Table of Contents

i. Title Page
   ii. Table of Contents
   iii. List of Figures
   iv. List of Tables
       1. Abstract
       2. Background and rationale
       3. Narrative
       4. Environment
       5. Procedure
          i. Chip Description
          ii. Sending Data to a Personal Computer
          iii. Real Time Clock
          iv. The Control Module
          v. Control Program
          vi. Program Design
          vii. Circuits and Equipment
          viii. Designing the interfacing circuitry
          ix. Designing and Etching of the PCB
          x. Control Program Testing
       6. Results
       7. References
       8. Generating boolean expressions (Appendix – A)
       9. PIC chip pin descriptions (Appendix – B)
      10. PicStic – PC communication (Appendix – C)
      11. Design of PCB (Appendix – D)
      12. Equipment Specifications (Appendix – E)
      13. COSS project members (Appendix – F)
      14. Control program listing (Appendix – G)
      15. Data Definition

List of figures

1. Block diagram of control scheme (figure 1) 8
2. View from component side of chip (figure 2) 14
3. Routine control diagram (figure 3) 20
4. Flow chart for the control program (figure 4) 21
5. Interfacing circuitry for the chip (figure 5) 30
6. DB-25 connections (figure 6) 15
7. DB-9 connections (figure 7) 16
8. Dimensions of float switches (figure 8) 28
9. Printout of the circuit diagram taken from the “Tango Schematics” software Appendix – D
10. Printout of PCB’s copper linings Appendix – D
11. Printout of the PCB with components mounted Appendix – D
12. Printout of the PCB containing both the components and the copper connections Appendix – D

List of Tables

1. Control decisions for the operation of vacuum receiver 9
2. Specifications for IEC miniature contactors 58
Abstract

This project is the design and implementation of a control system using the PIC16C84 chip of Microchip, Inc. to automate a filtration process which is a part of the Coastal Oil-Spill Simulation System (COSS) project. This involves a thorough investigation of the characteristics of the PIC16C84 to develop a program to provide the proper control functionality. It also involves designing circuits to interface the chip with the equipment.
Background and Rationale

Coastal Oil-Spill Simulation System (COSS) is a research facility that was originally designed to study the fate effects of oil-spill remediation agents on simulated nearshore marine and estuarine ecosystems. Research includes the use of oil dispersants, fine particle (clay) and oil interactions, and biological treatments.

Description of the Current Wastewater Treatment System

Currently, wastewater follows a treatment scenario that includes an Oil-Water Separator (OWS), followed by carbon filtration and UV sterilization when applicable. The OWS is designed and constructed by ACS Industries. These units are intended to recover large quantities of free oil from the water as typified in a major oil spill, not emulsified oils. The 200-gpm positive-displacement oil pump is an indicator of the unit's intended use. This pump’s function is to skim the oil from the top of the water inside the separator. In fact, the unit does not detect less than approximately 200 gallons of oil in the separator (the total capacity of the OWS is 3284 gallons). The unit primarily separates the oil and water via gravity. The unit also includes oil-coalescing pads to aid in the gravity separation of insoluble oil droplets. Following the OWS the effluent is treated with the carbon polishing units prior to being discharged. UV sterilization is the final treatment in the event that biological treatment agents (i.e. non-indigenous bacterial cultures) are used.

Drawbacks of the Wastewater Treatment System

Historical use of the system has demonstrated a number of deficiencies including the inability to handle emulsified oils and the apparent lack of solids-handling capabilities. Consulting with the group which uses the COSS facility for their research indicates that the Oil-Water Separator never worked as intended.
Solution

To overcome the design problems of the Oil-Water Separator, it is intended to convert the Oil-Water separator into a vacuum receiver, and then use it to feed the Oil-Water mixture to a subsequent filtration system in a controlled manner to carryout the separation. It is this flow of the Oil-Water mixture to the filtration system that needs to be automated. A control module is needed to operate the vacuum and drain pumps to automate the flow of the Oil-Water mixture.

The aim of this project is to implement the above solution and contribute to the research group, which is working to revive the COSS facility for their research.

A list of major persons involved in the COSS project are given in Appendix – F.
Narrative

1. Overview of the system

The goal of this project is to convert an oil-water separator into a vacuum receiver. It is necessary to utilize the existing equipment where ever possible.

The oil-water mixture from the vacuum receiver is fed to a filtration system. For this, the operation of the vacuum and the drain pumps needs to be automated. Sufficient time needs to be given for the pumps between their turn on and turn off times which is very critical for the operation of the pumps. This project is to develop the control module to safely operate the high capacity pumps and thereby automate the flow of the oil-water mixture.

2. Description of the system

The present oil-water separator is equipped with a vacuum pump, a drain pump that was supposed to drain out the upper oil layer, and an inlet for the incoming oil-water mixture. It is also equipped with a number of oil-water sensors. It is intended to replace these oil-water sensors with float switches which will be used as level indicators in the new system (refer to Figure 1). Based on these level indicators (float switches) the vacuum pump and the drain pump will be operated (switched on or switched off).

A vacuum is applied from an opening at the top of the oil-water separator, while there is an inlet near the bottom of the vessel for the oil-water mixture. The outlet for the drain pump is located at a point that is at a higher level than the inlet to the oil-water separator (see diagram in Figure 1). The three float switches (A, B, C) will be placed above this outlet of the drain pump. One of the requirements of the new system is to maintain the oil-water mixture level between the upper and lower float switches. From the viewpoint of the vacuum receiver it is desirable to maintain the oil-water mixture level between the upper two float switches, but it would greatly tax the vacuum receiver (the cycle time would be much less and would result in much wear and tear). To avoid this, three switches are used thereby increasing the cycle time for the pumps. The minimum required cycle time for the pumps is approximately 20 seconds.

There is a backup system for the vacuum pump. Whenever the fluid level in the tank increases beyond a certain threshold level, the fluid enters a small vacuum receiver attached to the oil-
water separator which shuts down the vacuum pump. The drain pump can be operated manually apart from being controlled by the control module. Whenever the fluid falls below the inlet for the drain pump, it pumps air which is innocuous for the operation of the filtration system and also does not affect the functioning of the drain pump.

The researchers who will be using this new system have specified the required operating conditions for the vacuum pump and the drain pump (with regard to the oil-water mixture level). A detailed analysis of the requirements was done and is shown in Appendix – A.

3. Control System:

The control system to implement the decisions shown in Table 1 was developed using a PIC16C84 chip from Microchip Inc. The chip has eight input/output pins (5VDC) of which three will be used for input (from the float switches), and two for output (to the pumps). The remaining three are kept in reserve to accommodate any new requirements to be specified by the group that will be using the system. The chip will be programmed using PicBasic, a version of BASIC designed specifically for the chip.

There have been some additions to the requirements specified by the research group that is going to use the control module which will be referred to as ‘float controller’ hereafter. It is desirable to have a serial communication between the chip and a PC or a printer, since the PIC chip can be programmed to send some log information about the functioning of the control module. Since the PIC chip implements decisions whenever the states of the float switches change, it can be made to simultaneously send information regarding the decisions taken to a PC (or a printer) via a serial port along with a time stamp. This information may be useful while conducting the COSS experiments and may be used to improve the performance of COSS equipment.
Figure 1: A schematic diagram of the proposed control scheme.

A, B, C : Float switches  
E : Drain pump  
D : Vacuum pump  
F : Electro-Pneumatic valve (proposed)  
G : Inlet valve for Oil-Water mixture

Of the five lines going to the chip, three are input to the chip from the float switches and two are outputs from the chip to the pumps.

The oil-water separator is a rectangular tank with a height of 5'9", breadth 5' and a length of 15'1/4". The distance between the float switches to allow for the required cycle time was calculated and the results are shown in the ‘Control Program’ sub-section.

Details regarding the control module.

It is intended that the proposed control module should have a very simple interface with the other equipment. This will make the connection of the control module with other equipment straightforward. Accordingly, it is desirable to have a black-box-like module where the necessary input and output lines can be easily connected (along with the necessary power feed and other
necessary lines). Then the controller will be able to take charge of the vacuum receiver.

To achieve the simple interface the control module must be shrunk to dimensions which will make it small and portable. This is planned by etching out a circuit board (the details of etching can be found in Appendix – D) and mounting the chip, solid state switches, terminals and other parts on this circuit board.

Operating the control module.

