Modern Altimeter and Barometer System using the MPL115A

by: John B. Young

INTRODUCTION

Portable electronics are starting to integrate sensors into handheld and desktop electronics with requirements of low power and a small form factor. Key markets are in the cellular, mobile electronics and desktop arenas. Pressure sensors are being utilized in small form factors to measure barometric pressure as a means to analyze surroundings. This is occurring in portable devices for weather measurement or in control systems for electronics dependent on operating pressures.

The MPL115A is an absolute pressure sensor with a digital output that meets the low cost, low power, and small footprint requirements. This is available in both SPI and I^2C digital variations. The unique aspect of the MPL115A is that compensated pressure is attained via a host microcontroller. MPL115A stores compensation coefficients in internal ROM. A host microcontroller can access these coefficients along with raw Pressure and Temperature values to calculate the correct compensated pressure. Over a range of -20°C to 85°C it has an accuracy of ±1 kPa.

This part with its form factor of 5 x 3 x 1.2 mm lends itself to customers that require a small part for a HDD, a GPS unit, portable weather station, and for basic Altimetry. Low power consumption on the order of 5 μA during Active operation (1 reading per second) or 1 μA during Shutdown (Sleep) mode also aligns this part for battery or power-conscious designs.

With the compensated digital output, the implementation of the pressure sensor removes the requirement to do an Auto-zero for offset shifts or to characterize for temperature changes. Application implementation is much faster; less factors skewing the output over pressure and temperature form the customer side. This is the advantage of a digital sensor that is trimmed for pressure and temperature changes at the factory. For portable weather stations or basic altimetry it is simply conversion from pressure to height, change in pressure over time all without dealing with changing temperature and offset value changes. This allows customers to work mainly on the algorithms and data analysis.

ALTIMETRY

Altimetry utilizes absolute pressure sensors. An absolute sensor measures the deflection of the surrounding barometric pressure with reference to a known pressure (usually a vacuum). This allows it to compare the air pressure at sea level (101.3 kPa) to the vacuum to gain an absolute pressure result. At a different elevation, the barometric (surrounding) pressure can be compared again to the vacuum for that absolute pressure result. Since both readings were taken against the same reference, they can be compared against each other.

Barometric pressure does not have a linear relationship with altitude. As altitude increases, the pressure decreases. Common reference points are given in Table 1.

Table 1.

Location	Altitude (m)	Altitude (ft)	Pressure (kPa)
Sea Level	0	0	101.3
Dead Sea (lowest surface on earth)	-396	-1300	106
Summit of Everest	10,058	+33,000	33

© Freescale Semiconductor, Inc., 2009. All rights reserved.

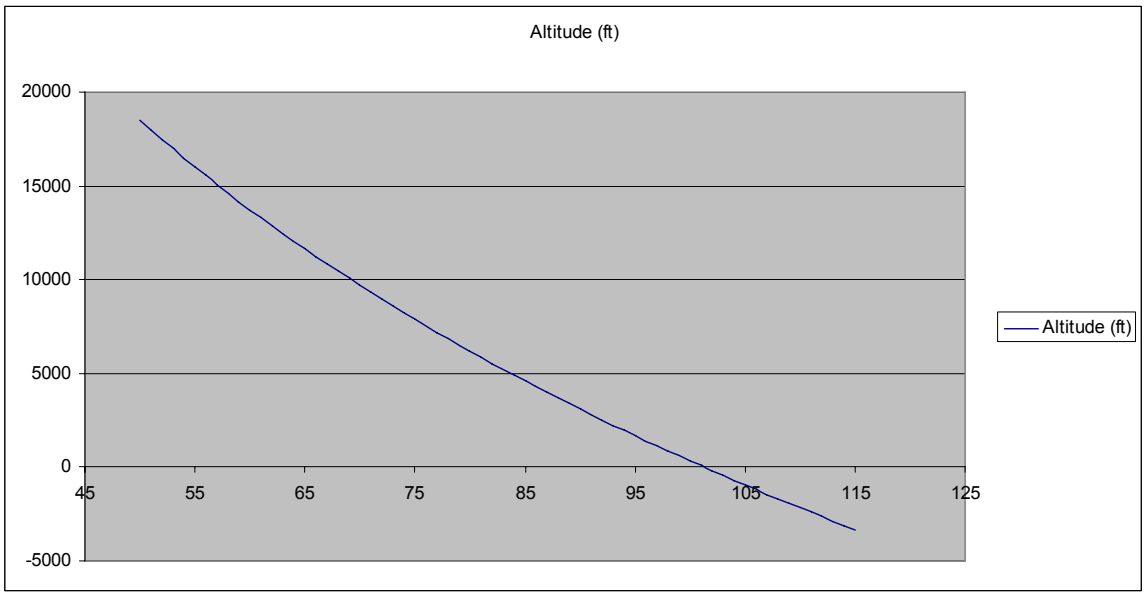

Figure 1. Pressure (kPa) vs. Altitude (ft)

At low elevations, a square meter on the earths surface has greater weight above it than at higher altitudes. This is due to the mass of water vapor and air that sits upon it. Imagine cubes of air and water vapor stacked from the ground to space. At the low altitude there is more cubic mass above, while higher altitudes will have less of these stacked above it.

