SANDIA REPORT
SAND2002-0770
Unlimited Release
Printed April 2002

Development of an In-Situ Monitoring / Recording Device for Aqueous Processes

Michael E. Partridge, Tedd A. Rohwer, and Randal R. Lockhart

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

A disclosure of invention relating to the subject of this publication has been filed with the U.S. Department of Energy.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

SAND2002-0770
Unlimited Release
Printed April 2002

Development of an In-Situ Monitoring / Recording Device for Aqueous Processes

Michael E. Partridge, Tedd A. Rohwer, and Randal R. Lockhart
Telemetry and Instrumentation Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0986

Abstract

The *In-Situ* Monitoring and Recording Device for Aqueous Processes was developed under a Cooperative Research and Development Agreement (CRADA) with The Procter & Gamble Company. The device is a miniaturized data recorder that measures and records temperature, pH, conductivity, and three-axis acceleration. Intended to autonomously record process data for up to two weeks, the device is tennis-ball-sized, waterproof, has the density of water, contains non-volatile memory, and is powered by a commercially available 9-V battery. The design details are presented, including explanation of circuit function, schematics, and microcontroller code listings.

Acknowledgements

The authors thank Willard B. Hunter, who was the initial Sandia point of contact and was responsible for initial interactions with Mike Rothgeb and Jim Jordan, of The Procter & Gamble Company, resulting in this development.

Ed Henry assisted in the design development, and Dennis Wilder constructed the prototype electronics. Bill Flounders provided invaluable advice regarding pH sensors and pH measurement. Ron Akau completed thermal transport modeling to improve the temperature sensor placement. Our Procter & Gamble counterparts Kris Gansle, Mark Cipollone, and Jonathan Joyce conducted the laboratory validation of our design, and uncovered challenging design problems as the prototypes were evaluated. Keith Fanta, also at P&G, helped correct an electronic design problem late in the project. Tony Mittas took over project responsibility in its final months and provided modifications to the pH, accelerometer, and conductivity circuits.

The authors also thank Toni Kovarik, who negotiated the original CRADA, and Vic Weiss, who took over CRADA management from Toni and negotiated the follow-on extensions.

Table of Contents

Introduction 7
- Sensor Ball Features 7
- Measurement Data List 8
- Battery Operating Time 8

Project Formation 9
- CRADA Timeline 9
- Requirements Summary 9

Measurement Transducers and Circuitry 10

pH Sensor 10
- pH Probe Details 10
- Ag-AgCl Reference Electrode 11
- pH Sensor Electronics Interface 12
- pH Sensor Electronics Design Issues 13

Conductivity Probe 14
- Explanation of Cell Constant's Relationship to Conductivity Measurements 14
- Conductivity Circuitry 14
- A Four-Electrode Design Option 15

Accelerometers 16
- Integrated Accelerometer and Signal Conditioning 16
- Accelerometer Circuit 17

Temperature Transducer 18
- Active Device for Temperature Measurement 18
- Thermal Transport 18
- Temperature Circuit 18

Electronic Design 18

Hardware 19
- Microcontroller 19
- Flash Memory 19
- RS-232 Interface 20
- Linear Regulators and Voltage Monitors 21
- Circuit Board Details 22
- Communication Cable 23

Microcontroller Firmware 23
- Main Code Module, Ball.asm 23
- Delay Timing Loop of Ball.asm 25
- Data Collection Section of Ball.asm 25
- Command Processing Section of Ball.asm 26
- Serial Interface Module, Ser_Hand.asm 26
- Text String and Character Transmit and Serial Interface Configuration Routines 27
- PC Interface Command Processing Module, Cmd_Hand.asm 27
- Flash Memory Control Module, Flash.asm 28
- Data Upload Module, Rd_Hand.asm 29
- Sensor Ball Status Reporting Module, St_Hand.asm 30
- Auxiliary Files 30

Mechanical Design 30
- Shock and Vibration 30
- Water and Chemical Resistance 31
- Sensor Ball Density and Center of Gravity 32

Design Revisions 33
- Substantial Changes in Sensor Ball Version 2 33
- Primarily Firmware Changes for Version 3 33
- Minor Circuit and Firmware Changes to Produce Version 4 34

Suggested Changes for a Future Design Revision 34

Acceptance Testing 36
Testing During Assembly 36
Functionality and Calibration Tests 36
P&G Laboratory and Field Tests 37
Appendix A: P&G Background and Initial Proposal 39
Appendix B: Requirements Document 41
Appendix C: PC Interface Software Guide 47
Appendix D: Sensor Ball Test Procedure 49
Appendix E: Schematics 52
Appendix F: Assembly Drawings and Electronics Materials Lists 58
Controller Board Bill of Materials 65
Signal Conditioner Board Bill of Materials 67
Microcontroller Programming Steps 68
Appendix G: Circuit Board Connector Pin Definitions 69
Appendix I: Mechanical Components 70
Appendix J: Mechanical Assembly Procedure 72
Appendix K: Microcontroller Code Listing 73
Appendix L: PC Interface (SensorBall) Code Listing 145
Distribution 173

Figures

Figure 1. Wire-Frame and Solid Views of the Sensor Ball 10
Figure 2. Construction of the Honeywell pH Probe Tip (figure has errors and needs text) 11
Figure 3. pH DuraFET Drive Circuit 12
Figure 4. pH Monitoring Circuit 13
Figure 5. Conductivity Measurement Circuit 15
Figure 6. Axis Definitions in the Sensor Ball 17
Figure 7. Accelerometer Circuitry 17
Figure 8. Connections to the RS-232 Driver Circuit 21
Figure 9. Battery Terminal Inputs and Battery Voltage Monitor 22
Figure 10. Voltage Regulator Circuits with Voltage Monitors 22
Figure 11. Flowchart for the Sensor Ball Firmware, Main Module 24
Figure 12. Sensor Ball View Showing the Sensor-Protecting Cage 31
Figure 13. P&G Staff Evaluating Sensor Ball Characteristics 38
Figure 14. Component Detail, Sensor Ball Exploded View 70
Figure 15. Exploded View of Disassembled Unit 72
Figure 16. Assembled Unit 72

Tables

Table 1. Data List 8
Table 2. Sensor Ball to PC Connection Cable Pin Definitions 23
Table 3. Directly Wired Connections 69
Table 4. Test Points 69
Table 5. Sensor Ball Mechanical Bill of Materials 71

Development of an *In-Situ* Monitoring / Recording Device for Aqueous Processes

Introduction

The *In-Situ* Monitoring/Recording Device for Aqueous Processes will provide The Procter & Gamble Company (P&G) with non-intrusive instrumentation that will accelerate in-house product development. The device, developed and fabricated under CRADA SC00/01585, records chemical and physical parameters during normal washing machine conditions. The data will assist P&G in evaluating consumer practices and the impact of those practices on product performance.

In order to stay competitive in today's market, P&G must develop better test methods to shorten the development time of new technologies. One way to accomplish this is to develop "smart" sensors that communicate what is occurring in consumers' homes. For example, by monitoring parameters such as wash temperature, pH, conductivity, turbidity, water hardness, available chlorine, available oxygen, product consumption, and washing machine agitation through a wash cycle, a more comprehensive understanding of consumer laundry practices and the impact of these practices on product performance can be developed.

The technological basis for this project is the earth penetrator instrumentation developed by Sandia National Laboratories (Sandia). This instrumentation has aspects similar to the requirements of this project, however, none of those previous instrumentation designs monitored aqueous chemistry. The P&G application is essentially a reformatting of the earth penetrator instrumentation concept with new features added to acquire additional measurements and somewhat different approaches for memory management. This project provided proof of concept for an autonomous monitoring device that can measure and record certain chemical and physical parameters. The work may lead to additional cooperation to develop novel chemical sensors that could have application to national security issues.

Sensor Ball Features

The instrumentation developed, the Sensor Ball, measures wash water pH, conductivity, temperature, and agitation (3-axis acceleration). The watertight device is approximately a sphere three inches in diameter weighing about half a pound – essentially the weight of water displaced by the ball. The shape and size are very similar to Procter & Gamble's Downey fabric softener dispenser ball. By monitoring the conductivity measurement once every minute, the Sensor Ball automatically determines when it is immersed in water and should begin collecting data. Once a data collection cycle starts, measurements are recorded about 2.6 times per second for 45 minutes. The data are stored in non-volatile, flash memory with a 32M-byte capacity. Each data collection cycle uses 128k-bytes memory, so the battery life is the limiting factor on how much data can be collected. Depending on battery life, the Sensor Ball should be able to collect about 54 data cycles over a two-week period, considerably beyond the minimum 20 data cycles defined in the Sensor Ball requirements shown in Appendix B.

Following the data collection period in a test-consumer's home, the Sensor Ball is returned to P&G. Simple disassembly allows access to an interface connector to upload data and to replace the battery. After removing a single security screw, the top half of the Lexan case can be unscrewed. Either a fresh battery must be installed or a separate supply must be connected to power the Sensor Ball for data uploading. Data are transferred serially using a cable between the Sensor Ball and a personal computer. To facilitate data extraction, a user interface program was developed that can upload the data, check current status of Sensor Ball parameters, and erase the memory in preparation for another data collection period. Following reassembly, the Sensor Ball is ready to return to the field to repeat another data collection period.

The design was intended for use in top-loading washing machines. The mechanical environment in front-load washing machines includes high shock impulses when a solid object strikes the lifting bars in the machine, and high

acceleration forces during the spin cycle. These environments exceed the design goals for this version of Sensor Ball. In addition, the conductivity and pH sensors require continuous immersion in water for valid measurements. A Sensor Ball design compatible with a front-load washing machine must abate the repetitious, high impact shock and provide some mechanism for stable water chemistry measurements given intermittent immersion.

Measurement Data List

The Sensor Ball data list with channel numbers and names, resolution, sample rate, and nominal calibration factors is shown in Table 1 below. In addition to acceleration, pH, conductivity, and temperature, the Sensor Ball data includes measurements and other information to help validate and interpret data. For example, voltage monitors indicate if the 3V and 6V reference voltages are stable, which may indicate the quality of the measurements. The battery voltage can be used to monitor battery performance and indicate potential circuit failures as revealed by increased power consumption. Record number and Frame number help separate data sets. The Time Most-Significant Word (MSW) and Least-Significant Word (LSW) are included to provide a rough time measure. This will assist data interpretation by noting when a data set was collected. Time measurement inaccuracy occurs because the watchdog timer delay is not exactly 2.4 seconds, and because the microcontroller oscillator frequency is not exactly 3.686 MHz. In addition, the time measurement is reset whenever a microcontroller reset occurs. Finally, the Flash memory address was included to help diagnose Flash memory and related problems.

Table 1. Data List

Chan	Title	Bits	Data Rate (sps)	Calibration Factor Slope	Offset
1	X Axis Acceleration	12	2.655	0.03212 G/Cnt	2048
2	Y Axis Acceleration	12	2.655	0.03212 G/Cnt	2048
3	Z Axis Acceleration	12	2.655	0.03212 G/Cnt	2048
4	pH	12	2.655	-0.00205 pH/Cnt	5350
5	Conductivity	12	2.655	3.495 uS/Cnt	1010
6	Temperature	12	2.655	0.09318 °C/Cnt	2932
7	6V Voltage Monitor	12	2.655	0.00354 V/Cnt	0.0
8	3V Voltage Monitor	12	2.655	0.00354 V/Cnt	0.0
9	Battery Monitor	12	2.655	0.00354 V/Cnt	0.0
10 msb	Record Number	8	0.095	1	0
10 lsb	Frame Number	8	0.095	1	0
11	Time MSW	16	0.095	72.83 seconds/Cnt	0
12	Time LSW	16	0.095	1.111 ms/Cnt	0
13	Memory Address	16	0.095	1	0

Note: the above table uses the equation Engineering Units = Slope * (Counts – Offset)

Battery Operating Time

The hardware and firmware for the Sensor Ball have been designed to maximize battery life. To minimize power consumption, the Sensor Ball remains in a low power, "sleep" mode unless collecting data. In this state the Sensor Ball draws less than one-half milliamp. Once a minute, the Sensor Ball powers all circuitry and checks whether the conductivity measurement is above a set threshold, about 6% of full-scale range. If this threshold is exceeded, then the Sensor Ball remains fully powered and begins a 45-minute data collection cycle, filling 128k-bytes memory. During full-power operation, the Sensor Ball draws about 25mA. Following the data collection cycle, the conductivity measurement is checked again, and if less than the threshold the Sensor Ball returns to the low-power state.

To achieve more than the minimum 20 data cycles over a two-week period, care must be taken when selecting a battery. The capacity of a standard, 9-V alkaline battery is about 600 mA-Hr. This is barely adequate to meet the desired duration of data collection. The appropriate battery, the Energizer Model L522 for example, has about a 1200-mA-Hr capacity and uses a lithium / manganese dioxide chemistry. Calculations below show that with this capacity battery, the Sensor Ball should be capable of acquiring about 54 data collection cycles over a two-week period. Similar calculations for an alkaline, 600-mA-Hr battery reveals that only about 23 records can be collected

using that type of battery. Actual battery performance is a function of the current draw among other factors, so these calculations are estimates only.

Low Power Sensor Ball Operation:
0.5 mA * 14 days * 24 hours/day = 168 mA-Hr

Full Power Sensor Ball Operation:
25 mA * 45 min * 54 cycles / 60 min/hr = 1013 mA-Hr

Total Battery Capacity Needed:
168 mA-Hr + 1013 mA-Hr = 1181 mA-Hr

Because the Sensor Ball uses non-volatile memory to hold the collected data, battery failure or battery voltage dropping below the minimum 6.5 V electronics operating voltage does not result in lost data.

Project Formation

P&G determined that a miniaturized data-logging device meeting their needs was not commercially available, so Sandia's assistance was sought to develop this device. The background on P&G initial involvement and their design concept are detailed in a P&G document included in Appendix A. Although P&G wanted a device that would measure more than temperature, pH, conductivity, and acceleration, the features of the device proposed for development were reduced to this set to demonstrate proof of concept for the approach. Further, sensors for some of P&G's measurements of interest were not available at all, or at least not in the size needed for the proposed instrumentation to be practical.

CRADA Timeline

Discussions regarding the creation of the recording device began May 1999, when P&G requested project timing and funding proposal from Sandia. This was followed by a P&G visit to Sandia on 25 June 1999 to discuss proposal options. As a result of those discussions, a CRADA was executed 11 January 2000 with P&G that provided $225k funding from P&G. This was a "funds-in" CRADA, with no funding contribution by the Department of Energy. The first joint P&G and Sandia project meeting, which began the detailed design requirements negotiation, occurred February 8, 2000.

Because the project start was delayed somewhat waiting for funds transfer, P&G requested a CRADA amendment in April 2000 that shortened the delivery time of the first prototype by two months but increased the cost by about $30k. This was intended to compensate in part for the month-long delay from CRADA execution until receipt of the initial funds at Sandia.

Sandia delivered the first Sensor Ball prototype to P&G in September 2000. After P&G tested this prototype, Sandia implemented substantial revisions in the Sensor Ball's mechanical, electronic, and firmware design, resulting in the Version 2 design that was delivered to P&G in December 2000. Following additional test experience with the Version 2 prototype, P&G requested further improvements to the Sensor Ball functionality that required minor electronic and substantial firmware changes. This Version 3 design was delivered in June 2001. The final prototype design, Version 4, required additional electronic modifications and minor firmware changes. Design Versions 2, 3, and 4 used the same circuit board design, but with some signal re-routing and component changes as a result of the circuit changes. The board was not redesigned to minimize costs. Five copies of prototype Version 4 were delivered to P&G in January 2002. To accommodate building these additional prototype units, P&G requested a second CRADA amendment that increased funding by $50k. Two time-only CRADA extensions were also executed to provide P&G additional evaluation time before finalizing requirements for the Version 4 design.

Requirements Summary

The final design requirements are detailed in a document prepared for P&G by Sandia, which is included in Appendix B. In addition to defining the basic parameters to be measured, requirements negotiations covered issues like the impact resistance of the completed Sensor Ball, the basic shape and chemical properties of the case, measurement ranges, and operating requirements. The requirements document also discussed human interface details such as how the Sensor Ball would indicate various operating modes. Deliverables were specified to be a

prototype Sensor Ball, interface cable, user interface software, and documentation. P&G intended to seek a manufacturing entity to produce the Sensor Ball in quantity after the concept was adequately proven. At P&G's request, Sandia produced five additional prototype units to facilitate concept evaluation.

The final form of the Sensor Ball is shown in Figure 1. The design is ballasted to maintain orientation with the chemical sensors on the bottom, where they will remain immersed. The cage protects the pH and conductivity transducers from damage by clothing in the washing machine, but was primarily intended to protect these transducers in the event the Sensor Ball is accidentally dropped. After a tamper-prevention fastener is removed, the top half of the Sensor Ball can be unscrewed to access the battery, serial interface connector, and "Attention" button. Only P&G personnel are intended to access these, not the consumer product tester.

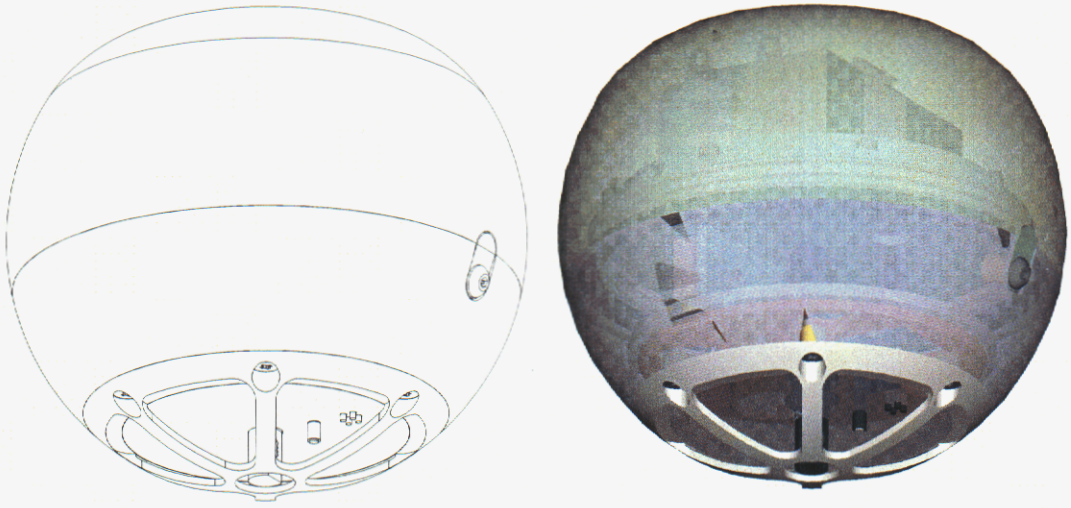


Figure 1. Wire-Frame and Solid Views of the Sensor Ball

Measurement Transducers and Circuitry

To measure the four parameters specified in the requirements, the Sensor Ball included four types of transducers. First, a Honeywell DuraFET pH sensor combined with a Microelectrodes Inc. pH reference was used to measure pH. Microelectrodes, Inc. also provided the conductivity probe, which consists of an array of four, platinum pins. The three-axis acceleration measurement relied on the recently developed, integrated accelerometers produced by Analog Devices. Finally, a semiconductor temperature transducer manufactured by Analog Devices was selected to simplify circuit design and data interpretation. The sensors represent a significant fraction of the Sensor Ball cost, totaling about $600 per Sensor Ball in small quantities. Each of the transducers is described in detail below.

pH Sensor

The wash water pH significantly affects product performance. The measurement range in the Sensor Ball is pH 5 to 11. The pH sensor was selected by P&G based upon their laboratory experience. The Honeywell DuraFET pH sensor is a robust process-monitoring probe that uses a field-effect transistor (FET) as the monitoring element for hydrogen ion concentration. The probe was available from Honeywell only as a complete assembly in the process-monitoring configuration. This seven-inch long cylinder required modification at Sandia to fit within the tennis-ball sized Sensor Ball.

pH Probe Details

Details of the probe construction is proprietary information that Honeywell was not inclined to share. From our observations and Honeywell (formerly Leeds and Northrup) conference papers[1], the pH probe consists of a quarter-inch diameter FET silicon window embedded in a Ryton cylinder. The cylinder side opposite to the FET contains a

[1] J. G. Connery, R.D. Baxter, and C. W. Gulczynski, "Development and Performance Characteristics of a New pH Electrode", 1992 Pittsburgh Conference, 10 March 1992

conductive Ryton plug. The Ryton is conductive to provide the counter-electrode connection to the liquid being monitored. Figure 2 shows a diagram of the pH sensor construction.

The DuraFET is part of a class of ion-selective field-effect transistor, or ISFET, devices. The substrate is a p-channel, enhancement-mode FET, with the gate insulator exposed to the solution to be monitored. The recommended operating mode is to maintain a constant drain-source current through the DuraFET by driving the potential of the solution with respect to the DuraFET with a counter-electrode. For the Honeywell device, the counter electrode is a region of conductive plastic on the probe body (the Ryton cylinder). With the voltage sensed at the source of the DuraFET as the negative feedback, an amplifier circuit modifies the solution's potential to keep the DuraFET drain current constant. Then, the overall potential of the solution is sensed using a silver / silver chloride reference electrode. The Honeywell conference paper indicated that the potential on the DuraFET gate decreases about 60 mV for every one-unit increase in pH.

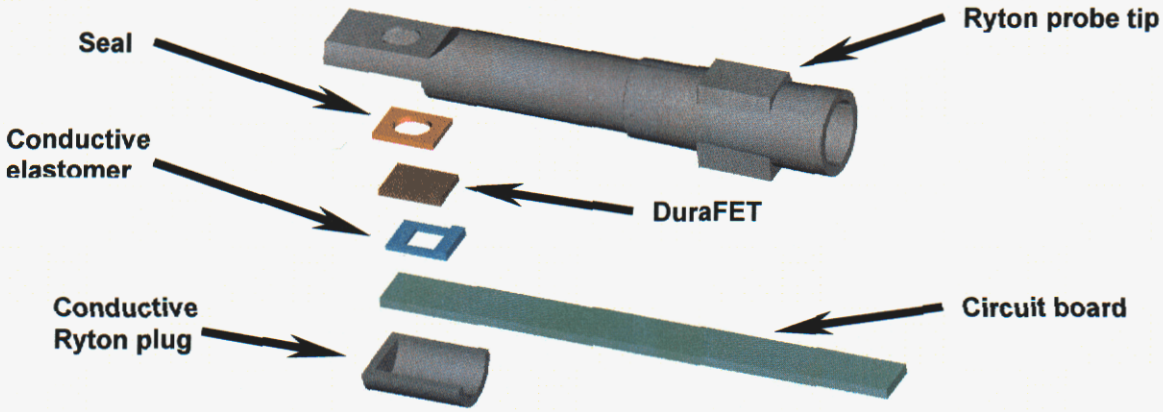


Figure 2. Construction of the Honeywell pH Probe Tip (figure has errors and needs text)

A Honeywell Model 51204976-002 pH probe was used[2], which is configured without an integral reference electrode but has the same outer diameter as that type. Modifications to the Honeywell probe included sawing the probe apart, leaving about a one-inch section of Ryton tube holding the DuraFET. In the first attempts to modify the probe, the original thin circuit board that made electrical contact to the DuraFET was replaced with a Sandia-designed version. A conductive elastomer, present in the original and retained in the modified unit, makes the final connection between the circuit board and the DuraFET. A small gasket seals the DuraFET to the Ryton body. During the step replacing the original board with the Sandia version, the gasket seal was broken, and the probe leaked. The procedure was modified to leave the original Honeywell board in place, and then making the electrical connection to the Sensor Ball by soldering fine wires to the circuit traces on that board. Modifying the probe is a very time-consuming step. When considering higher-volume production versions of the Sensor Ball, some agreement with Honeywell should be attempted to procure the probe tip or some variation configured specifically for the Sensor Ball.

Ag-AgCl Reference Electrode

Although the probe-only configuration was used for the Sensor Ball, the pH sensor from Honeywell can be supplied with an integral reference electrode configured as an annulus around the probe body. The annulus is a passage into the probe body, which contains a potassium-chloride gel saturated with silver chloride into which a silver wire is immersed. The reference electrode is an attractive feature of the Honeywell probe, but unfortunately it could not be used when the probe was modified to fit the Sensor Ball. Instead, the reference electrode function was filled by a slightly modified version of the model MI-402 Dip-type Reference microelectrode from Microelectrodes Incorporated[3]. This electrode, a 2-mm outside-diameter flexible tube closed on the immersed side with a glass frit,

[2] http://content.honeywell.com/sensing/control/pdf/sales_lit/70-82-58-66.pdf, page 3, column 1.
[3] http://www.microelectrodes.com/Products/MI_402.htm The pH reference electrode cost about $124 each when first ordered June 2000.

is filled with a 3 molar KCl gel solution saturated with AgCl. The silver contact wire that is immersed in the KCl gel is soldered directly to the main Sensor Ball circuit board.

One problem with the reference electrode is that it provides a passage to the inside of the Sensor Ball. When a pressure differential of more than a few pounds / square inch occurs between the interior and exterior of the Sensor Ball, the reference electrode solution is likely to leak by moving towards the lower pressure. This should not be a problem at the bottom of a two-foot column of water (as in a washing machine) because the pressure at that depth is less than 1 psi. Similarly, atmospheric pressure variations due to weather should be less than 3%, or about 0.5-psi variation. However, we found that a problem does occur when the Sensor Ball is assembled at one altitude, and then shipped to a significantly different altitude. The elevation at Sandia in Albuquerque is higher than 5000 feet, while P&G's Cincinnati location is about 500 feet, a differential of about 2 psi. When shipped via air, a pressurized shipping compartment may drop to about 11-psi absolute pressure, whereas an un-pressurized compartment would be about 3 psi absolute. This pressure-induced leakage problem was resolved by shipping the Sensor Ball partially disassembled so that no pressure differential occurs across the reference electrode.

pH Sensor Electronics Interface

Figure 3 and Figure 4 show the DuraFET drive circuit and the pH-sensing circuit, respectively, used in the Sensor Ball. Rather than use a switch-mode power converter to produce a negative supply voltage, the reference voltage for the pH circuit is shifted from ground (zero volts) to 3 V. The DuraFET source is connected to terminal X2, and the drain at X4, where the signal connects to ground through resistor R24. The DuraFET substrate is tied to the 3-V reference voltage at terminal X5. The non-inverting input of the OPA4343 operational amplifier, component number U1D, is connected to the 3-V reference voltage. The inverting input monitors the voltage at the DuraFET source, with current to the source provided through resistor R16 from the 6-V supply. Approximately 0.3 mA current flows through the DuraFET device. The output of operational amplifier U1D drives the counter electrode, with the voltage increasing as the current through the DuraFET decreases.

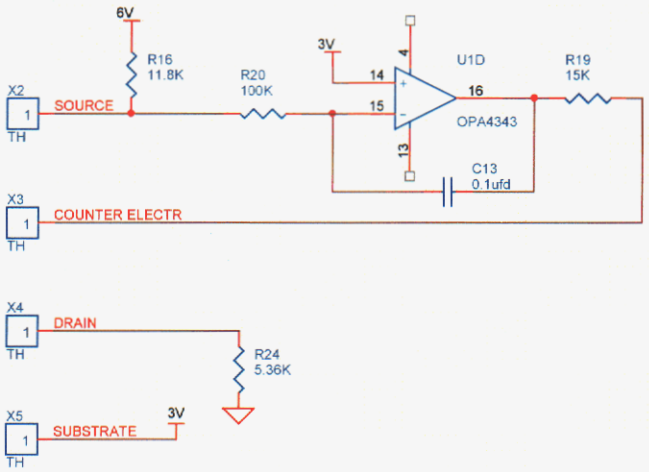


Figure 3. pH DuraFET Drive Circuit

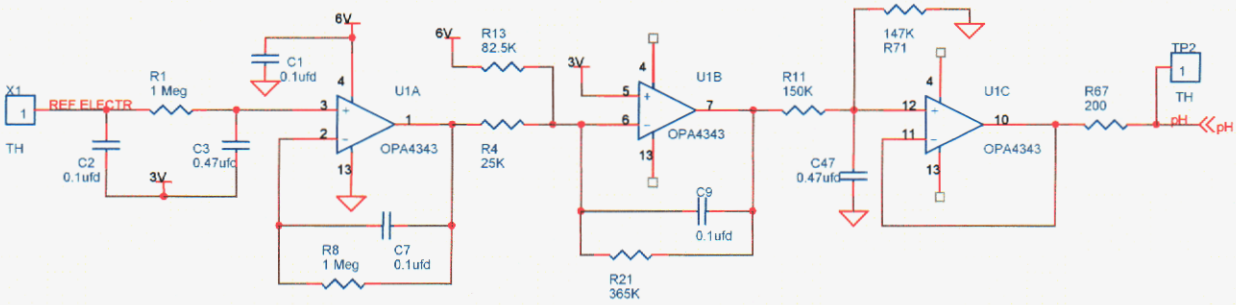


Figure 4. pH Monitoring Circuit

The solution potential, which is proportional to pH, is measured by the microcontroller after signal conditioning by operational amplifiers U1A, B, and C. The first stage using U1A is a unity-gain buffer to accommodate the high-impedance input signal from the reference electrode. The second stage using operational amplifier U1B corrects for the circuit reference voltage shifted to 3 V from ground, allows offset adjustment via R13, and signal amplification adjustment using R4. The nominal gain is about 15 (negative), which is then divided approximately in half by the subsequent resistor-divider network formed by R11 and R71. The subsequent operational amplifier U1C acts as a unity-gain buffer that supplies the signal to the analog-to-digital converter (ADC) included in the microcontroller. The intended range is pH 5 to pH 11.

pH Sensor Electronics Design Issues

Late in the evaluation of the Sensor Ball, we discovered that the pH voltage supplied to the microcontroller ADC exceeded 3 V, especially when the pH probe was not immersed. This caused problems as described later in the "Electronics Design" section, under "RS-232 Interface". To ensure that the voltage supplied to the microcontroller does not exceed 3 V, all of the needed gain was implemented in the first amplification stage using operational amplifier U1B. Its output cannot exceed the 6-V supply voltage. Then, dividing that intermediate signal by slightly more than two using a resistor-divider ensures a maximum final output voltage less than 3 V.

A problem with the delay between valid pH measurements and pH circuitry power-up was only partially resolved. Whenever analog power is removed from the Sensor Ball circuitry, a transient error occurs in subsequent pH measurements. The time before the error is negligible seems to increase with longer power-off time. After a one-minute low-power period, the transient usually lasts two to three minutes, while after an hour with power off, the delay may be five minutes. The problem was corrected partially in the Version 4 Sensor Ball firmware by eliminating the inter-record, one-minute delays, thus removing the discontinuity during a period of interest. The problem will still exist whenever the Sensor Ball goes from a low-power, shutdown state to a data collection mode. However, this delay is considered tolerable because the reference electrode is not completely functional for a few minutes after it has been left in air long enough for the tip to dry.

Based on observations during P&G tests, the delay until valid pH measurements was not seen with the Version 2 Sensor Ball. Three changes were made subsequent to the Version 2 Sensor Ball design. First, the microcontroller code was extensively revised. Changes in the way the microcontroller acquires data or does some other related function could have been a cause. Second, the specific pH probe and reference electrode were replaced. There could be some variation in the performance of individual units. And third, the process for modifying the pH probe for use in the Sensor Ball changed. We determined that the microcontroller was not a possible cause by creating a test setup that isolated the pH probe, reference electrode, and associated signal conditioning circuitry with the microcontroller excluded. Using a voltmeter to monitor the conditioning circuit output, we observed the delay whenever power was restored to the circuit. The discontinuity did not occur when the Sensor Ball remained powered, but was moved among various pH buffer solutions, demonstrating that the circuit response time was adequate. Therefore we concluded that the issue could be pH probe variation, or perhaps differences between the original Honeywell connection board and the Sandia version.

Conductivity Probe

The conductivity measurement provides an indication of ionic content, which in turn can indicate the water hardness. The calcium and magnesium ions that cause hard water combine with the product and deactivate it. Generally, if the concentration of ions is doubled, the conductivity is doubled, or in the case of magnesium and calcium ions, quadrupled because each has an ionic charge of two. The conductivity of a material is an inherent property - that is; pure water at a particular temperature will always have the same conductivity. The SI unit of conductance is the Siemen (S) or Ω^{-1}. This unit has also been known as Ohm spelled backwards, or mho. Conductivity is measured in units of Siemens / m. The Sensor Ball conductivity measurement appears to be linear in the zero to 3000-μS/cm range, which is the expected conductivity range in a consumer's wash. The maximum conductivity that the circuit can report is about 10000-μS/cm.

A secondary use of the conductivity measurement is to initiate a data collection cycle. The design originally used a tilt-switch to detect placement in the washing machine, and automatically initiate data collection. The switch was deleted because it contained mercury, a potentially hazardous material when included in a consumer appliance. Although non-mercury tilt switches are available, the units are much larger. In any case, using the conductivity measurement is an elegant approach to sense Sensor Ball placement in the washing machine. The design was revised so that the Sensor Ball periodically "wakes up" and checks the conductivity reading. The measurement is essentially zero when the probe is in air. If the value indicates immersion in water, a data collection cycle begins. The cost of this approach is slightly higher power drain, resulting in about a 10% reduction in the potential recording time of the Sensor Ball.

The Sensor Ball conductivity probe uses a two-electrode arrangement. The electrodes are 1 mm diameter, platinum rods coated with colloidal platinum, or platinum black, and obtained from Microelectrodes, Inc[4]. The coating is intended to absorb gas products resulting from water electrolysis. Current is sensed from a fixed-amplitude, square-wave voltage that is applied across the probe pair. The probe voltage is coupled using a capacitor to avoid electrode polarization and electrolytic build-up on the electrodes that in turn would affect measurement accuracy. An equally important consideration is to avoid affecting the solution potential as detected for the pH measurement. The capacitors block direct current, and thus do not cause a bias shift in solution potential. To create an alternating current, the conductivity probe voltage is pulsed at 7.2 kHz.

Explanation of Cell Constant's Relationship to Conductivity Measurements

The cell constant, K, for a conductivity measurement device is related to the area of an electrode and the distance between electrodes in the cell. K is defined for two flat, parallel measuring electrodes as the electrode separation distance divided by the electrode area. We need to be concerned about the cell constant when using a 2-electrode conductivity cell, as the response becomes non-linear outside of the electrode working range. For wash solutions, a cell constant of 1 cm^{-1} is sufficient. This cell constant provides a working range of zero to 2000 μS/cm. When the expected solution conductivity is large, for example 8,000 μS/cm, a larger cell constant such as 10 cm^{-1} is necessary. A four-electrode conductivity cell with a single cell constant can cover large concentration ranges from 0.1 to 10,000 μS/cm that would normally take three ordinary 2-electrode cells with three different cell constants. In a four-electrode cell, two electrodes drive current and two electrodes sense voltage. An alternating voltage is applied across the drive electrodes, resulting in an alternating current flow that is measured at the sensing electrodes. The voltage measured at the sensing electrode controls the amplitude of an alternating voltage applied across the drive electrodes; hence, the cell constant is continually being adjusted depending on the solution being analyzed and the cell field strength is maintained constant. The current flow at the **drive** electrodes is directly proportional to conductivity.

Conductivity Circuitry

The circuitry takes a root-mean-square measurement of the alternating current flowing through the wash water at the conductivity electrodes. The current is proportional to the solution conductance. Figure 5 shows the conductivity circuit described below.

[4] http://www.microelectrodes.com A four-electrode set of conductivity probes cost about $225 from Microelectrodes Inc. when first ordered June 2000.

To generate the alternating voltage applied to the conductivity electrodes, the Sensor Ball divides the approximately 1-MHz "Instruction Clock" produced by the microcontroller by 128 using CMOS counter U10, resulting in a 7.2-kHz square-wave signal. (The instruction clock is actually closer to 920 kHz.) Resistors R48 and R50 attenuate the 3V-clock signal, and operational amplifier U6D buffers the resulting 1-V square wave for application to the conductivity electrode via 10µF capacitor C44. This electrode is connected at X11. Matching 10µF capacitor C51 connects the second electrode at X10 to current sensing resistor R52, which is in turn connected to signal return. The alternating voltage across the resistor is proportional to the current, and also proportional to the conductivity. The conductivity probes connected to X12 and X13 are not used. They were originally included to accommodate a 4-probe conductivity measurement.

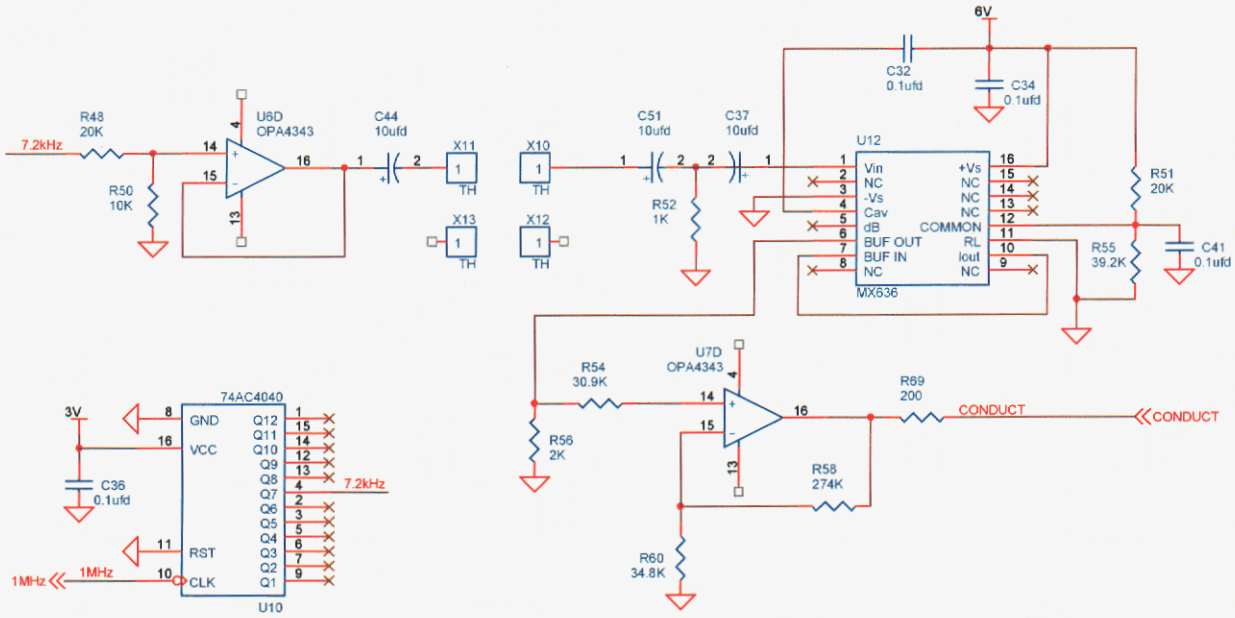


Figure 5. Conductivity Measurement Circuit

To convert the alternating voltage into a steady value acceptable to the analog-to-digital converter, a specialized component, Maxim part MX636 (component U12) calculates the root-mean-square voltage. The MX636 data sheet[5] describes a configuration for single-supply operation. Using the recommended configuration, the equivalent supply voltages become +2 V / -4 V, and the maximum input signal is ±2 V. Because the voltage driving the conductivity electrode is ±0.5 V, this maximum input signal value cannot be exceeded. Therefore, the output of the MX636 is at most 0.5V. The gain circuit including operational amplifier U7D amplifies that output by about 8, so a 4 V maximum signal would be produced with zero resistance between the conductivity electrodes. Although this would exceed the 3-V maximum, zero resistance is unlikely in normal Sensor Ball operation.

A Four-Electrode Design Option

With four electrodes, the sensing current for the conductivity measurement can be adjusted to compensate for electrode fouling. Essentially, the current is adjusted across the outer two electrodes to maintain a constant potential between the inner pair. The resulting current is proportional to the conductivity. Unfortunately, the circuitry to implement the four-probe approach could not readily be implemented to meet the size and power constraints of the Sensor Ball. Instead, the two-electrode arrangement described above was used that places a fixed voltage potential across the pair and measures the current.

To implement the 4-probe conductivity design, the drive voltage amplitude would be continuously adjusted to maintain a constant potential across the sensing electrode pair. This could be accomplished by amplifying the root-mean-square value (or absolute value) of the sensing electrode pair potential, and chopping that signal through a transistor before connecting it to the driven electrode. The 7.2kHz signal that directly drove the existing 2-electrode

[5] http://pdfserv.maxim-ic.com/arpdf/MX536A-MX636.pdf, page 8.

circuit, or a separate oscillator, would provide the transistor-switching signal. The resulting drive current indicates the conductivity.

If power consumption and component count is an issue, an alternative two-electrode measurement could be developed that applies a voltage pulse to the electrode pair and measures the current. Using a 10μF capacitor produces a pulse time constant of many tens to hundreds of milliseconds, depending on the conductivity. This time constant would produce less than one bit error in the 12-bit measurement as a result of signal droop in the 10μs required by the ADC to acquire the measurement. An advantage to using a pulse is reduced interference of the pH measurement, fewer components, and lower power consumption. But a disadvantage is further difficulty converting to a four-electrode measurement, should one be required in the future.

Accelerometers

The accelerometer measurement indicates the degree of agitation during the wash cycle. This may imply how tightly packed the clothes are in the wash. The measurement was implemented using three, single-axis accelerometers. The acceleration range on each axis is +/-50 g. Although the accelerometers could have measured the spin cycle acceleration, which implies the residual moisture in the clothes, the spin-cycle acceleration in a typical washing machine will exceed 100 g and is therefore beyond the range of the accelerometers used. However, duration of the spin cycle is available from the measurement.

Tri-axial acceleration measurements were selected because the orientation of the Sensor Ball to the acceleration vector is unpredictable, and an acceleration magnitude is desired. P&G laboratory measurements included a significant amount of data on load size and relative rate of agitation based on single-axis vibration measurements. But tri-axial acceleration measurements were expected to provide more insight than single-axis acceleration into how the clothing tumbles, bends and moves during the wash. The additional circuit board space and higher power consumption were considered when making this design trade-off. P&G's standard single-axis laboratory vibration measurements employed a piezoelectric crystal sensor that has no DC response. The transducer used in the Sensor Ball, however, can measure constant acceleration.

Integrated Accelerometer and Signal Conditioning

The acceleration transducer used, an Analog Devices model ADXL150[6], is a third-generation, ±50 g surface-micro-machined accelerometer. The ADXL150 is a single-axis accelerometer with signal conditioning included on a single monolithic integrated circuit. The sensitive axis of the ADXL150 is in the same plane as the silicon chip. Thus, producing a 3-axis measurement requires a second circuit board oriented orthogonally to the primary board. In the Sensor Ball, the x-axis accelerometer is located on the main circuit board while the y- and z-axis accelerometers are placed on the secondary board. In Figure 6 below, the circuit boards are shown in green.

[6] http://www.analog.com/productSelection/pdf/ADXL150_250_0.pdf,

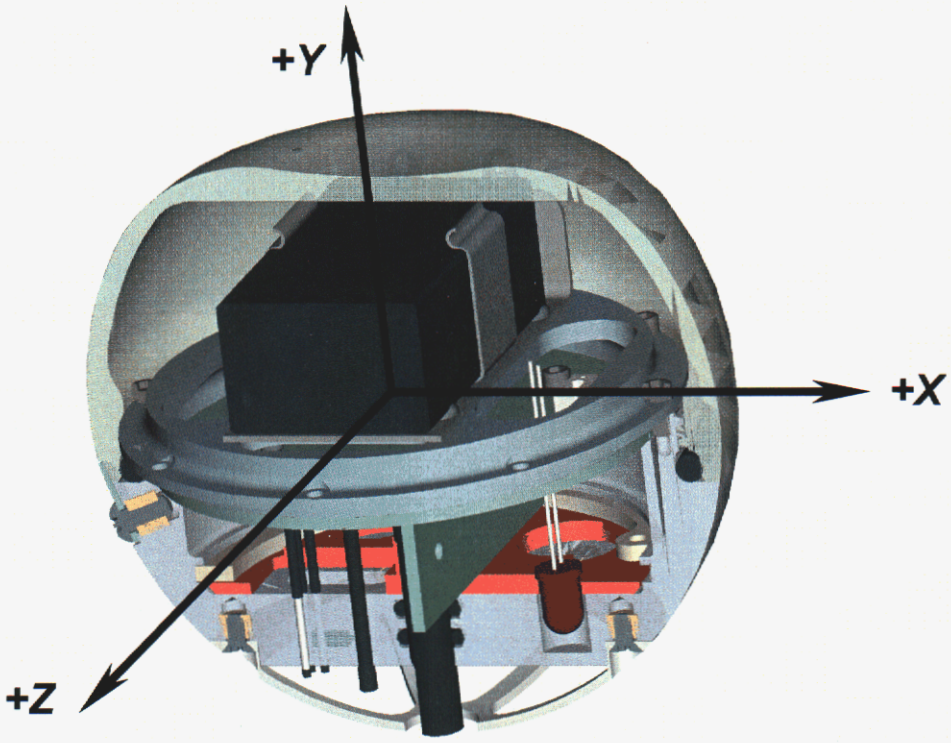


Figure 6. Axis Definitions in the Sensor Ball

Typical signal-to-noise ratio for the ADLX150 is 80 dB, allowing resolution of signals as low as 0.01 g within a ±50 g full-scale range. However, the signal conditioning gain selected and the 12-bit resolution of the analog-to-digital converter in the Sensor Ball limits resolution to about 0.03 g. The device scale factor is 38 mV/g when a 5-V supply voltage is used. Because the scale factor and zero-g output are ratiometric with the supply voltage, the Sensor Ball scale factor is 46 mV/g. The supply voltage in the Sensor Ball is 6 V. Zero g drift is 0.4 g over 0°C to 70°C, and power consumption is 1.8 mA.

Accelerometer Circuit

The accelerometer output signal can range from 0.25 V to 5.75 V, while the microcontroller's internal analog-to-digital converter is limited to a zero to 3 V input range. The ADLX150 accelerometer produces a reference signal output that is essentially half the supply voltage. As shown in Figure 7, operational amplifier component U6A takes the difference between the accelerometer reference signal and the acceleration measurement signal, and then reduces both the zero acceleration output and the measurement by 50% to ensure that the 3-V limit on the microcontroller is not exceeded. The intent of the circuit is to center the ADC input at 1.5 V to get the maximum dynamic range.

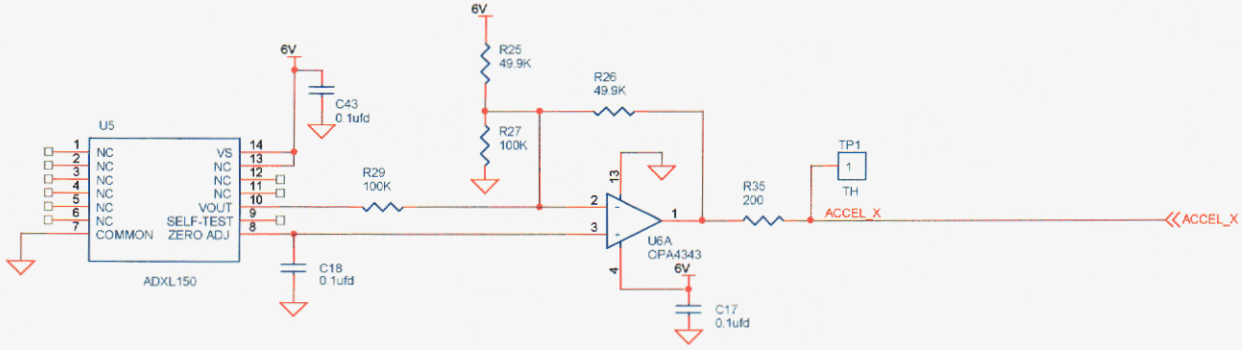


Figure 7. Accelerometer Circuitry

Temperature Transducer

Temperature is an important factor in product performance. The Sensor Ball uses an Analog Devices model AD590 temperature transducer, a two-terminal integrated circuit that produces an output current proportional to absolute temperature. The Sensor Ball's temperature measurement maximum is 100°C.

Active Device for Temperature Measurement

The AD590[7] acts as an ideal high-impedance, constant-current source passing current proportional to temperature, 1 μA/K. The device is trimmed by the manufacturer to calibrate the device to produce 273.2 μA output at 273.2K, which is 0°C. The AD590 uses a fundamental property of silicon to realize its temperature proportional characteristic. This relationship is if two identical transistors are operated at a constant ratio of collector current densities, r, then the difference in their base-emitter voltage will be:

$$V = \frac{kT}{q}\ln(r)$$

Since both k, Boltzman's constant and q, the charge of an electron, are constant, the resulting voltage is directly proportional to absolute temperature. The AD590 was chosen instead of a thermistor because of its inherent linear response to temperature, but it is much more expensive. A thermistor may be desirable to reduce component costs of a production design.

Thermal Transport

The location of the sensor was moved from the main circuit board, where it was initially placed for easy assembly in Version 1, to a machined pocket inside the Lexan case. This provided faster response to temperature changes. Limited modeling of the Sensor Ball temperature measurement response time guided the selection of the revised temperature sensor location. The issue is how fast heat can move from one material to the next. The modeling showed that placing the AD590 in a milled pocket of the Lexan case should work well, and would not be much different than direct contact with the water. Lexan wall thickness around 0.05 to 0.10 inches was expected to be acceptable. However, the contact between the temperature sensor and the Lexan must be tight. For comparison, a 0.001-inch air gap between the AD590 and the Lexan produces a heat flow resistance factor of 2.0, while the 0.050 inch thick Lexan has a resistance factor of 0.2. The resistance is directly proportional to the thickness, so would double for double the thickness. For this reason, a thermal compound should be considered to improve the contact of the AD590 to the Lexan. The Sensor Ball assembly included epoxy to provide good thermal contact between the AD590 and the Lexan wall. The response time may still be inadequate; see the "Suggested Changes" section.

Interestingly, a major factor for affecting the temperature measurement response time may be the conduction from the water to the Lexan. Different flow rates of water past the ball can account for two orders of magnitude difference in the heat flow resistance factor. A resistance factor of 17 was used for model calculations, and equates to a low liquid flow such as would be likely with the sensor floating in sloshing water. But if laboratory measurements monitor temperature in a rapidly flowing environment, such as the temperature sensor on a pump line, the response time would be much faster when compared to the Sensor Ball.

Temperature Circuit

The temperature circuit consists of the AD590 device and a 7.87kΩ resistor, R28, as shown in Appendix E: Schematics. The 6-V supply voltage biases the AD590. Since the AD590 acts as an ideal current source, the voltage across R28 is proportional to the current, and also the temperature. The resistor value was selected to produce a 3-V output at about 100°C. Although at the expense of slightly more complex circuitry, higher temperature measurement resolution could be achieved with an operational amplifier circuit that inserts a voltage offset so that 0°C produces a zero volt output.

Electronic Design

The Sensor Ball electronic design can be divided into a few functional areas. The transducer interface and signal conditioning functions were described above with each transducer used. These circuit elements, and the other

[7] http://www.analog.com/productSelection/pdf/1186_b.pdf

electronic components can be lumped into the category, "Hardware". The remaining Sensor Ball functionality is achieved using the algorithms encoded in the microcontroller, which is the "Firmware" category. Details for each are described in these sections below.

Hardware

A microcontroller incorporates the intelligence needed to control recording data from these transducers, and the user interface to extract the data. The algorithms for this functionality are described later in the "Firmware" section. The recorded data are held in a non-volatile, Flash memory. An RS232 interface component facilitates uploading the data to a computer. Linear regulators reduce the 9-V nominal battery voltage to levels acceptable to the circuitry, and permit portions of the design to be powered only as needed to conserve power. The complete schematics for the Sensor Ball are included in Appendix E: Schematics.

Microcontroller

The Sensor Ball operations are controlled by a Microchip PIC16C774[8] microcontroller, an 8-bit, reduced-instruction-set device designed to consume very little power. The microcontroller has a built-in oscillator circuit with the frequency set by an external resistor and capacitor. The Sensor Ball oscillator frequency, 3.686 MHz, was selected to be an integral multiple of the 19.2k Baud and 115.2k Baud serial communication rates. The frequency is set by R46 and C40. A universal, synchronous / asynchronous receiver / transmitter (USART) in the microcontroller facilitates serial communication. The PIC16C774 also includes a 10-channel, 12-bit Analog-to-Digital Converter (ADC). This is used to digitize all the measurements recorded by the Sensor Ball. To reduce power consumption even further, the low-power "Sleep" mode is used while waiting to initiate a data collection cycle. In this mode, a separate internal oscillator measures a 2.4 second (nominal) delay period, during which the microprocessor is virtually powered down. Following this "Sleep" period, normal processing continues. This delay timer function also is the basis for a Watch-Dog Timer, used to reset the microprocessor in the event the device stops normal operations. The microcontroller code is written in assembly language using Microchip's instruction set, and developed using MPLAB Version 5.40. The code is divided into six main routines plus three support files that reserve memory for program variables, define equivalence statements, and store message text. The code is described in the Sensor Ball Code section.

The inputs to the microcontroller include six analog sensor data channels (ACCEL X, ACCEL Y, ACCEL Z, pH, CONDUCT, TEMP); three analog voltage monitor channels (3V MON, 6V MON, 9V MON); a pushbutton depressed signal (ATTENTION); and a serial receive input (RX). The outputs from the microcontroller include LED control (LED); power control (POWER ON); memory interface (I/O1-I/O8, CLE, ALE, WP~, R/B~, RE~, CE~, WE~); and a serial transmit output (TX).

Flash Memory

Design requirements stated that sufficient memory capacity was needed to record 20 data measurement cycles, each cycle 45 minutes in duration. Further, the memory should be non-volatile so that battery power is not necessary to retain the information. The requirement translates into a 4 M byte minimum memory size. A Toshiba TC58256FT Flash memory device was selected. The device is a 3.3V, 32 M byte Electrically Erasable and Programmable Read-Only Memory (EEPROM) organized as 2048 blocks of 32 pages, each page containing 528 bytes (512 bytes plus extra 16 bytes extended area). For the Sensor Ball application, only 512 out of the 528 bytes are used on each memory page. The device has a 528-byte static register that acts as a buffer for programming functions. Data are written to the buffer and then transferred to the non-volatile memory when commanded.

The Toshiba TC58256FT is a serial-type memory that uses eight I/O pins for command and address input as well as data input / output. A typical command consists of a command byte followed by three address bytes. Once a read or write command is executed, the device can continue to read or write 528 bytes without an additional command step. In addition to read and write, the memory commands are Program, which transfers the 528-Byte static register to the memory cell array; Erase followed by Erase Confirm, which erases a 16k-Byte memory block; and Status, to determine whether a programming function has completed and report reading or writing malfunctions. The Program operation can take a maximum of 1 millisecond, and the Erase operation a maximum of 20 milliseconds.

[8] http://www.microchip.com/1000/pline/picmicro/families/16c77x/devices/16c774/index.htm

According to a note on Toshiba's data sheet, the TC58256FT memory is not guaranteed to have the entire 2048 blocks of 16k byte each available to write. Some blocks may contain malfunctioning memory cells and be unavailable for use. For this reason, the Sensor Ball code was revised in Version 3 to detect bad memory blocks, and compensate for them by skipping over them. The Sensor Ball uses 128k bytes to record 45 minutes of data, so if a 16k block were lost it would represent over five minutes of data. The down side of this feature is somewhat more complex code.

During the Flash memory erase function, the memory is tested superficially and an attempt made to mark bad blocks. During data storage, the first bytes of each new block are examined, and the entire block is skipped if the bytes are not in the erased state, a value of FF hexadecimal. The first bytes written to each 512-byte page during recording is a 16-bit synchronization pattern. Then when data are uploaded, the memory is first checked to see if the memory section is blank, or was marked as malfunctioning. If so, the data are skipped and the next memory section is read. If no more data are available during data upload, the Sensor Ball begins uploading zeroes for the data. In Version 2, the same situation would have resulted in apparently full-scale data since erased memory bytes contain a value of FF hexadecimal. A more extensive Flash memory test is included in the Sensor Ball firmware, but is not implemented in the SensorBall PC interface software. The extensive test takes about 30 minutes.

Adequate delay time must be provided between powering the Flash memory and first access, otherwise the memory will appear to be malfunctioning. About a quarter-second delay is currently implemented in the Version 4 Sensor Ball firmware.

Although the Sensor Ball data are recorded in 512-byte increments, this is expanded to 560 bytes during data upload to a personal computer. The expansion occurs because the variables for frame number, record number, time, and Flash memory address are stored only once for every 28 samples of the other analog channels, but must be expanded to create a symmetrical data structure that meets the Inter-Range Instrumentation Group Standard 106. This standard is followed in all instrumentation designs at Sandia, and allows previously developed software to be used for support functions like data uploading and plotting. When the data are uploaded, seven groups of 80 bytes are formed for each 560-byte "Block" uploaded. Within the 80 bytes are 4 sets of measurements: 9 channels, two bytes per channel, for 18 bytes per set and 72 bytes total. The remaining 8 bytes in the 80-byte group contain the frame number, record number, time (four bytes), and Flash memory address (2 bytes).

RS-232 Interface

The serial interface designed into the Sensor Ball allows data to be rapidly and easily extracted following a data collection cycle. A three-wire cable connects the Sensor Ball to a personal computer COM port. Special software on the personal computer sends commands to the Sensor Ball and receives data returned from the Sensor Ball. Two data rates are supported: a default speed of 19.2k Baud, plus a 115.2k Baud high-speed that can be selected to reduce the time needed to upload data. Checksums are used to ensure data integrity. A problem with the link was discovered and corrected as one of the final design changes.

The RS-232 communication standard specifies voltage levels for signaling a "1" or "0" bit. Because these voltages are outside the range of the Sensor Ball supply voltages, a Maxim MX3221 device (component U11) is used to translate signal levels from the zero to 3 V level of the microcontroller to RS-232 levels. Only conductors for transmit, receive, and ground signals are connected. The MAX3221's internal power supply consists of a regulated dual charge pump that provides output voltages of +5.5V (doubling charge pump) and -5.5V (inverting charge pump) even though the Sensor Ball provides only +3V to the device. The supply voltage range for the device is +3.0V to +5.5V. Each charge pump requires a flying capacitor (C38, C39) and a reservoir capacitor (C33, C42) to generate the V+ and V- supplies. The level translators that convert CMOS-logic levels to RS-232 levels guarantee a 120k bit per second data rate with worst-case loads of 3k ohms in parallel with 1000pF, providing compatibility with PC-to-PC communication software. Figure 8 shows the connections used for the Sensor Ball. The three-pin Molex connector, U3, provides a direct RS-232 communication connection to a PC.

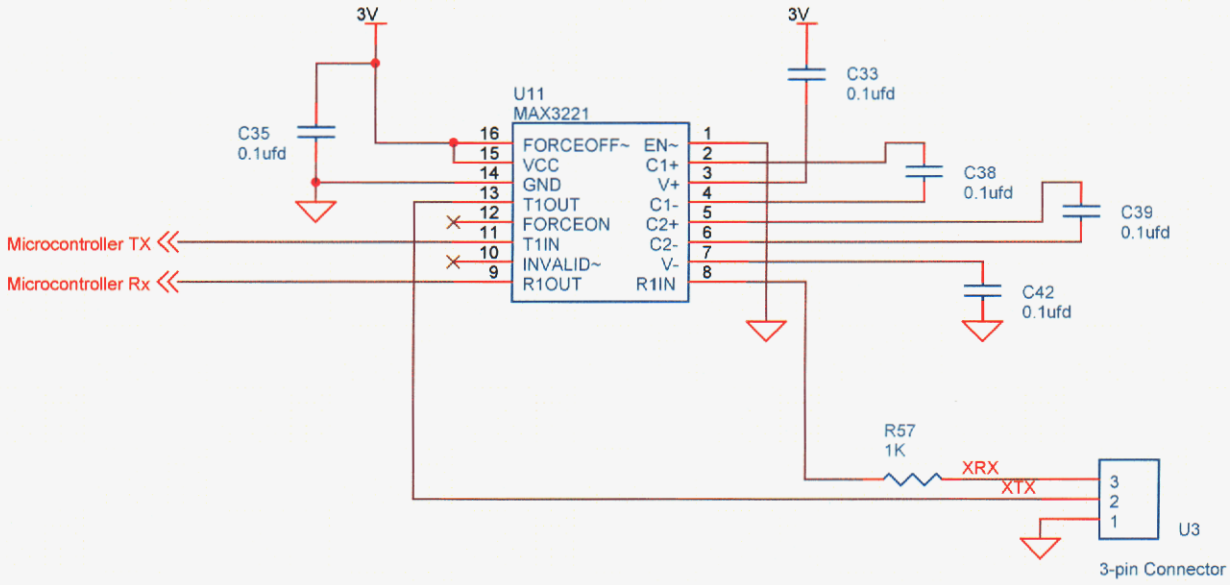


Figure 8. Connections to the RS-232 Driver Circuit

Although the PIC16C774 processor can support a maximum serial data rate of 230.4k Baud, the Maxim part limits the communication speed to the next lower standard speed of 115.2k Baud. This speed should be adequate for the Sensor Ball, since at this rate 4M bytes of memory can be uploaded in about 6 minutes. Using the slower, but more robust, 19.2k Baud rate would take 36 minutes to upload the same 4M Bytes. Each symbol exchanged consists of 8 data bits plus one start and one stop bit.

To ensure consistent communication performance with Baud rates matching standard speeds, the microcontroller's RC oscillator must have less than 3% error from 3.6864 MHz. In the design problem that affected communication, the microprocessor supply voltage shifted from 3V to nearly 5V. (The pH measurement output into the microcontroller's analog-to-digital converter exceeded the 3V supply voltage, effectively powering the microcontroller at a higher voltage.) This voltage shift also changed the microprocessor's oscillator frequency, and thus also changed the communication clock frequency. RS-232 transmits least- to most-significant bits. If the frequency shift is sufficient, the higher-order bits will not be sampled at the proper time and the symbol sent will be significantly misinterpreted.

Minimizing power consumption in the Sensor Ball design is very important to maximize recording time. Although the microprocessor receives power whenever a battery is installed, power to the RS-232 driver chip, the Flash memory, and the analog circuitry is switched on and off under program control. But even though power is applied to the RS-232 chip during the data recording process, the MAX3221 draws only 1μA supply current in this mode because of a special feature. If the device does not sense a valid signal level on their receiver inputs, the power supply and drivers on board the device shut down. This would be the situation when the Sensor Ball is collecting data with the RS-232 cable disconnected, the normal mode of operation. The RS-232 chip turns on again when a valid signal level is applied to the RS-232 receiver input.

Linear Regulators and Voltage Monitors

A commercial 9V battery supplies the raw power and is wired directly to the PCB at X14 and X15 as shown in Figure 9. The battery voltage monitor circuit reduces the voltage to be within the maximum 3-V for the microcontroller's analog-to-digital converter. The diode, D1, is intended to prevent damage to the Sensor Ball in the event the battery connection polarity is accidentally reversed.