The float controller will be connected to the three float switches and the two pumps (these pumps will be started and shutdown by the chip using motor starters). In order to use the vacuum receiver the inlet to the vacuum receiver should be opened, and since the vacuum pump should be on to start with, the control module must be switched on (since it controls the starting of the pumps). From here on the float controller will make decisions to maintain the oil-water-mixture level and turn on and off the pumps as (flow chart shown in figure 4. As the cycle time is controlled by placing three float switches and making decisions based on the float-switch states instead of measuring the actual cycle time for the pumps, the control module will work independent of the flow rates of the two pumps. For the oil-water-separator (which receives the oil-water-mixture from the vacuum receiver) to work, the fluid level has to be maintained above the outlet to the drain pump. The outgoing streams from the oil-water-separator will be subjected to further filtration.

4. Equipment:

A preliminary list of the main equipment (apart from the chip-related hardware) needed for the new system is as follows:

1. Float Switches
2. Electro-Pneumatic valve

The specifications for all the above equipment are given in Appendix – E.

A simplified diagram of the proposed new system was shown in Figure 1.
Environment

The control module has been developed using a PIC16C84 chip of the Microchip Inc. The chip will be programmed using PicBasic. This module can read the states of the float switches (magnetically actuated dry-reed switches) and turn on/off the motors using motor starters via solid-state switches. The module will also be capable of sending data to a PC via a serial port so that a person can monitor the working of the vacuum receiver from a control room. The printed circuit board (PCB) for the control module will be developed using software products called “Tango Schematics” and “Tango PCB”.

This module will contribute to the ongoing efforts of the Conrad Blucher Institute to make use of the Coastal Oil-Spill Simulation System in studying the effects of oil-spill remediation agents on simulated nearshore marine and estuarine ecosystems.
Procedure

The main objectives and background of the project were explained in the previous sections. In this section, the steps to implement this project and the detailed description of the design to implement this project are explained.

The following steps were taken before the design of the control module for the COSS equipment.
1. Interviewed Dr. James Bonner, director of the COSS project (and also the director for the Conrad Blucher Institute.), to determine what needs to be done with the current Oil-Water-Separator.
2. Visited the COSS site to know the layout of the equipment (pipe lines, reactors etc.,), and also to determine the requirements of the desired system.
3. Researched thoroughly the various characteristics of the PIC16C84 chip.
4. Studied the various equipment needed for the construction of the vacuum receiver.
5. Learned the process of designing and etching a circuit board.

Chip Description (hardware and software).

1. Architectural overview:

   The PIC16C84 uses a Harvard architecture. This architecture has the program and data accessed from separate memories. So the device has a program memory bus and a data memory bus. Separating program and data memory allows instructions to be sized differently than the 8-bit wide data word. PIC16C84 opcodes are 14-bits wide, enabling single word instructions. The full 14-bit wide program memory bus fetches a 14-bit instruction in a single cycle. A two stage pipeline overlaps fetch and execution of instructions. Consequently, all instructions execute in a single cycle except for program branches.

   The PIC16C84 addresses 1K x 14 program memory. All program memory is internal. PIC16CXX devices can directly or indirectly address its register files or data memory. All special function registers including the program counter are mapped into the data memory. These chips have an orthogonal (symmetrical) instruction set that makes it possible to carry out any
operation on any register using any addressing mode. The symmetrical nature and lack of ‘special optimal situations’ make programming with the PIC16XX simple yet efficient.

PIC16C84 has 36 x 8 SRAM and 64 x 8 EEPROM data memory. PIC16CXX devices contain an 8-bit ALU and working register. The ALU is a general purpose arithmetic unit. It performs arithmetic and Boolean functions between data in the working register and any register file.

The ALU is 8-bits wide and capable of addition, subtraction, shift and logical operations.

2. Hardware:
To run the software necessary to write programs in PicBasic (the language used to program the PIC chip), to compile them into assembly language code and to load the program on to the PicStic microprocessor controller, the following hardware is required as a minimum:

- IBM PC or compatible computer,
- MS-DOS 3.3 or higher,
- 512K of RAM,
- 1MB hard drive space,
- 3.5 inch double density disk drive,
- Parallel printer port with a DB-25 connector.

PicStic Programmer:
The PicStic programmer plugs into the parallel printer port of a PC and is powered by a plug-in wall transformer. The PicStic is inserted into the programmer and the program file can be loaded from the computer to the PicStic.

3. Software:
PicBasic Compiler (PBC.EXE)
After the PicBasic program is saved as an ASCII text file, it can be compiled into an assembly-language file with the PicBasic Compiler (PBC.EXE) program.

Assembler (PM.EXE)
If the PicBasic program is compiled successfully, then PBC.EXE will automatically start the assembler program (PM.EXE) to convert the compiled program into a machine-language file.

Downloader (EPIC.EXE)
The machine-language file created by the PM.EXE program can then be downloaded directly into the PicStic with the EPIC.EXE program. This program will also allow for erasing a program already in the PicStic.

4. Installing the software:
   The PicStic System Diskette contains all the software necessary to compile PicBasic code from ASCII text files and to assemble PicStic assembly language programs. The diskette also includes the EPIC programming software, which is used to download a machine-language file into the PicStic. Installing the PicStic software involves copying all files from the System Diskette into a sub-directory named 'pbc'.

5. Attaching the PicStic programmer:
   The PicStic programmer attaches to the parallel printer port of the PC. One end of the supplied J-11 cable must be attached to the DB-25 to RJ-11 adapter. This adapter is plugged into the DB-25 parallel printer port on the computer.
   
   The other end of the programming cable is attached to the RJ-11 socket on the programmer. The programmer was setup accordingly.

6. Power Supply:
   The programmer power supply can be plugged into any live wall outlet. The other end of the power cable must be plugged into the PicStic programmer.

7. Attaching the PicStic:
   The PicStic should not be inserted or removed from the socket on the programmer unless
   1. the PC is turned on,
   2. the programmer is powered up,
   3. the EPIC software is running and
   4. the programming LED is NOT lit.

   Unless all four of these conditions are met, it is not safe to insert or remove a PicStic from the programmer. All of the above precautions were taken during the development and testing of the program.
8. PicStic pin descriptions:

A list of the 16 physical pins on the PicStic and the functions they perform are given in Appendix – B. A summary of the some of the most used pins is given below.

<table>
<thead>
<tr>
<th>Physical Pin#</th>
<th>Functional Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>V+</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td>3</td>
<td>5VDC</td>
</tr>
<tr>
<td>4</td>
<td>RESET</td>
</tr>
<tr>
<td>7</td>
<td>PB0 – PB7</td>
</tr>
</tbody>
</table>

A diagram of the view from the component side of the chip is shown in figure 2.

Figure 2: View from the component side
9. Variables available for the programmer:

There are 22 bytes (0-21) available in memory for user defined variables. This memory can be used a 22 individual bytes or 11 two-byte words (0-10). In addition, the first two bytes can be read and written to as their individual 16 bits. The bits can have values of 0 or 1, the bytes have values from 0 to 255 and the words from 0 to 65535.

When the PicStic is powered up or reset, all the variables are initialized with a value of 0. These variables are given fixed names in PicBasic. The variables can be referred to by their PicBasic names or given other names by use of the Symbol command.

Sending Data to a Personal Computer
Attaching to a PC, direct connection:

The excellent I/O specifications of the PicStic make it possible for it to send data directly to the serial ports of a personal computer. Any terminal-communications program or a program in QBasic or some other language can be used to receive the data from the computer's serial port where the PicStic has sent it. The baud rate of both the sending and receiving programs must match. For these types of setups, it is usually best to use the TTL inverted-baud-rate modes like N2400 and N9600 with the SEROUT command.

Details of attaching the PicStic to a PC with diagrams are given in Appendix – C.

Real Time Clock

The PicStic 2 has a Real Time Clock/Calendar (RTC) that can be set and read directly from PicBasic. The set-and-read functions temporarily use the byte variables B15 through B21 for storing the following information:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Contents</th>
<th># Range</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>B15</td>
<td>year</td>
<td>00 – 99</td>
<td>use last two digits of year</td>
</tr>
<tr>
<td>B16</td>
<td>month</td>
<td>01 – 12</td>
<td>January = 1</td>
</tr>
<tr>
<td>B17</td>
<td>date</td>
<td>01 – 31</td>
<td></td>
</tr>
<tr>
<td>B18</td>
<td>day of week</td>
<td>01 – 07</td>
<td>Sunday = 1</td>
</tr>
<tr>
<td>B19</td>
<td>hours</td>
<td>00 – 23</td>
<td>24 hour clock, no AM or PM</td>
</tr>
<tr>
<td>B20</td>
<td>minutes</td>
<td>00 – 59</td>
<td></td>
</tr>
<tr>
<td>B21</td>
<td>seconds</td>
<td>00 – 59</td>
<td></td>
</tr>
</tbody>
</table>
The Real Time Clock actually receives and returns its information in hexadecimal format. This implies that we need to convert the decimal numbers we normally use into hexadecimal for use with the Real Time Clock.

