A Microprocessor Based Air Pressure Controller

by

Gregory T. Hulan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

APPROVED:

Dr. S. F. Midkiff, Chairman

Dr. C. E. Nunnally

Dr. J. G. Tront

May 1989
Blacksburg, Virginia
A Microprocessor Based Air Pressure Controller

by

Gregory Thomas Hulan

Dr. Scott Midkiff, Committee Chairman

Electrical Engineering

(ABSTRACT)

A microprocessor based air pressure controller is discussed. The particular implementation was designed around an existing pressure measurement, and display unit. The unit is controlled by a 6809 microprocessor. It is shown that due to the many functions that the unit must perform and control, a microprocessor based system is a good choice. The controller is economical since it uses standard chips, yet it is very accurate since it uses state of the art pressure transducers. Experimental results and the user friendly interface will also be discussed. A commented listing of the controller software, and the circuit diagrams are appended.
Acknowledgements

I would like to express my deepest gratitude to Dr. Scott Midkiff, Doug Juanarena, Dan Ridenour, and Scott Love. Without their financial and technical support, this thesis would not have been possible. Doug Juanarena, president of Pressure Systems Inc., provided financial support while Dan Ridenour and Scott Love offered technical support. Dr. Scott Midkiff provided helpful suggestions throughout the development stage of this thesis as well as provided unparalleled assistance revising the thesis.

I wish to express my appreciation to my committee members, Dr. C. E. Nunnally and Dr. J. G. Tront for their support.

With much love and gratitude, I thank my parents and sister for always being there when I need them. This thesis is dedicated to them.
# Table of Contents

**Introduction**  
1

**Air Pressure**  
2.1 Pressure Measurement  
4
2.2 Solartron™ Absolute Pressure Transducer  
6
2.3 S100™ Differential Pressure Transducer  
9

**The Sonix™ Display**  
10

**The Controller: Design Requirements**  
4.1 Functional Requirements  
17
4.2 Control Requirements  
19
4.3 Speed Requirements  
21

**Function and Design of the Pressure Controller**  
5.1 Functional Description of the PCU  
23
5.2 PCU Hardware  
24
5.2.1 Digital Part of the PCU  
27
5.2.2 Valves  
31
5.2.3 Analog Part of the PCU  
38
5.3 Software  
46
5.3.1 PCU Initialization Software  
46
5.3.2 Build Tables Routine (BLD TBL)  
49
5.3.3 User-defined Output Pressure Limits (LIMITS)  
58
5.3.4 Set Pressure Tolerance (TOLERANCE)  
61
5.3.5 Set Output Pressure (SET PRES)  
64
5.3.6 Update Output Pressure  
70
5.3.7 Stop and Vent Pressure (STOP & V)  
82
5.3.8 Stop and Clamp Pressure (STOP & C)  
83

**Analysis of Results**  
86
6.1 Simulation  
86
6.1.1 Simulation Model  
87
6.1.2 Simulation Results  
96
6.2 Did the Design Meet the Requirements?  
104

**Summary and Conclusions**  
107
7.1 Summary of this Work  
107

Table of Contents  
iv
Table of Contents

7.2 Conclusions about Microprocessor Control for the PCU 108
7.3 Suggested Design Enhancements 109

Bibliography 111
Hardware Schematic 112
Software Listing 116
Vita 186
List of Illustrations

<table>
<thead>
<tr>
<th>Figure</th>
<th>Illustration</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Basic Pilot-tube method of sensing static, dynamic, and total pressure.</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>Sonix™ Display</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Sonix™ Display front panel</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>Display configuration menu tree</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>Damping conditions</td>
<td>20</td>
</tr>
<tr>
<td>5.1</td>
<td>Control function</td>
<td>25</td>
</tr>
<tr>
<td>5.2</td>
<td>PCU menu tree</td>
<td>26</td>
</tr>
<tr>
<td>5.3</td>
<td>Pressure controller block diagram</td>
<td>28</td>
</tr>
<tr>
<td>5.4</td>
<td>PAL pin out</td>
<td>30</td>
</tr>
<tr>
<td>5.5</td>
<td>PAL program</td>
<td>31</td>
</tr>
<tr>
<td>5.6</td>
<td>PAL simulation results</td>
<td>32</td>
</tr>
<tr>
<td>5.7</td>
<td>PCU hardware identification circuit</td>
<td>34</td>
</tr>
<tr>
<td>5.8</td>
<td>Jumper settings for all PCU pressure ranges</td>
<td>35</td>
</tr>
<tr>
<td>5.9</td>
<td>Schematic of PCU pneumatics</td>
<td>36</td>
</tr>
<tr>
<td>5.10</td>
<td>Valve-driver circuitry</td>
<td>39</td>
</tr>
<tr>
<td>5.11</td>
<td>HSC valve driver circuitry</td>
<td>41</td>
</tr>
<tr>
<td>5.12</td>
<td>Response of the analog circuit with respect of time</td>
<td>43</td>
</tr>
<tr>
<td>5.13</td>
<td>HSC valve as the current input changes</td>
<td>44</td>
</tr>
<tr>
<td>5.14</td>
<td>PCU related information stored in battery-backed RAM</td>
<td>47</td>
</tr>
<tr>
<td>5.15</td>
<td>Typical graph of S100™ characteristics</td>
<td>50</td>
</tr>
<tr>
<td>5.16</td>
<td>Effect of atmospheric pressure on differential transducers</td>
<td>52</td>
</tr>
<tr>
<td>5.17</td>
<td>Tables stored in memory</td>
<td>55</td>
</tr>
<tr>
<td>5.18</td>
<td>Pseudo code for the LIMITS routine</td>
<td>59</td>
</tr>
<tr>
<td>5.19</td>
<td>Pseudo code of the TOLERNCE routine</td>
<td>62</td>
</tr>
<tr>
<td>5.20</td>
<td>Pseudo code of the SET PRES routine</td>
<td>65-66</td>
</tr>
<tr>
<td>5.21</td>
<td>Example of the characteristics of PCU with out rezeroing</td>
<td>70</td>
</tr>
<tr>
<td>5.22</td>
<td>Curve characterizing the hardware</td>
<td>72</td>
</tr>
<tr>
<td>5.23</td>
<td>Equations necessary to calculate the DAC value</td>
<td>74</td>
</tr>
<tr>
<td>5.24</td>
<td>Hardware characteristics as effected by changes in atmospheric pressure</td>
<td>76</td>
</tr>
<tr>
<td>5.25</td>
<td>Partial graph of PCU characteristics</td>
<td>77</td>
</tr>
<tr>
<td>5.26</td>
<td>Actual output pressure just below the tolerance limit</td>
<td>81</td>
</tr>
<tr>
<td>5.27</td>
<td>Valve configuration in clamp-off mode</td>
<td>85</td>
</tr>
<tr>
<td>6.1</td>
<td>Actual and simulated air pressure pneumatics</td>
<td>88</td>
</tr>
<tr>
<td>6.2</td>
<td>Simulated HSC valve response verses input</td>
<td>90</td>
</tr>
<tr>
<td>6.3</td>
<td>PCU hardware simulation schematic</td>
<td>92</td>
</tr>
<tr>
<td>6.4</td>
<td>PCU hardware simulation</td>
<td>93-94</td>
</tr>
</tbody>
</table>
Figure 6.5  Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_+ = +15V$, $V_- = 0V$, and $R_2=3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$

Figure 6.6  Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_+ = 0V$, $V_- = -15V$, and $R_2=3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$

Figure 6.7  Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_+ = +15V$, $V_- = -15V$, and $R_2=3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$

Figure 6.8  Simulated response of the PCU hardware with $C_1 = 1000\mu F$, $V_+ = +15V$, $V_- = 0V$, and $R_2=3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$

Figure 6.9  Simulated response of the PCU hardware with $C_1 = 100mF$, $V_+ = +15V$, $V_- = 0V$, and $R_2=3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$

Figure 6.10 Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_+ = +30V$, $V_- = 0V$, and $R_2=3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$

Figure 6.11 Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_+ = +15V$, and $V_- = 0V$
CHAPTER 1

Introduction

The objective of this thesis is to present a microprocessor-based implementation of an air pressure controller. It explains the hardware and software that was developed for the pressure controller, as well as the system in which it is installed. It also explains the performance of the pressure controller hardware through the use of simulation.

Pressure Systems Inc. (PSI) conducted a market study to determine what pressure controllers are available for testing the transducers that they sell. They found that there are few small, inexpensive, and portable products available on the market for controlling pressure to the accuracy that their own pressure transducers can read.
Companies that have bought PSI’s transducers are unable to test their transducers to see if they are still in calibration. These companies have had to remove their transducers from the systems they are installed in and send them to PSI for recalibration to see if they are calibrated. With an accurate and portable air pressure controller, the user could test the transducer while it is installed in his or her system. If the user finds that the transducers are out of calibration, they could send the transducers to PSI. They would no longer need to wonder if the transducers are still accurate or not.

After evaluation of the problem at hand, the Sonix™ Pressure Controller was conceived. It is a pressure generation instrument designed to be able to set absolute and pseudo-differential pressures accurately and with great stability. The Sonix™ Pressure Controller is installed in an existing pressure display offered by PSI. The display, the Sonix™ Digital Pressure Standard, plus a controller card and pneumatic valving make up the pressure controller. It is able to set a user-defined pressure from its front panel or through the IEEE-488 interface.

To introduce the properties of air pressure, Chapter 2 introduces explains the fundamental properties of pressure and pressure measurement devices. It also introduces the transducers used in the implementation of this pressure controller. Chapter 3 explains the system in which the pressure controller is installed. It indicates which functions are performed in hardware, and which are in software. Chapter 4 defines the design requirements of the pressure controller while Chapter 5 explains its implementation. Each function that the pressure controller performs is explained in detail in this chapter. Chapter 6 analyzes the results of testing and indicates if the requirements have been met. It also gives examples of possible features and functions that could be added to the
design to enhance its performance. A summary is contained in Chapter 7. The hardware schematic, software listing, and Sonix™ Display Menu Tree are included in the appendices.
CHAPTER 2

Air Pressure

Pressure is the force exerted by a fluid perpendicular to any bounding surface, \( P = \frac{dF}{dA} \). Therefore, if air is the fluid under consideration, pressure is the force which air exerts on all of the surfaces it contacts.

This chapter discusses the measurement of pressure as well as the theory of operation of the pressure measurement device used in this research.

2.1 Pressure Measurement

The devices used to measure pressure are referred to as transducers. A transducer is a device that takes energy from one system and supplies energy, in any form, to another system. In particular the essential feature of a conventional pressure transducer is an elastic element that converts energy from the pressure system to a mechanical or
electrical system. The most popular technique is to convert pressure energy into electrical energy because electrical signals can be amplified, transmitted, controlled, and measured with relative ease.

Transducers that measure air pressure are either absolute or differential devices. Absolute transducers output a pressure reading which is referenced to vacuum whereas differential device use atmospheric pressure as their reference. Therefore, if atmospheric pressure was 10 PSIA (Pounds per Square Inch Absolute) and the pressure being measured is 20 PSIA, an absolute device reads 20 PSI and a differential device reads 10 PSI.

When measuring pressure, one must realize that there are two elements of pressure that when added together equal total pressure. They are static and dynamic pressure.

Static pressure is the actual pressure of the fluid whether the fluid is in motion or at rest. In other words, if a transducer were travelling along with the fluid the pressure ‘felt’ would be the static pressure.

Dynamic pressure is the pressure component which is solely a result of fluid in motion. For example, if fluid is in motion and a component of its velocity is in the direction of a boundary, then that fluid exerts more force on that boundary than it would if it were at rest. This is due to the force calculated by $F = mV^2/2$ where $m$ is the fluid’s mass and $V$ is its velocity.

In systems where there is fluid flow, care must be taken in the placement of transducers. Otherwise, the pressures read may be something other than what is expected. Depending
on where the transducers are placed, they can read remarkably different pressures. In some cases only static pressure is read as opposed to total pressure. Figure 2.1 shows two open ended tubes placed in flowing water. The left tube has its lower opening perpendicular to the flow. The rise of water in the tube indicates static pressure. The other tube has a 90 degree bend in the direction of the flowing water. The water in the tube indicates total, i.e. both static and dynamic, pressure. Therefore, if two transducers are placed along the same tube but one is connected such that pressure is measuring only static pressure and the other is measuring total pressure, the two transducers could transmit very different results. The only way the results would be similar is if there is very little air flow such that the dynamic pressure contributes little to the total pressure.

Another aspect of fluid flow that needs to be considered is turbulence. Turbulence is a phenomenon that causes fluctuations in fluid velocity. If fluid flows through a bend causing a change in flow direction, fluid flow will become turbulent near this point. As a result, if a transducer is placed near one of these bends, its reading will fluctuate dramatically causing the system to appear unstable. To avoid this problem, all transducers should be placed away from all flow direction changes so that the turbulence has had sufficient distance to dissipate to an acceptable level.

2.2 Solartron™ Absolute Pressure Transducer

One of the transducers used in this research converts air pressure to electrical frequency. The transducer is powered by a 15 volt source. There are two cylinders, one inside the other, with an oil layer between them. The outer cylinder contains a diaphragm at one
Figure 2.1. Basic Pitot-tube method of sensing static, dynamic, and total pressure [1].
end, a diode in its wall, and is covered with an electrical coil. When the transducer is powered, the inner cylinder resonates at some frequency. This frequency is output from the transducer and can be read by the user. As the pressure on the diaphragm of the outer cylinder changes, so does the pressure on the inner cylinder. This change in pressure of the inner cylinder causes it to vibrate at a different frequency.

This frequency is extremely repeatable over the pressure range of the device. However, there is one other variable that affects the transducers accuracy - temperature. That is the reason for the diode. The voltage that is transmitted by the diode is a function of temperature. As a result, diode voltage is converted to temperature.

Through experimentation, a 15 term equation was developed by the producers of the Solartron™ that uses the frequency and diode voltage to determine absolute pressure. The resulting value is accurate to 0.01% full scale. In other words, if the transducer is designed to operated at a pressure range of 0 to 50 Pounds Per Square Inch (PSI) then it will be accurate to 0.0001 * 50 = 0.005 PSI.

The stability of the Solartron™ is even better than its accuracy. However, the stability is dependent on how long the frequency is read before the pressure is calculated. If the frequency is read for approximately 0.5 seconds, a stability of 0.0004% full scale can be expected.
2.3 S100\textsuperscript{TM} Differential Pressure Transducer

The S100\textsuperscript{TM} is a differential transducer that converts pressure to voltage. It has the capability of following a 20 kilohertz square wave. In other words, the voltage at the output has time to respond and stabilize for an input pressure square wave that oscillates at approximately 1/20,000th of a second.

Another feature of the S100\textsuperscript{TM} is that it is a differential transducer; its "zero" is atmospheric pressure. That means that if pressure is greater (less) than atmospheric pressure the S100\textsuperscript{TM} outputs a positive (negative) voltage.

As for the stability and accuracy of the S100\textsuperscript{TM}, they are not important in the application in which the S100\textsuperscript{TM} will be used in the pressure controller. The only property that is important is its repeatability. The S100\textsuperscript{TM} must repeatedly output approximately the same voltage for a given pressure value. The S100\textsuperscript{TM} is repeatable to approximately +/-2\% full scale. In other words, the S100\textsuperscript{TM} will consistently output the same voltage, +/-2\% full scale, for a given input pressure.

One of the main parameters that effects the repeatability of the S100\textsuperscript{TM} is temperature. As the temperature changes so does the output voltage. Unfortunately, the S100\textsuperscript{TM} does not compensate for temperature changes which results in the device being less repeatable.
CHAPTER 3

The Sonix\textsuperscript{TM} Display

The "box" in which the Pressure Controller Unit (PCU) is to be mounted is called a Sonix\textsuperscript{TM} Display [2], shown in Figure 3.1. This display contains a Solartron\textsuperscript{TM} pressure transducer, a 6809 microprocessor, RAM, ROM, a real-time clock, and more. Its main purpose is to read and display air pressure that is input into a connector on the rear panel of the display. The pressure is read by the Solartron\textsuperscript{TM} that is mounted inside the display. The result can then be displayed on its front panel or it can be sent to a host computer.

The Sonix\textsuperscript{TM} Display can be programmed from the front panel or from a host computer through the use of an IEEE-488 or RS-232 interface. Built-in menus allow the user to program the device from the front panel in order to configure the Sonix\textsuperscript{TM} Display.

The operating system is an interrupt-driven, queue-based system. Data is passed between tasks in message queues and display configuration is stored in tables. To collect data from the transducers, the transducers are first triggered. Data is then collected when the microprocessor is interrupted indicating that the data is valid. The transducers are then
Figure 3.1. Sonix™ Display.
retriggered and the previous data is processed and sent to the appropriate queues. The main routine then transmits the appropriate data to the front panel alphanumeric displays and, if required, sends data to the host computer [2]. What is sent to the front panel is defined by the user. He or she may configure the display through the use of a host computer or by using the built-in menus.

The Sonix™ Display can access up to six "virtual transducers" numbered 1 through 6. However, the information collected for these devices actually comes from up to two physical transducers. The information that is to be collected for any given virtual transducer is defined by the user, and will come from Solartron™-A or Solartron™-B. Once he or she selects which device to use, the user must decide how many readings should be averaged before it is saved. Next, the user must define the units of the virtual transducer (Diode Voltage, Period, PSI, Bar, mBar, inH2O, PSF, kPa, inMg, mmHg, or User-Definable). For the purpose of example, let transducer 1 be defined as Solartron™-A, two averages per saved pressure, and PSI. In that case if the display was configured to display transducer 1 then it would trigger Solartron™-A to start reading a pressure. Once the pressure was read, it would be triggered again and read again. At that time, the average of those two readings would be computed and the result would be displayed on the front panel in PSI units.

Before the PCU was developed, the Sonix™ Display’s primary function was to display Solartron™ pressure readings. A drawing of the front panel is shown in Figure 3.2. To display pressure the following hardware is included in the Sonix™ Display:

1) a front panel controller board,
2) a mother board with four slots,
Figure 3.2. Sonix™ Display front panel.
3) a microprocessor (MPU) board, 
4) a General Purpose Interface Board (GPIB) board, 
5) a Solartron™ Interface board, 
6) a power supply, and 
7) a Transducer Interface board 

Three of the boards fit into a mother board - the MPU, the GPIB, and Solartron™ Interface.

The MPU contains a 6809 microprocessor, 27512 EPROM, battery-backed RAM, and a real-time clock. This card is responsible for controlling the rest of the "box".

All of the software is in the form of firmware located in the EPROM of the MPU, while RAM contains the data, flags, and configuration information. This information is saved even on power down since it is stored in battery-backed RAM. The reason RAM is battery-backed is that much of the information saved is configuration information. In other words, the user, through the use of the firmware, is able to setup the display, etc., and is not required to reinitialize this information every time the unit powers up.

The Solartron™ Interface board is connected to a Transducer Interface board which is mounted in the rear of the Sonix™ Display. The Transducer Interface board powers the Solartron™ and relays signals to the Solartron™ Interface board. This board collects frequency and diode voltage information sent from the Solartron™ mounted in the rear of the display. Once the Solartron™ Interface board has collected the frequency information, it interrupts the microprocessor on the MPU using the interrupt request
signal IRQ. The microprocessor then receives the information and, using the firmware, calculates the pressure that the transducer read.

The GPIB board is a general purpose interface board which is used to communicate with the host computer. Firmware was written which allows the user to execute the same functions which are available at the front panel.

The front panel consists of a group of function and numeric keys. The function keys consist of Reset, Display, Control, and Program. When Reset is pressed the system is reset and the display operates in display mode. Display mode is defined by what the user configured it to display. To configure the display mode, the user first presses the Display key. At this point he or she can decide what the display contents and format will be. Some possible content options are time, date, transducer pressure, and transducer diode voltage. Similarly, the possible formats are default, fixed and floating point. To select the display configuration, the user follows through the menu tree and selects the options he or she desires. Figure 3.3 illustrates the menu tree which configures the display. In addition to the options mentioned above, there are many other functions which the Sonix™ Display performs. However, they are not pertinent to the use or design of the Pressure Controller Unit (PCU), and are not discussed here. However, a flow chart of all menu options can be found in Appendix C.
Figure 3.3. Display configuration menu tree.
CHAPTER 4

The Controller: Design Requirements

This chapter describes the functional, control and speed requirements of the Pressure Controller Unit (PCU) as well as the rationale for these requirements. It also introduces the parameters that must be controlled and the modes that the PCU should operate in.

4.1 Functional Requirements

The PCU should allow the user to input a pressure value via the front panel of the Sonix™ Display or a remote host computer. To implement this function from the front panel of the Sonix™ Display, the user should press the control button. At this point, the user-friendly menu for controlling pressure should appear on the two alphanumeric displays.
Once the value has been entered, that pressure should appear at the output as soon as possible. It should remain there until the user inputs another pressure value, or chooses to return the output to atmospheric pressure.

Another requirement is that the hardware should define a maximum pressure which the PCU can set. Since the internally-mounted Solartron™ will be used to read the pressure, the maximum pressure should be the maximum pressure that the Solartron™ is designed to read. If the PCU were able to set a pressure larger than the Solartron™ was designed for, the transducer could be damaged. Note that since the Solartron™ can read pressures down to vacuum, no minimum pressure setting is necessary.

In addition to defining this "hardware" maximum pressure, there should be a user defined minimum and maximum pressure. The pressure limit described in the previous paragraph is independent of which device the user may place at the output. It is quite likely that the user’s transducer cannot operate over the entire operating range of the PCU. Therefore, these limits enable the user to protect himself or herself from possibly damaging his or her own device.

Another requirement is that the PCU should operate in both absolute and differential modes. As mentioned before, there are two types of pressure transducers available in the marketplace: absolute and differential. If the user has a differential transducer, he or she will wish to operate in differential mode.

In addition to displaying and setting the pressure in both modes, the PCU must display and set the pressure in any unit the user desires. The software that exists in the Sonix™ Display already has the capability of displaying pressure in any unit, therefore, the
pressure controller must merely be aware of the virtual transducer's units. This is easily done since the units of each virtual transducer are stored in memory.

Although the above safety precautions have been mandated for not allowing the transducers to become damaged, there may be some instances that require the user to quickly abort setting an output pressure. As a result, the RESET key must cause the output pressure to return to atmospheric pressure.

In addition to this quick and dirty abort technique, an option should be offered in the control menu which also allows the user to return to atmospheric output pressure.

### 4.2 Control Requirements

One of the most important requirements of the PCU is that it is not the limiting factor in the pressure setting algorithm. As a result, the pressure should be controllable to the accuracy of the internal SolartronTM. A SolartronTM is accurate to 0.01% full scale.

Another control requirement is that the PCU must reach the desired pressure quickly without the system being underdamped. In an underdamped system, the pressure that is sent to the output line overshoots the desired pressure as shown in Figure 4.1. In other words, if the desired pressure is greater than (less than) atmospheric pressure, the pressure on the output line temporarily exceeds (falls below) that desired.
Figure 4.1. Damping conditions.
The reason for not allowing the PCU to overshoot is that it may damage the user’s transducer. If the user requests an output pressure that is near the edge of its operating range, the overshoot may extend outside of that boundary.

To reach the desired pressure without overshoot, the hardware should approach critical damping [3]. Critical damping is when the desired pressure is reached in the shortest time possible without overshotting that pressure.

The final control requirement is that once the pressure has been set, the software must periodically check it to make sure it is still within some user defined tolerance. The tolerance should be allowed to be any number greater than 0.01% full scale of the Solartron™. If the difference between the desired pressure and the actual output pressure is greater than the tolerance, the pressure output must quickly change to fall within this tolerance.

4.3 Speed Requirements

The only time that speed is critical is in the routines that set pressure and maintain pressure. The other routines will be related to menu options which can easily function as fast as the user can type.

To increase the speed of these pressure controlling algorithms, care must be taken to produce efficient code. The reason for this is twofold. Firstly, the entire time these algorithms are executing the normal display mode will not be active. Secondly, the
pressure is not within the desired tolerance. This is not acceptable and, therefore, the desired pressure must be reached quickly.
CHAPTER 5

Function and Design of the Pressure Controller

This chapter describes the design of the pressure controller unit (PCU). The hardware design uses many of the components of a similar pressure controller designed for another PSI system. The software algorithms, however, are all original and were developed with the cooperation of PSI.

The first section of this chapter introduces the function that the PCU performs and how the PCU works. It gives insight into which functions are performed in hardware and which are performed in software. The succeeding sections explain the hardware and software design in detail.
5.1 Functional Description of the PCU

When the user decides to use the pressure controller option, he or she must first press the CONTROL key on the front panel of the Sonix™ Display. When the control function is active, the control function light will illuminate as shown in Figure 5.1. At this point, the user may select any of the control options defined below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET PRES</td>
<td>Input a pressure value and output that pressure.</td>
</tr>
<tr>
<td>STOP &amp; V</td>
<td>Stop controlling the output pressure and return it to atmospheric.</td>
</tr>
<tr>
<td>STOP &amp; C</td>
<td>Stop controlling the output pressure and clamp that pressure on the output line.</td>
</tr>
<tr>
<td>LIMITS</td>
<td>Define the minimum and maximum pressure that can be output.</td>
</tr>
<tr>
<td>TOLERANCE</td>
<td>Define the pressure tolerance which the output pressure must be controlled to stay within.</td>
</tr>
<tr>
<td>BLD TBL</td>
<td>Build look-up tables for use by the pressure setting algorithm.</td>
</tr>
<tr>
<td>RULES</td>
<td>Scroll PCU configuration rules and information.</td>
</tr>
</tbody>
</table>

A menu tree, shown in Figure 5.2, allows a user to follow along with keystrokes through the top down menu. Since the menu is top down, no backtracking is allowed. Therefore once a function at the bottom of the tree has been executed, the user is required to start at the beginning of the menu.

Once the user builds look-up tables, inputs a tolerance, and defines a minimum and maximum pressure, he or she can input a pressure. This pressure value is used to
Figure 5.1. Control function.
Figure 5.2. PCU menu tree.
perform two functions. First, it is used to determine how the valves of the PCU will be
configured. Then, the requested pressure value is converted to a digital value that will be
sent to the DAC on the PCU board. This conversion is performed with the help of the
look-up tables that were formed previously.

Valves control how air will flow through the pneumatics. As a result, the valves are
configured differently depending on which pressure value the user has selected. The
DAC value is what is used to tell the hardware what pressure to set.