Air and water vapor will compress more at sea level and the air is significantly denser than at high altitude. The density is not uniform with altitude, and thus the pressure is not either. The reason for altitude's non-linear relationship is that air has infinite compressibility. It disproportionately compresses more as weight is placed upon it. Hence the graph of Pressure vs. Altitude seen in Figure 1.

The simplified mathematical equation used to calculate Altimetry in Table 2:

$$ph = p0 \cdot e^{\frac{-h}{7990m}} \qquad \textit{where:} \qquad h = 18400m \cdot \log^{\frac{p0}{ph}}$$

Assuming p0 = 101.3kPa

ph = Pressure (kPa) at height h (m)

p0 = Initial Pressure point at sea level or 101.3 kPa.

Table 2. Pressure vs. Altitude

Pressure kPa	ADC Counts	Altitude (m)	Altitude (ft)
50	0	5642.17	18511.06
51	15	5483.92	17991.86
52	31	5328.75	17482.78
53	47	5176.54	16983.4
54	62	5027.17	16493.34
55	78	4880.54	16012.27
56	94	4736.55	15539.86
57	110	4595.12	15075.85
58	125	4456.14	14619.88
59	141	4319.54	14171.72
60	157	4185.23	13731.07
61	173	4053.14	13297.7
62	188	3923.21	12871.42
63	204	3795.35	12451.94
64	220	3669.5	12039.04
65	236	3545.61	11632.58
66	251	3423.61	11232.32
67	267	3303.44	10838.06
68	283	3185.05	10449.64
69	299	3068.39	10066.9
70	314	2953.41	9689.67
71	330	2840.06	9317.78
72	346	2728.3	8951.12
73	361	2618.07	8589.47
74	377	2509.35	8232.78
75	393	2402.09	7880.87
76	409	2296.24	7533.6
77	424	2191.78	7190.88
78	440	2088.67	6852.59
79	456	1986.88	6518.64
80	472	1886.36	6188.85
81	487	1787.09	5863.16
82	503	1689.04	5541.47
83	519	1592.18	5223.69
84	535	1496.47	4909.68
85	550	1401.91	4599.44
86	566	1308.44	4292.78
87	582	1216.06	3989.7
88	598	1124.73	3690.06
89	613	1034.44	3393.83
90	629	945.15	3100.89
91	645	856.85	2811.19
92	661	769.52	2524.67
93	676	683.13	2241.24
94	692	597.66	1960.83
95	708	513.1	1683.4
96	723	429.42	1408.86
97	739	346.61	1137.17
98	755	264.65	868.27
99	771	183.53	602.13
100	786	103.21	338.62
101	802	23.7	77.76
102	818	-55.03	-180.54
103	834	-132.99	-436.32
104	849	-210.2	-689.63
105	865	-286.67	-940.52
106	881	-362.41	-1189.01
107	897	-437.45	-1435.2
108	912	-511.78	-1679.07
109	928	-585.43	-1920.7
110	944	-658.41	-2160.14
111	960	-730.73	-2397.41
112	975	-802.4	-2632.55
113	991	-873.43	-2865.58
114	1007	-943.84	-3096.59
115	1022	-1013.63	-3325.56

Table 3. Pressure and Temperature World-Wide at that Altitude

Altitude (m)	Altitude (ft)	Pressure (mmHg)	Pressure (kPa)	Temp Avg (°C)
0	0	706	94.13	15
305	1000	732.9	97.7	13
610	2000	706.6	94.2	11
914	3000	681.1	90.81	9.1
1220	4000	656.3	87.5	7.1
2133	7000	586.4	78.18	1.1
2743	9000	543.2	72.42	-2.8
3353	11000	502.6	67.01	-6.8
4572	15000	428.8	57.17	-14.7
6096	20000	349.9	46.65	-24.6

Note: For large range measurements, temperature shifts become important as the temperature compensation of the MPL115A sensor comes into a dramatic role.

Although the altitude is not going to be linear with pressure over the 50 to 115 kPa range, it can be assumed to be linear for very small sections. i.e. from a given location to 10 m in higher or lower elevation, the pressure can be assumed linear. Notice too that the pressure can be better resolved in ADC counts at lower altitude than at higher ones.

For example pressure at 50 kPa vs. 101 kPa (sea level): 50 kPa = 5642.17 m; 51 kPa = 5483.92 m. The difference is 158.25 m for the 1 kPa change in pressure.

At 101 kPa the altitude is 23.7 m. 102 kPa is -55.03 m (below sea level). This is 78.73 m for the 1 kPa change in pressure.