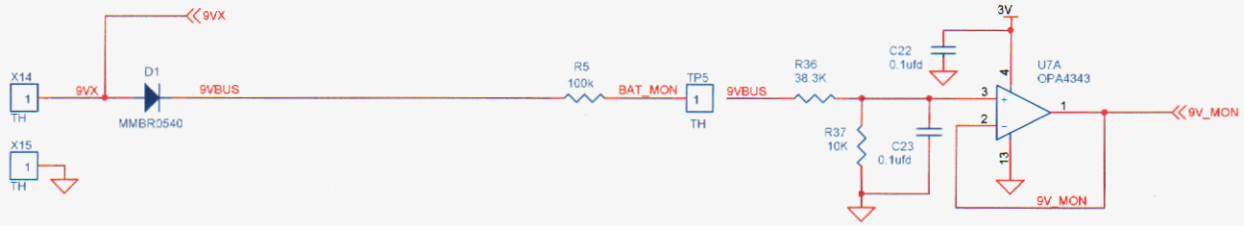


Figure 9. Battery Terminal Inputs and Battery Voltage Monitor

The conditioned power to run the system is supplied by three, low-dropout linear regulators, U13, U2, and U3 shown in Figure 10. With the battery connected, U13 is continuously powered and supplies 3V to the microcontroller on the VDD net. U2 and U4 remain powered down until the POWER_ON line is driven high by the microcontroller. When low, the POWER_ON line inhibits U2 and U3 through those devices' shutdown function. VDD powers only the microcontroller, while the 3V and 6V regulated voltages supply all other devices. When the microprocessor is in the low-power mode, less than 0.5 mA is required, as compared to about 25 mA when the Sensor Ball is fully powered.

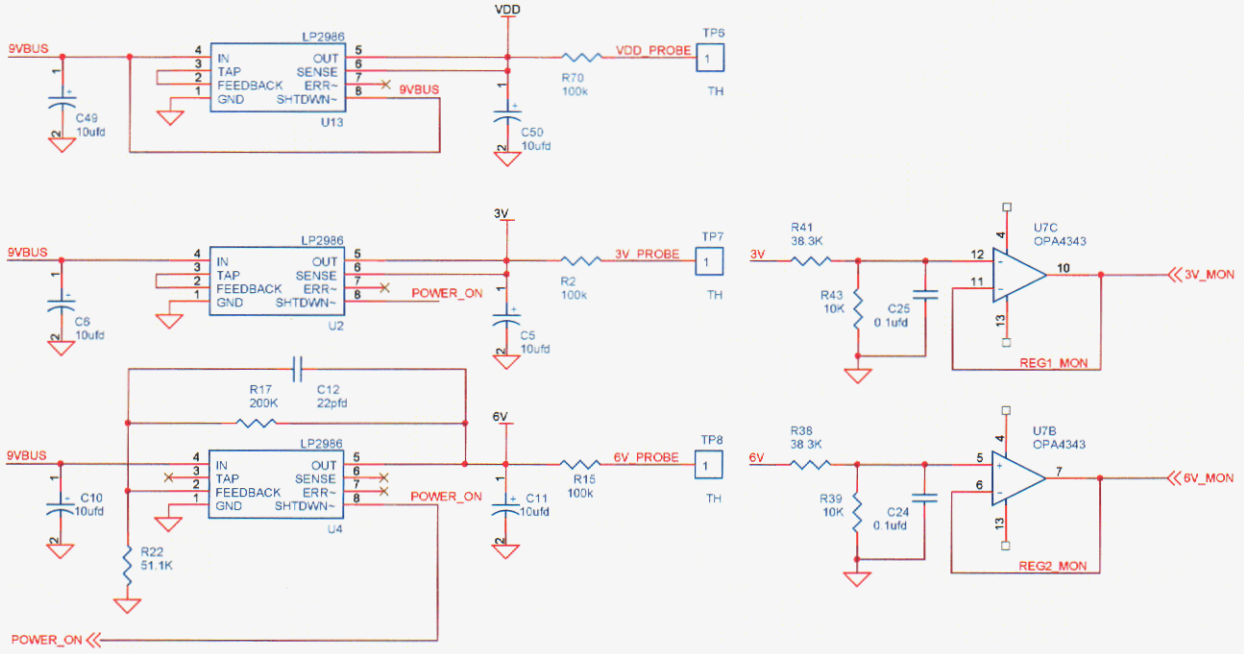


Figure 10. Voltage Regulator Circuits with Voltage Monitors

Operational amplifiers U7A, B, and C buffer the battery voltage, the regulated 6 V supply, and the regulated 3 V supply, respectively. The monitor circuits are identical and produce identical scale factors, about 21% of the monitored voltage. This voltage divider brings the monitored level within the range of the analog-to-digital converter.

Circuit Board Details

The Sensor Ball circuitry is contained on two printed-circuit boards (PCBs), named the Controller Board and the Signal Conditioning Board. The Controller Board, R62111, is constructed as a circular disk, is populated on both sides with components, and contains six circuit planes. The top plane includes surface mount integrated circuits, capacitors, resistors, a 3 pin Molex connector, and a push button switch. The switch, SW1, provides the user with control over the mode of operation of the Sensor Ball. An LED is directly wired to the PCB at X8 and X9 but is mounted at the bottom of the Sensor Ball. The LED provides an external indication of the status of the Sensor Ball. The bottom plane is populated with surface-mount passive components. The internal planes are from top down:

signal trace, ground reference, 3-V supply (analog only), and 6-V supply. The 3-V Vdd supply connects only to the microcontroller so is not allocated a routing plane.

The rectangular Signal Conditioning Board, R62110, is populated on both sides and contains four circuit planes. The top plane includes the accelerometer integrated circuits. The ground reference plane is the next layer under the top plane, followed by the 6-V supply plane, and then the bottom signal-routing plane. The bottom plane is populated with operational amplifiers and passive components for signal conditioning.

The two circuit boards are wired together at through-hole terminations F1 through F4. Terminations F1 and F2 provide the power and ground, respectively, to the Signal Conditioning board, while F3 and F4 are the Y- and Z-axis acceleration levels measured and conditioned on the Signal Conditioning board. Details showing the circuit plane layout and component placement for each board is contained in Appendix F: Assembly Drawings and Electronics Materials Lists.

Communication Cable

To extract data and perform other maintenance functions on the Sensor Ball, the communication cable is required. It is the only cable used with the Sensor Ball. One end of the cable attaches to the keyed, 3-pin connector on the Signal Conditioning board, and the other connects to COM1 on a personal computer. The 54-inch cable has a resin-filled fiberglass sheath protecting the wiring, which is firmly attached to the terminating connectors using cable lacing and hard epoxy. Definitions for the signals and pin numbers at each connector are listed in Table 2 below.

Table 2. Sensor Ball to PC Connection Cable Pin Definitions

Signal	Sensor Ball, 3-Pin Molex (U3)	Personal Computer, 9-Pin D Subminiature
Signal Return (Ground)	1	5
Transmit at SB, Receive at PC	2	2
Receive at SB, Transmit at PC	3	3

Microcontroller Firmware

The microcontroller code is almost 4k Words in size, nearly filling nearly all the available program memory space. Because the microcontroller is a one-time programmable device, any changes to the code will require replacement of the device. The code is divided into six modules plus three auxiliary files. A complete listing of the code is contained in Appendix K: Microcontroller Code Listing. The content and function of each of the modules is described here.

Main Code Module, Ball.asm

Ball.asm is the main section of code that controls the operation of the Sensor Ball. It provides four main functions: microcontroller initialization on reset, a loop monitoring the conductivity measurement, data acquisition and storage into memory, and a user-interface loop that communicates with the PC-based interface software.

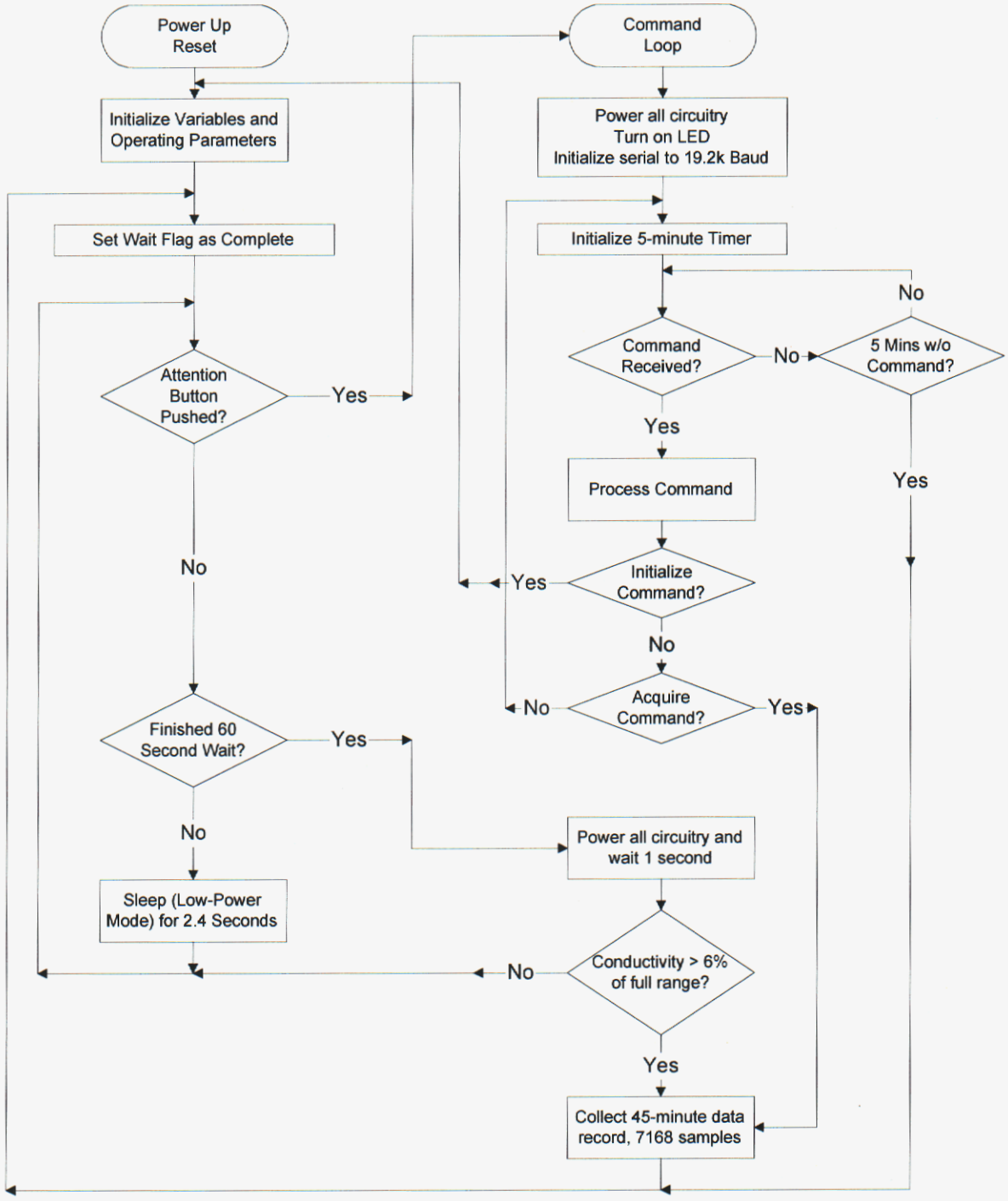


Figure 11. Flowchart for the Sensor Ball Firmware, Main Module

The initialization code sequence contained in Ball.asm executes whenever the Sensor Ball is reset by any of: initial power application, the initialize command from the PC interface software, or an interrupt from the Watch-Dog Timer. The latter would only occur in an error condition such as if the microcontroller was not executing the proper code or if it takes longer than was anticipated to compete a function. The first step in the initialization sequence is configuring input and output parallel ports. Next the serial port and A/D module are initialized. The outputs are then set to the desired level for operation. A short phrase, the Sensor Ball version and unit number, is transmitted on the serial interface as a debug test.

After initialization, the code then enters the main operating loop. For the majority of the time in this loop, the microcontroller is operating in a low-power state and other circuitry is not powered. Immediately upon entering this loop and every sixty seconds thereafter, the microcontroller powers up the rest of the Sensor Ball and interrogates the conductivity sensor. If the conductivity is above a preset threshold (about 6% of full-scale range), the recording process begins and one record is collected. If the threshold is not reached, then the microcontroller returns to the low-power loop.

A flaw in the firmware design causes the Sensor Ball to begin the recording process whenever the battery is initially connected. This happens because after power-up initialization, only a quarter-second delay occurs between power application to the conductivity circuitry and checking its output. From circuit tests, the conductivity circuit requires a half-second power-up delay, and the normal conductivity check loop waits over one second before acquiring the conductivity measurement. A simple movement of the starting label for the conductivity checks process would correct the problem. The work-around is to press and hold the "Attention" button until the LED remains on. This will terminate the data collection cycle.

Delay Timing Loop of Ball.asm

Two types of timing are accomplished in the microcontroller. The first, a watchdog timer, is based upon a special, internal oscillator with a frequency set at the time of manufacture. This timer controls the time that the Sensor Ball is in the low-power, "Sleep" mode. The second is based upon the instruction cycle clock, which is derived from an oscillator that is programmed with an external resistor and capacitor. This timer is used to establish the delay between sets of measurements during data collection and perform other delays as needed in the code. The time variable created to provide a pseudo-real-time clock is dependent upon both timers.

From the microcontroller data sheet[9], the typical value for the Watch-Dog Timer timeout period (Twdt) ranges from 7 to 33 milliseconds, with 18 milliseconds typical. This is measured with VDD = 5V, and over the temperature range -40°C to +85°C with the typical value measured at 25°C. However, Microchip states that these parameters are for design guidance only and are not tested. For the Sensor Ball, the PIC16C774 microcontroller is powered with a 3-V supply voltage. The delay seems to be very close to the typical, 18 millisecond value in the Sensor Ball. The Sensor Ball firmware applies a 1:128 scale factor on the watchdog timer delay, resulting in an incremental timing value of about 2.3 seconds for the "Sleep" delay loop. Thus, twenty-six passes through the loop are required to accumulate the desired 60-second delay between conductivity measurement checks. Once the Sensor Ball is operating in full-power mode, the instruction clock timer is used.

Resistor R46, 7.87kΩ and capacitor C40, 22pF set the instruction clock frequency. The desired frequency is 3.686MHz, which was selected to be an integral multiple of the 19.2k Baud rate. This frequency can vary somewhat, but should not be more than 3% from the desired target or serial communication will be affected. This primary oscillator frequency is divided by four to produce the instruction clock frequency. Then, a 1:4 scale value is applied to form the basis for Timer 1. Each increment of the Timer 1 clock represents about 1.11 milliseconds, and this is also the increment of the Time Least Significant Word that is stored with the recorded data. The Time value is adjusted each time the WAIT delay subroutine is called.

Data Collection Section of Ball.asm

If the conductivity check indicates that the Sensor Ball is immersed in water, that is, that the conductivity measurement is above a set threshold, a data collection cycle will begin. The data collection code loop controls the Analog-to-Digital Converter (ADC), reads the ADC result, and stores the information into the Flash memory. The microcontroller then waits a short delay before acquiring another data set. Pressing and holding the Attention button, accessible only when the Sensor Ball is opened, can halt the data collection cycle.

First, the Flash memory is interrogated to verify that the location pointed to by the current memory address value is actually blank. This value is cleared when the microcontroller is initialized, but a search routine find the first blank

[9] http://www.microchip.com/1000/pline/picmicro/families/16c77x/devices/16c774/index.htm PIC16C773/4 Datasheet, Table 15-7: Reset, Watchdog Timer, Oscillator Start-Up Timer, Power-Up Timer, And Brown-Out Reset Requirements, page 163.

section where data has not been stored. If the microcontroller has not been re-initialized since the previous data collection cycle, the memory address value will already point to the next usable Flash memory location. If the memory is full (unlikely since the 32 MB can contain 256 records), the data collection loop ends and he microcontroller returns to the low-power state.

Because the basic unit in the Flash memory is a 512-byte page, data are stored in this increment. The first two bytes on the page form a synchronization word that indicates the page contains valid data. This is followed by six bytes containing the record number, the frame number, and four bytes representing the time. The remaining 504 bytes hold 28 sets of measurements, with each set consisting of 9, 16-bit measurement values. Table 1 lists the data collected.

After collecting each complete set of 9 values, the microcontroller pauses for about 368 milliseconds (data collection takes about 9 milliseconds), producing a data collection frequency of 2.66 samples per second. Collection proceeds for about 45 minutes, filling 128 kB memory, and then control is returned to the main loop starting with an immediate check of the conductivity measurement. If the conductivity is still above the set threshold, the data collection loop is immediately re-entered. Otherwise, the low-power loop begins, with the conductivity subsequently checked every 60 seconds.

Command Processing Section of Ball.asm

The command processing section of the Ball.asm module is a simple loop that waits until the Sensor Ball receives a complete command sequence, then calls the command processing routine, Cmd_Hand.asm. The characters are captured by an interrupt routine contained in the Ser_Hand.asm module, which also provides the indication that a complete command is ready for processing. The only other function of the command loop is to monitor how long ago the last command was entered and return to the low-power mode if more than five minutes have elapsed since the last command. This ensures that a spurious "Attention" signal won't halt the Sensor Ball for too long. The command mode can be entered at any time by pressing and holding the Attention button. Response times vary depending upon the Sensor Ball's operating mode before the Attention button was pressed. In the low power, "Sleep" mode, response is within 3 seconds. At the other extreme, the response may be as long as 11 seconds if the Sensor Ball is acquiring data.

Serial Interface Module, Ser_Hand.asm

The serial interface module contains all code associated with the serial interface. It includes the interrupt service routine that captures and processes control characters, and that captures a command phrase and indicates to the main routine that a command sequence is complete. Also included are interface configuration, character and string transmit routines.

The Sensor Ball's microcontroller is configured to generate an interrupt whenever a character is received on the serial interface, and code execution changes to the interrupt service routine. This routine includes hardware error processing to clear buffer overflow or framing errors. The Sensor Ball expects serial data using the format of one start bit, eight data bits, and one stop bit. Because the least-significant bit is transmitted first, timing errors receiving the serial data affect the most significant bit to the greatest extent. Either 19.2k Baud or 115.2k Baud may be used, but the Sensor Ball must be commanded to expect the higher data rate.

In the ASCII character definition, control characters have numeric values of hexadecimal 1F or lower. The Sensor Ball recognizes only eight control characters: Control-C, line feed, carriage return, X-off, X-on, Acknowledge, and Negative Acknowledge. If a control character other than these eight is received it is ignored. Control-C stops any command phrase in progress and is also used to abort the data upload process. Line feed and carriage return are used to terminate a command phrase, and may occur in either order. X-on is used to signal the Sensor Ball to begin transmitting data during a Status command or data upload. X-on also clears X-off, which may be sent by the PC to temporarily suspend data transfer if the X-on / X-off protocol is used. The Acknowledge and Negative Acknowledge are used during data upload, at the end of each 560-byte block that is transferred. The Sensor Ball calculates a 16-bit checksum for each block, and transmits it when finished sending the block. The PC does a similar calculation, and transmits an Acknowledge control character if the checksum calculated by the PC matches what the Sensor Ball sent. Otherwise, the PC returns a Negative Acknowledge control character, and the Sensor Ball retransmits the data block.

A command phrase consists of the following character sequence:

1. A "0", the zero character, must be sent first by the PC interface.
2. An alphabetic command character (A through Z), either upper or lower case, follows.
3. Optionally, any parameters can follow the command character. These characters must have numeric equivalents of hexadecimal 20 or greater; otherwise they would be interpreted as a control character.
4. Finally, carriage return and line feed are sent in either order.

Following receipt of a complete command phrase, the Sensor ball echoes the phrase back to the PC. This provides a check to ensure the command and its optional parameters were received properly. If the command is echoed properly, the PC sends a command phrase including the "V" command character, indicating the expected command phrase was echoed. In response to this command phrase, the Sensor Ball echoes the previous command phrase, with the parameters processed so that the upper and lower nibbles are combined into one character. Because control characters cannot be included as parameter values, the parameters are encoded with only the least significant nibble in each parameter character used to represent a value. The nibbles from two sequential characters are required to represent an eight-bit value. Therefore, the parameters echoed back to the PC in this stage can include control characters. Following this second echo of the command phrase with processed parameters, the Sensor Ball processes the command. If the phrase initially echoed to the PC is incorrect, sending Control-C control character, or just repeating the intended command can cancel the incorrectly received command. If the Sensor Ball receives an incorrect command sequence, it replies with an error phrase: "0?x", where "x" represents the incorrect character.

Once a command phrase is complete and verified, the interrupt routine sets a flag and returns control to the previously running code, usually the command processing section of the main code module Ball.asm.

Text String and Character Transmit and Serial Interface Configuration Routines

The serial interface code module Ser_Hand also contains other related routines such subroutines to retrieve text strings, to transmit a character, and to configure the interface.

Text strings are recovered using a code "trick" that loads the starting location of the string into the program counter, and jumps to that location. This program location contains a return instruction with the desired character loaded in the working register. Each character of the string is recovered in turn by incrementing the program address location that is jumped to. This character is passed to the serial transmission subroutine, which checks to see if the serial output buffer is full, and moves the character to the buffer if it is empty. If the register is full, the serial transmission routine loops through its instructions to wait until the buffer is empty. When a single character is transmitted, it is simply loaded into the working register followed by a call to the serial transmission subroutine.

The serial interface configuration function was placed in a subroutine because it is called during power up / reset initialization, whenever the "Attention" button is pressed, and in response to the PC interface Baud Rate command. The routine selects the Baud rate based upon the value passed in the working register: a zero indicates 19.2k Baud, and a non-zero value 115.2k Baud.

PC Interface Command Processing Module, Cmd_Hand.asm

The Sensor Ball recognizes seventeen commands. After the serial interface module sets the command received flag, the command processing loop in the main code module Ball.asm calls the command handler subroutine. This is the only routine in the Cmd_Hand.asm module. Using a look-up table, the command handler subroutine matches the received command character with the associated instructions. Following execution of these instructions, the command handler subroutine returns control back the command-processing loop in the main code module. The actions taken by the six commands available through the PC interface software, "SensorBall", are described in Appendix C. The Read and Status command instructions are contained in separate code modules, Rd_Hand.asm and St_Hand.asm, respectively. These routines are described later in later sections. Each of the other commands is described below.

The Acquire command associated with command character "A" sets a flag that results in program control jumping to the data recording instructions. Control will not be returned to the command-processing loop after the data record is collected. Pressing the "Attention" button is required to allow other PC interface commands to be processed. The

action taken by the Acquire command is the same as if the Sensor Ball detected a conductivity measurement above threshold in normal operation.

Command character "B" is associated with the Baud-rate change instructions. The default data rate is 19.2k Baud, and the high-speed rate is 115.2k Baud. The current data transmission rate is determined from the configuration registers, and the rate is toggled to the other value. The change is accomplished by calling a subroutine in the Ser_Hand.asm code module.

The Sensor Ball's Flash memory is erased using the "E" erase command. The entire 32M-byte memory is erased by this command.

Command "F", memory fill, is not available using the Sensorball PC interface software, but requires HyperTerminal or another similar software package that allows direct control of the PC's COM port. This command is intended for testing Sensor Ball operation. Two parameters following the command determine the number of 128k-byte Flash memory records to be filled with values. The default, if no parameters are attached, is one record.

A PC interface session can be terminated using the "I" initialize command. The result is equivalent to removing power from the Sensor Ball. All variables are cleared, including those that track time. In the initialization sequence, the Flash memory is inspected to determine the last record number and memory location used. The "P" command, included as a legacy command, results in the same actions. The PC interface uses only the "I" command.

The Data Read command "R" is implemented in the Rd_Hand routine, described below.

Sensor Ball status is returned in response to the "S" command. This function is implemented in the St_Hand routine, which is described below.

The "T" test command is also unavailable using the Sensorball PC interface software. It is intended to be used during Sensor Ball check out. Two tests are run using two different data values in which the entire Flash memory is filled with the data value and read back to ensure the memory is operating properly. The routine attempts to mark any bad memory by writing the incorrect synchronization pattern into the start of the memory block. The test takes a little more than half an hour to complete. An error value is returned for each 1M-byte memory section tested.

Several lower-case commands in addition to the "r" and "s" commands described above are implemented as a first step towards standardizing the interface to Sandia's memory-based instrumentation systems. The only command in this group that is used by the Sensorball PC interface software is "w", which requests the device name and unit number. The Sensor Ball returns the phrase, "Sensor Ball Version 4 Unit nnn", where "nnn" represents the unit number. The complete list of these extended commands with their associated function is as follows:

b – replies with the high-speed data transfer rate represented in ASCII characters
c – lists the upper-case commands recognized by the Sensor Ball, "ABEFIPRST"
m – send the number of each type of channel: analog, bilevel, and self-test
n – transmit the name for the channel specified
r – as discussed above, uploads the contents of the Sensor Ball memory without checking for blanks, etc.
s – reports the status of each data channel in data frame order
w – described above, identifies the unit as a Sensor Ball with the version and unit number
z – replies with the total memory size and the upload data block size

Flash Memory Control Module, Flash.asm

The Flash memory module contains all the functions needed to read, write, and erase the Toshiba TC58256FT Flash memory. Using the functions instead of in-line code or macros conserved program memory space and made the code somewhat easier to read and maintain.

The Flash memory uses the same eight connections to accept a command, an address, and read or write data. The code functions were organized around the memory commands, and support the following commands:

0x00 Read Mode 1, Address bit A8=0. Allows sequential reads through a 512-byte memory page.
0x80 Write with Address bit A8=0, writes to the 528-Byte static register.

0x10 Program, transfers the 528-Byte static register to the memory cell array
0x60 Erase, erases a 16k-Byte memory block. Must be followed by Erase Confirm
0xD0 Erase confirm command
0x70 Status, provides Ready / Busy~, and write fail.

When transferring measurements to the Flash memory, the code must keep track of how much data has been written. After a 512-byte memory page is filled, a separate programming command must be issued to transfer the data to non-volatile storage. The erase command also requires two steps, but this combination is designed to reduce the change that a spurious command will accidentally erase the memory.

After a program or erase command, the memory device will be busy for a period and should not be interrupted with additional commands or data. A special memory status inquiry function was used to monitor the status. The status can also be ascertained by monitoring the Ready / Busy line on the memory chip. However, that occupies an additional input connection on the microcontroller. The memory datasheet notes that up to 1 millisecond is required for the program function, and as much as 20 milliseconds for the erase function. A read access requires 10 microseconds maximum, but that delay was achieved by padding the code with NOP (no operation) instructions.

Because the TC58256FT memory is not guaranteed to have the entire 2048 Blocks available to write, functions were included in the Sensor Ball firmware to detect a memory failure and hopefully avoid it. The status command returns an error indication along with the Ready / Busy information. If any write or erase operation returns an error condition, the code attempts to mark the memory block as bad. When that block is subsequently selected for writing or reading, the code skips ahead to the next functional memory block.

Two functions are included in the memory subroutines to assist testing the memory. Neither are accessible using the SensorBall personal computer interface software. The first, the memory fill command, "F", creates simulated data and writes it to memory. This can be used to generate data records to test the serial interface. Otherwise, uploading blank memory is not particularly taxing since the data are all zero. The second command, "T", or memory test, writes values to the memory then reads back and compares the read value with the intended value. The test is conducted page-by-page. First, the memory is totally erased. Then, pattern one is written to memory, which is a value of hexadecimal 0xAA. After the memory is filled, the memory is erased and a second value, hexadecimal 0x55, is written. Finally, the memory is erased again. Any errors are reported as messages using the serial interface. The memory test function takes about 30 minutes to complete.

Data Upload Module, Rd_Hand.asm

The Rd_Hand.asm routine is called from Cmd_Hand when the "R" or "r" command is processed. This routine reads the contents of the Flash memory and transmits it to the PC interface. The PC interface software allows increments of 128k bytes to be uploaded. The start of each block of Sensor Ball memory is examined to ensure that the correct synchronization pattern is stored in the first 16 bits of the memory page. Blank memory (values of hexadecimal FF) and memory pages with a bad synchronization pattern are skipped. An incorrect synchronization pattern indicates that the 16k-byte memory block failed and so should be skipped during data upload.

The Sensor Ball calculates a checksum for each block of data uploaded, and transmits it as the last two bytes before waiting for a response from the PC. The PC interface software also calculates a checksum, and compares the two values. If the values match, the PC interface software sends the Acknowledge control character and the next Flash memory block is read and transmitted. If the checksums do not match, the PC sends the Negative Acknowledge control character, and the block is re-read and re-transmitted by the Sensor Ball.

A variation of the read command is implemented that does not inspect the memory, so does not check for blank or invalid memory and therefore uploads these with the known good data. This command, "r", is not implemented in the PC interface software. It may be helpful to attempt data recovery if the Flash memory became faulty.

Both the "R" and "r" commands allow additional parameters to select where in the Flash memory data upload begins. The PC interface software does not use this feature. The parameter is loaded as two ASCII alphabetic characters, with the lower nibble of each combined to indicate the 128k-byte memory section of interest. The data

uploads continues from that point until terminated by the PC interface software, just like when no parameters are used.

Sensor Ball Status Reporting Module, St_Hand.asm

The status handler routine St_Hand.asm acquires real-time data from each of the six sensors, the three voltage monitors, and the current Flash memory location. These data are then transmitted to the interfaced PC. Once complete, the code returns control to the Cmd_Hand routine, which passes control to the command-processing loop in the main code section, Ball.asm.

Data are acquired using the 16-sample averaging algorithm implemented in the Sensor Ball data acquisition code. The same subroutine is used for normal data acquisition, but a averaging flag bit is set to no averaging for the data stored in Flash memory. The last-used Flash memory address is also sent so that the number of 128k-byte sections of memory available for uploading can be gauged. This information is helpful to know so that time is not wasted attempting to upload blank memory. The status command as implemented in the PC interface software follows the "S" command with the "w" command, which requests the unit to identify itself. The unit identification number will become more important since multiple Sensor Balls will be collecting data.

A second version of the status command is available, but is not implemented in the current PC interface software. The "s" command uploads data in the order that it is defined in the data frame. Thus, the record and frame number, the time variables, and the current Flash memory address all follow the analog channels. This is intended to be a simpler format that can be discerned by using the other "standardized" commands.

Auxiliary Files

Three additional files are used to build the Sensor Ball firmware, and are inserted into the code as "include" files. The Ball_Equ.inc file contains equivalence definitions for the various single-bit programming flags that are used, as well as the constants and assembly-time definitions. Ball_Dat.inc reserves memory for all the variables needed in the code including the data stack structures. Finally, Ball_Msg.inc holds the message text strings. The majority of the text strings are used to define error messages and measurement names that are not accessed by the present PC interface software. These were included in anticipation of an improved interface software development.

Mechanical Design

The mechanical design of the Sensor Ball offered a number of challenges. The primary role of the Sensor Ball housing is to protect the electronics and transducers from potentially damaging mechanical shock and vibration. The housing must also maintain a dry interior while simultaneously permitting easy access for data extraction and battery replacement, and a passage to the water to allow pH and conductivity to be measured. Operational considerations in both the intended consumer environment and the laboratory were important. Finally, the design must closely match the density of water while maintaining an orientation favorable for pH and conductivity measurements. The final prototype design achieved all these design goals.

Sandia provided a non-functional Sensor Ball mass mock-up to P&G in June 2000. Because electrical components were not included, the weight was slightly less than the functional recorder and the mock-up was provided unsealed to allow easy disassembly. The early mock-up also did not include conductivity probes, although their location on the flat surface next to the pH sensor was noted. Scrap circuit boards were cut to the expected size because functional boards had not been fabricated when the mock-up was produced.

Shock and Vibration

The requirements stated that the design must survive certain levels of shock and vibration while operating and a higher, non-operational mechanical shock level. These shock and vibration levels were not quantified but were intended to be the maximum environment expected during a wash cycle. A more difficult condition was described for non-operational mode, in that the Sensor Ball must survive a six-foot drop onto concrete. The latter requirement was meant to consider the possibility of a consumer dropping the unit while transferring it to or from the washing machine.

A major consideration when designing to meet the shock and vibration requirements was protection of the pH probe, its reference electrode, and the four-element conductivity probe. These must penetrate the housing wall to make contact with the wash water. But direct impact on any of the sensors would likely destroy them, so some mechanism for protecting them while still permitting adequate flow of wash water past the sensors was required. The solution selected was to build a cage over the sensors. Options for the cage included solid plastic and welded spring-wire, as well as machined steel. Initially, a plastic cage was designed, but that design proved too weak to withstand the required six-foot drop onto a hard surface. And, the spring structure did not attach well to the main Sensor Ball housing because any impact tended to push the spring wire hold-down points away from the housing. A machined cage was tried using Type 17-4 stainless, tempered, spring steel that was selected for its spring constant, but the material choice was unfortunate due to corrosion susceptibility when combined with the Type 303 stainless steel fasteners. A rust problem developed around the fasteners attaching the metal cage to the Sensor Ball case. Following an investigation that showed a material with a lower toughness was acceptable; the cage material was changed to Type 303 stainless steel to match the fasteners. Still, concerns were raised that any metal cage might mar the surfaces inside a washing machine. The edges of the cage were polished to minimize damage to the washing machine.

The stainless-steel cage, shown in detail in Figure 12, includes a ring that makes full contact with the housing. The steel is also strong enough to permit large open areas for good wash-water contact with the sensors. One of the six arms on the cage is positioned directly over the conductivity probe grouping. This prevents abrasion of the conductivity probes by clothing during the wash cycle. The same approach also protects the pH reference electrode. The pH probe, located on the center axis of the Sensor Ball, is covered by the center of the protective cage. The center location maximizes the internal height available to contain the pH probe body, which is about 2 inches long.

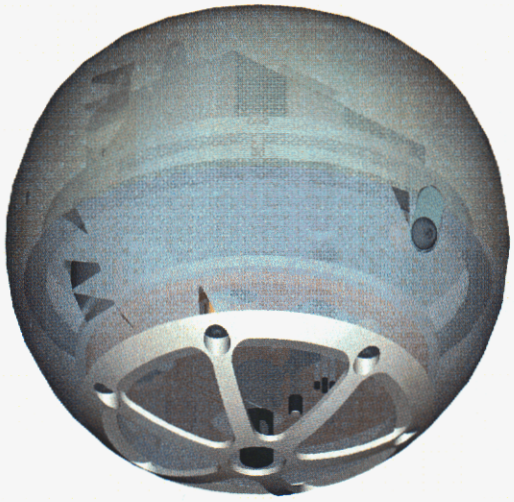


Figure 12. Sensor Ball View Showing the Sensor-Protecting Cage

Six, flat-head number 6 fasteners attach the cage to the Lexan housing. Numerous cracks were discovered in the Lexan case surrounding the fasteners in the first prototype. These were likely caused by several cycles of assembly and disassembly of the Sensor Ball. Since this was considered a severe problem, the case was replaced with an older, but unused case from a previous design version to allow laboratory testing to continue.

Water and Chemical Resistance

The primary design environment for the Sensor Ball is submerged in wash water, so the design must be watertight as well as insensitive to the chemicals in the wash water. Any exposed metallic components must not react with these chemicals, which would otherwise cause a potential failure mechanism or even stain the clothing with oxidation products. Testing of the first prototype design revealed material incompatibilities, which were corrected as described above. The selection of Lexan for the housing material was predicated on its resistance to the expected wash water chemicals as well as its good machining properties – a consideration primarily for the prototypes rather than potential production units.

The first Sensor Ball prototype developed a leaking problem, which could have been caused by three factors. The first issue discovered was that the Honeywell pH sensor admitted water. We determined that a broken seal between the DuraFET and the pH probe body caused this. Our previous assembly practice was to remove the original circuit board that contacted the DuraFET, and replace it with our own board design to make the electrical connections. We now leave the as-supplied Honeywell board in place, and attach wires to traces on the board. Although this unfortunately requires careful, detailed disassembly of the pH probe to re-use the board, the seal is preserved.

The second leak was more a potential issue than a confirmed one. The o-ring used to seal the two halves of the Lexan case together seemed to be inadequate. A change was made to replace the 0.103 square inch cross-section material with 0.139 square inch material. The groove holding the o-ring in the Lexan case was also deepened to accommodate the thicker material. This resulted in a very snug seal between the case halves.

Finally, we observed that the reference electrode could not tolerate a pressure differential, and admitted water. During the pH sensor retrofit, the probe orientation was mistakenly placed with the DuraFET away from the reference electrode. This was corrected to have the DuraFET facing the reference electrode.

All circuit board surfaces were conformally coated with a silicone rubber compound. A silicone rubber seal was also added inside the case around the pH probe and the conductivity probes. The temperature sensor was held in place with a thin layer of epoxy, and then covered with silicone rubber. Finally, the serial interface connector was attached using epoxy.

Sensor Ball Density and Center of Gravity

Finally, the Sensor Ball mechanical design sets the density of the unit to approximately that of water, and maintains the unit's orientation when it is floating in water. To accurately measure the conditions imposed on the clothing in the washing machine, the Sensor Ball must move through the column of water with the clothing. Also, the pH and conductivity sensors must be continuously submerged to collect valid measurements.

The Sensor Ball's volume was dictated in part by the space required for the electronic components. The original intent was to have a tennis-ball sized unit, but that volume was not quite sufficient to hold all the components. The battery location was a particular problem because it had to be accessible for replacement. Thus, it could not be placed under the circuit board where its mass would have helped orient the Sensor Ball. To offset the battery weight above the necessary center of gravity, we placed additional steel ballast below. But then the displacement of the Sensor Ball had to be increased to maintain the desired specific gravity. In its final form, the Sensor Ball closely resembled the shape of the P&G Downey Ball fabric softener dispenser. The resulting weight of the unit, which is essentially the equivalent weight of water displaced by the unit, is 269.1 grams as calculated by the mechanical computer-aided design software, and including a medical-grade lithium battery. The volume is 272.0 cubic centimeters, which results in a 0.997 specific gravity. The specific gravity of water varies from 0.9997 at 10°C to 0.9880 at 50°C. In actuality, the specific gravity of the Sensor Ball as assembled is somewhat lower than calculated, so that the unit is slightly buoyant. The measured weight of the Version 4 Sensor Ball and battery averages 269.8 grams with a standard deviation of 0.4 grams, resulting in a 0.9917 specific gravity.

In addition to steel ballast, the steel cage protecting the transducers helped shift the center of gravity. Because the Sensor Ball is free-floating, maintaining the orientation that keeps the pH and conductivity sensors continuously submerged requires very little center of gravity offset. This offset is the difference between the center of buoyancy and the center of mass. The mechanical computer-aided design software was used to model the weight and center of gravity location for each component in the Sensor Ball. Based on these calculations, we adjusted the Sensor Ball's center of gravity to about 0.266 inches below the displacement center, sufficient to maintain proper orientation. The lateral center of gravity locations were within 0.01 inches of the Sensor Ball centerline so that the unit did not list noticeably.

During tests at P&G in a special transparent washing machine, the Sensor Ball moved through the column of water with the clothes, and maintained the proper orientation. This validated this aspect of the mechanical design.

Design Revisions

The first prototype design, Version 1, was delivered to P&G in early September 2000. A number of requirements changes were identified once researchers used the device. Subsequent deliveries of each design version were as follows: Version 2 in December 2000, Version 3 design in June 2001, and the final prototype design, Version 4, in January 2002. Five units of Version 4 were produced, but only one each of the earlier versions. The Version 2 housing was damaged during testing and replaced, then the new housing reused when the design was transitioned from Version 2 to Version 3.

Substantial Changes in Sensor Ball Version 2

Version 2 included revisions to the circuit board design, and production of a new board. The mercury tilt switch was nixed because it could potentially be hazardous to the consumer. The switch was intended to detect consumer placement of the Sensor Ball into the washing machine, because moving the Sensor Ball would trip the switch with high probability. The design was revised to use the conductivity measurement as the trigger to collect data. Using the low-power feature of the microcontroller keeps power consumption very low while the Sensor Ball is not in a data collection mode. This produces a slightly shorter, but tolerable, data collection life.

The conductivity sensor in Version 1 was intended to be a 4- conductor design, but we realized that the circuit did not actually accomplish this. Part of the problem was the realization that the conductivity measurement would affect the pH measurement unless the conductivity detection signals were coupled to the water using capacitors. Including the feedback circuit required to implement the 4-conductor conductivity measurement seemed to add too much complexity to the design. Thus, we compromised and used a 2-conductor probe.

In Version 1, the temperature sensor was located on the main circuit board. However, this location caused unacceptable delays in acquiring an accurate temperature measurement. The design was modeled, and as a result a more appropriate location for the sensor was selected in the wall of the Lexan case. This change required a minor mechanical modification to cut a pocket in the Lexan.

P&G became interested in measuring the spin-cycle acceleration, so they requested that the accelerometer range be changed from a +/-50g device to a +/-100g device. This caused a circuit error that was only discovered later. The ADXL150 accelerometer operates at 6 V maximum, which is compatible with the Sensor Ball. However, the 100-g device, ADXL190, accepts only 5.5 V supply power maximum. We suspect that this resulted in slightly higher power consumption but otherwise seemed to be tolerated. This decision was later reversed, back to a 50-g accelerometer in Version 3.

A major mechanical design revision was undertaken in Version 2 to simplify the assembly and disassembly of the Sensor Ball. The changes were also intended to reduce the chance that the Sensor Ball would disassemble in the washing machine. In Version 1, six screws distributed around the circumference of the Sensor Ball held the case halves. Also, the battery case was attached to the upper case half, which made working with a disassembled unit clumsy. The Version 2 design included mating threads on either half of the case, so that they screw together. A single security screw was added to prevent consumer tampering. The cage covering the aqueous sensors was rotated to better protect the reference electrode and conductivity probes. The battery mount was moved to a structure above the circuit boards in the lower half of the case, removing all components from the top case half.

The serial interface connector for Version 1 used a Nanonics, 9-pin miniaturized connector. This connector is quite expensive and can be difficult to use. It was replaced in Version 2 by an inexpensive, simple, 3-pin Molex connector. However, this connector had to be epoxied to the board to keep it steady. Finally, the battery monitor measurement was not considered important, so was removed to allow the measurement set to fit better into the Flash memory format.

Primarily Firmware Changes for Version 3

Changes for Version 3 primarily included substantial revisions to the firmware. The code changes included a second, high-speed serial interface speed intended to reduce data upload times, a time-stamping feature to help determine when the data records were collected, a Flash memory failure detection and compensation to minimize potential data losses, addition of additional record information to more clearly separate data collection periods, and a

revision to the general code structure intended to make it easier to maintain. Because the 100-g accelerometer range was insufficient to monitor the spin cycle, the +/-50 g range accelerometer replaced it to provide better resolution. The connection board modifications to the Honeywell pH probe caused it to leak, so these modifications were revised. Instead of removing the original circuit board that connected to the DuraFET, the board was cut and connectors soldered to the board traces. Also, the material selected for the sensor cage was changed to Type 303 stainless to be compatible with the fasteners used. Previously, a material incompatibility caused corrosion. Because the additional data in the Flash memory made the format a non-integral multiple of the memory size, deletion of the battery monitor measurement was no longer justified and it was restored. The overall data collection frequency was reduced to 2.66 samples per second, but the 128k-byte total memory used for each data record was retained. This means fewer samples were collected but this was acceptable to P&G.

Minor Circuit and Firmware Changes to Produce Version 4

The final five prototypes produced were Version 4, sent to P&G on January 16, 2002. The first copy of this version, Unit 3, was sent to P&G January 4 to verify the design changes prior to the construction of the remaining four units. This test verified that the firmware and circuitry changes performed nearly as expected. One remaining firmware issue that was not deemed important enough to correct was the initial powering of the Sensor Ball. Immediately after a battery is installed, the Sensor Ball begins a data collection cycle. This flaw and how to correct it is discussed in the Microcontroller Firmware section.

Several circuitry flaws were discovered and corrected. Because the analog circuitry operates on 6 V and the microcontroller on 3 V, there is a possibility that the analog circuit could apply a voltage greater than the 3-V maximum allowed by the microcontroller. This situation existed on the pH and accelerometer measurements. The most noticeable impact was disruption of the serial data link during data uploads, primarily at the high-speed rate but to a lesser extent at the low-speed data rate as well. Applying the greater than 3-V signal increased the effective microcontroller voltage, which caused the microcontroller oscillator to shift frequency, thereby causing communication failures during data upload. A less noticeable result of the circuit changes was a reduction in power consumption from 35 mA to 25 mA during data collection. Power during low-power, "Sleep" mode remained at 300 microamps.

Finally, we discovered that the reference electrode gel will empty if the pressure differential inside-to-outside of the Sensor Ball exceeds more than a few pounds per square inch. This required a few attempts to correct. The final approach secured the reference electrode with hard epoxy in addition to silicone adhesive to make a leak less likely. A longer term, and better solution would be to replace the type of reference electrode, but this would have required additional mechanical design modifications. This option is discussed in the next section.

Suggested Changes for a Future Design Revision

A number of potential improvements were identified for the Sensor Ball, but we lacked time and funds to implement them. Several of these are discussed here.

Improve the pH Transducer

The most likely source of leaks into the Sensor Ball is the reference electrode. Using a reference electrode that is sealed on the inside of the Sensor Ball would help withstand pressure changes caused by shipping the unit, so that it can be shipped completely assembled. Perhaps pinching the tube closed with a wire clip or some other mechanism to plug the tube inside the Sensor Ball could stop the potential leak. A better seal between the reference electrode material and the Sensor Ball case would also help considerably. The reference electrode dries out when it is not submerged in solution, causing a delay in producing an accurate pH measurement, so perhaps a completely different reference electrode design should be used. Other reference electrode manufacturers provide electrodes that are virtually sealed, although they are much longer than the electrode currently used. From the P&G perspective, a reference electrode with larger AgCl-saturated KCl gel capacity is desirable to increase the useful life of the electrode.

Another issue is the pH probe. Modifying the pH probe as it is supplied from Honeywell requires the most skill of the Sensor Ball assembly process. We recommend negotiating with Honeywell to have them produce a shortened probe tip for the Sensor Ball that can be used without modification. Finally, the selection of capacitors in the pH

signal conditioning circuit may not be optimized for best performance. Some of the high values are probably too high, which may slow the pH measurement response time or could even cause the measurement signal to oscillate.

Use Connectors for the Transducers
All the transducers, excluding the accelerometers, are currently soldered directly into the main circuit board. This makes disassembly of the Sensor Ball for maintenance very difficult. Use of a connector that is accessible from the top of the main circuit board would correct this. However, connector mounting would need to be carefully considered to ensure that the connector did not come loose.

Select a Different Communication Connector
To extract data from the Sensor Ball, a simple, 3-wire cable is attached between the Sensor Ball and a personal computer. However, the connector is somewhat difficult to use. Further, the Attention button is difficult to reach and the LED, used to indicate operating mode, is difficult to view when interfacing to the Sensor Ball. Changing the connector to a standard circuit board header type with a 0.1-inch spacing between leads would make connections simpler to make. This type of connector would also allow adding more wires to automatically put the Sensor Ball in "Attention" mode whenever the connector is attached, add a reset button, allow external power to be applied, and make the LED more readily observable.

Add a True, Real-Time Clock
Sensor Ball Version 3 added code to emulate a real-time clock, allowing the data records to be time-stamped. This emulation is flawed in that the time variable is reset whenever the Sensor Ball is reset, and it does not provide the time in a format readily translated to day, hour, and minute. It measures time in 1.11 millisecond units since the last time the Sensor Ball was reset. Adding a real-time clock will require circuit changes to reallocate microcontroller input pins, but four unused connections to the microcontroller are already available to accomplish this. The Flash memory ready / busy signal is no longer used, so that connection would raise to five the spare signals available.

Combine the Accelerometer Measurements
The three accelerometer channels probably do not provide more information than a single, combined measurement would, and in fact, are combined by P&G during data analysis. The three measurements can be summed into a single channel if an absolute or root-mean square value was first derived for each. Incorporating circuitry that captured an average value of the combined result may also be helpful, and provide more information than the extremely under-sampled measurement that is now implemented. A further enhancement could be adding two ranges of accelerometers, one with high resolution to monitor agitation and a second set with lower resolution but higher full-scale range to monitor the spin cycle. Acceleration can be calculated from:

$$A = r \times \omega^2$$

where r is the washing machine tub radius and ω the rotation rate in radians per second. For a 2-ft diameter washing machine tub and a 500-RPM spin cycle speed, the acceleration is about 85 g. Analog Devices produces a ±100-g accelerometer that uses the same pin-out as the device used, but circuit changes would be required in the Sensor Ball to reduce the supply voltage from 6 V to no higher than 5.5 V.

Include a 4-Probe Conductivity Measurement
As discussed in the Conductivity Transducer section, the present 2-probe conductivity measurement should be replaced with a 4-probe version. This will allow linear measurement over a much broader range of conductivity.

Improve the Response Time of the Temperature Transducer
The temperature measurement takes too long to stabilize when the water that the Sensor Ball is immersed in suddenly changes temperature. This was a problem with the initial prototype, and was thought corrected when the sensor was moved from the circuit board to a pocked milled in the Lexan case. P&G reports a time constant of about 5 minutes. An additional change could be use of a less expensive sensor, although the resulting measurement would be non-linear. The AD590 costs about $10 each, while a thermistor costs less than $1. The Honeywell pH probe contained a thermistor in the probe tip, so this might provide faster response.

Use a Flash-Memory Based Microcontroller
If changes are needed to the Sensor Ball firmware, the microcontroller must be removed and replaced, a fairly invasive and expensive operation. But, a design that incorporated a microcontroller with code held in Flash memory

could be easily reprogrammed as new data collection opportunities are presented. When the Sensor Ball was originally designed a Flash-based microcontroller with a built-in, 12-bits resolution Analog-to-Digital converter was not available, and still may not be available. The design could be revised to use an external analog multiplexer and ADC. Using this option, a 16-bit resolution or higher is ADC is readily available.

Modify the Mechanical Design for Manufacturability
The Sensor Ball's mechanical design will be modified extensively when the transition is made to produce the units in higher volume. The prototype was machined from a solid rod of Lexan, an operation that is too expensive to be practical for more than just a small quantity of prototype units. The mechanical changes should also consider ways to reduce the skill level and time required to assemble the units, and should consider features that simplify transducer maintenance. An issue for P&G is adjusting the weight of the Sensor Ball to accommodate variation in battery weight, which results in a variation in Sensor Ball buoyancy. Also, P&G's test consumers were concerned that the steel cage protecting the sensors has sharp edges that may mar fabrics. The cage was supplied with polished edges, but that seems to have been inadequate to allay these concerns.

Acceptance Testing

The evaluation of the Sensor Ball must accomplish three objectives. First, because the Sensor Ball is still in prototype status, the design must be verified. This shows that the engineering design meets the stated requirements. A particular concern is whether the firmware is operating as intended. Second, the testing must show that the design was assembled properly and that all components are functional. This second objective is the only purpose of testing when designs are produced in volume, but this is not the case for the Sensor Ball prototypes. While the first two objectives can be accomplished with tests at Sandia, the third objective, design validation, must be done at P&G. Design validation will demonstrate that the Sensor Ball design, which meets the design requirements and is functional, will achieve the intended result. Many of the revisions to the Sensor Ball design during the course of this project resulted from P&G testing that showed the design requirements needed adjustment to meet the overall objective. The Sandia test process did not distinguish between tests intended to verify the design or to prove that it was assembled properly. The testing steps and associated data sheet are contained in Appendix D: Sensor Ball Test Procedure.

Testing During Assembly

A few resistor values are considered adjustable to meet functionality goals. Values are recorded for the gain and offset resistors used for the pH measurement circuit, and for the resistor selected to set the microcontroller oscillator frequency. The appropriateness of the values is determined by measuring the circuit response to different pH buffer solutions, and by measuring the microcontroller instruction clock frequency. Error-free serial communication between the Sensor Ball and a personal computer is an indirect indication that the clock frequency is correct.

The Honeywell pH probe requires difficult modifications to make it compatible with the Sensor Ball needs. To ensure that the DuraFET portion of the probe is properly connected electrically to the attached wires, the device is tested to show that a diode voltage drop occurs between the DuraFET drain and substrate, and between the source and substrate. This is tested using the diode check function on a digital multi-meter.

As a final coarse indication of unit functionality, a power supply is connected to the Sensor Ball battery terminals and the current measured when the Sensor Ball is in the data acquisition mode, the Attention mode, and the low-power sleep mode. These data are recorded on the test sheet. Anomalous power consumption is a good indication of malfunction.

Functionality and Calibration Tests

The functionality tests show that the Sensor Ball accomplishes the core, required tasks:

- Communicate with the PC to allow status checks and data upload
- Store data into the Flash memory and retrieve it without error
- Show nominal calibration for temperature, pH, conductivity, and acceleration.

The microcontroller provides the user a means of functionally testing the Sensor Ball without resorting to simply using the unit in an operational capacity. The Sensor Ball can acquire real-time data and send it to a PC when

connected with the interface cable. A status command sent by the PC to the recorder will initiate the microcontroller to take one measurement from each sensor and voltage monitor and send that data back to the PC. This command can be repeated as often as necessary to facilitate testing. Because the Sensor Ball case must be opened to access the interface cable connection, this precludes complete immersion of the unit into the standard solutions. However, all the calibration tests can be readily accomplished.

In most cases, adequate testing of the Sensor Ball can be achieved using the personal computer user interface software, SensorBall, which was written specifically for this design. When commanded, this software displays the measurement value acquired from each Sensor Ball transducer plus some additional status information. However, better information for testing the Flash memory can be acquired using general-purpose serial interface software such as HyperTerminal. All of the Sensor Ball interface commands can be executed plus any resulting error messages can be observed. These test commands are discussed in the "Microcontroller Firmware" section under "Cmd_Hand.asm". The memory test command, "T", writes and reads two data patterns into the Flash memory, and reports errors. It also attempts to mark any bad memory so that it will not be used during normal Sensor Ball data collection. The memory fill command, "F", inserts simulated data records into the Sensor Ball memory to facilitate later data communication tests.

Communication tests are best accomplished using the SensorBall interface software. One of the testing steps using HyperTerminal creates a number of simulated data records in the Sensor Ball Flash memory. The SensorBall software displays errors that occur during data uploads. Both the 19.2k-Baud low data rate and the 115.2k-Baud high data rate are tested.

The intent of the calibration tests is to demonstrate that the Sensor Ball calibration is nominal at room temperature. No attempt is made to make each measurement result consistent to a high degree among the units produced. P&G is responsible for determining the unique, precision calibration values for each unit. The tests at Sandia check the pH measurement using pH 5 and pH 12 buffer solutions. The actual range of the Sensor Ball is just short of 7 pH units, so depending on the measurement offset, one or the other of the pH buffer measurements will be out of range. Conductivity is checked using a 9.65 µSiemens/cm and a 1417 µSiemens/cm standard solution. The accelerometers are tested primarily to show that the zero-g offset point is approximately mid-range. The Sensor Ball is positioned to produce a positive and inverted acceleration measurement for each of the three axes. A difference of 2-g acceleration between the positive and inverted position is observable.

P&G Laboratory and Field Tests

Although Sandia's standard practice was to run each Sensor Ball in a washing machine before shipment to P&G, this test was superficial and primarily focused on ensuring that the Sensor Ball did not leak. P&G ran extensive laboratory tests to establish accurate calibration factors for the Sensor Ball measurements. The calibration factors were also collected over a wide temperature range.

P&G has test washing machines that were used to evaluate the Sensor Ball's performance compared to data collected previously using specialized instrumentation. During the course of the project, these tests helped establish needed revisions to the Sensor Ball requirements and design. For the final, Version 4, prototypes, the tests provided a final check of the Sensor Ball before attempting field trials in consumer test homes.

Field trial results are not yet available, but P&G staff members were very satisfied with the laboratory performance of the final design version. A total of five, Version 4 Sensor Ball units were produced at Sandia to support the field evaluation, and allowed some comparisons of unit-to-unit variation. Figure 13 below shows a P&G staff member testing the mechanical characteristics of the Sensor Ball.

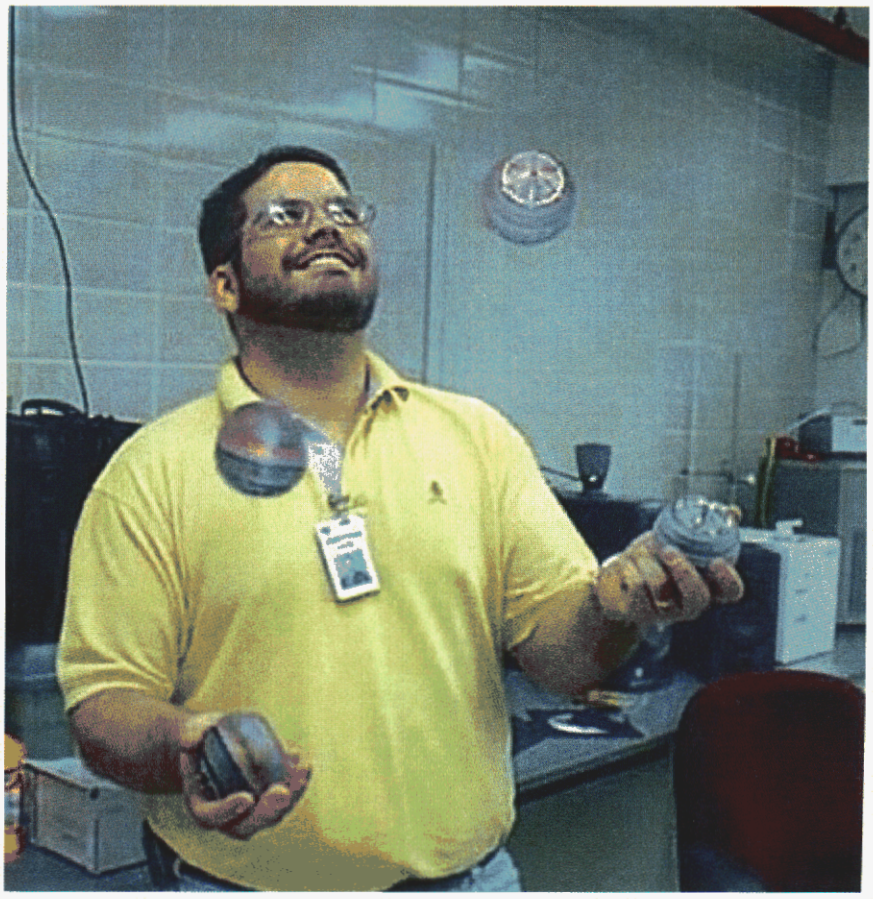


Figure 13. P&G Staff Evaluating Sensor Ball Characteristics

Appendix A: P&G Background and Initial Proposal

F&HC Analytical Sciences - North America
Ivorydale Technical Center

May 5, 1999

Development & Fabrication of a Device for *in-situ* Monitoring of Washing-Machine Chemistry

The following is a proposal to fabricate a miniaturized sensor-device for following "real time" chemistry and other key parameters occurring during the wash.

Background: In order to stay competitive in today's market, we must develop better test methods to shorten the development time of new laundry technologies. We believe one way to accomplish this is to develop "smart" sensors that communicate back to us what is occurring in consumers' homes. By monitoring parameters such as wash temperature, pH, conductivity, turbidity, water hardness, available chlorine, available oxygen, product consumption and washing machine agitation through a wash cycle, we can develop a more comprehensive understanding of consumer practices and the impact of these practices on product performance. Ultimately, we can develop better performance models for speeding in-house development.

A major problem we have encountered in our efforts to develop a smart sensor device is locating commercially available, miniaturized sensor devices with data-capturing capabilities. In addition, miniaturized sensors for some of the parameters we wish to measure are not commercially available, nor do we have the expertise to develop these technologies in-house. Secondary to sensor availability is the need to understand and develop appropriate data-mining techniques to correlate sensor output with consumer preference (model building).

Proposal: We are interested in working with outside laboratories to develop and fabricate miniaturized, data logging devices that measure (listed highest to lowest priority): temperature, pH, conductivity, acceleration/vibration, turbidity, available oxygen/chlorine and water hardness. We would like our initial prototype devices to measure the first 4 above parameters. Robust macro-scale technologies that we have identified for monitoring washing-machine chemistry include toroidal and four-electrode conductivity, ISFET pH, thermistor temperature sensors and piezoelectric acceleration/vibration sensors. Other suitable technologies for measuring these parameters in a miniaturized device could be investigated, but care must be taken to thoroughly validate them under wash conditions.

We would like the device to be capable of measuring: temperature from 5 to 95 °C, pH from 7 to 12, conductivity from 0 to 1500 µS/cm, acceleration from 0 to 100g_n and/or vibration from 0 to 50 mm/s. Data capture via telemetry or self-contained data-logging hardware is also needed. The device should be capable of collecting data on each of the above 4 (or 5) parameters every 30 seconds (or less) for a period of at least two weeks - the time a product is typically placed in a consumer's home for evaluation. We envision a miniaturized device that would be no larger than a tennis ball, contain non-volatile memory, completely waterproof, and have a density of ~ 1g/cm (so that it would not sink or rise during the wash). Event marking capabilities are desirable but not required. The sensor should be easily interfaced with a PC for facile data transfer and manipulation.

We would like to proceed with the fabrication of 10 - 50 of these devices as quickly as possible. The initial fabrication of this miniaturized sensor is just the first step to better understand washing machine chemistry and to gain experience in using this device in the hands of a consumer. If the device proves to be successful in these initial studies, development of more sophisticated devices would be pursued.

Next Steps: Should this be of interest, we would like to request a proposal detailing funding and timing needed to complete the fabrication of 10 sensor devices. If needed, we are always open to discussing this proposal in more detail.

Sensors proposal 5-5-99.doc

In-Situ Monitoring/Recording Device for Aqueous Processes Requirements Document

Submitted to:
The Procter & Gamble Company
Cincinnati, Ohio

Submitted by:
T. A. Rohwer
Sandia National Laboratories
Telemetry & Instrumentation Department
Albuquerque, New Mexico

Version 4, June 2001

Table of Contents

1 General 43
1.1 Scope 43
1.2 Description 43
1.3 Theory of Operation 43
1.4 Definitions 43
1.5 Customer, Supplier, Stakeholders 44
2 Requirements 44
2.1 Electrical Requirements 44
2.2 Mechanical Requirements 44
2.3 Environmental Requirements 45
2.4 Software Requirements 45
2.5 Delivery Requirements 46
2.6 Documentation Requirements 46

1 General

1.1 Scope

1.1.1 This document defines the requirements for a data acquisition system to be used as a proof of concept In-Situ Monitoring/Recording Device for Aqueous Processes.

1.2 Description

1.2.1 The data recorder will be based on the existing Miniature Penetrator (MinPen) design. The small, rugged, self-contained telemetry system developed by Sandia for use by P&G will measure acceleration, pH, temperature and conductivity in an aqueous process environment. The data acquired will be stored in system memory for post-test retrieval and analysis via an external connection between the data recorder and a PC.

1.3 Theory of Operation

1.3.1 The data recorder can automatically sense placement in the aqueous process, and begin a data collection cycle for the duration required. During the data collection cycle, data are measured and stored into memory. Multiple data collection cycles can be stored. Following retrieval from the monitored process, all collected data can be extracted. Simple disassembly will allow access to an interface connector to allow data uploading. Data uploading will be accomplished with a laptop/PC connected to the interface connector. The interface also controls memory erasure. After the data have been uploaded, memory erased, and battery replaced, the data recorder is ready to repeat another measurement period.

1.4 Definitions

1.4.1 **MinPen**. **Min**iature **Pene**trator Instrumentation. Data acquisition system developed by Sandia National Laboratories, Telemetry Organizations specifically for high shock environments.

1.4.2 **Resolution**. The fineness to which a measurement is reported. For example, a 12-bit system can resolve to about 0.02% of full-scale range.