5.2 PCU Hardware

The PCU hardware, shown in Figure 5.3, consists of a printed circuit board, valves, a
hardware-feedback differential transducer, and pneumatics. The PCU board contains the
control logic and valve drivers for controlling the valves and thus the air flow. On the
back of the PCU board are electrical connectors that connect the board with the valves
and pneumatics in the rear of the Sonix™ Display. Since these connectors are on the
back of the board, the board must be mounted in the forth slot of the mother board or the
connections cannot be made due to space requirements. The valves are mounted to an
aluminum manifold that provides the necessary pneumatics for the air flow requirements
of the pressure controller. The function of this hardware is described below.

The PCU board is responsible for four major functions:
Figure 5.3. Pressure controller block diagram.
1) decoding the addresses on the bus,
2) telling the Sonix™ Display that it is present and what kind of PCU it is,
3) turn the valves on and off, and
4) sending current to the servo valve.

The following sections explain these functions in detail, and how they are implemented.

5.2.1 Digital Part of the PCU

The decoding logic is implemented by a single 20L8 Programmable Array Logic (PAL) chip. All address and control bus signals that are needed to perform any function on the PCU are decoded by the PAL to produce chip selects and other control signals. No other decoding is used. The pin out of the 20L8 is shown in Figure 5.4 while the PAL program is shown in Figure 5.5. To test the program before actually burning the PAL a PAL simulation program was used [6]. This saves chips because the program can be tested before it is actually implemented in the circuit. The results of this simulation are shown in Figure 5.6.

/DASEND is the chip select for the DAC. When it is active, the microprocessor sends the low (/WR5 = 0 ) or high byte (/WR6 = 0 ) of a digital word to the DAC. This digital value represents a pressure that will be output. When the high byte is sent, the data is simultaneously latched. Therefore, once the data is latched, the DAC starts to convert the digital value to an analog voltage. This analog voltage then enters into the feedback circuit which are described later.
Figure 5.4. PAL pin out.
A0 /WR /WR6 /WR5 CKO /ICKO NC /DASEND /EO /G NC VCC

EQUATIONS
EO = RMTS*/A10*A9*A8*A7*A6*A5*A4*/A3*/A2*/A1*/A0
DASEND = WR5 + RMTS*WR*/A10*A9*A8*A7*/A6*A5*/A4*A3*A2*A1*/A0
G = EO + DASEND + /CKO
WR5 = RMTS*WR*/A10*A9*A8*A7*/A6*A5*/A4*A3*A2*A1*/A0
WR6 = RMTS*WR*/A10*A9*A8*A7*/A6*A5*/A4*A3*A2*A1*/A0
ICKO = RMTS*WR*/A10*A9*A8*A7*/A6*A5*/A4*A3*A2*/A1*/A0
/CKO = ICKO

SIMULATION
TRACE_ON /EO /DASEND /G CKO /WR6 /WR5
SETF /RMTS /A10 A9 A8 A7 A6 A5 A4 /A3 /A2 /A1 /A0
CHECK /G /EO /DASEND /WR5 /WR6 /CKO

SETF RMTS
CHECK G EO /DASEND /WR5 /WR6 /CKO

SETF /A6 /A4 A3 A2 A0 WR
CHECK G /EO DASEND WR5 /WR6 /CKO

SETF A1 /A0
CHECK G /EO DASEND /WR5 WR6 /CKO

SETF /A1
CHECK G /EO /DASEND /WR5 /WR6 CKO
TRACe_OFF

Figure 5.5. PAL program.
Figure 5.6. PAL simulation results.
/EO is the chip select signal that allows the microprocessor to read the jumper switches so the software knows what pressure range the hardware is designed for. U5 is the inverting latch that /EO selects. Figure 5.7 shows eight pull-up resistors and eight jumper switches connected to the inputs of the latch. When a jumper is installed on any given line, that line becomes grounded. Otherwise, the pull-up resistor will force it to 5 volts. Therefore, when the microprocessor selects U5, it reads the switch settings, and thus the pressure range of the PCU hardware. Figure 5.8 shows the possible PCU pressure ranges that are offered and their corresponding jumper settings.

Also included in the digital part of the hardware is a bidirectional tri-state latch used to control signal flow on the PCU board. When any of the PCU board control signals are generated, the PAL also generates the /G signal which selects this latch. As a result, any time the 6809 microprocessor addresses a chip on the PCU board, the bidirectional latch allows the board to communicate with the 6809. The direction of data flow is determined by the R//WR signal sent by the 6809. If R//WR is high (low) then the latch will allow data to flow from the PCU board (6809) to the 6809 (PCU) board, but not the other way.

5.2.2 Valves

The valves are a critical part of the hardware implementation. They are used to control all parts of the air flow dynamics. Figure 5.9 is a schematic of the pneumatics of the pressure controller.
Figure 5.7. PCU hardware identification circuit.
Figure 5.8. Jumper settings for all PCU pressure ranges.
Figure 5.9. Schematic of PCU pneumatics.
To set output pressure from near vacuum to pressures above atmospheric, a vacuum pump and an air compressor are needed. However, depending on what pressure the user is trying to set, the user may not need both of these sources. For example, if the user wishes to control a pressure significantly above atmospheric, no vacuum pump is necessary.

As a result of these different pressure source requirements, the PCU must be responsible for configuring the valves dependent on the pressure requested. To solve this problem three configurations are used.

The best way to explain how a pressure is set and what the valves contribute is through the use of examples. In Figure 5.9, the configuration shown is that at power-up - all valves are turned off. To set pressures from vacuum to near atmospheric, valve 0 is off valve 1 is on and, as for all pressure ranges, valves 2 and 3 are on. Since all pressures in this range are below atmospheric, there is no need to supply air pressure from an air compressor. Therefore, valve 0 is turned off and pressure is supplied from the atmosphere. Similarly, since the output pressure must be pulled below atmospheric, a vacuum pump is needed which explains why valve 1 is activated. Valves 2 and 3 are activated when any pressure is to be sent to the output. This is necessary if pressure is to reach the user's device since the air will be blocked otherwise.

After configuring the valves, air starts flowing from valve 0 through the HSC valve and out to the user's transducer, valve 1, the S100™, and the Solartron™. An explanation of the utility of the S100™, Solartron™ and valves 2 and 3 is explained later in this chapter. In addition, the analog section of this chapter explains how the HSC valve "gate" is controlled and its function.
To set pressures from near atmospheric to the maximum operating pressure of the Solartron\textsuperscript{TM}, valves 0, 2 and 3 are activated while valve 1 is deactivated. Since pressures in this range are above atmospheric, an air compressor is used to drive the pressure above atmospheric. A vacuum pump is not needed.

The final scenario includes pressures near atmospheric. This requires both valve 0 and 1 to be activated. Since the pressure controller is actually an air flow regulator, to accurately control flow there must be a substantial rate of flow. Therefore, it is necessary to have both a positive and negative differential pressure source to set pressures near atmospheric (zero differential) pressure. As a result, both the air compressor and vacuum pump are used to set pressures in this range.

\textbf{5.2.3 Analog Part of the PCU}

When the user defines a pressure to output, the software must decide which valve configuration to use. Once chosen, the software writes a byte to the 74LS174 shown in Figure 5.10. Each of the four valves corresponds to one of the bit positions of the low nibble of this byte. Valves 0 through 3 correspond to bits 0 through 3, respectively. If a 1 appears at a bit position corresponding to a valve, a 5 volt signal is sent to the voltage converter and optical isolator which steps the voltage up to activate the valve. Similarly, to deactivate a valve, a zero is written to its corresponding bit position.
Figure 5.10. Valve-driver circuitry.
Once the valves have been configured the software decides what value to send to the DAC. This value is calculated by the software from information previously stored in look-up tables in memory. The DAC will convert this value to a voltage and send it to the circuit that drives the HSC servo valve shown in Figure 5.11. At this point, the DAC voltage and the amplified signal from the S100\textsuperscript{TM} converge on the first unity gain amplifier.

The pressure that the S100\textsuperscript{TM} reads is that of the output line which leads to the user’s transducer. The S100\textsuperscript{TM} is used in the hardware feedback loop as part of the control circuitry that controls the output pressure.

The voltage from the S100\textsuperscript{TM} is linearly amplified such that the voltage out of the amplifier, U\textsubscript{14}, for the operating range of the PCU is approximately -15 to +15 volts. This is the same as the voltage range of the DAC. The reason that these ranges must match is that all DAC outputs should correspond to a pressure value that the PCU is designed to set. If the amplified S100\textsuperscript{TM} voltage matches that output by the DAC, the control circuitry is stable, and thus a output pressure is "locked in". Each time that same voltage is output by the DAC, the same pressure should appear at the output.

The difference between the amplified S100\textsuperscript{TM} voltage and the DAC voltage is output by U\textsubscript{11}, and is input into an integrator, U\textsubscript{12}. The output of the integrator is then added, via U\textsubscript{13}, to the output of the DAC. The sum of the result is then converted to a current that drives the HSC servo valve.

To better explain this analysis of the circuit an example is used. Assume that the user decides to output a pressure above atmospheric pressure. If so, the valves are configured
Figure 5.11. HSC valve driver circuitry.
such that the high pressure source is the air compressor and the low pressure source is atmosphere instead of the vacuum pump. Next, the pressure value input by the user is converted to a digital value which is sent to the DAC. Assume the DAC converts that value to \( V_3 \), shown in the top graph in Figure 5.12. Initially, \( V_3 \) equals 0 volts and once the DAC converts the digital value to its analog equivalent, a 10 volt step occurs at the output.

If the circuit is initially outputting atmospheric pressure, the voltage out of the S100\textsuperscript{TM}, \( V_2 \), begins at 0 volts shown in the top graph in Figure 5.12. The difference between the DAC and the amplified S100\textsuperscript{TM} voltage is then integrated by \( U_{12} \). Initially, the integral is 0 volts so that the output of \( U_{13} \) equals \( V_3 \). However, as time progresses and the difference in voltages into \( U_{11} \) continue to exist, the integrator continues to increase the voltage at its output. The result is a growing sum at the output of \( U_{13} \). This sum, \( V_7 \), is converted to a current and is input into the HSC valve. \( V_7 \) is shown in the bottom graph in Figure 5.12.

Initially \( V_7 \) equals \( V_3 \) and the HSC valve is configured as in Figure 5.13 (a). However, since the output pressure read by the S100\textsuperscript{TM} does not produce a voltage, \( V_2 \), that equals \( V_3 \) the sum, \( V_7 \), increases. The result is a larger current into the HSC that causes the valve to allow more air flow from the high pressure source, shown in Figure 5.13 (b).

As the output pressure increases \( V_2 \) also increases which results in a smaller difference coming from \( U_{11} \). As this difference decreases the some of the values into \( U_{13} \) decreases. Therefore the current into the HSC decreases which causes the valve to cutoff the high pressure source somewhat as is shown in Figure 5.13 (c).
Figure 5.12. Response of the analog circuit with respect to time.
Figure 5.13. HSC valve as current input changes.
Eventually, the output pressure increases to the point where the amplified voltage coming out of the S100™ equals the voltage coming out of the DAC. Therefore, the input and output of the integrator is 0 volts. The result is that the summed input of U13 is equal to V3. At this point the output pressure is at the correct pressure resulting in a "constant" current into the HSC. If the pressure on the output line changes slightly due to changes in the high pressure source or for other reasons, the S100™ will sense that change which causes the HSC to open up or close off the high pressure source as is needed. For a more in depth analysis of the control circuit response see Chapter 6.

Because the HSC valve has internal bushings used to seal the flap against the wall of the valve, friction comes in to play. For very small changes in current into the HSC the flap many not move due to static friction. Eventually, as the current is continuously increased, static friction is overcome and the flap quickly moves. This property makes it difficult to make small changes in the location of the flap. To alleviate this problem, a dither circuit was added to the reference voltage of U13. This circuit causes a 60 cycle ripple in the output voltage of U13. This ripple propagates through the voltage-to-current conversion and into the HSC valve. The result is that when the current into the HSC changes slightly, the HSC will move without sticking because the flap is continuously in motion. The amplitude of the ripple was designed such that its effect on output pressure is minimal.
5.3 Software

This section gives an in depth description of each software function and explains how they are implemented.

5.3.1 PCU Initialization Software

Upon system boot-up, routines are run to "check on" and initialize the PCU related software. The first routine executed is CHECK_PCU. The primary function of this routine is to check to see if the PCU is plugged into the mother board. It does so by reading the jumper switches on the PCU board. If a PCU board is plugged in, then the value read will be something other than a tristated value of 0FFH. The value read will indicate the maximum pressure value that the internal Solartron™ can sense. This value is then stored at a memory location called PCU_RANGE in the memory block relating to PCU flags and configuration information. It is stored by a routine that also updates a checksum which is stored in memory. The checksum is the sum of the values stored in a defined block of memory. If a value in that block is updated, the old value in that location must be subtracted from the checksum and the new value to be stored must be added to the checksum. The reason for having checksums is to detect faults, failures or incorrect data in that block of memory.

The PCU data that is stored in memory is broken into two parts which are shown in Figure 5.14. The "PCU flags and configuration information" and the look-up tables are
VAR_CONTROL: .EQU VAR_STORE3+33 ; Variables used by pressure controller
PCU_EXISTS: .EQU VAR_CONTROL+0 ; Flags PCU card (1 byte)
PCU_RANGE: .EQU VAR_CONTROL+1 ; Storage for value of PCU type (1 byte)
PCU_IN: .EQU VAR_CONTROL+2 ; Storage for value of floating point input
(USER_PCUMAX: .EQU VAR_CONTROL+6 ; Storage for user defined max pressure
(USER_PCUMIN: .EQU VAR_CONTROL+10 ; Storage for user defined min pressure
PCU_FP_MAX: .EQU VAR_CONTROL+14 ; Storage for hardware defined max pressure
FPLS_TABLE: .EQU VAR_CONTROL+18 ; Storage for low pressure lookup table
FPIS_TABLE: .EQU VAR_CONTROL+146 ; Storage for middle pressure lookup table
FPMS_TABLE: .EQU VAR_CONTROL+274 ; Storage for high pressure lookup table
TEMP_VAR: .EQU VAR_CONTROL+402 ; Temporary variable (1 byte)
FP_PRCNT_JM: .EQU VAR_CONTROL+403 ; Storage for FP value of percent of desired output will be sent in PCU (4 bytes)
FP_PRCNT_DIF: .EQU VAR_CONTROL+407 ; Storage for FP value of difference between actual and desired that is allowed (4 bytes)
CALING: .EQU VAR_CONTROL+411 ; Flag tells if calibration in progress (1 byte)
LAST_ATM: .EQU VAR_CONTROL+412 ; Contains last atmospheric reading (4 bytes)
FPSLOPE: .EQU VAR_CONTROL+416 ; Slope of PCU feedback (XDCR1) in FP (4 bytes)
XDCR1_FPP: .EQU VAR_CONTROL+420 ; XDCR1 last FP pressure read (4 bytes)
LAST_DOUT: .EQU VAR_CONTROL+424 ; Last output in HEX (2 bytes)
DESIRED_P: .EQU VAR_CONTROL+426 ; Desired FP pressure sent out by PCU (4 bytes)
TOLERANCE: .EQU VAR_CONTROL+443 ; FP value of user defined tolerance (4 byte)
BEFORE_SDM1: .EQU VAR_CONTROL+447 ; Flag saying in SDMMAIN before SDM1
REVS_SLOPE: .EQU VAR_CONTROL+450 ; Slope for calc DAC value when going from pressure
XDCR1_FPP2: .EQU VAR_CONTROL+454 ; Second-to-last pressure read (4 bytes)
PCUSUM: .EQU VAR_CONTROL+458 ; PCU related memory checksum (1 byte)
PCUTABLESUM: .EQU VAR_CONTROL+459 ; PCU tables related memory checksum (1 byte)

Figure 5.14. PCU related information stored in battery-backed RAM.

Function and Design of the Pressure Controller 47
the two blocks of memory. The checksums perform the same function for both memory blocks. The only difference is which memory block they check. To explain the function of the checksum, the following example is given.

The checksum routine used to store PCU_RANGE works as follows. Each time a memory location in the "PCU flags and configuration information" section is updated, a memory location containing the checksum is updated. For instance, in the above example PCU_RANGE is to be updated. First the old value of PCU_RANGE is subtracted from PCUSUM and then the new value of PCU_RANGE is added to PCUSUM. Finally, the new value is stored at PCU_RANGE.

After storing the pressure range value, CHECK_PCU finishes running, and the CHECK_TABLE routine executes. This routine checks to make sure the PCU look-up tables have not been corrupted. It checks this by summing the contents of all memory locations in the look-up tables and comparing the result with the checksum called PCUTABLESUM. If they are equal then everything is fine and no error is indicated. However, if the results are not equal "TABLES CORRUPT" is displayed on the front panel. This message continues to be displayed until a key is pressed. Once a key is pressed the old checksum is replaced by the one just computed; the user will not receive the error message again. Since the message is merely a warning, it is up to the user to decide if he or she wishes to build the tables again.

After CHECK_TABLE runs, a similar routine called CHECK_PCU_D is executed. The only difference between this routine and CHECK_TABLE is that this one checks a different area of memory. If the data in the "PCU flags and configuration information" section has not been incorrectly altered the routine is finished. Otherwise, a warning
message appears on the front panel of the Sonix™ Display indicating "PCU DATA CORRUPT".

After these initialization routines execute, nothing else related to the PCU will execute unless the CONTROL button is pressed or a host computer sends a PCU command to the Sonix™ Display.

If the PCU does not exist, PCU_RANGE = 0FFH, and when the CONTROL button is pressed the following message will be displayed on the front panel: "CONTROL DISABLED". Therefore, the user is unable to enter the control menu.

However, if a PCU does exist the user can enter the main CONTROL menu. From this menu there are seven options. By pressing the arrow keys, the user can scroll through the options. To select one, he or she must press the enter key. The following sections explain what these options do and how they do it.

5.3.2 Build Tables Routine (BLD TBL)

Through experimentation, the behavior of the S100™ feedback transducer has been shown to be extremely repeatable but not linear. Figure 5.15 shows possible outputs of the S100™ as a function of pressure. If the same pressures are input to the S100™ over a given time period, the same outputs will result, i.e. the behavior of the S100™ is repeatable. However, notice that the curve is not linear. As a result, it is not possible to set pressures quickly without knowledge of this curve unless overshoot is allowed.
Figure 5.15. Typical graph of S100™ characteristics.
Building tables allows this curve to be stored in memory so that when a pressure is asked to be set in the SET PRES routine, the software has the characteristics of the PCU stored in memory. Since the S100TM's behavior is repeatable, the values stored should continue to accurately describe the circuit. However, there are two pitfalls to this solution that must be dealt with.

The first pitfall is that the S100TM is a differential device. As atmospheric pressure changes, so does the "zero" of the S100TM. This creates a problem since the output voltage for a given pressure will change as atmospheric pressure changes. Consider the example shown in Figure 5.16. Figure 5.16(a) shows atmospheric pressure at 12 PSI and output pressure at 30 PSI. Therefore, the differential pressure is 18 PSI. Eighteen pounds per square inch is converted to 2 volts by the S100TM. Assume atmospheric pressure rises to 13 PSI, as shown in Figure 5.16 (b), and the user requests 30 PSI at the output. From the calculations above, we "know" that to produce 30 PSI at the output a 2 volt signal must result at the output of the S100TM. Therefore, the DAC outputs 2 volts and the S100TM matches it and the differential output is again 18 PSI. However, the actual pressure is 31 PSIA. To avoid this pitfall, the present atmospheric pressure, via the SolartronTM, is stored with the tables. Therefore, when the user defines a pressure to output, this change in pressure can be taken into account.

The second pitfall is that since the DAC has 65,535 possible input values, if output pressures were stored for all possible inputs the tables would fill all addressable memory locations. However, this problem can be solved. Experimentation has shown that the curve can be approximated to be piecewise linear for a change in DAC counts of 1000D.
Figure 5.16. Effect of atmospheric pressure on differential transducers.
Therefore, to limit the size of the tables without significantly reducing their usefulness, each pressure reading in the table can be 1000D DAC counts from the previous reading.

There are three tables, one for each valve configuration. When building these tables, the pressure controller needs to use the Solartron™ but should not output this pressure to the user. After all, some of the pressure values that the PCU can set may be out of the pressure range of his or her device. To satisfy these conditions, valve 2 must be active while valve 3 is inactive.

When reading a pressure that is to be stored in one of these tables, care must be taken to make sure that the reading is as accurate as is possible. Therefore, the tolerance used by this routine is not what the user has defined but it is the minimum tolerance of the PCU.

After temporarily changing the tolerance, the user defined units of XDCR 1 are changed to PSI. This is done for two reasons. First, the Solartron™ software was designed to calculate pressure in PSI units. Therefore, if the user requests that the Solartron™ display units other than PSI, then just before the pressure is displayed the pressure is multiplied by a slope and offset which converts the units from PSI to those defined by the user. As a result of these extra calculations, it takes longer to store a pressure reading.

Another reason for storing table values in PSI units is that the routine that uses the tables is less complicated. If the tables were stored in the user-defined units, the units that they were stored in would also have to be stored. Then, each time the table is used the routine would have to read what units it was stored in.
For this reason, and to speed up this build tables routine, the tables are stored in PSI units. The pressure values are in IEEE standard four-byte floating-point format. Pressure is calculated using floating point arithmetic because of the necessary accuracy involved.

Since the Solartron™ reads absolute pressure, the tables are saved in absolute pressure. As a result, XDCR1’s MODE must temporarily be changed to ABSolute MODE. Remember, the PCU uses XDCR1 for information about the virtual transducer configuration.

The tables will be stored in memory as shown in Figure 5.17. The first table formed will be used to set pressures below atmospheric. Therefore, valve 0 is off and valve 1 is active.

To avoid overshooting a desired pressure which is below atmospheric, the output pressure should start at atmospheric and decrease until it reaches that desired pressure. To reach that pressure as fast as is possible without overshoot, the tables need to be as accurate as is possible. Since experimental results confirm that the PCU hardware experiences hysteresis, the tables should be formed by decreasing the pressure and then storing the result. This should continue until the DAC input is less than 1000D. At this point the lowest table value will have been calculated.

The first value sent to the DAC is 8000H which results in 0 volts out of the DAC, and an output of atmospheric pressure. The reason atmospheric pressure results is that the S100™ hardware feedback transducer produces a zero output voltage when reading zero
Figure 5.17. Tables stored in memory.
differential pressure. Remember that in order for the output pressure to stabilize, the voltage coming from the S100™ must match that on the output of the DAC.

After reading the pressure from the Solartron™, that four-byte floating-point value is stored in the highest memory location of the low table and the table checksum value is updated.

At this point a new DAC value is calculated by subtracting 1000D from the previous value. If 0 is reached, the low table is complete. Otherwise, the new pressure is stored in the next lower available memory location. This process continues until the low table is complete.

To produce pressures above atmospheric, values are added to 8000H. However, to set pressures above atmospheric pressure an air compressor or high pressure source is necessary. The high pressure source inputs to the HSC and the vacuum pump is disconnected - valve 0 active and valve 1 inactive.

The high table is formed much like the low table. As before, the absolute value of the voltage that comes from the DAC must increase for the table values to be accurate. Therefore, the low value of this table corresponds to a small positive voltage from the DAC. This value results from a 8000H + 1000D sent to the DAC. Each term thereafter results from 1000D added to that. The table is complete when adding 1000D causes an overflow in the 16 bit unsigned integer.

The middle table is used to set pressures near atmospheric pressure. Since the PCU requires air flow to accurately control air pressure, it is necessary to have a high pressure
source as well as a vacuum active in setting pressures near atmospheric. Otherwise, flow would have to be nearly cut off. For example, assume that the user requests to set an output pressure just above atmospheric pressure. If the values were configured such that the high pressure source was active and the vacuum was inactive, the HSC valve would have to nearly cut off the high pressure source to relieve the pressure in the pneumatics. If the vacuum was active, however, it would be able to pull the air out of the pneumatics and the HSC could let more air flow through the pneumatics. This increases air flow rate and controllability. As a result, the middle table is formed using both the high pressure source and the vacuum pump - valve 0 and 1 active. This table contains two halves. The half containing pressures below atmospheric pressure and the half containing pressures above atmospheric pressure. The low (high) half is formed the same way that the low (high) table is formed.

There are only two minor differences between the middle table and the other two tables. The first difference is that the DAC input step is 200D instead of 1000D, and the second is that the size of these tables was calculated through experimentation. A 200D step was chosen because setting pressures near atmospheric turns out to be more difficult. By using a smaller step between table values, the piecewise linear approximation of the function tends to be a more accurate.

The size of the middle, low and high tables was calculated through experimentation. The only requirement of the middle table is that it allow the user to set pressures near atmospheric. Therefore, it must just slightly overlap the pressures that the low and high tables can set.
Experimentation has shown that by using a step in DAC input of 200D, 32 readings sufficiently overlap the low and high table pressure values.

5.3.3 User Defined Output Pressure Limits (LIMITS)

The LIMITS routine allows the user to define the minimum and maximum values that should be sent to the output. The primary purpose of this routine is to protect the user’s transducer which may be connected to the output. Therefore to use this routine properly, the user should input the minimum and maximum pressures that their device can accurately sense without being damaged.

To select the LIMITS option the user can input the appropriate command from the host computer, or he or she can scroll through the CONTROL menu until the front panel alphanumeric display displays "CONTROL LIMITS". If the front panel option is chosen the user should then press ENTER to execute the LIMITS routine.

The first thing the routine does is read the previously stored user-defined maximum pressure, as shown in the pseudo code in Figure 5.18. This value is stored as a floating point number in the "PCU flags and configuration information" section of battery-backed RAM. Once the value is read it is converted to binary coded decimal and is output to the front panel. If that pressure was 40 PSI, "MAX P 40.000000" would be displayed on the front panel of the Sonix™ Display.
Initialize data structure pointer to zero, IDP = 0.
VALUE = UserPCUmax.
Point to Rngelist, multiply IDP by 8 and add the result to Rngelist.
The result of the calculation points to the beginning of a list of eight letters. Display them on the top
display of the front panel.
Point to UserPCUmax, multiply IDP by 32 and add the result to UserPCUmax.
The result of the calculation, VALUE, points to the beginning of a four byte floating point value.
Convert the floating point number to BCD and display it on the bottom display of the front panel.
Blink the bottom display.
Get input.
If input is an arrow key, IDP is changed from 0 to 1 or 1 to 0 and the routine jumps to 1.
If input is a number, continue receiving numbers until ENTER is pressed or eight numbers have been
input. Display the number while it is input. When ENTER is pressed the number has been entered.
Store it at VALUE and update checksum. Jump to 1.
If CLEAR is pressed exit the routine.