**The Control Module**

The aim of the control module is to control the level of the oil-water-mixture level while making sure to give enough cycle time for (to turn on and off) the vacuum and drain pumps. To achieve this, three float switches are used, and the chip is programmed in such a way that the vacuum pump will turn off when the oil-water-mixture level crosses the third (upper most) switch but doesn’t turn back on until the water level falls below the second (middle) float switch.

Similarly, the drain pump will be started only when the oil-water-mixture level rises above the second float switch and doesn’t shut down until the level falls below the first (lower most) float switch. This scheme makes the function of the control module independent of the flow rates of the fluid entering and leaving the pumps.

Boolean expressions were generated and simplified using Quine-McClusky’s method to implement the above decisions. All the work related to developing boolean expressions is detailed in Appendix – A. When the program was developed using these simplified boolean expressions the results were found to be erroneous.

Reason for the failure of the above expressions:

The generated boolean expressions for the above decisions on further analysis showed that the result would be erroneous as the decisions for starting and shutting down the motors differ depending on whether the oil-water-mixture level is rising or falling. When these mutually contradicting conditions were fed to the boolean expression generator, the resulting expression was erroneous.

This problem was overcome by first setting up a flag which would indicate whether the liquid level is rising or falling and then use it as a variable in the boolean expression along with the new float state variables.

This resulted in a robust control mechanism which can take care of the condition where a float switch will keep switching on and off until it settles to a stable condition. In that case no harm
would be done to the vacuum and drain pumps as the control scheme takes different decisions depending on whether the liquid level rises or falls.

After determining this flag, taking a decision based on the flag and the three variables (new states of the floats) wouldn’t be a complex task.

The use of boolean expressions was eliminated due to the limited instruction set of the PicBasic and several other problems which are detailed in Appendix – A. The decisions were made using simple ‘if … then’ statements available in PicBasic.

**Error Conditions**

There are a number of cases wherein the states of the float switches would not result in meaningful decision that can be made by the chip. For example, if one or more of the float switches begin to malfunction then the chip should be able to handle the case appropriately.

When one (or more) of the switches malfunction, the chip can be made to detect it by testing whether one of the higher float switch is on while any of the lowers ones is off. In these cases the chip is made to signal an error message via the serial port.

Even after accounting for the above conditions there exist two conditions where the chip will not be able to detect the malfunctioning of the system. These conditions are

1. when upper most switch gets disabled in the ‘OFF’ state.
2. when the lower most switch get disabled in the ‘ON’ state.

In case of the first condition, there is vacuum receiver attached to the oil-water separator which can shut down the vacuum pump when the fluid level in the oil-water separator rises above a predetermined level. In case of the second condition, the drain pump can be shutdown manually. The drain pump pumps air when the fluid level in the tank falls below the lowest float switch which does not affect the filtration process and also the function of the drain pump.

**Control Program**

A program in PicBasic was written to perform the tasks mentioned above. The program is written such that the chip can take care of the control of the vacuum receiver irrespective of the initial state of the vacuum receiver i.e., level of oil-water mixture and the states of the float switches. The program initially reads the switches and makes the decisions to turn on and off the
two pumps using a minimum logic which doesn’t require a flag to indicate whether the liquid level is rising or falling (it sends the decisions with the time stamp even in this case). Once this initial decision is taken the, program enters an infinite loop wherein it reads the float switches, compares it with the old values, sets the flag (for rising or falling) and then makes the decisions to turn on and off the pumps as discussed earlier. A delay time of 1 second was introduced while reading the float switches which will allow the float switches to settle to a stable state.

Since a cycle time of 20 seconds is needed, the distance required between the float switches which will allow for a minimum of 20 seconds cycle time needs to be done. The calculations are shown below:

The maximum rate at which the drain pump can pump the oil-water mixture is 600gallons/min. The dimensions of the tank are 15’1/4”X5’X5’9”. The minimum distance between any two float switches is found by

\[
H = \frac{600 \text{gallon/min} \times 3.78 \text{liters/gallon} \times 1000 \text{cm/lit} \times 20 \text{seconds}}{60 \text{seconds} \times 15.0208 \text{ft} \times 5 \text{ft} \times (12 \text{inches/ft} \times 2.54 \text{cm/inch})^2}
\]

Thus, a distance of 11 cm needs to be maintained between the float switches. Enough height is available for the float switches to be placed at a distance more than 30 cm (1 ft).

The control program was compiled using the PBC compiler described earlier. The compiled code should not occupy more than 1K of memory as otherwise the program cannot be loaded on to the chip. The control program was written considering the above constraint. The final program size came out to be 904 bytes. The compiler gives an error whenever a program exceeds 1K limit.

The program is loaded on to the chip using the EPIC loader and the PicStic programmer, both of which have been described earlier.

**Program Design**

There is one main routine, which, with the help of subroutines, provides the control functionality of the chip. The design of the various routines is described below.
Figure 3: Top level Control Diagram.
Start

Read pins

Set flag. Make initial decision. Set pins

Send info to serial port

Read pins

If states changed

Yes

Set Flag

Send error signal if any

Make decision

Set pins

Send info

Wait

No

Figure 4: Flow chart for the control program.
Main Program body

Input: The routine reads the states of the three input pins that are set aside for input. The states of these pins are set by the three float switches connected to them. Additionally, it may also read serial information from one of its pins to set the the real-time clock which will be used for sending time information.

Function: This routine runs on the chip to read the three pins (float switches) and turn on/off the output pins which will be used to operate the vacuum pump and the drain pump. It also sends the float switch state information to the serial port. If the memory on the chip allows, it can also read serial data from one of its pins to set the real-time clock based on the input instead of a predetermined initial value.

Output: Sets the output pins to on/off to operate the vacuum and the drain pumps while sending the float switch state information to a serial port (to be displayed on a PC).

Send_info – subroutine

Input: This routine reads the data from predetermined registers which contain the time information and the state information for the input and the output pins.

Function: This subroutine does the actual job of converting the time from hexadecimal format to decimal format, and writing the time stamp and state information for the pins to the serial port.

Output: Sends ASCII characters to the serial output pins, which convey the state information of the pins along with the time stamp. Which can be displayed by any terminal emulation
program like ‘procomm’.

Get_info – subroutine

Input: This routine reads ASCII characters from the serial input pin, which specify the time to which the real-time clock should be set, relative to which the time is calculated to send the time information to the serial port. Input can be given from the any terminal communications program (i.e., ‘procomm’).

Function: This subroutine prompts the user for the time to which the real-time clock needs to be initialized.

Output: Stores the time information in the predetermined registers (these registers were discussed in Real – Time clock).

clockset – subroutine

Input: Time information in predetermined registers.

Function: This is the assembly language subroutine that sets the real-time clock on the PIC chip to the time specified in the predetermined registers.

Output: Sets the clock to the time specified in predetermined registers.

clockget – subroutine

Input: This routine reads real-time-clock on the chip and stores it in predetermined registers.
Function: This is the assembly language subroutine that reads from the real-time clock on the chip and writes the time information to the predetermined registers.

Output: Time information in the predetermined registers.

Vacuum_on – subroutine

Input: This routine is called by the main routine to set the predetermined pin associated with the vacuum-pump to a value of binary one (high).

Function: Sets the output pin corresponding to the vacuum pump to a value of binary one (high).

Output: Output pin for vacuum pump is set to binary one (high).

Vacuum_off – subroutine

Input: This routine is called by the main routine to set the pin associated with the vacuum-pump to a value of binary zero (low).

Function: Sets the output pin corresponding to the vacuum pump to a value of binary zero (low).

Output: Output pin for vacuum pump is set to binary zero (low).

Drain_on – subroutine

Input: This routine is called by the main routine to set the pin associated with the drain-pump to a value of binary one (high).
Function: Sets the output pin corresponding to the drain pump to binary one (high).

Output: Output pin for drain pump is set to binary one (high).