1.4.3 **Accuracy**. The relationship between the measured and true value of a signal. This cannot exceed the resolution, but could also be much worse than the resolution may imply.

1.5 Customer, Supplier, Stakeholders

1.5.1 P&G is the customer with Kris Gansle as the project lead. Telemetry and Instrumentation Department 2665 is the supplier with Tedd Rohwer as the project lead. Stakeholders include Technology Partnerships, represented by Vic Weiss and Willard Hunter.

2 Requirements

2.1 Electrical Requirements

2.1.1 The data recorder will measure 3-axis acceleration, pH, conductivity, and temperature.

2.1.1.1 The dynamic acceleration range will be ±50 g with a resolution of 0.2g. (The range was ±50 g originally, then ±100 g for Version 2, then ±50 g again following additional laboratory testing at P&G.)

2.1.1.2 Dynamic temperature range will be 5°C to 60°C with a resolution of 0.05°C.

2.1.1.3 Dynamic pH range will be 5 to 11 with a resolution of 0.01.

2.1.1.4 Dynamic conductivity range will be 0 to 3000uS/cm with a resolution of 3uS/cm.

2.1.2 Will specify a standard off-the-shelf battery specific to the system to power the data recorder. The recorder will have sufficient power to acquire data for the periods specified in paragraphs 2.1.4 and 2.1.5.

2.1.3 Data acquisition will be initiated when the monitored conductivity exceeds a predetermined threshold. An LED will indicate the data recorder status.

2.1.4 The data acquisition routine will sample 3-axis acceleration, pH, conductivity, and temperature at a minimum rate of 1 samples per second for 40 minutes.

2.1.5 Non-volatile memory capacity will allow the data acquisition routine to be run a minimum of 20 times before it is necessary to download the data.

2.1.6 Downloading the data will be accomplished by connecting the data recorder to a PC/laptop.

2.2 Mechanical Requirements

2.2.1 The data recorder will be water tight for use in aqueous processes.

2.2.2 The density of the data recorder will be that of water ± 10 %.

2.2.3 A connector for data download will be accessible after some disassembly.

2.2.4 The battery will be accessible for replacement after some disassembly.

2.2.5 The data recorder will be bright in color.

2.2.6 The external surface of the recorder will be grippable and resistant to identified chemicals (harshest being bleach).

2.2.7 The mechanical design will incorporate a flat surface to enable stable storage.

2.2.8 The accelerometers will be mounted in a tri-axis orientation.

2.2.9 Conductivity, temperature, and pH sensors will be mounted for direct contact with environment.

2.2.10 The sensor ball will be permanently marked with an ID number.

2.3 Environmental Requirements

2.3.1 Operational Requirements

2.3.1.1 Function in aqueous processes

2.3.1.2 °5C -- °60C

2.3.2 Non-Operational Requirements

2.3.2.1 Survive in aqueous processes

2.3.2.2 °5C -- °60C

2.3.2.3 Six-foot drop onto concrete

2.4 Software Requirements

2.4.1 Installation software compatible with Windows 95/NT

2.4.2 Download data into an ASCII file and reset the memory

2.4.3 Analog status of all data channels

2.4.4 Computer enabled arming of the recorder for bench testing

2.5 Delivery Requirements

2.5.1 One data acquisition system will be delivered to P&G by 8/17/00.

2.5.2 Hardware

2.5.2.1 One (1) sensor ball. (The CRADA was modified to add 5 copies of the final prototype version.)

2.5.2.2 One (1) interface cable

2.5.3 Software – GUI interface software

2.5.4 Documentation

2.6 Documentation Requirements

2.6.1 Department 2665 will maintain a Fabrication and Assembly book detailing assembly and testing issues of the data acquisition system.

2.6.2 User's Guide

2.6.3 Systems Requirements Document – this document

2.6.4 Critical Design Review documentation

2.6.5 Functional test results

Appendix C: PC Interface Software Guide

Establishing Serial Communication

Data are uploaded from the Sensor Ball to a computer using a direct wire connection to the computer's RS-232 serial port. When the top half of the Sensor Ball case is removed, the 9-V battery, "Attention" push-button, and serial port are accessible. To access data, the 3-pin connector on the Sensor Ball must be connected with the Sensor Ball to Computer cable, and the 9-Pin Serial Connector end plugged into the COM1 serial port on a personal computer. The cable contains only three wires and uses only receive, transmit, and ground signals of the RS-232 connection. Power must be supplied to the Sensor Ball, either by using the 9-V battery or an equivalent power supply. To begin the communication process, the small, white "Attention" button must be depressed and held until the LED near the Sensor Ball cage is illuminated continuously. This is the Command Mode, and will time out with no input after about five minutes. In autonomous data collection mode, the Sensor Ball serial interface is not monitored, so no external commands can be executed.

The next step is to run the PC interface software, "SensorBall", on the computer. This program's display has changed to include a debug frame on the right side of the screen. This shows the data exchanged with the Sensor Ball in hexadecimal format, with some interpretive comments. The "Clear" button below the debug frame clears the information received to that point.

Sensor Ball Commands

Some of the command options presented in the PC interface software, SensorBall, have changed over the course of the project. The commands available from SensorBall are listed below with their associated functions.

[I]Initializing – Reset the Sensor Ball, returning it to the power-up state, including returning the default communication speed to 19.2k Baud and zeroing the time counter. The effect is the same as removing and reapplying power on the Sensor Ball. The "Attention" button will need to be pressed again to reestablish communication after this command. The Flash memory is read to determine the number of data collection Records currently stored in the unit, and identify the next memory location to use when beginning another collection cycle.

[R]RAM Read – Read, or upload data from the Sensor Ball Flash memory. After the "radio button" is selected on the SensorBall software screen, additional configuration displays appear that are active after the "Send Command" button selected. The "Number of Records" dial actually uploads the number of 128kB Flash memory sections specified. (If an Acquire function were terminated before the complete Record was stored, less than the entire 128kB memory would be used for that one Record.) The computer file name must also be selected. While waiting for this input, the Sensor Ball LED will flash in a pause-blink-blink pattern. Once the final "Read" button is selected, the data are uploaded and stored in the specified file. The LED will toggle on or off for each Record Frame (28 measurement sets) uploaded. If the high Baud rate was selected, uploading 8-MB memory should take less than 20 minutes.

[A]Acquiring to Flash – Acquire one 128kB data Record, but additional records will be acquired if the Sensor Ball is in a conductive solution (the normal operation of the Sensor Ball.) This initiates the same recording function as if the Sensor Ball had detected sufficient conductivity to store a Record. The time to store a complete Record is now 45 minutes (measurement sample rate 2.66 Hz). However, pushing and holding the "Attention" button until the LED remains on continuously can halt the recording process. This halts the recording process and returns the Sensor Ball to the command-processing mode.

[S]Status Request –Returns the current values of the Sensor Ball measurements, the number of 128k byte memory sections containing data, and shows the unit identification (version and unit number). A blank Flash memory would return a zero for the Record count.

[E]Erase Memory – Erase all 32MB of Flash Memory in the Sensor Ball. The Sensor Ball monitors the result as each 16kB Flash Memory Block is erased, and if an error is detected, that Block is marked as defective and not used

to record data. The LED intensity will dip and flicker slightly during the 10 seconds or so that are required to erase memory.

[B]Baud Rate Change –This sends a message to the Sensor Ball to toggle between 19.2k Baud (the default speed) and 115.2k Baud. If the Sensor Ball and computer get out of sync with regard to communication speed, the situation can usually be corrected by issuing the Baud rate toggle command a couple of times to synchronize the speeds.

Appendix D: Sensor Ball Test Procedure

Unit Number: _____ Date: _____

Assembly information

Microcontroller instruction clock Frequency Test (U10-P10): _____ kHz
R13 Value for offset is: _____ kΩ.
R21 Value for Gain is: _____ kΩ.
R46 Value for μC Clock is: _____ kΩ.

Functionality Tests

The properties of the Sensor Ball that will be verified are as follows:

- Communication with PC
- Flash memory good
- Power consumption nominal
- Nominal pH and conductivity measurements in standard solutions

Connect the Sensor Ball to a PC, start the HyperTerminal program (configured at 19.2k Baud). Set the supply voltage to 10.0V. Configure a DMM to monitor current supplied to the Sensor Ball. Attach power clips to the appropriate battery clip terminals. Verify that "Sensor Ball Version 4 Unit _" was received, with the unit number matching the expected value.

The Sensor Ball should be in Acquisition mode, with the LED blinking at about a 3Hz rate. Record the supply current. Current in Acquisition mode: _____ (Should be less than 30mA)

Hold the "Attention" button until the LED is steady. This halts the Acquisition mode. Record the supply current. Current in Attention mode: _____ (Should be less than 30mA)

Use HyperTerminal to send the following commands. The first character is a zero, "0". In each case, type the command and hit carriage return. The command should be echoed back by the Sensor Ball. Then send the "Verify" phrase "0V", also followed by carriage return.

1. Send the memory Erase command "0E". Check that erase messages are received after every 4MB for the entire 32MB of memory. Y / N

2. Send the fill command "0F04". Check that messages are returned after every Record, and that 4 Records are written. Y / N

3. Send the Memory upload command, "0R". Verify that the LED has a Blink-Blink-Pause sequence. Y / N

4. Press the "Attention" button on the Sensor Ball to return it to command mode. The LED should be on steady. Y / N

Calibration Tests

Exit HyperTerminal, and run SensorBall12. In response to the Status command, the display should show "Sensor Ball Version 4 Unit _" was received, with the expected unit number matching the displayed value. Y / N

Set up pH and conductivity calibration standards. When the Sensor Ball is immersed in the standard, move it in the solution briefly in a stirring motion. Then, execute the Status command using the SensorBall12 software. Record the values in the table below. After removing the Sensor Ball from each standard, use the air hose to blow off solution clinging to the sensors.

Table 1. pH Standard Test Results

	Air (before immersion)	pH 5 Standard	pH 12 Standard
pH, Counts			
Temperature, Counts			

Table 2. Conductivity Standard Tests

	Air (before immersion)	9.65 µSiemens/cm Standard	1417 µSiemens/cm Standard
Conductivity, Counts			
Temperature, Counts			

Table 3. Static Accelerometer Tests

	Normal Orientation (+Y) / Inverted (-Y)	Rotate 90° Left – Attention button down (+X) / Inverted (-X)	Rotate 90° Back – battery terminals up (+Z) / Inverted (-Z)
Accel. X, Counts			
Accel. Y, Counts			
Accel. Z, Counts			

Table 4. Monitor Channels

Record Number	
Battery, V	
+3V Supply	
+6V Supply	

Data Upload / Memory Tests

Execute the Baud command. The display should show 115.2 k Baud. Execute the Status command to verify that communication still works. Y / N

Execute the Read command, selecting 5 records to upload. This step assumes that the Functionality tests were run, which places 4 records into memory. The upload should take about 3 minutes. Note error messages in right panel of SensorBall12 screen.
Number of errors: _____

Exit SensorBall12, and run HyperTerminal at 19.2 k Baud. Press the "Attention" button to reset the Sensor Ball serial interface to 19.2 k Baud. The display should increment with a "." character about every 4 seconds. Y / N

Use HyperTerminal to send the following commands. The first character is a zero, "0". In each case, type the command and hit carriage return. The command should be echoed back by the Sensor Ball. Then send the "Verify" phrase "0V", also followed by carriage return.

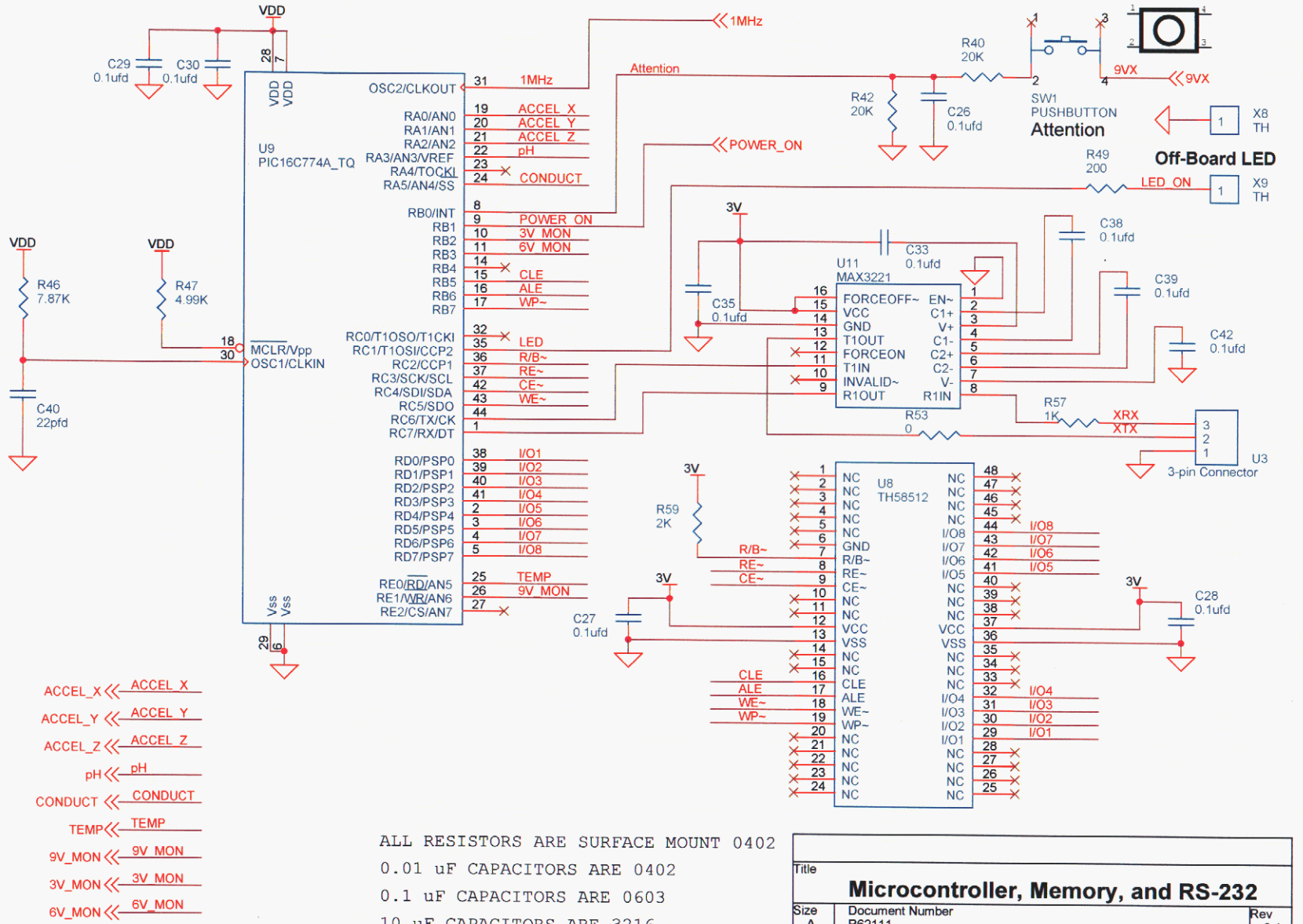
1. Send the memory Test command "0T". This process will take about 40 minutes. Two sets of messages will be generated. Check that erase messages are received after every 4MB for the entire 32MB of memory. Then, check that test messages were generated after every 1MB. Verify that all error values are 00. Y / N

2. Make sure the Sensor Ball is in "Sleep" mode. Do this by either letting the interface time out (about 5 minutes after completing the last command) or send the Initialize command, "0I" followed by Verify, "0V".
   Measure the supply current: _____ (Should be about 300 μA)

End of Testing

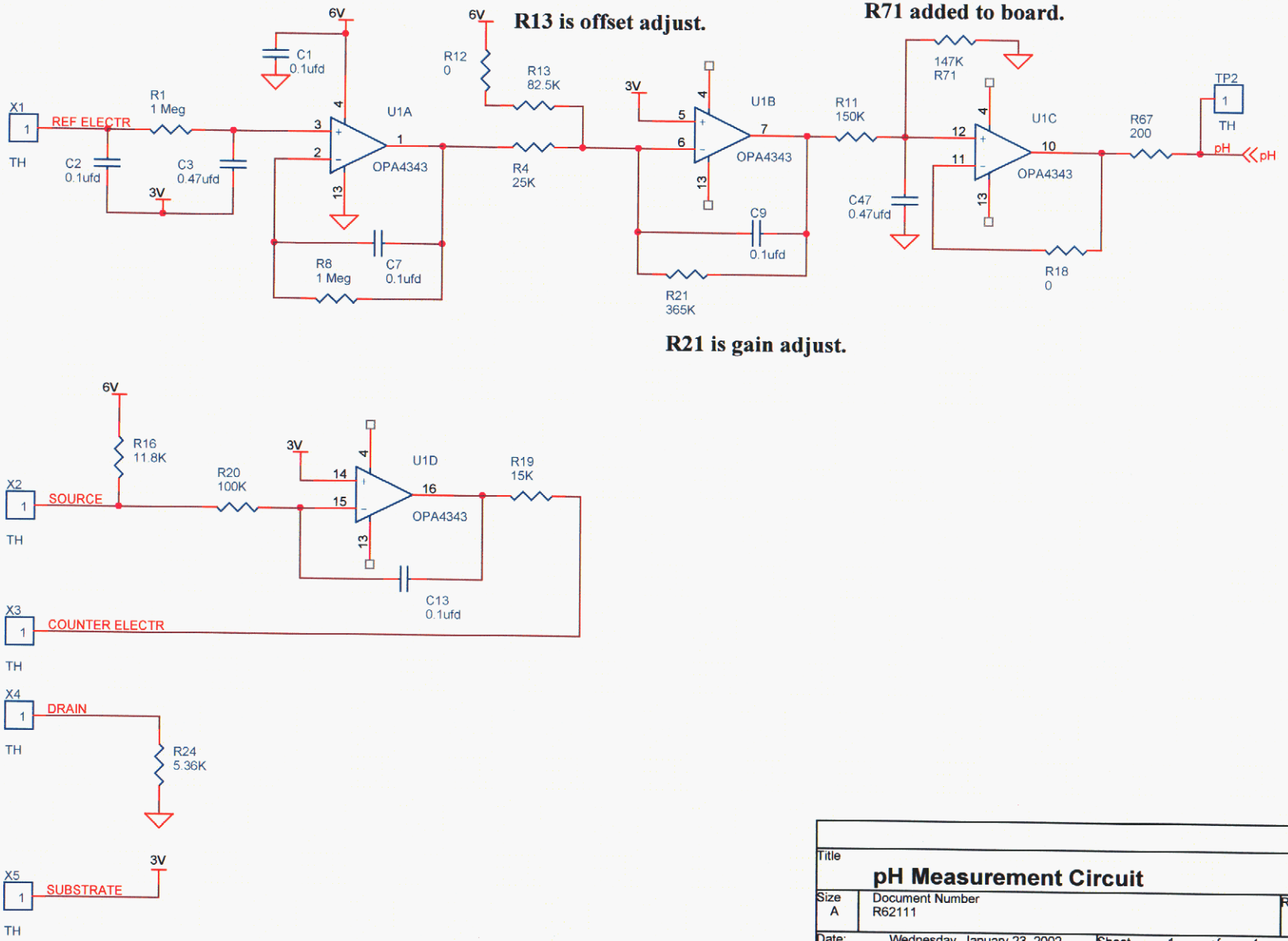
Remove power and the serial cable. Screw on the Sensor Ball top half. (Note that the Sensor Ball must be shipped with the halves separated to avoid a pressure differential across the reference electrode.)

Appendix E: Schematics

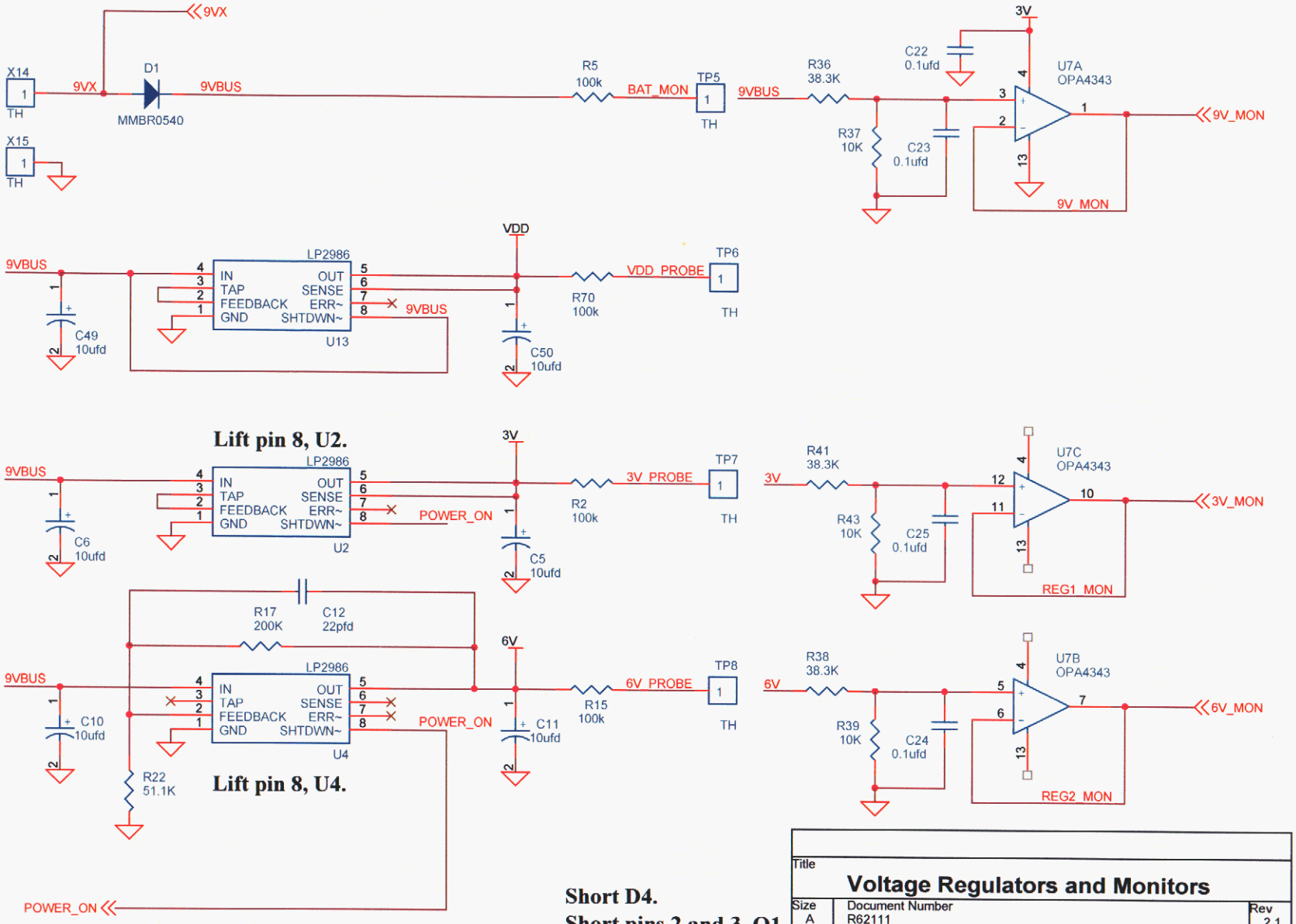


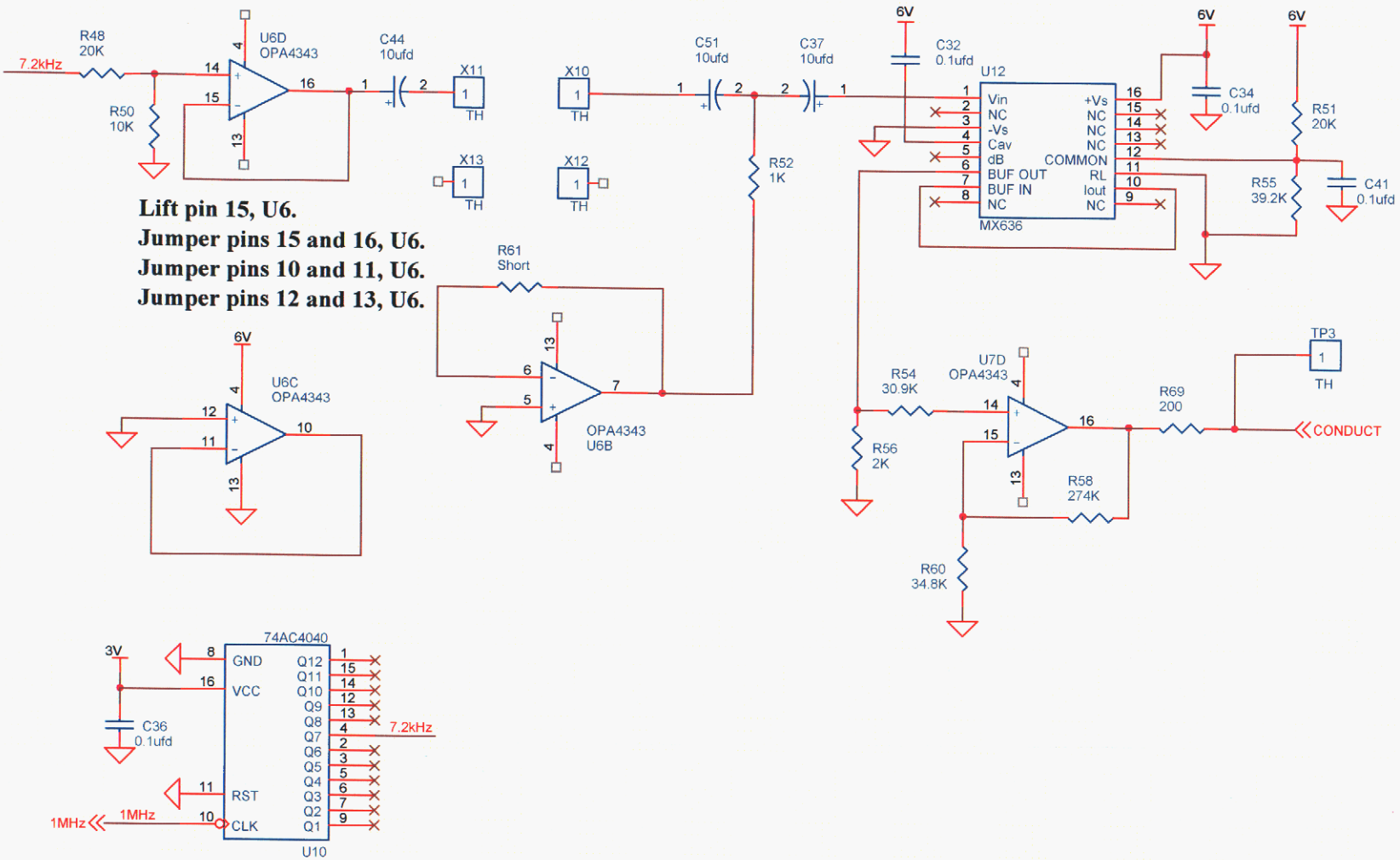
ALL RESISTORS ARE SURFACE MOUNT 0402
0.01 uF CAPACITORS ARE 0402
0.1 uF CAPACITORS ARE 0603
10 uF CAPACITORS ARE 3216

Title	**Microcontroller, Memory, and RS-232**	
Size A	Document Number R62111	Rev 2.1
Date: Wednesday, January 23, 2002	Sheet 1 of 1	



Title	**pH Measurement Circuit**		
Size A	Document Number R62111		Rev 2.1
Date: Wednesday, January 23, 2002	Sheet 1 of 1		

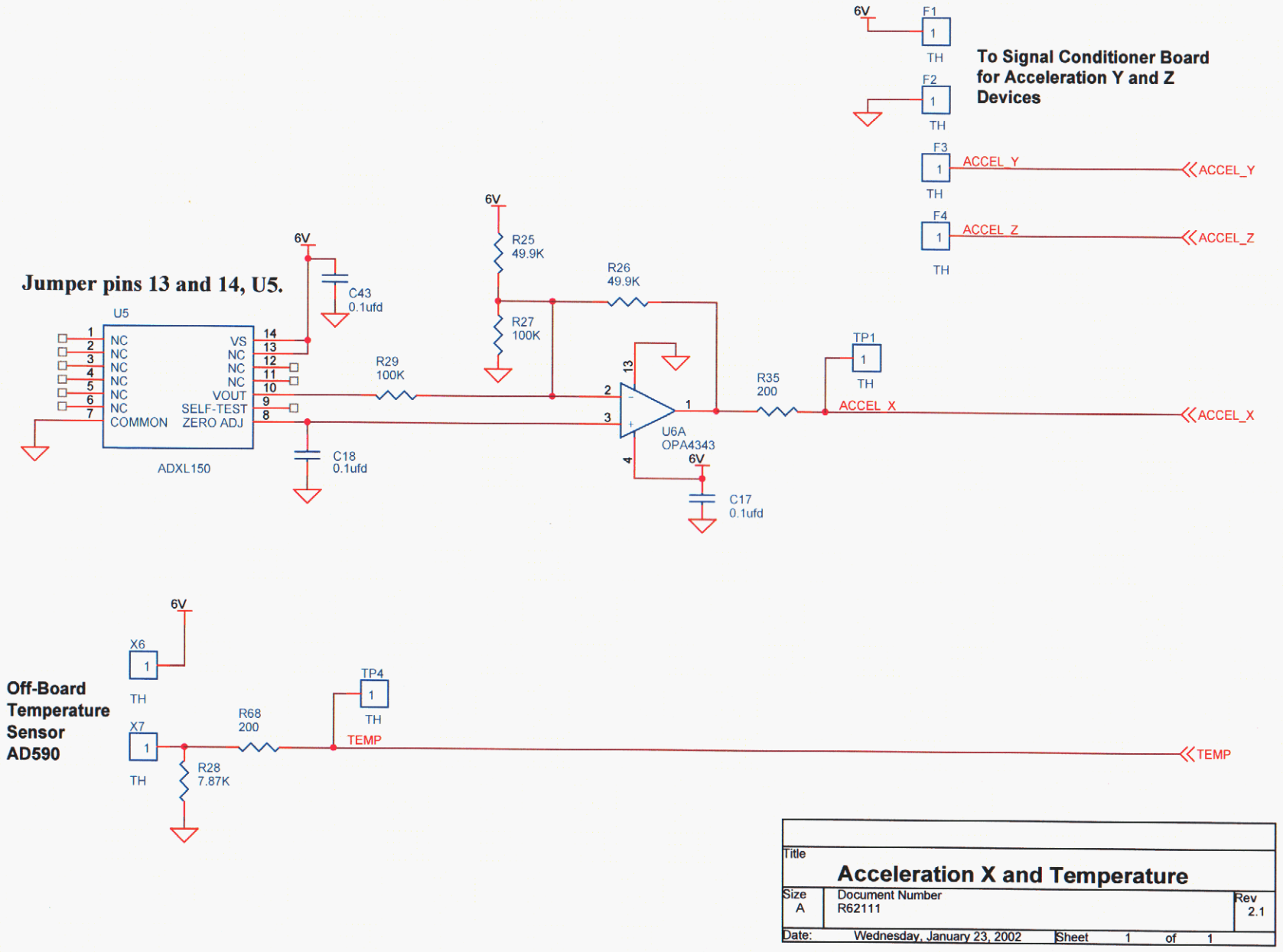

9VX
X14
1
TH
9VX
D1
MMBR0540
9VBUS
R5
100k
BAT_MON
TP5
1
TH
X15
1
TH
9VBUS
R36
38.3K
C22
0.1ufd
3V
U7A
OPA4343
R37
10K
C23
0.1ufd
9V_MON
9V_MON
9VBUS
LP2986
IN
TAP
FEEDBACK
GND
OUT
SENSE
ERR~
SHTDWN~
U13
C49
10ufd
VDD
VDD_PROBE
TP6
R70
100k
C50
10ufd
Lift pin 8, U2.
9VBUS
LP2986
U2
C6
10ufd
POWER_ON
3V
3V_PROBE
TP7
R2
100k
C5
10ufd
R41
38.3K
U7C
OPA4343
3V
R43
10K
C25
0.1ufd
3V_MON
REG1_MON
R17
200K
C12
22pfd
9VBUS
LP2986
U4
C10
10ufd
R22
51.1K
Lift pin 8, U4.
POWER_ON
6V
6V_PROBE
TP8
R15
100k
C11
10ufd
R38
38.3K
U7B
OPA4343
6V
R39
10K
C24
0.1ufd
6V_MON
REG2_MON
Short D4.
Short pins 2 and 3, Q1.
Title
Voltage Regulators and Monitors
Size
A
Document Number
R62111
Rev
2.1
Date:
Wednesday, January 23, 2002
Sheet
1
of
1



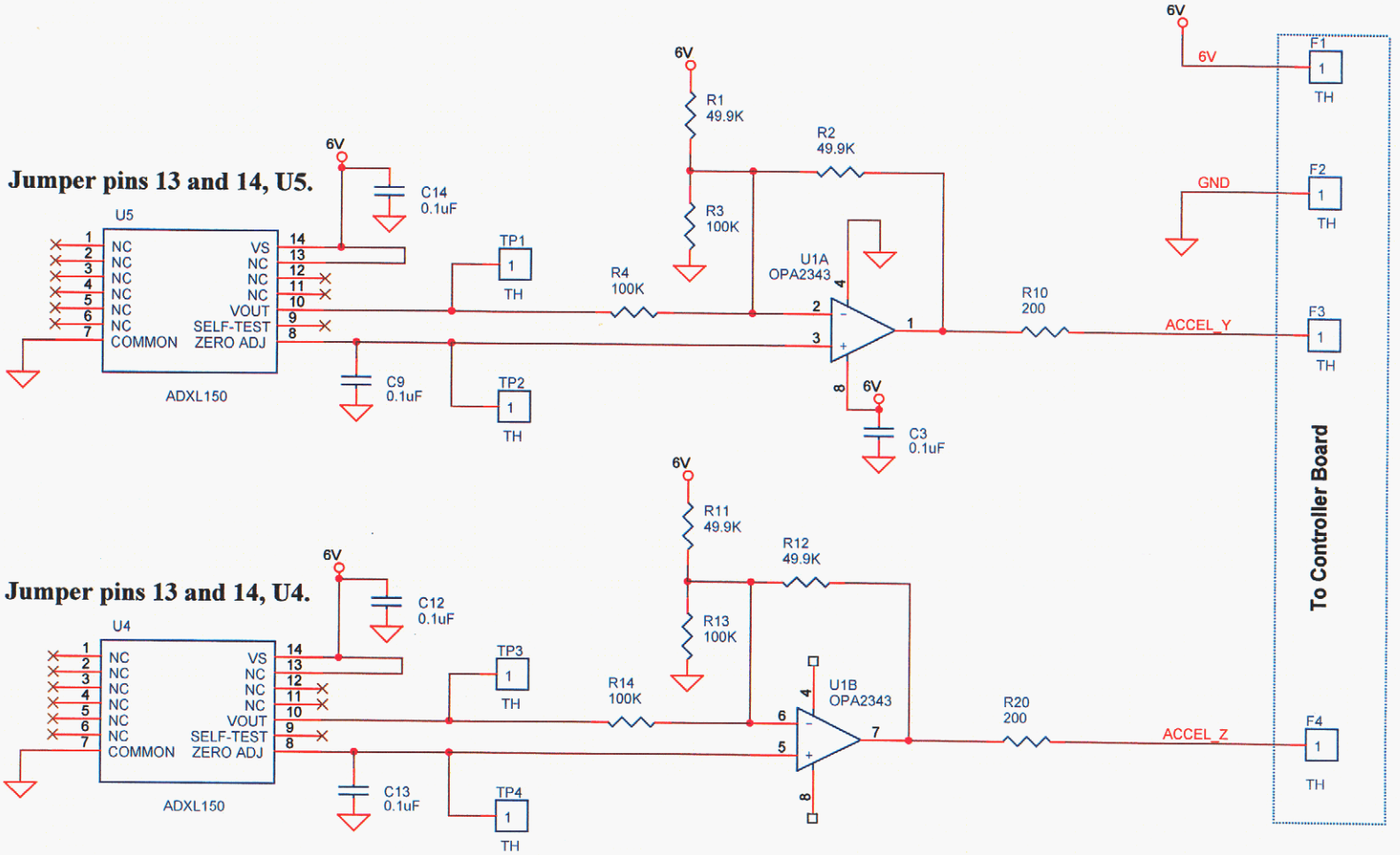
Lift pin 15, U6.
Jumper pins 15 and 16, U6.
Jumper pins 10 and 11, U6.
Jumper pins 12 and 13, U6.

Clock is microcontroller instruction cycle clock, so about 3.686MHz divided by four.

Title	**Conductivity Circuit**		
Size A	Document Number R62111		Rev 2.1
Date:	Wednesday, January 23, 2002	Sheet 1 of 1	



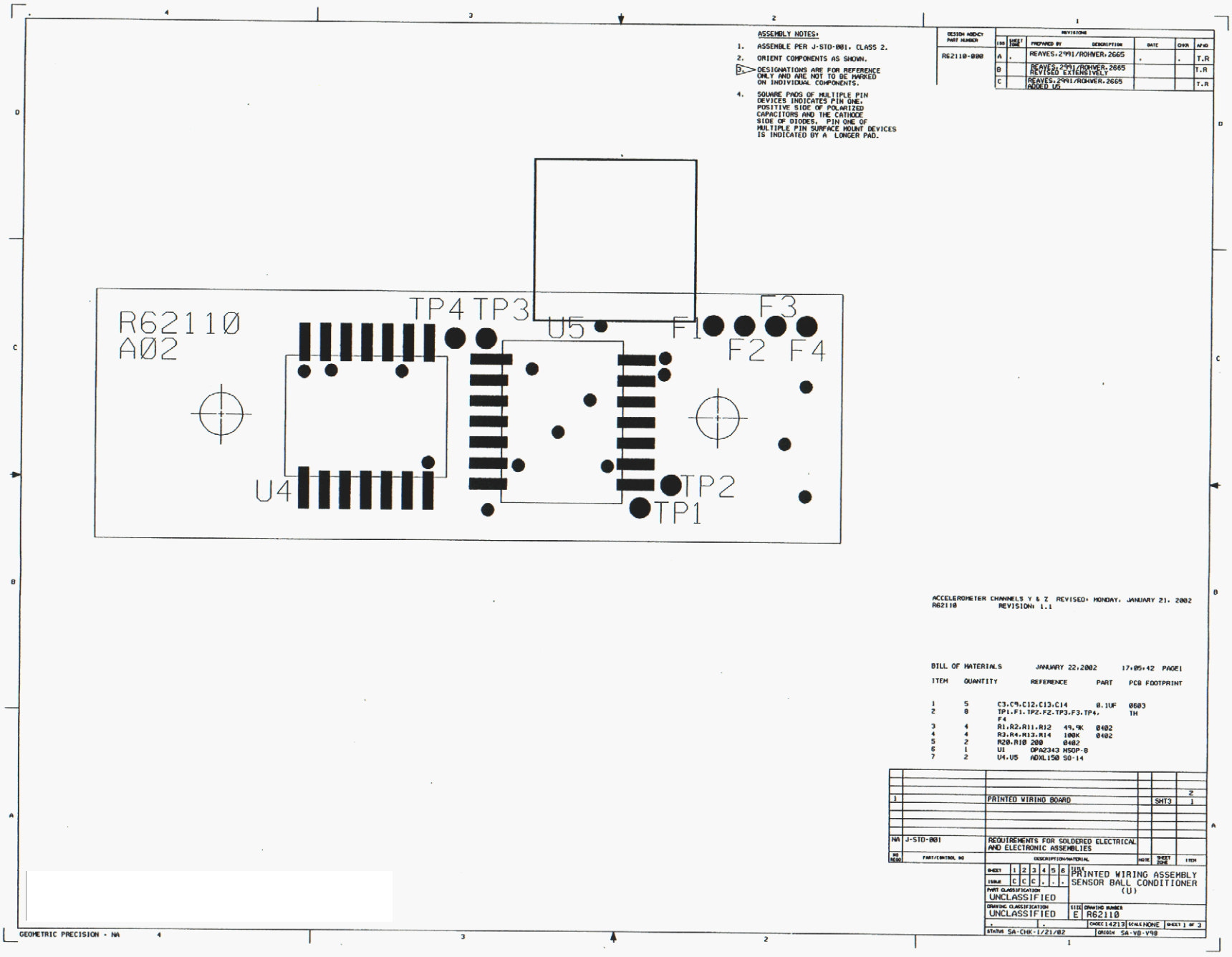
Title	**Acceleration X and Temperature**	
Size A	Document Number R62111	Rev 2.1
Date: Wednesday, January 23, 2002	Sheet 1 of 1	

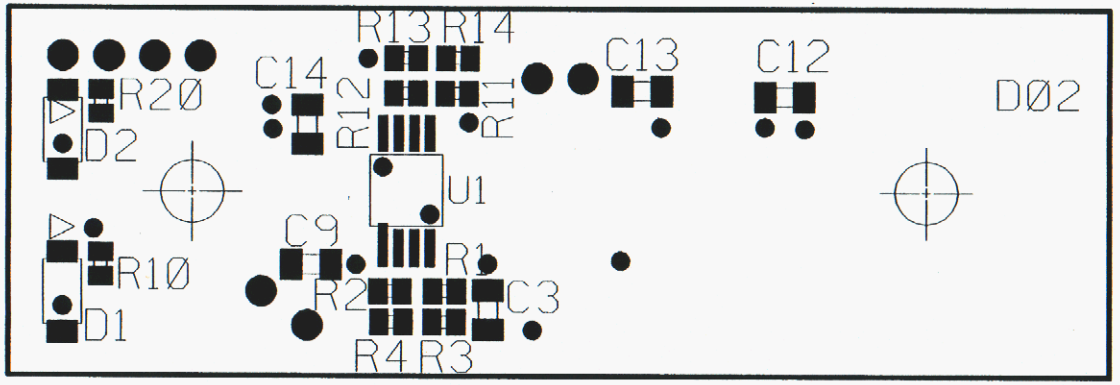


ALL RESISTORS ARE SURFACE MOUNT 0402
ALL CAPACITORS ARE 0402 EXCEPT
0.15 uF & 0.1 uF CAPACITORS ARE 0603
1 uF CAPACITORS ARE 0805
10 uF CAPACITORS ARE 3216

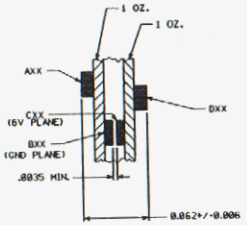
Title: **Accelerometer Channels Y & Z**		
Size: A	Document Number: R62110	Rev: 1.1
Date: Thursday, January 17, 2002	Sheet 1 of 1	

Appendix F: Assembly Drawings and Electronics Materials Lists





DRAWING NUMBER	R62110
DRAWING CLASSIFICATION	UNCLASSIFIED
SIZE E	CAGEC 14213 SCALE NONE
ISSUE C	SHEET 2 OF 3
STATUS SA-CHK-2/21/02	



TYPICAL SECTION
AFTER LAMINATION
SCALENONE

8. SOLDER MASK:
SOLDER MASKING OF PRIMARY AND SECONDARY SIDES OF THE BOARD SHALL BE PER MASKING ARTWORK MASTER, OR CAD DATA, OVER BARE COPPER USING PHOTOIMAGEABLE PERMANENT POLYMER FILM PER IPC-SM-840 TYPE B, CLASS 1. RESIZING FOR MINIMAL PAD TO MASK CLEARANCE IS PERMISSABLE.

9. MARKING:
MARKING SHALL BE PER MARKING ARTWORK, OR CAD DATA, USING WHITE NONCONDUCTIVE EPOXY INK OR ETCHED IN COPPER. BOARD SHALL BE MARKED WITH VENDOR LOGO FOLLOWED BY 94V-0 AND DATE CODE OF MANUFACTURE CONSISTING OF A FOUR DIGIT NUMBER (FIRST TWO, WEEK; LAST TWO, YEAR)

10. MANUFACTURE IN ACCORDANCE WITH PERFORMANCE STANDARD IPC-6012, TYPE 3, CLASS 3, PROCESS 3. FINAL FINISH X.
A. POSITIVE ETCHBACK OF PLATED THROUGH HOLES SHALL BE .0002 TO .003 (.0005 PREFERED).
B. FOR TYPE 3 PRODUCTS, ELECTRICALLY VERIFY FOR OPENS AND SHORTS.
C. COUPONS REQUIRED.

11. MODIFICATIONS OR REPAIRS SHALL BE IN ACCORDANCE WITH IPC-R-700.

6. CONDUCTOR WIDTH AND SPACING:
THE NOMINAL CONDUCTOR WIDTH AND SPACING ON THE FINISHED BOARD IS DESIGNED FOR:

WIDTH - 0.008
SPACING - 0.008

7. HOLE REQUIREMENTS:
A. MINIMUM ANNULAR RING SHALL BE 0.002 INCH (OUTER) AND 0.001 INCH (INNER) MEASURED AT TRACE PAD JUNCTION.
B. HOLE LOCATIONS TO BE 0.014 INCH DDT.
C. HOLES SIZES APPLY AFTER SOLDER COATING AND REFLOW. ALL HOLES ARE PLATED THRU EXCEPT AS IDENTIFIED. ALL HOLE SIZES IN THE DRILL FILE ARE NOMINAL. PLATED THRU HOLES HAVE A TOLERANCE OF +/- .003 AFTER PLATING. ALL HOLE SIZES 0.014 INCH OR BELOW MAY BE FILLED WITH SOLDER OR SOLDER MASK MATERIAL.
D. PLATED THROUGH HOLE SIZE AND QUANTITY ARE AS FOLLOWS:

Through Holes			
Symbol	Diameter(in)	Plated	Quantity
B	0.0260	Yes	8
P	0.0140	Yes	16
R	0.1040	No	2

FABRICATION NOTES:

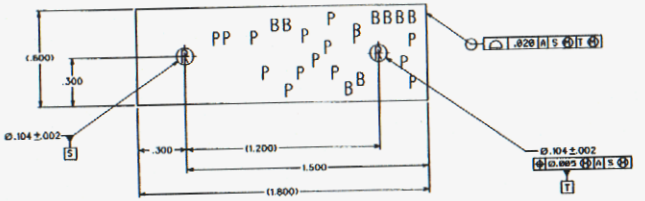
1. HANDLE ARTWORK PER IPC-D-310.
4 EA. CONDUCTIVE LAYERS
2 EA. SOLDER MASKS
2 EA. SILKSCREEN

2. MATERIAL:
LAMINATE GLASS FIBER EPOXY FLAME RETARDANT PER IPC-4101, TYPE GF, SIZE AND CONSTRUCTION IN ACCORDANCE WITH CROSS SECTION. PREPREG MATERIAL TYPE GF PER IPC-4101.

3. COPPER PLATE:
A. SURFACE - TOTAL COPPER ON THE SURFACE INCLUDING ORIGINAL CLADDING PLUS PLATED COPPER SHALL BE 2.0 OUNCE.
B. HOLES - COPPER COATING ON WALL OF HOLES SHALL BE 0.001 AVERAGE. NO VALUE LESS THAN 0.0000.

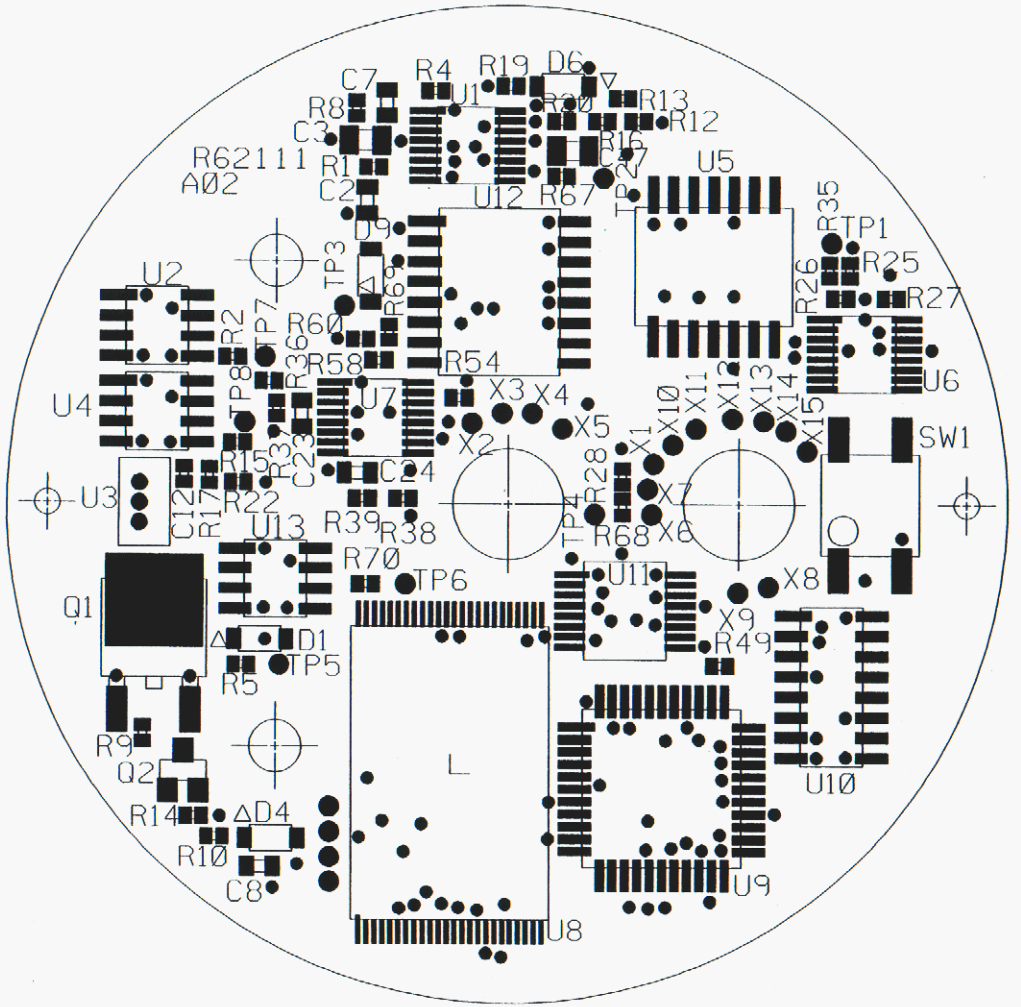
4. SOLDER:
HOLES - SOLDER COAT ON WALL OF PLATED THROUGH HOLES SHALL BE TIN-LEAD (60% TIN, 40% LEAD) IN ACCORDANCE WITH THE SOLDERABILITY REQUIREMENTS OF J-STD-003.

5. GOLD PLATE: N/A



AR	SK-R62110-D02	SILKSCREEN				
AR	SK-R62110-A02	SILKSCREEN				
AR	SM-R62110-B02	SOLDERMASK				
AR	SM-R62110-A02	SOLDERMASK				
AR	DD-R62110-D02	MASTER PATTERN				
AR	DD-R62110-C02	MASTER PATTERN				
AR	DD-R62110-B02	MASTER PATTERN				
AR	DD-R62110-A02	MASTER PATTERN				
NA	IPC-4101/24	SPECIFICATION FOR BASE MATERIALS FOR RIGID AND MULTILAYER PRINTED BOARDS (GF, TG 150-200 C, ELECTRODEPOSITED CU, MIE				
NA	IPC-SM-840	QUAL. AND PERF. OF PERMANENT POLYMER COATING (SOLDER MASK) FOR PWB'S				
NA	IPC-R-700C	GUIDELINES FOR MOD., REWORK AND REPAIR OF PRINTED BOARDS AND ASSYS				
NA	IPC-D-310	GUIDELINES FOR PHOTOTOOL GENERATION AND MEASUREMENT TECHNIQUES				
NA	IPC-6012	QUAL. AND PERF. SPECIFICATION FOR RIGID PRINTED BOARDS				
NA	J-STD-003	SOLDERABILITY TESTS FOR PRINTED BOARDS				
NO REQD	PART/CONTROL NO	DESCRIPTION/MATERIAL	NOTE	SHEET ZONE	ITEM	

DRAWING NUMBER
R62110
DRAWING CLASSIFICATION
UNCLASSIFIED
SIZE E | CAGEC 14213 | SCALE 4X1
ISSUE C | SHEET 3 OF 3
STATUS SA-CHK-1/21/02



ASSEMBLY NOTES:

1. ASSEMBLE PER J-STD-001, CLASS 2.
2. ORIENT COMPONENTS AS SHOWN.
3. DESIGNATIONS ARE FOR REFERENCE ONLY AND ARE NOT TO BE MARKED ON INDIVIDUAL COMPONENTS.
4. SQUARE PADS OF MULTIPLE PIN DEVICES INDICATES PIN ONE. POSITIVE SIDE OF POLARIZED CAPACITORS AND THE CATHODE SIDE OF DIODES. PIN ONE OF MULTIPLE PIN SURFACE MOUNT DEVICES IS INDICATED BY A LONGER PAD.

DESIGN AGENCY PART NUMBER	REVISIONS						
	ISO	SHEET ZONE	PREPARED BY DESCRIPTION	DATE	CHKR	APVD
R62111-000	A	.	REAVES,2991/ROHNER,2665	.	.	T.R
	B		REAVES,2991/ROHNER,2665 ADDED X14,X15,D8,D9, R67,R69,R59,R62			T.R
	C		REAVES,2991/ROHNER,2665 REVISED EXTENSIVELY			T.R

MICROCONTROLLER, MEMORY, AND RS-232 REVISED: TUESDAY, JANUARY 22, 2002
R62111 REVISION: 2.1

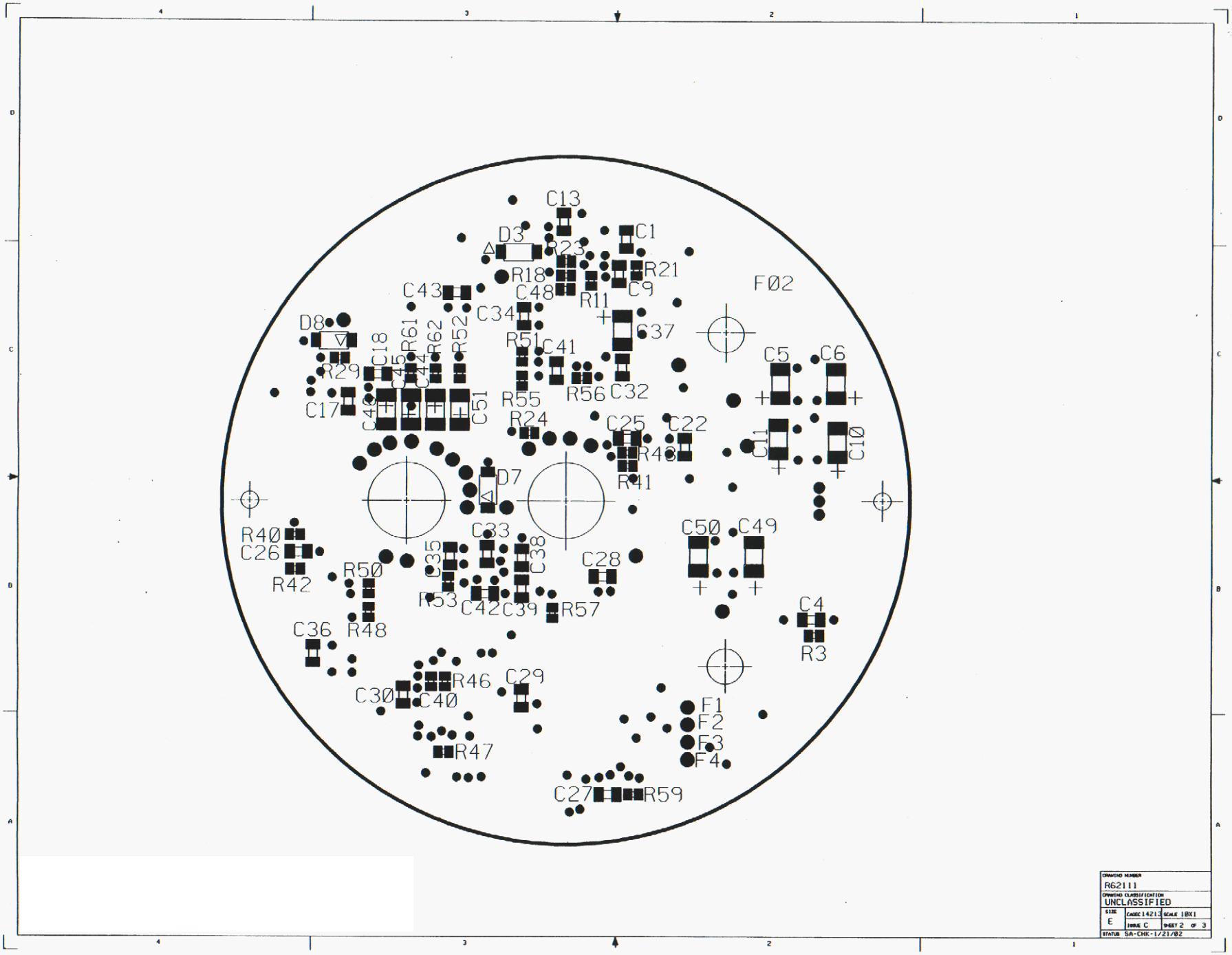
BILL OF MATERIALS JANUARY 22,2002 17:04:04 PAGE1

ITEM	QUANTITY	REFERENCE	PART	PCB FOOTPRINT
1	26	C1,C2,C7,C9,C13,C17,C18, C22,C23,C24,C25,C26,C27, C28,C29,C30,C32,C33,C34, C35,C36,C38,C39,C41,C42, C43	0.1UFD	0603
2	2	C3,C47	0.47UFD	0805
3	9	C5,C8,C10,C11,C37,C44, C49,C50,C51	10UFD	3216
4	2	C48,C12	22PFD	0402
5	1	D1	MMBR0540	SOD-123
6	27	X1,TP1,F1,X2,TP2,F2,X3, TP3,F3,X4,TP4,F4,X5,TP5, X6,TP6,X7,TP7,X8,TP8,X9, X10,X11,X12,X13,X14,X15	TH	
7	2	R8,R1	1 MEG	0402
8	7	R2,R5,R15,R20,R27,R29, R70	100K	0402
9	1	R4	25K	0402
10	1	R11	150K	0402
11	3	R12,R10,R53	0	0402
12	1	R13	82.5K	0402
13	1	R16	11.8K	0402
14	1	R17	200K	0402
15	1	R19	15K	0402
16	1	R21	365K	0402
17	1	R22	51.1K	0402
18	1	R24	5.36K	0402
19	2	R26,R25	49.9K	0402
20	2	R48,R28	7.87K	0402
21	5	R35,R49,R67,R68,R69	200	0402
22	3	R36,R30,R41	30.3K	0402
23	4	R37,R39,R43,R50	10K	0402
24	4	R40,R42,R40,R51	20K	0402
25	1	R47	4.99K	0402
26	2	R57,R52	1K	0402
27	1	R54	30.9K	0402
28	1	R55	39.2K	0402
29	2	R59,R56	2K	0402
30	1	R58	274K	0402
31	1	R60	34.8K	0402
32	1	R61	SHORT	0402
33	1	R71	147K	0402
34	1	SW1	PUSHBUTTON	
35	3	U1,U8,U7	OPA4343	SSOP-16
36	3	U2,U4,U13	LP2986	SO-8
37	1	U3	3-PIN CONNECTOR	
38	1	U5	ADXL150	SO-14
39	1	U6	TH58512	TSOP-48
40	1	U9	PIC16C774A	TQFP-44
41	1	U10	74AC4040	SOIC-16 NARROW
42	1	U11	MAX3221	SSOP-16
43	1	U12	MX836	SO-16 WIDE

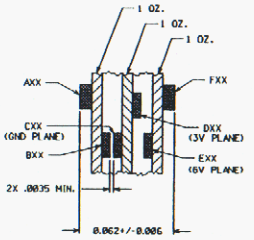
					2
1		PRINTED WIRING BOARD		SHT3	1
NA	J-STD-001	REQUIREMENTS FOR SOLDERED ELECTRICAL AND ELECTRONIC ASSEMBLIES			
NO REQD	PART/CONTROL NO	DESCRIPTION/MATERIAL	NOTE	SHEET ZONE	ITEM

SHEET 1 2 3 4 5 6 | DRAWING PRINTED WIRING ASSEMBLY SENSOR BALL CONTROLLER (U)
ISSUE C C C . . .
PART CLASSIFICATION UNCLASSIFIED
DRAWING CLASSIFICATION UNCLASSIFIED
SIZE E | DRAWING NUMBER R62111
CAGE 14213 | SCALE 10X1 | SHEET 1 OF 3
STATUS SA-CHK-1/21/02 | ORIGIN SA-VB-V98

GEOMETRIC PRECISION - NA



DRAWING NUMBER	R62111
DRAWING CLASSIFICATION	UNCLASSIFIED
SIZE E	CAGEC 14213 SCALE 10X1
	ISSUE C SHEET 2 OF 3
STATUS	SA-CHK-1/21/02



TYPICAL SECTION
AFTER LAMINATION
SCALE:NONE

8. SOLDER MASK:
SOLDER MASKING OF PRIMARY AND SECONDARY SIDES OF THE BOARD SHALL BE PER MASKING ARTWORK MASTER, OR CAD DATA, OVER BARE COPPER USING PHOTOIMAGEABLE PERMANENT POLYMER FILM PER IPC-SM-840 TYPE B, CLASS 1. RESIZING FOR MINIMAL PAD TO MASK CLEARANCE IS PERMISSABLE.

9. MARKING:
MARKING SHALL BE PER MARKING ARTWORK, OR CAD DATA, USING WHITE NONCONDUCTIVE EPOXY INK OR ETCHED IN COPPER. BOARD SHALL BE MARKED WITH VENDOR LOGO FOLLOWED BY 94V-0 AND DATE CODE OF MANUFACTURE CONSISTING OF A FOUR DIGIT NUMBER (FIRST TWO, WEEK; LAST TWO, YEAR)

10. MANUFACTURE IN ACCORDANCE WITH PERFORMANCE STANDARD IPC-6012, TYPE 3, CLASS 3, PROCESS 3, FINAL FINISH X.
A. POSITIVE ETCHBACK OF PLATED THROUGH HOLES SHALL BE .0002 TO .003 (.0005 PREFERED).
B. FOR TYPE 3 PRODUCTS, ELECTRICALLY VERIFY FOR OPENS AND SHORTS.
C. COUPONS REQUIRED.