Given that the ADC counts are about 15 to 16 counts for each kPa there is better resolution of 5.24 meters/count for the sea level (101 to 102 kPa) than at high elevation 10.55 m/count (50 to 51 kPa). Sea level and below will yield the best altitude to ADC count relationship, as it measures the smallest distance with the most amount of ADC counts. This is because the air is most dense in this region. It also makes the part more useful as the most populated regions do not reside in the 50 kpa area, but between sea level and 81 kPa. A reference table has been included in the appendix for cities and their altitudes.

Overview

The host microcontroller does the main calculations to determine the Pressure compensated for Temperature variations. MPL115A contains coefficients that are used via a second order compensation algorithm to determine the Pcomp (Compensated Pressure). The coefficients are read to the host microcontroller via I^2C or SPI in memory address locations where they are stored. The host microcontroller also reads raw Pressure and Temperature data directly from MPL115A. With the combination of the coefficients, Raw Temperature and Pressure, the compensation algorithm is used to calculate Pcomp. The value is a 10 bit number 0 to 1023 which represents 50 to 115 kPa respectively.

Note: Simple calculations would lead that (Pressure Range)/(ADC counts) would lead to a resolution of 65 kPa/1023 or 0.064 kPa per an ADC count. However, due to code-skipping this is not necessarily true. The data sheet has a specification of 0.15 kPa Resolution that is more accurate. This is that every 2.36 ADC counts represent a 0.15 kPa pressure change.

We guarantee a 1 kPa accuracy across the -40°C to 105°C. While we get better results than this, it is the conservative data sheet spec that is guaranteed. A 0.15 kPa change at sea level is the equivalent of 11.8 m for a 0.15 kPa pressure difference. This is done over 2.36 ADC counts, so it can be ‘pushed’ towards a possible 5 m per ADC count at this level. It will measure the 11.8 m interval more accurately than a 5 m interval. Note that this is not the same for all altitudes. Higher altitudes will have decreased performance, but can be calculated.

Pressure Change	Altitude Change (m)	ADC Change (counts)	(meters/count)
50 kPa to 49.85 kPa	-24	2.36	10
70 kPa to 69.85 kPa	-17.1	2.36	7.2
90 kPa to 89.85 kPa	-13.3	2.36	5.6
101 kPa to 101.15 kPa	11.9	2.36	5

The MPL115A has a specification of a 0.15 kPa Resolution. While searching online for the current barometric pressure, the correct pressure may not be found. Most airports and cities have their current barometric pressure listed for weather conditions. In order to determine weather patterns, the current barometric conditions are often normalized with sea level. This removes altitude variation from the readings. So if the weather is clear, it may be listed as 101.3 kPa, but in actuality it may be 92 kPa. Please keep this in consideration when looking up the local altitude in kPa.

WEATHER

The MPL115A is an absolute device that can be used to predict and measure the barometric pressure to deduce weather patterns. Weather prediction requires a static location for the sensor and 2-3 hours to analyze a full weather pattern. Typically the pressure changes due to weather are slow, requiring a few hours to determine the sloping of the pressure change. Vertical movement or a significant airflow can interfere with results due to only weather patterns in barometric pressure. The sensor should be kept in a relatively protected area from any strong air flows, and kept at that static location during analysis. Temperature effects can change the results of a normal pressure sensor especially if the measurement is done over several hours in varying temperature. Due to the nature of the calibration and temperature compensation, MPL115A meets these requirements, compensating for temperature swings over a large 0 to 85°C operating range. It will not require auto-zeroing for shifts in offset or span over temperature.

How Pressure Increases and Decreases with Weather

For weather pattern prediction, the MPL115A is a well suited device with its pressure range and resolution. Barometric pressure changes can directly correlate to changes in the weather. Low pressure is typically seen as the precursor to worsening weather. High pressure increases can be interpreted as improving or clear weather. The typical reasoning can be seen in a comparison of molecular weights. If air is approximately 21% O_2(g), 78% N_2(g), O_2(g) has a molecular mass of 32, N_2(g) has a mass of 28. H_2O (g) has a molecular mass of 18. So if there is a large amount of water vapor present in air, this air is going to be lighter than just regular dry air by itself. It's an interesting fact that explains how weather patterns lead to high or low pressure.

If bad weather originates in an area in the formation of water-vapor clouds, this is falling pressure on a barometer. The vapor will reduce the barometric pressure as the H_2O reduces the mass above that point on the earth. High pressure will signal the clearing of the water vapor as the air dries.

Another quandary is how weather during severe hurricanes/cyclones with high 150 mph winds be defined as low pressure? This is due to the fact that hurricanes are low pressure conditions surrounded by higher pressure. The rush of air from higher to low pressure creates the fast moving winds. The lower the pressure in the center, the greater differential pressure between high and low areas. This leads to a stronger cyclone or hurricane.

Some areas are harder to predict weather patterns. Cities located at the base of mountainous regions where condensation and fog are a daily occurrence is an example. An area like Hawaii where high colder mountains meet low warm sea regions can have harder to predict results. A network of sensors can give a more exact trend, but for a single sensor in a static location, there are a few ways to have a simple standalone weather station.