Drain_off – subroutine

Input: This routine is called by the main routine to set the predetermined pin associated with drain-pump to a value of binary zero (low).

Function: Sets the output pin corresponding to the drain pump to binary zero (low).

Output: Output pin for drain pump is set to binary zero (low).

Set_Current

Input: This routine is called by the main routine to set the flag required to indicate whether liquid level is rising or falling. It reads the float switch states stored in one of the byte registers.

Functions: This sets a flag which is used in determining whether the liquid level is rising or falling.

Output: Output of this routine is stored in a predetermined register

The Set_Current subroutine was introduced to reduce the size of the compiled code. This routine also sends an error signal on the serial port whenever the float switches settle in unexpected states (as mentioned in the Error Conditions section). The addition of code to handle this has increased the compiled size code (the compiler gives an error whenever the 1K limit is reached).
to which the subroutine to facilitate user input (to set the desired reference time) had to be
sacrificed. It does not affect the functioning of the control module as the getInfo subroutine gets the
user input to set the time in the real time clock of the chip (relative to which the time stamps are
sent to the terminal). The present program sends the time stamp relative to a fixed time.

Circuits and Equipment

Using the chip to start the 15HP 3-phase motors requires interfacing circuitry. The motors will
be operated using motor starters that will be controlled by solid-state relays operated by the I/O
pins of the chip. Photo isolation is desired in the solid-state switches to avoid damage to the chip.

The various equipment needed to implement the project were identified after an extensive
search of the various published catalogs and also the world wide web. Details of various equipment
identified are given in Appendix – E.

Designing the Interfacing circuitry

The PIC chip can be operated on one of the two input voltages (5V and 9V). The 9V input is
used when the chip drives external circuits. In this project the chip needs to operate external circuits
(float switches and the output signals for the pump operation and the serial port communication),
so the 9V input is chosen. When the 9V input is used, power can be drawn out of the 5V pin to
operate the float switch terminals. These three float switch terminals need pull down resistors. The
output pins of the chip (meant to operate the pumps) are connected to terminals via solid state-
switches (Type 3-15DC/240AC). The two pins meant for serial output and serial input are
connected to their respective terminals via resistors to limit the current.

Two LEDs have been introduced into the circuit to allow for the purpose of testing and also to
allow for the detection of errors in the functioning of the chip.

A detailed diagram of the circuit is shown in figure 5. A detailed diagram can be found in
Appendix – D.
Figure 5: Interfacing circuitry for the chip.

R1, R2, R3, R6: 1KΩ
R4, R5 : 330 KΩ
R7 : 22 KΩ
Designing and Etching of the PCB

As it is intended that the control module should have a simple interface with the other equipment to make the connections straightforward, it is planned to etch out a circuit board (size 6”X3”). Designing a PCP involved finalizing the circuitry and drawing it using a software called ‘Tango Schematics’ which is a tool for generating drawings for the PCBs. The copper layout was generated with the help of another program called ‘Tango PCB’ which helps in drawing the copper layouts for the circuit. All the other details of PCB design and etching are given in Appendix – D.

Control Program Testing

After etching out the PCB and assembling the circuitry, the chip was programmed and tested by manually altering the switches. Since the design of the control program does not depend on the flow rates, this test was sufficient to check all the features of the control program. A listing of all the signals that were sent to a PC (from the chip) for a test run is given below.

Notations:

low, mid, up: float switches
v, d: vacuum and drain pumps
0 : off
1 : on

Output of the test run:

Time(h:m:s) - 3:39:23 States(low,mid,up) - 0:0:0 (v,d) - 1:0
Time(h:m:s) - 3:39:49 States(low,mid,up) - 1:0:0 (v,d) - 1:0
Time(h:m:s) - 3:39:55 States(low,mid,up) - 1:1:0 (v,d) - 1:1
Time(h:m:s) - 3:39:66 States(low,mid,up) - 1:1:1 (v,d) - 0:1
*Err* Time(h:m:s) - 3:39:67 States(low,mid,up) - 1:0:1 (v,d) - 0:1
Time(h:m:s) - 3:39:68 States(low,mid,up) - 1:1:1 (v,d) - 0:1
Time(h:m:s) - 3:39:70 States(low,mid,up) - 1:1:0 (v,d) - 0:1
Time(h:m:s) - 3:39:80 States(low,mid,up) - 1:0:0 (v,d) - 1:1
Time(h:m:s) - 3:39:83 States(low,mid,up) - 0:0:0 (v,d) - 1:0
Time(h:m:s) - 3:39:84 States(low,mid,up) - 1:0:0 (v,d) - 1:0
Time(h:m:s) - 3:39:89 States(low,mid,up) - 0:0:0 (v,d) - 1:0
Time(h:m:s) - 3:40:7 States(low,mid,up) - 1:0:0 (v,d) - 1:0
As it is evident from reading the float switch values and the output values from the chip (on/off values for the vacuum and drain pumps), the chip operates the two pumps properly. It only turns off the vacuum pump when the fluid level has crossed the uppermost switch and doesn’t turn it back on until it falls below the second switch. Similarly the drain pump is turned on only when the fluid level rises above the second switch and is not turned off until the level falls below the lowermost switch. The minimum cycle time occurs when only the drain pump is running as the vacuum pump is operated at 10 – 20 atmospheres (which results in a lesser flow rate of the fluid than that caused by the drain pump). Thus, the minimum distance required between the float switches was calculated considering only the drain pump.

Whenever an error condition arises (on of the error conditions that can be detected by the control program as previously discussed under the section ‘Error Conditions’) the chip signals it on the serial port (which can be seen as ‘*Err*’ on the PC).

Some other error conditions that the chip cannot detect are sudden state transitions of the float switches without going through the intermediate states (both the states are valid while the sudden transition is not expected).

In all the cases of undetected error conditions, the chip makes decisions based on the upper most switch which is on. This has no ill – effects on the functioning of the vacuum receiver and also the functioning of the pumps.
Results

The end result of this project is a compact, robust, easy-to-use and easy-to-install control module that will provide the functionality for the intended vacuum receiver at the Coastal Oil-Spill Simulation System.

Since the control module along with the necessary equipment has not been installed and since the only inputs to the chip are the states of the three float switches, the control module was tested by manually simulating the float switch states. Testing of the control program was discussed in the previous section.

Further improvements can be made to the control module, some of which are given below.

1. Better error handling capability can be added by using later versions of the PIC chip which have larger memory.
2. Some electrical arrangement that ensures the proper functioning of the float switches can be made.

Future work:

Once the interface between the chip and the PC is finalized, software could be written to gather the information from the PIC chip, store it in a database, and analyze it to find the optimum operating conditions for the filtration equipment. The COSS project is to develop a filtration system that could be deployed at the base of a ship to recover oil during an oil spill to avoid environmental hazards involved therein. Therefore, the control-module, along with the software would greatly help the researchers at the COSS in developing such a system.
References

Bartee, Thomas C. *Computer Architecture and Logic Design.*


Jameco Electronics, *Jameco Electrical Components & Computer Products:*
Catalog no. 991 (Feb - Apr 99). Jameco Electronics, 1355 Shoreway Road,
Belmont, CA - 94002 - 4100.

Technology Services Institute, *PicBasic for the PicStic – Manual Version 2.0,*

Jameco Electronics: Home page.
Updated: 05/18/99.
Cited: 05/19/99.

Microchip Technology Inc: Home page.
Updated: 03/25/99.
Cited: 05/19/99.

Appendix - A

Generating Boolean Expressions

The researchers who will be using this new system have specified the required operating conditions for the vacuum pump and the drain pump (with regard to the oil-water mixture level). A detailed analysis of the requirements yields the following table of decisions to be taken. In the table, A, B, C are the states of the float switches (two sets, one for the previous cycle values i.e., before the float switch states changed, and one for the current cycle values i.e., after the float switch states changed). A value of ‘0’ indicates “OFF” condition and a value of ‘1’ indicates “ON” condition of the float switches.

<table>
<thead>
<tr>
<th>Previous</th>
<th>Current</th>
<th>Drain</th>
<th>Vacuum</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>Pump</td>
<td>Pump</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 0 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 0 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Control decisions for the operation of the vacuum receiver.