Rngelist:  "MAX P   "
          "MIN P   "
UserPCUmax: 4 byte floating point maximum pressure stored at this label in memory.
UserPCUmin: 4 byte floating point minimum pressure stored at this label in memory.

Figure 5.18. Pseudo code for the LIMITS routine.
At this point, the pressure value blinks indicating that the value can be edited. If the user wishes to edit that value, he or she must press the appropriate number keys followed by an ENTER. Again the entire number will blink. The value that the user inputs is not compared with anything to make sure it is a valid number. It is up to the user to input an accurate value. The value should be in the same units and mode as XDCR 1.

If the user wishes to change the minimum pressure value, he or she must press one of the arrow keys or enter the appropriate command from a host computer. If the front panel option is utilized and the present minimum pressure value is 10 PSI, the front panel will display "MIN P 10.000000".

The process of entering this value is the same as entering the maximum pressure value. The only difference is that this value is stored at the memory location allocated for the user-defined minimum pressure value.

Note that pressing the arrow keys merely scrolls the user through the two choices, it does not require the user to input a pressure value. Therefore the user is able to view the values without requiring that they be changed.

Once the user has viewed or defined the pressure values desired, the CLEAR key may be pressed to exit execution.
5.3.4 Set Pressure Tolerance (TOLERNCE)

The TOLERNCE routine allows the user to set control pressure tolerance. In other words, when the user defines a pressure to be output the actual pressure that is output must remain within that tolerance. If the user does not need the pressure controlled to the highest accuracy possible, the user can define a tolerance that better fits his or her needs. The smaller the tolerance, the more often the PCU software will have to modify the DAC value send to the PCU board. It modifies the DAC as it tries to return the pressure to a value within the desired tolerance.

To select the TOLERNCE option the user can input the appropriate command from the host computer, or he or she can scroll through the CONTROL menu until the front panel alphanumeric display displays "CONTROL TOLERNCE". If the front panel option is chosen the user should then press ENTER to execute the TOLERNCE routine.

The first thing the routine does is read the minimum allowed tolerance, as shown in the pseudo code in Figure 5.19. This value is stored as a floating point number in the "PCU flags and configuration information" section of battery-backed RAM. It is calculated in one of the initialization routines and results from information regarding the pressure range of the hardware. The value is in PSI units and absolute mode.

Since the user-defined tolerance should be in XDCR 1 units and mode, the minimum allowed tolerance is converted to XDCR 1 units and mode. This minimum value will be used later to compare the value entered to the user.
Get the minimum allowed tolerance which is stored in PSI units and absolute mode.
Multiply it by XDCR 1 units.
If XDCR 1 is in differential mode, subtract atmospheric pressure from it.
The result is used by this routine as the minimum tolerance.
Display "TOLERANCE" on the top display of the Sonix™ Display.
1. Get the floating point user-defined minimum tolerance from memory.
   Convert it to BCD and display it on the bottom display of the Sonix™ Display.
   Blink the tolerance value and wait for input from the keypad.
   If CLEAR is pressed exit the routine.
   If the input is a number, continue receiving numbers until ENTER is pressed or eight numbers have been
   input. Display the number while it is input. When ENTER is pressed the number has been entered.
   Compare this value with the minimum tolerance calculated above.
   If the user-defined tolerance is o.k. jump to 2.
   Otherwise, first display "TOLERANCE TO SMALL" then store the minimum allowed tolerance at
   TOLERANCE.
   Jump to 1.
2. Store it at TOLERANCE and update checksum. Jump to 1.

Figure 5.19. Pseudo code of the tolerance routine.
Next, the user-defined tolerance is read. This value is stored as a floating point number in the "PCU flags and configuration information" section of battery-backed RAM. Once the value is read it is converted to binary coded decimal and is output to the front panel. If the tolerance was 0.05 PSI, the front panel of the Sonix™ Display would display "TOLERANCE 0.0500000".

The user-defined tolerance blinks in the bottom display and the routine waits for input from the keypad. If CLEAR is pressed the routine is exited and the Sonix™ Display returns to its normal mode of operation.

If the user inputs a number, the routine continues receiving numbers and displaying them on the bottom display. This continues until ENTER is pressed or the bottom display is filled - eight numbers are entered. When ENTER is pressed the number has been entered. It is compared to the minimum tolerance computed earlier in the routine. If the user-defined tolerance is larger than the minimum, that tolerance is stored at TOLERANCE and the memory checksum is updated. Execution returns to nearly the beginning of the routine and displays the new tolerance value. If the user wishes to exit the routine, he or she must press CLEAR.

If the user inputs a number that is less than the minimum tolerance, "TOLERANCE TO SMALL" is displayed on the front panel. Execution returns to near the beginning of the routine, but this time the minimum tolerance is displayed. This value is not stored as the new tolerance value; it is merely displayed.
5.3.5 Set Output Pressure (SET PRES)

The SET PRES routine is the main routine of the PCU. Its first responsibility is to get the input from the user. If that value is in the range of pressures that are allowed to be output then it sets the output pressure.

To select the SET PRES option the user can input the appropriate command from the host computer, or he or she can scroll through the CONTROL menu until the front panel alphanumeric display displays "CONTROL SET PRES". If the front panel option is chosen the user should then press ENTER to execute the SET PRES routine.

The first thing the routine does is reads the last pressure value sent out, PCU_IN, as shown in the pseudo code in Figure 5.20. This value is stored as a floating point number in the "PCU flags and configuration information" section of battery-backed RAM. It is in the units and mode of XDCR 1.

Next, "OUT IS " is displayed on the top display of the Sonix™ Display and the floating point pressure value is converted to binary coded decimal and is displayed on the bottom display. The pressure value is displayed using the format of the top display.

The reason for using the format of the top display is that the Sonix™ Display is configured to display the Solartron™ used by the PCU when the display is in normal display mode. As a result, the top display will be configured to display the number of significant digits that the user requests. Therefore, for the purpose of consistency, the same format is used to display the last pressure that was output.
Get the floating point value of the last output pressure which is stored in XDCR 1 units and mode.
Display “OUT IS “ on the top display of the Sonix™ Display.
Convert floating point output pressure to BCD and display it on the bottom display of the Sonix™ Display. The value should be in the display format of the top display. (Note that in normal display mode, XDCR 1 is displayed in the top display)
Blink the value in the bottom display and wait for input from the keypad.
If CLEAR is pressed exit the routine.
If a digit or decimal point is pressed, display them on the bottom display as they are pressed. However, count the number of decimal places that have been entered after the decimal point. If the accuracy of the input value is more accurate than the TOLERANCE, replace the second to last digit pressed by the last digit pressed. Continue replacing the last digit until the ENTER key is pressed. The resulting value is the new output pressure. It should be in XDCR 1 units and mode.
Convert the hardware-defined maximum pressure allowed to XDCR 1 units and mode.
Compare the new desired output pressure to the hardware and user-defined maximum pressures. If it is smaller than both of these maximums then continue. Otherwise, display “OUT IS TOO BIG” on the front panel and return to 1.
Compare the new desired output pressure to the user-defined minimum pressure. If it is larger than the minimum allowed pressure then continue. Otherwise, display “OUT IS TOO SMALL” on the front panel and return to 1.
Save the new output pressure as a floating point number at PCU_IN and the checksum is updated.
Convert the desired output to BCD and display it on the bottom display of the front panel. It should be displayed using the format of the top display - that of XDCR 1.
Convert the desired pressure to PSI units and absolute mode.
Compare the desired pressure with the pressure values found in the look-up tables.
Two consecutive table values in one of the tables should have the characteristics that one pressure is above that desired, and the other below it. If not, scroll the message “MIN TBL VALUE IS ****. **** is the smallest table value in the low table. Jump to 1.
If XDCR 1 is in differential mode, take a new atmospheric pressure reading.
Depending on which table the values came from, turn on the valves necessary to output the desired pressure.
Turn off the Real Time Clock (RTC) one second interrupt.
Clear the CALING flag which indicates that output pressure is not presently being controlled.
Subtract the last atmospheric pressure from atmospheric pressure when the tables were stored. Call the result atm_diff.
Using the table values that surround the desired pressure, calculate a slope and offset.
Subtract the desired pressure from atmospheric pressure and multiply the result by 98%. Then, add the result to atmospheric pressure.
Add this new result to atm_diff, multiply by the slope and the offset. The result is the floating point version of new DAC value.

Figure 5.20. Pseudo code of the set pressure routine (continued on next page).
2 Convert the floating point DAC value to HEX and send it to the DAC. 
Read the resulting output pressure and display it in the top display in XDCR 1 units. 
Convert the value read to PSI units and absolute mode. 
is the difference between the desired pressure and the actual output within the minimum tolerance? 
Jump to 3 if so. 
Multiply the difference by 98% and the slope. Add the result to the last DAC value sent. The result is the new floating point DAC value. 
Jump to 2. 
3 Save the following data in the "Data flags and configuration information" section of RAM: Floating point DAC value, HEX DAC value, Desired pressure, floating point "forward" slope, and last pressure read. 
Turn on the Real Time Clock one second interrupt. 
Exit the routine.

Figure 5.20. Pseudo code of the set pressure routine (continued).
The last output pressure begins to blink indicating that the Display is awaiting an input. If the CLEAR key is pressed the routine is exited. If a digit or decimal point is pressed, they are displayed on the bottom display as the key is pressed. Once a decimal point is pressed a counter keeps track of how many numbers after the decimal point have been pressed. The reasoning behind this is that the user is not allowed to define an output pressure that has more significant digits than the tolerance. This forces the user to realize that the PCU will not control output pressure to any greater significance. If the user inputs another digit after the least significant digit is input, the new digit overwrites the last digit entered. This continues until the ENTER key or CLEAR key are pressed. If ENTER is pressed the resulting value is the new output pressure. It is displayed in the bottom display again using the format of the top display.

After the new desired output pressure has been entered, it must be checked to make sure it falls within the possible range of outputs. First the hardware-defined maximum pressure is converted to the units and mode of XDCR 1 so that the numbers are in the same units. Then the desired pressure is compared with that pressure, the user-defined maximum pressure, and the user-defined minimum pressure. If the desired output is larger than one of the maximum allowed pressures, "OUT IS TOO BIG" is displayed on the front panel. If the desired pressure is smaller, "OUT IS TO SMALL" is displayed. If either one of these error conditions occurs, the program returns to the beginning of the routine and awaits an input.

If the desired pressure satisfies these requirements, it is saved in memory as a floating point value at a location called PCU_IN and the "Display and configuration information section" checksum is updated. Next the desired pressure value is converted from floating
point to binary coded decimal and is displayed in the bottom display of the Sonix™ Display using the format of the top display. The reason for using the format of the top display is the same reason as was previously explained.

After displaying the desired pressure value, its units and mode are converted to PSI units and absolute mode. The units and mode of the desired pressure are assumed to be that of XDCR 1 since XDCR 1 is the Solartron™ which is used by the PCU.

The desired pressure is converted to PSI units and absolute mode because it is now going to be compared to the values found in the look-up tables that were stored in PSI units. If the desired pressure can be set by one of the valve configurations, two consecutive table values in one of the tables should have the characteristics that one pressure value is above and the other is below that desired. If two such values cannot be found, the PCU is unable to set that pressure value. If the desired pressure is smaller than all of the table values, "MIN TBL VALUE IS ****" scrolls across the display, where "****" represents the value of the lowest pressure value in the tables and is found at the first location of the low table. If the desired pressure is larger than all tables values, "MAX TBL VALUE IS ****" scrolls across the display, where "****" represents the value of the highest pressure value in the tables and is found at the last location of the high table. After displaying one of these messages, the routine returns to the beginning and awaits input of another output pressure value.

If the desired pressure is between two consecutive pressure values in a table, that pressure can be sent to the output. First the routine temporarily stores these two table values for use in calculating the slope and offset of the piecewise linear curve that characterizes the control circuitry. Next, the valves must be configured correctly to
enable the necessary pressure sources needed to set the desired output pressure. The appropriate valve configuration is easily determined because it, by necessity, is nearly the same configuration which formed the table from which the two consecutive pressure values came, the only difference being that when the tables were formed valve 3 (see Figure 5.9) was inactive so output pressure would not reach the user’s transducer. In this case valve 3 will be active.

At this time the values are configured and 8000H, or 0 volts, is sent to the DAC. The reason 0 volts is sent to the DAC is to force the output pressure to return to atmospheric pressure before an attempt is made to set the next pressure. Due to the hysteresis effects that this circuit exhibits the desired pressure could be overshot if the output pressure is not returned to atmospheric pressure before the next pressure is set. For example, assume that user requests a pressure X, above atmospheric pressure, as the desired output pressure. The routine then looks for that pressure in the look-up tables and finds it in the high table. The dots on curve LQ of Figure 5.21 illustrate these table values. Next the routine uses a piecewise linear approximation to calculate A as the DAC value necessary to set pressure X and end up at M. Since we started at atmospheric pressure at the output, the table is an accurate description of the circuit and the output pressure is set properly, with possibly some minor tweaking of the DAC.

Next, assume that the user inputs a new desired output pressure, Y, but does not return the output pressure to atmospheric. The routine then uses the tables to calculate the necessary DAC value and sends B to the DAC. However, instead of ending up at R the output pressure is set to N. The software must then continue decreasing the DAC until 0 is reached resulting in the correct output pressure Y.
Figure 5.21. Example of the characteristics of PCU without rezeroing.
Finally, assume the user again requests a new output pressure, Z, without returning the output pressure to atmospheric. The routine again uses the tables to calculate the necessary DAC value and send C to the DAC. However, instead of ending up at S the output pressure is overshoot and results at P.

As a result of the previous example, one can see that the output must return to atmospheric pressure before the next pressure is attempted to be output. Therefore, before any of the calculations take place and the appropriate DAC value is calculated, the DAC is rezeroed which enables the output pressure to return to atmospheric while these calculations take place.

Now that the desired pressure is found to be a pressure that can be set, a couple of other things must happen before the DAC value is calculated. One is that the one second interrupt sent to the NMI interrupt pin of the 6809 microprocessor must be masked. This interrupt is sent to the microprocessor by the real time clock. It indicates that an output pressure is being sent out and needs to be checked once a second to make sure the actual output pressure is within the user-defined tolerance. Since we are calculating a new pressure to output, the old output pressure is no longer desired, therefore there is no longer a need to control it.

The other task is to check to see if XDCR 1 is in differential mode. If it is, atmospheric pressure must be read to update what is stored as the "present" atmospheric pressure.

At this point, the routine begins calculating the value to send to the DAC. This requires that the curve, shown in Figure 5.22, be considered linear between the two table values.
Figure 5.22. Curve characterizing the hardware.
that "surround" the pressure value that the user has requested, \( X_D \). By doing so, the slope and offset can be calculated thus defining that region of the curve.

The calculations begin with the slope. The values used to calculate slope come from the tables. They will be represented by \( X_1, X_2, Y_1, \) and \( Y_2 \), as shown in Figure 5.22. It should be noted that Figure 5.22 could represent any one of the 3 tables. If it were to represent the high table, \( X_2 \) would be the table value for a higher pressure than \( X_1 \). If it were to represent the low table, \( X_2 \) would be the table value for a lower pressure than \( X_1 \). In both of these cases and in the two middle table cases, \( X_1 \) is always closer to atmospheric pressure than \( X_2 \). Similarly, the DAC value \( Y_1 \) is always closer to 8000H then \( Y_2 \).

First the slope, \( M \), and then the offset, \( B \), is calculated, as is shown in Figure 5.23. Next \( \Delta X_{DIFF} \) is computed. It is 98% of the change in pressure necessary to reach the desired pressure, \( X_D \), from the initial atmospheric pressure, \( X_{ATM} \). The reason for computing this term is that the first attempt at setting the desired output pressure should not try to reach the desired pressure in one shot. Instead, to avoid possible overshoot due to slight repeatability problems in the hardware, the first attempt should try to undershoot the desired pressure. It was found through experimentation that on occasion the first attempt would overshoot the desired pressure if an attempt was made to reach the desired pressure in one shot. However, if 98% of the desired change in pressure was attempted no such overshoot was detected.

After calculating \( \Delta X_{DIFF} \) atmospheric pressure is added to it which results in \( X_O \), the actual pressure value that will be attempted. However, there is one other component that must be considered before the DAC value is calculated. That is the effect that a varying
\[
M = \frac{Y_2 - Y_1}{X_2 - X_1}
\]

\[
B = Y - M \times X_1
\]

\[
\Delta X = X_0 - X_{ATM}
\]

\[
X_{DIFF} = 0.98(\Delta X)
\]

\[
X_0 = X_{DIFF} + X_{ATM}
\]

\[
\Delta Y = M(Y_{ATM} - X_{ATM,TBL})
\]

\[
X_0 + \Delta Y + B
\]

**Figure 5.23.** Equations necessary to calculate the DAC value.
atmospheric pressure has on the voltage emitted by the S100\textsuperscript{TM} of the hardware feedback loop, as shown in Figure 5.16. If atmospheric pressure rises (falls) the smaller (larger) the voltage emitted by the S100\textsuperscript{TM}. Therefore, the required DAC voltage is smaller (larger), as shown in Figure 5.24. That change in DAC can be calculated by subtracting the atmospheric pressure when the tables were formed from the present atmospheric pressure and multiplying the result by the slope of the curve. The resulting value is \( \Delta Y \). The final compensated DAC value to send out is the sum of the slope times \( X_o \) plus \( \Delta Y \) plus offset \( B \).

Once the DAC value has been calculated, it is converted from floating point to hexadecimal and sent to the DAC. The routine triggers the Solartron\textsuperscript{TM} and waits for the returning pressure value to be read at the output. The value read is displayed in the top display and is in the units and mode of virtual transducer XDCR 1.

The output pressure value, converted to PSI units and absolute mode, is subtracted from the desired pressure value which was previously converted to PSI and absolute mode. If the absolute value of this difference is less than the user-defined tolerance, converted to PSI, the output pressure is "accurate" enough and the DAC need not be altered. However, if the difference is greater than the tolerance a new DAC value must be calculated.

If the output is not within tolerance, the difference between the desired pressure, \( X_D \) and the actual output pressure, \( X_A \), is multiplied by the slope, \( M \), as shown in Figure 5.25. The resulting value should be the change in DAC, \( \Delta Y \), necessary to reach the desired pressure in one shot. However, for the same reasons as before this value is multiplied by 98\% to avoid possible overshoot. The result is added to the last DAC value sent, in
Figure 5.24. Hardware characteristics as affected by changes in atmospheric pressure.
\[ \Delta X = X_D - X_A \]
\[ \Delta Y = M \cdot \Delta X \]
\[ Y_{OUT} = Y_A + \Delta Y \]

\[ \text{(a)} \]

\[ \text{VOLTAGE} \]
\[ \hat{Y}_D \]
\[ \hat{Y}_A \]
\[ \text{PRESSURE} \]
\[ X_A \]
\[ X_D \]

\[ \text{(b)} \]

*Figure 5.25.* Partial graph of the PCU characteristics.
floating point format, producing the next DAC value to be sent. This value is converted to hexadecimal form and the routine jumps up to the point in the routine where the DAC value is sent to the DAC. This part of the routine continues looping until the desired pressure is within the user-defined tolerance.

Notice that regardless if the desired pressure is overshot or undershot, the above routine will work nearly equally well. If the output pressure overshoots, the slope will not accurately describe the circuit due to hysteresis. However, based on experimentation, overshoot rarely occurs and if it does, the slope will be accurate enough such that the inaccuracy of the slope is not noticed.

Once the desired pressure is within tolerance the following circuit parameters are saved for use by the CHECK_P routine to be discussed in the next section: DAC value in floating point, desired pressure in the XDCR 1 units and mode, slope in floating point format and in XDCR 1 units, and last pressure reading in XDCR 1 units and mode. CALING, a flag indicating that the output is in calibration mode, is also set to 0FFH. In other words, the PCU is controlling the output pressure.

The only remaining task that this routine must perform is to enable the one second interrupt of the Real Time Clock (RTC). This interrupt is connected to the NMI pin of the 6809 microprocessor and indicates that the CHECK_P routine should be executed.
5.3.6 Update Output Pressure

The update output pressure routine is activated by the NMI interrupt service routine as a result of an interrupt from the Real Time Clock. The object of this routine is to compare the last output pressure reading to the desired pressure. If the absolute value of the difference between them is less than the user-defined tolerance then the output pressure does not need to be changed and the routine is finished. However, if the difference is too large, a new DAC value is calculated.

The way this routine calculates the new DAC value is similar to the SET PRES routine except that this routine uses two different slopes. If the actual output pressure is closer to (farther from) the atmospheric pressure than the desired pressure the "forward" ("reverse") slope is used.

The forward slope was previously calculated in the SET PRES routine and saved for use by this routine. Therefore, if the output pressure is outside of the tolerance boundary in the direction approaching atmospheric, this slope is used to update the DAC value such that the output pressure is within the boundary.

If the output pressure is out of tolerance such that it has moved farther from atmospheric pressure the "reverse" slope should be used. The reason for having a reverse slope instead of using the "forward" slope is hysteresis. The characteristics of the circuit are different when the DAC is decreased and thus the slope is different as was shown in Figure 5.21.
The only problem is that there exists no reverse slope the first time this routine executes after SET PRES executes. Therefore, the "forward" slope is used instead. Then, once the output pressure is returned to within tolerance the "reverse" slope can be calculated.

How this "reverse" slope is calculated as well as how the routine works is now discussed in a more detailed manner. To ease the explanation, examples are given.

Assume that the actual output pressure is closer to atmospheric than that desired. If this is the case, the new DAC value is calculated as in Figure 5.25. Notice that the change in DAC is not multiplied by 98% as was done in the SET PRES routine. The reason being that when the new ΔDAC value is calculated, it can result in a Δpressure that is nearly 100% larger than expected and the output pressure will still fit within the user-defined tolerance. Since the interrupt from the RTC occurs so often, the output pressure can extend only slightly beyond the tolerance before this routine is called to return the pressure to within tolerance as is shown in Figure 5.26. Assume that the actual output pressure is \( X_A \), barely outside of the tolerance. If the slope used to calculate a ΔDAC attempts to hit the desired pressure exactly, it could overshoot the desired Δpressure by nearly 100%, and the pressure would still be within tolerance.

Assume that the actual output pressure is farther from atmospheric than that desired. If this is the case, the new DAC value is calculated as above except that instead of using the "forward" slope the "reverse" slope is used.

Once the ΔDAC has been calculated, it is added to the previous DAC value and sent to the DAC. The resulting pressure is read by the Solartron™. If the pressure is still not
Figure 5.26. Actual output pressure just below the tolerance limit.
within tolerance the routine starts again from the beginning and calculates a new DAC value.

Otherwise, a new slope is calculated. If the initial pressure was closer to (farther from) atmospheric pressure than the desired pressure the new slope calculated is the "forward" ("reverse") slope. The new slope is calculated by dividing the ΔDAC by the Δpressure and averaging it with the previous slope stored in memory. The initial pressure and DAC values used to calculate the slope are the pressure and DAC values upon entrance to this routine.

After calculating the new slope, the following necessary information is stored in memory and the checksum is updated: the new "forward" or "reverse" slope, the last pressure read, and the present DAC value. This information is used by the routine the next time it is executed.

5.3.7 Stop and Vent Pressure (STOP & V)

The STOP & V routine is executed by the user when he or she picks the STOP & V option of the CONTROL menu. Its purpose is to stop controlling the output pressure and vent the output pressure to atmospheric pressure.

To stop controlling the output pressure, the Real Time Clock one second interrupt is disabled. This keeps the CHECK_P routine from executing which is responsible for checking on the output pressure and updating the DAC if necessary.
Next, the DAC is modified to output 0 volts. This tells the hardware to try to set the output pressure to atmospheric. Valves 0 and 1 are deactivated which disables the high and low pressure sources replacing them with flow to atmospheric pressure.

The valves stay configured in this way until the output pressure reaches atmospheric. To determine when atmospheric pressure is reached, the Solartron™ is continuously read and the last reading is compared with the present reading. If the difference between them is less than the minimum definable tolerance, 0.01% full scale, the output pressure is considered to be stable at atmospheric pressure. At this point, all valves are turned off and the routine finishes execution.

5.3.8 Stop and Clamp Pressure (STOP & C)

The STOP & C routine is executed when the user picks the STOP & C option of the CONTROL menu. Its purpose is to stop controlling the output pressure but still clamp the output pressure in the line.

When this option is selected, the front panel of the Sonix™ Display displays "TURN OFF & VENT" during its execution. To stop controlling the output pressure, the Real Time Clock one second interrupt is disabled. This keeps the CHECK_P routine from executing which is responsible for checking on the output pressure and updating the DAC if necessary. Even though the output is still controlled by the hardware feedback loop, the software loop is disabled. This must be done first because the next function of this
routine is to cut off the Solartron\textsuperscript{TM} from the pressure source. Therefore, if CHECK\_P was to execute and try to alter the DAC, the software feedback from the Solartron\textsuperscript{TM} would not exist and that routine would never complete execution.

To clamp the pressure at the output, valve 3, shown in Figure 5.27, is activated and valve 2 is turned off. This traps the pressure in the output line but still enables the pressure to be read by the Solartron\textsuperscript{TM}.

Next, the program delays to allow the valves to completely close before the hardware control is disengaged. Once the delay is complete, the DAC is modified to output 0 volts. This tells the hardware to try to set the output pressure to atmospheric. Valves 0 and 1 are deactivated which disables the high and low pressure sources, replacing them with atmospheric pressure bleeds.
Figure 5.27. Valve configuration in clamp-off mode.
CHAPTER 6

Analysis of Results

This chapter discusses the results of analysis of the pressure controller. These results are used to determine if it has met the functional and control design requirements outlined in Chapter 4. The first section discusses simulation of the hardware to determine which system parameters affect its performance. The second section indicates whether or not the pressure controller met system requirements.

6.1 Simulation

To determine the affects that various parameters have on the performance of the pressure controller, simulation of the circuit is implemented. System parameters are easily
modified and the output results plotted to give insight into how a particular parameter effects the circuit.

6.1.1 Simulation Model

There are three components of the pressure controller that can be modified to change its performance:

1) the air pressure supplies,
2) the output volume, and
3) the RC time constant of the integrator.

This section will describe the effects of these components as well as describe the performance of the pressure controller for the three valve configurations.