Local Weather Stations

When implementing a weather station, it is best to will check results with a local forecast. When researching local weather pressures, such as barometric pressure at the closest airport, remember that the weather is normalized for altitude. Normalization takes local barometric pressure and shifts it to reflect sea level altitude. Sea Level is 101.3 kPa, and by normalizing various points on a map, a meteorologist can see the weather pattern over a region. Without the normalization, the effect of altitude on the pressure reported by collection points will lead to useless data. A mountain data point will have pressure affected by altitude and as it leads to the valleys, the pressure point there will be higher, telling nothing about the weather without the normalization.

Airports are typical reporting stations to check barometric pressure. Some display only normalized pressure during a web search. This is such that a pilot landing at any airport can deduce the weather conditions by knowing the barometric pressure. If the airport is located at the beach, or on a mountainous region, normalization of this value removes the barometric variation due to the altitude. It standardizes pressure so that weather patterns can be mapped.

Example:

An airport located at 600 m elevation would have pressure of 93.97 kPa according to our pressure to altitude equation. If the weather was sunny and mostly clear, it would most probably have a published pressure of 101.3 kPa for weather conditions. It may not be extremely clear skies as this would be a high pressure weather system. It would be a stable pressure with neither extreme in low or high pressure.

Remember to discern this information when trying to see if the MPL115A value matches the local weather barometric pressure. Sometimes a disparity in the value occurs due to normalization.

Algorithms for Weather

Simple Approach

How is weather predicted using the barometric sensor? There is a simple approach looking at increasing or decreasing pressure. Simply an increase over time is a trend that approaches “sunny” or “clear” days. Dropping pressure can signal a worsening “cloudy” or “rainy” day. This can be seen typically as a rising or falling bar on many simple solution weather stations. It can be interpreted as an increase/decrease gradient for the user to interpret, but the time interval is not used extensively to reach weather predictions. The user can look at the results for a 12 hour time frame to predict the weather trend.

This table is typically used:

Analysis	Output
dP > +0.25 kPa	Sun Symbol
-0.25 kPa < dP < 0.25 kPa	Sun/Cloud Symbol
dP < -0.25 kPa	Rain Symbol

Another approach that is more direct and quicker in calculating the weather in the simple approach is to know the current altitude. This cuts the need to wait and see a “trend”.

By using the equation below:

$$ph = p0 \cdot e^{\frac{-h}{7990m}}$$

Where p0 = 101.3 kPa, and h is the current altitude, the pressure for the local barometric can be calculated. This is the pressure for good sunny weather at current altitude location.

By using the pressure equation and knowing the normalized good weather pressure for the current location (best for a static weather station), the weather can be deduced by the difference. As in the table for the weather symbols, the ideal pressure is compared to the value from the MPL115A and the appropriate symbol of Sun/Cloud/Rain is selected.

Below simple C code from the DEMOAPEXSENSOR demo kit calculates which weather symbol to display on the LCD screen.

- CurrentAltitude - (m) Altitude in meters that is entered into the system by the user for that current static location.
- Pweather - (kPa) Pressure at the current altitude. It is calculated using the Height (m) to Pressure (kPa) exponential equation, inputting CurrentAltitude in meters. This is the ideal pressure for the current location on a stable relatively sunny day.
- decPcomp - (kPa) Value of compensated pressure from MPL115A.

Simple Weather Station Code

```
//////////////////////
      //SIMPLE WEATHER SECTION
      //////////////////////

      Pweather =  (101.3 * exp(((float)(CurrentAltitude))/(-7900)));


      Simpleweatherdiff = decPcomp-Pweather;
      if (Simpleweatherdiff >0.25)
        Simpleweatherstatus = 0; //Sun Symbol
      if ((Simpleweatherdiff <=0.25) || (Simpleweatherdiff >=(-0.25)))
        Simpleweatherstatus = 1; //Sun/Cloud Symbol
      if (Simpleweatherdiff <(-0.25))
        Simpleweatherstatus = 2; //Rain Symbol
```

Display Code to Output to LCD Screen

```
if (Simpleweatherstatus == 0){
        sprintf(LCDArray, "Sun Symbol");
        LCD_WR (LCDArray,LR3+0);
      }
      if (Simpleweatherstatus == 1){
        sprintf(LCDArray, "Sun/Cloud Symbol");
        LCD_WR (LCDArray,LR3+0);
      }
      if (Simpleweatherstatus == 2){
        sprintf(LCDArray, "Rain Symbol");
        LCD_WR (LCDArray,LR3+0);
      }
```

Lets look at some data:

decPcomp (kPa) (MPL115A Pressure)	PWeather (kPa) (Ideal weather)	Simpleweatherdiff (kPa)	Weather Type
96.6	96.85	-0.25	Sun/Cloud
96.4	96.85	-0.45	Rain
97.4	96.85	0.55	Sun
96.92	96.85	0.07	Sun/Cloud

For example: Altitude for Tempe, AZ = 359 m, thus PWeather = 101.3 x $e^{-359/7990m}$ = 96.85 kPa.