It was planned to generate boolean expressions for they can reduce the program code to a large extent eliminating the requirement of numerous ‘if-then’ statements. Boolean expressions were generated and simplified using Quine-McClusky's method to implement the above the decisions (a ‘C’ program was written to generate simplified boolean expression using Quine-McClusky’s method). When the program was developed using these simplified boolean expressions the results were found to be erroneous.
The boolean expression generated for vacuum and drain pumps are:
\[ V = a \cdot c' \cdot d \cdot f' + b' \cdot c' \cdot d' \cdot f' \]
\[ D = a \cdot c' \cdot d \cdot f' + a \cdot b \cdot d \cdot e \]

Reason for the failure of the above expressions:

The generated boolean expressions for the above decisions on further analysis showed that the result would be erroneous as the decisions for starting and shutting down the motors differ depending on whether the oil-water-mixture level is rising or falling. When these mutually contradicting conditions were fed to the boolean expression generator, the resulting expression was erroneous.

This problem was overcome by first setting up a flag which would indicate whether the liquid level is rising or falling and then use it as a variable in the boolean expression along with the new float state variables.

The following illustrates decisions using a flag bit which indicates rising or falling of the oil-water level.

Notations:
\[ t = 0 \Rightarrow \text{oil-water level is falling} \]
\[ t = 1 \Rightarrow \text{oil-water level is rising} \]
\[ t_{prev} < t_{cur} \Rightarrow 1 \]
\[ t_{prev} > t_{cur} \Rightarrow 0 \]
a, b, c --- float switch states (1 for 'ON' and 0 for 'OFF')
V, D --- values for the vacuum and drain pumps (1 for 'ON' and 0 for 'OFF')
# --- decimal representation of the terms in the boolean expression.
<table>
<thead>
<tr>
<th>t</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>V</th>
<th>D</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Simplified expression for Vacuum is
\[ v = f(t, a, b, c) = t \cdot a' + b' \cdot c' \]

Simplified expression for Drain is
\[ d = f(t, a, b, c) = t' \cdot a' + a \cdot b' \cdot c' + t \cdot a \cdot b \]

There were two problems in using the boolean expressions.

1. The PicBasic doesn’t support bit-wise operation on individual bits. The inputs from the pins have to be stored in 8-bit registers as integers and then manipulated. This increases the program size.

2. Though the user manual for the PicBasic contains ‘nand’ operator to compliment variables, the compiler doesn’t support it. This resulted in that the variables have to be complimented using ‘if-then’ statements which produced bulky code.

3. The PicBasic does not provide for the overriding of the operator precedence, which again leads to lengthy code.

The use of boolean expressions would have reduced the number of lines of code in spite of the above restriction if there were more than 5 variables, but in the present case the number of variables got reduced to 4 (and one of them – the ‘t’ flag needs to be set by using a number of if statements as it indicates the rising or falling of the fluid level). For these reasons and also to make the control program easily understandable, the use of boolean expressions was abandoned.
The 'C' program used to generate the boolean expressions is given below:

Programmer: Ravikanth Bairi

This program accepts a boolean expression of the form:

\[ f(x,y,z,\ldots) = \text{sum}(0,1,2,3,\ldots); \]

with a maximum of 10 literals and simplifies it using the QuineMcCluskey's method.

Scheme: This program utilizes 2 arrays of structures which act like two tables, matches the rows in the first table (active table) and copies the matched ones into the other (inactive table) and copies the unmatched ones to the answer buffer. The inactive table then becomes active and the process is repeated till the # of rows in the active table becomes zero.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define NOTSPECIAL 0 /*flag indicating the expression is not a special case*/
#define SPECIAL_0 1 /*indicates expression is a special case and answer=0*/
#define SPECIAL_1 2 /*indicates Exp is special case and answer = 1 */

/*define a structure to hold the row of a table used in QuineMcCluskey method*/
struct TableRow {
    int literal[10]; /*to hold the longest possible minterm. Contains just 1's or 0's (and -1 to indicate nothing)*/
    int matchflag; /*flag to indicate whether the row has matched 0 for unmatched*/
};
```

32
void getinput(char *funcChar, char *charArray, int *intArray, int *maxliterals,
    int *errFlag , int *endFlag, int *allowed_max);

void getNumber( int *num, int *err, int max );

void GetBinary( int decnum, int* array, int maxliterals);

void printFunc( char funcChar, char *charArray, int maxliterals, int special);

void QuineMcCluskey( int maxliterals, int *maxrows);

int Compare(int active, int maxliterals, int index1, int index2, int *place);

/*variables to hold the max # of rows, answer and answer buffer length
and 'p' which act as an index for the input string*/

int maxrows[2], ans_buff[10240], buff_len, p;

/* declare two arrays of rows which will act as two tables*/
struct TableRow row[2][7640];
char ch[5150]; /* array to hold the input string*/

main()
{
    char funcChar, charArray[10], c;
    /* intArr[] to hold the user entered integers, maxliterals to hold the
    # of literals entered, maxints to hold the maximum possible # of
    integers (minterms) that the user can enter for a given # of literals*/
    int i=0, intArr[1024], maxliterals = 0,
errFlag = 0, endFlag=0, maxints;

printf("Enter the boolean expression to be simplified(with no more\n\n");
printf("than 10 literals) in the following format :\n\n");
printf("f(a,b,c)=sum(0,1,2,3);\n\n");
printf("Terminate the expressions with a semicolon(;) if you have\n\n");
printf("more expressions to enter, otherwise terminate with a period(.)\n\n");
printf("Termination character should be followed by a carriage return\n\n");

/*@while endFlag is zero, get input, process and print result */
while( endFlag==0 ) {
    printf("Type in the expression :\n\n");
    getInput(&funcChar, charArray, intArr, &maxliterals, &errFlag,
            &endFlag,&maxints);
    /*getInput(..) stores the input string in ch[] array */
    /* if no error occured in getting the input */
    if(errFlag==0) {
        if(maxrows[0]== 0){/* special case: 0 minterms !!*/
            printfFunc(funcChar,charArray, maxliterals, SPECIAL_0);
        }
        else if(maxrows[0]==(maxints+1)){/* all possible minterms!!*/
            printfFunc(funcChar,charArray, maxliterals,SPECIAL_1);
        }
        else {
            for(i = 0; i< maxrows[0]; i++) {
                /*fill each row with the binary equivalent of minterm*/
                GetBinary(intArr[i], row[0][i].literal, maxliterals);
            }
            /*run QMQ method and print the answer. Answer is stored
             in ans_buff[] by the QMQ method */
            QuineMcCluskey( maxliterals,maxrows);
    }
}

34
printFunc(funcChar,charArray, maxliterals, NOTSPECIAL);
}

else { /* if error occured */
    printf("Expression not in the required format.\n");
    printf("err code : %d\n", errFlag);
    while(ch[p]!='$' && ch[p]!=';') { p++;} /* skip to last character
        in the input string*/
    if(ch[p]=='.' ) { endFlag=1;break;} /* set the end flag accordingly*/
}
}
printf("Thank you. Have a nice day !!\n\n");
exit();

*************************************************************************/

getInput function which the input till a '; or a '.'
and stores the information in variables and arrays shown below:

    funcChar<-- f (a, b, c, d, e, f) = sum (1, 2, 3, 4, 5, ...);
                 ------------------- -------------------
                 charArray[10]     intArray[1024]

maxliterals - # of literals the user has entered
allowed_max - Max possible value of integer the user could have entered
for a given number of variables(literals)
errFlag - Tells whether error has occurred (in input)
endFlag - Tells whether the user pressed '.' at the end
*************************************************************************/

void getInput(char *funcChar, char *charArray, int *intArray, int *maxliterals,
int *errFlag, int *endFlag, int *allowed_max)
{
    char tempstr[5], c;
    int i, num, numErr = 0; /* numErr to indicate error in gettin a number */

    /* initialize variables to appropriate values */
    *maxliterals = 0; *allowed_max = 1; maxrows[0] = 0; p = 0;  /****p****/
    *errFlag = 0;

    /* fill in the ch[] array with the input string */
    while (p < 5149) {
        c = getc(stdin);
        if (!isspace(c)) { ch[p] = c; p++; } /*ignore white space*/
        if (c == ';' || c == '.') break;
    }
    ch[p] = '\0';  /*terminate the string*/

    if (ch[p - 1] != '.' && ch[p - 1] != ';') {
        puts("Invalid input. Fatal error !!");
        puts("You may have to re-run the program");
        *errFlag = 31; ch[p - 1] = '.'; return;
    }

    /*
     Start parsing from the beginning of the string .
     Scheme : Set the error flag and return at the
               first error encountered. Do not consider
               white space as it is ignored.
     */