To simulate the first order effects on performance of the pressure controller hardware, the air path, air flow, and output volume had to be simulated. These pneumatics, shown in Figure 6.1(a), were simulated with the electrical circuit equivalent shown in Figure 6.1(b). \( V_{sS} \) and \( V_{S} \) represent the two pressure sources because they perform the same function. The pressure source (battery) provides a constant source of pressure (voltage) independent of the airflow (current) requirements of the pneumatics (circuit). If the high (low) pressure source is above (below) atmospheric pressure, \( V_{S} \) (\( V_{S} \)) is a positive (negative) value. The higher (lower) the pressure the supply can set, the larger (smaller) the voltage.
Figure 6.1. (a) Actual and (b) simulated air pressure pneumatics.
$R_{S1}$ and $R_{S2}$ represent the resistive effect that the HSC servo valve flap, shown in Figure 6.1 (b), has on air flow (current flow) from the high and low pressure sources. As the input current to the HSC increases, the flap moves to cut off flow out of the low pressure orifice. Thus the resistance to air flow (current) through that orifice increases while the resistance to air flow (current) through the high pressure orifice decreases. To simulate this effect, $R_{S1}$ and $R_{S2}$ are voltage controlled resistors, where the controlling voltage is $V(7)$, the voltage sent to the voltage-to-current converter of the HSC valve. The higher the voltage, the larger the output pressure. Note that in the actual circuit the current sent to the HSC valve controls the resistance of the flap but in simulation the linear conversion from voltage to current is not done because it is unnecessary.

The simulated pressure output for a given current sent to the HSC is shown in Figure 6.2. $V(7)$ represents the range of currents that could be sent to the HSC while the y-axis represents the pressure on the output line. The values of the axes will be explained later; at this point only the shape of the curve should be examined.

Note that when the HSC has a minimum current at its input, $V(7) = -15V$, the output pressure is at its minimum. This is because the HSC flap is blocking all airflow from the high pressure source ($R_2 = 0$ ohms). As the current sent to the HSC increases, the flap allows air (current) to flow from the high pressure (voltage) source which results in increased pressure (voltage) at the output.

When the maximum current is sent to the HSC valve, the flap cuts off airflow (current) through the low pressure (voltage) source ($R_1 = 0$ ohms) and the maximum pressure (voltage) is output.
Figure 6.2. Simulated HSC valve response verses input.
C1 represents the volume to be filled to a given pressure (voltage). The larger the volume the larger the capacitance. Air molecules are analogous to electrons. To fill a volume to a given pressure a certain number of molecules are needed. The more molecules the higher the pressure. The same is true for charge on the plate of a capacitor. The more charge, the higher the voltage.

The entire circuit used in the simulation is shown in Figure 6.3 and the Spice [5] model for the hardware is shown in Figure 6.4.

The values V+S and V-S depend on the pressure the user has selected. If the desired pressure is significantly above atmospheric pressure V+S is a positive voltage indicating positive differential pressure and V-S is zero volts indicating zero differential pressure. If the desired pressure is significantly below atmospheric pressure V-S is a negative voltage indicating negative differential pressure and V+S is zero volts indicating zero differential pressure. If the desired pressure is near atmospheric pressure V+S is a positive voltage and V-S is a negative voltage.

Assume that the user requests an output pressure significantly above atmospheric pressure. V+S is a positive voltage and V-S is grounded. For this example, V+S was chosen to be the 15 volts which corresponds to a pressure source that can set pressure up to the maximum pressure of the PCU. If V+S were greater that would represent that the supply pressure was greater than the maximum output that the PCU was designed to set. C1 was chosen be 100uF which, for this model, corresponds to a small output volume.
Figure 6.3. PCU hardware simulation schematic.
* Pressure Controller Hardware Simulation *

width out=80
.
*connections: non-inverting input
*     inverting input
*           positive power supply
*           negative power supply
*           output
*           
.subckt LM358 1 2 3 4 5
...
c1 11 12 2.887E-12
c2 6 7 30.00E-12
dc 5 53 dx
dc 54 5 dx
dip 90 91 dx
dln 92 90 dx
dp 4 3 dx
egnd 99 0 poly(2) (3,0) (4,0) 0 .5 .5
fb 7 99 poly(5) vb vc vlp vin 0 21.22E6 -20E6 20E6 20E6 -20E6
ga 6 0 11 12 188.5E-6
gcm 0 6 10 99 99.61E-9
iee 3 10 dc 15.09E-6
hlim 90 0 vlim 1K
q1 11 2 13 qx
q2 12 1 14 qx
r2 6 9 100.0E3
rc1 4 11 5.305E3
rc2 4 12 5.305E3
re1 13 10 1.845E3
re2 14 10 1.845E3
ree 10 99 13.25E6
ro1 8 5 50
ro2 7 99 25
rp 3 4 9.082E3
vb 9 0 dc 0
vc 3 53 dc 1.500
ve 54 4 dc 0
vlim 7 8 dc 0
vlp 91 0 dc 40
vin 0 92 dc 40
.model dx D(Is=800.0E-18)
.model qx PNP(Is=800.0E-18 Bf=166.7)
.ends

Figure 6.4. PCU hardware simulation circuit (continued on the next page).
Figure 6.4. PCU hardware simulation circuit (continued).
It should be noted that the exact values of $C_1$, $R_{S1}$, $R_{S2}$, $V_{+S}$, and $V_{-S}$ are arbitrary, however, certain relationships between these values must be maintained. First, $V_{+S}$ and $V_{-S}$ must maintain a linear relationship with their mechanical equivalents. For instance, if the user decides that $V_{-S} = 10$ volts is to represent a vacuum pump that can pull the pressure down to 5 PSI, then if the high pressure source can drive the output pressure to 50 PSI, $V_{+S}$ must equal 100 volts.

The second requirement that must be maintained is that $C_1$ must maintain a linear relationship with its mechanical equivalent. For instance, if the user decides that a 1uF capacitor corresponds to 1 ml of volume then 2uF corresponds to 2ml of volume.

$R_{S1}$ and $R_{S2}$ are determined through experimentation after relationships for $C_1$, $V_{+S}$, and $V_{-S}$ have been defined. These resistors represent the resistance of the tubing and the orifices. To determine the exact values of these resistors, the S100$^\text{TM}$ response of the actual pressure controller must be monitored for different desired output pressure values. After gathering this data, $R_{S1}$ and $R_{S2}$ are selected such that the simulation results match the actual circuits response for the respective desired output voltages.

In the simulations described in the next section, no attempt is made to relate $C_1$, $R_{S1}$, $R_{S2}$, $V_{+S}$, or $V_{-S}$ to any mechanical equivalents. Only the interrelationship between these components, and the effects of varying their values is explored.
6.1.2 Simulation Results

Assume the user requests a pressure that the DAC converts to 10 volts. The resulting output is shown in Figure 6.5, where V(3) is the DAC voltage, V(2) is the amplified S100 voltage and V(7) is the voltage sent to the linear voltage-to-current converter and into the HSC valve.

Since the volume is fairly small, the output pressure, represented by V(2), initially overshoots but quickly stabilizes. Notice that the voltage sent to the HSC is approximately 1 volt. That indicates that the flap is blocking both the high and low pressure sources approximately equally. Also, note that any pressure significantly above atmospheric pressure will produce a positive pressure and can be set by this pressure (voltage) source configuration.

If the user’s volume does not change but he or she requests a pressure significantly below atmospheric pressure, $V_{+S} = 0$ volts and, arbitrarily, $V_{-S} = -15$ volts.

As can be seen in Figure 6.6(a), the amplified S100 voltage initially overshoots since the user’s volume is so small, but it does quickly stabilize at the voltage, $V_3$, requested by the DAC. The voltage sent to the HSC also stabilizes, as is shown in Figure 6.6(b), indicating that the HSC flap is held at one spot.

If the user’s volume does not change but he or she requests a pressure near atmospheric pressure, $V_{+S} = 15$ volts and $V_{-S} = -15$ volts. As can be seen in Figure 6.7(a), the amplified S100 voltage initially overshoots since the user’s volume is so small, but it
Figure 6.5. Simulated response of the PCU hardware with $C_1 = 100\mu$F, $V_{as} = +15\text{V}$, $V_{ss} = 0\text{V}$, and $R_2 = 3740\text{ ohms}$ (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$.

Analysis of Results
Figure 6.6. Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_{+s} = 0V$, $V_{-s} = -15V$, and $R_2 = 3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$. 

Analysis of Results
Figure 6.7. Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_{+8} = +15\text{V}$, $V_{-8} = -15\text{V}$, and $R_2 = 3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$.

Analysis of Results
does quickly stabilize at the voltage, $V_3$, requested by the DAC. The voltage sent to the HSC also stabilizes, as is shown in Figure 6.7(b), indicating that the HSC flap is held at one spot.

Assume that the user changes the volume of the output to be filled. The result is that the response of the output pressure will change. Figures 6.8 and 6.9 show how the system will respond if the output volume (capacitance $C_1$) is increased from that shown in Figure 6.5. For the first example $C_1 = 1\text{mF}$. Notice that the overshoot is not as great but the frequency of oscillation is lower. The result is that the desired voltage is reached slower. The HSC flap also becomes less stable. This is due to the fact that small overshoots in the output pressure take longer to correct since more air must bleed off for a given $\Delta P$ given the greater the volume.

This fact is shown even more dramatically in Figure 6.9, where $C_1 = 100\text{mF}$. The circuit for this output volume is nearly critically damped. However, the HSC valve begins to oscillate substantially. If the output volume were even larger, the HSC valve flap would oscillate back and forth to its rails while keeping $V(2)$ stable.

Another parameter that can be varied that will effect the performance of the pressure controller is the supply pressures. Figure 6.10 shows the response of the circuit given a pressure source twice as large as that of Figure 6.5. Notice that the voltage, $V(7)$, sent to the HSC is smaller than that of the later. The reason for this is that when a higher pressure source is used to set a given pressure (capacitance voltage) more resistance must be applied to that source to reduce the air (current) flow. The greater the supply pressure, the harder it is to set the desired pressure. If the supply pressure is too large, the HSC will be unable to sufficiently block the current flow, and the desired pressure can
Figure 6.8. Simulated response of the PCU hardware with $C_1 = 1000\mu$F, $V_{+s} = +15V$, $V_{-s} = 0V$, and $R_2=3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$. 

Analysis of Results
Figure 6.9. Simulated response of the PCU hardware with $C_1 = 100\text{mF}$, $V_{+S} = +15\text{V}$, $V_{-S} = 0\text{V}$, and $R_2 = 3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$.

Analysis of Results

102
Figure 6.10. Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_{+} = +30V$, $V_{-} = 0V$, and $R_2 = 3740$ ohms (a) shows $V(2)$ and $V(3)$, (b) shows $V(7)$.
not be set. As a result, the optimal high pressure source produces a maximum output pressure approximately equal to the maximum pressure rating of the pressure controller.

The final option that can be varied is the RC time constant of the integrator. The smaller the time constant, the less the overshoot for a given volume as in shown in Figure 6.11. In Figure 6.11(a) R₂ equals 3740 ohms while in Figure 6.11(b) it equals 100 ohms. The only drawback for reducing the time constant is that the HSC will begin to oscillate significantly for a small output volume.

6.2 Did the Design Meet the Requirements?

Based on experimentation, the design met all requirements, provided that the output is not open-ended, i.e. there exists a finite output volume, and the output volume was sufficiently large. In addition, extra user-friendly features were added to many of the routines to give the user more information about the limitations of the hardware and the tables he or she created.

The way the hardware is designed, the output pressure will overshoot the desired pressure. However, the amount of overshoot is within the minimum tolerance. The hardware is very fast. The hardware feedback loop is not the limiting factor in determining how fast desired pressure is reached, rather it is the volume that is to be filled. In most applications the output will probably be a piece of tubing about ten feet long. If this is the case, the output will reach desired pressure in about six seconds. Of course, the larger the volume the longer it takes to reach the desired pressure.

Analysis of Results
Figure 6.11. Simulated response of the PCU hardware with $C_1 = 100\mu F$, $V_{+5} = +15V$, and $V_{-5} = 0V$. 

Analysis of Results
Experimentation has shown that there is no overshoot caused by the software algorithms. The possibility of overshooting was eliminated in the SET PRES routine since only 98% of the necessary ΔDAC is added to the present DAC value on each update of the DAC. By using this technique, the pressure is automatically undershot. The result is that it takes approximately three DAC updates before the output pressure is within the desired pressure range.

Once the pressure has been set, if the output pressure drifts outside of the tolerance band, it usually drifts toward atmospheric pressure. This occurs because the valves are turned on and begin to heat the manifold block which in turn heat the S100™. As the S100™ heats up the absolute value of its voltage output begins to increase. The hardware perceives this change in voltage as a change in output pressure. The result is that the HSC valve causes the absolute value of the differential output pressure to decrease. Eventually, the actual output pressure extends outside the tolerance boundary and the DAC value must be changed.
CHAPTER 7

Summary and Conclusions

This thesis introduced a microprocessor-based implementation of an air pressure controller. This chapter summarizes the work, discusses the advantages and disadvantages of microprocessor control for this application, and suggests possible enhancements.

7.1 Summary of this Work

The work presented in this thesis involves the design and implementation of a microprocessor-based air pressure controller. The hardware design uses many components of a similar pressure controller designed for another PSI system, the pieces were brought together and implemented in the system described in this thesis.
Simulation of the hardware was performed to determine the effects of various controllable parameters on the performance of the hardware circuitry. Software algorithms, which are all original, were developed to perform control and user interface functions.

### 7.2 Conclusions about Microprocessor Control for the PCU

The microprocessor is a flexible tool because it uses software that can be easily modified to perform different functions or restructure existing ones. This allows the system to become more powerful without requiring expensive hardware changes. The constants and tables may be changed on-line allowing for tuning of the controller for different operating conditions.

One of the major drawbacks of using the microprocessor as a feedback loop in the control algorithm is that it is slow in comparison to hardware feedback. The hardware feedback can respond in microseconds whereas the software feedback takes approximately 1.2 seconds. The Solartron™ Interface Card takes 0.6 seconds to send a new pressure reading and the control software requires two pressure readings to guarantee that the output pressure has stabilized. A detailed example of the uses of these two feedback loops was described in Chapter 6. It showed that if the output line's volume changes dramatically the software would try to react by sending a new DAC value. However, it is fairly slow to react. On the other hand, when the DAC value is sent to the hardware loop, the hardware loop reacts very quickly. One possible solution to
increasing the performance of the software feedback loop is to use a faster microprocessor in the place of the 6809 presently being used.

All in all, it is concluded that the advantages offered by the microprocessor far outweigh the drawbacks in the development of the air pressure controller. In fact, it is hard to imagine another solution to the problem given that the Solartron™ must be used to read the actual output pressure.

7.3 Suggested Design Enhancements

To help prevent the output pressure from changing due to temperature changes of the S100™, an insulator could be added between it and the manifold. The S100™ would output more accurate pressure readings resulting in a more stable output pressure.

Another option could be added to the pressure controller options. This option could allow the user to output a succession of pressures over a given amount of time. The user would input the output pressure range, number of values to output, and the time to output each pressure. The user would then be able to test the his device over a range of pressure values without having to manually input each pressure value.

As was stated previously, the pressure controller does not work well when the output is open-ended. However, there are instances when the user may wish to temporarily open-end the output and then close it off again. This would be fine if the DAC value was not changed to try and compensate for the temporary decrease in pressure. When the output

Summary
line closes off again, the pressure would return to the desired pressure. To prevent the DAC from being changed, a menu option could be added to disable the DAC update routine. Once the user picks this option, he or she can open-end the output as long as he or she wishes. Then after closing the output the DAC update routine could be reenabled.

The last suggested modification is that valve three of the controller pneumatics normally be open. The present configuration presents a problem. The normal configuration of valve 3 is such that the Solartron™ is cutoff from the output line. Therefore, if the Solartron™ is to be used just to read pressure and not with the pressure controller, valve 3 must be activated. This is not a good idea since the Solartron™ will often be used in this manner and valve 3 will be continuously active when it does not need to be. Changing this requirement will not eliminate any of the other functions presently offered by the pressure controller.
Bibliography


Appendix A

Hardware Schematic
Appendix B

Software listing

STX: .EQU $02
ETX: .EQU $03
ACK: .EQU $06
BS: .EQU $08
CR: .EQU $0D
XON: .EQU $11
XOFF: .EQU $13
NAK: .EQU $15
CAN: .EQU $18
DEL: .EQU $7F
EOIFLAG: .EQU $FF

; BLINKCOUNT: .EQU $8000
PRCENT_JMP: .EQU $0908 ; 98% the difference between last output and
; desired, will be the next output
PRCENT_DIF: .EQU $0100 ; .01% difference between the desired output
; and actual output -GTH

; .SUBTITLE 'Memory Map Definitions'
; .PAGE
; ******************************************************************************
;  * ERROR CODES & REASON MASKS *
; ******************************************************************************
ERR_NONE: .EQU 0
ERR_BAD_CMD: .EQU 1
ERR_BAD_TIME: .EQU 2
ERR_INTERNAL: .EQU 3
ERR_BAD_DATE: .EQU 4
ERR_BAD_DISP: .EQU 5
ERR_BAD_VALUE: .EQU 6
ERR_TOO_MANY: .EQU 7
ERR_TOO_FEW: .EQU 8
ERR_BAD_SCAN: .EQU 9
ERR_BAD_COEFF: .EQU 10
ERR_POD: .EQU 11
;
REASON_NONE: .EQU 0
BAD_DEVICE: .EQU 1
BAD_SLOT: .EQU 2
BAD_AVG: .EQU 3
BAD_UNITS: .EQU 4
BAD_SLOPE: .EQU 5
BAD_OFFSET: .EQU 6
BAD_FORMAT: .EQU 7
BAD_XDCR: .EQU 8
BAD_LETTER: .EQU 9
BAD_NUMBER: .EQU 10
INVALID_OPT: .EQU 11
MISSING_GROUP: .EQU 12
XDCR_ERROR: .EQU 13
BAD_BARO: .EQU 14
MISSING_DEV: .EQU 15
BAD_VALUE: .EQU 16
TIMOUT: .EQU 17

PAGE
;
**************************
;
* HELPFUL CONSTANTS *
;
**************************
TERSE_MODE: .EQU 1
VERBOSE_MODE: .EQU 0
;
DEFAULT: .EQU 0
FIXED: .EQU 1
FLOATING: .EQU 2
SCIENCE1: .EQU 3
SCIENCE2: .EQU 4
NDISPFMT: .EQU 2 ; formats 0-2 enabled
;
; CONTENTS CODES
; This number may be found in the 1st byte of a display configuration table.
; It indicates the desired contents of a display. Not the actual data, just
; its type.
;
PREAD: .EQU 0
PDIFF: .EQU 1
PSUM: .EQU 2
PMULT: .EQU 3
PDIV: .EQU 4
PAVE: .EQU 5
PERCENT: .EQU 6
TIME: .EQU 7
DATE: .EQU 8
WEEKDAY: .EQU 9
LABELOTHER: .EQU 10
LABELUSER: .EQU 11
;
;
PSIALT: .EQU 0
DIODE_Y: .EQU 1
PERIOD: .EQU 2
PSI: .EQU 3
BAR: .EQU 4
MBAR: .EQU 5
INH2O: .EQU 6
PSF: .EQU 7
KPA: .EQU 8
INHG: .EQU 9
MMHG: .EQU 10
USERUNITS: .EQU 11
ALTITUDE: .EQU 12
.PAGE
PCU RANGES

DIG6PSI: .EQU 0
DIG15PSI: .EQU 1
DIG23PSI: .EQU 2
DIG30PSI: .EQU 3
DIG45PSI: .EQU 4
DIG65PSI: .EQU 5
DIG100PSI: .EQU 6
DIG200PSI: .EQU 7
DIG300PSI: .EQU 8
DIG500PSI: .EQU 9
S19PSI: .EQU 10
S38PSI: .EQU 11
S50PSI: .EQU 12
S150PSI: .EQU 13
S500PSI: .EQU 14
NONE: .EQU 15

INPUT RANGE

IN_MAX: .EQU 0
IN_MIN: .EQU 1

CONTROL OPTS

IN_ON: .EQU 0
V_OFF: .EQU 1
C_OFF: .EQU 2
IN_RANGE: .EQU 3
TOLERANCE: .EQU 4
BUILD: .EQU 5
RULES: .EQU 6

MODE OPTS

ABS: .EQU 0
DIFF: .EQU 1

DEVICE FLAGS

LOCAL: .EQU 0
REMOTE_488: .EQU 1
REMOTE_232: .EQU 2
MAX_DEV: .EQU 2
CNTL_OPTS: .EQU 6
RNGE_OPTS: .EQU 2
MODE_OPTS: .EQU 1
DISP_LINE1: .EQU 0
DISP_LINE2: .EQU 8
;
SONIXA: .EQU 0
SONIXB: .EQU 1
SOLARA: .EQU 2
SOLARB: .EQU 3
DQZA: .EQU 4
DQZB: .EQU 5
MAXXDCR: .EQU 5
;
;
************************************
; * PROGRAMMER POD DATA PACKET FLAGS *
; ************************************
;
MAXTRYS: .EQU 10 ; retries before xmit error
PODCONFIG: .EQU $491E ; 9600-N-8-1
;
;
KEYPRESS: .EQU 0
MESSAGE: .EQU 1
P_KEY: .EQU 2
REQUEST: .EQU 3
BLINK_KEY: .EQU 4
BLINK_CHR: .EQU 5
GET_MESSAGE: .EQU 6
GET_VALUE: .EQU 7
COMMAND: .EQU 8
EOL_OUT: .EQU 9
EOI_OUT: .EQU 10
ERROR_MESSAGE: .EQU 11
;
MSG_ABORT: .EQU 0
MSG_DATA: .EQU 1
MSG_DISP1: .EQU 2
MSG_DISP2: .EQU 3
MSG_DISP3: .EQU 4
MSG_DISPFF1: .EQU 5
MSG_DISPFF2: .EQU 6
MSG_TIMEDATE: .EQU 7
;
REQ_TIMEDATE .EQU 0
.PAGE
**GLOBAL RAM DATA STRUCTURE AND VARIABLE DEFINITIONS**

**TRIGGER_BLOCK:** This table of TRIG_BLOCK entries is used as a storage point for XDCR trigger actions requested by the virtual transducer trigger routines - All transducers in an entry are actually triggered "simultaneously".

```
MAX_TB: .EQU 16 ; 16 TRIG_BLOCK entries
TB_SONIXA: .EQU 0 ; Trigger value for SONIX-A
TB_SONIXB: .EQU 1 ; Trigger value for SONIX-B
TB_SOLARA: .EQU 2 ; Frequency preload & # avg for SOLARTRON-A
TB_SOLARB: .EQU 6 ; Frequency preload & # avg for SOLARTRON-B
TB_ATOD: .EQU 10 ; MUX load for A/D converter
TB_CLOCK: .EQU 11 ; Flag reading of RTC clock desired
TB_DONE: .EQU 12 ; XDCR bitmask for completion of TRIG_BLOCK
TB_LOG: .EQU 13 ; timestamp at which block should be triggered
TB_SIZE: .EQU 16 ; FINAL SIZE OF ONE ENTRY
TB_TOTAL: .EQU TB_SIZE*MAX_TB ; FINAL SIZE OF TRIGGER_BLOCK

**XDCR_FLAGS:** This table of flag entries marks the status of each virtual transducer's trigger.