Observing Pressure over time will yield similar results over a 12 hour period. In this case, the changes in the pressure that take place over extended time will be enough for the simple method to figure out the weather patterns. This negates the need for user input of the approximate location/altitude. The weather algorithm will be more accurate at the end of the 12 hour interval as the trend is visible versus at initialization.

Knowing the altitude can also be useful for a dynamically changing location to predict simple weather. Take for example a GPS unit: a GPS unit can give an approximate altitude measurement. Measuring the difference from the MPL115A pressure sensor and calculated pressure from the GPS altitude, gives a close approximation of weather patterns quickly at that dynamically changing point. Weather approximation can be deduced in the symbol style as above.

Advanced Version of Weather Station

A more complex approach is to measure the P/t and see how the gradient is changing over time. As in the simple approach, this does need to be kept in a static location during measurement. Essentially as time progresses, the weather can be broken into more exact categories than the simple approach of basic symbols.

This can also use less time than waiting for a full 12 hours to see the pattern of pressure change. In Table 4, the ranges of pressure change over time leading to the definition of the weather patterns is shown. It is a change in the pressure per one hour. 2-3 hours are needed to deduce how the pressure is migrating.

Table 4. Advanced Weather Determination

Analysis	Output
dP/dt > 0.25 kPa/h	Quickly rising High Pressure System, not stable
0.05 kPa/h < dP/dt < 0.25 kPa/h	Slowly rising High Pressure System, stable good weather
-0.05 kPa/h < dP/dt < 0.05 kPa/h	Stable weather condition
-0.25 kPa/h < dP/dt < -0.05 kPa/h	Slowly falling Low Pressure System, stable rainy weather
dP/dt < -0.25 kPa/h	Quickly falling Low Pressure, Thunderstorm, not stable

In the provided source code, the pressure is sampled every minute for 3 hours/180 minutes into a data array. The first 5 minutes are averaged, followed by 5 minutes near the first ½ hour point. Consecutive ½ hour marks have 5 minute averaged data-points stored. This leads to 7 averaged results over the 180 minutes depicting the pressure every ½ hour. Once the data-points are collected, the patterns are deduced. A flowchart provides the method used in deducing the weather pattern. The initial starting point is the reference from which every ½ hour data point is compared to. As the pressure falls, the value is compared and divided so that the change in pressure per 1 hour is compared every half an hour.

Data Example with Calculation for a Basic 3 Hour Run

Data Analysis

Beginning → End

Time (min)	Current Average Value (counts)	(Current Value - Initial Value)	Multiplier to return to dP/dt for 1 hour	Value Multiplied by (65 kPa/1023 counts) = dP/dt for 1 hour	Designated Weather Pattern
0	740 Initial Value*	0	0	0	Too early, wait a little longer.
30	739.6	-0.2	x2 = -0.4	-0.4 x 0.0635 = -0.0254	Stable Weather
60	739	-1	÷ = -1	-1 x 0.0635 = -0.0635	Long Term Low Pressure Rain/Cloudy
90	738	-2	÷1.5 = -1.33	-1.33 x 0.0635 = -0.0844	
120	737	-3	÷2 = -1.5	-1.5 x 0.0635 = -0.0953	
150	736	-4	÷2.5 = -1.6	= -0.102	
180	736.4	-3.6	÷3 = -1.2	= -0.0762	

*The Initial Value is going to be reset to the value at 2 hours or 120 min = 737 counts and then the cycle is repeated again.