11. MODIFICATIONS OR REPAIRS SHALL BE IN ACCORDANCE WITH IPC-R-700.

6. CONDUCTOR WIDTH AND SPACING:
THE NOMINAL CONDUCTOR WIDTH AND SPACING ON THE FINISHED BOARD IS DESIGNED FOR:

WIDTH - 0.008
SPACING - 0.008

7. HOLE REQUIREMENTS:
A. MINIMUM ANNULAR RING SHALL BE 0.002 INCH (OUTER) AND 0.001 INCH (INNER) MEASURED AT TRACE PAD JUNCTION.
B. HOLE LOCATIONS TO BE 0.014 INCH ODT.
C. HOLES SIZES APPLY AFTER SOLDER COATING AND REFLOW. ALL HOLES ARE PLATED THRU EXCEPT AS IDENTIFIED. ALL HOLE SIZES IN THE DRILL FILE ARE NOMINAL. PLATED THRU HOLES HAVE A TOLERANCE OF +/- .003 AFTER PLATING. ALL HOLE SIZES 0.014 INCH OR BELOW MAY BE FILLED WITH SOLDER OR SOLDER MASK MATERIAL.
D. PLATED THROUGH HOLE SIZE AND QUANTITY ARE AS FOLLOWS:

Through Holes			
Symbol	Diameter(in)	Plated	Quantity
A	0.0190	Yes	3
B	0.0260	Yes	27
P	0.0140	Yes	147
Q	0.0640	No	2
S	0.1300	No	2
⊗	0.2750	No	2

FABRICATION NOTES:

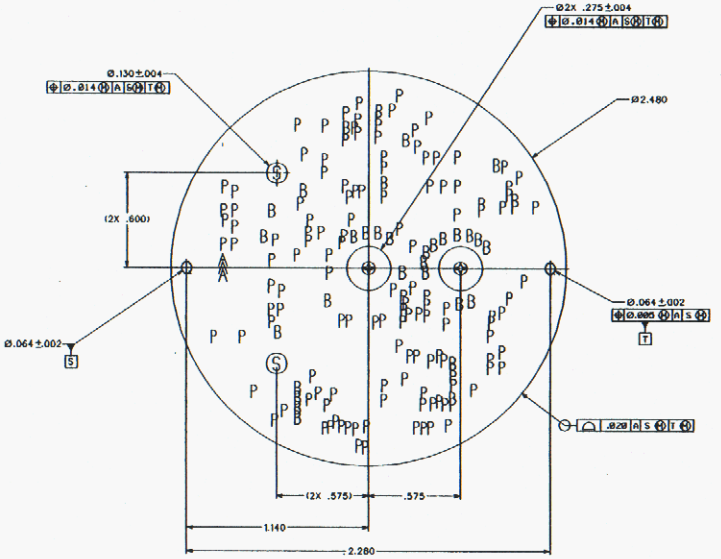
1. HANDLE ARTWORK PER IPC-D-310.
6 EA. CONDUCTIVE LAYERS
2 EA. SOLDER MASKS
2 EA. SILKSCREEN

2. MATERIAL:
LAMINATE GLASS FIBER EPOXY FLAME RETARDANT PER IPC-4101, TYPE GF. SIZE AND CONSTRUCTION IN ACCORDANCE WITH CROSS SECTION. PREPREG MATERIAL TYPE GF PER IPC-4101.

3. COPPER PLATE:
A. SURFACE - TOTAL COPPER ON THE SURFACE INCLUDING ORIGINAL CLADDING PLUS PLATED COPPER SHALL BE 2.0 OUNCE.
B. HOLES - COPPER COATING ON WALL OF HOLES SHALL BE 0.001 AVERAGE, NO VALUE LESS THAN 0.0008.

4. SOLDER:
HOLES - SOLDER COAT ON WALL OF PLATED THROUGH HOLES SHALL BE TIN-LEAD (60% TIN, 40% LEAD) IN ACCORDANCE WITH THE SOLDERABILITY REQUIREMENTS OF J-STD-003.

5. GOLD PLATE: N/A



AR	SK-R62111-B02	SILKSCREEN			
AR	SK-R62111-A02	SILKSCREEN			
AR	SM-R62111-B02	SOLDERMASK			
AR	SM-R62111-A02	SOLDERMASK			
AR	DD-R62111-F02	MASTER PATTERN			
AR	DD-R62111-E02	MASTER PATTERN			
AR	DD-R62111-D02	MASTER PATTERN			
AR	DD-R62111-C02	MASTER PATTERN			
AR	DD-R62111-B02	MASTER PATTERN			
AR	DD-R62111-A02	MASTER PATTERN			
NA	IPC-4101/24	SPECIFICATION FOR BASE MATERIALS FOR RIGID AND MULTILAYER PRINTED BOARDS (GF, TG 150-200 C, ELECTRODEPOSITED CU, HTE			
NA	IPC-SM-840	QUAL. AND PERF. OF PERMANENT POLYMER COATING (SOLDER MASK) FOR PWB'S			
NA	IPC-R-700C	GUIDELINES FOR MOD., REWORK AND REPAIR OF PRINTED BOARDS AND ASSYS			
NA	IPC-D-310	GUIDELINES FOR PHOTOTOOL GENERATION AND MEASUREMENT TECHNIQUES			
NA	IPC-6012	QUAL. AND PERF. SPECIFICATION FOR RIGID PRINTED BOARDS			
NA	J-STD-003	SOLDERABILITY TESTS FOR PRINTED BOARDS			
NO REQ	PART/CONTROL NO	DESCRIPTION/MATERIAL	NOTE	SHEET ZONE	ITEM

DRAWING NUMBER
R62111
DRAWING CLASSIFICATION
UNCLASSIFIED
SIZE E | CAGE 14213 | SCALE 4X1
ISSUE C | SHEET 3 OF 3
STATUS SA-REL-12/5/00

This page intentional left blank

Controller Board Bill of Materials

Microcontroller, Memory, and RS-232 Revised: Thursday, January 17, 2002
R62111 Revision: 2.1

Bill Of Materials January 17,2002 8:50:53 Page1

Item	Quantity	Reference	Part	PCB Footprint
1	26	C1,C2,C7,C9,C13,C17,C18, C22,C23,C24,C25,C26,C27, C28,C29,C30,C32,C33,C34, C35,C36,C38,C39,C41,C42, C43	0.1ufd	0603
2	2	C3,C47	0.47ufd	0805
3	9	C5,C6,C10,C11,C37,C44, C49,C50,C51	10ufd	3216
4	2	C40,C12	22pfd	0402
5	1	C48	230pfd	0402
6	1	D1	MMBR0540	SOD-123
7	27	X1,TP1,F1,X2,TP2,F2,X3, TP3,F3,X4,TP4,F4,X5,TP5, X6,TP6,X7,TP7,X8,TP8,X9, X10,X11,X12,X13,X14,X15	TH	
8	2	R8,R1	1 Meg	0402
9	7	R2,R5,R15,R20,R27,R29, R70	100K	0402
10	1	R4	25K	0402
11	1	R11	150K	0402
12	1	R13	82.5K	0402
13	1	R16	11.8K	0402
14	1	R17	200K	0402
15	1	R19	15K	0402
16	1	R21	365K	0402
17	1	R22	51.1K	0402
18	1	R24	5.36K	0402
19	2	R26,R25	49.9K	0402
20	2	R46,R28	7.87K	0402
21	5	R35,R49,R67,R68,R69	200	0402
22	3	R36,R38,R41	38.3K	0402
23	4	R37,R39,R43,R50	10K	0402
24	4	R40,R42,R48,R51	20K	0402
25	1	R47	4.99K	0402
26	2	R57,R52	1K	0402
27	1	R53	0	0402
28	1	R54	30.9K	0402
29	1	R55	39.2K	0402
30	2	R59,R56	2K	0402
31	1	R58	274K	0402

```
32    1     R60    34.8K 0402
33    1     R61    Short 0402
34    1     R71    147K  0402
35    1     SW1    PUSHBUTTON
36    3     U1,U6,U7     OPA4343      SSOP-16
37    3     U2,U4,U13    LP2986       SO-8
38    1     U3     3-pin Connector
39    1     U5     ADXL150      SO-14
40    1     U8     TH58512      TSOP-48
41    1     U9     PIC16C774A_TQ      TQFP-44
42    1     U10    74AC4040     SOIC-16 Narrow
43    1     U11    MAX3221      SSOP-16
44    1     U12    MX636 SO-16 Wide
```

Assembly Changes and Patches for R62111

Assembly R62111 contains patches, and has a number of deleted components. The components deleted are as follows:

Diodes Deleted
D3, D4, D6, D7, D8, D9. (Only D1 remains)

Transistors Deleted (all deleted)
Q1, Q2

Capacitors Deleted
C4, C8 (0.1uF)
C45, C46 (10uF)

Resistors Deleted
R3, R9, R10, R14, R23, R62

Resistors Added:
R71 – Connect from U1 pin 12 to ground (U1 Pin 13)

Patches:
Jumper pads 2 and 3 of the Q1 footprint
Lift pin 8 of U2
Lift pin 8 of U4
Tie U2-pin8 and U4-pin8 to D4
Short pads for the D4 footprint
Jumper U6 pin 10 to U6 pin 11
Jumper U6 pin 12 to U6 pin 12 (ground)
Lift U6 pin 15. Jumper U6 pin 15 to U6 pin 16
Jumper U5 pin 13 to U5 pin 14

Signal Conditioner Board Bill of Materials

Sensor Ball Signal Conditioner Revised: Thursday, January 17, 2002
Revision: 00

Bill Of Materials January 17,2002 14:01:33 Page1

Item	Quantity	Reference	Part	PCB Footprint
1	5	C3,C9,C12,C13,C14	0.1uF	0603
2	8	TP1,F1,TP2,F2,TP3,F3,TP4,F4		TH
3	4	R1,R2,R11,R12	49.9K	0402
4	4	R3,R4,R13,R14	100K	0402
5	2	R20,R10	200	0402
6	1	U1	OPA2343	MSOP-8
7	2	U4,U5	ADXL150	SO-14

Assembly Changes and Patches for R62110

Assembly R62110 contains patches, and has 2 deleted components. The components deleted are as follows:

Diodes Deleted (all diodes deleted)
D1, D2

Patches:
Jumper U4 pin 13 to U4 pin 14
Jumper U5 pin 13 to U5 pin 14

Microcontroller Programming Steps

The Microchip PIC16C774 microcontroller resides on assembly R62111, the Controller Board. The sequence of steps is listed below to modify the Sensor Ball code for unit identification, and then program a microcontroller component. The assembly language compiler used for this project was Microchip's Mplab Version 5.40. This seems to be a DOS application or at least exhibits those limitations in file name conventions and so on.

The unit number is encoded in two include files in the code, and must be changed in both locations. Comments in the files point to the lines to be changed for a new unit number. The file names are:
Ball_msg.inc
Ball_equ.inc

Next, compile the complete program set, which consists of the files Ball.asm, Cmd_Hand.asm, Flash.asm, Rd_Hand.asm, Ser_Hand.asm, and St_Hand.asm. One of the limitations with the Mplab compiler seems to be the number of characters allowed in the full directory path to the files being compiled. The limit seems to be about 40 characters.

Following compile, the next step is to program the part. This operation was done on a different computer than the compile at Sandia. A Microchip Mplab programmer must be connected to the computer. The programmer must have a PIC16CXX, 44-pin TQFP interface module.

After connecting the programmer, run the Mplab software, and then enable the programmer tool. Be sure to open the Ball.hex file to ensure that the compile process produced a new executable file. The Unit and Version numbers are displayed in the ASCII text near the top of the Ball.hex file. Ensure that the programmer displays the 16C774 part number.

Set the Device ID hex code to equal the Unit number. This is readable only by the verify tool, but may help track a series of parts before they are installed onto circuit boards.

Install a blank device in the programmer. Select the Blank Check function, and verify that the part reads as blank. Then, select Program Device. Verify that the process returned error-free. Next select the Verify device option and ensure that no errors are reported there.

Finally, remove the programmed microcontroller from the programmer, label it, and place it in an appropriate anti-static container. The leads are very easily bent so need some type of protection.

Appendix G: Circuit Board Connector Pin Definitions

Table 3. Directly Wired Connections

Through-Hole	Signal
F1	6V Regulated
F2	GND
F3	ACCEL Y
F4	ACCEL Z
X1	Reference Electrode
X2	Source – pH Electrode
X3	Counter Electrode – pH Electrode
X4	Drain – pH Electrode
X5	Substrate – pH Electrode
X6	AD590+
X7	AD590-
X8	LED-
X9	LED+
X10	Conductivity Electrode
X11	Conductivity Electrode
X12	Conductivity Electrode
X13	Conductivity Electrode

Table 4. Test Points

Pin	Signal
TP1	Acceleration X
TP2	pH
TP3	Conductivity
TP4	Temperature
TP5	Battery Monitor
TP6	VDD Probe (μP Power)
TP7	3V Probe
TP8	6V Probe

Appendix I: Mechanical Components

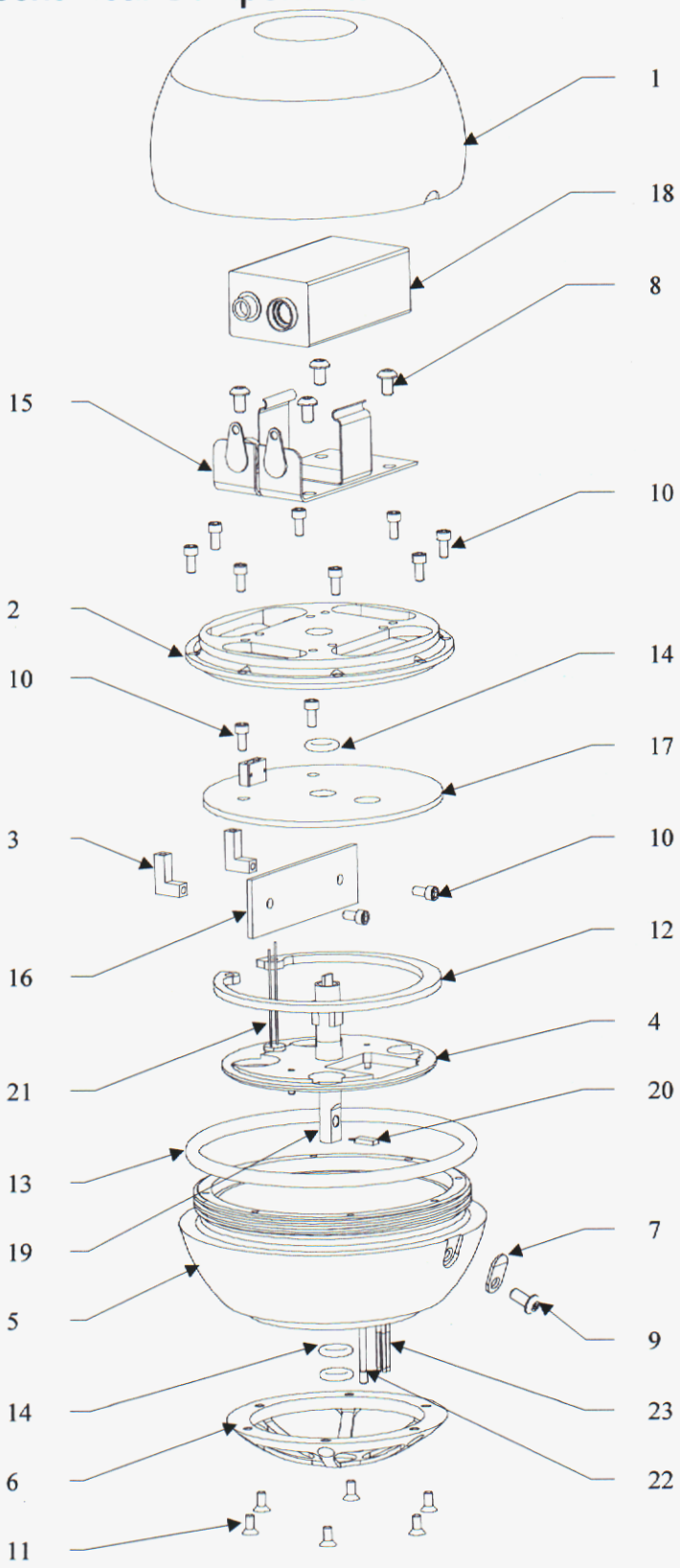


Figure 14. Component Detail, Sensor Ball Exploded View

Table 5. Sensor Ball Mechanical Bill of Materials

Item Number	Quantity Required	Part Number	Description
1	1		Upper Housing (Lexan)
2	1		Battery Plate (6061-T6 Aluminum)
3	2		L-Bracket (6061-T6 Aluminum)
4	1		Ballast Plate with ∅.0625 x .125 long pins (300 Series Cres)
5	1		Lower Housing (Lexan)
6	1		Cage (300 Series Cres)
7	1		Security Clip (300 Series Cres)
8	4		Screw, 82° Flat Head, #4-40unc x .188 Long (Cres)
9	1		Screw, Button Head, Security, #4-40unc x .250 Long (Cres)
10	12		Screw, Socket Head Cap, #2-56unc x .188 Long (Cres)
11	6		Screw, 82° Flat Head, #2-56unc x .188 Long (Cres)
12	1		Retaining Ring, Internal, ∅2.125 (Steel)
13	1	2-232 E540-80	O-Ring, 2.734 id x ∅.139 cross-section (Parker)
14	3	2-009 E540-80	O-Ring, .208 id x ∅.070 cross-section (Parker)
15	1	1295	Battery Holder, Plastic, 9V (Keystone Electronics)
16	1	R62110-001	Signal Conditioning Printed Circuit Board, Rectangular
17	1	R62111-001	Controller Printed Circuit Board, Round
18	1	L522	Battery, 9V (Energizer)
19	1		LED, T-1, Red
20	1	AD590	Temperature Sensor (Analog Devices)
21	1	51204976-002	Probe, ph Sensor, Modified (Honeywell)
22	1	MI-402 mod	Reference Electrode (Microelectrodes)
23	4		Conductivity Electrodes, platinum (Microelectrodes)

Appendix J: Mechanical Assembly Procedure

The Sensor Ball case separates into two halves, with the electronics and support structure in the lower half and a simple shell forming the upper half. The halves screw together, producing a watertight seal using a rubber O-ring. A critical issue in the Sensor Ball assembly is ensuring that the halves have been screwed tightly together while still properly positioning the keyway slot to accept the security clip. Step-by-step assembly instructions follow.

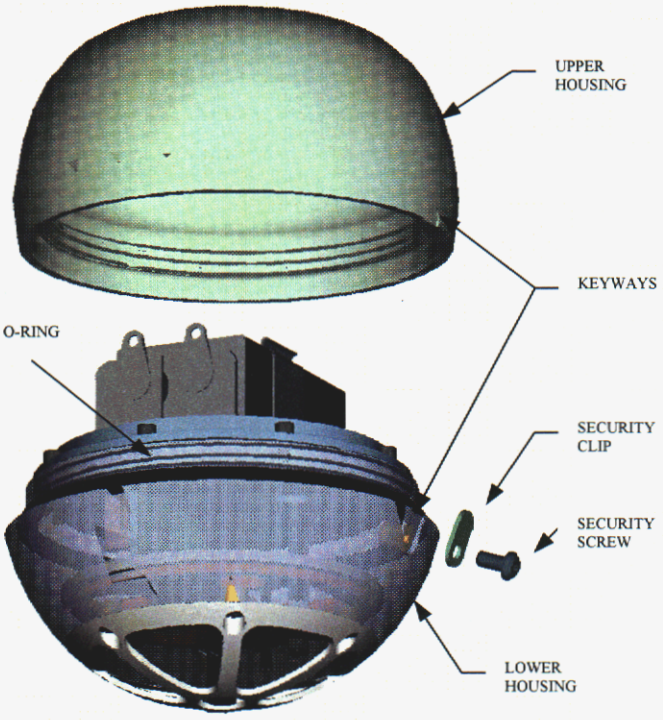


Figure 15. Exploded View of Disassembled Unit

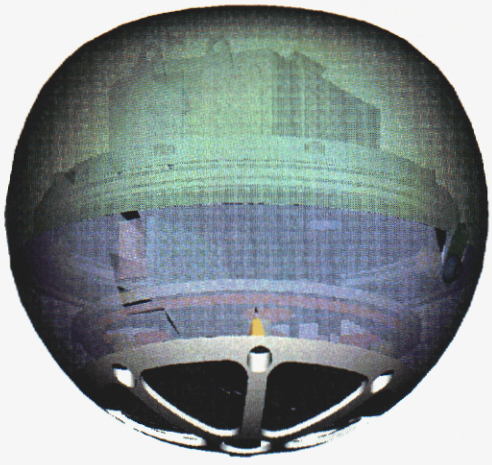


Figure 16. Assembled Unit

1. The lower half of the Sensor Ball housing is provided with an installed O-ring. A small amount of silicone-based lubricant applied to the O-ring will reduce the force needed to assemble the unit.

2. Thread the upper and lower housings together by tightening in a clockwise direction until no gap exists between the two halves, and the upper and lower keyways are aligned.

3. Place the security clip in the aligned keyways with the beveled edge facing both up and outwards. The clip should lay flat in the keyway and the holes in the clip and the lower housing should align.

4. Insert the #4-40unc button-head security screw thru the clips' hole and thread into the lower housing. Finger-tighten in the clockwise position until fully seated against the clip's surface.

5. Apply a torque value of 4 inch-pounds to the security screw using a calibrated torque wrench and the supplied insert bit.

6. To disassemble, reverse steps 2 thru 4, but remove the security screw and separate the housings in a counterclockwise direction.

Appendix K: Microcontroller Code Listing

```
1   ; File name: "ball.asm"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Main microcontroller code for the P&G Sensor Ball
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       09 Sep 1998 - Final Tested Code on Emulator and Hardware
11  ;       10 Sep 1998 - Correct RAMWRIT Timing and updated comments
12  ;       1999 - Version 1 Sensor Ball, Adapted from MilliPen.asm, TA Rohwer
13  ;       Nov 2000 - Version 2, TA Rohwer
14  ;       02 Mar 2001 - Version 3 Changes requested by P&G, ME Partridge
15  ;           Baud rate from 19.2k to 115.2kBAUD (limited by the Maxim 3221 chip)
16  ;           Average 16 samples for each measurement
17  ;           Changed WDT to 2.4 seconds for Sleep function
18  ;           User Timer1 for delays in 1.11 millisecond increments to 285ms.
19  ;           Times out from "Attention" mode (command download) in about 5 mins.
20  ;       13 Jun 2001 - Add checksum and data retransmit, ME Partridge
21  ;           These changes were needed to ensure reliable data at 115.2k Baud
22  ;        12 Aug 2001 - Version 4 Changes per P&G request, ME Partridge
23  ;           Anytime ATTENTION button is pressed, reset Baud to 19.2k.
24  ;           Change algorithm for 19.2k to Version 2 style, hope errors stop.
25  ;           Eliminate 1-min delay between records: check conductivity first.
26  ;           Remove auto-continue during data upload after 0.5 sec.
27  ;           Correct MEM_FIND call to skip a bad Block appearing blank
28  ;           Added string transmit function for messages
29  ;           Added "Smart PCM Device" commands b, c, m, n, r, s, w, z (CMD_HAND)
30  ;           Created subroutine for interface timeout check (BALL.ASM)
31  ;           Create Flash Page header subroutine (FLASH.ASM)
32  ;       11 Dec 2001 having problem with serial receive -- getting spurious
33  ;             characters.  Reset receive after each block transmitted in RD_HAND.
34  ;           Set RB4 (previously unused) as an output for ISR debug
35  ; Function:
36  ;  Initializes processor settings and clears flags & data.  Begins a loop to
37  ;  check either for commands or for conductivity above the specified threshold.
38  ;  Conductivity is checked about once a minute.  If Conductivity is detected, a
39  ;  data acquisition cycle is initiated, storing 45 minutes of data, then returns
40  ;  to the loop.  Once every 2.4 seconds nominal (1.5s minimum and 4.4s
41  ;  maximum), the ATTENTION button is checked, and if pressed a command
42  ;  processing sequence is started. This mode can be exited by selecting the
43  ;  "Re-initialize" command, or, the mode will time out about five minutes after
44  ;  the last command is completed.  The COMMAND_FLG is set after the command is
45  ;  echoed to the PC, and the verify ("V") character is received by the sensor
46  ;  ball.  Then, CMD_HAND is called to interpret commands.
47  ;
48  ;  The ATTENTION button can also terminate a data acquisition cycle for lab use.
49  ;
50  ;*****************************************************************************
51  ;
52  ; Project Files:
53  ;  ball.asm     Main Source File, contains code sections
54  ;               MAIN_INIT at location 0x00, PowerOn Vector
55  ;               MAIN_PROG, relocatable, Main code section.
56  ;               subroutines Timeout_Set, Timeout_Chk for PC interface
57  ;               subroutines WAIT_Sleep, WAIT1MS, WAITXXMS for time delays
58  ;               subroutines ADC_Read, Acquire to collect measurements
59  ;
60  ;  cmd_hand.asm        Command Handler routines, contains code sections
61  ;               CMD_PROG, relocatable
62  ;
63  ;  flash.asm    Flash Memory read, write, erase, and test subroutines
64  ;
65  ;  rd_hand.asm Memory read and upload
66  ;
67  ;  ser_hand.asm        Interrupt Service Routines (ISR) and Serial Subroutines
```

Code Listing – Ball.asm

```
68  ;              Serial port configuration subroutine with Hi/Lo Baud rate
69  ;              PERIPH_VEC at location 0x04, Peripheral Interrupt Handler
70  ;
71  ;  st_hand.asm Status Handler, uploads current Sensor Ball measurements
72  ;
73  ; Last Compile:
74  ;       Program Memory Usage:      0x0000 to 0x07C4, 4038 of 4096 Bytes
75  ;
76  ; Compiler Directives (See MPASM User's Guide) LIST - Listing Options
77  ;       Option  Default Description
78  ;       c=nnn   132     Set column width.
79  ;       n=nnn   60      Set lines per page.
80  ;       p=<type>        None    Set processor type; for example, PIC16C54.
81  ;       r=<radix>       hex     Set default radix: hex, dec, oct.
82  ;       st={ON|OFF}     On      Print symbol table in list file.
83  ;       t={ON|OFF}      Off     Truncate lines of listing (otherwise wrap).
84  ;
85          list P=16C774, C=145, T=ON
86  ;
87  ; CPU Configuration Bits (Register CONFIG, Address 0x2007):
88  ;       CP1:CP0, Code protection bits, protection off
89  ;       Vbor Brown-out voltage, set to 2.5V
90  ;       BODEN Brown-out reset, disabled
91  ;       PWRTE Power-up timer, enabled
92  ;       WDTE Watch-dog timer, enabled
93  ;       Fosc1:Fosc2, Oscillator select, RC oscillator
94          __config ( _CP_OFF & _WDT_ON & _BODEN_OFF & _VBOR_25 & _PWRTE_ON & _RC_OSC )
95  ;
96  ; Configuratioin information:
97  ;       Compiled using MPLAB Ver. 5.40
98  ;       Oscillator internal RC, set to 3.686MHz nominal
99  ;       Interface designed for SensorBall version 10 and above, 19.2k and 115.2k Baud
100 ;
101 ;*******************************************************************************
102 ;  Include files:
103         #include "P16C774.INC"        ;Standard Header File for PIC16C773
104                                       ;Includes all Register Definitions,
105                                       ;RAM Definitions, & Configuration Bits
106 ;
107 ;  Constant definitions
108         #include "ball_equ.inc"              ; Flag, Port, and Constant Definitions
109 ;
110 ;  Registers:
111         #include "ball_dat.inc"              ;RES Declarations, memory reserve
112 ;
113 ;  Subroutines in this file:
114         GLOBAL  INIT_UP               ; Reset vector routine
115         GLOBAL  Timeout_Set           ; Sets up timer to monitor PC interface
116         GLOBAL  Timeout_Chk           ; Tests if interface has timed out
117         GLOBAL  WAIT_Sleep            ; Uses Watch-Dog Timer for a 2.4 second low-power delay
118         GLOBAL  WAITXXMS              ; Delays for milliseconds set in Wreg
119         GLOBAL  WAIT1MS               ; calls WAITXXMS to create a 1 millisecond delay
120         GLOBAL  ADC_Read              ; Acquires ADC channel specified in Wreg
121         GLOBAL  Acquire               ; Puts all nine measurements in Data stack
122 ;
123 ;  Calls:
124         EXTERN  CMD_HAND              ; Cmd_Hand.asm, Processes command phrase
125         EXTERN  TX_WREG               ; Ser_Hand.asm, Transmits contents of Wreg
126         EXTERN  TX_String             ; Transmit a string message
127         EXTERN  Baud_Set              ; Ser_Hand.asm, Configures serial port
128 ;       EXTERN  SER_ECHO              ; Ser_Hand.asm, Uploads command echo
129         EXTERN  MEM_READ              ; Flash.asm, Return value from Flash memory
130         EXTERN  MEM_WRITE             ; Flash.asm, Writes Wreg value to Flash memory
131         EXTERN  MEM_FIND              ; Flash.asm, Locates free Flash memory block
132         EXTERN  MEM_RD_SET            ; Flash.asm, called from macro Record
133         EXTERN  MEM_WR_SET            ; Flash.asm, called from macro Record
134         EXTERN  MEM_Header            ; Fills out the once-per-Page values
135         EXTERN  MEM_PROGRAM           ; Flash.asm, Writes a 512-byte page in Flash
136 ;
137 ;  Public Variables:
138 ;
```

Code Listing – Ball.asm

```
139  ; Flag registers for program control and status
140         GLOBAL  PROG_FLAG               ; bits COMMAND,ADC_AVG,ACQUIRE_FLG,REINIT,CMDERROR
141         GLOBAL  SER_FLAG                ; bits XON,XOFF,ACK,NAK,CNTRLC control characters
142         GLOBAL  MEMORY_FLAG             ; Memory condition and test result flags, MEM_FULL
143         GLOBAL  SER_STATE               ; Bits indicating progress building Command Phrase
144  ;
145  ; Flash memory position data
146         GLOBAL  MEM_REC_NUM             ; Current Flash memory Record, 1 - 64
147         GLOBAL  MEM_FRM_NUM             ; Current Record Frame (Page), 0 - 255
148         GLOBAL  MEM_BAD_NUM             ; Bad blocks in Flash memory
149         GLOBAL   ADD0                   ; Flash Memory address A7 .. A0
150         GLOBAL  ADD1                    ; Flash memory address A16 .. A8
151         GLOBAL  ADD2                    ; Flash memory address A24 .. A17
152         GLOBAL  A8                      ; Indicates which 256B half of Page
153  ;
154  ; Serial input support for commands and parameters
155         GLOBAL   CMD_CHAR               ; Command character for CMD_HAND, in A-Z
156         GLOBAL  RC_CHAR                 ; ASCII Character Storage
157         GLOBAL  RC_TEMP                 ; Received Command temporary Storage
158  ;
159  ; Serial output starting location for strings in program memory
160         GLOBAL  Look_Hi                 ; Used to load PCLATH value
161         GLOBAL  Look_Lo                 ; Used to load PCL
162  ;
163  ; Analog-to-Digital results
164         GLOBAL  LSBYTE                  ; Analog Meas. LS Byte.  [A7,...A0]
165         GLOBAL  MSBYTE                  ; Analog Meas. MS Byte.  [X,X,X,X,A11,...A8]
166  ;
167  ; Time registers as data placeholders for an eventual Real-Time Clock chip
168  ; Used now as crude recording of time in milliseconds since last reset.
169  ; (units in parenthesis Placeholder for Real-Time Clock)
170         GLOBAL  TIME0                   ; LS Byte, Time in milliseconds (Seconds)
171         GLOBAL  TIME1                   ;  . . . next significant byte (Minutes)
172         GLOBAL  TIME2                   ;  . . . next significant byte (Hours)
173         GLOBAL  TIME3                   ;  . . . next significant byte (Days from Reset)
174         GLOBAL  TIME4                   ; =zero, not used yet (Month)
175         GLOBAL  TIME5                   ; =zero, not used yet (Year)
176  ;
177  ; Attention mode time-out registers
178         GLOBAL  CMD_TIMEL               ; Time-out LSByte, 285ms / bit
179         GLOBAL  CMD_TIMEH               ; Time-out MSByte
180  ;
181  ; Temporary values used when reading back files
182         GLOBAL  R_Sync_Byte0
183         GLOBAL  R_Sync_Byte1
184         GLOBAL  R_MEM_REC_NUM           ; Read value of Record #, Temporary value to find Record
185         GLOBAL  R_MEM_FRM_NUM
186         GLOBAL  R_ADD1                  ; Used in MEM_FIND, sequential blanks
187         GLOBAL  R_ADD2
188         GLOBAL  R_TIME3
189         GLOBAL  R_TIME2
190         GLOBAL  R_TIME1
191         GLOBAL  R_TIME0
192  ;
193  ; Loop counters, other temporary values
194         GLOBAL   MeasSet_Cnt            ; Loop counter, sets of 9 analog measurements
195         GLOBAL  Channel_Cnt             ; Loop counter, analog channels 0 to 9 skip 7
196         GLOBAL  Sleep_Cnt               ; Loop counter, sleep cycles between conductivity check
197         GLOBAL  Loop_Cnt                ; Generic Loop Counter
198         GLOBAL  TEMP                    ; Temporary Flash value, MEMORY_FLAG (Don't use for
199  Interrupt routines)
200         GLOBAL  PATTERN                 ; Test pattern passed in Wreg to MEM_TEST
201         GLOBAL  CKSM0                   ; Checksum LS Byte, used in memory to PC transfer
202         GLOBAL  CKSM1                   ; Checksum MS Byte
203         GLOBAL  BLCK0                   ; Count of Frames uploaded, LS Byte
204         GLOBAL  BLCK1                   ; Count of Frames uploaded, MS Byte
205  ;
206  ; Registers to push data during interrupt, accessible regardless of Memory Bank selected.
207         GLOBAL  STACK_Wreg              ; Working Register holding during interrupts
208         GLOBAL  STACK_Status            ; Status Register holding during interrupts
209         GLOBAL  STACK_FSR               ; Indirect pointer holding during interrupts
```

```
210         GLOBAL  TEMP_INT                ; Temporary for interrupt only
211         GLOBAL  LOOP_INT                ; Temporary for interrupt only
212     ;
213     ; Command Parameter Stack starting location
214     ; Note: Code does not check to see if stack exceeded, that is beyond address 0x6F
215         GLOBAL  CMD_PARAMS              ; Count of command parameters received
216     ;
217     ; String starting locations transmitting strings (TX_String subroutine)
218         GLOBAL  List_WhoID              ; Unit ID string
219         GLOBAL  List_Commands           ; Valid command character string
220         GLOBAL  List_Data00             ; First channel name
221         GLOBAL  List_Slft00             ; First self-test bit name
222         GLOBAL  List_Undef              ; Request undefined name
223         GLOBAL  List_Fail_Msg           ; String for Memory failure message
224         GLOBAL  List_Stat_Msg           ; String for Memory fill / test progress
225         GLOBAL  List_Eras_Msg           ; String for Memory erase progress
226         GLOBAL  List_Mem_Flag           ; String for Memory Flag value
227     ;
228     ;  Macros:
229     ;       none                        ; macro definitions -- not used
230     ;
231     ;  Interrupts:
232     ;       Enables Global, Peripheral, and Serial Port Interrupts
233     ;
234     ;*****************************************************************************
235     ;************************* Message string definition file ********************
236     ;*****************************************************************************
237         #include "ball_msg.inc"             ; Text strings for messages
238     ;
239     ;*****************************************************************************
240     ;************************* Power-on Reset Start Vector ***********************
241     ;*****************************************************************************
242     MAIN_INIT       CODE    0x00            ;PowerOn Vector
243                     Goto    INIT_UP
244     ;
245     ;*****************************************************************************
246     ;************************* Main program code, relocatable ********************
247     ;*****************************************************************************
248     MAIN_PROG       CODE
249     ;
250     ;
251     ; Power-on Reset Vector, Initialize Microprocessor
252     INIT_UP
253         Clrw
254         Movwf   STATUS                  ; Cannot directly clear arithmetic flags
255                                         ; Selects Bank 0 memory
256         Clrf    INTCON                  ; Disable all interrupts
257         Clrf    PIR1                    ; Clear peripheral interrupt flag bits
258         Clrf    PIR2                    ; Clear CCP2, etc. interrupt flag bits
259         Bsf     STATUS, RP0             ; Select Bank 1 Memory
260         Clrf    PIE1                    ; Clear individual peripheral enable bits
261         Clrf    PIE2                    ; Clear CCP2, etc. interrupt enable bits
262     ;
263     ; I/O Ports (See PIC16C77X Data Book, sect. 3.0 )
264         Bcf     STATUS, RP0             ; Select Bank 0 memory
265         Clrf    PORTA                   ; Initialize ports by clearing output
266         Clrf    PORTB                   ;   data latches
267         Clrf    PORTC
268         Clrf    PORTD
269         Clrf    PORTE
270     ;
271     ; Set Port direction
272         Bsf     STATUS, RP0             ; Select Bank 1 Memory
273         Movlw   0xFF
274         Movwf   TRISA                   ; Set RA(5:0) Inputs, A is 6-bit wide port)
275         Movlw   0x0D
276         Movwf   TRISB                   ; Set RB(7:4,1) outputs, RB(3:2,0) inputs
277         Movlw   0xC5                    ; TRISC<7:6> set to configure TX & RX
278         Movwf   TRISC                   ; Set RC(7:6,2,0) inputs, RC(5:3,1) outputs
279         Movlw   0x00
280         Movwf   TRISD                   ; Set RD(7:0) Outputs
```

```
281         Movlw   0x07
282         Movwf   TRISE                   ; Set RE(2:0) Inputs,
283 ;                                       ; If bit 4=1, Conf. Port D as Slave Port
284 ;
285 ; Set Port outputs
286 INIT_PORT
287         Bcf     STATUS, RP0             ; Select Bank 0 memory
288         Bcf     PORTB, WP               ; WP~ lo, Flash Memory write protected
289         Bsf     PORTB, POWER            ; POWER on analog, RS-232, & Flash
290         Bsf     PORTC, LED              ; LED hi turns LED on
291         Bsf     PORTC, CE               ; CE~ high, Flash Memory control lines
292         Bsf     PORTC, RE               ; RE~ high
293         Bsf     PORTC, WE               ; WE~ high
294 ;
295 ; Analog-to-Digital Converter Module (See PIC16C77x Data Book, sect. 11.0)
296 ; ADCON0 Register in Bank 0 Memory, ADCON1 in Bank 1
297 INIT_A2D
298         Bsf     STATUS, RP0             ; Select Bank 1 Memory
299         Clrf    ADCON1
300         Bsf     ADCON1, ADFM            ; Right justify A/D result
301 ;       Bcf     ADCON1, VCFG2           ; Select voltage reference
302 ;       Bcf     ADCON1, VCFG1           ;   using supply and ground
303 ;       Bcf     ADCON1, VCFG0
304 ;       Bcf     ADCON1, PCFG3           ; Select to use all 10 analog channels
305 ;       Bcf     ADCON1, PCFG2           ;   even though Analog7 is not connected
306 ;       Bcf     ADCON1, PCFG1
307 ;       Bcf     ADCON1, PCFG0
308 ;
309         Bcf     STATUS, RP0             ; Select Bank 0 Memory
310         Bsf     ADCON0, ADCS0           ; Set A/D conversion clock Fosc/8
311         Bcf     ADCON0, ADCS1           ;
312         Bsf     ADCON0, ADON            ; Turn on ADC
313 ;
314 ; Watch-dog Timer set up
315 ; See PIC16C77x Data Book, Section 4.0 Timer (for prescaler), Section 12.13 (Sleep)
316 INIT_WDT
317         Bsf     STATUS, RP0             ; Select Bank 1 Memory
318         Bsf     OPTION_REG, PSA                 ; Direct prescaler for WDT use
319         Bsf     OPTION_REG, PS2                 ; Use Prescale 1:128, so WDT is
320         Bsf     OPTION_REG, PS1                 ;   about 2.4 seconds
321         Bsf     OPTION_REG, PS0                 ;   and so is Sleep
322         Clrwdt
323 ;
324 ; Timer 1 set up - about 1 millisecond per clock tick (1.11ms @ 3.686MHz osc.)
325 INIT_TMR1
326         Bcf     STATUS, RP0             ; Select Bank 0 Memory
327         Bcf     T1CON, TMR1ON           ; Stop Timer 1 if running
328         Bcf     T1CON, TMR1CS           ; Timer 1 clock source Fosc/4
329         Bsf     T1CON, NOT_T1SYNC       ; Sleep input not selected
330         Bsf     T1CON, T1CKPS1          ; Set Prescale to 1:4 so Fosc/16
331         Bcf     T1CON, T1CKPS0
332         Clrf    TMR1H                   ; Clear timer contents, hi byte
333         Clrf    TMR1L                   ; Clear timer contents, low byte
334 ;
335 ; Initialize user-defined registers in Memory Bank 0, including Parameter Stack.
336 INIT_CLRREG
337         Bcf     STATUS, RP0             ; Select Bank 0 Memory
338         Movlw   0x20                    ; Starting address for user memory
339         Movwf   FSR                     ; Stack pointer
340         Movlw   0x50                    ; 0x50 locations, 0x020-0x06F
341         Movwf   LOOP_INT                ; Temporary used only by interrupt routine
342 INIT_Reg_Loop
343         Clrf    INDF
344         Incf    FSR, f
345         Decfsz  LOOP_INT, f             ; Loop counter, number of user registers
346         Goto    INIT_Reg_Loop
347 ;
348 ; Give the power about a quarter-second to stabilize
349         Movlw   0x00                    ; Prepare for delay, value is 0x100
350         Call    WAITXXMS                ; Wait for 285ms
351 ;
```

```
352   ; Find the first blank section of Flash memory, and read the last Record
353   ; number used if the memory is not totally erased.
354         Call    MEM_FIND
355   ;
356   ; Transmit characters on USART, used to verify RS-232 interface awake for testing
357   INIT_XMT
358         Clrw                        ; Clear Wreg
359         Call    Baud_Set            ; if Wreg=0, sets 19.2k Baud
360                                     ; if Wreg<>0, sets 115.2k Baud
361         Clrwdt                      ; Watch-Dog Timer reset
362         Movlw   "w"                 ; Get character to select Who command
363         Movwf   CMD_CHAR            ; Variable used by CMD_HAND
364         Call    CMD_HAND            ; Calls the Who command
365   ;
366   ;*****************************************************************************
367   ; Start of main command check - conductivity trigger check - record data - sleep loop
368   MAIN_START
369         Bcf     STATUS, RP0         ; Select Bank 0 Memory
370         Clrf    INTCON              ; Disable all interrupts
371         Clrf    Sleep_Cnt           ; Sleep_Cnt Zero = check conductivity
372   ;
373   MAIN_LOOP
374         Bcf     STATUS, RP0         ; Select Bank 0 Memory
375         Clrwdt                      ; Watch-Dog Timer is about 2.4 seconds
376   ;
377   ; Check to see if computer command, started by pushing Attention button
378         Btfsc   PORTB, ATTENTION    ; Bit Test of Command Flag
379         Goto    ATTEN_START         ; Button pushed, begin processing commands
380   ;
381   ; If the Flash Memory is full (32MB), no point trying to record more data
382         Btfsc   MEMORY_FLAG, MEM_FULL ; See if have full memory
383         Goto    MAIN_SLEEP          ;  Yes, just minimize power consumption
384   ;
385   ; Want to check Conductivity first before Sleep, so check the delay counter.
386         Movf    Sleep_Cnt, w        ; Get current count down value
387         Btfsc   STATUS, Z           ; See if count-down = zero
388         Goto    MAIN_ACQ_CHECK      ;  Yes & power is still on, so check cond.
389   ;
390   ; Microcontroller Sleep mode saves power. Duration of the Sleep mode is set by
391   ; the Watch-Dog Timer (WDT) delay multiplied by the Prescale from OPTION_REG.
392   ; In the PIC16C77x Timing Diagrams and Specifications, WDT minimum is 7ms,
393   ; typical 18ms, and maximum 33ms.  Because the Prescale has been set to 1:128,
394   ; one low-power Sleep cycle is 2.4 seconds.
395   ;
396   MAIN_SLEEP
397         Bcf     ADCON0, ADON        ; Turn off ADC
398         Bcf     PORTB, POWER        ; Power off analog, Flash, RS-232
399         Bcf     PORTC, LED          ; LED low turns the LED off
400         Call    WAIT_Sleep          ; Wait in low-power mode
401         Decfsz  Sleep_Cnt, f        ; See if Sleep cycles count = one minute (approximately)
402         Goto    MAIN_LOOP           ;   No, wait some more
403   ;
404   ; Wake up Sensor Ball to check conductivity
405         Bsf     PORTB, POWER        ; POWER on analog & Flash memory power
406         Bsf     PORTC, LED          ; LED high, LED on
407         Bsf     ADCON0, ADON        ; Turn on ADC
408   ;
409   ; The conductivity circuitry takes a little over 500ms to stabilize, so wait
410   ; for about a second before checking the value.  Each delay below uses a delay
411   ; value of 0x00, which is translated as 0x100 and about a 285ms delay.
412         Movlw   0x00                ; Prepare for delay, value is 0x100
413         Call    WAITXXMS            ; Wait for 285ms, returns Wreg=0
414         Bcf     PORTC, LED          ; LED off, was just a quick flash on
415         Call    WAITXXMS
416         Call    WAITXXMS
417         Call    WAITXXMS
418   ;
419   ; Check to see if Conductivity measurement above threshold to begin measurement recording cycle
420   MAIN_ACQ_CHECK
421   ;
422   ; ADC_Read subroutine returns the value for the channel specified in Wreg into
```

```
423   ; registers LSBYTE and MSBYTE.  MSBYTE is also returned in Wreg.  If the ADC_AVG
424   ; bit is set in PROG_FLAG, 16-sample averaging is used.
425         Bsf     PROG_FLAG, ADC_AVG      ; Set for 16-bit averaging
426         Movlw   0x04                    ; Acquire ADC channel 4 = Conductivity
427         Call    ADC_Read                ; ADC average in regs. LSBYTE and MSBYTE
428   ;
429   ; A result of measured conductivity greater than or equal to Conduct_Thresh
430   ; will set the carry bit in the Subwf instruction below
431         Movlw   Conduct_Thresh          ; Conductivity threshold limit
432         Subwf   MSBYTE, w               ; Subtracts reading from threshold
433         Btfsc   STATUS, C               ; C set says conductivity > threshold
434         Goto    MAIN_RECORD_BEGIN       ; Threshold exceeded, begin record
435   ;
436   ; Threshold not exceeded, reset one-minute delay in low-power mode
437         Movlw   Sleep_Cycles            ; = # of 2.4 second long Sleep cycles
438                                         ;   between conductivity checks
439         Movwf   Sleep_Cnt               ; Sleep Cycle Down count from 24 is
440                                         ;   about one minute total
441         Goto    MAIN_LOOP
442   ;
443   MAIN_RECORD_BEGIN
444   ; Valid arm signal based on Conductivity detected, Record data to Flash.  Can
445   ; be interrupted at the end of each 512-byte page by pressing the ATTENTION
446   ; button (may need to hold for 10 sec).
447   ;
448   ; Starting with the next 16kB block record pointed to by ADD1 and ADD2, read
449   ; the first two bytes to verify that the location is blank.  Blank memory
450   ; contains 0xFFFF, a used block begins with 0xEB90, and bad memory may have
451   ; something other than these two.  If not blank, (perhaps a RESET occurred),
452   ; search the start of each 16kB Block until a blank one is found.
453   ;
454         Call    MEM_FIND                ; Search next 128kB Record by 16kB Blocks
455         Btfsc   MEMORY_FLAG, MEM_FULL   ; See if have captured 32MB data records
456         Goto    MAIN_SLEEP              ;   Yes, just minimize power consumption
457   ;
458   ; The Frame counter MEM_FRM_NUM is cleared (set to 0x00, which is effectively
459   ; 0x100) to record 256 frames of 512 bytes each (128kB total).
460   ;
461         Bcf     PROG_FLAG, ACQUIRE_FLG ; Clear after acquire, set by CMD_HAND
462         Clrf    MEM_FRM_NUM             ; Clear Memory Page count
463         Incf    MEM_REC_NUM, f          ; Add one to number of records stored
464   ;
465   RECORD_LOOP
466   ;
467   ; Check to see if request to interrupt record collection process
468         Btfsc   PORTB, ATTENTION        ; Bit Test of Command Flag
469         Goto    ATTEN_START             ; Button pushed, abort data acq. cycle
470   ;
471   ; Set up memory to accept data at the 512-Byte Page address pointed to by
472   ; address Bytes ADD1 and 2.
473         Call    MEM_WR_SET
474   ;
475   ; Write information at start of memory Page.  Time, etc. written only once per page.
476         Call    MEM_Header
477   ;
478   ; Initialize measurement counter to record 28 analog sets per Frame
479         Movlw   0x1C                    ; Frame contains 28 analog meas. sets
480         Movwf   MeasSet_Cnt             ;   initialize counter variable
481   ;
482   ; Loop to build one data Frame, which also occupies one Flash memory Page
483   FRAME_LOOP
484         Clrwdt                          ; Reset 2.4 second Watch-dog Timer
485         Bsf     PORTC, LED              ; LED on during data acquire.
486   ;
487   ; Acquire one measurement set.  Reads all nine value pairs into the Data stack.
488         Bcf     PROG_FLAG, ADC_AVG      ; Set for no averaging
489         Call    Acquire                 ; Get all measurements in Data stack
490   ;
491   ; Now store the measurement set into memory
492         Movlw   Data_Start              ; top of memory stack
493         Movwf   FSR                     ;  . . . loaded into the stack pointer
```

```
494         Movlw   0x12                ; 9 channels * 2 Bytes each = 18
495         Movwf   Channel_Cnt         ; Re-use ADC_Read loop counter
496     ;
497     MEAS_LOOP
498         Movf    INDF, w             ; recover value from stack
499         Call    MEM_WRITE           ; Store in Flash memory
500         Incf    FSR, f              ; adjust stack pointer
501         Decfsz  Channel_Cnt, f
502         Goto    MEAS_LOOP
503     ;
504     ;
505     ; End of measurement loop - Wait 368 milliseconds.  Using 45 minutes data
506     ; collection per record and 7168 samples / record, so 376.7 milliseconds per
507     ; sample.  Sampling 9 analog channels takes about 1.1ms per channel, essentially
508     ; one delay timer clock tick each when the oscillator frequency Fosc = 3.686MHz.
509     ; So, delay for 330 additional delay timer ticks, = 0x14A, = 366.7 milliseconds
510     ;
511         Bcf     PORTC, LED          ; LED off, brief flash during acquire
512         Clrw                        ; Starting with Wreg = 0 gives 256
513         Call    WAITXXMS            ;    counts, about 285 ms delay
514         Movlw   0x4A                ; Gives 74 counts, or
515         Call    WAITXXMS            ;   about 82 ms delay
516     ;
517     ; Check if done with frame
518         Decfsz  MeasSet_Cnt, f      ; sub-frame counter, want 28 sets
519         Goto    FRAME_LOOP          ; not done with 28 sets yet
520     ;
521     ; End of Frame loop. Command transfer of Flash internal buffer to memory
522     FRAME_WRITE
523         Clrwdt                      ; Reset 2.4 second Watch-dog Timer
524         Call    MEM_PROGRAM         ; Transfer internal buffer to Flash
525     ; Note: Maximum write delay is 1 millisecond.  The call waits until done.
526     ;
527     ; Increment memory address to next 512-Byte Page.  Add0 is the Byte location
528     ; within the Page, so is not incremented.
529     FRAME_INCR
530         Incf    ADD1, f             ; Increment to next Page
531         Btfsc   STATUS, Z           ; See if rolled over, overflow
532         Incf    ADD2, f             ;   yes, increment next address
533     ;
534     ; Increment frame counter
535         Incfsz  MEM_FRM_NUM, f      ; Frame counter, rolls to 0x00 from 0xFF
536         Goto    RECORD_LOOP
537     ;
538     ; 128kB Record complete - End of memory acquisition / write.  The MEM_INCR
539     ; function sets the MEM_FULL flag if it incremented beyond the last Page of the
540     ; 32MB memory, indicating that all acquisition records are complete.
541     ;
542         Goto    MAIN_START
543     ;
544     ; - End of command check - conductivity trigger check - record data - sleep loop
545     ;*****************************************************************************
546     ;
547     ; Command handler loop.  Entered when the "ATTENTION" button is pressed.  This
548     ; approach is used because the loop above includes a Sleep function, which
549     ; disables the serial interface.  The serial interface chip, Flash memory, and
550     ; all analog circuits are also powered down before Sleep is run.  This loop can
551     ; be exited by requesting the reinitialize command, or waiting about 5 minutes
552     ; for this mode to time out.
553     ;
554     ATTEN_START
555     ; Make sure using 19.2k Baud communication
556         Clrw                        ; Clear Wreg
557         Call    Baud_Set            ; if Wreg=0, sets 19.2k Baud
558                                     ; if Wreg<>0, sets 115.2k Baud
559     ;
560     ; Set up timer for ATTENTION mode time-out
561     ATTEN_TIMER
562         Call    Timeout_Set
563         Bcf     PROG_FLAG, LED_BLINK   ; Keep LED steady, do dot char. heartbeat
564     ;
```

```
565  ; Turn on LED to indicate ATTENTION mode started, and power aux. circuitry
566          Bsf     PORTC, LED              ; LED high, LED illuminated
567          Bsf     PORTB, POWER            ; POWER high, analog, RS-232, & Flash on
568          Bsf     ADCON0, ADON            ; Turn on ADC
569  ;
570  ; Initialize serial input state registers
571          Clrf    PROG_FLAG
572          Clrf    SER_FLAG
573          Clrf    SER_STATE
574  ;
575  ATTEN_LOOP
576          Clrwdt                          ; Watch-Dog Timer is about 2.4 seconds
577          Call    Timeout_Chk             ; Test if interface has timed out
578          Btfss   STATUS, Z               ; Wreg non-zero if timed out
579          Goto    ATTEN_END               ; PC interface timed out
580  ;
581  ; The Serial Interrupt Handler captures and verifies a command phrase, then sets
582  ; the Command flag bit set and moves the command character to CMD_CHAR.
583  ATTEN_CMD
584          Btfsc   PORTB, ATTENTION        ; Bit Test of Command Flag
585          Goto    ATTEN_START             ; Button pushed, reset interface
586          Btfss   PROG_FLAG, COMMAND_FLG ; Bit set when command available
587          Goto    ATTEN_LOOP              ; No command, repeat arm/command check
588  ;
589  ; Have a command character available, process
590          Call    CMD_HAND                ; Process the Command
591          Btfsc   PROG_FLAG, REINIT_FLG  ; Bit Test on ReInit Flag
592          Goto    INIT_UP                 ;   ==1, Restart Program, Reinitialize
593          Btfsc   PROG_FLAG, ACQUIRE_FLG ; Request to acquire one Record?
594          Goto    MAIN_RECORD_BEGIN       ; . . . then start acquisition
595          Goto    ATTEN_TIMER             ; Reset command time-out, and loop
596  ;
597  ; End command. Disable all interrupts.  Returns to conductivity check loop
598  ; without resetting processor.
599  ATTEN_END
600          Goto    MAIN_START
601  ;
602  ;*******************************************************************************
603  ;*******************************************************************************
604  ;       Interface Time-Out Set-up Subroutine
605  ; Function:
606  ;  Timeout_Set - Sets up use of Timer1 to monitor whether the PC interface has
607  ;       been inactive beyond a specified time.
608  ;
609  ; Calls:
610  ;  none
611  ;
612  ; Registers:
613  ;  TIME0, TIME1, TIME2, TIME3 - Time since Sensor Ball reset
614  ;  CMD_TIMEH, CMD_TIMEL - time since last interface input
615  ;
616  ; Interrupts:
617  ;  Interrupts not used or changed in this subroutine
618  ;
619  ; Return:
620  ;  WREG=0
621  ;*******************************************************************************
622  ;
623  Timeout_Set
624          Bcf     STATUS, RP0             ; Select Memory Bank 0
625          Bcf     T1CON, TMR1ON           ; Stop Timer 1 if running
626          Bcf     PIR1, TMR1IF            ; Clear interrupt flag
627          Clrf    TMR1L                   ; Clear Timer 1 counters
628          Clrf    TMR1H                   ; Set for maximum time, about 285ms
629          Bsf     T1CON, TMR1ON           ; Start Timer 1
630          Clrf    CMD_TIMEL               ; Clear time-out registers
631          Clrf    CMD_TIMEH
632          Retlw   0                       ; Done with set-up
633  ;
634  ;*******************************************************************************
635  ;*******************************************************************************
```

```
636  ;       Interface Time-Out Check Subroutine
637  ; Function:
638  ;  Timeout_Chk - Tests whether Timer1 has rolled over.  If yes, checks if the
639  ;       total time elapsed exceeds Constant Atten_Timeout.  Adjusts the TIME
640  ;       registers in the process.
641  ;
642  ; Calls:
643  ;  none
644  ;
645  ; Registers:
646  ;  TIME0, TIME1, TIME2, TIME3 - Time since Sensor Ball reset
647  ;  CMD_TIMEH, CMD_TIMEL - time since last interface input
648  ;
649  ; Interrupts:
650  ;  Interrupts not used or changed in this subroutine
651  ;
652  ; Return:
653  ;  WREG=0
654  ;*******************************************************************************
655  ;
656  Timeout_Chk
657         Bcf     STATUS, RP0             ; Select Memory Bank 0
658         Clrw                            ; Prepare for Wreg=0 and Z bit set
659         Btfss   PIR1, TMR1IF            ; check timer completion
660         Return                          ; Timer not elapsed, return with Wreg=0
661  ;
662  ; Attention time-out counter increment expired.  Update time registers.  Timer1
663  ; continues to operate until stopped, so no need to set up again for the
664  ; subsequent 285 millisecond increments.
665  Timeout_Update
666         Bcf     PIR1, TMR1IF            ; Clear interrupt flag, timer still running
667         Movlw   0x01                    ; set to increment time registers
668         Addwf   TIME1, f                ;  . . . about 285ms / Time1 bit
669         Btfsc   STATUS, C
670         Addwf   TIME2, f
671         Btfsc   STATUS, C
672         Addwf   TIME3, f
673         Addwf   CMD_TIMEL, f            ; Increment time-out regs, 285ms / bit
674         Btfsc   STATUS, C
675         Incf    CMD_TIMEH, f            ; Incremented every 73 sec (256 * 285ms)
676  ;
677  ; Make LED blink if requested.  Otherwise do dot character
678         Btfss   PROG_FLAG, LED_BLINK
679         Goto    Timeout_Dot
680  ;
681  ; Blink LED with distinctive pause-blink-blink pattern, 2-second period
682  Timeout_LED
683         Movf    CMD_TIMEL, w
684         Andlw   0x04                    ; Mask to get every four counts, 1 sec
685         Sublw   0x04                    ; Four counts * 285ms
686         Btfsc   STATUS, Z               ; check if equal
687         Goto    Timeout_Test            ; Not even increment of four
688  ;
689  ; Make LED blink
690         Movlw   0x02                    ; LED is PORT C, line 1
691         Xorwf   PORTC, f                ; make LED blink
692         Goto    Timeout_Test
693  ;
694  ; Debug - send out dot every 4 seconds
695  Timeout_Dot
696         Movf    CMD_TIMEL, w
697         Andlw   0x0F                    ; retain bits 0 to 3
698         Sublw   0x08                    ; count on 4s increments
699         Btfss   STATUS, Z               ; see if equal
700         Goto    Timeout_Test
701  ;
702  ; On 4-second increment, send heartbeat character to PC
703         Movlw   "."                     ; Load dot character
704         Call    TX_WREG                 ; Transmit serial data and return
705  ;
706  ; Check if timed out
```

```
707   Timeout_Test
708           Movlw   Atten_Timeout          ; Maximum wait time without a command
709           Subwf   CMD_TIMEH, w           ; Compare with time accumulated
710           Btfsc   STATUS, C              ; If has timed out, negative (carry set)
711           Goto    Timeout                ; Carry set, timed out
712           Clrw                           ; Set Zero bit for Wreg
713           Return                         ; Carry clear, not timed out yet
714   ;
715   ; PC interface has timed out.  Stop the timer, and return a non-zero Wreg.
716   ;
717   Timeout
718           Iorwf   0xFF, w                ; Indicate timeout with non-zero Wreg
719           Return
720   ;
721   ;*****************************************************************************
722   ;*****************************************************************************
723   ;       WAIT Subroutine                                                   TAR
724   ; Function:
725   ;  WAIT_Sleep - Long (about 2.4 second) low-power delay using Sleep function
726   ;  WAIT1MS - 1 millisecond delay using Timer1
727   ;  WAITXXMS - variable delay read from Wreg and using Timer1
728   ;
729   ; Calls:
730   ;  none
731   ;
732   ; Registers:
733   ;  TIME0, TIME1, TIME2, TIME3
734   ;
735   ; Interrupts:
736   ;  Serial Interrupts not processed during Sleep
737   ;  Interrupts not used or changed in this subroutine
738   ;
739   ; Return:
740   ;  WREG=0
741   ;*****************************************************************************
742   ;
743   ; The Sleep function uses the internal WDT oscillator, which can be somewhat
744   ; variable.  Nominal time with the current configuration is 2.4 seconds, but it
745   ; can range from 1.5 to 4.4s.
746   ;
747   WAIT_Sleep                           ; Wait 2.4 seconds
748   ;
749   ; Increment the TIME2 to TIME0 counter by 2400 as a crude count of milliseconds
750   ; elapsed.  The Sleep function actually has a wide variation because an
751   ; internal microprocessor oscillator circuit generates the clock for the
752   ; Watch-Dog Timer.  TIME register maximum is about 50 days.
753           Movlw   0x60                   ; Adding 0x0960 = 2400ms to Time regs
754           Addwf   TIME0, f
755           Movlw   0x09                   ; Prepare for Time 1 addition
756           Btfsc   STATUS, C              ; If Time 0 overflowed, increment Time 1
757           Movlw   0x0A                   ;   . . . by one more
758           Addwf   TIME1, f
759           Movlw   0x01                   ; Prepare increment if needed
760           Btfsc   STATUS, C              ; Look for overflow on Time 1
761           Addwf   TIME2, f               ; Time 1 overflow, increment Time 2
762           Btfsc   STATUS, C              ; Still clear from Time 1, or clear/set by 2
763           Incf    TIME3, f               ; Time 2 overflow, increment Time 3
764   ;
765   WAIT_SL_GO
766           Clrwdt                         ; Watch-Dog Timer is about 2.4 seconds
767           Sleep                          ; Low power duration is Watch-dog Timer
768           Retlw   0
769   ;
770   ;*****************************************************************************
771   ; Entry at 1.1ms delay (WAIT1MS) or variable delay (WAITXXMS with delay in
772   ; Wreg).  About 1.11 milliseconds per count (with Fosc=3.686MHz) of the high-
773   ; order timer byte because the Timer 1 Prescale is set to 1:4 (Fosc/4 then
774   ; prescale by 1/4).  This means the low-order byte is incremented every four
775   ; instruction cycles, or roughly 4us.  The high-order timer byte is the
776   ; complemented value of the millisecond delay requested, then incremented by 1
777   ; because timer counts up to 0xFF, and stops on 0x00 (roll-over). Maximum input
```

```
778  ; is 0x00, represents about 284ms delay, and minimum 0x01, a delay of 1.11ms.
779  ;
780  WAIT1MS                              ; Wait 1.1ms (with Fosc=3.686MHz)
781         Movlw   0x01                  ; set Wreg with delay value
782  ;
783  WAITXXMS
784         Clrwdt                        ; Watch-Dog Timer is about 2.4 seconds
785         Movwf   TEMP                  ; Millisecond delay / time increment value
786         Bcf     STATUS, RP0           ; Select Memory Bank 0
787         Bcf     T1CON, TMR1ON         ; Stop Timer 1 if running
788         Bcf     PIR1, TMR1IF          ; Clear interrupt flag
789         Clrf    TMR1L
790         Movf    TEMP, w               ; Recover delay to set/clear Z on status
791         Btfsc   STATUS, Z             ; Value is 0x00, really represents 0x100
792         Goto    WAIT_TM_ADJ           ; Just increment next significant TIME byte
793         Addwf   TIME0, f              ; Value in Wreg not zero, so add to LS byte
794         Btfss   STATUS, C             ; Check if carry from TIME0 addition
795         Goto    WAIT_COM              ; No carry, no need to adjust next bytes
796  WAIT_TM_ADJ
797         Movlw   0x01                  ; Incr command does not alter Carry bit
798         Addwf   TIME1, f              ; Increment time registers, ~1.11ms/bit
799         Btfsc   STATUS, C             ; Look for overflow on Time 1
800         Addwf   TIME2, f              ; Time 1 overflow, increment Time 2
801         Btfsc   STATUS, C             ; Still clear from Time 1, or clear/set by 2
802         Incf    TIME3, f              ; Time 2 overflow, increment Time 3
803  ;
804  ; Inversion and increment to set up TMR1H up-counter
805  WAIT_COM
806         Comf    TEMP, f               ; Millisecond delay value, invert
807         Incf    TEMP, w               ; Add one, becomes value for up-counter
808         Movwf   TMR1H                 ; Variable delay, = 1.112ms * Wreg
809         Bsf     T1CON, TMR1ON         ; Start Timer 1
810  WAITXX_LOOP
811         Btfss   PIR1, TMR1IF          ; Test if Timer 1 overflow occurred
812         Goto    WAITXX_LOOP           ; Overflow still clear, loop again
813  ;
814         Bcf     T1CON, TMR1ON         ; Stop Timer 1
815         Bcf     PIR1, TMR1IF          ; Clear interrupt flag
816         Retlw   0                     ; Delay over, return with Wreg=0
817  ;
818  ;*******************************************************************************
819  ;*******************************************************************************
820  ;       ADC Read with Average Subroutine
821  ;               4/16/2001 ME Partridge
822  ; Function:
823  ;  Sets up the ADC multiplexer channel pointed to in Register W.  Calls the
824  ;  WAIT1MS function to allow time for the ADC multiplexer to stabilize, then
825  ;  converts and adds the programmed number of samples together.  Finally,
826  ;  divides (shifts right) the result by the programmed number of bits.
827  ;
828  ; Note: the ADC is a 12 bit result, but is right justified so the upper nibble
829  ;  of MSBYTE0 is zero.
830  ;
831  ; Calls:
832  ;  WAIT1MS      Delays one millisecond
833  ;
834  ; Macros used:
835  ;  none.
836  ;
837  ; Registers:
838  ;  PROG_FLAG    If ADC_AVG is set, take a 16-bit average of measurement
839  ;  Wreg         On entry, has the ADC channel to convert
840  ;  LSBYTE       On exit, contains the LS byte of the averaged data
841  ;  MSBYTE       On exit, contains the MS byte of the averaged data.
842  ;
843  ;  Interrupts:
844  ;       None used.
845  ;
846  ;  Return:
847  ;       LSBYTE          = On exit, contains the LS byte of the averaged data
848  ;       MSBYTE          = On exit, contains the MS byte of the averaged data
```

```
;       Wreg            = On exit, MS byte of data.
;
;  Revisions:
;       3/3/2001 First revision
;       4/16/2001 Change to macro, make averaging selectable
;       9/16/2001 Change to subroutine with ADC_AVG averaging flag
;*****************************************************************************
;
; Set ADC multiplexer channel based on channel identified in Register W
; ADCON0 Bits, MSB to LSB: ADCS1 ADCS0 CHS2 CHS1 CHS0 GO/DONE CHS3 ADON
;
ADC_Read
        Movwf   LSBYTE                  ; Use as temporary for channel select
        Movlw   0xC5                    ; Mask to clear CHS bits
        Andwf   ADCON0, f               ; Clear channel selection
;
        Btfsc   LSBYTE, 0
        Bsf     ADCON0, CHS0            ; LS bit was a 1
;
        Btfsc   LSBYTE, 1
        Bsf     ADCON0, CHS1            ; Bit 1 was a 1
;
        Btfsc   LSBYTE, 2
        Bsf     ADCON0, CHS2            ; Bit 2 was a 1
;
        Btfsc   LSBYTE, 3
        Bsf     ADCON0, CHS3            ; Bit 3 was a 1
;
; Clear temporary storage for ADC result, prepare for average function
        Clrf    LSBYTE                  ; Clear temporary ADC LS Byte
        Clrf    MSBYTE                  ; Clear temporary ADC MS Byte
;
; Averaging function only if ADC_AVG selected in PROG_FLAG.  Match the number
; of values averaged with the number of shifts right to "divide"
        Movlw   0x10                    ; Number of values averaged = 16
        Movwf   Loop_Cnt
        Call    WAIT1MS                 ; Wait for ADC multiplexer to settle
Avg_Loop
        Bsf     ADCON0, GO              ;Start A/D conversion
ADC_Conv
        Btfsc   ADCON0, GO              ;Check for conversion complete on ADC_ch
        Goto    ADC_Conv
        Bsf     STATUS, RP0             ; Select Memory Bank 1
        Movf    ADRESL, w               ; Get ADC Result Least significant byte
        Bcf     STATUS, RP0             ; Select Memory Bank 0
        Addwf   LSBYTE, f               ; ADD and store result in LSBYTE
        Btfsc   STATUS, C               ; check if LS byte addition carry
        Incf    MSBYTE, f               ; Add one to MSB if carry
        Movf    ADRESH, w               ; Get ADC Result Most significant byte
        Addwf   MSBYTE, f               ; The 12-bit ADC result is right
                                        ;  justified, so upper nibble is zero
;
; Skip remaining routine if averaging is not selected.
        Btfss   PROG_FLAG, ADC_AVG      ; Test to see if averaging requested
        Goto    ADC_END                 ; Skip averaging
;
; Check if all samples acquired
        Decfsz  Loop_Cnt, f
        Goto    Avg_Loop                ; Continue average acquisition loop
;
; Need to shift result right 4 bits for 16 samples
        Bcf     STATUS, C               ; prepare for Rotate Right with carry
        Rrf     MSBYTE, f               ; Divide by 2
        Rrf     LSBYTE, f
        Bcf     STATUS, C
        Rrf     MSBYTE, f               ; Divide by 4
        Rrf     LSBYTE, f
        Bcf     STATUS, C
        Rrf     MSBYTE, f               ; Divide by 8
        Rrf     LSBYTE, f
        Bcf     STATUS, C
```

```
920         Rrf     MSBYTE, f               ; Divide by 16
921         Rrf     LSBYTE, f
922     ;
923     ADC_END
924         Movf    MSBYTE, w               ; Put MS Byte in Wreg
925         Return                          ; Return with ADC result MS Byte in Wreg
926     ;
927     ;*******************************************************************************
928     ;*******************************************************************************
929     ;       Acquire ADC data set Subroutine
930     ;               9/21/2001 ME Partridge
931     ; Function:
932     ;  Acquires one set of ADC data, and puts the values in the Data_Stack.  Channels
933     ;  zero through 9, skipping channel seven, are stored in LS Byte, MS Byte order.
934     ;
935     ; Arguments:
936     ;  Instance     Unique identifier in the calling subroutine
937     ;  AVG          Set to 0 for no averaging, non-zero produces averaged results.
938     ;
939     ; Calls:
940     ;  ADC_Read     Gets ADC value for channel specified
941     ;  WAIT1MS      Delays one millisecond
942     ;
943     ; Macros used:
944     ;  none.
945     ;
946     ; Registers:
947     ;  Wreg         Used in loop to select the ADC channel to convert
948     ;
949     ;  Interrupts:
950     ;       None used.
951     ;
952     ;  Return:
953     ;       Wreg            = On exit, zero.
954     ;
955     ;  Revisions:
956     ;       9/21/2001 First revision
957     ;*******************************************************************************
958     ;
959     ; Acquire one set of data.  If data are acquired while the serial lines are
960     ; active, the measurements seem to be affected.  So the acquire is kept
961     ; separate from the upload function.
962     Acquire
963         Movlw   Data_Start              ; top of memory stack
964         Movwf   FSR                     ;  . . . loaded into the stack pointer
965         Clrf    Channel_Cnt
966     ;
967     Acq_Loop
968         Movf    Channel_Cnt, w          ; analog channel number
969         Call    ADC_Read                ; ADC average in regs. LSBYTE and MSBYTE
970         Movf    LSBYTE, w               ; Store LSB first, unique to ST_HAND
971         Movwf   INDF                    ; Push onto stack
972         Incf    FSR, f                  ; increment stack pointer
973         Movf    MSBYTE, w
974         Movwf   INDF
975         Incf    FSR, f
976     Acq_Incr
977         Incf    Channel_Cnt, f
978     ;
979     ; Check if pointing to Channel 7 (which is not used)
980         Movlw   0x07                    ; skip channel 7, not used
981         Subwf   Channel_Cnt, w
982         Btfsc   STATUS, Z
983         Goto    Acq_Incr                ; is channel 7, just increment to next
984     ;
985     ; See if all the channels have been read
986         Movlw   0x0A                    ; Done if beyond channel 9
987         Subwf   Channel_Cnt, w
988         Btfss   STATUS, Z
989         Goto    Acq_Loop
990     ;
```

```
991     Acq_END
992             Retlw   0
993     ; .
994     ;*******************************************************************************
995     ;************************** END of FILE Ball.asm ******************************
996     ;*******************************************************************************
997             END
```