```
MAX_XF: .EQU 7 ; 7 XDCR_FLAG entries
XF_WAIT: .EQU 0 ; Flag XDCR waiting for data ready
XF_DRQ: .EQU 1 ; Flag XDCR waiting for data valid
XF_TBN: .EQU 2 ; TRIG_BLOCK entry XDCR is triggered in
XF_DATA: .EQU 3 ; Data storage for XDCR
XF_DATA2: .EQU 7 ; Data storage for XDCR
XF_SIZE: .EQU 11 ; FINAL SIZE OF ONE ENTRY
XF_TOTAL: .EQU XF_SIZE*MAX_XF ; FINAL SIZE OF XDCR_FLAGS

**XDCR_CONFIG:** This table of configuration entries defines each virtual transducer's operating characteristics.

```
XC_DEV: .EQU 0 ; Physical device linked to virtual transducer
XC_ADDR: .EQU 1 ; Subaddress of physical device (**UNUSED-ESP**)
XC_UNITS: .EQU 2 ; Units code transducer should read
XC_AVG: .EQU 3 ; Averages code for number of averages
XC_TRIG: .EQU 4 ; Trigger equivalent of AVG code
XC_MTERM: .EQU 5 ; Slope value for units
XC_BTERM: .EQU 9 ; Intercept value for units
XC_LBL: .EQU 13 ; ASCII label for units
```
XC_FMT: .EQU 18 ; Default display format for units
XC_MODE: .EQU 20 ; XDCR is in absolute or differential mode
; XC_SIZE: .EQU 21 ; FINAL SIZE OF ONE ENTRY
XC_TOTAL: .EQU XC_SIZE*MAX_XF ; FINAL SIZE OF XDCR_CONFIG

********************************************************************************
* GLOBAL RAM DATA STRUCTURE AND VARIABLE DEFINITIONS *
********************************************************************************

; DISPLAY_FLAGS: This table of flag entries marks the status of each display
; line’s data.
;
MAX_DF: .EQU 4 ; 4 DISP_FLAGS entries
;
DF_DATA: .EQU 0 ; Data to be displayed
DF_NEEDS: .EQU 10 ; Bitmask of transducers needed
DF_COUNT: .EQU 11 ; Main loop cycles before timeout
DF_DONE: .EQU 13 ; Flag display function completed
;
DF_SIZE: .EQU 14 ; FINAL SIZE OF ONE ENTRY
DF_TOTAL: .EQU DF_SIZE*MAX_DF ; FINAL SIZE OF DISP_FLAGS

; DISPLAY_CONFIG: This table of configuration entries defines each display’s
; operating characteristics.
;
DC_CNT: .EQU 0 ; Contents of display
DC_FMT: .EQU 3 ; Format type for numeric output
DC_MSG: .EQU 4 ; Message for message output
DC_DGT: .EQU 12 ; Format digits for numeric output
;
DC_SIZE: .EQU 14 ; FINAL SIZE OF ONE ENTRY
DC_TOTAL: .EQU DC_SIZE*MAX_DF ; FINAL SIZE OF DISP_CONFIG

********************************************************************************
* MEMORY MAP LIMITS *
********************************************************************************

RAMBEGIN: .EQU $0000
RAMENDS: .EQU $1FFF
;
ROMBEGIN: .EQU $6000 ; Changed from 8000 - GTH
FP_ENTRY: .EQU $C03D
ROMENDS: .EQU $FFFF
;
; ****************************
; ***** RAM DEFINITIONS *****
; ****************************

; TRIG_BLOCK: .EQU   RAMBEGIN+0   ; TRIGGER BLOCK storage
XDCR_FLAGS: .EQU   RAMBEGIN+TB_TOTAL   ; XDCR status flags
DISP_FLAGS: .EQU   XDCR_FLAGS+XF_TOTAL   ; DISPLAY status flags

; VAR_STORE1: .EQU   DISP_FLAGS+DF_TOTAL

; GROUP_DATA: .EQU   VAR_STORE1+0   ; Array of grp readings
  (XF_TOTAL bytes)
GROUP_SET: .EQU   VAR_STORE1+XF_TOTAL   ; Array of grp settings
  (XC_TOTAL bytes)
GROUP_DEFN: .EQU   VAR_STORE1+XF_TOTAL+XC_TOTAL   ; Copy of XDCR_GROUP (1 byte)

; VAR_STORE2: .EQU   VAR_STORE1+XF_TOTAL+XC_TOTAL+1

; SOLARA_FREQ: .EQU   VAR_STORE2+0   ; Total averaging for SOLAR-A
  (4 byte)
SOLARA_ATOD: .EQU   VAR_STORE2+4   ; Latest A/D for SOLARTRON-A (2
byte)
SOLARA_AVG: .EQU   VAR_STORE2+6   ; # avg left for SOLAR-A (1
byte)
SOLARB_FREQ: .EQU   VAR_STORE2+7   ; Total averaging for SOLAR-B
  (4 byte)
SOLARB_ATOD: .EQU   VAR_STORE2+11   ; Latest A/D for SOLARTRON-B (2
byte)
SOLARB_AVG: .EQU   VAR_STORE2+13   ; # avg left for SOLAR-B (1
byte)
SYSTIMEDATE: .EQU   VAR_STORE2+14   ; Latest RTC reading (13 bytes)
ADPINGPONG: .EQU   VAR_STORE2+27   ; A/D MUX load for SOLARTRON (1
byte)
AD_LAST: .EQU   VAR_STORE2+28   ; Latest A/D load

; VAR_STORE3: .EQU   VAR_STORE2+29

; XDCR_SEL: .EQU   VAR_STORE3+0   ; Addr of xdcr edit selected (3
byte)
DISP_SEL: .EQU   VAR_STORE3+3   ; Addr of disp edit selected (3
byte)
REM_SEL: .EQU   VAR_STORE3+6   ; Remote selected for edit (1
byte)

; LEDSTATUS: .EQU   VAR_STORE3+7   ; Control register for LEDs (1
byte)
RTC_REG1: .EQU   VAR_STORE3+8   ; Copy RTC write-only ctrl reg
  (1 byte)
RTC_REG2: .EQU   VAR_STORE3+9   ; Copy RTC write-only ctrl reg
  (1 byte)
RTC_REG3: .EQU VAR_STORE3+10
            ; Copy RTC write-only ctrl reg
FACT_FSM:  .EQU VAR_STORE3+11
            ; State of factory mode fsm (1 byte)
GPIB_IOSAVE: .EQU VAR_STORE3+12
            ; Copy GPIB read-once reg (1 byte)
ACIA_SSSAVE: .EQU VAR_STORE3+13
            ; Copy ACIA read-once reg (1 byte)
XDCR_RDY:  .EQU VAR_STORE3+14
            ; Signals XDCRs ready (1 byte)
LAST_ERROR: .EQU VAR_STORE3+15
            ; Last error (2 bytes)
RMT.Exists: .EQU VAR_STORE3+17
            ; Flags remote card installed
SONIX.Exists: .EQU VAR_STORE3+18
                ; Flags SONIX card installed (1 byte)
SOLAR.Exists: .EQU VAR_STORE3+19
                ; Flags SOLARTRON card (1 byte)
DUMMY_RAM:  .EQU VAR_STORE3+20
                ; Dummy RAM address for editing

; PAGE
; ***************************************************
; ***** RAM DEFINITIONS (cont) *****
; ***************************************************

MAIN_PASS:  .EQU VAR_STORE3+24
            ; Counts when TB_DONE occured
CURRENT_TB:  .EQU VAR_STORE3+25
            ; Pointer to current TRIG_BLOCK
CURRENT_TBN:  .EQU VAR_STORE3+27
            ; Number of current TRIG_BLOCK
REMOTE_EXEC:  .EQU VAR_STORE3+28
                ; Remote command executed flag
                ; (If so retrigger displays)
                ; Signals POD communications (1 byte)
POD_FUNCTION:  .EQU VAR_STORE3+29
                ; Signals POD packet (1 byte)
POD_PACKET:  .EQU VAR_STORE3+30
                ; Len of POD packet in progress
POD_LEN:  .EQU VAR_STORE3+31
                ; # re—tries of last packet (1 byte)
POD_ERRCOUNT:  .EQU VAR_STORE3+32

; *****
VAR_CONTROL:  .EQU VAR_STORE3+33
                ; Variables used by pressure controller
PCU_EXISTS:  .EQU VAR_CONTROL+0
                ; Flags PCU card (1 byte)
PCU_RANGE:  .EQU VAR_CONTROL+1
                ; Storage for value of PCU type
PCU_IN:  .EQU VAR_CONTROL+2
                ; Storage for value of floating point input
                ; (4 bytes)
USER_PCUMAX:  .EQU VAR_CONTROL+6
                ; Storage for user defined max pressure
USER_PCUMIN:  .EQU VAR_CONTROL+10
                ; Storage for user defined min pressure
PCU_FP_MAXIN: .EQU VAR_CONTROL+14 ; (4 bytes)
            ; Storage for hardware defined
            ; max pressure
FPLS_TABLE: .EQU VAR_CONTROL+18 ; (4 bytes)
            ; Storage for low pressure
            ; lookup table
FPIS_TABLE: .EQU VAR_CONTROL+146 ; (32x4 bytes)
            ; Storage for middle pressure
            ; lookup table
FPMS_TABLE: .EQU VAR_CONTROL+274 ; (32x4 bytes)
            ; Storage for high pressure
            ; lookup table
TEMP_VAR: .EQU VAR_CONTROL+402 ; (32x4 bytes)
            ; Temporary variable
            ; (1 byte)
FP_PRCNT_JMP: .EQU VAR_CONTROL+403 ; Storage for FP value of
            ; percent of desired
            ; bytes)
FP_PRCNT_DIF: .EQU VAR_CONTROL+407 ; Storage for FP value of
            ; difference between
            ; allowed (4 bytes)
CALING: .EQU VAR_CONTROL+411 ; Flag tells if calibration in
            ; progress(1 byte)
LAST_ATM: .EQU VAR_CONTROL+412 ; Contains last atmospheric
            ; reading (4 bytes)
FPSLOPE: .EQU VAR_CONTROL+416 ; slope of PCU feedback (XDCR1)
            ; in FP (4 bytes)
XDCR1_FPP: .EQU VAR_CONTROL+420 ; XDCR1 last FP pressure read(4
            ; bytes)
LAST_DOUT: .EQU VAR_CONTROL+424 ; Last output in HEX (2 bytes)
DESIRED_P: .EQU VAR_CONTROL+426 ; Desired FP pressure sent out
            ; by PCU (4 bytes)
TOL_SET: .EQU VAR_CONTROL+430 ; Flag meaning that tolerance
            ; has been defined
ATM_TBL: .EQU VAR_CONTROL+431 ; if = ff (1 byte)
            ; Build Table routine was
            ; time the
executed. (4 bytes)
DAC_VALU: .EQU VAR_CONTROL+435 ; Last FP DAC value sent (4
            ; bytes)
IN_CONTROL: .EQU VAR_CONTROL+439 ; If in control routine = ff (1
            ; byte)
XDCR1_DIS: .EQU VAR_CONTROL+440 ; If XDCR1 is being displayed
then = 00 (1 byte)
HEX_DAC: .EQU VAR_CONTROL+441 ; Hex value of DAC last sent (2
            ; bytes)
TOLERANCE: .EQU VAR_CONTROL+443 ; FP value of user defined
tolerance (4 byte)
 BEFORE_SDM1: .EQU VAR_CONTROL+447 ; Flag saying in SDMMAIN before SDM1
 LAST_VALVE: .EQU VAR_CONTROL+448 ; Contains the last valve configuration(1 byte)
 G2_SET_BFOR: .EQU VAR_CONTROL+449 ; Flag saying that have going from pressure
 REVS_SLOPE: .EQU VAR_CONTROL+450 ; reverse slope(1 byte) going from pressure
 GT_SET_BFOR: .EQU VAR_CONTROL+449 ; Slope for calc DAC value when closer-to-ATM(4 bytes)
 XDCR1_FPP2: .EQU VAR_CONTROL+454 ; Second-to-last pressure read(4 bytes)
 PCUSUM: .EQU VAR_CONTROL+458 ; PCU related memory checksum
 PCUTABLESUM: .EQU VAR_CONTROL+459 ; PCU tables related memory
 ;******* - GTH

 IO_DCB: .EQU VAR_CONTROL+460 ; DCBs begin here -GTH

 FFQ: .EQU IO_DCB+0 ; Front panel Queue (10 bytes)
 IEEEQ: .EQU IO_DCB+10 ; IEEE-488 Queue (10 bytes)
 RSQ: .EQU IO_DCB+20 ; RS-232 Queue (10 bytes)
 PODRECQ: .EQU IO_DCB+30 ; Pod data received Queue (10 bytes)
 PODQ: .EQU IO_DCB+40 ; Pod data queue (10 bytes)
 PODAN: .EQU IO_DCB+50 ; Pod ACK/NAK signal queue (10 bytes)
 PODKEYQ: .EQU IO_DCB+60 ; Pod keystroke Queue (10 bytes)

 CONFIGAREA: .EQU IO_DCB+70 ; Instrument configuration storage

 ; *****************************
 ; ***** CONFIGURATION DATA STORAGE *****
 ; *********************

 MENUMODE: .EQU CONFIGAREA ; Terse/verbose flag (1 byte)
 ACIACONFIG: .EQU CONFIGAREA+1 ; ACIA baud/data/stop/parity (2 bytes)
 HELLO_1: .EQU CONFIGAREA+3 ; Hello message line 1 (8 bytes)
 HELLO_2: .EQU CONFIGAREA+11 ; Hello message line 2 (8 bytes)
 EXEC_DELIM: .EQU CONFIGAREA+19 ; Execution delimiter (1 byte)
 CMD_SEP_DELIM: .EQU CONFIGAREA+20 ; Command seperator (1 byte)
 GPIB_ADDRESS: .EQU CONFIGAREA+21 ; GPIB talk/listen address (1 byte)
BARO_OFFSET: .EQU CONFIGAREA+22 ; Current offset for barometer (4 bytes)
EOLSTRING: .EQU CONFIGAREA+26 ; Delimiter for EOL (16 bytes)
EOTSTRING: .EQU CONFIGAREA+42 ; Delimiter for EOT (16 bytes)
; DISP_CONFIG: .EQU CONFIGAREA+58 ; DISPLAY_CONFIG table
XDCR_CONFIG: .EQU CONFIGAREA+DC_TOTAL+58 ; XDCR_CONFIG table
; XDCR_GROUP: .EQU XDCR_CONFIG+XC_TOTAL ; SCAN GROUP definition (1 byte)
; CHECKSUM: .EQU XDCR_CONFIG+XC_TOTAL+1 ; Checksum for config area (2 byte)
; COEFFAREA: .EQU XDCR_CONFIG+XC_TOTAL+3

;*************************************************************************************************************************
;***** COEFFICIENT STORAGE AREA *****
;*************************************************************************************************************************
;
; DEVICE_CONFIG: This table defines the device configuration for each physical device connected to the Sonix Display.
;
;=============================================================================================================
DV_UNIT: .EQU 0 ; Device pressure units flag
;=============================================================================================================
DV_FLAG: .EQU 0 ; byte flag 500 lb (padded to longword)
DV_S: .EQU 4 ; longword S term
DV_D: .EQU 8 ; longword D term
DV_X: .EQU 12 ; longword X term
DV_Y: .EQU 16 ; longword Y term
DV_MSCON: .EQU 20 ; longword counter freq
DV_FRQPRE: .EQU 24 ; longword frequency counter preload
DV_VBITW: .EQU 28 ; longword A/D slope
DV_K00: .EQU 32 ; longword K terms 00-24
DV_K30: .EQU 92 ; longword K term 30 (last term)
MAX_DVSOLAR: .EQU 24 ; -- 24 entries
;=============================================================================================================
DV_TYPE: .EQU 0 ; Digiquartz model flag (padded to longword)
DV_DFRQPRE: .EQU 4 ; longword frequency counter preload
MAX_DVDQZ: .EQU 2 ; -- 2 entries
;=============================================================================================================
DV_SIZE: .EQU 96 ; FINAL SIZE OF LARGEST RECORD ENTRY
DV_TOTAL: .EQU MAXXDCR*DV_SIZE ;FINAL SIZE OF DEV_CONFIG
;
; DEV_CONFIG: .EQU COEFFAREA ; DEV_CONFIG table
DEV_SONIXA: .EQU DEV_CONFIG+SONIXA*DV_SIZE
DEV_SONIXB: .EQU DEV_CONFIG+SONIXB*DV_SIZE

Software 127
DEV_SOLARA: .EQU DEV_CONFIG+SOLARA*DV_SIZE
DEV_SOLARB: .EQU DEV_CONFIG+SOLARB*DV_SIZE
DEV_DQZA: .EQU DEV_CONFIG+DQZA*DV_SIZE
DEV_DQZB: .EQU DEV_CONFIG+DQZB*DV_SIZE

; COEFFSUM: .EQU COEFFAREA+DV_TOTAL ; GTH THINKS ADD -> +DV_SIZE
Checksum for coefficients

; ***********************
; * I/O MAPPED REGISTERS *
; ***********************
RTC_U_SEC: .EQU $4200 ; CLOCK: seconds (units)
RTC_T_SEC: .EQU $4201 ;
RTC_U_MIN: .EQU $4202 ; minutes (tens)
RTC_T_MIN: .EQU $4203 ; minutes (tens)
RTC_U_HRS: .EQU $4204 ; hours (units)
RTC_T_HRS: .EQU $4205 ; hours (tens)
RTC_U_DAY: .EQU $4206 ; CLOCK: days (units)
RTC_T_DAY: .EQU $4207 ; days (tens)
RTC_U_MOS: .EQU $4208 ; months (units)
RTC_T_MOS: .EQU $4209 ; months (tens)
RTC_U_YRS: .EQU $420A ; years (units)
RTC_T_YRS: .EQU $420B ; years (tens)
RTC_U_DOW: .EQU $420C ; day of week (0-6)

RTC_CTRL1: .EQU $420D ; CLOCK control register 1
RTC_CTRL2: .EQU $420E ; CLOCK control register 2
RTC_CTRL3: .EQU $420F ; CLOCK control register 3

; DSPDATA: .EQU $4400 ; Control register for displays
DSPCONTROL: .EQU $4401 ; Data register for displays
LEDDATA: .EQU $4600 ; RAM image of LEDDATA (write-only)

; GPIB_INT0: .EQU $5200 ; Interrupt mask/status 0
GPIB_INT1: .EQU $5201 ; Interrupt mask/status 1
GPIB_ADRS: .EQU $5202 ; Address status
GPIB_BUSS: .EQU $5203 ; Bus status
GPIB_AUX: .EQU $5203 ; Auxiliary cmd register
GPIB_ADDR: .EQU $5204 ; Address set register
GPIB_SP: .EQU $5205 ; Serial poll register
GPIB_PASS: .EQU $5206 ; Command pass-thru
GPIB_FP: .EQU $5206 ; Parallel poll register
GPIB_DATA: .EQU $5207 ; Data register

; ACIA_DATA: .EQU $5244 ; ACIA data register
ACIA_RESET: .EQU $5245 ; ACIA master reset (write-only)
ACIA_STATUS: .EQU $5245 ; ACIA status register (read-only)
ACIA_CMD: .EQU $5246 ; ACIA command register (16 bit)
I/O MAPPED TRANSDUCER DEVICES

SPS_1A: .EQU $5300 ; Serial-Parallel LSB (xdcr A)
SPS_2A: .EQU $5301 ; Serial-Parallel ISB
SPS_3A: .EQU $5302 ; Serial-Parallel MSB
SPS_1B: .EQU $5304 ; Serial-Parallel LSB (xdcr B)
SPS_2B: .EQU $5305 ; ISB
SPS_3B: .EQU $5306 ; MSB

SONIXPIA: .EQU $534C ; PIA for SONIX interface card
SONIXPIADA: .EQU $534C
SONIXPIADB: .EQU $534D
SONIXPIACA: .EQU $534E
SONIXPIACB: .EQU $534F

NOTE: THE SOLARTRON PERSONALITY CARD IS NOT FULLY DECODED (i.e. A3 and A2)
THEREFORE, DO NOT USE ANY OTHER ADDRESSES IN THE 53BX RANGE!!!!!! -GTH

SOLARPPIA: .EQU $53B0 ; PIA for SOLARTRON interface card
SOLARPPIADA: .EQU $53B0
SOLARPPIADB: .EQU $53B1
SOLARPPIACA: .EQU $53B2
SOLARPPIACB: .EQU $53B3

SOLARA_COUNT: .EQU $53C8 ; SOLARTRON-A counter preset
SOLARA_XFR: .EQU $53CA ; SOLARTRON-A transfer count
SOLARA_CLEAR: .EQU $53CB ; SOLARTRON-A counter reset

SOLARB_COUNT: .EQU $53CC ; SOLARTRON-B counter preset
SOLARB_XFR: .EQU $53CE ; SOLARTRON-B transfer count
SOLARB_CLEAR: .EQU $53CF ; SOLARTRON-B counter reset

SOLAR_1A: .EQU $53C8 ; SOLARTRON-A counter results MSB
SOLAR_2A: .EQU $53C9 ; SOLARTRON-A counter results ISB
SOLAR_3A: .EQU $53CA ; SOLARTRON-A counter results LSB
SOLAR_1B: .EQU $53CB ; SOLARTRON-B counter results MSB
SOLAR_2B: .EQU $53CC ; SOLARTRON-B counter results ISB
SOLAR_3B: .EQU $53CD ; SOLARTRON-B counter results LSB

SOLAR_AD: .EQU $53CE ; A/D converter for solartron/digiquartz

PCU_TYPE: .EQU $53F0 ; Address of PCU type found on card.
PCU_VALUE: .EQU $53AC ; PCU Card valves address.
PCU_DA_2: .EQU $53AD ; PCU Card D/A LSB address.
PCU_DA_1: .EQU $53AE ; PCU Card D/A MSB address and latch.

SOLARA_ADMUX: .EQU $08 ; Select for channel A
SOLARB_ADMUX: .EQU $48 ; Select for channel B

Software 129
Pressure Controller driver commands

* GLOBAL/EXTERNAL SYMBOL DEFINITIONS *

```
.GLOBAL  CONTROLLER, VENT_OFF, DEFN_RANGE, CTRL1
.GLOBAL  INPUT_ON, CHECK_P, REZERO, CHECK_PCU
.GLOBAL  C7a, PCU_RULES, CLAMP_OFF, BUILD_TABLE
.GLOBAL  PUT_PCU, CHECK_PCU_D, CHECK_TABLE

.EXTERNAL PRINT_LB, PRINT_L1, PRINT_L2, GET_ITEM, PUT_CHECK
.EXTERNAL TIME_DELAY, BLINKKEY, GET_FP, P_WAIT, FFLFP, FFXFP
.EXTERNAL GET_XDCR, ADDQUEUE, CNTL_LABELS, CNTL_DISP
.EXTERNAL PCU_LABELS, UNIT_LABELS, MODE_LABELS, MODE_DISP
.EXTERNAL RNGE_LIST, RESET, UNIT_EQ, TRIGGER_DISP, EXEC_TB
.EXTERNAL XDCR_ERR, CLEAR_TB, PCU_D_BAD, REMQUEUE, PCU_T_BAD
```

Software 130
This program assumes that the user will have the feedback transducer connected to XDCR 1 and displayed on the top display. The units of XDCR 1 are the units that the user defined. The tables will be built in PSI.

The user should follow this order: First set up XDCR 1; then build tables; set user defined min and max inputs allowed; input a value to be sent out.

CONTROLLER: PSHS X,Y,U,D ; Save registers used LDA PCU_RANGE ; Range of PCU Card CMPA #$8F ; Is the PCU installed LBEQ NOTINST ; Branch if not LDX #CNTL_LABELS ; Point to list of choices LDY #CNTL_DISP ; Point to menu title LDA #CNTL_OPTS ; # of items in the menu CLRBL LBSR GET_ITEM ; Get new setting TSTA ; Abort?

LBNE DONE ; Go exit if so CMPB #IN_ON ; Chose to change input value BNE CNTL1

LBSR INPUT_ON ; Yes, then jump BRA DONE

CNTL1: CMPB #V_OFF ; Is the chose to turn off pressure

BNE CNTL2

LBSR VENT_OFF ; Yes, then jump BRA DONE

CNTL2: CMPB #C_OFF ; Absolute or differential

Software 131
BNE CNTL3
LBSR CLAMP_OFF
BRA DONE ; Yes, then jump
CNTL3:
CMPB #IN_RANGE
BNE CNTL4
LBSR DEFN_RANGE
BRA DONE ; Yes, then jump
CNTL4:
CMPB #TOLERANCE
BNE CNTL5
LBSR PRES_TLRNCE
BRA DONE ; Yes, then jump
CNTL5:
CMPB #BUILD
BNE CNTL6
LBSR BUILD_TABLE
BRA DONE ; Yes, then jump
CNTL6:
LBSR PCU_RULES
BRA DONE ; Yes, then jump
DONE:
PULS X,Y,U,D ; Restore registers
RTS ; Return
;******************************************************************************
;* Procedure:  DISPLAY THE RULES OF PCU DISPLAY: PCU_RULES                *
;* Revision:   0.0 07/15/88                                                  *
;* Written by: Greg Hulan                                                   *
;* Purpose:    Provides required and given information regarding          *
;*             the pressure controller.                                    *
;* Inputs:     none                                                        *
;* Outputs:    none                                                        *
;* Uses:       none                                                        *
;* Alters:     none                                                        *
;* Registers:  none                                                        *
;* System Stack: 0 byte register storage                                   *
;* Macros:     none                                                        *
;* Calls:      PRINT_L1: Displays msg                                       *
;*              PRINT_L2: Displays msg                                       *
;*              TIME_DELAY: Delay that allows the message to stay on.      *
;* Is called by: CONTROLLER                                                *
;* Revisions:  07/15/88 Greg Hulan: Code written                          *
;******************************************************************************

PCU_RULES:  PSHS X,Y,U,D
            LDX   #CTRL1           ; Point to msg
            LDA   #09            ; Counter used to output ll

lines
Cl:
    LBSR    PRINT_LB         ; Display it
    LBSR    TIME_DELAY      ; Delay between displayed lines
    LBSR    TIME_DELAY      ; Delay between displayed lines
    LBSR    TIME_DELAY
    LEAX    08,X            ; Go to next lines to display
    DECA
    BNE    Cl              ; Counter not zero then update

display
    LBSR    PRINT_L1        ; Display top line
    LDX    #PCU_LABELS      ; Addr of PCU range labels
    LDA    PCU_RANGE        ; Addr of the range of the PCU
    ASL3    A               ; Mult by 8
    LEAX    A,X             
    LBSR    PRINT_L2        ; Display line 2.
    LBSR    TIME_DELAY      ; Delay
    LBSR    TIME_DELAY
    LBSR    TIME_DELAY
    PULS    X,Y,U,D         
RTS
.PAGE
;***********************************************************************
;** Procedure: EDIT PRESSURE TOLERANCE: PRES_TLRNCE
;** Revision: 0.0 07/15/88
;** Written by: Greg Hulan
;** Purpose: Provides SONIX DISPLAY interface to allow user to
;** set control pressure tolerance.
;** Inputs: none
;** Outputs: none
;** Uses: none
;** Alters: none
;** System Stack: 54 byte register storage
;** Macros: none
;** Calls: PRINT_L1: Displays msg
;** PRINT_L2: Displays msg
;** P_WAIT: Pauses for input
;** BLINKKEY: Gets input
;** FFLP: Puts BCD number in ASCII format
;** GET_FP: Gets floating point 8 byte number
;** TIME_DELAY: A delay routine
;** PUT_PCU: Puts a byte into storage and alters
;** the PCU area checksum
;** Is called by: CONTROLLER
;** Revisions: 07/15/88 Greg Hulan: Code written
;***********************************************************************

PRES_TLRNCE: PSHS X,Y,U,D
           LEAS -54,S ; Create local storage
           LDX #TOLERANCE ; User defined tolerance
           LD16 {0,S},{0,X}
           LD16 {2,S},{2,X}

PT2: LDX #PP_PRCNT_DIF ; Minimum tolerance allowed
      LD16 {42,S},{0,X}
      LD16 {44,S},{2,X}
      LDA XDCR_CONFIG+XC_UNITS ; If PSI don't multiply by slope
      CMPA #PSI
      BEQ PT3
      LDX #XDCR_CONFIG+XC_MTERM ; Get XDCR1 slope and offset
      LD16 {46,S},{0,X}
      LD16 {48,S},{2,X}
      LD16 {50,S},{4,X}
      LD16 {52,S},{6,X}
      MAKEFPBCB
      FP_MULTIPLY (46+8,S),(42+8,S),(42+8,S) ; Multiply minimum tolerance allowed
      FP_ADD {42+8,S},{50+8,S},{42+8,S} ; by XDCR1 slope to put it into XDCR1
      KILLFPBCB ; units.
      BRA PT3
      PT2a: LD16 {0,S},{42,S} ; Transfer min tol to display
            LD16 {2,S},{44,S} ; Units are PSI

Software 134
PT3: MAKEFPCB
FP_TO_BCD (0+8,S),(4+8,S),#$0008 ; Find the BCD equivalent
KILLFPCB
LEAY 31,S
LEAX 4,S ; Pnt to 10 byte storage area
LDA #$08 ; Pnt to BCD addr
LBSR FP_TO_BCD ; 8 byte fmt wanted
LDX #TOLERANCE ; Do it
LBSR PRINT_L1 ; Point to "TOLERANCE"
LEAX 31,S ; Prints to top display
LDB #DISP_LINE2 ; Address of # in ascii
LBSR BLINKKEY ; Get input and print to bottom
display
CMPA #$40 ; Menu?
LBGT T11
BRANCHIF A,#ETX,PT10 ; Clear?
BRANCHIF A,#CR,PT4 ; Enter?
BRANCHIF A,'#<',PT2 ; Increment?
BRANCHIF A,'#>',PT2 ; Decrement?
PUSHQUEUE FPQ,A ; Otherwise put back on stack
/PAGE ;****************************
;
;***** Pressure Tolerance (cont) *****
;*******************************
PT4: LEAX 4,S ; Point to number storage
LBSR GET_FP ; Get numeric input
CMPA #ETX ; Clear abort?
LBEQ PT2 ; Back up if so
MAKEFPCB
FP_COMPARE (4+8,S),(42+8,S) ; Compare FP # with min allowed
KILLFPCB
LBLT PT5 ; Branch if too small
LDU #TOLERANCE ; Point to destination
LEAX 4,S ; Point to number
LDB #$04 ; 4 bytes to store
PT4a:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,PT4a ; Decrement and jump if not zero
LDA #$FF ; Tolerance has been set
LDU #$TOL_SET ; Store byte and update
LBSR PUT_PCU

checksum
BRA T10 ; Done

PT5: LDX #TO_SMALL ; Display "TO SMALL"
LBSR PRINT_L2 ; Print to bottom display
LBSR TIME_DELAY ; Displays error for short time
LBSR TIME_DELAY ; Displays error for short time
LBSR TIME_DELAY ; Displays error for short time
LBRA PT2a ; Start again

Software 135
PT10: LEAS 54,S ; Release local storage
    PULS X,Y,U,D
    RTS

PT11: PUSHQUEUE FPQ,A ; Put menu key back on stack
    BRA PT10

TLNCE: .BYTE 'TOLERANCE'

;**************************************************************************
; Procedure: BUILD PCU LOOK-UP TABLES: BUILD_TABLE
; Revision: 0.0 07/15/88
; Written by: Greg Hulan
; Purpose: Builds look-up tables that the PCU uses to calculate
;          a DAC value to send to the PCU.
; * Inputs: none
; * Outputs: none
; * Uses: none
; * Alters: none
; * Registers:
; * System Stack: 17 byte register storage
; * Macros: none
; * Calls: PRINT_L1: Displays msg
;          PRINT_L2: Displays msg
;          P_WAIT: Pauses for input
;          BLINKKEY: Gets input
;          FFLP: Puts BCD number in ASCII format
;          GET_FP: Gets floating point 8 byte number
;          TIME_DELAY: A delay routine
;          PUT_PCU: Puts a byte into storage and alters
;          the PCU area checksum
; * Is called by: CONTROLLER
; * Revisions: 08/01/88 Greg Hulan: Code written
;**************************************************************************

; NOTE: If you abort this routine, your configuration and
tolerance values may be changed resulting in checksum errors.

