section. 2.5V is the voltage output that represents 0 acceleration for this accelerometer:

```
calibrated_voltage = voltage_from_accel-offset;    //sets initial value to 2.5V
```

The next line will square the sample taken from the accelerometer:

```
squared_volts = calibrated_voltage*calibrated_voltage;  
                //squares output form accel.
```

Next, these samples must be summed and averaged:

```
average_temp = average_temp + squared_volts;  
average = average_temp/count;
```

The variable “count” represents the number of samples that have been taken. Finally the square root of “average” must be calculated. Remember to add the right libraries or some of these calculations will not work. A library is a set of predefined functions that can be called by the user instead of having to write new algorithms every time. In this case for programming in C the “math.h” library must be included for the next line:

```
RMS = sqrt(average);    //takes the square root of the average calculated above
```

Follow this process for any calculations and the correct algorithm can be achieved.

## **DC Voltage Regulation:**

Most components of an electronic device require a constant DC voltage supply. Batteries directly connected to the circuit can not be used because battery voltages tend to drop the more they are used. To solve this problem voltage regulators are used. They can take any input and convert it to the desired output. The two most common types of regulators are linear and switching regulators. These can be either fixed or variable depending on their applications.

Linear voltage regulators use elements such as transistors operating in their linear region to control the voltage. These regulators have a very stable output and are very resistive to outside noise, and they have a fast response time to changing voltages.

Switching regulators use switches that turn on and off at a certain duty cycle controlling the amount of charge that is transferred to the circuit. The benefit of switching regulators is they can generate output voltages that are higher than their input, and they can also generate voltages of opposite polarity. Also, they are much more power efficient than linear voltage regulators.

A fixed voltage regulator always outputs the same voltage. Most circuit devices operate at a very common voltage such as 5V for example. For this, fixed voltage regulators can be used. If the voltage is uncommon however, a variable regulator needs to be used. A variable voltage regulator is the same as a fixed regulator except its output can be changed. These often require extra circuitry to function properly so they are avoided when space is important.

## **Conclusion:**

The initialization process for a sensing device is very important in gathering accurate data that can be analysed correctly. Although all electronic devices are different, they can be initialized in a way very similar to the one described in this application note.

The algorithms used to analyze data from a sensor can vary greatly, but the process to design those algorithms can be applied to all of them. This process can turn long complicated complications into a simple process.

DC voltage regulators play a very important role in the operation of circuits. Most components require a constant input voltage, but the voltage supplied may not be. These convert, stabilize, and clean up the power before it reaches the circuit.

## **References:**

Wikipedia Encyclopedia

[http://en.wikipedia.org/wiki/Root\\_mean\\_square](http://en.wikipedia.org/wiki/Root_mean_square)

[http://en.wikipedia.org/wiki/Voltage\\_regulator](http://en.wikipedia.org/wiki/Voltage_regulator)

MMA1250D Accelerometer Data Sheet

<http://www.compel.ru/images/catalog/120/MMA1250D.pdf>