```c
p = 0;
if(isalpha(ch[p])) {*funcChar = ch[p];} /*store the function char*/
else {*errFlag = 1; return;}
p++; if(ch[p] != '(') { *errFlag = 2; return; }
else { p++; }

if(isalpha(ch[p])) {
    /* its a variable; store in charArray */
    charArray[*maxliterals] = ch[p]; *maxliterals += 1;
}
else if(ch[p] == ')') { puts("Zero literals"); *errFlag = 21; return; }
else {*errFlag = 3; return; }
while (*maxliterals < 11) {
p++; if(ch[p] != ',' && ch[p] != ')') { *errFlag = 4; return; }
if(ch[p] == ')') { break; }
if(ch[p] == ',') {
p++;
if(*maxliterals > 9) {
    *errFlag = 40; puts("Too many literals");
    return;
}
if(isalpha(ch[p])) {
    for(i=0;i<*maxliterals;i++) {
        if(charArray[i] == ch[p]) {
            puts("literal repeated"); *errFlag = 41; return;
        }
    }
    charArray[*maxliterals] = ch[p]; *maxliterals += 1;
}
```
else {*errFlag = 5; return;}
}
}

/*start getting the right hand side of the expression*/
p++; 
if ( ch[p] != '=' ) { *errFlag = 6; return; }
else { p++; }

for(i = 0; i<3 ; i++) { tempstr[i] = tolower(ch[p]); p++; }
tempstr[3] = '\0';
if(strcmp(tempstr, "sum")!= 0 ) { *errFlag = 7; return; }

if(ch[p] != '(') { *errFlag = 8; return; }
else { p++; }

for(i = 1; i<=*maxliterals; i++) { *allowed_max *= 2; } /*find max*/
*allowed_max -= 1;

getNumber( &num, &numErr, *allowed_max );
if( numErr == 0 ) { intArray[maxrows[0]] = num; maxrows[0]++; }
else { *errFlag = numErr; return; }

while( maxrows[0] <= *allowed_max ) {
    if( ch[p] != ',' & & ch[p] != ')'){*errFlag = 10; return; }
    else if( ch[p] == ')' ) { break; }
    else if( ch[p] == ',' ) { p++; }
    getNumber( &num, &numErr, *allowed_max );
    if( numErr == 0 ) {
        for(i=0;i<maxrows[0];i++) {
            if(intArray[i]==num) {
                numErr = 0; 
                break; 
            }
        }
    }
}
puts("Number repeated"); *errFlag = 31; return;
  }
}
intArray[maxrows[0]] = num; maxrows[0]++;
}
else { *errFlag = numErr; return; }
}
p++;

/* analyse the last character to set the endFlag */
if( ch[p] != '.' && ch[p] != '!' ) { *errFlag = 12; return; }
else if( ch[p] == '!' ) { *endFlag = 1; }
return;
}

 **************************************************************
define a helper function to retrieve an integer passing it the maximum
value(max) the user can enter for an integer, variable(num) to store
the retrieved integer and another variable to convey error message (err)
***************************************************************/

void getNumber(int *num, int *err, int max )
{
  char tempstr[5];
  int i=0;
  if( isdigit(ch[p]) ) {
    /* grab the digits while checking for errors */
    do {
      if(i>3) { *err=201; return; }
      tempstr[i] = ch[p] ; p++;i++;
      }
  }
  num[0] = tempstr[0];
  num[1] = tempstr[1];
  num[2] = tempstr[2];
  num[3] = tempstr[3];
  num[4] = tempstr[4];
  return;
}
while(isdigit(ch[p]));

tempstr[i] = '0';
}
else { *err = 202 ; return ; } /**<error if first char is not digit*/
*num = atoi(tempstr);
if(*num > max) { *err = 203; return ; }

/**
 * QuineMcGluskey funtions receives the number of variables entered
 * and the number of rows in the first table. I has access to the table's
 * rows and the answer buffer(ans_buf) which it has to populate with the
 * simplified expression(in binary form). It assumes that the first table
 * is active to begin with.
 */

void QuineMcGluskey ( int maxliterals,int *maxrows)
{
    /*
     * set active to zero as data is stored in the first table to begin
     * simplification.
     * iterations : holds the # of tables fully processed
     * difference : # of literals any two rows differ in
     * i,j,k : convenience variables
     */
    int iterations=0, i, j, k, place,difference, active=0, inactive=1 ;
    buff_len = 0;
    for(i=0; i<maxrows[active]; i++){ row[active][i].matchflag = 0; }
    while(1) {
        iterations++;
        if(maxrows[active]<1) break;
maxrows[inactive]=0;
for(i=0; i<maxrows[active]; i++) {
  for(j=i+1; j<maxrows[active]; j++) {
    /* find in how many literals these 2 rows differ */
    difference = Compare(active, maxliterals, i, j, &place);
    if( difference == 1 )
      {
        /* copy the term to 2nd struct-array and inc the maxrows 
           set the matchflag of 2nd struct-array element to zero */
        row[inactive][maxrows[inactive]] = row[active][j] ;
        row[inactive][maxrows[inactive]].literal[place] = -1;
        row[inactive][maxrows[inactive]].matchflag = 0;

        /* set the matchflag(actives) to the iteration value */
        row[active][i].matchflag = iterations;
        row[active][j].matchflag = iterations;
        maxrows[inactive]++;
      }

    /* if duplicate, mark its matchflag with -2 to discard */
    if(difference==0) { row[active][j].matchflag = -2 ; }
  } /* end of j for */
} /* end of i for */

/* copy the unmatched ones to ansbuf-1(those w/-2<matchflag<iterations)*/
for(k=0; k<maxrows[active]; k++) {
  if(row[active][k].matchflag<iterations &&
    row[active][k].matchflag>-2)
    {
    for(i=0; i<maxliterals; i++)
      {
ans_buff[buf_len] = row[active][k].literal[i];
buf_len++;
}
}

/*switch the active and inactive*/
k = active; active = inactive; inactive = k;

}/*end of while(1)*/
}/*end of function*/

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

Compare function compares two rows(index1, index2) of a given active table
and tells in how many literals they differ and places the index where
they differ in the variable 'place'. If they differ in more than one place
'place' has no meaning. It needs the # of boolean variables entered
(maxliterals).
******************************************************************************

int Compare(int active, int maxliterals, int index1, int index2, int *place)
{
    int i, count = 0;

    for(i = 0; i < maxliterals; i++){
        if(row[active][index1].literal[i] != row[active][index2].literal[i]){
            count++; *place = i;
        }
    }
    return count;
}
printFunc(.) function prints out the simplified expression.
It needs to know:
funcChar: character used to represent the function.
charArray[]: character set used to represent variables.
maxliterals: # of variables entered.
special: integer to convey whether the expression is a special case
It can access the simplified expression stored in the ans_buf[] array
(and buff_len) which are declared global.

void printFunc( char funcChar, char *charArray, int maxliterals, int special)
{
    int pointer = 0, i, j, length = 5;

    printf("nSimplified Expression :");
    printf("n%c( %c", funcChar, charArray[0]);
    for(i = 1; i<maxliterals; i++) {printf(" ,%c", charArray[i]); length += 3;}
    printf(" ) = "); length +=5 ;
    if( special == NOTSPECIAL ) {
        while( pointer < buff_len ) {
            for( i = 0; i<maxliterals; i++) {
                if( ans_buf[pointer] == 1 ) {
                    printf("%c ", charArray[i]); length += 2;
                }
                else if( ans_buf[pointer] == 0 ){
                    printf("%c\ ", charArray[i]); length += 3;
                }
                pointer++;
            }
            if((pointer+1) < buff_len) { printf(" + "); }
        }
    }
}
else { printf("\n"); }
if(length >=70) { printf("\n "); length = 0; }
}
}
else if(special == SPECIAL_0 ) { printf(" 0 \n"); }
else if(special == SPECIAL_1 ) { printf(" 1 \n"); }
}

getBinary(..) function takes a decimal number (decnum), converts it into
binary form and stores in the array that it receives. It also needs to
know the number of variables for the function (maxliterals).