Code Listing – Cmd_Hand.asm

```
1   ; File name: "cmd_hand.asm"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Routines to interpret and execute commands sent to the P&G Sensor Ball
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       1999 - Adapted from MilliPen.asm for Sensor Ball, TA Rohwer
11  ;       November 2000 - Version 2, TA Rohwer
12  ;       17 Mar 2001 - Version 3 Changes requested by P&G, ME Partridge
13  ;       12 Aug 2001 - Version 4 Changes, ME Partridge
14  ;         Moved Baud change operation into Ser_Hand subroutine Baud_Set
15  ;         Used new string transmit function for "Who" identification command
16  ;         Added "Smart PCM Device" setup commands b,c,m,n,r,s,w,z
17  ;
18  ;*******************************************************************************
19  ;       Cmd_Hand(ler) Subroutine
20  ; Function:
21  ;   Works in conjunction with RC_ISR (Interrupt Service Routine).  The Sensor
22  ;   Ball must be in ATTENTION mode, set by pressing the ATTENTION button.
23  ;   CMD_HAND tests the CMD_CHAR variable for the known Sensor Ball commands and
24  ;   executes the appropriate procedure (see list in CMD_Jump below) or else
25  ;   error.
26  ;
27  ; Order of operations:
28  ;   1 PC sends string to package: 0, Cmd, <Parm1>, <Parm2>, <Parm . .> CR, LF
29  ;   2 Upon receipt of CR, LF, the ISR routine echoes the completed command phrase
30  ;   3 PC checks the echoed command, and sends the verify character if correct.
31  ;   4 Verify command received, echoes processed parameter values
32  ;   5 MAIN program sees COMMAND_FLG is set and calls CMD_HAND
33  ;   6 CMD_HAND processes the CMD_CHAR (which may set REINIT_FLG or others)
34  ;   7 MAIN program returns to loop checking for CMD_CHAR (unless REINIT_FLG)
35  ;
36  ; Subroutines in this file:
37        GLOBAL  CMD_HAND
38  ;
39  ;  Calls:
40        EXTERN  READ_HAND             ; Memory upload subroutine
41        EXTERN  STATUS_HAND           ; Sensor Ball status upload
42        EXTERN  SENDDATA              ; Sends standard completion message to PC
43        EXTERN  MEM_ERASE             ; Erases entire memory
44        EXTERN  MEM_TEST              ; Writes pattern supplied then erases
45        EXTERN  FILL_TEST             ; Fills Flash memory with known values
46        EXTERN  TX_WREG               ; Transmits on RS-232 serial interface
47        EXTERN  TX_String             ; Uses TX_WREG to send a string
48        EXTERN  Baud_Set              ; Sets Baud rate + configure serial port
49  ;
50  ;  Registers:
51        EXTERN  PROG_FLAG             ; bits COMMAND,REINIT,XON,XOFF,CMDERROR
52        EXTERN  TEMP                  ; Temporary for PROG_FLAG
53        EXTERN  SER_FLAG              ; Includes flag bit to acquire data
54        EXTERN  MEMORY_FLAG           ; Memory condition and test result flags, MEM_FULL
55        EXTERN  CMD_CHAR              ; Command character for CMD_HAND, in A - Z
56        EXTERN   CMD_PARAMS           ; Count of command parameters received
57  ;
58  ; Serial output starting location for strings in program memory
59        EXTERN  Look_Hi               ; Used to load PCLATH value
60        EXTERN  Look_Lo               ; Used to load PCL
61        EXTERN  List_Undef            ; Request undefined name
62        EXTERN  List_WhoID            ; Unit ID string
63        EXTERN  List_Commands         ; Valid command character string
64        EXTERN  List_Data00           ; First channel name
65        EXTERN  List_Slft00           ; First self-test bit name
66  ;
67  ;  Interrupts:
68  ;       Global, Peripheral, Serial Interrupts are enabled prior to call
69  ;       Serial Interrupts Disabled for routines responding with data to the PC
70  ;
71  ;  Return:
```

Code Listing – Cmd_Hand.asm

```
72  ;       WREG=0
73  ;
74  ;  include files:
75          #include "P16C774.INC"        ;Standard Header File for PIC16C773
76                                        ;Includes all Register Definitions,
77                                        ;RAM Definitions, & Configuration Bits
78          #include "ball_equ.inc"              ;EQU Declarations, equivalence
79  ;
80  ;*******************************************************************************
81  ;*******************************************************************************
82  ;
83  CMD_PROG        CODE            ;relocatable code in Program EPROM
84  ;
85  ; The command character was verified by SER_HAND to be in "A" . . "z"
86  ; Now, set Program Counter to select subroutine according to character
87  CMD_HAND
88          Clrf    PROG_FLAG               ; Clear Command Flag register
89          Movlw   high    CMD_Jump        ; Get upper Program Counter
90          Movwf   PCLATH                  ; set in upper 5 bits of PC
91          Movlw   "A"                     ; Character offset
92          Subwf   CMD_CHAR, w             ; Command Character minus Offset "A"
93          Addlw   CMD_Jump                ; Adds GOTO series starting address
94          Btfsc   STATUS, C               ; Check if address addition overflow
95          Incf    PCLATH, f               ; Yes, adjust program counter high byte
96          Movwf   PCL                     ; jump there
97  ;
98  CMD_Jump
99          Goto    CMD_A           ; acquire command, begin acquiring one record
100         Goto    CMD_B           ; Baud rate change, increase to 115.2K
101         Goto    CMD_Undef       ; command C
102         Goto    CMD_Undef       ; command D
103         Goto    CMD_E           ; E = erase memory command
104         Goto    CMD_F           ; command F
105         Goto    CMD_Undef       ; command G
106         Goto    CMD_Undef       ; command H
107         Goto    CMD_I           ; I = initialize command
108         Goto    CMD_Undef       ; command J
109         Goto    CMD_Undef       ; command K
110         Goto    CMD_Undef       ; command L
111         Goto    CMD_Undef       ; command M
112         Goto    CMD_Undef       ; command N
113         Goto    CMD_Undef       ; command O
114         Goto    CMD_I           ; P = initialize command
115         Goto    CMD_Undef       ; command Q
116         Goto    CMD_R           ; Read Records command
117         Goto    CMD_S           ; get unit status command
118         Goto    CMD_T           ; Test memory and erase
119         Goto    CMD_Undef       ; command U
120         Goto    CMD_Undef       ; command V - processed in SER_HAND, not here
121         Goto    CMD_Undef       ; command W
122         Goto    CMD_Undef       ; command X
123         Goto    CMD_Undef       ; command Y
124         Goto    CMD_Undef       ; command Z
125         Goto    CMD_Undef       ; command [ left-bracket open-square
126         Goto    CMD_Undef       ; command \ left-slash backslash bash
127         Goto    CMD_Undef       ; command ] right-bracket close-square
128         Goto    CMD_Undef       ; command ^ hat circumflex caret up-arrow
129         Goto    CMD_Undef       ; command _ UNT underscore underbar
130         Goto    CMD_Undef       ; command ` accent-grave backprime backquote
131         Goto    CMD_Undef       ; command a alpha
132         Goto    CMD_b           ; Send high Baud rate
133         Goto    CMD_c           ; Send recognized commands
134         Goto    CMD_Undef       ; command d delta
135         Goto    CMD_Undef       ; command e echo
136         Goto    CMD_Undef       ; command f foxtrot
137         Goto    CMD_Undef       ; command g golf
138         Goto    CMD_Undef       ; command h hotel
139         Goto    CMD_Undef       ; command i india
140         Goto    CMD_Undef       ; command j juliett
141         Goto    CMD_Undef       ; command k kilo
142         Goto    CMD_Undef       ; command l lima
```

```
143         Goto    CMD_m           ; Send channel numbers: analog, bilevel, self-test
144         Goto    CMD_n           ; Send name for specified channel
145         Goto    CMD_Undef       ; command o oscar
146         Goto    CMD_Undef       ; command p papa
147         Goto    CMD_Undef       ; command q quebec
148         Goto    CMD_r           ; Upload data starting at Page specified
149         Goto    CMD_s           ; Status in "Smart PCM Device" format
150         Goto    CMD_Undef       ; command t tango
151         Goto    CMD_Undef       ; command u uniform
152         Goto    CMD_Undef       ; command v victor
153         Goto    CMD_w           ; Who are you? Identify this unit
154         Goto    CMD_Undef       ; command x x-ray
155         Goto    CMD_Undef       ; command y yankee
156         Goto    CMD_z           ; Reply total memory and transfer block size
157 ;
158 ;*******************************************************************************
159 ;
160 ; Command error, command character invalid (not in A - z) or command undefined
161 CMD_Undef
162         Bsf     PROG_FLAG, CMDERROR_FLG       ;Set Command Error Flag
163         Goto    CMD_END                 ; Exit routine
164 ;*******************************************************************************
165 ;
166 ; Set ARM to create one Record of 256 data frames, each frame one 512-byte page.
167 ; Recording can be interrupted at the end of each 512-byte page by pressing the
168 ; ATTENTION button (may need to hold ATTENTION for 10 sec).
169 ;
170 ; Received command: "0", "A", CR, LF
171 CMD_A
172         Bsf     PROG_FLAG, ACQUIRE_FLG ; Flag - main routine acquire a Record
173         Goto    CMD_END
174 ;
175 ;*******************************************************************************
176 ;
177 ; Command B -- Toggle baud rate between 19.2k (default) and 115.2k Baud.  Also
178 ; returns to 19.2k anytime "ATTENTION" is pushed or by the "Reset" command.
179 ;
180 ; Received command: "0", "B", CR, LF
181 CMD_B
182 ;
183 ; Need to wait for transmit buffer to empty before doing the Baud change
184 ; SPBRG and TXSTA Registers in Bank 1 Memory
185         Bsf     STATUS, RP0             ; Select Bank 1 memory
186         Btfss   TXSTA, TRMT             ; Set to 1 when TSR empty
187         Goto    CMD_B                   ; Loop and wait
188 ;
189         Movf    SPBRG, w                ; Recover value
190         Sublw   0x01                    ; Value for 115.2k Baud
191         Call    Baud_Set                ; Wreg zero if now 115.2k Baud
192 ;                                         Wreg=0 sets 19.2k Baud
193 ;                                         Wreg<>0 sets 115.2k Baud
194 ;                                         Bank 0 memory selected at end of Baud_Set
195         Goto    CMD_END
196 ;
197 ;*******************************************************************************
198 ;
199 ; Command E -- Erase all memory.  Also marks bad blocks if found.
200 ; Received command: "0", "E", CR, LF
201 CMD_E
202         Call    MEM_ERASE               ;Call Flash Memory Erase routine
203         Goto    CMD_END
204 ;
205 ;*******************************************************************************
206 ;
207 ; Command F -- Fill memory with known values starting at the current memory
208 ; location for the number of Records optionally specified.  If no parameters are
209 ; provided, one Record is filled.
210 ; Received command: "0", "F", <MSB # Records>, <LSB # Records>, CR, LF
211 CMD_F
212         Call    FILL_TEST               ;Call Fill test in FLASH file
213         Goto    CMD_END
```

Code Listing – Cmd_Hand.asm

```
214  ;*******************************************************************************
215  ;
216  ; Command I -- Initialize.  Exits command mode (entered by pushing ATTENTION button)
217  ; Received command: "0", "I", CR, LF
218  CMD_I
219          Bsf     PROG_FLAG, REINIT_FLG  ;Set Re-initialize Flag
220          Goto    CMD_END
221  ;*******************************************************************************
222  ;
223  ; Command P -- Set Re-initialize flag
224  ; Received command: "0", "P", CR, LF
225  CMD_P
226          Bsf     PROG_FLAG, REINIT_FLG  ;Set Re-initialize Flag
227          Goto    CMD_END
228  ;*******************************************************************************
229  ;
230  ; Command R -- Uploads records starting at block (Frame or Page) optionally
231  ; specified.  If no parameters are provided begins at memory start.  Waits for
232  ; Ack or Nak character after each block (Frame or Page in this code).  If Nak,
233  ; the block is retransmitted.
234  ;
235  ; Received command: "0", "R", <MSB Page #>, <LSB Page #>, CR, LF
236  CMD_R
237          Bcf     PROG_FLAG, SIMPLE_FLG  ; Ensure SIMPLE_FLG clear
238          Call    READ_HAND              ; Call Read Handler
239          Goto    CMD_END
240  ;*******************************************************************************
241  ;
242  ; Command S -- Return status of unit
243  ; Received command: "0", "S", CR, LF
244  CMD_S
245          Bcf     PROG_FLAG, SIMPLE_FLG  ; Ensure SIMPLE_FLG clear
246          Call    STATUS_HAND            ; Call Status Handler
247          Goto    CMD_END
248  ;*******************************************************************************
249  ;
250  ; Command T -- Test Flash memory, erasing after each block & marking bad blocks
251  ; Erases entire memory, then writes pattern and erases on a per-block basis
252  ; Received command: "0", "T", CR, LF
253  CMD_T
254          Clrf    MEMORY_FLAG            ;Clear Test Result Flags
255          Call    MEM_ERASE
256  ;
257  ; Writes pattern then erases each block.  Attempts to mark block if bad.
258          Movlw   0xAA                   ; Pattern expected in Wreg
259          Call    MEM_TEST               ; Test Flash with 0xAA pattern
260  ;
261  ; Next pattern
262          Movlw   0x55                   ; pattern expected in Wreg
263          Call    MEM_TEST               ; Test Flash with 0x55 pattern
264  ;
265          Goto    CMD_END
266  ;
267  ;*******************************************************************************
268  ;
269  ; Command b -- Send high Baud rate, 115.2k Baud.  (Low is standardized as the
270  ; same for all systems, 19.2k Baud.)  Sends rate as ASCII string.
271  ; Received command: "0", "b", CR, LF
272  CMD_b
273          Movlw   "1"
274          Call    TX_WREG
275          Movlw   "1"
276          Call    TX_WREG
277          Movlw   "5"
278          Call    TX_WREG
279          Movlw   "2"
280          Call    TX_WREG
281          Movlw   "0"
282          Call    TX_WREG
283          Movlw   "0"
284          Call    TX_WREG
```

```
285 ;
286         Goto    CMD_CRLF
287 ;
288 ;*******************************************************************************
289 ;
290 ; Command c -- Send recognized commands.  Sends back a null terminated string
291 ; with each recognized command character (excluding setup commands).
292 ; Received command: "0", "c", CR, LF
293 CMD_c
294         Movlw   High    List_Commands  ; Load starting location of string
295         Movwf   Look_Hi
296         Movlw   Low     List_Commands
297         Movwf   Look_Lo
298         Goto    CMD_String
299 ;
300 ;
301 ;*******************************************************************************
302 ;
303 ; Command m -- Send number of each channel type: analog/digital, bilevel,
304 ; self-test bits using four bytes, with the last unused.
305 ; Received command: "0", "m", CR, LF
306 CMD_m
307         Movlw   0x00                    ; Byte 2 = bilevels
308         Call    TX_WREG
309         Movlw   0x0D                    ; Byte 3 = analog/digital, 13 = 0x0D
310         Call    TX_WREG
311         Movlw   0x00                    ; Byte 0 = unused
312         Call    TX_WREG
313         Movlw   0x08                    ; Byte 1 = self-test bits
314         Call    TX_WREG
315 ;
316         Goto    CMD_CRLF
317 ;
318 ;
319 ;*******************************************************************************
320 ;
321 ; Command n -- Send channel name specified in parameters as follows:
322 ;   Byte 1 = analog/digital channel name
323 ;   Byte 2 = bilevel channel name (not used in Sensor Ball)
324 ;   Byte 3 = self-test bit name
325 ;   Byte 4 = unused.
326 ; Received command: "0", "n", <byte 1>, <byte 2>, <byte 3>, <byte 4>, CR, LF
327 CMD_n
328 ;
329 ; Adjust Command Parameter stack pointer to last parameter passed
330         Movlw   CMD_Param_Start                 ; Added to count of params CMD_PARAMS
331         Movwf   FSR                     ; Put into pointer
332         Movf    INDF, w                 ; Get number of parameters
333         Btfsc   STATUS, Z               ; Test if no parameters passed
334         Goto    CMD_END                 ; No parameters, no response.
335 ;
336 ; The channel and self-test bit name strings are all 16 bits long, padded with
337 ; nulls.  So the selected name can be located by adding the appropriate multiple
338 ; of 16 as an offset to the starting name string.  Only one of the Bytes will
339 ; be populated, and since no channel or self-test bit number is greater than
340 ; 16, a simplified address adjustment can be made.
341 CMD_n_B1
342         Incf    FSR, f                  ; Adjust stack pointer
343         Movf    INDF, w                 ; Extract Parameter Byte 1
344         Btfsc   STATUS, Z               ; Will be zero if not selected
345         Goto    CMD_n_B2
346 ;
347 ; Select string for transmit
348         Movlw   High    List_Data00     ; Load location of first channel string
349         Movwf   Look_Hi
350         Movlw   Low     List_Data00     ; First channel name
351         Movwf   Look_Lo                 ; Store low address
352         Movlw   0x0E                    ; check if beyond 13 (0 to 12)
353         Goto    CMD_n_CHK_Msg
354 ;
355 ; Second Byte, bilevel channel name (not used in Sensor Ball)
```

```
356  CMD_n_B2
357          Incf    FSR, f                  ; Adjust stack pointer
358          Movf    INDF, w                 ; Extract Parameter Byte 2
359          Btfsc   STATUS, Z               ; Will be zero if not selected
360          Goto    CMD_n_B3
361  ;
362  ; No bilevels defined, so send message
363          Goto    CMD_Undef_Msg
364  ;
365  ; Third Byte, Self-Test bits
366  CMD_n_B3
367          Incf    FSR, f                  ; Adjust stack pointer
368          Movf    INDF, w                 ; Extract Parameter Byte 3
369          Btfsc   STATUS, Z               ; Will be zero if not selected
370          Goto    CMD_END                 ; All paramters were zero, no response
371  ;
372  ; Select string for transmit
373          Movlw   High    List_Slft00     ; Load first Self-test bit name location
374          Movwf   Look_Hi
375          Movlw   Low     List_Slft00     ; Low address of name
376          Movwf   Look_Lo                 ; Store low address
377          Movlw   0x09                    ; check if beyond 8 (0 to 7)
378  ;
379  ; Maximum name value was loaded in Wreg above.  Now test to see if the name
380  ; exists, and adjust name pointer if it does.
381  CMD_n_CHK_Msg
382          Subwf   INDF, w                 ; Subwf subtracts name number
383          Btfsc   STATUS, C               ; carry set means name number <= defined names
384          Goto    CMD_Undef_Msg           ; above maximum name number, error
385          Decf    INDF, f                 ; Adjust to use as pointer
386          Swapf   INDF, w                 ; Effectively, multiply by 16
387          Addwf   Look_Lo, f              ; Adjust address to point to proper string
388          Btfsc   STATUS, C               ; Check if overflow on address adjust
389          Incf    Look_Hi, f              ;  . . . was overflow, adjust
390          Goto    CMD_String
391  ;
392  CMD_Undef_Msg
393          Movlw   High    List_Undef      ; Load location of message string
394          Movwf   Look_Hi
395          Movlw   Low     List_Undef
396          Movwf   Look_Lo                 ; Store low address
397          Goto    CMD_String
398  ;
399  ;*******************************************************************************
400  ;
401  ; Command r -- Uploads records starting at block (Frame or Page) optionally
402  ; specified.  If no parameters are provided begins at memory start.  No checks
403  ; are made for valid Sync in memory, so bad and blank Blocks are uploaded.  The
404  ; checksum is calculated and transmitted, and Ack and Nak control characters are
405  ; waited for.  X-on and X-off are also still active.
406  ; Received command: "0", "r", <MSB Page #>, <LSB Page #>, CR, LF
407  CMD_r
408          Bsf     PROG_FLAG, SIMPLE_FLG ; Set for simple upload
409          Call    READ_HAND             ; Call Read Handler
410          Goto    CMD_END
411  ;
412  ;*******************************************************************************
413  ;
414  ; Command s -- Return status of unit in "Smart PCM Device" format.  Data are
415  ; returned in the order contained in command "n":
416  ;   all analog/digital channels
417  ;   bilevel channels (not used in Sensor Ball so none sent)
418  ;   all self-test bit data, sent as a 16-bit word.
419  ; Received command: "0", "s", CR, LF
420  CMD_s
421          Bsf     PROG_FLAG, SIMPLE_FLG ; Set for "Smart PCM Device" status
422          Call    STATUS_HAND           ;Call Status Handler
423          Goto    CMD_END
424  ;
425  ;*******************************************************************************
426  ;
```

Code Listing – Cmd_Hand.asm

```
427  ; Command w -- Who are you.  Sends back a phrase identifying this as a
428  ; Procter & Gamble Sensor Ball and includes the Version number.
429  ; Received command: "0", "w", CR, LF
430  CMD_w
431          Movlw   High    List_WhoID      ; Load starting location of string
432          Movwf   Look_Hi
433          Movlw   Low     List_WhoID
434          Movwf   Look_Lo
435  ;
436          Goto    CMD_String
437  ;
438  ;*******************************************************************************
439  ;
440  ; Command z -- Send memory size.  Sends six bytes, with the first four specifying
441  ; the device's memory size, and the last two bytes specifying data block size.
442  ; Received command: "0", "z", CR, LF
443  CMD_z
444  ;
445  ; Size is 32MB, with a block size of 560 + 2 Checksum Bytes (2 Sync and 2 Bytes
446  ; with block number not counted).  Use 32MB memory = 0x020000, and 562 byte
447  ; block = 0x0232
448          Movlw   0x00                    ; Byte 4, Intel format: LSByte, MSByte
449          Call    TX_WREG
450          Movlw   0x02                    ; Byte 5
451          Call    TX_WREG
452          Movlw   0x00                    ; Byte 2
453          Call    TX_WREG
454          Movlw   0x00                    ; Byte 3
455          Call    TX_WREG
456          Movlw   0x00                    ; Byte 0
457          Call    TX_WREG
458          Movlw   0x00                    ; Byte 1
459          Call    TX_WREG
460  ;
461  ; Block Size
462          Movlw   0x32                    ; Byte 0, Intel format: LSByte, MSByte
463          Call    TX_WREG
464          Movlw   0x02                    ; Byte 1
465          Call    TX_WREG
466  ;
467          Goto    CMD_CRLF
468  ;
469  ;*******************************************************************************
470  ;
471  ; End of Command handler routines, clean everything up
472  CMD_String
473          Call    TX_String               ; Call string transmit routine
474  CMD_CRLF
475          Movlw   "\r"                    ; Carriage Return
476          Call    TX_WREG                 ; Transmit
477          Movlw   "\n"                    ; Line Feed
478          Call    TX_WREG                 ; Transmit
479  ;
480  CMD_END
481          Clrf    CMD_CHAR                ;Clear CMD_CHAR
482  ;
483  ; Reset Command Parameter stack in case partial command received
484          Movlw   CMD_Param_Start                 ; Reset command parameter stack
485          Movwf   FSR                     ;   by setting pointer to top.
486          Clrf    CMD_PARAMS              ; Clear count of parameters received
487          Retlw   0                       ; Return to MAIN, Wreg=0
488  ;
489  ;*******************************************************************************
490  ;********************** End of FILE Cmd_Hand.asm ********************************
491  ;*******************************************************************************
492          END
```

Code Listing – Flash.asm

```
1   ; File name: "flash.asm"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Flash Memory auxiliary routines
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       1999 - Adapted from MilliPen.asm for Sensor Ball, TA Rohwer
11  ;       Nov 2000 - Version 2, TA Rohwer
12  ;       17 Mar 2001 - Version 3, Revised extensively by ME Partridge
13  ;       28 Jun 2001 - Minor revisions, ME Partridge
14  ;       12 Aug 2001 - Version 4, Check for sequential blank Blocks in MEM_FIND
15  ;
16  ;*******************************************************************************
17  ;       Flash Memory Subroutines
18  ; Function:
19  ;  Contains all functions needed to read, write, and erase the Toshiba TC58256FT
20  ;  Flash memory.  The device is a 3.3V, 256Mbit (32MB) NAND Electrically
21  ;  Erasable and Programmable Read-Only Memory (EEPROM) organized as 528 bytes x
22  ;  32 pages x 2048 blocks.
23  ;
24  ;       Page size: 528 Bytes (512 Bytes plus extra 16 Bytes extended area)
25  ;       Block size: 32 Pages, 16k-Bytes + 512 Bytes
26  ;       Total memory: 2048 Blocks, 32MB
27  ;
28  ;  The device has a 528-byte static register which allows program and read data
29  ;  to be transferred between the register and the memory cell array in 528-byte
30  ;  increments.  The Erase operation is implemented in single block units (16k
31  ;  Bytes + 512 bytes).  The TC58256FT is a serial-type memory that uses the I/O
32  ;  pins for command and address input as well as data input / output.
33  ;
34  ;       The following commands are supported by the device:
35  ; 0x00  Read Mode 1, Address bit A8=0, can read sequentially thru entire device
36  ; 0x80  Write with Address bit A8=0, writes to the 528-Byte static register.
37  ; 0x10  Program, transfers the 528-Byte static register to the memory cell array
38  ; 0x60  Erase, erases a 16k-Byte memory block.  Must be followed by Erase Confirm
39  ; 0xD0  Erase confirm command
40  ; 0x70  Status, provides Ready / Busy~, and write fail.
41  ;
42  ; Special Note -- Timing issues
43  ;  The TC58256FT memory has a few timing parameters that are significant with
44  ;  respect to instruction step time.  Short delays are accommodated by inserting
45  ;  NOPs into the code.  Longer delays are handled by looping on the Status
46  ;  command to monitor Ready / Busy~.  The R/B~ line on the part is not used.
47  ;       Read address to data available, 10 microseconds maximum, NOP padded
48  ;       Program - 1000 microseconds maximum.  Use Status command to check
49  ;       Erase - May take up to 20 milliseconds.  Use Status command to check
50  ;
51  ; Special Note -- Number of valid Blocks
52  ;  The TC58256FT memory is not guaranteed to have the entire 2048 Blocks
53  ;  available to write.  For this reason, checks are made for bad blocks, which
54  ;  are skipped if found.
55  ;
56  ; Subroutines in this file:
57          GLOBAL  MEM_READ                ; Return value from Flash memory
58          GLOBAL  MEM_WRITE               ; Writes Wreg value to Flash memory
59          GLOBAL  MEM_FIND                ; Finds the first blank Block
60          GLOBAL  MEM_RD_SET              ; Sets Flash address for reading
61          GLOBAL  MEM_WR_SET              ; Sets Flash address for writing
62          GLOBAL  MEM_Header              ; Fills out the once-per-Page values
63          GLOBAL  MEM_PROGRAM             ; Transfers internal buffer to Flash
64          GLOBAL  MEM_ERASE               ; Erases entire memory
65          GLOBAL  MEM_REC_FIND            ; Finds a particular Record number
66          GLOBAL  MEM_TEST                ; Writes pattern supplied then erases
67          GLOBAL  FILL_TEST               ; Writes known values to # recs specified
68  ;
69  ; Calls:
70          EXTERN  SENDDATA
71          EXTERN  Fail_Msg                ; Sends failure text message to PC
```

```
72          EXTERN  Status_Msg              ; Sends PC update message during memory test
73          EXTERN  Erase_Msg               ; Sends PC update message during memory erase
74          EXTERN  WAIT1MS
75    ;
76    ; Registers:
77          EXTERN  MEMORY_FLAG             ; Memory Status and test flags
78          EXTERN  TEMP                    ; Temporary MEMORY_FLAG and value read
79          EXTERN  PATTERN                 ; Test pattern passed in Wreg
80          EXTERN  R_MEM_REC_NUM           ; Temporary value to find Record
81          EXTERN  MEM_REC_NUM             ; Current Flash memory Record, 1 - 64
82          EXTERN  MEM_FRM_NUM             ; Current Record Frame (Page), 0 - 255
83          EXTERN  MEM_BAD_NUM             ; Bad blocks in Flash memory
84          EXTERN  ADD0                    ; Flash Memory address A7 .. A0
85          EXTERN  ADD1                    ; Flash memory address A16 .. A8
86          EXTERN  ADD2                    ; Flash memory address A24 .. A17
87          EXTERN  R_ADD1                  ; Temporary for ADD1
88          EXTERN  R_ADD2                  ; Temporary for ADD2
89          EXTERN  A8                      ; Half of memory page, 1st (FE) or 2nd (FF)
90    ;
91    ; Registers for test routine
92          EXTERN  CMD_PARAMS
93          EXTERN  R_MEM_REC_NUM
94          EXTERN  TIME3
95          EXTERN  TIME2
96          EXTERN  TIME1
97          EXTERN  TIME0
98          EXTERN  R_TIME3
99          EXTERN  R_TIME2
100         EXTERN  R_TIME1
101         EXTERN  R_TIME0
102         EXTERN  MeasSet_Cnt             ; Loop counter, 28 sets of 9 meas. / page
103         EXTERN  Channel_Cnt             ; Loop counter, analog channels 0 to 9 skip 7
104         EXTERN  TEMP                    ; Scratch register for blank check
105   ;
106   ; Interrupts:
107   ;       Global, Peripheral, Serial Interrupts are enabled prior to call
108   ;
109   ; Return:
110   ;       WREG=0
111   ;
112   ;  include files:
113         #include "P16C774.INC"          ;Standard Header File for PIC16C773
114                                         ;Includes all Register Definitions,
115                                         ;RAM Definitions, & Configuration Bits
116         #include "ball_equ.inc"                 ;EQU Declarations, equivalence
117   ;
118   MEM_PROG        CODE                  ; Relocatable Code
119   ;
120   ;*******************************************************************************
121   ;*******************************************************************************
122   ; Memory Read subroutine.  Returns with Wreg containg the data that was read.
123   ; Moves data from Port D (the Flash I/O lines) to Wreg, and controls the
124   ; active-low Read Enable signal.  On exit, Wreg has the data read.
125   MEM_READ
126         Bcf     PORTC, RE               ; RE~, Read active low
127         Movf    PORTD, w                ; Recover Flash data into Wreg
128         Bsf     PORTC, RE               ; RE~, Read active low
129         Return                          ; Return with value in Wreg
130   ;
131   ;*******************************************************************************
132   ;*******************************************************************************
133   ; Memory Write subroutine.  Moves data from Wreg to Port D (the Flash I/O
134   ; lines), and controls the active-low Write Enable signal.  On exit, Wreg
135   ; retains the value written.
136   MEM_WRITE
137         Movwf   PORTD                   ; Write one byte
138         Bcf     PORTC, WE               ; WE~, Active low write enable data
139         Bsf     PORTC, WE
140         Return                          ; Return with written value in Wreg
141   ;
142   ;*******************************************************************************
```

```
143  ;*******************************************************************************
144  ;
145  ;*******************************************************************************
146  ;*******************************************************************************
147  ;       Memory Find Subroutine                                          MEP
148  ;
149  ; Function:
150  ;   Starting from the last used address ADD2 & ADD1, checks the start of this
151  ;   location and subsequent 16kB Blocks find the next blank location to write
152  ;   data.
153  ; Calls:
154  ;   None (all calls internal to this file)
155  ; Registers:
156  ;   MEMORY_FLAG(MEM_FULL) - indicates that all 32MB memory is filled
157  ; Interrupts:
158  ;   None changed
159  ; Return:
160  ;   NONE.
161  ;
162  ;*******************************************************************************
163  ;
164  ; Search Flash memory to set address ADD2, ADD1 to point to next blank Block
165  MEM_FIND
166         Clrf    MEMORY_FLAG             ; Reset flags, will set accordingly
167         Clrf    A8                      ; Half-page flag, used in ext. blank ck.
168         Movlw   0xE0                    ; Mask even Block boundary, 16kB=0x20
169         Andwf   ADD1, f                 ; Strip off non-integral Block address
170  ;
171  MEM_LOOP
172         Call    MEM_RD_SET              ; Set up Flash to read Block pointed to
173  ;
174  ; Read first two Bytes.  Will equal 0xFFFF if blank, 0xEB90 if used, and 0xA5A5
175  ; or something else if the Block is bad.
176  MEM_USED_CK
177         Call    MEM_READ
178         Movwf   TEMP                    ; Hold Value
179         Incf    TEMP, w                 ; Blank is 0xFF, incf -> 0x00
180         Btfss   STATUS, Z               ; Will have become zero if was blank
181         Goto    MEM_SYNC_CK             ; Not blank, check if sync
182  ;
183         Call    MEM_READ                ; Read second byte in Block
184         Sublw   0xFF                    ; Check if blank = 0xFF
185         Btfss   STATUS, Z               ; If result zero, was blank
186         Goto    MEM_BAD_BLK             ; Was not blank, so memory bad
187  ;
188  ; Read two first Bytes in Block and both blank.  Repeat test on next Block
189  ; to ensure that it wasn't just an isolated Block or was bad.
190         Btfsc   A8, 0                   ; Bit zero marks previous blank success
191         Goto    MEM_DONE                ; Second Block blank, process done
192         Bsf     A8, 0                   ; Set flag for first Block blank
193         Movf    ADD1, w                 ; Recover memory address
194         Movwf   R_ADD1                  ; Hold in temporary
195         Movf    ADD2, w                 ; Next significant memory address
196         Movwf   R_ADD2                  ; Hold it also in temporary
197         Goto    MEM_NXT_BLK             ; Increment address, loop on next Block
198  ;
199  ; The first Byte was not blank, see if marked with Sync indicating valid data
200  MEM_SYNC_CK
201         Clrf    A8                      ; Cancel sequential blank Block flag
202         Movf    TEMP, w                 ; Recover value read
203         Sublw   Sync_Byte0              ; First Sync Byte if Block used
204         Btfss   STATUS, Z               ; Result zero if equals Sync0
205         Goto    MEM_BAD_BLK             ; not sync, so memory bad
206  ;
207  ; First Byte in Block is Sync0.  Now check if second Byte is Sync1.
208         Call    MEM_READ                ; Now look for second Sync
209         Sublw   Sync_Byte1
210         Btfss   STATUS, Z               ; if is second sync, zero
211         Goto    MEM_BAD_BLK             ; not sync, so memory bad
212  ;
213  ; Memory used, read Record number
```

```
214         Call    MEM_READ                ; Frame number
215         Call    MEM_READ                ; Record number
216         Movwf   MEM_REC_NUM             ; Hold last Record number used
217         Goto    MEM_NXT_BLK
218 ;
219 ; Bad 16kB block, so Skip block but don't increment block count
220 MEM_BAD_BLK
221         Incf    MEM_BAD_NUM, f          ; count number of bad blocks
222 ;
223 ; Assume entire 16kB Block is not blank or is bad - move to next 16k-Byte Block
224 ; and check if advanced beyond 32MB boundary
225 MEM_NXT_BLK
226         Movlw   0x20                            ; ADD1 increment for 16kB Block
227         Addwf   ADD1, f                         ; Store result in ADD1
228         Btfss   STATUS, C                       ; Carry set if rolled over
229         Goto    MEM_LOOP                        ; No need to check ADD2 yet
230 ;
231 ; ADD1 rolled over, so adjust ADD2.  If ADD2 rolls over, have incremented past
232 ; end of 32MB memory so set flag to indicate that no more records can be stored.
233         Incf    ADD2, f                         ; Increment ADD2 if ADD1 carry
234         Btfss   STATUS, Z                       ; Check if ADD2 rolled to zero
235         Goto    MEM_LOOP                        ; Not at end, continue search
236         Bsf     MEMORY_FLAG, MEM_FULL           ; Indicates memory filled
237         Retlw   0                               ; Leave ADD1 & ADD2 at zero
238 ;
239 ; Found sequential blank Blocks.  Assume OK, restore previous address
240 ; Calling routine must set up memory location found for read or write
241 MEM_DONE
242         Movf    R_ADD1, w               ; Recover temporary Address
243         Movwf   ADD1                    ; Restore value
244         Movf    R_ADD2, w               ; Recover temporary Address
245         Movwf   ADD2                    ; Restore value
246 ;
247         Retlw   0
248 ;
249 ;*******************************************************************************
250 ;*******************************************************************************
251 ; Find a particular record number by first calculating the minimum address
252 ; location for the record, then searching from that point to find the start
253 ; of the Record.  ADD1 counts the number of 512-Byte Pages, ADD2 the number of
254 ; 128kB Records.  A Block-by-Block search is conducted from that point forward
255 ; because a bad Block may have been encountered and skipped, throwing off the
256 ; Blocks / Record.  Flash address registers ADD2, ADD1, and ADD0 are modified.
257 ;
258 MEM_REC_FIND
259         Btfsc   STATUS, Z               ; Wreg has Record number, Verify not Zero
260         Movlw   0x01                    ; Wreg=0, default to Record=1
261         Movwf   TEMP                    ; Hold Record sought for compare
262         Movwf   ADD2                    ; ADD2 as minimum is Record number. . .
263         Decf    ADD2, f                 ;  . . . minus one.
264         Clrf    ADD1                    ; Start of Record
265         Bcf     MEMORY_FLAG, MEM_FIND_FLG       ; Record search fail flag
266 ;
267 ; Begin search with minimum possible address, 128kB per Record.
268 MEM_FIND_LOOP
269         Call    MEM_RD_SET              ; Set up Flash to Record start
270 ;
271 ; Read first two Bytes.  Will equal 0xEB90 if used, 0xFFFF if blank, and 0xA5A5
272 ; or something else if the block is bad.
273         Call    MEM_READ
274         Sublw   Sync_Byte0              ; First Sync Byte if Block used
275         Btfss   STATUS, Z               ; Result zero if equals Sync0
276         Goto    MEM_FIND_NXT            ; Not sync, assume bad block
277         Call    MEM_READ                ; Now look for second Sync
278         Sublw   Sync_Byte1
279         Btfss   STATUS, Z               ; if is second sync, zero
280         Goto    MEM_FIND_NXT            ; Not sync, assume bad block
281 ;
282 ; Good block, check if correct Record, which is fourth byte on Page
283 ; This seems to have problems.  Found good Sync, so just exit.
284         Goto    MEM_FIND_DONE           ; Found it, Return
```

```
285   ;
286   ; Get next block for search
287   MEM_FIND_NXT
288           Movlw   0x20                    ; ADD1 increment for 16kB Block
289           Addwf   ADD1, f                 ; Store result in ADD1
290           Btfss   STATUS, C               ; Carry set if rolled over
291           Goto    MEM_FIND_LOOP           ; No need to check ADD2 yet
292   ;
293   ; ADD1 rolled over, so adjust ADD2.  If ADD2 rolls over, have incremented past
294   ; end of 32MB memory so set flag to indicate that no more records can be stored.
295           Incf    ADD2, f                 ; Increment ADD2 if ADD1 carry
296           Btfss   STATUS, Z               ; Check if ADD2 rolled to zero
297           Goto    MEM_FIND_LOOP           ; Not at end, continue search
298   ;
299   ; At memory boundary, set flag to indicate that no more records can be stored.
300   MEM_FIND_FAIL
301           Bsf     MEMORY_FLAG, MEM_FIND_FLG     ; Sets Record search fail flag
302   ;
303   ; Return to calling point, That routine needs to set up memory for read or write
304   MEM_FIND_DONE
305           Retlw   0
306   ;
307   ;*******************************************************************************
308   ;*******************************************************************************
309   ; On entry, ADD1 & 2 are set up to point to the 512B memory Page to read.
310   ; Because the Toshiba TC58256FT Flash memory can take up to 10us between
311   ; sending the address and having memory ready, a 10 NOP delay is included.
312   ;
313   MEM_RD_SET
314           Bcf     STATUS, RP0             ; Select Bank 0 memory
315           Clrf    PORTD                   ; Port D is Address / Data bus to Flash
316           Bsf     STATUS, RP0
317           Clrf    TRISD                   ; Set PORTD=Outputs (memory bank 1)
318           Bcf     STATUS, RP0             ; Select Bank 0 memory
319           Bcf     PORTC, CE               ; Chip enable CE~, Active low
320           Bsf     PORTB, CLE              ; Command Latch Enable
321           Movlw   0x00                    ; Command 0x00 is Read Mode 1, Address A8=0
322           Call    MEM_WRITE
323           Bcf     PORTB, CLE
324           Bsf     PORTB, ALE              ; Address Latch Enable
325           Movlw   0x00                    ; Substitute for ADD0, A7 to A0, always zero
326           Call    MEM_WRITE
327           Movf    ADD1, w                 ; Wreg with A16 to A9
328           Call    MEM_WRITE
329           Movf    ADD2, w                 ; Wreg with A24 to A17
330           Call    MEM_WRITE
331           Bcf     PORTB,ALE
332   ;
333   ; Get Port D ready for read function.  Flash chip is left enabled.
334           Clrf    PORTD
335           Bsf     STATUS, RP0
336           Movlw   0xFF
337           Movwf   TRISD                   ;Set PORTD=Inputs
338           Bcf     STATUS, RP0
339   ;
340   ; Delay to allow Flash memory to access data requested, 10us maximum
341           Nop
342           Nop
343           Nop
344           Nop
345           Nop
346   ;
347           Retlw   0
348   ;
349   ;*******************************************************************************
350   ;*******************************************************************************
351   ; On entry, ADD1 & 2 are set up to point to the memory Page to write.  Contrary
352   ; to Toshiba documentation, write enable is necessary for both filling the
353   ; buffer and the Program function.
354   ;
355   MEM_WR_SET
```

```
356           Bsf     STATUS, RP0             ; Select Bank 1 memory
357           Clrf    TRISD                   ; Set PORTD=Outputs
358           Bcf     STATUS, RP0             ; Select Bank 0 memory
359           Bsf     PORTB, WP               ; WP~ high, writes enabled
360           Bcf     PORTC, CE               ; Chip enable CE~, Active low
361           Bsf     PORTB, CLE              ; Command Latch Enable
362           Movlw   0x80                    ; Command 0x80 Data Input first half-Page
363           Call    MEM_WRITE
364           Bcf     PORTB, CLE              ; End Command write
365           Bsf     PORTB, ALE              ; Address Latch Enable, address write
366           Movlw   0x00                    ; Substitute for ADD0, A7 to A0, always zero
367           Call    MEM_WRITE
368           Movf    ADD1, w                 ; Wreg with A16 to A9
369           Call    MEM_WRITE
370           Movf    ADD2, w                 ; Wreg with A24 to A17
371           Call    MEM_WRITE
372           Bcf     PORTB,ALE               ; End Address write
373     ;
374     ; Port D is left ready for write function, and Flash chip is enabled.
375     ;
376           Retlw   0
377     ;
378     ;*****************************************************************************
379     ;*****************************************************************************
380     ; On entry, ADD1 & 2 are set up to the start of a blank memory Page.  This
381     ; routine fills in the values that are written only once every Page, like TIME,
382     ; Record number, and so on.
383     ;
384     MEM_Header
385     ;
386     ; Write sync pattern into the first two bytes in each frame (512-byte Page).
387     ; This provides a known value to detect used, bad, or blank Flash Memory Pages.
388           Movlw   Sync_Byte0              ; Sync Bytes.
389           Call    MEM_WRITE
390           Movlw   Sync_Byte1
391           Call    MEM_WRITE
392     ;
393     ; Write Frame and Record number.
394           Movf    MEM_FRM_NUM, w          ; Get current frame count, 0 to 255
395           Call    MEM_WRITE               ; Store in Flash memory
396           Movf    MEM_REC_NUM, w          ; Record number, 1 to 255
397           Call    MEM_WRITE               ; Store in Flash memory
398     ;
399     ;   Write the TIMEn registers, MS Word, LS Word, with LS Byte first (Intel fmt)
400           Movf    TIME2, w                ; Write 4 Bytes of time information
401           Call    MEM_WRITE               ; . . .  - placeholder for RTC
402           Movf    TIME3, w                ; Bytes currently incremented w/o RTC
403           Call    MEM_WRITE               ; Store in Flash memory
404           Movf    TIME0, w
405           Call    MEM_WRITE               ; Store in Flash memory
406           Movf    TIME1, w
407           Call    MEM_WRITE               ; Store in Flash memory
408           Retlw   0
409     ;
410     ;*****************************************************************************
411     ;*****************************************************************************
412     ; Transfer internal Flash write buffer to Flash memory Page.  Deselects chip
413     ; and set Write Protect on completion.  Must follow with Read- or Write-set.
414     ; Note: enabling write (WP~ high and inactive) is needed only for the Erase and
415     ; Program functions.
416     ;
417     MEM_PROGRAM
418           Clrwdt                          ; Watch-Dog Timer reset
419           Bsf     STATUS, RP0
420           Clrf    TRISD                   ; Set PORTD=Outputs
421           Bcf     STATUS, RP0
422           Bsf     PORTB, WP               ; WP~ high and inactive, enable Flash Memory write
423           Bcf     PORTC, CE               ; Chip enable CE~, Active low
424           Bsf     PORTB, CLE              ; Command Latch Enable
425           Movlw   0x10                    ; Command 0x10 is Program, transfer buffer Flash
426           Call    MEM_WRITE
```

```
427         Bcf     PORTB, CLE
428 ;
429 ; MEM_STATUS Loops until Flash status bit Ready is set, which should take 0.2 to 1.0 millisecond
430         Call    MEM_STATUS              ; Reads Flash Status register into Wreg
431 ;
432 ; Check for error condition
433         Btfsc   MEMORY_FLAG, MEM_ER_FLG ; Set in MEM_STATUS if error
434         Bsf     MEMORY_FLAG, MEM_PROG   ; Sets condition flag for message
435 ;
436 ; Program function complete
437 MEM_PR_END
438 ;       Bcf     PORTB, WP               ; WP~ low and active, disable Flash Memory write
439         Bsf     PORTC, CE               ; Release CE~, Active low chip enable
440         Retlw   0
441 ;
442 ;*****************************************************************************
443 ;*****************************************************************************
444 ; Erase the entire 32MB of Flash memory.  This function must be done Block-by-
445 ; Block.  If erase failures are detected, the routine attempt to mark the Block
446 ; bad by writing 0xA5A5 and increments the bad Block count.
447 ;
448 MEM_ERASE
449         Clrf    ADD1                    ; Start at memory beginning
450         Clrf    ADD2
451         Clrf    MEMORY_FLAG             ; Clear error condition flags
452         Clrf    MEM_BAD_NUM             ; Zero count of bad blocks
453         Call    SENDDATA                ; Message expected by PC interface S/W
454 ;
455 MEM_ER_LOOP
456         Call    MEM_ERASE_BLOCK
457 ;
458 ; Make LED blink after each Block
459         Movlw   0x02                    ; LED is PORT C, line 1
460         Xorwf   PORTC, f                ; make LED blink
461 ;
462 ; If error condition, mark block bad
463         Btfss   MEMORY_FLAG, MEM_ER_FLG       ; Error from erase status
464         Goto    MEM_ER_NXT              ; Flag not set, continue
465 ;
466 ; Block bad.  The maximum Blocks in the 32MB memory is 2046, so can overflow.
467         Incf    MEM_BAD_NUM, f          ; count number of bad blocks
468         Call    MEM_WR_SET              ; Still pointing to start of block
469         Movlw   0xA5                    ; pattern to mark bad
470         Call    MEM_WRITE               ; write out 0xA5A5
471         Call    MEM_WRITE               ;   . . . might not be able to write since bad block.
472         Call    MEM_PROGRAM             ; Transfer buffer to Flash
473         Call    Fail_Msg                ; Readable message in HyperTerminal
474 ;
475 ; Increment memory to next 16kB block to erase
476 MEM_ER_NXT
477         Clrf    MEMORY_FLAG             ; Clear any condition flag
478         Movlw   0x20                            ; ADD1 increment for 16kB Block
479         Addwf   ADD1, f                         ; Store result in ADD1
480         Btfss   STATUS, C                       ; Carry set if rolled over
481         Goto    MEM_ER_LOOP                     ; No need to check ADD2 yet
482 ;
483 ; ADD1 rolled over, so adjust ADD2.  If ADD2 rolls over, have incremented past
484 ; end of 32MB memory so set flag to indicate that no more records can be stored.
485         Incf    ADD2, f                         ; Increment ADD2 if ADD1 carry
486         Btfsc   STATUS, Z                       ; Check if ADD2 rolled to zero
487         Goto    MEM_ER_END                      ; Rolled to zero, all done
488 ;
489 ; Write message every 4MB erased
490         Movf    ADD2, w                         ; Recover ADD2
491         Andlw   0x1F                            ; Strip upper bits on ADD2
492         Btfsc   STATUS, Z                       ; Lower ADD2 bits zero?
493         Call    Erase_Msg                       ;  yes, send debug message
494         Goto    MEM_ER_LOOP
495 ;
496 ; Finished with erase function
497 MEM_ER_END
```

```
498         Call Status_Msg
499         Retlw   0
500     ;
501     ;*****************************************************************************
502     ;*****************************************************************************
503     ;
504     ; Erase one 16kB Block of Flash memory. Note: enabling write (WP~ high and
505     ; inactive) is needed only for the Erase, Program, and Write buffer functions.
506     MEM_ERASE_BLOCK
507         Clrwdt                      ; Watch-Dog Timer reset
508         Bsf     STATUS, RP0
509         Clrf    TRISD               ; Set PORTD=Outputs
510         Bcf     STATUS, RP0
511         Bsf     PORTB, WP           ; WP~ high and inactive, enable Flash Memory write
512         Bcf     PORTC, CE           ; Chip enable CE~, Active low
513         Bsf     PORTB, CLE          ; Command Latch Enable
514         Movlw   0x60                ; Command 0x60 is Erase
515         Call    MEM_WRITE
516         Bcf     PORTB, CLE
517     ;
518     ; Write Block address.  Erase function is on 16kB blocks.
519         Bsf     PORTB,ALE
520         Movf    ADD1, w
521         Call    MEM_WRITE
522         Movf    ADD2, w
523         Call    MEM_WRITE
524         Bcf     PORTB,ALE
525     ;
526     ; Send confirm command
527         Bsf     PORTB,CLE
528         Movlw   0xD0                ; Erase confirm command
529         Call    MEM_WRITE
530         Bcf     PORTB, CLE
531     ;
532     ; Wait for completion.  Takes up to 20 milliseconds.  MEM_STATUS returns Flash
533     ; Memory status in Wreg
534         Call    MEM_STATUS          ; Loops until Flash sets Ready status
535         Bcf     PORTB, WP           ; WP~ low, Flash Memory write protected
536         Bsf     PORTC, CE           ; Release CE~, Active low chip enable
537         Return
538     ;
539     ;*****************************************************************************
540     ;*****************************************************************************
541     ;
542     ; Flash Status is in Wreg on return, loops until memory shows Ready
543     ; TC58256FT memory Status Bit definition for the TC58256FT memory:
544     ;       Bit 0   Pass = 0, Fail = 1 for a program or erase function
545     ;       Bits 1 - 5 are not used and returned as 0
546     ;       Bit 6   Ready = 1, Busy = 0 for any command
547     ;       Bit 7   Write protect status, 0 = protect, 1 = write enabled
548     ;  Expect result read to be 0xC0 when function is complete.
549     ;
550     MEM_STATUS
551     ; Set up to write command
552         Bsf     STATUS, RP0         ; Select Bank 1 Memory
553         Clrf    TRISD               ; Set PORTD=Outputs
554         Bcf     STATUS, RP0         ; Set Bank 0 Memory
555         Bsf     PORTB, CLE          ; Command Latch Enable
556         Movlw   0x70                ; Command to Read Flash Status
557         Call    MEM_WRITE
558         Bcf     PORTB,CLE
559     ;
560     ; Set up to read result
561         Clrf    PORTD
562         Bsf     STATUS, RP0         ; Select Bank 1 Memory
563         Movlw   0xFF
564         Movwf   TRISD               ; Set PORTD=Inputs
565         Bcf     STATUS, RP0         ; Select Bank 0 Memory
566         Call    MEM_READ            ; Read result in Wreg
567         Movwf   TEMP
568         Btfss   TEMP, 6             ; Bit 6 is Ready / Busy~, still processing if low
```

```
569         Goto    MEM_STATUS
570     ;
571     ; Check for error condition
572         Btfsc   TEMP, 0                 ; Failure flag on bit 0
573         Bsf     MEMORY_FLAG, MEM_ER_FLG       ; Sets condition flag for message
574     ;
575         Return
576     ;
577     ;*****************************************************************************
578     ;*****************************************************************************
579     ;       Test_Hand(ler) Subroutine
580     ;
581     ; Function:
582     ;  On a 16kB block basis, this routine erases the Flash memory, fills the memory
583     ;  with the pattern supplied, then erases again.  Error and progress update
584     ;  messages are sent to the PC after every 1MB.
585     ;
586     ;  The TC58256FT memory includes an auto-verify function that compares the
587     ;  contents of the memory with the buffer after a program operation.  If these
588     ;  do not match, the Fail bit is set on memory status.  The Erase command also
589     ;  does the equivalent with an Erase and Verify function.
590     ;
591     ; Calls:
592     ;       MEM_WR_SET
593     ;       MEM_ERASE_BLOCK
594     ;       MEM_PROGRAM
595     ;
596     ; Registers:
597     ;       Wreg has the test pattern on entry.
598     ;
599     ; Interrupts:
600     ;       Not changed.
601     ;
602     ; Return:
603     ;       WREG=0
604     ;
605     ; Execution Time:
606     ;       ??M Cycles
607     ;       ??sec at 4MHz
608     ;*****************************************************************************
609     ;
610     MEM_TEST
611         Movwf   PATTERN                 ; Wreg has Test Pattern on entry
612         Clrf    ADD1
613         Clrf    ADD2
614         Clrf    MEMORY_FLAG
615     MEM_TSW_LOOP                        ; Loop per memory Page
616     ;
617     ; Check to see if computer command, started by pushing Attention button
618         Btfsc   PORTB, ATTENTION        ; Bit Test of Command Flag
619         Goto    MEM_TST_END             ; Button pushed, begin processing commands
620     ;
621         Clrf    ADD0                    ; Used here to track if Page filled
622         Movlw   0xFE                    ; Set up for two loops
623         Movwf   A8                      ; Tracks which Page half of 512-byte page
624         Call    MEM_WR_SET              ; Starting a new Page, prepare to write
625         Movf    PATTERN, w              ; Recover test pattern to Wreg
626     MEM_TSW_inner                       ; Loop per memory Byte
627         Call    MEM_WRITE
628         Incfsz  ADD0, f                 ; Increment after each byte written
629         Goto    MEM_TSW_inner           ; Address not rolled over to zero, loop
630     ;
631     ; Check if finished one page = 512 bytes, or just half-page = 256 bytes
632         Incfsz  A8, f                   ; A8=0 @ 512 Bytes, A8=FF @ 256 Bytes
633         Goto    MEM_TSW_inner           ; Page not finished, continue writing
634     ;
635     ; Have written 512 bytes into buffer
636         Call    MEM_PROGRAM             ; Transfer from buffer, captures error
637     ;
638     ; Now read back page to see if matches.
639     MEM_TSR_LOOP                        ; Loop per memory Page
```

```
640         Movlw   0xFE                    ; Set up for two loops
641         Movwf   A8                      ; Tracks which Page half of 512-byte page
642         Call    MEM_RD_SET              ; Starting a new Page, prepare to write
643 MEM_TSR_inner                           ; Loop per memory Byte
644         Call    MEM_READ
645         Subwf   PATTERN, w              ; Compare test pattern with read value
646         Btfss   STATUS, Z
647         Bsf     MEMORY_FLAG, MEM_WR_FLG       ; Did not match
648         Incfsz  ADD0, f                 ; Increment after each byte written
649         Goto    MEM_TSR_inner           ; Address not rolled over to zero, loop
650 ;
651 ; Check if finished one page = 512 bytes, or just half-page = 256 bytes
652         Incfsz  A8, f                   ; A8=0 @ 512 Bytes, A8=FF @ 256 Bytes
653         Goto    MEM_TSR_inner           ; Page not finished, continue writing
654 ;
655 ; Check if finished last page in 16kB block.  ADD1=0x1F, 0x3F, etc.
656         Movlw   0x1F                    ; Mask out upper three bits
657         Andwf   ADD1, w                 ; Wreg left with lower five bits
658         Sublw   0x1F                    ; Address for last page in block
659         Btfsc   STATUS, Z
660         Goto    MEM_TS_ER               ; Finished writing Block, so erase
661 ;
662 ; Was not end of block so increment memory to next Page to test
663         Movlw   0x01
664         Addwf   ADD1, f                 ; Increment ADD1 if ADD0 overflow
665         Btfsc   STATUS, C               ; See if overflow
666         Incf    ADD2, f                 ;   yes, increment next address
667         Goto    MEM_TSW_LOOP            ; Start writing next memory Page
668 ;
669 ; Set address to erase this block, then increment to start of new block
670 MEM_TS_ER
671         Clrf    ADD0
672         Movlw   0xE0                    ; Retain only upper three address bits
673         Andwf   ADD1, f                 ; Set ADD1 to start address of this page
674         Call    MEM_ERASE_BLOCK
675 ;
676 ; Make LED blink after each Block tested
677         Movlw   0x02                    ; LED is PORT C, line 1
678         Xorwf   PORTC, f                ; make LED blink
679 ;
680 ; Check if any failures in this block
681         Movf    MEMORY_FLAG, w          ; recover memory error condition flags
682         Btfsc   STATUS, Z               ; Should be zero if no errors
683         Goto    MEM_TS_NXT              ; no errors, go to start of next block
684 ;
685 ; Some error in block, send failure message
686         Incf    MEM_BAD_NUM, f          ; count number of bad blocks
687         Call    Fail_Msg                ; Message with block address
688         Call    MEM_WR_SET              ; Still pointing to start of block
689         Movlw   0xA5                    ; pattern to mark bad
690         Call    MEM_WRITE               ; write out 0xA5A5
691         Call    MEM_WRITE               ;   . . . might not write since bad.
692         Call    MEM_PROGRAM             ; Transfer buffer to Flash
693 ;
694 ; Increment memory to next 16kB block to test
695 MEM_TS_NXT
696         Clrf    MEMORY_FLAG             ; Clear any condition flag
697         Movlw   0x20                            ; ADD1 increment for 16kB Block
698         Addwf   ADD1, f                         ; Store result in ADD1
699         Btfss   STATUS, C                       ; Carry set if rolled over
700         Goto    MEM_TSW_LOOP                    ; No need to check ADD2 yet
701 ;
702 ; ADD1 rolled over, so adjust ADD2.  If ADD2 rolls over, have incremented past
703 ; end of 32MB memory so end test.
704         Incf    ADD2, f                         ; Increment ADD2 if ADD1 carry
705         Btfsc   STATUS, Z                       ; Check if ADD2 rolled to zero
706         Goto    MEM_TST_END                     ; Rolled to zero, all done
707 ;
708 ; Write message every 1MB tested
709         Movf    ADD2, w                         ; Recover ADD2
710         Andlw   0x07                            ; Strip upper bits on ADD2
```

```
711         Btfsc   STATUS, Z                    ; Lower ADD2 bits zero?
712         Call    Status_Msg                  ;  yes, send debug message
713         Goto    MEM_TSW_LOOP
714 ;
715 ; Done with test
716 MEM_TST_END
717         Call Status_Msg
718         Retlw   0
719 ;
720 ;*******************************************************************************
721 ;*******************************************************************************
722 ;       Fill Test Subroutine
723 ; Function:
724 ;  Fills the number of Flash memory Records specified with known values.  This
725 ;  allows tests of the read routine.  Follows the same Frame structure as
726 ;  normal written data.  Use the HyperTerminal, and Command string:
727 ;  "0", "F", <parm MSB>, <parm LSB>
728 ;  where the <parm ..> is an ASCII character with the least significant nibble
729 ;  set to the value desired.  The value indicates the number of records to fill.
730 ;  For example, to fill 5 records, the command string is:
731 ;  0F05
732 ;  because the ASCII value for "5" is 35 hex, so the least sig. nibble is a 5 as
733 ;  needed.  (The serial interrupt routine combines the nibbles into 8-bit values)
734 ;
735 ; Interrupts:
736 ;  No change.  Serial Interrupts must be enabled prior to call to allow the PC
737 ;  to stop data upload with the Cntl-C character.
738 ;
739 ;  Return:
740 ; Wreg=0
741 ;
742 ;*******************************************************************************
743 ;*******************************************************************************
744 ;
745 FILL_TEST
746         Clrwdt                          ; Reset 2.4 second Watch-dog Timer
747         Bcf     STATUS, RP0             ; Make sure pointing to Memory Bank 0
748         Clrf    MEMORY_FLAG             ; Resets memory condition flags
749 ;
750 ; Locate the next blank memory location.  Also returns last Record number used.
751         Call    MEM_FIND
752 ;
753 ; Get parameter passed with command, if any.  If no parameters, set to fill one
754 ; Record.
755         Movlw   0x01                    ; Default to one Record fill
756         Movwf   R_MEM_REC_NUM           ; 0x01 is first 128kB Record
757         Movlw   CMD_Param_Start                 ; Start of parameter stack
758         Movwf   FSR                     ; Place into stack pointer
759         Movf    INDF, w                 ; Recover CMD_PARAMS=parameter count
760         Btfsc   STATUS, Z               ; Test if zero parameters on stack
761         Goto    FILL_END_REC            ; no parameters passed, start with default
762 ;
763 ; Adjust Command Parameter stack pointer to first parameter passed
764         Incf    FSR, f                  ; Point to first parameter
765         Movf    INDF, w                 ; Get value
766         Movwf   R_MEM_REC_NUM           ; Store as number Records to fill
767 ;
768 ; Now add to current record number to determine stopping record number.
769 FILL_END_REC
770         Movf    MEM_REC_NUM, w          ; Get current record count
771         Addwf   R_MEM_REC_NUM, f        ; Add to number of records to fill
772         Movlw   0xFF                    ; Does not affect carry bit
773         Btfsc   STATUS, C               ; See if caused overflow
774         Movwf   R_MEM_REC_NUM           ; Overflow, set to maximum
775 ;
776 ; Position memory to start filling memory.
777 FILL_START
778         Clrf    MEM_FRM_NUM             ; Clear Memory Page count
779         Incf    MEM_REC_NUM, f          ; Add one to number of records stored
780 ;
781 FILL_LOOP
```

```
782   ;
783   ; Check to see if computer command, started by pushing Attention button
784         Btfsc   PORTB, ATTENTION        ; Bit Test of Command Flag
785         Goto    FILL_COMPLETE           ; Button pushed, begin processing commands
786   ;
787   ; Set up memory to accept data at the 512-Byte Page address pointed to by
788   ; address Bytes ADD0, 1, 2.
789         Call    MEM_WR_SET
790   ;
791   ; Increment time values for next record, 16ms per Frame.
792         Movlw   0x10
793         Addwf   TIME0, f
794         Movlw   0x01                    ; increment value for subsequent
795         Btfsc   STATUS, C               ; If Time 0 overflowed, increment Time 1
796         Addwf   TIME1, f
797         Btfsc   STATUS, C               ; Look for overflow on Time 1
798         Addwf   TIME2, f                ; Time 1 overflow, increment Time 2
799         Btfsc   STATUS, C               ; Still clear from Time 1, or clear/set by 2
800         Incf    TIME3, f                ; Time 2 overflow, increment Time 3
801   ;
802   ; Write sync pattern into the first two bytes in each frame (512-byte Page).
803   ; This provides a known value to detect used, bad, or blank Flash Memory Pages.
804   ; Also, write all once-per-page values at the start of the page.
805         Call    MEM_Header
806   ;
807   ; Initialize measurement counter to record 28 analog sets per Frame
808         Movlw   0x1C                    ; Frame contains 28 analog meas. sets
809         Movwf   MeasSet_Cnt             ;   initialize counter variable
810   ;
811   ; Set up simulated analog measurement loop
812   FILL_F_LOOP
813         Movlw   0x08                    ; Fill 9 channels, 0 to 8
814         Movwf   Channel_Cnt
815   ;
816   ; Simulate Analog measurement with large value in LSB, small value in MSB.
817   FILL_M_LOOP
818         Comf    Channel_Cnt, w          ; simulates LSB of data
819         Call    MEM_WRITE               ; Store in Flash memory
820         Movf    Channel_Cnt, w          ; simulates MSB of data
821         Call    MEM_WRITE               ; Store in Flash memory
822   ;
823         Decfsz  Channel_Cnt, f          ; go to next value
824         Goto    FILL_M_LOOP
825   ;
826   ; Check if done with frame
827         Decfsz  MeasSet_Cnt, f          ; sub-frame counter, want 28 sets
828         Goto    FILL_F_LOOP             ; not done with 28 sets yet
829   ;
830   ; End of Frame loop. Command transfer of Flash internal buffer to memory
831         Clrwdt                          ; Reset 2.4 second Watch-dog Timer
832         Call    MEM_PROGRAM             ; Transfer internal buffer to Flash
833   ; Note: Maximum write delay is 1 millisecond
834   ;
835   ; Increment memory address to next 512-Byte Page.  Add0 is the Byte location
836   ; within the Page, so is not incremented.
837   FILL_F_INCR
838         Incf    ADD1, f                 ; Incf instruction doesn't set C flag
839         Btfss   STATUS, Z               ; See if rolled over, overflow
840         Goto    FILL_INC_FR             ; Not rolled over, don't need to check rest
841   ;
842   ; Make LED blink
843         Movlw   0x02                    ; LED is PORT C, line 1
844         Xorwf   PORTC, f                ; make LED blink
845   ;
846   ; ADD1 rolled over, so increment next significant address.  Check for end of
847   ; memory.  If end, both ADD1 and ADD2 will have rolled to zero.
848         Incf    ADD2, f                 ; increment next address
849         Btfsc   STATUS, Z               ; Will have rolled to zero if full
850         Goto    FILL_COMPLETE
851   ;
852   ; Increment frame counter
```