BUILD_TABLE: PSHS X,Y,U,D
    LEAS -17,S ; Local storage
    LDX #PCU_TABLE ; Addr of table for hardware

max in FP PSI
    LDR PCU_RANGE ; Range of XDCR 1
    ASLB ; Malt by 4
    ASLB
    LEAX B,X ; Addr of hardware max in PSI
    LD16 {9,S},{0,X} ; Put in local storage
    LD16 {11,S},{2,X}
    LDU #PCU_FP_MAXIN ; Point to destination
    LEAX 9,S
    LDB #$04 ; 4 bytes to store

BT1: LDA ,X+ ; Get byte
    LBSR PUT_PCU ; Store byte
    LEAU 1,U ; Advance to next store
    DJNZ B,BT1

Software 136
LBSR PCU_PRCENT
LD16 {13,S}, TOLERANCE ; Save user defined tolerance
LD16 {15,S}, TOLERANCE+2
LD16 TOLERANCE, FP_PRCNT_DIF ; Temp make the minimum tolerance

LD16 TOLERANCE+2, FP_PRCNT_DIF+2 ; the present tolerance
LDA XDCR_CONFIG+XC_UNITS ; Units being display by XDCR1
STA 0, S ; Save the units of XDCR 1
LDA #PSI ; Change XDCR1 units to PSI
STA XDCR_CONFIG+XC_UNITS
LDA XDCR_CONFIG+XC_UNITS
LDX #XDCR_CONFIG+XC_MTERM ; Addr of the slope and offset
LD16 {1,S}, {0,X}
LD16 {3,S}, {2,X}
LD16 {5,S}, {4,X}
LD16 {7,S}, {6,X}
LDA XDCR_CONFIG+XC_MODE ; Temp store of mode
STA 12, S
CLRA
STA XDCR_CONFIG+XC_MODE ; Temp change XDCR1 mode to absolute

LDA #PSI ; Change XDCR 1 slope and offset to PSI
ASL3 A
LDY #UNIT_EQ
LEAY A, Y
LD16 {0,X}, {0,Y} ; Contains the slope and offset
LD16 {2,X}, {2,Y} ; for the units conversion to PSI

LD16 {4,X}, {4,Y}
LD16 {6,X}, {6,Y}
LBSR READ_ATM ; Read atmospheric
LBSR CALC
LDA 0, S
STA XDCR_CONFIG+XC_UNITS ; Restore units
LDX #XDCR_CONFIG+XC_MTERM ; Restore slope and offset
LD16 {0,X}, {1,S} ; Contains the slope and offset
LD16 {2,X}, {3,S} ; for the units conversion.
LD16 {4,X}, {5,S}
LD16 {6,X}, {7,S}
LDA 12, S
STA XDCR_CONFIG+XC_MODE
LD16 TOLERANCE, {13,S} ; Restore tolerance
LD16 TOLERANCE+2, {15,S}
LEAS 17, S ; Removes local storage
PULS X, Y, U, D
RTS

; Table of floating point numbers in PSI that represents the max allowed
; output for all PCUs with that range
;
PCU_TABLE: .BYTE $40, $C0, $00, $00 ; DIG 6 PSI
; This is a table of values is used to compute the number of digits after the DP
; the user is allowed to input.

NUM_DP:

; BYTE  $3D,$CC,$CC,$CC  ; 0.1
; BYTE  $3C,$23,$D7,$0A  ; 0.01
; BYTE  $3A,$83,$12,$6E  ; 0.001
; BYTE  $38,$D1,$B7,$17  ; 0.0001
; BYTE  $37,$27,$C5,$AC  ; 0.00001
; BYTE  $35,$86,$37,$BD  ; 0.000001
; BYTE  $33,$D6,$BF,$94  ; 0.0000001

;
**Procedure:** READ ATMOSPHERIC PRESSURE: READ_ATM

**Revision:** 0.0 08/01/88

**Written by:** Greg Hulan

**Purpose:** Read atmospheric pressure in PSI and saves it with the table. Only the BUILD TABLE routine should use this. Atm value is in PSI ABS.

**Inputs:** none

**Outputs:** none

**Uses:** none

**Alters:** X register

**System Stack:** 4 byte register storage

**Macros:** none

**Calls:**
- CALC2: Gets the reading from XDCRL
- PUT_PCU: Puts a byte into storage and alters the PCU area checksum

**Is called by:** CONTROLLER

**Revisions:** 08/01/88 Greg Hulan: Code written

---

READ_ATM:

```assembly
PSHS Y,U,D
LEAS -4,S
LDA #$0C ; Close valve 2,3 and open the others

STA PCU_VALVE
LDU #$LAST_VALVE ; Store last valve setting
LBSR PUT_PCU
LEAY 0,S
LDD #$8000 ; Write zero to D/A
LBSR CALC2
LDU #$ATM_TBL ; Point to destination
LEAX 0,S ; Point to number
LDR #$04 ; 4 bytes to store

RA1:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,RA1
LEAS 4,S
PULS Y,U,D
RTS
```

---

Software 139
REZERO: PSHS X,Y,U,D
LEAS -5,S
LDA XDCR_CONFIG+XC_MODE
STA 0,S ; Create local storage

; Save XDCR1 mode

LDA #$00
; Temporarily make XDCR1 have ABS mode

LDA #$0C
; Close valve 2 and open the others

STA PCU_VALVE
LDU #LAST_VALVE ; Save valve setting
LBSR PUT_PCU ; Update checksum
LEAY 1,S
LDD #$8000 ; Write zero to D/A
LBSR CALC2 ;
LDA 0,S
; STA XDCR_CONFIG+XC_MODE ; Restore XDCR1 mode

;******************** -GTH 2-21-89
LDA 0,S ; Get mode
TSTA ; Is XDCR1 in ABS mode?
BEQ R20 ; Jump if so
MAKEFFPCB
FP_ADD (1+8,S),(LAST_ATM),(1+8,S) ; Add last ATM reading to it
KILLFFPCB

R20:
CLRA
STA PCU_VALVE ; Save valve setting
LDU #LAST_VALVE
LBSR PUT_PCU
LDU #LAST_ATM ; Point to destination
LEAX 1,S ; Point to number
LDB #$04 ; 4 bytes to store
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,RZ1 ; Release local storage
LEAS 5,S
PULS X,Y,U,D
RTS
.PAGE
Procedure: CALCULATE LOOKUP TABLE VALUES: CALC

Revision: 0.0 09/01/88

Written by: Greg Hulan

Purpose: Calculate the lookup table values for the PCU. These values are used to calculate the DAC value to send out.

Inputs: none

Outputs: none

Uses: None

Alters: During this routine, the mode is changed to ABS mode and PSI. If RESET is pressed during it, "config will result.

System Stack: 8 byte register storage

Macros: none

Calls: CALC2: Reads atmospheric

PUT_PCU: Puts a byte into storage and alters the PCU area checksum

PUT_TABLE: Puts a byte into storage and alters the PCU TABLE area checksum

PRINT_LB: Prints to top and bottom display

CALC2: Write to DAC and get FP value from XDCR1

Is called by: BUILD_TABLE

Revisions: 09/01/88 Greg Hulan: Code written

*******************************************************************************

CALS:

PSHS X,Y,U,D
LEAS -8,S
LDX #$ZERODONE ; Prints that 0/3 of table is done
LBSR PRINT_LB
LDA #$06 ; Turn on valve 1 and 2
STA PCU_VALVE
LDU #$LAST_VALUE ; Save valve setting
LBSR PUT_PCU
LDY #$FIS_TABLE ;
LEAY -4,Y ; Last addr in the low table
STY 0,S
LDD #$8000 ; First word to D/A
STD 2,S ; Save the counter
LT:
LEAY 4,S
LBSR CALC2 ; Write to DAC and get the table value
LDU 0,S ; Point to destination
LEAX 4,S ; Point to number
LDB #$04 ; 4 bytes to store
LT1:
LDA .X+ ; Get byte
LBSR PUT_TABLE ; Store table FP byte
LEAU 1,U ; Advance to next store
DJNZ B,LT1
LDY 0,S
storage
LEAY -4,Y ; Decrease addr to next table
STY 0,S
CMPY #FPIS_TABLE ; First location in low table
BLT MID_CALC ; Jump if done with low table
LDD 2,S
SUBD #1000 ; Decrement to next D/A value
STD 2,S
BRA LT ; Continue

;
; *************************************
; *** Calculating the middle table ***
; *************************************

; MID_CALC:
LBSR REZERO ; Rezero output to ATM
LDX #ONEDONE ; Print that 1/3 of table is
LBSR PRINT_L2 ; done

LDA #$07 ; Turn on valve 0, 1 and 2
STA PCU VALVE ; Write to valves
LDU #LAST_VALUE ; Save valve setting
LBSR PUT_PCU ;
LDY #FPIS_TABLE ;
LEAY 60,Y ; Middle of the middle table
STY 0,S
LDD #$8000 ; First word to D/A
STD 2,S ; Save the counter
MC: LEAY 4,S ; Store FP returned value here
LBSR CALC2
LDU 0,S ; Point to destination
LEAX 4,S ; Point to number
LDB #$04 ; 4 bytes to store
MC1: LDA ,X+ ; Get byte
LBSR PUT_TABLE ; Store FP table byte and

LEAU 1,U ; Advance to next store
DJNZ B,MC1
LDY 0,S
LEAY -4,Y ; Decrease addr to next table

storage
CMPY #FPIS_TABLE ; Lowest middle table entree
BLT OTHERHALF ; Calculate the other half of

table
STY 0,S
LDD 2,S
SUBD #200 ; Decrement to next D/A value
STD 2,S
BRA MC ; Continue

; OTHERHALF:
LBSR REZERO ; Rezero output to ATM
LDA #$07 ; Turn on valve 0, 1 and 2

Software 143
STA PCU_VALUE
LDU #LAST_VALUE
LBSR PUT_PCU
LDY #FPIS_TABLE
LEAY 64,Y
STY 0,S
LDD #$80C8
STD 2,S
OH:
LEAY 4,Y
LBSR CALC2
LDU 0,S
LEAX 4,S
LDB #$04
OH1:
LDA ,X+
LBSR PUT_TABLE
update checksum
LEAU 1,U
DJNZ B,OH1
LDY 0,S
LEAY 4,Y
storage
CMPY #FPMS_TABLE-4
BGT HIGH_CALC
STY 0,S
LDD 2,S
ADDX #200
STD 2,S
BRA OH
HIGH_CALC:
LBSR REZERO
LDX #TWODONE
LBSR PRINT_L2
LDA #$05
STA PCU_VALUE
LDU #LAST_VALUE
LBSR PUT_PCU
LDY #FPMS_TABLE
table
STY 0,S
LDD #$83E8
STD 2,S
HC:
LEAY 4,Y
LBSR CALC2
LDU 0,S
LEAX 4,S
LDB #$04
HC1:
LDA ,X+
LBSR PUT_TABLE
update checksum
LEAU 1,U
Software

DJNZ B, HC1 ; Branch until all stored
LDY 0, S
LEAY 4, Y ; Increase addr to next table
storage
CMPY #TEMP_VAR-4 ; Last high-table location
BGT OUT ; If GT, then done calculating
tables
STY 0, S
LDD 2, S
ADD #1000
STD 2, S ; Increase D/A to next value
BRA HC

; OUT:
LBSR REZERO ; Rezero output to ATM
LEAS 8, S ; Release local storage
PULS X, Y, U, D
RTS
ZERO_DONE: .BYTE 'TABLE IS', 'O/3 DONE'
ONE_DONE: .BYTE '1/3 DONE'
TWODONE: .BYTE '2/3 DONE'
; .PAGE
PROCEDURE  WRITE TO DAC AND GET XDCR1 FP PRESSURE: CALC2

* Revision: 0.0 10/01/88
* Written by: Greg Hulan

* Purpose: Writes to the PCU DAC and reads the transducer of XDCR1
* Inputs: D  Contains the HEX value to be written to the DAC
         Y  Contains the address of the returning FP #
* Outputs: none
* Uses: none
* Alters: none
* Registers:
* System Stack: 34 byte register storage
* Macros: none
* Calls:  CALC2: Reads atmospheric
         PUT_PCU: Puts a byte into storage and alters the PCU area checksum
* Is called by: MANY ROUTINES
* Revisions: 10/01/88 Greg Hulan: Code written

CALC2:

PSHS  X,Y,U,D
LEAS  -34,S           ; Create local storage
STY  0,S
STB  PCU_DA_2         ; Send low byte to D/A
STA  PCU_DA_1         ; Send high byte to D/A and latch it
LDD  #$00
STD  8,S              ; Second to last XDCR1 value
STD  10,S             ; Last XDCR1 value
STD  12,S
STD  14,S
LDX  #FP_PRNT_DIF     ; Min. tolerance allowed.
LD16  {16,S},{0,X}
LD16  {18,S},{2,X}

CALC3:

CLRA
CLRB
STD  26,S           ; Initialize timeout counts
STD  10,S
LDX  #TRIG_BLOCK    ; Address of 1st trigger block
CLRB

CC1:
LDX  #TRIG_BLOCK    ; Address of first trigger block (256 bytes)
CLR .X+             ; Clear trigger block (256 bytes)
DJNZ  B,CC2

CC2:
LDX  #TRIG_BLOCK
STX  CURRENT_TB     ; Address of current trigger block (0)
CLR  CURRENT_TBN    ; Number of current trigger block (0)
CLR SOLARA_ATOD ; Clear out A/D
CLR SOLARA_ATOD+1 ; Clear out A/D

CLR XDCR_RDY ; Xducers NOT ready (bit mapped)

; Start the displays by setting up
will

; set up trigger blocks for any transducer input they need.

CC3:
LDA #$03 ; A register set to 3
LBSR TRIGGER_DISP ; Trigger internal display (3)

LDD #$8000 ; Set timeout values
STD DISP_FLAGS+DF_COUNT+3*DF_SIZE ; Timeout counter for internal display

CC3A: LBSR EXEC_TB ; Execute triggers. Use of trigger block located in CURRENT_TB;

CC4: ENABLE IRQ,FIRQ ; TRIGGER DEVICES REQUESTED
LD8 MAIN_PASS,XDCR_RDY ; Copy Xdcr_Rdy to Main_PASS
DISABLE IRQ,FIRQ

TST DISP_FLAGS+DF_DONE+3*DF_SIZE ; Internal display finished
LBNE CC9 ; Skip if so
LDA XDCR_RDY ; Get completion status
ANDA DISP_FLAGS+DF_NEEDS+3*DF_SIZE ; Mask out needed flags
CMFA DISP_FLAGS+DF_NEEDS+3*DF_SIZE ; Display 3 ready?
BNE CC8 ; Skip if not

LDY #$XDCR_CONFIG+XC_MTERM ; Addr. of slope
LEAU 4,S ; Pointer for FP of XDCR1
LDX #$XDCR_FLAGS+XF_DATA ; Data storage for XDCR1
LBSR GET_XDCR
LDA XDCR_CONFIG+XC_MODE ; Mode of XDCR1
TSTA BEQ CC5
MAKEFPCB
FP_SUBTRACT {LAST_ATM},{4+8,S},{4+8,S} ; Subtract ATM if in DIFF mode
KILLFPCB

CC5: LDD #$8000 ; Reset timeout
STD DISP_FLAGS+DF_COUNT+3*DF_SIZE ; Reset timeout

Software 147
DEC DISP_FLAGS+DF_DONE+3*DF_SIZE ; Flag display complete
BRA CC10

; CC7:
LBSR PRINT_L2 ; Put it on front panel
BRA CC9

; CC8:
LDD DISP_FLAGS+DF_COUNT+3*DF_SIZE ; Bump timeout for bottom
ADDD #$0001
STD
BNE DISP_FLAGS+DF_COUNT+3*DF_SIZE ; Timeout?

STD 29,S ; Saves A
LDD #$7FF0 ; BOTTOM DISPLAY COUNT TIMEOUT
STD 31,S
LDA 26,S ; COUNTER VALUE
INCA
CMPA #$02 ; IS TIMEOUT TWO TIMES IN A ROW.
BNE TO2 ; TIMEOUT TWICE?
LDA #$01 ; COUNTER EQUALS 1
STA 26,S ; SAVE IT.
LDD 29,S ; GETS A.
LDX #XDCR_ERR ; IF TIMEOUT TWICE THEN DISPLAY XDCR_ERR.
BRA CC7 ; XDCR_ERR.

TO2:
STA 26,S ; SAVE TIMEOUT COUNT
LDD 29,S
LBRA CC1 ; NO THEN RESET A/D AND FREQ

COUNT CC9:
STD 30,S ; NO timeout.
LDD 32,S
ADDD #$0001 ; INCREMENT BOTTOM DIS. COUNT
STD 32,S
BNE TO4 ; DID COUNT REACH 0?
CLR A ; YES, THEN DIDN’T GET TWO
STA 27,S ; TIMEOUTS IN A ROW.
; RESET TIMEOUT COUNTER

TO4:
LDX 30,S ; GETS A
LDA CURRENT_TB ; Get current trigger block
ANDA TB_DONE,X ; Get transducers ready
ANDA TB_DONE,X ; Mask transducers which complete block
CMPA TB_DONE,X ; Is block complete?
LBNE CC4 ; Go continue loop if not
LDA MAIN_PASS ; Check to see if XDCR_RDY has changed
CMPA XDCR_RDY ; If so repeat display checks!
LBNE CC4 ; CLEAR TB
LBSR CLEAR_TB ; Clear out current TB

Software 148
TST    DISP_FLAGS+DF_DONE+3*DF_SIZE ; Is internal display done
LBEQ   CC3A                        ; Loop if not
LBRA   CC3

CC10:  MAKEFFCB
FP_SUBTRACT(12+8,S),(4+8,S),(20+8,S) ; Subtract last reading from present reading
LDA    28,S
ANDA   #$7F
STA    28,S
FP_COMPARE(16+8,S),(20+8,S) ; Is the difference too big
KILLFPCB
BLT    NOTSTABLE                  ; Jump if so
LDY    0,S
LD16   {0,Y},{4,S}
LD16   {2,Y},{6,S}
LEAS   34,S
PULS   X,Y,U,D
RTS

: NOTSTABLE: LD16 (12,S),(4,S) ; Copy present read to last read
LD16   {14,S},{6,S}
LBRA   CALC3 .PAGE

Software 149
;******************************************************************************************
; Procedure: READ AND DISPLAY XDCR1 ON TOP DISPLAY: READ_DISP1
; Revision: 0.0 11/01/88
; Written by: Greg Hulan
; Purpose: Read and display XDCR1's pressure value on the top display.
; Inputs: D Contains the HEX DAC value
; Y Contains the address of the returning FP pressure
; X Equals zero if don't want pressure displayed, just saved
; Outputs: none
; Uses: none
; Alters: none
; Registers:
; System Stack: 34 byte register storage
; Macros: none
; Calls: CALC2: Reads atmospheric
; PUT_PCU: Puts a byte into storage and alters the PCU area checksum
; Is called by: MANY ROUTINES
; Revisions: 11/01/88 Greg Hulan: Code written
;******************************************************************************************

READ_DISP1: PSHS X, Y, U, D
LEAS -34, S
STX 32, S
STY 0, S

pressure
LEAY 2, S
LBSR CALC2
MAKFPCB
FP_TO_BCD {2+8, S}, {6+8, S}, #0008
KILLFPCB
LEAX 6, S
LDY #DISP_FLAGS+DF_DATA
LDU #DISP_CONFIG+DC_CNT
TST DC_FMT, U
BNE RD24
BNE RD26
LDD #X_SIZE
LDU #XDCR_CONFIG+XC_FMT
MUL
LDD D, U
BRA RD25

RD24: LDA DC_FMT, U
DECA
BNE RD26
LDD DC_DGT, U
RD25: LBSR FFXFP
BRA RD27

Software 150
RD26: LDD DC_DGT, U ; Floating?
      LBSR FFLFP ; Go format output
RD27: LDX #DISP_FLAGS+DF_DATA ; Location of data to display
      TST 33, S ; Zero if don’t want it
displayed
      BEQ RD28 ; Print it to top display
      LBSR PRINT_LL
RD28: LDY 0, S ; Save FP value
      LD16 (0,Y), (2,S)
      LD16 (2,Y), (4,S)
      LEAS 34, S ; Release local storage
      PULS X,Y,U,D
      RTS
      .PAGE
 ; Procedure: DISPLAY NUMBER USING XDCR1 FORMAT: DISP_X1_FMT
 ; Revision: 0.0 02/01/89
 ; Written by: Greg Hulan
 ; Purpose: This routine prints a number to the top or bottom display in XDCR1 format. This routine uses the format of the top display.
 ; Inputs: X contains address of BCD to display
 ; Outputs: none
 ; Uses: none
 ; Alters: none
 ; Note: XDCR1 must be displayed in the top display. The top display format is used.
 ; System Stack: 5 byte register storage
 ; Macros: none
 ; Calls: PRINT_L1: Print to the top display
 ; PRINT_L2: Print to the bottom display
 ; FFLFP: Format number for display
 ; FFXFP: Format number for display
 ; Is called by: Many routines
 ; Revisions: 02/01/89 Greg Hulan: Code written

 DISP_X1_FMT: PSHS X,Y,U,D ;
 LEAS -5,S ; Create local storage
 STX 0,S ; BCD addr.
 STB 2,S ; Display line # to print
 TSTB
 BNE DX0 ; Branch if bottom display
 LDY #DISP_FLAGS+DF_DATA ; Save ascii data here for top display

 display
 STY 3,S ; Temp store
 BRA DX0a

 DX0: LDY #DISP_FLAGS+DF_DATA+DF_SIZE; Save ascii for bottom display
 STY 3,S ; Temp store

 DX0a: LDU #DISP_CONFIG ; Points to top display configuration

 TST DC_FMT,U ; Check for default output format
 BNE DX1 ; Skip if not
 LDU #$XDCR_CONFIG+XC_FMT ; Point to format field of XDCR1
 LDD ,U ; Get proper format
 BRA DX2 ; & do it

 ;
 DX1: LDA DC_FMT,U ; Fixed?
 DECA
 BNE DX3 ; Skip if not
 LDD DC_DGT,U

 DX2: LBSR FFXFP ; Go format output
 BRA DX4

 ;
 DX3: LDD DC_DGT,U ; Floating?
DX4:
LBSR  FFLFP
LDX   3, S
LDB   2, S
CMFB  #$01
BEQ   DX5
LBSR  PRINT_L1
BRA   DX6

DX5:
LBSR  PRINT_L2

DX6:
LEAS  5, S
PULS  X, Y, U, D
RTS

; Go format output
; Location of data to display
; Bottom display?
; Jump if so
; Otherwise print line 1
; Release local storage
;****************************************************************************
;/* Procedure:  STARTS PRESSURE OUTPUT ON: INPUT_ON  *
;/* Revision:  0.0 12/01/88  *
;/* Written by:  Greg Hulan  *
;/* Purpose:  This routine calculates the value to send to DAC and  *
;/* stays here until desired output pressure is reached  *
;/* Inputs:  none  *
;/* Outputs:  none  *
;/* Uses:  none  *
;/* Alters:  none  *
;/* System Stack:  -196 byte register storage  *
;/* Macros:  none  *
;/* Calls:  CALC2:  Reads atmospheric  *
;/* PUT_PCU:  Puts a byte into storage and alters  *
;/* the PCU area checksum  *
;/* Is called by:  Many routines  *
;/* Revisions:  02/01/89 Greg Hulan:  Code written  *
;****************************************************************************

INPUT_ON:

PSHS  X,Y,U,D
LEAS -196,S
LDX  #PCU_FP_MAXIN

PSI

STX  44,S
LDX  #USER_PCU_MAX
STX  46,S

STX  #USER_PCU_MIN
STX  48,S

C2:

LDY  #PCU_IN

XDCR1 units and mode

STY  1,S

C3:

MAKEPFCB

FP_TO_BCD  {{1+8,S}},{3+8,S},#$0008
KILLPFCB

LDB  #$01
LEAX  3,S
LBSR  DISP_X1_FMT
LDX  #CTRL4
LBSR  PRINT_L1
LDX  #$DISP_FLAGS+DF_DATA+DF_SIZE
LDB  #$DISP_LINE2
LBSR  BLINKKEY

display

CMPA  #$40
LBGT  C41
BRANCHIF  A,#RTX,C42
BRANCHIF  A,#CR,C04
BRANCHIF  A,#<',C2

Software  154
BRANCHIF A,'>','C2 ; Decrement?
PUSHQUEUE FPQ,A ; Otherwise put back on stack C04: LDD #NUM_DP ; Table of FP numbers to compare tolerance to STD 194,S LDD #$01 ; Initial count STD 192,S C04alpha: MAKEFPCB FP_COMPARE {TOLERANCE},{[194+8,S]} ; Is tolerance larger than table value KILLFPCB BGE C04beta ; Branch if so INC 193,S ; Increment count LDD 194,S ; Point to next table value ADDD #$04 STD 194,S BRA C04alpha C04beta: LDY 192,S ; Number of digits after DP. LEAX 3,S ; Point to number storage LBSR GET_FP ; Get numeric input CMPA #ETX ; Clear abort? LBEQ C2 ; Back up if so TSTA ; Successful input? LBNE C42 ; Abort edit if not ; LDX #XDCR_CONFIG+XC_MTERM ; Store value of slope and offset LD16 {11,S},{0,X} ; in units of XDCR 1 LD16 {13,S},{2,X} LD16 {15,S},{4,X} LD16 {17,S},{6,X} MAKEFPCB FP_COMPARE {3+8,S},{[48+8,S]} ; Compare FP # with min allowed KILLFPCB LBET C5 LDA XDCR_CONFIG+XC_UNITS ; Units of XDCR1 CMPA #PSI ; Is it in PSI? BEQ C04a ; Branch if so MAKEFPCB FP_MULTIPLY {[44+8,S]},{11+8,S},{180+8,S} ; Multiply hardware max by slope FP_ADD {180+8,S},{15+8,S},{180+8,S} ; and add offset of XDCR 1 units KILLFPCB BRA C04b ; See if input is differential. If so, subtract LAST_ATM C04a: LDX 44,S LD16 {180,S},{0,X} ; Transfer PCU_MAXIN to 180,S LD16 {182,S},{2,X} C04b: LDA XDCR_CONFIG+XC_MODE ; XDCR1 in ABS mode?
TSTA
BEQ C04c; Jump if ABS mode
MAKEFPCB
FP_SUBTRACT (LAST_ATM),(180+8,S),(180+8,S); Add ATM reading to value
KILLFPCB
C04c:
MAKEFPCB
FP_COMPARE (3+8,S),(180+8,S) ; Compare two FP #'s
BGT C4; Out-of-range? jump
FP_COMPARE (3+8,S),[[46+8,S]] ; Compare to software max

pressure limit
BLE C7; If in range jump
C4: KILLFPCB
LDX #TOO_BIG ; Kill FP stack
BRA C6
C5: LDX #TO_SMALL ; Display "TOO SMALL"
C6: LBSR PRINT_L2
LBSR TIME_DELAY ; Displays error for short time
LBSR TIME_DELAY ; Displays error for short time
LBSR TIME_DELAY ; Displays error for short time
LBRA C2 ; Start again
C7: KILLFPCB
C7a: LDU #PCU_IN ; Point to destination
LEAX 3,S ; Point to number
LDB #04 ; 4 bytes to store
C8: LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,C8
;
MAKEFPCB
FP_TO_BCD (3+8,S),(144+8,S),#0008 ; Convert the desired output to BCD
KILLFPCB
LDB #01 ; Put it on bottom display
LEAX 144,S ; Point to BCD number
LBSR DISP_XL_FMT
LDA #FP_PRCNT_JMP ; Put addresses in local storage
STX 50,S
LDX #FPIS_TABLE-8
STX 52,S
LDX #FPMS_TABLE+4
STX 54,S
LDA XDCR_CONFIG+XC_MODE ; XDCR1 in ABS mode?
TSTA BEQ C8a ; Jump if so
MAKEFPCB
FP_ADD (LAST_ATM),(3+8,S),(180+8,S) ; Desired Pres in XDCR1 units
ABS mode
KILLFPCB
BRA C8b
C8a: LD16 (180,S),(3,S)

Software 156
LD16
{182,S},{5,S}

C8b:
LDA           XDRC_CONFIG+XC_UNITS
CMFA          #PSI
BEQ           C8c ; Jump if already in PSI
MAKEFPCB
FP_SUBTRACT   {15+8,S},{180+8,S},{180+8,S}
FP_DIVIDE     {180+8,S},{11+8,S},{180+8,S} ; Desired P in ABS PSI
KILLFPCB

C8c:
MAKEFPCB
FP_COMPARE    {180+8,S},[{52+8,S}] ; Is the desired value in the
KILLFPCB
LBLT          C15 ; Jump if so
MAKEFPCB
FP_COMPARE    {180+8,S},[{54+8,S}] ; Is it in the high table
KILLFPCB
LBGT          C20 ; Jump if so
LDD           52,S

; *****************************************************************
; *** It is in the middle table ***
; *****************************************************************
LDX           #FPIS_TABLE ; Address of middle table value
STX           56,S
LDD           #$7448
STD           58,S ; Hex DAC value
C11:
MAKEFPCB
FP_COMPARE    {180+8,S},[{56+8,S}] ; Compare desired out to value
in table(PSI ABS)
KILLFPCB
BLT           C12 ; Hex DAC value
LDD           58,S
ADD            #200
STD           58,S ; Increase to next table Hex value
LDD           56,S
ADD            #04
STD           56,S ; Increase to next location in table
CMPD          #FPMS_TABLE ; First location of high table
LBEQ          C50 ; If so, pressure can not be set so jump
BRA           C11
C12:
LDD           58,S ; High Hex DAC value
CMPD          #$8000
LBGT          C13 ; Branch if in high half of middle table

; *****************************************************************
; *** In low half of middle table ***
; *****************************************************************
LDX           56,S

Software

157
LD16 (60,S),(0,X) ; High side of desired FP
LD16 (62,S),(2,X)
LDD 56,S
SUBD #$04
STD 56,S
TFR D,X
LD16 (64,S),(0,X) ; Low side of desired FP
LD16 (66,S),(2,X)
LDD #$00
STD 128,S
LDD 58,S
STD 130,S
MAKEFPCB D32_TO_FP (128+8,S),(68+8,S) ; High side of desired DAC in Hex
KILLFPCB
LDD 58,S
SUBD #200
STD 130,S
MAKEFPCB D32_TO_FP (128+8,S),(72+8,S) ; Low side of desired DAC in Hex
KILLFPCB
LBRA C14
C13: LDX 56,S
LD16 (64,S),(0,X) ; High side of desired FP
LD16 (66,S),(2,X)
LDD 56,S
SUBD #$04
STD 56,S
TFR D,X
LD16 (60,S),(0,X) ; Low side of desired FP
LD16 (62,S),(2,X)
LDD #$00
STD 128,S
LDD 58,S
STD 130,S
MAKEFPCB D32_TO_FP (128+8,S),(72+8,S) ; High side of desired DAC in Hex
KILLFPCB
LDD 58,S
SUBD #200
STD 130,S
MAKEFPCB D32_TO_FP (128+8,S),(68+8,S) ; Low side of desired DAC in Hex
Software 158
KILLFPCB
LDA XDCR_CONFIG+XC_MODE
TSTA
BEQ C14
LBSR REZERO
C14:
LDA #$0F ; Close all valves
STA PCU_VALVE
LDU #$LAST_VALVE
LBSR PUT_PCU
LBRA C30
C15:
LDX #FPIS_TABLE-4 ; Address of the low tables
hiighest value
STX 56,S
LDD #$8000
STD 58,S
C16: MAKEFPCB
FP_COMPARE (180+8,S),([56+8,S]) ; Compare desired out to value
in table (PSI ABS)
KILLFPCB
BGT C17 ; Hex DAC value
SUBD #1000
STD 58,S ; Increase to next table Hex value
LDD 56,S
SUBD #04
STD 56,S ; Increase to next location in table
CMPD #FPLS_TABLE-4 ; Last low table value already compared
LBEQ C50a ; Branch since it is lower than table goes.
BRA C16
C17:
LDX 56,S
LD16 {64,S},(0,X) ; Low side of desired FP pressure
LD16 {66,S},(2,X)
LDD 56,S
ADD #04
STD 56,S
TFR D,X
LD16 {60,S},(0,X) ; High side of desired FP pressure
LD16 {62,S},(2,X)
LDD #$00
STD 128,S ; DAC low limit = Y2
LDD 58,S
STD 130,S
MAKEFPCB
D32_TO_FP (128+8,S),(72+8,S) ; Low side of desired DAC in Hex

Software 159
KILLFPCB
LDD 58,S
ADD #1000
STD 130,S
MAKEFPCB
D32_TO_FP (128+8,S),(68+8,S)

; High side of desired DAC in
Hex

KILLFPCB
LDA XDCR_CONFIG+XC_MODE
TSTA
BEQ C18
LBSR REZERO

C18:
LDA #$0E
STA PCU_VALVE
LDU #LAST_VALVE
LBSR PUT_PCIE
LBRA C30
;

;***********************
;***  High table  ***
;***********************

C20:
LDX #FPMS_TABLE
STX 56,S
LDD #$83E8
STD 58,S

C21:
MAKEFPCB
FP_COMPARE (180+8,S),([56+8,S])

; Compare desired out to value in table (PSI ABS)

KILLFPCB
BLT C22
LDD 58,S
ADD #1000
STD 58,S

value
LDD 56,S
ADD #04
STD 56,S

; Increase to next table Hex

LDD 56,S
ADD #04
STD 56,S

; Increase to next location in table

CMPD #TEMP_VAR

; Last table value has been reached
BNE C21
LDX #TEMP_VAR+4

; Last high-table value
STX 56,S
LBRA C50b

; will be printed.
BRA C21

C22:
LDX 56,S

; High side of desired FP pressure
LD16 (64,S),{0,X}
LDD 56,S
SUBD #$04
STD 56,S

Software
V TFR
D,X
LD16 (60,S),(0,X) ; Low side of desired FP pressure
LD16 (62,S),(2,X)
LDD #$00
STD 128,S ; DAC high limit = Y2
LDD 58,S
STD 130,S
MAKEFPCB
D32_TO_FP (128+8,S),(72+8,S) ; High side of desired DAC in

Hex
KILLFPCB
LDD 58,S
SUBD #$1000
STD 130,S
MAKEFPCB
D32_TO_FP (128+8,S),(68+8,S) ; Low side of desired DAC in

Hex
KILLFPCB
LDA XDCR_CONFIG+XC_MODE ; In ABS mode
TSTA BEQ C23 ; Branch if so
LBSR REZERO ; Otherwise get an atmospheric reading
C23:
LDA #$00 ; Close valves 0, 2, and 3
STA PCU_VALVE
LDU #LAST_Valve ; Save last valve setting
LBSR PUT_PCU

;******************************************************
; *** Common code used by all three pressure ranges ***
; ******************************************************
;
C30:
LDA #$00
STA RTC_REG1
STA RTC_CTRL1 ; Clear IRQ flag
LDA #$01
STA RTC_REG2
STA RTC_CTRL2 ; Mask the interrupt from RTC on NMI
;
LDA #$00
LDU #CALING ; Since zero, we are not presently calibrating
LBSR PUT_PCU
LDA #$FF ; Flag saying that we are in this routine (in case of RESET key pressed)
LDSR PUT_PCU
C31:
LDY #LAST_ATM ; Addr of last atmospheric reading(XDCR1 units ABS mode)

Software 161
STY 136, S
LDY #ATM_TBL ; Addr of atmospheric when

 table was read(PSI ABS mode)
STY 138, S
LDX #FP_PRCNT_DIF ; Min. tolerance
LD16 (120,S),(0,X)
LD16 (122,S),(2,X)
LDA XDCR_CONFIG+XC_UNITS
CMPA #PSI
BEQ C31b ; Branch if PSI already
MAKEFPCCB
FP_SUBTRACT (15+8, S),(120+8, S),(120+8, S) ; Subtract off offset
FP_DIVIDE (120+8, S),(11+8, S),(120+8, S) ; Tolerance value in PSI
KILLFPCCB
C31b:
MAKEFPCCB
FP_DIVIDE {{136+8, S}},(11+8, S),(184+8, S) ; Most recent ATM in PSI
FP_SUBTRACT {{138+8, S}},(184+8, S),(140+8, S) ; Diff of present and table
atmospheric(in PSI)
KILLFPCCB
C32:
MAKEFPCCB
FP_SUBTRACT (68+8, S),(72+8, S),(76+8, S) ; Y2-Y1
FP_SUBTRACT (60+8, S),(64+8, S),(80+8, S) ; X2-X1
FP_DIVIDE (76+8, S),(80+8, S),(84+8, S) ; M=(Y2-Y1)/(X2-X1)
FP_MULTIPLY (84+8, S),(60+8, S),(88+8, S) ; M*X1
FP_SUBTRACT (88+8, S),(68+8, S),(92+8, S) ; B=Y1-M*X1
FP_SUBTRACT (184+8, S),(180+8, S),(132+8, S) ; DESIRED - LAST_ATM(in PSI)
FP_MULTIPLY (50+8, S),(132+8, S),(132+8, S) ; A percentage of above = X3
FP_ADD (132+8, S),(184+8, S),(132+8, S) ; LAST_ATM + % of diff btw
desired and X1= X3
in PSI
FP_SUBTRACT (140+8, S),(132+8, S),(132+8, S) ; X = X3 - (LAST_ATM - ATM_TBL)
FP_MULTIPLY (84+8, S),(132+8, S),(96+8, S) ; M*X
FP_ADD (96+8, S),(92+8, S),(100+8, S) ; Y3=M*X+B
KILLFPCCB
C33:
MAKEFPCCB
FP_TO_D32 (100+8, S),(128+8, S) ; Desired DAC value in Hex
KILLFPCCB
LD16 (104,S),(130,S)
LDD 104, S ; Points to FP value read
LEA 106, S ; X is not zero so display
LDA XDCR_CONFIG+XC_MODE
TSTA
BEQ C34 ; Jump if in ABS mode
MAKEFPCCB
FP_ADD (LAST_ATM),(106+8, S),(188+8, S); Add atmosphere to value
KILLFPCCB
BRA C35
C34:
LD16 (188, S),(106, S) ;

Software 162
C35:
LD16 \{(190,S),(108,S)\}

LDA XDCR_CONFIG+XC_UNITS ; Is XDCR1 in PSI
CMPA #PSI
BEQ C36 ; Branch if in PSI
MAKEFPCB

FP_SUBTRACT \{(15+8,S),(188+8,S),(188+8,S)\} ; Subtract off offset
FP_DIVIDE \{(188+8,S),(11+8,S),(188+8,S)\} ; Read value in PSI ABS
KILLFPCB

C36:
MAKEFPCB

FP_SUBTRACT \{(188+8,S),(180+8,S),(124+8,S)\} ; Subtract actual output from desired

LD16 \{(132+8,S),(124+8,S)\} ; Save this value
LD16 \{(134+8,S),(126+8,S)\}
LDA 124+8,S
ANDA #$7F ; Strip off the sign
STA 124+8,S
FP_COMPARE \{(120+8,S),(124+8,S)\} ; Is the difference in range
KILLFPCB

LBGT C40 ; Jump if so
MAKEFPCB

FP_MULTIPLY \{(50+8,S),(132+8,S),(132+8,S)\} ; % of diff
FP_MULTIPLY \{(132+8,S),(124+8,S),(124+8,S)\} ; M * % diff
FP_ADD \{(100+8,S),(132+8,S),(100+8,S)\} ; Y3 = Y2 + M* (%) * (DESIRED_FP - X2)

KILLFPCB

LBRA C33

; C40:
LDU #DAC_VALU ; Point to destination
LEAX 100,S ; Point to number
LDB #04 ; 4 bytes to store
C40a:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,C40a ;

; LDU #HEX_DAC ; Point to destination
LEAX 104,S ; Point to number
LDB #02 ; 4 bytes to store
C40b:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,C40b ;

; LDU #DESIRED_P ; Point to destination
LEAX 3,S ; Point to number
LDB #04 ; 4 bytes to store
C40c:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,C40c ;
LDA XDCR_CONFIG+XC_UNITS ; Units of XDCR1
CMPA #PSI ; Is it in PSI?
BEQ C40ca ; Branch if so
MAKEFCB
FP_MULTIPLY {84+8,S},{11+8,S},{84+8,S} ; Convert slope to XDCR 1 units
FP_ADD {84+8,S},{15+8,S},{84+8,S} ;
KILLFCB
C40ca:
LDU #FPSLOPE ; Point to destination
LEAX 84,S ; Point to number
LDB #$04 ; 4 bytes to store
C40d:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,C40d
; LDU #XDCR1_FPP2 ; Point to destination
LEAX 106,S ; Point to number
LDB #$04 ; 4 bytes to store
C40e:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,C40e
; C40f:
LDU #XDCR1_FPP ; Point to destination
LEAX 106,S ; Point to number
LDB #$04 ; 4 bytes to store
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,C40f
; CLRA #GT_SET_BFOR ; Clears the fact that
; revs_slope has been
; calculated before
LBSR PUT_PCU ; Store byte in checksum area
LDA #$FF ; Calibrating.
LDU #CALING ; Store byte and update
LBSR PUT_PCU
; checksum
; LDA RTC_REG1 ; Memory storage of RTC Cd reg.
ANDA #$FB
STA RTC_REG1
STA RTC_CTRL1
LDA #$06
STA RTC_REG2
STA RTC_CTRL2 ; Enable interrupt from RTC on
NMI
;
CLRA

Software 164
LDU
LBSR
IN_CONTROL
; No longer in control routine
PUCU
; Store byte and update
checksum
C42:
LEAS
196, S
PULS
X, Y, U, D
RTS
C41:
PUSHQUEUE
FPQ, A
BRA
C42
;
C50:
LDX
#TEMP_VAR-4
STX
56, S
MAKEFCB
FP_COMPARE
{180+8, S}, {[56+8, S]}
; Last location of high table
high table
KILLFCB
BLT
C50a
; Branch if smaller than max.
Therefore it
C50b:
LDX
#SAY_BIG
BRA
C51
C50a:
LDX
#FPLS_TABLE
; First location of low table
i.e. smallest value
STX
56, S
LDX
#SAY_SMALL
C51:
LDA
#02
; Count used to output 2 lines
C52:
LBSR
PRINT_LB
; Print line
LBSR
TIME_DELAY
; Delay between displayed lines
LBSR
TIME_DELAY
; Delay between displayed lines
LBSR
TIME_DELAY
; Go to next lines to display
DECA
; Decrement counter
BNE
C52
; Counter not zero then update
display
LBSR
PRINT_L1
; Display top line
MAKEFCB
FP_TO_BCD
{[56+8, S]}, {3+8, S}, #$0008
; Converts FP number to BCD
with 8 digits
KILLFCB
LDB
#$01
; Print on bottom line
LEAX
3, S
LBSR
DISP_X1_FMT
LBSR
TIME_DELAY
LBSR
TIME_DELAY
LBSR
TIME_DELAY
LBSR
TIME_DELAY
LBRA
C2
;
NOTINST:
LDX
#NOT_IN
LBSR
PRINT_L1
LDX
#PCU_LABELS
LEAX
A, X
; Points to line to display
; Print top line
; Address of PSI range
; Points to "NOT INST"

Software  165
LBSR PRINT_L2 ; Print bottom line
LBSR P_WAIT ; Wait for key pressed
LBRA C40

CTRL1: .BYTE ' ', 'INTERNAL', 'PRESSURE'
CTRL1: .BYTE 'IS FROM', 'XDCR 1'
CTRL1: .BYTE ' ', ' ', 'PCU CARD'
CTRL3: .BYTE 'INSTALLD', ' IS '
CTRL4: .BYTE 'OUT IS '
TOO_BIG: .BYTE 'TOO BIG '
TOO_SMALL: .BYTE 'TO SMALL'
SAY_SMALL: .BYTE ' ', 'MIN TBL ', 'VALUE IS'
SAY_BIG: .BYTE ' ', 'MAX TBL ', 'VALUE IS'
NOT_IN: .BYTE 'CONTROLR'

Software 166
Procedure: EDIT INPUT PRESSURE RANGE: DEFN_RANGE

Revision: 0.0 07/15/88

Written by: Greg Hulan

Purpose: Allows the user to define the maximum and minimum pressure values allowed as output.

Inputs: none

Outputs: none

Uses: none

Alters: none

Registers: byte register storage

System Stack: none

Macros: none

Calls: PRINT_LB: Displays msg

P_WAIT: Pauses for input

Is called by: /////

Revisions: 07/15/88 Greg Hulan: Code written

;******************************************************************************

DEFN_RANGE: PSHS X,Y,U,D

LEAS
-44,S
; Allocate local storage

DRO:
LDY #USER;PCUMAX
; Points to max in (user definable)

STY 40,S
CLR 0,S
; Start with max in

DR1:
LDY 40,S
LDB 0,S
ASLB
ASLB
LEAY B,Y
; Point to this term

DR2:
STY 1,S
MAKEFPCB
FP_TO_BCD [{1+8,S},{3+8,S},#$0008
KILLFPCB
LEAY 30,S
LEAX 30,0
; Pnt to storage

LEA 3,0
; Pnt to BCD

LDA #$08
; 8 byte fmt wanted

LBSR FFLFP
; Do it

LEAX 30,0

DR3:
STX 43,0
LBSR PRINT_L2
; Save addr of value to print

LDB 0,0
; Display setting

LDB #RNGE_LIST
; Point to term titles

LDB 0,0
; Get term #

ASL3 B
; x8

CLR

LEAX D,X
; Point to value

LBSR PRINT_L1
; Displays top line

LDB 43,0
; Displays top line

LDB #DISP_LINE2
LBSR BLINKKEY
; Get input

CMPA #$40
BGT DR7
; Menu?
BRANCHIF A, #ETX, DR6 ; Clear?
BRANCHIF A, #CR, DR4 ; Enter?
BRANCHIF A, #’<’, DR8 ; Increment?
BRANCHIF A, #’>’, DR9 ; Decrement?
PUSHQUEUE FPQ, A ; Otherwise put back on stack

DR4:
LDX #RNGE_LIST
LDB 0, S
ASL3 B
CLRA
LEAX D, X
LBSR PRINT_LI
LEAX 3, S
LBSR GET_FP
CMPA #ETX
LB EQ DR1 ; Back up if so
TSTA ; Successful input?
BNE DR6 ; Abort edit if not
LDU 1, S
LDB #$04 ; 4 bytes total

DR5:
LDA , X+ ; Get byte
LBSR PUT_PCU
LEAU 1, U ; Advance to next
DJNZ B, DR5 ; & continue edit session

DR6:
LEAS 44, S ; Release local storage
PULS X, Y, U, D
RTS

DR7: PUSHQUEUE FPQ, A ; Put menu key back on stack
BRA DR6 ; & abort

DR8: INC 0, S ; INCREMENT
LDA 0, S
CMPA #RNGE_OPTS ; wraparound?
LBNE DR1
CLA 0, S
LBRA DR1

DR9: DEC 0, S ; DECREMENT
LDA 0, S
CMPA #$FF ; wraparound?
LBNE DR1
LD8 {0, S}, #RNGE_OPTS-1
LBRA DR1

효율성 168
**Procedure:** TURN OFF PRESSURE AND VENT TO ATM: VENT_OFF

**Revision:** 0.0 02/01/89

**Written by:** Greg Hulan

**Purpose:** Turn off pressure control output and vent the pressure to atmosphere

**Inputs:** none

**Outputs:** none

**Uses:** none

**Alters:** none

**System Stack:** 1 byte register storage

**Macros:** none

**Calls:**
- CALC2: Calculate FP pressure and displays it.
- PUT_PCU: Stores value and updates checksum

**Is called by:** RESET and CONTROLLER

**Revisions:** 02/01/89 Greg Hulan: Code written

---

**Software**

```
VENT_OFF: PSHS X,Y,U,D
          LEAS -1,S
          
LDA #$00 ; Controlling real time clock
STA RTC_REG1
STA RTC_CTRL1 ; Clear IRQ flag
LDA #$01
STA RTC_REG2
STA RTC_CTRL2 ; Mask the interrupt from RTC on NMI

LDX #TO ; Addr of words to display
LBSR PRINT_LB
LDA #$0C ; Close valve 2 and 3. open the others
           STA PCU_YALVE
           LDU #LAST_VALVE
           LBSR PUT_PCU ; Last valve setting

           LEAY 0,S ; Dummy store
           LDD #$8000 ; Write zero to D/A
           LBSR CALC2
           LDA #$00
           STA PCU_YALVE ; Turns off all valves
           LDU #LAST_VALVE
           LBSR PUT_PCU ; Last valve setting

           LEAY 1,S ; Store byte and update
           LDD #$8000
           LBSR CALC2
           LDA #$00
           STA PCU_YALVE ; No longer calibrating
           LDU #CALING
           LBSR PUT_PCU ; Store byte and update

           LEAS 1,S
           PULS X,Y,U,D
           RTS

TO: .BYTE 'TURN OFF', ' & VENT ' ; To be written to display
```
Procedure: TURN_OFF

Purpose: Turn off pressure control output and clamp the pressure in the line

Inputs: none

Outputs: none

Alters: none

System Stack: 1 byte register storage

Revision: 0.0 02/01/89

Written by: Greg Hulan

Turn off pressure source but clamp pressure: CLAMP_OFF

CALL 2: Calculate FP pressure and displays it.