getBinary( int decnum, int* array, int maxliterals)
{
    int i, temp;

    for(i= maxliterals-1; i>=0; i--){
        array[i] = decnum%2;
        decnum = (int) (decnum/2);
    }
}
Appendix – B
PIC chip pin descriptions

A list of the 16 physical pins on the PicStic and the functions they perform are as follow:

<table>
<thead>
<tr>
<th>Physical Pin#</th>
<th>Functional Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>V+</td>
<td>Positive voltage input for the PicStic (7-18 VDC)</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>Common ground for the PicStic power supply and all I/O pins.</td>
</tr>
<tr>
<td>3</td>
<td>PA3</td>
<td>Additional digital I/O pin that can be used by means of a short assembly-language subroutine or by using the PEEK or POKE.</td>
</tr>
<tr>
<td>4</td>
<td>PA4</td>
<td>Additional digital I/O pin that can be used by means of a short assembly language subroutine or by using the PEEK or POKE.</td>
</tr>
<tr>
<td>5</td>
<td>5VDC</td>
<td>Positive 5V DC output from the PicStic’s internal voltage regulator that can be used to power other parts of the external circuit. The current draw can be up to 100 mA. This can also be used as a 5 volt input to avoid using the internal regulator.</td>
</tr>
<tr>
<td>6</td>
<td>RESET</td>
<td>Bringing this pin to a logic low (0 volts) an releasing it will reset the PicStic and cause the program to start over from the beginning.</td>
</tr>
<tr>
<td>7</td>
<td>PB0</td>
<td>Digital I/O pin addressable in PicBasic as the number 0.</td>
</tr>
<tr>
<td>8</td>
<td>PB1</td>
<td>Digital I/O pin addressable in PicBasic as the number 1.</td>
</tr>
<tr>
<td>9</td>
<td>PB2</td>
<td>Digital I/O pin addressable in PicBasic as the number 2.</td>
</tr>
<tr>
<td>10</td>
<td>PB3</td>
<td>Digital I/O pin addressable in PicBasic as the number 3.</td>
</tr>
<tr>
<td>11</td>
<td>PB4</td>
<td>Digital I/O pin addressable in PicBasic as the number 4.</td>
</tr>
<tr>
<td>12</td>
<td>PB5</td>
<td>Digital I/O pin addressable in PicBasic as the number 5.</td>
</tr>
<tr>
<td>13</td>
<td>PB6</td>
<td>Digital I/O pin addressable in PicBasic as the number 6.</td>
</tr>
<tr>
<td>14</td>
<td>PB7</td>
<td>Digital I/O pin addressable in PicBasic as the number 7.</td>
</tr>
<tr>
<td>13-2</td>
<td>ADC1</td>
<td>On the PicStic3, this is the 12-bit A/D converter input measuring positive (+) with respect to ground. In dual channel differential measurements, this is the negative (-) channel.</td>
</tr>
</tbody>
</table>
On the PicStic3, this is the 12-bit A/D converter input measuring positive (+) with respect to ground. In dual channel differential measurements, this is the positive (+) channel.
Appendix – C
PicStic – PC communication

The following describes how to attach the PicStic to a personal computer:

The grey arrows interconnecting the pins on the DB-25 port, shown in figure 6, eliminate the need for the hardware handshaking protocols that are required with some personal computers. Figure 7 shows the connections for a DB-9 port.

For DB-25

<table>
<thead>
<tr>
<th>Connect pins</th>
<th>DB-25</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 &amp; 5</td>
<td>pin #7 to ground</td>
</tr>
<tr>
<td>6, 8 &amp; 20</td>
<td>pin #3 to 1 KOhm to PicStic port</td>
</tr>
</tbody>
</table>

![Figure 6: DB-25 connections](image)

For DB-9

<table>
<thead>
<tr>
<th>Connect pins</th>
<th>DB-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 &amp; 8</td>
<td>pin #5 to ground</td>
</tr>
<tr>
<td>1, 4 &amp; 6</td>
<td>pin #2 to 1KΩ to PicStic port</td>
</tr>
</tbody>
</table>

![Figure 7: DB-9 connections](image)
Appendix – D
Design of PCB

Designing a PCP involved finalizing the circuitry and drawing it using a software called ‘Tango Schematics’ which is a tool for generating drawings for the PCBs. This is a complex and time consuming task as the software requires precise measurements of the various components used in the circuit (solid, state switches, resistors, terminals etc). The measurements were taken using vernier callipers. After drawing out all the connections between the components, a net-list (an internal representation of the circuit) is generated. This net-list is fed to another program called ‘Tango PCB’ which helps in drawing the copper layouts for the circuit. After finalizing the copper layout and other details (such as width of the copper lines and others), a printout of the PCB is taken. A photocopy of this printout is then taken on a slide that acts like a negative in the etching process of the PCB.

Etching process:

Polymer panels in standard sizes with copper coating (with a photosensitive organic layer on it) are available commercially for etching PCBs. One with the dimensions of 6”X3” is chosen for the present purpose. The slide with the line layout developed previously is placed on this panel and then can be exposed to sunlight for about 10 hours or can be exposed to a 150W bulb at a distance of 20 cm for about 15 to 30 minutes (the later method was followed) so that the exposed organic layer changes its composition so as to be washed away in chemical solution. After that, the panel is agitated in hot condition in a solution of Ferrous Chloride, which will dissolve the excess copper coating leaving only the layout of the copper connections.

All the required components like resistors, terminals, solid-state switches are then soldered onto the PCB. The PIC chip is then mounted on this board and tested for its functionality.

The various phases during the development of the control module are illustrated in the following figures.
Printout of the circuit diagram taken from the “Tango Schematics” software.
Printout of the PCB's copper linings.
Printout of the PCB with components mounted.
Printout of the PCB containing both the components and the copper connections.
Appendix – E

Equipment Specifications

After searching various sources (like Jameco Electronic Components etc.,) a low-cost, robust, solid-state relay has been identified. The product is from Potter & Brumfield co. It has the following specifications.

Control Voltage: 3 – 15VDC
Type : DC/AC
Contact Voltage: 240VAC
Contact Current: 3A
Pins : 4
Terminal Type : Solder Lead
Weight : 0.2 lbs.
Screw terminal with hardware.
Photo isolation
Metal mounting base
Non-mechanical for reliability

Other equipment needed (float switches, motor starters) that were identified after searching through the Grainger Catalog and other product catalogs, are given in the following pages:
IEC Miniature Contactors:

The following table shows the specifications of the IEC Miniature Contactors.

<table>
<thead>
<tr>
<th>Amp Rating</th>
<th>Horsepower Ratings</th>
<th>Coil Voltage @60Hz</th>
<th>Model</th>
<th>Stock No:</th>
</tr>
</thead>
<tbody>
<tr>
<td>73.0</td>
<td>1-Phase: 5.0 15.0</td>
<td>208V 230V 460V 575V</td>
<td>24</td>
<td>CKH 6P118</td>
</tr>
<tr>
<td></td>
<td>3-Phase: 20.0 25.0 50.0 60.0</td>
<td></td>
<td>120</td>
<td>CKA 6A320</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>208</td>
<td>CKE 6P119</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>240</td>
<td>CKB 6P117</td>
</tr>
</tbody>
</table>

Table 3: Specifications for the IEC miniature contactors.

Dimensions: H : 5.12”        Each : $248.00
            W : 3.54”
            D : 4.76”

Madison Liquid Level Controls:
(Float Switches)
1. Magnetically actuated dry reed switch
2. Used for alarm circuits controlling motor starters, contactors, solenoids, and relays
3. Switches have 22 gauge, 24” leads
4. Use selectable normally open or normally closed switch operation