Code Listing – Flash.asm

```
853     FILL_INC_FR
854             Incfsz  MEM_FRM_NUM, f          ; Frame counter, rolls to 0x00 from 0xFF
855             Goto    FILL_LOOP
856     ;
857     ; 128kB Record complete - See if need to loop again.
858     FILL_CK_COMP
859             Call    Status_Msg              ; Sends Output location to terminal
860             Movf    R_MEM_REC_NUM, w        ; Get finish Record value
861             Subwf   MEM_REC_NUM, w          ; Subtract current Record number
862             Btfss   STATUS, C               ; Carry set if R_ < MEM_REC_NUM
863             Goto    FILL_START
864     ;
865     ; End of memory acquisition / write.
866     FILL_COMPLETE
867             Return
868     ;
869     ;*******************************************************************************
870     ;************************ END of FILE Flash.asm ********************************
871     ;*******************************************************************************
872     ;
873             END
```

Code Listing – Rd_Hand.asm

```
1   ; File name: "rd_hand.asm"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Routines to read memory and send data via serial interface
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       1999 - Adapted from MilliPen.asm for Sensor Ball, TA Rohwer
11  ;       November 2000 - Version 2, TA Rohwer
12  ;       17 Mar 2001 - Version 3 Changes requested by P&G, ME Partridge
13  ;       14 Jun 2001 - Include checksum features for data integrity at 115.2k Baud
14  ;       12 Aug 2001 - Version 4 Changes, ME Partridge
15  ;         Remove the automatic continuation function during upload. This may
16  ;              have caused 512-Byte Pages to be missed on upload.
17  ;         Add reduced-function upload command "r" w/o Sync check
18  ;         Use interface timeout subroutines
19  ;         Correct upload of Time LSW
20  ;       11 Dec 2001 having problem with serial receive -- getting spurious
21  ;           characters.  Reset receive after each block transmitted.
22  ;
23  ;*****************************************************************************
24  ;       Read Hand(ler) Subroutine
25  ; Function:
26  ;  Reads Records from memory from Start Record until either Cntl-C is received
27  ;  from the PC uploading the data, or the end of the 32MB Flash memory is
28  ;  reached.  Default for Start Record is 1, the first record. Data are stored
29  ;  in Flash memory as 16-bit words, Least Significant Byte first, and are
30  ;  uploaded to the PC this way.
31  ;
32  ;  Seven, 80-Byte Major Frames are aligned within one 512-Byte Flash memory
33  ;  Page.  The data are stored in the Flash memory Page as follows:
34  ;       Sync bytes Sync0, Sync1 (0xEB90)
35  ;       Page number and Record number
36  ;       Time count, 32 bits with LSB first
37  ;       28 sets of nine, two-byte analog measurements, Channel 0 thru 9 skip 7
38  ;  These are rearranged to create the seven Major Frames with some values
39  ;  repeated among the seven Major Frames (Page number, Record number, Page
40  ;  address LS Byte, Page address MS Byte) to the seven 80-Byte Major Frames
41  ;  totaling 560 Bytes.  The Time variable is incremented for each Major Frame,
42  ;  based upon the initial value stored in the Flash memory Page.  So, the Major
43  ;  Frame is constructed of the following values:
44  ;       one set of nine, two-Byte analog measurements
45  ;       Page number, Record number
46  ;       next set measurements
47  ;       Time LS+2 Byte, Time MS Byte
48  ;       next set measurements
49  ;       Time LS Byte, Time LS+1 Byte
50  ;       next set measurements
51  ;       Flash Memory Page LS Byte address, Flash Memory Page MS Byte address
52  ;
53  ;  To accommodate re-transmitting a data Frame if errors are detected, the
54  ;  Flash memory Page is re-read when a Negative Acknowledge (NAK) is sent. After
55  ;  each Byte is transmitted, its value is added to the 16-bit checksum value.
56  ;  Then, the Frame is uploaded to the PC followed by the checksum.  If the data
57  ;  are error-free, an Acknowledge character (hexadecimal 0x06) is sent from the
58  ;  PC and data transmission continues with the next Flash Memory Page.
59  ;  Otherwise, the NAK character (hexadecimal 0x15) is sent and the Frame is
60  ;  transmitted again.
61  ;
62  ; Subroutines in this file:
63        GLOBAL  READ_HAND
64  ;
65  ; Calls:
66        EXTERN  TX_WREG
67        EXTERN  Timeout_Chk          ; Tests if interface has timed out
68        EXTERN  MEM_READ             ; Flash.asm, Return value from Flash memory
69        EXTERN  MEM_WRITE            ; Flash.asm, Writes Wreg value to Flash memory
70        EXTERN  MEM_REC_FIND         ; Finds a particular Record number
71        EXTERN  MEM_RD_SET           ; Sets Flash memory to begin reading
```

Code Listing – Rd_Hand.asm

```
72  ;
73  ; Registers:
74          EXTERN  MEM_REC_NUM             ; Current Flash memory Record, 1 - 64
75          EXTERN  MEM_FRM_NUM             ; Current Record Frame (Page), 0 - 255
76          EXTERN  ADD0                    ; Flash Memory address A7 .. A0
77          EXTERN  ADD1                    ; Flash memory address A16 .. A8
78          EXTERN  ADD2                    ; Flash memory address A24 .. A17
79          EXTERN  SER_FLAG                ; Control character flags Xon, Ack, etc.
80          EXTERN  MEMORY_FLAG             ; Memory Status and test flags
81          EXTERN  PROG_FLAG               ; Program control, SIMPLE_FLG used here
82          EXTERN  CMD_PARAMS
83          EXTERN  R_Sync_Byte0            ; Values read from memory
84          EXTERN  R_Sync_Byte1
85          EXTERN  R_MEM_REC_NUM
86          EXTERN  R_MEM_FRM_NUM
87          EXTERN  R_TIME3
88          EXTERN  R_TIME2
89          EXTERN  R_TIME1
90          EXTERN  R_TIME0
91          EXTERN  CMD_TIMEL               ; Time-out LSByte, 285ms / bit
92          EXTERN  CMD_TIMEH               ; Time-out MSByte
93          EXTERN  CKSM0                   ; Checksum LS Byte, data upload to PC
94          EXTERN  CKSM1                   ; Checksum MS Byte
95          EXTERN  BLCK0                   ; Count of Frames uploaded, LS Byte
96          EXTERN  BLCK1                   ; Count of Frames uploaded, MS Byte
97          EXTERN  MeasSet_Cnt             ; Loop counter, 28 sets of 9 meas. / page
98          EXTERN  Channel_Cnt             ; Loop counter, analog channels 0 to 9 skip 7
99          EXTERN  TEMP                    ; Scratch register for Debug
100 ;
101 ; Interrupts:
102 ;  No change.  Serial Interrupts must be enabled prior to call to allow the PC
103 ;  to stop data upload with the Cntl-C character.
104 ;
105 ;  Return:
106 ; Wreg=0
107 ;
108 ; include files:
109         #include "P16C774.INC"         ;Standard Header File for PIC16C773
110                                         ;Includes all Register Definitions,
111                                         ;RAM Definitions, & Configuration Bits
112         #include "ball_equ.inc"                 ;EQU Declarations, equivalence
113 ;
114 ;*****************************************************************************
115 ;*****************************************************************************
116 ; Macro used only in the Rd_Hand routine to extract data from memory and pass
117 ; it on to the PC.
118 ;*****************************************************************************
119 ;*****************************************************************************
120 ;       Build Minor-Frame Macro
121 ;               4/16/2001 ME Partridge
122 ; Function:
123 ;  Reads the ADC data from Flash memory in sets of nine, two-byte values and
124 ;  uploads it to the PC.  Then, two bytes sub-commutated values are sent. The
125 ;  Major Frame consists of four Minor-Frames.
126 ;
127 ; Arguments:
128 ;  Loop         Unique identifier for each instance of this macro
129 ;  Value0       The first sub-commutated value sent
130 ;  Value1       The second sub-commutated value
131 ;
132 ; Calls:
133 ;  TX_WREG      Passes data to the USART to be transmitted to the PC
134 ;
135 ; Macros used:
136 ;  MEM_READ     Handles Flash control signals
137 ;  ADC_Read     Sets up the analog mux and waits for ADC response
138 ;
139 ; Registers:
140 ;  Channel_Cnt Loop counter to read one data set
141 ;
142 ; Revisions:
```

```
143   ;  4/16/2001 First revision
144   ;
145   ;*******************************************************************************
146   Sub_Frame      MACRO   Loop, Value0, Value1
147   Sub_Frame_#v(Loop)
148          Movlw   0x12
149          Movwf   Channel_Cnt           ; Counter to move 9 values x 2 Bytes ea
150   ;
151   ; Analog data loop: get data for Analog channels 0 to 6, 8, 9 for 9 values total
152   Sub_Frame_#v(Loop)_LOOP
153          Call    MEM_READ                      ; Analog LSBYTE
154          Call    TX_WREG               ; Upload Byte to PC
155   ;
156   ; Do checksum calculation.  TX_WREG leaves data in Wreg
157          Addwf   CKSM0, f              ; calculate checksum LS Byte
158          Btfsc   STATUS, C
159          Incf    CKSM1, f              ; adjust checksum MS Byte if Carry
160   ;
161   ; Increment the measurement number, then test if greater than 18.  If not, repeat
162   ; with the next analog channel.
163          Decfsz  Channel_Cnt, f        ; Check if completed one set of data
164          Goto    Sub_Frame_#v(Loop)_LOOP       ; Acquire and write next channel
165   ;
166   ; Close with two bytes of other values
167          Movf    Value0, w
168          Call    TX_WREG               ; Upload Byte to PC
169   ;
170   ; Do checksum calculation.  TX_WREG leaves data in Wreg
171          Addwf   CKSM0, f              ; calculate checksum LS Byte
172          Btfsc   STATUS, C
173          Incf    CKSM1, f              ; adjust checksum MS Byte if Carry
174   ;
175   ; Second value
176          Movf    Value1, w
177          Call    TX_WREG               ; Upload Byte to PC
178   ;
179   ; Do checksum calculation.  TX_WREG leaves data in Wreg
180          Addwf   CKSM0, f              ; calculate checksum LS Byte
181          Btfsc   STATUS, C
182          Incf    CKSM1, f              ; adjust checksum MS Byte if Carry
183   ;
184                  ENDM
185   ;
186   ;*******************************************************************************
187   ;*******************************************************************************
188   ;
189   READ_PROG      CODE                   ;relocatable code in Program EPROM
190   ;
191   READ_HAND
192          Clrwdt                        ; Reset 2.4 second Watch-dog Timer
193          Bcf     STATUS, RP0           ; Make sure in Memory Bank 0
194          Bsf     PROG_FLAG, LED_BLINK  ; Setup for LED blink pattern
195          Clrf    CMD_TIMEL             ; Time-out LSByte, 285ms / bit
196          Clrf    CMD_TIMEH             ; Time-out MSByte
197          Clrf    BLCK0                 ; LS count of Frames uploaded
198          Clrf    BLCK1                 ; MS count
199          Clrf    SER_FLAG              ; Clears left-over control characters
200          Clrf    MEMORY_FLAG           ; Resets memory condition flags
201          Clrf    ADD1                  ; Flash Page address
202          Clrf    ADD2
203   ;
204   ; Get parameter passed with command, if any.  If no parameters, leave ADD1 set
205   ; to 0x00, the start of memory.  A "block" of data uploaded is equal to one Page
206   ; of Flash memory (512 Bytes), expanded to 560 Bytes when converted to standard
207   ; telemetry frames.  A "block" does not equal a Flash memory Block, which is
208   ; 16k Bytes, rather, it is the term used in the host computer software.  One
209   ; "block" contains 7 frames of 80 bytes each.  Data are uploaded until stopped
210   ; by the host PC when it transmits a Cntl-C control character.
211   READ_PARM
212          Movlw   CMD_Param_Start              ; Memory location for parameter stack
213          Movwf   FSR                   ; Place in stack pointer
```

```
214         Movf    INDF, w             ; =CMD_PARAMS, count of parameters sent
215         Btfsc   STATUS, Z           ; Test if zero parameters on stack
216         Goto    READ_RECORD         ; no parameters passed, start with default
217 ;
218 ; Read first parameter passed, MS Byte of address
219         Incf    FSR, f              ; Adjust stack pointer
220         Movf    INDF, w             ; Byte 1, MSByte address
221         Movwf   ADD2
222 ;
223 ; If no more parameters, proceed with read
224         Decfsz  CMD_PARAMS, f       ; Recover number of parameters
225         Goto    READ_2ND_PARM       ; Parameter cound still > 0
226         Goto    READ_RECORD         ; only one parameter passed
227 ;
228 ; Read second paramter passed, LS Byte of address
229 READ_2ND_PARM
230         Incf    FSR, f              ; Adjust stack pointer
231         Movf    INDF, w             ; Recover parameter
232         Movwf   ADD1                ; Place in address
233 ;
234 ; Position memory to start of Record selected.
235 READ_RECORD
236 ;
237 ; Test if "Smart PCM Device" upload, command "r"
238         Btfsc   PROG_FLAG, SIMPLE_FLG ; Set if Command "r"
239         Goto    READ_XON            ; "Smart PCM Device" upload, skip to Read
240         Call    MEM_REC_FIND        ; Find first valid data Record
241 ;
242 ; Wait for OK from PC before begining data upload loop.  Need WDT reset because
243 ; need operator input to select file name and number of Records to upload.
244 READ_XON
245         Clrwdt                      ; Reset 2.4 second Watch-dog Timer
246         Call    Timeout_Chk         ; Used to blink LED while wait
247         Btfsc   PORTB, ATTENTION    ; Check if operator wants to terminate
248         Goto    READ_END            ; Button pushed, skip to end of read
249         Btfsc   SER_FLAG, NAK_FLG   ; Negative acknowledge
250         Goto    READ_LOOP           ; . . . take it as Go signal
251         Btfss   SER_FLAG, XON_FLG   ; PC sends Xon when ready
252         Goto    READ_XON
253 ;
254 ;****************** Start of Flash Memory Read Loop ***************************
255 ;
256 ; Begin loop to read Flash memory and write formatted results to the PC.
257 READ_LOOP
258         Clrwdt                      ; Reset 2.4 second Watch-dog Timer
259         Btfsc   SER_FLAG, CNTRLC_FLG ;Test if transfer Interrupt Flag
260         Goto    READ_END            ;  !=0, Transfer interrupted by Ctrl-C
261 ;
262 ; Blink LED for each Memory Page read
263         Movlw   0x02                ; LED is PORT C, line 1
264         Xorwf   PORTC, f            ; make LED blink
265 ;
266 ; See if end of memory has been reached, if so will be sending blank records
267         Btfsc   MEMORY_FLAG, MEM_FIND_FLG   ; End of memory flag
268         Goto    READ_SEND_SYNC              ; Yes at end, skip to send sync
269 ;
270 ; Read the preamble bytes
271         Call    MEM_RD_SET          ; Position to read next Memory page
272         Call    MEM_READ            ; Sync Byte 0
273         Movwf   R_Sync_Byte0
274         Btfsc   PROG_FLAG, SIMPLE_FLG ; Set if Command "r"
275         Goto    READ_Sync1          ; Simple upload, proceed with Read
276 ;
277 ; Test if equal to Sync 0.  If not, bad memory Page.
278         Sublw   Sync_Byte0
279         Btfss   STATUS, Z           ; if match, will be zero
280         Goto    Next_Page           ; no match, skip to next memory page
281 ;
282 READ_Sync1
283         Call    MEM_READ            ; Sync Byte 1
284         Movwf   R_Sync_Byte1
```

Code Listing – Rd_Hand.asm

```
285         Btfsc   PROG_FLAG, SIMPLE_FLG ; Set if Command U
286         Goto    READ_REST             ; Simple upload, proceed with Read
287     ;
288     ; Test if equal to Sync 1.  If not, bad memory Page.
289         Sublw   Sync_Byte1
290         Btfss   STATUS, Z             ; if match, will be zero
291         Goto    Next_Page             ; no match, skip to next memory page
292     ;
293     ; Read remaining preamble data bytes.  Need to match sequence written by the
294     ; RECORD macro
295     READ_REST
296         Call    MEM_READ
297         Movwf   R_MEM_FRM_NUM
298         Call    MEM_READ
299         Movwf   R_MEM_REC_NUM
300         Call    MEM_READ
301         Movwf   R_TIME2
302         Call    MEM_READ
303         Movwf   R_TIME3
304         Call    MEM_READ
305         Movwf   R_TIME0
306         Call    MEM_READ
307         Movwf   R_TIME1
308     ;
309     ; Send the Sync Bytes and the Frame-group being transmitted
310     READ_SEND_SYNC
311         Movlw   Sync_Byte0            ; Start Frame with sync bytes
312         Call    TX_WREG               ; USART Transmit to PC
313         Movlw   Sync_Byte1
314         Call    TX_WREG
315         Movf    BLCK0, w              ; Count of Frame groups uploaded
316         Call    TX_WREG
317         Movf    BLCK1, w
318         Call    TX_WREG
319     ;
320     ; Each Flash memory Page (512 Bytes) contains 28 sets of nine, two-Byte analog
321     ; measurements.  The following loop reads the Page to upload 7 Major Frames of
322     ; 72 analog Bytes (504 Bytes read) plus the previously read eight Bytes of
323     ; preamble data (512 Bytes total read).  One set of nine, two-Byte analog
324     ; measurements is sent in each sub-Frame plus two Bytes from preamble.
325         Clrwdt                        ; Reset 2.4 second Watch-dog Timer
326         Movlw   0x07
327         Movwf   MeasSet_Cnt           ; Set to read 7 Frames on one Flash Page
328     ;
329     ; Get ready to calculate checksum as data Frame bytes are transmitted
330         Clrf    CKSM0                 ; Checksum LS Byte
331         Clrf    CKSM1
332     ;
333     ; Loop to read and upload 80-Byte Major Frames.  LS Byte is sent first.
334     FRAME_LOOP
335     ;
336     ; Check if at Memory end, if so just send blanks
337         Btfsc   MEMORY_FLAG, MEM_FIND_FLG     ; End of memory flag
338         Goto    Blank_Page
339     ;
340         Sub_Frame       0, R_MEM_FRM_NUM, R_MEM_REC_NUM
341         Sub_Frame       1, R_TIME2, R_TIME3
342         Sub_Frame       2, R_TIME0, R_TIME1
343         Sub_Frame       3, ADD1, ADD2
344     ;
345     ; Skip over blank data transmit
346         Goto    FRAME_INCR
347     ;
348     ; Memory end has been reached, keep sending blank data
349     Blank_Page
350         Movlw   0x50                  ; 80-byte Frame
351         Movwf   Channel_Cnt           ; Re-use this counter
352     ;
353     ; All-zero data add to zero Checksum, so don't need to accumulate here
354     Blank_Frame
355         Movlw   0x00                  ; Send data as all zero
```

```
356         Call    TX_WREG                 ; Transmit blank value
357         Decfsz  Channel_Cnt, f          ; Count of bytes sent in blank Frame
358         Goto    Blank_Frame             ; not done with 80 Bytes yet
359   ;
360   ; Check to see if all Frames uploaded from current memory Page
361   FRAME_INCR
362         Decfsz  MeasSet_Cnt, f          ; Frame counter, want 7 ea. 80-byte
363         Goto    FRAME_LOOP              ; not done with 7 sets yet
364   ;
365   ; Having problem with receive function.  Reset receive queue to ensure able
366   ; to read the ACK / NAK character.
367         Bcf     STATUS, RP0             ; Select Bank 0 Memory
368         Bcf     RCSTA, CREN             ; Momentarily disable receive, clear OERR
369         Movf    RCREG, w                ; Empty receive buffer, two deep
370         Movf    RCREG, w                ; Potential second byte from receive
371   ;
372   ; Clear all control character flags.
373         Clrf    SER_FLAG
374   ;
375   ; Send Checksum bytes, LS Byte first
376   FRAME_CKSM
377         Movf    CKSM0, w
378         Call    TX_WREG
379         Movf    CKSM1, w
380         Call    TX_WREG
381   ;
382   ; Wait until transmit shift register (TSR) is empty is empty.  If the TSR is not
383   ; empty, wait.  Note: this bit is only set after the first transmit
384   FRAME_TX_WAIT
385         Bsf     STATUS, RP0             ; Select Bank 1 memory
386         Btfss   TXSTA, TRMT             ; Set to 1 when TSR empty
387         Goto    FRAME_TX_WAIT           ; shift register empty, load new data
388   ;
389         Bcf     STATUS, RP0             ; Select Bank 0 memory
390         Bsf     RCSTA, CREN             ; Continuous receive enable
391   ;
392   ; Having troubles missing ACK character from PC.  If missed, PC read will time-
393   ; out, and ask for re-transmit, so will be corrected there.  This loop can only
394   ; be exited by receiving a Ctrl-C, Ack, Nak, or pressing the ATTENTION buttion.
395   WAIT_ACK_NAK
396         Clrwdt                          ; Reset 2.4 second Watch-dog Timer
397         Btfsc   PORTB, ATTENTION        ; Check if operator wants to terminate
398         Goto    READ_END                ; Button pushed, skip to end of read
399         Btfsc   SER_FLAG, CNTRLC_FLG    ; Test if transfer Interrupt Flag
400         Goto    READ_END                ; Transfer interrupted by Ctrl-C
401         Btfsc   SER_FLAG, ACK_FLG       ; Acknowledge flag test
402         Goto    WAIT_OVER               ; OK to proceed
403   ;
404   ; The PC interface program may send Ack (0x06), Nak (0x15), or if the interface
405   ; program had some communication time-outs, Xon (0x11).  Treat Nak and X-on
406   ; control characters the same way.
407         Btfsc   SER_FLAG, NAK_FLG       ; Negative acknowledge
408         Goto    READ_LOOP               ; . . . got error, retransmit
409         Btfsc   SER_FLAG, XON_FLG       ; Treat like NAK
410         Goto    READ_LOOP
411         Goto    WAIT_ACK_NAK            ; Nothing received, keep checking
412   ;
413   ; Acknowledge received - Frame group received error-free.
414   ; Adjust count of Frame groups successfully uploaded
415   WAIT_OVER
416         Movlw   0x01                    ; Increment count of Frames uploaded
417         Addwf   BLCK0, f                ; Frame count LS Byte
418         Btfsc   STATUS, C               ; See if LS Byte increment caused Carry
419         Incf    BLCK1, f                ; Got Carry, adjust MS Byte
420   ;
421   ; End of Frame loop. Increment the memory address to the next 512-byte Page.
422   Next_Page
423   ;
424   ; Check to see if already read all Records stored in Flash memory
425         Btfsc   MEMORY_FLAG, MEM_FIND_FLG     ; End of memory flag
426         Goto    READ_LOOP                     ; Just loop, will send blanks
```

Code Listing – Rd_Hand.asm

```
427   ;
428           Movlw   0x01
429           Addwf   ADD1, f               ; Incr instruction doesn't set C flag
430           Btfsc   STATUS, C             ; See if overflow
431           Incf    ADD2, f               ;   yes, increment next address
432   ;
433   ; Check for end of memory.  If end, both ADD1 and ADD2 will have rolled to zero
434           Movf    ADD1, w
435           Iorwf   ADD2, w
436           Btfsc   STATUS, Z
437           Bsf     MEMORY_FLAG, MEM_FIND_FLG     ; Set the End of memory flag
438           Goto    READ_LOOP                     ; Keep going until PC says stop
439   ;
440   ;******************** End of Flash Memory Read Loop *************************
441   ;
442   ; Finished reading all frames
443   READ_END
444           Bcf     STATUS, RP0           ; Make sure in Memory Bank 0
445           Clrf    SER_FLAG              ; Reset transfer Interrupt Flag
446           Bsf     PORTC,CE              ; Disable Chip Enable
447           Retlw   0                     ; Return to CMD_HAND, WREG=0
448   ;
449   ;*******************************************************************************
450   ;************************** END of FILE Rd_Hand.asm ****************************
451   ;*******************************************************************************
452           END
```

Code Listing – Ser_Hand.asm

```
1   ; File name: "ser_hand.asm"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Routines to process a serial interrupt and other serial interface functions
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       1999 - Adapted from MilliPen.asm for Sensor Ball, TA Rohwer
11  ;       November 2000 - Version 2, TA Rohwer
12  ;       17 March 2001 - Version 3 Changes requested by P&G, ME Partridge
13  ;       14 Jun 2001 - Add control characters for data transfer integrity
14  ;       12 Aug 2001 - Version 4, ME Partridge
15  ;         Put Baud setting as a subroutine for consistency
16  ;         Combine nibbles of successive parameters to build Bytes
17  ;         Echo input Parameters on first echo, then combined Bytes on "V" reply
18  ;         Expand allowable command character set to A - z (lower case z)
19  ;         Use String transmit function for messages
20  ;         Add a Hexidecimal translate / transmit step for messages
21  ;       11 December 2001 - having problems with data upload, receiving characters
22  ;           in the > 0x1F range instead of the control character transmitted.
23  ;         Change so serial interrupt does not call subroutine - stack concern
24  ;
25  ;*****************************************************************************
26  ;       Receive SER(ial)_HAND(ler) -- Interrupt Handler
27  ; Function:
28  ;  Processes Serial Interrupts.  Control characters are received, and the
29  ;  appropriate bit in PROG_FLAG or SER_FLAG is set.  If a command phrase is
30  ;  being sent, the progress is tracked using flags in SER_STATE.  The command
31  ;  character is temporarily stored in RC_TEMP.  Once the command phrase is
32  ;  terminated by CR, LF, The command is echoed to the PC.  If the proper command
33  ;  was echoed, then the PC transmits the "V" verify command, and RC_TEMP is
34  ;  moved to CMD_CHAR for processing by CMD_HAND elsewhere, and the Command Flag
35  ;  is set.
36  ;
37  ;  The command phrase sequence is:
38  ;         1 Receive character "0" (zero)
39  ;         2 Receive command character, must be in A - z
40  ;         3 Optionally, receive any parameters, placed in the CMD_PARAMS stack
41  ;         4 Terminate the phrase with CR and LF, in either order
42  ;         5 Sensor Ball echoes command and any parameters to PC
43  ;         6 PC responds with "0", "V", CR, LF if the correct, Cntl-C otherwise.
44  ;         7 Sensor Ball echoes command to PC again, but with processed parameters.
45  ;         8 Command processing begins if Step 6 received.
46  ;
47  ;  The phrase will be aborted if a Ctrl-C is received, a character other than
48  ;  "0" is sent first, or if a command character not in A - z is sent following
49  ;  the "0" character.
50  ;
51  ;  All valid command sequences, without the passed parameters, are Echoed back
52  ;  to the PC when the command phrase is complete.  An invalid command is Echoed
53  ;  back with a "?" in the response.  Cntl-C, X-On, X-Off, and any other control
54  ;  character are not Echoed.
55  ;
56  ; Subroutines in this file:
57        GLOBAL  SENDDATA              ; Sends standard completion message to PC
58        GLOBAL  Fail_Msg              ; Sends failure text message to PC
59        GLOBAL  Status_Msg            ; Sends PC update message during memory test
60        GLOBAL  Erase_Msg             ; Sends PC update message during memory erase
61        GLOBAL  TX_WREG               ; Transmits on RS-232 serial interface
62        GLOBAL  TX_String             ; Uses TX_WREG to send a string
63        GLOBAL  Baud_Set              ; Sets Baud rate + configure serial port
64  ;
65  ; Calls:
66  ;       none
67  ;
68  ; Registers:
69        EXTERN  ADD0, ADD1, ADD2      ; Flash memory address
70        EXTERN  RC_TEMP               ; Received temporary Command Storage
71        EXTERN  RC_CHAR               ; ASCII Character Storage
```

Code Listing – Ser_Hand.asm

```
72          EXTERN  CMD_CHAR              ; Command character for CMD_HAND, in A - Z
73          EXTERN  SER_STATE             ; Progress building the Command Phrase
74          EXTERN  PROG_FLAG             ; bits COMMAND,REINIT,CMDERROR,STRERROR
75          EXTERN  SER_FLAG              ; Control character flags
76          EXTERN  MEMORY_FLAG           ; Error condition from memory
77          EXTERN   CMD_CHAR             ; ASCII character for command
78          EXTERN   CMD_PARAMS           ; Command Parameter count, top of stack
79          EXTERN  TEMP                  ; Temporary, used here in Baud set routine
80          EXTERN  TEMP_INT              ; Temporary for interrupt routine
81          EXTERN  LOOP_INT              ; Loop counter for interrupt routine
82    ;
83    ; Serial output starting location for strings in program memory
84          EXTERN  Look_Hi               ; Used to load PCLATH value
85          EXTERN  Look_Lo               ; Used to load PCL
86          EXTERN  List_Fail_Msg         ; String for Memory failure message
87          EXTERN  List_Stat_Msg         ; String for Memory fill / test progress
88          EXTERN  List_Eras_Msg         ; String for Memory erase progress
89          EXTERN  List_Mem_Flag         ; String for Memory Flag value
90    ;
91    ; Attention mode serial interface time-out registers
92          EXTERN  CMD_TIMEL             ; Time-out LSByte, 285ms / bit
93          EXTERN  CMD_TIMEH             ; Time-out MSByte
94    ;
95    ; Registers to push data during interrupt, accessible regardless of Memory Bank selected.
96          EXTERN  STACK_Wreg            ; Working Register holding during interrupts
97          EXTERN  STACK_Status          ; Status Register holding during interrupts
98          EXTERN  STACK_FSR             ; Indirect pointer holding during interrupts
99    ;
100   ; Interrupts:
101   ;  Global, Peripheral, Serial Interrupts are enabled prior to call.
102   ;
103   ; Return:
104   ;       Wreg=0
105   ;
106   ; include files:
107         #include "P16C774.inc"        ;Standard Header File for PIC16C773
108                                       ;Includes all Register Definitions,
109                                       ;RAM Definitions, & Configuration Bits
110         #include "ball_equ.inc"              ;EQU Declarations, equivalence
111   ;
112   ;*******************************************************************************
113   ;*******************************************************************************
114   ;*************************** Interrupt Vector **********************************
115   PERIPH_VEC     CODE    0x04           ; Peripheral Interrupt Handler
116         Goto    PERIPH_ISR             ; Not enough room to locate ISR here
117   ;
118   ;*******************************************************************************
119   ;************************* Interrupt Service Routine ***************************
120   ;
121   ; Function:
122   ;  To identify source of Peripheral Interrupt.  Currently only testing for
123   ;  serial receive interrupt -- all other interrupts disabled.
124   ;       4-5 cycle delay expected for Hardware to call PERIPH_HAND
125   ;       7 cycles are executed before RCSER_HAND called.
126   ; Calls:
127   ;  RC1_ISR in file Ser_Hand.asm
128   ;
129   ; Registers:
130   ;  STACK_FSR                  ; Shared memory, indirect address register
131   ;  STACK_Status                       ; Shared memory, temporary STATUS contents
132   ;  STACK_Wreg                 ; Shared memory, temporary Working Register contents
133   ;
134   ; Interrupts:
135   ;  Global Interrupts auto-disabled at start, auto-enabled at close
136   ;
137   ; Return:
138   ;  Wreg, STATUS, FSR returned to pre-Interrupt state
139   ;
140   ;*******************************************************************************
141   ;
142   ISR_PROG       CODE                          ;relocatable code in Program EPROM
```

Code Listing – Ser_Hand.asm

```
143   ;
144   PERIPH_ISR
145   ; Note: STACK_Wreg, _Status, and _FSR are in shared memory, so are available
146   ; regardless of the current Memory Bank pointers RP0 and RP1
147           Movwf    STACK_Wreg            ; save Wreg first or will overwrite
148           Movf     STATUS, w             ; next, save Status register
149           Movwf    STACK_Status          ;   into reserved location
150           Movf     FSR, w                ; finally, save the FSR
151           Movwf    STACK_FSR
152           Bcf      STATUS, RP0           ; Select Bank 0 memory
153   ;
154   ; Transition on RB4, Port C to note entry into ISR.
155           Movlw    0x10                  ; ISR_TEST is PORT B, line 4
156           Xorwf    PORTB, f              ; Change RB4 state
157   ;
158   ; ----- Serial Interrupt Test (for some reason, RCIF not being set)
159   ISR_RC1TEST
160           Btfss    PIR1, RCIF            ; Test Rc Interrupt Flag
161           Goto     SER_Cntl_ERR          ;  . . . not receive, clear error conditions
162   ;
163   ; Even at 115.2k Baud, almost 320 instructions can be executed in the time to
164   ; transmit one Byte, so something is wrong if a serial overrun has occurred.
165   ISR_OERR_TST
166           Btfsc    RCSTA, OERR           ; Test if serial register overrun
167           Goto     SER_Cntl_ERR          ;  . . . overrun error, process
168   ;
169   ; Framing Error Test
170   ISR_FERR_TST
171           Btfsc    RCSTA, FERR           ; Test if framing error occurred
172           Goto     SER_Cntl_ERR          ;  . . . framing error, process
173   ;
174   ; Begin processing serial interrupt
175   ;
176   ; RCIF is a read-only bit cleared when RCREG has been read and is empty. RCREG
177   ; is a double-buffered register, i.e. it is a two deep FIFO. It is possible for
178   ; two bytes of data to be received and transferred to the RCREG FIFO and a
179   ; third byte begin shifting to the RSR. On detection of the stop bit of the
180   ; third byte, if the RCREG is still full, then the overrun error bit,
181   ; OERR (RCSTA<1>) will be set. The word in the RSR will be lost. RCREG can be
182   ; read twice to retrieve the two bytes in the FIFO. The OERR bit is cleared by
183   ; resetting the receive logic (CREN is set). If the OERR bit is set, transfers
184   ; from the RSR to RCREG are inhibited, so it is essential to clear the OERR bit
185   ; if it is set. The framing error bit FERR (RCSTA<2>) is set if a stop bit is
186   ; not detected.
187   ;
188   ISR_RECEIVE
189           Movf     RCREG, w              ; capture character from USART receive buffer
190           Movwf    RC_CHAR               ;    and put into working location
191           Movlw    0x10                  ; ISR_TEST is PORT B, line 4 of 0 - 7
192           Xorwf    PORTB, f              ; make ISR_TEST change state
193   ;
194   ; Handle the character received
195   ; Check for special control characters
196   ISR_SP_CHAR
197           Movf     RC_CHAR, w
198           Sublw    0x1F                  ; subtract control char range
199           Btfss    STATUS, C             ; Check if Carry set
200           Goto     SER_STR               ; Carry clear, so not a control character
201   ;
202   ;*******************************************************************************
203   ;************** Start of Control character processing section *****************
204   ;*******************************************************************************
205   ;
206   ; Now, set up to jump to control character routine.
207           Movlw    high    SER_Jump      ; Get upper Program Counter
208           Movwf    PCLATH                ; set in upper 5 bits of PC
209           Movf     RC_CHAR, w            ; recover received character
210           Addlw    SER_Jump              ; Adds GOTO series starting address
211           Btfsc    STATUS, C             ; Check if address addition overflow
212           Incf     PCLATH, f             ; Yes, adjust program counter high byte
213           Movwf    PCL                   ; change Program Counter to jump there
```

```
214  ;
215  SER_Jump
216         Goto    SER_END                 ; Null, ignore
217         Goto    SER_Cntl_ERR            ; 0x01 not used, SOH
218         Goto    SER_Cntl_ERR            ; 0x02 not used, STX
219         Goto    SER_Cntl_C              ; 0x03 is Control-C
220         Goto    SER_Cntl_ERR            ; 0x04 not used, EOT
221         Goto    SER_Cntl_ERR            ; 0x05 not used, ENQ
222         Goto    SER_ACK                 ; 0x06 is ACK, received data
223         Goto    SER_Cntl_ERR            ; 0x07 not used, BEL
224         Goto    SER_Cntl_ERR            ; 0x08 not used, Backspace
225         Goto    SER_Cntl_ERR            ; 0x09 not used, Horizontal Tab
226         Goto    SER_LF                  ; 0x0A is Line Feed
227         Goto    SER_Cntl_ERR            ; 0x0B not used, Vertical Tab
228         Goto    SER_Cntl_ERR            ; 0x0C not used, Form Feed
229         Goto    SER_CR                  ; 0x0D is Carriage Return
230         Goto    SER_Cntl_ERR            ; 0x0E not used, SO
231         Goto    SER_Cntl_ERR            ; 0x0F not used, SI
232         Goto    SER_Cntl_ERR            ; 0x10 not used, SLE
233         Goto    SER_Xon                 ; 0x11 is X-on, DC1
234         Goto    SER_Cntl_ERR            ; 0x12 not used, DC2
235         Goto    SER_Xoff                ; 0x13 is X-off, DC3
236         Goto    SER_Cntl_ERR            ; 0x14 not used, DC4
237         Goto    SER_NAK                 ; 0x15 is NAK, retransmit
238         Goto    SER_Cntl_ERR            ; 0x16 not used, SYN
239         Goto    SER_Cntl_ERR            ; 0x17 not used, ETB
240         Goto    SER_Cntl_ERR            ; 0x18 not used, CAN
241         Goto    SER_Cntl_ERR            ; 0x19 not used, EM
242         Goto    SER_Cntl_ERR            ; 0x1A not used, STB
243         Goto    SER_Cntl_ERR            ; 0x1B not used, ESC
244         Goto    SER_Cntl_ERR            ; 0x1C not used, FS
245         Goto    SER_Cntl_ERR            ; 0x1D not used, GS
246         Goto    SER_Cntl_ERR            ; 0x1E not used, RS
247         Goto    SER_Cntl_ERR            ; 0x1F not used, US
248  ;
249  ; Control-C routine, stops any command phrase in progress without error message
250  SER_Cntl_C
251         Bsf     SER_FLAG, CNTRLC_FLG    ;  ==CNTL-C, set CNTLC Flag
252         Clrf    SER_STATE               ; clear command phrase state
253         Clrf    RC_CHAR                 ; clear the received character buffer
254         Clrf    CMD_CHAR                ; clear any command character
255         Clrf    CMD_PARAMS              ; Clear parameter count
256         Goto    SER_END
257  ;
258  ; Line Feed routine.  Either LF or CR can be first, but both must occur to terminate command.
259  SER_LF
260         Bsf     SER_STATE, SER_ST_LF    ; Bit 4 of SER_STATE
261         Goto    SER_CHECK_DONE          ; If both CR and LF received, parse parameters
262  ;
263  ; Carriage Return routine.  Either CR or LF can be first, but both must occur to terminate
264  command.
265  SER_CR
266         Bsf     SER_STATE, SER_ST_CR    ; Right sequence, Bit of SER_STATE set
267         Goto    SER_CHECK_DONE          ; If both CR and LF received, parse parameters
268  ;
269  ; X-Off routine.  Sets flag to halt serial transmission in TX_WREG routine.
270  ; Xoff is an inhibit control, while Xon both clears Xoff and signals go-ahead
271  ; to begin data upload.
272  SER_Xoff
273         Bsf     SER_FLAG, XOFF_FLG      ; Set the X-Off Command Flag
274         Bcf     SER_FLAG, XON_FLG       ;   . . and clear the X-On Command Flag
275         Goto    SER_END
276  ;
277  ; X-On routine.  Clears Xoff flag to enable serial transmission in the TX_WREG
278  ; routine. Xoff is an inhibit-only control, while Xon both clears Xoff and
279  ; signals go-ahead to begin data upload in the ST_HAND and RD_HAND routines.
280  SER_Xon
281         Bcf     SER_FLAG, XOFF_FLG      ; Clear the X-Off Command Flag
282         Bsf     SER_FLAG, XON_FLG       ;   . . and set the X-On Command Flag
283         Goto    SER_END
284  ;
```

```
285  ; ACK routine.  Sets flag indicating PC received data block with proper
286  ; checksum. The flag is cleared by the routine monitoring the flag.
287  SER_ACK
288         Bsf     SER_FLAG, ACK_FLG       ; Set Acknowledge flag, continue xmit
289         Goto    SER_END
290  ;
291  ; NAK routine.  Sets flag that data ckecksum in error, need to retransmit.
292  ; The flag is cleared by the routine monitoring the flag.
293  SER_NAK
294         Bsf     SER_FLAG, NAK_FLG       ; Set Negative Ack, retransmit data
295         Goto    SER_END
296  ;
297  ; None of the characters above, or other error.  Set flag for debug and ignore.
298  SER_Cntl_ERR                           ; just illegal control character
299         Bsf     SER_FLAG, CNTL_ER_FLG   ; Note error for debug
300         Bcf     RCSTA, CREN             ; Reset receive enable to clear OERR
301          Movf   RCREG, w                ; Clear Framing error by reading RCREG
302          Movf   RCREG, w                ; Buffer is two registers deep
303         Bsf     RCSTA, CREN             ; Re-enable receive
304         Goto    SER_END
305  ;
306  ;*******************************************************************************
307  ;**** End of Control character processing section, start of Command phrase ****
308  ;*******************************************************************************
309  ;
310  ; A command sequence consists of a zero, then a capital letter, optional
311  ; parameters each with a value between 0x20 and 0xFF, and ends with carriage
312  ; return and line feed.  The command is not processed until the CR LF.
313  ;
314  ; Check to see if sequence has started
315  SER_STR
316         Btfsc   PROG_FLAG, COMMAND_FLG ; Set if already processing a command phrase
317         Goto    SER_PHRASE_DONE              ; Ignore new commands until current is done
318         Btfss   SER_STATE, SER_ST_0    ; See if command start received
319         Goto    SER_ZERO               ; Start not received, make sure is zero
320         Btfss   SER_STATE, SER_ST_AZ   ; Test if command character previously read
321         Goto    SER_COMMAND            ; Check if character in range "A" - "z"
322  ;
323  ; Everything else received, so must be a Parameter.  Push Parameter characters
324  ; as received onto parameter stack for later processing.  Need to build stack
325  ; pointer because this is an interrupt routine, and FSR changes on exit.
326  SER_PARAMETER
327         Movlw   Param_Stack_Size       ; Prepare to check if exceeded stack size
328         Subwf   CMD_PARAMS, w          ; Test against number received so far
329         Btfsc   STATUS, C              ; Carry set means # params. >= limit
330         Goto    SER_ERR                ; Parameter Stack overflow
331  ;
332         Incf    CMD_PARAMS, f          ; Count parameters received
333         Movlw   CMD_Param_Start
334         Addwf   CMD_PARAMS, w          ; Add Parameter stack offset location
335         Movwf   FSR                    ; Stack points to next location
336         Movf    RC_CHAR, w             ; get received character
337         Movwf   INDF                   ; store on parameter stack
338         Bsf     SER_STATE, SER_ST_PAR  ; At least one parameter stored
339         Goto    SER_END
340  ;
341  ; No command phrase has started, so to be valid the first character must be a zero.
342  ; If the character is not zero, clear the character and exit.
343  SER_ZERO
344         Clrf    SER_STATE              ; Must be first character, reset state
345         Movlw   "0"                    ; will compare with zero character
346         Subwf   RC_CHAR, w             ;
347         Btfss   STATUS, Z              ; If RC_CHAR equal to "0", Zero flag set
348         Goto    SER_ERR                ;  !=0, so we're outta here.
349         Bsf     SER_STATE, SER_ST_0    ; mark that the starting zero has been received.
350         Goto    SER_END
351  ;
352  ; Expecting to see a character in "A" thru "z".  If not a character, then error out.
353  SER_COMMAND
354         Movlw   "z"+1                  ; check if command character beyond "z"
355         Subwf   RC_CHAR, w             ; Subwf subtracts "z"+1 from character
```

```
356         Btfsc   STATUS, C               ; carry clear means character <= "z"
357         Goto    SER_ERR                 ; above "z", error
358         Movlw   "A"                     ; now test lower end
359         Subwf   RC_CHAR, w              ; will be zero or positive if valid
360         Btfss   STATUS, C               ; carry set means character >= "A"
361         Goto    SER_ERR                 ; less than "A", error
362         Bsf     SER_STATE, SER_ST_AZ    ; Valid character - not tested if command
363 ;
364 ; Now test if this is the Verify command
365         Movlw   "V"                     ; Check to see if command is Verify = V
366         Subwf   RC_CHAR, w
367         Btfsc   STATUS, Z               ; will be zero if Verify character received
368         Goto    SER_VER_CMD             ; Is "V", process Verify character
369 ;
370 ; Save command character in temporary location and prepare to receive Parameters
371         Movf    RC_CHAR, w              ; Recover character received
372         Movwf   RC_TEMP                 ; Hold in temporary
373         Clrf    CMD_PARAMS              ; Clear received Parameter count
374         Goto    SER_END
375 ;
376 ; Is Verify, set Verify flag but don't overwrite temporary command character
377 SER_VER_CMD
378         Bsf     SER_STATE, SER_VERIFY   ; Received Verify character
379         Bcf     SER_FLAG, XON_FLG       ; Clear Xon flag - prepare for upload
380         Goto    SER_END
381 ;
382 ;*******************************************************************************
383 ;**** End of Command phrase processing section, start of Parameter parse ******
384 ;*******************************************************************************
385 ;
386 ; Test to see if phrase is complete (minimum "0", command, CR, LF)
387 SER_CHECK_DONE
388         Movf    SER_STATE, w
389         Andlw   0x0F                    ; Strip off Parameter, Verify flag bits
390         Sublw   0x0F                    ; Test if entire command phrase done
391         Btfss   STATUS, Z
392         Goto    SER_END                 ; not complete phrase
393         Btfss   SER_STATE, SER_VERIFY   ; see if this is Verify phrase
394         Goto    SER_PHRASE_DONE                 ; Not Verify, skip finish-up for now
395 ;
396 ; Activity from PC, reset command interface timeout count.  Otherwise, may
397 ; get a "heartbeat" dot character in echo, and cause interface error
398         Clrf    CMD_TIMEL               ; Clear time-out registers
399         Clrf    CMD_TIMEH
400 ;
401 ; Received Verify phrase, so finish up command setup.
402         Movf    RC_TEMP, w              ; recover temporary Command character
403         Movwf   CMD_CHAR                ; store Command character for CMD_HAND use
404         Bsf     PROG_FLAG, COMMAND_FLG ; Mark command ready for processing
405 ;
406 ; Build parameter bytes from nibbles transmitted.  The PC cannot send parameters
407 ; with values less than 0x20 because they would be interpreted as control
408 ; characters. Therefore, only the lower nibble of each parameter byte is used.
409 ; The assumed sequence is Byte 1, Byte2 => MS Nibble, LS Nibble.  If the last
410 ; Byte is odd-numbered, it is translated as a MS Nibble.
411         Movlw   CMD_Param_Start                 ; Get start of parameter table
412         Movwf   FSR                     ; Place in stack pointer
413         Movf    INDF, w                 ; =CMD_PARAMS, count of parameters passed
414         Btfsc   STATUS, Z               ; Test if zero parameters on stack
415         Goto    SER_PHRASE_DONE                 ; No parameters passed
416         Movwf   LOOP_INT                ; Interrupt use only Loop Counter
417 ;
418 ; Rebuild parameter table with combined nibbles
419 SER_PAR_EXTR
420         Incf    FSR, f                  ; Point to first (odd) Parameter
421         Movlw   0x0F                    ; Set mask to strip upper nibble
422         Andwf   INDF, f                 ; Strip upper nibble
423         Swapf   INDF, f                 ; Make this MS nibble of Parameter
424         Decf    LOOP_INT, f             ; Track Parameters processed
425         Btfsc   STATUS, Z               ; See if was last Parameter
426         Goto    SER_PAR_DONE            ; Done with parameters
```

```
427   ;
428   ; Process LS Nibble of combined parameter
429         Incf    FSR, f                  ; Point to second (even) Parameter
430         Andwf   INDF, f                 ; Strip upper nibble
431         Movf    INDF, w                 ; Recover stripped value
432         Decf    FSR, f                  ; point to upper nibble
433         Addwf   INDF, f                 ; Combine lower with upper nibble
434         Decf    LOOP_INT, f             ; Find Parameters remaining
435         Btfsc   STATUS, Z               ; See if was last Parameter
436         Goto    SER_PAR_DONE            ; Done with parameters
437   ;
438   ; Move up rest of parameter stack to fill void
439         Movf    FSR, w                  ; Get adjusted FSR
440         Movwf   TEMP_INT                ; Hold this value
441         Incf    FSR, f                  ; Points to what will be next value
442         Movf    LOOP_INT, w             ; Get Parameters remaining
443         Movwf   RC_CHAR                 ; Use as temporary loop counter
444   ;
445   ; Loop through stack, sliding all values up by one location
446   SER_PAR_LOOP
447         Incf    FSR, f                  ; Point to next unprocessed parameter
448         Movf    INDF, w                 ; Get this value
449         Decf    FSR, f                  ; Point to new location
450         Movwf   INDF                    ; Put the value there
451         Incf    FSR, f                  ; Point to just-moved parameter
452         Decfsz  RC_CHAR, f              ; Loop counter
453         Goto    SER_PAR_LOOP            ; Still more parameters in stack
454   ;
455   ; Done moving up Stack, restore pointer
456         Movf    TEMP_INT, w             ; Get the FSR held before
457         Movwf   FSR
458         Goto    SER_PAR_EXTR            ; Loop back to process next
459   ;
460   ; Adjust parameter count to be half, 1&2 => 1, 3&4 => 2, etc.
461   SER_PAR_DONE
462         Incf    CMD_PARAMS, f           ; Prepare param count for divide
463         Bcf     STATUS, C               ; Clear carry bit
464         Rrf     CMD_PARAMS, f           ; Effectively, divide by two
465         Goto    SER_PHRASE_DONE                 ; close out processed phrase
466   ;
467   ;
468   ;*******************************************************************************
469   ;**** Finished with serial routine, close out in error or normal mode *********
470   ;*******************************************************************************
471   ;
472   ; Problem in command phrase.  Reset all variables.
473   SER_ERR
474         Clrf    CMD_CHAR                ; clear any command character
475         Clrf    CMD_PARAMS              ; Clear parameter count
476         Bsf     SER_FLAG, STR_ER_FLG    ; Note error for debug
477   ;
478   ; Either Phrase complete or error.  Echo the command character or "?" if
479   ; a command phrase error.  In a normal command sequence, echo is sent twice,
480   ; once when the Command character phrase is complete, and once when the "V"
481   ; verify character is sent.  In the former, unprocessed Parameters (characters)
482   ; are sent. In the latter, processed parameters in final form are sent.
483   ;
484   SER_PHRASE_DONE
485   ;       Transmits   | "0" | Temp Cmd Char | <parameters> | CR | LF |
486         Clrwdt                          ; Watch-Dog Timer reset
487         Movlw   "0"                     ; Load zero character
488         Call    TX_WREG                 ; Transmit zero character and return
489         Btfss   SER_FLAG, STR_ER_FLG    ; Check if command string error
490         Goto    SER_ECHO_CHAR           ; No error, just echo command character
491   ;
492   ; Was an error, insert error character before one received
493         Movlw   "?"                     ;  Yes, send error character flag
494         Call    TX_WREG                 ; Transmit error character and return
495   SER_ECHO_CHAR
496         Movf    RC_TEMP, w              ; get the temporary command character
497         Call    TX_WREG                 ; Transmit command character
```

```
498   ;
499   ; Check to see if any parameters were sent.  If so, transmit the parameters in
500   ; processed form (nibbles combined to form bytes).
501          Movlw   CMD_Param_Start              ; Get start of parameter table
502          Movwf   FSR                  ; Place in stack pointer
503          Movf    INDF, w              ; =CMD_PARAMS, count of parameters passed
504          Btfsc   STATUS, Z            ; Test if zero parameters on stack
505          Goto    SER_ECHO_END         ; No parameters passed
506          Movwf   LOOP_INT             ; Hold number of parameters passed
507   ;
508   ; Send parameter values
509   SER_ECH_Loop
510          Incf    FSR, f               ; Point to Parameter
511          Movf    INDF, w              ; Recover value
512          Call    TX_WREG              ; Transmit Parameter
513          Decfsz  LOOP_INT, f          ; See if all parameters sent
514          Goto    SER_ECH_Loop         ; . . . Continue sending
515   ;
516   ; Done with echoed command and parameters, close out string
517   SER_ECHO_END
518          Movlw   "\r"                 ; Carriage Return
519          Call    TX_WREG              ; Transmit
520          Movlw   "\n"                 ; Line Feed
521          Call    TX_WREG              ; Transmit
522   ;
523          Clrf    SER_FLAG             ; Clear all serial flags
524          Clrf    SER_STATE            ; clear command phrase state
525   ;
526   ;*******************************************************************************
527   ;*******************************************************************************
528   ;
529   ; Exit Interrupt Service Routine
530   SER_END
531          Movf    STACK_FSR, w         ; Retrieve pre-isr FSR contents
532          Movwf   FSR                  ;   and restore the FSR
533          Movf    STACK_Status, w              ; Retrieve pre-isr STATUS contents
534          Movwf   STATUS               ;   and restore Microprocessor state
535          Movf    STACK_Wreg, w        ; Finally, restore Wreg
536   ;
537   ISR_END
538          Retfie                       ;Return from Interrupt
539   ;                                   ; auto-enable global interrupts
540   ;
541   ;*******************************************************************************
542   ;*******************************************************************************
543   ;
544   ;       TX_WREG and TX_HEX Subroutines
545   ;
546   ; Function:
547   ;  Expects serial port initialized (in Ball.asm).  The data byte to be
548   ;  transmitted is passed to this routine in Wreg.  Waits for the transmit buffer
549   ;  to empty if necessary, then move Wreg to TXREG and sets TXEN to transmit.
550   ;
551   ;  Two registers are involved in transmitting data.  Data are first moved to the
552   ;  transmit buffer, TXREG.  After the Stop bit is transmitted from the previous
553   ;  load, the transmit (serial) shift register (TSR) loads the new data from the
554   ;  TXREG register.  Once the TXREG register transfers the data to the TSR
555   ;  register, the TXREG register is empty and flag bit TXIF (PIR1<4>) is set.
556   ;  Flag bit TXIF will reset only when new data is loaded into the TXREG
557   ;  register.  Bit TRMT (TXSTA<1>) is a read-only bit that is set when the TSR
558   ;  register is empty.
559   ;
560   ; WARNING: You must not trust the TXIF bit to accurately reflect the state of
561   ; the transmitter on power up. (The flag shows FULL even though there are NO
562   ; characters in either the TXREG or TSR! The flag bit TXIF is ONLY true AFTER
563   ; the first DUMMY character's STOP bit is clocked out of the TSR producing the
564   ; FIRST load pulse to the TXREG and setting the correct flags!)
565   ;
566   ;  115.2kBaud * (8/10) = effective bits/second at 115.2kBaud = 92160
567   ;  This equates to 11520, 8-bit Bytes/second, or 0.2 hours to upload 8MBytes
568   ;  At 19.2kBaud, 1.2 hours are required to upload 8MBytes.
```

```
569   ;
570   ; Calls:
571   ;       NONE
572   ;
573   ; Registers:
574   ;  Wreg has byte to transmit on entry
575   ;
576   ; Interrupts:
577   ;  Interrupts not used or changed in this subroutine
578   ;
579   ; Return:
580   ;  Wreg still holds data transmitted on exit
581   ;
582   ;*******************************************************************************
583   ;
584   ; Converts lower nibble of Wreg to ASCII representation.  Character is left in
585   ; Wreg to be processed by TX_WREG.
586   TX_HEX
587           Andlw   0x0F                    ; Mask off upper nibble
588           Addlw   0xF6                    ; Test if greater than 9
589           Btfsc   STATUS, C               ; If so, set for A - F
590           Addlw   0x07                    ; is in A - F, add offset
591           Addlw   0x3A                    ; ASCII offset for zero + restore subtract
592   ;
593   TX_WREG
594   ;
595   ; First, check if PC requested transmission halt: Xoff flag set.  This flag is
596   ; cleared by the PC when an Xon character is sent.
597           Bcf     STATUS, RP0             ; Select Bank 0 memory
598           Btfsc   SER_FLAG, XOFF_FLG
599           Goto    TX_WREG                 ; X-off sent, wait until X-on clears
600   ;
601   ; Xoff not set, next check if transmit shift register is empty
602           Bsf     STATUS, RP0             ; Select Bank 1 memory
603           Btfsc   TXSTA, TRMT             ; Set to 1 when TSR empty
604           Goto    TX_DO_IT                ; shift register empty, load new data
605   ;
606   ; If the TSR is not empty, might not need to wait.  Just check if TXREG is empty
607   ; Note: this bit is only set after the first transmit
608           Bcf     STATUS, RP0             ; Select Bank 0 memory
609           Btfss   PIR1, TXIF              ; Check TX Register, 1=empty
610           Goto    TX_WREG                 ;   need to wait to empty if not.
611   ;
612   ; Choosing to load TXREG, then start transmission by enabling transmit.  Data
613   ; book says this is faster than leaving enabled and transmit on load.
614   TX_DO_IT
615           Bcf     STATUS, RP0             ; Select Bank 0 memory
616           Movwf   TXREG                   ; Load Tx Register from Wreg
617           Bsf     STATUS, RP0             ; Select Bank 1 memory
618           Bsf     TXSTA,TXEN              ; Start Transmission
619           Bcf     STATUS, RP0             ; Select Bank 0 memory
620           Return                          ; Return with data still in Wreg
621   ;
622   ;*******************************************************************************
623   ;*******************************************************************************
624   ;       TX_String       Subroutine
625   ;
626   ; Function:
627   ;  Uses TX_WREG to send a string of characters to the serial port.  The string
628   ;  location is loaded into the stack pointer.  The string is transmitted one
629   ;  character at a time until a null (hex 0) is found.  The null is transmitted
630   ;  as well.
631   ;
632   ; Calls:
633   ;       TX_WREG
634   ;
635   ; Registers:
636   ;  Look_Hi, Look_Low contain start of string on entry.  These are modified as
637   ;       each value in the string is recalled.
638   ;  Wreg is used to send the byte to TX_WREG
639   ;
```

```
640   ; Interrupts:
641   ;  Interrupts not used or changed in this subroutine
642   ;
643   ; Return:
644   ;  Wreg holds last data transmitted on exit (null)
645   ;
646   ;*******************************************************************************
647   ;
648   ;
649   TX_String
650          Call    TX_Lookup              ; Get character to transmit
651          Call    TX_WREG                ; Send character, returns Wreg=character
652          Andlw   0xFF                   ; Refresh STATUS register on Wreg
653          Btfss   STATUS, Z              ; Check if just sent null
654          Goto    TX_String              ; Wasn't null, loop again
655   ;
656   ; Done with transmit
657   TX_Str_End
658          Retlw   0                      ; Return, WREG=0
659   ;
660   ; This section recalls the character from program data, where it is stored as
661   ; a the return value in a Retlw expression.  Jumps program counter to the
662   ; desired Retlw, and completes the call return from there.
663   TX_Lookup
664   ;
665   ; Set Program counter to location of string character
666          Movf    Look_Hi, w             ; set PCLATH to next string location
667          Movwf   PCLATH
668          Movf    Look_Lo, w             ; Now set lower program counter, PCL
669   ;
670   ; Now adjust the Program Counter for the next access (if used)
671          Incf    Look_Lo, f             ; Adjust to next location for subsequent access
672          Btfsc   STATUS, Z              ; see if incremented to zero
673          Incf    Look_Hi, f             ; lower address incremented, adjust upper
674   ;
675   ; Now jump to address: an Retlw command returning Wreg with the string value
676          Movwf   PCL                    ; ok, now jump
677   ; Note: since this is a called routine, the RETLW command will reset the program
678   ; counter to the code that called TX_Lookup.
679   ;
680   ;***************************** End of TX_String ******************************
681   ;*****************************************************************************
682   ;*****************************************************************************
683   ;       Fail_Msg, Status_Msg, Erase_Msg, SENDDATA Subroutines
684   ; Function:
685   ;       Transmits <Message> (memory address) "Mem Flag Value" (memory flag) CR LF
686   ;       <Messages> are:
687   ;       Fail_Msg : "Failed block"
688   ;       Status_Msg: "Finished block"  -- used for Fill and Test functions
689   ;       Erase_Msg: "Erased block"
690   ;       SENDDATA: | Sync0 | Sync1 | 0x00 | 0x02 | Version | Unit # |
691   ;
692   ; Calls:
693   ;       TX_WREG, TX_HEX, TX_String
694   ;
695   ; Registers:
696   ;       MEMORY_FLAG, ADD2, ADD1
697   ;
698   ; Interrupts:
699   ;  None changed.
700   ;
701   ; Return:
702   ;       Wreg=0
703   ;
704   ;*******************************************************************************
705   Fail_Msg
706          Movlw   High    List_Fail_Msg  ; Load starting location of string
707          Movwf   Look_Hi
708          Movlw   Low     List_Fail_Msg
709          Movwf   Look_Lo
710          Goto    Msg_Finish
```

```
711  ;
712  Status_Msg
713         Movlw   High    List_Stat_Msg  ; Load starting location of string
714         Movwf   Look_Hi
715         Movlw   Low     List_Stat_Msg
716         Movwf   Look_Lo
717         Goto    Msg_Finish
718  ;
719  Erase_Msg
720         Movlw   High    List_Eras_Msg  ; Load starting location of string
721         Movwf   Look_Hi
722         Movlw   Low     List_Eras_Msg
723         Movwf   Look_Lo
724         Goto    Msg_Finish
725  ;
726  SENDDATA
727         Clrwdt                         ; Watch-Dog Timer reset
728         Movlw   Sync_Byte0             ;Load First Sync Byte
729         Call    TX_WREG                ; Transmit Data
730         Movlw   Sync_Byte1             ;Load Second Sync Byte
731         Call    TX_WREG                ; Transmit Data
732         Movlw   0x00                   ;Load NULL
733         Call    TX_WREG                ; Transmit Data
734         Movlw   0x02                   ;Load ASCII STX
735         Call    TX_WREG                ; Transmit Data
736         Movlw   Hardware_Ver           ;Load Current Hardware Rev #
737         Call    TX_WREG                ; Transmit Data
738         Movlw   Unit_Number            ;Load Unit #
739         Call    TX_WREG                ; Transmit Data
740         Goto    Msg_Addr_Stat
741  ;
742  ; Send String message
743  Msg_Finish
744         Clrwdt                         ; Watch-Dog Timer reset
745         Call    TX_String              ; Call string transmit routine
746  ;
747  ; Send Memory MSB address location and Memory status
748  Msg_Addr_Stat
749         Swapf   ADD2, w                ; Load Memory MS Byte of address
750         Call    TX_HEX                 ; Convert lower nibble to ASCII & TX
751         Movf    ADD2, w
752         Call    TX_HEX                 ; Convert lower nibble to ASCII & TX
753         Swapf   ADD1, w                ; Next significant Flash Memory address
754         Call    TX_HEX                 ; Convert lower nibble to ASCII & TX
755         Movf    ADD1, w
756         Call    TX_HEX                 ; Convert lower nibble to ASCII & TX
757  ;
758  ; Memory error flag
759         Movlw   High    List_Mem_Flag  ; Load starting location of string
760         Movwf   Look_Hi
761         Movlw   Low     List_Mem_Flag
762         Movwf   Look_Lo
763         Call    TX_String              ; Call string transmit routine
764  ;
765         Swapf   MEMORY_FLAG, w         ; Prepare memory error flag MS nibble
766         Call    TX_HEX                 ; Convert lower nibble to ASCII & TX
767         Movf    MEMORY_FLAG, w
768         Call    TX_HEX                 ; Convert lower nibble to ASCII & TX
769  ;
770         Movlw   "\r"                   ; Carriage Return
771         Call    TX_WREG                ; Transmit
772         Movlw   "\n"                   ; Line Feed
773         Call    TX_WREG                ; Transmit
774         Retlw   0                      ; Return, WREG=0
775  ;
776  ;*******************************************************************************
777  ;*******************************************************************************
778  ;       Baud_Set Subroutine
779  ; Function:
780  ;       Initializes serial port and sets Baud rate based upon Wreg contents.
781  ;       A Wreg value of zero sets Baud to 19.2k Baud, otherwise, set to a rate
```

```
;       of 115.2k Baud.
;
; Calls:
;       None
;
; Registers:
;       Wreg, Zero sets Baud to 19.1k Baud
;             non-zero sets Baud to 115.2k Baud
;
; Interrupts:
;  None changed.
;
; Return:
;       Wreg=0
;
;*****************************************************************************
Baud_Set
;
; Universal Synchronous/Asynchronous Receiver/Transmitter (PIC16C77x Data Book,
; sect. 9.0) Configure for either 19.2k Baud (standard Penetrator rate) or high
; speed at 115.2k Baud (the maximum, limited by the Maxim RS-232 chip used in
; the design).  The PIC microcontroller's RC oscillator must be close to
; 3.6864MHz for the Baud rates to match the standard speeds.
; SPBRG and TXSTA Registers in Bank 1 Memory
;
; USART Transmit configuration
; Check which Baud rate.  Wreg value selects rate
        Bsf     STATUS, RP0             ; Select Bank 1 Memory
        Btfsc   STATUS, Z               ; Wreg zero if selecting 19.2k Baud
        Goto    Baud_19                 ; Z set, select 19.2k
;
; Set rate to 115.2k Baud
Baud_115
        Clrf    TXSTA                   ; Reset any error condition
        Movlw   0x01                    ; value for 115.2k Baud @ high speed
        Movwf   SPBRG                   ; Set Baud Rate Hi=Fosc/(16*(SPBRG+1))
        Bsf     TXSTA, BRGH             ; Select high speed
        Goto    Baud_TX
;
; Set rate to 19.2k Baud
Baud_19
        Clrf    TXSTA                   ; Reset any error condition
        Movlw   0x02                    ; value for 19.2k Baud @ low speed
        Movwf   SPBRG                   ; Set Baud Rate Low=Fosc/(64*(SPBRG+1))
        Bcf     TXSTA, BRGH             ; Select low speed
;
; Set rest of transmit configuration
Baud_TX
;       Bcf     TXSTA, SYNC             ; Asynchronous operation
        Bsf     TXSTA, SPEN             ; Serial Port enable - Configure RC7/RX/DT
;                                           and RC6/TX/CK as serial port pins
;       Bcf     TXSTA, TXEN             ; Enable Transmit  -- don't do this here
;       Bcf     TXSTA, TX9              ; 8-bit transmission
;
; USART Receive configuration
Baud_RX
        Bcf     STATUS, RP0             ; Select Bank 0 Memory
        Clrf    RCSTA                   ; Reset any error condition
        Bsf     RCSTA, SPEN             ; Serial port enable
;       Bcf     RCSTA, RX9              ; Selects 8-bit reception
;       Bcf     RCSTA, SREN             ; Don't care - Synchronous configuration
        Bsf     RCSTA, CREN             ; Continuous receive enable
;       Bcf     RCSTA, ADDEN            ; Disable address detection
;
; Interrupt configuration
        Bsf     STATUS, RP0             ; Select Bank 1 Memory
        Bsf     PIE1, RCIE              ;Enable Serial Port Rx Interrupt
        Bcf     STATUS, RP0             ; Select Bank 0 Memory
        Clrf    INTCON                  ; Clear interrupt control register
        Bsf     INTCON, PEIE            ;Enable Peripheral Interrupts
        Bsf     INTCON, GIE             ;Enable Global Interrupts
```

```
853     ;
854     Baud_END
855             Retlw   0                       ; Return with Wreg=0
856     ;
857     ;*******************************************************************************
858     ;;********************** End of FILE Ser_Hand.asm ******************************
859     ;*******************************************************************************
860             END
```

