Bosch BMP085 Barometer
Floating Point Pressure Calculations

...and some analysis...

April 25, 2011

Here's a set of equations for computing pressure with the Bosch BMP085 pressure sensor that use floating point math instead of the integer math published by Bosch. There are two advantages over the integer math.

- Integer math results in stair-step, jumpy corrections as the input values vary smoothly. This is due to round-off errors in the integer calculations. The floating point math does not suffer from this problem and corrections vary smoothly with changes in input values.
- The equations are much simpler and easier to implement without error. Some new calibration values have been defined in terms of the Bosch calibration constants. Temperature and pressure calculations have been reduced to some simple ratios and three second-order polynomials.

1 The Elusive Header File

It is relatively easy to find the C-language reference file `bmp085.c` (which contains an example of the integer math) on the internet. It is not easy to find the accompanying header file (`bmp085.h`) which contains some very important details. This file has finally been located and reveals that not all of the 16-bit calibration constants contained in the BMP085's non-volatile memory are signed values. In fact, the values AC_4, AC_5, AC_6 are all unsigned 16-bit integers. At least one BMP085 unit has been seen which has an AC_4 value higher than 32,767 so this distinction is indeed important.

2 The Constants

There were a couple of places in the integer math where a constant was added to an expression before right-shifting to provide a rounding behavior instead of truncation. These constants have been removed from the floating point equations.

All references to the BMP085 calibration values (e.g. AC_4 or M_C) use the integer values of these constants. As mentioned above, be sure to interpret values as unsigned where necessary.

These floating point formulas start with a new set of floating point calibration factors. These are derived from the EEPROM integer calibration data, which is described on the Bosch data sheet. The first three values are only used in computing the final constants below and can be discarded afterwards.

$$
\begin{aligned}
c_3 &= 160 \cdot 2^{-15} \cdot AC_3 \\
c_4 &= 10^{-3} \cdot 2^{-15} \cdot AC_4 \\
b_1 &= 160^2 \cdot 2^{-30} \cdot B_1
\end{aligned}
$$

The next four constants are used in the computation of temperature.

$$
\begin{aligned}
c_5 &= \frac{2^{-15}}{160} \cdot AC_5 \\
c_6 &= AC_6 \\
m_c &= \frac{2^{11}}{160^2} \cdot M_C \\
m_d &= \frac{M_D}{160}
\end{aligned}
$$

Three second order polynomials are used to compute pressure, and they require another nine constants (three for each polynomial).

$$
\begin{aligned}
x_0 &= AC_1 \\
x_1 &= 160 \cdot 2^{-13} \cdot AC_2 \\
x_2 &= 160^2 \cdot 2^{-25} \cdot B_2
\end{aligned}
$$

$$
\begin{aligned}
y_0 &= c_4 \cdot 2^{15} \\
y_1 &= c_4 \cdot c_3 \\
y_2 &= c_4 \cdot b_1
\end{aligned}
$$

$$
\begin{aligned}
p_0 &= \frac{3791 - 8}{1600} \\
p_1 &= 1 - 7357 \cdot 2^{-20} \\
p_2 &= 3038 \cdot 100 \cdot 2^{-36}
\end{aligned}
$$

In subtracting 8 in the numerator of the equation for p_0, it is assumed the original value was offset for the purpose of integer rounding. If this assumption is wrong, there will be a small offset in the result of 0.005mb. However, since the device will almost always have an initial offset error much greater than this it is not of much concern.

The coefficients above have been scaled so that the final results will be temperature (T) in degrees Celsius, and pressure (P) in millibars (a.k.a hecto-Pascals).

This wraps up the calculation of floating point constants that are used for converting the raw, integer temperature and pressure data into degrees Celsius and millibars.

3 The Formulas

The raw real-time data from the BMP085 will be represented by t_u for temperature and p_u for pressure. The temperature value is just equal to the integer value that is read from the barometer. The pressure reading is formatted a bit different however; it is treated like a fixed point fraction with the radix point just to the right of the LSB. After these integer values are acquired, all remaining calculations below are floating point. These first two formulas treat the integer register values as unsigned quantities.

$$t_u = 256 \cdot \mathrm{MSB} + \mathrm{LSB}$$

$$p_u = 256 \cdot \mathrm{MSB} + \mathrm{LSB} + \frac{\mathrm{XLSB}}{256}$$

Defining p_u in this way alleviates the need to make any further adjustments for the over-sampling ratio (OSS). If OSS is zero, it is not necessary to read the XLSB register and its value can be set to zero.