PUT_PCU: Stores value and updates checksum

Is called by: RESET and CONTROLLER

Revisions: 02/01/89 Greg Hulan: Code written

CLAMP_OFF:

PSHS X, Y, U, D

LDA #$00
STA RTC_REG1
STA RTC_CTRL1
LDA #$01
STA RTC_REG2
STA RTC_CTRL2

NMI

LDX #TOC
LBSR PRINT_LB
LDA LASTVALVE
ORA #$08
ANDA #$0B
STA PCU_VALVE

first

LDU #LASTVALVE
LBSR PUT_PCU

checksum

LBSR TIME_DELAY
LDA #$08

others

STA PCU_VALVE

valves

LDU #LASTVALVE
LBSR PUT_PCU

checksum

LDD #$8000
STB PCU_DA_2
STA PCU_DA_1
LDA #$00
LDU #CALING

Software 170
LBSR PUT_PCU ; Store byte and update checksum
PULS X, Y, U, D
RTS

TOC: .BYTE 'TURN OFF', '& CLAMP'

;------------------------------------------------------------------------------
; * Procedure: Calculate FP values: PCU_PRCENT
; * Revision: 0.0 07/22/88
; * Written by: Greg Hulan
; * Purpose: Calculates the floating point value of the PCU input
; * precentage that is sent out, and the % full scale that
; * the output can differ from that desired.
; * Inputs: none
; * Outputs: none
; * Uses: none
; * Alters: none
; * Registers: *
; * System Stack: byte register storage
; * Macros: none
; * Calls: PRINT_LB: Displays msg
; * P_WAIT: Pauses for input
; * Is called by: // //
; * Revisions: 07/22/88 Greg Hulan: Code written

;------------------------------------------------------------------------------

PCU_PRCENT: PSHS X, Y, U, D ; Save registers
LEAS -46, S ; Creates local storage
LEAX 0, S ; Beginning of FP #
CLRB ; Clear FP #
LDA #25 ; Counter
PC1:
STB , X+ ; Clear
DECA
BNE PC1
LDD #PCENT_JMP ; % of desired that is output
STD 21, S ; FPCB =

0000000000000000000000000000000000X2
LDB #$04
STB 25, S ; 2 #’s to the right of dec.

point
MAKEFPCB
BCD_TO_FP {0+8, S}, {26+8, S} ; 4 byte FP # for PCENT_JMP
KILLFPCB ; at S+26, 27, 28, 29
LDX #PCU_FP_MAXIN ; Hardware max pressure
LD16 {34, S}, {0, X}
LD16 {36, S}, {2, X}
LDD #PCENT_DIF ; % difference allowed betw

desired and actual
STD 21, S ; FPCB =

0000000000000000000000000000XXXX5
LDB #$07 ; 5 #’s to the right of dec.

point

Software 171
STB 25, S
MAKEFPCB
BCD_TO_FP (0+8,S),(30+8,S) ; 4 byte FP # for PRCNT_DIF
FP_MULTIPLY (30+8,S),(34+8,S),(30+8,S) ; #diff scaled for PCU range in PSI
KILLFPCB
LDU #FP_PRCNT_DIF ; Point to destination
LEAX 30, S ; Point to number
LDB #$04 ; 4 bytes to store
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1, U ; Advance to next store
DJNZ B,PC1a ; Has user defined a tolerance
LDA TOL_SET ; Branch if so
CMPA #$FF
LBEQ PC2 ; Is XDCR1 in PSI
LDA XDCR_CONFIG+XC_UNITS
CMPA #$PSI
BEQ PC1b ; Jump if so
LDX #XDCR_CONFIG+XC_MTERM ; Otherwise get the slope and offset
LD16 (38,S),(0,X)
LD16 (40,S),(2,X)
LD16 (42,S),(4,X)
LD16 (44,S),(6,X)
MAKEFPCB
FP_MULTIPLY (38+8,S),(30+8,S),(30+8,S) ; Change default tolerance to XDCR1 units
FP_ADD (42+8,S),(30+8,S),(30+8,S)
KILLFPCB
PC1b:
LDU #TOLERANCE ; Point to destination
LEAX 30, S ; Point to number
LDB #$04 ; 4 bytes to store
PC1c:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1, U ; Advance to next store
DJNZ B,PC1c ; since they have not defined it yet. (in PSI)
PC2:
LDU #FP_PRCNT_JMP ; Point to destination
LEAX 26, S ; Point to number
LDB #$04 ; 4 bytes to store
PC3: ; Get byte
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1, U ; Advance to next store
DJNZ B,PC3 ; Release registers
LEAS 46, S
PULS X,Y,U,D

Software 172
; Procedure: CHECKS CAL. PRESSURE:CHECK_P
; Revision: 0.0 10/18/88
; Written by: Greg Hulan
; Purpose: Checks to see if the calibration pressure sent out is in the range required, and if not, puts it to this desired value.
; Inputs: none
; Outputs: none
; Uses: none
; Alters: none
; System Stack: byte register storage
; Macros: none
; Calls: 'k
; Is called by: NMI interrupt service routine
; Revisions: 10/18/88 Greg Hulan: Code written

;CHECK_P: PSHS X,Y,U,D ; Save registers
; LEAS -40,S ; Create local storage
; LDY #XDCR_CONFIG+XC_MTERM
; LD16 {32,S},{0,Y} ; Slope of XDCR1
; LD16 {34,S},{2,Y}
; LD16 {36,S},{4,Y} ; Offset
; LD16 {38,S},{6,Y}
; LD16 {0,S},DESIRED_P ; Desired FP pressure sent out (in xdcrl units and mode)
; LD16 {0+2,S},DESIRED_P+2
; LD16 {4,S},XDCR1_FPP ; Last FP value of XDCR1 sent out (in XDCR1 units and mode)
; LD16 {6,S},XDCR1_FPP+2
; LD16 {8,S},TOLERANCE ; Percentage different they can be (in PSI)
; LD16 {10,S},TOLERANCE+2
;CP0: LDA XDCR1_DIS ; Zero if display has not updating last value
; TSTA ; Branch if do not want to display
;CP0a: LDD HEX_DAC ; Last DAC value sent
; LEAY 4,S
; LDX #$00 ; Do not display XDCR1 FP value read
; LBSR READ_DISP1
;CP0b: MAKEFPCB ; Subtract actual from desired(XDCR1 units and mode)
; LD16 {16+8,S},{12+8,S} ; Save difference
LD16 (18+8,S),(14+8,S) ; Approx. equals zero in FP
LD16 (24+8,S),#$03AA
LD16 (26+8,S),#$2424
LDA 12+8,S
ANDA #$7F
STA 12+8,S
FP_COMPARE (8+8,S),(12+8,S) ; Compare them
KILLFPCB
LBGT CP10
LDD HEX_DAC
STD 30,S
;CP1: MAKEFPCB
FP_COMPARE (16+8,S),(24+8,S) ; Is difference > zero
KILLFPCB
BGT CP1a ; then branch
LDD 30,S ; Desired DAC in Hex
SUBD #$01 ; Decrement DAC
BRA CP1b ;CP1a:
LDD 30,S ; Desired DAC in Hex
ADD0 #$01 ; Increment DAC
;CP1b:
STD 30,S ; Returning address of XDCR1
LEAY 4,S ; read
LDX #$00 ; zero so don't display
pressure
LBSR READ_DISP1 ; Read XDCR1 and display it on
; top display
MAKEFPCB
FP_SUBTRACT (4+8,S),(6+8,S),(12+8,S) ; Subtract actual from
desired(XDCR1 units and mode)
LD16 (16+8,S),(12+8,S)
LD16 (18+8,S),(14+8,S)
LDA 12+8,S
ANDA #$7F
STA 12+8,S
FP_COMPARE (8+8,S),(12+8,S) ; Compare them
KILLFPCB
LBLT CP1
LDU #HEX_DAC ; Point to destination
LEAX 30,S ; Point to number
LDB #$02 ; 4 bytes to store
;CP2:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store
DJNZ B,CP2
LDU #XDCR1_FPP ; Point to destination
LEAX 4,S ; Point to number
LDB #$04 ; 4 bytes to store
;CP3:
LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte
LEAU 1,U ; Advance to next store

Software 174
; DJNZ B, CP3
; CP10: LEAS 40, S ; Release local storage
; FULS X, Y, U, D
; RTS

; There is a problem with this routine. If the user lets the
; output line have open air flow for 2 readings, the DAC will be
; altered to restore the desired. Then when the user replaces the
; output and air flow is changed, the pressure will overshoot and
; it will take at least 1.5 seconds to restore the correct value.
; However, if the user is required to never allow the output
; to be free flow, everything will be fine. If the user wishes to
; protect his device by putting a cutoff valve on the output and
; letting the pressure stabilize before allowing it to reach his
; transducer, he should put the cutoff as close to the device as
; is possible.

CHECK_P: PSHS X, Y, U, D ; Save registers used
; LEAS -73, S
; LDA XDCR1_DIS ; Zero if display has not
updating last value
; TSTA
; BNE CP0b ; Branch if do not want to
display

; LDU #XDCR1_FPP2 ; Point to destination
; LDX #XDCR1_FPP ; Point to number
; ; CP0a: LDA ,X+ ; Get byte
; LBSR PUT_PCU ; Store byte and update
checksum
; LEAU 1, U ; Advance to next store
; DJNZ B, CP0a
; LDD HEX_DAC ; Last DAC value sent
; LEAY 0, S ; Address of store FP read
; LDX #$00 ; Do not display XDCR1 FP value
read
; LBSR READ_DISP1
; LDU #XDCR1_FPP ; Point to destination
; LEAX 0, S ; Point to number
; LDB #$04 ; 4 bytes to store
; ; CP0aa: LDA ,X+ ; Get byte
; LBSR PUT_PCU ; Store byte and update
checksum
; LEAU 1, U ; Advance to next store
; DJNZ B, CP0aa

; CP0b: MAKEFPCB
FP_SUBTRACT {XDCR1_FPP}, {DESIRED_P}, {0+8, S} ; Desired pressure - last
read pressure value

Software 175
FP_SUBTRACT (XDCR1_FPP2), (DESIRED_P), (4+8, S); Desired pressure - sec. to last pressure val
LD16 (8+8, S), (0+8, S) ; Save diff including sign
LD16 (10+8, S), (2+8, S)
LD16 (12+8, S), (4+8, S) ; Save diff including sign
LD16 (14+8, S), (6+8, S)
LDA 8+8, S
ANDA #$7F ; Strip off sign
STA 8+8, S
LDA 12+8, S
ANDA #$7F ; Strip off sign
STA 12+8, S
FP_COMPARE (8+8, S), (12+8, S) ; Which is closer to desired
KILLFFPCB
BLT CP0 ; Branch if last read is farther from desired
LD16 (69, S), (12, S) ; Positive difference
LD16 (71, S), (14, S)
LD16 (16, S), (4, S) ; Use this diff to calc delta
DAC
LD16 (18, S), (6, S)
LD16 (20, S), (XDCR1_FPP2) ; Second to last pressure read is closer so use it
LD16 (22, S), (XDCR1_FPP2+2)
BRA CP1
CP0: LD16 (69, S), (8, S) ; Positive difference
LD16 (71, S), (10, S)
LD16 (16, S), (0, S) ; Use this diff to calc delta
DAC
LD16 (18, S), (2, S)
LD16 (20, S), (XDCR1_FPP) ; Use last pressure read since delta is smaller
LD16 (22, S), (XDCR1_FPP+2)
CP1: MAKEFFPCB
FP_COMPARE (TOLERANCE), (69+8, S) ; Is pressure in tolerance
KILLFFPCB
LBGT SO14 ; What mode is it in
LDA XDCR_CONFIG+XC_MODE ; Is it in ABS mode
TSTA ; Branch if so
BEQ SET_ABS ; Otherwise it's DIFF mode
BRA SET_DIFF ;
;
SET_ABS: MAKEFFPCB
FP_COMPARE (DESIRED_P), (LAST_ATM) ; Is desired > last atm.
KILLFFPCB
BGT GREATER_THAN ; Branch if so
BRA LESS_THAN ;

Software 176
SET_DIFF: MAKEFPCB
FP_COMPARE \( (\text{DESIRED}_P),(\text{ZERO}) \); Is desired > zero
KILLFPCB
BGT GREATER_THAN; Branch if so
BRA LESS_THAN; Otherwise it's less than
;
; Routine compares desired > atm
;
GREATER_THAN: MAKEFPCB
FP_COMPARE \( (\text{DESIRED}_P),(20+8,S) \); Is desired > reading
KILLFPCB
BGT SET_GT; Jump if so
BRA SET_LT; Otherwise <
;
; Routine compares desired < atm
;
LESS_THAN: MAKEFPCB
FP_COMPARE \( (\text{DESIRED}_P),(20+8,S) \); Is desired < reading
KILLFPCB
BLT SET_GT; Jump if so
BRA SET_LT; Otherwise >
;
SET_GT: LDA \#00; Transfer flag saying use FPSLOPE
BRA SET_OUT; and calc new FPSLOPE
;
SET_LT: TST GT_SET_BFOR; Flag equals 1 if we have a reverse slope
BNE SL1
LDA \#01
STA GT_SET_BFOR; Will calc REV_SLOPE
BRA SET_OUT; Use regular slope and calc reverse slope
;
SL1: LDA \#02; Use REV_SLOPE and calc new
REVS_SLOPE
BRA SET_OUT
;
;
SET_OUT: STA 68,S; Save flag
CMPA \#02
BEQ SO0
LD16 \( (24,S),(\text{FPSLOPE}) \); Get last slope stored
LD16 \( (26,S),(\text{FPSLOPE}+2) \)
BRA SO1
SO0: LD16 \( (24,S),(\text{REV_SLOPE}) \); Get last reverse slope stored
LD16 \( (26,S),(\text{REV_SLOPE}+2) \)
;
LDA XDCR_CONFIG+XC_UNITS; Is XDCR1 in PSI
;
CMPA \#PSI
;
BEQ SG1; Branch if so
; LDX #XDCR_CONFIG+XC_MTERM ; Otherwise must mult slope so
; LD16 (28,S),(0,X) ; it's in desired units
; LD16 (30,S),(2,X)
; LD16 (32,S),(4,X)
; LD16 (34,S),(6,X)
; MAKEFPCB
; FP_SUBTRACT (32+8,S),(24+8,S),(24+8,S)
; FP_DIVIDE (24+8,S),(28+8,S),(24+8,S) ; Slope * Units conversion
; KILLFPCB

; SG1:

S01: LD16 (36,S),(DAC_VALU) ; Last DAC value sent and stored
     LD16 (38,S),(DAC_VALU+2)
S02: MAKEFPCB
     FP_MULTIPLY (16+8,S),(24+8,S),(40+8,S) ; M*delta pressure= delta DAC
     FP_ADD (36+8,S),(40+8,S),(36+8,S) ; New DAC value in FP
     FP_TO_D32 (36+8,S),(44+8,S) ; New DAC value in HEX
     KILLFPCB
     LDD 44,S
     CMPD #$0000
     ; If not zero than DAC value overflowed
     BEQ S02b
     MAKEFPCB
     FP_COMPARE (DESIRED_P),(LAST_ATM) ; Is roll over from trying to set too
     KILLFPCB
     BLT S02a
     LDD #$FFFF
     ; Branch if so
     ; Else it must have rolled over by being too high
     ; Output max DAC value
     BRA S02c
S02a: LDD #$0000
     ; Rolled over from zero so sent DAC
     ; minimum possible = 0
S02b: LDD 46,S
     LEAY 48,S
     LDX #$FF
     LBSR READ_DISP1
     MAKEFPCB
     FP_SUBTRACT (48+8,S),(DESIRED_P),(16+8,S) ; Delta pressure
     LD16 (52+8,S),(16+8,S) ; Store it here too
     LD16 (54+8,S),(18+8,S)
     LDA 52+8,S
     ANDA #$7F
     STA 52+8,S
     FP_COMPARE (TOLERANCE),(52+8,S) ; Is tolerance > diff betw desired and actual

Software 178
I KILLFPCB
BGT S03 ; Done if so
LBRA S02 ; Otherwise set pressure again

S03: MAKEFPCB
FP_SUBTRACT (DAC_VALU),(36+8,S),(56+8,S) ; New DAC - Old DAC = Y
FP_SUBTRACT (20+8,S),(48+8,S),(60+8,S) ; New pressure - Old pressure = X
FP_DIVIDE (56+8,S),(60+8,S),(64+8,S) ; New slope
KILLFPCB
LDA 68,5 ; Load A with flag
CMPA #00 ; Is the slope calculated
FPSLOPE
BEQ S06 ; Branch if so
;
CMPA #$02 ; Has REVS_SLOPE been calculate before
BNE S04 ; Branch if not
MAKEFPCB
FP_ADD (REV5_SLOPE),(64+8,S),(64+8,S); Calculating avg slope
FP_DIVIDE (64+8,S),(FP_TWO),(64+8,S) ; (New slope + Old slope)/2
KILLFPCB

S04: LDU #REV5_SLOPE ; Point to destination
LEAX 64,S ; Point to number
LDB #$04 ; 4 bytes to store
S05: LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte and update checksum

LEAU 1,U ; Advance to next store
DJNZ B,S05
BRA S010 ; Done

; S06: MAKEFPCB
FP_ADD (FPSLOPE),(64+8,S),(64+8,S) ; Calculating avg slope
FP_DIVIDE (64+8,S),(FP_TWO),(64+8,S) ; (New slope + Old slope)/2
KILLFPCB
;

LDU #FPSLOPE ; Point to destination
LEAX 64,S ; Point to number
LDB #$04 ; 4 bytes to store
S07: LDA ,X+ ; Get byte
LBSR PUT_PCU ; Store byte and update checksum

LEAU 1,U ; Advance to next store
DJNZ B,S07

; S010: LDU #XDCR1_FPP ; Point to destination
LEAX 48,S ; Point to number
LDB #$04 ; 4 bytes to store
S011: LDA ,X+ ; Get byte
LBSR PUT_PCU
; Store byte and update checksum
LEAU 1, U
DJNZ B, SO11
;
LDU #XDCR1_FPP2
LEAX 48, S
LDB #$04
SO12:
LDA ,X+
LBSR PUT_PCU
; Point to destination
; Point to number
; 4 bytes to store
; Get byte
; Store byte and update checksum
checksum
LEAU 1, U
DJNZ B, SO12
;
LDU #DAC_VALU
LEAX 36, S
LDB #$04
SO13:
LDA ,X+
LBSR PUT_PCU
; Point to destination
; Point to number
; 4 bytes to store
; Get byte
; Store byte and update checksum
checksum
LEAU 1, U
DJNZ B, SO12
SO14:
LEAS 73, S
PULS X, Y, U, D
RTS
;
ZERO: .BYTE $00, $00, $00, $00
FP_TWO: .BYTE $40, $00, $00, $00
.PAGE
I

Procedure: CHECK PCU INTERFACE: CHECK_PCU

Revision: 3.02 7/8/88

Written by: Greg Hulan

Purpose: Test to ensure PCU interface exists

Inputs: none

Outputs: none

Uses: none

Alters: PCU_EXISTS, PCU_RANGE

Registers: none

System stack: 1 byte local storage

Macros: LD8: Performs 8 bit data transfer

Calls: none

Revisions: 07/08/88 Greg Hulan: Code written

CHECK_PCU: PSHS U,D ; Save registers used
LEAS -1,S ; Allocates local storage
LDA PCU_TYPE ; Loads type of PCU
STA 0,S ; Save PCU type
CMPA #$FF ; Is it neither solar or dig?
BEQ NO_PCU ; Branch it so
ANDA #$10 ; Mask all but Solar/Dig bit
BNE SOL_PCU ; Solartron PCU - branch if so
LDA 0,S ; Otherwise it's a Digiquarts
ANDA #$0F ; Mask off high half
DECA ; Now it is value of PCU RANGE
LDU #PCU_RANGE ; Save range and checksum
LBSR PUT_PCU
BRA CKP1

SOL_PCU: LDA 0,S ; Loads type of PCU
ANDA #$0F ; Mask off high half
ADDA #$9 ; Add 9 to get to PCU range of solar.
LDU #PCU_RANGE ; Save range and checksum
LBSR PUT_PCU
BRA CKP1

NO_PCU: LDA #NONE ; Load A with PCU range meaning no card
LDU #PCU_RANGE ; Save range and checksum
LBSR PUT_PCU
BRA CKP1

CKP1: LDU #DISP_CONFIG+3*DC_SIZE+DC_CNT ; Contents of internal display
CLRA
LBSR PUT_CHECK
LEAU 1,U
LBSR PUT_CHECK
LEAU 1,U
LBSR PUT_CHECK

Software 181
LEAS L,S ; Remove local storage
FULS U,D
RTS

;************************************************************************************

;* Procedure: VERIFY PCU RELATED RAM: CHECK_PCU_D *
;* Revision: 1.0 1/30/89 *
;* Written by: Greg Hulan *
;* Purpose: Verifies PCU-related ram against stored checksum & *
;* warns user if checksum error is detected. *
;* Inputs: none *
;* Outputs: none *
;* Uses: none *
;* Alters: none *
;* Registers: A register calculates checksum *
;* X register points to data area *
;* System stack: 4 byte register storage *
;* Macros: PULLQUEUE: Removes data from a queue *
;* Calls: SET_DEF: Sets all configuration data to defaults *
;* PRINT_LB: Displays msg on front panel *
;* P_WAIT: Wait for key *
;* REMQUEUE: (via macro) *
;* Revisions: 01/29/89 Greg Hulan: Code written *

;************************************************************************************

CHECK_PCU_D: PSHS X,D ; Save registers used
LDA PCU_RANGE ;
CMPA #$0FFH
BEQ CPD2 ; Skip if PCU not installed
LDX #VAR_CONTROL ; Point to PCU related data
CLRA ; Initialize checksum

CPD1: ADDA ,X+ ; Add in data
CMPX #FFLS_TABLE ; At end of data?
BNE CPD1 ; Repeat if not
LDX #TEMP_VAR ; Beginning of next section

CPD1a: ADDA ,X+ ; Add in data
CMPX #PCUSUM ; At end of data?
BNE CPD1a ; Repeat if not
CMPA PCUSUM ; Does checksum match?
BEQ CPD2 ; Go exit if so
STA PCUSUM ; Store new checksum
LDX #PCU_D_BAD ; Warn user config is bad
LBSR PRINT_LB ; Display msg
LBSR P_WAIT ; Wait for key
PULLQUEUE FPQ,A ; Clear out queue

CPD2: FULS X,D ; Restore registers
RTS
.PAGE
;*************************************************************************
;* Procedure: WRITE_BYTE_TO_PC-RELATED_AREA: PUT_PCU
;* Revision: 1.0 01/30/89
;* Written by: Greg Hulan
;* Purpose: Updates checksum & stores byte into PCU-related RAM
;* Inputs: U register points to address
;* Outputs: none
;* Uses: none
;* Alters: PUTSUM: Coefficient table checksum
;* Registers: B register used to adjust checksum
;* System stack: 1 byte register storage
;* Macros: none
;* Calls: none
;* Is called by: /////<
;* Revisions: 01/30/89 Greg Hulan: Code written
;*************************************************************************

PUT_PCU:
PSHS B ; Save registers used
LDB PCUSUM ; Alter checksum
SUBB ,U ; Pull out old value
STA ,U ; Save new value
ADDB ,U ; Add in new value
STB PCUSUM ; Update checksum
PULS B ; Restore registers used
RTS
;**********************************************************************************************

;* Procedure: VERIFY PCU RELATED RAM: CHECK_TABLE
;* Revision: 1.0 1/30/89
;* Written by: Greg Hulan
;* Purpose: Verifies PCU-related ram against stored checksum & warns user if checksum error is detected.
;* Inputs: none
;* Outputs: none
;* Uses: none
;* Alters: none
;* Registers: A register calculates checksum
;* X register points to data area
;* System stack: 4 byte register storage
;* Macros: PULLQUEUE: Removes data from a queue
;* Calls: SET_DEF: Sets all configuration data to defaults
;* PRINT_LB: Displays msg on front panel
;* P_WAIT: Wait for key
;* REMQUEUE: (via macro)
;* Revisions: 01/29/89 Greg Hulan: Code written

;**********************************************************************************************

CHECK_TABLE: PSHS X,D ; Save registers used LDA PCU_RANGE CMPA #$OFFH BEQ CTB2 ; Skip if PCU not installed LDX #FPLS_TABLE CLRA ; Initialize checksum

CTB1: ADDA ,X+ ; Add in data CMPX #TEMP_VAR BNE CTB1 ; At end of data? CMPA PCUTABLESUM BEQ CTB2 ; Does checksum match? STA PCUTABLESUM BEQ CTB2 ; Go exit if so STA PCU_T_BAD ; Warn user config is bad LBSR PRINT_LB ; Display msg LBSR P_WAIT ; Wait for key FULQUEUE FPQ,A ; Clear out queue

CTB2: PULS X,D ; Restore registers RTS

.SPAGE

Software 184
 Procedure: WRITE_BYTE_TO_TABLE: PUT_TABLE
 Revision: 1.0 01/30/89
 Written by: Greg Hulan
 Purpose: Updates checksum & stores byte into PCU Table RAM
 Inputs: U register points to address
 Outputs: none
 Uses: none
 Alters: PCUTABLESUM: Coefficient table checksum
 Registers: B register used to adjust checksum
 System stack: 1 byte register storage
 Macros: none
 Calls: none
 Is called by: /////
 Revisions: 01/30/89 Greg Hulan: Code written

;*************************************************************************

; Procedure: WRITE_BYTE_TO_TABLE: PUT_TABLE
 ; Revision: 1.0 01/30/89
 ; Written by: Greg Hulan
 ; Purpose: Updates checksum & stores byte into PCU Table RAM
 ; Inputs: U register points to address
 ; Outputs: none
 ; Uses: none
 ; Alters: PCUTABLESUM: Coefficient table checksum
 ; Registers: B register used to adjust checksum
 ; System stack: 1 byte register storage
 ; Macros: none
 ; Calls: none
 ; Is called by: /////
 ; Revisions: 01/30/89 Greg Hulan: Code written

;*************************************************************************

PUT_TABLE: PSHS B ; Save registers used
LDB PCUTABLESUM ; Alter checksum
SUBB ,U ; Pull out old value
STA ,U ; Save new value
ADDB ,U ; Add in new value
STB PCUTABLESUM ; Update checksum
PULS B ; Restore registers used
RTS
.END
Vita

G. T. Hulan was born in U.S. military hospital in Okinawa, Japan in 1964. He completed high school in 1982 at First Colonial, Va. Beach, and received a Bachelor of Science degree in Electrical Engineering from Virginia Tech, in June, 1987. Winter quarter 1986, he and his roommate withdrew from school to work at Keystone Resort in Colorado so as to enjoy skiing on a daily basis. Since receiving his B.S. degree, he has been pursuing a Master’s degree in Electrical Engineering and has completed the necessary requirements to receive that degree. His interests include surfing, windsurfing, snow skiing, softball, and other outdoor sports.