Code Listing – St_Hand.asm

```
1   ; File name: "st_hand.asm"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Status Handler returns Sensor Ball status to PC via serial interface
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       1999 - Adapted from MilliPen.asm for Sensor Ball, TA Rohwer
11  ;       November 2000 - Version 2, TA Rohwer
12  ;       17 March 2001 - Version 3 Changes requested by P&G, ME Partridge
13  ;       12 Aug 2001 - Version 4 Changes, ME Partridge
14  ;         Correct value returned for Records in memory to last memory address
15  ;         Add "Smart PCM Device" status logic for command "s"
16  ;
17  ;*****************************************************************************
18  ;       Status_Hand(ler) Subroutine
19  ; Function:
20  ;  Acquires one set of analog measurements (Channels 0 thru 9 skip 7) and the
21  ;  remaining non-ADC channels and holds it in the Data Stack.  Then, these
22  ;  16-bit data words are uploaded to the PC with least-significant byte first
23  ;  (Intel format).  The Data Stack is used in this routine; it is separate from
24  ;  the Parameter Stack used by the Ser_Hand interrupt routine.
25  ;
26  ;  Two Status formats are supported: the original format expected by the
27  ;  SensorBall PC software (Command S upper-case), and the "Smart PCM Device"
28  ;  status format (Command s lower-case).
29  ;
30  ; Subroutines in this file:
31        GLOBAL  STATUS_HAND
32  ;
33  ; Calls:
34        EXTERN  TX_WREG              ; Ser_Hand.asm, Transmits contents of Wreg
35        EXTERN  WAITXXMS             ; Delays for milliseconds set in Wreg
36        EXTERN  WAIT1MS              ; calls WAITXXMS to create a 1 millisecond delay
37        EXTERN  Acquire              ; Puts all nine ADC measurements in Data stack
38        EXTERN  MEM_FIND             ; Locates the next blank Flash memory
39  ;
40  ; Macros used:
41  ;       none
42  ;
43  ; Registers:
44        EXTERN  Loop_Cnt             ; Generic Loop counter
45        EXTERN  Channel_Cnt          ; Loop counter, analog channels 0 to 9 skip 7
46        EXTERN  MeasSet_Cnt          ; Loop counter, sets of analog meas.
47        EXTERN  PROG_FLAG            ; bits COMMAND,ADC_AVG,ACQUIRE_FLG,REINIT,CMDERROR
48        EXTERN  SER_FLAG             ; Program Command Flags
49        EXTERN  MEMORY_FLAG          ; Memory condition and test result flags, MEM_FULL
50        EXTERN  MEM_REC_NUM          ; Number of Records in Flash memory (or in progress)
51        EXTERN  MEM_FRM_NUM          ; Current Frame number in Record
52        EXTERN  ADD0                 ; Flash Memory address A7 .. A0
53        EXTERN  ADD1                 ; Flash Memory address A16 .. A8
54        EXTERN  ADD2                 ; Flash Memory address A24 .. A17
55        EXTERN  LSBYTE               ; ADC_READ, Least Sig. Byte.  [A7,...A0]
56        EXTERN  MSBYTE               ; ADC_READ, Most Sig. Byte.  [X,X,X,X,A11,...A8]
57        EXTERN  TIME0                ; LS Byte, Time in milliseconds (Counter for Seconds)
58        EXTERN  TIME1                ;   . . . next significant byte (Minutes)
59        EXTERN  TIME2                ;   . . . next significant byte (Hours)
60        EXTERN  TIME3                ;   . . . next significant byte (Count of days from Reset)
61  ;
62  ;
63  ; Interrupts:
64  ;  Serial Interrupts not used or changed in this subroutine.  However, the
65  ;  serial port is used for both Rc and Tx via Flags.
66  ;
67  ; Return:
68  ;  Wreg=0
69  ;
70  ;  include files:
71        #include "P16C774.INC"       ; Standard Header File for PIC16C773
```

Code Listing – St_Hand.asm

```
                                        ; Includes all Register Definitions,
                                        ; RAM Definitions, & Configuration Bits
        #include "ball_equ.inc"                 ; EQU Declarations, equivalence
;
;*******************************************************************************
;*******************************************************************************
STAT_PROG       CODE                    ; Relocatable code in Program EPR0M
;
STATUS_HAND
        Clrwdt                          ; Reset 2.4 second Watch-dog Timer
        Bcf     STATUS, RP0             ; uP Memory Bank 0
;
        Clrf    MEM_FRM_NUM             ; Count of Frames transferred
        Clrf    SER_FLAG                ; Clear Control character Flags (Xon)
        Clrf    ADD1                    ; Clear memory location to ensure
        Clrf    ADD2                    ; . . . current one found by MEM_FIND
;
; Note: the next blank address is reported for Record count.  This equals the
; Record count unless partial Records have been made by interrupting the data
; collection cycle by pressing the "ATTENTION" button.
        Call    MEM_FIND                ; Find next blank address
;
; Acquire one measurement set.  If data are acquired while the serial lines are
; active, the measurements seem to be affected.  So the acquire is kept
; separate from the upload function.  These are PIC16C774 ADC 0 thru 9 skip 7,
; so effectively Channels 0 thru 8.
        Bsf     PROG_FLAG, ADC_AVG      ; Set for 16-bit averaging
        Call    Acquire                 ; Get all measurements in Data stack
;
; Put remaining channels on Stack.  Acquire left FSR at next stack location.
; Ch. 9 = Record & Frame number
        Movf    MEM_FRM_NUM, w          ; Get current frame count, 0 to 255
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
        Movf    MEM_REC_NUM, w          ; Record number, 1 to 255
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
;
; Write the TIMEn registers, MS Word, LS Word, with LS Byte first (Intel fmt)
; Ch. 10 = Time MSW
        Movf    TIME2, w                ; Write 4 Bytes of time information
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
        Movf    TIME3, w                ; Bytes currently incremented w/o RTC
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
; Ch. 11 = Time LSW
        Movf    TIME0, w
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
        Movf    TIME1, w
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
;
; Write the Flash memory address with LS Byte first (Intel fmt)
; Ch. 12 = Memory Address
        Movf    ADD1, w
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
        Movf    ADD2, w
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
;
; Write Self-test bits (Memory status Flag).
        Movf    MEMORY_FLAG, w          ; Get memory flag
        Movwf   INDF                    ; Push onto the stack
        Incf    FSR, f                  ; adjust stack pointer
        Clrf    INDF                    ; put zero on MSB location
        Incf    FSR, f                  ; adjust stack pointer
;
; Reset WDT to maximize time waiting for X-on before uP resets
```

```
143         Clrwdt                          ; Reset 2.4 second Watch-dog Timer
144     ;
145     ; Wait for OK from PC
146     STA_XON
147         Btfss   SER_FLAG, XON_FLG       ; PC sends Xon when ready
148         Goto    STA_XON
149         Clrf    SER_FLAG                ; Clear Xon flag & other Control Flags
150     ;
151     ; Test if "Smart PCM Device" Status command "s"
152         Btfsc   PROG_FLAG, SIMPLE_FLG   ; Set if Command "s"
153         Goto    STA_SYNC                ; Skip channel info
154     ;
155     ; PC expecting number of data channels and housekeeping monitor channels
156         Movlw   NumChan                 ; Load # of Channels into W
157         Call    TX_WREG                 ; Transmit to PC
158         Movlw   NumMon                  ; Load # of Monitors into W
159         Call    TX_WREG
160     ;
161     ; Next, Sync bytes
162     STA_SYNC
163         Movlw   Sync_Byte0              ; Start Frame with sync bytes
164         Call    TX_WREG
165         Movlw   Sync_Byte1
166         Call    TX_WREG
167     ;
168     ; Test if "Smart PCM Device" Status command "s"
169         Btfss   PROG_FLAG, SIMPLE_FLG   ; Set if Command "s"
170         Goto    STA_BLK_CNT             ; No, Standard Command "S", skip over
171     ;
172     ; Insert set-up for transmit loop for "Smart PCM Device" status.  Data sent once.
173         Movlw   0x01                    ; Set for one pass thru loop
174         Movwf   MeasSet_Cnt             ; Universal Status, Xmit only one set
175         Movlw   0x1C                    ; 14 channels * 2 Bytes each = 28
176         Movwf   Loop_Cnt                ; Re-use ADC_Read loop counter
177         Goto    STA_Stack               ; join transmit loop
178     ;
179     ; SensorBall combines the following two bytes as low and hi bytes of a block
180     ; count.  The value expected is zero for status check.
181     STA_BLK_CNT
182         Clrw                            ; Load 0x00 (old MinPen = DPH)
183         Call    TX_WREG                 ; send LS Byte first
184         Clrw                            ; Had counter here of data sets
185         Call    TX_WREG                 ; send MS Byte
186     ;
187     ; Set up loop count for Version 3.  Version 2 SensorBall expected a 256-byte
188     ; data block.  Version 3 uses 560 bytes.  Neither counts the preamble bytes
189     ; above. This loop uploads one 560-byte block of data to the PC, with the first
190     ; 16-bit value equal to the current memory location.  No Pad values are needed.
191     ;
192         Movlw   0x1C                    ; 28 sets of measurements
193         Movwf   MeasSet_Cnt
194     SET_LOOP
195         Movlw   0x12                    ; 9 channels * 2 Bytes each = 18
196         Movwf   Loop_Cnt                ; Re-use ADC_Read loop counter
197         Movf    ADD2, w                 ; LS Byte of current memory location
198         Call    TX_WREG                 ; Transmit W Register
199         Movlw   0x00                    ; MS Byte of memory location
200         Call    TX_WREG                 ; Transmit W Register
201     ;
202     ; Entry point for "Smart PCM Device" Status
203     STA_Stack
204         Movlw   Data_Start              ; top of Data Stack
205         Movwf   FSR                     ;  . . . loaded into the stack pointer
206     Tx_Loop
207         Movf    INDF, w                 ; recover value from stack
208         Call    TX_WREG                 ; Upload
209         Incf    FSR, f                  ; adjust stack pointer
210         Decfsz  Loop_Cnt, f
211         Goto    Tx_Loop
212     ;
213         Decfsz  MeasSet_Cnt, f          ; Decrement Loop Index and Test
```

Code Listing – St_Hand.asm

```
214         Goto    SET_LOOP                ;  !=0, Not end of frame, do another
215     ;
216     STA_EXIT
217         Retlw   0                       ;Return, WREG = 0
218     ;
219     ;*******************************************************************************
220     ;************************ End of File St_Hand.asm ******************************
221     ;*******************************************************************************
222         END
```

Code Listing – Ball_Dat.inc

```
1   ; File name: "ball_dat.inc"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Program memory definition include file in main microcontroller code for the P&G Sensor Ball
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       4/26/99  Modified for MilliPen TA Rohwer
11  ;       30 Mar 2001 - Version 3 Changes requested by P&G, ME Partridge
12  ;           New variables added to support changes
13  ;           ADC variables MSBYTE0 - 9 and LSBYTE0 - 9 deleted
14  ;           Time data registers TIME0 - 6 and others for time functions
15  ;       13 Jun 2001 - Checksum generation and data retransmit, ME Partridge
16  ;       17 Jun 2001 - Minor revisions, ME Partridge
17  ;       12 Aug 2001 - Version 4, add temporary variable for address, ME Partridge
18  ;           Add temporary for interrupt routine
19  ;           Move starting location of parameter stack
20  ;
21  ; Function:
22  ;       Includes all Reserved Register Declarations
23  ;       Only #include once in the main program (ball.asm)
24  ;
25  ;*****************************************************************************
26  ;*****************************************************************************
27  SWVARS          UDATA   0x20            ;Bank0 General Purpose Data RAM
28                                          ;  Addresses 0x20 to 0x4F for variables
29  ;
30  ; Flag registers for program control and status
31  PROG_FLAG       RES     1       ;20     Program Command Flags
32  SER_FLAG        RES     1       ;21     Control character received flags
33  MEMORY_FLAG     RES     1       ;22     MEM_HAND        Memory function failure flags
34  SER_STATE       RES     1       ;23     SER_HAND        State machine for command receive
35  ;
36  ; Flash memory position data
37  MEM_REC_NUM     RES     1       ;24     Number of Records in Flash memory (or in progress)
38  MEM_FRM_NUM     RES     1       ;25     Current Frame number in Record
39  MEM_BAD_NUM     RES     1       ;26     Bad blocks counted in Flash memory
40  ADD0            RES     1       ;27     Flash Memory address A7 .. A0
41  ADD1            RES     1       ;28     Flash Memory address A16 .. A8
42  ADD2            RES     1       ;29     Flash Memory address A24 .. A17
43  A8              RES     1       ;2A     Indicates if in first (0) or second (1) half of Page
44  ;
45  ; Serial input support for commands and parameters
46  CMD_CHAR        RES     1       ;2B     SER_ & CMD_HAND, Valid New Command
47  RC_CHAR         RES     1       ;2C     SER_HAND, New Serial Character
48  RC_TEMP         RES     1       ;2D     SER_HAND, Received Character
49  ;
50  ; Serial output string location registers
51  Look_Hi         RES     1       ;2E
52  Look_Lo         RES     1       ;2F
53  ;
54  ; Analog-to-Digital results
55  LSBYTE          RES     1       ;30     ADC_READ, Least Sig. Byte.  [A7,...A0]
56  MSBYTE          RES     1       ;31     ADC_READ, Most Sig. Byte.  [X,X,X,X,A11,...A8]
57  ;
58  ; Time count registers (Placeholder for Real-Time Clock)
59  TIME0           RES     1       ;32     LS Byte, Time in milliseconds (Counter for Seconds)
60  TIME1           RES     1       ;33       . . . next significant byte (Minutes)
61  TIME2           RES     1       ;34       . . . next significant byte (Hours)
62  TIME3           RES     1       ;35       . . . next significant byte (Count of days from Reset)
63  TIME4           RES     1       ;36     =zero, not used yet (Month)
64  TIME5           RES     1       ;37     =zero, not used yet (Year)
65  ;
66  ; Attention mode time-out registers
67  CMD_TIMEL       RES     1       ;38     Time-out LSByte, 285ms / bit
68  CMD_TIMEH       RES     1       ;39     Time-out MSByte
69  ;
70  ; Temporary values used when reading back files
71  R_Sync_Byte0    RES     1       ;3A     Variable for reading data
```

Code Listing – Ball_Dat.inc

```
72    R_Sync_Byte1    RES     1       ;3B
73    R_MEM_REC_NUM   RES     1       ;3C
74    R_MEM_FRM_NUM   RES     1       ;3D
75    R_ADD1          RES     1       ;3E
76    R_ADD2          RES     1       ;3F
77    R_TIME3         RES     1       ;40
78    R_TIME2         RES     1       ;41
79    R_TIME1         RES     1       ;42
80    R_TIME0         RES     1       ;43
81    ;
82    CKSM0           RES     1       ;44     LS Byte for checksum
83    CKSM1           RES     1       ;45     MS Byte for checksum
84    BLCK0           RES     1       ;46     LS Byte, count of Frames uploaded
85    BLCK1           RES     1       ;47     MS Byte, count of Frames uploaded
86    ;
87    ; Loop counters, other temporary values
88    MeasSet_Cnt     RES     1       ;48  Loop counter, sets of 9 analog measurements
89    Channel_Cnt     RES     1       ;49  Loop counter, analog channels 0 to 9 skip 7
90    Sleep_Cnt       RES     1       ;4A  Loop counter, sleep cycles between conductivity check
91    Loop_Cnt        RES     1       ;4B  Generic Loop Counter
92    TEMP            RES     1       ;4C  Generic temporary (Don't use for Interrupt routines)
93    PATTERN         RES     1       ;4D  Test pattern in memory test
94    ;
95    ;*****************************************************************************
96    ;*****************************************************************************
97    ; ---- Registers accessible regardless of Memory Bank selected
98    ; Used to push data during interrupt.  Uses shared region of user memory,
99    ; locations 0x070-0x07F, 0x0F0-0x0FF, 0x170-0x17F, 0x1F0-0x1FF
100   Stack_Vars      UDATA_SHR  0x70         ; Accessible from all Banks General Purpose Data RAM
101   TEMP_INT        RES     1       ;70  Temporary for interrupt routine only
102   LOOP_INT        RES     1       ;71  Temporary for interrupt routine only
103   STACK_Wreg      RES     1       ;72 Working Register holding during interrupts
104   STACK_Status    RES     1       ;73 Status Register holding during interrupts
105   STACK_FSR       RES     1       ;74 Indirect pointer holding during interrupts
106   ;
107   ;
108   ;*****************************************************************************
109   ;*****************************************************************************
110   ; ---- Command Parameter Stack
111   ; Note: Code does not check to see if stack exceeded
112   ; Allocation cannot exceed address 0x6F
113   Parm_Stack      UDATA   CMD_Param_Start        ; Stack start for command parameters passed from PC
114   CMD_PARAMS      RES     Param_Stack_Size  ; Maximum 32 values, 0x50 to 0x6F
115   ;
116   ;*****************************************************************************
117   ;*****************************************************************************
118   ; ---- Data Hold Stack
119   ; Holds data read from memory to allow re-transmit if checksum error
120   Data_Stack      UDATA   Data_Start      ; Bank 1 memory
121   Data_Values     RES     0x50    ; Maximum 80 values in stack, addr 0xA0 - 0xEF
122   ;
123   ;*****************************************************************************
124   ;********************* End of file Ball_Dat.inc ******************************
125   ;*****************************************************************************
```

Code Listing – Ball_Equ.inc

```
1   ; File name: "ball_equ.inc"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Program memory definition include file in main microcontroller code for the P&G Sensor Ball
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       4/26/99  Modified for MilliPen TA Rohwer
11  ;       30 Mar 2001 - Version 3 Changes requested by P&G, ME Partridge
12  ;           New variables added to support changes
13  ;       12 Aug 2001 - Version 4, ME Partridge
14  ;           Add ADC_AVG flag to control subroutine ADC_Read
15  ;           Add LED_BLINK flag for Timeout_Chk subroutine
16  ;
17  ; Function:
18  ;       Includes all equivalence and constant Declarations
19  ;       EQU's are compiler directives and can't be declared global
20  ;       #include in each program file
21  ;
22  ;*******************************************************************************
23  ;*******************************************************************************
24  ;
25  ; NOTE: If Version and/or Unit are changed, change List_Who_ID in Ball_Msg.inc
26  Hardware_Ver    set     "4"             ; Version 4 Hardware / Firmware
27  Unit_Number     set     "7"             ; Unit number 3
28  ;
29  ;
30  ;------------------------------Flag Bits for Flag Registers
31  ;---- PROG_FLAG
32  COMMAND_FLG      EQU    0       ; Command Flag, Valid Command String Received
33  ADC_AVG         EQU     1       ; Take average for ADC acquisition
34  LED_BLINK       EQU     2       ; Used in Timeout_Chk to request "Waiting" blink
35  ;               EQU     3       ;
36  REINIT_FLG      EQU     4       ; INIT_UP Flag, starts power-up initialization
37  ACQUIRE_FLG     EQU     5       ; Acquire a data Record, Command A
38  SIMPLE_FLG      EQU     6       ; Simple, stupid upload in RD_HAND, Command U
39  CMDERROR_FLG     EQU    7       ; Command Character is undefined command
40  ;
41  ;---- SER_FLAG
42  CNTRLC_FLG       EQU    0       ;CNTRL-C Flag, READ Abort Command
43  XON_FLG          EQU    1       ;XON Flag, S/W Handshaking starts transmission
44  XOFF_FLG         EQU    2       ;XOFF Flag, S/W Handshaking stops transmission
45  ACK_FLG         EQU     3       ;Acknowledge Flag, Data matches checksum, continue
46  NAK_FLG         EQU     4       ;Negative Ack Flag, Data and checksum mismatch, retransmit
47  CNTL_ER_FLG     EQU     5       ;Received undefined Control Char (0x00 - 0x1F)
48  STR_ER_FLG      EQU     6       ;Command character not in "A" - "Z"
49  ;               EQU     7
50  ;
51  ;---- MEMORY_FLAG
52  MEM_FULL        EQU     0       ;Memory Full Flag, 32MB data recorded
53  MEM_PROG        EQU     1       ;Program Fail Flag, Failure during Flash program
54  MEM_ER_FLG      EQU     2       ;Block Erase Failure
55  MEM_WR_FLG      EQU     3       ;Pattern read not equal written
56  TEST55_FAIL     EQU     4       ;Test Write of 0x55 Failure
57  TESTAA_FAIL     EQU     5       ;Test Write of 0xAA Failure
58  MEM_FIND_FLG    EQU     6       ;Search for a Record failed
59  END_OF_DATA     EQU     7       ;Set flag during read handler at end
60  ;
61  ;---- SER_STATE
62  SER_ST_0        EQU     H'00'   ; Received starting "0" character
63  SER_ST_AZ       EQU     H'01'   ; Received a command character in A - Z
64  SER_ST_CR       EQU     H'02'   ; Received CR for command phrase
65  SER_ST_LF       EQU     H'03'   ; Received LF for command phrase
66  SER_ST_PAR      EQU     H'04'   ; Received a command parameter
67  SER_VERIFY      EQU     H'05'   ; Received Verify character on subsequent phrase
68  ;
69  ;------------------------------I/O PORT Pin References
70  ;                               ;See P17C756.inc for PORT address references
71  ;---- PORTA, Bank0
```

Code Listing – Ball_Equ.inc

```
72   ; Note: analog channels are accessed by their channel number, NOT the Port line
73   ACCEL_X         EQU     0x00    ;Analog Channel 0, ACCEL_X
74   ACCEL_Y         EQU     0x01    ;Analog Channel 1, ACCEL_Y
75   ACCEL_Z         EQU     0x02    ;Analog Channel 2, ACCEL_Z
76   pH              EQU     0x03    ;Analog Channel 3, pH
77   CONDUCT         EQU     0x05    ;Analog Channel 4, CONDUCT
78   ;
79   ;---- PORTB, Bank0
80   ; Note: analog channels are accessed by their channel number, NOT the Port line
81   ATTENTION       EQU     0x00    ;Command attention button Input
82   POWER           EQU     0x01    ;POWER 3V & 6V control output
83   REG1_MON        EQU     0x02    ;Analog Channel 8, REG1_MON, 3V
84   REG2_MON        EQU     0x03    ;Analog Channel 9, REG2_MON, 6V
85   ISR_TEST        EQU     0x04    ;Test output, state change on ISR entry
86   CLE             EQU     0x05    ;Flash Command Latch Enable
87   ALE             EQU     0x06    ;Flash Address Latch Enable
88   WP              EQU     0x07    ;Flash Write Protect (active low)
89   ;
90   ;---- PORTC, Bank0
91   ;PD             EQU     0x00    ;Power down (active low)
92   LED             EQU     0x01    ;LED Power output, active high
93   RB              EQU     0x02    ;Input, Ready/Busy~ output on Flash, not used
94   RE              EQU     0x03    ;Flash Read enable (active low)
95   CE              EQU     0x04    ;Flash Chip enable (active low)
96   WE              EQU     0x05    ;Flash Write enable (active low)
97   TX              EQU     0x06    ;USART Transmit Line
98   RX              EQU     0x07    ;USART Receive Line
99   ;
100  ;---- PORTD, Bank0
101  IO1             EQU     0x00    ;Flash Address, Command Input, Data I/O
102  IO2             EQU     0x01    ;
103  IO3             EQU     0x02    ;
104  IO4             EQU     0x03    ;
105  IO5             EQU     0x04    ;
106  IO6             EQU     0x05    ;
107  IO7             EQU     0x06    ;
108  IO8             EQU     0x07    ;
109  ;
110  ;---- PORTE, Bank0
111  ; Note: analog channels are accessed by their channel number, NOT the Port line
112  TMP             EQU     0x00    ;Analog Channel 5, TEMP
113  MON9V           EQU     0x01    ;Analog Channel 6, 9V_MON
114  ;               EQU     0x02    ;Analog Channel 7, not connected
115  ;
116  ;*****************************************************************************
117  ; ---- Constants
118  Sync_Byte0      set     0xEB            ; Standard 16-bit Sync Bytes
119  Sync_Byte1      set     0x90
120  BLK_per_REC     set     8               ; 8 ea. 16kB Blocks per 128kB Record
121  NumChan         set     7               ; Number of Channels, used by SensorBall
122  NumMon          set     3               ; Number of Monitors, used by SensorBall
123  Atten_Timeout   set     0x04            ; 5-minute Timeout, command "ATTENTION" mode
124  Sleep_Cycles    set     0x19            ; Set for 25 cycles, or about one minute with WDT = 2.4sec
125  Conduct_Thresh set      0x01            ; 6.3 percent of full-scale Conductivity reading
126  ;
127  ; Define
128  #define CMD_Param_Start 0x50            ; Memory start location for parameter stack pointer
129  #define Param_Stack_Size 0x20           ; 32 locations for parameters
130  #define Data_Start 0xA0                         ; Bank 1 memory for data stack
131  ;
132  ;*****************************************************************************
133  ;************************ END of FILE Ball_Equ.asm ***************************
134  ;*****************************************************************************
```

Code Listing – Ball_Msg.inc

```
1   ; File name: "ball_msg.inc"
2   ; P&G Sensor Ball, MicroChip PIC16C774 MicroController Assembly Code
3   ; Message string definition include file in main microcontroller code
4   ;
5   ; Date: 11 December 2001
6   ; File Version: 4
7   ; Author: Tedd A Rohwer, Sandia National Laboratories
8   ;
9   ; Change history:
10  ;       26 Oct 2001 - Created for Version 4
11  ;           Put message strings in separate include file
12  ;
13  ; Function:
14  ;       Includes message strings as Retlw "value" instructions.  Called
15  ;       from the TX_String subroutine in file SER_HAND.
16  ;       Only #include once in the main program (ball.asm)
17  ;
18  ;*****************************************************************************
19  ;*************** Define text strings for transmit, relocatable ***************
20  ;*****************************************************************************
21  ; These strings are used by the TX_Lookup routine, where a Program Counter
22  ; value is generated equal to the Retlw line with the requested value.
23  ;
24  List_Strings   CODE
25  ;
26  ; NOTE: If Version and/or Unit are changed, change the Hardware_Ver and
27  ; Unit_Number Ball_Equ.inc
28  List_WhoID
29          Retlw   'S'
30          Retlw   'e'
31          Retlw   'n'
32          Retlw   's'
33          Retlw   'o'
34          Retlw   'r'
35          Retlw   ' '
36          Retlw   'B'
37          Retlw   'a'
38          Retlw   'l'
39          Retlw   'l'
40          Retlw   ' '
41          Retlw   'V'
42          Retlw   'e'
43          Retlw   'r'
44          Retlw   's'
45          Retlw   'i'
46          Retlw   'o'
47          Retlw   'n'
48          Retlw   ' '
49          Retlw   '4'
50          Retlw   ' '
51          Retlw   'U'
52          Retlw   'n'
53          Retlw   'i'
54          Retlw   't'
55          Retlw   ' '
56          Retlw   '0'
57          Retlw   '0'
58          Retlw   '7'
59          Retlw   0
60  ;
61  List_Commands
62          Retlw   'A'
63          Retlw   'B'
64          Retlw   'E'
65          Retlw   'F'
66          Retlw   'I'
67          Retlw   'P'
68          Retlw   'R'
69          Retlw   'S'
70          Retlw   'T'
71          Retlw   0
```

Code Listing – Ball_Msg.inc

```
72  ;
73  ; Data channel names.  All must be 16 Bytes long, including terminating null
74  List_Data00
75          Retlw   'A'
76          Retlw   'c'
77          Retlw   'c'
78          Retlw   'e'
79          Retlw   'l'
80          Retlw   'e'
81          Retlw   'r'
82          Retlw   'a'
83          Retlw   't'
84          Retlw   'i'
85          Retlw   'o'
86          Retlw   'n'
87          Retlw   ' '
88          Retlw   'X'
89          Retlw   0
90          Retlw   0
91  List_Data01
92          Retlw   'A'
93          Retlw   'c'
94          Retlw   'c'
95          Retlw   'e'
96          Retlw   'l'
97          Retlw   'e'
98          Retlw   'r'
99          Retlw   'a'
100         Retlw   't'
101         Retlw   'i'
102         Retlw   'o'
103         Retlw   'n'
104         Retlw   ' '
105         Retlw   'Y'
106         Retlw   0
107         Retlw   0
108 List_Data02
109         Retlw   'A'
110         Retlw   'c'
111         Retlw   'c'
112         Retlw   'e'
113         Retlw   'l'
114         Retlw   'e'
115         Retlw   'r'
116         Retlw   'a'
117         Retlw   't'
118         Retlw   'i'
119         Retlw   'o'
120         Retlw   'n'
121         Retlw   ' '
122         Retlw   'Z'
123         Retlw   0
124         Retlw   0
125 List_Data03
126         Retlw   'p'
127         Retlw   'H'
128         Retlw   0
129         Retlw   0
130         Retlw   0
131         Retlw   0
132         Retlw   0
133         Retlw   0
134         Retlw   0
135         Retlw   0
136         Retlw   0
137         Retlw   0
138         Retlw   0
139         Retlw   0
140         Retlw   0
141         Retlw   0
142 List_Data04
```

```
143             Retlw   'C'
144             Retlw   'o'
145             Retlw   'n'
146             Retlw   'd'
147             Retlw   'u'
148             Retlw   'c'
149             Retlw   't'
150             Retlw   'i'
151             Retlw   'v'
152             Retlw   'i'
153             Retlw   't'
154             Retlw   'y'
155             Retlw   0
156             Retlw   0
157             Retlw   0
158             Retlw   0
159     List_Data05
160             Retlw   'T'
161             Retlw   'e'
162             Retlw   'm'
163             Retlw   'p'
164             Retlw   'e'
165             Retlw   'r'
166             Retlw   'a'
167             Retlw   't'
168             Retlw   'u'
169             Retlw   'r'
170             Retlw   'e'
171             Retlw   0
172             Retlw   0
173             Retlw   0
174             Retlw   0
175             Retlw   0
176     List_Data06
177             Retlw   'B'
178             Retlw   'a'
179             Retlw   't'
180             Retlw   't'
181             Retlw   'e'
182             Retlw   'r'
183             Retlw   'y'
184             Retlw   ' '
185             Retlw   'V'
186             Retlw   'o'
187             Retlw   'l'
188             Retlw   't'
189             Retlw   0
190             Retlw   0
191             Retlw   0
192             Retlw   0
193     List_Data07
194             Retlw   '+'
195             Retlw   '3'
196             Retlw   'V'
197             Retlw   ' '
198             Retlw   'R'
199             Retlw   'e'
200             Retlw   'g'
201             Retlw   'u'
202             Retlw   'l'
203             Retlw   'a'
204             Retlw   't'
205             Retlw   'o'
206             Retlw   'r'
207             Retlw   0
208             Retlw   0
209             Retlw   0
210     List_Data08
211             Retlw   '+'
212             Retlw   '6'
213             Retlw   'V'
```

```
214             Retlw   ' '
215             Retlw   'R'
216             Retlw   'e'
217             Retlw   'g'
218             Retlw   'u'
219             Retlw   'l'
220             Retlw   'a'
221             Retlw   't'
222             Retlw   'o'
223             Retlw   'r'
224             Retlw   0
225             Retlw   0
226             Retlw   0
227     List_Data09
228             Retlw   'R'
229             Retlw   'e'
230             Retlw   'c'
231             Retlw   'o'
232             Retlw   'r'
233             Retlw   'd'
234             Retlw   ' '
235             Retlw   '&'
236             Retlw   ' '
237             Retlw   'F'
238             Retlw   'r'
239             Retlw   'a'
240             Retlw   'm'
241             Retlw   'e'
242             Retlw   0
243             Retlw   0
244     List_Data10
245             Retlw   'T'
246             Retlw   'i'
247             Retlw   'm'
248             Retlw   'e'
249             Retlw   ' '
250             Retlw   'M'
251             Retlw   'S'
252             Retlw   'W'
253             Retlw   0
254             Retlw   0
255             Retlw   0
256             Retlw   0
257             Retlw   0
258             Retlw   0
259             Retlw   0
260             Retlw   0
261     List_Data11
262             Retlw   'T'
263             Retlw   'i'
264             Retlw   'm'
265             Retlw   'e'
266             Retlw   ' '
267             Retlw   'L'
268             Retlw   'S'
269             Retlw   'W'
270             Retlw   0
271             Retlw   0
272             Retlw   0
273             Retlw   0
274             Retlw   0
275             Retlw   0
276             Retlw   0
277             Retlw   0
278     List_Data12
279             Retlw   'M'
280             Retlw   'e'
281             Retlw   'm'
282             Retlw   ' '
283             Retlw   'A'
284             Retlw   'd'
```

```
285         Retlw   'd'
286         Retlw   'r'
287         Retlw   'e'
288         Retlw   's'
289         Retlw   's'
290         Retlw   0
291         Retlw   0
292         Retlw   0
293         Retlw   0
294         Retlw   0
295 ;
296 ; Self-Test bit names. All must be 16 Bytes long, including terminating null
297 List_Slft00                ;Memory Full Flag, 32MB data recorded
298         Retlw   'M'
299         Retlw   'e'
300         Retlw   'm'
301         Retlw   'o'
302         Retlw   'r'
303         Retlw   'y'
304         Retlw   ' '
305         Retlw   'F'
306         Retlw   'u'
307         Retlw   'l'
308         Retlw   'l'
309         Retlw   0
310         Retlw   0
311         Retlw   0
312         Retlw   0
313         Retlw   0
314 List_Slft01                ;Program Fail Flag, Failure during Flash program
315         Retlw   'P'
316         Retlw   'r'
317         Retlw   'o'
318         Retlw   'g'
319         Retlw   'r'
320         Retlw   'a'
321         Retlw   'm'
322         Retlw   ' '
323         Retlw   'F'
324         Retlw   'a'
325         Retlw   'i'
326         Retlw   'l'
327         Retlw   0
328         Retlw   0
329         Retlw   0
330         Retlw   0
331 List_Slft02                ;Block Erase Failure
332         Retlw   'E'
333         Retlw   'r'
334         Retlw   'a'
335         Retlw   's'
336         Retlw   'e'
337         Retlw   ' '
338         Retlw   'F'
339         Retlw   'a'
340         Retlw   'i'
341         Retlw   'l'
342         Retlw   0
343         Retlw   0
344         Retlw   0
345         Retlw   0
346         Retlw   0
347         Retlw   0
348 List_Slft03                ;Pattern read not equal written
349         Retlw   'R'
350         Retlw   'e'
351         Retlw   'a'
352         Retlw   'd'
353         Retlw   'b'
354         Retlw   'a'
355         Retlw   'c'
```

```
356           Retlw   'k'
357           Retlw   ' '
358           Retlw   'E'
359           Retlw   'r'
360           Retlw   'r'
361           Retlw   'o'
362           Retlw   'r'
363           Retlw   0
364           Retlw   0
365     List_Slft04                   ;Test Write of 0x55 Failure
366           Retlw   'P'
367           Retlw   'a'
368           Retlw   't'
369           Retlw   't'
370           Retlw   'e'
371           Retlw   'r'
372           Retlw   'n'
373           Retlw   ' '
374           Retlw   '5'
375           Retlw   '5'
376           Retlw   ' '
377           Retlw   'F'
378           Retlw   'a'
379           Retlw   'i'
380           Retlw   'l'
381           Retlw   0
382     List_Slft05                   ;Test Write of 0xAA Failure
383           Retlw   'P'
384           Retlw   'a'
385           Retlw   't'
386           Retlw   't'
387           Retlw   'e'
388           Retlw   'r'
389           Retlw   'n'
390           Retlw   ' '
391           Retlw   'A'
392           Retlw   'A'
393           Retlw   ' '
394           Retlw   'F'
395           Retlw   'a'
396           Retlw   'i'
397           Retlw   'l'
398           Retlw   0
399     List_Slft06                   ;Search for a Record failed
400           Retlw   'R'
401           Retlw   'e'
402           Retlw   'c'
403           Retlw   ' '
404           Retlw   'S'
405           Retlw   'e'
406           Retlw   'a'
407           Retlw   'r'
408           Retlw   'c'
409           Retlw   'h'
410           Retlw   ' '
411           Retlw   'F'
412           Retlw   'a'
413           Retlw   'i'
414           Retlw   'l'
415           Retlw   0
416     List_Slft07                   ;Set flag during read handler at end
417           Retlw   'M'
418           Retlw   'e'
419           Retlw   'm'
420           Retlw   'o'
421           Retlw   'r'
422           Retlw   'y'
423           Retlw   ' '
424           Retlw   'E'
425           Retlw   'n'
426           Retlw   'd'
```

```
427             Retlw   0
428             Retlw   0
429             Retlw   0
430             Retlw   0
431             Retlw   0
432             Retlw   0
433     ;
434     ; Undefined channel or test bit name requested.
435     List_Undef
436             Retlw   '?'
437             Retlw   ' '
438             Retlw   'U'
439             Retlw   'n'
440             Retlw   'd'
441             Retlw   'e'
442             Retlw   'f'
443             Retlw   'i'
444             Retlw   'n'
445             Retlw   'e'
446             Retlw   'd'
447             Retlw   0
448     ;
449     ; Messages
450     List_Fail_Msg                   ; "Failed Block "
451             Retlw   'F'
452             Retlw   'a'
453             Retlw   'i'
454             Retlw   'l'
455             Retlw   'e'
456             Retlw   'd'
457             Retlw   ' '
458             Retlw   'B'
459             Retlw   'l'
460             Retlw   'o'
461             Retlw   'c'
462             Retlw   'k'
463             Retlw   ' '
464             Retlw   0
465     List_Stat_Msg                   ; "Finished Block "
466             Retlw   'F'
467             Retlw   'i'
468             Retlw   'n'
469             Retlw   'i'
470             Retlw   's'
471             Retlw   'h'
472             Retlw   'e'
473             Retlw   'd'
474             Retlw   ' '
475             Retlw   'B'
476             Retlw   'l'
477             Retlw   'o'
478             Retlw   'c'
479             Retlw   'k'
480             Retlw   ' '
481             Retlw   0
482     ;
483     List_Eras_Msg                   ; "Erased Block "
484             Retlw   'E'
485             Retlw   'r'
486             Retlw   'a'
487             Retlw   's'
488             Retlw   'e'
489             Retlw   'd'
490             Retlw   ' '
491             Retlw   'B'
492             Retlw   'l'
493             Retlw   'o'
494             Retlw   'c'
495             Retlw   'k'
496             Retlw   ' '
497             Retlw   0
```

Code Listing – Ball_Msg.inc

```
498   ;
499   List_Mem_Flag                 ; " Mem Flag Value "
500           Retlw   ' '
501           Retlw   'M'
502           Retlw   'e'
503           Retlw   'm'
504           Retlw   ' '
505           Retlw   'F'
506           Retlw   'l'
507           Retlw   'a'
508           Retlw   'g'
509           Retlw   ' '
510           Retlw   'V'
511           Retlw   'a'
512           Retlw   'l'
513           Retlw   'u'
514           Retlw   'e'
515           Retlw   ' '
516           Retlw   0
517   ;
518   ;*******************************************************************************
519   ;******************** End of file Ball_Msg.inc ******************************
520   ;*******************************************************************************
```