The temperature computation is quite simple. First we define a new value, α and then compute temperature in terms of this new variable:

$$\alpha = c_5(t_u - c_6)\ ;\ \ T = \alpha + \frac{m_c}{\alpha + m_d}$$

Computing pressure requires the temperature (T) computed above, and uses three second-order polynomials. The first two generate offset (x) and scaling factors (y) which are functions of the difference between the current temperature and 25C (which we'll call s).

$$s = T - 25$$

$$x = x_2 s^2 + x_1 s + x_0$$

$$y = y_2 s^2 + y_1 s + y_0$$

These two factors are now used to compute an initial pressure value, z:

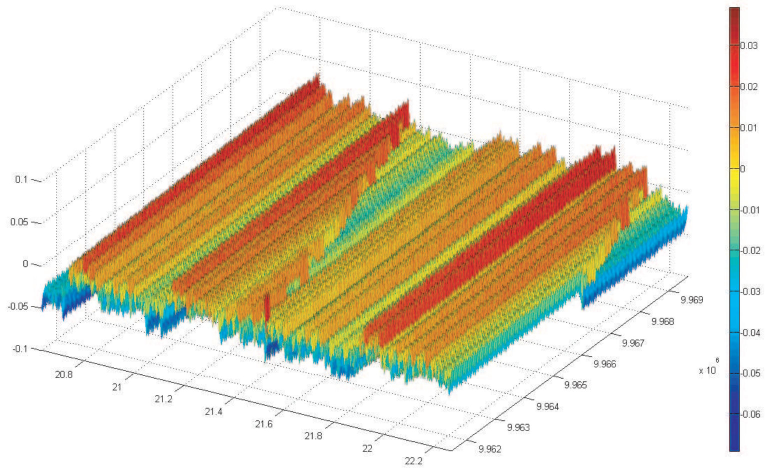

Figure 1: Comparison of Pressure Computations

$$z = \frac{p_u - x}{y}$$

A polynomial in z gives the final corrected pressure value.

$$P = p_2 z^2 + p_1 z + p_0$$

4 Verification

These results were compared the published Bosch integer code, over a large grid of temperature and pressure values. The difference varies over a range of approximately ± 0.05mb and the average offset is around 0.02mb. Figure 1 shows a 3-dimensional plot of the difference between integer and floating point values. The x/y axes cover different values of temperature and un-corrected pressure input values. The vertical scale is in millibars, and is color coded as shown by the bar on the right. The jagged appearance of this plot is entirely due to round-off errors in the integer math.

These formulas therefore accurately express the proper corrections over a large range of inputs, without the stair-step, jumpy nature of corrections from integer math.

5 Usefulness

To keep this in perspective, it is worthwhile to take a look at the Bosch specifications:

Absolute accuracy	± 2.5mb
Relative accuracy	± 0.2mb

The improved floating point algorithm gives an improvement on the order of 0.05mb, which is about one-fourth of the relative accuracy specification.

So, the improvements are of limited value – but still may be of interest to some folks. These equations are also much simpler to implement and may be preferable if the floating point resources are available on the hardware used to make the conversion.

6 An Example

Here is an example of the calculations to dispel any questions about the equations above.

First, the values of the calibration coefficients. The values AC_4, AC_5, AC_6 are unsigned 16-bit integers. All other values are signed (two's complement) 16-bit quantities.

BMP085 Calibration Data			
AC_1	7911	B_1	5498
AC_2	-934	B_2	46
AC_3	-14306	M_B	-32768
AC_4	31567	M_C	-11075
AC_5	25671	M_D	2432
AC_6	18974		

Next, the derived values – not given to a great deal of precision:

Derived Calibration Data					
c_3	-69.8535	x_0	7911	p_0	2.3644
c_4	0.000963	x_1	-18.2422	p_1	0.993
c_5	0.0049	x_2	0.0351	p_2	0.000442
c_6	18974	y_0	31.567		
b_1	0.1311	y_1	-0.0673		
m_c	-886	y_2	0.000126		
m_d	15.2				

This data remains constant for any given sensor and need not be read and computed every time the sensor is powered on. Once all values are computed, it is not necessary to retain the values of c_3, c_4, b_1.

Given a hexadecimal temperature reading of 0x69EC and a pressure reading of 0x982FC0, the calculations are as follows:

$$\begin{aligned}
t_u &= 256 \cdot 0\text{x}69 + 0\text{xEC} = 27116 \\
p_u &= 256 \cdot 0\text{x}98 + 0\text{x2F} + \frac{0\text{xC0}}{256} = 38959.75 \\
T &= 23.7764 \\
s &= -1.2236 \\
x &= 7933.4 \\
y &= 31.6495 \\
z &= 980.3108 \\
P &= 980.0456
\end{aligned}$$

7 Resolution

In the above example, it is easy to verify that one LSB in temperature data is equivalent to roughly 0.006 degC.

The value of y is approximately equal to the number of integer p_u counts per millibar of pressure. In this example, the OSS was 3, so the actual resolution of p_u is 2^{-3} or 0.125. From this, the resolution of the raw readings is approximately $31.65 \cdot 8 \approx 253$ counts per millibar. One LSB in the data is then equal to about 0.004mb or 0.4Pa. This corresponds to about 1.5 inches of altitude change at sea level.

8 Performance

To demonstrate the improvement in using floating point math, two plots were generated using a fixed pressure reading of 0x980000. The temperature reading was incremented from 0x6800 to 0x68FF (roughly, from 20.6 to 22.3 degC). Pressure computations were performed with these inputs using both the Bosch integer code and the floating point formulas developed above.

The figure 2 shows the two results plotted on top of one another. There is not a lot of difference, although the blue line (integer math) is noticeably noisier.