Algorithm Code for Advanced Weather Station

```
if (!Tmr2ms8b_weather)
{

//Minutes:  Number of minutes that have elapsed.
//dP_dt[]: Array to store current compensated Pressure siPcomp.  Stores new value every 1 min, 180 min or 3 hours of current weather.
//If more than 3 hours is stored, it goes back to storing in array cell [0].
//weather_cntr:  counter variable for weather dP_dt[] array.


 Minutes = Minutes +1;
 PTGD_PTGD3 ^= 1;    //D1  RED   LED

if (weather_cntr > 180)
 weather_cntr =6;

 WeatherPressArray[weather_cntr] =siPcomp;

 weather_cntr = weather_cntr++;


if (weather_cntr ==5){
 //Avg pressure in first 5 min, value averaged from 0 to 5 min.
 Pressure_1st_5min = ((WeatherPressArray[1]+WeatherPressArray[2]+WeatherPressArray[3]+WeatherPressArray[4]+WeatherPressArray[5])/5);
}

if (weather_cntr ==35){
 //Avg pressure in 30 min, value averaged from 0 to 5 min.
 Pressure_2nd_30min = ((WeatherPressArray[30]+WeatherPressArray[31]+WeatherPressArray[32]+WeatherPressArray[33]+WeatherPressArray[34])/5);

 Weather_change = (Pressure_2nd_30min - Pressure_1st_5min);

if(pressure_second_round_flag ==0)               //first time initial 3 hour
 dP_dt = ((65.0/1023.0)*2*Weather_change);          //note this is for t = 0.5hour

if(pressure_second_round_flag ==1)               //more than inital 3 hour.
 dP_dt = (((65.0/1023.0)*Weather_change)/1.5);      //divide by 1.5 as this is the difference in time from 0 value.

}
if (weather_cntr ==60){
 //Avg pressure at end of the hour, value averaged from 0 to 5 min.
 Pressure_3rd_55min = ((WeatherPressArray[55]+WeatherPressArray[56]+WeatherPressArray[57]+WeatherPressArray[58]+WeatherPressArray[59])/5);

 Weather_change = (Pressure_3rd_55min - Pressure_1st_5min);

if(pressure_second_round_flag ==0)           //first time initial 3 hour
 dP_dt = ((65.0/1023.0)*Weather_change);       //note this is for t = 1 hour

if(pressure_second_round_flag ==1)           //more than initial 3 hour.
 dP_dt = (((65.0/1023.0)*Weather_change)/2);    //divide by 2 as this is the difference in time from 0 value

}

if (weather_cntr ==95){
 //Avg pressure at end of the hour, value averaged from 0 to 5 min.
 Pressure_4th_90min = ((WeatherPressArray[90]+WeatherPressArray[91]+WeatherPressArray[92]+WeatherPressArray[93]+WeatherPressArray[94])/5);

 Weather_change = (Pressure_4th_90min - Pressure_1st_5min);

if(pressure_second_round_flag ==0)           //first time initial 3 hour
 dP_dt = (((65.0/1023.0)*Weather_change)/1.5);  //note this is for t = 1.5 hour

if(pressure_second_round_flag ==1)           //more than initial 3 hour.
 dP_dt = (((65.0/1023.0)*Weather_change)/2.5);    //divide by 2.5 as this is the difference in time from 0 value

}
```

```
if (weather_cntr ==120){
 //Avg pressure at end of the hour, value averaged from 0 to 5 min.
 Pressure_5th_115min = ((WeatherPressArray[115]+WeatherPressArray[116]+WeatherPressArray[117]+WeatherPressArray[118]+WeatherPressArray[119])/5);

 Weather_change = (Pressure_5th_115min - Pressure_1st_5min);

if(pressure_second_round_flag ==0)          //first time initial 3 hour
 dP_dt = (((65.0/1023.0)*Weather_change)/2);       //note this is for t = 2 hour

if(pressure_second_round_flag ==1)          //more than initial 3 hour.
 dP_dt = (((65.0/1023.0)*Weather_change)/3);    //divide by 3 as this is the difference in time from 0 value


}
if (weather_cntr ==155){
 //Avg pressure at end of the hour, value averaged from 0 to 5 min.
 Pressure_6th_150min = ((WeatherPressArray[150]+WeatherPressArray[151]+WeatherPressArray[152]+WeatherPressArray[153]+WeatherPressArray[154])/5);

 Weather_change = (Pressure_6th_150min - Pressure_1st_5min);

if(pressure_second_round_flag ==0)          //first time initial 3 hour
 dP_dt = (((65.0/1023.0)*Weather_change)/2.5);       //note this is for t = 2.5 hour

if(pressure_second_round_flag ==1)          //more than initial 3 hour.
 dP_dt = (((65.0/1023.0)*Weather_change)/3.5);    //divide by 3.5 as this is the difference in time from 0 value

}
if (weather_cntr ==180){
 //Avg pressure at end of the hour, value averaged from 0 to 5 min.
 Pressure_7th_180min = ((WeatherPressArray[175]+WeatherPressArray[176]+WeatherPressArray[177]+WeatherPressArray[178]+WeatherPressArray[179])/5);

 Weather_change = (Pressure_7th_180min - Pressure_1st_5min);

if(pressure_second_round_flag ==0)          //first time initial 3 hour
 dP_dt = (((65.0/1023.0)*Weather_change)/3);       //note this is for t = 3 hour

if(pressure_second_round_flag ==1)          //more than initial 3 hour.
 dP_dt = (((65.0/1023.0)*Weather_change)/4);    //divide by 4 as this is the difference in time from 0 value

 Pressure_1st_5min = Pressure_6th_150min;  //Equating the pressure at 0 to the pressure at 2 hour after 3 hours have past.
 pressure_second_round_flag =1; // flag to let you know that this is on the past 3 hour mark.  Initialized to 0 outside main loop.

}


if ((dP_dt  > (-0.05)) && (dP_dt <0.05))
 weatherstatus = 0; // Stable weather

if ((dP_dt  >0.05) && (dP_dt <0.25))
 weatherstatus = 1; // Slowly rising HP stable good weather

if ((dP_dt  > (-0.25)) && (dP_dt <(-0.05)))
 weatherstatus = 2; // Slowly falling Low Pressure System, stable rainy weather

if (dP_dt  > 0.25)
 weatherstatus = 3; // Quickly rising HP, not stable weather

if (dP_dt  < (-0.25))
 weatherstatus = 4; // Quickly falling LP, Thunderstorm, not stable

//if ((weather_cntr <35) && (weatherfirstpass ==1))
if ((weather_cntr <35) && (pressure_second_round_flag !=1))   //if time is less than 35 min on the first 3 hour interval.
 weatherstatus =5; //Unknown, more TIME needed

Tmr2ms8b_weather = 29500;    //Sampling every minute

}
```