Appendix L: PC Interface (SensorBall) Code Listing

```
1   unit Main2Form;
2
3   interface
4
5   uses
6     Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
7     StdCtrls, ExtCtrls, About, MainUtil, CommUtil, Spin, ComCtrls;
8
9   type
10    TForm_Main = class(TForm)
11      Group_PortStatus: TGroupBox;
12      GroupBox_Reply: TGroupBox;
13      Group_Readout: TGroupBox;
14      Group_Status: TGroupBox;
15      Button_Send: TButton;
16      Edit_Outfile: TEdit;
17      Button_Read: TButton;
18      Label_Reply: TLabel;
19      Button_About: TButton;
20      Button_Close: TButton;
21      Label_Status: TLabel;
22      Label_ChanData1_3: TLabel;
23      Label_VoltData1_4: TLabel;
24      SaveDialog1: TSaveDialog;
25      Group_SelfTest: TGroupBox;
26      Label_ARM_Lvl: TLabel;
27      Label_Cycle_Lvl: TLabel;
28      Label_ILock_Lvl: TLabel;
29      Label_Bit3A: TLabel;
30      Label_Bit4A: TLabel;
31      Label_Bit5A: TLabel;
32      Label_Bit6A: TLabel;
33      Label_Bit7A: TLabel;
34      Label_RAMData: TLabel;
35      Label_RAMAddrRW: TLabel;
36      Label_EEPRAddrRW: TLabel;
37      Label_Bit3B: TLabel;
38      Label_Bit4B: TLabel;
39      Label_Bit5B: TLabel;
40      Label_Bit6B: TLabel;
41      Label_Bit7B: TLabel;
42      Label_KBytes: TLabel;
43      Panel1: TPanel;
44      Label_RW: TLabel;
45      ProgressBar_RW: TProgressBar;
46      Label_AAL: TLabel;
47      Label_ADL: TLabel;
48      Label_ACL: TLabel;
49      Label_AIL: TLabel;
50      Label_RAB4: TLabel;
51      Label_RAB5: TLabel;
52      Label_RAB6: TLabel;
53      Label_RAB7: TLabel;
54      Label_RARW: TLabel;
55      Label_RD: TLabel;
56      Label_EARW: TLabel;
57      Label_ED: TLabel;
58      Label_RBB4: TLabel;
59      Label_RBB5: TLabel;
60      Label_RBB6: TLabel;
61      Label_RBB7: TLabel;
62      Button_XOut: TButton;
63      GroupBox_Command: TGroupBox;
64      RadioButton_I: TRadioButton;
65      RadioButton_R: TRadioButton;
66      RadioButton_A: TRadioButton;
```

```
67        RadioButton_S: TRadioButton;
68        RadioButton_P: TRadioButton;
69        RadioButton_T: TRadioButton;
70        SpinEdit_Read: TSpinEdit;
71        Label2: TLabel;
72        RadioButton_H: TRadioButton;
73        RadioButton_C: TRadioButton;
74        RadioButton_G: TRadioButton;
75        RadioButton_D: TRadioButton;
76        GainGroup: TGroupBox;
77        Label1: TLabel;
78        Label3: TLabel;
79        EditGain: TEdit;
80        Label_SR: TLabel;
81        RadioButton_B: TRadioButton;
82        ListHexIn: TListBox;
83        Button1: TButton;
84        DebugDisplayOn: TCheckBox;
85        Label5: TLabel;
86        Label_ChksumNum: TLabel;
87
88        procedure ReadData(Block_Cnt : Integer);
89        procedure WriteData(Block_Cnt : Integer);
90        procedure Button_AboutClick(Sender: TObject);
91        procedure Button_SendClick(Sender: TObject);
92        procedure FormClose(Sender: TObject; var Action: TCloseAction);
93        procedure Edit_OutfileClick(Sender: TObject);
94        procedure Button_CloseClick(Sender: TObject);
95        procedure Button_ReadClick(Sender: TObject);
96        procedure Read_Analog_Status;
97    {     procedure Read_Unit_Status; }
98        procedure Read_SelfTest;
99        procedure SpinEdit_ReadChange(Sender: TObject);
100       procedure FormCreate(Sender: TObject);
101       procedure Button_XOutClick(Sender: TObject);
102       procedure RadioButton_CommandClick(Sender: TObject);
103       procedure Button1Click(Sender: TObject);
104       procedure DebugDisplayOnClick(Sender: TObject);
105
106     private
107       { Private declarations }
108     public
109       { Public declarations }
110     end;
111
112   const
113     Revision_Code = '12.01';
114     Cal_Factor    = 0.003549;              {Set Voltage Mons to 4.88 mV/Count}
115     Cal_FactorT   = 0.20000;              {Set Temp    Mon  to 0.20 °C/Count}
116
117   const
118     Gain_Length = 10;
119
120   var
121     Form_Main: TForm_Main;
122     SpEdPrV         : Integer;             {Spin Edit Previous Value}
123     OutFileValid    : Boolean;             {Enable Read Button if Outfile Valid}
124     Buff_Ptr        : Pointer;
125     Num_Data_Chans,
126     Num_Mon_Chans   : Byte;
127     Block_Cnt       : Integer;              {Counter for Number of Data Blocks Read}
128     HW_Rev       : Char;
129     Unit_ID      : Byte;
130     DebugOn      : Boolean;
131
132   implementation
133
134   {$R *.DFM}
135
136   {-------------------------------------------------------------------------}
137   procedure TimeOutError(Msg_Str : String);
```

```
138
139   begin
140   Comm_Err := True;
141   {MessageDlg('A read timeout has occurred while waiting for a ' + Msg_str +
142             '.', mtWarning, [mbOK], 0);
143   }
144     Msg_Str := 'Read timeout waiting for a ' + Msg_str +  '.' ;
145     Form_Main.ListHexIn.Items.Add(Msg_Str);
146   end;
147
148   {-------------------------------------------------------------}
149   procedure TForm_Main.ReadData(Block_Cnt : Integer);
150   {12/6/01 This section revised to read the entire group of Sync, Block count,
151   data block, and Checksum in one read.  This matches the way Sensor Ball sends
152   the data, as one group.  Sensor Ball then waits to receive either the Acknowledge
153   (Ack) or Negative Acknowledge (Nak) to indicate whether the data were received
154   OK.}
155
156   const
157     Cksm_Length = 2;                  {Checksum String consist of 2 Bytes}
158     Sync_Length = 4;                  {Two Sync bytes plus two Block count bytes}
159   var
160     Offset,
161     Idx       : LongInt;
162     MissBlock,
163     Read_Cnt,
164     Block_Num : Integer;
165     NB_Read   : DWord;
166     Checksum  : Word;  {010618ed unsigned 16 bit}
167     WaitLoopCount,
168     CksmErrCount,
169     SyncErrCount  : Integer; {010618ed}
170     Index     : Integer;
171     DisplayStr : String;
172     GotData : Boolean;
173
174   begin
175     Block_Num := 0;                             {Initialize Block Counter}
176     MissBlock := 0;
177     CksmErrCount := 0; {010618ed initialize counter for counting checksum errors}
178     SyncErrCount := 0;
179     Label_ChksumNum.Caption := 'C:'+ IntToStr(CksmErrCount) + ' S:' + IntToStr(SyncErrCount);
180     Label_RW.Caption := 'Reading Data from Package.';
181     Label_RW.Visible := True;
182     ProgressBar_RW.Visible := True;
183     Label_ChksumNum.Visible := True;
184     Form_Main.Refresh;
185     Cursor    := crHourGlass;
186     Comm_Err := False;
187     Repeat
188       ProgressBar_RW.Position := Round((Block_Num / Block_Cnt) * 100);
189       WaitLoopCount := 0;
190       Checksum := 0;
191
192       {Monitor communications until Sync, Block num, Data, and Cksm are in buffer.
193       On the first time through, the interface program needs to send X-on to the
194       Sensor Ball to indicate it is ready to receive data.  This loop is repeated
195       in case the X-on byte was not received.  Once the Sensor Ball has received
196       X-on, only either Ack or Nak should be sent.  In the event an error occurred
197       in the Block 0 transmission, a compare is also made for Comm_Error.}
198       Repeat
199         FlushQueue(Receive);
200         if (Block_Num = 0) AND NOT Comm_Err then    {12-5-01 mike}
201           SendChar(XonChar)
202         else
203           if Comm_Err then
204             SendChar(NakChar)
205           else
206             SendChar(AckChar);      {7-5-01 10:13 mike}
207         if DebugOn then Form_Main.ListHexIn.Items.Add('  ');
208         {Changed to read all data in one statement.  It is sent as one group.
```

```
209         For the first read, the Sensor Ball should be in a wait loop and respond
210         in less than a millisecond.  Thus the short timeout value.  }
211         GotData := WaitForCommData(Sync_Length, 1000);   { 4 bytes received, 1 second timeout}
212         if GotData then
213           GotData := (ReadFile(Comm_Handle, Data_Block, Sync_Length, NB_Read, lpOverLapped)
214             AND (NB_Read = Sync_Length));
215         if GotData then {Sucessful ReadFile Processing}
216           if (Data_Block [0] = Ord(SyncByte1)) AND (Data_Block [1] = Ord(SyncByte2)) then
217             begin {Sync OK Processing}
218               if DebugOn then
219                 begin
220                   DisplayStr := 'Sync = ' + IntToHex(Data_Block[0], 2)
221                     + IntToHex(Data_Block[1], 2) + ', First Frame of Block '
222                     + IntToStr(Block_Num);
223                   Form_Main.ListHexIn.Items.Add(DisplayStr);
224                 end;
225             end  {Sync OK Processing}
226           else   {Sync error}
227             begin
228               GotData := False;
229               SyncErrCount := SyncErrCount + 1;
230               DisplayStr := 'Block ' + IntToStr(Block_Num) + ' Sync Error, Received '
231                 + IntToHex(Data_Block[0], 2) + ' ' + IntToHex(Data_Block[1], 2);
232               Form_Main.ListHexIn.Items.Add(DisplayStr);
233             end;  {of Sync bytes not matching}
234         WaitLoopCount := WaitLoopCount + 1;
235       until GotData OR (WaitLoopCount > 20);
236
237       if GotData then {Verified that the proper Sync was received, now check block count.}
238         begin
239           Read_Cnt := (Data_Block[3] * 256) + Data_Block[2];
240           if NOT (Read_Cnt + MissBlock = Block_Num) then
241             begin
242               DisplayStr := 'Expected block ' + IntToStr(Block_Num)
243                       + ', Sensor Ball sending block ' + IntToStr(Read_Cnt);
244               Form_Main.ListHexIn.Items.Add(DisplayStr);
245               MissBlock := Block_Num - Read_Cnt;
246             end
247         end;  {End of check block number}
248
249       {Make sure data block is waiting in receive queue}
250       if GotData then
251         if WaitForCommData(Block_Length, 1000) then  { Wait 1000ms for block}
252           GotData := (ReadFile(Comm_Handle, Data_Block, Block_Length, NB_Read, lpOverLapped)
253              AND (NB_Read = Block_Length));
254
255       {Read and store data block and calculate checksum }
256       if GotData then
257         begin {Sucessful ReadFile Processing}
258           DisplayDebug2(false, 80, Data_Block);
259           Checksum := 0;
260           Offset := Block_Num * Block_Length;
261           for Idx := 0 to (Block_Length - 1) do
262             begin
263               Byte_Record[Offset + Idx] := Data_Block[Idx];
264               Checksum := Checksum + Data_Block[Idx];
265             end
266         end;   {of Sucessful ReadFile Processing}
267
268       {Make sure checksum is waiting in receive queue}
269       if GotData then
270         if WaitForCommData(Cksm_Length, 50) then  { Wait 50 ms for checksum}
271           GotData :=(ReadFile(Comm_Handle, Data_Block, Cksm_Length, NB_Read, lpOverLapped)
272              AND (NB_Read = Cksm_Length));
273
274       if GotData then  {Test if Checksum correct}
275         if (Data_Block[1] * 256 + Data_Block[0] = Checksum) then
276           begin  {Checksum correct}
277             Comm_Err := False;
278             Block_Num := Block_Num + 1;
279             DisplayStr := 'Checksum Received '
```

```
280             + IntToHex(Data_Block[1], 2) + ' '
281             + IntToHex(Data_Block[0], 2);
282           if DebugOn then Form_Main.ListHexIn.Items.Add(DisplayStr);
283         end  {of Checksum correct}
284       else
285         begin  {Checksum Error}
286           Comm_Err := True;
287           CksmErrCount := CksmErrCount + 1;
288           DisplayStr := 'Block ' + IntToStr(Block_Num)
289             + ' Cksm Error, Rcvd '
290             + IntToHex(Data_Block[1], 2)
291             + IntToHex(Data_Block[0], 2) + ' = '
292             + IntToStr(Data_Block[1]*256 + Data_Block[0])
293             + ', Calc= '
294             + IntToStr(CheckSum);
295           Form_Main.ListHexIn.Items.Add(DisplayStr);
296         end;  {of Checksum error}
297
298     {Summarize if had communications error, either bad Sync or bad Chksm}
299     if Comm_Err then
300       Label_ChksumNum.Caption := 'C:'+ IntToStr(CksmErrCount)
301         + ' S:' + IntToStr(SyncErrCount) ;       {010618ed display # checksum errors}
302
303   {Terminate loop when all blocks recovered, or if Sync search times out}
304   Until ((Block_Num = Block_Cnt) OR NOT GotData);
305
306   {Clean up for exit}
307   Cursor := crDefault;
308   Label_RW.Visible := False;
309   ProgressBar_RW.Visible := False;
310
311   {Stop data transmit}
312   for Index := 0 to 3 do  { May have missed first}
313     begin
314       SendChar(Cntl_C);                          {Cntl_C to Stop Data}
315       Delay(500);                                {Wait 500 mSec for Data to Stop}
316       FlushQueue(Receive);                       {Empty Receive Buffer}
317     end;
318 end;
319
320 {-------------------------------------------------------------}
321 procedure TForm_Main.WriteData(Block_Cnt : Integer);
322
323 var
324   Offset,
325   Idx       : LongInt;
326   Block_Num : Word;
327
328 begin
329 Label_RW.Caption := 'Writing Data to Disk.';
330 Label_RW.Visible := True;
331 ProgressBar_RW.Visible := True;
332 Form_Main.Refresh;
333 Block_Num := 0;
334 Cursor    := crHourGlass;
335 Repeat
336   ProgressBar_RW.Position := Round((Block_Num / Block_Cnt) * 100);
337   Offset := Block_Num * Block_Length;
338   for Idx := 0 to (Block_Length - 1) do
339     Data_Block[Idx] := Byte_Record[Offset + Idx];
340   BlockWrite(OutFileB, Data_Block, 1);
341 Block_Num := Block_Num + 1;
342 until (Block_Num = Block_Cnt);
343 Cursor := crDefault;
344 Label_RW.Visible := False;
345 ProgressBar_RW.Visible := False;
346 end;
347
348 {-------------------------------------------------------------}
349 procedure TForm_Main.Button_AboutClick(Sender: TObject);
350
```

```
351  var
352    Compile_DateTime : TDATETIME;
353
354  begin
355  Compile_DateTime             :=  GetFileDT(Application.ExeName);
356  AboutBox.Caption             := 'About '          + Application.Title;
357  AboutBox.ProgramName.Caption := 'Program Name : ' + UpperCase(Application.Title);
358  AboutBox.Version.Caption     := 'Version : '      + Revision_Code;
359  AboutBox.Date.Caption        := 'Compiled : '     + FormatDateTime('mm/dd/yy',
360                                                      Compile_DateTime);
361  AboutBox.Copyright.Caption   := 'Copyright : '    + 'Sandia National Labs';
362  AboutBox.Comments.Caption    := 'Written by : '   + 'David J. Bello' + CR
363                                + 'Revisions by Frank Wunderlin, Ed Henry,' + CR
364                                + 'and Mike Partridge';
365  AboutBox.ShowModal;
366  end;
367
368  {-------------------------------------------------------------}
369  procedure TForm_Main.Button_SendClick(Sender: TObject);
370
371  {var
372   TimeChar : Char;
373   ByteEntered : Integer;
374   Idx       : LongInt; {010606ed added for baud change}
375
376  begin
377
378  {010618ed change from A to E}
379  if (Cmd_Char = 'E')
380    then
381      if MessageDlg('Warning! You are about to ERASE ALL DATA!!!' + CR +
382                'Proceed with Erase Command?', mtConfirmation, mbOKCancel, 1) = mrCancel
383        then
384          Exit;
385
386  SendVerifyCommand(Cmd_Char, Cmd_Echo);
387  Edit_OutFile.Enabled := False;
388  Button_Read.Enabled  := False;
389
390    {***************************** start new baud stuff **************}
391  if (Cmd_Char = 'B') then  {010606ed add code to change baud rate}
392    begin
393      if (BaudRate = 115200) then  {allows you to toggel between baud rates}
394        Begin
395          BaudRate := 19200;
396        end
397      else
398        Begin
399          BaudRate := 115200;
400        end;
401    end;
402    CloseCommPort;                              {Disconnect Comm Device}
403    OpenCommPort('COM1');
404    SetCommMODE(BaudRate,'N',8,1);
405    GetCommStatus;
406    Group_PortStatus.Caption := 'COM' + IntToStr(CommStat.PortID);
407    Label_Status.Caption := 'BaudRate : ' + IntToStr(CommStat.BaudRate) + '    ' +
408                            'Parity : ' + (CommStat.Parity)             + '    ' +
409                            'ByteSize : ' + IntToStr(CommStat.ByteSize) + '    ' +
410                            'StopBits : ' + IntToStr(CommStat.StopBits);
411  {***************************** end new baud stuff **************}
412
413  Case Cmd_Echo of
414    'A' : Label_Reply.Caption := '[A]Acquiring to Flash';
415    'B' : Label_Reply.Caption := '[B]Baud Rate Change' + IntToStr(BaudRate);{010606ed added baud
416  change}
417    'C' : Label_Reply.Caption := '[C]Cycled';
418    'E' : begin
419          Label_Reply.Caption := '[E]Erase Memory'; {010611ed changed from powerdown to Erase mem}
420          end;
421    'F' : Label_Reply.Caption := '[F]Fill Memory Test';  {Placeholder for command}
```

```
422       'I' : Label_Reply.Caption := '[I]Initializing';
423       'R' : begin
424             Label_Reply.Caption := '[R]RAM Read';
425             Edit_OutFile.Enabled := True;
426             if OutFileValid
427               then Button_Read.Enabled := True;
428             end;
429       'S' : begin
430             Label_Reply.Caption := '[S]Status Request';
431             Read_Analog_Status;
432             end;
433       'T' : begin
434             Label_Reply.Caption := '[T]Test Memory'; {010611ed changed self test to mem test}
435             Read_SelfTest;
436             end;
437       'w' : Label_Reply.Caption := '[w]Who are you?';  {Placeholder for command}
438       'z' : Label_Reply.Caption := 'No Response'
439
440       else
441         begin
442         MessageDlg('Undefined ECHO Character: ' + Cmd_Echo +
443                    '.', mtWarning, [mbOK], 0);
444         Label_Reply.Caption := Cmd_Echo + 'UNDEFINED.';
445         end;
446       end; {Case}
447     end;
448
449     {-------------------------------------------------------------}
450     procedure TForm_Main.FormClose(Sender: TObject; var Action: TCloseAction);
451
452     begin
453     CloseCommPort;                              {Disconnect Comm Device}
454     end;
455
456
457     {-------------------------------------------------------------}
458     procedure TForm_Main.Edit_OutfileClick(Sender: TObject);
459
460     begin
461     SaveDialog1.FileName := '';
462     SaveDialog1.Filter   := 'All Files (*.*)|*.*';
463     SaveDialog1.Options  := [ofHideReadOnly, ofPathMustExist];
464     if SaveDialog1.Execute
465       then
466         begin
467         OutFileName         := SaveDialog1.FileName;
468         Edit_OutFile.Text   := ExtractFileName(OutFileName);
469         OutFileValid        := True;
470         Button_Read.Enabled := True;
471         end;
472     Edit_OutFile.SelStart  := 0;
473     Edit_OutFile.SelLength := 0;
474     end;
475
476     {-------------------------------------------------------------}
477     procedure TForm_Main.Button_CloseClick(Sender: TObject);
478     begin
479     Close;
480     end;
481
482     {-------------------------------------------------------------}
483     procedure TForm_Main.Button_ReadClick(Sender: TObject);
484
485     var
486       ByteCount : int64;
487       KBytes    : int64;
488
489     begin
490     if FileExists(OutFileName)
491       then
492         if MessageDlg('File ' + OutFileName + ' already exist. Overwrite?',
```

```
                    mtWarning, [mbYes, mbNo], 0) = mrNo
      then exit;

ByteCount := SpinEdit_Read.Value * 256 * Block_Length;   {fkw - convert records to bytes}
KBytes    := ByteCount DIV 1024;                 {fkw - convert bytes to kbytes}

Block_Cnt := KBytes * 1024 DIV Block_Length; {010531ed changed 256 to Block_Length}
Err_Code := OpenOutFile(OutFileB, OutFileName,
            Block_Length, (Block_Cnt * Block_Length) DIV 1024);  {010531ed changed 256 to
Block_Length}{Open Binary Output File}
if Err_Code = 0
  then
    begin
      ReadData(Block_Cnt);
      if Comm_Err AND (Block_Cnt=0) then
        MessageDlg('Transmission Errors Detected! No Data Available.',mtWarning, [mbOK], 0)
      else
        WriteData(Block_Cnt);
      CloseFile(OutFileB);                              {Close Binary Output File}
    end;
Edit_OutFile.Enabled := False;
Button_Read.Enabled  := False;
end;

{-------------------------------------------------------------------}
procedure TForm_Main.SpinEdit_ReadChange(Sender: TObject);
begin
SpEdPrV := SpinEdit_Read.Value;
end;

{-------------------------------------------------------------------}
procedure TForm_Main.Read_Analog_Status;

var
  NB_Read      : DWord;
  CmdEcho      : Char;
  WhoID        : Array[0..50] of Char;
  Channel      : Byte;
  IdxStr       : String;
  BiasAvg      : Array[1..16] of Float;      {Max of 16 Data and Monitor Chans}
  BiasStr      : String;
  Total_Chans  : Byte;
  HighByte_Loc : Byte;
  LowByte_Loc  : Byte;
  ChanName,
  UnitStr      : String;
  EdIdx        : Integer;

begin
Comm_Err  := False;
FlushQueue(Receive);                {Clear Receive Buffer}
SendChar(XonChar);                    {SendXOn}
GetStatusInfo(Num_Data_Chans, Num_Mon_Chans); {First Bytes are Num Data and Mon Chans}
Total_Chans := Num_Data_Chans + Num_Mon_Chans;
if Total_Chans = 0
  then Exit;
Sync_OK := FindSync(0);                   {Block_Num = 0 for Analog Status Frame}
GetDataBlock;                             {Read Analog Status Data}
SendChar(Cntl_C);                         {Cntl_C to Stop Data}
Delay(500);                               {Wait 500 mSec for Data to Stop}
FlushQueue(Receive);                      {Empty Receive Buffer}
if NOT(Sync_OK) OR (Comm_Err)
  then
    Exit;

  for Channel := 1 to Total_Chans do      {for Channels 1 - Total_Chans}
    begin
    HighByte_Loc := ((2 * Channel) - 1);
    LowByte_Loc  := ((2 * Channel) - 2);
    BiasAvg[Channel] := Data_Block[HighByte_Loc] * 256
                      + Data_Block[LowByte_Loc]; {mep remove $0F AND w/ upper byte}
```

```
564       end;
565
566   for Channel := (Num_Data_Chans + 1) to (Total_Chans) do {Scale Volt Mon Chans}
567     BiasAvg[Channel] := BiasAvg[Channel] * Cal_Factor;
568
569   Label_ChanData1_3.Caption  := '';              {Clear Channel Data 1-3}
570   Label_VoltData1_4.Caption  := '';              {Clear Voltage Data 1-4}
571   EdIdx := 0; {010618ed}
572   for Channel := 1 to Num_Data_Chans do          {Channel Data = Chan 1 - 3}
573     begin
574     Str(Channel : 2, IdxStr);                    {Convert Channel to Str ##}
575     Str(Round(BiasAvg[Channel]) : 4, BiasStr);
576     {010618ed added case statement to set labels}
577     EdIdx := EdIdx + 1;
578     case  EdIdx of
579           1 :     ChanName := 'Rec. # ';
580           2 :     ChanName := 'Accel X';
581           3 :     ChanName := 'Accel Y';
582           4 :     ChanName := 'Accel Z';
583           5 :     ChanName := 'pH     ';
584           6 :     ChanName := 'Cond.  ';
585           7 :     ChanName := 'Temp.  ';
586     end;
587      {010618ed removed 'Chan' + IdxStr + & inserted ChanName}
588     Label_ChanData1_3.Caption  := Label_ChanData1_3.Caption
589                       + ChanName + ':  ' + BiasStr + ' Counts' + CR;
590     end;
591     EdIdx := 0; {010618ed}
592     for Channel := 1 to Num_Mon_Chans do           {Voltage Data = Mon 1 - 4}
593       begin
594       UnitStr := ' Volts';
595       Str(Channel : 2, IdxStr);
596       Str(BiasAvg[Channel + Num_Data_Chans]: 5 : 2, BiasStr);
597
598       {010618ed added case statement to set labels}
599       EdIdx := EdIdx + 1;
600       case  EdIdx of
601           1 :     ChanName := 'Batt Mon  ';
602           2 :     ChanName := '3V Reg Mon';
603           3 :     ChanName := '6V Reg Mon';
604       end;
605       {010618ed removed 'Mon' + IdxStr + & inserted ChanName}
606       Label_VoltData1_4.Caption := Label_VoltData1_4.Caption
607                           + ChanName + ': ' + BiasStr + UnitStr + CR;
608       end;
609   Label_VoltData1_4.Caption := Label_VoltData1_4.Caption + CR;
610
611   {Get unit ID}
612   SendCommand('w');  {Command 'w' is Who are you?}
613   FlushQueue(Transmit);                {Clear Transmit Buffer}
614   FlushQueue(Receive);                 {Clear Receive Buffer}
615   SendCommand('w');
616   ReadCommandEcho(CmdEcho);
617
618   if ('w' = CmdEcho) then
619     begin
620       SendCommand('V');              {Send Verification Character}
621       if WaitForCommData(40, 1000) then  { Wait 1000ms for block}
622         if (ReadFile(Comm_Handle, WhoID, 40, NB_Read, lpOverLapped)) then
623           begin
624             for EdIdx := 8 to 40 do Label_VoltData1_4.Caption :=
625               Label_VoltData1_4.Caption + WhoID[EdIdx];
626           end;
627     end
628   else
629     Label_VoltData1_4.Caption := Label_VoltData1_4.Caption + 'Unknown Unit';
630
631   end;
632
633
634   {-------------------------------------------------------------}
```

```
635  procedure TForm_Main.Read_SelfTest;
636
637  Const
638    Stat_Length = 4;
639    Pass = '- P';
640    Fail = '- F';
641
642  var
643    NB_Read      : DWord;
644    Stat_Str     : Array[0..Stat_Length] of Char;
645    HW_Rev       : Char;
646    Unit_ID      : Byte;
647    TResultA     : Byte;
648    TResultB     : Byte;
649
650  begin
651  Sync_OK := False;                          {Used by delay to show early exit}
652  FlushQueue(Receive);                 {Clear Receive Buffer}
653  SendChar(XonChar);                         {SendXOn}
654  if NOT(Sync_OK)                            {Block_Num = 2 for Self Test}
655    then exit;
656  if NOT(FindSync(2))                        {Block_Num = 2 for Self Test}
657    then exit;
658  if WaitForCommData(Stat_Length, FiveSec) {If Characters in Rec Queue - Read Info}
659    then
660      begin
661      if NOT (ReadFile(Comm_Handle, Stat_Str, Stat_Length, NB_Read, lpOverLapped) AND
662             (NB_Read = Stat_Length))
663        then
664          begin
665         HandleCommError(true, 0, 'Self Test');
666          exit;
667          end
668        else
669          begin
670          HW_Rev     := Stat_Str[0];
671          Unit_ID    := Ord(Stat_Str[1]);
672
673          TResultA   := Ord(Stat_Str[2]);
674          TResultB   := Ord(Stat_Str[3]);
675
676
677          if (TResultA AND $01) = $00
678            then Label_AAL.Caption    := Fail
679            else Label_AAL.Caption    := Pass;
680
681          if (TResultA AND $02) = $00
682            then Label_ADL.Caption    := Fail
683            else Label_ADL.Caption    := Pass;
684
685          if (TResultA AND $04) = $00
686            then Label_ACL.Caption  := Fail
687            else Label_ACL.Caption  := Pass;
688
689          if (TResultA AND $08) = $00
690            then Label_AIL.Caption  := Fail
691            else Label_AIL.Caption  := Pass;
692
693          if (TResultA AND $10) = $00
694            then Label_RAB4.Caption    := Fail
695            else Label_RAB4.Caption    := Pass;
696
697          if (TResultA AND $20) = $00
698            then Label_RAB5.Caption    := Fail
699            else Label_RAB5.Caption    := Pass;
700
701          if (TResultA AND $40) = $00
702            then Label_RAB6.Caption  := Fail
703            else Label_RAB6.Caption  := Pass;
704
705          if (TResultA AND $80) = $00
```

```
706             then Label_RAB7.Caption  := Fail
707             else Label_RAB7.Caption  := Pass;
708
709           if (TResultB AND $01) = $00
710             then Label_RARW.Caption  := Fail
711             else Label_RARW.Caption  := Pass;
712
713           if (TResultB AND $02) = $00
714             then Label_RD.Caption    := Fail
715             else Label_RD.Caption    := Pass;
716
717           if (TResultB AND $04) = $00
718             then Label_EARW.Caption := Fail
719             else Label_EARW.Caption := Pass;
720
721           if (TResultB AND $08) = $00
722             then Label_ED.Caption   := Fail
723             else Label_ED.Caption   := Pass;
724
725           if (TResultB AND $10) = $00
726             then Label_RBB4.Caption  := Fail
727             else Label_RBB4.Caption  := Pass;
728
729           if (TResultB AND $20) = $00
730             then Label_RBB5.Caption    := Fail
731             else Label_RBB5.Caption    := Pass;
732
733           if (TResultB AND $40) = $00
734             then Label_RBB6.Caption := Fail
735             else Label_RBB6.Caption := Pass;
736
737           if (TResultB AND $80) = $00
738             then Label_RBB7.Caption   := Fail
739             else Label_RBB7.Caption   := Pass;
740           end;
741       end;
742   end;
743
744   {-------------------------------------------------------------}
745   procedure TForm_Main.FormCreate(Sender: TObject);
746
747   var
748     Idx     : Integer;
749     IdxStr,
750     ChanName,
751     UnitStr : String;
752
753   begin
754   Label_ChanData1_3.Caption  := '';          {Clear Channel Data 1-3}
755   Label_VoltData1_4.Caption  := '';          {Clear Voltage Data 1-4}
756
757   for Idx := 1 to 10 do
758     begin
759     Str(Idx : 2, IdxStr);                    {Convert Idx to Str ##}
760     UnitStr := ' Counts';
761     case  Idx of
762           1 :     ChanName := 'Rec. # ';
763           2 :     ChanName := 'Accel X';
764           3 :     ChanName := 'Accel Y';
765           4 :     ChanName := 'Accel Z';
766           5 :     ChanName := 'pH     ';
767           6 :     ChanName := 'Cond.  ';
768           7 :     ChanName := 'Temp.  ';
769           8 : begin
770                ChanName := 'Batt Mon  ';
771                UnitStr  := ' Volts';
772               end;
773           9 : begin
774                ChanName := '3V Reg Mon';
775                UnitStr  := ' Volts';
776               end;
```

```
777           10 : begin
778                  ChanName := '6V Reg Mon';
779                  UnitStr  := ' Volts';
780                end;
781       end;
782        {010618ed removed 'Chan' + IdxStr + & inserted ChanName}
783       Label_ChanData1_3.Caption  := Label_ChanData1_3.Caption
784                       + ChanName + ':  xxxx ' + UnitStr + CR;
785     end;
786     SpEdPrV   := SpinEdit_Read.Value;      {Intitialize SpinEdit Previous Value}
787     Edit_OutFile.Enabled    := False;       {Disable OutFile Box}
788     Button_Read.Enabled     := False;       {Disable Read Button}
789     Label_RW.Visible        := False;       {Hide Read/Write Label}
790     ProgressBar_RW.Visible  := False;       {Hide Read/Write Progress Bar}
791     OutFileValid            := False;       {OutFile Not Yet Specified}
792     Cmd_Char                := 'S';         {Set Cmd_Char to 'I'nitialize}
793     RadioButton_S.Checked   := True;
794     Label_Reply.Caption     := 'No Reply';  {Initialize Reply Label}
795
796     BaudRate := 19200;
797     OpenCommPort('COM1');
798     SetCommMODE(BaudRate,'N',8,1);
799     GetCommStatus;
800     FlushQueue(Receive);                        {010618ed added Empty Receive Buffer}
801     Group_PortStatus.Caption := 'COM' + IntToStr(CommStat.PortID);
802     Label_Status.Caption := 'BaudRate : ' + IntToStr(CommStat.BaudRate) + '    ' +
803                             'Parity : ' + (CommStat.Parity)             + '    ' +
804                             'ByteSize : ' + IntToStr(CommStat.ByteSize) + '    ' +
805                             'StopBits : ' + IntToStr(CommStat.StopBits);
806     end;
807
808     {--------------------------------------------------------------}
809     procedure TForm_Main.Button_XOutClick(Sender: TObject);
810
811     var
812       Idx     : Integer;
813       IdxStr  : String;
814       UnitStr : String;
815
816     begin
817     Label_ChanData1_3.Caption  := '';           {Clear Channel Data 1-3}
818     Label_VoltData1_4.Caption  := '';           {Clear Voltage Data 1-4}
819
820     for Idx := 1 to 3 do                        {Channel Data = Chan 1 - 3}
821       begin
822       Str(Idx : 2, IdxStr);                     {Convert Idx to Str ##}
823       Label_ChanData1_3.Caption  := Label_ChanData1_3.Caption +
824                                     'Chan ' + IdxStr + ':  xxxx Counts' + CR;
825       end;
826
827     for Idx := 1 to 4 do                        {Voltage Data = Mon 1 - 4}
828       begin
829       if Idx < 4
830         then UnitStr := ' Volts'
831         else UnitStr := ' °C';
832       Str(Idx : 2, IdxStr);                     {Convert Idx to Str ##}
833       Label_VoltData1_4.Caption := Label_VoltData1_4.Caption +
834                                    'Mon  ' + IdxStr + ':  xx.xx' + UnitStr + CR;
835       end;
836     Label_AAL.Caption  := '- X';
837     Label_ADL.Caption  := '- X';
838     Label_ACL.Caption  := '- X';
839     Label_AIL.Caption  := '- X';
840     Label_RAB4.Caption := '- X';
841     Label_RAB5.Caption := '- X';
842     Label_RAB6.Caption := '- X';
843     Label_RAB7.Caption := '- X';
844     Label_RARW.Caption := '- X';
845     Label_RD.Caption   := '- X';
846     Label_EARW.Caption := '- X';
847     Label_ED.Caption   := '- X';
```

```
848   Label_RBB4.Caption := '- X';
849   Label_RBB5.Caption := '- X';
850   Label_RBB6.Caption := '- X';
851   Label_RBB7.Caption := '- X';
852   end;
853
854   {-------------------------------------------------------------}
855   procedure TForm_Main.RadioButton_CommandClick(Sender: TObject);
856   begin
857   GainGroup.Visible := False;
858   Group_Readout.Visible := False;
859   if RadioButton_A.Checked
860     then Cmd_Char := 'A';      {Arm Package - RAM Only}
861   if RadioButton_B.Checked
862     then Cmd_Char := 'B';      {010607ed added baud command}
863   if RadioButton_C.Checked
864     then Cmd_Char := 'C';      {Cycle Package}
865   if RadioButton_P.Checked
866     then Cmd_Char := 'E';      {Erase Memory 010611ed changed to erase memory}
867   if RadioButton_I.Checked
868     then Cmd_Char := 'I';      {Initialize}
869   if RadioButton_R.Checked
870     then
871     begin
872     Cmd_Char := 'R';      {Read Data from RAM}
873     Group_Readout.Visible := True;
874     end;
875   if RadioButton_S.Checked
876     then Cmd_Char := 'S';      {Status Requested}
877   if RadioButton_T.Checked
878     then Cmd_Char := 'T';      {Self Test}
879   end;
880
881
882   procedure TForm_Main.Button1Click(Sender: TObject);
883   begin
884   ListHexIn.Clear;
885   end;
886
887
888   procedure TForm_Main.DebugDisplayOnClick(Sender: TObject);
889   begin
890   DebugOn := DebugDisplayOn.Checked;
891   end;
892
893   end.
894
```

Code Listing – About.pas

```
1   unit About;
2
3   interface
4
5   uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls,
6        Buttons, ExtCtrls;
7
8   type
9     TAboutBox = class(TForm)
10      Panel_Main  : TPanel;
11      OKButton    : TBitBtn;
12      ProgramIcon : TImage;
13      ProgramName : TLabel;
14      Version     : TLabel;
15      Date        : TLabel;
16      Copyright   : TLabel;
17      Comments    : TLabel;
18
19  private
20    { Private declarations }
21  public
22    { Public declarations }
23  end;
24
25  var
26    AboutBox: TAboutBox;
27
28  implementation
29
30  {$R *.DFM}
31
32  end.
33
34
```

Code Listing – ComUtil.pas

```
unit CommUtil;

{*****************************************************************************}
{*** CommUtil.PAS                                    Ver.    Date     Init. ***}
{*** Delphi 2.0 Communications Utility               1.00  12/15/96   (DJB) ***}
{*****************************************************************************}
{*** 010531ed changed from 256 to 80                    ***}

interface

uses Dialogs, Windows, SysUtils, MainUtil;

type
  QueueType    = (Transmit, Receive);      {Comm Port Queues}
  PString      = Array[0..80] of Char;     {Strings for PChar}
  CommStatRec  = Record
                   PortID     : Byte;      {Comm Port Number}
                   BaudRate   : LongInt;   {BuadRate}
                   Parity     : String;    {None, Odd, Even, Mark, Space}
                   ByteSize   : Byte;      {Bits Per Char : 4-8}
                   StopBits   : Byte;      {No. of Stop Bits : 1,2}
                 end;  {End of Record}

const
  InQueueSize  = 16384;                 {16K Input  Buffer}
  OutQueueSize = 16384;                 {16K Output Buffer}
  XonLim       = 2048;                  {COMM_DCB Values}
  XoffLim      = 2048;                  {COMM_DCB Values}
  AckChar = Chr($06);
  XonChar      = Chr($11);              {COMM_DCB Values}
  XoffChar     = Chr($13);              {COMM_DCB Values}
  NakChar = Chr($15);
  ErrorChar    = Chr($00);              {COMM_DCB Values}
  EofChar      = Chr($00);              {COMM_DCB Values}
  EvtChar      = Chr($00);              {COMM_DCB Values}
  ASCII_0      = Chr($30);              {ASCII 0 = 30H}
  Cntl_C       = Chr($03);              {Control C = 03H}
  CR           = Chr($0D);              {Carriage Return Character = 0DH}
  LF           = Chr($0A);              {Line Feed Character = 0AH}
  SyncByte1    = Chr($EB);              {First Sync Byte = EBH}
  SyncByte2    = Chr($90);              {Second Sync Byte = 90H}
  FiveSec      = 5000;                  {Five Seconds = 5000  MilliSec}
  ECHO_GOOD    = 'V';                   {Echo Verification Character}
  Block_Length = 560;                   {010531ed changed from 256 to 80 now to 560
010618ed}{Data Block Transfer Length}

var
  Comm_Handle  : THandle;
  CommStat     : CommStatRec;
  DCommStat    : TComStat;
  Comm_Err     : Boolean;
  Sync_OK      : Boolean;
{  Data_Block   : Array[0..Block_Length - 1] of Byte; {Byte Array for Data Storage}
  Data_Block   : Array[0..Block_Length + 5] of Byte; {added space for sync, block#, data, and
Cksm}
  Cmd_Char     : Char;
  Cmd_Echo     : Char;
  BaudRate     : DWord;
  lpOverlapped : POverlapped;
  lpSec_Attr   : PSecurityAttributes;

function WaitForCommData(NumBytes : Word; WaitTime : LongInt): Boolean;
function FindSync(Block_Cnt: WORD) : Boolean;

procedure DisplayDebug(Sent : Boolean; NumChars : Integer; DebugStr : Array of char);
procedure DisplayDebug2(Sent : Boolean; NumChars : Integer; DebugStr : Array of byte);
procedure HandleCommError(Clear : Boolean; Errors : DWord; Msg_Str: String);
procedure OpenCommPort(Com_ID: PString);
procedure CloseCommPort;
procedure SetCommMode(BaudRate: DWord; Parity: Char; ByteSize: Byte; StopBits: Byte);
procedure GetCommStatus;
```

```
72  procedure SyncError(Msg_Str : String);
73  procedure FlushQueue(TR_Queue: QueueType);
74  procedure SendChar(Chr: Char); {Send Single Character to Port}
75  procedure SendCommand(Cmd: Char); {Embed Char in Command String & Send to Port}
76  procedure ReadCommandEcho(var CmdEcho: Char); {Read Echoed Char From Comm Port}
77  procedure SendVerifyCommand(var CmdChar, CmdEcho : Char);
78  procedure GetQueueStatus; {Loads DCommStat.cbInQue & DCommStat.cbOutQue}
79  procedure GetDataBlock;
80  procedure GetStatusInfo(var Num_Data_Chans, Num_Mon_Chans : Byte);
81
82  {<f>}
83  implementation
84
85  uses Main2Form;
86
87  {***** Local Error Handling Procedures *****}
88
89  {-------------------------------------------------------------------------}
90  procedure FatalError(Err_Str: String);
91
92  begin
93  MessageDlg('A FATAL Error has occurred while ' + Err_Str + CR + CR +
94             'Program will be Aborted!', mtError, [mbOK],0);
95  Halt;
96  end;
97
98  {-------------------------------------------------------------------------}
99  procedure HandleCommError(Clear : Boolean; Errors : DWord; Msg_Str : String);
100
101 var
102   ErrorFlags  : DWord;
103   Stats       : PComStat;
104
105 begin
106 Comm_Err := True;
107 if (Clear) then
108   begin
109   if NOT ClearCommError(Comm_Handle, ErrorFlags, Stats) then
110     FatalError('attempting to clear a comm error.');
111     Exit;
112   end
113 else
114   ErrorFlags := Errors;
115
116 if (ErrorFlags <> 0) then
117   begin
118   if (ErrorFlags AND CE_RXOVER) = CE_RXOVER
119     then Msg_Str := Msg_Str + 'Receive queue overflow.';
120   if (ErrorFlags AND CE_OVERRUN) = CE_OVERRUN
121     then Msg_Str := Msg_Str + 'Receive overrun.';
122   if (ErrorFlags AND CE_RXPARITY) = CE_RXPARITY
123     then Msg_Str := Msg_Str + 'Receive parity error.';
124   if (ErrorFlags AND CE_FRAME) = CE_FRAME
125     then Msg_Str := Msg_Str + 'Receive framing error.';
126   if (ErrorFlags AND CE_BREAK) = CE_BREAK
127     then Msg_Str := Msg_Str + 'Break detected.';
128   if (ErrorFlags AND CE_TXFULL) = CE_TXFULL
129     then Msg_Str := Msg_Str + 'Transmission queue overflow.';
130   if (ErrorFlags AND CE_PTO) = CE_PTO
131     then Msg_Str := Msg_Str + 'LPTx Timeout.';
132   if (ErrorFlags AND CE_IOE) = CE_IOE
133     then Msg_Str := Msg_Str + 'LPTx I/O Error.';
134   if (ErrorFlags AND CE_DNS) = CE_DNS
135     then Msg_Str := Msg_Str + 'LPTx Device not selected.';
136   if (ErrorFlags AND CE_OOP) = CE_OOP
137     then Msg_Str := Msg_Str + 'LPTx Out-Of-Paper.';
138   if (ErrorFlags AND CE_MODE) = CE_MODE
139     then Msg_Str := Msg_Str + 'Requested mode unsupported.';
140   if (Msg_Str <> '') then
141 {    MessageDlg('COM Device Error.' + CR + CR + Msg_Str, mtError,[mbOK],0);
142 }
```

Code Listing – ComUtil.pas

```
143     Form_Main.ListHexIn.Items.Add(Msg_Str);
144
145     end;
146   end;
147
148   {--------------------------------------------------------------------}
149   procedure TimeOutError(Msg_Str : String);
150
151   begin
152   Comm_Err := True;
153   {MessageDlg('A read timeout has occurred while waiting for a ' + Msg_str +
154               '.', mtWarning, [mbOK], 0);
155   }
156     Msg_Str := 'Read timeout waiting for a ' + Msg_str +  '.' ;
157     Form_Main.ListHexIn.Items.Add(Msg_Str);
158   end;
159
160   {--------------------------------------------------------------------}
161   procedure SyncError(Msg_Str : String);
162
163   begin
164   {MessageDlg('Sync Error :' + CR + CR +  Msg_Str, mtError, [mbOK], 0);
165   }
166     Msg_Str := 'Sync Error :' + CR + CR +  Msg_Str;
167     Form_Main.ListHexIn.Items.Add(Msg_Str);
168   end;
169
170   {--------------------------------------------------------------------}
171   procedure OpenCommPort(Com_ID: PString);
172                                   {Valid ID's = 'COM1', 'COM2', 'COM3', 'COM4'}
173   var
174     PCom_ID : PChar;
175
176   begin
177   PCom_ID := @Com_ID;                             {Set Pointer @ Com_ID String}
178   Comm_Handle := CreateFile(PCom_ID,              {Get Handle for Specified Port}
179                 GENERIC_READ OR GENERIC_WRITE,    {Access (read-write) mode}
180                 0,                                {Prevents port from being shared}
181                 lpSec_Attr,                       {Pointer to Security Attributes}
182                 OPEN_EXISTING,                    {How to Create}
183                 FILE_ATTRIBUTE_NORMAL,            {File Attributes Normal}
184                 0);                               {Fails if Not Zero}
185   if (Comm_Handle = INVALID_HANDLE_VALUE)
186     then FatalError('opening ' + Com_ID + '.');
187   if NOT(SetupComm(Comm_Handle, InQueueSize, OutQueueSize))
188     then FatalError('initializing ' + Com_ID);
189   CommStat.PortID := StrToInt(Com_ID[3]);
190   {lpOverlapped    := @NULL; }                     {Set lpOverlapped Variable}
191   lpOverlapped    := NIL;                        {Set lpOverlapped Variable}
192   end;
193
194   {--------------------------------------------------------------------}
195   procedure CloseCommPort;
196
197   begin
198   if NOT(CloseHandle(Comm_Handle))
199     then MessageDlg('Error Closing Communications Port COM' +
200                     IntToStr(CommStat.PortID) + '.', mtError,[mbOK], 0);
201   end;
202
203   {--------------------------------------------------------------------}
204   procedure SetCommMode(BaudRate: DWord; Parity: Char; ByteSize: Byte; StopBits: Byte);
205
206   {Flag Definition (*) = Flag is Set     COMM_DCB.Flags = $3011 is the Default}
207
208   {* fBinary           = $00000001 {Must Be Set to True}
209   {  fParity           = $00000002 {If Set Parity Checking is Performed}
210   {  fOutxCtsFlow      = $00000004 {If Set CTS is Monitored for Flow Control}
211   {  fOutxDsrFlow      = $00000008 {If Set DSR is Monitored for Flow Control}
212
213   {* fDtrControl_Dis   = $00000000 {Disables DTR Line when Device is Opened}
```

```
214   {  fDtrControl_Ena   = $00000010 {Enables  DTR Line when Device is Opened}
215   {  fDtrControl_Hsk   = $00000020 {Enables  DTR Handshaking}
216   {  fDsrSensitivity   = $00000040 {If Set Data is Ignored until DSR Line High}
217   {  fTXContinueOnXOff = $00000080 {If Set TX Continues After Sending Xoff}
218
219   {  fOutX             = $00000100 {If Set Xon/Xoff controls affect TX}
220   {* fInX              = $00000200 {If Set Xon/Xoff controls sent during RX}
221   {  fErrorChar        = $00000400 {If Set and fParity Error Bytes are Replaced}
222   {  fNull             = $00000800 {If Set NULL Bytes are Discarded}
223
224   {  fRtsControl_Dis   = $00000000 {Disables RTS Line when Device is Opened}
225   {  fRtsControl_Ena   = $00001000 {Enables  RTS Line when Device is Opened}
226   {  fRtsControl_Hsk   = $00002000 {Enables  RTS Handshaking}
227   {  fRtsControl_Tog   = $00003000 {RTS High when Byte in TX buff, Low when empty}
228   {  fAbortOnError     = $00004000 {If Set Rx/Tx Terminated on Error}
229
230   {  fDummy2           = $FFFF8000;    {Reserved DO NOT USE}
231
232   var
233     Comm_DCB     : TDCB;
234
235   begin
236   if NOT(GetCommState(Comm_Handle, COMM_DCB))    {Get Comm Port Parameters}
237     then FatalError('retrieving the current port configuration.');
238   COMM_DCB.BaudRate  := BaudRate;                {Modify Desired Fields}
239   Case Parity of
240     'N','n' : COMM_DCB.Parity    := NOPARITY;
241     'O','o' : COMM_DCB.Parity    := ODDPARITY;
242     'E','e' : COMM_DCB.Parity    := EVENPARITY;
243     'M','m' : COMM_DCB.Parity    := MARKPARITY;
244     'S','s' : COMM_DCB.Parity    := SPACEPARITY;
245     else
246       begin
247       Parity := 'N';
248       COMM_DCB.Parity := NOPARITY;
249       end;
250     end; {Case}
251   COMM_DCB.ByteSize   := ByteSize;
252   Case StopBits of
253     1 : COMM_DCB.StopBits  := ONESTOPBIT;
254     2 : COMM_DCB.StopBits  := TWOSTOPBITS;
255     else
256       COMM_DCB.StopBits    := ONESTOPBIT;
257     end; {Case}
258   if (Parity = 'N') OR (Parity = 'n')
259     then COMM_DCB.Flags     := $0201   {Parity Checking is Not Performed}
260     else COMM_DCB.Flags     := $0203;  {Parity Checking is Performed}
261   COMM_DCB.XonLim    := XonLim;
262   COMM_DCB.XoffLim   := XoffLim;
263   COMM_DCB.XonChar   := XonChar;
264   COMM_DCB.XoffChar  := XoffChar;
265   COMM_DCB.ErrorChar := ErrorChar;
266   COMM_DCB.EofChar   := EofChar;
267   COMM_DCB.EvtChar   := EvtChar;
268   if NOT(SetCommState(Comm_Handle, COMM_DCB))    {Set Comm Port to New Parameters}
269     then FatalError('setting port configuration.');
270   end;
271
272   {--------------------------------------------------------------------}
273   procedure GetCommStatus;
274
275   var
276     Comm_DCB     : TDCB;
277     p_TDCB       : PDCB;
278
279   begin
280   if NOT(GetCommState(Comm_Handle, COMM_DCB))    {Get Comm Port Parameters}
281     then FatalError('trying to retrieve the current port configuration.');
282   p_TDCB := @COMM_DCB;
283   CommStat.BaudRate := p_TDCB.BaudRate;
284   Case p_TDCB.Parity of
```

```
285       NOPARITY    : CommStat.Parity := 'None';
286       ODDPARITY   : CommStat.Parity := 'Odd';
287       EVENPARITY  : CommStat.Parity := 'Even';
288       MARKPARITY  : CommStat.Parity := 'Mark';
289       SPACEPARITY : CommStat.Parity := 'Space';
290       end; {Case}
291     CommStat.ByteSize := p_TDCB.ByteSize;
292     Case p_TDCB.StopBits of
293       ONESTOPBIT   : CommStat.StopBits := 1;
294       TWOSTOPBITS  : CommStat.StopBits := 2;
295       end; {Case}
296     end;
297
298     {--------------------------------------------------------------------}
299     procedure FlushQueue(TR_Queue: QueueType);
300
301     begin
302     Case TR_Queue of
303       Transmit : if NOT (PurgeComm(Comm_Handle, PURGE_TXCLEAR))
304                    then HandleCommError(true, 0, 'Purge Queue Transmit');
305       Receive  : if NOT (PurgeComm(Comm_Handle, PURGE_RXCLEAR))
306                    then HandleCommError(true, 0, 'Purge Queue Receive');
307       end; {Case}
308     end;
309
310     function WaitForCommData(NumBytes : Word; WaitTime : LongInt): Boolean;
311
312     var
313       Entry_Time,
314       Loop_Time    : LongInt;
315
316     begin
317     Entry_Time    := GetTickCount;     {Get Loop Entry Time}
318     Repeat
319       GetQueueStatus;
320       Loop_Time := GetTickCount - Entry_Time;
321     Until (DCommStat.cbInQue >= NumBytes) OR (Loop_Time > WaitTime);
322     if (DCommStat.cbInQue >= NumBytes)
323       then WaitForCommData := True
324       else WaitForCommData := False;
325     end;
326
327     {--------------------------------------------------------------------}
328     procedure SendChar(Chr: Char); {Send Single Character to Port}
329
330     const
331       Chr_Length = 1;
332
333     var
334       Chr_Str     : Array[0..Chr_Length] of Char;
335       NB_Written : DWord;
336
337     begin
338     Chr_Str[0]    := Chr;
339     Chr_Str[1]    := #0;
340     if NOT (WriteFile(Comm_Handle, Chr_Str, Chr_Length, NB_Written, lpOverLapped) AND
341           (NB_Written = Chr_Length))
342       then HandleCommError(true, 0, 'Character sent to port');
343     DisplayDebug(true, NB_Written,  Chr_Str);
344     end;
345
346     {--------------------------------------------------------------------}
347     procedure SendCommand(Cmd: Char); {Embed Char in Command String & Send to Port}
348
349     const
350       Cmd_Length = 4;                {Command String consist of 4 Bytes}
351
352     var
353       Cmd_Str     : Array[0..Cmd_Length] of Char;
354       NB_Written : DWord;
355
```

```
356  begin
357  Cmd_Str[0]    := ASCII_0;
358  Cmd_Str[1]    := Cmd;
359  Cmd_Str[2]    := CR;
360  Cmd_Str[3]    := LF;
361  Cmd_Str[4]    := #0;
362  if NOT (WriteFile(Comm_Handle, Cmd_Str, Cmd_Length, NB_Written, lpOverLapped) AND
363         (NB_Written = Cmd_Length))
364    then HandleCommError(true, 0, 'Command sent to port');
365  DisplayDebug(true, NB_Written, Cmd_Str);
366  end;
367
368  {--------------------------------------------------------------------}
369  procedure ReadCommandEcho(var CmdEcho: Char); {Read Echoed Char From Comm Port}
370
371  const
372    Echo_Length = 4;              {Echo String consist of 4 Bytes}
373    HalfSec = 500;  {500 Milliseconds}
374  var
375    Echo_Str    : Array[0..Echo_Length] of Char;
376    NB_Read     : DWord;
377
378  begin
379    CmdEcho := '?';             {Error Flag}
380    if WaitForCommData(Echo_Length, HalfSec) then {If Characters in Rec Queue - Read Echo}
381      begin
382        if NOT (ReadFile(Comm_Handle, Echo_Str, Echo_Length, NB_Read, lpOverLapped)
383            AND (NB_Read = Echo_Length)) then
384          begin
385            HandleCommError(true, 0, 'Command Echo');
386            Exit;
387          end;
388        DisplayDebug(false, NB_Read, Echo_Str);
389        if (Echo_Str[0] = ASCII_0) then
390          CmdEcho := Echo_Str[1]
391        else
392          MessageDlg('Command echo format is incorrect.', mtError,[mbOK], 0);
393      end
394    else
395      TimeOutError('Command Echo');
396  end;
397
398  {--------------------------------------------------------------------}
399  procedure DisplayDebug(Sent : Boolean; NumChars : Integer; DebugStr : Array of char);
400
401  var
402    Index : Integer;
403    DisplayStr : String;
404
405  begin
406    if DebugOn then
407      begin
408        if (Sent) then
409          DisplayStr := 'Sent '
410        else
411          DisplayStr := ' ';
412        for Index := 0 to NumChars - 1 do
413          begin
414            DisplayStr := DisplayStr + IntToHex(Ord(DebugStr[Index]), 2) + ' ';
415            if (((Index + 1) MOD 20) = 0) then
416              begin
417                Form_Main.ListHexIn.Items.Add(DisplayStr);
418                DisplayStr := ' ';
419              end;
420          end;
421        if (DisplayStr <> ' ') then
422           Form_Main.ListHexIn.Items.Add(DisplayStr);
423      end;
424  end;
425
426  {--------------------------------------------------------------------}
```

```
427  procedure DisplayDebug2(Sent : Boolean; NumChars : Integer; DebugStr : Array of byte);
428
429  var
430    Index : Integer;
431    DisplayStr : String;
432
433  begin
434    if DebugOn then
435      begin
436        if (Sent) then
437          DisplayStr := 'Sent '
438        else
439          DisplayStr := ' ';
440        for Index := 0 to NumChars - 1 do
441          begin
442            DisplayStr := DisplayStr + IntToHex(Ord(DebugStr[Index]), 2) + ' ';
443            if (((Index + 1) MOD 20) = 0) then
444              begin
445                Form_Main.ListHexIn.Items.Add(DisplayStr);
446                DisplayStr := ' ';
447              end;
448          end;
449        if (DisplayStr <> ' ') then
450          Form_Main.ListHexIn.Items.Add(DisplayStr);
451      end;
452  end;
453
454  {---------------------------------------------------------------------}
455  procedure SendVerifyCommand(var CmdChar, CmdEcho : Char);
456
457  var
458    Bit_Bucket : Char;
459
460  begin
461    FlushQueue(Transmit);                {Clear Transmit Buffer}
462    FlushQueue(Receive);                 {Clear Receive Buffer}
463    SendCommand(CmdChar);
464    ReadCommandEcho(CmdEcho);
465    if (CmdChar <> CmdEcho) then
466      begin
467        MessageDlg('COMMAND Echo Error.' + CR + CR +
468          'Command CANCELLED!', mtWarning, [mbOK], 0);
469        Exit;
470      end;
471    SendCommand(ECHO_GOOD);              {Send Verification Character}
472    ReadCommandEcho(Bit_Bucket);         {Verify Echo sent to bit bucket}
473  end;
474
475  {---------------------------------------------------------------------}
476  procedure GetQueueStatus; {Loads DCommStat.cbInQue & DCommStat.cbOutQue}
477
478  var
479    Errors  : DWord;
480    Status  : PComStat;
481
482  begin
483  Status := @DCommStat;
484  if NOT(ClearCommError(Comm_Handle, Errors, Status))
485    then
486      FatalError('getting queue status.');
487  if Errors <> 0
488    then
489      HandleCommError(false, Errors, 'Get Queue Status ');
490  end;
491
492  {---------------------------------------------------------------------}
493  function FindSync(Block_Cnt: WORD) : Boolean;
494                                       {Sync [$EB] [$90] [Blk_Cnt_Hi] [Blk_Cnt_Lo]}
495  const
496    Sync_Length = 4;                   {Sync String consist of 4 Bytes}
497
```

```
498  var
499    Sync_Str    : Array[0..Sync_Length] of Char;
500    NB_Read     : DWord;
501    Read_Cnt    : Word;
502  {  Index      : Integer;
503    DisplayStr : String;
504  }
505  begin
506  FindSync := False;
507
508  {If Characters in Rec Queue - Read Sync}
509
510  if WaitForCommData(Sync_Length, FiveSec) then
511    begin
512    if NOT (ReadFile(Comm_Handle, Sync_Str, Sync_Length, NB_Read, lpOverLapped) AND
513          (NB_Read = Sync_Length)) then
514      begin
515      CloseFile(OutFileB);                              {Close Binary Output File}
516      HandleCommError(true, 0, 'Find Sync');
517      Exit;
518      end;
519
520    DisplayDebug(false, NB_Read, Sync_Str);
521
522
523    if (Sync_Str[0] <> SyncByte1) OR (Sync_Str[1] <> SyncByte2) then
524      begin
525  {=        SyncError('Incorrect Sync Characters [$EB90] Not Found.');}
526      if DebugOn then Form_Main.ListHexIn.Items.Add('Sync Error, Wrong sync characters');
527      FindSync := False;
528      Exit;
529      end;
530    Read_Cnt := (Ord(Sync_Str[3]) * 256) + Ord(Sync_Str[2]);
531    if DebugOn then Form_Main.ListHexIn.Items.Add('SensorBall sending block ' +
532  IntToStr(Read_Cnt));
533    if Read_Cnt = Block_Cnt then
534      FindSync := True
535    else
536      begin
537      SyncError('Incorrect Block Count.' + CR +
538                'Block Number = ' + IntToStr(Block_Cnt) + CR +
539                'Block ID     = ' + IntToStr(Read_Cnt));
540      Exit;
541      end
542    end
543  else
544    TimeOutError('Sync Code');
545  end;
546
547  {-------------------------------------------------------------------------}
548  procedure GetDataBlock;
549
550  var
551    NB_Read      : DWord;
552
553  begin
554  if WaitForCommData(Block_Length, FiveSec) then {If Characters in Rec Queue - Read Block}
555    begin
556    if NOT (ReadFile(Comm_Handle, Data_Block, Block_Length, NB_Read, lpOverLapped)
557          AND (NB_Read = Block_Length)) then
558      begin
559      CloseFile(OutFileB);                              {Close Binary Output File}
560      HandleCommError(true, 0, 'Get Data Block');
561      Exit;
562      end;
563    DisplayDebug2(false, 20, Data_Block);  {Just list first 20 bytes in debug mode}
564    end
565  else
566    begin
567    TimeOutError('Data Block');
568    Exit;
```

```
569       end
570     end;
571
572     {-------------------------------------------------------------------------}
573     procedure GetStatusInfo(var Num_Data_Chans, Num_Mon_Chans : Byte);
574
575     const
576       Stat_Length  = 2;                    {Status String consist of 2 Bytes}
577
578     var
579       NB_Read      : DWord;
580       Stat_Str     : Array[0..Stat_Length] of Char;
581
582     begin
583     Num_Data_Chans := 0;
584     Num_Mon_Chans  := 0;
585
586     {If Characters in Rec Queue - Read Info}
587     if WaitForCommData(Stat_Length, FiveSec) then
588       begin
589       if NOT (ReadFile(Comm_Handle, Stat_Str, Stat_Length, NB_Read, lpOverLapped) AND
590              (NB_Read = Stat_Length)) then
591         begin
592         HandleCommError(true, 0, 'Get Status Info');
593         Exit;
594         end
595       else
596         begin
597         DisplayDebug(false, NB_Read, Stat_Str);
598         Num_Data_Chans := Ord(Stat_Str[0]);
599         Num_Mon_Chans  := Ord(Stat_Str[1]);
600         if Num_Data_Chans > 16 then              {Max Data Channels = 16}
601           Num_Data_Chans := 16;
602         if Num_Mon_Chans > 8 then               {Max Mon Channels = 8}
603           Num_Mon_Chans := 9;
604         end;
605       end
606     else
607       begin
608       TimeOutError('Status Information');
609       Exit;
610       end
611     end;
612
```

Code Listing – MainUtil.pas

```
unit MainUtil;

{*************************************************************************}
{*** <t> MainUtil.PAS - WinLook Main Utilities Unit        Delphi V2.00 ***}
{***                                                                   ***}
{*** <t> 1.00    03-28-96   Original Version                    (DJB) ***}
{*************************************************************************}

interface      { Public Declarations }

Uses  Dialogs, SysUtils, WinProcs;

type Float = Real;

{*** Global Constants ***}

const
  CR                      = Chr($0D);  {Carriage Return}
  LF                      = Chr($0A);  {Line Feed}
  MaxDataChans            = 32;        {Maximum Data Channels Allowed}
  MaxMuxChans             = 32;        {Maximum Multiplexer Channels Allowed}
  MaxSubChans             = 4;         {Maximum SubComm Channels Allowed}
  MaxMuxChansPerDataChan  = 16;        {Maximum Multiplexer Chans/Data Channel}
  MaxSubChansPerDataChan  = 2;         {Maximum SubComm Chans/Data Channel}
  BreakChar               : set of CHAR = [' ',',',':',';','/','=',#09];
  ScaleSet                : set of CHAR = ['C','c','G','g','U','u','V','v'];
  SizeOfData              : set of CHAR = ['B','b','W','w']; {B - Byte, W - Word}
{<f>}

{*** Global Types ***}

type
  BYTERECORD     = array[0..MaxMuxChans-1]  of BYTE;
  CHARSET        = set of CHAR;
{<f>}
{*** Global Variables ***}

var
  OutFileB       : File;                      {Binary Output File}
  OutFileName    : String;                    {Output File Path and Name}
  FileExt        : String;                    {File Extention}
  ByteRec        : ByteRecord;
  SSR            : LongInt;                   {Speed Shift Record}
  Err_Code       : Integer;                   {Error Code Return Variable}
  KB_Req         : Word;                      {K-Bytes Required for File}
  ChanNum        : Integer;                   {Channel Number to Process}
  Data_Current   : Boolean;                   {Read DataRecord is Current}
  Byte_Record    : Array[0..8388608]  of Byte;   {8meg - Byte Record}

{<f>}
{***** Procedure Specifications *****}
procedure InitializeGlobals;                      {Initialize Global Varaibles}
procedure Strip(var Str : String; var StrLen: INTEGER; Break : CHARSET);
procedure Parse(var Str, Aword : String; Break : CHARSET);
procedure GetUnitStr(var Str, UnString : String; Break: CHARSET);
procedure GetNumStr (var Str, NumString : String; Break: CHARSET);
procedure Delay(mSec : LONGINT); {Delay Procedure Resolution limited to 55 mSec}

{***** File Utilities *****}
function  GetFileDT(FileName : String) : TDATETIME;
procedure SetFileDT(FileName : String ; FileDT : TDateTime);
function  OpenOutFile(var OutFile : FILE; OutFileName : String;
                      RecSize: WORD ; KB_Required : WORD) : INTEGER;
{<f>}
implementation

procedure InitializeGlobals;

begin
Data_Current := False;
SSR       := 0;
```

```
72    Err_Code  := 0;
73    KB_Req    := 0;
74    ChanNum   := 0;
75    end;
76
77    {<f>}
78    {***** String Routines *****}
79
80    {***** Remove Leading Break Characters from String *****}
81    procedure Strip(var Str : String; var StrLen: INTEGER; Break : CHARSET);
82
83    var
84      Idx : Integer;
85
86    begin
87    StrLen := Length(Str);
88    if StrLen > 0 then
89      begin
90      Idx := 0;
91      while (Str[Idx+1] in Break) and (Idx < StrLen) do
92        Idx := Idx + 1;
93      Delete (Str,1,Idx);
94      StrLen:= StrLen-Idx
95      end;
96    end;
97
98    {***** Parse a Word from a String *****}
99    procedure Parse(var Str, Aword : String; Break : CHARSET);
100
101   var
102     StrLen,
103     Idx   : Integer;
104
105   begin
106   Aword:= '';
107   Strip (Str,StrLen,Break);
108   if StrLen = 0
109     then Exit;
110   Idx:= 0;
111   while not (Str[Idx+1] in Break) and (Idx < StrLen) do
112     Idx:= Idx+1;
113   Aword:= Copy(Str,1,Idx);
114   Delete (Str,1,Idx);
115   Strip (Str,StrLen,Break)
116   end;
117
118   {***** Get Unit Character from String *****}
119   procedure GetUnitStr(var Str, UnString : String; Break: CHARSET);
120
121   const
122     UnitSet : Set of Char = ['a'..'z','A'..'Z'];
123
124   begin
125   UnString:= ' ';
126   while (Length(Str)>0) and not (UnString[1] in UnitSet) do
127     Parse (Str,UnString,Break)
128   end;
129
130   {***** Get Next Number String from String *****}
131   procedure GetNumStr (var Str, NumString : String; Break: CHARSET);
132
133   const
134     NumSet : Set of Char = ['-','.','0'..'9'];
135
136   begin
137   NumString:= ' ';
138   while (Length(Str)>0) and not (NumString[1] in NumSet) do
139     Parse (Str,NumString,Break)
140   end;
141
142   (*** STRING ENDS ***)
```

```
143
144   {<f>}
145   procedure Delay(mSec : LONGINT); {Delay Procedure Resolution limited to 55 mSec}
146
147   var
148     EntryTime,
149     NowTime     : LongInt;
150
151   begin
152   EntryTime := GetTickCount;       {Number of milliSeconds Windows Running}
153   Repeat
154     NowTime := GetTickCount;
155   until (NowTime - EntryTime) > mSec;
156   end;
157
158   {<f>}
159   {***** File Utilities *****}
160   function GetFileDT(FileName : String) : TDATETIME;
161
162   var
163     DOS_FileDT : LongInt;     {DOS    - Date/Time Format}
164     DateTime   : TDateTime;   {Delphi - Date/Time Format}
165
166   begin
167   DOS_FileDT := FileAge(FileName);
168   DateTime   := FileDateToDateTime(DOS_FileDT);
169   GetFileDT  := DateTime;
170   end;
171
172   procedure SetFileDT(FileName : String ; FileDT : TDateTime);
173
174   var
175     File_Handle : Integer;    {Windows File Handle}
176     DOS_FileDT  : LongInt;    {DOS    - Date/Time Format}
177
178   begin
179   DOS_FileDT  := DateTimeToFileDate(FileDT);
180   File_Handle := FileOpen(FileName, Of_Share_Compat);
181   FileSetDate(File_Handle, DOS_FileDT);
182   FileClose(File_Handle);
183   end;
184
185   {<f>}
186   function OpenOutFile(var OutFile : FILE; OutFileName : String;
187                            RecSize: WORD ; KB_Required : WORD) : INTEGER;
188
189   var FilePath      : String;
190       DriveLetter   : Char;
191       BytesAvail,
192       BytesRequired : Int64;
193
194   begin
195   FilePath      := ExtractFilePath(ExpandFileName(OutFileName));
196   DriveLetter   := UpCase(FilePath[1]);
197   BytesAvail    := DiskFree(Ord(DriveLetter) - $40);
198   BytesRequired := KB_Required * 1024;
199   if BytesRequired > BytesAvail
200     then
201       begin
202       MessageDlg('Insufficient disk space on drive : ' + DriveLetter + '.',
203                  mtWarning, [mbOK], 0);
204       OpenOutFile := -1                                {Return Error Code}
205       end
206     else
207       begin
208       AssignFile(OutFile, OutFileName);
209       Rewrite(OutFile, RecSize);
210       OpenOutFile := 0;                                {No Error}
211       end;
212   end;
213
```

```
214   (*** FILE ENDS ***)
215
216   begin
217   end.
218
219
220   ; Routines to interpret and execute commands sent to the P&G Sensor Ball
221
222
223
```

Distribution

(10 Copies) T. Michael Rothgeb
The Procter & Gamble Company
5299 Spring Grove Avenue
Cincinnati, OH 45217

	MS0487	Tedd A. Rohwer, 2121
	MS0986	Randal R. Lockhart, 2665
(5 Copies)	MS0986	Michael E. Partridge, 2665
	MS0790	Dennis J. Wilder, 5851
	MS0986	David L. Faucett, 2665
	MS0986	John F. Heise II, 2665
	MS0986	Edward Henry, 2665
	MS0986	Felipe V. Reyes, 2665
	MS0986	Antonio Mittas, 2665
	MS0986	Vincent P. Salazar, 2660
	MS0986	Lorraine S. Baca, 2661
	MS0986	Larry J. Dalton, 2662
	MS0986	Robert J Longoria,
	MS0986	Ronald J. Franco, 2664
	MS0986	Jay B. Vinson, 2666
	MS0529	Bruce C. Walker, 2600
	MS1380	Victor Weiss, 1323
	MS1425	Carol I. Ashby, 1744
	MS9951	Andrew W. Walker, 8130
	MS9951	A. William Flounders, 8130
	MS9018	Central Technical Files, 8945-1
(2 Copies)	MS0899	Technical Library, 9616
	MS0612	Review & Approval Desk, 9612 for DOE/OSTI
	MS1380	CRADA Administration, 1323
	MS0161	Patent and Licensing Office, 11500