To make the differences easier to see, each of the lines was “flattened” by subtracting a best-fit straight line from each result. The next two graphs show this result. In the case of integer math (figure 3), the only thing clearly visible is the jumpy, staircase behavior resulting from integer rounding. Notice that the magnitude of these jumps is typically on the order of 0.03 to 0.04mb which is about 10 times larger than the resolution of 0.004mb with OSS=3.

The data computed with floating point math (figure 4) shows a completely different picture. First, the y-axis scale is 10 times smaller than in the preceding graph. The parabolic behavior of the corrections (due to the various second-order polynomials in the math) is clearly visible with almost no apparent noise. Over this temperature range, the corrections only change by a little more than 1-bit of resolution (0.004mb), but the corrections are not going to add 10 or more LSB counts of noise to the data as in the case with integer math.

Figure 5 shows a similar comparison of corrections over a wider temperature range. Here, the parabolic nature of corrections is just barely visible in the integer data. The floating point results are striking in comparison.

Again, keeping this in perspective, the correction jumps of about 0.04mb seen with integer math correspond to about one foot of altitude change at sea level. Can the sensor really detect these small changes in level, especially superimposed upon local changes in barometric pressure?

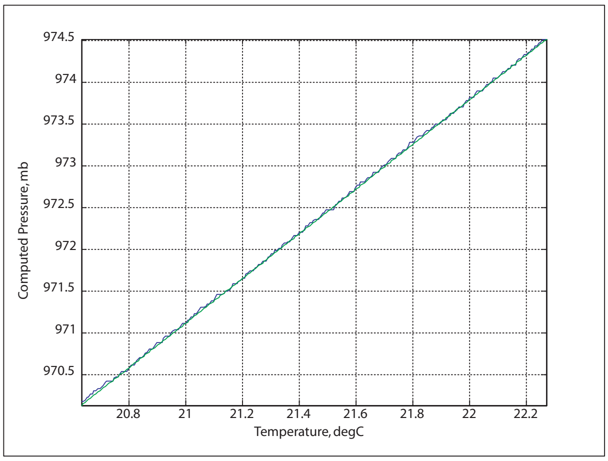

Figure 2: Comparison of Pressure Computations

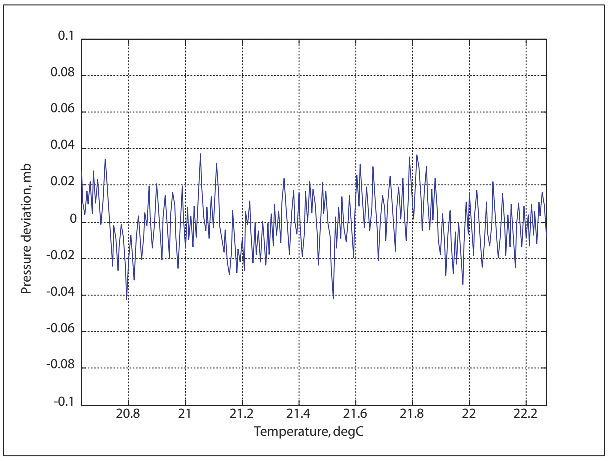

Figure 3: Deviation of integer math from straight line.

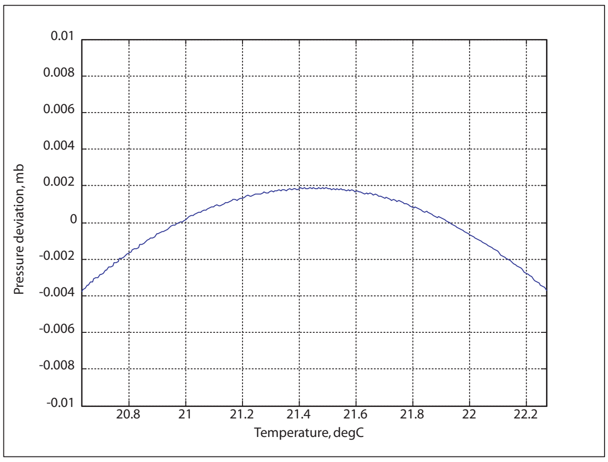

Figure 4: Deviation of floating point math from straight line.

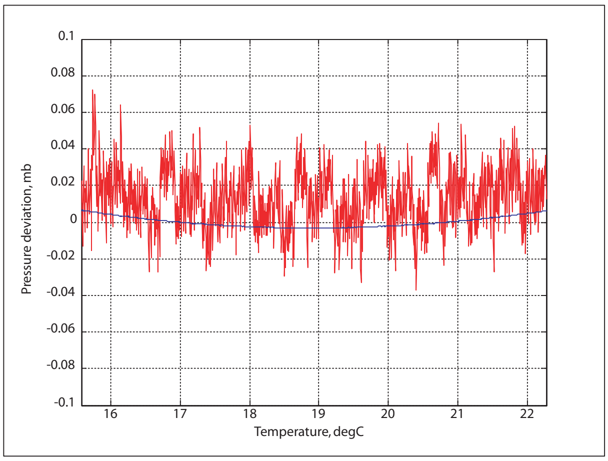

Figure 5: Comparison over a wider temperature range.