Display Code to Output to LCD Screen:

```
//ADVANCED DP/Dt SECTION

    if (weatherstatus == 0){
     sprintf(LCDArray, "Stable Weather Patt ");
     LCD_WR (LCDArray,LR3+0);
     sprintf(LCDArray, "            ");
     LCD_WR (LCDArray,LR1+8);
    }
    if (weatherstatus == 1){
     sprintf(LCDArray, "Slowly rising Good Weather");
     LCD_WR (LCDArray,LR3+0);
     sprintf(LCDArray, "Clear/Sunny ");
     LCD_WR (LCDArray,LR1+8);
    }
    if (weatherstatus == 2){
     sprintf(LCDArray, "Slowly falling L-Pressure ");
     LCD_WR (LCDArray,LR3+0);
     sprintf(LCDArray, "Cloudy/Rain ");
     LCD_WR (LCDArray,LR1+8);
    }
    if (weatherstatus == 3){
     sprintf(LCDArray, "Quickly rising H-Press");
     LCD_WR (LCDArray,LR3+0);
     sprintf(LCDArray, "Not Stable");
     LCD_WR (LCDArray,LR1+8);
    }

    if (weatherstatus == 4){
     sprintf(LCDArray, "Quickly falling L-Press");
     LCD_WR (LCDArray,LR3+0);
     sprintf(LCDArray, "Thunderstorm");
     LCD_WR (LCDArray,LR1+8);
    }

    if (weatherstatus == 5){
     sprintf(LCDArray, "Unknown (More Time) ");
     LCD_WR (LCDArray,LR3+0);
     sprintf(LCDArray, "            ");
     LCD_WR (LCDArray,LR1+8);
    }
```