Specific to No.2XC13:
1. Polypropylene float and stem
2. For general purpose and highly acidic conditions
3. Mounts inside tank or outside tank

Watts Voltage Current (Resistive)
220VAC  0.14A
30  110VAC  0.28
120VDC  0.07
24VDC  0.28

220VAC  0.4
60  110VAC  0.5
120VDC  0.2
24VDC  0.5

<table>
<thead>
<tr>
<th>Float Material</th>
<th>Max. Temp</th>
<th>Max. Pres.</th>
<th>Madison Watts</th>
<th>Stock Model</th>
<th>No:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polypropylene</td>
<td>100</td>
<td>100Psi</td>
<td>30 M8700</td>
<td>2XC13</td>
<td></td>
</tr>
</tbody>
</table>

Dimensions:
Figure 8 shows the dimensions in detail.
No. 2XC13

½"NTP  12.7" max.

Figure 8: Float switch dimensions.
Appendix – F

COSS project members

A list of major persons involved in the COSS (as of 15th July 1999) project is given below:

1. Research and Development Director
   Robin Jamail
   rjamail@glo.state.tx.us

2. Project Director
   Dr. Jim Bonner (Director, CBI)
   bonner@acs.tamu.edu

3. Operations Director
   Dr. Andy Ernest
   Environmental Eng. (Tamu-Kingsville)
   A-Ernest@tamuk.edu

4. COSS Facility Manager
   Robert B. Kitchen
   COSS Facility
   4301 Waldron Road
   Corpus Christi, Texas 78418
   bkitchen@pipeline.tamuk.edu
Appendix – G

Control Program Listing

'Programmer: Ravikanth Bairi
'Date: 06/16/99.

'THIS PROGRAM READS THE THREE FLOAT SWITCHES AND SETS THE PINS
ASSOCIATED WITH THE VACUUM AND DRAIN PUMPS.
'THIS PROGRAM TRIES TO GIVE ERROR MESSAGES WHenever IT IS
POSSIBLE FOR THE CHIP TO DETECT THE ERROR.

'ALLOCATE INPUT AND OUTPUT PINS
input 0 : input 1 : input 2
output 3 : output 4 : output 5 : input 6

    symbol  tprev = b3          'SYMBOLS FOR THE VARIABLES
    symbol  tcur = b4

'SET THE CLOCK

b15 = 99 : b16 = 03 : b17 = 26 : b18 = 6
b19 = 16 : b20 = 31 : b21 = 0
    call clockset

'READ THE INITIAL STATES OF THE SWITCHES

first : bit0 = pin0 : bit1 = pin1 : bit2 = pin2
    tprev = 0             'SET THE tprev(tprevious) FLAG TO ZERO

'IMPLEMEN'T THE INITIAL DECISION AND SET THE tcur(tcurrent) FLAG

bit3 = bit0 : bit4 = bit1 : bit5 = bit2
    gosub Set_Current    'SET CURRENT FLAG

    if bit0 = 0 or bit1 = 0 then setpins1
        goto aftersetpins1
setpins1 : gosub vacuumOn : gosub drainOff
    aftersetpins1 :

    if bit1 = 1 and bit2 = 0 then setpins2
        goto aftersetpins2
setpins2: gosub vacuumOn : gosub drainOn
    aftersetpins2 :

    if bit2 = 1 then setpins3
goto aftersetpins3
setpins3: gosub vacuumOff : gosub drainOn
     aftersetpins3 :

' FINISHED MAKING INITIAL DECISION

' DISPLAY THE INFORMATION

b6 = bit3 : b7 = bit4 : b8 = bit5
     gosub displaystat
     tprev = tcur
     pause 1000

' KEEP READING SWITCHES AND IMPLEMENT DECISIONS REPETETIVELY

second : bit3 = pin0 : bit4 = pin1 : bit5 = pin2

' IF THE NEW STATES ARE DIFFERENT FROM PREVIOUS ONES THEN
' TAKE DECISION, ELSE GO TO LAST STATEMENT TO LOOP BACK

     if bit0 = bit3 and bit1 = bit4 and bit2 = bit5 then skip
     goto afterskip
     skip : goto lastline
     afterskip:

' STATES HAVE CHANGED, SET THE TCUR FLAG
     gosub Set_Current

' MAKE DECISION

     if bit5 = 1 then setvd1
     goto aftersetvd1
     setvd1: gosub vacuumOff : gosub drainOn : goto vd_set
     aftersetvd1:

     if bit4 = 1 and bit5 = 0 then setvd2
     goto aftersetvd2
     setvd2:
     if tprev < tcur then setvd2a
     gosub vacuumOff : gosub drainOn : goto vd_set
     setvd2a: gosub vacuumOn : gosub drainOn : goto vd_set
     aftersetvd2:

     if bit3 = 1 and bit4 = 0 then setvd3
     goto aftersetvd3
     setvd3:
if tprev<tcur then setvd3a
    gosub vacuumOn : gosub drainOn : goto vd_set
setvd3a: gosub vacuumOn : gosub drainOff : goto vd_set
aftersetvd3:

if bit3 = 0 then setvd4
    goto aftersetvd4
setvd4: gosub vacuumOn : gosub drainOff : goto vd_set
aftersetvd4:

vd_set:

' NOW THAT V AND D BITS ARE SET, OVERWRITE THE
' PREVIOUS VALUES WITH THE NEW ONES AND LOOP BACK

bit0 = bit3 : bit1 = bit4 : bit2 = bit5
tprev = tcur

' DISPLAY THE INFORMATION

b6 = bit3 : b7 = bit4 : b8 = bit5
gosub displaystat

lastline: pause 1000
    goto second

'SUBROUTINE TO DISPLAY THE STATUS
displaystat:
    call clockget
    serout 5, N9600, ("Time(h:m:s) - ", #b19, ":", #b20, ":", #b21)
    serout 5, N9600, (" States(low,mid,up) - ", #b6, ":", #b7, ":", #b8)
    serout 5, N9600, (" (v,d) - ", #b10, ":", #b11)
    serout 5, N9600, (10, 13)
    return

'SUBROUTINE TO SET THE FLAG THAT INDICATES WHETHER THE FLUID
'LEVEL IS RISING OR FALLING
Set_Current:
    if bit3 = 0 then settc1
        goto aftersettc1
    settc1: tcur = 0
        aftersettc1:

    if bit3 = 1 and bit4 = 0 then settc2
        goto aftersettc2
settc2: tcur = 1
aftersettc2:

if bit4 = 1 and bit5 = 0 then settc3
  goto aftersettc3
settc3: tcur = 2
aftersettc3:

if bit5 = 1 then settc4
  goto aftersettc4
settc4: tcur = 3
aftersettc4:

'CHECK FOR ERROR CONDITIONS AND SIGNAL AN ERROR MESSAGE IF ANY

if bit3 = 0 and bit4 = 0 and bit5 = 1 then error1
if bit3 = 0 and bit4 = 1 and bit5 = 1 then error1
if bit3 = 1 and bit4 = 0 and bit5 = 1 then error1
  if bit4 = 1 and bit3 = 0 then error1
  goto aftererror
error1: serout 5, N9600, ("Err")
aftererror:

tcur_set:
return

'H TEPER SUBROUTINES TO TURN ON/OFF THE OUTPUT PINS

'SUBROUTINE TO TURN ON THE PIN ASSOCIATED WITH VACUUM PUMP
vacuumOn: pin4 = 1 : b10 = 1
return

'SUBROUTINE TO TURN OFF THE PIN ASSOCIATED WITH VACUUM PUMP
vacuumOff: pin4 = 0 : b10 = 0
return

'SUBROUTINE TO TURN ON THE PIN ASSOCIATED WITH DRAIN PUMP
drainOn: pin3 = 1 : b11 = 1
return

'SUBROUTINE TO TURN OFF THE PIN ASSOCIATED WITH DRAIN PUMP
drainOff: pin3 = 0 : b11 = 0
return
Data Definition:

1. b3, b4: These are the byte variables used to store the tprev and tcurl flags.
2. tprev, tcurl: These are the symbols used to refer to the b3, and b4 register.
3. bit0, bit1, bit2, bit3, bit4, bit5: These are the individual bits of the b0 byte register that can be accessed. These are used to store the float switch states.
4. b15 – b21: These are the predefined registers that the ‘clockset’ routine looks for time information to set the real time clock in the chip.
5. b6 – b8: These are the registers that contain the state information that the ‘displaystat’ subroutine sends to the serial port.
6. b10, b11: These are the registers used to store the states of the vacuum and drain pumps.
7. first: This is the line label for the code that reads the float switch states for the first time when the control program starts.
8. second: This is the line label for the code that reads the float switch states after the initial decision has been taken.
9. lastline: This is the line label for the code which pauses for 1000 milliseconds before looping back to read the float switch states.
10. bit3, bit4, bit5: These are the bits the ‘Set_Current’ subroutine reads to determine the value of the tcurl flag.
11. clockget: This built-in subroutine fills the b15 – b21 registers with the time information.
12. displaystat: This subroutine displays the state information of the three float switches and those of the vacuum and drain pumps (ON/OFF) present in the various registered allocated for them.
13. Set_Current: This subroutine sets the tcurl flag based on the float switch states. It also detects error conditions and sends error signal on the serial port.
14. vacuumOn: Subroutine to set the pin4 on (used for vacuum pump).
15. vacuumOff: Subroutine to set the pin4 off (used for vacuum pump).
16. drainOn: Subroutine to set the pin3 on (used for drain pump).
17. drainOff: Subroutine to set the pin3 off (used for drain pump).