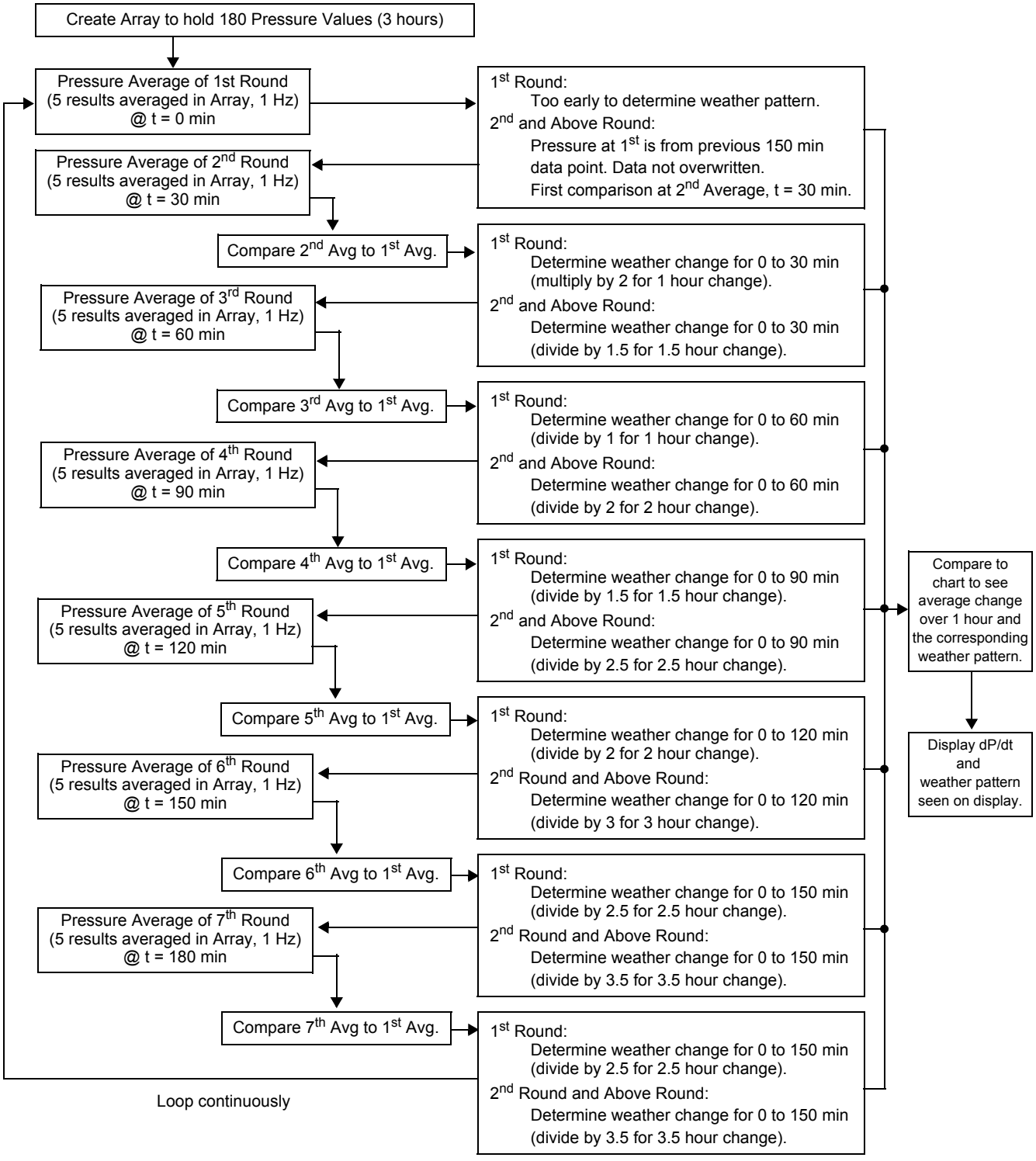

Figure 2. Advanced Weather Flowchart

APPENDIX

Table 5. Worldwide Altitudes

City	Altitude	
	Feet (ft)	Meters (m)
Abu Dubai, UAE	33	10
Accra, Ghana	98	30
Algiers, Algeria	328	100
Amman, Jordan	2297	700
Amsterdam, Netherlands	0	0
Ankara, Turkey	2923	891
Athens, Greece	328	100
Atlanta, Georgia	983	300
Baghdad, Iraq	98	30
Bangkok, Thailand	26	8
Belgrade, Serbia	453	138
Berlin, Germany	164	50
Bern, Switzerland	1640	500
Bogota, Colombia	8727	2660
Bonn, Germany	262	80
Boston, Massachusetts	20	6
Brasilia, Brazil	2297	700
Brussels, Belgium	230	70
Bucharest, Romania	230	70
Budapest, Hungary	656	200
Buenos Aires, Argentina	82	25
Cairo, Egypt	164	50
Canberra, Australia	2297	700
Caracas, Venezuela	3419	1042
Charlotte, North Carolina	721	220
Chicago, Illinois	596	182
Colombo, Sri Lanka	23	7
Copenhagen, Denmark	66	20
Dar es Salaam, Tanzania	230	70
Darwin, Australia	98	30
Denver, Colorado	5184	1580
Doha, Qatar	66	20
Dublin, Ireland	164	50

City	Altitude	
	Feet (ft)	Meters (m)
Haifa, Israel	328	100
Harare, Zimbabwe	3937	1200
Havana, Cuba	98	30
Helsinki, Finland	66	20
Hong Kong, China	108	33
Jakarta, Indonesia	98	30
Jerusalem, Israel	2694	821
Kabul, Afghanistan	5902	1799
Karachi, Pakistan	13	4
Khartoum, Sudan	1280	390
Kinshasa, Congo	984	300
Kuala Lumpur, Malaysia	492	150
Kuwait City, Kuwait	66	20
La Paz, Bolivia	10499	3200
Lagos, Nigeria	98	30
Lima, Peru	420	128
Lisbon, Portugal	312	95
London, United Kingdom	157	48
Los Angles, California	371	113
Lusaka, Zambia	3937	1200
Luxembourg	984	300
Madrid, Spain	2297	700
Manama, Bahrain	66	20
Manila, Philippines	46	14
Mexico City, Mexico	7411	2259
Miami, Florida	82	25
Montevideo, Uruguay	131	40
Moscow, Russia	548	167
Nairobi, Kenya	4921	1500
Nassau, Bahamas	98	30
New Delhi, India	715	218
New York, New York	315	96
Oslo, Norway	394	120

City	Altitude	
	Feet (ft)	Meters (m)
Ottawa, Canada	236	72
Paris, France	262	80
Beijing, China	125	38
Prague, Czech Republic	984	300
Pretoria, South Africa	4265	1300
Quito, Ecuador	9186	2800
Rabat, Morocco	98	30
Reykjavik, Iceland	98	30
Riyadh, Saudi Arabia	1969	600
Rome, Italy	377	115
Salt Lake City, Utah	4330	1320
Santiago, Chile	1703	519
Seoul, South Korea	197	60
Singapore	16	5
Sofia, Bulgaria	2461	750
Stockholm, Sweden	98	30
Taipei, Taiwan	197	60
Tehran, Iran	3999	1219
Tel Aviv, Israel	114	34
Tokyo, Japan	69	21
Tunis, Tunisia	131	40
Vienna, Austria	656	200
Warsaw, Poland	230	70
Wellington, New Zealand	10	3

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2009. All rights reserved.

AN3914
Rev. 0
7/2009